Noname manuscript No.
(will be inserted by the editor)

Automatic deployment of an RPAS Mission Manager to an ARINC-653 compliant system

Hèctor Usach · Juan A. Vila · Alfons Crespo · Pedro Yuste

Received: date / Accepted: date

Abstract The development process of avionics system requiring a high level of safety is subjected to rigorous development and verification standards. In order to accelerate and facilitate this process, we present a testbed that uses a suite of methods and tools to comply with aerospace standards for certification. To illustrate the proposed methodology, we designed a Mission Management System for Remotely Piloted Aircraft Systems (RPAS) that was deployed on a particular run-time execution platform called XtratuM, an ARINC-653 compliant system developed in our research group. The paper discusses the system requirements, the software architecture, the key issues for porting designs to XtratuM, and how to automatize this process. Results show that the proposed testbed is a good platform for designing and qualifying avionics applications.

Keywords Software design methodologies · Software architectures · Integrated Modular Avionics · Mission Managers · RPAS

1 Introduction

Avionics applications requiring a high level of safety and security must satisfy rigorous development and verification standards. The process for developing applications meeting these requirements can be time-consuming and expensive. For

This research has been financed by the Institute of Control Systems and Industrial Computing (Ai2), and by projects GVA AICO/2015/126 (Ayudas para Grupos de Investigación Consolidables) and GVA ACIF/2016/197 (Ayudas para la contratación de personal investigador en formación de carácter predoctoral) of the Spanish Regional Government "Generalitat Valenciana".

H. Usach · J. Vila · A. Crespo
Instituto de Automática e Informática Industrial.
Universitat Politècnica de València, Camí de Vera s/n, Valencia, Spain.
E-mail: hecusmo@doctor.upv.es, jvila@disca.upv.es, acrespo@disca.upv.es

P. Yuste
Inst. de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas.
Universitat Politècnica de València, Camí de Vera s/n, Valencia, Spain.
E-mail: pyuste@disca.upv.es

this reason, it is essential to carefully analyze requirements, standards, design methodologies, and tools from the early steps of a software project.

Software design methodologies for critical software used in avionics must follow the guidelines and activities defined by the DO-178 standard [12]. This document establishes the Design Assurance Level (DAL) of a given software project by examining the effects of a failure condition in the system. This level ranges from A (catastrophic) to E (no effect). Each level has a number of objectives that must be met in the software development, configuration, and verification processes.

Although the DO-178B/C model does not prescribe a particular design methodology, it clearly specifies some development and verification activities that must be addressed to meet the certification objectives. One of the most important issues is using a design model that allows to properly capture and specify the system requirements, to design it, and to verify it. Some of the most advocated methods for meeting the DO-178B/C objectives are the V-model and the Model-Based Design (MBD).

One of the key issues of any software design methodology is the *deployment phase*. It deals with the run-time environment and the deployment process for porting the software prototype to this run-time environment. The run-time environment for the avionics of the last generation of aircrafts (A350, A380, B777, B767) is based on the Integrated Modular Avionics (IMA) concept [1], which structures the system as a network of partitions. The partitioning concept provides protection and separation among applications running on the same hardware.

In order to accelerate and facilitate the design of avionics applications running in partitioned architectures, this paper presents a testbed where a suite of methods and tools have been used to comply with aerospace standards for certification. The process is illustrated through the design of a Mission Management System for Remotely Piloted Aircraft Systems (RPAS) flying in integrated airspace, and its deployment on a particular real-time execution environment called XtratuM [8], based on the IMA concept and developed in our research group.

The proposed Mission Manager software architecture is inspired on the ideas of the 3T architecture [2]. It structures Intelligent Reactive Agents into three main layers: the *deliberative* layer synthesizes the goals into a list of tasks; the *sequencer* layer decomposes the tasks into a set of actions; and the *reactive* layer executes the actions at the pace of the events that the system monitors. The ideas of this architecture have been redefined for the case of our Mission Manager System.

The deployment process deals with the specific details that must be taken into account to map and to customize the software design to the specific features of XtratuM. The main advantage of the proposed methodology is that this process can be automatized through several steps, and the resulting source code complies with the ARINC-653 specification for IMA architectures.

The rest of the paper is organized as follows: Section 2 motivates the use of Integrated Modular Avionics architectures and the proposed design methodology; Section 3 introduces the Mission Management System as an application partitioning example; Section 4 describes XtratuM and the development platform that will be used in this approach; Section 5 presents the software design of the system in study; Section 6 explains the process of porting designs to the proposed development platform; Section 7 shows the results of the deployment process of the Mission Management System on XtratuM; Section 8 discusses these results; and Section 9 concludes the paper.

2 Problem statement

The software integrated in airborne systems like an RPAS flying in integrated airspace shall demonstrate a level of confidence in safety that complies with the requirements of DO-178B/C. This document is the reference manual for the avionics industry and is also accepted as the interface with the Certifying Authority. It defines an explicit correlation between the severity of system hazards and the scrutiny to which that system is subjected [11,14]. However, according to that document, different architectural configurations could justify downgrading the DAL of a system, thus reducing the effort of the software verification process.

One of the architectural choices that can limit the impact of failures is *partitioning*. Partitioned architectures, called IMA architectures in aerospace, provide protection and separation among applications from the spatial and temporal point of views. This means that a fault in some partition does not affect the execution of other partitions running on the same processor nor the processor time allocated to each partition.

