Modeling of Real-time Java programs within Polychrony

(Partly supported by the RNTL project EXPRESSO)

Abdoulaye GAMATIÉ
(joint work with J.-P. TALPIN, B. LE DEZ, D. BERNER and P. LE GUERNIC)
IRISA / INRIA / University of Rennes 1

Background and motivation

- POLYCHRONY: a multi-clocked synchronous design workbench
  - A refinement-based design methodology by formal model transformations
  - Model of an API based on the ARINC 653 specification (APEX services - RTOS functionalities)
- Formal methodology for embedded system design exploration
  - Virtual model of the execution architecture of a software
  - Specification of software functionalities
  - Mapping software on architecture by model transformation

Outline

1. Real-time Java specifications
2. APEX service models within Polychrony
3. From Real-time Java to Polychrony
4. Conclusion

Outline

1. Real-time Java specifications
2. APEX service models within Polychrony
3. From Real-time Java to Polychrony
4. Conclusion

Java for embedded system programming

- A major goal
  - Provide (embedded systems) developpers with a reliable and cost-effective platform-independant environment
- Relevant advantages
  - Object-oriented
  - Portability
  - Automatic Garbage Collector
  - Security

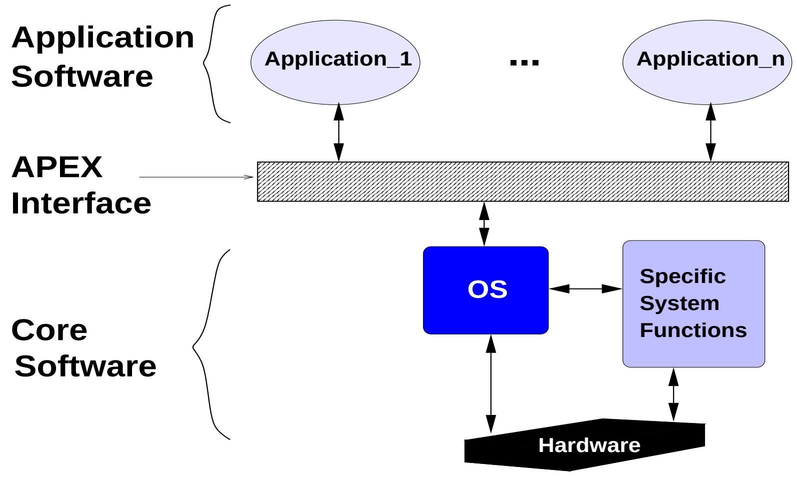
Real-time extensions for Java

- A challenging task
  - Automatic garbage collection and dynamic class loading mechanisms
  - Poor support for real-time threads and real-time scheduling
  - Complexity of the Virtual Machine
- Real-time Java specifications
  - *Java 2 Platform Micro Edition* (J2ME)[Sun00], *Real-Time Specification for Java* (RTSJ)[Bollela+00], *Real-Time Core Extensions*[J-Consortium02]
  - Modification of the original Java semantics
  - Additionnal classes (real-time features)
  - Defining APIs targeted at particular application domains

