سیر پیدایش تکنولوژی CORBA

گردآورنده : حمید یگانه

چالش های برنامه های توزیع شده

همزمان با رشد وب ، عمومیت یافتن استفاده از کامپیوترهای شخصی و پیشرفت های مهم در زمینه دستیابی به شبکه های با سرعت بالا ، پردازش های توزیع شده بشدت مورد توجه قرار گرفته است . در این نوع پردازش ها ، همواره می بایست بر دو اصل مهم تاکید و راهکارهای مناسب را دنبال کرد. اولین مسئله توجه به معماری مبتنی بر Component (عنصر) برای تولید نرم افزار و دومین مسئله نحوه تبین ارتباط بین عناصر ذیربط و تشکیل دهنده یک نرم افزار در محیط هائی با پردازش های توزیع شده است . همانگونه که قبلا" اشاره گردید، برنامه های مبتنی بر وب که خود نمونه ای از پردازش های توزیع شده می باشند از مدل N-Tier پیروی می کنند. کلید طلائی طراحی این نوع نرم افزارها ، توانائی نوشتن عناصر (اجزاء) بگونه ای است که از یکطرف امکان بکارگیری آنها بسادگی در لایه ها و حتی چندین برنامه فراهم شده و از طرف دیگر امکان ارتباط این عناصر با یکدیگر صرفنظر از زبان برنامه نویسی استفاده شده و سایر موارد ذیربط ، فراهم گردد. ما می بایست جعبه های سیاهی را طراحی کنیم که صرفنظر از ماهیت درون هر یک ، قادر به استفاده از توان آنها در بخش یا بخش های از یک و یا چندین نرم افزار باشیم .

سیر تکامل پردازش های توزیع شده

از گذشته تا کنون دو مدل اساسی در پردازش های توزیع شده مورد توجه قرار گرفته است . RPC(Remote Procedure Call) و Client Server . ارتباطا ت مبتنی بر RPC ، نسبت به Client Server دارای قدمت بیشتری بوده و بعنوان شاه کلید برنامه های توزیع شده در محیط یونیکس مطرح بوده است . یونیکیس یکی از اولین سیستم های عامل در زمینه استفاده کامل از امکانات ارتباطی پروتکل TCP/IP است . پروتکل فوق بهمراه استانداردهای مربوطه آن بعنوان ستون فقرات شبکه های مبتنی بر یونیکس مطرح بوده است . مثلا؛ استاندارد DNS(Domain Name System) جهت همترازی آدرس یک کامپیوتر و نام آن ، FTP(File Transfer Protocol)، امکانی جهت تبادل فایل ها و پروتکل TelNet ، ارائه دهنده تسهیلات

لازم جهت دستابی به ترمینال ها . اگر امروز ما در دنیائی زندگی می کنیم که پروتکل TCP/IP محور اساسی گفتمان در شبکه های کامپیوتری است ، بیش از بیست سال قبل یونیکیس چنین وضعیتی را دارا بوده است . برنامه نویسان تحت یونیکیس بخوبی از توانائی های آن برای نوشتن برنامه های توزیع شده استفاده کرده اند. برنامه نویسان فوق از ارتباطات مبتنی بر Socket جهت نیل به اهداف خود استفاده می کردند. بر اساس رویکرد فوق ، اگر برنامه ای قصد ارتباط با برنامه دیگری را داشت ، بر اساس آدرس TCP/IP و یک شماره پورت ، یک لینک با آن برنامه ایجاد می کرد.این رویکرد تا مدت ها بعنوان یک راه حل مناسب جهت طراحی و اجرای برنامه های توزیع شده حضوری موفق در عرصه برنامه های توزیع شده داشت .پس از مدت زمانی رویکرد فوق با دو چالش جدی مواجه گردید : 1 – برنامه نویسان مجبور بودند که نام و یا آدرس سرویس دهنده و شماره پورت مورد نیاز جهت برقراری ارتباط را در Source برنامه ها مستقیما" مشخص نمایند . 2 – برنامه نویسان گوناگون می توانستند از پورت های یکسان برای برنامه های متفاوت استفاده نمایند .بدیهی است در چنین حالتی Conflict (تعارض) بین شماره پورت ها امری اجتناب ناپذیر بود. بمنظور برخورد با دو چالش فوق ، کمیته یونیکیس مفهوم ارتباطات مبتنی بر RPC را مطرح کرد. بر اساس رویکرد فوق برنامه ای با نام Portmapper بر روی هر سرویس دهنده اجرا و بین برنامه های اجرائی بر روی سیستم ها ی متفاوت ، حکمیت خواهد کرد. بر این اساس هر برنامه بجای تلاش جهت ایجاد یک ارتباط با یک پورت خاص بر روی یک سیستم ، درخواست خود را برای Portmapper ارسال و وی مسئول ایجاد اطلاعات لازم جهت برقراری ارتباط خواهد بود. راه حل فوق با اینکه مسئله ارتباطات بین پردازه های توزیع شده را بگونه ای حل کرده بود ، ولی در رابطه با فورمت داده های مبادله شده بین برنامه ها سکوت اختیار کرده بود.در این راستا تکنولوژی دیگری با نام XDR(eXternal Data Representation)، روشی را جهت تشریح داده های یک برنامه برای برنامه دیگر تعریف نمود. می توان گفت که XDR پیش کسوت XML است . RPC یک روش نسبتا" ساده ، انعطاف پذیر برای پردازش های توزیع شده را ارائه کرد. شاید این سوال مطرح شود که چرا تکنولوژی فوق نتوانست تسلط و چیرگی خود را بر روی پردازش های مبتنی بر Client/Server ادامه و مستمر نماید؟

مدل ارتباطی RPC تسلط مقتدر خود را در دنیای یونیکس بخوبی ادامه داد ولی با پیدایش و نیاز به ارتباطات مبتنی بر Client Server (PC-to-server) با یک مانع جدی مواجه گردید. مشکل اساسی پروتکل هائی بوند که در اغلب سیستم های Client Server استفاده می

گردید.پروتکل TCP/IP استاندارد تمامی تولیدکنندگان نبود و هر تولیدکننده پروتکل های اختصاصی خود را داشت مثلا؛ شرکت ناول از IPX و شرکت ماکروسافت از NetBEUI استفاده می کردند.چون پروتکل TCP/IP بعنوان استاندارد در دنیای سرویس دهندگان مبتنی بر PC ، هنوز مطرح نشده بود و ارتباطات مبتنی بر RPC گزینه ای مناسب در این زمینه نبودند، چراکه ستون فقرات تکنولوژی فوق بر پروتکل TCP/IP استوار بود. بنابراین در مقطعی با رشد شدید روش های ارتباطی نظیر ODBC برای دستیابی به بانک های اطلاعاتی ، صف بندی پیامها برای تبادل همزمان ، IPC و ... مواجه شدیم . پس از اینکه پروتکل TCP/IP به میدان Client Server قدم گذاشت ، مجددا" ارتباطات مبتنی بر RPC مورد توجه قرار گرفت . در این راستا تکنولوژیهای ارتباطی متفاوتی نظیر : OLE ، Com ، Dcom ، Corba ، J2EE ، Java Enterprise، Tuxedo و... مطرح گردیدند. تمامی تکنولوژیهای فوق بدنبال ارائه تسهیلات ، انعطاف پذیری و اعتماد سازی بیشتر در برنامه های توزیع شده بودند. مطلب فوق شاید مهمترین دلیل رویکرد شرکت های عظیم نرم افزاری جهت ارائه یک ساختار استاندارد برای تولید این عناصر باشد. دو مدل استاندارد عمده تاکنون ، در این زمینه مطرح و ارائه شده است .

