# Hummingbird

# A Service Based Open Source Ground Segment for Small Satellites

**Mark Doyle[1], Johannes Klug[2]**
*mark.doyle@logica.com, johannes.klug@logica.com*
*Logica Deutschland GmbH & Co. KG, Darmstadt, Hessen, D-64295, Germany*

**The space industry is entering a new age, driven by low-cost hardware and software solutions. These modern solutions are driven by practical considerations, focusing on integrating and adapting "off-the-shelf" products in place of specialised solutions. To address this need, for both new and experienced space industry software teams, the authors present Hummingbird, an open source monitoring and control framework. By reusing and integrating existing software components, the Hummingbird software packages provide a ground segment for small satellites. As a consequence of heavy reuse of open source software, we have shown that implementation effort and cost can be reduced significantly whilst stability and quality increase. As an open source project, Hummingbird encourages contributions from users in the field, and is available free of charge to interested parties. Hummingbird is licensed under an open source license, making it an ideal starting point for small commercial or university teams.**

## I. Introduction

This paper gives a current status report of the Hummingbird monitoring and control framework for small satellites. The paper will explain the motivation that led up to the project, followed by a summary of the methodologies and technology stack employed. It will provide a brief report on the current status of development, split into subsections for the architecture's three logical tiers, namely Transport, User Interface, and Application level. Finally, in the Results section we review our work from a software perspective, which feeds into this paper's conclusion.

## II. Motivation

As software engineers working on systems commissioned by Space Agencies, the authors have developed a good understanding of what makes their software special. Monitoring and controlling spacecraft and their payloads is a specialised task, but not all aspects of that task have to be carried out by specialised software.

We are experts at implementing the application logic, but we are not necessarily experts at designing and building message brokers, mediation frameworks, or 3D visualisation tools. However, there are groups of developers who are building high quality software packages that solve these problems. Some of these packages are open source and free to use, and it is these that we are focusing on. When the systems currently in use by the Agencies were designed, the open source market was not nearly as diverse, and there was neither the quantity nor quality that exists today. In addition, even the operating systems we run our systems on were not nearly as capable. When faced with software engineering challenges, too often the only choices were building a certain functionality from scratch, or licensing it from a commercial vendor. The first choice means additional effort in creating the desired functionality in the first place, plus maintenance effort later on. The latter choice often involves a price tag, possibly including additional licensing costs and the potential for vendor lock in, that is, your business is now dependant upon another.

[1] Software Engineer at Logica, Rheinstraße 95, 64295 Darmstadt, Germany
[2] Software Engineer at Logica, Rheinstraße 95, 64295 Darmstadt, Germany

In the past engineers were forced to develop their own solutions to the problems they faced. This meant covering a lot of areas outside of their expertise using the techniques of the time. The risk averse nature of the space domain has led to this software still being in use today. When compared to current products in industry this software is showing its age. It has a large number of known bugs, requires esoteric knowledge to configure and stabilise, is overly complex and offers little in the way of modern interfaces.

As a consequence of this there is no enterprise grade monitoring and control software available for universities. Agencies do provide systems free of charge but aside from the bureaucracy of obtaining a license they suffer from the issues noted above. They are far too complex for a university to use; even the agencies themselves with the help of experts in the field can take more than a year to tailor a system to a mission.

In recent years the open source market has developed considerably covering many areas of software, each one the focus of experts in that chosen field. Utilising this software we thought we should able to emulate other industries and create enterprise level system without the cost of large scale in-house development or commercial products. The space domain can stand on the shoulders of other software giants.

The other key motivation was to examine the use of scalable service based architecture in Space, specifically ground segment systems. Our aim is to have a system that can be configured to run on a laptop in the field or scaling on demand in the cloud.

## III. Methodology and Technology

At the core of Hummingbird lie a number of ideas, all of which stem from years of experience inside and outside the space domain. First and foremost, we attacked the problem of creating a mission control system from a software perspective. Our experience in the domain provided a set of requirements but the main drivers were from the world of software. The MCS is a piece of enterprise software and should be designed using software best practices, for example, dependency inversion, SOA, design patterns, and a  test driven design approach.

Following the Service-Oriented Architecture (SOA) principles, we aimed at providing cohesive, decoupled components that do not interfere with each other's' internal workings. Each component ought to be simple, well-tested, and ideally rather light in terms of lines of source code. This forces developers to focus on the true domain-specific tasks, and relieves them from having to write wiring and boilerplate code. These problems have been tackled and solved by experts in their field, as described the next paragraph. Additionally, service based designs would allow us to swap implementations, add or remove new functionality, and scale the system, without reworking, and more importantly, breaking the existing codebase. Change is the source of complexity in software, missions and clients have different requirements so we aimed to remove the risk and cost of carrying out changes.

