

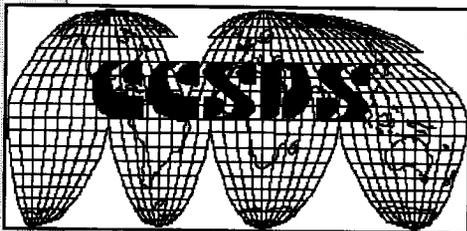
JPL

Message Transfer Service for Space Applications

Peter Shames

NASA/JPL

Mgr, JPL Information System Standards Program



18 June 2002



Acknowledgement

- Concepts developed within CCSDS Panel 1K, Spacecraft Onboard Interfaces (SOIF)
- Message Transfer Service developed by Messaging Special Interest Group

Peter Shames, NASA/JPL, Chair

Nick Achilleos, BNSC/Logica

Ewan Carr, BNSC/Logica

Laverne Hall, NASA/JPL

Stuart Fowell, BNSC/SciSys

Ed Greville, NASA/GSFC

Chialan Hwang, NASA/JPL

Gavin Kenny, BNSC/Logica

Norm Lamarra, NASA/JPL

Gary Lay, BNSC/Logica

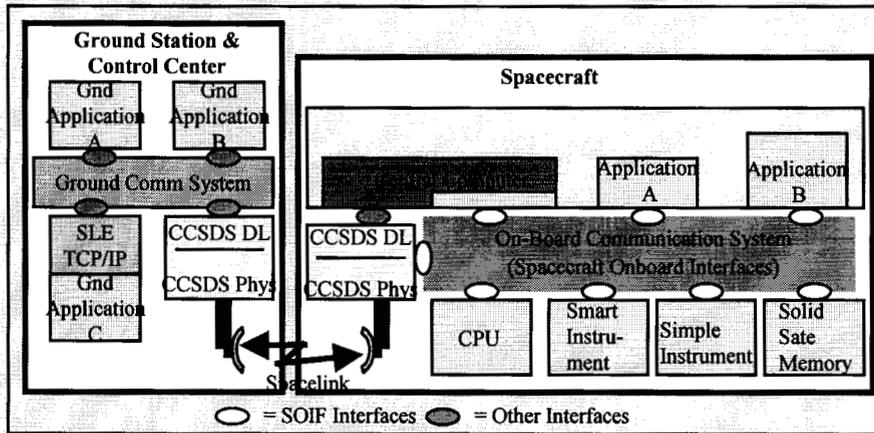
Stuart Mills, ESA/U of Dundee

Nick Rouquette, NASA/JPL

Joe Smith, NASA/JPL

SOIF Background & Context

The Context of SOIF: Flight and Ground



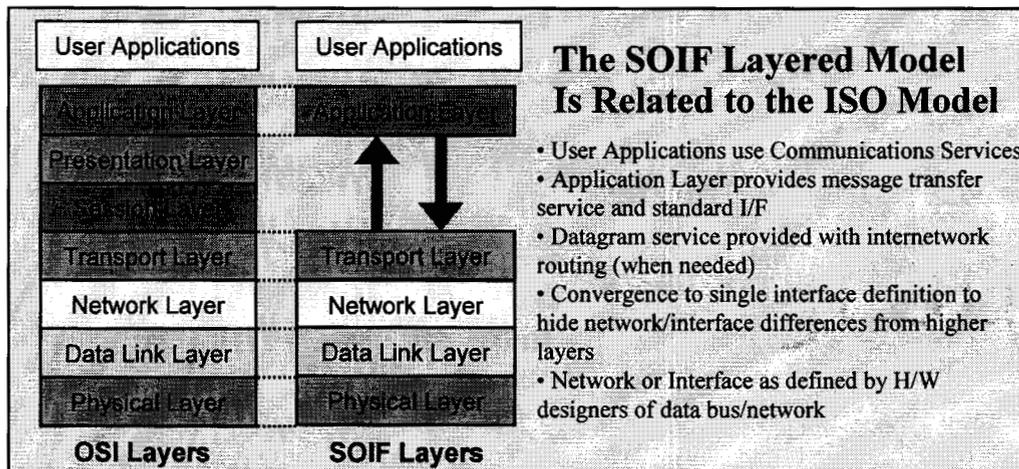
- The context of the Spacecraft Onboard Interface (SOIF) is related to the communications bus (network) onboard the spacecraft
- Does not necessarily include ground, spacelink, or special onboard interfaces
- Scope includes all types of spacecraft, from micro and small to large and/or manned, from LEO to deep space, because a bit is a bit
- Scope also includes all types of devices from large Instruments to small engineering sensors like a temperature sensor

SOIF Reduces Impact on Applications

- When communications standards are used, then the application can use the services provided by the protocol
- Without communications standards, then the applications either need to supply the service themselves, or do without the service
- Without standards, when the underlying bus changes, then the effects will ripple up into the applications

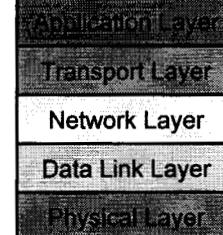
The SOIF Layered Model Is Related to the ISO Model

- User Applications use Communications Services
- Application Layer provides message transfer service and standard I/F
- Datagram service provided with internetwork routing (when needed)
- Convergence to single interface definition to hide network/interface differences from higher layers
- Network or Interface as defined by H/W designers of data bus/network



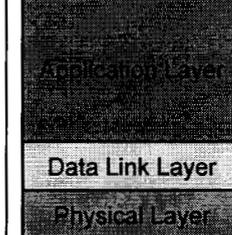
With standards, applications can use the standard services provided by the stack

