UNIVERSITY OF VALLADOLID
(Spain)

EcosimPro and its EL Object-Oriented Modeling Language

ALBERTO JORRÍN
CESAR DE PRADA
PEDRO COBAS

The paper:

- EcosimPro : modeling and simulation tool

  ***new version 4 (4.4) : object orientation in Ecosimpro Language (EL) ***

  ->*official language :* ESA

- The use of classes gives power to EcosimPro.

What is ECOSIMPRO?

Simulation tool:

-> EA International

-> Modeling simple/comlpex physical systems

*** expressed:

a) differential/albegraic equations

b) ordinary/differential equations

+

discrete events

- Runs on the various Windows platforms
- Uses its own graphic environment for model design

What is EL?

The language used in EcosimPro

-> modeling systems:

*** combined: continuos-discrete

-> intuitive representation

-> Also to prepare experiments on models:

-> calculate steady states

-> transients

-> perform parametric studies

-> Generate reports, plots

-> Reuse C/FORTRAN functions and C++ classes

-> designed to be used in industry directly

(very complex systems / hundred var-eq)

*** succesfully used for aerospace applications

Key concepts in eEcosimPro

Component:

- Represent a model of a system
  - -> Variables
  - -> Differential-algebraic equations
  - -> Topology
  - -> Event-based behaviour
- Equivalence: “class” concept in OOP
- All components have:
  - *** CONTINUOUS block: continuos equations
  - *** DISCRETE block: discrete events

Port connection type

set of variables:

- to be interchanged

set of restrictions:

- to be shared

Example: Electric connection uses:

voltage and current

- ***Partition***

  - Associated mathematical model

    -> necessary to simulate a component

  - A component may have more than one partition

  - Defines the causality of the final model

- ***Experiment***

  Simulation case for:

    -> A partition of a component

- ***Library of components***

  Clasify components by disciplines

Mathematical capabilities

✓ Symbolic handling equations

- derivation
- equations reduction…

✓ Robust solvers for:

- non-linear equations (Newton-Raphson)
- DAE systems (DASSL, Runge-kutta)

✓ Uses dense and sparse matrix formats

Allows problems with thousands of state variables to be simulated

- ✓ Has math wizards for:
  - - Defining design problems
  - - Defining boundary conditions
  - - Solving algebraic loops
  - - Reducing high-index DAE problems

- ✓ Clever mathematical algorithms
  - - based on the graph teory -> minimize:
    - * number of unknown variables
    - * number of equations

- ✓ Powerful discrete events handler

EL and Object Orientation

- In Ecosim the complexity is hidden:

to solve systems of differential-algebraic equations …
…*<u>The user</u>*: define high level equations
by high level object-oriented language

EL is object-oriented:
components can:
- inherit from one another
- be aggregated to create other more complex
- Reuse ones to create other more complex
*** incrementally

Object oriented modeling

- **To outlive inevitable changes**:

  -> growht/ageing (any dynamic system)

- **Provide the modeler**

✓ *POWERFUL FEATURES*:

---**To hide complexity** **by**:

1- encapsulation

*** main elements: libraries + componentes

-> Convencional EOO language (C++)

Interface: data + methods (public)

-> With EL:

Components interface:

ports + construct parameters + data

--- **To enable reuse** by:

2- inheritance

3- aggregation

Many components being developed:

-> Share behaviour

-> EL bring:

common data + equations

(parent components)

*** EL also provide multiple inheritance

---To create independent models

---To create models easy to mantain

- SO:

EL is Bottom-up:

- Basic library components can be combined to create (increasingly) complex components by combining two methods:

  a) Extension: by inheritance from existing components

  b) Instantiation and aggregation of existing components

Application: create a component which represents a complete system.

