TOOLCHAIN TO CONNECT EDS AND TASTE

Model Based Space Systems and Software Engineering - MBSE2022

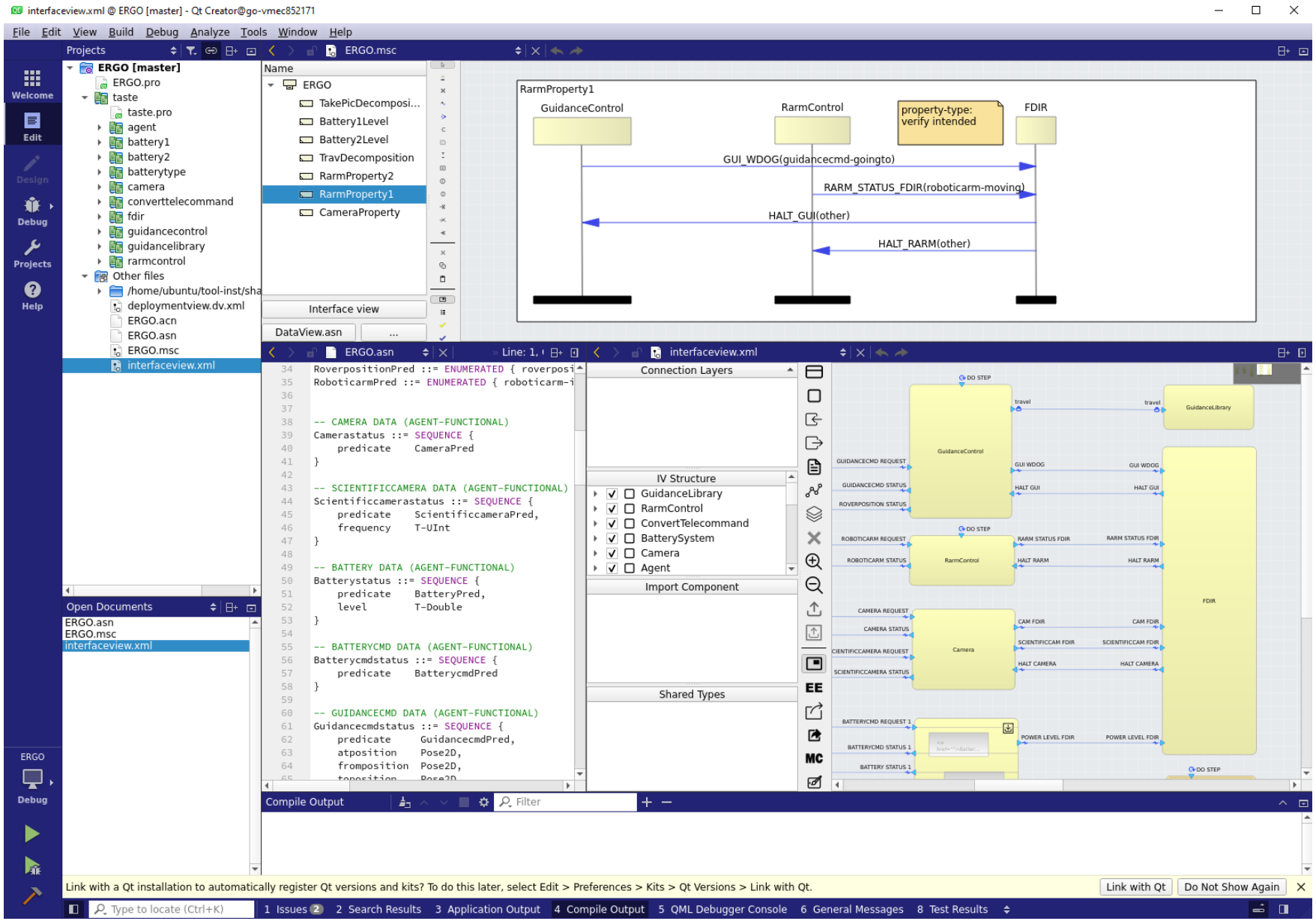
2022

Agenda

- TASTE
- EDS
- Project goals
- EDS <-> TASTE translation
  - EDS <-> ASN.1
  - EDS <-> InterfaceView
  - EDS -> SDL
- TASTE integration
- HWAS
- Demonstration use case
- Lessons Learned
- Wishlist

