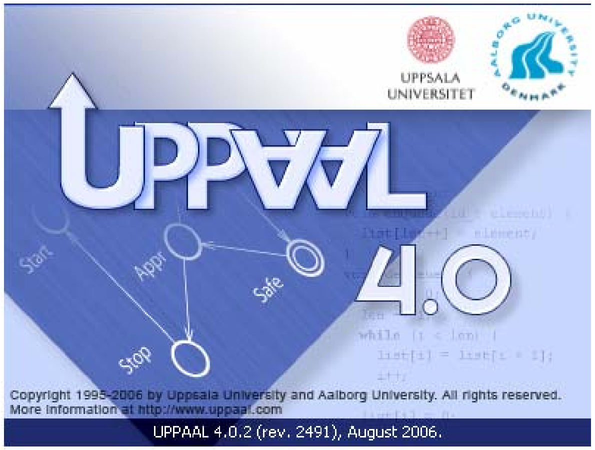


Validation of Real Time and Embedded Systems

using UPPAAL

Kim Guldstrand Larsen

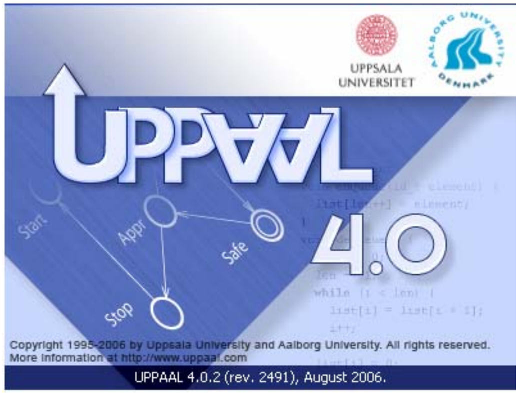
Aalborg University Denmark

BRICS Basic Research in Computer Science

CISS Center for Indlejrede Software Systemer

UPPAAL Branches

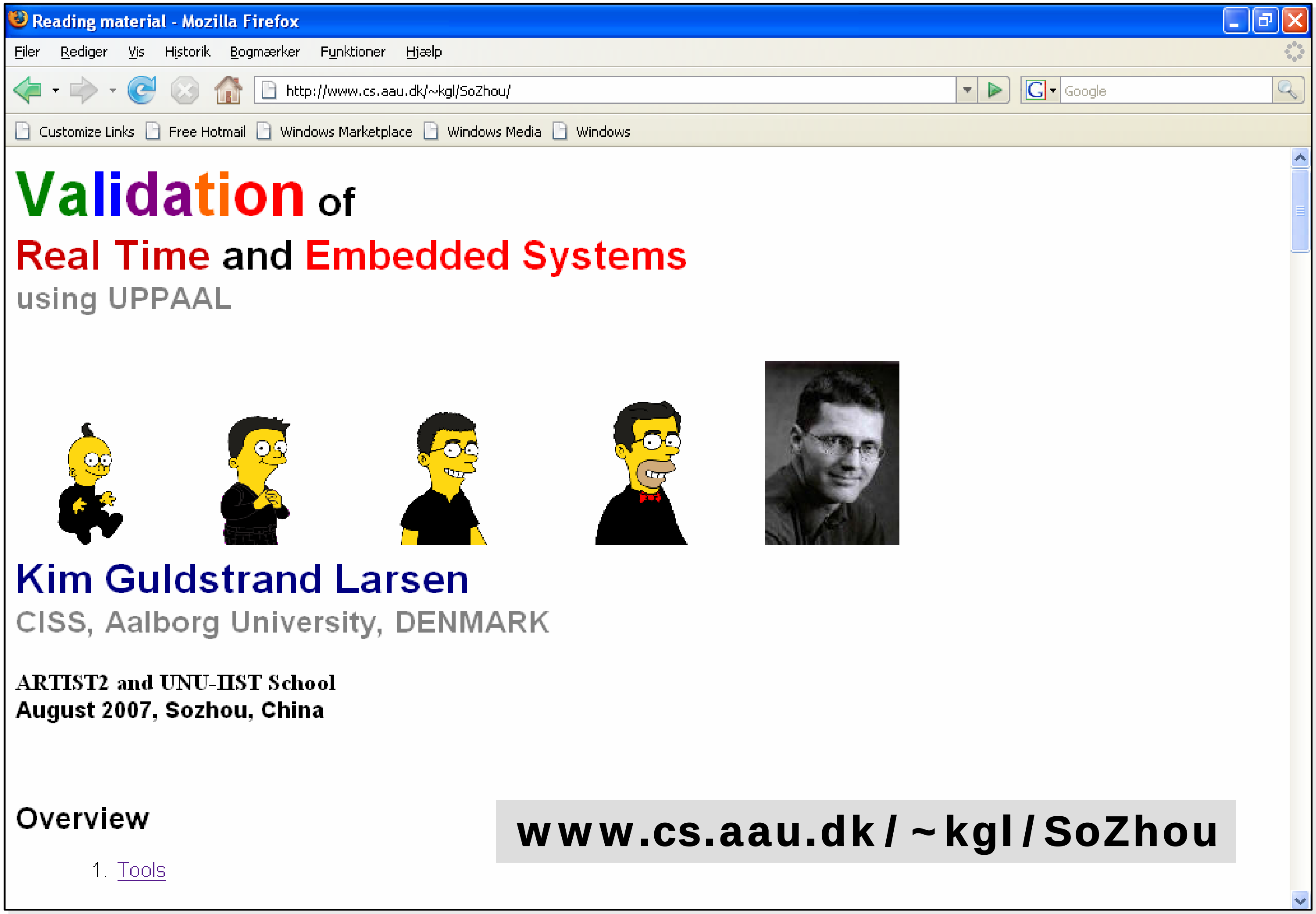
- **Real Time** **CLASSIC**
  Verification
- **Real Time** **CORA**
  Scheduling &
  Performance Evaluation
- **Real Time** **TIGA**
  Controller Synthesis
- **Real Time** **TRON**
  Testing



CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

Reading Material

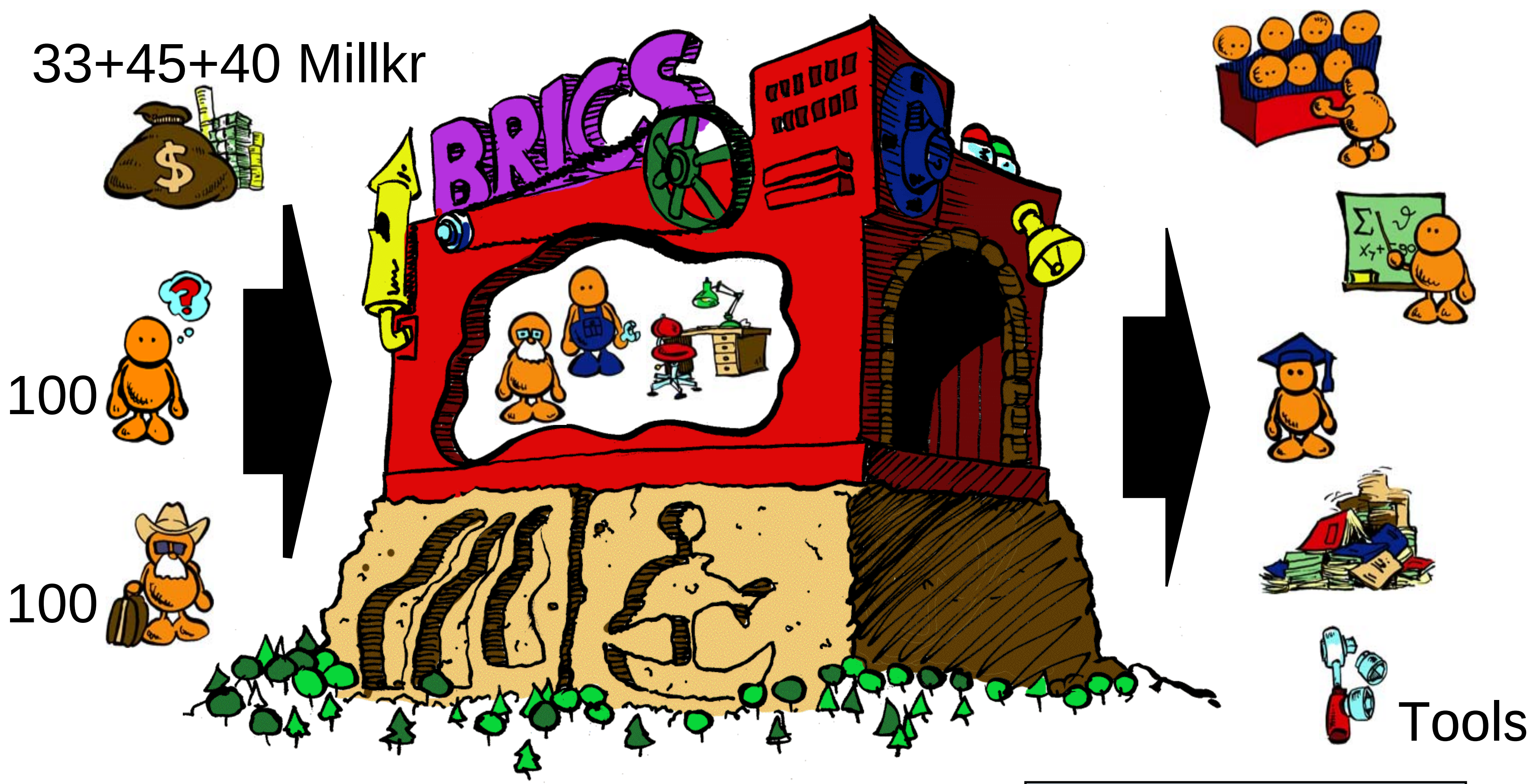


BRICS Basic Research in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS Machine

Basic Research in Computer Science, 1993-2006



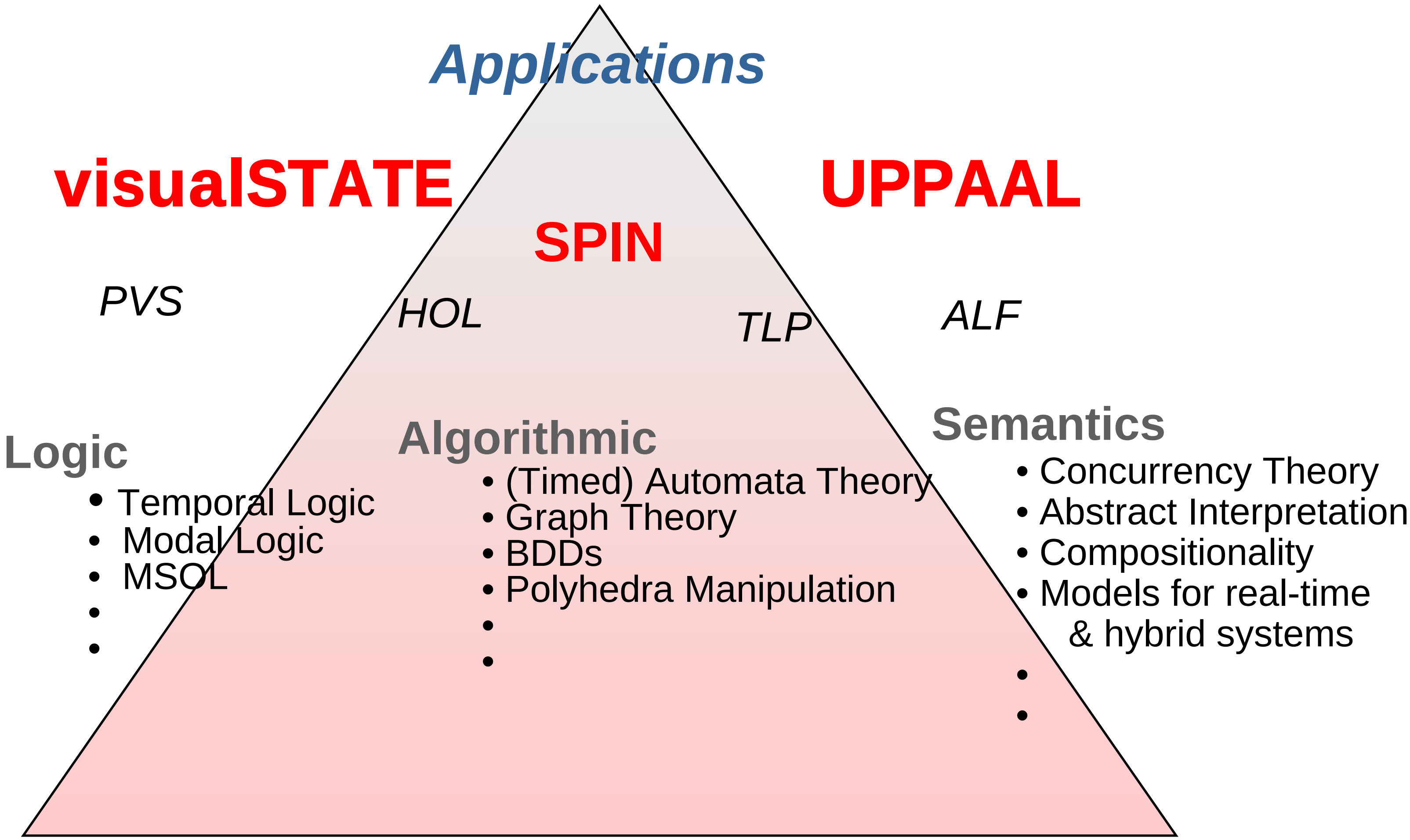
Aalborg

Aarhus

Other revelant projects
ARTIST, AMETIST

BRICS
Basic Research
in Computer Science

Tools and BRICS



CISS: *Center for Embedded Software Systems*

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Kim Guldstrand Larsen
kgl@cs.auc.dk
96358893

or

CISS
www.ciss.dk
info@ciss.dk
96357220

Aalborg Universitet
Fr. Bajersvej 7B
9220 Aalborg Ø

BRICS
Basic Research
in Computer Science

Why CISS ?

- 80% of all software is embedded
- Demands for
  increased functionality
  with
  minimal resources
- Requires multitude of skills
  - Software construction
  - Hardware platforms,
  - Communication
  - Automation
  - Testing & Verification

- **Goal**:
  Give a qualitative lift to current industrial practice
  !!!!!

BRICS
Basic Research
in Computer Science

CISS in Numbers

- National Comptetence Center 2002:

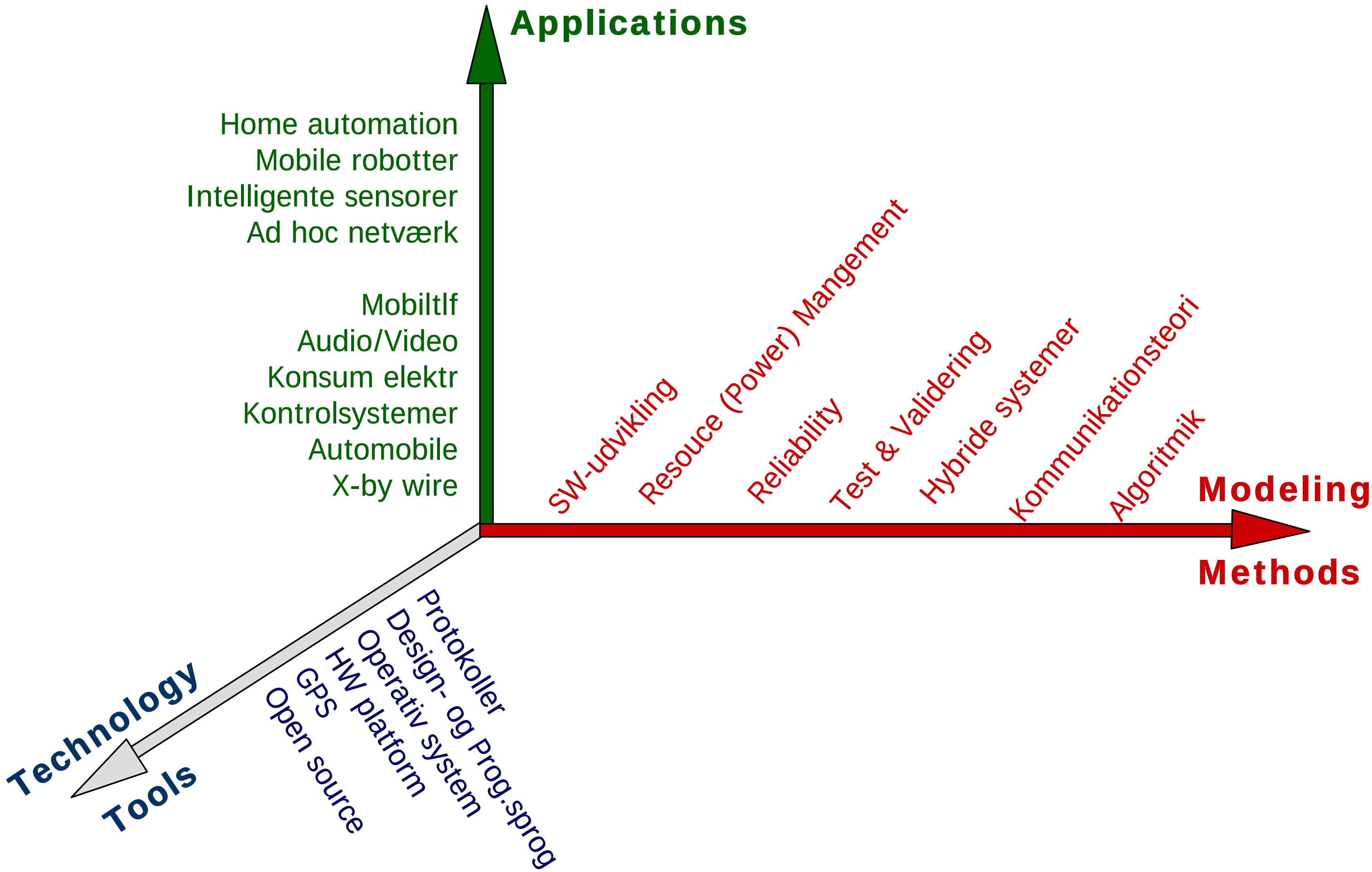
  25,5 mil. kr Ministry
  6 mil. kr North Jutland
  6 mil. kr Aalborg City
  12,75 mil. kr Companies
  12,75 mil. kr AAU

