Structuring Documentation to Support State Search
A Laboratory Experiment about Protocol Programming

Joshua Sunshine
Jim Herbsleb
Jonathan Aldrich
/
 * Precondition: `conn` is connected to a valid URL.
 * Postcondition: Return up to the first 1000 characters of the data
 * at the URL to which `conn` is connected
 */

```java
public static String readInput(URLConnection conn) throws IOException {
    InputStreamReader reader = new InputStreamReader(conn.getInputStream());
    char[] urlChars = new char[1000];
    int numBytes = reader.read(urlChars, 0, 1000);
    return new String(urlChars, 0, numBytes);
}
```
Example URLConnection task

/*
 * Precondition: `conn` is connected to a valid URL.
 * Postcondition: Return up to the first 1000 characters of the data
 * at the URL to which `conn` is connected
 */

public static String readInput(URLConnection conn) throws IOException {
    InputStreamReader reader = new InputStreamReader(conn.getInputStream());
    char[] urlChars = new char[1000];
    int numBytes = reader.read(urlChars, 0, 1000);
    return new String(urlChars, 0, numBytes);
}

✗ Cannot read from connection, doInput is false.

Programmer given test to reveal this bug
public static String readInput(URLConnection conn) {

    InputStreamReader reader = new ... 
    char[] urlChars = new char[1000];
    int numBytes = reader.read(urlChars, 0, 1000);
    return new String(urlChars, 0, numBytes);
}

Example URLConnection task

```java
class Example {
    public static String readInput(URLConnection conn) {
        conn.setDoInput(true);
        try {
            InputStreamReader reader = new InputStreamReader(conn.getInputStream());
            char[] urlChars = new char[1000];
            int numBytes = reader.read(urlChars, 0, 1000);
            return new String(urlChars, 0, numBytes);
        } catch (IOException e) {
            System.out.println(e.getMessage());
            return null;
        }
    }
}
```
Example URLConnection task

```java
public static String readInput(URLConnection conn) {
    conn.setDoInput(true);  
    InputStreamReader reader = new . . .
    char[] urlChars = new char[1000];
    int numBytes = reader.read(urlChars, 0, 1000);
    return new String(urlChars, 0, numBytes);
}
```

- **IllegalStateException**: Already connected
Example URLConnection task

public static String readInput(URLConnection conn) {
    conn.setDoInput(true);  // X
    InputStreamReader reader = new ... char[] urlChars = new char[1000];
    int numBytes = reader.read(urlChars, 0, 1000);
    return new String(urlChars, 0, numBytes);
}

✗ IllegalStateException: Already connected

Method: setDoInput
Throws:
IllegalStateException - if already connected
public static String readInput(URLConnection conn) {
    conn.setDoInput(true);
    InputStreamReader reader = new ...;
    char[] urlChars = new char[1000];
    int numBytes = reader.read(urlChars, 0, 1000);
    return new String(urlChars, 0, numBytes);
}

Method: connect
URLConnection objects go through two phases: first they are created, then they are connected. After being created, and before being connected, various options can be specified (e.g., doInput and UseCaches). After connecting, it is an error to try to set them.
public static String readInput(URLConnection conn) {
    URLConnection c2 = conn.getURL().openConnection();
    c2.setDoInput(true);
    c2.connect();
    InputStreamReader reader = new InputStreamReader(c2.getInputStream());
    char[] urlChars = new char[1000];
    int numBytes = reader.read(urlChars, 0, 1000);
    return new String(urlChars, 0, numBytes);
}
public static String readInput(URLConnection conn) {
    UrlConnection c2 = conn.getURL().openConnection();
c2.setDoInput(true);
c2.connect();
InputStreamReader reader = new InputStreamReader(c2.getInputStream());
char[] urlChars = new char[1000];
int numBytes = reader.read(urlChars, 0, 1000);
return new String(urlChars, 0, numBytes);
}
To use:
1) create an instance
2) set request parameters
3) connect
4) access header fields and remote object content

