DisCo Space-Oriented Middleware: Architecture of a Distributed Runtime Environment for Complex Spacecraft On-board Applications

M. Prochazka[1], S. Fowell[1], L. Planche[2]

1: SciSys Ltd, Clothier Road, BS4 5SS Bristol, United Kingdom
2: EADS Astrium SAS, 31 rue des Cosmonautes, 31402 Toulouse cedex 4, France

Abstract: The DisCo project funded by the European Space Agency addresses future space missions with complex payload data handling applications. The key features are hard real-time constraints for applications running in embedded environment of a medium to large size space mission, partitioning between applications having different levels or criticality, and distributed computing. This paper presents an overview of the DisCo Space-Oriented Middleware (SOM) which provides support for DisCo applications running in distributed environment and having stringent requirements for predictability, fault detection and handling. The paper identifies key motivations for building DisCo SOM, provides a brief overview of the SOM architecture, compares it with other solutions and makes suggestions for future work.

Keywords: Spacecraft on-board software, middleware, temporal and spatial partitioning, Integrated Modular Avionics

1. Introduction

1.1 Motivation

Current spacecraft on-board systems become increasingly complex, providing more functionality such as coordination, cooperation, interoperability, autonomy, and improved fault tolerance. The costs to develop radiation-tolerant hardware components remain high so simply developing faster CPUs does not address the requirement for more functionality. Instead distributed and decentralised architectures are being addressed, based around high-speed networks such as SpaceWire. While at the same time, there are cost saving pressures – optimisation of mission efficiency, use of system resources, software development process.

Standard solutions exist for efficient use of system resources and the required improved functionality. But these are sporadically deployed on different missions with little direct re-use between them. By defining standard software architectures with common building blocks that can be reused between missions, e.g. COTS or third-party software, the software development process can be reduced while improving the functionality of the system.

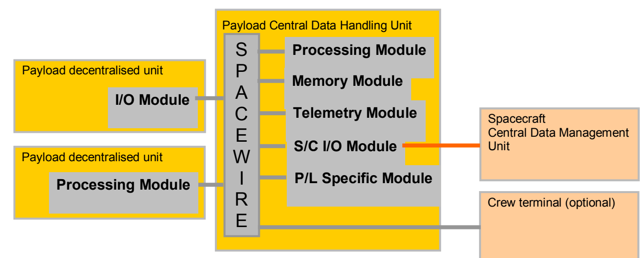

Figure 1: Distributed Scalable Data Handling and Processing Architecture

The key objective of DisCo was to provide a re-useable software infrastructure to address the increasing complexity of spacecraft on-board systems through the following key principles:

- Applications are distributed on a set of processing modules. They are developed independently but integrated onto the same distributed platform. Through the SpaceWire network the software running on any processing module can access any memory, telemetry, I/O or payload specific module.
- Predictability in both time and space domain is assessed as late as when applications are integrated.
- Detection of software and hardware failures and triggering of associated predefined recovery actions.
- Automatic fault recovery with low/high availability applications involving reconfigurations on multiple processing modules (PMs).
- Tailoring and runtime configuration allowing system budgets optimisation.
- High level of flexibility provided to the system integrator.

These objectives were met through the development of a Space-Oriented Middleware (SOM).

1.2 Use Case Applications

The SOM has been primarily developed for complex payload applications requiring distributed processing. Within the domain of the spacecraft control and data systems, the payload part is the one exhibiting the highest complexity and variability, usually solved through ad-hoc solutions that shall be redeveloped for each new mission. Besides, a general trend is towards augmentation of this complexity, including improvements in terms of coordination or even cooperation, while the space domain constraints limit the volume of hardware and software that can be embedded. SOM targets include multi-payload systems like the ones implemented for science exploration missions as well as individual payload systems with single or multiple CPU where:

- Multiple applications sharing a single CPU are guaranteed not to exceed their execution time, memory and communication budgets.
- Distribution is completely transparent and hence applications are developed the same way no matter whether they run on the same processor or a different one.
- Location of system units (processing modules, instrument front ends or mass memory units) is fully transparent which allows applications to be developed the same way, not taking account whether they run on the same processor or a different one.
- Different levels of availability at application software level are allowed and simultaneously supported.

