# RE: "Traditional" Software .v. ECOA Software

- The ECOA programme has pondered the advantages (or otherwise) of using the ECOA approach to s/w design and construction over "traditional" approaches used in the aerospace industry;

- These slides present *one way* the issue might be addressed, using a simple example;

- The idea is to show that the principals of a "traditional" high-integrity software methodology can be preserved, but still take on the componentization and re-use benefits of the ECOA;

- Assumptions:
  - "Traditional" high-integrity software means (in this context):
    - Single application on a single processing node;
    - Single threaded, fixed frame (deterministic) scheduling;
    - Separation of "application" code from "I/O" and "interface" code;
    - Representable in UML.

# Trad_v_ECOA

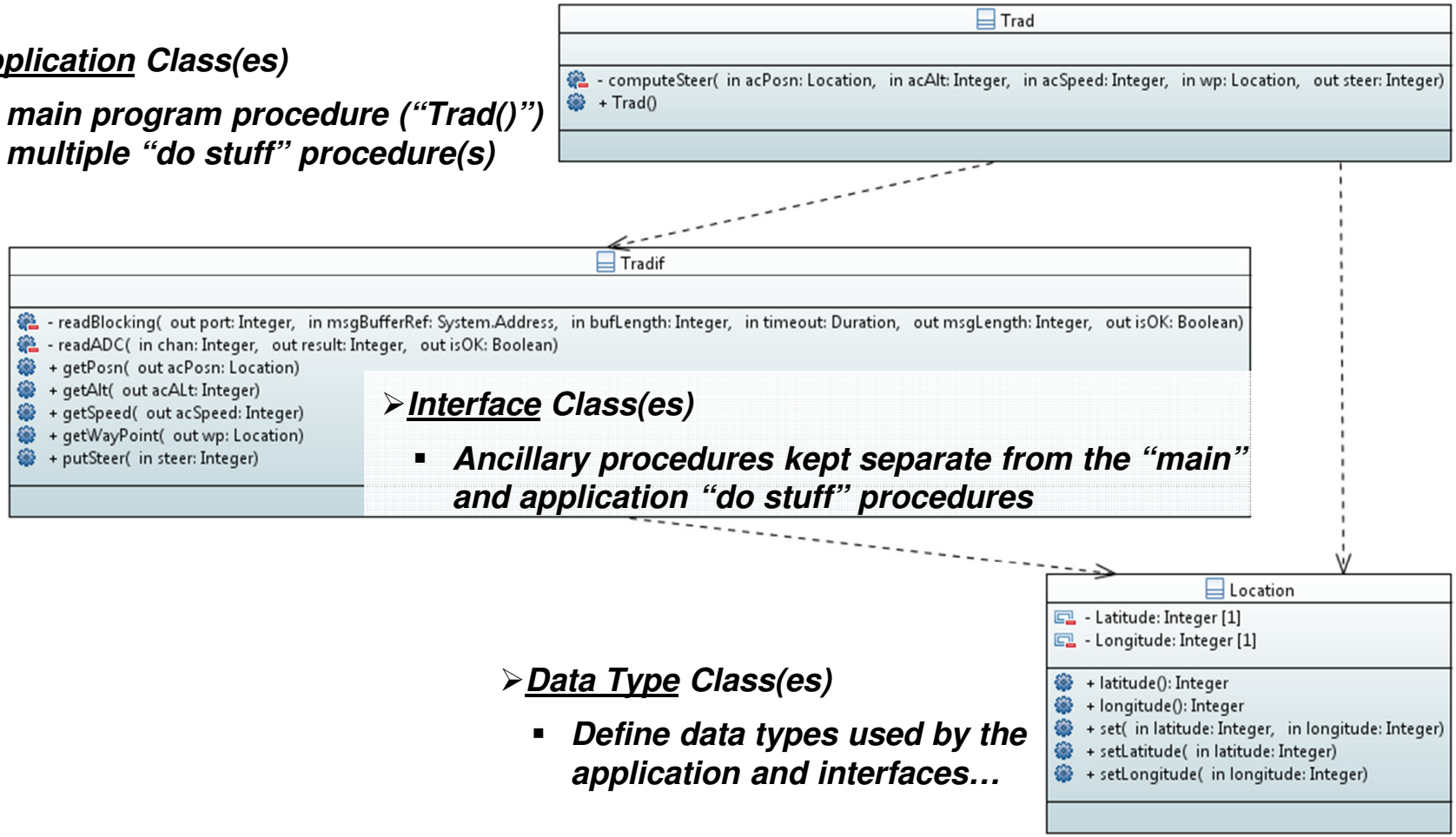
- This document presents information about some of the principal user generated artefacts required to compare a “traditional” aerospace approach with an ECOA approach.
- It is assumed that the reader is thoroughly conversant with the ECOA Architecture Specification (ref.[1]) and the process of defining and declaring ECOA Assemblies, ASCs (components), Modules, and deployments in XML, and then using code generation to produce Module framework (stub) code units and ECOA Container and Platform code.
- If not, then let me suggest working through some of the other examples/samples provided, starting with “*Hello World*” and working your way up to (say) “*Pub Sub*”.

