Masaryk university
Faculty of Informatics

Conceptual modelling using the **HIT** method

Practical tutorial

Marek Winkler, Michal Oškera, Zdenko Staníček

Version 1.0

Contents

1 Introduction

This introductory section provides a brief explanation of the mutual relationships between data modelling, conceptual modelling and the HIT method.

1.1 Why to deal with conceptual modelling?

For any kind of effective teamwork the mutual understanding of all people involved is essential. This applies to any form the team can take, e.g. SW development team, organizational unit, company staff, or even family. In other words, mutual understanding is the prerequisite for any group of people, that is expected to work together in synergistic way, to succeed. The conceptual modelling discipline is about explicit and systematic facilitation of such *understanding process*. One may ask: how *conceptual modelling* and *understanding process* is related?

The explicit mutual understanding within a group of people is about agreement or conformity in *meaning of words or expressions of language* being used in mutual communication. One of the philosophical and logical approaches[1] at intuitive level explains that *a concept may become meaning of given language expression* [**?**]. In other words, if we know *the meaning of the word 'car'*, we know *the concept of 'CAR'*. Straightforwardly, the word 'car' in spoken or written form *represents* the concept 'CAR' in external world, i.e. we people as lingual creatures use the words 'car', 'auto', 'das Auto' to represent the concept 'CAR' outside of our minds. We do this in order to communicate with other people. But the communication would be useless or even harmful if the words received by our communication partner represented different concepts in their mind. Therefore, the *conceptual modelling* discipline emerged to help people to achieve agreement on key concepts (meanings) represented by words in their mutual communication.

There are endless situations in our lives in which we communicate with other people and in many of them we think that we understand each other, just because we know the meanings of words. But we know only one or few of many possible meanings a single expression may have in minds of other people. Very often, we do not notice or even care of shifts in the meaning and in many ordinary situations we can handle this somehow. Actually, the essence of whole category of jokes lies in the ambiguity of language – these

[1] by coincidence underlying one for the HIT method

are called *puns*[2].

However, there are many situations where one cannot successfully continue in their work just sharing an approximate or even vague meaning. Consider an emergency communication between medicine doctors. They need to be as accurate as possible to avoid misunderstanding that can threaten one's life. Therefore, huge number of concepts (represented by unique language expressions) was revealed, each of them identifying very single piece of human body to enable clear communication.

Another example where accuracy in communication matters are all technical and engineering disciplines. Both complex machines and SW programs cannot be constructed just approximately. They need to be projected in an exact way and they are usually results of team effort. Therefore, people need to clearly communicate about what they are doing. What's more they also need to clearly communicate with people whom they do it for; and this people may be far from *homo technicus*. These people, let us call them users or customers, have their own different expertise and have no clue and also do not care what for instance the term *'event handler'* denotes. They want to be impressed by results not by technical means.

Typically in IT, the *business-IT consultants* are the people who are responsible for translation of business talk (spoken by people with non-IT expertise – *domain experts*) to IT talk and vice versa. This process is called analysis, it is iterative and requires at least good communication skills.

It is clear, that the translation may be successful only after clear understanding of terms being translated. More precisely, consultants need to understand terms such as *'budget'*, *'rate'*, *schedule item* in exactly the same way as the domain experts do – they need to have the same concepts in their minds. Otherwise, software configured or created according to different meanings of the same words will disappoint the customer very soon.

The more complex the domain is (by its scope, specificity, people involved, ...) the more difficult is to understand it and come to an agreement among all parties involved. In these situations the visual, schematic, intuitive but not oversimplifying methods are needed.

The HIT method is one of the kind which may help to accomplish the process of mutual understanding. And in accordance with aforementioned facts, this method must be well understood before it can be applied correctly.

This tutorial is about explanation of concepts behind the *the HIT method*.

[2]Did you hear about the guy whose whole left side was cut off? He's all right now.

Its goal is to reveal and explain key concepts to prospective conceptual modellers and to support continuous understanding within already practising community – everything within the meaning of the preceding paragraphs.

Facts to remember are as follows:

- The primary motivation for *conceptual modelling* is *to understand* and not *to design* or *to specify* a database for instance[3].
- Conceptual modelling is very helpful and essential in *interdisciplinary cooperation* as well as in *team communication*.
- You can do a lot of conceptual models by yourself and for yourself, but the meaningfulness of these models will be checked only when they had been accepted and followed by other people helping them achieve their goals.

1.2 What is conceptual modelling?

More technically, the conceptual modelling can be described as *the process of conceptual understanding and rendering modelled part of reality. The modeller tries to find out which objects of interest the system should keep and provide. Conceptual model should be totally independent of intended implementation and intelligible for users.* [**?**]

In other words, the conceptual model describes the concepts and their relationships identified in the modelled domain. It is important to realize that *a conceptual model's elements are the objects from the modelled domain*, not any database tables, programming language classes or any other artifacts of the system being built for the modelled domain.

A good conceptual modelling technique should make the analyst reduce the amount of the resulting model ambiguity as much as possible. For this reason, the conceptual modelling method HIT is focused on well-formed (semi-formal) definitions of all the concepts and the relationships in the model.

