دانشگاه آزاد اسلامی
واحد تهران جنوب

دانشکده فنی- مهندسی
بخش مهندسی کامپیوتر

پروژه درس شبکه های کامپیوتری

عنوان پروژه :

برنامه نویسی در محیط Client/Server

با استفاده از مدل

CORBA

"Common Object Request Broker Architecture"

تهیه و تنظیم :

امین جلالی و محمد مرتضوی و علیرضا فتحی

استاد : آقای فیروزبخت

سال تحصیلی ۸۱-۸۰

۱- مقدمه :

در ابتدا و قبل از اینکه به ساختار و مفهوم *CORBA* بپردازیم لازم است تا در این فصل شما خوانندگان محترم را با یک سری از مفاهیم اولیه آشنا و یا به یادآوری آنها بپردازم:

۱-۱ *Intended Audience:*

ساختار و شرحی که در اینجا به آن اشاره میشود جهت طراحان و پیاده سازانی است که میخواهند برنامه هایی بر مبنای استانداردهای گروه *OMG* و یا *Object) (Management Group* و با استفاده از روش *ORB* و یا *(Object Request Broker)* پیاده سازی نمایند.

۱-۲ مفهوم *CORBA (Context of CORBA)* :

کلید فهمیدن ساختار *CORBA* مدل مرجع آن است که تشکیل شده از قسمتهای زیر است:

۱-۲-۱ *واسط درخواست شیی و یا ORB* :

که این قابلیت را به اشیاء میدهد که به صورت شفاف *(Transparent)* درخواستها و جوابها *(Request & Response)* را در یک محیط پخشی و توزیعی *(Distribute)* ساخته و یا دریافت نمایند. این موضوع پایه ساخت برنامه هایی با *Object* های پخش شده میباشد و برای عملیات وابسته به برنامه ها در محیط های همگون *(Homogeneouse)* و یا محیط های ناهمگون و متضاد *(Hetrogeneouse)* میتوان از آن استفاده نمود.

ساختار و مشخصات *ORB* کاملاً در فصل ۲ مورد بحث قرار گرفته است.

۱-۲-۲ سرویسهای اشیاء و یا *Object Services*:

مجموعه ای از سرویسها شامل اشیاء (*Objects*) و محیط های واسط (*Interfaces*) که قادر هستند توابع پایه ای، را اجرا نمایند تا بتوانند از اشیاء استفاده و توابع آنها را اجرا نمایند.
سرویسها برای ایجاد کردن برنامه های *Distribute* لازم هستند و معمولاً از قلمرو برنامه مستقل هستند، به عنوان مثال سرویس *Life Cycle* یک برنامه قواعدی را برای ساخت و حذف و کپی و جابجا کردن *Object* ها تعریف مینماید ولیکن چگونگی اجرای آنها را در برنامه امر نمی نماید.
مشخصات سرویسهای اشیاء در *CORBA Services* تعریف شده است.

۱-۲-۳ امکانات متعارف ویا *Common Facilities*:

مجموعه ای از سرویسها که تعداد زیادی از برنامه ها میتوانند به اشتراک بگذارند و *Share* نمایند ولیکن به اندازه *Object Service* ها پایه ای نیستند به عنوان مثال مدیریت سیستم ویا تسهیلات یک نامه الکترونیکی می توانند در این قسمت کلاسه بندی شوند.

۱-۲-٤ اشیاء برنامه های اجرایی (*Application Objects*) :

Object هایی است که جهت کنترل *Interface* ها و توسط اشخاص تولید میشود و توسط *OMG* استاندارد نشده است، این برنامه ها در مدل مرجع در بالا ترین لایه قرار گرفته اند.(اشکال مربوطه به مدل مرجع را میتوانید در فصل بعد ملاحظه نمایید)

۱-۳ ***Object Model*:**

در این قسمت به بشریح واقعی *Object Model* بر اساس ساختار *CORBA* خواهیم پرداخت که این مدل خلاصه ای از *Core Object Model* میباشد که

توسط *OMG* تهیه شده است، و در سه بخش بـه شـرح ایـن موضـوع خـواهیم پرداخت:

۱-۳-۱ خلاصه و دید اجمالی

۱-۳-۲ معنی *Object* ها

۱-۳-۳ پیاده سازی *Object (Object Implementation)*

۱-۳-۱ خلاصه و دید اجمالی:

Object Model شامل یک ارائه سازمان یافته از مفهوم *Object* و یک سری اصطلاحات فنی است و یک مدل مرجعی را جهت محاسبات تعریف مینماید کـه این مدل شامل کلیه مشخصات *Object* ها است که توسط تکنولوژی های ارائـه شده درک شود.

Object Model ای که توسط *OMG* بیان شده است بـه صـورت خلاصـه اسـت و مستقیماً از طریق یک تکنولوژی و مدل خاص درک نمیشود. ولـیکن مـدلی کـه در اینجا به آن اشاره مینماییم و آنرا توضیح میدهیم مـدل واقعـی اسـت و در نقاطی با مدل خلاصه شده تفاوت دارد:

- ممکن است که *Object Model* خلاصه را با جزیئات بیشتری شرح دهند و آنرا به صورت خاص پیاده نمایند، به عنـوان مثـال یـک روش خـاص برای پارامترهای *Request* و یا یک زبان خاص جهت پیاده سازی که تنها یک مدل خاص را تعریف مینماید.
- ممکن است که به وسیله معرفی نمونه های خاصی که به وسیله ایـن مدل تعریف میشود، مدل را عمومی نمایند. به عنوان مثال : *Object* هـای خاص، *Operation* های خاص ویا *Type* های خاص.
- ممکن است که مـدل را بـا حـذف موجودیتهـا و یـا بـا اضـافه کـردن محدودیتهای اضافی برای استفاده محدود نمایند.

یک *Object System* مجموعه ای از *Object* ها است که درخواستهای *Client* ها را از سرویس دهندگان بوسیله یک واسط کپسوله کننده جدا می سازد. *Client* ها از سرویسهای اجرایی و سرویسهای ارائه دهنده و کد اجرایی جدا هستند .

Object Model در ابتدا مفاهیمی که برای *Client* ها معنی دار است را شرح میدهد که این مفاهیم شامل تولید یک *Object* و مشخصات آن، درخواستها و عملیات ، *Type* ها ،*Signature* ها و ... میباشد و پس از این به شرح مفاهیم مربوط به اجرای *Object* ها شامل مفاهیمی از قبیل *Method*، *Engine* های اجرایی و *Activation* خواهیم پرداخت.

Object Model در تعریف مفاهیم معنی دار برای *Client* ها بسیار خاص و معین است. در صورتیکه بخواهیم راههای متفاوتی را جهت اجرای *Object* ها داشته باشیم لازم است که بیشترین حد آزادی را برای تکنولوژی های *Object* های گوناگون در نظر بگیریم.

Object Model به ما نمیگوید که *Client* ها و یا *Server* ها به صورت *Single Thread* هستند و یا *Multi-Thread* ، مشخص نمی نماید که چگونه چرخه های رویداد برنامه ریزی میشوند و یا اینکه چگونه *Thread* ها *Creat*، *Destroy* و یا *Synchroniz* میشوند.

این *Object Model* یک نمونه از مدل کلاسیک آن است که *Client* یک پیغام و یا *Message* را ارائه میدهد. در مدل کلاسیک پیغامها میتوانند شامل صفر و یا تعداد بیشتری پارامتر باشند که در بیشتر آنها اولین پارامتر جهت متمایز ساختن پیغامها لازم است و انجام عملیات تفسیر، مبتنی بر انتخاب یک *Method* بر اساس پارامترهای ورودی است که این *Method* ها را میتوان بر اساس *ORB* نیز مشخص نمود.

۱-۳-۲ معنای اشیاء و یا *Object Semantics* :

سیستمهای مبتنی بر*Object* سرویسهایی را برای *Client* ها آماده می نمایند، و یک *Client* ، هر موجودیتی است که قابلیت درخواست یک سرویس را داشته باشد.

در این قسمت به مفاهیمی خواهیم پرداخت که با معنی *Object* ها در ارتباط هستند که این مفاهیم عبارتند از :

۱-۳-۲-۱ *Object* ها :

یک سیستم مبتنی بر *Object* شامل موجودییتهایی است که به عنوان *Object* نامیده میشوند و یک *Object* با این تعریف به این صورت قابل شناسایی میباشد که یک موجودیت کپسوله شده که یک و یا چند سرویس را که یک *Client* میتواند درخواست نماید را تهیه مینماید.(این سرویسها در واقع همان *Method* های داخل *Object* هستند)

۱-۳-۲-۲ درخواست ها (*Requests*) :

Client ها سرویس های درخواستی خود را به وسیله *Request* ها بیان می نمایند. یک *Request* یک رخداد است (مانند چیزی که در یک زمان مشخص و خاص اتفاق می افتد). اطلاعات وابسته به یک *Reuest* عبارتند از: *Operation*، *Object* هدف ، پارامترها و یک قسمت اختیاری با عنوان مفهوم درخواست.

یک *Request form* توضیح و یا الگویی است که میتواند چندین مرتبه ارزیابی شود تا باعث ارائه *Request* ها بشود و از طریق *Binding* های خاص زبانی که با آن کار میشود، مشخص خواهد شد.

Value : هر مقداری است که در پارامتر یک درخواست مشروع باشد.

Object Reference : یک مقداری است که صریحاً یک *Object* خاص را مشخص مینماید و ممکن است این *Object Reference* در درخواستهای مختلف یک *Object* را مشخص نماید.(به صورت یک *Pointer* به *Object* عمل می نماید.)

یک درخواست ممکن است از پارامترهایی جهت انتقال اطلاعات به شیی هدف استفاده نماید و حتی ممکن است مفهوم درخواست نیز داشته باشد که اطلاعات اضافه تری را درباره درخواست به ما بدهد.