DCOM (Distributed Component Object Model)

CORBA (Common Object Request Broker Architecture)

مدل های استاندارد شده در این زمینه می باشند.

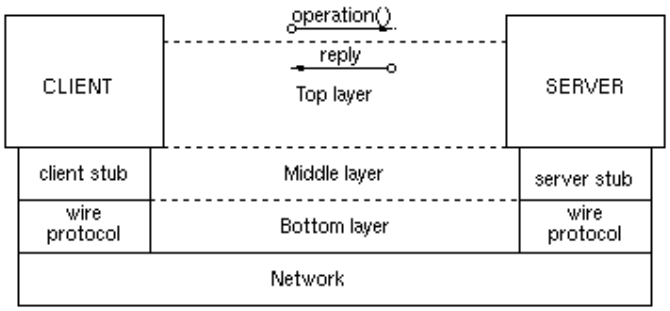
تعاریف و اصطلاحات

- Interface . مجموعه ای از متدها که مسئولیت ارائه عملیات وارائه قابلیت ها را برعهده خواهند داشت .
- Object class or class . نام مورد نظر برای پیاده سازی یک و یا چندین اینترفیس
- Object (or object instance . نمونه ئی از برخی کلاس ها
- Object server . پردازه ای که مسئولیت ایجاد و میزبانی نمونه هائی از برخی کلاس ها را برعهده دارد.
- Client . پردازه ای است که متدی ازیک شی را فرا می خواند

مقایسه DCOM و CORBA

دو استاندارد فوق از مدل سرویس گیرنده / سرویس دهنده برای ارتباطات خود استفاده می نمایند. در این راستا سرویس گیرنده بمنظور اخذ یک سرویس ، متدی را توسط یک شی راه دور که بعنوان یک سرویس دهنده در مدل سرویس گیرنده / سرویس دهنده رفتار می نماید ، فرا می خواند.سرویس ارائه شده توسط سرویس دهنده بصورت یک شی کپسوله می گردد. اینترفیس مربوط به شی توسط یک زبان تعریف اینترفیس (IDL) مشخص خواهد شد. اینترفیس های تعریف شده در یک فایل IDL بعنوان یک پیمان ارتباطی بین سرویس دهنده و سرویس گیرنده ایفای وظیفه می نماید.سرویس گیرندگان با فراخوانی متدهای تشریح شده در IDL با سرویس دهنده ارتباط برقرار می نمایند.در این راستا مدل و نحوه پیاده سازی شی از دیدگاه سرویس گیرنده مخفی نگاه داشته می گردد. برخی از مفاهیم برنامه نویسی شی گراء نظیر : Encapsulation ,Polymorphism ,Single inheritance در سطح IDL ارائه شده است . تکنولوژی CORBA در سطح IDL از ویژگی Multiple inheritance حمایت می کند.

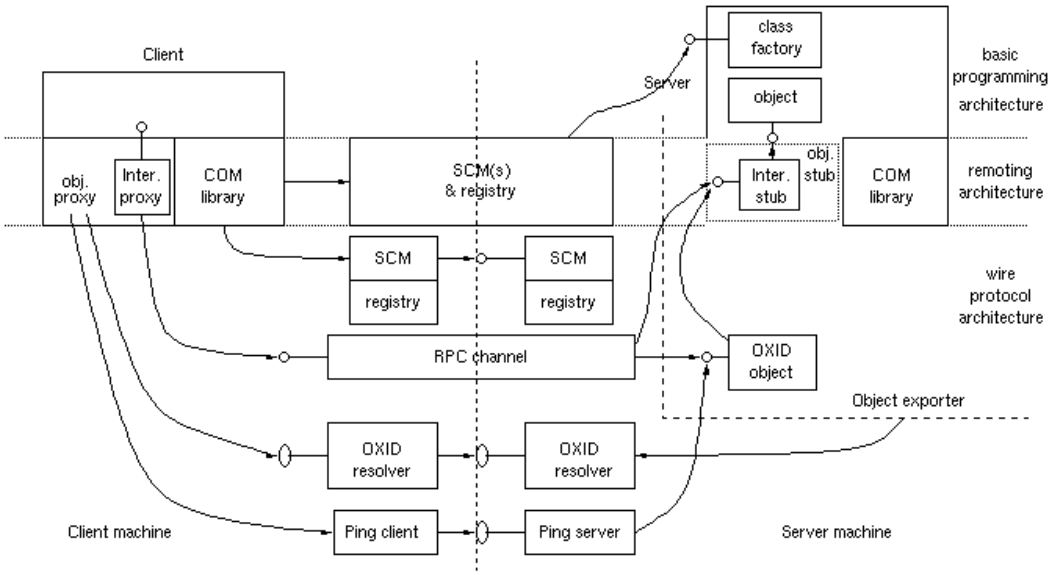
مدل ارتباطی بین یک پردازه سرویس گیرنده ویک شی سرویس دهنده در تکنولوژی های CORBA,DCOM ، تابع مدل مبتنی بر شی RPC است . شکل زیر ساختار RPC را نشان می دهد.