[3]However, the result of conceptual modelling can be very helpful for subsequent database design.

1.3 What is a conceptual model?

A conceptual model is a special case of a data model. As mentioned in the previous section, a conceptual model describes the *concepts* and their mutual *relationships* in the modelled domain.

1.4 Relationship to data modelling

The conceptual modelling is a special case of data modelling. Data modelling is a discipline focusing on the description of *data* using different kinds of *models*. The models, along with the relevant data modelling sub-disciplines, are usually grouped to the following three levels:

- `conceptual models` capture the elements of the modelled domain, they are totally implementation independent; the entities have not been assigned any attributes yet (unless an attribute is really necessary for the description of the modelled domain), the many-to-many relationships are not decomposed into associative entities;
- `logical models` describe the elements of the information system (IS) which is being built for the domain, but they are independent of any particular database or other means used to store IS data; a logical model contains entities which can be specific for the IS implementation, the entities have all their attributes defined, etc.;
- `physical models` define the data structures with all details necessary to create these structures in a particular database engine; in case of a SQL database, this involves assigning an SQL type to each attribute, decomposition of many-to-many relationships into associative entities, defining database indices, etc.

1.5 What is the difference between conceptual modelling and HIT method?

Conceptual modelling is a discipline focused on understanding and description of the concepts and their relationships in the modelled domain. *HIT method* is one particular method or technique used to achieve the goals of conceptual modelling.

1.6 What is the relationship between HIT method, HIT conceptual model and E-R model?

The *HIT method* provides a technique how to construct conceptual models. The result of an application of HIT method is a *HIT conceptual model* which consists of the base of sorts, the so-called kernel set of HIT attributes, and the set of integrity constraints[4].

On the contrary, an *E-R model (Entity-Relationship model)* is not any particular data modelling method, it is merely a standard notation for the conceptual modelling product, i.e. the conceptual model. Therefore, it is comparable to the HIT conceptual model rather than the HIT method itself. In fact, every HIT conceptual model can be transformed into the appropriate E-R model[5].

Another conceptual model notation is a *concept map*. An example can be found at `http://www.w3.org/TR/ws-arch/#gsom`.

1.7 What is the difference between E-R model and E-R diagram?

The *E-R model* consists of the *E-R diagram* and the text description of each *E-R diagram* element.

1.8 Exercises

1. Find a word representing at least two different concepts. How words that possess such property are called?

2. Find two words representing at least the same concept. How words that possess such property are called?

3. Form groups of three. Presuming that all members know the meaning (concept) of word 'component', let each team member independently write his or her own representation of the concept *'COMPONENT'* to the paper. Then compare and discuss the externalized concepts. What do you observe?

[4]More detailed HIT conceptual model definition can be found in the section 2.6 of this document.

[5]See section 2.7 of this document.

2 HIT method – concepts and procedures

This section introduces the HIT conceptual modelling method – the process of creating the HIT conceptual model, HIT method basic concepts, and finally the transformation of the HIT conceptual model into the E-R model notation.

2.1 The steps of the modelling process

Let us start with forming an initial idea what steps the process of making a conceptual model according to the HIT method involves:

- the identification and definition of sorts,
- the identification and definition of HIT attributes,
- normalizing the set of HIT attributes,
- transforming the HIT conceptual model into ER model notation.

Individual steps are illustrated on a concrete domain in section **??**. This section continues with brief introduction of basic ideas behind HIT. You are encouraged to read this section first, but it is possible to skip directly to the example application in section **??** and refer to the rest of this section when needed.

2.2 Functional approach

Everything we encounter in HIT can be regarded as a *function*. In fact, HIT defines all of its concepts as functions. Right now you have probably asked 'Why? The most common approach is the relational calculus which has been widely used in databases nowadays.'. Let us try to shed some light on this matter.

The functional approach (as compared to the relational approach) allows for more natural mapping of the meaning (usually expressed in natural language) of modelled domain elements onto the conceptual model elements definitions. One reason is that a function is oriented while a relation is not.

Imagine a relationship between a student and a seminar. There is no doubt that you can make up various kinds of such relationship – *students*

who have enrolled in the seminar, or *students who have successfully passed the seminar*. If we want to make clear what kind of relationship between a student and a seminar was meant during the making of the model, we have to provide a description telling the model reader what kind of the relationship it is. We say that such a description defines the *semantics* or *meaning* of the relationship.

Consider the semantics of the relationship from the example – *students who have enrolled in the seminar*. This expression can be understood as a description of a function which, being given a seminar, returns the collection of all students who have enrolled in the given seminar. This function can be perceived as a query which a user can resolve with using the knowledge from the modelled domain.

The notion of a query is much more natural to domain experts, expressing themselves in natural language, than relations. Therefore, HIT maps the natural language expressions onto semi-formally defined functions.

HIT distinguishes two kinds of functions: *intensions*, which depend on the current state-of-affairs, and *extensions*, which are the current state-of-affairs independent. But what is the current state-of-affairs?

