BLECH -
A SAFE SYNCHRONOUS LANGUAGE
FOR EMBEDDED REAL-TIME ROGRAMMING

FRANZ-JOSEF GROSCH

JOINT WORK WITH FRIEDRICH GRETZ AND JENS BRANDT

BOSCH

Bosch – a global company
Research and development 2017

- 62500 associates in research and development
- 125 engineering locations world-wide
- € 7.3 bn research and development expenditure
- € 300 m invested in artificial intelligence

Bosch – a Global Company
Four Business Sectors

Mobility Solutions

- One of the world's largest suppliers of mobility solutions

Industrial Technology

- Leading in drive and control and process technology

Energy & Building Technology

- One of the leading manufacturers of security & communication technology
- Leading manufacturer of energy-efficient heating products and hot-water solutions

Consumer Goods

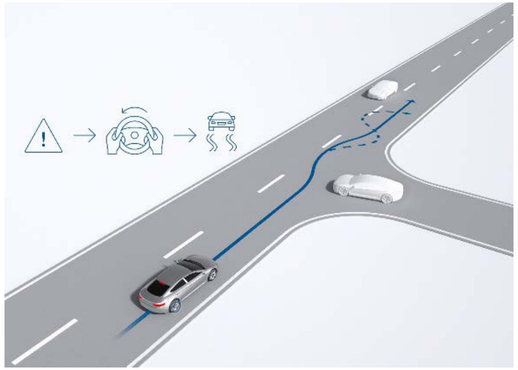
- Leading supplier of power tools and accessories
- Leading supplier of household appliances

