

# Teaching formal specifications

## What about abstraction ?

Christine Choppy

LIPN, Institut Galilée - Université Paris XIII, France

**Abstract.** One of the issues in teaching formal specifications is abstraction. When confronted with the same exercise or case study specification, some students will address the abstract problem, while others will struggle with the concrete one (which is usually much more - and unnecessarily - complex). To which extent and how can we help and guide students to the abstract way ? Some experience in teaching algebraic specifications and Petri nets is related here.

**Keywords:** specification method, modelling method, teaching formal methods, patterns, algebraic specifications, (coloured) Petri nets.

**Topics:** Experience from teaching formal methods, Specification and modelling techniques

## 1 Introduction

Abstraction is one of the major issues in human thinking as studied by psychologists [19]. It is stated that human beings are able to think abstractly, and to use symbols related to abstract concepts, now how does this happen ?

What is unconsciously involved is described as follows [10]

- One learn new concepts by constructing mental objects.
- One solves a problem by mentally manipulating the involved objects (or structures).
- When one lacks a mental object to hang onto, one reduces the level of abstraction.

To select an appropriate level of abstraction is one of the issues in order to have an efficient problem solving process.

Through experience we grasp a new concept, and another important point is that we are later able to generalize/extend it to adapt to other relevant experiences.

While abstraction seems to be a characteristic of human thinking, it is claimed in many disciplines as diverse as painting, music, mathematics, computer science and software engineering. “Computational thinking is using abstraction and decomposition when ... designing a large complex system” [24].

However for computer science more is required as noted by Jeff Kramer [14] who stresses that in computing one should be “skilled at moving from an informal and complicated real world to a simplified abstract model”.

Teaching to write formal specifications addresses different issues that are intermingled. One issue is how to extract, from the addressed problem description, the relevant elements that should be specified (this means e.g. to remove repetitions, as well as descriptions that do not bring any matter as regards a computer processing, ...). These relevant elements often encompass a complex information structure that can in turn be described in a very precise, mathematical way. So this first issue addresses both finding information in the informal text, but also providing a structure to present this information in a meaningful, understandable way (as opposed to an unstructured myriad of yet important details).

After a given specification language is chosen, another issue is how to make the best use of it to produce a clear, elegant, legible specification of the above selected relevant elements of the problem. While some general rules (such as to always provide a natural language accompanying explanation - possibly with some schemas - together with the formal specification) may be given, this issue is clearly closely related with the specification language used. This paper reflects experience in teaching algebraic specifications, lately in the CASL language (Common Algebraic Specification Language) [2, 18], and CASL-LTL [22], an extension using Labelled Transition Logic for specifying dynamic systems. Experience in teaching Petri nets and coloured Petri nets [12] is also related, but the style issue for these is not addressed.

There are two remaining issues which are (i) does the specification comply with the requirements, (ii) are relevant properties expressible, and possibly proved or checked.

Now, when dealing with human brains, some facts are known from psychology, with of course some variations between individuals, that may be worth paying attention to when asking how students can be helped for this task.

### 1.1 The “magic” number seven

Miller [17] presents results concerning the human brain ability to process information, considering that “two bits of information enable us to distinguish among four equally likely alternatives”. When asked to distinguish between a number of inputs (along one variable dimension), for instance when asked to distinguish between different tones in (audible) frequency, between 2 to 14 would be recognized without error, that is from 1 to 3.8 bits of information. When several variables are used, this capacity is increased, but less than in an additive way. However, when using in an experiment “six different acoustic variables with each five different values, that is  $5^6$  different tones”, “the transmitted information was 7.2 bits of information, that is about 150 different categories identified without error.” [17] This number seven appears as striking as a (statistical) limit of our communication channel, given that if we make use of several variables we can manage to process significantly more information.

Let us note at this point that, while our physiological system is thus equipped that we deal with several variables in an auditive task without being aware of them, an issue is, when faced with a lot of information conveyed e.g. in a problem description (to be solved by a software system), how to find the relevant analysis variables that will help us distinguish and identify the relevant factors.

