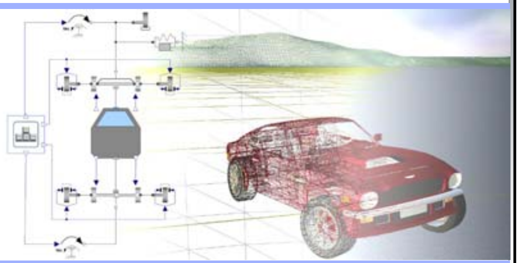
Virtual Physics
Equation-Based Modeling

TUM, October 07, 2014

Equation-based modeling: first steps

```
equation
  sx0 = cos(frame_a.phi)*sx_norm + …
  sy0 = -sin(frame_a.phi)*sx_norm + …
  vy = der(frame_a.y);
  w_roll = der(flange_a.phi);
  v_long = vx*sx0 + vy*sy0;
  v_lat = -vx*sy0 + vy*sx0;
  v_slip_lat = v_lat - 0;
  v_slip_long = v_long - R*w_roll;

  v_slip = sqrt(v_slip_long^2 + …
  -f_long*R = flange_a.tau;
  frame_a.t = 0;
  f = N*. S_Func(vAdhesion,vSlide,…
  f_long =f*v_slip_long/v_slip;
  f_lat  =f*v_slip_lat/v_slip;
  f_long = frame_a.fx*sx0 + …
  f_lat = -frame_a.fx*sy0 + …
```



Dr. Dirk Zimmer

German Aerospace Center (DLR), Robotics and Mechatronics Centre

Modeling Example

- Let us start with a simple modeling example:

  Let us brew beer! (or ferment wine.. for the non-ba(rb/v)arians)

- In this example, we are going to model the fermentation of sugar into alcohol and the corresponding growth and decay of yeast.

- In the process of fermentation each molecule of sugar is transformed into a molecule of alcohol (plus CO_2)

$$C_6H_{12}O_6 \rightarrow 2\ C_2H_5OH + 2\ CO_2$$

Variables and Parameters

TUM + DLR Robotics and Mechatronics Centre

- These are our model variables:
  - Population of yeast: *p*
  - Birth-Rate: *b*
  - Death-Rate: *d*

  - Concentration of sugar: *s*
  - Concentration of alcohol: *a*
  - Consumption of sugar: *f*

  - Current Temperature: *T*

- These are our model parameters:
  - Volume of vessel: V = 1
  - Initial concentration of sugar: s0 = 0.2
  - Initial population of yeast: p0 = 0.001

  - Feeding-Rate Coefficient: C_f
  - Reproductivity: R
  - Sensitivity to poison: S
  - Reference Temperature: T_{ref}

Algebraic Equations

TUM + DLR Robotics and Mechatronics Centre

Let us start with the algebraic equations:

- The consumption of sugar (*f*) is proportional to concentration of sugar (*s*) multiplied by the population of yeast (*p*). The proportionality is determined by the feeding-rate (C_f) and the temperature (*T*)

$$f = s \cdot p \cdot C_f \cdot (T/T_{ref})$$

- Since each molecule of alcohol was transformed from one molecule of sugar at time 0 (s0), the current concentration of alcohol (a) is:

$$a = s0 - s$$

Algebraic Equations

Robotics and Mechatronics Centre

Let us continue with the algebraic equations:

- The Birth-Rate is proportional to concentration of sugar (s). The proportionality is determined by the reproduction (R):

$$b = R \cdot s$$

- The Death-Rate is dependent on the level of poisonous alcohol (a) and the sensitive (S) of the yeast.

$$d = S \cdot a$$

Algebraic Equations

Robotics and Mechatronics Centre

The algebraic equations are:

$$f = s \cdot p \cdot C_f \cdot (T/T_{ref})$$

$$a = s0 - s$$

$$b = R \cdot s$$

$$d = S \cdot a$$

T is determined from outside (input-variable)

Differential Equations

Robotics and Mechatronics Centre

The differential equations describe the change over time:

- The change in population (dp/dt) equals the birth-rate (b) minus the death rate (d) and is proportional to the current population (p):

$$dp/dt = p \cdot (b-d)$$

- The change in concentration of sugar (ds/dt) multiplied by the Volume (V) equals the negative consumption rate (f) of sugar :

$$V \cdot ds/dt = -f$$

or

$$ds/dt = -f/ V$$

Differential Equations

Robotics and Mechatronics Centre

The differential equations are:

$$dp/dt = p \cdot (b-d)$$

$$ds/dt = -f/V$$

Differential Equations

Let us plug in the algebraic equations:

$dp/dt = p \cdot (b-d)$

$ds/dt = -f/V$

Differential Equations

Let us plug in the algebraic equations:

$dp/dt = p \cdot (b-d)$

$dp/dt = p \cdot (R \cdot s - S \cdot a)$

$ds/dt = -f/V$

Differential Equations

Let us plug in the algebraic equations:

$dp/dt = p \cdot (b-d)$

$dp/dt = p \cdot (R \cdot s - S \cdot a)$

$dp/dt = p \cdot (R \cdot s - S \cdot (s0-s))$

$ds/dt = -f/V$

Differential Equations

Let us plug in the algebraic equations:

$dp/dt = p \cdot (b-d)$

$dp/dt = p \cdot (R \cdot s - S \cdot a)$

$dp/dt = p \cdot (R \cdot s - S \cdot (s0-s))$

$\underline{dp/dt = p \cdot ((R+S) \cdot s - S \cdot s0)}$

$ds/dt = -f/V$

Differential Equations

Robotics and Mechatronics Centre

Let us plug in the algebraic equations:

$$dp/dt = p \cdot (b\text{-}d)$$
$$dp/dt = p \cdot (R \cdot s - S \cdot a)$$
$$dp/dt = p \cdot (R \cdot s - S \cdot (s0 - s))$$
$$\underline{dp/dt = p \cdot ((R+S) \cdot s - S \cdot s0)}$$

$$ds/dt = -f/V$$
$$\underline{ds/dt = -s \cdot p \cdot C_f \cdot (T/T_{ref}) \cdot 1/V}$$

Differential Equations

Robotics and Mechatronics Centre

Let us plug in the algebraic equations:

$$\underline{dp/dt = p \cdot ((R+S) \cdot s - S \cdot s0)}$$

$$\underline{ds/dt = -s \cdot p \cdot C_f \cdot (T/T_{ref}) \cdot 1/V}$$

Time Discretization

Robotics and Mechatronics Centre

- Let us discretize the advance of time by the quantum h:
- Given x_t, we can compute x_{t+h} *by using the Taylor-series expansion:*

$$x_{t+h} = x_t + (dx/dt)_t \cdot h + (dx/dt^2)_t \cdot (h^2/2) + (dx/dt^3)_t \cdot (h^3/6) + \ldots$$

- Let us drop all higher derivatives. We get:

$$x_{t+h} = x_t + (dx/dt)_t \cdot h$$

- This discretization scheme is called: **Forward Euler**

Time Discretization

Robotics and Mechatronics Centre

Let us apply Forward Euler to our differential equations:

$$p_{t+h} = p_t + (dp/dt)_t \cdot h$$
with
$$(dp/dt)_t = p_t \cdot ((R+S) \cdot s_t - S \cdot s0)$$

$$s_{t+h} = s_t + (ds/dt)_t \cdot h$$
with
$$(ds/dt)_t = -s_t \cdot p_t \cdot C_f \cdot (T_t/T_{ref}) \cdot 1/V$$

Simulation

TUM + DLR Robotics and Mechatronics Centre

- These four explicit equations are used to perform a simulation:

$$p_{t+h} = p_t + (dp/dt)_t \cdot h \text{ with } (dp/dt)_t = p_t \cdot ((R+S) \cdot s_t - S \cdot s0)$$

$$s_{t+h} = s_t + (ds/dt)_t \cdot h \text{ with } (ds/dt)_t = -s_t \cdot p_t \cdot C_f \cdot (T_t/T_{ref}) \cdot 1/V$$

- We can simply punch them into a Phyton3 script:

```
while time < 10:
    dp_dt = p*((R+S)*s - S*s0)
    ds_dt = -s*p*C_f*(T/T_ref)*1/V
    p += h*dp_dt
    s += h*ds_dt
    a = s0-s
    time += h
    print(time,"\t",p,"\t",s,"\t",a)
```

- Here, there are computed within a loop. Each iteration represents one time-step: an advance of *h* in time.