User Applications



Without standards, applications must provide these standard services themselves

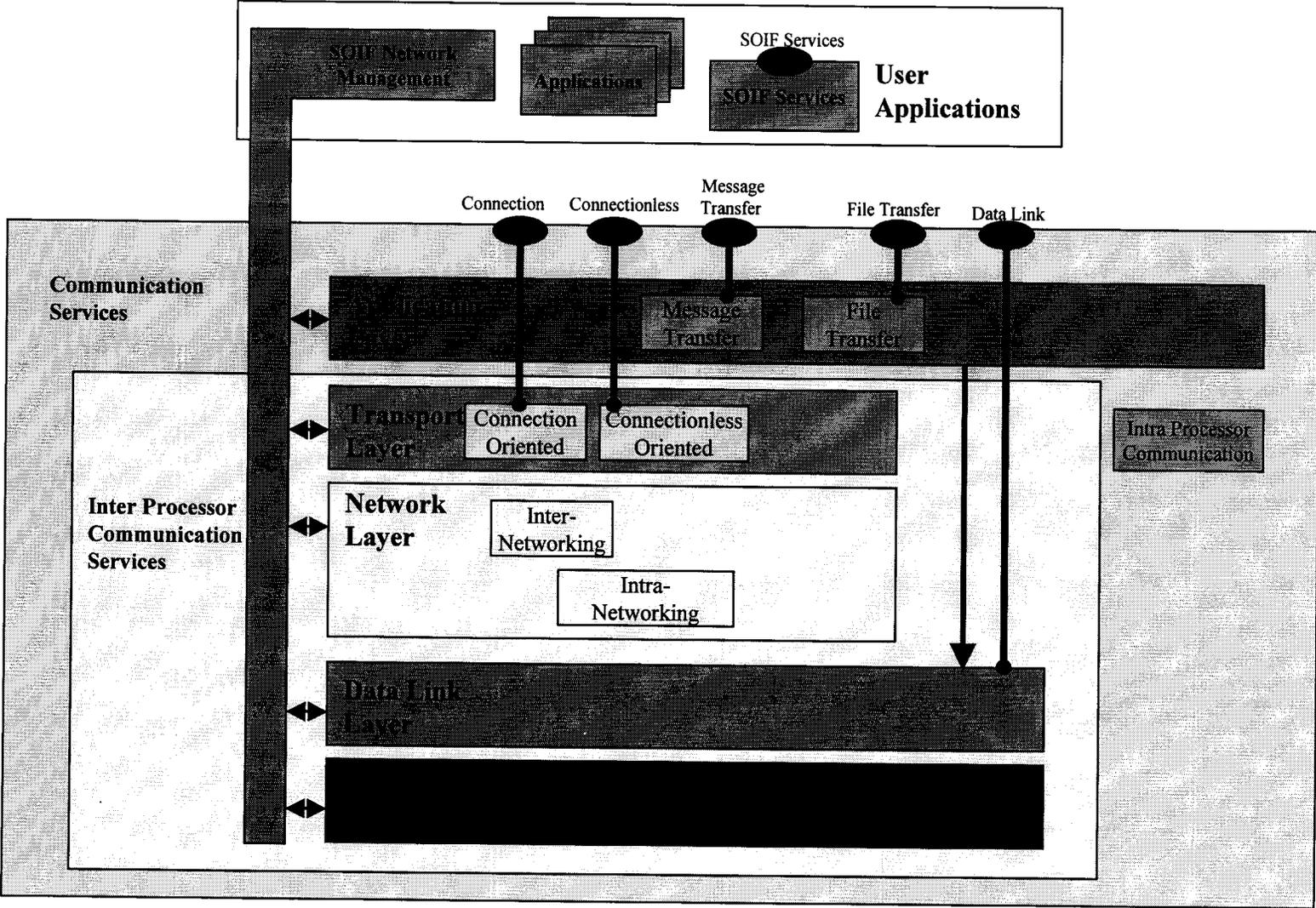
User Applications



SOIF Objectives & Goals

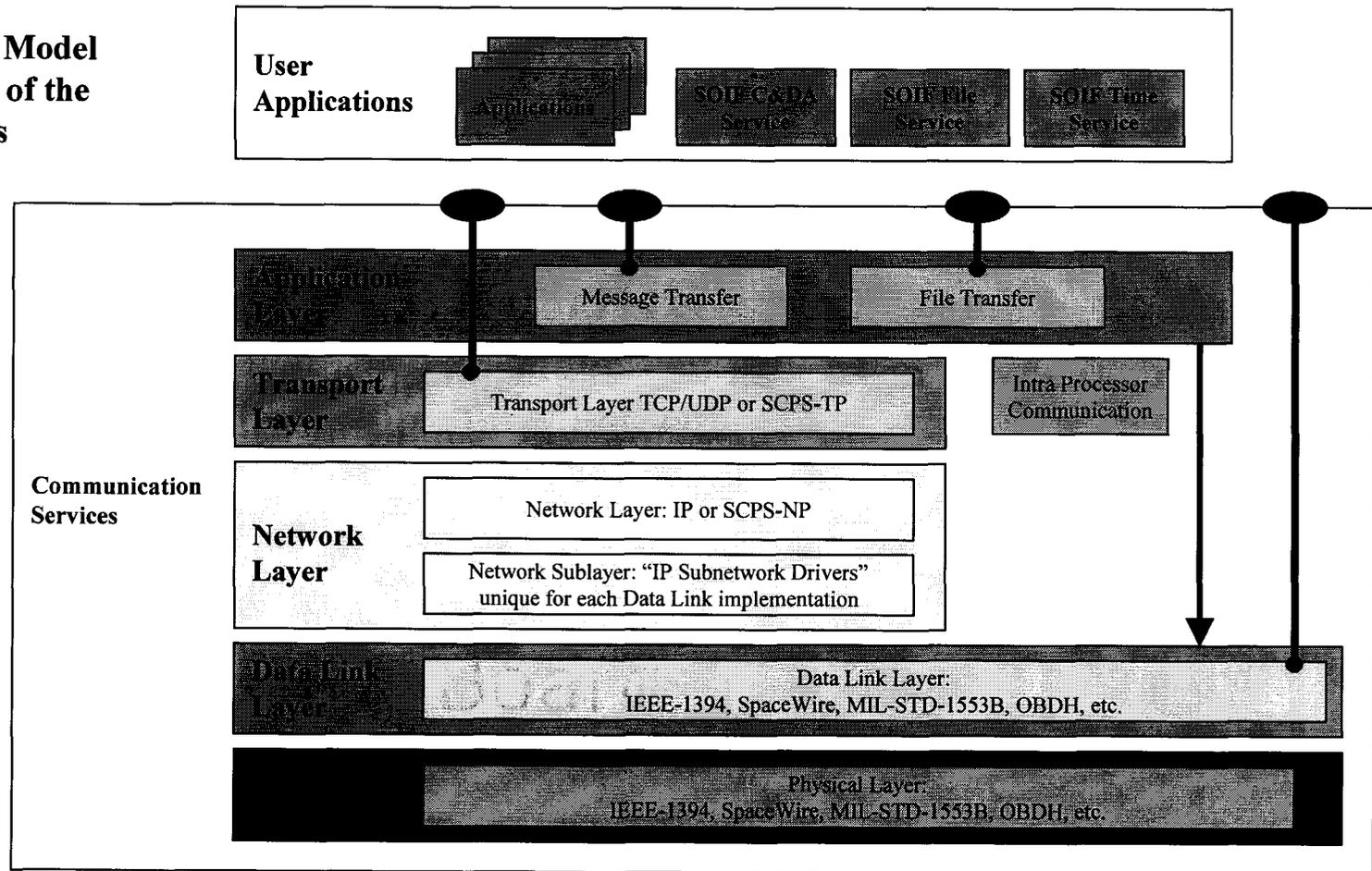
- Develop communications services for User Applications
 - That will meet the needs of spaceflight systems
 - Without excessive overhead or resources
 - Have standard software I/Fs
- Provide a selection of protocols that make sense for spacecraft
 - TCP/IP, with selected RFCs
 - SCPS, where required, i.e. over longer space links
 - Low overhead links as required
- Support changing the underlying data bus to meet the needs of the application
 - High Speed bus for science needs
 - Medium and/or Low Speed for command & control
 - Low Speed for sensor bus & non-critical applications
 - Can interface to wireless and long haul telecommunications links as required
- Will also support the seamless communications between elements in different nearby spacecraft, such as for constellations, formation flyers, and co-operating spacecraft

SOIF Reference Model



Spacecraft On-Board Interface (SOIF) Implementation Model

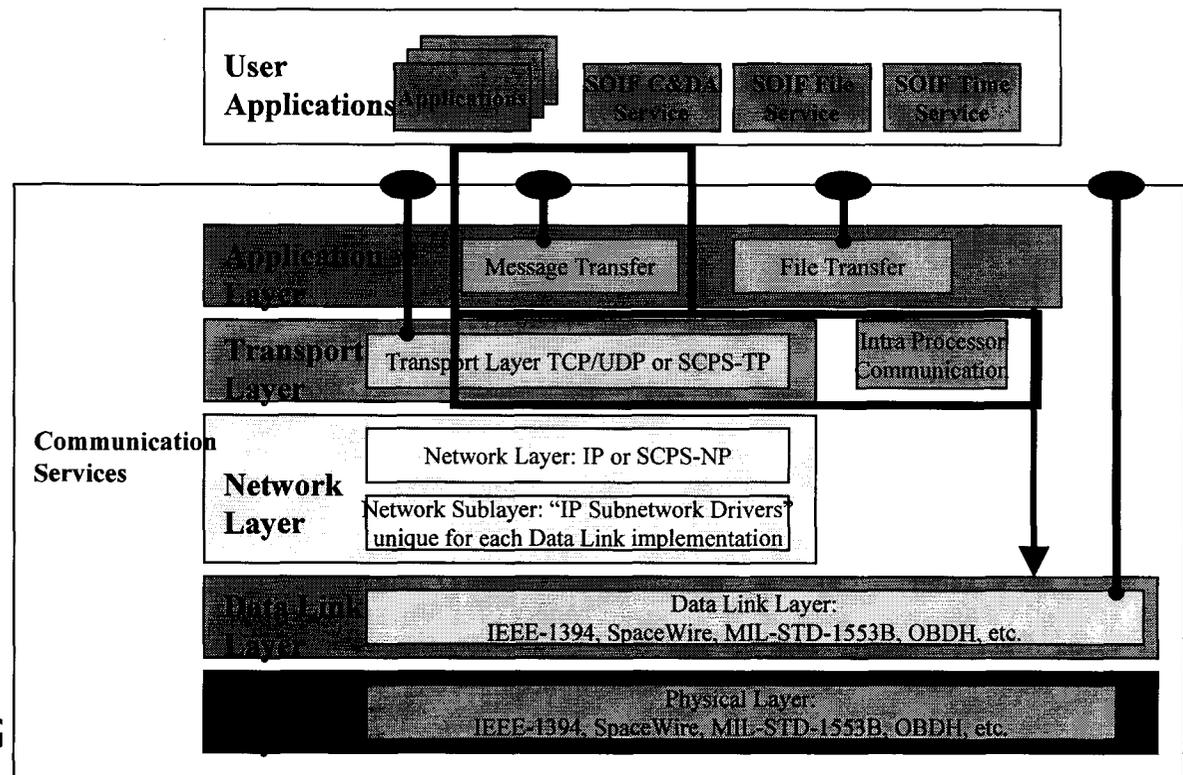
SOIF Implementation Model shows the relationship of the protocols to the Layers



