

D R A F T

Real-Time Message Passing Interface (MPI/RT) Standard “Best
Engineering Practices” Document

Real-Time Message Passing Interface (MPI/RT) Forum
<http://www.mpirt.org>

May 18, 1998

Contents

		1
		2
		3
		4
		5
		6
		7
		8
		9
		10
		11
Acknowledgments	v	12
Preface	1	13
Glossary	3	14
		15
		16
		17
I Optional Non-Real-Time Services	5	18
		19
1 MPI Profiles	7	20
1.1 Overview	7	21
		22
2 Issues in Resource Constrained Systems	9	23
2.1 Resource Constrained MPI	9	24
		25
3 MPI 1.2 Synopsis	11	26
		27
4 Resource Constrained MPI 1.2	12	28
4.1 Items included in the Resource Constrained MPI-1.2 Subset	14	29
4.2 Items excluded from the Resource Constrained MPI-1.2 Subset	16	30
		31
5 MPI 2.0 Synopsis	18	32
		33
6 Resource Constrained MPI 2.0	19	34
		35
II Interoperability With Other Standards	21	36
		37
7 MPI	23	38
7.1 Layering MPI on top of MPI/RT	23	39
		40
8 Vector Signal and Image Processing Forum (VSIP)	24	41
8.1 Introduction	24	42
8.2 VSIP Programming Model	24	43
8.3 MPI Programming Model	25	44
8.4 Interoperability	25	45
8.4.1 Using VSIP and MPI Interoperably	25	46
8.4.2 Using MPI/RT and VSIP Interoperably	29	47
		48

1	9 MPI/RT and Virtual Interface Architecture	33
2	9.1 Introduction to Virtual Interface Architecture (VIA)	33
3	9.2 MPI/RT Channel Admission	34
4		
5	10 Common Operating Environments (COE)	35
6	10.1 DoD Joint Technical Architecture	35
7	10.2 Status of JTA Effort	36
8	10.3 DII COE	36
9	10.4 Status of DII COE Effort	37
10	10.4.1 Navy TASP COE	37
11	10.4.2 Army WSTAWG OE	37
12	10.4.3 Air Force DII COE IPT	37
13	10.5 Opportunities for MPI/RT	37
14	10.6 Obstacles for MPI/RT	38
15		
16	11 PacketWay and Real-Time Messaging	39
17		
18	12 RSVP	40
19		
20	13 Data Reorganization	41
21		
22	14 IMPI/RT - Interoperability of MPI/RT Implementations	42
23		
24	15 Appendix	43
25		
26	Index	43
27		
28		
29		
30		
31		
32		
33		
34		
35		
36		
37		
38		
39		
40		
41		
42		
43		
44		
45		
46		
47		
48		

©1997-1998 Mississippi State University. Permission to copy without fee all or part of this material is granted, provided that the Mississippi State University copyright notice and the title of this document appear, and notice is given that copying is by permission of Mississippi State University is given.

Portions ©1993-1997 University of Tennessee, from “MPI-1: Message Passing Interface (MPI) Standard,” “MPI-2: Extensions to the Message Passing Interface (MPI) Standard,” and “MPI-2 Journal of Development (JOD)” documents. Copying is by permission of University of Tennessee.

Acknowledgments

This will be filled in later.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

Preface

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

Glossary

Current Status: No votes taken.

The following is a glossary of terms used in this document. This includes both terminology and acronyms relevant to MPI, and MPI/RT in particular.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

Part I

Optional Non-Real-Time Services

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

Chapter 1

MPI Profiles

Current Status: No votes taken.

There are several profiles of the MPI that are suitable for embedded systems, or for layering on top of MPI/RT. These profiles are independent of the programming paradigms that this chapter provides. All implementations are required to provide all profiles, though implementors may concentrate their efforts on the profile or profiles most relevant to their systems. This will provide for wide portability, and allow for appropriate investment in particular system niches.

1.1 Overview

Advice to implementors. Such profiles may be handled at compile/link time. We have an expectation that “smart linking” will be used to minimize the overhead associated with unused functions. This is compatible with the MPI-1 expectation of profiling libraries, that essentially puts one function per source file in most implementations. (*End of advice to implementors.*)

The support levels of profiles to be recognized by this standard are as follows:

- Resource constrained MPI-1.2 and MPI/RT,
- MPI-1.2 and MPI/RT,
- Resource constrained MPI-1.2, subset of MPI-2 and MPI/RT,
- MPI-1.2, subset of MPI-2 and MPI/RT.

The resource constrained MPI-1.2 in itself is not a real-time profile of MPI. The subset of MPI-2 to be supported will be determined in future discussion. It is anticipated to emphasize dynamic process management.

The underlying profiles are defined in the part ??.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

