A LIGHTWEIGHT COMMUNICATION PROTOCOL FOR EMBEDDED SYSTEMS

Antonio Latorre, José A. Pulido, Carlos Valle, Juan Pérez, Sergio Gómez de Agüero y Pedro Palomo

Elecnor-Deimos, Ronda de Poniente 19, 28760 Tres Cantos (Madrid), Spain.
antonio.latorre@deimos-space.com

ABSTRACT

This paper presents a communication protocol based on the concepts and definition of the SOIS (Spacecraft Onboard Interface Services) standard. The main objective has been to design a lightweight protocol and execution services layer, suitable for small projects in the field of embedded systems, but flexible and configurable enough to be reused in future projects and re-targeting of current ones with minimum effort.

1. INTRODUCTION

Embedded systems are based on a wide range of architectures and hardware elements, such as different number and model of processors, operative systems and communication buses. However, developing new *ad-hoc* communication protocols for each mission is slow and expensive.

For the last years, a large number of European public institutions and private companies involved in space missions have carried out a strong research effort focused in defining a reference architecture [1. Terraillon, J.L., Jung, A., Arberet, P. (SAVOIR-FAIRE working group) et al (2010). Space On-Board Software Reference] and give a boost to the component-based engineering [2][3]. This way, a new component developed in compliance with the reference architecture would be re-usable in different projects with minimum effort and thus, reducing development periods and costs.

Currently, our company is involved in a number of projects which are in the preliminary phases and even belonging to different technical areas, they share common basic aspects (e.g. multi-node systems which need a communication protocol). For instance:

- GNSS receiver: is a Global Navigation Satellite System receiver platform for the demonstration of advanced receiver technologies and applications (e.g. multi-constellation with GALILEO and GPS).
- UAV control system: for UAV platforms, used to validate new solutions and tackle existing challenges in the aerospace engineering.
- FLYCON: defines a Formation Flying wireless signal enabling the exchange data between spacecrafts at high bit rate for short and medium range operations, integrating also ranging capabilities for achieving relative navigation.

Due to the size and characteristics of these projects, a full compliance with the ESA reference architecture is considered an excessively ambitious objective for our particular needs. However, the concepts fit perfectly, so it is worth to make an effort at this moment, following the general ideas and recommendations made by these domain experts, applying them in order to reuse the developments in several projects, reduce global costs and also minimize the impact of a future migration to the reference architecture.

In summary, the main concepts to be taken into account are the following ones:

- Module-based, distributed, extendable and based on components.
- Layer-based, in order to minimize the impact of changes (variability) in any of the parts of the systems, for instance, the execution platform or the communication data buses.
- Consider interoperability aspects and applicable standards in the more usual domains of our projects: space missions, Unmanned Aerial Vehicles (UAV) and perhaps, radio-communications.
- Take into account non functional requirements, such as the computational model.
- Ease the implementation of safety aspects, fault tolerance and fault detection.

2. STATE OF THE ART

Since several years ago, a number of projects and working groups have been focused in reusing previous software and not starting every project from scratch, centering their efforts in techniques such as a reference architecture, model driven engineering or component-based development. These are some of the relevant projects and activities:

- COrDeT (Component Oriented Development Techniques): definition of a generic architecture for satellite on board applications.
- DOMENG (Framework for Domain Engineering): establishes a work framework in the space systems domain engineering, defining methodologies, models and tools to be used.
- ASSERT (Automated Poof-Based Systems and Software Engineering for Real-Time Systems): bases for the definition of a reference architecture for the ESA according to a model based development process and the correctness by construction theory.
- Securely Partitioning Spacecraft Computing Resources: research on partitioned architectures.
- IMA for Space: workshop to provide ideas and preliminary surveys about how to use the IMA

(Integrated Modular Avionics) standard in space missions.
- SAVOIR-FAIRE (Space AVionics Open Interface aRchitecture): working group focused in reference architectures for space systems.
- CCSDS: working group oriented to the development of standards for Communications and data Systems in space missions, and specifically SOIS (Spacecraft Onboard Interface Services).

