| | | |
|---|---|---|
| | **OPEN ETCS** | |

# openETCS SCADE Modelling Guide

| | |
|---|---|
| Author : Uwe Steinke | Date : 2015-04-20 |
| **Subject :**<br>**OpenETCS WP3**<br>- provisional - | Document Review : ☑<br>Design Review : ❒<br>Other : |

| | | |
|---|---|---|
| | | |
| | | |

| Name | Position | Company / Department |
|---|---|---|
| | | |

**Distribution to:**

| Name | Position | Company / Department |
|---|---|---|
| | | |

# TABLE OF CONTENT

# 1. OPENETCS SCADE MODELLING GUIDE

## 1.1 REFERENCES

- see SCADE Suit documentation:

## 1.2 OBJECT

This guide is intended to support the SCADE modelling task within openETCS WP 3 in addition to the SCADE documentation itself.

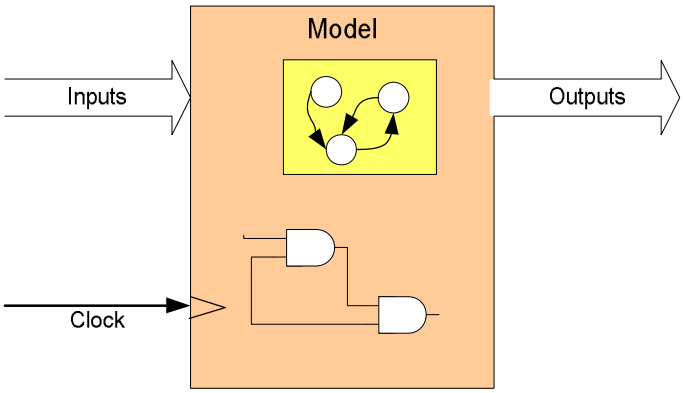
A basic knowledge of the SCADE technology is presumed as learned in a SCADE training course or by working through the tutorial delivered with SCADE.

# 2. FUNDAMENTAL CONSIDERATIONS

## 2.1 THE SCADE PARADIGM

SCADE Suite models are

- synchronously
- clocked
- data flow and state machines
- any combinations of these.




SCADE supports

- hierarchies: data flows and state machines can be nested into each other.
- parallelism: data flows and state machines are thought to be calculated in parallel.
- solving interdependencies: the execution order of model components depends on the interdependencies of states and data flows and is determined by the SCADE code generator automatically.

SCADE models are

- strictly formal
- implementations: there is no gap between the model diagrams and the implementation. SCADE diagrams are in no case only a painting.
- concrete and executable at any time

**SCADE modeling implies**

**solving the task by the means of**

**synchronously clocked data flow and state machines!**

Typically, it's not a good idea to re-implement existing solutions from imperative programming languages like C into SCADE models line by line.
Instead: **model by starting directly from the requirements**.

## 2.2 THE MODEL CLOCK

SCADE provides sophisticated clocking capabilities, but for the beginning we will focus on the basic principle.

The clock in the simple case causes a complete recalculation of the model.

SCADE assumes that the calculation takes 0 ps CPU time, but does not make any assumptions regarding the clock period.

In reality, the CPU time required will be > 0. Therefore, the clock period should be chosen so that

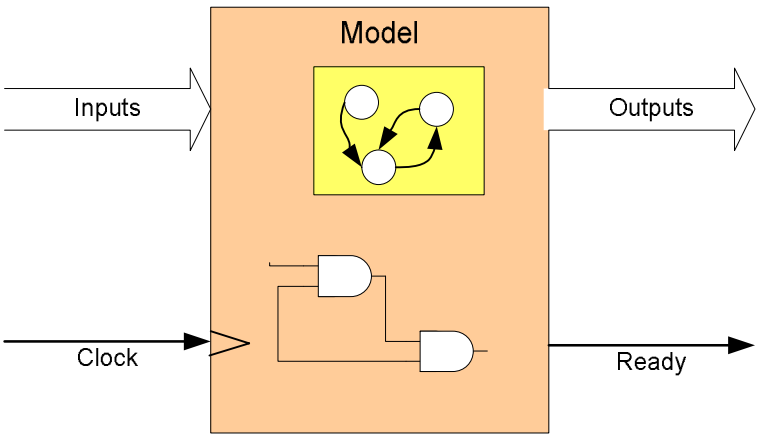
$$Clock\ period\ \geq max.CPU\ calculation\ time$$

If the function to be modelled comprises real time processing of analogous signals like digital filters, digital controllers, digital (de-)modulators, a constant equidistant clock must be chosen. In the case of operations requiring several clocks, the results will be available after these several clocks. Therefore, a constant equidistant clock may lead to long response times.