درصورتیکه در زمان اجرای *Request* یک موقعیت غیر نرمال به وجود آید یک پیغام استثنا (*Exception*) برگشت داده میشود که شامل پارامترهای اضافی است که منحصر به آن استثناء خاص هستند و در برنامه قادر هستیم آنها را در قسمت *Exception Handler* تشخیص داده و عملیات مناسب را برای مواجه با آن پیشبینی نماییم.

پارامتر های یک *Request* از طریق موقعیتشان مشخص میشوند. یک پارامتر ممکن است یک پارامتر ورودی و یا یک پارامتر خروجی و یا یک پارامتر ورودی – خروجی باشد، که در آینده درباره اینگونه پارامتر ها بیشتر بحث می شود.

۱-۳-۲-۳ *ایجاد و از بین بردن Object ها:*

Object ها را میتوان به وجود آورد و از بین برد، از نقطه نظر *Client* ها هیچ نوع مکانیزم خاصی جهت ایجاد و از بین بردن *Object* ها وجود ندارد. *Object* ها پس از درخواست یک *Request* ساخته و یا از بین میروند، در واقع مدیریت *Object* ها توسط *Server* انجام می شود.

۱-۳-۲-٤ نوع (Type) :

تعدادی از *Value* های مجاز داخل یک *Request* در زیر آمده است :

Basic Value: Short, Long, LongLong, Ushort, Ulong, Ulonglong, Float, Double, Fixed, Char, String, Boolean, Any .
Constructed Values: Array, Union, Sequence, Struct .

یک *Type* یک موجودیت قابل شناسایی است که با مفهوم *Value* تعریف میشود.

۱-۳-۲-٥ عملگرها (Operations) :

یک *Operation* یک موجودیت قابل شناسایی است که یک سرویس را مشخص مینماید که این سرویس می تواند درخواست شود و به وسیله *Operation Identifier* شناسایی میگردد.

۱-۳-۲-٦ Interface ها (واسطهاو یا رابط ها):

Interface شرحی از یک سری عملیات است که یک *Client* قادر است آنها را از طریق آن *Interface* درخواست نماید. و نوع واسط برای یک واسط خاص وابسته به نوع شیی است که *Object Reference* نیز به آن اشاره مینماید.

Interface ها در *OMG IDL* تعریف و مشخص شده اند. وراثت در *Interface* ها باعث می شود که یک *Object* قادر باشد تا از چندین *Interface* بهره بگیرد و بتواند از تمامی عملیاتی که در واسطهایی که در زیر آن هستند استفاده نماید.

۱-۳-۲-۷ *پارامتر ها (Parameters) :*

یک پارامتر بـه وسـیله *Mode* و *Type* خـود مشـخص میشـود، در صورتیکه مقداری از *Client* به سمت *Server* فرستاده بشـود، پـارامتر از نوع ورودی (*In*) خواهد بود و در صورتیکه از سمت *Server* به سـمت *Client* فرستاده شود از نوع خروجی (*Out*) اسـت، و یـا از هـردو نـوع است که به آن پارامتر ورودی- خروجی (*InOut*) میگویند.

۱-۳-۲-۸ *جواب و نتیجه بازگشتی :*

نتیجه بازگشتی بوسیله پارامتر های خروجی منتقل میشود، بدین معنی که بعد از رسیدن درخواست به *Server*، *Method* مورد نظـر اجـرا مـی شـود و نتیجـه آن بـه صـورت یـک پـارامتر خروجـی بـه *Client* برگردانده می شود.

۱-۳-۲-۹ *استثناء ها (Exceptions) :*

یک اسـتثنا نمایـانگر آن اسـت کـه یـک درخواسـت *Operation* بـه صورت موفقیت آمیز خاتمه نیافته است. و آن استثنا هـایی کـه بوجـود آورنده آنها سیستم باشد با آنهایی که در حـین برنامـه و بـه علـت یـک عملیات در برنامه بوجود می آیند متفاوت و قابل تشخیص برای زبـانی است که این استثنا ها را کنترل مینماید.

۱-۳-۳ *Object Implementation (پیاده سازی اشیاء):*

پیاده سازی یک سیسـتم *Object* ای باعـث بـه وجـود آمـدن فعالیتهـایی خواهد شد که جهت رفتار سرویس های درخواست بر روی *Object* ها پیشبینی شده است. این فعالیتها میتواند شامل محاسبه بر روی نتایج درخواستها باشـد و بوسیله نتایج آن وضعیتهای موجود سیستم را به روز نماید.

مدل پیاده سازی شامل دو قسمت میباشد:

- مدل پیاده سازی
- مدل ساختمانی و یا *Construction Model*

که مدل اول نمایانگر این مطلب است که چگونه سرویسها پیاده سازی می شوند.

و مدل دوم به شرح چگونگی تعریف سرویسها می پردازد.

۱-٤ آشنایی با CORBA:

معماری *CORBA* محصول شرکت *OMG* (گروه مدیریت *Object*) یک ساختار مستقل از زبان و مستقل از *Platform* را برای نوشتن برنامه های شیی گرا به صورت توزیع شده، فراهم می سازد، *Object* های *CORBA* قادر هستند تا در یک پروسس روی یک ماشین مقیم شوند.

CORBA را میتوان با زبانهای مختلفی برنامه نویسی نمود ولیکن در این پروژه تمامی برنامه ها به زبان *Java* نوشته شده است و این به علت دو مزیت *Java* نسبت به زبانهای دیگر بوده است . اولین مزیت این است که زبان *Java* با *OMG IDL* سازگار میباشد و قادر هستیم فایلهای *Java* را به *IDL* و یا برعکس تبدیل نماییم و دومین مزیت زبان برنامه نویسی جاوا قابلیت *Garbage Collection* آن در محیط و زمان اجرا است.

۱-٤-۱ علت استفاده از *CORBA*:

به سه علت عمده از *CORBA* استفاده می نماییم و این سه عبارتند از :

- *Data* توزیع شده : بعضی از برنامه ها روی چندین کامپیوتر اجرا میشوند زیرا نیاز به اطلاعاتی دارند که پراکنده هستند، به عنوان مثال زمانیکه بخواهیم یک برنامه را بر روی چند

سیستم اجرا نماییم و این برنامه فقط اجازه دسترسی به فایلها را داشته باشد و نتواند چیزی را ذخیره نماید.

- *Computation* موازی : درصورتی که بخواهیم برنامه بر روی چند سیستم و چند پردازنده به صورت موازی اجرا شود.
- استفاده کنندگان توزیع شده : درصورتی که کاربران به عنوان مثال از طریق یک برنامه بخواهند با هم گفتگو و محاوره نمایند.

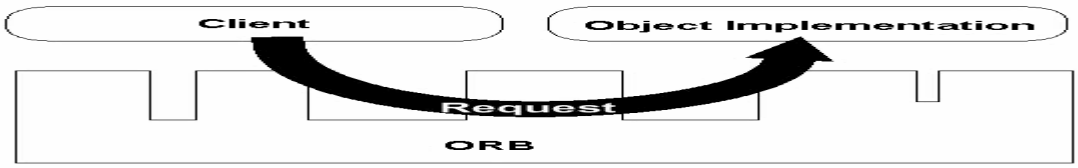
۲- ساختار *CORBA* :

۲-۱ پیشگفتار:

جهت برنامه نویسی در سطح شبکه های پخشی دو مدل مبتنی بر *Object* تهیه شده است که این دو مدل عبارتند از *CORBA* و *DCOM* ، که *DCOM* محصول شرکت *Microsoft* و *CORBA* محصول *Object Management Group* میباشد. هر دوی این مدل ها برطبق مدل مرجع *RPC (Remote Procedure Call)* عمل می نمایند و در این فصل به بررسی و شرح مدل *CORBA* خواهیم پرداخت.

۲-۲ ساختار واسط درخواست شیی(*ORB*) :

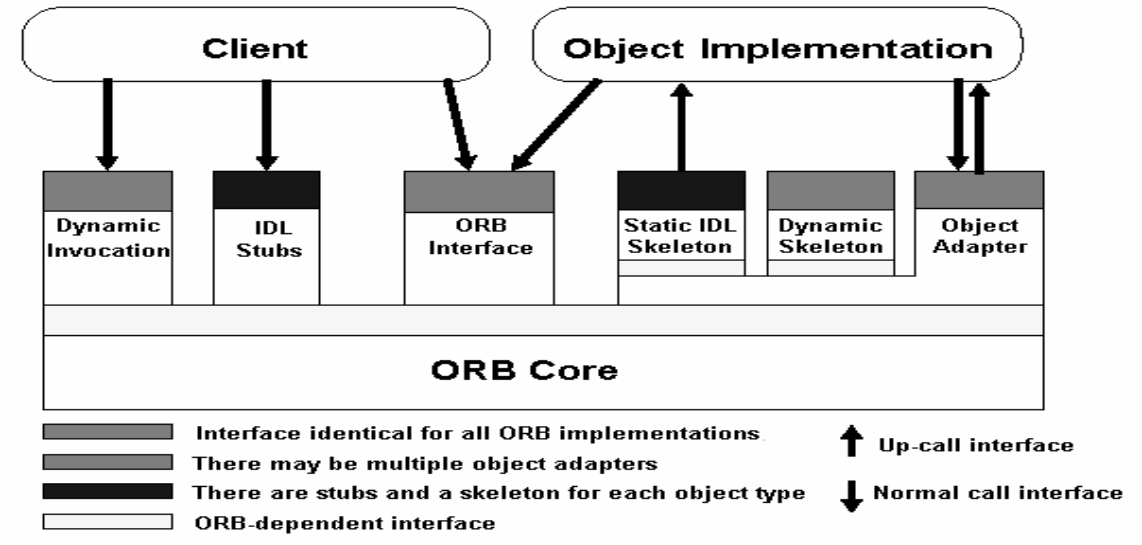
شکل ۲-۱ نمایانگر یک درخواست و یا *Request* میباشد که توسط *Client* به *Object Implementation* و یا قسمت پیاده سازی *Object* فرستاده میشود. *Client* موجودیتی است که می خواهد یک *Operation* را بر روی *Object* انجام دهد. *Object Implementation* کد و داده ای است که واقعاً *Object* را اجرا می نماید.



