# JaMOS a MDL2e based Operating System for Jasmine

- Multitasking OS
- Finite State Machine OS
- Finite State Machine
- Regular expressions
- MDL2e (Motion Description Laguage 2 extended)
- JaMOS Architektur
- Optimisation of JaMOS

# JaMOS

## Multitasking, task switch

PRESENT

| Task *1* running | | Task ***2*** waiting |
|---|---|---|
| <u>CPU Registers</u><br><br>contain execution context of Task 1 | | <u>memory</u><br><br>stores execution context of Task 2 |

- A multitasking OS switches between tasks
  to give the appearance of many task
  running concurrently
- While switching,
  the OS saves the context of a stopped task,
  and loads the context of starting task in the registers

# JaMOS

## Multitasking, task switch

PAST

Task ***1*** running

**CPU Registers**

contain execution context of Task 1

Task ***2*** waiting

**memory**

stores execution context of Task 2

PRESENT

Task ***2*** running

**CPU Registers**

contain execution context of Task 1

Load execution context of Task 2 in the cpu registers

Store execution context of Task 1

Task ***1*** waiting

**memory**

stores execution context of Task 2

# JaMOS

## Multitasking, task switch

PAST

Task ***1*** running

Task ***2*** waiting

<u>CPU Registers</u>

contain execution context of Task 1

<u>memory</u>

stores execution context of Task 2

PRESENT

Task ***2*** running

Task ***1*** waiting

<u>CPU Registers</u>

contain execution context of Task 2

<u>memory</u>

stores execution context of Task 1

# JaMOS

## Finite State Machine OS

- A FSMOS (finite state machine operating system) is an OS that is described by a finite state machine.
- FSMOS has no concurrent tasks and no task-switch.
- All tasks are mapped to the states of the finite state machine.
- Advantages of a FSMOS
  - easy to analyze the running OS
  - all tasks share the whole memory space
  - easy to develop (with Description Languages)

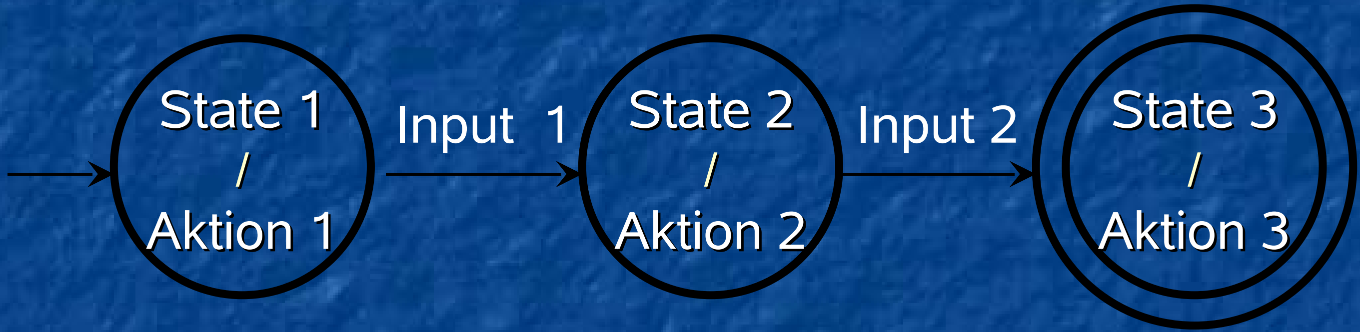
# JaMOS

## Finite State Machine

- A finite state machine (FSM) or finite automaton consist of
  - input alphabet
  - state transition function
    - (takes as arguments a state and an input symbol and returns a state and the corresponding action)
  - finite set of states
  - set of final states
  - initial state
  - Actions

# JaMOS

## Finite State Machine, example

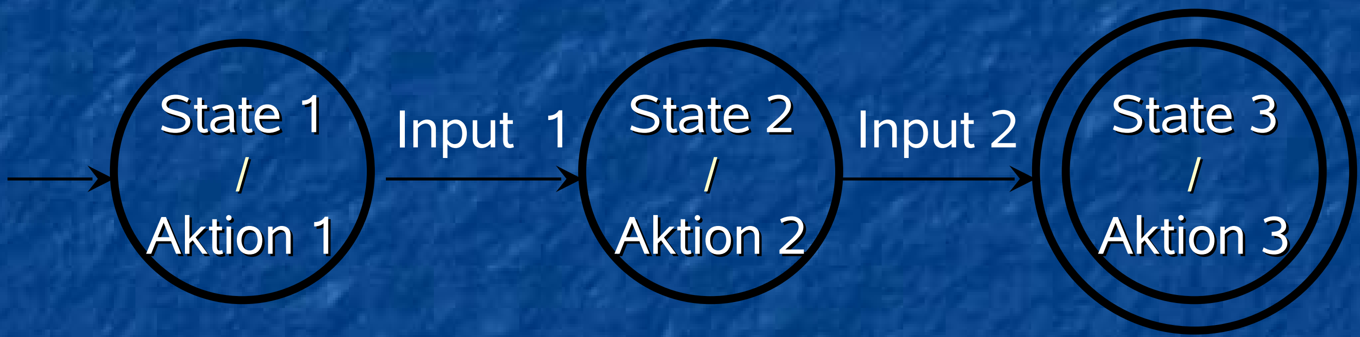



initial state – State 1
final state – State 3

set of states – State 1, State 2, State 3
input alphabet – Input 1, Input 2
actions – Action 1, Action 2, Action 3