Since the ETCS onboard unit does not need real time processing of analogous signals, a variable clocking period is applicable. The principle is

**Clock the model repeatedly in bursts until the operation is finished!**

The model itself is able to inform, when an operation is finished. For this purpose, the model should provide a **"Ready" handshake output**: when set to true, the clock burst can be stopped.




A code fragment for clocking the model in this way:

```
void Model::activate() //Activate phase (s. API)
{
      setModelInputs(); // Transfer external input information to the model
      do // As often as the model wants to be clocked ...
      {   clockModel(&Model_Input, … );// Clock the model once
          distributeModelOutputs();   // Deliver the model results
      } while ($NOT$ Model.ready);
}
```

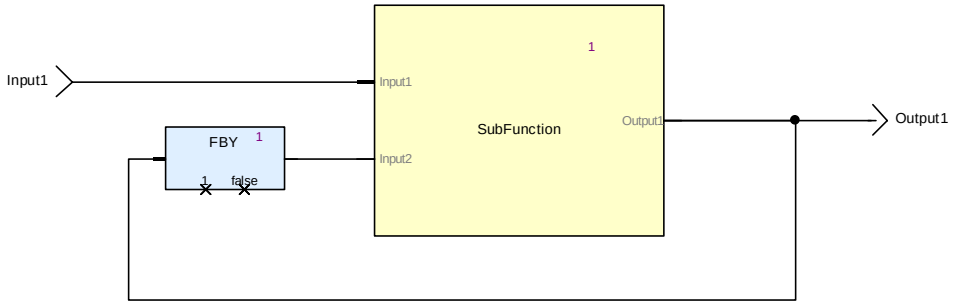
## 2.3 FEEDBACK LOOPS

If an input of a sub-function in the model is directly or indirectly connected with the output of the same sub-function, a feedback loop is created.

Feedback loops typically cause the need for more than one clock until all results are produced and tend to increase complexity. Therefore, care has to be taken on this aspect:

- **Avoid unnecessary feedback loops!**
  Feedback loops are unnecessary, if they are caused by inappropriate model structures only.
- Unavoidable feedback loops are justified by the function to be modelled itself.
- Implement unavoidable feedback loops as local as possible in the model.

  Implement the feedback loop by using the “FollowedBy” or “Last” primitives.

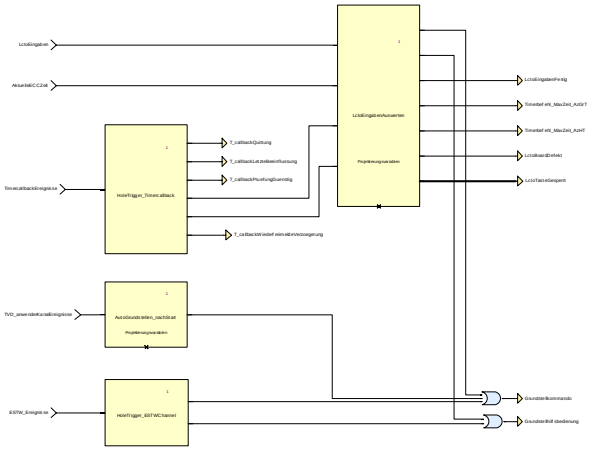



## 2.4 STATE MACHINES VS. DATA FLOWS

For more complex (sub-) functions a designs decision has to be made to create a data flow or a state machine at top level.