Outline

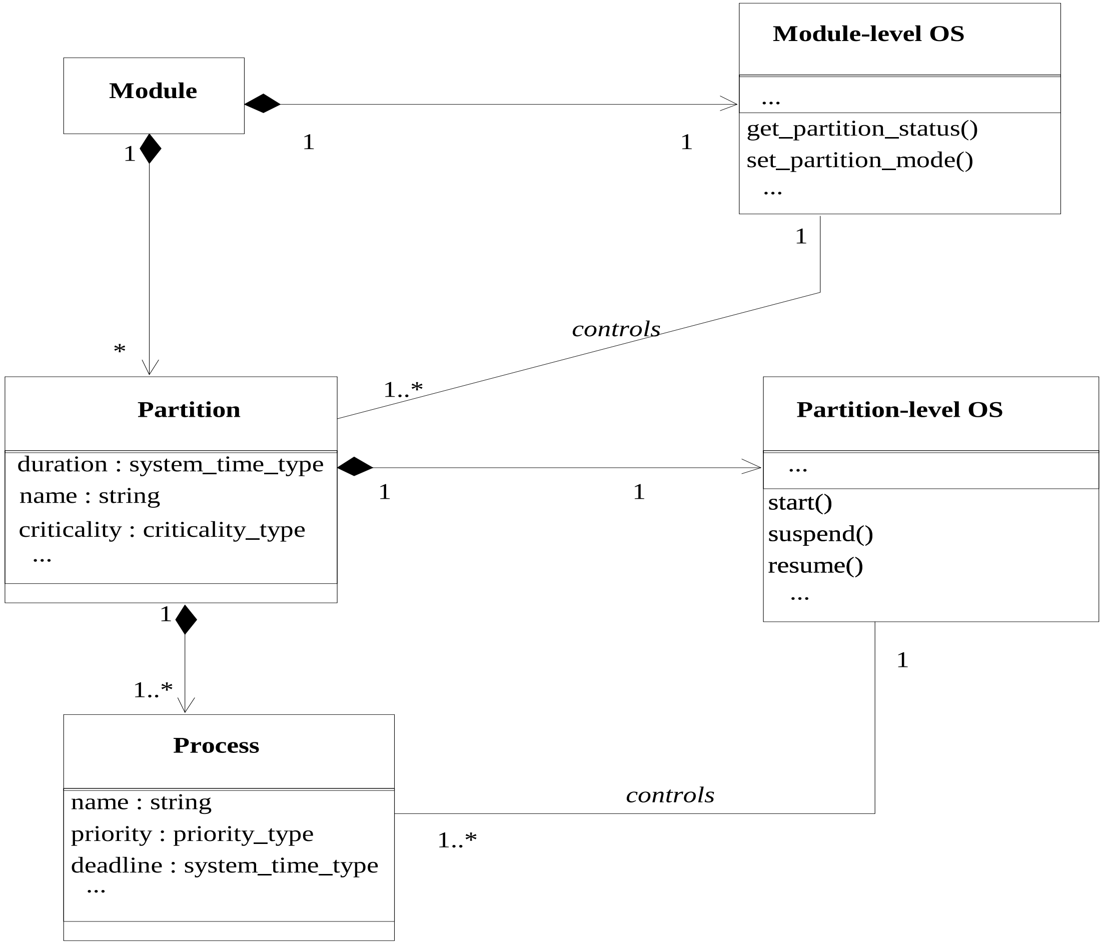
1. Real-time Java specifications
2. APEX service models within Polychrony
3. From Real-time Java to Polychrony
4. Conclusion

APEX services



	Partitions	Processes
Management	`set_partition_mode()`...	`start()`, `suspend()`...
Comm.	`write_sampling_message()`...	`send_buffer()`, `set_event()`...
Synch.		`wait/signal_semaphore()`...
	Time	**Error**
Management	`timed_wait()`, `get_time()`...	`raise_application_error()`...

APEX services (cont'd)



Outline

1. Real-time Java specifications
2. APEX service models within Polychrony
3. From Real-time Java to Polychrony
4. Conclusion

An API: the Ravenscar High Integrity Profile for java [Kwon+02]

- Fit within *J2ME* of Sun
- Meet non functionnal requirements for certification
  - Predictability of memory utilisation
  - Predictability of timing
  - Predictability of control and data flow
- Programming guidelines (e.g. fixed number of threads in any program, resource allocation at the startup, no dynamic memory management during operational mode)

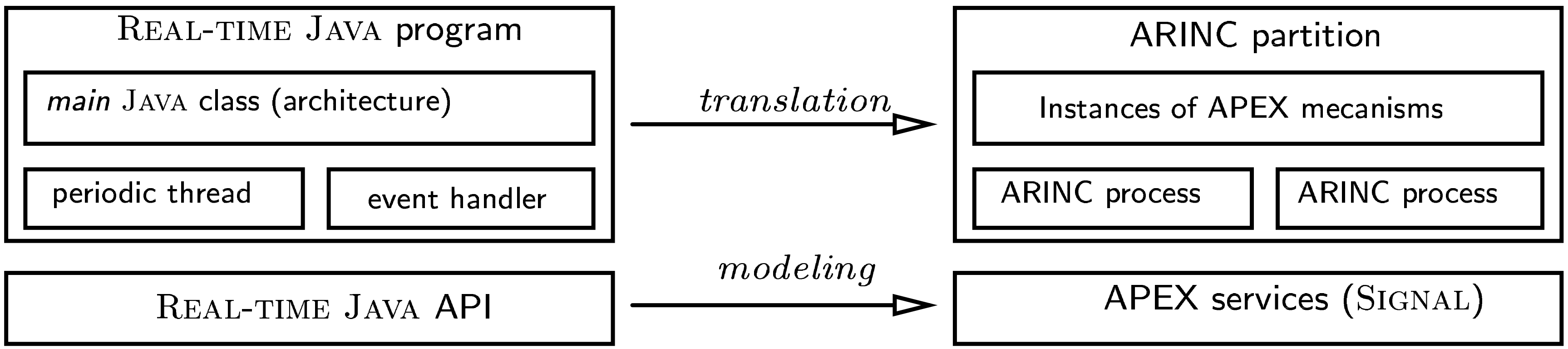
An excerpt of a Real-time Java class hierarchy [Kwon+02]

class SchedulingParameters	class RealtimeThread
class AsyncEventHandler	class NoHeapRealtimeThread
class BoundAsyncEventHandler	class PeriodicThread
class SporadicEventHandler	class MonitorControl