Franz-Josef Grosch | 2018-09-11

Bosch technology to enhance quality of life

Example products

- ESP® – the Bosch anti-skidding system



- Engine Control – Gasoline direct injection

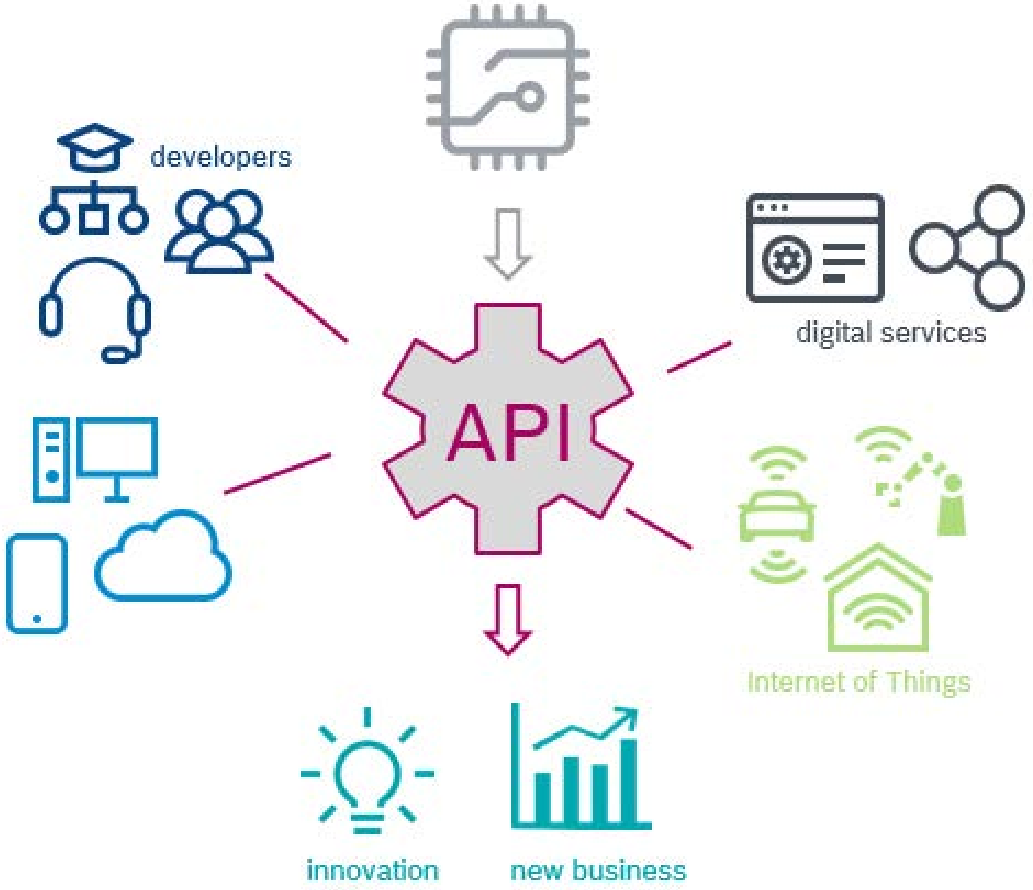
- Home appliances – Series 8 ovens

- Power tools – the Bosch Ixo

Franz-Josef Grosch | 2018-09-11

BOSCH

Bosch “Things” in a connected world
The importance of embedded software

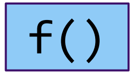


- Bosch's biggest strength in the IoT ecosystem are the Bosch “Things”
- These devices and physical products cover a multitude of domains
- Each with high market penetration typically among the TOP 3
- “Bosch is a giant in embedded software” (Dr. Volkmar Denner, CEO)

The structure of embedded software

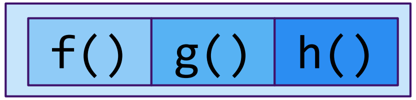
Timing behaviour expressed via the environment

- "One-step" functions ...



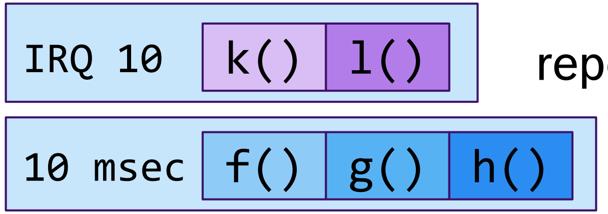
no inputs, no outputs, operates on global variables

- ... composed in operating system tasks



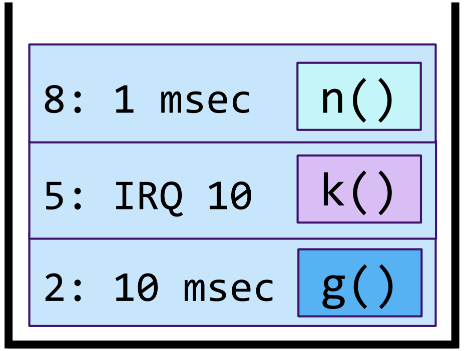
sequentially ordered

- ... activated periodicly (time-triggered), sporadicly (event-triggered) or even rate-adaptive



repeated on clock-tick or on interrupt

- ... scheduled according to priorities



high priority task pre-empts lower, task switch is a function call, only one stack for all tasks

More details: *Real world automotive benchmark for free*, Kramer et al., 2015

BOSCH

The structure of embedded software
Questions causing trouble

- One-step functions
  - How do we manage state between two activations?
  - How do we reason about the behaviour of a function over repeated activations?
- Single task composition
  - Which function is writing what variable and when?
  - What is a suitable order of functions in a task?
  - How do we reason about combinations of functions in a task?
- Execution of parallel tasks
  - How is the dataflow between tasks?
  - How do we reason about combinations of functions in parallel tasks?

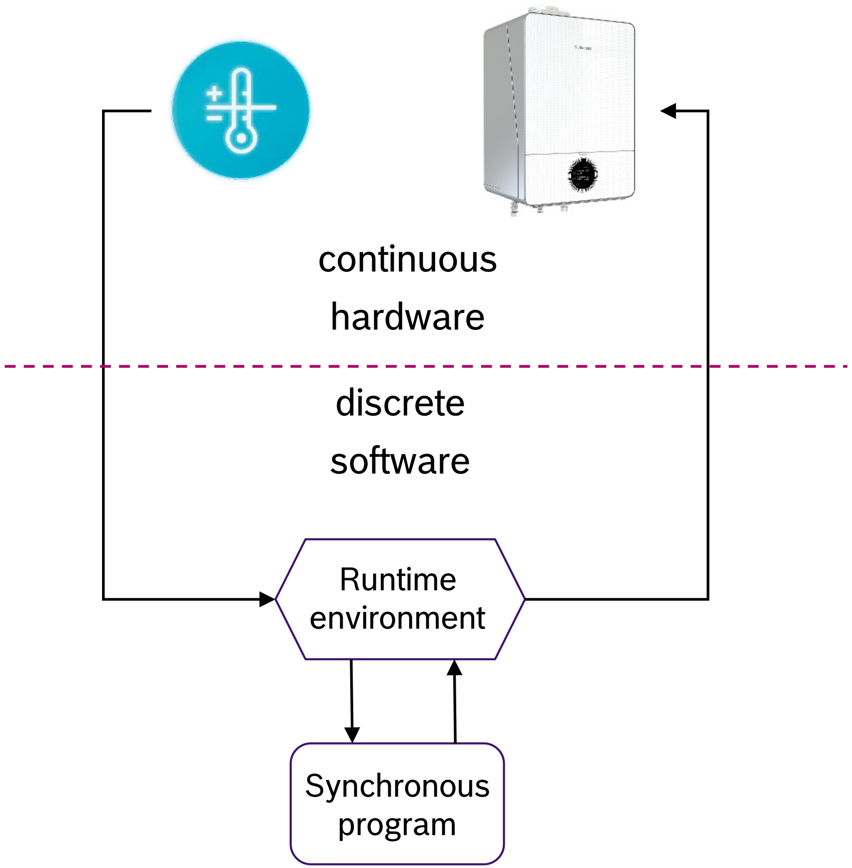
Do we need a programming language better suited to embedded requirements?