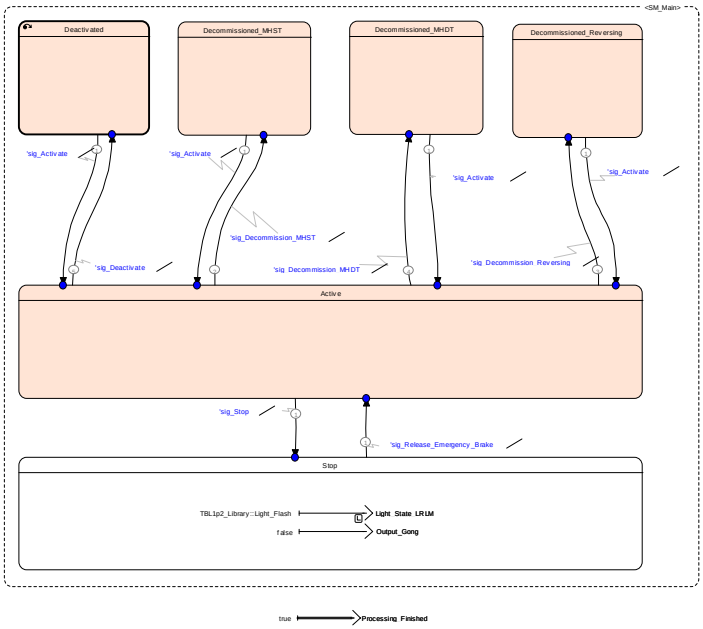
A data flow machine

- represents the function as a structure of separate sub-functions
- shows the interdependencies between sub-functions explicitly as data flows



A state machine

- represents the function as a structure of states
- interdependencies of sub-functions buried deeply in different states tend to be not obvious




In summary, choose a data flow or a state machine at top level, what is more appropriate to the function to be modelled, but be aware of the implications.

## 2.5 STAGED MODEL STRUCTURE

For many systems the sub-functions can be grouped to 3 stages:

- An input stage for pre-processing the input data
- A main calculation stage (“business logic”) in the middle
- An output stage for post-processing and preparing the output data.

# 3. OPENETCS WP3 COMMON DESIGN PRINCIPLES

The quality of the models created by WP 3 can be encouraged by relying on the following design principles.

## 3.1 ACTUAL SYSTEM TIME AND TIME STAMPS

Since the model will not be clocked with an equidistant period, the actual system time is available for the model as an input. The following rules shall apply for the model and its environment:

- The model must not rely on a fixed period between the model clocks. Timing functions based on clock cycle counting are not feasible.
- The model is allowed to use the actual system time for those calculations only that cannot be based on input event time stamps.
- The openETCS API and Basic Runtime System will keep the actual system time applied to the model as an input unchanged within the same Basic Runtime System cycle.
- The actual system time provided by the openETCS API and Basic Runtime System will be strictly monotonic increasing between two subsequent Basic Runtime System cycles (caution: these cycles are not identical with the model clocks).
- The interval between two subsequent Basic Runtime System cycles provided by the openETCS API will be ??? ms at maximum.

Practically, the actual system time will be used in the model for the implementation of time delays.

## 3.2 TIME STAMPS

The purpose of time stamps is to provide the model with the most accurate timing relations the OBU hardware can achieve and decouple it from the OBU software timing. The following rules shall apply:

- All events, originating from the openETCS API as inputs, must be equipped by the underlying API and Basic Runtime System with time stamps referring to the moment when the event arrived and consistent with the actual system time.
- The model must use these time stamps for calculating the intervals between events if needed.
- The time stamps of subsequent events applied as inputs to the model must be monotonic increasing.

## 3.3 STATIC RESOURCE ALLOCATION

For dependable software, all resources must be allocated at the system start statically. SCADE models have all resources allocated from the beginning automatically.

Therefore **all resources must be dimensioned for their worst case maximum at run time**. A maximum value (in form of an integer constant) must be set at "compile time" for

- the size of data arrays (arrays of simple types, arrays of structures, memory buffers)
- the size of arrays of functions (maximum number of loop iterations)

Several structured types of the ETCS language use iterations of substructure loops of arbitrary lengths; an actual iterator value then indicates the actual length of the loop. Since such dynamic allocations are not suitable in the model, they have to be replaced by static structure arrays of a constant maximum length.

## 3.4 MARKING ARRAY ELEMENTS AS VALID

Since all resources must be allocated statically and dimensioned with maximum capacity required at runtime, there at any point in time will be some array elements filled with valid data while the rest is "empty". This requires a method to mark each array element as filled with valid data or empty.

