MODULAR AVIONICS FOR SEAMLESS RECONFIGURABLE UAS MISSIONS

Juan López, Pablo Royo, Cristina Barrado and Enric Pastor
Department of Computer Architecture, Technical University of Catalonia
08860 Castelldefels (Barcelona), Spain

Abstract

Integrated Modular Avionics (IMA) architecture is a trend in current avionics that employs a partitioned environment in which different avionics functions share a unique computing environment. UAS avionics, especially in small UAS, are usually of less complexity than not the present on airliners, however, in real autonomous UAS, the onboard avionics should control not only the flight and navigation but also the mission and payload of the aircraft. This involves more complex software as it should implement "intelligent" or at least autonomous behavior.

This need of both flexibility and complexity management while keeping low costs in the UAS avionics field requires new architectures to cope with.

In this article, we describe a modular avionics architecture based on services. The avionics functionality is divided in distributed elements, the services, which are interconnected by a communication middleware. This article also proposes a configuration and deployment infrastructure and its related procedures that complete our vision of UAS avionics.

Introduction

Unmanned Aerial Systems (UAS) are becoming a valid option for many civil missions. UAS are a low-cost alternative for some of the so called "D-cube" applications, i.e. situations identified as Dangerous, Dirty or Dull. Multiple manufacturers and platforms are appearing in the market; however most of them are currently focused in one specific mission. Real acceptation of UAS will only happen when the same aerial platform can be used for different missions and it can be easily adapted in case different payload (airborne sensors or actuators) are needed. In addition, to keep their costs low, UAS hardware and avionics should be shared and reused between different platforms.

Integrated Modular Avionics (IMA) architecture is a trend in current avionics that employs a partitioned environment in which different avionics functions share a unique computing environment. This sharing involves weight and power savings since resources can be used more efficiently. UAS avionics, especially in small UAS, are usually of less complexity than not the present on airliners… less instrumentation, less engines, no need to monitor and control the pressurization, etc. However, on the other hand, in real autonomous UAS, the onboard avionics should control the flight, navigation, mission and payload of the aircraft. This involves more complex software as it should implement "intelligent" or at least autonomous behavior.

This need of both flexibility and complexity management while keeping low costs in the UAS avionics field requires new architectures to cope with. We propose a modular architecture based on services. The avionic system is composed of set of distributed elements, known as services, which operate on top of a middleware communication framework. The services are collocated over the different computational nodes that are connected by a low-cost Ethernet network. This interconnection scheme is very flexible and cost-effective.

To be operative, this architecture definition and abstraction layer also need a definition of the operations and procedures to convert a set of "standardized" services into a flyable and operational system. This article proposes a configuration and deployment infrastructure and its related procedures that complete our vision of UAS avionics.

Avionics Architectures

Modern digital avionics are mainly implemented as distributed computing

978-1-4244-2208-1/08/$25.00 ©2008 IEEE.

architectures. Two different approaches are given: federated and modular.

Federated avionics architectures appeared on the early 80s. In this architecture, distribution is understood as self-contained, independent packaging of avionics functionalities. Federated avionics have a univocal relation between functionalities and resources: Every avionics functionality is integrated into a back-box and none resource is shared between avionics systems other than the communication buses. A typical example of federated avionics is a standalone Flight Control Systems like AP04[1] or Piccolo[2].

Since 2001, with the developments of Boeing 787 and later Airbus A380, the civil distributed avionics architectures are moving to the concept of Integrated Modular Avionics (IMA) [3]. In the IMA approach the avionics functionalities are distributed into logical Partitions which may be allocated into a same physical computing Module or into a different one. A computing Module is a hardware board with one or more micro-processors. All available Modules, connected through avionics buses, are highly integrated by a common software layer, typically the ARINC 653 APEX[4].

Airbus calls IMA Modules as Modular Avionics Units while Boeing names them Common Core Systems. In general IMA Modules are Line Replaceable Units (LRU) that follow the ARINC 600 physical standard. For their connectivity any avionics bus can be used (ARINC 429, AFDX, etc.).

