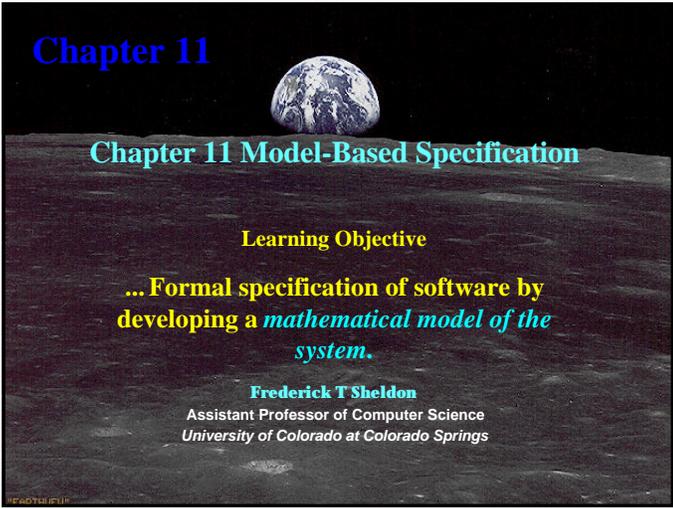


**Chapter 11**



**Chapter 11 Model-Based Specification**

**Learning Objective**

**... Formal specification of software by developing a *mathematical model of the system.***

**Frederick T Sheldon**  
Assistant Professor of Computer Science  
University of Colorado at Colorado Springs

---

---

---

---

---

---

---

---

**Objectives**

---

- To introduce an approach to formal specification based on mathematical system models
- To present some features of the Z specification language
- To illustrate the usefulness of Z by describing small examples
- To show how Z schemas may be used to develop incremental specifications

CS-422 Software Engineering Principles  
From Software Engineering by I. Sommerville, 1996.

Chapter 11  
Slide 2

---

---

---

---

---

---

---

---

**Topics covered**

---

- Z schemas
- The Z specification process
- Specifying ordered collections

CS-422 Software Engineering Principles  
From Software Engineering by I. Sommerville, 1996.

Chapter 11  
Slide 3

---

---

---

---

---

---

---

---

## Model-based specification

Defines a model of a system using well-understood mathematical entities such as sets and functions.

The state of the system is not hidden (unlike algebraic specification).

State changes are straightforward to define.

VDM and Z are the most widely used model-based specification languages.

---

---

---

---

---

---

---

---

## Z as a specification language

Based on typed set theory

Probably now the most widely-used specification language

Includes schemas, an effective low-level structuring facility

Schemas are specification building blocks

Graphical presentation of schemas make Z specifications easier to understand

---

---

---

---

---

---

---

---

## Z schemas

Introduce specification entities and defines invariant predicates over these entities

A schema includes

- A name identifying the schema

- A signature introducing entities and their types

- A predicate part defining invariants over these entities

Schemas can be included in other schemas and may act as type definitions

Names are local to schemas

---

---

---

---

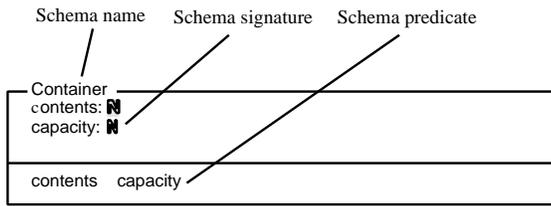
---

---

---

---

## Z schema highlighting



---

---

---

---

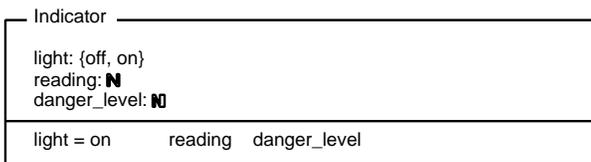
---

---

---

---

## An indicator specification



---

---

---

---

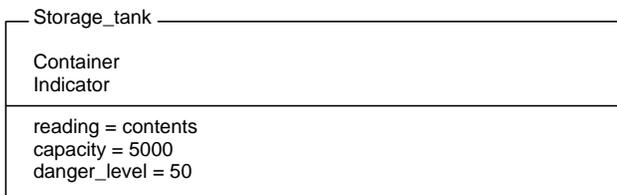
---

---

---

---

## Storage tank specification



---

---

---

---

---

---

---

---

## Full specification of a storage tank

Storage\_tank

contents: **N**  
capacity: **N0**  
reading: **N**  
danger\_level: **N0**  
light: {off, on}

contents capacity  
light = on reading danger\_level  
reading = contents  
capacity = 5000  
danger\_level = 50

