A Timed-automata based Middleware for Time-critical Multicore Applications.¹

Dario Socci, Peter Poplavko, Saddek Bensalem and Marius Bozga

Verimag - Université Joseph Fourier - Grenoble

SEUS 15

¹Research supported by the European ICT Collaborative Project no. 288175 (CERTAINTY) and no. 332987 (ARROWHEAD).
Real-Time System Design

Timing in Embedded software

- Languages were conceived without any concern on timing[2]
- Different models of computation (MoCs): synchronous languages, timed Petri nets, synchronous dataflow (SDF) and its extensions etc.
- There is a gap between MoCs and real-time scheduling policies[1]

KPN Real-time Scheduling

```
Process p1 {
  while (true) {
    compute1(x,y); read(in1, &x); compute2(x,y); write(out1, &y);
  }
```

Real-Time System Design

Timing in Embedded software

- Languages were conceived without any concern on timing[2]
- Different models of computation (MoCs): synchronous languages, timed Petri nets, synchronous dataflow (SDF) and its extensions etc.
- There is a gap between MoCs and real-time scheduling policies[1]

Proposed Solution

- Compile MoCs and scheduling policies into expressive **backbone language** (timed-automata based)
- Provide a **Runtime Environment** for the backbone language to execute the desired MoC and scheduling

Design flow

2. Compile Model of Computation
3. Translate Hardware Architecture
4. Hardware (BIP)
5. BIP Compiler
6. Binary executable

Multi-core Platform Parameters
WCET Analysis
Multiprocessor Scheduling

system level

software level

hardware level

Multi-thread BIP RTE

Schedulability Validation
System Model (BIP)

D. Socci et al. (UJF / Verimag)
MCPI
SEUS 15
Outline

1. Backbone Language: BIP

2. Model of Computation: FPPN

3. Compiling MoC into BIP
   - Compiling The processes
   - Compiling the Inter-process Channels

4. Compiling The scheduling policy into BIP

5. Implementation and Experiments
Outline

1. Backbone Language: BIP
2. Model of Computation: FPPN
3. Compiling MoC into BIP
   - Compiling The processes
   - Compiling the Inter-process Channels
4. Compiling The scheduling policy into BIP
5. Implementation and Experiments
The Backbone Language: BIP

- **BIP**: component-based and automata-based language.

Behavior - Interaction - Priority

- **Behavior**: Specifies the **components** as timed-automata
- **Interaction** + **Priorities** = **glue**
- **Interaction**: Specifies how components interact
- **Priority**: Specifies the order of concurrent interactions

Priorities (Conflict resolution)
Interaction Model (Collaboration)
example component that generates output data every $T_A$ seconds

- clock $x$ measures the time
- conditions for transitions are shown in blue
- actions are shown in red
- action StartA is connected to an external port
BIP Example - Composition

- The green line is an Interaction
- Interactions may occur when all actions connected to involved port are ready
  - In PeriodicA $x = T_A$
  - CPU1 is in state $S_0$
- The thick arrow stands for **continuous transition** (which takes time)
Outline

1. Backbone Language: BIP

2. Model of Computation: FPPN

3. Compiling MoC into BIP
   - Compiling The processes
   - Compiling the Inter-process Channels

4. Compiling The scheduling policy into BIP

5. Implementation and Experiments
Design flow

Model of Computation & Functional Code

system level

Model of Computation

Multi-core Platform Parameters

Multi-thread BIP RTE

Hardware (BIP)

Compile Model of Computation

Compile Scheduling Policy

Translate Hardware Architecture

Software (BIP)

BIP Compiler

Binary executable

Schedulability Validation

system level

Multi-core Scheduling

WCET Analysis

Hardware (BIP)

Software level

BIP Compiler

Schedulability Validation
As an example MoC we use FPPN

- **Fixed Priority Process Networks (FPPN)**
  - deterministic thanks to priorities between events.

- **Kahn Process Networks (KPN)**
  - Widely used in literature deterministic

- **Reactive Process Networks (RPN)**
  - extend KPN by events non-deterministic

Restricts

Extends
**Processes**: events are “embedded” (process frequency)

**Internal Data channels**: Blackboard and FIFO - others can be defined

**External Data channels**: Sensors and Actuators
Outline

1. Backbone Language: BIP
2. Model of Computation: FPPN
3. Compiling MoC into BIP
   - Compiling The processes
   - Compiling the Inter-process Channels
4. Compiling The scheduling policy into BIP
5. Implementation and Experiments
Compiling MoC into BIP

Design flow

- Model of Computation & Functional Code
- WCET Analysis
- Multi-core Platform Parameters
- Multiprocessor Scheduling
- Compile Model of Computation
- Compile Scheduling Policy
- Translate Hardware Architecture
- Hardware (BIP)
- System Model (BIP)
- Schedulability Validation

- Software (BIP)
- BIP Compiler
- Binary executable
  - Multi-thread BIP RTE
  - Multi-thread BIP RTE

- Model of Computation & Functional Code
- WCET Analysis
- Multi-core Platform Parameters
- Multiprocessor Scheduling
- Compile Model of Computation
- Compile Scheduling Policy
- Translate Hardware Architecture
- Hardware (BIP)
- System Model (BIP)
- Schedulability Validation

- Software (BIP)
- BIP Compiler
- Binary executable
  - Multi-thread BIP RTE
  - Multi-thread BIP RTE
Outline

