



FOKUS

REAL-TIME TTCN-3

**Diana Alina Serbanescu, Victoria Molovata, Ina Schieferdecker,
Jürgen Grossmann**



Fraunhofer
Institute for Open
Communication Systems



Real-time Definition

FOKUS

A real-time system is one in which the correction of the computations not only depends upon the logical correctness of the computation but also upon the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred.



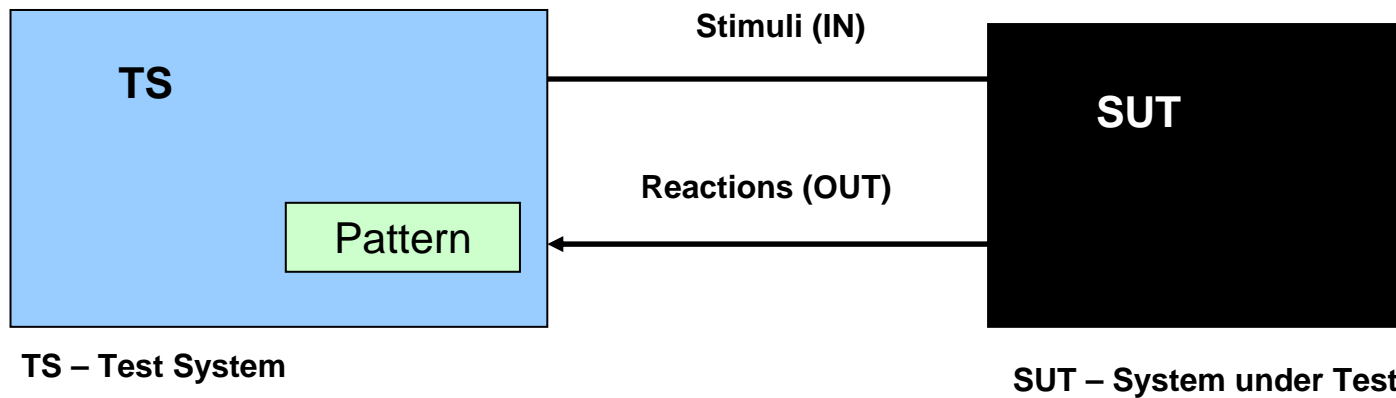
- Motivation
- What is a Real-time Test System?
- Our Approach (Real-time Test System Architecture)
- Real-time Language Requirements
- New Real-time Concepts for TTCN-3
- Practical Implementation of Concepts
 - Real-time Operating System Selection
 - Practical Example
- Further Work and Open Issues

- Need for automated testing in different domains where respecting timing constraints is crucial
 - E.g. automotive, avionics and robotic controllers

- TTCN-3
 - *Alive standard* - constantly developed and maintained by the European Telecommunications Standards Institute (ETSI)
 - *Popular* during the last years for its successful applications, especially in the telecommunication domain
 - *Modular and well-structured language*
 - *Allows multiprogramming and distribution*

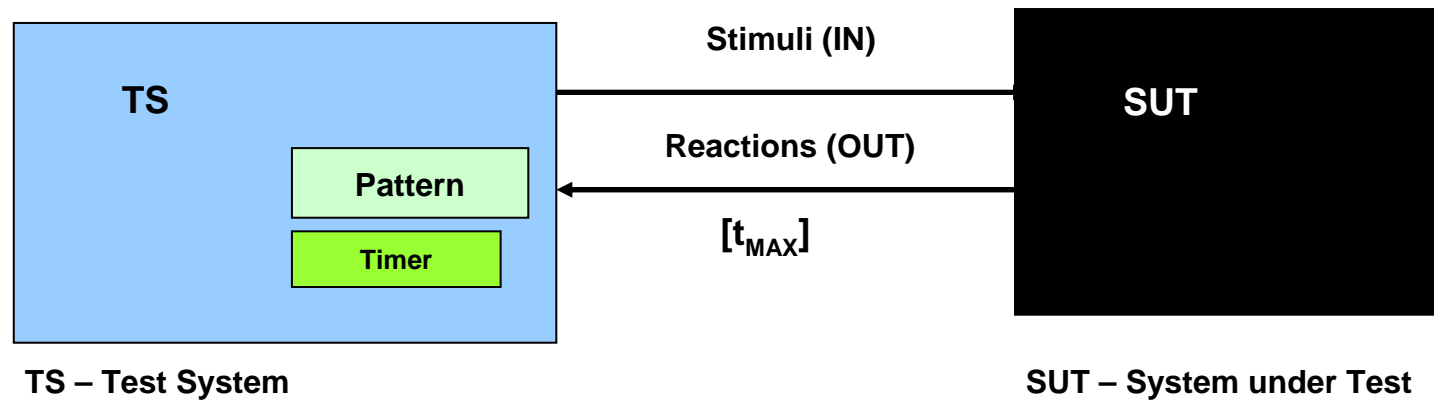
- *TTCN-3 was not designed with real-time testing in mind*

