Use of Model Driven Code Generation on the ASIM Project

Steen Palm
Terma A/S
sup@terma.com

Terma at a Glance

Terma covers four business segments:

Aero: Development and production of advanced structures for defense and non-defense aircraft and helicopters.

Space: Mission-critical electronics, software, and services for space applications.

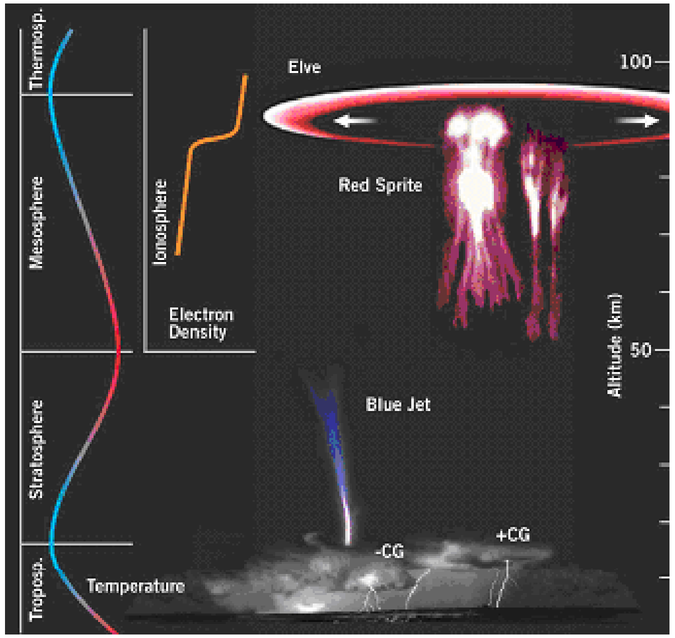
Defense: Network and tactical systems, airborne and naval self-protection systems, and electronics manufacturing services for mission-critical defense and security applications.

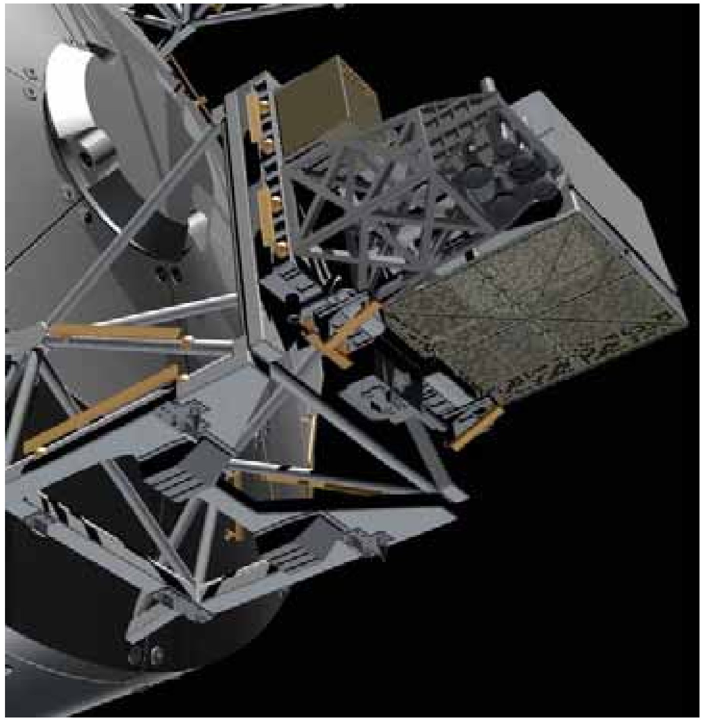
Security & Surveillance: Advanced security systems and surveillance radar solutions for ports, airports, coast guards, first responders, emergency organizations, and defense forces.