Therefore, all data arrays that might not be completely filled with valid data all the time, must equip each of the array elements with a Boolean "valid" flag.

Then, in the model**, an array element marked with "valid = false" must not be evaluated**.

## 3.5 MARKING EVENTS AS PRESENT

The model operates on data flows. Data flows are present all the time.

The data the openETCS OBU receives from the outside world, have mainly event character like messages from a balise group, messages from the RBC etc.. Contrary to continuous data flows, events have a transient character. They are present for a discrete moment in time and then gone.

For the model, the transient events have to be converted into a steady data flow. This can be achieved in a similar way as with array elements. A Boolean "present" flag can be used to mark an event as present. Therefore, all event data types must be equipped with a "present" mark.

Then, in the model**, an event marked with "present = false" must not be evaluated**

## 3.6 INPUT EVENTS

Most of the input data applied to the model are received from the openETCS API as messages. The input data shall be grouped into events, so that each event with its information contained can be evaluated by the model separately.

**Input information that can be evaluated separately shall be applied to the model in separate events and sequentially.**

In the simplest case a message received from the API form exactly one input event. But it is also possible, that

- One message from the API has to be split into several input events.
- The information from more than one message from the API has to be combined to one input event.

A reasonable set of input events has to be figured out while analyzing and modelling the OBU functions.

It is strongly recommended, to **apply at most one event to the model within the same clock**. This prevents from the need of processing different potential conflicting events in concurrency. The input event should be kept present until the models “ready” handshake output is true. **The next input event is allowed only after the model has set its "ready" output**.

This proceeding is recommended for the matter of complexity, understandability and testing.

Be aware: the model is able to deal with several events in parallel, the modeller often not.

## 3.7 OUTPUT EVENTS

The model produces its outputs in a similar form of events as specified for the input events. A “present” marks an output event as present. The model

- will be able to mark more than one output event as present within the same clock
- will be able to set the “present” flag for clock with its “ready” output set to false
- must not set the “present” flag for the same output event for longer than one clock

Therefore, **the model outputs must be scanned for events to be delivered to the openETCS API after every clock**.

This proceeding enables the model to generate output events in a defined order.

## 3.8 MEMORIES, GLOBAL DATA AND GLOBAL STORAGES

The concept of global data is not supported by SCADE, because it would be useless.

If absolutely wanted, a central data storage can be implemented in a model, where all functions store their results and get their inputs from. Since the inputs and outputs of all functions will be coupled via the central data storage, it will create one big feedback loop for all functions and causes to handle feedback all over the model, even if it is not necessary.

It’s to your mind if this idea creates a lot of fun for you or completely spoils the model.

Instead:

**Store data in dedicated memories within the sub-functions as local as possible.**

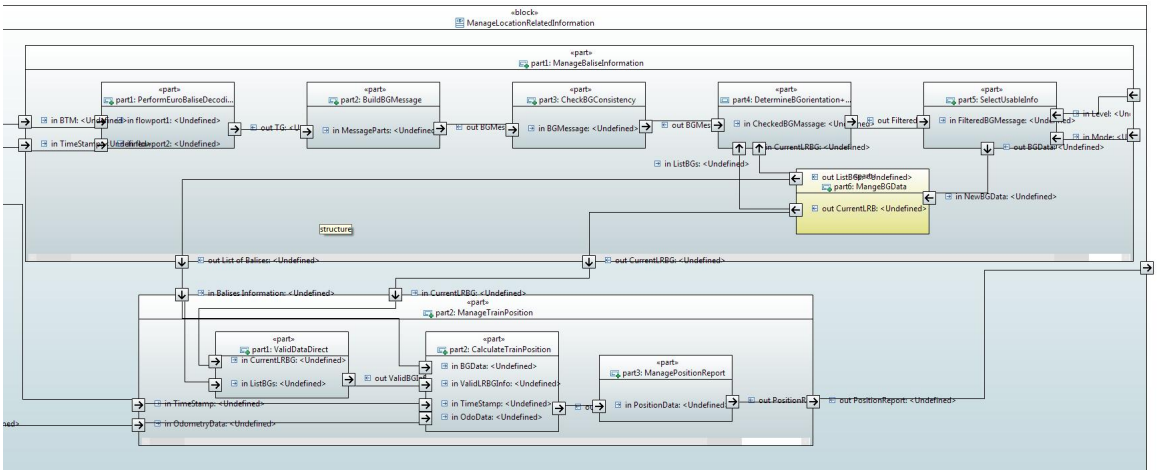
# 4. How to Model in Practice

This chapter will introduce the modelling environment for openETCS and give practical hints for modelling.

