# Architectural Design/Architectural Evaluation

## Architectural Prototyping

# Literature

## Main

- [Bardram et al., 2004]
  - Bardram, J., Christensen, H. B., and Hansen, K. M. (2004). Architectural Prototyping: An Approach for Grounding Architectural Design and Learning. In *Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, pages 15–24, Oslo, Norway

## Background (i.e., not part of curriculum)

- [Hansen and Wells, 2006]
  - Hansen, K.M. and Wells, L. (2006). Dynamic Design and Evaluation of Software Architecture in Critical Systems Development. In *Proceedings of the 11th Australian Conference on Safety Related Programmable Systems,* Melbourne, Australia

# Styles, patterns, tactics are great

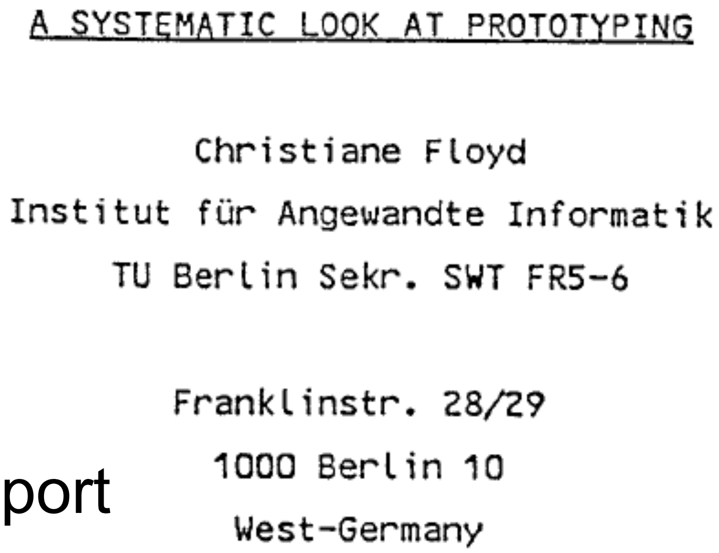
However, what do we do when...

- we are *uncertain* whether one or the other style / pattern / tactic is the better to choose?
- when we are *uncertain* whether the favourite architecture will have the right balance of conflicting quality attributes?
- when we are *uncertain* that the specs of the third party vendor are *real* and not just empty sales talk?
- when we want to *explore* the design space for learning – and becoming better architects?

# Prototyping!

Seminal paper by Floyd [1984].

- executable systems that “involve an early practical demonstration of relevant parts of the desired software”.
- “a learning vehicle providing more precise ideas about what the target system should be like.”
- “the discussion focuses on software intended as direct support for human work.”



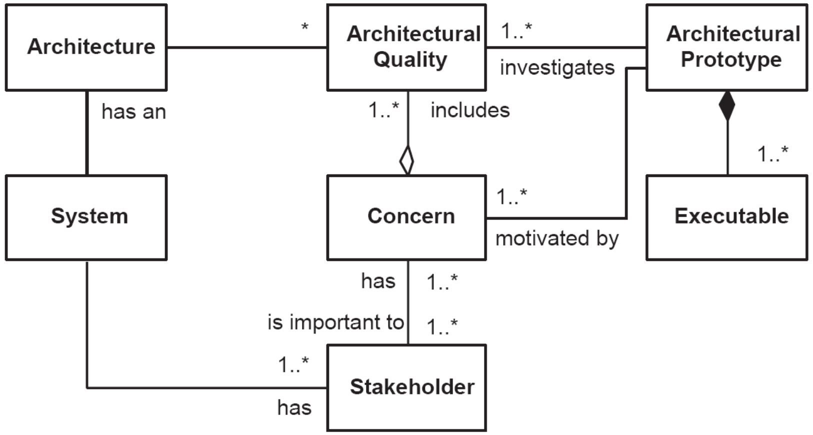
A SYSTEMATIC LOOK AT PROTOTYPING

Christiane Floyd
Institut für Angewandte Informatik
TU Berlin Sekr. SWT FR5-6

Franklinstr. 28/29
1000 Berlin 10
West-Germany

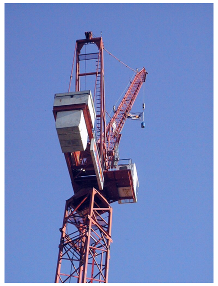
# Applying this to software architecture?

An *architectural prototype* consists of a set of executables created to investigate architectural qualities related to concerns raised by stakeholders of a system under development



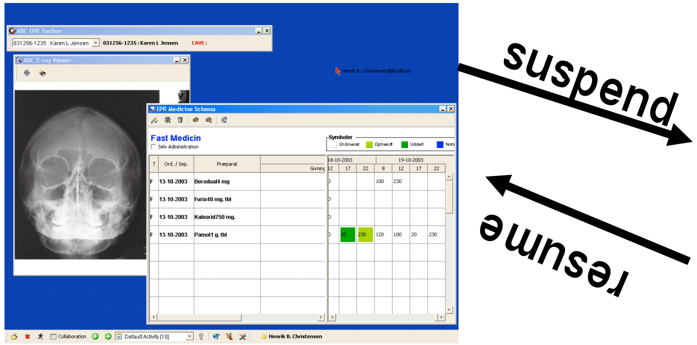
*Architectural prototyping* is the process of designing, building, and evaluating architectural prototypes.

# AP Examples



SafeInverter
– safety

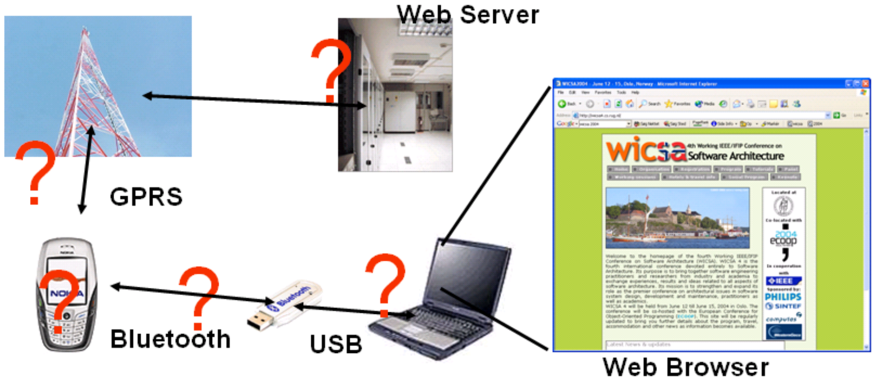
Closed Loop Process Control
– performance



State space
in XML

```
<activity id="local_1" status="2" name="Local copy of hello" type="0"
  creator="jonathan">
  <meta>
    <description>This activity is a local,
      non-collaborative copy made |lp|09-03-2005|rp| of the activity named
      |an|hello|an| |lp|with id 1|rp|
    </description>
    <participants>
      <person id="jonathan">
      </person>
    </participants>
  </meta>
  <state>
    <service id="590642#632467386353120976" type="browser">
      <component id="frame" type="">
        451,72,1033,853
      </component>
      <component id="HorizontalScrollPos" type="">
        0
      </component>
      <component id="VerticalScrollPos" type="">
        0
      </component>
      <component id="fullscreen" type="">
        False
      </component>
      <component id="location" type="">
        http:|sl||sl|www.hp.com|sl|
      </component>
      <component id="WindowState" type="">
        normal
      </component>
    </service>
  </state>
</activity>
```

Activity-Based Computing (ABC)
– modifiability, buildability



Palpable Computing
– availability, usability

# Types of Architectural Prototypes

- Exploratory prototypes
  - Clarify architectural requirements with stakeholders
  - Explore aspects of target system
  - Discuss alternative solutions

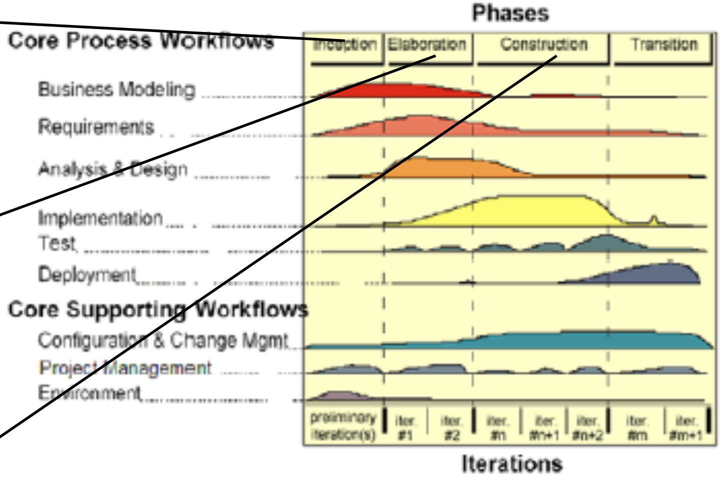
i.e., finding proposals

- Experimental prototypes
  - Gauge adequacy of proposed architecture
  - Quantitative measurements of quality attributes like performance

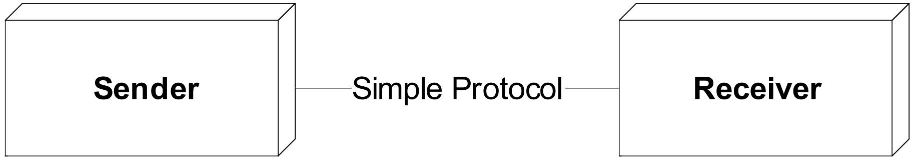
i.e., evaluating proposals

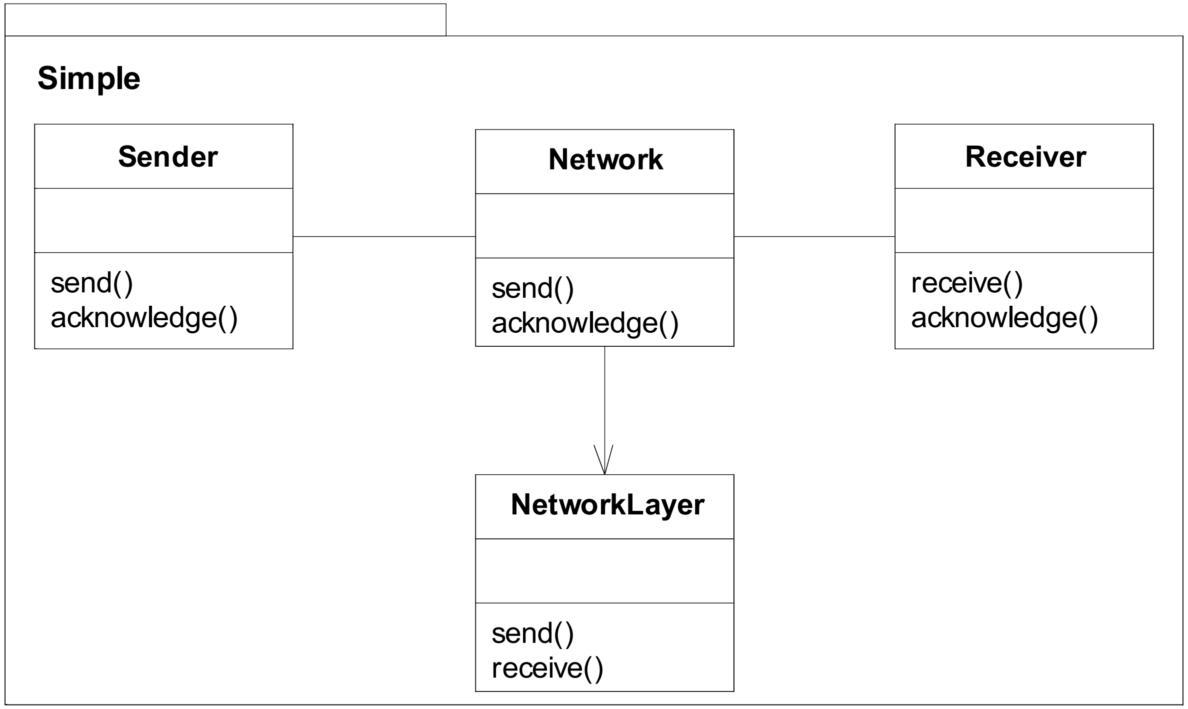
- Evolutionary prototypes
  - Target environment
  - Keep evolving prototypes

