Automatic generation of discrete handlers of real-time continuous control tasks

Soufyane Aboubekr, Gwenaël Delaval, Roger Pissard-Gibollet, Eric Rutten, Daniel Simon

INRIA Grenoble — *LIG & GIPSA-Lab*

CAR — May 2010

Real-Time Operating Systems and reactive control

- Programming control systems

  continuous control loops $\leftrightarrow$ tasks on RTOS

  performance & quality $\leftrightarrow$ periods, latencies

  $\longrightarrow$ Orccad design environment

Real-Time Operating Systems and reactive control

- Programming control systems

  continuous control loops ↔ tasks on RTOS

  performance & quality ↔ periods, latencies

  → Orccad design environment

- Discrete, reactive controllers

  events, states, control modes ↔ automata (e.g., StateFlow)

  model-based design ↔ synchronous languages

  discrete control loops ↔ discrete controller synthesis (DCS)

  → BZR programming language

Real-Time Operating Systems and reactive control

- Programming control systems

  continuous control loops ↔ tasks on RTOS

  performance & quality ↔ periods, latencies

  → Orccad design environment

- Discrete, reactive controllers

  events, states, control modes ↔ automata (e.g., StateFlow)

  model-based design ↔ synchronous languages

  discrete control loops ↔ discrete controller synthesis (DCS)

  → BZR programming language

- Contributions

  Discrete control handlers of continuous control tasks

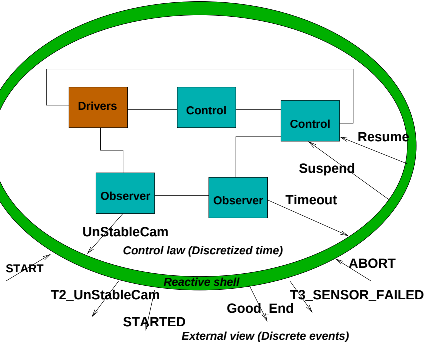
  1. integration of DCS via BZR in Orccad
  2. case study: robot arm controller

Programming control systems in Orccad (continuous)

Orccad: design, validation, implementation of robotic applications

Real-time tasks for continuous control:

- fixed-rate sampling, or multi-rate
- control/scheduling co-design : periods, latencies, gains
- Robot-Task (RT): encapsulation in a reactive shell



Programming control systems in Orccad (discrete)

Automata for task management

- Generic control of RTs, with events for: synchronizations, exceptions (3 types), pre & postconditions
- Missions design: assembling RTs (abstracted to automata) into hierarchical Robot Procedures (RPs)
- Specification and validation: Esterel synchronous language
- Real-time execution machine for the synchronous automata

Programming control systems in Orccad (discrete)

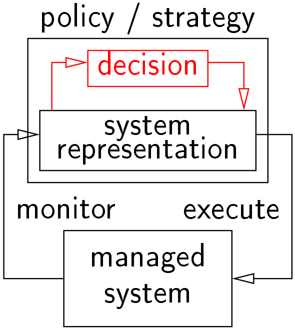
Automata for task management

- Generic control of RTs, with events for: synchronizations, exceptions (3 types), pre & postconditions
- Missions design: assembling RTs (abstracted to automata) into hierarchical Robot Procedures (RPs)
- Specification and validation: Esterel synchronous language
- Real-time execution machine for the synchronous automata

Position of the contribution in this work:
instead of programming then verifying,
use DCS to generate correct task controllers

Control Techniques for Adaptive Computing

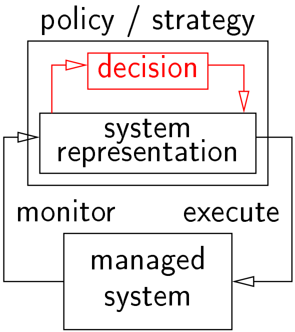
- Control of computation adaptation as a closed control loop