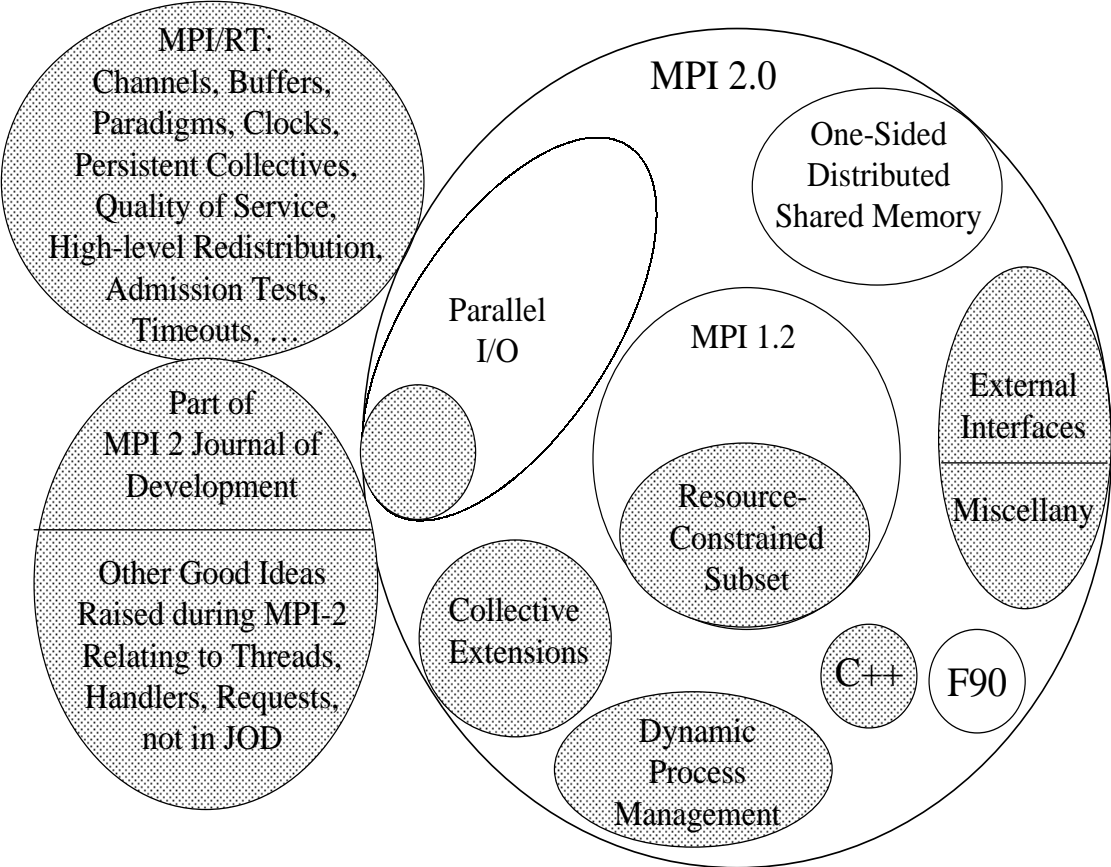


Figure 1.1: A taxonomy for MPI/RT, MPI-2, and MPI-2 JOD. *This figure is to be modified.*

Chapter 2

Issues in Resource Constrained Systems

Current Status: No votes taken.

For resource-constrained systems, the following issues become more important than in other uses of MPI:

- Small amounts of program space (code bloat unacceptable),
- Small amount of data space (buffering limited, maximum message sizes may be limited),
- Simplified programming environments as compared to full-blown OS's,
- Static loading environments more nearly compatible with MPI-2's view of the world.

Such systems may also have real-time requirements.

MPI is already migrating into resource-constrained prototype systems; hence it is interesting to offer suggestions, and possibly additional profile language, to address this particular space of applications and systems. One particular approach will be to offer a set of subset implementation profiles for resource-constrained systems, so that if subsetting should occur, it can be done according to a systematic convention documented in the MPI-2 standard.

2.1 Resource Constrained MPI

There are users within the resource-constrained computing arena that would like to use MPI, but are constrained by extreme limitations on local memory and storage space. These users often run code stored in non-volatile memory, such as FLASH, or ROM. While the use of an efficient linker can dramatically reduce the size of MPI libraries, the resulting binaries are still far too large to be incorporated into firmware. There are three particular areas of concern.

- Large executable size due to the many functions in MPI and their interdependence in many cases. (i.e., resource-constrained MPI may only be able to support a subset of MPI functionality.)
- The amount of buffer space on the receiver side. (i.e., the illusion of infinite slack may be particularly untenable.)

- The amount of buffer space on the sender side. (i.e., the ability to pack derived datatypes may be restricted.)

As a result, we recommend that only a core set of MPI functionality be required for resource-constrained MPI. (Exactly which routines should be preserved remains to be discussed.) In fact, because buffer space may be limited or nonexistent, the user should only expect that the synchronous and ready send operations are available (these routines require no explicit buffer space at the receiver) and that only the built-in datatypes are provided (neither requires buffer space at the sender or the receiver). The combination of these two restrictions would seem to completely eliminate the need for buffer space (except, perhaps, for the case of collective communication, which may require space for intermediate values). Alternatively, the user could be expected to provide buffer space explicitly via routines such as `MPI_BUFFER_ATTACH` and be restricted to using buffered send operations (as well as synchronous and ready ones). Also, direct memory transfer between source and destination buffers may be implementable only when the source of a message is explicitly specified in a receive call, so the use of `MPI_ANY_SOURCE` may be precluded.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

Chapter 3

MPI 1.2 Synopsis

Current Status: No votes taken.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

Chapter 4

Resource Constrained MPI 1.2

Current Status: No votes taken.

T (latest)

Resource Constrained MPI-1.2 is a set of restrictions to the MPI-1.2 standard intended for systems that have limited resources (typically embedded systems that have limited program and/or data memory). This document explicitly lists those functions, constants, structures, etc. that should, at a minimum, exist in a resource constrained implementation of MPI-1.2. This document also explicitly lists those functions, constants, etc. that may not be available in resource constrained implementations of MPI-1.2.

The resource constrained MPI-1.2 subset contains functions from the following categories:

- Point-to-point Communication (36)
- Collective Communication (14)
- Groups, Contexts, and Communicators (7)
- Environmental Inquiry (9)
- Profiling functions (1)

The number in parentheses is the number of API functions from these categories. It is stressed that this document does not intend to specify any changes to the semantics of the MPI-1.2 functions that are included in the subset.

The following is a list of the features currently excluded from the resource constrained MPI-1.2 subset:

- Process topologies (16)
- Explicit groups (14)
- Derived datatypes (10)
- Intercommunicators (5)
- Buffered send mode (5)
- Attribute caching (4)
- User defined error handlers (2)

- 1 • Default error handlers (2)
- 2
- 3 • User-defined collective computation operations (2)
- 4
- 5 • Send-receive replace (1)
- 6
- 7 • Error strings (1)
- 8
- 9 • Collective computation extended datatypes/MINLOC/MAXLOC (0)

10 The number in parentheses is the number of API functions removed from the standard due to the
11 removal of the feature.

12 For resource constrained implementations of MPI-1.2, it may be useful for users to know some
13 implementation defined limits.

14 *Advice to users.* MPI is not mandated to assume buffering, but most non-real-time imple-
15 mentations provide it. This is a strong advice to users and implementors of the resource-
16 constrained versions to expect programs that use any buffering to deadlock or otherwise fail.
17 (*End of advice to users.*)

18

19 *Advice to implementors.* High quality implementations will detail implementation specific
20 limitations such as:

- 21 • How much buffering does MPI_Send provide?
- 22 • What protocols are used for different sized messages?
- 23 • How many posted sends/recvs may be outstanding?
- 24 • Maximum number of communicators existing at one time
- 25 • Maximum number of communicators during lifetime of program (if an implementation
26 does not reuse contexts)
- 27 • Maximum number of unmatched envelopes
- 28 • Maximum number of persistent requests
- 29 • etc.

30

31

32

33 (*End of advice to implementors.*)

34

35

36 **Discussion:** Should there be extra version information for the resource constrained subset? Currently,
37 there is an MPI version and subversion for MPI-1.2.

38

39

40 **Discussion:** No mention of bindings is currently included in this document. Do we say that the
41 bindings are as specified in MPI-1.2 or do we want to say that the Fortran bindings are not expected to be
42 supported on resource constrained systems or ?