TASTE

- "A tool-chain targeting heterogeneous embedded systems, using an MBSE development approach"
  - Includes graphical and text editors, build management, compilers, code generators, runtimes, communication drivers....
- Developed by ESA and based on standards:
  - AADL – logical and physical architecture description
    - Since the move to SpaceCreator, AADL is an intermediate representation, and XML is used for persistence
  - ASN.1 (ACN) – data model specification (with additional encoding information)
  - SDL – for high-level behaviour description
  - But can also integrate other technologies – e.g., C, Ada, C++, Simulink... and now EDS
- Can generate executables for various hardware targets
  - e.g., x86 (Linux), GR712RC (RTEMS) or SAMV71 (FreeRTOS)...
- It is open-source – free to use by companies and individuals
- Please visit https://taste.tools/



*sourced from https://taste.tuxfamily.org/wiki/index.php?title=Overview

EDS

- CCSDS 876-0-B-1 Spacecraft Onboard Interface Services Electronic Data Sheets
  - Referred simply as EDS within the scope of this activity
- EDS is an XML specification describing:
  - Data types
    - e.g., Integers, Containers, Arrays...
  - Interfaces
    - Commands and parameters, both synchronous and asynchronous
  - Behaviours
    - State machines
- EDS is intended to replace traditional Interface Control Documents and proprietary data sheets with machine readable interface specifications
- The format is suitable for describing devices and applications in terms of their data interfaces and internal behaviour

EDS

- There is a significant overlap between TASTE and EDS model scopes
- Translation between TASTE and EDS models can:
  - Provide indirect authoring tools for EDS
    - TASTE includes SpaceCreator, which contains editors for ASN.1 and InterfaceView
  - Provide code generation for EDS
    - TASTE includes asn1scc, kazoo and OpenGEODE to generate code from ASN.1, InterfaceView and SDL respectively
  - Strengthen TASTE function as a technology hub, providing interoperability with EDS

Project goals

- The main objective of the activity is to:

  "provide a working toolset for EDS by extending TASTE with import/export functionalities, allowing to make actual use of EDS in real-life applications"

- In order to achieve the objective, the following tasks were foreseen:
  - Creation of EDS to ASN.1, InterfaceView and SDL translator
  - Creation of ASN.1 and InterfaceView to EDS translator
  - Integration of the tooling within SpaceCreator
  - Creation of a TASTE runtime for SAMV71, compatible with EDS-based components
  - Creation of a demonstration application running on a microcontroller connected to sensors and actuators

EDS to ASN.1 translation example

EDS

```
<IntegerDataType name="SignedInteger16Big">
  <LongDescription>A simple 16-bit signed integer</LongDescription>
  <IntegerDataEncoding encoding="twosComplement" byteOrder="bigEndian" sizeInBits="16" />
  <Range>
    <MinMaxRange rangeType="inclusiveMinInclusiveMax" min="-32768" max="32767" />
  </Range>
</IntegerDataType>
```

ASN.1

```
-- A simple 16-bit signed integer
SignedInteger16Big ::= INTEGER(-32768 .. 32767)
```

ACN

```
SignedInteger16Big [size 16, encoding twos-complement, endianness big]
```

EDS to InterfaceView translation example

```
<Interface name="SyncInterface">
  <CommandSet>
    <Command name="SyncNoArgsCommand" mode="sync" />
    <Command name="SyncInCommand" mode="sync">
      <Argument name="SyncInArgument1" type="MyInteger" mode="in" />
      <Argument name="SyncInArgument2" type="MyInteger" mode="in" />
    </Command>
    <Command name="SyncOutCommand" mode="sync">
      <Argument name="SyncOutArgument1" type="MyInteger" mode="out" />
      <Argument name="SyncOutArgument2" type="MyInteger" mode="out" />
    </Command>
  </CommandSet>
</Interface>
```

EDS

InterfaceView

```
<Provided_Interface name="SyncInterfacePI_SyncNoArgsCommand_Pi" kind="Protected" layer="default">
</Provided_Interface>

<Provided_Interface name="SyncInterfacePI_SyncInCommand_Pi" kind="Protected" layer="default">
    <Input_Parameter name="SyncInArgument1" type="MyInteger" encoding="ACN"/>
    <Input_Parameter name="SyncInArgument2" type="MyInteger" encoding="ACN"/>
</Provided_Interface>

<Provided_Interface name="SyncInterfacePI_SyncOutCommand_Pi" kind="Protected" layer="default">
    <Output_Parameter name="SyncOutArgument1" type="MyInteger" encoding="ACN"/>
    <Output_Parameter name="SyncOutArgument2" type="MyInteger" encoding="ACN"/>
</Provided_Interface>
```