Discrete Control Techniques for Adaptive Computing

Use of Discrete Event Systems and supervisory control:
Petri nets, language theory (R&W), automata (synchronous)

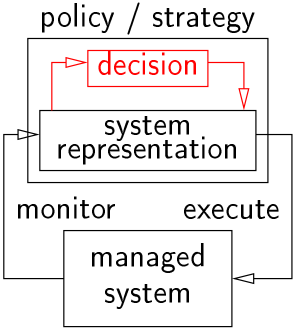
- Control of computation adaptation as a closed control loop

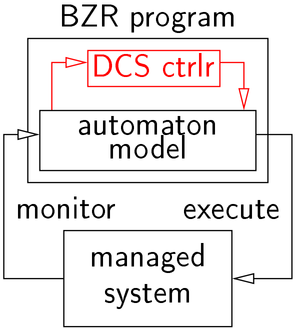


Discrete Control Techniques for Adaptive Computing

Use of Discrete Event Systems and supervisory control:
Petri nets, language theory (R&W), automata (synchronous)

- Control of computation adaptation as a closed control loop
- BZR programming language, and Discrete Controller Synthesis to compute the decision component (controller)



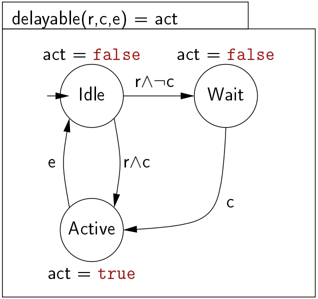


Examples of discrete computing modes

state ↔ configuration

resource access, level of consumption/quality, ...

- computation task control (example of Heptagon node)
- modes: algorithm variants for a functionality (resource, QoS)
- placement and migration: task T_i on processor/core P_j
- resource budgeting: proc./core taken for other application



- fault tolerance: migration/rollback upon processor failure
- architecture control: frequency, DVS, stand-by in MPSoC

Discrete controller synthesis: principle

Goal

Enforcing a temporal property Φ on a system (on which Φ does not a priori hold)

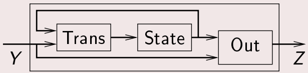
Discrete controller synthesis: principle

Goal

Enforcing a temporal property Φ on a system (on which Φ does not a priori hold)

Principle (on implicit equational representation)

State	memory
Trans	transition function
Out	output function



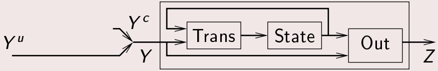
Discrete controller synthesis: principle

Goal

Enforcing a temporal property Φ on a system (on which Φ does not a priori hold)

Principle (on implicit equational representation)

State	memory
Trans	transition function
Out	output function



- Partition of inputs into controllable (Y^c) and uncontrollable (Y^u) inputs

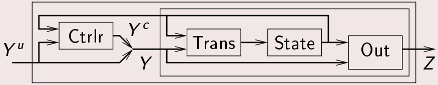
Discrete controller synthesis: principle

Goal

Enforcing a temporal property Φ on a system (on which Φ does not a priori hold)

Principle (on implicit equational representation)

State memory
Trans transition function
Out output function



- Partition of inputs into controllable (Y^c) and uncontrollable (Y^u) inputs
- Computation of a controller, *maximally permissive*, such as the controlled system satisfies Φ
- tool: `sigali` (H. Marchand, INRIA Rennes)

BZR: contracts and DCS

$f(x_1, \ldots, x_n) = (y_1, \ldots, y_p)$
$e_A \Longrightarrow e_G$
with $c_1, \ldots, c_q$

$y_1 = f_1(x_1, \ldots, x_n, c_1, \ldots, c_q)$

$\ldots$

$y_p = f_p(x_1, \ldots, x_n, c_1, \ldots, c_q)$

- built on top of heptagon synchronous nodes (M. Pouzet e.a.)