The current *state-of-affairs* can be understood as a (potentially infinite) *list of facts* which are currently valid. Any *intension* is a function which is able to access such a list of facts (you can imagine it as a method which can query a database table, for instance). They are closely connected with the so-called *Entity sorts* (see later). On the other hand, an *extension* can never access such a list of facts, it can compute only information independent on the current state-of-affairs. *Extensions* are also called *analytical* functions because they cannot access the empirical facts. They are connected with *Descriptive sorts* (see later).

Because this tutorial is meant to serve as a practical guide to HIT method, we are not going to formally define all HIT concepts[6]. Instead, we provide an explanation of the background concepts sufficient for the modelling purposes.

[6]More detailed and rigorous definitions of the HIT method concepts can be found in [**?**] or [**?**].

2.3 Modelling concepts using sorts

In HIT, a *concept* is modelled as a *sort*. You can imagine a *sort* as a set of all objects having a common property distinguishing the sort[7]. An example can be a sort of all individuals who are active students.

HIT distinguishes between *entity* and *descriptive* sorts. An entity sort contains entities and is denoted with the hash character, e.g. **(#Car)**, **(#Student)**. A descriptive sort contains descriptive attributes of entities and is denoted without the hash, e.g. **(Phone number)**.

2.3.1 Entity sorts

An entity sort E is defined using a function which, being applied to an object (or an *individual*) and the current state-of-affairs, returns `true`, if the given object is an instance of the sort being defined, `false` otherwise. This function can be viewed as a *characteristic function* of the sort E. Because this function depends on the current state-of-affairs, it is an *intension*.

In HIT, this function is written as a Natural Language expression following specific pattern. The reason is that providing logically exact definitions built upon some very basic concepts would be so demanding, that such definitions would be practically unusable.

Have a look at one possible definition of **(#Car)**:

> An object of category **(#Car)** is every road vehicle, typically with four wheels, powered by an internal combustion engine and able to carry a small number of people.

This is not the only "right" definition of **(#Car)**, you could invent other ones which would be "right" as well. The goal we are trying to achieve with definitions is, that a definition is comprehensible to all intended model readers and appropriately delimits the concept boundary in the modelled domain. By "delimiting the concept boudary" we mean not only identifying all objects that satisfy the definition, but also denying the objects that we do not consider to be part of the concept.

The generic pattern of the Natural Language expression describing the function is the following:

[7] This definition is very vague, but sufficient for practical usage of HIT. An interested reader should refer to the formal sort definition given in the PB114/PA116 lectures, or in [**?**], [**?**].

An object of category[8] **(#Sort_name)** is every/any ....

An entity sort cannot be directly represented in computers. The representation is usually done by using a corresponding descriptive sort **(ID)**.

2.3.2 Descriptive sorts

A descriptive sort D is defined using a function which, being applied to an object (or an *element*) returns `true`, if the given object is an instance of the sort being defined, `false` otherwise. This function can be viewed as a *characteristic function* of the sort D. Because this function does not depend on the current state-of-affairs, it is an *extension*.

Descriptive sorts can be viewed as the sets of elements which can be directly represented in computers. Consider the following example:

> An element of category **(Phone number)** is every string value having the maximal length of 13, containing only digits and optionally a plus sign as the first character.

The generic pattern of the expression describing the function is the following:

An element of category[9] **(Sort_name)** is every/any ....

2.4 Modelling relationships among concepts using HIT attributes

A relationship among sorts $T_1, \ldots, T_n, S$, where at least one sort $T_i, 1 \leq i \leq n$ is an entity sort, is defined using a function, which being applied to the current state-of-affairs and the given n-tuple $(t_1, \ldots, t_n)$, where $t_i \in T_i, 1 \leq i \leq n$, returns a single $s \in S$, a set $S' \subseteq S$, or is undefined. Such functions are called *HIT attributes* in HIT. Because these functions depend on the current state-of-affairs, they are *intensions*. The number $n + 1$ is called the *complexity* of the HIT attribute.

A HIT attribute can be defined either in *graphical*, or *linear* notation. The linear notation is used in models, while the graphical notation is usually more illustrative for discussions about the decomposability of a HIT attribute (see later).

[8] You can sometimes encounter *type* instead of *category*. *Category* is newer.
[9] You can sometimes encounter *type* instead of *category*. *Category* is newer.

2.4.1 Graphical notation

Graphical notation simply depicts a HIT attribute as a function (remember that every HIT attribute *is* a function) illustrating the mapping of the function's input arguments to the output. The graphical notation consists of the following elements:

- *circles* represent *sorts*; a *crossed circle* denotes an *entity sort*, an empty circle represents a *descriptive sort*,
- the *rectangle*[10] which represents the HIT attribute itself.

Have a look at Figure 1 displaying an example HIT attribute called *Grading*.