- **40** projects
- **20** CISS employees
- **25** CISS associated researcher at 3 different research groups at AAU

- 19 industrial PhDs

BRICS
Basic Research
in Computer Science

Focus Areas

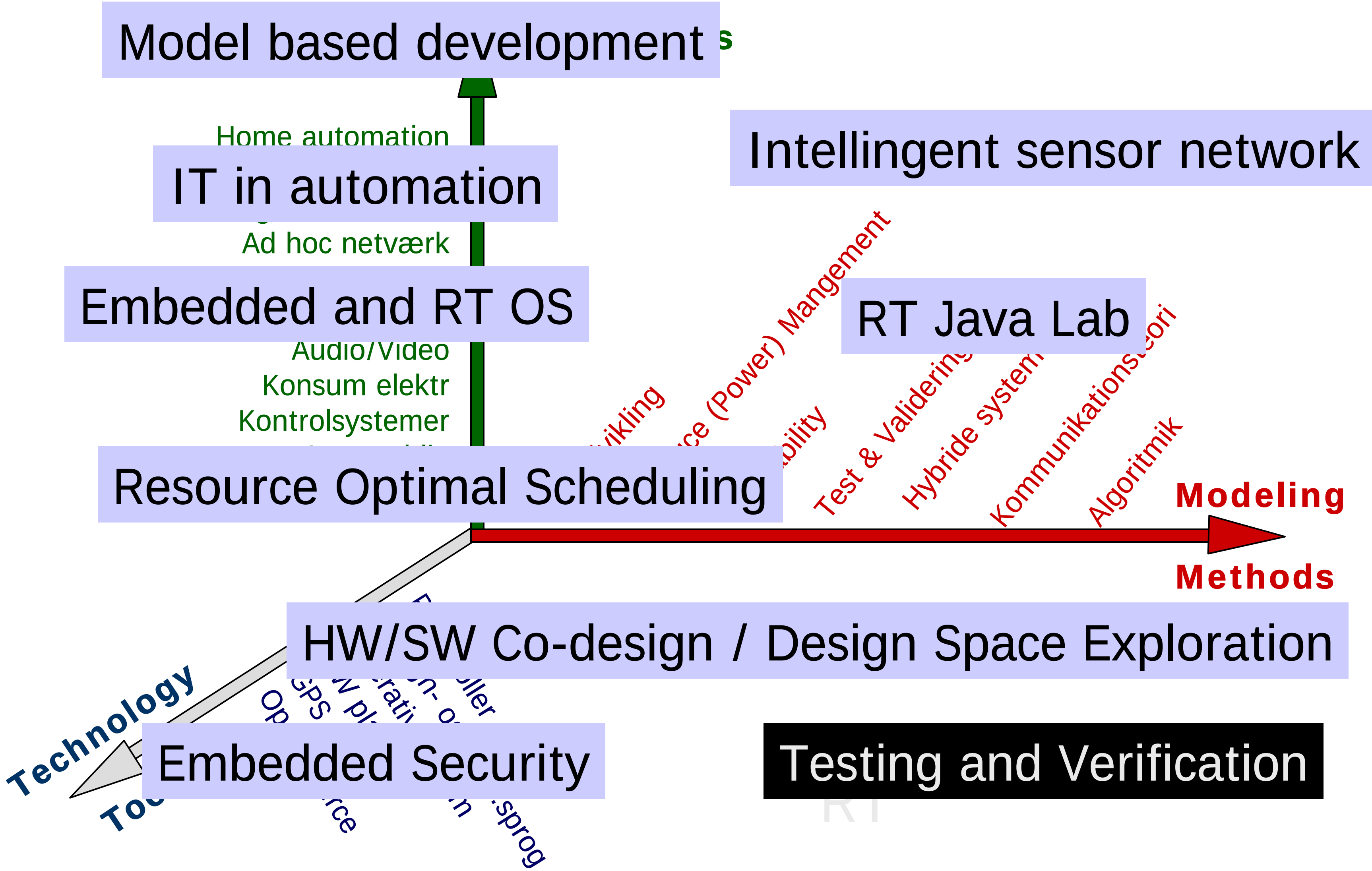


CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Focus Areas



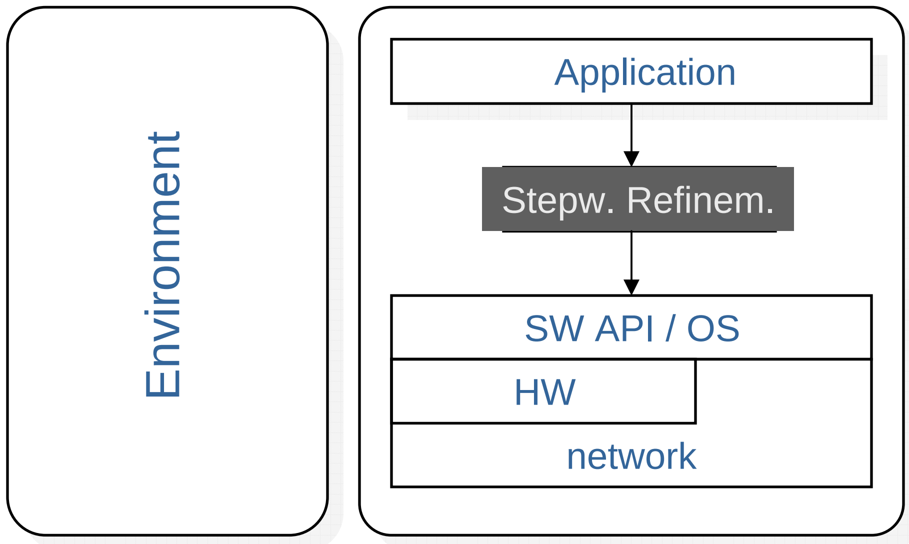
BRICS
Basic Research
in Computer Science

DaNES

Danish Network for Intelligent Embedded Systems

CISS

CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

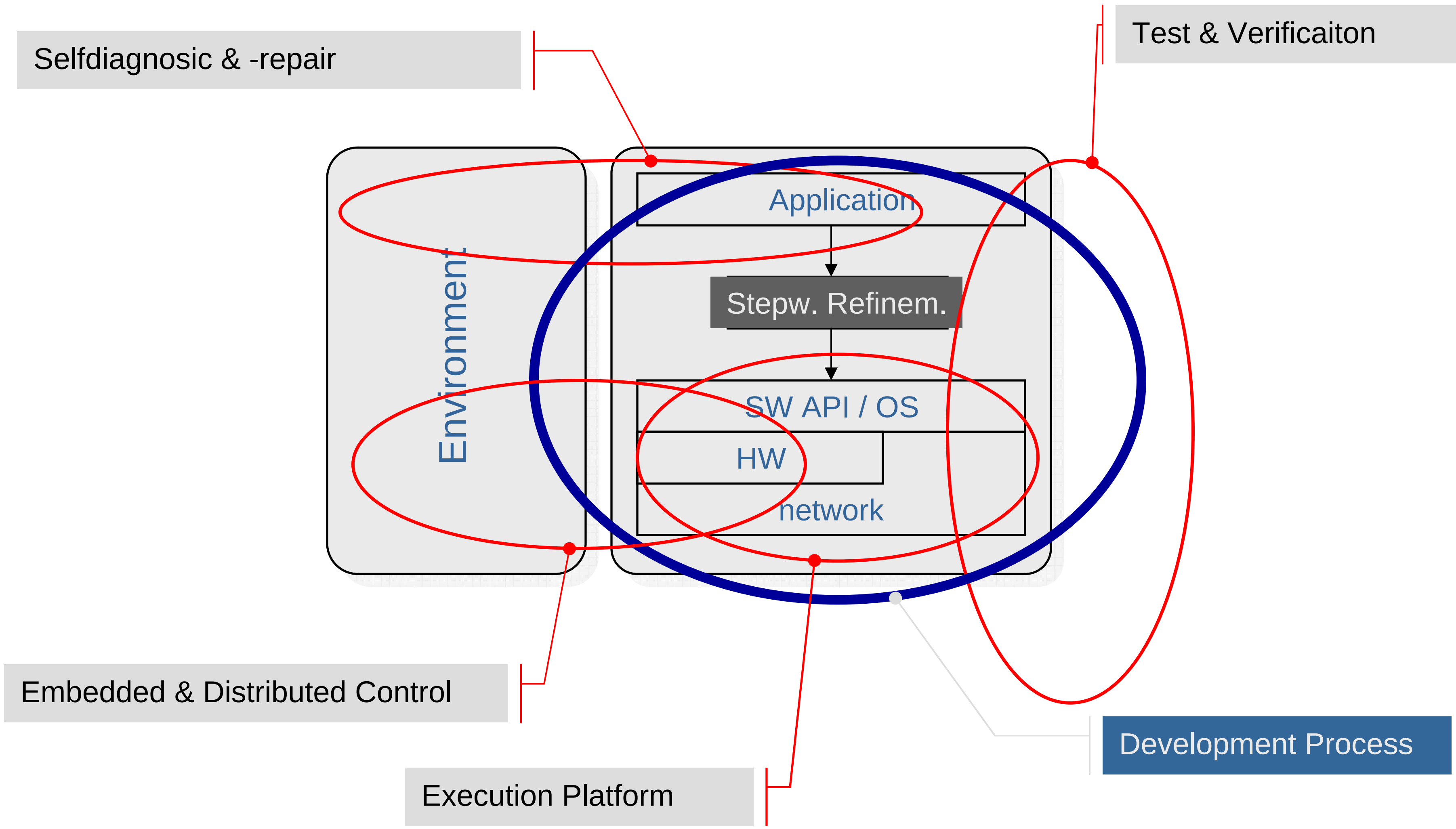


Funded by
Danish Advanced Technology Foundation

Budget 9 MEuro / 4 years

BRICS
Basic Research
in Computer Science

Challenges



Selfdiagnosic & -repair
Test & Verificaiton
Environment
Application
Stepw. Refinem.
SW API / OS
HW
network
Embedded & Distributed Control
Development Process
Execution Platform

DaNES

Danish Network for Intelligent Embedded Systems

Challenges

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Selfdiagnosic & -repair

SKOV

TERMA

novo nordisk

Test & Verificaiton

Model Driven and Component Based Development

mci

GATEHOUSE

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

ice power

Embedded & Distributed

Informatik og Matematisk Modellering

Platform

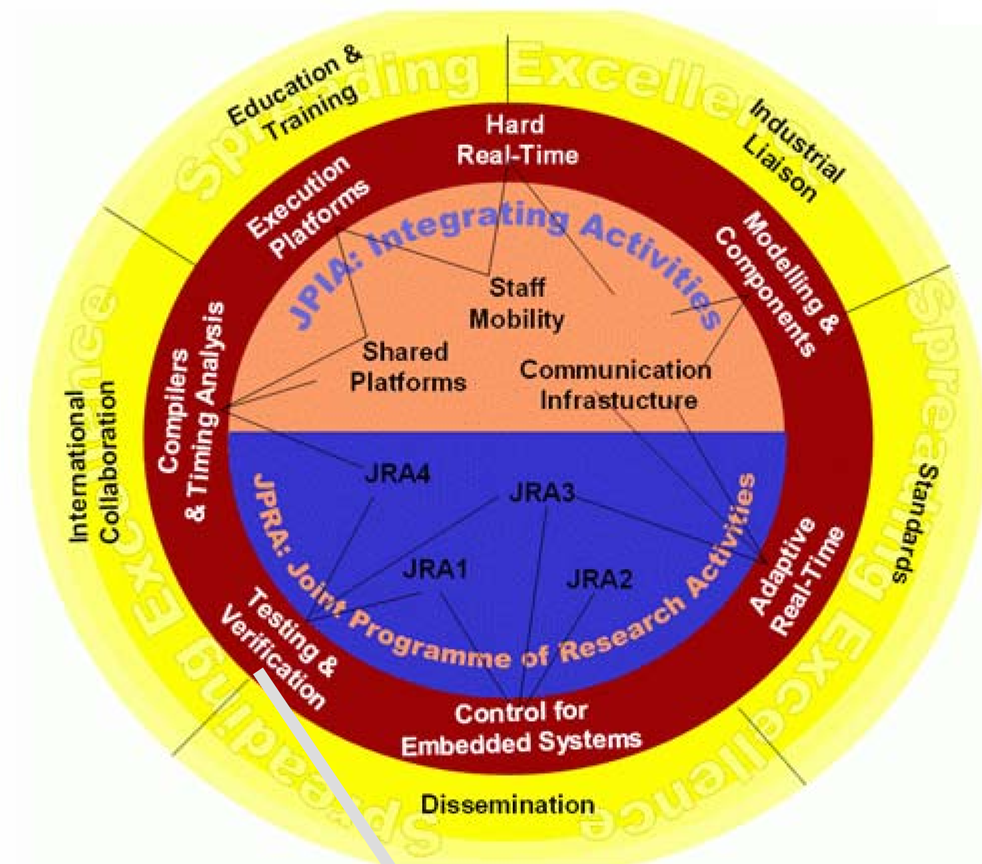
IO TECHNOLOGIES

Developmen

PAJ SYSTEMTEKNIK

BRICS
Basic Research
in Computer Science

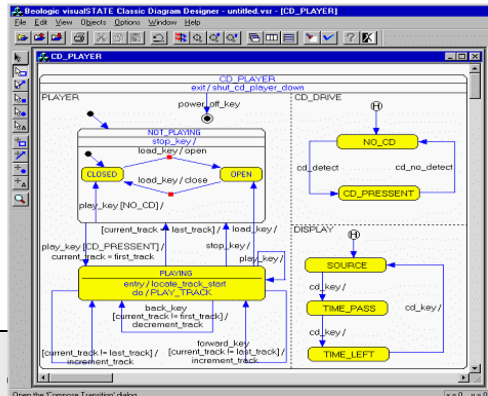
European Network of Excellence



Testing & Verification coordinator

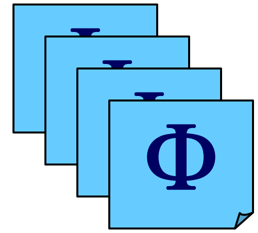
Verification and Testing

Model




```
/* Wait for 
void OS_Wait(void);

/* Operating system visualSTATE process. Mimics a OS process for a
 * visualSTATE system. In this implementation this is the mainloop
 * interfacing to the visualSTATE basic API. */
void OS_VS_Process(void);

/* Define completion code variable. */
unsigned char cc;

void HandleError(unsigned char ccArg)
{
  printf("Error code %c detected, exiting application.\n", ccArg);
  exit(ccArg);
}


/* In d-241 we only use the OS_Wait call. It is used to simulate a
 * system. It purpose is to generate events. How this is done is up to
 * you.
 */
void OS_Wait(void)
{
  /*  Ignore the parameters; just retrieve events from the keyboard and
   *  put them into the queue. When EVENT_UNDEFINED is read from the
   *  keyboard, return to the calling process. */
  SEM_EVENT_TYPE event;
  int num;
```

Code



Req

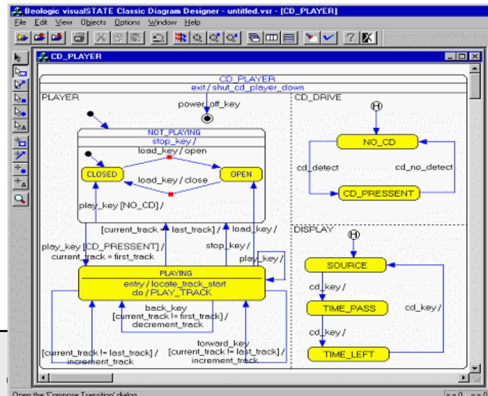
Running System

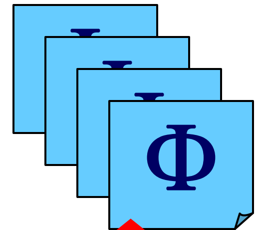
BRICS Basic Research in Computer Science

Verification and Testing

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Model





Req

```
/* Wait for
void OS_Wait(void);

/* Operating system visualSTATE process. Mimics a OS process for
 * visualSTATE system. In this implementation this is the mainloop
 * interfacing to the visualSTATE basic API. */
void OS_VS_Process(void);

/* Define completion code variable. */
unsigned char cc;

void HandleError(unsigned char ccArg)
{
  printf("Error code %c detected, exiting application.\n", ccArg);
  exit(ccArg);
}


/* In d-241 we only use the OS_Wait call. It is used to simulate a
 * system. It purpose is to generate events. How this is done is up to
 * you.
 */
void OS_Wait(void)
{
  /*  Ignore the parameters; just retrieve events from the keyboard and
   *  put them into the queue. When EVENT_UNDEFINED is read from the
   *  keyboard, return to the calling process. */
  SEM_EVENT_TYPE event;
  int num;
```

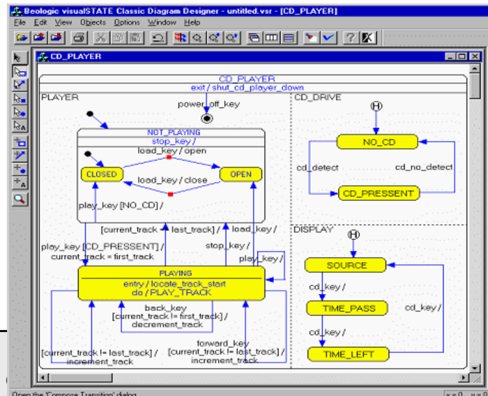
Code

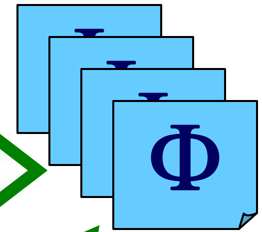
Running System

Basic Research
in Computer Science

Verification and Testing

Model





Req

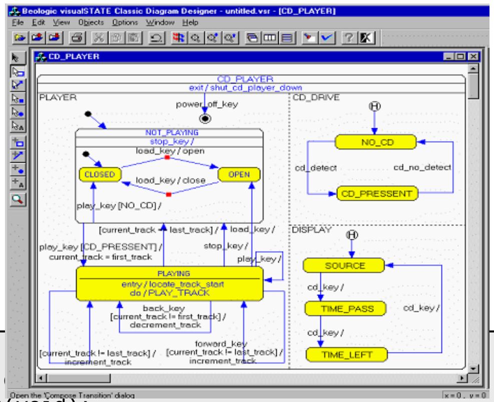
```
/* Wait for
void OS_Wait(void);

/* Operating system visualSTATE process. Mimics a OS process for a
 * visualSTATE system. In this implementation this is the mainloop
 * interfacing to the visualSTATE basic API. */
void OS_VS_Process(void);

/* Define completion code variable. */
unsigned char cc;

void HandleError(unsigned char ccArg)
{
  printf("Error code %c detected, exiting application.\n", ccArg);
  exit(ccArg);
}


/* In d-241 we only use the OS_Wait call. It is used to simul
 * system. It purpose is to generate events. How this is done    to
 * you.
 */
void OS_Wait(void)
{
  /*  Ignore the parameters; just retrieve events from the keyboard and
   *  put them into the queue. When EVENT_UNDEFINED is read from the
   *  keyboard, return to the calling process. */
  SEM_EVENT_TYPE event;
  int num;
```