Franz-Josef Grosch | 2018-09-11

Why a new language?
Build a better tool!

Franz-Josef Grosch | 2018-09-11

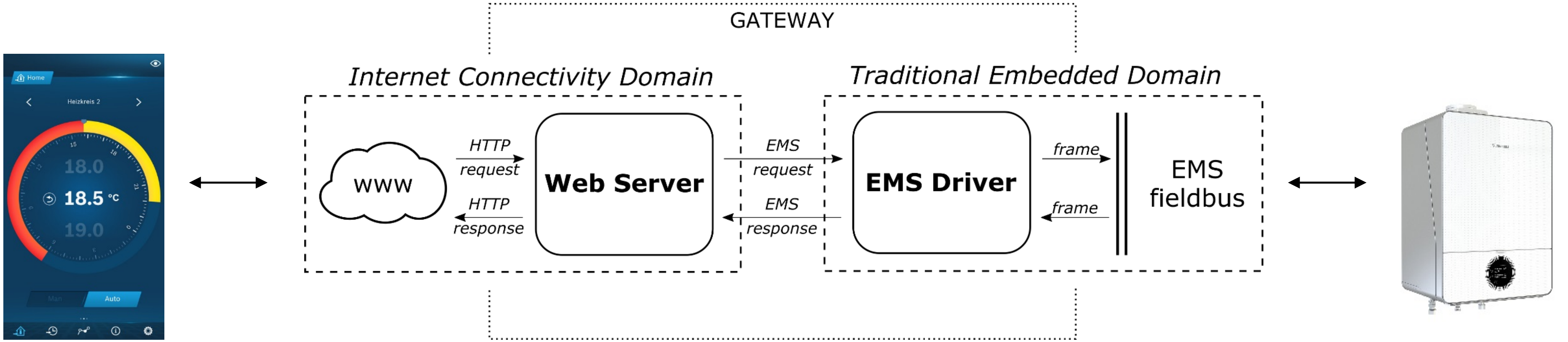
Should the language be synchronous?
The synchronous paradigm



- Environment communicates asynchronously with physical world, drives synchronous programs
- A program is executed is *steps*
  - A sequence of steps is called a thread (we prefer trail)
- Assume a step takes no time (happens instantaneously)
  - No change of input data throughout computation
- Sequences of steps can be composed concurrently
  - Accesses to shared data happen in a deterministic, causal order

Franz-Josef Grosch | 2018-09-11

Is a synchronous language "better" than C?
An experiment with Céu



www.ceu-lang.org

Function-Oriented Decomposition for Reactive Embedded Software,
Matthias Terber, SEAA 2017

Do we need a new synchronous language?
Available alternatives do not fulfill our requirements

▶ Céu	purely event-triggered, no causality, soft-realtime
▶ Esterel	no longer supported, focus on control flow and coordination
▶ Lustre	not imperative, difficult to transfer as a textual language
▶ Quartz	focus too broad: specification of hardware and software

Create a safe synchronous imperative language - Blech

Goal: Synchronous control for an imperative language
Express behaviour over time

```
function times2 (x: int32) returns int32
    return x * 2
end

activity A (inA: int32)(outA: int32)
    repeat
        await true
        outA = times2(inA)
        if outA >= 0 then
            await inA > 0
        end
        outA = times2(inA)
    end
end
```

- Start with a safe imperative core language
  - Focus on readability
  - Safe saturation arithmetic, precisely sized types
  - No global variables
- Add a statement to execute in steps
  - `await <condition/event/clock tick>`
  - `await true` ⇔ `await tick`
- Introduce two kinds of subprograms
  - `function` – one step, no await
  - `activity` – multiple steps, at least one await
- Introduce two kinds of parameter lists
  - Inputs – read only
  - Outputs – read/write

Franz-Josef Grosch | 2018-09-11

How is this executed?