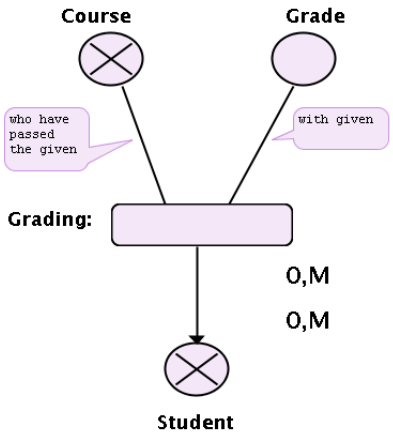

Figure 1: The HIT attribute example.

- The upper part declares the *input arguments* of the HIT attribute. Every input argument is a *sort* accompanied with a compulsory textual description of the semantics (the meaning) of this argument. The semantics of the input argument can be imagined as the definition of the role that the input argument plays in the HIT attribute.
- The lower part declares the *result* of the HIT attribute, optionally accompanied with a textual description of its semantics.

[10] sometimes an *oval* shape is used as well

- The middle part (the oval) represents the *HIT attribute* itself. You can notice the four values on the right side of the HIT attribute which define the *cardinality* of the HIT attribute.

To sum it up, Figure 2 displays the generic form of a HIT attribute in graphical notation. The `card_orig` is the cardinality of the function described by the HIT attribute. The `card_rev` is the cardinality of the *reversed function*. See the last paragraph of this section for the description of cardinalities.

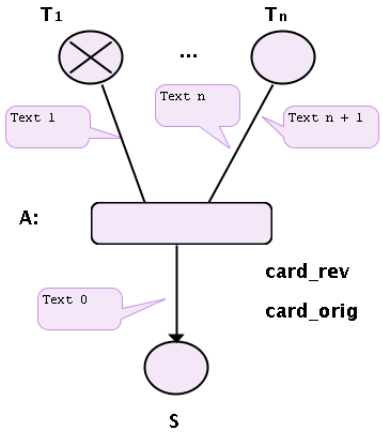

Figure 2: The generic form of HIT attribute graphical notation.

The meaning of the HIT attribute can be determined by reading the diagram starting with the result sort and then reading the description of each input argument semantics followed by the name of the sort:

Students who have passed the given course with given grade.

In fact, this procedure gives us (almost) the *linear notation* mentioned before.

2.4.2 Linear notation

Linear notation of a HIT attribute is constructed using the following rule:

- if the HIT attribute returns at most one result, then the linear notation follows this pattern:
  $Text_0$ (S) $Text_1$ (T_1) $\ldots Text_n$ (T_n) $Text_{n+1}$ / `card_orig`:`card_rev`

- if the HIT attribute can return more than one result for given input, then the linear notation follows this pattern:
  $Text_0\ (S) {-} s\ Text_1\ (T_1) \ldots Text_n\ (T_n)\ Text_{n+1}$ / `card_orig:card_rev`

Using again the HIT attribute *Grading* as an example, the linear notation is the following:

> Students **(#Student)**-s who have passed the given course **(#Course)** with given grade **(Grade)** / 0,M:0,M

There are a few rules that must be respected when constructing a linear notation:

- every input argument is explicitely marked with the word *given*,
- the result of the HIT attribute must be stated in *singular* or *plural* form according to the following rules:
  - *singular* – when the function can return at most one instance of the result *sort*,
  - *plural* – when the function can return a set of instances of the result *sort*.

  The plural form must also be marked by adding the suffix "-s" to the name of result sort (see the example).

Let us have a look at the same example HIT attribute linear form with the parts affected by the aforementioned rules underlined:

> Students **(#Student)**-s who have passed the given course **(#Course)** with given grade **(Grade)** / 0,M:0,M

To highlight the differences between singular and plural forms, imagine a singular form of the same HIT attribute:

> Student **(#Student)** who has passed the given course **(#Course)** with given grade **(Grade)** / 0,1:0,M

Both graphical and linear notations are equivalent in terms that they carry the same amount of information and one can be transformed into the other without requiring any additional information.

2.4.3 HIT attribute rotation

A HIT attribute rotation is every HIT attribute that is constructed from the original HIT attribute by swapping the result sort with one of the input argument sorts. HIT attributes that differ only in the order of their input arguments are considered equal, and therefore they are not different rotations.

The following example illustrates all *admissible* rotations of the HIT attribute *Grading*:

- Students **(#Student)**-s who have passed the given course **(#Course)** with given grade **(Grade)** / 0,M:0,M
- Grade **(Grade)** which has been achieved by the given **(#Student)** in passing the given course **(#Course)** / 0,1:0,M
- Courses **(#Course)**-s which have been passed by the given student **(#Student)** with given grade **(Grade)** / 0,M:0,M

Each rotation can be either singular or plural, thus the list of rotations could be extended with rotations such as

> Student **(#Student)** who has passed the given course **(#Course)** with given grade **(Grade)** / 0,1:0,M.

However, such a rotation does not describe the modelled domain appropriately – clearly, more than one student can pass a given course with a given grade. Such rotations are called *inadmissible*. During the modelling we concentrate on the *admissible* rotations only.

