Everything You Always Wanted To Know About MIG*

*But Were Afraid To Ask

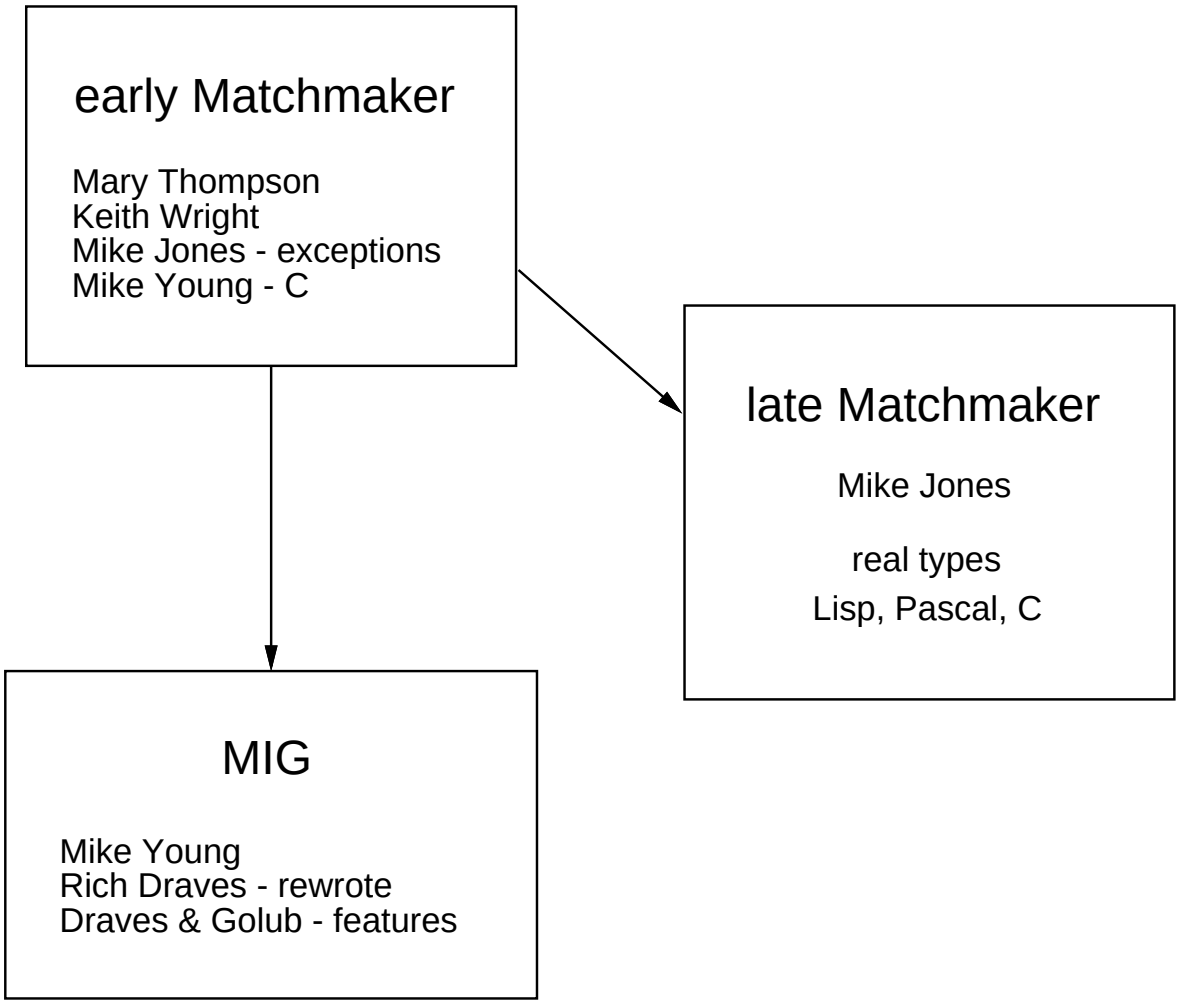
Richard P. Draves

rpd@cs.cmu.edu
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Mig

- Stub generator for Mach IPC.
- Pack, unpack, demultiplex messages.
- NOT transparent language-level RPCs.
- Doesn't hide IPC features.
- Assumes client/server model.