4.1 Items included in the Resource Constrained MPI-1.2 Subset

The following is the categorized list of items included in the resource constrained MPI-1.2 subset.

- Point-to-point Communication

MPI_Send	MPI_Recv	MPI_Get_count
MPI_Ssend	MPI_Rsend	MPI_Address
MPI_Sendrecv	MPI_Type_extent	MPI_Type_size
MPI_Isend	MPI_Issend	MPI_Irsend
MPI_Irecv	MPI_Wait	MPI_Test
MPI_Request_free	MPI_Waitany	MPI_Testany
MPI_Waitall	MPI_Testall	MPI_Waitsome
MPI_Testsome	MPI_Get_elements	MPI_Pack
MPI_Unpack	MPI_Pack_size	
MPI_Send_init	MPI_Ssend_init	MPI_Rsend_init
MPI_Recv_init	MPI_Start	MPI_Startall
MPI_Iprobe	MPI_Probe	
MPI_Cancel	MPI_Test_cancelled	

- Collective Communication

MPI_Bcast	MPI_Gather	MPI_Barrier
MPI_Gatherv	MPI_Scatter	MPI_Scatterv
MPI_Allgather	MPI_Allgatherv	MPI_Alltoall
MPI_Alltoallv	MPI_Allreduce	MPI_Reduce_scatter
MPI_Reduce	MPI_Scan	

- Groups, Contexts, and Communicators

MPI_Comm_size	MPI_Comm_rank	MPI_Attr_get
MPI_Comm_compare	MPI_Comm_dup	MPI_Comm_split
MPI_Comm_free		

- Environmental Inquiry

MPI_Init	MPI_Finalize	MPI_Get_version
MPI_Initialized	MPI_Wtime	MPI_Wtick
MPI_Error_class	MPI_Get_processor_name	MPI_Abort

- Profiling functions

MPI_Pcontrol

- Return codes

MPI_SUCCESS	MPI_ERR_BUFFER	MPI_ERR_COUNT
MPI_ERR_TYPE	MPI_ERR_TAG	MPI_ERR_COMM
MPI_ERR_RANK	MPI_ERR_REQUEST	MPI_ERR_ROOT
MPI_ERR_ARG	MPI_ERR_UNKNOWN	MPI_ERR_OP
MPI_ERR_TRUNCATE	MPI_ERR_OTHER	MPI_ERR_INTERN
MPI_PENDING	MPI_ERR_IN_STATUS	MPI_ERR_LASTCODE

4.1. ITEMS INCLUDED IN THE RESOURCE CONSTRAINED MPI-1.2 SUBSET

- 1 • Environmental inquiry keys
- 2
- 3 MPI_TAG_UB MPI_IO MPI_HOST
- 4 MPI_WTIME_IS_GLOBAL
- 5
- 6 • Assorted constants
- 7 MPI_PROC_NULL MPI_ANY_SOURCE MPI_ANY_TAG
- 8 MPI_UNDEFINED MPI_COMM_NULL MPI_REQUEST_NULL
- 9
- 10 • version constants
- 11
- 12 MPI_VERSION MPI_SUBVERSION
- 13
- 14 • Maximum sizes for strings
- 15 MPI_MAX_PROCESSOR_NAME
- 16
- 17 • Handles to assorted structures
- 18 MPI_Comm MPI_Datatype MPI_Status
- 19 MPI_Request MPI_Op
- 20
- 21 • reserved communicators
- 22
- 23 MPI_COMM_WORLD MPI_COMM_SELF
- 24
- 25 • Elementary datatypes
- 26 MPI_CHAR MPI_SHORT MPI_INT
- 27 MPI_LONG MPI_UNSIGNED_CHAR MPI_UNSIGNED_SHORT
- 28 MPI_UNSIGNED MPI_UNSIGNED_LONG MPI_FLOAT
- 29 MPI_DOUBLE MPI_LONG_DOUBLE MPI_BYTE
- 30 MPI_PACKED
- 31
- 32 • Optional datatypes
- 33
- 34 MPI_LONG_LONG_INT
- 35
- 36 • Collective computation operations
- 37 MPI_MAX MPI_MIN MPI_SUM
- 38 MPI_PROD MPI_BAND MPI_BOR
- 39 MPI_BXOR MPI_LAND MPI_LOR
- 40 MPI_LXOR
- 41
- 42 • Result of communicator comparisons
- 43 MPI_IDENT MPI_CONGRUENT MPI_SIMILAR
- 44 MPI_UNEQUAL
- 45
- 46 • Additional opaque types
- 47
- 48 MPI_Aint

4.2 Items excluded from the Resource Constrained MPI-1.2 Subset

The following is the list of items excluded from the resource constrained MPI-1.2 subset.

- Point-to-Point (16)

MPI_Type_contiguous	MPI_Type_vector	MPI_Type_hvector
MPI_Type_indexed	MPI_Type_hindexed	MPI_Type_struct
MPI_Type_lb	MPI_Type_ub	MPI_Type_commit
MPI_Type_free	MPI_Sendrecv_replace	
MPI_Bsend	MPI_Buffer_attach	MPI_Buffer_detach
MPI_Ibsend	MPI_Bsend_init	

- Collective Communication (2)

MPI_Op_create	MPI_Op_free
---------------	-------------

- Process Topologies (16)

MPI_Cart_create	MPI_Dims_create	MPI_Topo_test
MPI_Cartdim_get	MPI_Cart_get	MPI_Cart_rank
MPI_Cart_coords	MPI_Cart_sub	MPI_Cart_map
MPI_Cart_shift		
MPI_Graph_create	MPI_Graphdims_get	
MPI_Graph_neighbors	MPI_Graph_neighbors_count	
MPI_Graph_map	MPI_Graph_get	

- Communicators, groups, contexts (23)

MPI_Group_size	MPI_Group_rank	MPI_Group_translate_ranks
MPI_Group_compare	MPI_Comm_group	MPI_Group_union
MPI_Group_difference	MPI_Group_incl	MPI_Group_intersection
MPI_Group_excl	MPI_Group_range_incl	MPI_Group_range_excl
MPI_Group_free	MPI_Comm_create	
MPI_Comm_test_inter	MPI_Comm_remote_size	MPI_Comm_remote_group
MPI_Intercomm_create	MPI_Intercomm_merge	
MPI_Keyval_create	MPI_Keyval_free	MPI_Attr_put
MPI_Attr_delete		

- Environmental Inquiry (5)

MPI_Errhandler_create	MPI_Errhandler_get	MPI_Errhandler_free
MPI_Errhandler_set	MPI_Error_string	