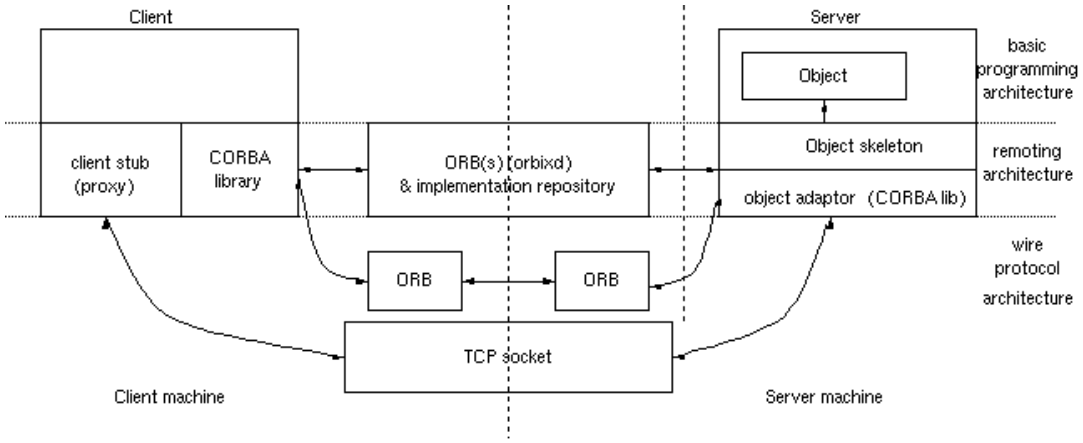
برای فراخوانی یک تابع راه دور ، سرویس گیرنده یک فراخوانی به Client Stub را انجام خواهد داد. Stub پارامترهای مربوط به فراخوانی تابع را در یک پیام درخواستی بسته بندی و یک Wire Protocol را بمنظور حمل پیام برای سرویس دهنده فرا می خواند. پس از حمل ، در سرویس دهنده ، پیام در اختیار Server stub گذاشته شده تا عملیات بازگشائی بسته ارسالی انجام و زمینه فراخوانی متدهای مربوط به شی مورد نظر فراهم گردد.در تکنولوژی DCOM ، وClient Stub بعنوان Proxy و Server Stub بعنوان Stub نامیده می شوند. در تکنولوژی CORBA ، و stub Client بعنوان stub و Server stub بعنوان Skeleton نامیده می گردد.

تکنولوژی **COM** . مهمترین ویژگی تکنولوژی فوق قابلیت استفاده مجدد و ارتباط متقابل برای عناصر(اشیاء) توزیع شده است . بدین ترتیب پیاده کنندگان نرم افزار این امکان را پیدا خواهند کرد تا با در کنار هم قرار دادن این عناصر و استفاده متعدد از آنان (حتی اگر تولیدکنندگان آنها متفاوت باشند) ، قادر به خلق آثار ماندگار در سریعترین زمان ممکن و متکی بر اصول مهندسی نرم افزار باشند. تکنولوژی *Com* بصورت ناگهانی مطرح نگردید و ریشه در تلاش هائی دارد که از مدت ها قبل بعنوان یک نیاز مطرح شده بود ، معرفی تکنولوژی *OLE(Object Linking & Embedding)* در سال ،1991 اولین تلاش در این زمینه بود که توسط شرکت مایکروسافت برای ارتباط و پیوستگی بین مستندات مجموعه برنامه های آفیس مطرح گردید. حوزه عملکرد تکنولوژی فوق بر روی مستندات (*Documents*) متمرکزاست. در ادامه شرکت مایکروسافت به این نکته پی برد که تکنولوژی فوق نباید صرفا" متمرکز بر روی مستندات باشد و می تواند عملکردی جامع تر را داشته باشد. بدین منظور نسخه شماره 2 ، *OLE* در سال1995 مطرح گردید و این نسخه در ادامه تمامی عناصر و اجزای موجود در محیط ویندوز را شامل گردید و بدین ترتیب COM مطرح شد. در اوایل ، تکنولوژی فوق در رابطه با عناصر و اجزای توزیع شده امکانات قابل توجه ای ارائه نکرده بود .شاید یکی از مهمترین دلایل آن ، عدم عرضه یک سیستم عامل شبکه ای از طرف مایکروسافت تا آن زمان بود.همزمان با عرضه ویندوز 95 و ویندوز NT در سال 1996 و مطرح شدن امکانات شبکه ای و ضرورت توزیع ، اجراء و ارتباط بین عناصر توزیع شده، تکنولوژی *DCOM(Distributed COM)* مطرح گردید.سرانجام در سال 1997

نسخه توسعه یافته این تکنولوژی با نام *COM+* توسط شرکت مایکروسافت ارائه گردید.شکل زیر معماری بکارگرفته شده درتکنولوژی DCOM را نشان می دهد.



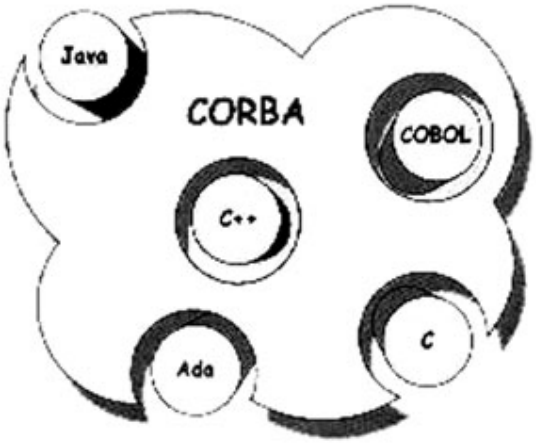
تکنولوژی **CORBA** . همزمان با گرایش بسمت طراحی و پیاده سازی نرم افزارهای متکی بر مدل N-Tier از یکطرف و نیاز شدید به پیاده سازی نرم افزار های متکی بر وب ، ضرورت توجه و بازنگری در نحوه طراحی و پیاده سازی عناصر توزیع شده مورد اهتمام جدی شرکت های بزرگ نرم افزاری قرار گرفت . شرکت ماکروسافت در این زمینه منادی تکنولوژی های COM/DCOM/COM+ ، Internet Explorer و ActiveX و سایر شرکت ها " کوربا" را مطرح کردند . اولین نسخه تکنولوژی فوق درسال 1992 توسط OMG که بالغ بر هفتصد عضو دارد، ارائه گردید. آخرین نسخه آن (نسخه شماره 2) در سال 1996 عرضه شده است . عملکرد تکنولوژی فوق شباهت زیادی به DCOM دارد. شکل زیر معماری بکارگرفته شده در تکنولوژی فوق را نشان می دهد.



بهرحال هدف تمامی تکنولوژی های عرضه شده درزمینه پردازش های توزیع شده ،ارائه امکانات و استانداردهای لازم برای تولید و بکارگیری عناصر بمنظور ایجاد نرم افزارهای توزیع شده بوده تا از این طریق زمینه استفاده از سرویس ها و خدمات در بر نامه های توزیع شده بصورت محلی و یا از راه دور فراهم گردد.