Terma is owned by the Thomas B. Thrige Foundation

The ASIM Observatory

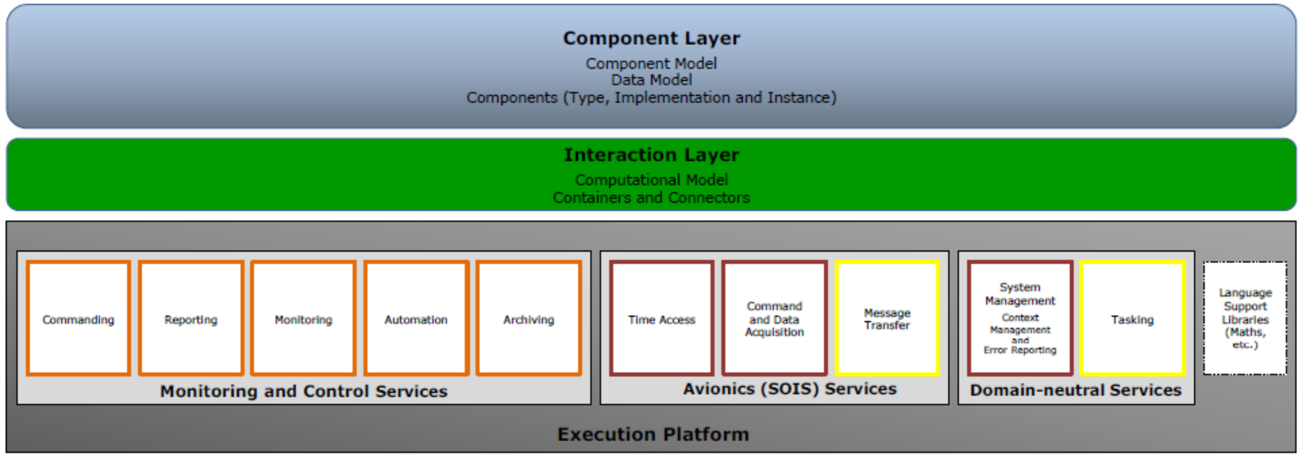
ASIM is an observatory to be placed on the exterior of the ISS. It will measure high altitude lightning that is discharged from thunderclouds, stretching up to the ionosphere at altitudes of 90-100 km. These formations of lightning are known as "red sprites", "blue jets", and "elves". In addition, the ASIM project will study the discharges observed in the form of energetic bursts of X-rays and gamma rays, likewise discharged from violent lightning

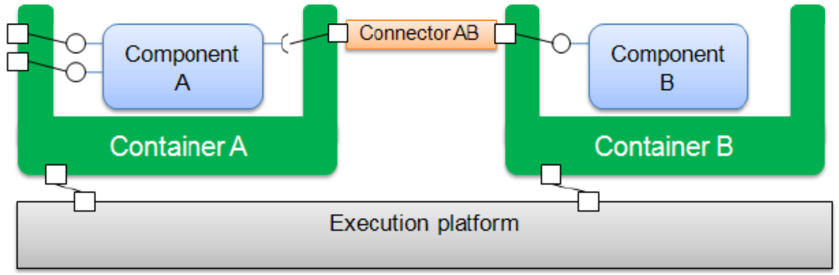




SAVOIR

- SAVOIR (Space AVionics Open Interface aRchitecture) is an initiative to improve the way that the European space community builds avionics subsystems.
- SAVOIR FAIRE (SAVOIR Fair Architecture and Interface Reference Elaboration) is an industrial working group, who is working towards the definition of a reference architecture for software on-board spacecraft platforms.





ASSERT

When using ASSERT, a platform independent component model is constructed containing the following views:

- ***The Interface View*: Characterizes the provided and required services of components and declares their intended concurrent behaviour. In this view, a provided service can be specified to execute, for example, as a cyclic operation.**
- ***The Functional View*: Specifies the functional services provided by components and expresses their sequential behaviour.**
- ***The Deployment View*: Specifies the physical architecture of the system and the way in which components are to be deployed on it.**

From the platform independent specification, a platform dependent model is automatically constructed (by vertical transformation) containing the single view:

- ***The Concurrency View:* Specifies the concurrent architecture of the system needed to implement the platform independent specification of it; this view is designed to be compliant with the Ravenscar Computational Model by construction.**