# JaMOS

## Finite State Machine, accepted language




- inputs to FSM consist of strings over the input alphabet
- Because input alphabet is "***Input 1***, ***Input 2***", possible inputs could be :
  - "***Input 1***"
  - "***Input 2***"
  - "***Input 1***, ***Input 2***"
  - "***Input 2***, ***Input 1***"

This FSM accepts only the sequence "***Input 1***, ***Input 2***"
(or in other words,
FSM accepts the language "***Input 1***, ***Input 2***")

# JaMOS

## Comparison : Regular Expressions, FSM

- ***finite state machine*** is a good "***visual***" aid
  - but it is not very suitable as a specification

- ***regular expressions*** are a ***more compact*** way to define a language that can be accepted by an FSM

- FSM can be converted into a regular expression

- regular expressions can be converted into the FSM (but with exponential cost )

# JaMOS

## Regular Expressions, definition

Regular Expressions can be defined recursively :

*Basis* :

- The empty string  is a regular expression.
- For every character c in the input alphabet, c is a regular expression.

*Induction* :

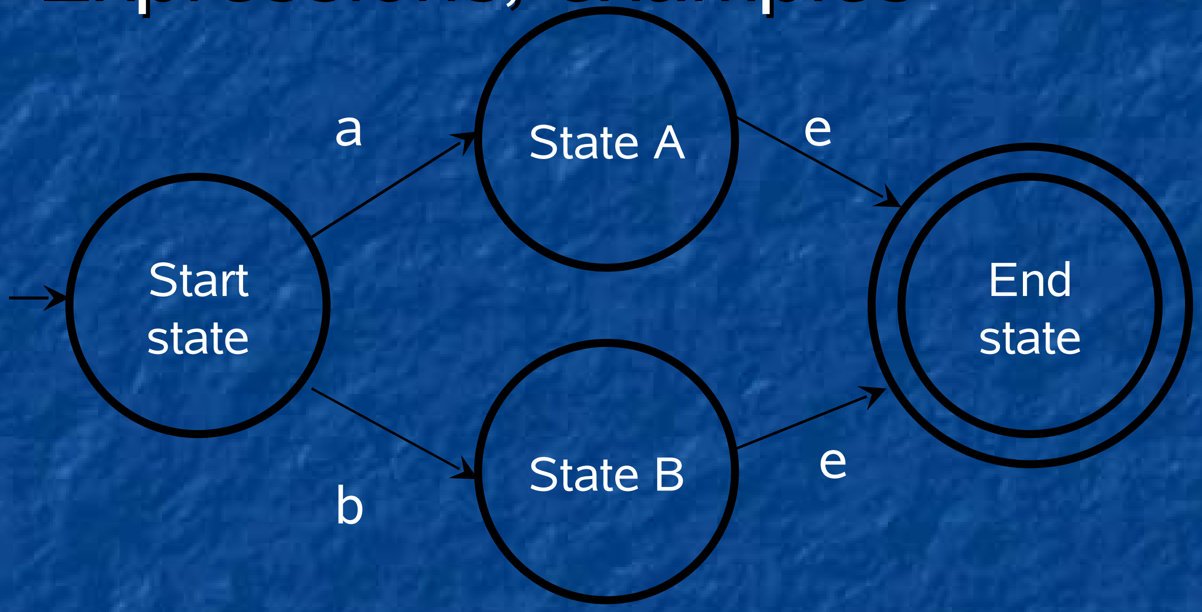
- If X and Y are regular expressions, then the ***Union***: “X + Y” is a regular  expression.
  - ( + means “OR” )
- If X and Y are regular expressions, then the ***Concatenation***: “XY” is a regular  expression.
- If X is a regular expression, then ***Closure*** : X* is a regular Expression
  - * means concatenation of 0 or more X
- if “X" is a regular Expression then a ***parenthesed***  x :  "(x)" is a regular Expression

# JaMOS

## Regular Expressions, examples

- ***Union***

a+b :



