13

Domain Analysis and Concept Formation

- The **prerequisite** for studying this chapter is that you have studied the chapters on stakeholders, domain attributes, domain facets and domain acquisition.
- The **aims** are to introduce ideas of concept formation, and to introduce concepts of domain consistency, completeness and conflict.
- The **objective** is to bring you further along the road to becoming a professional software engineer who is versatile in domain engineering.
- The **treatment** is informal, yet systematic.

13.1 Introduction

Characterisation. By *domain analysis* we mean a reading of rough domain acquisition description units, with the aim of forming concepts from these, as well as with the aim of discovering inconsistencies, conflicts and incompletenesses within these domain description units. ∎

The title of the chapter says Domain Analysis and Concept Formation. The above characterisation of the domain analysis mentions both concept formation and inconsistencies, conflicts and incompletenesses. But the latter concepts were not mentioned in the title of the chapter. So what do we mean? We mean that we must get rid of negatives: inconsistencies, conflicts and incompletenesses, while achieving the positive: concept formation. Hence we emphasise the positive in the title. But treat both, in the order, first concept formation, then inconsistencies, conflicts and incompletenesses.

13.2 Concept Formation

We see a spectrum of concepts emerging from proper concept formation. We see this from the simply abstracted concepts, treated in the first section below,

to the cleverly abstracted, "discovery" or breakthrough concepts mentioned further on.

13.2.1 Simply Abstracted Concepts

We exemplify the formation of some concept as based on given description units. So first we give some description units. On the basis of these we can then, in a following example, illustrate the formation of concepts.

Example 13.1 *Some Logistics System Domain Description Units:* We bring some (nonindexed) domain description units gathered when examining a freight logistics domain:

- *Freight can be delivered to trucking depots, railway freight terminals, harbours and air cargo centres.*
- *Freight can be transported by trucks, freight trains, ships* and *Trucks move along highways, trains along railway lines, ships along sea lanes and aircraft along air corridors.*
- *Freight may be transferred between trucks, trains, ships and aircraft at trucking depots, railway freight terminals, harbours, and air cargo centres, and between trucks, trains, ships and aircraft at such places where trucking deports coincide with railway freight stations, harbours and air cargo centres, etc.*
- *Freight can be picked up from trucking depots, railway freight terminals, harbours and air cargo centres.*

The above description units are construed, i.e., made to order. ∎

Characterisation. *Domain Concept Formation:* By *domain concept formation* we understand the abstraction of domain phenomena into concepts. ∎

Whereas description units may refer to categories of immediately instantiable, that is, designatable (phenomenological), entities, concepts usually refer to classes of such categories.

Example 13.2 *A Logistics System Analysis and Concept Formation:* We bring in first some analysis and then we form concepts (from the above exemplified description units).

- *Trucking depots, railway freight terminals, harbours and air cargo centres are all of the same kind: They (temporarily) store (delivered or transferred) freight for further transport or delivery. Hence we shall abstract them into one class: hubs.*
- *Trucks, trains, ships and aircraft are all of the same kind: They move freight. Hence we shall abstract them into one class: conveyor vehicles.*

- *Highways, railway lines, sea lanes and air corridors are all of the same kind: They are "paths" along which freight can be conveyed. Hence we shall abstract them into one class: routes.*

Now you see how our "made to order" examples of Example 13.1 are being disposed of in this example. ▪

Concept formation usually leads to much simpler descriptions.

Example 13.3 *A Partial Logistics System Narrative:*

- *Freight can be delivered to hubs.*
- *Freight can be transported by conveyor vehicles.*
- *Trucks move along routes.*
- *Freight may be transferred from one conveyor vehicle to another conveyor vehicle at hubs.*
- *Freight can be picked up at hubs.*

Our simple sequence of three sets of construed examples, Examples 13.1–13.3, has been brought to an end. ▪

Concept formation is for the experienced engineer, and it is an art — but some principles and techniques can be taught and learned. Principles and techniques of abstraction apply here — to their fullest — and form the major principles and techniques used in concept formation. As we have covered these principles and techniques of abstraction elsewhere[1] we shall not cover abstraction much here. But we remind the reader that in reading domain descriptions we shall seek out texts wherever a phenomenon [or a concept] can be abstracted (into a [further abstracted] concept), or two or more phenomena or concepts can be "merged", i.e., abstracted into one concept.

Each abstracted concept needs to be carefully defined. Concept formation (i.e., identification) then necessitates a rewriting of domain descriptive texts into such where the concepts replace that which they abstract, as well as the insertion of the defined concepts into a terminology.

13.2.2 Breakthrough Abstracted Concepts

The trouble, in a sense, with the "simply abstracted" concepts is that they really do not introduce anything new! Well, why must things be new? Well, sometimes "breakthrough" concepts may turn our world upside down and simplify matters considerably.

It is obvious that we need some informal examples here.

[1] We cover abstraction principles, techniques and tools in Vols. 1 and 2 of this series of textsbooks on software engineering.