Mig History


early Matchmaker
Mary Thompson
Keith Wright
Mike Jones - exceptions
Mike Young - C
late Matchmaker
Mike Jones
real types
Lisp, Pascal, C
MIG
Mike Young
Rich Draves - rewrote
Draves & Golub - features

Mig Programming Model

- Clients call stubs

```
mach_port_t server_port;

kr = getbalance(server_port, &balance);
```

- Server loop processes requests

```
extern boolean_t bank_server();
mach_port_t server_port;
#define MAX_SIZE 512

(void) mach_msg_server(bank_server, MAX_SIZE,
                       server_port);

kern_return_t getbalance(server_port, balance)
    mach_port_t server_port;
    int *balance;        /* out */
{
    ...
}
```

Mig Example

```
/* file example.defs */

subsystem example 36270;

#include <mach/std_types.defs>

/* manipulate a bank account */

routine getbalance(
                server  : mach_port_t;
        out     balance : int);
```

Client call:

```
#include "example.h"

    mach_port_t server;
    int balance;
    kern_return_t kr;

    kr = getbalance(server, &balance);
```

Server function:

```
kern_return_t getbalance(server, balance)
    mach_port_t server;
    int *balance;               /* out */
{
    *balance = server_balance;
    return KERN_SUCCESS;
}
```

Type Definitions

```
/* from mach/std_types.defs */

type mach_port_t = MACH_MSG_TYPE_COPY_SEND;

type int = MACH_MSG_TYPE_INTEGER_32;

type kern_return_t = int;

/* pointer to out-of-line memory */

type pointer_t = ^array[] of MACH_MSG_TYPE_BYTE;

/* a 36-byte structure */

type mach_port_status_t = struct[9] of int;

/* C declarations for these types */

import <mach/std_types.h>;
```

Asynchronous Messages

```
/* file example.defs */

subsystem example 36270;

#include <mach/std_types.defs>

/* set bank account balance */

routine setbalance(
                server  : mach_port_t;
        in      balance : int);

/* set it with an asynchronous message */

simpleroutine asetbalance(
                server  : mach_port_t;
        in      balance : int);
```

- Client and server code looks like RPC.
- Message-ordering problems in multi-threaded servers.
- Control-flow worries.
- Not recommended for general use.

Asynchronous Messages

Client call:

```
#include "example.h"

    mach_port_t server;
    int balance;
    kern_return_t kr;

    kr = setbalance(server, balance);
    ...
    kr = asetbalance(server, balance);
```

Server function:

```
kern_return_t setbalance(server, balance)
    mach_port_t server;
    int balance;                /* in */
{
    server_balance = balance;
    return KERN_SUCCESS;
}

kern_return_t asetbalance(server, balance)
    mach_port_t server;
    int balance;                /* in */
{
    server_balance = balance;
    return KERN_SUCCESS;
}
```

Structure Arguments

```
/* file example.defs */

subsystem example 36270;

#include <mach/std_types.defs>
#include "example_types.defs"

routine structdemo(
                server  : mach_port_t;
        in      arg1    : small_struct_t;
        out     arg2    : small_struct_t;
        inout   arg3    : small_struct_t);

/* file example_types.defs */

#include <mach/std_types.defs>
type small_struct_t = struct[3] of int;
import "example_types.h";

/* file example_types.h */

#ifndef _EXAMPLE_TYPES_H_
#define _EXAMPLE_TYPES_H_

typedef struct small_struct {
        int one;
        int two[2];
} small_struct_t;

#endif  _EXAMPLE_TYPES_H_
```

Structure Arguments

Client call:

```
#include "example.h"

    mach_port_t server;
    small_struct_t arg1, arg2, arg3;
    kern_return_t kr;

    kr = structdemo(server, arg1, &arg2, &arg3);
```

Server function:

```
kern_return_t structdemo(server, arg1, arg2, arg3)
    mach_port_t server;
    small_struct_t arg1;        /* in */
    small_struct_t *arg2;       /* out */
    small_struct_t *arg3;       /* inout */
{
    *arg2 = *arg3;
    *arg3 = arg1;
    return KERN_SUCCESS;
}
```