After analyzing the state of the art, we decided to use the results of SAVOIR-FAIRE [6] and more specifically the proposed use of the SOIS standard as a reference [4]. The main reason behind this decision is that SOIS standard is thought to be an important part of the ESA reference architecture, in the future. It has the objective of improving the systems design and development process by defining generic services that will simplify the way flight software interacts with flight hardware, permitting interoperability and also reusability, which are also our main objectives.

3. PROPOSED ARCHITECTURE

Currently, there is not a commercial SOIS implementation available yet. However, the final version of the standard is about to be published and the existence of COTS providing the SOIS service in the near future is highly possible. While a full implementation of SOIS is out of our scope due to its complexity, it would be really positive to use of the ideas that make up the core of the standard in order to design a light-weight communication protocol similar to SOIS (especially in its layered approach and the interfaces provided to applications), easy to implement, to be used immediately in current projects and extensible in order to use it in future projects.

There are two basic ideas that have been extracted from SOIS and used as our reference, namely, the layered architecture of the protocol and the service oriented schema. The former is similar to other ones, such as PolyOrb [5], which distinguishes between application personality and protocol personality layers; SOIS is divided in three layers: application support, transfer (optional) and subnetwork. This stratification provides the isolation and flexibility needed to minimize the impact produced by a change at a given level in the other levels of the architecture. The later consist of the fact that every layer contains a number of service sets, i.e. Message Transfer Service, File Service, etc. In turn, every set provides a number of services, i.e. wallclock, file transfer, etc. The services range is wide enough to support the usual applications needs, nevertheless, it can be extended to support future needs.

Thank to these properties, the isolation among the different elements (application, drivers, RTOS…) of a processing node is maximized. Furthermore, it is an expandable structure that makes easier to add new features in the future. Both characteristics are crucial to ease the maintenance of an application because, when it comes to introduce a change, the affected code will be easily bound. Thus:
- The number of modules to modify is lower.
- Introducing new errors is not so easy.
- The impact in the subsequent V&V campaign is considerably reduced.

Figure 1 shows the lightweight protocol architecture, composed by three layers: the *Application Support* layer the *Subnetwork* layer and the *Configuration* layer. In general, the SOIS most sophisticated characteristics as auto-detection, advanced interaction with hardware devices or dynamic re-configuration have been dropped down in the lightweight implementation in order to simplify it as much as possible. The *Application Support* layer, which is the closest to the application, has been reduced to four simplified services, considered enough to satisfy the basic application needs at present and in future applications:
- Command and Data Acquisition Service: commanding and data acquisition by applications for hardware devices such as sensors and actuators independently of their locations. Through the *Device Access* Service provides basic reading from and writing to devices regardless of their location. *Device Data Pooling* and *Device Virtualization* have been discarded at this stage due to the complexity of their implementation.
- Message Transfer Service: for communication between applications using asynchronous, ad-hoc, discrete messaging with a bounded latency. It includes:
    - · Asynchronous Send/Receive: the sender continues its execution and does not wait for an ACK.
    - · Synchronous Query: A message is sent to the destination user in a synchronous manner and a corresponding reply message is received from it.
    - · Publish/Subscribe (static): a static table keep a list of message types publishers-subscribers; this way, the sender is able to disseminate information sets to several receivers with a single instruction (from the application point of view).
- File and Packet Store Service: reduced to just the services related to files, *File Transfer* and *File Management*, it is used by applications to, management (create, read, write, etc) and transference of files.
- Time Access Service: provides access for applications to the system time with known accuracy independent of their locations thanks to the *Wall Clock Capability*, which enables the application to read the time on demand. Also, given the fact that one of our projects is a GNSS receiver, it is quite feasible that a future extension contains the *Alarm* and *Metronome* features, really useful to provide functionalities such as *Pulse per Second*.