Simulation Code

TUM + DLR Robotics and Mechatronics Centre

This is the complete Phyton3-Script:

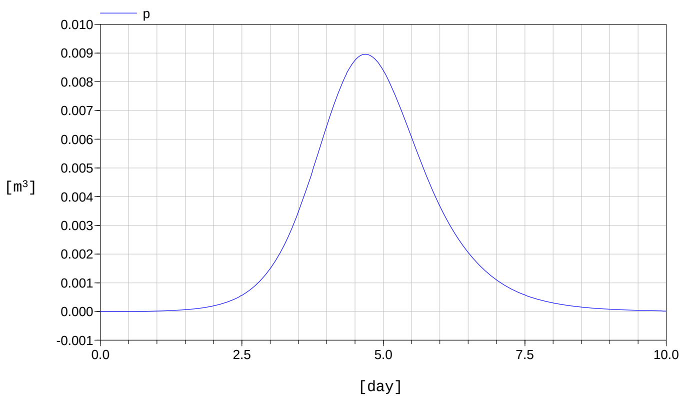
```
#!/usr/bin/env python3
#Setting the parameters
V = 1   #volume of fermentation vessel
s0 = 0.2 #inital concentration of sugar
p0 = 1e-6 #initial population of yeast [m3]
C_f = 50 #feeding Coefficient [1/day]
R = 10 #reproductivity [1/day]
S = 15 #sensitivity w.r.t. alcohol [1/day]
T_ref = 300 #reference temperature [K]
h = 0.01 #time-step of forward Euler integration

#Setting the initial values
p = p0
s = s0
a = s0 - s;
time = 0
```

```
#Setting the input-value
T = 310

#perform time-integration
while time < 10:
    dp_dt = p*((R+S)*s - S*s0)
    ds_dt = -s*p*C_f*(T/T_ref)*1/V
    p += h*dp_dt
    s += h*ds_dt
    a = s0-s
    time += h
    print(time,"\t",p,"\t",s,"\t",a)
```


Simulation Results

TUM + DLR Robotics and Mechatronics Centre

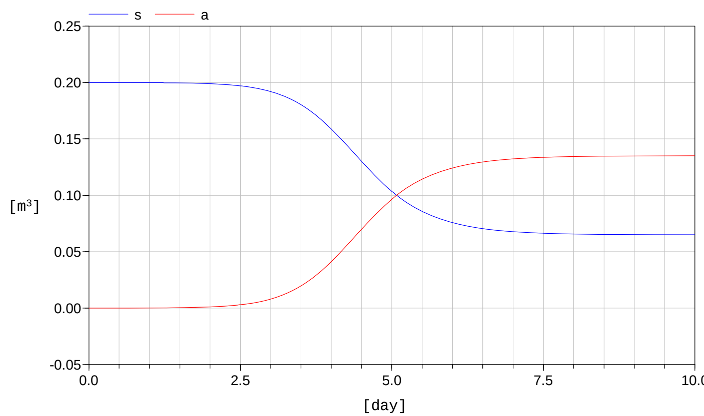
- And this is the result for the yeast population:



Simulation Results

TUM + DLR Robotics and Mechatronics Centre

- Concentration of sugar and alcohol:



Simulation Results

Interpretation of the simulation results:

- The population of yeast first grows exponentially. There seems to be an endless supply of sugar available.
- Then the population has reached a critical level and the concentration of sugar and alcohol are rapidly changing.
- Then, there is a sudden die-off due to the combination of starvation and self-poisoning.