# The Example "Traditional" Design

➢The design can be thought of as composed of three types of design class…

➢***Application* Class(es)**

- ***main program procedure ("Trad()")***
- ***multiple "do stuff" procedure(s)***




➢***Interface* Class(es)**

- ***Ancillary procedures kept separate from the "main" and application "do stuff" procedures***

➢***Data Type* Class(es)**

- ***Define data types used by the application and interfaces…***

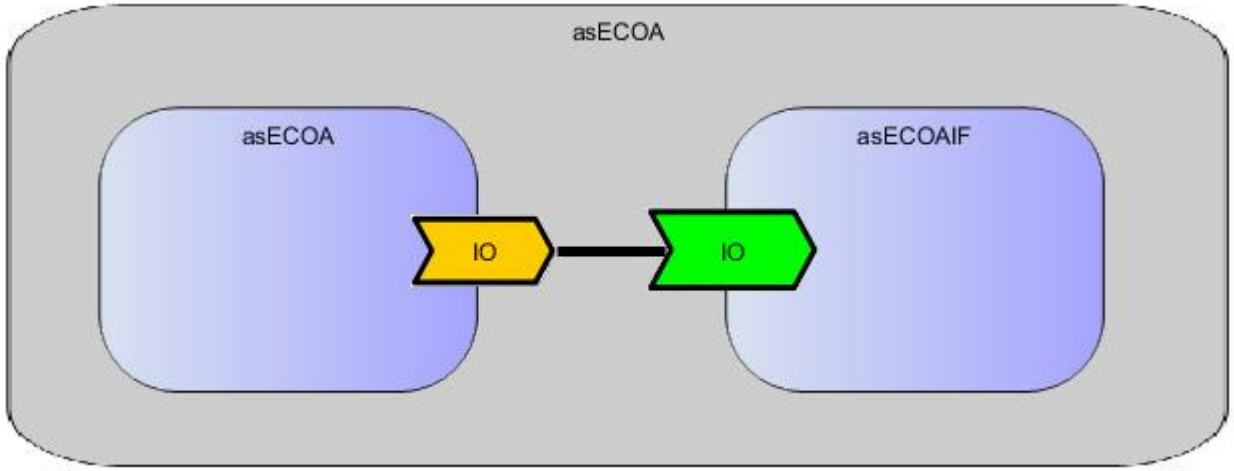
# The Example "Traditional" Demonstration Code

- Created and built in Ada:
  - "Traditional" approach coded per the design (slide 3)…

```
procedure Trad is
  :
begin
  loop
    -- Get the inputs
    TradIF.getPosn( acPosn );
    TradIF.getAlt( acAlt );
    TradIF.getSpeed( acSpeed );
    TradIF.getWayPoint( nextWP );
    --
    -- Do computations
    computeSteer( acPosn, acAlt, acSpeed, nextWP, steer );
    --
    -- Output results
    TradIF.putSteer( steer );
    --
    -- Wait until the epoch
    nextTime := nextTime + Ada.Real_Time.Milliseconds (50);
    delay until nextTime;
  end loop;
end Trad;
```

# The Example ECOA Design

- In the ECOA, the separation of “application” from “interface” functionality is preserved in the design, appearing explicitly in the componentization of the application…