- Handles to assorted structures

MPI_Group	MPI_Errhandler
-----------	----------------

- Collective computation operations

4.2. ITEMS EXCLUDED FROM THE RESOURCE CONSTRAINED MPI-1.2 SUBSET

```
1      MPI_MAXLOC          MPI_MINLOC
2
3      • Datatypes for reduction functions
4
5      MPI_FLOAT_INT      MPI_DOUBLE_INT      MPI_LONG_INT
6      MPI_2INT           MPI_SHORT_INT       MPI_LONG_DOUBLE_INT
7
8      • Assorted constants
9
10     MPI_KEYVAL_INVALID  MPI_BSEND_OVERHEAD
11
12     • Null handles
13
14     MPI_OP_NULL         MPI_DATATYPE_NULL   MPI_GROUP_NULL
15     MPI_ERRHANDLER_NULL
16
17     • Maximum sizes for strings
18
19     MPI_MAX_ERROR_STRING
20
21     • Topologies constants
22
23     MPI_CART            MPI_GRAPH
24
25     • Empty group
26
27     MPI_GROUP_EMPTY
28
29     • Special datatypes for constructing derived datatypes
30
31     MPI_UB              MPI_LB
32
33     • Return codes
34
35     MPI_ERR_GROUP       MPI_ERR_TOPOLOGY    MPI_ERR_DIMS
36
37     • Prototypes for user-defined functions
38
39     typedef void MPI_User_function(...)
40     typedef int  MPI_Copy_function(...)
41     typedef int  MPI_Delete_function(...)
42     typedef void MPI_Handler_function(...)
43
44     • Error-handling specifiers
45
46     MPI_ERRORS_ARE_FATAL MPI_ERRORS_RETURN
```

Chapter 5

MPI 2.0 Synopsis

Current Status: No votes taken.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

Chapter 6

Resource Constrained MPI 2.0

Current Status: No votes taken.

The following functionality of MPI 2.0 is considered to be RC MPI 2.0 along with the functions listed as RC MPI 1.2.

- The Process Creation and Management Chapter topics:
 - `MPI_Comm_spawn`
 - `MPI_Comm_spawn_multiple`
 - Info Argument Reserved Keys
 - `MPI_UNIVERSE_SIZE`
 - Singleton `MPI_Init`
- The Extended Collective Operations Chapter functionality as relevant to and pertaining to the collective operations in RC MPI 1.2.
- The External Interfaces section called MPI and Threads.
 - `MPI_Init_thread`
 - `MPI_Query_thread`
 - `MPI_Is_thread_main`

Categorically excluded from RC MPI 2.0 are the chapters:

- One-Sided Communications
- I/O

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

Part II

**Interoperability With Other
Standards**

Chapter 7

MPI

Current Status: No votes taken

7.1 Layering MPI on top of MPI/RT

A goal of MPI/RT is to provide a layer upon which MPI functionality can be implemented effectively. In order to be strictly compliant with the MPI standard, applications must call `MPI_INIT` to initialize the MPI environment. Due to the layering of these two software components, both the application and the MPI implementation must take care when initializing and finalizing both the MPI/RT and the MPI environments. An application or library which uses only the outer MPI layer need only call `MPI_INIT` and `MPI_FINALIZE`. In this case, it is the responsibility of the MPI implementation to set up and tear down the MPI/RT environment by calling both `MPIRT_INIT` and `MPIRT_FINALIZE`. An application or library which uses both the outer MPI layer and the inner MPI/RT layer may call both `MPIRT_INIT` and `MPI_INIT` in succession. Similarly, `MPI_FINALIZE` and `MPIRT_FINALIZE` must be called in succession. Both the application and the MPI implementation must be aware of this relationship. `MPI_INIT` must check whether the application has previously called `MPIRT_INIT` before attempting to do so. The implementation of `MPI_FINALIZE` must take care to only tear down the MPI environment, leaving the MPI/RT environment in a usable state. The MPI implementation should only call `MPIRT_FINALIZE` (within `MPI_FINALIZE` if it previously called `MPIRT_INIT`).

In the two-level middleware approach, MPI-1 is layered over MPI/RT. We then take the liberty at the MPI level of extending the API to accept handlers and timeouts, with the ability to use “ignorable” arguments. The specialization of this API with arguments set to ignorable recovers the original MPI API.

Discussion: Why are we mentioning MPI functionality here? What is the relationship between our handlers and MPI-2 handlers (Are they in the final MPI-2 document?) and MPI-1 error handlers?

`MPI_ERRHANDLER_CREATE` in MPI-1 is not general enough for our purposes, but clearly, the layering of this functionality on top of MPI/RT is possible. The actual function argument list for the handler is left open with an `stdargs` approach in MPI-1, in order to support variable argument lists for various usages, which are platform and implementation dependent. This may also aim at Fortran support.

We still need to consult the final form of MPI-2 for their results on external interface chapter, to see what was achieved there in conclusion.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

Chapter 8

Vector Signal and Image Processing Forum (VSIP)

Current Status: No votes taken

Discussion: This is the initial text for this chapter, and is subject to significant revisions.

8.1 Introduction

The Vector Signal Image Processing (VSIP) standard [?] provides an API for vector, signal, and image processing primitives for embedded real-time systems. This chapter describes the different issues involved in achieving base interoperability between the VSIP and MPI/RT libraries. Furthermore, because both VSIP and MPI/RT are components of the Tactical Advanced Signal Processor Common Operating Environment (TASP COE) [?], common underlying issues of basic datatypes and a derived datatype library are introduced as mechanisms for achieving better interoperability.

8.2 VSIP Programming Model

VSIP library uses abstract datatypes to represent conventional C datatypes. The implementation of the abstract datatypes is hidden from the user (opaque), both for software engineering reasons, and because the private data may itself be “special storage” of a given system with higher performance, but potentially complex access rules.

Because of the private data approach, the user must use the VSIP library functions to operate on the abstract datatypes. The VSIP library necessarily provides various constructors, destructors, and accessors to operate on the abstract datatypes. The data managed by the VSIP library is called *private array* and users cannot access the private arrays directly. By way of comparison the C arrays created by the user are called *public array*. The VSIP library does not support any operations on the public arrays other than copying data from public to private arrays and vice-versa. In addition to the copy function, VSIP allows the user to bind an abstract datatype to a public array to avoid copies when large arrays are used [?].

8.3 MPI Programming Model

MPI and MPI/RT support MIMD or SPMD programming model. Messages in MPI are non-opaque, and are defined in terms of typed or untyped data. Users can address such messages within communicators (which provide separation of communication via group and context) by destination/source, and user defined tag. Since MPI supports a MIMD or SPMD model the user programs know about “each other” unlike a client-server model.

