UAV autopilot architectures versus AADL

Emmanuel Grolleau

UAV autopilot architectures versus AADL

Emmanuel Grolleau
LIAS, ISAE-ENSMA, Chasseneuil, France; email: grolleau@ensma.fr

Abstract

In the Comp4Drones European projects, the software architecture of several open source off-the-shelf autopilots was studied in order to propose a common abstraction, the reference architecture. Several tools were developed in order to give means to drone manufacturer to assess the safety of their underlying autopilot altogether with their custom components. AADL was one of ADLs closest to the needs for such a representation, but did not completely fulfil the needs. This paper addresses the main points where AADL lacked in offering all the required concepts.

Keywords: AADL, UAV, autopilot.

1 Introduction

The Comp4Drones ECSEL JU European project took place from 2019 to 2023, and was aiming at providing a framework of key enabling technologies for safe and autonomous drones' applications. The drone manufacturers involved in the project were using, in general, existing open source off-the-shelf (OtS) autopilots, such as PX4, Ardupilot or Paparazzi. They have, nevertheless, always to add software and sometimes hardware components to give an added-value to their drones.

Custom components, depending on the expected response time range, may be located on a companion board, limiting the impact on the stability and responsiveness of the autopilot, but also implying response times in the order of magnitude of tenths of milliseconds.

In order to ease integration of such custom components, UAV autopilots rely on middlewares, such as ROS, or the de facto communication standard MAVLink. They can also provide their own specific middleware, like uORB for PX4, or ABI for Paparazzi.Several middlewares can be interconnected through software bridges. Customising a drone autopilot using middlewares is very common, and usually well documented, but it is at the cost of performance. If the custom component were to have a shorter response time, in the magnitude of milliseconds, then the custom component should reside on the same processor, either in a specific process, if the underlying operating system allows so, or in a specific thread. Paparazzi even allows custom functions to reside within the control loop of the attitude controller, giving the availability for a custom component to be called each time the attitude controller executes. The frequency of such controller varies depending on the frame of the UAV, typically from hundreds of hertz for fixed-wing, to up to several kilo-hertz for a very unstable rotary-winged drone.

In this paper, we deal the way the custom components of an autopilot and an UAV can be represented in order to allow the system safety to be assessed.

2 From COTS to MOTS

From the point of view of a drone manufacturer, an OtS autopilot can be seen not as a Component-of-the-shelf (COTS), but more as a Modifiable-off-the-shelf (MOTS). He has to design and develop his own component, but integrate it in an existing, very complex, piece of software and hardware.

The term open source OtS autopilot is not corresponding to the reality of most autopilots (e.g., PX4, Ardupilot, Paparazzi). Indeed, they provide software repositories for hundreds of autopilots to be generated depending on the frame, controller, operating system, sensors, and actuators. As shown in [1], Paparazzi autopilot repository regroups over 2100 C (mostly) and C++ files, for almost a million lines of code, at least 366 compilation files for over 50'000 lines, over 1200 configuration files for over 160'000 total lines. PX4 is in the same order of magnitude, while more relying on C++ than on C as a programming language. A specific instance of an autopilot uses less than 10% of the whole content of an autopilot repository. The selection of the required source files is made through configuration files and makefiles. As a result, it is quite challenging for a drone manufacturer, supposed to add its custom components to such systems, to vouch for their safety.

3 Specific autopilot architectures

Both for historical reasons and efficiency reasons, the low level code of autopilots mainly relies on a unique thread (called core thread hereafter) in charge of collecting the sensors data and telemetry, then state estimation of the drone, attitude control, actuation of the sensors, position, trajectory control and error handling. If the system is bare-metal (no operating system), then input-outputs are done in interrupt service routines, while if an operating system is present, threads are used to handle input-outputs (see [2] for a detailed presentation).

The core thread works like a cyclic executive, as each of its functions has a specific execution pattern. Some functions are executed each time the core thread is executed, while others have a specific period, are are triggered by events (e.g., specific data available). Some patterns may even include precedence constraints, or transactional behaviour, such as execute this function 20 cycles after this other function is executed.

4 AADL shortcomings

The use of an ADL, hierarchically representing existing autopilots would help drone manufacturers both to understand where and how to add their components, especially if they are to be added at the core of the autopilot. It would also allow for some analysis, validation, or design exploration tools to be used. We therefore created a framework, presented in **??**, taking as imput an autopilot instance in Paparazzi, and presenting the set of threads, as well as a functional decomposition of the threads. This decomposition is obtained through the use of LOW GIMPLE, a low level language artefact generated during compilation with GCC. This choice comes from the fact that the same representation is used for C and C++, and that the code is decomposed in 3-address instructions, simplifying the detection of uses of variables, calls to functions, etc.

The main shortcomings of AADL to finely represent the behaviour of the autopilots were:

- Difficulty to represent the cyclic executive core thread behaviour, with the specific execution pattern of the functions.
- Difficulty to abstract the (very common) utilisation of the middlewares, without representing the whole path followed by the data, knowing that middlewares have specific scopes (e.g., intra-thread for ABI, inter-thread for uORB, inter-processes and systems for ROS).
- Since hundreds of functions are called within a thread, ability to represent hierarchically the content of a function, sub-functions, etc.

References

[1] S. Kamni, A. Bertout, E. Grolleau, G. Hattenberger, and Y. Ouhammou, "Easing the tuning of drone autopilots through a model-based framework," *Journal of Computer Languages*, vol. 77, p. 101240, 2023.

[2] G. Hattenberger, F. Bonneval, M. Ladeira, E. Grolleau, and Y. Ouhammou, "Design of micro-drone autopilot architecture with static scheduling optimization," *Unmanned systems*, pp. 1–14, 2023.