Aurora

Toolchain Demonstration Report

D5.9

Document Code: AUR-N7S-RP-0003
Document Version: 1.3
Document Date: 28/04/2023
Internal Reference: DOC00316753

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101004291

Signature Control

Written	Checked	Approved Configuration Management	Approved Quality Assurance	Approved Project Management
M. Kurowski	F. Florczyk	R. M. León	A. López	A.I Rodríguez
Date and Signature	Date and Signature	Date and Signature	Date and Signature	Date and Signature
Signature not needed if electronically approved by route				

Changes Record

Rev	Date	Author	Affected section	Changes
1	2023-01-31	M. Kurowski	All	Initial issue
2	2023-02-03	M. Kurowski	Requirement coverage	Updated requirement coverage
3	2023-02-06	M. Kurowski	Requirement coverage	Updated requirement coverage, added justification
4	2023-04-28	M. Kurowski	Requirement coverage Code metrics Reference documents	Updated requirement coverage and removed justification Added appendix with code metrics

Index

1. Introduction 5
1.1. Purpose, scope and content 5
1.2. Project motivation and objectives 5
2. Applicable and reference documents 6
2.1. Applicable documents 6
2.2. Reference documents 6
3. Terms, definitions and abbreviated terms 8
4. Toolchain Overview 9
4.1. Introduction 9
4.2. Software behaviour summary 9
5. Toolchain Development 12
5.1. Context 12
5.2. Method 12
5.3. SpaceCreator tests 13
5.4. Kazoo tests 13
6. Toolchain Demonstration 15
6.1. Overview 15

6.2. Reference Component Set 15
6.2.1. Overview 15
6.2.2. TimeService 15
6.2.3. DataStore 16
6.3. Requirement coverage 17
6.3.1. N to M asynchronous communication 17
6.3.2. Fault Detection 19
6.3.3. Events 21
6.3.4. Component Management 23
6.3.5. CBI Runtime 26
6.3.6. Reference Component Set and DataStore 27
6.3.7. AOCS/GNC SW components 29
6.3.8. General 31
6.3.9. Requirements for Tool-suite integration 32
6.3.10. Summary 38
7.Recommendations 39
7.1. General 39
7.2. Interface View 39
7.3. Function Tester 39
7.4. Simulink Importer 39
7.5. Runtime 39
8. Lists 41
8.1. List of Tables 41
8.2. List of Figures 41

9. Annex A – Code metrics 43

1. Introduction

1.1. Purpose, scope and content

The purpose of this document is to report the status of the AURORA Toolchain for TASTE with respect to the requirements of the AURORA project stated in the Component-Based Interface (CBI) Requirements Specification [RD1].

This report provides an overview of the TASTE Toolchain, explains the development process of the AURORA software contributions, and describes the Toolchain demonstration activity for the requirements fulfilment evidence. Finally, the report summaries some recommendations for future activities that could benefit the TASTE toolchain and its users.

1.2. Project motivation and objectives

The AURORA project aims to provide a European tool suite for the process of development and validation of a critical Auto-coded Flight software product in the Space domain and the demonstration of Autocoding technology in an industrially relevant environment.

The solution uses QGen to transform Simulink/MATLAB models into source code to be directly integrated into embedded software items. This process is orchestrated by ESA's TASTE MBSE toolchain and exposed to the users via a friendly Integrated Development Environment. In order to ensure interoperability, the developed Component Based Interfaces Model integrates state-of-the-art concepts inspired by similar solutions (AUTOSAR, CFS) and will be able to be deployed independently from TASTE (e.g., for manual code). Practical verification are enabled by TASTE runtime components targeting Leon3 processor.

The technology demonstration is carried out by exercising the automated code in AURORA with the already validated and verified results of the auto-generated code from the ESA's Euclid mission (where SENER is the prime contractor).

In addition to the software design, implementation, integration and validation, the project defines the Autocoded Flight Software Life-cycle process and methodology for the Specification, Development and Validation of Autocoded-SW.

2. Applicable and reference documents

2.1. Applicable documents

ID	Title	Reference	Rev.
AD1	ECSS – Space engineering Software	ECSS-E-ST-40C	2009/03/06
AD2	ECSS – Space engineering Software engineering handbook	ECSS-E-HB-40A	2013/12/11
AD3	ECSS – Space Engineering Telemetry and telecommand packet utilization	ECSS-E-ST-70-41C	2016/04/15
AD4	CCSDS – Space Packet Protocol	CCSDS 133.0-B-2	2020/06

2.2. Reference documents

ID	Title	Reference	Rev.
RD1	D5.1 CBI Requirements Specification	AUR-UPM-TN-0001	1.4
RD2	D5.2 CBI Technical Architecture	AUR-N7S-RP-0001	1.5
RD3	D5.3 AURORA Component Model	AUR-UPM-RP-0010	3.4
RD4	D5.4 AURORA Interface Specification	AUR-N7S-RP-0002	1.6
RD5	D5.7 CBI Demonstration Report	AUR-UPM-RP-0012	1.2
RD6	TASTE toolchain	https://taste.tools/	N/A
RD7	TASTE wiki	https://taste.tuxfamily.org/wiki/	
RD8	RTEMS SMP QDP	https://rtems-qual.io.esa.int/	N/A
RD9	TASTE main repository	https://gitrepos.estec.esa.int/taste/taste-setup	N/A
RD10	Kazoo repository	https://gitrepos.estec.esa.int/taste/kazoo	N/A
RD11	SpaceCreator repository	https://gitrepos.estec.esa.int/taste/spacecreator	N/A
RD12	AURORA Reference Component Set repository	https://github.com/n7space/AURORA-Reference-Component-Set	N/A
RD13	AURORA Validation repository	https://github.com/n7space/AURORA-Validation	N/A

ID	Title	Reference	Rev.
RD14	Custom Sparc Instruction Simulator repository	https://github.com/n7space/sis	N/A
RD15	D5.2 AURORA SW Development Plan	AUR-SEN-PL-0002	1.3

3. Terms, definitions and abbreviated terms

This document's acronyms and abbreviations are listed here under.

AADL	Architecture Analysis & Design Language
BSP	Board Support Package
CBI	Component Based Interfaces
CBIM	Component Based Interfaces Model
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HW	Hardware
IDE	Integrated Development Environment
I/O	Input and Output
MBSE	Model Based Software/System Engineering
N7S	N7 Space
OBET	On-Board Elapsed Time
RAM	Random Access Memory
RTEMS	Real-Time Executive for Multiprocessor Systems
RTOS	Real-Time Operating System
SDL	Specification and Description Language
SMP	Symmetric Multiprocessing
SW	Software
QDP	Qualification Data Package
VM	Virtual Machine
XMI	XML Metadata Interchange
XML	eXtensible Markup Language

4. Toolchain Overview

4.1. Introduction

TASTE [RD6], The ASSERT Set of Tools for Engineering, traces its origins to the EU/FP6 ASSERT project, led by the European Space Agency in the early 00s. It includes a suite of tools (editors, compilers and code generators/transpilers), as well as a range of software libraries providing (parts of) runtimes, drivers and middleware. The used technologies include, but are not limited to, Architecture Analysis and Description Language (AADL), Abstract Syntax Notation 1 (ASN.1), Specification and Description Language (SDL), Simulink, Qt, C, C++, Ada, Python, RTEMS, FreeRTOS, PolyORB and SOIS Electronic Data Sheets. It contains contributions from tens of entities from across both the industry and academia.

The toolchain focuses on model-based solutions ("Model Based Systems/Software Engineering") for the production of space on-board software. However, it can be also used e.g., in robotics.

As of 2023, the main user interface is the SpaceCreator IDE, which replaced some of the earlier legacy tools still used at the beginning of the AURORA project. However, many of the tools, including the build system, compilers, and transpilers, expose command line interface, enabling integration with other GUIs (such as Capella) or automation (e.g., continuous integration environments).

TASTE toolchain is publicly available and free to download and use, as its core technologies are available under open-source licenses. TASTE is distributed as a Virtual Machine, available for download [RD6] for the convenience of end-users. It is also possible to deploy it inside a Docker image or natively within a compatible Linux distribution, though the amount of effort required may vary significantly. As the range of supported technologies is quite wide and includes both some specialized and proprietary software that cannot be freely distributed, some additional tools and libraries may need to be procured separately (e.g., Simulink, QGen, etc.). There is also some freely available software that is not included in the base distribution to avoid bloat (e.g., RTEMS SMP QDP package [RD8]), but which can be downloaded by dedicated installation scripts. AURORA toolchain framework release includes a pre-configured Virtual Machine which includes the additional open-source software required by or created within the scope of AURORA. Proprietary software is not included due to licensing restrictions and must be procured by the end users.

4.2. Software behaviour summary

Software definition in TASTE is logically divided into the following parts: data type definition (Data View), logical architecture (Interface View), physical architecture (Deployment View) and behaviour definition (can be C, Ada, SDL, Simulink, etc.). SpaceCreator is used to define the Data View, Interface View and Deployment View. Default Deployment View targeting a single Linux application can be generated automatically. If the behaviour implementation is to be defined in C, C++ or Ada, SpaceCreator can be also used to edit the relevant files. Otherwise, the appropriate tool is launched (e.g., OpenGEODE or Simulink), if present in the system. Figure 1, Figure 2 and Figure 3 present the Interface View, Deployment View and Data View of the same simple system, which contains 2 components. Behaviour of the Controller component is specified using C code, and the behaviour of Simulink component is specified using Simulink model, with QGenC as the selected code generator. The presented Data View is a part of the combined Data View generated from both the user provided (in this case, via the Simulink importer implemented within the scope of the AURORA project) ASN.1 and TASTE default data types.

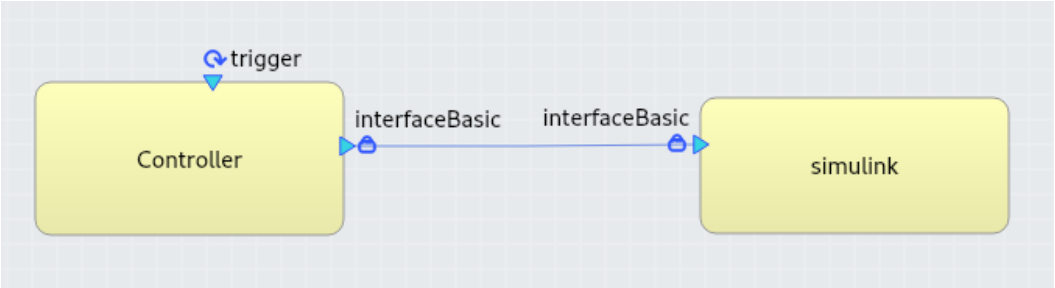

Figure 1 Interface View of a simple system (graphical representation of XML file)

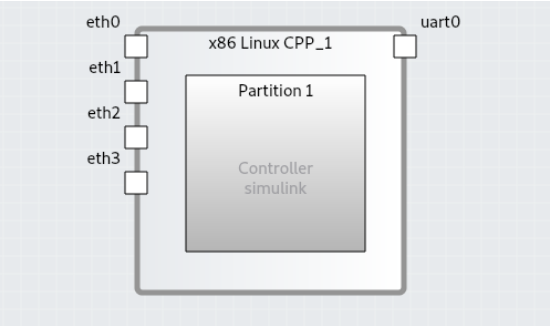

Figure 2 Deployment View of a simple system (graphical representation of XML file)

```
MATLAB-STANDARD-DATATYPES DEFINITIONS ::= BEGIN
MatLab-Boolean ::= BOOLEAN
MatLab-Double ::= REAL(-1.79769e+308 .. 1.79769e+308)
MatLab-Single ::= REAL(-3.40282e+38 .. 3.40282e+38)
MatLab-Int8 ::= INTEGER(-128 .. 127)
MatLab-Uint8 ::= INTEGER(0 .. 255)
MatLab-Int16 ::= INTEGER(-32768 .. 32767)
MatLab-Uint16 ::= INTEGER(0 .. 65535)
MatLab-Int32 ::= INTEGER(-2147483648 .. 2147483647)
MatLab-Uint32 ::= INTEGER(0 .. 4294967295)
END


TASTE-BasicTypes DEFINITIONS ::=
BEGIN

-- Set of TASTE predefined basic types

T-Int32 ::=  INTEGER (-2147483648 .. 2147483647)

T-UInt32 ::= INTEGER (0 .. 4294967295)

T-Int8 ::= INTEGER (-128 .. 127)

T-UInt8 ::= INTEGER (0 .. 255)

T-Boolean ::= BOOLEAN

T-Null-Record ::= SEQUENCE {}

END
```

Figure 3 Data View of a simple system (ASN.1 form)

Project initialization is managed by a dedicated “taste” script.