(شکل ۲-۱)

مسئولیت مکانیزمهای مورد نیاز جهت اجرای *Object* برای یک درخواست به عهده *ORB* است، و تدارک دیدن *Object Implementation* و دریافت درخواست و ارتباط با *Data* ای که این درخواست را ساخته است به عهده *ORB* می باشد. واسط و یا *Interface* یک *Client* کاملاً به محل قرار گرفتن *Object* ها و اینکه توسط چه زبانی برنامه نویسی می کنید وابسته می باشد.

شکل ۲-۲ نمایانگر ساختار یک واسط درخواست *Object* و یا *ORB* منحصر به فرد میباشد. واسطهایی که به *ORB* متصل هستند بوسیله جعبه های مارک دار نشان داده شده است و فلشها نمایانگر این موضوع هستند که کدام *Interface* و یا واسط *ORB* صدا زده شده است و یا اینکه یک *Up-Call* را از میان *Interface* نمایش میدهند.



(شکل ۲-۲)

جهت ساختن یک درخواست به یک *Client* میتوان از واسط *Dynamic Invocation* که یک *Interface* مشابه و وابسته به واسط *Object* هدف می باشد استفاده نمود و یا اینکه از *IDL Stubs* که *Stub* مخصوص وابسته به واسط *Object* هدف میباشد، استفاده نمود. همانگونه که در شکل صفحه قبل نیز ملاحظه نمودید یک*Client* همچنین میتواند برای یکسری از عملیات و کارها مستقیماً با خود *ORB* نیز در تماس باشد.

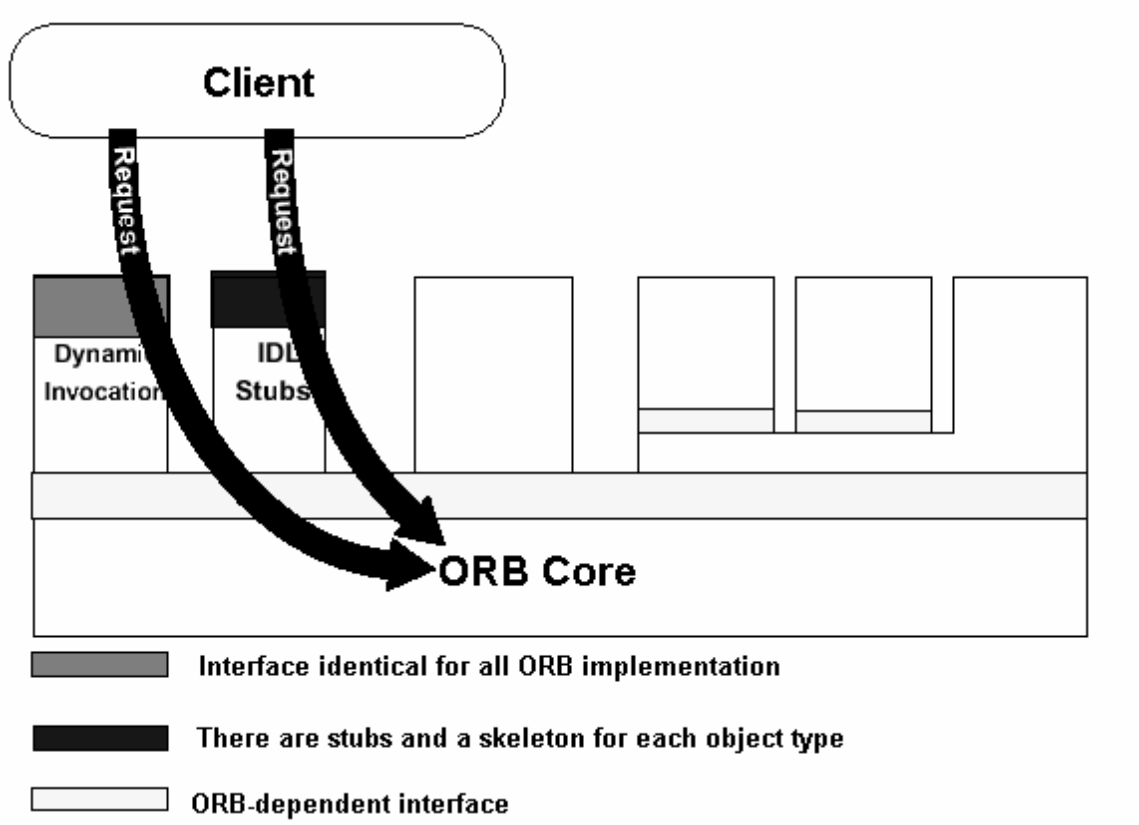
Object Implementation قادر است یک درخواست را از طریق *OMG IDL* *generated Skeleton* دریافت نماید و یا از طریق *Dynamic Skeleton* این دریافت صورت پذیرد. قسمت *Object Implementaion* میتواند *Object Adapter* و *ORB* را زمانیکه در حال پردازش یک درخواست است و یا در هر زمان دیگری صدا بزند.

به دو صورت میتوان در واسط ها بر روی *Object* ها تعاریفی را که مایل هستیم انجام دهیم. واسط ها میتوانند به صورت ایستا (*Static*) این تعریف را با استفاده از زبان تعریف واسط (*Interface Definition Language*) انجام دهند که به آن *OMG IDL* گویند. این زبان به تعریف انواع *Object* ها مطابق با عملیاتی که بر روی آنها انجام میشود و پارامتر هایی که در طی این عملیات به آنها نیاز داریم می پردازد. مضاف بر اینکه واسط ها می توانند به سرویس *Interface Repository* و یا مخزن واسط اضافه شوند. این سرویس اجزاء یک واسط را به صورت اشیاء ارائه میدهد و اجازه دسترسی به این اجزاء را در زمان اجرا فراهم می آورد. در هر *Object Implementation* هر دوی این روشها *IDL* و *Interface Repository* دارای کارایی یکسانی هستند و شما میتوانید از هر کدام از آندو استفاده نمایید.

یک *Client* یک درخواست را به وسیله دسترسی به *Object Reference* آن شیی ارائه می دهد و اینکه آن *Object* و یا شیی از چه نوعی است و چه عملیاتی را مایل است انجام دهد،*Client* به بررسی درخواست می پردازد و به وسیله صدا کردن *Method* های *Stub* که مخصوص آن شیی هستند این کار را انجام میدهد

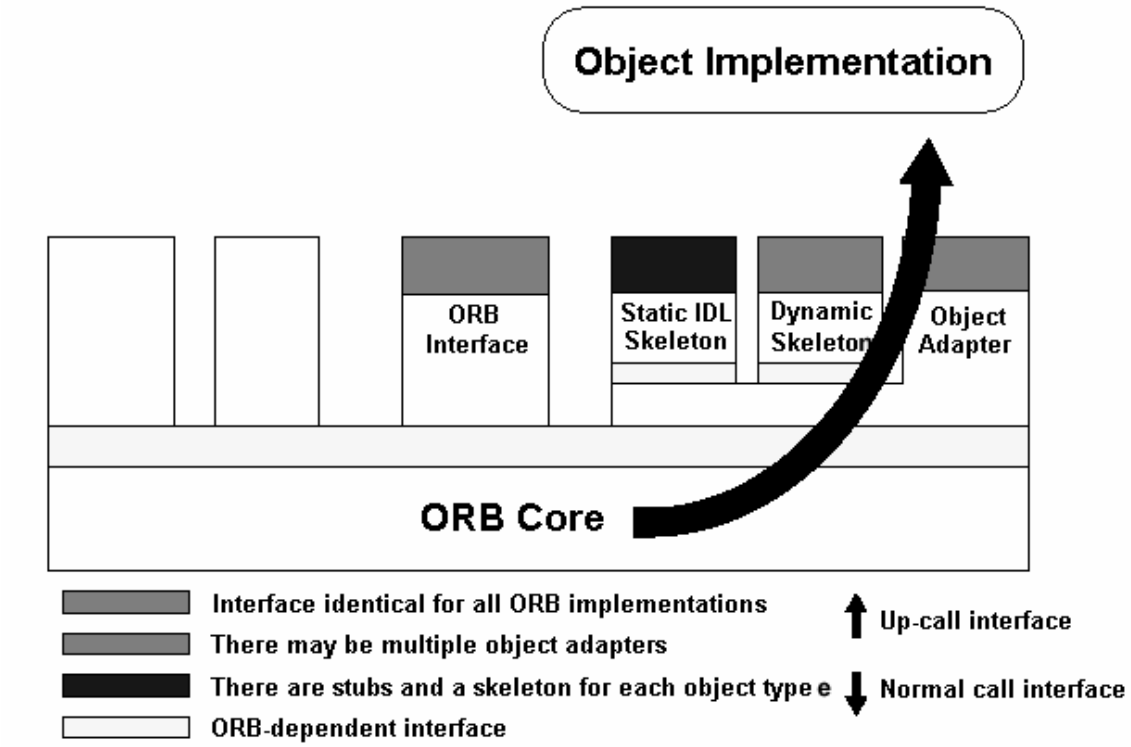
و یا اینکه درخواست را به صورت پویا و *Dynamic* می سازد.(به شکل زیر توجه فرمایید.)