BZR: contracts and DCS

$f(x_1, \ldots, x_n) = (y_1, \ldots, y_p)$

$e_A \Longrightarrow e_G$

with $c_1, \ldots, c_q$

$y_1 = f_1(x_1, \ldots, x_n, c_1, \ldots, c_q)$

$\ldots$

$y_p = f_p(x_1, \ldots, x_n, c_1, \ldots, c_q)$

- built on top of heptagon synchronous nodes (M. Pouzet e.a.)
- contract construct :
  - assuming e_A (on the environment), enforce objective e_G
  - by constraining the additional controllable variables $c_1, \ldots, c_q$ local to the component (with)

BZR: contracts and DCS

$f(x_1, \ldots, x_n) = (y_1, \ldots, y_p)$

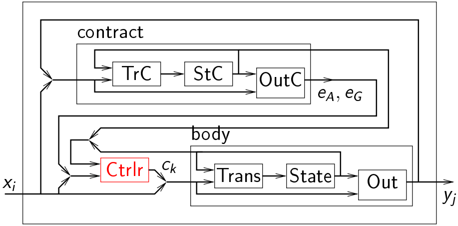
$e_A \Longrightarrow e_G$

with $c_1, \ldots, c_q$

$y_1 = f_1(x_1, \ldots, x_n, c_1, \ldots, c_q)$

$\ldots$

$y_p = f_p(x_1, \ldots, x_n, c_1, \ldots, c_q)$



- built on top of heptagon synchronous nodes (M. Pouzet e.a.)
- contract construct :
  - assuming e_A (on the environment), enforce objective e_G
  - by constraining the additional controllable variables $c_1, \ldots, c_q$ local to the component (with)
- encoded as a DCS problem (invariance) computes a local controller for each component

BZR: contracts and DCS

$f(x_1, \ldots, x_n) = (y_1, \ldots, y_p)$

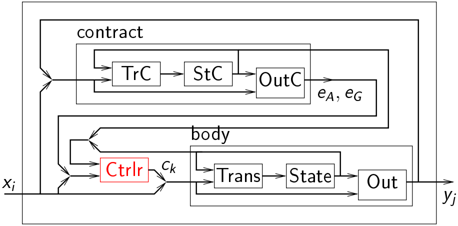
$e_A \Longrightarrow e_G$

with $c_1, \ldots, c_q$

$y_1 = f_1(x_1, \ldots, x_n, c_1, \ldots, c_q)$

$\ldots$

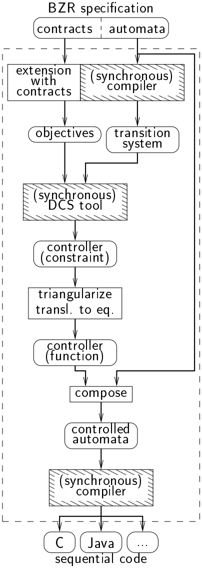
$y_p = f_p(x_1, \ldots, x_n, c_1, \ldots, c_q)$



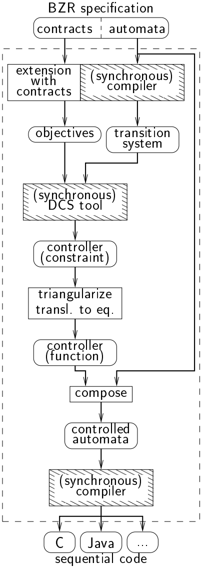
- built on top of heptagon synchronous nodes (M. Pouzet e.a.)
- contract construct :
  - assuming e_A (on the environment), enforce objective e_G
  - by constraining the additional controllable variables $c_1, \ldots, c_q$ local to the component (with)
- encoded as a DCS problem (invariance) computes a local controller for each component

[ACM LCTES'10]