- Restriction to a functional subset (e.g. memory management is not considered)

A methodology for mapping a virtual design on a target

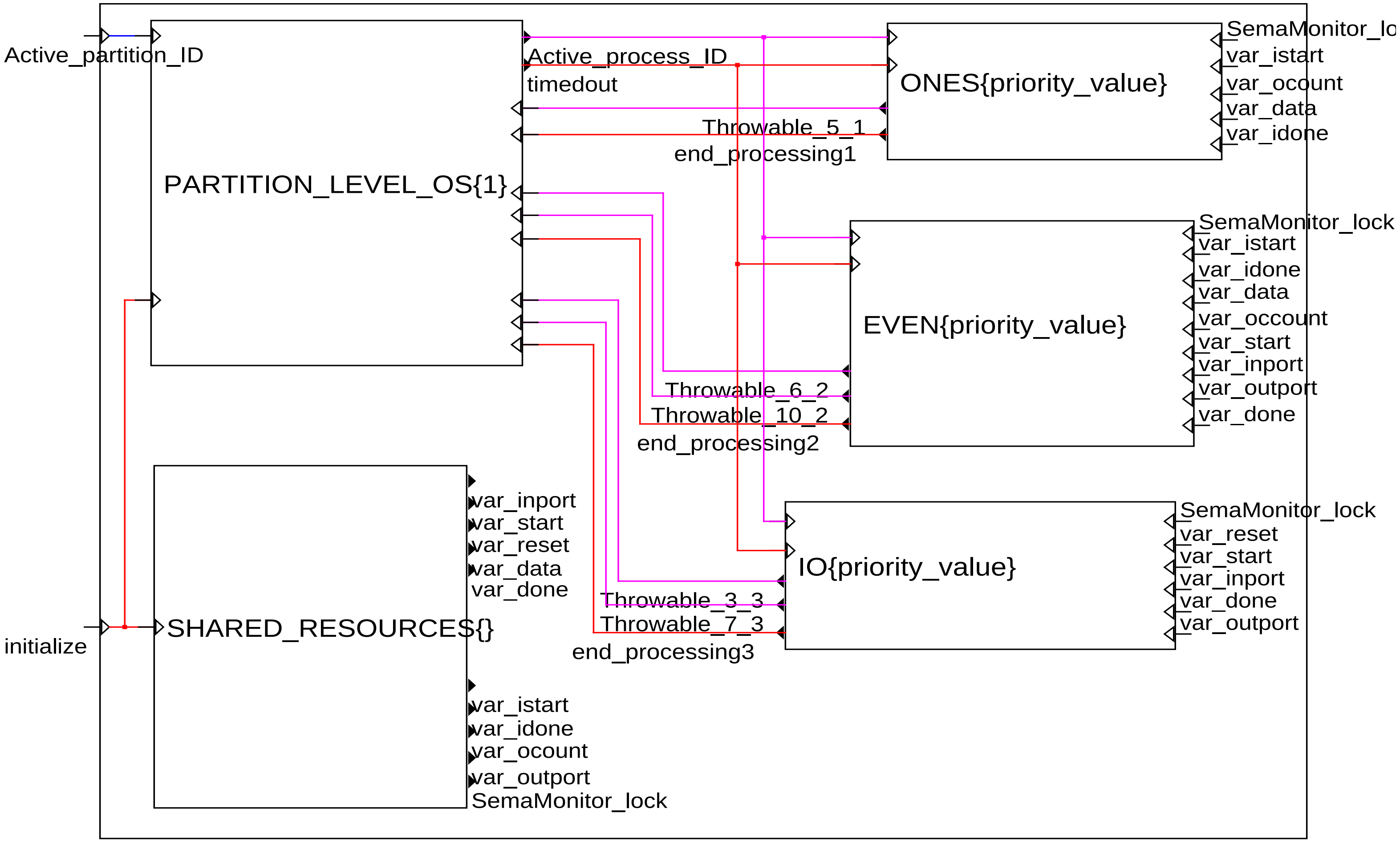
1. Generic model of the runtime system (using APEX services)
2. Analysis of the application architecture (threading and resource parameters)
3. Instanciation of the runtime system model with real-time parameters
4. Translation of periodic real-time threads and sporadic event handlers



