Specification and Testing of Protocols in Distributed Object Systems

Prakash Krishnamurthy
Paul A.G. Sivilotti

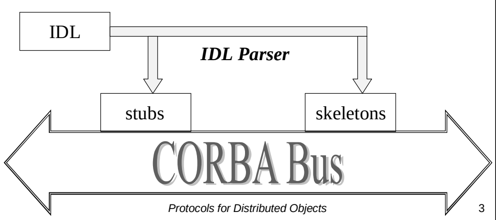
Dept. of Computer & Info. Science
The Ohio State University

Background: CORBA and IDL

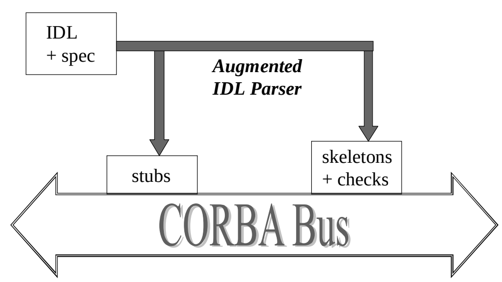
- CORBA
  - Industry standard for distributed object computing
- IDL
  - Notation for defining object interface
  - Programming language neutral

IDL - Interface Definition Language

- IDL description is given to a parser.
- Creates repositories, skeletons, stubs.



Extending the IDL



Example: GUI

```
interface GUI {
   state bool button_down;
   state enum {idle, ready, done} status;

   (status == idle) next
                      (status == idle || status == ready)
   (button_down) next
                      (button_down || status == ready)

   transient.(status == idle && button_down)
};
```


Challenge

- Temporal logic has high “intimidation factor” for many practitioners
- Focus on a specific subclass of safety properties: protocols
- Design goals:
  - Clean and simple specification notation
  - Automated support for run-time validation

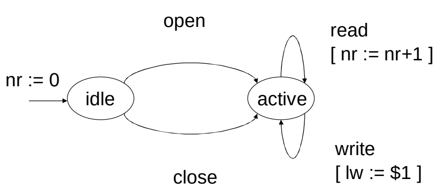
Labeled Transition System

- State
  - Finite enumeration of "phases"
  - Collection of simply typed variables
- Transitions
  - Arcs labeled with messages
  - Method names, argument values
  - Indicate how variables are modified

Example: File Access IDL

```
interface FileAccess {
  void open ( );
  void close ( );
  int read ( );
  void write (int);
};
```


Example: File Access



Protocol Specification in CertIDL

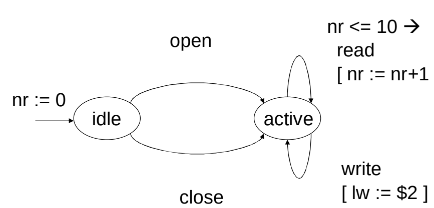
```
protocol FileAccess {
  state >idle, active;
  var int nr = 0, lw;
  <idle> open <active>
  <active> read <nr := nr + 1; active>
  <active> write <lw := $1; active>
  <active> close <idle>
};
```


Guarded Transitions

- A transition is enable according to state (phase and variable values)
  - "with" clause indicates guard on variables

```
<active> read with nr <= 10
                <nr := nr + 1; active>
```


Example 2: File Access with Guards



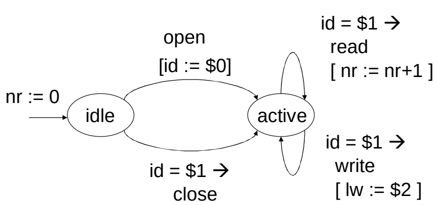
Multiplicity of Protocols

- One component may support several protocol *types*
  - protocol FileAccess {…};
  - protocol UserPrivileges {…};
- Each protocol type, may have several *instances*
  - Each instance has its own phase and its own copy of the variables
  - A transition updates the state of a single instance

IDL Declaration

```
interface MultipleFileAccess {
  int open ( );
  void close (int);
  int read (int);
  void write (int, int);
};
```