Another experiment shows that recalling a sequence of 18 bits (0 or 1) will be achieved much more efficiently as they are grouped by *chunks* of 2, 3, 4 or 5 to produce less numbers to recall. “We must recognize the importance of grouping or organizing the input sequence into units or chunks. Since the memory span is a fixed number of chunks, we can increase the number of bits of information that it contains simply by building larger and larger chunks, each chunk containing more information than before. . . this process would be called *recoding*. . . the simplest is to group the input events, apply a new name to the group, and then remember the new name rather than the original input events.” [17]

Thus human reasoning is easier if less elements have to be dealt with. A consequence for the specification concern is that it is desirable to keep this number low when structuring/decomposing into subproblems or subsystems that make sense (represented by a given name). There is of course no way to control what is the nature of the problem to be solved (and it may involve often numerous issues), the work to be done is to find out how to cluster related issues into composed ones, that is how to achieve the *recoding* mentioned above. This may also involve to differ some issues to a further step of processing where issues considered as “details” will be reintroduced and dealt with, in other words, details are *abstracted* away until reintroduced in a *refinement* step. As stated by Jeannette Wing [24], “Computational thinking is thinking in terms of multiple layers of abstractions, . . . and defining the relationships between those layers.”

## 1.2 Left brain, right brain

The left and right parts of our brain play different roles in our handling of tasks [16, 4]. The left part of the brain [4] is the logical, analytical, rational hemisphere involved in speech, reading, writing. The right part of the brain [4] is the the intuitive, creative hemisphere, involved in dealing with the visual world, with patterns, etc., and it is also active for synthesis activities. An individual organization of the brain integrating optimally these two parts will result in better performances.

Those results were considered for a better teaching activity, with a recommendation to try and address both brain hemispheres (for instance, talking is only addressing the left hemisphere, so it could be accompanied with some visual artifacts). A nice example is shown in [15] where colored simple geometrical figures are proposed to illustrate and explain calculus for elementary algebra, simple polygons areas, finite series, and differential calculus.

When working with and teaching formal specifications, this may be kept in mind, and specifications that combine both text/mathematical formulas and visual graphics are likely to be best written and understood, and should therefore be promoted. The analytical gifts associated with the left hemisphere are needed

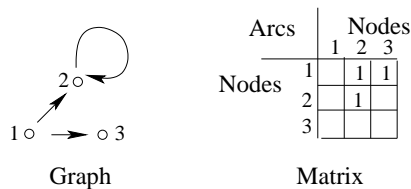
so as to adequately analyse all the elements of the problem to be specified. While the synthesis capacity associated with the right hemisphere is essential in the elaboration of a structure that will put the various elements in a meaningful setting. The use of *patterns* may also be quite useful for this activity.

## 2 Finding the problem relevant elements

### 2.1 Searching at an abstract level

As pointed out in the introduction, the activity of finding out the relevant elements to be described in the specification should be done having in mind abstraction and clustering goals. If this is not kept in mind, one may be overwhelmed with a great number of different issues, and trying to understand them thoroughly just adds to the complexity, while the way to an integrated structure is pushed further away.

It turns out that reasoning at an abstract level may not be that obvious to some students. A number of years ago, while a cognitive study of the software engineering course was done [23], the students were given in an exam a specification exercise to produce an algebraic specification of graphs. Half of them produced a specification that involved operations like adding a node, adding an arc, etc. The other half went on specifying the adjacency matrix, which turned out to be (i) quite concrete since it is one of the possible graph representations they learned in the algorithmics course, (ii) much more complicated to write, very difficult to read, and very much error-prone, and this may be illustrated by Figure 1.



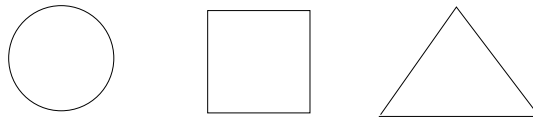
**Fig. 1.** Abstract and concrete views of a graph

Obviously, half of the students quite naturally reasoned at the abstract level, while the other half felt the need to switch to the concrete representation. A question then is whether it is possible to help this second half on the path of abstraction. All of them knew that they should look for generators/constructors and other operations, but it was not enough since they did that at a different level. The issue here seems to be as if one should “unplug” a brain connection between analysing a problem and writing a program, and accept to ground one’s thoughts in this abstract ground.