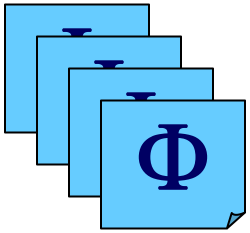
Code

Running System

Verification and Testing

Model





Req

```
/* Wait for 
void OS_Wait(void);

/* Operating system visualSTATE process. Mimics a OS p
 * visualSTATE system. In this implementati
 * interfacing to the visualSTATE
void OS_VS_Process(void);

/* Define completion code v
unsigned char cc;

void HandleError(unsigned ch
{
  printf("Error code %c detec
  exit(ccArg);
}


/* In d-241 we only use the OS_V      call. It is used to simulate a
 * system. It purpose is to generate events. How this is done is up to
 * you.
 */
void OS_Wait(void)
{
  /*  Ignore the parameters; just retrieve events from the keyboard and
   *  put them into the queue. When EVENT_UNDEFINED is read from the
   *  keyboard, return to the calling process. */
  SEM_EVENT_TYPE event;
  int num;
```

- Verification
  - Code/Model wrt Req
- Testing
  - System wrt Model/Req

Running System

Code

BRICS Basic Research in Computer Science

CISS CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Why Verification and Testing

- **POTENTIAL**:
  30-40% of production time is currently spend on elaborate, ad-hoc testing
  - Errors expensive and difficult to fix
  - The potential of existing/improved testing methods and tools is enormous
  - Time-to-market may be shortened considerable by verification and performance analyses of early designs

- **COMMONALITY**:
  Transversal topic, interacts with all other topics in embedded systems design:
  - Modelling and Components (verification, model-based testing)
  - Hard and adaptive real time (optimal scheduling & schedulability analysis)
  - Execution platform (performance analysis, security)
  - Compilers and timing analysis (WCET and compact code-generation)

Why Verification and Testing

- **IMPORTANCE for EMBEDDED SYSTEMS**
  - Often safety critical
  - Often economical critical
  - Hard to patch

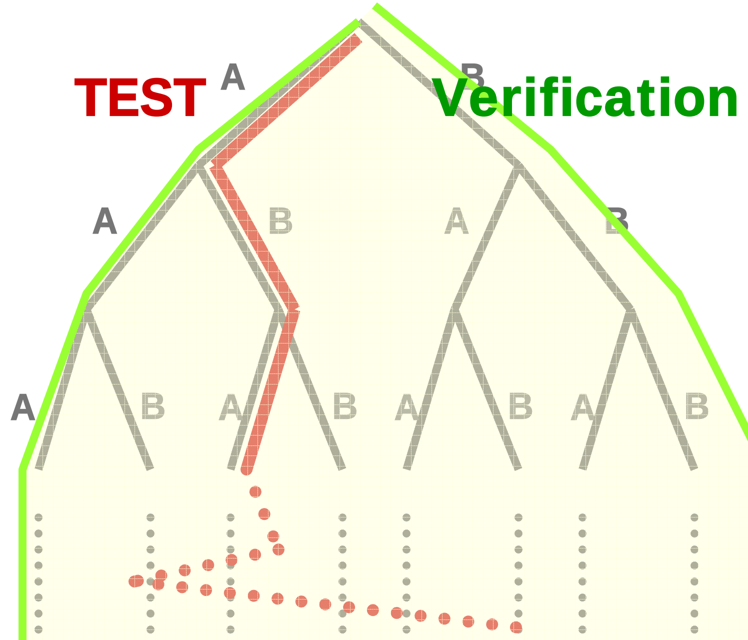
- **CHALLENGES for EMBEDDED SYSTEMS**
  - Correctness of embedded systems depend crucially on use of *resources* (real-time, memory, bandwidth, energy). Need for verification of and conformance testing with respect to quantitative models.
  - Participation in mobile ad-hoc networks require particular attention to *security* aspects.

BRICS
Basic Research
in Computer Science

Test versus Verification

E F E E G H ... H A

T1 T3 T5 T1 ... T4 T3



2^n sequences of length n

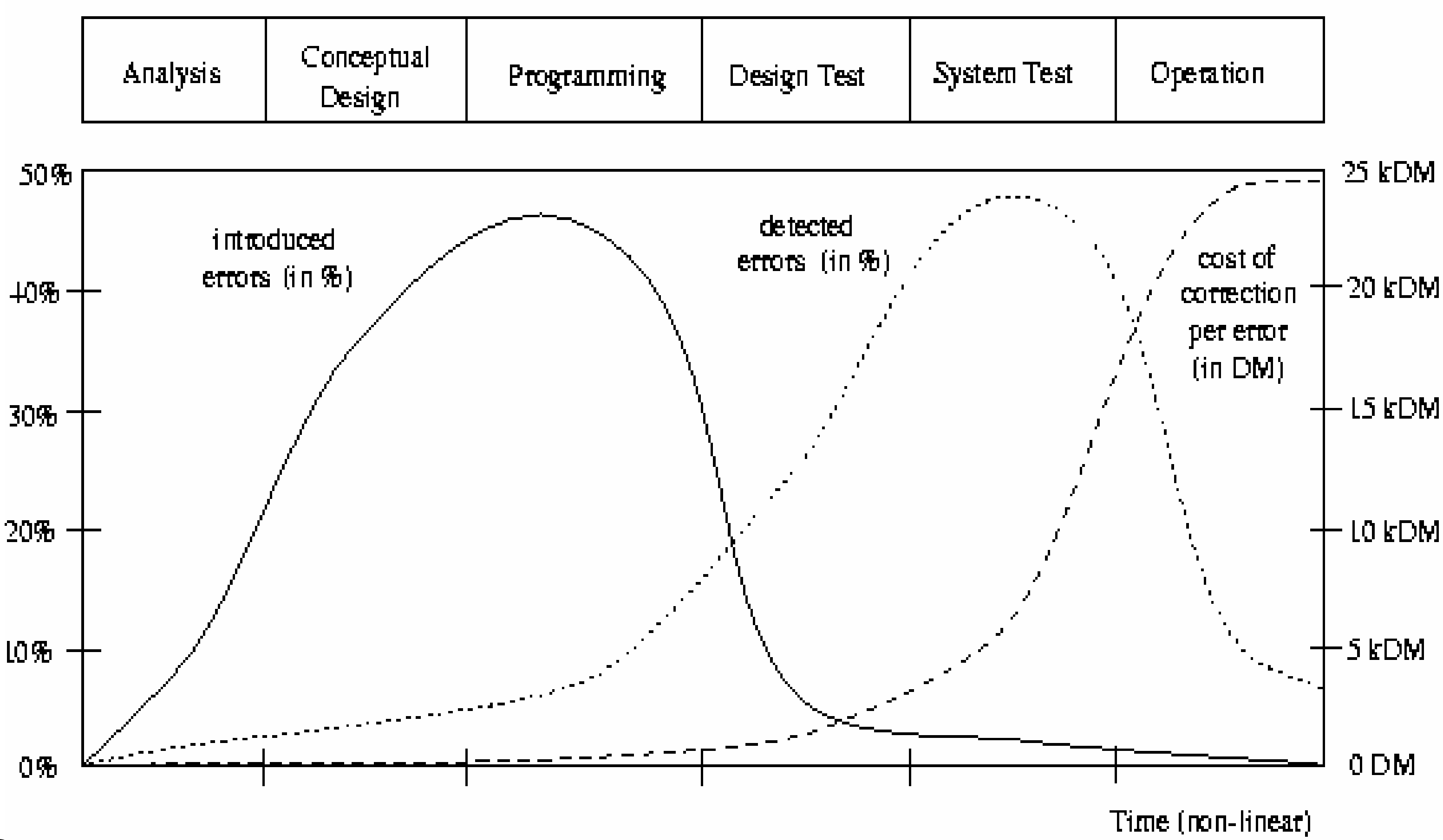
Deadlock identified using
Verification
After sequence of
2000
telegrams / < 1min

UPPAAL

BRICS
Basic Research
in Computer Science

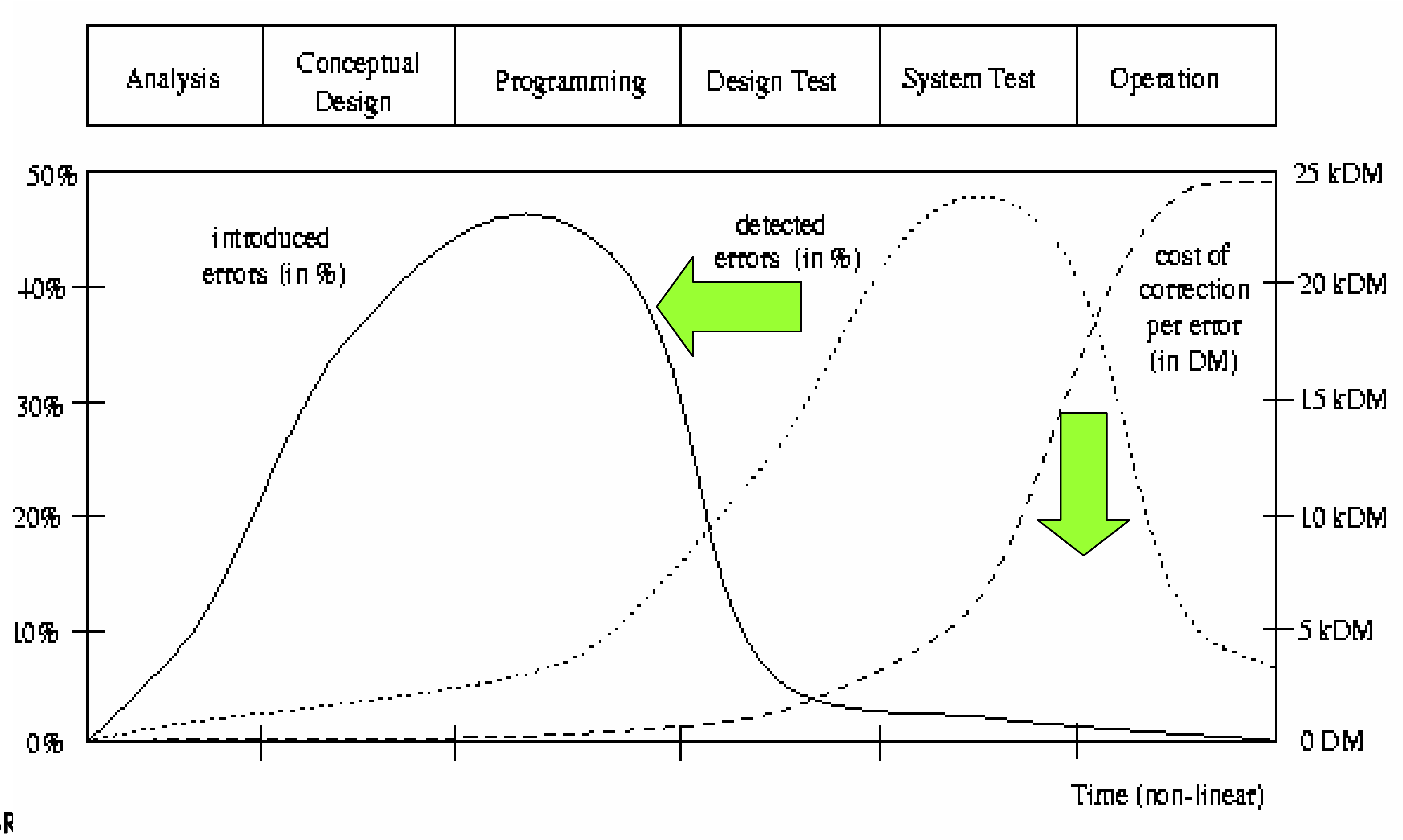
Introducing, Detecting and Repairing Errors

Liggesmeyer 98



Introducing, Detecting and Repairing Errors

Liggesmeyer 98



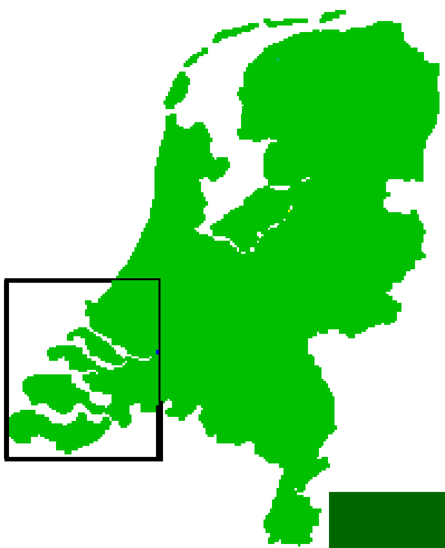
A very complex system

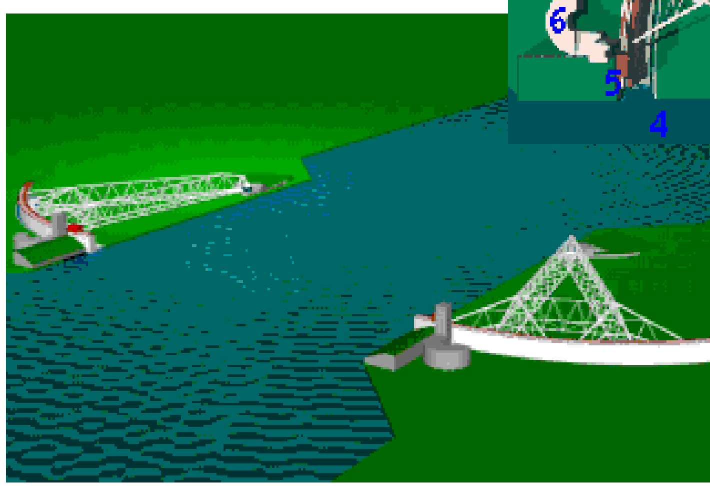
Klaus Havelund, NASA

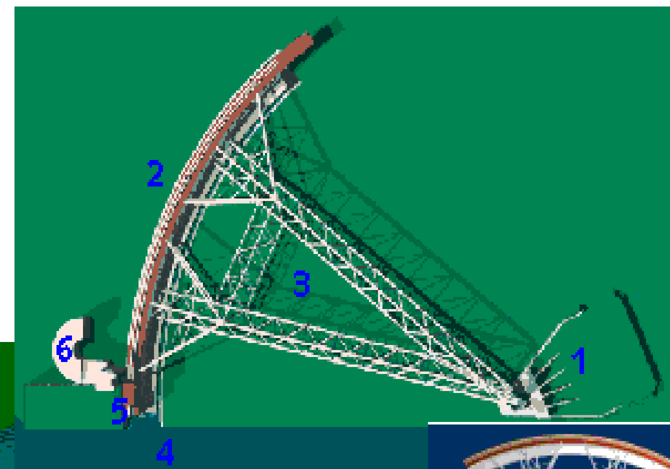
BRICS Basic Research in Computer Science

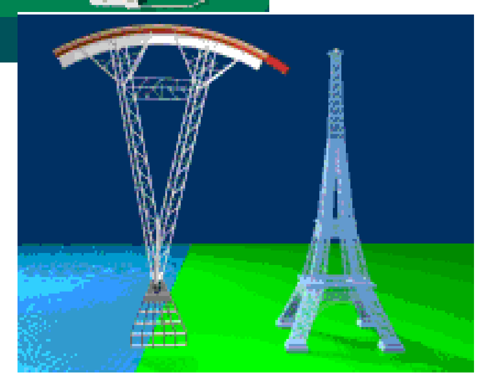
CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Rotterdam Storm Surge Barrier









BRICS
Basic Research
in Computer Science

Spectacular software bugs
Ariane 5

- The first Ariane 5 rocket was launched in June, 1996. It used software developed for the successful Ariane 4. The rocket carried two computers, providing a backup in case one computer failed during launch. Forty seconds into its maiden flight, the rocket veered off course and exploded. The rocket, along with $500 million worth of satellites, was destroyed.

- Ariane 5 was a much more powerful rocket and generated forces that were larger than the computer could handle. Shortly after launch, it received an input value that was too large. The main and backup computers shut down, causing the rocket to veer off course.

BRICS
Basic Research
in Computer Science

Spectacular software bugs U.S.S. Yorktown, U.S. Navy

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

- In 1998, the USS Yorktown became the first ship to test the US Navy's Smart Ship program. The Navy planned to use off-the-shelf computers and software instead of expensive U.S.S. Yorktown, courtesy of U.S. Navy custom-made machines. A sailor mistakenly entered a zero for a data value on a computer. Within minutes, Yorktown was dead in the water. It was several hours before the ship could move again.
- When the sailor entered the mistaken number, the computer tried to divide by zero, which isn't possible. The software didn't check to see if the inputs were valid before computing and generated an invalid answer that was used by another computer. The error cascaded several computers and eventually shut down the ship's engines.

BRICS
Basic Research
in Computer Science

Spectacular software bugs
Moon or Missiles

- The United States established the Ballistic Missile Early Warning System (BMEWS) during the Cold War to detect a Soviet missile attack. On October 5, 1960 the BMEWS radar at Thule, Greenland detected something. Its computer control system decided the signal was made by hundreds of missiles coming toward the US.

- The radar had actually detected the Moon rising over the horizon. Unfortunately, the BMEWS computer had not been programmed to understand what the moon looked like as it rose in the eastern sky, so it interpreted the huge signal as Soviet missiles. Luckily for all of us, the mistake was realized in time.