1.3 Middleware Survey

Before doing any decisions with respect to the SOM architecture and design, we wanted to gather information about existing technologies, standards and studies which could be used, adapted, or implemented from scratch (“adopt, adapt, innovate”). In the Middleware Technology Survey we have first of all provided a definition of middleware used throughout the study: *a software layer that stands between the underlying operating system and applications and primarily facilitates the distribution of applications and services*. Next, we have identified various groups of relevant technologies to look at:

1. General Computing Technologies

- Standard Middleware Technologies (Distributed Computing Environment (DCE), Open Distributed Processing (ODP), Ada Distributed Systems Annex, Component Object Model (COM / DCOM / .NET), Common Object Request Broker Architecture (CORBA), Java 2 Platform)
- Component Technologies (Fractal, SOFA)
- Technologies for Parallel and Distributed Computing (Open Multi Processing (OpenMP), Parallel Virtual Machine (PVM), Message Passing Interface (MPI), OSE Direct Message Passing)

2. Real-Time Operating Systems (POSIX RT, RTEMS, VDK)

3. Technologies particular for Avionics Applications

- Packet Utilization Standard (PUS)
- Spacecraft On-Board Interface Services (SOIS)
- Integrated Modular Avionics (IMA – in particular the ARINC 653, 629 and 659 standards)
- SpaceWire
- Mass Memory Based Architectures (GAMMA and Magus)
- UNIONICS
- Generic Upgradeable Architecture for Real-time Dependable Systems (GUARDS)
- Advanced Avionics Architecture and Modules (A3M)
- Remote Agent Management Architecture (RAMA)
- Real-time Embedded CORBA over SpaceWire (RECS)

	Remote Invocation Service	Event Service	Mass Memory Service	Time Access Service	Device Access Service	Scheduling Service	FDIR Service	Configuration and Deployment Service	Information Flow Authorisation Service	Monitoring and Logging Service	Communication Service
CORBA	■	■	□	□	□	■	□	□	□	▒	□
SOIS	□	□	□	■	■	▒	□	□	□	□	■
Mass Memory	□	□	■	□	□	□	□	□	□	□	■
UNIONICS	□	□	□	□	□	□	▒	▒	□	□	▒
GUARDS	□	□	□	□	□	□	▒	□	▒	□	□
A3M	□	□	□	□	□	□	□	□	□	□	■
RAMA/RECS	■	□	□	□	□	■	□	▒	□	□	■
Bespoke	□	□	□	□	□	□	▒	▒	▒	□	□

■ Full adoption
▒ Partial adoption/augmentation
□ Non

Figure 2: Technology Selection for DisCo SOM

In the next step we have considered the requirements specification and architectural decision to come with the following proposal as to which technologies would be used:

2. SOM Architecture

This section provides an overview of the SOM architecture.

2.1 Services

The SOM provides a set of *functional services* as well as a set of *non-functional services (Figure 2)*. Functional services are used directly by the applications (i.e. they provide a functional interface used by applications). On the other hand, non-functional services are used only indirectly, i.e. by the SOM itself, so that they transparently contribute to functionality provided by functional services, or they change non-functional properties or quality-of-service parameters (i.e. they are used or configured through *management interfaces*). The functional services are as follows:

Remote Invocation Service (RIS): This is a key service which allows application modules to perform remote invocation of functions provided by another module (either in the same or within another application). RIS provides distribution transparency, hence application modules are written the same way, not having to take into account where other applications are deployed. RIS is based on Minimum CORBA and Real-Time CORBA [1, 2, 3].

Event Service (EVS): This service provides support for producer/consumer data exchange ("publish-subscribe") and it is based on the CORBA Lightweight Event Service [4]. EVS is designed to use different types of event dispatchers for different event channels, depending on application requirements (efficiency vs. predictability). The implementation uses the CORBA Asynchronous Method Invocation (AMI) which is supported by the RIS and which makes it possible to call a method in an asynchronous fashion without modifying the servant interface or implementation. This gives EVS the ability to deliver an event asynchronously to multiple consumers without having to use extra dispatcher threads and thus adding unnecessary overhead.

Mass Memory Service (MMS): Access to persistent virtual mass storage is one of the key features required by on-board applications. MMS provides a file-oriented API and location, distribution and communication transparency. MMS was reused from the MAGUS Mass Memory Study [5].

