Fully Communication Oriented Information Modeling (FCO-IM)

Guido Bakema
Jan Pieter Zwart
Harm van der Lek

The Dutch software company **Bommeljé Crompvoets en partners** (BCP) brings their FCO-IM tool **CaseTalk** in three editions:

a **Book Edition** (free download, limited storage capacity)

an **Educational Edition** (free download for university students and lecturers all over the world, almost full functionality, 6 months expiration time)

and

a **Professional Edition**

See the website: www.CaseTalk.com

Contents

1

Information Systems, Information Systems Development and Information Modeling

In this introductory chapter we present in brief the context in which the subject of this book, *Fully Communication Oriented Information Modeling (FCO-IM),* should be placed.

First we define what we mean by the term information system. Next we discuss the process of information systems development and show at which stage of this process information modeling is to be placed. Finally we present the basic principles of Fully Communication Oriented Information Modeling.

The reader who is only interested in the present state-of-the-art with respect to the power, the concepts, the schema techniques and the methodology of FCO-IM can safely skip this chapter.

1.1 Information Systems

Information plays a dominant role in society nowadays. The competitive position of companies and the proper functioning of non-profit institutes depend strongly on the ability to provide management, employees, customers and government in time with the information they require. Whereas but a few decades ago all information was processed by hand, today information handling and exchange is automated to a very large extent. This is known as *automation of information services*. Commercial companies took the lead here, multinationals and large businesses first. With the advent of the personal computer small and medium-sized businesses followed. Non-profit institutes initially lagged behind in this trend, but in the meantime hospitals, educational institutes, churches and associations have automated their information systems to a considerable degree.

At first this automation was done by computer departments in large businesses. But it was not long before an entirely new branch of industry consisting of *automation experts* developed, providing goods and services for the purpose of automating the information processing of their customers. This branch consists of computer system suppliers, software houses, system houses, consultancy bureaus and the like. In all industrialized (or better: automated) countries this branch of industry accounts for a considerable percentage of the gross national product. Either standard products are used as ready-to-wear solutions to automation problems, or software and hardware are tailored to the specific needs. In the latter case the communication processes in the organization are charted (or even redefined), the information that is being exchanged by these communication processes is analyzed, and complete *automated information systems* are designed, built and implemented. Another possibility is that support is provided to organizations that perform the development of their desired information systems, or the improvement of their existing information systems, themselves.

To make matters more concrete we give a few examples of information systems:

- flight reservation systems for airlines
- information systems for registering insurance policies and dispatching claims
- information systems for financial transactions (transfers) in banks
- information systems for processing orders in mail-order firms
- booking systems for travel agencies
- stockkeeping systems
- systems for the Registry of Births, Deaths and Marriages of civil administrations
- student administration and monitoring (mark recording) systems for educational institutes
- systems for the membership record of churches and associations

Initially, in the process of automation of information services the emphasis was strongly on the automation (computerization) side of the matter: “We want to automate our administration of orders, invoices and stock.”. By now, computers are taken for granted everywhere; they are even able to communicate with each other without too grave technical difficulties. In addition, relational database management systems (RDBMS’s), application generators and front-end tools such as form and report generators enable automation experts to meet the requirements on the automation side of affairs without too much trouble. So presently, there is an increasing interest in the real *informatization* aspects: “Which communication processes do we actually need for good management and how do we get the relevant information for that purpose?” (*data warehousing*). *Business consultants, organization experts* and *information analysts* try to answer these questions and also questions like “What information do we actually have (often without knowing it) in our information systems, which may be used to derive new valuable information from?” (*data mining*).

This shift in emphasis is also manifest in the definition of an information system that we use in this book:

An information system is a subsystem of an organization with the purpose to support the organization as efficiently as possible in:

- **recording the facts that are relevant to the organization;**
- **maintaining these facts;**
- **deriving other relevant facts from these facts;**
- **retrieving these facts when desired.**

This definition highlights the main focus of this book: the *facts* that are relevant to the organization concerned. In this book it hardly matters therefore whether the information systems are hand-operated ones or whether they have been partly or completely automated.

The demand for automation experts is more or less stable nowadays, but the demand for *information experts* is still increasing. Today the number of highly qualified information experts graduating from institutes of higher education or from commercial trainings is insufficient to fill the need on the job market. These experts must be able to determine systematically the information requirements of the domain experts (administrators, managers, technicians etc.) and lay them down in a *conceptual information model*. Furthermore, if required, they must be able to simply translate this conceptual information model into a *logical data model*: a data model that can be implemented (in a computer system). The word ‘simply’ is used deliberately: we will show in chapter 4 that such a translation from a

conceptual information model to a relational database schema, which can be implemented directly using an RDBMS, can be done completely algorithmically and can therefore be automated. Relational application generation has also been automated to a large extent: the FCO-IM tool that comes with this book can generate a simple but working sample relational application directly from a conceptual information model. Why do we translate to a *relational* data model? Because the bulk of all information systems currently being developed is to be so implemented (cf. section 1.2).

Three remarks conclude this section:

1 We defined an information system as a subsystem of an organization. Such a subsystem also includes the *users* of the information system in a narrow sense (those who record information in, maintain the information in and withdraw information from the information system) and in a wider sense (those who supply or consume that information). Aside: suppliers and consumers of information need not be human beings: they may be machines as well, for example machines in automated production processes (in that case those mechanical users in the wider sense are usually - but not necessarily - also users in the narrow sense). We will use the term 'information system' also in a more restricted sense, namely without including the users. We did so in fact in the second sentence in this remark, where we wrote 'users of the information system'. Obviously we should not be overly consistent on this point. The exact meaning will be clear from the context.

2 A similar point can be made with respect to the popular distinction between *data* and *information*. In all communication processes we are dealing with facts that mean something to the participants in the communication, in a context they collectively share and understand. In addition to this aspect of meaning, facts also have an aspect of form, manifest in what we will call: *representations of facts* (namely as spoken words, or in the form of written sentences, tables, graphs, schema's and the like, recorded on paper, visible on viewing screens or stored in digital form on data carriers). By *information* we will mean this duality of meaning and form. When just the aspect of form is meant (that is when only the representations of the facts are considered) the term *data* is generally used. But when people use the terms *data analysis* and *information analysis* they always mean the same.

3 We can view information systems from three different *perspectives*:
   1 the *information oriented perspective*, which concerns the *information* that is of interest in the communication processes that are to be supported by the information system. The analysis of this relevant information determines to a very large extent the structure and

the integrity aspects of a database design;

2 the *process oriented perspective*, which concerns the *processes* that act on this information (input, output, maintenance and derivation processes). Which processes should the information system offer and what exactly should they do?

3 the *behavior oriented perspective*, which concerns the triggers (such as choosing a menu option) that start a process. Which triggers are important? This perspective is especially important in the case of real-time systems.

Today the insight is predominant that in information systems development (cf. section 1.2.1) the main emphasis is on the information-oriented perspective (data oriented information systems development). In the first place, data structures are generally more stable than processes or triggers: information (meaning types of facts here) taking part in communication processes is usually less prone to changes than the way in which this information is processed. In the second place, the process oriented and behavior oriented perspectives can only be specified precisely in terms of the information that is to be processed. Finally, desired changes in the process oriented and behavior-oriented perspectives can generally be carried out much easier with modern tools than changes in the information-oriented perspective. Therefore, the problem: how to realize changes in the database schema of an operational database without too much trouble, is one of the most important research issues nowadays (this is known as the *delta problem*). All this underlines the importance of meticulous information modeling.

1.2 Information Systems Development and Information Modeling

The examples of automated information systems, presented in section 1.1, have something in common: they are all examples of information systems that have been automated for the very reason that huge amounts of information have to be processed. In such a situation automation is bound to pay off. It was no coincidence that the automation of information processing started with automating administrative processes.

1.2.1 Information Systems Development versus Software Engineering

In this book, we are interested in developing information systems for processing information in *data intensive fields of application*. We mean by this, that the amount of information is vast, not only at the *instance level* (a lot of facts), but especially at the *type level* (many different *sorts* of facts: *fact types*). This is usually the case in administrative information systems and/or management information systems (decision support systems), and especially in data warehouses, data marts etc. As an example, consider a student monitoring system in an educational organization, which should not only have a purely administrative function (recording, maintaining and retrieving school results, such as marks), but which should also supply the school management with all sorts of required information, often statistical in nature (obtained by aggregation of the recorded data), which partly is to be sent on to the government (often aggregated even further). However, *technical information systems* also often concern data intensive fields of application. Consider for example the large amounts of data of various sorts required in running modern, almost completely automated, power stations.

It is especially in such data intensive areas of application that Fully Communication Oriented Information Modeling can be applied fruitfully. Of course, FCO-IM can be used in fields of application that have little data intensity as well, but this powerful method of information modeling has proved to be pre-eminently useful where there is a large number of fact types. Such areas of application are usually (but not always) of little algorithmic complexity. The reverse (algorithmically complex systems often have simple information structures) is also true, but to a lesser extent. As a consequence of this phenomenon, the two different disciplines of *software engineering* and *information systems development* are seldom applied in an integrated way. Software engineering concentrates on the analysis and design of complex algorithms, which may or may not be data intensive at the instance level, but which generally have little data intensity at the type level (do not concern many fact types): see figure 1.1.

These data structures, together with the algorithms that are defined on them, are usually implemented in programming languages of the third generation (3GL), nowadays often with the use of object oriented (OO) techniques. FCO-IM can make significant contributions here as well, especially with respect to the class structure of OO implementations. Conversely, the discipline of information systems development is primarily concerned with developing information systems with complex data structures, which are data intensive at the type level (many fact types), but which are often simple from an algorithmic point of view: see figure 1.1. Such information systems are excellently suitable for a relational implementation and for data manipulation with SQL. This explains why we confine this book, and the FCO-IM tool, which goes with it, to relational implementations. We remark, however, that both disciplines should be deployed in fields of application with both algorithmic complexity and data intensity. In such a case relational interfaces are frequently developed, which feed the 3GL/OO data structures, and vice versa (see figure 1.1).

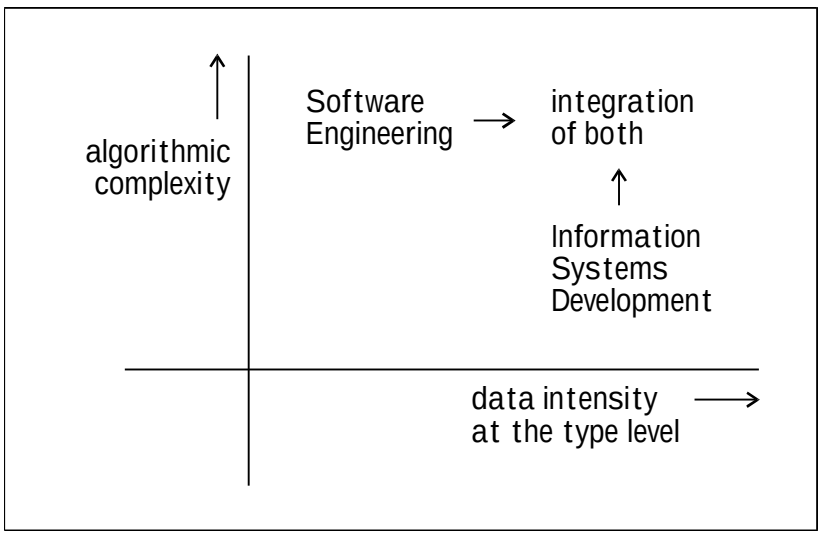


Figure 1.1: information systems development and software engineering

An *information systems development method* (*ISDM*) is a set of coherent techniques and procedures that can be deployed in the development of information systems. ISDM's can be distinguished, among other things, by the relative emphasis they place upon the three different perspectives (see concluding remark number 3 in section 1.1). We will not go into this further, nor into embedding FCO-IM in a complete information systems development method. A more important remark is: more and more, information modeling occupies the central and most important position in information systems development; the heart of the matter is to draw up an optimal conceptual information model.

A final remark:
The algorithmic transformation of an FCO-IM conceptual information model to an OO implementation schema does not appear to be more difficult than the transformation to a relational implementation schema. We are presently exploring this avenue in close cooperation with Ascaris Software, the builders of the FCO-IM tool that comes with this book, and intend to build such an algorithm into the next release of the FCO-IM tool.

1.2.2 Information Modeling as a Phase in the Information Systems Life Cycle

This book is about *information modeling*, or *information analysis*. In this section, we intend to clarify what we mean by these terms, and to place information analysis in the context of the complete information system life cycle: an idealized linear process for designing, building and relationally implementing a new automated information system. Although this is an oversimplification (in practice, this process is less straightforward and partly cyclic in nature), as well as a limitation (information analysis can also take place in a re-engineering project, for instance), it provides a sufficient basis for our line of thought.

The interpretation of the term information analysis varies. Some people use it to designate the activity of roughly charting information flows in an organization. Others understand it as giving an overall specification of the information sets that make up those flows. Still others count both activities in. We would rather situate these activities in the *definition phase*, which should yield what is called a global functional model of the nascent information system. Such a functional model specifies input and output functions of the information system globally in terms of the accompanying input and output information sets.

The main objective in the definition phase, then, is in our view not so much a detailed analysis, but rather a summary overview of information flows and information sets. For this purpose, various diagramming techniques are being used in practice, such as data flow diagrams (DFD's) and ISAC diagrams (A and I diagrams for the present and desired situations). We will not discuss these techniques in this book.

In most accepted *phasings* of the information system life cycle (see figure 1.2 for a commonly used phasing with *milestone products*), this definition phase follows the *planning phase* (or *information planning phase*), in which it is decided whether or not information systems will be developed, and if so which ones, with what priority: financial and organizational feasibility studies are performed, and (if one or more information systems pass) a project planning for

the development is made.

Phases	Milestone products
Planning phase	Feasibility study, Project planning
Definition phase	Global functional model
Analysis phase	Conceptual model
Design phase	Logical and Technical design reports
Building and testing phase	Realized information system
Introduction phase	Manual, Plan of introduction
Maintenance phase	Additional documentation

Figure 1.2: phasing and milestone products

Information analysis takes place in the analysis phase, which follows the definition phase. First, the analysts, together with domain experts, determine the relevant information sets much more precisely than in the definition phase (exactly which kinds of facts are involved?). Next, these information sets are subjected to an analysis process yielding a *conceptual information model*. This latter analysis process, in the restricted sense of drawing up an information model, is also called *information modeling*, and it is the main topic of this book. To the former activity of the analysis phase (the precise determination of the relevant information sets), we will pay no more attention than is necessary to provide a suitable starting point for the information modeling. The reason for this is not that we consider it as less important or easier to carry out. On the contrary, it is in fact a very important and difficult step, which is why it cannot be treated in an introductory section in a cursory manner. It is best learned in a large practical project (a case). However, not until one has acquired sufficient skill in information modeling in smaller cases, (that is what this book is intended for), one has reached the stage to apply this in larger cases drawn from practice. This demonstrates the difficulty of determining information flows and information sets in detail.

Chapters 2 and 3 show how to draw up such a conceptual information model, which we call an *information grammar* in FCO-IM. A relational database schema can subsequently be derived from such an information grammar. This is discussed in chapter 4. This derivation takes place in the *logical design* part of the design phase of the life cycle. It is followed by the

technical design part of the design phase (sometimes also called *implementation design*), in which it is decided how the database schema will be implemented and how input and output functions will be realized. This depends strongly on the choice of the hardware platform and the implementation tools (RDBMS, application generators, forms and reports generators). Such chooses also belong to the technical design. We will pay no further attention to the technical design in this book.

After this follow the *building and testing phase*, the *introduction phase* (the plan of introduction also comprises the schooling of end users), and finally, as soon as the information system is used operationally, the *maintenance phase*, in which it is vital again to document every change or addition well.

1.3 Basic Principles of FCO-IM

Fully Communication Oriented Information Modeling is founded on a number of basic principles, which we will state and discuss in this section.

> FCO-IM must model all conceptual aspects of the communication that the information system should support, and nothing but those aspects.

This *principle of 100 % conceptuality* has been adopted as a standard starting point for information modeling in the international scientific literature (as an example, see nr. 6 in the literature list). It means that implementation elements (for example elements having to do with the technical realization) do not belong to the conceptual information model. Neither does the way the data is represented externally (on screens, in reports etc.). However, *all* conceptual aspects (i.e. aspects necessary for understanding the communication) must be included explicitly in the conceptual information model. FCO-IM completely meets this requirement of 100 % conceptuality (see nrs. 6 and 11 in the literature list), although one subtle modeling construct is not covered in this book: the *set type*, which is mainly of theoretical interest and is perhaps better avoided in practice (so: only 99.99 % conceptuality).

> FCO-IM does not model reality itself, but communication about reality.

It is after all not possible to record actual objects from reality in information systems (nor properties of such objects or relationships between such objects). We can only register alphanumerical representations of relevant facts about objects from reality. Therefore it is better to analyze the representations of those facts (in whatever form they are available) than reality itself. This is not a novel insight either, as it was previously formulated as foundation for *NIAM* (*Natural language Information Analysis Methodology*), a tradition in which FCO-IM has its roots (see appendix B about FCO-IM and NIAM). In NIAM, this insight led to the important methodological principle of *verbalizing* representative examples of facts that are considered to be relevant to the communication processes, thus obtaining declarative sentences in natural language: representations of facts in the form of sentences (see section 2.3). Any such sentence expresses, in alphanumerical form, an exemplary fact that is, or can be, of interest in the communication to be modeled, and as such it must be recordable in the intended information system, perhaps in abbreviated form. In FCO-IM, we not only use these natural language sentences as the starting point to arrive at a conceptual information model,

but we also want to include a complete and redundancy free modeling at the type level of these natural language sentences themselves in the final product of the information modeling process: an *FCO-IM information grammar*.

Figure 1.3 shows schematically the perspectives of domain experts (their concrete reality and verbalizations of representative facts that can be part of the communication about this reality) and information analyst (the same verbalizations and the FCO-IM information grammar he/she is to draw up).

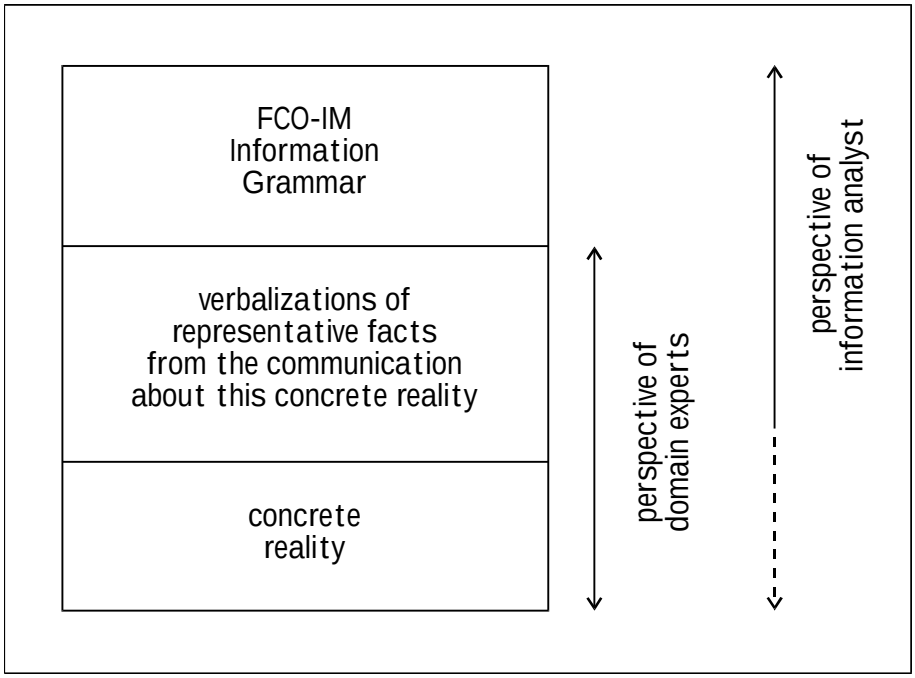


Figure 1.3: perspectives in FCO-IM

The dialogue between information analyst and domain experts (interviews of the latter by the analyst to arrive at an FCO-IM information grammar) takes place at the middle level: that of exemplary verbalizations. The analyst also gains some insight into the way domain experts view their reality during these interviews.

Domain experts must be able to validate the correctness of the modeling of their communication in an FCO-IM information grammar.

During and after the modeling process, analysts must be able to regenerate sentence representations of exemplary facts used in the process, using a simple substitution algorithm,

and present them to the domain experts. Only if they approve these sentences, i.e. if they confirm that they did (or could) express their relevant information in this way, then the analyst did a good job, otherwise he/she did not. This substitution process is implemented in the FCO-IM tool in such a way, that it can be activated at any time during the modeling process so the domain experts can *validate* the (intermediate) results.

It must be possible to represent both FCO-IM information grammars
and relational schemas using the same FCO-IM diagramming technique.

This *genericity* of FCO-IM implies that an FCO-IM information grammar diagram can be transformed step by step into a redundancy free relational schema, which can subsequently be implemented in an RDBMS. This transformation process can be automated as well and can be followed completely graphically using the FCO-IM tool. In fact, this transformation is carried out in the repository (itself a relational database) of the FCO-IM tool, based on stepwise changes in the population of this repository (it is comparable with the system catalog of an RDBMS). The graphical representation is nothing more or less than an external manifestation of the contents of this generic repository, running synchronously along with its changes.

These four basic principles together enable us to derive a redundancy free relational schema with corresponding integrity rules (*not null* constraints, primary keys, foreign keys, and the like: the *hard semantics* of the relational database) from an FCO-IM information grammar in a fully transparent way. Moreover, we can completely take verbalizations of the recorded data in natural language (the *soft semantics* of the database) along in this process. Especially this last point is uniquely characteristic of FCO-IM and the FCO-IM tool, in which this transformation to a relational schema with corresponding soft semantics is automated. We even deem this worthy of a formulation as principle:

FCO-IM must be able to model the soft semantics as well, and
preserve them completely as a supplement to relational database schemas.

We mentioned earlier that FCO-IM is in the NIAM tradition. This is not only true of its modeling constructs and diagramming technique, but also of its operational procedure, which we can summarize in four basic principles:

Domain experts are required to supply verbalizations, in natural language, of representative examples of facts that are of interest in their communication.

Domain experts must be interviewed in their own language and in their own jargon.

Design decisions must be based on concrete examples.

Analysis must be carried out methodically, i.e. stepwise and according to instructions.

In the next chapters, the operational procedure according to these basic principles will be presented and illustrated comprehensively. The accompanying instructions will be systematically elaborated, until the complete FCO-IM procedure has been treated. This is of course (see the above) always done starting from concrete examples: small cases.

2

Modeling the Communication

In this chapter, we discuss the basic concepts and the working method in the first steps of Fully Communication Oriented Information Modeling: verbalization, classification, qualification and constructing a provisional information grammar diagram (IGD). We also show how to regenerate the communication from the IGD.

To illustrate the procedure concretely, we use a small case study, which concerns information about students undertaking projects as a part of their information systems development studies at an institution of higher education.

2.1 Starting Document

Our starting point is a short description of the student-project case (see figure 2.1). Such a *starting document* describes in general terms the relevant information and information processes. This starting document was drawn up by the School's Project Coordinator.

The School's Project Coordinator and the students are *domain experts*: people familiar with the part of reality that is relevant to the case (also called the *Universe of Discourse*, abbreviated *UoD*). In general, an information analyst will not be a domain expert, but will often be employed from outside to formulate the specifications for the information system that is to be developed. In this case, the School's Project Coordinator is the most important domain expert for the information analyst, because he drew up the starting document, which identifies the UoD.

Our students are required to participate in an industrial project in the first term of their fourth year. During such a project, the students carry out a task in the field of information systems development in a business or in a non-profit organization. Students can choose their first, second and third preference (all different) from a list of project tasks. This list shows each project offered, with the project supervisor (a teacher coordinating the project) and a short project description. The students base their preferences on these data. They can enter their choices on another list, on which I have already filled in their name and their mentor (each student has a teacher as a personal student adviser). If they do so before the date on which I assign them to their project, then I try to meet their preferences as best I can. A list of project assignments is subsequently posted on the notice board. Students who have not given their preferences in time will not be assigned to a project at this stage; I allocate remaining projects to them later. In any one year, there are about 200 students, so I would welcome any form of computerized support for the administration of this task.

Figure 2.1: starting document

In information analysis, we are only concerned with the *information* that is considered to be important by the domain experts. The *processes* operating on this information are left aside. In other words: in information analysis, we only consider the *information perspective* and not the *process perspective*.

The starting document mentions three lists. It is a reasonable assumption that these lists together embody the information perspective of the student-project case. The information analyst underlines the parts of the starting document that refer to these lists:

Our students are required to participate in an industrial project in the first term of their fourth year. During such a project, the students carry out a task in the field of information systems development in a business or in a non-profit organization. Students can choose their first, second and third preference (all different) from a list of project tasks. This list shows each project offered, with the project supervisor (a teacher coordinating the project) and a short project description. The students base their preferences on these data. They can enter their choices on another list, on which I have already filled in their name and their mentor (each student has a teacher as a personal student adviser). If they do so before the date on which I assign them to their project, then I try to meet their preferences as best I can. A list of project assignments is subsequently posted on the notice board. Students who have not given their preferences in time will not be assigned to a project at this stage; I allocate remaining projects to them later. In any one year, there are about 200 students, so I would welcome any form of computerized support for the administration of this task.

Figure 2.2: starting document with information perspective underlined

2.2 Concrete Examples

In order to clarify the information perspective, the analyst asks the School's Project Coordinator for concrete examples of each list mentioned in the starting document. Figures 2.3, 2.4 and 2.5 show these *concrete example documents*. It is not necessary that these documents contain every student or project, but enough data should be entered to exemplify clearly all the information concerned.

The example document in figure 2.3 shows data on projects offered.

Project code	Supervisor	Project Description
P101	BLC	Developing a timekeeping system
P102	BAK	Exploring various CASE-tools
P110	LEK	Introducing an RDBMS into a business
P115	ENG	Building an automated project assignment system
P120	FEL	Converting a dBASE system to Foxpro
P200	BAK	Developing a technical information system
P201	BLC	Analyzing complex information systems
P203	BAK	Writing course material on FCO-IM
P204	FEL	Implementing a design for a database

Figure 2.3: available projects

In figures 2.4a, 2.4b and 2.4c, the list of preferences is depicted at the following three points in time, which are mentioned in the starting document:

1 before any student has filled in his or her preferences
2 after a few students have filled in their preferences
3 at the moment the allocation has taken place of projects to those students who have made their preferences known in time.

The situation before any student has made his preferences known:

First Name Student	Surname Student	Mentor	First Project Preference	Second Project Preference	Third Project Preference
Peter	Johnson	BLC	---	---	---
John	Hartman	BLC	---	---	---
Elsa	Doyle	BAK	---	---	---
Maria	Jones	VRM	---	---	---
Fred	Smith	GPB	---	---	---
Peter	Groves	JPC	---	---	---
Lizzy	Johnson	BAK	---	---	---
Frank	Seymour	VRM	---	---	---
Tom	Dakota	HVL	---	---	---

Figure 2.4a: project preferences 1

The situation when a few students have made their choice:

First Name Student	Surname Student	Mentor	First Project Preference	Second Project Preference	Third Project Preference
Peter	Johnson	BLC	P101	P203	P110
John	Hartman	BLC	P203	P101	P200
Elsa	Doyle	BAK	---	---	---
Maria	Jones	VRM	---	---	---
Fred	Smith	GPB	---	---	---
Peter	Groves	JPC	---	---	---
Lizzy	Johnson	BAK	---	---	---
Frank	Seymour	VRM	P102	P201	P101
Tom	Dakota	HVL	P201	P101	P110

Figure 2.4b: project preferences 2

The situation just before the initial project allocation takes place:

First Name Student	Surname Student	Mentor	First Project Preference	Second Project Preference	Third Project Preference
Peter	Johnson	BLC	P101	P203	P110
John	Hartman	BLC	P203	P101	P200
Elsa	Doyle	BAK	P204	P203	P200
Maria	Jones	VRM	P110	P115	P201
Fred	Smith	GPB	---	---	---
Peter	Groves	JPC	P101	P200	P120
Lizzy	Johnson	BAK	---	---	---
Frank	Seymour	VRM	P102	P201	P101
Tom	Dakota	HVL	P201	P101	P110

Figure 2.4c: project preferences 3

The latter case has the most information, so we choose to proceed with that list.

Figure 2.5 presents a concrete example of the list of allocated projects.

First Name Student	Surname Student	Assigned to Project
Peter	Johnson	P101
John	Hartman	P203
Elsa	Doyle	P204
Maria	Jones	P110
Fred	Smith	---
Peter	Groves	P200
Lizzy	Johnson	---
Frank	Seymour	P102
Tom	Dakota	P201

Figure 2.5: allocated projects

2.3 Verbalization

The example documents in section 2.2 illustrate the information perspective in a concrete way. Each line in these documents can be regarded as a compact representation of *facts*, which are of importance to the School's Project Coordinator and the students, and which are being exchanged between them. These facts make up the relevant *communication* between the domain experts. The goal of Fully Communication Oriented Information Modeling is to draw up an exact model of this communication (these facts).

Facts in example documents are usually displayed in a highly compact form, which is convenient and intelligible for domain experts, but which does not explicitly bring to light the full meaning of these facts. Therefore, the information analyst asks the School's Project Coordinator to *verbalize* the facts in *natural language*, the form of communication with which everybody is familiar.

The Project Coordinator will now express the concrete facts in a number of sentences, such as "The school is offering project P101.". Such a sentence expressing a concrete fact is called a *fact expression*. In this book, we will use this formal term 'fact expression' by preference, but occasionally we will use the word 'sentence' as well.

Preferably, verbalization is done in *elementary fact expressions*. An elementary fact expression is a sentence, which puts just one complete separate fact (also called an *elementary fact*) into words. For instance, a fact expression like "Project P101 is supervised by BLC and

concerns developing a time keeping system." is not elementary, because it can be split up without loss of information in the two separate expressions "Project P101 is supervised by BLC." and "Project P101 concerns developing a time keeping system.". The latter two expressions are indeed elementary, because they cannot be split up any further without loss of information.

Just to be absolutely clear, we present the three example documents once more in figure 2.6.

The students and their project preferences:

First Name Student	Surname Student	Mentor	First Project Preference	Second Project Preference	Third Project Preference
Peter	Johnson	BLC	P101	P203	P110
John	Hartman	BLC	P203	P101	P200
Elsa	Doyle	BAK	P204	P203	P200
Maria	Jones	VRM	P110	P115	P201
Fred	Smith	GPB	---	---	---
Peter	Groves	JPC	P101	P200	P120
Lizzy	Johnson	BAK	---	---	---
Frank	Seymour	VRM	P102	P201	P101
Tom	Dakota	HVL	P201	P101	P110

Projects offered:

Project code	Supervisor	Project Description
P101	BLC	Developing a timekeeping system
P102	BAK	Exploring various CASE-tools
P110	LEK	Introducing an RDBMS into a business
P115	ENG	Building an automated project assignment system
P120	FEL	Converting a dBASE system to Foxpro
P200	BAK	Developing a technical information system
P201	BLC	Analyzing complex information systems
P203	BAK	Writing course material on FCO-IM
P204	FEL	Implementing a design for a database

Allocations:

First Name Student	Surname Student	Assigned to Project
Peter	Johnson	P101
John	Hartman	P203
Elsa	Doyle	P204
Maria	Jones	P110
Fred	Smith	---
Peter	Groves	P200
Lizzy	Johnson	---
Frank	Seymour	P102
Tom	Dakota	P201

Figure 2.6: the three example documents

Complete verbalization of all the information in figure 2.6 by the School's Project Coordinator yields the following elementary fact expressions, each representing an elementary fact. For convenience, we have grouped all facts of the same sort of fact (all facts of the same *fact type*) together.

1) "There is a student Peter Johnson."
2) " " " " John Hartman
3) " " " " Elsa Doyle
4) " " " " Maria Jones
5) " " " " Fred Smith
6) " " " " Peter Groves
7) " " " " Lizzy Johnson
8) " " " " Frank Seymour
9) " " " " Tom Dakota

10) "The mentor of student Peter Johnson is BLC."
11) " " " " John Hartman " BLC
12) " " " " Elsa Doyle " BAK
13) " " " " Maria Jones " VRM
14) " " " " Fred Smith " GPB
15) " " " " Peter Groves " JPC
16) " " " " Lizzy Johnson " BAK
17) " " " " Frank Seymour " VRM
18) " " " " Tom Dakota " HVL

19) "The school is offering project P101."
20) " " " " " P102
21) " " " " " P110
22) " " " " " P115
23) " " " " " P120
24) " " " " " P200
25) " " " " " P201
26) " " " " " P203
27) " " " " " P204

28) "Project P101 is supervised by BLC."
29) " P102 " " " BAK
30) " P110 " " " LEK
31) " P115 " " " ENG
32) " P120 " " " FEL
33) " P200 " " " BAK
34) " P201 " " " BLC
35) " P203 " " " BAK
36) " P204 " " " FEL

37) “Project P101 concerns developing a timekeeping system.”
38) " P102 " exploring various CASE-tools
39) " P110 " introducing an RDBMS into a business
40) " P115 " building an automated project assignment system
41) " P120 " converting a dBASE system to Foxpro
42) " P200 " developing a technical information system
43) " P201 " analyzing complex information systems
44) " P203 " writing course material on FCO-IM
45) " P204 " implementing a design for a database

46) “The first preference of student Peter Johnson is project P101.”
47) " second " " " Peter Johnson " " P203
48) " third " " " Peter Johnson " " P110
49) " first " " " John Hartman " " P203
50) " second " " " John Hartman " " P101
51) " third " " " John Hartman " " P200
52) " first " " " Elsa Doyle " " P204
53) " second " " " Elsa Doyle " " P203
54) " third " " " Elsa Doyle " " P200
55) " first " " " Maria Jones " " P110
56) " second " " " Maria Jones " " P115
57) " third " " " Maria Jones " " P201
58) " first " " " Peter Groves " " P101
59) " second " " " Peter Groves " " P200
60) " third " " " Peter Groves " " P120
61) " first " " " Frank Seymour " " P102
62) " second " " " Frank Seymour " " P201
63) " third " " " Frank Seymour " " P101
64) " first " " " Tom Dakota " " P201
65) " second " " " Tom Dakota " " P101
66) " third " " " Tom Dakota " " P110

67) “Student Peter Johnson was allocated project P101.”
68) " John Hartman " " " P203
69) " Elsa Doyle " " " P204
70) " Maria Jones " " " P110
71) " Peter Groves " " " P200
72) " Frank Seymour " " " P102
73) " Tom Dakota " " " P201

We have put all the facts from figure 2.6 into words to stress the point that all information can indeed be verbalized. In general, however, it suffices to verbalize only a few specimen facts of each fact type, noting that we must not omit any fact types. Composite (non-elementary) fact expressions, if any, will be detected later in the method and reduced to their elementary form (see section 3.3.1). So, the more elementary the initial sentences are, the more effort is saved later on.

Verbalization should be done by a domain expert, because only these experts are able to phrase the facts correctly. In practice, however, an information analyst is often quite capable of expressing a large part of the facts in sentences too, especially when such purely administrative documents are involved. In general, it is even desirable that the information analyst is to some extent familiar with the UoD. In that case, however, it is absolutely essential that these tentative verbalizations by the analyst are validated by domain experts to ensure the correctness of the analyst's interpretation of the data in the example documents.

2.4 Classification and Qualification

The next step in the analysis procedure is the *classification* and *qualification* of the fact expressions. To classify means: arranging things into classes (groups). To qualify means here: giving a meaningful name to each class. We divide the classification and qualification step into two stages. We will begin with the first stage, and introduce the second stage via an intermediate step, after having discussed the identifiability of objects in the UoD and the principle of redundancy free modeling. The working method in the second stage will then be illustrated using the example student-project case study, and subsequently formalized in a summary procedure. We will conclude this section with a discussion of the concept of nominalization and with a few final remarks.

2.4.1 First Stage of Classification and Qualification

In the first stage of classification and qualification, all fact expressions are grouped into classes (classified). In section 2.3, we have already done so for all 73 elementary fact expressions. The seven classes are: fact expressions 1 through 9, 10 through 18, 19 through 27, 28 through 36, 37 through 45, 46 through 66 and finally 67 through 73. All fact expressions in one particular class are verbalizations of elementary facts of the same type (or kind) of fact, which is why we call such a class an *elementary fact type.* Next, we must give each elementary fact type a meaningful name: this is called *qualification*. For the seven classes in our example, we choose respectively: 'Student', 'Mentorship', 'Project', 'Supervision', 'Project Description', 'Preferences' and 'Allocation'. This completes the first stage of classification and qualification.

2.4.2 Intermediate Step

In the second stage of classification and qualification, the structure of the fact expressions for each fact type is analyzed. For didactical reasons, we will start the illustration of this procedure with an intermediary step, which is not included in the formal procedure presented later on.

All fact expressions of one particular fact type have sentence parts in common, but they also have places containing something different in each fact expression. In section 2.3, the common parts are designated by ditto marks ("). We now write down the common parts of each fact type and leave the remaining places blank, indicating these with three dots. This yields preliminary *elementary fact type expressions* (or *elementary sentence types*), which show for each fact type how all its facts can be verbalized:

Elementary fact type	Preliminary elementary fact type expression
Student	F1: "There is a student … … ."
Mentorship	F2: "The mentor of student … … is … ."
Project	F3: "The school is offering project … ."
Supervision	F4: "Project … is supervised by … ."
Project Description	F5: "Project … concerns … ."
Preferences	F6: "The … preference of student … … is project … ."
Allocation	F7: "Student … … was allocated project … ."

Please note: two blanks are indicated where student names are to be filled in: one for their first name and one for their surname. This makes it clear, that we now have to indicate what *sort of values* are to be entered in these blanks. For instance, we can regenerate complete fact expressions from preliminary fact type expression F1 by filling in first names and surnames of students. The values to be entered are called *labels*, and the sort of values they belong to are called *label types*. For each preliminary fact type expression, we now indicate which label types are to be used in the blanks. This yields *fact type expressions at the label type level,* abbreviated as *LTL-FTEs*:

F1: "there is a student <first name> <surname>."
F2: "the mentor of student <first name> <surname> is <teacher code>."
F3: "the school is offering project <project code>."
F4: "project <project code> is supervised by <teacher code>."
F5: "project <project code> concerns <description>."
F6: "the <ordinal number> preference of student <first name> <surname> is project <project code>."
F7: "student <first name> <surname> was allocated project <project code>."

In this book, we prefer not to use capital letters at the beginning of formal fact type expressions. We will explain the reason for this in section 2.4.7.

The decision as to which sort of labels is to be entered in the blanks is another classification: a class of labels is assigned to each blank. Giving a meaningful name to these label types is another qualification. Above, we recognized the following label types: 'first name', 'surname', 'teacher code', 'project code', 'description' and 'ordinal number'. The LTL-FTEs F1 through F7 are formulated in terms of invariable text and blanks referring to label types.

Having carried out this intermediate step, we are ready to continue the analysis of the fact expressions, but first we will consider the necessity of identifiability and modeling in a redundancy free way.

2.4.3 Identifiability and Redundancy Free Modeling

In communication about objects (things) in the UoD, it is of vital importance that no misunderstandings can arise about which objects are being referenced. Therefore, each object in the UoD must have a unique identifier; in other words: all objects must be *identifiable*. An identifier often consists of a single name, number, or code, but frequently combinations are used as well, such as first name + surname. (In The Netherlands, an address even has two different compound identifiers: postal code + house number, and street name + house number + town name.) In the student-project case study, it appears that students are identified by the combination first name + surname, projects by a project code and teachers by a teacher code. Analysts should always verify the identifiability with the domain experts. In our case, the project coordinator confirms these identifiers and adds in a comment that the teacher code is a three-letter mnemonic for his or her surname. Apparently, there can be no two students having the same first name and surname in this UoD! In section 2.9 we will make the necessary changes to accommodate the more realistic situation where students can have the same names.

Information models should avoid *redundancy* as much as possible. Redundancy (superfluity) arises for instance if the same fact is modeled more than once, but also if the same part of the communication is modeled more than once. As a case in point, all objects of the same sort are usually identified in the same way (such as: all students are identified by first name + surname). We want to record such a way of identification only once somewhere, even if it is used in many different sorts of fact expressions.

The purpose of the second stage of classification and qualification is to clarify these identification structures and to record them in a redundancy free way.

2.4.4 Second Stage of Classification and Qualification

In the first place, please note that sentences 1 through 9 of fact type expression F1 only declare that certain students *exist* in the world of the domain expert (the UoD) described in the starting document. For instance, fact expression 1: "There is a student Peter Johnson.", *postulates* (states) *the existence* of an object present in the UoD, namely the student with first name 'Peter' and surname 'Johnson'. We call such a fact expression an *existence postulating*

fact expression for a given object. Sentences 1 through 9 postulate the existence of nine different objects (students), which all belong to the same *object type* (sort of object), namely: Student. At the type level we can say: fact type expression F1 is an *existence postulating fact type expression* for object type Student. Just to be clear, we will use a capital first letter in names of object types.

In the second place, please note that there is a sentence part 'student Peter Johnson' in fact expressions 10, 46, 47, 48 and 67, which refers to the same student whose existence was postulated in fact expression 1, namely the student called Peter Johnson. This reference 'student Peter Johnson' *identifies* an object (in this case: a student) in the UoD: everybody knows exactly which student is indicated. Such a sentence part that identifies an object is called an *object expression*. We define an object expression to be *the largest connected (unbroken) part of a fact expression with the exclusive purpose of identifying an object in the UoD*. So it is not just the part 'Peter Johnson' that is an object expression in fact expressions 10 (of fact type expression F2), 46, 47, 48 (of F6) and 67 (of F7), but rather the part 'student Peter Johnson'. Formulated at the level of fact *type* expressions: there is a common part in F2, F6 and F7, viz. 'student <first name> <surname>', that serves (after filling in) as an identifier for (objects of) object type Student. Such a largest connected part of a fact type expression, which serves exclusively as an identifier for an object type, is called an *object type expression*. Sentence type part 'student <first name> <surname>' is an object type expression for object type Student. As a consequence of the requirement to model in a redundancy free way, we must set this object type expression apart from fact type expressions F2, F6 and F7.

We give the object type expression a name:
O1: 'student <first name> <surname>'

We can now specify the three fact type expressions F2, F6 and F7 more concisely:
F2: "the mentor of <Student:O1> is <teacher code>."
F6: "the <ordinal number> preference of <Student:O1> is project <project code>."
F7: "<Student:O1> was allocated project <project code>."

The notation <Student:O1> therefore means: an object type expression for object type Student must be entered in this blank, namely O1.

In the third place, please note that the sentence type part 'student <first name> <surname>' also occurs in the existence postulating fact type expression F1. Here, however, we do *not* replace F1 with: "there is a <Student:O1>.", because the following rule applies in FCO-IM:

In an existence postulating fact type for a certain object type, there may not occur an object type expression for the same object type.

A full explanation of the reasons for this rule would overburden the discussion at this point. We therefore only indicate briefly that violation of this rule would here either result in an infinitely recursive fact type structure (see section 6.2.5), or in superfluous unary fact types (see section 5.2).

The information analyst, together with the School's Project Coordinator, now looks for other object type expressions. They observe that teacher codes, such as 'BAK', are identifiers for teachers. For this reason, the analyst decides to model an object type Teacher, with its identifying object type expression O2: '<teacher code>'. This object type expression contains no fixed text, because '<teacher code>' is the largest connected part of F2 and F4, which serves exclusively as identifier for object type Teacher (the preposition 'by' in F4 has nothing to do with the identification). However, there is no existence postulating fact type expression for object type Teacher. The Project Coordinator has not stated any sentences like "There is a teacher BAK." or "Teacher BAK exists." in his original verbalizations. Such rather obvious existence postulating fact expressions are often absent in initial verbalizations. This is not a problem. We will indicate in section 3.4 the circumstances in which existence postulating fact expressions are required and those where they are optional. The analyst here only uses the criterion whether each label of a certain label type identifies an object of a certain object type in the UoD (or: whether each *combination* of labels of one or more label types identifies an object of a certain object type in the UoD, as is the case with Student).

In a similar way, the analyst recognizes an object type 'Project', with object type expression O3: 'project <project code>', because each object of this object type is identified by its project code. Sentence parts like 'project P101' are used in fact expressions of F3, F4, F5, F6 and F7. The last four of these obviously are not existence postulating fact type expressions, but for F3 this is not immediately clear. Therefore, the analyst asks the Project Coordinator whether a sentence such as "The school is offering project P101." implies that there are many projects, some of which are being offered (in which case he would treat 'project P101' as an object expression), or whether this sentence could be replaced with a sentence like "There is a project P101." without loss of information (in which case it is an existence postulating fact expression and he would treat 'P101' only as a label in accordance with the rule given above). The Project Coordinator replies that the latter is the case: as soon as any project is available, it is offered as a choice to the students by putting it on the project list; i.e. there are no other projects. The analyst concludes that F3 is an existence postulating fact type expression.

No object type is modeled in fact type expression F5 for the blank where descriptions are filled in, because the Project Coordinator feels that such a description does not identify any object in the UoD (but if he would have said that a description identifies an object of object type Project Sketch, then the analyst would have entered this object type with the corresponding object type expression into the model). A description is not an alternative identifier for a project either, because the Project Coordinator, prompted by the analyst, declares that occasionally different projects have the same description. In F6, no object type is modeled for the blank '<ordinal number>' because such an ordinal number does not identify an object in the UoD (see section 2.7 for a different treatment of sentences 46 through 66).

We have now come to fact type expressions consisting of invariable text and blanks that refer to object types with corresponding object type expressions, or to label types where this cannot be done. We call these *fact type expressions at the object type level*, abbreviated as *OTL-FTEs*:

O1: ‘student <first name> <surname>’
O2: ‘<teacher code>’
O3: ‘project <project code>’
F1: “there is a student <first name> <surname>.”
F2: “the mentor of <Student:O1> is <Teacher:O2>.”
F3: “the school is offering project <project code>.”
F4: “<Project:O3> is supervised by <Teacher:O2>.”
F5: “<Project:O3> concerns <description>.”
F6: “the <ordinal number> preference of <Student:O1> is <Project:O3>.”
F7: “<Student:O1> was allocated <Project:O3>.”

In this book, we prefer to use lower case letters at the beginning of OTL-FTEs, just as we did with LTL-FTEs in section 2.4.2, see section 2.4.7.

We can regain the LTL-FTEs from the OTL-FTEs by substituting (i.e. replacing, filling in) the object type expressions themselves for the blanks referring to object types with object expression codes.

2.4.5 Operational Procedure in the Second Stage of Classification and Qualification

The considerations in sections 2.4.2 through 2.4.4 lead to the operational procedure illustrated in figure 2.7 below for the example student-project case study.

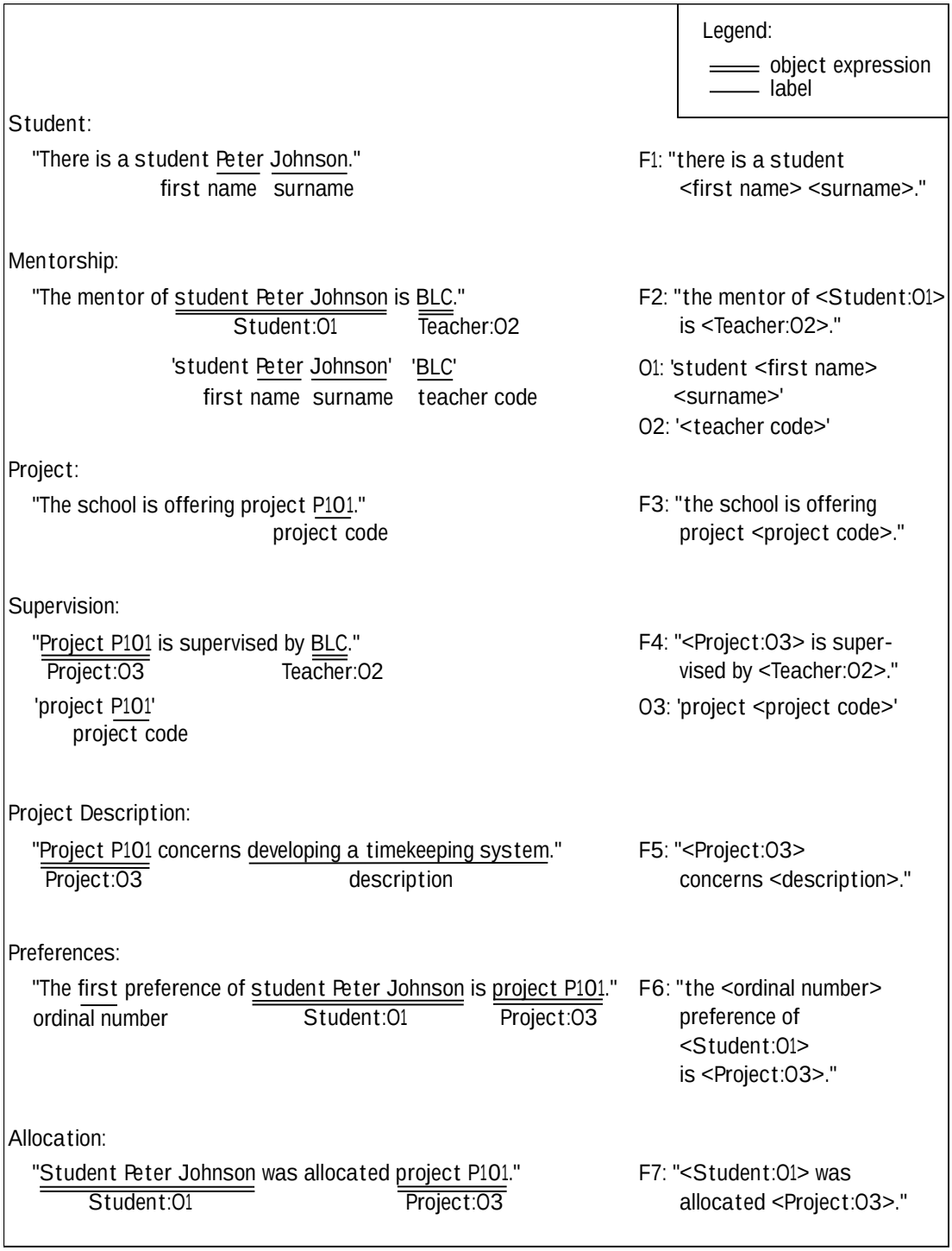


Figure 2.7: illustration of operational procedure for classification and qualification

This operational procedure is as follows:

1 Take one concrete fact expression for each fact type expression. The invariant text is all the parts that are the same in all conceivable fact expressions belonging to this fact type expression; the remainder consists of one or more blanks, with labels filled in.

2 If a label (or a combination of labels) identifies a meaningful object in the opinion of the domain experts, then the analyst and the expert look for the largest connected (unbroken) sentence part with the exclusive purpose of identifying this object. This sentence part is then *classified* as an *object expression* by double underscoring. Next it is *qualified* by giving a meaningful name to the corresponding object type.

Hint: in any fact expression, first look for the *largest* combination of labels that identifies an object. Sometimes a reformulation needs to be made by the domain experts, as in the following example (from a different UoD than our example student-project case study):

Sentence 1: "In project P66, EUR 10,000 is the budget of subproject 3."
Sentence 2: "In project P52, EUR 80,590 is the budget of subproject 3."

In these fact expressions, the sentence parts 'subproject 3' do not identify a subproject, because if nothing more than 'subproject 3' is said, then it is not clear whether subproject 3 of project P66 or subproject 3 of project P52 is meant. Therefore, the project is also required in the identification of a subproject. Here is a reformulation, in which the complete identifiers of the subprojects occur connected:

Sentence 1': "Subproject 3 of project P66 has a budget of EUR 10,000."
Sentence 2': "Subproject 3 of project P52 has a budget of EUR 80,590."

In sentence 1', the connected part 'subproject 3 of project P66' is now classified as an object expression and qualified as Subproject (see figure 2.8).

3 Exception to item 2: In an existence postulating fact type expression for a certain object type, there may not occur an object type expression for the same object type. All labels are usually *classified* separately as *label* in such a case.

4 Remaining labels are each *classified* as *label* by single underscoring. They are subsequently *qualified* by adding a meaningful name for the corresponding label type.

5 All object expressions (OEs) must be analyzed further. The corresponding object *type* expressions (OTEs) must be given a name (in this book: a code consisting of a capital O followed by a number) and must next be treated just like a complete fact expression (FE). Other OEs may occur within a given OE. The analysis of a FE and the OEs it contains is continued until finally no new OEs are found and the label type level is reached throughout.

As an example, consider the analysis of sentence 1' above, which belongs to a fact type called Budget of Subproject and which contains an OE: 'subproject 3 of project P66'. We will call the corresponding object *type* expression (OTE) O1 and continue the analysis. The part '3' does not identify any meaningful object in the UoD, so it is classified as a label, with a

corresponding label type ‘serial number’. The part ‘project P66’ in O1 identifies a project, so it is classified as an OE. It is qualified by adding the name of object type Project. We will call the corresponding new OTE O2. We now have:

O1: ‘subproject <serial number> of <Project:O2>’.

On further analysis of O2, we find no new OE in ‘project P66’, only the label type ‘project code’. The analysis ends with:

O2: ‘project <project code>’.

Figure 2.8 summarizes this analysis and completes it for the remainder of the fact expression.

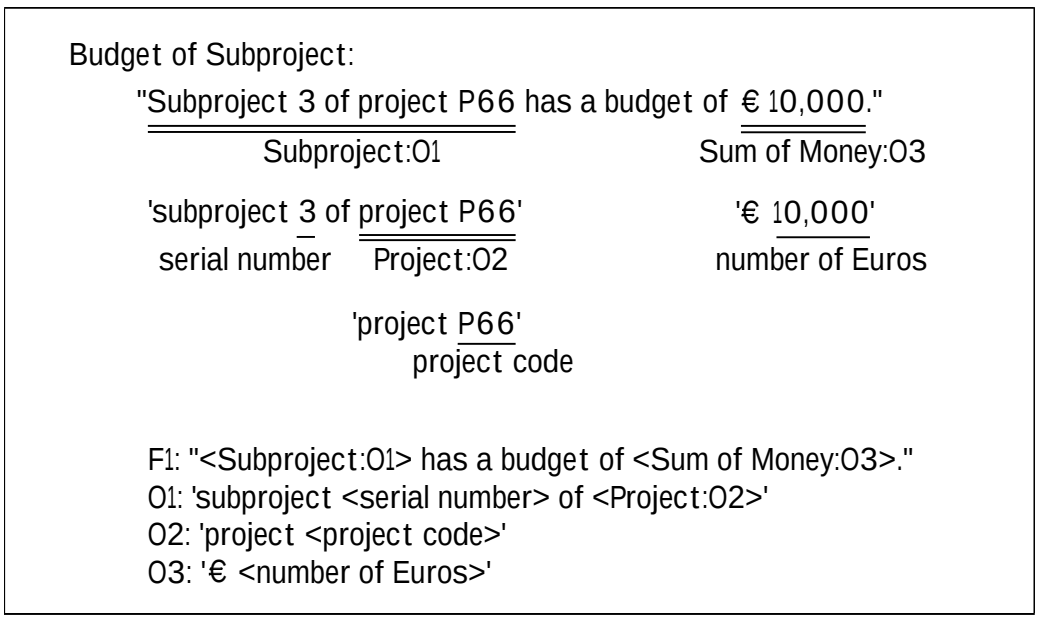


Figure 2.8: analysis of sentence 1'

2.4.6 Nominalization

In the first stage of classification and qualification in section 2.4.1, we used the name ‘Student’ for the *fact* type Student, with corresponding fact type expression F1: “there is a student <first name> <surname>.”. In the second stage of classification and qualification in section 2.4.4, we used the *same* name ‘Student’ for the *object* type Student, with corresponding object type expression O1: ‘student <first name> <surname>’. The reason for this is as follows.

In Fully Communication Oriented Information Modeling (FCO-IM), object expression ‘student Peter Johnson’, which identifies an object in the UoD, is regarded as the result of a *transformation* of the existence postulating fact expression “There is a student Peter Johnson.”. Formulated at the type level: object type expression O1: ‘student <first name> <surname>’ is the result of a transformation of fact type expression F1: “There is a student <first name> <surname>.”. Such a transformation from a sentence (type) to an object (type) expression is called a *nominalization*. The nominalization concept is borrowed from

linguistics, where it means converting a verb into a noun (or nominal group), but in FCO-IM it is used in a broader sense. In FCO-IM, *all* object type expressions are the result of a nominalization of an existence postulating fact type expression.

How such a nominalization takes place is of no concern to us; only the outcome is important for FCO-IM information analysts. Quite often, object expressions are indeed found in the classification and qualification step, but without a verbalization of a corresponding existence postulating fact, as is the case with Teacher. Such an existence postulator may always be added if desired, however, and sometimes this is even necessary (see section 3.4). Because fact type expressions always belong to a fact type, we also use the term *nominalized fact type* if a nominalization of its corresponding fact type expression is used somewhere. This term is meaningful even if we do not know the sentence type that was nominalized, as is the case with Teacher.

Seven fact types were recognized in the student-project case study: Student, Mentorship, Project, Supervision, Project Description, Preferences and Allocation, each with a fact type expression associated with it. The two fact types Student and Project were nominalized to object types with the same name. A third object type Teacher was recognized, which is also a nominalization of a fact type Teacher, although an existence postulating fact type expression is missing.

2.4.7 Final Remarks

1 Concerning the notation of fact type expressions and object type expressions: an object type expression (OTE) cannot begin with a capital letter because it may end up in the middle of an LTL-FTE after substitution in an OTL-FTE (for example O3: 'project <project code>' in F7). However, the same OTE may appear in the front of an LTL-FTE after substitution in another OTE-FTE and should then have its first letter capitalized (the same O3 in F4). A similar problem occurs if a label must be entered in a blank in an LTL-FTE. Only if the blank is in the front should a lower case first letter of the label be changed to upper case. The simplest procedure then is not to use capital letters in LTL-FTEs and OTEs, but to capitalize the first letter only after concrete fact expressions have been reached. For simplicity, we use the same rule with OTE-FTEs.

2 It is sometimes difficult during analysis to decide whether a sentence part should be classified as a label or as an object expression. The rule given before - classify it as an object expression if it identifies a meaningful object in the UoD in the opinion of the domain expert - is a rule of thumb, which follows from the principle to model without redundancy and which works well in practice, but which may lead to fruitless discussions in some cases. When that happens, it is best to take a pragmatic decision, letting the above-mentioned principle outweigh other considerations.

3 Hint: always model a (physical) quantity having a unit of measurement as an object expression in which the unit is treated as fixed text in the object type expression. For example: “Item 34567 weighs 12 pounds.” and “Item 34567 is 34 cm long.”. Classify ‘12 pounds’ and ‘34 cm’ as object expressions and qualify them respectively as Weight and Length. This yields:

O1: ‘item <item number>
O2: ‘<number of pounds> pounds’
O3: ‘<number of cm> cm’
F1: “<Item:O1> weighs <Weight:O2>.”
F2: “<Item:O1> is <Length:O3> long.”

Different label types ‘number of pounds’ and ‘number of cm’ instead of a single label type ‘number’ are used in O2 and O3, because two fundamentally different kinds of numbers are concerned here; in terms of the Relational Model: there are two different domains. It is impossible to compare pounds to centimeters because they are quantities of different dimensions.

4 In a fact expression such as: “Employee E1 has 4 children.”, there are two ways to model this amount. The first is to classify ‘4’ as a label with corresponding label type ‘number’. The second is to classify ‘4’ as an object expression with corresponding object type Amount, because an amount is considered to be a meaningful object in the UoD. The label type identifying Amount could be ‘whole number’. Here it is best to leave ‘children’ in the fact type expression, since an amount does not have a unit of measurement, it is dimensionless.

2.5 Information Grammar Diagram (IGD)

In the previous sections, we represented the elementary facts of the student-project case study in the form of elementary fact expressions. We arrived at seven elementary fact type expressions in the classification and qualification step of information analysis by FCO-IM. Figure 2.7 summarizes this process. In practice however, there often are several hundreds of fact types. The connection between the fact types is easily lost with such large amounts. Therefore, information analysts like to use diagrams, called *information grammar diagrams* (IGDs), that show fact types, label types, object types, fact type expressions and object type expressions in their mutual relationships. Figure 2.9 lists the main graphic symbols used in IGDs.

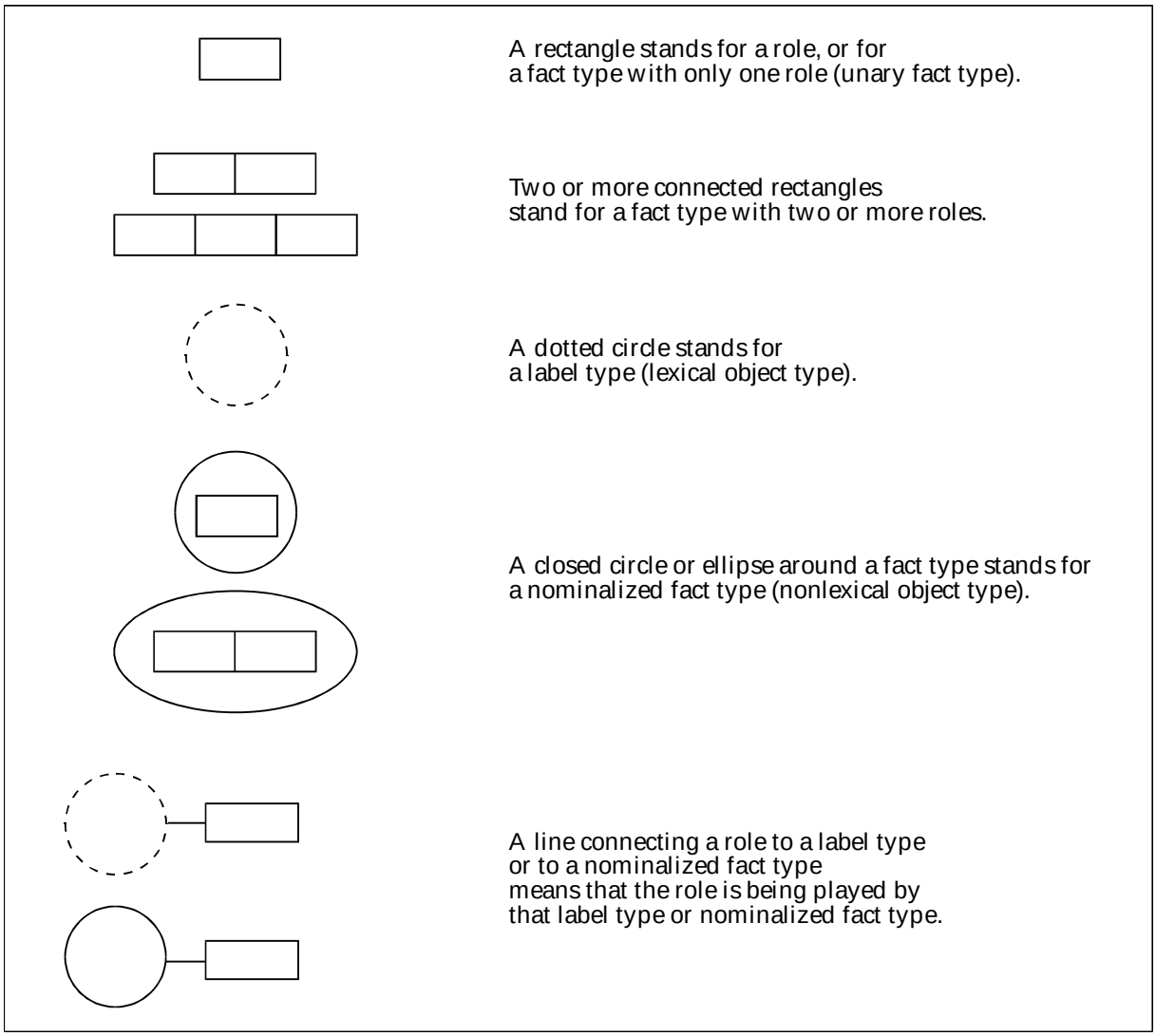


Figure 2.9: graphic symbols

Explanatory remarks on figure 2.9:

1 a fact type is depicted by a number of horizontally connected rectangles, which are called the *roles* of the fact type. There is one role for each blank in a corresponding fact type expression (or in a corresponding object type expression). The ordering in which the roles appear is of no concern. A *unary fact type* has only one role, a *binary fact type* has two roles and a *ternary fact type* has three roles. A fact type with n roles, in which 'n' stands for any number, is called an *n-ary fact type*;
2 a label type is drawn as a dotted circle;
3 a nominalized fact type is drawn as a closed circle or ellipse around the fact type of which the object type is a nominalization;
4 each role is connected by a line to the label type or the nominalized fact type that *plays* the role. That is to say: the role is connected to the object type that supplies an object type expression to be entered into the blank corresponding to the role, or the role is connected to the label type that supplies labels to be entered into the blank corresponding to the role.

2.5.1 Building up an IGD

In an IGD, we draw all the fact types (nominalized or not) and label types that we find in the classification and qualification step, complete with their names. We also put in all fact type expressions and object type expressions complete with their codes. Furthermore, we will add a *population* to each fact type in the IGD: per fact type, we will write labels from one or more sample fact expressions underneath the corresponding roles. We show in figures 2.10 through 2.16 step by step how we build up an IGD for the elementary fact expressions we analyzed in figure 2.7, followed by a brief explanation. We conclude with some well-formedness rules for IGDs.

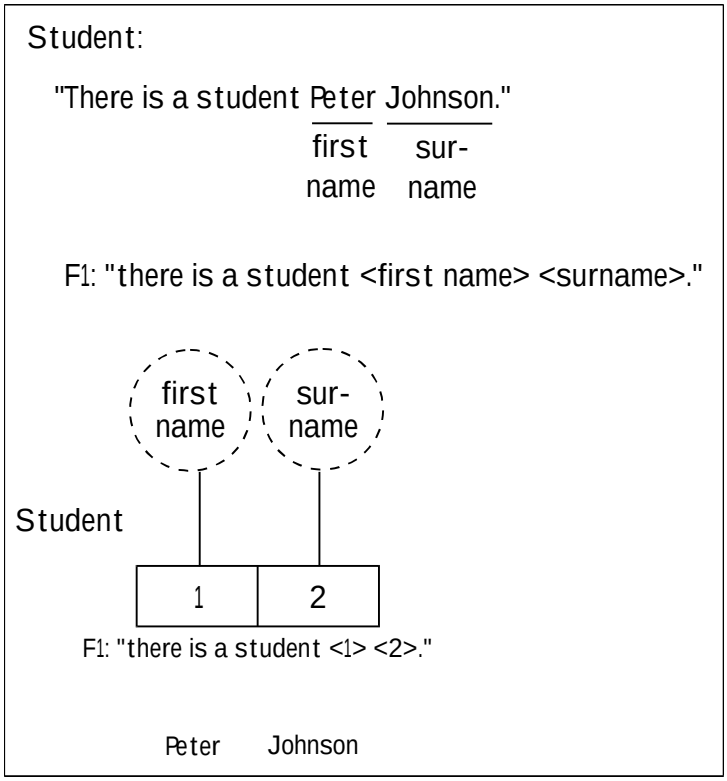


Figure 2.10: building up an IGD, fact type Student

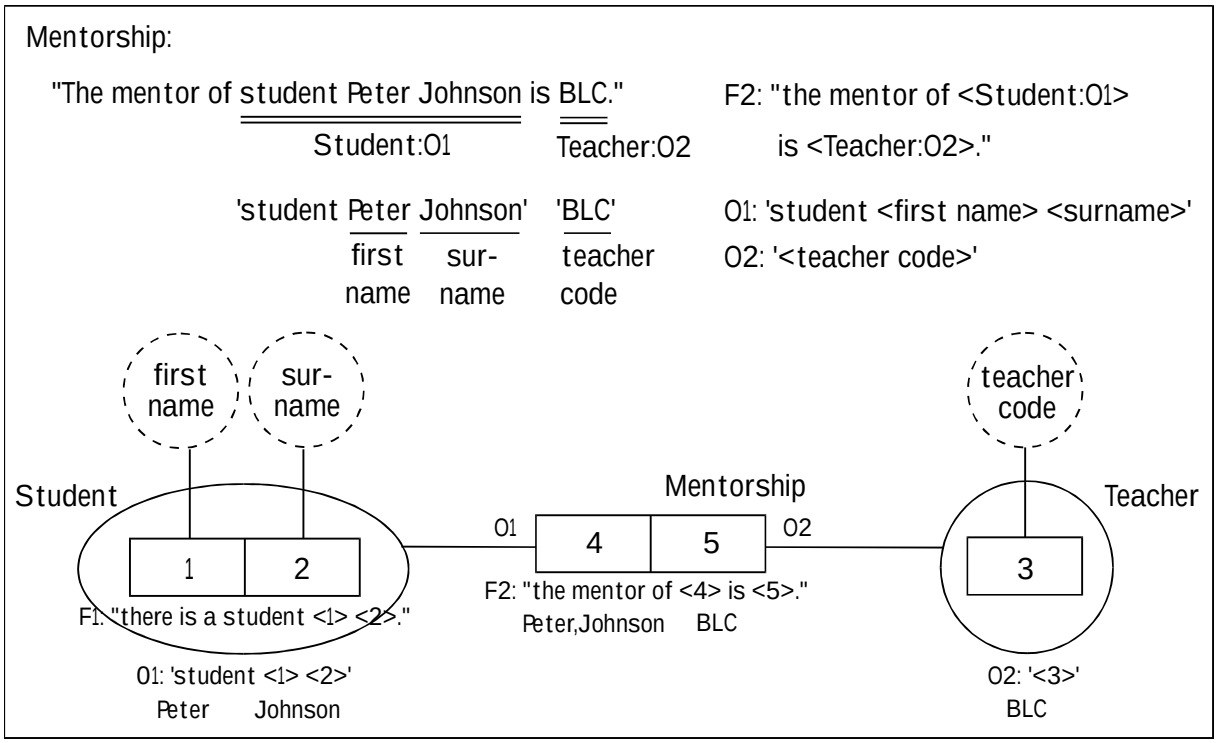


Figure 2.11: building up an IGD, fact type Mentorship

In figure 2.11, object type Teacher arises only because of object type expression O2, without first modeling a fact type of the same name, based on an existence postulating fact type expression, as in the case of Student in figure 2.10 (see also fact type Project in figure 2.12).

In figure 2.11, role 4 of fact type Mentorship is played by object type Student, which is a nominalization of the binary fact type Student (having roles 1 and 2). Consequently, two labels must be written underneath the single role 4 (here: ‘Peter’ and ‘Johnson’), separated by a comma. There is only one label underneath role 5, because this role is played by object type Teacher, which, even though there is no existence postulating fact type expression, is a nominalization of the unary fact type Teacher (having role 3).

There is only one label, or combination of labels, written underneath each role in figures 2.10 through 2.16, because we took just one fact expression per fact type to supply these labels. So the population per fact type consists of just one *tuple,* i.e. a row of labels. We show the final IGD again in figure 2.17, now populating each fact type with several tuples, each tuple from one of the sample fact expressions.

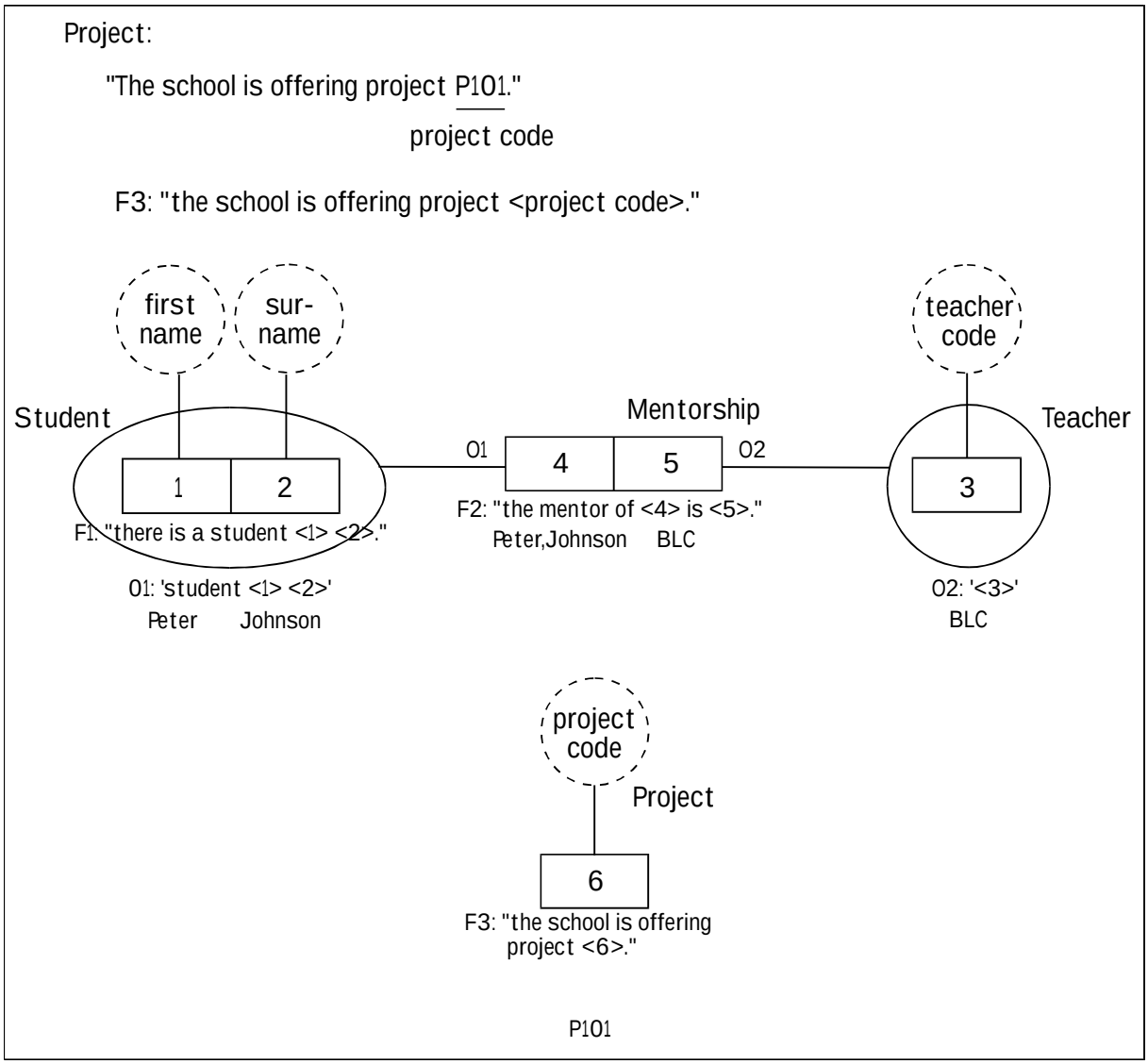


Figure 2.12: building up an IGD, fact type Project

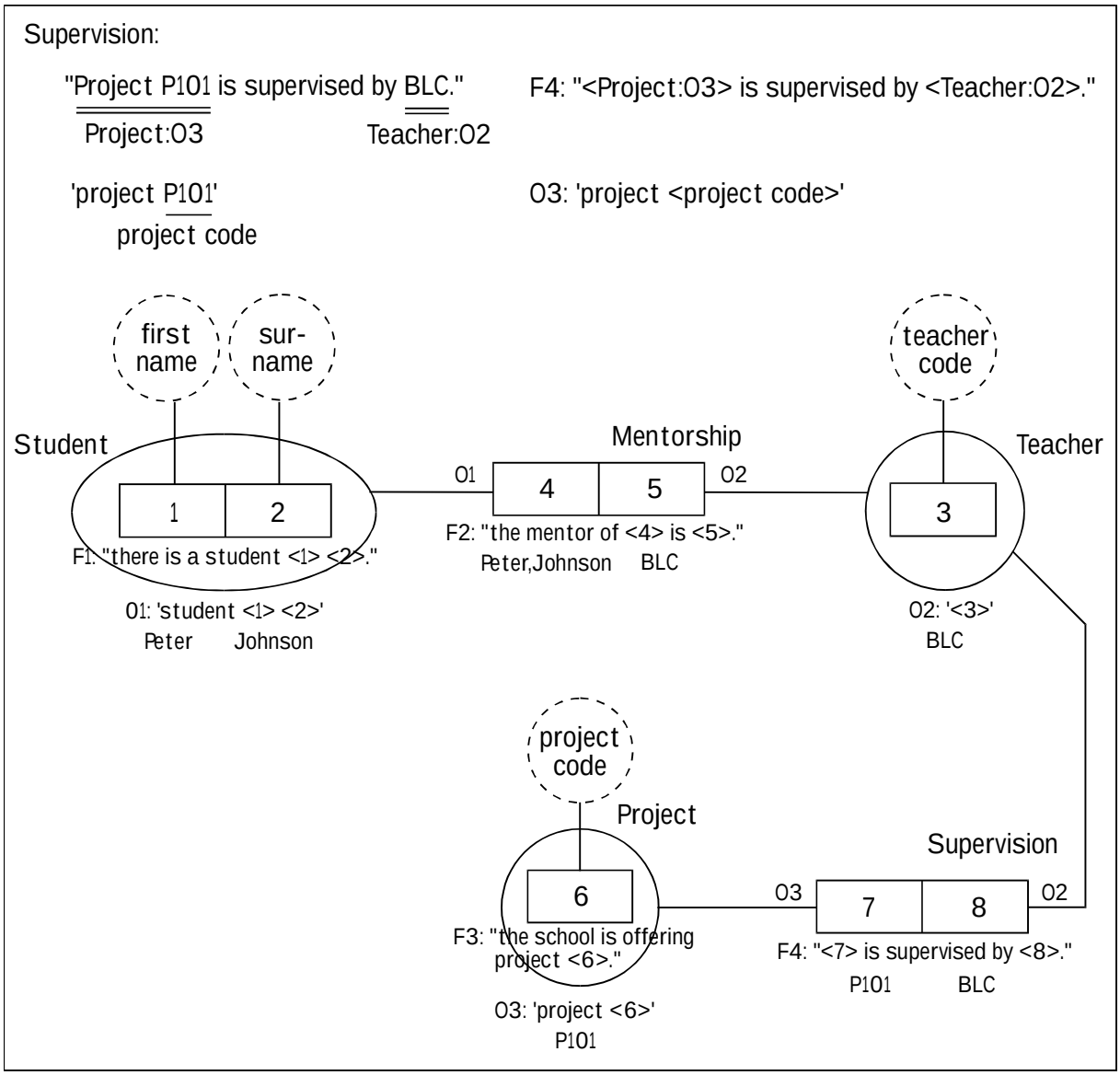


Figure 2.13: building up an IGD, fact type Supervision

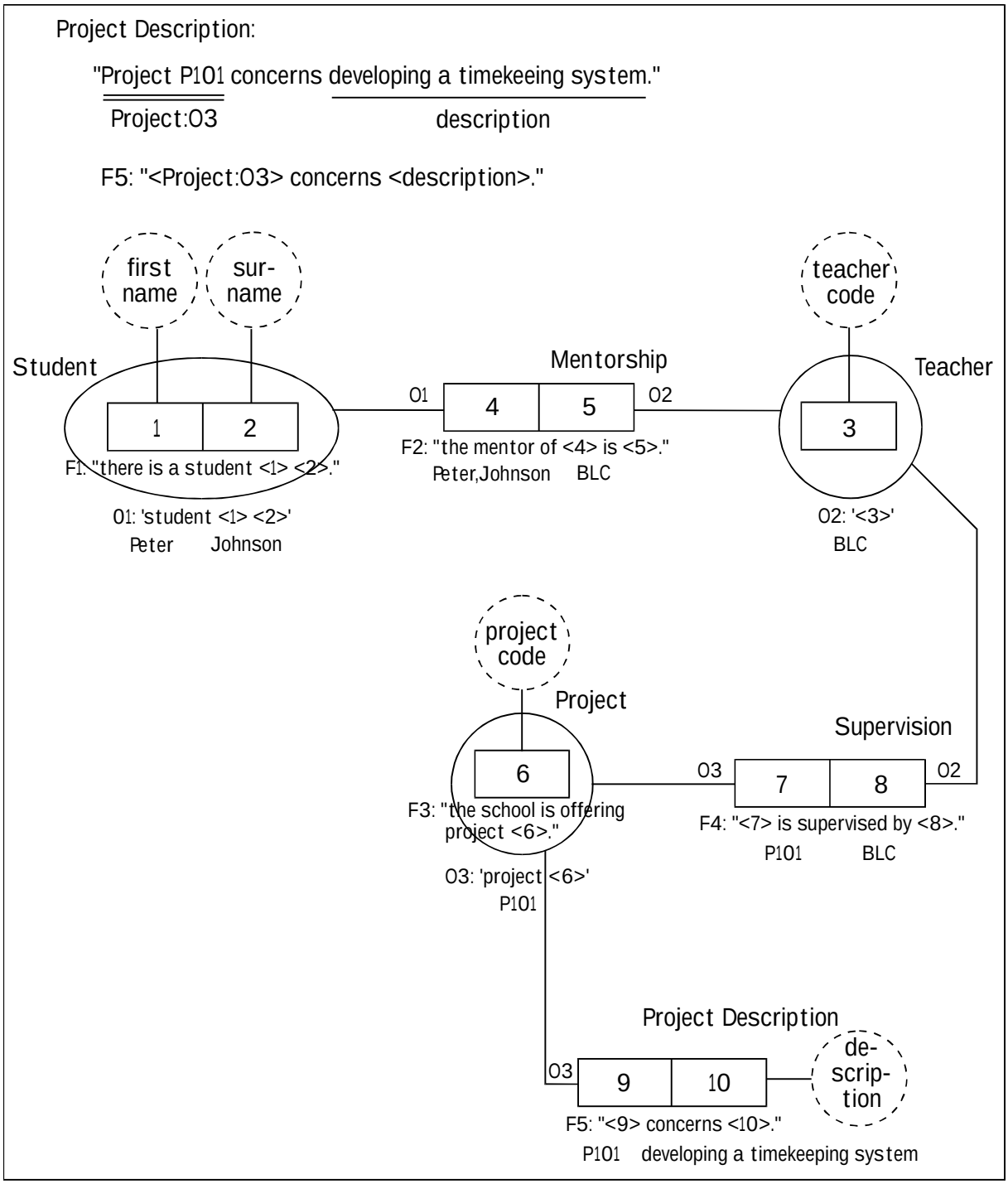


Figure 2.14: building up an IGD, fact type Project Description

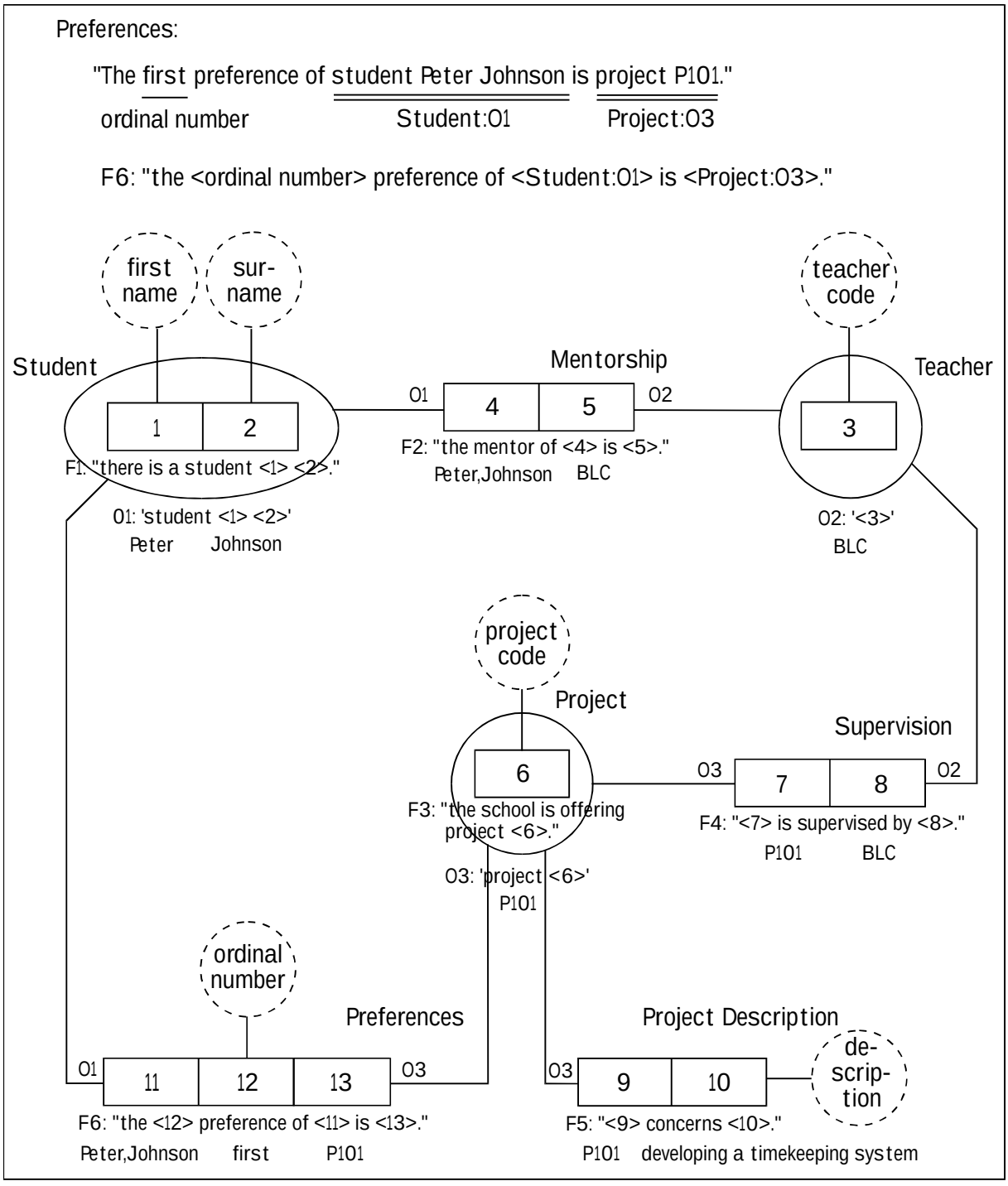


Figure 2.15: building up an IGD, fact type Preferences

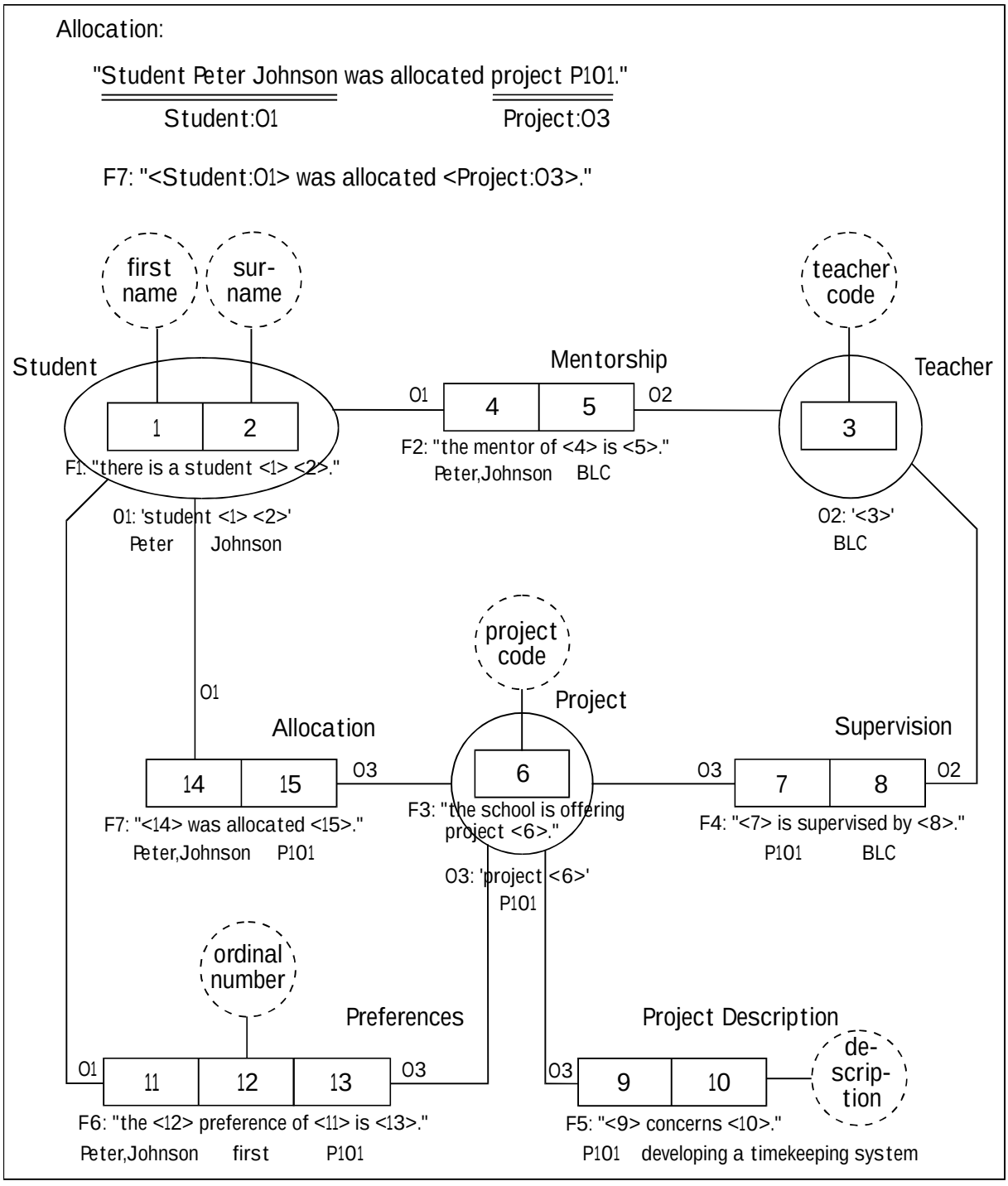


Figure 2.16: building up an IGD, fact type Allocation

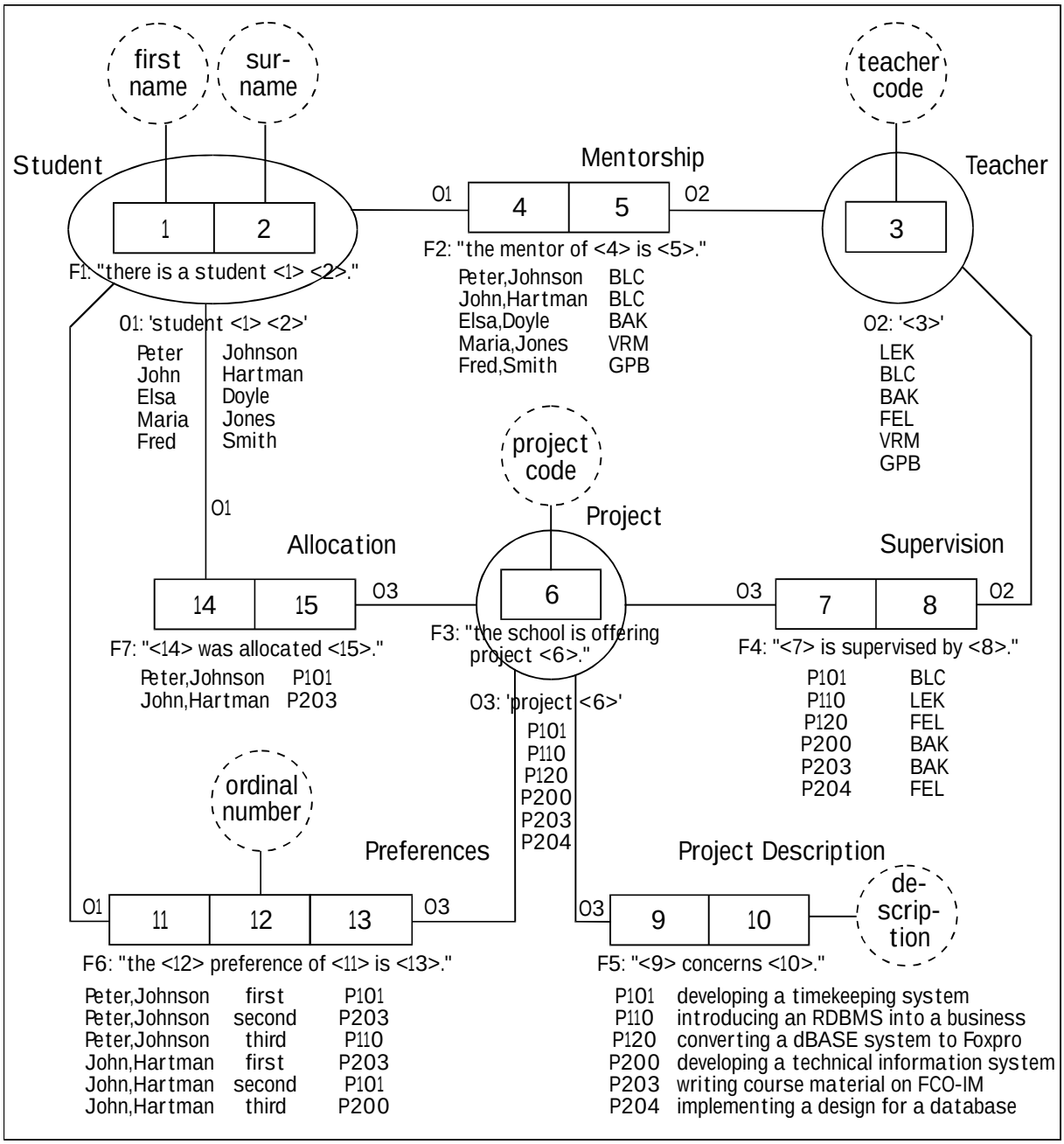


Figure 2.17: IGD with several tuples

The following procedure works well in practice: take a fact type, analyze a corresponding fact expression until no new object expressions can be found and draw the accompanying IGD. This way of working is supported by the FCO-IM tool. Then take the next fact type and proceed in the same way. In this manner, the IGD grows incrementally and the overall picture remains clear. Figures 2.10 through 2.16 illustrate this procedure.