2.4.4 Cardinality

The cardinality of a relationship A provides information about the minimum and maximum number of objects that can be involved in a single instance of the relationship A. But how to interpret the cardinality in the context of a HIT attribute, i.e. a function?

Consider our example HIT attribute *Grading*:

> Students **(#Student)**-s who have passed the given course **(#Course)** with given grade **(Grade)** / 0,M:0,M

The cardinality information, located after the slash ("/") character, consists of two pairs of characters. The first pair of characters is the cardinality of the function directly described by the HIT attribute *Grading*:

- the first element indicates if the function is partial or total; informally, how many results *at least* the function returns for an *arbitrarily chosen* input argument. The possible values are *0* (partial) and *1* (total).

  In the example, (0,M:0,M) states that the function is partial, i.e. it can be undefined for some input arguments. Because it is possible that no student passed the given course with the given grade, the function for such input arguments returns nothing (it is undefined).

- the second element defines if the function returns a single result or a set; in other words, how many results *at most* the function returns for an *arbitrarily chosen* input argument. The possible values are *1* (single result) and *M* (many results)[11].

  In our example, (0,M:0,M) indicates that the function returns a set of students who have passed the given course with the given grade. Indeed, several students might have achieved the same grade for the same course.

The second pair of characters (in the example (0,M:0,M)) defines cardinality of the so-called *reversed function*. The reversed function can be obtained by swapping the input and output arguments of the function[12]:

> The pairs (**(#Course)**, **(Grade)**)-s which correspond to the passing of this course with this grade by the given **(#Student)**.

The cardinality is again 0,M: there can be zero or many such pairs returned by the function for an *arbitrarily chosen* student.

2.5 Normalization of HIT attributes

Some of the identified HIT attributes can be in fact a composition of more basic HIT attributes. Using a composed HIT attribute in the model is a problem, because it results in the loss of information. Why?

Consider the following HIT attribute called *Enrolls*:

[11] Sometimes, the value of *N* can be encountered as well – it has the same meaning as *M*.

[12] It is similar, but not completely identical, to the *inverse function* in mathematics.

> *Enrolls*: Students **(#Student)**-s who have enrolled in a given seminar **(#Seminar)** taught by a given lecturer **(#Lecturer)** / 0,M:0,M

It is tempting to believe that you can obtain the information which seminars are taught by a given lecturer from this HIT attribute as well. However, this is not completely true – what if there is a seminar taught by the given lecturer but no student has enrolled in this seminar yet? This information cannot be captured by the HIT attribute above.

This step of the modelling process aims at removing such compositions and redundancies.

2.5.1 Definability

Let us begin with the HIT concept which allows to describe what it means when we consider a HIT attribute redundant.

A HIT attribute A is said to be *definable* over the set of HIT attributes $\{B_1, \ldots, B_n\}$ when there is an *analytical* function that is able to compute all values of A by using the HIT attributes $B_1, \ldots, B_n$.[13] It is denoted as $A \leftarrow \{B_1, \ldots, B_n\}$.

What is an *analytical* function? A function is *analytical*, if it does not depend on any *empirical* knowledge, i.e. on the current *state-of-affairs*[14]. In programming terms, you can imagine an analytical function as a procedure which does not depend on any list of facts (for instance a database table of current lecturers) about the "outside" world for its computation.

The fact that the function has to be *analytical* ensures that the function itself is not another "hidden" HIT attribute bringing some additional "unspotted" information into the model.

Let us make it clear on the example of HIT attribute from the introduction to this section:

> *Enrolls*: Students **(#Student)**-s who have enrolled in a given seminar **(#Seminar)** taught by a given lecturer **(#Lecturer)** / 0,M:0,M

Now, imagine two simpler HIT attributes *Teaches* and *Enrolls'*:

[13] For precise formal definition please see the PB114/PA116 lectures or the literature.
[14] see Section 2.2

Teaches: Lecturer **(#Lecturer)** teaching a given seminar **(#Seminar)** / 0,1:0,M

Enrolls': Students **(#Student)**-s who have enrolled in a given seminar **(#Seminar)** / 0,M:0,M

We can simply compute all values of the original HIT attribute *Enrolls* using only the HIT attributes *Teaches* and *Enrolls'* and some *analytical* operations. The computation procedure indeed can be constructed as *analytical* because all necessary empirical information is provided by the HIT attributes *Teaches* and *Enrolls'* which you can imagine as other procedures having access to the appropriate data store.

Therefore, we can say that $Enrolls \leftarrow \{Teaches, Enrolls'\}$.

2.5.2 Decomposability

HIT attribute decomposition essentially means reducing the original HIT attribute to the set of its subattributes without losing any information. What does it mean?

First, what is a subattribute? A subattribute A' of HIT attribute A is every HIT attribute constructed from A by omitting one of the *sorts* involved in A.[15]

We can see that the HIT attributes *Teaches* and *Enrolls'* are subattributes of the HIT attribute *Enrolls*.