Device Access Service (DAS): This service allows applications to perform commanding and data acquisition to simple networked devices (e.g. simple payloads), in this case over SpaceWire though extensible to others. DAS is an implementation of the SOIS Device Access Service and Device Data Pooling Service [6, 7]. It provides location and communication protocol transparency.

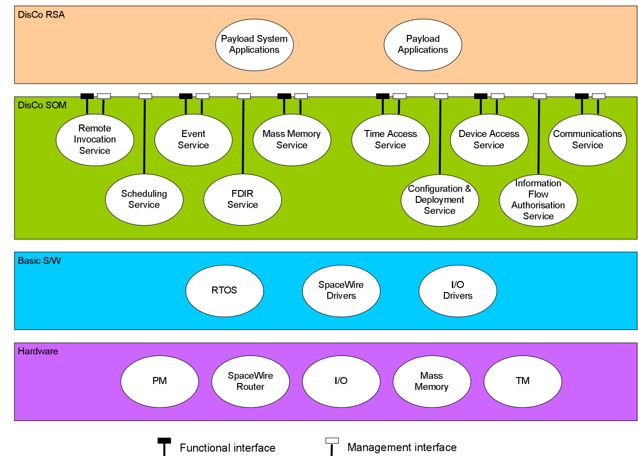

Figure 2: SOM Architecture

Time Access Service (TAS): TAS provides applications with access to correlated on-board time, providing transparency from the correlation protocol. It is an implementation of the SOIS Time Access Service [8].

The non-functional services are as follows:

Scheduling Service (SCS): This service base on the CORBA Scheduling Service allows to define execution eligibility criteria and global-to-local priority mapping [3].

Failure Detection Isolation and Recovery Service (FDIR): This service provides a distributed framework for mapping failure detection and identification to a set of recovery policies and mechanisms appropriate to the corresponding SOM services, as well as fault notification in the FDIR hierarchy.

Configuration and Deployment Service (CDS): CDS provides distributed management of application deployment, lifecycle, configuration, and upgrade across the processing modules as a result of mission phase changes and fault recovery. A simple application framework was designed which all applications implement in order to let the SOM control their lifecycle (start, activate, deactivate, stop, checkpoint). CDS also provides a software repository to which upgrades of existing applications as well as new applications could be uploaded, and which is used to load appropriate applications during a configuration update.

Information Flow Authorisation Service (IFAS): The SOM must ensure that application integrity is

preserved while operating on shared resources. IFAS provides an extensible framework for implementing algorithms for the authorisation of information flows between applications, devices and mass memory (in conjunction with RTOS). The authorisation criteria implemented by an algorithm can vary from a simple credentials-based authorisation to more complex rules for communication between different criticality levels as used in the GUARDS project [9].

Communications Service (CMS): CMS is an implementation of a proposed SpaceWire mapping of the SOIS Subnetworking Layer's Packet Service [10]. It provides resolution of abstract location to concrete network address and both best effort and assured SOIS classes of service. In addition it provides network integrity fault detection (i.e. monitoring of bandwidth usage against allocation).

2.2 Support for Partitioning

DisCo allows different applications, potentially with different criticality levels and potentially developed by different software vendors, to be executing on the same processing module. The SOM treats the applications as fault containment regions and provides means for partitioning between them:

- **Spatial partitioning**: Memory pooling techniques and FDIR policies to deal with illegal memory access were designed. However, due to RTEMS not using the LEON2 memory write protection mechanisms, spatial partitioning is not supported by the SOM implementation for the DisCo demonstrator.
- **Temporal partitioning**: RTEMS was extended with CPU-Time timers which allow the SOM to enforce CPU budgets of individual application tasks. An associated CPU Budget Overrun FDIR policy is implemented.
- **Network partitioning**: CMS allows to associate applications with communication budgets defined as amount of data (packets or bytes) allowed to be sent per a time window. An associated CMS Budget Overrun FDIR policy is implemented.
- **Information flow checking**: IFAS makes it possible to define a data filter which prevents faulty or unexpected data to be sent from one application to another.