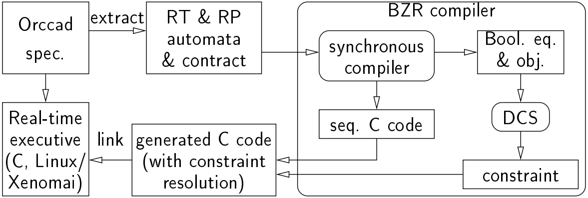
Compilation & implementation



Compilation & implementation

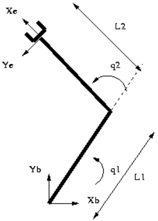


Development process:
integration in computing system
(here: Orccad):



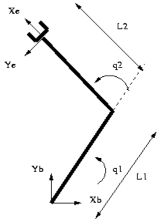
Case study: ArmX robot arm

- two links
  rotational joints (q1,q2)



Case study: ArmX robot arm

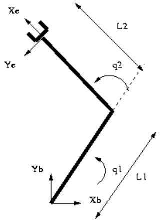
- two links
  rotational joints (q1,q2)



- robotic tool changer
  two tools: gripper, pointer

Case study: ArmX robot arm

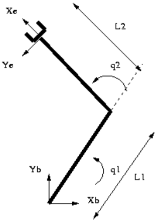
- two links
  rotational joints (q1,q2)



- robotic tool changer
  two tools: gripper, pointer
- application: when target is
  inside workspace: follow
  outside: point towards
  with appropriate tool

Case study: ArmX robot arm

- two links
  rotational joints (q1,q2)



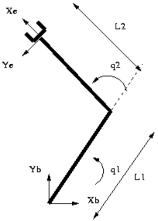
- robotic tool changer
  two tools: gripper, pointer
- application: when target is
  inside workspace: follow
  outside: point towards
  with appropriate tool

Four RTs:

- joint space move
- cartesian space move
- target aiming
  (trajectory following)
- tool change (at initial position $(q1 = 0, q2 = 0)$)

Case study: ArmX robot arm

- two links
  rotational joints (q1,q2)

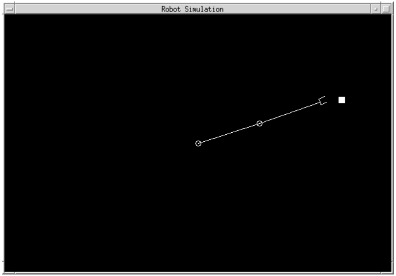


- robotic tool changer
  two tools: gripper, pointer
- application: when target is *inside* workspace: follow *outside*: point towards with appropriate tool

Four RTs:

- joint space move
- cartesian space move
- target aiming
  (trajectory following)
- tool change (at initial position $(q1 = 0, q2 = 0)$)

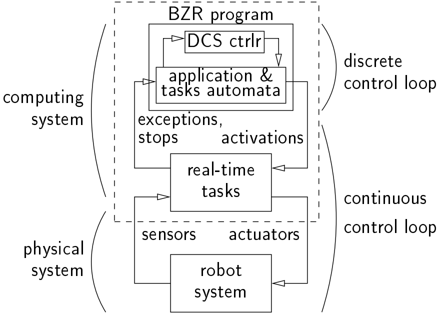
ArmX 2D simulation



Discrete control handlers of continuous control tasks

Discrete control of tasks sequencings and mode changes

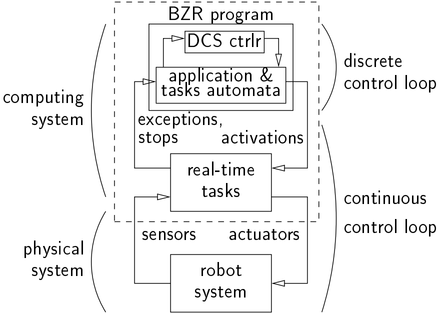
- Discrete and continuous layers



Discrete control handlers of continuous control tasks

Discrete control of tasks sequencings and mode changes

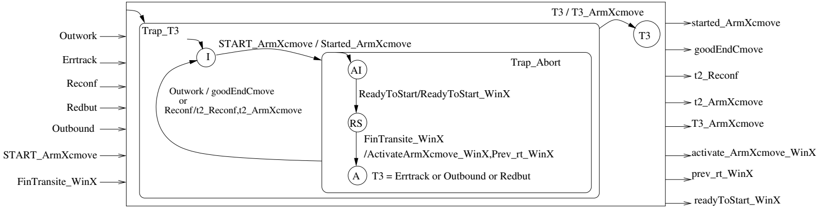
- Discrete and continuous layers



- Local task automata, coordinated by application automata with discrete supervisor, enforcing logical objective

