Language Technology for Internet Telephony Service Creation

Charles Consel, Laurent Réveillère, et al.
Phoenix Research Group, INRIA–Futurs/LaBRI

Presenter: Lenin Singaravelu, Georgia Tech.

Internet Telephony Services

- Rapidly evolving with addition of new functionality such as email, database access and web services

Service Creation

Issues in Service Development

- Software Intensive
- Robustness
- High Performance

- Interaction with low-level hardware
- Varying network types and capabilities
- Customer Needs

- Reuse Commodity services
- Safety, security, dependability

- Multimedia activities
- Multiple processing layers

Solution

Programming Language $\equiv$ Enabling Technology

- Software intensive $\rightarrow$ Language design
- Robustness $\rightarrow$ Program analysis
- High Performance $\rightarrow$ Program transformation
Talk Outline

- Introduction to Domain Specific Languages (DSL)
- Overview of SIP
- SPL: A DSL for communication services
- Properties of SPL
- Summary
Domain-Specific Languages: The Basic Idea

Problem 1
Problem 2
Problem 3

Program 1
Program 2
Program 3

Generator

Problem family
Program family

• Specification
• Declaration
• DSL program

• Application generator
• Preprocessor
• DSL compiler

Domain-Specific Languages: Issues

- When to develop a DSL?
- What is the scope of the DSL?
- How to design for the DSL?
  - Key concept: program family
- How to implement the DSL?
  - Key concept: program family

Key concept: program family

Approach: stepwise methodology
- Data coding/marshalling components
- Device interface layers

How to assess the benefits?
- Key concept: program family
A Program Family with a DSL: Machine Independent Data Coding

- **Commonalities**: data traversal, type description...
- **Variations**: coding direction, size, format...

![Diagram]

- User Application
  - XDR
    - TCP
    - UDP
      - IP
        - Hardw. Interf.
A Program Family with a DSL: Device Driver Interface

- Commonalities: API, bit operations...
- Variations: parameters, registers...

```
Application
  API
System
  API
Driver
    DI
Device
    DevIL
```

A Program Family with a DSL: Application-Specific Protocols

- Commonalities: API, packets operations
- Variations: routing policies, packet processing...

Plan-P

Programmable Router

ASP1

ASP2

Resident Serv 1
Resident Serv 2
Resident Serv 3
Resident Serv 4
... Resident Serv n

Program Family Characterization: A Key Step Towards DSL Design

- **Nature of program family**
  - Existing (Devil)
  - Virtual ($\approx$ Plan-P)
  - Both ($\approx$ Bossa)

- **Scope of programs**
  - Family of layers (Devil)
  - Family of components (Yacc)
  - Family of systems (Scripting language)

- **Range of commonalities**
  (size of program family captured)

- **Scope of computations**
  (kind of expressible computations)
Beyond Program Families

- Technical literature
- Documentation
- Domain experts
- Current and future requirements
Domain-Specific Languages: A (Conceptual) Definition

- Program family
- Domain-specific abstractions and notations
  - Conciseness, readability
- Declarative (often)
  - What to compute, not how to compute it
- Restricted/enriched semantics
  - Making critical properties decidable
Approach to DSLs

- Resulting structure
  - Static semantics: interpreter/compiler
  - Dynamic semantics: Abstract machine

- Stepwise methodology:
  - Program family
  - Library/components
  - Abstract machine
  - DSL
Program Family: Libraries / Components

- Commonalities ⇒ Library entries
- Variations ⇒ Parameters

- Bloated code
  - Numerous cases
Coding Programs: XDR Library

xdr_int
xdr_long
XDR_PUTLONG
xdrmem_putlong
htonl

Integer size
Coding direction
Generic marshalling
Buffer write + overflow check
Byte ordering

Application 1
Application 2
Application 3
Application 4
Application n

Proc 1
Proc 2
Proc 3
Proc 4
Proc 5
Proc 6
Proc 7
Proc 8
Proc 9
Proc 10
Proc m
Program Family: Libraries / Components

- For a given set of problems: *invariants*
Libraries/Components: Need For Customization

- For a given set of problems: 
  - Optimize time (and/or)
  - Optimize space

For a given set of problems:

- Invariants

Programs:

Libraries / Components:


Application 1  Application 2  Application 3  Application 4  Application n
XDR Library: Need For Customization

xdr_int
xdr_long
XDR_PUTLONG
xdrmem_putlong
htonl

Integer size
Coding direction
Generic marshalling
Buffer write + overflow check
Byte ordering

Programs

Libraries / Components

L. Revemere, Phoenix-INRIA, Georgia Tech.
Program Family: Applications

- General-purpose language
- Lack of software architecture

- Prologues/epilogues
  - Prepare invocations
  - Maintain state
  - Perform checks

Results: complex, repetitive, expertise required

Programs

Libraries / Components

Applications: *Need for Abstract Machine* (intermediate step)

- Interface to libraries
- Explicit run-time model
  - State definition and operations
  - Dynamic checks
  - Operation encapsulation
  - Operation combination

- More concise
- More usable
- More explicit expertise
- Abstraction layer

Programs

Libraries / Components

Coding Applications: *Need for Abstract Machine* *(intermediate step)*

- More concise
- More usable
- More explicit expertise
- Abstraction layer

**XDR** Data Coding Machine

- Instruction set: XDR library
- State: XDR structure
- Parameterized instruction set

![Diagram showing the connection between applications and the abstract machine](image-url)
Program Family: Applications

- General-purpose language

- For each call
  - Prologues/epilogues (still)
- For each call sequence
  - Repetitive/repeated call patterns
  - Repetitive/repeated checks

Programs

Abstract Machine

Libraries / Components

Applications: *Need for Domain-Specific Language*

- Interface abstract machine
- Designate a family member
- Domain specific constructs (abstractions/notations)
- Restricted/enriched semantics

Programs:

- Application 1
- Application 2
- Application 3
- Application 4
- Application n

Domain-Specific Language

Abstract Machine

Libraries / Components:

- Proc 1
- Proc 2
- Proc 3
- Proc 4
- Proc 5
- Proc 6
- Proc 7
- Proc 8
- Proc 9
- Proc 10
- ...
Key concept: **Data Structure Descriptions**

**XDR language**
```
union result switch (int error){
    case 0:
        char data[MAX_SIZE];
    default:
        void;
}
```

- Re-use
- Expertise
- Conciseness
- Verification
- Productivity
- Performance

Coding Programs

Domain-Specific Language

Abstract Machine

XDR library

Assessing a DSL

- Existing program family
  - (Almost) objective comparisons:
    - Conciseness
    - Robustness
    - Performance
  - (Likely) subjective comparisons:
    Readability, maintainability, usability
A Program Family with our DSL: Device Driver Interface

- Commonalities: API, bit operations...
- Variations: parameters, registers...

![Diagram showing the relationship between Application, System, Driver, and Device layers, with API and DevIL connections.]

Driver Example : Linux mouse

```
base + 2
1..00000 Direct register

index

base + 0
0       **..**
1       **..**
2       **..**
3       **..**
```

Register array

buttons
dx
dy
Driver Example: Linux mouse

```c
#define MSE_CONTROL_PORT 0x23e
```

Register array

(busmouse.h)
Driver Example : Linux mouse

```
#define MSE_CONTROL_PORT      0x23e
#define MSE_DATA_PORT         0x23c
```

(busmouse.h)
Driver Example : Linux mouse

```
#define MSE_DATA_PORT         0x23e
#define MSE_CONTROL_PORT      0x23c
#define MSE_READ_X_LOW        0x80
```

(busmouse.h)
Driver Example: Linux mouse

```
#define MSE_DATA_PORT      0x23e
#define MSE_CONTROL_PORT   0x23c
#define MSE_READ_X_LOW     0x80
#define MSE_READ_X_HIGH    0xa0
```

(busmouse.h)
Driver Example: Linux mouse

```c
outb(MSE_READ_X_LOW, MSE_CONTROL_PORT);
```

(busmouse.c)
Driver Example: Linux mouse

```c
outb(MSE_READ_X_LOW, MSE_CONTROL_PORT);
dx = (inb(MSE_DATA_PORT) & 0xf);
```

(busmouse.c)
Driver Example: Linux mouse

```
outb(MSE_READ_X_LOW, MSE_CONTROL_PORT);
dx = (inb(MSE_DATA_PORT) & 0xf);
outb(MSE_READ_X_HIGH, MSE_CONTROL_PORT);
```