---

---

---

---

---

---

---

---

## Z conventions

A variable name decorated with a quote mark ( $N'$ ) represents the value of the state variable  $N$  after an operation

A schema name decorated with a quote mark introduces the dashed values of all names defined in the schema

A variable name decorated with a ! represents an output

---

---

---

---

---

---

---

---

## Z conventions

A variable name decorated with a ? represents an input

A schema name prefixed by the Greek letter Xi ( $\xi$ ) means that the defined operation does not change the values of state variables

A schema name prefixed by the Greek letter Delta ( $\Delta$ ) means that the operation changes some or all of the state variables introduced in that schema

---

---

---

---

---

---

---

---

## Operation specification

Operations may be specified incrementally as separate schema then the schema combined to produce the complete specification

Define the 'normal' operation as a schema

Define schemas for exceptional situations

Combine all schemas using the disjunction (or) operator

---

---

---

---

---

---

---

---

## A partial spec. of a fill operation

Fill-OK

Storage\_tank  
amount?:  $\mathbb{N}$

contents + amount? < capacity  
contents' = contents + amount?

---

---

---

---

---

---

---

---

## Storage tank fill operation

OverFill

Storage-tank  
amount?:  $\mathbb{N}$   
r!: seq CHAR

capacity < contents + amount?  
r! = "Insufficient tank capacity - Fill cancelled"

Fill

Fill-OK  $\vee$  OverFill

---

---

---

---

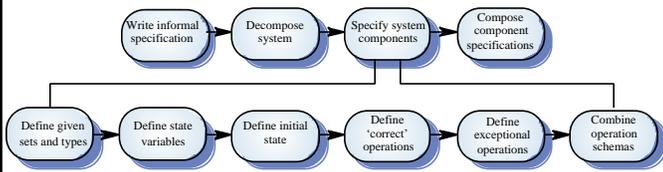
---

---

---

---

## The Z specification process



---

---

---

---

---

---

---

---

## Data dictionary specification

Data dictionary, introduced in Chapter 6, will be used as an example. This is part of a CASE system and is used to keep track of system names

### Data dictionary structure

Item name

Description

Type. Assume in these examples that the allowed types are those used in semantic data models

Creation date

---

---

---

---

---

---

---

---

## Given sets

Z does not require everything to be defined at specification time

Some entities may be 'given' and defined later

The first stage in the specification process is to introduce these given sets

[NAME, DATE]

We don't care about these representations at this stage

---

---

---

---

---

---

---

---

## Type definitions

There are a number of built-in types (such as INTEGER) in Z

Other types may be defined by enumeration

Sem\_model\_types = { relation, entity, attribute }

Schemas may also be used for type definition

The predicates serve as constraints on the type

---

---

---

---

---

---

---

---

## Specification using functions

A function is a mapping from an input value to an output value

SmallSquare = { 1 1, 2 4, 3 9, 4 16, 5 25,  
6 36, 7 49 }

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

dom SmallSquare = { 1, 2, 3, 4, 5, 6, 7 }

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

rng SmallSquare = { 1, 4, 9, 16, 25, 36, 49 }

---

---

---

---

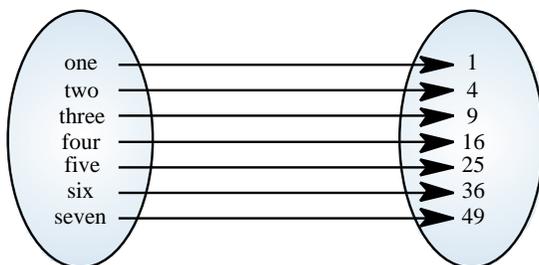
---

---

---

---

## The function SmallSquare



Domain (SmallSquare)

Range (SmallSquare)

---

---

---

---

---

---

---

---

## Data dictionary modeling

A data dictionary may be thought of as a mapping from a name (the key) to a value (the description in the dictionary)

Operations are

Add. Makes a new entry in the dictionary or replaces an existing entry

Lookup. Given a name, returns the description.

Delete. Deletes an entry from the dictionary

Replace. Replaces the information associated with an entry

---

---

---

---

---

---

---

---

## Data dictionary entry

DataDictionaryEntry

entry: NAME  
desc: seq char  
type: Sem\_model\_types  
creation\_date: DATE

#description 2000

---

---

---

---

---

---

---

---

## Data dictionary as a function

DataDictionary

DataDictionaryEntry  
ddict: NAME → {DataDictionaryEntry}

---

---

---

---

---

---

---

---

## Data dictionary - initial state

Init-DataDictionary

DataDictionary'

ddict' =  $\emptyset$

---

---

---

---

---

---

---

---

## Add and lookup operations

Add\_OK

DataDictionary  
name?: NAME  
entry?: DataDictionaryEntry

name? dom ddict  
ddict' = ddict {name? : entry?}

Lookup\_OK

DataDictionary  
name?: NAME  
entry!: DataDictionaryEntry

name? dom ddict  
entry! = ddict (name?)