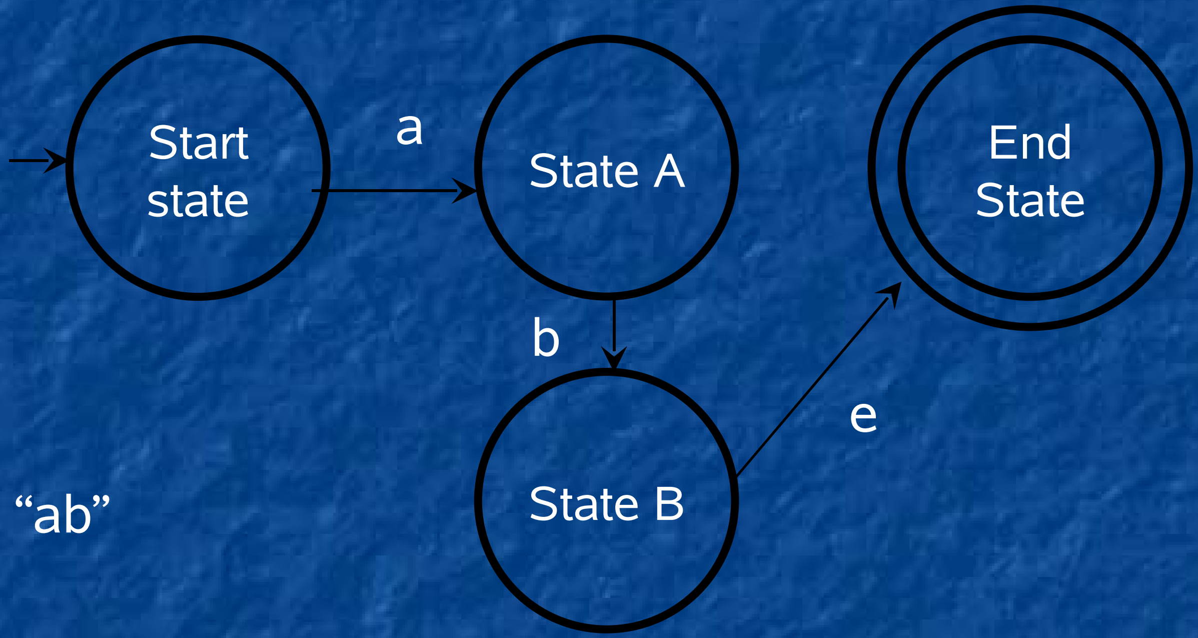

accepted language : “a” or “b”

a, b : regular expressions,
e : empty string

# JaMOS
## Regular Expressions, examples

- ***Concatenation***

ab :




accepted language :  “ab”

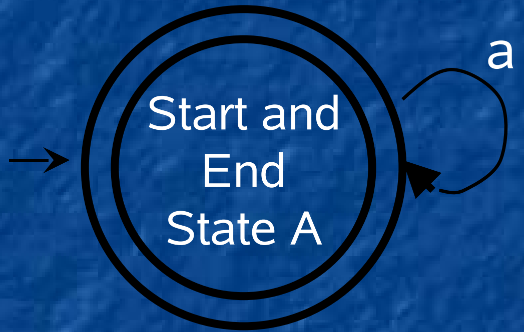
a, b : regular expressions,
e : empty string

# JaMOS

## Regular Expressions, examples

- ***Closure***

a* :




accepted language : “” or “a” or ”aa” or “aaa” or “aaaa” …

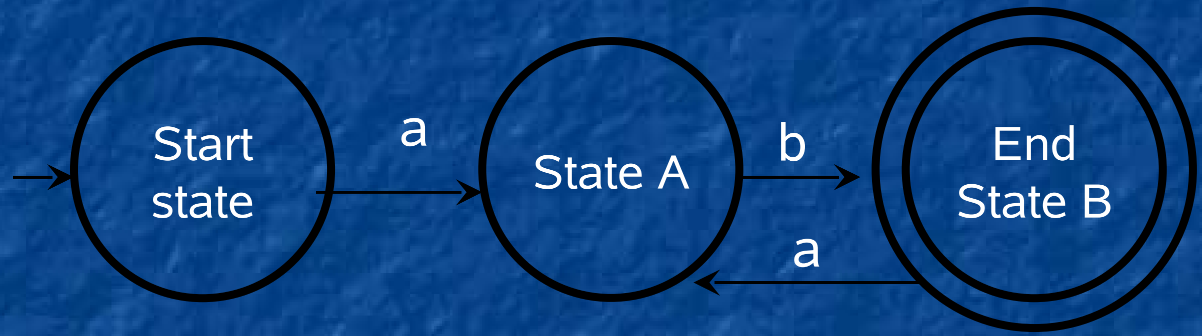
a : regular expression

# JaMOS

## Regular Expressions, examples

- ***Parantheses***

(ab)* :




accepted language : “ab” or “abab” or ” ababab” …

a, b : regular expressions,

# JaMOS

## MDL2e (Motion Description Laguage 2 extended)

- MDL2e describes the behavior of the robot with regular expressions

- MDL2e consists of following elements:
  - atom
  - plan
  - behavior
  - mult
  - union
  - runion
  - plan

# JaMOS

## MDL2e, element Atom

- Atoms are the simplest elements in MDL2e
- They are defined as triple “action, interrupt, duration”

- They correspond to a basic regular expression
  - Duration describes how long an atom should be executed
  - Interrupts are boolean expression
  - Action is a function that executes if
    - interrupt returns true AND
    - the time of execution is not up

# JaMOS
## MDL2e, element Atom, Interrupt

- An Interrupt can be
  - a basic interrupt or
  - a boolean expression  with basic interrupts as variables
- List of all MDL2e operators
  - AND (< basic interrupt >, < basic interrupt >)
  - OR   (< basic interrupt >, < basic interrupt >)
  - NOT (< basic interrupt >)
  - EQ   ( < value >, < value >)
  - GEQ ( < value >, < value >)
  - GT   ( < value >, < value >)

- < value >,  could be a variable ar a constant value
- < basic interrupt> is treated like boolean variable.
  - for example “IOBSTACLE” means an obstacle in front of the robot.

