جلسه دوم

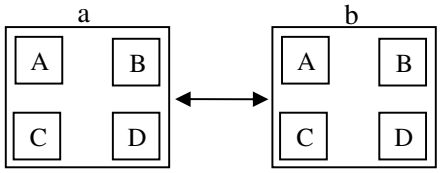
باز بودن(openess)

اگر بخواهیم نیاز های جدید یا سرویس های جدید ارائه شود middelware باید بتواند سرویس های جدید را ارائه دهد لازمه این کار این است که از یک زبان تعریف مشترک (IDL) مثل جاوا برای middelwar استفاده شود. به عبارت دیگر یک برنامه کاربردی که نوشته می شود برایش مهم نباشد که بر روی linux اجرا می شود یا بر روی windows . جهت داشتن oppeness می بایست تمام تعریف ها کامل و فارغ از پیاده سازی باشد. کامل بودن خیلی مشکل است زیرا طراح یک سری مفروضاتی را داشته است و کسی که بخواهد سیستم را توسعه دهد ممکن است یک سری مفروضات دیگری نیاز داشته باشد.

برای کامل بودن و فارغ بودن از پیاده سازی(داشتن oppeness) بایستی ویژگی هی زیر را داشته باشیم

portable : بتواند بر روی هر ماشینی قابل حمل باشد.

Flexible : یک سیستم توزیع شده باز بایستی قابلیت انعطاف داشته باشد، طوریکه یک سیستم را بتوان با component های مختلف از توسعه دهنده های مختلف پیکر بندی کرد. به طوریکه بتوان component جدیدی را اضافه کرد بدون آن که component جدید تاثیرات جانبی داشته باشد. مثلا به جای component، B از a ، B از b را قرار دهیم.



(قابلیت توسعه)Scalability یک سیستم در سه بعد زیرمی تواند مورد ارزیابی قرار گیرد.

- **اندازه(size):** یعنی این که ما به راحتی بتوانیم کاربران یا منابع زیادی را به سیستیم اضافه کنیم
- **جغرافیا(geographica):** ماکسیمم فاصله بین منابع و کاربران
- **مدیریتی(Administrative):** یعنی این که سیستم با داشتن تعداد زیاد سازمان های اجرائی مستقل باز هم به راحتی قابل مدیریت باشد

نکته قابل توجه این است که اگر سیستمی بخواهد در یکی با چند تا ابعاد گفته شده قابل توسعه باشد اغلب تا حدی از کارائی سیستم کاسته می شود.

زمانی که یک سیستم نیاز به توسعه دارد مسائل بسیار زیادی وجود دارد که بایستی حل شوند به عنوان مثال در توسعه اندازه(size) اگر کاربران یا منابع زیادی را سرویس دهیم با محدودیت های زیر مواجه خواهیم شد.

محدودیت

- سرویس های متمرکز(centralized services)
- داده متمرکز(centralized Data)
- الگوریتم های متمرکز(centralized algorithms)

از جهت Data:

اگر Data به صورت متمرکز باشد مانند دفترچه تلفن، گسترش آن مشکل می باشد به عنوان مثال برای سازمان بزرگی مانند ثبت احوال، قرار دادن تمام اطلاعات در یک جدول سبب ایجاد مشکل در توسعه می شود.

از جهت service:

قرار دادن تمام سرویس ها بر روی یک ماشین(متمرکز) باز توسعه آنرا مشکل می کند پس لزومی ندارد data و service مثل هم باشند بلکه یکی می تواند distributed و دیگری centralize باشد.

از جهت algorithm:

الگوریتم بهینه برای مسیر یابی، تصمیم گرفتن بر اساس اطلاعات تمام ماشین ها می باشد(دید سراسری). بدین معنا که عمل Routing را یک ماشین انجام می دهد و اطلاعات سراسری را از طریق تمام ماشین هاتحویل می گیرد ولی این عمل سبب ایجاد مشکلاتی از قبیل ازدحام ترافیک می شوند. پس روش بهتر آن است که از الگوریتم های غیر متمرکز که خصوصیان زیر را دارند استفاده کنیم.

- هیچ ماشینی اطلاعات کامل از کل سیستم را ندارد.
- ماشین ها بر مبنای اطلاعات محلی خود تصمیم می گیرند
- از کار افتادن یک ماشین سیستم را از کار نمی اندازد.
- هیچ فرضی در مورد این که یک ساعت سراسری وجود دارد نیست.

