XMulator

شبیه ساز شیئ گرای مبتنی بر شنونده چند لایه

عباس نایبی
دانشکده مهندسی
دانشگاه صنعتی شریف

زمستان ۸۴

۱. مقدمه
۱-۱-انگیزه
۱-۲-مبانی شبیه سازی
۱-۳-معرفی مفاهیم اولیه
۱-۴-نکاتی در مورد مهندسی نرم افزار
۲. معماری XMulator
۲-۱-مجتمع سازی مبتنی بر شنونده
۲-۲-معرفی لایه ه
۲-۳-بررسی یک مثال
۲-۴-گزارش گیری
۳. کلاسهای پایه
۳-۱-XObject
۳-۲-XEvent
۳-۳-XEngine
۳-۴-Buffer
۳-۵-MetaComponent
۳-۶-End Port & Middle Port
۳-۷-BaseXEventGenerator
۳-۸-TimeIntervalXEventGenerator
۳-۹-BaseNode
۴. بسته Interconnection Network
۴-۱-InterConnectionBuffer
۴-۲-InputVCBuffer & OutputVCBuffer
۴-۳-BaseMessage
۴-۴-PhysicalChannelSender & PhysicalChannelReceiver
۴-۵-Switch
۴-۶-Node
۴-۷-VCArbiter
۴-۸-InjectionChannel
۴-۹-PE
۴-۱۰- Flit, HeaderFlit, HeaderFlitSingle
۴-۱۱- SwitchingInfo
۴-۱۲- MessageConsumer
۴-۱۳- PhysicalEjectionChannel و PhysicalInjectionChannel
۴-۱۴- RE
۴-۱۵- Routing Function
۴-۱۶- SimpleMessageGenerator
۴-۱۷- بسته NetworkGeneration
۵. بسته Queuing Network

۵-۱-معرفی کلاسها

۵-۲-مثال ساده

۶. بسته Meta K_ary N_cube

۶-۱-ساختن یک شبکه میان ارتباطی

۶-۲-الگوریتم مسیریابی E_cube

۶-۳-الگوریتم مسیریابی Duato

۷. جمع آوری اطلاعات حاصل شده از شبیه سازی

۸. نمونه های اجرایی

۸-۱-صف دو مرحله ای با سررسید[1]

۸-۲-شبکه فوق مکعب و Kary_Ncube

۸-۳-شبکه Kary_Ncube با اشکال

۸-۴-شبیه سازی مدارهای منطقی و شبیه ساز اشکال

۸-۵-شبکه فوق مکعب با گردنبند

[1] Deadline

۱. مقدمه

۱-۱- انگیزه

۱-۲- مبانی شبیه سازی

به طور کلی سه روش پایه ای برای شبیه سازی وجود دارد :

روش کوانتوم زمانی

در این روش زمان را به فاصله های زمانی کوچک[2] تقسیم می کنیم و برنامه را اجرا می کنیم. مشکل این روش این است که اگر فاصله های زمانی را خیلی کوچک در نظر بگیریم ، دقت زیاد ولی زمان شبیه سازی نیز بسیار زیاد می شود و اگر فاصله های زمانی را بزرگ در نظر بگیریم ، دقت کم و در مواقعی غیر قابل قبول می شود.

مشکل دیگر این روش ، پیوستگی زمان در شبیه سازی می باشد. بنابراین همیشه باید زمان واقعی را به یک زمان تقریبی گرد کنیم.در واقع دقت زمانی را از دست می دهیم.در ضمن وقتی می توانیم از این روش استفاده کنیم که سیستم با یک ساعت همزمان شده باشد و تمام فعالیت ها در آغاز فاصله های زمانی انجام شود.

روش مبتنی بر فرایند[3]:

در این روش هر جزء با یک فرایند (یا شبه فرایند) مدل می‌شود. ویژگی این روش سادگی تعریف رفتار اجزاء می‌باشد و عیب آن کاهش کارایی است. در این روش موتور اصلی شبیه ساز بسیار پیچیده می شود و باید از Multithreading پشتیبانی کند.

روش رویداد گرا[4] :

در این روش چون نمی توانیم برای هر مولفه واقعی سیستم یک thread در نظر بگیریم ، مجبوریم با استفاده از ثبت رویداد[5] ها (رخدادهایی که وضعیت سیستم را تغییر می دهند) رفتار سیستم را شبیه سازی کنیم.

مشکلی که در برنامه های شبیه ساز وجود دارد این است که ما یک فرایند[6] یا یک thread داریم و این فرایند باید کل کار شبیه سازی را انجام دهد.یعنی هر زمان که یک رویداد رخ میدهد، فرایند آنرا اجرا می کند و دوباره به اجرای خود بر می گردد.معمولا وقتی یک رویداد پردازش می شود ، یک یا چند رویداد دیگر را برای زمانهای بعدی شبیه سازی تولید می کند.

به طور مثال فرض کنید که می خواهیم رفتار یک تولید کننده پیام[7] را شبیه سازی کنیم. یک را ه این است که برای آن یک thread در نظر بگیریم که یک پیام را تولید کند و در شبکه قرار دهد، یک ثانیه صبر کند (یک ثانیه لزوما زمان واقعی نیست و یک ثانیه شبیه سازی یا در واقع یک ثانیه از زمان مجازی است.) و پیام بعدی را تولید کند. اما اینکار عملی نیست ، یعنی نمی توانیم برای تولید کننده پیام یک thread در نظر بگیریم.از اینرو ما اینطور در نظر می گیریم که در

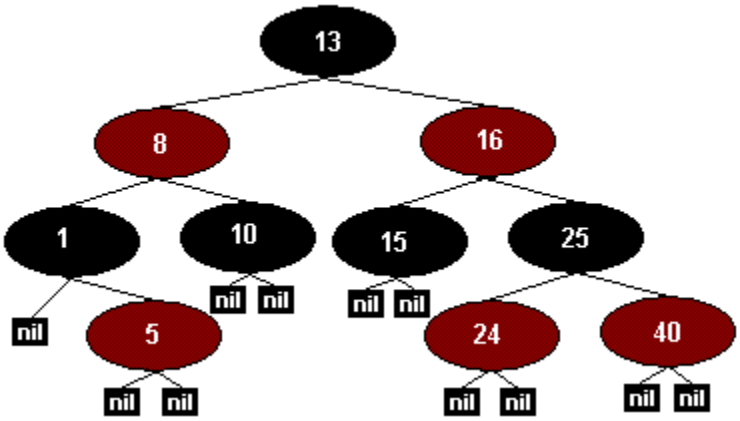
---

[2] Quantum
[3] Process based
[4] Event based
[5] Event
[6] Process
[7] Message Generator

زمان صفر یک رویداد باید رخ دهد و این رویداد بیانگر تولید یک پیام در ثانیه اول می باشد. همزمان با پردازش این رویداد یک رویداد جدید نیز در ثانیه اول باید تولید شود که مبین تولید پیام بعدی در ابتدای ثانیه دوم می باشد.
بنابراین در روش رویدادگرا وجود رویداد در یک زمان خاص برای هر مولفه به این معنی است که شبیه ساز باید در آن زمان خاص ، آن مولفه را صدا کند و مولفه کار خود را آغاز کند.