# JaMOS

## MDL2e, element Atom, example

```
< Atom name = “AMOVE”                    // action “AMOVE”
interrupt = “NOT(IOBSTACLE)”             // interrupt
arg0 = 10                                // argument 0 of “AMOVE” :
                                         // velocity  = 10
duration = 15 />                         // duration 15 time steps
```

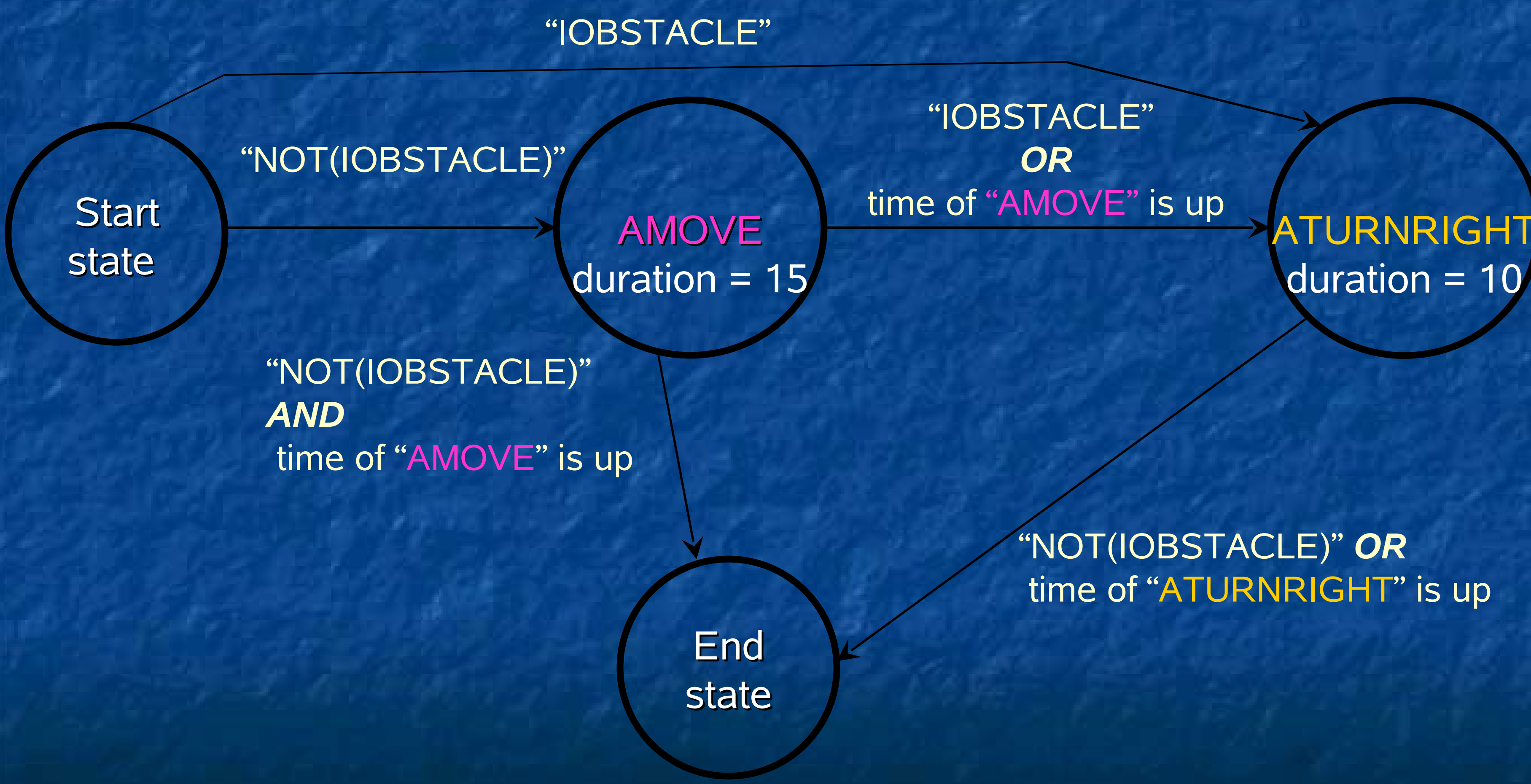
Robot moves forward for 15 time steps with velocity 10
if there is no obstacle

```
< Atom name = “ATURNRIGHT”               // action “AMOVE”
interrupt = “IOBSTACLE”       // interrupt
duration = 10 />              // duration 15 time steps
```