اهمیت مورد آخر به این دلیل است که همگام کردن ساعت ها به ویژه در شبکه های بزرگ ملی امکان پذیر نیست.

محدودیت جغرافیا(geographica) :

از جهت جغرافیا مشکل تاخیر داریم، تاخیر در شبکه های محلی حدود چند صد میکرو ثانیه ولی در یک شبکه گسترده تاخیر در حدود چند صد میلی ثانیه می باشد. در شبکه های گسترده ارتباطات به صورت ذاتی غیر قابل اعتماد می باشد.

پس اگر مولفه های ما centralized باشند سبب به هدر رفتن منابع سیستم می شوند.

مدیریت شبکه:

اگر سیستم جدیدی به سیستم توزیع شده اضافه شود، چون هر سیستم سیاست های domain خودش را دارد، تداخل پیش می آید، برای جلوگیری از این کار می بایست domain سیستمی را که اضافه می شود سیاستی را اعمال کند که دیگر سیستم ها به اطلاعات آن به صورت read only و قابل اجرا دسترسی داشته باشند.

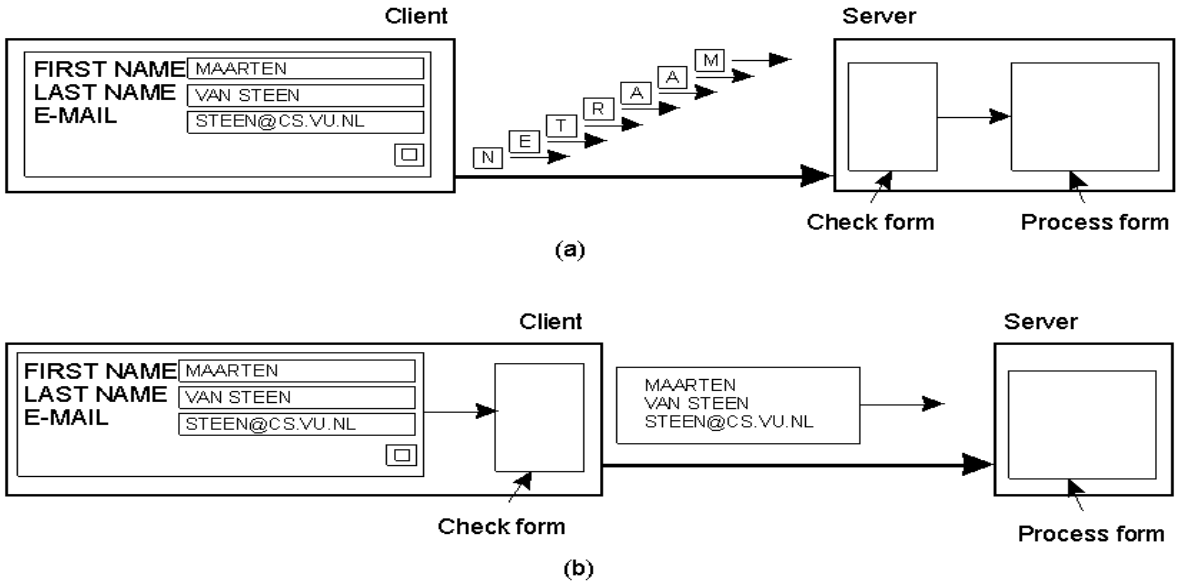
مشکل توزیع شده از لحاظ domain ← امنیت را به خطر می اندازد.

تکنیک های توسعه دادن(scaling techniques)

- **مخفی کردن تاخیر ارتباطات(hiding communication latenices)**
- **توزیع(distributed)**
- **تکرار(Replication)**