An example of program model (Even-Parity Checker)

```
import javax.realtime.*;
class parity {
 public static int inport, outport, data, ocount;
 public static boolean start, done, istart, idone;
 public static void main (String argv[]) {
  io Io = new io ();
  even Even = new even ();
  ones Ones = new ones ();
  Io.start();
  Even.start();
  Ones.start();}
}
```

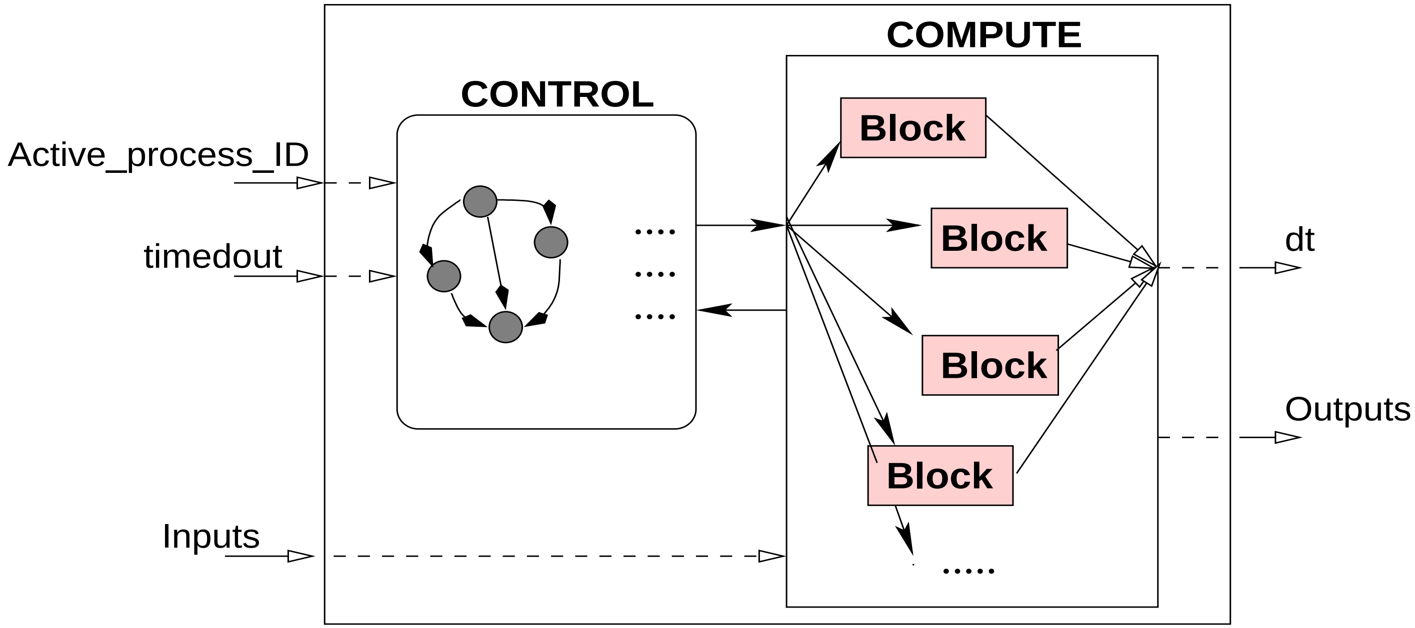
An example of program model (cont'd)



An example of program model (cont'd)

```
class ONES extends PeriodicThread {
  ...
  public void run () {
  int data = 0, ocount = 0;
  synchronized (parity.lock) {
    data  = parity.data;
    ocount = 0;
    while (data != 0) {
     ocount = ocount + (data & 1);
     data = idata >> 1; }
    parity.count = ocount;
    parity.idone = true;}}
}
```

An example of program model (cont'd)



Two relevant aspects in the modeling

- Non functionnal features
  - Real-time parameters (collected by profiling)
- Functionnal features
  - Mainly control part (by using a particular representation)

The JIMPLE format (http://www.sable.mcgill.ca)

- An intermediate representation for Java bytecode
  - Looks like Java, but instructions are in 3-address code form
  - Unstructured: *while*'s, *for*'s, *i f-then-else*'s are broken down into multiple statements; *goto*'s are allowed
  - Like Java, JIMPLE's local variables are typed
- The associated tool : SOOT (a Java bytecode analysis and optimization framework)

Example of a transformation of Java code into JIMPLE

```
if (x+y != z)
    return;
else
    System.out.println("foo");
```

⇒

```
    temp = x+y;
    if temp == z goto label0;
    return;

label0:
        ref = System.out;
        ref.println("foo");
```