Next, we need to make sure that we do not lose any information if we replace the original HIT attribute *Enrolls* with its subattributes. This follows from the definability of *Enrolls* over the set $\{Teaches, Enrolls'\}$. Why is this true? Because we can create an *analytical* function which will compute all values of the original HIT attribute by using the subattributes.

You have probably realized by now that *decomposability* is closely related to *definability* which is reflected in the definition of decomposability itself:

A HIT attribute A is said to be decomposable into the set of HIT attributes $\{B_1, \ldots, B_n\}$, when the following conditions hold:

1. every $B_i, 1 \leq i \leq n$ is a subattribute of A,

[15] This definition of *subattribute* corresponds to the notion of *proper subattribute* defined in the lectures. According to the HIT method, the definition of *subattribute* allows A to be its own *subattribute*. For simplicity, we do not consider this possibility.

2. $A \leftarrow \{B_1, \ldots, B_n\}$.

It is denoted as $A \Diamond \{B_1, \ldots, B_n\}$.

Therefore, we can say that $Enrolls \Diamond \{Teaches, Enrolls'\}$.

2.5.3 The kernel of a set of HIT attributes

The kernel of a set of HIT attributes can be imagined as the *normalized* variant of the original set. Thus the whole *normalization* step of the modelling process can be rephrased as finding the kernel set of the initially identified HIT attributes. This kernel set will be the part of the final conceptual model.

Let K and A be sets of HIT attributes. The set K is called the *kernel* of the set A, iff the following conditions hold:

1. K and A are *informationally equivalent*; this means that every HIT attribute from K is *definable* over A and every HIT attribute from A is *definable* over K,

2. all HIT attributes in K are elementary; this means no HIT attribute from K can be decomposed,

3. K does not contain any redundant HIT attributes; this means that there is no HIT attribute $A' \in K$ such that $K \leftarrow K - \{A'\}$.

The above conditions can be rephrased as follows: the kernel is every minimal set of elementary HIT attributes defining exactly the same information as the original HIT attributes set. The properties following from the three conditions given in the definition are underlined.

Unfortunately, no algorithm which would be capable of finding the kernel is known. Determining the kernel also requires domain knowledge to decide definability and decomposition of HIT attributes.

2.6 HIT conceptual model

Having found the kernel set of HIT attributes, we have constructed the *HIT conceptual model*. It is defined as the triple (B, K, C), where:

- B is the base of *sorts* that we have identified in the domain,

- K is the *kernel* of the set of relevant HIT attributes identified in the domain,
- C is the set of *integrity constraints* formulated over HIT attributes from K.

Note that the model does not include any diagram. The next section describes the procedure how to create an ER model (including an ER diagram) according to the HIT conceptual model.

2.7 Transformation into ER model

The HIT conceptual model is transformed into the ER model notation mainly for the following reasons:

- ER model includes ER diagram which is very useful tool for getting quick awareness of the model,
- ER model is a well-known modelling artifact,
- ER model is often used at later phases of software design to accomodate logical and physical data models.

Provided with the kernel set of HIT attributes, we are almost ready to transform the HIT conceptual model into the ER model. All HIT attributes are now either *binary* (of complexity 2), or cannot be further decomposed. All HIT attributes add some information to the model, none of them can be omitted.

The transformation task is to map the *sorts* and *HIT attributes* onto a graph structure of *nodes* (*entities* in ER model) and their *connections* (*relationships* in ER model). The first intuition is straightforward: the entity sorts are mapped onto entities and the HIT attributes are mapped onto relationships[16].

But plain relationships can connect only two entities. This way we can map only binary HIT attributes. The non-binary HIT attributes cannot be decomposed because the HIT conceptual model includes only HIT attributes from the kernel. Fortunately, the *binarization principle* deals with them.

[16]The transformation of descriptive sorts and descriptive HIT attributes is omitted in this first intuition – see section 2.7.2.

2.7.1 Binarization principle

The binarization principle transforms a non-decomposable HIT attribute with complexity greater than 2 into one new created sort (called *concatenated type*) and a set of the so-called *projection* HIT attributes.

Let us illustrate the binarization principle on our example HIT attribute *Grading* shown in Figure 3:

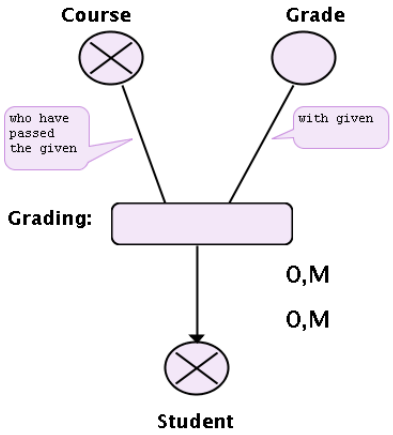

Figure 3: The non-decomposable HIT attribute *Grading*.

First, the binarization principle instructs us to create a new sort[17] called *concatenated type*. It contains *n-tuples* corresponding to the original HIT attribute; using the *Grading* HIT attribute, the concatenated type might contain the following n-tuples:

$$(student\_A, PB114, A),$$
$$(student\_B, PB114, C),$$
$$\ldots$$

where:

- $student\_A$ and $student\_B \in$ **(#Student)**,
- $PB114 \in$ **(#Course)**,

[17] More precisely a *type* rather than a *sort* – see lectures or references for the type theory used in HIT.