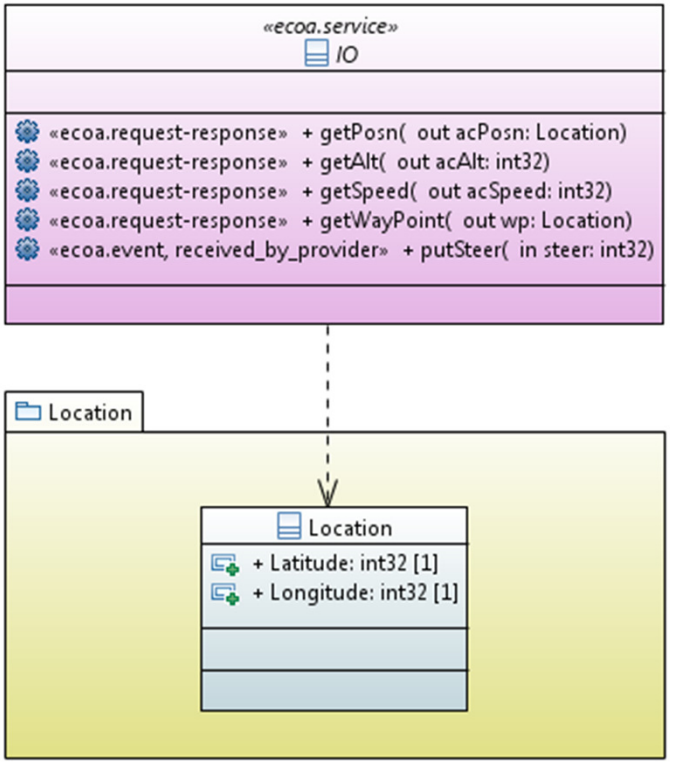


# The Example ECOA Design (2)

- A UML version of the Composite (Assembly) diagram

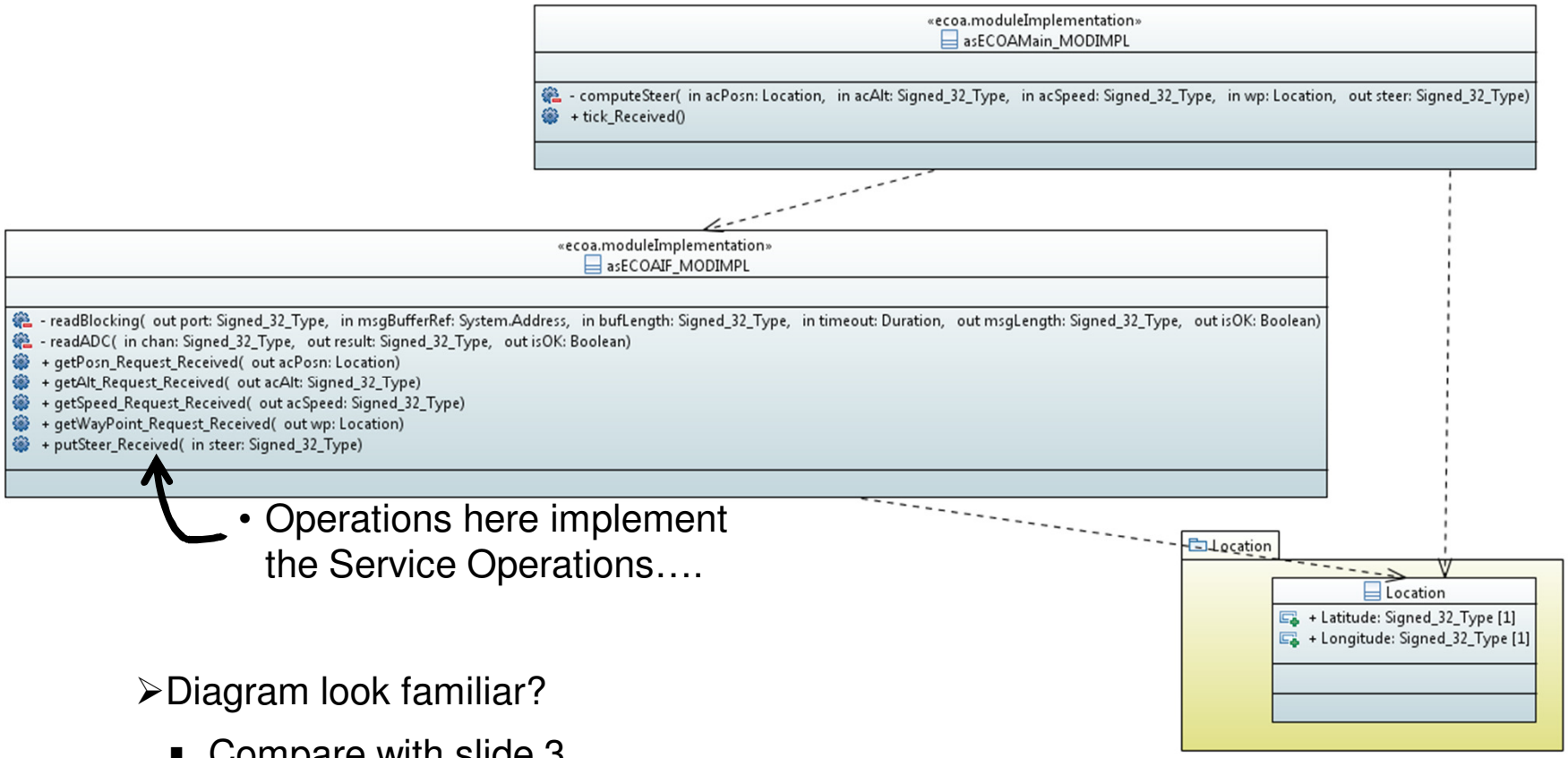

- The Service definition as ECOA XML and UML:

```
<serviceDefinition …>
  <use library="Location"/>
  <operations>
    <requestresponse name="getPosn">
      <output name="acPosn" type="Location:Location"/>
    </requestresponse>
    <requestresponse name="getAlt">
      <output name="acAlt" type="int32"/>
    </requestresponse>
    <requestresponse name="getSpeed">
     <output name="acSpeed" type="int32"/>
    </requestresponse>
    <requestresponse name="getWayPoint">
     <output name="wp" type="Location:Location"/>
    </requestresponse>
    <event name="putSteer" direction="RECEIVED_BY_PROVIDER">
      <input name="steer" type="int32"/>
    </event>
  </operations>
</serviceDefinition>
```



