CHAIMS Notes

Louis Perrochon, Thursday, 19. December 1996

1 Contents

2 Things to do

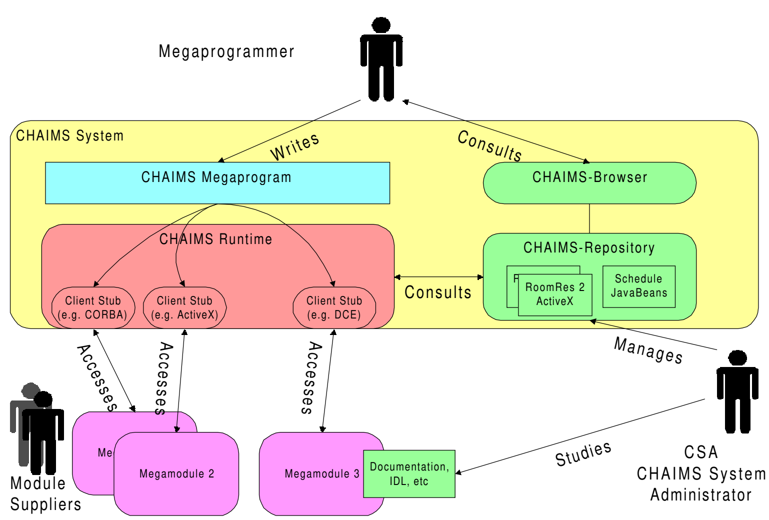
- Deliver a one-paragraph-report every first Monday of the month
- We need a *killer application*, the one application, that cannot be done without CHAIMS and can be done with CHAIMS very easily:
  - Implementing core processes across a heterogeneous IT structure?
- Gio's notes from the December 5 meeting:

  To be done by Catherine { C}, Gio { G}, Louis { L}, Mehul { M}, Pankaj { P}, Ron { R}, or { All}

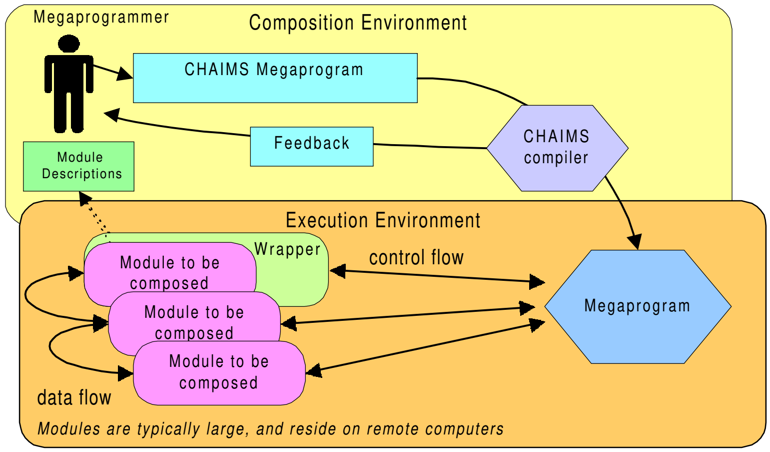
```
A: DCE, CORBA, OLE running examples {  C, M, P} due 1Feb97.
Means: executing client and server communicating.{  G}
     A.1  Investigate ActiveX {  P} due 1Feb97.
B  Definition and description of primitives used to implement A {  A}
C  Working Code <A>
```

D. paper { L,G,R} version 1 done, version for London 16Dec96,
E. High level architecture
E.1 CMU paper, London paper 18Dec96 { L,R,G}
E.2 update 1Jul97 { R,G}
F. Room reservation example documented, synchronized
G. Servers for at least 2 examples DCE, CORBA, OLE <3Feb97>
H. Defining features of client stubs { M}
I. Decide on programming language for client stubs 15Mar97
J. More examples <A> { G} 6Jan97
K. Mission statement { R} 6Jan97
L. Define and populate the tables in the client repository.
M Defining the syntax and semantics of CHAIMS in BNF <A>
M.1 V1. 3Mar97 <A>,
M.2 May97 <H>
M.3 Jun97 { R}
N. Running examples for dumb demo
O. Low level architecture,
O.1 Definition of Library interfaces <P1, P2,P3> 15Apr97
O.2 Other definitions
P Support libraries, for each statement, for creation and invocation.
P.1 Created as we go along, for each interface
P.1.C Library material for DCE { C} 5May96
P.1.P Library material for OLE { P} 5May96
P.1.M Library material for CORBA { M} 5May96
P.2 Moving library contents architecture specs <O.1, P.1> 1Jun97
Q. Dumb demo, equivalent to <N>
S. Progress report update.{ C} 6Jan97, 3Feb97, etc first Monday of the month
S.1 After final some Monday meetings { C} 16Dec06, 3pm; others
S.2 Get room.{ C} 10Jan97
S.3 Topic participation in DCE, Active-X, CORBA, etc. as they occur { all}
T Automatic repository management LongTerm
U. Major demo { all} 15Sep97
V. Review Meeting 2Apr97 { R organizer, C, G, L, M, P}

3 Architecture



Components of the CHAIMS megaprogramming paradigm.



The CHAIMS Megaprogramming Process and Information Flow.

