MITSUBISHI ELECTRIC
Changes for the Better

SysML to B Translation Principles

File name:	2014-12-04-SYSMLTOB-TRANSLATIONPRINCIPLES_MFR14-ARC-840--MFR14-ECD-531.DOCX
Version:	0.2
Date:	2014-12-04
Author(s):	T. Bardot
Classification Status:	Non confidential
Export Control Status:	NLR
Edition Status:	Draft

Approved by:	DME	Date:	2014-12-03
Diffusion list:	openETCS, Public		

Revision history

Version	Date (DD/MM/YYYY)	Description	Editor(s)
0.1	17/11/2014	Draft	T.Bardot
0.2	2014-12-03	Review	DME
0.2	2014-12-04	openETCS release	DME

Table of Content

1 INTRODUCTION

This document is under openETCS license.

This document describes how the SysML elements are translated in B code by the SysML to B translator tool.

2 REFERENCE DOCUMENTS

Tag	Document
[1]	D2_4 *Definition of the methods used to perform the formal description* https://github.com/openETCS/requirements/blob/master/Reference/D2_4.pdf?raw=true
[2]	SysML Modeling rules for SysML to B (MFR14-ARC-839)

3 B MODEL OVERVIEW

3.1 ARCHITECTURE

The generated B model has the architecture shown on Figure 1.

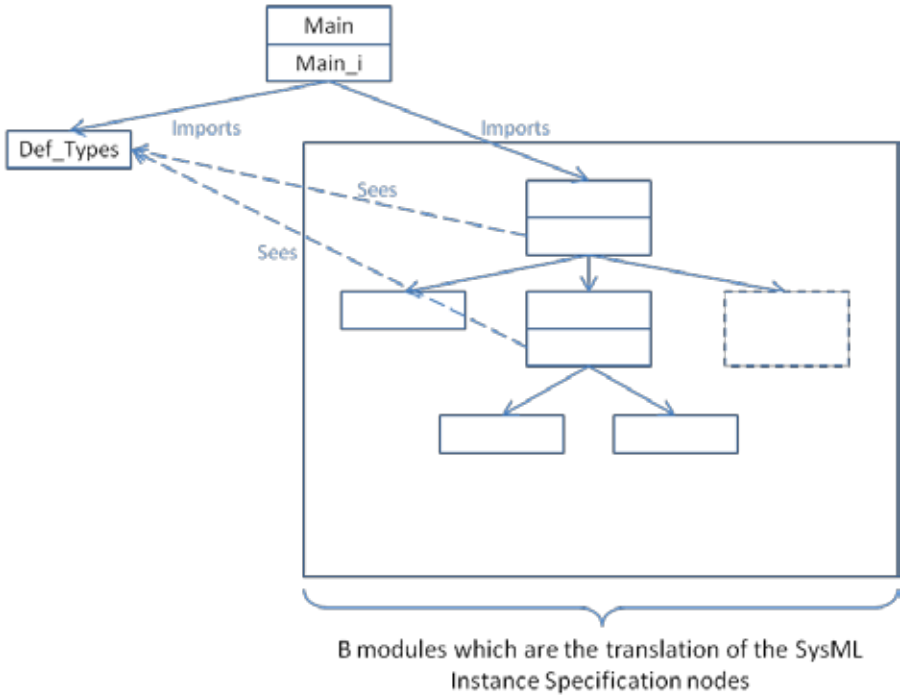


Figure 1: graphical representation of the architecture of the B model automatically generated by the translator tool

A root machine named Main is automatically generated. It has an operation named *run_cycle* which calls the schedule operation of the imported B machine. Figure 2 and Figure 3 shows the B code of the machine *Main*.

```
MACHINE
   Main
OPERATIONS
run_cycle =
   BEGIN
      skip
   END
END
```

Figure 2: Main.mch

```
IMPLEMENTATION
   Main_i
REFINES
   Main
IMPORTS
   Def_Types,
   CoffeeMachine_1
OPERATIONS
run_cycle =
   BEGIN
      CoffeeMachine_1_schedule
   END
END
```

Figure 3: Main_i.imp where *CoffeeMachine_1_schedule* is the schedule operation of the B machine which corresponds to the root SysML Instance Specification

The machine *Def_Types.mch* is used to defined the enumerated sets that are used by the model. An example of the Def_Type code is given in Figure 4.

```
MACHINE
   Def_Types
SETS
   E_Phase={in,out,scheduleReq};
   E_SugarLevel={level0,level1,level2,level3,level4};
   Status={active,nonActive}
END
```

Figure 4: *Def_Types.mch*, where *E_SugarLevel* and *Status* are an example of a translation of SysML Enumeration Nodes. *E_Phase* is always generated

3.2 SCHEDULING

A B machine which is a translation of an instance of a SysML block can have inputs and outputs.

The scheduling of one B machine which corresponds to an instance of a SysML block must follow these steps:

1. Set the value of the inputs: *Write* operations (see §6.2);
2. Determine the value of the outputs: *Schedule* operation (see §6.4);
3. Set the Value of the outputs: *Read* operations (see §6.1).

A variable called *Phase* is used to distinguish these steps. The *Phase* variable is set to the value "in" during the setting of the inputs, to the value "ScheduleReq" during the calculation of the outputs and to the value "out" when the new value of the outputs is set.