Stackless execution in macro steps

```
function times2 (x: int32) returns int32
    return x * 2
end

activity A (inA: int32)(outA: int32)
    repeat
        await true
        outA = times2(inA)
        if outA >= 0 then
            await inA > 0
        end
        outA = times2(inA)
    end
end
```

time steps	inA	outA
t_i	7	14
t_{i+1}	-1	14
t_{i+2}	5	10
t_{i+3}	-2	-8

time

steps

A standard imperative core language implies

Sequentially Constructive Concurrency,

R. v. Hanxleden et al., 2013

Franz-Josef Grosch | 2018-09-11

How is this compiled?

Functions called on every step

```
// C-like pseudocode

void mainloop () {
    step_of_A();
    ...
}
```

```
void step_of_A () {
    // restore code location

    // check await condition

    // execute corresponding computation

    // save location for next reaction
}
```

Boilerplate state management code

Hard to code manually

“Business” logic

Interesting part of the program

Franz-Josef Grosch | 2018-09-11

Combine behaviours over time

Concurrent composition with improved readability and flexibility

```
activity A(inA: int32)(outA: int32)
...
end

activity B(inB: int32)(outB: int32)
...
end

activity main()
    var x: int32
    var y: int32
    cobegin weak
        run A(x)(y)
    with
        run B(y)(x)
    end
end
```

- Add a control flow statement for concurrent composition
  - Focus on readability: `cobegin ... with ... with ... end`
  - Usable as an orthogonal statement
- Entering `cobegin` blocks (also called fork)
  - Execute multi-step trails (also called threads) concurrently
- Exiting `cobegin` blocks (also called join)
  - Terminate all trails in the same step
  - Strong trails run to their end, `weak` trails can be terminated early
- Execute in causal order of statement sequences
  - Concurrent `cobegin` blocks compile to sequential code
  - Causality analysis does not look into activities and functions
- Express parallel and/or
  - `cobegin ... with ... end                // parallel and`
  - `cobegin weak ... with weak ... end  // parallel or`

Franz-Josef Grosch | 2018-09-11

Deterministic sequential execution of concurrent code

Non-global causality analysis

```
activity main()
    var x: int32
    var y: int32
    cobegin weak
        run A(x)(y)
    with
        run B(y)(x)
    end
end
```

Error: causality cycle

Solve causality cycle

```
activity main ()
var x: int32
var y: int32
    cobegin weak
        run A(x)(y)
    with
        run B(prev y)(x)
    end
end
```