Structure of a General Test System



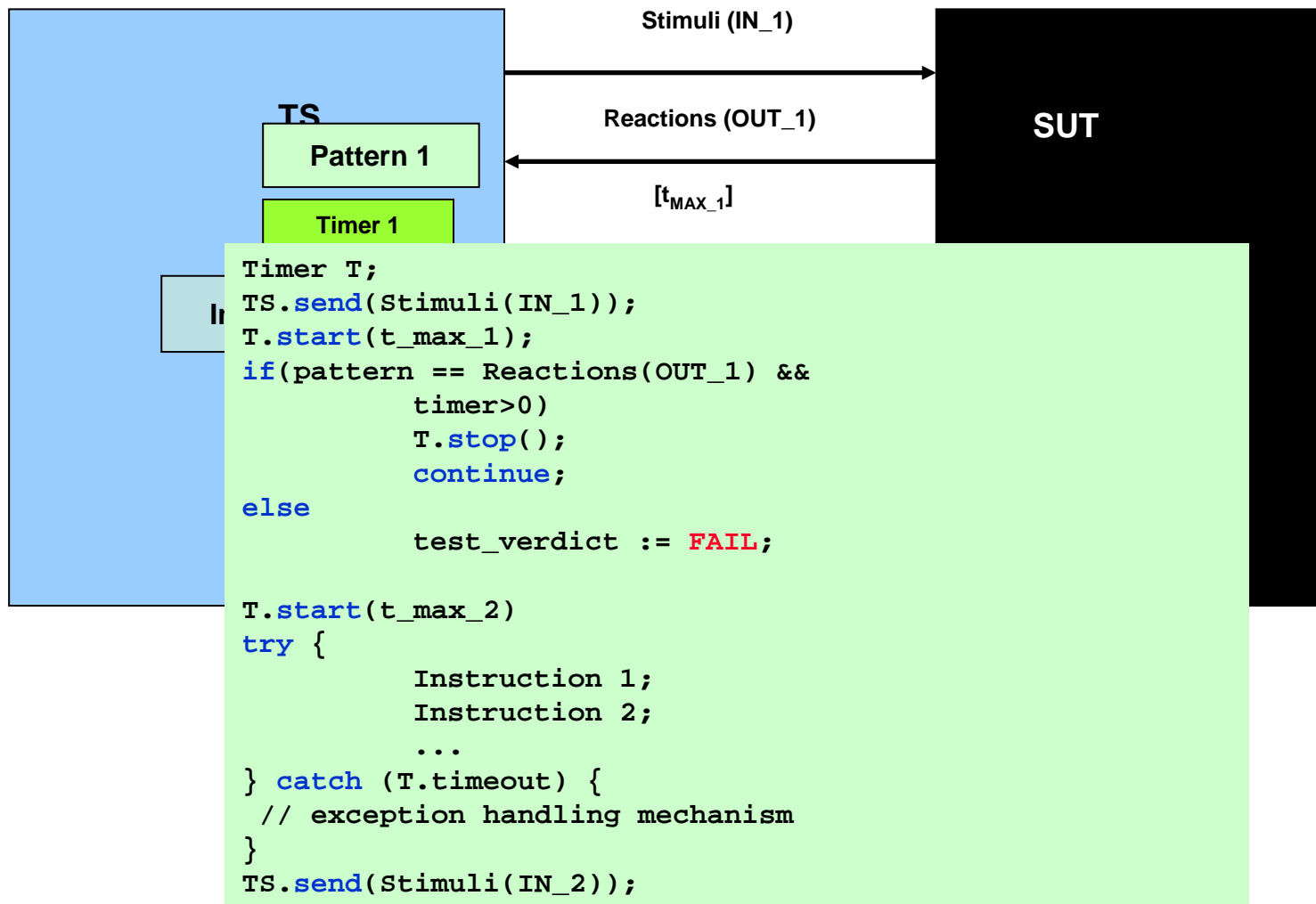
```
if (pattern == out)
    test_verdict := PASS;
else
    test_verdict := FAIL;
```