NOTE: No security protocol is shown in order to simplify this diagram

Message Transfer Service API

- Provides Message Transfer interface offering confirmed & unconfirmed message delivery
 - Compliance Test Point for Message API
 - Socket I/F to TCP/UDP protocols
 - Interface to shared mem/pipes/Data link
- Possible Use of:
 - CORBA
 - RT/Corba
 - JAVA RT
 - Native MSG



NOTE: No security protocol is shown in order to simplify this diagram

SOIF

Use Cases

- Spacecraft Environments
 - Single Spacecraft
 - Multiple Spacecraft
 - Constellations / formation flyers
- Bus Environments
 - Simple single bus
 - Multiple bus, differential rates
 - Fail-over, fault tolerant
- Bus Topologies
 - Simple
 - Hierarchical
 - Redundant
- Instrument Models
 - Simple instruments, sensors, effectors
 - "Smart" instruments w/ micro-processors
- Processors
 - Single processors
 - Multiple processors
 - Heterogeneous processors
- Connectivity Models
 - Continuous (interactive)
 - Dis-continuous (store & forward)

Message Transfer Service

Definition

A set of common functions for Message Transfer and Quality of Service (QoS) management that sit above the transport layer in the communications stack and provide a service API for mediating the transfer of data between processes in a distributed system.

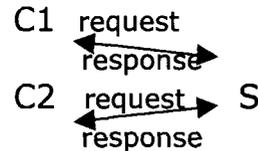
Assumptions:

- The processes run on a single system or multiple systems that may be heterogeneous
- The network can be LAN, WAN (Wire or RF) and Internet
- The Transport Layer functionality may be provided by TCP/IP or SCPS, or by a direct Data Link, or by efficient mechanisms such as shared memory, pipes, or IPC.

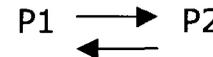
Message Transfer Service Context

Application Layer
Interaction Model

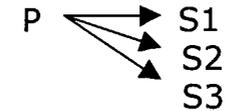
**1. Client/server
(listen/connect/accept)**



2. Peer-to-peer



3. Pub/Sub



Message Service API
(peer-to-peer)

msgInit()
bind()/resolve()
openChannel()
recvMsg()/sendMsg()

Transport Layer

Shared
Memory

Message
Queue

PIPEUDP

SCPS-TP

TCP

SCPS_TP

CFDP

- store & forward
- Msg = File
- Msg = DGRM

Data Link Layer

memory

O.S.
Service

Space Link
(TC & TM)

Proximity 1

Other Data Link
Bus Protocol

Message Transfer Service

High Level Requirements

1. The Message Transfer specification is to provide applications with the following services:
 - Transfer of information of arbitrary structure between processes over point to point or multi-point connections
 - Control and specification of the quality of service and service delivery mechanisms (timeliness, reliability, throughput, determinism, reporting, policies, transfer services)
 - Responsive control, monitoring and reporting of the service and/or connections
2. Application portability and interoperability across heterogeneous implementations

Message Transfer Service

High Level Goals

1. The Message Transfer implementation is to exhibit the following characteristics:
 - Low overhead
 - Low latency
 - Small "footprint"
 - Thread - safe
2. The Message Transfer implementation goals include
 - Portability & reusability across several different platforms
 - Interoperability among different language and OS platforms and over different communication protocols
 - Dynamic handling of connection state and failure modes
 - Support for multi- spacecraft constellations and formations (swarms)
 - Support for strongly typed messages & type safety checking during connection establishment (optional)

Message Transfer Service

Major Capabilities

- Name Space Lookup - to translate a process name to an address/path of some form
- Connection establishment - to link two or more processes for communication (early/late binding, service negotiation, policies)
- Synchronization - events, triggers, and asynchronous or synchronous communication

- Message Delivery - to send or receive messages
- Message Type - classes of messages that are transferred

- Transfer Mechanism - control of the transport protocol or link

- Error Handling - to detect, report, and recover from errors
- QoS - link performance and management