# The Example ECOA Design (3)

➢ECOA Module Design, represented in UML…



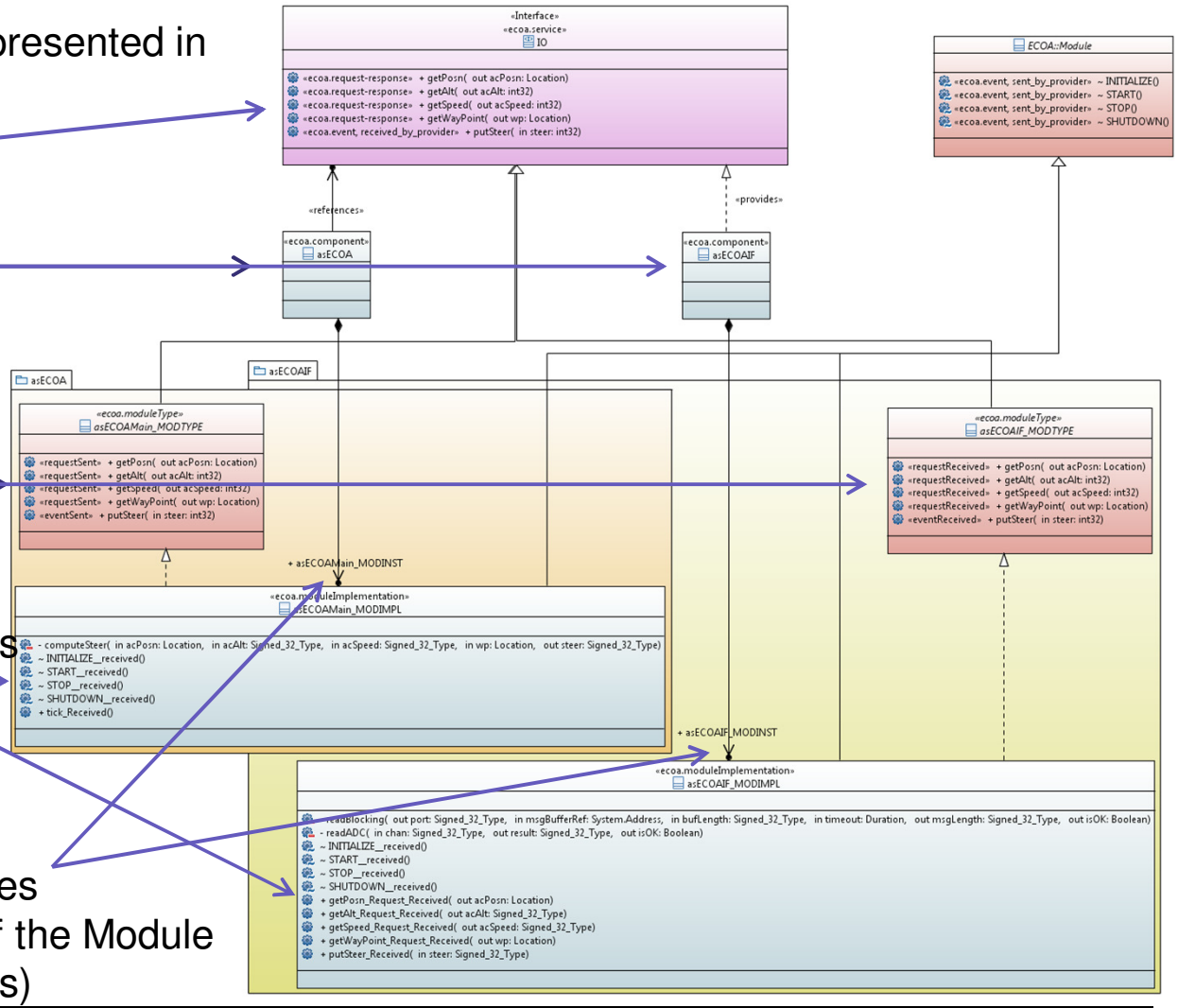

- Operations here implement the Service Operations….

➢Diagram look familiar?

- Compare with slide 3…

# The Example ECOA Design (4)

- ECOA Design, represented in UML…
  - Service
  - Components
  - Module Types
  - Module Implementations
  - Module Instances (as *instances* of the Module Implementations)



# The Example ECOA Design (4)

➢ECOA Module Design (one of), captured in ECOA XML…

```
<componentImplementation … componentDefinition="asECOAIF">
    <use library="ECOA" />
    <use library="Location" />
  <moduleType name="asECOAIF_MODTYPE" isSupervisionModule="true">
    <operations>
      <requestReceived name="getPosn">
        <output name="acPosn" type="Location:Location"/>
      </requestReceived>
      <requestReceived name="getAlt">
        <output name="acAlt" type="int32"/>
      </requestReceived>
      <requestReceived name="getSpeed">
        <output name="acSpeed" type="int32"/>
      </requestReceived>
      <requestReceived name="getWayPoint">
        <output name="wp" type="Location:Location"/>
      </requestReceived>
      <eventReceived name="putSteer">
        <input name="steer" type="int32"/>
      </eventReceived>
    </operations>
  </moduleType>
  <moduleImplementationname="asECOAIF_MODIMPL" moduleType="asECOAIF_MODTYPE" language="Ada"/>
    : etc.
</componentImplementation>
```

# The Example ECOA *Module* Code

- Created and built in Ada:
  - ECOA ASC and Container frameworks (stubs) created using an ECOA "API" Code Generator.