- Intermediate components can also be simulated

->->->->-> Reduction: development + maintenance time

Classes

- Equivalents to classes in classic OOPL (C++, Java, …)

  -> Use: more restricted

  more simple

- Compilation -> EL -> C++ (internally) :

  "High level wrappers"

**** Final users are engineers and not programmers ****

- Difference component vs. Class

  - **component**

    elements to be solved by the simulation tool

    +++ dynamic equations

    +++ discrete events

  - **class**

    set of behaviour

    +++ variables

    +++ methods

…. so…

classes are normally used in EL:

-> to support the modeling of complex systems:

… improving the use of functions:

all the functions referring to the same utility ->->-> group together ->-> share memory

(its common variables)

```
class_def : CLASS IDENTIFIER ( IS_A
scoped_id_s)?
DESCRIPTION?
( DECLS var_object_decl_s )?
( OBJECTS class_instance_stm_s )?
( METHODS method_def_s )?
END CLASS
```

- DECLS BLOCK

  Any kind of basic EL variable can be defined:

  - simple variable
  - multidimensional array

- OBJECTS BLOCK

  Declaration of instances of classes

- METHODS BLOCK

  - Defines the functional interface of a class

    (are subroutines connected to a definition of a class)

  - Can return a basic EL type (like functions)

Using classes

- Defined in EL can be used in:
  - functions
  - components
  - experiments
  - other classes

use: the same way as in other OOPL.

point operator:

- all their variables
- public mehotds

Class associated with a partition

- When generating a partition…
  then automatically generate:

***internal class**: *represent the mathematical model*

Advantages:

- Any partition can be encapsulated in a single class
- This class provides an interface for interacting with the partition:
  - initialization of variables
  - steady and transient calculations,
  - get values of variables,etc

- Simulations can be embedded in:
  - components
  - functions
  - experiments
  - classes

  …since they are programmed with the class interface

- Multiple experiments can be executed in the same run

- Child classes can be created by adding new variables and methods:

  a child class could provide complex experiments embedded in a single method

- *Makes the language very powerfull*

  *** embedding mathematical models inside others*

An ilustrative example:

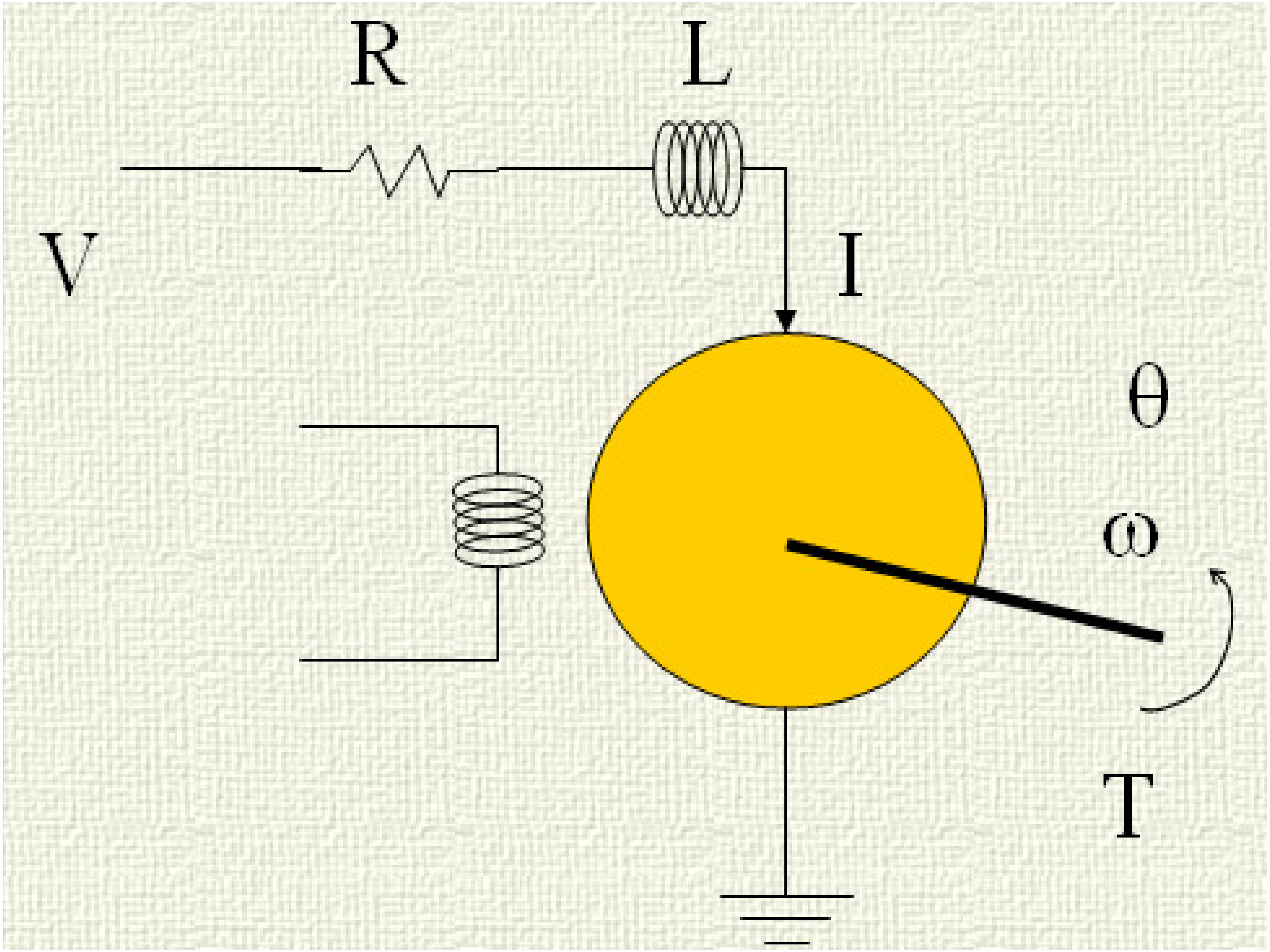
Initialization of models

- Objective: allow to start a simulation from stationary conditions

  -> classes are used to formulate it

Example:

Problem electrical engine


R
L
V
I
θ
ω
T

$$J\frac{d\omega}{dt} = k_1 i - f\omega - T$$ Newton Law

$$V = Ri + L\frac{di}{dt} + k_2\omega$$ Ohm Law

ω angular velocity

J moment of inertia

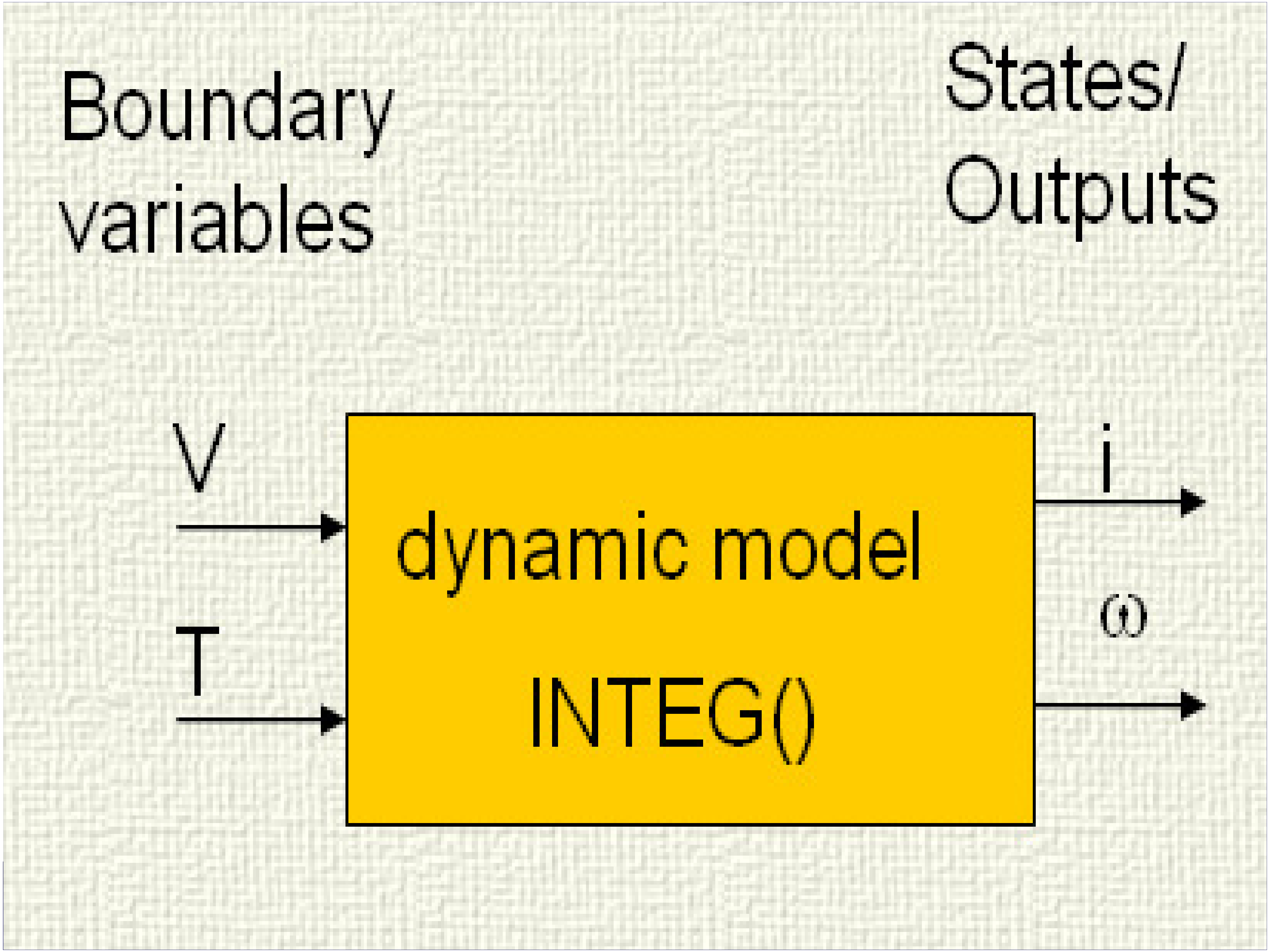
V source voltage

I armatura current

T external torque

$k_2\omega$ f.c.e.m

```
COMPONENT engine

   DATA

      REAL R = 0.2         -- electric resistance (ohmios)
      REAL L = 0.01        -- electric inductance (H)
      REAL k1 = 0.006      -- armature constant (lbs-pie/A)
      REAL f               -- damping ratio of the mechanical system(lbs-pie/rad/seg)
      REAL J = 0.001       -- moment of inertia of the rotor (slug-pie2)
      REAL k2 = 0.055      -- speed constant (V.seg/rad)

   DECLS

      REAL V               -- source voltage (V)
      REAL omega           -- angular velocity
      REAL i               -- armatura current
      REAL T               -- motor torque applied to the shaft


   CONTINUOUS

      J * omega' = k1*i - f *omega - T
      L*i' = V - R*i - k2*omega


END COMPONENT
```


Boundary variables
States/ Outputs
V
T
dynamic model
INTEG()
i
ω

- Intuitively…

to start with stationary conditions:

$$i(0) = \frac{V(0) - k_2\omega(0)}{R}$$

$$T(0) = k_1 i(0) - f\omega(0)$$

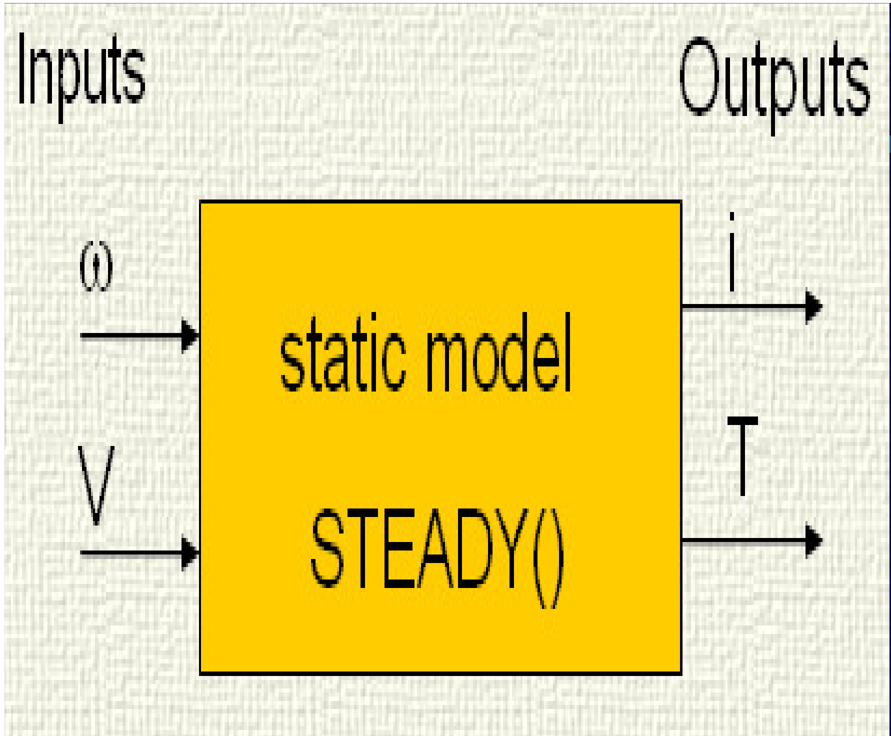
equations in the INIT block:

--- only if the model is easy:

->->-> use of classes to solve it:

creating a static partition with the component

… in this case the partition would be….

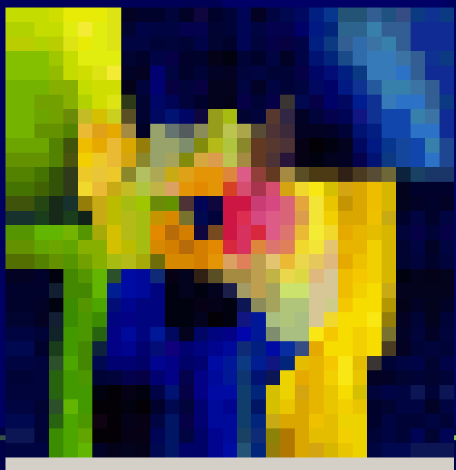


With “w” and “v” values could be computed the i(0) value

$$i(0) = \frac{V(0) - k_2\omega(0)}{R}$$

with the i(0) value, could be computed the T(0) value

$$T(0) = k_1 i(0) - f\omega(0)$$



COMPONENT motor

Data

Decls

INIT

CONTINUOUS

W'= ....

I'=....

END

```
CLASS Stationary IS_A engine_static

METHODS
   METHOD NO_TYPE run()

      BODY
         STEADY()

   END METHOD

END CLASS


COMPONENT engine_test

   DATA

      REAL R = 0.2        -- electric resistance (ohmios)
      REAL L = 0.01       -- electric inductance (H)
      REAL k1 = 0.006     -- armature constant (lbs-pie/A)
      REAL f              -- damping ratio of the mechanical system(lbs-pie/rad/seg)
      REAL J = 0.001      -- moment of inertia of the rotor (slug-pie2)
      REAL k2 = 0.055     -- speed constant (V.seg/rad)

   DECLS

      REAL V              -- source voltage (V)
      REAL omega          -- angular velocity
      REAL i              -- armatura current
      REAL T              -- motor torque applied to the shaft
```

```
  OBJECTS

     Stationary stac

  INIT

     --i0 = (V0-k2*omega0)/R
     --T0 = k1*i0-f*omega0


     omega = 8.
     V = 5.

     stac.setValueReal("omega",omega)
     stac.setValueReal("V",V)

     stac.setTraceProgramme(TRUE)
     stac.STEADY()

     i = stac.getValueReal("i")
     T = stac.getValueReal("T")


  CONTINUOUS

     J * omega' = k1*i - f *omega - T
     L*i' = V - R*i - k2*omega


END COMPONENT
```

... conclusions

EL, is therefore one of the ***pioneer languages*** that has to deal with this new way of Modeling physical systems.

I- Advantages for the modeller

- minimise global data
- hides the complexity
- comprises:
    * parameters
    * data
    * ports

privacy:

* discrete events
* equations

- complexity grows in a linear

- reuse

- inheritance: simplifies the modelling

- equations inserted at the time of simulation

- use of virtual equations

- equations format declarative

*** algorithms symbolically transform the equations

II- Considerations in the revolution of OOM

- Modelling is non-causal

- Tried and tested components are constantly reused

- Extensive use of:
  - # hidden information
  - # encapsulated data

  … to deal with the complexity

- Gift for:

  make change in the models