MANAGING SOFTWARE-DEFINED AND RECONFIGURABLE PAYLOADS FOR ADAPTIVE AND FLEXIBLE MISSIONS

Peter Mendham[(1)], Andrew Nairn[(2)]

[(1)] Bright Ascension Ltd, 1 Laurel Bank, Dundee, DD3 6JA, Scotland, UK; +44 (0) 1382 602041; peter@brightascension.com
[(2)] Bright Ascension Ltd, 1 Laurel Bank, Dundee, DD3 6JA, Scotland, UK; +44 (0) 1382 602041; andrew.nairn@brightascension.com

ABSTRACT

With the miniaturisation of high-performance computing an increasing number of CubeSat and small satellite missions are flying powerful computing platforms as payloads. Whilst these platforms offer significant mission flexibility, this flexibility introduces additional management complexity for the payload and mission as a whole. This paper describes the needs of payload software applications and places these in the content of a concrete example software architecture from In-Space Missions' Faraday Phoenix mission. Initial approaches to the problems of software payloads are presented, together with the experience of where this was effective, and where it was less so giving rise to operational inefficiencies and potentially limiting the potential of such a flexible space system.

To conclude, lessons learned are drawn together and placed in the wider context of flexible computer platforms and reconfigurable payloads. Looking to the future, a path to building on the experience gained so far is presented, suggesting ways of improving the capabilities of a management software framework, and helping to unlock the future potential of a new class of software defined payloads and missions.

1 INTRODUCTION

With the miniaturisation of high-performance computing, and its availability in industrial and/or automotive grades, an increasing number of CubeSat and small satellite missions are flying payloads with a powerful and flexible computing platform. In some cases, these include GPUs, FPGA-based systems, AI accelerators or other custom hardware. These platforms commonly run Linux and are used in a payload context either directly as a payload or for on-orbit payload data processing. Whilst these platforms offer significant mission flexibility, this flexibility introduces additional management complexity for the payload and for the mission as a whole.

In most cases this complexity is managed in an *ad hoc* way, on a per-mission basis. This means that similar problems are being solved multiple times, in different mission context, with limited opportunity for sharing of best practice or leveraging of prior solutions. The creation of a more standard, re-usable way of operating these flexible compute platforms would be of great benefit not just to the developers and operators of the spacecraft, but also to those using the satellite platforms.

2 SOFTWARE-DEFINED PAYLOADS

Over the past few years Bright Ascension has gained mission experience with the flight use of high-performance FPGA-based computer platforms from different payload vendors, especially when used as software defined radios (SDRs). These missions have varied needs, but in common they have to be able to easily replace one or more payloads, or payload data processing applications, together with FPGA logic and supporting elements of the operating system such as device drivers.

The replacement operation may need to be completed as often as more than once per orbit, depending on the pattern of use of the payload.

Software-defined payloads can be used across a very wide range of applications, including:

- onboard image processing such as feature extraction;
- image classification using Machine Learning (ML);
- software defined radio applications;
- data and image fusion; and
- onboard autonomy.

Such applications are typically composed of a combination of software applications, low-level software (such as drivers) and hardware-specific applications (such as for GPUs or ML co-processors) or logic (for FPGAs). There may be a need for a single hardware compute platform to be able to support the concurrent execution of multiple software payloads, or applications, provided that there is no contention for resources.

3 CASE STUDY: FARADAY PHOENIX

To better illustrate the needs of software payloads, and to present a prototypical solution, we will use the Faraday Phoenix mission as a case study. Faraday Phoenix is a hosted payload mission built and operated by In-Space Missions Ltd, with Bright Ascension responsible for flight and ground operations software. Faraday Phoenix was launched on 30th June 2021 and is currently engaged in a mixture of payload operations and commissioning.