The main differences between both architectures are the possibility of sharing resources between avionics systems and the avionics interfaces. While federated avionics do not share computing resources, IMA avionics share computing resources and also displays, other devices, and even busses. On the other side, avionics interfaces on federated avionics are limited to a number of hardware connectors, while in IMA the interfaces are mainly software definitions, and a large number of them may exist.

Icarus Service-Based Architecture

UAS avionics requires specially suited architectures to cope with its low cost and high flexibility requirements. We propose a modular architecture based on services. The avionic system is composed of a set of distributed elements, known as services, which operate on top of Marea, a middleware communication framework. The services are collocated over the different computational nodes that are connected by a low-cost Ethernet network. This interconnection scheme is very flexible and cost-effective.

Architecture

In contrast with typical IMA, the computational nodes are not necessarily homogeneous. Their design and capabilities will depend on the functions to implement (resources) and the mission objectives (hardware sensors and actuators). See Figure 1.

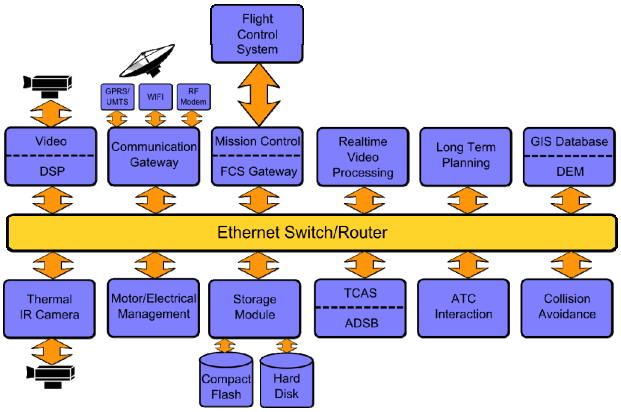

Figure 1. Architecture Based on Heterogeneous Nodes Interconnected by Ethernet Network

The UAS distributed architecture we propose is a mixed approach of both federated and integrated avionics architectures. We have considered the avionics systems into two categories, the critical avionics and the non critical avionics, understanding critical as to be certified. For each type of avionics system we decide to use the federated or the IMA approach. The Flight Control System (FCS) we use is a federated avionics system, a black-box with full contained functionality and the required certification level. The FCS has two redundant processors fully dedicated to control the flight. This system is directly connected with a dedicated ground station using also a dedicated radio channel for communication. All the benefits of a federated

avionics are given, in particular the clear responsibility about flight.

However, we propose a modular approach for the rest of the avionics. The mission management and the related decision systems are less critical avionics systems. They give the UAS autonomy and intelligence on the use of the payload and even interact with the FCS and modify the flight plan while the UAS operator allows it. In case of failure, the FCS and its independent radio link recovers the control and allows the operator to safely guide the UAS to base.

In our proposal the IMA modules are heterogeneous, and their integration is achieved with Marea [5] a middleware software layer that handles the communications of the avionics.

Services

Over this architecture of computational nodes we deploy the different services that will implement the UAS avionics functionalities. A hosted-function in the IMA sense can be composed by several services. Therefore, services have a finer granularity than a hosted-function; this allows sharing not only resources but also generated or computed information.

This finer granularity also possibilities redundancy at lower levels. For example in Figure 2, altitude information is needed in lot of different UAS avionics functionalities: navigation, terrain avoidance, photo normalization, etc. A lot of components also provide this information with different levels of precision and ranges: GPS, inertial measurement units, radio altimeter, etc.

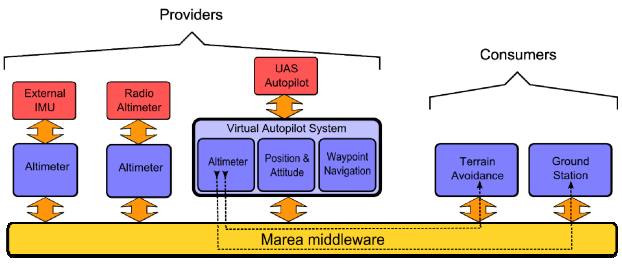

Figure 2. Managing Altitude Service Redundancy

The Icarus architecture allows that a service uses the altitude provided by several other services. The system automatically chooses the most accurate source of altitude. In case of failure less sensible providers of altitude are used transparently.