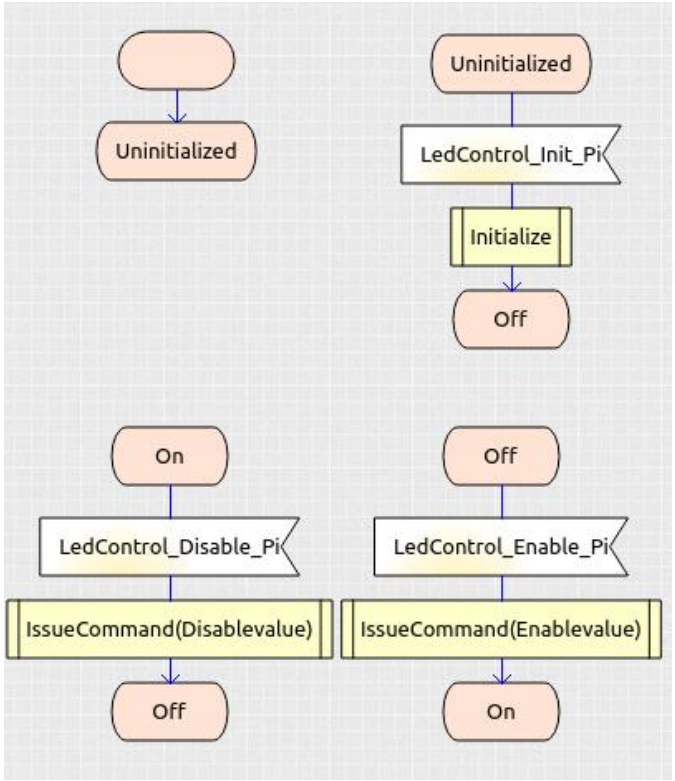
EDS to SDL translation example

EDS

```
<StateMachine name="VirtualLED">
  <EntryState name="Uninitialized"></EntryState>
  <State name="Off" />
  <State name="On" />
  <Transition fromState="Uninitialized" toState="Off" name="Initialize">
    <OnCommandPrimitive command="Init" interface="LedControl" />
    <Do activity="Initialize" />
  </Transition>
  <Transition fromState="Off" toState="On" name="ToggleOn">
    <OnCommandPrimitive command="Enable" interface="LedControl" />
    <Do activity="IssueCommand">
      <ArgumentValue name="value">
        <VariableRef variableRef="enableValue" />
      </ArgumentValue>
    </Do>
  </Transition>
  <Transition fromState="On" toState="Off" name="ToggleOff">
    <OnCommandPrimitive command="Disable" interface="LedControl" />
    <Do activity="IssueCommand">
      <ArgumentValue name="value">
        <VariableRef variableRef="disableValue" />
      </ArgumentValue>
    </Do>
  </Transition>
</StateMachine>
```

```
/* CIF Start (0, 1050), (200, 75) */
START;
    /* CIF NextState (0, 1237), (200, 50) */
    NEXTSTATE Uninitialized;
/* CIF State (0, 1312), (200, 50) */
state Off;
    /* CIF Input (0, 1312), (200, 50) */
    input LedControl_Enable_Pi;
        /* CIF ProcedureCall (0, 1387), (100, 50) */
        call IssueCommand(Enablevalue);
        /* CIF NextState (0, 1462), (200, 50) */
        NEXTSTATE On;
endstate;
/* CIF State (600, 1312), (200, 50) */
state On;
    /* CIF Input (600, 1312), (200, 50) */
    input LedControl_Disable_Pi;
        /* CIF ProcedureCall (600, 1387), (100, 50) */
        call IssueCommand(Disablevalue);
        /* CIF NextState (600, 1462), (200, 50) */
        NEXTSTATE Off;
endstate;
/* CIF State (1200, 1312), (200, 50) */
state Uninitialized;
    /* CIF Input (1200, 1312), (200, 50) */
    input LedControl_Init_Pi;
        /* CIF ProcedureCall (1200, 1387), (100, 50) */
        call Initialize;
        /* CIF NextState (1200, 1462), (200, 50) */
        NEXTSTATE Off;
endstate;
```