i.e., evolving proposals

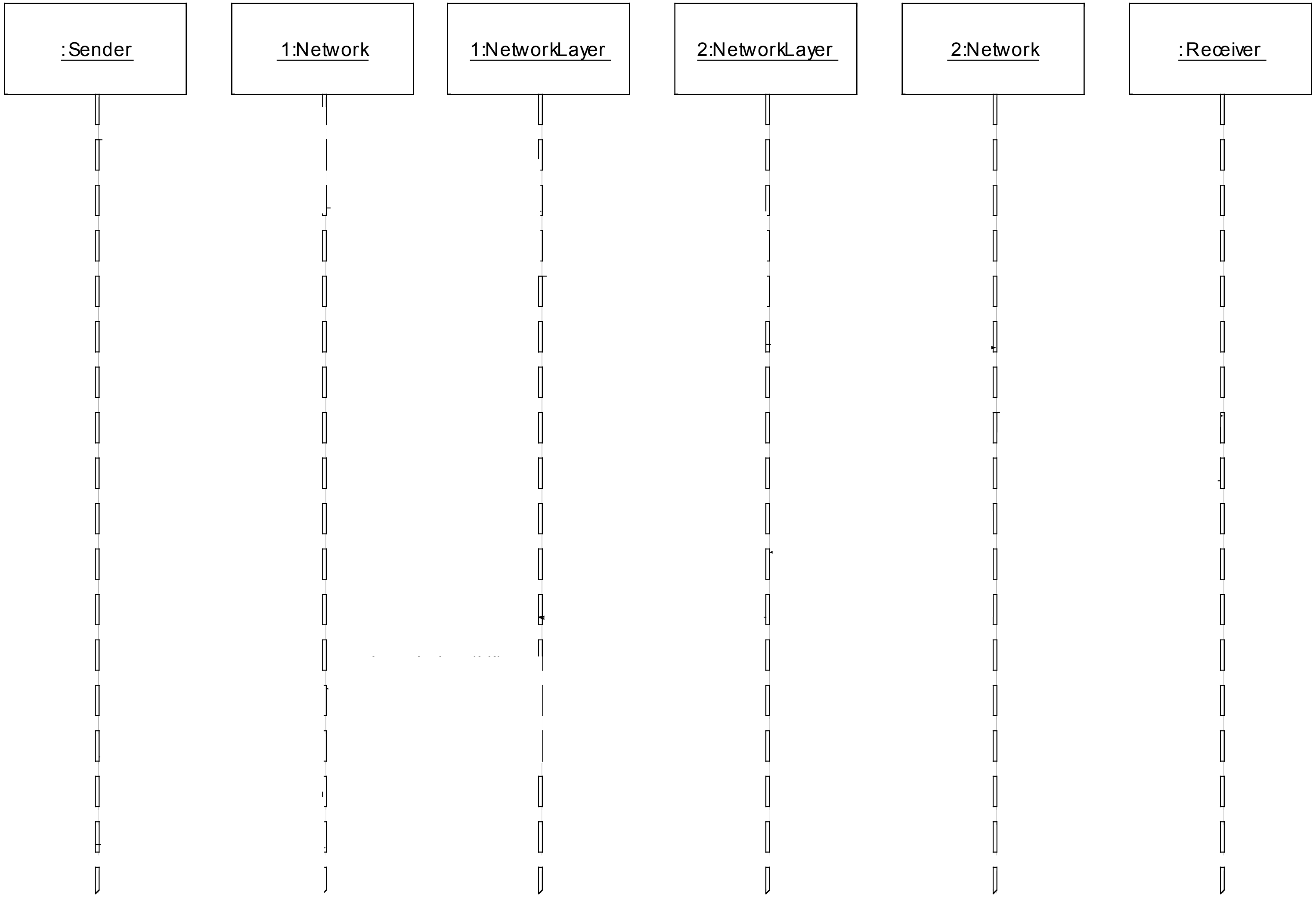


# Simple Example: Simple Protocol





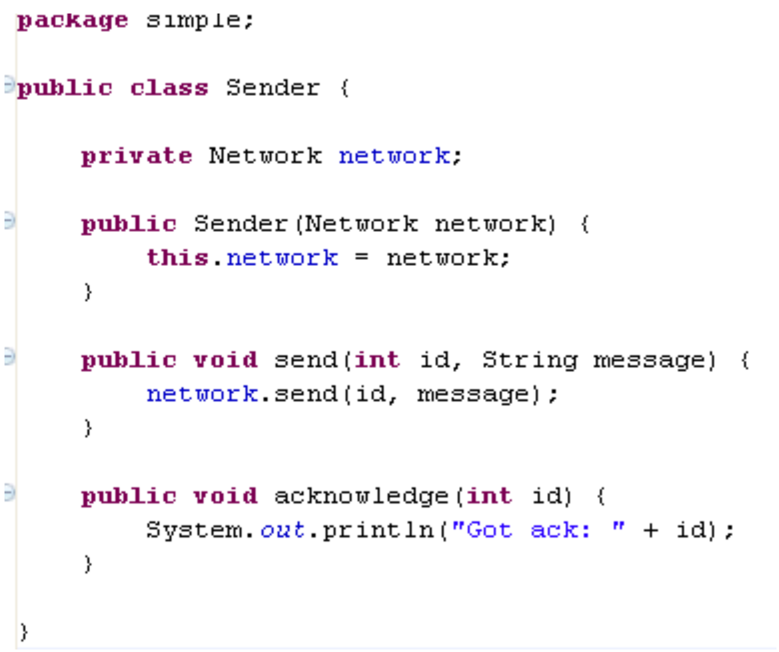
# Simple Protocol: C&C View



# Simple Protocol: Architectural Prototype

Sender

- Trivial implementation

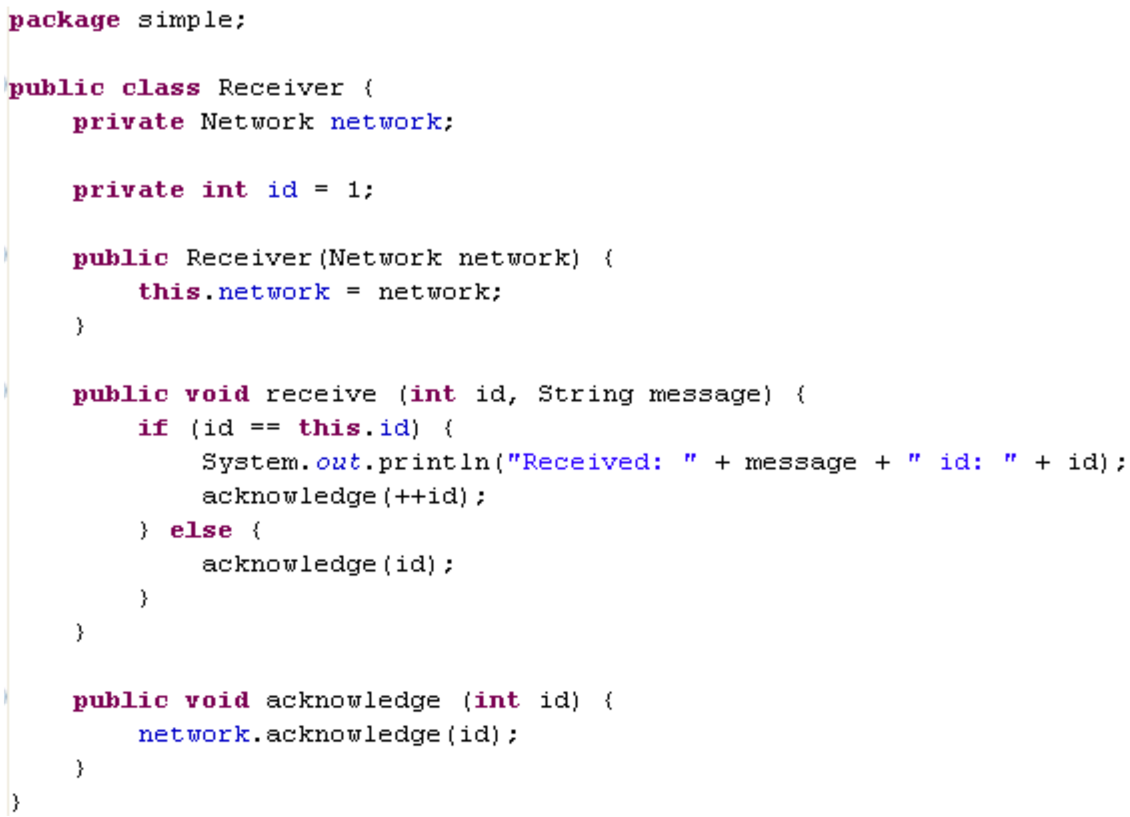


```
package simple;

public class Sender {

    private Network network;

    public Sender(Network network) {
        this.network = network;
    }

    public void send(int id, String message) {
        network.send(id, message);
    }

    public void acknowledge(int id) {
        System.out.println("Got ack: " + id);
    }

}
```

# Simple Protocol: Architectural Prototype

## Receiver

- Trivial implementation

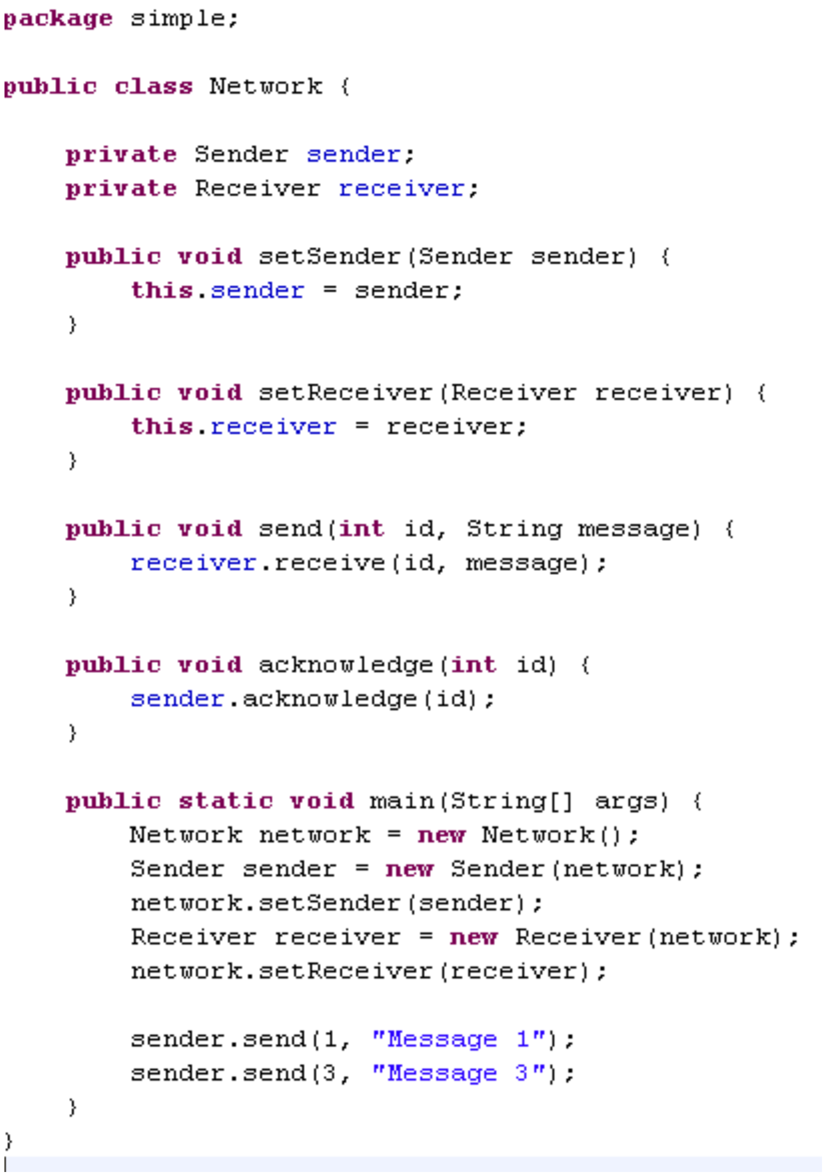


```
package simple;

public class Receiver {
    private Network network;

    private int id = 1;

    public Receiver(Network network) {
        this.network = network;
    }

    public void receive (int id, String message) {
        if (id == this.id) {
            System.out.println("Received: " + message + " id: " + id);
            acknowledge(++id);
        } else {
            acknowledge(id);
        }
    }

    public void acknowledge (int id) {
        network.acknowledge(id);
    }
}
```