It is important to use the variable *Phase* for a B machine which is a translation of an instance of a SysML block that have a constraint. Indeed, a block constraint is translated to an invariant of the B

machine (see §5.1.3). This invariant is valid only when the new value of the outputs is set, so when *Phase* value is "out".

4 TYPE TRANSLATION

4.1 PRIMITIVE TYPE NODE

A SysML element typed with a Primitive Type Node will be translated to a B variable typed with a type which has the same name than the Primitive Node (see [2]).

For example, if a SysML Primitive Type called "NAT" is created, then a SysML element which is typed with this Primitive Type will be translated in B code to a variable typed with a type called NAT.

4.2 ENUMERATION NODE

A SysML Enumeration node is translated in B code to a set. All the sets which correspond to an Enumeration Node are declared in the B machine *Def_Types.mch*.

4.3 DATA TYPE NODE

A SysML element typed with a Data Type Node will be translated to several variables. One variable will be created for each property of the Data Type typed with a Primitive Type or an Enumeration.

An example is shown with the Figure 5 and Figure 6. The SysML output flow port *MakeCoffee* is translated in two B variables: *MakeCoffee_CoffeeSelected* and *MakeCoffee_SugarLevel.*

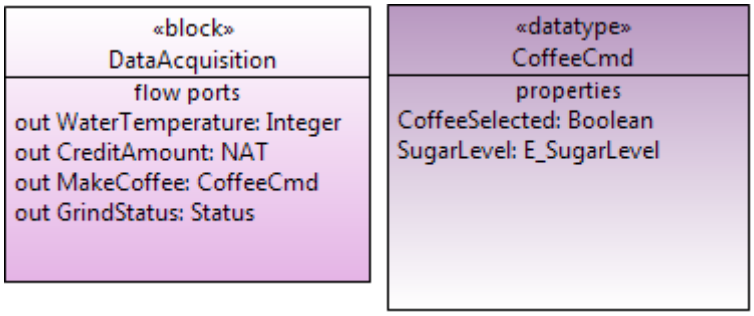


Figure 5: the block *DataAcquisition* has an output *MakeCoffee* which is typed with the Data Type *CoffeeCmd*

```
MACHINE
   Input_Interface_1
SEES
    Def_Types
ABSTRACT_VARIABLES
   /* Inputs */

   /* Outputs */
   CreditAmount,
   GrindStatus,
   MakeCoffee_CoffeeSelected,  MakeCoffee_SugarLevel,
   WaterTemperature

   /* Local Variables */

INVARIANT

   /* Inputs */

   /* Outputs */
   CreditAmount:NAT &
   GrindStatus:Status &
   MakeCoffee_CoffeeSelected:BOOL &  MakeCoffee_SugarLevel:E_SugarLevel &
   WaterTemperature:INT
```

Figure 6: a view of the translation of the SysML block *DataAcquisition*. The output *MakeCoffee* is translated into two variables: *MakeCoffee_CoffeeSelected* and *MakeCoffee_SugarLevel*

4.4 Flow Specification Node

A Flow Specification node is translated like a Data Type node.

5 Block translation

For each instance of a block, defined with an Instance Specification node, a B machine is created. So, a B machine corresponds to a unique instance of a unique SysML block.

5.1 B machine clauses

The generated B machine contains these B clauses:

- SEES
- ABSTRACT_VARIABLES
- INVARIANT
- INITIALISATION
- OPERATIONS

5.1.1 Clause SEES

The SEES clause is used to see the B machine *Def_Types* which contains the enumerated sets used in the model.

5.1.2 Clause ABSTRACT_VARIABLES

In the ABSTRACT_VARIABLES clause are declared the inputs, outputs and local variables of the B machine.

The inputs variables of the B machine are the translation of the flow ports with "In" direction of the corresponding block.

The outputs variables of the B machine are the translation of the flow ports with "Out" direction of the corresponding block.

The local variables of the B machine are the translation of the properties of the corresponding block.

A variable called *Phase* can also be declared in this clause. The *Phase* variable is used when at least one constraint is added to the block which is translated in the B machine (see §5.1.3).

An example of the ABSTRACT_VARIABLES clause is shown by Figure 7.

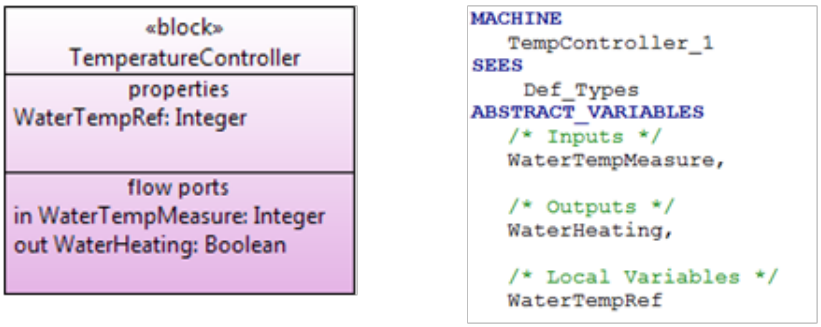


Figure 7: The ABSTACT_VARIABLES clause of the translation of the block *TemperatureController*. There is no variable *Phase* because the block *TemperatureController* does not have a constraint.