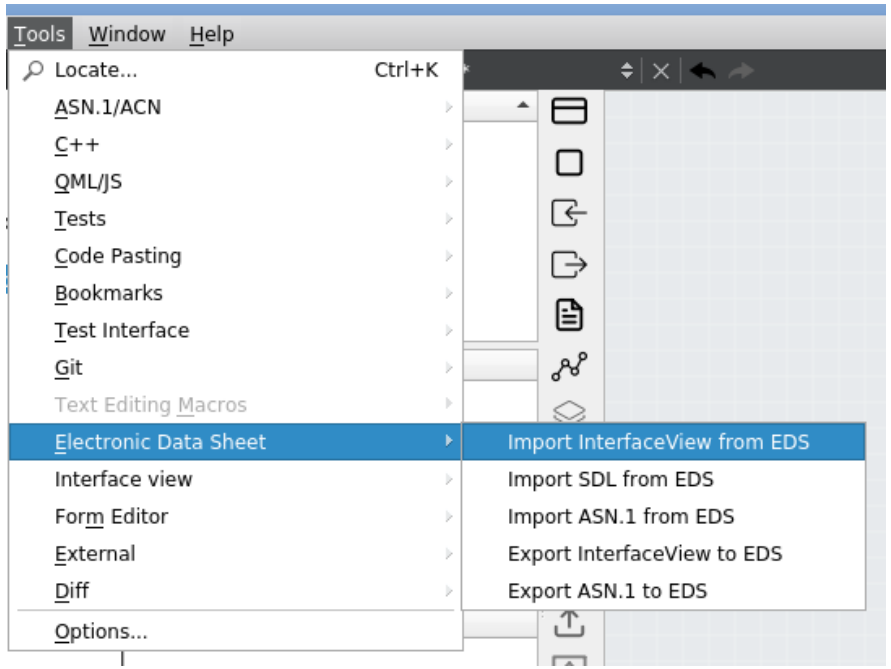
SDL



*graphical layout adjusted for readability

TASTE integration

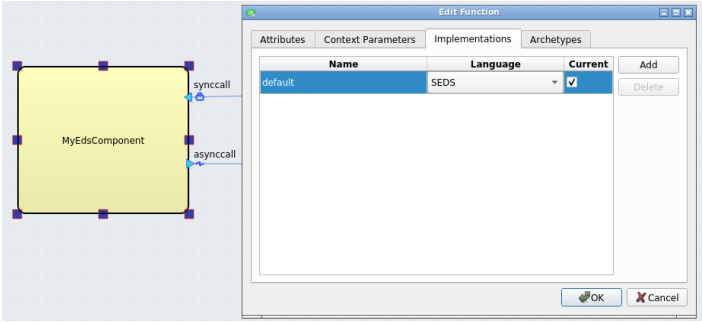
- All translation functionality is available via command line
  - For inclusion in build systems, continuous integration, and advanced users
- The translation functionality is also wrapped in convenient import/export functions available in SpaceCreator GUI:



TASTE integration

- SOIS EDS is also supported as a function implementation language
- The translation to SDL is done on-the-fly
- Experimental feature

InterfaceView (using ASN.1 data model)



system_structure_types.xml

```
<Package name="SEDSDEMO_DATAVIEW">
 <DataTypeSet>
  <IntegerDataType name="MyInteger">
   <Range>
    <MinMaxRange min="0" max="10000" rangeType="inclusiveMinInclusiveMax"/>
   </Range>
  </IntegerDataType>
 </DataTypeSet>
</Package>
```

system_structure.xml

```
<Component name="MyEdsComponent">
 <ProvidedInterfaceSet>
  <Interface type="MyEdsComponentsynccallPi" name="synccallPi"/>
 </ProvidedInterfaceSet>
 <RequiredInterfaceSet>
  <Interface type="MyEdsComponentasynccallRi" name="asynccallRi"/>
 </RequiredInterfaceSet>
 <DeclaredInterfaceSet>
  <Interface name="MyEdsComponentsynccallPi">
   <CommandSet>
    <Command name="synccallPi" mode="sync">
     <Argument type="MyInteger" name="x" mode="in"/>
     <Argument type="MyInteger" name="y" mode="out"/>
    </Command>
   </CommandSet>
  </Interface>
  <Interface name="MyEdsComponentasynccallRi">
   <CommandSet>
    <Command name="asynccallRi" mode="async">
     <Argument type="MyInteger" name="xparam" mode="in"/>
    </Command>
   </CommandSet>
  </Interface>
 </DeclaredInterfaceSet>
</Component>
```