## 4.1 WP3 SCADE Modelling Repository

The repository for all modelling activities on github is located at https://github.com/openETCS/modeling/tree/master/model/Scade

The OBU kernel functionality that should be modelled first is given by the SysML model at https://github.com/openETCS/modeling/tree/master/model/sysml/WP3-Initial-Architecture/ManageLocationRelatedInformation and is presented in the following diagram.




The directory structure at https://github.com/openETCS/modeling/tree/master/model/Scade reflects the SysML model structure. Each of the SysML blocks represents a subfunction and has been assigned to a directory in the file structure.

The reason of a separate directory for every block is to enable collaboration on github. For each of the function blocks a modeller can be made responsible; he then is allowed exclusively to modify the files in the appropriate directory.

Each of the directories comprises a separate SCADE project, prepared especially for the dedicated function.

Additional files and directories house common resources like type libraries or are intended for the stepwise integration of functions.

```
+---Scade
|  +---ScadeTemplates                          SCADE project and configuration template: Do not modifiy!
|  \---System
|      +---ETCS_Language
|      |  +---S026_7                           Subset 26-7 type library (ETCS Language): Do not modifiy!
|      |  \---S026_8                           Subset 26-8 type library (ETCS Language): Do not modifiy!
|      \---ObuFunctions
|         +---ManageLocationRelatedInformation
|         |  +---BaliseGroup
|         |  |  +---BG_Types                   Type library for balise group subfunctions: to be shared
|         |  |  +---BuildBGMessage                        Subfunction to be modelled
|         |  |  +---CheckBGConsistency                    Subfunction to be modelled
|         |  |  +---DetermineBG_Orientation_and_LRBG      Subfunction to be modelled
|         |  |  +---ManageBaliseGroupData                 Subfunction to be modelled
|         |  |  +---ManageBaliseInfomation_Integration    For the integration of BG subfunctions
|         |  |  +---PerformEuroBaliseDecoding             Subfunction to be modelled
|         |  |  \---SelectUsableInfo                      Subfunction to be modelled
|         |  +---MLRI_Integration                         For the integration of MLRI subfunctions
|         |  +---MLRI_Types                 Type library for ManageLocationRelatedInformation: to be shared
|         |  \---TrainPosition
|         |     +---CalculateTrainPosition             Subfunction to be modelled
|         |     +---ProvidePositionReport              Subfunction to be modelled
|         |     +---TrainPosition_Integration          For the integration of TrainPosition subfunctions
|         |     +---TrainPosition_Types                Type library for TrainPosition: to be shared
|         |     \---ValidateDataDirection             Subfunction to be modelled
|         \---Obu_BasicTypes                    Type library for all basic types of the OBU: to be shared
```

## 4.2 FILE TYPES RELEVANT FOR VERSION CONTROL

While modelling, the modelling directories will be populated with different file types. The following file types are relevant for configuration management and must be uploaded to github:

- *.etp: SCADE project file
- *.xscade, *.scade: SCADE source code
- *.ann: SCADE annotations
- *.rqtf, *.trace, *.rqtfimage: requirements gateway files (for linking to requirements)
- *.types: openETCS standard type definition file for requirements gateway

## 4.3 MANUAL RESERVED CHECKOUT PROCEDURE FOR GITHUB

A primitive workaround until a better solution is provided by the openETCS tool chain could be based on **a manual file lock procedure**:

1. Before starting with a modification, check that no one else actually has locked the appropriate directory.
2. If no other person has reserved the directory, upload a file with the name "**LockedBy_YourName**" into the directory on github.
3. If the upload of the lock file succeeded and no one else has uploaded his lock file in parallel, pull the directory to you local computer.
4. Perform your modelling work and modify the files in directory.
5. If you're done and your model is in a syntactically and semantically clean state, delete the lock file on your computer and upload the directory to github.