Robot turns right for 10 time steps if there is an obstacle

# JaMOS

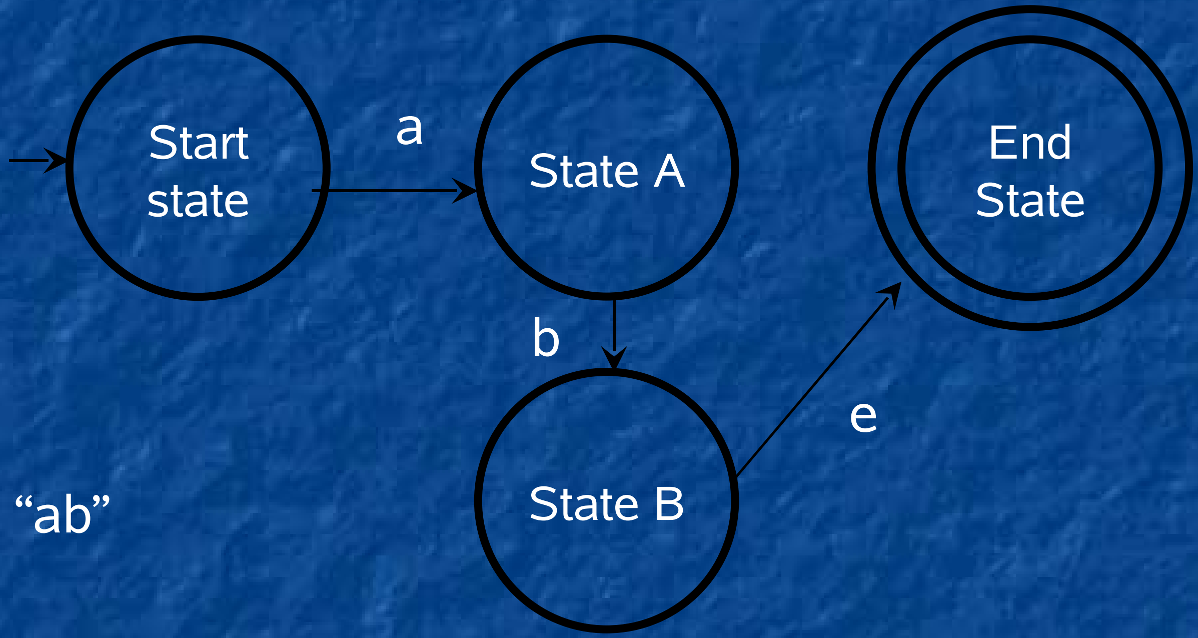
## MDL2e, element Atom, state diagram



# JaMOS
## Regular Expressions, examples

- ***Concatenation***

ab :




accepted language : “ab”

***NOTE*** :

MDL2e Atoms create much more complex FSM‘s as simple regular expressions

a, b : regular expressions,
e : empty string

# JaMOS

## MDL2e, element Behavior

- Behaviors are like parantheses  in regular expressions
- They group all MDL2e elements.
- Behaviors can construct high level behaviors,by building groups from other behaviors

Behaviors have as parameter a name, an interrupt and a duration,

Example :

<BEHAVIOR name = “BAVOID”, interrupt = “ITRUE”, duration = “infinite”>

< Atom name = “AMOVE” interrupt =NOT(IOBSTACLE)” arg0 = 10 duration = 15 />

< Atom name = “ATURNRIGHT” interrupt = “IOBSTACLE” duration = 10 />

</BEHAVIOR>

# JaMOS

## MDL2e, element Mult

- Mults will loop over the internal elements
- Mult works like closure in regular expressions
- Mults have as parameter a variable “multiplicity” that indicates the number of loops

Example :

<MULT multiplicity = 2> // execute ATOM 2 times

< ATOM name = “AMOVE” interrupt =NOT(IOBSTACLE)” arg0 = 10 duration = 15 />

</MULT>

# JaMOS

## MDL2e, element RUnion

- Runion “random union”
  - picks one random element from it internal elements
  - has an argument “probability”
    - Helps to calculate the probability distribution within a union

Example :

<RUNION probability = 2>

< ATOM name = “AMOVE” interrupt =“NOT(IOBSTACLE)” arg0 = 10 duration = 15 />

< ATOM name = “ATURNRIGHT” interrupt = “IOBSTACLE” duration = 10 />

</RUNION>

# JaMOS

## MDL2e, element Plan

- Plan is simply the first behavior, that contains all other MDL2e elements.