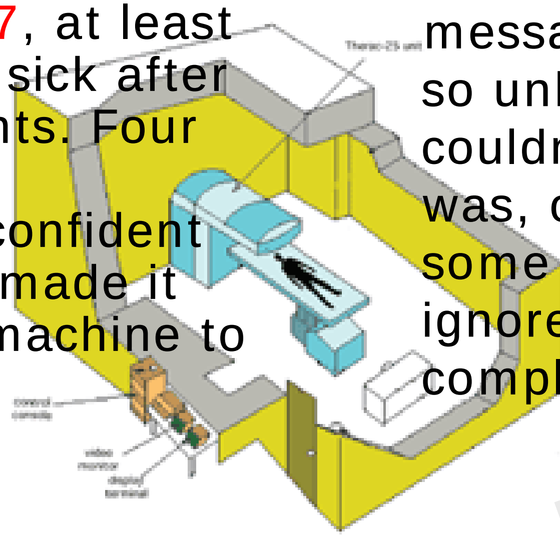
BRICS Basic Research in Computer Science

Spectacular Software Bugs .... continued

- INTEL Pentium II floating-point division
  470 Mill US $
- Baggage handling system, Denver
  1.1 Mill US $/day for 9 months
- Mars Pathfinder
- ....

Spectacular software bugs Therac 25

- The Therac-25 radiation therapy machine was a medical device that used beams of electrons or photons to kill cancer cells. Between 1985-1987, at least six people got very sick after Therac-25 treatments. Four of them died. The manufacturer was confident that their software made it impossible for the machine to harm patients.



- The Therac-25 was withdrawn from use after it was determined that it could deliver fatal overdoses under certain conditions. The software would shut down the machine before delivering an overdose, but the error messages it displayed were so unhelpful that operators couldn't tell what the error was, or how serious it was. In some cases, operators ignored the message completely.

"Malfunction 54"

"H-tilt"

IEEE Computer, Vol. 26, No. 7, July 1993, pp. 18-41

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

More complex systems

BRICS
Basic Research
in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

A simple program

```
int x=100;

Process INC
        do
        :: x<200 --> x:=x+1
        od

Process DEC
        do
        :: x>0 --> x:=x-1
        od

Process RESET
        do
        :: x=200 --> x:=0
        od

( INC || DEC || RESET )
```

Which values may x take ?

Questions/Properties:

- E<>(x>100)
- E<>(x>200)
- A[](x<=200)
- E<>(x<0)
- A[](x>=0)

Possibly (E<>)

Always (A[])

BRICS
Basic Research in Computer Science

Another simple program

What are the possible final values of x ?

```
int x=0;

Process P
    do
        x:=x+1
    10 times

( P || P )
```

```
int x=0;

Process P
int r
    do
        r:=x; r++; x:=r
    10 times

( P || P )
```

Atomic stm

Yet another simple program

```
int x=1;

Process P
    do
        x:=x+x
    forever

( P || P )
```

What are the possible values that x may posses during execution?

```
int x=1;

Process P
int r
    do
        r:=x; r:=x+r; x:=r
    forever

( P || P )
```

Atomic

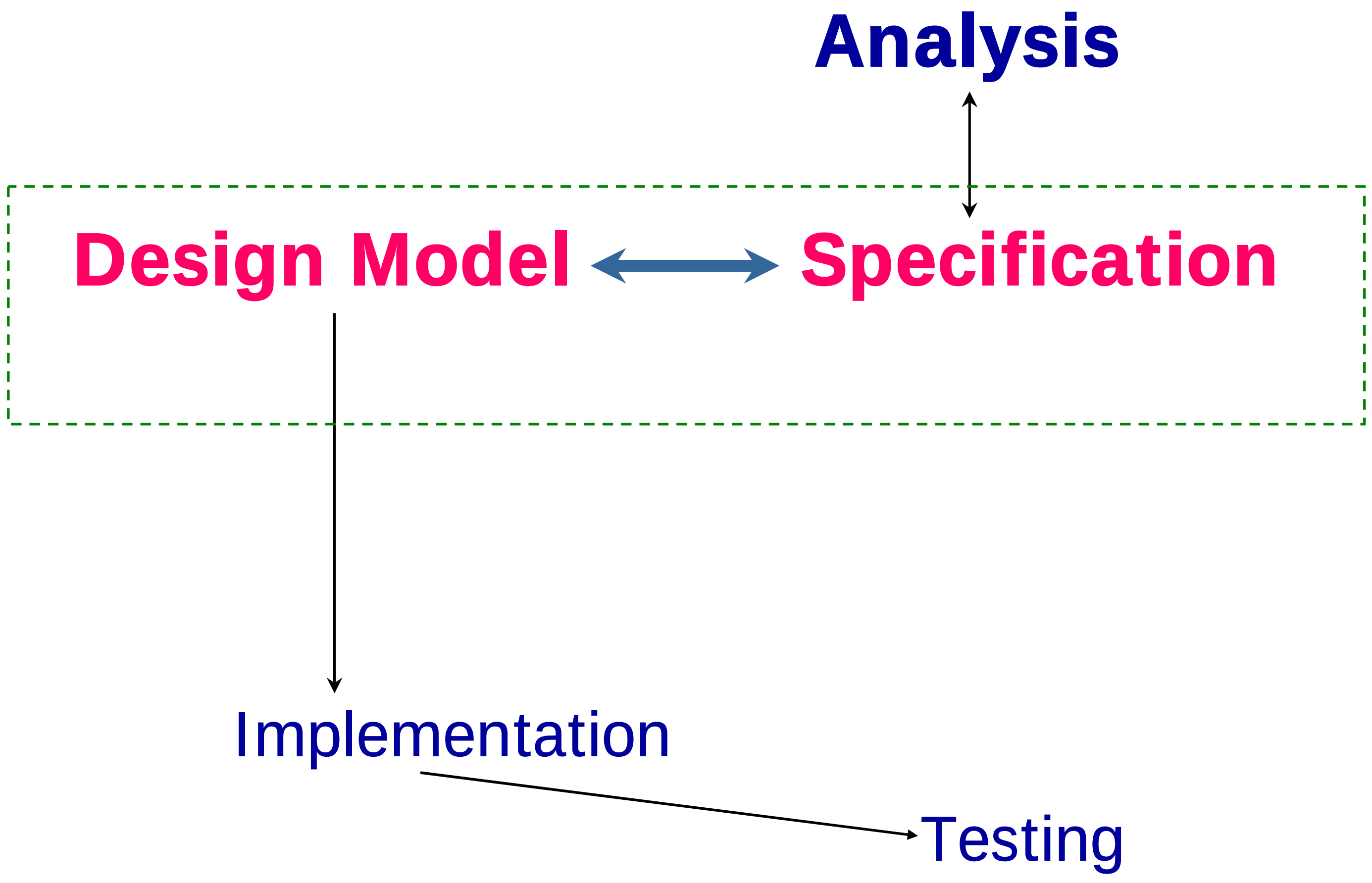
Model-based Approach

Suggested Solution?

