USAGE CONTRACTS

KIM MENS
UNIVERSITÉ CATHOLIQUE DE LOUVAIN (UCL)

JOINT WORK WITH ANGELA LOZANO & ANDY KELLENS

SOME OF MY RESEARCH INTERESTS

Programming languages
- Context-Oriented Programming
- Language interoperability between logic and OO
  (Aspect-oriented programming ™)

Tool support for software development, maintenance and evolution
- source code mining
- source-code based recommendation tools
- structural source-code regularities (e.g. usage contracts)
Often you find code comments like

```java
/**
 * Deactivates the tool. This method is called whenever the user switches to another tool
 * Use this method to do some clean-up when the tool is switched.
 * Subclassers should always call super.deactivate.
 * An inactive tool should never be deactivated.
 */

public void deactivate() {
    if (isActive()) {
        if (getActiveView() != null) {
            getActiveView().setCursor(new AWTCursor(java.awt.Cursor.DEFAULT_CURSOR));
        }
        getEventDispatcher().fireToolDeactivatedEvent();
    }
}
```
Often you find code comments like

```
/**
 * Deactivates the tool. This method is called whenever the user switches to another tool
 * Use this method to do some clean-up when the tool is switched.
 * Subclassers should always call super.deactivate.
 * An inactive tool should never be deactivated.
 */
public void deactivate() {
    if (isActive()) {
        if (getActiveView() != null) {
            getActiveView().setCursor(new AWTCursor(java.awt.Cursor.DEFAULT_CURSOR));
        }
        getEventDispatcher().fireToolDeactivatedEvent();
    }
}
```

We studied JHotDraw for occurrences of “should, may, must, can(not), could, ought, have, has, need, require, ....” and found 22 structural regularities like:

<table>
<thead>
<tr>
<th>subclassers of this class</th>
<th>should</th>
<th>call ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>this class</td>
<td>should not</td>
<td>do a supercall</td>
</tr>
<tr>
<td>...</td>
<td>must</td>
<td>implement ...</td>
</tr>
<tr>
<td>methods in this class</td>
<td>should (not)</td>
<td>override</td>
</tr>
<tr>
<td>this method</td>
<td>...</td>
<td>only be called by ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>only be called internally</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be called after ...</td>
</tr>
</tbody>
</table>
We want a tool that allows encoding such regularities and offering immediate feedback on violations of such structural source-code regularities.

The tool should be proactive (violations reported ‘on the fly’ during coding).

The tool should be “developer-friendly” (like unit testing but for usage expectations):

- desired regularities expressed in the same programming language
- tight integration with the integrated development environment
- not coercive
METAPHOR

Provider

uses

Consumer

Usage Contract

describes expectations of

should comply with

expects

complied

should
describes
EXAMPLE

**copyFrom:** anEntity **within:** aVisitor

- inherits from

All overrides of **copyFrom:** **within:** should start with a super call

**copyFrom:** anEntity **within:** aVisitor

**super** **copyFrom:** anEntity **within:** aVisitor

...
EXAMPLE

copyFrom: anEntity within: aVisitor

All overrides of copyFrom:within: should start with a super call

describes expectations of

should comply with

copyFrom: anEntity within: aVisitor

super copyFrom: anEntity within: aVisitor

...
UCONTRACTS: THE LANGUAGE

Liable classes
- **liableClass**: regExp / **exceptClass**: regExp
- **liableHierarchy**: className / **exceptHierarchy**: className
- **liablePackage**: regExp / **exceptPackage**: regExp

Liable methods
- **selector**: regExp / **exceptSelector**: regExp
- **protocol**: regExp / **exceptProtocol**: regExp
- **exceptClass**: className **selector**: selector

Contract terms
- **require**: condition
- **suggest**: condition
- **require**: condition **if**: anotherCondition
- **suggest**: condition **if**: anotherCondition

Contract conditions
- **assigns**: regExp
- **calls**: regExp
- **references**: regExp
- **returns**: expression
- **doesSuperSend**: regExp
- **doesSelfSend**: regExp
- **inProtocol**: regExp
- **isOverridden**: selector
- **isImplemented**: selector
- **custom**: visitor
UCONTRACTS: THE TOOL

Contracts (3)
ECAbstractElementContract (3)
must be in protocol visiting (1)
must call method named do* (1)
must implement a double dispatch

Code Critics: All checks on eContracts-Examples-Visitor (24 problems)

Contracts (3)
ECAbstractElementContract (3)
must be in protocol visiting (1)
must call method named do* (1)
must implement a double dispatch
VALIDATION ON AN INDUSTRIAL CASE

- An interactive web application for event & resource planning
- developed in Pharo Smalltalk
- uses the Seaside web development framework.

