The software architecture for TET and AsteroidFinder satellites

Sergio Montenegro, Frank Dannemann

German Aerospace Center (DLR)
Institute of Space Systems – Core Avionic
Robert-Hooke-Str. 7
28359 Bremen, Germany

sergio.montenegro@dlr.de, frank.dannemann@dlr.de

Introduction

The software architecture for TET [1] and AsteroidFinder [2] satellites is a further step in the development line first used for the BIRD [3] satellite. The new improvements add dependability, flexibility and simplicity to a core avionic system already implemented, simulated and tested in similar ESA and industrial projects, proving the basic concept of the architecture. The new core avionic concept targets the problems of complexity, software-hardware interfaces, and the difficulties of merging many different interfaces into a single system by providing a very simple solution of integrated software and hardware, thus eliminating the barrier between the two. Through this concept both the bus control and payload control can be handled through one system. Implementing a complex parallel system safely requires the composition of a network of simple sequential cooperating applications which can communicate by using well defined interfaces. The basic communication principles common to all target systems are decribed in this paper.

The Common Application Layer

The functionality of the board computer is implemented in a network of applications that communicate with each other using simple application interfaces. The communication between e.g. commanding and telemetry is performed using a CommonApplicationLayer and an ApplicationInterface which provide command and telemetry interfaces. The CommonApplicationLayer provides interfaces to all applications, a means of communication between applications, common structure definitions and common data. Each application implements an ApplicationInterface common to all applications. Many applications also require the use of the ApplicationManager which provides some services implemented centrally in the CommonApplicationLayer. Examples of these services are acknowledgement messages ('I'm alive') and registries of unasked housekeeping records. Although implemented in the CommonApplicationLayer, the interface to these methods simply acts as a distributor for these requests to the corresponding executer. 'I'm alive' messages would be forwarded to the Watchdog application and housekeeping records to the Housekeeper application. This distribution is implemented internally in the application interface and is not required to be implemented by each application.

An overview of the described interfaces is given in the following two diagrams (UML and intuitiv):

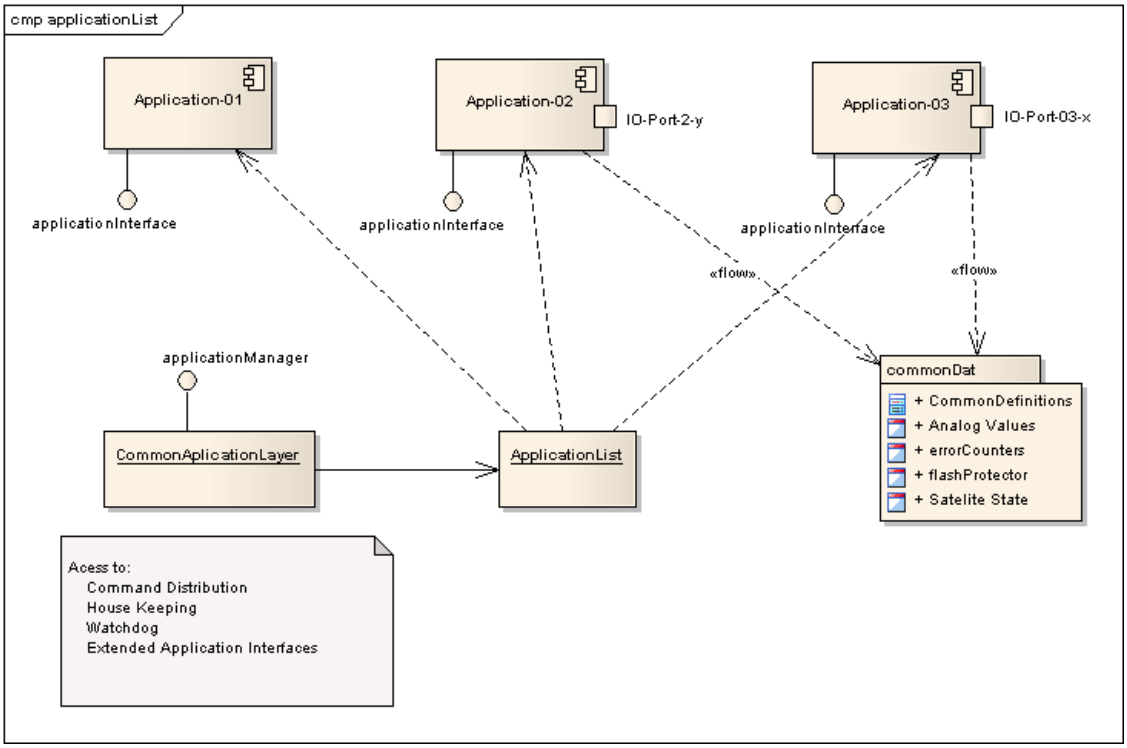

Abbildung 1 Communication Interfaces (UML Notation)