Another architectural choice that can help to reduce the DAL of a software component is *redundancy*. Redundant configurations mitigate hazards by executing replicas of a given application in different processors. However, having a dedicated hardware for each replicated application increases the system complexity. In contrast, IMA architectures can ease redundancy using a reduced number of processors: since software faults do not necessarily require to allocate an application on different processors, it is possible to design a partitioned architecture where each partition executes a replicated application. Afterwards, hardware faults can be addressed repeating this partitioning scheme in different processors.

The support for IMA architectures is defined by ARINC-650 and ARINC-651 documents that specify general purpose hardware and software standards, and especially by ARINC-653 (Avionics Application Standard Software Interface) which specifies the APEX (APplication/EXecutive).

Within this approach, this support is provided by XtratuM, an hypervisor for safety-critical systems that will be further discussed in Sec. 4. XtratuM offers all the advantages stated above, including support for different Real-Time Operating Systems (RTOS), see Fig. 1. One of them is LithOS, which is ARINC-653 compliant.

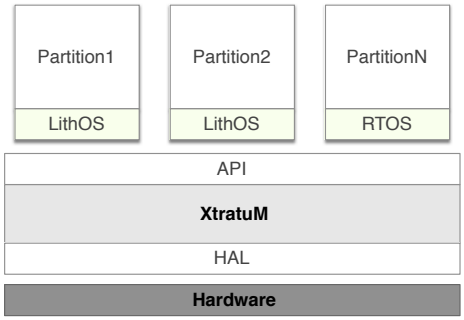


Fig. 1 Integrated Modular Avionics architectures and the XtratuM hypervisor

Finally, this paper advocates for the use of the MBD methodology for meeting the objectives of DO-178B/C. Basically, the development process of MBD consists on the following steps: *a*) System modeling, *b*) Model design and implementation, *c*) Model validation, and *d*) Deployment. The advantage is that these activities can be easily mapped to the DO-178B/C workflow.

The rest of the paper is focused on the experience of designing, porting and testing the prototype of an RPAS Mission Management System to an IMA architecture. This is basically a problem of making a MBD design of the proposed system, mapping MBD abstractions to IMA abstractions (i.e. partitions, ports, etc.), generating the executable code from the design model, and porting it to the IMA run-time system.

3 Application partitioning example

In this paper the previous scenario will be illustrated with the following example: the on-board Mission Management System of an RPAS flying in integrated airspace. We consider this system to be representative of the problem in study as it is composed of several modules with different impact on the system safety interacting between each other. It roughly consists on the following components:

- *Navigation System*: it receives inputs from the aircraft sensors to estimate the aircraft state, including position, speed, attitude, etc.
- *Mission Manager System*: it is responsible for handling and flying the route defined in a Mission Plan in an automatic manner. From a functional point of view, it receives navigation data and Mission Plan information to compute the reference state of the aircraft, i.e. the target value for the controlled variables.
- *Flight Control System*: it executes the control loops required to follow the intended path, producing the deflections of the control surfaces (i.e. elevators, ailerons, and rudder), and the throttle position.
- *Flight Data Recorder*: it stores flight data from the previous components of the system.

A functional schema is presented in Fig. 2. It is equivalent to the Flight Management System in manned aviation but the proposed system provides a higher level of automation. An increased level of automation is required to respond to the lower *situational awareness* of the Remote Pilot and to the unreliable communication links between the Remote Pilot Station and the vehicle. The key component is the Mission Manager System, that will be further discussed in Sec. 5.

The problem that rises from the software design point of view is how to allocate the previous functions into a partitioned architecture. Although this is a design issue, some considerations are to be noted. To start with, each of these components have a different impact on the safety of the operation (i.e. a different Design Assurance Level according to [11]). In this way, a first proposal using a serial implementation with a single partition implies that a failure at a non-critical system such as the Flight Data Recorder would cause the complete loss of the system functionality, what is not acceptable. The opposite is allocating each function at a different partition, but this increases the system complexity. Intermediate solutions shall take into consideration the following requirements:

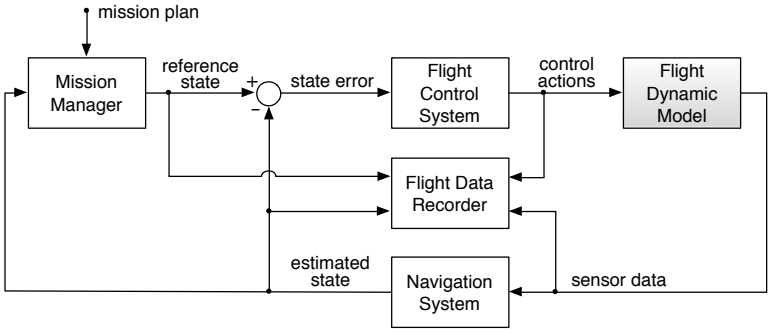


Fig. 2 Functional diagram of the RPAS Mission Management System

a) Safety impact: the failure of a component should not cause the loss of another one with a higher assurance level. As in the first example, the loss of the Flight Data Recorder should not affect the remaining components.
b) Inter-dependability: different components should be allocated at the same partition when they have common failure modes. For example, a failure occurring at the Navigation System is automatically propagated to the Flight Control System (thus they are not independent).
c) Execution environment: a function requiring specific services should be executed at a partition running the appropriate operating-system. For example, critical control systems require a real-time operating-system, while multi-tasking applications require thread/process services, etc.