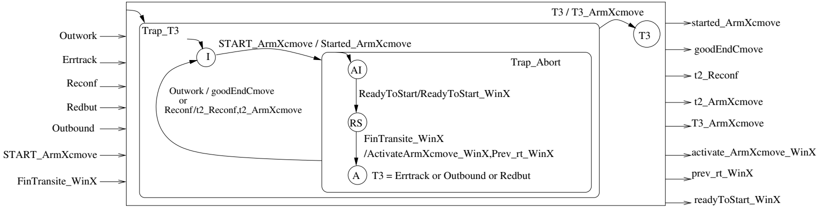
Language-level integration: Robot Tasks

BZR/Heptagon programming of the generic RT control automaton
Example of `ArmXcmove`:



Language-level integration: Robot Tasks

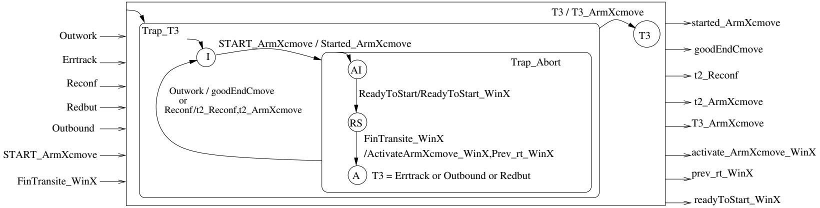
BZR/Heptagon programming of the generic RT control automaton
Example of `ArmXcmove`:



- inputs & outputs: interaction with application level, and from sensors and RTOS, to RTOS

Language-level integration: Robot Tasks

BZR/Heptagon programming of the generic RT control automaton
Example of `ArmXcmove`:



- inputs & outputs: interaction with application level, and from sensors and RTOS, to RTOS
- behaviour: phases (initialization, control) exceptions (T2, T3)

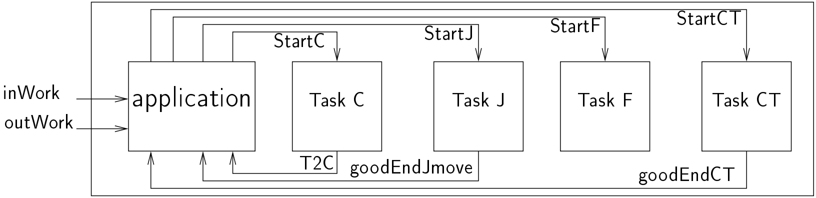
Language-level integration: Robot Procedure

Global automaton: synchronous composition
of local task automata and application

Language-level integration: Robot Procedure

Global automaton: synchronous composition of local task automata and application

Example: complete BZR program (simplified).



Global BZR node, with contract

```
node procRobot (goodEndCT,goodEndJmove,t2,outWork,inWork:bool)
               returns ( startC, startF, startJ, startCT:bool)

with (ok1,ok2,ok3:bool)
```

BZR programming methodology:

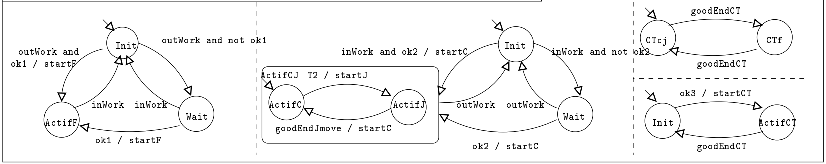
- possible behaviors
- declarative contract