Out-Of-Line Arguments

```
/* file example.defs */

subsystem example 36270;

#include <mach/std_types.defs>
#include "example_types.defs"

routine oolmemorydemo(
                server  : mach_port_t;
        in      arg1    : some_memory_t;
        out     arg2    : some_memory_t);

/* file example_types.defs */

/*
 * Variable-sized out-of-line memory:
 * Mig type name is some_memory_t,
 * C type name is vm_address_t
 */

type some_memory_t = ^array[] of MACH_MSG_TYPE_BYTE
        ctype: vm_address_t;

/* for C declaration of vm_address_t */

import <mach/machine/vm_types.h>;
```

Out-Of-Line Arguments

Client call:

```
#include "example.h"

    mach_port_t server;
    vm_address_t arg1, arg2;
    vm_size_t size1, size2;
    kern_return_t kr;

    (void) vm_allocate(mach_task_self(), &arg1, size1, TRUE);

    kr = oolmemorydemo(server, arg1, size1, &arg2, &size2);

    (void) vm_deallocate(mach_task_self(), arg1, size1);
    (void) vm_deallocate(mach_task_self(), arg2, size2);
```

Server function:

```
kern_return_t oolmemorydemo(server, arg1, size1, arg2, size2)
    mach_port_t server;
    vm_address_t arg1;          /* in */
    vm_size_t size1;            /* in */
    vm_address_t *arg2;         /* out */
    vm_size_t *size2;           /* out */
{
    *arg2 = arg1;
    *size2 = size1;
    return KERN_SUCCESS;
}
```

Deallocation

```
/* file example.defs */

subsystem example 36270;

#include <mach/std_types.defs>
#include "example_types.defs"

/* deallocate arg2 while sending the reply */

routine oolmemorydemo(
                server  : mach_port_t;
        in      arg1    : some_memory_t;
        out     arg2    : some_memory_t, dealloc);
```

Server function:

```
kern_return_t oolmemorydemo(server, arg1, size1, arg2, size2)
    mach_port_t server;
    vm_address_t arg1;          /* in */
    vm_size_t size1;            /* in */
    vm_address_t *arg2;         /* out, dealloc */
    vm_size_t *size2;           /* out */
{
    /*
     *  Out-of-line memory from the client in arg1 is
     *  returned to the client with automatic deallocation.
     */

    *arg2 = arg1;
    *size2 = size1;
    return KERN_SUCCESS;
}
```

Dynamic Deallocation

```
/* file example.defs */

subsystem example 36270;

#include <mach/std_types.defs>
#include "example_types.defs"

/* generate extra dealloc argument for arg1,
   always deallocate arg2 while sending the reply */

routine oolmemorydemo(
                server  : mach_port_t;
        in      arg1    : some_memory_t, dealloc[];
        out     arg2    : some_memory_t, dealloc);
```

Client call:

```
#include "example.h"

    mach_port_t server;
    vm_address_t arg1, arg2;
    vm_size_t size1, size2;
    kern_return_t kr;

    (void) vm_allocate(mach_task_self(), &arg1, size1, TRUE);

    /* deallocate arg1 while sending the request */
    kr = oolmemorydemo(server, arg1, size1, TRUE,
                       &arg2, &size2);

    (void) vm_deallocate(mach_task_self(), arg2, size2);
```

Port Arguments

- Normal port rights:
  - MACH_MSG_TYPE_COPY_SEND,
    MACH_MSG_TYPE_MAKE_SEND,
    MACH_MSG_TYPE_MOVE_SEND
  - MACH_MSG_TYPE_MAKE_SEND_ONCE,
    MACH_MSG_TYPE_MOVE_SEND_ONCE
  - MACH_MSG_TYPE_MOVE_RECEIVE
- "Polymorphic" port rights:
  - MACH_MSG_TYPE_PORT_SEND
  - MACH_MSG_TYPE_PORT_SEND_ONCE
  - MACH_MSG_TYPE_PORT_RECEIVE

```
subsystem example 36270;

routine portdemo(
                server  : mach_port_t;
        in      arg1    : mach_port_t =
                                MACH_MSG_TYPE_PORT_SEND;
        out     arg2    : mach_port_t =
                                MACH_MSG_TYPE_PORT_SEND_ONCE;
        in      arg3    : mach_port_t =
                                MACH_MSG_TYPE_MOVE_RECEIVE);
```

Port Arguments

Client call:

```
#include "example.h"

    mach_port_t server;
    mach_port_t arg1, arg2, arg3;
    kern_return_t kr;

    kr = portdemo(server, arg1, MACH_MSG_TYPE_MOVE_SEND,
                  &arg2, arg3);
```