For typed transfers, MPI datatypes describe the type of the data (*e.g.* float, double, and so on) and also provide a means to transfer non-contiguous data with out packing and unpacking data explicitly. Also datatypes support heterogeneous communication. Datatypes supported by the language are called *primitive datatypes* and datatypes that can be built from existing datatypes are called *derived datatypes*. Derived datatypes help the user to specify the layout of the data buffer [?].

MPI/RT compliance levels include MPI-1.2 (or a subset thereof), and also includes channel-based communication, as well as extensions to the collective operations of MPI. Emphasis is on early-binding, collective admission tests, and the use of buffer sets attached to ends of channels, coupled with queueing strategies. The memory associated with buffers in buffer sets is typically allocated by MPI/RT, not the user, so it may be “special,” analogous to VSIP private data. As such, users normally copy data into/out of MPI/RT buffers when transmitting information, whereas MPI data comes from arbitrary memory locations in general. MPI/RT also supports single data at end points of channels, in which case arbitrary gather/scatter of data can be accomplished. When buffer sets are used, data is assumed to be typed, but contiguous.

8.4 Interoperability

First, we consider how MPI and VSIP can interoperate in terms of point-to-point communication. Then, we discuss a collective example. Subsequent to this, we offer commentary on interoperability between VSIP and MPI/RT.

8.4.1 Using VSIP and MPI Interoperably

The problem to be addresses is that VSIP library user does not have direct access to the private arrays and the MPI library requires that such private arrays be used as the arguments for send or receive buffers. In this section several approaches are offered and then critiqued. Note that for non-quality-of-service-oriented communication, this section also applies to MPI/RT.

Approach #1

In order to transfer VSIP abstract datatypes between two or more processes the following approach can be followed:

Send Side	Receive Side
Create an abstract datatype	Create an abstract datatype
Initialize the abstract datatype	
Create a public array	Create a public array
Bind the public array to the abstract datatype	Bind the public array to the abstract datatype
Copy data from abstract datatype to public array	
Send the public array	Receive the public array
	Copy data from the public array to abstract datatype (private)
Free public array	Free public array

This sequence of events assumes that data starts out private on the sender side, and ends up private on the receiver side. Because MPI and MPI/RT work strictly on public data (data pointers), it is necessary to make two copies (one on sender side, one on receiver side). These copies are a priori an overhead from the point-of-view of performance.

A sample code to send and receive a vector view is shown in figure 8.1 and 8.2 respectively. In the following code segment we assumed that the value L is known to all processes. That assumption is reasonable for typical data parallel programs. See also Example #1.

```

vsip_vview_d *vecview = vsip_vcreate_d(L, 0);
vsip_scalar_d *buffer;
vsip_vview_d *data;

/* create a vsip vector with a public data space */
buffer = (vsip_scalar_d*) malloc(L * sizeof(vsip_scalar_d));

/* bind the public data with a vector view */
data = vsip_vbind_d(vsip_blockbind_d(buffer, L, 0), 0, 1, L);

/* copy the data into the public area */
vsip_vcopy_d_d(vecview, data);

/* send the vector */
MPI_Send((void*) buffer, L, MPI_DOUBLE, dest, tag, comm);

/* destroy allocated memory */
vsip_vdestroy_d(data);
free(buffer);

```

Figure 8.1: Sample code segment for send side with Approach #1

Advantages This approach is simple to implement and use. There is no additional functionality to be supported either by VSIP or MPI. Working code with the VSIP reference implementation and MPICH have already been demonstrated (see Appendix).

Disadvantages The main disadvantage of this approach is that data has to be copied from the private array to the public array on the send side and on the receive side data has to be copied

```

1      vsip_vview_d *vecview = vsip_vcreate_d(L, 0);
2      vsip_scalar_d *buffer;
3      vsip_vview_d *data;
4
5      /* allocate buffer to receive the data */
6      buffer = (vsip_scalar_d*) malloc(L * sizeof(vsip_scalar_d));
7
8      /* bind the public data buffer to a vector view */
9      data = vsip_vbind_d(vsip_blockbind_d(buffer,L,0),0,1,L);
10
11     /* receive the data */
12     MPI_Recv((void*) buffer, L, MPI_DOUBLE, src, tag, comm, &status);
13
14     /* copy data into private vsip vector */
15     vsip_vcopy_d_d(data,vecview);
16
17     /* free allocated memory */
18     vsip_vdestroy_d(data);
19     free(buffer);
20
21

```

Figure 8.2: Sample code segment for receive side with Approach #1

from the public array to the private array. This approach results in extra copies.

Datatype Compatibility Assumptions This code assumes that the MPI double-precision datatype, represented by “double” in C, and by `MPI_DOUBLE` as the abstract type name in MPI, are compatible with the underlying double-precision datatype of VSIP. That need not be so in practical implementations, nor need the size of integers be compatible, in principle. We will address such issues in a later paper.

Approach #2

In order to preserve the simplicity of approach # 1 as well as avoid the extra copies, David Schwartz of Hughes Research Laboratories proposed two additional VSIP functions that will provide a pointer to the private data block of a VSIP object [?]. The two functions proposed were:

- export function - returns a public block data array for a given private block data array.
- import function - returns a private block data array for a given public block data array.

Using the import and export functions the operations performed to transfer VSIP abstract datatypes between two or more processes will be as follows:

Send Side	Receive Side
Create an abstract datatype	Create an abstract datatype
Export data from abstract datatype (private) to public array	Export data from abstract datatype (private) to public array
Send the public array	Receive the public array
Import data from the public array to abstract datatype (private)	Import data from the public array to abstract datatype (private)

A sample code to send and receive a vector view is shown in figure 8.3 and 8.4 respectively.

```

1
2
3
4
5
6 vsip_vview_d *vecview = vsip_vcreate_d(L, 0);
7 vsip_scalar_d *buffer;
8
9 /* export the private data into the public area,
10    or get a pointer to the private block */
11 buffer = vsip_export_d(vecview);
12
13 /* send the vector */
14 MPI_Send((void*) buffer, L, MPI_DOUBLE, dest, tag, comm);
15
16 /* import public data into private vsip vector */
17 vsip_import_d(vecview);
18

```

Figure 8.3: Sample code segment for send side with Approach #2

```

19
20
21
22
23
24
25
26
27
28
29
30 vsip_vview_d *vecview = vsip_vcreate_d(L, 0);
31 vsip_scalar_d *buffer;
32
33 /* export the private data into the public area,
34    or get a pointer to the private block */
35 buffer = vsip_export_d(vecview);
36
37 /* receive the data */
38 MPI_Recv((void*) buffer, L, MPI_DOUBLE, src, tag, comm, &status);
39
40 /* import public data into private vsip vector */
41 vsip_import_d(vecview);
42

```

Figure 8.4: Sample code segment for receive side with Approach #2

1 Advantages Since we have access to the private data block through a pointer there will no extra
 2 copies involved on most implementations. On implementations where the private data is in special
 3 memory and a pointer to that data cannot be provided the export function would perform the
 4 necessary memory allocation and copying and the import function would perform the copying and
 5 memory deallocation.