Project build is managed by the Makefile generated during the project initialization, and optionally adjusted by the user (e.g., to provide additional defines or paths to libraries). Makefile invocation is possible directly from the SpaceCreator GUI, via the “build” command.

Build process includes two major steps:

- Generation of skeletons for the user code, based on the Data View and Interface View
- Final compilation of the target binaries.

Build process relies on Kazoo templating engine, which generates component wrappers, parts of the runtime, as well as intermediate build scripts, responsible for compilation, linking, inclusion of additional libraries, etc. The intermediate build scripts, if necessary, call the required code transpilers, e.g., OpenGEODE or QGenC.

It should be noted that changes in the Data View or Interface View are propagated to the interfaces of the code or models responsible for behaviour definition. The logic contained within the code or models needs to be manually adjusted. This is particularly important for Simulink models, as unlike SDL, C or Ada, the models are not divided into separate header/declaration/interface and body/definition/process files. Major updates of the TASTE toolchain may also require adjustment of the model/code logic to the update interfaces.

Detailed TASTE documentation is available on the TASTE Wiki [RD7].

Figure 4 presents a simplified component definition workflow, relevant to AURORA, and described in [RD4].

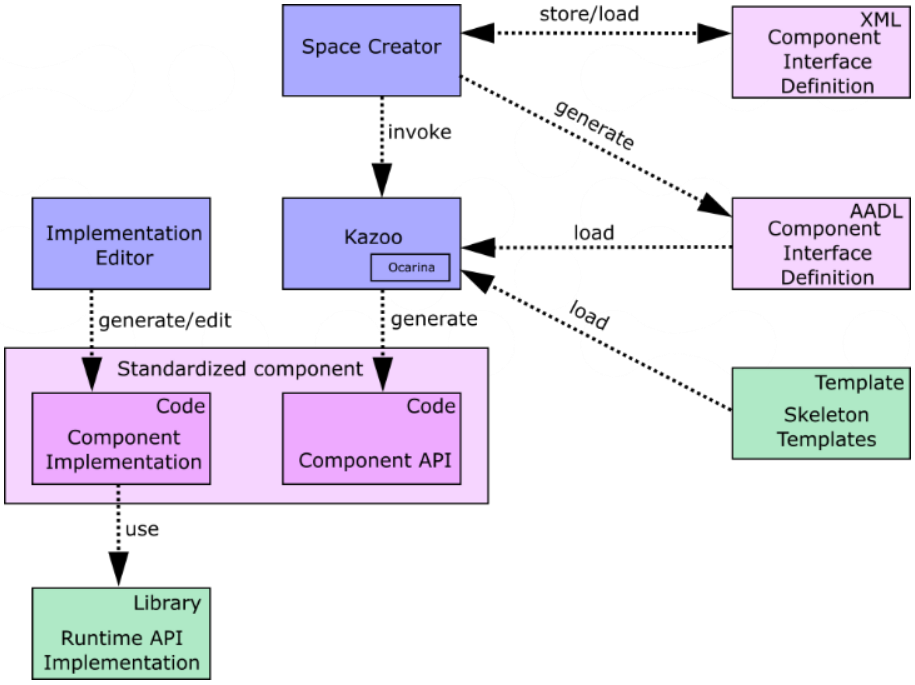

Figure 4 Component definition workflow

5. Toolchain Development

5.1. Context

At the start of the AURORA project, the toolchain was built on around 15 years of experience, and included both mature tools (Ocarina AADL processor, asn1scc ASN.1 compiler), relatively new additions (Kazoo template generator), as well as still-under-development software such as SpaceCreator, which, at the time, had not yet reached operational capacity.

The toolchain is open source and continuously receives various contributions from projects that are independent from AURORA, also by entities from outside of the AURORA consortium - a good example of this is the development of the base SpaceCreator IDE by a third-party commercial entity. TASTE also has active users, who should not be discouraged by software instability, maintenance issues and fragmentation.

TASTE evolution is led and guided by ESA, who has the final say about the accepted features. These factors posed some challenges and introduced constraints throughout the software design and implementation phase. The objective was to both fulfil the requirements consolidated by the consortium [RD1], and make sure that compatibility is maintained, so that the software will be eventually merged with the mainline TASTE distribution. A software fork would divide the user-base, going against one of the primary AURORA objectives, which was Component Standardization, and would create maintenance issues in the future. The above resulted in the design presented in [RD2], which includes:

- Contributions to the SpaceCreator IDE,
- Contributions to the Kazoo template generator, and the relevant templates and Ocarina,
- Contributions to Data Modelling Tools,
- Contributions to TASTE Linux Runtime,
- Contributions to TASTE configuration (AADL definitions, AADL and XML templates, property files and installation scripts),
- Contributions to Sparc Instruction Simulator,
- Development of TASTE Leon3 Runtime,
- Development of BSP supporting the TASTE Leon3 Runtime,
- Development of UART communication device driver for the TASTE Leon3 Runtime,
- Development of TASTE Reference Component Set.

5.2. Method

The development process started with the requirement consolidation (resulting in [RD1]), followed by the design (resulting in [RD2]) and the actual software implementation (resulting in the aforementioned software contributions). Both the design and the implementation were taking into the account the latest developments within the TASTE toolchain, such as the release of the base SpaceCreator IDE, its update to the newest Qt version, ESA's development of OpenGEODE, development of the base TASTE Linux Runtime and third-party contributions to the Kazoo templates. Adjustments to the implementation required by the changing software were being iteratively propagated to the design.

The implementation followed a simple process:

- Selection of a feature to be implemented,
- Implementation of the feature, together with relevant tests,
- Review of the implementation and the relevant tests,
- Implementation of the review feedback, if necessary, followed by the iteration of the review process.

The tests created during the feature implementation are included in the relevant repository. Tests for the high-level AURORA related functionality are contained in SpaceCreator [RD11] and Kazoo [RD10] repositories.

5.3. SpaceCreator tests

SpaceCreator repository [RD11] contains unit, integration and manual tests of the many functionalities integrated within the IDE, including Simulink import, function testing and CBI definition (layers, archetypes and multicast). Unit and integration testing is used whenever possible, and the tests are automatically executed within the continuous integration environment. Manual testing is reserved mainly for GUI testing and testing dependent on third-party software that cannot be included in or accessed from the publicly available SpaceCreator repository.

The most relevant tests for the AURORA project are contained in the following folders:

- tests/unittests/conversion/asn1/simulinktoasn1translator
- tests/unittests/conversion/iv/simulinktoivtranslator
- tests/unittests/csv
- tests/unittests/Simulink
- tests/unittests/testgenerator
- tests/integrationtests/testgenerator
- tests/integrationtests/conversion/simulinktoasn1
- tests/integrationtests/conversion/simulinktoiv
- tests/manual/functiontester
- tests/manual/qgenc
- tests/manual/simulinkimporterplugin

The developed code relies on re-used modules with pre-existing (though sometimes extended) tests, which are not listed above. This affects mostly the changes made to the Interface View related to layer, archetype and multicast support, as well as generic model-to-model translation framework.

It should be noted that the manual tests can be used to demonstrate some of the toolchain capabilities from the user's perspective. The integration tests can be used to demonstrate some inner workings of the toolchain.

5.4. Kazoo tests

Kazoo repository [RD10] contains integration tests relevant to the developed TASTE Leon3 Runtime and the extended TASTE Linux Runtime. The most relevant tests for the AURORA project are contained in the following folders:

- test/linux-cpp-n-2-m-demo
- test/linux-cpp-n-2-m-with-routing-table-demo
- test/linux-cpp-one-2-n-demo
- test/linux-cpp-one-2-n-protected
- test/linux-cpp-one-2-n-sporadic
- test/linux-cpp-one-2-n-unprotected
- test/linux-cpp-startup-priority-demo
- test/linux-get-sender-demo
- test/linux-get-sender-with-broker-demo
- test/RTEMS6_SMP_QDP_cyclic_call
- test/RTEMS6_SMP_QDP_protected_call
- test/RTEMS6_SMP_QDP_sporadic_call
- test/RTEMS6_SMP_QDP_unprotected_call
- test/RTEMS6_SMP_QDP-get-sender-demo
- test/RTEMS6_SMP_QDP-get-sender-with-broker-demo
- test/RTEMS6_SMP_QDP-n-2-m-demo
- test/RTEMS6_SMP_QDP-n-2-m-with-routing-table
- test/RTEMS6_SMP_QDP-perf-mon

The tests are automatically executed within a continuous integration environment of the main TASTE repository [RD9]. Some of them involve the execution of the generated code (natively, for the TASTE Linux Runtime, and via an emulator, for the TASTE Leon3 Runtime), others test only the code generation alone.

Each of the tests can be used to demonstrate some of the toolchain capabilities from the user's perspective.

6. Toolchain Demonstration

6.1. Overview

As mentioned in chapter 5, multiple tests, both manual and automatic, were created during the software development process. However, for the purpose of toolchain demonstration and requirement fulfilment evidence, a set of validation artefacts was prepared and delivered [RD13]. As stated in [RD1], each requirement is to be validated by at least one of:

- Test (manual or automated)
- Analysis
- Review of Design
- Inspection
- Example Model
- Example Executable Model

The validation repository [RD13] has the following structure:

- AURORA-Reference-Component-Set – submodule containing the reference component set [RD12].
- archetypes – archetype libraries used in the example models.
- requirements – validation evidence.

The requirements folder contains subfolders, one for each requirement from [RD1], containing the relevant evidence, in the form compliant with the validation method selected for the given requirement. Example models can be opened and reviewed in TASTE. Testing – if applicable - can be performed by issuing “make test” command, unless specific instructions are contained in the accompanying text files. Inspection and review documentation is also contained in text files.

6.2. Reference Component Set

6.2.1. Overview

During the requirement consolidation and design activities it was concluded that some capabilities, like data storage and time queries, are application specific and so difficult to reliably standardize. Therefore, instead of integrating them inside the runtimes and the component interfaces model supported directly within SpaceCreator, which could result in feature creep and software bloat, they are provided in the form of discrete components – AURORA Reference Component Set [RD12]. The components are provided bundled with example models illustrating their use.

6.2.2. TimeService

TimeService component, illustrated in Figure 5, was designed as a reference implementation satisfying the requirements for TimeService defined in [RD1]. The implementation can be used both in TASTE Linux and Leon3 Runtimes, as it contains code specific for each platform. The exposed interfaces are platform independent, and, if needed, the implementation can be extended to support more runtimes.

TimeService provides interfaces for querying elapsed time, converting it to CUC and cFS timestamps, and comparing the timestamps. All interfaces all synchronous, for simplicity and performance.

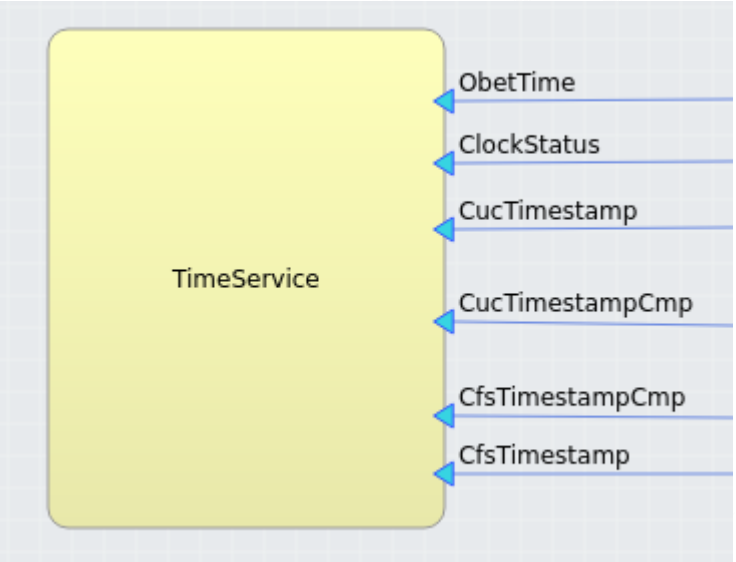

Figure 5 TimeService component

6.2.3. DataStore

DataStore component, illustrated in Figure 6, was designed as a reference implementation satisfying the requirements for DataStore defined in [RD1]. The implementation can be used in any runtime supporting C code (which includes TASTE Linux and Leon3 Runtimes), as it has been written in a portable manner. Time-dependent queries depend on the TimeService, abstracting the platform-specific details. The data manipulation interfaces follow a CRUD (Create, Read/Retrieve, Update and Delete) pattern. Additional interfaces are added for events, allowing monitoring of the DataStore activities, via subscriptions by the interested clients. The implementation of the reference DataStore component showcases the multicast capabilities introduced during the AURORA project.