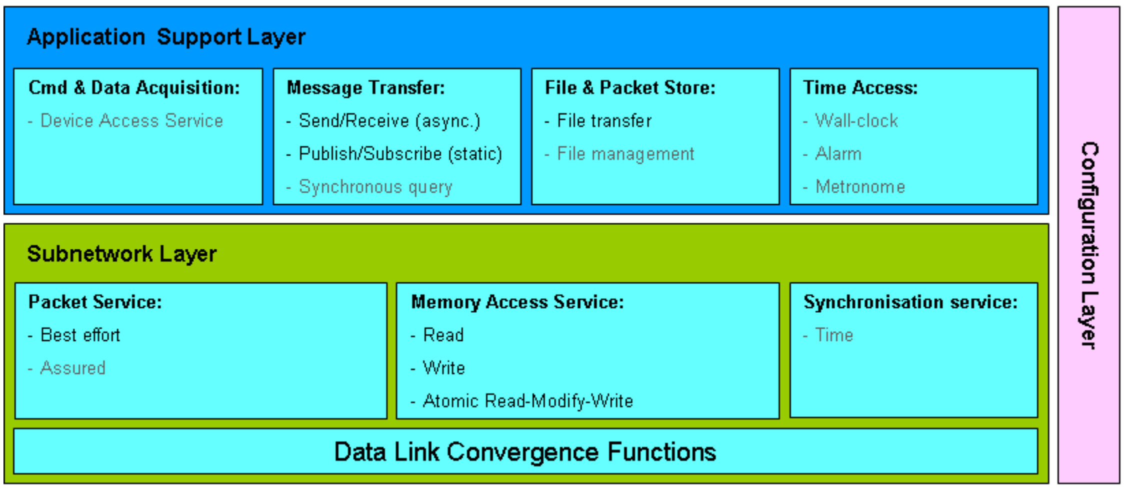

Figure 1: Communication protocol architecture

The *Application Support* layer is, in turn, supported by the S*ubnetwork* layer, which is composed by a set of services together with the *Data Link Convergence Layer*. The former deals with low-level details, such as, memory access, synchronization, packets management and so on, while the latter is responsible to adapt the details related to the physical interface (Ethernet, CAN, SpaceWire…). The subset of SOIS subnetwork services selected is the following ones:

- Packet Service: The SOIS Subnetwork Packet Service transfers Service Data Units, which comprise variable length, delimited octet strings, from one endpoint on a data link/subnetwork to another endpoint on the same data link/subnetwork, using the SOIS data link functions to move the information across it. The original four service classes have been reduced to two:
  - Best effort: provides for non-reserved, try once communication. It makes no promises about the time of delivery, the network bandwidth available, or the error rate of the traffic. Several priority levels are provided for Best Effort traffic. Traffic with a higher priority level is treated preferentially compared to traffic with a lower priority level.
  - Assured: provides for non-reserved communication with retries. It tries to ensure that the traffic arrives at the intended destination. If the data does not arrive safely at the destination then it is resent. To support this, the destination acknowledges the receipt of Assured traffic. Several priority levels are provided for Assured traffic, which are the same levels as those for Best Effort traffic.
- Memory Access Service: provides a means for a user entity to retrieve or change data located in memory hosted by a node on a data link-subnetwork, including:
  - Read: to retrieve the contents of memory from specific locations(s) in a specific memory resident at a specific subnetwork location.
  - Write: to change the contents of memory at specific location(s) in a specific memory resident at a specific subnetwork location.
  - Read/Modify/Write: to request the service of retrieving the contents of memory resident at a specific subnetwork location and to modify that data whilst blocking attempts by other entities to modify it, which is especially useful in multi-core architectures where a data may be accessed concurrently from several processing nodes.
- Synchronisation Service: The SOIS Subnetwork Synchronisation Service provides a means for a user entity to maintain knowledge of time which is common to all data systems on the subnetwork.

Finally, in order to make the implementation as simple as possible there is a transversal *Configuration Layer*. It has been defined as a container of meaningful information in order to perform the communication between parts. This information is used to create static communication links that shall be used by the user for different purposes. Of course it is very important that both sides of the communication agree a clear definition of all the information contained in this configuration layer. This configuration layer is defined, in principle, as static; nevertheless, the design of the architecture allows to provide some dynamism to this layer by offering an interface to the user to modify the contents.

