Event Driven Real-Time Programming

CHESS Review
University of California, Berkeley, USA

May 10, 2004

Arkadeb Ghosal
Joint work with Marco A. Sanvido, Christoph M. Kirsch and Thomas A. Henzinger

University of California, Berkeley

Overview

- Introduction
- Language Features
  - The LET model
  - Language Constructs
  - Event Scoping
- Analysis
- Implementation
- Ongoing Work
Control System

Fuel Injection → Automotive Engine → Speed Sensor

AFR Controller

Implementation Strategies

- Traditional
  - Uses priorities to specify the relative deadlines of software tasks
  - Supports efficient code generation based on scheduling theory
  - Run-time behavior is highly non-deterministic

- Synchronous Languages
  - Esterel, Lustre, Signal
  - Based on synchrony assumption: task computation takes negligible execution time
  - Shows deterministic behavior

- Timed Languages
  - Based on Logical Execution Time (LET) for tasks
  - Giotto
    - Time Triggered
  - xGiotto
    - Event Triggered
    - Scoping of events
The logical and physical execution times are depicted below. The events controlling a task behavior are:

Event generated by the platform:
- start
- preempt
- resume
- completion

Events generated by the environment:
- release
- termination

Logical Execution Time (LET)

- global input ports copied to local ports in logical zero time
- local ports copied to global output ports in logical zero time
- value of output port remains invariant at any instant independent of execution pattern

Time determinism

Value determinism
Reactions and Triggers

- A trigger maps an event to a reaction
  - When the event occurs the reaction is invoked
- A reaction defines
  - New triggers
  - A termination event
- Events of a reaction block are the events of its triggers and the termination event
- A reaction block defines a scope for its events
  - When a trigger is invoked
    - The events of the new reaction block are enabled (active)
    - The events of the callee reaction block become passive (inactive)

**Reaction Block**

<table>
<thead>
<tr>
<th>Reaction Name</th>
<th>Triggers</th>
<th>Until Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>react R1</td>
<td>when [e2] react R2; when [e3] react R3;</td>
<td>until [e7];</td>
</tr>
</tbody>
</table>

Events of R1: e2, e3, e7  
Scope of e2, e3, e7: R1

Reactions and Triggers

![Diagram showing reaction blocks and triggers](image)
Tasks

- Tasks instances are defined by release statements
- Tasks instances
  - released with the invocation of the reaction block
  - terminated with the termination of the reaction block
- LET of the task is given by the life-span of the reaction block

Release Statement

```
react R1 {
  release t1 (i1) (o1);
  when [e2] react R2;
  when [e3] react R3;
} until [e7];
```

Tasks instances are defined by release statements
- Tasks instances
  - released with the invocation of the reaction block
  - terminated with the termination of the reaction block
- LET of the task is given by the life-span of the reaction block

Releasing Tasks

```
port : P_in ;
P_out ;

event
start ;
stop ;
task T (i) output (o)
{ /* compute */}

react R {
  release T (P_in) (P_out) ;
} until [stop]
{ when [start] react R ;

reads P_in

updates P_out

Logical Execution Time

start
stop
```

- p_in is copied to local port of task T
- P_out is updated by the local copy of task T
Tasks

- Tasks are released with the invocation of the reaction block
- Tasks are terminated with the termination of the reaction block

Handling Events

- A reaction block defines a scope; this implicitly denotes the scope of an event
- When an active trigger is invoked, the called reaction becomes the active scope and the caller reaction, the passive scope
- The event of a passive scope can be:
  - Ignored (forget)
  - Postponed until its scope becomes active again (remember)
Parallelism

- A trigger may invoke multiple reaction blocks in parallel.
- When the trigger is invoked all the reactions become active simultaneously.
- The parent block is active only when all the parallel reaction blocks have terminated.