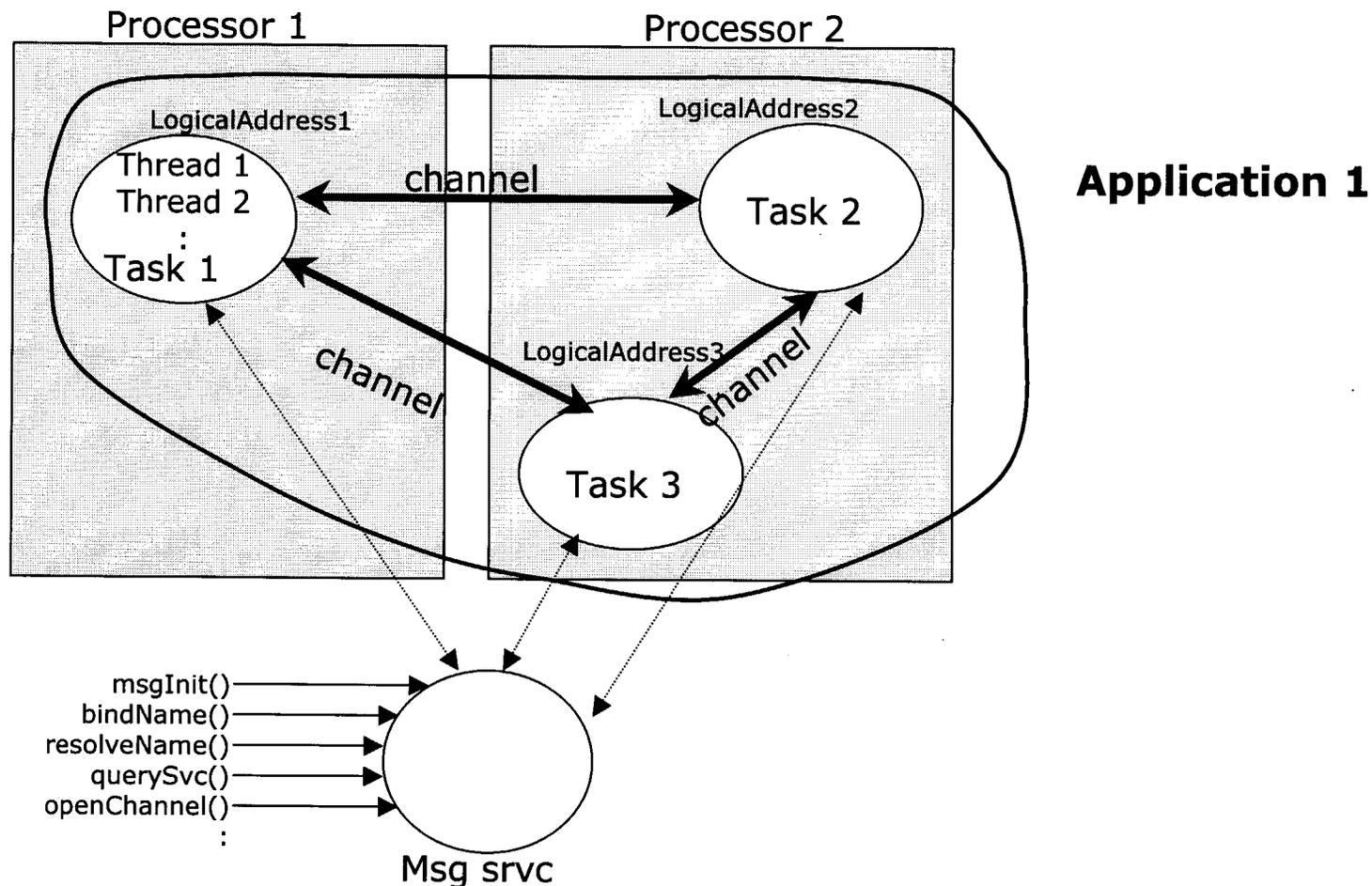
Message Transfer API

Proposed High-Level Functionality

- Communication Mechanisms
 - Service initiation / termination, authentication
 - Process - Process communication
 - send/receive (two-sided), put/get (one-sided), blocking/non-blocking
 - Multicast / Broadcast - specified set or promiscuous recipients
 - Publish/subscribe - managed delivery of data
 - Callback - event notification and handling
 - Quality of Service
 - Service & connection negotiation, policy management
 - Priority/scheduling/resource management
 - Reliability, determinism, ordering, timeliness
 - Message buffer management (make/free/get/set/init)
 - Error Handling, monitoring, signaling, reporting
 - Group management (joingroup/leavegroup)
-
- Derived data types (optional / future)
 - Packing/unpacking (optional / future)

Message Service Illustration

Major Elements (an example)



Message Transfer Service

Requirements on Lower Layers

1. The Message Transfer requirements on lower layer protocols include
 - Mechanisms to support early and late binding of connections
 - Mechanisms to support dynamic (re-)binding of connections
 - Support for resource reservation protocols such as RSVP
 - Mechanisms for negotiation of connection characteristics and QoS parameters
 - Means for signaling link state changes which may be link dependent (at least Red, Yellow, Green)

Implementation Notes

- The message service PDU shall contain mechanisms for encapsulating messages in a variety of formats
 - Initial message format will just be byte stream
 - Other message formats must be able to be encapsulated, including, at a minimum, CCSDS packets
- The message service PDU shall contain a version id such that future extensions may be gracefully managed
- The full envisioned QoS functionality may only be possible where underlying transport layers provide such services
 - The message service must be able to negotiate for such services and signal when the requested level of service cannot be achieved
 - The message service must be able to negotiate between deployments using different programming paradigms (e.g. threaded multi-cast vs publish / subscribe)
- The message service should be as parsimonious as possible, using optional “plug-ins” or similar approaches to reduce footprint of unused functions
- The Message Transfer Service shall be able to support a broad range of programming and dispatching paradigms, and mediate among them

Working Group Approach

- Study existing message transfer approaches
- Evaluate and validate or modify identified requirements, characteristics & functionality
- Develop alternative prototype approaches and evaluate APIs & results
- Evaluate fit of approaches to agreed requirements
- ▶ ▶ ▶ ■ Create integrated specification of API for Phase 1 subset functionality
- Prototype the selected approach in realistic simulated space environment

Message Passing API

High-Level Functionality - *Prototype*

■ Communication Mechanisms

- ***Service initiation / termination***, authentication, ***name space lookup***
- ***Process - Process communication***
 - ***send/receive (two-sided)***, put/get (one-sided), ***blocking/non-blocking, (trigger, polling)***
- Multicast / Broadcast - specified set or promiscuous recipients
- Publish/subscribe - managed delivery of data
- ***Event notification and handling (Callback)***

■ Quality of Service

- Service & connection negotiation, policy management
- Priority/scheduling/resource management
- Reliability, determinism, ordering, timeliness

■ ***Message buffer management (make/free/get/set/init) (local)***

■ ***Error Handling, monitoring, reporting (simple)***

■ Group management (joingroup/leavegroup)

Prototype Subset in bold italic

Initial Prototype Use Cases

- Single Processor / Single Bus
 - Simplest S/C configuration
 - Assumes Simple instruments
- Two Processors / Single Bus
 - Homogeneous processors
 - Assumes Smart instrument as instance of second processor

No RT extensions in initial tests

Only support continuous connectivity, reliable and unreliable QoS

Draft Message Transfer Service API

- Service Establishment
 - **initializeMsgService**
 - **finalizeMsgService**
 - **resolveName**
 - **bindName**
- Level of Service
 - **queryLoSCapability**
 - **setLoSCapability**
- Communication Mechanisms
 - **openChannel**
 - **closeChannel**
 - **sendMessage**
 - **sendMessagePoll**
 - **sendMessageAsynch**
 - **receiveMessage**
 - **receiveMessagePoll**
 - **registerAsynchRequestHandler**
 - **eventLoop**

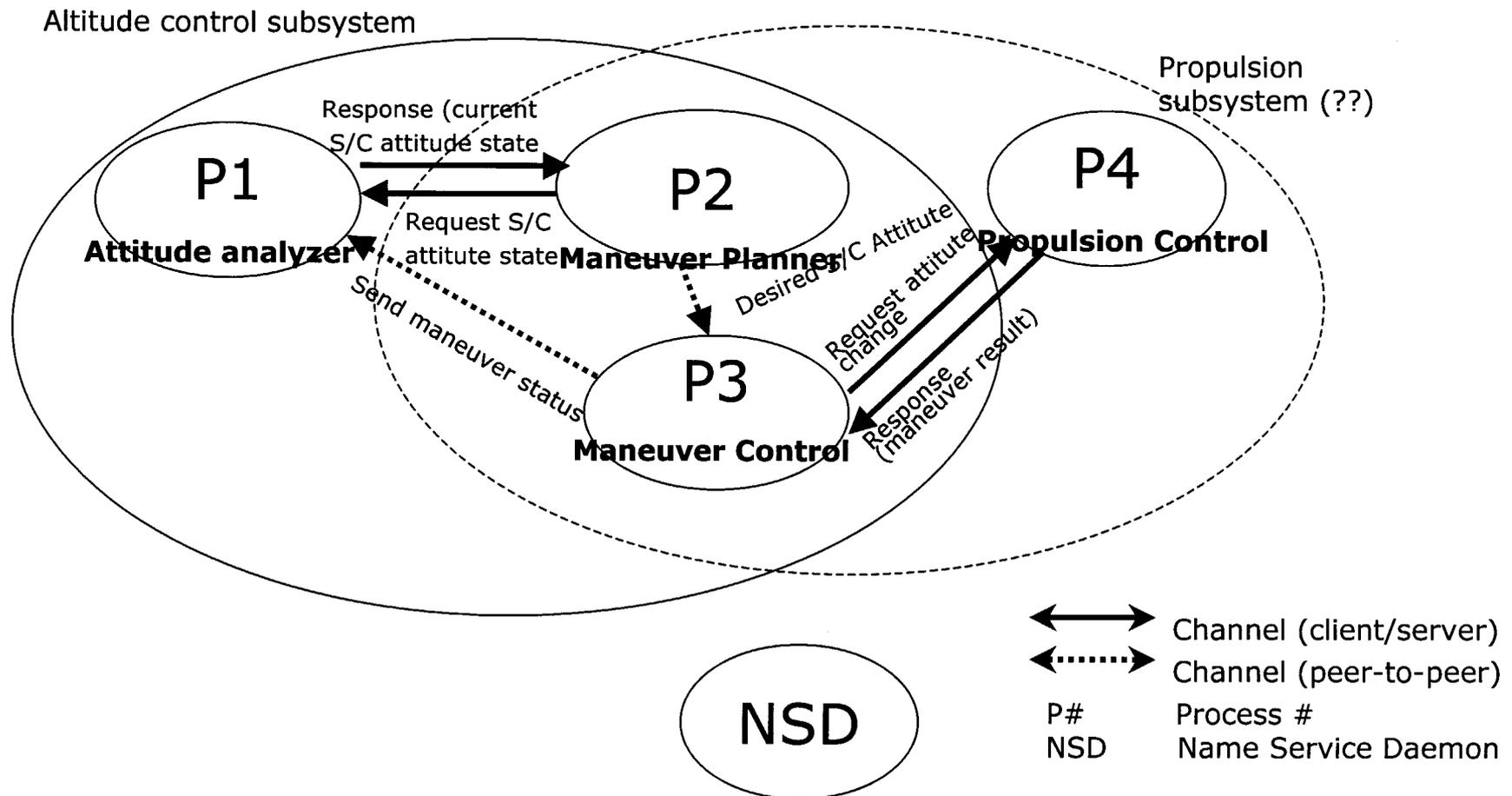
Summary

- We have high level agreement on requirements, goals, capabilities, and functionality of a message transfer service
- We have done three different prototypes to evaluate functionality, performance, footprint, and ease of use of approaches
- We have compared the results of these approaches and identified commonalities and differences
- We have a draft Phase 1 API based upon the results of this prototype activity
- Next generation prototyping of the common specification is underway