Behaviors have as parameter a name, an interrupt and a duration,

Example :

<PLAN name = "main_plan", interrupt = "ITRUE", duration = "infinite">

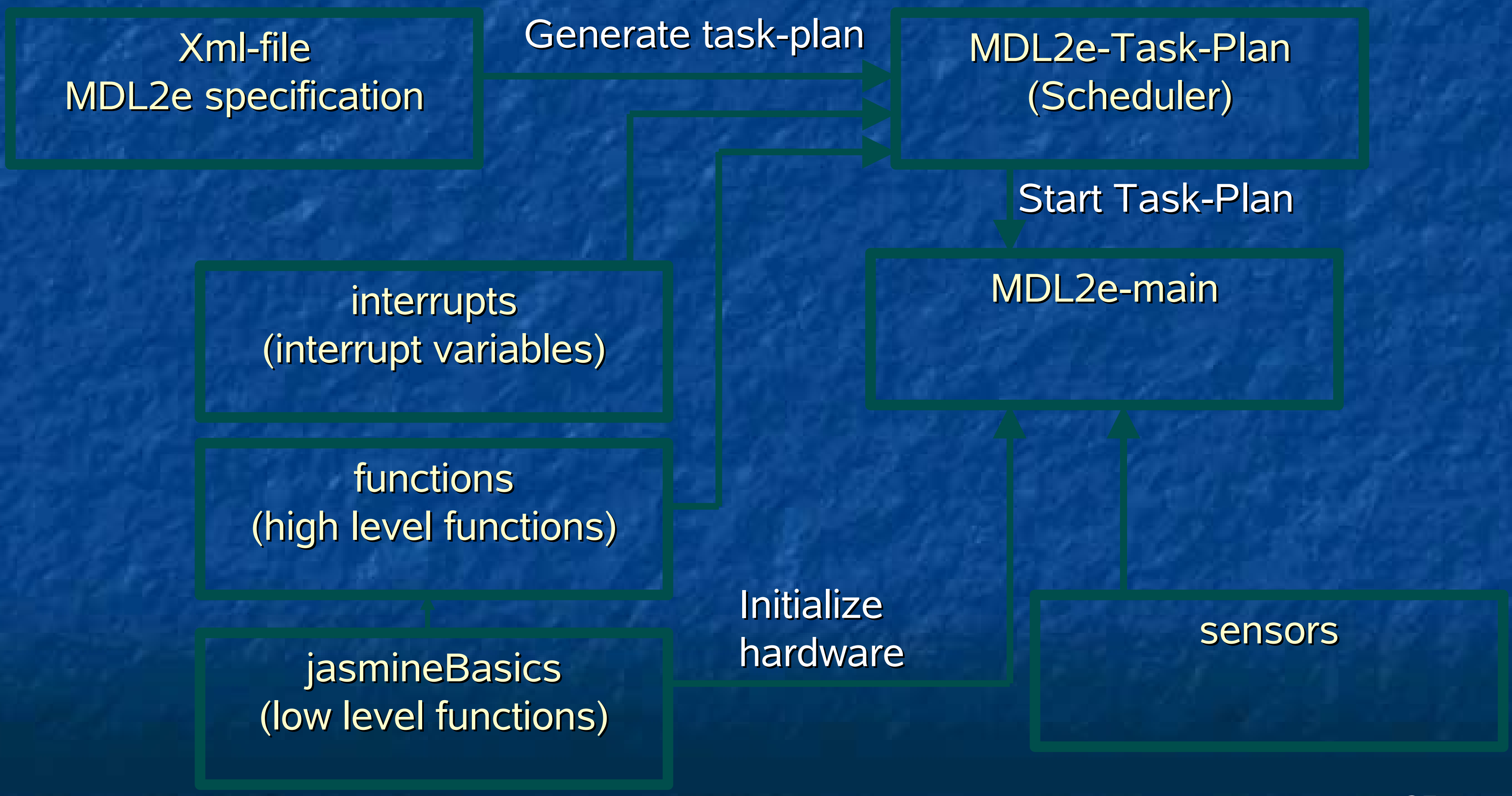
< Atom name = "AMOVE" interrupt =NOT(IOBSTACLE)" arg0 = 10 duration = 15 />

< Atom name = "ATURNRIGHT" interrupt = "IOBSTACLE" duration = 10 />

</PLAN>

# JaMOS

## Architecture



# JaMOS

## Architecture

Xml-file
MDL2e specification

<PLAN>
< BEHAVIOR >
<ATOM />
< /BEHAVIOR >
<MULT>
<ATOM />
</MULT>

</PLAN>

Generate task-plan

MDL2e-Task-Plan
(Scheduler)