(busmouse.c)
Driver Example: Linux mouse

```c
outb(MSE_READ_X_LOW, MSE_CONTROL_PORT);
dx = (inb(MSE_DATA_PORT) & 0xf);
outb(MSE_READ_X_HIGH, MSE_CONTROL_PORT);
dx |= (inb(MSE_DATA_PORT) & 0xf) << 4;
```

(busmouse.c)
Driver Example: Linux mouse

```c
outb(MSE_READ_X_LOW, MSE_CONTROL_PORT);
dx = (inb(MSE_DATA_PORT) & 0xf);
outb(MSE_READ_X_HIGH, MSE_CONTROL_PORT);
dx |= (inb(MSE_DATA_PORT) & 0xf) << 4;
```

(busmouse.c)
Driver Example: Linux mouse

```
outb(MSE_READ_X_LOW, MSE_CONTROL_PORT);
dx = (inb(MSE_DATA_PORT) & 0xf);
outb(MSE_READ_X_HIGH, MSE_CONTROL_PORT);
dx |= (inb(MSE_DATA_PORT) & 0xf) << 4;
...
```

(index)

(base + 2)

(base + 0)

(Direct register)

(Register array)

(busemouse.c)
Family of Layers: Device Interface Library

- Bit operations
- Input/output operations
- Low level
- Fine grained
- Directly mapped into machine instructions
- Implicit state

Device drivers

Device Interface Library

Application 1

Proc 1
Proc 2
Proc 3
Proc 4
Proc 5
Proc 6
Proc 7
Proc 8
Proc 9
Proc 10
Proc m

Application 2

Application 3

Application 4

Application n
Applications: **Device Interface Abstract Machine**

- Bit extraction operations
- Bit concatenation operation
- Input/output abstract operations
- Input/output checks

- Dedicated set of bit instructions
- Higher level
- More concise
- More robust

---

**Device drivers**

- Application 1
- Application 2
- Application 3
- Application 4
- Application n

---

**Abstract Machine**

---

**Device Interface Libraries**

- Proc 1
- Proc 2
- Proc 3
- Proc 4
- Proc 5
- Proc 6
- Proc 7
- Proc 8
- Proc 9
- Proc 10
- Proc m

---

Program Family: Applications

- General-purpose language

- General purpose glue for device communications
- Checks subject to programmer’s rigor
- Repetitive communication sequences
Key concept: **Layered Device Interface**

- **Port**
  - Communication point
  - Different media: I/O, memory mapped
- **Register**
  - Repository of data in the device
  - Indexed / paged, read / write, size, mask, ...
- **Variable**
  - Collection of register fragments
  - Semantic value

---

**Domain-Specific Language**

**Abstract Machine**

---

**Device Interface Libraries**

Direct Access to Register

\[ \text{register index\_reg} = \text{write base@2, mask '1..00000': bit[8]}; \]
Private Variable

register index_reg = write base@2, mask '1..00000' : bit[8];
private variable index = index_reg[6..5] : int(2);
Indexed Registers

... register index_reg = write base@2, mask '1..00000' : bit[8];
private variable index = index_reg[6..5] : int(2);
...
register x_low  = read base@0, pre {index = 0}, mask '****....' : bit[8];
register x_high = read base@0, pre {index = 1}, mask '****....' : bit[8];
...
```plaintext
... register index_reg = write base@2, mask '1..00000' : bit[8];
private variable index = index_reg[6..5] : int(2);
...
register x_low = read base@0, pre {index = 0}, mask '****....' : bit[8];
register x_high = read base@0, pre {index = 1}, mask '****....' : bit[8];
...
variable dx = x_high[3..0] # x_low[3..0], volatile : signed int(8);  
variable dy = y_high[3..0] # y_low[3..0], volatile : signed int(8);
```
Comparison

outb(MSE_READ_X_LOW, MSE_CONTROL_PORT);
dx = (inb(MSE_DATA_PORT) & 0xf);
outb(MSE_READ_X_HIGH, MSE_CONTROL_PORT);
dx |= (inb(MSE_DATA_PORT) & 0xf) << 4;
outb(MSE_READ_Y_LOW, MSE_CONTROL_PORT);
dy = (inb(MSE_DATA_PORT) & 0xf);
outb(MSE_READ_Y_HIGH, MSE_CONTROL_PORT);
buttons = inb(MSE_DATA_PORT);
dy |= (buttons & 0xf) << 4;

variable dx = x_high[3..0] # x_low[3..0], volatile : signed int(8);
variable dy = y_high[3..0] # y_low[3..0], volatile : signed int(8);
Talk Outline

- Introduction to Domain Specific Languages (DSL)
- Overview of SIP
- SPL: A DSL for communication services
- Properties of SPL
- Summary
What is SIP?