The Terma Modeling Tool Chain

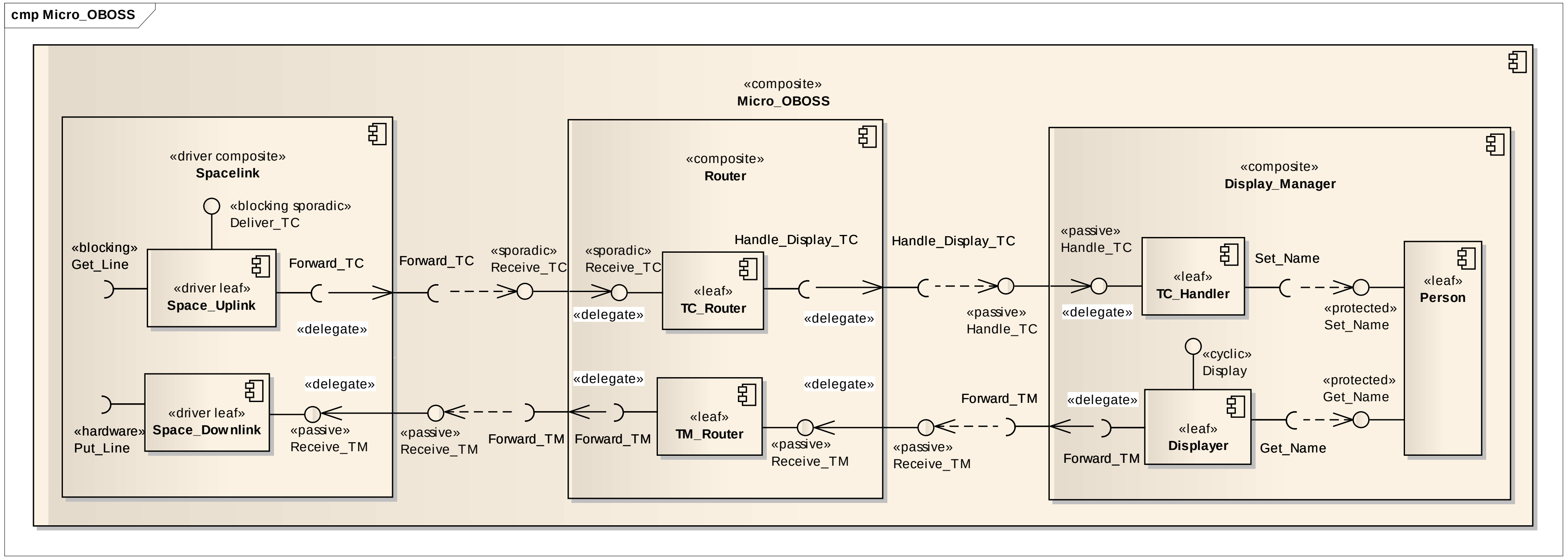
Terma has developed its own modeling tool chain based on SAVOIR/ASSERT principles:

- The Interface View is expressed as a component model in UML . In the model interfaces provided by components are decorated with UML stereotypes like ‹‹protected›› and ‹‹cyclic››

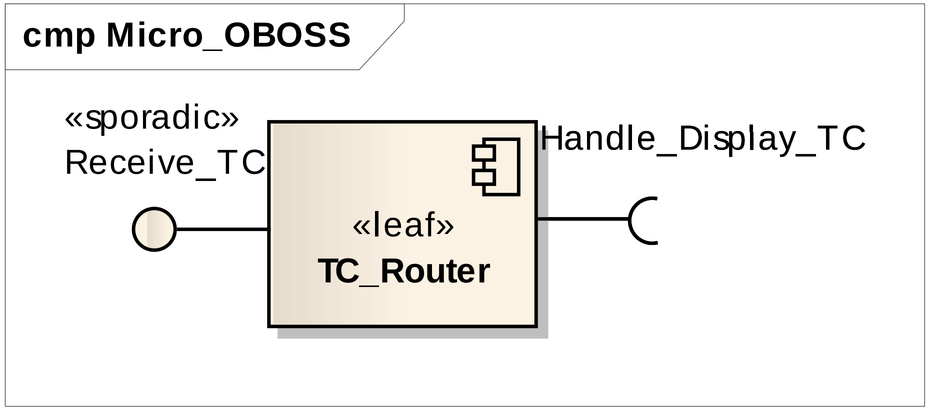
  Compared to ASSERT, the Interface View has been extended with stereotypes supporting interaction with device drivers (e.g. ‹‹interrupt sporadic››)

- The Functional View is implemented as passive Ada packages
- The Deployment View is not supported
- The Concurrency View is automatically generated from the Interface View and the Functional View. It consists of Ada tasks and protected objects, which will call the passive Ada subprograms defined in the Functional View

Interface View (Micro OBOSS Example)



Functional View



```
with Micro_OBOSS.Router.TC_Router_RI;
package body Micro_OBOSS.Router.TC_Router is

   package RI renames
      Micro_OBOSS.Router.TC_Router_RI;

   procedure Receive_TC(TC : in Data_View.TC_T) is
   begin
      case TC.Destination is
         when Data_View.DISPLAYER =>
            RI.Handle_Display_TC(TC);
         when others =>
            null;
      end case;
   end Receive_TC;

end Micro_OBOSS.Router.TC_Router;
```

```
with Data_View;
package Micro_OBOSS.Router.TC_Router is

   procedure Receive_TC(TC : in Data_View.TC_T);

end Micro_OBOSS.Router.TC_Router;
```

```
with Data_View;
package Micro_OBOSS.Router.TC_Router_RI is

   procedure Handle_Display_TC(TC : in Data_View.TC_T);

end Micro_OBOSS.Router.TC_Router_RI;
```

Concurrency View

```
with Data_View;
package CV.Micro_OBOSS.Router.TC_Router is

  procedure Receive_TC(TC : in Data_View.TC_T);

end CV.Micro_OBOSS.Router.TC_Router;
```

