



Ahoo Engineering Group

Software Engineering

by Nguyen Xuan Huy

Institute of Information Technology,

National Center for Natural Science and Technology

Email: nxhuy@ioit.ncst.ac.vn

Duration: 45 hours



Objectives

This chapter concerns both formal and informal specification techniques and their use in the software process. The role of specification in the software process and the major problems concerned with producing complete and consistent specifications are discussed in the first part of the chapter. Two techniques for specification, algebraic and model-based specification are described in the rest part of the chapter. Algebraic specification is a specification technique where the actions on an object are specified in terms of their relationships. This is particularly appropriate when used in conjunction with an object-oriented approach to design as it follows the object classes to be formally specified. Model-based specification is a complementary technique where the system is modeled using mathematical entities, such as sets, whose properties are well understood.

Contents

- 2.1 Structure and contents of the requirements definition
 - 2.2 Quality criteria for requirements definition
 - 2.3 Fundamental problems in defining requirements
 - 2.4 Algebraic specification
 - 2.5 Model-based specification
 - References and selected reading
-

2.1 Structure and contents of the requirements definition

The sectional organization of the requirements definition (ANSI/IEEE guide to *Software Requirement Specification* [ANSI 1984]) are:

1. Initial situation and goals
2. System application and system environment
3. User interfaces
4. Functional requirements
5. Nonfunctional requirements
6. Exception handling
7. Documentation requirements
8. Acceptance criteria



9. Glossary and index

1. Initial situation and goals

Contents:

A general description of

- ❖ The initial situation with reference to the requirements analysis,
- ❖ The project goals, and
- ❖ A delimitation of these goals with respect to the system environment.

2. System application and system environment

Contents:

- ❖ Description of the prerequisites that must apply for the system to be used.

Note:

- *The description of all information that is necessary for the employment of the system, but not part of the implementation.*
- ❖ Specification of the number of users, the frequency of use, and the jobs of the users.

3. User interfaces

Contents:

- ❖ The human-machine interface.

Notes:

- This section is one of the most important parts of the requirements definition, documenting how the user communicates with the system.
- The quality of this section largely determines the acceptance of the software product.

4. Functional requirements

Contents:

- ❖ Definition of the system functionality expected by the user
- ❖ All necessary specifications about the type, amount and expected precision of the data associated with each system function.

Notes:

- *Good specifications of system functionality contain only the necessary information about these functions.*
- *Any additional specifications, such as about the solution algorithm for a function, distracts from the actual specification task and restricts the flexibility of the subsequent system design.*
- *Only exact determination of value ranges for data permits a plausibility check to detect input errors.*



5. Nonfunctional requirements

Contents:

- ❖ Requirements of nonfunctional nature: reliability, portability, and response and processing times ...

Note:

- *For the purpose of the feasibility study, it is necessary to weight these requirements and to provide detailed justification.*

6. Exception handling

Contents:

- ❖ Description of the effects of various kinds of errors and the required system behavior upon occurrence of an error.

Note:

- *Developing a reliable system means considering possible errors in each phase of development and providing appropriate measures to prevent or diminish the effects of errors.*

7. Documentation requirements

Contents:

- ❖ Establish the scope and nature of the documentation.

Note:

- *The documentation of a system provides the basis for both the correct utilization of the software product and for system maintenance.*

8. Acceptance criteria

Contents:

- ❖ Establishing the conditions for inspection of the system by the client.

Notes:

- *The criteria refer to both functional and nonfunctional requirements*
- *The acceptance criteria must be established for each individual system requirement. If no respective acceptance criteria can be found for a given requirement, then we can assume that the client is unclear about the purpose and value of the requirement.*

9. Glossary and index

Contents:

- ❖ A glossary of terms
- ❖ An extensive index

Notes:



- *The requirements definition constitutes a document that provides the basis for all phases of a software project and contains preliminary considerations about the entire software life cycle*
- *The specifications are normally not read sequentially, but serve as a reference for lookup purposes.*

2.2 Quality criteria for requirements definition

- It must be *correct* and *complete*.
- It must be *consistent* and *unambiguous*.
- It should be *minimal*.
- It should be *readable* and *comprehensible*.
- It must be readily *modifiable*.