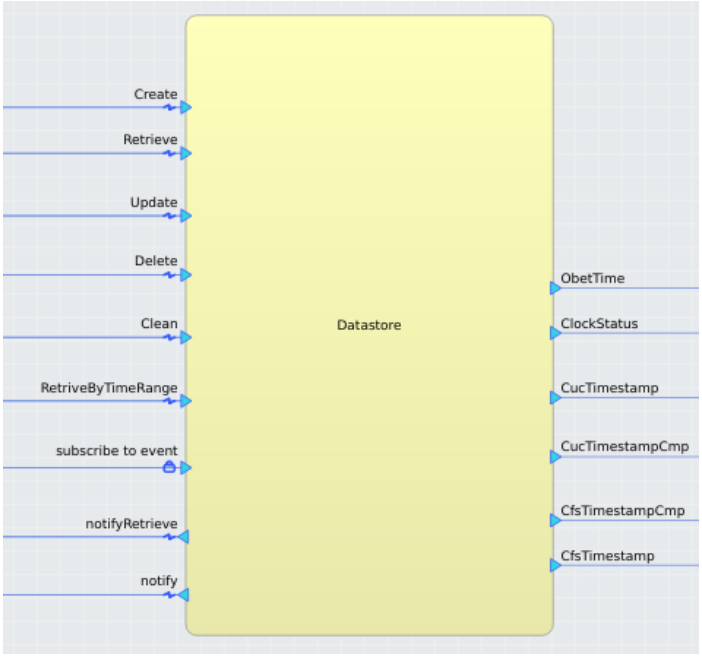

Figure 6 DataStore component

6.3. Requirement coverage

The subsequent chapters correspond to groups of requirements from [RD1], which are representative of the capabilities of the toolchain. DataStore and Reference Component Set group were merged together as the requirements are complementary.

6.3.1. N to M asynchronous communication

TASTE originally supported 1:1 and N:1 communication. Multicast (1:N) capability has been implemented during the AURORA project, which, combined with the N:1 communication, allows to create N:M communication patterns, as presented in Figure 7. Publishers broadcast messages to Subscribers, and Subscribers report to the Arbiter to count the messages. The Subscribers can query the source of the received message. TASTE is capable of automatically generating a graphical user interface to create and receive messages, as illustrated in Figure 8. The contents of the messages are defined via user-provided ASN.1 data type definitions, and, if required, can carry timestamps, as illustrated in Figure 9.

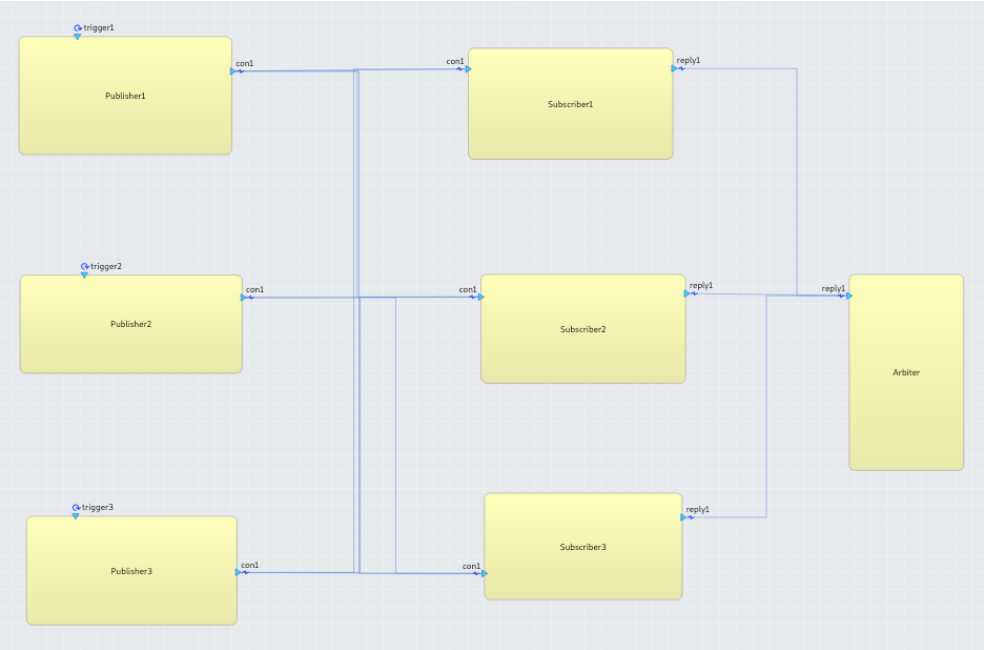

Figure 7 Example model with N to M asynchronous communication

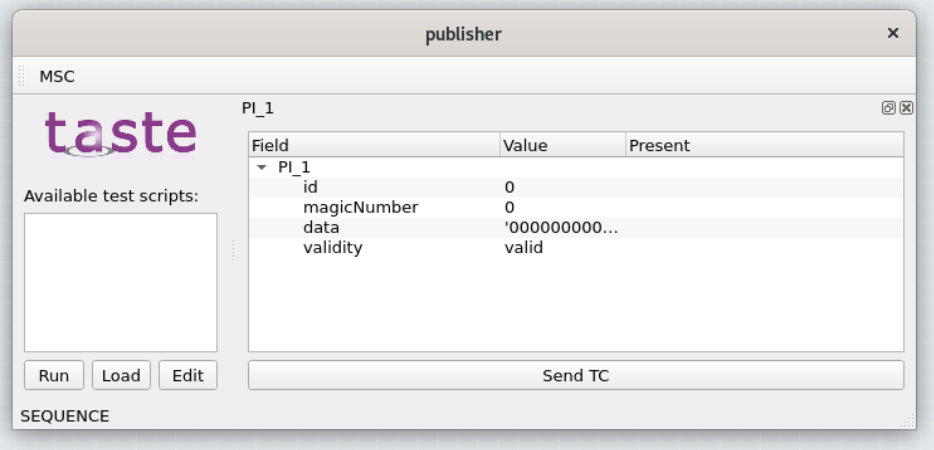

Figure 8 MMI generated by TASTE

```
TestMessage       ::= SEQUENCE {
    id            MyInteger,
    timestamp     INTEGER (0 .. 18446744073709551615)
}
```

Figure 9 Example message with a timestamp

Table 1 N to M asynchronous communication requirements

ID	Requirement	Validated by
NtoM_COM_AURORA_0001	The component model shall support 1 to N asynchronous communication	M
NtoM_COM_AURORA_0002	The component model shall support N to 1 communication	M
NtoM_COM_AURORA_0003	The component model shall support N to M communication	M
NtoM_COM_AURORA_0004	The publisher should have the capability to create messages	M
NtoM_COM_AURORA_0005	The publisher should have the capability to provide a human interface to create messages	I
NtoM_COM_AURORA_0006	The messages should have the capability to carry a timestamp indicating when they were sent	M

6.3.2. Fault Detection

The error communication uses the same facilities as normal communication, and so supports N:1, 1:N and N:M communication patterns. In order to facilitate interface harmonization, a set of archetypes has been established (see Figure 10). The archetypes can be user defined, as so tailored to application specific cases. The archetypes created within the scope of AURORA project are intended to be a reference, and a demonstration of the relevant capabilities. The IDs can be both included in the message, if defined via ASN.1, and queried via the API for message sender retrieval. The timestamps, if needed, can be defined via ASN.1. TASTE can detect message decoding issues (be it ASN.1, or checksum, if supported by the selected packetizer), and user can subscribe to such information, as illustrated in Figure 11. It should be noted though that as the errors are due to decoding issue, the decoded information is not available. Instead, the entire frame can be inspected.

```
<ArchetypeLibrary
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://www.example.org/ArchetypeLibrary"
    xsi:schemaLocation="http://www.example.org/ArchetypeLibrary ArchetypeLibrarySchema.xsd">

        <FunctionArchetype name="Error_receiver_async">
            <InterfaceArchetype name="Error_receive" interfaceType="PROVIDED" kind="SPORADIC" layer="ErrorCommunication">
                <ParameterArchetype name="ErrorMessage" type="T-ErrorMessage" direction="IN"/>
            </InterfaceArchetype>
        </FunctionArchetype>

        <FunctionArchetype name="Error_receiver_sync">
            <InterfaceArchetype name="Error_receive" interfaceType="PROVIDED" kind="PROTECTED" layer="ErrorCommunication">
                <ParameterArchetype name="ErrorMessage" type="T-ErrorMessage" direction="IN"/>
            </InterfaceArchetype>
        </FunctionArchetype>

        <FunctionArchetype name="Error_sender_async">
            <InterfaceArchetype name="Error_send" interfaceType="REQUIRED" kind="SPORADIC" layer="ErrorCommunication">
                <ParameterArchetype name="ErrorMessage" type="T-ErrorMessage" direction="IN"/>
            </InterfaceArchetype>
        </FunctionArchetype>

        <FunctionArchetype name="Error_sender_sync">
            <InterfaceArchetype name="Error_send" interfaceType="REQUIRED" kind="PROTECTED" layer="ErrorCommunication">
                <ParameterArchetype name="ErrorMessage" type="T-ErrorMessage" direction="IN"/>
            </InterfaceArchetype>
        </FunctionArchetype>

        <CommunicationLayerTypes>
            <CommunicationLayerType name="ErrorCommunication"/>
        </CommunicationLayerTypes>
</ArchetypeLibrary>
```

Figure 10 Archetypes for error communication

```
static void error_manager_broker_error_detected(const Broker_ErrorType err, uint8_t* const data, const size_t length)
{
    error_manager_RI_error();
}

void error_manager_startup(void)
{
    Broker_register_error_callback(&error_manager_broker_error_detected);
}
```

Figure 11 Communication error callback registration

Table 2 Fault Detection requirements

ID	Requirement	Validated by
FAULT_DETECTION_AURORA_0001	The components shall have the capability to perform N to 1 synchronous error communication	M
FAULT_DETECTION_AURORA_0002	The components shall have the capability to perform N to 1 asynchronous error communication	M
FAULT_DETECTION_AURORA_0003	The error communication protocol shall state different component status	M

ID	Requirement	Validated by
FAULT_DETECTION_AURORA_0004	The error communication shall state ID of the component	M
FAULT_DETECTION_AURORA_0005	The error communication should have a timestamp	M
FAULT_DETECTION_AURORA_0006	The subscriber component in the communication shall have the capability to check for errors from the error communication received	M
FAULT_DETECTION_AURORA_0007	Components shall have the capability to write their status in data structures of their own and consulting other components status	M