A service is a software application that behaves as a producer of data and as a consumer of data from other services present on the system. In contrast to ARINC APEX, we do not define partitions because the main focus of Marea is the ease reconfiguration. Services are designed to be outwardly descriptive so that they can be found via discovery mechanisms. In this case, when some service needs functionality that it is not provided by itself, it asks the system for the required service. If other available component of the system has this capability, its location will be provided and finally the client component will consume the service using the interface of the provider component. All services describe their interface by means of an XML file. Of course there is a physical partition given by the hardware nodes and in the next section we explain how we do the allocation of services to nodes using resource requirements annotations in the service description.

Middleware and Communication Primitives

The services management and especially their inter-communication is in charge of the Marea middleware [5]. Marea handles the redundancy and fall-back mechanisms and efficiently distributes sensors and services data. It uses the multicast capabilities of the Ethernet local network for minimizing the cost of concurrently issuing data to several services. Currently there is not priority management in the service scheduling of Marea, but in a near future we will introduce it using the APEX approach for soft real-time.

The communication primitives that provides to the services are capable to transparently locate and attach to the provider services with no need of knowing the final physical location of them. Four communication primitives (Variable, Event, Remote Invocation and File Transmission) give the avionics developer a wide field of possibilities to interconnect and to make interact services.

A *Variable* is a structured, and generally short, information offered by one service in a publish-subscribe way. This information may be sent at regular intervals or when changes occur. An *Event* is similar to a Variable but the middleware

.

guarantees the reliability of the transmission. Events are used to inform of occasional or important facts to other services. *Remote Invocation* is the classical way to model interactions between distributed components. It mimics a procedure call in non distributed environment. Finally a *File Transmission* is a data transfer of continuous information. This includes photography images, video, configuration files or even program code.

A Marea communication primitive identifies exchanged data rather than their providers or consumers. This functioning principle is similar to actual avionics buses such as ARINC 429. This bus broadcasts transmitted data, with an extra information (label), to all linked equipment and only the ones who have recognized the label use data. In that sense, Marea like ARINC 429 or APEX ports allows allow to link producers and receivers of data, without a priori knowledge of the physical location of them.

Comparing Marea and ARINC APEX communication primitives we observe that Marea does not differentiate between intra and inter partitions communications. Marea services can be distributed on any node, thus their communications are assumed to be always remote. It is a Marea implementation issue to avoid network transmission and use local inter process communication when possible.

ARINC APEX mailbox intra-partition communications like Buffers or Blackboards can be offered in Marea as independent services. I.e. we may create a service that subscribes to mailbox data, stores it using the Buffer or the Blackboard semantics and provides it on demand to any consumer service using Remote Invocation. On the other side, Variables and Events are similar to the inter-partition communication mechanisms offered by the ARINC APEX, in particular to the Sampling and Queuing mode Channels.

In our vision, a service is an independent producer and consumer of data, and the mesh formed by the set of all the services completes complete avionics functionality. In this sense, our architecture is data-centric and the data diffusion has to be very efficient.

UAS Service Abstraction Layer

To have an operative UAS executing on Marea we need to implement several services (see Figure 3). Since most of the missions will require a number of them, we have defined the UAS Service Abstraction Layer (USAL). This is a set of basic service definitions that can be reused between avionics systems. New services should implement some of the interfaces defined in the UAS to be able to interoperate with pre-existing services. USAL defines the shared information types, their meaning and behavior and some guidelines to make "equivalent" services interchangeable.

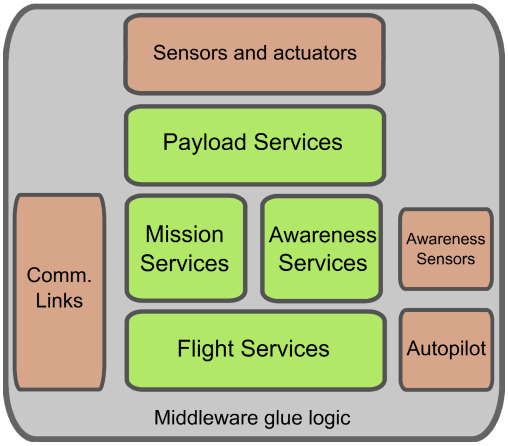

Figure 3. USAL Services

USAL defines a set of services comprising not only the flight and navigation functionalities of the UAS avionics, but also the mission part. The USAL services are grouped in the following categories: Flight, Mission, Payload and Awareness [6].