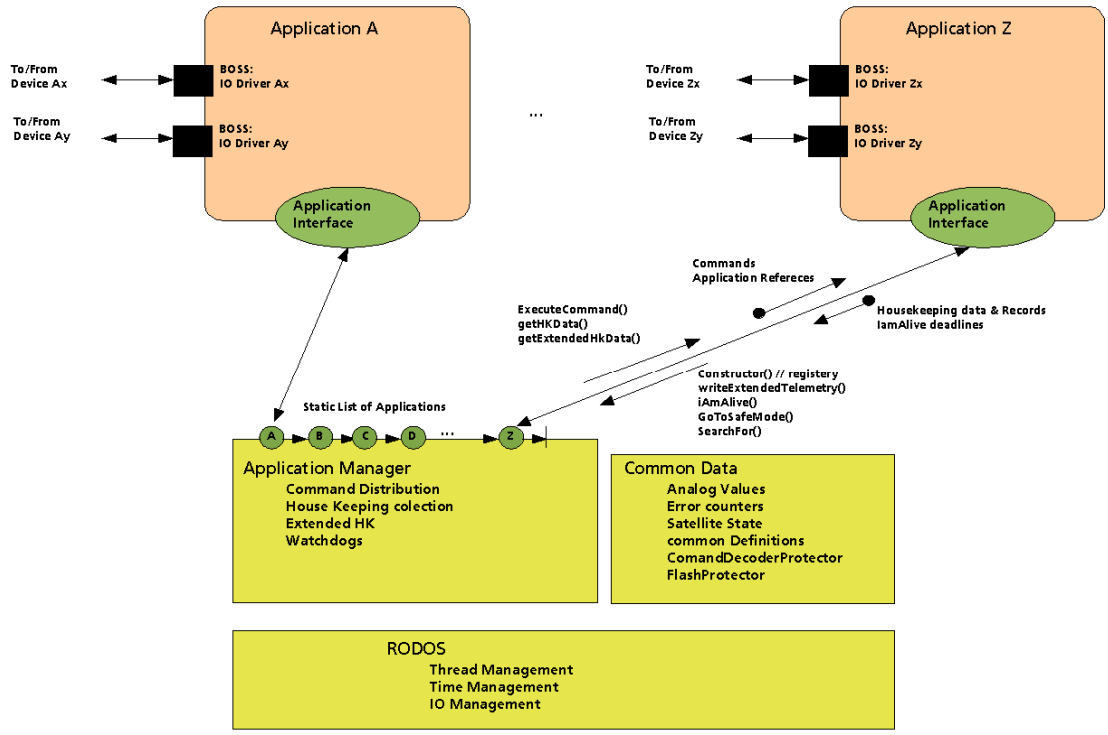

Abbildung 2 Communication Interfaces (intuitive)

Communication Interfaces

Each application implements an application interface which is common for all applications. It has two sides. One is the ApplicationInterface which is provided by each application:

```
class ApplicationInterface : public Application {
  Semaphore protector;
public:
  ApplicationInterface(char *name, long id): Application(name, id) { }
  virtual ~ApplicationInterface() { ERROR("ApplicationInterface deleted"); }

  /** Methods to be implemented by each application ***/
  virtual bool executeCommand(Command &cmd) = 0; ///< returns true if execution ok
  virtual bool getHkData(unsigned char* buff) = 0; ///< returns current true if done
  virtual bool getExtendedHkData(int apid, HouseKeepingEntry& extendedHk)  = 0;
};
```

And additionaly the ApplicationManager which is used (required) by many applications:

```
class ApplicationManager {
public:
            void iAmAliveUntil(TTime until);
            void writeExtendTelemetry(HouseKeepingEntry& extendedHk);
            void goToSafeMode();
            void executeRelayCommand(unsigned char relayCmd);
            ApplicationInterface* searchFor(long id);
};
```

The ApplicationManager is provided by the CommonApplicationLayer to access standard services provided by central applications like the housekeeper, the commander, the watchdog, etc.

The ApplicationManager provides methods to accept commands, deliver standard housekeeping and extended housekeeping.