6.3.3. Events

The event communication uses the same facilities as normal communication, and so supports N:1, 1:N and N:M communication patterns. In order to facilitate interface harmonization, a set of archetypes has been established (see Figure 12). Information carried by Events can be defined via ASN.1. The runtime provides the sender ID and allows to enable or disable individual routes between components (see Figure 14 for an example). The routes still have to be statically defined within the Interface View. Users can implement the publisher-subscriber pattern in a variety of ways, best suitable for their needs. Figure 13 illustrates an example of a custom event bus (the EventBus component), which distributes the submitted events among the listeners subscribed for each particular event. The code presented in Figure 14 is part of the EventBus component implementation.

```
<ArchetypeLibrary
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://www.example.org/ArchetypeLibrary"
    xsi:schemaLocation="http://www.example.org/ArchetypeLibrary ArchetypeLibrarySchema.xsd">

        <FunctionArchetype name="Event_generator">
            <InterfaceArchetype name="notify" interfaceType="REQUIRED" kind="SPORADIC" layer="EventCommunication">
                <ParameterArchetype name="EventMessage" type="T-EventMessage" direction="IN"/>
            </InterfaceArchetype>
        </FunctionArchetype>

        <FunctionArchetype name="Event_bus">
            <InterfaceArchetype name="receive_event" interfaceType="PROVIDED" kind="SPORADIC" layer="EventCommunication">
                <ParameterArchetype name="EventMessage" type="T-EventMessage" direction="IN"/>
            </InterfaceArchetype>
            <InterfaceArchetype name="dispatch_event" interfaceType="REQUIRED" kind="SPORADIC" layer="EventCommunication">
                <ParameterArchetype name="EventMessage" type="T-EventMessage" direction="IN"/>
            </InterfaceArchetype>
            <InterfaceArchetype name="subscribe_to_event" interfaceType="PROVIDED" kind="PROTECTED" layer="EventCommunication">
                <ParameterArchetype name="eventId" type="T-UInt32" direction="IN"/>
                <ParameterArchetype name="shouldSubscribe" type="T-Boolean" direction="IN"/>
            </InterfaceArchetype>
        </FunctionArchetype>

        <FunctionArchetype name="Event_listener">
            <InterfaceArchetype name="receive_event" interfaceType="PROVIDED" kind="SPORADIC" layer="EventCommunication">
                <ParameterArchetype name="EventMessage" type="T-EventMessage" direction="IN"/>
            </InterfaceArchetype>
            <InterfaceArchetype name="subscribe_to_event" interfaceType="REQUIRED" kind="PROTECTED" layer="EventCommunication">
                <ParameterArchetype name="eventId" type="T-UInt32" direction="IN"/>
                <ParameterArchetype name="shouldSubscribe" type="T-Boolean" direction="IN"/>
            </InterfaceArchetype>
        </FunctionArchetype>

        <CommunicationLayerTypes>
            <CommunicationLayerType name="EventCommunication"/>
        </CommunicationLayerTypes>
</ArchetypeLibrary>
```

Figure 12 Archetypes for event communication

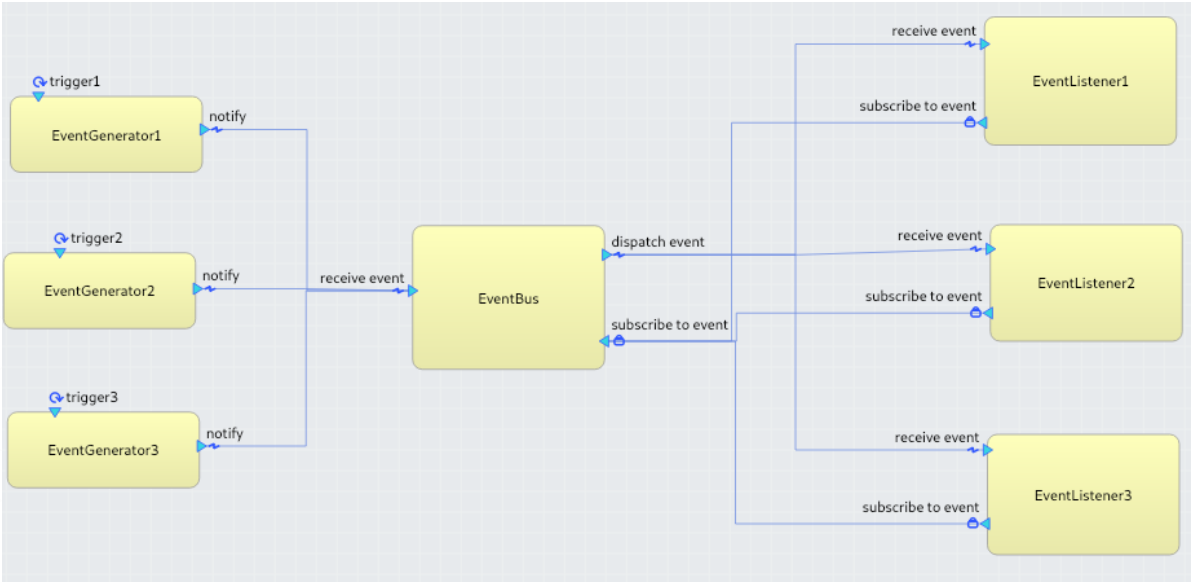

Figure 13 Example user implementation of an event bus for subscriber-publisher pattern

```
void eventbus_enableRoutesForEvent(asn1SccT_UInt32 eventId)
{
    for(int i = 0; i < subscriptionCounter; i++)
    {
        if(subscriptionTable[i].eventId == eventId)
        {
            eventbus_dispatch_event_routing_table[subscriptionTable[i].pid] = true;
        }
    }
}

void eventbus_disableRoutesForEvent(asn1SccT_UInt32 eventId)
{
    for(int i = 0; i < subscriptionCounter; i++)
    {
        if(subscriptionTable[i].eventId == eventId)
        {
            eventbus_dispatch_event_routing_table[subscriptionTable[i].pid] = false;
        }
    }
}
```

Figure 14 Example usage of CBI API for altering the message routing within the runtime glue code

Table 3 Events requirements

ID	Requirement	Validated by
EVENT_SERVICES_AURORA_0001	The component shall have the capability to register event listeners	M
EVENT_SERVICES_AURORA_0002	The component shall have the capability to send event notifications	M
EVENT_SERVICES_AURORA_0003	Each event shall have a unique ID	M
EVENT_SERVICES_AURORA_0004	Each event should include additional information	M
EVENT_SERVICES_AURORA_0005	The component should have the capability to unregister an event listener	M

6.3.4. Component Management

TASTE is designed with safety critical systems in mind, and as such avoids, if possible, use of facilities not suitable for such systems, including dynamic resource allocation. Initial component startup is performed automatically, in the order derived from the priorities specified in the Interface View. Different behaviours can be invoked depending on reset reason, as presented in Figure 18. Additional application and component specific behaviours can be implemented by providing relevant interfaces and additional controller components, as illustrated in Figure 15. Interface standardization can be supported by using archetypes for component management (Figure 16) and power management (Figure 17).

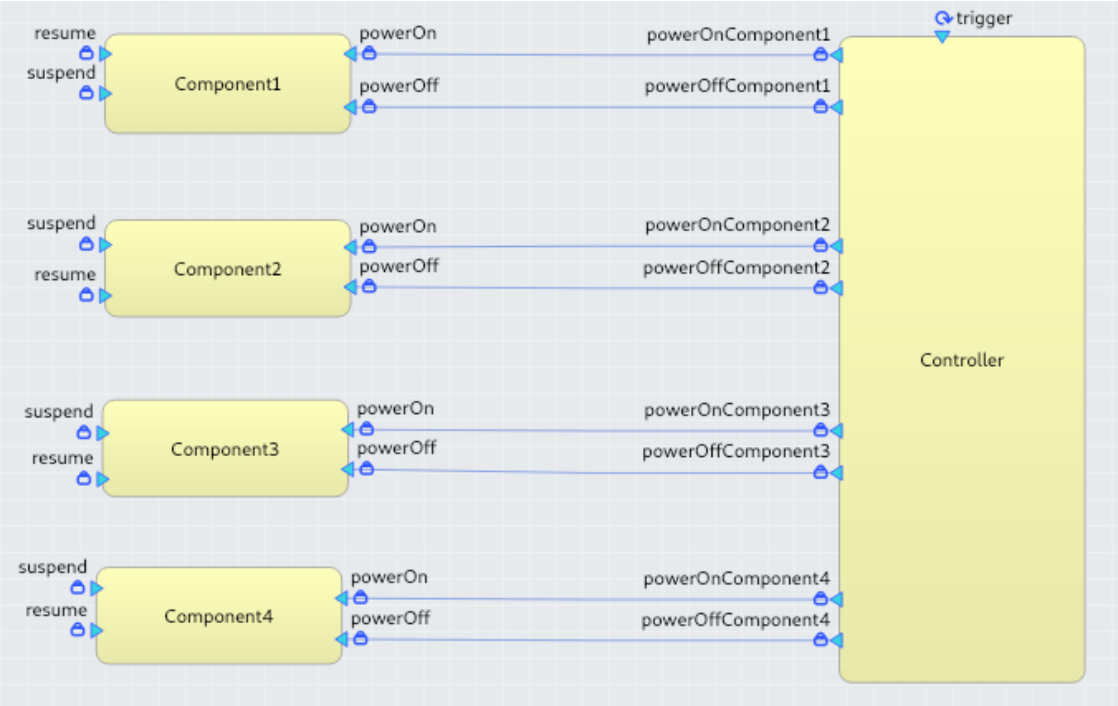

Figure 15 Example of component management via a dedicated controller component

```
<ArchetypeLibrary
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://www.example.org/ArchetypeLibrary"
    xsi:schemaLocation="http://www.example.org/ArchetypeLibrary ArchetypeLibrarySchema.xsd">

        <FunctionArchetype name="Dynamic_component">
                <InterfaceArchetype name="start" interfaceType="PROVIDED" kind="PROTECTED" layer="Component_management">
                </InterfaceArchetype>
                <InterfaceArchetype name="stop" interfaceType="PROVIDED" kind="PROTECTED" layer="Component_management">
                </InterfaceArchetype>
                <InterfaceArchetype name="resume" interfaceType="PROVIDED" kind="PROTECTED" layer="Component_management">
                </InterfaceArchetype>
                <InterfaceArchetype name="suspend" interfaceType="PROVIDED" kind="PROTECTED" layer="Component_management">
                </InterfaceArchetype>
        </FunctionArchetype>

        <CommunicationLayerTypes>
                <CommunicationLayerType name="Component_management"/>
        </CommunicationLayerTypes>
</ArchetypeLibrary>
```

Figure 16 Archetypes for component management

```
<ArchetypeLibrary
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://www.example.org/ArchetypeLibrary"
    xsi:schemaLocation="http://www.example.org/ArchetypeLibrary ArchetypeLibrarySchema.xsd">

        <FunctionArchetype name="Powered_Component">
            <InterfaceArchetype name="powerOn" interfaceType="PROVIDED" kind="PROTECTED" layer="Component_management">
            </InterfaceArchetype>
            <InterfaceArchetype name="powerOff" interfaceType="PROVIDED" kind="PROTECTED" layer="Component_management">
            </InterfaceArchetype>
            <InterfaceArchetype name="resume" interfaceType="PROVIDED" kind="PROTECTED" layer="Component_management">
            </InterfaceArchetype>
            <InterfaceArchetype name="suspend" interfaceType="PROVIDED" kind="PROTECTED" layer="Component_management">
            </InterfaceArchetype>
        </FunctionArchetype>

        <CommunicationLayerTypes>
            <CommunicationLayerType name="Component_management"/>
        </CommunicationLayerTypes>
</ArchetypeLibrary>
```

Figure 17 Archetypes for power management

```
void manager_startup(void)
{
    print_rtems("Initialization\n");
    Reset_Reason reason = Cbi_Partition_Api_getResetReason();
    switch (reason)
    {
    case Reset_Reason_Power:
        print_rtems("ResetReason: POWER\n");
        startup_on_power_reset();
        return;
    case Reset_Reason_Reset:
        print_rtems("ResetReason: RESET\n");
        startup_on_processor_reset();
        return;
    case Reset_Reason_Unknown:
        print_rtems("ResetReason: UNKNOWN\n");
        break;
    }
}
```

Figure 18 Example of query for a reset reason

Table 4 Component Management requirements

ID	Requirement	Validated by
COMP_MANA_AURORA_0001	The component model shall provide the capability to perform a startup of itself on power-on res	M
COMP_MANA_AURORA_0002	The component model shall provide the capability to perform a startup of itself on processor reset	M
COMP_MANA_AURORA_0003	The component model shall provide the capability to specify a set of interfaces to allow starting, stopping, suspending, and resuming behaviors	M

ID	Requirement	Validated by
COMP_MANA_AURORA_0004	The component model shall provide support to specify behaviors for the order of startup and shutdown of components	M
COMP_MANA_AURORA_0005	The component model shall provide the capability to specify a set of interfaces for handling Power modes management	M

6.3.5. CBI Runtime

TASTE Leon3 Runtime has been developed and is available as GR712RC RTEMS6 SMP QDP within Deployment View (see Figure 19). In addition to supporting the previously described communication patterns, it includes support for performance monitoring, as illustrated in Figure 20.