myedscomponent.xml

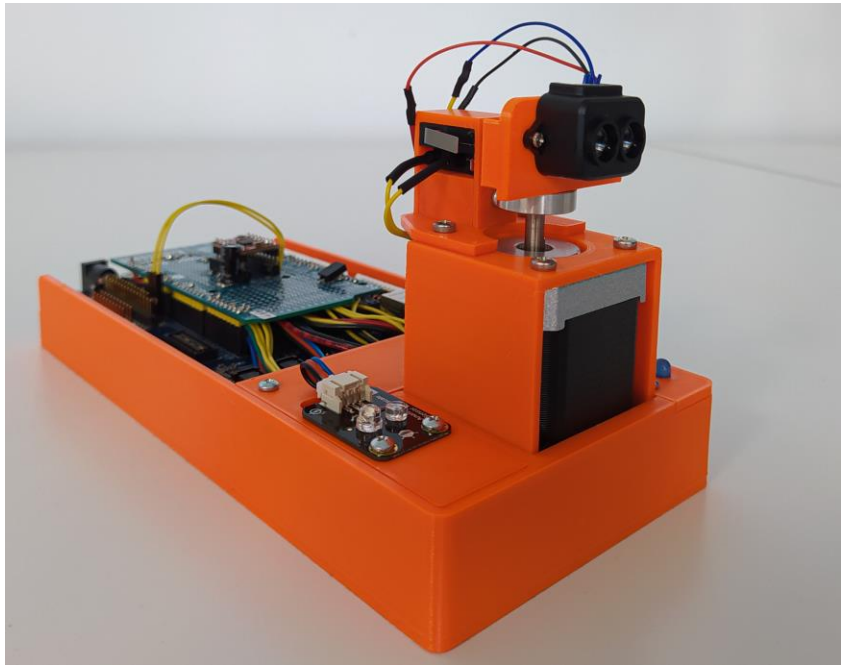
```
<Package name="myedscomponent">
  <DataTypeSet>
    <!-- Include all data types from asn models - generated code -->
    <xi:include href="system_structure_types.xml" parse="xml" xpointer="xpointer(//*[local-name()='DataTypeSet']/*)"/>
  </DataTypeSet>
  <ComponentSet>
    <Component name="myedscomponent">
      <!-- Include all components for MyEdsComponent - generated code -->
      <xi:include href="system_structure.xml" parse="xml" xpointer="xpointer(//*[@name='MyEdsComponent']/*)"/>
      <!-- user code - put your SEDS MyEdsComponent code here -->
      <Implementation>
        <StateMachineSet>
          <StateMachine name="StateMachine">
            <EntryState name="dummy" />
          </StateMachine>
        </StateMachineSet>
      </Implementation>
    </Component>
  </ComponentSet>
</Package>
```

Hardware Access EDS

- Management of MCU peripherals (such as GPIO, ADC, and UART) requires access to memory-mapped registers and interrupts
- Both the EDS component implementation and the SDL dialect it is translated to does not have the concept of a raw "pointer", memory operations or interrupts
- The approach taken in the activity uses a proxy component – Hardware Access EDS (HWAS)
- HWAS consists of:
  - EDS interface specification – to be used by other EDS components
  - C implementation – for handling memory accesses and interrupts
- The developed HWAS implementation is specific to ARM SAMV71 MCU, but the interface could be reused for various other embedded platforms

Demonstration use case

- A small platform with a combination of TASTE and EDS components
- LIDAR instrument
  - Mock "Sun sensor", based on a greyscale sensor connected to SAMV71's ADC
  - LIDAR, combining COTS TF Luna sensor handled via UART, contact sensors read via GPIO and a stepper motor driven via GPIO
  - LEDs for visual output, driven via GPIO
  - 3D printed case