From the above, application in Fig. 2 is configured using the partitioning schema presented in Fig. 3. It contains 3 partitions: navigation and flight control functions are allocated at the same partition as one depend on the other, while the Mission Manager and the Flight Data Recorder are executed at individual partitions. The reason for separating the Mission Manager and the Flight Control System is that, although the Mission Manager relies on the Flight Control System, the latter can work with independence of the Mission Manager, for example receiving Remote Pilot commands (see Sec. 5.1 for the operational modes of the proposed system).

Another remark is that safety-critical systems such as the Flight Control System often require redundancy (e.g., see RNP AR APCH navigation specification [5]). However, redundant configurations are out of the scope of this paper.

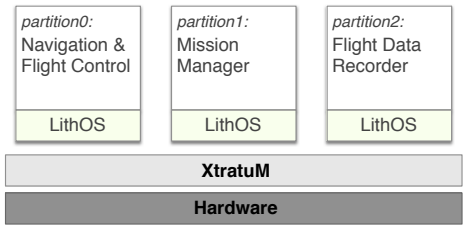


Fig. 3 Application partitioning example: RPAS Mission Management System allocating functions in three partitions

4 XtratuM and the development platform

The proposed execution environment for an application like the one presented above is based on XtratuM, a hypervisor for real-time embedded systems developed in our research group [8]. It is based on the Integrated Modular Avionics (IMA) concept standardized by ARINC-653, in which each partition integrates an application and an associated guest operating system (OS) (see Fig. 1). Temporal isolation is achieved implementing a fixed cyclic scheduler, what is consistent with the ARINC-653 scheduling policy for partitions; while spatial isolation relies on fixed memory allocation.

XtratuM supports several real-time operating systems; one of them is LithOS, an ARINC-653 compliant execution environment also developed at our research group [9]. It basically provides the services defined by the ARINC-653 specification, i.e. partition management, process management, time management, inter-partition and intra-partition communication, and health monitoring. Regarding process management, LithOS implements the priority based scheduling policy for concurrent applications.

In order to accelerate the design of aerospace applications to be executed on top of XtratuM, we have developed a simulation environment that allows to validate the functional behavior of a partitioned application from the early stages of the design process. The proposed architecture consists on an emulator version of XtratuM (Sec. 4.1) and the X-Plane flight simulator (Sec. 4.2). Configuration issues for this platform are presented in Sec. 4.3.

4.1 The XtratuM hypervisor emulator

The emulation environment of XtratuM presented in [3] can be used to prototype partitioned applications when a board running XtratuM natively is not yet available. It allows to debug and validate the functional behavior of a software application running on top of a Linux system.

The XtratuM emulator, also called Separation Kernel Emulator (SKE), runs as a Linux process that controls the execution of a set of processes that configure the partitioned system. In the emulation environment LithOS partitions are executed as Linux processes, where LithOS is included as an internal OS of each of these processes.

The emulation is functionally equivalent in all aspects except time management. The SKE process implements its own clock which provides emulated time, not real-time, and thus cannot be validated. But on the other hand, a LithOS partition executed as a Linux process can benefit from services provided by Linux, such as sockets or other libraries that can be integrated into the testbench.

4.2 The flight simulator

The flight simulator is a key component since it runs the flight dynamic model of the aircraft. We use as a demonstrator the Super Heron HF, a fixed-wing RPAS for reconnaissance operations (see Fig. 4(a)). One of the main advantages of X-Plane is that it provides full access to the simulator's property tree using UDP

(a) Super Heron HF model

(b) Remote Pilot Station interface

Fig. 4 Simulation environment in X-Plane

communication. This enables reading and writing flight simulation data, such as the aircraft state, the control actions, etc. [13]. Furthermore, X-Plane provides the following functions:

- *World modeling:* Environment information, such as terrain data, weather, other traffics, or navaids are simulated in X-Plane and can be accessed through the simulator's property tree.
- *Flight control functions:* Most of the aircraft models in X-Plane include their own autopilot system. This allows to delegate control tuning tasks to the developers of the aircraft model, what helps focusing on other design issues.
- *Remote Pilot Station interface:* Pilot commands such as the required operational mode can be introduced through the flight simulator interface, as shown in Fig. 4(b).

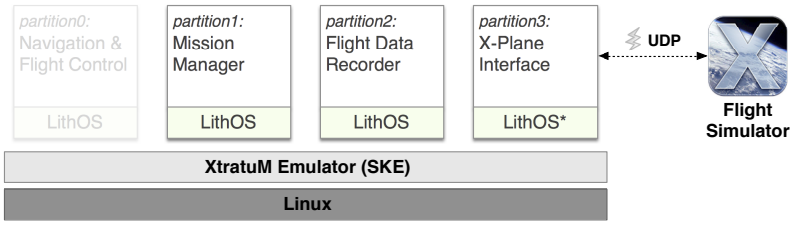


Fig. 5 Partitioning schema of the RPAS Mission Management System running in the development platfrom. Note that *partition3* runs LithOS but uses Linux services for UDP communication

4.3 Configuring the development platform

As it was introduced in Sec. 4.2, having access to the simulator's property tree from an external application requires UDP communication. In order to do so from an application running on top of the XtratuM's SKE, it is necessary to design a dedicated partition that reads and writes simulation data. This partition does not respond to a need of providing fault isolation but to provide the required execution environment: it exploits SKE features for using Linux services (in this case, UDP sockets, see Sec. 4.1) that are not available on LithOS.

As a result, executing an application like the one in Fig. 3 on the development platform here presented requires adding a supporting system partition that is in charge of running the X-Plane interface, see Fig. 5.