For instance, using the previous example of the altitude, the USAL will define the priorities for the different altitudes provided by the different components (GPS, inertial measurement units, radio altimeter, etc.) depending of their precision or ranges. The priority mechanism establishes a clear protocol in case of failure of one altitude provider.

In Figure 4 we can see an excerpt of the XML describing the Altimeter service. The <description> tag contains a textual definition of both the services and its primitives. Variables and events can declare some additional characteristics. In this case we can see that the altitude variable is expressed in meters, its value is between 0 and 15000 meters and has a refresh rate of 20Hz.

.

```
<service name="Altimeter">
  <description>
    This service provides altitude data.
  </description>
  <interface>
    <variable name="altitude">
      <description>
        This variable contains the current altitude
        in meters over the sea level.
      </description>
      <unit>m</unit>
      <range min="0" max="15000"/>
      <rate default="20Hz"/>
      <priority>1</priority>
    </variable>
  </interface>
```

Figure 4. Icarus Service Description XML

The underlying idea is to be able to implement a high number of UAS missions only reconfiguring the USAL services. The existence of the USAL, an open-architecture avionics package specifically designed for UAS, may alleviate the developments costs by reducing them to a simple parameterization.

Reconfiguration Process

The service oriented architecture and abstraction layer presented in the previous sections can be seen as a modular avionics architecture for UAS. To be operative, this architecture definition and abstraction layer also need a definition of the operations and procedures to convert a set of "standardized" services into a flyable and operational system. This section will detail the configuration and deployment infrastructure and its related procedures that complete our vision of UAS avionics.

Figure 5 shows the configuration and deployment process. This process begins with the definition of the mission to accomplish and finishes with the services required to achieve the mission objectives. All the services are assigned to and configured for the different computational nodes of the UAS airframe.

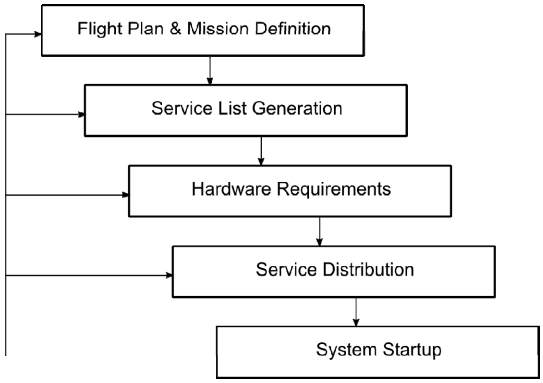

Figure 5. Icarus Reconfiguration Workflow

Flight Plan & Mission Definition

The first step of the configuration process is the definition of a flight plan and the mission objectives: The flight of the UAS is important as far as the visited way points are used for data acquisition or for actuators activation in order to obtain an autonomous UAS mission. In [7] the process of dispatching a UAS for a mission is presented.

For our purposes we obtain a Mission Description File where the flight plan is included with the necessary mission annotations. This file is the starting point for our reconfiguration process.

Service List Generation

Within our infrastructure, the process of configuring and deploying a new avionics system over a UAS starts from a mission description and a set of "standardized" services (the USAL). From this mission description the needed services and the aircraft and payload requirements are extracted (see Figure 6.

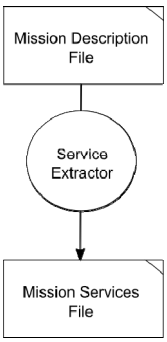

Figure 6. Service List Generation

.

Each service describes itself through the XML Service Description File. This file includes at least two sections with the external requirements of the service: the <dependencies> section and the <implementations> section. The <dependencies> section lists other services which provide information needed for this service. This way, during the service extraction process, the list of services is extended with all the dependent services. In example, an Autopilot service which depens on a FlightPlan service may have the following sections as shown in Figure 7.

```
<dependencies>
    <service name="FlightPlan"/>
    <variable name="position"/>
    <variable name="altitude"
        autoSubscribe="true"/>
    <event name="newWaypoint"/>
    <function name="fireParachute"/>
    <file name="aircraftPerformance"/>
    <variable name="gps.position"/>
</dependencies>
```

Figure 7. Service Dependencies XML

In the example of Figure 7, we see that, in addition to services, dependences identify also some USAL data like the Variable "position", or the Event "newWayPoint". These dependences do not specifically indicate which services offer them and it is responsibility of the Service Extractor to obtain them. Other clear examples of dependences are given for a Terrain Avoidance service which needs the current altitude given by a DEM service.