Assuming that the students are able to work at the abstract level (which seems to be true if they can think in terms of a matrix, but in general may be a question [14]), would there be a way to phrase the problem in such a way that abstraction would be enforced ?

One approach to provide abstractions ready to use is given through the use of *patterns*. Patterns are abstract schemas representing structures that are identified as often found in software engineering. They offer families of frequently met structures that the user is invited to “try” (or even to adapt) on the problem to be solved, so as to benefit from “ready to wear” structuring concepts. Although Figure 2 exhibits shapes that are much simpler, the parallel may be interesting. What is proposed is, through trial and error, to acquire a first broad view of the nature of the problem.

Patterns may be viewed as an elaborated mean of reusing knowledge acquired from experience. They address different levels. Problems frames [11] are proposed by M. Jackson to provide a global structure to problems, while architectural styles [9, 1] bring structures of a finer granularity that are often usable at the design level.



**Fig. 2.** What is the shape of my problem ?

At the requirement specification level, the use of problem frames may be quite helpful in getting the specifier started. A limited number of basic problem frames is given, and the approach is to try and find whether the problem studied may be matched against one or several of them (in combination). This matching goes with identifying the problem frames different elements in the problem under study. While this may seem oversimplified because it may be that the matching is not fully adequate and the closest problem frame should be adapted or combined with another one, the thinking process got started and some relevant elements will emerge. The questions raised (by whether a problem frame is matched) help in finding the nature of the problem (and also in finding what it is not), and a first sketch of a structure. Therefore this activity involves the pattern and the visual graphics of the problem frame diagram together with the analytical reasoning about the requirements description text, thus potentially addressing optimally the brain capacities.

## 2.2 Searching in the informal description

Another issue is how to analyze a text in natural language (but also possibly graphics) so as to extract the relevant items for the specification. Usually an

informal description will include some redundancy and some irrelevant text, possibly some ambiguity, and also some contradictions or inconsistencies. We do not address here work done with natural language processing to help sorting things out. Again here providing a set of questions, of elements to look for, of potential properties to express, may *guide* the process of extracting the relevant information from the text. In fact the idea is similar to that of a pattern to match.

In what follows, method paths will be shortly described in the light of these ideas.

### 3 Formal specification methods

As mentioned above, some patterns address a coarse granularity and are useful to sketch a first rough structure of a problem, and some patterns are of a finer grain and are needed when subparts are detailed and thoroughly specified. To start and find a rough decomposition of a problem, problem frames are rather adequate and will be discussed in Section 3.1, use cases may also be helpful. The “in the large” patterns are diverse and may be combined when a large system is addressed. As in Figure 2, the issue is whether trying out some shapes with the help of some guidance/questions, a relevant concept will emerge. Some “in the small” patterns will then be presented in Sections 3.2 and 3.3.

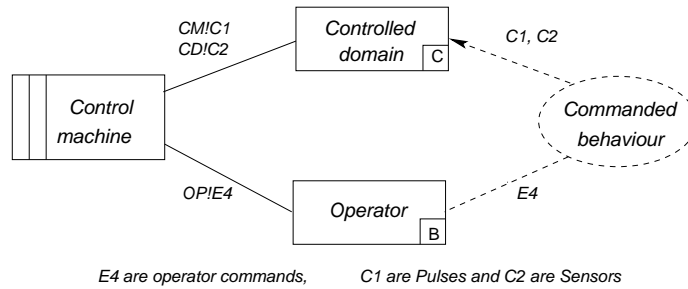
#### 3.1 Using problem frames patterns

Problem Frames [11] present, classify, understand software development problems, and provide a characterisation of a class of problems in terms of their main components and the connections between these components, as shown in Figure 3. A problem frame diagram includes the involved domains (here the *Controlled Domain*, and the *Operator*), the requirements (*Commanded Behaviour*), the design (*Control Machine*), and their interfaces (straight or dashed lines, labelled with shared phenomena). The involved domains may be of different kinds (here the *Controlled Domain* is **C** for causal - which means it can cause some evolution in its state, while the *Operator* is **B** for biddable, which applies for people), and this information helps in choosing which problem frame to match with.