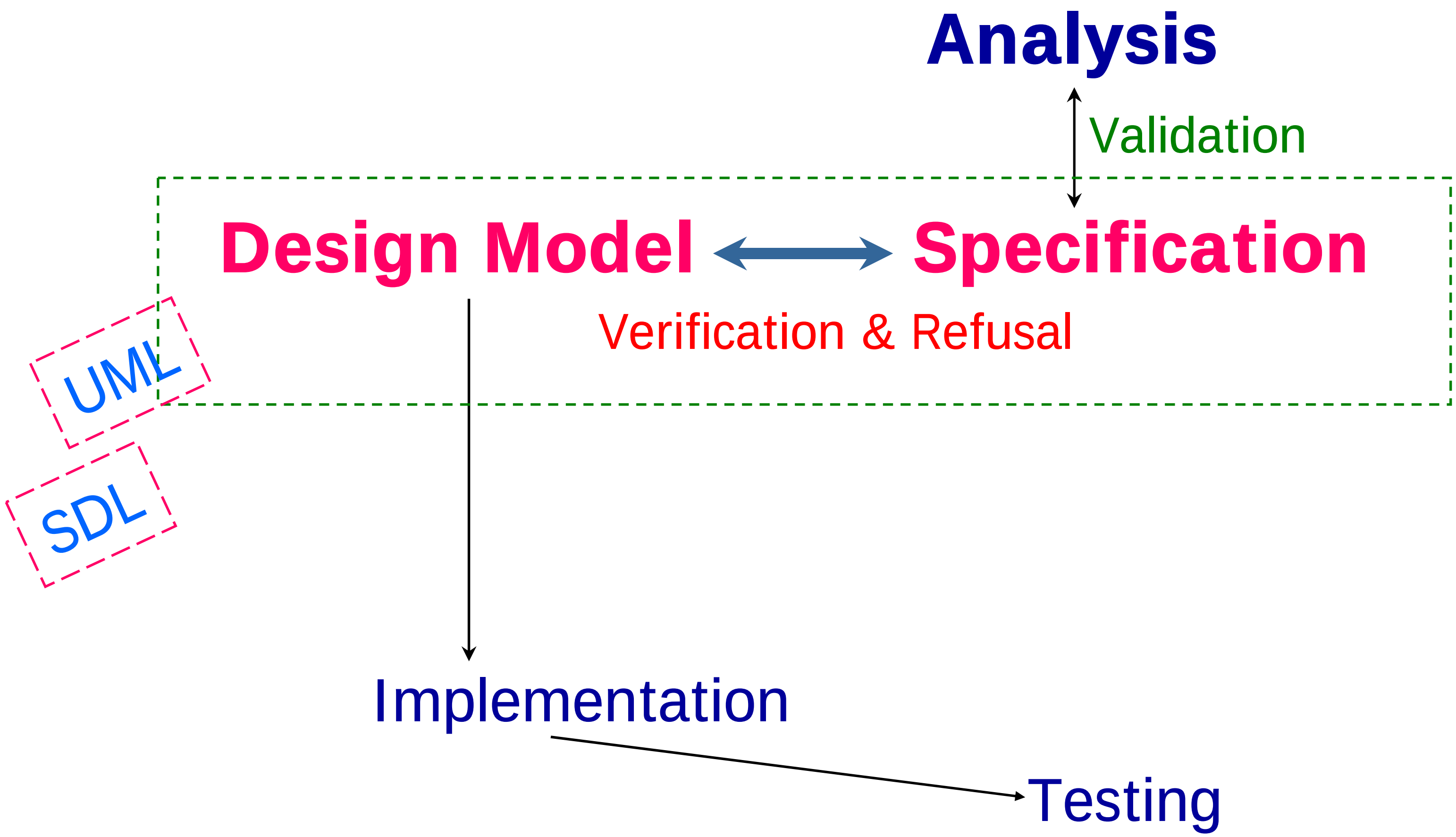
Model based

validation, verfication and testing
of software and hardware

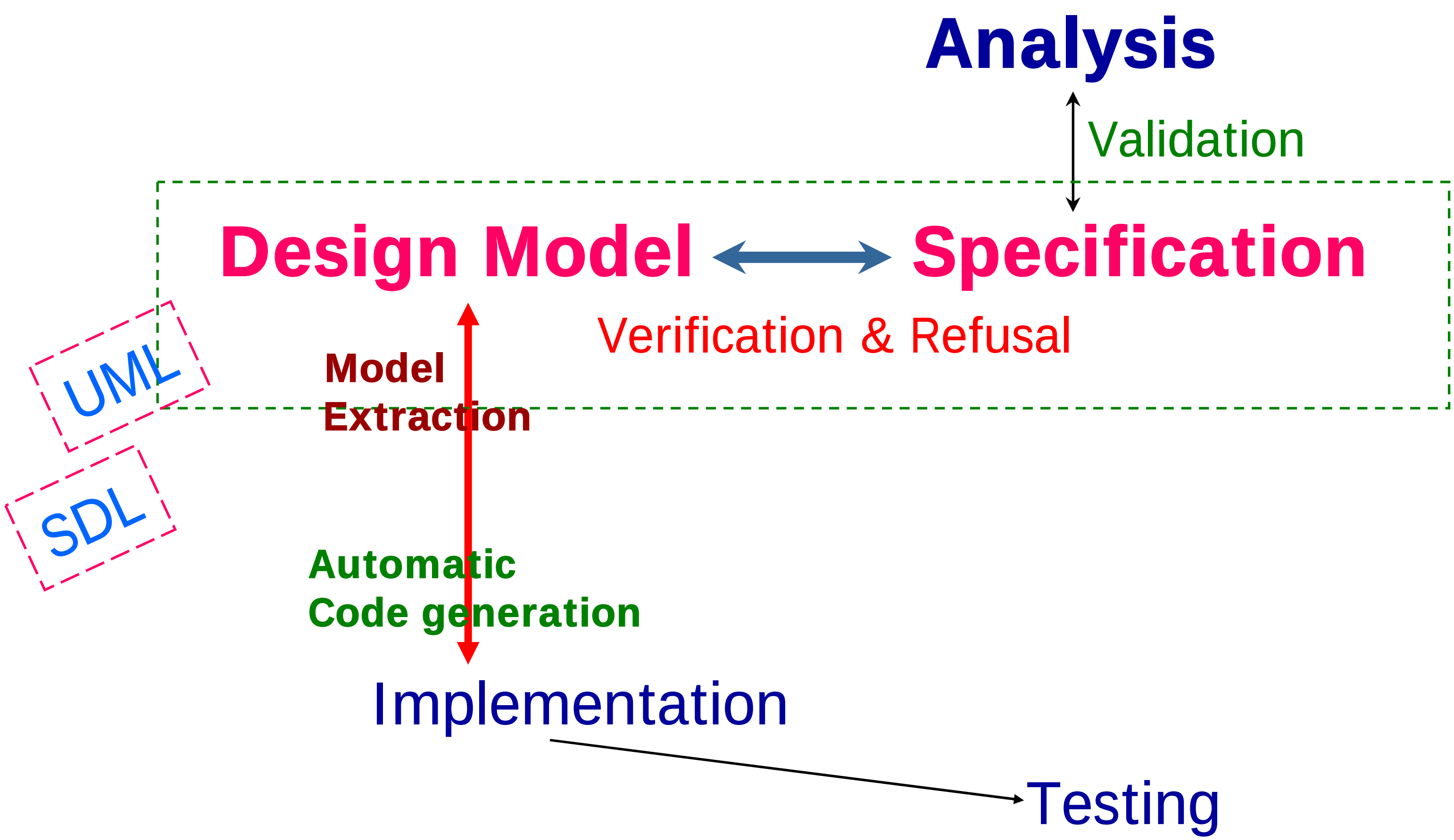
Verification & Validation



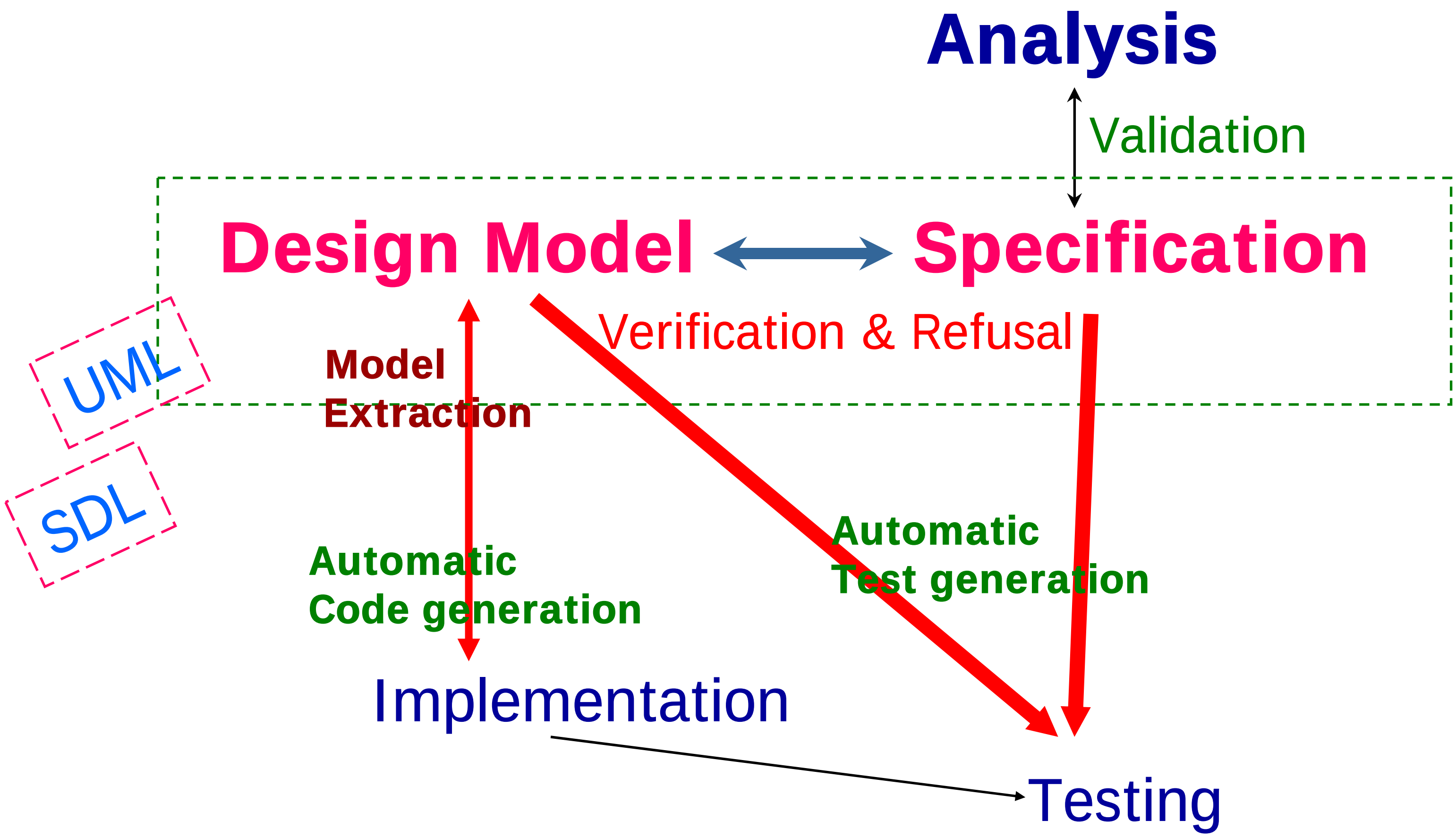
Verification & Validation



Verification & Validation



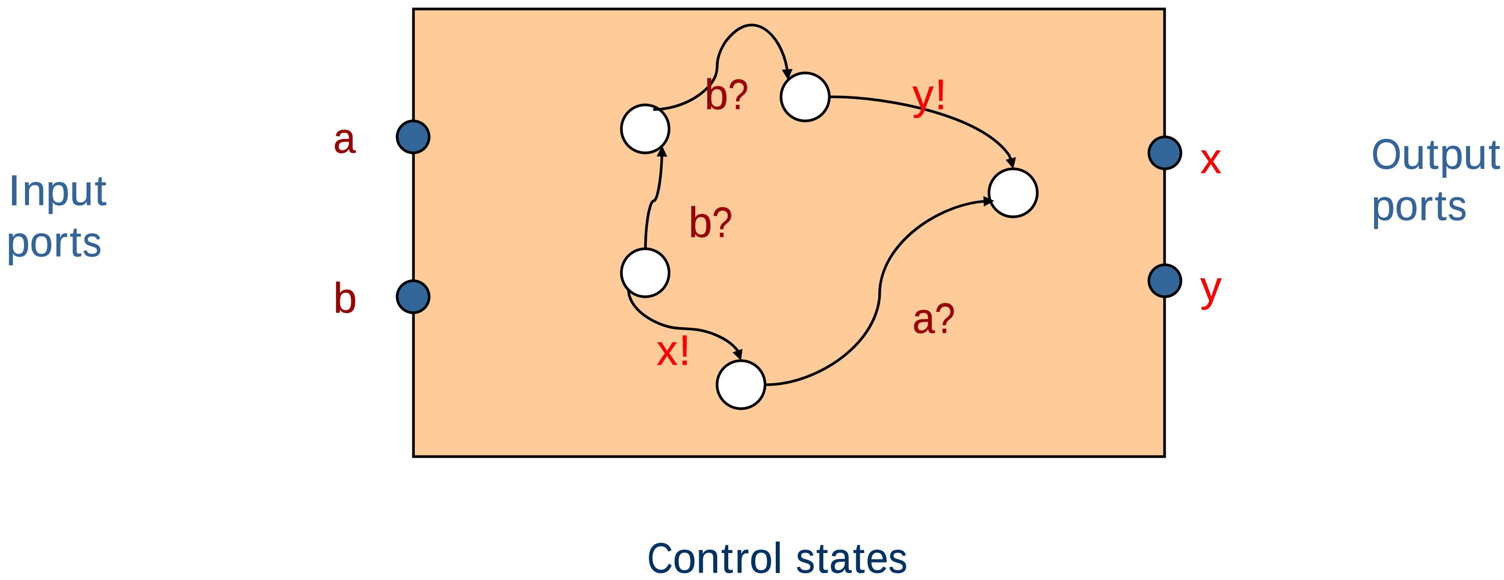
Verification & Validation



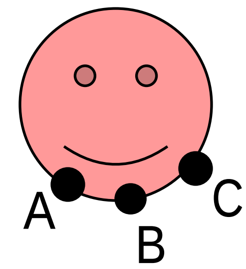
BRICS
Basic Research
in Computer Science

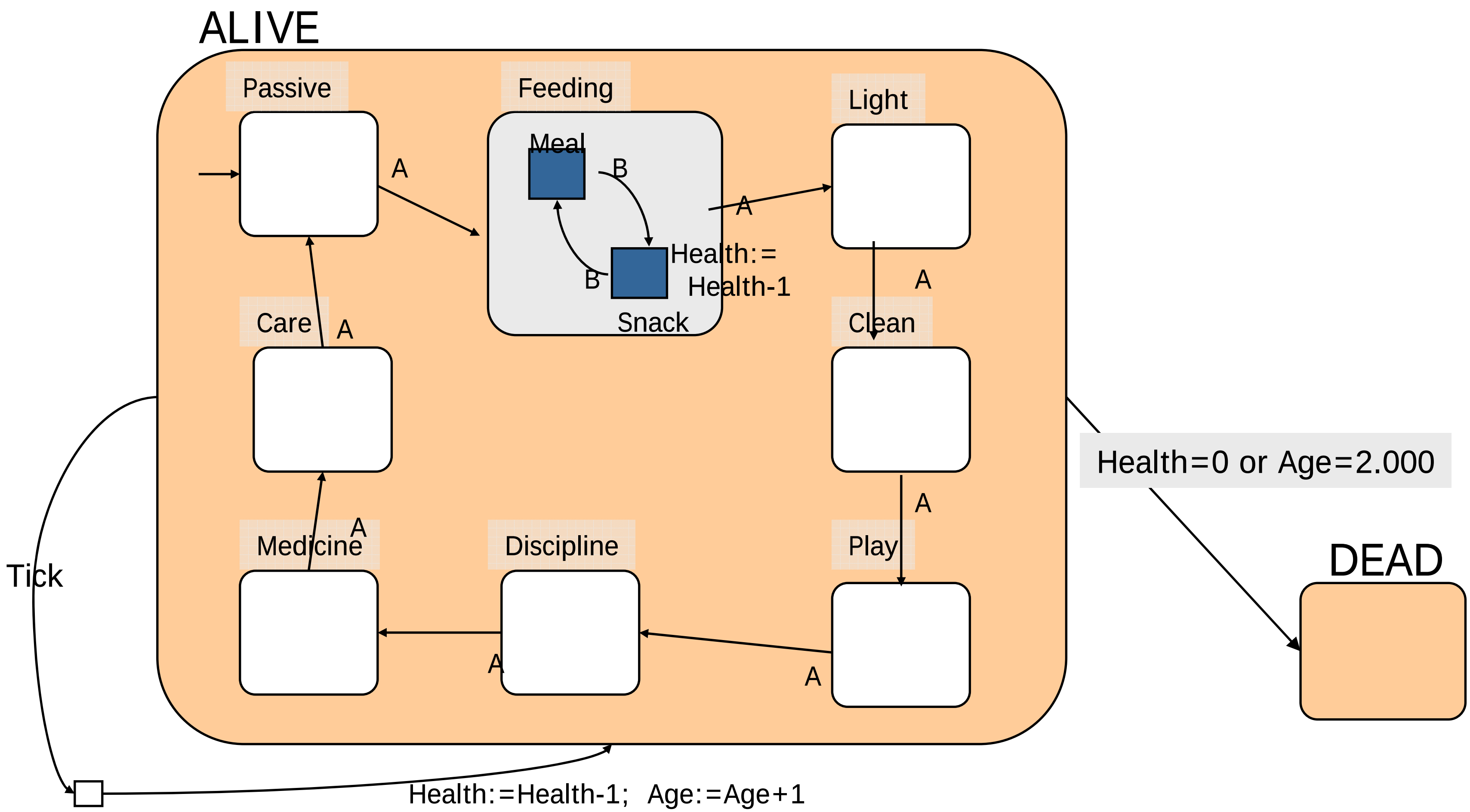
How?

Unified Model = State Machine!

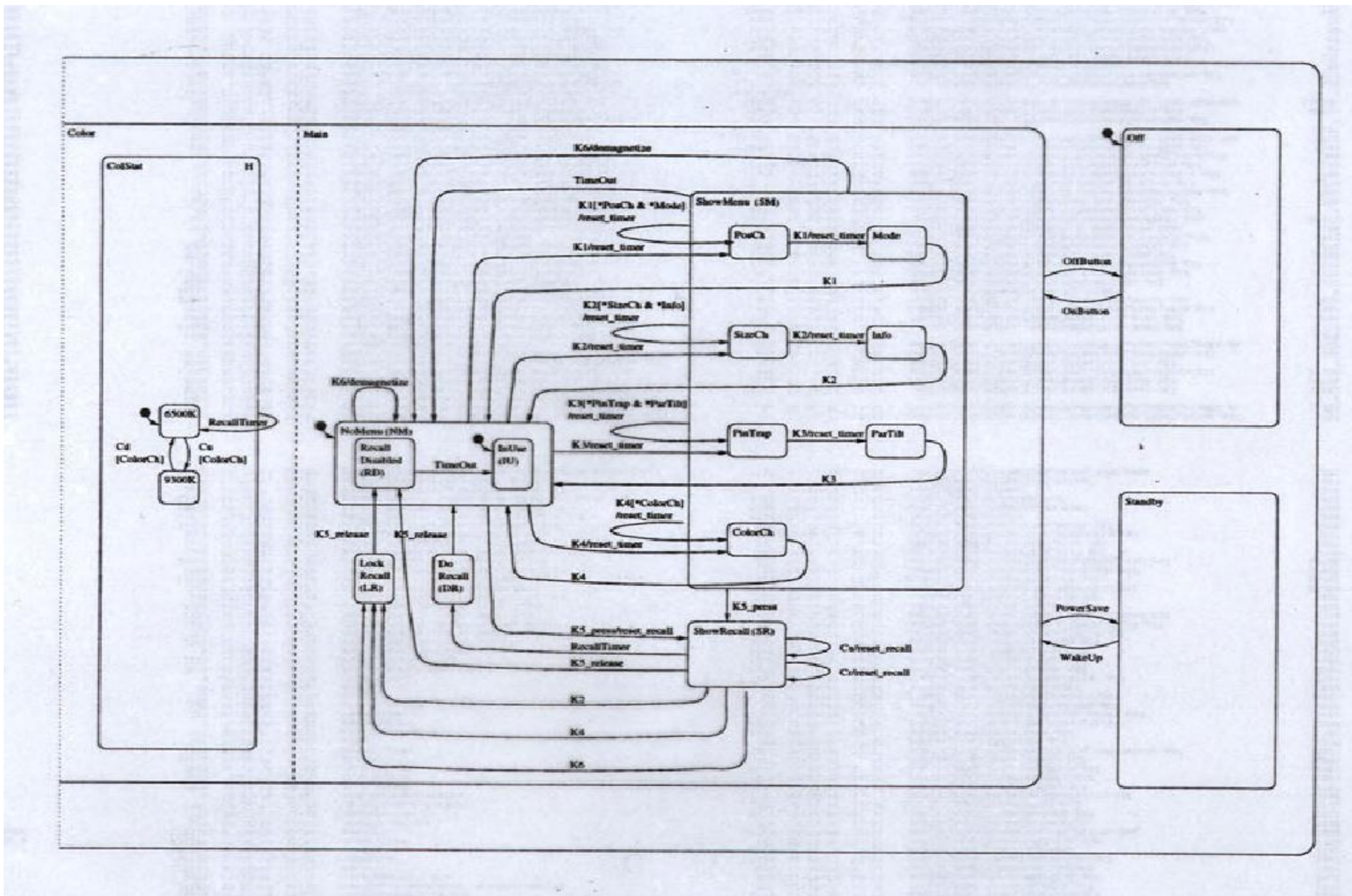


Tamagotchi


A
B
C


ALIVE
Passive
Feeding
Meal
Snack
Health:=
Health-1
Light
Care
Clean
Medicine
Discipline
Play
A
B
Tick
Health:=Health-1; Age:=Age+1
Health=0 or Age=2.000
DEAD

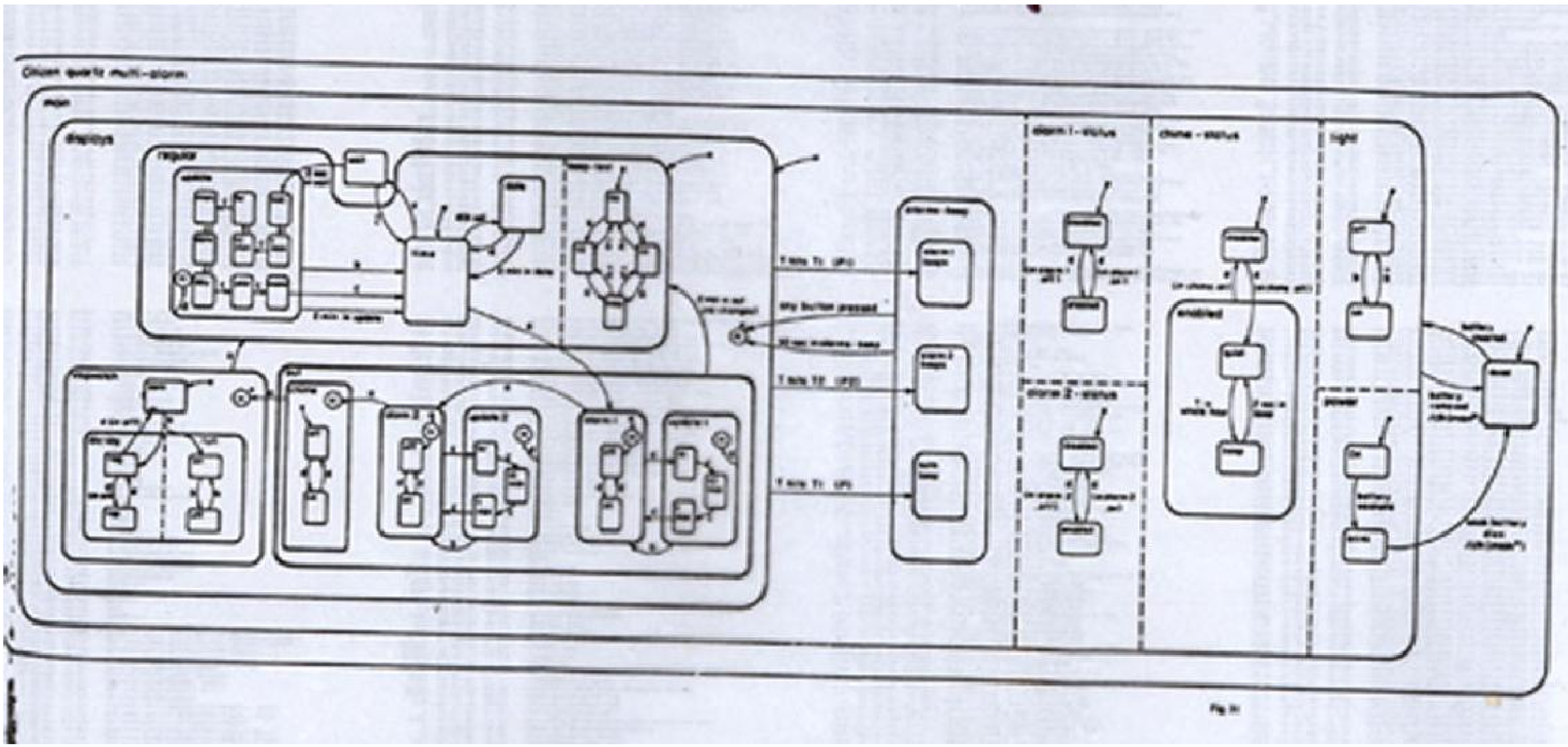
SYNCmaster



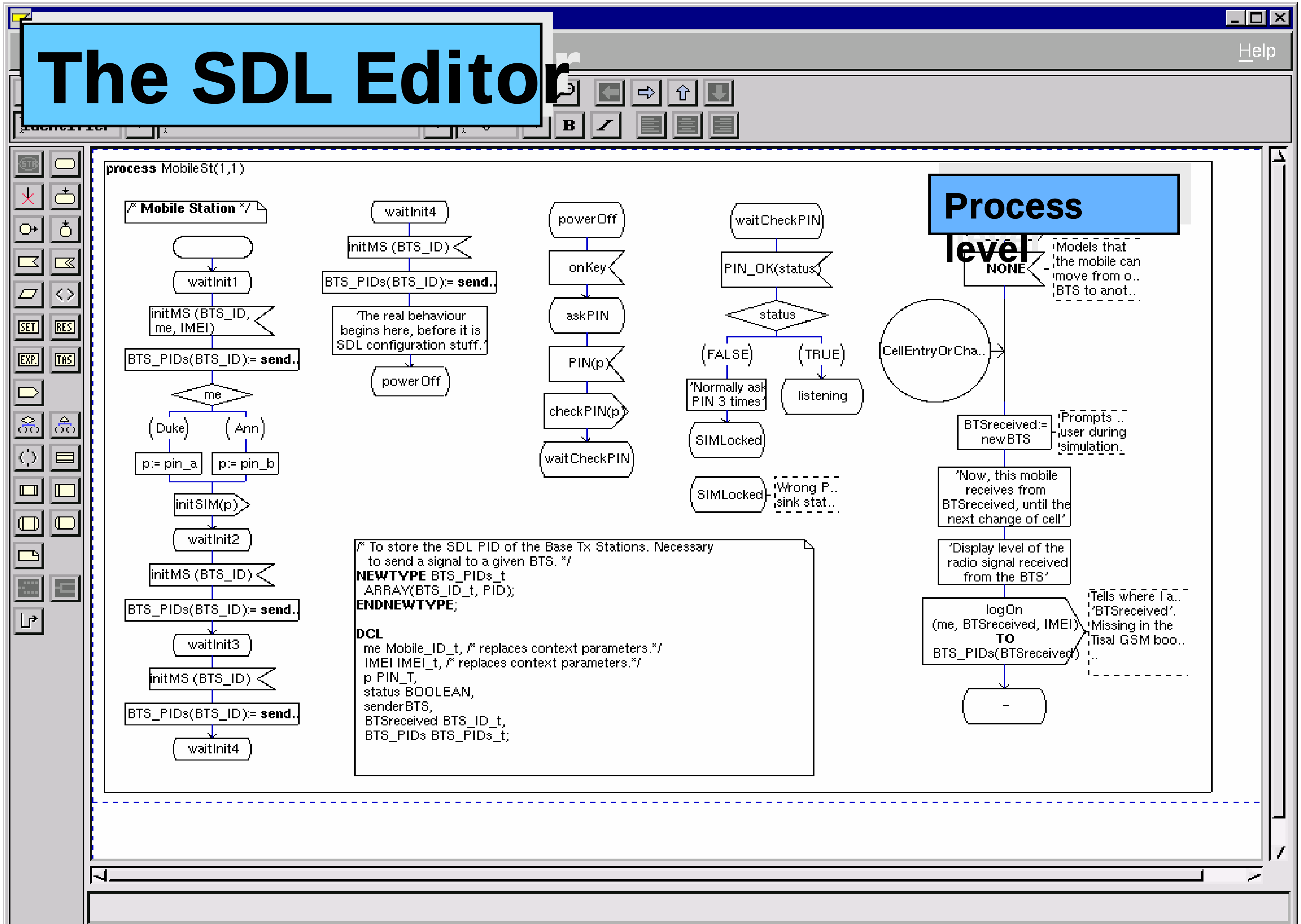
CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