Start Task-Plan

MDL2e-main

# JaMOS

## Architecture

Xml-file
MDL2e specification

encode
mdl2e byte code

MDL2e-Task-Plan
(Scheduler)

```
Plan () {
   // behavior1
    if (interrupt 1) {
       atom();
    }
   // behavior1
   if (interrupt 2) {
       atom();
   }
         // behavior1
         if (interrupt 3) {
       atom();
         }
    }
```

decode
mdl2e byte code

MDL2e-main

execute decoded
mdl2e byte code

# JaMOS

## Optimisation: eliminating of not possible mdl2e elements

Generate task-plan

Xml-file
MDL2e specification

<PLAN>
< BEHAVIOR interrupt = „A“>
<ATOM interrupt =
„NOT(A)“/>
< /BEHAVIOR >

<ATOM interrupt =„B“/>

</PLAN>

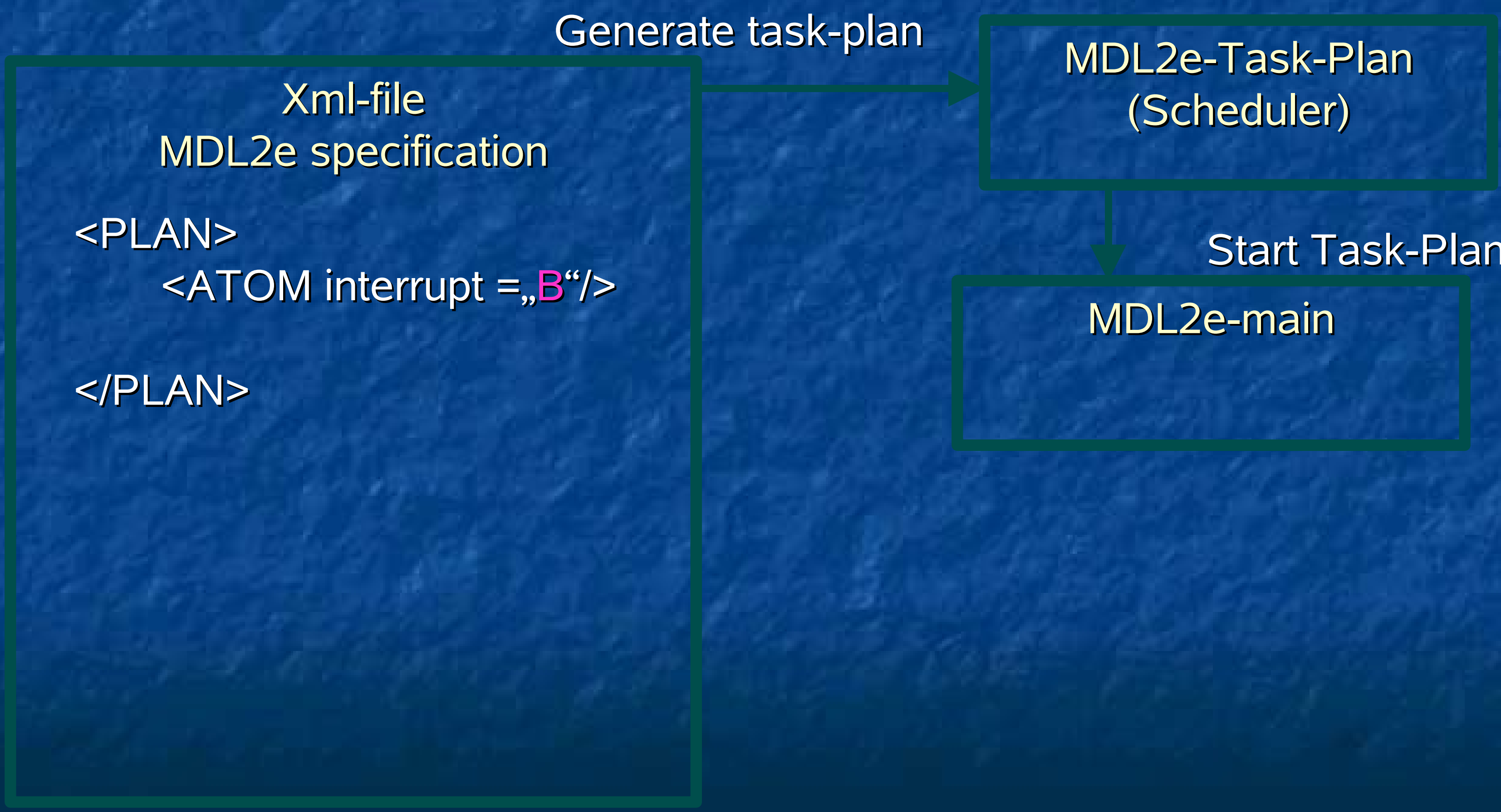
MDL2e-Task-Plan
(Scheduler)

Start Task-Plan

MDL2e-main

# JaMOS

## Optimisation: eliminating of not possible mdl2e elements



# JaMOS

## Optimisation: generate declaration and invoke behavior-functions

Xml-file
MDL2e specification

encode
mdl2e byte code

MDL2e-Task-Plan
(Scheduler)

```
Plan () {
  // behavior1
   if (interrupt 1) {
      atom();
   }
  // behavior1
  if (interrupt 2) {
      atom();
  }
      // behavior1
      if (interrupt 3) {
     atom();
      }
   }
```

decode
mdl2e byte code

MDL2e-main

execute decoded
mdl2e byte code

# JaMOS

## Optimisation: generate declaration and invoke behavior-functions

Xml-file
MDL2e specification

encode
mdl2e byte code

MDL2e-Task-Plan (Scheduler)