1. Backbone Language: BIP

2. Model of Computation: FPPN

3. Compiling MoC into BIP
   - Compiling The processes
   - Compiling the Inter-process Channels

4. Compiling The scheduling policy into BIP

5. Implementation and Experiments
Process Example

```c
struct Square::internal_var {
    int index = 0;
    int length = 200;
}

void Square::Job(internal_var *X) {
    float x, y;
    if (X->index < X->length) {
        read(PORT_IN, &x);
        if (PORT_IN.valid) {
            y = x * x;
            write(PORT_OUT, &y);
        }
    }
    X->index = X->index + 1;
}
```

- **Read** and **Write** primitives access to channels
Process

```cpp
struct Square::internal_var {
    int index = 0;
    int length = 200;
};

void Square::Job(internal_var *X) {
    float x, y;
    if (X->index < X->length) {
        read(PORT_IN, &x);
        if (PORT_IN.valid) {
            y = x * x;
            write(PORT_OUT, &y);
        }
    }
    X->index = X->index + 1;
}
```

Diagram:

Process()
- internal variables: index, len, x, y
- $R_{IN}, W_{IN}$: reference (pointer) to the data read and written
- $V_{IN}$: data validity flag

- start
  - index := 0
  - Len := 200
- ifc := index < len
- read $R_{IN}, V_{IN}$
- read Ack
- write $W_{IN}$
- write Ack
- if (V_{IN}) x := *$R_{IN}$
- end
  - Index := index + 1
  - *$W_{IN}$ := y
  - Write Ack
Outline

1. Backbone Language: BIP

2. Model of Computation: FPPN

3. Compiling MoC into BIP
   - Compiling The processes
   - Compiling the Inter-process Channels

4. Compiling The scheduling policy into BIP

5. Implementation and Experiments
BlackBoard

- Wait for request in INIT state
- Wait for the Read (Write) to be performed in state R (W)
- Grants Read_Ack (Write_Ack)

\[ \text{BlackBoard}(S) \]

\[ S = \text{size of data} \]
\[ W_{OUT} = R_{OUT} - \text{references to the (same) variable of size } S \]
\[ V_{OUT} = \text{validity flag} \]
Compiling MoC into BIP

Compiling the Inter-process Channels

**FIFO**

- **Read_REQ(ROUT,VOUT)**
- Read_Ack
- **Write_REQ(WOUT)**
- Write_Ack

**FIFO(S,f)**

- S = size of data
- f = size of queue
- F(S,f) = queue
- ROUT, WOUT = reference to read and write variable
- VOUT = validity flag

- F.Init()
- \( VOUT := false \)
- WOUT := F.Allocate()

- if(VOUT){
  - F.Pop()
  - \( VOUT := \neg F.Empty() \)
  - ROUT := F.Head()
}

D. Socci et al. (UJF / Verimag)
Outline

1. Backbone Language: BIP
2. Model of Computation: FPPN
3. Compiling MoC into BIP
   - Compiling The processes
   - Compiling the Inter-process Channels
4. Compiling The scheduling policy into BIP
5. Implementation and Experiments
Design flow
The Scheduling Policy

- Non-Preemptive
- The schedule is a repeated cycle that is divided into frames
- For each core and frame the scheduling order is determined by a compile-time scheduler and modeled by a simple BIP component
Connecting Frames to Periodic Processes in BIP

- Periodic processes are directly connected to frames
- EventGenerator triggers the arrival of the Process
Sporadic processes are triggered by external events (EventGenerator).

In case the event is not present, the PeriodicServer component tells the frame to schedule the next job.
Outline

1. Backbone Language: BIP

2. Model of Computation: FPPN

3. Compiling MoC into BIP
   - Compiling The processes
   - Compiling the Inter-process Channels

4. Compiling The scheduling policy into BIP

5. Implementation and Experiments
Implementing on Kalray MPPA

- We implemented our design-flow on Kalray MPPA architecture.
- Kalray architecture has 256 cores divided into 16 clusters of 16 cores.
Implementation and Experiments

Mapping on Kalray Cluster

Cycle

Frame f1
Frame f3

thread 0: RT-BIP runtime environment (RTE)

thread 1: middleware components

Process Source/Sink
Cycle
Frame
FF
BB

thread 2..15: process-to-core mapping

thread 2: p2
thread 3: p1
thread 4: p3
thread 5: p4
thread 6: p5
thread 7: p6

schedule components
application components
Experiment
Subsystem of avionics Flight Management System

- GPSCfg is triggered by pilot command (Sporadic Process)
- $Z_1$ and $Z_2$ provide double buffering
Scheduling
The Gantt chart shows the correctness of the approach.

However the middleware takes a big overhead (reductions are possible).
Summary

Conclusion

- We proposed a common approach to program both applications and middleware for time-critical systems
- We demonstrated the approach on a concrete MoC and scheduling policy
- The tool is publicly available
- The proposed approach potentially opens the possibility of unifying different MoCs and scheduling policies in common frameworks.

Future work

- Automatic support of different MoCs and scheduling policies
- Including preemptive scheduling (EDF)
Time for questions!

Thanks for the attention!
Questions?

Tool download:
http://www-verimag.imag.fr/Multicore-Time-Critical-Code,470.html

or simply google:
multicore time-critical verimag
Hauke Fuhrmann, Jens Koch, Jörn Rennhack, and Reinhard von Hanxleden.
Model-based system design of time-triggered architectures—an avionics case study.
In 25th Digital Avionics Systems Conference (DASC’06), Portland, OR, USA, October 2006.

Edward A Lee.
Absolutely positively on time: what would it take?[embedded computing systems].