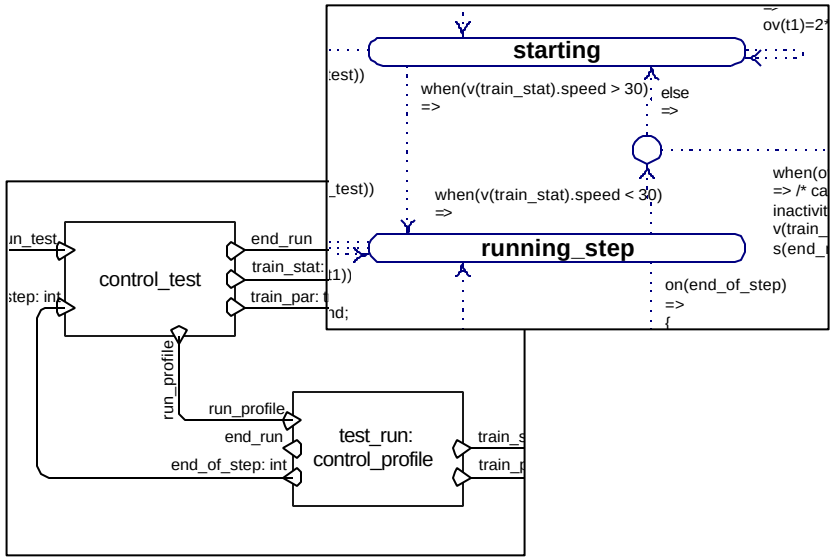
Introduction to ReaGeniX

Ari Okkonen

OBP Research Oy

What is ReaGeniX

- a graphical language for state and event-response behaviour
- a graphical language for concurrent system architecture
- an automatic code generator to implement the state behaviour and the architecture



ReaGeniX

```
    end_transition_to (running_profile)

state (running_profile)


  when(timeout(t1))  transition
     {   int new_stepnum;
        long proftime_s = hr_m_s2s(&v(train_stat)
       run_profile_get_by_time(&v(train_par).run_
       if(new_stepnum != ov(stepnum)){     send(e
         ov(stepnum) = new_stepnum;
       } } ov(t1)=Second;
  end_transition_to (transitory_12)


  when(!v(run_profile))   transition
    no_actions
```