//declaration
<u>*behavior1*</u> (interrupt i) {
}
***Plan*** () {
Behavior1( interrupt 1);
Behavior1( interrupt 2);
Behavior1( interrupt 3);
}

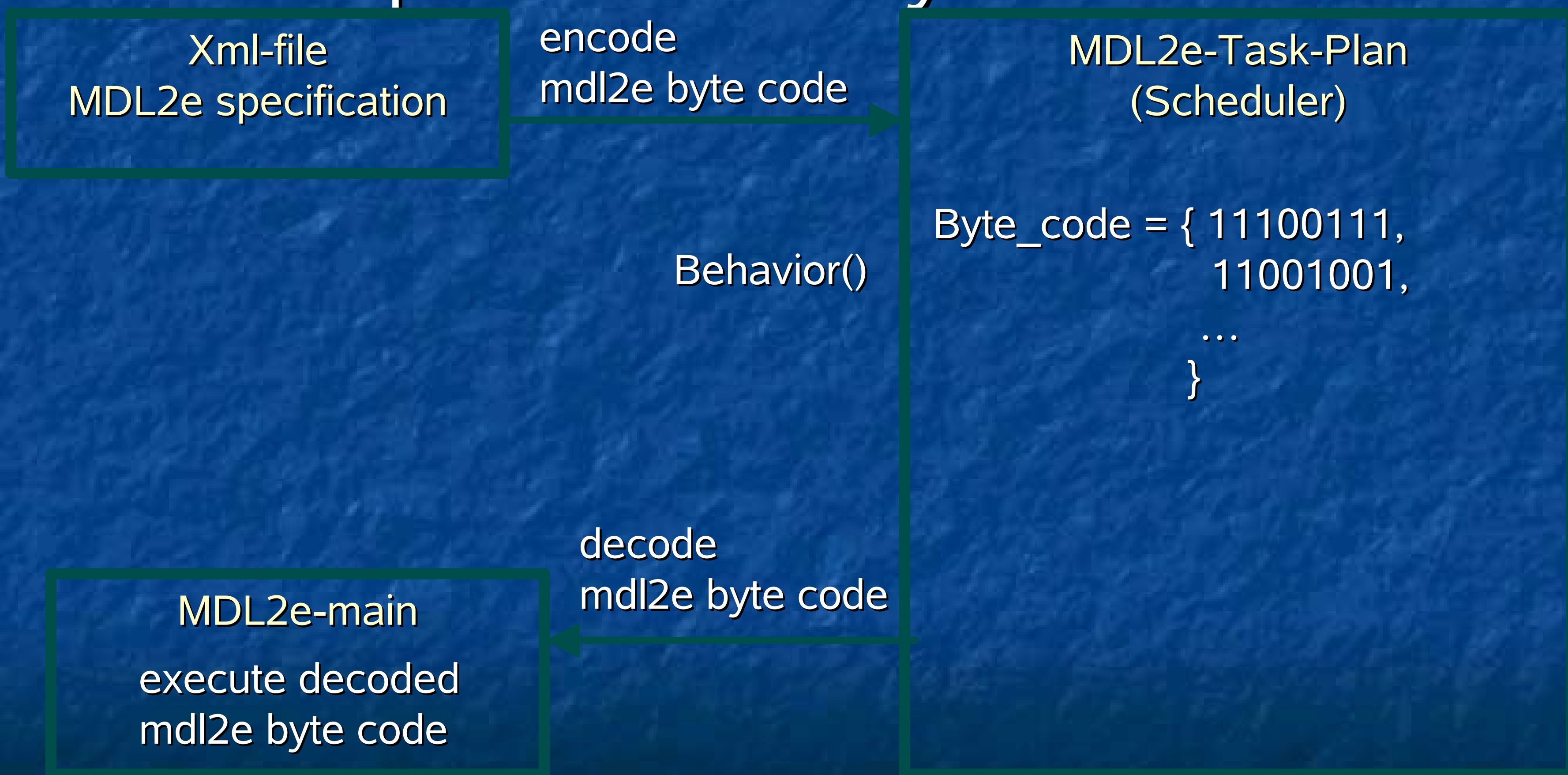
decode
mdl2e byte code

MDL2e-main

execute decoded
mdl2e byte code

# JaMOS

## Optimization encode the MDL2e specification in byte code



# JaMOS

## Optimization encode the MDL2e specification in byte code

Xml-file
MDL2e specification

< BEHAVIOR interrupt = „A“>
<ATOM interrupt =„B“/>
< /BEHAVIOR >

Xml-file
MDL2e specification

behavior
1
Atom
0

Interrupt A
1

Interrupt B
0

Byte code = {11000000}

# JaMOS a MDL2e based Operating System for Jasmine summary

- Multitasking OS
- Finite State Machine OS
- Finite State Machine
- Regular expressions
- MDL2e (Motion Description Laguage 2 extended)
- JaMOS Architektur
- Optimisation of JaMOS