The <implementations> section is used for another type of dependence: the hardware. Imagine a Camara service which clearly depends on a camera device. Although the Camera service could be executed in any UAS node, only those with cameras connected will be available for a correct deployment.

In the section listing of Figure 8, we observe a service that requires two hardware components: a camera and a FPGA, but also it shows dependences on files and on computing resources.

```
<implementations>
  <implementation name="i386-dotnet">
    <files>
      <file name="fcsgw.dll"/>
    </files>
    <resources>
      <cpu name="i386" cycles="100"/>
      <ram size="1M"/>
      <powerConsumption
         max="2A" min="0.1A" mean="0.5A"/>
      <hw name="camera"/>
      <hw name="fpga"/>
    </resources>
  </implementation>
```

Figure 8. Service Resources XML

To conclude, this phase extracts the list of services and hardware requirements from the original services list and, by extension, from the given UAS mission definition.

Hardware Discovery and Analysis

Once the complete list of services are extracted we should check it and merge with the payload and resources actually installed on the selected airframe. We define the Aircraft Nodes Discovery & Analysis System (ANDAS) as an Icarus configuration service which connects to the aircraft internal network and extracts the hardware and payload available in the UAS (see Figure 9).

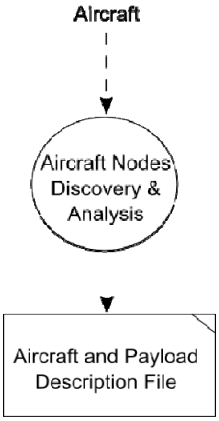

Figure 9. Nodes Discovery & Analysis

During the airframe startup each node runs an initial service called Node Manager that it is on charge of locating its own configuration. This includes all the computing resources: CPUs, memory, disks, etc. And it also includes the payload

.

devices connected to the node. The resulting information from all Node Managers is used to generate a global file with the Aircraft and Payload Description.

Service Distribution

The services have to be distributed over the different computational nodes of the airframe, checking that the required hardware for each service is present in the assigned node. During the distribution process the available and used resources in each node (CPU cycles, RAM, etc.) are computed and validated. Finally, a configuration is generated specifying the services that need to be assigned to each node. The objective is first to obtain a valid distribution, and second, to obtain an optimum configuration based on the load allocation.

Algorithms for optimum allocation on limited resources are a huge area of research that extends from economical problems (i.e. salesman problem) to map coloring. We do not pretend to propose a new allocation algorithm, but to apply existing ones like genetic algorithms or backtracking.

In general, the computational nodes of any airframe will usually be the same; however payload can be very different depending on the mission and the restrictions of the platform (size, weight, cost). The Service Distributor system can detect the differences between nodes and assign the different services to the correct node. For example in the case of a service implementing the access layer of a sensor (see Figure 10), the service will be obviously attached to the node connected to the sensor. This heterogeneity is a key difference with “classic” IMA architectures.

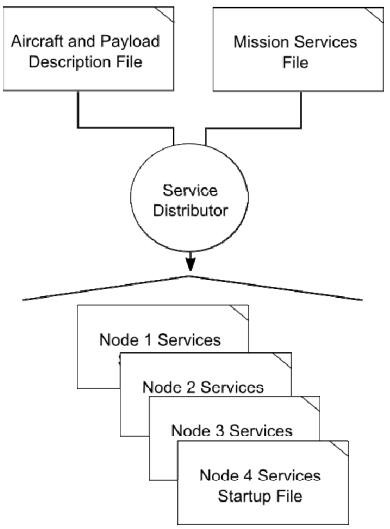

Figure 10. Service Distribution

Services Deployment and Startup

The Node Manager service presented above has still two more important function to offer: the services deployment and their initiation (see Figure 11). Using the Service Distribution output file, which includes all the services to be loaded to each node and all the additional data files needed, the Node Manager checks if the service code is allocated on the node and, if not, asks for it to an external Deploy and Startup Manager service. File distribution containing the service executable is efficiently provided by the underlying middleware using its multicast file transfer. If more than one node needs the same service file then the corresponding file can be sent simultaneously to multiple nodes. Then each Node Manager is responsible of starting all its services. Finally, the Deploy and Startup Manager is notified about the correct finalization and the UAS can proceed to pre-flight check and flight. In case of failure, the process is restarted back at some previous phase for allowing the maintenance team to solve the detected problems.

