

Compiler and Tool Support for Debugging Object Protocols

Sergey Butkevich, Loyola U Chicago

Marco Renedo, U of Chicago

Gerald Baumgartner, Ohio State U

Michal Young, U of Oregon

Example:

java.util.zip.ZipOutputStream

```
public class ZipOutputStream extends DeflaterOutputStream {
    public ZipOutputStream(OutputStream out);
    public static final int DEFLATED;
    public static final int STORED;
    public void close() throw IOException;
    public void closeEntry() throw IOException;
    public void finish () throws IOException;
    public void putNextEntry(ZipEntry e) throws IOException;
    public void setComment(String comment);
    public void setLevel(int level);
    public void setMethod(int method);
    public synchronized void write(byte[] b, int off, int len)
        throws IOException;
}
```

Example:

java.util.zip.ZipOutputStream

This class is a subclass of DeflaterOutputStream that writes data in API file format to an output stream. Before writing any data to the ZipOutputStream, you must begin an entry within the ZIP file with putNextEntry(). The ZipEntry object passed to this method should specify at least a name for the entry. Once you have begun an entry with putNextEntry(), you can write the contents of that entry with the write() methods. When you reach the end of an entry, you can begin a new one by calling putNextEntry() again, or you can close the current entry with closeEntry(), or you can close the stream itself with close().

Before beginning an entry with putNextEntry(), you can set the compression method and level with setMethod() and setLevel(). The constants DEFLATED and STORED are the two legal values for setMethod(). If you use STORED, the entry is stored in the ZIP file without any compression. If you use DEFLATED, you can also specify the compression speed/strength tradeoff by passing a number from 1 to 9 to setLevel(), where 9 gives the strongest and slowest level of compression. You can also use the constants Deflater.BEST_SPEED, Deflater.BEST_COMPRESSION, and Deflater.DEFAULT_COMPRESSION with the setLevel() method.

Reference

D. Flanagan, *Java in a Nutshell*. O'Reilly & Associates, 1997.

Protocol Description as Regular Expression

```
((setMethod | setLevel)*,  
putNextEntry,  
write*,  
closeEntry?,  
)*,  
close;
```

Related Work

- **Outside programming language**
 - ADLs, e.g., Wright or Darwin
 - UML StateCharts
- **Multiple object collaboration**
 - Path expressions
 - Puntigam's Ada extension
- **Type-theoretic foundation**
 - Nierstrasz' active object types

What's New?

- **Protocols as part of class/interface declarations in Java**
- **Protocol conformance check as part of subtype relationship**
- **Static conformance check refined by dynamic check**

Protocol Declarations

```
protocol {  
    ((setMethod | setLevel)*,  
    putNextEntry,  
    write*,  
    closeEntry?,  
    )*,  
    close;  
}
```

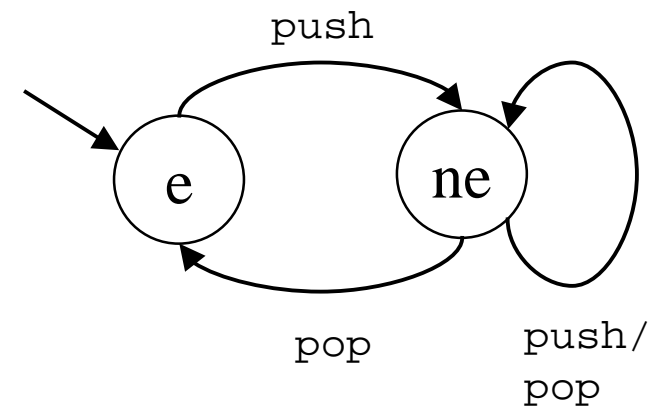
Protocols in Classes/Interfaces

```
interface ReadOnly {  
    protocol { open, read*, close; }  
    // ...  
}
```

```
class File implements ReadOnly {  
    protocol { open, (read|write)*, close; }  
    // ...  
}
```

Explicit State Declarations

```
interface Stack {  
  protocol {  
    start final state e;  
    final state ne;  
  
    <*> push <ne>;  
    <ne> pop <*>;  
  }  
  // ...  
}
```



Example:

java.util.zip.ZipOutputStream

```
protocol {
    start state DEFLATED;
    state STORED;
    final state DONE;
    <DEFLATED>
        putNextEntry, write*, closeEntry?
        <DEFLATED>;
    <STORED>
        putNextEntry, write*, closeEntry?
        <STORED>;
    <DEFLATED,STORED> setMethod <DEFLATED,STORED>;
    <STORED> setLevel <STORED>;
    <DEFLATED,STORED> close <DONE>;
}
```

Properties Associated with States

```
interface Stack {
  protocol {
    start final state e = isEmpty();
    final state ne = ! isEmpty();
    <*> push <ne>;
    <ne> pop <*>;
  }
  // ...
}
```

Protocol Conformance Test

- **Failure semantics (Nierstrasz):**

protocol C conforms to protocol I, iff

- **$\text{traces}(I) \subseteq \text{traces}(C)$**
- **$\text{failures}_I(C) \subseteq \text{failures}(I)$**

Failure Semantics

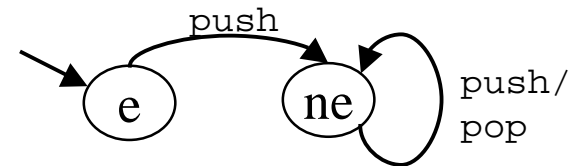
- **traces(I) \subseteq traces(C)**
 - **interface ReadOnly**
 - protocol { o, r*, c; }
 - o,c; o,r,c; o,r,r,c; ...
 - **class File implements ReadOnly**
 - protocol { o, (r|w)*, c; }
 - o,c; o,r,c; o,w,c; o,r,r,c; ...