URLConnection has a protocol

**state machine** URLConnection

- connect
- getContent
- getContentEncoding
- getContentLength
- getContentLengthLong
- contentType
- getDate
- getExpiration
- getHeaderField
- getHeaderFieldDate
- getHeaderFieldInt
- getHeaderFieldKey
- getHeaderFieldLong
- getHeaderFields
- getInputStream
- getOutputStream

Disconnected ➔ Connected
Protocols are common and hard

- Protocols are common [Beckman 2011]:
  - 3x as many Java stdlib. classes define protocols as use type parameters
- Libraries with protocols can be difficult to use [Jaspan 2011]
  - ¾ of ASP.NET forum posts involved temporal constraints
  - StackOverflow search results (November 2013):

<table>
<thead>
<tr>
<th>Rank</th>
<th>Exception type</th>
<th># of results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NullPointerException</td>
<td>16,879</td>
</tr>
<tr>
<td>2</td>
<td>RuntimeException</td>
<td>13,756</td>
</tr>
<tr>
<td>3</td>
<td>ClassNotFoundException</td>
<td>7,654</td>
</tr>
<tr>
<td>4</td>
<td>IllegalStateException</td>
<td>5,890</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>IndexOutOfBoundsException</td>
<td>856</td>
</tr>
<tr>
<td>12</td>
<td>UnsupportedOperationException</td>
<td>818</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>— Total for all 27 java.lang exception types</td>
<td>63,933</td>
</tr>
</tbody>
</table>
Protocol programmer information needs

Programmer time was dominated by four categories of state search:

<table>
<thead>
<tr>
<th>Category</th>
<th>Example instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>What abstract state is the object in? Is the TimerTask scheduled? Is [the ResultSet] x scannable?</td>
</tr>
<tr>
<td>B</td>
<td>What are the capabilities of object in state X? Can I schedule a scheduled TimerTask? What can I do on the insert row?</td>
</tr>
<tr>
<td>C</td>
<td>In what state(s) can I do operation Z? When can I call doInput? Which ResultSets can I update?</td>
</tr>
<tr>
<td>D</td>
<td>How do I transition from state X to state Y? How do I move from the insert row to the current row? Which method schedules a TimerTask?</td>
</tr>
</tbody>
</table>

[Sunshine 2013]
Plaiddoc

• Documentation
  – Javadoc
    Generated from annotated Java source

URLConnection
  addRequestProperty
  connect
  getDoInput
  setDoInput
  getContent
PlaidDoc – methods by state

- Documentation:
  - Javadoc
  - PlaidDoc

- PlaidDoc generated from:
  - Lightweight manually-created state specifications
  - Java code

```
URLConnection
addRequestProperty
connect
getDoInput
setDoInput
getContent
```

```
URLConnection
getDoInput
Disconnected
addRequestProperty
connect
setDoInput
Connect
getContent
```
PlaidDoc – transitions

<table>
<thead>
<tr>
<th>URLConnection</th>
<th>Ret Type</th>
<th>Method Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>URLConnection</td>
<td>boolean</td>
<td>getDoInput</td>
</tr>
<tr>
<td></td>
<td>void</td>
<td>addRequestProperty</td>
</tr>
<tr>
<td></td>
<td>void</td>
<td>connect</td>
</tr>
<tr>
<td>Connected</td>
<td>void</td>
<td>setDoInput</td>
</tr>
<tr>
<td></td>
<td>Object</td>
<td>getContent</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State Name</th>
<th>Postcondition</th>
<th>Ret Type</th>
<th>Method Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>URLConnection</td>
<td>--</td>
<td>boolean</td>
<td>getDoInput</td>
</tr>
<tr>
<td>Disconnected</td>
<td>--</td>
<td>void</td>
<td>addRequestProperty</td>
</tr>
<tr>
<td>Connected</td>
<td>--</td>
<td>void</td>
<td>connect</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>void</td>
<td>setDoInput</td>
</tr>
<tr>
<td>Connected</td>
<td>--</td>
<td>Object</td>
<td>getContent</td>
</tr>
</tbody>
</table>
Plaiiddoc – transitions

