A Technique for Software Module Specification with Examples

Communications of the ACM, May 1972, Volume 15 Number 5
Presented by:
Keisha Hill and Kanayo Orji

Overview

- Module Specification Technique
- A module is a device having a set of switch inputs and readout indicators, that provide a number of subroutines or functions which can cause changes in state, and other functions or procedures that can give a user program the values of variables making up that state.
- Goals of the Specification Outline:
  - Provide all information that user will need to use the program and NOTHING MORE!
  - Provide the implementer with all information about the use of the module and needed information to complete the program and NOTHING MORE
  - Must be sufficiently formal that I can conceivably be machine tested for consistency, completeness and other desirable properties of a specification
  - Should discuss the program in the terms normally used by user and implementer rather than some other area of discourse.

Impact

- B Formal Method, TROLL, UML
- TROLL – developed for specifying information systems at a high level of abstraction.
- Modules – A Module is a CLOSED static unit that encapsulates embedded abstractions, such as types, variables, constants, functions, procedures and classes.
- Information Hiding
- Construction, Validation, Maintaining and Reuse of Software
- Modularization
- Parameterization
- Refinement – allowing developers to show that program parts satisfy their specifications.
- Encapsulation
- Object Oriented Specification Languages
- Frameworks:
  - Formal methods provide a framework within which people can specify, develop, and verify systems in a systematic, rather than ad hoc, manner.
Towards a Module Concept for Object Oriented Specification Languages
S. Eckstein,
Proceedings of the Third International Baltic Workshop on Databases and Information Systems

- Concurrency and communication issues that modules can be allowed to have between each other as well as between interfaces that can be developed.
- Proposed –
  - Channels provide an asynchronous communication mechanism between modules
  - Domains that will allow easy extension and better understanding of large systems. Subsystems refer to modules that are structuring units and modules refer those modules which allow reuse.
  - OFFER and REQUEST actions use objects to allow subsystems to pose and acquire information from other subsystems.

Why does it extend Parnas?
- Parnas looked at the ways in which modules should be specified while Eckstein looked also at the concurrency and communication issues between modules and also interfaces that can be developed.
Formal Definition of UML’s Package Concept

Schurr and Winter, 1998

Overview

- UML:
  - First OO modeling language with useful modularization and information hiding concept.
  - Nesting, Import, Refinement.
  - Visibility: Public > Protected > Private
  - Problems: Ambiguity, Contradictions, Need more Constraints.

Example of Ambiguity

- “Elements made available to another package by the use of generalization have the same visibility in the her as they have in the owning package.”
- Packages P, Q, R; Element E.
- P is owner of E with Public visibility.
- Package Q imports E. Restricts visibility of reference to Protected.
- Package R references E in Q.
- Problem: Target of generalization dependency and element owner are different packages.
- Solution: Ensure that a package does not only make available its own elements by generalization, but also imported elements from other packages.
Example of Contradiction:

- "Private inheritance (generalization) hides the inherited features of a class (package)."
- Statement is made only for classes.
- Statements concerning generalization should hold for both classes and packages.
- Specification forces elements made available by a private generalization to be visible at the same level as the generalization itself.
- Contradiction: Implies that when the dependency is private, the elements made available by the generalization should also be hidden.

Technique

- Predicate Logic Formulas: Extensions to UML to solve shortcomings.
- Examples
  - The owner of any element is unique:
    Formula: \((P \text{ owns } w E) \land (P' \text{ owns } w' E) \Rightarrow (P = P') \land (w = w')\)
  - Elements from other packages may be referenced if visible. The visibility of the reference may be restricted:
    Formula: \((P \text{ references } w E) \Rightarrow (P \text{ sees } w'E) \land (w < w')\)
  - The "contains" relationship comprises "owns" and "reference" relationship:
    Formula: \((P \text{ contains } w E) \Leftrightarrow (P \text{ owns } w E) \lor (P \text{ references } w E)\)
  - Many others.
- Most important consequence:
  Package sees all public elements of contained packages with the visibility of the container itself.

