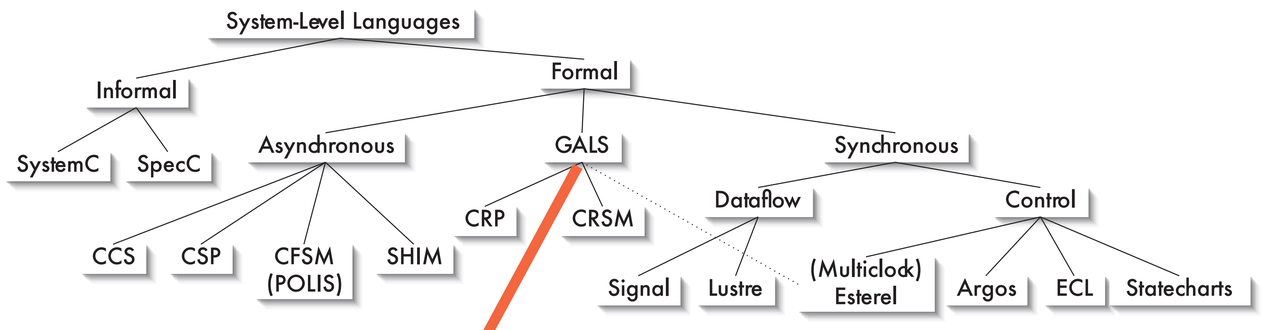
A classification and comparison of a few popular system-level languages



The six major comparison criteria	SpecC	SystemC	Esterel	ECL	CRSM	SHIM	SystemJ
Separation of computation from communication	+++	++	++	++	++	++	+++
Mix of a range of semantics or MoCs	+	++	-	-	++	+	++
Behavioural hierarchy	+	-	+++	+++	+++	+	+++
Support for exceptions and exception handling	++	-	+++	+++	++	-	+++
Mix of data and control dominated processing	++	++	+	++	-	++	++
Support for formal verification	-	-	+++	++	+++	+++	+

The SystemJ Approach to System-level Design

MemoCode 2006

Flavius Gruian[1], Partha Roop[2], Zoran Salcic[2], Ivan Radojevic[2]

1. flavius.gruian@cs.lth.se
Dept. of Computer Science
Lund Institute of Technology
Sweden

LUNDS UNIVERSITET
Lunds Tekniska Högskola

2. {p.roop,z.salcic,i.radojevic}@auckland.ac.nz
Dept. of Electrical and Computer Engineering
The University of Auckland
New Zealand

THE UNIVERSITY OF AUCKLAND
NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau

SystemJ, the language
designed for specification, modeling and synthesis of GALS systems

Java
- object orientation
- basic data and control processing
- platform support
  GC, compilers, libraries
- desktop and embedded use

+

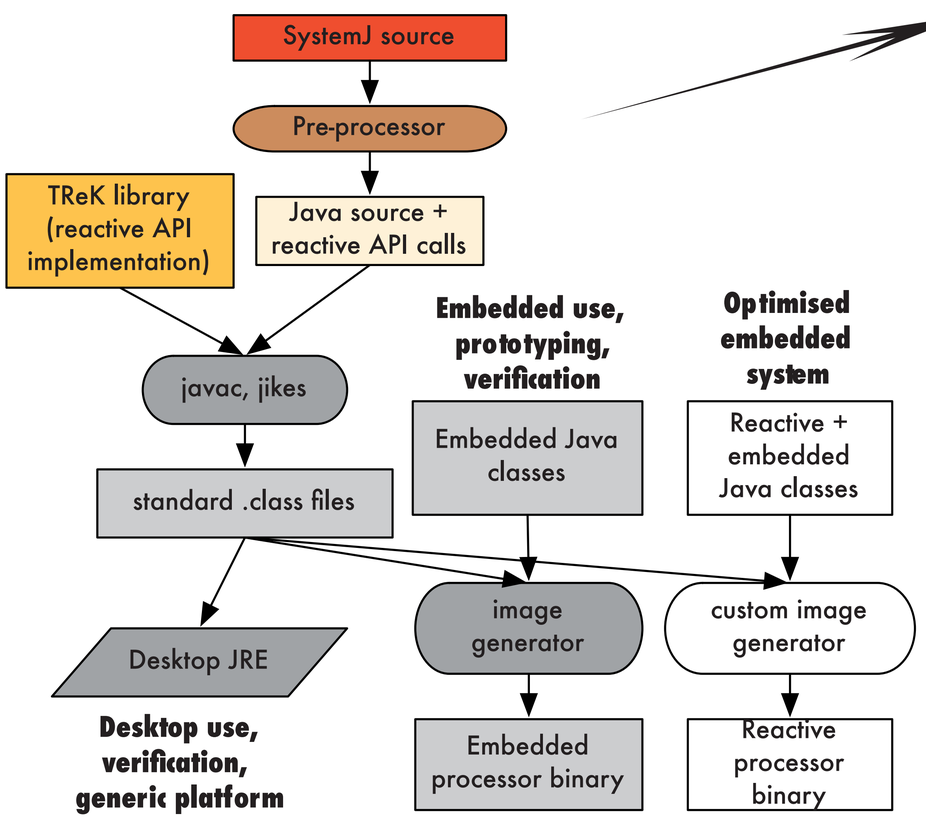
Synchronous Constructs (Esterel-like)
- composition ||
- signal operations
  emit, present, await, RHS, ...
- pure/valued signals
- preemptions
  abort, suspend, trap