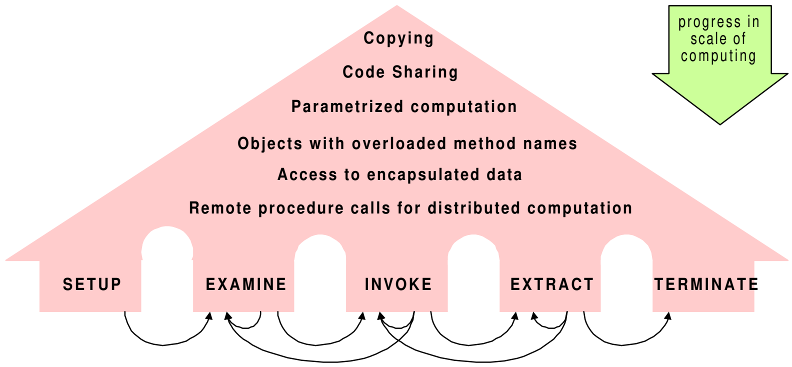
4 CHAIMS Sample Program

```
MEGAPROGRAM Arrange Meeting;
            (* At least the core of it *)
DECLARE
  MEGAMODULE (* The MMDL-definitions for these must come from the repository *)
    RR      : GatesRooms;
    F       : PaloAltoPizzaHut; (* Not yet needed *)
  VAR
    hRR     : HANDLE;
    room    : STRING;
    stime   : DATE;
    seats   : INTEGER;
    windows: BOOLEAN;

BEGIN
  INPUT(time,people);
            (* Calling input service, don't care about handle *)
  IMPORT SETUP RR;
            (* Not really required here, MProrammer allready knows, what he/she wants *)
  SETUP hRR:=RR.GetRoom(time:=stime,seats:=people);
            (* Give it the fixed parameters *)
  INVOKE hRR(windows:=TRUE);
            (* Pass on variable parameters*)
  REPEAT
    ESTIMATE wait := hRR();
            (* ESTIMATE returns 0 if job is finished *)
  UNTIL wait=0; (* while *)
  EXTRACT hRR(roomno:=room);
  OUTPUT(roomno);
END
```

5 CHAIMS Reserved Words

The "big five" in CHAIMS are: SETUP. ESTIMATE, INVOKE, EXTRACT, TERMINATE. They replace the traditional call statement. The other statements are necessary programming language add-ons.



Splitting up the call statement

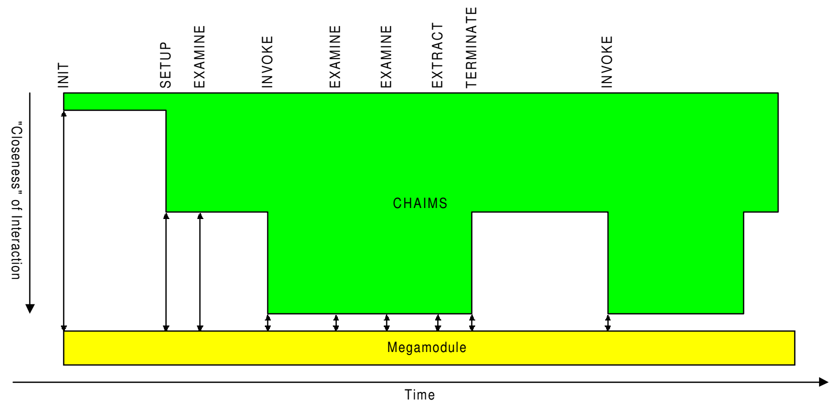
Please note, that the big five are different from the big three in the CACM Paper (SUPPLY, INVOKE, EXTRACT) or the big five in the Proposal: *SETUP, ESTIMATE, INVOKE, EXAMINE, EXTRACT. Italics refer to the original proposal or the CACM paper.*

Current Statements	*Proposal*	*CACM Paper*	Semantics
not yet, repository		*INSPECT*	
not yet, use repository		*IMPORT SUPPLY*	get SUPPLY data structures
SETUP	***SETUP***	***SUPPLY***	provide global arguments to MM
no more, use EXAMINE	***ESTIMATE***	*ESTIMATE*	check on the progress of the MM
INVOKE	***INVOKE***	***INVOKE***	provide additional arguments start processing at MM
EXAMINE	***EXAMINE***	*EXAMINE*	provide a performance estimate
not yet, use repository		*IMPORT EXTRACT*	Get EXTRACT data structures
EXTRACT	***EXTRACT***	***EXTRACT***	receive results from MM
no more, use SETUP		*CONSTRAIN*	resetting of megamodule parameters
not yet		*LIMIT*	limits MM in time or cost
TERMINATE			

Overview over the CHAIMS statements

5.1 SETUP (*SUPPLY, SETUP*)

- *The **Set-Up** substatement supplies the bulk of the parameters, and allows the module to prepare itself for efficient execution For instance, a travel planning module can obtain from its databases data on the likely regions that will be involved in the planning activity. Optimization will attempt to move SET-UP statements as much as possible in advance of module invocation.*
- *SUPPLY provides global arguments from a megaprogram to the megamodule.*
- How many handles? Proposal: Every SETUP gives you exactly one handle back. The mega-programmer has to take care not to do stupid things with it. Megamodules must support multiple handles. One megaprogram can have several handles open at the same time (even several handles to the same megamodule at the same time).
- How to match arguments provided by mega-program with arguments expected by mega-module? MMDL (Mega-Module Definition Language)



5.2 INVOKE (*INVOKE*)

INVOKE actually starts the process. In earlier texts, it didn't have arguments, now it can still contain the arguments that change for different INVOKEs

- *INVOKE causes the megamodule to actually process these arguments and prepare results.*
- *The Invoke substatement initiates the actual module execution in parallel, and returns to the mega-program. It can be repeatedly executed if major set-up parameters do not change.*

5.3 EXAMINE (*ESTIMATE, EXAMINE, INSPECT*)

Returns 0, when megamodule has finished it's job, otherwise time megamodule expects to run (performance estimate). If the megamodule has no idea on how long it will take it, EXAMINE should return some reasonable polling interval.
Several statements are used for inspecting and controlling megamodule execution. The most important are:

- *EXAMINE to allow the megaprogram to check on the progress being made in obtaining results.*
- *ESTIMATE, to cause the megamodule to provide a performance estimate, so that the megaprogram can judge whether invocation is likely to be effective.*
- *The Estimate substatement obtains meta-data from the modules permitting global mega-program optimization.*
- *The Inspect substatement allows verification of computation progress and/or completion of execution within the module.*
- *The parameters for ESTIMATE and INSPECT are expected to be fixed. ESTIMATEs will include time, cost, and volume of results. They will be reported with a certainty factor, to enable alternatives in scheduling the modules. For some soft real-time environments it will be preferable to have a high-certainty of module performance, even if the estimate of average execution time is higher.*
- *A megamodule maintains a number of state variables. The EXAMINE statement accesses the most critical variable, namely STATE. Reasonable values for STATE variables in megamodules are: DONE, IN_PROGRESS, IDLE, IN_ERROR, or WAITING. For instance, STATE=DONE indicates that the megaprogram can safely extract results from the megamodule. Other variables that may be provided by a megamodule include, for instance, measures of nearness to a solution for an iterative program, the number of choices being considered for a search-type megamodule, and so on. These variables may be accessed at any time for a megamodule that has been invoked. The operation of EXAMINE is synchronous with respect to the megaprogram.*
- *For megaprogramming optimization, estimates ofexecution cost of megamodules may be needed [Wiederhold 1992a; Wiederhold 1992b] ; because of the encapsulation provided by megamodules, such estimation can only be provided from inside the megamodule. The result of execution of an statement is a simple estimate of execution cost (time, dollars, result volume), with the degree of certainty of that estimate (0..1).*