BACKUP

Message Service Example

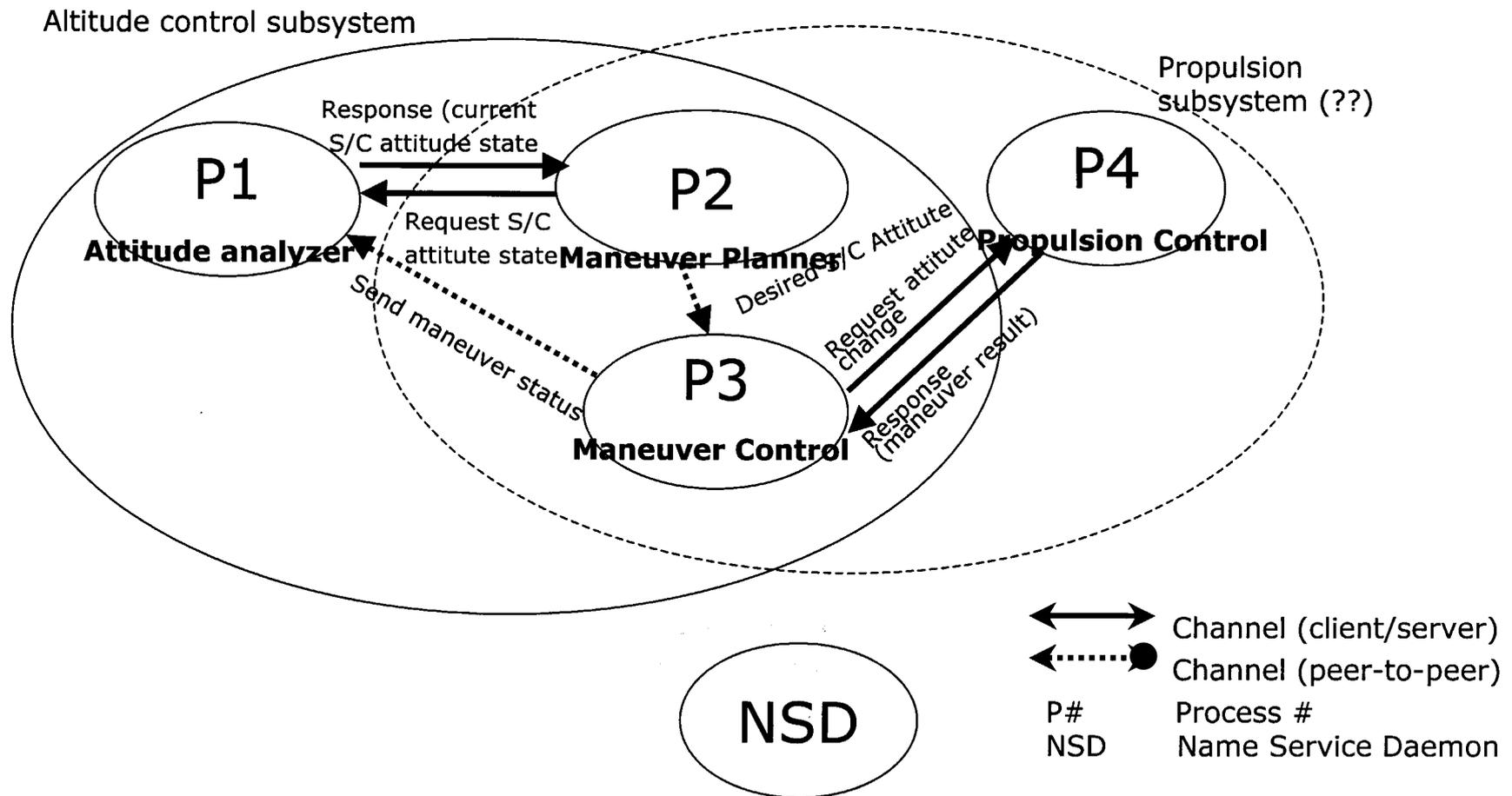
Use Case: Client/Server and Peer-to-Peer Interaction



BACKUP

Message Service Example

Use Case: Client/Server and Peer-to-Peer Interaction



Proposed Prototype Approaches

- JPL
 - MPI-RT analogue using ACE
- Science Systems
 - Minimum CORBA ORB, consideration of GIOP
- Logica
 - CDA Messaging using socket I/F

Prototype Message Traffic

- Status Traffic
 - 10 byte message, 10 byte response
- Control Traffic
 - 10 byte message, 1K byte response
- Configuration Traffic
 - 1K byte message, 10 byte response
- Data Transfer Traffic
 - 10 byte message, 100K byte response
- Test max rate synchronous communication
 - Count messages transferred during 1 minute
 - Each transfer waits for prior one to complete

Prototype Environment

- OS: Linux (probably Red Hat)
- Language: C, C++
- Networks: Ethernet 10 Mb, 100 Mb, isolated LAN
- Specify test configuration, processor, memory, interfaces, OS, compiler, libraries, network protocol version (TCP/IP, which options)

- Future: OS (VxWorks, RTEMS?, ENEA?), Networks (Firewire, Spacewire, RF, serial link), Protocols (SCPS)

Prototype Evaluation

- Functionality
- Performance
 - Throughput
 - Jitter
 - Latency
 - Protocol overhead
- Footprint
 - Memory load
 - CPU load