Franz-Josef Grosch | 2018-09-11

Software structure and design

Structured data types, references, objects, modules

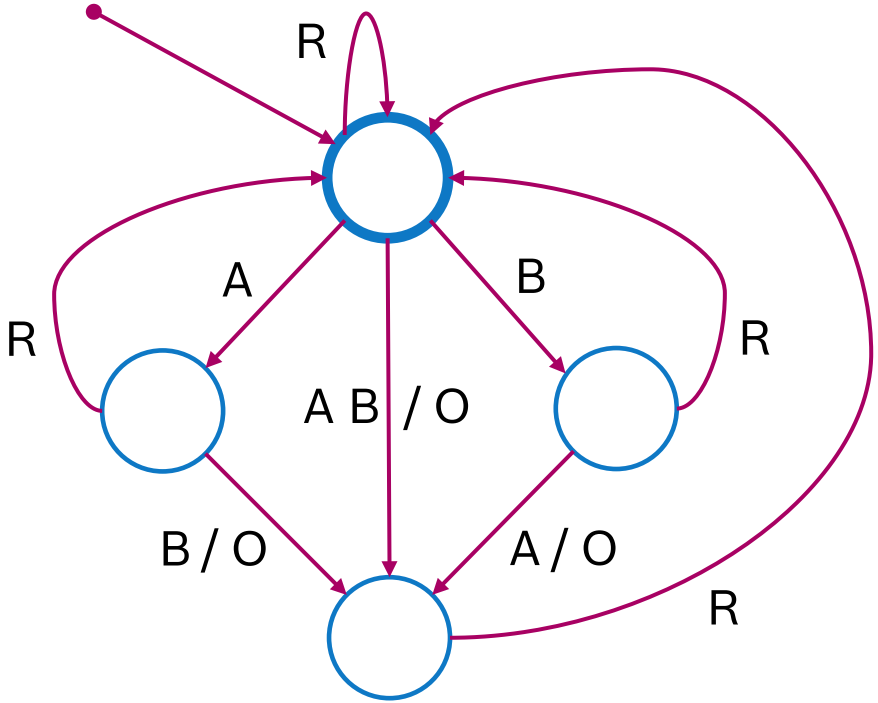
```
struct Value
    var first: int32
    var second: float32
end

ref struct MyType
    var flag: bool
    var ref value: Value // initialised at declaration
with
    const c: int32 = 42     // compile time constant
    param p: float32 = 9.81 // hex file constant
    enum Color              // scoped type declaration
        Red Green Blue
    end
    function f() returns int32       // static subprogram
    end
    mutating activity mt:actMethod() // method subprogram
        mt.value.first = f() // deref 'value' taken automatically
    end
end
...
    var v: Value = {first = 1} // second gets default value
    var mt: MyType =
        {flag = true, value = v} // ref 'v' taken automatically
```

- Introduce two kinds of types
  - value types
  - reference types
- Introduce structured value types
  - Atomic for causality analysis
  - Useful for data exchange
  - `prev` and `next` allowed, shallow copying
- Introduce reference types
  - Atomic for causality analysis
  - Useful for structuring
  - Non-cyclic dependencies required
  - Bound during instantiation
- Introduce modules
  - Unit of separate compilation
  - Non-cyclic import hierarchy required

Franz-Josef Grosch | 2018-09-11

Write things once - preemptions and hierarchy
ABRO – the synchronous “Hello world”




```
activity abro(a: bool, b: bool, r: bool)(o: bool)
    repeat
        o = false
        abort when r before // watching
            cobegin
                await a
            with
                await b
            end
            o = true
            await false      // halt
        end
    end
end
```

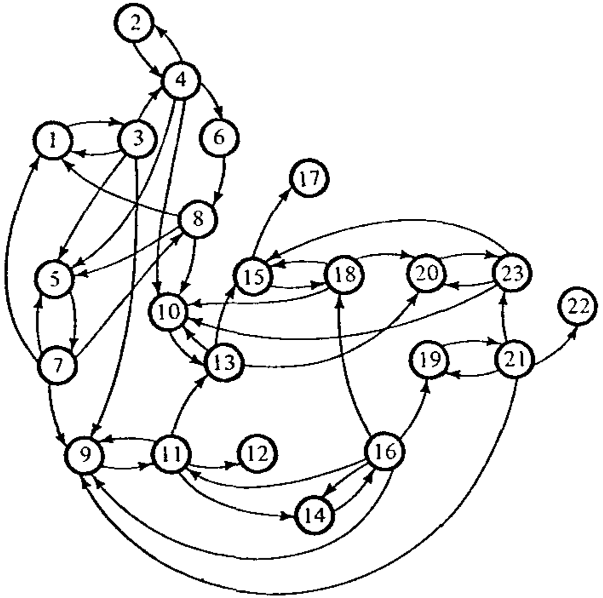
“Output O gets true as soon as both inputs A and B have been true.

The behaviour is always restarted if reset input R is true.

Franz-Josef Grosch | 2018-09-11

Modes are more important than pure state machines

True tail calls – an efficient way to implement modes



- 7 independent flags
- 128 possible combinations
- 23 permissible states

Any flowchart can be written as a program which uses only sequencing, conditionals, and procedure calls.

```
PROCEDURE A; BEGIN <processing>; CALL B END;
PROCEDURE C; IF <predicate>
                  THEN CALL D ELSE CALL E:
```

- Objections of ‘77

(1) It requires recursion to implement loops in the flowchart.

(2) Procedure calls are expensive.
They shouldn't be!

(3) The chain of procedure calls will keep pushing stack, and the stack will overflow.

(4) This style of programming is unnatural: "That's not what procedures are for!"
This is largely a matter of taste.