BRICS
Basic Research
in Computer Science

Digital Watch

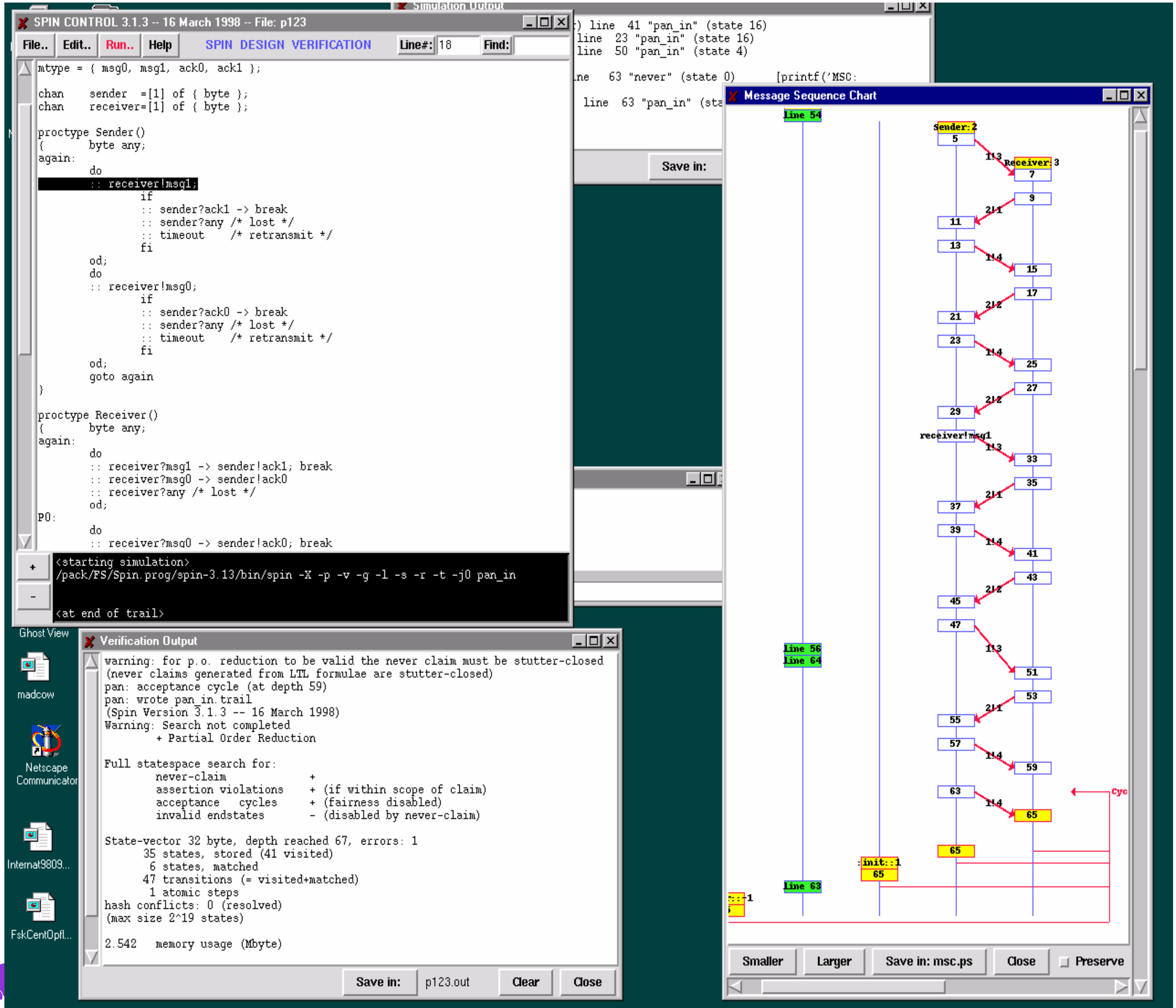


The SDL Editor



Process level

SPIN, Gerald Holzmann AT&T

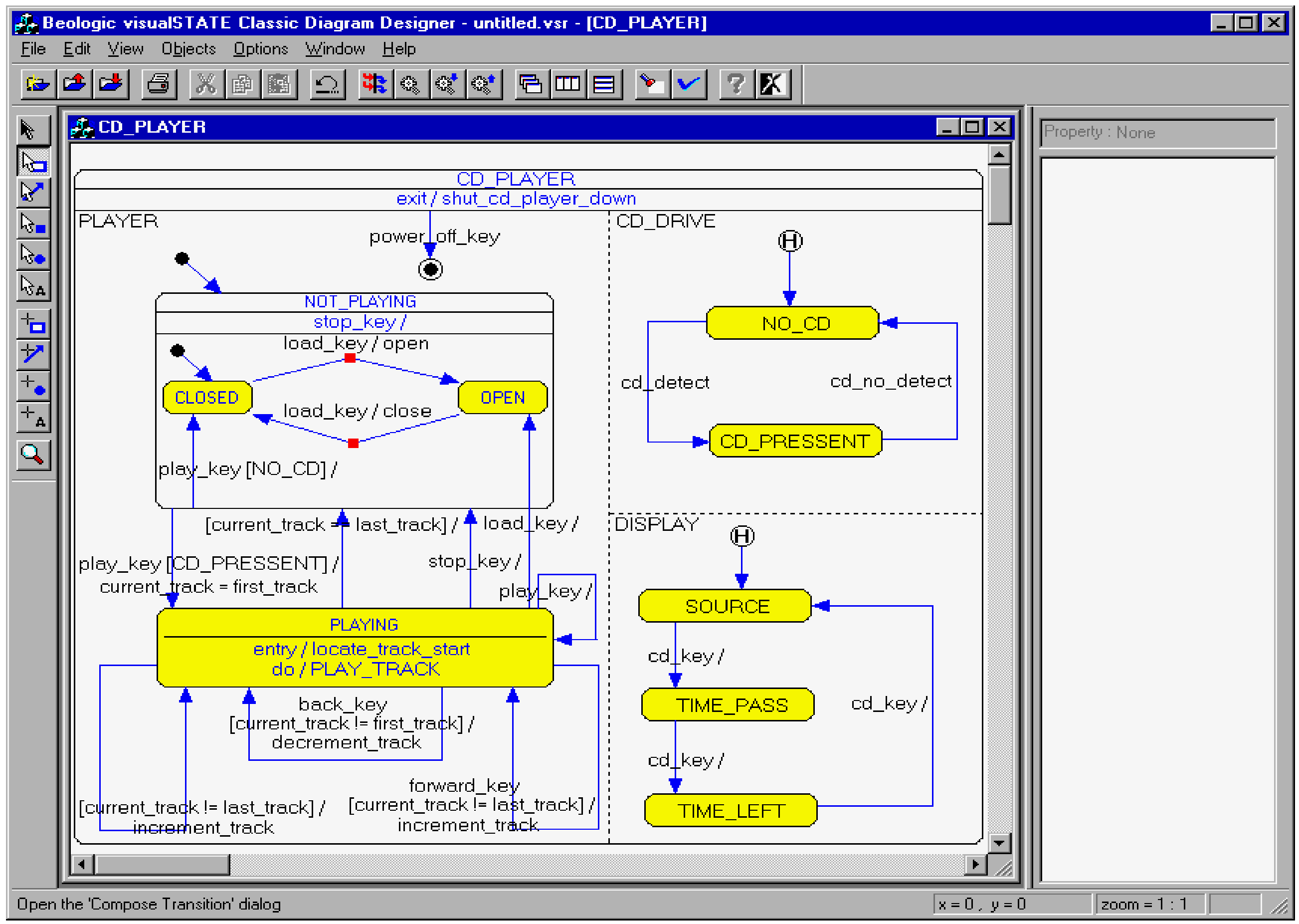


visualSTATE

VVS

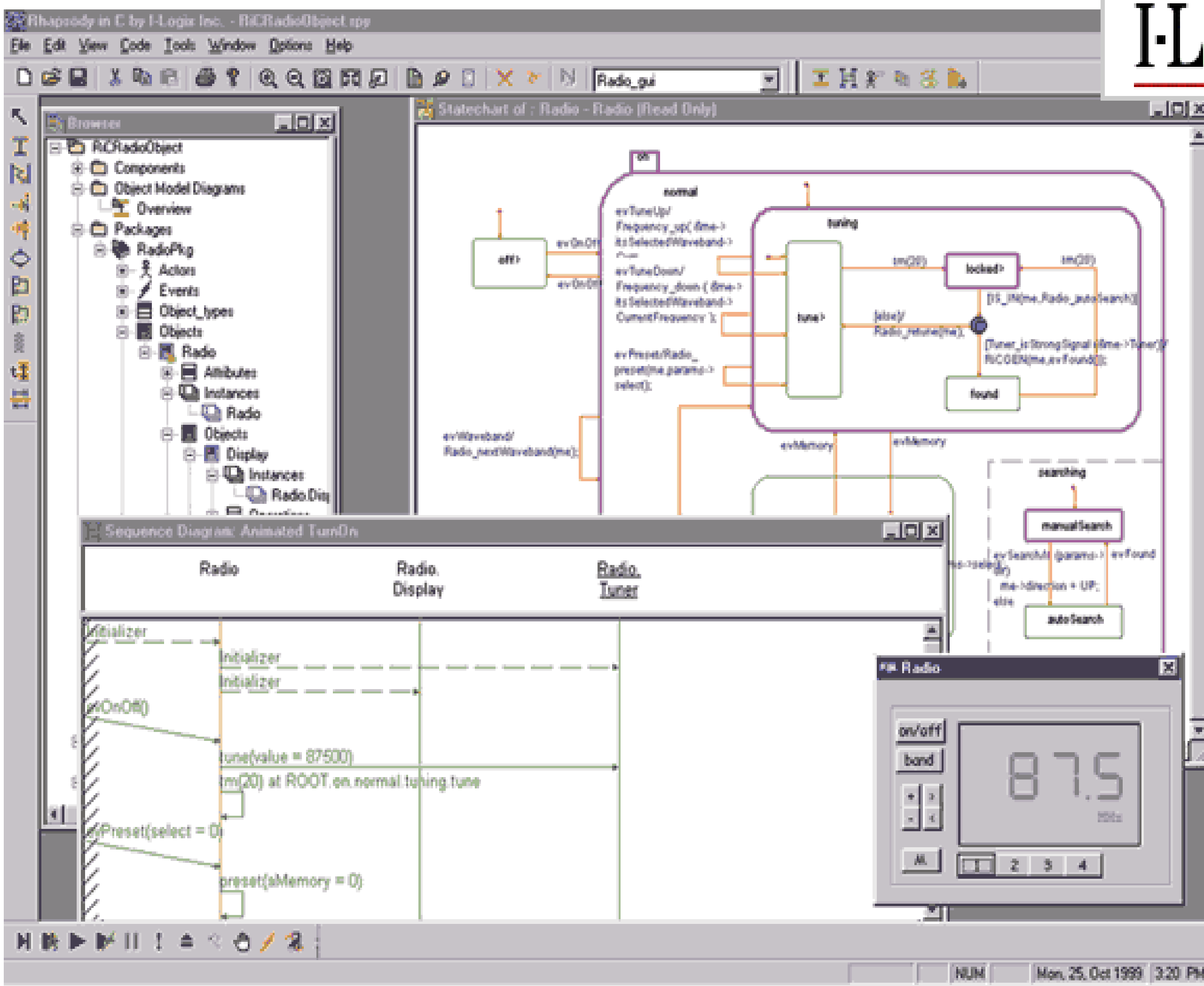
w Baan Visualstate, DTU (CIT project)

- Hierarchical state systems
- Flat state systems
- Multiple and inter-related state machines
- Supports UML notation
- Device driver access



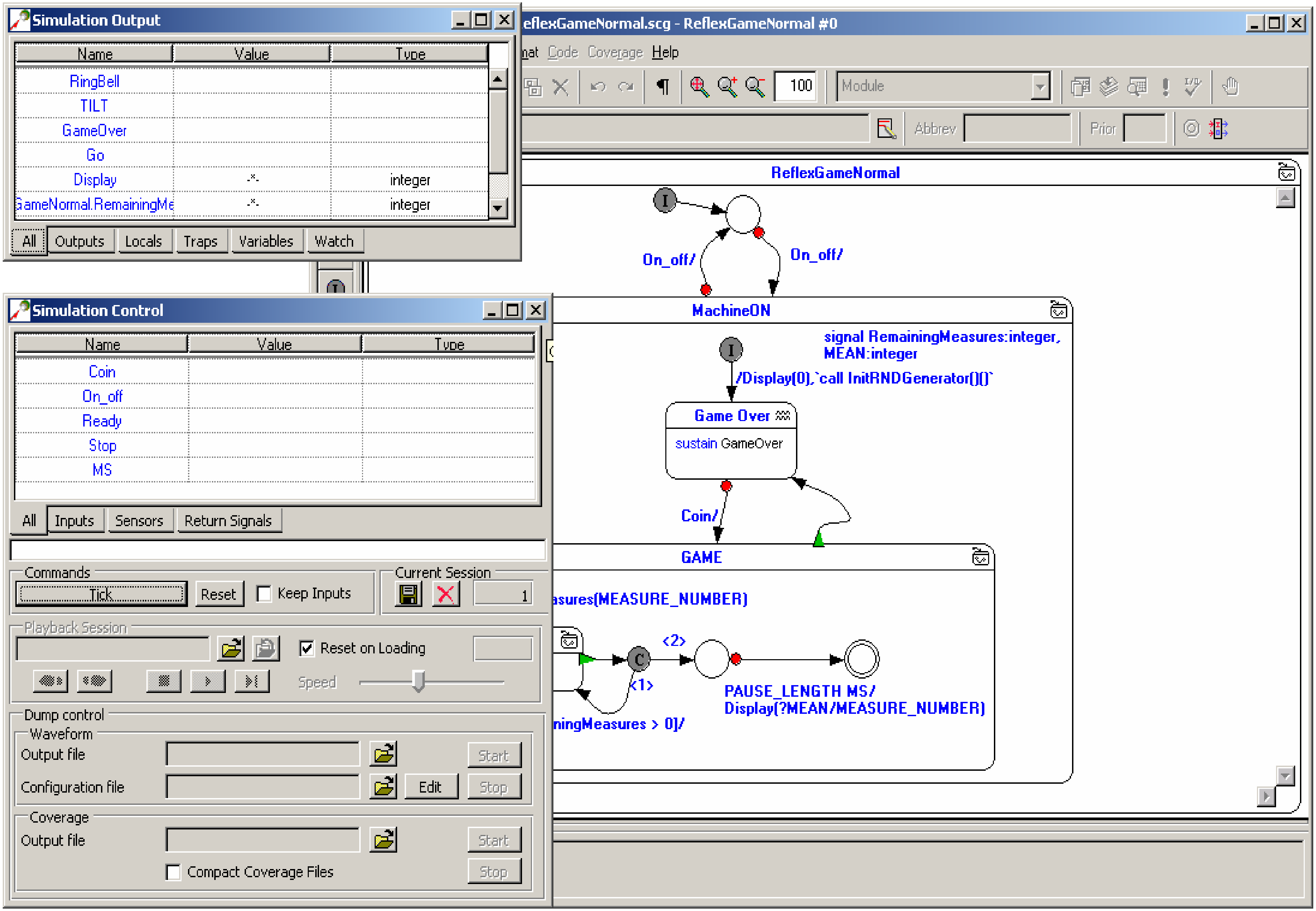
BRICS
Basic Research
in Computer Science

Rhapsody



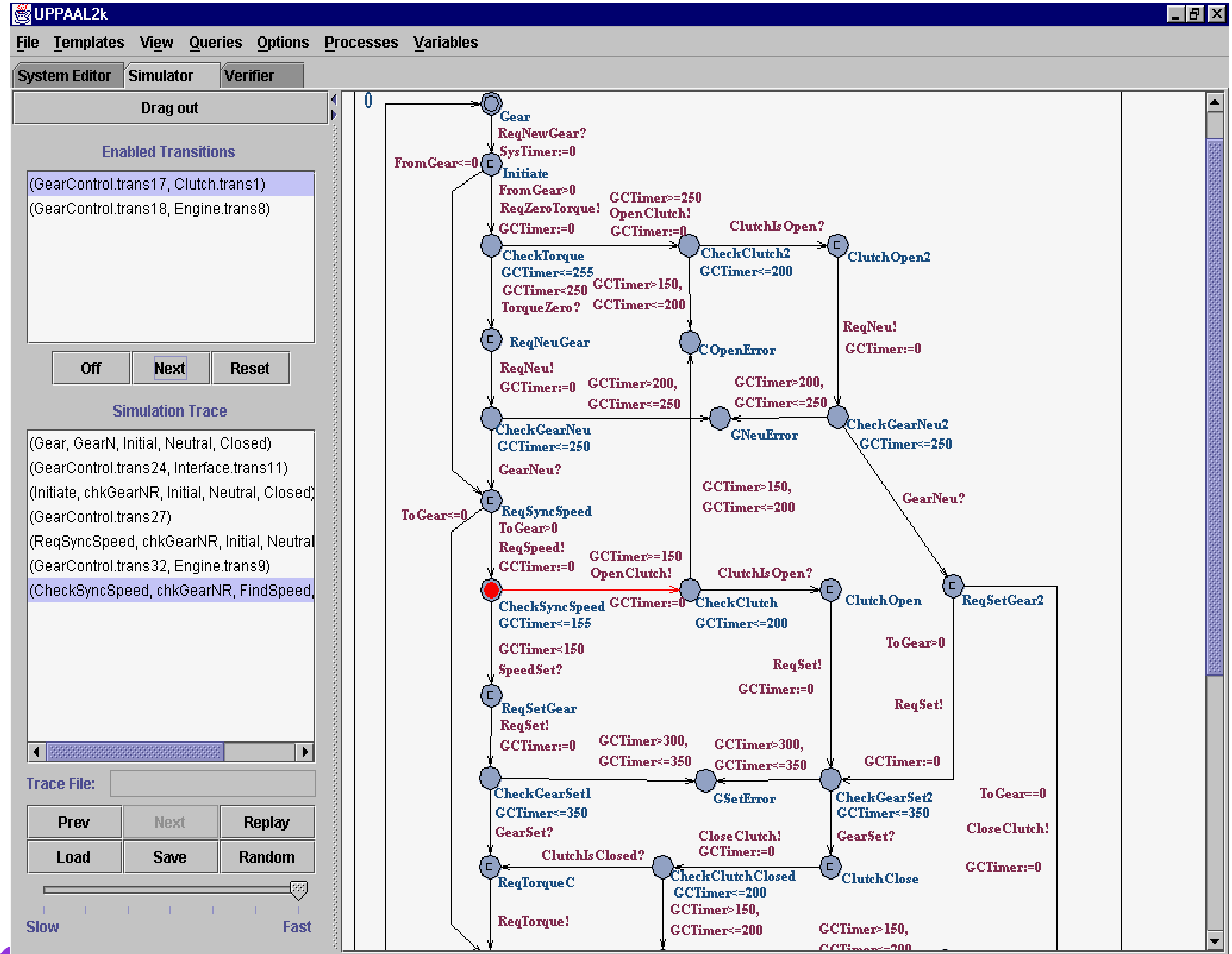
Rhapsody in C by I-Logix Inc. - RiCRadioObject.rpy
File Edit View Code Tools Window Options Help
Radio_gui
Browser
RiCRadioObject
Components
Object Model Diagrams
Overview
Packages
RadioPkg
Actors
Events
Object_types
Objects
Radio
Attributes
Instances
Display
Statechart of : Radio - Radio (Read Only)
normal
tuning
off
locked
found
searching
manualSearch
autoSearch
Sequence Diagram: Animated TurnOn
Radio
Radio. Display
Radio. Tuner
Initializer
evOnOff()
tune(value = 87500)
tm(20) at ROOT.on.normal.tuning.tune
evPreset(select = 0)
preset(sMemory = 0)
Radio
on/off
band
87.5
NUM
Mon, 25, Oct 1999
3:20 PM