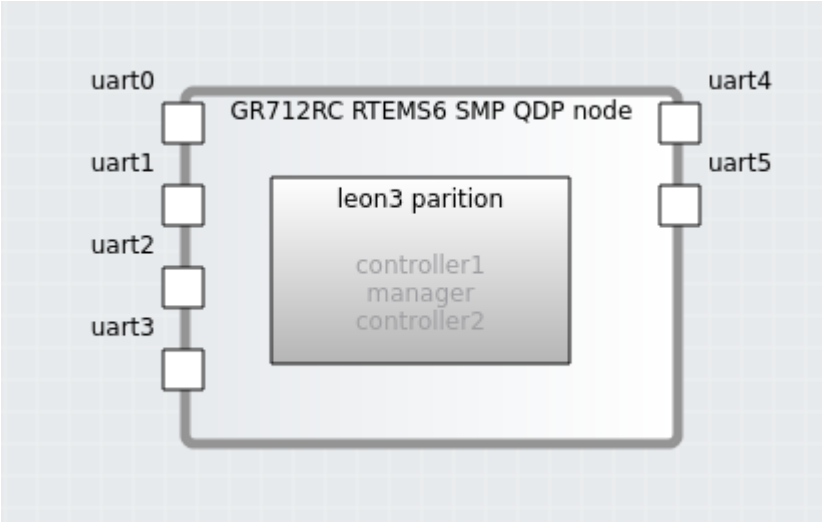

Figure 19 Example deployment view with TASTE Leon3 Runtime

```
const Interface_Usage_Data data_function_1_pi_1 =
    Perf_Mon_getUsageData(&G_perf_mon_ctx, function_1_pi_1, 1000000);
const Interface_Usage_Data data_function_1_pi_2 =
    Perf_Mon_getUsageData(&G_perf_mon_ctx, function_1_pi_2, 1000000);
```

Figure 20 Performance query example

Table 5 CBI Runtime requirements

ID	Requirement	Validated by
AUR-RT-0010	Architecture Messaging System included in the TASTE Leon3 Runtime shall support communication patterns defined in CBI Model	T
AUR-RT-0020	TASTE Leon3 Runtime shall target a Leon3 processor	D

ID	Requirement	Validated by
AUR-RT-0030	TASTE Leon3 Runtime shall target RTEMS version provided by ESA's RTEMS SMP QDP	D
AUR-RT-0040	TASTE Leon3 Runtime shall provide the reset reason	T
AUR-RT-0050	TASTE UART Communication Device Driver shall be provided for the TASTE Leon3 Runtime	D
AUR-RT-0060	TASTE Leon3 Runtime shall support timers with microsecond resolution	D
AUR-RT-0070	TASTE Leon3 Runtime shall support the measurement of minimum, maximum and average time spent by a provided sporadic interface	T
AUR-RT-0110	TASTE Leon3 Runtime shall support the measurement of minimum, maximum and average time spent by a provided cyclic interface	T
AUR-RT-0120	TASTE Leon3 Runtime shall support the measurement of minimum, maximum and average number of system ticks spent by a provided sporadic interface	T
AUR-RT-0130	TASTE Leon3 Runtime shall support the measurement of minimum, maximum and average number of system ticks spent by a provided cyclic interface	T
AUR-RT-0140	TASTE Leon3 Runtime shall support the measurement of minimum, maximum and average CPU load	T

6.3.6. Reference Component Set and DataStore

Requirements for the Reference Component Set were used to design and test the Reference Component Set described in chapter 6.2.

Table 6 Reference Component Set and DataStore requirements

ID	Requirement	Validated by
DATA_STORE_AURORA_0001	The component model shall support Data Store Subsystem with publisher and subscriber capabilities	M
DATA_STORE_AURORA_0002	The component model shall support Data Store Subsystem with a Data Interface to interact with the stored data	M
DATA_STORE_AURORA_0003	The Data Interface shall provide the capability to save data in memory	M
DATA_STORE_AURORA_0004	The Data Interface shall provide the capability to read data in memory	M
DATA_STORE_AURORA_0005	The Data Interface shall provide the capability to update data in memory	M
DATA_STORE_AURORA_0006	The Data Interface shall provide the capability to delete data in memory	M

ID	Requirement	Validated by
DATA_STORE_AURORA_0007	The Data Interface shall ensure data consistency	D
DATA_STORE_AURORA_0008	The Data Store Subsystem should have a memory entries log	M
AUR-CBI-MDL-0010	CBI Reference Component Set shall include a TimeService component	E
AUR-CBI-MDL-0020	TimeService shall be compatible with TASTE Leon3 Runtime	E
AUR-CBI-MDL-0030	TimeService component shall provide an unprotected interface for querying OBET time, encoded as an integer number of elapsed nanoseconds	E
AUR-CBI-MDL-0040	TimeService component shall provide an unprotected interface for querying clock status	E
AUR-CBI-MDL-0050	TimeService component shall provide an unprotected interface for converting the integer number of elapsed nanoseconds into a PTC 10 (relative time) PFC 18 (full CUC format) timestamp	E
AUR-CBI-MDL-0070	TimeService component shall provide an unprotected interface for performing comparisons of CUC timestamps	E
AUR-CBI-MDL-0080	TimeService component shall provide an unprotected interface for performing comparisons of CFS Time Format timestamps	E
AUR-CBI-MDL-0090	CBI Reference Component Set shall include a DataStore component	E
AUR-CBI-MDL-0100	DataStore component shall be compatible with any CBI compliant runtime	E
AUR-CBI-MDL-0110	DataStore component shall store a configurable number of DataItems	E
AUR-CBI-MDL-0120	The type of the DataItems stored by the DataStore component shall be user configurable	E
AUR-CBI-MDL-0130	The DataStore component shall have the capability to automatically remove the oldest DataItem when it is full and receiving a new DataItem	E
AUR-CBI-MDL-0140	It shall be possible to configure the DataStore to either automatically remove the oldest DataItem or reject the new DataItem when it is full	E
AUR-CBI-MDL-0150	The DataStore component shall have the capability to report an Event when automatically removing a DataItem	E
AUR-CBI-MDL-0160	The DataStore component shall have the capability to report an Event when rejecting a new DataItem	E

ID	Requirement	Validated by
AUR-CBI-MDL-0170	The DataStore component shall timestamp all received DataItems using the time of item reception	E
AUR-CBI-MDL-0180	The DataStore component shall have the capability to receive a new DataItem	E
AUR-CBI-MDL-0190	Upon successful reception of a new DataItem, the DataStore component shall return the storage key of the DataItem	E
AUR-CBI-MDL-0200	The DataStore component shall have the capability to retrieve a DataItem by key	E
AUR-CBI-MDL-0210	The DataStore component shall have the capability to update a DataItem using its key	E
AUR-CBI-MDL-0220	The DataStore component shall have the capability to delete a DataItem using its key	E
AUR-CBI-MDL-0230	Key used to identify a stored DataItem shall not be changed by any operation on other DataItems (deletion, updated, addition)	E
AUR-CBI-MDL-0240	The DataStore component shall have the capability to retrieve DataItems by time range	E
AUR-CBI-MDL-0250	The DataStore component shall have the capability to clear its contents	E

6.3.7. AOCS/GNC SW components

The capability to integrate AOCS/GNC products within TASTE systems is demonstrated by a modified version of the ACS model provided by UPM on TASTE Wiki [RD7]. The model includes an additional GUI for controlling and inspecting the simulation, as depicted in Figure 21. The Simulink model responsible for actual control algorithm is presented in Figure 22, and output of the simulation is presented in Figure 23.

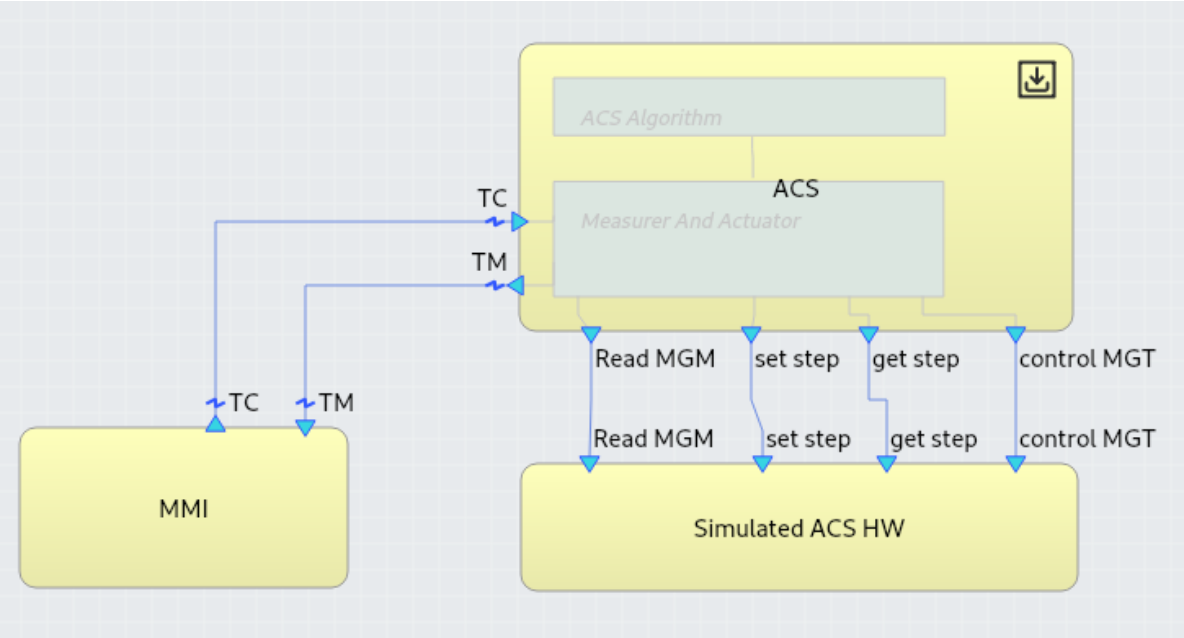

Figure 21 Example ACS algorithm integration within TASTE system

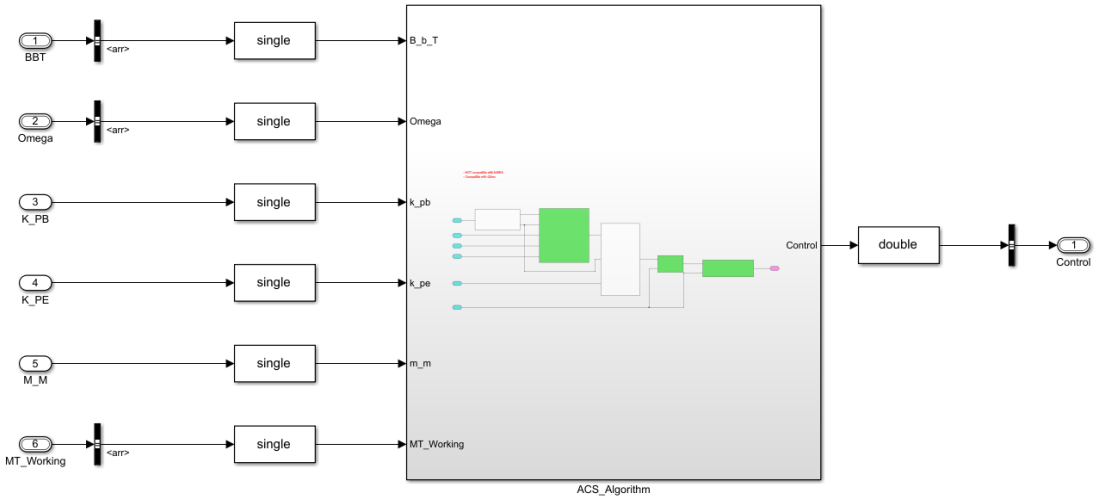

Figure 22 Example ACS algorithm

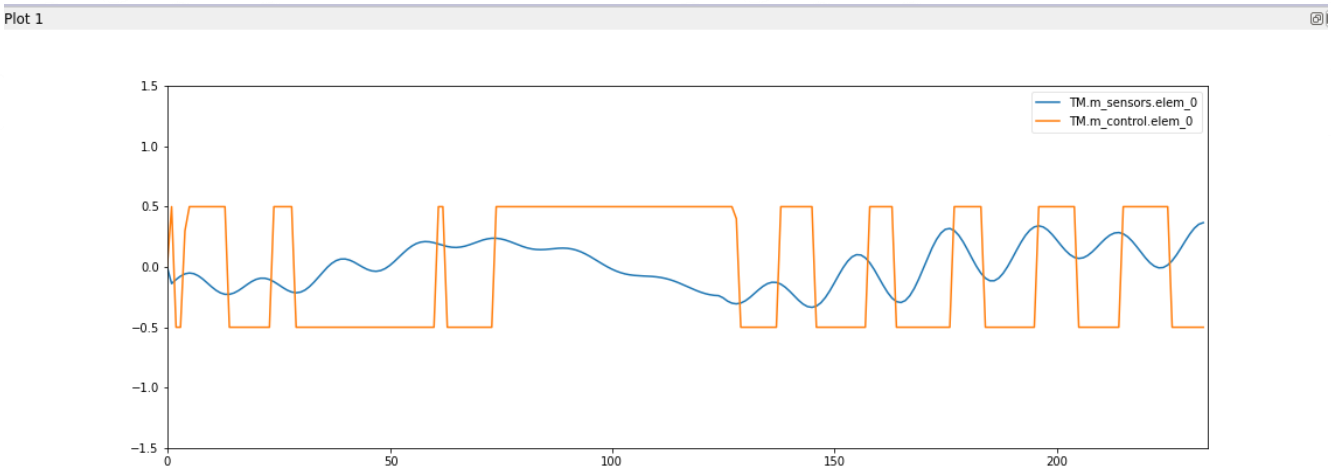

Figure 23 Example output of the ACS algorithm

Table 7 AOCS/GNC SW components requirements

ID	Requirement	Validated by
AOCS-GNC_AURORA_0001	CBI shall support the information exchanged at runtime to support interoperability between SW applications and AOCS/GNC product	M