Structure of a Real-time Test System (1)

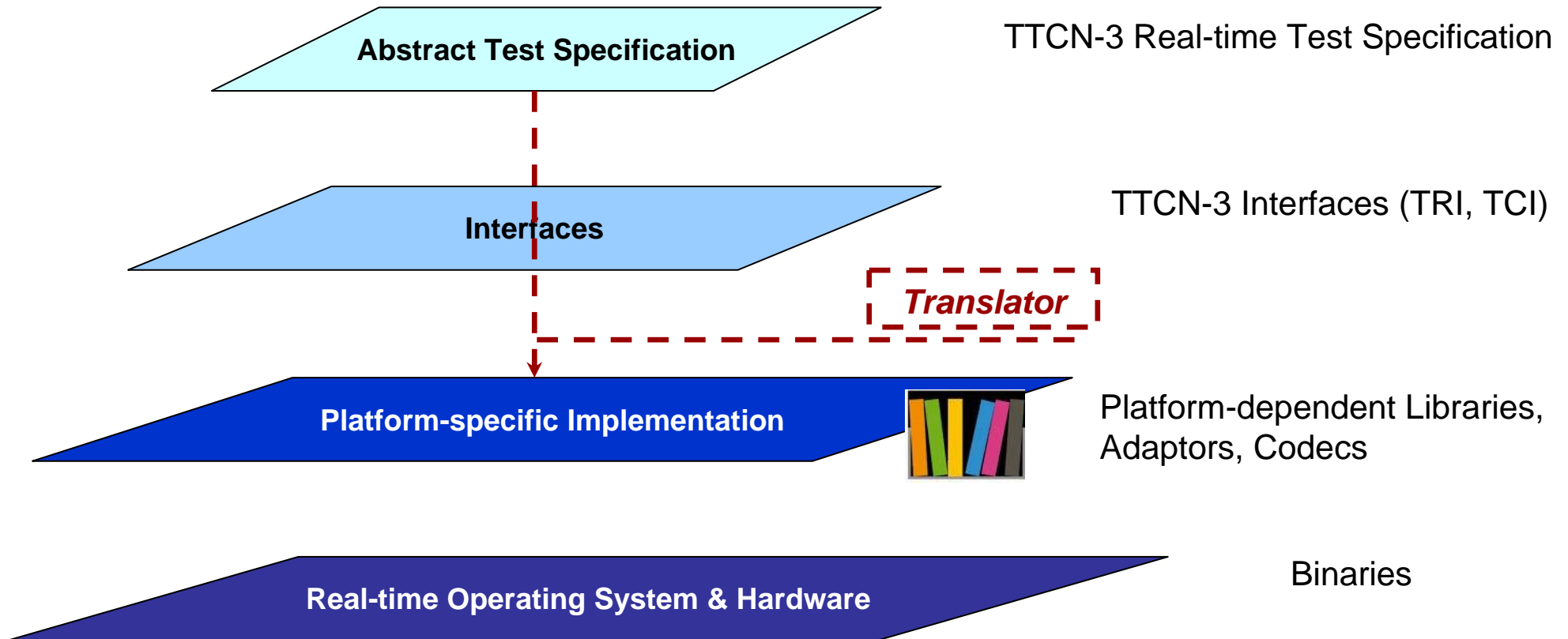


```
Timer T;  
TS.send(Stimuli(IN));  
T.start(t_max);  
if(pattern == Reactions(OUT) && timer>0)  
    test_verdict := PASS;  
else  
    test_verdict := FAIL;
```

Structure of a Real-time Test System (2)



Real-time Test System Architecture



Real Time Languages should provide means for:

- Interfacing with *time*
 - accessing clocks so that the passage of time can be measured
 - delaying processes until some future time
 - programming timeouts so that the non-occurrence of some event can be recognized and dealt with
- *Representing* timing requirements
 - specifying rates of execution
 - specifying deadlines
- *Satisfying* timing requirements
 - scheduling mechanisms
 - verification mechanism

■ Real-time Languages

- Ada, RT Java, POSIX, RT Euclid, etc.