Another side-effect concerns timing requirements. Tests indicate that the frequency rate of UPD packets sent by X-Plane becomes unstable with frequencies above 20 Hz. Consequently, the major frame (MAF) of control applications where the control period is a very sensitive parameter shall be limited to that value.

Finally, and putting the focus back on the application described in Sec. 3, as our research group is putting all the development effort on the Mission Manager component, we have decided to rely on the Flight Control System provided by X-Plane to validate the design. Although in previous works we presented an autopilot testbed [17], this component strongly depends on the aircraft model; as we want the Mission Manager to be independent on the type of vehicle, using the autopilot provided by X-Plane simplifies the task.

The same occurs with the Navigation System. In [16] we designed a Navigation System for Performance-Based Navigation (PBN) applications. However, for simplicity we can read the actual state of the aircraft from the simulation environment. In conclusion, besides both Navigation and Flight Control functions can be implemented in the application code, we delegate these tasks to the flight simulator engine. This is why *partition0* in Fig. 5 is shown faded to white.

5 Mission Management System design

This section describes in more detail the design and implementation of the Mission Management System introduced in Sec. 3. The proposed system executes automatic guidance and control functions for a generic RPAS like the one in Fig. 4(a).

Its most distinctive requirement is flying in integrated airspace. Although the process of insertion of RPAS in integrated airspace is still under discussion [7], it is agreed that RPAS must fit into the Air Traffic Management (ATM) system transparently. This implies that RPAS need to: *1)* demonstrate an Equivalent Level of Safety (ELOS) to that of a human piloted aircraft, *2)* operate in compliance with existing aviation regulations, and *3)* appear transparent to other airspace users.

All the above is hard to fulfill specially under some contingencies, like the loss of the Command & Control (C2) link, when the RPAS is flying in a completely autonomous way without the possibility of pilot intervention. To address this issue, the proposed system implements the following features:

- Extended operational modes that provide a higher degree of automation (see Sec. 5.1).
- Automatic Contingency Management to keep safety levels without human intervention (see Sec. 5.1).
- Ability to fly standard (RNAV/RNP) flight procedures, allowing the operation in controlled areas (see Sec. 5.2).
- Ability to fly extended (RPAS-specific) flight procedures, providing more flexibility for defining the route in non-controlled airspace (see Sec. 5.2).

With respect to the software design –that will be further discussed in Sec. 5.3, the application is to be executed into the development platform of Sec. 4 using a partitioning scheme like the one in Fig. 5. This imposes fulfilling a scheduling plan with a major frame up to 20 Hz, as it was stated in previous sections.

5.1 Operational modes

Operational modes range from completely manual to completely automatic. The level of automation of an RPAS Mission Manager and the human factors that come into play when decision acts must be taken by a human operator interacting with a highly-automated system are one of the most interesting fields that are still open and require further research [10]. The proposed Mission Management System implements the operational modes recommended by ICAO in [6]:

- *Direct control*: the aircraft is controlled by the remote pilot allowing inputs from a control stick.
- *Autopilot control*: it provides tactical commands of a typical autopilot system, setting the following parameters: heading, altitude, speed and vertical speed.
- *Mission control*: it is the strategic operation and extends ICAO's *Waypoint control*, flying the route defined in a Mission Plan (see Sec. 5.2) in an automatic way.

However, RPAS have introduced different and greater human factor challenges. They arise primarily from the fact that operator and aircraft are not co-located, and can be summarized as the loss of situational awareness, and C2 link communication problems. This is why a fourth operational mode has been introduced:

- *Contingency control*: it allows to execute automatic contingency procedures triggered by an event caused by a safety-critical failure.

Contingency procedures have to be approved and certified by aviation authorities. In addition, the required contingency procedure may be different depending on some factors, specially the segment of flight in which the failure occurs.

5.2 Mission Plans

A Mission Plan is the specification of the RPAS route and operations, including the payload. In our proposal this specification also includes alternative plans to deal with different contingencies. The Mission Plan has to be coherent with the ICAO Flight Plan required by Air Traffic Control (ATC) when flying in controlled areas. However, not all the phases of an RPAS mission take place entirely in controlled airspace: some flight phases like take-off, en-route or landing usually go through controlled areas, but the flight procedures of the phase where the RPAS is in the working area performing payload specific operations, as a surveillance, inspection, search, are not under ATC. The proposed solution to deal with this scenario is based on specifying standard (RNAV/RNP) flight procedures using the *path terminator* (PT) concept defined by the ARINC-424 standard (such as *Initial fix*, *Track to fix*, or *Course to an altitude*) and using *extended path terminators* (EPTs) to define as many new flight procedures as needed in non-controlled areas in a flexible way. These include *Dubins paths* or *scan* circuits. They also include RPAS specific procedures as *autoland* to perform an automatic final approach based on GPS. The whole set of flight procedures (including PT and EPT) currently supported in the proposed Mission Manager can be found in [18].

5.3 Software architecture

This section describes the software architecture for the proposed Mission Management System. As it was discussed in Sec. 4.3, the only components to be implemented in the application code from those of the functional diagram in Fig. 2 are the Mission Manager System and the Flight Data Recorder; the remaining modules (Navigation System and Flight Control System) will be executed within the flight simulator. Furthermore, from a software design point of view the Flight Data Recorder is not significant as it only stores flight data for flight analysis purposes, so it will be no longer discussed.