6
 7 Disadvantages Currently, the VSIP import and export functions are not part of the specification,
 8 it's still remains as a proposal to the VSIP Forum.

10 Approach #3

11
 12 A third approach would be to develop a separate, generic datatype language and library specification
 13 to be supported by both VSIP and MPI libraries as part of the Tactical Advanced Signal Processor
 14 Program Common Operating Environment (TASP COE) [?]. This would involve joint collaboration
 15 between the VSIP, MPI/RT, and Real-time Operating System (RTOS) groups to develop such a
 16 specification. This approach has been accepted in principle to assist with TASP COE compliant
 17 systems, but would not be available on non-TASP-COE-compliant systems. Our goal is to refine
 18 this proposal in further communications, with an introduction provided here.

19 At present, the datatype language would address issues at two levels:

- 20 1. Define the sizes and precisions of MPI and VSIP datatypes, as well as overlap of the two
 21 systems,
- 22 2. Define the gather/scatter specifications suitable for use in both VSIP and MPI. This is pri-
 23 marily a concrete library for implementors to use in building MPI and VSIP strided datatypes.
 24
 25

26 MPI provides a sophisticated datatype language, and VSIP provides a useful gather/scatter specifi-
 27 cation. In a separate communication, we will offer a comprehensive underlying library that would
 28 support both systems. In this model, it is assumed that MPI would be extended to support a
 29 "MPI_DESCRIPTOR" datatype that works with underlying datatype specifications suitable for de-
 30 scribing gathers/scatters of private data of VSIP or similar libraries.

31 The generic datatype language would generalize MPI's single base address approach to offer
 32 comparable power for both absolute and relative datatypes. It would dissociate datatypes from
 33 the specifics of either VSIP or MPI, and provide generic interfaces for constructing and destructing
 34 both relative and absolute datatypes, emphasizing both early and late bindings, where appropriate.
 35

36 8.4.2 Using MPI/RT and VSIP Interoperably

37 In this part of the paper, we turn from the classical MPI, to issues involving MPI/RT. Some of the
 38 above arguments are also relevant to MPI/RT's interoperation with VSIP.
 39

40 Case #1 - Single Endpoint Buffer Sets

41
 42 For the case where single buffers are used with MPI/RT channels, the gather/scatter issues defined
 43 above with MPI are the same. For Approach #1 above, a single copy at each end of the transmission
 44 will occur. For Approach #2, involving the datatype language, the copies are also avoided.
 45
 46
 47
 48

Case #2 - General Endpoint Buffer Sets

For the general case, each endpoint of an MPI/RT channel has a set of equivalent buffers, allocated normally by the system. These buffers are contiguous in memory. For contiguous VSIP objects utilizing the private memory abstraction, a copy must be made from such memory to the MPI/RT buffer set entry to be transmitted.¹ There must be a copy in on send, and a copy out on receive.

The copies can be avoided by associating VSIP private memory with MPI/RT special memory, as part of a TASP-COE-compliant implementation, specifically through the generic datatype library, plus the mechanism of `MPIRT_MALLOC` for allocation of special memory usable jointly by VSIP and MPI/RT. Since MPI/RT accepts both standard memory, and its own special memory for buffer set elements, this constitutes no new extension of the MPI/RT model, but may totally avoid copies between VSIP objects for contiguous vectors or contiguous views thereof. For non-contiguous views, there must be a copy between VSIP memory and MPI/RT buffer pools at present in order to transmit or receive.

However, MPI/RT's specification could be relaxed to support gather/scatter with virtual buffer pool entries, if desired.² If this specification change is made, MPI/RT can allow zero-copy gather/scatter operations to be connected with VSIP views, provided the generic datatype library is also present.

Discussion: Specific examples will be added in the next release of this document.

¹Currently, whereas VSIP assumes that private data may have special address space properties as well as software engineering protections, MPI/RT assumes that buffer set addresses will be accessible to users from the native language features.

²Because datatypes are currently slow in MPI-1 implementations, and because datatypes do not generally lead to higher performance without hardware acceleration, it is often the case that user code does as well or better than derived datatype code with MPI at present. Furthermore, the architecture of MPICH, a commonly used implementation, is specifically suboptimal in that it mandates copies into contiguous buffers of strided data. This guarantees that the user experiences no more copies with his or her own compression of data into contiguous memory prior to transmission with either MPI or MPI/RT. This experience does not mean that MPI/RT with suitable gather/scatter hardware would be slower than user code, but this promise of higher performance must overcome early performance results mentioned above.

Example #1

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

```

/*This program done by Randy Judd NRaD D881 July 31 1997*/
/*This is the work of a Government Employee*/
/*This work is funded by PMS 428*/

/* Modified by Puri Bangalore (MSU) 08/08/97 */

#include <stdio.h>
#include "mpi.h"
#include "vsip_vview_d.h"
#include "vsip_block_d.h"
#include "vsip_scalar_d.h"
#define PI 3.1415926535
#define N 10 /*length of vsip vectors*/

int main( int argc, char **argv)
{
    int i, rank, size;
    vsip_length L;
    vsip_vview_d *ramp, *data;
    vsip_scalar_d *buffer;
    MPI_Status status;

    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    if(size < 2){
        printf("You need 2 processes to run this\n");
        MPI_Finalize();
        exit(-1);
    }
    if(rank == 0){/*process zero computes a ramp*/
        ramp = vsip_vbind_d(vsip_blockcreate_d(2*N,0),1,2,N);

        /*make a dataRamp from 0 to 2 pi inclusive*/
        vsip_vramp_d(0,(2 * PI / (N - 1)),ramp);
        /*print out the results*/
        printf("This is a ramp from 0 to 2 pi\n");
        for(i=0; i<N; i++)
            printf("%5.3f ", vsip_vget_d(ramp,i));
        printf("\n");

        L = vsip_vgetlength_d(ramp);
        /* create a vsip vector with a public data space */
        buffer = (vsip_scalar_d*) malloc(L * sizeof(vsip_scalar_d));

        /* bind the public data with a vector view */
        data = vsip_vbind_d(vsip_blockbind_d(buffer, L, 0),0,1,L);

        /* copy the data into the public area */
        vsip_vcopy_d_d(ramp,data);

        for(i=1; i<size; i++) {