2.5.2 Well-formedness Rules for IGDs

1 Each label type or fact type, nominalized or not, must have its name written next to it; label type names are usually written inside the circle.

2 Each role must have a unique role number. (Other naming conventions are also possible, such as enumerating the roles per fact type, but in this book and in the FCO-IM tool this is the choice we made.)

3 Each role must be part of exactly one fact type.

4 Each role must be played by exactly one label type or nominalized fact type.

5 Each label type and each nominalized fact type must play at least one role.

6 Underneath each fact type that is not nominalized, at least one fact type expression must be written, in which the role numbers (see rule 2) recur. These role numbers replace the label type names and the object type names with object type expression codes that are in the OTE-FTEs. The code of any object type expression involved is written next to the line joining the role with the corresponding object type, so no information will be lost. Fact type expressions with blanks referring to roles are called *IGD-FTEs*. Fact type expressions belonging to nominalized fact types are written inside the ellipses as far as is possible.

7 An existence postulating fact type expression may be given to a nominalized fact type, but this is not always required. We return to this matter in section 3.4.

8 Next to each nominalized fact type, at least one object type expression must be written, in which the role numbers (see rule 2) recur. Within this book, object type expressions are written outside the ellipses.

9 Each fact type expression is given a unique code F1, F2 and so on, and each object type expression gets a unique code O1, O2 etc. (Other conventions are possible, but we chose this one.)

10 Next to each line connecting a role to a nominalized fact type, at least one object type expression code that applies to that role must be written.

11 Each fact type must be populated with at least one tuple.

2.5.3 Final Remarks

1 On the meanings of the words ‘object type’. In the NIAM tradition, in which FCO-IM stands, the words ‘object type’ have different meanings depending on context. This is the result of an historical growth process that is still continuing. However, we chose to introduce no new, more consistent terminology in this text. In the first place, it would not be the final nomenclature either, and in the second place, we want to remain in keeping with the tradition. Consequently, a certain ambiguity in terminology cannot be avoided.

In FCO-IM, it is the communication about the UoD that is being modeled, rather than the UoD itself. An object type in the UoD itself is not the same thing as an object type in an IGD, which has to do with communication (blanks in fact type expressions). We will indicate in what follows whether the UoD itself or the communication about the UoD is meant. The term ‘object type’ carries four different meanings in FCO-IM: one where the UoD itself is concerned (see point a below), and three where an IGD and the communication modeled therein is concerned (see points b, c and d below). The correct meaning, however, is always clear from the context.

a If the context is the UoD itself, then ‘object type’ designates a class of objects (things) that exist in the UoD according to domain experts (students, projects, teachers and so on). Each object type in the UoD is modeled as a (nominalized) fact type in FCO-IM. Sometimes, there is only an existence postulating fact type, which is not used in other fact type expressions in a nominalized form. In such a case, the object type is modeled as a non-nominalized fact type.

b A label type is also called a *lexical object type* (LOT), since labels are lexical (printable, pronounceable) things. This name stems from the time before nothing but the communication about the UoD was being modeled, when labels were regarded as lexical objects also belonging the UoD. *In FCO-IM, ‘label type’ or ‘lexical object type’ means a source of a sort of labels that can be entered in blanks in FTEs and OTEs.* For clarity, we will almost always use the term ‘label type’ in this book.

c What we have consistently called an object type in FCO-IM up to now (namely, a nominalized fact type), is also called a *nonlexical object type* (NOLOT). This name stems from the time before nothing but the communication about the UoD was being modeled, when an analyst modeled non-printable kinds of objects in the UoD (students and teachers of flesh and blood, projects etc.) themselves. In verbal (i.e. lexical) communication about objects in the UoD, a nonlexical object can only be *indicated* (identified) by an object expression in which finally only lexical objects (labels) occur. *In FCO-IM, ‘nominalized fact type’ or ‘nonlexical object type’ means a source of a sort of object expressions that can be entered in blanks in FTEs and OTEs.* Because there is a large measure of correspondence between object types in the UoD and nominalized fact types in an IGD, we often use the term ‘object type’ in this text for short where ‘nonlexical object type’ is meant. The correspondence mentioned is not perfect, however. The given rule: “classify as an object expression that which identifies an object in the UoD” (section 2.4.5, operational procedure point 2), is only a rule of thumb, a useful heuristic (see also section 2.4.7, remark 2).

d Both label types (lexical object types) and nominalized fact types (nonlexical object types) play roles. *In FCO-IM, the term 'object type' means the union of these two role-playing concepts.* The term is being used in this sense in discussing semantically equivalent transformations (sections 2.7 and 5.1), for example.

2 The FCO-IM tool that goes with this book interactively supports the analysis of fact expressions in the way discussed in the text and depicted in figure 2.7. It automatically generates sequential role numbers, FTE codes and OTE codes, based on the order in which the user identifies blanks in fact expressions. Users can automatically generate an IGD (or any part thereof) with the integrated Diagram Designer and change its layout graphically. The FCO-IM tool can also easily enter a population consisting of several tuples.

2.6 Regenerating Fact Expressions from an IGD

In this section, we show how we regain the original fact expressions from an information grammar diagram (IGD).

In an IGD, we find IGD-FTEs (see well-formedness rule 6 in section 2.5.2), with blanks containing role numbers. We will have to convert these into fact type expressions at the label type level (LTL-FTEs, see section 2.4.2) again. We simply treat the blanks containing role numbers as follows.

1 For each blank referring to a role played by a label type (*lexical role*): replace the role number with the name of the label type playing the role.
2 For each blank referring to a role played by a nominalized fact type (*nonlexical role*): replace the role number with the object type expression indicated by the code written next to the line joining the role to the object type. If this object type expression contains blanks that refer to yet more roles, then apply rule 1 and/or rule 2 to these as well.

The end result is an LTL-FTE.

To regain the original fact expressions, we finally write the labels from each tuple underneath the corresponding blank in the LTL-FTE. The initial fact expressions are then easily readable.

Figure 2.18 illustrates this substitution process in the case of fact type expression F6, which belongs to fact type Preferences of figure 2.17. F6 reads: "the <12> preference of <11> is <13>.". Role 12 is a lexical role, so rule 1 applies and we replace '<12>' with '<ordinal number>'. We now move on to role 11, which is played by object type Student, so it is a nonlexical role and rule 2 applies. The code O1 is written next to the line joining role 11 to Student. O1 reads: 'student <1> <2>', and we now substitute this text for the role number. The result so far reads: "the <ordinal number> of student <1> <2> is <13>.". The text of O3 is substituted analogously for role number 13, which yields the intermediate result: "the <ordinal number> preference of student <1> <2> is project <6>.". Roles 1, 2 and 6 are all played by

label types, so we can apply rule 1 in each case. The end result is then: “the <ordinal number> preference of student <first name> <surname> is project <project number>.”. We finally add the labels from all the tuples underneath the correct blanks.

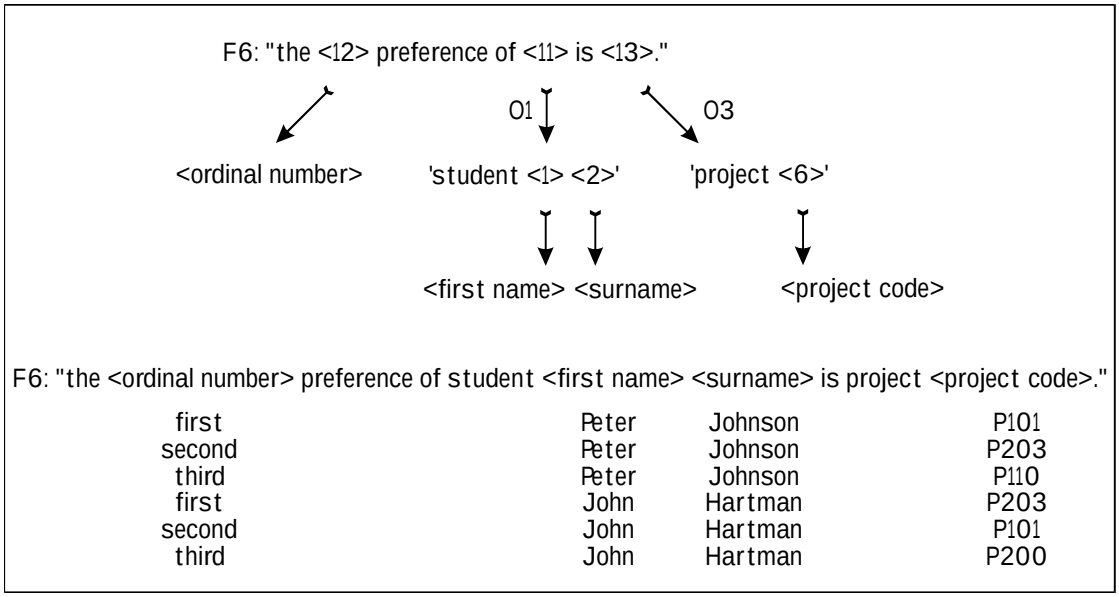


Figure 2.18: illustration of the substitution process

We can regenerate the original sentences for all tuples in the IGD of figure 2.17 in the same way. The result is shown in figure 2.19. This regeneration of sentences is important for the *validation* of the IGD, i.e. the process to check and approve the IGD. If a domain expert, who generally cannot read IGDs and does not want to learn this either, confirms that figure 2.19 does indeed contain the right information, then the information analyst knows that the IGD he/she drew up is correct.

F1: "there is a student <first name> <surname>."

Peter	Johnson
John	Hartman
Elsa	Doyle
Maria	Jones
Fred	Smith

F2: "the mentor of student <first name> <surname> is <teacher code>."

Peter	Johnson	BLC
John	Hartman	BLC
Elsa	Doyle	BAK
Maria	Jones	VRM
Fred	Smith	GPB

F3: "the school is offering project <project code>."

P101
P110
P120
P200
P203
P204

F4: "project <project code> is supervised by <teacher code>."

P101	BLC
P110	LEK
P120	FEL
P200	BAK
P203	BAK
P204	FEL

F5: "project <project code> concerns <description>."

P101	developing a timekeeping system
P110	introducing an RDBMS into a business
P120	converting a dBASE system to Foxpro
P200	developing a technical information system
P203	writing course material on FCO-IM
P204	implementing a design for a database

F6: "the <ordinal number> preference of student <first name> <surname> is project <project code>."

first	Peter	Johnson	P101
second	Peter	Johnson	P203
third	Peter	Johnson	P110
first	John	Hartman	P203
second	John	Hartman	P101
third	John	Hartman	P200

F7: "student <first name> <surname> was allocated project <project code>."

Peter	Johnson	P101
John	Hartman	P203

Figure 2.19: all sentences from the IGD in figure 2.17

2.7 Semantically Equivalent Models

It is often possible to analyze the same set of elementary fact expressions in different ways in the classification and qualification step of FCO-IM. This leads to different IGDs, which all yield the same sentences on regeneration. We call alternative IGDs from which the same fact expressions can be regenerated *semantically equivalent.* In this section, we will classify and qualify one set of fact expressions in three different ways.

Our example here is a UoD containing two-dimensional graphical figures. The greatest width and height of these figures is of interest. A domain expert phrases the following four fact expressions (FEs):

FE 1: "The width of figure 7 is 102 mm."
FE 2: "The height of figure 7 is 61 mm."
FE 3: "The width of figure 8 is 61 mm."
FE 4: "The height of figure 8 is 48 mm."

2.7.1 First Way: One Fact Type Expression

In the first way of analysis, the four sentences are all assigned to a single class called Measurement in the first stage of classification and qualification. Figure 2.20 shows the result of the rest of the analysis following from the dialogue between the analyst and the domain expert. Three object type expressions are identified for object types Dimension, Figure and Distance. Label type 'dimension name' can supply only two values: 'height' and 'width'. Figure 2.21 shows the corresponding IGD.

Measurement:

"The width of figure 7 is 102 mm."
Dimension:O1 Figure:O2 Distance:O3

F1: "<Dimension:O1> of <Figure:O2> is <Distance:O3>."

'the width' 'figure 7' '102 mm'
dimension name
figure number
number of mm

O1: 'the <dimension name>'
O2: 'figure <figure number>'
O3: '<number of mm> mm'

Figure 2.20: FE analysis, first way

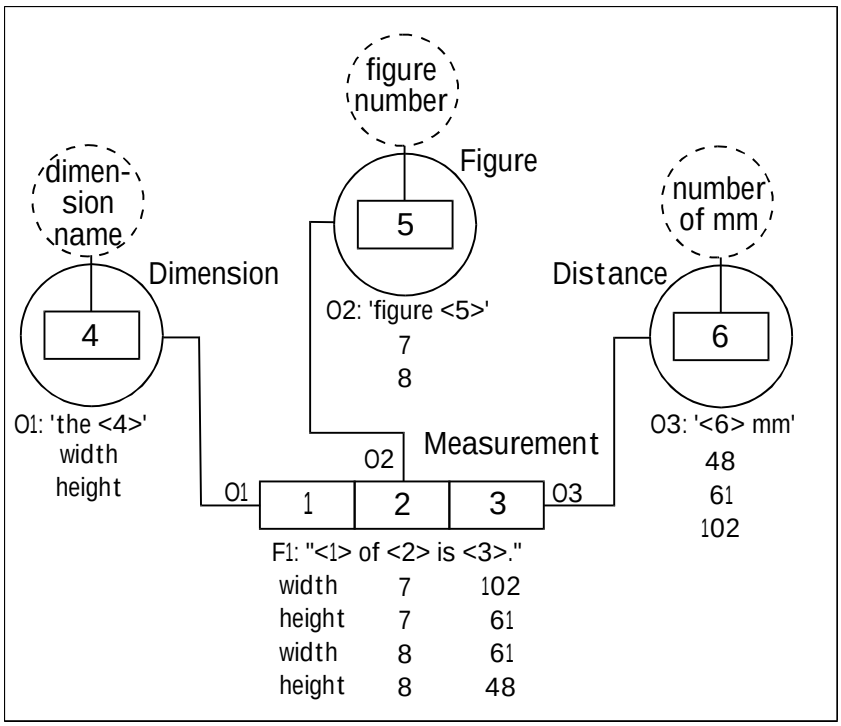


Figure 2.21: IGD, first way

2.7.2 Second way: One Fact Type Expression with an extra Nominalization

In the second way of analysis, the four sentences are all assigned to one class again, but in the second stage of classification and qualification, only two object type expressions are recognized at first: Figure Dimension and Distance (see figure 2.22, in which the codes for the object type expressions etc. are kept the same as in figure 2.20 as much as possible). The domain experts consider the sentence part ‘the width of figure 7’ to identify an object of object type Figure Dimension. The term ‘figure dimension’ is common jargon in the UoD. The rest of the analysis is shown in figure 2.22. The corresponding IGD is in figure 2.23.

Measurement:

"The width of figure 7 is 102 mm."
Figure Dimension:O4 Distance:O3

'the width of figure 7' '102 mm'
Dimension:O1 Figure:O2 number of mm

'the width' 'figure 7'
dimension name
figure number

F1: "<Figure Dimension:O4> is <Distance:O3>."

O4: '<Dimension:O1> of <Figure:O2>'

O1: 'the <dimension name>'

O2: 'figure <figure number>'

O3: '<number of mm> mm'

Figure 2.22: FE analysis, second way

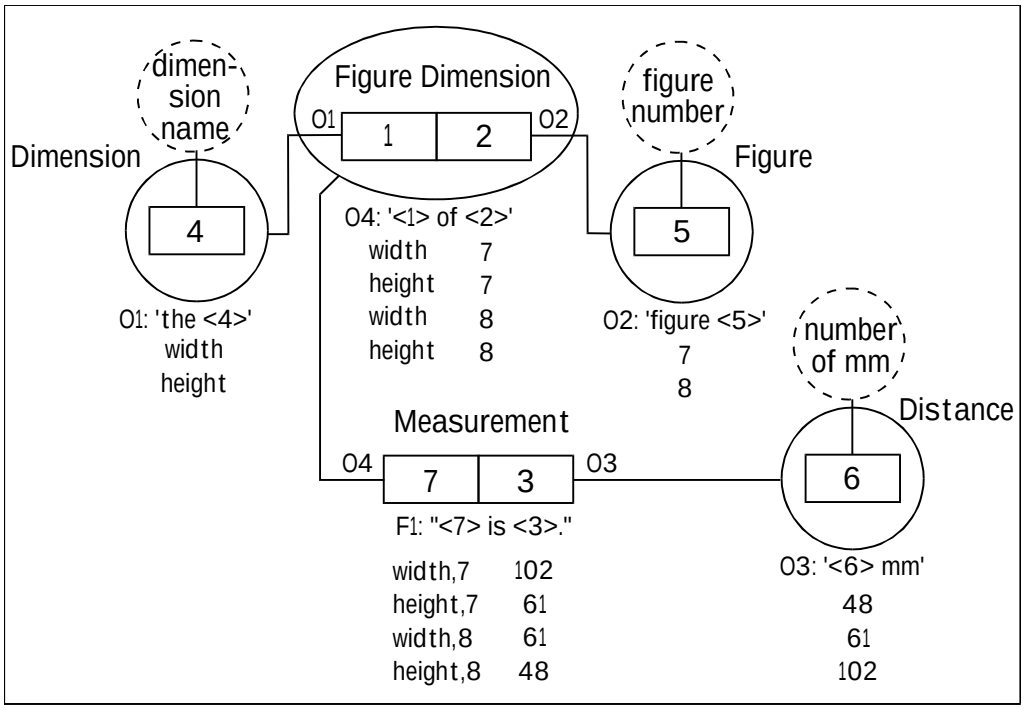


Figure 2.23: IGD, second way

2.7.3 Third Way: Two Fact Type Expressions

In the third way of analysis, the four sentences are assigned to two classes: FE 1 and FE 3 to class Measurement of Width, and FE2 and FE4 to class Measurement of Height. Sentence parts ‘the width’ and ‘the height’ are now taken as constant text instead of an object expression. The result of the further analysis is shown in figure 2.24 and the corresponding IGD is in figure 2.25.

Measurement of Width:

"The width of figure 7 is 102 mm." — F1a: "the width of <Figure:O2> is <Distance:O3>."
Figure:O2 Distance:O3

'figure 7' '102 mm' — O2: 'figure <figure number>' — O3: '<number of mm> mm'
figure number — number of mm

Measurement of Height:

"The height of figure 7 is 61 mm." — F1b: "the height of <Figure:O2> is <Distance:O3>."
Figure:O2 Distance:O3

Figure 2.24: FE analysis, third way

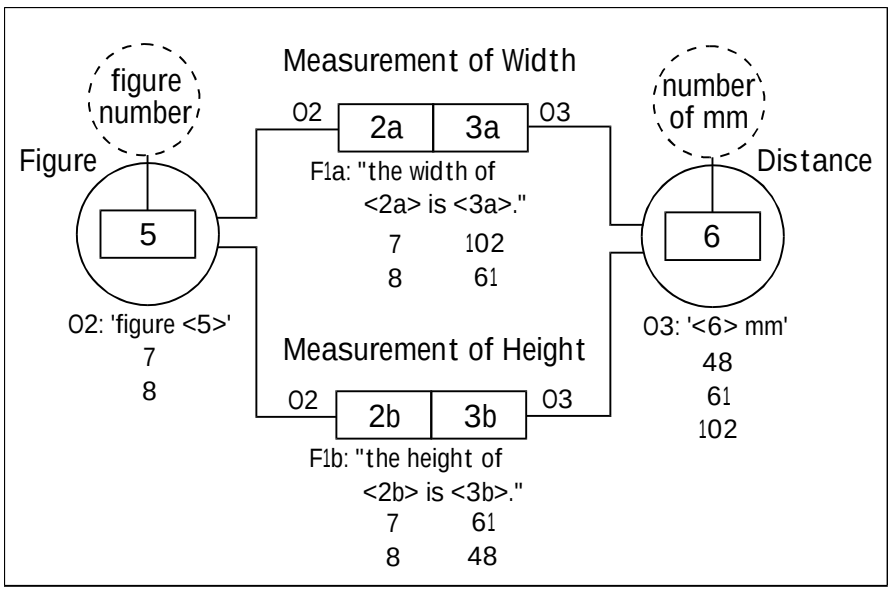


Figure 2.25: IGD, third way

2.7.4 Semantically Equivalent Transformations

The three IGDs in figures 2.21, 2.23 and 2.25 all generate exactly the same fact expressions, which is why we call them *semantically equivalent*. The IGD in figure 2.21 can be converted to (*transformed* into) that in figure 2.23 by introducing an extra nominalization. The reverse route (from figure 2.23 to figure 2.21) is also possible. Therefore, this is called a *nominalization - denominalization transformation*. The IGD in figure 2.21 can be transformed into that of figure 2.25 by dropping object type Dimension and creating a separate fact type - with one less role- for each of the two possible values of the corresponding label type 'dimension name'. The reverse route is also possible. Therefore, this is called an *object type - fact type transformation*.

These different possibilities of analyzing the same fact expressions occur frequently in practice. It depends on the analyst and on the domain expert, which form of model, is arrived at initially: others might model the same fact expressions differently. Which way would be best should be examined case by case, and is usually strongly determined by considerations of implementation, such as performance, which in turn depends on the software to be used and on the frequency of transactions. We will give some guidelines for the nominalization - denominalization transformation in section 3.3.2, and in section 5.1 we will demonstrate that semantically equivalent IGDs may lead to different relational schemas. These transformations can therefore be used to tune the database design.

2.8 Derivable Fact Types

Let us suppose that the management of the educational institution of our student-project case study needs the following data:

- for each teacher: the number of students he/she is to mentor
- for each project: the number of times the project was entered as being preferred.

A concrete example specifying this information is depicted in figure 2.26. We have derived this information from the data in the example documents of figure 2.6. Such information is therefore called *derivable information.*

Verbalization in elementary fact expressions yields sentences such as "BAK is the mentor of 2 students." and "Project P101 was entered 5 times as a preference.". Consequently, we arrive at two new *derivable fact types* Pupil Count and Preference Count, with corresponding fact type expressions:

F8: "<Teacher:O2> is the mentor of <number> students."

F9: "<Project:O3> was entered <number> times as a preference."

Teacher	Pupil Count
BAK	2
FEL	0
BLC	2
LEK	0
GPB	1
VRM	2
JPC	1
HVL	1

Project	Preference Count
P101	5
P102	1
P110	3
P115	1
P120	1
P200	3
P201	3
P203	3
P204	1

Figure 2.26: concrete examples of derivable information

These fact types are shown in figure 2.27 in an IGD. It is a supplement to the IGD in figure 2.17. Derivable fact types are marked with an asterisk (*) following the fact type name.

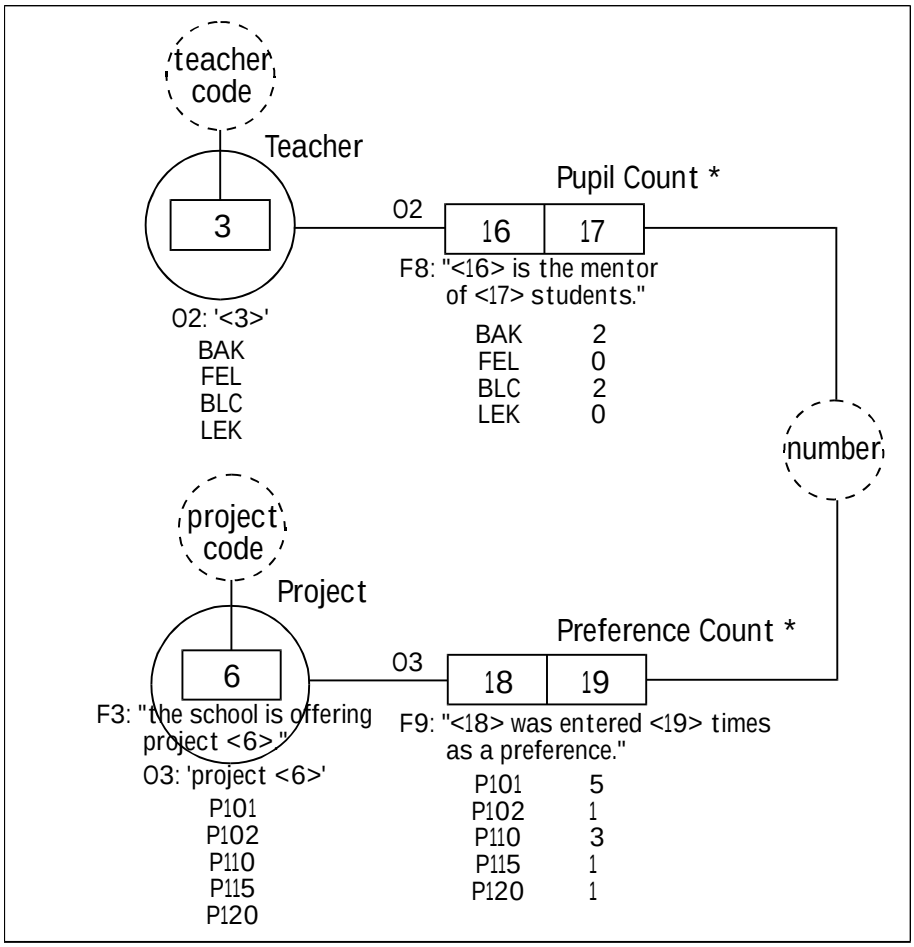


Figure 2.27: IGD of derivable fact types

The population of the derivable fact type Pupil Count is calculated for each teacher name in the population of fact type Teacher in figure 2.17 by counting the number of times that this teacher name occurs in the population of role 5 in fact type Mentorship. This calculation is the *derivation rule* for fact type Pupil Count. Similarly, the population of the derivable fact type Preference Count is calculated for each project code in the population of fact type Project by counting the number of times that the project code occurs in the population of role 13 in fact type Preferences. This calculation is the derivation rule for fact type Preference Count.

When the population of one or more of the fact types Project, Teacher, Mentorship or Preferences changes, the populations of the derivable fact types must be recalculated according to their derivation rules.

Derivable fact types appear frequently on example documents, such as (sub)totals on invoices or receipts. We recommend to draw up an IGD for the non-derivable fact types (the *basic fact types*) first and to model the derivable fact types later in a separate IGD*.

2.9 Introducing New Identifiers

Sporadically, there turns out to be no proper verbal identifier for the objects of a certain object type, for instance if one 'identifier' can refer to more than one object (see the example below), or if the communication is mainly of a graphical nature (e.g. drawings, schemas), so that different objects can be pointed out on the drawing ('this item here'), but cannot be designated verbally. In such cases, new identifiers must be introduced that are not present in the original example documents.

In our student-project case study, students are identified by the combination of their first name and surname. This is not very realistic: it would mean that a second student with the same first name and surname would not be admitted to this educational institution because its information system cannot distinguish between these two students. The analyst should always check that each object type is identified properly, so he asks the School's Project Coordinator: "What if there are two students called Peter Johnson? Would you name them Peter Johnson 1 and Peter Johnson 2? Wouldn't it be better to start identifying students with a unique student number for each student?" The Project Coordinator indeed decides to assign such a unique identifying number to each student.

It is not hard to see how the concrete example documents (figure 2.6) should be changed. In the list of preferences and the list of allocations, an extra column 'student number' should be added and for each student a unique number should be chosen. The verbalizations in section 2.3, fact expressions 1 through 9, should now for instance be changed to:

1) "There is a student with number S1."
2) " " " " " " " S2."

and so on.
The corresponding existence postulating fact type expression is:

F1: "there is a student with number <student number>."

In the other fact expressions, students are now referred to by 'student S1', 'student S2' and so on, so the new object type expression (replacing the old O1) is:

O1: 'student <student number>'

This also illustrates the benefit of modeling in a redundancy free way: a completely different way of identification causes changes in the IGD in one place only, namely in object type Student. All we have to do is change O1 to be able to use this new identifier in all other fact type expressions.

The names of students can next be modeled in two fact types Student First Name and Student Surname, with fact type expressions:

F8: "the first name of <Student:O1> is <first name>."
F9: "the surname of <Student:O1> is <surname>.".

The new IGD is depicted in figure 2.28.

Sometimes, there is indeed a proper identifier, but the analyst and/or the domain experts consider it to be very impractical, for instance because it consists of many components. They may decide to introduce a new identifier for reasons of efficiency. When this happens, a certain object type will have two identifiers. Often one of these identifiers is chosen as the only one allowed in the communication. That identifier is then known as the *primary identifier*. Only the primary identifier will then appear as a fact type inside the ellipse in the IGD, and as an object type expression outside the ellipse. If this were the situation in our student-project case study, then we would say that a student number is the *primary identifier* of an object of object type Student, and that the combination of first name and surname offers an *alternative identifier*.

In the case where there are two equivalent identifiers, neither of which is to be the primary identifier, we have a *synonym* problem. Synonyms (i.e. different names for the same object) can be modeled in FCO-IM without any trouble using the concept of *generalization*. This is discussed in section 6.2.3.

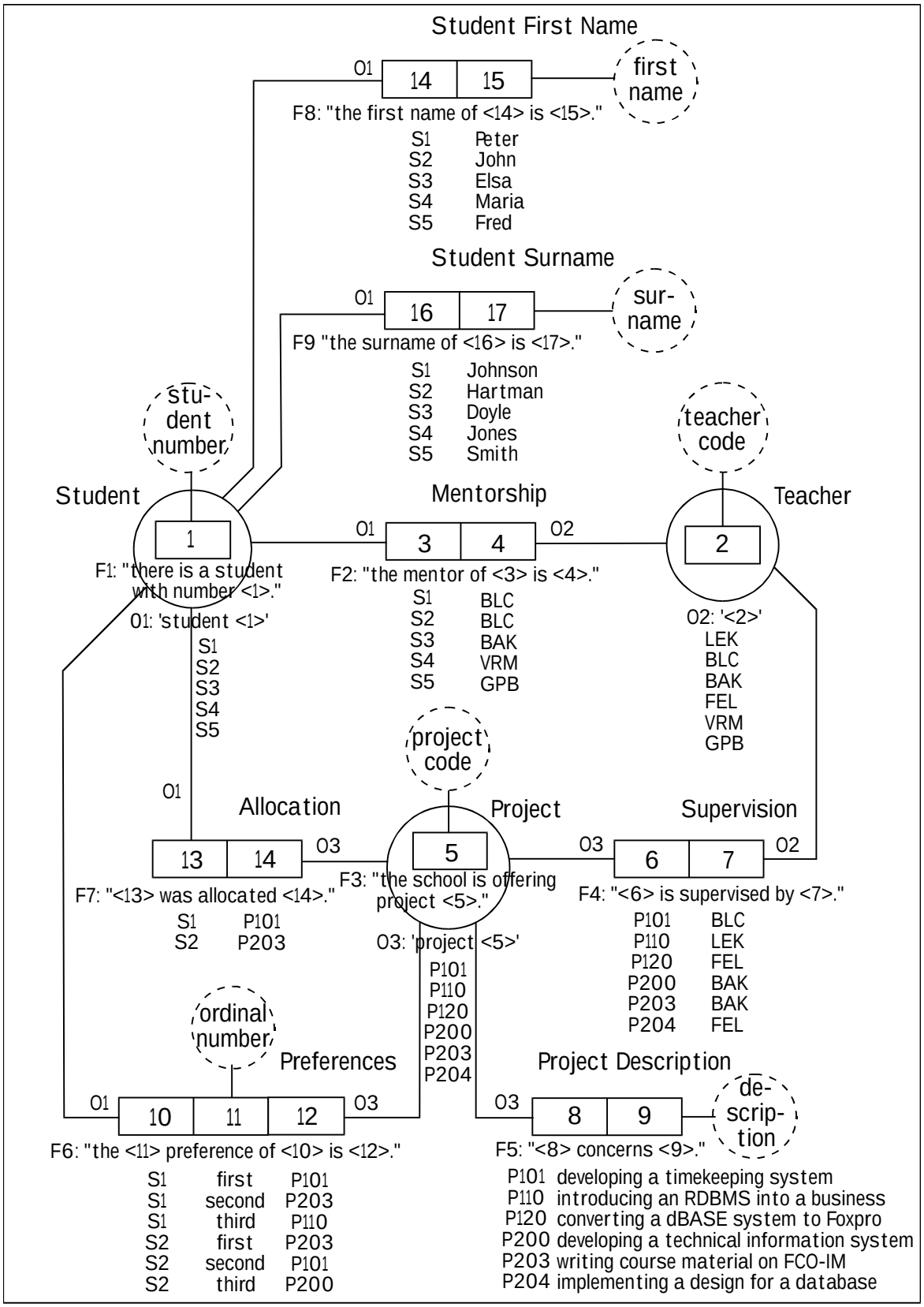


Figure 2.28: IGD with new identifier for Student

2.10 Subtle Substitution

In figure 2.17, the codes of the object type expressions to be substituted are placed next to the lines connecting roles to nominalized fact types. It would still be possible to regenerate the fact expressions from this IGD if we leave these codes out and even if we also omit the identifying codes of the fact type expressions and object type expressions. The reason is that there is only one fact type expression for each fact type and only one object type expression for each object type in figure 17, so there can be no doubt which fact type expression or object type expression we should choose. However, it is possible in FCO-IM for an object type to have more than one object type expression. It is also possible to have more than one fact type expression for each fact type, although this seldom occurs in an IGD with elementary fact types. However, we will show in chapter 4 that this does occur frequently during the derivation of a relational schema, in which IGDs containing non-elementary fact types arise. Here, we will give an example of both phenomena in an elementary IGD.

Let us suppose that the domain expert, on validating the regenerated fact expressions in figure 2.19, remarks that the sentences of fact type expressions F1 up to F6 are fine, but that he prefers to verbalize the facts of fact type Allocation differently, using fact expressions such as: "Peter Johnson was allocated project P101.", omitting the word 'student'. In addition to the existing object type expression O1: 'student <1> <2>' for object type Student, a second object type expression O4: '<1> <2>' must now be given, which applies for role 14. Figure 2.29 shows that the code O4 is now written next to the line joining Student to role 14, whereas O1 remains written next to roles 4 and 11. As soon as more than one object type expression belongs to an object type, the codes must be indicated explicitly next to the roles, otherwise we would not know which object type expression to choose. Since this happens regularly, we will put the codes in everywhere, although it would be no problem to do so only where necessary.

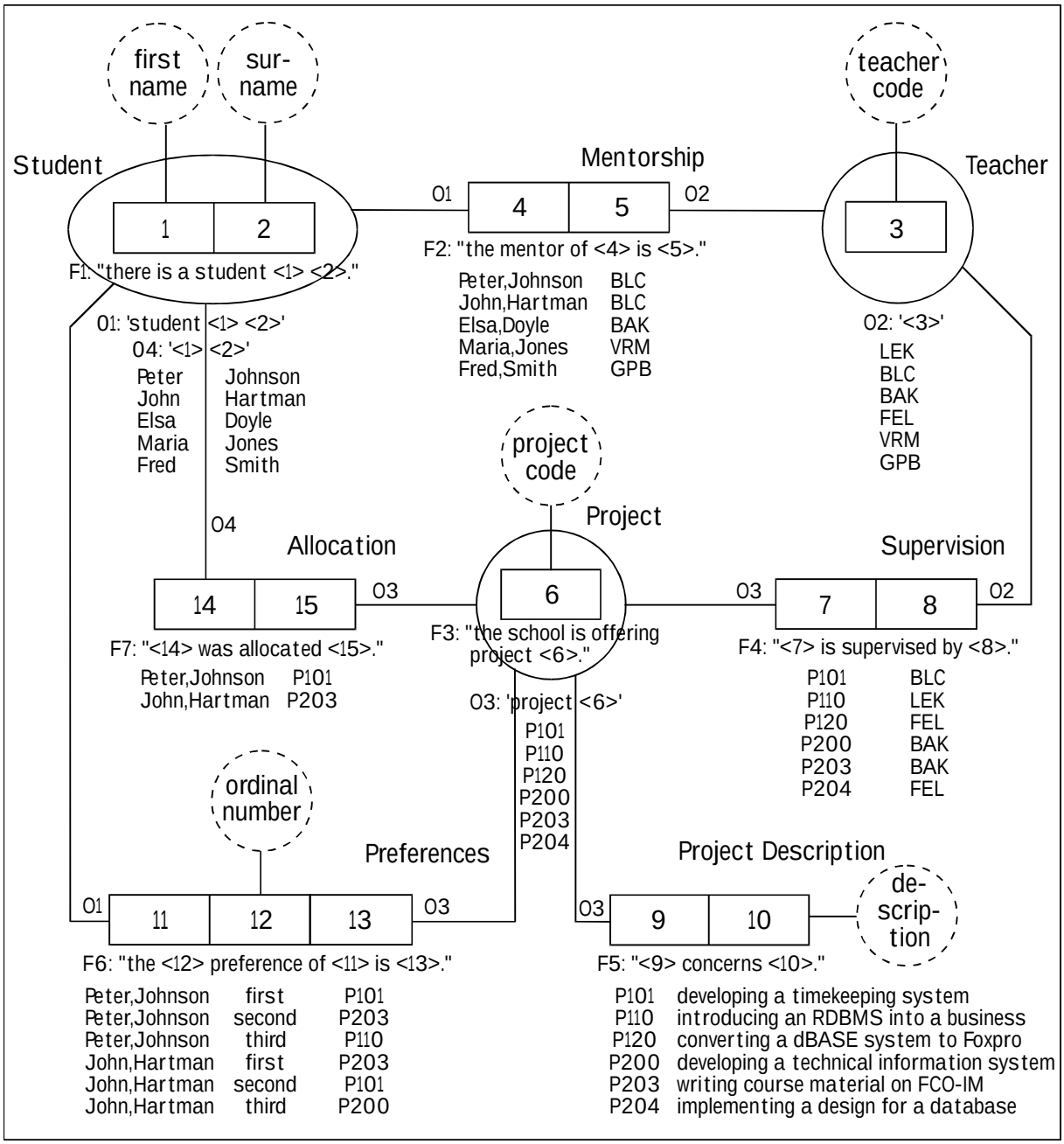


Figure 2.29: IGD with more than one OE per object type

An even more subtle construct is possible. Let us suppose that the School's Project Coordinator approves sentences 10 through 18, but that the administrative department of the school wants to retain the following verbalization, which they always use:

74) "Peter Johnson is a pupil of BLC."
75) John Hartman " " " " BLC
76) Elsa Doyle " " " " BAK
...
82) Tom Dakota " " " " HVL

On the basis of fact expressions 74 through 82, the analyst arrives at a new fact type expression F8 for verbalizing facts of fact type Mentorship, in addition to the existing fact type expression F2:

F2: "the mentor of <Student:O1> is <Teacher:O2>."
F8: "<Student:O4> is a pupil of <Teacher:O2>."

Please note that O1 should be substituted into F2, and O4 into F8. Substituting different object type expressions into different fact type expressions of the same fact type is called *subtle substitution*. In the IGD, we indicate this by writing O1:F2 (O1 goes into F2) and O4:F8 (O4 goes into F8) next to the line joining role 4 to object type Student.

The complete IGD is given in figure 2.30, in which this notation is used throughout, even where it is not necessary. In the rest of this book, we will use it only where it is essential. The FCO-IM tool always records the object type expressions to be substituted in this subtle way in order to enable the use of subtle substitution, but it displays the simpler notation wherever possible.

Please check, that all fact expressions are regenerated correctly from the IGD in figure 2.30:

F2: "the mentor of student <first name> <surname> is <teacher code>."

Peter	Johnson	BLC
John	Hartman	BLC
Elsa	Doyle	BAK
...	...	...

F8: "<first name> <surname> is the pupil of <teacher code>."

Peter	Johnson	BLC
John	Hartman	BLC
Elsa	Doyle	BAK
...	...	...

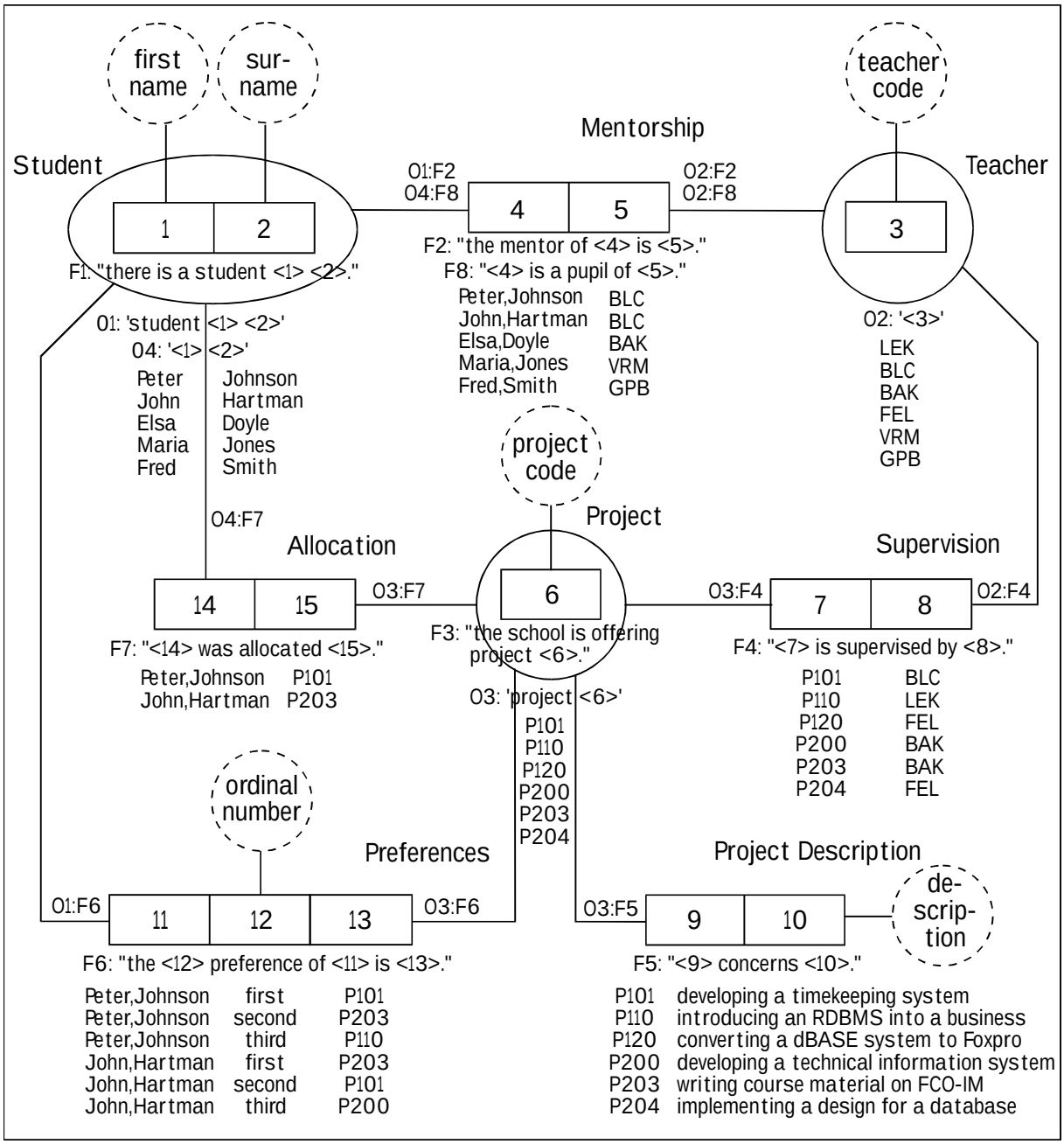


Figure 2.30: IGD with subtle substitution

2.11 Tuple Numbers and Tuple Pointers

So far, all IGDs have been populated with tuples consisting of labels. Although this way of populating an IGD is generally the most convenient for the information analyst, it does cause labels to be recorded *redundantly* (i.e. more often than strictly necessary). For example, we would need to correct a spelling mistake in such a label in more than one place. In this section, we will explain how to populate an IGD in a redundancy free way, which is implemented in the FCO-IM tool. We will use *tuple numbers* and *tuple pointers*.

We start with giving each tuple an arbitrary tuple number, which is unique for each tuple per fact type. Figure 2.31 shows a part of the IGD of our student-project case study. In the top half of figure 2.31, a tuple number has been assigned to all tuples. To distinguish tuple numbers from role populations, we append a colon and do not write them underneath roles.

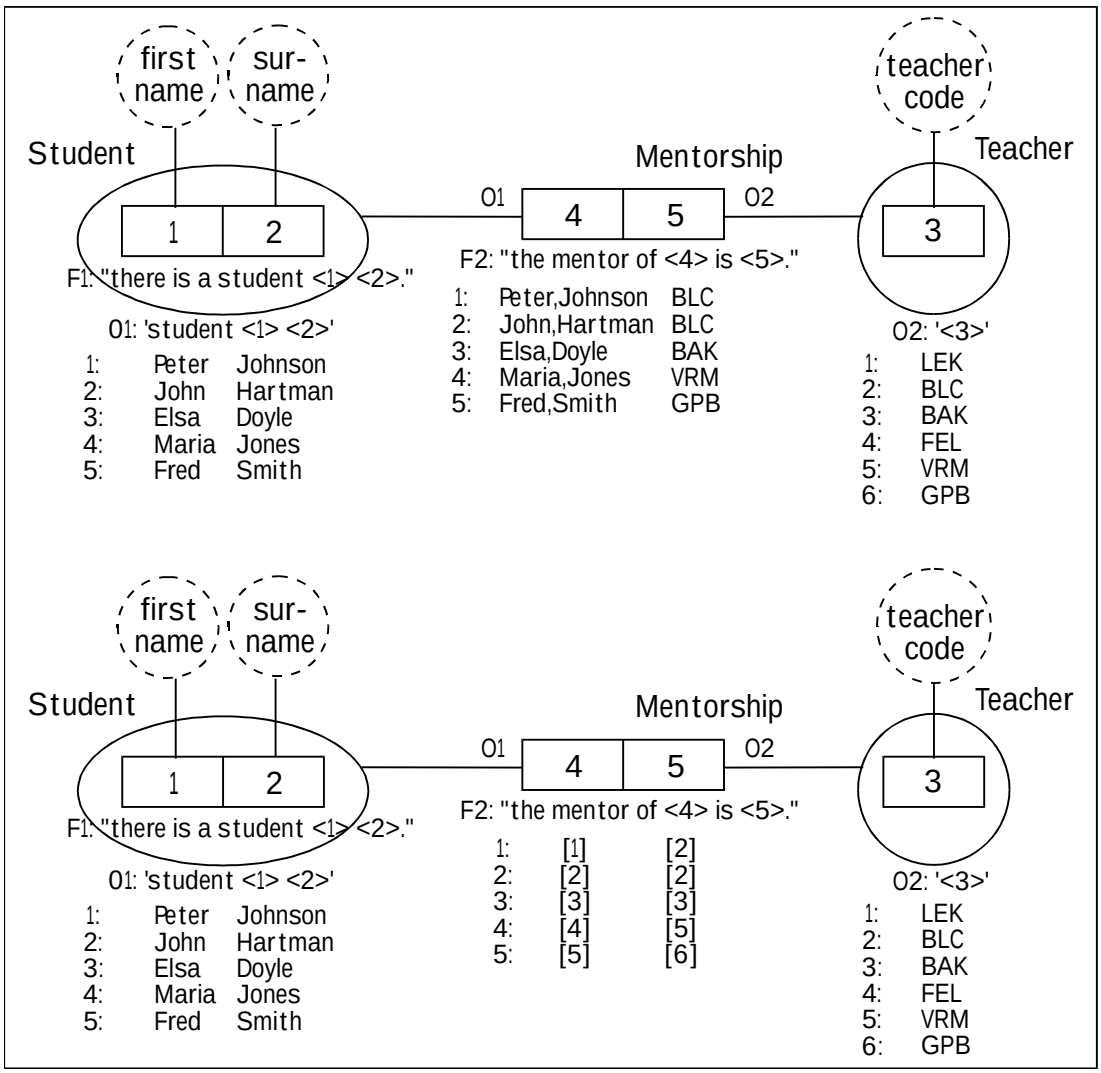


Figure 2.31: tuple numbers and tuple pointers

If a role is played by a label type, then labels are written underneath the role as usual. But if a role is played by a nominalized fact type, then we replace each label, or combination of labels, with a pointer to the correct tuple in the population of the nominalized fact type itself. Such a pointer consists of the number of the tuple pointed to between square brackets. The brackets serve to distinguish tuple pointers from labels. In the bottom half of figure 2.31, tuple pointers are being used. For example, tuple number 5 of fact type Mentorship reads '[5] [6]', in which '[5]' sits underneath role 4 and points to tuple number 5 of nominalized fact type Student, which plays role 4. Tuple pointer [5] therefore means that we should actually consider tuple 5 of fact type Student here. Tuple 5 of Student reads: 'Fred Smith'. Tuple pointer '[6]' underneath role 5 points to tuple number 6 of fact type Teacher, since Teacher plays that role. Tuple 6 of Teacher reads: 'GPB'. If we keep following the tuple pointers (often down more than one level), then we will eventually retrieve all the labels to be filled in.

Of course, only tuple pointers to actual tuples may be used. Translated back to label populations, this means that all labels underneath roles played by nominalized fact types must also be present underneath these fact types themselves.

In regenerating fact expressions from an IGD, all tuple pointers must now be processed as well. This is done by replacing the tuple pointers with the tuples to which they refer. This process continues until the label level is reached everywhere. In this way, tuple 5 of fact type Mentorship generates the fact expression: "The mentor of student Fred Smith is GPB.". For easy legibility, in this book we will present IGDs populated completely with labels instead of tuple numbers and tuple pointers. The FCO-IM tool can generate IGDs populated one way or the other as desired.

Figure 2.32 contains the complete IGD of figure 2.17 with tuple numbers and tuple pointers.

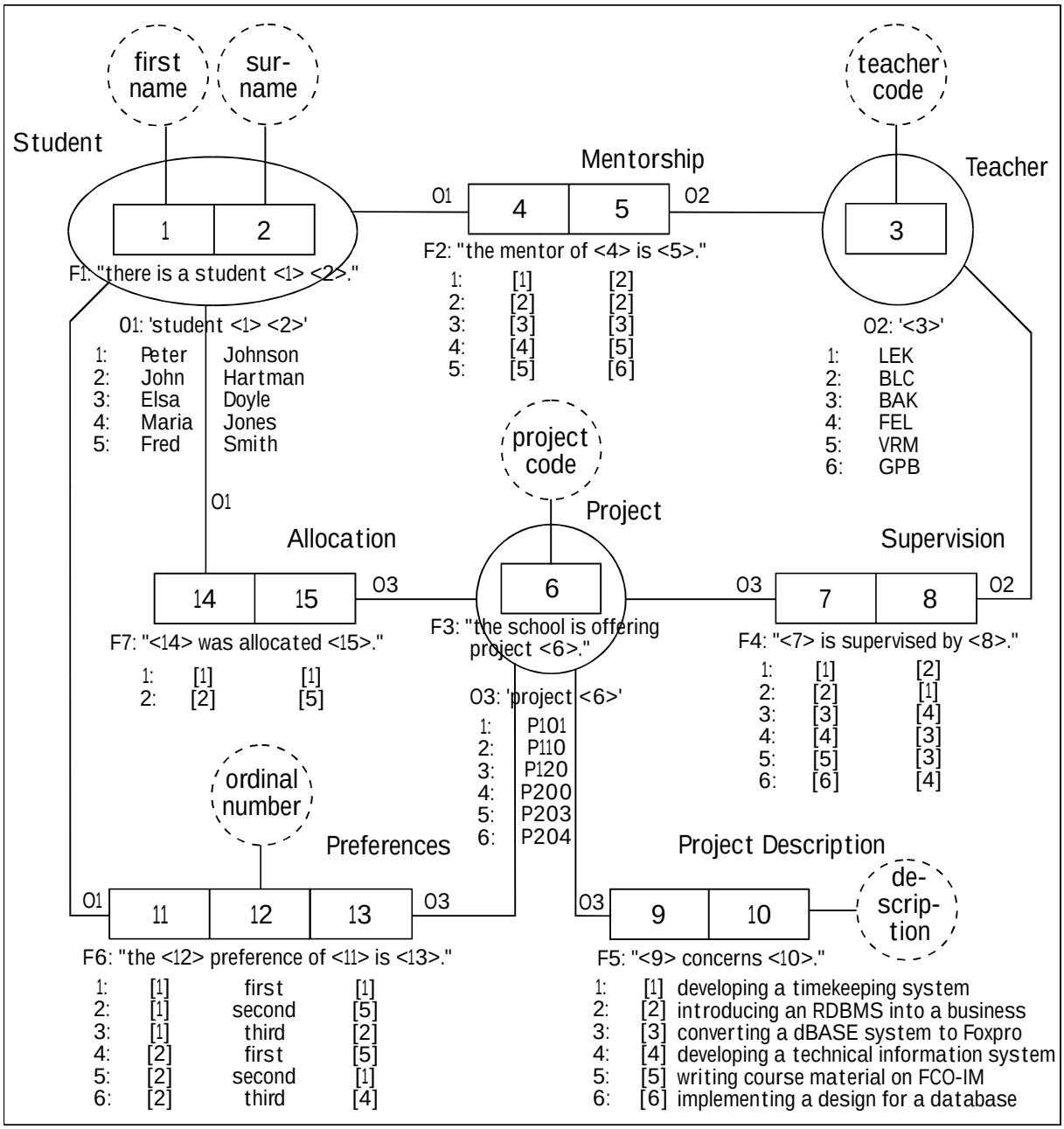


Figure 2.32: IGD without redundant labels

3

Constraints

In the previous chapter, we saw how the first steps of information analysis using FCO-IM work through the verbalization of concrete examples and the classification and qualification of these verbalized expressions. The results can be arranged and reproduced conveniently and in a redundancy free way in an information grammar diagram (IGD). From a populated IGD, the original sentences in which the facts were expressed can be precisely regenerated.

But that is not enough. A grammar should not only be able to regenerate the communication out of a valid population, but should also exclude invalid populations as far as possible. In the IGD from figure 2.17, for example, it is still not precluded:

1. …that a fact can be recorded twice, for example the fact that Peter Johnson has mentor BLC: the IGD does not preclude that fact type Mentorship can be populated with the same two tuples 'Peter,Johnson BLC'.
2. ...that a fact such as "The fourth preference of student Peter Johnson is project P204." can be recorded, whereas a student can give only three preferences.
3. …that the following two fact expressions are both recorded: "The mentor of student Peter Johnson is BLC." and "The mentor of student Peter Johnson is BAK.".
4. …that a student is recorded in fact type Student without recording a mentor for the same student in fact type Mentorship.

Situation 1 leads to redundancy, while situations 2, 3 and 4 appear to be contrary to the starting document and the example documents. The IGD must still be refined in the sense that the redundant tuples are forbidden (situation 1), in the sense that tuples (situation 2) or combinations of tuples (stiuation.3) that are not allowed in the UoD are forbidden, and in the sense that if a certain tuple is given another tuple must also occur (situation 4). We do this with so-called *constraints*, of which the most important are discussed in this chapter.

We begin in section 3.1 with the simplest sort: *value constraints*. Then *uniqueness constraints* follow in section 3.2. These are by far the most important constraints because they largely determine whether a fact type is elementary or not. In section 3.3 we will also discuss tests that have to be done after the determination of the uniqueness constraints. These tests might

still change the fact type structure. After that we will successively deal with *totality constraints* (section 3.5), *exclusion constraints* (section 3.6), *cardinality constraints* (section 3.7) and *other constraints* (in the final remarks in section 3.8). In an IGD, all constraints can be added using graphic symbols.

By the way, constraints cannot prevent the recording of a fact that is simply incorrect in content: if Peter Johnson gives the wrong mentor, then there will be an incorrect tuple in the population of fact type Mentorship. All constraints particularly concern the *form* of (combinations of) tuples, not their content. In situation 3, for example, it is clear that the tuples, irrespective of their content, cannot both occur, but which tuple is correct (or are they both wrong?) cannot be decided without gaining further information.

Figure 3.1 shows once more the starting document from section 2.1; we now have words underlined that concern the constraints:

Our students are required to participate in an industrial project in the first term of their fourth year. During such a project, the students carry out a task in the field of information systems development in a business or in a non-profit organization. Students can choose their first, second and third preference (all different) from a list of project tasks. This list shows each project offered, with the project supervisor (a teacher coordinating the project) and a short project description. The students base their preferences on these data. They can enter their choices on another list, on which I have already filled in their name and their mentor (each student has a teacher as a personal student adviser). If they do so before the date on which I assign them to their project, then I try to meet their preferences as best I can. A list of project assignments is subsequently posted on the notice board. Students who have not given their preferences in time will not be assigned to a project at this stage; I allocate remaining projects to them later. In any one year, there are about 200 students, so I would welcome any form of computerized support for the administration of this task.

Figure 3.1: starting document with constraints underlined

We note that a starting document is practically always very incomplete where it concerns the constraints. The same applies for concrete example documents, from which the constraints can practically never be obtained without further interviews with the domain expert.

3.1 Value Constraints

A label type can be seen as a source from which, without further limitation, values of a certain type can be drawn. Often, however, there are clear boundaries on the values that can be used. For example: with a label type ‘total percentage’ only values between 0 and 100 are allowed, or with a label type ‘month-code’ only the values ‘jan’, ‘feb’, ‘mar’, ‘apr’, ‘may’, ‘jun’, ‘jul’, ‘aug’, ‘sep’, ‘nov’, ‘dec’ can be used. These sorts of constraints on the possible values of the label types are expressed through the use of value constraints.

Value constraints are imposed on label types. They specify the collection of labels that in principle can be filled in under the roles played by the label types. The possible role populations are then restricted in the following sense: only values allowed by a value constraint can occur in the populations of the roles played by the label type with this value constraint. A value constraint is displayed graphically next to the concerned label type by listing the allowed values between curly brackets. In the above examples the notation would be: {0..100} (which means: from 0 up to and including 100) and {jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec}.

Look at the following underlined part from figure 3.1:

> Students can choose their first, second and third preference (all different) from a list of project tasks.

In the concrete example documents (figure 2.4), there is no column for a preference other than first, second, or third. Inquiries to the School’s Project Coordinator confirm that only the ordinal numbers first, second, or third are allowed. So we impose value constraint 1: {first, second, third} on label type ‘ordinal number’. Figure 3.2 shows the relevant part of the IGD. Situation 2 on page 3.1 is now precluded.

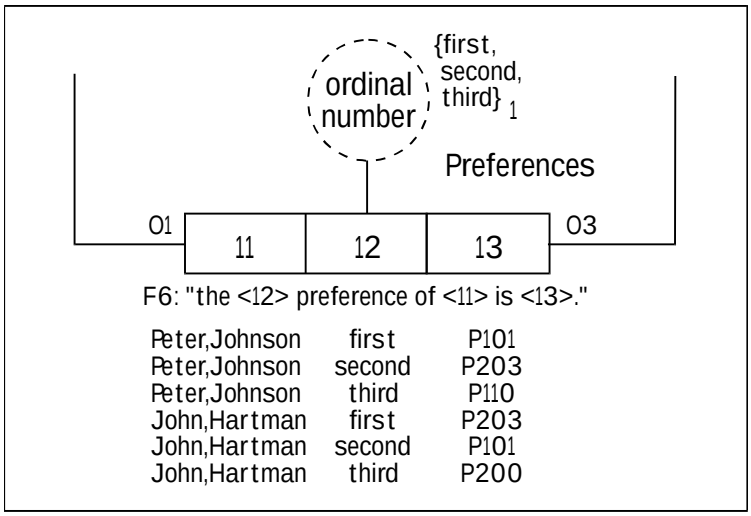


Figure 3.2: value constraint on ‘ordinal number’

3.2 Uniqueness Constraints

A uniqueness constraint (UC) is a constraint that indicates that values (or combinations of values) can occur only once (i.e. they must be *unique*) in the population of a fact type. It will be shown that uniqueness constraints can exclude situations 1 and 3 from page 3.1. In section 3.2.1, we introduce the uniqueness constraints through the help of the student-project case study. In section 3.2.2, a systematic way of working is given for the determination of the uniqueness constraints. Finally in section 3.2.3, some final comments are given.

3.2.1 Uniqueness Constraints in the Student-Project Case Study

A uniqueness constraint (UC) can apply to the population of one or more roles. In an IGD a UC is displayed graphically as an arrow with two heads above the concerned roles. See figure 3.3, to which all the UCs of the student-project case study have been added. We discuss them one by one below. Under a UC, a label (or combination of labels) can occur only once. Putting it another way, an arrow means: no duplicates below me. A uniqueness constraint on only one role is called a *single role uniqueness constraint.* A UC on more than one role is called a *multiple role uniqueness constraint.*

To begin with, we impose the requirement on every fact type, nominalized or not, that every tuple can occur only once in its population. This is to rule out redundancy through the recording of the same fact expression or object expression more than once (situation 1 on page 3.1). For this reason, every unary fact type is required to have a uniqueness constraint on its single role, since this UC (no duplicates below me) forbids the occurrence of a tuple more than once in the population. That is why in figure 3.3 uniqueness constraint 2 is on role 3 (from fact type Teacher) and UC 4 on role 6 (from Project). We could for the same reasons put a UC on all fact types with more than one role as well, but often there are UCs on a smaller number of roles, as is the case in figure 3.3 with all fact types having more than one role, except fact type Student. If there is a UC on less than the total number of roles in a fact type, then it is also impossible to have the same tuple more than once in the population, as can be verified easily. We will preclude the recording of redundant tuples (situation 1 on page 3.1) with the following well-formedness rule:

> For each fact type, there must be at least one uniqueness constraint that concerns one or more roles of this fact type only.

Uniqueness constraint 2 can also be regarded as the confirmation of the fact, that a teacher is identified by a teacher code: there cannot be two objects of object type Teacher with the same teacher code. The same is true for UC 4: projects have an identifying project code. For the same reasons, UC 1 is on both roles 1 and 2 from fact type Student: the combination of first name and surname identifies a student.

We will now look at the remaining uniqueness constraints in figure 3.3 and at the same time deal with situation 3 from page 3.1. Can we have next to tuple 'Peter,Johnson BLC' also the tuple 'Peter,Johnson LEK' in the population of fact type Mentorship? Or: can a student have two mentors? No, that is not possible according to the School's Project Coordinator; in any case the coordinator registers only one mentor per student. A UC on both roles 4 and 5 would still allow both tuples, because it would only require that the combination of values under roles 4 and 5 is different in every tuple. The depicted UC 3 on role 4 only does prevent recording both tuples together, because now 'Peter,Johnson' can appear only once under role 4, and so only one mentor can be allocated to him. With this, situation 3 is now precluded.

It is also clear from the above paragraph, that UC 3 on only role 4 is stronger (it prohibits more) than a UC on roles 4 and 5 together. Next to UC 3, we could include another UC on both roles, but that would not add anything: everything that it forbids, UC 3 forbids as well, and UC 3 forbids even more. Should there be two uniqueness constraints p and q on any fact type, in which all the roles under p are also under q (i.e.: q is wider than p), then p is stronger than q, and q must be dropped (the smaller, the stronger).

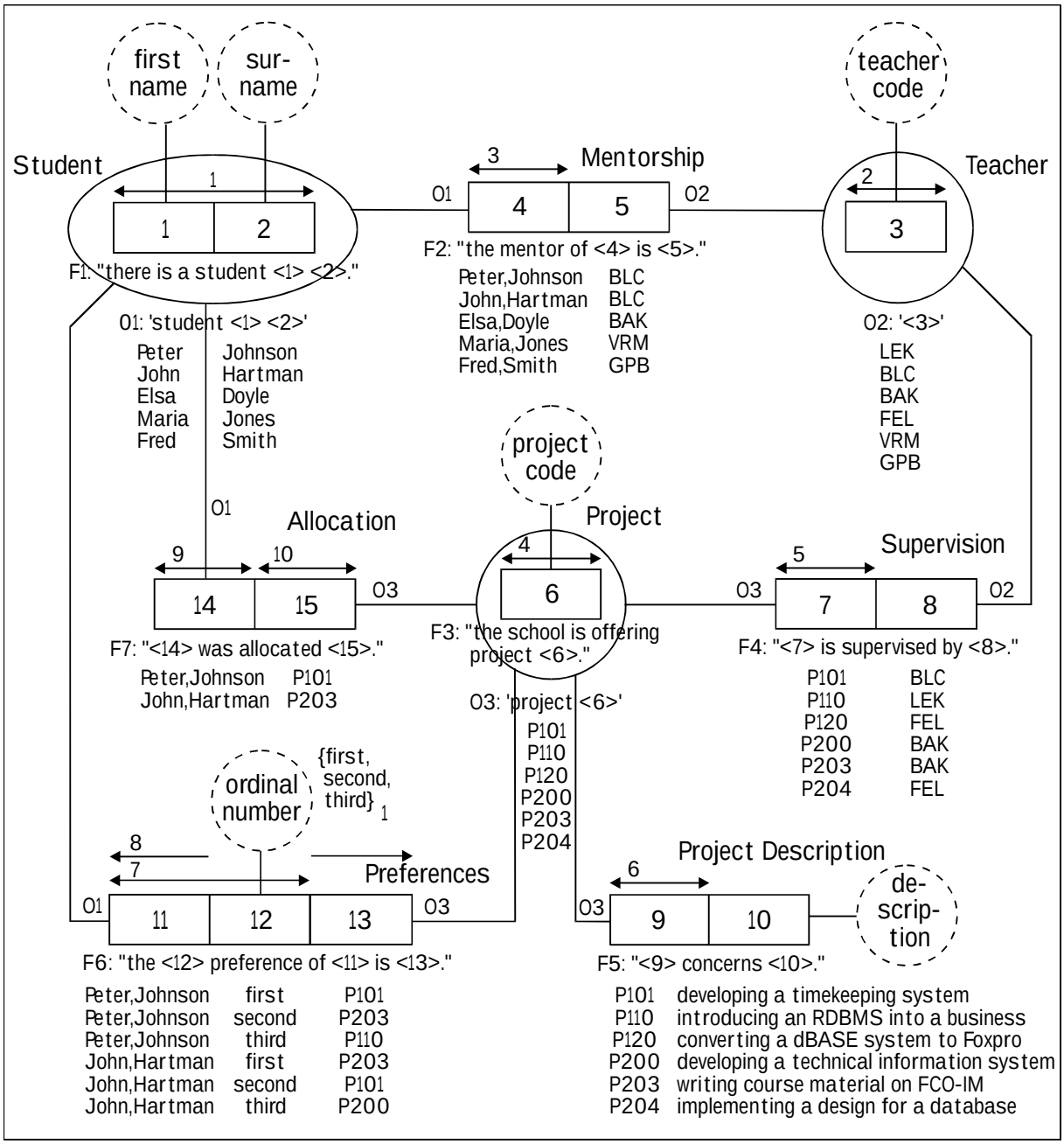


Figure 3.3: IGD with uniqueness constraints

There is no UC on only role 5 because the same mentor can have different students as pupils, as can be seen from the example documents and from the population of fact type Mentorship. Figure 3.4 illustrates the method the analyst uses in looking for possible single role uniqueness constraints. The analyst invents two concrete tuples and asks the domain expert if these two tuples can both occur simultaneously in the population.

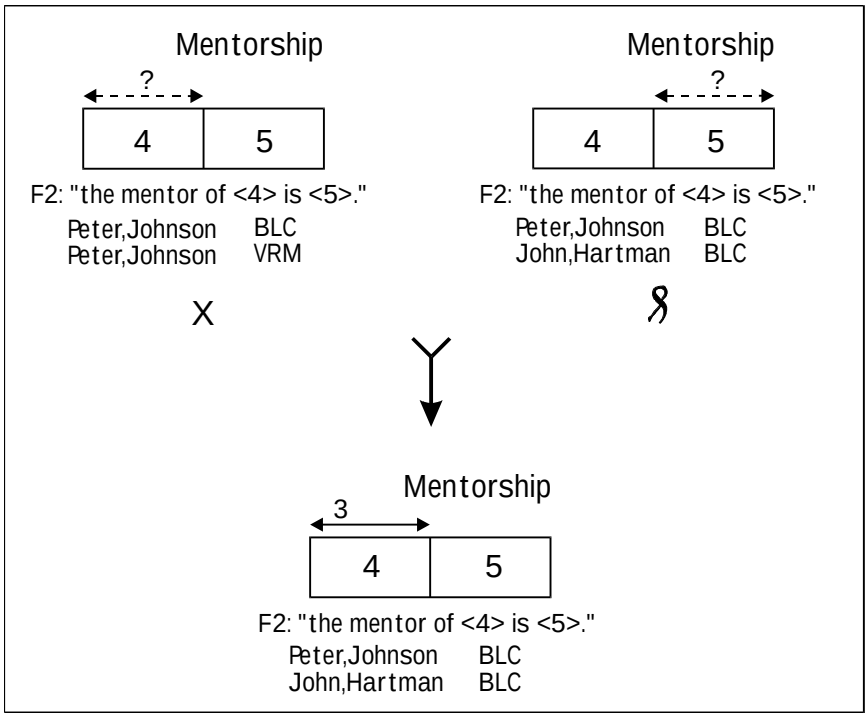


Figure 3.4: UC determination in fact type Mentorship

In the same way, we establish in fact types Supervision and Project Description that single role uniqueness constraints must lie on roles 7 and 9. Figure 3.5 illustrates the method in the determination of the UCs in fact type Allocation. Here two single role UCs are found. Figure 3.6 illustrates the method in the verification of the multiple role UC on fact type Student. It is only after he has checked explicitly that no smaller UCs exist, that the analyst can conclude that a multiple role UC on all the roles exists.

Notice that the tuples used in figures 3.4, 3.5 and 3.6 have not all been taken from the example documents. That would not be possible: examples of 'forbidden' tuples are never found there. Such tuples must be devised by the analyst and presented to the domain expert. Often (but not always) the *absence* of a UC is clear from the example documents. In the document from figure 2.4, for example, two different students appear with the same first name and also two with the same surname, so the analyst does not need to present the tuples in figure 3.6 to the domain expert. However, no two projects with the same project description appear in the examples, so that the absence of a single role UC on role 10 can only be concluded from an explicit question to the School's Project Coordinator. (Analyst: Can the sentences "Project P333 concerns carrying out an information analysis." and "Project P444 concerns carrying out an information analysis." occur together? School's Project Coordinator: "Yes".). The *presence* of a UC, however, can always only be concluded from explicitly asked questions.

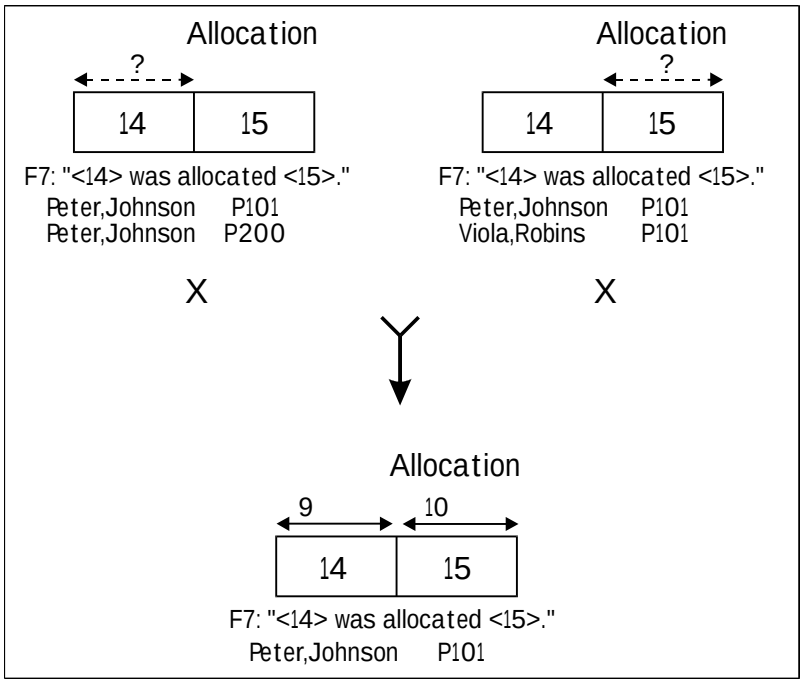


Figure 3.5: UC determination in fact type Allocation

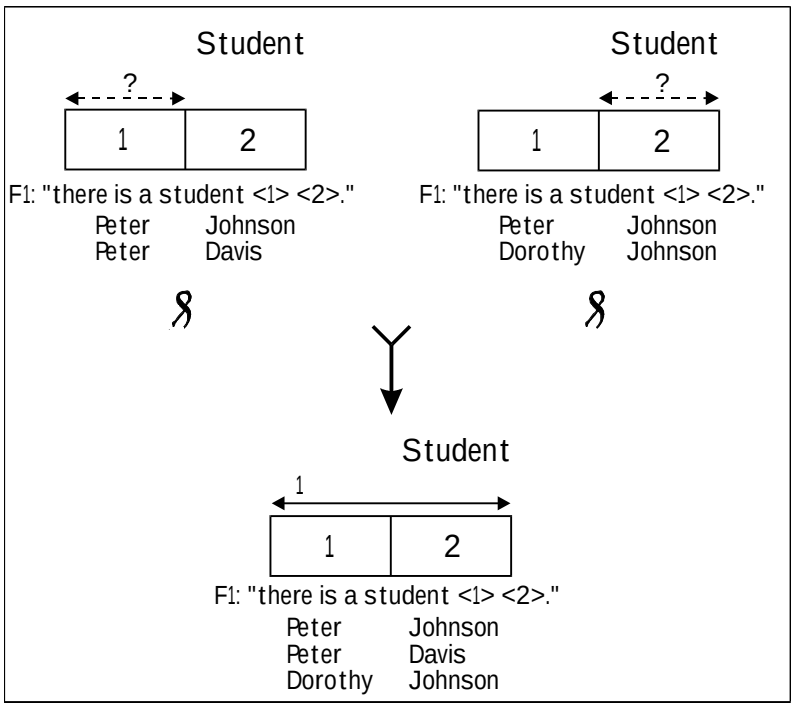


Figure 3.6: UC determination in fact type Student

With a ternary fact type, we first look at UCs on two of the three roles. In fact type Preferences, two out of the three possibilities turn out to exist indeed. Figure 3.7 shows how these are determined using concrete examples and counter-examples.

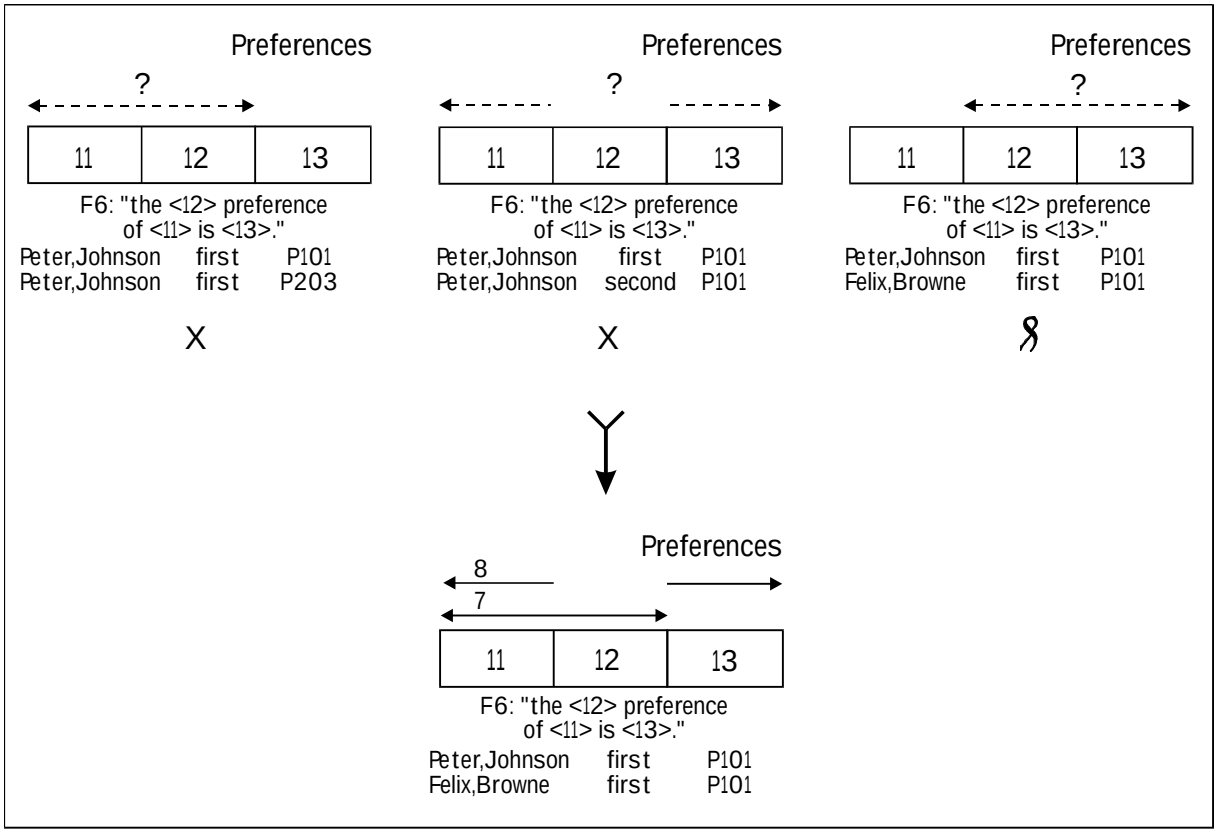


Figure 3.7: UC determination in fact type Preferences

Perhaps there exist smaller UCs in fact type Preferences than the UCs 7 and 8 just found. A look at the example documents shows that this is not so: 'Peter,Johnson' appears more than once under role 11, 'first' appears more than once under role 12 and 'P101' appears more than once under role 13. So there are no single role UCs. All UCs in figure 3.7 have now been discussed.

In the starting document there is only a phrase from which UC 8 follows (but not UC 7), which illustrates the incompleteness of starting documents where constraints are concerned:

> Students can choose their first, second and third preference (all different) from a list of project tasks.

UC 8 forbids the recording of the same project as a preference more than once for the same student (figure 3.7, top center).

3.2.2 Operational Procedure in Determining Uniqueness Constraints

Determination whether a uniqueness constraint exists on a role or combination of roles. See figure 3.8.
Test: does a UC exist on roles 12 + 13?

1. Make up two facts (here: tuples 1 and 2) that have the same values in the role(s) below the UC, but different values in the other role(s).
2. Ask the domain expert whether these two concrete sentences can occur together (at the same time) in the population.
   If yes, then the tested UC does not exist.
   If no, then the tested UC does exist.

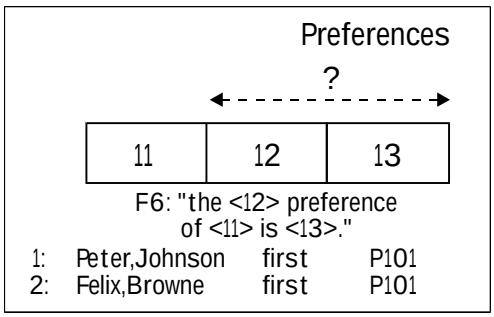


Figure 3.8: determination of a UC on role(s)

Algorithm for the determination of all the uniqueness constraints within a fact type.

1. Unary fact type: verify that the object type is indeed identified by the label type that plays the role (this rule does not apply to subtypes, see chapter 6).
2. Fact type with n roles (n stands for a number greater than 1): there are n possibilities for a UC on n-1 roles. Test all these possible UCs whether they exist.
   a No UC on n-1 roles is found. Then there is exactly one UC on all the n roles. Draw this UC in the diagram.
   b At least one UC on n-1 roles is found (at most, n UCs on n-1 roles can be found). Draw all UCs found in the diagram.
3. In fact types with more than 2 roles: for every UC on n-1 roles found in step 2: check whether a smaller UC on n-2 roles exists that completely falls within the UC that was found. There are n-1 possibilities for a UC on n-2 roles per UC found in step 2. Test the existence of all the possible UCs.
   a No UC on n-2 roles is found. Then the UC on n-1 roles is correct.
   b At least one UC on n-2 roles is found. Then: keep looking until the smallest UC is found and split the fact type (see section 3.3.1.1).

A practical aid for the analyst for this procedure is the table in figure 3.9, with examples of the procedure given for four fact types. An explanation of the table is given below it.