Environment Assumption

- Task T (i) output (o)
  
  \[
  \text{react R \{}
  \text{release T(pin)(pout);}
  \text{when \{start\} react R;}
  \text{when \{now\} react \{} \text{ until \{3ms\};}
  \text{react R \{}
  \text{release T(pin)(pout);}
  \text{when \{now\} react \{} \text{ until \{3ms\};}
  \text{react R \{}
  \text{release T(pin)(pout);}
  \text{when \{now\} react \{} \text{ until \{3ms\};}
  \text{react R \{}
  \text{release T(pin)(pout);}
  \text{when \{now\} react \{} \text{ until \{3ms\};}
  \text{react R \{}
  \text{release T(pin)(pout);}
  \text{when \{now\} react \{} \text{ until \{3ms\};}
  \text{react R \{}
  \text{release T(pin)(pout);}
  \text{when \{now\} react \{} \text{ until \{3ms\};}
  \text{react R \{}
  \text{release T(pin)(pout);}
  \text{when \{now\} react \{} \text{ until \{3ms\};}
  \text{react R \{}
  \text{release T(pin)(pout);}
  \text{when \{now\} react \} }
  \]
xGiotto: Basic Constructs

- Reaction Blocks
  - Basic programming blocks in xGiotto
  - Consists of release statements and trigger statements along with an termination information
  - Releases tasks and invokes triggers
    - `react (reaction block) until [event];`

- Release Instruction
  - Tasks are released with the invocation of the reaction block
  - Tasks are terminated with the termination of the reaction block
    - `release task (input ports) (output ports);`

- Trigger Statements
  - Defines the invoking action associated with an event
    - `when [event] reaction block;`
  - Repetition construct using `whenever`

```
reaction() {
  release task1 (i1) (o1);
  release task2 (i2) (o2);
  when event1 react block1;
  whenever event2 react block2;
} until event;
```

Structuring Events

- Scoping of events
  - A reaction block defines a scope: this implicitly denotes the scope of an event.
  - When an active trigger is invoked, the called reaction becomes the active scope and the caller reaction, the passive scope.
  - The tree of scopes and the state of program variables denotes the state of the program.

- Handling of events (of a passive scope)
  - It may be ignored (`forget`)
  - It may be postponed until its scope becomes active again (`remember`)
  - It may disable trigger statements of all descendent blocks and thus speeding up their termination (`asap`)

- Invoking reactions in parallel
  - Wait-parallelism
  - Asap-parallelism

- Embedding Environment Assumption
  - Event calculus
**The Program Flow**

**Event Filter:**
The Event Filter implements the event scoping mechanism and filters the incoming event. It determines which event needs to be reacted upon depending upon the event qualifiers—forget, remember or asap.

**Reactor:**
The Reactor executes the specified reaction and activates new events (when/whenever/until) and activates and terminates tasks (release).

**Scheduler:**
The Scheduler chooses from the active tasks, a task to be executed on the given platform (CPU). The scheduler generates an event at task completion.

**AFR Controller**

```plaintext
port /* fuel ports */
/* pulse ports */

event teeth; synth; stop;

react channel2 {
    react () until [5ms : teeth];
    when remember [5ms : teeth] react (release set) until [ms];
    react loop react (release reset; dec) until [ms];
} until asap [50ms : 9teeth]

react controller {
    react calcFuel;
    when remember [teeth] react channel1 || react channel2 || ...
} until remember [10teeth]

react calcFuelInj {
    react calcFuelInj;
    when remember [10ms : teeth];
}

react start {
    whenever remember [10teeth] react controller;
} until [stop];
```

10 May CHESS Review 2004

10 May CHESS Review 2004
Analysis

- Platform independent
  - Race Condition Detection
    - Verifying whether a port may be updated by multiple task invocations and thus leading to non-determinism
  - Resource Size Analysis
    - Predicting the run-time memory requirements for executing an xGiotto program: the bound on the size of the event filter and scopes (trigger queue size and active task set size).

- Platform dependant
  - Schedulability Analysis
    - Ensuring that all the task invocations get access to the executing platform at least equal to their worst-case-execution times before their termination

Implementation

- xGiotto Program
- Compiler
- Possible Execution Traces
- Check Race
- Check Time Safety
- WCET
- Code Generator
- Scheduling Strategy
- execute E code
- task release
- Event Filter
- xGiotto Reactions
- Modified Embedded Machine
- xGiotto Tasks
- Scheduler
- Sensors
- Environment
- Actuators
- Platform
Implementation

Ongoing Works

- Implementation
  - Generate code
    - Embedded Virtual Machine code
    - Metropolis Meta-model
  - Porting to RTOS
    - EVM, JVM, OSEK
  - Case studies
    - Porting AFR controller on OSEK

- Analysis
  - Defining the run-time system for xGiotto
  - Schedulability check in time polynomial to the size of the program

- Future Direction
  - Sub-classes of xGiotto
    - Definition, inter relation and effectiveness towards event-driven programming
  - Type Checking
Thank You!