Steele, Jr., Guy Lewis. (1977). Debunking the “expensive procedure call” myth, or procedure call implementations can be considered harmful, or Lambda, the ultimate GOTO

Implementation of modes

Recursive tail runs - simple and effective

```
rec activity abro(a: bool, b: bool, r: bool)
                 (o: bool)
        o = false
        await a or b
        if a and b then
            return run emitO(a, b, r)(o)
        elseif a then
            return run aSeen(a, b, r)(o)
        elseif b then
            return run bSeen(a, b, r)(o)
        end
    end

and activity aSeen(a: bool, b: bool, r: bool)
                  (o: bool)
        await b or r
        if r then
            return run abro(a, b, r)(o)
        elseif b then
            return run emitO(a, b, r)(o)
        end
    end
```

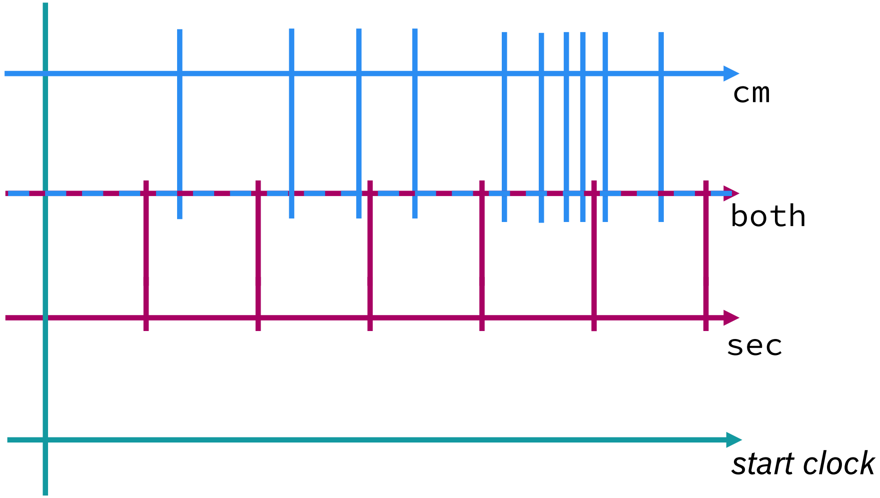
```
and activity bSeen(a: bool, b: bool, r: bool)
                  (o: bool)
        await a or r
        if r then
            return run abro(a, b, r)(o)
        elseif a then
            return run emitO(a, b, r)(o)
        end
    end

and activity emitO(a: bool, b: bool, r: bool)
                  (o: bool)
        o = true
        await r
        return run abro(a, b, r)(o)
    end
```


Franz-Josef Grosch | 2018-09-11

Clocks – a way to express multi-form time

Speed – the other synchronous “Hello world”




```
clock cm
clock sec
clock both = cm join sec
```

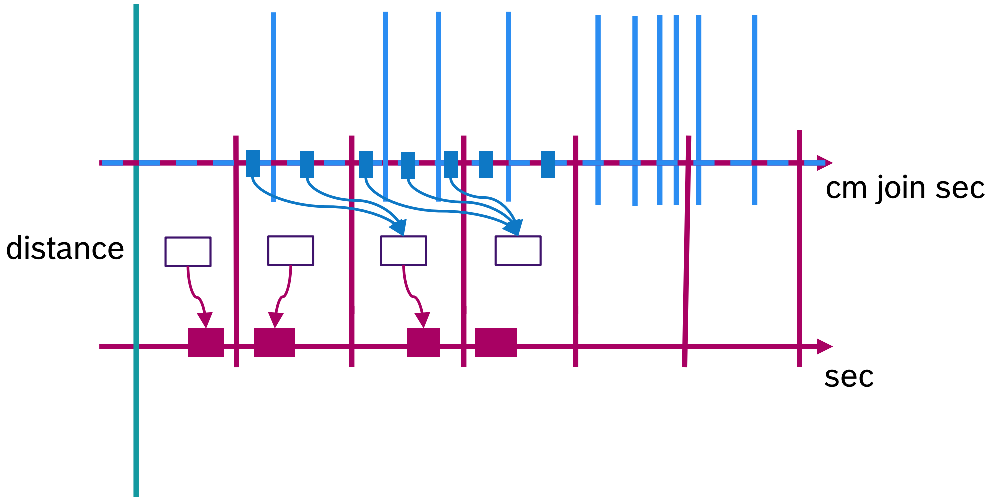
```
activity countingCmBetweenSeconds()(distance: int32) on both
    repeat await tick  // any tick
        if tick cm then
            distance = distance + 1
        elseif tick sec then
            distance = 0
        end
    end
end

activity updatingSpeed(distance:int32)(speed: int32) on sec
    repeat await tick sec
        speed = distance
    end
end

activity main() on both
    var distance: int32 = 0
    var speed: int32 = 0
    cobegin
        run countingCmBetweenSeconds()(distance)
    with
        run updatingSpeed(distance)(speed)
    end
end
```