CORBA ؛ استانداردی برای سیستم‌های توزیع‌یافته

اشاره :



CORBA **سرنام واژه‌های** Common Object Request Broker Architecture **استانداردی برای نرم‌افزارهای ترکیبی** (componentry) **است که توسط شرکت** object managementgroup) OMG)**طراحی شده و پشتیبانی می‌گردد. این استاندارد** API، **پروتکل‌های رابطه‌ای ومدل‌های اطلاعاتی** object/service **را تعریف می‌کند که می‌تواند نرم‌افزارهای ناهمگون (نوشته شده به زبان‌های متفاوت) را به هم مرتبط‌سازد. بنابراین با استفاده از** CORBA **می‌توانیم ازشی** (object) **در پلتفرم‌های توزیع‌یافته** (distributed)**، به صورت مشترک استفاده کنیم؛ بدون نگرانی از این که شی در چه**

موقعیت مکانی قرارگرفته یا به چه پلتفرمی متعلق است. CORBA می‌تواند کدهای نوشته شده (در برخی زبان‌های برنامه‌نویسی) را بسته بندی و به آن اطلاعاتی از قبیل توانایی اجرایی کدها و چگونگی اجرای آن‌ها را اضافه نماید. به‌طوری که این کدهای بسته بندی‌شده (یا شی‌ها) بتوانند از برنامه‌های دیگر (یا حتی شی‌های CORBA) که تحت شبکه قرار دارند، اجرا شود. CORBA برای تعیین‌کردن اینترفیس‌هایی که به دیگران ارائه می‌کند، از یک زبان رابطه‌ای به نام IDL استفاده می‌کند و از طریق این زبان می‌تواند کدهای اجرایی (مثلا جاوا یا C++) را بشناسد. امروزه می‌توان زبان‌های معروفی مثل جاوا، Smalltalk، C++،Python، آدا، سی و لیسپ را با استفاده از این زبان (IDL) به هم مرتبط ساخت و از توانایی‌های آن‌ها استفاده نمود. قسمت اول این مقاله در آغاز نرم‌افزارهای توزیع یافته را معرفی کرده و دلایل نیاز به آن را مشخص می‌نماید. سپس این نرم‌افزارها را با سیستم‌های متمرکز یا توزیع یافته مقایسه می‌کند و در ادامه با طرح این سؤال که چرا به CORBA نیاز داریم، CORBA را معرفی و ساختار آن را تشریح می‌کند و بعد چگونگی کار این استاندارد را توضیح می‌دهد. درپایان نیز مزایا و ضعف‌های آن را به اختصار بیان می‌نماید.

نرم افزارهای توزیع‌یافته (Distributed)

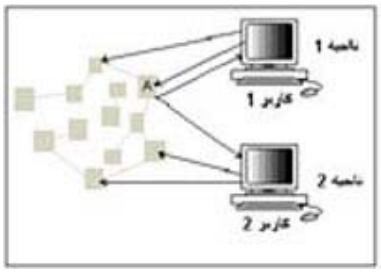
همان‌طور که اشاره شد، کار اصلی CORBA ارائه استانداردی برای ایجاد و اجرای نرم افزارهای توزیع‌یافته می‌باشد. اگرچه توزیع‌یافته بودن نرم‌افزارها می تواند مشکلاتی برای تولید کنندگان و کاربران آن داشته باشد، ولی به‌دلیل غیرمتمرکز بودن اطلاعات نرم‌افزارها، پراکندگی کاربران، و نیاز به اجرای برخی برنامه‌ها با استفاده از بیش از یک پردازنده ، راه دیگری جز استفاده از این نرم‌افزارها نیست.

برخی از نرم‌افزارها به دلیل این که کاربران آن باید از طریق این نرم افزارها با هم در ارتباط باشند، بر روی چند کامپیوتر اجرا می شوند. بدین ترتیب که هر کاربر قسمتی از این نرم افزار را برروی کامپیوتر خود اجرا می‌کند و شی‌های مشترک بر روی یک چند سرور اجرا می شوند. شکل 1 نشان می دهد که دو کاربر از شی‌های به اشتراک گذاشته شده استفاده می کنند. همان‌طور که در این شکل مشخص شده‌است، دو کاربر یکی در ناحیه 1 و دیگری در ناحیه

2 بدون در نظر گرفتن موقعیت جغرافیایی، به شی‌های گوناگونی دسترسی‌دارند و در برخی موارد، بر حسب نیاز به یک شی (object A) دسترسی خواهند داشت.

همان‌طور که قبلا اشاره شد، یکی دیگر از دلایل نیاز به نرم‌افزارهای توزیع‌یافته، پراکندگی اطلاعات می‌باشد. برخی از نرم‌افزارها، از آنجایی که به اطلاعاتی نیاز دارند که برروی چند کامپیوتر است (به‌دلایل امنیتی و حفاظت اطلاعات شخصی) و صاحب اطلاعات فقط اجازه دسترسی از راه دور را به کاربران می‌دهد، باید درچند کامپیوتر اجرا شوند. از طرف دیگر، نیاز به نرم‌افزارهای توزیع‌یافته در سیستم‌هایی که می‌خواهند از مزایای استفاده از چند پردازنده (به‌صورت موازی) استفاده کنند، محسوس به نظر می رسد.

تا این جا دلایل نیاز دنیای امروز به نرم افزارهای غیرمتمرکز توضیح داده شد. ممکن است از خود سؤال کنید چه فرقی بین سیستم‌های متمرکز و توزیع‌یافته وجوددارد؟ جدول 1 برخی از این تفاوت‌ها را نشان می‌دهد.



شکل 1- استفاده مشترک از شی‌ها

دلایل نیاز به CORBA

دلایل نیاز به CORBA را می‌توان در پنج‌مورد زیر فهرست کرد:

1- گوناگونی سیستم‌عامل‌ها

2- تنوع پلتفرم‌های سخت افزاری

3- گوناگونی پروتکل‌های شبکه (برای مثال ATM،TCP/IP و Ethernet)

4- گوناگونی زبان‌های برنامه‌نویسی

(C ,C++ ,Java ,COBOL ,Basic ,Perl ,Smalltalk)

5- نیاز ارتباط یافتن سیستم‌های جدید با سیستم‌های قدیمی

ویژگی	سیستم های متمرکز	سیستم های توزیع‌یافته
امنیت	زیاد	کم
در سیستم‌های توزیع‌یافته که object ها درسیستم‌های مختلفی قرار دارند، به امنیت بیشتری نسبت به سیستم های		

محلی نیاز است .		
مشکلات و ازکارافتادگی شی‌ها	همه شی ها با هم خراب می‌شوند	شی‌ها جداگانه خراب می‌شوند
اگر پردازنده‌ای که دو object را اجرا می‌کند دچار مشکل شود، هر دو object دچار مشکل خواهند شد.		
دسترسی همزمان	فقط با استفاده از با چند Thread	دارد
برنامه‌ها بر روی سیستم‌های متمرکز، به‌صورت پیش فرض از یک thread استفاده می‌کنند. ولی در سیستم‌های توزیع‌یافته به‌صورت پیش فرض از thread استفاده می‌شود.		
ارتباط بین سیستم ها	سریع	آهسته
از آن‌جایی که ارتباط بینobject ها دریک پردازنده سریع تر از ارتباط بینobject ها درچند پردازنده می باشد، پیشنهاد می‌شود اگر ارتباط تنگاتنگی بین شی‌ها وجود دارد، از سیستم‌های متمرکز استفاده شود .		
جدول 1- جدول مقایسه‌ای سیستم‌های متمرکز و توزیع‌یافته		