```

CHAPTER 8. VECTOR SIGNAL AND IMAGE PROCESSING FORUM (VSIP)

```

        MPI_Send((int *)&L,1,MPI_INT,i,0,MPI_COMM_WORLD);
        MPI_Send((void*)buffer,L,MPI_DOUBLE,i,0,MPI_COMM_WORLD);
    }
    /* free the buffer and vector view */
    free(buffer);
    vsip_blockdestroy_d(vsip_vdestroy_d(ramp));
}
else{/*all other processes compute the sin of the ramp*/
    MPI_Recv((int *)&L,1,MPI_INT,0,0,MPI_COMM_WORLD,&status);

    /* ramp = vsip_vcreate_d(L,0); */
    ramp = vsip_vbind_d(vsip_blockcreate_d(2 *L,0),1,2,L);

    /* allocate buffer to recieve the data */
    buffer = (vsip_scalar_d*) malloc(L * sizeof(vsip_scalar_d));

    /* bind the public data buffer to a vector view */
    data = vsip_vbind_d(vsip_blockbind_d(buffer,L,0),0,1,L);

    /* receive the data */
    MPI_Recv((void*) buffer, L, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,&status);

    /* copy data into private vsip vector */
    vsip_vcopy_d_d(data,ramp);

    /* compute the sin of the ramp */
    vsip_vsin_d(ramp,ramp);

    /* display the results */
    printf("My rank is %i\n",rank);
    printf("This is the sin of the ramp from 0 to 2 pi\n");
    for(i=0; i<L; i++)
        printf("%5.3f ", vsip_vget_d(ramp,i));
    printf("\n");
    /* free allocated memory */
    free(buffer);
    vsip_blockdestroy_d(vsip_vdestroy_d(ramp));
}
MPI_Finalize();
return 0;
}

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

Chapter 9

MPI/RT and Virtual Interface Architecture

Current Status: No votes taken

9.1 Introduction to Virtual Interface Architecture (VIA)

VIA specifies the interface between a computer system and a system area network (SAN). The main objective of VIA is to reduce the message passing latency by optimizing the critical data path from user buffers to the physical medium of the SAN. This results in an efficient data transmission, better utilization of network resources, and more deterministic behavior of the communication processes.

The basic abstraction that VIA provides to the user applications is the Virtual Interface (VI.) A VI is a set of software mechanisms for data transfer, notification, and synchronization between application processes and network hardware. VIA defines a VI connection as a point-to-point logical link between two VIs, possibly residing on remote computer systems connected by a VIA network.

An MPI/RT implementation can benefit from the following VIA features:

1. Zero-copy message passing,
2. Scatter/gather mode of data transfer,
3. Exclusion of host OS from the critical data path,
4. Remote DMA operations,
5. Optimizations that accelerate persistent calls,
6. VI connections,
7. Notion of QoS associated with VIs,
8. Passive and active open for VI connections.

The zero-copy message passing and remote DMA operations facilitate a high-speed low-latency data transfer. The exclusion of the host OS from the actual data transfer leads to reducing message passing latency and achieving a higher degree of predictability of the communication procedures.

Scatter/gather mode optimizes the transmission of complex data types by avoiding extra gathering copies. This mode also allows the MPI/RT messaging subsystem to process efficiently packets by gathering/scattering user data and packet headers from/to different memory locations in operations performed by the VIA hardware.

MPI/RT channels can be naturally mapped to the VI connections; the high-level MPI/RT channel abstraction and its QoS have corresponding low-level mechanisms in VIA. A VI connection can be associated with each MPI/RT channel. The channel QoS can be represented with QoS of the end-point VIs participating in a connection. Since VIA defines QoS as a VI attribute (not as a connection attribute), QoS could be specified independently at both ends of a VI connection. When a connection is established the active side requesting the connection provides its QoS to the VI that is in state of passive open. Depending on the remote VI QoS the passive VI can either accept or reject the requested connection. Thus, MPI/RT channels that cannot satisfy their real-time requirements can fail at initialization time.

In its current specification [?], VIA does not define the types of QoS that VIA networks should support. In order an MPI/RT implementation to provide real-time message passing services to applications, as a minimum VIA should support the following QoS:

1. Maximum rate of fixed size messages
2. Upper bound of end-to-end delay
3. Upper bound of jitter

A form of probabilistic descriptions of QoS parameters is a favorable VIA feature that will allow for a more efficient use of network resources.

9.2 MPI/RT Channel Admission

An MPI/RT implementation based on a VIA network can employ different schemes for channel admission. The goal of channel admission is to allocate VIA network resources to MPI/RT channels according to the message-passing requirements and generate a schedule for the communication activities, if real-time constraints are satisfied. Channel admission is a crucial mechanism for the correct functionality of MPI/RT.

A channel admission scheme can use a sequential on-line admission algorithm that is applied to channels by dynamically reflecting the consumption of network resources. An alternate admission scheme may perform collective admission of all channels simultaneously. In the latter case, global information about all channel requests in the system is needed. This information should be distributed prior to the admission procedure to all VIA nodes that participate in an MPI/RT instance. Two approaches for collective admission are possible. In the first approach, network resources are allocated based on the assumption for the worst case QoS. This is a pessimistic view on the behavior of a real system and, although this approach can create schedules with high likelihood for success, it may reject a large number of feasible schedules. In the second approach, a more precise mathematical model of the real-time system can be used. This model will take into account the probabilistic nature of the communication processes. An admission scheme for collective channel admission using the second approach could generate more efficient schedules that meet a wider range of real-time constraints.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

Chapter 10

Common Operating Environments (COE)

T (latest)

Common operating environments integrate middleware in order to enhance platform portability, and improve software lifecycle characteristics MPI/RT will figure strongly in several of the developing COEs. The purpose of this chapter is to address issues concerning MPI/RT cooperation and interoperability with Department of Defense (DoD) COE efforts. Opportunities and obstacles to integrating with those efforts will also be addressed

10.1 DoD Joint Technical Architecture

In November 1995 the Assistant Secretary of Defense (ASD) Command, Control, Communications, and Intelligence (C3I) issued a memorandum directing the establishment of a single, unifying DoD technical architecture that is binding on all future DoD C4I acquisitions so that “new systems can be born joint and interoperable, and existing systems will have a baseline to move towards to ensure interoperability.”

The resulting document is the DoD Joint Technical Architecture (JTA) whose stated purpose is to:

- Provide the foundation for a seamless flow of information and interoperability among all tactical, strategic, and sustaining base systems that produce, use, or exchange information electronically.
- Mandate standards and guidelines for system development and acquisition which will significantly reduce cost, development time, and fielding time for improved systems, while minimizing the impact on program performance wherever possible.