Global BZR node, with contract

```
node procRobot (goodEndCT,goodEndJmove,t2,outWork,inWork:bool)
                      returns ( startC, startF, startJ, startCT:bool)



with (ok1,ok2,ok3:bool)
```



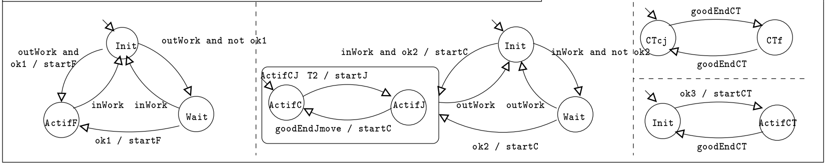
BZR programming methodology:

- possible behaviors: 4 automata in $||$: 1 observer, 3 task mgrs
  Tasks F and C/J can be delayed by control (ok_1, ok_2)
  Task CT can be triggered by control (ok_3)
- declarative contract

Global BZR node, with contract

```
node procRobot (goodEndCT,goodEndJmove,t2,outWork,inWork:bool)
                returns ( startC, startF, startJ, startCT:bool)
  goodtool = ( ActifCJ implies CTcj) & (ActifF implies CTf);

assume (not (inWork & outWork))
enforce (goodtool)
with (ok1,ok2,ok3:bool)
```

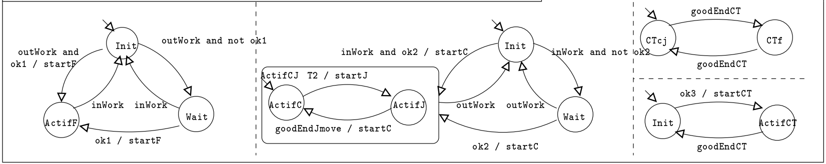


BZR programming methodology:

- possible behaviors: 4 automata in || : 1 observer, 3 task mgrs
  Tasks F and C/J can be delayed by control (ok_1, ok_2)
  Task CT can be triggered by control (ok_3)
- declarative contract: (with assumption)
  - right tool for right task: goodtool

Global BZR node, with contract

```
node procRobot (goodEndCT,goodEndJmove,t2,outWork,inWork:bool)
                    returns ( startC, startF, startJ, startCT:bool)
  goodtool = ( ActifCJ implies CTcj) & (ActifF implies CTf);
  ex = ActifF xor ActifCJ xor ActifCT;
assume (not (inWork & outWork))
enforce (goodtool & ex)
with (ok1,ok2,ok3:bool)
```

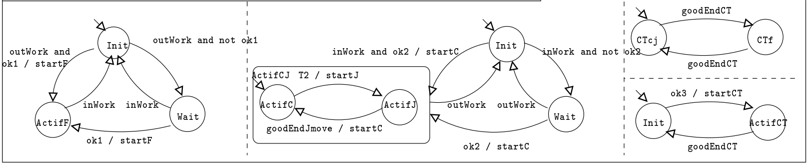


BZR programming methodology:

- possible behaviors: 4 automata in || : 1 observer, 3 task mgrs
  Tasks F and C/J can be delayed by control (ok_1, ok_2)
  Task CT can be triggered by control (ok_3)
- declarative contract: (with assumption)
  - right tool for right task: `goodtool`
  - mutual exclusion and default control: `ex`

Typical scenario

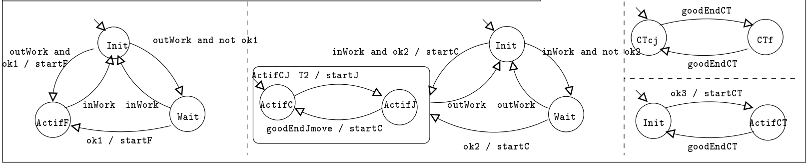
```
node procRobot (goodEndCT,goodEndJmove,t2,outWork,inWork:bool)
                    returns ( startC, startF, startJ, startCT:bool)
   goodtool = ( ActifCJ implies CTcj) & (ActifF implies CTf);
   ex = ActifF xor ActifCJ xor ActifCT;
assume (not (inWork & outWork))
enforce (goodtool & ex)
with (ok1,ok2,ok3:bool)
```



- CJmove is Active (F or CT not), tool observer is in CTcj.

