Integration of Giotto and Simulink

Wolfgang Pree
University of Salzburg, Austria
www.SoftwareResearch.net

A joint project of
C. Kirsch and W. Pree

Contents

- **Relevant Simulink concepts**
- **Integration options**
- **Chosen seamless integration:**
  - gTranslator tool & Giotto component library for Simulink
  - Harnessing Simulink‘s code generation
- **Case study: electronic throttle control**
  - model preparation and transformation
  - implications

Relevant Simulink concepts

- data-flow paradigm
- model execution engine
- S-functions

Simulink paradigm

- data-flow orientation as core principle:
  - blocks + data-flow connections
  - subsystems
- but:
  - imperative blocks
  - mixing of continous and discrete blocks is regarded as too complex:
    variable step solvers, multiple rates, major and minor time steps

Model execution

- **initialization phase:**
  - **block sorting** determines execution order; user-defined priorities might change the order
  - socalled non-virtual (:: atomic) **subsystems are flattened**
- **execution phase:**
  - **iterative computation of**

    (1) block outputs

    (2) block states

    (3) next time step

Customization

- **no programming:**
  parameters for subsystems through masks (= dialogs)
- **S(ystem)-function blocks:**
  - can be programmed in C, Ada, Fortran or Matlab
  - have to adhere to Simulink‘s callback architecture

Simulink's callback architecture

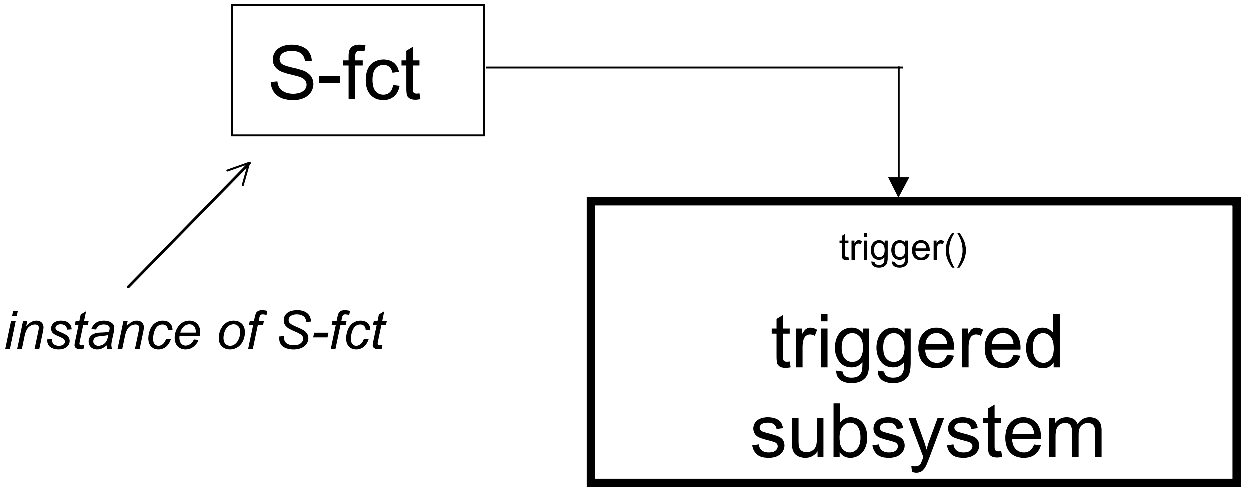
The following callback functions are invoked by Simulink's runtime system for each block that contains an S-function:

```
mdlInitializeSizes(...)
mdlCheckParameters(...)
mdlInitializeSampleTimes(...)

for each time step in the simulation
        mdlOutputs(...)
        mdlUpdate(...)

mdlTerminate(...)
```