State-Space Form

Let us look at the computational structure of our model. We can classify our variables into vectors of…

- Input Variables: **u** = (T)
- State Variables: **x** = (p, s)
- State Derivatives: d**x**/dt = (dp/dt, ds/dt)
- Output Variables: **y** = (a)

- The system was then transformed into two functions:

$$d\mathbf{x}/dt = f(\mathbf{x},\mathbf{u},t)$$
$$\mathbf{y} = g(\mathbf{x},\mathbf{u},t)$$

- This specific form is called: **state-space form**

State-Space Form

- We form a row vector out of **x**,**u**, and t: (p,s,T,t)
- We form a column vector out of d**x**/dt and **y**: (dp/dx, ds/dx, a)
- Now we can represent the dependences of our computational structure by a Boolean incidence matrix.

		x		(**u**, t)			
		p	s	T	t		
d**x**/dt	dp/dt	**X**	**X**			$dp/dt = p\cdot ((R+S)\cdot s - S\cdot s0)$	$f(\dots)$
	ds/dt	**X**	**X**	**X**		$ds/dt = -s\cdot p\cdot C_f\cdot (T/T_{ref})\cdot 1/V$	
y	a		**X**			$a = s0 - s$	$g(\dots)$

State-Space Form

- The incidence matrix can be decomposed into four blocks: **A**, **B**, **C**, **D**.
- If $g(\dots)$ and $f(\dots)$ represent linear functions (not the case here!), the system can indeed be expressed by real-valued matrices:

$$d\mathbf{x}/dt = \mathbf{Ax} + \mathbf{B}(\mathbf{u}, t)$$
$$\mathbf{y} = \mathbf{Cx} + \mathbf{D}(\mathbf{u}, t)$$

	p	s	T	t
dp/dt	X	X		
ds/dt	X	X	X	
a		X		

	p, s	T, t
dp/dt, ds/dt	**A**	**B**
a	**C**	**D**

Summary

Let us summarize the development process of our simulation:

- First, we had to analyze our model and select the variables of interest.
- Then, we formulated a set of *differential-algebraic equations* (DAEs).
- Next, we had to transform this set of expressions into a computable/solvable form (*state-space form*).
- Finally, a time-discretization scheme was applied and a numerical integration could be performed (*numerical ODE-solver*).

Deficiencies

Even for this small and rather trivial example, this development process was rather laborious.

- Larger models cause much more work.
- Also there are more complicated models that are difficult to transform into state-space form.
- If we change the model, the complete process has to be redone.
- Programming a simulation manually turns out to be very inconvenient and is also very error-prone.
- For these reasons, a number of computer languages have been developed that aim to automate this process.
- Let us take a look back in history…

MIMIC (History)

- The language MIMIC was developed mainly for the Control Data super-computers in 1964.
- The listing presents the MIMIC code for the simulation of a swinging pendulum.
- Successors of these language were CSMP and ACSL. They prevailed up to the 80s.

```
   CON(G)         Declaration of constants
   PAR(1X0,X0)    Declaration of parameters
DT 0.05           Definition of time step

1X INT(-G*Z,1X0)  Integration
 X INT(1X,X0)
 Y 1.-COS(X)      Equation for y position
 Z SIN(X)         Equation for z position

   FIN(T,4.9)     Command for integration

   PLO(T,X,Y,Z)   Commands for plotting
   ZER(0.,-5,0.,-1)
   SCA(5.,5.,2.,1.)

   END  End of program
```


MIMIC

CDC 6600

- 40 MHz, roughly 1MFLOPS, 64K 60-bit words of memory
- Roughly 400´000 transistors, over 100 miles of wiring
- A predecessor of the RISC-Architecture. Developed by Seymour Cray
- Prize: 7 – 10 Million $ (and by that time, the dollar was worth something)

MIMIC (Advantages)

Robotics and Mechatronics Centre

- The model could be formulated by assignments and integrators.
- These model "equations" could be arbitrarily ordered.
- The appropriate order for the state-space form is automatically derived.
- The time-discretization is not part of the model anymore. Different numerical ODE-solvers can be applied (better than FE)

```
   CON(G)        Declaration of constants
   PAR(1X0,X0)   Declaration of parameters
DT 0.05          Definition of time step

1X INT(-G*Z,1X0)  Integration
 X INT(1X,X0)
 Y 1.-COS(X)      Equation for y position
 Z SIN(X)         Equation for z position

   FIN(T,4.9)    Command for integration

   PLO(T,X,Y,Z)  Commands for plotting
   ZER(0.,-5,0.,-1)
   SCA(5.,5.,2.,1.)

   END  End of program
```


MIMIC (Deficiencies)