Example 13.4 *Breakthrough Concepts:*

Consider the year 1925, or so. Flying aircraft over or near the Magnetic North Pole (currently, 2006, at 82.7 latitude °north and 114.4 longitude °west) was not possible. If one had asked scientists to devise a gadget to help on this matter they might conceivably well have invented a very involved and tricky sextant to have the aircraft guided by the stars. Instead the inertial guidance concept based on gyroscopes, solved the, problem. That is, an abstraction from conventional physical astronomical gadgets had replaced these by, in a sense, other physical gadgets which one would commonly not associate with astronomy.

Consider the year 1880. Performing surgery on the brain — to see what a problem was and, if necessary, to remove spots of cancer — was not possible. But prospects were looming. Things improved so much that a call for designing suitable instruments for such surgery might have led to very intricate mechanical and manually operated minirobots of saws, scissors and scalpels. Instead, some time later X-ray techniques replaced "all that". That is, the phenomenon of X-rays replaced a phenomenon of a mechanical minirobot of saws, scissors and scalpels. An abstraction from having to open up the skull and instead penetrating it invisibly had taken place.

From our own universe of discourse: Computing and, in particular, software, let us consider function abstraction: that of lifting a function from A into B to become a temporal function from Time into functions from A into B. Or, making the data type X of Y elements into an abstract, parameterised data type X(E) of instantiable E elements, one instance being, for example, Y, another being Z. The idea is also shown in Sect. 11.7 on Scripts, where we can factor out the ready-made, "baked into the banking software system", banking transactions into programs of a separately "programmable" bank script language, which could, perhaps, qualify as a discovery. ∎

But, in fact, we have something far more "revolutionary" in mind when we refer to concepts as possibly being innovative discoveries, i.e., breakthroughs. These types of things are not done, i.e., made, really, in the daily life of a software engineer. We cannot plan for them. We can, however, plan for and expect "delivery" of simply abstracted concepts. The breakthroughs one can hope for. They become corporate treasures, which may not be tradeable, but are of advantage in competitive situations.

13.3 Consistencies, Conflicts and Completeness

Conflicts are one form of inconsistency. "Remaining" inconsistencies can be resolved between domain stakeholders — as aided by the domain engineers. Completeness is a relative issue. We shall cover these three ideas now.

13.3.1 Inconsistencies

Characterisation. By *inconsistency* of a *domain description* we shall understand some pairs (or more) of texts where one text describes one (set of) property (properties), while another text (of the pair or more) describes (describe) an "opposite" property (set of properties), that is, property P and property $\mathsf{not}\,P$. ∎

Example 13.5 *A Domain Description Inconsistency:* The following two non-indexed description units express an inconsistency: *Between 5 am and 2 pm trains run at 12-minute intervals;* and *Between 11 am and 7 pm trains run at 15-minute intervals.* ∎

The P in the above example is *Between 11 am and 2 pm trains run at 12-minute intervals;* and the $\mathsf{not}\,P$ in above example is *Between 11 am and 2 pm trains do not run at 12-minute intervals.*

Current research, basically with a base in requirements engineering, investigates (and the same researchers also propose engineering solutions to) problems of more or less automatically detecting inconsistencies in sets of domain description (or, rather, requirements prescription) units. Such proposals imply the representation of the de/prescription units in logical form, and the proposals then focus on the particulars of such forms and on what tool support can then be mechanised. We thus alert the reader to look out for viable tools of this type, if and when they become commercially available and supported.

13.3.2 Conflicts

Characterisation. By a *domain description conflict* we shall understand a domain description inconsistency, in which some domain stakeholders strongly adhere to one set of domain descriptions, which are inconsistent with another set of domain descriptions, strongly adhered to by other domain stakeholders — and such that this conflict can only be resolved through negotiation, including possible business process reengineering, between up to several levels of management. ∎

Inconsistencies which can be resolved (i.e., removed) by discussion between domain stakeholders are not conflicts. Conflicts are usually deeply rooted, and are not within the prerogative of the domain engineers to solve.

13.3.3 Incompleteness

Characterisation. By *incompleteness* of a *domain description* we shall understand a description which leaves open some of the values of some entities, some of the function argument/result value relations or some of the process behaviours; or which indicates some alternative possibilities of entity attributes,

function argument/result pairs or behaviour alternatives, without indicating (i.e., describing) all (obvious) such. ∎

Example 13.6 *A Domain Description Incompleteness:* We give a few examples:

Adult passengers pay full fare, children half. An incompleteness could be that there are no domain descriptions for what is meant by passengers and hence by adult and child passengers, or that, say, soldier, student and pensioner transport fees are not covered.

A free rail unit is one which is available for scheduling. A locked train unit is one which has been scheduled but is as yet not occupied by a train. An occupied train unit has a train passing it. An incompleteness could be that there are no descriptions of, say, rail units that have just been passed by a train but are possibly not yet (made) available for scheduling. ∎

Incomplete domain descriptions may be made complete, or left as such. Final incomplete domain descriptions may be a source for expressing domain requirements that "repair" any conceived incompletenesses.