۱-۳- معرفی مفاهیم اولیه

در حالت کلی می توانیم این طور در نظر بگیریم که ما تعدادی مولفه و رخداد داریم و یک موتور شبیه ساز[8] که کار اصلی شبیه سازی را انجام می دهد. این موتور یک صف از رویداد ها را باید در خود نگه دارد.
کار موتور بسیار ساده است. ساختار داخلی آن در واقع یک حلقه تکرار while می باشد.به این ترتیب که رویدادی که سر صف وجود دارد ، توسط موتور پردازش شده و رویدادهای بعدی به ترتیب پردازش می شوند. چون هر رویداد منبع تولید یک یا چند رویداد بعدی نیز می باشد، بنابراین صف رویدادهای موجود در موتور در شرایط طبیعی هیچگاه تمام نمی شود.
هر چه ساختار موتور ساده تر باشد، برنامه از نظر پیمانه ای بودن[9] قویتر و منعطف تر است. در واقع موتور نباید از نوع رویدادها و نحوه کار آنها باخبر باشد، فقط کافی است بداند که هر رویداد مربوط به کدام مولفه می باشد.
صف رویدادها بر اساس زمان وقوع آنها باید مرتب شده باشد. طریقه نگهداری این صف در کارایی شبیه ساز بسیار موثر است. آرایه ها و لیستهای پیوندی[10] کارایی کمی دارند، بنابراین در این شبیه ساز از درخت Red Black[11] که نوع خاصی از درخت AVL می باشد، استفاده می شود. رویدادها در این درخت ذخیره می شوند و هر عضو درخت خود شامل صفی از رویدادهاست که زمان رخدادن همگی آنها دقیقا برابر است. مثلا رویدادهای که در زمان ۵۰ باید اجرا شوند، در دنیای واقعی بصورت موازی انجام می گیرند، بنابراین ترتیب اجرا شدن آنها در برنامه شبیه ساز نباید تاثیری بر نتیجه اجرا داشته باشد. از این رو رویدادهایی که زمان رخداد آنها یکی است، در یک صف نگهداشته می شوند. در شکل زیر یک نمونه از درخت Red-Black نشان داده شده است.



شکل ۱

[8] Simulation Engine
[9] Modularity
[10] Linked List
[11] http://www.codeproject.com/csharp/redblackcs.asp

قبل از بررسی یک مثال شبیه سازی لازم است به نکاتی در مورد طراحی نرم افزار در مهندسی نرم افزار دقت شود.

۱-۴- نکاتی در مورد مهندسی نرم افزار

در مهندسی نرم افزار برای طراحی یک نرم افزار باید اصول لایه بندی در آن رعایت شود. هر لایه از تعدادی بسته[۱۲] تشکیل شده است که این بسته ها نیز خود از تعدادی کلاس تشکیل شده اند. در زبان #C یک بسته معمولا با namespace مشخص می شود.

ترتیب رویهم قرارگرفتن لایه ها در طراحی نرم افزار بسیار با اهمیت است. زیرا هر لایه می تواند فقط از لایه های زیرین خود مطلع باشد و در موقع لزوم از آنها استفاده کند. یک لایه پایینی هیچگاه نباید از لایه یا لایه های بالایی خود مطلع باشد. همچنین در یک لایه نیز نباید بسته ها با هم ارتباط افقی دو طرفه داشته باشند زیرا به طور مثال اگر دو بسته A و B در یک لایه به نحوی ساخته شوند که A به کلاسهای B و B به کلاسهای A نیاز داشته باشد ، اصول طراحی رعایت نشده و اولین مشکل ممکن، مشکل در ترجمه[۱۳] پروژه خواهد بود.

۲. معماری XMulator

Xmulator یک شبیه ساز لایه ای می باشد که در آن اصول مهندسی نرم افزار به طور کامل رعایت شده است. قبل از بررسی ساختار لایه ای آن بهتر است کمی راجع به تکنیک مجتمع سازی مبتنی بر شنونده توضیح داده شود.

۲-۱- مجتمع سازی مبتنی بر شنونده[۱۴]

در شبیه ساز XMulator دو نوع رویداد وجود دارد. دسته اول رویدادهایی هستند که برای پیشبرد زمان شبیه سازی از آنها استفاده می شود. این دسته در واقع XEvent ها هستند که در بخش بعدی معرفی می شوند.

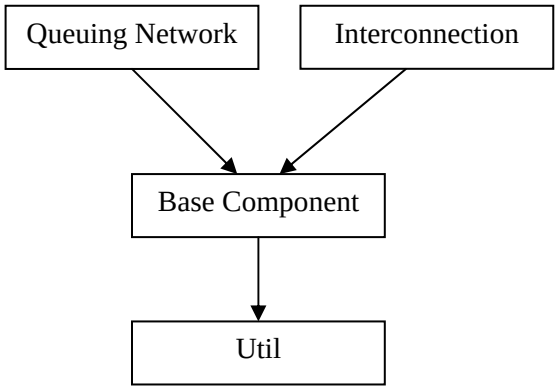
دسته دوم رویدادهایی هستند که برای برقراری ارتباط بین مولفه ها در نظر گرفته شده اند.این دسته همان delegateها در زبان #C می باشند. به طور مثال فرض کنید که یک بافر خالی است و مولفه هایی که به آن متصل شده اند ، مانند سرور ، منتظر قرار گرفتن اولین پیام روی بافر هستند. در این حالت بافر یک رویداد از نوع دوم بنام OnFirstSlotFilled() تولید می کند و مولفه هایی که به بافر متصل هستند، روی این رویداد به اصطلاح گوش می کنند[۱۵]، به محض اینکه یک پیام روی بافر گذاشته شود، متد OnFirstSlotFilled() آن صدا زده شده و این متد نیز سایر مولفه هایی را که منتظر آن هستند، صدا می زند.

۲-۲- معرفی لایه ها

شکل ۲ تعدادی از لایه های موجود در شبیه ساز مورد بحث و همچنین تعدادی از بسته ها را نشان می دهد.

---

[12] Package
[13] Compile
[14] Listener-based integration (LBI)
[15] Listen



شکل ۲

همانطور که در شکل می بینید، اصول لایه بندی کاملا رعایت شده است. به طور مثال موتورکه از بسته Base Component میباشد، نباید از وجود تولید کننده پیام در بسته Queuing Network مطلع باشد. یعنی کد موتورشبیه ساز نباید به‌گونه‌ای نوشته شود که کوچکترین وابستگی به کد تولیدکننده پیام داشته باشد. زیرا در غیر اینصورت اگر روزی بخواهیم به جای تولیدکننده پیام از مولفه دیگری استفاده کنیم، مجبوریم که موتورشبیه ساز را دوباره بازسازی کنیم.