■ Extensions for TTCN & TTCN-3

■ *PerfTTCN*

- *Concepts for performance testing: performance test scenarios, traffic models, measurement points & declarations, performance constraints & verdicts*

■ *RT-TTCN*

- *Timestamps for earliest and latest execution time, refined TTCN snapshot semantics, mapping onto timed transition systems*

■ *TimedTTCN-3*

- *New test verdict for real-time behavior, absolute time (now), delays (resume), timed synchronization for test components, timezones, online & offline evaluation of real-time properties*

■ *ContinuousTTCN-3*

- *Stream-based ports, sampling, equation systems, and additional control flow structures to be able to express continuous behavior, sampling and sampling time*

- Time (duration & point in time)
 - “Time” type and predefined time units
 - microsec, nanosec, millisec, sec, min, hour, day
 - Origin of absolute time:
 - inside testcase: testcase start,
 - inside module control: control start.
 - Conversion function for absolute points in time to internal absolute time
- Observation of point in time of observable communication operation/event (SUT-Test System interaction)
 - E.g. “receive/send -> time myTimeVar”
 - Time measured as near as possible to the SUT. (TRI extension/change required)
- Specify point in time when observable communication operation/event (SUT-Test System interaction) shall occur
 - E.g. “@t1 send” (or interval – problem: pretends precision that does not exist in practice)
 - Time as near as possible to the SUT (i.e. e.g. after encoding). (TRI extension/change required)
- If needed: now, suspend

■ Basic Concepts

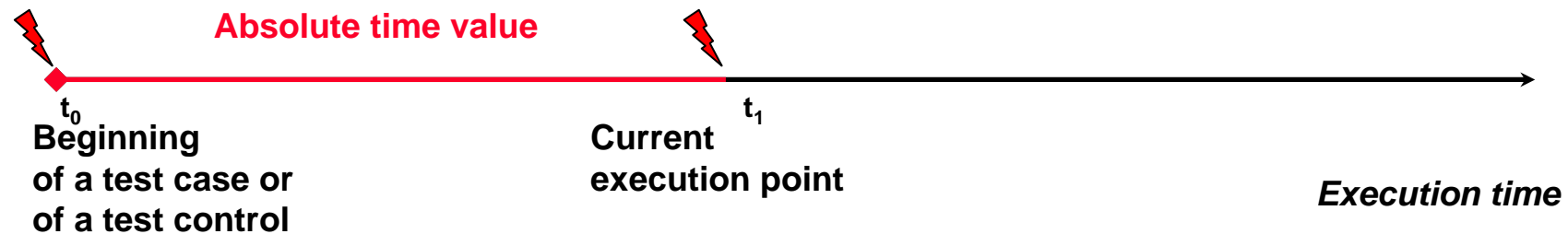
■ Handling Time

- Absolute & Relative Time Values
- Time Operators: NOW, DELAY, DELAYUNTIL

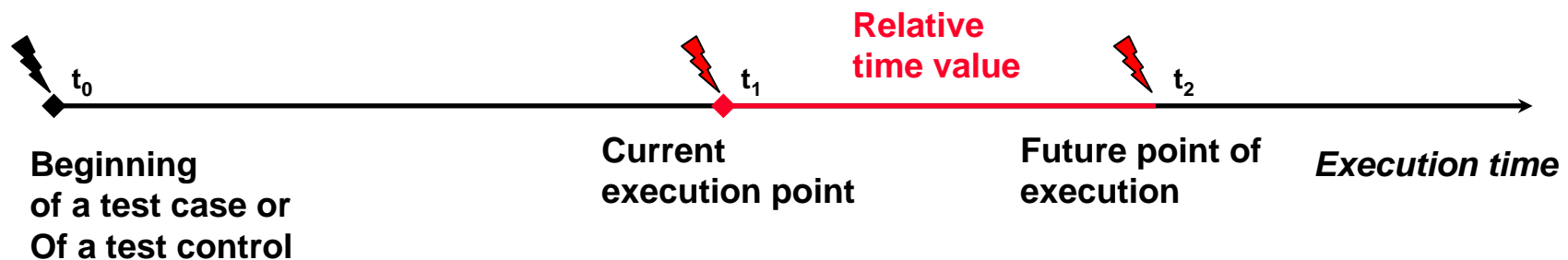
■ Guarding Statements

- Time Guarding Statement: duration
- Accessing The Receiving Time For Messages
- Critical Zones
- Priority For Components