(شکل ۲-۳)



هر دو روش *Dynamic* و واسط *Stub* هر دو جهت احضار یک درخواست از نظر معنایی به یک صورت عمل می نمایند، و دریافت کننده پیغام قادر نیست تشخیص دهد که این *Request* چگونه و از کدام روش *Invoke* (احضار) شده است. *ORB* محل تقریبی کد اجرایی را معین می سازد، پارامتر ها را انتقال میدهد و کنترل را از طریق *IDL Skeleton* و یا *Dynamic Skeleton* به قسمت *Object Implementation* منتقل می سازد.(به شکل ۲-٤ در صفحه بعد دقت نمایید)

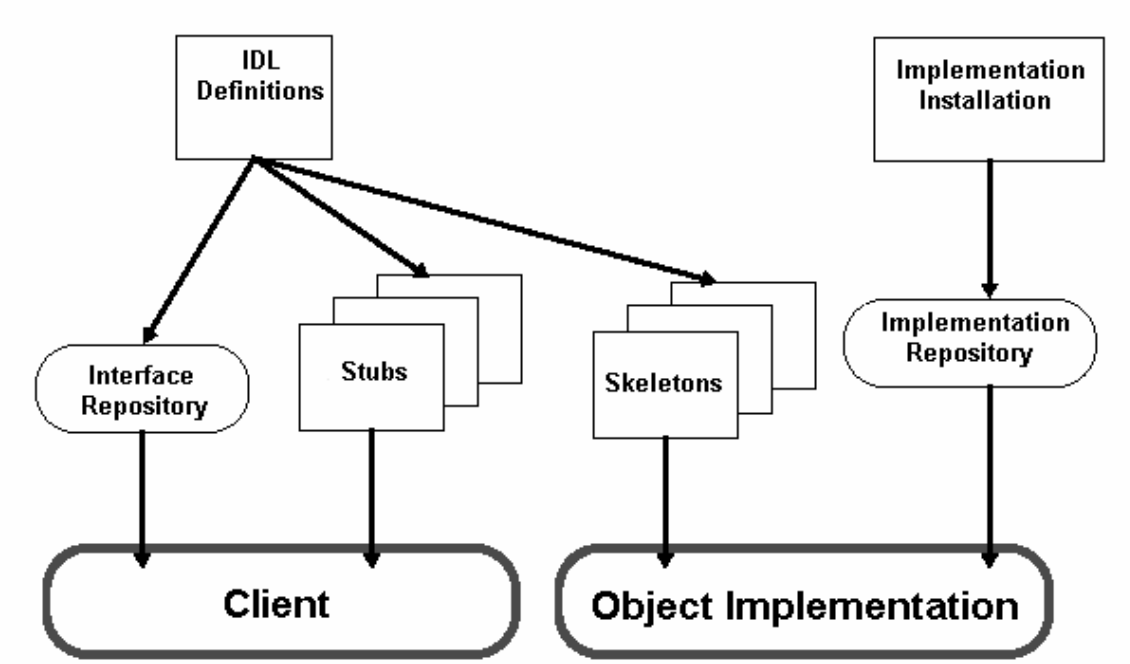
Skeleton ها برای *Interface* و همچنین *Object Adapter* مشخص و معین هستند. در اجرای یک *Request* ، قسمت *Object Implementation* تعدادی از سرویسهای خود را از *ORB* و از طریق *Object Adapter* بدست می آورد و زمانیکه درخواست کامل میشود و در واقع اجرا میشود، کنترل و مقادیر خروجی به *Client* باز گردانده میشود.



(شکل ۲-٤)

قسمت اجرای اشیاء و یا همان *Object Implementation* قادر است انتخاب نماید که از چه *Object Adapter* ای میخواهد استفاده نماید و این تصمیم گیری را بر اساس نوع سرویسی که احتیاج دارد خواهد گرفت.

شکل ۲-۵ نمایش دهنده این موضوع است که چگونه *Interface* و اطلاعات اجرایی برای *Client* ها و *Object Implementation* قابل دسترسی میباشد. این



واسط توسط *OMG IDL* و یا *Interface Repository* تعریف شده است، این تعاریف تولید *Stub* ها برای *Client* ها و *Skeleton* ها برای *Object Implementation* می باشد.

(شکل ۲-۵)

اطلاعات قسمت *Object Implementation* در زمان نصب تهیه میشود و در قسمت *Implementation Repository* ذخیره میشود تا در زمان تحویل *Request* از آن استفاده شود.

۲-۲-۱ *واسط درخواست شیی و یا ORB:*

در ساختار و معماری لازم نیست که *ORB* همانند یک جزء تنها اجرا شود. بلکه بیشتر مواقع *ORB* بوسیله واسط ها تعریف میشود. هر اجراء *ORB* که یک واسط مناسب را تهیه نماید قابل قبول است.

واسط ها و یا *Interface* ها به سه گروه تقسیم میشوند:

- عملیاتی که برای تمام اجراهای *ORB* همانند هم هستند.

- عملیاتی که مخصوص گروه خاصی از *Object* ها هستند.
- عملیاتی که مخصوص حالات بخصوصی از *Object Implementation* ها هستند.

ORB های گوناگون قادر هستند گونه های مختلفی از اجراء را بسازند و بوسیله *IDL Compiler* ها و *Repository* ها و انواع گوناگون *Object Adapter* ها یک سری سرویسها را برای *Client* ها و *Object Implementation* ها که دارای خصوصیات متفاوتی هستند تهیه نمایند.

شاید چندین اجرا از *ORB* داشته باشیم(که این مسئله به عنوان *Multiple ORBs* نیز تعبیر می شود) که در آنها نمایشهای گوناگونی برای *Object Reference* و معانی گوناگون از اجرای *Invocation* ها اعمال شده باشد و این امر یک *Client* را قادر میسازد تا به چندین *Object Reference* به صورت همزمان دسترسی داشته باشد، که این دسترسی بوسیله *ORB Implementation* های گوناگون و متفاوت مدیریت می شود.

زمانیکه دو *ORB* قصد دارند که با همدیگر کار نمایند، هرکدام از آنان باید قادر باشند که *Object Reference* خود را تشخیص دهند و این مسئولیت و وظیفه *Client* ها نیست که این کار را انجام دهند.

قسمت *ORB Core* آن قسمتی از *ORB* است که مسئولیت تهیه و ارائه اولیه *Object* ها و ارتباط درخواستها را برعهده دارد. طراحی *CORBA* این عمل را بوسیله ساختن *ORB* بوسیله اجزایی بر روی *ORB Core* انجام داده است، این اجزاء واسطهایی را تهیه مینماید که تفاوتهای میان *ORB Core* های مختلف را بپوشاند.

۲-۲-۲ *Client* ها :

یک *Client* جهت دسترسی به یک *Object* به مرجع آن یعنی *Object Reference* دسترسی دارد و بوسیله این *Reference* ها است که انجام عملیات بر

روی *Object* را درخواست می نماید. یک *Client* تنها ساختار منطقی *Object* را می داند و آنرا از طریق واسط آن و یا از طریق مشاهداتی که از شیی در زمان درخواستها از خود نشان داده است میتواند بفهمد. همچنین ما معمولاً ملاحظه می نماییم که یک *Client* یک برنامه است و یا یک پردازش که یک *Request* را بر روی یک *Object* وارد می سازد ، این مسئله بسیار مهم است که بتوانیم یک *Client* را به یک *Object* بخصوص منصوب نمائیم. به عنوان مثال : پیاده سازی یک *Object* ممکن است که یک *Client* باشد برای *Object* ای دیگر.

Client ها معمولاً اشیاء و واسط های *ORB* را از طریق یک زبان رویت می نمایند، که *ORB* را تبدیل به یک سطح برنامه نویسی کرده است. *Client* ها عموماً *Portable* هستند و باید این توانایی را داشته باشند که بدون تغییر *Source* ، آنها را بر روی هر *ORB* اجرا نماییم.

Client ها هیچگونه آگاهی از *Object Implementation* ندارند و یا حتی اینکه چه *Object Adapter* ای بوسیله قسمت اجرایی مورد استفاده قرار می گیرد و یا اینکه از چه *ORB* برای دسترسی به آن استفاده می شود.

۲-۲-۳ *اجرای اشیاء و یا Object Implementation :*

وظیفه یک *Object Implementation* تهیه معنی یک *Object* است و معمولاً این کار را با تعریف داده برای *Object Instance* و کد برای *Object Method* ها انجام میدهد. گاهی اوقات *Implementation* از اشیاء دیگر و یا نرم افزارهای اضافی استفاده می نماید تا بوسیله آنها رفتار شیی را اجرا نماید.

گونه های مختلف *Object Implementation* قابل پشتیبانی است که شامل *Server* های مجزا، یک برنامه برای هر *Method* ، یک برنامه کپسوله شده، یک پایگاه اطلاعاتی شیی گرا و غیره .

در استفاده از *Object Adapter* های اضافه شده قابلیت پشتیبانی مجازی از هر نوع *Object Implementation* فراهم آمده است.
معمولاً *Object Implementation* به *ORB* وابستگی ندارد و یا به اینکه چگونه یک *Client* یک *Object* را درخواست می نماید، قسمت *Object Implementation* واسط های سرویسهای وابسته به *ORB* را انتخاب می نماید که این انتخاب بر اساس *Invoke* ها باعث انتخاب و گزینش *Object Adapter* می شود.