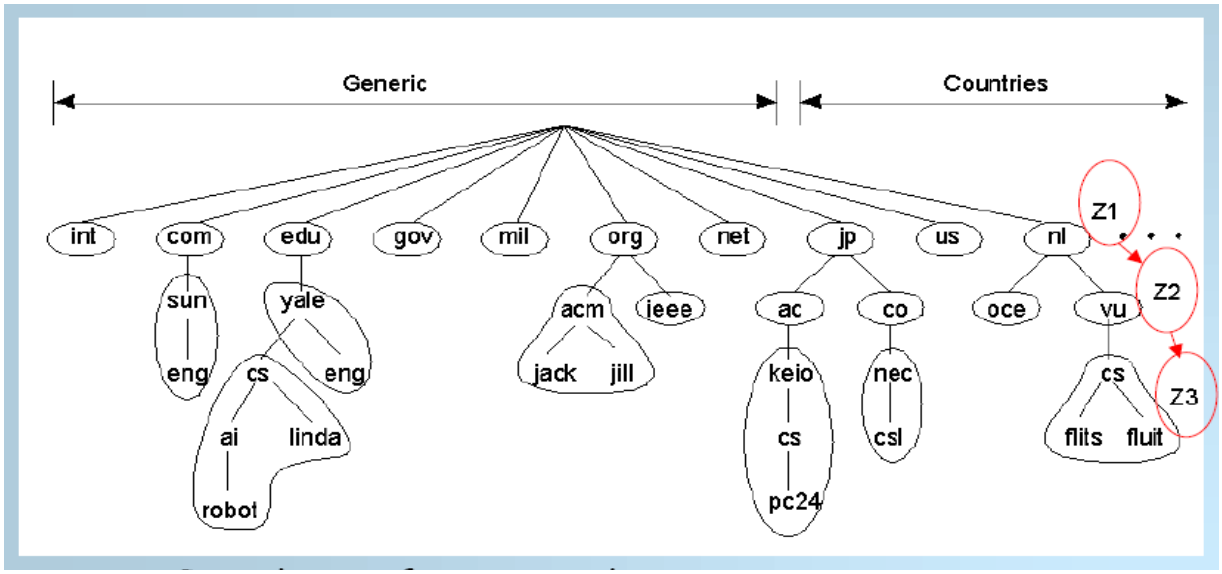
مخفی کردن تاخیر ارتباطات برای رسیدن به قابلیت توسعه جغرافیائی مناسب است، در واقع ایده اصلی این است که تا حد امکان از زمان زیاد پاسخ دهی سرورهای دوردست جلوگیری کنیم. چون در این روش نمی تون برنامه نویسی سنکرون داشت می بایست ارتباطات را به صورت آسنکرون پیاده سازی کرد که این روش برای کار های دسته ای یا batch مفید می باشد. به عنوان مثال وقتی پردازه p عمل send را انجام داد به جای این که منتظر دریافت نتیجه بماند، اعمال دیگری را انجام دهد و این زمانی قابل اجراست که اعمال پردازه p مستقل از هم باشند و این را می بایست هنگام انتساب وظایف در نظر گرفت این روش برای اعمال محاوره ای مناسب نیست زیرا کاربر منتظر پاسخ است.مثال دیگر در صفحه بعد

در قسمت a شکل زیر وقتی client درخواستی را از server دارد با توجه به این کـه عمـل Chek form در server انجام می گیرد و با داشتن client های زیاد و درخواست های زیاد client ها بایستی یک مدت زمان نسبتا زیـادی را بـرای بدست آوردن پاسخ منتظر بمانند در صـورتی کـه در شـکل b بـا گذاشـتن عمـل Chek form بـه عهـده خـود clint هـا بار Server زیاد نمی شود و زمان پاسخ دهی نیز کم می شود.



توزیع(distributed): توزیع در واقع تبدِل یک مولفه یا Component به اجزاء کوچکتر و قرار دادن این قسمت ها در کنار هم می باشد. مثالی از DNS، Distribution می باشد که یک ساختار سلسله مراتبی دارد. با توجه به شکل زیـر ملاحظـه مـی شود که DNS رابه سه ناحیه تقسیم کردیم، DNS بایستی به آدرس IP تبدیل بشود اگر از روش متمرکز اسـتفاده مـی کـردیم ترافیک ایجاد می شد با قرار دادن هر Zon بر روی یک Server از شدت ترافیک کاسته می شود

Internet Domain Name system(DNS)



فرض کنید nl.vu.cs.flits تقاضا شود این آدرس ابتدا بر روی Z1 ارسال می شود، server ی که مربوط به Z1 است آدرس Server ، Z2 را بر می گرداند و Z2 آدرس Z3 را بر می گرداند که در نهایت Z3 آدرس IP معادل را بر می گرداند. این کار از تک سروری جلوگیری می کند.

مثال دیگر. www(word wide web) می باشد که برای اکثر کاربران هر سندی نام یکتای خودش را به فرم URL دارد که این ممکن است در ظاهر یک سرور به نظر برسد ولی web به صورت فیزیکی بین چندین server توزیع شده است که هر کدام یک document را اداره می کنند. به راحتی می توانیم سندی را اضافه کنیم به همین خاطر قابلیت گسترش در web بالاست.

مثال دیگر: اپلت های جاوا(Java applets) می باشد که محاسبات و کار ها را بر روی client ها انتقال می دهد(توزیع کارها)