2.3 Fundamental problems in defining requirements

The fundamental problems that arise during system specification are [Keller 1989]:

- The goal/means conflict
- The determination and description of functional requirements
- The representation of the user interfaces

The goal/means conflict in system specification.

- The primary task of the specification process is to establish the *goal* of system development rather than to describe the *means* for achieving the goal.
- The requirements definition describes *what* a system must do, but not *how* the individual functions are to be realized.

Determining and describing the functional requirements.

- Describing functional requirements in the form of text is extremely difficult and leads to very lengthy specifications.
- A system model on the user interface level serving as an executable prototype supports the exploration of functional, nonfunctional and interaction-related requirements.
- It simplifies the determination of dependencies between system functions and abbreviates the requirements definition.
- A prototype that represents the most important functional aspects of a software system represents this system significantly better than a verbal description could do.

Designing the user interfaces.

- User interfaces represent a user-oriented abstraction of the functionality of a system.
- The graphical design of screen layouts requires particular effort and only affects one aspect of the user interface, its appearance.
- The much more important aspect, the dynamics behind a user interface, can hardly be depicted in purely verbal specifications.

Therefore the user interface components of the requirements definition should always be realized as an executable prototype.

2.4 Algebraic specification

Algebraic specification [Guttag 1977] is a technique whereby an object is specified in terms of the relationships between the operations that act on that object.

A specification is presented in four parts (Figure 2.1):

1. **Introduction** part where the sort of the entity being specified is introduced and the name of any other specifications which are required are set out
2. **Informal description** of the sort and its operations
3. **Signature** where the names of the operations on that object and the sorts of their parameters are defined
4. **Axioms** where the relationships between the sort operations are defined.

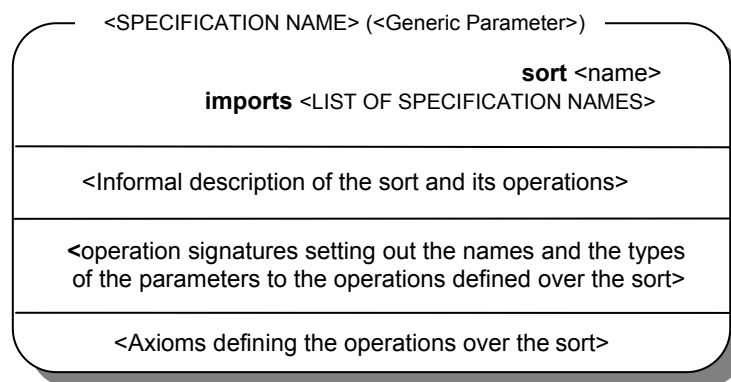


Figure 2.1 The format of an algebraic specification.

Note:

- *The introduction part of a specification also includes an **imports** part which names the other specifications which are required in a specification.*

Description part:

- ❖ Formal text with an informal description

**Signature part:**

- ❖ Names of the operations which are defined over the sort,
- ❖ Number and sort of their parameters, and
- ❖ Sort of the result of evaluating of the operation.

Axioms part:

- ❖ Operations in terms of their relationships with each other.

Two classes of operations:

- *Constructor operations*: Operation that create or modify entities of the sort which is defined in the specification.
- *Inspection operations*: Operations that evaluate attributes of the sort which defined in the specification.

Example (Figure 2.2)

Sort: Coord.

Operations:

- Creating a coordinate,
- Testing coordinates for equality, and
- Accessing the X and Y components.

Imports:

Two specifications:

- BOOLEAN
- INTEGER.

Note:

- *In the specification of the Eq operation the operator '=' is overloaded.*

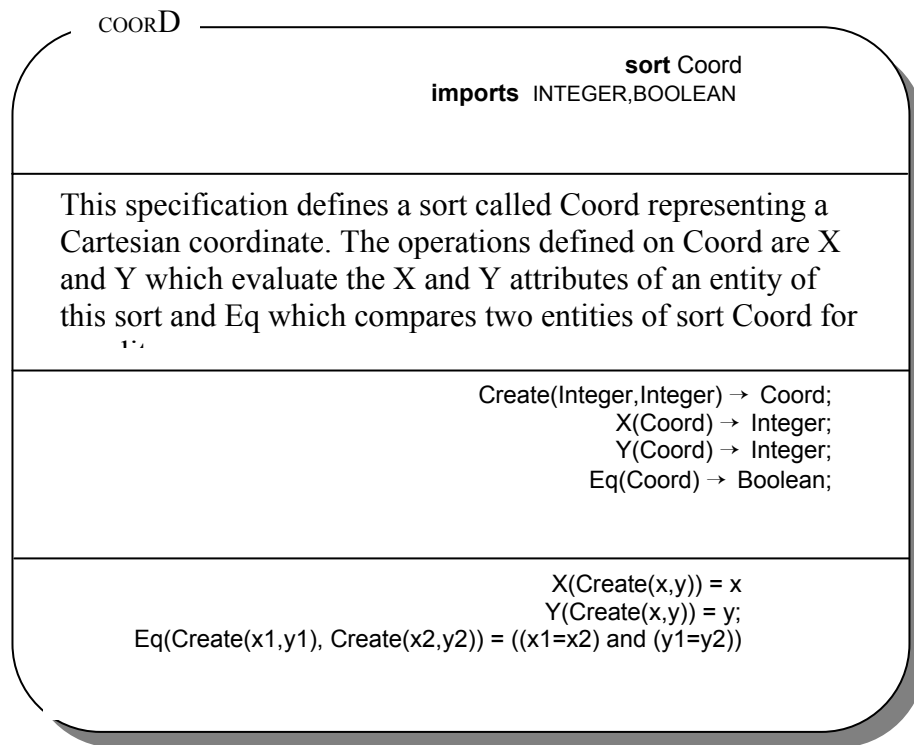


Figure 2.2 The specification of Coord.

2.5 Model-based specification

Specification languages:

- Z ([Abrial 1980], [Hayes 1987])
- VDM ([Jones 1980], [Jones 1986])
- RAISE ([RAISE 1992])

Note:

- *Z is based on typed set theory because sets are mathematical entities whose semantics are formally defined.*

Advantages:

- ❖ Model-based specification is a technique that relies on formulating a model of the system using well understood mathematical entities such as sets and functions
- ❖ System specifications are specified by defining how they affect the overall system model
- ❖ By contrast with the algebraic approach, the state of the system is exposed and a richer variety of mathematical operations is available
- ❖ State changes are straightforward to define
- ❖ All of the specification for each operation is grouped together
- ❖ The model is more concise than corresponding algebraic specifications.



A specification in Z is presented as a collection of schemas where a schema introduces some specification entities and sets out relationships between these entities.

Schema form: (Figure 2.3)

- *Schema name* (the top line)
- *Schema Signature* sets out the names and types of the entities introduced in the schema.
- *Schema predicate* (the bottom part of the specification) sets out the relationships between the entities in the signature by defining a predicate over the signature entities which must always hold.

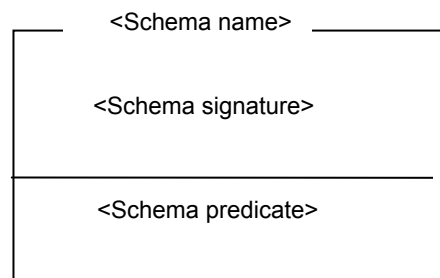


Figure 2.3 Schema form

Example 2.1

A specification of a container which can be filled with “thing” (Figure 2.4).

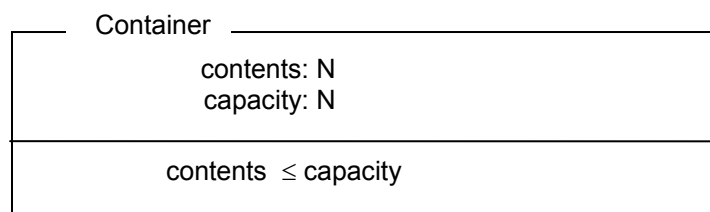


Figure 2.4 The specification of a container

- *Schema name:* Container
- *Schema signature:*
contents: a natural number
capacity: a natural number
- *Schema predicate:* contents \leq capacity
(*contents cannot exceed the capacity of the container.*)



Example 2.2

A specification of an indicator (Figure 2.5).