Robotics and Mechatronics Centre

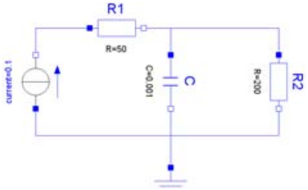
- MIMIC could not handle real equations, only causal assignments.
- There were hardly any means to structure the program. The language was almost completely flat and there is only one global namespace.

```
   CON(G)        Declaration of constants
   PAR(1X0,X0)   Declaration of parameters
DT 0.05          Definition of time step

1X INT(-G*Z,1X0)  Integration
 X INT(1X,X0)
 Y 1.-COS(X)      Equation for y position
 Z SIN(X)         Equation for z position

   FIN(T,4.9)    Command for integration

   PLO(T,X,Y,Z)  Commands for plotting
   ZER(0.,-5,0.,-1)
   SCA(5.,5.,2.,1.)

   END  End of program
```


Dymola

Robotics and Mechatronics Centre

- The **Dy**namic **Mo**deling **La**nguage was developed by Hilding Elmquist in 1978.
- The listing on the left displays the code of an assembled electric circuit and of its capacitor component.




```
model type capacitor
  cut A (Va / I) B (Vb / -I)
  main cut C [A B]
  main path P <A - B>
  local V
  parameter C
  V = Va -Vb
  C*der(V) = I
end

model Network
 submodel ( resistor ) R1 R2
 submodel ( capacitor ) C
 submodel ( current ) F
 submodel Common
 input i
 output y
 connect Common to F to R1 to (C par R2)
    to Common
 E.I = i
 y = R2.Va
end
```


Dymola

Robotics and Mechatronics Centre

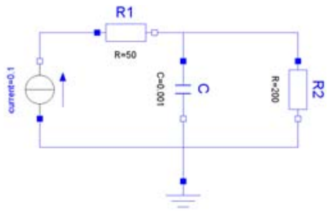
- Dymola is a **declarative** language. It only contains code for the model-equations. The simulation is completely decoupled from the model description.
- This language enabled the formulation of hierarchic elements such as sub-components.
- These components could be automatically connected.

```
model type capacitor
  cut A (Va / I) B (Vb / -I)
  main cut C [A B]
  main path P <A - B>
  local V
  parameter C
  V = Va -Vb
  C*der(V) = I
end

model Network
 submodel ( resistor ) R1 R2
 submodel ( capacitor ) C
 submodel ( current ) F
 submodel Common
 input i
 output y
 connect Common to F to R1 to (C par R2)
    to Common
 E.I = i
 y = R2.Va
end
```

Dymola

Robotics and Mechatronics Centre

- Dymola can handle non-causal equations such as u = R*i



- In R1, the causality is: u := R*i
- In R2, the causality is: i := u/R
- In Dymola, one can use the same, non-causal equations for both resistor components.

```
model type capacitor
  cut A (Va / I) B (Vb / -I)
  main cut C [A B]
  main path P <A - B>
  local V
  parameter C
  V = Va -Vb
  C*der(V) = I
end

model Network
 submodel ( resistor ) R1 R2
 submodel ( capacitor ) C
 submodel ( current ) F
 submodel Common
 input i
 output y
 connect Common to F to R1 to (C par R2)
    to Common
 E.I = i
 y = R2.Va
end
```


Omola

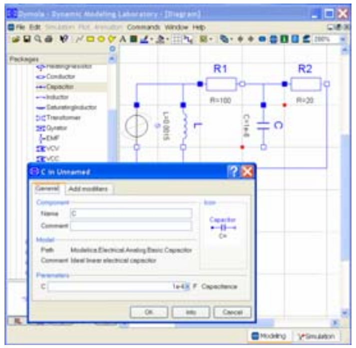
Robotics and Mechatronics Centre

- *Dymola* never had any real impact in industry, it remained within academia.
- There, its main ideas were preserved and extended by *Omola*. This language enables a truly object-oriented modeling, featuring inheritance, wrapping etc.
- Modeling in Omola was also performed graphically. Only the fundamental equations are entered in textual form. All higher-level model are assembled graphically.
- Also Omola remained within academia. Things started to change as *Modelica* was born in 1997.