Plaiiddoc also generates: Preconditions and ASCII state diagrams

See paper for details

<table>
<thead>
<tr>
<th>State Name</th>
<th>URLConnection</th>
<th>Ret Type</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disconnected</td>
<td>URLConnection</td>
<td>boolean</td>
<td>getDoInput</td>
<td>Method to check if the connection is in input mode.</td>
</tr>
<tr>
<td>Connected</td>
<td>URLConnection</td>
<td>void</td>
<td>addRequestProperty</td>
<td>Method to add a request property.</td>
</tr>
<tr>
<td>Connected</td>
<td>URLConnection</td>
<td>void</td>
<td>connect</td>
<td>Method to establish a connection.</td>
</tr>
<tr>
<td>Connected</td>
<td>URLConnection</td>
<td>void</td>
<td>setDoInput</td>
<td>Method to set the connection to input mode.</td>
</tr>
<tr>
<td>Connected</td>
<td>URLConnection</td>
<td>Object</td>
<td>getContent</td>
<td>Method to get the content of the connection.</td>
</tr>
</tbody>
</table>
• State search tasks
  – 3 JSL APIs (URLConnection, ResultSet, Timer)
  – Each task asked participants to answer instances of 4 state search questions

• Primary comparison: Plaiddoc vs. Javadoc

• Primary output variables:
  – Task completion time
  – Correctness
Experiment - research questions

• RQ1 Can programmers answer state search questions more efficiently using Plaiddoc than Javadoc?
• RQ2 Are programmers as effective answering non-state questions using Plaiddoc as they are with Javadoc?
• RQ3 Will programmers who use Plaiddoc answer state search questions more correctly than programmers who use Javadoc?
• RQ4 Will programmers get better at answering state search questions as they get more practice?
Experiment - background

- Search tasks can be much easier with diagrams than unstructured text [Larkin 1987]
- Demonstrations of API use improvements:
  - eMoose directives significantly improved task completion times [Dekel 2009]
  - MAPO example search significantly reduced errors [Zhong 2009]
Experiment – methodology

- 20 undergraduate and masters students
- Two by two design:
  - Plaiddoc vs. Javadoc
  - Two task orderings: Timer batch first vs. URLConnection batch first
- Between subjects: five participants in each of the four conditions
- All participants trained in all interventions
- 21 tasks – 16 state-search and and five non-state search tasks
Results – time on task

- Time between end of question and final answer

- State task mean:
  - PlaidDoc: 10.3 min
  - Javadoc: 22.4 min
  - p<0.001

- Non-state mean:
  - PlaidDoc: 5.77 min
  - Javadoc: 5.95 min
  - p=0.802
Results - correctness

• Participants confirmed “final answers”
• Sometimes wrong
  – Experiment proceeded regardless of correctness
  – Participants were not told if they were correct

<table>
<thead>
<tr>
<th>Category</th>
<th>Plaiddoc</th>
<th>Javadoc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct</td>
<td>151</td>
<td>143</td>
</tr>
<tr>
<td>Incorrect</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>Timed-out</td>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>

• Contingency table analysis, p=0.002
Incorrect responses

• Method does not exist
  – How do I transition the ResultSet from the ForwardOnly to the Scrollable state?
  – Question resulted in all 7 timeouts and 6 wrong answers

• Tripped up by abnormal mode of use

• Used heuristics to digest a lot of text, sometimes these heuristics failed
Results – learning

- Ratio of first task batch completion time to last batch
  - PlaidDoc means:
    - Timer first: 1.5
    - UrlCon first: 0.74
    - p = 0.03
  - Javadoc means:
    - Timer first: 1.07
    - UrlCon first: 0.95
    - p = 0.69
- ANOVA - marginally significant interaction between order and doc type (p = 0.08)
Conclusions

• Plaiddoc is effective and usage barrier is small
  – Training was quick and required no specialized knowledge
  – 1-3 annotations per method

• Lightweight type annotations can
  – provide benefits as documentation alone
  – even for well-documented APIs
The End

Any Questions?