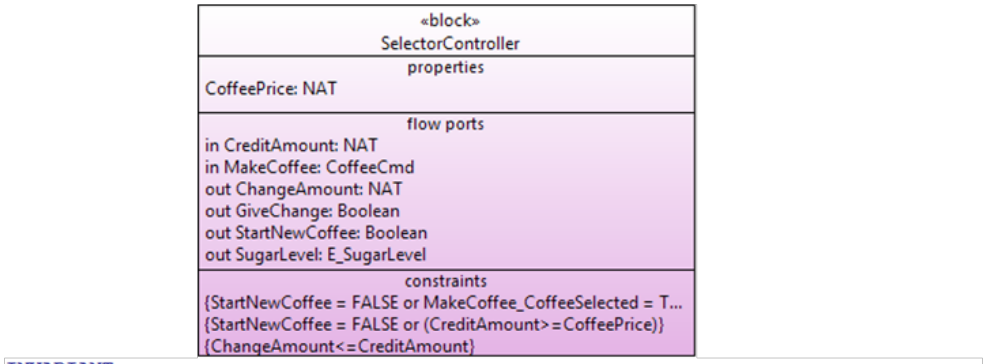
5.1.3 CLAUSE INVARIANT

In the INVARIANT clause, the inputs, outputs and local variables of the B machine are typed.

Also, if they exist, the constraints of the translated block are added to the INVARIANT clause. In this case, the constraints shall be valid only when *Phase* variable value is at out. *Phase* variable is also typed in this clause.

The translated constraint of a block shall be verified only after the outputs calculation. The *Phase* variable is set to the "out" value when the outputs have been calculated (see §6.4.2). It is why the specific invariants of a B machine (the translation of a block constraint) are verified only when the *Phase* variable value is "out".

An example shown with the Figure 8: the block constraint "`ChangeAmount<=CreditAmount`" is translated in the invariant "`Phase = out => (ChangeAmount<=CreditAmount)`".



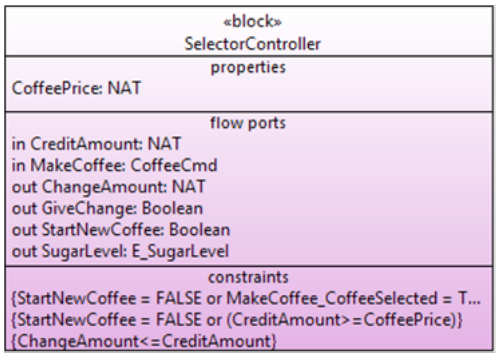

```
INVARIANT
   /* Phase variable */
   Phase : E_Phase &

   /* Inputs */
   CreditAmount:NAT &
   MakeCoffee_CoffeeSelected:BOOL &  MakeCoffee_SugarLevel:E_SugarLevel &

   /* Outputs */
   ChangeAmount:NAT &
   GiveChange:BOOL &
   StartNewCoffee:BOOL &
   SugarLevel:E_SugarLevel &

   /* Local Variables */
   CoffeePrice:NAT &

   /* instance rule(s) */
   (Phase = out => (ChangeAmount<=CreditAmount))&
   (Phase = out => (StartNewCoffee = FALSE or MakeCoffee_CoffeeSelected = TRUE))&
   (Phase = out => (StartNewCoffee = FALSE or (CreditAmount>=CoffeePrice)))
```

Figure 8: *SelectorController* block and the clause INVARIANT of its translated B machine. The block constraints are translated into invariants pre-conditioned by *Phase = out*.

5.1.4 CLAUSE INITIALISATION

In the INITIALISATION clause of the B machine, the inputs, outputs, local variables and *Phase* variable (if it exists) are initialized with a non-deterministic substitution. All the variables are weak-initialized. Only the *Phase* variable is initialized to a specific value: In.

An example shown with the Figure 9.




```
INITIALISATION
   MakeCoffee_CoffeeSelected,
   MakeCoffee_SugarLevel,
   ChangeAmount,
   GiveChange,
   CoffeePrice,
   CreditAmount,
   SugarLevel,
   StartNewCoffee,
   Phase:(
   MakeCoffee_CoffeeSelected:BOOL &
   MakeCoffee_SugarLevel:E_SugarLevel &
   ChangeAmount:NAT &
   GiveChange:BOOL &
   CoffeePrice:NAT &
   CreditAmount:NAT &
   SugarLevel:E_SugarLevel &
   StartNewCoffee:BOOL &
   Phase=in)
```

Figure 9: *SelectorController* block and the INITIALISATION clause of its translated B machine

5.1.5 CLAUSE OPERATIONS

The OPERATIONS clause of a B machine contains the following operations:

- Schedule operation: a schedule operation is generated for all the B machine which are the translation of an instance of a SysML block (see §6.4);
- Write operations: Write operations are generated for each input of the B machine (see §6.2);
- Read operations: Read operations are generated for each output of the B machine (see §6.1);
- Phase functions: when the B machine is a translation of an instance of a SysML block which have a constraint, then *Phase_In* and *Phase_ScheduleReq* operations have to be generated (see §6.3)

5.2 B IMPLEMENTATION CLAUSES

The generated B implementation contains these B clauses:

- SEES
- IMPORTS
- CONCRETE_VARIABLES
- INITIALISATION
- OPERATIONS

5.2.1 CLAUSE SEES

The SEES clause is used to see the B machine *Def_Types* which contains the sets used in the model.