Example: S-function triggering the execution of a subsystem

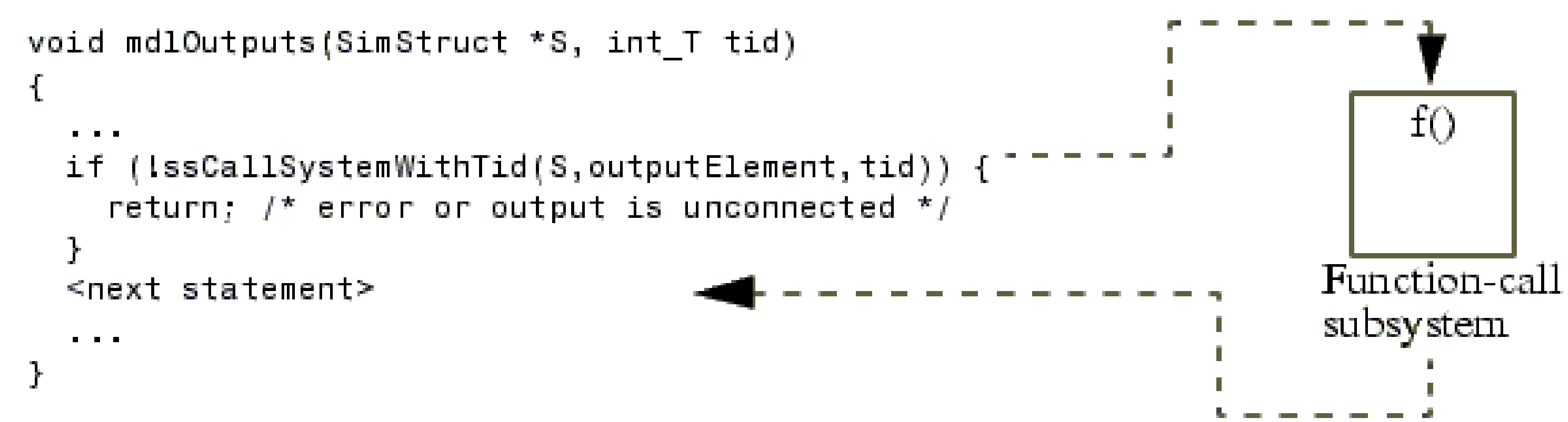


S-fct

instance of S-fct

trigger()

triggered subsystem



```
void mdlOutputs(SimStruct *S, int_T tid)
{
  ...
  if (!ssCallSystemWithTid(S,outputElement,tid)) {
    return; /* error or output is unconnected */
  }
  <next statement>
  ...
}
```

Integration options

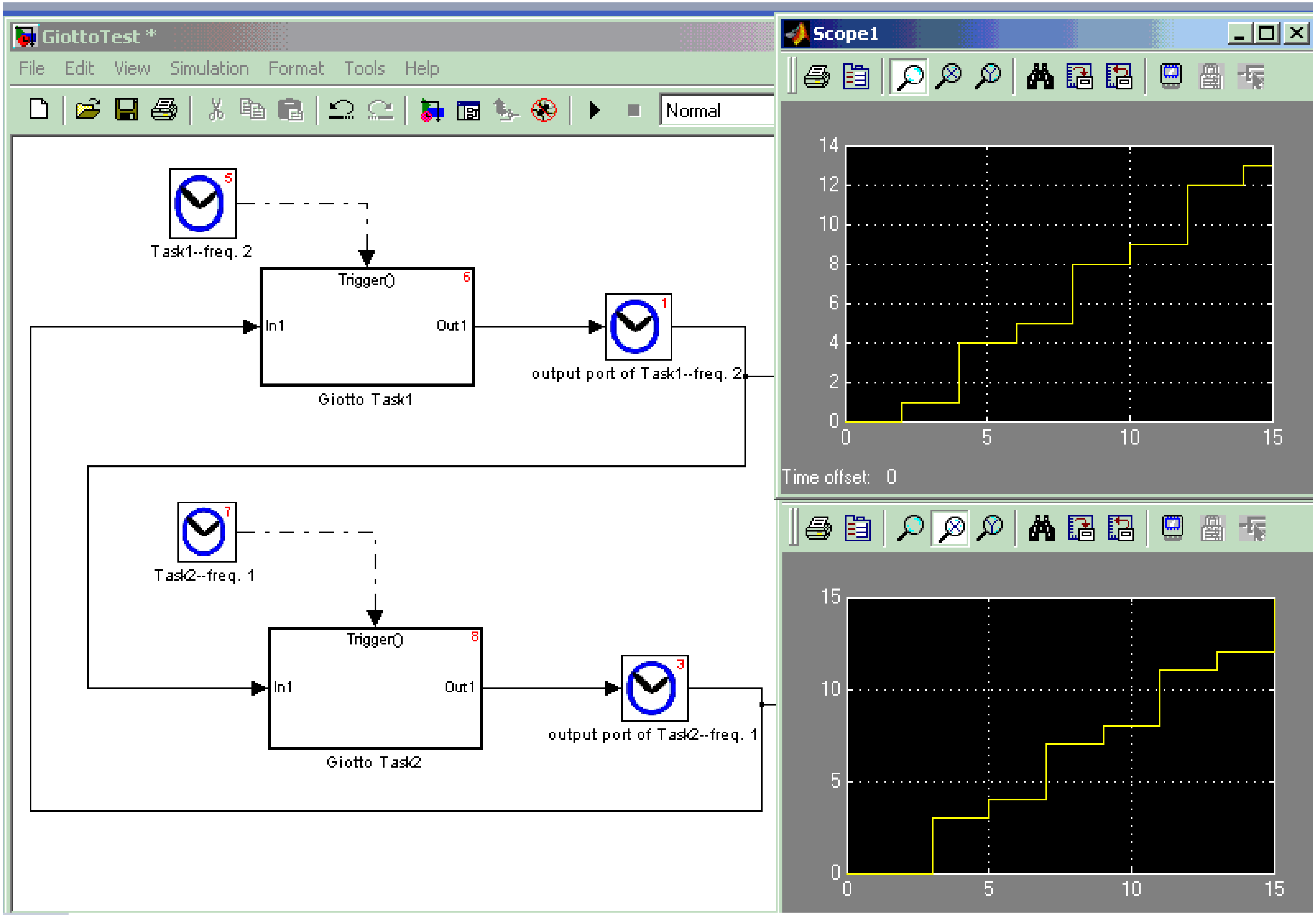
- "inside": S-functions
- "on top": seamless integration by means of Simulink‘s own blocks