Decomposition of statements into blocks

```
process FOO = (? integer x, y z ; ! )
  (| (|                                  % block 1
        temp = x+y;
        if temp == z goto label0;
     |) where integer temp; end
   | (|                                  % block 2
        return; |)
   | (|                                  % block 3
        label0:
        ref = System.out;
        ref.println("foo"); |)
  |)
```

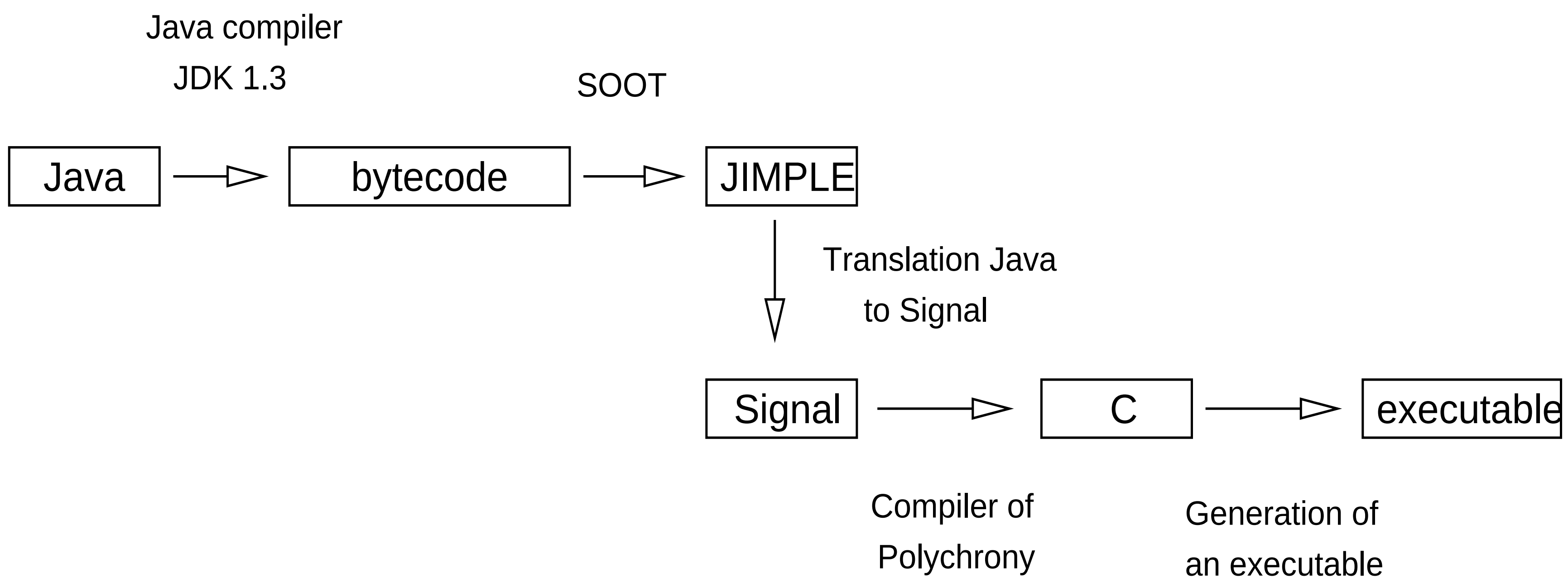
Introduction of activation clocks

```
process FOO = (? integer x, y z ; ! )
  (| (|                                  % block 1
        active when Clk_1
        temp = x+y;
        if temp == z goto label0;  (Clk_3)
      |) where integer temp; end
   | (|                                  % block 2
        active when Clk_2 (≡ Clk_1\Clk_3)
        return; |)
   | (|                                  % block 3
        active when Clk_3
        label0:
        ref = System.out;
        ref.println("foo"); |)
  |)
```

Summary

- Instance of runtime system model to the application's real-time parameters
- Hierarchical decomposition of a thread into critical sections
- Separation of critical sections into control and computation

Implementation



Outline

1. Real-time Java specifications
2. APEX service models within Polychrony
3. From Real-time Java to Polychrony
4. Conclusion

Conclusion

- Formal methods support for embedded system design
  - Modeling of middleware services (Java Virtual Machine API)
  - Modeling of software functionalities (Java threads)
  - A way for applying static analysis techniques
  - Design exploration by model transformations
- Status: a prototype model of real-time JAVA in Polychrony (17K lines of code)
  - Scalability (needs some optimizations)
  - Case studies (more complex programs)
- More details : see INRIA report n 5020 (December 2003)

The end

THANKS!