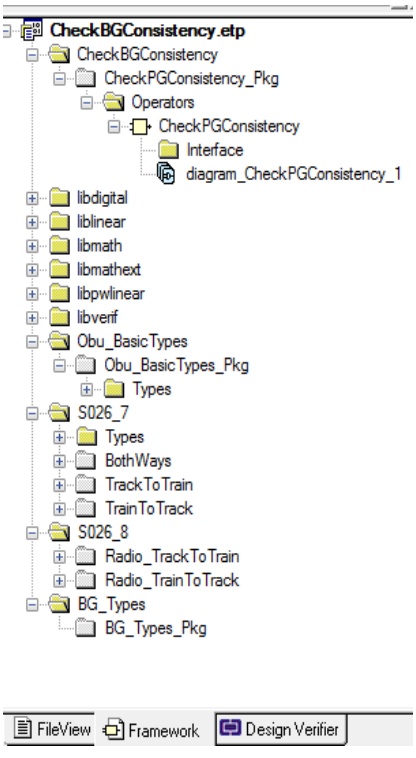
This workaround of course requires every committer to comply with this procedure.

## 4.4 OPEN A PREPARED SCADE PROJECT

After pulling the directory structure from https://github.com/openETCS/modeling/tree/master/model/Scade to your computer, one of the projects can be opened:

- Start the SCADE IDE
- With the windows explorer, go to the appropriate modelling directory and drag & drop the *.etp file to the Framework, Fileview or editor window of the SCADE IDE.

The projects prepared for modelling are of a similar structure as shown here:




Project file

Package for model

Folder for the sub-functions of the model. Create them here.

Main function of this model

Libraries that come with SCADE

Type library for all basic types of the OBU

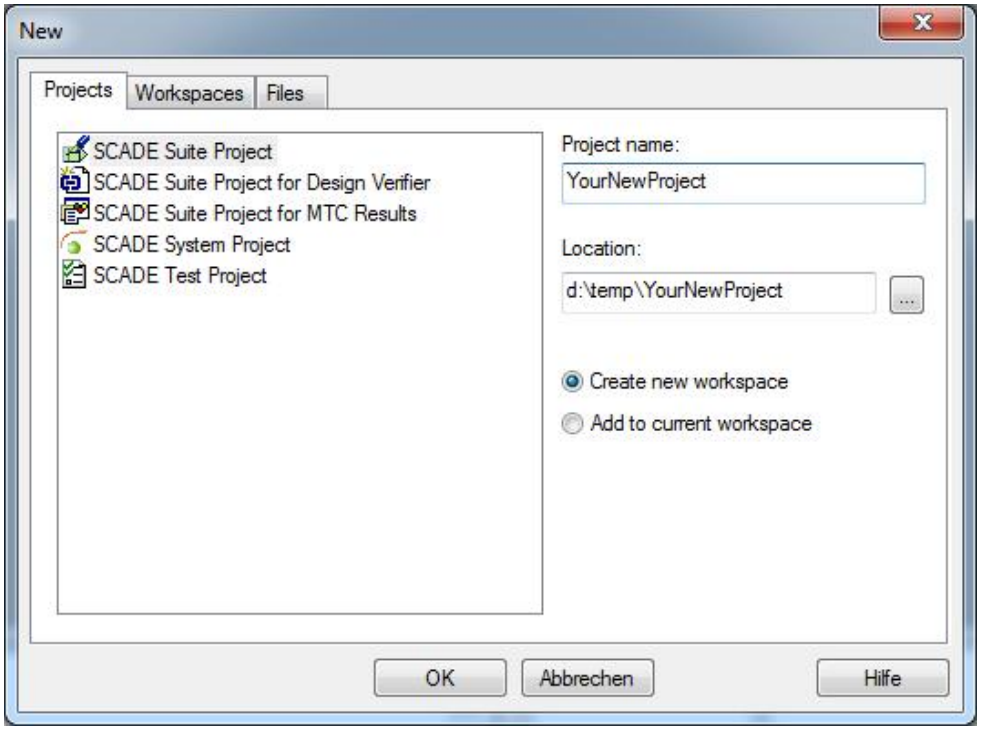
Subset 26-7 und -8 type libraries (ETCS Language)