Modelica

Robotics and Mechatronics Centre

Demonstration



Modelica

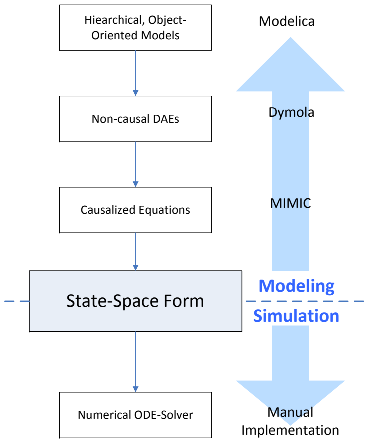
Robotics and Mechatronics Centre

- As you see: Dymola is still alive, but not as modeling language but as an M&S environment for Modelica.
- In Modelica, we can directly punch in our model equations.
- There is no need anymore to derive the state-space form by paper and pencil.

```
model Yeast
  parameter Real V = 1 "volume of fermentation vessel";
  parameter Real s0 = 0.2 "initial concentration of sugar";
  parameter Real p0 = 1e-6 "initial population of yeast";
  parameter Real C_f = 50 "Feeding Coefficient [1/day]";
  parameter Real R = 10 "Reproductivity [1/day]";
  parameter Real S = 15 "Sensitivity w.r.t. alcohol [1/day]";
  parameter Real T_ref = 300 "reference temperature";
  Real p "population of yeast";
  Real b "birth rate";
  Real d "death rate";
  Real s "concentration of sugar";
  Real a "concentration of alcohol";
  Real f "consumption of sugar (feeding)";
  Real T "current temperature";

initial equation
  p = p0;
  s = s0;
equation
  f = s * p * C_f * (T/T_ref);
  a = s0 - s;
  b = R * s;
  d =  S * a;
  T = 310;
  der(p) = p*(b-d);
  V*der(s) = -f;
 end Yeast;
```

Modeling and Simulation

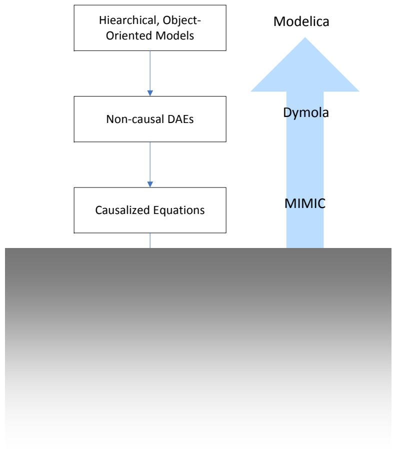
Robotics and Mechatronics Centre



- In the field of programming languages, there are high-level languages (Phyton, C++) and low-level languages (Assembler)
- The same is true for modeling languages.
- The state-space form is a common target of their compilation scheme (the Assembler language of a modeler).

Modeling and Simulation

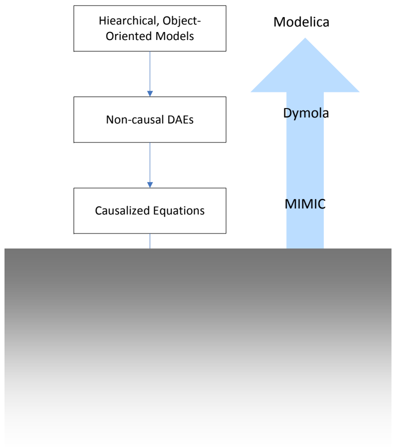
Robotics and Mechatronics Centre



- The first (and larger) part of the lecture concerns the modeling side.
- You will learn to model in Modelica using the software Dymola.

Modeling and Simulation

Robotics and Mechatronics Centre

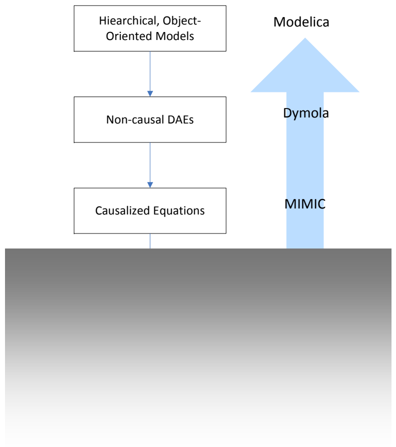


The way up:

- To this end, we have to learn how to formulate the laws of physics in an object-oriented way.