6.3.8. General

Figure 24 presents a component which contains internal data which can be accessed by an external component. The decision which, how and when data is accessible is defined by the user.

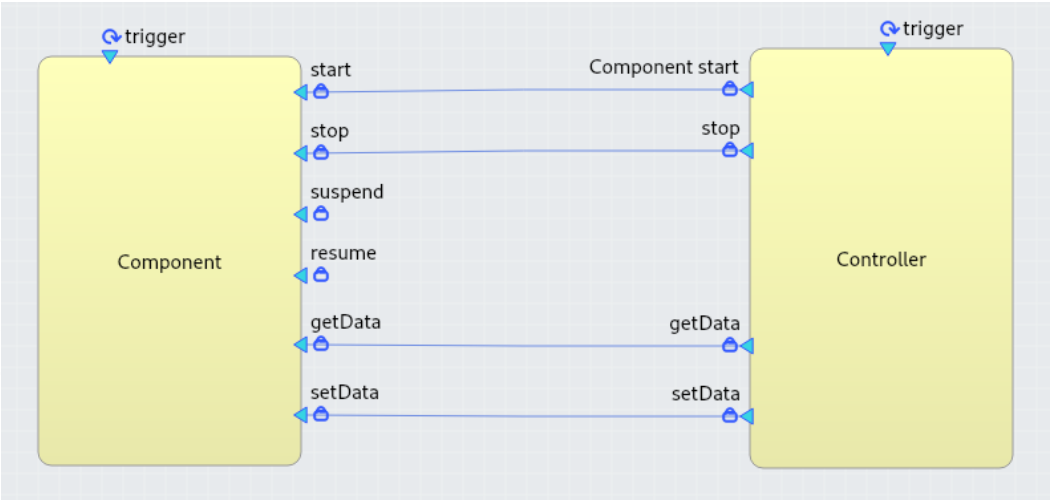

Figure 24 Example component with data accessors

Table 8 General requirements

ID	Requirement	Validated by
GENERAL_AURORA_0001	The components shall have the capability to have data structures of their own	M

6.3.9. Requirements for Tool-suite integration

Multiple new functionalities were added to TASTE in order to support the creation of safety critical software for use in aerospace and robotics. Functions described using Simulink models can now use required protected interfaces, as long as the selected code generator is QGenC (see Figure 25). The protected interfaces are exposed to the models via S-Functions (see Figure 26).Similarly, access to input and output vectors of Simulink models can be split into multiple interfaces, separately for input updates, output queries and step invocation (see Figure 27 and Figure 28).

While traditionally TASTE is responsible for defining the interfaces of Simulink models and generating relevant stubs, Simulink Importer created within the scope of AURORA project allows to import the interfaces of an existing Simulink model and integrate it within TASTE system, enabling more flexible workflows (see Figure 29). It shall be noted however, that that only a subset of Simulink interface definitions is supported. In particular, arrays are not supported, as they would require significant, potentially breaking changes to TASTE.

Components defined in TASTE (see Figure 30) can be used outside of the TASTE ecosystem, within stand-alone applications (see Figure 31), provided that the necessary glue code and data definitions are provided (as described in RD4).

Protected provided interfaces of Components can be validated against sets of reference test vectors using Function Tester (see Figure 32). The data is to be provided via CSV files (at this moment, only integer, floating-point and boolean values are supported), and the Components can be exercised both on host system (using TASTE Linux Runtime), and target embedded platform (using TASTE Leon3 Runtime), as seen in Figure 33. The execution settings can be customized (see Figure 34). It should be noted that only the two abovementioned runtimes were tested, and within SpaceCreator they are available under the names of x86 Linux Cpp and GR712RC RTEMS6 SMP QDP respectively.

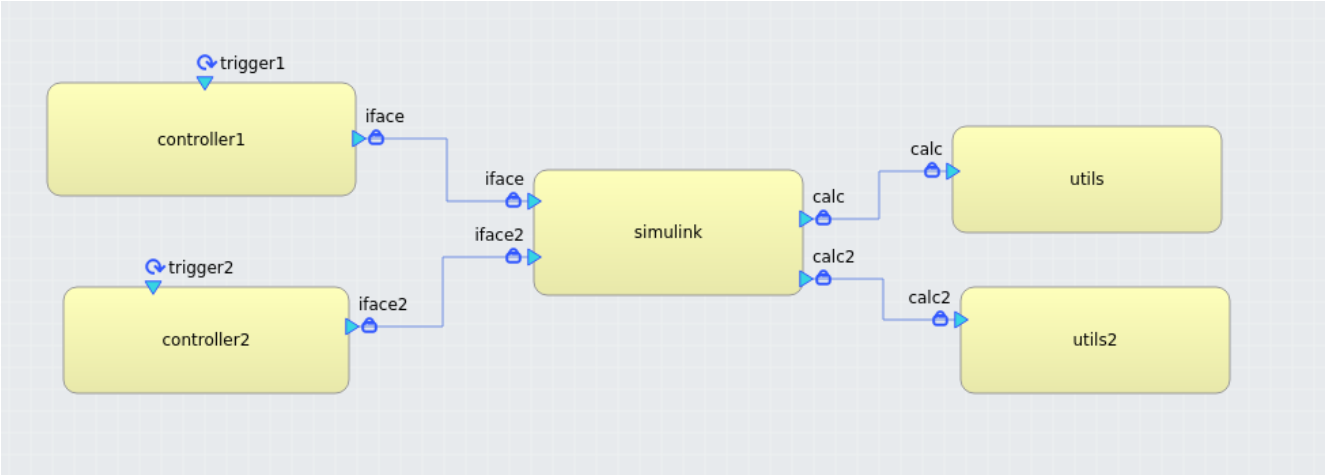

Figure 25 Example QGenC based function (simulink) with 2 required interfaces

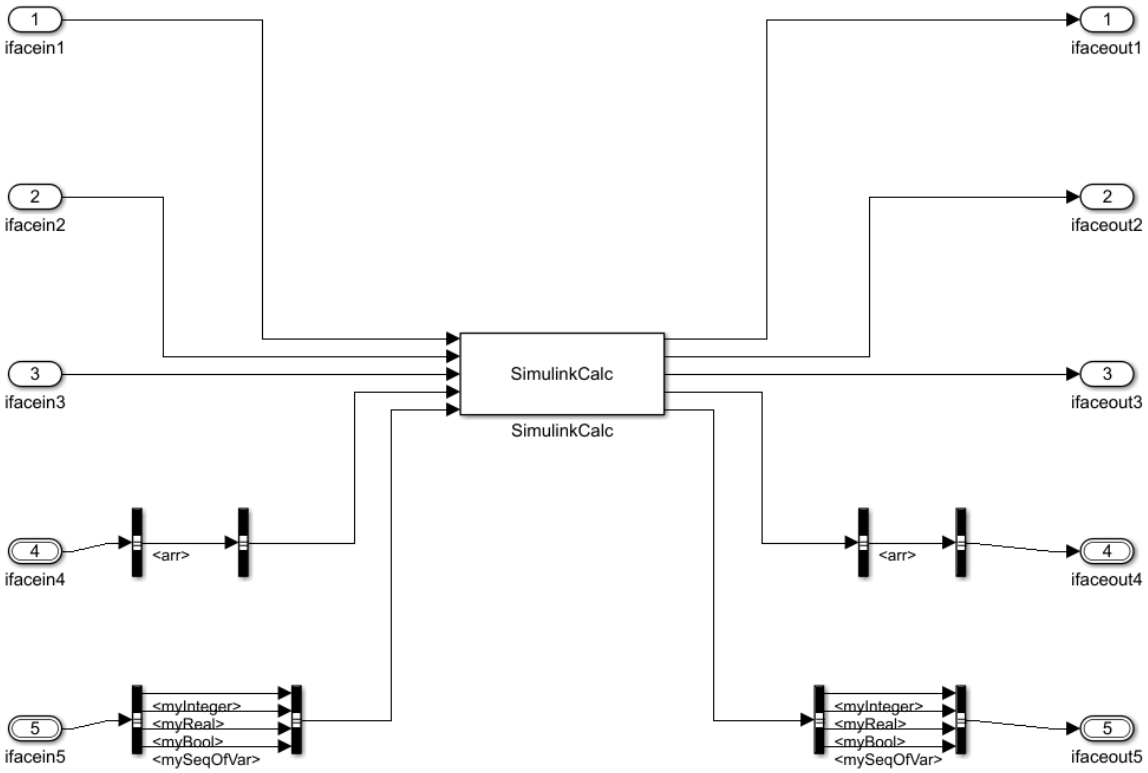

Figure 26 Trivial Simulink model using a required interface via S-Function

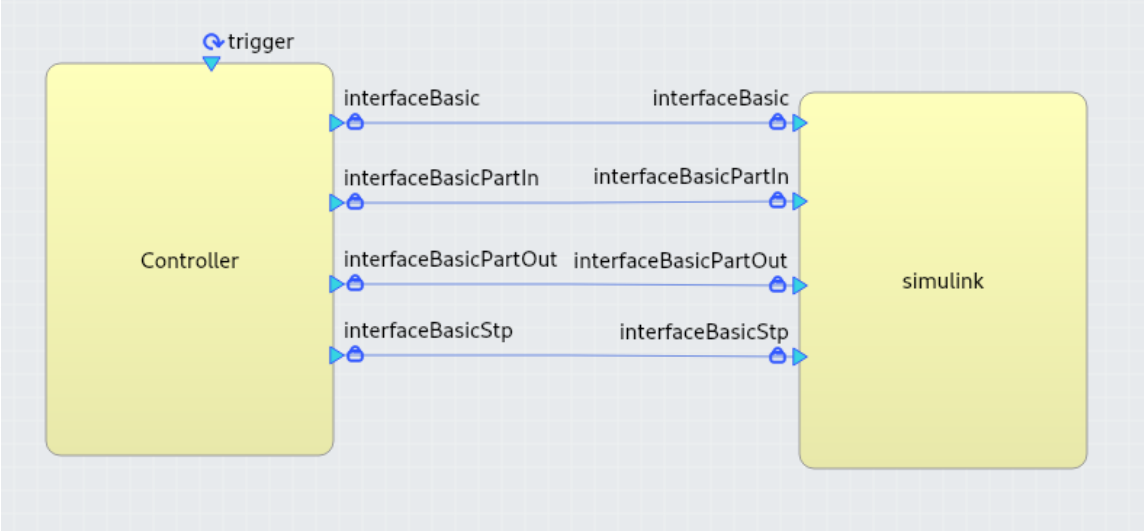

Figure 27 Example Simulink component which provides interfaces for partial vector update and query