# Simple Protocol: Architectural Prototype

Network

- NetworkLayer abstracted away
- Drives prototype



```
package simple;

public class Network {

    private Sender sender;
    private Receiver receiver;

    public void setSender(Sender sender) {
        this.sender = sender;
    }

    public void setReceiver(Receiver receiver) {
        this.receiver = receiver;
    }

    public void send(int id, String message) {
        receiver.receive(id, message);
    }

    public void acknowledge(int id) {
        sender.acknowledge(id);
    }

    public static void main(String[] args) {
        Network network = new Network();
        Sender sender = new Sender(network);
        network.setSender(sender);
        Receiver receiver = new Receiver(network);
        network.setReceiver(receiver);

        sender.send(1, "Message 1");
        sender.send(3, "Message 3");
    }
}
```

# Concerns to Investigate?

[Bass et al., 2003]

- Performance?
- Avaliability?
- Modifiability?
- Testability?
- Security?
- Usability?

- Conceptual integrity?
- Correctness and completeness?
- Buildability?

# Coloured Petri Nets

Formal, graphical, behavioral modeling language with well-defined syntax and semantics

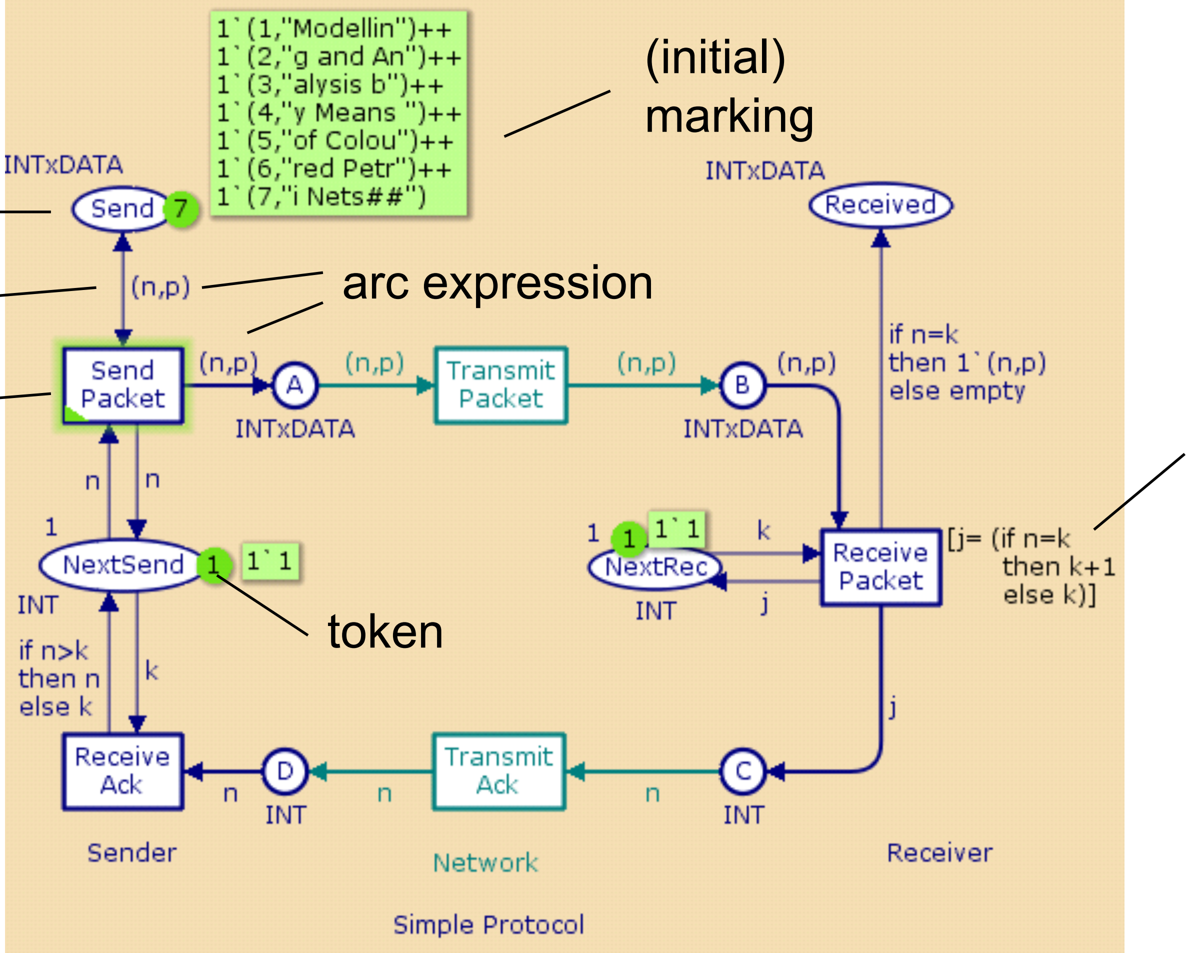
- ”Petri Nets with types”

Formal definition

$$CPN = (\Sigma, P, T, A, N, C, G, E, I)$$

- $\Sigma$: finite set of non-empty types, ”colour sets”
- P, T, A: non-empty, disjoint sets of of places, transitions, and arcs, respectively
- N: Node function defined from A into $(P \times T) \cup (T \times P)$
- C: Colour function from P into $\Sigma$
- G: Guard function defined from T into boolean expressions
- E: Arc expression function defined from A into expressions
- I: Initialization function defined from P into expressions