Core concepts of the Giotto S-function

- separation of task communication and task triggering
- only one Giotto-S-function
- we use mdlUpdate as hook and do the following at each simulation time step if the frequency of an instance of a Giotto-S-function requires it:

  if the Giotto-S-function instance is at an output port the outputs are updated

  if the Giotto-S-function triggers a subsystem, it lets it execute


GiottoTest *
File
Edit
View
Simulation
Format
Tools
Help
Normal
Task1--freq. 2
5
Trigger()
6
In1
Out1
Giotto Task1
1
output port of Task1--freq. 2
Task2--freq. 1
7
Trigger()
8
In1
Out1
Giotto Task2
3
output port of Task2--freq. 1
Scope1
14
12
10
8
6
4
2
0
0
5
10
15
Time offset: 0
15
10
5
0
0
5
10
15

Hitting the wall: code generation (I)

The straight-forward option, ie, 1:1 code generation

- **does not allow preemption:**
  - the time intervals between simulation steps have to be as small as determined by the fastest Giotto task
  - all task computations have to be done within that interval

- is inefficient:
  An S-function's C-code is used as it is in the generated real-time system

Hitting the wall: code generation (II)

- Simulink's Real-Time Embedded Coder (eg, for Windows) would allow the generation of C-functions for each subsystem corresponding to a Giotto-Task