Typical scenario

```
node procRobot (goodEndCT,goodEndJmove,t2,outWork,inWork:bool)
                  returns ( startC, startF, startJ, startCT:bool)
   goodtool = ( ActifCJ implies CTcj) & (ActifF implies CTf);
   ex = ActifF xor ActifCJ xor ActifCT;
assume (not (inWork & outWork))
enforce (goodtool & ex)
with (ok1,ok2,ok3:bool)
```



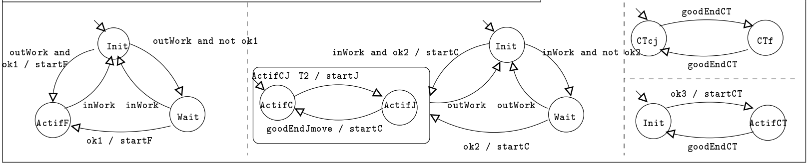
- CJmove is Active (F or CT not), tool observer is in CTcj.

- the user clicks outside of the workspace → input outWork

transition: CJ to its initial state; F quits initial, condition ok_1
contract goodtool: $\Rightarrow ok_1 = $ false: F to Wait;
contract ex: $\Rightarrow ok_3 = $ true: CT to Active

Typical scenario

```
node procRobot (goodEndCT,goodEndJmove,t2,outWork,inWork:bool)
                 returns ( startC, startF, startJ, startCT:bool)
   goodtool = ( ActifCJ implies CTcj) & (ActifF implies CTf);
   ex = ActifF xor ActifCJ xor ActifCT;
assume (not (inWork & outWork))
enforce (goodtool & ex)
with (ok1,ok2,ok3:bool)
```



- CJmove is Active (F or CT not), tool observer is in CTcj.

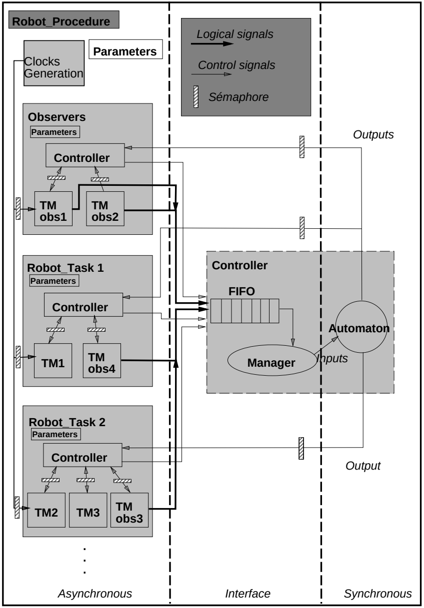
- the user clicks outside of the workspace → input outWork
  transition: CJ to its initial state; F quits initial, condition ok_1
  contract goodtool: $\Rightarrow ok_1 = \texttt{false}$: F to Wait;
  contract ex: $\Rightarrow ok_3 = \texttt{true}$: CT to Active

---

- CT ends (no inWork) → input GoodEndCT
  transition: CT to Init; tool observer to CTf
  contracts ex and goodtool: $\Rightarrow ok_1 = \texttt{true}$: F to Active