# Coloured Petri Nets: Simple Protocol



colour

place

arc

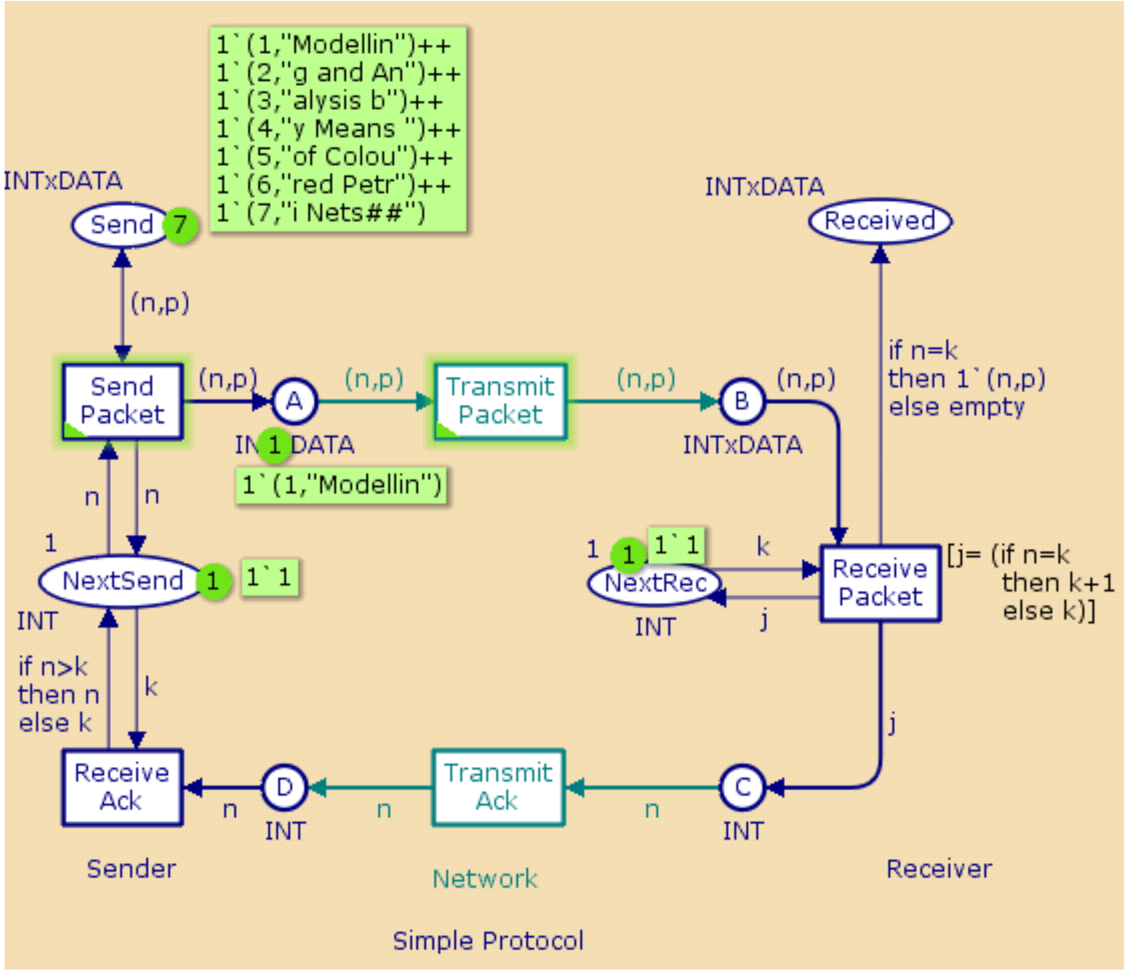
transition

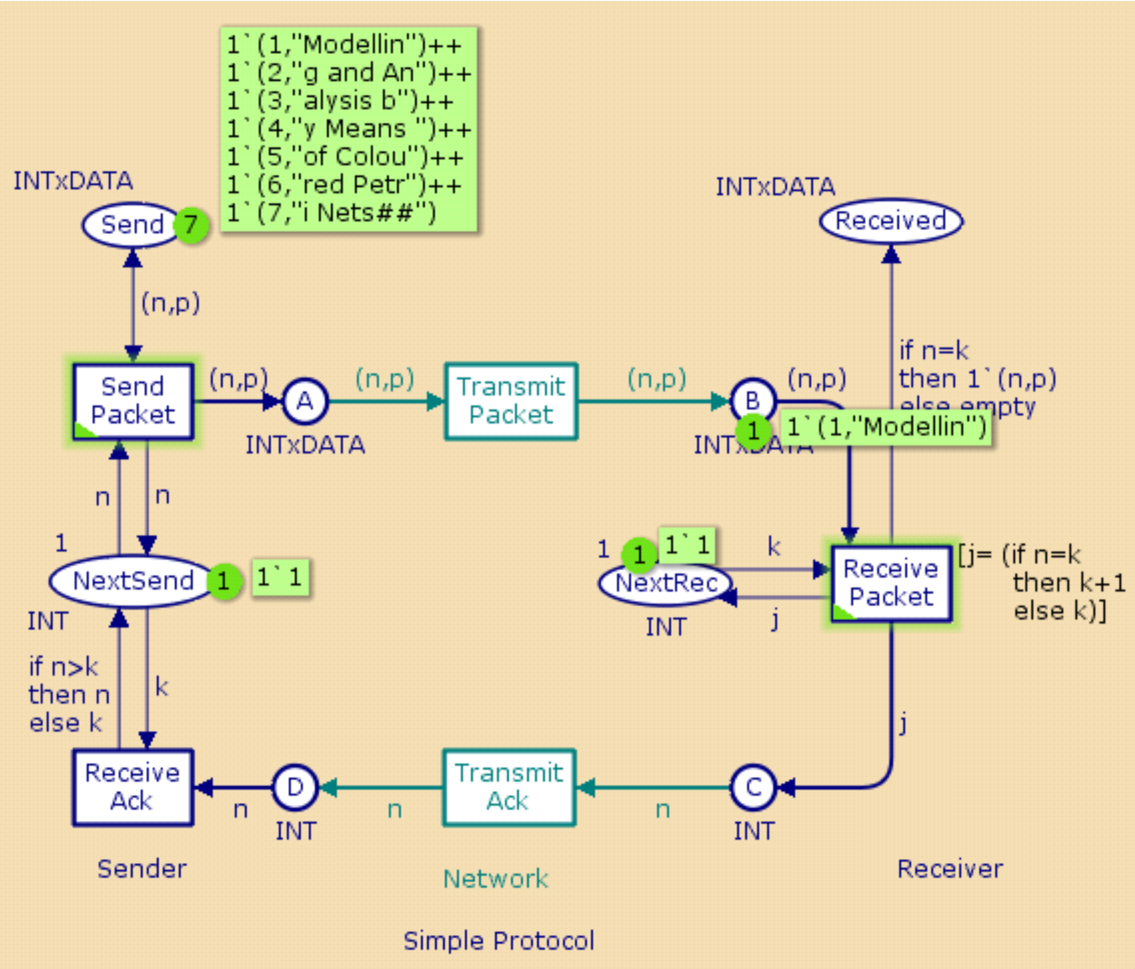
(initial) marking

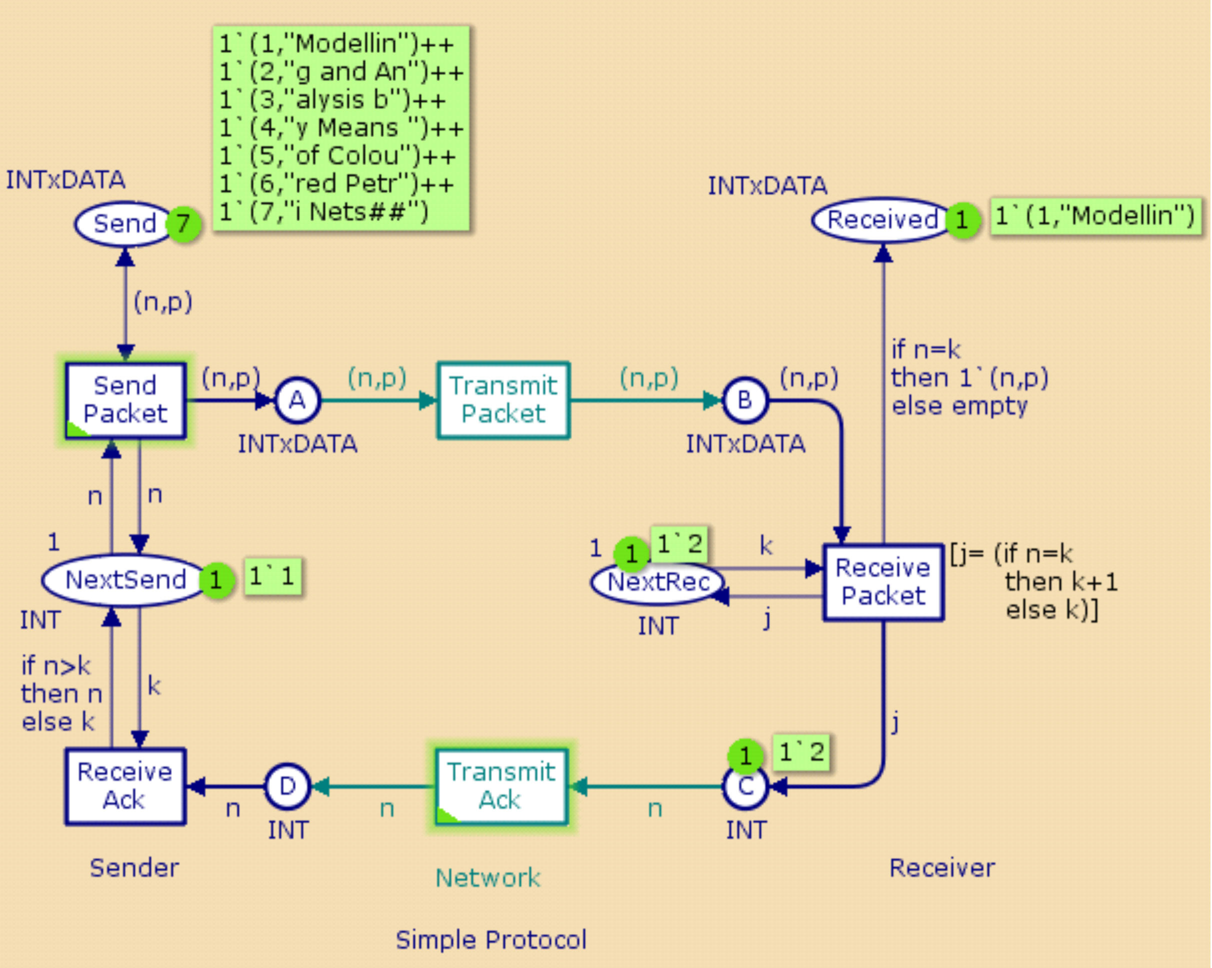
arc expression

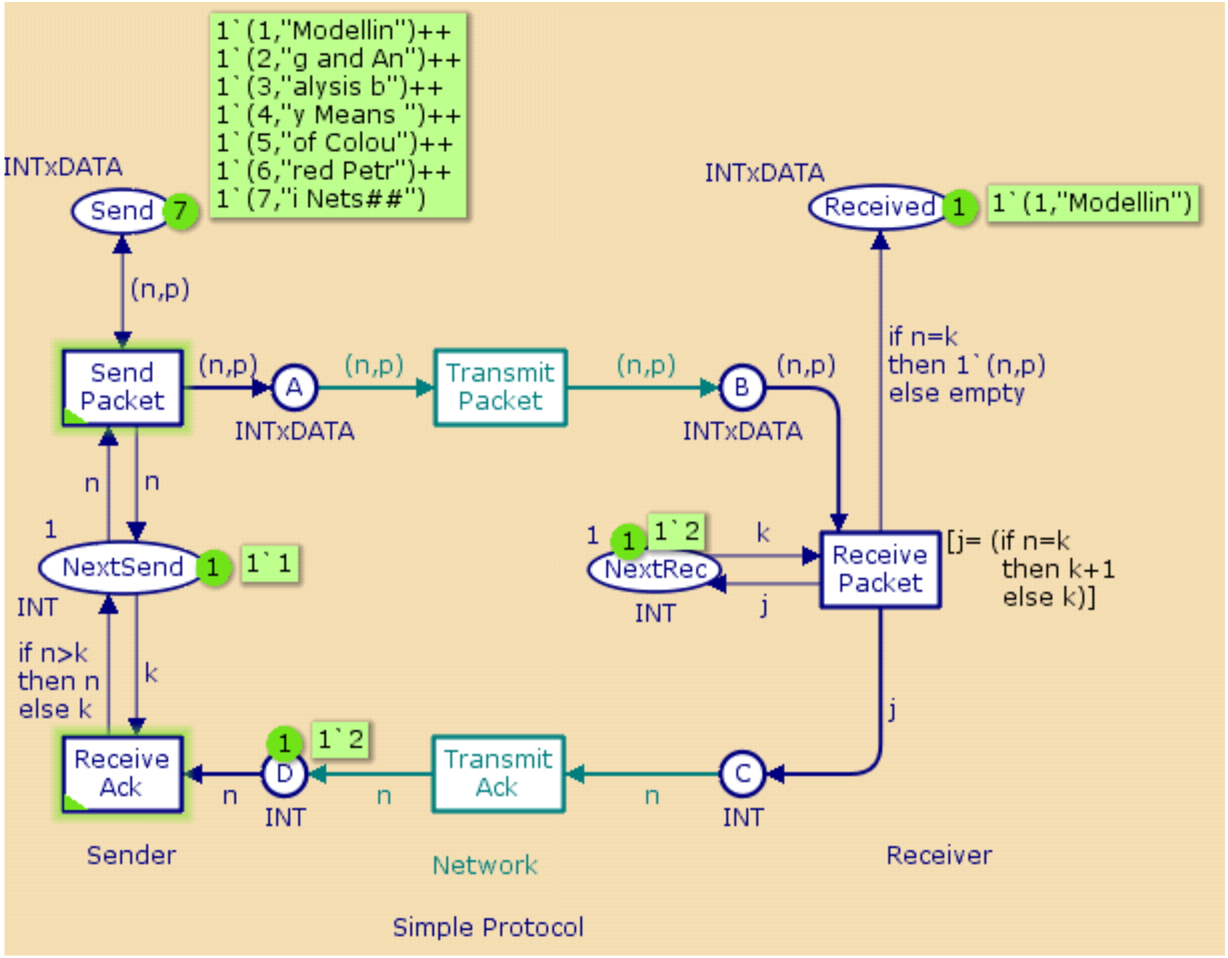
guard expression

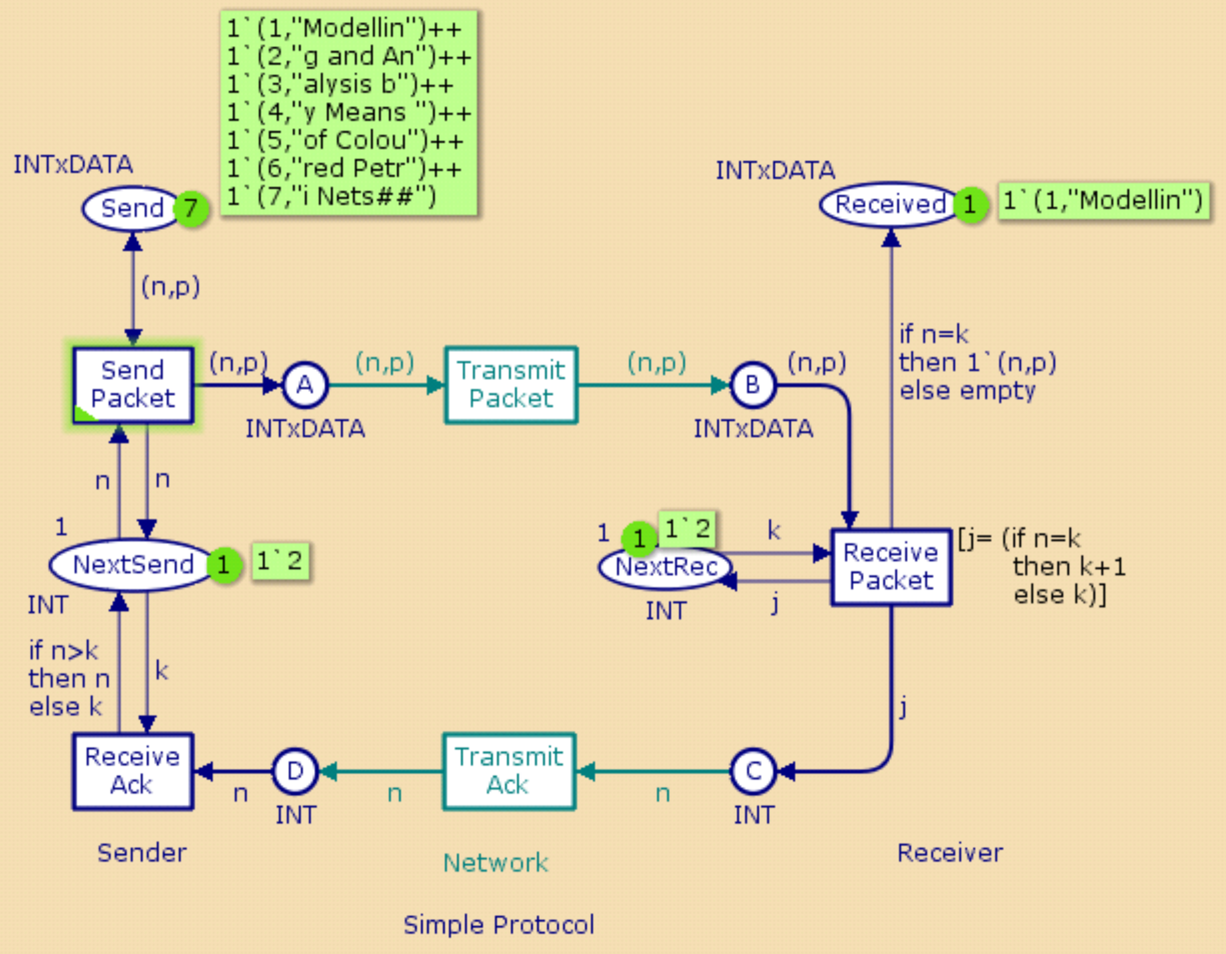
token

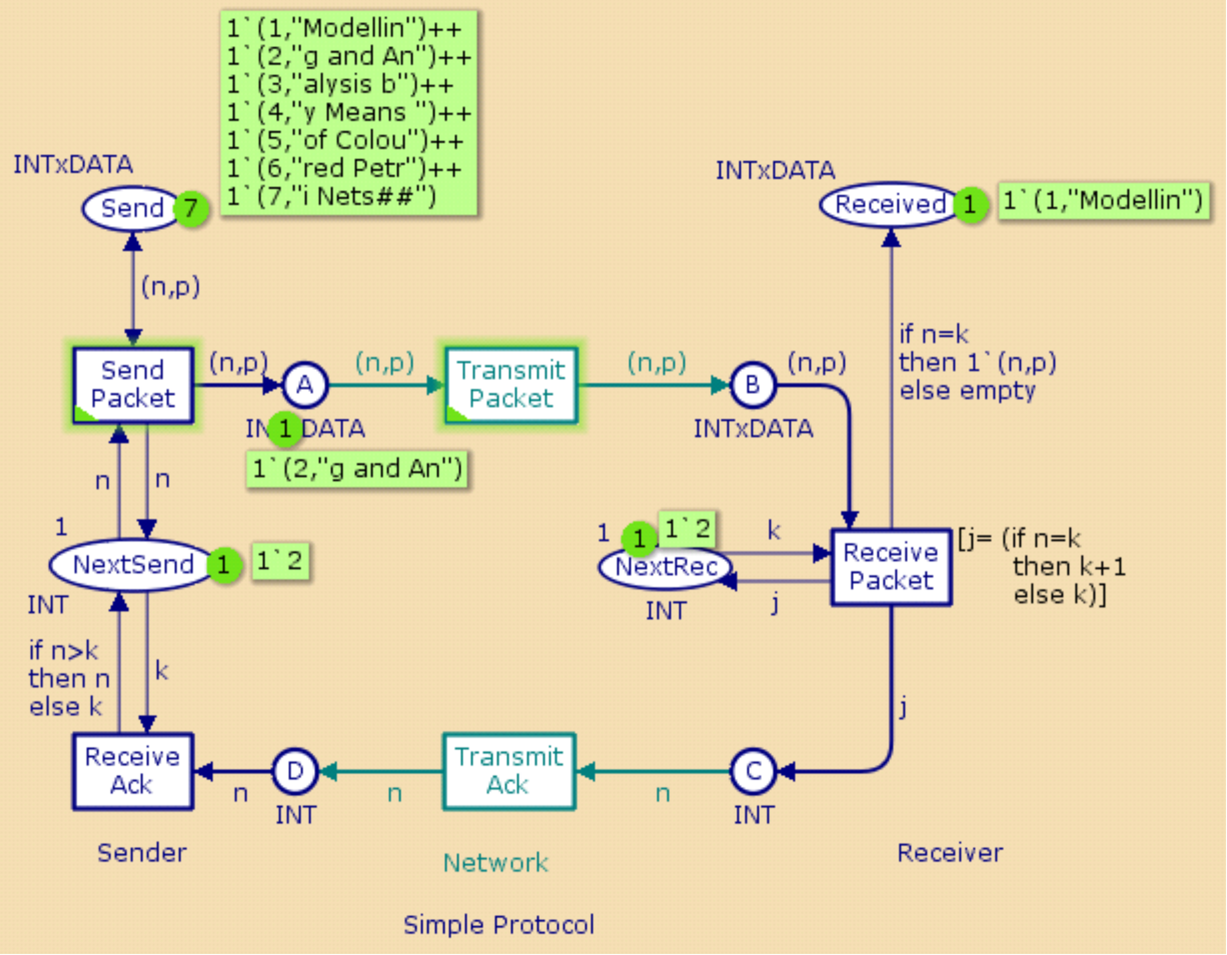

1`(1,"Modellin")++
1`(2,"g and An")++
1`(3,"alysis b")++
1`(4,"y Means ")++
1`(5,"of Colou")++
1`(6,"red Petr")++
1`(7,"i Nets##")
INTxDATA
Send
7
(n,p)
Send Packet
(n,p)
A
(n,p)
Transmit Packet
(n,p)
B
(n,p)
INTxDATA
Received
if n=k
then 1`(n,p)
else empty
INTxDATA
1
1`(1,"Modellin")
INTxDATA
n
n
1
NextSend
1
1`1
INT
1
1
1`1
k
NextRec
j
INT
Receive Packet
[j= (if n=k
then k+1
else k)]
if n>k
then n
else k
k
j
Receive Ack
n
D
INT
n
Transmit Ack
n
C
INT
Sender
Network
Receiver
Simple Protocol


1`(1,"Modellin")++
1`(2,"g and An")++
1`(3,"alysis b")++
1`(4,"y Means ")++
1`(5,"of Colou")++
1`(6,"red Petr")++
1`(7,"i Nets##")
INTxDATA
Send
7
(n,p)
Send Packet
(n,p)
A
INTxDATA
(n,p)
Transmit Packet
(n,p)
B
(n,p)
1
1`(1,"Modellin")
INTxDATA
INTxDATA
Received
if n=k
then 1`(n,p)
else empty
n
n
1
NextSend
1
1`1
INT
if n>k
then n
else k
k
Receive Ack
n
D
INT
n
Transmit Ack
n
C
INT
j
1
1
1`1
k
NextRec
INT
j
Receive Packet
[j= (if n=k
then k+1
else k)]
Sender
Network
Receiver
Simple Protocol


1`(1,"Modellin")++
1`(2,"g and An")++
1`(3,"alysis b")++