Type library for balise group sub-functions

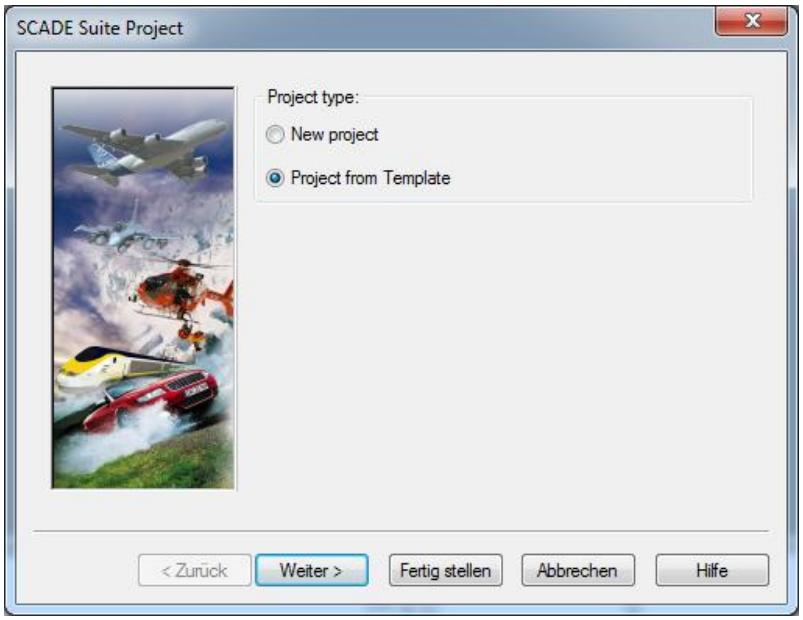
## 4.5 CREATE A NEW OPENETCS SCADE PROJECT

New openETCS SCADE projects should be created when a new component or function has to be modelled. The location should be chosen at https://github.com/openETCS/modeling/tree/master/model/Scade\System\ObuFunctions or in one suitable subdirectory there. A new project must have its own directory.

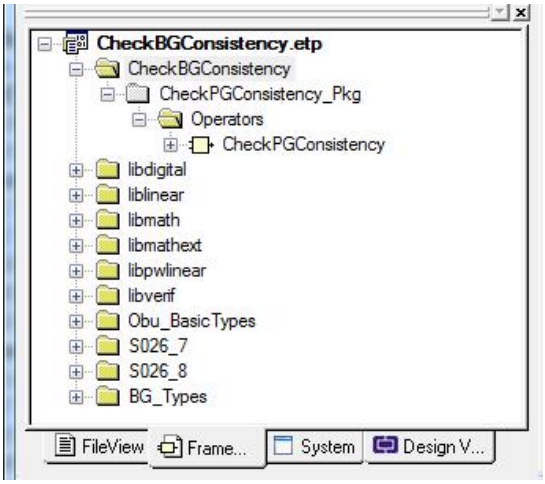
With the SCADE IDE, select. File → New → Projects. Enter the project name in the following dialog and take the “Create a new workspace” option.




In the subsequent dialog, check “Project from Template” and then in the file select box pick the template file https://github.com/openETCS/modeling/tree/master/model/Scade\ScadeTemplates\openETCS_Scade_Template.ett .



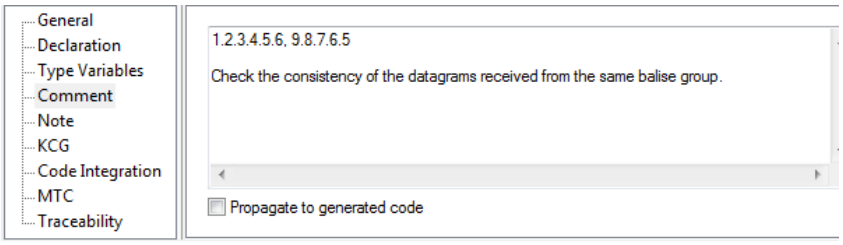
Then, in the framework view, create a new package "*_Pkg" and a main operator for the new function.