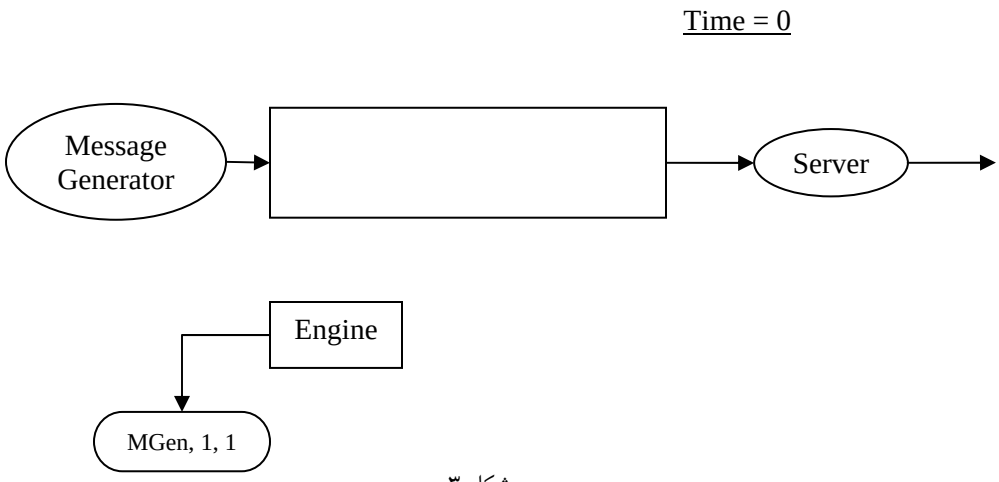
۲-۳- بررسی یک مثال

در XMulator ، کلاسی بنام XObject وجود دارد که پدر همه مولفه های پایه می باشد. این مولفه ها در واقع مدل عناصری هستند که در دنیای واقعی وجود دارند و هر کدام رفتار خاص خود را دارند و همگی رویدادهای خاص خود را تولید می کنند. XObject در لایه Base Component قرار دارد و دو متد مهم بنامهای Start() ، ProcessXEvent() دارد. هر مولفه ای که از XObject ارث می برد و ایجاد می شود، به طور خود کار هنگام فراخوانی متد سازنده، خود را با متد RegisterXObject() در موتور شبیه سازی ثبت می کند. بنابراین موتورشبیه ساز در ابتدای کار لیست تمام مؤلفه ها را دارد و درست قبل از شروع شبیه سازی متد Start() همه این مؤلفه ها را صدا می‌کند. موتور همگی این مؤلفه ها را به چشم یک XObject می‌بیند. بنابراین موتور از نوع مولفه های مختلف که در صف آن ثبت شده است اطلاعی ندارد.

در ادامه برای روشن موضوع به یک مثال ساده توجه کنید.

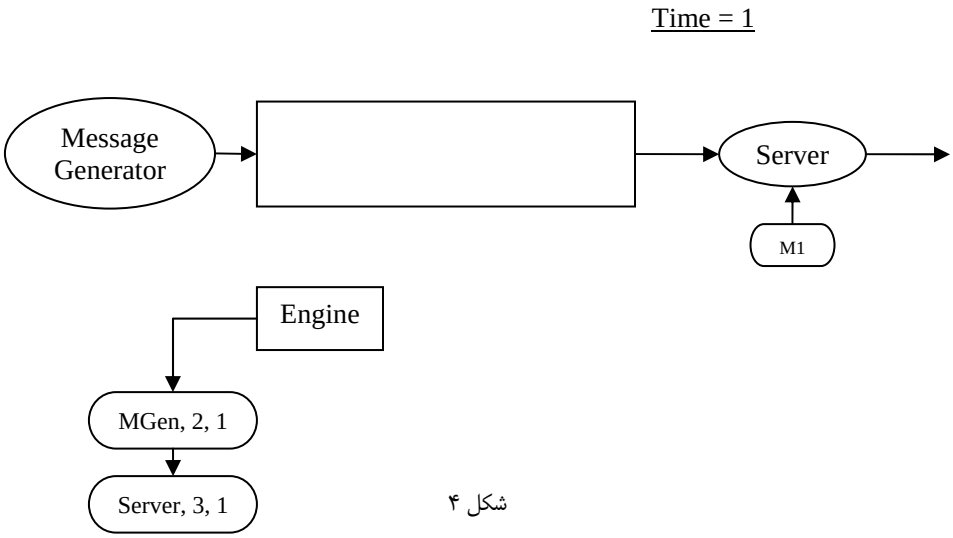
فرض کنید سیستمی از یک تولیدکننده پیام، یک صف و یک سرور و موتور تشکیل شده است. تولیدکننده پیام به طور ثابت در هر ثانیه یک پیام تولید می کند و در صف قرار می دهد. زمان پردازش یک پیام توسط سرور نیز ۲ ثانیه در نظر گرفته شده است.

در ابتدای کار موتور متد start() همه مولفه ها را صدا می زند. متد start() از تولیدکننده پیام یک رویداد را در ثانیه اول در صف موتور ثبت می کند.این رویداد مربوط به تولید یک پیام در این ثانیه می باشد. شکل ۳ وضعیت سیستم را در ثانیه صفر نشان می دهد.



شکل ۳

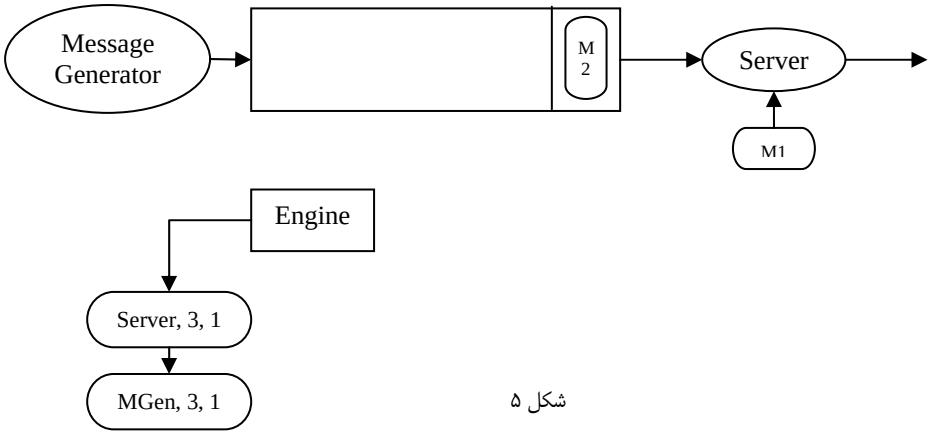
چون در ثانیه صفر کاری برای اجرا نداریم، زمان به سمت ثانیه اول پیش می رود. دراین ثانیه متد دوم تولیدکننده پیام (processXEvent) که از XObject به ارث برده شده، توسط موتور صدا زده می شود. با صدا زده شدن این متد، نوع رویداد توسط آن بررسی می شود و در این مورد یک پیام توسط تولیدکننده پیام تولید شده و در صف قرار می گیرد. در همین ثانیه تولیدکننده پیام یک رویداد جدید برای تولید یک پیام جدید در ثانیه دوم ثبت می کند.
با قرار گرفتن یک پیام در صف، متدی بنام Enqueue() از صف صدا زده می شود.چون صف از قبل خالی بوده است، بنابراین متد start() از سرور را صدا می کند. پیام تولید شده از صف به درون سرور برای اجرا شدن می رود. چون زمان اجرا برای سرور ۲ ثانیه در نظر گرفته شده، یک رویداد در موتور در زمان ۳ ثبت می شود. وضعیت سیستم در ثانیه اول به صورت زیر است.