*Module Code*

```
procedure tick_Received
      (Context : in out asECOAMain_MODIMPL_Container.Context_Type)
   is
    :
begin
  -- Get the inputs
  getPosn_Request_Sync( Context, acPosn, e
  getAlt_Request_Sync( Context, acAlt, err
  getSpeed_Request_Sync( Context, acSpeed,
  getWayPoint_Request_Sync( Context, nextW
  --
  -- Do computations
  computeSteer( Context, acPosn, acAlt, ac
  --
  -- Output results
  putSteer_Send( Context, steer );
  --
  -- Wait until the epoch
  --
  -- In ECOA, that simply means wait until the next tick of the TriggerInstance...
  --
end tick_Received;
```

*ECOA TriggerInstance Implementation*

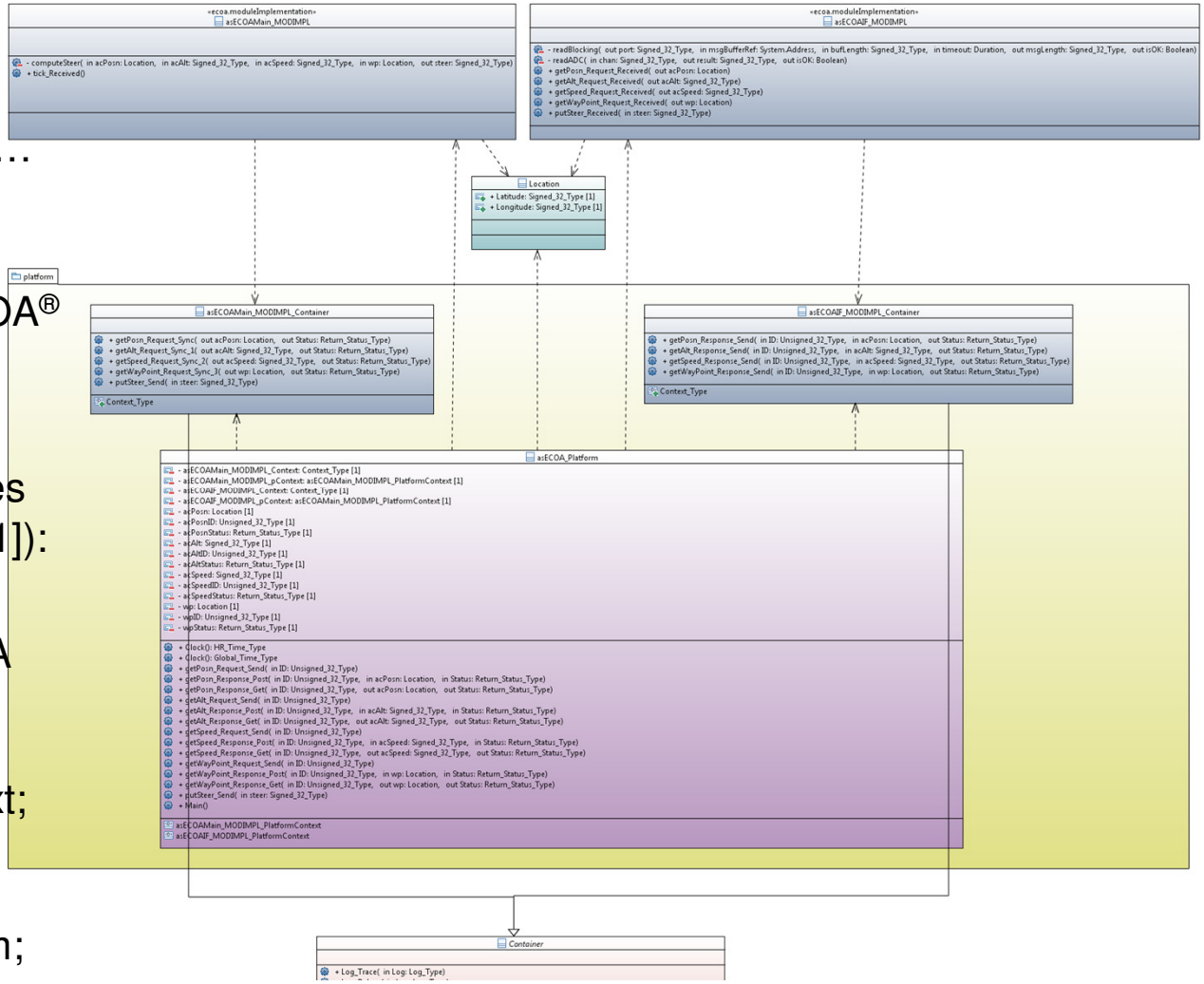
```
loop
  --
  if asECOAMain_MODIMPL_pContext.tickerState =
                                  ECOA.Module_States_Type.RUNNING then
    asECOAMain_MODIMPL.tick_Received( asECOAMain_MODIMPL_Context );
  end if;
  --
  -- Wait until the epoch
  nextTime := nextTime + Ada.Real_Time.Milliseconds (50);
  delay until nextTime;
end loop;
```