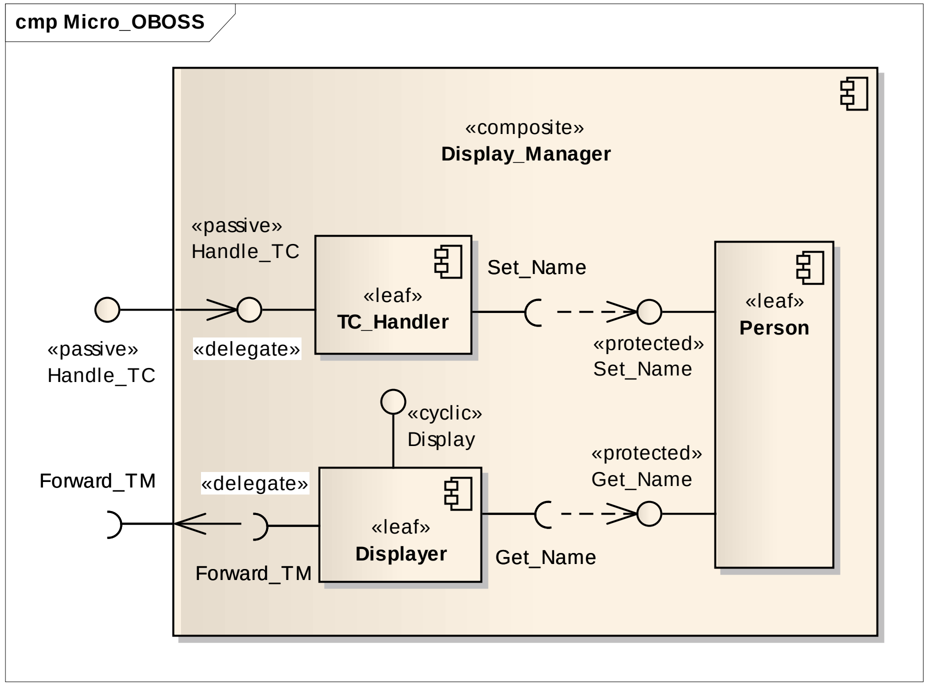
```
with Micro_OBOSS.Router.TC_Router;
with Tasking_Properties;
with Sporadic_Task;
package body CV.Micro_OBOSS.Router.TC_Router is

  package Spo_Task is new Sporadic_Task
    (Deadline    =>
         Tasking_Properties.
         CV_Micro_OBOSS_Router_TC_Router_Receive_TC.Deadline,
     ...
     Parameter_T =>
         Data_View.TC_T,
     Operation   =>
         Standard.Micro_OBOSS.Router.TC_Router.Receive_TC);

  procedure Receive_TC(TC : in Data_View.TC_T) is
  begin
    if not Spo_Task.Request_Sporadic_Operation(TC) then
      -- protected queue full
      ...
    end if;
  end Receive_TC;

end CV.Micro_OBOSS.Router.TC_Router;
```

Concurrency View (cont)



```
with CV.Micro_OBOSS.Display_Manager.TC_Handler;
package body CV.Micro_OBOSS.Display_Manager is

  procedure Handle_TC(TC : in Data_View.TC_T) renames
    CV.Micro_OBOSS.Display_Manager.TC_Handler.Handle_TC;

end CV.Micro_OBOSS.Display_Manager;
```

```
with Micro_OBOSS.Display_Manager_RI;
with CV.Micro_OBOSS.Display_Manager.Person;
package body Micro_OBOSS.Display_Manager.Displayer_RI is

  procedure Forward_TM(TM : in Data_View.TM_T) renames
    Micro_OBOSS.Display_Manager_RI.Forward_TM;

  function Get_Name return Data_View.Name_T renames
    CV.Micro_OBOSS.Display_Manager.Person.Get_Name;

end Micro_OBOSS.Display_Manager.Displayer_RI;
```

Extension of ASSERT

The Terma tool chain supports:

- ❑ **Modeling of Interrupt service routines via the stereotype ‹‹interrupt sporadic››**
- ❑ **Watch-dog protocol for checking aliveness of tasks**
- ❑ **Common components shared among different designs**

Current Status of the Development

Trace matrices: Trace information has been automatically extracted from the component models as trace matrices, which have been checked for completeness and included in the design documentation.

Validation of design: A validation tool has been developed that validates:

- ❑ *Compliance to rules*, e.g. that all interfaces provided by a leaf component have a description
- ❑ *Completeness of design*, e.g. that all required interfaces are connected to provided interfaces.
- ❑ *Consistency of design*, e.g. that a provided interface is not connected to another provided interface.

Schedulability analysis: Based on the Interface Views and the Functional Views of the instrument designs, schedulability analyses have been performed showing that all Ada tasks in the instrument software will meet their deadlines.