- A and $C \in$ **(Grade)**.

The concatenated type represents the original HIT attribute, therefore the definition of concatenated type contains the semantics of original HIT attribute in linear notation along with its cardinality.

In the example, the definition of concatenated type *Grading* looks like:

> An object of category **(#Grading)** is every representation of a relationship between **(#Student)** and **(#Course)** with the following meaning:
> Students **(#Student)**-s who have passed the given course **(#Course)** with given grade **(Grade)** / 0,M:0,M

Now we have to map the n-tuples from the concatenated type onto the referenced instances of sorts **(#Student)**, **(#Course)** and **(Grade)**. This mapping is done by the so-called *projection* HIT attributes depicted in Figure 4.

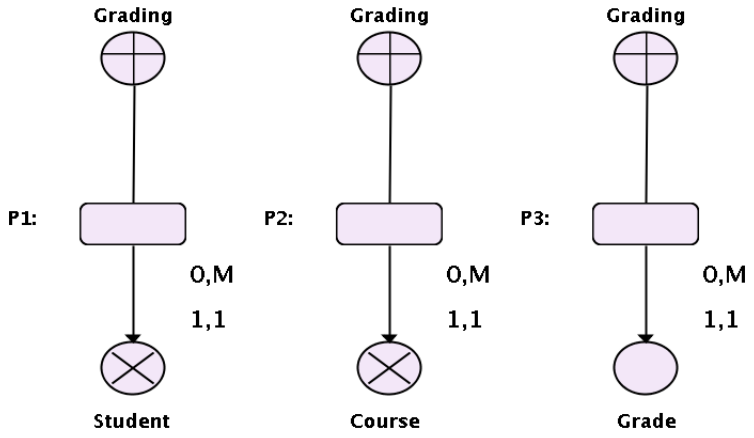

Figure 4: The projections of the concatenated type onto the components of the original HIT attribute.

Notice that there is no textual description of the semantics of the HIT attributes – it seems as an exception to the rule that every HIT attribute has to have its semantics defined. In fact, the semantics of projection HIT attributes is always the same – the projection of the concatenated type to

one of its elements. The textual description could be for example "of given" but is usually omitted.

Another consequence of the same semantics of all projection HIT attributes is *always the same cardinality* of all projection attributes.

This section concludes with the illustration of binarization principle for general HIT attributes – see Figures 5 and 6. The consequence of the existence of the binarization principle is the ability to transform any HIT conceptual model into a graph structure of nodes and edges which is required by the transformation to ER model.

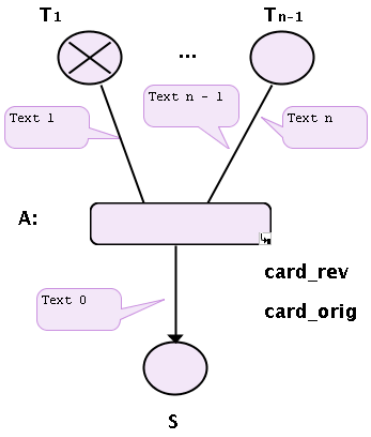

Figure 5: The non-decomposable HIT attribute with complexity > 2.

2.7.2 The transformation procedure

The process of transformation of the HIT conceptual model[18] $M = (B, K, C)$ into ER model consists of the following steps[19]:

- Derive entities:

    1. For every HIT attribute $A \in K$ having the complexity > 2 apply the binarization principle, i.e. introduce new concatenated type R and replace A with the set of projection HIT attributes B_i.

[18] See section 2.6.

[19] The precise algorithm can be found in the PB114/PA116 lectures.

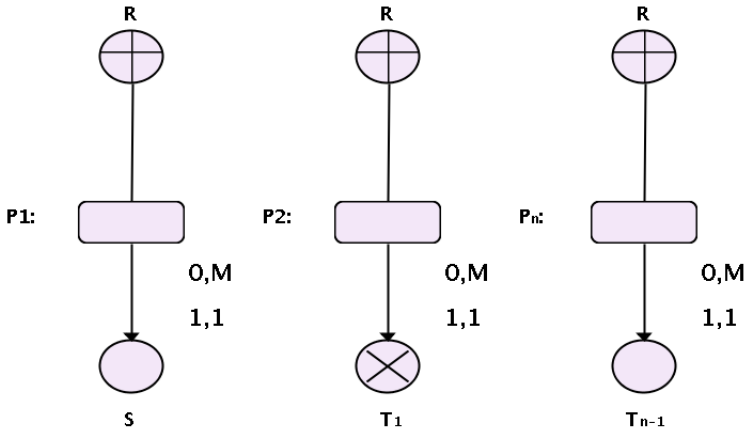

Figure 6: The resulting set of projections of the concatenated type onto the components of the original HIT attribute.