Foundation Fieldbus Industrial Process Control Layered Model

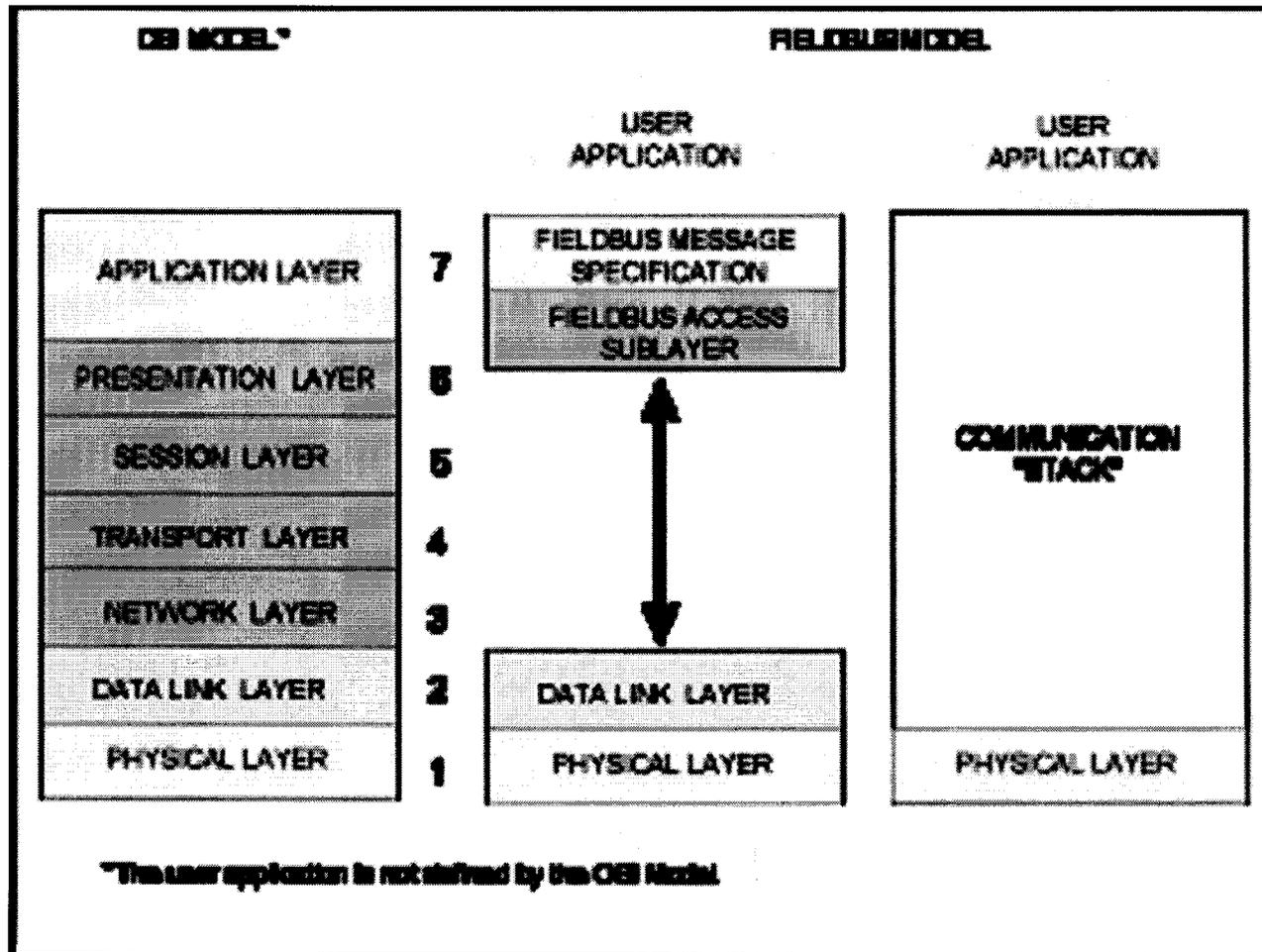
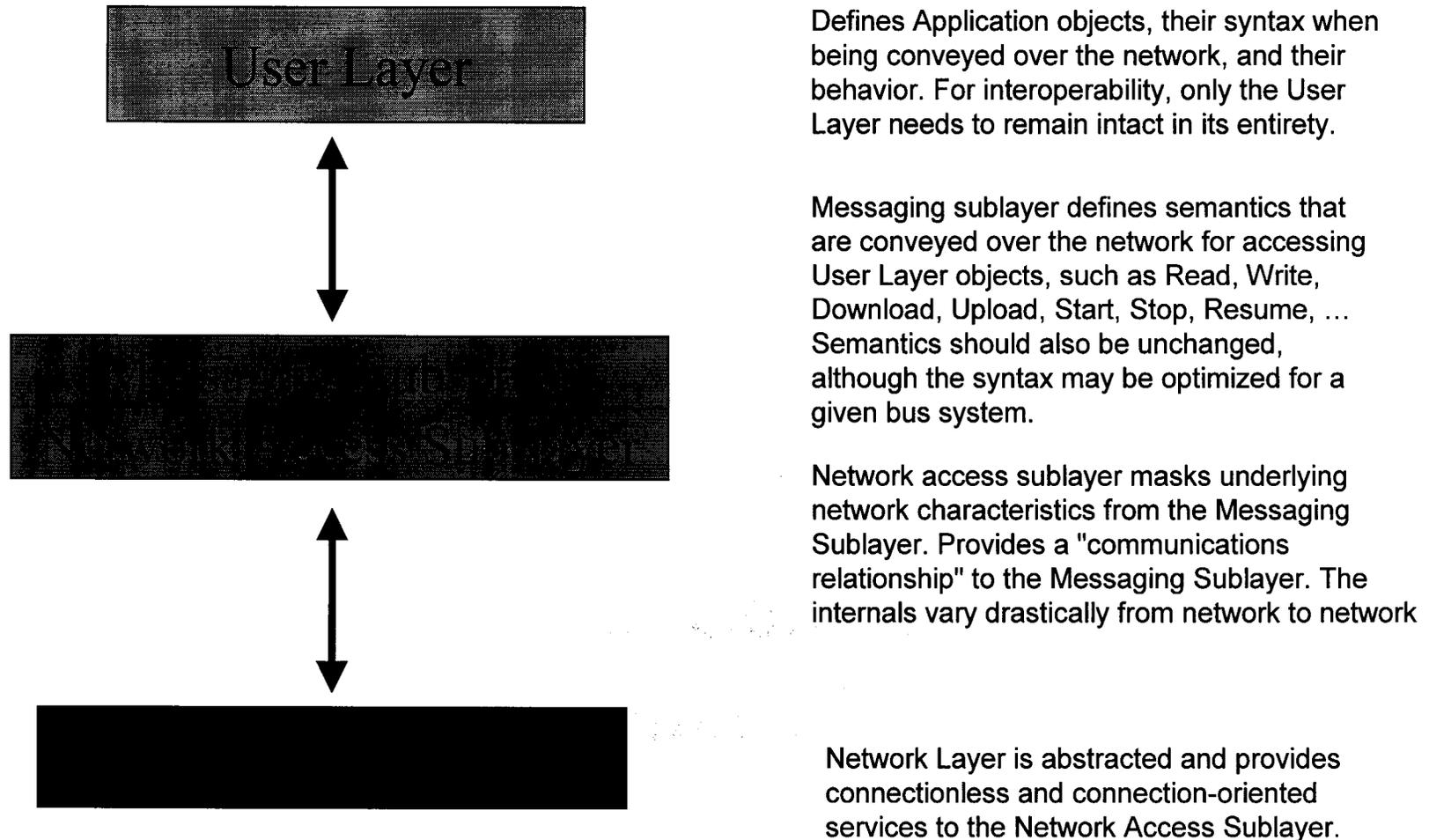
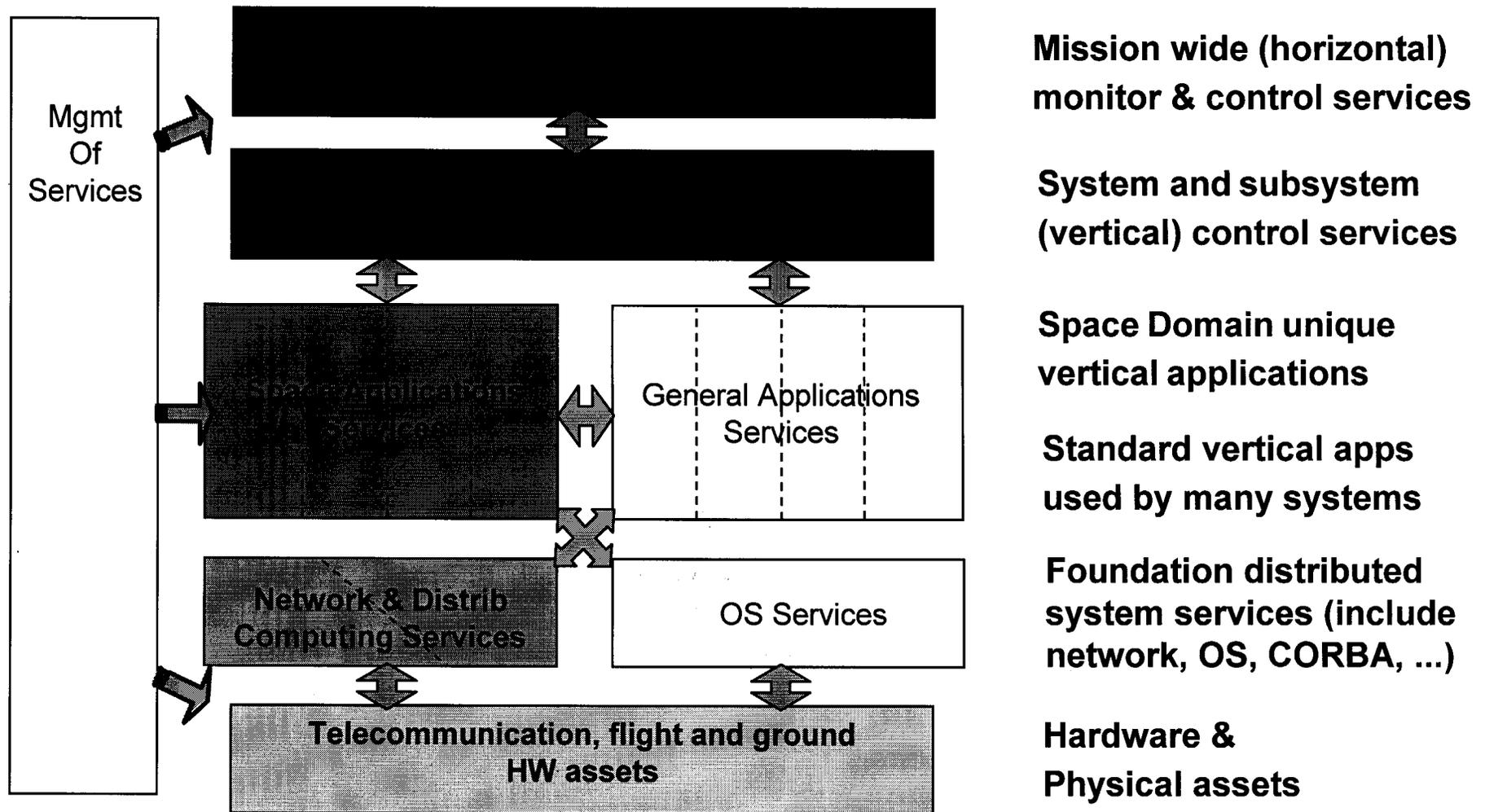
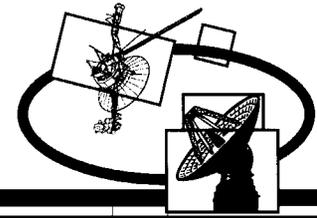