# The Example ECOA Design - The ECOA Platform

- In this example we'll CREATE the Container and ECOA Platform code (rather than Code Generate it):
  - It will be an Application Specific, bare minimum, ECOA Platform.
- The Container/Platform code will need to provide:
  - the ECOA *TriggerInstance* implementation (shown previously);
  - Container Interface (ref.[1]) implementations;
  - Inter-Component data transfer.
- The example will not be deployed across protection Domains or ECOA Computing Platforms so there will be no ELI implementation or Platform Management.
- The Container/Platform code MUST maintain the expected Component Behaviour according to the ECOA Architecture Spec. (ref.[1]).
- The ECOA Assembly will be single-threaded – ALL code will run on the same thread.

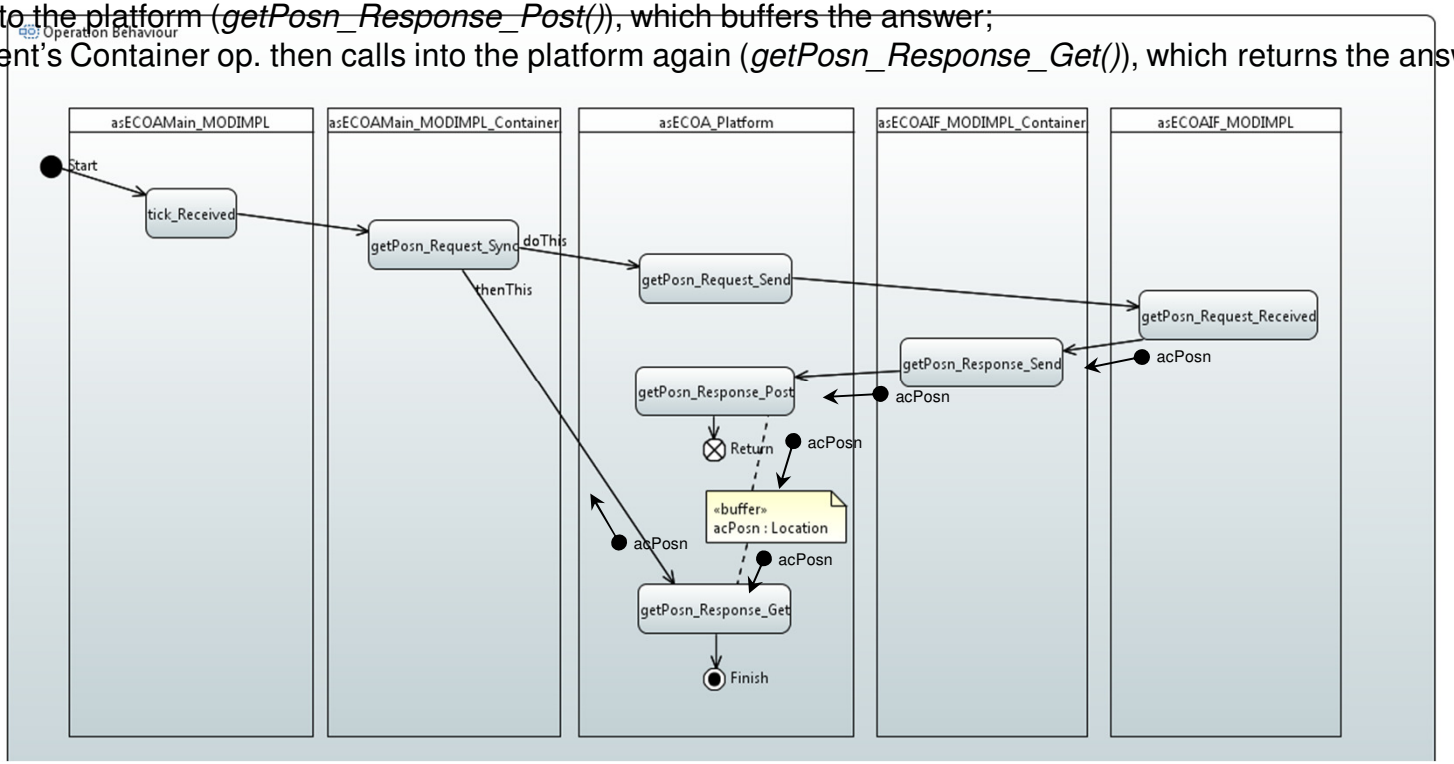
# The ECOA Example Design - The ECOA Platform (2)