با استفاده از CORBA، دیگر دلیلی برای نگرانی در مورد این‌که از چه زبان برنامه‌نویسی، سیستم‌عامل یا پلتفرمی استفاده می‌کنید، وجود ندارد. از آن‌جایی که CORBA با مکانیزم ساده‌ای می‌تواند سیستم‌های مختلف از هر نوع و هراندازه را به هم متصل کند، درخیلی از موقعیت‌ها بسیار مفید به نظر می‌رسد. یکی از مصارف CORBA در سرورهایی است که با تعداد زیادی کلاینت و ترافیک سنگین اطلاعاتی باید به خوبی کارکنند. استفاده از CORBA فقط محدود به نرم‌افزارهای بزرگ نمی‌شود و حتی درسیستم‌های بلادرنگ نیز کاربرد دارد.

CORBA و ساختار آن

گروه OMG در سال 1989 با هدف ایجاد و پشتیبانی استاندارد برنامه‌های شی‌گرای غیر متمرکز به‌وجود آمد. البته، این گروه در واقع تولید کننده هیچ نرم‌افزاری نیست، بلکه مشخصات (Specification) این برنامه‌ها را با استفاده از فناوری و نظرات اعضا مشخص می‌کند.

این گروه شامل بیش از 700 شرکت و سازمان تولیدکننده فناوری سیستم‌های توزیع‌یافته، بانک‌های اطلاعاتی، پلتفرم‌ها و سازندگان بزرگ نرم‌افزار می‌باشد. تلاش اصلی این گروه در راستای تعریف امکانات و ساختار لازم برای سیستم‌های شی‌گرای غیرمتمرکزاست و یکی ازمکانیزم‌های اصلی در این رابطه که هسته مرکزی این گروه نیز می‌باشد، Object Request Broker (ORB) نام دارد. سال 1991 را می‌توان سال تولد اولین نسخه CORBA (Common Object Request Broker Architecture) توسط این گروه نامید.

CORBA ساختار استانداردی برای سیستم‌های توزیع‌یافته شی‌گراست که می‌تواند سیستم‌های ناهمگون و پراکنده را به هم مرتبط سازد. همان‌طور که گفته شد، CORBA ساختاری برای objectهای توزیع‌یافته مشخص می‌کند. محوراصلی این ساختار بر اساس درخواست سرویس از شی‌های توزیع‌یافته می‌باشد. این شی‌ها سرویس‌ها را از طریق اینترفیس‌هایشان که به زبان Definition Language (IDL Interface) مشخص شده‌اند، ارائه می‌دهند. Objectهای غیر متمرکز را می توان از object referencesها که توسط اینترفیس‌های IDL تایپ شده اند، شناسایی کرد.

همان‌طورکه در شکل 2 می‌بینید، درخواست‌کننده سرویس (کلاینت) شماره مرجع object شی سرویس‌دهنده را از طریق اینترفیس آن در اختیاردارد (Interface A) و ORB درخواست کلاینت را به شی سرویس‌دهنده و جواب درخواست را به شی درخواست کننده می رساند.

ORB درواقع یک سرویس توزیع‌یافته است که به درخواست شی‌های دوردست (remote object) رسیدگی می‌کند. بدین ترتیب که این شی‌ها را در شبکه قرار می دهد، با شی‌ها ارتباط برقرار کرده، و درخواست سرویس را مطرح می‌کند. سپس برای جواب سرویس درخواستی صبر می‌کند و سرویس موردنظر را به کلاینت درخواست کننده منتقل می‌کند. ORB این کار را بدون توجه به موقعیت مکانی سرویس‌دهنده و زبان برنامه درخواست‌کننده

انجام می‌دهد. نیازی نیست که کلاینت درخواست کننده به زبان CORBA تقاضای خود را مطرح کند. ORB زبان برنامه درخواست‌کننده را شناسایی کرده (برای اکثر زبان‌های برنامه نویسی) و این زبان را ترجمه می کند.

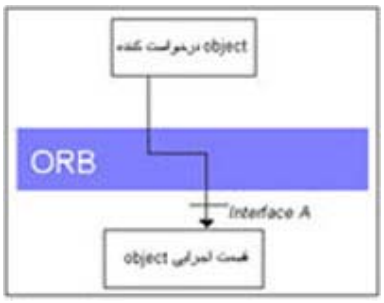
کارکرد CORBA

نرم‌افزارهایی که از CORBA استفاده می کنند، حاوی شی‌هایی‌هستند. البته معمولا از یک شی چندین نمونه(Instances) از یک نوع وجود دارد. مثلا در یک سایت تجارت الکترونیک، نمونه سبدهای خرید زیادی وجود دارد که با وجودی که همه آن‌ها کاری یکسان انجام می‌دهند، هر کدام به مشتریان متفاوتی تعلق دارد و اطلاعات خرید آن مشتری را ذخیره می کنند.

برای هر نوع از شی‌ها، مثلا سبد خرید می‌باید یک اینترفیس در OMG IDL تعریف کنیم که نشانگر سرویس‌هایی است که شی سرور در اختیار کلاینت‌های تقاضاکننده قرارمی‌دهد. هر کلاینت که تقاضای اجرای عملیاتی از شی را دارد، باید از این اینترفیس‌ها استفاده و آرگومان ها را کنار هم چیده و ارسال‌نماید. وقتی که این تقاضا به شی سرویس‌دهنده می‌رسد، این شی با استفاده ازاینترفیس مشابه آن، عملیات درخواستی کلاینت را انجام می دهد و آرگومان ها را جدا سازی می کند. پس از انجام عملیات درخواستی، جواب درخواست، بسته بندی شده و با استفاده از تعریف اینترفیس از همان مسیری که آمده بود، بر می گردد. چنان که گفته شد، اینترفیس IDL برای اکثر زبان‌های برنامه‌نویسی از جمله IDLscript , Python, Lisp, Ada , Smalltalk , COBOL, java, C++, C و از طریق استاندارد OMG تعریف شده است.

جداسازی اینترفیس و قسمت اجرایی که توسط OMG IDL مهیا شده است، یکی از مهم‌ترین پایه‌های اصولی CORBA می باشد و ایجاد اینترفیس برای هر شی در CORBA ضروری است. ازطرف دیگر قسمت اجرایی هر شی از دید سیستم مخفی می‌ماند و کلاینت از آن چیزی نمی‌داند.

