

# ECOA - A New Architecture Concept for Complex Military Software Systems

Thierry Cornilleau  
Dassault Aviation

Pierre Linard  
DGA

Paul Moxon  
BAE Systems

Christopher Nicholas  
DSTL

## ABSTRACT

ECOA is an active software architecture research programme conducted by the French Republic and United Kingdom. It is one product of the recent Defence and Security Co-operation Treaty signed between the two nations. This paper provides an overview of the programme goals and progress as well as an introduction to the technology being developed and comparison to related initiatives.

The goal of the ECOA programme is to define an open software architecture that enables collaborative development of mission system software. The ECOA programme is needed to reduce development and lifecycle costs of future military air programmes. For this reason the programme has a specific focus on combat-air mission systems but the underlying technology is general purpose, applying to multiple military and civil domains.

At present, the programme has defined a concept, delivered a set of initial technical standards and produced a joint demonstrator to validate the technology developed.

**CITATION:** Cornilleau, T., Linard, P., Moxon, P., and Nicholas, C., "ECOA - A New Architecture Concept for Complex Military Software Systems," *SAE Int. J. Aerosp.* 7(2):2014, doi:10.4271/2014-01-2227.

---

## INTRODUCTION

The European Component Oriented Architecture (ECOA) is a collaborative research programme initiated by French and United Kingdom governments to define an open software architecture to support the development of next generation combat air mission systems. It builds on previous successful European software architecture programmes.

The programme is led by Dassault Aviation in France and BAE Systems Military Air and Information in the UK and involves the following UK and French industry partners: AgustaWestland, BAE Systems Electronic Systems, Bull SAS, General Dynamics UK, GE Aviation Systems, Thales Systèmes Aéroportés and SELEX ES. The Anglo-French ECOA programme is split into a number of stages. The first stage commenced in 2011 with work planned to continue until 2017.

Second section presents the goals of the programme. Third section provides an overview of the technology developed by the ECOA programme. Fourth section gives an update on the current status followed by a comparison to related initiatives in following section. Final sections present some of the planned future work followed by a summary and conclusions.

## GOALS

Previous avionics software architecture initiatives have focussed on the need to protect systems from obsolescence, support rapid product upgrade and support the needs of increasingly large and complex software-intensive systems. A large amount of success has been achieved in this area, evidenced by the deployment of modular and open architecture

principles on manned military air platforms. For example, the BAE Systems Hawk and Tornado [1], Dassault Aviation Rafale and Mirage 2000 [2].

Future development programmes are likely to require more efficient collaborative software development as cost pressures increase and systems continue to become larger. Work share is expected to be an increasingly important factor in future contracts. In addition there is a desire to support an expanded software supplier base, driving innovation and reducing cost. All this is set in an environment of increasingly complex and connected capability.

The ECOA programme was launched to meet these needs, therefore reducing development and through-life costs for new collaborative development programmes. It aims to do this by reducing risk in the integration of complex mission systems, promoting software reuse and improving competition in the software supplier base.

The ECOA programme seeks to exploit architectural concepts and systems engineering techniques that are already widespread in other industry sectors and used to manage the development of complex information systems. These include:

- Component-based software engineering;
- Loosely-coupled service-oriented architectures;
- Model-Driven Engineering (MDE) and model-driven-development (MDD);
- Publisher-subscriber data-oriented architectures.

In some cases, these approaches provide increased flexibility, but the ECOA programme also recognises the rigorous qualification and certification demands in the production avionics software environment.

One of the key constraints in introducing a new avionic technology is legacy, both in terms of application software and underlying HW/SW platform. The ECOA programme is designed to support legacy in two ways through the ability to:

- Deploy ECOA technology on a variety of hardware and software platforms;
- Integrate ECOA software with non-ECOA applications.

It is a goal to provide an unambiguous open technical standard which maximises both the *portability* of new application software and *interoperability* between different ECOA and non-ECOA software platforms

## TECHNICAL OVERVIEW

The technology used in the ECOA programme is based on Service Component Architecture (SCA) [6] and shares much of the terminology. Some aspects have been simplified and additional features have been integrated to cater for the specific requirements of avionic applications.

## Components and Services

At the heart of the architecture, software applications are comprised of portable *Application Software Components*. The functional and non-functional interface to the component is fully described and the intellectual property associated with a component implementation is protected. A component supplier is free to implement the component in any way he or she chooses. Components provide *Services* to other components and require services of other components. *Properties* allow component behaviour to be tailored and the *Insertion policy* describes aspects such as Quality-of-Service (QoS). This view of a component is shown in [Figure 1](#).

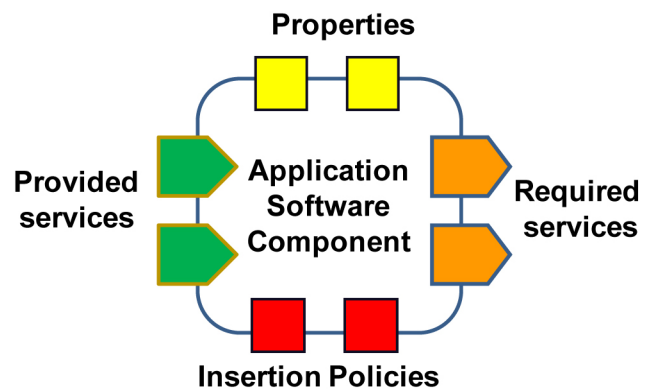


Figure 1. An Application Software Component

The system engineering process results in a system comprising a set of components, potentially from different suppliers, connected together by *service links*. A set of connected components is shown in [Figure 2](#). This logical view of the system is known as the *Assembly Schema*. One-to-one and many-to-one connections are possible.

