SE 2A04 Fall 2000

08. Software Design with Modules

Instructor: W. M. Farmer

Revised: 1 December 2000

Overview

- A software product design is decomposed into a set of **modules**
- Each module:
  - Provides a set of **services** to the other modules via the **interface** of the module
  - Hides a **secret** (which is expected to change) from the other modules via the **implementation** of the module
  - Is designed, implemented, and verified **separately**

References

1. M. Mohrenschildt, M. Directo, and K. Kowalski, *Introduction to Software Engineering: A Collection of Formal Software Specifications*, McMaster Custom Courseware, SE 2A04, September 2000.

2. D. Parnas, "On the criteria to be used in decomposing systems into modules", *Communications of the ACM*, vol. 15, no. 12, December 1972.

3. D. Parnas, P. Clements, and D. Weiss, "The modular structure of complex systems", *IEEE Transactions on Software Engineering*, vol. SE-11, no. 3, March 1985.

Design Documents

1. Module Guide

2. Module Interface Specification (MIS) for each module

3. Module Internal Design (MID) for each module

Module Guide

- The Module Guide lists all the modules of the software product
- The following information is given for each module:
  1. Module name
  2. Module nickname (2 or 3 letters)
  3. Service: Short informal description of what service the module provides
  4. Secret: Short informal description of what secret the module hides

Two Good Approaches

1. For a module **with state**, the MIS and MID are specified using the **state transition model**
2. For a module **without state**, the MIS and MID are specified using the **fundamental model**

State Transition MIS

1. Imported module interfaces
2. Exported interface data types specified as abstract sets or abstract data types
3. Exported interface constants
4. Exported interface exceptions
5. State variables (with initial values)
6. Exported interface functions specified by:
   (a) Input and output types
   (b) Next state transitions
   (c) Output value productions
   (d) Exceptions produced from undefined applications

State Transition MID

1. Imported module interfaces
2. Exported
   (a) Interface data types
   (b) Interface constants
   (c) Interface functions
   (d) Interface exceptions
3. Local
   (a) Implementation data types
   (b) Implementation constants
   (c) Implementation variables (with initial values)
   (d) Implementation functions
   (e) Implementation exceptions
4. Abstraction function

State Transition Abstraction Function

- An **abstraction function** maps:
  1. MIS interface components to MID interface components
  2. MIS states (defined by MIS state variables) to MID states (defined by MID local variables)
- The MID **satisfies** the MIS via an abstraction function A if, for each MIS interface function f, MIS input i, and MIS state α, it follows that:
  1. When the application $f(i)$ is defined in state α, f and $A(f)$ yield corresponding next states and outputs:
     $A(\text{ns}(f, i, \alpha)) = \text{ns}(A(f), A(i), A(\alpha))$
     $A(\text{out}(f, i, \alpha)) = \text{out}(A(f), A(i), A(\alpha))$
  2. When the application of $f(i)$ is undefined in state α, f and $A(f)$ yield corresponding exceptions:
     $A(\text{exc}(f, i, \alpha)) = \text{exc}(A(f), A(i), A(\alpha))$

Fundamental Model MIS

Axiomatic theory composed of:

1. Imported MIS theories
2. Exported interface language
   - Types
   - Constants
   - Functions
3. Axioms specifying the interface language

Fundamental Model MID

A. Axiomatic theory composed of:

1. Imported MIS theories
2. Local implementation language
   - Types
   - Constants
   - Functions
3. Axioms specifying the implementation language

B. Interpretation of the MIS interface theory in the MID implementation theory

Module Structure

- Simple structure:
  - All module interfaces are accessible to all module implementations
  - All modules are indivisible units
- Access structure: Module interfaces are only available to certain module implementations
  - Example: Some modules may be **local** to a module implementation
- Submodule structure: Modules may be decomposed of into submodules
  - A **horizontal submodule** is a module that is a portion of a module's interface and implementation
  - A **vertical submodule** is a module that is a portion of a module's local data and procedures