Ensuring Consistency between Classifiers and Classes

Joe Kiniry
IT University of Copenhagen
22 September 2010
AVOCS 2010
Our Challenge

• How can we keep non-code development artifacts synchronized with source code?

• requirements and feature descriptions (Microsoft Word or text documents)

• architecture descriptions (Viseo drawings, SVG or text documents)

• design artifacts (UML documents as encoded by XML files)

• specifications (structured English + pragmas)
Current State of Affairs

• requirements are written by customers
• features are written by marketing
• analysis and design documents are written by “architects”
• specifications are written by programmers
• tests are written by Q/A and programmers

• in general, there is no consistency check!
Our Solution: Refinement using Type-logical Grammars
### Informal EBON

```plaintext
class_chart LOGICAL_CLOCK

explanation

A logical clock's query

What is the current time of this clock?

command

Advance the clock; update the clock's time.

constraint

The time must be non-negative.

Must support concurrent use by multiple clients.

end
```

### Formal EBON

#### Feature

- `my_time`: INTEGER -- The current time of this clock.
  
  -- What is the current time of this clock?
  
  deferred `get_logical_time`: INTEGER
  
  -- concurrency: CONCURRENT
  
  -- modifies: QUERY
  
  ensure
  
  Result = my_time;
  
  end

- `advance`: -- Advance this clock's time.
  
  -- concurrency: GUARDED
  
  -- modifies: my_time
  
  ensure
  
  -- This clock's time has monotonically increased.
  
  old my_time < my_time;
  
  end

#### Invariant

0 <= my_time;

end -- class LOGICAL_CLOCK

end -- component

### JML

```plaintext
** A logical clock

```java
** * A logical clock implementation. *

* Author "Joseph Kiniry"

public class LogicalClockImpl implements LogicalClock {

    /** The current logical time. */
    private long my_time = 0; // my_time;

    public long getLogicalTime() {
        return my_time;
    }

    public void advance() {
        my_time++;
    }

}
```
Example Refinement

class_chart ALARM
explanation
   "An alarm."
query
   "Is this alarm on?"
command
   "Turn this alarm on.",
   "Turn this alarm off."
constraint
   "The alarm is either on or off."
end
An alarm.