- *Absolute & Relative Time Values (float type is used):*
 - *Absolute: the number of seconds that passed from the beginning of a test run*



- *Relative: the number of seconds that passed from a relative point of execution*



■ The NOW Operator

- Retrieve the current local time (absolute time of the current component)
- Returns a *float* value
- Used with a *self*, *mtc*, or any *ptc* handle
- Used without a handle inside a *control* block

EXAMPLE 1:

```
var float abs_time := self.now; // retrieves the absolute local time
```

EXAMPLE 2:

```
var float mtc_time := mtc.now; // retrieves the absolute time for mtc
```

EXAMPLE 3:

```
var MyPTCType ptc:= MyPTCType.create; // PTC creation
ptc.start(MyFunction()); // start PTC execution
var float ptc_time := ptc.now; // absolute local time for the
                               //associated components
```

■ The DELAY Operator

- Delays the execution of a test component for a specified period of time
- One *float* argument: represents a relative time value in seconds
- Returns “0” for success and “1” for failure
- May receive as a parameter any expression that evaluates to a *float* value
- Delay only the flow of execution for the context to which it is associated e.g. for the component associated with the behavior that contains the construct.

EXAMPLE 1:

```
var float block_time:= 10.0;  
delay(block_time); // delays the execution of the container process for 10 sec
```

EXAMPLE 2:

```
delay(10.0 + 3.0);
```

■ The DELAYUNTIL Operator

EXAMPLE 1:

```
var float dx:= 10.0;  
delayuntil(self.now + dx);
```

EXAMPLE 2:

```
var float dx:= 10.0;  
var float fix_time:= self.now + dx;  
delayuntil(fix_time);
```

EXAMPLE 3:

```
var float fix_delay:= 100.0;  
var float current_time:= self.now;  
delayuntil(fix_delay);
```

- If the value used as parameter is not calculated on the basis of the retrieved current time (see EXAMPLE 3) the following situations may occur:
 - If $\text{fix_delay} \leq \text{current_time}$, then the instruction returns with value "1"
 - If $\text{fix_delay} > \text{current_time}$, then the instruction will block the flow of execution to which it is associated for $(\text{fix_delay} - \text{current_time})$ seconds

■ The DURATION Directive

```
duration(lowerbound, upperbound)
```

```
{
  // if lowerbound2 < lowerbound1, then lowerbound2 will have no effect
  // if upperbound1 < upperbound2, then the upperbound2 will have no effect
  // if lowerbound1 > upperbound2, there is a conflict (bad programming)
  // if lowerbound2 > upperbound1, there is a conflict (bad programming)
}
duration(lowerbound1, upperbound1)
{
  C      statementblock1 ;
        duration(lowerbound2, upperbound2){statementblock2 ;}
        catch{...}
}
catch
{
  upper: { // handling behavior 1
          log("execution too slow");
          setverdict(fail);
        }
  lower: { // handling behavior 2
          log("execution too fast");
          setverdict(fail);
        }
}
}
```

Accessing The Receiving Time For Messages

- The precise time for receiving the messages may be accurately registered in the adaptor, and then passed further to the test system through the **receive** operation

- The adaptor registers the time, after the message was fully received.

```
p.receive(integer: ?) -> float timevar;
```

```
p.send(t) -> float timevar;
```

■ The CRITICAL Directive

- Indicates that the flow of execution of one component is not interrupted by the preemption mechanism or by intervention of a garbage collector
- It can be used in the control part of a module and in TTCN-3 functions, altsteps and test cases
- Delimits a block of TTCN-3 statements being composed from any TTCN-3 statement excepting: any blocking operations, function calls or altsteps
- Assumption: The thread containing the critical zone is the only one with hard real-time requirements

EXAMPLE:

```
critical
{
    delay(10.0);
    p.send(msg);
}
```

- The priority scale should be defined

```
var MyComponentType MyNewComponent;  
var integer PriorityLevel := 5;  
  
MyNewComponent := MyComponentType.create(PriorityLevel);
```

```
var MyComponentType MyNewComponent;  
var integer PriorityLevel := 5;  
  
MyNewComponent := MyComponentType.create;  
MyNewComponent.setpriority(PriorityLevel);
```