Purpose	Test UC on Role(s):	Tu-ple No.	Fact Type Fact Type Expression Tuples	Can Tuples Occur Together?	Ans-wer Y/N	Conclusion
			Supervision F4: "<7> is supervised by <8>."			
		1:	P101 BLC			
Testing the 2 possible UCs on 1 role.	7 8	2: 3:	P101 GPB P333 BLC	1 + 2 ? 1 + 3 ?	N Y	UC 5 on role 7 no UC
			Project Description F5: "<9> concerns <10>."			
		1:	P333 carrying out an inf. anal.			
Testing the 2 possible UCs on 1 role.	9 10	2: 3:	P333 improving performance P444 carrying out an inf. anal.	1 + 2 ? 1 + 3 ?	N Y	UC 6 on role 9 no UC
			Student F1: "there is a student <1> <2>."			
		1:	Peter Johnson			
Testing the 2 possible UCs on 1 role.	1 2	2: 3:	Peter Davis Dorothy Johnson	1 + 2 ? 1 + 3 ?	Y Y	no UC no UC So: UC 1 on 1+2
			Preferences F6: "the <12> preference of <11> is <13>."			
		1:	first Peter,Johnson P101			
Testing the 3 possible UCs on 2 roles.	11 + 12 11 + 13 12 + 13	2: 3: 4:	first Peter,Johnson P203 second Peter,Johnson P101 first Felix,Browne P101	1 + 2 ? 1 + 3 ? 1 + 4 ?	N N Y	UC 7 on 11+12 UC 8 on 11+13 no UC
Testing for smaller UCs within UC 7.	11 12	5: 6:	third Peter,Johnson P400 first Robert,Keyes P311	1 + 5 ? 1 + 6 ?	Y Y	no UC no UC
Testing for smaller UCs within UC 8.	11 13	7:	Done already, see tuple 5. second Anna,Northrup P101	1 + 7 ?	Y	no UC

Figure 3.9: table for determining UCs

Explanation of the table in figure 3.9:
An analyst wants to test all UCs on the roles of fact type Supervision (figure 3.9, top). In the central column ‘Fact Type, Fact Type Expression, Tuples’, he/she writes the name of the fact

type with the appropriate fact type expression. The analyst writes a first tuple 'P101 BLC' under the fact type expression. In the column 'Tuple No.', every tuple receives a unique number per fact type. Underneath tuple 1 of every fact type a line is drawn because all other tuples will contain changes (variations) with respect to the values in the first tuple. Fact type Supervision has two roles (n=2). The analyst, following step 2 of the procedure, wants to test the 2 possible UCs on 1 role and writes this in the column 'Purpose'. In each row in the remaining columns, one UC is now tested. First, the analyst tests whether a UC exists on role 7, and therefore writes '7' in the column 'Test UC on Role(s)'. Following the procedure, the analyst constructs a tuple 2, which *compared to tuple 1* has the same value in role 7 and a different value in role 8. He/she asks the domain expert whether tuples 1 and 2 can occur together and therefore writes '1 + 2 ?' in the column 'Can Tuples Occur Together?'. The answer is recorded in the column 'Answer Y/N'. It is 'N', and the conclusion that there is then a UC on role 7 is written in the column 'Conclusion'. Next, it is tested whether there is a UC on role 8. Tuple 3 then receives *compared to tuple 1* the same value in role 8 and a different value in role 7. The analyst asks whether tuples 1 and 3 can occur together (tuple 2 is not considered here). The answer is yes, so there is no UC.

The UC determination for fact type Project Description goes in an analogous way. For fact type Student, no UC is found on only one role, and so there is a UC on both roles.

For fact type Preferences (n=3), more steps are involved. First, the 3 possible UCs on 2 roles are tested, two of which do indeed exist. Next (step 3 of the procedure), it is tested whether a smaller UC exists within the roles of UC 7. This turns out not to be the case, so UC 7 is correct. In the third step, it is tested whether a smaller UC exists within UC 8, which is also not the case. So UC 8 is correct as well.

In the table from figure 3.9, it is invariably asked whether tuple 1 can occur together with another tuple. All other tuples are variations on tuple 1. This is not necessary (two new tuples per tested UC are also possible) and even impossible sometimes, but this approach requires the least amount of work.

3.2.3 Final Remarks

1 We advise to fill in the table from figure 3.9 completely for all UC determinations. In this way, simultaneous with the analysis, good documentation is generated that accounts for all UCs present or absent. Experienced analysts will quickly spot trivial UCs without explicit verification, but they run the risk of drawing wrong conclusions; in addition the justification to the client is missing.

2 We use unique numbers to identify UCs in each IGD. That is not strictly necessary: we could also identify a UC by a list of the roles that the UC covers. A unique number is, however, more convenient: thus we can refer to UCs more easily (see also section 2.9).

3 The analyst does not need to present populated fact types in IGD form to the user, or to show the table from figure 3.9 (but if the user is prepared to learn the notation then there is nothing against this). A practical approach in the interview with the domain expert is to use LTL fact type expressions (see figure 2.19 in section 2.6), which are easily understood by everyone, under which pairs of tuples can be written. Often, however, the number of blanks in the LTL-FTEs is greater than the number of roles in the fact type considered. Fact type Mentorship, for example, has two roles (OTL-FTE F2: "the mentor of <4> is <5>."), but the LTL-FTE has three blanks to be filled in (LTL-FTE F2: "the mentor of student <first name> <surname> is <teacher code>."). The analyst must then treat these three blanks as two roles.

4 Instead of using concrete examples, the analyst could also ask questions on a more abstract level, such as: "Is it possible for a student to have more than one mentor?" We strongly advise against this, however, because it is found in practice that this way of phrasing questions will easily lead to misunderstandings and incorrect answers. A domain expert, for example, could truthfully answer 'yes' to the above question (which would lead to a multiple role uniqueness constraint on fact type Mentorship), whereas he/she would still *register* only one (single role UC), but this was not precisely asked for. The use of concrete examples prevents this kind of misunderstandings.

5 The concrete examples should be chosen with care. If in figure 3.9 a tuple 5 would be introduced for fact type Preferences: 'fourth Peter,Johnson P400', then the user might answer 'no' to the question whether both tuples 1 and 5 can occur together or not, because a fourth preference will not be recorded. This would lead to a wrong conclusion about a UC on role 11. It is a good habit to ask for each negative answer *why* these two facts cannot occur together, and to write down the answer.

6 All uniqueness constraints treated so far concern roles within one fact type (*intra fact type UCs*). There are, however, also UCs on roles that belong to different fact types (*inter fact type UCs*). The notation for such a UC is the letter 'u' (from unique) in a small circle that is connected to all the involved roles by lines. As an example, figure 3.10 contains a part of the IGD from figure 2.28 in section 2.9, in which object type Student has a student no. as primary identifier. UCs 1, 11 and 12 are determined in the usual way. Now suppose that the combination of first name and surname is also still identifying (i.e.: unique) for Student. This is expressed by UC 13. (Formulated in terms of the Relational Model, UC 13 means: after carrying out the natural join over roles14 and 16 from fact types Student First Name and Student Surname, the combination of values under roles 15 and 17 is different for each tuple).

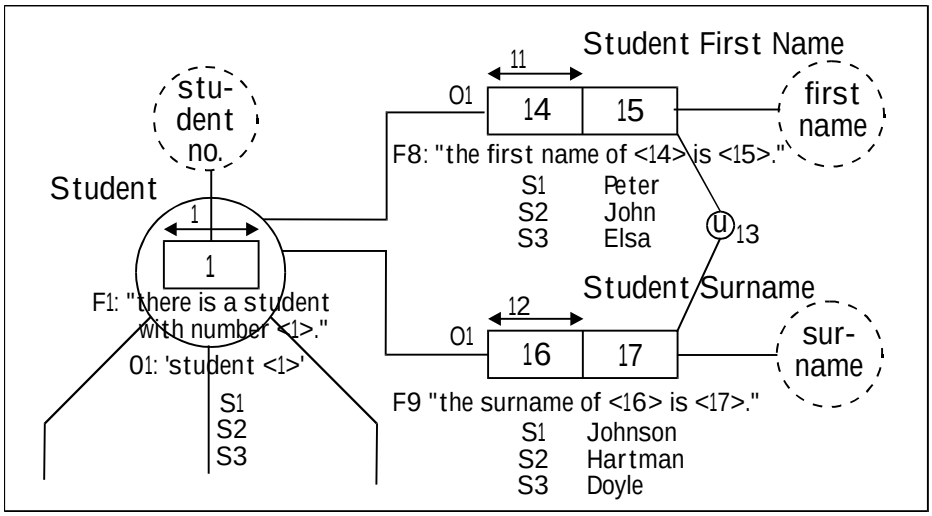


Figure 3.10: inter fact type UC

There is no systematic method for the determination of inter fact type UCs. They usually appear by chance in the starting document or in interviews.

3.3 Tests after Determining Uniqueness Constraints

After the uniqueness constraints have been determined, two kinds of tests must be carried out. There are elementarity tests (section 3.3.1), with which any non-elementary fact type expressions can be found: the *n-1 rule test*, the *n rule test* and the *projection/join test*. The *nominalization test* (section 3.3.2) is used to look for object types that have not yet been recognized, which play roles in different fact types.

3.3.1 Elementarity Tests

In this section, we formulate two well-formedness rules for IGDs, which can also be used to check whether the fact type expressions are indeed elementary: the n-1 rule (section 3.3.1.1) and the n rule (section 3.3.1.2). We also give an example of the projection/join test (section 3.3.1.3), which covers all cases that cannot be found with the two other tests mentioned above.

3.3.1.1 The n-1 Rule Test

In section 2.3, we discussed that the information analyst asks a domain expert to verbalize the concrete example documents. This should preferably be done in elementary fact expressions,

which means that sentences should be chosen that cannot be split into two or more smaller sentences without loss of information. This usually works well, but sometimes non-elementary fact expressions are expressed nevertheless. Let us suppose that we have not noticed the splittability of the expressions below:

"Project 101 concerns developing a time keeping system and is supervised by BLC."

"Project 203 concerns writing course material on FCO-IM and is supervised by BAK."

That would lead to a ternary fact type Project Data, shown in figure 3.11, instead of the two binary fact types Supervision and Project Description. The fact type expression was given the code F45 to show that it is made up of F4 and F5.

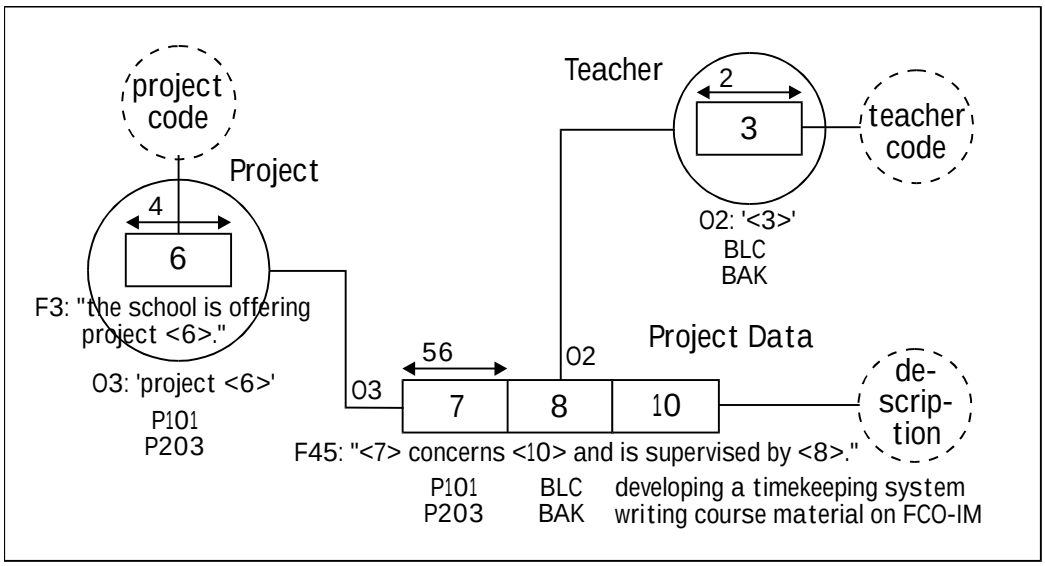


Figure 3.11: non-elementary fact type

Figure 3.12 summarizes the procedure of the determination of the uniqueness constraints. First UC 5 on roles 7 and 8 and UC 6 on roles 7 and 10 are found. By checking whether smaller UCs exist it turns out that there is indeed a single role UC on role 7, which replaces both UCs 5 and 6 because it is stronger than both the others (the number 56 emphasizes this).

A single role uniqueness constraint on a ternary fact type is a sign for the information analyst that the corresponding fact expressions are splittable. The way the splitting should be done is obvious here because of the word 'and', and after the domain expert has rephrased the sentences we regain the binary fact types Supervision and Project Description, with their separate fact type expressions F4 and F5 (see figure 3.3). The splitting of a ternary fact type usually is not a difficult task, but especially where fact types with more than three roles are concerned, it is seldom clear how the splitting should be done. The analyst can offer some suggestions of course, but in the end it is the domain experts who must rephrase the facts or who must validate the separate new sentences. After splitting, we must redetermine the UCs on the new fact types. Some of the UCs on the non-elementary fact type may lead to inter fact type UCs between the new fact types.

Purpose	Test UC on Role(s):	Tu-ple No.	Fact Type Fact Type Expression Tuples			Can Tuples Occur Together?	Ans-wer Y/N	Conclusion
			Project Data F45: "<7> concerns <10> and is supervised by <8>."					
		1:	P101	dev. timek. syst.	BLC			
Testing the 3 possible UCs on 2 roles.	7 + 8 7 + 10 8 + 10	2: 3: 4:	P101 P101 P222	information mining dev. timek. syst. dev. timek. syst.	BLC LEK BLC	1 + 2 ? 1 + 3 ? 1 + 4 ?	N N Y	UC 5 on 7 + 8 UC 6 on 7+ 10 no UC
Testing for smaller UCs within UC 5.	7 8	5: 6:	P101 P321	re-engineering debugging	VRM BLC	1 + 5 ? 1 + 6 ?	N Y	UC 56 on role 7 no UC
Testing for smaller UCs within UC 6.	7 10		Done already, see tuple 5.					
		7:	P444	dev. timek. syst.	GPB	1 + 7?	Y	no UC

Figure 3.12: UC determination in a non-elementary fact type

The above criterion, formulated more generally for n-ary fact types (n greater than 2), is called the *n-1 rule*:

> If a fact type with n roles has at least one uniqueness constraint on less than n-1 roles, then the fact type is splittable.

Note: the reverse of this rule is not valid. The absence of a UC on less than n-1 roles does not guarantee that a fact type is elementary. To be absolutely sure, the projection/join test from section 3.3.1.3 must be used. But it is useful that most fact types that can be split can be spotted at a glance.

The n-1 rule-test simply consists of the check that there are no UCs that are too small.

3.3.1.2 The n Rule Test

We consider uniqueness constraints on *nominalized* fact types here. A nominalized fact type models an object type in the UoD. From the communication viewpoint it is required that all the objects from a certain object type can be uniquely identified, otherwise conversation partners will not know exactly which object they are talking about. That is why the fact type inside an object type represents the identifier for the objects of this object type: we can after all furnish it with an existence postulating fact type expression (a pre-eminently identifying FTE). In section

3.2.1, we already remarked that UCs on nominalized fact types reflect identification. We can even reverse this relationship between identifiers and UCs: in a nominalized fact type there cannot be a role that does not fall under a UC, because such a role is clearly not necessary to identify objects from the object type (otherwise the UC would have stood over that role as well). This is the reason for the existence of the following well-formedness rule, which is known as the n rule for nominalized fact types:

> Every nominalized fact type with n roles only has exactly one uniqueness constraint over all n roles

The n rule test simply consists of the check whether all nominalized fact types comply with the n rule (if not, then all the fact types with a role played by such an object type are splittable because there is redundancy in the population of all these roles).

The n-rule is valid for everything discussed in chapters 2 and 3, namely elementary IGDs without specialization or generalization. In chapter 4 we will derive a relational schema from elementary IGDs. In this process, non-elementary fact types will arise, to which the n rule then does not apply. In chapter 6, we will discuss specialization and generalization, which will require us to refine the n rule.

3.3.1.3 The Projection/Join Test

The n-1 rule test suffices for most situations to detect splittable fact types. (Formulated in terms of the Relational Model: at least for ternary fact types, the procedure for determining UCs guarantees Boyce-Codd normal form, but not fourth normal form.) Certainty about the elementarity of an n-ary fact type (n greater than 2) that has passed the n-1 rule test can only be obtained by the projection/join test, but in practice it is used only when in doubt.

We will only discuss this test briefly. *Projection* and *join* are terms from relational algebra, a subject outside the scope of this book, assumed to be familiar to the reader. Any book on the Relational Model can be used as reference. A more detailed discussion of this test can be found in the book ‘Conceptual Schema and Relational Database Design’ by G.M. Nijssen and T.A. Halpin, Prentice Hall, 1989, and in its second edition (T.A. Halpin, 1995) (literature list 3).

A prerequisite for this test is that you have a *significant population* at your disposal that is a population that illustrates all the valid possible combinations of values in its tuples. There is, however, no criterion to establish whether a population is significant or not. This methodological problem is one of the reasons that this test is only carried out when serious doubts exist. In any case do not take a population that is too small, and together with the domain expert, try to find at least one tuple of every allowed kind. A tuple too many is not a problem, but a tuple too few can give an incorrect result.

We start with a fact type with n roles (n greater than 2) that has passed the n-1 rule test. There are two versions of the test. We give an algorithm for both versions.

Version 1: determines whether a fact type is splittable or not, but not how it is to be split.

1 Provide the fact type with a significant population.
2 Make all n possible projections on fact types with n-1 roles (treat a fact type as a table and a role as a column).
3 Join all the n fact types that were created in step 2 together again with natural joins, as follows: take two fact types and join them; next join a third fact type with the result of the first two, and continue in this way until all n-1 joins have been made. The result is a single fact type with n roles again.
4 If the population of the result of step 3 is the same as the population of the original fact type, then the original fact type is splittable.
   If there are extra tuples in the population of the result of step 3, which are not there in the population of the original fact type, then the original fact type cannot be split.

Version 2: determines whether a fact type can be split in one particular way, or not.

1 Provide the fact type with a significant population.
2 Split the fact type by making k projections of it (k stands for a number greater than 1), in such a way that each role appears in at least one projection.
3 Join all the k-fact types that were created in step 2 together again with natural joins, as follows: take two fact types and join them; next join a third fact type with the result of the first two, and continue in this way until all k-1 joins have been made. The result is a single fact type with n roles again.
4 If the population of the result of step 3 is the same as the population of the original fact type, then the original fact type can indeed be split in the chosen way.
   If there are extra tuples in the population of the result of step 3, that are not there in the population of the original fact type, then the original fact type cannot be split in the chosen way.
5 Carry out steps 2, 3 and 4 for all possible splittings of the original fact type, until a suitable splitting is found.

In figure 3.13, a short illustration of both versions is given.

Figure 3.13a contains a fact type with three roles E, D and P. For brevity, we have only drawn the most essential parts of the fact type. Role E is played by the object type Employee (not shown), role D by Department and role P by Project.
A corresponding fact expression could be for example: “Employee E1 works for department D1 on project P1.”. Let EDP be the name of the fact type. EDP is provided with a small, but significant population. It is clear from this population that there are no UCs on two roles, so there is a UC on all three roles. EDP therefore passes the n-1 rule test.

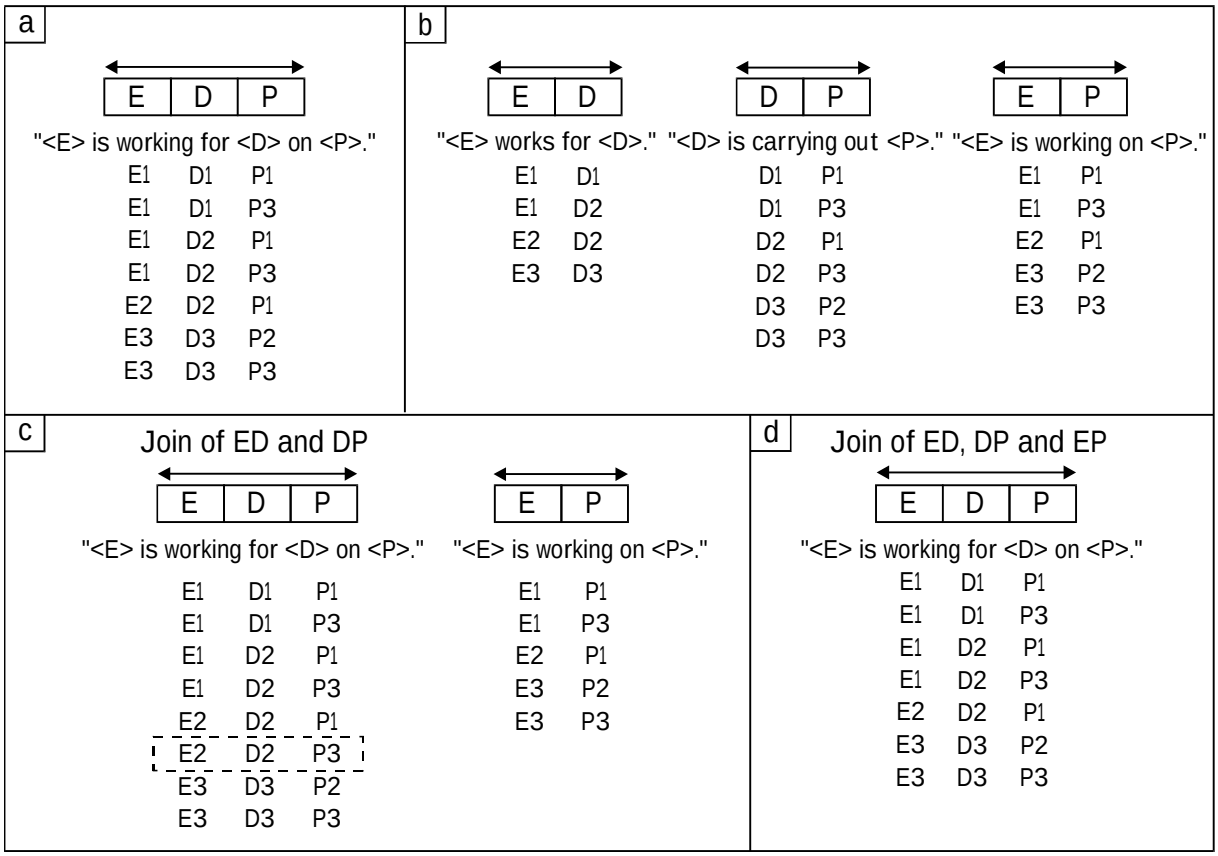


Figure 3.13: illustration of projection/join test

An analyst doubts about whether EDP can be split or not. Is it really the case, that there is absolutely no connection between the departments for which an employee works and the projects that he/she is involved in? If so, then EDP indeed cannot be split. Or is it for example the case that employees are involved in all the projects that are carried out by the department for which they work? In that case, EDP would be splittable, namely in a fact type ED and a fact type DP with corresponding fact expressions such as “Employee E1 works for department D1.” and “Department D1 is carrying out project P1.”. There are three other splitting possibilities as well: in ED + EP, in EP + DP and in ED + DP + EP.

The analyst decides to test the splittability of EDP with version 1 of the projection/join test. Step 1: figure 3.13a. Step 2: the three possible projections of EDP to fact types with two roles appear in figure 3.13b (duplicate tuples under ED, DP or EP have been removed). Step 3: Figure 3.13c shows an intermediate stage after the natural join of ED and DP, with EP as yet unchanged next to it. So far, one extra tuple has arisen (highlighted in the dotted frame), but we are not yet finished. After the join of both fact types in figure 3.13c, the end result is shown in figure 3.13d. Step 4: the population in 3.13d is identical to the original in 3.13a, so EDP is indeed splittable.

How is the fact type to be split? Let us try the splitting in ED + DP with version 2. Step 1: 3.13a. Step 2: 3.13b, only fact types ED and DP. Step 3: 3.13c, only the join of ED and DP. Step 4: the population has acquired an extra tuple, so the fact type cannot be split in ED + DP. Step 5: in an attempt to split EDP in EP + DP, four extra tuples appear (the readers can check this for themselves). In an attempt to split EDP in ED + EP, however, no extra tuples appear after the join (result as in 3.13a), so the fact type can indeed be split thus. (If this last attempt had also resulted in extra tuples, EDP would have been in fourth normal form but not in fifth normal form and would have to be split in ED + EP + DP).

Evidently, in this UoD employees work for departments, and independently from this, they work on projects in a kind of matrix organization. The domain experts confirm that for each project, a project team will be formed out of people from different departments. The analyst could also have established the splittability of fact type EDP in dialogues with domain experts in which he/she would suggest splittings of the example fact expressions, and use the answers obtained for further questions about the organization structure.

It is clear from these examples, that this test is rather tedious to complete. It is also independent of the original fact expressions. In step 5 of version 2, many possibilities must be worked out (for a ternary fact type: 4; the number increases quickly for fact types with more than 3 roles), most of which will turn out to be nonsensical if the original fact expressions are taken into consideration.

A sensible procedure then seems to be: use version 1 to establish whether a fact type is splittable, and if so, consult domain experts about the question how it must be split. The proposed splitting can then be checked with version two. Sometimes, there are various splitting possibilities.

3.3.2 The Nominalization Test

In this section, we assume that in the example student-project case study, facts about the participation of students in workshops are also of importance, in addition to the familiar project information. Students take part in various workshops during their studies. Generally, several students work together in the same workshop. For each workshop participation, students are coached and assessed by a certain teacher. Verbalization of example documents (not shown) by a domain expert yields, among other things, the following fact expressions:
FE 1: “The participation of student Peter Johnson in workshop W6 was coached by OVL.”
FE 2: “The participation of student John Hartman in workshop W6 was coached by KLP.”
FE 3: “The participation of student Peter Johnson in workshop W4 was coached by KLP.”
FE 4: “Student Peter Johnson got a grade S for workshop W6.”
FE 5: “Student John Hartman got a grade G for workshop W6.”

Figure 3.14 shows an analysis of these fact expressions: there are two fact types Coaching and Assessment, with three roles each. Figure 3.15 shows the relevant part of the corresponding IGD, which is an extension of the IGD in figure 3.3. The rest of the IGD from figure 3.3 is not shown: the three loose lines down from object type Student indicate the presence of fact types Mentorship, Preferences and Allocation without actually drawing them. Role 22 is played by object type Workshop. The connecting line is drawn broken, as if it passes under fact type Coaching, to avoid an unattractive layout. The uniqueness constraints were determined in the standard way.

The gray shading in figure 3.15 accentuates the following pattern: roles 16 and 17 from fact type Coaching are played by Student and Workshop respectively, and they fall under a UC (UC 12). Likewise, roles 21 and 22 from fact type Assessment are played by Student and Workshop respectively, and fall under a UC (UC 13).

Whenever such a pattern arises, it is very probable that the analyst has made the mistake to fail to recognize an object type. The combination student + project is unique in both fact types Coaching and Assessment according to UC 12 and UC 13, so perhaps this combination identifies an object from a certain object type. Here, an object type Participation appears to be obvious: fact expressions FE 1, FE 2 and FE 3 already mention a participation. In the manner discussed in section 2.7.2, fact type Coaching can also be analyzed with an extra object type Participation, see figure 3.16. An extra nominalized fact type then arises in the IGD, see figure 3.17. So we used a nominalization - denominalization transformation here (see section 2.7.4).

If we would apply the same transformation to fact type Assessment, then the nominalized fact type Participation would appear there as well. Fact expressions FE 4 and FE 5, however, do not lend themselves for this easily. Therefore, the analyst asks the domain expert if he can replace fact expressions FE 4 and FE 5 with FE 4' and FE 5':

FE 4': "The participation of student Peter Johnson in workshop W6 was assessed with a grade S."

FE 5': "The participation of student John Hartman in workshop W6 was assessed with a grade G."

Coaching:

"The participation of student Peter Johnson in workshop W6 was coached by OVL."

Student:O1 Workshop:O4 Teacher:O2

'student Peter Johnson' 'workshop W6' 'OVL'

first name surname workshop code teacher code

F8: "the participation of <Student:O1> in <Workshop:O4> was coached by <Teacher:O2>."
O1: 'student <first name> <surname>'
O4: 'workshop <workshop code>'
O2: '<teacher code>'

Assessment:

"Student Peter Johnson got a grade S for workshop W6."

Student:O1 grade Workshop:O4

F9: "<Student:O1> got a grade <grade> for <Workshop:O4>."

Figure 3.14: analysis without object type Participation

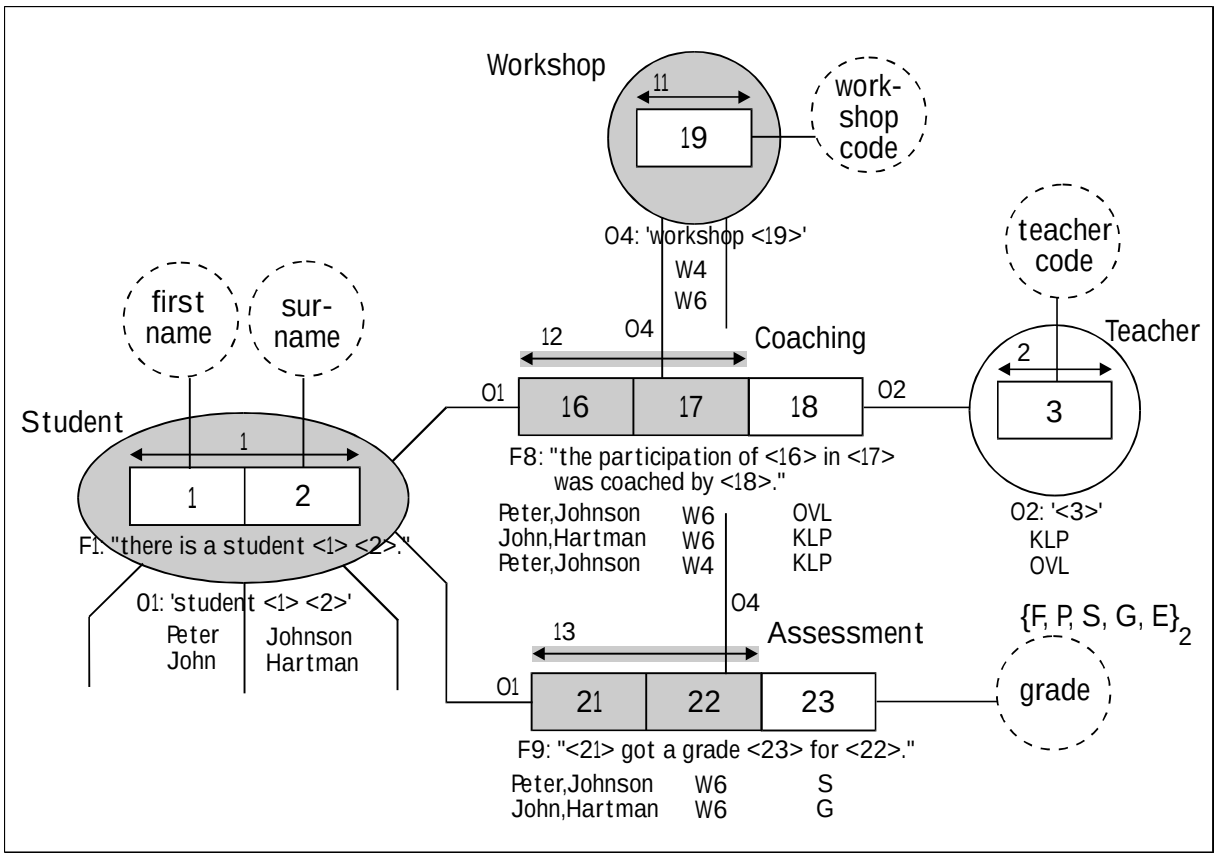


Figure 3.15: IGD with nominalization pattern

Coaching:

"The participation of student Peter Johnson in workshop W6 was coached by OVL."

Participation:O6 Teacher:O2

'the participation of student Peter Johnson in workshop W6'

Student:O1 Workshop:O4

F8: "<Participation:O6> was coached by <Teacher:O2>."
O6: 'the participation of <Student:O1> in <Workshop:O4>'

Assessment:

"The participation of student Peter Johnson in workshop W6 was assessed with a grade S."

Participation:O6 grade

F9: "<Participation:O6> was assessed with a grade <grade>."

Figure 3.16: analysis with object type Participation

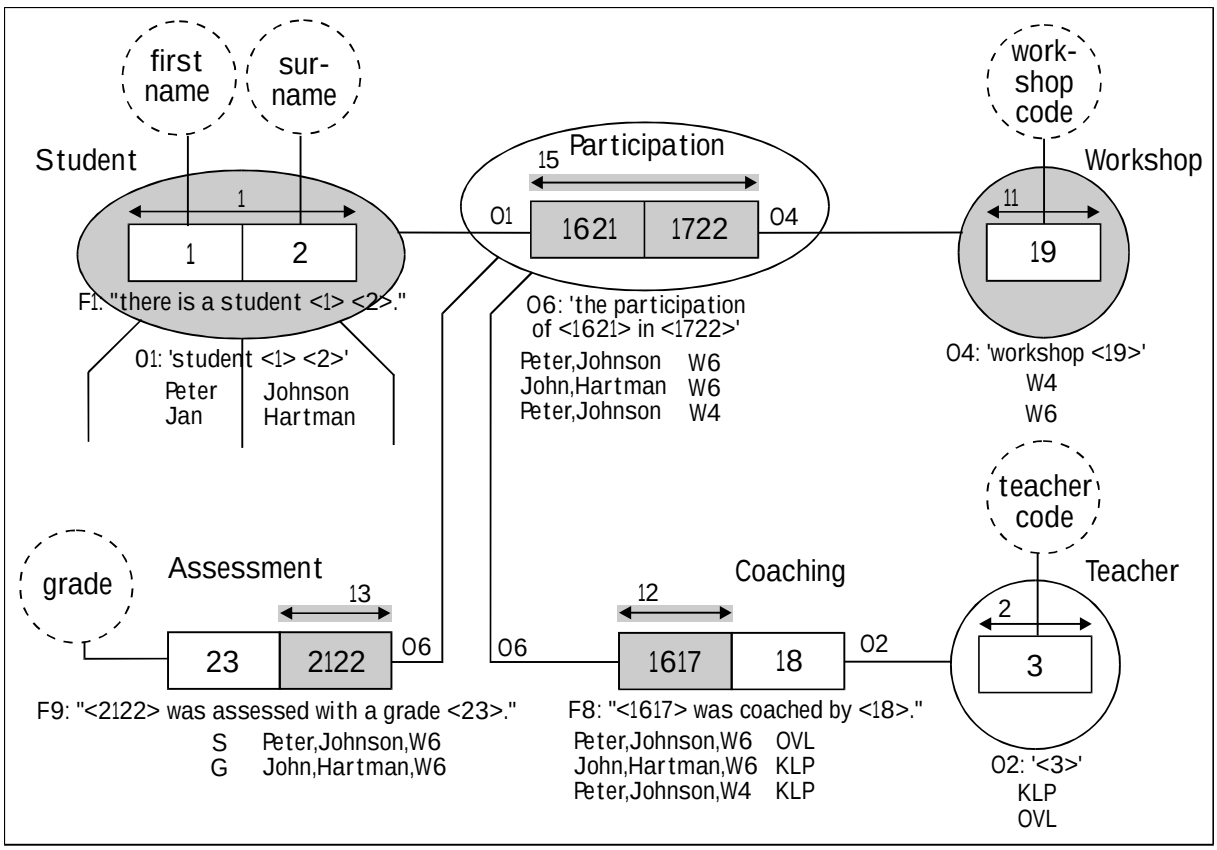


Figure 3.17: IGD after carrying out nominalization

The domain expert declares that sentences FE 4' and FE 5' indeed mean the same as sentences FE 4 and FE 5. The analysis appears in figure 3.16 and the resulting IGD in figure 3.17, in which the role numbers have been chosen so that they emphasize the correspondence with figure 3.15: Role 1621 in figure 3.17 is played by Student whereas in 3.15 roles 16 and 21 are played by Student. Role 1617 in Coaching in figure 3.17 replaces roles 16 and 17 in Coaching in figure 3.15. (In practice, and in the FCO-IM tool, usually new consecutive numbers are used.)

This result can also be attained from another approach: a participation is the contribution of a certain student to a certain workshop. That is in itself a useful object in the UoD. For this *same* participation (or: for the *same* object) the coach and the grade are to be recorded in fact types Coaching and Assessment. An object expression for the *same* object therefore appears in different fact expressions. In fact expressions FE 1 and FE 4', for example, this can be clearly seen, because the object expression is the same ('the participation of student Peter Johnson in workshop W6'). So, an object type expression for the same object type occurs in different fact type expressions. See O6 in figure 3.16. It then follows from the requirement to model in a redundancy free way, that this object type-expression will be modeled once only together with the corresponding object type. It also follows from this line of reasoning, that it is mandatory in FCO-IM to perform the nominalization - denominalization transformation in the nominalization direction, as soon as it is found that the same object type arises in more than one fact type.

On the basis of the above reasoning, the following *nominalization rule* applies in FCO-IM:

If the following pattern occurs in two or more fact types..
- in all the concerned fact types, a combination of 2 or more roles occurs which is played by the same combination of object types (lexical or non-lexical) in all these fact types
- in all the concerned fact types, this combination of roles falls entirely under at least one uniqueness constraint (which can extend over more roles)

..then the analyst must check whether this combination of roles identifies the same meaningful (in the eyes of the domain expert) object type in all the concerned fact types.
If so, then this object type must be added (in the form of a nominalized fact type), and all the concerned fact types must be remodeled so that the combination of roles is replaced by a single role, played by the new object type.

The nominalization test consists of checking all fact types with a multiple role uniqueness constraint whether the above pattern occurs. If so, then the appointed nominalization must be carried out.

We conclude this section with a few remarks.

1 The analyst was only prompted to examine whether a nominalization had been overlooked by the occurrence of the pattern from the nominalization rule (shaded in figure 3.15). From fact expressions FE 1 and FE 4 for example (see figure 3.14), it was not directly clear that the sentences concerned the same object from object type Participation.

2 In practice, it is almost always necessary to apply the nominalization rule. We therefore advise to always thoroughly carry out the nominalization test.

3 If a nominalization is overlooked nonetheless, then the result after deriving a relational schema will mostly be a greater number of tables than necessary (see chapters 4 and 5).

4 The requirement to verify with the domain expert that it is really the same objects which appear in all the concerned fact types is essential. Sometimes the pattern does occur, but it turns out that no meaningful object type can be associated with it, or it turns out that the pattern concerns different objects types in the different fact types. In these cases the nominalization should not be carried out.

5 It is indeed necessary to rephrase sentences FE 4 and FE 5, because the words that identify a participation are not connected: in FE 4 for example, the parts ‘student Peter Johnson’ and ‘workshop W6’ are separated by other words in the sentence. Presently, FCO-IM still requires that an object (type) expression is a single connected part of a fact (type) expression. If it were possible for an object (type) expression to consist of several separated sentence parts, then we could have left FE 4 and FE 5 as they are and simply have added a second OTE to Participation. This is something for the future, however.

6 Suppose we would have had the following FE 4" instead of FE 4:

FE 4": “For workshop W6, student Peter Johnson got a grade S.”.

Then we could have classified the part ‘workshop W6, student Peter Johnson’ as an object expression leading in figure 3.17 to a second OTE O7: ‘<1621>, <1722>’ for Participation without any problem, even though the resulting FTE for Assessment looks awkward: F9": “For <2122>, got a grade <23>.”. In general, we recommend to rephrase all FTEs so the same OTE for the new OT appears in all the FTEs, to avoid ungraceful FTEs and OTEs (for instance: OTEs containing verbs are almost always in poor style).

7 Long chains of nominalizations can arise, particularly where hierarchical structures are involved. See the concise example in figure 3.18, in which the object types that play the roles were not drawn but indicated by a letter in the role (all roles marked A are played by object type A).

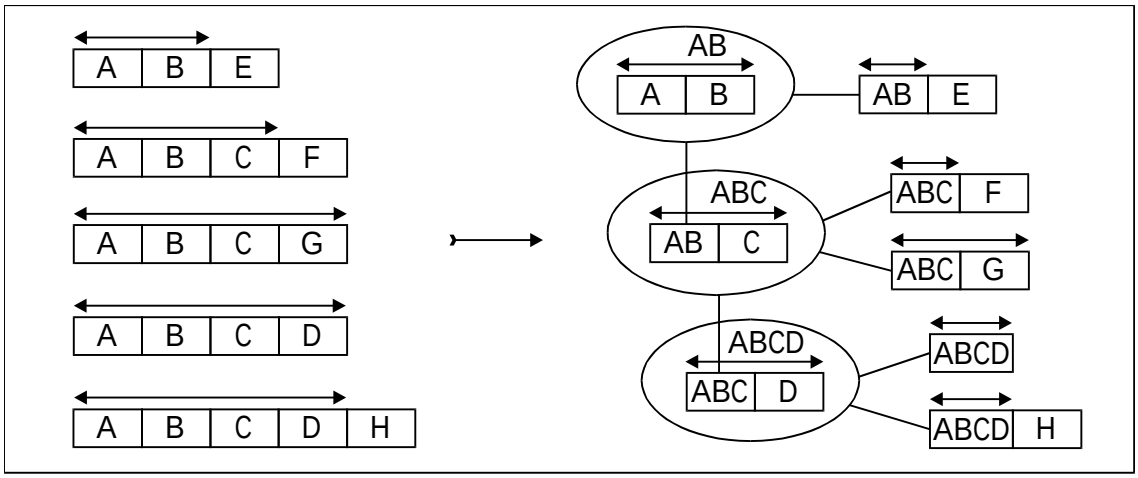


Figure 3.18: complex application of nominalization rule

There are three patterns here: A + B is in all fact types, A + B + C in four of the five and A + B + C + D in two. The easiest way to carry out the nominalizations is to start with the pattern with the smallest number of roles (A + B), to take the second smallest pattem next and to go on like this until all the patterns have been dealt with.

The uniqueness constraint on fact type ABCG (figure 3.18, left) also extends over role G, in addition to roles A, B and C, which satisfy the nominalization pattern. That is why the binary fact type with roles ABC and G (figure 3.18, right) gets a multiple role UC after nominalization).

Fact types ABCD and ABCDH (figure 3.18, left) show the nominalization pattern in the four roles A + B + C + D. After nominalization of the ABCD part of fact types ABCD and ABCDH, there are no more roles left from fact type ABCD, so the result is a unary fact type played by object type ABCD (see also section 5.2). The remaining role H from ABCDH ends up in a binary fact type.

3.4 Totality Constraints

In the starting document we underlined a part of the sentence:

> They can enter their choices on another list, on which I have already filled in their name and their mentor (...).

This suggests that for every student the mentor must be known. The analyst, therefore, asks the project coordinator: "Must the mentor be known for every student, or could it be, for example, that the mentor of student Peter Johnson has not been filled in?". The School's Project Coordinator answers that the mentor must always be known. This means that the whole population under roles 1 and 2 from fact type Student must also appear under role 4 from fact type Mentorship (see figure 3.19). Such a requirement that the whole population of a nominalized fact type must also appear under roles played by this nominalized fact type is called a *totality constraint*.

A totality constraint (TC) is a constraint which can only be imposed on roles that are all played by the same nominalized fact type, and which states that *every* tuple from the population of this nominalized fact type must also appear at least once under one or more of the concerned roles. In other words: the combination of the populations of all the concerned roles must be the same as the population under the nominalized fact type (leaving duplicate occurrences aside). It will be shown that a totality constraint can preclude situation 4 on page 3.1.

A totality constraint can apply to the population of one or more roles. In an IGD, a totality constraint is graphically depicted as a big polka dot (•). See figure 3.19, to which all the TCs of the example student-project case study have been added. We will discuss them below. If a totality constraint concerns only one role, then we place the dot on the line connecting this role with the object type that plays it, at the side of the object type: see TCs 1, 3 and 4 in figure 3.19. If a totality constraint concerns several roles, then we place the dot in a little circle, which is attached to the lines connecting these roles with the object type: see TC 2 in figure 3.19. A TC on only one role is called in *single role totality constraint*. A TC on more than one role is called a *multiple role totality constraint*.

We will now consider all the TCs in the example student-project case study, see figure 3.19.

From the starting document and from the interview cited above, it was already established that a TC applies to role 4: TC 1. With that, situation 4 from page 3.1 is precluded: if a fact about the existence of a student is added to fact type Student, then TC 1 forces us to add a fact about the mentorship of this student in fact type Mentorship as well. Would there be still other totality constraints on the roles played by Student?

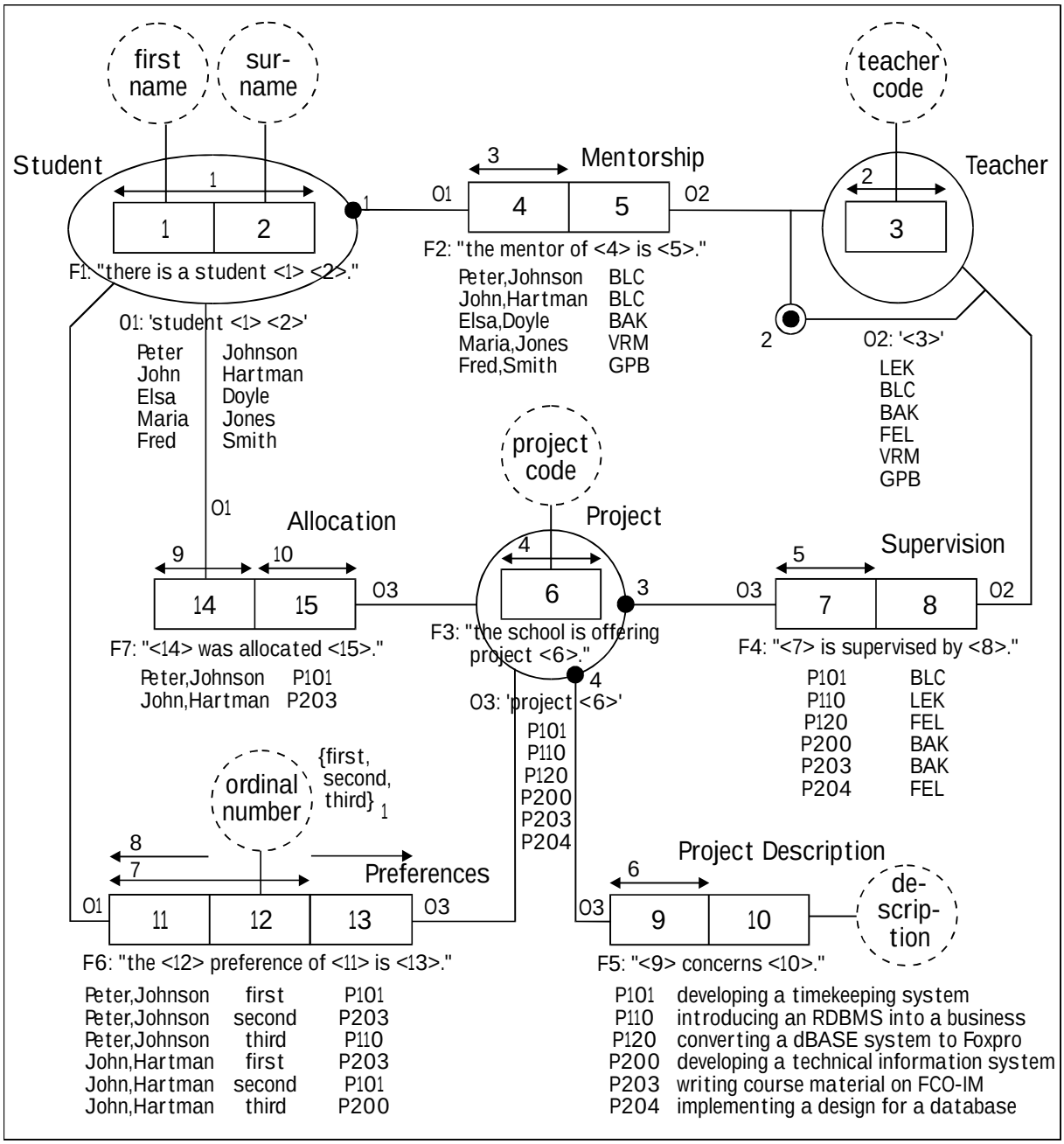


Figure 3.19: IGD with totality constraints

We also underlined in the starting document:

> Students who have not given their preferences in time will not be assigned to a project at this stage;

This suggests that no totality constraint applies to role 11. The analyst asks: “Could it at any moment occur that no project preference for a certain student is recorded?”. One look at the example document in figure 2.4 suffices to see that the answer is ‘yes’: at the moment the student data are recorded, no project preferences are known yet, and moreover some students fail to give their project preferences on time. So there is no TC on role 11: the population of role 11 does not need to be the same as that of roles 1 and 2, but can be just a subset.

Next, consider this underlined part from the starting document:

> Students who have not given their preferences in time will not be assigned to a project at this stage;

This suggests that no TC applies to role 14 either, which the School's Project Coordinator confirms. According to the same sentence from the starting document there also cannot be a TC on the combination of roles 11 and 14, because students who still have not given their preferences (i.e.: who do not occur under role 11) also do not yet get assigned to a project (i.e.: do not occur under role 14).

Since a totality constraint can only concern roles that are all played by the same nominalized fact type, we consider all the roles played by a certain object type when determining TCs. In doing so, we will systematically look at each role and combination of roles in turn. This is what was done above for object type Student. A practical aid in this procedure is a table such as is shown in figure 3.20, in which it is worked out at the concrete level which population patterns are allowed for tuple 'Peter Johnson' from object type Student in the fact types that Student plays a role in.

In such a table, we create one column for the nominalized fact type, and one column for each role played by this object type. In the first column, we enter concrete examples of objects from this object type. In a column for one of the roles played by the object type however, we enter values from all the *other* roles from the fact type that the role being played is in, and *no* value from the role that the column is for.

Object Type: Student	Role 4 from Mentorship	Role 11 from Preferences	Role 14 from Allocation	OK?	Conclusion
Peter Johnson	BLC	first,P101	P101	Y	Method Applicable
Peter Johnson	-	first,P101	P101	N	TC 1 on role 4
Peter Johnson	BLC	-	P101	Y	no TC on 11
Peter Johnson	BLC	first,P101	-	Y	no TC on 14
Peter Johnson	BLC	-	-	Y	no TC on 11+14

Figure 3.20: table for determining TCs for Student

The procedure for determining TCs is as follows:

1 In the first row in each column we put a value and ask the domain expert whether all these facts can be filled in. If the answer is 'no', then other constraints are involved: subset constraints (see section 3.5), exclusion constraints (see section 3.6) or subtypes (see chapter 6.). We will not discuss that situation further here, see remark 3 at the end of this section. If the answer is 'yes', then we systematically determine all TCs, by adding new rows in which we leave out one or more values, which we indicate with a hyphen. The domain expert must indicate whether such a row is permitted in its entirety.

If not, a TC applies to the combination of roles for which there are hyphens in the table. If the row is permitted, then no TC applies to this role combination. In this way, simultaneous with the analysis, a validated account of the TC modeling is generated, just like it was in determining the unicity constraints.

2 First, we consider all the possible single role TCs in turn. Next, we investigate all the combinations of two roles, for which we only need to consider the roles to which no single role TC applies. As was the case with uniqueness constraints, the following holds for totality constraints as well: should there be two TCs p and q, in which all the roles from p also occur in q, then p is stronger than q, and q must be dropped. Next, we consider all combinations of three roles that do not include any TC that was found earlier in its entirety, and we carry on like this until we cannot go any further. In contrast with the UC determination, in the TC determination it is usually less work to find the strongest TCs first.

Figure 3.21 contains an illustration of this method for the TC determination for object type Project, to which the interview below with the project coordinator belongs:

Object Type: Project	Role 7 from Supervision	Role 9 from Project Description	Role 13 from Preferences	Role 15 from Allocation	OK?	Conclusion
P101	BLC	developing...	Peter,Johnson,first	Peter,Johnson	Y	Method Applicable
P101	-	developing...	Peter,Johnson,first	Peter,Johnson	N	TC 3 on role 7
P101	BLC	-	Peter,Johnson,first	Peter,Johnson	N	TC 4 on role 9
P101	BLC	developing...	-	Peter,Johnson	Y	no TC on 13
P101	BLC	developing...	Peter,Johnson,first	-	Y	no TC on 15
P101	BLC	developing...	-	-	Y	no TC on 13+15

Figure 3.21: table for determining TCs for Project

TEST: Can everything be filled in?
Analyst: Can all this be filled in for one student?
Project Coordinator: Yes.
Conclusion: No other constraints.
TEST: TC on role 7?
Analyst: Must the supervisor always be known for each project?
Project Coordinator: Yes.
Conclusion: TC 3 on role 7.
TEST: TC on role 9?
Analyst: Must each project always have a project description?
Project Coordinator: Yes.
Conclusion: TC 4 on role 9.

TEST: TC on role 13?
Analyst: At the moment that a project becomes available you probably still do not know the student preferences. Is that correct?
Project Coordinator: Yes it is, students decide their preferences gradually, and then there are projects that appear to be very unpopular and are not chosen by anyone.
Conclusion: No TC on role 13.
TEST: TC on role 15?
Analyst: Then of course you do not know the allocation yet when you record the availability of a project?
Project Coordinator: No. Besides, sometimes there are more projects than students, so that some projects do not become allocated at all.
Conclusion: No TC on role 15.
TEST: TC on the combination of roles 13 + 15?
Analyst: So both preferences and allocations can be left out?
Project Coordinator: Yes.
Conclusion: No TC on roles 13 + 15.

The method applied to object type Teacher is shown in figure 3.22.

Object Type: Teacher	Role 5 from Mentorship	Role 8 from Supervision	OK?	Conclusion
BLC	Peter,Johnson	P101	Y	Method Applicable
BLC	-	P101	Y	no TC on 5
BLC	Peter,Johnson	-	Y	no TC on 8
BLC	-	-	N	TC 2 on roles 5+8

Figure 3.22: table for determining TCs for Teacher

TEST: Can everything be filled in?
Analyst: Can a teacher be a mentor as well as a supervisor?
Project Coordinator: Of course.
Conclusion: No other constraints.
TEST: TC on role 5?
Analyst: There probably are teachers who are not the mentor of any student but who do supervise a project. Is this true?
Project Coordinator: Yes.
Conclusion: No TC on role 5.
TEST: TC on role 8?
Analyst: And teachers who are not supervisors but are mentors?
Project Coordinator: Yes that's possible as well of course.
Conclusion: No TC on role 8.

TEST: TC on the combination of roles 5 + 8?

Analyst:	But is every teacher that you record always either a project supervisor and/or a student mentor, or are there teachers who are neither? If so, would you like a list of all available teaching staff to choose from?
Project Coordinator:	No, I don't see much use in that. We only enter a teacher code when we need it for a mentor or a supervisor, not for everybody else.
Conclusion:	TC 2 on roles 5 + 8.

Now suppose that the project coordinator does want a list of all the teachers. Then no TC would apply to the combination of roles 5 + 8. In that case, all the teachers who are neither supervisors nor mentors must be included in the population of role 3. But we have no way to add teachers to the population of role 3, other than through a fact expression of fact type Mentorship or fact type Supervision: there is no existence postulating fact type expression for Teacher. So we have to add such a fact type expression. This is the reason for the following well-formedness rule in FCO-IM:

> If no totality constraint at all applies to any of the roles played by a certain nominalized fact type, then that fact type must have an existence postulating fact type expression.

Conversely, a nominalized fact type does not need to have an existence postulator if at least one TC applies to one of more roles that the fact type plays, but it is not forbidden: an existence postulator can always be there. In the example student-project case study, no object type has to have an existence postulator.

We close this section with a few remarks.

1 The population of a role played by a nominalized fact type is always a part of the population of this nominalized fact type itself, but such a role can never contain anything that does not appear under the fact type itself. At best, the populations are equal, in which case there is a totality constraint.

2 The final remarks 1, 2, 3 and 5 from section 3.2.3 on UCs also apply to TCs. In contrast with UCs, inter fact type TCs often occur. Most multiple role TCs are inter fact type constraints, and all single role TCs are intra fact type constraints.

3 The procedure with the table in figures 3.20, 3.21 and 3.22 cannot be used if there are other constraints (subset constraints, exclusion constraints or subtypes). A waterproof method, as in determining uniqueness constraints, with which all these types of constraints can be determined simultaneously via concrete examples, is still being developed. The table method is also more difficult to use if an object type plays two or more roles in the same fact type (it is possible, though). In all these situations the analyst can, however, work well with questions such as those in the interview above, with which he/she must beware of possibly wrong interpretations and answers. Always insist on making a question or an answer concrete by giving an example.

4 Do not impose totality constraints too lightly, in practice they work very restrictively. Suppose, for example, that we want to record many fact types about people, including the blood group for each person. If we were to put a TC on the role played by object type Person in the corresponding fact type, then it would be impossible to record anything about a person so long as we do not know the blood group (which might still have to be determined). In practice therefore totality constraints are considered to be less important than uniqueness constraints, partly because a time aspect is involved as well: maybe the supervisor of every project should be known, but sometimes it can take some time before such a fact is known. ("We are still in the process of assigning teachers as supervisors to projects, but we are understaffed and overworked so we have not reached an agreement yet."). Perhaps a distinction between a 'hard' totality constraint (must be known at once) and a 'soft' totality constraint (must be known eventually) would solve this problem. A database administrator might also decide not to adopt all the not-null rules. In short: it is better to have one totality constraint too few than one too many.

5 A tip: typical attribute-like object types such as Sum of Money, Weight, Length and so on almost always have only one TC on the combination of all the roles played by them (a selection list for possible sums of money is practically without exception absurd). An open eye for this helps to cut the procedure short.

3.5 Subset Constraints and Equality Constraints

In the starting document we underlined a part of the sentence:

> If they do so before the date on which I assign them to their project, then I try to meet their preferences as best I can.

If the project coordinator would always succeed in assigning students to a project for which they have expressed a preference, then the population of roles 14 and 15 from fact type Allocation will always be a part of the population of roles 11 and 13 from fact type Preference (see the partially depicted IGD in figure 3.23). Such a requirement that the population of roles must be a subset of the population of other roles is called a *subset constraint.*

A subset constraint (SC) is a constraint that applies to two sets of roles, which says that the sum of the populations of the one set is a subset of the sum of the populations of the other set. In the simplest situation, both sets of roles consist of only one role; this is called a *single role subset constraint.* In that case the SC says that each value that appears in the population of the one role must also appear in the population of the other role. The population of the one role is then a subset of the population of the other role. Both roles must be played by the same object type. Another simple situation is when a SC applies to two combinations of roles, with each combination coming from one fact type. Then the SC says that every value combination that occurs in the population of the one role combination must also occur in the population of the

other. The population of the one combination is then a subset of the population of the other. Both combinations of roles must be played by the same combination of object types (lexical or non-lexical). See figure 3.23.

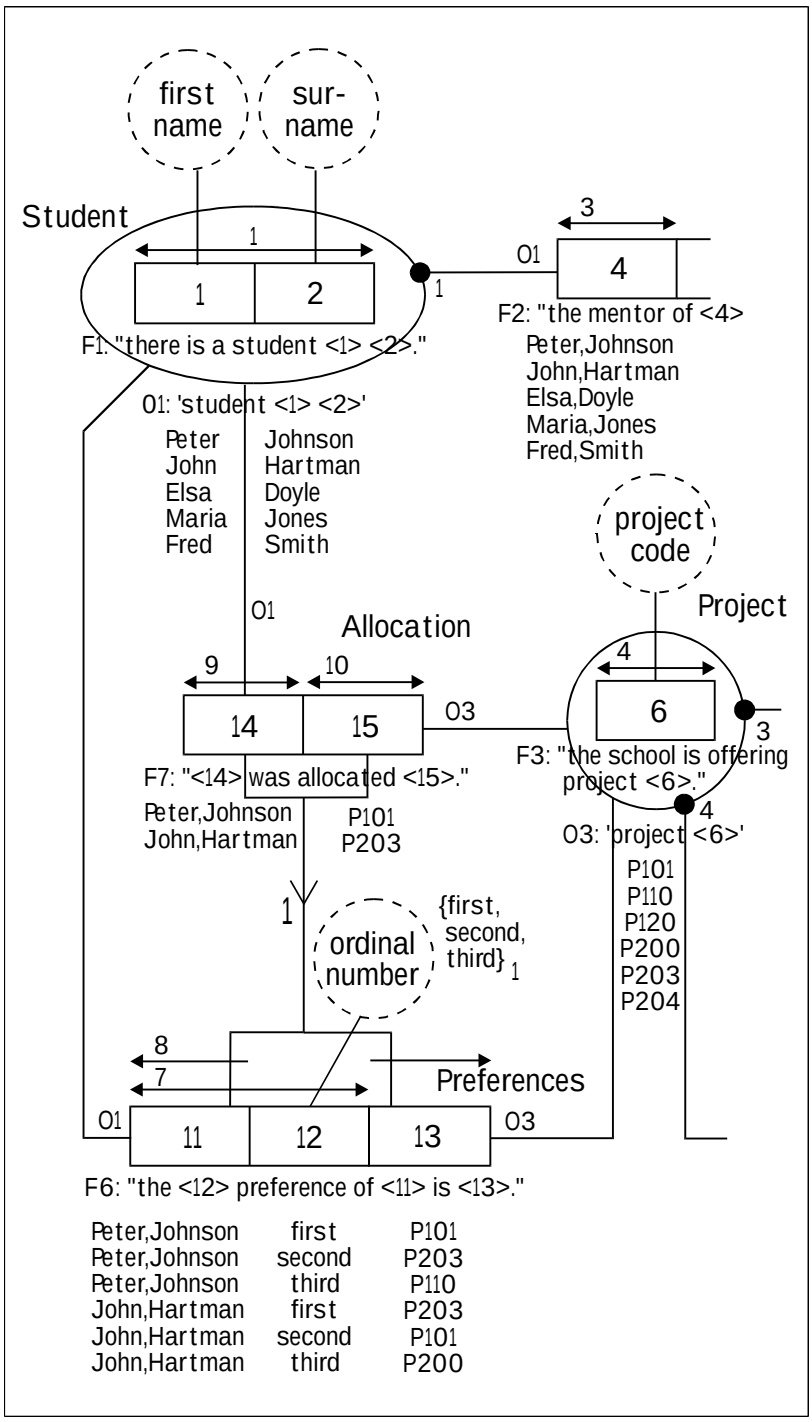


Figure 3.23: IGD with subset constraint

A subset constraint can be represented graphically in an IGD with an arrow, with a head and tail that branch themselves where necessary. The arrow points 'from small to large', that is from the subset to the whole set. See the IGD in figure 3.23, to which subset constraint 1 has been added, which points from the role combination 14 + 15 from fact type Allocation to the role combination 11 + 13 from fact type Preferences. We also often give SCs textually however, to minimize layout problems and to avoid a crowded IGD. Here we could write somewhere at the bottom of the IGD: SC 1: Allocation(14,15) -->-- Preferences(11,13). The role numbers must be written in corresponding order between the brackets. Very rarely, the graphical notation is inadequate, in which case the textual notation must be used. The FCO-IM tool always supplies such a textual notation in the IGD.

But does SC 1 actually apply in the student-project case study? If we look at the concrete example document from figure 2.4, then it indeed appears to be so. But the presence of a constraint cannot follow with certainty from examples alone (only the absence of it could), so it has to be validated. The analyst asks: "Could it happen that you assign a project to a student, for which he or she did not give a first, second or third preference?'. The answer from the School's Project Coordinator is: "Yes, that could happen. It is not always possible to meet all preferences. In addition, I assign projects to students who did not give me their preferences on time, which, of course, then do not correspond with any given preference." The analyst concludes that the subset constraint does not apply after all, and SC 1 in figure 3.23 is dropped.

If a non-lexical object type plays different roles, and some of these roles have a single role totality constraints and others do not, then automatically all the possible single role subset constraints apply from any of these roles (with or without a single role TC) to a role with a single role totality constraint. For example, in the IGD from figure 3.19, subset constraints implicitly apply from role 11 to role 4 and from role14 to role 4 (there are still more in the rest of the IGD). Such SCs that derive from TCs are redundant, and are therefore not mentioned.

An equality constraint (EC) is a constraint that consists of two subset constraints 'back and forth', meaning that the populations of both sets concerned in the subset constraints must be the same. We only give an example of an implicit equality constraint: It can be seen in figure 3.19 that there are implicit single role SCs from role 13 to role 7, from 13 to 9, from 15 to 7, from 15 to 9, from 7 to 9 and from 9 to 7. The last two form an implicit equality constraint. Naturally we also do not draw redundant ECs.

An equality constraint can also occur explicitly. We indicate an EC as two SCs or as a double pointed arrow for short. In the FCO-IM tool, ECs are represented and stored as two SCs.

There is no systematic method for determining subset constraints and equality constraints. There might be an SC or EC involved if the answer to the first question in the determination of totality constraints (can everything be filled in?) is 'no'. The analyst then, of course, asks why this is not allowed so that the constraint, if any, will follow from the rest of the interview. The analyst should in any case keep an eye open for possible SCs and ECs on roles to which no single role TC applies. For the rest, they might appear from the starting document or from interviews by chance.

A final remark: in principle each of the two sets of roles to which an SC or EC applies can be spread over more than one fact type. It is then necessary to make at least one (natural) join to compare the populations. We will not discuss such join subset constraints here any further.

3.6 Exclusion Constraints

Let us suppose in this section that some students cannot yet take part in a project, for example because they have not yet acquired the necessary skills and knowledge, and that the School's Project Coordinator has a black list of students who are not yet allowed to do a project (we do not show this list). The verbalization of this list yields fact expressions such as: "Student Fred Smith cannot do the project (yet).". Analysis yields a unary fact type Black List, which is shown in figure 3.24. The analyst suspects that a student who cannot begin a project will not be allocated a project. Then the population of role 14 from fact type Allocation cannot contain students from the population of role 16 from fact type Black List, and vice versa. Such a requirement that the populations of roles cannot have values in common is called an *exclusion constraint*.

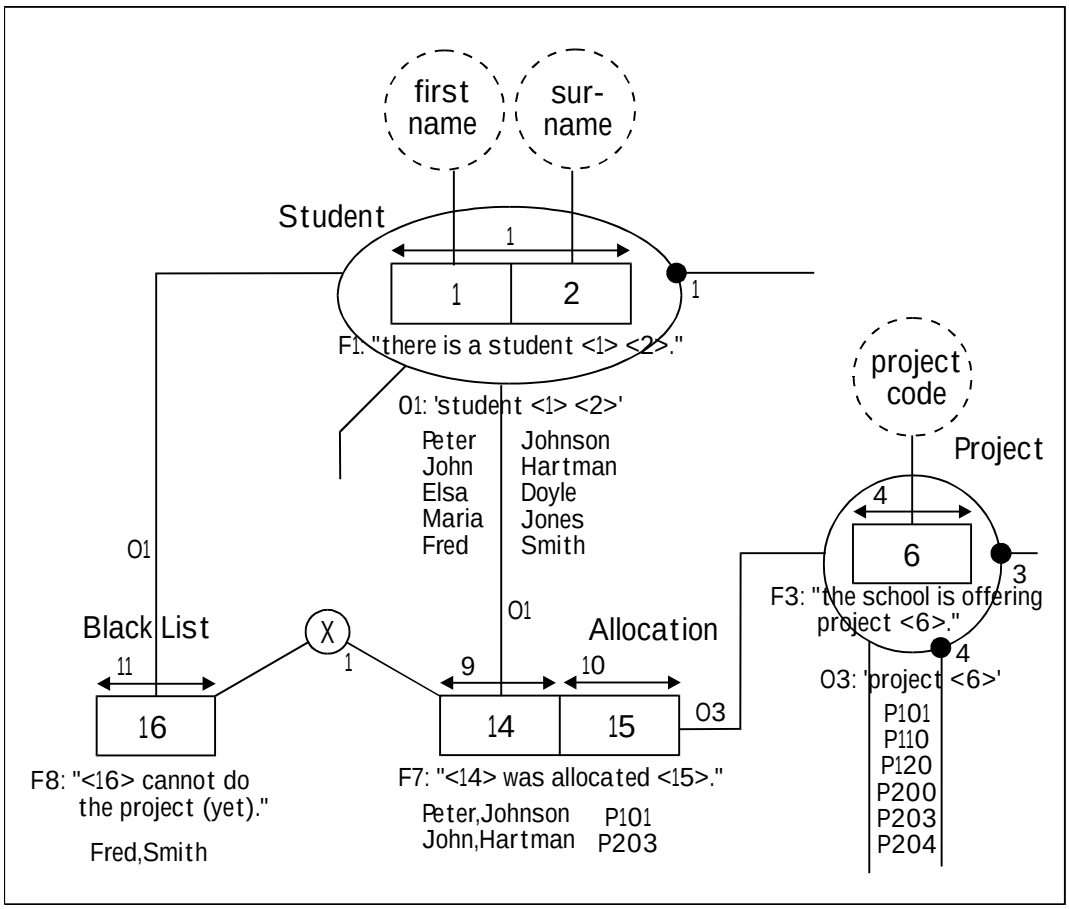


Figure 3.24: IGD with exclusion constraint

An exclusion constraint (XC) is a constraint that applies to two or more roles (or combinations of roles), which says that the populations of these roles (or combinations of roles) can have no *overlap*, that is to say they cannot have a label (or combination of labels) in common. Exclusion constraints can only be imposed on roles that are played by the same (combination of) object types, lexical or non-lexical, as is the case with subset constraints and equality constraints.

We represent an exclusion constraint in an IGD by a capital X in a little circle that we connect to the participating roles (or combinations of roles). In figure 3.24, XC 1 is drawn between role 16 and role 14. An exclusion constraint can also be represented non-graphically. We then write at the bottom of the IGD:

Exclusion constraint 1: Black List (16) --X-- Allocation (14)

The analyst must verify his/her conjectures with the School's Project Coordinator. All constraints can only be determined with certainty from explicit, concrete questions to the domain expert, never from examples alone. It might also be the case that projects are assigned in advance to students who still cannot participate, so that they can begin as soon as they get permission to start. The analyst then asks: "Is it possible you allocate a project in advance to Fred Smith, who cannot do his project yet?" "No," answers the coordinator, "I postpone allocation until he is allowed to participate." Conclusion: XC 1 does indeed apply. "Can Fred Smith make his preferences known in the meantime, and if so, do you record that?" "Yes, that is possible and I will register it as well." Conclusion: no XC between role 16 from fact type Black List and role 11 from fact type Preferences (not shown in figure 3.24, see for example figure 3.23).

There is no systematic method for determining exclusion constraints. There might be an XC involved if the answer to the first question in the determination of totality constraints (can everything be filled in?) is 'no'. The analyst then, of course, asks why this is not allowed so that the constraint, if any, will follow from the rest of the interview. For the rest, they might appear from the starting document or from interviews by chance.

3.7 Cardinality Constraints

In the starting document we underlined a part of the sentence below. Now we will direct the attention particularly to the words 'and' and 'all':

Students can choose their first, second and third preference (all different) from a list of project tasks.

When a student makes his or her preferences known, then evidently none of the three can be missing. The School's Project Coordinator confirms this conjecture. This means that all students who have stated their preferences must appear exactly three times in the population of role 11 from fact type Preferences. Such a requirement that values in a population must occur a certain number of times is called a *cardinality constraint*.

A cardinality constraint (CC) is a constraint that can be imposed on a role (or combination of roles) from a fact type, which says: if a certain value (or value combination) occurs at all in the population of this role (combination), then this value (or value combination) must occur a specified number of times in the population of this role (combination). Cardinality constraints are graphically depicted by a little circle in which the number of times is given. In the circle, whole numbers, *comparison operators* (=, <, >, <= and >=) and abbreviations (for instance: '..') can be used. Here are some examples: exactly three times: '=3'; at most five times: '1..5' or '<6' or '<=5'; exactly twice or more than four times: '=2, >4' and so on. The little circle is connected to the involved role(s) by lines.

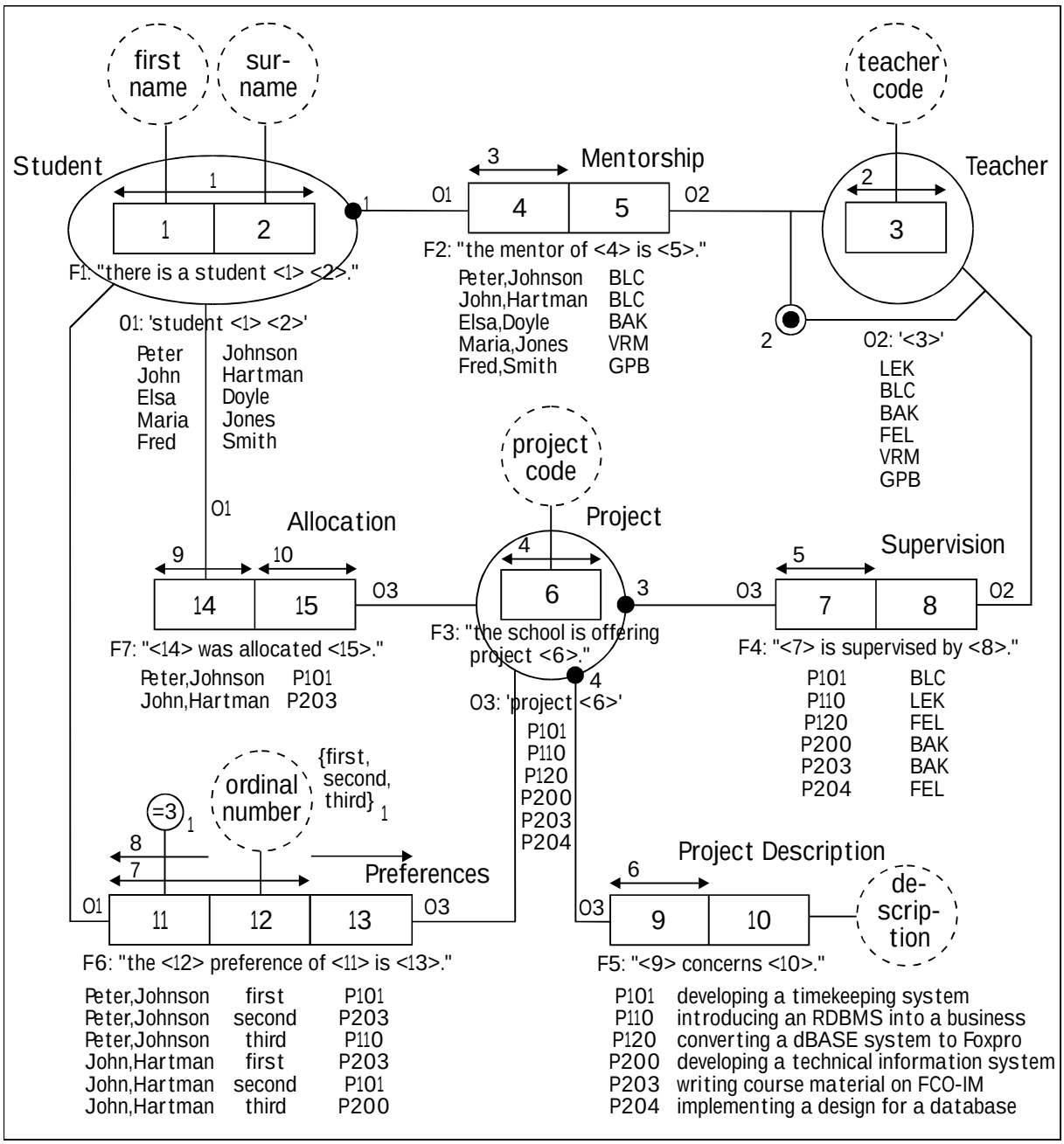


Figure 3.25: IGD with cardinality constraint

In figure 3.25 we have placed cardinality constraint 1 on role 11, which says that every value (or value combination) that appears under role 11 must appear there precisely three times. Taken together, uniqueness constraints 7 and 8, value constraint 1 and cardinality constraint 1 say that for each student whose project preferences are recorded, exactly three different preferences must be registered with the ordinal numbers first, second and third.

There is no systematic method for determining cardinality constraints. They might appear from the starting document or from interviews by chance.

A uniqueness constraint is actually the same as a cardinality constraint with '=1'. Inter fact type cardinality constraints can therefore exist as well. For historic reasons, and because of the rarity of other cardinality constraints, and because of the algorithm to find intra fact type UCs, we prefer to discuss the UCs separately.

In figure 3.25, the final IGD with all the constraints is shown.

3.8 Final Remarks

1 The kinds of constraints we have discussed so far make up a widely accepted standard set in the NIAM tradition, with standard graphic symbols. We therefore call them *standard constraints*. There are also many other kinds of constraints, such as: the sum of all percentages must equal 100. We call these *non-standard constraints*. We can depict each non-standard constraint by a small circle that is connected to the roles to which the constraint applies. In the circle we write the letter C (from constraint) with a sequence number. Textually we give elsewhere a description of the constraint with a concrete example of a violation of the constraint (see also remark 2).

2 The FCO-IM tool supports the recording of the following constraints: uniqueness constraints, totality constraints, subset constraints and value constraints that can be specified by enumeration. Taking these constraints into account guarantees the correct derivation of the most important integrity rules of the Relational Model during the GLR algorithm: not null constraints, uniqueness constraints (those in primary keys as well as others), references (foreign key references as well as other references) and domain constraints. The behavior of other constraints under the GLR algorithm is quite irregular and often leads to non-standard integrity rules for which no standard formulation exists (this even happens sometimes with the constraints mentioned): see various examples in this book.

3 According to the general principles of FCO-IM, every modeling decision is made on the basis of concrete examples or counter-examples, which are judged by the domain expert. The account of the determination of the constraints should always include a concrete example of a violation of each constraint. This happens automatically during the determination of uniqueness constraints and totality constraints in the tables used in those procedures. For the other constraints, the analyst must supply these him/herself.

4 A certain economy in the addition of constraints is sensible, since each constraint makes something impossible. For psychological reasons the user will sooner answer 'yes' to the question: "Must every student express exactly three preferences?" than to the question: "Do you want to make it impossible for a student to ever express more or less than three preferences?". Cardinality constraint 1 (see figure 3.25) is therefore a bit dubious: would the School's Project Coordinator really want to treat a student who has only given two preferences as if he/she had given no preference at all (the coordinator cannot record just two preferences if CC 1 really applies)?

5 A population in an IGD that satisfies all the constraints is called a *proper population*. The IGD in figure 3.25 contains such a population. It is often very difficult (sometimes impossible) to create a proper population with just a few tuples per fact type. It therefore does not really matter if an example population is not proper.

6 The definitions and descriptions of all constraints have been given in terms of label populations, not in terms of tuple pointers. It is a very simple matter, however, to rephrase everything in terms of tuple pointers, so we will not do this explicitly.

4

Derivation of a Relational Schema

In chapters 2 and 3, we showed how to build up an information grammar diagram (IGD) from a starting document by collecting concrete examples, verbalizing, classifying and qualifying, drawing a diagram and adding constraints. In the last named process, we also carry out some tests to guarantee that the result consists only of elementary fact types. We call an IGD that contains only elementary fact types an *elementary IGD.* In this chapter we will discuss how to derive a relational database schema from an elementary IGD, that is to say a database schema with which we can create a relational database for recording the modeled information in a *redundancy free* way. We assume that the reader is familiar with the concepts and the terminology of the Relational Model.

The derivation of a relational schema is done by carrying out three transformations on the elementary IGD: *grouping* (section .4.1), *lexicalizing* (section 4.2) and *reducing* (section 4.3). We therefore call this the *GLR algorithm.* Almost always during this process, an IGD arises with non-elementary fact types, however without introducing redundancy. Moreover, the IGD still generates exactly the same fact expressions after each step (but the user might decide to drop some sentences in the reducing step). In a fourth step we will use standard conventions to derive meaningful table names and column names from the IGD (section 4.4).

All this is also implemented in the FCO-IM tool, which accompanies this text. The FCO-IM way of modeling information leads to a very simple CASE-tool architecture, in which the derivation of the relational schema can de done through simple updates in the repository itself. It is then an easy matter to generate DDL for arbitrary implementation platforms. We give an example of this in section 4.5.

In section 4.6 we conclude the chapter with a few final remarks.

4.1 Grouping

Facts from different elementary fact types can often be placed in one table when a relational schema is derived from an IGD. The aim of *grouping* is to combine as many fact types as possible in the same table without introducing redundancy. The resulting number of tables is then as small as possible. In this section we will show how an *elementary* IGD (El-IGD) is transformed by grouping into a *grouped* IGD (G-IGD). Although almost always fact types arise during grouping that are not elementary anymore, the G-IGD still models the same elementary fact expressions as the El-IGD from which it originated. We will first illustrate the process with the example student-project case study in section 4.1.1, and give a general grouping algorithm in section 4.1.2.

4.1.1 Grouping in the Student-Project Case Study

We start from the last IGD of the example student-project case study (see figure 4.1). The only fact types that can be combined are binary fact types in the elementary IGD with a single role uniqueness constraint on a role played by a nominalized fact type (*non-lexical role*). Therefore, we mark with a 'G' all the roles in the elementary IGD that satisfy the following three conditions (see figure 4.1):

1 The role is part of an n-ary fact type, with n greater than 1.
2 The role is played by a non-lexical object type.
3 The role falls under a single role uniqueness constraint.

In section 4.1.2 two further conditions will be added.

Condition 1 is formulated a little more generally because fact types can be formed during the grouping process with more than two roles, amongst which roles that satisfy conditions 2 and 3 (see for example figure 4.3, role 15). Because of the n-1 rule (see section 3.3.1.1), such a fact type is no longer elementary. But in an elementary IGD only roles in binary fact types can be considered for grouping. (We will adjust this when we discuss specialization (see section 6.1) and generalization (see section 6.2), and we will refine condition 1 in section 7.1.1.)

The nominalized fact types that play the marked roles will 'swallow up' the fact types with the marked roles, deleting the marked roles and moving the remaining roles into the nominalized fact types. We illustrate this process in figures 4.2 and 4.3, in which we have left out the fact type expressions and the populations for clarity; we will deal with those later.

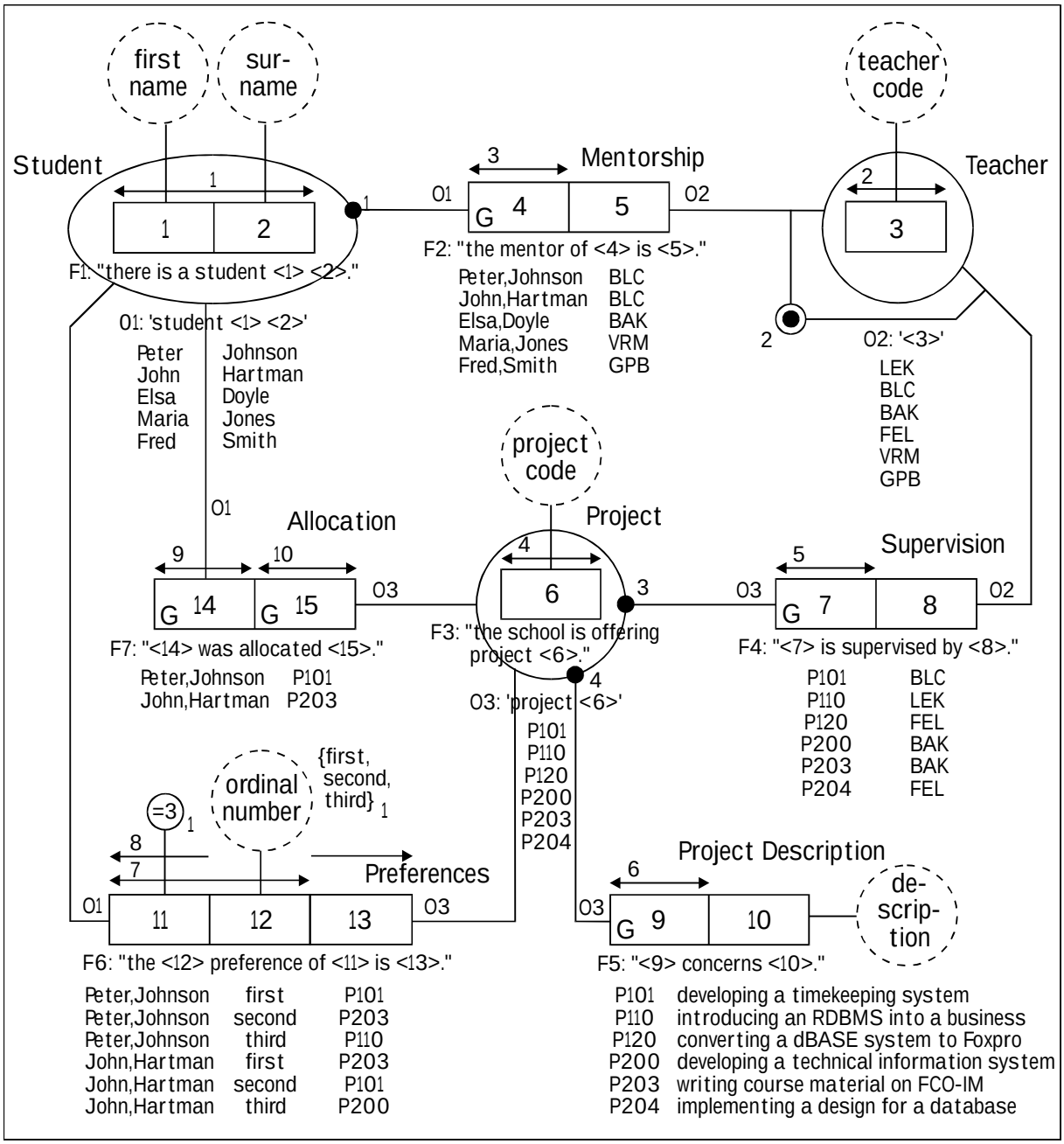


Figure 4.1: El-IGD with roles marked for grouping

Figure 4.2 shows the situation just after deleting the marked roles. The remaining part of fact type Mentorship (role 5) is absorbed by fact type Student, and fact type Project absorbs the rest of both fact types Supervision and Project Description. For fact type Allocation there are two options. The first option is to delete role 14 so that role 15 is absorbed by the nominalized fact type Student. In that case we will eventually get a table 'Student', in which the assigned project can be recorded for each student. The second option is to delete role 15 so that fact type Project absorbs role 14. Then we eventually get a table 'Project', in which we can record for each project the student who has the project assigned to her/him. The choice between these

two options is completely free here (see step 2a in section 4.1.2). We choose the first option because the School's Project Coordinator is accustomed to looking up an assigned project in the student list, and not the other way around.