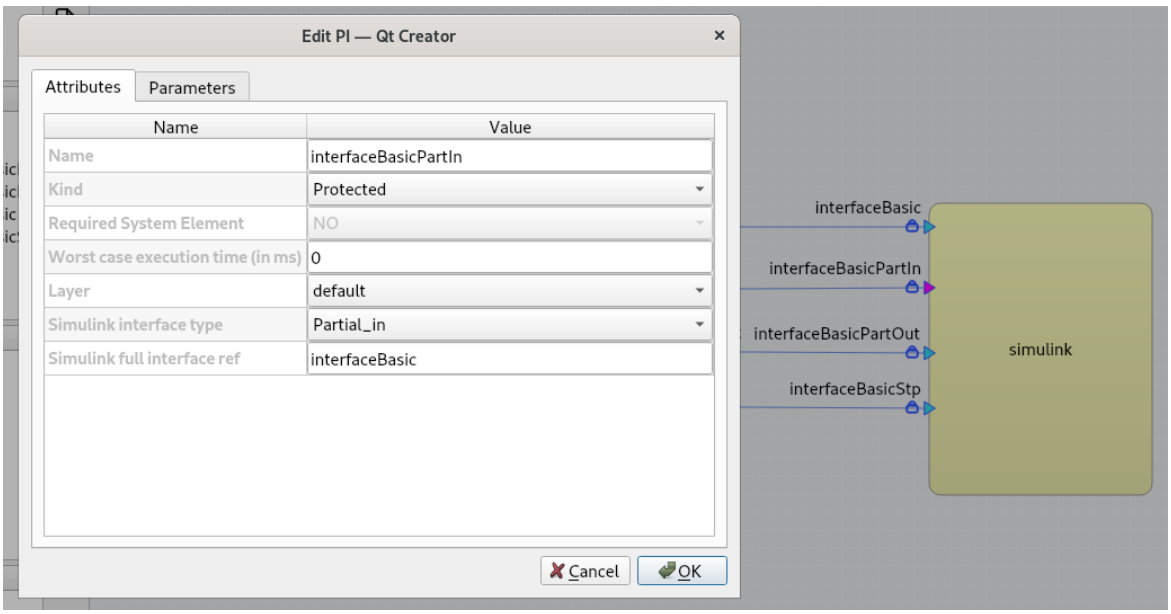

Figure 28 Additional properties used to define the interfaces for partial vector update and query

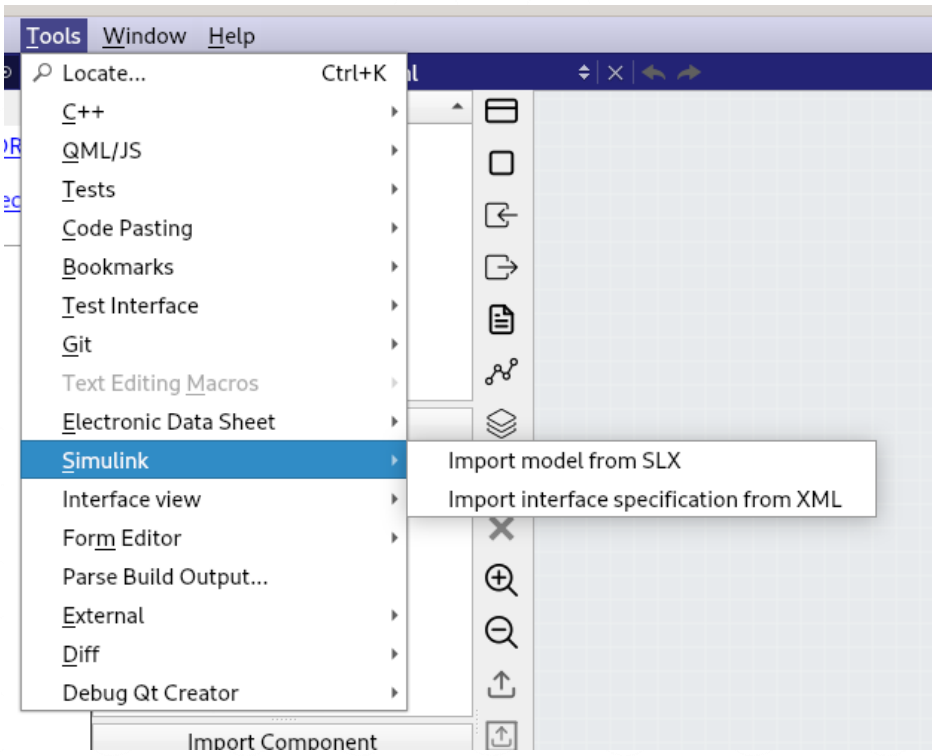

Figure 29 Simulink Importer integration within SpaceCreator

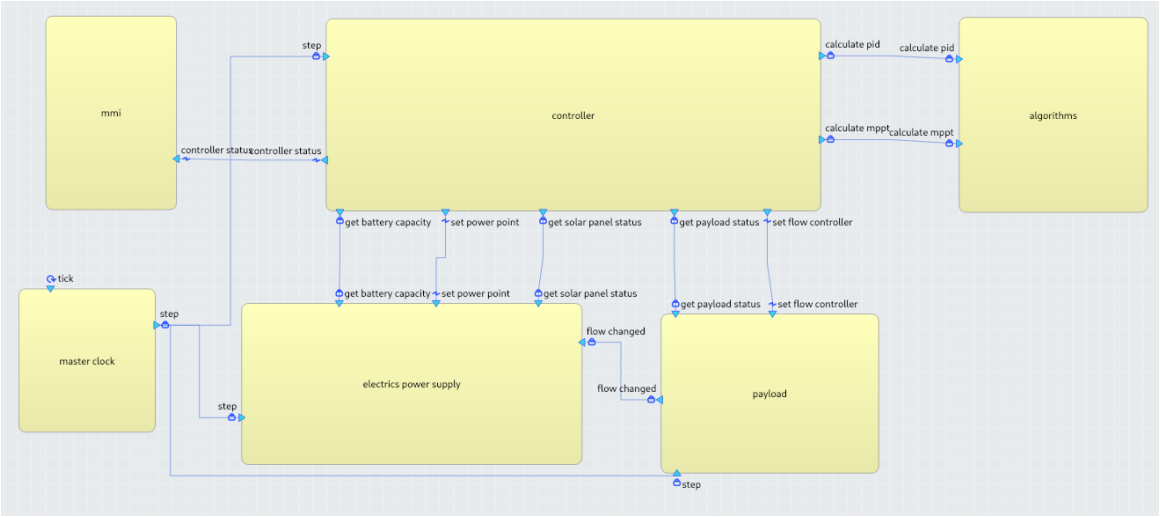

Figure 30 Example TASTE model which contains 2 components (one defined in C and one in Simulink) to be extracted for use independently of TASTE

```
[TASTE] Initialization completed for function algorithms
[TASTE] Initialization completed for function controller
Status | Temp: 305.000000 | Flow: 0.000000 | Capacity: 0.480000 | Voltage: 5.000000 | Current: 0.000000
Status | Temp: 303.983942 | Flow: 0.800000 | Capacity: 0.480880 | Voltage: 3.843014 | Current: 0.192056
Status | Temp: 302.798633 | Flow: 0.502569 | Capacity: 0.481801 | Voltage: 3.298815 | Current: 0.164355
Status | Temp: 301.797726 | Flow: 0.638741 | Capacity: 0.482703 | Voltage: 3.230626 | Current: 0.161303
Status | Temp: 300.929145 | Flow: 0.737967 | Capacity: 0.483591 | Voltage: 3.018502 | Current: 0.150738
Status | Temp: 300.235671 | Flow: 0.842467 | Capacity: 0.484464 | Voltage: 2.459778 | Current: 0.122893
Status | Temp: 299.739489 | Flow: 0.939132 | Capacity: 0.485323 | Voltage: 2.352066 | Current: 0.117589
Status | Temp: 299.413459 | Flow: 0.822078 | Capacity: 0.486200 | Voltage: 2.477394 | Current: 0.123862
Status | Temp: 299.333809 | Flow: 0.859146 | Capacity: 0.487071 | Voltage: 2.752394 | Current: 0.137602
```

Figure 31 Example output of a standalone application, independent of TASTE runtime, integrating TASTE components