5.2.2 CLAUSE IMPORTS

The clause IMPORTS is created only when the SysML block which corresponds to the B implementation contains a part property.

The B implementation imports each B machine which corresponds to the value of the slots of its corresponding SysML instance.

For example, the block *System* has three parts: *Acquisition*, *Ctrl* and *Outputs* (see Figure 11). *CoffeeMachine_1* is an instance of this block. It has three slots which define the value of the parts *Acquisition*, *Ctrl* and *Outputs*. These value are respectively the block instances Input_Interface_1, Controller_1 and OutputInterface_1 (see Figure 10). So the block instance *CoffeeMachine_1* is translated into the B implementation CoffeeMachine_1_i which imports the B machines *OutputInterface_1*, *Input_Interface_1* and *Controller_1* (see Figure 12).

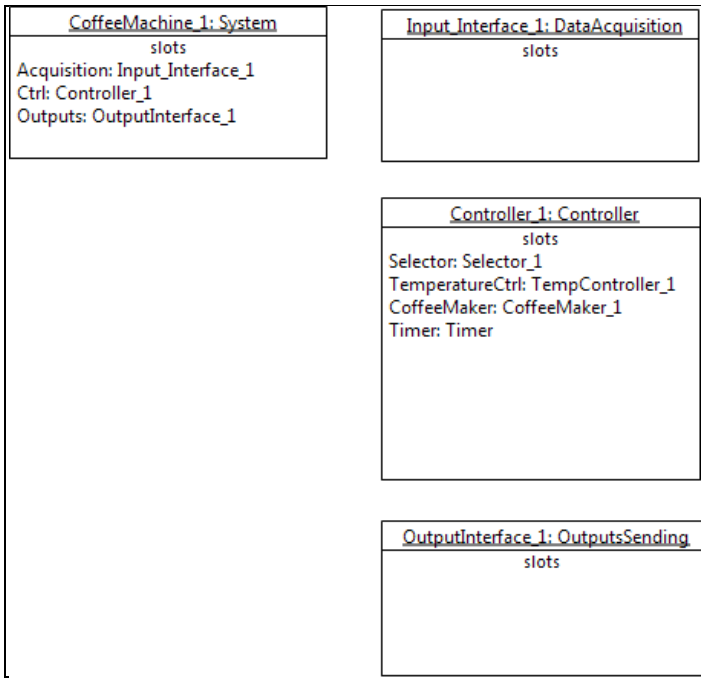


Figure 10: example of three Instance Specification nodes

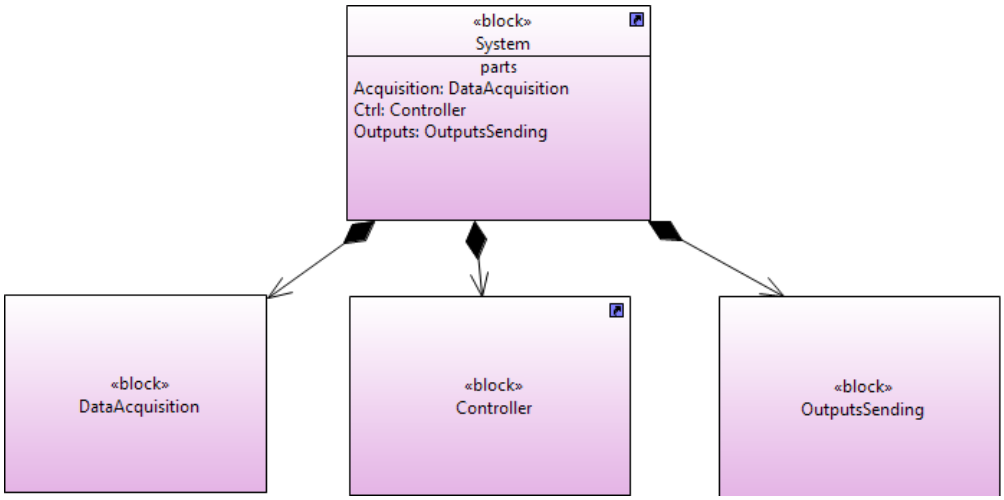


Figure 11: example of a simple model composed of four blocks

```
IMPLEMENTATION
   CoffeeMachine_1_i
REFINES
   CoffeeMachine_1
SEES
    Def_Types
IMPORTS
    part1.OutputInterface_1,
    part2.Input_Interface_1,
    part3.Controller_1
```

Figure 12: the IMPORTS clause of the B implementation of CoffeeMachine_1

5.2.3 CLAUSE CONCRETE_VARIABLES

The variables declared in the clause CONCRETE_VARIABLES are the same than the variables declared in the clause ABSTRACT_VARIABLES of the B machine (see §5.1.2).

5.2.4 CLAUSE INITIALISATION

In this clause, the variables are initialized to the default value specified in the SysML model.

Phase variable, when it exists, is initialized to the value "In".