- SIP: Session Initiation Protocol
  - Voice over IP and 3G mobile phones
  - Standardized by IETF, adopted by ITU
- Protocol for creating, modifying and terminating sessions between devices
- Based on a client/server model
- Mobility: SIP URI
SIP: Mobility

alice@1.2.3.4

Location server: [alice→4.3.2.4]

alice@4.3.2.1

alice@organization.org

REGISTER

200 OK

© 2002, 2005, 2007 C. Consel,
L. Réveillère, Phoenix-INRIA, Georgia Tech.
SIP: Typical Call Flow

INVITE

bob@company.com

company.com?

 alice

1.2.3.4

organization.org

INVITE

OK

communication (audio, video, game, ...)

INVITE

ok

company.com

location server

bob?

19.21.22.23

11.12.13.14

15.16.17.18

19.21.22.23

bob

Programming Telephony Services

- A service processes SIP messages (incoming/outgoing) on behalf of a person or a group of persons
  - Service logic
  - Signalling actions
    - Reject
    - Forward

Preferences

Time

Request fields

Environment

Existing Solutions

- Use General Purpose Languages:
  - Microsoft SIP APIs – C#
  - SIP Express Router – C-like and C
  - JAIN SIP – Java

- Requires extensive knowledge of existing protocols

Expressive, but unsafe
Why a DSL for Telephony Services?

- Telephony services are a rich domain
- SIP is a rich protocol
- SIP has many companion protocols
- Programming services requires expertise in multiple areas
  - Protocols
  - Networking
  - Platform
- Few trained programmers
- Reliability is critical
- Performance is critical
DSL Based Approach

Domain analysis:
SIP-based call routing services
Abstraction Layer

JAIN
JAVA
C#
C

SER module

SPL
Example

- Abstract service specification
  - When registering
    - Set a counter to 0
  - When a call is received
    - If the callee rejects the call:
      - Forward the call to the secretary
    - Otherwise
      - Increment the counter
  - When un-registering
    - Log the value of the counter
Example: JAIN SIP Code

```java
public class Counter implements SipListener {

  public void processRequest(RequestEvent requestEvent) {
    try {
      Request rq_request = requestEvent.getRequest();
      SipProvider rq_sipProvider = (SipProvider) requestEvent.getSource();
      String method = rq_request.getMethod();
      if (method.equals(Request.REGISTER)) {
        if (!registrar.hasExpiresZero(rq_request)) {
          if (!registrar.hasRegistration(rq_request)) {
            State state = new State();
            int ident = env.getId(rq_request);
            env.setEnv(ident, state);
            rq_sipProvider.sendRequest(rq_request);
            return;
          } else {
            rq_sipProvider.sendRequest(rq_request);
            return;
          }
        } else if (lib.registrar.hasRegistration(rq_request)) {
          int ident = env.getId(rq_request);
          State state = (State)env.getEnv(ident);
          log(state.count); env.delEnv(ident);
          rq_sipProvider.sendRequest(rq_request);
          return;
        } else {
          rq_sipProvider.sendRequest(rq_request);
          return;
        }
      } else if (method.equals(Request.INVITE)) {
        ClientTransaction ct = rq_sipProvider.getNewClientTransaction(rq_request);
        ct.sendRequest(); return;
      } else {
        rq_sipProvider.sendRequest(rq_request);
        return;
      }
    } catch(Exception ex) { [...]
  }
}

public void processResponse(ResponseEvent responseEvent) {
  try {
    Response rs_response = responseEvent.getResponse();
    SipProvider rs_sipProvider = (SipProvider) responseEvent.getSource();
    rs_responseCode = rs_response.getStatusCode();
    ClientTransaction ct = responseEvent.getClientTransaction();
    if (clientTransaction != null) {
      Request rs_request = ct.getRequest();
      String method = rs_request.getMethod();
      int ident = env.getId(rs_request);
      State state = (State)env.getEnv(ident);
      if (method.equals(Request.INVITE)) {
        if (rs_responseCode >= 300) {
          AddressFactory addressFactory = lib.getAddressFactory();
          SipURI sipURI = addressFactory.createSipURI("secretary","company.com");
          rs_request.setRequestURI(sipURI);
          int ident = env.getId(rs_request);
          env.delEnv(id);
          sipProvider.sendRequest(rs_request);
          return;
        } else {
          state.count++; ServerTransaction st = rs_request.getNewServerTransaction();
          st.sendResponse(rs_response);
          return;
        }
      } else {
        rs_sipProvider.sendResponse(rs_response);
        return;
      }
    } else {
      rs_sipProvider.sendResponse(rs_response);
      return;
    }
  } catch (Exception ex) { [...]
  }
}
```

