

1 Public interface for the generation of PsyC code

In the ITBC 5 of the INTERESTED project, a transformation has been developed from Scade models to PsyC language used to develop applications for execution on the OASIS real-time safety-critical platform. This transformation relies on a template that can be used in conjunction with any dataflow synchronous source language under certain hypotheses. Below we present these assumptions and the template.

1.1 Assumptions for PsyC code generation

The INTERESTED Scade to OASIS transformation is implemented as a SCADE Suite adaptor. The adaptor provides proper encapsulation in PsyC of the C functions generated from the SCADE Suite model.

The following assumptions are made about the source environment:

1. Models in the source language are networks of operators communicating by data flow and executed cyclically with synchronous operational semantics, which means here, by repeating the following three steps:
 - a. Acquisition of inputs;
 - b. Execution of one computation step of the network;
 - c. Publication of the produced outputs.
2. For each operator F , two C functions are provided: a function F executing one computation step, and F_reset initialising the state of the operator.

The parameters of the function F correspond to the input variables of the operator and an additional parameter of type `outC_F`, which is a structure containing all the state and output variables of the operator. This function does not return any values.

Function F_reset takes a single parameter of type `outC_F` and does not return any values.

3. All the operator instances (nodes of the network) are assigned unique id numbers (0, 1, 2 ...) respecting the computation causality order.
4. Inputs and outputs of the model are connected to some fake operators, designed as imported operators. The reason is that a PsyC description uses specific agent for input/output interconnection to the environment. Providing these fake operators allows the user to finally replace them with the actual agents that are target dependant.

1.2 PsyC code generation template

The PsyC code generator using this interface must generate:

- `App_defs.h` – an application header file with the description of the clocks of the agents
- `Operator_n.psy` – one PsyC file for each instance of an operator

The numbers are used to differentiate between several instantiations of the same operators, and are global.

Below is a template for generation of PsyC code for an agent corresponding to an instance of an operator.

```

#include "App_defs.h"
#include "[Operator].h"

agent AG [Operator] [n] (starttime = 1 with CK [Operator] [n]) with CBASE {

    global {
        outC_[Operator] outC;
    }

    temporal {
        [Output variable type] 0$[Output variable name];
    }

    display {
        [Output variable name]: [List of consumers for the output variable];
    }

    consult {
        [Producer] 1$[Input variable name];
    }

    body start {
        [Operator]_reset (&outC);
        next main;
    }

    body main {
        [Operator] ([Producer]`0$[Input variable name], &outC)
        [Output variable name] = outC.out;

        advance (1);
        advance (1) with CK [Operator] [n];
    }
}

```

In this template, the elements that need to be substituted by information specific to the operator, for which the code is being generated. More precisely:

[Operator] is the name of the operator.

[n] is a global number assigned to operator instances (see Section 1.1).

[Output variable type] is the type of the output variable.

[Output variable name] is the name of the output temporal variable.

[List of consumers for the output variable] is the list of the agents that consult the output temporal variable in question.

[Input variable name] is the name of the input temporal variable.

[Producer] is the agent that produces the input temporal variable in question.

Finally, one should observe that, in this template, only one input and one output variable are used. Should several input or output variables be necessary, the corresponding elements of the template should be reproduced accordingly in a straightforward manner.

1.3 Application header file

The template for the application header file App_defs.h is presented below. The elements that need to be supplemented in the template are self-explanatory (cf. previous sections).

```
clock CBASE = [OASIS clock definition]

#define NAGENTS [Number of operators]

#define [Operator instance 0]_NUM 0
#define [Operator instance 1]_NUM 1
#define [Operator instance 2]_NUM 2
...

clock CK [Operator] 0 = NAGENTS*CBASE + [Operator instance 0] NUM;
clock CK_[Operator]_1 = NAGENTS*CBASE + [Operator instance 1]_NUM;
clock CK_[Operator]_2 = NAGENTS*CBASE + [Operator instance 2]_NUM;
...

#include "...
...
```

The #include instructions at the end are used to include definitions of any application specific types and/or constant values.