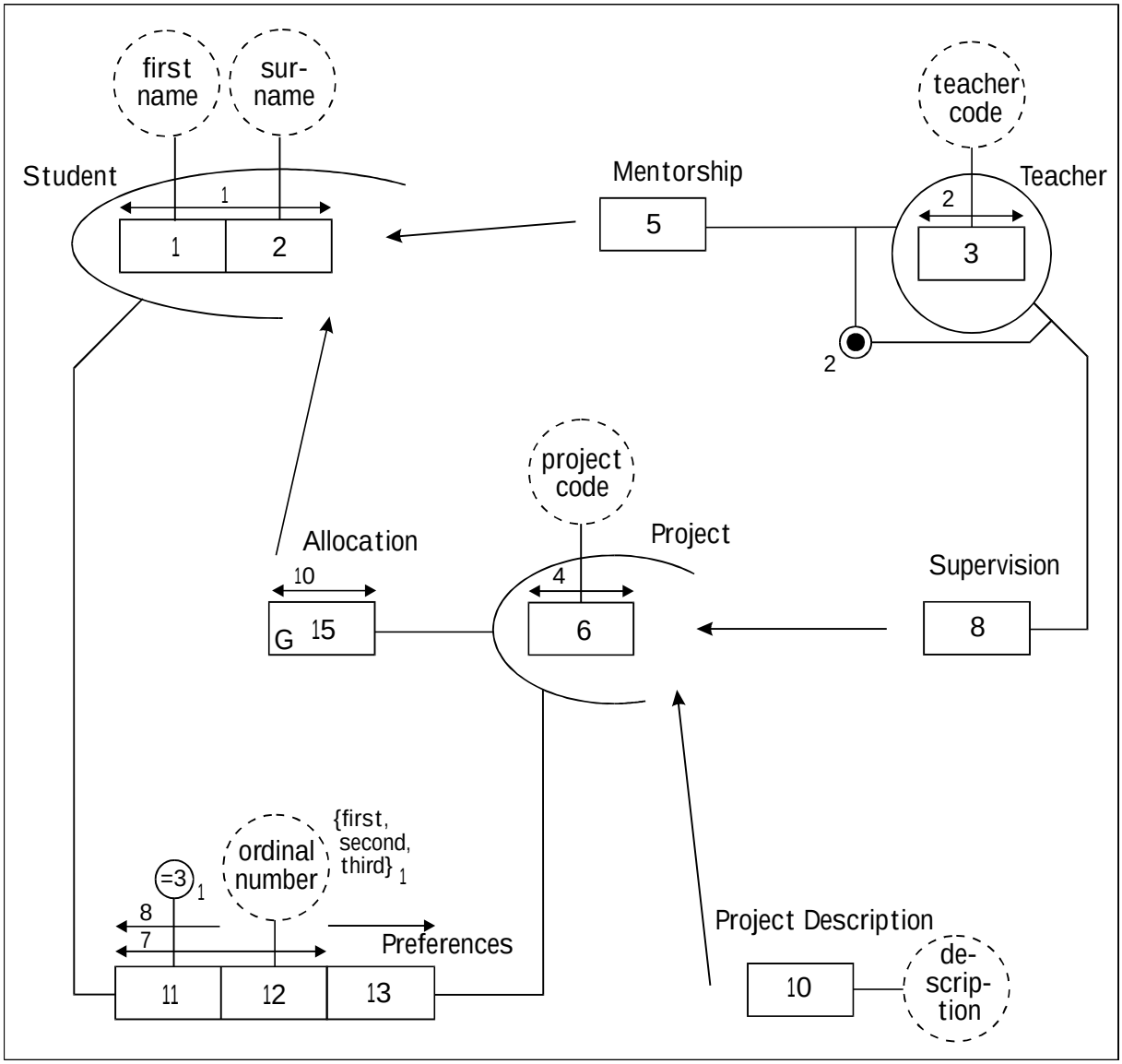


Figure 4.2: the grouping process (fact types only) halfway through

Figure 4.3 shows the resulting grouped IGD, still without fact type expressions and populations. The fact types that the marked roles were a part of have disappeared; we say that these fact types have been *grouped away*. If the fact types that were grouped away had been nominalized fact types themselves, that is to say had themselves played one or more roles, then these roles would now be played by the object type that absorbed the remaining roles. This does not occur in the student-project case study, however. Fact types Student and Project are now not elementary anymore: they no longer satisfy the n-1 rule. Absorbed roles retain their uniqueness constraints if they had any, such as role 15. Role 15 is still available for grouping according to conditions 1, 2 and 3 shown above, and role 15 is now also marked 'OP'. We will go into both these issues later.

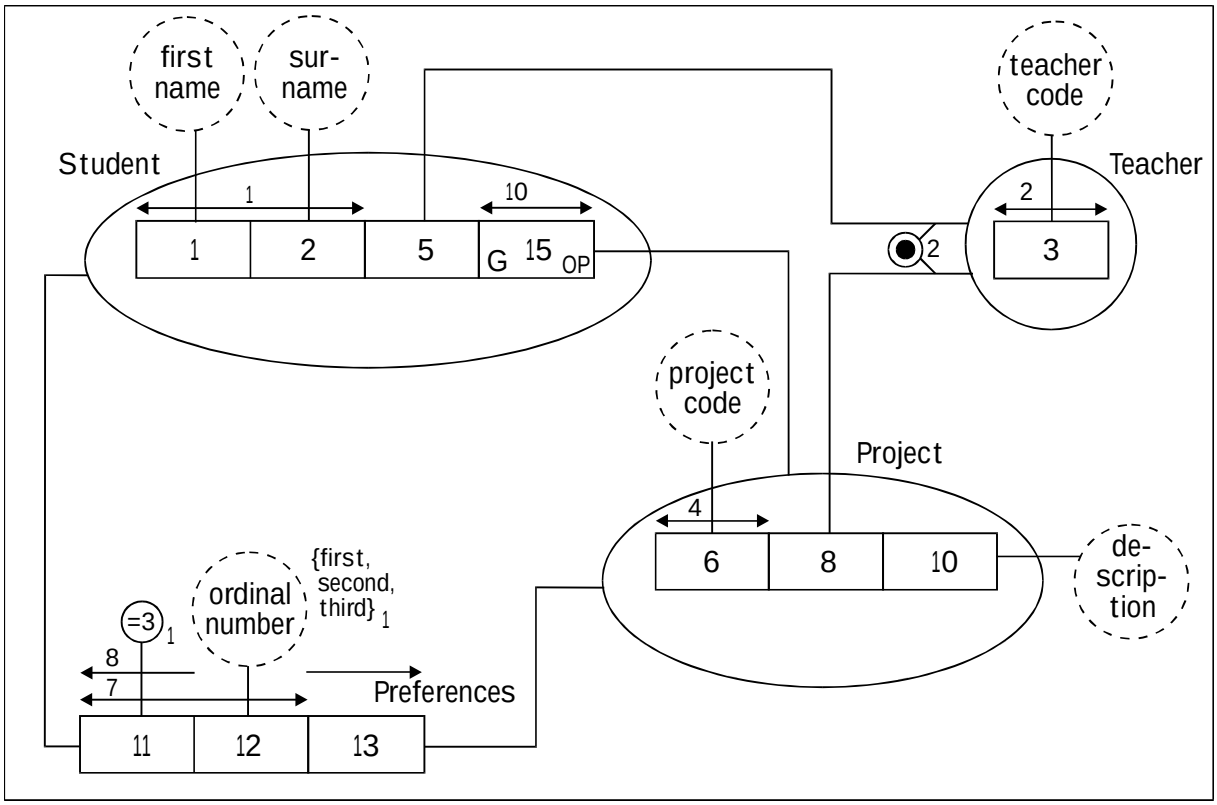


Figure 4.3: the grouping process (fact types only) completed

What happens to the fact type expressions, the object type expressions (if any) and the populations of the fact types that were grouped away? See figure 4.4 for this. The fact type expressions and object type expressions (if any) of a disappeared fact type are added to the fact type that absorbed the remaining roles. Object type expressions that correspond with the now deleted roles are filled in in the blanks in the fact type expressions and object type expressions that refer to the deleted roles.

So after deleting role 4, we change fact type expression F2: “the mentor of <4> is <5>.” from fact type Mentorship to: F2: “the mentor of student <1> <2> is <5>.”, and add it to fact type Student, which absorbs the remaining role 5 from Mentorship. Fact type expressions F7 from Allocation, F4 from Supervision and F5 from Project Description are changed and moved likewise. If a nominalized fact type has absorbed all the fact types it played a role in, then the nominalization is dropped together with the corresponding object type expression because there is no role left to substitute it in. That, however, does not happen here.

Of course, the population of each absorbed role must be taken along as well. The original facts must stay intact, and so each value (or value combination) from the population of the absorbed roles must be added to the right tuple under the fact type that absorbed the role. So we must add ‘GPB’ under role 5 to the tuple ‘Fred Smith’ under Student, because ‘GPB’ was coupled originally to the value combination ‘Fred,Smith’ under role 4. Analogously, ‘P203’ under role

15 cannot be inserted into any tuple except the one with 'John Hartman'. There is always only one tuple to which we can add a value (or value combination) from the remaining role(s) of a fact type that was grouped away, because there was a single role uniqueness constraint on the deleted role.

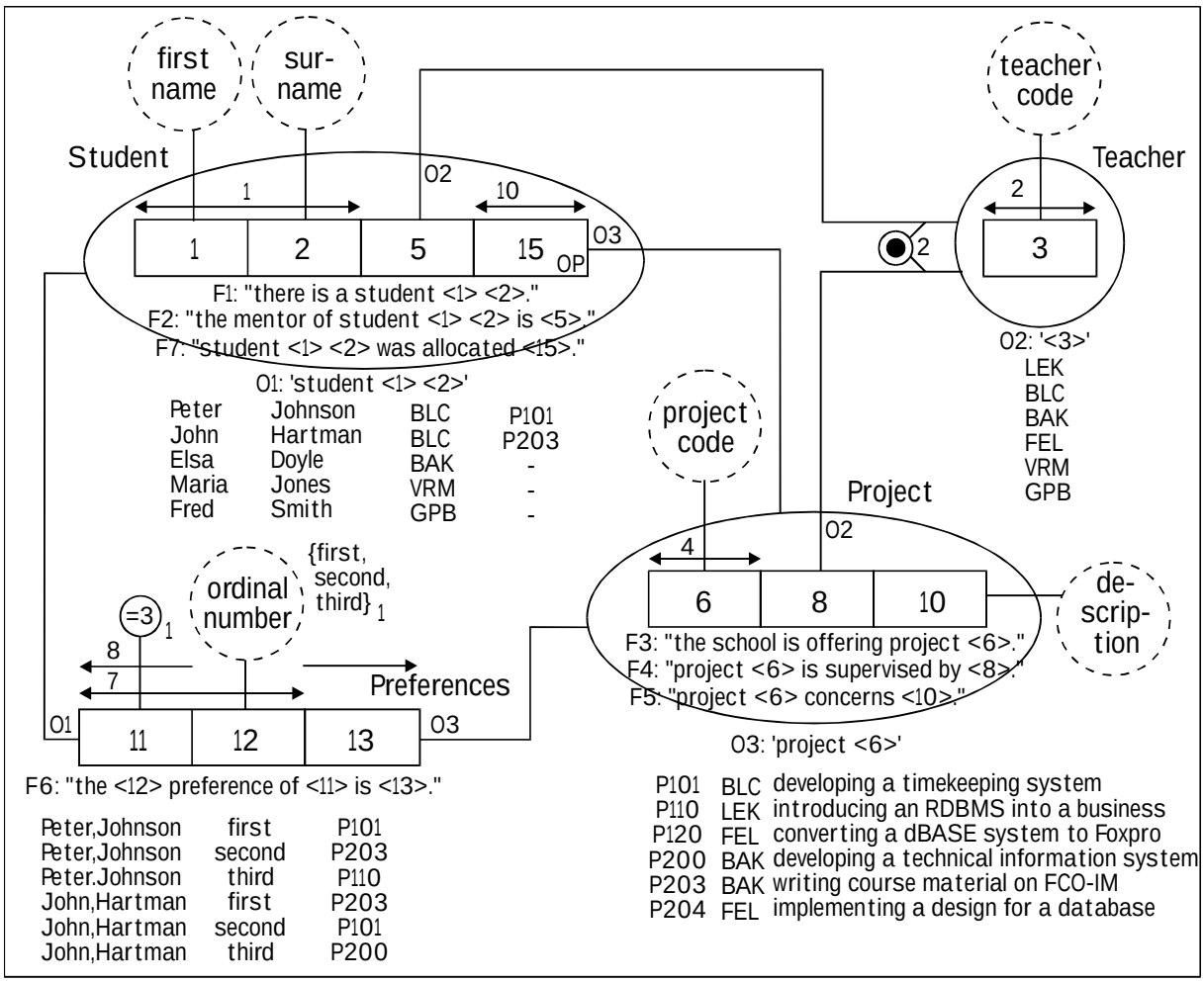


Figure 4.4: grouped IGD

There is not a value for all the students under role 15 in figure 4.4 however, because not all students occur in the original fact type Allocation. This reflects the fact that the deleted role 14 did not have a single role totality constraint. This is why the population of fact type Student contains *null values*, indicated by hyphens, for all students without an assigned project. Null values cannot occur in an elementary fact type, they arise only as a result of the grouping (we will adjust this for generalization, see section 6.2). A role under which null values can occur is called an *optional role*. The mark 'OP' in role 15 is an abbreviation of 'optional' and indicates that null values can appear under role 15 (or rather: that values may be missing in the population). Roles under which null values cannot occur is often marked 'NN', short for *not null*. Because all roles are NN in an elementary IGD, we usually leave out this indication and only write 'OP' in all optional roles.

If a fact type is grouped away, then the absorbed remaining roles from this fact type become optional, unless the (deleted) marked role had a single role totality constraint. This is because in the absence of a single role TC, the populations of the deleted role and of the object type that played it do not have to be the same. Because no totality constraint applies to role 14, role 15 becomes optional after deleting role 14. Because a single role totality constraint does apply to role 7, role 8 does not become optional; for the same reason, role 5 and role 10 remain NN.

Finally: we have removed the mark 'G' (marking the role for grouping) from role 15 in the IGD in figure 4.4. This was done because the role has become optional. Further grouping could then lead into trouble (see section 7.2.3). Condition 4 in section 4.1.2, step 1 below is meant to prevent these difficulties. In section 4.6 and in chapter 7 we will look into this further. (Condition 5 in section 4.1.2, step 1 prevents difficulties resulting from recursive structures; we will not discuss this further.)

4.1.2 Procedure for Grouping

We now give an algorithm for grouping. We only show the main lines and do not claim to be complete, especially where the treatment of all kinds of constraints is concerned. Some parts of the algorithm have not been discussed in section 4.1.1, but can be simply verified with the use of concrete examples.

1 Mark each role that satisfies all five conditions below; such a role can be deleted during the grouping process.
   1 The role is part of an n-ary fact type, with n greater than 1.
   2 The role is played by a non-lexical object type.
   3 The role falls under a single role uniqueness constraint.
   4 The role is not optional.
   5 The role is not *directly recursive*, i.e. the role is not played by a nominalized fact type in which it sits itself.

2 For each marked role in turn, carry out steps 2a - 2h.

   a If more than one role is marked in the same fact type, then deal with these roles one by one as well. There are two cases:
      1 There is only one role with a single role totality constraint. Then deal with this role first.
      2 There is more than one marked role with a single role totality constraint, or none of the marked roles has a single role totality constraint. Then it is a free choice which role is dealt with first. Different choices can lead to different equivalent relational schemas. Together with the domain expert we can choose one at will, for example based on existing habits, or on performance considerations.

b If the fact type is nominalized, playing roles itself, then choose another fact type that is not nominalized first if possible. Otherwise: after grouping-away, let these roles be played by the nominalized fact type that absorbed the remaining role(s). Adjust all the constraints.

c Delete the (chosen) marked role and its uniqueness constraint, and put the remaining roles of the fact type into the nominalized fact type that played the deleted role. Any uniqueness constraints on the remaining role(s) do not change. Delete the object type/fact type the marked role was a part of.

d Process the totality constraints (if any) that applied to the deleted role. There are three cases:
   1 If no single role TC or multiple role TC applied to the deleted role, then all the roles absorbed in step 2c become optional.
   2 If a multiple role TC applied to the deleted role, amongst others, then all the roles absorbed in step 2c become optional. The multiple role TC is deleted and must be replaced later with an equivalent constraint on the columns of the eventually resulting tables. No general algorithm can be given for this yet. It is generally not difficult, however, to formulate such a constraint.
   3 If a single role TC applied to the deleted role, then there is no change in the existing NN or OP indications of the remaining (absorbed) roles. The single role totality constraint is deleted.

e Process all the other constraints that concern the deleted role: they must be deleted and replaced later with equivalent constraints on the columns of the eventually resulting tables. No general algorithm can be given for this yet.

f Process the fact type expressions and object type expressions (if any) from the fact type that was grouped away: fill in the relevant object type expression in the blanks referring to the deleted role and move everything to the nominalized fact type that absorbed the remaining roles. If the nominalized fact type that played the deleted role does not play any other roles anymore, then cancel the nominalization and delete all corresponding object type expressions.

g Add each tuple from the population of the remaining role(s) of the grouped-away fact type to the right tuple in the population of the fact type that absorbed these roles; put null values under optional roles where necessary.

h Remove the 'G' mark from a marked role if it no longer satisfies condition 4 or 5 from step 1 as a result of steps 2a up to and including 2g (the properties in conditions 1, 2 and 3 cannot change).

3 If there are no marked roles left, then the grouping process is finished, otherwise step 2 is carried out again for another marked role.

4.2 Lexicalizing

In the further derivation of a relational schema from a grouped IGD, in principle a separate table will arise for each fact type. In this next step, we must fulfill a requirement of the Relational Model, namely that each attribute (column) of each relation (table) has a *lexical domain*. Translated back to IGDs in FCO-IM, this means that each role must be played by a label type. In a grouped IGD, however, there are almost always roles that are still being played by non-lexical object types (see figure 4.5). The aim of *lexicalizing* is: to transform the fact types so that all roles are played by label types, of course without introducing redundancy. In this section we will show how a *grouped IGD* (G-IGD) is transformed by lexicalizing into a *grouped and lexicalized IGD* (GL-IGD).

The resulting GL-IGD still models the same elementary sentences, and therefore the same information as the El-IGD and the G-IGD from which it arises. We will first illustrate the process with the example student-project case study in section 4.2.1, and give a general lexicalizing algorithm in section 4.2.2.

4.2.1 Lexicalizing in the Student-Project Case Study

We start from the G-IGD of the example student-project case study (see figure 4.5). We mark with an 'L' all the roles that are still not played by a label type: roles 5, 8, 11, 13 and 15. We must lexicalize these roles, that is to say we must transform the IGD in such a way that eventually all the roles are played by label types. This boils down to eliminating all nominalizations in the IGD.

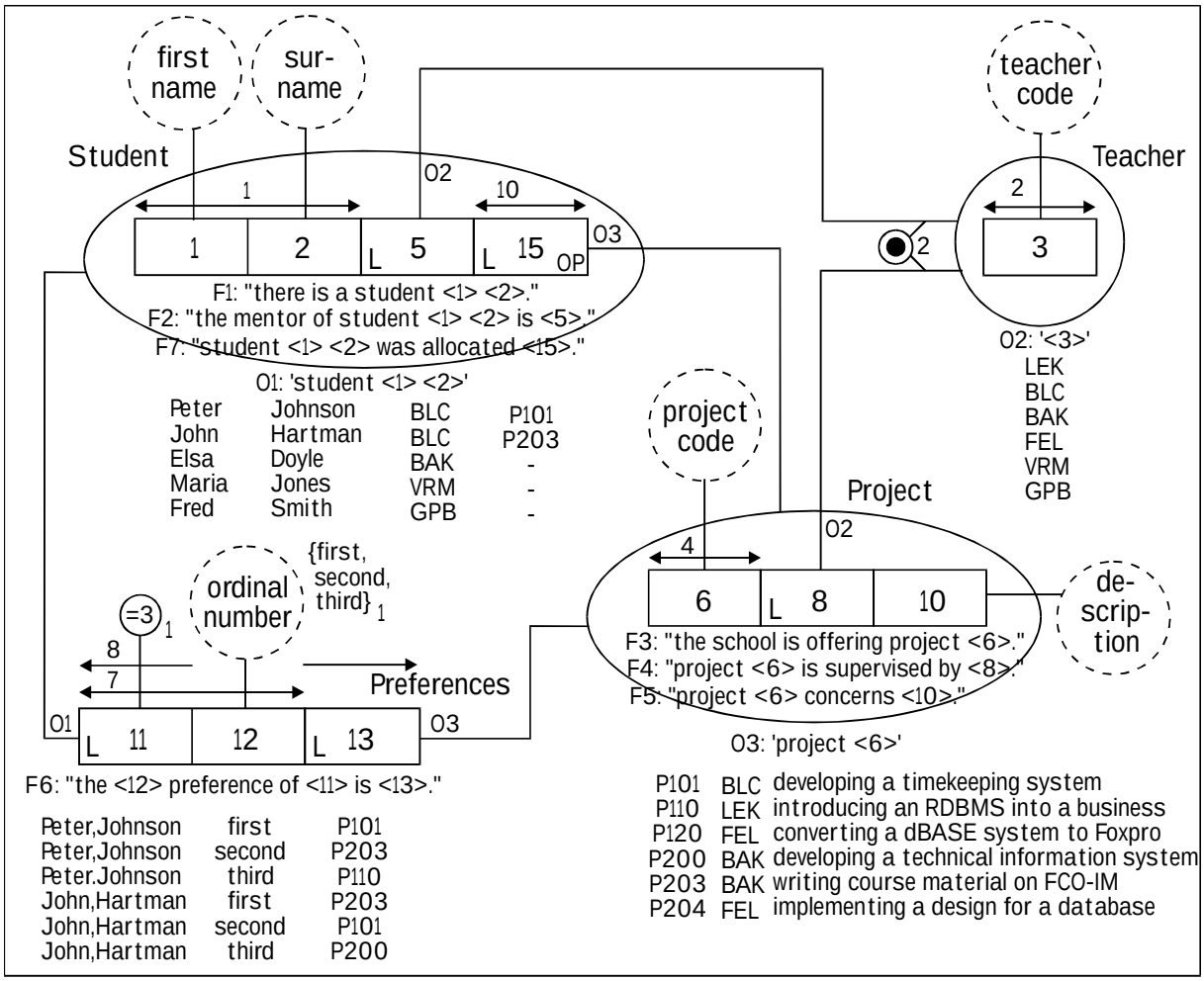


Figure 4.5: G-IGD with roles marked for lexicalizing

We will now lexicalize all the marked roles. We start with role 5, which is played by object type Teacher. The identifier of object type Teacher consists of one role, namely role 3. We can tell by looking at object type expression O2, which is to be filled in for role 5, and in which only role 3 occurs (we can of course also see this here from the fact that there is only one role in fact type Teacher, but that is not the case in general). To lexicalize role 5, we 'cut it off' from object type Teacher and attach it to the (lexical) object type that plays the identifying role of object type Teacher, namely label type 'teacher code'. We do the same with role 8. Figure 4.6 shows the situation halfway through the lexicalizing process, in which we have left out the fact type expressions and the populations for clarity: we will deal with those later. We now analogously cut off roles 13 and 15 from object type Project. Object type expression O3 is to be substituted into both roles, which tells us that the identifier of Project consists only of role 6 (we can also see this here from the fact, that there is only one UC on fact type Project over role 6, but that is not the case in general). We therefore attach role 13 and role 15 to the (lexical) object type 'project code', which plays this identifying role. Next, we will lexicalize role 11. It is clear from object type expression O1, which applies to role 11, that the identifier of object type Student consists of the two roles 1 and 2. We therefore split role 11 in two and

give the new roles the numbers 11.1 and 11.2, and we attach these two roles to the (lexical) object types that play the identifying roles of Student: label types ‘first name’ and ‘surname’. See figure 4.6.

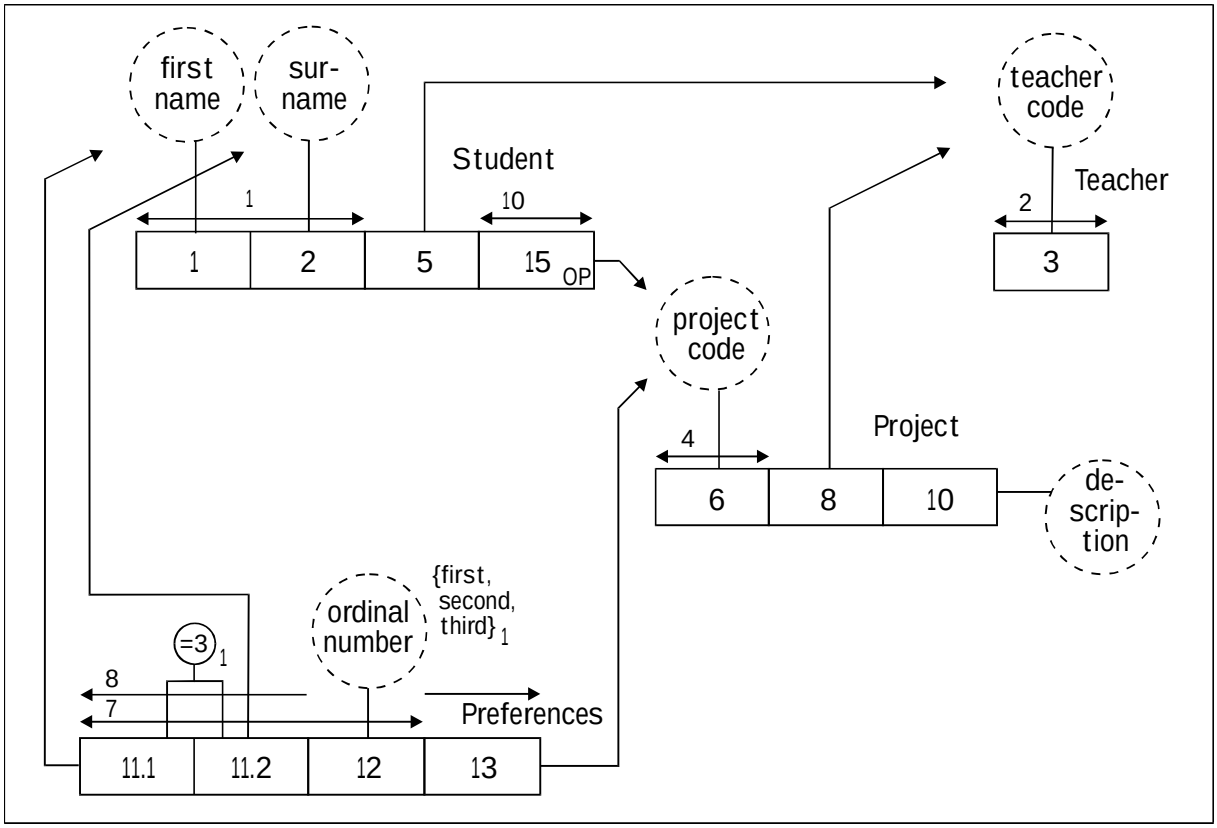


Figure 4.6: the lexicalizing process (fact types only) halfway through

It is often the case, that there are non-lexical roles among the identifying roles of an object type that plays the role, which we want to lexicalize: there are then other nominalized fact types ‘higher up’. In such a situation it is best to lexicalize the ‘highest’ non-lexical role first: this is the non-lexical role that is played by an object type of which all the identifying roles are lexical already. There always exists such a role, and from there on one can work ‘down’ again. There is no such situation in the G-IGD of the example student-project case study, however.

The process of lexicalizing a certain role can also be regarded as deleting this role and replacing it with a copy of the identifying role(s) that occur in the object type expression that applies to the deleted role.

The processing of the fact type expressions, object type expressions and populations is simple. We substitute the object type expression that applies to a role being lexicalized into each blank that refers to this role (in a fact type expression or in an object type expression), as was discussed in section 2.6. We eventually obtain the LTL-FTEs (see section 2.4.2), except for the use of role numbers in the blanks instead of the names of the label types that play the roles.

After all roles have been lexicalized, there no longer are any nominalizations and all object type expressions are substituted into the fact type expressions. See figure 4.7.

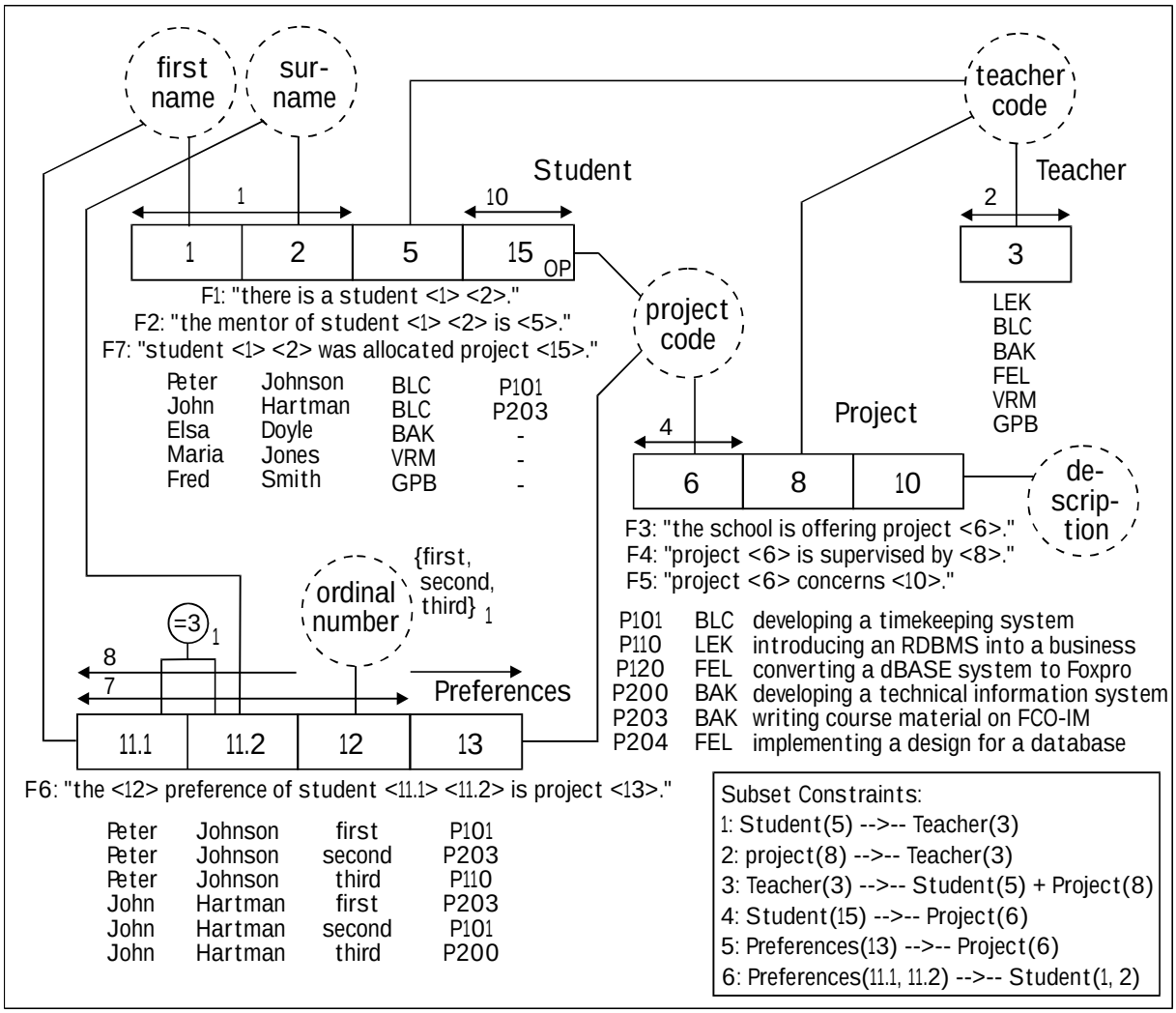


Figure 4.7: lexicalized IGD, still with lost tuples and lost fact type

The populations change only if a role is split, in which case the value combination under the role to be split is divided over the correct new roles in each tuple. After lexicalizing, there are no more roles with a compound population. When tuple numbers and tuple pointers are used (see section 2.11): the tuple pointers under each lexicalized role are replaced with the corresponding labels. After lexicalizing, there are no more roles with tuple pointers.

Next, we process the constraints. We will first consider subset constraints that arise from detaching roles, and then we will consider subset constraints that arise from deleting totality constraints, and finally we will consider all remaining constraints.

At least one subset constraint is always generated during the lexicalizing of a role. The reason for this is that the population of a non-lexical role in FCO-IM is necessarily a part of the

population of the object type that plays the role (see section 3.4, final remark 1). So, the population of role 5 before it is detached from object type Teacher is automatically a part of the population of role 3, see figure 4.5. This must remain so even after role 5 is detached and attached elsewhere. For that purpose we must now give an explicit subset constraint: SC 1 in figure 4.7. In an analogous way, SCs 2, 4 and 5 arise respectively from the lexicalizing of roles 8, 15 and 13. The direction of each subset constraint is from the lexicalized role (or roles, if the role was split) to the role(s) occurring in the object type expression that applies to the lexicalized role. After the lexicalizing of role 11, SC 6 therefore points from roles 11.1 and 11.2 to roles 1 and 2, because these roles were in O1, which applies to role 11. In the example student-project case study, five roles were lexicalized so that five SCs were generated. The sixth SC (SC 3 in figure 4.7) follows from the totality constraints and is discussed below.

There are no totality constraints anymore after lexicalizing because there are no nominalized fact types left. Each totality constraint (TC) that disappears during lexicalizing produces a subset constraint as well. In figure 4.5, TC 2 says that each tuple under role 3 must also occur under role 5 and/or role 8, so that the population of role 3 must be the same as the sum of the populations of roles 5 and 8. That is why SC 3 is added in figure 4.7. SCs 1, 2 and 3 together form an equality constraint, which is equivalent with the disappeared TC 2. In general the following holds: each TC that disappears yields an SC, which forms an equality constraint together with one or more SCs that 'point the other way' and that arose from detaching roles.

The remaining constraints that apply to roles involved in the lexicalizing must be adjusted. For example: UCs 7 and 8 now concern three roles because role 11 was split, and cardinality constraint 1 now applies to the role combination 11.1 + 11.2.

Finally, we notice that something is wrong with the population of fact type Teacher, see figure 4.7. Fact type Teacher has no fact type expression: there was no existence postulator and Teacher has absorbed no roles during grouping. Moreover, the object type expression for Teacher was dropped (after substituting it in fact type expressions F2 and F4), because now Teacher is no longer nominalized. Therefore, the population of Teacher cannot generate any verbalization anymore: neither in the form of a fact expression, nor as an object expression. Such tuples that cannot regenerate any (part of) the original verbalization are called *lost tuples.* We must remove lost tuples from the GL-IGD. Because fact type Teacher has no fact type expression, we must remove all its tuples. Fact type Teacher is then always empty and loses its right to exist. Such a fact type that can have no population is called a *lost fact type*. We therefore also remove fact type Teacher including all constraints that have anything to do with it. Here, those are the subset constraints 1, 2 and 3. The end result of the lexicalizing of the G-IGD from figure 4.5 is shown in figure 4.8.

A few final remarks:

1 Also in the situation of working with tuple pointers, the labels in the population of fact type Teacher are not needed anymore: in figure 4.5, roles 5 and 8 would still contain tuple pointers, but in figure 4.7 these would be replaced with the corresponding real labels because roles 5 and 8 have become lexical there.

2 The removal of lost tuples does not always lead to lost fact types: sometimes only some of the tuples of a fact type are lost, for example when a fact type without an existence postulator absorbs roles during grouping.

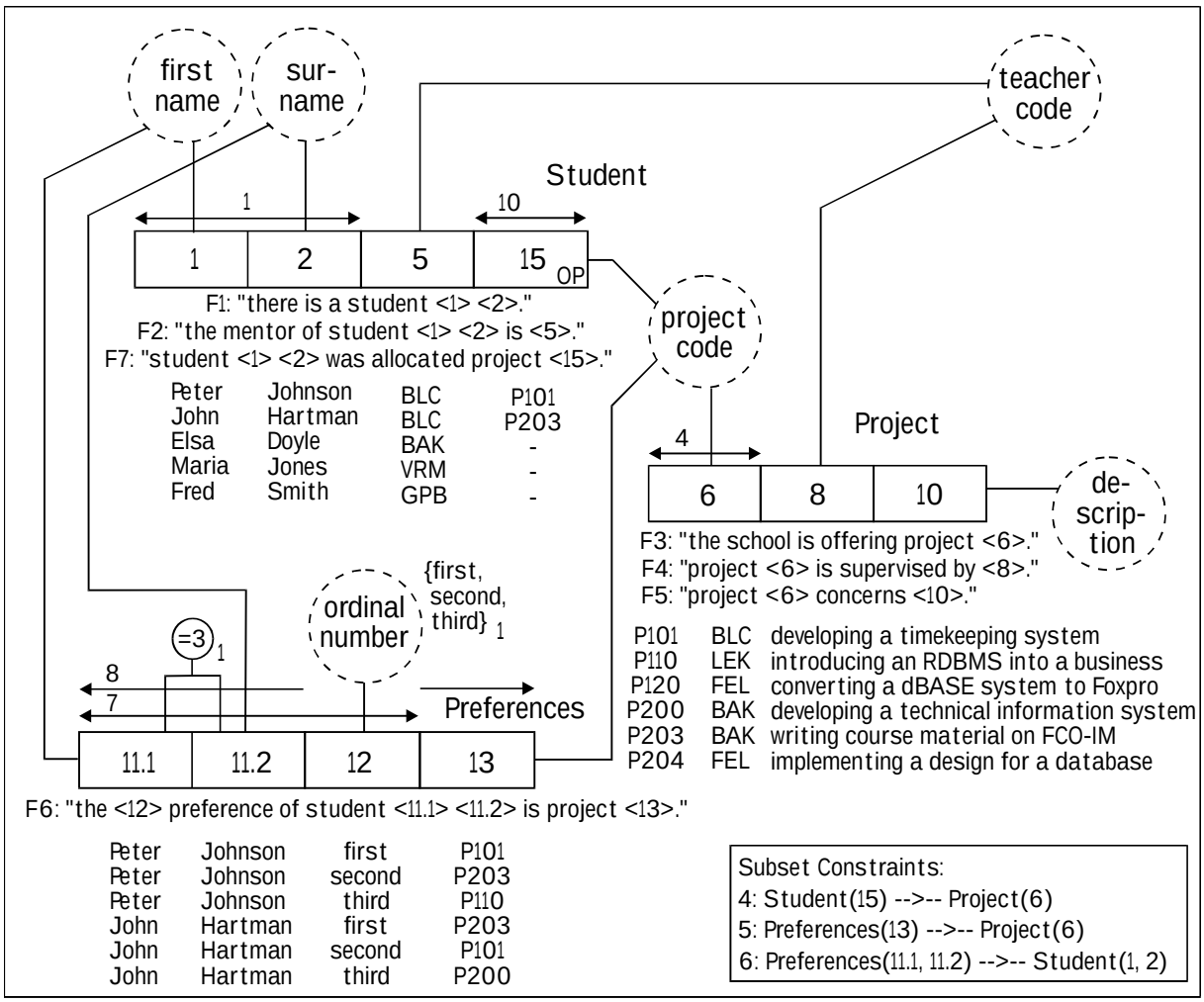


Figure 4.8: grouped and lexicalized IGD

4.2.2 Procedure for Lexicalizing

We now give an algorithm for lexicalizing. We show only the main lines, and do not claim to be complete (sometimes more than one object type expression applies to a certain role, for instance in the case of subtle substitution, but the algorithm can be extended easily for such situations). Some parts of the algorithm have not been discussed in section 4.2.1, but can be simply verified with the use of concrete examples.

1 Mark all non-lexical roles.

2 For each marked role in turn, carry out steps 2a - 2g:

a Find a marked role with the property that all roles that appear in the object type expression for this marked role are already lexical (there is always such a role).

b Detach the marked role from the object type that played it. If the object type expression that applies to the marked role concerns only one role, then attach the role to the label type that plays the role referred to in the object type expression. If the object type expression that applies to the marked role concerns more than one role, then split the marked role in the same number of roles and attach these roles to the label types that play the roles referred to in the object type expression.

c Substitute the object type expression that applies to the marked role into all fact type expressions and object type expressions (if any) that refer to the marked role, and process the populations.

d Add a subset constraint pointing from the lexicalized role (after splitting: from all resulting roles) to the role(s) referred to in the object type expression that applies to the lexicalized role.

e Each totality constraint that (in part) concerned the marked role is dropped. Add a subset constraint for each totality constraint that is dropped, so that this SC together with one or more SCs that were created in step d form an equality constraint, which is equivalent to the totality constraint that was dropped.

f Process all the other constraints that concern the lexicalized role.

g As soon as a nominalized fact type plays no more roles as a result of steps 2a - 2f: cancel the nominalization and delete all object type expressions from the fact type. Remove any lost tuples. If the fact type has no fact type expression, then it is lost itself also; then delete it including all constraints that (in part) apply to its roles.

4.3 Reducing

In the final steps of deriving a relational schema from the lexicalized IGD, each fact type will lead to a separate table, with a separate column for each role. Sometimes there are one or more tables among these that can actually be done without, at the cost of a small loss of information. Translated back to IGDs in FCO-IM, this means that some fact types could still be deleted. The aim of *reducing* is: to delete the appropriate fact types, naturally only with permission from the database administrator. In this way the number of resulting tables becomes as small as possible. In this section, we will show how a *grouped and lexicalized IGD* (GL-IGD) is transformed by reducing into a *grouped, lexicalized and reduced IGD* (GLR-IGD).

In the example student-project case study, the GL-IGD in figure 4.8 cannot be reduced: the number of fact types is already minimal there. To still give a concrete example of reducing, we will consider a variant of the example student-project case study with an extra fact type expression in section 4.3.1, and give a general reducing algorithm in section 4.3.2.

4.3.1 Reducing in a Variant of the Student-Project Case Study

Let us assume in this section that existence postulating sentences for teachers have indeed been expressed during the verbalization of the concrete example documents, with fact expressions such as: “Teacher BAK works here.” and “Teacher LEK works here.” This leads in all IGDs to the fact type expression F8: “teacher <3> works here.”, which belongs to fact type Teacher (see figure 4.9, in which F8 can be seen in the GL-IGD). In this case, fact type Teacher has no lost tuples after lexicalizing, and therefore it is not lost itself either, so that the IGD in figure 4.9 is the final GL-IGD, including all six subset constraints.

In the final steps of deriving a relational schema, a separate table ‘Teacher’ will now arise for fact type Teacher, with just one column ‘teacher code’, populated with all the teacher codes from the population of role 3. This table can be regarded as a list of all supervisors and/or mentors. However, all these teacher codes can be found elsewhere as well, namely in the population of role 5 and/or role 8, because of the three subset constraints 1, 2 and 3, which together form an equality constraint that is equivalent to the dropped totality constraint 2 (see figure 4.5). We would therefore not loose any teacher codes if we would delete fact type Teacher.

We ask the database administrator whether he/she would object to deleting this table. The consequences of deleting fact type (table) Teacher are that we would lose the originally verbalized fact expressions of fact type expression F8 (the above mentioned small loss of information), and that generating a list of all recorded teacher codes becomes a little more difficult because it must be composed from the population of two other fact types (tables). The database administrator does not consider this to be a problem and prefers to delete fact type (table) Teacher. We therefore delete fact type Teacher from figure 4.9 with everything that belongs to it: fact type expression, population and constraints. With that, we have again obtained the GLR-IGD shown in figure 4.8.

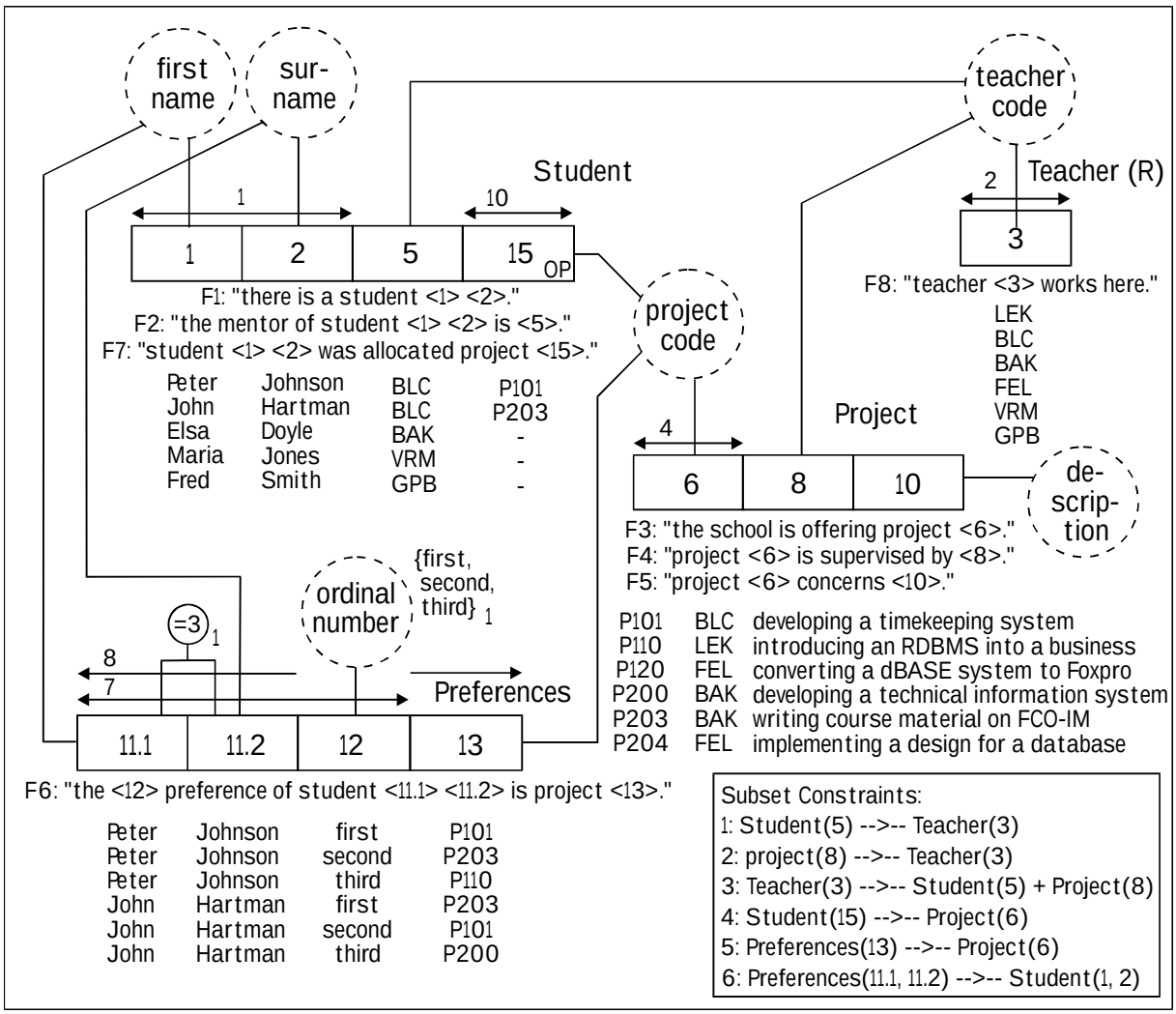


Figure 4.9: GL-IGD of a variant of the example case study

Apart from a possible small loss of information, namely that some (usually existence postulating, and therefore less important) fact type expressions cannot be regenerated anymore from a GLR-IGD, a GLR-IGD still models the same elementary fact expressions, and therefore the same information as the El-IGD, G-IGD and GL-IGD from which it arises.

4.3.2 Procedure for Reducing

We now give an algorithm for reducing. We do not claim to be complete (sometimes, in special circumstances concerning constraints, a fact type can be deleted although it does not satisfy the condition in step 1 below).

1 A fact type that satisfies the following condition can be considered for reducing: after lexicalizing, at least one equality constraint applies to all the roles of this fact type on the one hand, and on the other hand to roles of at least one other fact type. The FCO-IM tool marks fact types eligible for reducing by placing an '(R)' behind the fact type name (see figure 4.9).

2 Ask the databank administrator whether a fact type eligible for reducing can be deleted, showing him/her the consequences that some originally verbalized (usually existence postulating) sentences can then no longer be regenerated, and that answering certain questions becomes a little more complicated.

3 After receiving permission, delete the fact type including its fact type expression(s), its population and all constraints that concern (in part) the deleted roles.

Fact types eligible for reducing can also be recognized in a G-IGD from the fact that they are nominalized, that they have not absorbed any roles, and that there is at least one totality constraint that applies to one or more roles they play. These fact types can also be easily recognized in an El-IGD: they will not absorb any roles during grouping, they are nominalized and there is at least one totality constraint that applies to one or more roles they play.

4.4 Towards a Relational Schema

The final step in the derivation of a relational schema brings no further changes in the structure: each fact type in the GLR-IGD yields a separate table with a separate column for each role. A GLR-IGD is, as it were, a relational diagram in disguise. We can remove this 'disguise' by adapting the terminology to that of the Relational Model (for example, replacing the term 'role' with the term 'column' or 'attribute') and by generating meaningful table names and column names to replace the role numbers. Further, the Relational Model requires that we designate a primary key in each table, and that we give data types for all label types (domains). In this section, we will show how to transform a GLR-IGD into a relational schema. The relational schema still models the same elementary fact expressions, and therefore the same information as the GLR-IGD from which it arises. We therefore also record the meaning (the soft semantics) of the information in the tables along with the relational schema, in the form of elementary LTL-fact type expressions, in the words of the domain expert. After a short overview of the terminology change in section 4.4.1, we will first

illustrate the process with the example student-project case study in section 4.4.2, and give a general conversion algorithm in section 4.4.3.

4.4.1 Terminology Correspondence between FCO-IM and the Relational Model

In figure 4.10 we give a 'translation' of the terms from FCO-IM in terms from the Relational Model, and vice-versa.

FCO-IM	Relational Model
fact type	relation type, table type
role	attribute, column
mandatory role (= NN role)	mandatory attribute (column), or attribute (column) with not-null (NN) constraint
optional role (= OP role)	optional attribute (column)
label type, lexical object type	domain
label, value	value
tuple	tuple, row
identifier: NN role, or combination of NN roles, with a UC that concerns no other role(s)	candidate key
primary identifier	primary key
subset constraint (SC)	reference
SC to a primary identifier	foreign key reference
role, or combination of roles, with an SC to a primary identifier	foreign key
value constraint	domain constraint, domain integrity rule
other constraints (XC, CC, etc)	integrity rule

Figure 4.10: terminology correspondence FCO-IM / Relational Model

The list gives the most important correspondencies and is not exhaustive. The fact that a lexicalized IGD can be seen with a simple change in perspective as a relational schema (and vice versa) is one of the strongest points of FCO-IM. Because of this, it is possible to build CASE tools, such as the FCO-IM tool that goes with this book, with a repository that is generic for both FCO-IM and the Relational Model. This means that such a repository can contain information models (IGDs, relational schemas) from both modeling disciplines. This enables us to derive a redundancy free relational schema from an elementary IGD simply by doing a series of updates in the same repository. It is possible to extend the genericity in the same way over other modeling disciplines, such as Entity-Relationship Modeling and static object modeling.

We will not go into the meanings of the relational terms; we assume them to be understood. The word 'table' is often used (also in this text) where actually 'table type' or 'relationship type' is meant.

4.4.2 Converting in the Student-Project Case Study

The Relational Model requires that each relation has at least one candidate key, i.e. a minimal set of columns, all with a not null (NN) constraint, in which the value (combination) must be different in each tuple. In other words: there must be a uniqueness constraint on only these columns, and they must all be NN. (We shall see in section 7.2.2 that the NN requirement is actually too strong: identification of tuples can also be done with a weaker form of the NN requirement.) If there is more than one candidate key, then a primary key must be chosen from them (with only one candidate key it automatically becomes the primary key). In figure 4.11, we have indicated the primary key for each fact type (= table) by adding the letter 'p' to the corresponding uniqueness constraint, rather than to the roles themselves, for convenience. There is only one candidate key for fact type Student, because UC 10 is on an optional role. That is also in agreement with the E1-IGD and the G-IGD, in which we can tell from the object type expression for Student that the identifier for Student is the role combination 1 + 2. Project also has only one candidate key, but Preferences has two. We ask the School's Project Coordinator which of these two keys should become the primary key. He indicates the roles under UC 7 as the primary identifier, because he often wants the answer to a question such as: "What is the first preference of student Peter Johnson?" but almost never to a question such as: "What is the preference number of student Peter Johnson for project P203?". The roles under UC 8 are now called an *alternative key*, but we do not indicate that explicitly in the IGD.

We must choose meaningful table names and column names to get a comprehensible relational diagram. The Relational Model requires unique table names and column names that are unique within the table they sit in. The choice of these names is free in principle, but it turns out to be very desirable in practice to generate the names using a fixed convention. One useful convention is to give each table the name of the corresponding fact type and every column the name of the domain that applies to it (= the name of the label type that plays the role).

We will use another convention in this text, however, because in our opinion it gives clearer column names and is less dependent on the domains. This convention is implemented in the FCO-IM tool as the default (the other convention is also available as an option). In this convention, we start from an elementary IGD (E1-IGD) and transform it into a lexicalized IGD (L-IGD), possibly grouping and/or reducing it as well, but these processes are not considered here. A table gets the name of the corresponding fact type in the L-IGD. A column gets the name of the object type (lexical or non-lexical) that *originally* played the corresponding role in the L-IGD. The word 'originally' is to be interpreted as follows. If a role in the L-IGD was already lexical in the E1-IGD, then the column gets the name of the label type. If a role in the L-IGD arose from lexicalizing a non-lexical role in the E1-IGD without splitting it, then the column gets the name of the object type that played the role in the E1-IGD. If a role in the L-IGD arose from splitting a role in the E1-IGD at least once during lexicalizing, then the column gets the name of the object type (lexical or non-lexical) that played the role just after the final splitting. In the last case, it is best to imagine lexicalizing just the one role in the E1-IGD from which the role concerned in the L-IGD eventually arises.

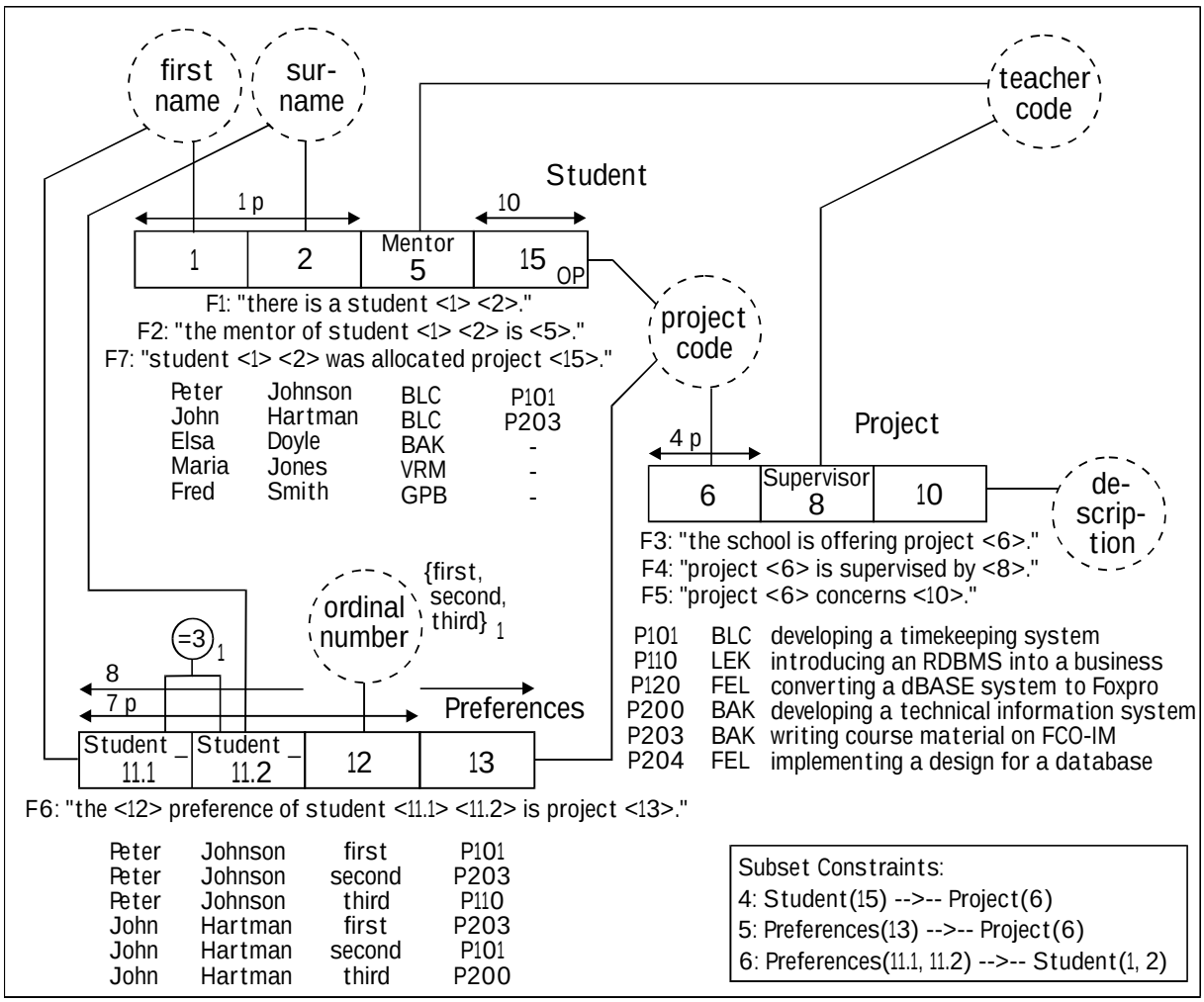


Figure 4.11: GLR-IGD with primary UCs and role fixes

The relational schema that follows from the GLR-IGD in figure 4.11 using the convention described above is shown in figure 4.12. The table names are the same as the fact type names. The columns corresponding to roles 1, 2, 6, 10 and 12 have received the names of the label types that played these roles already in the E1-IGD. The columns corresponding to roles 13 and 15 have received the name of object type Project, which played these roles in the E1-IGD: during lexicalizing they were not split.

Relational Schema Student-Project Case Study:

Student

first name (first name) NN	surname (surname) NN	Mentor (teacher code) NN	Project (project code) OP
← PK →			← →
Peter	Johnson	BLC	P101
John	Hartman	BLC	P203
Elsa	Doyle	BAK	-
Maria	Jones	VRM	-
Fred	Smith	GPB	-

Domains:
first name = char(12)
surname = char(30)
teacher code = char(3)
project code = char(4)
description = char(256)
ordinal number = char(6)

Domain Constraint:
ordinal number = {first, second, third}

F1: "There is a student <first name> <surname>."
F2: "The mentor of student <first name> <surname> is <Mentor>."
F7: "Student <first name> <surname> was allocated project <Project>."

Project

project code (project code) NN ← PK →	Supervisor (teacher code) NN	description (description) NN
P101	BLC	developing a timekeeping system
P110	LEK	introducing an RDBMS into a business
P120	FEL	converting a dBASE system to Foxpro
P200	BAK	developing a technical information system
P203	BAK	writing course material on FCO-IM
P204	FEL	implementing a design for a database

F3: "The school is offering project <project code>."
F4: "Project <project code> is supervised by <Supervisor>."
F5: "Project <project code> concerns <description>."

=3

Preferences

Student first name (first name) NN	Student surname (surname) NN	ordinal number (ordinal number) NN	Project (project code) NN
← PK →			
Peter	Johnson	first	P101
Peter	Johnson	second	P203
Peter	Johnson	third	P110
John	Hartman	first	P203
John	Hartman	second	P101
John	Hartman	third	P200

Foreign Keys:
1: Student(Project) -->--
Project(project code)
2: Preferences(Project) -->--
Project(project code)
3: Preferences(Student first name,
Student surname) -->--
Student(first name, surname)

F6: "The <ordinal number> preference of student
<Student first name> <Student surname> is project <Project>."

Figure 4.12: logical relational schema

The columns corresponding to roles 5 and 8 would in the same way both be called ‘Teacher’. However, in roles 5 and 8 we have entered the texts ‘Mentor’ and ‘Supervisor’ respectively, see figure 4.11. These are so-called *replace fixes*: texts to replace the names that are generated according to the convention.

In roles 11.1 and 11.2, we have also entered texts: ‘Student _’. These are so-called *prefixes*: texts to be appended at the front of the names that are generated according to the convention. The underscore ‘_’ is there to show where the convention name is to be put. These names for the columns corresponding to roles 11.1 and 11.2 are ‘first name’ and ‘surname’ respectively, because role 11 from the El-IGD was split once during lexicalizing and just after that the new roles 11.1 and 11.2 were played respectively by the (lexical) object types ‘first name’ and ‘surname’. Adding the prefixes results in the final names ‘Student first name’ and ‘Student surname’, see figure 4.12.

There are also *postfixes*, texts to be appended at the end of the names that are generated according to the convention, and which therefore have an underscore at the front, to show where the convention name is to be put. This mechanism of prefixes, postfixes and replace fixes to alter the column names generated according to the convention gives us enough flexibility to avoid rigidity in applying the convention and to obtain extra clear names if desired, while the basis remains a uniform convention. In the FCO-IM tool it is also possible to give an *alias* (synonym) for a fact type to replace the table name from the convention. Finally, we remark that the analyst is required to give a prefix, postfix or replace fix if two or more roles in the same fact type are played by the same object type, because otherwise two or more identical column names are generated in one table.

Every column in figure 4.12 lists, between brackets, the name of the corresponding domain (= the label type the corresponding roles play in the GLR-IGD). A list with data types for each domain is supplied as well. In order to remain independent of any implementation, we only use the following data types: char(n) for a text of a maximum of n characters, numeric(n) for an integral amount of n digits and numeric(n,m) for a decimal amount with n digits before the comma and m digits after the comma. The FCO-IM tool automatically derives a proposal for data types from the example population, which the analyst or implementer can change in consultation with the domain expert. Any domain constraint mostly follows directly from value constraints.

Non-optional columns are marked ‘NN’, an abbreviation for ‘not null’. In these columns no null values are allowed. Optional columns are marked ‘OP’, like the corresponding roles. We indicate the primary key with the abbreviation ‘PK’ in the corresponding uniqueness arrow.

References in the relational schema follow directly from subset constraints in the L-IGD. Often a reference points to a column (or column combination) that is the primary key of a table. The column (combination) from which such a reference departs is called a foreign key in relational jargon, and the reference itself is called a foreign key reference. In the relational schema in figure 4.12, all references coincidentally are foreign key references, but in general references can occur that do not point to a primary key.

In figure 4.12, the cardinality constraint from figure 4.11 is taken over, although it is not customary to display cardinality constraints graphically in relational schemas. We could instead have written down a special integrity constraint textually: it must be true for the population of table Preferences that each value combination in columns 'Student first name' and 'Student surname' occurs exactly three times. Together with the domain constraint: ordinal number = {first, second, third} and the uniqueness constraint from the primary key, this integrity rule ensures that for each student all three ordinal number occur once each.

We also include the fact type expressions from figure 4.11 in the relational schema (in which we replace the role numbers with the column names), so that we have also recorded the *meaning* (= *soft semantics*) of the information for the users of the information base. Besides the soft semantics we also distinguish the *hard semantics* that concern the structure aspects: table schemas, domains, integrity rules. Finally we include an example population in order to keep in touch with the concrete level. The population is even a proper one here: it satisfies all the integrity rules. In general however, it is impracticable to give a proper example population. The FCO-IM tool will therefore allow non-proper populations as well.

4.4.3 Procedure for Converting

We now give an algorithm for converting a lexicalized IGD into a relational schema, without claiming completeness.

1 Use the terminology transformation from section 4.4.1.

2 Each fact type becomes a separate table and each role a separate column. Names for the tables and columns follow from the convention discussed in section 4.4.2. If desired, modify the names generated according to the convention using an alias or a prefix, postfix or replace fix.

3 Choose a primary key for each table.

4 Specify a domain for each column, equal to the label type that plays the corresponding role. Specify a data type for each domain. Convert each value constraint into a domain constraint.

5 Convert all subset constraints into references. It is customary to list the foreign key references separately.

6 Convert all remaining constraints from the IGD into integrity rules. There is no general algorithm for this, but eventually we must be able to find every constraint from the elementary IGD again as (part of) an integrity rule in the relational schema.

4.5 Generating DDL

The relational schema derived in section 4.4 is a schema on the *logical* level: it is purely drawn up in terms of the Relational Model. This means that we do not consider the limitations, extra features or other peculiarities of a specific software package, such as a relational database management system, which will be used for the implementation: the diagram in figure 4.12 is therefore implementation independent. From this logical relational schema, it is easy to generate DDL (data definition language) instructions automatically for creating the tables and the integrity rules for an arbitrary implementation platform. So the FCO-IM tool can do this as well. There are, however, many Relational Data Base Management Systems (RDBMSs) on the market. That is why we use a DDL syntax in the example in figure 4.13 that is not intended for any specific RDBMS. Adaptation to a specific RDBMS is, however, not difficult.

It can be seen in figure 4.13 that we create a new schema with the CREATE SCHEMA instruction. In the new schema, we first define the domains with CREATE DOMAIN instructions. We realize the domain constraint on domain 'ordinal number' through the CHECK option. Then we use CREATE TABLE instructions to make the tables, with their primary key and other uniqueness constraints (if any). There are such a non-key uniqueness constraints in table Student on an optional column and in table Preferences on a combination of NN columns.

To avoid sequence problems during execution, we accommodate the remaining integrity rules in separate instructions after creating all the tables. So we use the ALTER TABLE instruction with the ADD FOREIGN KEY option for the foreign key references. We could use an ALTER TABLE instruction with CHECK option for any other references as well, but these do not occur here. With such instructions, by far the greatest part of the DDL is automatically generated. We do not automatically generate more complex integrity rules, such as the cardinality constraint in figure 4.12. It is not practicable to cover all possibilities and we also come across big differences in syntax and in power between different RDBMSs. The implementers must see to it themselves that the remaining integrity rules are enforced in some way, for instance through more extensive check options, stored procedures, triggers and so on.

```
CREATE SCHEMA              STUDENT-PROJECT CASE STUDY

CREATE DOMAIN   FIRST NAME          AS  CHAR(12)
CREATE DOMAIN   SURNAME             AS  CHAR(30)
CREATE DOMAIN   TEACHER CODE        AS  CHAR(3)
CREATE DOMAIN   PROJECT CODE        AS  CHAR(4)
CREATE DOMAIN   DESCRIPTION         AS  CHAR(256)
CREATE DOMAIN   ORDINAL NUMBER      AS  CHAR(6),
                CHECK   (VALUE      IN  ('FIRST', 'SECOND', 'THIRD'))

CREATE TABLE              STUDENT
(FIRST NAME             FIRST NAME          NOT NULL,
 SURNAME                SURNAME             NOT NULL,
 MENTOR                 TEACHER CODE        NOT NULL,
 PROJECT                PROJECT CODE                ,
 PRIMARY KEY            (FIRST NAME, SURNAME)       ,
 UNIQUE                 (PROJECT)                   )

CREATE TABLE              PROJECT
(PROJECT CODE           PROJECT CODE        NOT NULL,
 SUPERVISOR             TEACHER CODE        NOT NULL,
 DESCRIPTION            DESCRIPTION         NOT NULL,
 PRIMARY KEY            (PROJECT CODE)              )

CREATE TABLE              PREFERENCES
(STUDENT FIRST NAME     FIRST NAME          NOT NULL,
 STUDENT SURNAME        SURNAME             NOT NULL,
 ORDINAL NUMBER         ORDINAL NUMBER      NOT NULL,
 PROJECT                PROJECT CODE        NOT NULL,
 PRIMARY KEY            (STUDENT FIRST NAME, STUDENT SURNAME,
                         ORDINAL NUMBER)            ,
 UNIQUE                 (STUDENT FIRST NAME, STUDENT SURNAME,  PROJECT))

ALTER TABLE             STUDENT
ADD FOREIGN KEY         (PROJECT)   REFERENCES      PROJECT(PROJECT CODE)

ALTER TABLE             PREFERENCES
ADD FOREIGN KEY         (PROJECT)   REFERENCES      PROJECT(PROJECT CODE)

ALTER TABLE             PREFERENCES
ADD FOREIGN KEY         (STUDENT FIRST NAME , STUDENT SURNAME)
                        REFERENCES  STUDENT (FIRST NAME, SURNAME)
```

Figure 4.13: automatically generated DDL

4.6 Final Remarks

1 It is not necessary to perform grouping to derive a relational schema: just lexicalizing suffices. The result would then be a relational diagram with almost always more tables and more references than when grouping had been done. In the example student-project case study, we would lexicalize the E1-IGD from figure 3.25; we advise the reader to do this as an exercise and to verify the following comments. All fact types remain except fact type Teacher, which turns out to be lost again and is deleted. Fact type Project is then eligible for reducing, in which case we would lose the fact expressions of fact type expression F3. If the database administrator does not object to this, then six fact types, with all roles now lexical, remain (otherwise seven), which are all converted into tables, with many references between them. The same information is still modeled thus, albeit in a greater number of tables than necessary, with a great number of integrity rules that must be enforced separately. Queries are more complex as well and require more joins. Therefore, in general the aim is to achieve a minimum number of tables.

2 In spite of remark 1 above, it is sometimes desirable to exclude one or more fact types from the grouping process although they are eligible for grouping away. A separate table for each such fact type then arises. This is why the FCO-IM tool gives a *grouping proposal*, which lists all the roles that satisfy the grouping conditions. The user can then decide to remove the 'G' mark from one or more roles in the proposal. In this way we can control the grouping process completely. (Extra roles could also be marked for grouping by placing a 'G' mark in a role that does not satisfy the grouping conditions. This always introduces redundancy into the tables (the readers can verify this themselves), so protection routines will have to be written to prevent the redundancy from causing problems. That is why this is generally considered to be undesirable. See, however, the postal code example in chapter 5. In the FCO-IM tool it is possible to override the grouping proposal, both by removing the 'G' mark from a role marked for grouping so it will be retained, as by adding a 'G' mark to a role not marked for grouping so it will be grouped away after all.)

3 If the tests from section 3.3 were carried out well, then the relational schema is in fifth normal form, whether or not grouping is done, and therefore it is redundancy free. Grouping practically always results in the minimum number of tables (occasionally a special configuration of constraints will enable us to reduce the number of tables further without introducing redundancy). A relational schema in fifth normal form with a minimum number of tables is said to be in *optimal normal form*. In chapter 5, we will explain that this minimum number of tables depends on the semantically equivalent modeling chosen (see section 2.7). In other words: carrying out a semantically equivalent transformation could lead to a different set of tables, possibly also different in number. 'Minimum number of tables' then applies only relative to a certain E1-IGD.

4 The FCO-IM tool generates a reducing proposal, analogous to the grouping proposal in remark 2, so that we can also control the reducing process completely.

5 Eventually all the constraints in the E1-IGD must correspond with integrity rules in the relational schema, and vice versa. The transformations from constraints to integrity rules are often complex and also often concern different constraints/rules at the same time. A practical aid to ensure that no constraint will escape implementation is a correspondence table. We give an example of such a table in figure 4.14 (in which 'PK' stands for primary key and 'FK' for foreign key), see also figures 3.25 and 4.12.

We advise to always make such a table, especially one as is shown here, between the E1-IGD and the logical relational schema, but also between the logical relational schema and the implementation design. Integrity rules will often have to be programmed separately, so an overview of which integrity rules are enforced in which way is important. The use of correspondence tables can also be valuable in the intermediate steps grouping, lexicalizing and reducing.

El-IGD	Logical Relational Schema
UC 1	PK of Student
UC 2	cancelled during lexicalizing (lost FT Teacher)
UC 3	cancelled during grouping
UC 4	PK of Project
UC 5	cancelled during grouping
UC 6	cancelled during grouping
UC 7	PK of Preferences
UC 8	UNIQUE-rule on Preferences
UC 9	cancelled during grouping
UC 10	UNIQUE-rule on Student
TC 1	NN on Student(Mentor)
TC 2	SC6, cancelled during lexicalizing (lost FT Teacher)
TC 3	NN on Project(Supervisor)
TC 4	NN on Project(description)
absence of a TC on role 14	OP on Student(Project)
NN on remaining roles	NN on remaining columns
CC 1	CC on Preferences(Student first name, Student surname)
lexicalizing role 15	FK Student(Project)
lexicalizing role 13	FK Preferences(Project)
lexicalizing role 11	FK Preferences(Student first name, Student surname)

Figure 4.14: correspondence between constraints and integrity rules

5

Various Modeling Issues

In the chapters 2, 3 and 4, we showed how to build up an information grammar diagram (IGD) and how to derive a logical relational schema from it. In this chapter, we will go into a number of modeling issues connected with everything discussed earlier. In section 5.1, we will elaborate on semantically equivalent transformations and their consequences for the resulting relational schemas. In section 5.2, unary fact types and how to handle them in the derivation of a relational schema will be treated. Finally in section 5.3, we will discuss Dutch addresses with postal codes, as an example of a particularly difficult modeling case (in terms of redundancy free modeling and the handling of rare exceptions to an otherwise simple structure), and to illustrate the controlled introduction of redundancy by denormalizing a relational schema.

5.1 Semantically Equivalent Transformations

Two different IGDs 1 and 2 that both model the same communication are called *semantically equivalent*. This means that the two IGDs contain the same elementary fact type expressions (FTEs), and that every population that is allowed according to the constraints of IGD 1, is also allowed according to the constraints of IGD 2, and vice versa. Semantically equivalent IGDs can arise from different choices during classification and qualification of the same collection of elementary fact expressions, see section 2.7. Semantically equivalent IGDs can therefore always generate exactly the same sentences, so they make no difference for the validation.

Quite often (but not always), two semantically equivalent IGDs will lead to differences in the relational database schemas that we derive from them with the GLR-algorithm (see chapter 4), such as a different number of tables and/or differences in the table structures (other columns, other keys and so on). The database administrator might prefer one of the alternative schemas, for example because of differences in the performance of frequently used transactions, or because standard information needs are simpler to satisfy (fewer joins). Semantically equivalent transformations can therefore be used to optimize the database structure without introducing redundancy. In this section, we will discuss the consequences of the two most important semantically equivalent transformations: the *object type - fact type transformation* and the *nominalization - denominalization transformation*.

By the way, grouping (section 4.1) and lexicalizing (section 4.2) do not change the modeled communication either. These are therefore semantically equivalent transformations as well. The same applies for reducing (section 4.3), apart from the small loss of information there.

5.1.1 Object Type - Fact Type Transformations

We will first carry out an object type - fact type transformation in the example student-project case study. After that we will derive a relational schema from the transformed IGD, and compare this with the relational schema that we derived from the original IGD.

5.1.1.1 Transformation of the Student-Project Case Study

In section 2.3, the content of the example documents was verbalized in a collection of elementary fact expressions (sentences). We classified and qualified these fact expressions in section 2.4 (figure 2.7). We drew the corresponding IGD, and in chapter 3 we added all the constraints. The final IGD is shown in figure 3.25. With the GLR algorithm we derived a logical relational schema, which is in figure 4.12.

In section 2.4.1, we assigned all the fact expressions 46 - 66 below to one class, and called the corresponding fact type: 'Preferences'.

46) "The first preference of student Peter Johnson is project P101."
47) " second " " " Peter Johnson " " P203
48) " third " " " Peter Johnson " " P110
49) " first " " " John Hartman " " P203
50) " second " " " John Hartman " " P101
51) " third " " " John Hartman " " P200
.....
64) " first " " " Tom Dakota " " P201
65) " second " " " Tom Dakota " " P101
66) " third " " " Tom Dakota " " P110

Now let us classify and qualify these fact expressions in a different way. Because it is clear from the starting document that only the first, second, and third preferences are important, we can divide the fact expressions into three different classes:

First Preference:

46) "The first preference of student Peter Johnson is project P101."
49) " " " " " John Hartman " " P203
.....
64) " " " " " Tom Dakota " " P201

Second Preference:
47) “The second preference of student Peter Johnson is project P203.”
50) " " " " " John Hartman " " P101
.....
65) " " " " " Tom Dakota " " P101

Third Preference:
48) “The third preference of student Peter Johnson is project P110.”
51) " " " " " John Hartman " " P200
.....
66) " " " " " Tom Dakota " " P110

The further classification and qualification of these fact expressions is shown in figure 5.1, in which, for clarity, the old analysis is shown as well. The analysis of the other fact expressions does not change.

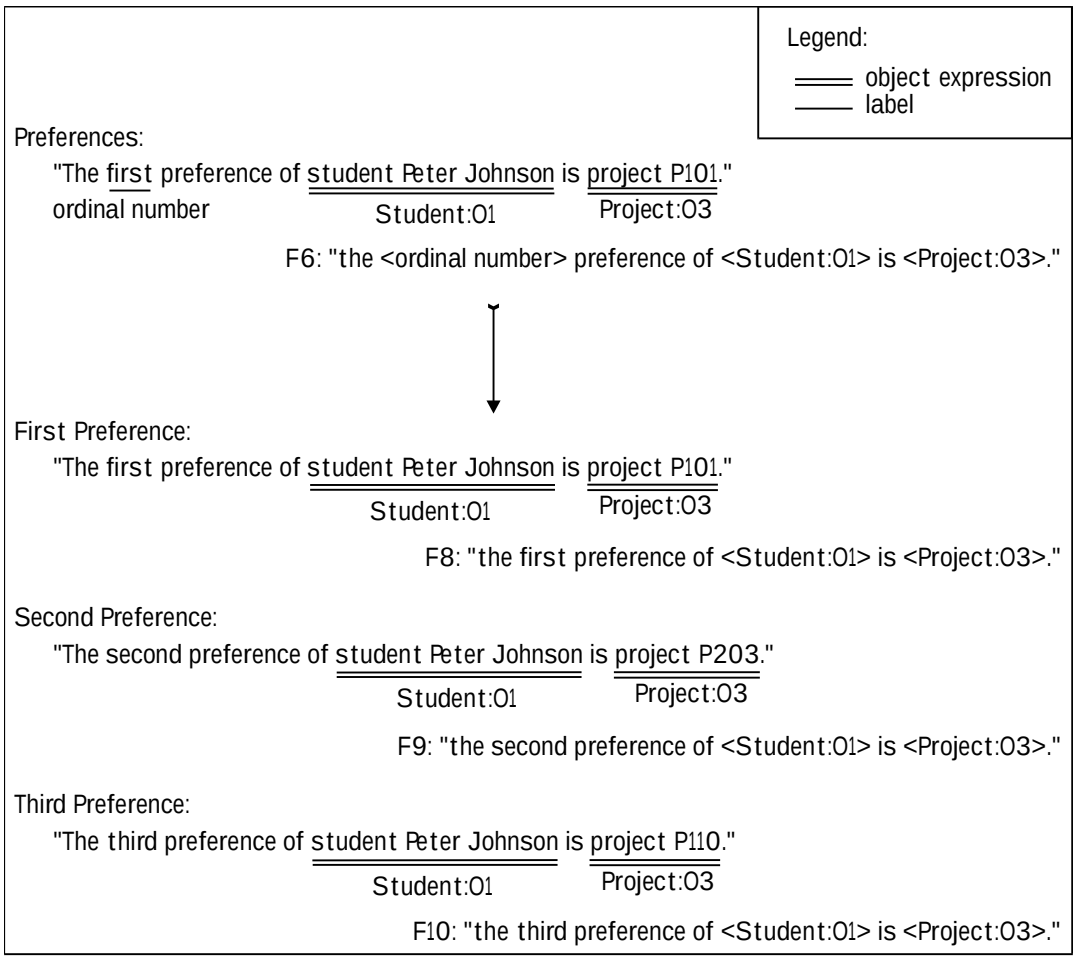


Figure 5.1: another classification and qualification of fact expressions 46 - 66

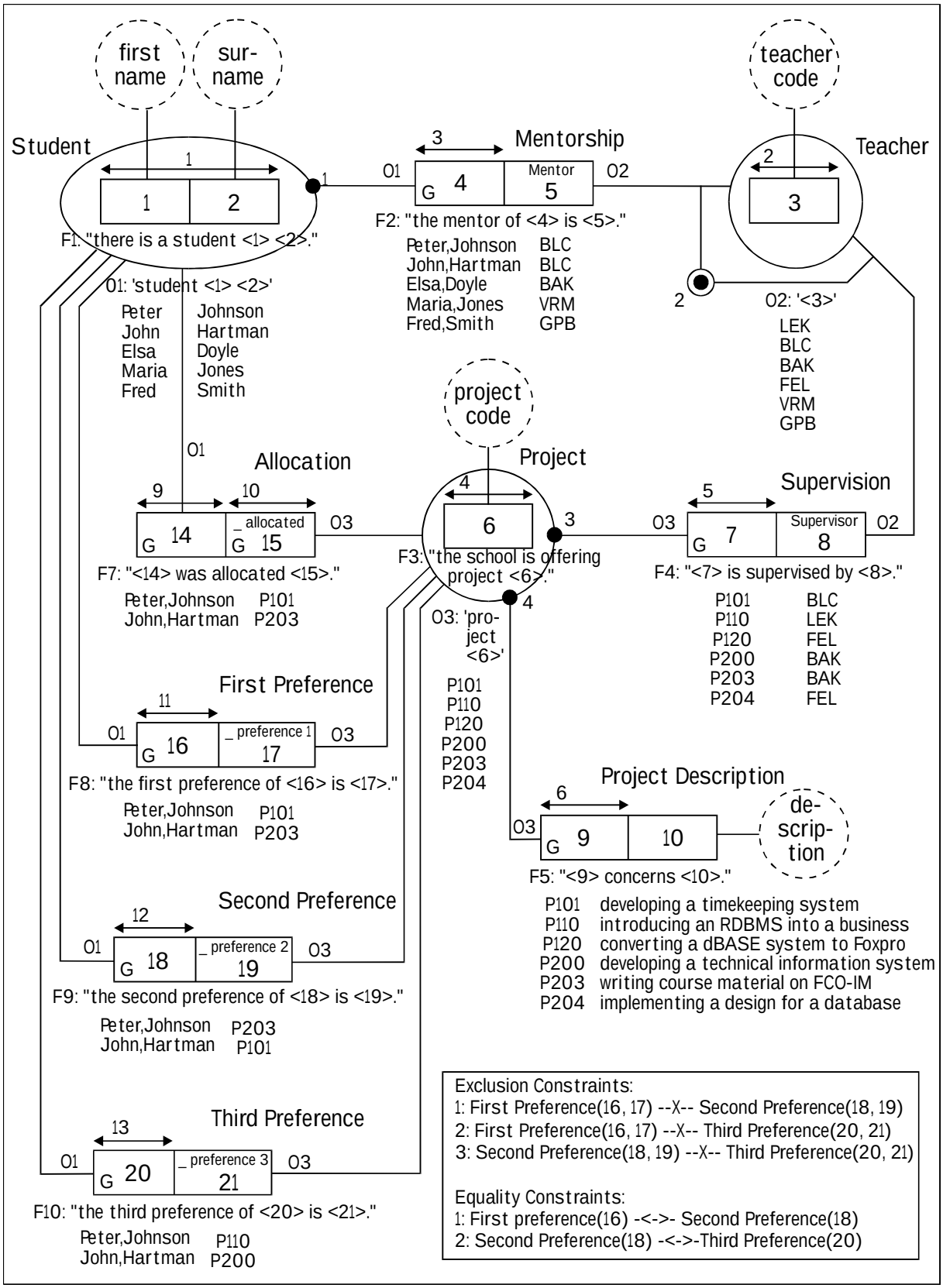


Figure 5.2: El-IGD after object type - fact type transformation

When we draw the diagram and add the relevant constraints we get the IGD in figure 5.2. This IGD is semantically equivalent with the IGD in figure 3.25. The label type 'ordinal number' is now gone and instead of the ternary fact type Preferences there now are three binary fact types: First Preference, Second Preference, and Third Preference.

All possible values from the old object type 'ordinal number' (listed in value constraint 1 in figure 3.25) are now found as a part of the fact type expressions of the new fact types. An object type has been 'turned into' fact types, as it were, hence the name object type - fact type transformation.

No totality constraints apply to (combinations of) roles 16, 17, 18, 19, 20 and 21 in figure 5.2, for the same reason that no TCs apply to roles 11 and 13 in figure 3.25 (see section 3.4).

The three project preferences of the students must all be different, see the starting document. This is expressed in the IGD of figure 5.2 by exclusion constraints (XCs) 1, 2, and 3, which enforce that the combination of a student and a project in the population can only occur in one of the three fact types First Preference, Second Preference and Third Preference. These exclusion constraints are therefore the equivalent of uniqueness constraint 8 in the IGD of figure 3.25 (see also section 3.2.1, last paragraph).

If a student makes his or her preferences known, then none of the three choices can be missing. This is expressed by the two equality constraints (ECs) 1 and 2 in figure 5.2, which enforce that the populations of roles 16, 18 and 20 are always equal to each other. These equality constraints are therefore the equivalent of cardinality constraint 1 in the IGD of figure 3.25 (also see section 3.7).

This object type - fact type transformation of the old ternary fact type Preferences illustrates the following generally valid point. An object type - fact type transformation can only be carried out (in the fact type direction) for a fact type with a role that can only contain a value from a predetermined fixed set of values, such as {M, V}, {jan, feb, ….., dec}, {-, 0, + } and so on. This role could be played by a label type with a value constraint, as in section 5.1.1.1, but it could also be played by an object type (nominalized fact type) such as Gender, which is identified by a label type 'gender code' with a value constraint {man, woman}. Instead of this one fact type containing this role, there arise as many fact types containing one role less as there are possible values: one fact type for each possible value. From one binary fact type with a role played by Gender (with for example a fact type expression such as: F1 "<Student:O1> is a <Gender:O2>."), two unary fact types could be formed, with fact type expressions such as F11: "<Student:O1> is a man." and F12: "<Student:O1> is a woman.".

For an application of the object type - fact type transformation in the object type direction, see section 5.2.2.

5.1.1.2 Logical Relational Schema after Transformation

We will now derive a logical relational schema from the IGD in figure 5.2 and compare the derivation with the one in chapter 4. In figure 5.2, the roles have already been marked for grouping, and suitable postfixes and replace fixes have been added as well. Compared to figure 4.1, three extra roles have now been marked: 16, 18 and 20.

The IGD after grouping is shown in figure 5.3. Roles 17, 19 and 21 have been absorbed by fact type Student after deleting roles 16, 18 and 20. These three roles are now optional, because a single role totality constraint applied to none of the deleted roles. On account of equality constraints 1 and 2 from figure 5.2, roles 17, 19 and 21 must either all three have a value, or all three have a null value in each tuple after the transformation. We formulate this in a non-standard constraint C1, which now replaces both equality constraints.

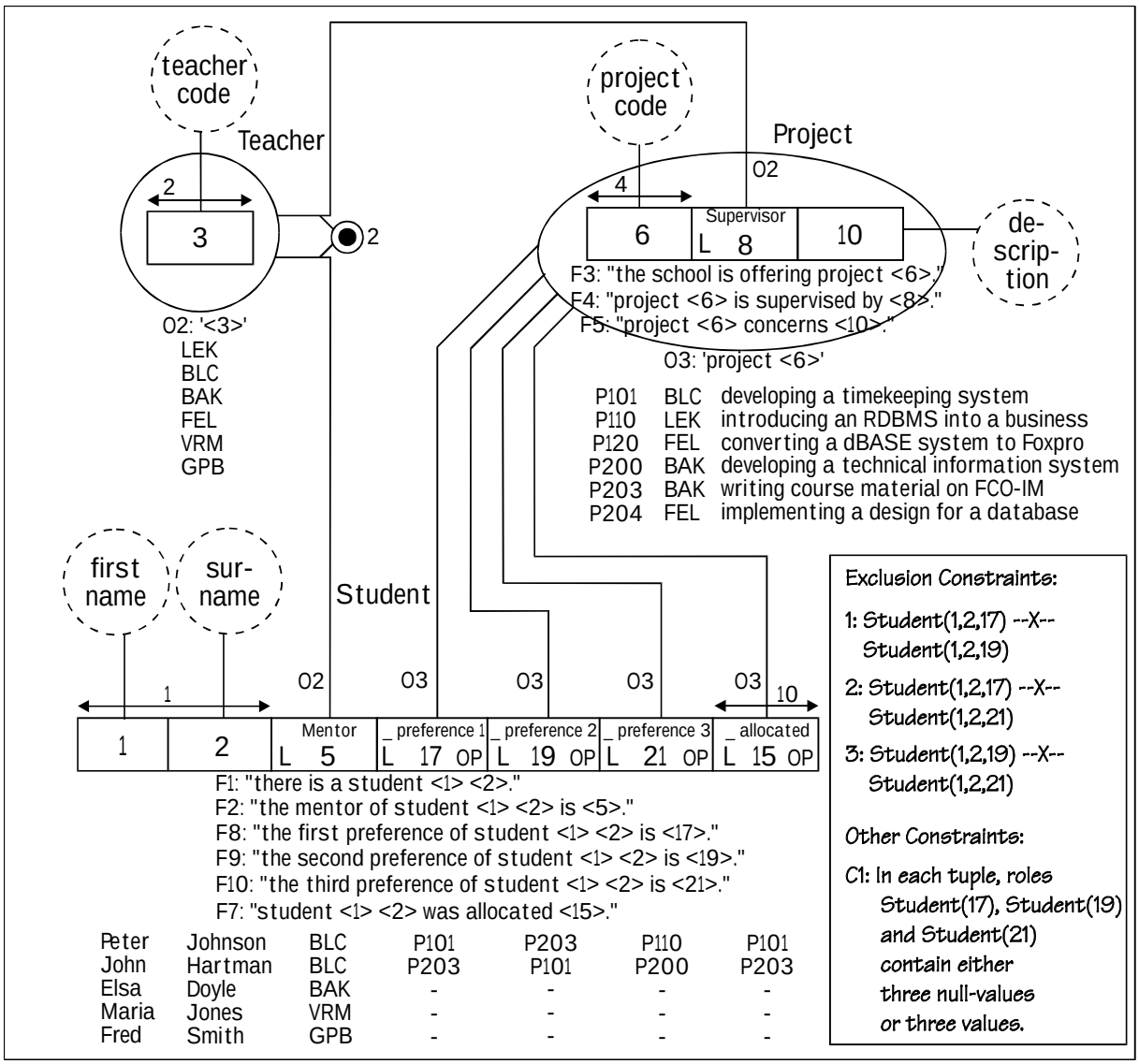


Figure 5.3: G-IGD after object type - fact type transformation

Exclusion constraints 1, 2 and 3 from figure 5.2 must be changed as well, since roles 16, 18 and 20, to which they both applied, were deleted. We can tell from object type expression O1 that the role combination 1 + 2 comes in the place of the deleted roles, so the new XCs now concern combinations of three roles. The effect of the three XCs is now, however, that the three project preferences must be different *in each tuple.* This last requirement is easier to enforce than the three separate XCs. It is often possible after transforming an IGD to formulate other equivalent constraints, but there is no general rule for this. We will replace the three XCs with a non-standard constraint only in the relational schema, because we want to demonstrate the transformation of the old XCs here, which can de done automatically.