در نتیجه کلاینت‌ها فقط می‌توانند از طریق اینترفیس‌ها به سرویس‌ها دسترسی داشته و تنها سرویسی را اجرا کنند که توسط اینترفیس‌های IDL معرفی شده اند.

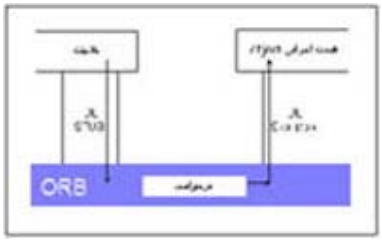


شکل 2- ساختار ساده CORBA

همان‌طور که در شکل 3 نشان داده شده‌است، یک کلاینت توسط اینترفیس IDL تقاضای سرویس می‌کند. برای این کار IDL باید به کلاینت stubs و object skeletons کامپایل شود. stubs و skeletonsها درواقع نقش نمایندگان کلاینت و سرور را بازی می‌کنند و از آن جایی که IDL تعریف مشخصی برای اینترفیس ها دارد، اگر حتی این دو نماینده به دو زبان متفاوت یا حتی برروی دو ORB از دو شرکت مختلف اجرا شوند، می‌توانند بدون مشکل به هم مرتبط شوند.

همان‌طور که اشاره شد، در CORBA هر نمونه ازاشیا دارای یک کد یکتا (unique) به نام object reference هستند. کلاینت‌ها از این اشیا برای هدایت درخواست‌هایشان و شناساندن خود به ORBها استفاده می‌کنند. اگرچه کلاینت‌ها درخواست‌هایشان را به نماینده سرور یا IDL stub منتقل می‌کنند نه به خود سرور، چنین به نظر می‌رسد که کلاینت مستقیما به سرور دسترسی دارد. این درخواست سپس از طریق ORB و skeleton به قسمت اجرایی می‌رسد و این قسمت به این درخواست رسیدگی می‌کند. شکل 4 مکانیزم درخواست سرویس ازراه دور را نشان می دهد.

برای ایجاد ارتباط و درخواست از شی دوردست، در اولین قدم، کلاینت object reference شی موردنظر را از راه‌های گوناگونی مثل Naming Service به‌دست‌میآورد و با استفاده از مکانیزم درخواست محلی، درخواست خود را مطرح می کند. وقتی که ORB متوجه می‌شود object reference موجود، مخصوص یک شی دوردست است، مسیرخود را عوض کرده و از طریق شبکه (پروتکل IIOP) به‌دنبال ORB شی دوردست می‌گردد و درخواست خود را



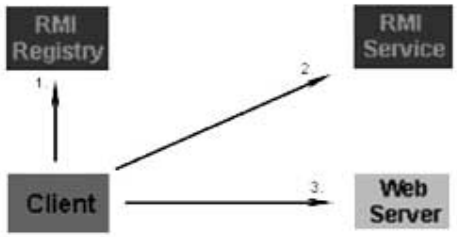
شکل 3- درخواست از کلاینت به object implementationمنتقل می‌شود.

مطـــــــ

نرم‌افزار ایجاد شده‌اند، اما RMI بر خلاف بسیاری از آن‌ها، محدود به انواع داده‌ای اولیه یا ساختارهایی متشکل از داده‌های ساده نیست و به‌کمک آن می‌توان اشیای نرم‌افزاری را به‌تمامی همانند یک پارامتر عبور داد یا بازگرداند.

این ویژگی از RMI یک مکانیسم منحصربه‌فرد می‌سازد. چنین خاصیتی به این معنی است که یک برنامه‌نویس جاوا می‌تواند به‌کمک RMI، کدهای جدیدی را در شبکه انتقال دهد و در ماشین‌های مجازی راه‌دور به‌صورت دینامیک آن‌ها را اجرا نماید. بدین ترتیب برنامه‌نویسان جاوا در زمان برنامه‌نویسی سیستم‌های گسترده، آزادی عمل زیادی به‌دست خواهند آورد. در یک محیط گسترده، کلاینت‌های RMI می‌توانند به نسخه‌های جدید سرویس‌های جاوا دسترسی داشته باشند و نیازی به توزیع‌کردن برنامه به کلاینت‌ها نخواهد بود. این ویژگی همانطور که در محیط‌های شبکه محلی اجرا می‌شود، در محیط وب نیز قابل اجرا است.

در مکانیسم RMI، از ویژگی رجیستری هم پشتیبانی می‌شود. کلاینت‌های RMI با مراجعه به رجیستری RMI از وجود سرویس‌های جدید آگاه می‌شوند. در شکل 4، ارتباط بین بخش‌های مختلف یک سیستم RMI نمایش داده شده است. کلاینت‌هایی که از سرویس مشخصی استفاده می‌کنند، می‌توانند با مراجعه به رجیستری RMI، به آخرین نسخه سرویس دسترسی یابند. در صورت نیاز به کلاس جدید، می‌توان آن‌را از طریق وب‌سرور بارگذاری کرد.



شکل 4- کلاینت RMI با اتصال به رجیستری و دسترسی به سرویس RMIمی تواند کدجدید را از وب سرور دریافت کند.

RMI دارای ویژگی‌های دیگری علاوه بر آنچه در بالا به آن اشاره شد، نیز هست. پردازش راه‌دور و تقسیم بار پردازنده‌ها نمونه‌هایی از قابلیت‌های ساده RMI تلقی می‌شوند. RMI به جهت قابلیت انعطاف و سازگاری زیاد، از طرف برنامه‌نویسان به سرعت به عنوان یک ابزار مهم برای توسعه سیستم‌های گسترده، پذیرفته شد. اما زمانی‌که برنامه‌هایی به‌زبان‌های مانند C/C++ یا زبان‌های دیگری (غیر جاوا) بخواهند با RMI ارتباط بیابند، مشکلاتی ایجاد خواهد شد. در واقع RMI تکنولوژی ویژه جاوا است و نمی‌توان از آن به‌راحتی برای ارتباط دادن برنامه‌های جاوا با نرم‌افزارهایی که به‌زبان‌هایی غیر جاوا توسعه یافته‌اند، استفاده کرد. به همین

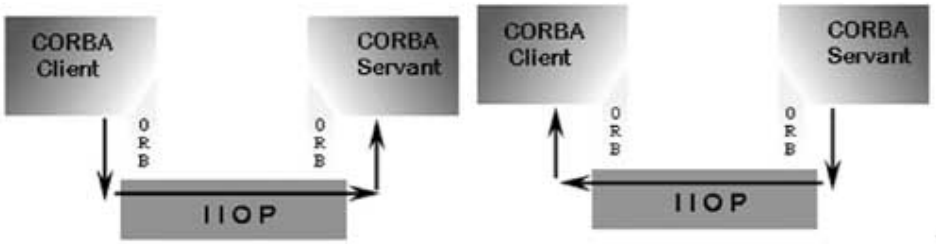
دلیل، استفاده از RMI به همان سرعتی که رواج یافت، روبه کاهش گذاشت. در واقع چنین استدلال می‌شد که در دنیای امروز، روزبه‌روز بر مواردی که نیاز به ارتباط دادن نرم‌افزارهایی که با تکنولوژی‌هایی متفاوت توسعه یافته‌اند، بیشتر می‌شود که در صورت استفاده از RMI، امکان توسعه آتی نرم‌افزار محدود خواهد بود.