The Mission Manager is thus the most relevant component. It performs mission planning and guidance functions, and supports the *Mission control* and *Contingency control* modes presented in Sec. 5.1. Regarding its interface, it receives the Mission Plan from the Remote Pilot Station, and navigation data and contingency events from the Navigation System; on the other hand, it outputs the required control modes and the target value for the control loops that are sent to the Flight Control System interface (see Fig. 6).

The previous figure also shows the Mission Manager software architecture. It is based on a layered architecture, called 3T, in which each layer provides a different level of abstraction on the guidance process. Basically, its goal is to split the Mission Plan into elementary flight legs that can be flown using traditional control modes of an autopilot system, i.e. heading/track hold (HDG/TK), altitude hold (ALT), vertical speed hold (VS), speed hold (SPD), etc. The three layers, named *Mission*, *Sequencing*, and *Guidance*, are briefly described in the following sections; further details can be found in [18].

The main advantage of this architecture is modularity and flexibility: having three interacting layers with a different level of abstraction allows the designer to introduce modifications at a reduced time and effort. For example, supporting

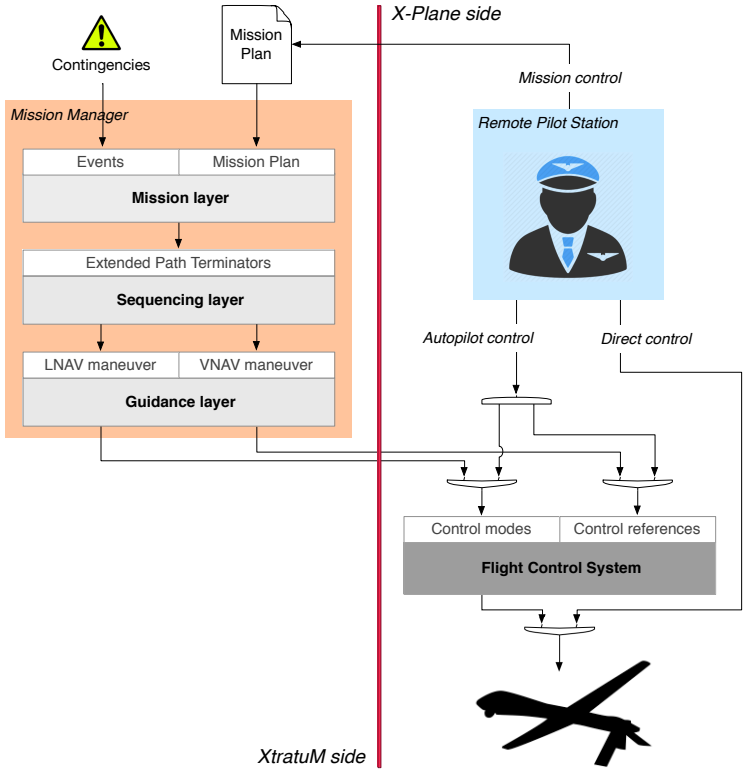


Fig. 6 Mission Manager interface and its software architecture. For simplicity, Navigation System and Flight Data Recorder are omitted in this figure

Mission Plans that are defined with another specification simply requires adapting the Mission layer interface. In the same way, the proposed Mission Manager can be mounted with a different Flight Control System with minor modifications on the bottom layer. Finally, it is possible to enhance the system functionality by adding new flight procedures at the Sequencing layer, having the other levels unchanged.

5.3.1 Mission layer

The Mission layer is at the top of the architecture and thus processes the Mission Manager inputs. It is responsible for interpreting and sequencing the Mission Plan, selecting the next Extended Path Terminator (EPT) to be executed and sending it to the Sequencing layer. It also monitors the occurrence of contingency events (safety conditions) that could jeopardize the integrity of the operation, and selects the required operational mode from those presented in Sec. 5.1.

This layer is implemented as a state automaton where transitions are triggered by events. Event handling is a preemptive action: a contingency event preempts the execution of the current EPT. Up to now, we restrict our attention to the following alerts: *a*) C2 link loss, *b*) GPS loss, *c*) Traffic advisory (loss of separation minima), and *d*) autopilot disengagement. A specially critical situation is the prevention of

the *autopilot disengagement* once *C2 link loss* has occurred, what would imply the flight termination, but this aspect goes beyond the scope of this paper.

5.3.2 Sequencing layer

This level decomposes the EPT sent by the Mission layer into a sequence of lateral (LNAV) and vertical (VNAV) maneuvers. These maneuvers describe the intended path of the vehicle. For example, LNAV maneuvers include straight legs with constant heading, straight legs with constant track (that is, balancing the wind effect), or arcs with constant radius. VNAV maneuvers include level-off segments, flight level changes at constant airspeed, flight level changes at constant vertical speed, etc. The Sequencing layer is in charge of initiating and terminating the execution of each of the scheduled maneuvers, and sending them to the Guidance layer.

5.3.3 Guidance layer

The Guidance layer is responsible for activating the required control mode according to the maneuvers that have been selected at the Sequencing layer. Furthermore, it computes the target value of the controlled variables. This is done by using different guidance algorithms that also depend on the type of LNAV/VNAV maneuver. Control modes and control targets are the outputs of the Mission Manager, and thus are sent to Flight Control System interface. The Guidance layer also provides flight envelope protection by limiting the range of the target values according to the aircraft performance, and thus preventing the loss of control.