Franz-Josef Grosch | 2018-09-11

Parallel programming with clocks
Logical execution time and clock refinement




```
clock cm
clock sec
clock both = cm join sec

activity startup()() on sec
    var speed: int32 = 0 on sec
    var distance: int32 = 0 on sec
    cobegin
        next run countingCmBetweenSeconds
                         ()( next distance) on both
    with
        run updatingSpeed
                         (distance)(speed) on sec
    end
end
```

From control models to real-time code using Giotto, Henzinger et al., 2003

Clock refinement in imperative synchronous languages, Gemünde, Brandt, Schneider, 2013

“Bosch is a giant in embedded software” (Dr. V. Denner, CEO)
Wishlist for an embedded real-time programming language

BOSCH

Core Business
“Things” driven by embedded software

- Hybrid: Time-driven and event-driven
- Predictable and deterministic
- Synchronous concurrency
- Hard real-time
- Bounded memory usage and execution time
- Easy integration of C code
- Prepared for multi-core
- Explicit control of deployment and variable placement
- Compile-time mechanisms for structuring and variants
- Safe shared memory
- Safe type system
- Expressive and productive
- A “real cool” development environment

Elevate embedded real-time programming
Bridging the gap between models and C code

Analysis & Modelling

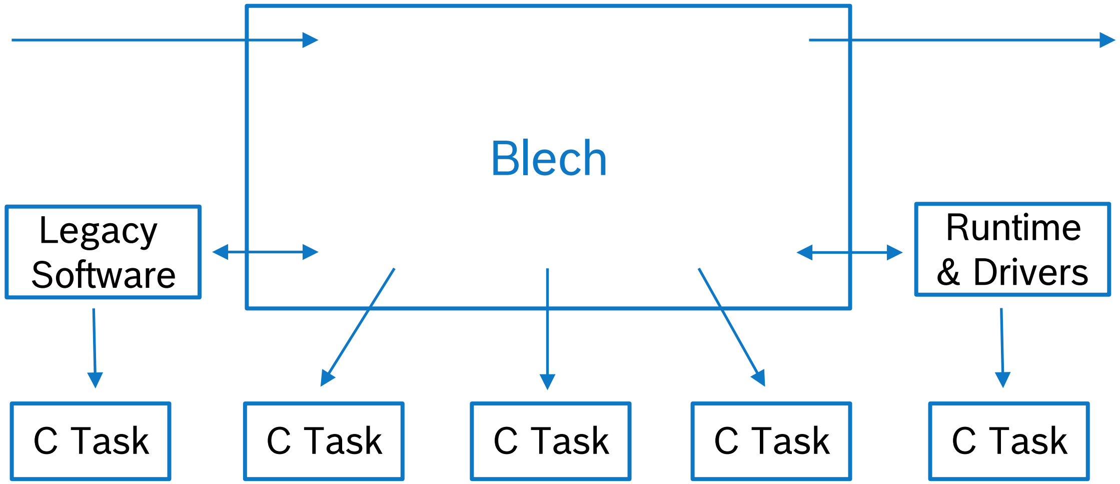
Simulink® MODELICA Scade®

ASCET-DEVELOPER ASCET-CONGRA

Simulation & Transformation

Design & Implementation

- Real-time requirements
- Reactive concerns
- Software design
- Built-in concurrency
- Deterministic parallelism