ESTEREL



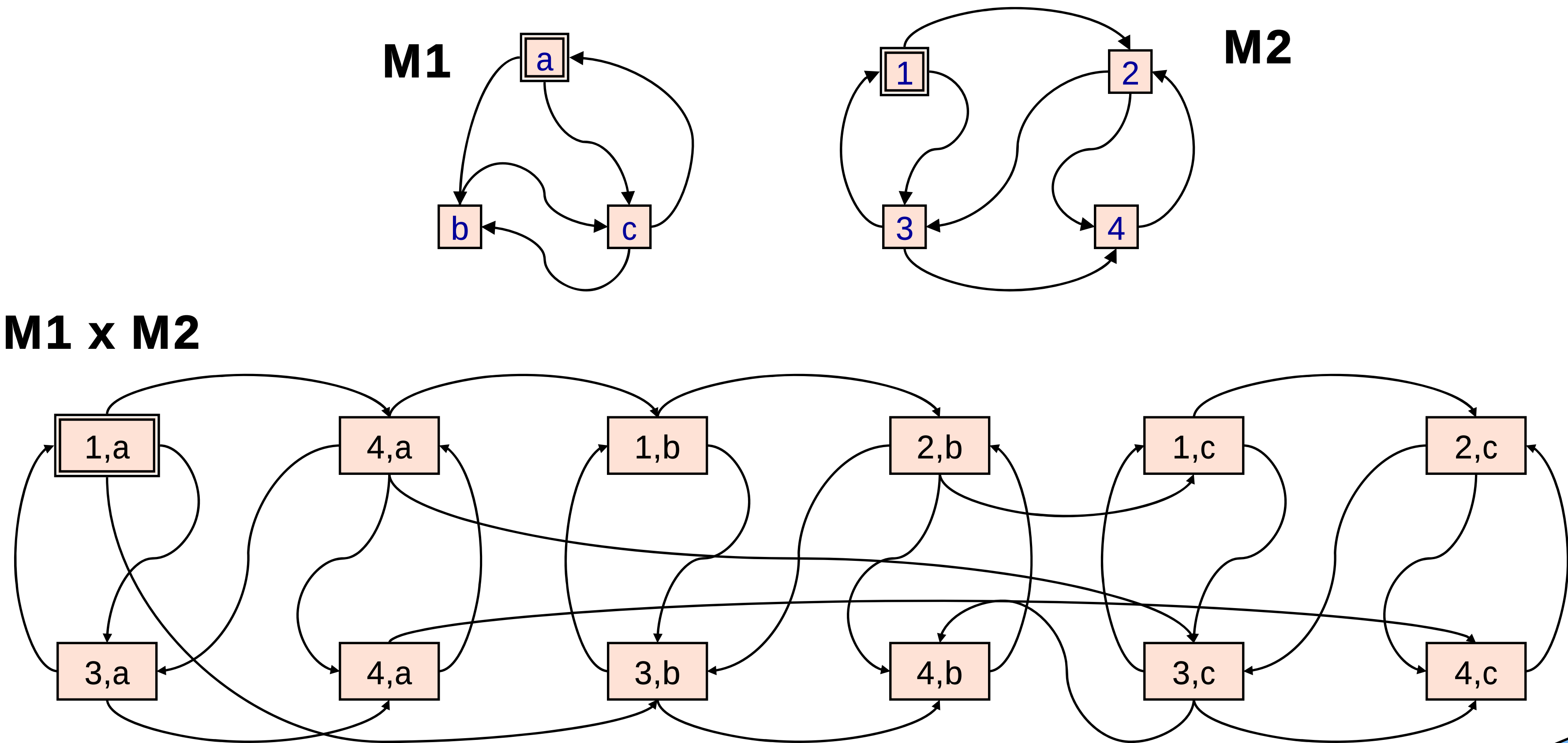
Simulation Output
Name
Value
Type
RingBell
TILT
GameOver
Go
Display
integer
GameNormal.RemainingMe
integer
All
Outputs
Locals
Traps
Variables
Watch
Simulation Control
Coin
On_off
Ready
Stop
MS
Inputs
Sensors
Return Signals
Commands
Tick
Reset
Keep Inputs
Current Session
1
Playback Session
Reset on Loading
Speed
Dump control
Waveform
Output file
Configuration file
Edit
Start
Stop
Coverage
Compact Coverage Files
eflexGameNormal.scg - ReflexGameNormal #0
Code
Coverage
Help
100
Module
Abbrev
Prior
ReflexGameNormal
On_off/
MachineON
signal RemainingMeasures:integer,
MEAN:integer
/Display(0),`call InitRNDGenerator()()`
Game Over
sustain GameOver
Coin/
GAME
asures(MEASURE_NUMBER)
<2>
<1>
PAUSE_LENGTH MS/
Display(?MEAN/MEASURE_NUMBER)
ningMeasures > 0]/

UPPAAL



Basic Research
in Computer Science

‘State Explosion’ Problem



All combinations = exponential in no. of compo

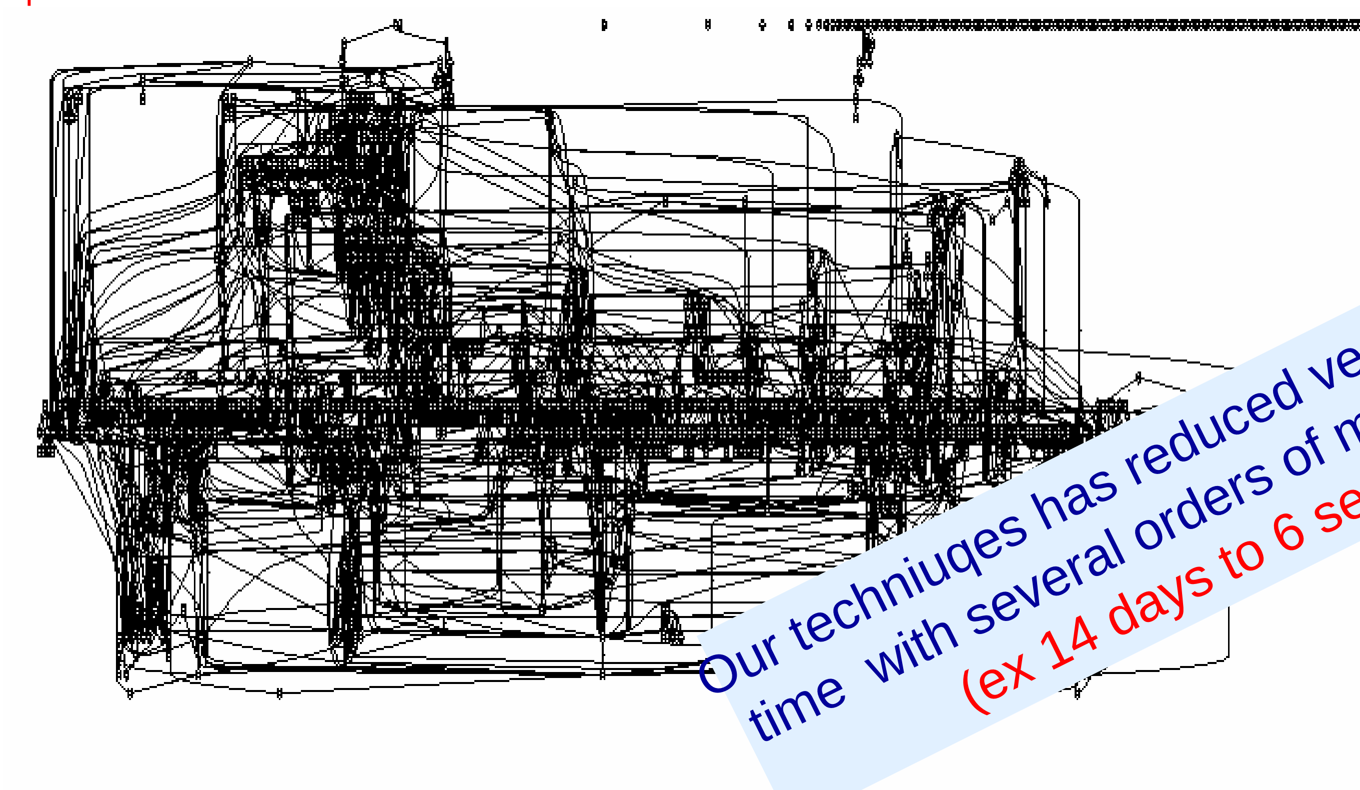
Provably theoretical intractable

Train Simulator

VVS
visualSTATE

1421 machines
11102 transitions
2981 inputs
2667 outputs
3204 local states
Declare state sp.: 10^476

BUGS ?

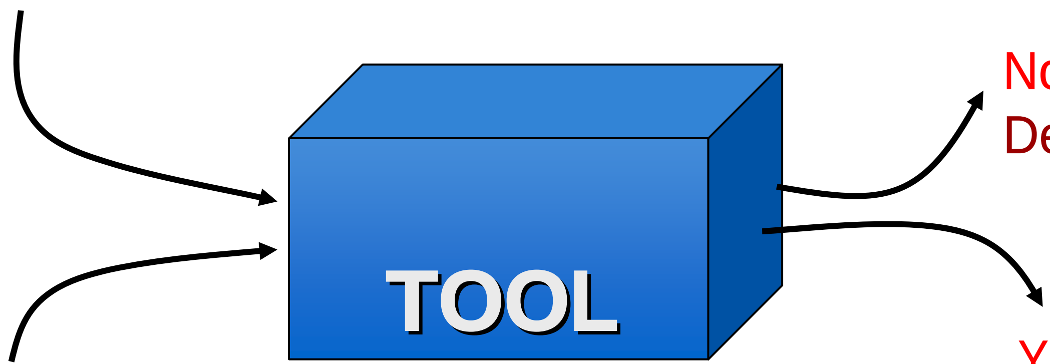


Our techniuqes has reduced verification time with several orders of magnitude
(ex 14 days to 6 sec)

Modelling and Analysis

Software Model **A**

Requirement **F**



No!
Debugging Information

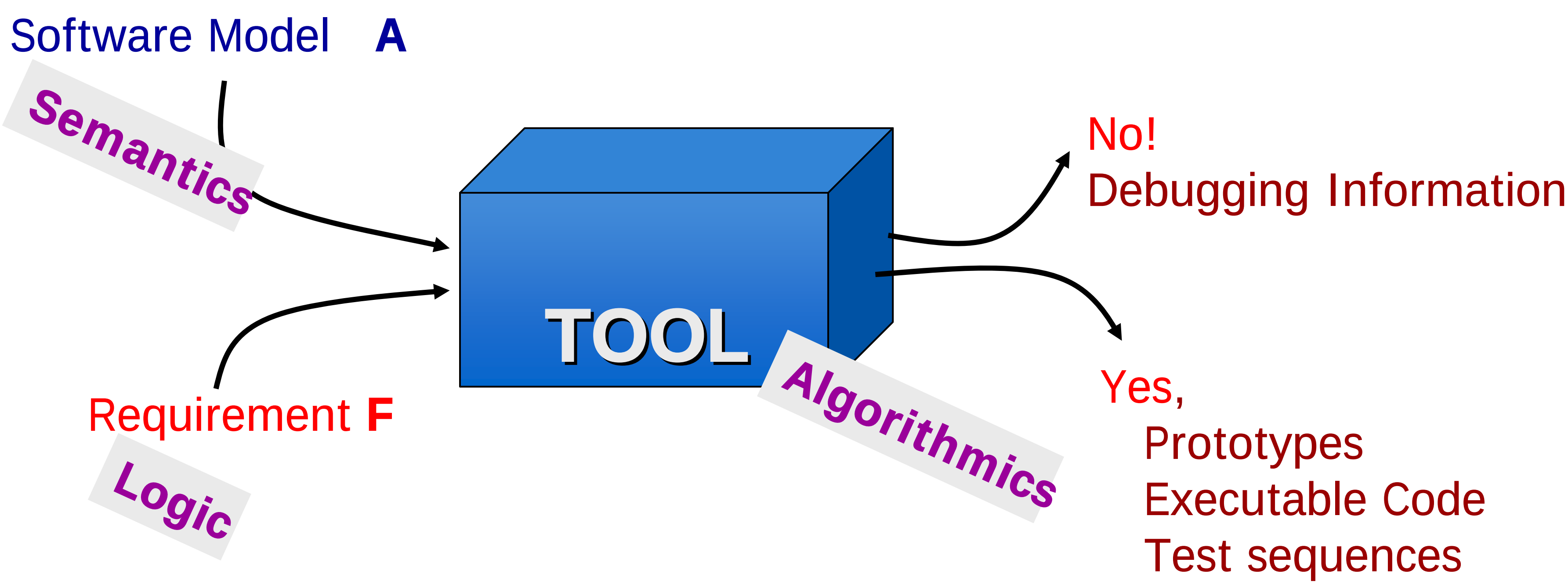
Yes,
- Prototypes
- Executable Code
- Test sequences

Tools: UPPAAL, visualSTATE,
ESTEREL, SPIN, Statemate, FormalCheck,
VeriSoft, Java Pathfinder,...

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Modelling and Analysis

BRICS



Tools: UPPAAL, visualSTATE,
ESTEREL, SPIN, Statemate, FormalCheck,
VeriSoft, Java Pathfinder,...

BRICS
Basic Research
in Computer Science

Most fundamentae
model in Computer Science:
Kleene og Moore

Finite State Machines

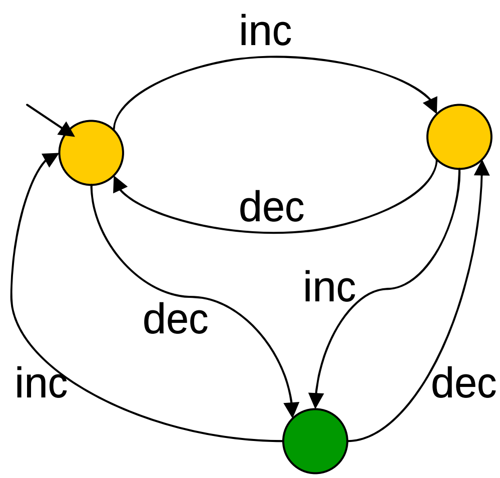
- Language versus behaviour
- Determinism versus non-determinism
- Composition and operations
- Variants of state machines
  Moore, Mealy, IO automater, UML ....

State Machines

Model of Computation

- Set of states
- A start state
- An input-alfabet
- A transition function, mapping input symbols and state to next state
- One ore more accept states.
- Computation starts from start state with a given input string (read from left to right)

Modulo 3 counter



inc inc dec inc inc dec inc ☹

inc inc dec inc dec inc dec inc ☺

input string

BRICS Basic Research in Computer Science

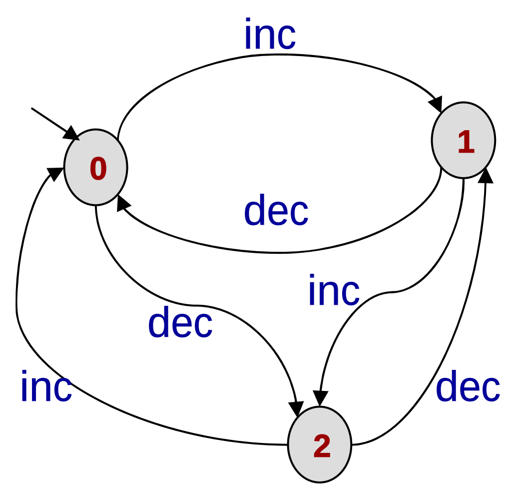
State Machines

Variants

Machines may have actions/output associated with state– Moore Machines.

inputstreng

inc inc dec inc inc dec inc



0 1 2 1 2 0 2 1

outputstreng

State Machines

Variants

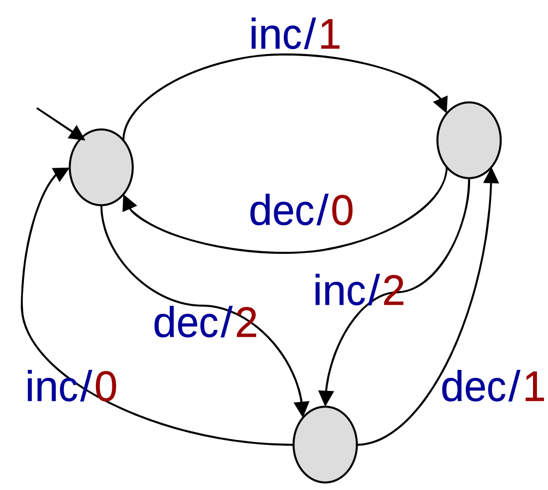
Machines may have actions/output associated with med transitions – Mealy Maskiner.

Transitions unconditional of af input (nul-transitions).