Executive-level integration

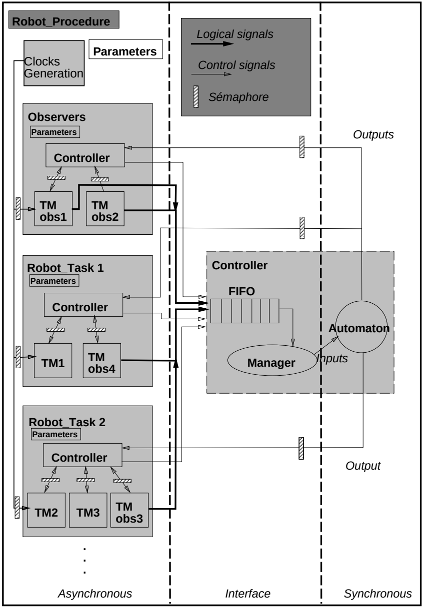
Implementation of the execution machine:



Executive-level integration

Implementation of the execution machine:

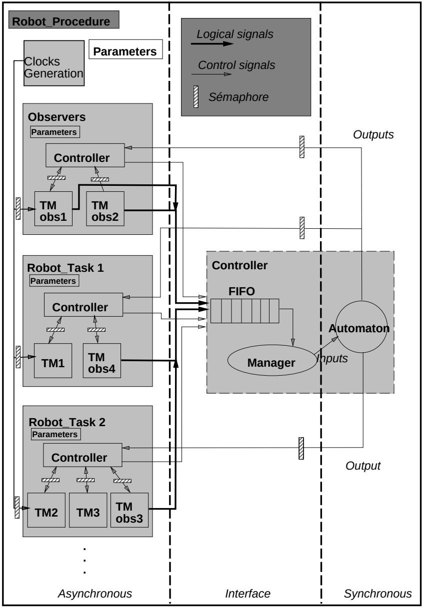
- real-time threads, triggered by clocks



Executive-level integration

Implementation of the execution machine:

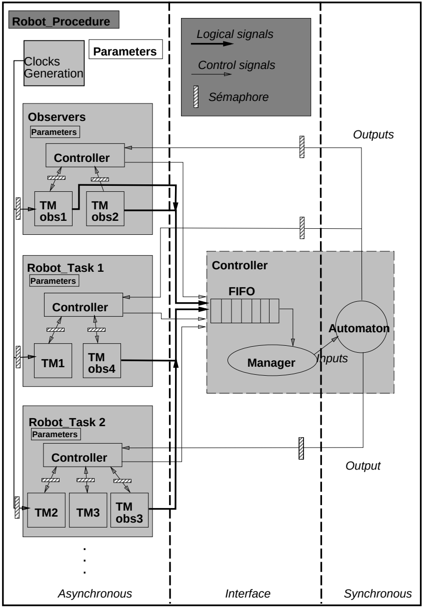
- real-time threads, triggered by clocks
- automaton:
  - highest-priority task
  - events received through FIFO
  - fast transition (μsecs)



Executive-level integration

Implementation of the execution machine:

- real-time threads, triggered by clocks
- automaton:
  - highest-priority task
  - events received through FIFO
  - fast transition (μsecs)
- Linux/Posix threads, Xenomai



Conclusion & Perspectives

- Conclusions
  - **Discrete control of real-time continuous control tasks**
    application of DCS to computing system
  - **Integration of tools**
    BZR synchronous language & Orccad design environment
  - **Case study** Robot arm, specification & simulation

Conclusion & Perspectives

- Conclusions
  - **Discrete control of real-time continuous control tasks**
    application of DCS to computing system
  - **Integration of tools**
    BZR synchronous language & Orccad design environment
  - **Case study** Robot arm, specification & simulation
- Perspectives
  - **more integration** designing controllable runtime executives
  - **more elaborate models**
    finer grain, e.g. fault tolerance [FMSD09]
  - **more DCS** costs on paths, reachability, dynamical controllers
  - **more applications** e.g. GreenIT (sustainable IT)
    Green4IT: energy/power consumption models
    for sensor networks, servers and parallel computing
    IT4Green: applying control programming techniques
    to program e.g., "intelligent" buildings