۲-۲-٤ *Object Reference* ها :

یک *Object Reference* اطلاعاتی است که جهت مشخص ساختن یک *Object* در یک *ORB* به کار گرفته می شود. و هر دو قسمت *Client* و *Object Implementation* مفهوم مبهمی از *Object Reference* ها را مطابق با نگاشت زبان (*Language Mapping*) بیان میدارند و بنابر این آنها از ارائه واقعی خودشان فاصله میگیرند. در اجرای دو *ORB* ممکن است در قسمت انتخاب ارائه *Object Reference* ها با هم تفاوت داشته باشند.
ارائه یک *Object Reference* که توسط یک *Client* انجام می شود تنها در مدت زمان حیات (*Life Time*) آن *Client* اعتبار دارد.
تمامی *ORB* ها باید نگاشت زبان (*Language Mapping*) یکسانی را برای یک *Object Reference* در یک زبان خاص تهیه نمایند. این موضوع به ما اجازه میدهد که برنامه را با یک زبان مشخص و خاص بنویسیم تا بتوانیم به *Object Reference* ها مستقل از یک *ORB* مشخص دسترسی داشته باشیم.
همچنین نگاشت زبان راههای بیشتری را جهت دسترسی به *Object Reference* ها ارائه میدهد که باعث راحتی کار برنامه نویس خواهد شد.
یک نوع *Object Reference* متمایز نیز وجود دارد که مطمیناً از تمامی *Object Reference* های دیگر متمایز است و آن *Reference* به *no Onject* است.(مانند *NULL* در زبانهای دیگر)

۲-۲-۵ *زبان تعریف واسط OMG:*

OMG IDL بوسیله مشخص ساختن واسط آنها به تعریف انواع *Object* ها میپردازد. یک واسط شامل یک دسته از عملگرهای نام گذاشته شده و پارامتر هایی به آن عملگرها میباشد. به این نیز توجه داشته باشید که *IDL* یک محیط و بدنه قابل درک را بوجود می آورد تا بوسیله *ORB* بتوان تغییرات لازم را بر روی *Object* ها انجام داد و *ORB* جهت انجام عملیات و کار احتیاجی به *Source Code* ندارد. و تا زمانیکه اطلاعات مشابه از طریق روتینهای *Stub* و یا *Interface Repository* در زمان اجرا در دسترس هستند یک *ORB* بخصوص قادر است تا بدرستی عمل نماید.

IDL از طریق *Object Implementation* مخصوص به *Client* مورد نظر خود میگوید که چه عملیاتی در دسترس است و چگونه باید *Invoke* (خوانده) شوند. بوسیله تعاریف *IDL* این امکان وجود دارد که قادر باشیم اشیاء *CORBA* را بوسیله زبانهای برنامه نویسی و یا سیستمهای شیی (*Object Systems*، *Map*) نماییم.

۲-۲-۶ *نگاشت OMG IDL به زبانهای برنامه نویسی:*

انواع گوناگون زبانهای برنامه نویسی *Object-Oriented* و یا غیر *Object-Oriented* ترجیح میدهند که به اشیاء *CORBA* به روشهای مختلفی دسترسی داشته باشند. برای زبانهای برنامه نویسی مبتنی بر *Object* این اشتیاق وجود دارد که اشیاء *CORBA* را همانند *Object* های زبانهای برنامه نویسی مشاهده نمایند. حتی برای زبانهای برنامه نویسی که مبتنی بر *Object-Oriented* نیستند نیز این ایده جالبی است که ارائه *ORB* کامل را از *Object Reference* و یا اسامی *Method* ها و غیره جدا نماییم.

Map کردن یک *OMG IDL* به یک زبان برنامه نویسی باید برای تمام اجرا های *ORB* یکسان باشد، و این عملیات *Mapping* شامل تعریف انواع داده و محیط روتین ها است که چگونه به اشیاء از طریق *ORB* دسترسی داشته باشیم و

تمـامی مفـاهیم دیگـر از قبیـل *Dynamic Invocation Interface* و *Implementation Skeleton* و *Object Adapter* ها و واسط *Direct ORB* تحت تـاثیر عملیـات *Mapping* قرار دارد.

نگاشت زبان همچنین به تعریف فعـل و انفعـالات میـان *Object Invocation* هـا و *Thread* ها می پردازد. معمول ترین نوع نگاشت فراخوانی همزمان را ارائـه مـی دهد. نگاشتهای دیگر ممکن است به این صورت عمل نمایند که اجازه بدهند یک *Call* وارد شود و کنترل را به برنامه بر میگرداند. در اینگونه موارد روتینهـای آن زبان باید *Thread* های برنامه را به صورت همزمان بسازند.

۷-۲-۲ *Client Stubs:*

جهت نگاشت در یـک زبـان *Non-Object-Oriented* احتیـاج بـه یـک واسـط برنامه نویسی برای *Stub* ها در هر یک از *Interface Type* ها داریم. معمولاً این *Stub* ها هستند که ارائه دهنده دسترسی به عملیات تعریـف شـده در *OMG IDL* را فراهم میسازند و این عملیات در زمانی که نگاشـت بـر روی آن انجـام مـی شود باید به صورتی پیاده سازی شود که درک آن توسـط برنامـه نـویس بـه راحتـی انجـام شـود و برنـام نـویس بـه آن انـس بگیـرد. *Stub* هـا معمـولاً درخواستهای خود را از *ORB* بوسیله واسط هـای خاصـی انجـام میدهنـد کـه توسط *ORB Core* مشخصی قابل استفاده و بهینه سازی میباشد. چنانچه بیشتر از یک *ORB* در دسترس باشد احتیاج به *Stub* هـای دیگـری داریـم کـه عملیـات متناظر را با *ORB* های دیگر انجام دهد. در اینگونه موارد برای *ORB* و نگاشـت زبان این مسئله خیلی مهم است که *Stub* هـا را بـا *Object Reference* مخصـوص خودشان پیوند دهد.

زبانهای برنامه نویسی *Object-Oriented* از قبیل *C++* و *Smaltalk* در ایـن قسـمت به واسط *Stub* نیازی ندارند.

۸-۲-۲ واسط *Dynamic Invocation*:

یک *Interface* همچنین این امکان را فراهم می آورد که بتوانیم *Object Invocation* های پویا تولید نماییم، و در آن به جای اینکه یک روتین *Stub* را *Call* نماید یک سری عملیات و یا *Operation* خاص را مشخص می نماید که بر روی *Object* های خاص عمل می نمایند و یک *Client* ممکن است *Object* معینی را جهت احضار (*Invoke*) معین نماید و یا حتی عملیاتی که باید صورت پذیرد و یا حتی مجموعه پارامتر هایی که جهت یک *Operation* در حین عملیات فراخوانی و یا ترتیب فراخوانی ها را معین سازد.

کد برنامه *Client* باید اطلاعاتی درزمینه عملیاتی که باید اجرا شود و انواع پارامتر هایی که باید *Pass* شوند را تأمین نماید. (که این اطلاعات ممکن است از یک سری منابع در زمان *Run-Time* و یا *Interface Repository* مشتق شده باشد.)

واسط درخواست پویا (*Dynamic Invocation*) با توجه به زبان برنامه نویسی که با آن کار خواهید کرد ممکن است نام دیگری به خود بگیرد.

۹-۲-۲ *Implementation Skeleton*:

برای یک زبان خاص و وابسته به *Object Adapter* یک واسطی وجود دارد که بوسیله آن قادر هستید به *Method* هایی که هر نوع *Object* را پیاده سازی و اجرا مینمایند دسترسی یافت. این واسط به طور معمول یک واسط *Up-Call* است که در این واسط های *Up-Call* ، روتینهایی که در *Object Implementation* نوشته شده است را از طریق *Skeleton*، *Call* می نماید و جهت فرستادن تأیید (*Confirm*) به واسط از آنها استفاده می نماید.

وجود *Skeleton* ها منافاتی با منتاظر بودن با *Client Stub* ها ندارد (*Client* ها قادر هستند از طریق واسط *Dynamin Invocation* نیز همین عملیات را انجام دهند.)

این امکان وجود دارد که یک *Object Adapter* ای بنویسیم که از *Skeleton* جهت *Invoke* و درخواست اجرای روتین ها استفاده ننماید.

۲-۲-۱۰ واسط *Dynamic Skeleton*:

این واسط ، واسطی است که اجازه اداره کردن پویای درخواستهای *Object* ای را میدهد. کد اجرایی باید توضیحی برای تمامی پارامترهای *Operation* به *ORB* تهیه نماید و *ORB* یک سری *Value* و یا مقدارهایی را برای هر پارامتر ورودی تهیه مینماید که مورد استفاده آنها در اجرای *Operation* ها است. کد اجرایی برای پارامتر های خروجی مقادیری را تهیه می نماید و بعد از اجرای *Operation* آنها را برای *ORB* ارسال می نماید.

ماهیت واسط *Dynamin Skeleton* اساساً با نگاشت یک زبان برنامه نویسی و یا *Object Adapter* به یکدیگر تفاوت دارد ولیکن معمولاً به صورت یک واسط *Up-call* عمل مینماید.

Dynamic Skeleton ممکن است بوسیله *Client Stub* ها و یا *Dynamic Invication Interface* درخواست شود. هر دوی این روشهای درخواست جواب یکسانی را ارائه خواهند داد.

۲-۲-۱۱ *Object Adapter* ها :

Object Adapter ها عمده ترین روش دسترسی *Object Implementation* را به سرویسهای داخل *ORB* فراهم می سازد. انتظار وجود چند *Object Adapter* که معمولاً در دسترس باشند و همچنین در دسترس بودن واسطهایی که برای یک سری انواع مخصوص *Object* ها مناسب هستند نیز الزامی به نظر می رسد.

سروسهایی که در *ORB* و از طریق *Object Adapter* ها ارائه میشود معمولاً شامل:

- تولید *Object* ها.
- شرح *Object Reference* ها.
- درخواست *Method* .
- امنیت در عملیات *Interaction* .
- *Object* و اجرای فعال و غیر فعال.
- نگاشت *Object Reference* ها به *Implementation* .
- ثبت *Implementation* ها.

وجود محدوده بزرگ از اشیاء با *Lifetime* ها و *Policy* ها و حالات اجرای گوناگون و خواص دیگر آنها باعث شده است که یک *ORB Core* قادر نباشد که با استفاده از یک *Interface* عمومی و موثر برای تمامی *Object* ها استفاده شود بنابر این از طریق *Object Adapter* ها این مهم برای *ORB* ها امکانپذیر میشود که گروه خاصی از *Object Implementation* ها را به یک *Interface* مناسب آنها ارسال نماید.

