European Component Oriented Architecture (ECOA®) Collaboration Programme: Guidance Document: Container level checking and Time synchronization

Date: 29/08/2017

Prepared by
Dassault Aviation

Contents

0 Executive Summary

This document defines guidance to time synchronisation and container level QoS checking within an ECOA system.

1 Scope

This document is intended to provide guidance on container level checking and time synchronization.

The document is structured as follows:

Section 2 gives a brief introduction to the topic.

Section 3 expands abbreviations used in this report.

Section 4 provides definitions for the key terms used in this report.

Section 5 lists key documents referenced by this report.

Section 6 discusses time synchronization.

Section 7 discusses container level checking.

2 Introduction

This document defines guidance to time synchronisation and container level QoS checking within an ECOA system.

3 Abbreviations

API	Application Programming Interface
ASC	Application Software Component
DSTL	Defence Science and Technology Laboratory
ECCPF	ECOA Compliant Computing Platform
ECOA	European Component Oriented Architecture
ELI	ECOA Logical Interface
FR	French
IAWG	Industrial Avionics Working Group
I/O	Inputs-Outputs
OS	Operating System
PF	Platform
QoS	Quality of Service
RR	Request-Response
STD	Standard
TR	Technical Report
TRL	Technology Readiness Level
UDP	User Datagram Protocol
UK	United Kingdom
XML	eXtensible Markup Language

4 Definitions

For the purpose of this document, the definitions given in the ECOA Architecture Specification *(ref. [AS])* Part 2 and those given below apply.

Term	***Definition***
(currently none)	

5 References

AS	European Component Oriented Architecture (ECOA) Collaboration Programme: Architecture Specification (Parts 1 to 11) "ECOA" is a registered mark.

6 Guidance to Time Synchronisation

This section provides guidance to time synchronisation within an ECOA system.

Currently the ECOA Architecture Specifications does not define any way to synchronize time between elements of an ECOA system since it is considered that the decision as to whether time synchronisation is required is system specific. As a result, ECOA does not mandate a particular method for achieving time synchronisation. This allows the ECOA Compliant Computing PlatForm (ECCPF) suppliers to choose the most appropriate solution for their platforms.

Time synchronisation may be required at two levels:

- Between computing nodes of the same platform,
- Between platforms of the same system.

To synchronize computing node clocks within an ECOA Compliant Computing Platform, the platform provider is responsible of his design choices. As a consequence, he is required to provide time characteristics of his platform (maximum drift, precision between computing nodes, etc.).

To synchronize time between several ECOA Compliant Computing Platforms, there are multiple ways depending on physical choices made by the system designer.

For his design, the system designer may select one or many of the following solutions depending on his availability constraints:

- One given ECCPF is the clock master broadcasting time to other platforms through a standardized protocol (e.g. NTP, SNTP) or a specific one.
- One specific piece of equipment (e.g. GPS receiver time server, IRIG time code generator, NTP server) is the clock master broadcasting time to all platforms through a dedicated network or through the avionics network.
- One network piece of equipment (e.g. IEEE1588 switch) distributes time over the avionics network.

Those choices may lead to specific procurement requirements towards platform providers so that their platforms can be synchronized with one or many external clocks. Those requirements may cover interface definition, time scale, origin of time, expected accuracy of computing nodes against the external reference time, etc.

7 Guidance to Container-level QoS Checking

7.1 Introduction

This section provides guidance about what the container may check at its level based on information saved in the model.

It is important to note that Container-level QoS checking will result in additional overheads (e.g. CPU resources). As a consequence the system designer should take careful consideration regarding its use in an embedded system. For an on-ground reference platform, it is however recommended the container-level QoS checking should be run systematically.

In this guidance, two types of checking are considered:

- Validity of input and output data
- Validity of temporal characteristics

7.2 Validity of input and output data

7.2.1 Principles

The current metamodel allows strong typing of data by setting the basic type (byte, char8, etc.), the minimum value, the maximum value and the precision of each elementary data item (field, simple type, etc.).

The following XML snippet provides an example of such typing:

```
<simple type="double64" name="range" minRange="0.0" maxRange="12000.0" unit="NM"
        precision="10"/>
```

Then the container may check for each input or output parameter of an entry point or a container API if the value is within the defined range or if the value is rounded to the proper precision.

This check is useful when:

- Integrating and checking on-going developments in order to fix bugs,
- Integrating and running components of low trust in relation with other components.

In case of trusted components, the check should not be useful once the system has been qualified. The associated verification and validation shall have check that data coupling is correct. Moreover, if a software quality assurance standard has been followed, the likelihood that the application source checks by itself data validity is very high and in this case it is redundant with the container-level checking. So the system designer must clearly indicate where the data validity is done to avoid superfluous processing.

In case of non-trusted components, the check could be done near those components (either their container or the container of the components just beside); it is not necessary to handle in the same way all I/O data within an assembly mixing different levels of trust.

7.2.2 Fault handling

When the check succeeds, the execution flow may continue.

However, when the check fails, it is recommended that any container should carry on the processing of the current data/control flow (i.e. invoking the entry point with the faulty parameters or continuing the container API call) for the following rationale:

- Detecting a problem at this level does mean the cause of the problem is at this level. So the decision to invoke such or such recovery action should be left to a more intelligent entity such as the fault handler.
- This allows a reproducible behaviour between all platforms
- Real systems generally implement data validity at application source code level.

In addition, it should log the error and it should inform the fault handler (ILLEGAL_INPUT_ARGS, ILLEGAL_OUTPUT_ARGS error codes).

7.3 Validity of temporal characteristics

7.3.1 Principles

The current metamodel stores time information regarding service operations and request-response timeouts.

Service operation QoS defines the arrival law and the maximum expected processing time to handle an event or a request-response.

The client and the server of the same service instance may respectively request or provide different QoS; their containers may check them appropriately.

The client container may check that:

- the service operations sent at its level respect the expected arrival law (frequency, minimum inter-arrival time),
- the response is received before the max response time (time between the sending of the request and the local queuing of the response),
- the age of a versioned data, when accessed, does not exceed the max ageing (this means that the container should calculate and save locally the timestamp when the data is updated by a writer).
- The server container may check that:
    - the service operations arriving at its level respect the expected arrival law (frequency, minimum inter-arrival time),
    - the entry-point associated to an event finishes before the maximum handling time (time between the queuing of the event and the end of the entry-point),
- the response is sent before the max response time (time between the queuing of the request and the send of the response).

As a module entry-point can be connected simultaneously to service-level operations and module operations internal to the component through the same link and as there is currently no QoS information available regarding these later operations, the container implementation shall distinguish both kinds of operations in order to conditionally check the temporal characteristics of the service operations.

A platform supplier may enrich the checking by taking into account module-level behaviours, but these behaviours are currently not normative; they are provided as guidance.

7.3.2 Fault handling

When the check succeeds, the execution flow may continue.

However, when the check fails, it is recommended that any container should carry on the processing of the current data/control flow (i.e. invoking the entry point) for the following rationale:

- Detecting a problem at this level does mean the cause of the problem is at this level. So the decision to invoke such or such recovery action should be left to a more intelligent entity such as the fault handler.
- This allows a reproducible behaviour between all platforms
- Real systems implement temporal checks at application source code level.

In addition, it should log the error and it should inform the fault handler (OPERATION_OVERRATED, OPERATION_UNDERRATED error codes).