شکل ۴

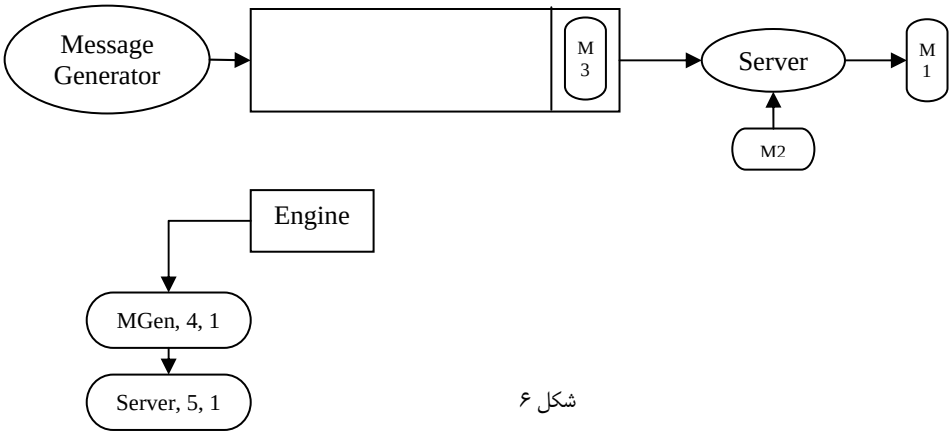
بهمین ترتیب وضعیت سیستم در زمان های ۲و۳و۴ نشان داده شده است.

Time = 2

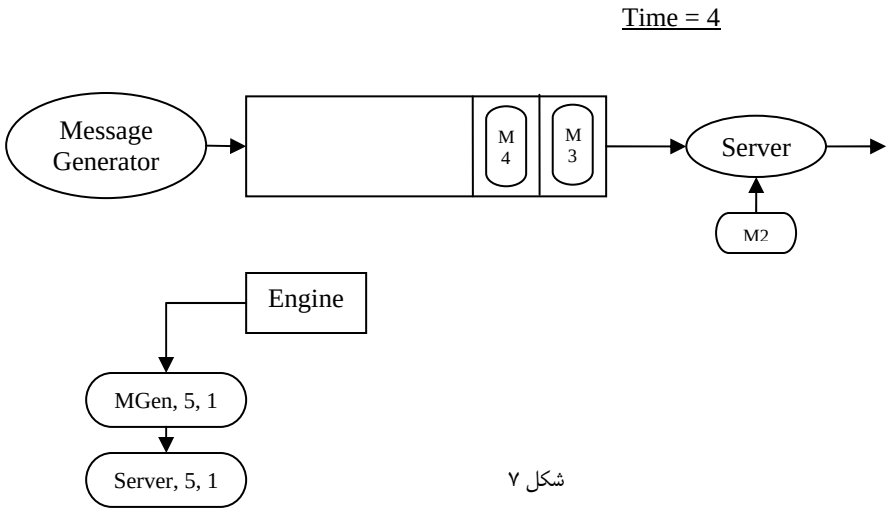


شکل ۵

Time = 3



شکل ۶



شکل ۷

در ثانیه سوم، رویداد مربوط به خاتمه سرویس از سرور (OnSend) صدا زده می شود و پیامی که در سرور بود به بیرون منتقل می شود.سرور برای اجرا کردن پیام بعدی باید به صف سر بزند و اگر عنصری درون صف بود آنرا اجرا کند.همچنین صف نیز باید هر وقت عنصری روی آن قرار گرفت، سرور را که از قبل بیکار بوده مطلع سازد. این وابسته بودن سرور و صف به یکدیگر در مهندسی نرم افزار امروزه جایز نیست. زیرا در اینصورت هنگام نوشتن کد سرور باید از کد صف مطلع باشیم و بالعکس. از طرف دیگر اگر بخواهیم روزی مثلا نسخه جدیدی از صف را بنویسیم ممکن است با سرور ناسازگاری داشته باشد و مجبور شویم سرور را هم باز نویسی کنیم. برای حل این مشکل از تکنیک مجتمع سازی مبتنی بر شنونده که درقسمت قبلی توضیح داده شد، استفاده می کنیم.

۲-۴- گزارش گیری

اکثر کلاسهای موجود در بسته InterConnectionComponent متدی دارند بنام Snapshot. وظیفه این متد ثبت تمامی وقایع مربوط به مولفه ای که از آن کلاس مشتق شده، در زمان شبیه سازی می باشد. این وقایع در طول زمان اجرای برنامه در دو فایل از نوع XML ذخیره می شوند. فایل res.xml شامل تمام وقایع اتفاق افتاده در زمان شبیه سازی می باشد. در مقابل فایل resMB.xml فقط وقایع بلوکی که از داده ها را که پارامتر msgBlockSize مشخص می کند نشان می دهد. پس از اتمام شبیه سازی برای گرفتن خروجی های لازم می توان محتوای این دو فایل را مشاهده کرد. یک روش مشاهده این فایلها استفاده از نرم افزار Microsoft Excel 2003 می باشد. در هنگام باز کردن فایل با این نرم افزار باید گزینه Use the XML Source task pane را انتخاب کرد. حال می توان پارامترهایی که قرار است مشاهده شوند را به درون یک صفحه Excel کشیده و سپس گزینه Refresh XML Data را انتخاب کرد.

۳. کلاسهای پایه

۳-۱- XObject

این کلاس پدر تمامی کلاسهای دیگر می باشد که در زیر به معرفی متدهای آن می پردازیم.

()Start: اگر آماده سازی خاصی داشته باشیم توسط این متد انجام می دهیم.این متد درست قبل از زمان شبیه سازی صدا زده می شود.

()RegisterXEvent :هر وقت بخواهیم یک رویداد را در موتور ثبت کنیم از این متد استفاده می کنیم.

ProcessXEvent(): متدی است که توسط موتور صدا زده می شود. پس از آنکه یک رویداد توسط متد RegisterXEvent() در موتور ثبت شد، در زمان رخدادن آن، موتور متد ProcessXEvent() را صدا زده تا رویداد مربوطه اجرا شود.
ResetDelegates():

SetupDelegates():

ResetState(): با این متد می توانیم بدون اینکه شبکه را مجددا پیکربندی کنیم، به حالت اولیه (بدون هیچ پیام) برگردانیم. این متد باید توسط هر کلاس مشتق شونده از کلاس XObject در صورت لزوم بازنویسی گردد.

۳-۲- XEvent

همانطور که قبلا ذکر شد، XEvent ها دسته ای از رویدادها هستند که برای پیشبرد زمان شبیه سازی از آنها استفاده می شود. این رویدادها دارای صفات[۱۶] زیر می باشند.
Time: زمانی است که رویداد باید در آن زمان اجرا شود.
TargetXObject: مشخص کننده شیئی از نوع XObject است که رویداد باید روی آن اجرا شود.
Parameters: متغییری است که هنگام اجرا شدن رویداد، مطلب خاصی را به شیئ مورد نظر یادآوری می کند.
Type: هر شیئ می تواند چندین نوع رویداد داشته باشد که این انواع با متغییر type که یک عدد صحیح می باشد، نشان داده می شود. نکته قابل توجه این است که نوع رویداد برای شیئ شناخته شده است نه برای موتور.