5.4 EXTRACT (*EXTRACT*)

- How to match arguments provided by mega-program with arguments expected by mega-module?
- *The Extract substatement obtains final or intermediate results, as needed, depending on the information returned by the Inspect substatement.*
- *The* data structures *supported by a mega programming declaration enable providing input and output parameters, of arbitrary complexity, between a megaprogram and mega-modules; data structures are defined in the context of mega-modules and are accessible by mega-programs. The MPL type system should be powerful enough to support complex data*

structures; therefore, it should include generalized type constructors, including records, sets, multisets, lists, sequences, updatable arrays, sparse arrays, etc.

- *Data structures of mega-modules are designed independently; they have arbitrary internal type systems and are defined asynchronously from megaprogram definitions. Mega-modules also provide self-describing* `EXPORT` *data structures that should be compatible with the MPL type system. However, a megamodule need not to be able to handle all of the MPL type system, its capability must only be adequate to support all of its* `EXPORT` *data structures.*
- *The technology we propose to borrow and adapt here derives from database schemas, the description of database contents and formats that permits many transaction programs to share a database and obtain data selectively and associatively. Accordingly, data-structures of mega-modules are declared in terms of a schema definition, but significant extension of database schema concepts will be required, since:*
  1. *The data structures permissible in general mega-programs are more complex than the simple relations seen in most DBMSs [CCZLL:90].*
  2. *Enough information must be provided to permit transduction of information between data structures that differ. Transduction occurs when data extracted from a megamodule are submitted to a successor megamodule: optimizations, described in Section 4, combine consecutive* `EXTRACT` *and* `SUPPLY` *operations to generate a direct flow of information among mega-modules.*

5.5 TERMINATE (new)

We may need two of these, one to terminate an INVOKE, one to terminate a SETUP.

5.6 DECLARE (new)

Not very important in terms of concept. DECLARE is used to declare variables etc.

5.7 INSPEC

- *Inspection of a megamodule provides information about its ontology and specific parameters; the result of an {\tt INSPECT} operation does not depend on the current state of the module being inspected. Inspection could yield documentation, version information, or formal descriptions of methods and scope of applicability.*

5.8 IMPORT SUPPLY/EXTRACT

- *The IMPORT SUPPLY statement causes the megamodule to export the supply data structures. Standard names are used to describe data structures, formats, extents, and sizes, according to MPL types. %We envisage a syntax and %expressive power similar to Ada declarations or database schemas. Elements of data structures may have default values which can be retrieved by an INSPECT command.*
- *The IMPORT EXTRACT statement, similar to the IMPORT SUPPLY, causes the megamodule to export the extract data structure.*

5.9 CONSTRAIN

- *The CONSTRAIN statement permits the resetting, by a megaprogram, of internal megamodule parameters obtained by inspection.*

5.10 LIMIT

- *The LIMIT function constrains megamodule execution in terms of time or cost (as a surrogate for computational time spent). Not all megamodules will be able to accommodate such a request, but if they can, certain megaprogram tasks would be greatly simplified.*

6 Megamodule Definition Language (MDDL)

We need some kind of a Mega-Module Definition Language (MMDL), analogue to the DDL. Every Mmdescription needs at least the following information

Information visible to the megaprogrammer using the browser

- Name of megamodule
- Funktionality. Natural language text?

- Name of arguments
  Here we only have the CHAIMS datatypes

```
ARGUMENTS GatesRoom;
  room: STRING; (* A CHAIMS string. Official name of room, e.g. Gates 459 *)
  time: DATE;   (* A CHAIMS date including year, month, day, hour, minute, second *)
  seats: INTEGER;  (* A CHAIMS Integer *)
  windows: BOOLEAN;
```

- Dependencies between arguments:
  If you want to get output X, then you must supply Input Y,Z and can supply P,Q

```
DEPENDENCIES GatesRoom;
  GetRoom(out: room; req: time, seats; opt: windows)

FEATURES GatesRoom;
  ESTIMATE: STANDARD;
  SETUP. STANDARD;
  INSPECT NONE;
```

As long as the above information doesn't change, the CHAIMS megaprogram will be working. But there is more information needed by the CHAIMS runtime This information not visible to the megaprogrammer. (For the first implementations this will be coded by hand anyway.)

- Details of invocation. What is needed here?
  Which protocols, etc.
- Name of all arguments including default values. Here we need the data types of the megamodule.

7 More Stuff

7.1 General

CHAIMS does not make legacy software running (or does not try to integrate legacy software). It's a new paradigm. SETUP is actually sending data over the net, so legacy code that cannot deal with this must be wrapped.

- CHAIMS shall build infrastructure for other peoples projects.

7.2 Notions

Maybe use *service* instead of *module* because you normally don't have control over the behavior and interface of a service, but you have usually control over a module.

7.3 On non/blocking and a/synchronous

What is the difference between blocking/non-blocking and synchronous/asynchronous?

- Blocking/non-blocking affects the flow of the program issuing the call. It only influences the one program, that does the call.
- Synchronous/Asynchronous affects communication between two programs. Both programs are influenced by it.

The two concepts are orthogonal, i.e. a call can be any combination of the two features. However, normally blocking calls come together with synchronous communication and non-blocking calls come with asynchronous communication.

7.4 Economics

CHAIMS must be more cost effective, at least once it is commercially available.