1`(4,"y Means ")++
1`(5,"of Colou")++
1`(6,"red Petr")++
1`(7,"i Nets##")
INTxDATA
Send
7
(n,p)
Send Packet
(n,p)
A
INTxDATA
(n,p)
Transmit Packet
(n,p)
B
INTxDATA
(n,p)
INTxDATA
Received
1
1`(1,"Modellin")
if n=k
then 1`(n,p)
else empty
n
n
1
NextSend
1
1`1
INT
if n>k
then n
else k
k
1
1
1`2
k
NextRec
INT
j
Receive Packet
[j= (if n=k
then k+1
else k)]
j
1
1`2
C
INT
n
Transmit Ack
n
D
INT
n
Receive Ack
Sender
Network
Receiver
Simple Protocol


1`(1,"Modellin")++
1`(2,"g and An")++
1`(3,"alysis b")++
1`(4,"y Means ")++
1`(5,"of Colou")++
1`(6,"red Petr")++
1`(7,"i Nets##")
INTxDATA
Send
7
(n,p)
Send Packet
(n,p)
A
INTxDATA
(n,p)
Transmit Packet
(n,p)
B
INTxDATA
(n,p)
INTxDATA
Received
1
1`(1,"Modellin")
if n=k
then 1`(n,p)
else empty
n
n
1
NextSend
1
1`1
INT
1
1
1`2
k
NextRec
INT
j
Receive Packet
[j= (if n=k
then k+1
else k)]
if n>k
then n
else k
k
Receive Ack
1
1`2
D
n
INT
n
Transmit Ack
n
C
INT
j
Sender
Network
Receiver
Simple Protocol


1`(1,"Modellin")++
1`(2,"g and An")++
1`(3,"alysis b")++
1`(4,"y Means ")++
1`(5,"of Colou")++
1`(6,"red Petr")++
1`(7,"i Nets##")
INTxDATA
Send
7
(n,p)
INTxDATA
Received
1
1`(1,"Modellin")
Send Packet
(n,p)
A
(n,p)
Transmit Packet
(n,p)
B
(n,p)
INTxDATA
INTxDATA
if n=k
then 1`(n,p)
else empty
n
n
1
NextSend
1
1`2
INT
1
1
1`2
k
NextRec
INT
j
Receive Packet
[j= (if n=k
then k+1
else k)]
if n>k
then n
else k
k
j
Receive Ack
n
D
INT
n
Transmit Ack
n
C
INT
Sender
Network
Receiver
Simple Protocol