۲-۲-۱۲ واسط *ORB* :

ORB Interface واسطی است که مستقیماً با خود *ORB* کار می نماید که طریقه عملکرد آن برای تمامی *ORB* ها یکسان است و وابستگی به *Object Interface* و یا *Object Adapter* ندارد. به این علت که اکثر عملیات *ORB* از طریق *Object Adapter* و یا *Stub* ها و یا *Skeleton* ها و یا *Dynamic Invocation* انجام میشود تنها تعداد محدودی از عملیات باقی می ماند که بین *Object* ها مشترک است، که این عملیات باقی مانده هم برای *Client* ها و هم *Object Implementation* ها مفید هستند و می توان آنها را از طریق ارتباط مستقیم با *ORB* انجام داد.

۱۳-۲-۲ *Interface Repository*:

Interface Repository یک سرویسی است که به ارائه *Object* های ماندگاری می پردازد که وظیفه آنها ارائه اطلاعات *IDL* به صورتی است که در زمان *Runtime* قابل اجرا باشند. اطلاعات *Interface Repository* ممکن است که از طریق *ORB* فراخوانده شده و باعث اجرای یک درخواست شود. علاوه بر این استفاده از اطلاعاتی که در این قسمت وجود دارد برای یک برنامه این قابلیت را ایجاد می نماید که در زمان *Compile* یک برنامه چگونه با یک *Object* برخورد نماید و باعث میشود که متوجه شود که چه عملیاتی بر روی *Object* معتبر و امکان پذیر است و بنابر آن *Invocation* را تهیه نماید.

اضافه بر عملیات *Functioning* بر روی *ORB*، *Interface Repository* یک محل مشترک است که در آن اطلاعات اضافی مربوط به واسط هایی به اشیاء *ORB* ذخیره میشود. به عنوان مثال اطلاعات *Debugging*،*Library* های مربوط به *Stub* ها و یا *Skeleton* ها ، روتینهایی که قادر هستند *Object* های خاصی را شکل دهی و اجرا نمایند و... که ممکن است به *Interface Repository* پیوسته شده باشد.

۱۴-۲-۲ *Implementation Repository*:

Implementation Repository شامل اطلاعاتی است که این امکان را به *ORB* میدهند تا قادر باشد *Object Implementation* را آدرس دهی و فعال نماید. بنابر این بیشتر اطلاعاتی که در *Implementation Repository* قرار دارد مختص به یک *ORB* و یا محیط عملیاتی است، *Implementation Repository* مکان مناسب و مرسومی جهت ذخیره سازی اینگونه اطلاعات است. معمولاً نصب *Implementation* ها و کنترل روشهایی که مربوط به فعال سازی و اجرای *Object Implementation* ها است از طریق عملیات داخل *Implementation Repository* صورت می پذیرد.

همچنین *Implementation Repository* محلی است که در آن اطلاعات اضافی که در زمان اجرای *Object* ها احتیاج به ذخیره سازی آنها وجود داشته باشد از این مکان استفاده می شود ، به عنوان مثال اطلاعات مربوط به عملیات *debug*، *Administrative Control*، *Resource allocation*، *Security* و ... را میتوان نام برد.

۲-۳ نمونه هایی از ORB ها:

در داخل ساختار *CORBA* انواع گوناگونی از *Object Implementation* ها وجود دارند که قابل استفاده هستند، در این قسمت به شرح چند نمونه از آنها میپردازیم ، به این نکته توجه داشته باشید که یک *ORB* معین ممکن است *Option* ها و *Protocol* های متعددی را جهت ارتباط مورد استفاده قرار دهد.

۲-۳-۱ *Client –and Implementation resident- ORB :*

درصورتیکه یک مکانیزم مناسب ارائه شود *ORB* میتواند به صورت روتینهای *Resident* در *Client* ها و *Implementation* ها اجرا شود. همچنین *Stub* ها در *Client* نیز از مکانیزم *Location Trasparent IPC* و یا دسترسی مستقیم به موقعیت و *Location* سرویس استفاده می نماید. *Code Link* های همراه *Implementation* مسئول برپایی *DataBase* های مناسب جهت استفاده *Client* ها هستند.

۲-۳-۲ *Server-Based ORB :*

جهت متمرکز نمودن مدیریت بر روی *ORB* ها تمامی *Client* ها و *Implementation* ها با یک و یا چند *Server* که وظیفه آنها مسیر یابی است در ارتباط هستند و این وظیفه *Router* ها است که مسیر در خواست را به

Implementation معین سازند. *ORB* می تواند یک برنامه معمولی باشد بدور از لایه *Operating System* و می توان از یک *IPC* معمولی جهت ایجاد ارتباط با *ORB* استفاده نمود.

۲-۳-۳ *System-Based ORB:*

جهت بالا بردن امنیت و کارایی می توان *ORB* را به عنوان یکی از سرویسهای پایه ای سیستم عامل معرفی نمود، و *Object Reference* ها را به صورت فراموش نشدنی (*Unforgeable*) تهیه نمود و به این صورت از هزینه مستند سازی برای هر درخواست کاست. سیستم عامل قادر است موقعیت و ساختار *Clint* ها و *Implementation* ها را تشخیص دهد و بدین صورت سیستم عامل قادر خواهد بود تا *Optimization* های متعددی را انجام دهد به عنوان مثال از عملیات مرتب سازی (*Marshalling*) جهت ارسال و *Unmarshalling* در هنگام دریافت یک *Request* که بر روی یک سیستم واحد شامل *Client* و *Implementation* صورت میپذیرد، جلوگیری نماید.

۲-۳-٤ *Library-Based ORB:*

این سرویس مربوط به *Implementation* هایی است که میتوان آنها را به اشتراک اشتراک گذاشت و در واقع آن *Implementation* ممکن است یک *Library* باشد. در اینگونه موارد *Stub* میتواند متدهای واقعی باشد. این مطلب بدان معنی است که این امکان برای برنامه های *Client* وجود دارد که بتوانند به *Data* برای *Object* ها دسترسی داشته باشند و *Implementation* نیز به *Client* اطمینان دارد که اطلاعات در آن باقی می ماند و خراب نمی شود.

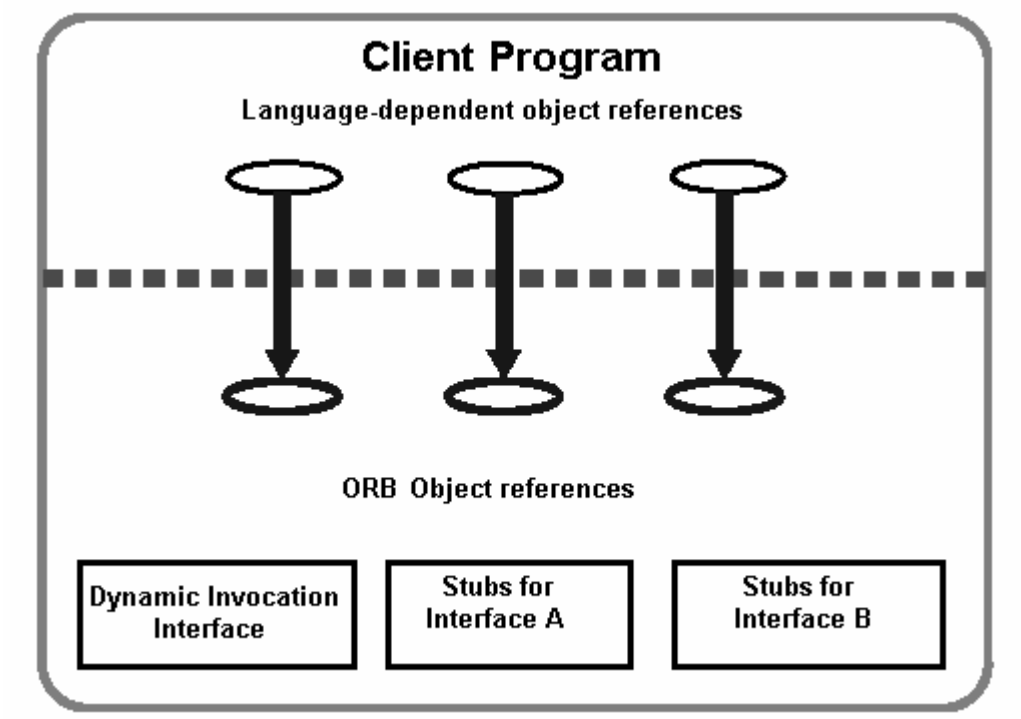
۲-٤ ساختار یک *Client*:

یک *Client* جهت دسترسی به یک *Object*، یک *Reference* به آن *Object* را در اختیار دارد. *Object Reference* یک *Token* است که میتوان آنرا در زمان *Request* به عنوان یک پارامتر به *Object* های دیگر انتقال داد.

مدیریت انتقال کنترل و انتقال داده به عهده *ORB* است و *ORB* از زمانی که به *Object Implementation* فرستاده میشود و تا زمانیکه به خود *Client* باز میگردد آنها را کنترل می نماید.

در صورتیکه *ORB* قادر به اتمام *Request* نباشد یک پیغام *Exception* را تهیه کرده و آنرا ارسال می نماید.که در متن برنامه میتوان این مسئله را پیشبینی نمود و این پیغامهای خطا را دریافت و نمایش داد، نمونه هایی از انجام این عملیات در فصل ششم آورده شده است.