The spacecraft is a 6U CubeSat and includes five flexible computer platforms, primarily used as SDR payloads but which may be employed for other purposes. A range of customers share access to the spacecraft and have the option of providing in flight software (and FPGA firmware) updates as part of nominal operations. The software framework is responsible for enabling the management of these payloads, as well as providing a range of services to payload applications to support their operation.

Figure 1. Faraday Phoenix Spacecraft

There are two types of payload compute platform present on Faraday Phoenix:

- GomSpace Z7000 SDR based on the Xilinx Zynq 7030; and
- Xiphos Q8S configured for use as an SDR and based on the Xilinx UltraScale+.

In both cases the computer runs a custom distribution of Linux, and the FPGA portion is at least partially available for use by payload applications. Interfacing to the rest of the platform is via a CAN bus. The power to payload computers is switched separately, with computers being powered for specific payload operations.

In addition to the flexible computer platforms, Faraday Phoenix has a main onboard computer and a separate, dedicated AOCS and avionics computer, both of which run a real time operating system. These latter two computers are responsible for platform operations.

To facilitate integration with the spacecraft platform, each payload computer runs a data handling application which is started on boot and is responsible for management of the computer platform. The aim was to make the concept of operations for uplinking, updating and executing a payload application equivalent, irrespective of the payload computer platform. Additionally, to make development of payload applications easier, the intention was to make the concept of operations for applications as uniform as possible, irrespective of the underlying hardware. Clearly, full hardware abstraction (e.g. including the programmable logic, co-processors etc.) is neither possible nor desirable, but a standard set of operating principles simplifies the learning curve for payload developers.

4 USE CASES AND USER NEEDS

In assessing the needs from software payloads, a good starting point is to consider the roles of those involved in the software payload development and operations process.

- A **payload software developer** is responsible for creating the software payload, comprising the application and any other necessary elements (FPGA logic, device drivers, etc.). In practice this is likely to be multiple engineers, due to the breadth of skills involved.
- A **platform manager** is responsible accepting software payloads for execution on the spacecraft platform. This usually involves the validation of the software payload and the demonstration that the payload cannot have a negative impact on the platform.
- A **payload operator** passes operational requests for the software payload to platform operator who will then operate the software payload on the payload operator's behalf. These operational requests will need to specify the actions to be performed on the software payload, together with any constraints (e.g., time, location etc.).
- A **platform operator** is responsible for operating the complete spacecraft, including payloads. As such, the platform operators must take operational requests from, potentially multiple, payload operators and integrate these requests into an operations plan. The platform operator is also responsible for arranging and managing data transfer: both of data produced by the payload, and any data inputs required by the payload.

On the Faraday Phoenix mission, the primary need is for software-defined payloads which make use of the payload computers as SDRs. These payloads are comprised primarily of a software application, which executes as a Linux user mode process, but also other artefacts to configure the hardware and facilitate interfacing:

- an FPGA logic definition "bitfile";
- one or more kernel mode device driver modules; and
- a device tree overlay, which assists with hardware and kernel configuration.

These artefacts generally have requirements from their execution environment, including:

- access to file system storage space;
- provision of “live” platform information such as synchronised time, position and attitude in a well-defined frame of reference;
- ability to access platform information or facilities, such as the determination of future location through orbit propagation or the placing of onboard planning requests; and
- access to operating system utilities and libraries for common data handling tasks such as compression or encryption.

Although based on the specific case of Faraday Phoenix, our experience with other missions indicates that these needs are broadly representative.

The payload software developer needs a **clear specification of the execution environment** for the software payload, together with a definition of what facilities are available and how to access them. These facilities will typically need to include **timing, position, and attitude services** as a minimum. There should be simple and **well-defined operational mechanisms** for the software payload to expose its functionality to the wider system. It is these mechanisms which can be used by payload and platform operators to work with the payload. A **representative test environment** is also necessary, to prepare for the passing of the developed payload to the platform manager.