The roles to be lexicalized have already been marked in figure 5.3. The resulting IGD is in figure 5.4. Fact type Teacher turns out to be lost again and is deleted (see section 4.2.1). Four subset constraints arise. There is nothing to reduce, so figure 5.4 contains the final GLR-IGD.

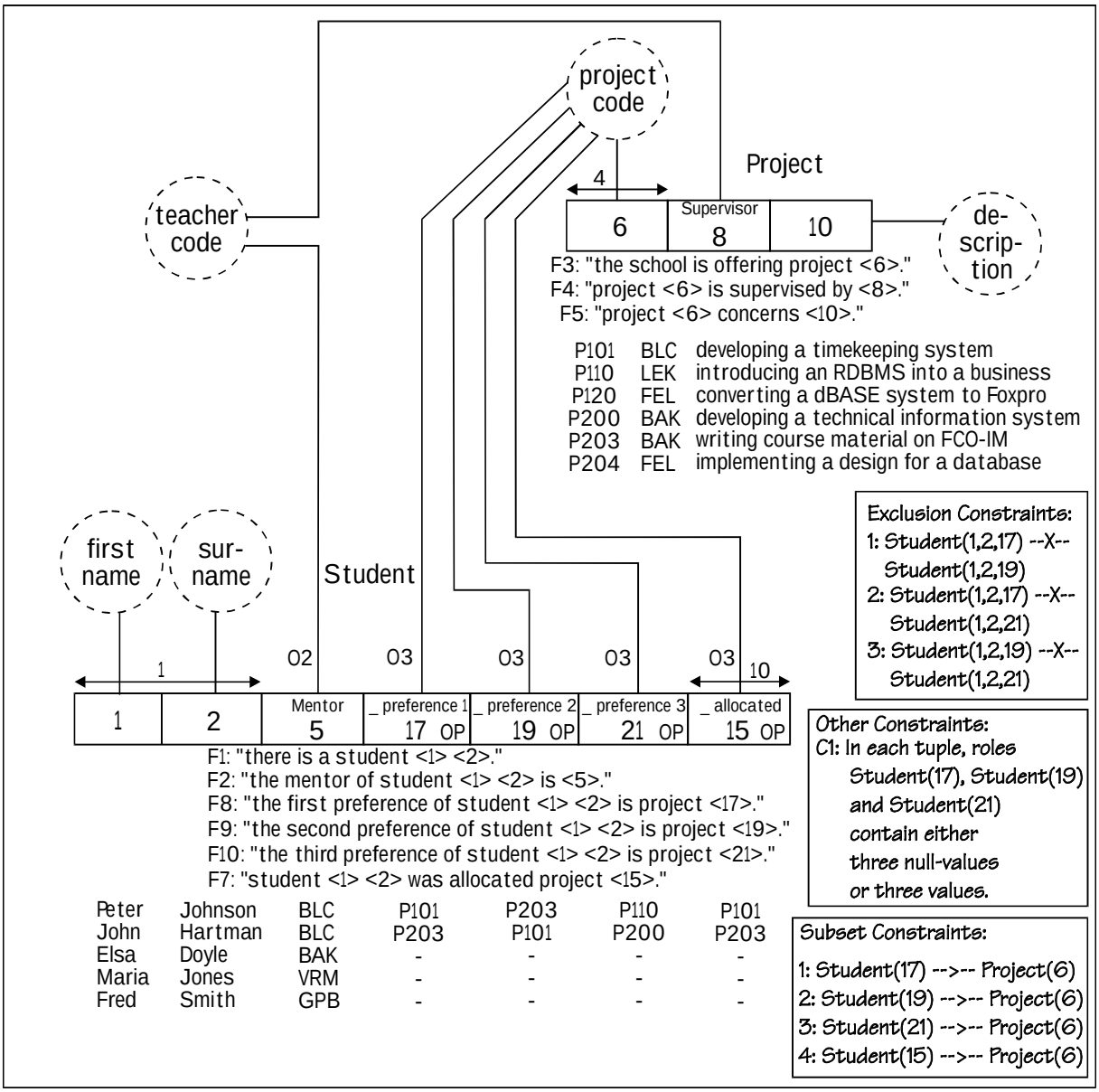


Figure 5.4: GLR-IGD after object type - fact type transformation

Finally, the relational schema is shown in figure 5.5. In this schema, we have replaced the three exclusion constraints with an equivalent non-standard integrity rule that is easier to enforce.

Relational Schema Student-Project Case Study after OT / FT Transformation:

Student

first name (first name) NN	surname (surname) NN ← PK →	Mentor (teacher code) NN	Project preference 1 (project code) OP	Project preference 2 (project code) OP	Project preference 3 (project code) OP	Project allocated (project code) OP ← →
Peter	Johnson	BLC	P101	P203	P110	P101
John	Hartman	BLC	P203	P101	P200	P203
Elsa	Doyle	BAK	-	-	-	-
Maria	Jones	VRM	-	-	-	-
Fred	Smith	GPB	-	-	-	-

F1: "There is a student <first name> <surname>."
F2: "The mentor of student <first name> <surname> is <Mentor>."
F8: "The first preference of student <first name> <surname> is project <Project preference 1>."
F9: "The second preference of student <first name> <surname> is project <Project preference 2>."
F10: "The third preference of student <first name> <surname> is project <Project preference 3>."
F7: "Student <first name> <surname> was allocated project <Project allocated>."

Project

project code (project code) NN ← PK →	Supervisor (teacher code) NN	description (description) NN
P101	BLC	developing a timekeeping system
P110	LEK	introducing an RDBMS into a business
P120	FEL	converting a dBASE system to Foxpro
P200	BAK	developing a technical information system
P203	BAK	writing course material on FCO-IM
P204	FEL	implementing a design for a database

F3: "The school is offering project <project code>."
F4: "Project <project code>is supervised by <Supervisor>."
F5: "Project <project code> concerns <description>."

Domains:
first name = char(12)
surname = char(30)
teacher code = char(3)
project code = char(4)
description = char(256)

Foreign Keys:
1: Student(Project preference 1)
-->-- Project(project code)
2: Student(Project preference 2)
-->-- Project(project code)
3: Student(Project preference 3)
-->-- Project(project code)
4: Student(Project allocated)
-->-- Project(project code)

Other Integrity Rules:
1: In each tuple, the values in
Student(Project preference 1),
Student(Project preference 2) and
Student(Project preference 3)
must be all different.
2: In each tuple, columns
Student(Project preference 1),
Student(Project preference 2) and
Student(Project preference 3)
must contain either three null-values
or three values.

Figure 5.5: logical relational schema after object type - fact type transformation

The relational database schema in figure 5.5 has one table less than the relational database schema in figure 4.12: two tables instead of three. Table Student has, however, got three extra optional columns.

As we have already pointed out in final remark 3 from section 4.6, the GLR-algorithm leads to a redundancy free database schema with a minimum number of tables, if the El-IGD is fully grouped. Such a database is said to be in *optimal normal form*. We have just shown here, that different semantically equivalent El-IGDs can produce a different minimum number of tables, although exactly the same elementary facts are modeled. The 'optimality' is therefore certainly relative to the El-IGD, and not an absolute property.

It cannot be said which of these two database schemas is to be preferred in general. Some database administrators would perhaps prefer the database schema of figure 4.12 (with three tables) over that of figure 5.5 (with two tables), for instance because it is more convenient for queries (for example SQL-SELECTs) to have all the project codes of student preferences in one column (column Project from table Preferences in figure 4.12) than in three columns (Project preference 1, Project preference 2 and Project preference 3 from table Student in figure 5.5). Furthermore, there are no null values in table Preferences from figure 4.12, whereas null values do occur in the three columns concerned in table Student from figure 5.5 for all students who have given no project preferences. Other database administrators would perhaps prefer the database schema of figure 5.5, because they do not always have to look in two tables when they are assigning projects to students and because the performance is much faster there. So from an operational point of view, 'optimality' is a relative property as well. The relative weight of all these considerations can only be established for each implementation separately, by analyzing how often which transaction are to be carried out and how long the processing time per transaction is. The Schema from figure 4.12 is likely to be preferred if there are relatively few tuples per table, and if there are relatively few transactions that concern more than one table, whereas in the opposite situation the schema from figure 5.5 is likely to be favored.

5.1.2 Nominalization - Denominalization Transformations

A second important semantically equivalent transformation is the nominalization - denominalization transformation. We will mostly want to carry this out in one direction only: nominalize what was not nominalized yet (see also section 3.3.2). We will also just call this *nominalization* for short.

We will first discuss the advantages and disadvantages of a nominalization in one fact type from the example student-project case study, and the consequences for the relational schema. Next, we will discuss the same issues for a nominalization that concerns two fact types, in a separate example.

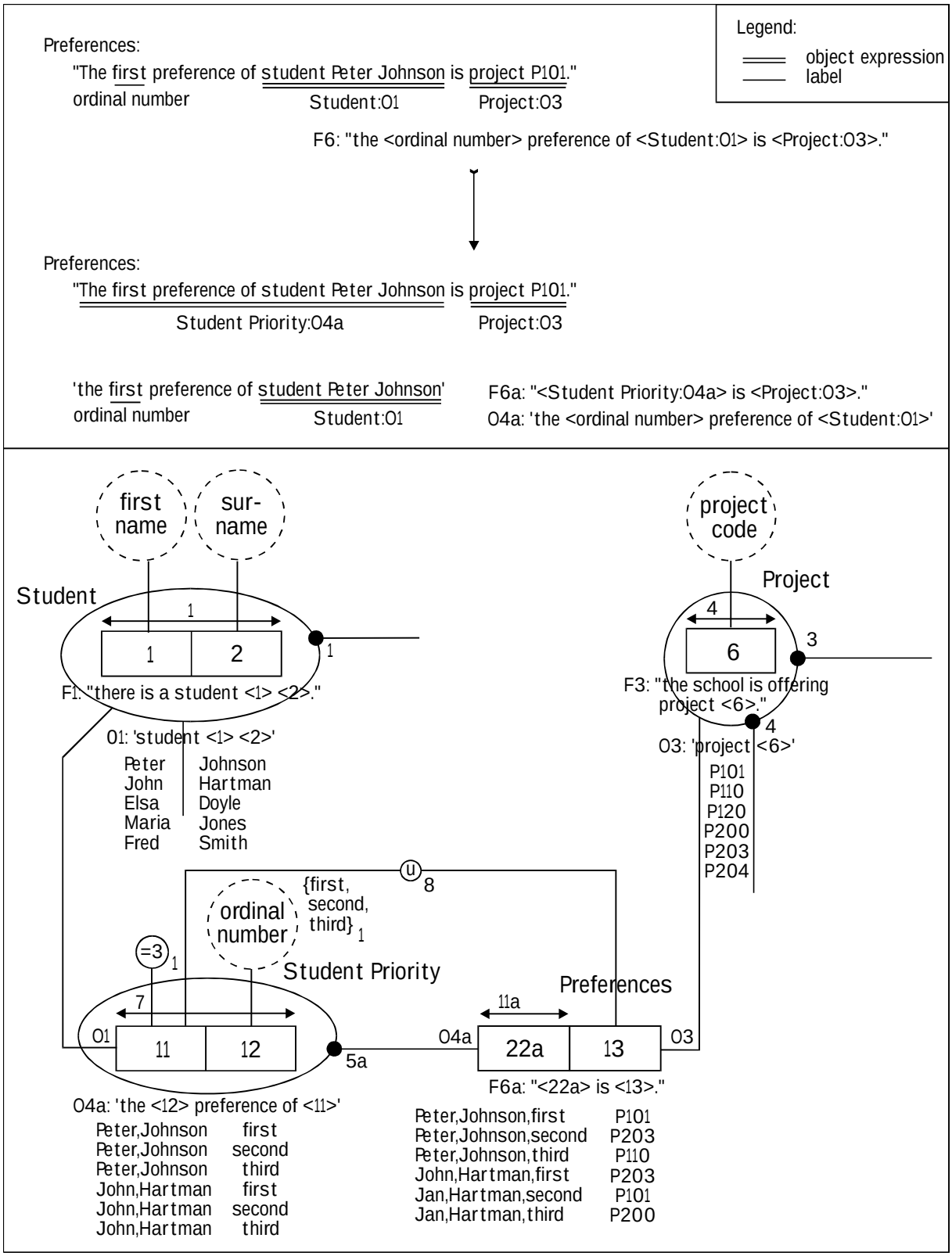


Figure 5.6: different analysis of fact type Preferences, manner a

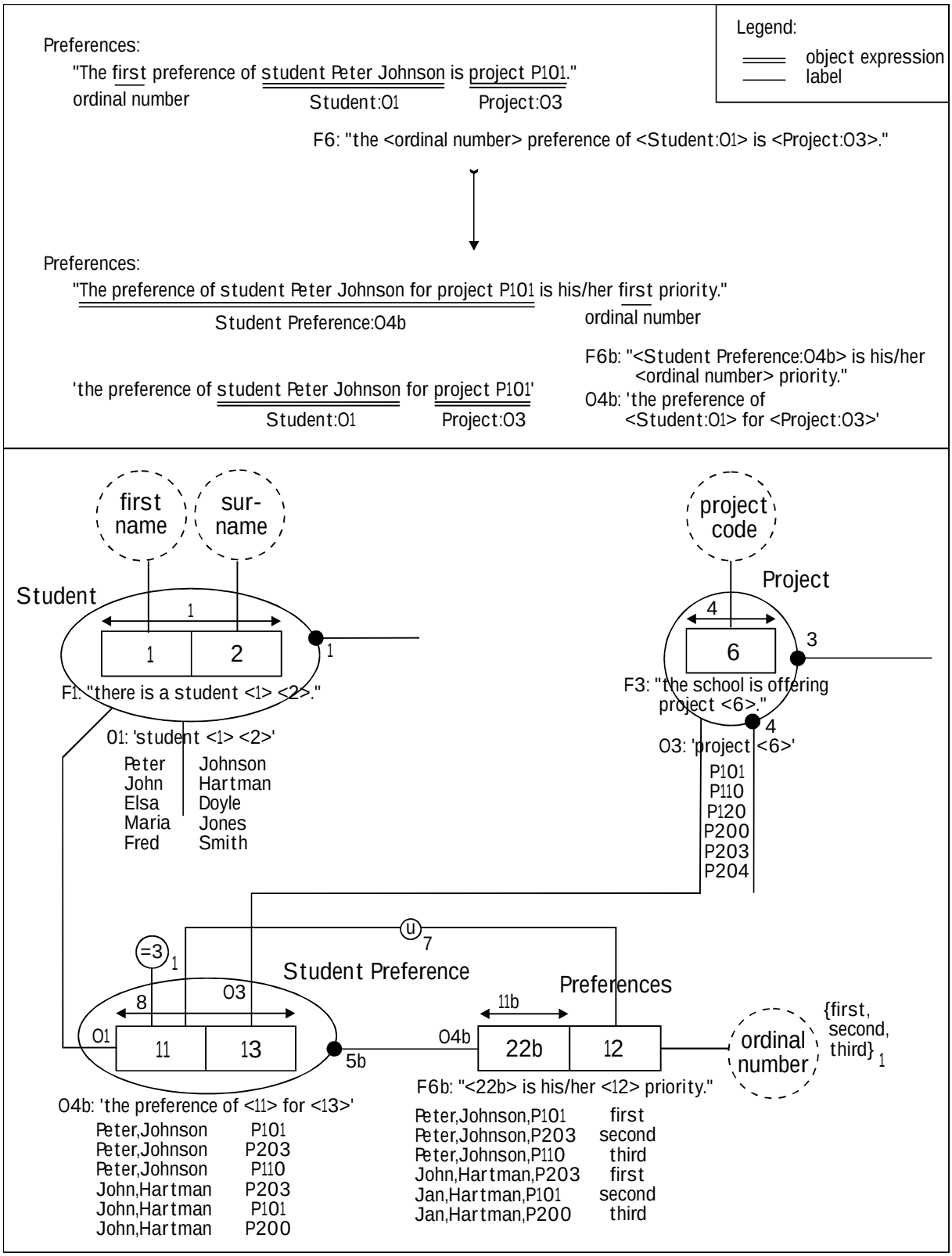


Figure 5.7: different analysis of fact type Preferences, manner b

5.1.2.1 Nominalization in One Fact Type

Although it is often (but not always) possible to analyze exactly the same fact expressions in different ways where semantic equivalence is concerned, we decided in this section to change the verbalization slightly. We already pointed out in section 3.3.2 that it is sometimes desirable to do so, for example because the (main) verb would otherwise end up in an object type expression (OTE). Rephrasing the sentences is even mandatory if an OTE is spread over a fact type expression (FTE) in disconnected pieces, because FCO-IM (still) requires that an OTE is a connected part of an FTE. Both situations occur in the example below.

In section 2.3, the contents of the example documents were verbalized in a collection of elementary fact expressions (sentences). In section 2.4, we classified and qualified these fact expressions (figure 2.7). We drew the corresponding IGD, and added all the constraints in chapter 3. The final IGD is shown in figure 3.25. We derived a logical relational schema from this with the GLR-algorithm, which is shown in figure 4.12.

In section 2.4.1, we analyzed fact expressions 46 - 66 in such a way that a ternary fact type Preferences resulted. We will now present two different analyses, which will both lead in different manners to an extra nominalized fact type, in figures 5.6 and 5.7. The first (manner a in figure 5.6) uses the same fact expressions as in section 2.4.1, the second (manner b in figure 5.7) uses a slightly different verbalization of the same facts.

We will first discuss figure 5.6. Fact expressions 46a - 66a below are the same as fact expression 46 - 66 from section 2.4.1, but will be analyzed differently:

46a) “The first preference of student Peter Johnson is project P101.”
…...
66a) " third " " " Tom Dakota " " P110

The classification and qualification of fact expression 28a is shown at the top of figure 5.6, in which for clarity also the old analysis is given. In consultation with the domain expert, a new object type Student Priority is distinguished now. The corresponding IGD is shown at the bottom of figure 5.6. Uniqueness constraint 8, on roles 11 and 13 in the old IGD, has now become an inter fact type UC, since these roles have now ended up in different fact types. UC 7 stays on roles 11 and 12, so the new nominalized fact type Student Priority satisfies the n rule (see section 3.3.1.2). UC 11a has now taken over the meaning of the old UC 7 in figure 3.25, namely that a student can only name one project per priority. Because no null values can occur in the old ternary fact type Preferences in figure 3.5 (it is an El-IGD after all), the project must be known for each Student Priority in figure 5.6 (the entire population of Student Priority must also occur under the new role 22a). This is why totality constraint 5a applies to role 22a.

We will next discuss figure 5.7. Suppose that the domain expert had chosen a different verbalization, namely fact expressions 46b - 66b below:

46b) "The preference of student Peter Johnson for project P101 is his/her first priority."
…..
66b) " " " " Tom Dakota " " P110 " " third "

The classification and qualification of fact expression 28b is shown at the top of figure 5.7, in which for clarity also the old analysis is given. In consultation with the domain expert, a new object type Student Preference is recognized this time. The corresponding IGD is shown at the bottom of figure 5.7. Uniqueness constraint 7, on roles 11 and 12 in the old IGD, has now become an inter fact type UC. The new nominalized fact type Student Preference satisfies the n rule. UC 11b has now taken over the meaning of the old UC 8 in figure 3.25, namely that the project preferences of a student must all be different. Totality constraint 5b applies to role 22b for the same reason as was explained for TC 5a in figure 5.6.

These two different nominalization transformations of the old ternary fact type Preferences illustrate the following generally valid points:

1 For a successful nominalization transformation, there must be one or more UCs to ensure that the new nominalized fact type satisfies the n rule after the transformation. So it is possible in principle to carry out a nominalization transformation in every fact type with two or more roles. One simply chooses (a part of) the roles under the existing UCs and turns them into a new nominalized fact type. Even if there would be a multiple role UC on all three roles of the ternary fact type Preferences from figure 3.25, the same nominalization transformations as in figures 5.6 and 5.7 could still be done, which would lead to a multiple role UC on roles 22a + 13 (in figure 5.6) or 22b + 12 (in figure 5.7) of the new binary fact type Preferences. A third transformation would then also be possible: nominalization of roles 12 and 13 with a new binary fact type with roles 22c and 11. The readers can easily verify this for themselves.

2 During a nominalization transformation in an elementary IGD on just one fact type, a single role totality constraint will apply automatically to the new role, since the original fact type cannot contain null values. In the other direction (denominalization), the transformation can only be carried out in an El-IGD if such a TC exists, since otherwise null values would appear in the denominalized fact type after the transformation.

When we derive the relational schemas from the three IGDs of figures 3.25, 5.6 and 5.7 using the GLR algorithm, then the same schema results in all three cases (see figure 4.12), except for the name of a table. In figure 5.6, role 22a will be marked for grouping. Then role 13 will be absorbed by fact type Student Priority, and the diagram of figure 3.25 will have been formed again, in which the ternary fact type is now called 'Student Priority' however, instead of 'Preferences'. The same will happen in figure 5.7, here with the name 'Student Preference'. Further execution of the GLR algorithm will result in the same tables with different names. (By the way, if we exchange the names 'Preferences' and 'Student Priority' before classifying and qualifying in figure 5.6, then there will even be no difference in table names; the same holds for figure 5.7).

A nominalization-transformation that only concerns one fact type has no influence on the structure of the relational schemas that are derived from the IGDs. (This is true also when only a part of the roles under a UC are nominalized, as in point 1 above. The new nominalized fact type will then absorb no roles during grouping, and it will disappear during lexicalizing, because it will turn out to be lost; the readers can verify this for themselves).

In view of the above, there is hardly a point in carrying out a nominalization transformation on just one fact type: often several nominalization transformations are possible and they have no influence (apart from table names) on the structure of the relational schema. In the examples in figures 5.6 and 5.7, we have carried out a nominalization transformation in two separate ways, one by taking UC 7 as the starting point (figure 5.6) and one by taking UC 8 (figure 5.7). There is no reason to prefer one possibility over the other, however. On the contrary, by choosing either one of the two possibilities, we needlessly disturb the symmetry of the old ternary fact type Preferences from figure 3.25: both UCs 7 and 8 are equivalent in figure 3.25, whereas they have different weights in figures 5.6 and 5.7. It is in general a good modeling principle not to break an existing symmetry, unless there is a good reason to do so. Such a reason could be: the same nominalization can be carried out in another fact type, resulting in the same new object type. The principle to model in a redundancy free way weighs heavier than the principle to conserve symmetry This is the raison d'etre for the nominalization test from section 3.3.2. We will discuss an example of this in the following section 5.1.2.2. Another reason to break the symmetry could be the following. After drawing the IGD in figure 5.6, the analyst proposes to remove object type Student Priority on the grounds of symmetry considerations. The domain experts however attach great value to distinguishing objects of object type Student Priority during classification and qualification, because they are used to consider these as meaningful objects in the UoD, whereas an object type Student Preference (the other possible nominalization) does not appear in their perception of the UoD at all.

5.1.2.2 Nominalization in More than One Fact Type

In this section we will use a new example, which stands apart from the example student-project case study. It concerns a library where students can borrow and return books. Figure 5.8 contains the classification and qualification of four example sentences.

In figure 5.9, the corresponding IGD is shown, populated with the example fact expressions and provided with all constraints. During the determination of the uniqueness constraints it is made clear that a book copy can only be lent out once a day (returned books are checked for damage and put back on the shelves only after closing time at the end of the day): this explains UC 4 on fact type Student Book Loan and UC 6 on fact type Book Return. A book copy can also only be returned once a day: this explains UC 5. SC 1 expresses that each returned book must first be lent out. C1 says that in each tuple the return date cannot be earlier than the loan date. Finally, C2 states that loan periods of the same book cannot overlap one another.

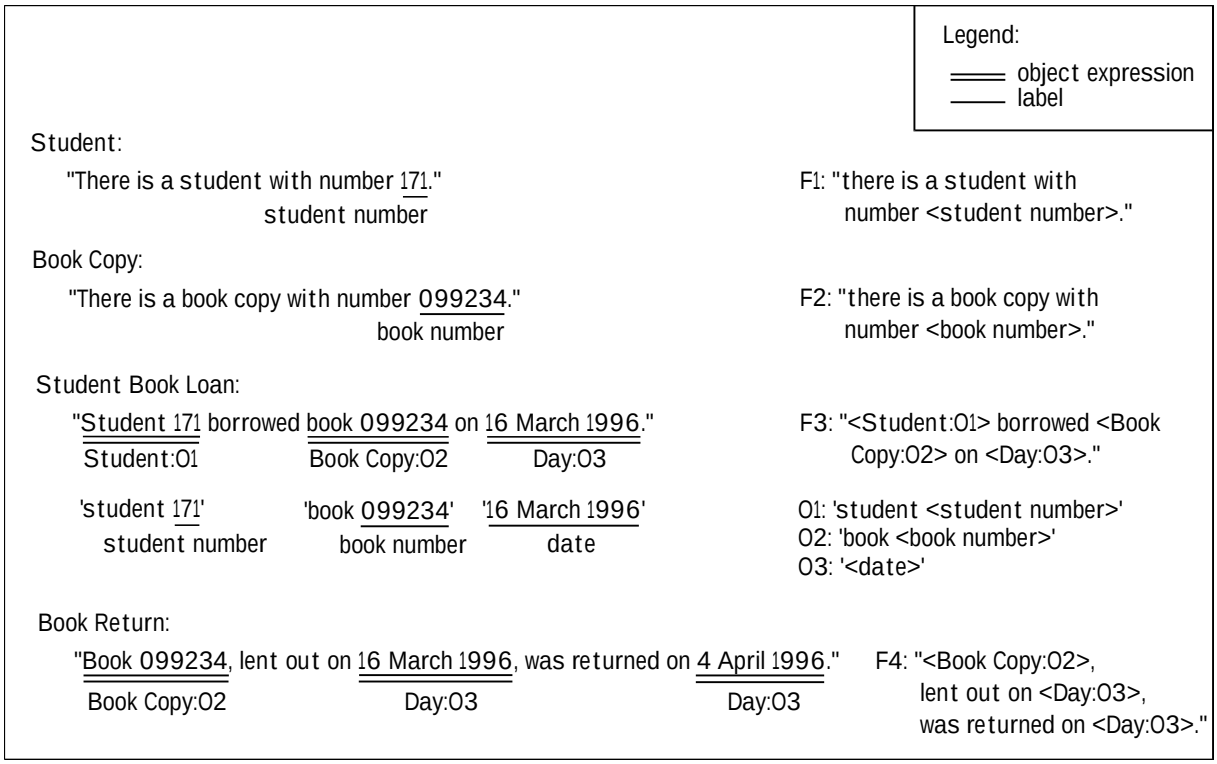


Figure 5.8: library classification and qualification 1

Is fact type Book return actually elementary? Or will fact expressions such as: “Book 099234 was returned on 04 April 1996.” be sufficient? Clearly, the users want to record the history (i.e. they want to save the facts about earlier book loans), but even then the constraints make it possible in principle to find the corresponding loan date for each return date. However, this would require the librarian to go through the complete list of all loans and returns for each returned book copy (the oldest loan must be associated with the oldest return and so forth). So, strictly speaking, role 8 in fact type Book Return is redundant, but the domain experts highly value this manner of verbalizing a book return. Moreover, introducing this redundancy simplifies updates and retrievals considerably, and so the database administrator prefers this modeling over a redundancy free one as well. The redundancy is controlled by subset constraint 1, so that the integrity of the information is assured.

The nominalization pattern (see section 3.3.2) occurs in fact types Student Book Loan and Book Return. We will not carry this nominalization out here yet, however, in order to illustrate the effects of not doing so on the relational schema that is to be derived for this UoD.

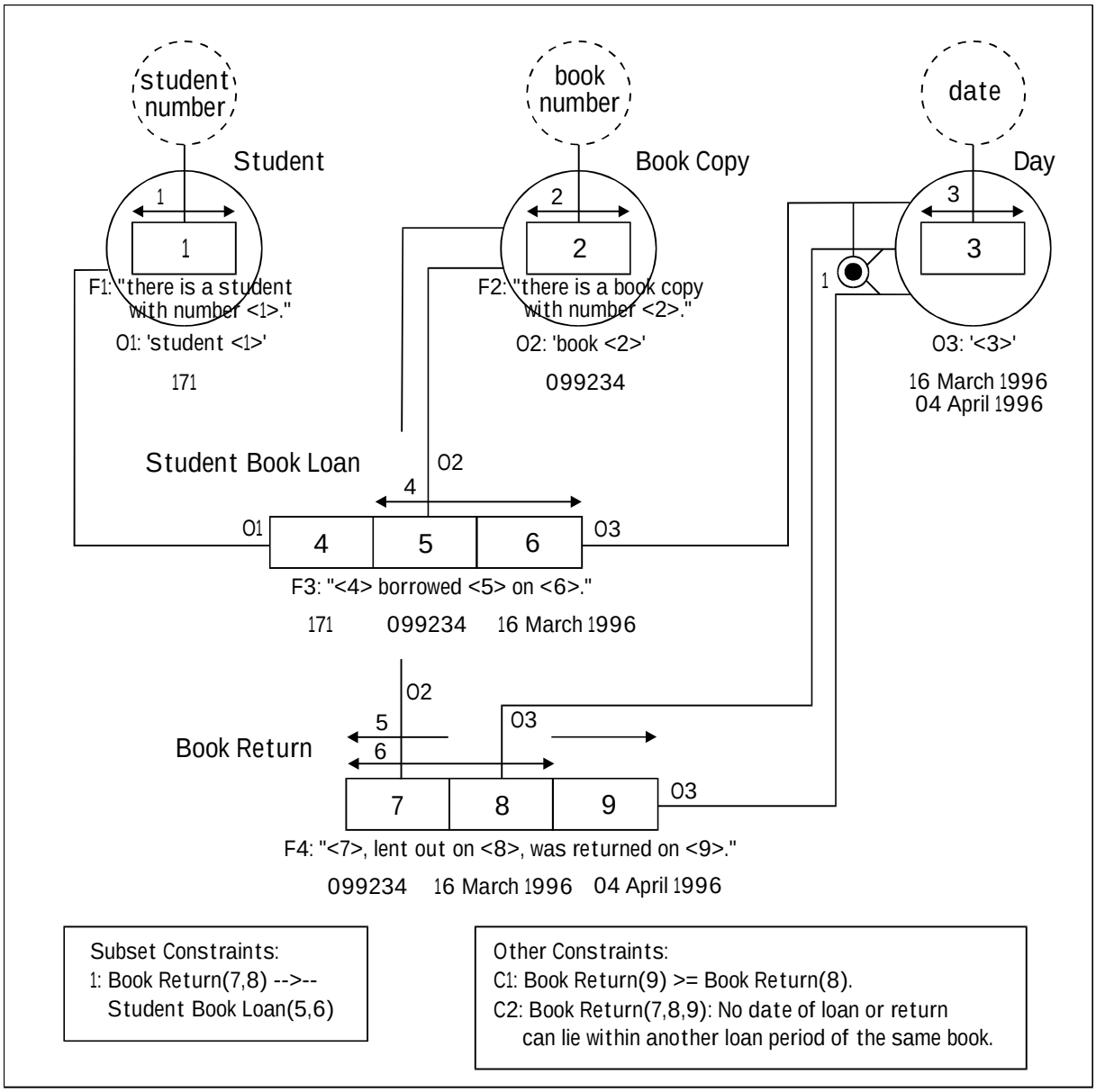


Figure 5.9: library IGD 1

The relational schema that follows from the IGD in figure 5.9 using the GLR algorithm is shown in figure 5.10 (fact type Day was deleted because it turns out to be lost after lexicalizing). Appropriate role fixes (see section 4.4.2) were used during the derivation to achieve clear column names. Note that the two tables Student Book Loan and Book Return have the same primary key.

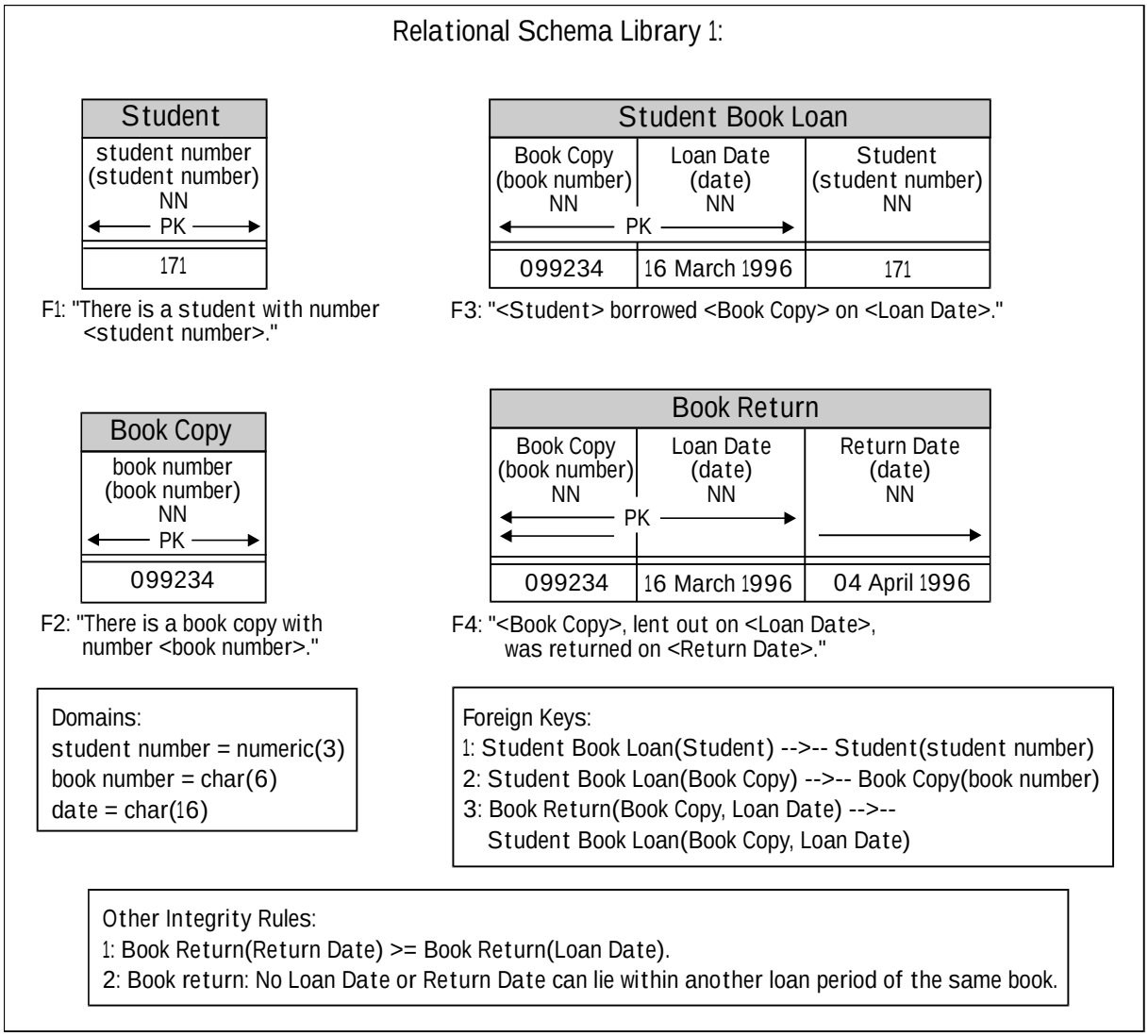


Figure 5.10: relational schema library 1

We will look next at what happens if we carry out a nominalization in fact types Student Book Loan and Book Return. Two nominalization patterns occur in these fact types in figure 5.9: the first pattern concerns roles 5 + 6 under UC 4 on the one side and roles 7 + 8 under UC 6 on the other side; the second patterns again concerns roles 5 + 6 under UC 4 on the one side but this time roles 7 + 9 under UC 5 on the other side. It is made clear in further interviews with the domain experts that only the first pattern can be considered for nominalization, because the same objects, belonging to an object type Loan, occur under roles 5 + 6 as well as under roles 7 + 8,. Under roles 7 + 9, however, there are objects from another object type, which could for example be called Return. (This also illustrates the necessity to check the equality of the objects and their object types for each nominalization pattern, as was required in section 3.3.2). Figure 5.11 contains another classification and qualification of the same example fact expressions as in figure 5.8, this time with an extra object type Loan. Note that there are two object type expressions for Loan if we do not change the fact expressions.

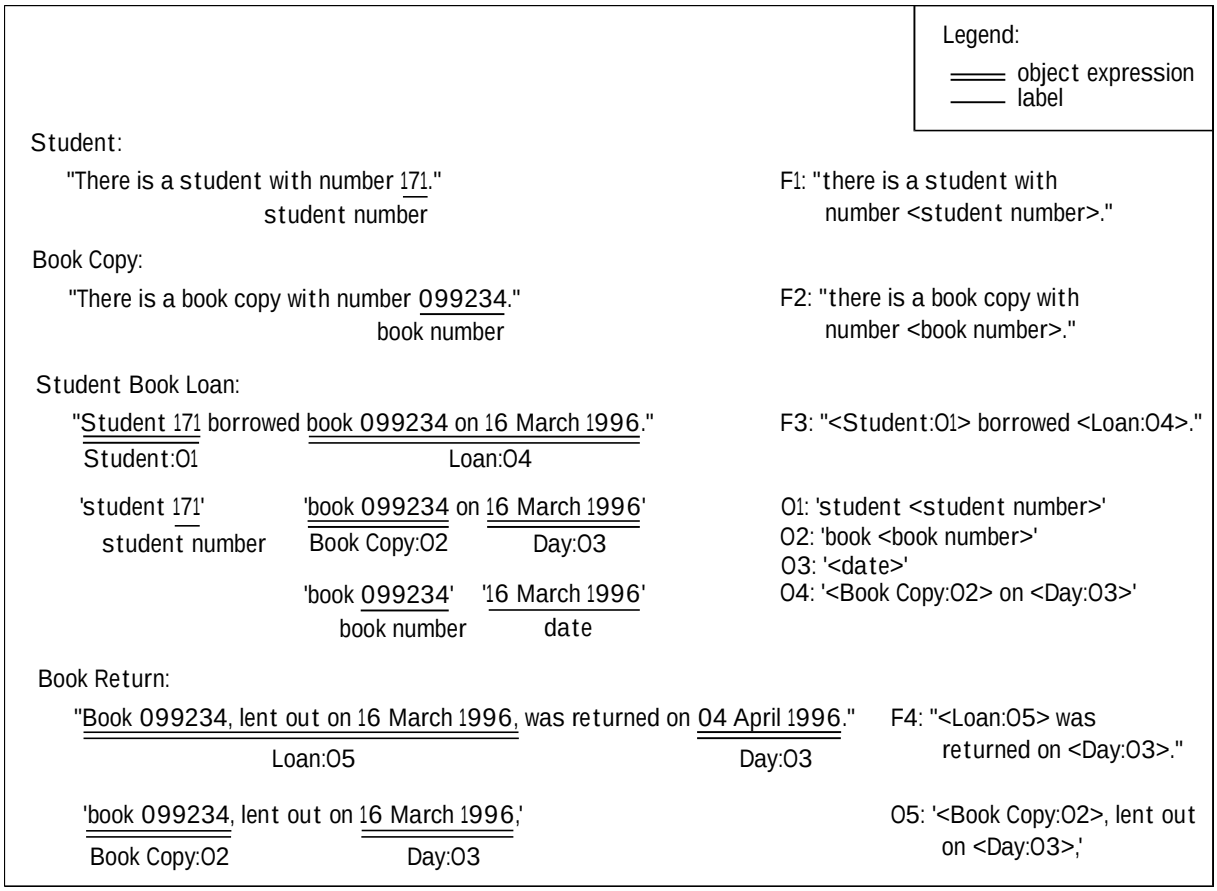


Figure 5.11: library classification and qualification 2

The corresponding IGD is in figure 5.12. The nominalization of Loan breaks the symmetry of fact type Book Return: UC 5 has now become an inter fact type UC. But because the same nominalization is also carried out in fact type Student Book Loan, the principle to model in a redundancy free way weighs heavier here and the symmetry breaking is justified (cf. section 5.1.2.1, last paragraph). The presence of TC 2 on role 12, and the absence of a TC on role 13 follow respectively from the fact that the old IGD in figure 5.9 is elementary (no null values) and from the old SC 1. (This also illustrates the necessity to redetermine all the constraints for all new or changed fact types after a change in structure.) SC 1 itself has now become redundant.

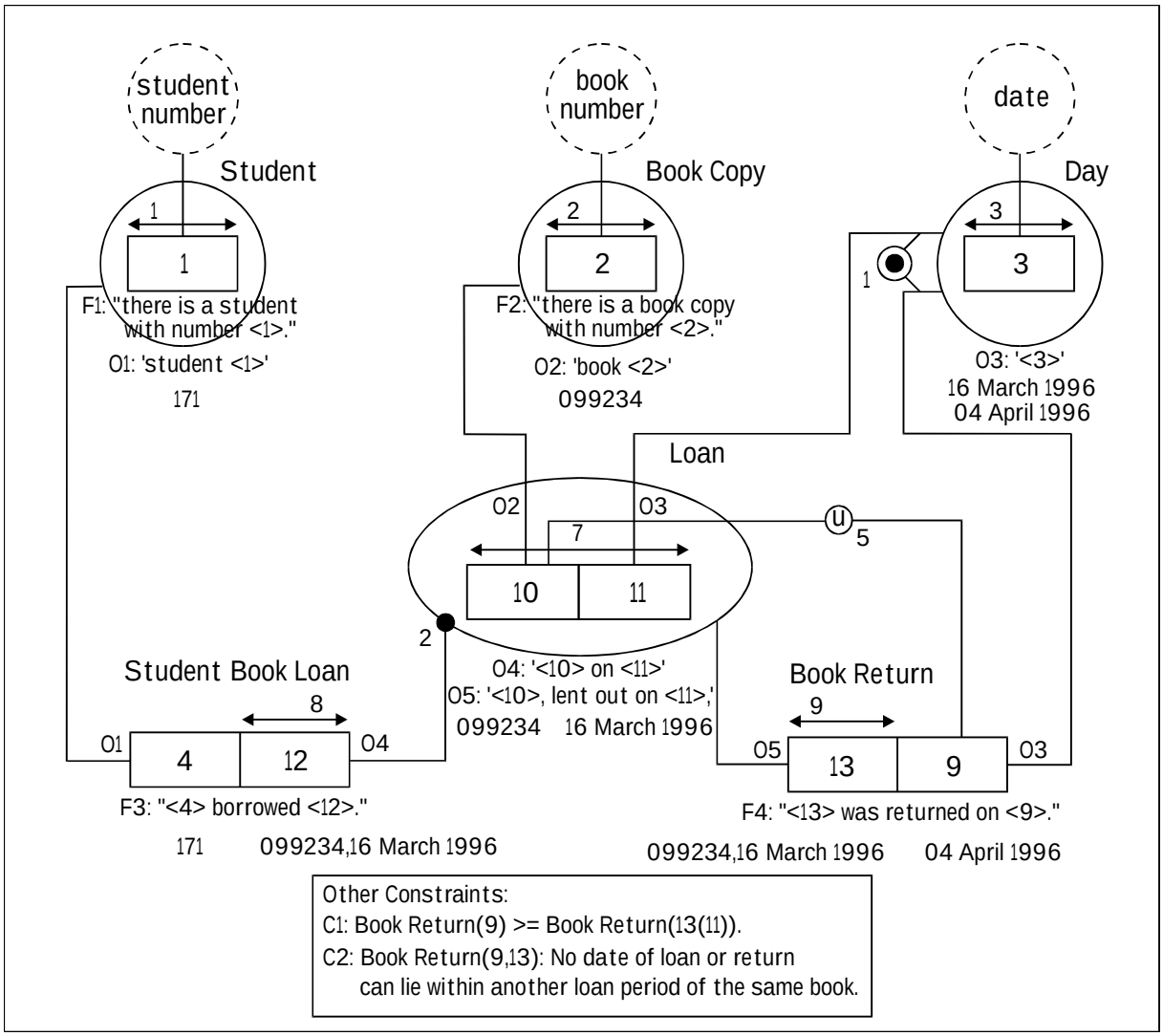


Figure 5.12: library IGD 2

The relational diagram that follows from figure 5.12 using the GLR algorithm is shown in figure 5.13. Compared to figure 5.10, there is now one table less: table Loan replaces the two old tables Student Book Loan and Book Return. Table Loan has the same primary key as both old tables had.

A nominalization transformation that concerns more than one fact type often results in a simpler structure of the relational schema. There is also little reason to prefer the schema from figure 5.10 above that of figure 5.13 from the viewpoint of optimization of storage or speed of performance. It is therefore advisable to deviate from the nominalization requirement from section 3.3.2 only in special circumstances.

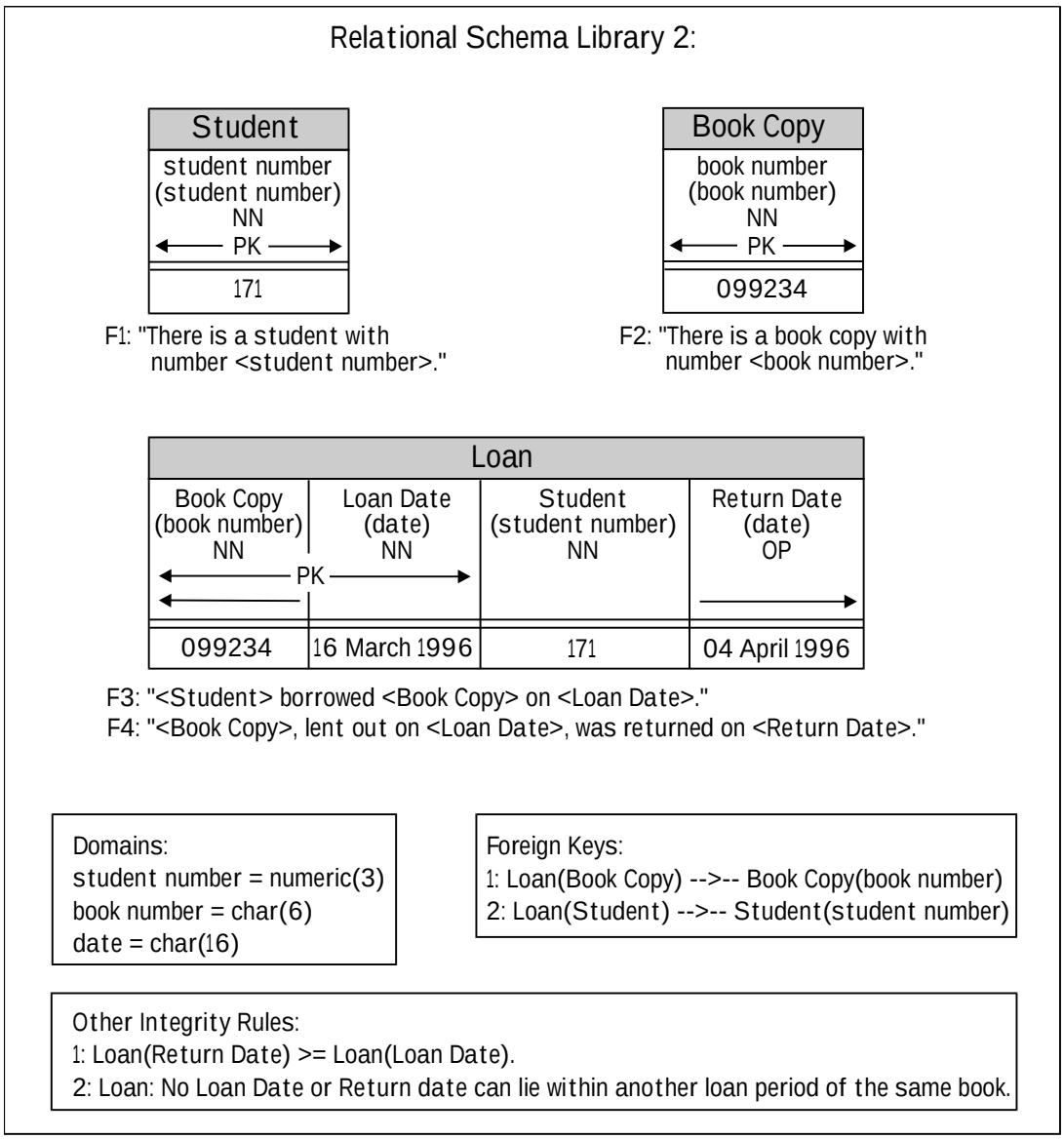


Figure 5.13: relational diagram library 3

5.1.3 Final Remarks

We conclude this section on semantically equivalent transformations with a few comments.

1 Next to the standard semantically equivalent transformations (those discussed in sections 5.1.1 and 5.1.2, and grouping and lexicalizing transformations), sometimes non-standard semantically equivalent transformations are also possible. These are usually the result of specific properties of the information in an UoD and cannot be discussed in general.

2 After carrying out a semantically equivalent transformation, we must redetermine all the constraints for each new or altered fact type. A correspondence table (see section 4.6, comment 5) is a useful aid for this.

5.2 Unary Fact Types

In this section we will discuss two aspects of unary fact types: a well-formedness rule in section 5.2.1, and how to treat them in the derivation of a relational schema in section 5.2.2.

5.2.1 Well-Formedness Rule for Unary Fact Types

Please consider figure 5.14, which contains a faulty unary fact type on the left-hand side.

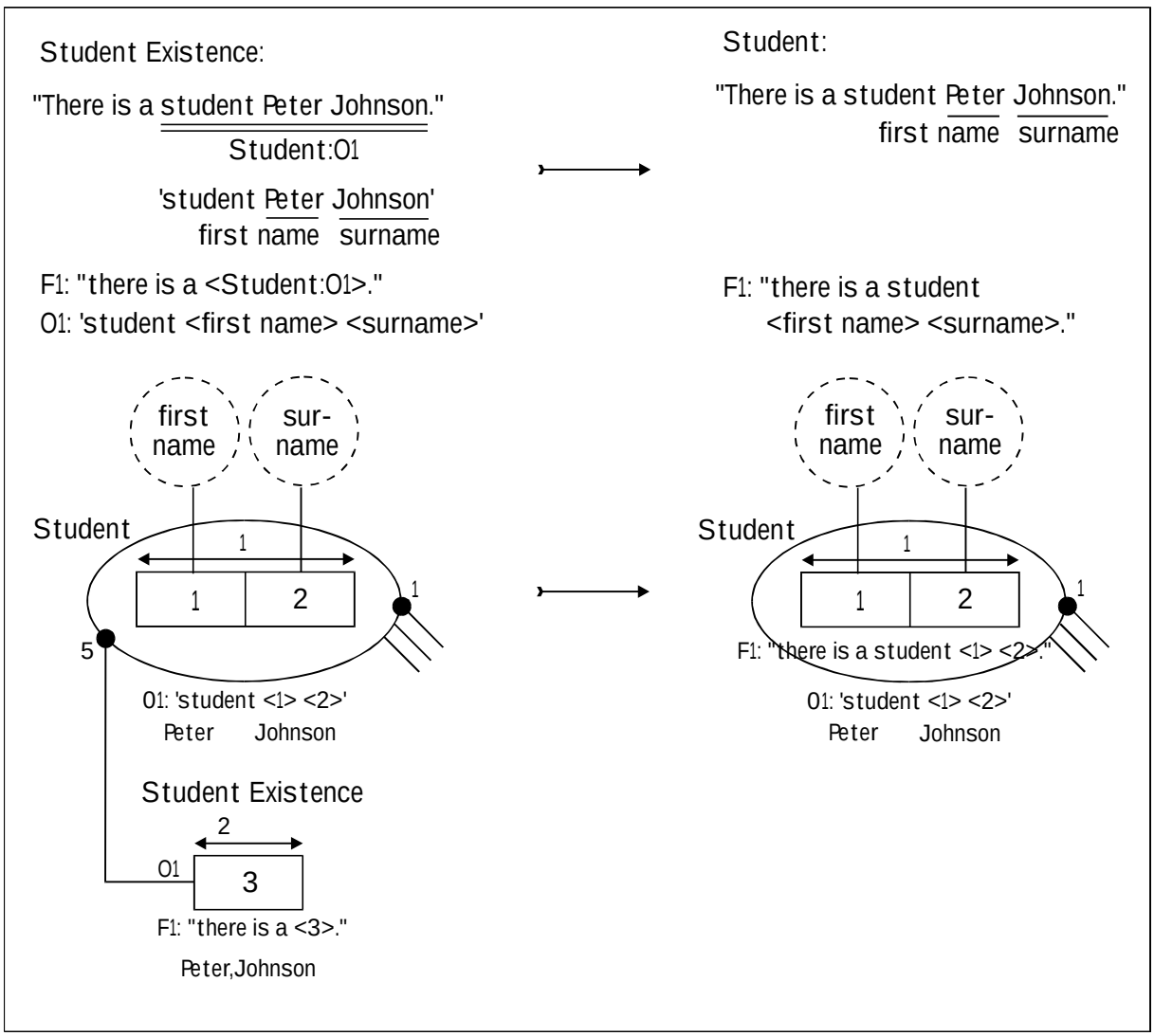


Figure 5.14: correcting a faulty unary fact type

Let us suppose that we had done a wrong classification and qualification of the existence postulating fact expressions for Student in the example student-project case study, and in the way that is shown on the left in figure 5.14. We need the awkward name 'Student Existence' now because the name 'Student' cannot be used for two different fact types. The analysis of the remaining fact expressions is the same as in chapter 2. A part of the corresponding IGD is also shown in figure 5.14: an extra unary fact type Student Existence has appeared. The constraint analysis yields the depicted uniqueness and totality constraints, of which TC 5 on role 3 is particularly important here. The population of role 3 is the same as that of roles 1 + 2, so the existence must be known for each student.

The diagram on the right-hand side of figure 5.14 is simpler (uses less fact types), however, and achieves the same goal: for each student the existence must be known. The existence postulating fact type expression is written under the nominalized fact type Student itself. Moreover, on the left-hand side of figure 5.14, object type expression O1 is not a nominalization of fact type expression F1, but of another unspecified fact type expression for Student, whereas O1 can very well be seen as a nominalization of F1.

On these grounds we formulate the following well-formedness rule:

There can be no single role totality constraint on the role of a unary fact type.

If a unary fact type breaks this rule, then it must be removed, and its fact type expression must be moved to the object type that played the role of the unary fact type, with the object type expression filled in. The population can be deleted because the same population occurs under the object type that played the role of the unary fact type. The rule also applies to other fact type expressions than existence postulating ones. Naturally, this only concerns unary fact types with a non-lexical role.

Unary fact types, played by non-lexical object types, are then only allowed if the role has no single role totality constraint. Such unary fact types are called *subtypes*, because they contain a *real subset* (that is a part of, but not the whole collection) of the objects in the object type that plays the role. We will discuss subtypes fully in section 6.1.

Unary fact types can arise in object type - fact type transformations as well as in nominalization transformations, see for example figure 3.18. However, it could only happen after a nominalization transformation that a unary fact type would get a single role TC as a result of the redetermination of TCs, and so it must then be deleted and the corresponding fact type expression must be moved.

5.2.2 Derivation of a Relational Schema from Unary Fact Types

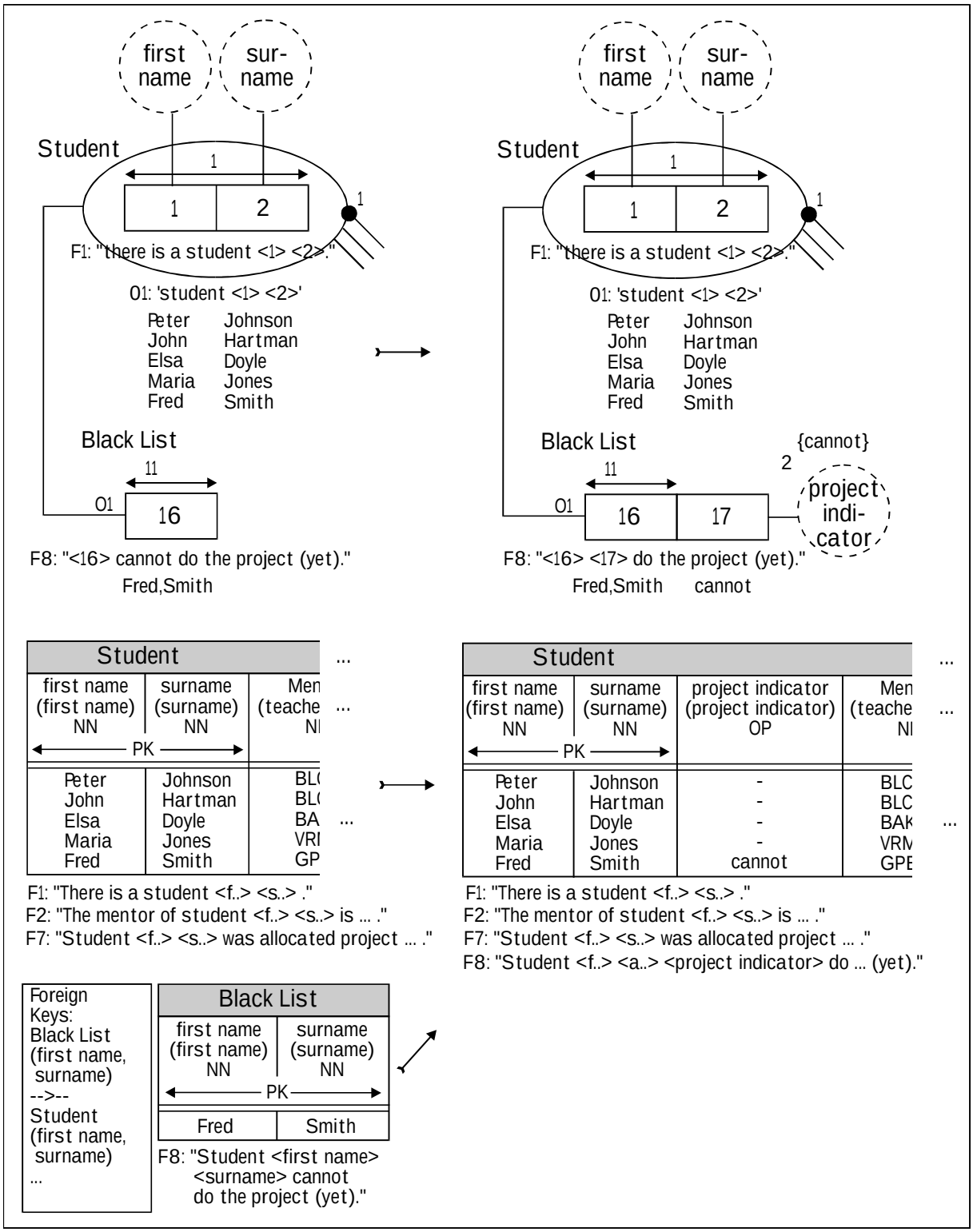


Figure 5.15: derivation of a relational schema for unary fact types

We introduced a unary fact type Black List in figure 3.24 in section 3.6, to give an illustration of an exclusion constraint. We show it again in figure 5.15, leaving out the other fact types. Fact type Black List satisfies the well-formedness rule. Here we will consider the effect of a semantically equivalent transformation on the derivation of a relational schema from an IGD with unary fact types that are not themselves nominalized.

The left-hand side of figure 5.15 contains a part of the IGD with the unary fact type Black List, above a part of the relational schema that follows from it. Role 16 will not be marked for grouping, because it is not in an n-ary fact type with n larger than 1 (condition 1 from section 4.1.2). Otherwise the information that Fred Smith cannot do his project yet will be lost. Fact type expression F8 cannot be moved to Student either: we would not know anymore which students cannot start their projects yet: F8 applies only to Fred Smith in this example(there is no single role totality constraint for role 16). So after lexicalizing a separate table Black List appears, with a foreign key reference to table Student, which is also partly shown on the left-hand side in figure 5.15.

The right-hand side of figure 5.15 shows the situation after the application of an unusual, but nevertheless standard object type - fact type transformation in the object type direction: there is now a label type ‘project indicator’ with only one allowed value: ‘cannot’, listed in value constraint 2. Naturally, the same sentences are still being modeled. The new classification and qualification is simple; we do not give this separately.

Now role 16 will be marked for grouping, and the new role 17 will be absorbed by Student. There is now no separate table Black List. The resulting table Student is partly shown in figure 5.15 on the right-hand side below the IGD. So, semantically equivalent transformations affect the structure of the resulting logical relational schema here as well.

5.3 Addresses with Postal Code: Denormalization

In this section we will give a redundancy free modeling of postal addresses with postal codes in The Netherlands. The context is a company that records the postal address of its clients, amongst other things. We will do this firstly to illustrate information modeling in an unfamiliar context (even most Dutch readers never consider the structure of the Dutch postal code even though they use it every day), in which a redundancy free verbalization is less obvious than it seems at a first glance. Secondly, this is a good example to introduce the concept of *denormalization*: the conscious and controlled introduction of redundancy (which must be kept well under control subsequently) in order to reduce the number of tables when this is desired, but not possible in a redundancy free way anymore. Thirdly, this example enables us to show how to deal with very exceptional cases within an otherwise simple structure.

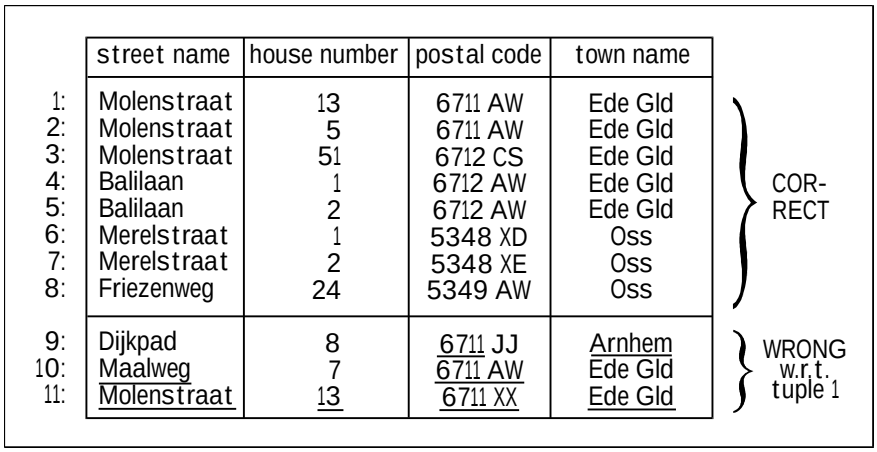
In section 5.3.1, we will sketch the relevant properties of the postal code, leaving a very exceptional case out of consideration just yet. We will give the redundancy free modeling of this in section 5.3.2, and the logical relational schema that follows from it. In section 5.3.3, we will denormalize this schema in a number of steps. Finally in section 5.3.4 we will also deal with the exceptional case.

5.3.1 Properties of the Postal Code

Sources we used for this section: Postal Code Directory from PTT Post, edition 1978, with supplement nr.27, May 1994, and the folder "The How and Why of the Postal Code" from PTT Post, undated. Further information was obtained from the Customer Service of PTT Post.

Every postal address in The Netherlands has a postal code, consisting of four digits followed by two letters. The postal code is built up hierarchically: the first two digits identify a region in The Netherlands, as is shown on the inside of the cover of the Postal Code Directory. The third digit divides these regions further down to the level of a municipality (small city, town or village) or an area (in a large city). The fourth digit breaks these sub regions down further. The four digits together identify a *district*. Small towns consist of only one district, others have more districts. There always belongs only one town name to each district. The two letters after the four digits break each district further down to the level of a street section, which is a part of a single street. A full postal code identifies a number of postal addresses situated in the same street section, for example a row of houses on one side of the street (we ignore a few exceptions here, see the discussion of tuple 10 from figure 5.16 below). The house number is needed finally to identify a unique postal address. Small streets sometimes have only one postal code, but often a street consists of a number of street sections. By the way, a street can run through more than one district and even through more than one town, as can be easily verified by leafing through the Postal Code Directory.

Figure 5.16 summarizes a few possibilities and impossibilities of postal coding. The correct examples are chosen from the Postal Code Directory, the faulty ones were made up of course. Just to be clear: we do not want to model all the information from the Postal Code Directory, but are only interested in postal addresses of clients. So we will not consider the range of a postal code (postal code 6711 AW applies to Molenstraat in Ede, house numbers 5 up to and including 45 uneven, and 80 up to and including 126 even), but if a client lives in the Molenstraat in Ede, we do want to record the corresponding postal code 6711 AW. (By the way, because there are two different towns with the name Ede in The Netherlands, one in the province of Gelderland, and one elsewhere, PTT Post uses the name Ede Gld for this one.)



	street name	house number	postal code	town name	
1:	Molenstraat	13	6711 AW	Ede Gld	COR-RECT
2:	Molenstraat	5	6711 AW	Ede Gld	
3:	Molenstraat	51	6712 CS	Ede Gld	
4:	Balilaan	1	6712 AW	Ede Gld	
5:	Balilaan	2	6712 AW	Ede Gld	
6:	Merelstraat	1	5348 XD	Oss	
7:	Merelstraat	2	5348 XE	Oss	
8:	Friezenweg	24	5349 AW	Oss	
9:	Dijkpad	8	6711 JJ	Arnhem	WRONG w.r.t. tuple 1
10:	Maalweg	7	6711 AW	Ede Gld	
11:	Molenstraat	13	6711 XX	Ede Gld	

Figure 5.16: addresses with postal codes

Explanation per tuple from figure 5.16:

- Tuples 1 and 2: a number of postal addresses can have the same postal code.
- Tuples 2 and 3: a street can be divided into a number of street sections. The district often remains the same, in which case there is only a difference in one or both letters. Here apparently the Molenstraat in Ede Gld also passes through another district, causing the fourth digit to change as well.
- Tuples 4, 5, 6 and 7: the Postal Code Directory consistently makes a distinction between even and uneven house numbers. Often the postal codes for even and uneven numbers are different, as in tuples 6 and 7, but they are also often the same, as in tuples 4 and 5. Upon inquiry to PTT Post, we learned that this depends on the length of the street, the number of delivery points (i.e. mailboxes etc), the location of the houses and so on. There are no fixed rules for this, and it is examined from case to case.
- Tuples 1, 5 and 8: the same combination of letters can occur with different combinations of digits, in the same or other towns, but of course not within the same district.
- Tuples 1 and 9: a district always lies in one town. A town has only one town name (in the case of synonyms only the name as defined by PTT Post applies), which is unique for each town (see the remark on Ede Gld above), so it is not possible that two different town names occur with the same postal code digits. Tuples 1 and 9 therefore cannot appear together in a correct population.
- Tuples 1 and 10: here we will first make a simplifying assumption different from reality. We assume that a street section always lies in just one street. Under this assumption it is not possible for two different street names to occur with the same postal code (digits plus letters). Tuples 1 and 10 therefore cannot occur together in a correct population.

  In reality this is possible nevertheless in a few highly exceptional cases, where there are two small streets with few delivery points, with only uneven house numbers in the one street, and only even house numbers in the other street. The combination of postal code plus house number then determines in which street the postal address can be found. An example of this (with thanks to the employees of PTT Post to whom we inquired): in the town of IJlst, postal code 8651 BH is for the Holtropweg, house numbers 1 up to, and including 5 uneven, as well as for the Jonker Rispenstraat, house numbers 2 up to, and

including 14 even. Because of the exceptional nature of this example, we will continue the discussion under the assumption above and correct for the exceptions later in section 5.3.4.

- Tuples 1 and 11: it cannot be that one postal address has two different postal codes, not even if both other bans (see tuples 9 and 10) are being respected. In tuple 11, as well as in tuple 1, district 6711 lies in Ede Gld, and street section 6711 XX might well be a part of the Molenstraat in Ede Gld, but according to tuple 1 house number 13 in the Molenstraat in Ede Gld already lies in another street section, identified by postal code 6711 AW. Tuples 1 and 11 therefore cannot occur together in a correct population.

5.3.2 Redundancy Free Modeling

The UoD is a company with clients that are identified by a client code. Amongst other things, the company records the postal codes of their clients, one postal address per client. A typical non-elementary verbalization would be: "The postal address of client C1 is Molenstraat 13, postal code 6711 AW in Ede Gld.". The company will only record a postal address if it belongs to a client: the rest of the Postal Code Directory stays out of consideration. So the range of house numbers belonging to the same postal code are not of interest, and neither is information about the location of towns in regions part of the UoD (such as: "Ede Gld lies in region 67."). However, this last restriction will enable us to record an incorrect town name with a district, for instance "District 3435 is part of Ede Gld.", because the information is missing that Ede Gld actually lies in region 67 instead of in region 34 as implied by this faulty sentence. The company managers, however, have no problem with this. They would rather take the risk of recording such a wrong fact than pay for putting in an extra fact type to enforce correctness, which would imply an extra table with a foreign key or other protection routine. For the discussion in this section it makes little difference: it is a good and straightforward exercise to expand the following with a fact type for the location of towns in regions.

We must take the fact into account that a district can lie in only one town (from the collection of towns defined by PTT Post). We verbalize this as: "District 6711 is part of Ede Gld.". Town names are unique for a town. We have assumed that there can be only one street name for each postal code. We verbalize this as: "Street section 6711 WA is part of a street called Molenstraat.". A street name by itself does not identify a street (there are many streets called Juliana Lane, after the former queen of The Netherlands), but because a street section (6711 WA) is part of a district (6711) of which it is known in which town it lies, we do not lose any information by leaving out the town name. Finally, the verbalization of a postal address: "Client C1 has postal address 6711 AW 13.", in which 13 is the house number. Postal code + house number identifies a postal address, as explained in section 5.3.1, second paragraph. Classification, qualification, and further analysis are easy to reconstruct from the IGD in figure 5.17, and so we do not give it separately. Uniqueness constraint 7 expresses that each district lies in only one town, and UC 5 expresses that each street section has only one street name. Now how do we express that each combination of street name, house number and town name has only one postal code (see tuple 11 from figure 5.16)? Inter fact type uniqueness constraint 9 requires this: after carrying out all the necessary natural joins, one fact type with

four roles results: roles 1, 2, 9 and 12, with UC 1 on role 1 and 2 and UC 9 on roles 2, 9 and 12, see figure 5.18.

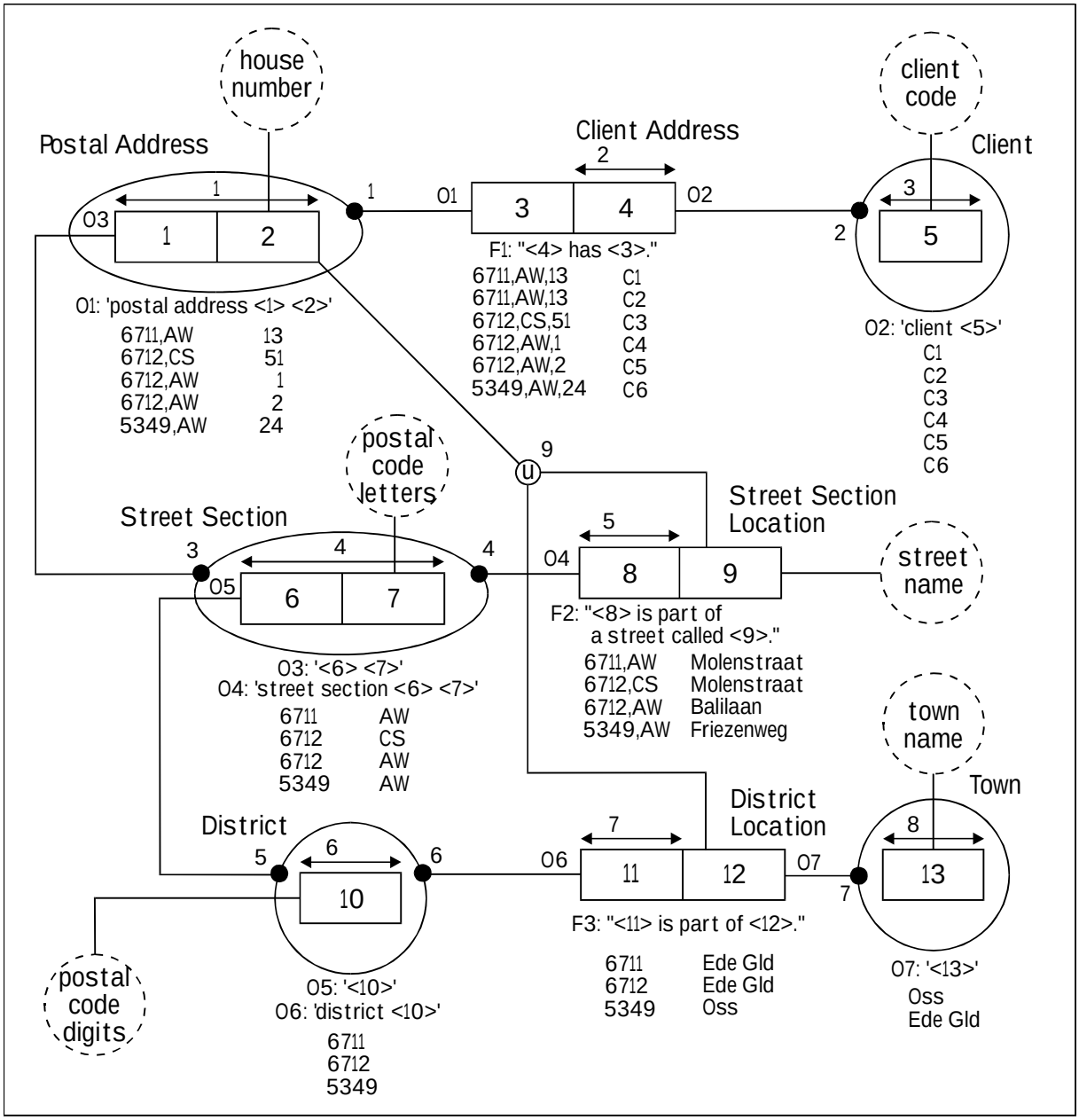


Figure 5.17: El-IGD addresses with postal code

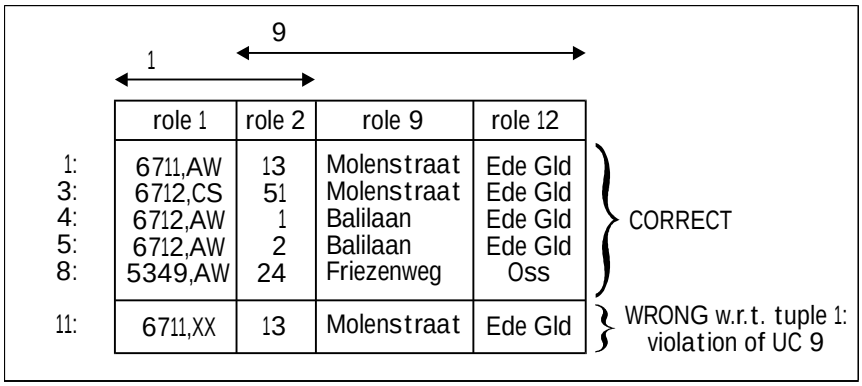


Figure 5.18: clarification of UC 9

Figure 5.18 contains all the tuples from the population of fact type Postal Address from figure 5.17. The tuple numbers in figure 5.18 refer to the same tuples in figure 5.16. UC 9 prevents taking up both tuples 1 and 11 in figure 5.18 (and so in figure 5.17 as well).

Totality constraints 1, 3, 5 and 7 express that we will not record postal addresses, street sections, districts or towns if we do not need them for the postal address of a client.

The relational schema that follows from figure 5.17 using the GLR algorithm is shown in figure 5.19. There are no tables Town or Postal Address because fact types Town and Postal Address turn out to be lost (see section 4.2). During lexicalizing, totality constraint 5 yields an equality constraint between tables District and Street Section, which is transformed in the relational schema into a foreign key reference and another reference in the opposite direction (see figure 5.19). Analogously, two equality constraints arise at first from TCs 1 and 3, but after deleting the lost fact type Postal Address only one remains, which again transforms into two references. UC 9 transforms into an other integrity rule, verbally shown in figure 5.19.

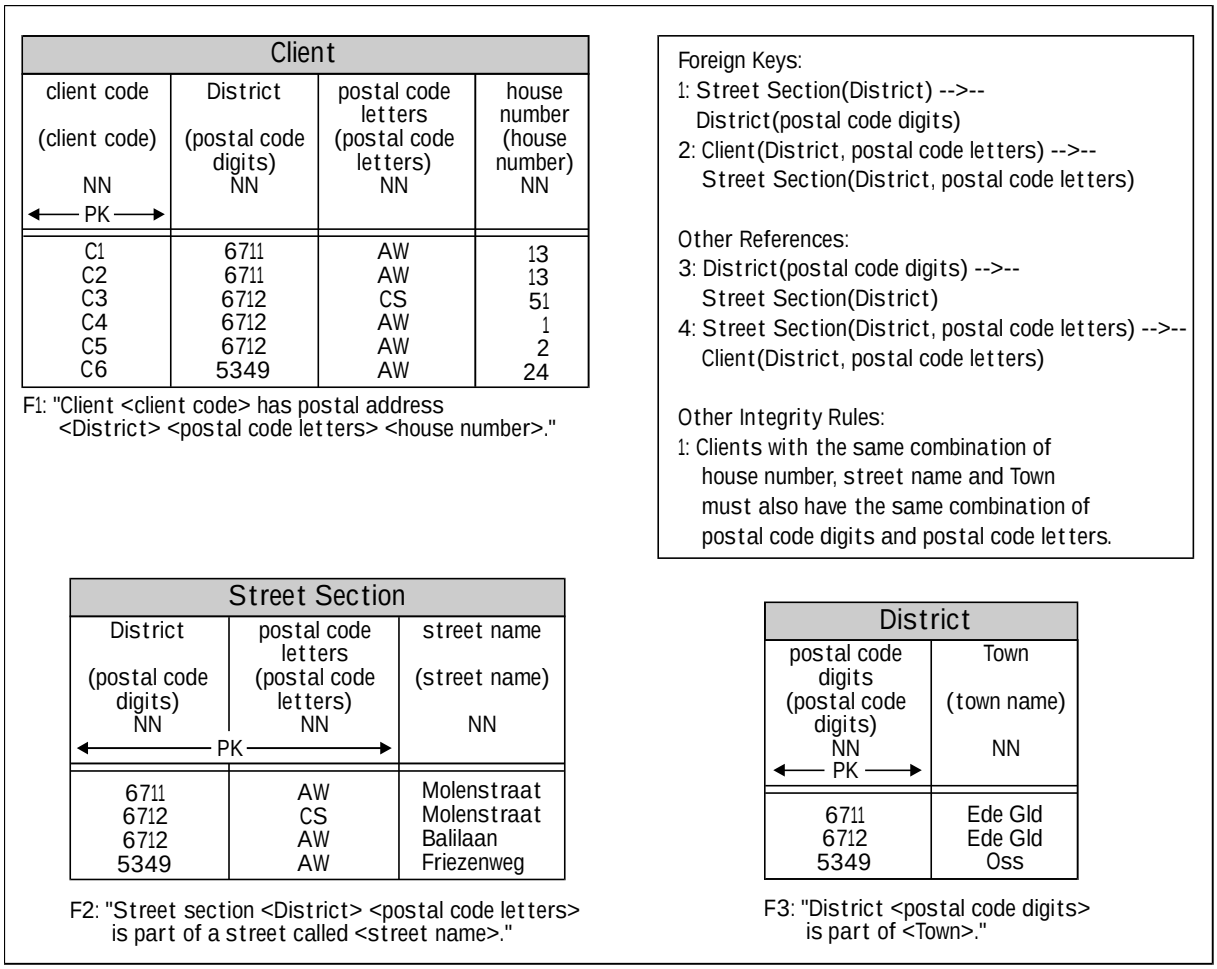


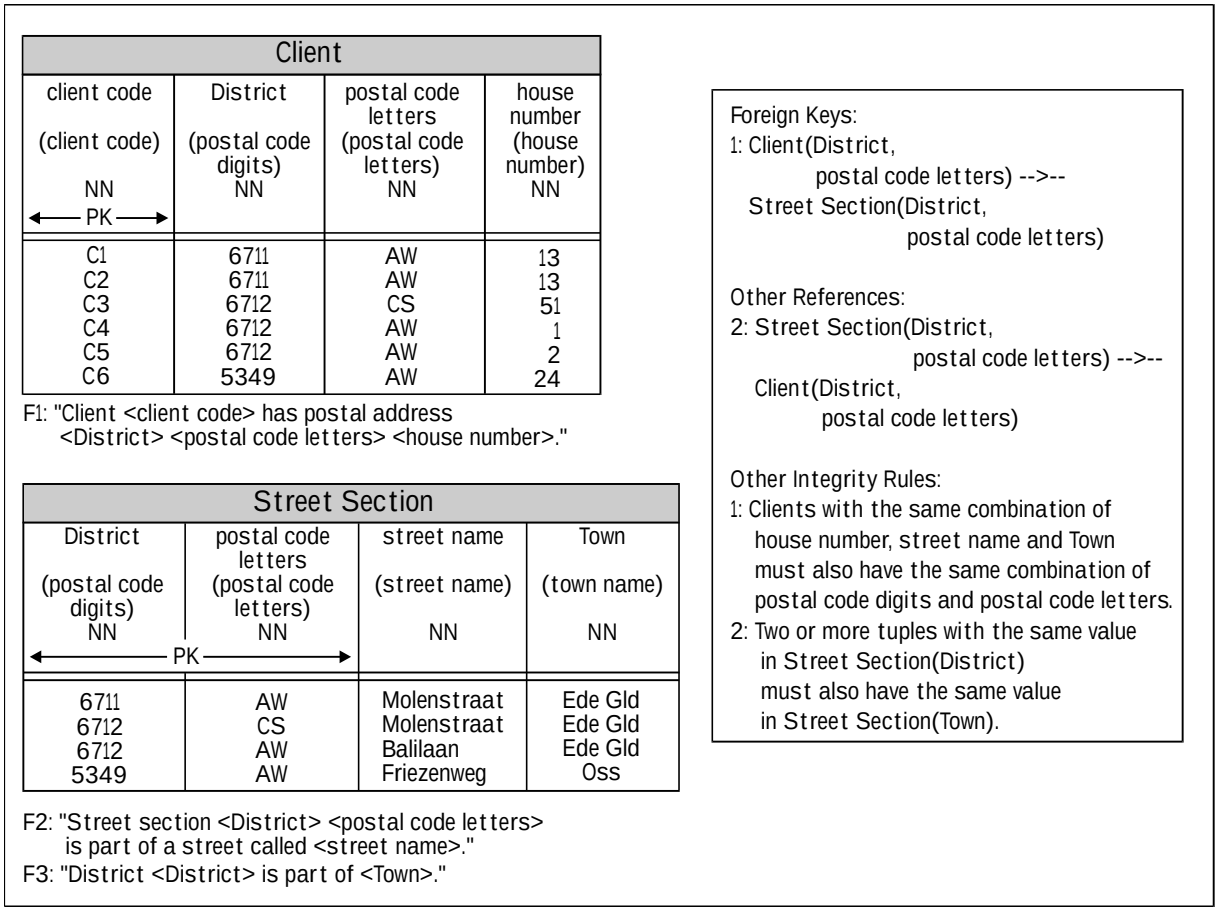
Figure 5.19: redundancy free relational schema

5.3.3 Denormalization

The tables in figure 5.19 are free from redundancy (adding a fact type about the location of towns in regions is therefore not necessary from a redundancy point of view). Each transaction concerning a postal address (retrieving, adding or deleting) involves, however, three tables with four references (many joins, many checks). The price for the elimination of all redundancy may be too high, especially if the number of clients is not very large. The number of tables can be reduced by consciously introducing redundancy in a controlled way, and keeping it under control subsequently. This process is called *denormalizing*, a term from the jargon of the Relational Model. The reverse process, namely the step-by-step process to remove redundancy, is called *normalizing* and leads to ever higher so-called *normal forms* (the tables in figure 5.19 are in fifth normal form).

First we remove table District by transferring its column Town to table Street Section. The result is shown in figure 5.20. This introduces redundancy: the fact that district 6712 is part of Ede Gld appears twice in the table (middle two tuples). To prevent that two different town names can now be entered yet for the same district, we must add an extra integrity rule. On the

other hand, there are now two references less. Because there is also a table less, we have a simpler schema in this way. (Table Street Section is now no longer in fifth normal form, but only in first normal form.)



Client

client code (client code) NN ← PK →	District (postal code digits) NN	postal code letters (postal code letters) NN	house number (house number) NN
C1	6711	AW	13
C2	6711	AW	13
C3	6712	CS	51
C4	6712	AW	1
C5	6712	AW	2
C6	5349	AW	24

F1: "Client <client code> has postal address <District> <postal code letters> <house number>."

Street Section

District (postal code digits) NN	postal code letters (postal code letters) NN	street name (street name) NN	Town (town name) NN
← PK →			
6711	AW	Molenstraat	Ede Gld
6712	CS	Molenstraat	Ede Gld
6712	AW	Balilaan	Ede Gld
5349	AW	Friezenweg	Oss

F2: "Street section <District> <postal code letters> is part of a street called <street name>."
F3: "District <District> is part of <Town>."

Foreign Keys:
1: Client(District, postal code letters) -->-- Street Section(District, postal code letters)

Other References:
2: Street Section(District, postal code letters) -->-- Client(District, postal code letters)

Other Integrity Rules:
1: Clients with the same combination of house number, street name and Town must also have the same combination of postal code digits and postal code letters.
2: Two or more tuples with the same value in Street Section(District) must also have the same value in Street Section(Town).

Figure 5.20: relational schema after one denormalization step

We can apply the same procedure again by removing table Street Section from figure 5.20 and transferring its columns ‘street name’ and Town ton table Client. The result is shown in figure 5.21. This introduces still more redundancy: the fact that district 6712 is part of Ede Gld is shown three times in the table, and the fact that district 6711 is part of Ede Gld is now shown twice. Furthermore, the two facts: “Street section 6711 AW as part of a street called Molenstraat.”, and “Street section 6712 AW is part of a street called Balilaan.” are recorded twice in the table. To prevent that two different town names can now be entered yet for the same district or that the same postal code can now have two different street names yet, we must add two extra integrity rules. On the other hand, there are no references left anymore. Because there is only one table left we have an even simpler schema in this way. (Table Client is now no longer in fifth normal form, but only in first normal form.)