و اما CORBA

CORBA تکنولوژی دیگر یا رقیبی است که برای تأمین همان اهداف RMI توسعه داده شده است. تکنولوژی CORBA هرچند که مانند RMI وابسته به یک سکوی نرم‌افزار معین نیست، اما بر خلاف RMI محدود به یک زبان برنامه‌نویسی مشخص نیست و به‌کمک آن می‌توان سیستم‌های نرم‌افزاری گوناگون را در محیط‌های گسترده به یکدیگر ارتباط داد.

اما از نگاه برنامه‌نویسان جاوا (یا کسانی که فقط به فکر پروژه‌هایی هستند که در آن‌ها تنها از زبان جاوا استفاده خواهد شد)، تکنولوژی CORBA قابلیت انعطاف کمتری از خود به نمایش می‌گذارد و دشواری‌هایی را به برنامه‌نویس تحمیل می‌کند. علت این ماجرا هم در آن نهفته است که در CORBA امکان پاس کردن کد اجرایی همانند آنچه در RMI امکان‌پذیر بود، فراهم نیست.

سرویس‌های CORBA از طریق یک رابط یا Interface که به زبان IDL یا Interface Definition Language نوشته شده است، توصیف می‌شوند. نگاشت‌های IDL برای بسیاری از زبان‌های برنامه‌نویسی تهیه شده‌اند و در آینده نیز در هر لحظه می‌توان نگاشت‌هایی برای زبان برنامه‌نویسی دیگری که قصد پشتیبانی از این تکنولوژی را دارند، فراهم کرد.CORBA به اشیای نرم‌افزاری این امکان را می‌دهد که درخواست‌هایی به مقصد اشیای نرم‌افزاری راه‌دور ارسال کنند و بدین ترتیب متدهای شی نرم‌افزاری راه‌دور را فراخوانی نمایند. به همین ترتیب هم می‌توان به‌کمک CORBA، بین سیستم راه‌دور به تبادل داده اقدام کرد. اما این قابلیت در CORBA فقط به رد و بدل کردن انواع داده‌ای ساده یا حداکثر ساختارهای داده‌ای تشکیل شده از انواع داده‌ای ساده، محدود است.



شکل 5- سمت راست کلاینت **CORBA** از طریق **ORB** محلی درخواست خود را به **ORB** سرویس گیرنده ارسال می کند. سمت چپ، سرویس گیرنده **CORBA** پاسخی را **ORB** راه دور ارسال می کند.

در ساختار ارتباطی بین کلاینت‌های CORBA و سرویس‌های CORBA، درخواست‌های فراخوانی متدها به اشیایی به‌نام ORB، رد می‌شوند (پاس داده می‌شوند). اشیای ORB هم از طریق پروتکل ارتباطی Internet Inter ORB یا IIOP با یکدیگر ارتباط می‌یابند. از طرف دیگر، تراکنش‌های IIOP قابلیت جاری شدن بر بستر پروتکلTCP یا پروتکل‌های دیگری نظیر HTTP در مواقعی که یکی از طرفین ارتباط پشت یک دیواره‌آتش واقع شده‌اند، را دارند. اشکال نمایش داده شده در شکل دو، چگونگی ارتباط بین کلاینت و سرور CORBA به تصویر در آمده است.

CORBA یا RMI همان‌طور که بیان شد، هر کدام از این دو تکنولوژی‌ها دارای ویژگی‌ها و مزایای خاص خود هستند و نمی‌توان هیچ یک را طور مطلق جایگزین دیگری کرد. خواص هر یک از دو تکنولوژی موجب شده است که هر یک از آنها در حوزه فعالیت مشخصی به‌کار روند. خلاصه آن‌که اگر از مقایسه این دو تکنولوژی در اینجا نام برده می‌شود، علت آن است که از این طریق بهتر می‌توان قابلیت‌ها و نقاط قوت یا ضعف آن‌ها را شناسایی کرد.

نقاط ضعف	نقاط قوت

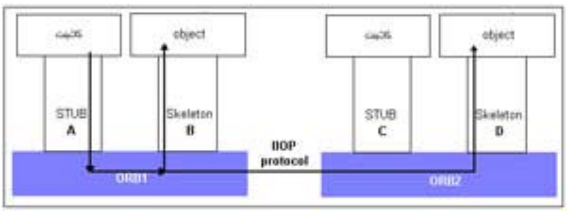
سرویس‌ها را می‌توان به زبان‌های متفاوت و تحت سکوهای گوناگون نوشت و از طریق هر زبان دلخواهی به‌کمک نگاشت IDLمربوطه، به آن دسترسی یافت .	توصیف سرویس‌ها، نیازمند استفاده از زبان توصیفی اینترفیس یا IDL دارد. زبانی که باید آموخته شود. استفاده از سرویس‌ها نیازمند به موجود بودن نگاشت IDL به زبان برنامه‌نویسی مربوطه است. در واقع انتظار برای عرضه نگاشتی برای زبانی که قبلاً نگاشت IDL برای آن فراهم نشده است، معقول نخواهد بود.
با وجودIDL ، کد رابط یا اینترفیس به‌طور کامل از کد نرم‌افزار جدا نگه‌داشته می‌شود و به این دلیل برنامه‌نویسان قادر خواهند بود، پیاده‌سازی‌های متفاوتی برای کار با یک اینترفیس طراحی کنند.	ابزارهای تولید نگاشت IDL برای زبان‌های گوناگون، زیربنای کد اینترفیس‌ها را تولید می‌کنند. هرچند که برخی از چنین ابزارهایی احتمالاً تغییرات جدید را در کد پیاده‌سازی شده لحاظ نمی‌کنند.
تکنولوژی CORBA برای کار با سیستم‌های نرم‌افزاری قدیمی‌تر ایده‌آل بوده وامکان توسعه آتی هر سیستم نرم‌افزاری را فراهم و تضمین می‌کنند.	از انتقال و تبادل اشیای نرم‌افزاری یا کدهای اجرایی، پشتیبانی نمی‌کند.
Corbaروش ساده‌ای برای برقراری ارتباط بین اشیای نرم افزاری و سیستم‌ها، ارائه می دهد.	آینده این تکنولوژی به‌نوعی در ابهام قرار دارد. در صورت عدم رواج و همه‌گیری CORBAدر صنعت، این تکنولوژی به جمع تکنولوژی‌های قدیمی خواهد پیوست CORBA .روش ساده‌ای برای برقراری ارتباط بین اشیای نرم‌افزاری و سیستم‌ها، ارائه می‌دهد.
کارایی و سرعت سیستم‌های مبتنی بر CORBA می تواند در حدقابل توجهی بالا باشد .	تمام کلاس‌های نرم‌افزار نیاز به کارایی RealTimeندارند و می‌توان از سرعت و کارایی در برابر سهولت کاربرد در سیستم‌های تماماً جاوا و خالص، چشم‌پوشی کرد. کارایی و سرعت سیستم‌های مبتنی بر CORBAمی‌تواند در حد قابل توجهی بالا باشد.