In 2003, Gregor Hohpe and Bobby Woolf published their book "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions". They compiled a fairly comprehensive collection of integration patterns that get used over and over in enterprise application development. By understanding each pattern, it becomes easier to recognise when to use which, and it helps having a common language across the development and client teams. In that sense, Hohpe's and Woolf's book can be regarded a descendant of the original "Gang Of Four" book on Design Patterns in software. The book covers four styles of application integration, and a great number of interaction, mediation, and integration patterns. These patterns include Message Splitter, Router, Resequencer, Aggregator, and many others. The full list[3] ought to cover the majority of most integration scenarios.

Based upon Hohpe's and Woolf's work, a team of software engineers implemented most of the patterns in Java, and gave their work to the Apache Software foundation, who license the software under the Apache Software License. The project is called "Apache Camel", and was started as a sub-project of Apache ActiveMQ, an open source message broker. Together, these two software products form a remarkable message-oriented middleware, and one that, at zero licensing cost, is very affordable compared to the industry's commercial offerings.

The aforementioned products are Java-based, which opens development to a number of other, brilliant technologies available for the Java platform. Apache Maven was chosen as the build management software, but it also takes over duties that belong to technical project management, such as dependency management and life-cycle management, including automated system tests.

From the very beginning, software tests were written alongside each new piece of functionality added to Hummingbird. In many software projects the authors have seen in the Space domain and outside it, automated

[3] http://www.enterpriseintegrationpatterns.com/toc.html

software tests were an afterthought. Too often, this has proven to result in tests of poor quality, and sometimes to bad design of the software itself. The Hummingbird team has chosen to do things right and written only testable and well-tested code so far.

Most software developers are aware of the Eclipse Rich Client platform, as this is the foundation of the popular Eclipse IDE. Its modular approach, using OSGi, to building User Interface applications makes it a perfect candidate for Hummingbird, which is why it was chosen as the baseline for the Rich Client.

NASA has developed and published as open source a visualisation framework that renders three-dimensional displays of the Earth and objects around it (WorldWind). This can be integrated into an Eclipse application, providing 3D displays with little effort.

To sum up this section, the Hummingbird development team makes use of modern and proven technologies that weren’t specifically designed for the space industry but rather for enterprise software as a whole.

## IV. Current Status of Development

Hummingbird is under active development, but a number of components are already in a usable state. This section gives an overview of those components, grouped by functionality.

All components are OSGi bundles and as such plug into any OSGi container and service bus.

### A) Transport

All components that deal with the reception, decoding and encoding of data can be found in the *transport* package of Hummingbird. This is the bridge between the hardware and software.

Using the rxtx library[4], radio communications hardware can be connected via serial links, such as USB. Two frame synchronisation mechanisms have been implemented. The first one conforms to RFC 1055, as proposed by the Internet Engineering Task Force (IETF). The second frame synchronisation mechanism implements the Attached Synchronisation Marker (ASM) method as defined by the Consultative Committee for Space Data Systems (CCSDS)[5].

Also specified by the CCSDS are Transfer Frames (CCSDS 132.0-B-1, TM Space Data Link Protocol), which act on the Data Link Layer as defined in the OSI[6] model. Transfer Frames are the standard means of communication on many Agency space missions. Hummingbird supports encoding and decoding CCSDS Transfer Frames, including full Virtual Channel support.

Moving up in the OSI model, we find the Network Layer, whose role in a CCSDS stack is being filled by the Space Packet Protocol (CCSDS 133.0-B-1). The implementation in Hummingbird offers both decoding and encoding facilities. Both Telecommands and Telemetry packets are handled. Payloads larger than a single packet are supported up to the maximum length specified in the standard.

Once these payloads are extracted, they need processing to turn them into meaningful information. This is where one of Hummingbird’s most powerful components comes in: the Generic Payload Codec. Configurable using uniquely identified parameter layouts, it encodes or decodes bit-packed structures down to single parameters. As far as the authors are aware, this is the only implementation of a bit-packed encoder/decoder that supports parameters at arbitrary positions within the binary stream, and it also handles parameters correctly even if they do not begin and end at byte boundaries. The Generic Payload Codec can be configured using an XML file that describes the overall Telemetry and Telecommand structure. Aiming at broad standards compatibility, Hummingbird’s authors chose to implement a fourth CCSDS standard: the XML Telemetric and Command Exchange (XTCE), which is designed to facilitate inter-agency data exchange. With that design goal in mind it is understandable that XTCE is quite comprehensive. For the sake of quick prototyping the Hummingbird team has therefore chosen not the implement the full width and breadth of the standard, but rather a sub-set that is applicable to the task of decoding and encoding of Telemetry and Telecommands.

[4] http://rxtx.qbang.org/

[5]published standard CCSDS 101.0-B-6 (Telemetry Channel Coding, section 5)

[6] Open Systems Interconnection, a reference model for communications architectures. Defined by the International Organisation for Standardisation (ISO)

The system can leverage existing protocols to send data to the payload codec, the decoupled service based nature is

**B) User Interfaces**

We have a collection of RCP and OSGi plugins that can run in the Equinox OSGi container. This provides a customisable or pluggable user interface. Functionality can be added or removed at runtime. We have integrated the WorldWind OpenGL globe into RCP as a plugin and provide a graphing mechanism.