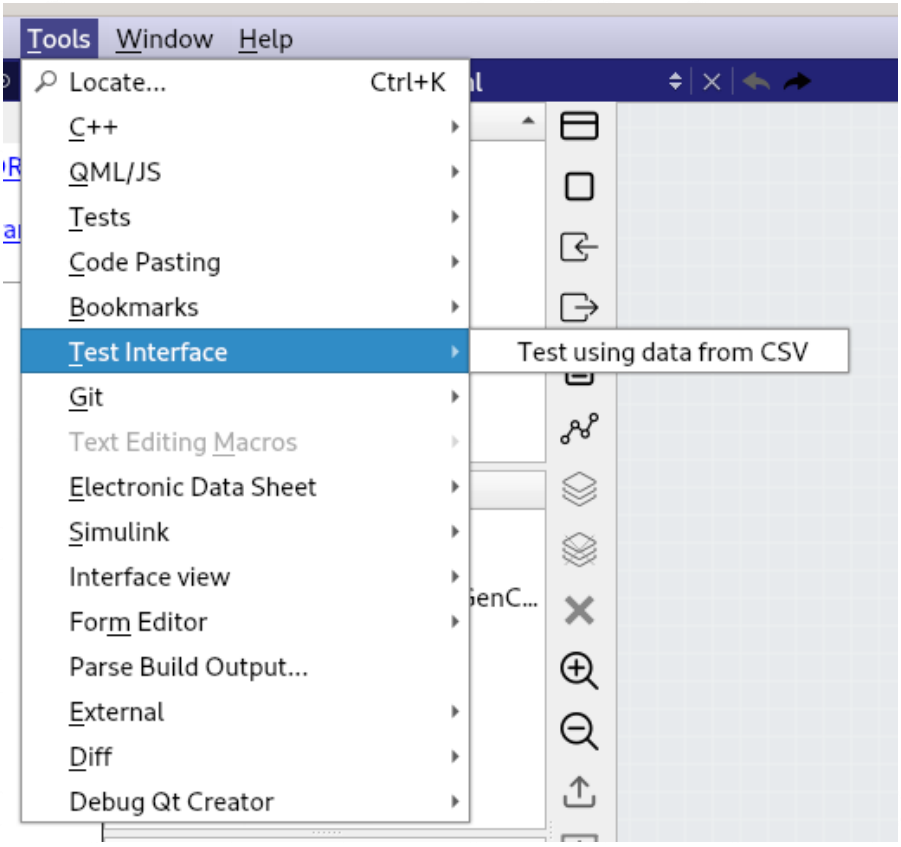

Figure 32 Function Tester integration within SpaceCreator

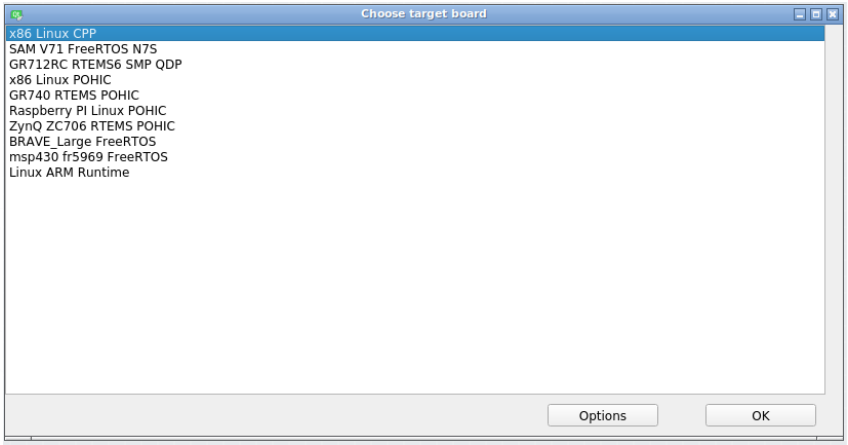

Figure 33 Target platform selection for Function Tester

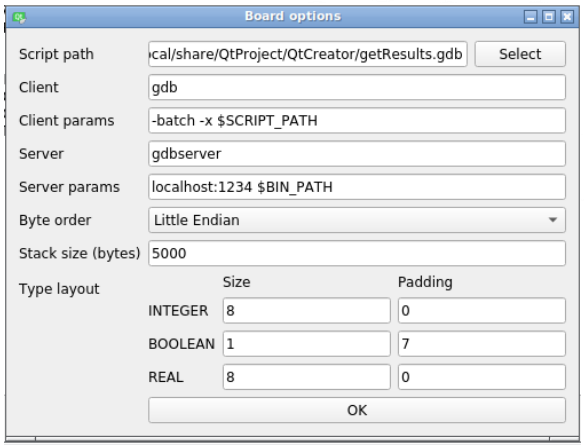

Figure 34 Target platform execution configuration for Function Tester

Table 9 Tool-suite integration requirements

ID	Requirement	Validated by
AUR-CBI-INT-0010	TASTE Interface View capabilities shall be extended to support CBI Model	M,I
AUR-CBI-INT-0020	It shall be possible to serialize CBI Model compliant Interface View using AADL	T
AUR-CBI-INT-0030	The CBI Model design shall be based on AADL concepts	D
AUR-CBI-INT-0035	The CBI Model design shall consider AADL constraints	D
AUR-CBI-INT-0040	SpaceCreator shall be used for CBI Model compliant Interface View definition	D
AUR-CBI-INT-0050	SpaceCreator shall support QGenC Function type, with behavior specification to be described via a Simulink model, and intermediate C code generation via Qgen	M,D
AUR-CBI-INT-0060	It shall be possible to automatically generate a Simulink model skeleton for TASTE QGenC Function	T
AUR-CBI-INT-0065	It shall be possible to import a Simulink model into TASTE Interface View as a TASTE QGenC Function	T
AUR-CBI-INT-0070	It shall be possible to include a CBI Model compliant component in a standalone application that does not depend on a TASTE runtime	T
AUR-CBI-INT-0080	TASTE QGenC Function shall support one provided protected interface, with input and output arguments, which updates the entire input vector, performs a processing step, and exposes the output vector	M
AUR-CBI-INT-0090	TASTE QGenC Function shall support required protected interfaces with input and output arguments	M
AUR-CBI-INT-0130	TASTE QGenC Function shall support ASN.1 Real data type	T

ID	Requirement	Validated by
AUR-CBI-INT-0140	TASTE QGenC Function shall support ASN.1 Integer data type	T
AUR-CBI-INT-0150	TASTE QGenC Function shall support ASN.1 Boolean data type	T
AUR-CBI-INT-0160	TASTE QGenC Function shall support ASN.1 fixed size Sequence Of data type	T
AUR-CBI-INT-0170	TASTE QGenC Function shall support ASN.1 Sequence data type	T
AUR-CBI-INT-0180	TASTE QGenC Function shall support ASN.1 IA5String data type	T
AUR-CBI-INT-0190	It shall be possible to define a protected provided interface for a TASTE QGenC Function to perform partial input vector update	M
AUR-CBI-INT-0200	It shall be possible to define a protected provided interface for a TASTE QGenC Function to perform partial output vector query	M
AUR-CBI-INT-0210	It shall be possible to define an independent protected provided interface for a TASTE QGenC Function to perform a processing step using the current input vector values	M
AUR-CBI-INT-0220	It shall be possible to test the code generated for a TASTE QGenC Function within TASTE host environment by providing a set of reference inputs and outputs	T
AUR-CBI-INT-0230	It shall be possible to test the code generated for a TASTE QGenC Function within the target environment by providing a set of reference inputs and outputs	T

6.3.10. Summary

The requirement coverage summary is presented in Table 10. It should be noted that as some requirements have more than one validation method assigned, the sum of requirements validated by different methods exceeds the total number of requirements.

Table 10 Requirement coverage summary

Type	Count	Percent of total
All	90	100%
Validated	90	100%
Validated by model	38	42%
Validated by executable model	24	27%
Validated by test	19	21%
Validated by inspection	2	2%
Validated by design	9	10%

7. Recommendations

7.1. General

This chapter presents the recommendations for future activities that could benefit the TASTE toolchain and its users.

7.2. Interface View

Archetypes, as currently implemented, provide the definition of component interfaces relying on externally supplied ASN.1 types. This allows flexibility but introduces some burden on the users and does not guarantee full interface compatibility between separately developed systems sharing the same archetype libraries. The capability to optionally include ASN.1 (or SEDS, if XML format is preferred) type definitions would mitigate these issues.

Currently, archetype libraries need to be coded defined manually directly in XML. Capability to extract archetype definition automatically from Interface View, based on an existing component definition, would make the process faster and more user friendly. Additional visual editor could be also considered.

Interface layers allow to logically group interfaces (and the relevant connections), both for user convenience (to increase diagram clarity) and possible analyses. While the layer visibility can be turned on and off, it would be beneficial to provide visual distinction for the different visible layers, for example in the form of user-configurable colours.

Logical architecture is currently stored in a monolithic Interface View, affecting the possibilities of component sharing and re-using, as well as concurrent development. The capability to import and export parts of the Interface View mitigates this issue, but only to a small extent, as it is mainly applicable to waterfall-like approaches and introduces significant overhead in agile environments.

7.3. Function Tester

The current implementation of the Function Tester allows testing of only protected provided interfaces of components that do not contain required interfaces. The focus during the design was on testing components with behaviours specified via Simulink models, which could not posses required or sporadic interfaces. However, as the Function Tester can be used for components specified using different languages, and – due to the developments executed during the AURORA project – the Simulink models can now posses required interfaces if QGenC is selected as the code generator, this limitation can be constraining.

Additionally, the current implementation supports only scalar types. Support for complex types would make the solution applicable to a broader range of use-cases.

7.4. Simulink Importer

The current Simulink Importer implementation does not support arrays due to the difficulties in bi-directional mapping of ASN.1 sequences of and Simulink buses/arrays. The necessary changes would be effort-intensive and compatibility breaking. However, they could be reconsidered in the future.

7.5. Runtime

TASTE Leon3 Runtime was developed with the help of Sparc Instruction Simulator (SIS) [RD14], which was extended with proper support for GR712RC Timer and UART modules. The open-source nature of SIS allowed to extend it and use it in the publicly available TASTE repositories, as well as easily scale the number of concurrently

working developers. Such things are either impossible or costly due to the licensing restrictions of commercially available solutions. It could be beneficial to extend SIS with more GR712RC or GR740 peripherals to lower the costs of development for Leon processors and support continuous integration efforts in open-source projects.

Alternatively, it could be considered to choose and extend a different open-source simulator or emulator, e.g., QEMU or Renode.

Currently, TASTE Leon3 Runtime provides only UART communication device driver for inter-partition communication. However, the target GR712RC processor includes also SpaceWire, Ethernet and CAN links, which could also be used for communication. Additional drivers, especially for SpaceWire and Ethernet, could be very useful.

One of the barriers for mission deployment of systems developed in TASTE is the adherence to ECSS standards and proper quality assurance. ESA's RTEMS SMP QDP operating system was chosen as the base for TASTE Leon3 Runtime with this in mind. However, a pre-qualification package for the entire runtime could increase its quality and broaden TASTE appeals for use in actual missions.

8. Lists

8.1. List of Tables

Table 1 N to M asynchronous communication requirements 19
Table 2 Fault Detection requirements 20
Table 3 Events requirements 23
Table 4 Component Management requirements 25
Table 5 CBI Runtime requirements 26
Table 6 Reference Component Set and DataStore requirements 27
Table 7 AOCS/GNC SW components requirements 31
Table 8 General requirements 32
Table 9 Tool-suite integration requirements 37
Table 10 Requirement coverage summary 38
Table 11 Code metrics 43

8.2. List of Figures

Figure 1 Interface View of a simple system (graphical representation of XML file) 10
Figure 2 Deployment View of a simple system (graphical representation of XML file) 10
Figure 3 Data View of a simple system (ASN.1 form) 10
Figure 4 Component definition workflow 11
Figure 5 TimeService component 16
Figure 6 DataStore component 17
Figure 7 Example model with N to M asynchronous communication 18
Figure 8 MMI generated by TASTE 18
Figure 9 Example message with a timestamp 18
Figure 10 Archetypes for error communication 20
Figure 11 Communication error callback registration 20
Figure 12 Archetypes for event communication 22
Figure 13 Example user implementation of an event bus for subscriber-publisher pattern 22
Figure 14 Example usage of CBI API for altering the message routing within the runtime glue code 23
Figure 15 Example of component management via a dedicated controller component 24
Figure 16 Archetypes for component management 24
Figure 17 Archetypes for power management 25
Figure 18 Example of query for a reset reason 25

Figure 19 Example deployment view with TASTE Leon3 Runtime 26
Figure 20 Performance query example 26
Figure 21 Example ACS algorithm integration within TASTE system 30
Figure 22 Example ACS algorithm 30
Figure 23 Example output of the ACS algorithm 31
Figure 24 Example component with data accessors 31
Figure 25 Example QGenC based function (simulink) with 2 required interfaces 32
Figure 26 Trivial Simulink model using a required interface via S-Function 33
Figure 27 Example Simulink component which provides interfaces for partial vector update and query 33
Figure 28 Additional properties used to define the interfaces for partial vector update and query 34
Figure 29 Simulink Importer integration within SpaceCreator 34
Figure 30 Example TASTE model which contains 2 components (one defined in C and one in Simulink) to be extracted for use independently of TASTE 35
Figure 31 Example output of a standalone application, independent of TASTE runtime, integrating TASTE components 35
Figure 32 Function Tester integration within SpaceCreator 36
Figure 33 Target platform selection for Function Tester 36
Figure 34 Target platform execution configuration for Function Tester 37

9. Annex A – Code metrics

In addition to validation with respect to D5.1 [RD1] and in response to D2.3 [RD15], the following code metrics were gathered:

- Cyclomatic complexity,
- Nesting level (reported by tools as maximum indentation),
- Lines of code (LOC).

For new repositories (such as Leon3-BSP) the entire codebase has been measured. For existing repositories (such as TASTE-Linux-Runtime), which were extended in the scope of the AURORA project, the codebase has been measured before the start of the AURORA related modifications and after the modifications, so that two sets of values were produced – total and delta. It has to be noted however, that Kazoo and SpaceCreator repositories were being concurrently modified by different entities, within the scope of several projects, and the changes were impossible to be automatically separated. Therefore, the respective deltas also include external contributions unrelated to AURORA itself, and as such should be considered as overestimated approximations.

The gathered metrics are presented in Table 11.

Table 11 Code metrics

Repository	Cyclomatic complexity	Nesting	Non-empty LOC	Total LOC
Leon3-BSP	13	5	1720	2648
TASTE-LEON3-Drivers	6	3	278	395
TASTE-LEON3-Runtime	2	3	231	490
AURORA-Reference-Component-Set	8	5	1274	1651
AURORA-Validation	21	7	21768	27982
SIS (all)	325	7	15846	20972
SIS (new)	325	7	1056	1345
TASTE-Runtime-Common (all)	11	5	1034	1895
TASTE-Runtime-Common (new)	11	5	569	778
TASTE-Linux-Runtime (all)	7	3	347	16
TASTE-Linux-Runtime (new)	7	3	846	28
Kazoo (all)	49	6	61066	107796
Kazoo (new)	49	7	11305	15625
SpaceCreator (all)	129	10	103209	143785
SpaceCreator (new)	135	10	96324	144953
DMT (all)	87	N/A	33989	N/A
DMT (new)	87	N/A	91	N/A

Comments on method:

- For Leon3-BSP, TASTE-LEON3-Drivers, TASTE-LEON3-Runtime, AURORA-Reference-Component-Set, AURORA-Validation, TASTE-Linux-Runtime, code metrics have been prepared for the entire projects.
- For TASTE-Linux-Runtime, changes made in only one branch (n7s-aur#141-get-sender) were taken into account. For SIS, SpaceCreator, Kazoo, TASTE-Runtime-Common, two sets of metrics have been prepared

– one for the most recent state of the project (as of 6th March 2023) and one for the time of the first commit by N7S team. Because of that, there may be a small surplus of the reported amount of code written, as some of the changes might have been introduced by other teams working in parallel.
- Reports have been generated with metrixpp, except for DMT.
- For DMT code metrics have been prepared using lizard.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101004291