Enabling portability across diverse hardware with a model-based approach

Mark McCrum (mark@brightascension.com)
Peter Mendam (peter@brightascension.com)

Bright Ascension Ltd, Scotland, UK

Nanosatellite developers benefit from the availability of a wide range of COTS subsystems from a diverse range of suppliers. Indeed, it is common to find subsystems from multiple competing vendors within a single satellite. Subsystem suppliers vary greatly in their architectural philosophies, selection of interface protocols and adoption of standards. This presents particular challenges for software reuse, because subsystems that are functionally very similar appear completely different from the point of view of software. This paper describes a Model-Based approach to tackling the problem of reuse in such an environment. The approach has been developed commercially by Bright Ascension for a number of years and proven through application to a wide range of missions.

Context

Other than their small size, nanosatellites are characterised by low cost and rapid development times. This, together with standardised form-factors for cubesats, has driven extensive use of COTS subsystems and the emergence of a bewildering range of vendors, each with different and often incompatible ideas of how a satellite subsystem should be built.

To take the example of an Electrical Power System, the AAC Clyde Space ‘Starbuck’ EPS and GOMSpace P60 are broadly comparable from a functional perspective. From an interface perspective however, they could hardly be more different. The Clyde subsystem makes use of an I2C interface with a custom communications protocol and is designed to be driven directly by an OBC. The GOMSpace subsystem uses the CSP protocol over one of two possible physical interfaces. It is designed to function as part of a network which spans the space-ground interface so that it can be commanded directly from the ground without any involvement from an OBC.

Such diversity creates a challenge for software development given the need to produce reliable software in a rapid and cost-effective manner. We can meet this need through software reuse, but to do so we must identify suitable units of reuse and provide a way to integrate these units in different combinations to accommodate the very different scenarios encountered.

GenerationOne

Since 2012, Bright Ascension has been developing a technology called GenerationOne. This combines

- Model-based software engineering, permitting machine comprehension of software architecture and the use of tools to assist with software development and product / quality assurance

- Component-based software engineering, enabling reuse of software across a wide range of scenarios and applications, combining software with its documentation and tests within libraries
- Service-oriented architecture, providing consistent and well-defined semantics for component interactions at all levels, enabling low-level aspects of the system to be expressed as components whilst improving operability.

GenerationOne technology comprises a meta-model definition; a language-independent set of service and protocol definitions; and cross platform tools, and framework implementations for target platforms. The GenerationOne approach permits almost the entirety of a software system to be expressed as components, from hardware drivers and communication protocols to applications. The underlying framework is lightweight and most components are portable across platforms and operating systems.

The flexibility of GenerationOne largely derives from the loose coupling of components which is achieved by the fact that components may only interact by consuming or providing services which are fully defined using the meta-model. This ensures that components can be used interchangeably provided only that their service requirements can be met. The fact that the scope of the meta-model includes the definition of services means that services can evolve and new services be developed to meet emerging needs without the need to change existing tooling or infrastructure. This is important because the appropriate definition for a service may only become clear over time as more real-world examples are encountered and mission experience is accumulated.

Services in GenerationOne

Within the meta-model, a service defines one or more operations. Operations follow one of a number of interaction patterns (for example, request-response) and may have both input and output arguments. A type system allows custom argument types to be defined. Components that consume a service may invoke the service operations either synchronously or asynchronously. All service operations specify a particular end-point, to which they are directed. End-points are identified using the concepts of service channels and selection IDs.

Every service access point provided by a component defines one or more channels on which that service is provided. These are fixed at deployment time. A service channel may represent a node on an onboard bus (for example an address on an I2C bus, or a chip-select on an SPI bus). It may also represent a route through a network.

Selection IDs operate within a given service channel and are defined at run-time. Typically these represent dynamically-managed resources such as file handles.

Components that consume a service do so via the concept of a binding to one or more providers of that service. These bindings are defined at deployment time. They may be ‘open’ which allow the consumer to access any provider or ‘closed’ in which case the provider(s) are fixed at deployment time.

Standard Services

The standard GenerationOne product includes a core set of basic services of proven value. Of these, three are commonly used to represent hardware interfaces and communications protocols:

- The Packet Service (PS) provides the ability to send packets of data to and receive from an endpoint. This maps well on to interfaces between layers of a communications protocol stack and asynchronous communication technologies such as CAN or UART.
- The Memory Access Service (MAS) provides the ability to write to and read from particular addresses of an endpoint. This is representative of memory-like interfaces with a clear initiator-target relationship such as I2C or SPI. Or, at a higher level of abstraction, a flash memory.
- The Abstract Messaging Service (AMS) provides the facility to send abstract messages representing the operations of other services. By providing a mapping for AMS to a communications channel between two nodes in a network, providers and consumers on those nodes (or other nodes reachable via those nodes) can interact. AMS is used as the basis for building distributed systems with nodes communicating via the onboard data-handling network and over the spacelink.

Results

At the time of writing, GenerationOne has been used successfully on 18 spacecraft, with many more under development. These have included subsystems from most of the major CubeSat subsystem manufacturers and have included both highly distributed and single-OBC architectures, bus protocols such as CSP and CAN-TS and a wide range of CCSDS communications protocols. All of this has been achieved with a high degree of code reuse and corresponding savings in development cost and time.

Conclusion

The use of reusable software components within a model-based system has been fundamental to the successful use of GenerationOne technology. It has enabled the rapid production of flight software for satellites which have radically different architectures and which use a diverse range of COTS subsystems and communications protocols. A loosely coupled architecture is achieved by restricting the interaction between components to the provision and consumption of well-defined services. This provides great flexibility in the way that components can be deployed to match the needs of a given scenario. Of course, this flexibility depends on having the right services in the first place. The GenerationOne meta-model allows new services to be defined simply, expressed through a range of interaction patterns with both synchronous and asynchronous operation. It further enables a wide range of service topologies to be captured through the concepts of service channels, service selections and bindings.