2.3 End-to-End Schedulability Analysis

A DisCo system is composed of two parts: the SOM and applications. To perform schedulability analysis of DisCo, we have to take into account that the SOM only gives a framework to the software system, but schedulability will differ for different DisCo deployments containing particular applications. In order to make the DisCo system analysable, a computational model has been specified which defines abstract system entities representing computations, information exchanges between computations (communication), and their temporal and concurrency properties. To consider remote invocations, communication costs must be added to the worst case execution time computed. The delays for messages being sent between processors and the overheads due to communications must be bounded. In the holistic schedulability approach, communications schedulability is integrated with the processor schedulability so that execution time of operations which use communication includes overheads due to packet handling and communications protocol [11]. As a necessary prerequisite, a timing analysis of the communications layer must be performed. In particular for DisCo, this means:

- Timing analysis of a single SpaceWire packet delivery. This includes considerations with respect to the network topology, overheads due to use of SpaceWire routers and packet processing by the SpaceWire drivers.
- Timing analysis of a CMS packet delivery. CMS adds overheads due to segmentation, buffer management and acknowledgment reply messages.
- Timing analysis of a RIS remote invocation. On top of SpaceWire and CMS, the RIS adds its own protocols used to perform remote method invocation.

We use the notion of an *end-to-end task* to denote the system where a task may execute on several processors before it completes. An end-to-end task could be viewed as a chain of subtasks. Each subtask is then a continuous execution thread of the task running on a single processor. In other words, these subtasks represent tasks as treated in the single-processor schedulability analysis presented above. In DisCo we are interested in meeting all end-to-end deadlines of end-to-end tasks, that is, deadlines which take into account all subtasks in the end-to-end task chain.

4. Evaluation

DisCo provides software infrastructure for distributed data processing and control in spacecraft on-board applications (Figure 3). An early evaluation shows that the SOM addresses many issues of current complex spacecraft on-board applications dealing with multiple processing, memory, I/O and payload modules, and having stringent requirements for predictability, fault detection and handling.

3.1 Benefits of DisCo

The DisCo SOM is a complex middleware which combines both well know mature technologies applied in the space domain with emerging and state-of-the-art technologies:

- Several mature and standard technologies were adapted to space systems constraints (RIS, EVS, SCS).
- Several standards from the space domain were adopted and implemented (DAS, TAS, CMS).
- In addition, some technologies were newly developed (CDS, IFAS).

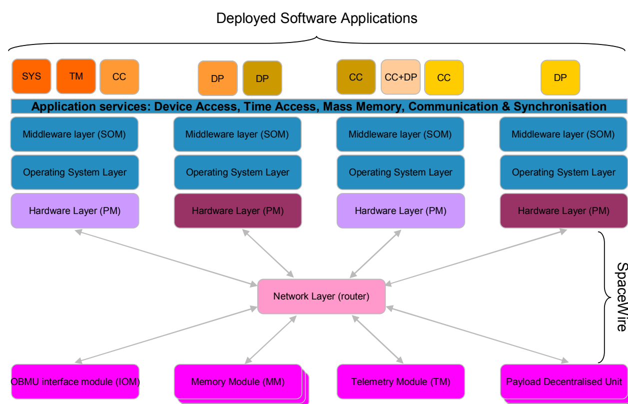

Figure 3: DisCo High-Level Physical Architecture

The SOM provides a rich set of services which meet requirements of complex on-board applications and stringent hard real-time and dependability constraints.

The SOM services are tailorable so that only required functions are deployed. DisCo applications are composed of smaller components which gives more flexibility in terms of configuration, deployment, CPU and memory load. They are focussed on the provision of their functionality not on how the functionality is invoked, nor on how fault-tolerance is achieved. Greater flexibility and component reuse also provides opportunities for on-board software suppliers to invest in validated products with costs covered across multiple deployments.

3.2 Space-Oriented IMA

Integrated Modular Avionics (IMA) is a concept of aircraft systems hosting multiple software modules certified to different criticality levels. The IMA features are layered architecture using standard programming interface layers to hide hardware and applications from one another, reconfiguration of applications (statically or in flight), protection mechanisms to allow resources like memory to be shared by multiple criticality level applications, and to allow applications to be inserted/altered without impact on the rest of the system (partitioning), flexible scheduling to meet deadlines of all the applications for each permitted configuration and when system is upgraded, code re-use and portability, physical integration of networks, modules and IO devices and design for growth and change.