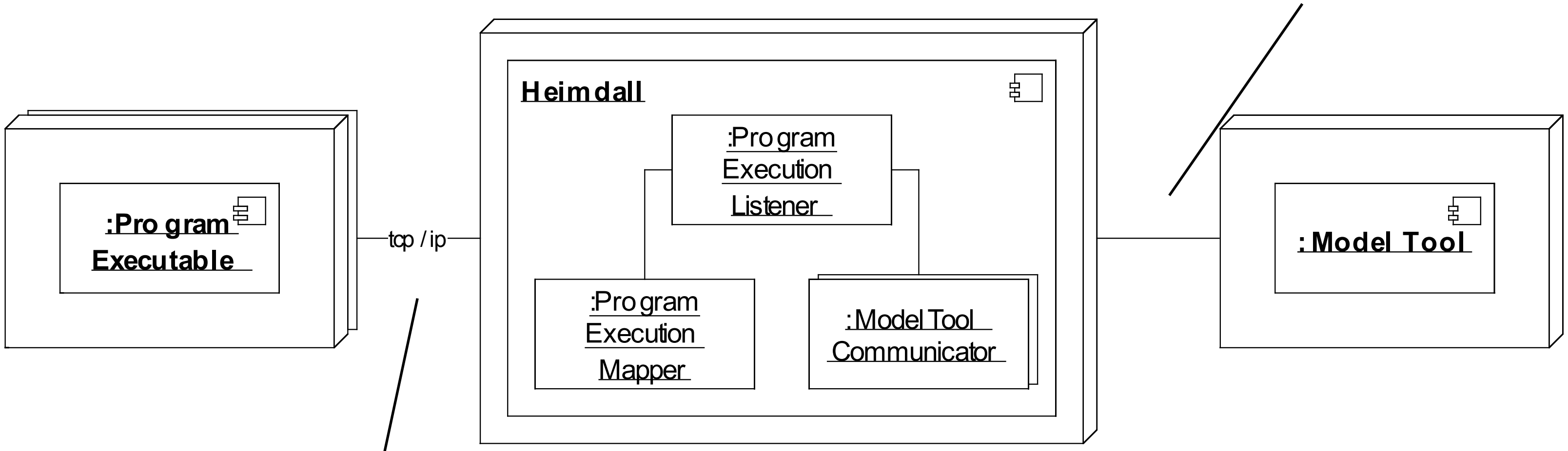

1`(1,"Modellin")++
1`(2,"g and An")++
1`(3,"alysis b")++
1`(4,"y Means ")++
1`(5,"of Colou")++
1`(6,"red Petr")++
1`(7,"i Nets##")
INTxDATA
Send
7
(n,p)
Send Packet
(n,p)
A
(n,p)
Transmit Packet
(n,p)
B
(n,p)
INTxDATA
1
1`(2,"g and An")
INTxDATA
INTxDATA
Received
1
1`(1,"Modellin")
if n=k
then 1`(n,p)
else empty
n
n
1
NextSend
1
1`2
INT
if n>k
then n
else k
k
1
1
1`2
k
NextRec
INT
j
Receive Packet
[j= (if n=k
then k+1
else k)]
j
Receive Ack
n
D
INT
n
Transmit Ack
n
C
INT
Sender
Network
Receiver
Simple Protocol

# The Heimdall Tool

## General idea

- Map program execution events to discrete-state model execution events

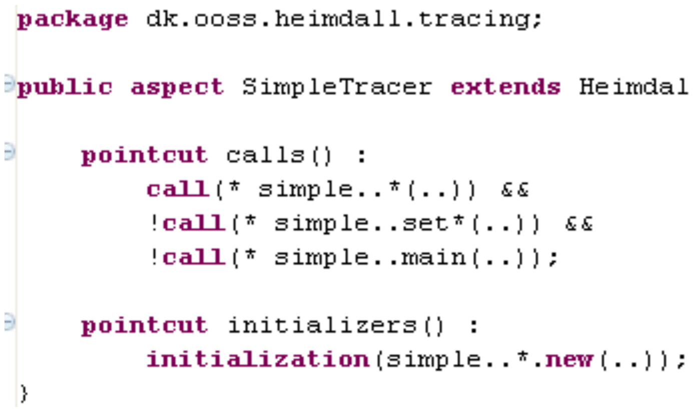
model execution events



program execution events

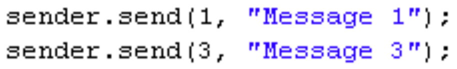
# Heimdall: Tracing Java through AspectJ

## Specific tracing definition



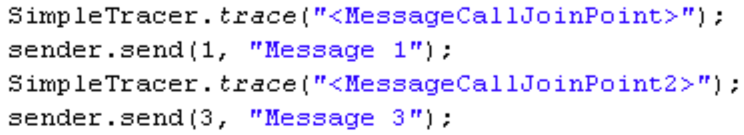
```
package dk.ooss.heimdall.tracing;

public aspect SimpleTracer extends HeimdallTracer {

    pointcut calls() :
        call(* simple..*(..)) &&
        !call(* simple..set*(..)) &&
        !call(* simple..main(..));

    pointcut initializers() :
        initialization(simple..*.new(..));
}
```

## Before weaving



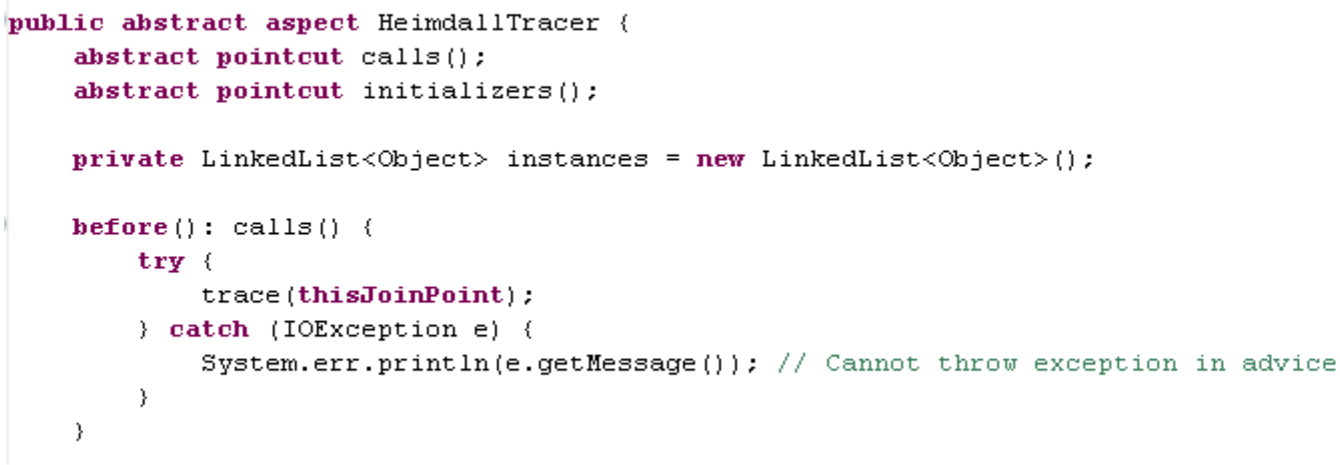
```
sender.send(1, "Message 1");
sender.send(3, "Message 3");
```

## After weaving (conceptually)



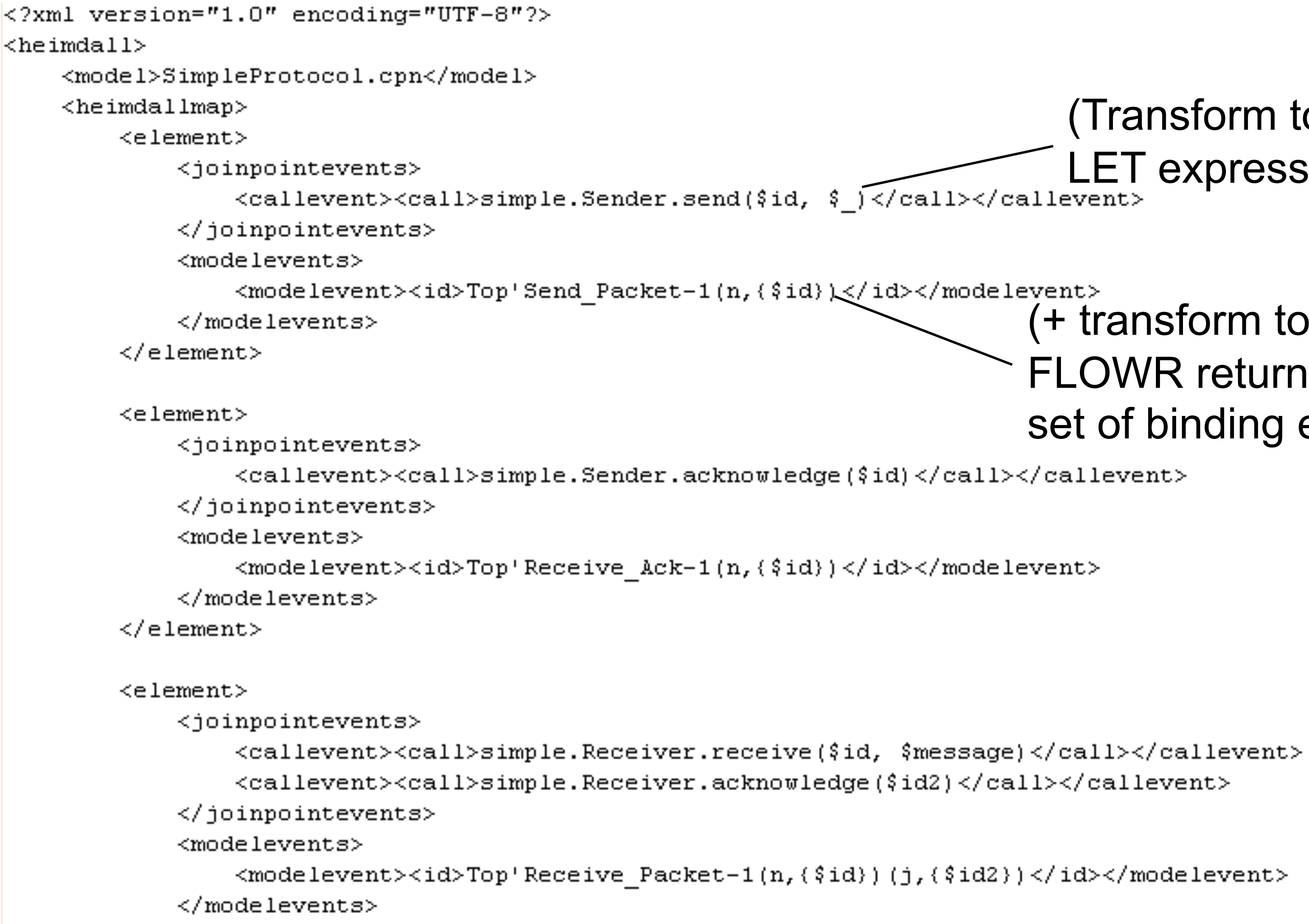
```
SimpleTracer.trace("<MessageCallJoinPoint>");
sender.send(1, "Message 1");
SimpleTracer.trace("<MessageCallJoinPoint2>");
sender.send(3, "Message 3");
```

## General tracing definition



```
public abstract aspect HeimdallTracer {
    abstract pointcut calls();
    abstract pointcut initializers();

    private LinkedList<Object> instances = new LinkedList<Object>();

    before(): calls() {
        try {
            trace(thisJoinPoint);
        } catch (IOException e) {
            System.err.println(e.getMessage()); // Cannot throw exception in advice
        }
    }
```

# Heimdall: Example Mapping



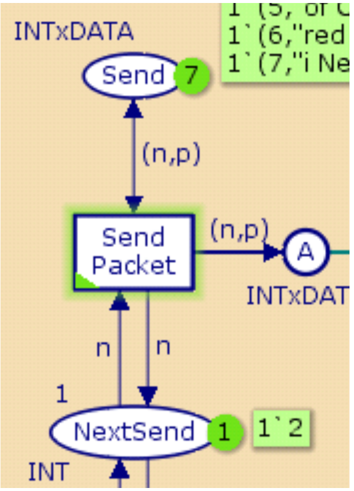
```
<?xml version="1.0" encoding="UTF-8"?>
<heimdall>
    <model>SimpleProtocol.cpn</model>
    <heimdallmap>
        <element>
            <joinpointevents>
                <callevent><call>simple.Sender.send($id, $_)</call></callevent>
            </joinpointevents>
            <modelevents>
                <modelevent><id>Top'Send_Packet-1(n,($id))</id></modelevent>
            </modelevents>
        </element>

        <element>
            <joinpointevents>
                <callevent><call>simple.Sender.acknowledge($id)</call></callevent>
            </joinpointevents>
            <modelevents>
                <modelevent><id>Top'Receive_Ack-1(n,($id))</id></modelevent>
            </modelevents>
        </element>

        <element>
            <joinpointevents>
                <callevent><call>simple.Receiver.receive($id, $message)</call></callevent>
                <callevent><call>simple.Receiver.acknowledge($id2)</call></callevent>
            </joinpointevents>
            <modelevents>
                <modelevent><id>Top'Receive_Packet-1(n,($id))(j,($id2))</id></modelevent>
            </modelevents>
```