---

---

---

---

---

---

---

---

## Add and lookup operations

Add\_Error

DataDictionary  
name?: NAME  
error!: seq char

name? dom ddict  
error! = "Name already in dictionary"

Lookup\_Error

DataDictionary  
name?: NAME  
error!: seq char

name? dom ddict  
error! = "Name not in dictionary"

---

---

---

---

---

---

---

---

## Function over-riding operator

ReplaceEntry uses the function overriding operator (written  $\leftarrow$ ). This adds a new entry or replaces an existing entry.

```
phone = { Ian 3390, Ray 3392, Steve 3427 }
```

The domain of phone is {Ian, Ray, Steve} and the range is {3390, 3392, 3427}.

```
newphone = { Steve 3386, Ron 3427 }
```

```
phone ← newphone = { Ian 3390, Ray 3392, Steve 3386, Ron 3427 }
```

---

---

---

---

---

---

---

---

## Replace operation

Replace\_OK

```
DataDictionary  
name?: NAME  
entry?: DataDictionaryEntry
```

```
name? dom ddict  
ddict' ← {name? | entry?}
```

---

---

---

---

---

---

---

---

## Deleting an entry

Uses the domain subtraction operator (written  $\setminus$ ) which, given a name, removes that name from the domain of the function

```
phone = { Ian 3390, Ray 3392, Steve 3427 }
```

```
{Ian} \ phone
```

```
{Ray 3392, Steve 3427 }
```

---

---

---

---

---

---

---

---

## Delete entry

Delete\_OK

DataDictionary  
name?: NAME

name? dom ddict  
ddict' = {name?} ↯ ddict

---

---

---

---

---

---

---

---

## Specifying ordered collections

Specification using functions does not allow ordering to be specified

Sequences are used for specifying ordered collections

A sequence is a mapping from consecutive integers to associated values

---

---

---

---

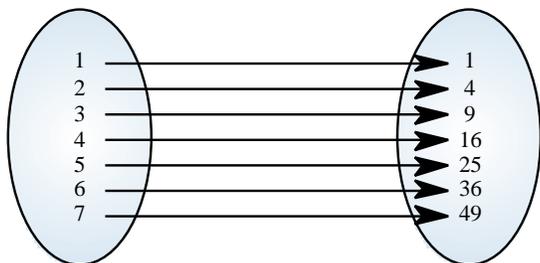
---

---

---

---

## A Z sequence



Domain (SqSeq)

Range (SqSeq)

---

---

---

---

---

---

---

---

## Data dictionary extract operation

The Extract operation extracts from the data dictionary all those entries whose type is the same as the type input to the operation

The extracted list is presented in alphabetical order

A sequence is used to specify the ordered output of Extract

---

---

---

---

---

---

---

---

## The Extract operation

Extract

```
DataDictionary
rep!: seq {DataDictionaryEntry}
in_type?: Sem_model_types
```

```
n : dom ddict • ddict(n). type = in_type?   ddict (n)   rng rep!
i : 1 i #rep! • rep! (i).type = in_type?
i : 1 i #rep! • rep! (i)   rng ddict
i, j: dom rep! • (i < j)   rep. name(i) <NAME rep.name (j)
```

---

---

---

---

---

---

---

---

## Extract predicate

For all entries in the data dictionary whose type is `in_type?`, there is an entry in the output sequence

The type of all members of the output sequence is `in_type?`

All members of the output sequence are members of the range of `ddict`

The output sequence is ordered

---

---

---

---

---

---

---

---

## Data dictionary specification

The\_Data\_Dictionary

DataDictionary  
Init-DataDictionary  
Add  
Lookup  
Delete  
Replace  
Extract

---

---

---

---

---

---

---

---

## Key points

*Model-based specification* relies on building a system model using *well-understood mathematical entities*

Z specifications are made up of *mathematical model* of the *system state* and a definition of operations on that *state*

A Z specification is presented as a number of *schemas*

*Schemas may be combined to make new schemas*

---

---

---

---

---

---

---

---

## Key points

Operations are specified by defining their effect on the system state. *Operations may be specified incrementally then different schemas combined to complete the specification*

Z functions are a set of pairs where the *domain* of the function is the set of valid inputs. The *range* is the set of associated outputs. A sequence is a special type of function whose *domain* is the consecutive integers

---

---

---

---

---

---

---

---