A example is given with the Figure 13 and Figure 14. We can see that in the B generated code the variable *MakeCoffee_SugarLevel* is initialized to *level2* because the property *SugarLevel* of the data type *CoffeeCmd* has a default value set to *level2*.

5.2.5 CLAUSE OPERATIONS

Same operation is defined in §5.1.5.

Figure 13: the property *SugarLevel* of the data type *CoffeeCmd* has a default value set to *level2*

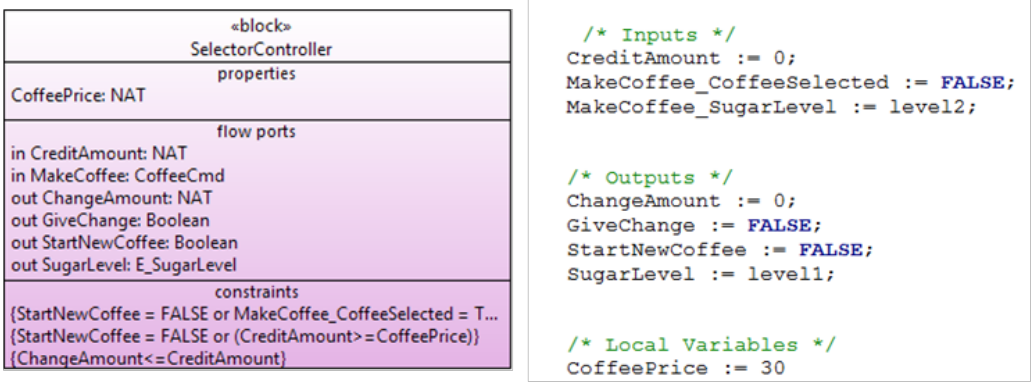


Figure 14: *SelectorController* block and the clause INITIALISATION of its translated B implementation

6 OPERATIONS DESCRIPTION

6.1 READ OPERATIONS

The *Read* operations are generated for each output of the B machine.

A *Read* operation of an output permits to access the value of the output produced by the owning B machine.

The name of a *Read* operation is the name of the output prefixed with "Read_".

A *Read* operation returns the value of the corresponding output.

Because a *Read* operation read the value of a B machine output, a *Read* operation is called at the third step of the B machine scheduling (see §3.2), when the outputs have been calculated. So, if the B machine has specific invariants which are the translation of Block constraints, then these invariants have to be verified at the end of the *Read* operations. That is why the pre-condition *Phase = out* is added to the *Read* operation when the B machine is the translation of an instance of a SysML block which have block constraint.

An example of *Read* operations is given with the Figure 15.