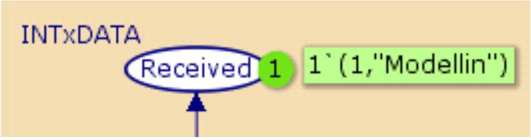
(Transform to XQuery
LET expressions)

(+ transform to XQuery
FLOWR returning
set of binding elements)

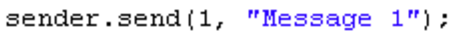
# Heimdall: Results

3)



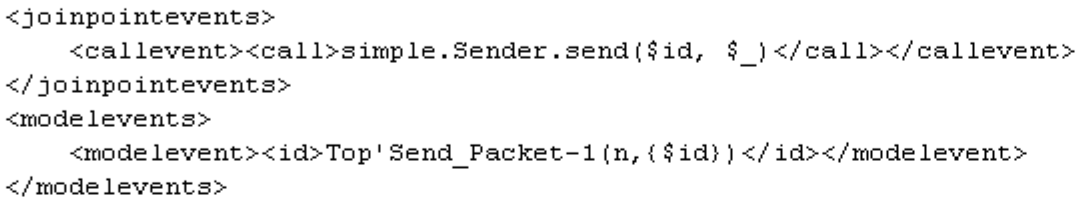


1)



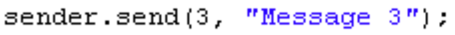
```
sender.send(1, "Message 1");
```

2)



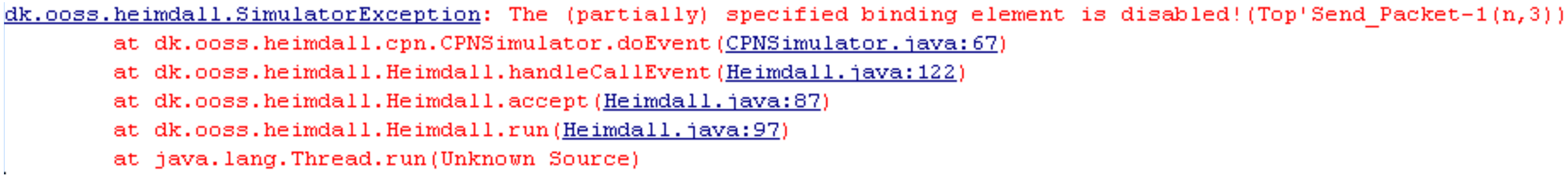
```
<joinpointevents>
    <callevent><call>simple.Sender.send($id, $_)</call></callevent>
</joinpointevents>
<modelevents>
    <modelevent><id>Top'Send_Packet-1(n,($id))</id></modelevent>
</modelevents>
```

4)



```
sender.send(3, "Message 3");
```

5)



```
dk.ooss.heimdall.SimulatorException: The (partially) specified binding element is disabled!(Top'Send_Packet-1(n,3))
        at dk.ooss.heimdall.cpn.CPNSimulator.doEvent(CPNSimulator.java:67)
        at dk.ooss.heimdall.Heimdall.handleCallEvent(Heimdall.java:122)
        at dk.ooss.heimdall.Heimdall.accept(Heimdall.java:87)
        at dk.ooss.heimdall.Heimdall.run(Heimdall.java:97)
        at java.lang.Thread.run(Unknown Source)
```

# Characteristics of Architectural Prototypes

- *Architectural prototypes are constructed for exploration and learning of the architectural design space*.
- *Architectural prototyping addresses issues regarding architectural quality attributes* in the target system.
- *Architectural prototypes do not provide functionality per se*.
- *Architectural prototypes typically address architectural risks*.
- *Architectural prototypes address the problem of knowledge transfer and architectural conformance*.

# Conclusion

Use architectural prototypes and prototyping…

APs is a viable analysis technique because

- Executable software does not overlook aspects ☺
- Cost-efficient
- Demonstrate concerns clearly to stakeholders
- Address architectural conformance when used as basis in constructive phase

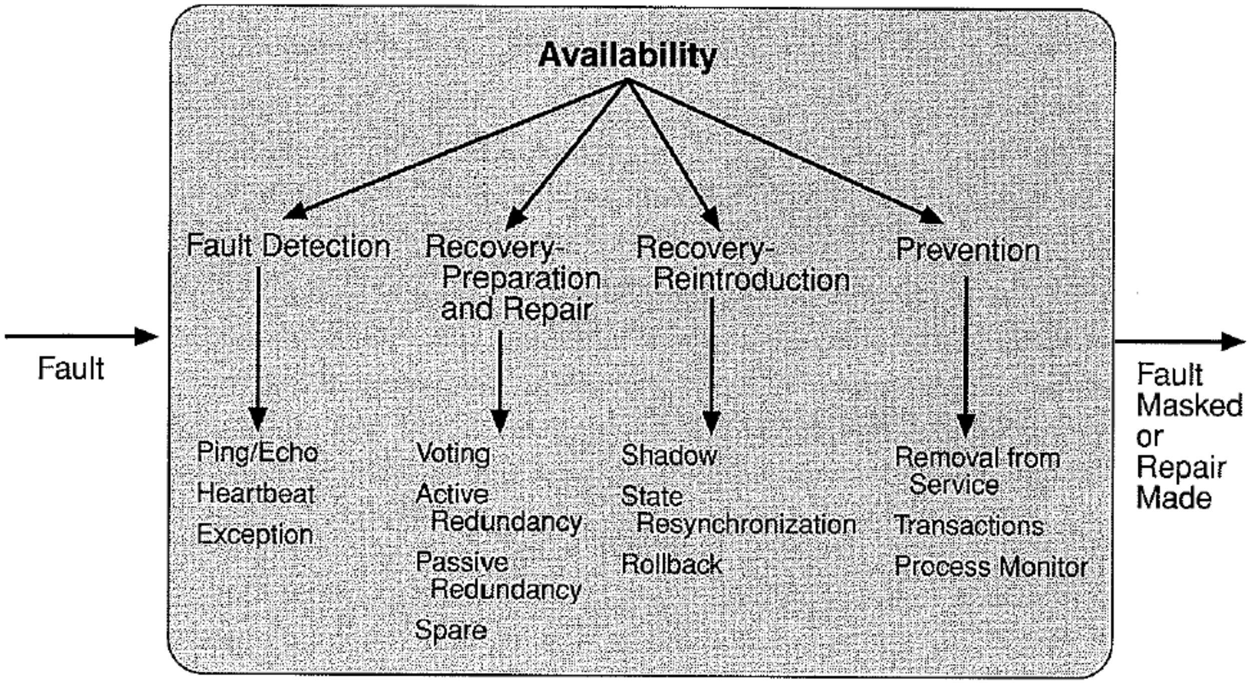
AP as technique is a complement

- Not a substitute for more analytical techniques…

# Availability

## AP is a viable tool

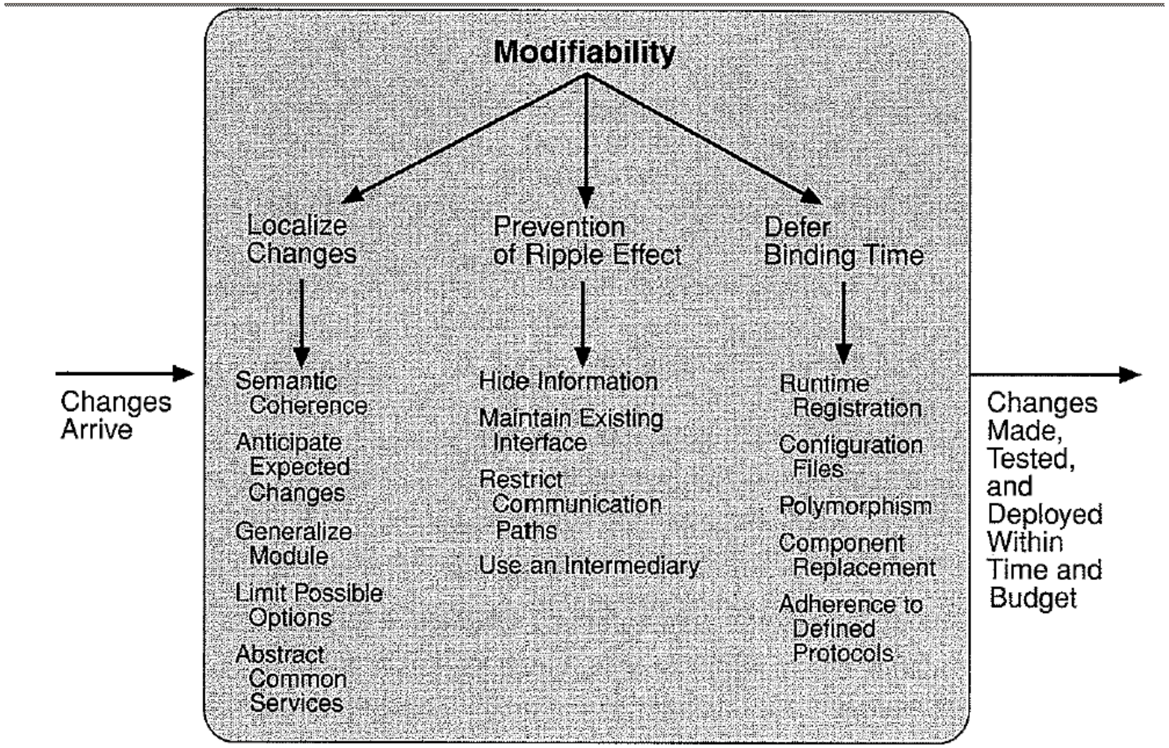
- Availability tactics design space exploration via APs
- Experimental APs for estimation



# Modifiability

APs definition states: ”executable systems”

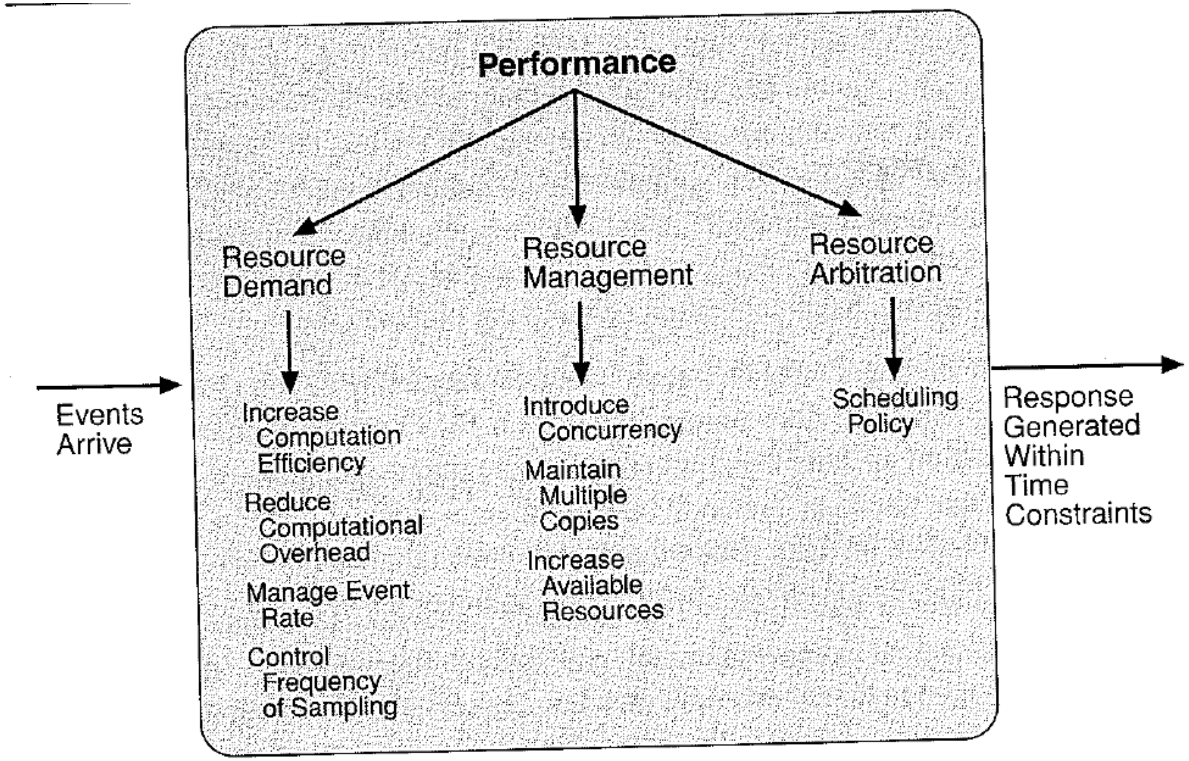
- Per se AP cannot address design-time modifiability
- … but we got the source code for analysis!