تکنولوژی RMI قابلیت‌ها و برتری‌های قابل‌توجهی نسبت به CORBA دارد که در این نوشته از مهم‌ترین آن‌ها یاد کردیم. این تکنولوژی از زمان معرفی JDK 1.20 به این طرف در دسترس همگان قرار داده شده است. به همین دلیل هم بسیاری از برنامه‌نویسان جاوا با این تکنولوژی آشنایی دارند و از تجربه استفاده از آن برخوردارند و باز به همین دلیل نیز در بسیاری از شرکت‌ها و سازمان‌ها، از همان زمان‌ها سیستم‌هایی به‌کار گرفته شده‌اند که بر اساس RMI عمل می‌کنند. این موارد باعث شده‌اند که با وجود آنکه تکنولوژی CORBA در بسیاری از موارد کارها را برای برنامه‌نویسان سهل کرده است، اما هنوز بسیاری از برنامه‌نویسان به کار با RMI ادامه می‌دهند و صنعت نرم‌افزار نیز به چنین برنامه‌نویسانی نیاز دارد.

از طرف دیگر تکنولوژی CORBA به‌دلیل سهولت کاربرد و قابلیت انعطاف‌پذیری و از همه مهم‌تر، پرتابل بودن آن (امکان استفاده در زبان‌های گوناگون)، طرفداران زیادی در بین برنامه‌نویسان یافته است. اما صرف نظر از این موضوع، استفاده از این تکنولوژی در سازمان‌هایی که سیستم‌های نرم‌افزاری گسترده‌ای دارند و امکان جایگزینی تمامی بخش‌های آن‌ها با اجزای نرم‌افزاری جدید میسر نیست، امری غیر قابل اجتناب یا غیر قابل جایگزین تلقی می‌شود. تکنولوژی CORBA به‌راحتی می‌تواند بین بخش‌های نرم‌افزاری متفاوت و سکوی‌های گوناگون ارتباط برقرار کرده و همانند پل ارتباطی به‌کار گرفته شود. در واقع تنها محدودیت CORBA در این زمینه آن است که برای آن‌که بتوان از این تکنولوژی در زبان‌های برنامه‌نویسی متفاوت استفاده کرد، باید قبلاً پشتیبانی از CORBA در آن زبان فراهم شده باشد.

نکته مهم دیگر، وضعیت سرعت و کارایی این دو تکنولوژی در مقایسه با یکدیگر است. مشخص شده است که CORBA از نظر سرعت اجرا تا حدودی بر RMI برتری دارد و به همین دلیل جذابیت بیشتری در بین برنامه‌نویسانی که در حوزه برنامه‌نویسی realtime کار می‌کنند، یافته است.

مزایا و معایب CORBA



شکل 4- ارتباط ORB به ORB مکانیزم درخواست سرویس از راه دور

با توجه به آنچه گذشت، اکنون‌به‌نظر می‌رسد بحث درباره مزایا و معایب این استاندارد، ضروری باشد. از این‌رو به پاره‌ای از مزایا و معایب CORBA اشاره می‌شود:

- کلاینت نیازی به دانستن موقعیت مکانی شی ندارد. یک شی می‌تواند هر جا باشد، به کلاینت متصل باشد یا برروی سروری در آن سوی کره زمین، هیچ فرقی نمی‌کند.

- موقعیت مکانی شی می‌تواند بدون بروز اشکال در نرم افزار تغییر یابد.

- کلاینت نیازی به دانستن این‌که سروری برای جواب دادن به درخواست وجود دارد یا نه، ندارد.

- کلاینت و سرور می‌توانند به دو زبان کاملا متفاوت نوشته شوند و این مزیت مهم CORBA اجازه استفاده ازقدرت زبان‌های مختلف را ازطریق (IDL) برای تهیه نرم‌افزارهای بزرگ می‌دهد.

- کلاینت نمی‌داند یک شی چگونه عمل می‌کند. به همین خاطر یک سرور می‌تواند بدون این که کلاینت اطلاع پیدا کند یک بار از فایل ساده و بار دیگر از پایگاه اطلاعاتی شی‌گرا برای ذخیره اطلاعات استفاده نماید.

- سرعت و کارایی سیستم‌هایی که از CORBA استفاده می کنند بسیار بالا است.

- سیستم‌های عامل و پروتکل سرور و کلاینت می تواند متفاوت باشد.

اگرچه همان‌طورکه می‌بینید استفاده از CORBA می‌تواند مزایای زیادی داشته باشد، اما دارای نقاط ضعف زیر نیز هست:

- از انتقال و جابه‌جایی شی (objectها) پشتیبانی نمی کند.

- CORBA به IDL هایی نیاز دارد که هنوز برای برخی از زبان‌ها تعریف نشده است و وقت زیادی برای یادگیری آن‌ها نیاز است.
- اگر CORBA نتواند نیازهای صنایع امروز را برآورده سازد و رواج نیابد، آینده مبهمی در انتظار خواهدداشت و ممکن است به جمع سیستم‌های قدیمی بپیوندد.
- از آنجایی که خصوصیات CORBA هر چند وقت یک‌بار عوض می‌شود، نیاز به آموزش و بروزآوری بیشتری دارد.

- از آن جایی که شعار CORBA، سرعت و کارایی بیشتر نسبت به سیستم‌های توزیع‌یافته جاوا مثل RMI است و همه نرم‌افزارهای کاربردی نیازی به سرعت بالای آن ندارند، شاید بتوان گفت اگر بخواهیم فقط با جاوا کار کنیم آسانی و کاربرد سیستم‌هایی مثل RMI از CORBA بیشتر است.

گردآورنده : حمید یگانه