4. IMPLEMENTATION

4.1. First Implementation Features

The first implementation of the communication protocol is intended to support the Monitoring & Control (M&C) module of a GNSS receiver and it is being carried out in an extensible way. The main goal of this initial approach is to offer the *Message Transfer Service* (MTS) to the M&C system through an IP network. This M&C system must support the sending/receiving of telecommands, telemetry and files. Of course not all the functionality available in the architecture is needed for this purpose so a reduced version of the architecture is being implemented containing only the Message Transfer Service and File Store in the Application Support Layer and, the Packet and Memory Access Service in the Subnetwork Layer.

When designing the architecture of the receiver and its communication services, several challenges arose:

- The TM/TC to the receiver has to be defined following a very flexible approach, given that the list of properties and functionalities of the receiver and the commands it can accept is not fully defined yet. In addition, while present commanding needs are small, future evolutions of the receiver can change the situation.
- The physical network access of the receiver is thought to change across the different possible uses of the receiver (eg. Ethernet, bus CAN, others).
- The resources available to the project are limited.

The method to overcome these challenges was to define a set of robust layers and interfaces but at the same time giving flexibility for its implementation in a progressive approach. This was achieved by defining the following elements of the proposed architecture:

- Message headers and encapsulation: the size, type and encoding of each element is clearly defined to allow the communication between layers of the same level on different systems. Each field has a specific objective, like defining the type of message or service being used. This structure allows adding as many future message types as needed.
- Layer services and information to be exchanged: the services offered in each layer are detailed as a set of interfaces to be used by upper layers or needed by lower layers. The properties of the parameters used in the interfaces are specified.

As a result of this flexibility, it has been possible to implement the architecture over two different platforms:

- Receiver M&C application in a resource limited CPU Soft-Core, using structured programming (C language) and send/receive style execution flow.
- Standard PC client, using object oriented and event driven programming (Java language).

The use of the configuration layer provides also to the system a good degree of flexibility at the same time it makes the implementation easier. For example it is foreseen to send different TM blocks to several destinations. This objective will be achieved just by editing properly the configuration layer. This makes that the user level code does not have to be changed by adding new sending calls to different destinations, but only accessing to the appropriate configuration services through its interface.

4.2. Difficulties to Overcome

As the architecture design matured, the complexity of managing multiple devices and message flows became apparent. For instance:

- Resource blocking is an outstanding issue: as communication through physical devices can imply blocking for an unknown lapse of time, it is necessary that the implementation allows using the rest of the data links even if one of them is currently busy. This is achieved by using message queues and threads to process them. Furthermore, the implementation of some services in the intermediate layers can lead to blocking too, as is the case of fragmented file transfer; and in this case it is important to carefully design the message flows to discover the choke points and use queues, threads or other mechanisms to avoid blocking. Figure 2 shows how this problem has been managed.
- In the case of a reliable (assured) connection, that requires establishing a dedicated channel of communication before starting to exchange information, it is important to note that one of the sides of the communication can remain blocked until the other side start the connection dialog. This could be an issue and must be considered in the implementation if the sides are expected to run independently.

4.3. Extensions

Figure 1 above showed in black font the services that have been included in the first implementation and in grey font those that are planned to be included in the near future. The first service to be added in the Application Support layer is the transmission of synchronous messages, where a message response is expected for each message sent. In order to avoid a blocking behaviour when using this service, we have added a transaction identifier in the message structure. By using this field it is possible to send a synchronous message on a non-blocking interface, and obtaining later the corresponding response received (or a notification if a timeout has expired). It is also our priority to add assured transmission of packets, which will be supported first of all by an Ethernet/TCP link. In the near future is also planned to support other links such as bus CAN or RS-422 as needed.

The proposed architecture is independent of the role played by the entities in the system, allowing master-slave or master-master communications. Thanks to this, the use of the protocol is not limited to communication between remote systems, and therefore future upgrades of the implementation will be used to access hardware specific modules or ease the interaction between multiple processor cores.