Client

client code (client code) ◄— PK —► NN	street name (street name) NN	house number (house number) NN	District (postal code digits) NN	postal code letters (postal code letters) NN	Town (town name) NN
C1	Molenstraat	13	6711	AW	Ede Gld
C2	Molenstraat	13	6711	AW	Ede Gld
C3	Molenstraat	51	6712	CS	Ede Gld
C4	Balilaan	1	6712	AW	Ede Gld
C5	Balilaan	2	6712	AW	Ede Gld
C6	Friezenweg	24	5349	AW	Oss

F1: "Client <client code> has postal address <District> <postal code letters> <house number>."
F2: "Street section <District> <postal code letters> is part of a street called <street name>."
F3: "District <District> is part of <Town>."

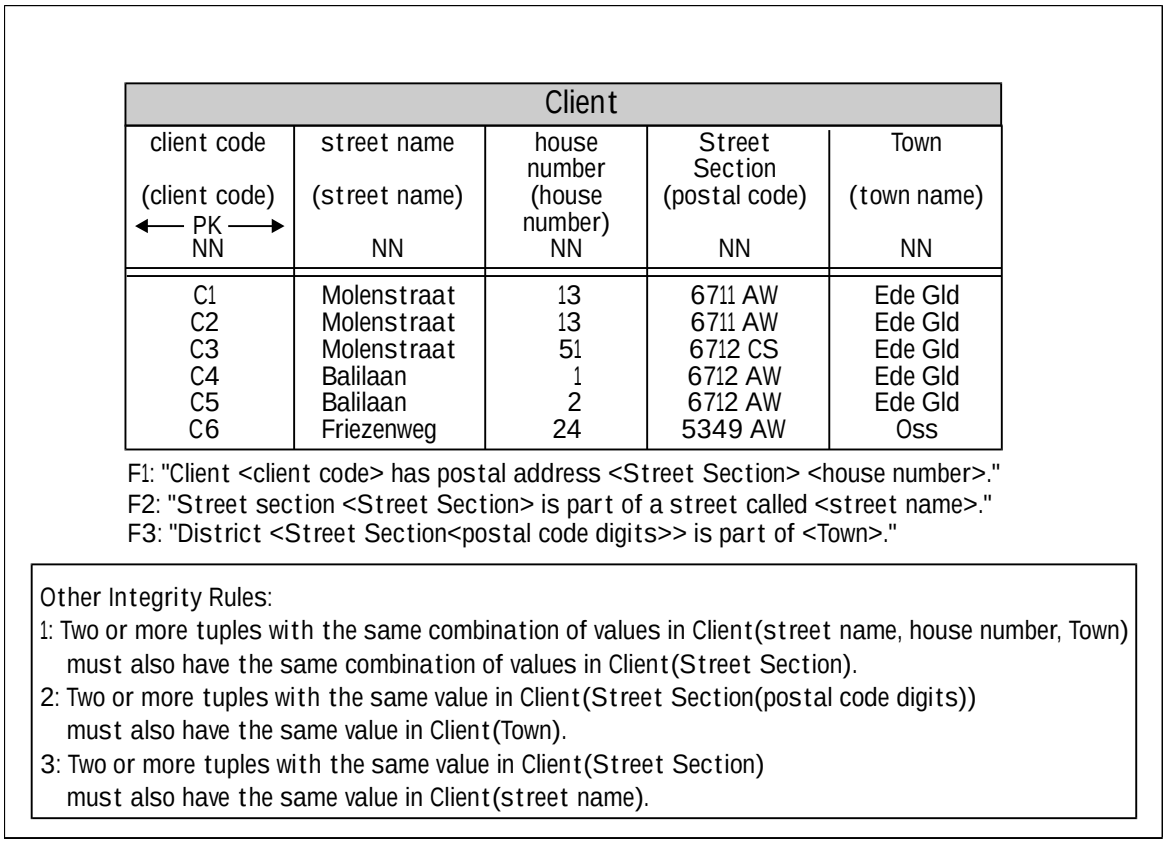
Other Integrity Rules:
1: Two or more tuples with the same combination of values in Client(street name, house number, Town) must also have the same combination of values in Client(District, postal code letters).
2: Two or more tuples with the same value in Client(District) must also have the same value in Client(Town).
3: Two or more tuples with the same combination of values in Client(District, postal code letters) must also have the same value in Client(street name).

Figure 5.21: relational schema after two denominalization steps

Finally, we can join the columns District and 'postal code letters' together in one column Street Section. This will simplify transactions concerning the whole postal code. The result is shown in figure 5.22. Column Street Section now contains composite (non-elementary) values: one can also say that column Street Section is now no longer *atomic*. We must now be able to have access to parts of columns (for transactions concerning districts): fact type expression 3 only uses the digit part from column Street Section. The notation in figure 5.22 with nested pointed brackets: <Street Section <postal code digits>> expresses this. In the implementation we must use subatomic operators as well (in SQL for example the LIKE operator). (Table Client is now even no longer in first normal form.)

Notice how much redundancy table Client contains. An intuitive verbalization (as in section 5.3.2 above) is often based on exactly this table structure.

Finally: denormalization does not change the modeled communication. Some fact expressions, however, will be regenerated more than once: redundancy. Which schema is to be preferred depends on statistical factors. If the number of clients is very large, then the balance will sooner tip to the redundancy free side than if the number of clients is small. Mixed forms between these schemas are also possible, such as joining the columns District and 'postal code letters' in figure 5.20.



Client

client code (client code) ⟵ PK ⟶ NN	street name (street name) NN	house number (house number) NN	Street Section (postal code) NN	Town (town name) NN
C1	Molenstraat	13	6711 AW	Ede Gld
C2	Molenstraat	13	6711 AW	Ede Gld
C3	Molenstraat	51	6712 CS	Ede Gld
C4	Balilaan	1	6712 AW	Ede Gld
C5	Balilaan	2	6712 AW	Ede Gld
C6	Friezenweg	24	5349 AW	Oss

F1: "Client <client code> has postal address <Street Section> <house number>."
F2: "Street section <Street Section> is part of a street called <street name>."
F3: "District <Street Section<postal code digits>> is part of <Town>."

Other Integrity Rules:
1: Two or more tuples with the same combination of values in Client(street name, house number, Town) must also have the same combination of values in Client(Street Section).
2: Two or more tuples with the same value in Client(Street Section(postal code digits)) must also have the same value in Client(Town).
3: Two or more tuples with the same value in Client(Street Section) must also have the same value in Client(street name).

Figure 5.22: relational schema after three denominalization steps

5.3.4 Handling the Exceptional Case

We now come back to the assumption we made in section 5.3.1 in the discussion of tuples 1 and 10 in figure 5.16. We assumed that only one street name belongs a certain postal code, but actually in very exceptional cases, two street names can belong to a postal code, if the one street contains only even house numbers and the other only uneven ones. In such cases the combination postal code + house number will still determine the street name. We could encompass all cases by changing the verbalization of fact type Street Section Location to: “Postal address 8651 BH 1 is located in a street called Holtropweg.” and “Postal address 8651 BH 2 is located in a street called Jonker Rispenstraat.”. The effect in figure 5.17 would be that role 8 from fact type Street Section Location is not played by object type Street Section, but instead by object type Postal Address. Nothing else would change in figure 5.17. In the relational schema from figure 5.19, table Street Section would then not arise, but we would have a table Postal Address instead, which compared with table Street Section would have an extra column ‘house number’ in its primary key. See figure 5.23, in which this table is shown, populated with the population from figure 5.17 together with two new examples. The references between table Postal Address and table Client now concern three columns.

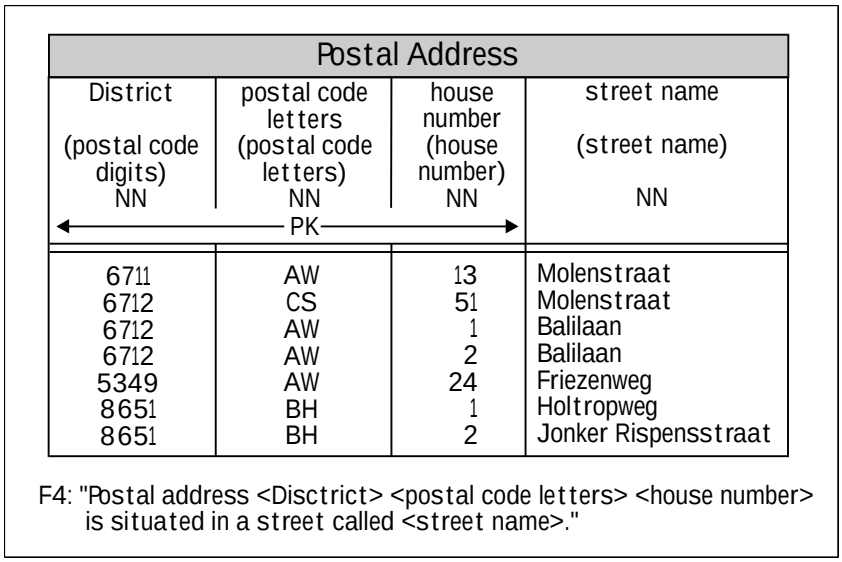


Figure 5.23: table Postal Address instead of table Street Sections

This model can accommodate the exceptions. This is achieved, however, at the cost of recording the house number needlessly in most cases. This way of modeling makes a very redundant impression: the fact that 'Balilaan' belongs to street section 6712 AW appears twice in the table. The more postal addresses having a postal code with only one street name are recorded the more serious this becomes. If the percentage of postal codes with more than one street name is very small, as is the case here (probably less than 0.1 %), this is clearly not a good way to model. The exceptions can better be handled separately.

We therefore include an extra fact type Exceptional Location in the IGD of figure 5.17, in which only the postal codes with two street names occur. See figure 5.24.

For clarity, we have not shown uniqueness constraint 9 graphically but textually in figure 5.24. There is now also a similar uniqueness constraint 11. In addition there are three other constraints:

C1: The notation Pop(8) means: the population of role 8. The notation Pop(14(1)) means: the part of the population of role 14 that comes from the population of role 1. C1 expresses that the street name must be known for each street section, either in fact type Street Section Location or in fact type Exceptional Location. C1 replaces totality constraint 4 from figure 5.17.

C2: This constraint says that each street section must either occur in fact type Street Section Location or in fact type Exceptional Location, but not in both.

C3: The notation COUNT(Pop(14(1))) means: the number of times that a value occurs in the part of the population of role 14 that comes from the population of role 1. This cardinality constraint says that each street section that occurs in fact type Exceptional Location must have exactly two street names.

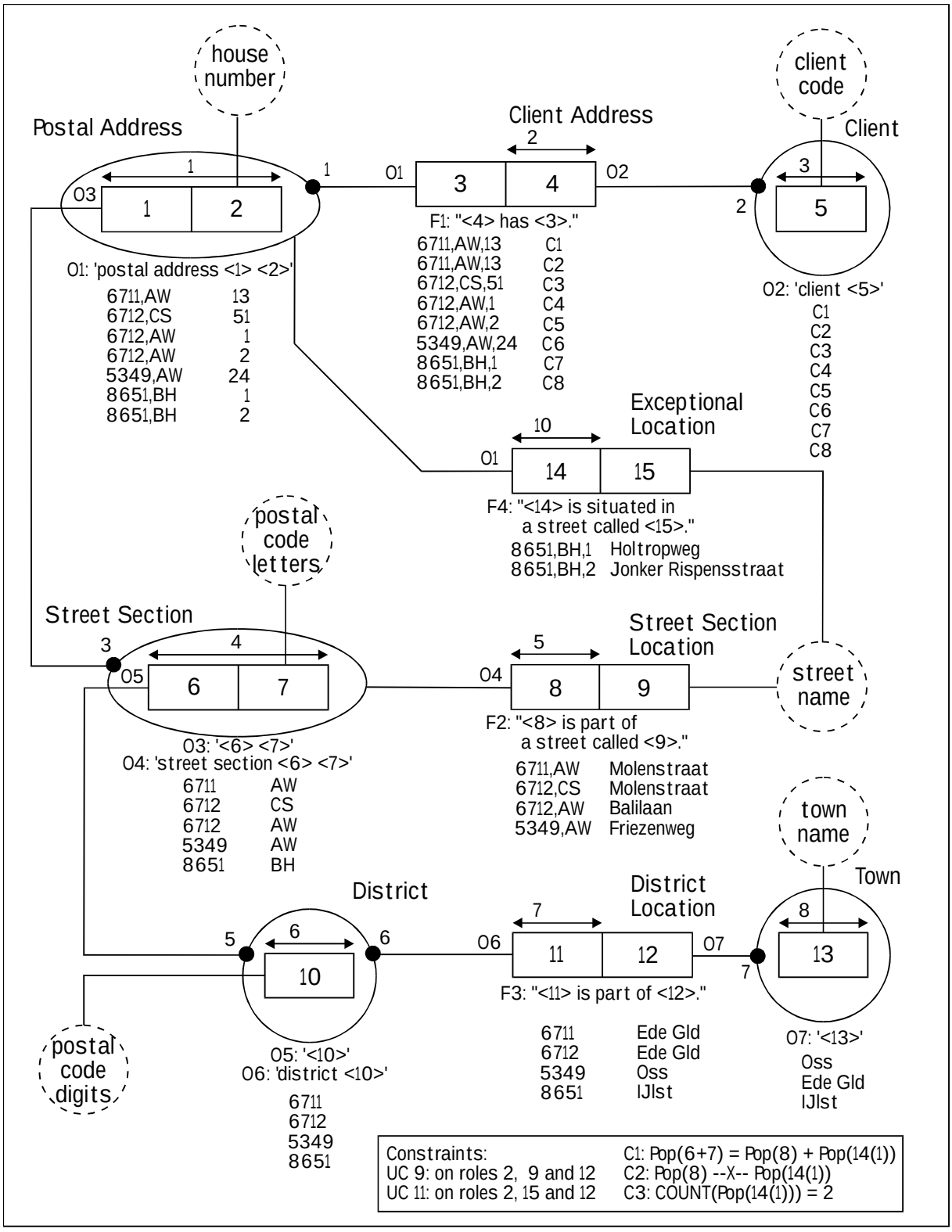


Figure 5.24: El-IGD addresses with postal code including exceptions

During the derivation of a logical relational schema from figure 5.24, it is best not to group fact type Exceptional Location away. The result is then an extra table, compared with the schema from figure 5.19, which is shown in figure 5.25. (If we would group the new fact type away, then the table Postal Address from figure 5.23 would arise again, this time with an extra optional column ‘street name’, in which only a very small number of tuples would contain a value.). Further, there is an extra foreign key reference and an extra other reference in the opposite direction between tables Client and Exceptional Location, on columns District, ‘postal code letters’ and ‘house number’. UC 11, C1, C2 and C3 transform to other integrity rules, which we do not give here separately.

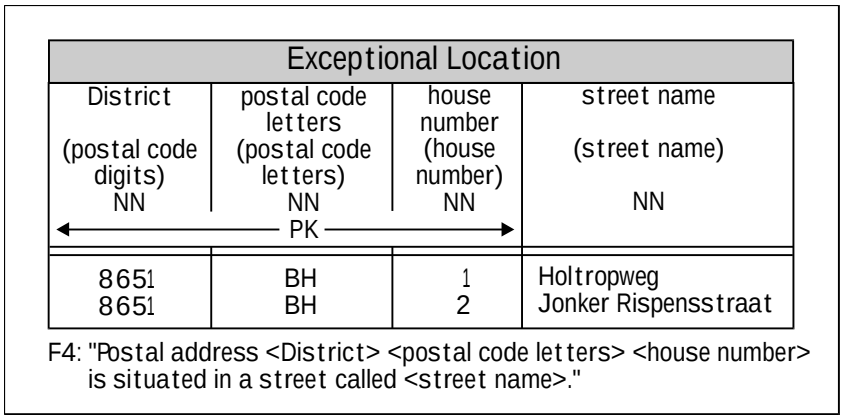


Figure 5.25: extra table Exceptional Location

The exceptions now come in a separate table, and only when we do indeed need them. Denormalization can be carried out again if desired, also with respect to the new table.

6

Specialization and Generalization

In this chapter we will discuss two related phenomena: in the first place *specialization* (the explicit determination of one or more new object types that represent special subsets of the objects of an existing object type) and in the second place *generalization* (the determination of a new generalized object type that is the union of two or more different existing object types).

The consequences of specialization and generalization for the derivation of a logical relational schema are discussed in chapter 7.

6.1 Specialization

Specialization is the often occurring phenomenon, that a role can only be populated with a special well-defined *subset* of the object type that plays the role. For example: let there be an object type Person that plays all sorts of roles in fact types in which the name, gender, date of birth and so forth are recorded, amongst which there is also a role in a fact type in which the total number of childbirths is registered. Of course, only *female* persons can occur in the population of this last role. We can therefore form a sub-object type Woman from object type Person, which contains only females. Such a sub-object type is called a *subtype*. The object type that the subtype is a part of is called a *supertype.*

In section 6.1.1, we will introduce the way specialization is modeled in FCO-IM by using a small example. A single supertype can have a complex network of subtypes, which in turn can have subtypes themselves as well. In section 6.1.2, we will discuss a practical method to find all the subtypes: the *subtype matrix method.* In section 6.1.3 we will give a second example in which subtypes arise with more than one supertype. We will close with some final remarks in section 6.1.4.

6.1.1 Declarative and Derivable Subtypes

To introduce the concept of specialization, we will consider a small UoD in this section. It concerns a small company with employees who are identified by an employee number. Various facts about these employees are to be recorded. There is an IGD in figure 6.1, which models this information.

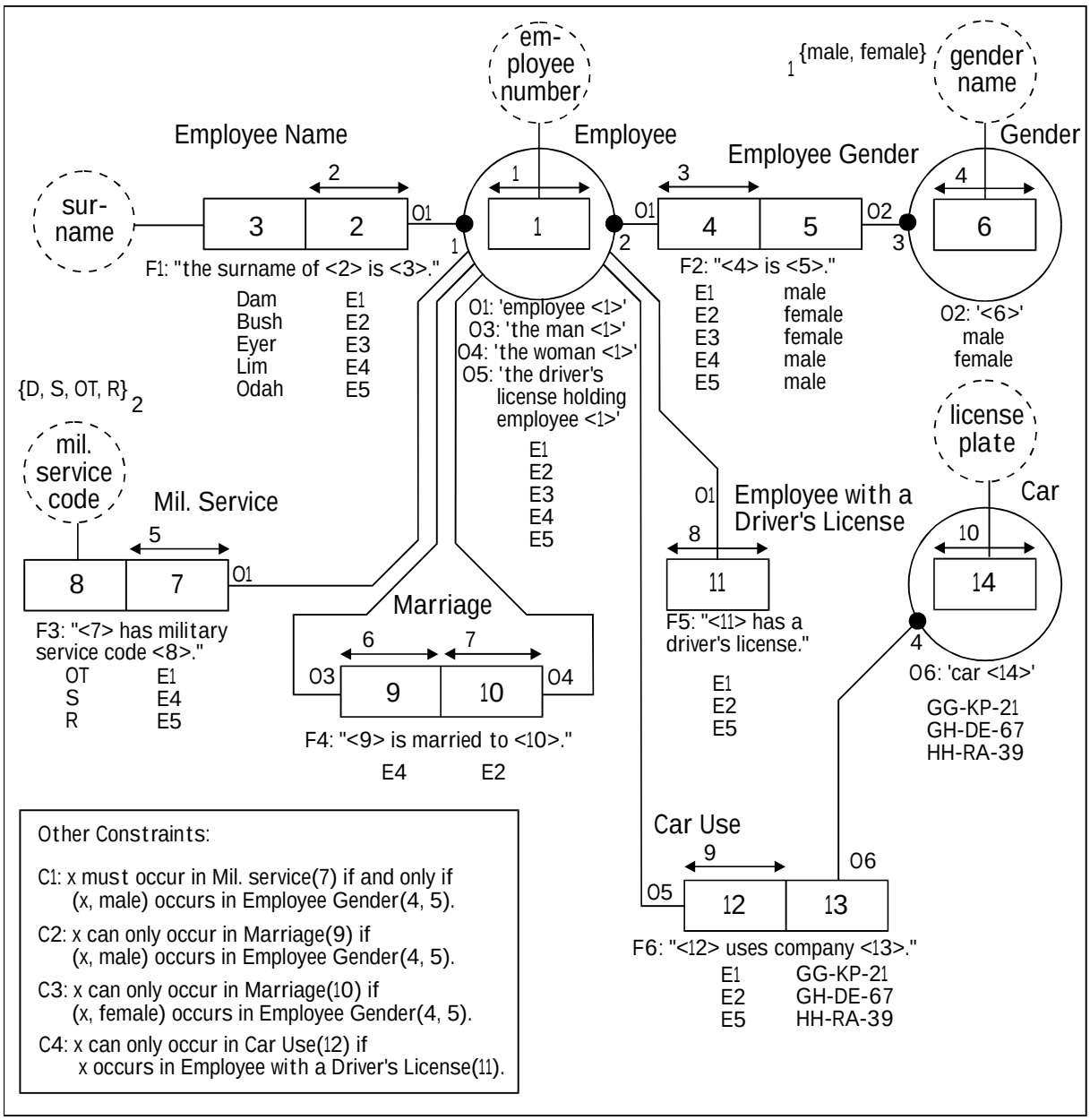


Figure 6.1: IGD before subtype analysis

Please note the following points in this IGD:

1 The surname and the gender must be known for all employees: see fact types Employee Name, with TC 1 on role 2, and Employee Gender, with TC 2 on role 4.

2 The military service status is recorded for all male employees: fact type Mil. Service, with constraint C1 on role 7. C1 says: if we know that a certain employee x is male (because the tuple (x, male) occurs in fact type Employee Gender), then we must know the military service status for x (then employee x must also occur under role 7 of fact type Mil. Service).

3 Any marriage between two employees is also recorded: fact type Marriage, with constraints C2 and C3. C2 says: role 9 can only be populated with men, but not all male employees have to be there. C3 is analogous for women.

4 Fact type Employee with a Driver's License registers which employees have a driver's license.

5 Only employees with a driver's license can receive a company car: fact type Car Use, with constraint C4. C4 says: only if we know that a certain employee x has a driver's license (because x occurs under role 11), then we can record which car x receives (then can employee x occur also under role 12 of fact type Car Use). The company policy is that all employees with a driver's license will get a car, but often some time elapses between obtaining a driver's license and receiving the car (a 'soft' TC should be used here, see final remark 4 in section 3.4).

In short: roles 7 and 9 can only be populated with male employees, role 10 with female employees, and role 12 only with driver's license holders. The left half of figure 6.2 contains the classification and qualification from which the IGD of figure 6.1 follows, so before the subtype analysis takes place. We will explain the right half, which contains the situation after determining all the subtypes, gradually in the course of the discussion.

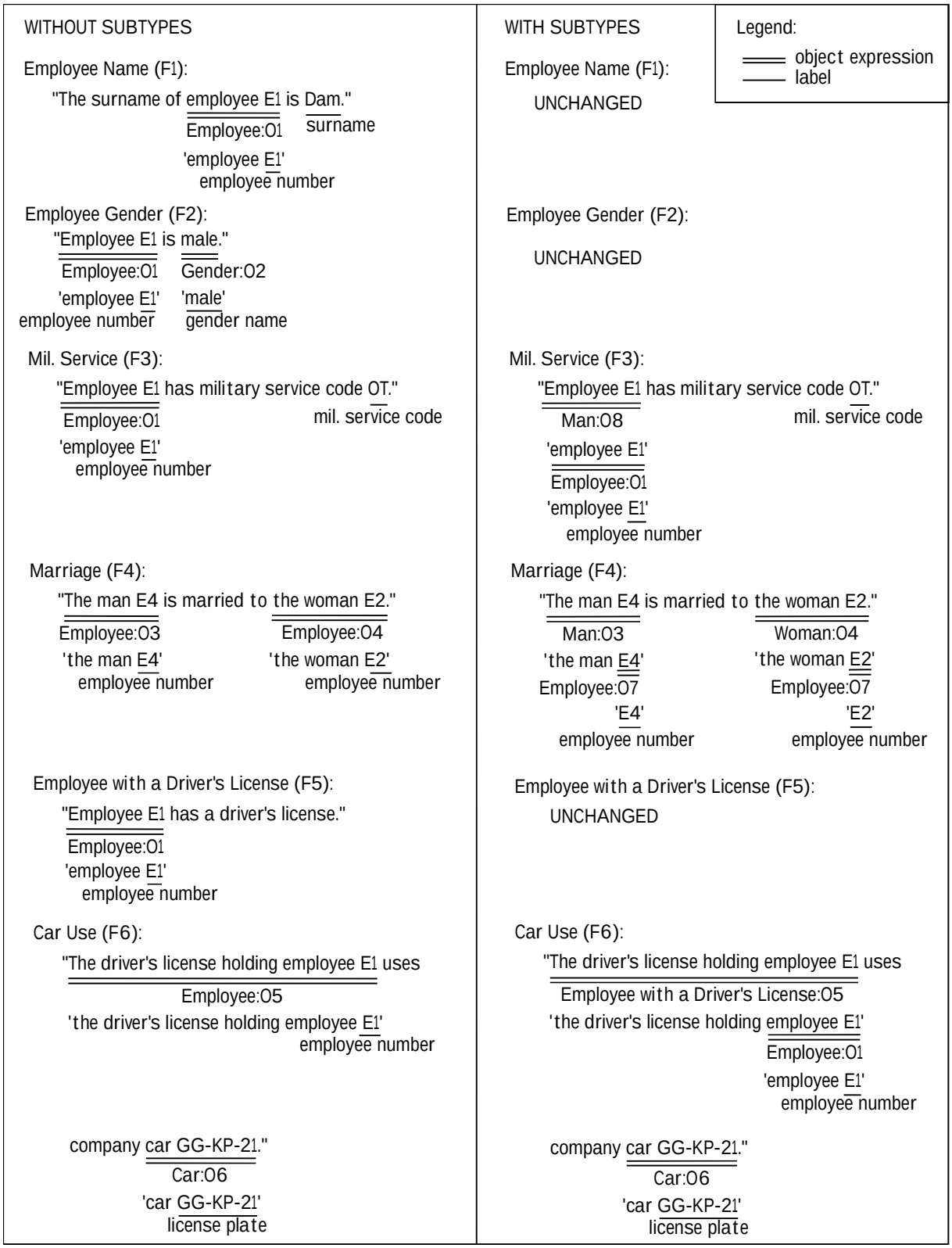


Figure 6.2: classification and qualification before and after introducing subtypes

6.1.1.1 Declarative Subtypes

We will consider fact types Employee with a Driver's License and Car Use first. According to constraint C4 in figure 6.1, only employees who have a driver's license are entitled to a company car. But all the employees who have a driver's license are in the unary fact type Employee with a Driver's License. Therefore, role 12 can better be played directly by (the nominalization of) this fact type Employee with a Driver's License: see figure 6.3.

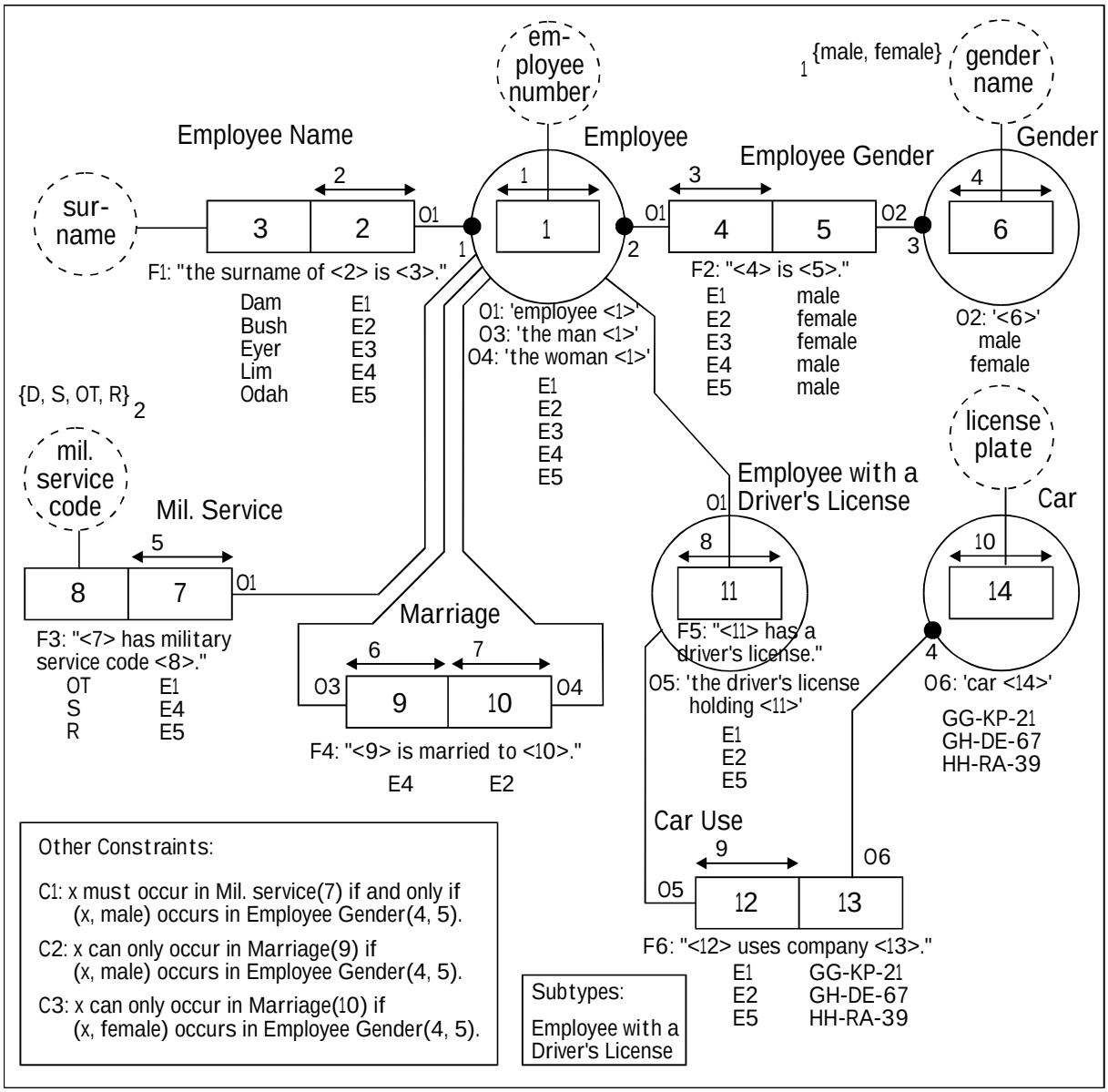


Figure 6.3: IGD with declarative subtype

Object type Employee with a Driver's License can be considered as a special well-defined subset of object type Employee, which contains all employees who have a driver's license. Such a sub-object type is called a *subtype*. The object type of which it is a part is called a *supertype*. Subtype Employee with a Driver's License now plays role 12, which can after all be populated only with driver's license holders. Now constraint C4 can be dropped, because it

is equivalent with letting role 12 be played by object type Employee with a Driver's License. For clarity, we include a list of all subtypes in the IGD; this list can be displayed by the FCO-IM tool as well.

Because all the employees who have a driver's license are listed in the unary fact type Employees with a Driver's License, it is clear that fact type Employees with a Driver's License already is a subtype in figure 6.1: its population is a well-defined subset of the population of Employee, namely those employees of whom it is explicitly declared that they have a driver's license. Unary fact types with a non-lexical role are therefore subtypes by definition, whether they play roles (that is to say are nominalized) or not. Therefore, the same list of subtypes can be included in figure 6.1 as in figure 6.3; the FCO-IM tool will generate it there as well if desired. By the way, we will come across subtypes with non-unary fact types in section 6.1.3.

If the fact type of a subtype is not derivable, as is the case here with subtype Employee with a Driver's License in figure 6.3, then it is a *declarative* subtype.

The communication can always remain unchanged when subtypes are being determined, only the classification and qualification changes. So the fact type expressions stay the same and only the object type expressions must be adapted. O5 in figure 6.1 reads: 'the driver's license holding employee <1>'. The part 'employee <1>' from this OTE is the same as O1, so that O5 can be moved to the subtype in shortened form reading: 'the driver's license holding <11>', in which there is now a reference to role 11 (see figure 6.3). The corresponding changes in classification and qualification are shown in the right half of figure 6.2, for fact type Car Use.

The content of an object type expression such as O5 in figure 6.1: 'the driver's license holding employee <1>' is for that matter an indication that there must be a subtype involved here. Each concrete object expression such as 'the driver's license holding employee E1', clearly contains a separate fact, namely the fact that this employee has a driver's license. It is therefore better to verbalize these separate facts independently. An analogous example: the sentence "The second-hand car GH-KL-12 has done 45000 km." contains an independent fact: "Car GH-KL-12 is second-hand." and leads to a subtype Second-Hand Car with supertype Car.

We give an advise here for declarative subtypes, intending to prevent that such separate facts remain implicitly buried in OTEs too much, and also to avoid that all sorts of superfluous subtypes will be created (see also final remark 1 in section 6.1.4):

A declarative subtype preferably has at least one fact type expression.

This is an advise and not a requirement, because the rule given in section 3.4 still applies, which says that a fact type expression can only be missing from an object type if at least one totality constraint applies to the roles played by this object type. Therefore, the fact type expression cannot be absent from Employee with a Driver's License in figure 6.3, but even if a TC would indeed apply to role 12, we still advise to add an FTE to Employee with a Driver's License. Subtypes that do not play any role (unary non-nominalized fact types) obviously cannot do without an FTE, because each non-nominalized fact type must have at least one FTE. So the FTE for Employee with a Driver's License cannot be missing from

figure 6.1. This is another reason to have an FTE in figure 6.3 for Employee with a Driver's License, because of the equivalence of the IGDs in both figures.

6.1.1.2 Derivable Subtypes

In figure 6.3, it would now seem natural if roles 7 and 9 were played by a subtype Male Employee and if role 10 were played by a subtype Female Employee, both with Employee as supertype. Such unary fact types, however, are not there. We do know which employees are male and which are female, though: this information is available in fact type Employee Gender. We can therefore add two derivable unary fact types, with corresponding fact type expressions, see figure 6.4.

We will call these fact types Man and Woman for short. They are subtypes: their single role is non-lexical. The corresponding derivation rules are displayed in figure 6.4 at the bottom.

For clarity we also draw totality constraint 7 and exclusion constraint 1 between roles 15 and 16, although they are actually redundant: TC 7 follows from TC 2 and XC 1 from UC 3. To indicate the redundancies we append an asterisk both to the derivable fact types and to the redundant constraints. We choose a short object type expression O7 for object type Employee in the verbalization of the derivable fact types, which will prove to be convenient later as well.

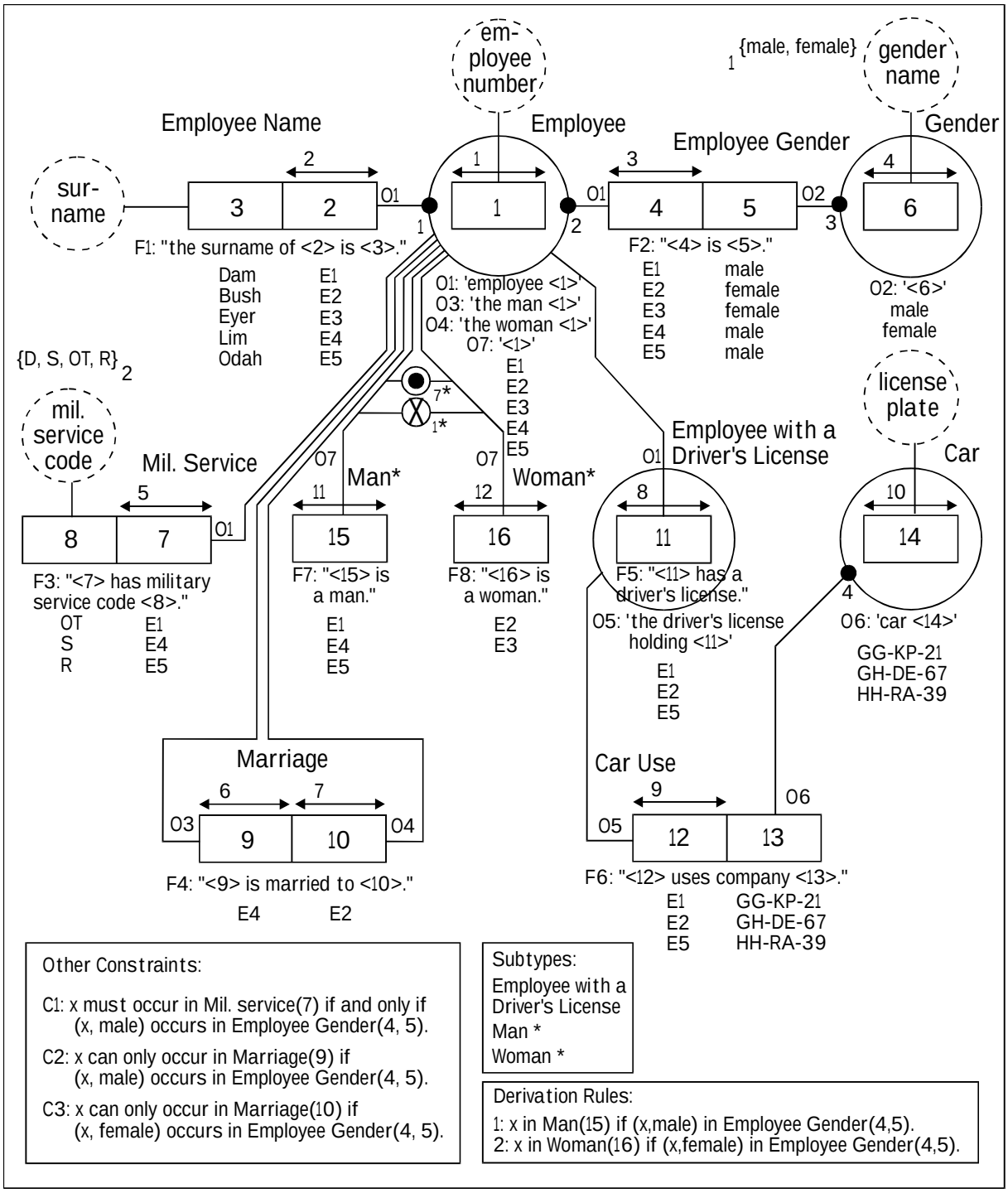


Figure 6.4: IGD with extra derivable fact types

In the same way as we did for the declarative subtype Employee with a Driver's license, we will now let roles 7 and 9 be played directly by (the nominalization of) fact type Man, and role 10 by (the nominalization of) fact type Woman, see figure 6.5. Object type expression O4 is now moved to object type Woman, and O3 to object type Man.

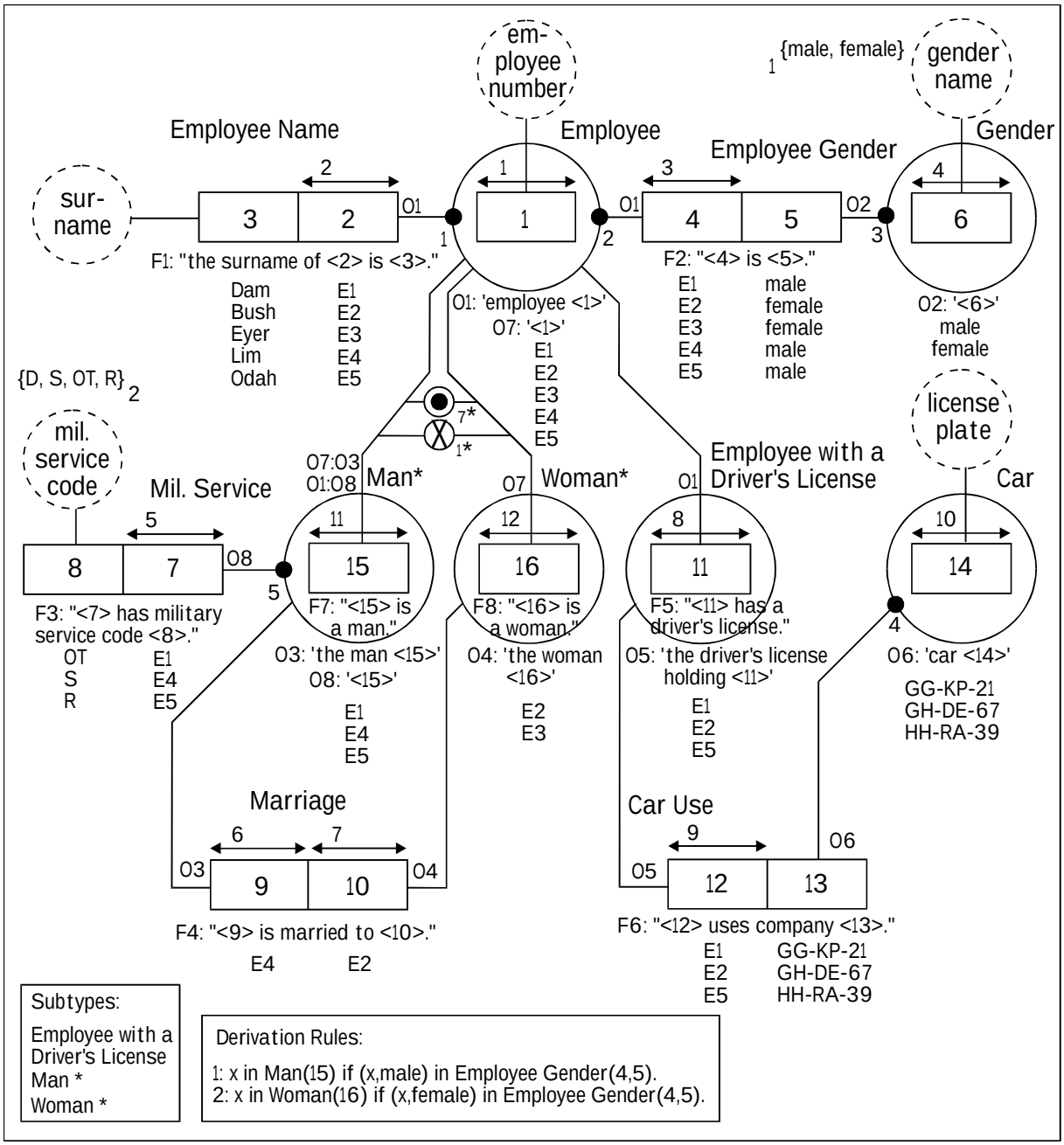


Figure 6.5: IGD with derivable subtypes Man and Woman

If we want the IGD to model still exactly the same communication, then the word 'employee' from object type expression O1 must eventually end up in fact expressions of fact type expression F3, whereas the words 'the man' from O3 must end up in expressions of F4. In figure 6.5 this is done by adding O8 to Man, and by using subtle substitution for role 15 (see section 2.10): O1 is to be substituted into O8 and O7 into O3. By the way, we would obtain a simpler result if we would change the communication a bit by replacing fact expressions such as "Employee E1 has military service code OT." with fact expressions such as "The man E1 has military service code OT.". In that case O8 could be dropped and only O7 would apply to role 15, as it does to role 16 of fact type Woman. This is carried out in figure 6.8 at the end of

section 6.1.2. The corresponding changes in classification and qualification are shown in the right half of figure 6.2, for fact types Mil.Service and Marriage.

TC 5 on role 7 in figure 6.5 expresses that the military service code must be known for all male employees. Roles 9 and 10 get no TC even now. Constraints C1, C2 and C3 from figure 6.4 can now be dropped because they are equivalent with letting roles 7, 9 and 10 be played by the derivable subtypes, together with TC 5.

Subtypes with a derivable fact type are called *derivable subtypes.*

It is important to avoid circular arguments when formulating the derivation rules for the derivable subtypes. For example: role 7 can only be populated with male employees. A circular derivation rule for subtype Man would be: "x in Man(15) if x in Mil. Service(7)": role 7 can only contain men, but which employees are men would here be *defined* as those employees that occur in the very same role 7. (So the army really does make a man out of you after all, even if you actually are a woman?) The derivation rules in figure 6.5 avoid such tautologies by stating the derivation rules in terms of a fact type that is not played by the subtypes.

To avoid circular reasoning as in the example above, the following well-formedness rule applies concerning derivation rules:

> Each derivable subtype must have a derivation rule, in which only roles can occur (apart from the role of the subtype itself) from fact types that
> - either are played by one or more of its supertypes
> - or are found in one or more of its supertypes

That is: derivation rules must concern only fact types 'higher up' in the subtype network.

Subtype Man does not need to have any fact type expression, because of totality constraint 5, but subtype Woman does because of the absence of a TC on role 10. Subtype Employee with a Driver's License also does not need to have an FTE, but falls under the advice for declarative subtypes from section 6.1.1.1 (see final remark 1 in section 6.1.4).

The subtypes in figure 6.5 now make clear at first sight what can only be gathered with difficulty from constraints C1, C2, C3 and C4 in figure 6.1.

6.1.2 The Subtype Matrix Method

A subtype can have subtypes itself (and so be a supertype for these), a supertype can have more than one subtype and a subtype can have more than one supertype, and so a complex subtype network can exist. It is often very difficult to determine the structure of such a complex subtype network by intuition. There is a systematic method to establish the structure, which was developed in practice: the subtype matrix method.

In this section we will introduce the subtype matrix method by applying it to the example from section 6.1.1, of which we already know the (simple) subtype structure. In doing so we will also give a step-by-step operational procedure. We start from the IGD in figure 6.1, which only contains the declarative subtype Employee with a Driver's License, which plays no role yet.

Which non-lexical object types should we examine for subtypes? In principle we should look at every object type that plays at least one role without a single role totality constraint. For each such object type we will construct a subtype matrix. (In practice, however, we will leave out of consideration typical attribute-like object types (such as Sum of Money), which play roles without a single role uniqueness constraint, and which have just one totality constraint on all their roles together, even though subtypes can also be present there.) So in figure 6.1, we will check only object type Employee.

Step 1

We will construct a subtype matrix for object type Employee, see figure 6.6. A matrix is just a two-dimensional table: it has rows and columns.

First we make the columns. The first column is for the concrete examples of the objects of the object type concerned. Above this column we write the name of the object type, which also makes it clear which object type is being considered in the matrix; in this case object type Employee. Next we make the other columns: one column for each role played by the object type; in this case roles 2, 4, 7, 9, 10, 11 and 12. If the IGD would already contain subtypes that play roles themselves, then we must include columns for these roles in the subtype matrix as well; in this case that does not occur (if we had started from figure 6.3, then we would have to make a column for role 12 after all). We write the role number above each column, and for clarity we mention between brackets from which fact type the role comes. For homogeneous fact types such as Marriage a further description can be helpful (roles 9 and 10). We separate the first column from the other columns with a double line to emphasize the difference.

Next we make the rows. We populate the first column with a significant example population, that is a set of concrete example objects (here: employees) that contains all possible situations of facts known or not known about these objects. In section 6.1.4 we will come back to the problem how to obtain such a *significant example population*. Figure 6.6 has been supplied with a significant population. There is a row for each concrete example.

Step 2

In steps 2 and 3 we will fill in the remaining columns. Each cell (i.e.: intersection of a column and a row) will contain either a cross (X) or a dash (-). A cross is entered into a cell if a concrete object actually occurs in the population from a role. In our example, employee E5 for instance occurs in the population of roles 2, 4, 7, 11 and 12, so a cross is entered into these columns in the row for E5. If an object appears in the population of a role, then obviously a fact is known about this object that is considered relevant to record. A cross therefore means: the role is *relevant* for the concrete object. In step 2, we enter all the crosses that follow from populating the roles with the complete significant set of concrete examples, see the top half of figure 6.6.

Matrix after step 2:

Employee	role 2 (Employee Name)	role 4 (Employee Gender)	role 7 (Mil. Service)	role 9 (Marriage (Man))	role 10 (Marriage (Woman))	role 11 (Emp. with Driv. Lic.)	role 12 (Car Use)
E1	X	X	X			X	X
E2	X	X			X	X	X
E3	X	X					
E4	X	X	X	X			
E5	X	X	X			X	X

Matrix after step 4:

Employee	role 2 (Employee Name)	role 4 (Employee Gender)	role 7 (Mil. Service)	role 9 (Marriage (Man))	role 10 (Marriage (Woman))	role 11 (Emp. with Driv. Lic.)	role 12 (Car Use)
E1	X	X	X	X (!)	-	X	X
E2	X	X	-	-	X	X	X
E3	X	X	-	-	X (!)	X (!)	-
E4	X	X	X	X	-	X (!)	-
E5	X	X	X	X (!)	-	X	X
	A	A	A	C	C	D	A	B

Figure 6.6: subtype matrix for object type Employee

Step 3

In step 3, we consider the cells that are still empty after step 2. For each empty cell we must establish whether the role is relevant after all for the object (in which case a cross is entered yet), or that the role is not relevant (in which case a dash is entered). Interviews with the domain experts are indispensable here.

Below we will first give a formal definition of the concept of a relevant role, and next we will

discuss all the empty cells in the subtype matrix in turn, referring back to this definition.

Formal definition of a *relevant role*:
a role is relevant for an object if:

1 the object appears in the population of the role.
2 the object does not appear in the population of the role, but the domain experts state that a proposed concrete fact would be recorded at once (so the object would then indeed appear in the population of the role), without having to:
   a add an other fact to a declarative subtype.
   b change an other fact from a fact type that is not a declarative subtype.

If a role does not satisfy the above definition then it is not relevant.

We will now discuss all the empty cells in the subtype matrix, starting with the column for role 7 with empty places for E2 and E3. The analyst asks a domain expert: "If I would tell you that employee E2 has military service code R, then would you record this?". The domain expert replies: "No.". Analyst: "Why not?". Domain expert: "E2 is a woman and we only register the military service code for men.". Conclusion: role 7 is not relevant for E2 (the role does not satisfy the definition) and the analyst enters a dash in the cell. It is clear that the same applies to the empty cell for E3, since E3 is also a woman. Please note that the questions about the reasons for not recording the proposed fact give important indications about the nature of the subtypes. This is why the analyst must always ask for these reasons if the expert does not give them spontaneously. Now suppose that the expert had replied to the first question: "Yes, I would if it is a man." Then the analyst would find that he would have to change a fact from fact type Employee Gender: "Employee E2 is female." would have to be changed to "Employee E2 is male.". In that case a dash would still be entered in the cell because point 2b from the definition would apply here: fact type Employee Gender is not a declarative subtype.

Next we consider the columns for role 9 and role 10, both from fact type Marriage. It is already clear from the verbalization that the population of role 9 is to contain only men, and the population of role 10 is to contain only women (see F4 in figure 6.1). So the analyst enters dashes for E2 and E3 under role 9, and for E1, E4 and E5 under role 10. The analyst and the domain expert then have the following dialogue:

Analyst: "Suppose that E88 is a woman. Would you then record the fact: "The man E1 is married to the woman E88." as soon as the two get married?"
Expert: "Of course."
Analyst: "And also the fact: "The man E5 is married to the woman E77.", if E77 is a woman?"
Expert: "Yes."
Analyst: "And also the fact: "The man E99 is married to the woman E3.", if E99 is a man?"
Expert: "Yes."

Conclusion: role 9 is relevant for E1 and E5, and role 10 is relevant for E3, even though these employees presently are not married to another employee. The corresponding cells therefore get a cross after all. In figure 6.6, we emphasize these 'hidden' crosses, which only come to light via interviews, by adding an exclamation mark between brackets: '(!)'. The exclamation mark has no other meaning and can be omitted: it is only of interest whether there is a cross or a dash in a cell.

Now let us look at the empty cells in column 11. The analyst asks: “If I would tell you that employee E3 has a driver’s license, would you record that?”. The domain expert: “Yes, and then I would also assign her a company car.”. The analyst concludes that role 11 is relevant for E3 as well and enters a cross in the subtype matrix. Note that point 2a from the definition does not apply: the expert would also add another fact to a fact type, but not to a declarative subtype. Point 2b does not apply either because the domain expert would not change an other fact in a fact type, but only add one. The analyst treats E4 analogously and gets the same answer, so that yet another cross appears.

Finally, we deal with the empty cells in the column for role 12.

Analyst: “If I share the fact with you: “The driver’s license holding employee E3 uses company car KK-LL-33.”, would you record that?”

Expert: “At this moment it is not known to me that E3 has a driver’s license. I would check if that is indeed true, and if yes, then I would also register which car E3 uses, because we only assign cars to employees with a driver’s license.”

Obviously another fact is to be recorded as well in fact type Employee with a Driver’s License (namely: “Employee E3 has a driver’s license.”), if the proposed fact is registered. Fact type Employee with a Driver’s License is, however, a declarative subtype (unary, non-derivable fact type with non-lexical role), so that point 2a from the definition applies and the role is *not* relevant for E3. The same applies to the empty cell for E4. Both empty cells therefore get a dash and not a cross. This completes step 3.

Step 4

We can now tell which subtypes there are from the subtype matrix. Roles from columns with the same number of crosses in the same rows are played by the same subtype. In this step, we give all columns a symbolic name; columns with the same pattern of crosses get the same name. We will give meaningful names to the subtypes later.

There is always at least one column without dashes (in rare situations it will be only the first column with the concrete examples, but usually there are other columns without dashes as well). We mark all these with the letter A as a symbolic name; here: the first column and the columns for roles 2, 4 and 11. Next we consider the column(s) with the second highest number of crosses. Here these are the columns for roles 7, 9 and 12. Those for roles 7 and 9 have the same pattern of crosses and so receive the same name, but the column for role 12 has a different pattern and so receives another name. We arbitrarily give the column for role 12 the letter B, and the other two receive the letter C (we could have done this the other way around as well). The last column gets the letter D. So we name the columns in order of the number of crosses: the columns with the most crosses first, and those with the least crosses last. The order does not matter for columns with the same number of crosses but in different rows. The result of step 4 is shown in the bottom half of figure 6.6.

The letter A always represents the highest supertype, which is the object type from which we started: here: Employee. So next to the supertype there are three subtypes: B, C and D.

Step 5

In this step, we will determine the structure of the subtype network (in this simple example the structure is very easy; in section 6.1.3 we will discuss a slightly more complex example). First we will give an operational procedure, which we will next illustrate with an example.

Procedure: We order all symbolic names alphabetically (here: A up to and including D). Next, we systematically compare all the patterns of crosses with one another. If all the crosses from a certain column P also appear in another column Q (in other words, if the crosses from P are a *subset* of those from Q), then P is a subtype of Q. Notation: P $\rightarrow$ Q. When comparing the patterns of crosses, we successively consider all subtypes in alphabetical order (so here: first B, then C and finally D). For each subtype, we compare its pattern of crosses with the patterns of all the subtypes to the *left* of this subtype and with the pattern of A, starting with its left neighbor and continuing further leftward until we reach A (so here we successively compare B with A, C with B, C with A, D with C, D with B and D with A). We write down all the subset relationships we find, leaving out redundant relationships: if P $\rightarrow$ Q and Q $\rightarrow$ R, then P $\rightarrow$ R applies automatically, but because this last relationship follows from the two others it is redundant and we do not write it down.

The results of this procedure applied to our example are shown in figure 6.7, at the top. First we compare B with A. All the crosses in B also occur in A, so B is a subtype of A (this is even always the case automatically). In figure 6.7 this is indicated by subset arrow 1. Next we take C. First, we compare C with B. The crosses in C do not all occur in B (B does not have a cross for E4), so C is not a subtype of B. Then we compare C with A, which does produce a subset relationship: arrow 2 in figure 6.7. Finally, we deal with D. D is not a subtype of C, and neither of B, but it is a subtype of A (arrow 3 in figure 6.7). The subtype network here is very simple: all the subtypes are immediate subtypes of A, but in general this is not always the case (see section 6.1.3).

The procedure described above requires a minimum number of comparisons because of the ordering by the number of crosses during the naming of the columns. Without this ordering we would have to compare each letter with all the others, requiring n(n-1) comparisons for n different letters. With this ordering we only have to make ½n(n-1) comparisons: exactly half as many.

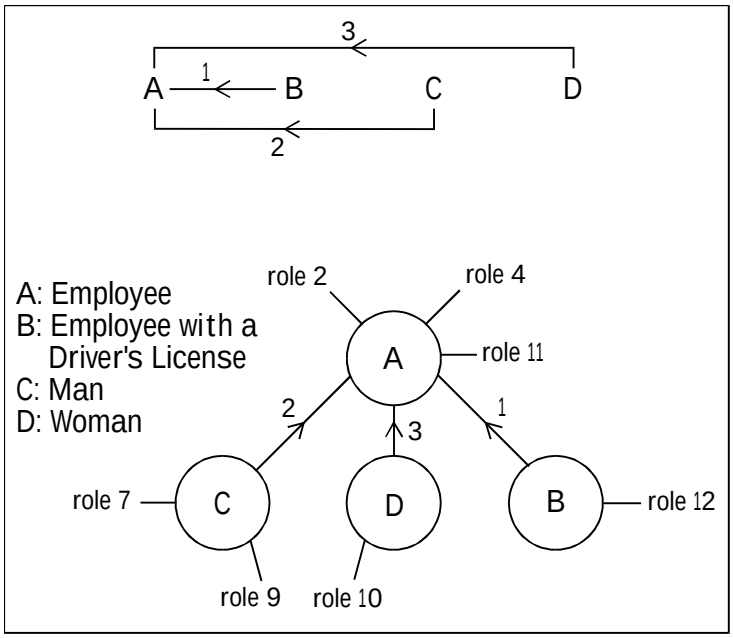


Figure 6.7: result of steps 5 and 6

Step 6

As an intermediate step in drawing the full IGD, a sketch such as in figure 6.7 at the bottom is useful. All supertypes and subtypes are drawn as closed circles (they are nominalized fact types after all), connected by the subset arrows found in step 5. For clarity, we always draw the arrows upward. Because there are no redundant subset relationships, we only see the *direct* supertype - subtype relationships. We read directly from the subtype matrix (see figure 6.6) which roles are played by which subtype. For example: role 7 and role 9 are played by C, since this letter is written under the columns for roles 7 and 9.

Meaningful names must now be given to the subtypes. If the analyst has always asked in step 3 why a proposed fact is not to be recorded, then the nature of the subtypes is usually already clear (if not, then further interviews with the domain expert will throw light on the matter). 'A' always stands for the highest supertype (here: Employee). The remaining names are obvious and are listed in figure 6.7.

Step 7

Now that we know which subtypes there are, we can complete the IGD. Each subtype must be supplied with a unary fact type (see section 6.1.3 for the case of a subtype with than one supertype).

We must first deal with all the declarative subtypes, which are already present in the IGD we started from, because no extra fact types are to be introduced for them. Each non-derivable unary fact type with a non-lexical role is a declarative subtype. Fact type Employee with a Driver's License, with role 11, is a declarative subtype in our example (see figure 6.1). The existence of all subtypes follows from the subtype matrix method also however, including the declarative ones that are already there. In our example, B stands for the subtype Employee with a Driver's License. Fact type Employee with a Driver's License therefore becomes the fact type in subtype Employee with a Driver's License. In figure 6.7 it can be seen that role 11 (from fact type Employee with a Driver's License) is played by A (Employee). This is consistent with the fact that subtype B (Employee with a Driver's License) is a direct subtype of A. In short: role 11 is still played by A, but finds itself inside B. See figure 6.3 for the situation after dealing with all declarative subtypes.

The remaining subtypes are all derivable subtypes. For each derivable subtype, a derivable fact type is made, which is supplied with a derivation rule, and with a fact type expression if desired (because these FTEs model derivable sentences, no new information is added; furthermore, a subtype with a totality constraint on any role played by it, does not need to have an FTE). Each derivation rule must satisfy the well-formedness rule from section 6.1.1.2: it can only concern fact types 'higher-up' in the subtype network. See figure 6.5 for the situation after dealing with all the derivable subtypes.

If no correct derivation rule can be found for a certain subtype, then the verbalization was incomplete: information is missing that is evidently needed. There are two remedies: either the subtype is dropped (the necessary information is not to be recorded), or the necessary information is added yet, in the form of an extra fact type (which can also be a new declarative subtype).

As an optional step, the communication could be improved by introducing a suitable object type expression for each subtype that does not yet have one. See the comment about this in the fourth paragraph in section 6.1.1.2. The communication then becomes clearer because the fact expressions becomes more explicit, and the substitutions become simpler. We therefore strongly recommend this, see figure 6.8.

Step 8

Finally in step 8, the totality constraints are determined for all subtypes in the usual way. This completes the subtype analysis. Figure 6.8 shows the end result with clearer communication.

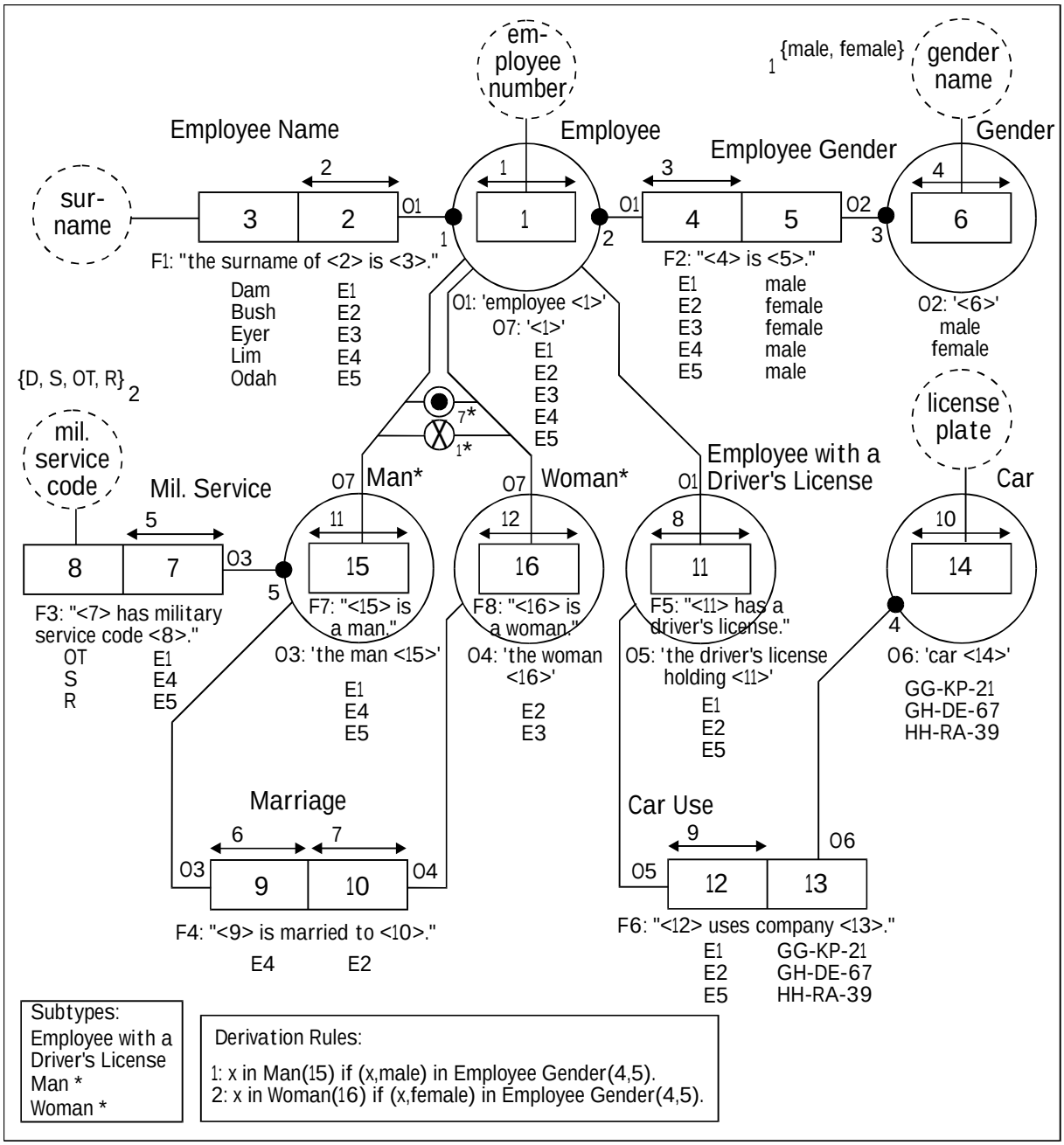


Figure 6.8: IGD with subtypes and clearer communication

6.1.3 Subtypes with More than One Supertype

In this section we will give a second example of the subtype matrix-method in the form of a small case study. No declarative subtypes appear, but there is a more complex subtype network with a subtype that has two supertypes.

The case study concerns a company that offers vehicles for rent. The starting document is shown in figure 6.9.

> The owner of a one-man vehicle rental company wants to automate his administration. Presently, he records all the information about the vehicle rental by hand on paper, but he intends to use a personal computer for this in the future. Because he is not familiar with PC's, he wants to carry out the automation in stages. The first stage just concerns data about the vehicles themselves, not yet about the contracts or the accounting.

Figure 6.9: starting document vehicle rental company

The owner and the information analyst together draw up a concrete example document, shown in figure 6.10.

VEHICLE	KIND OF VEHICLE	RENTAL PRICE	WEIGHT	KIND OF FUEL	WHEEL SIZE	CYLINDER CAPACITY
ZT-13-KD	car	115,00	---	diesel	---	---
MT-15-RT	motorcycle	75,00	---	gasoline	---	250
F1001	bicycle	15,00	25	---	28	---
MZ-47-SR	motorcycle	70,00	100	gasoline	---	500

Figure 6.10: concrete example document vehicle rental company

The analyst and the owner together carry out the verbalization, classification and qualification. The analyst draws the IGD and in dialogues with the owner he defines value constraints, uniqueness constraints and totality constraints. The resulting IGD is shown in figure 6.11. Verbalization, classification and qualification can be easily reconstructed from the IGD.

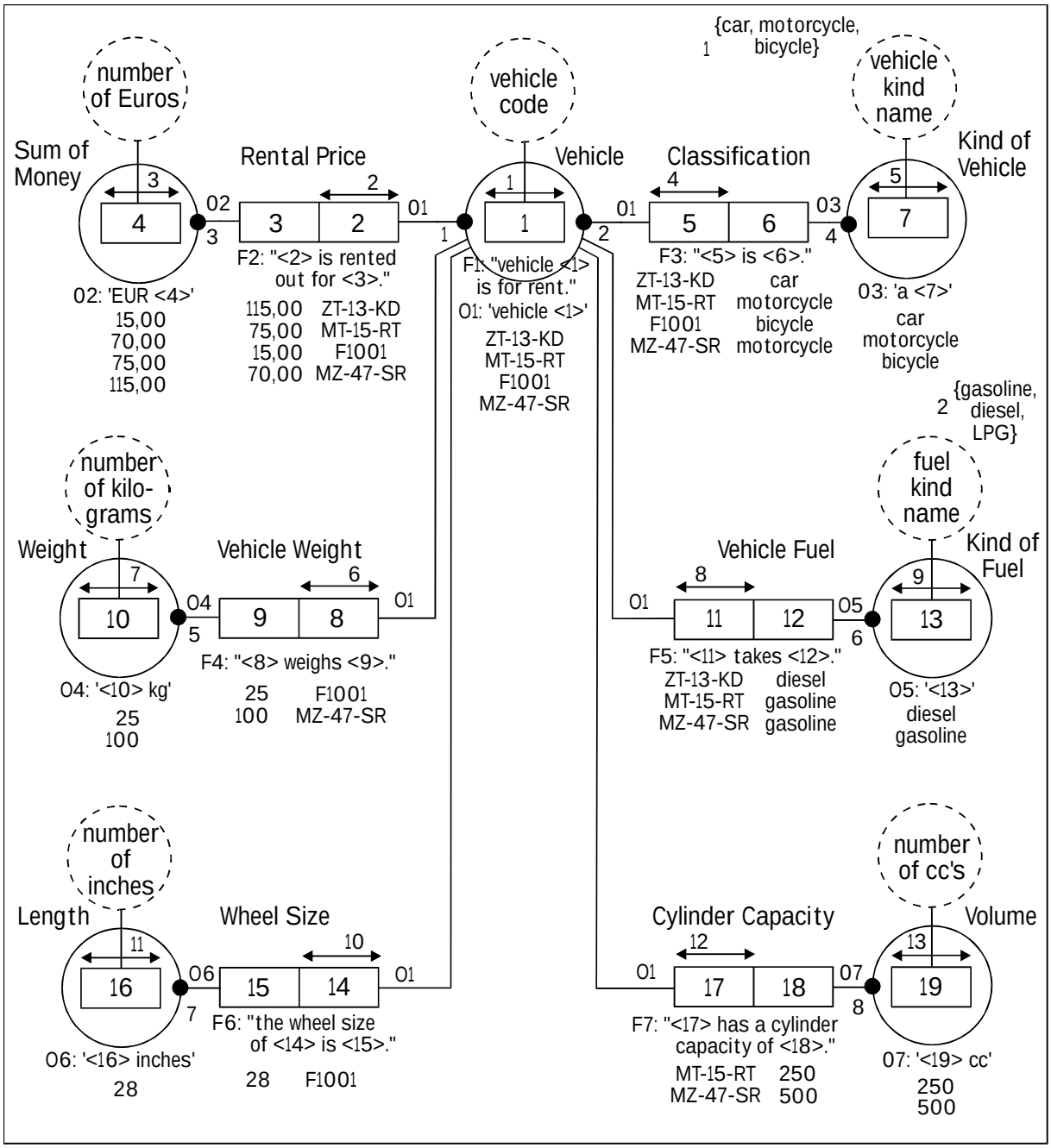


Figure 6.11: IGD vehicle rental company before subtype analysis

Only object type Vehicle is considered for subtype analysis, because it does not have a single role totality constraint on all its roles like all the others. Notice that in the verbalization there is now no indication to be found for the existence of subtypes: all fact type expressions use ‘vehicle <vehicle code>’ as object type expression. We now apply the subtype matrix method, giving each step a short explanation. The subtype matrix is depicted in figure 6.12.

Vehicle	role 2 (Rental Price)	role 5 (Classifi-cation)	role 8 (Vehicle Weight)	role 11 (Vehicle Fuel)	role 14 (Wheel Size)	role 17 (Cylinder Capacity)
ZT-13-KD	X	X	-	X	-	-
MT-15-RT	X	X	X (!)	X	-	X
F1001	X	X	X	-	X	-
MZ-47-SR	X	X	X	X	-	X
	A	A	A	B	C	E	D

Figure 6.12: subtype matrix for object type Vehicle

We give a short summary of the subtype matrix method.

Step 1 Next to the column for object type Vehicle itself there is a column for each role played by Vehicle. The names of the fact types in which the roles are found are added between brackets as a reminder. The example population in the column for Vehicle is significant.

Step 2 All crosses follow from the example population, except the one in the cell for role 8 and vehicle MT-15-RT.

Step 3 Role 8: The owner says upon being asked that he does not want to register the weight of vehicle ZT-13-KD because he only records that for bicycles and motorcycles. For MT-15-RT he would record the weight at once: the weight of this motorcycle just happens to be unknown still. That is why a cross is placed after all in the cell (emphasized with an exclamation mark).
Role 11: The owner (naturally) will not record any fuel consumption for bicycle F1001; he does do so for all motor vehicles.
Role 14: Wheel size turns out to be relevant only for bicycles.
Role 17: Cylinder capacity is relevant only for motorcycles.

Step 4 The column without dashes gets the letter A as a symbolic name. There are two different columns with three crosses, which are named B and C in arbitrary order. The column with two crosses is called D, and the last column with only one cross receives the name E. A is the highest supertype (object type Vehicle) and B, C, D and E are subtypes.

Step 5 B: The crosses in B are a subset of those in A (arrow 1).
C: The crosses in C do not all occur in B (the cross for ZT-13- KD is missing there), but they do in A (arrow 2).
D: The crosses in D also occur in C (arrow 3). The crosses in D are also a subset of those in B (arrow 4). Of course they are also a part of those in A, but including $D \rightarrow A$ should be redundant, because we already have $D \rightarrow C$ and $C \rightarrow A$ (and in addition also $D \rightarrow B$ and $B \rightarrow A$), and we will draw only the direct subset relationships. In other words: we can already go from D to A following arrows 3 and 2 (or 4 and 1) via C (or B), so that a direct arrow from D to A is superfluous.

E: The cross in E is no subset of the crosses in D, neither of those in C, but it is of those in B (arrow 5). Again, no arrow E → A is drawn because arrows 5 and 1 already exist.

Step 6 In the sketch at the bottom in figure 6.13, we see that D is a direct subtype of both B and C. D is an indirect subtype of A, as is E. B is supertype of E and D, but subtype of A. The roles played by the highest supertype and the subtypes can be read directly from the subtype matrix. In consultation with the owner meaningful names are chosen. A is object type Vehicle, of course. B concerns motorcycles and bicycles, so it receives the name Vehicle with Two Wheels. C contains only cars and motorcycles so it receives the name Motor Vehicle. D concerns only motorcycles and E only bicycles so their names are obvious.

The result of steps 5 and 6 is shown in figure 6.13.

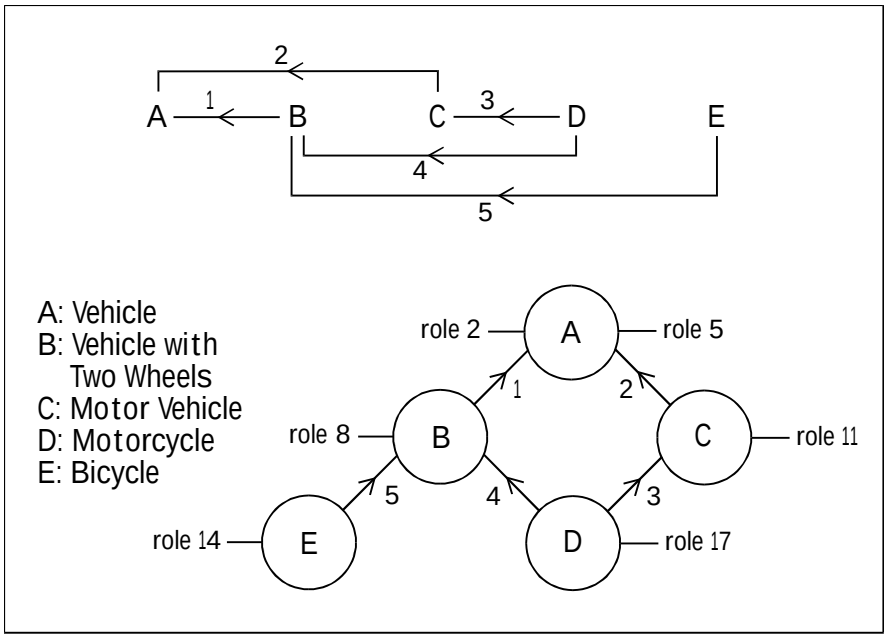


Figure 6.13: subtype network with meaningful names

Step 7 The complete IGD is shown in figure 6.14. There are no declarative subtypes, so all the subtypes are derivable and must be furnished with a derivable fact type, a derivation rule and preferably also with a fact type expression.
For subtypes that have only one subtype (Vehicle with Two Wheels, Motor Vehicle and Bicycle), this means we must make a derivable unary fact type, of which the only role is played by its direct supertype. The fact type expressions are obvious.
For subtypes with more than one direct subtype (here: Motorcycle), however, we must make a derivable fact type with as many roles as direct supertypes. Each direct supertype plays one role in it. Here, Motorcycle has two supertypes, so fact type Motorcycle gets two roles: role 23, played by Vehicle with Two Wheels, and role 24, played by Motor Vehicle.

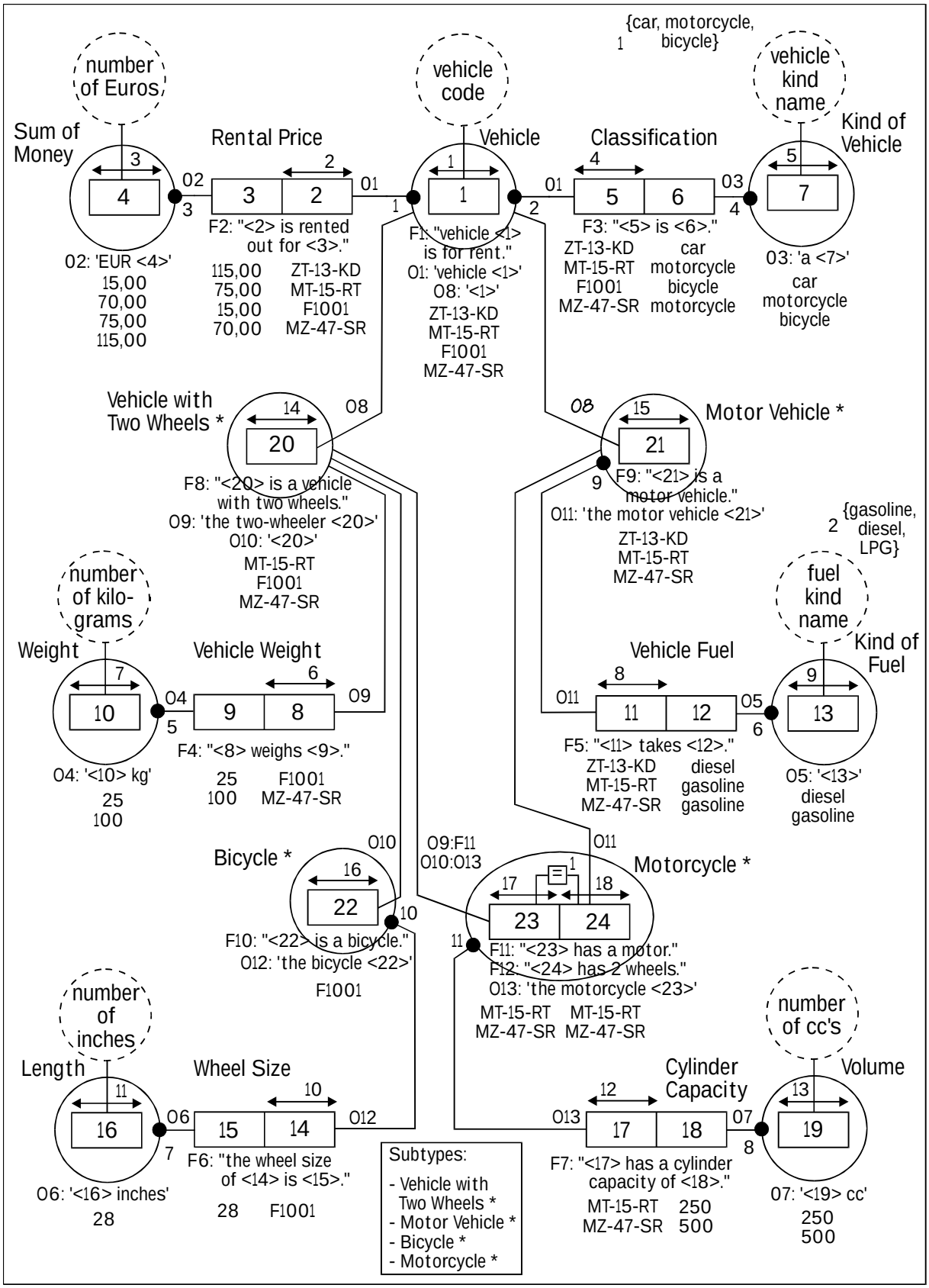


Figure 6.14: IGD vehicle rental company after subtype analysis

Fact type Motor Vehicle will now preferably be furnished with two fact type expressions: one to express that a certain vehicle with two wheels is a motorcycle (in which only role 23 occurs), and one to express that a certain motor vehicle is a motorcycle (in which only role 24 occurs). After all, the information that a certain vehicle with two wheels is a motorcycle is different from the information that a certain motor vehicle is a motorcycle. To emphasize this we have chosen clearly different fact type expressions. An example fact expression from both: "The two-wheeler MT-15-RT has a motor." (F11) and "The motor vehicle MT-15-RT has 2 wheels." (F12). For that matter, though, there would be nothing against expressions such as: "The two-wheeler MT-15-RT is a motorcycle." and "The motor vehicle MT-15-RT is a motorcycle." as long as both roles in principle get their own fact type expression.

So the procedure is the same for all subtypes: provide all subtypes with a fact type with as many roles as there are direct supertypes, and give a separate fact type expression for each role.

A consequence of this is that the populations of both roles of subtype Motorcycle are exactly the same in each tuple. This seems redundant, but actually it is not because from a purely structural point of view there are two separate subtypes: Motorcycle 1 as a subtype of Vehicle with Two Wheels, with its own derivable unary fact type with role 23 and F11, and Motorcycle 2 as a subtype of Motor Vehicle, with its own derivable unary fact type with role 24 and F12. Because Motorcycle 1 and Motorcycle 2 always have exactly the same population, and therefore represent the same object type in the UoD, it is better to unite both subtypes Motorcycle 1 and Motorcycle 2 in a single subtype Motorcycle, otherwise two identical fact types Cylinder Capacity would arise (this is an example of *abridged generalization*, see section 6.2.2). To enforce that the populations of roles 23 and 24 are the same *in each tuple*, we must add the so-called *strict equality constraint* 1, drawn as a little square with '=' in it.

Strictly speaking, subtype Motorcycle would now need two object type expressions: O13: 'the motorcycle <23>', and O14 'the motorcycle <24>', which could both be used for role 17. However, we only leave one because the other does not contribute anything new. Because of this, there is only one value per tuple in the population of role 17, namely from the population of role 23, which is the only one occurring in O13.

Finally the owner agrees to the changes in the original communication by applying clarifying object type expressions, namely O9, O11, O12, and O13. We need the other OTEs O8 and O10 then to generate good sentences, as the reader can easily verify. For role 23, subtle substitution is then required (see section 2.10).

We give the derivation rules for the derivable subtypes here separately, for lack of space in figure 6.14:

1: x in Vehicle with Two Wheels(20)	if	(x, motorcycle) in Classification (5,6)
	or if	(x, bicycle) in Classification (5,6).
2: x in Motor Vehicle(21)	if	(x, motorcycle) in Classification (5,6)
	or if	(x, car) in Classification (5,6).
3: x in Bicycle(22)	if	(x, bicycle) in Classification (5,6).
4: x in Motorcycle(23)	if	(x, motorcycle) in Classification (5,6).
5: x in Motorcycle(24)	if	(x, motorcycle) in Classification (5,6).

Please note that both roles in Motorcycle receive a derivation rule. All derivation rules satisfy the well-formedness rule that they can only concern fact types 'higher up' in the subtype network (see section 6.1.1.2).

Step 8 For the determination of totality constraints, we only need to consider roles 8, 11, 14 and 17, since nothing changes for the other roles. Role 8 does not get a TC because evidently the weight is sometimes missing for a vehicle with two wheels (if extra crosses arise in step 3 (marked by an exclamation mark in the subtype matrix), then there automatically cannot be a single role TC on the role involved). The owner wants to record the kind of fuel for all motor vehicles, the wheel size for all bicycles, and the cylinder capacity for all motorcycles, so TCs 9, 10 and 11 are added.

This completes the subtype analysis. The subtype matrix method has systematically brought to light a great number of constraints, which were still missing from figure 6.11, and which were made clear in the IGD of figure 6.14.

6.1.4 Final Remarks

1 The advice from section 6.1.1.1 saying that every declarative subtype should have a fact type expression, was not given from a structural point of view but from a methodical one. In FCO-IM, next to the structure also the method is of great importance. Of all aspects, this method was also always one of the strongest points in NIAM (see appendix B). A clear criterion is desired for the issue of which declarative subtypes are or are not to be distinguished during classification and qualification, to avoid creating unnecessary or tautological subtypes. At the moment we use a number of heuristics for this, which, however, are transferable only by illustrating the way of working in many examples. That is why we prefer to give this advice in this book, which demands explicit verbalization for all declarative subtypes and so offers a moment to decide whether this verbalization is actually desired by the user. However, anyone who is convinced he/she will not make any superfluous subtypes can safely ignore the advice. Apart from that, the rule remains valid that derivation rules for derivable subtypes can only concern fact types 'higher up' in the network.

2 The result in figure 6.8 would also be achieved with the subtype matrix method if we would not have known constraints C1, C2, C3 and C4 from figure 6.1. There is no method available to determine these constraints: they appear out of thin air in section 6.1.1. With the subtype matrix method, however, we can systematically determine subtypes, derivation rules and totality constraints that are equivalent with these constraints. In the example from section 6.1.3, such constraints were missing before we carried out the subtype analysis. Such a situation is usual in practice.

3 A difficult point is how to obtain a significant population for the subtype matrix method. Abstractly speaking, care must be taken that an example is included for each valid possibility of facts being present or absent. In the example from section 6.1.2 it is soon clear that factors such as male or female, married or unmarried, and having a driver's license or not are of importance. Eight different possibilities can be constructed from this (unmarried man without a driver's license, unmarried man with a driver's license,

and so on), for which all valid combinations of known and missing facts can be determined next. For example: take as E6 an unmarried man without a driver's license: then E6 must appear under roles 2, 4 and 7, and E6 does not appear under roles 9, 10, 11 and 12. In step 3, the same pattern of crosses would appear for E6 as for E4, because roles 9 and 11 are still relevant for E6. Continuing in this way, a larger example population would arise, of which we would be certain that it is significant. The same subtypes would indeed be derived from this. In the example from section 6.1.3, we could take a car, a motorcycle and a bicycle. For bicycles and motorcycles the weight may be missing, so we would include two bicycles and two motorcycles in the example population, one with and one without weight for both types of vehicle. Next to F1001, we would then include an F1002, which would receive an extra cross in role 8 in step 3 (role 8 is also relevant for F1002), so it would end up with the same pattern of crosses as F1001. A significant example population can also be defined as a population in which all the possible valid patterns of crosses (seen as rows) appear after step 3. In the manner sketched above, however, this is generally achieved with much larger populations. In practice, analysts usually just collect a (fairly large) number of examples with as many mutual differences as possible, which of course does not guarantee significance.

4 Roles played by the highest supertype that have a single role totality constraint of course cannot be played by a subtype after subtype analysis. They can even stay out of consideration in the subtype matrix method; the first column in the matrix will always get the letter A, so there is no danger of naming the columns wrongly.

5 Unary fact types with a non-lexical role are subtypes by definition. There is no structural characteristic, however, by which subtypes with more than one supertype can be recognized in general (subtypes are structurally nothing but normal fact types). That is why such subtypes must be explicitly marked as a subtype. This is important in the derivation of a relational schema, see section 7.1. The analysts must therefore include them in the subtype list themselves. In the FCO-IM tool, the fact type of such a subtype must be explicitly marked as a subtype, otherwise it will not appear in the list (the subtypes with a unary fact type are put there by the Tool itself).

6 Nominalized subtypes with more than one supertype do not satisfy the n-rule: there is no UC over all roles. Non-nominalized subtypes with more than two supertypes do not satisfy the n-1 rule: there are UCs on less than n-1 roles. This is because a subtype with several supertypes is actually a form of generalization (see section 6.2, and also the remark in section 6.1.3, step 7). The rules mentioned above are not valid for generalization (in the way they are formulated in section 3.3.1). We will briefly come back to this in section 6.2. Here we formulate a well-formedness rule as an exception to the n rule and to the n-1 rule, which is valid for subtypes, and call it the *subtype rule*:

> In a fact type that is a subtype, all roles have a single role uniqueness constraint. For nominalized subtypes, all object type expressions concern exactly one role.

7 If a single role totality constraint applies to a subtype role after step 8, then the well-formedness rule from section 5.2.1 comes into effect, and the subtype must be deleted; it is identical to its direct supertype and so it is superfluous.

8 Subtyping can be regarded as a semantically equivalent transformation, as is evident from figures 6.1, 6.2 and 6.5 (see also section 2.7 and section 5.1), because exactly the same fact expressions are modeled in the IGDs before and after subtyping: only the classification and qualification is different. This is true except for any final clarifying changes in the communication, which is not required.