DisCo addresses most of these IMA features by:

- Systematic building of software applications composed of well-specified building blocks – software modules. The SOM application framework is being extended in the COrDeT study funded by ESA [12]. The component framework facilitates software reuse, incremental design, static and dynamic reconfiguration.
- Integration of different communication protocols, networks and devices (CCSDS SOIS).
- Static and runtime mechanisms to enable secure sharing of resources amongst applications with different criticality levels (I/O, CPU, memory).

IMA has been used in Airbus A380, Boeing 777 and 787 Dreamliner, as well as in several military aircrafts. The goals of IMA for spacecraft on-board software are similar to those in avionics systems:

- Weight and energy savings: Boeing reports that its 787 Dreamliner saves 2000 pounds of weight thanks to IMA. Airbus reports that A380 saves a half of the processing units thanks to IMA [13].
- Savings in software engineering: Focus could be directed on application layer, instead of hardware, communication and systems integration.
- Software reuse and portability: Standardised integration of software components made by different vendors.
- Advanced features: IMA gives opportunity to new features such as flexible scheduling and support for static and dynamic reconfiguration.

The DisCo/COrDeT approach is similar to the Automotive Open System Architecture (AUTOSAR), with the Virtual Functional Bus providing functionality similar to the SOM's RIS, standardised interaction between components, standardised component implementations and well-defined functional interfaces (provided, required) [14].

DisCo has investigated a set of technologies capable of supporting the increasing complexity of the payload systems. A key challenge was to address the software engineering issue and particularly the verification and validation process (which is also central to the IMA approach).

4. Future Enhancements

In this section we analyse which way the SOM could be enhanced in the future.

4.1 Functionality

There are various functionality enhancements we are considering for the future:

Application framework: One of the key enhancements is to improve the SOM application framework. The control of the software module lifecycle through the use of module-specific interfaces could be refined through the adoption of a component framework. This also allows for the definition and binding of provided and required interfaces, necessary to determine that a system is consistent and complete, i.e. all the interfaces required by all the software modules/components are present in the system. Design of a component model for the SOM is already being considered by the COrDeT project. A more sophisticated model should allow to check component implementation not only based on their functional interface, but also based on their behaviour, or semantics of functions they provide, and their non-functional properties.

Dynamic linking: The CDS Deployment Manager is in charge of PM bootloading, downloading an image for the processing module to execute, and also module loading. The CDS Software Repository allows for uploading and downloading applications composed of software modules, version upgrade, etc. However, this functionality has not been fully implemented due to unavailability of a dynamic linking technology which could be easily adopted on the RTEMS platform. At present all software modules must be present in the current PM image, although they might be unused or inactive in the current configuration, and being used after a configuration update. Both Software Repository and software module loading are only simulated. Both the SOM design and all the APIs support configuration updates at the granularity of software modules. Various options for dynamic linking have been already studied and safe and efficient dynamic linking could be added to CDS in the near future.

Plug-and-Play: The CCSDS SOIS area is currently defining a concept for Device Plug-and-Play to allow for dynamic discovery and reconfiguration of new devices inserted (mechanically, power-up or by other activation methods) into a spacecraft on-board bus/network. The SOM is implemented assuming a system application configures the SOM with the available devices and the SpaceWire network (following the current standard practise in on-board software). If there are any discrepancies between configuration and reality, they may be detected by the FDIR Fault Detectors and reported to the registered System FDIR application. Insertion of new devices must be controlled by the system application authorised to re-configure the SOM. By implementing the SOIS Device Plug-and-Play services, the device and network configuration knowledge that must be encoded in the system application is removed, removing the risk of a discrepancy between configuration and reality.

Task group CPU budgets: The CPU-Time Timers extension to RTEMS supports a CPU-Time timer associated with a single thread. This does not allow for the tracking of CPU utilisation of a group of related threads which constitute an application. To allow the assignment of a CPU budget for an application instance on a particular processing module, the RTEMS extension should be modified so as to be applied to a group of RTEMS threads.

Spatial partitioning: The upgrade to RTEMS to make use of the LEON2 memory "write" windowing mechanism should be made, so as to be used to provide spatial partitioning.