Server function:

```
kern_return_t portdemo(server, arg1, arg2, arg2type, arg3)
    mach_port_t server;
    mach_port_t arg1;                   /* in */
    mach_port_t *arg2;                  /* out */
    mach_msg_type_name_t *arg2type;     /* out */
    mach_port_t arg3;                   /* in */
{
    /* dispose of the port rights we received */
    if (MACH_PORT_VALID(arg1))
        (void) mach_port_deallocate(mach_task_self(), arg1);
    if (MACH_PORT_VALID(arg3))
        (void) mach_port_mod_refs(mach_task_self(), arg3,
                                  MACH_PORT_RIGHT_RECEIVE, -1);
    /* return a send-once right to the server port */
    *arg2 = server;
    *arg2type = MACH_MSG_TYPE_MAKE_SEND_ONCE;
    return KERN_SUCCESS;
}
```

Pseudo-Arguments

```
subsystem example 36270;
#include <mach/std_types.defs>

ServerPrefix do_;

/* give the server the reply port and sequence number
   from the request message header */

routine pseudodemo(
                server  : mach_port_t;
        SReplyPort reply: mach_port_t =
                                MACH_MSG_TYPE_MAKE_SEND_ONCE;
        MsgSeqno seqno  : mach_port_seqno_t;
        in      realarg : int);
```

Client call:

```
    mach_port_t server;

    kr = pseudodemo(server, 10);
```

Server function:

```
kern_return_t do_pseudodemo(server, reply, seqno, realarg)
    mach_port_t server;
    mach_port_t reply;
    mach_port_seqno_t seqno;
    int realarg;
{
    printf("Received %d.\n", realarg);
    return KERN_SUCCESS;
}
```

Choosing Subsystem Ranges

```
subsystem <name> <msgid>;

/* request msg is <msgid>, reply is <msgid> + 100 */
routine first( ... );

skip; /* skip <msgid> + 1 */

/* request msg is <msgid> + 2, reply is <msgid> + 102 */
routine second( ... );
```

- For binary compatibility, MsgIDs and real in/out arguments should be preserved.
- MsgID ranges handled by a single server should not overlap.
- Helpful but not essential to keep interfaces under 100 routines.

Message Layout

```
/* file example.defs */

subsystem example 36270;

#include <mach/std_types.defs>

routine layoutdemo(
                server  : mach_port_t;
        in      arg1    : int;
        out     arg2    : int;
        inout   arg3    : int);

/* from file exampleUser.c */

typedef struct {
        mach_msg_header_t Head;
        mach_msg_type_t arg1Type;
        int arg1;
        mach_msg_type_t arg3Type;
        int arg3;
} Request;

typedef struct {
        mach_msg_header_t Head;
        mach_msg_type_t RetCodeType;
        kern_return_t RetCode;
        mach_msg_type_t arg2Type;
        int arg2;
        mach_msg_type_t arg3Type;
        int arg3;
} Reply;
```

Server Loop

`kern_return_t mach_msg_server(demux, max_size, rcv_name)`

Given the name of a port or port set, loops receiving request messages and sending replies. Uses Mig-generated demux functions to process request messages and generate replies. Needs maximum size for request and reply messages.

`boolean_t demux(request-msg, reply-msg)`

Returns TRUE if this demux function processed the request. In any case, must initialize the reply message.

Desired properties:

- Uses RPC form of `mach_msg` for performance.
- Doesn't leak port rights or memory regions.

Consume-On-Success

What happens to resources like port rights and out-of-line memory in request messages?

- The server function is responsible, *if* it returns success.

What happens to resources in reply messages?

- `mach_msg_server` is responsible. A server function should only produce outgoing resources when it returns success.

Return code from asynchronous server functions is important!

Mig Hints

- Asynchronous messages not recommended.
- InOut arguments not recommended.
- Put type declarations in a separate file.
- Use ServerPrefix.
- Use polymorphic port types.
- Use dynamic dealloc flag.

Mig Problems

- Confusion between interface and implementation.
- Only open-coded stubs, no descriptor-based stubs.
- Nits:
  - Message size information not exported.
  - Demultiplexing multiple interfaces.
  - MsgID assignment.