۳-۳- XEngine

RegisterXEvent () : مهمترین تابع موتور است که برای ثبت رویدادها از آن استفاده می شود.
Now () : زمان مجازی شبیه سازی را در هر لحظه نشان می دهد.
RegisterXObject () : هنگامیکه یک XObject ساخته می شود به طور خودکار متد سازنده آن توسط این متد در موتور ثبت می شود.
StartSimulation () : برای شروع شبیه سازی در یک بازه زمانی یا به تعداد مشخصی از Xeventها از این متد استفاده می شود.

نکته: برای دسترسی به یک نمونه از XEngine که بین همه کلاسها مشترک است از کلاس XEngineFactory استفاده می شود.

۳-۴- Buffer

یکی از مولفه های پایه می باشد که از XObject مشتق شده است. بنابراین علاوه بر متدهای XObject دارای متدهای زیر می باشد.
IsFull(): اگر بافر پر باشد تنظیم می شود.

[16] Attributes

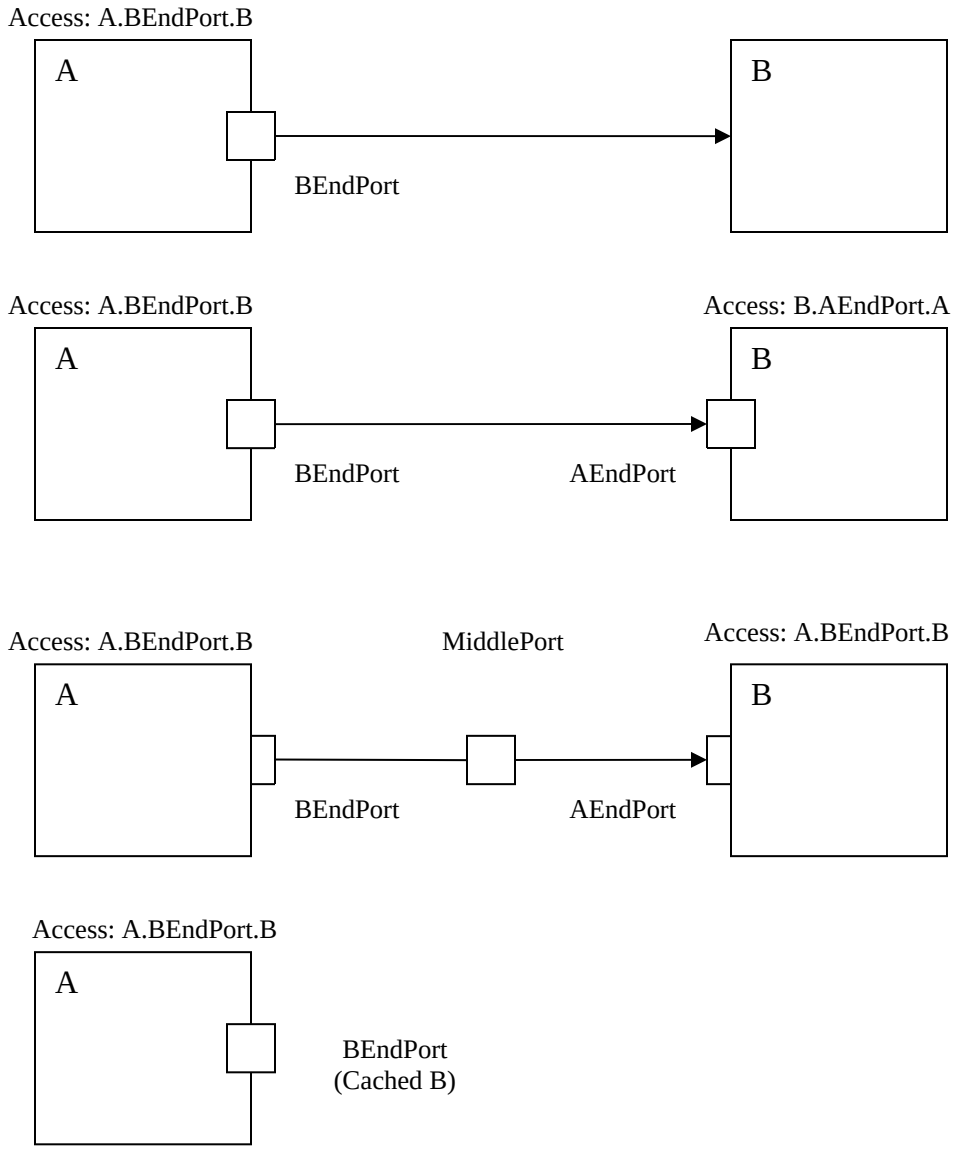
()IsEmpty: اگر بافر خالی باشد تنظیم می شود.
()OnLastSlotFreed: وقتی آخرین خانه بافر خالی می شود یعنی صف از وضعیت پر درآید این متد فراخوانی می شود.
()OnFirstSlotFilled: وقتی اولین خانه بافر پر شود یعنی از وضعیت خالی درآید این متد فراخوانی می شود.
()Enqueue: برای قرار دادن یک شیئ در بافر از این متد استفاده می شود.
()Dequeue: برای برداشتن یک شیئ از بافر از این متد استفاده می شود.
()OnEmpty: وقتی بافر خالی شود این متد فراخوانی می شود.
علاوه بر متدهای فوق بافر دو صفت به شرح زیر دارد:
Count: تعداد اشیاء داخل بافر را نشان می دهد.
MaxSize: بیشترین سایز بافر را نشان می دهد.

۳-۵- Meta Component

گاهی اوقات لازم است که چند شیئ را کنار هم قرار دهیم و یک مولفه جدید بسازیم. مثلا در یک شبکه توری سه بعدی شاید لازم داشته باشیم که مجموعه گره های هر صفحه را به عنوان یک مولفه در نظر بگیریم. این کار توسط کلاس MetaComponent انجام می شود.
این کلاس خود یک XObject است که از تعدادی XObject دیگر تشکیل شده است. هر MetaComponent تعدادی لینک دارد (لینک لزوما به معنای کانال فیزیکی نیست) که از آن برای برقراری ارتباط بین MetaComponentهای مختلف استفاده می شود.
برای مثال برای ساختن یک فوق مکعب ۵بعدی می توان به تر تیب زیر عمل کرد:
هر گره پردازشی خود یک MetaComponent است که از RE,Switch,... تشکیل شده است. دو گره پردازشی را که به هم متصل کنیم یک MetaComponent جدید بنام فوق مکعب یکبعدی شکل می گیرد.بهمین ترتیب دو تا MetaComponent از نوع فوق مکعب یکبعدی را به هم وصل می کنیم تا یک MetaComponent جدید بنام فوق مکعب دو بعدی ساخته شود. اگر همین طور ادامه دهیم MetaComponent آخر که یک فوق مکعب ۵ بعدی است از متصل کردن دو MetaComponent بنام فوق مکعب ۴ بعدی ساخته خواهد شد.

۳-۶- MiddlePort & EndPort

هر XObject تعدادی EndPort دارد. از این پورتها برای برقراری اتصال بین مولفه ها استفاده می شود. هر MetaComponent نیز تعدادی MiddlePort دارد. شکل زیر چهار حالت مختلف اتصال مولفه ها به یکدیگر را نشان می دهد.



شکل ۸

۳-۷- BaseXEventGenerator

کار این مولفه تولید Xevent می باشد. علاوه بر متدهایی که از کلاس پایه Xobject به ارث برده است، دارای یک متد دیگر بنام ()OnXEvent نیز می باشد که هرگاه رویدادی تولید شود این متد فراخوانی می شود.