- Simplified, Application Specific, ECOA Platform implementation…
  - Absolute minimum to do ***THIS*** job…
  - Is therefore **NOT** ECOA® compliant.
- BUT the ECOA Components and Modules ***are*** per Arch.Spec. (ref.[1]):
  - Framework code generated from ECOA XMLs;
  - Are fully reusable in another ECOA context;
  - Application functional changes have zero impact on the platform;



# The Example ECOA Design - The ECOA Platform (3)

- Single-threaded Service Operation behaviour for a synchronous Request-Response:
  - The client (function *tick_Received()* of Module Implementation *asECOAMain_MODIMPL*) calls it's Container op. (*getPosn_Request_Sync()*), which calls the platform op. (*getPosn_Request_Send()*), which calls the server op. (*getPosn_Request_Received()*), which calls ***it's*** Container op. (*getPosn_Response_Send()*) with the answer, which calls into the platform (*getPosn_Response_Post()*), which buffers the answer;
  - The client's Container op. then calls into the platform again (*getPosn_Response_Get()*), which returns the answer…




  - Why like this, with a buffer? Because the *EwECOAIF* Module Operations don't have a returned "response" parameter…

# The Example ECOA Container & Platform Code

*Snippets…*

```
getPosn_ID     : ECOA.Unsigned_32_Type := 0;

procedure getPosn_Request_Sync
      (Context : in out Context_Type;
       acPosn : out Location.Location;
       Status  :    out ECOA.Return_Status_Type)
is
       pContext : asECOA_Platform.asECOAMain_MODIMPL_PlatformContext;
       for pContext'address use Context.Platform_Hook;
       begin
       getPosn_ID := getPosn_ID + 1;
       asECOA_Platform.getPosn_Request_Send( getPosn_ID );
       asECOA_Platform.getPosn_Response_Get( getPosn_ID, acPosn, Status
end getPosn_Request_Sync;
```

- *getPosn_Request_Sync()*
  ECOA Container Function

```
procedure getPosn_Request_Send
    (ID : ECOA.Unsigned_32_Type ) is
begin
       asECOAIF_MODIMPL.getPosn_Request_Received(
       asECOA_Platform.asECOAIF_MODIMPL_Context, ID );
end getPosn_Request_Send;

procedure getPosn_Response_Post
    (ID : ECOA.Unsigned_32_Type;
       acPosn : in Location.Location;
       Status : in ECOA.Return_Status_Type ) is
begin
       asECOA_Platform.acPosn := acPosn;
       asECOA_Platform.acPosnStatus := Status;
       asECOA_Platform.acPosnID := ID;
end getPosn_Response_Post;

 procedure getPosn_Response_Get
    (ID : ECOA.Unsigned_32_Type;
       acPosn : out Location.Location;
       Status : out ECOA.Return_Status_Type ) is
begin
       if ID = asECOA_Platform.acPosnID then
              acPosn := asECOA_Platform.acPosn;
              Status := asECOA_Platform.acPosnStatus;
       else
              Status := ECOA.Return_Status_Type_INVALID_HANDLE;
       end if;
end getPosn_Response_Get;
```

- *getPosn_xxxx()*
  ECOA Platform Functions

# References


| | |
|---|---|
| **1** | European Component Oriented Architecture (ECOA®) Collaboration Programme: Architecture Specification (Parts 1 to 11) "ECOA" is a registered trade mark. |
| **2** | ECOA Software Description with UML Part of the *BAE Systems ES (UK)* ECOA Samples documentation set. |
| | |
| | |