Practical Implementation of Concepts

- Real-time operating system selection
 - General criteria as: supported languages, portability, latest update, commercial status, available API and information about development and support
 - Specific criteria as: scheduling algorithms, type of RT (soft or hard), priority levels, kernel ROM size, kernel RAM size, multi-process support, interrupt latency, task switching time, type of inter process communication (IPC) mechanism, memory management, task management and so forth
- Implementation of concepts using the provided API
- Designing a concrete test system as a proof of concept
- Testing a Real-time application



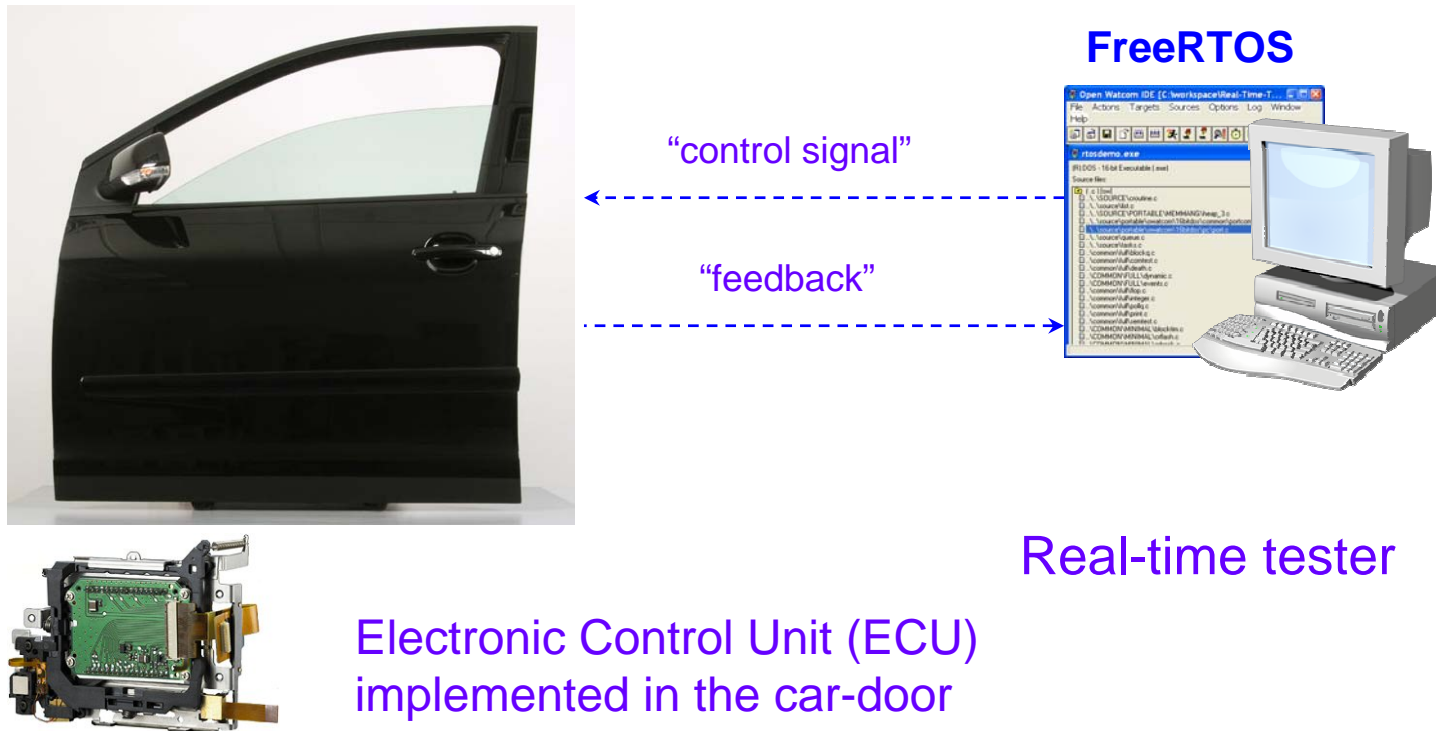
Platform Elements

Operating Systems Table

| RTOS | Language | Supported architectures | Last update | Open source/ Commercial |
|-----------------|------------------|---|--|------------------------------|
| RTLinux | C | x86, PowerPC, Alpha, MIPS | RTLinux 3.2 2003, Developed for kernel versions < 2.6 | Open source -> Commercial |
| eCos | C | ARM, CalmRISC, FR-V, Hitachi, IA-32, Motorola 68000, MIPS, PowerPC, SPARC, SuperH | On the eCos web-site only a version 2.0 of eCos (from 20 th May, 2003) can be downloaded | Open source |
| FreeRTOS | C | ARM architecture AVR, HCS12, MicroBlaze, MSP430, PIC microcontroller Renesas H8/S, x86, 8052. | April 4, 2007 | Open source |
| RTAI | For Linux Kernel | x86, x86_64, PowerPC, ARM | RTAI 3.5 27 February, 2007 | Open source |
| RTEMS | C (Ada) | m68k, ColdFire, Hitachi SH, Intel i386, Intel i960, MIPS, PowerPC, SPARC, AMD A29k, HP PA-RISC | RTEMS 4.6.2 March, 2007 | Open source |
| Prex | C | | Prex 0.4.3 13 April, 2007 | Open source |

- **Multi tasking**
- **Features**
 - Choice of RTOS scheduling policy
 - Pre-emptive, Cooperative
 - Co-routines (light weight tasks that utilize very little RAM).
 - Message queues
 - Semaphores
 - Trace visualization ability (requires more RAM)
 - Majority of source code common to all supported development tools
 - Wide range of ports and examples
- **Design Philosophy**
 - Simple, Portable, Concise

Practical Example



TTCN-3 Real-time Test Specification

```

testcase rtTest() runs on RTComponent system System {
    ...
    critical{ map(self.P, system.P); }
    // sending the string for bringing
    // the system into initial state
    P.send(SYS_INIT_CODE);

    duration(0.0, 0.03){
        P.receive(feedbackInit);
    }catch {
        upper:{
            log( "execution too slow" );
            setverdict( fail );
        }
        lower:{ }