۳-۸- TimeIntervalXEventGenerator

مولفه ای است که از مولفه BaseXEventGenerator مشتق شده است. کار این مولفه تولید Xevent در فاصله های زمانی مشخص می باشد. این فاصله های زمانی می تواند نمایی، فیکس و یا هر توزیع آماری دلخواهی باشد. متد GenerateNextXEvent() کار تولید رویداد بعدی را به عهده دارد.

۳-۹- BaseNode

کلاسی پایه است که هر گره در شبکه از این کلاس ارث برده خواهد شد.

۴. بسته InterConnectionNetwork

۴-۱- InterConnectionBuffer

مولفه ای است که از کلاس پایه Buffer مشتق شده است.

Switched: خاصیتی است که نشان می دهد آیا بافر در حال رد و بدل کردن اطلاعات هست یا نه.

SwitchingInfo: مشخص می کند که بافر مورد نظر به کدام بافرها وصل شده است. اطلاعات مربوط به سوئیچینگ درون بافرها قرار دارد، زیرا با اینکار سرعت رد و بدل پیام بین بافرها افزایش می یابد.

onTheFlyFlit: اگر همه زمانهای انتقال یک واحد زمانی در نظر گرفته شود، بین ارسال دو flit یک حباب[۱۷] قرار می گیرد. یک کانال فیزیکی را در نظر بگیرید که هم بافر ورودی و هم بافر خروجی آن پر باشد. کانال باید ابتدا منتظر بماند تا بافر خروجی اش خالی شود و سپس داده های بافر ورودی را بپذیرد.این کار باعث می شود بین ارسال دو پیام، یک پیام خالی نیز ارسال شود. برای حل این مشکل هرگاه پیام روی بافر ورودی کانال قرار گیرد، در پالس ساعت بعدی اطلاعات درون onTheFlyFlit قرار می گیرد و بافر ورودی کانال آزاد می شود. به محض خالی شدن بافر خروجی پیام از مکان موقتonTheFlyFlit به بافر خروجی کانال منتقل می شود.

۴-۲- InputVCBuffer & OutputVCBuffer

هردو حالت خاصی از کلاس Buffer هستند. درواقع ورودی و خروجی کانالهای مجازی را از دیدگاه سوئیچ تامین می کنند.

()OnHeadeInFirstSlot هرگاه در اولین خانه بافر flit قرار گرفته از نوع سر[۱۸] باشد، این متد فعال می شود. چون مسیریاب منتظر این متد است بنابراین فعال شده و عملیات مسیریابی را انجام می دهد.

۴-۳- BaseMessage

کلاس پایه ای برای ساختن پیام می باشد. یک خاصیت مهم دارد که از XObject به ارث برده بنام ID. یک پیام به کمک ID در هنگام گزارش گیری و در هنگام رفع خطا[۱۹] شناخته می شود.

GenerationTime: زمان تولید پیام را نشان می دهد.

SingleMessageHeaderFlit: پیامی است که تنها از یک flit آنهم از نوع سر تشکیل شده است.(لازم به ذکر است که سر یک flit در نظر گرفته شده است).

IsTail: هرگاه آخرین flit از یک پیام از سوئیچ رد شود، سوئیچ باید راه را برای عبور پیام بعدی باز کند.خاصیت IsTail نشان دهنده آخرین flit از یک پیام می باشد.

۴-۴- PhysicalChannelSender & PhysicalChannelReceiver

هر کانال فیزیکی به دو قسمت تقسیم می شود.یکی فرستنده و دیگری گیرنده است. بنابراین برای ایجاد یک کانال فیزیکی باید این دو قسمت را بهم وصل کنیم. اتصال این دو قسمت و تشکیل یک کانال فیزیکی توسط دو خاصیت physicalChannelReceiverEndPort و physicalChannelSenderEndPort انجام می شود.

---

[17] Bubble
[18] Header
[19] Debug

ChannelDelay: مدت زمانی است که پیام روی کانال فیزیکی انتقال داده می شود.
() Deliver: توسط این متد یک پیام روی کانال فیزیکی فرستاده می شود.
() OnDeliverd: هرگاه یک پیام به سر دیگر کانال می رسد این متد فعال می شود.
() IsBusy: نشان دهنده مشغول بودن یک کانال فیزیکی می باشد.
() PushFlit: فلیت داده را از کانال مجازی گرفته و روی کانال فیزیکی قرار می دهد.

۴-۵- Switch

هر سوئیچ تعدادی کانال فیزیکی فرستنده و تعدادی کانال فیزیکی گیرنده دارد که بوسیله آرایه ای از EndPortهای نوع دوم به آن متصل می شود. برای دسترسی به یک کانال فیزیکی خاص مثلا کانال پنجم طبق کد زیر عمل می کنیم:
Sw.outputPhysicalChannelEndPorts[5]. PhysicalChannelSender
متد ()ResetDelegate از این کلاس روی دو وضعیت از کانالهای فیزیکی و کانالهای تزریقی[۲۰] و خروجی[۲۱] گوش می کند: هنگامیکه مکان اولیه یک کانال پر شود (OnFirstSlotFilled) و هنگامیکه مکان آخر یک کانال خالی شود(OnLastSlotFreed).
()LockSwitch: سوئیچ را قفل می کند تا پیام درون آن منتقل شود.
()UnLockSwitch: سوئیچ را باز می کند تا پیام بعدی وارد آن شود.
()SetInputChannel,()SetOutputChannel: توسط این دو متد کانالهای ورودی و خروجی سوئیچ تنظیم می شوند.
()OnUnlockSwitch: هنگامیکه سوئیچ باز شود این متد فعال می شود.

۴-۶- Node

طرز تشکیل یک گره از سلسله مراتب خاصی پیروی می کند که این سلسله مراتب در هنگام ثبت وقایع بو سیله XML بسیار با اهمیت است.
هر گره از تعدادی مولفه مانند switch, RE, injectionChannel, ejectionChannel.... تشکیل شده است. کلاس Node در واقع اطلاعات مربوط یک گره را جمع آوری می کند.
هرگره بجای داشتن PE و RE یک PeEndPort و یک ReEndPort دارد که می تواند از طریق این دو به PE و RE متصل شود. همانطور که در کد اصلی برنامه می بینید (Node.cs) این دو متغییر در ابتدای کد کلاس Node نمونه گیری شده اند.کلاس Node سه تابع سازنده دارد که تابع سازنده بدون مقدار آن جهت ثبت وقایع در XML بکار برده می شود.
هر کلاسی یک متدی بنام ()Validate دارد که وظیفه آن بررسی شرایط لازم قبل از شبیه سازی می باشد. مثلا در کلاس Node اگر گره ای Pe یا Re نداشته باشد شبیه سازی نباید شروع شود و اینکار با ساختن یک استثناء[۲۲] انجام می شود. به این ترتیب کد با خطا روبرو شده و اجرا نمی شود.

---
[20] InjectionChannel
[21] EjectionChannel
[22] Exception

کلاس Node یک خاصیت[۲۳] بنام Switch دارد که در واقع به متغییر switch از کلاس Pe اشاره می کند و برای راحتی در نوشتن از آن استفاده می شود. همچنین دو خاصیت دیگر بنام های Pe و Re وجود دارد که وظیفه آنها برقراری اتصال بین PE و RE می باشد. در قسمت set این دو خاصیت ارتباط از نوع چهارم در EndPort برقرار می شود.