- You can solve the integration problem manually with phone calls to the right people, each of them sitting on a computer and having access to his megamodule. This is sub-optimal, because human interaction is involved and you may still not find an optimal integration.
  This means, you have no additional cost for integration, however the cost of doing business may be higher because you have more manual work and you may not find optimal solutions.
- You can try to define a standard, that solves all you integration problems. However defining standards is a tedious task.
  This means, you will probably have to adapt all your system to this new standard.

- You can also solve the integration problem on a case by case basis by programming the "glue" manually in whatever suits the case best (CORBA, OLE, low level, etc.). Wrappers may be of some help, if you have little heterogeneity and find the necessary wrappers (i.e. CORBA-to-OLE, etc.).
  This means, you have probably high cost for the integration. The resulting program will be big and expensive to maintain too.
- You can purchase CHAIMS and just develop a short CHAIMS-Language program. (If your systems are not CHAIMS compatible (i.e. not CORBA, OLE etc. compliant), you have to use existing wrappers or write wrappers yourself).
  This means, that you have probably just a short CHAIMS program to maintain.

8 Appendix

8.1 CORBA/DCOM

8.1.1 Similarities

Separation of interfaces and implementation (IDL), Location Transparency, Inheritance + polymorphism, Interface repository, Dynamic Method invocation., Server registration and activation.
Many people seem to think that OLE and CORBA compete, and that Microsoft is firmly against CORBA. They actually interoperate and Microsoft has submitted an OLE Interoperability Specification to the OMG (the group the sets the CORBA standard). There's no reason why a company can't develop their objects on their prefered development platform (pick one) and use them on their prefered client platform (pick one).

8.1.2 Differences

8.1.2.1 CORBA

more than 20 platforms, IDL mapping to C, C++, Smalltalk and ADA95, Java, COBOL. IDL support both static and dynamic method invocation, gateways to DCE, DCOM/COM/OLE, adopted languages mapping make it a better fit for integration with, legacy systems., proven track record since 1992 (more mature)

8.1.2.2 DCOM

Windows NT 4.0 (Software AG and Digital doing UNIX port). MIDL mapping to C, C++, MIDL different if using DII. Complicated to use, slow, contains too much code, and involves too many APIs. More complex: developer has to deal with reference count, interface negotiation, class factories. More complex IDL (developer has to deal with pointers and parameters passing mechanism). Available for NT December 1996, Mac 1Q97.

DCOM provides a consistent programming model for BOTH clients and servers. CORBA only defines a programming model for the client. There is no standard programming model for servers; each ORB has it's own.

This summary is mostly from [Roy, 1996 #163]

8.2 OLE (pj)

- GET CLSID (ClassID) for application. Can be avoided and hardcoded, if already known.
- Launch the automation server Creation of the instance of the object.
- Allocate a VARIANTARG data structure for each argument.
  Are complex arguments possible?
- Get a pointer to IUnknown interface of object. IUnkown->Queryinterface tells all the interfaces supported.
- Use IUnkown->QueryInterface to get IDispatch. IDispatch describes the query interface.
- IDispatch->GetIDofName() gives ID of the methods and propoerties you need to use. Avoided if already known, then it can be hardcoded.
- INVOKE method/property. Result comes back in a VARIANTARG structure, that has to be parsed.
- Free VARIANTARG datastructure.

```
// ***********************************************
//
```

```
// OLE Client for invoking Room Reservation service
//
// Date : 11/5/96
//
// **********************************************

// Variable declarations

CLSID clsid;                    // class ID
HRESULT hresult;                // diagnosis of call
IUnkown FAR *punk = NULL        // pointer to IUnkown interface
IDispatch FAR *pIDispatch;      // pointer to IDispatch interface
char FAR * szMethodName = "getroom"
                                // The name of the method to be used
DISPID dispid;
DISPARAMS disp;

VARIANTARG varg[2];             // 2 argument example
VARIANTARG vargResult;          // result returned by the service

//  Name -> CLSID translation
//  Not reqd. if CLSID already known
//  then it can be hardcoded

hresult = CLSIDFromProgId ("RR.aplication", &clsid);
if ( FAILED(hresult)) // do error stuff ;

// Launch the automation server
// and also get a pointer to the IUnkown interface

hresult = CoCreate_Instance (clsid, NULL, CLSCTX_SERVER, IID_IUnkown,
                             (void FAR *FAR *)&punk);
if ( FAILED(hresult)) // do error stuff ;

// Get a pointer to the IDispatch interface thru the queryinterface

hresult = punk->QueryInterface (IID_IDispatch, (void FAR *FAR *)&pIDispatch);

if ( FAILED(hresult)) // do error stuff ;

// VariantArgs - Initialize them

VariantInit (&varg[0]);
VariantInit (&varg[1]);

VariantInit (&vargResult);

// Initialize parameters to the proper values
// Perhaps, this requires user input

// ************************************
// Code for initializing values
// ************************************

disp.rgVarg = varg;
disp.rgdispidNamedArgs = NULL; // Assume no named parameters
disp.cArgs = 2;
disp.cNamedArgs = NULL;

// get the ID of the required method
// can be avoided if the dispid is known in advance
// then it can be hardcoded

hresult = pDispatch->GetIDsofNames (IID_NULL, &szMethodName, 1, lcid,
                                    &dispid);

if ( FAILED(hresult))
{
```

```
VariantClear(&varg[0]);
VariantClear(&varg[1]);

// Error Stuff

}

// Finally invoke !!!

hresult = pIDispatch->Invoke (dispid, IID_NULL, lcid, DISPATCH_METHOD,
                              &disp, &vargResult, NULL, NULL);

if ( FAILED(hresult))
{

VariantClear(&varg[0]);
VariantClear(&varg[1]);

VariantClear(&vargResult);

// Error Stuff

}

// ******************************
// Code to use vargResult
// ******************************

// Clear arguments and results

VariantClear (&vargResult);
VariantClear (&varg[0]);
VariantClear (&varg[1]);
```

8.3 CORBA (mb)

Each object is associated with a server (which is a mapping from a set of objects to a name). The Interface Description Language (IDL) provides transparency of details of execution of the objects by providing a high level interface (like a C++ class) so that clients can use this interface to call server objects whether server objects are running on the same host machine or a remote machine. Also the client code and the server object code can be written in different languages. This is the distributed computing environment which CORBA implementations are supposed to provide if they need to be certified by OMG - Orbix is one such implementation studied here.