Failure Semantics

- $\text{failures}_1(\mathbf{C}) \subseteq \text{failures}(\mathbf{I})$

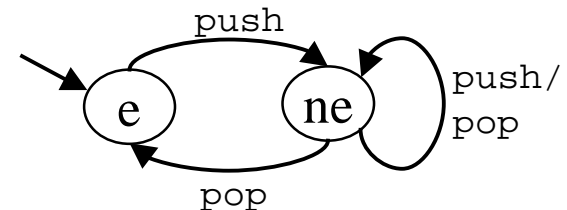
- **interface Var**

- $\langle e \rangle \text{ push } \langle ne \rangle;$
 $\langle ne \rangle \text{ pop} | \text{push } \langle ne \rangle;$
- $\text{push}; \text{push, pop}; \text{push, pop, pop}; \dots$



- **class Stack ~~implements~~ Var**

- $\langle * \rangle \text{ push } \langle ne \rangle;$
 $\langle ne \rangle \text{ pop } \langle * \rangle;$
- $\text{push}; \text{push, pop}; \text{push, pop, pop}; \dots$
- **might fail:** $\text{push, pop, pop}; \dots$



Compilation

- **Class protocol**
 - conformance checked with interface protocols
- **Interface protocol**
 - compiler generates LTS in wrapper class
- **Object creation: `I p = new C();`**
 - compiler wraps object:
`I p = new I_Wrapper(new C(), ...);`

Run-Time Debugging

- **LTS state transitions on method call**
- **Error for illegal method calls**
- **Error for termination in non-final state**

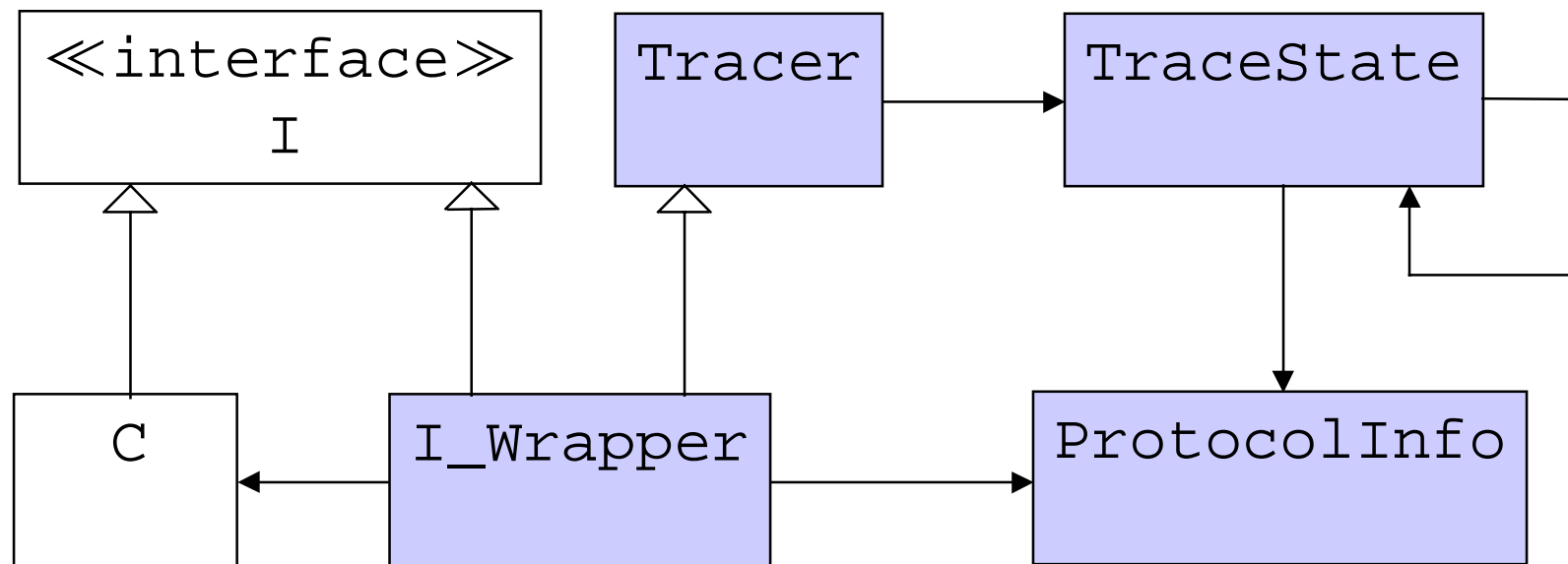
Method Call

- **Method call on wrapper results in**
 - **check whether method call is legal**
 - **call method on server object**
 - **perform LTS state transitions**
 - **remove LTS states with false predicates**
 - **check that set of LTS states is non-empty**

Termination in Non-Final State

- **Error if wrapper object is GCed**
- **User interface for reporting errors on all active protocols in non-final state**

Class Diagram of Debug Tool



Summary

- **Protocols as part of class/interface**
- **Protocol conformance relationship**
- **Run-time check of state properties**
- **Status & plans**
 - **Implemented in JDK-1.1.7 (wrapper generation/insertion broken)**
 - **Future: formal JavaDoc comment, preprocessor implementation**