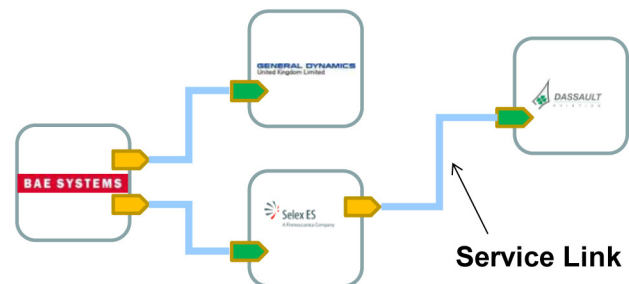


Figure 2. An Example Assembly Schema

## Communication

Services are comprised of *operations*. In order to ensure flexibility, operations can be of three different types as shown in [Figure 3](#):

- Request-Response;
- Events;
- Versioned Data.

The *Request-Response* operation type is analogous to Remote Procedure Call (RPC), can be synchronous or asynchronous, and is associated with a number of typed input and output parameters. When a component provides a request-response operation, it is acting as the server and the requiring component is the client.

The *Event* operation type can carry a number of typed parameters. Events can be sent in both directions between a providing component and requiring component. In other words a component can provide a source of events and act as a sink for events.

*Versioned data* is a mechanism that allows exchange of typed data between components. Providing components write data and requiring components are able to read the latest copy of the data. The versioned data concept allows the construction of a publish/subscribe paradigm.

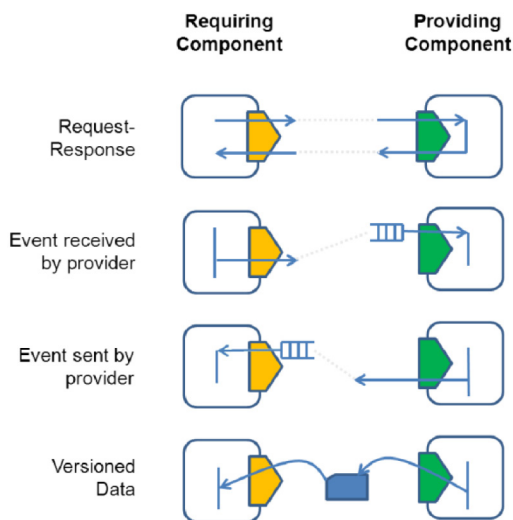


Figure 3. ECOA Communication Mechanisms

### Containers and Inversion of Control

A fundamental requirement for the ECOA programme is to ensure the component software is able to be mapped onto different underlying platforms. This is because different programmes and organisations have dissimilar requirements and wish to build on their existing investment. For example BAE Systems MAI has a mature Integrated Modular Systems (IMS) product as has Dassault Aviation, with its MDPU (Modular Data Processing Unit) software architecture.

In order to meet this requirement an application software component must be *deployed* within a *container*. A container maps the platform independent code onto the underlying hardware/software platform. All the communication is routed via the container which, for example, queues events.

The *Inversion-of-Control* principle refers to the fact that the container is responsible for the execution, scheduling and synchronisation of application software, rather than the application itself. Inversion-of-control minimises platform

dependencies and maximises portability. It is shown in Figure 4 where a thread (horizontal arrow) in the container dispatches operations within the component.

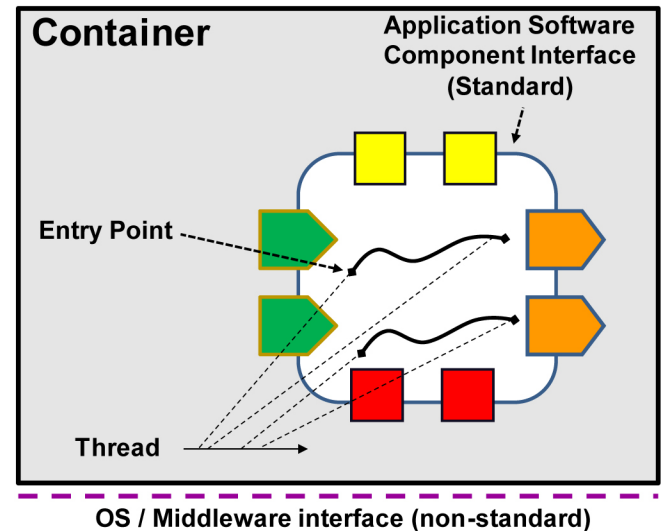


Figure 4. Containers and Inversion of Control

### Modules

Internally, components can be decomposed into *modules* that contain the executable code that implements the software component. Modules support safe development of multi-threaded components. This is because within a module all executable code is guaranteed to be executed non-concurrently; hence the developer does not have to ensure mutually exclusive access to shared data.

Modules communicate with the same operation types as described above (request-response, events and versioned data). This enables distribution of a component implementation to multiple processors, cores or partitions; ensuring flexibility.

Figure 5 shows an example component implemented as two modules. A request-response service operation ('requestOp') is provided (i.e. this component is a server). This diagram is designed to indicate that a call to requestOp results in the reading of a required versioned data service operation ('dataOp') and then a call to the internal module2 request-response operation ('internalReqOp'). The component also requires an event ('evOp') to be sent to it. This is linked to module2.

### Standardised Interfaces and Instantiated APIs

The standardised interface between component and the container is called the Application Software Component Interface and can be thought of as two interfaces:

1. The *module-to-container* interface which provides the module with operations to communicate with other components (e.g. send an event) and the underlying infrastructure