13.3.4 Looseness and Nondeterminism

Characterisation. By a *loose domain description* we shall understand the same as an incomplete domain description, but one in which that looseness has been left so on purpose. ∎

Characterisation. By a *nondeterministic domain description* we shall understand a description which deliberately leaves open such things as function argument/result values, choice amongst alternative behaviours, ordering of events, etc. ∎

Example 13.7 *Nondeterministic Domain Descriptions:* We give some examples:

Persons are not further defined.

Function f when applied to any integer value yields a value 3 or more.

Users issue series of inquiry and order commands in any order. ∎

13.4 From Analysis to Synthesis

Once a reasonably thorough analysis and concept formation has been done, the domain engineer can create a domain description, i.e., a domain model, according to the principles and techniques outlined in earlier sections. In this part of the book, which consists of several chapters on domain engineering, we

have covered principles and techniques of the major issues of domain descriptions, i.e., synthesis, before we covered analysis in this chapter. The reason, to repeat, for covering principles and techniques of some development stages in the reverse order of their actual sequence in domain engineering is that we must first know what should go into, i.e., what can constitute, a domain description before we cover the principles of domain acquisition. And analysis follows acquisition.

13.5 Discussion

13.5.1 General

This chapter is closely related to Chap. 21: Requirements Analysis and Concept Formation. The reader is advised to carefully review this material before reading Chap. 21. After having studied Chap. 21, the reader is encouraged to compare the two chapters: 13 and 21.

Which tool support is available for discovering inconsistencies and incompleteness? To answer that question in the positive, we need to reflect a bit on the form of our description units: If they are formulated in an informal language, a notation without any formal semantics or proof system, then there is only the human brain and informal, but precise reasoning to resort to. If they are formulated in a formal language, a notation with a formal semantics, or a proof system, then model checking and theorem proving may be attempted. We shall, in the present edition of these volumes, not venture further into this important area, other than saying that there do indeed exist tools and techniques for assisting in the discovery of domain inconsistencies and incompleteness.

13.5.2 Principles, Techniques and Tools

We summarise:

Principles. The principle of *domain analysis* applies to domain descriptions and shall ensure that descriptions are consistent, relatively complete and based on the "narrow bridge" principle of pleasing sets of a relatively small set of designations and defined concepts. ∎

Principles. The principle of *concept formation* applies to domain descriptions and shall ensure that descriptions are based on "telling", i.e., pleasing concepts. ∎

Techniques. When applied to informal domain descriptions, including informal description units, the *domain analysis* techniques are likewise informal. When applied to formalised domain descriptions, including informal description units, the preferred techniques focus on 'abstract interpretation': from

static data and control flow analysis, to symbolic "execution" of the formal descriptions texts, followed by human interpretation of the results. ∎

Techniques. The techniques of discovering concepts are, naturally, those of exploration and experimentation, of scepticism, and of conjecturing and refuting. These investigative techniques again require further techniques, usually those of an inquisitive, scientific mind. ∎

The reader will have discerned, in the characterisation of concept formation, that (human) ingenuity is called for. So be it.

Tools. When *domain analysis* is applied to informal domain descriptions, including informal description units, the tools are likewise informal: human brainpower, and good text-processing facilities. When domain analysis is applied to formalised domain descriptions, including informal description units, the tools include those that can perform abstract interpretation, i.e., flow analysis on formal texts. ∎

Tools. The only *concept formation* tool "presently available" is that of human brainpower. ∎

13.6 Bibliographical Notes

The following researchers — who have contributed significantly to the field of requirements engineering — are mentioned in this chapter due to the tool-oriented nature of their work and its relevance also to domain engineering: John Mylopoulos, A. Borgida, L. Chung, S.J. Greenspan, and E. Yu: [126, 127, 253–255, 380]; Joseph A. Goguen, M. Girotka and C. Linde: [123, 124]; and Axel van Lamsweerde, A. Dardenne, R. Darimont, M. Feather, S. Fikas, R. De Landtsheer E. Letier, C. Ponsard and L. Willemet: [76–78, 102, 211, 214, 215, 360–366].

13.7 Exercises

13.7.1 A Preamble

We refer to Sect. 1.7.1 for the list of 15 running domain (requirements and software design) examples; and we refer to the introductory remarks of Sect. 1.7.2 concerning the use of the term "selected topic".

13.7.2 The Exercises

Exercise 13.1 *Domain Inconsistencies.* For the fixed topic, selected by you, try to create 3 or 4 examples of inconsistent domain description units.

Exercise 13.2 *Domain Conflicts.* For the fixed topic, selected by you, try to create, say, two examples of conflicting domain description units.

Exercise 13.3 *Domain Concepts.* For the fixed topic, selected by you, try to create a number, say, four or more domain description units from which, by simple analysis, you can come up with at least two concepts.