Several transitions for given for input and state
(non-determinisme).

inputstreng

inc inc dec inc inc dec inc



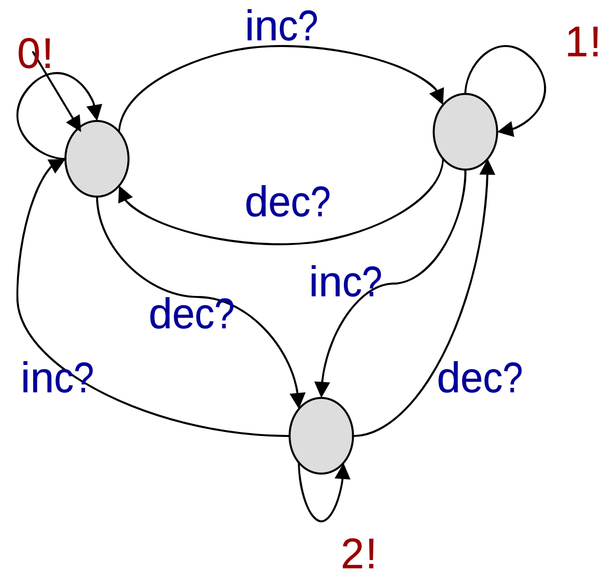
1 2 1 2 0 2 1

outputstreng

State Machines

Variants

Symbols of alphabet patitioned
in input- and output-actions
(IO-automata)



0! 0! 0! inc? inc? 2! 2! dec? 1!

interaction

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Bankbox Code

To open a bank box
the code most contain at least 2

To open a bank box
the code most end with

To open a bank box
the code most end with
or with

To open a bank box
the code most end with a palindrom
e.g:.

....

O
B
G

BRICS
Basic Research
in Computer Science

Fundamental Results

- Every FSM may be determinized accepting the same language (potential explosion in size).
- For each FSM there exist a language-equivalent *minimal* deterministic FSM.
- FSM's are closed under $\cap$ and $\cup$
- FSM's may be described as regular expressions (and vise versa)

Interacting State Machines

AALBORG UNIVERSITY DENMARK

BRICS
Basic Research
in Computer Science

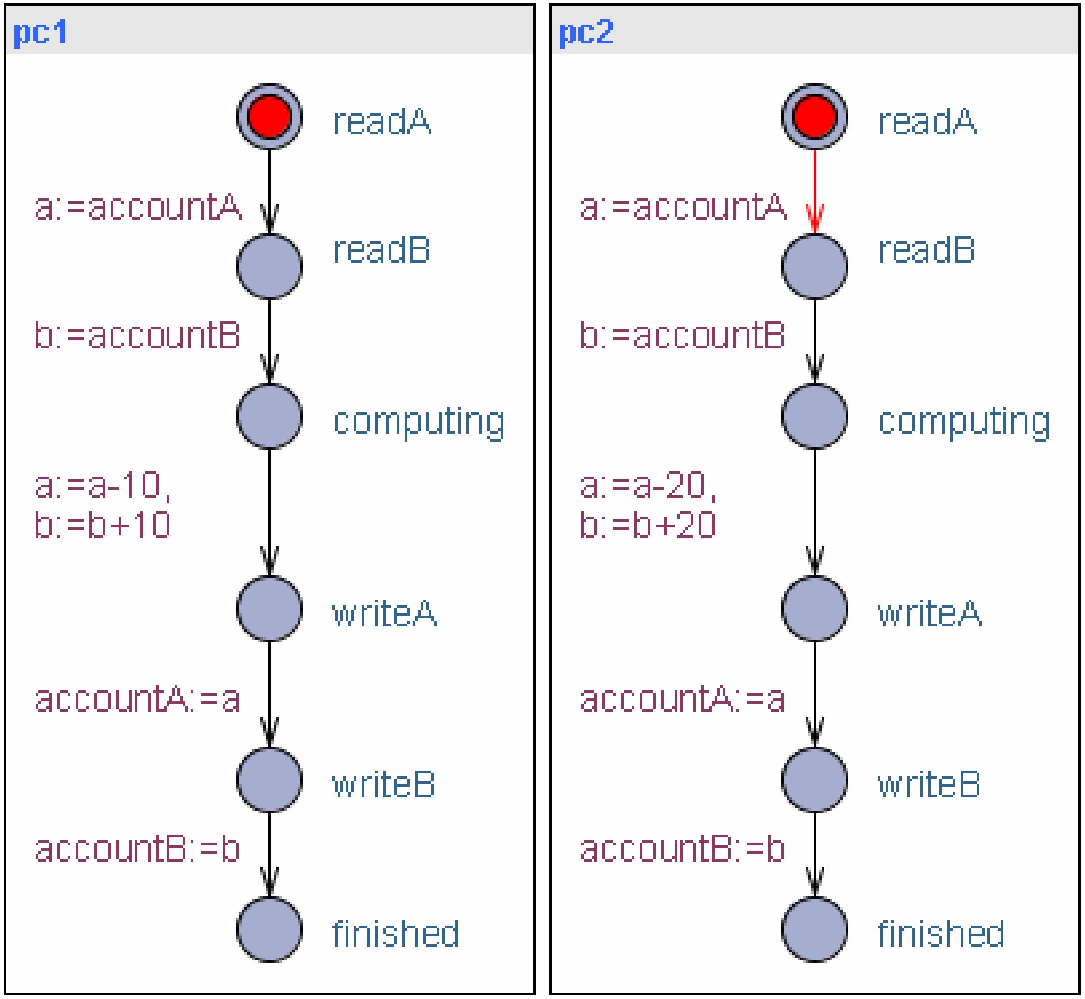
CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Home-Banking?

```
int accountA, accountB; //Shared global variables
//Two concurrent bank costumers

Thread costumer1 () {
  int a,b; //local tmp copy

  a=accountA;
  b=accountB;
  a=a-10;b=b+10;
  accountA=a;
  accountB=b;
}

Thread costumer2 () {
  int a,b;

  a=accountA;
  b=accountB;
  a=a-20; b=b+20;
  accountA=a;
  accountB=b;
}
```

- Are the accounts in balance after the transactions?

Home Banking



A[] (pc1.finished and pc2.finished) imply (accountA+accountB==200)?

Home Banking

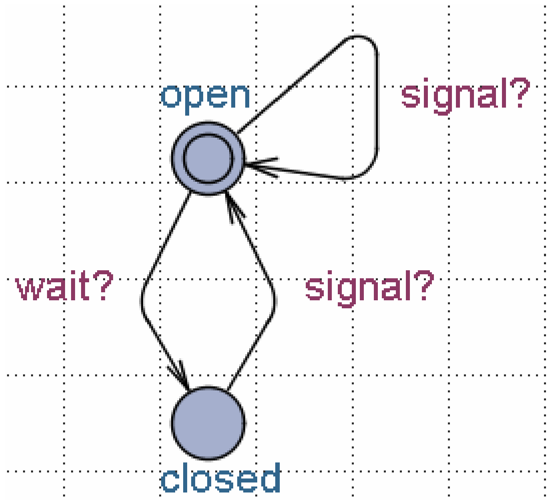
```
int accountA, accountB; //Shared global variables
Semaphore A,B;          //Protected by sem A,B
//Two concurrent bank costumers
```

```
Thread costumer1 () {
  int a,b; //local tmp copy

  wait(A);
  wait(B);
  a=accountA;
  b=accountB;
  a=a-10;b=b+10;
  accountA=a;
  accountB=b;
  signal(A);
  signal(B);
}
```

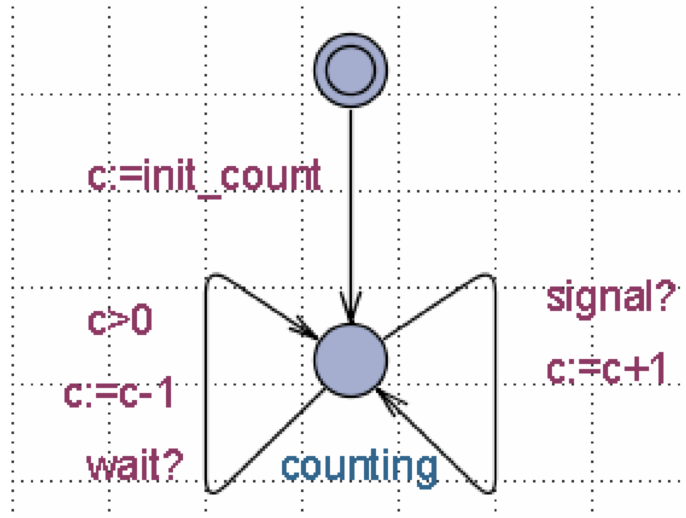
```
Thread costumer2 () {
  int a,b;

  wait(B);
  wait(A);
  a=accountA;
  b=accountB;
  a=a-20; b=b+20;
  accountA=a;
  accountB=b;
  signal(B);
  signal(A);
}
```

Semaphore FSM Model

Binary Semaphore



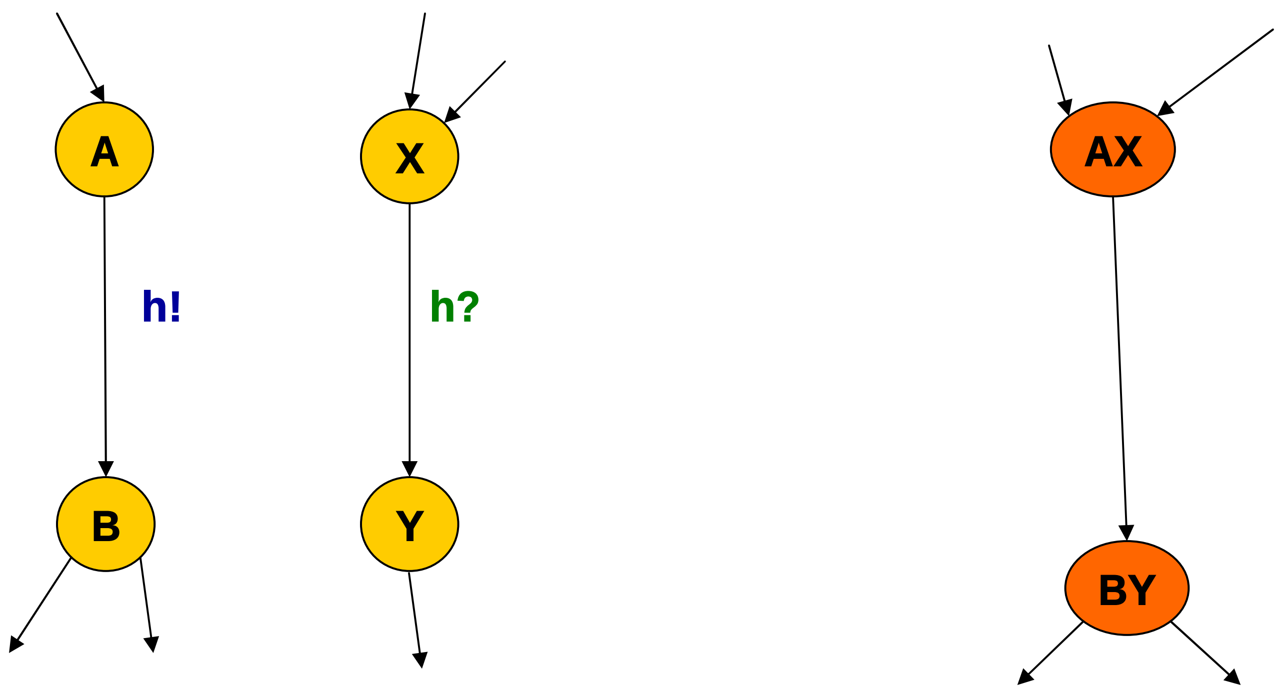
Counting Semaphore



BRICS
Basic Research
in Computer Science

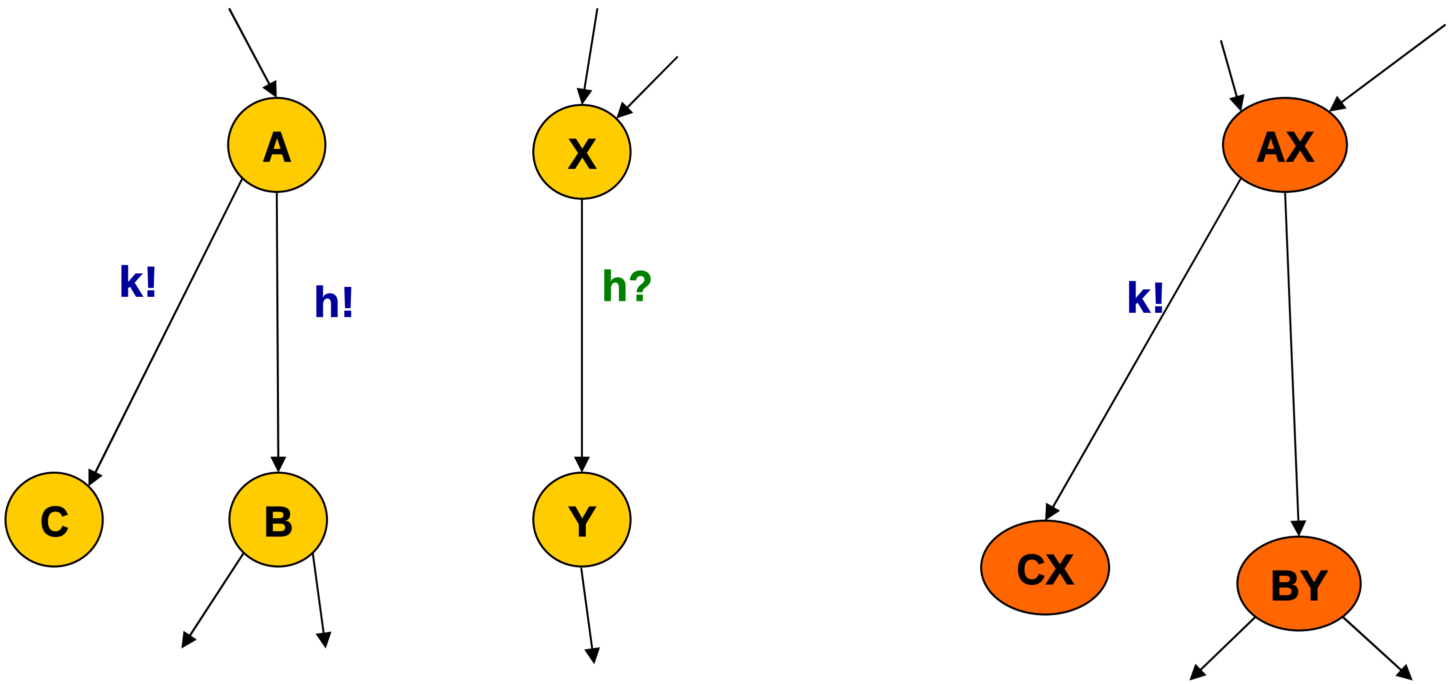
Composition

IO Automater (2-vejs synkronisering)

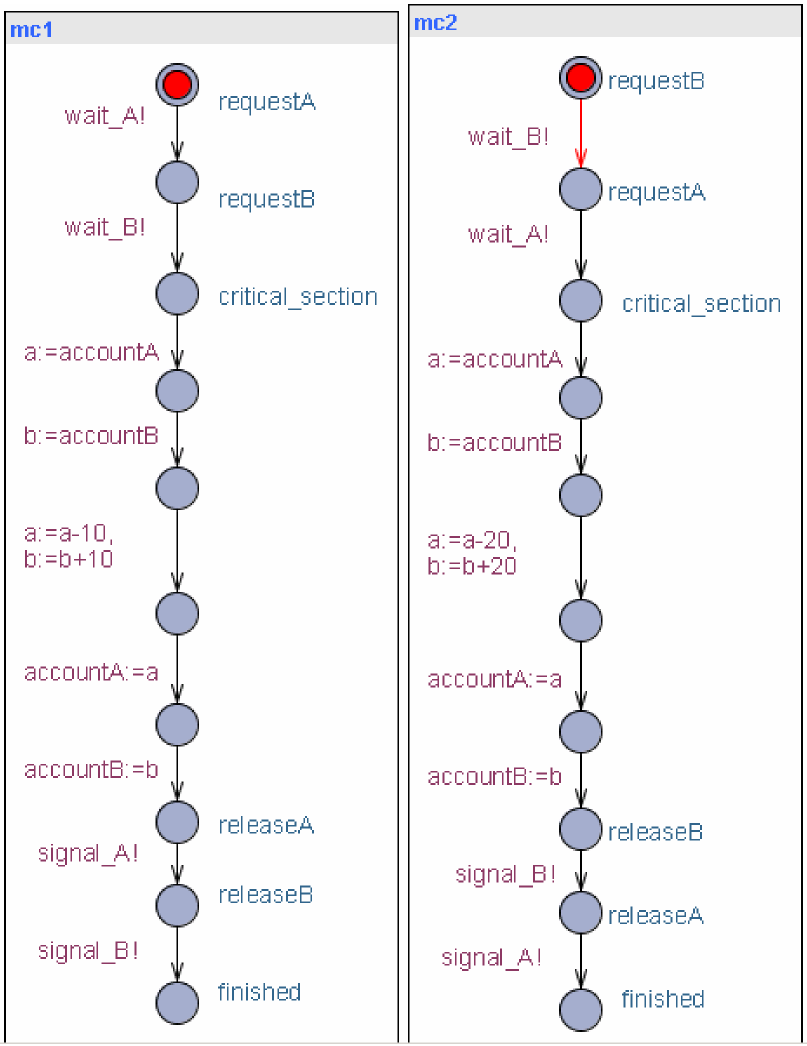


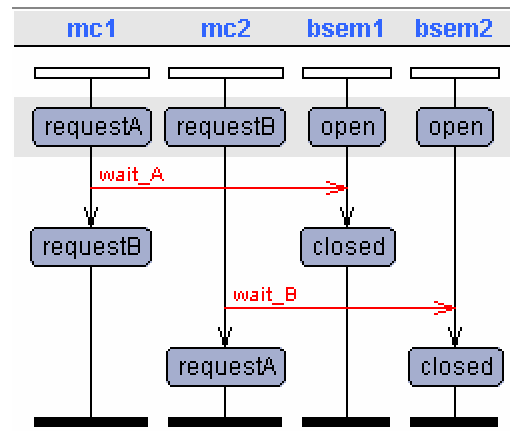
Composition

IO Automater



Semaphore Solution?





1. Consistency? (Balance)
2. Race conditions?
3. Deadlock?

1. A[] (mc1.finished and mc2.finished) imply (accountA+accountB==200) ✓
2. E<> mc1.critical_section and mc2.critical_section ✓
3. A[] not (mc1.finished and mc2.finished) imply not deadlock ÷