- *Schema name:* Indicator
- *Schema signature:*
light: {off, on}
reading: a natural number
danger: a natural number
- *Schema predicate:* $\text{light} = \text{on} \Leftrightarrow \text{reading} \leq \text{danger}$

Notes:

- *Light is modeled by the values off and on,*
- *Reading is modeled as a natural number*
- *Danger is modeled as a natural number.*
- *The light should be switched on if only if the reading drops to some dangerous value*

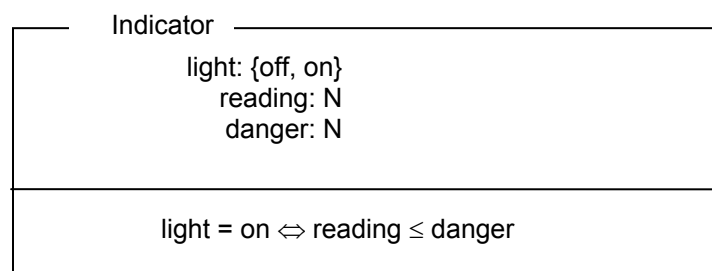


Figure 2.5 The specification of an indicator

Example 2.3

Given the specification of an indicator and a container, they can be combined to define a hopper which is a type of container (Figure 2.6).

- *Schema name:* Hopper
- *Schema signature:*
Container
Indicator
- *Schema predicate:*
reading = contents
capacity = 5000
danger = 50

Notes:

- *Hopper which has a capacity of 5000 “things”*
- *Light comes on when contents drops to 1% full*
- *We need not specify what is held in the hopper.*

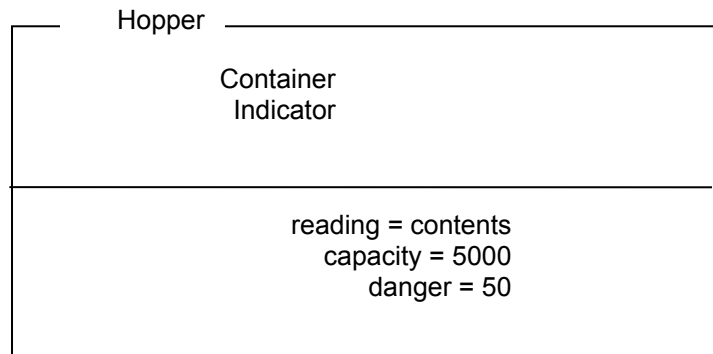


Figure 2.6 The specification of a hopper

The effect of combining specifications is to make a new specification which inherits the signatures and the predicates of the included specifications. Thus, hopper inherits the signatures of Container and Indicator and their predicates. These are combined with any new signatures and predicates which are introduced in the specification. In Figure 2.7, three new predicates are introduced.

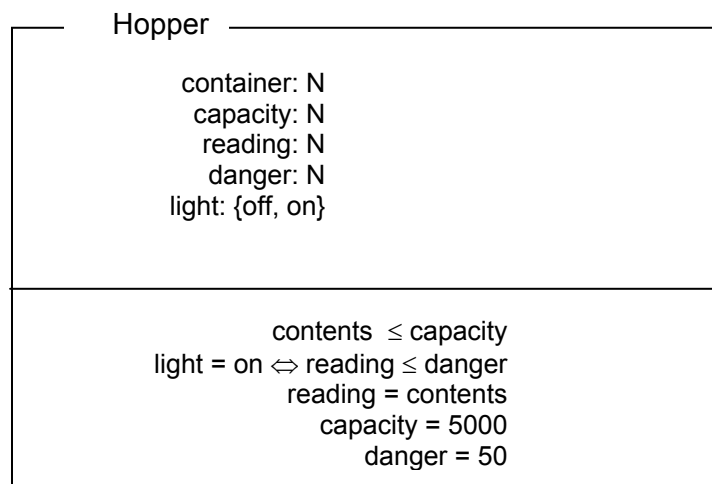


Figure 2.7 The expanded specification of a hopper

Example 2.4

Operation FillHopper (Figure 2.8)

- ❖ The fill operation adds a specified number of entities to the hopper.