8.3.1 Writing CORBA applications

The are 6 steps in writing a corba application:

8.3.1.1 Specifying an IDL

An interface in IDL is specified by operations and attributes. This is just a text file although it looks like a C++ class interface. eg. The room reservation IDL could be written as

```
// room_res.idl

// IDL definition for room_res

interface room_res {
   // read-write attributes which can be set & retrieved
   // by the client
         attribute string time ;
         attribute short people ;

         readonly attribute short room_no ;
         // like an output only whose value can only be read by the client

         // IDL operations
```

```
        // call the room-reservation function
        void reserve (in string time, in short people) ;
};
```

You can have read only attributes (specified as "readonly attribute") & also read-write attributes (specified as simply "attribute"). The valid types include float, short, long, char, boolean, octet (8-bit quantity guaranteed not to undergo any conversion during transmission) & any (can express any IDL type - like (void *) in C).
Note that the IDL only specifies operations and not state, meaning that the only public operations allowed by the client are functions, and not data. Hence, the data elements are converted to set and/or get functions depending whether they are read-write or readonly. All the resulting methods are appended with the environment parameter (which has a default value) for debugging purposes, as shown later in the client application.

8.3.1.2 Compilation

Compiling the above the IDL using "idl -B -S rooRes.idl" we get the following files :

```
room.ic         room.ih         roomS.C
room.hh         room.idl        roomC.C
```

-B generates BOAImpl classes (explained later)
-S produce the initial skeleton files for the object implementation (eg room.ic & room.ih)

There are a lot of operations and classes defined in these files but only a few are shown below for understanding :Asssume that for now both the server implementation of the room object and the client invocations are written in C++. Every object in CORBA is derived from the base CORBA object. Also, there is dynamic binding, and to avoid ambiguity due to multiple inheritance, all inheritance is virtual.

FILE room.hh
Header file to be included into the room implementation and all clients. Defines the C++ view of the IDL.

```
class room_res: public virtual CORBA::Object {
public:
    room_res (char *IT_OR);
    room_res () : CORBA::Object (1) {}
    room_res* _duplicate(
            CORBA::Environment &IT_env=CORBA::default_environment) {
       CORBA::Object::_duplicate (IT_env); return this; }

   // Overloading _bind (client binding to a server object)
   static room_res* _bind (const char* IT_markerServer, const char* host,
     const CORBA::Context &IT_c,
     CORBA::Environment &IT_env=CORBA::default_environment);
   static room_res* _bind (CORBA::Environment &IT_env);
   static room_res* _bind (const char* IT_markerServer=NULL, const char* host=NULL,
                 CORBA::Environment &IT_env=CORBA::default_environment);
    static room_res* _narrow (CORBA::Object* , CORBA::Environment
&IT_env=CORBA::default_environment);

   // SET/GET Functions
    virtual void time (const char * time, CORBA::Environment
&IT_env=CORBA::default_environment)\
;
    virtual char * time (CORBA::Environment &IT_env=CORBA::default_environment);
    virtual void people (short people, CORBA::Environment &IT_env=CORBA::default_environment) ;

    virtual short people (CORBA::Environment &IT_env=CORBA::default_environment);


   // Room Reservation Invocation

    virtual short room_no (CORBA::Environment &IT_env=CORBA::default_environment);
```

```
        virtual void reserve (const char * time, short people, CORBA::Environment
&IT_env=CORBA::default_environment);

}
```

FILE room.ic
a template file for the room implementation.

```
void room_res==:: time (const char * time, CORBA::Environment &IT_env)\
{
}

char * room_res==::  time (CORBA::Environment &IT_env) {
}

void room_res==:: people (short people, CORBA::Environment &IT_env)\
 {
}

short room_res==::  people (CORBA::Environment &IT_env) {
}

short room_res==::  room_no (CORBA::Environment &IT_env) {
}

void room_res==:: reserve (const char * time, short people, CORBA::Environment &
IT_env) {
}
```

FILE room.ic:
Header file for the functions which will be implemented by the server - note that the server implementation class room_res== can have extra operations but the ones specified in room.ic are mandatory to be implemented.

```
ifndef room_ih
#define room_ih

#include "room.hh"

class room_res== {
public:
    virtual void time (const char * time, CORBA::Environment &IT_env=CORBA::defa
ult_environment)\
;
    virtual char * time (CORBA::Environment &IT_env=CORBA::default_environment);
    virtual void people (short people, CORBA::Environment &IT_env=CORBA::default
_environment)\
;
    virtual short people (CORBA::Environment &IT_env=CORBA::default_environment)
;
    virtual short room_no (CORBA::Environment &IT_env=CORBA::default_environment
);
        virtual void reserve (const char * time, short people, CORBA::Environmen
t &IT_env=CORBA::default_environment);
};

#endif
```

FILE roomC.C
source file to be compiled and included into the clients. It will have a implementations of network connectivity i.e. establish connection with server and return results as is done by the CORBA::Request operation.

```
shrt room_res::people (CORBA::Environment &IT_env) {

    if (IT_env || m_isNull) return 0;

    CORBA::Request IT_r (this, "_get_people",IT_env,1,0);
    IT_r.invoke(CORBA::Flags(0), IT_env);
```

```
    if (!IT_r.isException (IT_env)) {
        short people;
        IT_r >> people;
        IT_r.checkEnv (IT_env);
        return people;
    }
    if (IT_r.exceptionKind () == CORBA::StExcep:: SYSTEM_EXCEPTION)
        IT_r.checkEnv (IT_env);
    return 0;
}
```

FILE roomS.C
source file to be compiled and included into the server of the room. It has similar functions as roomC.C. It includes roomC.C and has some other operations also.

8.3.1.3 CREATING A SERVER PROGRAM :

File room_res.c
Create a room object, indicate to Orbix that the server's initialisation has completed (by calling impl_is_ready), and wait for any client connections.

```
#include "grid.ih"

main () {
   room_res Myroom ;

   CORBA:: Orbix.impl_is_ready("roomRes");
}
```

Compiling the above program and linking with roomS.C, room.ih, room.hh, we get the executable "server". The server can be manually invoked from the command prompt but will kill itself if it does not receive client connection within a certain time.