Related Work

- Transform formulas into specification based on PROgrammed Graph REwritng Systems (PROGRES).
- Specification can then be used to generate UML editor.
Behavioral Interface Contracts for Java.
Findler and Felleisen, 2000
Rice University

Overview

- Programs should consist of off-the-shelf, interchangeable, black-box components.
- Components should have type signatures and contracts to ensure proper execution.
- Four levels of contracts:
  - Syntactic (Type Streams)
  - Behavioral (Conditions)
  - Synchronization
  - QoS.
- Focus on Behavioral Contracts.

Technique

- Contracts expressed as pre- and post-conditions for externally visible functions.
- Contracts specified in interfaces.
- Conditions must be validated for function calls.
- Multiple pre- and post-conditions may be specified, but only one of each must be met.
- If function fails to satisfy contract, system should blame the faulty component.
- Implemented Contract Java, a version of Java with behavioral interface contracts.
- Pre- and post-conditions can be weakened or strengthened. (Four Scenarios.)
**Strengthening Pre-Conditions**

- **Motivation:**
  Often desirable, for instance, in enforcing an input type.
- **Problem:**
  Implicit cast may cause the wrong method to be blamed.
- **Solution:**
  No solution without compromising Java compatibility.
  Pre-condition strengthening disallowed.

**Weakening Pre-Conditions**

- **Motivation:**
  Often desirable, for instance, to allow different instantiations when extending an interface.
- **Problem:**
  - Java lacks decision procedure to determine if one contract is weaker than another.
  - Cannot simultaneously provide pre-condition weakening and disallow pre-condition strengthening.
- **Solution:**
  Contract Java requires that pre-conditions be syntactically the same as their overridden counterparts.

**Strengthening Post-Conditions**

- **Motivation:**
  To provide a stronger guarantee for components using an extension to an interface.
- **Solution:**
  The guarantee is simply put in a contract.
  Easily accommodated in Contract Java.
Weakening Post-Conditions

- **Motivation:**
  Useful in extending a class, for instance, extending from just floating to complex numbers.
- **Problem:**
  Instances of an extended class can be implicitly cast to the parent class.
  (Complex numbers may be incorrectly blamed for not being floats.)
- **Solution:**
  No solution without compromising Java compatibility.
  Post-condition weakening disallowed.

---

The Structure and Value of Modularity in Software Design (Non-Citing Paper)

K. Sullivan, Y Cai, B. Hallen and W.G Griswold
Technical Report
University of Virginia
Department of Computer Science
December 2001

---

Overview

- Draws upon concept of information hiding.
- Extends it to propose a new theory based on design structure matrices (DSMs) and real options techniques.
- Basic Idea:
  - Modularity creates value in the form of *options*.
  - Improves system by experimenting with new implementations, substituting superior implementations.
Motivation

- Current modularization formulation does not address:
  - Which of the available modularizations is best?
  - At what point should one be selected in the face of uncertainty?
- We lack models for assessing the relative value of proposed modularizations.

Goal

- To lend insight into information-hiding modularity.
- To find new ways of determining the relative value of different modularizations.

Technique

- Evaluate modular structures relative to non-modular designs. Modularizations can then be evaluated against each other.
- Design Structure Matrices (DSMs)
  - Intuitive, qualitative framework for design.
  - Represent dependencies among design parameters (data structures, algorithms, type signatures, ...).
  - Rows and columns of matrix labeled by design parameters.
  - Mark in (Row B, Column A) means that choice for B depends on choice for A.
  - Interdependence: Symmetric marks in (A,B) and (B,A).
  - Cyclic Interdependence (A->B->C->A): Splitting required.
  - Independence: Design parameters that can be changed individually.
  - New design parameters added as knowledge improves.
Technique (Continued)

- Net Options Value (NOV)
  - Options give value.
  - Finance Industry: “A portfolio of options is worth more than an option on a portfolio.”
  - Example: A system with two modules has four options.
  - Used to quantify consequences of a design.
  - Flexibility modeled as a fraction of value of the base system (based on information hiding).
  - Used to quantify module complexity, design costs, visibility costs.

Related Work

- Built upon Parnas’ work extensively:
  - DSM-based analysis of KWIC.
  - NOV-based analysis of KWIC.

- Should have cited Parnas’ work on Technique to Module Specification because:
  - Their technique is based on module design parameters.
  - The purpose of DSMs and NOVs is to yield testable module specifications.

References