    }

    // periodically sending the receive statement
    for(var integer i:=0; i <= nrOfTimes; i:=i+1) {
        duration(sendPeriod, sendDelay){
            P.send(BLINK_ON_CODE);
        }catch {
            upper: {
                log("execution too slow");
                setverdict(fail);
            }
            lower: {
                log("too fast rate o transmission");
                delayuntil(mtc.now + sendPeriod);
            }
        }
    }
    for(var integer i:= 0; i<=nrOfTimes; i:=i+1){
        duration(0.0, 0.03){
            P.receive(feedbackBlinkOn){ ... }
        }catch { } } // end for } // end testcase
    }
}

```

C Code Sample for Send Operation

```
void v_tSend(void *pvParameters) {
    ...
    for( ;; ) {
        /* Suspend the current task for a given time */
        if(i==0) vTaskDelay(tSendDelay);
        else vTaskDelay(tSendPeriod);
        ...
        /* Send the string to the serial port */
        time1[i] = xTaskGetTickCount();
        vSerialPutString( xPort, codes[codeId], strlen(codes[codeId]));

        /* Create the tExp task in order to wait for
        the response for this data set */
        xTaskCreate( v_tExp, tExpName, STACK_SIZE, &i,
                    mainMTC_TASK_PRIORITY, &tExpHandle[i] );

        //increment the counter i
        i++;
    }
}
```



| Sent_at (ticks) | Recv_at (ticks) | Interval (ticks) | Constraint (ticks) | Verdict |
|-----------------|-----------------|------------------|--------------------|---------|
| 2000 | 2002 | 2 | 3000 | Pass |
| 3000 | 3001 | 1 | 3000 | Pass |
| 4000 | 4001 | 1 | 3000 | Pass |
| 5000 | 5118 | 118 | 3000 | Pass |
| 6000 | 6002 | 2 | 3000 | Pass |
| 7000 | 7001 | 1 | 3000 | Pass |
| 8000 | 8031 | 31 | 3000 | Pass |
| 9000 | 9001 | 1 | 3000 | Pass |
| 10000 | 10001 | 1 | 3000 | Pass |
| 11000 | 11541 | 541 | 3000 | Pass |

■ Tick rate = 105Hz, 3000 ticks = 30 ms

- Definition of concepts:
 - Absolute and relative time as float values
 - Methods for handling time: now, delay, delayuntil
 - Retrieving time stamp at the reception of the message
 - Methods for guarding blocks of instructions: critical & duration
- Proof of concept with FreeRTOS and auto door
- Outlook
 - Efficient implementations of real-time concepts for different platforms (libraries)
 - Translator implementation
 - Scheduling mechanisms and optimization
 - The solution targets stand alone test system implementation without distribution
 - The solution is intended to be installed on an embedded system itself



Thank You for Your Attention!
Questions?