Activity 9-2: Architectural Specification Notations

Why?

Software architectures can be specified in many ways, but some notations are more widely used than others. This activity reviews the most widely used notations and asks you to use several of them.

Learning Objectives

- Read architectural specifications in several notations
- Create architectural specifications in several notations

Success Criteria

- Be able to indicate notations appropriate for specifying each of the DeSCRIPTR aspects in an architectural design
- Be able to read and write box-and-line diagrams
- Be able to read and write UML diagrams with packages, comments, constraints, stereotypes, and dependency relations

Resources

ISED sections 9.2 and 9.3

Vocabulary

DeSCRIPTR, syntax, semantics, pragmatics, legend, comment, constraint, stereotype, dependency relation, package

Plan

1. Review *ISED* sections 9.2 and 9.3 individually.
2. Answer the Key Questions individually, and then evaluate the answers as a team.
3. Do the Exercises as a team, and check your answers with the instructor.
4. Do the Problems (based on the Case Study) and Assessment as a team.
5. Turn in the Problems and Assessment as a team deliverable.

Key Questions

1. How can architectural component responsibilities be expressed in design?
2. What are box-and-line diagram legends for?
3. What is a dependency relation? What is an example of a dependency relation?
4. When do dependency arrows appear between package symbols?

Exercises

1. What is the difference between syntax, semantics, and pragmatics—illustrate your answer with a Java example.
2. Are box-and-line diagrams for static or dynamic modeling?

Case Study

The *Madison Design Tool* (MDT) is a CASE tool supporting software engineering design. It must support an end-to-end engineering design process that includes architectural and detailed design. It must generate a Software Architecture Document (SAD), a Detailed Design Document (DDD), and a Design Document consisting of the SAD and DDD.

MDT must run as a web service using a data repository on the client's computer. In other words, the MDT tool must reside at a website and must run in web clients' browsers or special client programs, but the data it manipulates must all reside on the clients' machines.

The format of the data in the repository must be XML.

The MDT must provide editing tools for each of the following: UML Use Case Diagrams, Use Case Descriptions, Box-and-Line Drawings, Textual Interface Specifications, UML Class Diagrams, UML Sequence Diagrams, UML State Diagrams, Operation Specifications, Textual Design Rationale, Decision Matrices, Glossary.

MDT must provide configuration control to the level of individual design models. All versions and revisions of design models and design specifications must be kept in the repository.

MDT must provide a linking mechanism between model elements to support traceability and other connections between model elements. All links must be maintained in the repository.

MDT must generate Java code from design models. Generated code will not be kept in the repository.

The MDT must never lose data saved to the repository or corrupt the repository.

The MDT must be at least as reliable as major object-oriented CASE tools currently on the market.

The MDT is an experimental tool so it must accommodate high rates of change.

The MDT may support simplified versions of notations.

The MDT may provide minimal editing, checking, and support features—in other words, solid basic functioning is much more important than lots of advanced features that do not work reliably.

Problems (Deliverable)

1. Draft an architectural design for the MDT. Express your design with a box-and-line diagram.
2. Show the major constituents in your design using UML package symbols connected with dependency arrows.
3. Choose a unit from your design and describe its responsibilities.

Assessment (Deliverable)

1. Did your team improve its performance from last week?
2. How could you improve your team's performance next time?