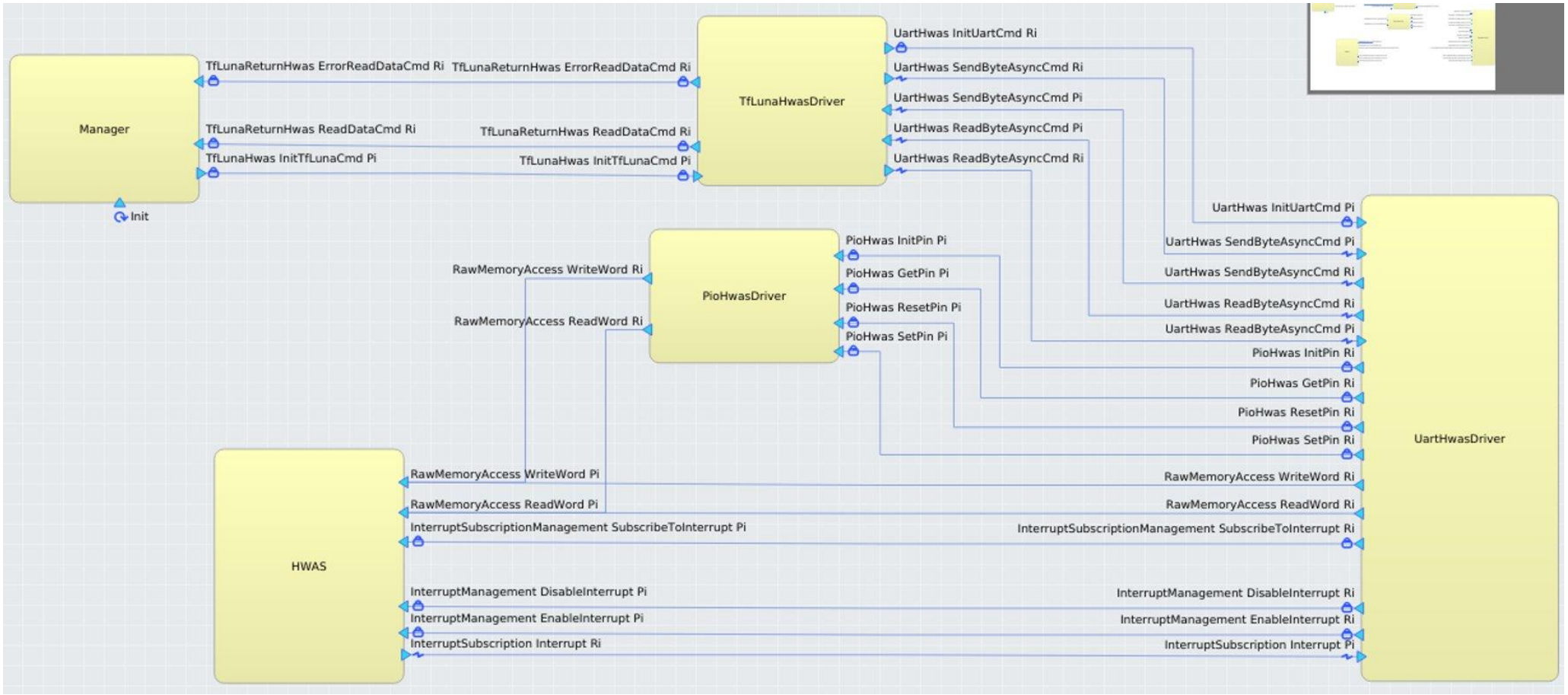


Demonstration use case

- The following EDS specifications were created:
  - HWAS - containing only data types, as well as memory and interface declarations
  - PIO HWAS - for manipulating SAMV71's GPIO, accessing the HW via HWAS
  - UART HWAS - for handling SAMV71's UART, accessing the HW via HWAS and PIO HWAS
  - AFEC HWAS - for reading analogue signals using SAMV71's ADC, accessing the HW via HWAS
  - SunSensor - for acquiring readings from the greyscale sensor, accessed via AFEC HWAS
  - TfLuna - for acquiring range data from TF Luna LIDAR, accessed via UART HWAS
  - MP6500 - for controlling the stepper motor via a HW motor driver, accessed via PIO HWAS
  - Lidar - forming the combined sensor/actuator, forwarding the data from TfLuna, and managing its rotation by controlling the stepper motor via MP6500 and contact sensors via PIO HWAS