Figure 3. The Open Systems Interconnect (OSI) layered communications model.

IEC Industrial Communications Reference Model



OMG Reference Architecture Generic Functional Structure

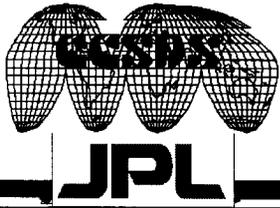


Source: OMG Space DTF

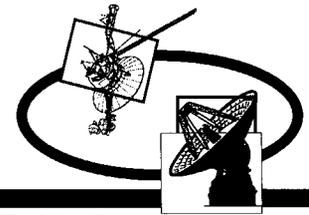
February, 1998

TMO Technology Program

PMBS 31

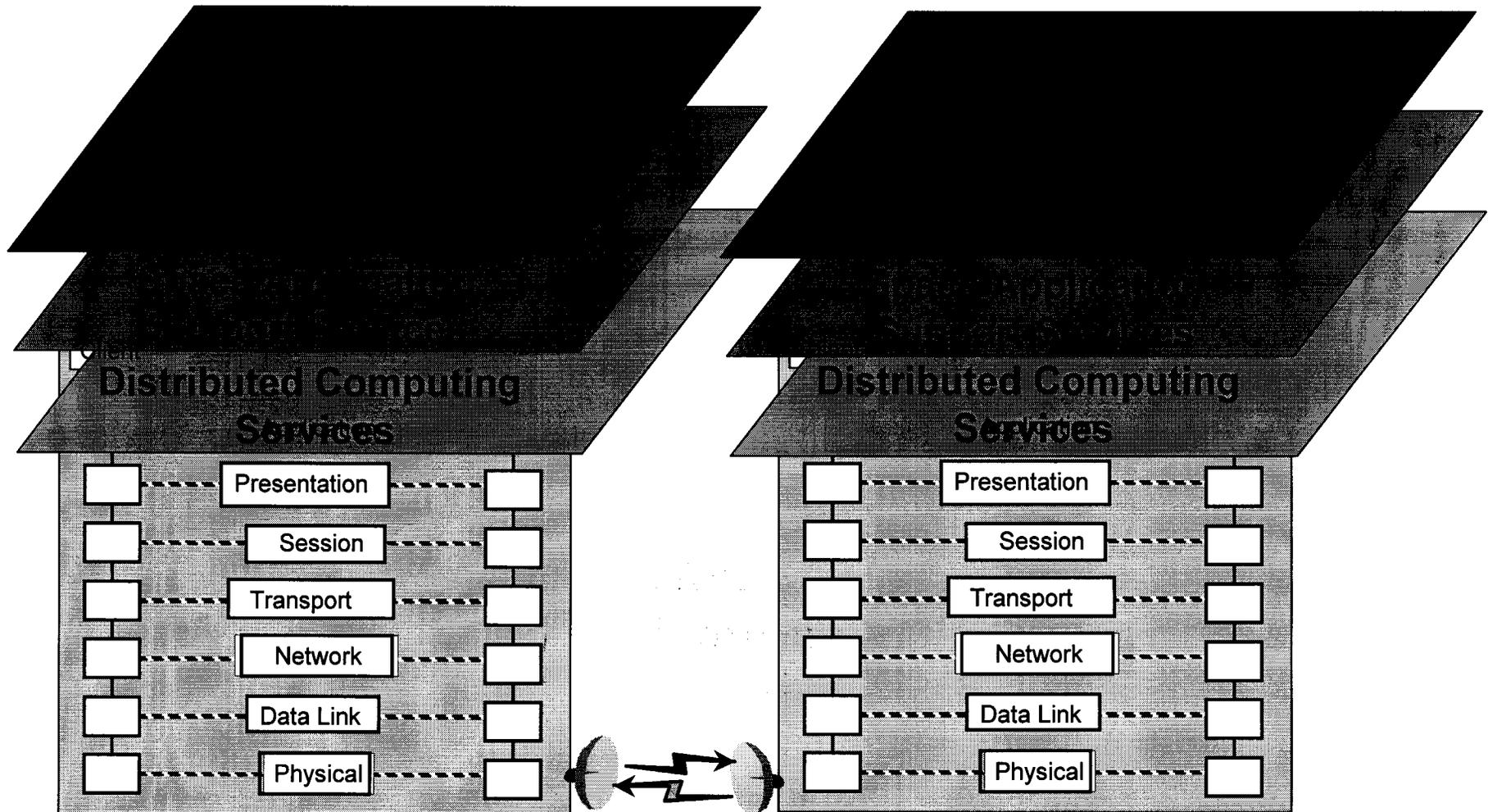


INTERPLANETARY NETWORK DIRECTORATE
**Orthogonal Communication Stack
and Applications Stacks**