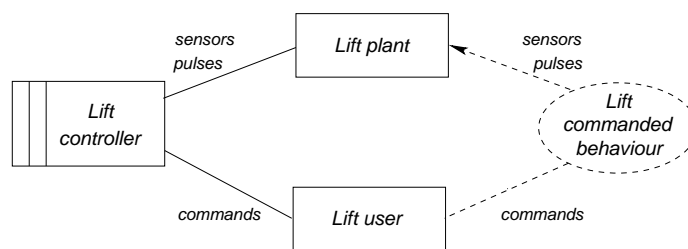
Jackson identified five basic problem frames together with some variants. In fact, questioning the nature of the domains and whether a user is involved may already lead to sort out most possible basic problem frames. The Commanded Behaviour frame in Figure 3 addresses the following class of problems :

“There is some part of the physical world whose behaviour is to be controlled in accordance with commands issued by an operator. The problem is to build a machine that will accept the operator’s commands and impose the control accordingly.”

Jackson explains that once a problem is successfully fitted to a problem frame, its most important characteristics are known. For instance, Figure 4 provides



**Fig. 3.** The Commanded Behaviour problem frame



**Fig. 4.** The Lift instance of the Commanded Behaviour problem frame

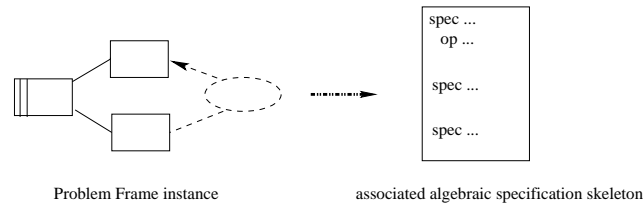
an instance of the Commanded Behaviour Frame for a lift problem, and some information on what to take into account is put to light, that is the lift plant and its sensors, the lift user commands, the actuators managed by the lift controller.

While the diagrams provided for the basic problem frames are quite simple, as mentioned in Section 2.1, trying to match one of them leads to progress in understanding the nature of the problem. It may be the case that the matching depends on the viewpoint and that several candidates are eligible, so one of them has to be chosen, possibly with some argument for the choice. It may be also the case that some adaptation or combination of problem frames should be done before a successful matching is obtained. In all cases, a better understanding is gained, and different parts are sorted out for further precise formal specifications.

In [7], algebraic specifications skeletons are associated with basic problem frames, and these skeletons show which elements should appear in the specification to comply with the nature of the system (Figure 5). Thus guidance to a much more detailed and precise specification is provided, and a combined use of the problem frame diagrams and the mathematical specification is proposed.

### 3.2 Using guidelines

As mentioned in Section 2.2, guidelines, e.g. expressed by questions and elements or properties to look for, help in finding relevant ingredients in an informal text



**Fig. 5.** Associating an algebraic specification skeleton to a problem frame

describing the requirements so as to provide a formal specification of them. At this stage, we do not think these guidelines are necessarily tightly linked with a specific language or paradigm. Although the works presented in [8, 6, 20, 21] were applying specification method ideas for a specific target language, a number of ideas may be considered independently from a given specification language. As stated earlier in Section 1, the issue of making the best use of a given language should be considered separately.

Another pitfall is the wide variety of applications and systems, while it may be tempting to design different methods for different kinds of applications, one may end up with too many different methods to learn to use. In [8], the software items distinguished are (i) a simple dynamic system (where "simple" means that it does not contain subsystems), (ii) a structured dynamic system (which may be considered as a specialisation of a simple one), and (iii) a datatype. For each item, the relevant characteristic features to look for are presented, together with a graphic presentation to express them. For a simple system, these features are state observers (that are a way to characterize a state of the system) and the elementary interactions (that yield state transitions). In turn, for these features, the possible properties to look for are proposed.

All these guidelines are *patterns* expressing structures drawn from experience. It should be noted that these patterns address a finer grain, while the problem frames are coarse grain. Of course the two can be combined as shown in [8].