Example: SPL Code

```c
service Counter {
    processing {
        local void log (int);

        registration {
            int count;

            response outgoing REGISTER() {
                count = 0;
                return forward;
            }

            void unregister() {
                log (count);
            }

        } dialog {
            response incoming INVITE() {
                response resp = forward;
                if (resp != /SUCCESS) {
                    return forward 'sip:secretary@company.com';
                } else {
                    count++;
                    return resp;
                }
            }
        }
    }
}
```
SPL Components

- Event handlers and signaling operations
- Session
- Hierarchical sessions
  - Service
  - Registration
  - Dialog
- Inter-event control flow
Event Handlers and Signaling
Operations

SPL Service

response incoming INVITE() {
    [...] 
    response resp = forward;
    if (resp == /ERROR) {
        resp = forward 'sip:phoenix.secretary@inria.fr';
    }
    return resp;
}

// Deny Service
response incoming INVITE() {
    return /ERROR/CLIENT/BUSY_HERE;
}
Session

- INVITE
- BYE
- REINVITE
- ...

Dialog = ID

uri caller; time start; ...

Session: The Dialog Session

Example

dialog {
    uri caller;
    time start;

    response incoming INVITE() {
        caller = FROM;
        return forward;
    }

    void incoming ACK() {
        if (caller == 'sip:my.wife@home.fr')
            log("Personal call");
        start = getTime();
    }

    response BYE() {
        string duration = time_to_string(getTime() - start);
        log("Call: " + duration + " "+uri_to_string(caller));
        return forward;
    }
}
Hierarchical Sessions

- REGISTER
- REREIGISTER
- unregister

- INVITE
- BYE
- REINVITE
- ...

Registration = ID

Dialog = ID1
- uri caller;
- time start;
- ...

Dialog = ID2
- uri caller;
- time start;
- ...

Hierarchical Sessions: Example

registration {
  uri employee;
  time startWorkDay;

  response outgoing REGISTER() {
    startWorkDay = getTime();
    employee = FROM;
    return forward;
  }

  void unregister() {
    string duration = time_to_string(getTime() - startWorkDay);
    log("WorkDay: "+ duration +" "+uri_to_string(employee));
    return;
  }

  dialog {
    uri caller; time start;

    ...
  }
}
Hierarchical Sessions

- deploy
- undeploy
  - REGISTER
  - REREREGISTER
  - unregister
  - INVITE
  - BYE
  - REINVITE
  - ...

Service = ID

FIFO uri<> employees;
...

Registration = ID

uri employee;
time startWorkDay;
...

Dialog = ID1
Dialog = ID2

uri caller;
time start;
...