# Performance

AP may be viable tool

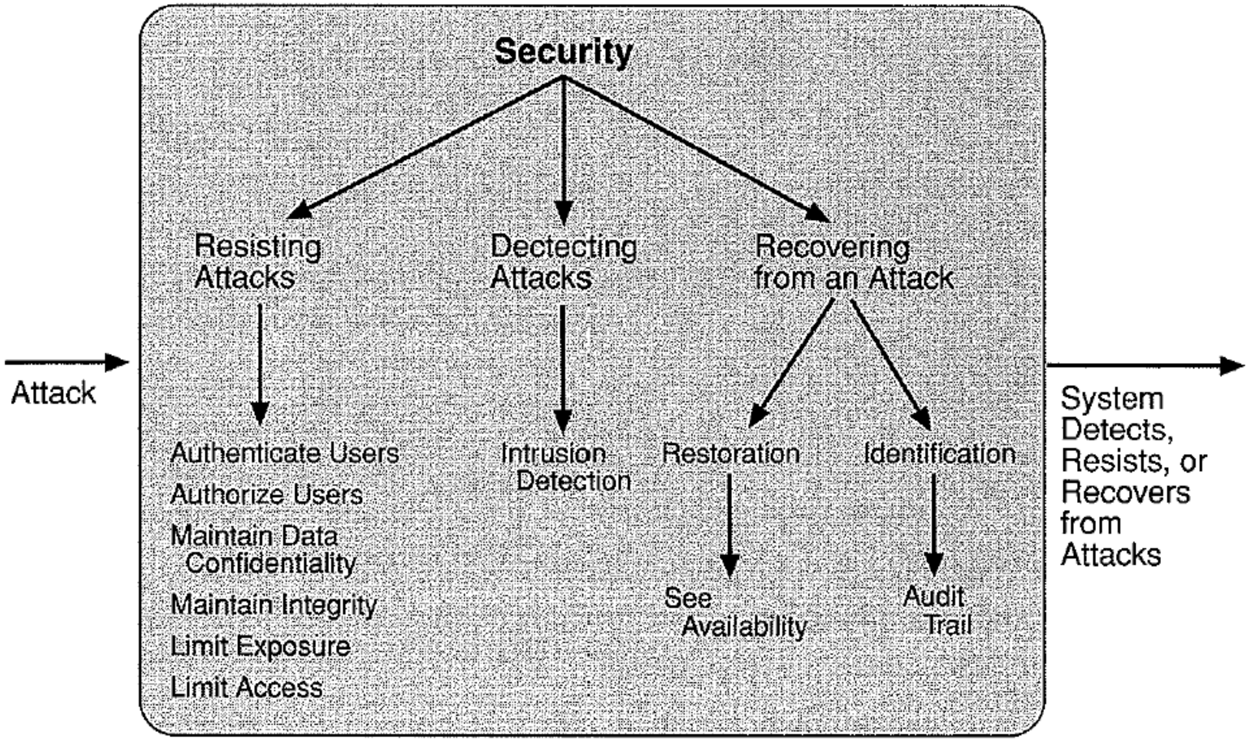
- Under the constraint that accurate stress by *functionality* can be simulated


Performance
Resource Demand
Resource Management
Resource Arbitration
Events Arrive
Increase Computation Efficiency
Reduce Computational Overhead
Manage Event Rate
Control Frequency of Sampling
Introduce Concurrency
Maintain Multiple Copies
Increase Available Resources
Scheduling Policy
Response Generated Within Time Constraints

# Security

Not in the general case but...

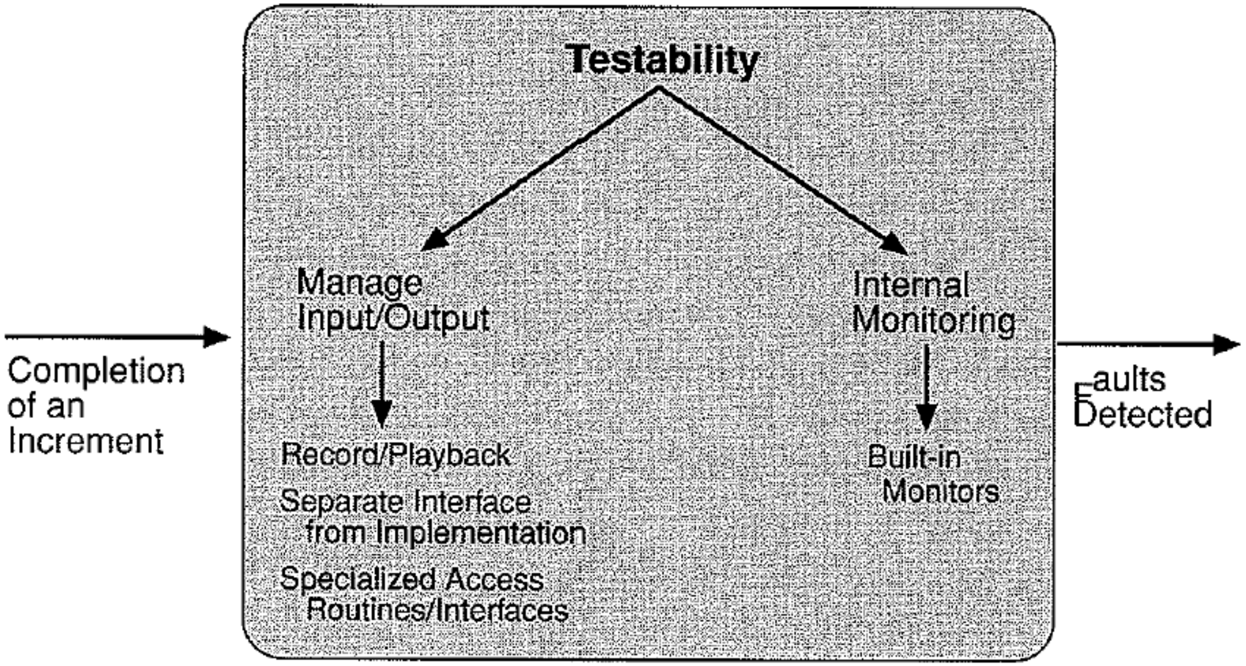
- Denial of service?



# Testability

AP is a viable although not obvious tool

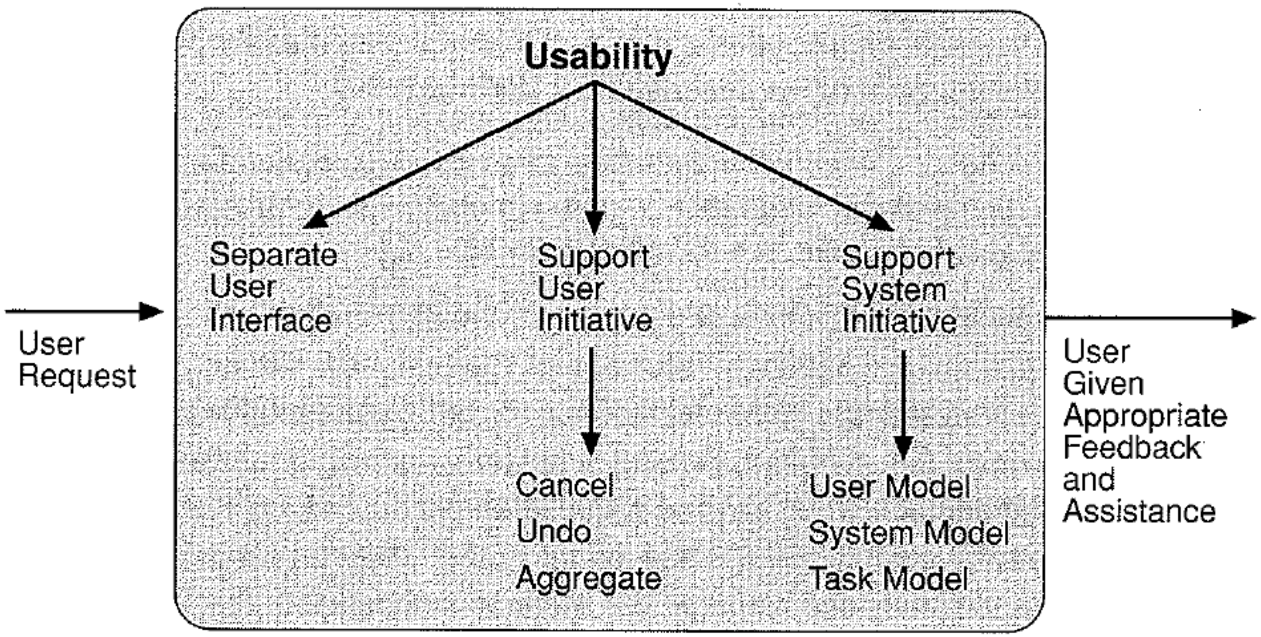
- AP's strength in the balancing of opposing qualities?
- Monitoring cost performance – what is the balance…?


Testability
Manage Input/Output
Record/Playback
Separate Interface from Implementation
Specialized Access Routines/Interfaces
Internal Monitoring
Built-in Monitors
Completion of an Increment
Faults Detected

# Usability

Yes

- E.g., separate user interface
- E.g., building systems models



# Business

Business quality attributes

- Time to market
  - AP basis for analysis of reuse
  - AP delimits modules of independent development (modifiability)
- Cost/benefit
  - Coupled to risk
  - AP addresses risk.
- Projected lifetime
  - Indirectly through modifiability
- Targeted market
  - Coupled to system QAs
- Rollout schedule
- Integration with legacy systems

# Architecture

## Architecture quality attributes

- Conceptual integrity
  - Explore whether a set of concepts are suitable for whole architecture
  - APs are good for transferring architecture-as-designed into architecture-as-built and thus preserve conceptual integrity
- Correctness and completeness
  - Depending on the AP but it could be addressed by an AP that *really* is the full blueprint of the final system
- Buildability
  - Argues this as one of the main advantages of APs…!

# APs vs Quality Attributes

Difficult to say *nay* or *yeah*

- APs may address ’sub-issues’ of a given QA and may not address other ’sub-issues’

APs as one of several available tools for analysis

- Not the silver bullet
- Appropriateness must be considered.

But – *seems* viable for all QAs.

- Problem: How to reject this hypothesis ☺ ?