**C) Application Level**

On the application level, a prototype for parameter calibration has been developed. It outputs parameters that weren't received in the original telemetry downlink, hence they are synthetic parameters.

For the commanding subsystem, a number of components have been implemented that offer lock states and command verification.

A navigation module has been added, which encapsulates OREKIT, a free space flight dynamics toolkit. With it, orbit predictions can be calculated.

These are three example applications, more can be found at the hbird-business repository[7].

## V. Results and conclusion

By removing all non-business logic from the components, utilising interfaces and leveraging Camel we were able to create a protocol agnostic system. Communication between components can travel using 80 different methods, from JMS, CORBA, TCP, UDP, files, email, even Twitter! The interesting result is that the components themselves have no idea how the data travels, it's all taken care of in configuration. Serialisation is another non business issue, i.e., it's a software problem not a space problem. We currently use Java serialisation due to simplicity but using Camel it's a simple matter to swap in something like protocol buffers or XML for example.

This makes all components protocol-agnostic. As discussed, Hummingbird supports the standard CCSDS Transfer Frame and Space Packet protocol stack. If a certain spacecraft does not require these protocols, it is very easy to adapt the routing in the mission control system configuration to not use these protocols but others instead. This flexibility is thanks to the service-oriented approach to designing individual components and the system as a whole.

Unit testing worked! The project gained the obvious quality advantages of stability and assurance, but we found the extra unit testing work actually increased the total development rate. Confidence in refactoring was key to this and also allowed us to redesign large portions of the framework each iteration. Too often in software fear of the cost of redesigning leads to technical debt that ultimately costs more as maintenance effort piles up.

Open source software really works. The authors of the open source software Hummingbird depends on are real experts in their fields working on real life projects across all industries. Support is free via mailing lists and countless websites. The software itself is more robust than many of the commercial packages we have worked with. It is simply not necessary to reinvent the wheel.

Using third party libraries enabled us to think outside the box. Whilst testing the ability to swap out protocols we successfully sent telemetry in ASCII form via email to a running Hummingbird system.

As the first user of Hummingbird, the Thunderbird amateur rocket was successfully flown in late 2011 with Hummingbird as the ground station, receiving live telemetry while the rocket was in-flight. The next Thunderbird models will offer telecommanding facilities, which the ground software will reflect. The Hummingbird team is in touch with two university-driven satellite projects, namely ESTcube at the University of Tartu and STRaND at the Surrey Space Centre (SSC), and one research satellite project (TechDemoSat-1 in the UK). Any one of these missions would elevate the Hummingbird project to full "flight-proven-in-space" status, thereby making it attractive to a wider audience, including university teams developing new spacecraft or operating current missions.
Hummingbird is licensed under the Apache Software License[8] (ASF) 2.0, which is an attractive license both for university and commercial teams.

[7] https://github.com/Villemos/hbird-business
[8] http://www.apache.org/licenses/LICENSE-2.0.html

Because Hummingbird is modular and each component utilises interfaces and services it became a matter of configuration to monitor and control different hardware. Assuming the hardware accepts commands or outputs telemetry Hummingbird's transport layer can be configured to understand that. The application layer that creates commands or displays telemetry is not affected and can therefore, for example, display parameters in a user interface without any changes. This allowed us to use Hummingbird to monitor and control a ground station. The service nature of the components also means we can monitor and control a satellite at the same time, we simply start up another instance of the components.

The Hummingbird project has proven that it is possible to write software that covers the basic functionality of a monitoring and control system for small spacecraft with a minimum amount of own code. This stands in contrast to typical software developments found in the space sector that implement large portions of middleware and other auxiliary application within project scope. Software quality metrics show excellent results[9], and all code is unit tested to a degree not achievable in legacy agency-built applications. The source code itself is very cohesive with nearly no boilerplate code - all of that is managed by the frameworks outside the project scope. This Service-Oriented Architecture (SOA) approach makes it fairly effortless to swap out protocols or other components or even monitor and control different hardware.

## Appendix A
## Acronym List

*ASM = Attached Synchronisation Marker*
*CCSDS= Consultative Committee for Space Data Systems*
*IDE = Integrated Development Environment*
*IETF = Internet Engineering Task Force*
*ISO = International Organisation for Standardisation*
*OSGI = Open Services Gateway initiative*
OSI = *Open Systems Interconnection*
*RCP = Rich Client Platform*
*RDMS = Relational Database Management System*
*RFC = Request For Comments*
*SLIP = Serial Line Internet Protocol*
*SOA = Service oriented architecture*
*USB = Universal Serial Bus*
*XML = Extensible Markup Language*
*XTCE = XML Telemetric and Command Exchange*

[9] Based upon Sonar metrics (PMD, checkstyle, Findbugs, Clover, Cobertura).