Example 3: Multiple File Access



Protocol Specification in CertIDL

```
protocol MultipleFileAccess {
  state >idle, active;
  var int nr = 0, lw, id;
  <idle> open <id := $0; active>
  <active> read with $1 = id
                <nr := nr + 1; active>
  <active> write with $1 = id
                <lw := $2; active>
  <active> close with $1 = id <idle>
};
```


Protocol Types: Shared State and Transitions

- Some state is common to all instances
  - eg, the number of open files
  - "shared state": phases and variables
- Instance transitions:
  - Can use shared state in guards
  - Can not modify shared state
- Shared transitions
  - Can not use or modify instance state
  - Enabled when an instance can accept

Protocol Specification in CertIDL

```
protocol MultipleFileAccess {
  state local >idle, active;
  var local int nr = 0, lw, id;
  var shared int n = 0;

  open <n := n+1>
  close <n := n-1>
  <idle> open <id := $0; active>
  <active> read with $1 = id && n <= 10
                <nr := nr + 1; active>
  <active> write with $1 = id
                <lw := $2; active>
  <active> close with $1 = id <idle>
};
```

Singleton Protocols

- Some protocols are not meant to be multiply instantiated
  - Original FileAccess protocol
  - As written, would accept:
    open read open close close
- "Singleton"
  - No shared state or shared transitions

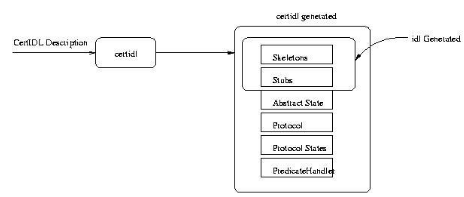
Singleton Protocol

```
singleton protocol FileAccess {
  state local >idle, active;
  var local int nr = 0, lw;
  <idle> open <active>
  <active> read <nr := nr + 1; active>
  <active> write <lw := $1; active>
  <active> close <idle>
};
```

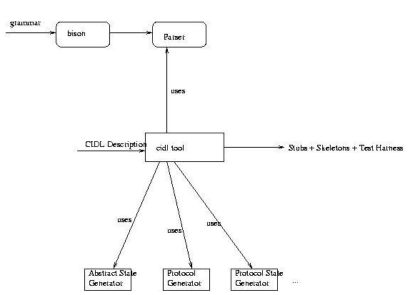

Other Features

- Synchronized transitions
  - Multiple instances accept single message
- Global state
  - Abstract state, implementation provided
- Synchronous / asynchronous support
- Messages received and sent

CertIDL

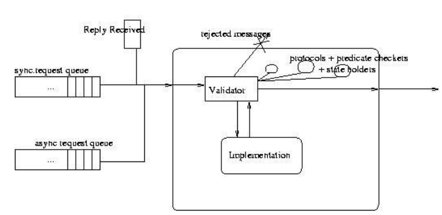


Tool Architecture

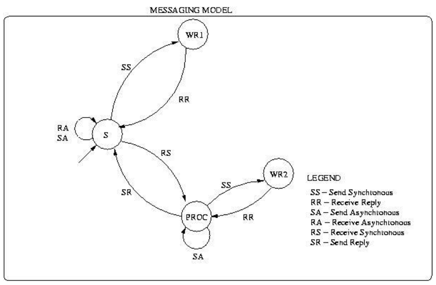


Run-Time Validation

RUN TIME MONITORING



Synchrony vs Asynchrony



Related Work

- OO community
- CFSM (S, Prospec)
- Petri nets (P-nut, PROTEAN)
- Estelle (NBS)
- LOTOS (Sedos)
- SDL

Distinguishing Features

- Design focus: run-time validation
  - Testing harness implementation
- Hierarchy of protocol "granularity"
  - Instance, class, abstract state
  - Multiplicity of instantiation
    - Coupling transitions, sharing state
- Inclusion of message arguments

Current Status

- Specification notation
  - Limitations in mixing send/receive, and synchronous/asynchronous messages
- Implementation
  - CertIDL language and tool suite extended to support protocol validation
- Evaluation
  - Industrial scale application: phone number activation

Acknowledgements

- Distributed Components research group at Ohio State
  - Charlie Giles, Ramesh Jagannathan
- Funding sources:
  - NSF ITR
  - Lucent Technologies
  - Ohio Board of Regents