VCArbiter –۷–۴

مولفه ای است که به کانال فیزیکی وصل شده و بصورت دوره ای پیام های درون کانالهای مجازی متصل به یک کانال فیزیکی را روی کانال فیزیکی قرار می دهد. البته این کار به شرطی انجام می شود که بافر مقصد پیام خالی باشد، و گرنه پیام در کانال مجازی مربوطه مسدود می شود. متد Arbitrate از این کلاس این وظیفه را بعهده دارد.این متددرصورت برقراری شرایط لازم برای انتقال، متد Deliver از کلاس PhysicalChannelSender را صدا می زند. متد Arbitrate در سه صورت فعال می شود ، هنگامیکه پیامی به مقصد برسد، هنگامیکه اولین مکان خالی یک کانال فیزیکی آزاد پر شود و هنگامیکه آخرین مکان پر یک کانال فیزیکی آزاد خالی شود. (این سه وضعیت در قسمت Event Handler در کد اصلی مشخص شده است.)

InjectionChannel –۸–۴

این مولفه تعدادی بافر ورودی از نوع InputVCBuffer بنام SwitchVCBuffer دارد. این بافر ها از طرف سوئیچ می آیند. تعدادی بافر ورودی دیگر بنام PEVCbuffer نیز وجود دارد که از سوی PE تامین می شود.

PE –۹–۴

مولفه PE[24] شامل یک مصرف کننده پیام[۲۵] می باشد که می تواند آماده یا غیرآماده باشد. این مصرف کننده پیام از کلاس پایه baseMessageConsumer ارث برده شده است که متدی بنام ()Consume دارد.این متد بیشتر در جمع آوری اطلاعات آماری سیستم کاربرد دارد.

هر پیام دو زمان دارد، یکی زمان تولید[۲۶] و دیگری زمان ورود به شبکه[۲۷]. به کمک این زمانها و متد ()Consume می توان انواع تاخیر های ممکن برای یک پیام را حساب کرد.

هر PE یک messageGeneratorEndPort و یک NodeEndPort دارد که به وسیله ایندو به ترتیب به یک messageGenerator و یک Node متصل می شود. هر کدام از این دو مولفه یک PeEndPort دارند که بوسیله آن اتصال برقرار می شود. این اتصال از نوع دوم در EndPortها می باشد. نکته قابل توجه در متد ()Validate این است که اگر PE تولید کننده یا مصرف کننده پیام نداشته باشد، فقط یک اخطار صادر می شود، ولی اگر Node نداشته باشد از ادامه شبیه سازی جلوگیری بعمل می آید.

Flit, HeaderFlit, HeaderFlitSingle –۱۰–۴

---

[23] Property
[24] Processing Element
[25] Message Consumer
[26] Generation Time
[27] Enter Network Time

کلاسی است که از کلاس پایه TransferUnit مشتق شده است. در واقع این کلاس نوع خاصی از داده های عبوری از شبکه از نوع فلیت را مشخص می کند.کلاس HeaderFlit نوع خاصی از فلیت بنام فلیت سر را نشان می دهد و کلاس HeaderFlitSingle نوع خاصی از فلیت سر را که تک فلیتی است ، نشان می دهد.

SwitchingInfo –۱۱-۴

هر بافر از نوع InterconnectionBuffer یک اطلاعات مربوط به سوئیچینگ دارد که این اطلاعات از نوع کلاس SwitchingInfo می باشد. در این کلاس مشخص می شود که هر بافر در کدام کانال فیزیکی قرار گرفته است و شماره کانال مجازی مربوط به آن کانال فیزیکی چند است. همچنین مشخص می شود که بافر مربوطه به کدام کانال خروجی سوئیچ شده است.

MessageConsumer –۱۲-۴

کلاسی است که ازکلاس پایه BaseMessageConsumer مشتق شده است. این کلاس در واقع کار خاصی جز آمارگیری انجام نمی دهد. متغییری بنام warmUp در این کلاس به کار رفته شده که زمان پایان دوره گذرا در سیستم را نشان می دهد. بعد از زمان warmUp سیستم وارد دوره پایدار خود می شود و بدیهی است که معیارهای آماری مختلف مانند میانگین تاخیر پیام و ... در دوره پایدار سیستم با ارزش می باشند. متغییر هایی که آخر نامشان با Wختم می شود، نشان دهنده تعداد آنها بعد از زمان warmUp می باشد. متدی بنام ()consume در این کلاس وجود دارد که کار آمارگیری را انجام می دهد. اگر واحد انتقالی[۲۸] که وارد گره می شود، آخر پیام باشد، نشان دهنده این است که این پیام باید مصرف شود بنابراین خاصیت consumed از این کلاس مقدار درست می گیرد.

PhysicalInjectionChannel و PhysicalEjectionChannel –۱۳-۴

در هنگام اتصال یک سوئیچ به یک PE گاهی از یک کانال فیزیکی واقعی استفاده می کنیم که در اینصورت با توجه به نوع اتصال از دو کلاس فوق استفاده می کنیم.

RE[29] –۱۴-۴

RE یکی از مهمترین کلاسهای پایه می باشد. هر RE به سه مولفه از نوع Node، Switch و RoutingFunction متصل می شود.این اتصال بوسیله EndPort نوع دوم برقرار می شود.کلاسRoutingFunction ، شامل یک متد بنام ()Route می باشد که برای الگوریتم های مسیریابی مختلف این تابع باید بازنویسی شود.RE همچنین یک صف از پیام های بلوک شده دارد که هرگاه اتفاقی رخ دهد که مسیریابی ممکن شود (مثلا بافری خالی شود یا سوئیچی باز شود) این صف به صورت دوره ای[۳۰] بررسی شده و اگر پیامی بتواند مسیریابی شود، از صف بلوکه ها خارج می شود. این کار توسط متد ()CheckBuffer صورت می گیرد.

RoutingFunction –۱۵-۴

[28] Transfer Unit
[29] RoutingElement
[30] Round Robin

کلاسی است که فقط یک متد بنام ()Route دارد و وظیفه آن مسیریابی است. اطلاعاتی که متد ()Route نیاز دارد شامل: آدرس گره جاری، فلیت سر(برای دسترسی به پیام) ، بافر ورودی (برای بدست آوردن شماره کانال مجازی ورودی و خروجی) و سوئیچ (برای بدست آوردن اطلاعات کانالهای فیزیکی خروجی قابل استفاده) می باشد.

