Specifying Executable Platform-Independent Models using OCL

Pierre Kelsen, Elke Pulvermueller and Christian Glodt
The Problem

• Vision of Model-Driven Architecture
  – Full code generation from models
  – Models become real assets
• Requires precise and complete description of a system using models
  – Structure and behavior of the system need to be described
  – How?
The Structure

• Static structure of the system can be conveniently described using UML
  – UML is a standard (current version 2.0)
  – Although full specification is large and complex, we only need to use a small part
  – Use class diagrams for describing the static structure
Class Diagrams

- In class diagram
  - Define initial values of properties using OCL “init” constraint
  - List as operations only query operations; body defined using OCL “body” constraints
  - Add compartment with events -> modifier operations (formal definition using UML profile)
The Behavior

• Could also use UML for behavior
• However: UML behavioral diagrams describe behavior only partially
• Option 1:
  – Augment UML behavioral diagrams with action language based on UML Action Semantics
  Drawback: action languages are imperative in style: clash with declarative style of diagrams
The Behavior (2)

• Option 2:
  – Define semantics of operation using OCL pre- and post-conditions
  
  Drawback: precise but not executable; not suitable for full code generation

• Option 3:
  – Describe behavior using another MOF-based metamodel
The EP-Language

- Recall that events in class diagram represent modifier operations.
- Event can impact (modify) a property and can have child events.
- Event may have parameters.
- Parameters of child event expressed in terms of parameters of parent event.
The EP-Language (2)

• When an Event impacts a Property, we associate a function that computes the new value with the impacts link

• With each parameter of an event and each incoming link from a parent event, associate a function that computes the value of the parameter

• These functions are side-effect free and are expressed as OCL-expressions
An Example

• Event tree in Flight Reservation System
The EP Metamodel

- Metamodel of the EP-language
Why OCL?

• We need a language with the following properties
  – Describes side-effect free functions (-> EP metamodel)
  – Platform-independent (to describe PIMs)
  – Allows object navigation
  – Turing-complete

• OCL satisfies all these requirements
Tool Support

- Advantage of OCL: wide-spread tool support, e.g., in Eclipse
- Democles tool
What’s missing?

• OCL not originally intended as a “programming language”

• Missing features
  – Missing built-in types, e.g., Date
  – Built-in types are lacking basic operations (even needed for constraints), e.g.,
    • is a String a substring of another String
    • convert String to Integer
  – Object creation
Conclusion

• OCL can be used for precise behavioral modeling
• Expresses side-effect free functions within EP-models
• OCL needs to be extended to describe realistic systems
Questions

• Questions?