Demonstration use case

- Software prototype demonstrating TF Luna sensor usage



Lessons Learned

- All EDS specifications for the hardware peripherals were implemented, deployed onto the target hardware and tested, confirming the feasibility of fully describing onboard hardware devices via Electronic Data Sheets
  - Both physical (PIO, AFEC and UART) and virtual (LIDAR, MP6500, SunSensor)
- The design of the translator involved discussions between N7S and ESA regarding the exact interpretation of certain EDS constructs and their practical mapping onto TASTE
- As a consequence of the specific design choices made within this activity, EDS implementations not sharing the same assumptions and limitations may exhibit compatibility issues

Lessons Learned

- Once the translator was implemented, the required EDS specifications were created by an embedded engineer without previous exposure to MBSE, TASTE and EDS, providing a good perspective from a validation point of view
- Definition of data types and interfaces was done directly in EDS XML without the help of TASTE->EDS translator and was executed without major issues
  - Understanding of EDS interface directions and their mapping onto TASTE was problematic
  - The issue was quickly resolved after several concrete examples were provided and exercised

Lessons Learned

- EDS component implementation has proven itself to be difficult and time consuming
  - Translation from SDL to EDS was not available
  - N7S was not in possession of any dedicated tools, except for a schema aware XML editor
  - Understanding of complex state machines, with long lists of actions and embedded conditional statements, is difficult, both for authoring and review
  - Code executed on the target hardware was the assembly generated from Ada, translated from SDL, which was derived from the input EDS state machine
    - This made troubleshooting quite challenging
    - The issue was mitigated to some extent by prototyping in C

Lessons Learned

- As the engineer had to directly edit a format that is supposed to be just machine readable, the lack of authoring tools was deemed to be an issue
- EDS commands have both input and output/notify arguments, but activities have arguments without any mode, and thus they are assumed as being input only
  - This makes writing helper utilities more difficult, as any calculation results must be stored in state machine variables
- SDL allows to define state machines in a more compact and flexible way than EDS
  - Each EDS transition has "fromState" and "toState" attributes, explicitly defining single state names, while SDL may use special state symbols (*, -) and state lists, which make writing sets of logically identical transitions much faster, more readable and easier to maintain
  - In EDS, for each transition trigger and start state, the single end state is explicitly defined, while SDL allows to perform computation during the transition and decide on the target state in runtime. This limitation was worked around by manually defining an additional, separate, internal state machine in the activity set.
  - EDS on timer transition has a fixed "nanosecondsAfterEntry" attribute, while SDL timers are schedulable with variable time
  - SDL contains the concept of continuous signals, while EDS does not

Lessons Learned

- Both EDS and SDL use abstractions decoupled from hardware, and so any interactions with it have to be performed via a proxy component written in a native language
- In the current implementation provided via TASTE, each memory (and so e.g., register) access is done via function calls, though "glue-code"
  - The result is that GPIO toggle timings were measured to be in the range between 7.5 and 75 microseconds, depending on the cache settings
    - Sufficient for many cases
    - Insufficient for e.g., custom protocols implemented via "bit-banging"
- In the current implementation provided via TASTE, interrupts are reported via sporadic interfaces, and so reporting involves message passing via a queue
  - High-speed interrupt handling may be a challenge, or infeasible at all
- Memory consumption, requiring a few kB of stack per each thread, created for each sporadic interface (translated from each async command), is considered higher than it could be
  - Not a problem on SAMV71 with 2 MB of SDRAM, but could be an issue for large systems or smaller MCUs

"Wishlist"

- Authoring tools for EDS state machines
  - Data types and interfaces can be generated using TASTE... but are also easy to write manually in an XML editor
  - State machines could benefit from a native editor, but also from e.g., SDL to EDS translator
- Special state symbols and state lists as EDS transition start and end states
- Choice of the target state during EDS transitions
  - e.g., via omitting "to" attribute and providing an activity body element indicating the end state
- On timer transitions with variable timing
- [maybe] SDL continuous signals as EDS transition triggers
- [maybe] Connection with lower-level standards for hardware access
- [maybe] Linking EDS component implementations to external specifications
  - For example, C or SDL, like it is done in TASTE InterfaceView

Thank you for your attention

Michał Kurowski
mkurowski@n7space.com

Filip Demski
fdemski@n7space.com

+48 22 299 20 50
www.n7space.com