به طور معمول یک *Client* یک روتین را در برنامه خود *Call* می نماید و آن باعث به وجود آمدن *Invocation* میشود و زمانیکه عملیات بر روی آن تمام گردد به *Client* باز می گردد. همانطور که در شکل ۲-٦ مشاهده می نمایید یک *Client* به *Stub* های یک نوع *Object* مخصوص به صورت روتینهای کتابخانه ای در برنامه خودش دسترسی می یابد. بنابر این برنامه *Client* قادر خواهد بود تا روتینها را به صورت معمول در زبان برنامه نویسی خود مشاهده نماید. تمامی *Implementation* ها یک نوع *Data Type* خاص جهت رجوع به *Object* ها تهیه می نمایند. (معمولاً از *Pointer* مبهم و یا *Opaque Pointer* استفاده می نمایند) و *Client* بعد از آن *Object Reference* را به روتینهای *Stub* میفرستد تا آنها *Invocation* را شروع نمایند. *Stub* ها با دسترسی به *Object Reference* و تماس با *ORB* قادر خواهند بود تا یک *Request* و یا *Invocation* (احضار) را اجرا نمایند.



(شکل ۲-٦)

یک سری از *Library* های دیگر نیز جهت اجرای یک *Request* بر روی *Object* ها در دسترس است ، به عنوان مثال زمانیکه یک *Object* در زمان *Compile* تعریف نشده باشد در این هنگام برنامه *Client* اطلاعاتی را جهت نام گذاری نوع آن *Object* فراهم می آورد و روتین را درخواست مینماید و با ایجاد یک سری از *Call* ها پارامترهای آنرا معین ساخته و سپس عملیات *Invocation* را آغاز می نماید.

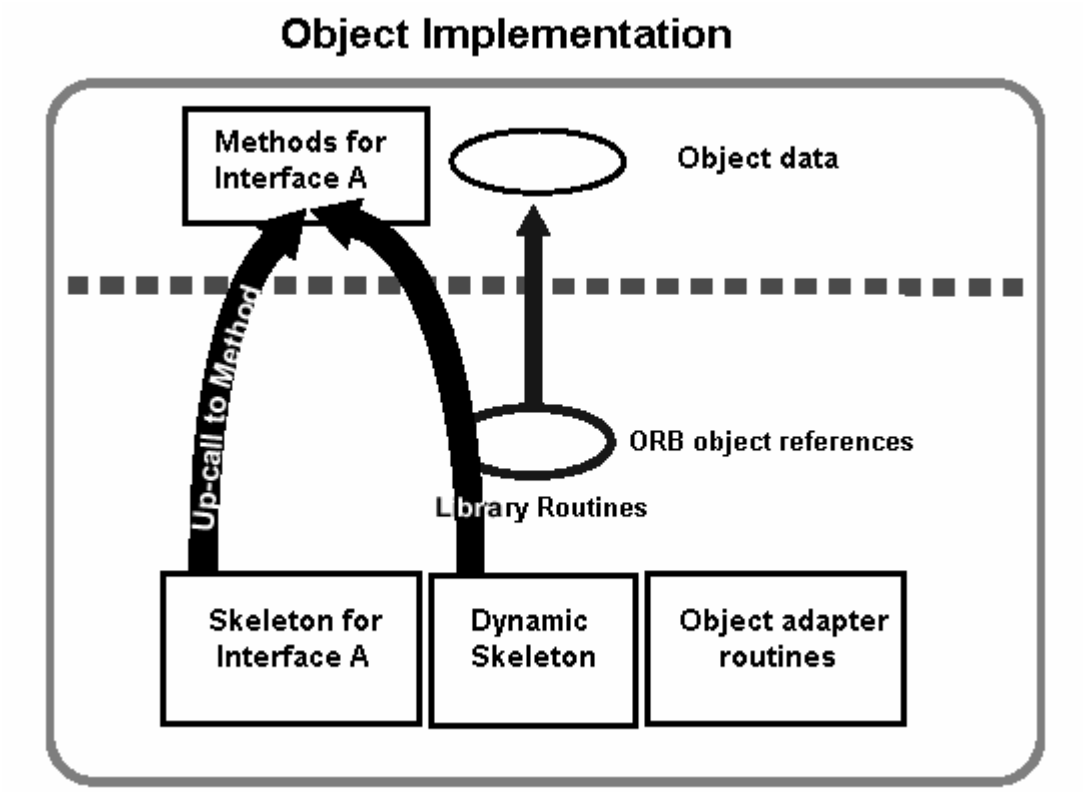
Client ها به طور معمول *Object Reference* ها را به صورت پارامتر های خروجی از درخواستهایی به *Object* های دیگر که *Reference* آنها را در اختیار دارند بدست می آورد. زمانیکه در یک *Client* عملیات *Implementation* نیز صورت بگیرد ، در این صورت *Object Reference* ها را از طریق پارامتر های ورودی در *Request* هایی که باید *Implementat* و اجرا شوند، بدست خواهد

آورد. *Object Reference* ها را نیز می توان به صورت رشته ای برگرداند و آنها را در یک فایل ذخیره نمود تا بتوان در آینده نیز به آنها دسترسی یافت و *ORB* قادر است که این رشته های کاراکتری را دوباره تبدیل به *Object Reference* نماید.

۲-۵ ساختار یک *Object Implementation*:

یک *Object Implementation* حالت و رفتار واقعی یک *Object* را تهیه مینماید. *Object Implementation* میتواند به صورت انواع گوناگونی ساخته شده باشد، از طرف دیگر تهیه *Method* ها جهت عملیات به عهده این قسمت است و یک *Implementation* معمولاً *Procedure* هایی را جهت فعال و غیر فعال نمودن *Object* ها تعریف می نماید و از اشیاء دیگر و یا امکانات بدون *Object* استفاده می نماید تا موقعیت *Object* را تداوم بخشد و دسترسی به آنها را کنترل نماید ، همانگونه که وظیفه اجرای *Method* ها را بر عهده دارد.

همانگونه که در شکل ۲-۷ مشاهده می نمایید، *Object Implementation* بوسیله راههای مختلف با *ORB* در تماس است تا بتواند عملیات شناسایی و یا ساختن یک *Object* جدید و یا بدست آوردن سرویس *ORB-dependent* را انجام دهد و این دسترسی را با استفاده از *Object Adapter* انجام می دهد.



(شکل ۲-۷)

به علت انواع گوناگونی که از *Object Implementation* در دسترس است این امکان وجود ندارد که ساختار *Object Implementation* را به صورت کامل مورد بررسی قرار داد.

زمانیکه یک درخواست و *Invocation* (احضار) اتفاق می افتد، *Object*، *ORB Core*، *Adapter*، *Skeleton* به سازماندهی مراحلی میپردازند که نتیجه آن اجرای *Method* های مناسبی است که در *Request* درخواست شده است. پارامترایی که به آن *Method* فرستاده می شود معین کننده آن *Object* ای است که درخواست شده است، *Method* قادر است تا مکان *Data* را جهت استفاده *Object* معین سازد. پارامترهای دیگر بر اساس تعریف *Skeleton* تهیه میشوند. زمانیکه یک *Method* کامل شود باعث برگشت پارامترهای خروجی و یا یک *Exception* به *Client* خواهد شد.

زمانیکه یک *Object* جدید ساخته میشود، *ORB* ممکن است که آنرا اعلام نماید و در این صورت *ORB* می داند که کجا می تواند *Implementation* مربوط به آن

Object را پیدا نماید. به طور معمول *Implementation* نیز خودش را به صورت *Object* های اجرایی در یک *Interface* بخصوص *Register* می نماید و معین می نماید که چگونه یک *Implementation* شروع شود. (درصورتیکه در حال حاضر درحال اجرا نباشد)

اکثر *Object Implementation* ها رفتار خودشان را با استفاده از امکاناتی که به *ORB* و *Object Adapter* اضافه مینماید، تهیه می نمایند.

۲-۶ ساختار یک *Object Adapter*:

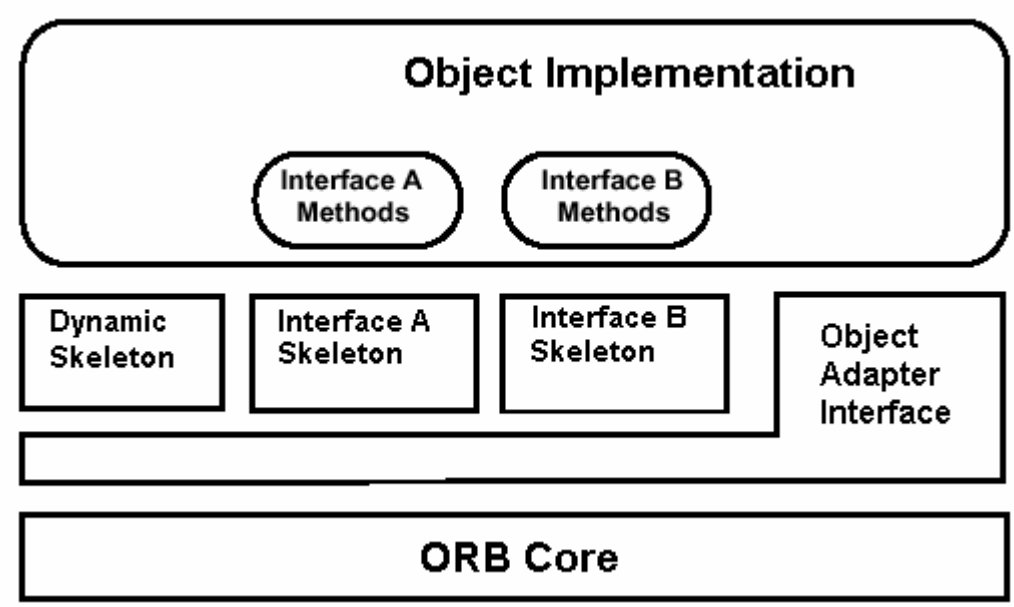
یک *Object Adapter* همانطور که در شکل ۲-۸ مشاهده می نمایید اصلی ترین واسطی است که بین *Object Implementation* و *ORB Services* قرار دارد و جهت دسترسی *Object Implementation* به سرویسهای *ORB* از آن استفاده می شود. یک *Object Adapter* واسط عمومی(*Public Interface*) را به *Object Implementation* ارسال می نماید و واسط های خصوصی (*Private Interface*) را به *Dynamic Skeleton* ارسال مینماید که بوسیله واسط خصوصی *ORB-dependent* ساخته شده است.