8.3.1.4 REGISTERING THE SERVER

Its done using "putit roomRes <server pathname>" so that server will run automatically when a client calls any operation on the room_res object. The CORBA implementation (here Orbix) needs to be running first. Doing

```
   putit -h hostname roomRes <server pathname>
```

will run the server on a different machine "hostname" from the one in which it has been compiled - this can be used to distribute the client and the server.

8.3.1.5 WRITING A CLIENT

This involves using a pointer to the IDL named object (here room_res), binding to a server, and invoking the required operations. E.g,

```
#include "room.hh"
#include "stream.h"

main ( ) {
   room_res * p ; char time[MAXSIZE] ; int people ;
   p = room_res::_bind(":roomRes") ;
   cout << "input the time & people " << endl ;
   cin >> time >> people ;
   p->time(time) ; p->people (people) ; // SET calls

   cout << "calling the room reservation function" ;
   p->room_res(p->time(), p->people()) ; // GET calls

   cout << " the room reserved is " << p->room_no() << endl ;
   p->_release() ;
}
```

Note that the client is not concerned with the implementation of roomi.C, which makes it somewhat independent of the server.

8.3.1.6 RUNNING THE CLIENT

After the client is compiled, it can be run as "room_client <hostname>", where room_client is the client executable, and hostname defaults to current host. This is the last step.

8.3.2 CORBA Concepts/functions

Below are description of some CORBA concepts/functions.

8.3.2.1 RELEASE

The C++ delete should not be called, as CORBA needs to do additional cleaning up, hence release is called.

8.3.2.2 BIND

The server name is mandatory, if the object name is unspecified, any room_res object is chosen in the roomRes server as registered by the putit command in the Implementation Repository. In the above example, the server name is looked up in the local host machine, or remote machine, if the locator service is properly configured.
Full declaration of Bind is

```
_bind (char * markerserver, char * host, CORBA::Context &,
    CORBA::Enviroment &)
```

Marker server is "marker : server" where marker is the object name associated with the object. One way of obtaining this is for eg. Calling room_res * MyRoom("room_name") and in the constructor, calling room_resBOAImpl(marker). BOA stands for Base Object Adapter.

BOA CLASS

For every C++ class associated with the IDL interface (eg room_res), there is a room_resBOAImpl which is derived from this class, and from which the implementation class room_res== is derived.

OBJECT REFERENCE / NAMING :

The name of an object (its unique object reference) contains its marker (unique name within its server), server name (sometimes called the implementation name), and host name of the server.

CONTEXT OBJECTS

A client maintains one or more CORBA Context objects, which are mapping from string names to string values. An operation can specify that it will requires as input, context values. Eg.

```
interface room_res {
    short room_operation (in string time,
        context ("sys_*", "people")) ;
}
```

The identifier sys_* would match context entries such as sys_accuracy, sys_outputsize, etc.

8.3.2.3 MODULES

An IDL module provides a way to group interfaces and other IDL type definitions into logical units, causing scoping just like in Pascal. Eg

```
module reservation_services {

    typedef short reservation_class ;

    interface room_res {
        // details

    }

    interface plane_res {
```

```
    }
}
```

Hence the scoped name of room_res is reservation_services::room_res.

8.3.2.4 ERROR HANDLING

Orbix provides exceptional handling macros like TRY, CATCHANY & ENDTRY which work as follows. E.g.

```
TRY {
   p = room_res::_bind(":roomRes", "IT_X") ;
} CATCHANY {
   cerr << bind failed with exception << IT_X << endl ;
}
ENDTRY
```

TRY defines variable IT_X which is passed to all function calls, whose value is tested by CATCHANY to determine if any exception has been raised.

8.3.2.5 LAUNCHING SERVERS

- shared activation mode:
  all objects under a server are managed by the same process. whenever a client invokes a operation onany of these objects, the server is launched, if its not already running.
- unshared activation mode:
  individual objects of a server are registered, and one process created for each invocation of the object. Each object can be registered with a different executable file name, or a group of them can share executable file.
- per-method call activation mode:
  individual operations are registered with Implementation Repository. a process is created to handle each individual operation and is destroyed once the operation is completed.

For each of these modes, a server can also be launched in per-client, per-client-process or multiple-client activation modes :

- per-client:
  activation of same server result in a separate process for each such end-user.
- per-client-process:
  activation of same server result in a separate process for each client process.
- multiple-client activation mode:
  different end-users share the same invocation of the server.

The above description is a summary of some of the basic features of CORBA. Advanced features include filtering, dynamic invocation interface, smart proxies, and configuration. These will be explained later.

8.4 Java/JavaBeans (ct)

A Bean is a component (= piece of reusable software), that can be visually manipulated in builder tools (like Java Workshop, Java Studio, Symantec Café). It needs Introspection, Customization, Events and Persistance.

- Introspection means, that there is no separate IDL. All the information the builder tool needs is in the code, the builder tools needs to understand Java
- JavaBeans are not useful for CHAIMS at the moment, because they are not distributed. A JavaBean is loaded into the local environment, it cannot be executed on the server.

9 References

[Agarwal, Keller et al. 1995] Agarwal, S., A. M. Keller, K. Saraswat and G. Wiederhold: "Flexible Relation: An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases". *IEEE Data Engineering Conference,* Taipei, Taiwan. 1995.

[Allen and Garlan 1994] Allen, R. and D. Garlan: "Beyond Definition/Use: Architectural Interconnection". *Workshop on Interface Definition Language*. 1994.

[Andrews 1993] Andrews, T.: "Designing Linguistic Interfaces to an Object Database or What do C++, SQL, and Hell have in Common?". *Database Programming (DBPL-4)*. C. Beeri, A. Ohori and D. E. Shasha, Springer-Verlag: 3-8. 1993.

[Arnold 1993] Arnold, R. S.: "Software Reengineering", IEEE Press. 1993.

[Bach 1994] Bach, J.: "Process Evolution in a Mad World", Borland Int, Scotts Valley CA. 1994.

[Bell and Sharon 1994] Bell, R. and D. Sharon: “Tools to Engineer New Technologies into Applications.” *IEEE Software* 12(2): 11-16. 1994.