.

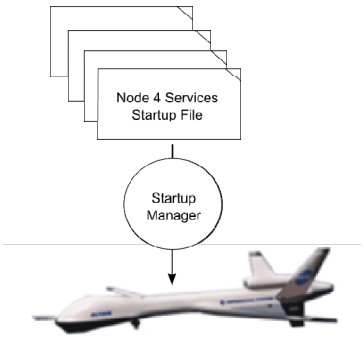

Figure 11. Service Deployment & Startup

In Figure 12 it is shown the final result of an standard UAS mission deployment. The UAS disposes of three different computational nodes. Over them, five services have been deployed: Virtual Autopilot System, Flight Plan Manager, Mission Control, Camera and Image Processing. The services have been distributed over the nodes without exhausting the nodes resources (CPU, RAM and Storage).

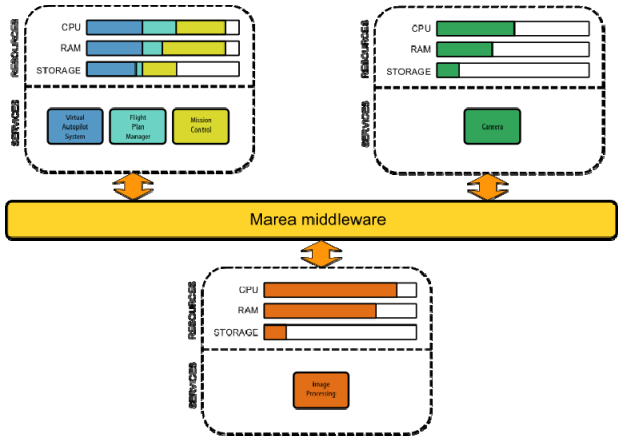

Figure 12. Final Services Assignation

These proposed configuration and service checking mechanisms allows that our avionics architecture can be adapted very quickly and easily to mission changes or completely new operations, as most of the operative can be automated and the subsystems reused among different missions.

Previous Work

Digital avionics, and specially integrated modular avionics, has been an active research topic the last years in both the industry and the academia.

GE Aviation has shared its large experience on IMA on different papers:

Watkins & Walter [8] give some advices for a successful transition from avionics federated architectures into IMA: 1) work hard in the Interface Control Document before any implementation in order to define clearly the system interfaces and 2) decide for an Open IMA system.

Littlefield & Viswanathan [9] go further in the proposal of an Open IMA system and present a notional architecture framework for IMA, the GOIMA, based on GOA (Generic Open Architecture [10]) to extend the opportunities of the existing ARINC 653 open standard. In the GOIMA three interfaces standardize the interaction between the 4 levels defined from Physical up to Applications: A Common Hardware Interface, a Common Platform Abstraction Layer and an Enhanced APEX interface that makes IMA applications independent format the actual Operating System. They claim that the extensive use of standards in other industries like automotive and telecom has promoted reusability (COTS components), reliability and decreased product development cycle times. In this paper the IMA hardware is assumed to be heterogeneous, and level 2 creates an abstraction layer for them (they call it PAL -Platform Abstractions Layer-) in the same way we have done with the autopilot and the VAS.

Garside & Pighetti [11] present the IMA integration challenges and contrast them with the previous avionics integration approach: Before IMA, integration was a hardware task mainly devoted to wire sensors and systems and it was done by the airplane manufacturer; now the responsibilities of the many avionics suppliers and the IMA platform provider have not always clear limits for the airplane manufacturer. The authors propose the creation of a new engineering role for IMA integration, as an independent third party that will extensively use simulation and testing tools.