© 2002, 2005, 2007 C. Consel,
L. Réveillère, Phoenix-INRIA, Georgia Tech.
Hierarchical Sessions: Example (cont’d)

```java
service hotline {
    ...
    processing {
        uri<100> employees = <>;

        void deploy() {...}
        void undeploy() {...}

        registration {...

            response outgoing REGISTER() {
                startWorkDay = getTime();
                employee = FROM;
                push employees employee;
                return forward;
            }
            ...

            dialog {...
                response incoming INVITE() {
                    return forward employees;
                }
            }
        }
    }
}
```
Inter-Event Control Flow

dialog {
  response incoming INVITE() {
    response r;
    ...
    if (...) {
      ...
      return r branch hotline;
    } else {
      ...
      return r branch personal;
    }
  }

  void incoming ACK(){
    branch hotline {... }
    branch default {... }
  }

  response BYE() {
    branch hotline {... }
    branch personal {... }
    branch default {... }
  }
}
Talk Outline

- Introduction to Domain Specific Languages (DSL)
- Overview of SIP
- SPL: A DSL for communication services
- Properties of SPL
- Summary
Portability of SPL Services

SPL

JAIN SIP

Asterisk

LCS

...
Abstraction Layers

- Programming language
- Framework
- Middleware
- Network protocols

© 2002, 2005, 2007 C. Consel,
L. Réveillère, Phoenix-INRIA, Georgia Tech.
public class Example implements SipListener {
    private AddressFactory factory = getAddressFactory();

    public void processRequest (RequestEvent requestEvent) {
        Request rq_request = requestEvent.getRequest();
        SipProvider rq_sipProvider = (SipProvider) requestEvent.getSource();
        String method = rq_request.getMethod();
        """
        if (method.equals (Request.INVITE)) {
            SipURI uri = factory.createSipURI ("bob", "phone.example.com");
            rq_request.setRequestURI (uri);
            ClientTransaction ct = rq_sipProvider.getNewClientTransaction(rq);
            ct.sendRequest (rq_request);
        }
        """
        public void processResponse (ResponseEvent responseEvent) {
            ClientTransaction rs_ct = responseEvent.getClientTransaction();
            if (rs_ct != null) {
                Request rs_request = rs_ct.getRequest();
                Response rs_response = responseEvent.getResponse();
                SipProvider rs_sipProvider = (SipProvider) responseEvent.getSource();
                String method = rs_request.getMethod();
                rs_responseCode = rs_response.getStatusCode();
                if (method.equals (Request.INVITE)) {
                    if (rs_responseCode == 486) {
                        SipURI uri = factory.createSipURI ("bob", "voicemail.example.com");
                        rs_request.setRequestURI (uri);
                        rs_sipProvider.sendRequest (rs_request);
                    } else if (rs_responseCode > 300) {
                        if (rs_request.getHeader("FROM").equals("sip:boss@example.com ")) {
                            TelURL tel = factory.createTelURL ("tel:+19175554242");
                            rs_request.setRequestURI (tel);
                            rs_sipProvider.sendRequest (rs_request);
                        } else {
                            rs_sipProvider.sendResponse (rs_response);
                        }
                    }
                }
            }
    }
}
Abstraction Layers

- Programming language
- Framework
- Middleware
- Network protocols

GAP
- Programming skills
- Wide range of applicability
- Programming-oriented errors
- Programming-oriented safety properties

Domain Expert?
A Layered Domain-Specific Language Approach

- Defining a solution
  - High level
  - Simple
- Verifying a solution
  - Domain properties
- High-level tools for
  - Compilation
  - Verification

DSML

MODELING

Programming language
Framework
Middleware
Network protocols

Programming

Implementation

Domain Expert
A Layered Domain-Specific Language Approach

- Defining a solution
  - Visual
  - Activity diagram
- Verifying a solution
  - Domain properties
- Compilation
  - SPL
  - Formulas

Domain Expert

Modeling

Programming

Implementation

- SPL
- CPL

• Programming language
• Framework
• Middleware
• Network protocols
A CPL Telephony Service

<?xml version="1.0" encoding="UTF-8"?>
<cp1>
  <incoming>
    <location url="sip:bob@phone.example.com">
      <proxy>
        <busy>
          <location url="sip:bob@voicemail.example.com">
            <proxy />
          </busy>
          <otherwise>
            <address-switch field="origin">
              <address is="sip:boss@example.com">
                <location url="tel:+19175554242">
                  <proxy />
                </location>
                <proxy />
              </address>
            </address-switch>
          </otherwise>
        </proxy>
      </location>
    </proxy>
  </location>
</incoming>
</cp1>
Introducing Multiple DSMLs