but

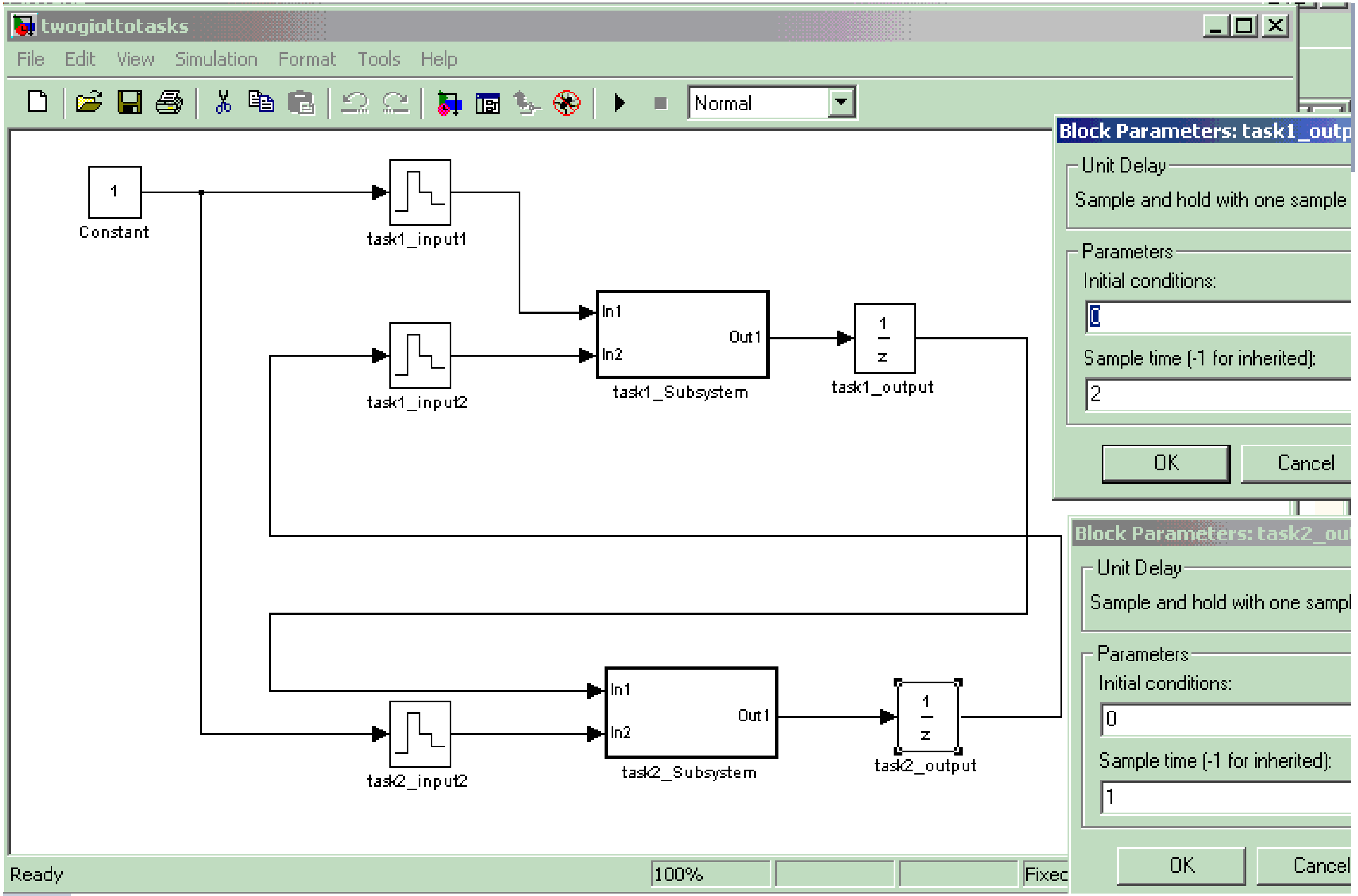
- the generated code **does not provide a clean parameter passing** to the functions
- thus the code generated by Simulink would have had to be modified:
  - maybe for each different target ??
  - generated code might change for each new version of coder generation tools ??

being "inside Simulink" is considered harmful anyway

- **the execution mechanism has changed from version 6.0 to 6.1 without any notice** in the documentation:
  C-code from mdlOutput had to be moved to mdlUpdate in the Giotto S-function
- subtle differences between simulation and real-time versions for S-function implementations
- **problems with the semantics of blocks,**
  eg, an atomic subsystem causes errors that a virtual one does not

Seamless integration

- Basic concepts
- gTranslator tool & Giotto component library
- Harnessing Simulink's code generation



Block Parameters: task1_outp

Unit Delay

Sample and hold with one sample

Parameters

Initial conditions:

0

Sample time (-1 for inherited):