The ApplicationManager provides some methods which are implemented centrally in the CommonApplicationLayer. These methods are, for example, 'I'm alive until' messages and a registry of unasked housekeeping records. Although the interface to these methods is implemented in the CommonApplicationLayer, it just distributes the request to the corresponding executer as, for example, 'I'm alive until' messages go to the watchdog and unasked house keeping records go to the housekeeper. This distribution, however, is internally implemented in the ApplicationInterface and does not have to be implemented by each application.

Application Discovery and Forwarding of Calls

To identify an application each has to have a numeric identifier called the application ID (APID). This list of APIDs is placed in a common header file and can be used by all applications on the space craft and in the ground station for the telecommanding and housekeeping systems. When defining an application the corresponding number has to be passed to the constructor of the application interface. This ID will be used by the application discovery protocol in order to locate applications, to distribute commands, and to collect housekeeping data.

Like described above, the standard ApplicationInterface has to be implemented and provided by all applications. This interface is used to propagate commands and to collect housekeeping data. Each application shall create an object of this class. This object will be the

interface between the outside and the application. The ApplicationInterface implements the automatic registry of applications. This is done in the constructor of the application interface.

If one application wants to find another, it has to make use of the application discovery protocol (implemented in the class Application) and the Application-ID. If the application is not loaded (linked), for example in a preliminary version where not all applications shall be loaded, then the discovery protocol reports a find failure. Therefore the caller can know that the application and its interfaces are not available.

In the following UML sequence diagram it is shown how an example application can find the Houskeeper-application in order to send extended Housekeeping records to it:

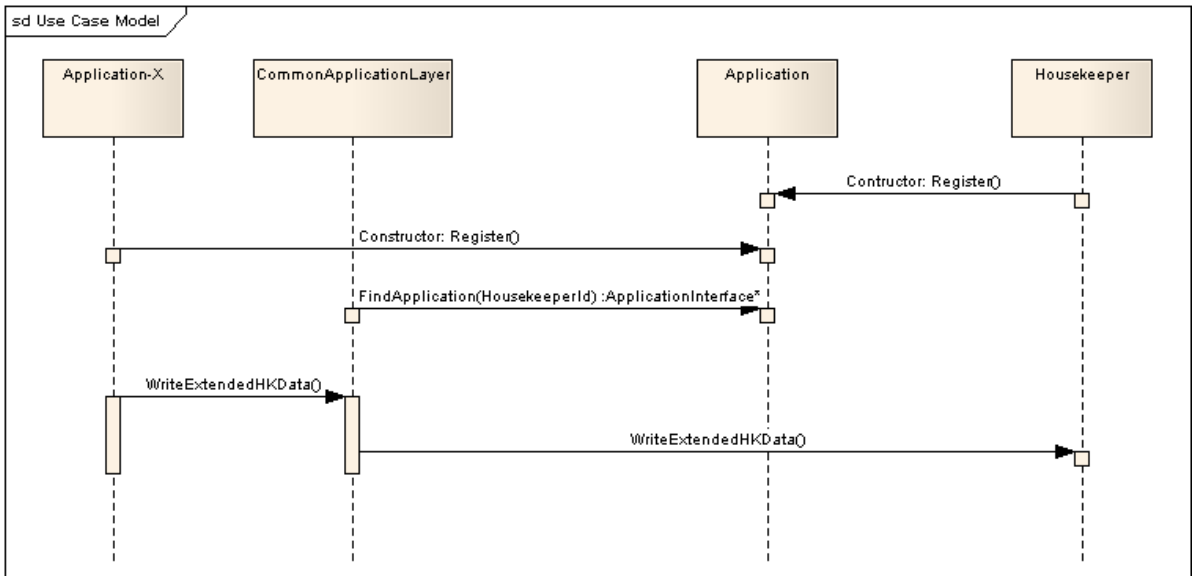

Abbildung 3 Application Discovery and Forwarding of Calls

References

[1] Mottola, S., Börner, A., Grundmann, J.T. , Hahn, G., Kazeminejad B., Kührt, E., Michaelis, H., Montenegro, S., Schmitz, N., Spietz P.: *AsteroidFinder: Unveiling the Population of Inner Earth Objects,* 59th International aeronautical congress, 29th September to 3th October 2008, Glasgow, Scotland.

[2] *Technologieerprobungsträger TET*, http://www.dlr.de/rd/desktopdefault.aspx/tabid-2274//3396_read-5085/

[3] Bärwald W. and Montenegro S., *BIRD-Spacecraft bus controller*, Small Satellites for Earth Observation, Vol. 3, 371-373, 2001