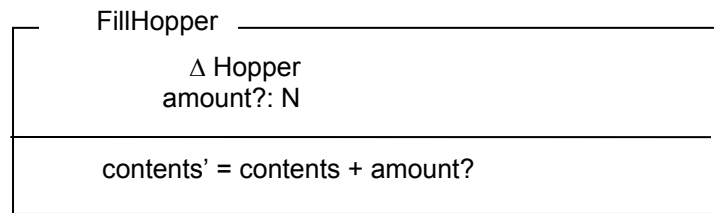


Figure 2.8 The specification of the hopper filling operation

New notions:

- Delta schemas (Figure 2.9), and
- Inputs.

? symbol:

- ? is a part of the name
- Names whose final character is a ? are taken to indicate inputs.

Predicate: contents' = contents + amount?

(the contents after completion of the operation (referred as contents') should equal the sum of the contents before the operation and the amount added to the hopper).

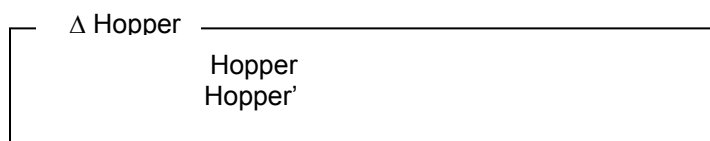


Figure 2.9 A delta schema

New notions:

- Xi schemas (Figure 2.10)

Some operations do not result in a change of value but still find it useful to reference the values before and after operation.

Figure 2.10 (a Xi schema) shows a schema which includes the delta schema and a predicate which states explicitly that the values are unchanged.

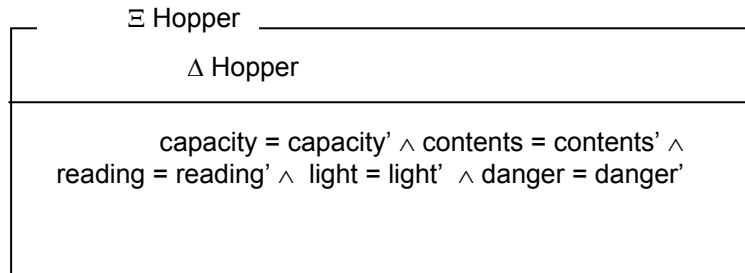


Figure 2.10 A Xi schema

One of the most commonly used techniques in model-based specifications is to use functions or mappings in writing specifications. In programming languages, a *function* is an abstraction over an expression. When provided with an input, it computes an output value based on the value of the input. A *partial function* is a function where not all possible inputs have a defined output.

The *domain* of a function is the set of inputs over which the function has a defined result.

The *range* of a function is the set of results which the function can produce.

If ordering is important, *sequences* can be used as a specification mechanism. Sequences may be modeled as functions where the domain is the natural numbers greater than zero and the range is the set of entities which may be held in the sequence.

References and selected reading

- [Abrial 1980] Abrial J. R., *The specification language Z: basic library*, Oxford Univ. Programming Res. Group, 1980
- [ANSI 1984] *ANSI/IEEE Std. 830-1984 Software Requirements*, IEEE 1984 Specification
- [Guttag 1977] Guttag J., Abstract data types and the development of data structures, *CACM*, 20 (6) 1977, 396-405
- [Hayes 1987] Hayes I. (ed.) *Specification Case Studies*, Prentice Hall, 1987
- [Jones 1980] Jones C. B., *Software Development – A Rigorous Approach*, Prentice Hall, 1980
- [Jones 1986] Jones C. B., *Systematic Software Development Using VDM*, Prentice Hall, 1986
- [Keller 1989] Keller R.K., *Prototypingorientierte Systemspezifikation*, Verlag Dr. Kovac, 1989
- [Pomberger 1996] Pomberger G. and Blaschek G., *Object Orientation and Prototyping in Software Engineering*, Translated by Bach R., Prentice Hall, 1996
- [RAISE 1992] RAISE Language Group, *The RAISE specification language*, Prentice Hall, 1992



Ahoo Engineering Group

- [Pressman 1997] Pressman R., *Software Engineering: A practitioner's Approach*, (4th ed.), McGraw-Hill, 1997
- [Sommerville 1996] Sommerville I., *Software Engineering* (5th ed.), Addison-Wesley, 1996.