While this method was originally used to produce modular algebraic specifications of complex systems, it was felt that it could be used for a variety of languages, and even for UML descriptions. Recently it was experimented and adapted for coloured Petri nets [6].

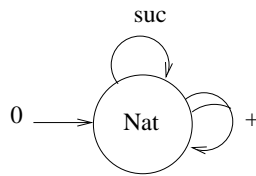
### 3.3 Addressing a specific language

In this section, just a snapshot of what an algebraic specification language may require is given since this issue was addressed by many [18, 2], with different viewpoints (sometimes hinted by the tools one had in mind).

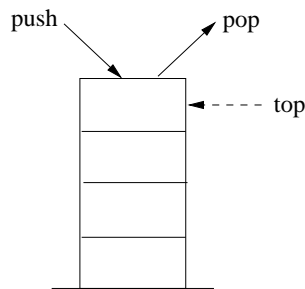
Algebraic specifications of complex systems are written in a modular way, relevant datatypes are identified and equipped with generators (sometimes called constructors), predicates and operations. As mentioned earlier, it is useful to use graphical representations associated with them. For instance, Figure 6 illustrates

the operations associated with the Nat domain, while Figure 7 illustrates the stack operations behaviour.

The (generators, predicates, operations) properties are expressed through axioms. Styles in writing axioms help in checking that cases are not forgotten (the constructive style relies on the generators/constructors, while the observational style relies on selected observers - operations and/or predicates). The logic used for writing the axioms is first order when dealing with datatypes, and a temporal logic when dealing with dynamic systems.



**Fig. 6.** Graphic representation of the Nat signature



**Fig. 7.** Graphic representation of the Stack structure

Modularity should be dealt with care so that the information is not spread out in irrelevant modules.

When a specification is refined, some constructions are provided so that the changes do not alter the semantics in an unintended way.

## 4 Conclusion

Much work was achieved to provide better languages and good tools for formal specifications so as to ease their use and make it more attractive and machine equipped.

While formal specifications bring an opportunity to produce models with which it is possible to reason about and hence to feel more confident with, they remain somewhat difficult to address when a system is complex.

Our experience is that to be able to work at an abstract level with formal specifications is essential. While some students may not do this, a question is whether it is possible to help them. Some facts of psychology shed some light on why some help may then come from visual presentations (to help to read formal specifications), from patterns (to help to structure them), and from guidelines (to help to write them). It is also shown here that different kind of patterns may be used in a combined way at different steps.

The patterns provided by problem frames are used by researchers using different formal specification languages [3, 13]. Let us note that a use case decomposition may also be considered [5].

The validation and verification issues were not addressed in this paper, however it is recommended to carefully check how the informal requirements description is reflected in the formal specification, and to provide a precise account on questions raised and answers given or choices made during the process, for further reference upon maintenance. As regards properties, they may be expressed in different ways depending on the specification language. When an algebraic specification language is used, they may be expressed by the axioms. When using coloured Petri nets, pre and postconditions (complying with some adequate form) are expressed through the net transitions, while other properties may either be expressed through guards or datatypes axioms, or model checked.

We plan to extend the specification method ideas to deal more extensively with hierarchy, and with real-time systems. It seems that visual artifacts for properties are still very few, and that more are needed to help to manipulate them more "fluently".

**Acknowledgements :** My interest in psychology comes from my parents and their dedicated passion in their profession. The chance to teach students and observe their reactions gave me good opportunities to think about this work. Sylvain Surcin, an ex-student, came to offer to study my software engineering course from a didactic viewpoint, discussing with him was a chance to interact with someone who both knows what formal specifications and software engineering are about, and learned on cognitive processes. The workshops organized by Alain Finkel on cognitive sciences for teaching computer science and mathematics are a good place to keep this motivation alive. The work and discussions on formal specification methodology I had with my colleagues Gianna Reggio, Maritta Heisel, Laure Petrucci, and Anne Haxthausen, were stimulating and enlightening, and I could apply these ideas in teaching.

## References

1. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998.