Management and execution of the software payload can be facilitated by the **clear association of all relevant artefacts** (applications etc.) such as through packaging. These can then be transferred to the spacecraft and managed on board easily. The execution or application of a software payload may require the computer platform to be reconfigured or even restarted. As the specifics of the compute platform are irrelevant to the functional operation of the software payload, there should be a **clear software payload life cycle** to facilitate development and operations by the platform operator. This also permits the payload operator to specify payload **operations which are independent of the specifics of the platform**.

These needs became clear across both the development and operational phases of Faraday Phoenix and led to the development of prototypical solution for software payload management.

5 A SERVICE-ORIENTED SOLUTION

The platform and operations software on the Faraday Phoenix mission is built using Bright Ascension’s GenerationOne technology, a framework and development approach which is model-based, component-based and service oriented. It is the latter of these characteristics which was leveraged in creating a solution to the problem of software payload management on Faraday Phoenix. All possible software payload interactions are captured as well-defined services in a language- and technology-independent way. A software payload is characterised by:

- the services it provides, including the identities and characteristics of the concrete functional elements of the payload provided via those services (such as telemetry or configuration parameters); and
- the services it requires, and which are expected to be provided by the execution environment.

The definition of the payload, in terms of these services, is captured in a machine-readable form and provides a contractual baseline for the function and operation of the payload. Development tools were provided to payload software developers to facilitate the integration of bespoke payload software with standard platform services. Payload services used in this way include:

- an action service to permit commanding of the payload;
- a parameter service to expose payload read-only telemetry parameters and read-write configuration parameters;
- an event service to permit notification events to be passed between payload and platform (and *vice versa*);
- a file system service, to permit uniform access to a physical or virtualised file system;
- a record store service, to permit uniform access to data structured as regularly-formatted, indexed records (can be used as a packet store);
- a time distribution and synchronisation service, providing synchronised time across the spacecraft; and
- a position distribution service, providing regular and frequent updates of the spacecraft position and attitude.

The services provided by the software payload can be used as part of payload operations with the same semantics as any other element of the spacecraft, platform or payload, making operations seamless and uniform. An additional software management service is provided by the data handling application on the computer platform which a uniform way of managing available software payloads, including their execution. The various artefacts comprising a software payload are packaged together as an archive, which can be used for transfer and management of the software payload as a whole and additionally indicate to the data handler which elements are necessary for execution. This assist with management of the life cycle of a software payload, especially where there is a need to restart the payload computer platform in order to apply one or more of the artefacts. This is the case on both types of computer platform on Faraday Phoenix for the application of alternative FPGA logic.

6 OPERATIONAL EXPERIENCE

The use of an existing software framework, with an established and mature set of development tools, provided a robust starting point for payload software developers for Faraday Phoenix. This approach had the advantage that that the development principles for the payloads were well-defined and guaranteed to integrate with the spacecraft platform.

The most important factor in easing integration, irrespective of the specific software framework in use, was the clear definition, and use of, a uniform set of services for the interactions between a software payload and its execution environment, including the wider spacecraft platform. This approach carried significant benefits for:

- payload onboarding;
- payload testing;
- exchange and agreement on operations; and
- transfer of data and payload artefacts.

A core set of services has been provided on Faraday Phoenix, with the primary focus on real time interactions. There is the potential for a more interesting range of software payloads through the provision of additional services, such as future/past position information derived from orbit propagation. The machine-readable definition of the payload, in terms of these services, eased coordination and communication between payload and platform teams and proved to highly beneficial in reducing ambiguity and speeding up joint activities.

The packaging of software payloads, using the simple archive mechanism, proved to be beneficial but limited in flexibility as each software payload had to be completely self-contained. A description of the archive's contents was provided effectively through file naming, an approach which has not limited operations on Faraday Phoenix so far, but could easily do in more general cases, where multiple artefacts of the same type are required.