+

Asynchronous Constructs
- composition ><
  creates new clock domains
- channel operations
  (blocking) read,write
  also in combination with signal expressions

SystemJ Example: A Protocol Stack

```
1  class Packet {
2    public final static int HDRSIZE = 6;
3    public final static int DATASIZE = 56;
4    public final static int CRCSIZE = 2;
5    public final static int PKTSIZE =
6                HDRSIZE+DATASIZE+CRCSIZE;
7    protected byte packet[];
8    public Packet() { packet = new byte[PKTSIZE]; }
9    public int getCRC()
10   {return packet[PKTSIZE-2]<< 8+packet[PKTSIZE-1];}
11   public int computeCRC()
12   { for (int i=0,crc=0;i <PKTSIZE;i++)
13       crc = (crc ^ packet[i]) << 1;
14     return crc; }
15 } // Packet
16 reaction Assemble(in channel reset,
17   in channel Byte in_byte,
18   out signal Packet outpkt)
19 { int cnt; Packet buffer = new Packet();
20   while (true)
21     abort (reset) {
22       for (cnt = 0; cnt < Packet.PKTSIZE; cnt++) {
23         receive in_byte;
24         buffer.buffer[cnt] = in_byte;
25       }
26       emit outpkt(buffer);
27     } // abort
28 } // reaction Assemble
29 reaction Checkcrc(in channel reset,
30   in signal Packet inpkt,
31   out signal boolean crc_ok)
32 { while (true)
33     abort (reset) {
34       await inpkt;
35       int crc = inpkt.computeCRC();
36       emit crc_ok(crc == inpkt.getCRC());
37     } // abort
38 } // reaction Checkcrc
39 reaction Prochdr(in channel reset,
40   in signal boolean crc_ok,
41   in signal Packet inpkt)
42 { signal kill_check; boolean match_ok;
43   while (true)
44     abort (reset) {
45       await inpkt;
46       { abort (kill_check) {
47           // some lenghty computation,
48           // determining the value of match_ok
49         }
50       } || {
51         await crc_ok;
52         if (!crc_ok) emit kill_check;
53       } // parallel
54       if (crc_ok && match_ok)
55         System.err.println("Address match!");
56     } // abort
57 } // reaction Prochdr
58 reaction TheStack(in channel reset,
59   in channel Byte in_byte)
60 { signal Packet packet; signal Boolean crc_ok;
61   Assemble(reset,in_byte,packet) ||
62   Checkcrc(reset,packet,crc_ok) ||
63   Prochdr(reset,crc_ok,packet);
64 }
65 reaction TestBench(out channel reset,
66   out channel Byte out_byte)
67 { send reset;
68   pause;
69   byte tosend[] = {13,73,127,100, // ... more
70   for (int i=0;i<tosend.length;i++)
71     send out_byte(new Byte(tosend[i]));
72   System.err.println("Test completed.");
73 }
74 system { // a SystemJ GALS
75   channel reset; channel Byte data;
76   TestBench(reset,data) >< TheStack(reset,data); }
```



TReK: A True Reactive Kernel
Java1.5 Library

- offers support for desktop execution of SystemJ specifications
- sixteen classes implementing reactions, signals, channels, clock domains, and scheduling
- extends standard Java threads and employs two queues to achieve the synch-asynch behaviour at run-time
- uses try-catch to handle aborts and traps
- signals and signal operations implemented using bit arrays
- employs generics to implement valued signals and channels

The SystemJ Pre-processor

- translates SystemJ to Java plus TReK calls
- detects syntactic errors, incorrect uses of channels and signals
- introduces signal resolves to help the run-time scheduler
- based on ReRAGs with JastAdd, a Java1.4 frontend and generates Java1.5 code

Examples of SystemJ constructs translated to Java/TReK

await(A or B and not C)	await(new Or(A, new And(B, new Not(C))))
signal byte S	SignalValued<Byte> S = new SignalValued<Byte>("S")
abort(E) { ... }	try { setAbort(E); ... } catch(AbortException ae) {handle(ae, E.getName());} unsetAbort();
send C(5)	C.send(5)

reaction R(in signal int S1,
out signal S2) { ... }
	class R extends Reaction {
	SignalValued<Integer> S1; Signal S2;
	... // more internal signals
	public R(SignalValued<Integer> S1, Signal S2)
	{ this.S1 = S1; this.S2 = S2; }
	public void run() throws TReKException
	{ super.run(); ... }
	}

R1:R(S1,S2) || R2:R(S1,S3)	Reaction rl[] = {new R("R1",S1,S2),new R("R2",S1,S3)}; Signal outputsl[][] = {{S2},{S3}}; synch_parallel(rl,outputsl);

Possible uses of SystemJ
(Java based design flows)

Ongoing & Planned Work

- reactive Java optimized processor (a JOP-ReMIC mix)
- multi-processor architectures (based on HiDRA and Emperor)
- custom JVM for efficient scheduling
- debug and visualisation tools in Eclipse
- formal semantics

Formal Verification

- no formal semantics YET
- can be translated to extended CRSM

The equivalent ECRSM of the Protocol Stack Example