Modeling and Simulation

Robotics and Mechatronics Centre

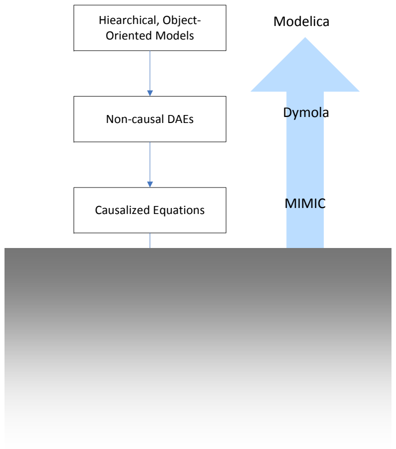


The way up:

- To this end, we have to learn how to formulate the laws of physics in an object-oriented way.
- This is a sole matter of physics. It has nothing to do with computer science.

Modeling and Simulation

Robotics and Mechatronics Centre

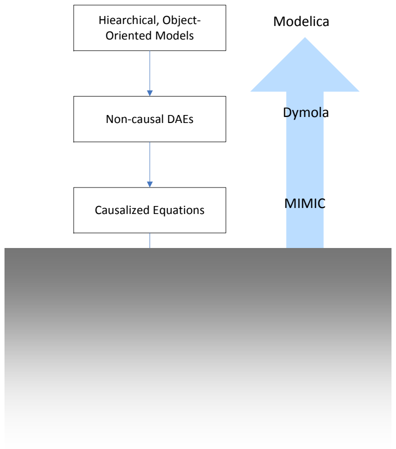


The way down:

- Then we have to learn how the languages are compiled, and how the state-space form is automatically derived.

Modeling and Simulation

Robotics and Mechatronics Centre

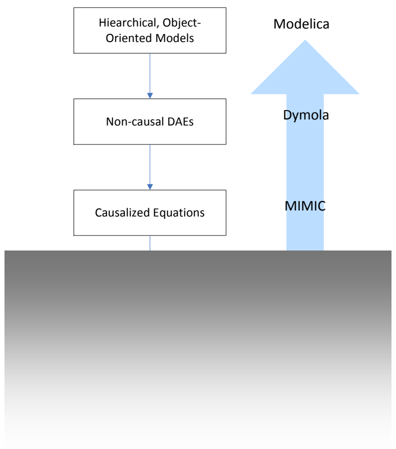


The way down:

- Then we have to learn how the languages are compiled, and how the state-space form is automatically derived.
- This is a sole matter of computer science. It has nothing to do with physics.

Modeling and Simulation

Robotics and Mechatronics Centre

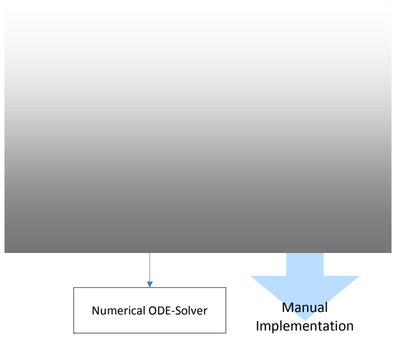


The way down:

- Then we have to learn how the languages are compiled, and how the state-space form is automatically derived.
- This is a sole matter of computer science. It has nothing to do with physics.
- (And by the way, we are going to model a lot of cool systems...)

Modeling and Simulation

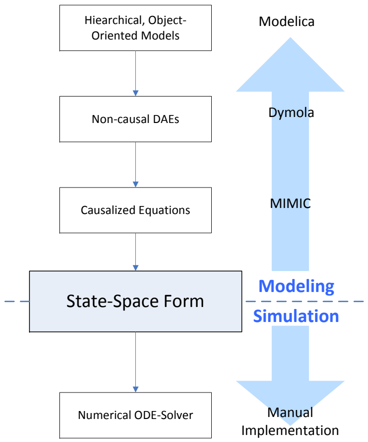
Robotics and Mechatronics Centre



- The second (and smaller) part of this lecture series concerns simulation.
- You will learn different techniques how to implement numerical ODE solvers, and how they influence the simulation result.
- In addition, the handling of events will be discussed.

Modeling and Simulation

Robotics and Mechatronics Centre



Questions?