Perhaps the most prototypical aspect of the software payload framework implementation for Faraday Phoenix is the data handler management of software payload life cycle. Whilst different artefacts, such as drivers and device tree overlays, are managed by the data handler, the specific needs of the underlying hardware platform, such as a need for a restart under certain circumstances, are not fully abstracted. This means that the platform operator has to be aware of specific operational differences when working with different software payloads or computer platforms.

Additionally, each software payload is treated entirely in isolation: there is no means for specifying shared artefacts or dependencies between software payloads. Without human operator intervention, based on an understanding of the implementation of a software payload, it is possible that a transition from one software payload to another would involve unnecessary computer platform restarts as artefacts such as FPGA logic is treated as being different when in fact it may be the same.

7 LESSONS LEARNED AND FUTURE EVOLUTIONS

The lessons learned from across the development and operational life cycle on Faraday Phoenix present an opportunity for the evolution of the basic software payload approach to a more fully featured solution which better meets the full range of user needs and provides a more flexible and complete platform for a wider range of software payloads.

Some aspects of the existing solution, notably the service-oriented and model-based approach, have been demonstrably successful, reducing development time and risk, and improving coordination between platform and payload teams. However, in other areas there is the opportunity for improvement.

- The basic **life cycle management** present in the current solution is insufficiently flexible and does not cover the full range of needs of both the underlying hardware, and also the many different permutations of artefacts which may comprise a software payload. The lack of a mechanism for sharing payload artefacts and managing dependencies imposes some in efficiencies which cannot be avoided without manual operator intervention. This limits the extent to which automation can be easily applied to payload operations.
- The set of services offered as part of the Faraday Phoenix solution addressed the basic payload needs but a broader and more exciting range of payload applications would be enabled by the provision of a **fuller set of services**. As there is nothing to prevent the addition of new services, or the evolution of existing ones (provided that backwards compatibility is preserved), there is nothing to prevent this occurring as an incremental process either across multiple missions or through in-flight software updates of the underlying framework.
- Whilst the mechanisms for describing software payloads in terms of their services which are provided by the GenerationOne technology have proven to be useful, so they also have proven to be limited in their expressiveness for the description of payload operations. A more **expressive model for capturing the specification of payload operations**, and the potential implications for the platform in terms of data produced and even resources consumed (such as power or storage), would provide a powerful foundation for coordination

of payload test and operations. Additionally, such information may form a useful base for supporting automated planning of payload operations.

- The testing and validation of software payloads on Faraday Phoenix was primarily supported through the provision of dedicated hardware test environments. A more flexible, and scalable, approach would be to provide a suitable **sandbox test environment** based on a simulated execution environment which may be able to represent some, or all, of the resources available to the payload computer environment. The ability of a test environment to simulate complex hardware, such as an FPGA, will always be limited, but if facilities are provided for the injection of pre-prepared data (such as that produced by a dedicated logic simulation tool) then a sandbox could be invaluable for the testing of application-level software.

With these main improvements in place, the opportunity is to then expand the approach to cover the management of additional, special-purpose, hardware resources, such as co-processors and accelerators.

8 SUMMARY AND CONCLUSIONS

A clear need has been identified for the use of software-defined payloads in wide range of missions. Through practical experience, especially through the Faraday Phoenix mission presented here as an illustrative case study, a generic set of user needs for software payloads have been identified. In reaction to these user needs, a prototypical solution has been developed and operated. The solution is based on Bright Ascension's GenerationOne technology, making specific use of the service-oriented and model-based aspects to directly address key user needs. This has proven successful in providing a foundation for software payload development and execution but has also highlighted some important areas where there is opportunity for improvement in order to realise the potential of software payloads.

The next steps will be to improve the expressiveness and interactivity of the software payloads which can be specified, and to define more tightly the associated execution life cycle. By combining improvements in development tools, which support the increased expressiveness, with additional execution and test tools, it is hoped that a new framework version could be applied to a future class of spacecraft platform to enable a broader range of payload applications.