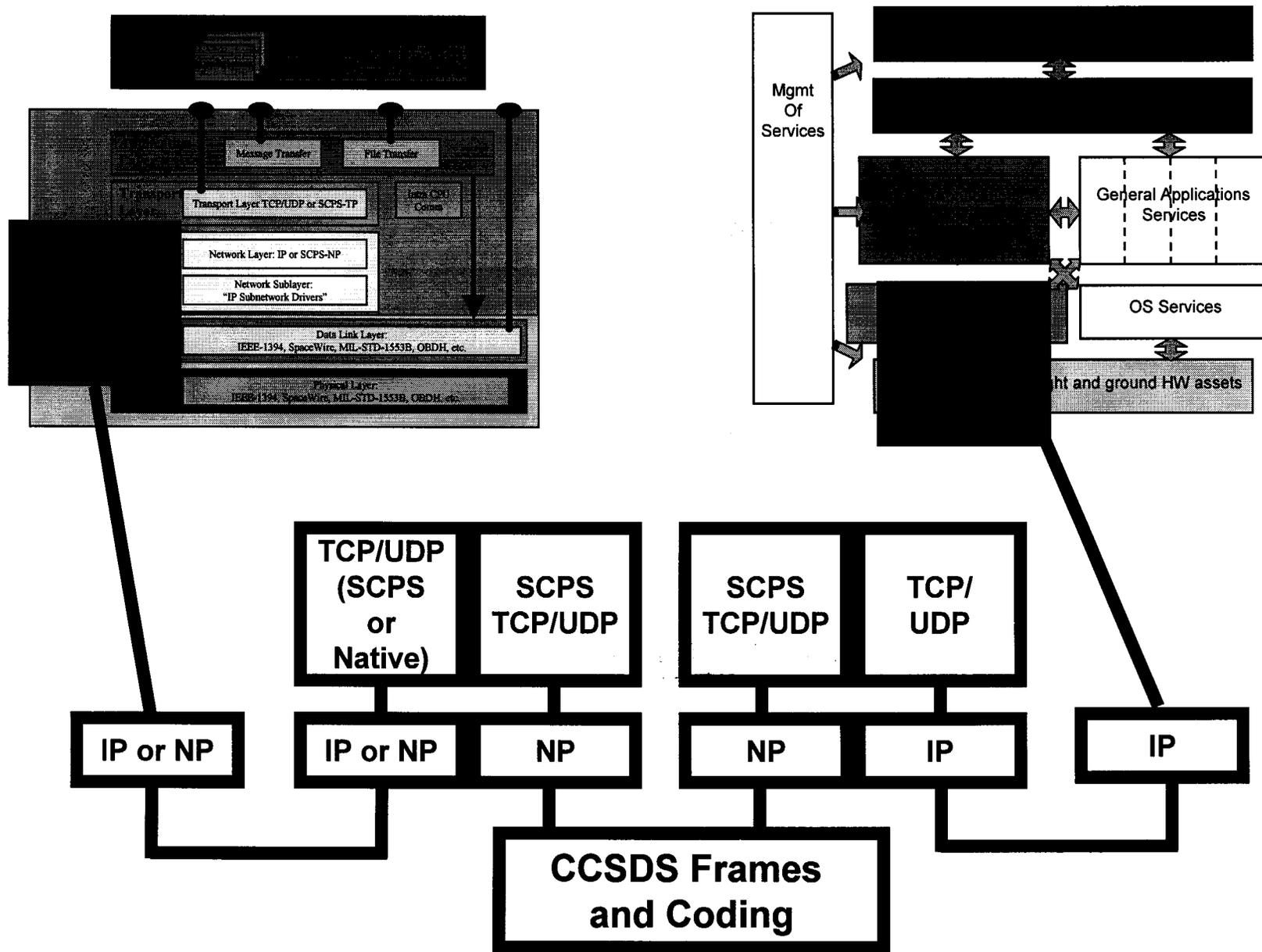


Flight

Ground



One View of Flight / Ground Integration



MPI/RT *(Prototype Example)*

Features

- **Early binding to achieve resource reservation and admission control**
- **"Channel" concept for message transfer**
 - **Persistent virtual connection model for communications**
 - **Point-to-point/collective** (broadcast, gather, scatter, reduce, barrier)
 - Quality of service guarantees and timing correctness
- **Programming paradigms**
 - Time-driven – starting time, deadline, activity interval (timeout)
 - **Event-driven – events trigger or stop application or MPI operations**
 - **trigger – "Send" side of event delivery abstraction**
 - **receptor – "Receive" side of event delivery abstraction**
 - **handler – a function that runs when certain conditions are met and is attached to receptors with QoS**
 - Priority-driven – only specified as additional constraints to event and time driven models
- **Two-sided, one-sided and zero-sided communication**
- **OO API**
- **Quality of Service**
 - QoS constraints – constrain start and/or finish times of an operation
 - QoS requirements – specified by users in terms of priorities or times (start/deadline) for an operation
 - Support Best-effort QoS
 - Applied to channels, handlers and event objects

MPI RT (*Prototype Example, cont*)

Core objects

- General
 - Group, CSet, cvector, Keyval, Dataspec, **Task_Address**
- Buffer Management
 - **Buffer**, BufIter
- Transfer Mechanism
 - **Ptchannel**
 - Bcast_Channel
 - Gather_Channel, Scatter_Channel, Reduce_Channel, Barrier_Channel
- Event Delivery
 - **Trigger, Receptor, Handler**
- QoS objects
 - Qos_Channel, Qos_Trigger, Qos_Handler, Qos_Receptor

MPI RT (*Prototype Example, cont*)

API (I)

■ Channel operations

- set_inbufiter/set_outbufiter/set_group/set_qos/**start/activate/deactivate/wait/test/set_name**/attach_attr

■ Ptchannel operations

- **create**/set_rank/set_direction/set_inbufiter/set_outbufiter/set_group/set_qos/**start/activate/deactivate/wait/test**/attach_attr

■ Broadcast channel operation

- create/set_inbufiter/set_outbufiter/set_group/set_qos/start/activate/deactivate/wait/test/attach_attr

■ Gather channel operation

- create/set_root/set_inbufiter/set_outbufiter/set_group/set_qos/start/activate/deactivate/wait/test/attach_attr

■ Scatter channel operation

- create/set_root/set_recvinbufiter/set_recvoutbufiter/set_inbufiter/set_outbufiter/set_group/set_qos/start/activate/deactivate/wait/test/attach_attr

MPI RT *(Prototype Example, cont)*

API (II)

■ Reduce channel channel operations

- create/set_recvinbufiter/set_recvoutbufiter/set_op/set_root/set_inbufiter/set_outbufiter/set_group/set_qos/start/activate/deactivate/wait/test/set_name/attach_attr

■ Barrier channel operations

- create/set_inbufiter/set_outbufiter/set_group/set_qos/start/activate/deactivate/wait/test/set_name/attach_attr

MPI RT (*Prototype Example, cont*)

API (III)

- QoS specific operations
 - Get_name/set_name/dup/free/is_equal/attach_attr/retrieve_attr/delete_attr
- Time-driven channel QoS specific operations
 - Create/set_period/set_release_time/set_deadline
- Event-driven channel QoS specific operations
 - Create/set_priority/set_start_event/set_start_scope/set_start_priority/set_start_time_spec/set_stop_event/set_stop_scope/set_stop_priority
- Event and Time-driven channel QoS specific operations
 - Create/set_priority/set_start_event/set_start_scope/set_start_priority/set_start_time_spec/set_stop_spec
- Buffer operations
 - Create/set_bufsize**/set_dataspec/set_base/assign_label/retrieve_label/set_name/attach_attr
- Buffer Iterator operations
 - Create/set_maxbufcount/set_dataspec/set_allowed_buffer_list/set_initial_buffer_list/set_policy/set_name/attach_attr/insert/remove

MPI RT *(Prototype Example, cont)*

API (IV)

- Handler QoS specific operations
 - create/set_priority/set_timeout/set_name
- Trigger QoS specific operations
 - create/set_bound/set_period/set_occurrence/set_name/attach_attr
- Receptor QoS specific operations
 - create/set_priority/set_time_spec/set_name/attach_attr
- Trigger operations
 - **set_event_name/set_scope/create**/set_qos/**raise**/set_time/attach_attr
- Receptor operations
 - **set_event_name/set_scope/create**/set_qos/**set_handlers/set_initial_state/enable/disable/set_name**/attach_attr
- Handler operations
 - **create/set_handler_fn**/set_qos/set_sync/set_extra_state/**set_name**/attach_attr

MPI RT (cont)

References

- *Real-Time Message Passing Interface (MPI/RT) Forum: Document for the Real-Time Message Passing Interface*, March 6, 2001
- *MPI/RT: An Emerging Standard for High-Performance Real-Time* by Systems A. Kanevsky, A. Skjellum, A. Rounbehler
- *The MPI/RT 1.1 Standard*, May 15, 2001 by Zhenqian Cui, Arkady Kanevsky, Anthony Skjellum