CIS 771: Software Specifications

Lecture -- Week 14: Design by Contract

Exploiting Design Information

- Alloy and UML/OCL provide a means for expressing properties of designs
  - Early design refinement saves time
- Ultimately, we want this effort to impact the quality of implementations
- How can we transition design information to the code?
  - State information (multiplicities, invariants, ...)
  - Operations info (pre, post, frame conditions, ...)

Design By Contract...

- is a method that emphasizes the precise description of interface semantics
  - not just syntax, e.g., signatures
  - but behavior, e.g., effects of a method call
- It is supported by tools that
  - allow semantic properties of the design to be propagated to the code
  - support various forms of validation of those properties

Basic Idea

- Software is viewed as
  - a system of communicating components
  - all interaction is governed by contracts
  - contracts are precise specifications of mutual obligation
- Note that contracts are bi-directional
  - both parties are obligated by them

Specifications are Necessary

- but not sufficient for software quality
- The Law of Excluded Miracles

  Without a specification we have no hope of a system that works properly
- Content of specification varies
  - Lightweight (partial)
  - Complete behavioral specification
- Tool support varies
  - Commercial/research, Static/dynamic checking

Contracts

- Two parties are involved in a contract
  - The supplier performs a task
  - The client requests that the task be performed
- Each party
  - has obligations
  - receives some benefits
- Contracts specify those obligations and benefits

Air Travel

Client (Traveler)

- Obligation
  - check in 10 minutes before boarding
  - <3 small carry-ons
  - buy ticket
- Benefit
  - reach Boston

Supplier (Airline)

- Obligation
  - take traveler to Boston
- Benefit
  - don't need to wait for late travelers
  - don't need to store arbitrary amounts of luggage
  - money

Contracts

- Specify what should be done
  - they are implementation independent
- This same idea can be applied to software using the building blocks we've learned
  - Pre-conditions
  - Post-conditions
  - Frame-conditions
  - Invariants

Taking a flight

```
Class Flight {
  /**
    * @pre time < this.takeoff – 10
    * @pre l.number < 3
    * @pre p in this.ticketed
    * @post result = this.destination
    */
  Destination takeFlight(Person p, Luggage l) {...}
}
```

Specification or Coding Language

- Why not both?
- Refinement methodology
  - rather than develop signatures alone
  - develop contract specification
  - analyze client-supplier consistency
  - fill in implementation details
  - check that code satisfies contract
- Natural progression from design to code

Java Example

```
import java.util.Vector;

public interface ICompany {
  public Vector getEmployees();
  public Vector getRooms();
  public void hire(IEmployee employee);
  public void move(IEmployee employee, IRoom newOffice);
  public boolean roomsAvailable();
}
```

Java Example

```
import java.util.Vector;
public interface ICompany {
  public Vector getEmployees();
  public Vector getRooms();

/**
 * @pre employee != null
 * @pre !getEmployees().contains(employee) // do not employ twice
 * @pre !employee.hasOffice() // does not own an office somewhere else
 * @pre roomsAvailable() // there must be an office left
 *
 * @post getEmployees().contains(employee) // added to list of employees
 * @post getRooms().contains(employee.getOffice()) // assign one of our offices
 * @post employee.hasOffice() // office assigned
 * @post employee.getOffice().getOwner() == employee // correct office owner?
 */
  public void hire(IEmployee employee);
  public void move(IEmployee employee, IRoom newOffice);
  public boolean roomsAvailable();
}
```

Source Specifications

- Pre/post conditions
  - Boolean expressions in the host language
- What about all of the expressive power we have in, e.g., OCL?
  - Balance power against checkability
  - Balance abstractness against language mapping
- No one right choice
  - Different tools take different approaches

Java Example with OCL

```
import java.util.Vector;
/**
 * Each employee gets a single office (uniqueness constraint)
 * @invariant forall IEmployee e1 in getEmployees().elements() |
 *                  forall IEmployee e2 in getEmployees().elements() |
 *                     (e1 != e2) implies e1.getOffice() != e2.getOffice()
 */
public interface ICompany {
  public Vector getEmployees();
  public Vector getRooms();
  public void hire(IEmployee employee);
  public void move(IEmployee employee, IRoom newOffice);
  public boolean roomsAvailable();
}
```

Mapping OCL

- The OCL iterate operation
  - Properties should be independent of order
  - So, any order will do
- Variants of iterate can be mapped to fragments of code for classes with java.util.Enumerations

```
@invariant forall C c in o.elements() | P(c)

boolean result = true;
for (Enumeration e = o.elements();
     e.hasMoreElements() && result; ) {
   c = (C)e.nextElement();
   result = P(c);
}
```

For You To Do (pause here)

- How could you express the “exists” quantifier in OCL as a fragment of code in the style we just looked at?
- How about “select” or “isUnique”?

Important Issues

- Contract enforcement code is executed
  - It should be side-effect free
  - If not, then contracts change behavior!
- Frame conditions
  - Explicitly mention what can change
  - Anything can change
- Failed contract conditions
  - Most approaches will abort the execution
  - How can we continue?

Contract Inheritence

- Inheritence in most OO languages
  - Sub-type can be used in place of super-type
  - Sub-type provides at least the capability of super-type
- Sub-types weaken the pre-condition
  - Require no more than the super-type
  - Implicit or of inherited pre-conditions
- Sub-types strengthen the post-condition
  - Guarantee at least as much the super-type
  - Implicit and of inherited post-conditions
  - Invariants are treated the same as post-conditions

Tool Support

For dynamic contract enforcement

- Parasoft's Jtest (Jcontract)
  - www.parasoft.com
- ReliableSystems iContract
  - Free, but with lots of support tools
- Java dynamic proxies and assertions
  - Easy to build your own framework
  - See JavaWorld Feb. 2002 issue
- Jass, JMSassert, ...