```
indexed
  about: "An alarm.";
static_diagram
component
defered class ALARM
feature
defered on -- Turn this alarm on.
  ensure
    is_on();
end
defered off -- Turn this alarm off.
  ensure
    not is_on();
end
defered is_on: BOOLEAN -- Is this alarm on?
invariant
  on xor off -- The alarm is either on or off.
end
end
```
/**
 * An alarm that is either on or off.
 * ...
 */
public interface AlarmInterface {
  /**
   * Turn this alarm on.
   * @ensures isOn();
   */
  public void on();

  /**
   * Turn this alarm off.
   * @ensures !isOn();
   */
  public void off();

  /**
   * @returns Is this alarm on?
   */
  public/*@ pure @*/ boolean isOn();
}
Type-logical Semantics

- Bob Carpenter (while at CMU in early 90s)
  - *lexical semantics* characterizing the meaning of expressions
  - *compositional semantics* explains the meanings of arbitrarily complex linguistic expressions in terms of the meaning of their subexpressions and the manner in which they are combined
Logical Foundations

• simple grammars in lambda-calculus and higher-order (modal) logic
• simple sentences in English with argument and adjunct structure
• categorical grammar based upon type-theory and its associated logic
• coordination, quantifier scope, anaphora, unbounded dependency constructions
The Grammatical Framework (GF)

- Aarne Ranta’s group in Gothenburg, Sweden
- FLOSS special-purpose functional language and NLP framework for writing grammars
- interfaces with Haskell, Java, JavaScript, etc.
- encode categorical grammars a la Carpenter
- is a logical framework a la PVS/Coq/Isabelle
- has been used to translate: mathematical proofs into English, formal specifications into English, natural languages, etc.
GF Language Features

• static type checking
• higher-order functions
• dependent types
• pattern matching with data constructors and regular expressions
• module system with multiple inheritance and parameterized modules
Categories in GF
Categories in GF
CFGs in GF

Pred. S ::= NP VP;
Compl. VP ::= V2 NP ;
John. NP ::= "John" ;
Mary. NP ::= "Mary" ;
Love. V2 ::= "loves" ;

John loves Mary

Returns the natural logarithm (base e) of a double value.
Separating Abstract and Concrete Syntax

- each rule is converted to two **judgements**
- **fun**, declaring a **syntactic function**
- **lin**, giving its **linearization rule**

Pred. \( S ::= NP \ VP \implies \) fun Pred : NP \( \rightarrow \) VP \( \rightarrow \) S
lin Pred np vp = np ++ vp
GF Grammars

- grammars are divided into two **modules**

  - **abstract** module (cat and fun judgements)
  - **concrete** module (lincat and lin judgements)

<table>
<thead>
<tr>
<th>Judgement</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat C</td>
<td>C is a category</td>
</tr>
<tr>
<td>fun f: T</td>
<td>f is a function of type T</td>
</tr>
<tr>
<td>lincat C = L</td>
<td>C has linearization type L</td>
</tr>
<tr>
<td>lin f xs = t</td>
<td>f xs has linearization t</td>
</tr>
</tbody>
</table>
Execution Strategy

- develop an abstract grammar for software engineering artifacts
- categories for system, cluster, class, description, explanation, query, command, constraint, and other semantic properties
- create concrete grammars for each concrete syntax
- restricted English found in documentation
- EBON CFG
- define type refinements between categorical types
Kind Theory

• paraconsistent, autoepistemic, categorical logic of reuse

• used to describe relationships between reusable artifacts and their instances

• informal description ⇝ formal model-based specification ⇝ BISL ⇝ source ⇝ object code

• informal EBON ⇝ formal EBON ⇝ JML ⇝ Java source code ⇝ Java bytecode
Commuting Diagrams

if

• U is a part of V and

• V interprets to W

then

• U interprets to P

• P is a part of W

• an interpretation of “part of” exists

\[
\begin{align*}
U \xrightarrow{\subseteq_p} V \\
\downarrow \quad \quad \quad \quad \downarrow \\
\sim \quad \quad \quad \quad \sim \\
P \xrightarrow{\subseteq_p} W
\end{align*}
\]
Feature Refinement

(query) “Current time?” $\subseteq_p$ (BON informal class) LOGICAL_CLOCK

(feature) my_time: INTEGER $\subseteq_p$ deferred BON class LOGICAL_CLOCK

(pure method) long getLogicalTime() $\subseteq_p$ Java interface LogicalClock
In Action

class_chart ALARM
explanation
  "An alarm."
query
  "Is this alarm on?"
command
  "Turn this alarm on.",
  "Turn this alarm off."
constraint
  "The alarm is either on or off."
end
class_chart ALARM
explanation
"An alarm."
query
"Is this alarm on?"
command
"Turn this alarm on.",
"Turn this alarm off."
constraint
"The alarm is either on or off."
end
The BON Tool Suite

- **BONc**: compiler-like framework for EBON (parser, type-checker, doc generation, etc.)
- **Beetlz**: refinement checker and generator between EBON and JML (fully round-trip)
- **BON Eclipse perspective**: (integrates graphical and textual view of EBON)
- **GF-based refinement**: (refinement checker and generator between informal and formal EBON)
Next Steps

• mechanical formalization of refinement in HOL
• integration of type-logical grammar work
• verification of refinement implementation
• extended static checking for English
• integration with WordNet and OpenCyc
• semantic types via JML contracts
Thanks

• Fintan Fairmichael (BONc)
• Eva Darulova (Beetlz)
• Aidan Morrissey (GF refinement)
• GF team (Krasimir Angelov, Björn Bringert, Aarne Ranta, et al.)
• BON (Kim Waldén, Jean-Marc Nerson, Richard Paige, Jonathan Ostroff)