6 Automatic deployment to XtratuM

The deployment process deals with the process of porting designs from a design platform to a target execution platform. In our case the design platform uses MBD on Matlab/Simulink and the execution platform is the XtratuM platform described in Sec. 4. Deployment is the last step in the development phase.

MBD technologies usually provide automatic code generation from a symbolic or high level model. This is the case of the Simulink Coder. It is able to generate C/C++ code from Simulink models, Stateflow charts, and Matlab functions. Run-time execution targets include POSIX or ARINC-653 compliant systems, like XtratuM. It is worth noting that the automatic coding process is not certified, but in any case it helps the coding task very much, as long as the produced code is understandable, well structured, and does not make use of non approved language constructions for certification.

However, some additional issues need to be addressed before automatic code generation can be performed. These problems are mainly related to how to map Simulink abstractions to XtratuM abstractions and services, and to target configuration issues. This requires developing some tools to fully automatize the process. In summary, the deployment process consists of the following steps:

6.1 Configuring partitions

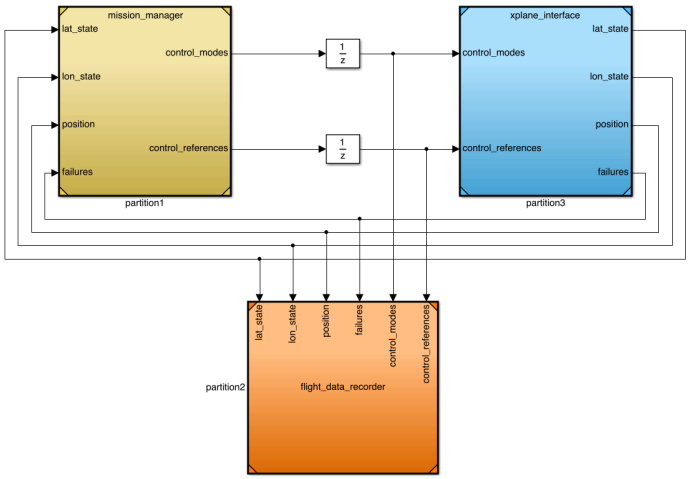
The first stage of the porting process is related to the partitioning of the application. The goal is allocating the Simulink blocks that comprise the application to different XtratuM partitions and configuring them. This consists of:

- *Mapping Simulink abstractions to XtratuM abstractions.* Allocating Simulink blocks to different partitions is done by using Simulink *Referenced models*, which allow to include one model into another by referencing it. Simulink blocks belonging to the same partition are grouped into a same Simulink Referenced model of the top-level diagram (see Fig. 7(a)). Moreover, Referenced models have an interface that consists on a series of inputs, outputs and parameter arguments, which are mapped to XtratuM sampling ports. All these tasks are accomplished through a tool that *identifies* partitions, ports, and channels from the Simulink top-level model.
- *Setting the operating system for each system partition.* Setting the OS is done through a Simulink menu as shown in Fig. 7(b). Several guest real-time OSs are available, such as LithOS, Partikle (POSIX type), or Linux. The partition can be also configured to run without OS (bare partitions). If the selected OS supports multitasking and it has been enabled in the Simulink Model, the user is asked to enable it in the XtratuM application too.

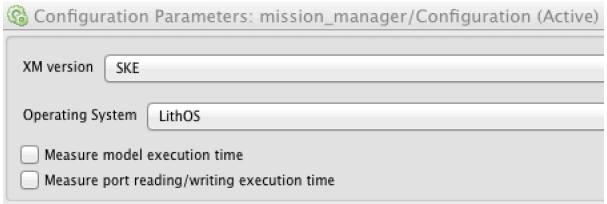
6.2 Code generation

This is mainly done through the automatic code generator (Simulink Coder) that generates C/C++ code from Matlab, Simulink, and Stateflow blocks. Even though, there are some important aspects that need to be addressed:

- *Customized generated code.* Some Simulink generated code needs to be tuned to the target partition. In the case of the SKE, this includes the calls to the ARINC-653 services, but in general, it is related to the sampling ports creation, reading and writing mechanisms, the generation of some support files with constants definitions, or the way POSIX threads and semaphores are used, among others. The way to accomplish this is using the *Target Language Compiler* (TLC) to customize the generated code (see [15]). To this end, we have created some TLC files and modified some of the ones provided by Simulink in order to make the generated code runnable on XtratuM.
- *Concurrency model.* The multitasking model must match the multitasking model of the target partition, so it has to be properly configured. Multitasking in LithOS partitions is implemented through the ARINC-653 process concept, whilst Linux and Partikle systems use POSIX threads. In bare applications (those running on XtratuM directly), there is no support for concurrent execution so no multitasking calls to the OS must be generated.
- *External libraries.* Simulink Coder assumes that the standard C libraries are available, and generates calls to their functions or defined constants, waiting for the links to be resolved when compiled. Some of them are not included in XtratuM; in theses cases, the user may have to provide the missing resources. However, SKE can have access to all underlying Linux services (see Sec. 4.1), so this problem can be avoided.



(a) Referenced models and signals in the top-level diagram of a Simulink design represent partitions and communication channels in XtratuM



(b) XtratuM configuration window (detail)

Fig. 7 Automatic deployment process using Matlab/Simulink

6.3 Configuring the XtratuM project directory

XtratuM requires a number of makefiles and configuration files which are difficult to generate. An automatic deployment tool has been developed that automatizes this task from the information contained in the Simulink models. In the case of the SKE version, this tool generates the following files:

- *Makefiles* describing the rules to compile the source code of each system partition. Although Simulink generates its own makefile, it is barely useful because compiling applications for XtratuM requires some particular rules [8].
- *Hypervisor configuration files* defining system resources, and how they are allocated to each partition. This includes aspects like the number of partitions and their communication ports, or cyclic plan information. In the bare metal hypervisor version, it also includes information regarding memory allocation that must be supplied by the user at a later stage.
- *LithOS configuration files* specifying the maximum number of the different resources used by the partition (processes, events, semaphores, etc.). Each LithOS partition has its own configuration file.