2

Block Parameters: task2_ou

Unit Delay

Sample and hold with one sampl

Parameters

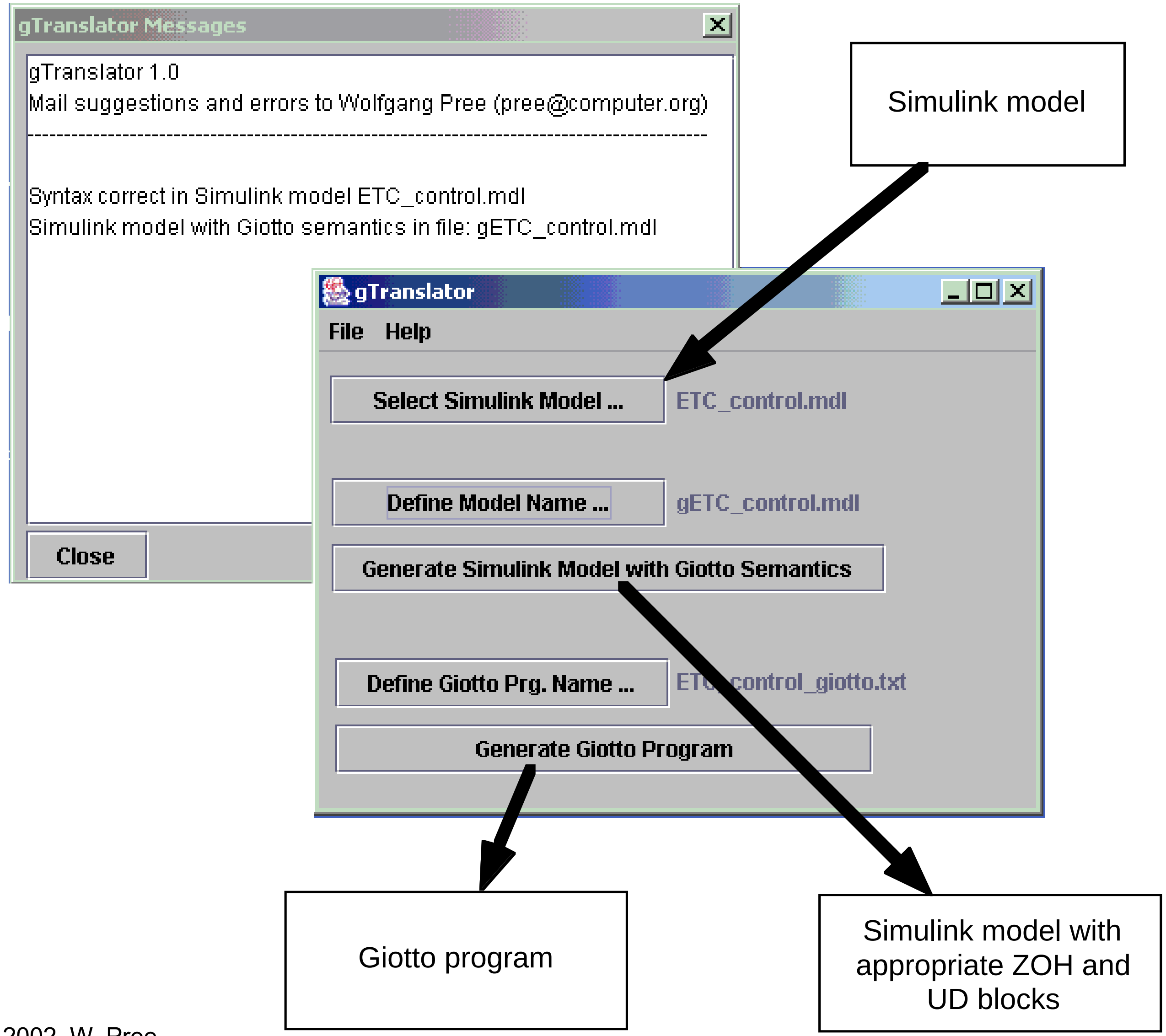
Initial conditions:

0

Sample time (-1 for inherited):

1

Automating the model transformation



```
gTranslator 1.0
Mail suggestions and errors to Wolfgang Pree (pree@computer.org)
--------------------------------------------------------------------------------

Syntax correct in Simulink model ETC_control.mdl
Simulink model with Giotto semantics in file: gETC_control.mdl
```

Simulink model

Giotto program

Simulink model with appropriate ZOH and UD blocks

gTranslator's parsing

the Simulink model is stored as plain text adhering to the following simplified syntax described in EBNF:

```
MDLModel := "Model {" MDLHeader MDLSystem "}".
MDLHeader:= CharSeq.
MDLSystem:= "System {" MDLSystemHeader
                       MDLBlock
                       (MDLBlock|MDLLine)*
                   "}".
MDLSystemHeader:= CharSeq.
MDLBlock:= "Block {" MDLBlockDescription.
MDLBlockDescription:= CharSeq "}".
MDLLine:= "Line {" MDLLineDescription.
MDLLineDescription:= CharSeq "}".
CharSeq:= (ASCII-char)*.
```

gTranslator demonstration

Demonstration of the preparation and translation of the ETC model (Mobies)

Future plans

Next steps

- **integration of Giotto modes into Simulink**

- **enhancing reusability through combining**
  - **Giotto** as composition standard for safety-critical embedded control components
  - **Frameworks** for high-level, less time-critical management functionality

- **gTranslator as Web service**

The end

Thank you for your attention!