9 Two derivable subtypes Man* and Woman* were made in figure 6.4. We could also have carried out an object type - fact type transformation on fact type Employee Gender, so that two declarative subtypes Man and Woman would replace fact type Employee Gender (see also the second to last paragraph in section 5.1.1.1). This is not always possible in general, however: in figure 6.14 for example, the subtypes cannot arise from an object type - fact type transformation of fact type Classification, and it is really necessary to use derivable subtypes.

10 Finally we give an example of derivation rules that concern fact types *in* supertypes (see the well-formedness rule in section 6.1.1.2, second hyphen).

Suppose in the example of the vehicle rental company, that the first letter from the vehicle code bears information separately: for bicycles the code begins with an F, for motorcycles with an M and for cars with any letter except F or M (the population from figure 6.14 is consistent with this). In that case, fact type Classification is dropped because it is superfluous and fact type Vehicle consists of two roles: a role 1a played by a label type ‘first letter’, and a role 1b played by a label type ‘rest of vehicle code’. The population from the old role 1 is then distributed over role 1a and role 1b. In the population of roles 2, 20, 21, 8, 11, 22, 23, 24, 14 and 17, a comma would stand between the first letter and the rest of the vehicle code.

The derivation rules for the subtypes would then become (in which the notation x(1a) means: the part of x from role 1a):

x in Vehicle with Two Wheels(20)	if	x in Vehicle and x(1a) = M
	or if	x in Vehicle and x(1a) = F.
x in Motor Vehicle(21)	if	x in Vehicle and x(1a) ≠ F.
x in Bicycle(22)	if	x in Vehicle and x(1a) = F.
x in Motorcycle(23)	if	x in Vehicle and x(1a) = M.
x in Motorcycle(24)	if	x in Vehicle and x(1a) = M.

6.2 Generalization

It often occurs that the same kind of facts is to be recorded for different object types, which are identified in different ways. For example: in a certain company people distinguish projects, identified by a project number, and assignments identified by the name of the department carrying them out together with a sequence number. Next to fact types that are only registered for projects or only for assignments, the budget must be known for both projects and assignments. Projects and assignments are both regarded as tasks subjected to budgeting.

Generalization is a way to unite two or more different object types into a new object type that contains all those object types. The new object type does not have an identification of its own but obtains this from the identifiers of the component object types. In the above example, a new object type Task is created, without an own identification, which by definition consists of all projects and assignments, which all retain their own identification. We say that object type Task is the *generalization* of object types Project and Assignment. So in a certain sense, generalization is the opposite of specialization, in which on the contrary special subsets of an object type are distinguished.

The way in which generalization is modeled in FCO-IM enables us to solve classical problems such as correctly modeling synonymy (different names for the same object) and homonymy (the same name for different objects) without using artificial tricks (such as appointing a primary identifier in synonym cases or introducing a new artificial name in homonym cases). Moreover, recursive identification structures can be modeled correctly as well.

In the following sections we will discuss all these forms of generalization using a number of typical examples (section 6.2.1: ordinary generalization; section 6.2.2: abridged generalization; section 6.2.3: generalization with synonymy; section 6.2.4: generalization with homonymy; section 6.2.5: recursive fact types). Further research on the method for generalization in FCO-IM is required (when and how precisely should we generalize?), and even structurally the possibilities have not yet been explored in every detail, or the problems all been solved. In final remark 1 in section 6.2.6, we will go further into this matter.

The forms of generalization that are illustrated in the examples are also all supported by the FCO-IM tool.

6.2.1 Ordinary Generalization

To introduce the concept of generalization, we will consider a small UoD in this section. It concerns an educational institution with students and teachers. In figure 6.15 is an IGD that models the communication about this UoD. It shows the following points:

- students are identified by a student number and teachers by a teacher code;
- only for students is recorded which education type (full-time or part-time) they enrolled in: fact type Enrollment;
- only for teachers is registered which room they occupy: fact type Office;
- for both students and teachers the surname is recorded, respectively in fact type Student Name and fact type Teacher Name;
- for both students and teachers telephone numbers are stored, respectively in fact type Student Phone Number and fact type Teacher Phone Number.

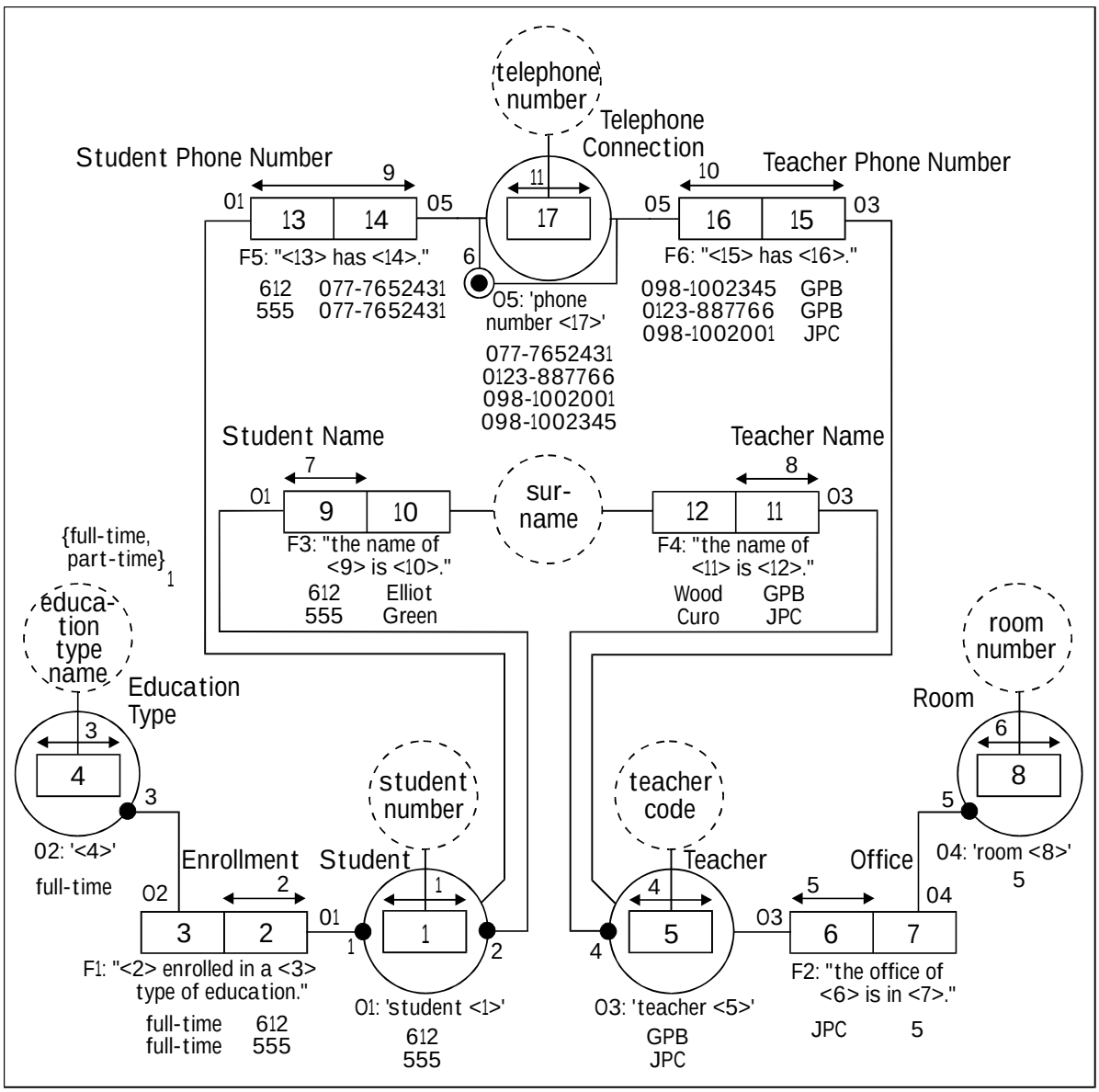


Figure 6.15: Student/Teacher IGD without generalization

The pairs of fact types Student Name / Teacher Name and Student Phone Number / Teacher Phone Number make a strong impression of being duplicates. If all sorts of other personal information would also have to be recorded for both object types, then we would end up with even more 'double' fact types. We would like to replace each pair of fact types with one fact type, which applies to both object types Student and Teacher. In FCO-IM this can be done by generalizing both separate object types to a new object type Student/Teacher that unites (the population of) both separate object types: see figure 6.16.

The new object type Student/Teacher gets a fact type with two roles, one for each object type of which it is the generalization. By definition, role 18 is populated with all the students and role 19 with all the teachers (the union of two sets contains all the elements from both separate sets). Therefore, roles 18 and 19 both have a single role totality constraint by definition.

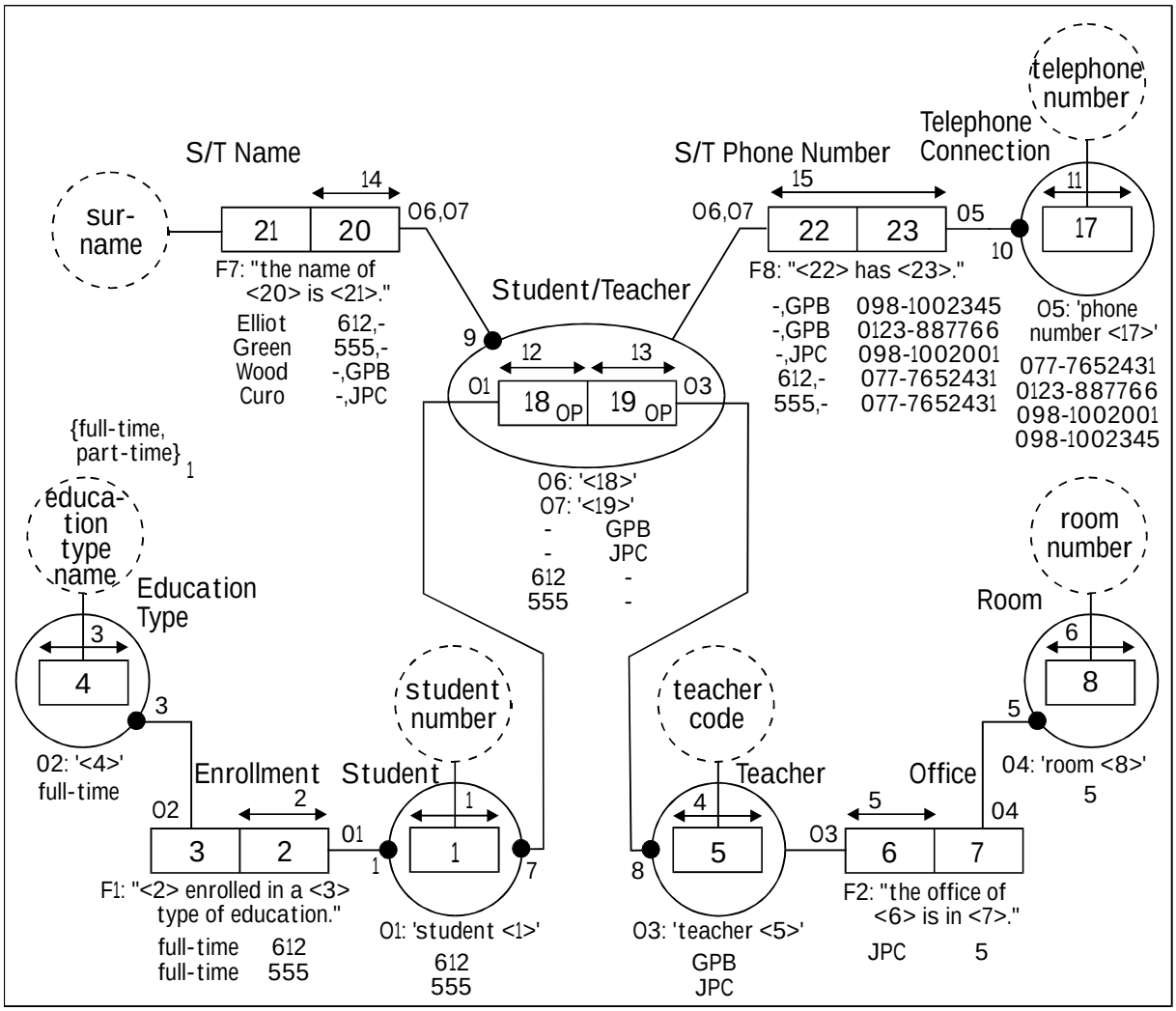


Figure 6.16: Student/Teacher IGD with generalization

Both roles also have a single role uniqueness constraint: every teacher or student can occur there only once. Both roles are optional: a tuple is either populated with a student under role 18 and a null value under role 19, or with a teacher under role 19 and a null value under role 18 (only in cases with synonymy can more than one role be populated per tuple, see section

6.2.2). There are two object type expressions for object type Student/Teacher: one that only refers to role 18 (for a student) and one that only refers to role 19 (for a teacher).

Fact types Student Name and Teacher Name have now been replaced by one new fact type S/T Name, of which role 20 is played by object type Student/Teacher. Role 20 is populated with tuples from object type Student/Teacher, as is always the case in an elementary IGD (for the same IGD with tuple numbers and tuple pointers see figure 6.18). How can we tell now whether a tuple from fact type S/T Name concerns a student or a teacher? Let us illustrate this by regenerating the fact expression from the top tuple of S/T Name in figure 6.16: 'Elliot 612,-'. We begin with fact type expression F7: "the name of <20> is <21>." We find a reference to role 20 in it. It is indicated next to role 20 that object type expressions O6 and O7 can be used. Which one should we take? The value under role 20 that we must substitute eventually is: '612,-'. That is the third tuple from Student/Teacher. This tuple has a value under role 18 but not under role 19. So we cannot take O7 but have to substitute O6. We now have: "the name of <18> is <21>." For role 18 we must next substitute O1 and so we get: "the name of student <1> is <21>.". Now there are only references to lexical roles left, so the final fact expression becomes: "the name of student 612 is Elliot." In the same way, we obtain the fact expression "the name of teacher JPC is Curo." from the bottom tuple of fact type S/T Name, because this time the null value occurs under role 18, so we cannot use O6, but must take O7. In short: the position of the null values dictates which OTE is eligible for substitution into roles played by a generalization.

As was the case for specialization, it is always possible to create an IGD with or without generalization by just using another way of classification and qualification: the IGDs in figures 6.15 and 6.16 model exactly the same communication.

We illustrate the differences in classification and qualification for fact type S/T Name in figure 6.17. The top half of the figure shows the classification and qualification of fact expressions that leads to the two fact types Student Name and Teacher Name from figure 6.15, using two example fact expressions. The bottom half of figure 6.17 shows the classification and qualification of the same two example sentences that leads to the one fact type S/T Name and the generalization Student/Teacher from figure 6.16. The arrows in the middle of figure 6.17 indicate that the two different sentences (and the different analyses of them) are now attributed to the same fact type. Both on the left and right sides we now recognize the same new generalized object type Student/Teacher, with O6 on the left and O7 on the right. In the further analysis of O6 and O7 we next indicate whether a student (left) or a teacher (right) is concerned. The rest of the analysis is the same as before. F7 now replaces F3 and F4, in which we integrate the left and right analysis by writing both O6 and O7 behind Student/Teacher.

We could add existence postulating fact type expressions to fact type Student/Teacher in figure 6.16, for example F9: "<18> is a student." and F10: "<19> is a teacher.", with appropriate adaptation of the OTEs, but we skip that here because object type Student/Teacher does not have to have an existence postulator due to totality constraint 9 (see the well-formedness rule from section 3.4).

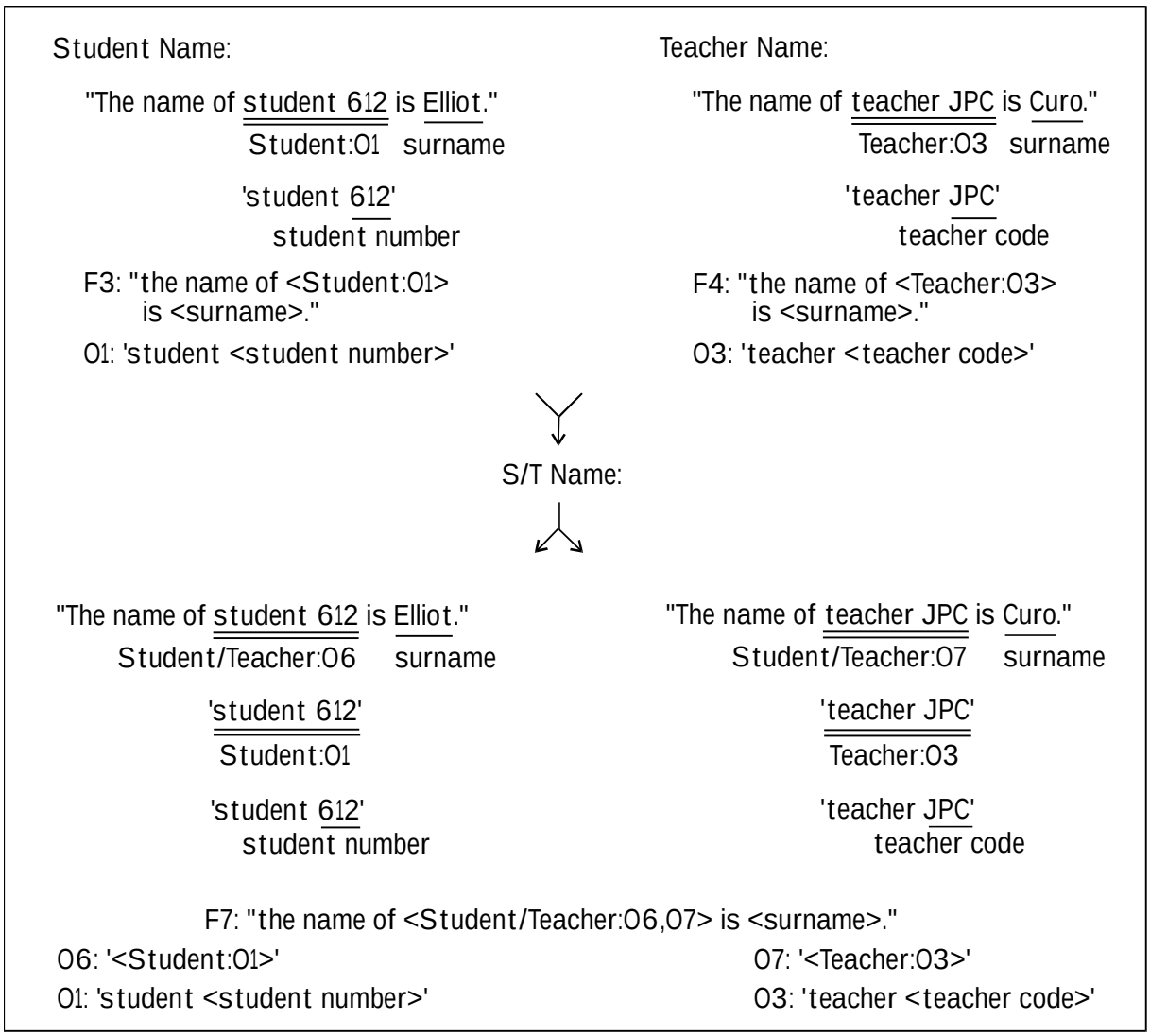


Figure 6.17: classification and qualification with and without generalization

For the sake of completeness, we show the IGD from figure 6.16 once more in figure 6.18, but this time with tuple numbers and tuple pointers (see section 2.11). This makes the way O6 and O7 work in regenerating fact expression even more clear: tuple 1 from fact type S/T Name refers under role 20 to tuple 3 from Student/Teacher, which only has a value under role 18, so that we must choose O6.

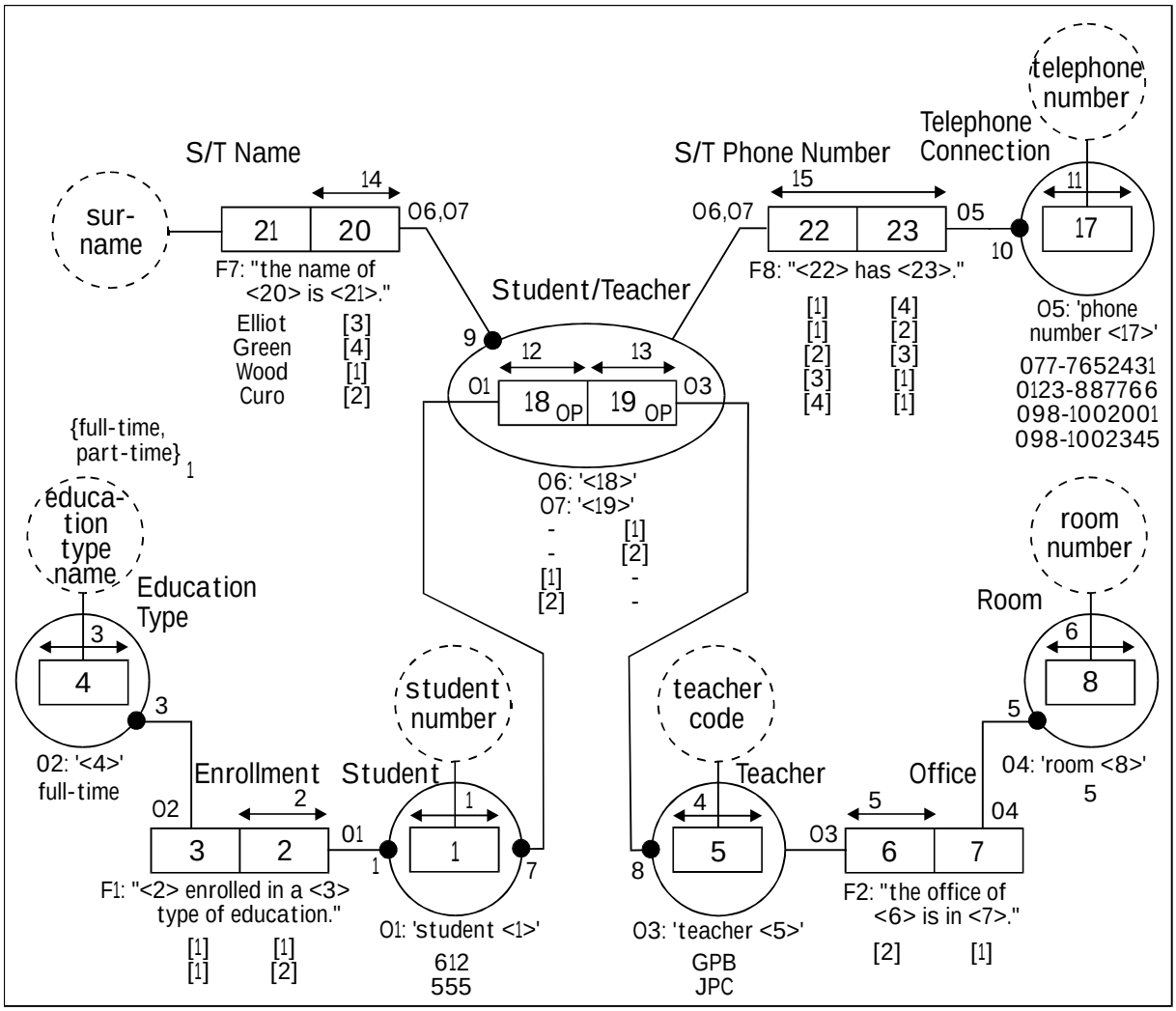


Figure 6.18: generalization with tuple numbers and tuple pointers

6.2.2 Abridged Generalization

We will now briefly discuss a second example of generalization, in which a shorter form of it appears. It concerns the identification of Dutch cabinets (i.e. all the ministers of the government of The Netherlands). A cabinet is named after the prime minister, such as the cabinet Den Uyl. Only in the case that different cabinets have the same person as prime minister, a sequence number is used to distinguish them, such as the cabinets Lubbers I, Lubbers II and Lubbers III. A similar way of identification also often occurs in other contexts. We do not want to model the way of identifying cabinets by adding sequence number I anyway even for cabinets that do not have a sequence number (which is often done as a 'solution'), because it just does not happen in the communication. We use generalization to model cabinets with and without a sequence number, as is shown in figure 6.19.

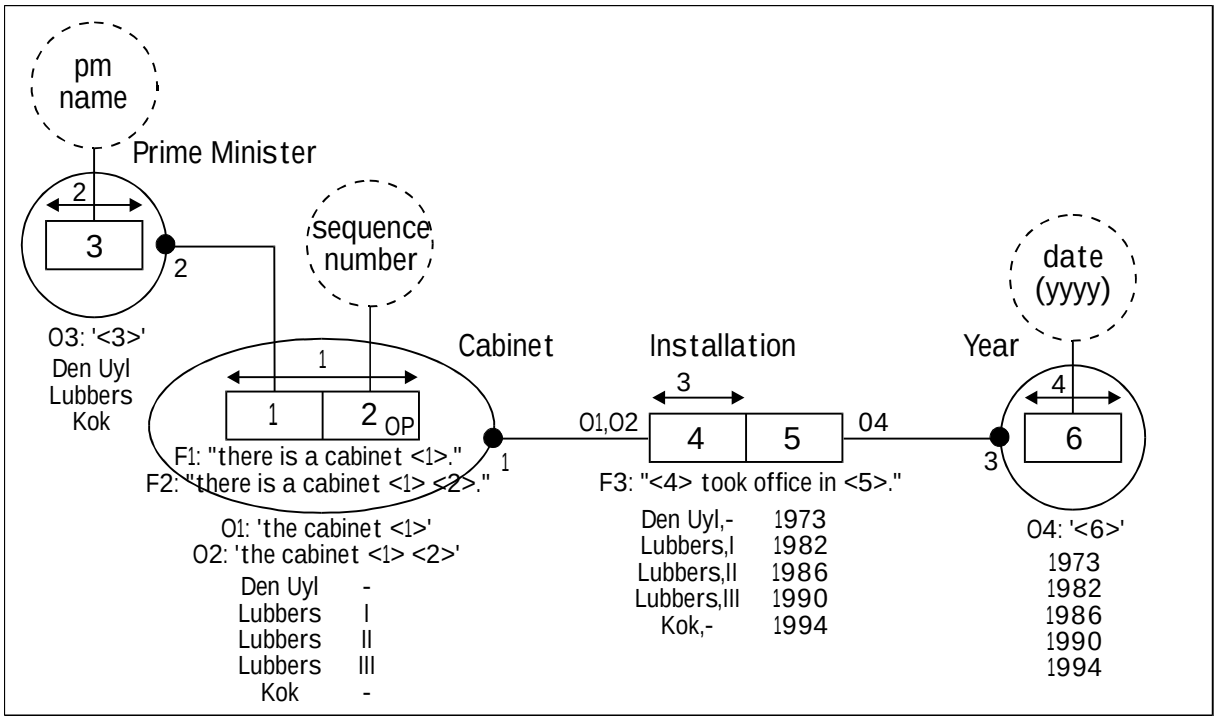


Figure 6.19: IGD cabinets with and without sequence number

The population of fact type Cabinet shows null-values under role 2 for cabinets without a sequence number. During the regeneration of fact expressions from fact type Installation, the choice between O1 and O2 for role 4 is again dictated by the presence of null values: in the top tuple of Installation the value ‘Den Uyl,-’, is written under role 4, with a null value in the position of role 2, so we must choose O1 here, whereas in the second tuple the value ‘Lubbers,I’ appears with non-null values in the position of both roles 1 and 2, so there we must choose O2.

So why is this a shorter form of generalization? We can formally regard object type Cabinet as a generalization of an object type Cabinet1, which is identified by a prime minister only, and an object type Cabinet2, which is identified by the prime minister together with a sequence number. If we model this explicitly in an IGD, then the IGD that is shown in figure 6.20 arises, which is completely analogous to the example in section 6.2.1.

The unabridged form has a number of drawbacks. Firstly, the distinction between object types Cabinet1, Cabinet2, and Cabinet makes a very artificial impression: users do not think of cabinets with and without reference numbers as different sorts of objects. Secondly, there are no fact types that are played only by Cabinet1 or by Cabinet2: the only roles played by Cabinet1 and Cabinet2 are the roles that are needed to create the generalization. Therefore, we strongly prefer the abridged form. Anyway, both forms can be made in the FCO-IM tool by using different ways of classifying and qualifying (we do not show that here separately).

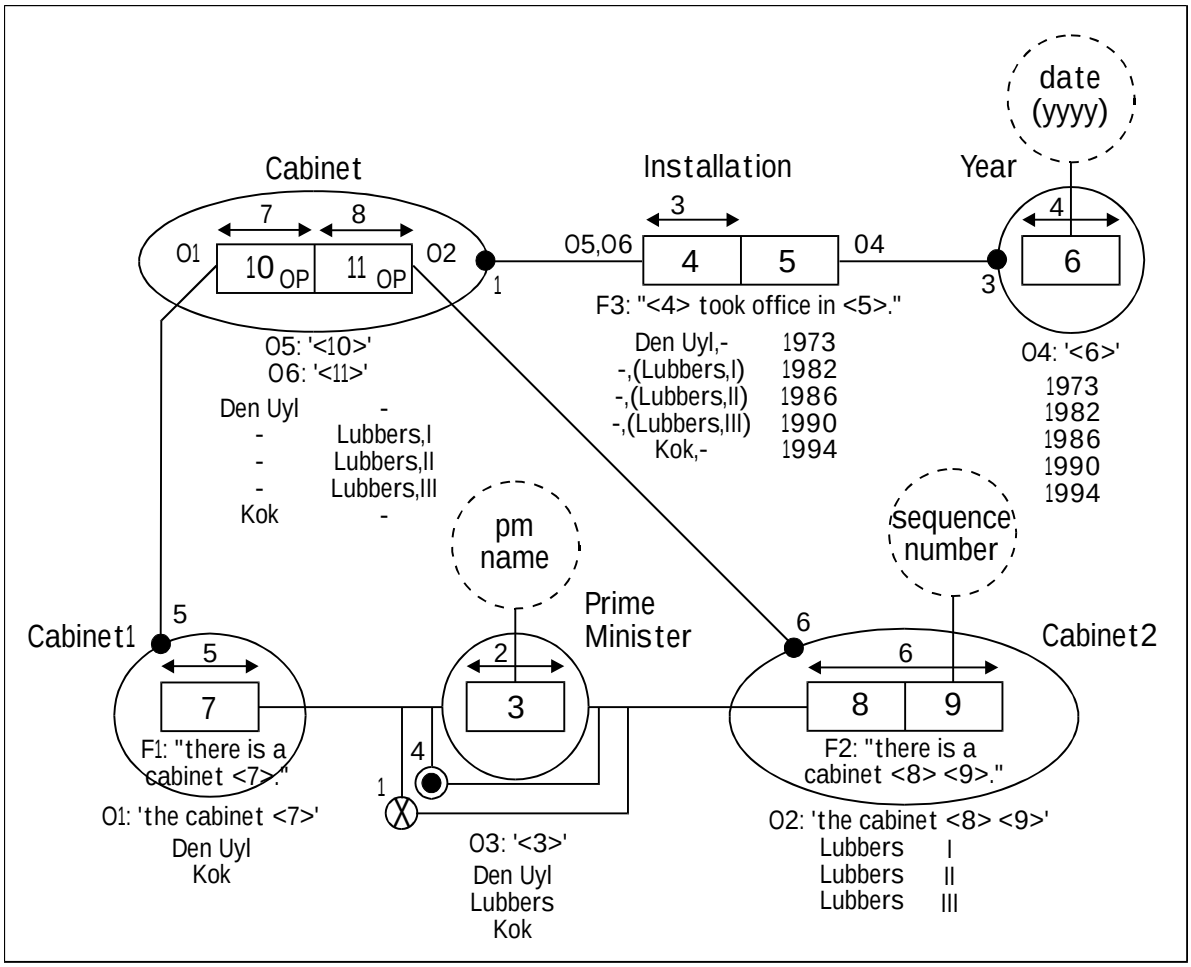


Figure 6.20: IGD cabinets with generalization in unabridged form

6.2.2.1 Uniqueness Constraints on Optional Roles

In ordinary generalization we encounter single role uniqueness constraints on optional roles, and in abridged generalization we even have multiple role uniqueness constraints on optional roles. What do such UCs actually mean? Since null values formally only indicate the absence of a value (so if a null value occurs then there is absolutely nothing there) we give the following definition (which is illustrated in figure 6.21):

For a certain UC x on a (combination of) optional role(s) the following holds:
1 UC x only applies to tuples with a value under all the roles from UC x.
2 For tuples with one or more null-values under the roles from UC x, the imaginary UC applies that would arise by making UC x narrower until it covers just all the roles with a value.

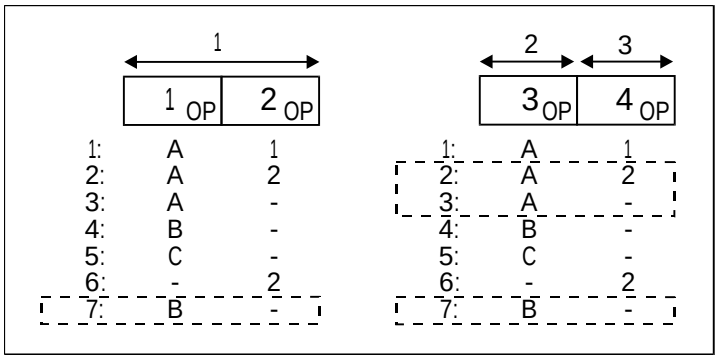


Figure 6.21: illustration of UCs on optional roles

Under roles 1 and 2 in figure 6.21, tuples 1 up to and including 6 can all appear together at the same time. Tuples 1 and 2 fall under part 1 of the definition. Tuples 3 up to and including 7 fall under part 2: for tuples 3, 4, 5 and 7 the imaginary UC on only role 1 applies, and for tuple 6 the one on only role 2.

Tuples 4 and 7 violate the imaginary UC on only role 1; that is why tuple 7 is highlighted in the figure (if tuple 4 is already present, then tuple 7 cannot be added). Note that tuples 1, 2 and 3 (values 'A,1' and 'A,2' and 'A,-') can all occur together at the same time.

On the right hand side in figure 6.21, UC 2 applies to tuples 1, 2, 3, 4, 5 and 7, and UC 3 to tuples 1, 2 and 6. Tuples 1,2 and 3 violate UC 2, the reason why tuples 2 and 3 are highlighted (if tuple 1 is already present, then tuples 2 and 3 cannot be added). Tuples 4 and 7 also violate UC 2, which is why tuple 7 is highlighted as well. Note that UC 3 does not forbid the occurrence of more than one null value under role 4: null values do not behave as 'real' values. That is why the definition above is formulated only in terms of positions without null values.

6.2.2.2 Regenerating Fact Expressions with Generalization

For abridged generalization, the question now also arises which fact expressions we can regenerate from an IGD. Here we give the precise rule for regenerating fact expressions from IGDs with generalization:

During the regeneration of fact expressions from an IGD, any FTE or OTE can be used, in which all role references:

1 either concern exactly all roles without null values in the tuple

2 or concern a part of all roles without null values in the tuple,
but then there must also be a uniqueness constraint
on just this same part of all roles.

So, in figure 6.19 we cannot regenerate the fact expression “there is a cabinet Lubbers.”, because F1 does not satisfy condition 2: there is no UC on only role 1. For the same reason, O1 cannot be used for tuple ‘Lubbers I’. In figure 6.23 however, we can regenerate the fact expression “there is a room number 3.”, because F1 satisfies both conditions (here there is indeed a UC on only role 1). The same applies for O1.

6.2.3 Generalization with Synonymy

Synonymy is the phenomenon that an object (type) can be identified in more than one way. For example, in a certain company people might be uniquely designated by their social security number as well as by an employee number. The usual practice in information modeling in such cases is to appoint one identifier as the primary one, and to use only this primary identifier in the communication. In FCO-IM however, we can avoid such an artificial constraint using generalization.

As the first example, we will consider the educational institution from section 6.2.2. Let us suppose that the same person can be a teacher as well as a student: see figure 6.22.

The information that a student is the same person as a teacher must of course be modeled explicitly. So we verbalize: “Student 149 is the same person as teacher HVL.”. The analysis of this fact expression leads to fact type expression F9 for fact type Student/Teacher. There now is a tuple in the population of fact type Student/Teacher with values under both roles 18 and 19.

The tuple ‘Adam 149,HVL’ in the population of fact type S/T Name now generates two fact expressions: “the name of student 149 is Adam.” and “the name of teacher HVL is Adam.”, depending on the choice for O6 or O7. Both OTEs can be used. This is how the different ways to designate the person 149/HVL can both be used in FCO-IM, whereas only one tuple needs to be taken up in fact type S/T Name for the surname of this person 149/HVL (if 149 under role 18 were in a different tuple from HVL under role 19, then we would need two tuples in the population of S/T Name: a form of redundancy).

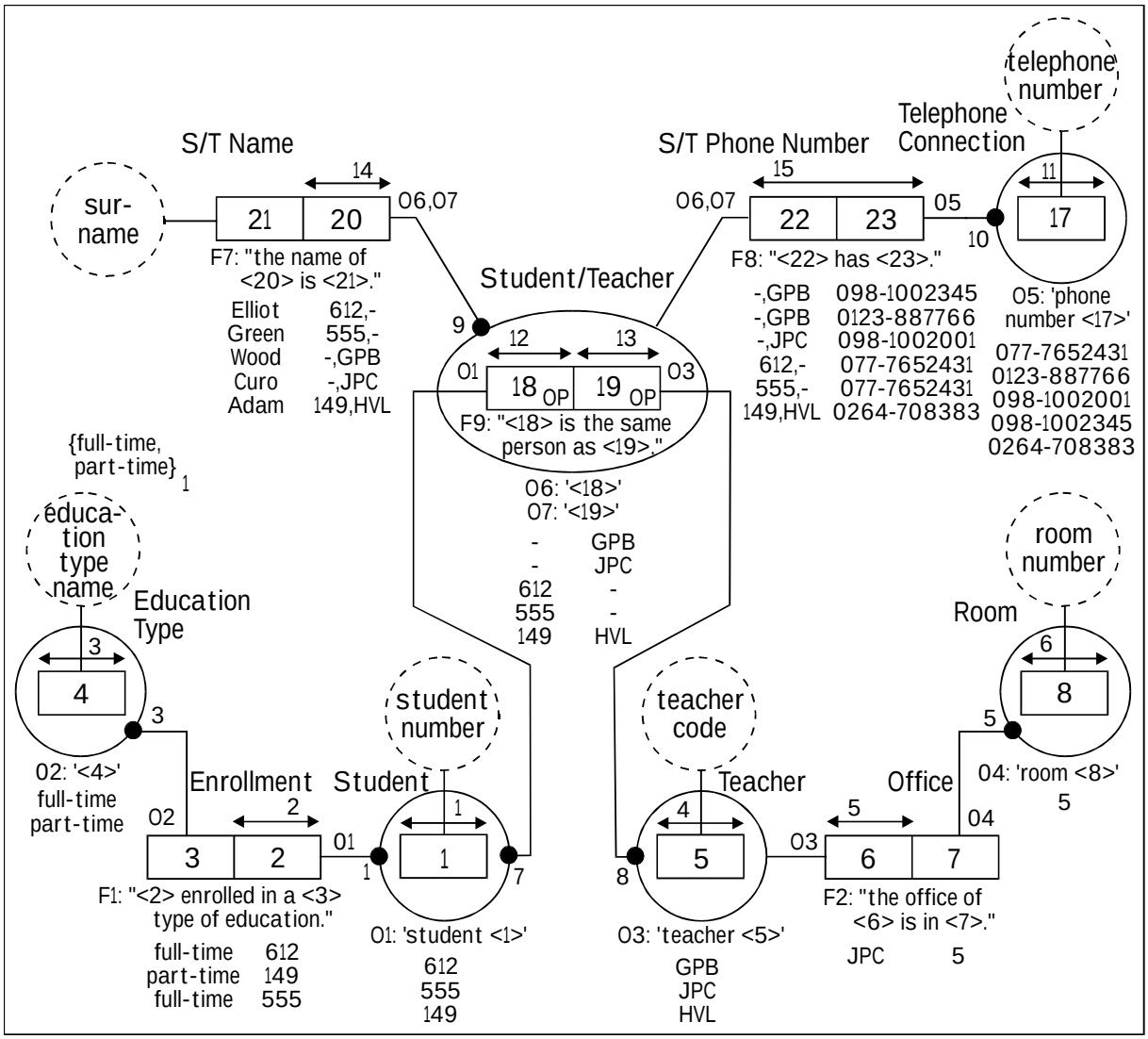


Figure 6.22: Student/Teacher IGD with synonymy

We give yet a second example of generalization with synonymy, this time with abridged generalization (see section 6.2.2). The IGD in figure 6.23 concerns the daily occupation overview of a conference center that rents rooms to organizations (only the occupation for today is shown). Some rooms have a unique room number, others a unique room name, and some have both a name and a number. The generalization is abridged because ‘number rooms’ and ‘named rooms’ are not regarded as separate objects, and there are no fact types for ‘number rooms’ or ‘named rooms’ separately (both reasons usually go together).

By request of the domain experts for the sake of convenience, a (redundant) object type expression O3 is added to object type Room to have an extra verbalization possibility, which leads to fact expressions such as “Elma Inc. occupies room 3, also known as the Bach room.”, in which the synonymy is explicitly taken up. There now are three OTEs for role 3.

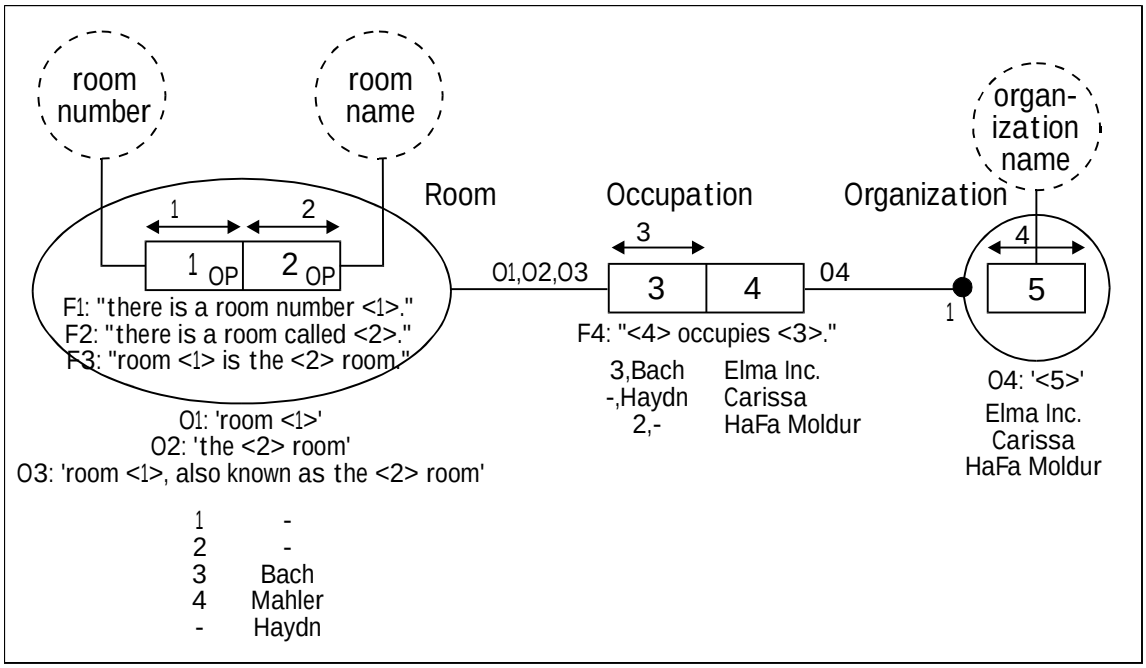


Figure 6.23: IGD with abridged generalization with synonymy

If all the rooms would have a name as well as a number in figure 6.23, then roles 1 and 2 would both be non-optional. We would then still be able to model the synonymy correctly in FCO-IM without having to appoint a primary identifier.

6.2.4 Generalization with Homonymy

Homonymy is the phenomenon that one name indicates two or more different objects. If the objects belong to different object types (otherwise there would be a real identification problem), then FCO-IM can also model this phenomenon correctly using generalization.

We give an example from a UoD with bus routes and flights. The IGD is in figure 6.24. Some fact types are only relevant for flights and others only for bus routes. There are also fact types that apply for bus routes as well as for flights, so that an object type Itinerary arises: a real generalization of Bus Route and Flight.

As is clear from the population in figure 6.24, the code AX11 can refer to a flight as well as to a bus route. No misunderstanding is possible during the regeneration of fact expressions from the IGD, however, because of the position of the null values: the top tuple from fact type Departure Day generates the fact expression: “the flight AX11 departs on mo.” whereas the bottom tuple from Departure Day yields the fact expression “the bus route AX11 departs on we.”. The only demand on the verbalization (and so on the domain expert) is that in the case of

homonymy only OTEs will be used that explicitly mention the object type an object belongs to. So here: ‘the bus route <bus route code>’ and ‘the flight <flight number>’.

Aside: an alternative modeling would be: all itineraries are identified by the combination of a number or code together with a type indicator (bus route or flight), and object types Flight and Bus Route are modeled as subtypes of Itinerary. Except for existence postulators, this would be a semantically equivalent modeling underlining that specialization and generalization are two sides of the same coin.

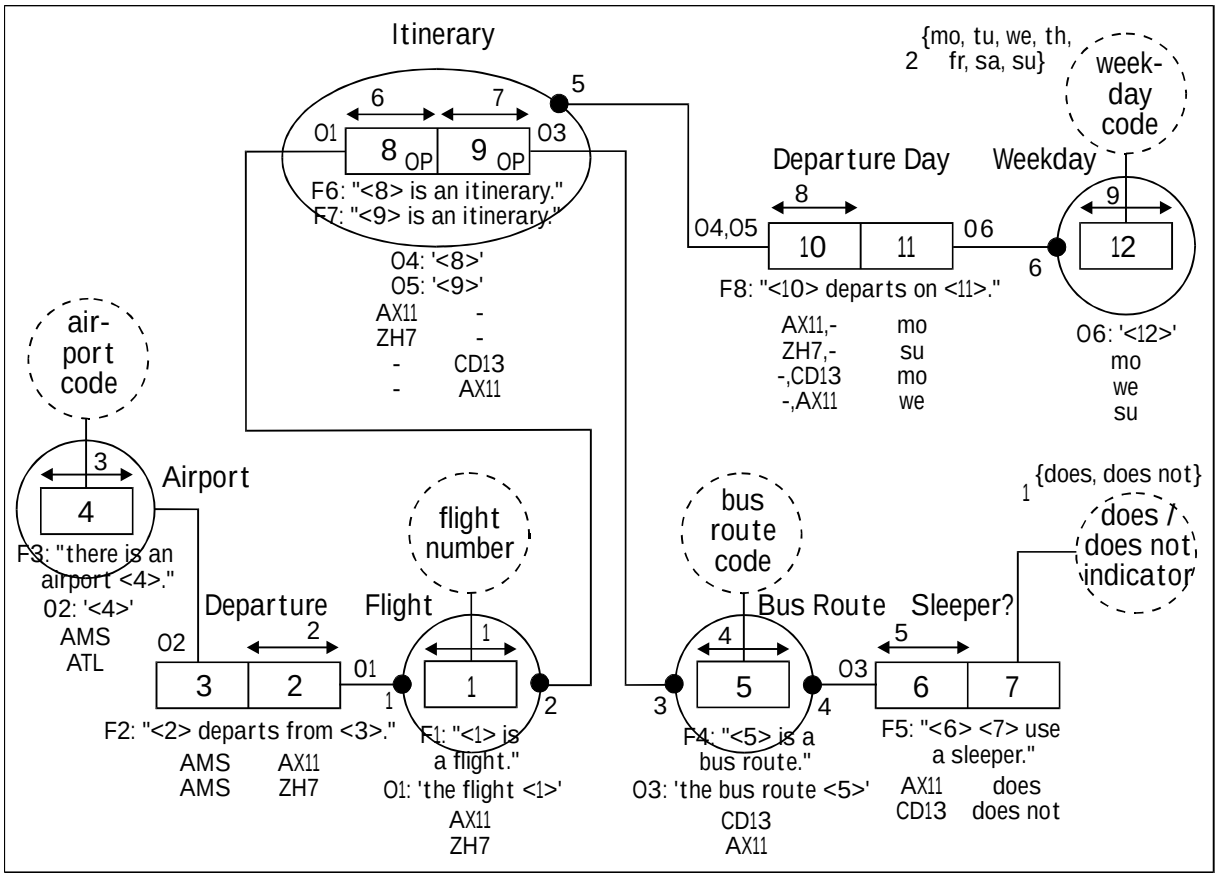


Figure 6.24: IGD with homonymy

6.2.5 Recursive Fact Types

A recursive fact type is a fact type with a role that is played by the nominalization of this same fact type (see the IGD in figure 6.26). In this section we will give a small example of the application of a recursive fact type in the identification of objects that have an identifier of variable length. For another more extensive example of a recursive fact type from practice, we refer to our paper ‘Fully Communication Oriented NIAM’ [literature list 11], which is available from the authors on demand.

This book consists of chapters that are themselves divided into sections that are often again split into (sub)sections that sometimes again consist out of (subsub)sections that in principle could be refined into still further parts that could yet again….The identification of chapters and sections reflects this limitless hierarchical structure: this book contains a chapter 6 that contains among others a section 6.1 that contains among others a section 6.1.1 that contains among others a section 6.1.1.2. In principle this row of numbers that identifies a section can become as long as we like, and the length varies from section to section: the identifier of a (chapter or) section has a variable length. How can we model this in FCO-IM? We will illustrate this below by analyzing facts about the title of a (chapter or) section. We use the two following example sentences in the analysis (see figure 6.25): "The title of chapter 6 is: 'Specialization and Generalization'." and "The title of section 6.1.1.2 is: 'Derivable Subtypes'."

An observation beforehand: we can see no essential difference between chapters and sections (and subsections and subsubsections and…), but we experience them as objects belonging to a single object type. We could name this object type 'Book Part', and also use the words 'book part' instead of 'chapter' or 'section' (or 'subsection' or…). We will stick to the usual phrases however, and talk about chapters and sections (but not about subsections and so forth). We will therefore call the object type 'Chapter/Section'. Aside: this object type Chapter/Section can be seen formally as an abridged generalization of an infinitely long series of object types Chapter, Section, Subsection, Subsubsection,…., which is why we discuss recursive fact types as a (special) form of generalization.

The classification and qualification of the two example fact expressions is shown in figure 6.25. The two different fact expressions are assigned to the same fact type Naming. The analysis on the left hand side speaks for itself (we abbreviate the long name 'Chapter/Section' in this figure under the underlining to 'C/S').

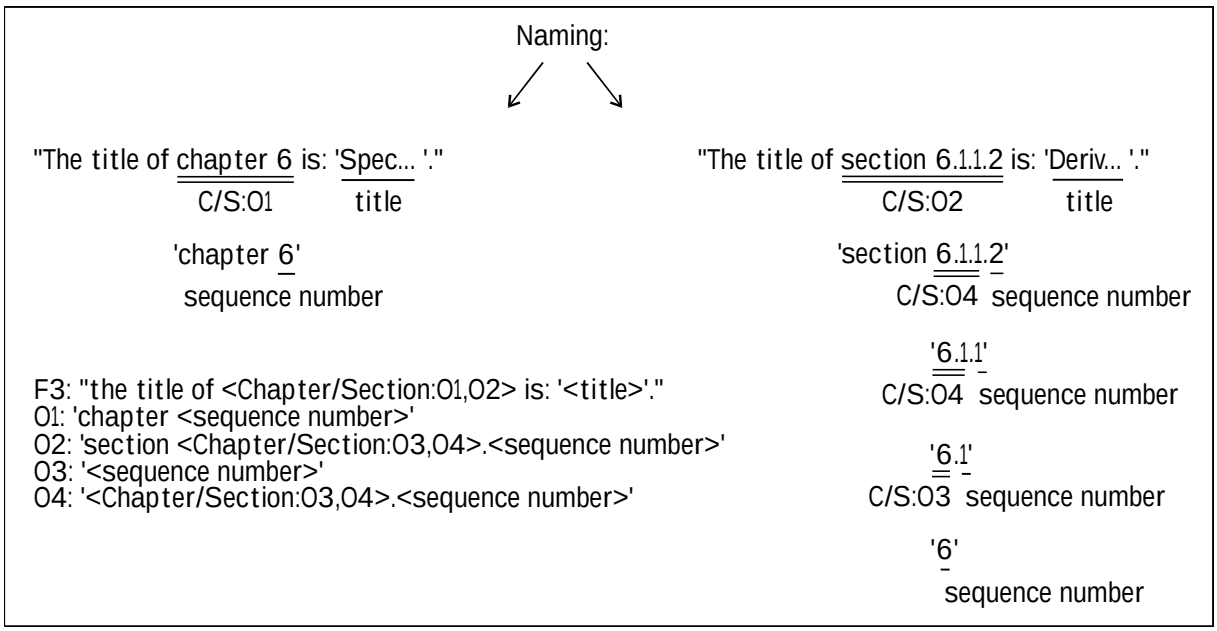


Figure 6.25: classification and qualification of a recursive identification structure

In the example sentence on the right hand side of figure 6.25, we find a second object type expression O2 for object type Chapter/Section. During the further analysis of ‘section 6.1.1.1’, we bear in mind that section 6.1.1.2 is a part of section 6.1.1. So ‘6.1.1’ again identifies a section. This yields a new OTE: O4 (which is the same as O2 without the fixed text ‘section’). In the further analysis of ‘6.1.1’, ‘6.1’ yet again identifies a section. In the further analysis of ‘6.1’, ‘6’ identifies a chapter. That also yields a new OTE O3, because there are no period and sequence number here anymore. The IGD that follows from this analysis is shown in figure 6.26.

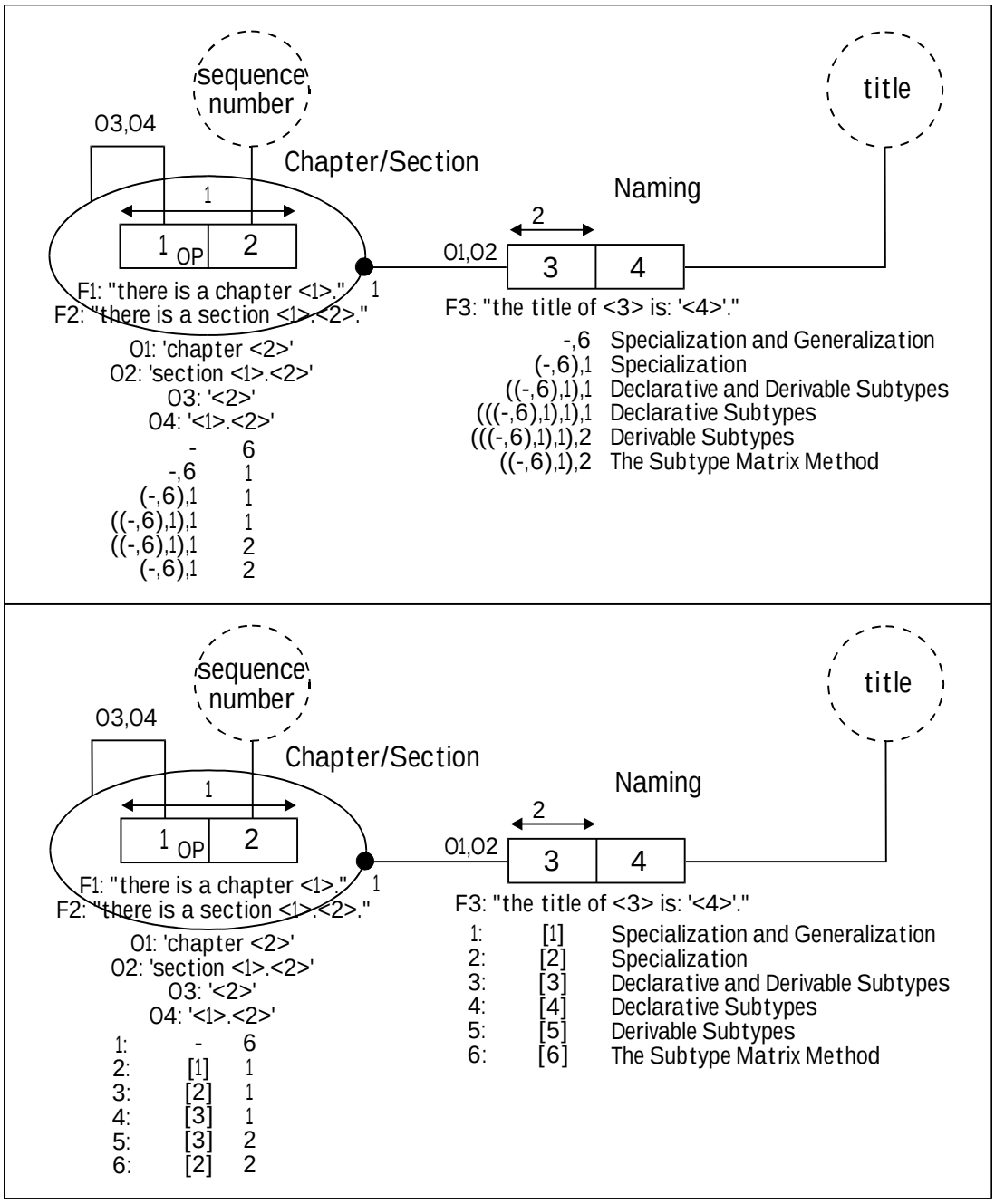


Figure 6.26: IGD with recursive fact type

In the top half of figure 6.26, we have given the population of the non-lexical roles as labels; in the bottom half there are tuple pointers. Here again the position of the null values dictates which OTE can be chosen. For clarity, two extra existence postulating FTEs F1 and F2 have been added, but these can also be left out because of totality constraint 1.

Finally, in figure 6.27 we show the regeneration of fact expressions for recursive fact types. In the top half, the population of the non-lexical roles is given as labels again, and in the bottom half as tuple pointers. Next to the vertical arrows, which represent substitution, we show which tuple or tuple part is to be filled in, and which OTE is appropriate because of that.

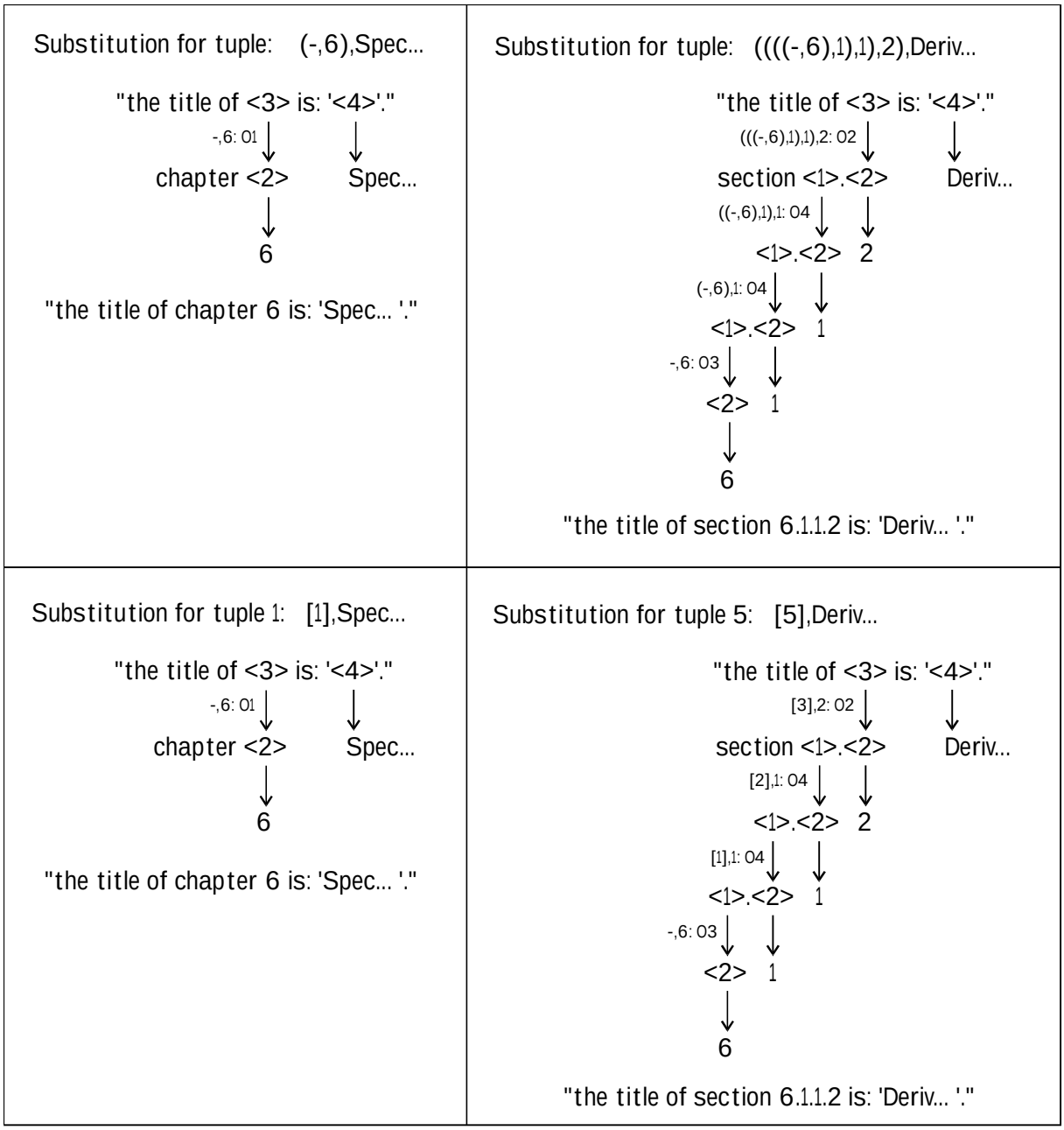


Figure 6.27: regeneration of fact expressions for a recursive fact type

6.2.6 Final Remarks

1 We have mainly described the structural side of generalization. The methodical side (when and how exactly should we generalize?) needs to be further explored. The most important heuristic for ordinary generalization (see section 6.2.1) that we use is: consider to apply generalization if the same type of facts are to be recorded for different object types, but only do so if the resulting generalized object type fits in with the view of the domain experts on their UoD. A generalized object type Person/Car, for example, is not a natural one if the weight is to be recorded for them both.

2 Subtypes with more than one supertype, described in section 6.1.3, can be regarded as an abridged generalization, see the remark in step 7 from section 6.1.3. This illustrates that specialization and generalization are two sides of the same coin. Structurally the only difference is that subtype roles can never have a TC, and generalization roles must always have a TC. Conceptually the most important difference is that in specialization the subtypes inherit their identification from the highest supertype, whereas in generalization the 'lower' object types each have their own identifiers, and it is the 'higher' object types that inherit all these identifiers.

3 In generalization, the n rule and the n-1 rule do not apply anymore. Nominalized generalized object types do not satisfy the conditions of the n rule: there is not just one multiple role UC on all the roles. Non-nominalized fact types that model a generalization do not satisfy the n-1 rule: there are UCs on less than n-1 roles (see also final remark 6 from section 6.1.4). So the n rule and the n-1 rule must be refined to include specialization and generalization as well. Presently, we can formulate well-formedness rules that is necessary, but not sufficient to describe all cases correctly, such as:

> In a nominalized fact type, all roles must fall under at least one UC,
> and each corresponding OTE can refer only to roles
> that fall under (a part of) exactly one UC.

In the future we hope to develop the theory further on this point.

4 The form of generalization discussed in section 6.2 is also supported by the FCO-IM tool. The Tool, however, gives a warning as soon as users enter generalizations, because complete well-formedness rules are not yet available (see remark 3 above), and therefore we cannot guarantee that the resulting IGDs will always be well-formed. The readers do not have to be held back from using generalization because of that, however.

7

Derivation of a Relational Schema with Specialization and Generalization

In this chapter we will discuss the particular details in the derivation of a logical relational schema from IGDs in which specialization or generalization occurs. We will first consider specialization in section 7.1, illustrating different possibilities with an example. In section 7.2 we will briefly discuss the complications that arise in cases with generalization.

7.1 Derivation of a Relational Schema with Specialization

During the derivation of a relational schema from an IGD with specialization, lexicalizing (see section 4.2), reducing (section 4.3) and conversion to a relational schema (section 4.4) go as usual without further details. For grouping (section 4.1) however, we have to refine the conditions a little, and add an intermediate grouping and reducing step between 'ordinary' grouping and lexicalizing.

7.1.1 Refinement of the Grouping Conditions

The first step in the grouping process is to mark all roles that can be deleted during grouping. Here we repeat conditions 1, 2 and 3 from section 4.1.2, step 1:

1 The role is part of an n-ary fact type, with n greater than 1.
2 The role is played by a non-lexical object type.
3 The role falls under a single role uniqueness constraint.

Condition 1 is meant to exclude roles in unary fact types from being deleted. But unary fact types with a non-lexical role are subtypes by definition. There would indeed be a serious loss of information if the role of a *declarative* subtype would be deleted (see also section 5.2): the population of the subtype itself as well as its verbalization (fact type expressions) would disappear, and so we would lose all the facts in the subtype. Therefore, roles from declarative

subtypes (also from subtypes with more that one supertype) can never be deleted.

The situation is less serious for a *derivable* subtype. If the role of a derivable unary fact type would be deleted, then we would lose the population as well as the FTE from the subtype, but not the derivation rule. We could therefore reconstruct the population of the subtype with the help of the derivation rule at any time, because we would still know which objects from the supertype belong to the subtype.

Let us look at what happens in figure 6.5 if we mark role 16 (subtype Woman) for grouping. We delete role 16 together with its population, but object type Employee absorbs no roles because now there are no roles left in fact type Woman. Fact type expression F8 cannot go to object type Employee (which tuples from Employee would we then be allowed to fill in?), and so it is deleted as well. Role 10 is now played by object type Employee. How can we now ensure that only female employees will be entered into role 10? We can use the information from derivation rule 2 to formulate a special constraint on role 10, namely: an employee x can only occur in Marriage(10) if (x,female) occurs in Employee Gender(4,5). But this is the old C3 from figure 6.3! Finally, we drop derivation rule 2, which refers to a role that does not exist anymore. If we carry out all these actions for subtype Man as well, then we regain figure 6.3 from figure 6.5. So the end result of grouping away all derivable subtypes (with conversion of derivation rules into special constraints) is an IGD that is equivalent to the IGD with derivable subtypes except for the verbalization of the derivable subtypes themselves. The net result therefore is the same as with reducing (losing a few less important FTEs, see section 4.3). If users accept this loss, then derivable subtypes can be grouped. Because this grouping of derivable subtype roles involves reducing as well, the process is better considered apart from 'ordinary' grouping, with a separate choice by the users(see section 7.1.3).

So condition 1 above is too strong: roles from derivable subtypes can in principle also be deleted during grouping, but roles from declarative subtypes cannot. We therefore drop condition 1 and replace it with a new one. For the sake of completeness, we restate here the whole step 1 from section 4.1.2 (the conditions could now be given in a more logical order, but we maintain the old numbering):

1 Mark each role that satisfies all five conditions below; such a role can be deleted during the grouping process.
    1 The role is not in a declarative subtype.
    2 The role is played by a non-lexical object type.
    3 The role falls under a single role uniqueness constraint.
    4 The role is not optional.
    5 The role is not *directly recursive*, i.e. the role is not played by a nominalized fact type in which it sits itself.

At the beginning of the grouping algorithm, the FCO-IM tool marks all the roles that satisfy these conditions. The user can accept this groupings proposal, or remove markings.

As an example of deriving a relational schema from an IGD with subtypes, we use the IGD of the vehicle rental company from section 6.1.3, see figure 7.1.

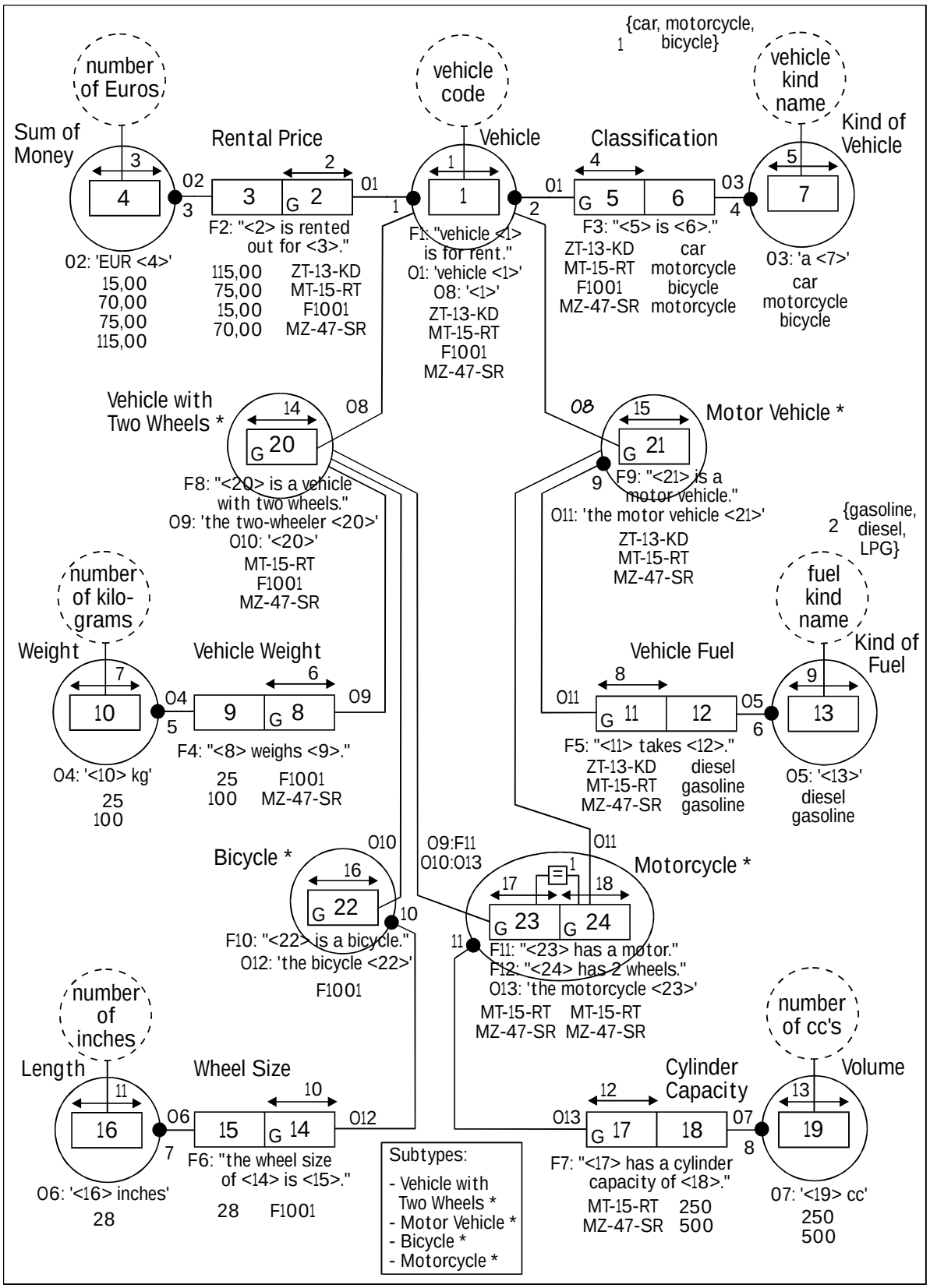


Figure 7.1: IGD vehicle rental company with roles marked for grouping

The corresponding derivation rules for figure 7.1:

1:	x in Vehicle with Two Wheels(20)	if	(x, motorcycle) in Classification (5,6)
		or if	(x, bicycle) in Classification (5,6).
2:	x in Motor Vehicle(21)	if	(x, motorcycle) in Classification (5,6)
		or if	(x, car) in Classification (5,6).
3:	x in Bicycle(22)	if	(x, bicycle) in Classification (5,6).
4:	x in Motorcycle(23)	if	(x, motorcycle) in Classification (5,6).
5:	x in Motorcycle(24)	if	(x, motorcycle) in Classification (5,6).

The roles that satisfy the grouping conditions are marked already in figure 7.1. Another change was made as well: the asterisk that indicates the derivability of a fact type is removed, and an asterisk is added (where possible) to the fact type expressions that belong to these derivable fact types. As soon as we begin grouping, it is no longer meaningful to mark entire fact types as derivable. If a derivable fact type has absorbed roles after grouping from a non-derivable fact type (which often happens with derivable subtypes), then a non-elementary fact type results with both derivable and non-derivable components (see for example figure 7.2, in which this arises for each subtype). For clarity, we indicate which FTEs are now derivable (in an El-IGD as in figure 7.1, that would be all the FTEs of a derivable fact type). The derivable components follow from the derivation rules always as well.

7.1.2 Grouping of Non-Subtype Roles

Figure 7.2 shows the result of deleting all the marked roles that are not in a subtype, i.e. after the ‘ordinary’ grouping without reducing has taken place. The derivation rules (shown in the figure this time) have been adjusted to the grouped situation. We keep the information about which subtypes there are (and whether they are derivable) in the subtype list; we need to have this information explicitly for all subtypes after grouping in principle (no unary fact types with a non-lexical role are left here after grouping).

This intermediate result (maximally grouped IGD with all subtypes still in place) is also of importance for transformations to other information models than to the Relational Model, and fits in well with an Object Oriented style of working for example.

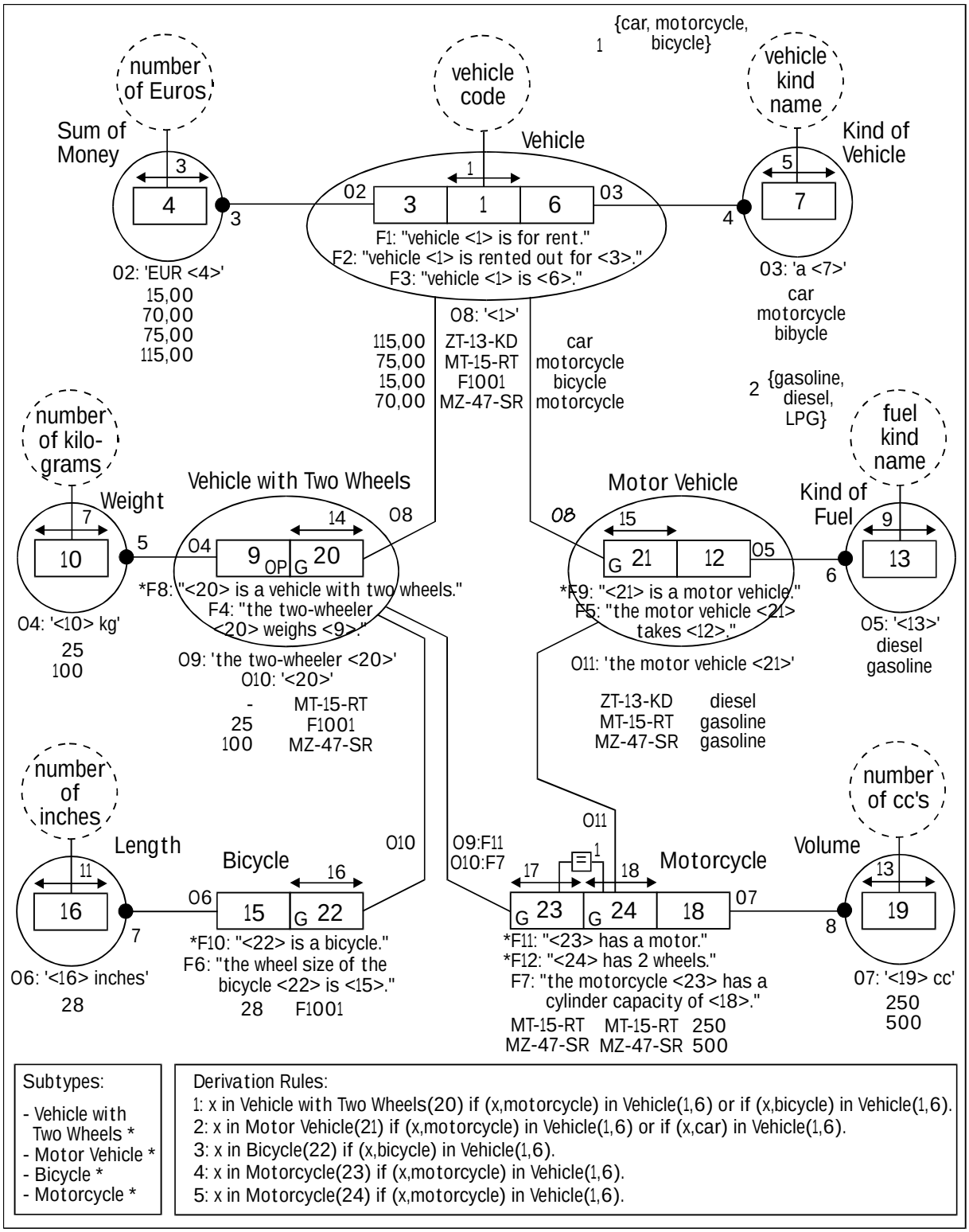


Figure 7.2: IGD vehicle rental company after grouping of non-subtype roles

7.1.3 Intermediate Reducing for Subtypes with More than One Supertype

If we would not group the IGD in figure 7.2 any further (so the subtype roles would remain), but would only lexicalize it, reduce it and convert it to a relational schema, then two columns would arise in table Motorcycle from roles 23 and 24 with exactly the same population. This is superfluous in the Relational Model, and so we must prevent the occurrence of such redundant columns. Please note that this would also have to be done if Motorcycle were a declarative subtype, i.e. if roles 23 and 24 were not marked for grouping. Also because of this last point a mandatory reducing step is now inserted.

This mandatory reducing step means that we must arbitrarily remove all subtype roles except one for each subtype with more than one supertype (derivable or not). In figure 7.2, we choose role 24for this purpose, because this role was not chosen earlier for object type expression O13 (see figure 7.1). We delete role 24 together with its population, derivation rule 5 and F12 (which both only relate to role 24), and all constraints that concern role 24: UC 18 and the strict equality constraint 1. The result is shown in figure 7.3. However, we can now no longer see directly that the population of role 23 is a subset of the population of role 21. The just deleted role 24 was played by Motor Vehicle, and so the population of role 24 was a part of that of role 21 (this was clear from O11). Because of the strict equality constraint, the population of role 23 is also a part of that of role 21. This also follows implicitly from the derivation rules, by the way. Still, we will explicitly state this for clarity in a subset constraint (SC 1 in figure 7.3), and indicate its derivability with an asterisk (see also the remark about this in section 7.1.4, for figure 7.5).

We call this intermediate *reducing* because a small loss of information occurs here, just as in 'ordinary' reducing (see section 4.3: we lose fact type expression F12 in figure 7.3, and so we cannot generate fact expressions such as "The motor vehicle MT-15-RT has 2 wheels." anymore).

At this point we could group/reduce still further by deleting all the remaining marked roles (of the derivable subtypes) yet, losing the derivable FTEs in the process (see section 7.1.1). We will do so in section 7.1.5, but in section 7.1.4 we will first derive the relational schema from figure 7.3 without carrying out that optional step. We will overrule the grouping proposal here and remove the remaining role markings. (We do not show that separately. In the FCO-IM tool there is a standard checkbox to indicate whether derivable subtype roles are to be deleted during grouping or not).

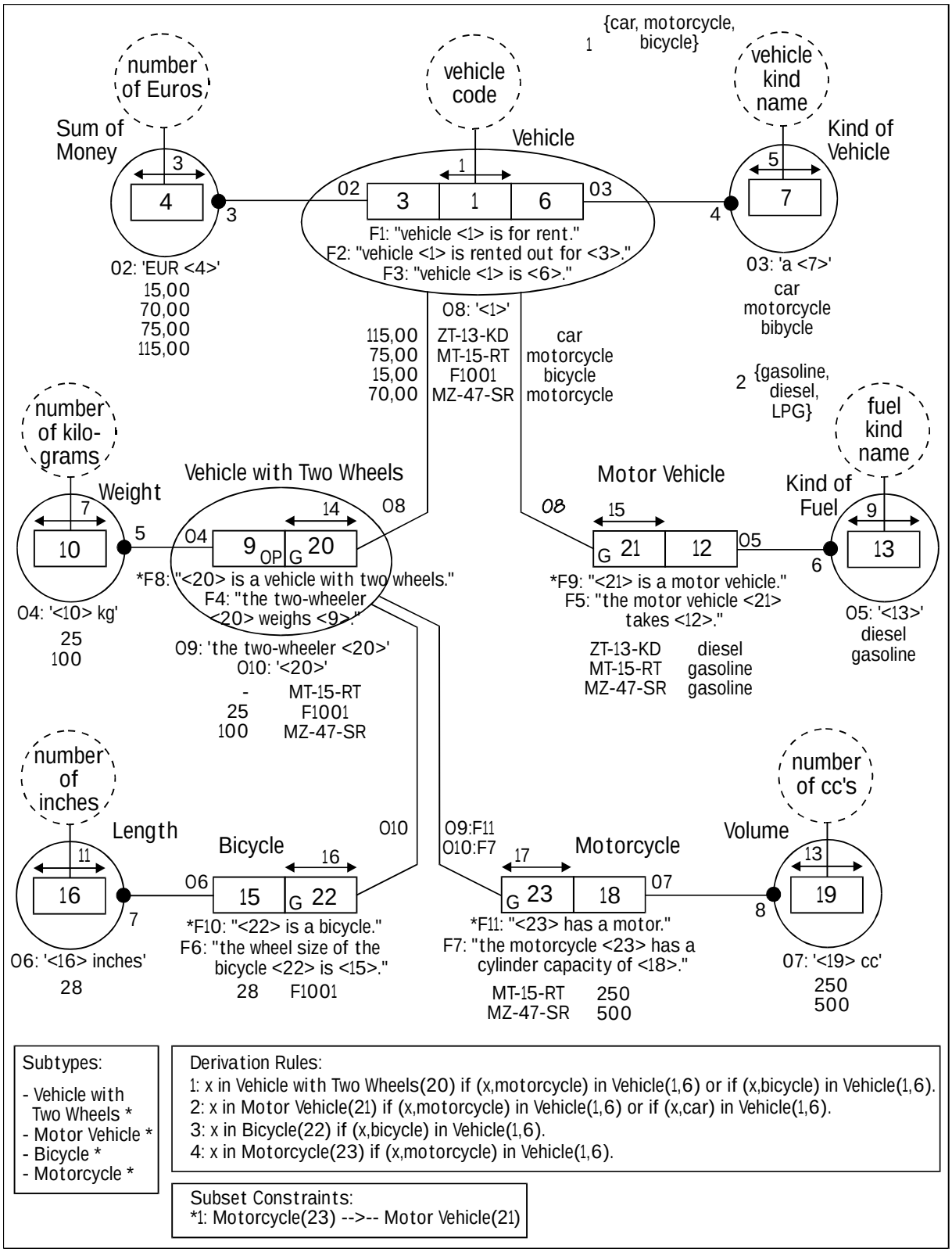


Figure 7.3: G-IGD vehicle rental company after intermediate reducing

7.1.4 Derivation of a Relational Schema Without Deleting Subtype Roles

We now lexicalize the IGD from figure 7.3. The resulting GL-IGD is shown in figure 7.4.

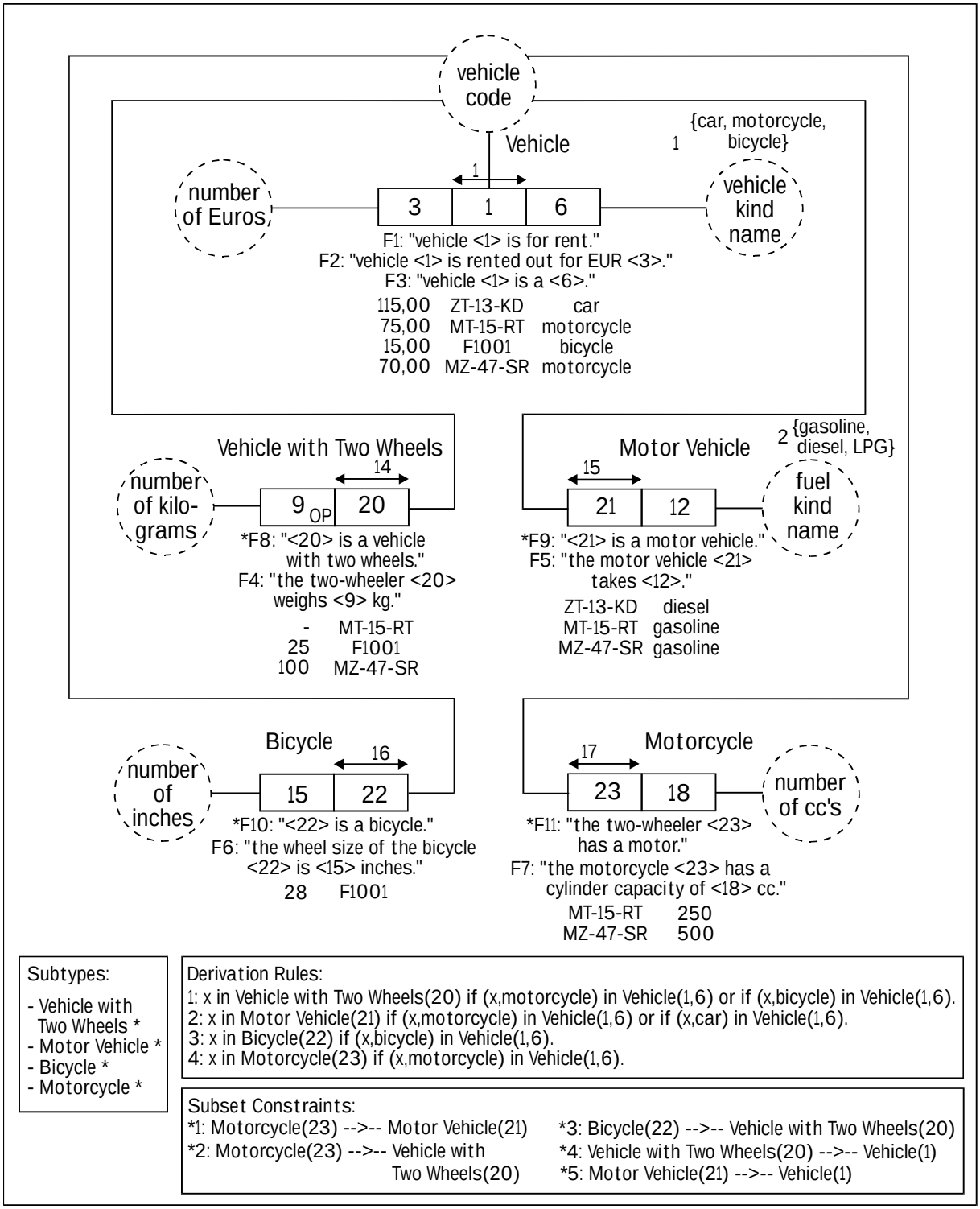


Figure 7.4: GL-IGD vehicle rental company without deleting derivable subtype roles

Four SCs arise during the lexicalization of roles 20, 21, 22, and 23, which all four also follow implicitly from the derivation rules. All unary fact types are lost The relational schema that follows from figure 7.4 is shown in figure 7.5.

Relational Schema Vehicle Rental Company 1. Derivable subtype roles were retained.

Vehicle		
vehicle code (vehicle code) NN ← PK →	Sum of Money (number of Euros) NN	Kind of Vehicle (vehicle kind name) NN
ZT-13-KD	115,00	car
MT-15-RT	75,00	motorcycle
F1001	15,00	bicycle
MZ-47-SR	70,00	motorcycle

F1: "Vehicle <vehicle code> is for rent."
F2: "Vehicle <vehicle code> is rented out for EUR <Sum of Money>."
F3: "Vehicle <vehicle code> is a <Kind of Vehicle>."