The last step of the XtratuM configuration process consists on generating the structure of the project directory and merging there the different files that are

required to compile the application, including source files, libraries, makefiles, and configuration files.

7 Results

This section presents the results of the automatic deployment process of the Mission Management System presented in Sections 3 and 5 to the development platform in Sec. 4. Note that flight performance of the proposed system flying in the simulation environment is out of the scope of this paper as it was evaluated in previous works of the authors [4, 17, 18].

The starting point is the Simulink design in Fig. 7(a). It shows a periodic application with 3 *referenced models*: the yellow box is the Mission Manager component, the orange box is the Flight Data Recorder, and the blue one is the X-Plane interface. Different signals communicate each model.

From that design and following the workflow explained above, the automatic porting process detected 3 partitions and 12 communication channels. All partitions are executed on top of LithOS and run at the same sampling rate. Regarding the concurrency model, partitions allocating the Mission Manager and the Flight Data Recorder are single-tasking, while the one executing the X-Plane interface implements 2 processes: one reading and the other writing simulation data, respectively. These results are summarized in Table 1.

After mapping Simulink abstractions to XtratuM abstractions, next step is invoking the code generation function. In this case, it produced 78 source files with more than 15.000 lines of code (see Table 2).

Last step concerns time management. Even though time requirements cannot be validated using the SKE (see Sec. 4.1), it is necessary to define a cyclic plan for the application running in the development platform. This is a design issue that requires estimating the Worst-Case Execution Time (WCET) of each partition. In such a complex system, guaranteeing that all possible execution paths are exercised

Table 1 Results of the interpreter function in the Mission Management System example

Partition ID	Name	Guest OS	Sampling rate	Processes	Input/output ports
1	mission_manager	LithOS	20 Hz	1	4/2
2	flight_data_recorder	LithOS	20 Hz	1	6/0
3	xplane_interface	LithOS	20 Hz	2	2/4

Table 2 Results of the code generation function in the Mission Management System example in terms of number of source files and code lines

Category	Source files	Code lines
Partition ID1	54	13088
Partition ID2	8	855
Partition ID3	1	618
System files	15	590
Overall	78	15151

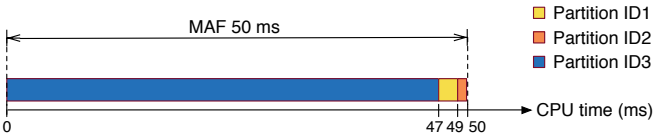


Fig. 8 Proposed cyclic plan for the Mission Management System

is a difficult task that can be only proved using a profiling tool. Simulink includes one, but execution time in Simulink is not real-time, so results cannot be exported to XtratuM. The same occurs with the SKE.

This is why an statistical approach is followed: WCET is estimated measuring the execution time of each partition during a hugh amount of time in different test conditions. Although this does not guarantee that all possible paths are visited, results are statistically valid (i.e. probability of finding a worst case is low).

The resulting cyclic plan is presented in Fig. 8. It is defined with a major frame of 50 ms, and 3 time slots, one per partition: slots for the Mission Manager and the Flight Data Recorder are 2 ms long and 1 ms long, respectively (equal to their WCET), and the remaining 47 ms have been allocated to the X-Plane interface partition.

8 Discussion

Previous results show that an application prototyped in Simulink can be ported to an execution environment based on XtratuM using the proposed design methodology. The porting process can be automatized through several steps, and the resulting application code is readable and well structured. Automatic code generation is the key feature as it avoids coding errors, thus saving time and effort. This is specially relevant in such a complex application, with several interdependent modules and up to 15.000 lines of code.

The use of partitioned architectures in avionics systems poses two major advantages on the design. The first one is related to the certification process. Spatial and temporal isolation allows the designer to separate critical and non-critical functions. In the proposed example, the Mission Manager System is often listed as a DAL C or D, while the Flight Data Recorder is a non critical function (thus Level E). Allocating these functions in separate partitions (as in Fig. 3) allows to reduce the number of code lines that require certification (in this case, a reduction of 5.65% according to Table 2).

The second advantage is also related to fault isolation but from a functional point of view. The choice of an appropriate architectural design permits that the loss of a given partition does not affect the remaining components. Considering the system in study, the Mission Manager is clearly the most complex system. An exhaustive testing strategy is required, but still software errors may appear at later stages of the design process. If this occurs during a flight, losing the Mission Manager partition will limit the system functionality as *Mission* and *Contingency* control modes will not be available; however, this partitioned design still allows to fly the aircraft through the *Direct* and *Autopilot* control modes, which do not rely on mission planning functions.