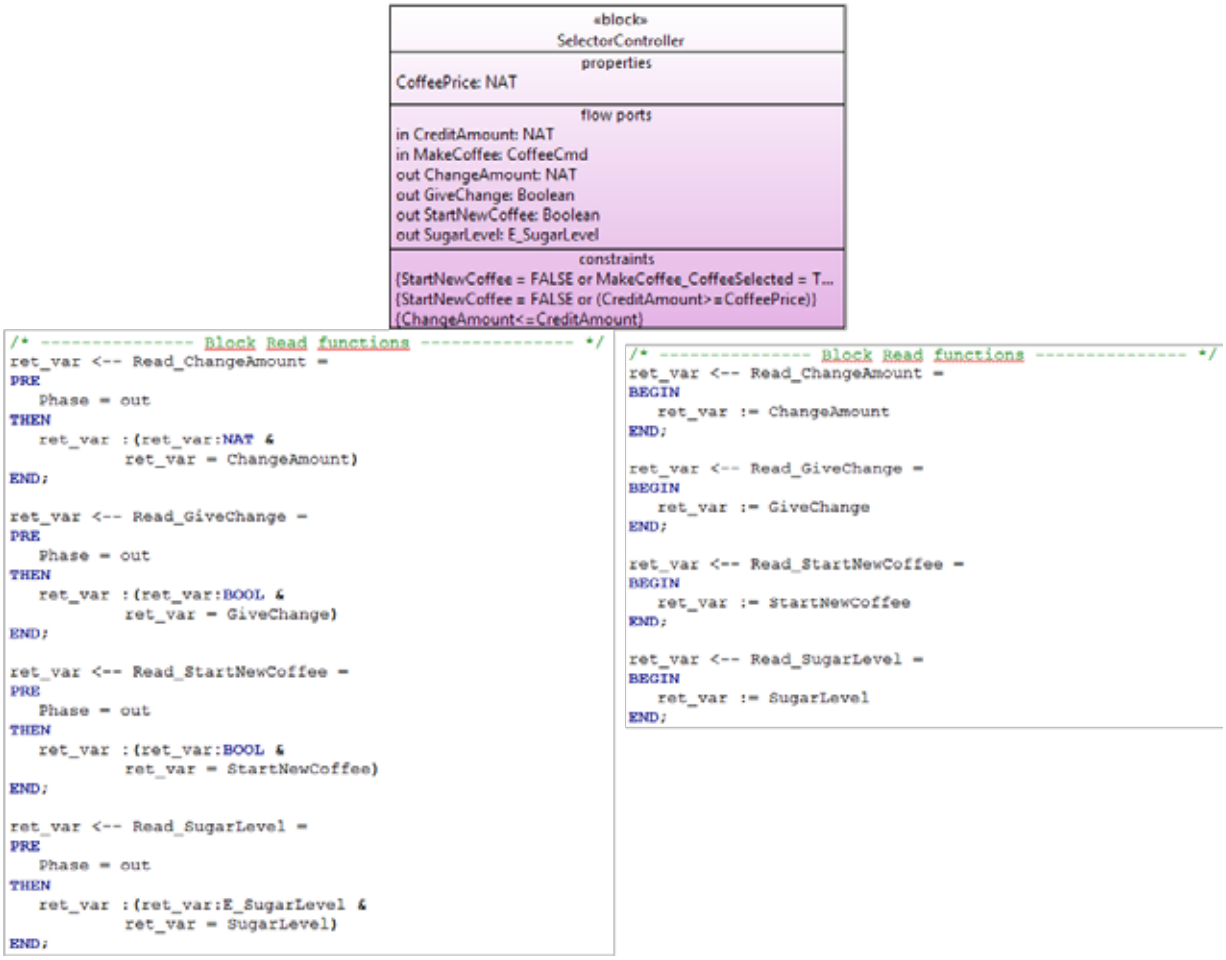


Figure 15: read operations for the outputs of the B module which corresponds to the block *SelectorController*. On the left is the B machine, on the right the B implementation.

6.2 WRITE OPERATIONS

The *Write* operations are generated for each input of the B machine.

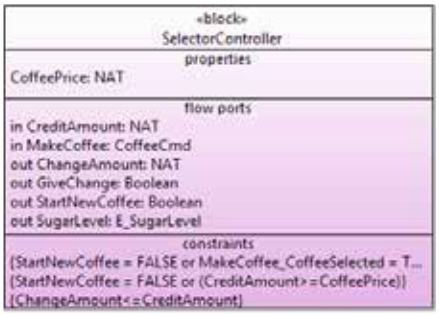
A *Write* operation of an input permits to modify the value of the input.

The name of a write operation of an input is the name of the input prefixed with "Write_".

A *Write* operation takes as argument the new value of the corresponding input.

Because a *Write* operation set the value of a B machine input, a *Write* operation is called at the first step of the B machine scheduling (see §3.2). So, if the B machine has specific invariants which are the translation of Block constraint, then these invariants do not have to be verified at the end of the *Write* operations. That is why the pre-condition *Phase = in* is added to the *Write* operation when the B machine is the translation of an instance of a SysML block which have block constraint.

An example of *Write* operations is given in Figure 16.




```
/* --------------- Block Write functions --------------- */
Write_CreditAmount(temp_var)=
PRE
   temp_var:NAT &
   Phase = in
THEN
   CreditAmount := temp_var
END;

Write_MakeCoffee_CoffeeSelected(temp_var)=
PRE
   temp_var:BOOL &
   Phase = in
THEN
   MakeCoffee_CoffeeSelected := temp_var
END;

Write_MakeCoffee_SugarLevel(temp_var)=
PRE
   temp_var:E_SugarLevel &
   Phase = in
THEN
   MakeCoffee_SugarLevel := temp_var
END;
```

```
/* --------------- Block Write functions --------------- */
Write_CreditAmount(temp_var)=
BEGIN
   CreditAmount := temp_var
END;

Write_MakeCoffee_CoffeeSelected(temp_var)=
BEGIN
   MakeCoffee_CoffeeSelected := temp_var
END;

Write_MakeCoffee_SugarLevel(temp_var)=
BEGIN
   MakeCoffee_SugarLevel := temp_var
END;
```

Figure 16: write operations for the inputs of the B module which corresponds to the block *SelectorController*. On the left is the B machine, on the right the B implementation.

6.3 PHASE OPERATIONS

The *Phase* variable is set to one of these values: In, Out, ScheduleReq.

The value of the *Phase* variable is set to "In" by the *Phase_In* operation, and to " ScheduleReq " by the *Phase_ScheduleReq* operation. The *Phase* variable is set to "Out" by the Schedule operation of the same B machine.

The *Phase_In* and *Phase_ScheduleReq* operation are called by the Schedule operation of the owning B machine. An example of the specification and of the implementation of these operation is given by the Figure 17.

The Phase operation are generated only when the B machine contains invariant coming from SysML block constraint.

```
/* --------------- Block Phase functions --------------- */
Phase_In =
BEGIN
   Phase := in
END;

Phase_ScheduleReq =
BEGIN
   Phase := scheduleReq
END
```

Figure 17: operation Phase_In and Phase_ScheduleReq. The specification is identical to the implementation of these operations.

6.4 SCHEDULE OPERATION

Every B machine which is the translation of an instance of a SysML block has a schedule operation.

6.4.1 B MACHINE WITH NO IMPORTED MACHINE

If the B machine does not import other B machine (because the B machine is the translation of an instance of a block which does not have parts), then the schedule operation is the behavior of the B machine. This schedule operation should be refined and implemented manually.

6.4.2 B MACHINE WITH IMPORTED MACHINES

If the B machine imports other B machines (because the B machine is the translation of an instance of a block which contains parts), then the schedule operation manage the exchange of data between each imported B machine. Indeed, because the imported B machine are the translation of parts, the inputs and outputs of these machines shall exchange data.

The connection between the inputs and outputs of the imported B machines are deduced from the internal block diagram (*ibd*) of the block corresponding to the translated SysML block.