Verification & Testing

- Assertion checking
- Unit testing
- Debugging
- Closed-loop simulation

Deployment

Hardware-in-the-loop

Bosch products

Field testing

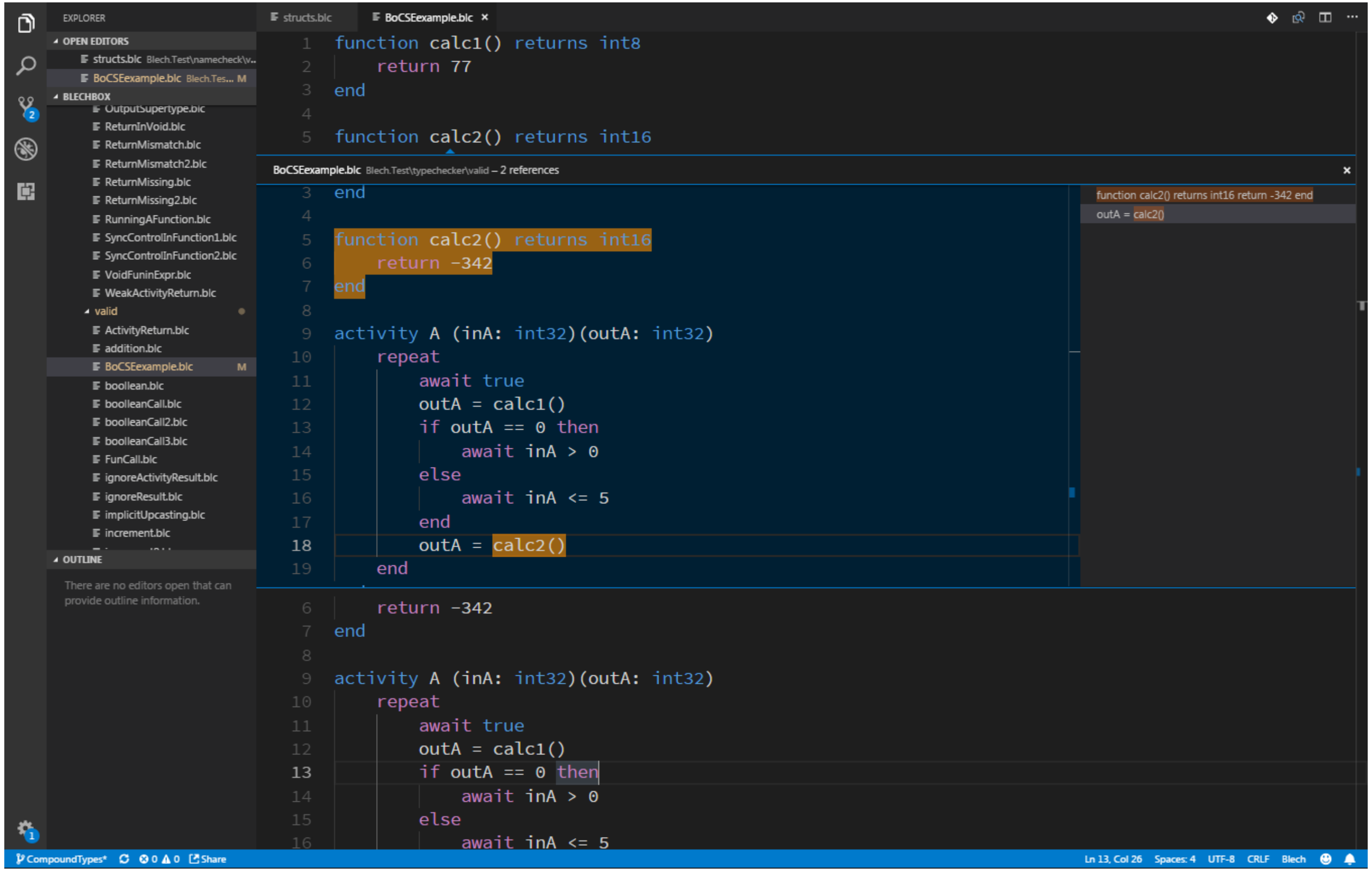
Franz-Josef Grosch | 2018-09-11

Elevate embedded real-time programming
Our embedded software vision

- Take care of multi-disciplinary engineering
- Express timing behavior in the program (not in the environment)
- Enable clean embedded software architectures
- Re-enable reasoning about parallel programs
- Improve productivity, agility, maintainability, testability, modularity, abstraction
- Support and attract software professionals

Franz-Josef Grosch | 2018-09-11

First steps on a "cool" development environment
A Blech Language Server used with Visual Studio Code



Franz-Josef Grosch | 2018-09-11

BOSCH

Where we stand
... and where to go

- We have a clear vision of Blech’s features

... we are open for discussion

- We are a small team

... we are open for cooperation

- We implement the compiler, the language server and the build system in F#

... in the mid-term we plan to go open-source

Franz-Josef Grosch | 2018-09-11

BOSCH

THANK YOU

www.bosch.com