## 4.6 SETTING REQUIREMENT REFERENCES

SCADE provides a requirements management gateway (Reqtify) for linking requirements to model artefacts. Actually, the openETCS tool chain has not set up an interface to this gateway. Until this becomes available, a workaround is required to decorate the model artefacts with requirement references.

Workaround: **Add the requirement IDs to the comment fields of the model artefacts!**



## 4.7 KEEP THE MODEL CLEAN

While modelling it is advisable to keep the model syntactically and semantically clean all the time. The benefits of proceeding so are

- Dealing with very few error messages is much easier than with a lot.
- Fixing few bugs is easier and much more efficient.
- Code generation and simulation is always possible.

The SCADE “Check” function is intended for this purpose.

Keep your model clean:

- “**Check” the model quite often**
- “Generate” code accasionally.


## 4.8 LIMIT THE DIAGRAM SIZE

Each single SCADE function can be split into more than one diagram. It is a good idea to make use of several diagrams for more complex functions. Large diagrams are unhandy and at the end, a large diagram printed as part of an A4 document will become unreadable.

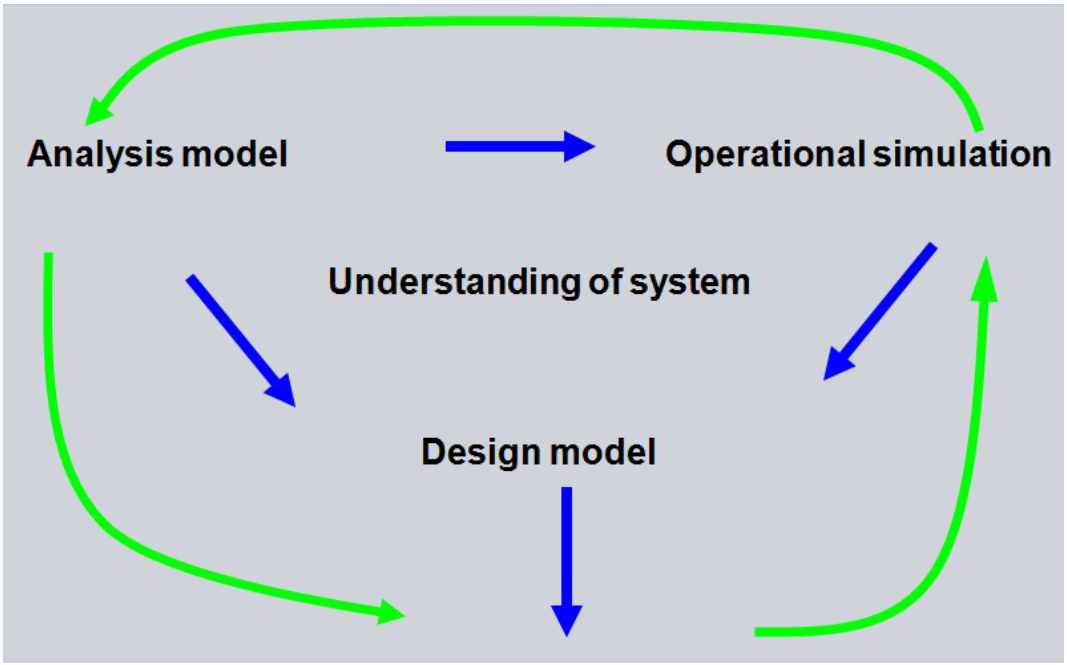
**Limit the diagram size to a printable format!**

Instead of splitting a function into diagrams, creating a new sub-function might be often more appropriate.

## 4.9 ANALYZE REQUIREMENTS BY MODELLING

While reading and analyzing the requirements of a function, model the referring parts of the function in parallel. This typically is more efficient than writing a further more document: modelled functions can be simulated and tested, documents not.

Understanding a system is an iterative process that can be enhanced by modelling and execution of the model.



### 4.10 DON'T OPTIMIZE

For the phase we are at the moment, it is more important to focus on clarity and understandability than on saving resources like CPU time or memory.

**Don't try to optimize CPU resources on model level.**

**First make it right, then make it fast!**