Object Adapter ها در مقابل توابع زیر مسئول هستند:

- تولید و شرح *Object Reference* ها
- درخواست متد (*Method Invocation*)
- امنیت *Invocation* ها
- فعال و غیرفعال نمودن *Object* و *Implementation*
- نگاشت *Object Reference* ها به *Object Implementation*
- ثبت *Implementation* ها

این توابع وظیفه استفاده از *ORB Core* و *Component* های دیگری که در موقع لزوم اضافه میشوند را بر عهده دارد.

گاهی اوقات *Object Adapter* موقعیت خود را نگهداری می نماید تا بتواند *Task* های مربوطه را انجام دهد. این امکان برای یک *Object Adapter* خاص وجود دارد که قادر باشد یک و یا چند عدد از مسئولیتهای خود را به قسمت *ORB*



Core واگذار نماید.

(شکل ۲-۸)

همانطور که در شکل بالا مشاهده می نمایید *Object Adapter* به صورت ضمنی شامل و درگیر درخواست متدها است، بنابر این واسط مستقیم از طریق *Skeleton* عمل می نماید، به عنوان مثال *Object Adapter* ممکن است که درگیر فعال سازی *Implementation* و یا تصدیق یک درخواست باشد.

Object Adapter بیشتر سرویسهایی که از *ORB* صادر می شوند را تعریف مینماید که *Object Implementation* به آن تعاریف وابسته است و *ORB* های مختلف ممکن است که سطوح مختلفی از سرویسها را تهیه نمایند و محیطهای عملیاتی ممکن است که خواصی را به طور ضمنی به وجود بیاورند، به همین سبب لازم است تا بقیه چیزهایی که لازم دارد توسط *Object Adapter* به آن اضافه شود. به عنوان مثال این مسئله بین *Object Implementation* ها بسیار

عادی است که بخواهند مقادیر خالی را در داخل *Object Reference* ها ذخیره نمایند تا بتوانند در زمان یک *Request*، *Object* ها را به راحتی شناسایی نمایند.
اگر *Object Adapter* به *Implementation* اجازه دهد که اینچنین مقادیری را اختصاص بدهد، زمانیکه یک *Object* جدید به وجود می آید، این امکان وجود دارد که بتوانیم *Object Reference* آنها را در آن *ORB* هایی که اجازه داریم ذخیره سازیم. درصورتیکه *ORB Core* این امکان را به ما ندهد، *Object Adapter* مقادیر را در قسمت *Storage* مربوط به خودش ذخیره می نماید و در زمان درخواست آنرا به *Implementation* ارائه خواهد داد. بوسیله *Object Adapter* ها این امکان برای *Object Implementation* فراهم می شود که قادر باشد به سرویسها دسترسی داشته باشد حتی اگر واسطی برای آن تهیه نکرده باشد. در غیر این صورت *Adapter* باید آنرا در قسمت بالای *ORB Core* اجرا نماید.
هر قسمت و نمونه از یک *Adapter* به خصوص باید واسط مشابه و سرویس مشابهی را برای *ORB* هایی که در حال اجرا هستند تهیه نماید.
همچنین برای تمام *Object Adapter* ها لازم نیست که واسط مشابه و یا عملیات مشابهی را تهیه نمود. بعضی از *Object Implementation* ها دارای نیازمندی های مخصوصی هستند، به عنوان مثال یک سیستم *DataBase* مبتنی بر *Object-Oriented* به صورت ضمنی مایل است تا تمامی هزاران *Object* خود را بدون حتی فراخوانی های منحصر بفرد به *Object Adapter* به ثبت برساند، در اینگونه موارد امکان ندارد که *Object Adapter* قادر باشد وضعیت هر *Object* را نگهداری نماید. بوسیله استفاده از یک واسط *Object Adapter* به سمت اینگونه *Object Implementation* ها هدایت خواهیم شد، و این امکان فراهم می شود که قادر باشیم از فواید یک *ORB Core* بخصوص استفاده نماییم تا در نهایت بتوانیم یک روش موثر جهت دسترسی به *ORB* را فراهم آوریم.

۲-۷ CORBA به Object Adapter ها احتیاج دارد:

انـواع گونـاگونی از *Object Adapter* هـا در دسـترس هسـتند و *Object Implementation* ها به واسطهای *Object Adapter* وابسته هستند.

بیشتر *Object Adapter* ها جهـت پوشـش دادن یـک تعـداد *Object Implementation* طراحی شده اند، بنابر این فقط زمانیکه یک *Implementation* به یک سرویس و یا *Interface* متفاوت احتیاج داشته باشـد یـک *Object Adapter* جدیـد مطـرح شـده و بکار گرفته می شود.

در این قسمت بـه تعریـف و شـرح *Object Adapter* هـای تعریـف شـده خـواهیم پرداخت :

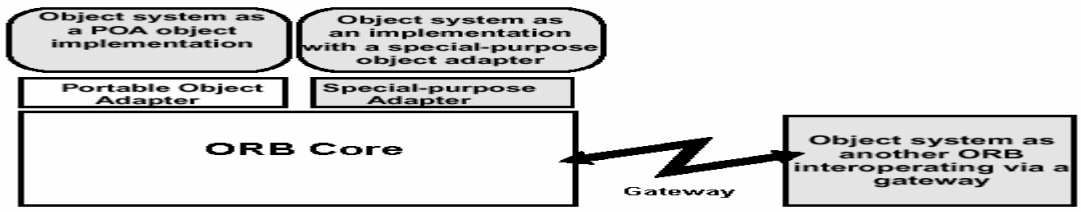
۲-۷-۱ *Object Adapter Portable:*

این مشخصه به تعریف *Object Adapter* قابل حمل می پردازد که میتـوان از آن در اکثر *Object ORB* ها و یا *Implementation* های قراردادی استفاده نمود. قصد *POA* همانطور که از نامش معلوم است این مهم است که بـه تهیـه *Object Adapter* ای بپردازد که قادر باشیم آنرا با *ORB* های مختلف استفاده نماییم، بـه این صورت که از حد اقل دوباره نویسی (*Rewriting*) جهت برقراری ارتبـاط بـا *Implementation* های مختلف بهره مند شویم. ایـن مشخصـه امکـان اسـتفاده از سرویسها را به روشهای گوناگون بـه مـا میدهـد ولـیکن بـا ایـده ارائـه مـدیر سیستم در شروع کار برنامه های *Server* هماهنگی ندارد. به عنوان مثـال کـد یک *Servant* برای یـک *Method Call* ، شروع می شود و خاتمه می پـذیرد و یـک *Servant* مجزا برای هر *Object* و یا یک *Servant Shared* برای تمام انواع *Object* هـا در نظر گرفته میشود. این بـه گروههـایی از اشـیاء اجـازه میدهـد کـه وابسـته باشند با توجه به این معنـی کـه بوسـیله انـواع گونـاگون *Object POA* بـه ثبـت رسیده باشند و به *Implementation* اجازه میدهد که تکنیکهای فعال سازی خـود را بتواند معین و مشخص نماید. در صورتیکه *Implementation* بعد از اجرای یک *Invokation* فعال نشود این وظیفه *POA* است که یکی از آنها را فعال نمایـد. *POA*

در *IDL* تعریف شده است ، بنابر این به صورت اتوماتیک بر روی زبانهای برنامه نویسی *Map* شده است و از قوانین آن زبان برنامه نویسی پیروی می نماید.

۲-۸ یکپارچگی بین سیستمهای شیی خارجی:

ساختار *ORB* های مشترک و یا *CORBA* جهت انجام رنج وسیعی از *InterOperation* ها با سیستمهای مبتنی بر *Object* طراحی شده است. همانند شکل ۲-۹ که مشاهده می نمایید، به این علت که در حال حاضر انواع گوناگونی از سیستمهای مبتنی بر *Object* موجود هستند و این میل مشترک وجود دارد که بتوانیم بوسیله *ORB* به *Object* های داخل آن سیستمها دسترسی داشته باشیم. آن دسته از سیستمهای *Object* ای که خودشان دارای *ORB* هستند، می توانند تا با استفاده از مکانیزمهایی با *ORB* های دیگر ارتباط برقرار نمایند.



(شکل ۲-۹)

دسته ای از سیستمهای مبتنی بر *Object* که می خواهند اشیاء خود را به داخل *ORB Object*، *Map* نمایند و *Request* هایشان را از طریق *ORB* دریافت کنند و همانند یک *Client* عمل نموده و *Request* را به خارج ارائه دهند نیز میتوانند با استفاده از این مدل به این هدف برسند.

در برخی موارد این امر که یک سیستم مبتنی بر *Object* به صورت یک *POA* *Object Implementation* عمل نماید غیر عملی است. یک *Object Adapter* باید برای اشیایی که مابین *ORB* ها انتقال می یابند و *Request* هایی که از طریق *ORB* های مختلف فرستاده می شود طراحی شده باشد. بعضی از سیستمهای *Object* ای ممکن است که بخواهند بدون مشورت با *ORB*، *Object* های خود را بسازند و شاید انتظار داشته باشند که درخواستها در داخل خودشان تولید شود و از طریق *ORB* صورت نگیرد، در اینگونه موارد یک *Object Adapter* مناسب به *Object* ها اجازه میدهد که به صورت مجازی در زمانیکه از داخل*ORB* عبور می نماید ثبت و *Register* شود.