IMA has moved avionics development from the hardware world into the software one. From this perspective [12] proposes a modeling formalism to design IMA components. Their approach is a top/down model where UML is first used to form the basic building blocks of the avionics metamodel and the bottom layer is dedicated to domain specific technologies. For the top layer they use the freely

.

available tool GME (Generic Modelling Environment [13]) which is entirely object-oriented and has an IMA library facility. The bottom layer uses the XML files generated previously to generate the application description using the SIGNAL data-flow language [14].

Conclusions

Seamless reconfigurable UAS are essential to make autonomous unmanned a reality for commercial missions. In the same way that in general avionics the publication of IMA standards represented a turning point for the flexibility of the avionics development, we have proposed Icarus, an IMA based architecture for UAS mission.

In the proposed architecture the avionics functionalities are called services and they are distributed over a network system with heterogeneous computational nodes. The IMA partition concept, which was mainly devoted to certification issues, is here used for an ease reconfiguration of the UAS. We do not address UAS avionics certification, which mainly includes to the autopilot on board and their security mechanisms. Our target is the mission related functionalities, which we consider "non-critical" tasks. Since small UAS introduce strict limits on power and weight of these avionics, we propose a clear process to achieve a full operative UAS rapidly.

This process, which starts with the definition of a UAS mission, consists on cascade phases and tools. These phases automatically extract the mission requirements (the list of services and hardware); verify them for the selected UAS; and finally distribute the services upon the aircraft resources. We have also included two additional operative phases: the services deployment and the services startup. Again they are part of the automation tools provided by the Marea middleware, which is the software layer of the distributed architecture proposed.

References

[1] UAV Navigation AP04 autopilot http://www.uavnavigation.com/uavprod/uavprod_01.htm

[2] B. Vaglienti, R. Hoag, M. Niculescu, Piccolo System User's Guide. Cloud Cap Technologies, http://cloudcaptech.com , 2005

[3] Cary R. Spitzer. Avionics Development and Implementation, Chapter 6. 2007. Digital Avionics Handbook, Second ed. CRC Press 20.

[4] ARINC Specification 653, Avionics Application Software Standard Interface, Published by ARINC, 2551 Riva Road, Annapolis, MD 21401. http://www.arinc.com

[5] J. López, P. Royo, E. Pastor, C. Barrado, E. Santamaria. A Middleware architecture for unmanned aircraft avionics, Proceedings of the 8th ACM/IFIP/USENIX international conference on Middleware, Newport Beach, California, 2007.

[6] P. Royo, J. Lopez, C. Barrado, E. Pastor. Service Abstraction Layer for UAV Flexible Application Development. 46th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NE, 2008.

[7] X. Prats, E. Pastor, P. Royo and J. Lopez. Flight Dispatching for Unmanned Aerial Vehicles. AIAA Guidance, Navigation and Control Conference and Exhibit, Honolulu, HI, 2008.

[8] C.B. Watkins, R. Walter. Oct 2007. Transitioning from federated avionics architecture to integrates modular avionics. 26th Digital Avionics Systems Conference (DASC).

[9] J. Littlefield, R. Viswanathan. Advancing open standards in Integrated Modular Avionics: An industry analysis. IEEE/AIAA 26th Digital Avionics Systems Conference, 2007. DASC '07. Dallas, TX, Oct. 2007

[10] Generic Open Architecture (GOA) Framework. http://www.sae.org. Document Number: AS4893.

[11] R. Garside, F.J. Pighetti. Integrating Modular Avionics: A new role emerges. IEEE/AIAA 26th Digital Avionics Systems Conference, DASC '07. Dallas, TX, Oct. 2007.

[12] A. Gamati, C. Brunette, R. Delamare, T. Gautier, J. Talpin. A Modeling Paradigm for Integrated Modular Avionics Design. 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, 2006.

.

[13] J. Davis. GME: the generic modeling environment. ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications archive. Anaheim, CA, 2003.

[14] P. LeGuernic, T. Gautier, M. Le Borgne, C. Le Maire. Programming real-time applications with SIGNAL. Proceedings of the IEEE, Vol 79, N. 9, Set 1991.

Acknowledgments

This work has been partially funded by Ministry of Education of Spain under grant number TIN2007-63927.