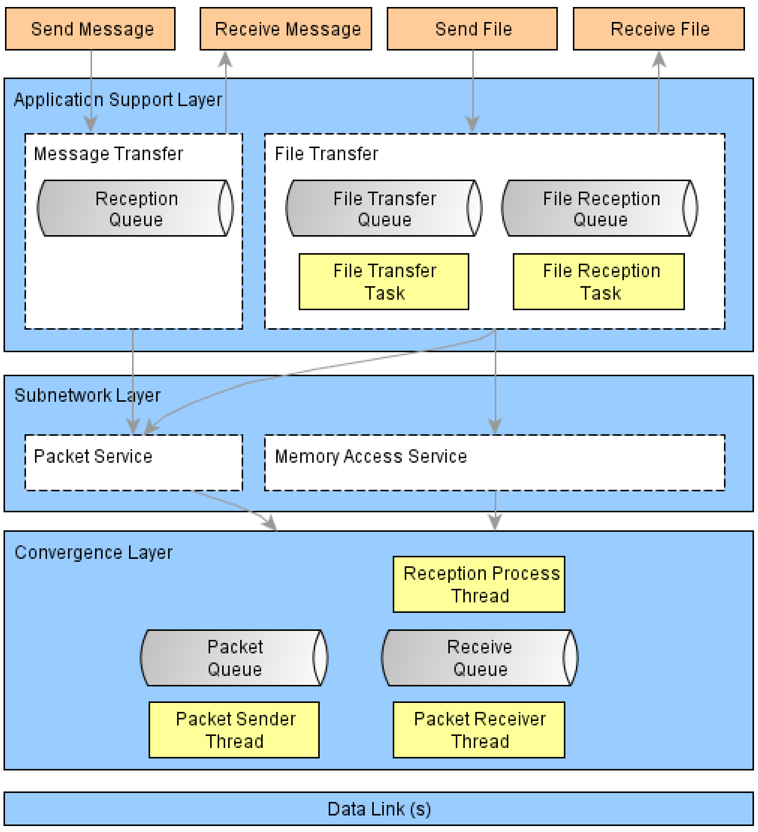

Figure 2*: Concurrency view of the implementation*

5. CONCLUSIONS

The component-based development techniques and reference architectures are key issues to reduce costs and development periods. The definition of a light-weight communication protocol, based on the recent results of European researches and standards, allows us to have at our disposal a flexible and extensible component to be used in present and future small and medium-sized projects, backed by the experience of domain experts, that will avoid to design *ad-hoc* protocols once and again, reducing costs and development periods. Moreover, following the line drawn by previous ESA-related projects will ease a future full-compliance with their requirements and standards in the development of applications / building-blocks, helping our company to succeed in establishing a position for itself among the providers of applications and building-blocks compliant to ESA needs for future projects.

REFERENCES

1. Terraillon, J.L., Jung, A., Arberet, P. (SAVOIR-FAIRE working group) et al (2010). Space On-Board Software Reference Architecture. In Proceedings of the *Data Systems in Aerospace, DASIA conference*. Budapest, Hungary.
2. de la Puente, J.A., Zamorano, J., Pulido, J.A. and Urueña, S. (2008). The ASSERT Virtual Machine: A Predictable Platform for Real-Time Systems. In Myung Jin Chung, Pradeep Misra (eds.), Proceedings of the 17th IFAC World Congress. IFAC-PapersOnLine.
3. Panunzio, M., Vardanega, T. (2009). On Component-Based Development and High-Integrity Real-Time Systems. In Proceedings of the *15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 09*. ISSN: 1533-2306, pages 79-84.
4. Consultative Committee for Space Data Systems (CCSDS). Spacecraft Onboard Interface Services. Green Book. Issue 1. June 2007. CCSDS 850.0-G-1.
5. Vergnaud. T., Hugues, J., Pautet, L. and Kordon, F. (2004) PolyORB: a schizophrenic middleware to build versatile reliable distributed applications. In Proceedings of the *9th International Conference on Reliable Software Techologies, Ada-Europe 2004 (RST'04)*, volume LNCS 3063, pages 106 - 119, Palma de Mallorca, Spain. Springer Verlag.
6. Savoir-Faire working group. Savoir-Faire Onboard software reference architecture. TECSWE/09-289/AJ