Modeling

Trust level / Purpose / Abstraction level / Preferences / …

DSML 1  DSML 2  DSML 3  DSML 4

Domain-Specific Programming Language

Programming

Implementation

Implementation 1  Implementation 2  Implementation 3

© 2002, 2005, 2007 C. Consel,
L. Réveillère, Phoenix-INRIA, Georgia Tech.
Processing DSMLs

- High-level tools
- Compilation
  - DSPL Implementation
  - Formal modeling
- Verification
  - Domain-oriented properties
Processing A DSML

- Model Checking
  - Modeling in TLA
  - TLC model checker
- Properties
  - No call loss
  - No duplicate redirect
  - No infeasible path

From CPL to SPL

service example {
  processing {
    dialog {
      response incoming INVITE() {
        response r = forward 'sip:bob@phone.example.com';
        if (r == /ERROR/CLIENT/BUSY_HERE) {
          return forward 'sip:bob@voicemail.example.com';
        } else if (r == /ERROR) {
          if (FROM == 'sip:boss@example.com') {
            return forward 'tel:+19175554242';
          } return r;
        }
      }
    }
  }
}
Domain-Oriented Properties

\[
\begin{align*}
\text{AtLeastOneSigAction} & \overset{\triangleq}{=} \text{currentNode} = \text{“End”} \Rightarrow \text{Len(sigActions)} \neq 0 \\
\text{No Twice Redirect To The Same URI} & \overset{\triangleq}{=} \\
& \quad \Box(\forall n \in 1 \ldots \text{Len(sigActions)} : \forall m \in n + 1 \ldots \text{Len(sigActions)} : \\
& \quad \quad \quad \text{sigActions}[n] \neq \text{“Continuation”} \Rightarrow \text{sigActions}[n] \neq \text{sigActions}[m]) \\
\text{Consistency} & \overset{\triangleq}{=} \\
& \quad \land \Box(\exists x \in \text{Addresses} : \forall n \in 1 \ldots \text{Len(addrTest)} : x \in \text{addrTest}[n]) \\
& \quad \land \Box(\text{date} \neq \{\})
\end{align*}
\]
Checking A Telephony Service

TLC Version 2.0 of January 16, 2006
Model-checking

Finished computing initial states: 1 distinct state generated.
Error: Invariant line 143, col 23 to line 143, col 35 of module CPL is violated.
The behavior up to this point is:
STATE 1: <Initial predicate>
\[ \land addrTest = \square \square \land currentNode = "Incoming" \land sigActions = \square \square \land date = \{ \{ day \mapsto "sun", dayNum \mapsto 1, month \mapsto "January" \}, \ldots \{ day \mapsto "sun", dayNum \mapsto 365, month \mapsto "December" \} \} \]
STATE 2: <Action line 51, col 9 to line 53, col 60 of module CPL>
\[ \land addrTest = \square \square \land currentNode = "WeeklyMeeting" \land sigActions = \square \square \land date = \{ \{ day \mapsto "sun", dayNum \mapsto 1, month \mapsto "January" \}, \ldots \{ day \mapsto "sun", dayNum \mapsto 365, month \mapsto "December" \} \} \]
STATE 3: <Action line 98, col 9 to line 101, col 54 of module CPL>
\[ \land addrTest = \square \square \land currentNode = "AnnualHolidays" \land sigActions = \square \square \land date = \{ \{ day \mapsto "tue", dayNum \mapsto 3, month \mapsto "January" \}, \ldots \{ day \mapsto "tue", dayNum \mapsto 192, month \mapsto "July" \}, \ldots \{ day \mapsto "tue", dayNum \mapsto 360, month \mapsto "December" \} \} \]
STATE 4: <Action line 110, col 9 to line 113, col 55 of module CPL>
\[ \land addrTest = \square \square \land currentNode = "RedirectToBobVoiceMail" \land sigActions = \square \square \land date = \{ \} \]

5 states generated, 5 distinct states found, 2 states left on queue.
The depth of the complete state graph search is 4.
Summary

- SPL: Session Processing Language
- Contributions
  - High level abstractions and notations
    - Specific to the telephony domain
    - Ease the development process
    - Conciseness (factor of 4)
  - Performance
    - Domain-specific optimization (e.g., state management)
  - Robustness
    - Critical properties guaranteed at different levels:
      - protocol, platform, and service