9 Conclusion

Aerospace applications requiring a high level of integrity must satisfy rigorous design methods and verification standards. This paper has presented an approach for developing avionics applications using Model-Based Design and porting those designs to an ARINC-653 compliant execution platform called XtratuM. Developing and deploying a real and complex application on XtratuM is key for gaining experience in this process. The design of an RPAS Mission Management System has been a good benchmark for analyzing all requirements and validate XtratuM. The paper outlines the design methodology, the software architecture, the relevant standards for developing an avionics application, and the key issues for porting it and testing it on an emulator version of XtratuM. The experience has shown that the proposed design methodology and the execution environment are a good platform for designing and qualifying avionics applications, as they allow rapid prototyping and the design of test cases to check requirements from the early stages of the design process. Future work is to deploy the proposed application to an embedded platform running the native version of XtratuM, what requires minor modifications of the porting process here presented.

References

1. Aeronautical Radio, Inc.: ARINC specification 653-1. Avionics Application Software Standard Interface (2003)
2. Bonasso, R., Kerri, R., Jenks, K., Johnson, G.: Using the 3T architecture for tracking Shuttle RMS procedures. In: Proceedings of the IEEE International Joint Symposia on Intelligence and Systems. IEEE (1998)
3. fentISS: XtratuM Hypervisor Emulator (SKE) start guide. Tech. rep., Universidad Politècnica de València (2015)
4. Fons, B.: Plataforma para diseño y ejecución de aplicaciones de aviónica. Master's thesis, Universitat Politècnica de València (2013)
5. International Civil Aviation Organization: Doc. 9613 AN/937: Performance-based Navigation (PBN) Manual, 4th edn. (2013)
6. International Civil Aviation Organization: Doc. 10019, AN/507: Manual on Remotely Piloted Aircraft Systems (RPAS), 1st edn. (2015)
7. Koehl, D.: SESAR initiatives for RPAS integration. In: ICAO Remotely Piloted Aircraft Systems Symposium. Montreal (2015)
8. Masmano, M., Ripoll, I., Crespo, A., Metge, J.: XtratuM: a hypervisor for safety critical embedded systems. In: 12^{th} Real-Time Linux Workshop (2009)
9. Masmano, M., Valiente, Y., Balbastre, P., Ripoll, I., Crespo, A., Metge, J.: LithOS: a ARINC-653 guest operating for XtratuM. In: 12^{th} Real-Time Linux Workshop. Kenia (2009)
10. McCarley, J.S., Wickens, C.D.: Human factors implications of UAVs in the national airspace. Tech. Rep. AHFD-05-05/FAA-05-01, University of Illinois, Institute of Aviation, Aviation Human Factors Division (2005)
11. North Atlantic Treaty Organization: STANAG 4703: Light Unmanned Aircraft Systems Airworthiness Requirements. NATO Standarization Agency (2014)
12. Radio Technical Commission for Aeronautics (RTCA): DO-178C/ED-12C Software Considerations in Airborne Systems and Equipment Certification. RTCA (2011)
13. Ribeiro, L.R., Oliveira, N.M.R.: UAV autopilot controllers test platform using Matlab/Simulink and X-Plane. In: 40th ASEE/IEEE Frontiers in Education Conference. IEEE, Washington (2010)
14. Spitzer, C.R.: Digital Avionics Handbook: Elements, software and functions, 2nd edn. CRC Press (2006)
15. The MathWorks Inc.: Simulink Coder Target Language Compiler (2012)

16. Usach, H.: Integridad y tolerancia a fallos en sistemas de aviónica. Master's thesis, Universitat Politècnica de València (2014)
17. Usach, H., Fons, B., Vila, J., Crespo, A.: An autopilot testbed for IMA (Integrated Modular Avionics) architectures. In: Automatic Control in Aerospace, vol. 19, pp. 435–440 (2013)
18. Usach, H., Vila, J., Crespo, A., Yuste, P.: A highly-automated RPAS Mission Manager for integrated airspace. In: Proceedings of the 5th International Conference on Application and Theory of Automation in Command and Control Systems, ATACCS '15, pp. 11–20. ACM (2015). DOI 10.1145/2899361.2899363

Hèctor Usach is a Ph.D. candidate at the Universitat Politècnica de València (UPV), Spain. He received his B.S. in Aeronautical Engineering in 2012, and his M.S. in Automation and Industrial Computing in 2014, both at the UPV. His research interests include the insertion of RPAS in integrated airspace and mission management issues. He is member of the UPV's research group on avionics and also supports the teaching of different courses of the Aerospace Degree.

Juan A. Vila is a professor of Air Navigation and Air Traffic Management at the Universitat Politècnica de València (UPV), Spain. He received his B.S. degree in Industrial Engineering from the UPV in 1985 and his M.S. and Ph.D. degrees, both in Computer Science, from the UPV in 1994. His research initially concentrated in the field of real-time embedded systems where he has published about 15 papers in journals and more than 50 contributions to conferences. He collaborated in the introduction of the Aeronautics curriculum at the UPV where he coordinates the specialization of Air Navigation and where he leads a research group on avionics.

Alfons Crespo is a professor of the Department of Computer Engineering at the Universitat Politècnica de València (UPV). He received his Ph.D. in Computer Science in 1984. He held the position of associate professor in 1986 and full professor in 1991. He is the head of the Industrial Informatics group and has lead several European and Spanish research projects. His main research interest is real-time systems, including real-time virtualization, scheduling, hardware support, and integration. He has published more than 80 papers in specialized journals and conferences in the area of real-time systems.

Pedro Yuste is a professor in the Design Engineering School at the Universitat Politècnica de València (UPV), Spain. He received his M.S. and Ph.D. in Computer Science in 1997 and 2004, respectively, both at the UPV. Current research interests include dependable real-time embedded systems, air navigation, mission control systems and Air Traffic Management.