For example, the B machine *Controller_1* is the translation of an instance of the SysML block *Controller* (see Figure 18) which is composed of the parts *Selector*, *TemperatureCtrl*, *CoffeeMaker* and *Timer*. The ibd of the block *Controller* shows how these parts are connected to each other and to their owning block (see Figure 19).

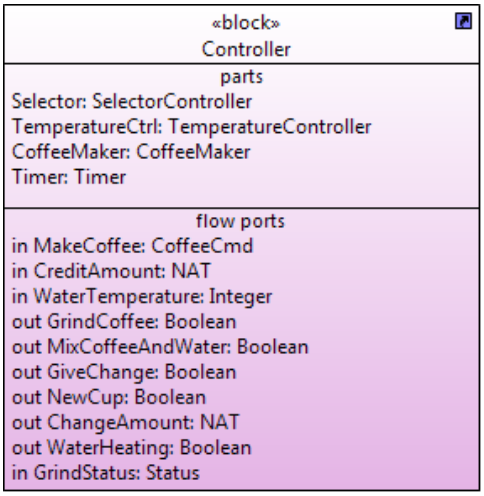


Figure 18: block *Controller*, which has four parts: *Selector*, *TemperatureCtrl*, *CoffeeMaker* and *Timer*

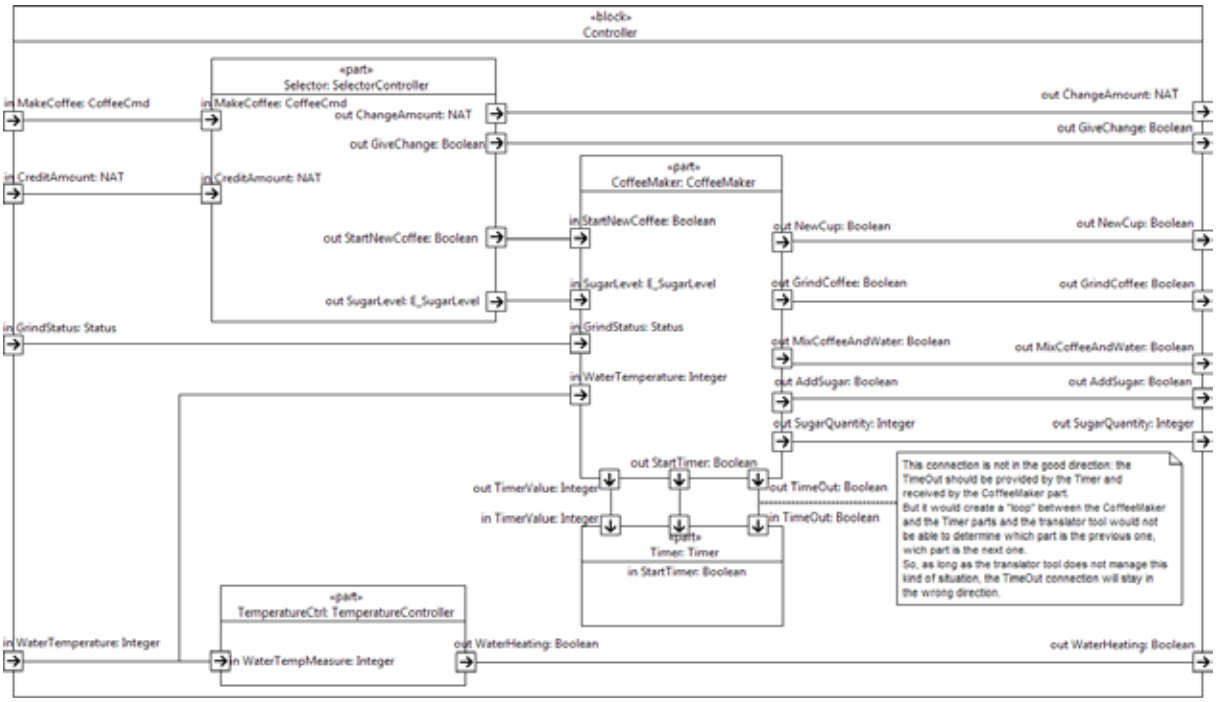


Figure 19: internal block diagram of the block Controller.

The schedule operation of a B machine which have imported machines do these task for each imported B machine:

1. Call the *Phase_In* operation of the imported B machine if the imported machine has specific constraints;
2. Set the value of the inputs of the imported B machine by calling the corresponding Write Operations;
3. Call the *Phase_ScheduleReq* operation of the imported B machine if the imported machine has specific constraint;
4. Run the behavior of the imported B machine by calling the *Schedule* operation of the imported machine;
5. Save into local variables the value of the outputs of the imported B machine by calling the corresponding *Read* operations.

All the imported B machines shall be scheduled in a particular order: first those which correspond to the parts which are connected to the inputs of their owning block, then those which are connected to the outputs of the first one.

For example, in Figure 19, parts *Selector* and *TemperatureCtrl* are the parts that shall be processed at first because their inputs are connected only to the inputs of the owning block *Controller*. Then, the part *CoffeeMaker* shall be processed because it is connected to the outputs of *Selector* and *TemperatureCtrl*. Finaly, the last part to be processed is *Timer*.