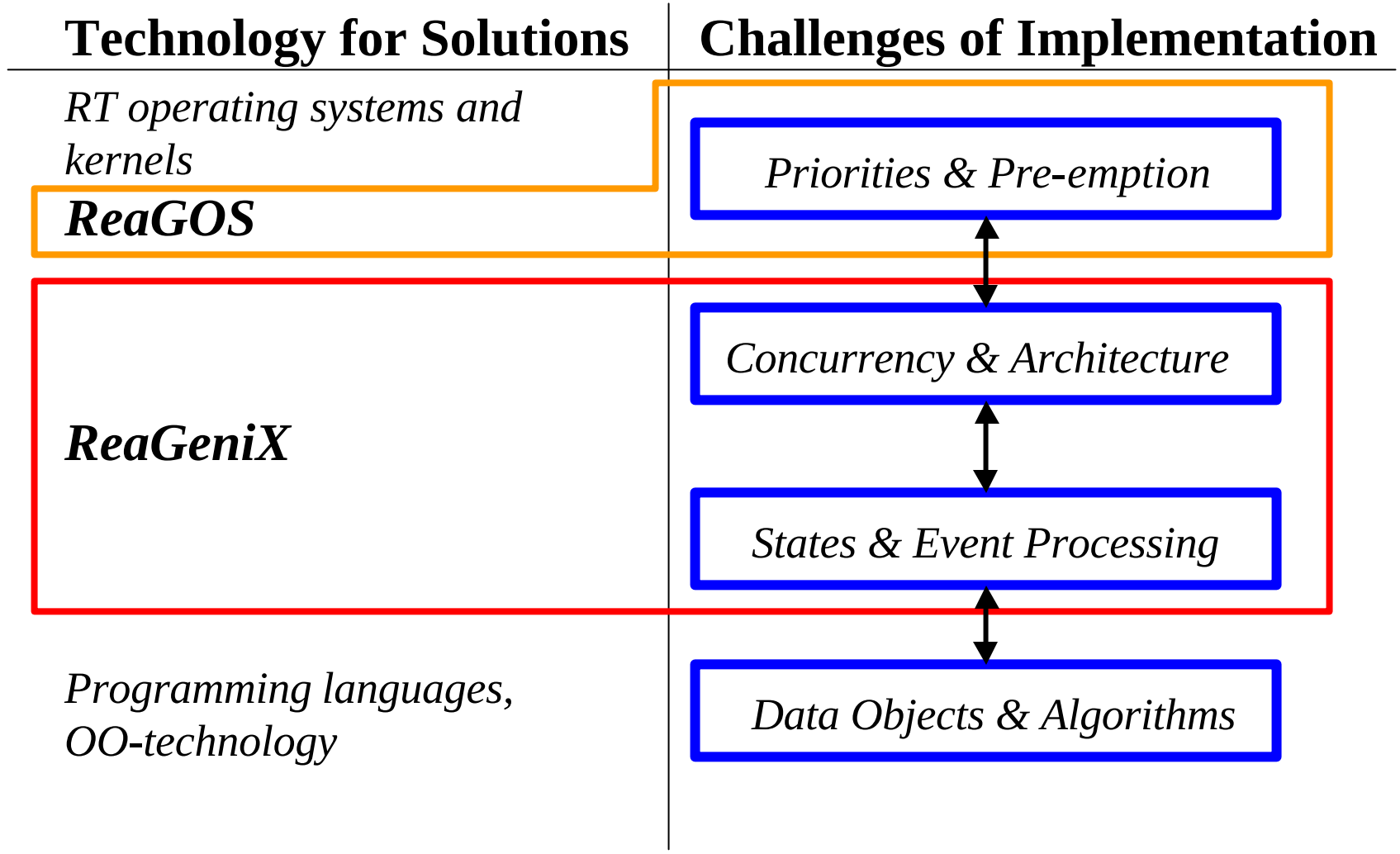
To Whom is ReaGeniX Intended

- designers of system architecture
- designers of state and event-response behaviour

What is ReaGeniX for

- design and implementation of **state behaviour**
  - waiting for a choice of external events
  - timing
- design and implementation of **system architecture**
  - concurrent operations
  - inter-component communications
- design and implementation of **reusable components**
- to **validate** the behaviour in early phases of design
- to **demonstrate** the behaviour of the main parts early

The Scope of ReaGeniX



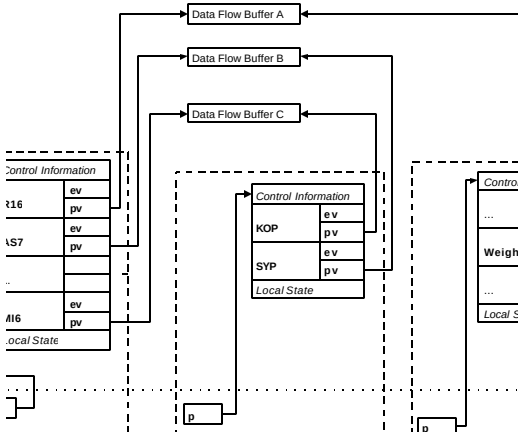
Problems NOT Suitable for ReaGeniX

- data processing algorithms
  - numeric and nonnumeric computation
  - data structure and object processing
- data base architecture and access
- interrupting a heavy computation for an urgent one

These problems are addressed using other technologies in conjunction with ReaGeniX.

$$\triangle \mathbf{A} = \mathbf{B} \cdot \triangle t$$

$$x \in \mathbf{B} \wedge f(x,y) \rightarrow g(x,y)$$



Right Tool for Right Task

- Combine several technologies to solve combined problems:
  - ReaGOS or other RT-OS to give priority to urgent operations
  - ReaGeniX to organise the concurrent architecture
  - ReaGeniX for the event-response behaviour
  - Efficient programming languages, OO-generators, or subject matter oriented generators for operations on data
- Be careful with the user interface
  - Experiment to find the technology for your specific needs

Why to Use ReaGeniX

- to get early feedback of validity of the system concept
- to freely use asynchronous concurrent components as building blocks of the system (software ICs)
- to avoid error-prone and tedious programming of state-machines, timer handling, concurrent processing, message sending / receiving
- to improve maintainability of the system
- to produce and use reusable software and design components
- to separate design of data processing operations from architectural and event processing concerns
- to get the system up and running earlier

Why to Select the ReaGeniX

- ReaGeniX is based on long experience in real-time system development in connection to theoretical work.
- Its scope is limited (concurrent architecture and state behaviour) and it is optimised to the scope.
- It produces compact and fast code.
- The produced code is especially easy to interface both to any environment and to the data processing functions.

ReaGeniX has been Used for

- Industrial automation
  - cable machinery co-ordination, Nokia Maillefer Oy
  - numerical wood milling machine control, Hirviset Oy
- Fitness equipment automation
  - Tunturi TIE 300 & TIE 200 treadmill and ergometer control
- Space satellite communications
  - SIXA particle measurement data transmission protocol
- Measurement equipment control
  - infrared interference spectrometer control (VTT)
  - two phase gas chromatograph control (VTT)

Additional Solutions

- ReaGOS Real-Time Kernel
  - allows urgent responses to **interrupt** less urgent ones
  - **generated automatically** from architecture diagrams with priority and buffer capacity annotations
  - only interrupt handlers and data processing functions are left to be written manually
  - **easy to port** to other hardware platforms
- ReaGeniX Display Interface
  - **architecture** for mapping UI data to a variety of display devices
  - definition of the mapping is **declarative**, not procedural
  - **isolates** the display description from data structures and from the event handling programs