- every HIT attribute in K is *binary* (its complexity = 2) now
- binarization principle does not operate with constraints, so they need to be updated

2. Replace every constraint $c \in C$ valid for A with corresponding constraint(s) for B_i.
   - the constraints match the results of binarization principle application
3. Represent every entity sort as a *kernel entity* in ER model.
4. Represent every concatenated type as an *associative entity* in ER model.
5. For every *descriptive* HIT attribute $D \in K$ with a *plural* rotation having a *descriptive sort* S as the result[20]:
   - represent *descriptive sort* S as a *characteristic entity* in ER model,
   - represent the HIT attribute D as a relationship in ER model. The cardinality of the relationship is derived from the cardinality of the HIT attribute.

[20] In other words, the cardinality of this rotation is ?,M:?,?.

- Derive relationships:

    1. represent every HIT attribute in K which does not operate with any *descriptive sort* as a *relationship* in ER model. The cardinality of the relationship is derived from the cardinality of the HIT attribute.

- Derive attributes (only when transforming into $ERAM$[21]):

    1. Represent every *descriptive* HIT attribute $D \in K$, with a *singular* rotation having a *descriptive sort* S as the result[22], as an $ERAM$ attribute of the corresponding kernel or associative entity.

- Provide definitions:

    1. Write definitions of all kernel, associative and characteristic entities by using definitions of sorts and concatenated types from B.
    2. Write definitions of all relationships by using linear notation of HIT attributes from K.

The complete transformation procedure described in PB114/PA116 lectures deals with the supertype-subtype hierarchy which is not covered by this tutorial. The reason is the non-trivial amount of required theoretical background, while the effect for the practical utilization of HIT would not be as significant.

Let us illustrate the transformation procedure on a simple example. Imagine the following HIT conceptual model:
The sorts:

> An object of category **(#Lecturer)** is every person contracted by the university as a teacher of a course.
>
> An object of category **(#Course)** is every series of lectures or lessons in a particular subject offered by the university.

[21] The transformation procedure can be used for transformation into Entity-Relationship-Attribute Model. This model type is often used for logical or physical data models.

[22] In other words, the cardinality of this rotation is ?,1:?,?.

An object of category **(#Student)** is every person currently enrolled in a university study programme.

An element of category **(Grade)** is every character from the following list: A, B, C, D, E, F.

The kernel set of HIT attributes:

Teaches: Lecturer **(#Lecturer)** teaching a given course **(#Course)** / 0,1:0,M

Enrolls: Students **(#Student)**-s who have enrolled in a given course **(#Course)** / 0,M:0,M

Grading: Students **(#Student)**-s who have passed the given course **(#Course)** with given grade **(Grade)** / 0,M:0,M

The set of integrity constraints is empty.

First, we apply the binarization principle to the *Grading* HIT attribute because it is the only HIT attribute with complexity greater than 2. Next, we transform the entity sorts into ER model entities, and the HIT attributes into the corresponding relationships.

The final model consists of the ER diagram (see Figure 7) and the following textual description. Notice that the projection HIT attributes (obtained from the binarization principle application) are not part of the textual description:

Entities:
Name: Lecturer
Type: kernel

An object of category **(#Lecturer)** is every person contracted by some faculty department to give lectures to students.

Name: Course
Type: kernel

An object of category **(#Course)** is every series of lectures or lessons on particular subject approved by the faculty scientific committee for being taught at the faculty.

Name: Student
Type: kernel

> An object of category **(#Student)** is every person signed for study in a bachelor's, master's or doctoral study programme. The study must not be currently interrupted.

Name: Grading
Type: associative

> An object of category **(#Grading)** is every representation of a relationship between **(#Student)** and **(#Course)** with the following meaning:
> Students **(#Student)**-s who have passed the given course **(#Course)** with given grade **(Grade)** / 0,M:0,M

Relationships:

> *1*: Lecturer **(#Lecturer)** teaching a given course **(#Course)** / 0,1:0,M
>
> *2*: Students **(#Student)**-s who have enrolled in a given course **(#Course)** / 0,M:0,M

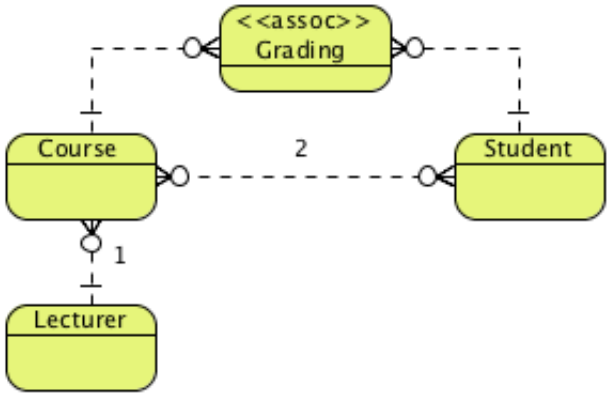

Figure 7: The ER diagram part of the resulting ER model.

The description of the transformation procedure concludes the first part of the tutorial. The next section illustrates the application of HIT method to the construction of conceptual model of the domain of a *restaurant*.