- Influence the direction of the information industry’s standards-based product development by stating the DoD’s direction and investment so that information industry’s development can be more readily leveraged in systems within DoD.
- Communicate DoD’s intent to use open systems products and implementations to industry. DoD will buy commercial products and systems, which use open standards, to obtain the most value for limited procurement dollars.

Section 2 of the JTA contains a detail of the DoD Technical Reference Model (TRM) which is the Architecture’s “..basis for the specification of standards.” The MPI/RT standard conceptually

fits into the *Application Platform* layer of this model and provides capabilities that overlap three Service areas: *Communications*, *Operating System* and *Distributed Computing*.

10.2 Status of JTA Effort

Both the JTA and the DII COE (addressed below) were created to reduce stovepipe development of C4I systems. Due to the perceived benefits of the JTA/DII COE model, the scopes of these efforts are expanding outside of the C4I domain. As of this writing the JTA Version 2.0 is being released (comments are based on draft document and can be subject to change). Two of the new domains that JTA will mandate standards for, are Weapon Systems and Airborne Reconnaissance. The addition of these annexes means that the JTA is starting to address real-time and resource constrained systems, which will be of interest to MPI/RT. Broken down by Service areas, JTA 2.0 specifically mandates the following:

1. *Communications Services*: The JTA identifies Information Transfer Standards, which are meant to be used for implementing TRM Communication Services. The current standards addressed are for accessing subnetworks, and internetworking between subnetworks. Information Transfer Standards for a System Area Network (SAN) are not explicitly identified. There is a End System Standard Section (under Emerging Standards) that would be a good fit for MPI/RT as a SAN standard.

Of note, the Reservation Protocol (RSVP) standard that is being developed by the IETF is indicated in the JTA as an emerging standard.

2. *Operating System Services*: POSIX.21 is listed as an emerging standard that is expected to be adopted when finalized. There is no currently mandated real-time distributed communication standard.
3. *Distributed Computing Services* interoperability with CORBA is mandated for Distributed Object Computing. Other technologies may be used however the burden of interoperability is on the non-CORBA system. Open Group's DCE is mandated for Remote Procedure Computing. There is no mention of OMG's pending RT CORBA standard yet.

10.3 DII COE

The Defense Information Infrastructure Common Operating Environment (DII COE) is a software implementation that has been developed in concert with the JTA vision. The JTA actually mandates the use of DII COE products in C4I systems. The DII COE emphasizes both software reuse and inter-operability. Specifically, the COE concept emphasizes:

- an architecture and approach for building interoperable systems
- an infrastructure for supporting mission area applications
- a rigorous definition of the run-time execution environment
- a rigorous set of requirements for achieving COE compliance
- an automated tool-set for enforcing COE principles and measuring COE compliance
- an automated process for software integration

- 1 • a collection of reusable software components
- 2
- 3 • an approach and methodology for software reuse
- 4
- 5 • a set of APIs for accessing COE components
- 6

7 10.4 Status of DII COE Effort

8
9 Starting in July 1997 a DII COE Real-time Technical Working Group (RT TWG) was formed to
10 address the needs of DoD systems that have hard and soft real-time requirements. The group's final
11 product will be real-time extensions to the existing DII COE. An Integrated Product Team has
12 been formed which is responsible for coordinating the integration of components into the real-time
13 COE based on the requirements generated by the TWG.

14 The TWG is initially focusing on issues such as certification procedures for a real-time plat-
15 form, fault tolerance, real-time databases, and real-time distributed computing/multi-processing.
16 Eventually the group will also address the embedded (resource limited) problem space. Currently,
17 several products are being evaluated for applicability, including MPI/RT. The emerging RT CORBA
18 is being considered for introduction into the RT COE to support distributed computing by CY00.

19 Since the RT DII COE is a nascent effort individual Services have developed their own COE
20 efforts that can ultimately contribute to a unified COE.

21 10.4.1 Navy TASP COE

22
23 The purpose of the Navy Tactical Advanced Signal Processor Program (TASP) is to enable Navy
24 and other DoD programs to use commercially developed and available digital signal processing
25 equipment in military weapon system applications. A major objective of the TASP program is to
26 encourage industry to develop and adopt a set of open standards for key Application Programming
27 Interfaces that will greatly aid in the migration of application software. In order to achieve this
28 objective, DoD is facilitating the creation or adaptation of open standards such as MPIRT, that
29 can be incorporated in the TASP COE. The DoD sponsors intend for these standards to be fully
30 open commercial standards and not in any way unique to the military.
31

32 10.4.2 Army WSTAWG OE

33
34
35
36 **Discussion:** To be added later.
37

38 10.4.3 Air Force DII COE IPT

39
40
41 **Discussion:** To be added later.
42
43

44 10.5 Opportunities for MPI/RT

45
46 MPIRT embraces the many of the same goals as the COE efforts. Support for concepts such
47 as portability, and interoperability in a heterogeneous environment, etc. makes MPIRT a logical
48

philosophical fit. Since many of the RT COE efforts are still in the formative stages, and the probability of finding a place for MPI/RT within them is good.

For the DII COE, the prevailing expectation is that CORBA will become the distributed computing architecture of choice. The OMG's RT-CORBA specification could be available as early as mid-to-late 1998 with products to follow by late 1999. It would be valuable for MPIRT vendors to work with the group of potential RT-ORB vendors so that they can develop a specialized Inter-ORB protocol (IOP) that utilizes MPIRT.

In the JTA, POSIX.21 is the only real-time distributed process communications standard mentioned. Fortunately it is likely that MPIRT and POSIX.21 will both support the same underlying communication protocol. If that is the case then POSIX.21 could be used for internetworking with real-time requirements and MPIRT could be used for real-time system level networking with a reasonable expectation of interoperability.

Discussion: To be added later.

10.6 Obstacles for MPI/RT

MPIRT comes from a MPI pedigree. MPI is very widely recognized in the high performance computing problem space however it has less name recognition among the developers of DoD "front end" processing systems. It will be a challenge to make the developers of these systems aware of MPI/RT's strengths. **Discussion:** To be added later.

⊥ (latest)

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

Chapter 11

PacketWay and Real-Time Messaging

Current Status: No votes taken

Chapter 12

RSVP

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

Chapter 13

Data Reorganization

Chapter 14

IMPI/RT - Interoperability of MPI/RT Implementations

This chapter will define how MPI/RT 1.1 implementations may offer interoperability at different qualities of service, and with distinct protocols.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

Chapter 15

Appendix

Current Status: No votes taken.