4.2 Performance

When analysing the performance figures of the SOM product, a number of factors must be taken into account:

- **Emulated Memory Arrays, devices and partially emulated SpaceWire network**: It must first be considered that a number of elements of the DisCo Demonstrator are emulated using a Linux PC attached to the actual SpaceWire network, in particular the Memory Arrays (MAs), Devices and also a portion of the SpaceWire network (a Virtual SpaceWire Router running as an application, SpaceWire links between Virtual SpaceWire Router and Memory Array Emulation emulated using UDP/IP – this allows multiple SpaceWire nodes to be emulated, each as a Linux applications). All of these must be considered to operate at orders of magnitude slower than corresponding hardware implementations. A more reasonable emulation would run the Memory Array and Device Emulation on individual LEON2-based PMs (PM-Leons), blocking directly on the SpaceWire driver and with the PM-Leon directly connected to a SpaceWire port of the USB-SpaceWire Router, eliminating at least the overhead of the emulated part of the SpaceWire network.
- **Tooling support for optimisations**: Optimisations of RIS performed were based on data generated using the GNU *gprof* tool when running SOM tests under Linux. Unfortunately it was not possible to gather timing information, only number of invocations. Thus it was not possible to objectively determine where optimisations would be most efficient, that is to

say to identify the functions which use CPU the most (number of their invocations multiplied by their average execution time). Instead analysis of the most frequently invoked functions highlighted a number of functions that were identified as being "likely" to affect performance, e.g. acquiring and releasing of a mutex. Availability of tools to obtain execution times as well as number of invocations would allow for more effective optimisations to take place.

- **Limited refinement**: Due to schedule and resource constraints, only limited amount of time has been spent on analysing performance issues. Only the client side of RIS remote invocations were optimised (see Tooling Support for Optimisations). Plenty of scope remains for optimisations within the rest of the SOM.

During the DisCo "Refinement of the middleware to the level of a product" phase, a number of optimisations were identified and implemented for the Remote Invocation Service. Beyond those already implemented, the following list identifies further optimisations that may be performed:

- **Hardware support for CRC generation and checking**: The SpaceWire checksum algorithms are computationally intensive. Analysis of the SOM implementation has shown that in scenarios where a number of packets are being generated or received, considerable time is spent generating CRC checksums. Hardware support for the generation and checking of CRCs would considerably reduce the overhead imposed by the SOM.
- **Reduction of blocking operations per RIS invocation**: The SOM uses a number of dynamically allocated objects. To avoid heap fragmentation, the classic technique of memory pools is used. Each memory pool is protected by an exclusive lock, so as to avoid interference between different threads allocating from and freeing objects to the memory pool. The RIS in particular has a large number of objects that are allocated and freed on a per-invocation basis, resulting in many (tens) of exclusive locks being acquired and released on each invocation. This clearly provides a significant overhead. By more detailed consideration of the scope of the objects that are acquired and released per invocation and their reuse between invocations, as well as by adopting non-blocking data structures the number of exclusive locks acquired and released can be reduced. Some of this has already taken place during the SOM refinement, however there is still scope for further improvements.
- **RIS invocation request dispatching optimisation**: RIS request processing could be optimised by merging some of the task involved in message dispatching. This would imply fewer threads involved in message dispatching and hence less context switches performed per a single message dispatch.
- **Local RIS invocation optimisations**: The SOM product refinement was focussed on optimising remote RIS invocation optimisations. Depending upon application granularity and deployment, the majority of RIS invocations may turn out to be local. Further optimisations reducing unnecessary marshalling of CORBA headers could be performed.
- **CORBA message optimisations**: The CORBA request message could be reduced by simplifying the message header from current 70-100 bytes to approximately 20 bytes, offering a considerable saving in network overhead. The impact of this is that the RIS would no longer implement the CORBA standard and would not be able to interoperate with any other CORBA implementation.
- **Marshalling and unmarshalling**: It has also been considered that the CORBA data representation encoding mechanism, the Common Data Representation (CDR), also imposes an overhead of the network. The notion of *marshalling* and *unmarshalling* data into messages is fundamental to CORBA. Each data structure is walked through marshalling or unmarshalling each field in turn, rather than directly memory copying the whole data structure into the CORBA message. Thus marshalling and unmarshalling impose an overhead. However, CDR uses a mechanism known as *reader-makes-good*, that is to say the RIS invocation requester marshals the operation request into the CORBA request message in its local representation together with a flag indicating whether big endian or little endian was used. If the RIS invocation receiver uses the same data representation, no translation is required, i.e. byte swapping, and the data is simply unmarshalled. On the other hand, if it is different each field must be translated. However, it can be seen that in the majority of real circumstances within an on-board system, the CPU of the requester and receiver's PM would be the same type and so no translation would be required.