[Birrell and Nelson 1984] Birrell, A. D. and B. J. Nelson: “Implementing Remote Procedure Calls.” *ACM Transactions on Computer Systems* 2(1): 39-59. 1984.

[Blum 1992] Blum, B. I.: "Software Engineering - A Holistic View". New York, NY, Oxford University Press. 1992.

[Boehm and Scherlis 1992] Boehm, B. and W. Scherlis: "Megaprogramming (preliminary version)". *DARPA SW Conference*, Meridien. 1992.

[Booch 1987] Booch, G.: "Software Components with Ada", Benjamin Cummings. 1987.

[Brockschmidt 1996] Brockschmidt, K.: "Inside OLE 2", Microsoft Press. 1996.

[Chappell 1996] Chappell, D.: "Understanding ActiveX (TM) and OLE". 1996. <http://www.microsoft.com/mspress/text/books/abs/1-216-5b.htm>.

[Charette 1979] Charette, R. H.: "Software Engineering and Risk Analysis Management", McGraw-Hill. 1979.

[Courtney, Janssen et al. 1994] Courtney, A., W. Janssen, D. Severson, M. Spreitzer, et al.: "Inter-Language Unification, rel.1.5", Xerox PARC. 1994. <http://www.omg.org/>.

[Delis and Roussopoulos 1992] Delis, A. and N. Roussopoulos: "Performance and Scalability of Client-Server Database Architectures". *18th Int. Conf. on Very Large Data Bases,* Vancouver, British Columbia, Canada. 1992.

[Dertouzos 1986] Dertouzos, M. L.: “The Multiprocessor Revolution: Harnessing Computer Together.” *Technology Review* 89(2): 44-57. 1986.

[Feldman 1979] Feldman, J. A.: “High Level Programming for Distributed Computing.” *Communications of the ACM* 22(6): 353-368. 1979.

[Feldman, Low et al. 1978] Feldman, J. A., J. A. Low and P. D. Rovner: "Programming Distributed Systems". *Annual Conference of the ACM,* Washington, D.C. 1978.

[Feldman, Low et al. 1972] Feldman, J. A., J. R. Low, D. C. Swinehart and R. H. Taylor: "Recent Developments in SAIL". *Fall Joint Computer Conference (AFIPS)*. 1972.

[Finin, Fritson et al. 1994] Finin, T., R. Fritson, D. McKay and R. McEntire: "KQML as an Agent Communication Language". *CIKM 3*, ACM. 1994.

[Frakes and Terry 1996] Frakes, W. and C. Terry: “Software Reuse: Metrics and Models.” *ACM Computing Surveys* 28(2): 415-435. 1996.

[Garlan, Nierstrasz et al. 1995] Garlan, D., O. Nierstrasz, T. D. Meijler, G. Wiederhold, et al.: “Research Directions in Software Engineering.” *ACM Comp.Surveys* 27(2): 256-276. 1995.

[Grady and Caswell 1987] Grady, R. and D. Caswell: "Software Metrics: Establishing a Company-Wide Program", Prentice-Hall. 1987.

[IEEE 1993] IEEE: "IEEE Standard 1278: Protocols for Distributed Simulation Applications: Entity Information and Interaction", IEEE Computer Society. 1993.

[ISO 1991] ISO: "ISO Remote Procedure Call Specification ISO/IEC CD 11578 N6561", ISO/IEC. 1991.

[Jackson 1982] Jackson, M. A.: "System Development", Prentice Hall. 1982.

[Jahanian and Mok 1994] Jahanian, F. and A. Mok: “Modechart: A Specification Language for Real-Time Systems.” *IEEE Transactions on Software Engineering* 20(12): 933-947. 1994.

[JavaSoft 1996] JavaSoft: "Java(tm) Beans: A Component Architecture for Java", Sun Microsystems. 1996. <http://splash.javasoft.com/beans/WhitePaper.html>.

[Jones 1994] Jones, C.: "Assessment and Control of Software Risks", Yourdon Press. 1994.

[Kapoor 1991] Kapoor, M.: "A Software Design Manifesto: Time for a Change". *Dr. Dobbs*. 1991.

[Libicki 1995] Libicki, M. C.: "Information Technology Standards", Digital Press. 1995.

[Lientz and Swanson 1980] Lientz, B. P. and E. B. Swanson: "Software Maintenance Management". Reading, MA, Addison-Wesley. 1980.

[Lockhart 1995] Lockhart, H. W., Jr.: "OSF DCE, A Guide to Developing Distributed Applications", McGraw-Hill. 1995. <http://www.osf.oorg/faq-mauney.html>.

[Lorie and Nilsson 1979] Lorie, R. A. and J. F. Nilsson: “An Access Pattern Specification Language for a relational database system.” *IBM Journal Research and Development* 23: 286-298. 1979.

[Lowry and McCartney 1991] Lowry, M. R. and R. D. McCartney: "Automating Software Design", AAAI press/ MIT Press. 1991.

[Luckham and Vera 1996] Luckham and Vera: “An Event-Based Architecture Definition Language.” *IEEE Transactions on Software Engineering*. 1996.

[Mandrioli, Zicari et al. 1985] Mandrioli, D., R. Zicari, C. Ghezzi and F. Tisato: “Modeling the Ada task system by Petri Nets.” *ACM Computing Reviews* 10(1): 43--61. 1985.

[McIlroy 1976] McIlroy, D.: "Mass-produced Software Components". *Software Engineering Concepts and Techniques*. J. Buxton and et al., Van Norstrand Reinhold. 1976.

[Meyer 1992] Meyer, B.: "Eiffel: The Language", Prentice Hall, New York. 1992.

[Meyer 1994] Meyer, B.: "Reusable Software: The Base Object-oriented Component Libraries", Prentice-Hall. 1994.

[Michaelson 1995] Michaelson, G.: "Elementary Standard ML", UCL Press. 1995. <ftp://ftp.cee.hw.ac.uk/pub/funcprog/gjm.book95.ps.Z>.

[Miller and Thorpe 1995] Miller, D. C. and J. A. Thorpe: “SIMNET: The Advent of Computer Networking.” *Proceedings of the IEEE* 83(8): 1116-1123. 1995.