2. M. Bidoit and P. D. Mosses. *CASL User Manual, Introduction to Using the Common Algebraic Specification Language*. Number 2900 in Lecture Notes in Computer Science. Springer-Verlag, 2004.
3. D. Bjorner, S. Kousoube, R. Noussi, and G. Satchok. Michael Jackson's Problem Frames: Towards Methodological Principles of Selecting and Applying Formal Software Development Techniques and Tools. In M. Hinchey and L. ShaoYing, editors, *Proc. Intl.Conf. on Formal Engineering Methods, Hiroshima, Japan, 12-14 Nov.1997*, pages 263–270. IEEE CS Press, 1997.
4. J. T. Bruer. In Search of . . . Brain-Based Education. *Phi Delta Kappan*, pages 649–57, May 1999.
5. C. Choppy and M. Heisel. Une approche à base de "patrons" pour la spécification et le développement de systèmes d'information. In *Proceedings Approches Formelles dans l'Assistance au Développement de Logiciels - AFADL'2004*, pages 61–76, 2004.
6. C. Choppy, L. Petrucci, and G. Reggio. Designing coloured Petri net models : a method. In *Proc. CPN'07, Eighth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, 2007.
7. C. Choppy and G. Reggio. Using CASL to Specify the Requirements and the Design: A Problem Specific Approach. In D. Bert, C. Choppy, and P. D. Mosses, editors, *Recent Trends in Algebraic Development Techniques, 14th WADT, Selected Papers*, LNCS 1827, pages 104–123. Springer Verlag, 2000.
8. C. Choppy and G. Reggio. A formally grounded software specification method. *Journal of Logic and Algebraic Programming*, 67(1-2):52–86, 2006.
9. D. Garlan and M. Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, volume 1. World Scientific Publishing Company, 1993.
10. O. Hazzan. Reducing abstraction level when learning computability theory concepts. *ACM SIGCSE Bulletin*, 34(3), 2002.
11. M. Jackson. *Problem Frames. Analyzing and structuring software development problems*. Addison-Wesley, 2001.
12. K. Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use, vol. 1, vol. 2 et vol. 3*. Springer-Verlag, London, UK, 1995.
13. J. Jorgensen. Addressing Problem Frame Concerns Using Coloured Petri Nets and Graphical Animation. In *International Workshop on Advances and Applications of Problem Frames*, 2006.
14. J. Kramer. Is abstraction the key to computing? *Commun. ACM*, 50(4):36–42, 2007.
15. Y. Lafont. Quelques représentations géométriques du calcul, 2003.
16. J. Levy. Right brain, left brain: Fact and fiction. *Psychology Today*, pages 43–44, May 1985.
17. G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63:81–97, 1956.
18. P. D. Mosses, editor. *CASL, The Common Algebraic Specification Language - Reference Manual*. Number 2960 in Lecture Notes in Computer Science. Springer-Verlag, 2004.
19. J. Piaget and B. Inhelder. *The psychology of the child*. Routledge and Kegan Paul, 1969.
20. P. Poizat, C.Choppy, and J.-C. Royer. Concurrency and Data Types: a Specification Method. An Example with LOTOS. In J. Fiadero, editor, *Recent Trends in Algebraic Development Techniques, Selected Papers of the 13th International*

- Workshop WADT98*, number 1589 in Lecture Notes in Computer Science, pages 276–291. Springer Verlag, Berlin, 1999.
21. P. Poizat, C. Choppy, and J.-C. Royer. From Informal Requirements to COOP: a Concurrent Automata Approach. In J. Wing, J. Woodcock, and J. Davies, editors, *FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems*, number 1709 in Lecture Notes in Computer Science, pages 939–962. Springer Verlag, Berlin, 1999.
  22. G. Reggio, E. Astesiano, and C. Choppy. CASL-LTL: A CASL extension for dynamic reactive systems – version 1.0 – summary. Technical Report DISI-TR-03-36, Università di Genova, Italy, 2003.
  23. S. Surcin, C. Choppy, and M.-G. Séré. Analyse didactique d'un cours de génie logiciel basé sur l'approche orientée objets. *Revue Sciences et Techniques Educatives*, 2(3):265–286, 1995.
  24. J. Wing. Computational thinking. *Commun. ACM*, 49(3):33–35, 2006.