4.3 Process

The SOM offers technology allowing different applications to be developed separately and integrated later in the development lifecycle. It is crucial to define a development process taking into account the applications' ability to be easily

integrated, allowing for smooth transition from independent validation to integrated validation. This should involve developing a process for performing separate timing and memory usage analysis of applications and the SOM platform.

In addition, modelling techniques should be either developed or adopted to model software applications, modules, functional interfaces and bindings between them.

5. Conclusion

The paper has presented the architecture of the DisCo Space-Oriented Middleware. An early evaluation shows that the SOM addresses many issues of current complex spacecraft on-board applications dealing with multiple processing, memory, I/O and payload modules, and having stringent requirements for predictability, fault detection and handling. The paper has also analysed possible future enhancements of the SOM, its functionality, performance and also corresponding development process.

6. Acknowledgements

The authors would like to thank the DisCo SOM development team, Patricia Lopez-Cueva, Matthew McBraida, Julian Milby and Nick Clark (all SciSys), as well as Yong-Dan Shi (EADS Astrium SAS) leading development of the reference space application.

Many useful comments were provided by Jean-Charles Fabre and Yves Deswarte (LAAS-CNRS), Christophe Honvault (EADS Astrium SAS), Andy Wellings and Neil Audsley (University of York).

Last but not least, the authors would like to thank the ESA Technical Officer for DisCo, Philippe Chevalley.

7. References

[1] The Object Management Group, "*The Common Object Request Broker Specification*", Version 3.0.3, formal/04-03-01, 2004.

[2] The Object Management Group, "*Minimum CORBA*", Version 1.0, formal/02-08-01, 2002.

[3] The Object Management Group, "*Real-Time CORBA Specification (static scheduling)*", Version 1.1, formal/02-08-02, 2002.

[4] The Object Management Group, "*Lightweight Services Specification*", Version 1.0, formal/04-10-01, 2004.

[5] "*Data Management Software for Mass Memory Based Payload Processors: Prototype Design*", MMA-TN-SSL-DC-0004, Issue 1.1, 2005.

[6] "*CCSDS Spacecraft Onboard Interface Services – Device Access Service Red Book*", CCSDS 871.0-R-1, 2007.

[7] "*CCSDS Spacecraft Onboard Interface Services – Device Data Pooling Service Red Book*", CCSDS 871.1-R-1, 2007.

[8] "*CCSDS Spacecraft Onboard Interface Services – Time Access Service Red Book*", CCSDS 872.0-R-1, 2007.

[9] D. Powell (editor), "*A Generic Fault-Tolerant Architecture for Real-Time Dependable Systems*", Kluwer Academic Publishers, 2001.

[10] "*CCSDS Spacecraft Onboard Interface Services – Subnetwork Packet Service*", CCSDS 851.0-R-1, 2007.

[11] Ken Tindell, John Clark: "*Holistic Schedulability Analysis for Distributed Hard Real-Time Systems*", Microprocessing and Microprogramming, Vol. 40, Special Issue on Parallel Embedded Real-Time Systems, pp. 117–134, 1994.

[12] Component-Oriented Development Techniques, ESA AO/1-5237/06/NL/JD, 2007-2008.

[13] James W. Ramsey: "*Integrated Modular Avionics: Less is More*", Aviation Today's Avionics Magazine, http://www.aviationtoday.com/av/categories/commercial/8420.html, 2007.

[14] Automotive Open System Architecture (AUTOSAR), http://www.autosar.org.

8. Glossary

API: Application Programming Interface

CCSDS: Consultative Committee for Space Data Systems

CDS: SOM Configuration and Deployment Service

CMS: SOM Communication Service

CORBA: Common Object Request Broker Architecture

DAS: SOM Device Access Service

ESA: European Space Agency

EVS: SOM Event Service

FDIR: SOM FDIR Service or Fault Detection Identification and Recovery

IFAS: SOM Information Flow Authorisation Service

IMA: Integrated Modular Avionics

MMS: SOM Mass Memory Service

PM: Processing Module

RIS: SOM Remote Invocation Service

SOM: Space Oriented Middleware

SOIS: Spacecraft Onboard Interface Services

SCS: SOM Scheduling Service

TAS: SOM Time Access Service