Model driven paradigm

- a) The system is defined as an executable specification which is an object-oriented analysis model.
- b) The system is validated at the analysis model level.
- c) A software and execution architecture is defined as a set of class templates in an object-oriented programming system.
- d) The executable system is realized by compilation of the validated analysis model to the software execution architecture.

Why

Using models to design complex systems is de rigeur in traditional engineering disciplines. No one would imagine constructing an edifice as complex as a bridge or an automobile without first constructing a variety of specialized system models. from using models and modeling techniques. Models help us understand a complex problem and its potential solutions through abstraction. Therefore, it seems obvious that software systems, which are often among the most complex engineering systems, can benefit greatly

Why

MDD’s defining characteristic is that software development’s primary focus and products are models rather than computer programs.The major advantage of this is that we express models using concepts that are much less bound to the underlying implementation technology and are much closer to the problem domain relative to most popular programming languages. This makes the models easier to specify, understand, verify and maintain.

Requirement – code generation

If models are merely documentation, they are of limited value, because documentation all too easily diverges from reality. A key premise behind MDD that programs are automatically generated from their corresponding models.

Therefore

Models must be executable.

Further

The resulting executable must be reasonably competitive wrt resource consumption.

Classes, Attributes and Relationships

Model Specification – Structure + Behavior

Structure – Classes + Relationships/Associations -> Information Model

- Classes – entities in the application
- Attributes – properties of classes
- Associations/Relationships among classes – “has_a”,”is_a”, etc.
- Constraints – quantification of relationships among attributes, etc.

Behavior – Transitions among states of entities -> State Model

- States – defined by assignments of values to attributes
- Transitions – governed by state machines
- Events – cause transitions

Analysis process

1. Define Information Model
    1. Define classes and select attributes
    2. Define associations/relationships among classes
    3. Construct associative classes
    4. Iterate steps 1,2 & 3 until consistent
2. Construct State Models
    1. Construct state model for each object with multiple states
    2. For each state model, define actions effecting state changes
3. Iterate until consistent

Development of Information Model

Methods for recognizing classes, attributes and relationships

1. Linguistic analysis of requirements specification
2. Operational analysis of system behavior (OBA, Use Cases)

Apply both methods and compare resulting information model

Linguistic Analysis Method

1. Write requirements specification
2. Generate Class Diagram (except for state models) from requirements specification
3. Construct Use Cases from requirements
4. Generate state model from Information Model and Use Cases

Information Models via Semantic Analysis

1. Identify the classes. The classes derive from noun phrases. Go through the requirements statement and pick out all of the important nouns. Categorize these nouns into tangible items or devices, physical classes, roles, interactions, instance specifications, etc.. Decide which ones are significant and which ones are redundant and select classes.
2. Define attributes for each object. Attributes come from possessive phrases, as descriptions of the nouns. Recall that the attributes define the identity and the state of an object.
3. Define relationships. Relationships can be derived by looking at the verb phrases of the requirements analysis. Verb phrases include such things as the PC board “is made up of” chips and connectors, etc..
4. Specialize and generalize classes into subtypes and supertypes and create associations, create object definitions for m to n associations or dynamic associations which fall under the heading of events, interactions which have to be remembered.

Classes candidates

Tangible entity or device – airplane
Role – professor, student
Incident or persistent event – car registration
Interaction – contract terms
Specification – sets of entities with related characteristics
Organization - university

Classes candidates

Tangible entity or device – airplane
Role – professor, student
Incident or persistent event – car registration
Interaction – contract terms
Specification – sets of entities with related characteristics
Organization - university
External systems – sensor drives

Attributes and Attribute Types

Naming – unique identifier for instances of object types.
    may be single or multiple attributes
Descriptive – intrinsic properties of classes – color, size, etc.
Referential – specifies relationships, identifies partners in “has_a” relationships
State or Status – values define current state of object.
    may be changed during lifetime of object instance.
    status – on/off, position of robot arm, etc.

Test planning