- Medium-sized
- Packages: 45
- Classes: 827
- Methods: 11777
- LOCs: 94151
INDUSTRIAL VALIDATION: SET-UP OF THE EXPERIMENT

- Qualitative assessment

- Ideally we would have liked the tool to be used directly by the developers, but instead we had to perform an offline experiment.

- Together with the developers, during 2 days we defined 13 contracts documenting important regularities in their framework.

- We checked all contracts in December and reported all contract breaches to the developers.

- 3 months later, we reverified compliance of the code against the same contracts.
INDUSTRIAL VALIDATION: ABOUT THE CONTRACTS

- contracts related to the model of the web application
  - for 3/5 of them violations were found
  - 214 liable classes, 88 violations
- contracts related to the classes dealing with persistency
  - for 2/2 of them violations were found
  - 75 liable classes, 2 violations found
- contracts about how the UI is constructed with the Seaside framework
  - for 4/6 of them violations were found
  - 598 liable classes, 8 violations found
Private methods should not be called directly

```
interfaceCode

<package: 'App-*>'>
<exceptPackage: 'App-Model*'>
<exceptPackage: 'App-Database*'>

noCallsToPrivate

<selector: '*'>

contract require:
  (condition not: (condition calls: 'private*'))
```
In domain classes, state changes must mark model objects as dirty so that they can be re-rendered.

domainClasses

< hierarchy: #AppDomainObject >

dirtyFlag

< selector: '*' >

contract

require: ( condition calls: # markAsChanged: )

if: ( condition assigns: '*' )
INDUSTRIAL VALIDATION: EXAMPLE OF A CONTRACT

Overridden initialisation methods should start with a super call (and be put in an appropriate protocol)

```persistentDomainClasses
<hierarchy:#{AppPersistentDomainObject}>

initializationOfDatabase
  <selector:#{initializeWithDatabase:}>
  contract
    require:
      (condition beginsWith:(condition doesSuperSend))
    if: (condition isOverride).
    contract suggest:
      (condition methodInProtocol: 'initialize-release')
```
Certain messages need to be sent at the end of a method cascade.

```xml
interfaceCode

   <package: ’App-*’>
   <exceptPackage: ’App-Model*’>
   <exceptPackage: ’App-Database*’>

withShouldBeTheLastMessageInACascade
   <selector: ’render*’>

contract
   require:
   (condition not:
      (condition
         custom: WithInCascadeVisitor
description: ’With: should be last’)))
```
Certain messages need to be sent at the end of a method cascade. For example:

```solidity
class WithInCascadeVisitor extends CustomConditionVisitor {
  interfaceCode

  acceptCascadeNode: aNode
    super acceptCascadeNode: aNode.
    (aNode messages allButLast
      anySatisfy: [:msg | msg selector = #with:] )
    ifTrue: [self match: aNode]

    (condition not:
      condition
      custom: WithInCascadeVisitor
      description: 'With: should be last')
}
```
## INDUSTRIAL VALIDATION: RESULTS

<table>
<thead>
<tr>
<th>Contract</th>
<th>Liable Methods</th>
<th>Exceptions</th>
<th>Errors December</th>
<th>Errors March</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private methods should not be called directly</td>
<td>7410</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Marking dirty objects</td>
<td>333</td>
<td>5</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Initialisation methods should start with super</td>
<td>44</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Call ordering within method cascade</td>
<td>531</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
UCONTRACTS : CONCLUSION

- uContracts offer a simple unit-testing like way for letting programmers document and check conformance to structural source-code regularities using a “contract” metaphor
- focus on immediate feedback during development
- embedded DSL close to the programming language
- tight integration with the IDE

FUTURE WORK

• More validation

• Improve / extend the DSL

• Port to most recent version of Pharo

• uContracts for other languages (e.g., Ruby)