Motor Vehicle	
Vehicle (vehicle code) NN ← PK →	Kind of Fuel (fuel kind name) NN
ZT-13-KD	diesel
MT-15-RT	gasoline
MZ-47-SR	gasoline

*F9: "<Vehicle> is a motor vehicle."
F5: "The motor vehicle <Vehicle> takes <Kind of Fuel>."

Vehicle with Two Wheels	
Vehicle (vehicle code) NN ← PK →	Weight (number of kilograms) OP
MT-15-RT	-
F1001	25
MZ-47-SR	100

*F8: "<Vehicle> is a vehicle with two wheels."
F4: "The two-wheeler <Vehicle> weighs <Weight> kg."

Bicycle	
Vehicle with Two Wheels (vehicle code) NN ← PK →	Length (number of inches) NN
F1001	28

*F10: "<Vehicle with Two Wheels> is a bicycle."
F6: "The wheel size of the bicycle <Vehicle with Two Wheels> is <Length> inches."

Motorcycle	
Vehicle with Two Wheels (vehicle code) NN ← PK →	Volume (number of cc's) NN
MT-15-RT	250
MZ-47-SR	500

*F11: "The two-wheeler <Vehicle with Two Wheels> has a motor."
F7: "The motorcycle <Vehicle with Two Wheels> has a cylinder capacity of <Volume> cc."

Domains:
vehicle code = char(8)
number of Euros = numeric(6,2)
vehicle kind name = char(10)
number of kilograms = numeric(5)
fuel kind name = char(8)
number of inches = numeric(3)
number of cc's = numeric(4)

Domein Constraints:
vehicle kind name = {car, motorcycle, bicycle}
fuel kind name = {gasoline, diesel, LPG}

Foreign Keys:
*1: Motorcycle(Vehicle with Two Wheels) -->-- Motor Vehicle(Vehicle)
*2: Motorcycle(Vehicle with Two wheels) -->-- Vehicle with Two Wheels(Vehicle)
*3: Bicycle(Vehicle with Two Wheels) -->-- Vehicle with Two Wheels(Vehicle)
*4: Vehicle with Two Wheels(Vehicle) -->-- Vehicle(vehicle code)
*5: Motor Vehicle(Vehicle) -->-- Vehicle(vehicle code)

Other Integrity Rules: Derivation Rules:
1: x in Vehicle with Two Wheels(Vehicle) if (x,motorcycle) in Vehicle(vehicle code, Kind of Vehicle) or if (x,bicycle) in Vehicle(vehicle code,Kind of Vehicle).
2: x in Motor Vehicle(Vehicle) if (x,motorcycle) in Vehicle(vehicle code, Kind of Vehicle) or if (x,car) in Vehicle(vehicle code,Kind of Vehicle).
3: x in Bicycle(Vehicle with Two Wheels) if (x,bicycle) in Vehicle(vehicle code,Kind of Vehicle).
4: x in Motorcycle(Vehicle with Two Wheels) if (x,motorcycle) in Vehicle(vehicle code,Kind of Vehicle).

Figure 7.5: relational schema without deleting derivable subtype roles

So this way of deriving a relational schema (retaining all derivable subtypes) leads here to a separate table for each subtype, with few optional columns. Integrity rules guard the subtype structure.

The subset constraints have become foreign keys, which are still implied by the derivation rules. We could leave them out, but that is very unusual in the Relational Model, so for the sake of clarity we leave them as they are.

7.1.5 Despecialization: an Optional Extra Grouping/Reducing Step

Roles from derivable subtypes can be deleted during grouping without a serious loss of information, as was explained in section 7.1.1. We do lose the verbalization belonging to the derivable subtypes (the FTE). We call this removal of subtypes *despecialization*.

To illustrate this we will carry it out in the IGD from figure 7.3, after the mandatory reducing of roles from subtypes with more than one supertype, but still before lexicalizing. We will now also delete the marked subtype roles. We start with the subtypes that are lowest in the subtype network: Bicycle (role 22) and Motorcycle (role 23). The result is shown in figure 7.6. The procedure is identical to the one for 'ordinary' grouping: roles 22 and 23 are removed and roles 15 and 18 are absorbed by fact type Vehicle with Two Wheels. Roles 15 and 18 both become optional because the removed roles had no single role totality constraints (in fact subtype roles can never have a single role TC, see also section 5.2). Fact type expressions F10 and F11 are dropped as well: that is the reducing side of this step.

Derivation rules 3 and 4 are removed also, but the information they carry is used to draw up special constraints C1 and C2. These rules ensure that role 15 is only populated with bicycles (moreover, with *all* bicycles, because role 15 in figure 7.3 is not optional, or because role 14 in figure 7.1 has a single role TC), and that role 18 is only populated with motorcycles (moreover, with *all* motorcycles because role 18 is not optional in figure 7.3, or because role 17 in figure 7.1 has a single role TC). So constraints C1 and C2 have the same function in figure 7.6 as the derivable subtypes Bicycle and Motorcycle have in figure 7.1: to ensure that the wheel size and cylinder capacity are recorded only for the correct vehicles. Such special constraints cannot be drawn up for declarative subtypes, because of the absence of derivation rules for them. This illustrates once again, that only roles from derivable subtypes can be deleted in this step, losing their verbalization in the process.

The subtypes that were just deleted are removed from the subtype list as well.

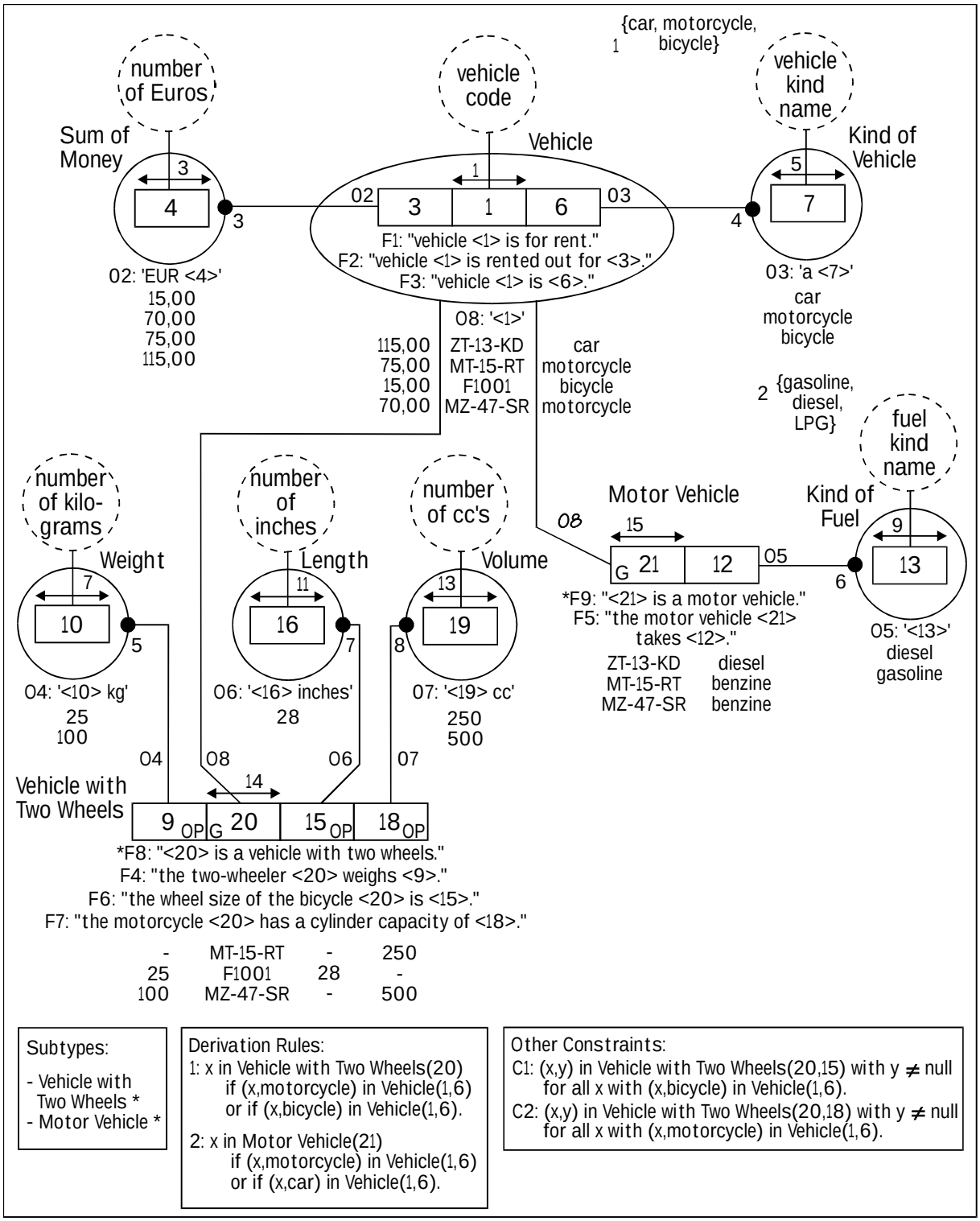


Figure 7.6: IGD vehicle rental company after grouping Bicycle and Motorcycle away

Next we also group/reduce the subtype roles from subtypes Vehicle with Two Wheels and Motor Vehicle. The result is shown in figure 7.7. Derivation rules 1 and 2 are now replaced with special constraints C3 and C4. Please note that role 9 can only have a value for vehicles with two wheels, but that it does not have to contain a value for all vehicles with two wheels (role 9 was already optional in figure 7.3 because role 8 in figure 7.1 had no single role TC). All derivable fact type expressions are now removed, and all derivation rules are now converted into special constraints with the same function as the original derivable subtypes. All subtypes are removed from the subtype list.

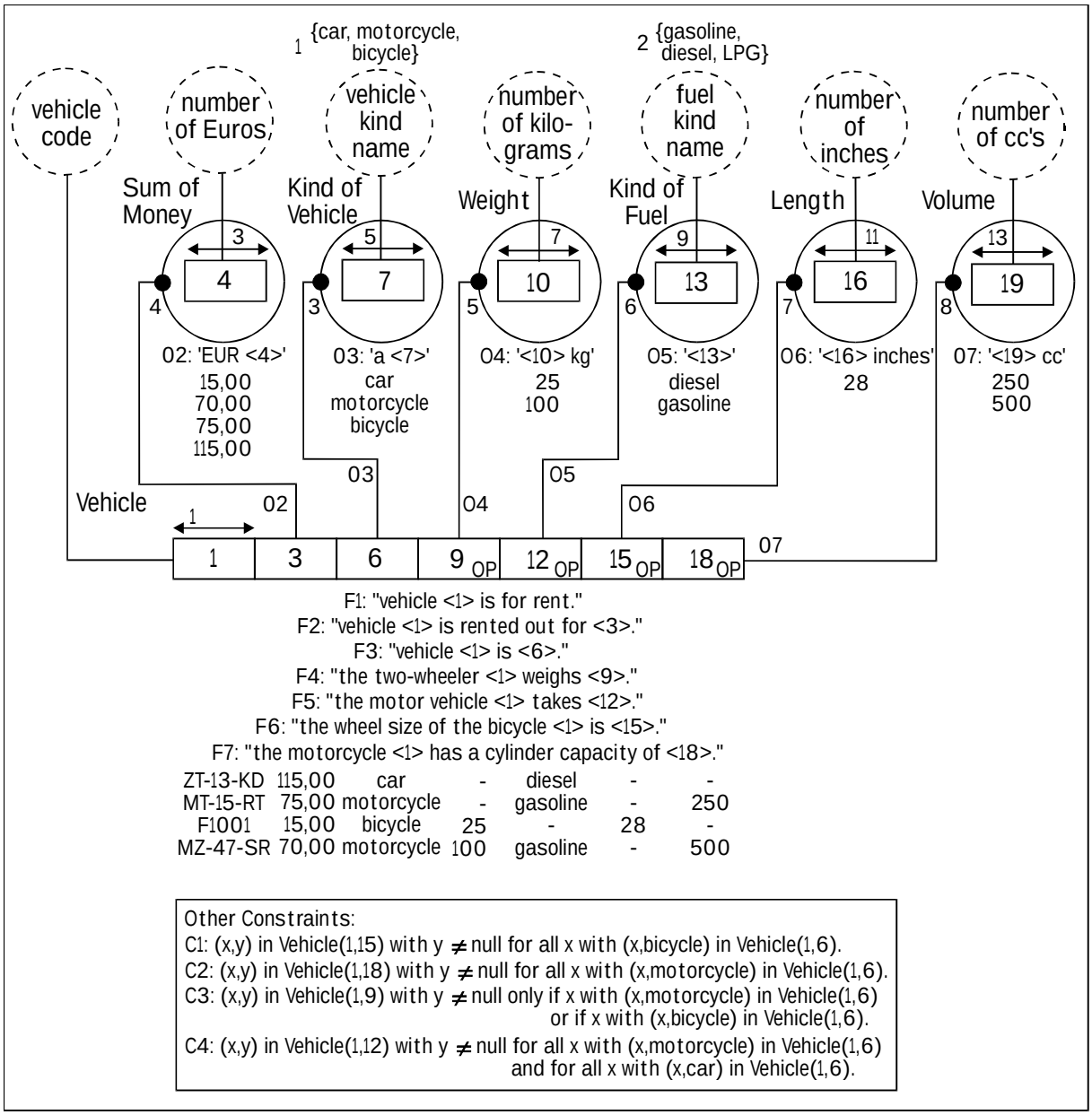


Figure 7.7: IGD vehicle rental company after complete despecialization

7.1.6 Derivation of a Relational Schema after Despecialization

Lexicalizing the IGD in figure 7.7 is simple: no role splitting, all nominalized fact types are lost, so after removing them no subset constraints remain. We do not give the GL-IGD separately. The resulting relational schema is shown in figure 7.8. Constraints C1, C2, C3 and C4 can be directly translated into integrity rules 1, 2, 3 and 4.

Relational Schema Vehicle Rental Company 2. Derivable subtype roles were deleted.

Vehicle						
vehicle code (vehicle code) NN ◄— PK —►	Sum of Money (number of Euros) NN	Kind of Vehicle (vehicle kind name) NN	Weight (number of kilograms) OP	Kind of Fuel (fuel kind name) OP	Length (number of inches) OP	Volume (number of cc's) OP
ZT-13-KD	115,00	car	-	diesel	-	-
MT-15-RT	75,00	motorcycle	-	gasoline	-	250
F1001	15,00	bicycle	25	-	28	-
MZ-47-SR	70,00	motorcycle	100	gasoline	-	500

F1: "Vehicle <vehicle code> is for rent."
F2: "Vehicle <vehicle code> is rented out for EUR <Sum of Money>."
F3: "Vehicle <vehicle code> is a <Kind of Vehicle>."
F4: "The two-wheeler <vehicle code> weighs <Weight> kg."
F5: "The motor vehicle <vehicle code> takes <Kind of Fuel>."
F6: "The wheel size of the bicycle <vehicle code> is <Length> inches."
F7: "The motorcycle <vehicle code> has a cylinder capacity of <Volume> cc."

Domains:
vehicle code = char(8)
number of Euros = numeric(6,2)
vehicle kind name = char(10)
number of kilograms = numeric(5)
fuel kind name = char(8)
number of inches = numeric(3)
number of cc's = numeric(4)

Domain Constraints:
vehicle kind name =
{car, motorcycle, bicycle}
fuel kind name =
{gasoline, diesel, LPG}

Other Integrity Rules:
1: Column Weight is null for all vehicles
with the value 'car' in column Kind of Vehicle.
2: Column Kind of Fuel is only not null for all vehicles
with the value 'car' or 'motorcycle' in column Kind of Vehicle.
3: Column Length is only not null for all vehicles
with the value 'bicycle' in column Kind of Vehicle.
4: Column Volume is only not null for all vehicles
with the value 'motorcycle' in column Kind of Vehicle.

Figure 7.8: relational schema after complete despecialization

So this way of deriving a relational schema (deleting all derivable subtypes) leads here to just one table for all vehicles, with many optional columns. Integrity rules guard the subtype structure.

7.1.7 Summary of the Procedure

Summarized, the procedure for deriving a relational schema from an IGD with subtypes is as follows:

1 Marking each role that satisfies all five conditions below:
   1 The role is not in a declarative subtype.
   2 The role is played by a non-lexical object type.
   3 The role falls under a single role uniqueness constraint.
   4 The role is not optional.
   5 The role is not *directly recursive*, i.e. the role is not played by a nominalized fact type in which it sits itself.

2 Grouping marked roles that are not in a subtype as usual.

3 Mandatory reducing of all subtypes with more than one super-type:
   1 Delete all subtype roles together with their corresponding FTEs and population, except one arbitrarily chosen subtype role. Ensure that all remaining FTEs and OTEs (if any) that belong to the subtype refer to the remaining subtype role.
   2 Generate a subset constraint for each deleted role, directed from the remaining subtype role to the subtype role of the supertype that played the deleted role.

4 Despecializing (optional grouping/reducing of derivable subtype roles):
   Each derivable subtype role can still be deleted, if desired, losing the corresponding derivable FTE in doing so. The derivation rule must then be transformed into an equivalent special constraint to guard the subtype structure.

5 Further treatment (lexicalizing, reducing, converting) as usual.

7.1.8 Final Remarks

1 Figures 7.5 and 7.8 illustrate two extreme cases: deleting none (figure 7.5) or all (figure 7.8) of the derivable subtype roles. These are two possible choices often made in practice as well, although intermediate solutions are also possible by only partially following the grouping proposal in the FCO-IM tool. Which choice is the best depends on many case-specific factors. The general picture is: on the one side many tables, but few optional columns (figure 7.5), and on the other side few tables, but many optional columns (figure 7.8). In the last situation there generally are more special integrity rules, namely as many integrity rules as there are roles played by a subtype, whereas in the first situation the number of integrity rules is just the same as the number of subtypes.

2 The table in figure 7.8 appears to be very similar to the table in figure 6.10: the concrete example document. In figure 6.10, however, the integrity rules are missing that reflect the subtype structure. These requirements can only be systematically determined with the subtype matrix method. If we would know beforehand that all the derivable subtypes will be grouped away (but watch out if there are also declarative subtypes), then the subtypes would not have to be modeled explicitly as is done in figure 6.14, and we could suffice with taking up special constraints in figure 6.11, such as: “x in Wheel Size(14) if (x,bicycle) in Classification (5,6).”, the equivalent of derivation rule 3 in figure 6.14. But we would then miss the visual overview of the structure of the subtype network, and moreover we would not have the possibility to derive the relational schema in figure 7.5 anymore. On the other side of the balance would be a simpler IGD that can be entered into the FCO-IM tool with less typing work.

7.2 Derivation of a Relational Schema with Generalization

If we derive a relational schema from an IGD with generalization, then this almost always yields tables that are not well-formed according to the Relational Model, unless we introduce (sometimes drastic) alterations in the IGD and in the communication itself. The Relational Model is more limited than FCO-IM in its modeling capabilities (which is why we have to carry out lexicalization for example, which would not be necessary if domains could be table types themselves also), and this makes the transformation from an IGD to a logical relational model more complex.

In section 7.2.1, we will look at what happens when we apply the GLR-algorithm to IGDs with generalization. In section 7.2.2, we will discuss two standard ways to obtain a well-formed logical relational schema after all, using the examples from section 6.2. In section 7.2.3, we will very briefly sketch a less strict form of grouping, which automatically leads to generalization, and we will close the chapter with a few final remarks in section 7.2.4.

7.2.1 Weakly Identifiable Tables

We apply the GLR-algorithm to the IGD in figure 7.9, which contains generalization with synonymy (the example from section 6.2.3). The roles that satisfy the conditions for grouping are marked. Roles 18 and 19 are not marked because they are optional (condition 4 from section 7.1.1).

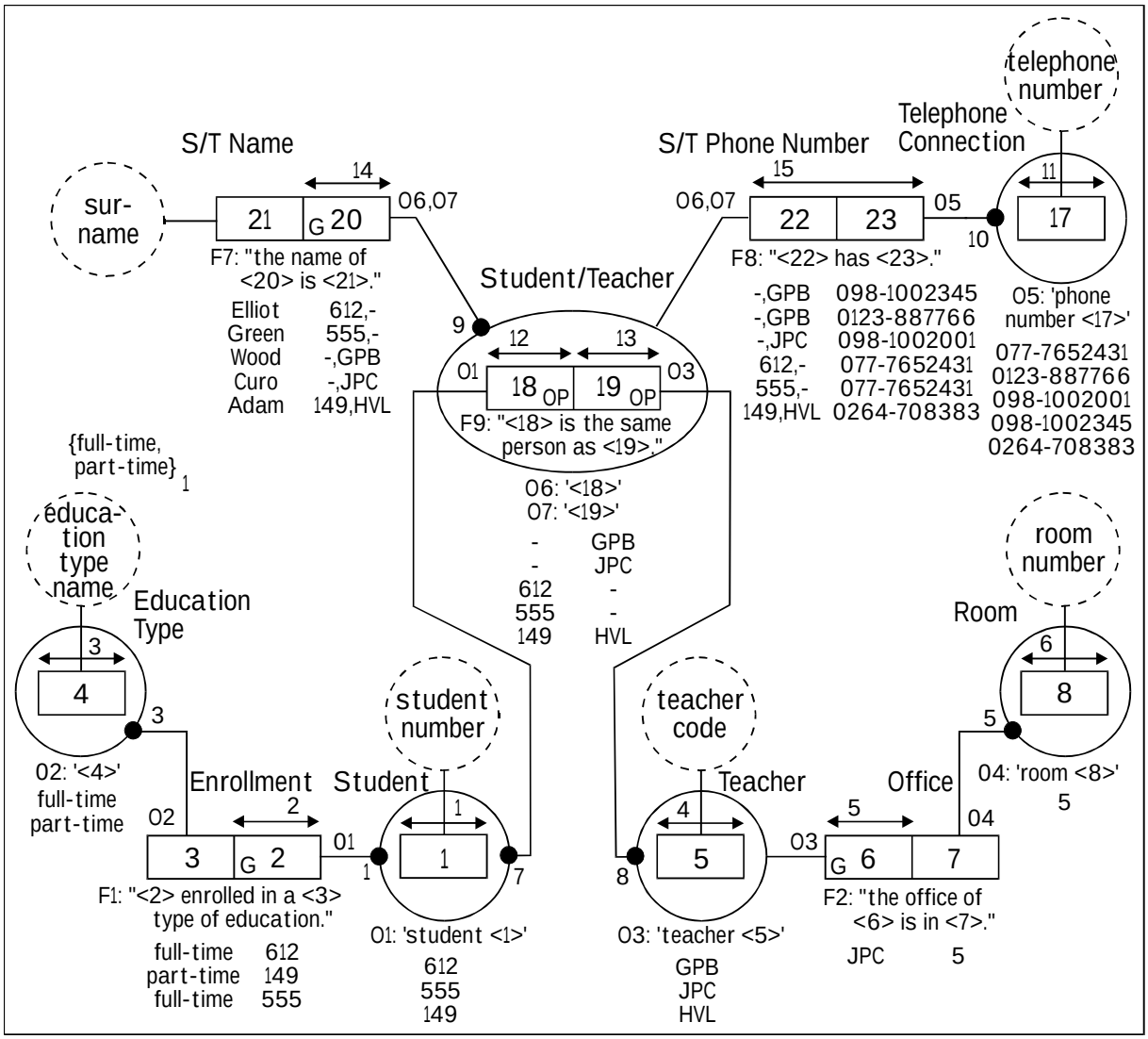


Figure 7.9: IGD Student/Teacher with roles marked for grouping

The grouped IGD is shown in figure 7.10. After deleting role 20, fact type expression F7 is moved to fact type Student/Teacher. Because in figure 7.9 both O6 and O7 can be used for role 20 (depending on the position of the null value, see section 6.2.1), F7 is split in figure 7.10 into F7.1 (with O6 substituted) and F7.2 (with O7).

After grouping it is no longer clear that roles 18 and 19 cannot both have a null-value in the same tuple (there would be a completely empty tuple in figure 7.9). That is why we must now add constraint C1.

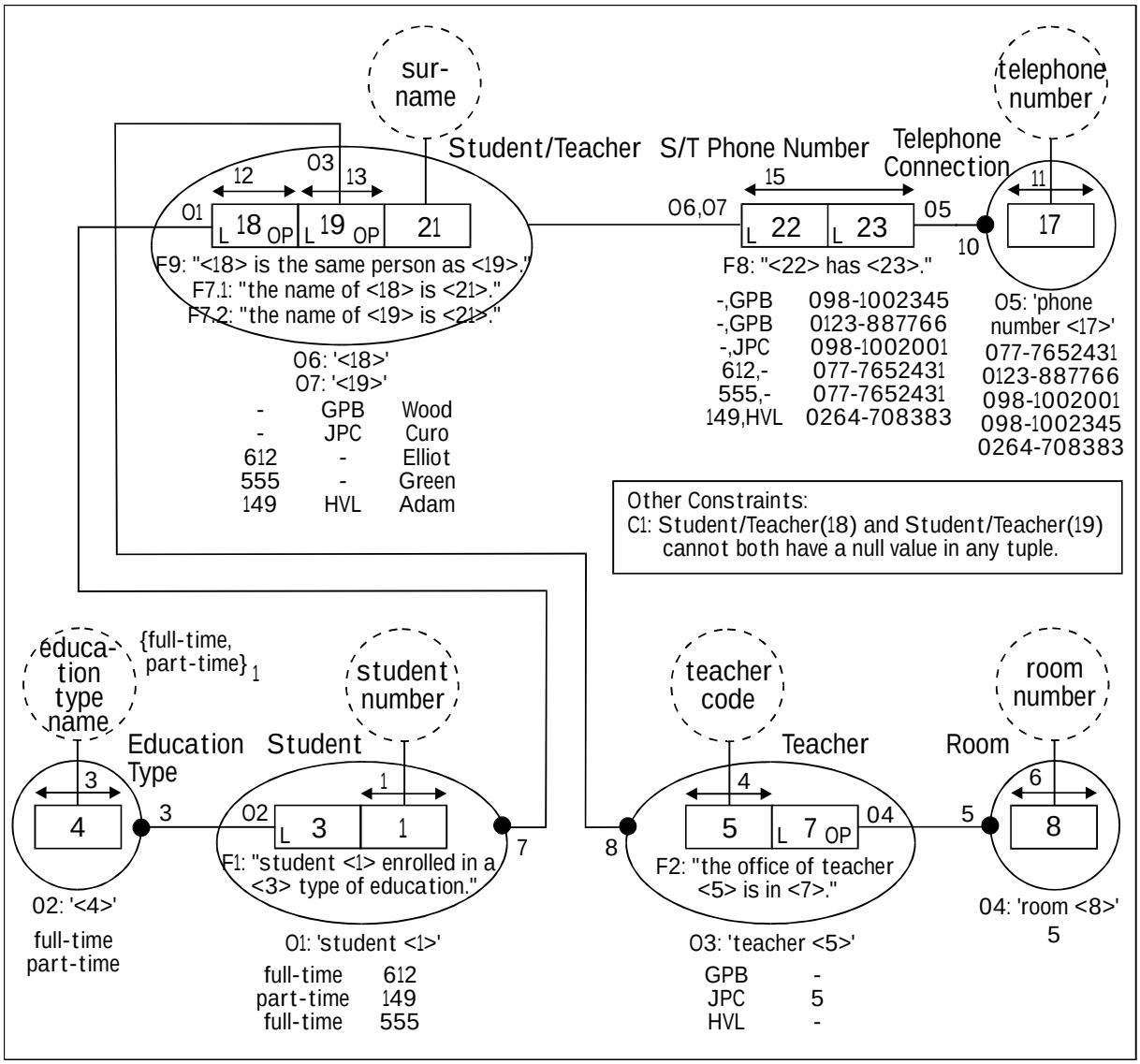


Figure 7.10: G-IGD Student/Teacher with roles marked for lexicalizing

We will only discuss the lexicalization of role 22 in detail because the other roles go as usual. The identifier of object type Student/Teacher consists of both roles 18 and 19, so we split role 22 during lexicalizing into role 22.1 and role 22.2, both optional because 18 and 19 are optional as well (the population also clearly illustrates this here). Next, roles 22.1 and 22.2 are connected in the first instance respectively to object type Student and object type Teacher, after which they are further 'diverted' to respectively label type 'student number' and label type 'teacher code', see figure 7.11.

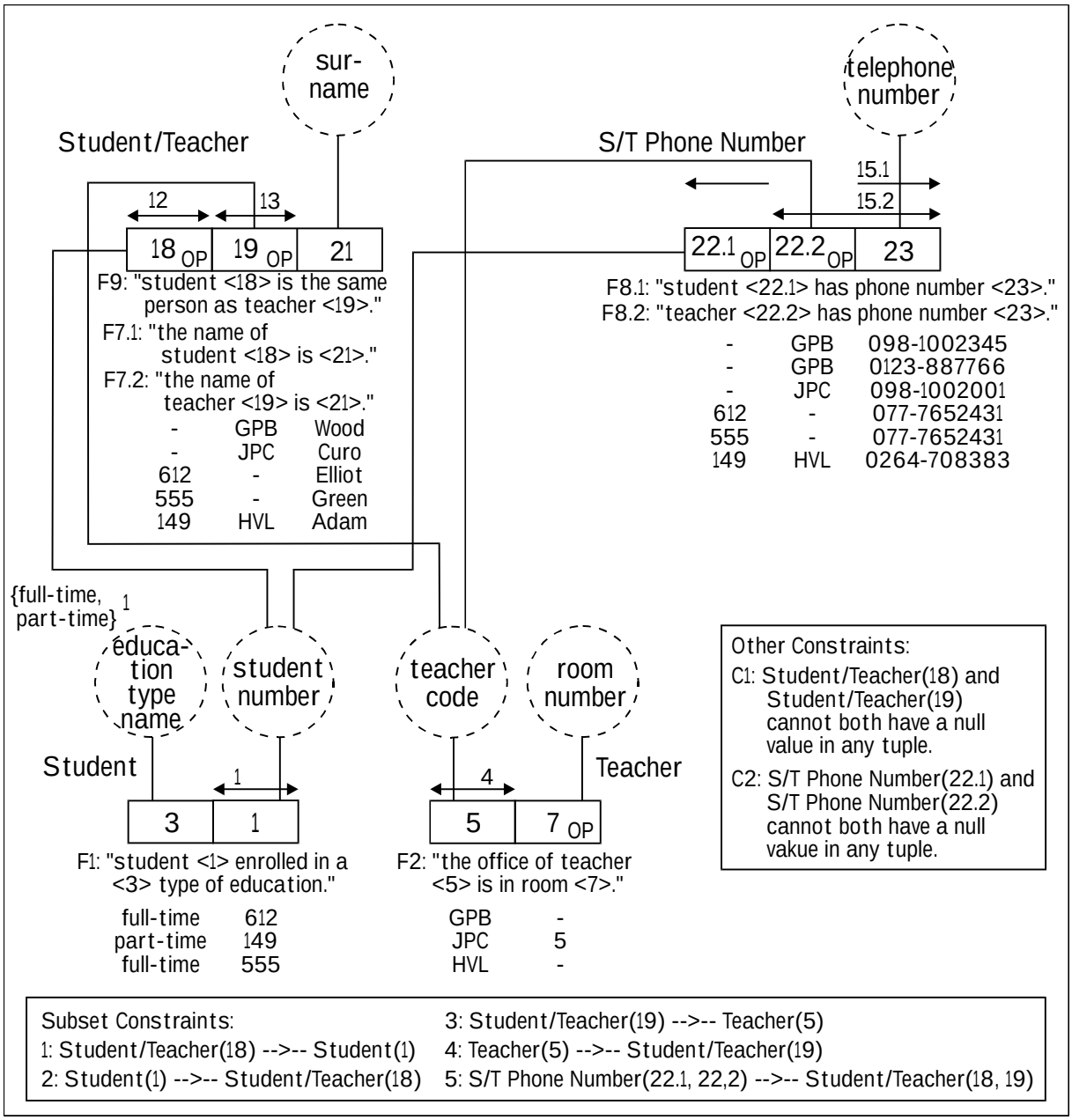


Figure 7.11: GLR-IGD Student/Teacher

Uniqueness constraint 15 is now also split into two new UCs 15.1 and 15.2. UC 15.1 corresponds to the old UC 15 in combination with UC 12, and UC 15.2 corresponds to the old UC 15 together with UC 13. Aside: lexicalizing can also be seen as replacing a role (here: role 22) with a copy of the identifying role of the object type that played the role (here: roles 22.1 and 22.2 as a copy of roles 18 and 19) including their constraints (here: both UCs 12 and 13, which are each processed separately with UC 15, and the optionality of roles 22.1 and 22.2).

During the processing of O6 and O7, F8 splits into two FTEs F8.1 and F8.2 for the same reason as in the earlier splitting of F7. After deleting the lost fact types, no other fact types come into consideration for reducing. Figure 7.11 shows the GLR-IGD.

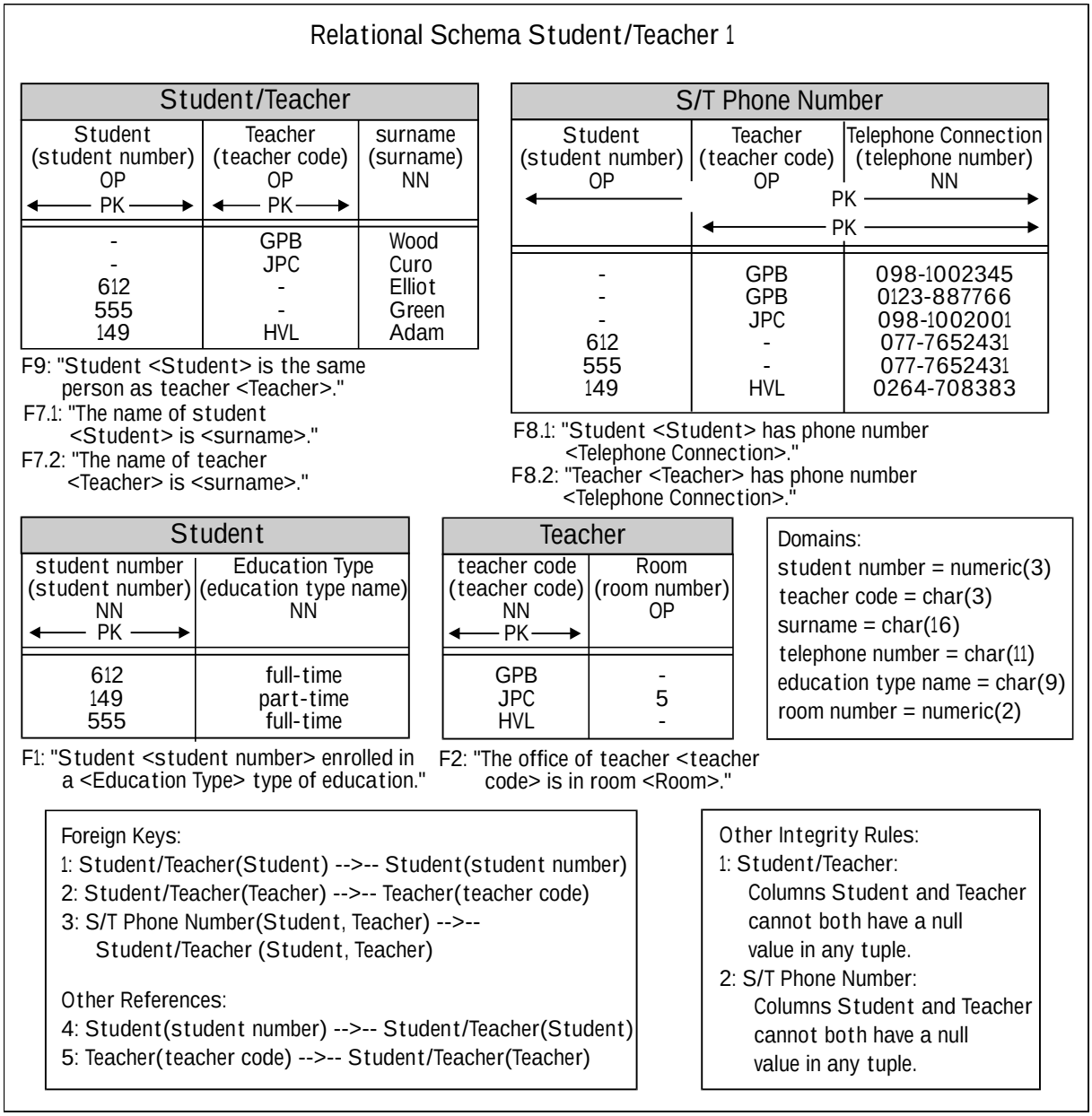


Figure 7.12: relational schema with weakly identifiable tables

The relational schema that follows from this is shown in figure 7.12. There are remarkable primary keys in this relational schema. According to the relational model, the columns that make up the primary key must all be 'not null': no null values can appear in them. But table Student/Teacher, for example, does not satisfy this. Moreover, this table has one primary key: columns Student and Teacher *together* form the identifier of Student/Teacher, whereas each column has its *own* uniqueness rule (arising from the old UCs 12 and 13). That is impossible as well in the Relational Model.

The same is true in table S/T Phone Number: it has one primary key (all columns), some of which are optional, and with two separate uniqueness rules (from the old UCs 15.1 and 15.1). We indicate the primary keys by adding the letters 'PK', from 'primary key', to the arrows of the corresponding uniqueness rules.

The rule from section 6.2.2.1 about uniqueness constraints on optional roles applies in a completely analogous way to uniqueness rules on optional columns as well. Therefore, all the tuples in the tables from figure 7.1.2 can indeed be identified with the remarkable primary keys. Because the tables nevertheless are still not well-formed according to the Relational Model, we call a table with such primary keys *weakly identifiable*. We call a table with a primary key of which all columns are 'not null' and with one uniqueness rule on all the columns together *strongly identifiable*. So in figure 7.12 tables Student and Teacher are strongly identifiable, but Student/Teacher and S/T Phone Number are weakly identifiable.

7.2.2 Towards Strongly Identifiable Tables

There almost always arises a relational schema with weakly identifiable tables from an IGD with generalization after application of the GLR-algorithm (exceptions: subtypes with more than one supertype, which are a form of abridged generalization, as we showed in section 7.2.1, and cases of complete synonymy (all objects having two identifiers)). There is nothing wrong with weakly identifiable tables however, neither from the point of view of modeling the communication, nor even from the point of view of unique tuple identification. Some database administrators therefore use such tables with pleasure. There are also relational database management systems that allow (coincidentally) the implementation of weakly identifiable tables. Still, it is often necessary or desirable to use only strongly identifiable tables.

We will therefore discuss two standard ways to arrive at strongly identifiable tables in this section: *degeneralizing* (to undo the generalization) and *nicknaming* (to introduce new, artificial identifiers). Both ways have disadvantages: degeneralization usually introduces redundancy, which must subsequently be guarded, and nicknaming changes the communication.

7.2.2.1 Degeneralization

If generalization yields weakly identifiable tables, then an obvious remedy is to undo the generalization, that is to degeneralize. Applied to the IGD for the educational institution in figure 7.9, this means we have to remove object type Student/Teacher. Next, we must duplicate the fact types that Student/Teacher played a role in, and distribute each pair over object types Student and Teacher, see figure 7.13 (for a generalization of three object types: triplicate the fact types and distribute each trio, and so on).

The so obtained IGD in figure 7.13 strongly resembles the IGD in figure 6.15, from which we started to introduce generalization. There is only an extra fact type Alias because of the synonymy. Please note that the surname and telephone number must now be recorded twice for people who are both a student and a teacher. To guarantee that this redundancy cannot cause any information pollution, constraints C1 and C2 are needed (loosely formulated in the IGD).

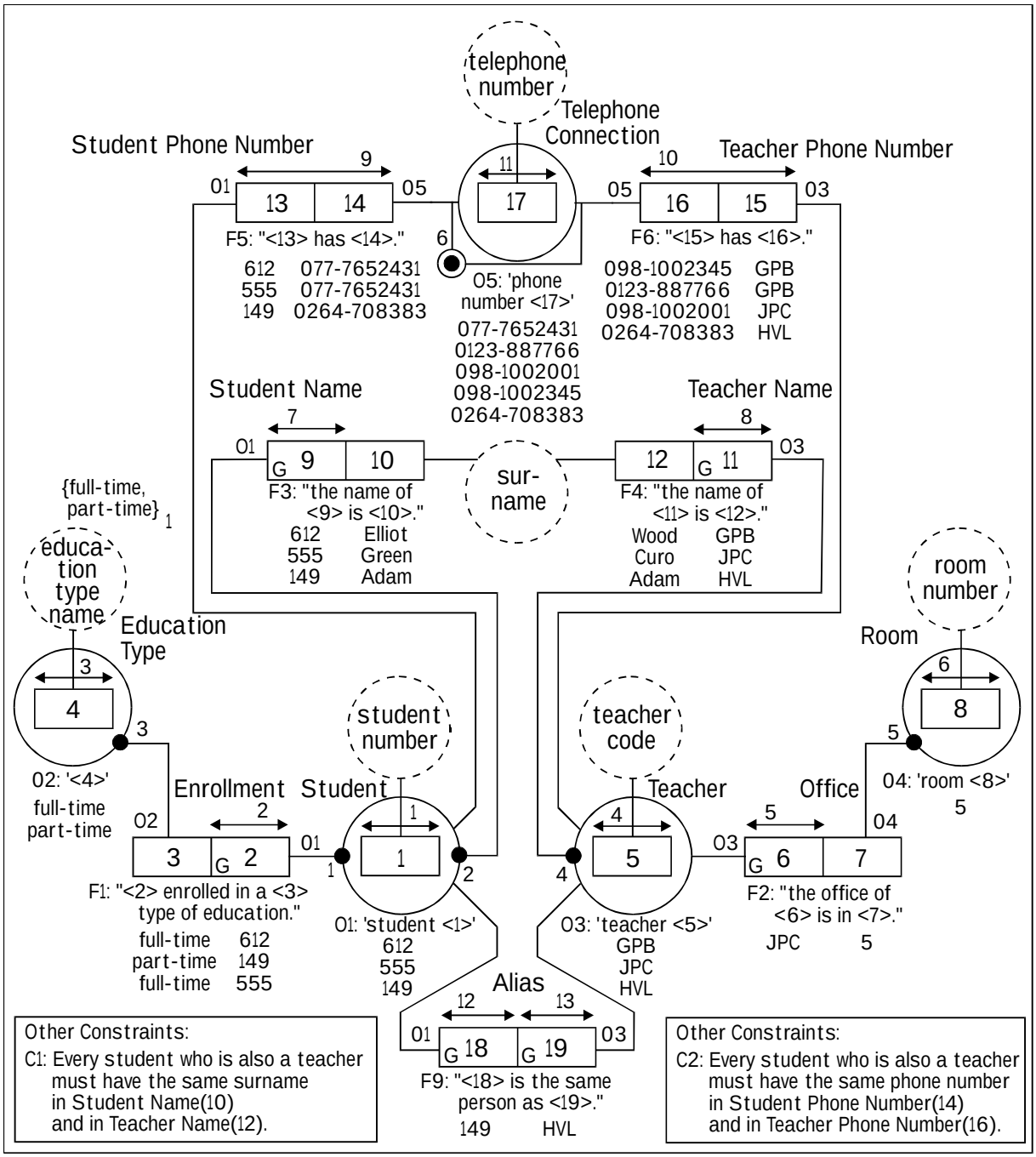


Figure 7.13: IGD Student/Teacher after degeneralization

The derivation of the relational schema from the IGD in figure 7.13 is done in the usual way. All the roles that satisfy the grouping conditions are already marked in figure 7.13. Both roles from fact type Alias can be deleted. Since there is no single role totality constraint, we arbitrarily choose to treat role 18 first (see step 2.a.2 from section 4.1.2). The grouped IGD is shown in figure 7.14, in which role 19 has now lost its grouping mark because it has become optional (just as in the student-project example case study in chapter 4). Lexicalizing presents no points of special interest, and we can reduce no fact types. The relational schema is shown in figure 7.15.

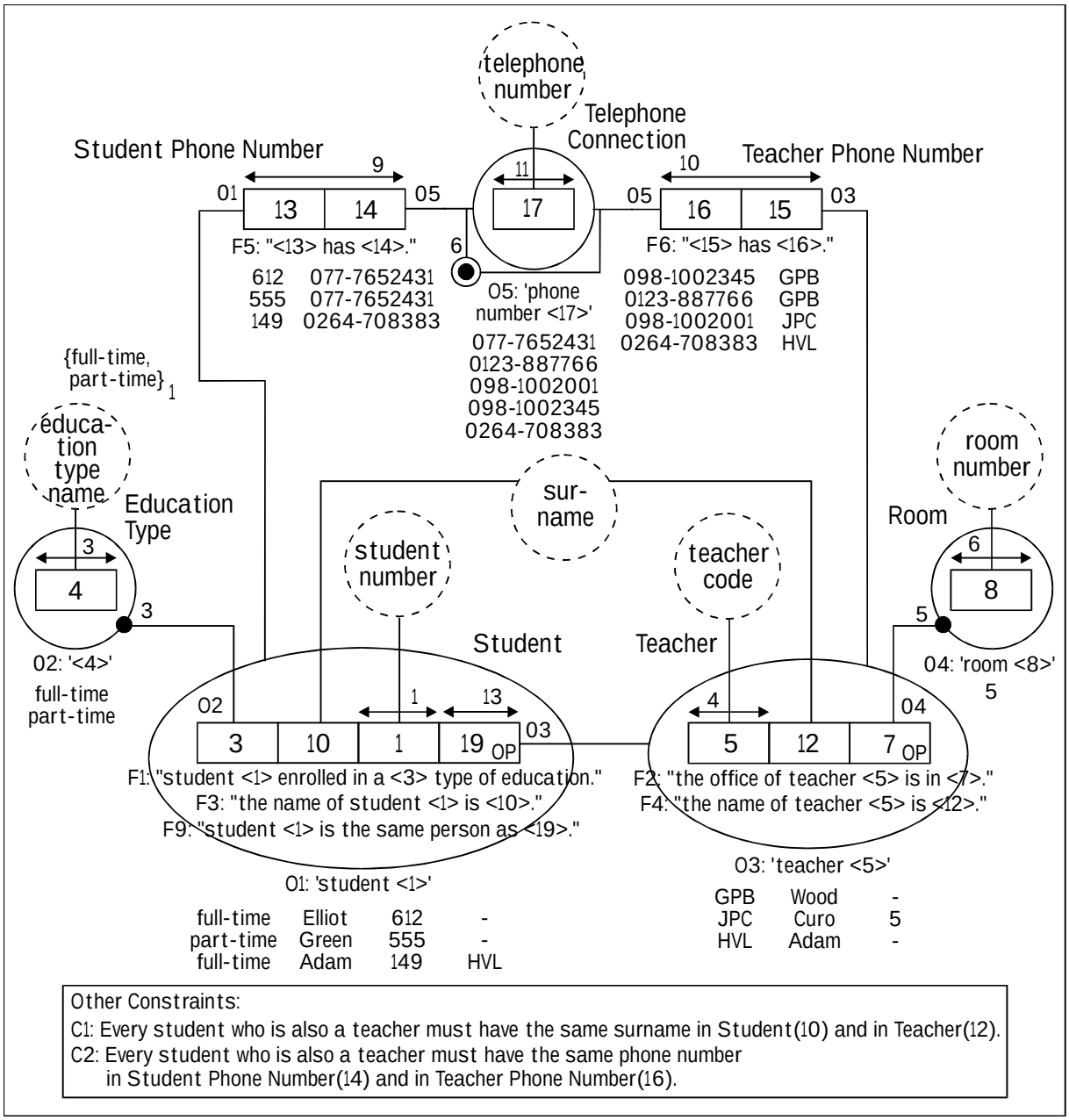


Figure 7.14: G-IGD Student/Teacher after degeneralization

There are no weakly identifiable tables in the relational schema of figure 7.15, in contrast to the schema in figure 7.12. The price to pay is only the redundant recording of the surname and telephone number for students that are also teachers. If role 19 had been deleted first during grouping, then column Teacher would be missing from table Student, and there would be a column Student in table Teacher. F9 would then be associated with table Teacher, with a few minor alterations.

Relational Schema Student/Teacher 2

Student

student number (student number) NN PK	surname (surname) NN	Education Type (education type name) NN	Teacher (teacher code) OP
612	Elliot	full-time	-
555	Green	part-time	-
149	Adam	full-time	HVL

Student Phone Number

Student (student number) NN PK	Telephone Connection (telephone number) NN PK
612	077-7652431
555	077-7652431
149	0264-708383

F1: "Student <student number> enrolled in a <Education Type> type of education."
F5: "Student <Student> has phone number <Telephone Connection>."
F3: "The name of student <student number> is <surname>."
F9: "Student <student number> is the same person as teacher <Teacher>."

Teacher

teacher code (teacher code) NN PK	surname (surname) NN	Room (room number) OP
GPB	Wood	-
JPC	Curo	5
HVL	Adam	-

Teacher Phone Number

Teacher (teacher code) NN PK	Telephone Connection (telephone number) NN PK
GPB	098-1002345
GPB	0123-887766
JPC	098-1002001
HVL	0264-708383

F2: "The office of teacher <teacher code> is in room <Room>."
F4: "The name of teacher <teacher code> is <surname>."
F6: "Teacher <Teacher> has phone number <Telephone Connection>."

Domains:
student number = numeric(3)
teacher code = char(3)
surname = char(16)
telephone number = char(11)
education type name = char(9)
room number = numeric(2)

Foreign Keys:
1: Student(Teacher) -->-- Teacher(teacher code)
2: Student Phone Number(Student) -->-- Student(student number)
3: Teacher Phone Number(Teacher) -->-- Teacher(teacher code)

Other Integrity Rules:
1: Every student who is also a teacher must have the same surname in Student(surname) and in Teacher(surname).
2: Every student who is also a teacher must have the same phone number in Student Phone Number(TelePhone Connection) and in Teacher Phone Number(Telephone Connection).

Figure 7.15: relational schema after degeneralization: with redundancy

7.2.2.2 Nicknaming

Degeneralization to obtain strongly identifiable tables is not always possible or desired. Recursive fact types (see section 6.2.5) cannot be degeneralized, due to the series of object types that is infinite in principle. Degeneralization is often undesirable for abridged generalization (see section 6.2.2). In the case of the rooms from figure 6.23 in section 6.2.4, for example, we would rather not have a separate table for the occupation of rooms with names, and another for rooms with a number (even apart from the redundancy).

Next to degeneralizing, there is a second general way to obtain strongly identifiable tables: by introducing new artificial identifiers for all the objects from the generalized object types. As an example we take the cabinets from figure 6.19 in section 6.2.2. The relational schema that follows from the IGD in figure 6.19 is shown in figure 7.16. Fact type expression F3 is split for the same reason as the splitting of FTEs F7 and F8 in section 6.2.1. Table Cabinet is weakly identifiable.

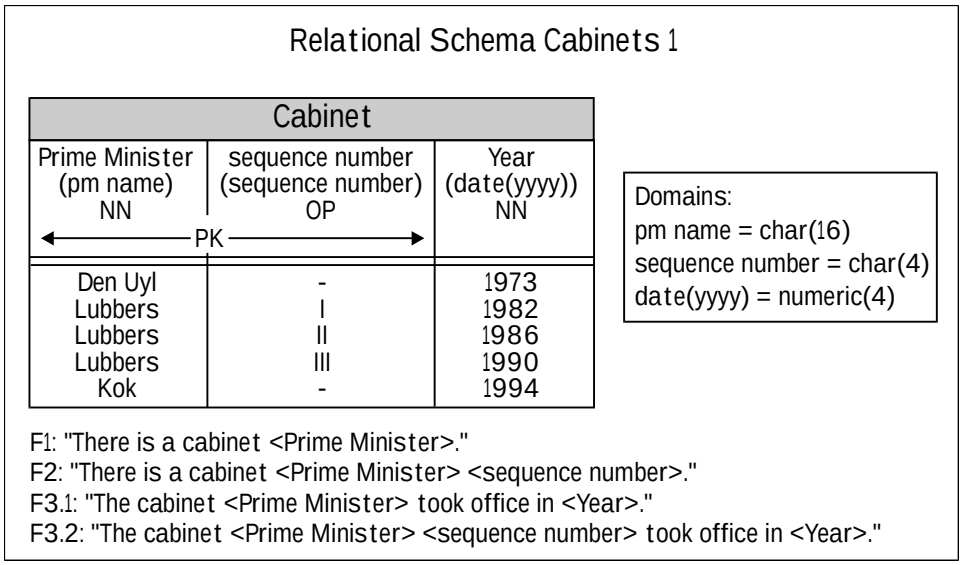


Figure 7.16: relational schema Cabinets, with weakly identifiable table

We now introduce a cabinet code for all cabinets to serve as a new identifier. As a result, the communication changes radically. Instead of a fact expression such as “The cabinet Lubbers II took office in 1986.” we now get fact expressions such as “The cabinet C3 took office in 1986.”, “Prime minister Lubbers presided over the cabinet C3.” and “The cabinet C3 had sequence number II.”. The last two fact expressions are necessary in order not to lose the old name (the new name is after all introduced for implementation reasons only). We call this giving of a new name to get rid of the generalization: *nicknaming*.

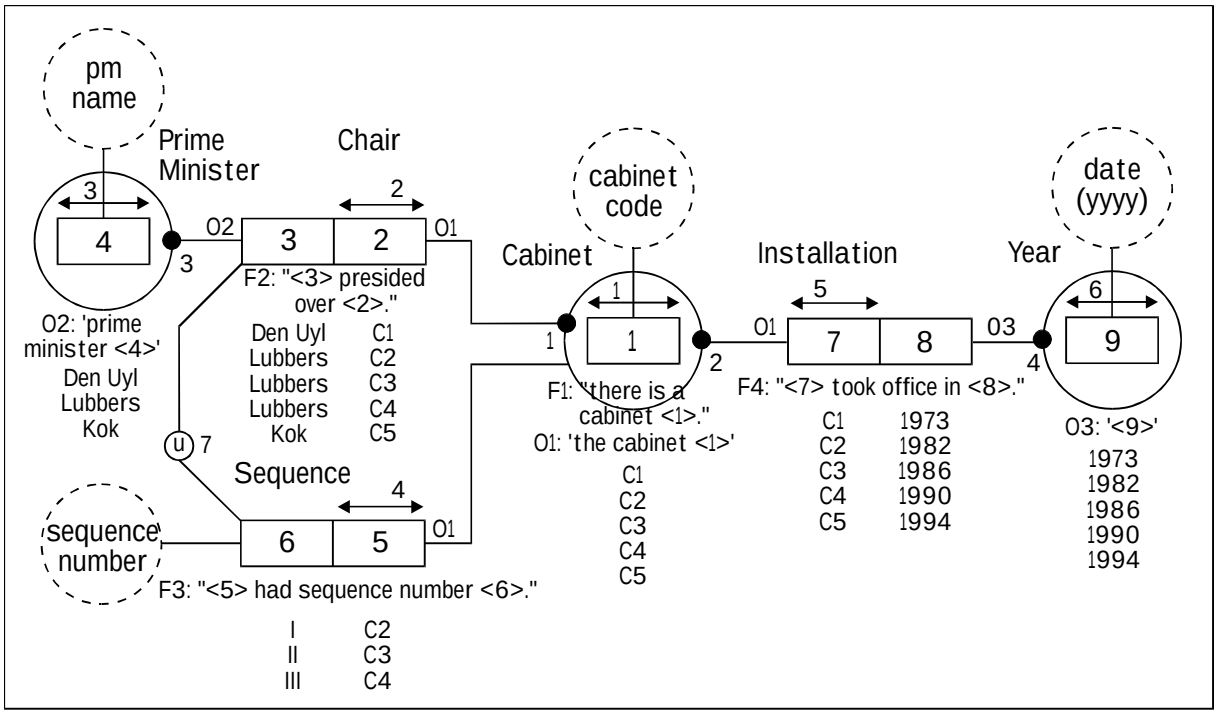


Figure 7.17: IGD cabinets after nicknaming

Figure 7.17 contains the IGD after introducing the new cabinet code. Uniqueness constraint 7, an inter fact type constraint, means that a combination of prime minister and reference number (if any) is unique for each cabinet. So UC 7 is the equivalent of UC 1 from figure 6.19.

The relational schema that follows from the IGD in figure 7.17 is shown in figure 7.18.

Relational Schema Cabinets 2

Cabinet			
cabinet code (cabinet code) NN PK	Prime Minister (pm name) NN	sequence number (sequence number) OP	Year (date(yyyy)) NN
C1	Den Uyl	-	1973
C2	Lubbers	I	1982
C3	Lubbers	II	1986
C4	Lubbers	III	1990
C5	Kok	-	1994

Domains:
cabinet code = char(4)
pm name = char(16)
sequence number = char(4)
date(yyyy) = numeric(4)

F1: "There is a cabinet <cabinet code>."
F2: "Prime minister <Prime Minister> presided over the cabinet <cabinet code>."
F3: "The cabinet <cabinet code> had sequence number <sequence number>."
F4: "The cabinet <cabinet code> took office in <Year>."

Figure 7.18: relational schema after nicknaming: different communication

We can also obtain a strongly identifiable schema for cabinets using degeneralization. We indicate that here only very briefly. Remove the object type Cabinet from figure 6.20, and duplicate fact type Installation. This will give rise to two tables with only non-optional columns and good primary keys: Cabinet1(Prime Minister, Year) for all cabinets without a sequence number and Cabinet2(Prime Minister, sequence number, Year) for all cabinets with a sequence number. There is no redundancy because there is no synonymy. Users will have to consult two tables for information about cabinets, however.

Nicknaming can always be applied, in contrast to degeneralization, which cannot be done for recursive fact types (see the next paragraph). We could use nicknaming in the Student/Teacher example also: see the IGD in figure 7.9. Create a unique identifier for students and teachers together, for example 's/t code'. Fact type Student/Teacher would then have only one role, played by label type 's/t-code'. Object types Student and Teacher would now be declarative subtypes of Student/Teacher. Two binary fact types would be added with the following FTEs: "<Student:O1> has student number <student number>." and "<Teacher:O3> has teacher code <teacher code>"., in order not to lose the old identifications. During the nicknaming process in ordinary generalization, the generalization network is always transformed into a subtype network with the same structure; this illustrates once again the connection between generalization and specialization.

As a final example of nicknaming, we will consider recursive fact types (see section 6.2.5). A relational schema cannot be derived from an IGD with recursive fact types by the GLR-algorithm: the recursive role will not be marked for grouping (condition 5 from section 7.1.1), and during lexicalizing it would split in an infinite number of roles (see, for example, the IGD in figure 6.26 from section 6.2.5). Nicknaming is the only possibility.

Let us take the example of the titles of chapters and sections from section 6.2.5. We introduce a new 'c/s-code', which identifies both chapters and sections. The IGD in figure 7.19 shows the situation after nicknaming. The old names can be reconstructed from the information in fact type Classification (only a part of the population is shown.

The relational schema that follows from the IGD in figure 7.19 consists of two strongly identifiable tables; we do not present this in a separate figure.

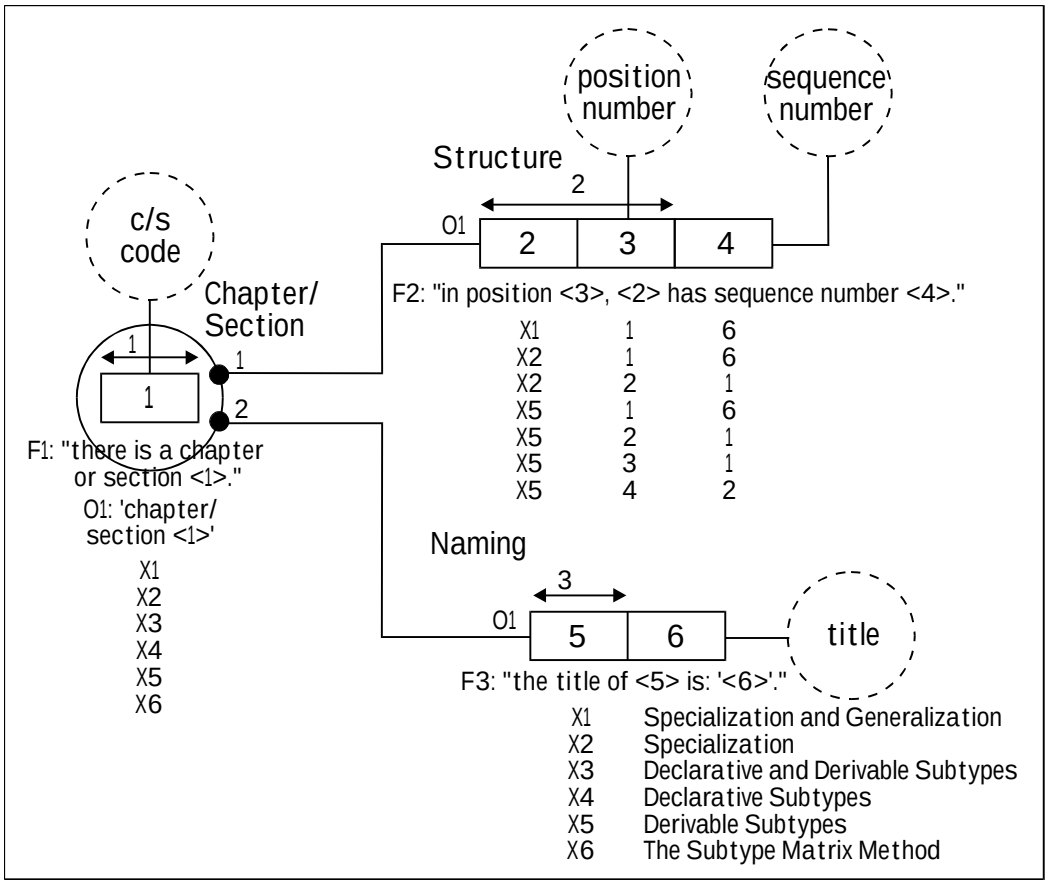


Figure 7.19: IGD after nicknaming a recursive fact type

7.2.3 Strict and Mild Grouping

In this section, we will briefly point out the reasons that an optional role cannot be marked for grouping (condition 4 from section 7.1.1). If an optional role is deleted during grouping nevertheless, then almost always a weakly identifiable table will arise. Consider for example the grouped IGD of the student-project case study in figure 4.4 from section 4.4.1. If we delete role 15 all the same, then roles 1, 2 and 5 are absorbed by fact type Project. Roles 1, 2 and 5 become optional because role 15 did not have a single role TC (tuples from projects that have not been allocated to any student receive null values under roles 1, 2 and 5). But roles 6, 8 and 10 become optional as well, since role 15 was optional (because tuples from students without an allocated project receive null-values under roles 6, 8, and 10). So after deleting an optional role during grouping, all roles in the absorbing fact type become optional. In short, since *all* roles of fact type Project become optional, only a weakly identifiable table can follow from it.

Now suppose that we would accept weakly identifiable tables. Then there would be no reason to forbid deleting optional roles during grouping (so long as they satisfy the remaining four conditions from section 7.1.1). We therefore distinguish a *strict grouping algorithm*, in which we do not delete optional roles during grouping (condition 4 does apply), and a *mild grouping algorithm*, in which we do delete optional roles during grouping (condition 4 does not apply).

Since a weakly identifiable relational schema results almost always anyway from IGDs with generalization, let us look at what the mild grouping algorithm yields in such cases. The strict grouping algorithm was applied to the IGD of the educational institution with students and teachers in section 7.2.1, which yielded the grouped IGD in figure 7.10. If we now also group roles 18 and 19 away, then the IGD in figure 7.20 arises.

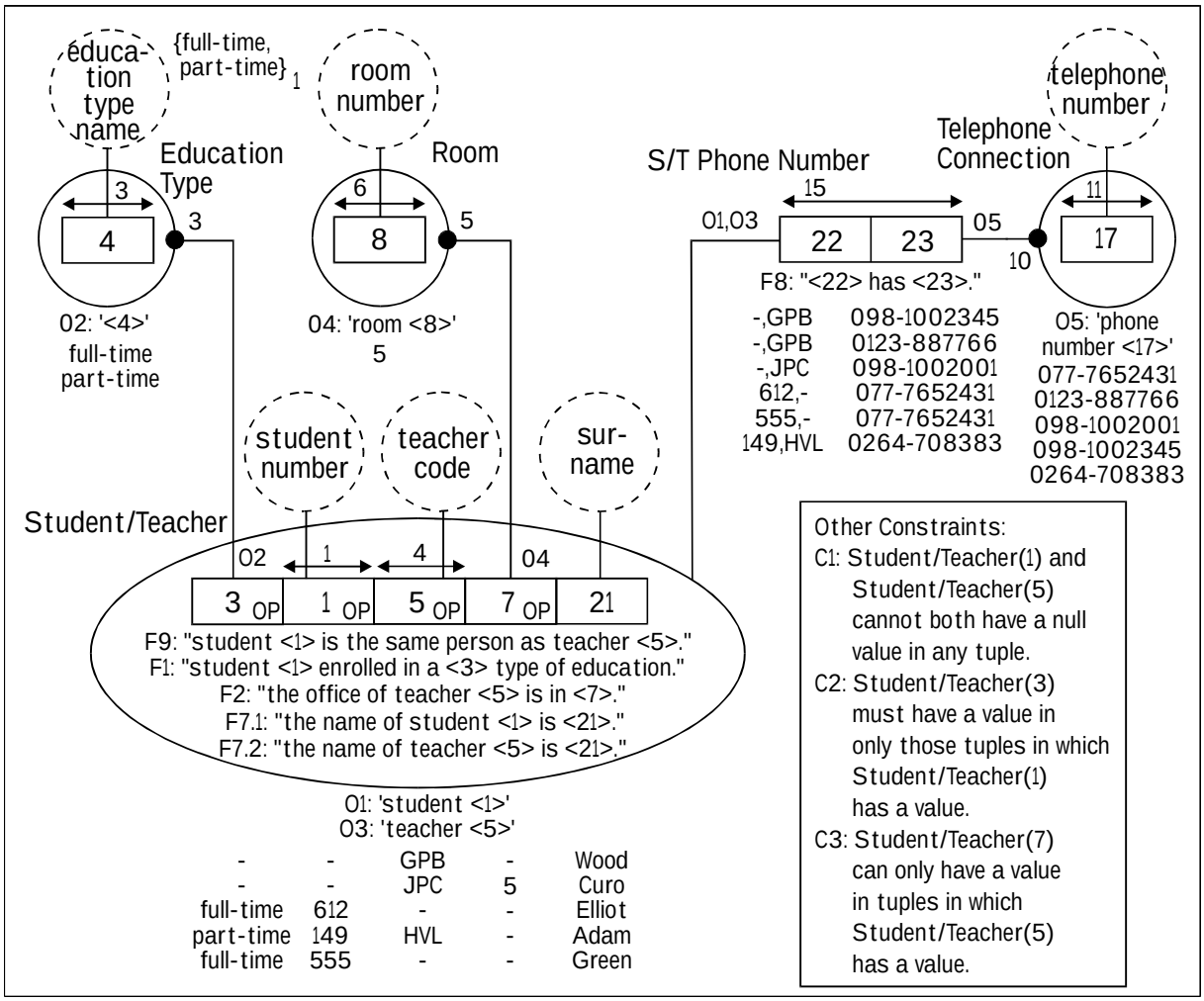


Figure 7.20: G-IGD Student/Teacher after mild grouping

If we first group away role 18, then roles 19 and 21 are absorbed by fact type Student, which we can now better rename Student/Teacher because all students and teachers now occur in the population of roles 1 and 19. Roles 1 and 3 become optional because role 18 was optional. O1 replaces O6. Next, we delete role 19 and let roles 1, 3 and 21 be absorbed by Teacher, which we rename Student/Teacher again. Roles 5 and 7 become optional (role 7 already was) because role 19 was optional. O3 replaces O7. Finally, we add constraints C2 and C3.

The resulting relational diagram is shown in figure 7.21. There are only two tables left, both weakly identifiable.

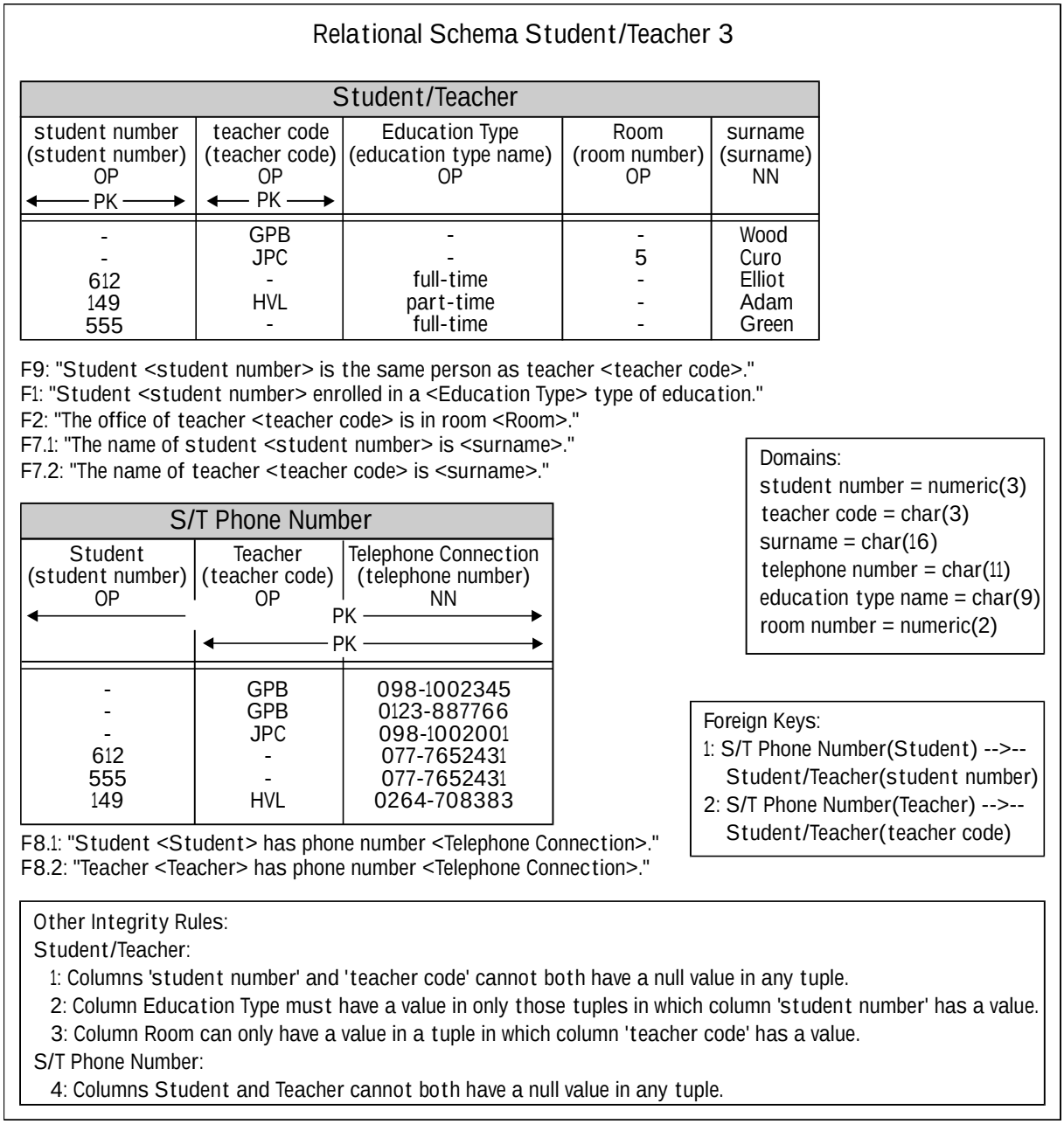


Figure 7.21: relational schema after mild grouping

It is striking that the same relational schema will follow from applying the mild grouping algorithm to the IGD from figure 7.13, that is to the degeneralized IGD: if we also group away role 19 in figure 7.14, then roles 1, 3, and 10 are absorbed by Teacher, which we rename Student/Teacher. All roles become optional in the first instance. Roles 10 and 12, however, can be combined (because of F9, they have either only one value or the same value in each tuple), and thus form one non-optional role. Fact types Student Phone Number and Teacher

Phone Number can be combined as well. In short: mild grouping can lead to (re)generalization. This is not surprising because mild grouping almost always leads to weakly identifiable tables, which are a hallmark of generalization.

We have only discussed mild grouping here by giving an example, because it requires further investigation. The FCO-IM tool therefore only carries out the strict grouping algorithm.

7.2.4 Final Remarks

1 One should not take changing the communication by nicknaming too lightly: in general domain experts do not appreciate that they are compelled to speak about the UoD in a totally different way just for implementation purposes, whereas the old way of communication was quite satisfactory. We have witnessed a few interesting conflicts between information analysts and domain experts concerning this matter [literature list 11]. We always agree with the domain experts in such cases.

2 There is in principle yet another way to obtain strongly identifiable tables from IGDs with generalization during the design of an implementation of a relational schema in a relational database management system that allows no weakly identifiable tables, seemingly without changing the communication. A dummy identifying not-null column (such as column 'cabinet code' in figure 7.18) could be added to the weakly identifiable tables, which is invisible for the users, but which does produce a formally correct primary key. Only the other columns are made available to the users by the interface. Behind the screens, the interface translates all references to the weakly identifiable keys into references to the hidden primary key. From a conceptual point of view, the communication from figure 7.18 is used behind the screens, but the communication from figure 7.16 is derived from that by the interface. A further discussion of this point is outside the scope of this book, however.

Appendix A:

Recommended Operational Procedure

We present here a step-by-step operational procedure in a very concise form for carrying out an information analysis. The order in which these steps are discussed in the book differs from the order we recommend to use in practice, because a textbook sets its own requirements on connections and didactical structure. The details of every step can be found in the rest of the book (but see the closing comment in appendix B as well). We present the step by step plan as a linear process, although an information analysis in practice will often be done in a cyclic way, because results of a later step might require doing an earlier step over again.

1 Collect or draw up concrete example documents (lists, overviews, forms and so on). This is not discussed explicitly in this book, although it is an important and often underrated step.
2 Let the contents of the example documents be verbalized by (analysts and) domain experts, preferably in elementary fact expressions.
3 Classify and qualify the fact expressions (done by the analyst in interviews with the domain experts. The analyst draws the provisional IGD that follows from this and populates every fact type with at least one tuple).
4 Validate the modeling of the communication. All verbalizations are regenerated from the IGD and submitted to the domain expert for approval. The IGD is corrected if necessary.
5 Determine uniqueness constraints by asking the domain expert concrete questions.
6 Carry out the elementarity tests (n-1 rule-test, n rule test, projection/join-test), (done by the analyst independently), and make any compound fact expressions elementary with the help of the domain experts; repeat steps 4 and 5 for any new fact types.
7 Carry out the nominalization test (done by the analyst independently), and introduce extra nominalizations if necessary (change the verbalization where needed or desired, together with the domain experts); repeat the procedure from step 4 for all the concerned fact types.
8 Determine the totality constraints (analyst together with domain experts). If in doubt, it is better to have too few TCs than too many.
9 Determine the subtypes (analyst together with the domain expert).
10 Determine all other constraints (analyst together with domain experts; concrete examples of violations are always useful).

After this, a relational schema can be derived with the help of the GLR-algorithm. To control the resulting structure, semantically equivalent transformations can de carried out. In addition, denormalization can be applied and so on.

Appendix B:
FCO-IM and NIAM

Fully Communication Oriented Information Modeling (FCO-IM) closely resembles *NIAM (Natural language Information Analysis Method)* in respects such as the conceptual framework, the modeling constructs, the diagramming technique and the methodical way of working (i.e. the step-by-step operational procedure). We will go further into this close relationship between NIAM and FCO-IM in this appendix.

NIAM originated from an analysis project in the area of information modeling that was carried out from 1967 by Control Data Nederland, commissioned by the director, Mr. R. Endert. He was in those days already of the opinion that the information perspective (or file organization as it was called then) is more important for the development of information systems (because it is more stable) than the process perspective and event perspectives (or program structures as they were called then). This proved to be a prophetic, since it still took a long time before data oriented design was more or less was generally accepted instead of process-oriented design. This innovative research under the supervision of G.M. Nijssen, who was at that time an electrotechnical engineer at Control Data, lead to NIAM (*Nijssen's Information Analysis Method*). Professor Nijssen himself meanwhile prefers to speak of Natural language Information Analysis Method, to stress in the N from NIAM that the basic principle of NIAM is: the consistent verbalization of information in natural language.

More important than that FCO-IM fits in with the structural aspects of NIAM (modeling constructs and diagramming technique), is that FCO-IM also completely stands in the NIAM tradition in the methodical sense (operational procedure). The most typical aspect of NIAM is the emphasis it puts from the beginning on methodical aspects of information analysis. This has lead to the *Conceptual Schema Design Procedure* (also known as the 'NIAM cookbook'), which prescribes step by step how the NIAM analyst must carry out the analysis process to yield a conceptual information model (or *information grammar* in NIAM terminology). This intimate connection between structure and methodical procedure explains why NIAM is especially successful in complex information modeling projects, which sometimes failed earlier with other diagramming techniques (we intentionally use this word here instead of 'modeling method'). On the other hand, an often heard criticism on NIAM can be understood from this point of view: "Let's make a simple ER diagram, because NIAM is much too cumbersome!" There is nothing against that, but the issues are of course: to what extent can a method cope with complex situations, and is the method teachable and learnable so that a trained information analyst will prove to have surplus value in such complex projects.

For a detailed documentation about the historical development of NIAM and and FCO-IM, see literature list [1] (unfortunately only inpublished in Dutch language). Here we restrict ourselves to a broad outline.

From the start, it was possible to have n-ary fact types with n>2 (i.e. fact types in which more than two object types play a role) in NIAM, next to unary and binary fact types, and it was possible to objectify fact types (Nijssen later chose the more linguistic term 'nominalization'). This nominalization leads to so-called nested fact type structures, comparable with aggregated relationship types in E-ERM. Within Control Data, however, a binary non-nested variant was chosen later on, for reasons that would carry us too far to explain here. This variant rooted strongly in The Netherlands because of the book 'Informatie-analyse volgens NIAM', published in 1985 (see literature list [2]), and also because this NIAM version was taught in AMBI courses under the name *Binary Method*. The first CASE-tools that supported NIAM were all based on this binary non-nested NIAM-variant.

Professor Nijssen, who was appointed to a chair at the University of Queensland (Australia) in 1982, chose to teach and further develop the n-ary and nested form of NIAM (called 'the real NIAM' by him) there. The result, among other things, was the book 'Conceptual Schema and Relational Database Design' that was first published in 1989 (literature list [3]). In The Netherlands, education in n-ary nested NIAM and scientific investigations on fact type orientated information modeling and object-role modeling in general, got a new impulse when Professor Falkenburg was appointed at the University of Nijmegen in 1987. This resulted in a complete formalization of the n-ary nested form of NIAM and extensions thereof with new modeling concepts (literature list [4]).

The first two authors of this book first learned about NIAM in the lectures by professor Falkenburg in the course year 1987/1988. Already in the following year, they taught NIAM in their own university: HAN University of professional education. They then introduced NIAM to their old colleague Harm van der Lek, meanwhile senior consultant at BSO.

In 1989, Nijssen returned to The Netherlands, where he started his own consultancy bureau, next to a part-time professorship. With respect to further developing NIAM, he has since been active refining the 'NIAM-cookbook' and extending NIAM in a broad sense. In this context he introduced the terms *NIAM-ISDM* (*NIAM based Information Systems Development Method*) and *UI* (*Universal Informatics*). In Australia, his former colleague T.A. Halpin continued the scientific side, further standardizing the n-ary nested form of NIAM and renaming it *ORM* (*Object Role Modeling*), which from 1993 on was supported by the CASE-tool Infodesigner, later on called Infomodeler.

In 1989, NIAM was introduced to the Dutch Railways by co-author Van der Lek in his capacity of information architect in a long-term mega project VPT (Vervoer Per Trein, i.e. Transport By Train). An important subproject (the modeling of the infrastructure) that had got stuck was rescued with NIAM. At first N-ary nested modeling was used, whereas a CASE-tool based on binary non-nested NIAM was used to record the over 2500 fact types, and to generate project documentation and data definition instructions for relational databases. In a little subsidiary project to improve the support by CASE-tools, Van der Lek came across some structural inelegancies in both NIAM variants, which were the cause that the transformation of a NIAM information grammar to a relational schema was difficult to implement in a

transparent way. Both NIAM versions, for example, use several essentially different sorts of non-lexical object types (which do or do not arise from nominalizing fact types). He also regarded as a methodical shortcoming the fact that the then available NIAM-tools were not capable of precisely regenerating the verbalizations that the domain experts had given of their facts, which strongly hampered the validation of NIAM information grammars by the domain experts.

FCO-IM (*Fully Communication Oriented Information Modeling*) can best be seen as a descendant in the NIAM family, in which the above named shortcomings have been solved. We originally called it FCO-NIAM, but later on we decided for FCO-IM in order to stress the generic aspects of FCO-IM. Nijssen and others referred to NIAM as being communication oriented, because verbalizations in a natural language serve as a starting point for the modeling process. By adding the word 'fully', we like to emphasize the validation power that was added (i.e. regeneration of the verbalizations given by the domain experts from the resulting information model).

The transformation from an information grammar to a redundancy free relational schema can be simplified considerably by considering information grammars and relational schemas as different manifestations of a single information model that is generic for both FCO-IM and the Relational Model. This makes it possible to transform an information grammar step by step (we call these steps grouping, lexicalization and reducing) until an 'information grammar' is obtained that is the equivalent of a redundancy free relational schema. This GLR algorithm turns out to be relatively simple to implement, and to this day, a first experimental CASE-tool based on this idea is used in the above-mentioned VPT project. FCO-IM, as presented in this book, is the information modeling method based on the above named generic model on the basis of full communication orientation, methodically supported by the NIAM-step by step operational procedure, adjusted where necessary (see appendix A).

Nijssen was involved as external advisor in an Information Systems Development graduation profile, which started in course year 1990-1991 at HAN University. From academic year 1995-1996 on, the program was extended and became a postgraduate course Master of Science in Information Systems Development. Apart from Nijssen, also Van der Lek was an external advisor of HAN University. This led to close cooperation between the authors, on the theoretical, practical and didactical development of FCO-IM. From 1991 on his lead to several publications (literature list [5]), many joint presentations about the fully communication oriented ideas, the development of Dutch course materials: FCO-IM has been taught from course year 1993-1994 on in HAN University. Student-assistants and students in graduating projects built two successive prototypes of FCO-IM supporting CASE-tools.

At the international 1993 NIAM ISDM conference, it was shown in (literature list [6]) that a set-theoretical formalization of FCO-IM needs less fundamental concepts than n-ary nested NIAM does. At the 1994 NIAM ISDM conference, the fully matured FCO concepts were presented as a candidate for a new NIAM standard (literature list [7]), which was positively received. At the conference a third prototype of an FCO-IM supporting CASE tool was demonstrated, which was developed by students of the University of professional education of

Amsterdam in an graduation project under the supervision of the authors. During some years this collaboration was continued after the students had graduated and founded the company Ascaris Software, which realized in 1996 a first professional FCO-IM tool that was straightforwardly called FCO-IM Casetool version 4.0. The 1997 version 4.1 has successfully been used in several big projects in practice. From 2002 on, FCO-IM tool development under supervision of the authors, is done by the Dutch software company Bommeljé Crompvoets and partners (BCP), which provides their FCO-IM tool CaseTalk in a free Book Edition (limited storage capacity), a free Educational Edition (almost full functionality, 6 months expiration time) and a Professional Edition (see: www.CaseTalk.com).

Meanwhile, FCO-IM is being taught in higher education in many universities in The Netherlands and abroad, and FCO-IM is applied in various companies. We name two examples:
The consultancy company TLO Holland Controls from Papendrecht has applied FCO-IM successfully in the ETIM-project, in which an overarching information model was developed that acts as a collective reference model for the branch of industry of electro-technical installation companies.
At the American semi-governmental company Sandia National Laboratories (Albuquerque, New Mexico, USA), which already applied NIAM, an important streamlining of their data dictionaries was realized by basing them on the generic metamodel of FCO-IM.

We close this appendix with a final remark: different cooks working with the same cookbook, in the same kitchen, with the same ingredients, will mostly bring somewhat different dishes on the table, even in a qualitative respect. NIAM information models too, drawn up by different analysts to model the communication about the same Universe of Discourse (or even the same communication about the same UoD), will mostly be somewhat different, even in a qualitative way, no matter how careful the 'NIAM-cookbook' may have been followed. We therefore find the claim made in some NIAM textbooks: 'fully reproducible, even by different analysts', too strong. The NIAM step-by-step operational procedure, when followed precisely, does indeed guarantee redundancy free relational database schemas and this applies in full to FCO-IM as well, but that is not all there is to it: some semantic gaps principally cannot be bridged algorithmically. One of the deepest ravines for analysts turns out to be classification and qualification process. In the training of information analysts therefore, apart from learning how to make concrete examples and verbalizing these, a high priority should be given to the classifying and qualifying process. Fortunately, the FCO-IM tool that goes with this book strongly supports everything else.

Literature List

For an more detailed overview of the history and development of NIAM and FCO-IM, we refer to the following Dutch article:

[1] H. van der Lek, *Natuurlijke taal Informatie Analyse Methode: een overzicht*
Handboek Database Systemen, pages 5143-1-5143-37, Array Publications, Alphen a/d Rijn (1993)

Some books and articles that have inspired us very much (restricted to English language):

[2] J.J.V.R. Wintraecken, *Informatie-analyse volgens NIAM: in theorie en praktijk*
Control Data & Academic Service, Den Haag (1985)
The standard book on binary non-nested NIAM.

[3] G.M. Nijssen, T.A. Halpin, *Conceptual Schema and Relational Database Design: a fact oriented approach*
Prentice-Hall, Sydney (1989), second edition: T.A. Halpin (1995)
The standard book on n-ary nested NIAM (in het meantime called ORM).

[4] A.H.M. ter Hofstede, *Information Modeling in data intensive domains*
Thesis at the Catholic University of Nijmegen, Nijmegen (1993)
Set-theoretical formalization of n-ary nested NIAM, including complex constraints, and incorporating new modeling concepts: generalization, recursive identification, set type and array type; ter Hofstede calls this PSM (the Predicator Set Model).

Articles published by the authors (restricted to English language):

[5] H. van der Lek, G.P. Bakema, J.P.C. Zwart, *The Unification of object types and fact types, a practically and didactically fruitful theory*
Handout at the NIAM-ISDM 1993 Conference, Utrecht, The Netherlands(1993)
Proposals for removing certain inelegancies in n-ary nested NIAM, the generalisation of information models, grouping and lexicalizing and experiences with that in practice and in higher education.

Notes:
In 1991 an earlier Dutch version of this article, with the title *Objecten zijn in feite Feiten* (i.e. Objects are in fact Facts), was published as an internal co-publication of the Dutch system house BSO (now: Atos Origin) and Hogeschool Gelderland (now: HAN University). Later on a Dutch version that is identical to the above-mentioned English version, was published with title: *De unificatie van objecttypen en feittypen, een praktisch en didactisch vruchtbare theorie*
Informatie, volume 34, nr 5, pp. 279-295 (1992)

[6] H. van der Lek, *On the structure of an Information Grammar*
NIAM-ISDM 1993 Conference, Working Papers, 31 pages, Utrecht (1993)
Set-theoretical formalization of FCO-IM.

[7] G.P. Bakema, J.P.C. Zwart, H. van der Lek, *Fully Communication Oriented NIAM*
NIAM-ISDM 1994 Conference, Working Papers, pages L1-L35, Albuquerque, New Mexico (1994)
Also available via the website of the FCO-IM Foundation: www.FCO-IM.com
Overview of FCO-IM (here still called FCO-NIAM) with among other things the incorporation of generalization and recursive identification, a relational form of the kernel of the FCO-IM metagrammar and the GLR algorithm.