[Mills 1994] Mills, E.: “Users fear hassles in using component software.” *InfoWorld* 16(38): 19-20. 1994.

[Musen, Gennari et al. 1994] Musen, M. A., J. H. Gennari, H. Eriksson, S. W. Tu, et al.: "PROTEGE II: Computer Support for Development Of Intelligent Systems From Libraries Of Components", Knowledge Systems Laboratory, Medical Computer Science, Technical Report KSL-94-60. 1994.

[Newport 1993] Newport, J. P.: “The Growing Gap in Software.” *Fortune* 113: 32-142. 1993.

[Newton and Browne 1992] Newton and Browne: "The CODE 2.0 Graphical Parallel Programming Language". *ACM International Conference on Super Computing*. 1992.

[OMG 1991] OMG: "The Common Object Request Broker: Architecture and Specification", Object Management Group (OMG) and X/Open. 1991. <http://www.omg.org/>.

[Orbix 1995a] Orbix: "Orbix Distributed Object Technology Advanced Programmer's Guide", Iona Technologies Ltd. 1995a.

[Orbix 1995b] Orbix: "Orbix Distributed Object Technology IDL and C++ Mapping", Iona Technologies Ltd. 1995b.

[Orbix 1995c] Orbix: "Orbix Distributed Object Technology Programmer's Guide", Iona Technologies Ltd. 1995c.

[Orfali, Harkey et al. 1996] Orfali, R., D. Harkey and J. Edwards: "The essential distributed objects survival guide". New York, Wiley. 1996.

[Papakonstantinou, Gupta et al. 1995] Papakonstantinou, Y., A. Gupta, H. Garcia-Molina and J. Ullman: "A Query Translation Scheme for Rapid Implementation of Wrappers". *International Conference on Deductive and Object-Oriented Databases*. 1995. <http://www-db.stanford.edu/pub/papakonstantinou/1995/querytran.ps>.

[Parnas 1979] Parnas, D. L.: “Designing Software for Ease of Extension and Contraction.” *IEEE Trans. SE* 5(2): 128-138. 1979.

[Parnas 1995] Parnas, D. L.: “Fighting Complexity.” *IEEE TC EECS Newsletter* 2(2). 1995.

[Perrochon 1996] Perrochon, L.: "Gateways in globalen Informationssystemen". *Computer Science*. Zürich, Swiss Federal Institute of Technology (ETH Zürich), Switzerland**:** 155. 1996. <http://www.perrochon.com/>.

[Pyle 1985] Pyle, I. C.: "The Ada Programming Language", Prentice-Hall. 1985.

[Rosenberry, Kenney et al. 1994] Rosenberry, Kenney and Fisher: "Understanding DCE", OReilly. 1994.

[Samadzadeh and Zand 1995] Samadzadeh, M. and M. Zand, Eds.: "Proc. ACM SIGSoft Symposium on Software Reusability", ACM. 1995.

[Schneidewind 1987] Schneidewind, N. F.: “The State of Software Maintenance.” *IEEE Trans.on Software Engineering* 13: 303-310. 1987.

[Smith 1990] Smith, D. R.: “KIDS: A Semi-Automated Program Development System.” *IEEE Trans. Special Issue on Formal Methods* 16(9): 1024-1043. 1990.

[Stickel, Waldinger et al. 1994] Stickel, M., R. Waldinger, M. Lowry, T. Pressburger, et al.: "Deductive Composition of Astronomical Software from Subroutine Libraries". *Twelfth Intern.,Conf. on Automated Deduction,* Nancy, France. 1994.

[Stroustrup 1986] Stroustrup, B.: "The C++ Programming Language", Addison-Wesley. 1986.

[Summet 1969] Summet, J.: "Programming Languages", Prentice-Hall. 1969.

[Swartz 1994] Swartz, J.: "OpenDoc, OLE squaring off". *MacWeek*. 8**:** 2. 1994.

[Tomlinson, Cannata et al. 1993] Tomlinson, C., P. Cannata, G. Meredith and D. Woelk: “The Extensible Services Switch in CARNOT.” *IEEE Parallel & Distributed Technology* 1(2): 16-20. 1993.

[Tracz 1995] Tracz, W.: "Confessions of a Used Program Salesman", Addison-Wesley. 1995.

[Tracz 1996] Tracz, W.: "Cost of Documentation in DoD", Personal communication (e-mail). 1996.

[Udell 1994] Udell, J.: “Component Ware.” *Byte* 5: 46-56. 1994.

[Vestal 1994] Vestal, S.: "A Cursory Overview and Comparison of Four Architecture Description languages". Minneapolis, MN, Honeywell Technology Center. 1994. <ftp://src.honeywell.com/pub/ARCHIVE/dssa/papers>.

[Wasserman and Gutz 1982] Wasserman, A. and S. Gutz: “The Future of Programming.” *Communications of the ACM* 25(3). 1982.

[Wiederhold 1992a] Wiederhold, G.: “Mediators in the Architecture of Future Information Systems.” *IEEE Computer*(3): 38-49. 1992a.

[Wiederhold 1992b] Wiederhold, G.: "Model-free Optimization". *DARPA Software Technology Conference,* Loas Angeles, CA, Medridien Corporation, Arlington VA. 1992b.

[Wiederhold 1993] Wiederhold, G.: "Intelligent Integration in Simulation". *MORS Mini-symposium,* Fairfax VA, Military Operations Research Society. 1993.

[Wiederhold, Wegner et al. 1992] Wiederhold, G., P. Wegner and S. Ceri: “Towards Megaprogramming.” *Comm. ACM*(11): 89-99. 1992.

[Wolf 1985] Wolf, A. L.: "Language and Tool Support for Precise Interface Control". Amherst, Mass, Computer and Information Science Department, University of Massachusetts. 1985.

[Xerox 1983] Xerox: "Interlisp Reference Manual", Xerox Corporation. 1983.

[Zelkovitz, Yeh et al. 1984] Zelkovitz, M. V., R. T. Yeh, R. G. Hamlet, J. D. Gannon, et al.: “Software Engineering Practices in the US and Japan.” *IEEE Computer* 17(6): 57-66. 1984.