Note that the SysML to B translator tool does not manage yet the case of parts connected to each other in both directions, like the example depicted by the Figure 20. Thus, all the parts of a block shall be connected without cycle.

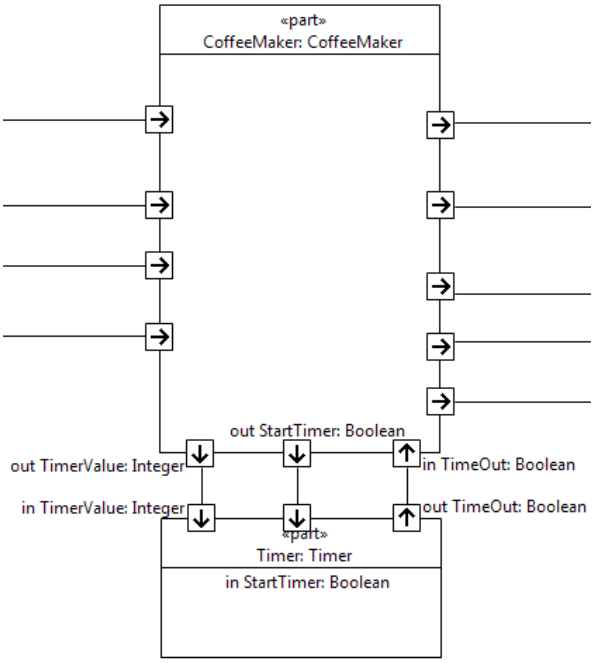


Figure 20: *CoffeeMaker* and *Timer* are connected to each other in both direction, *CoffeeMaker* transmits data to *Timer* and *Timer* transmits data to *CoffeeMaker*, which is forbidden

When all these tasks are done for all the imported machines, then the schedule operation set the value of the machine outputs with the value of the corresponding local variables.

As example, the generated code for the B machine that corresponds to an instance of the block *Controller* is shown as follow. l_AddSugar, l_GrindCoffee, l_MixCoffeeAndWater, l_NewCup, l_StartTimer, l_SugarQuantity, l_TimeOut, l_TimerValue, l_ChangeAmount, l_GiveChange, l_StartNewCoffee, l_SugarLevel, l_WaterHeating are local variables used to store the value of the outputs of the imported B machines. part1 corresponds to the machine *Timer*, part2 to the machine *CoffeeMaker*, part 3 to the machine *TemperatureCtrl* and part4 corresponds to the machine *Selector*.

```
Controller_1_schedule =
      VAR
            /* local variables used for buffering values */
            l_AddSugar,
            l_GrindCoffee,
            l_MixCoffeeAndWater,
            l_NewCup,
            l_StartTimer,
            l_SugarQuantity,
            l_TimeOut,
            l_TimerValue,
            l_ChangeAmount,
            l_GiveChange,
            l_StartNewCoffee,
            l_SugarLevel,
            l_WaterHeating
      IN
      part4.Phase_In;
      part4.Write_CreditAmount(CreditAmount);
      part4.Write_MakeCoffee_CoffeeSelected(MakeCoffee_CoffeeSelected);
      part4.Write_MakeCoffee_SugarLevel(MakeCoffee_SugarLevel);
      part4.Phase_ScheduleReq;
      part4.Selector_1_schedule;
```

```
        l_ChangeAmount <-- part4.Read_ChangeAmount;
        l_GiveChange <-- part4.Read_GiveChange;
        l_StartNewCoffee <-- part4.Read_StartNewCoffee;
        l_SugarLevel <-- part4.Read_SugarLevel;
        part3.Write_WaterTempMeasure(WaterTemperature);
        part3.TempController_1_schedule;
        l_WaterHeating <-- part3.Read_WaterHeating;
        part2.Write_GrindStatus(GrindStatus);
        part2.Write_StartNewCoffee(l_StartNewCoffee);
        part2.Write_SugarLevel(l_SugarLevel);
        part2.Write_WaterTemperature(WaterTemperature);
        part2.CoffeeMaker_1_schedule;
        l_AddSugar <-- part2.Read_AddSugar;
        l_GrindCoffee <-- part2.Read_GrindCoffee;
        l_MixCoffeeAndWater <-- part2.Read_MixCoffeeAndWater;
        l_NewCup <-- part2.Read_NewCup;
        l_StartTimer <-- part2.Read_StartTimer;
        l_SugarQuantity <-- part2.Read_SugarQuantity;
        l_TimeOut <-- part2.Read_TimeOut;
        l_TimerValue <-- part2.Read_TimerValue;
        part1.Write_StartTimer(l_StartTimer);
        part1.Write_TimeOut(l_TimeOut);
        part1.Write_TimerValue(l_TimerValue);
        part1.Timer_schedule;

        /* Update instance outputs */
        AddSugar := l_AddSugar;
        ChangeAmount := l_ChangeAmount;
        GiveChange := l_GiveChange;
        GrindCoffee := l_GrindCoffee;
        MixCoffeeAndWater := l_MixCoffeeAndWater;
        NewCup := l_NewCup;
        SugarQuantity := l_SugarQuantity;
        WaterHeating := l_WaterHeating;
        /* Update Phase */
              Phase := out
        END;
```