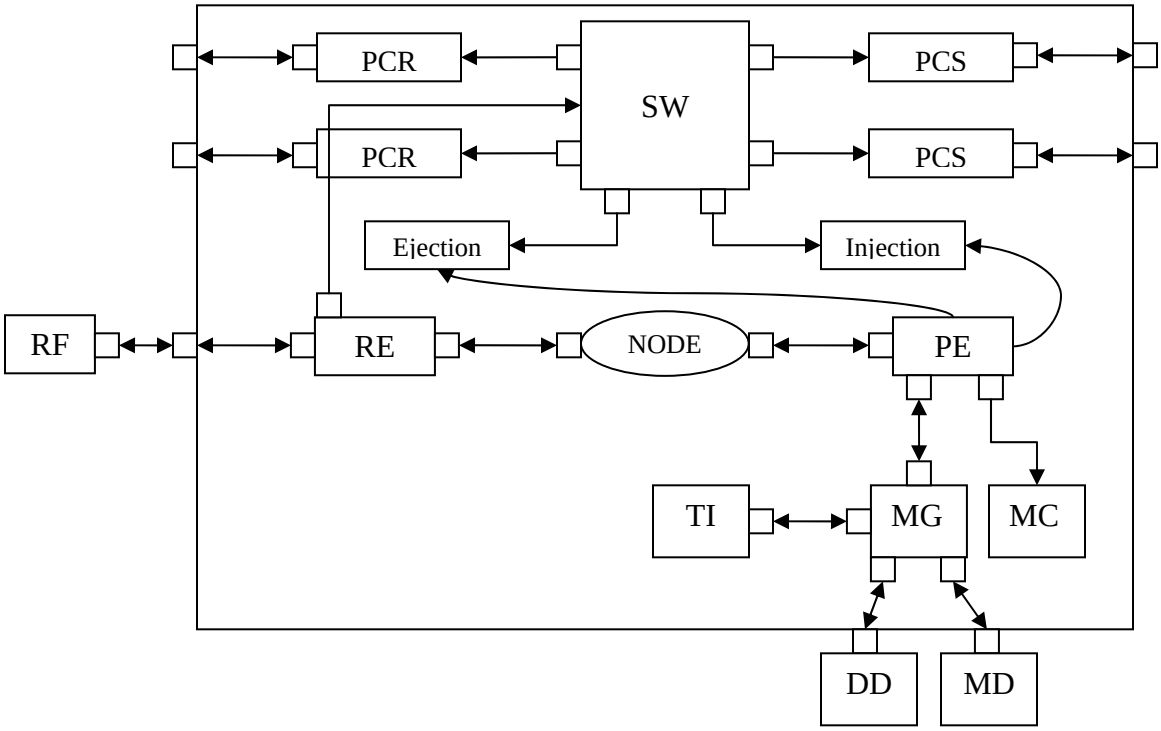
۴-۱۶- SimpleMessageGenerator

کلاسی است که از کلاس پایه BaseMessageGenerator مشتق شده است. این کلاس پیامها را بر اساس توزیع طول پیام، توزیع مقصد پیام و توزیع زمان بین دو پیام تولید می کند. همانطور که در کد برنامه می بینید این سه توزیع توسط EndPort نوع دوم به این کلاس متصل می شوند.

متد ()GenerateMessage در این کلاس وظیفه تولید پیام بر اساس سه توزیع مذکور را دارد.

۴-۱۷- بسته NetworkGeneration

این بسته زیر مجموعه ای است از بسته اصلی InterconnectionNetwork. دارای یک کلاس بنام NodeGenerator می باشد که وظیفه آن ساختن یعنی بهم متصل کردن اجزای یک گره پردازشی است. هر گره از بهم متصل کردن مولفه های مختلف بوجود می آید. شکل زیر چگونگی اتصال این مولفه ها را نشان می دهد.



PCR: PhysicalChannelReceiver	TI:TimeIntervalGenerator	RF:RoutingFunction
PCS: PhysicalChannelSender	MD: MessageLengthDistribution	MC: MessageConsumer
MG: MessageGenerator	DD:DestinationDistribution	

شکل ۹

یک گره در واقع یک MetaComponent می باشد.تابع BuildSimpleNode از کلاس MetaComponent مشتق شده است و وظیفه آن بهم متصل کردن اجزای سازنده یک گره پردازشی می باشد. همانطور که در کد برنام می بینید (NodeGenerator.cs) مولفه های مختلف، تولید شده و توسط متد AddXObject به گره پردازشی اضافه می شوند. هنگامی که یک مولفه جدید ساخته می شود، تابع سازنده بدون مقدار آن صدا زده شده و در خطوط بعدی پارامترهای آن مقدار می گیرند. این کار به خاطر ثبت وقایع در XML صورت گرفته است.

برای برقراری ارتباط بین مولفه ها EndPortها و MiddlePortها ساخته می شوند و توسط متد AddAndConnectLink این ارتباطات میسر می شود.

۵. بسته Queuing Network

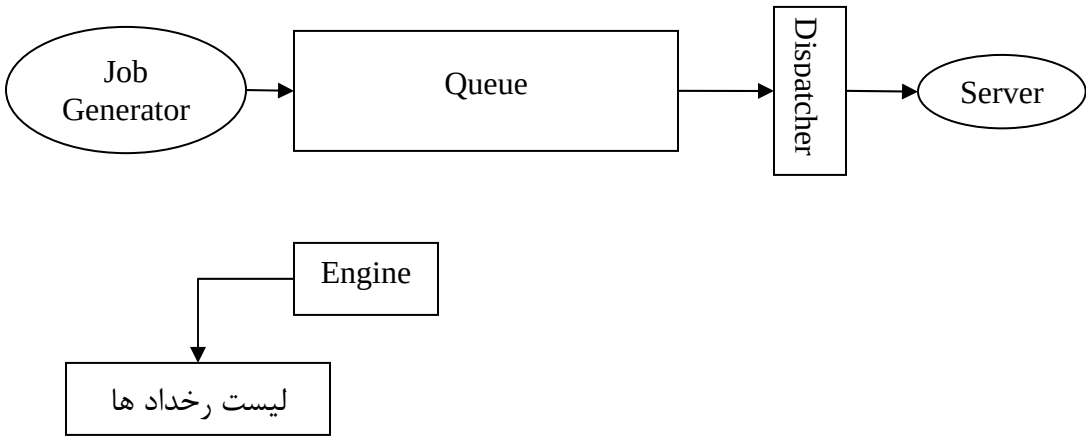
۵-۱- معرفی کلاسها

۵-۲- بررسی یک مثال ساده

در این قسمت به نحوه پیاده سازی یک سیستم صف M/M/1 توسط XMulator می پردازیم.

صف M/M/1، صفی است که ورود مشتریان به داخل آن و همچنین زمان خدمت آن طبق تابع توزیع نمایی می باشد و فقط یک خدمت دهنده وجود دارد.

مولفه های اصلی این سیستم عبارتند از: تولید کننده پیام[۳۱]، صف[۳۲]، توزیع کننده پیام[۳۳]، سرور و موتور اصلی برنامه.نحوه ارتباط این اجزا در شکل زیر نشان داده شده است.



شکل ۱۰

ابتدا یک پروژه جدید بنام SimpleQueue از نوع ConsoleApplication ایجاد می کنیم. چون مولفه های مورد استفاده در این مثال از کلاسهایBaseComponent و QueuingNetwork استفاده می کنند، بنابراین در ابتدا با راست کلیک کردن بر روی نام پروژه در قسمت SolotionExplorer این دو پروژه را به مثال مورد بحث اضافه می کنیم. چون در طول برنامه می خواهیم از کلاسهای این دو پروژه استفاده کنیم، بنابراین در ابتدای فایل اصلی پروژه (class1.cs) نام این دو کلاس را به صورت زیر ذکر می کنیم:

```
using Xmulator.BaseComponents;
using Xmulator.QueuingNetwork;
```

حال شروع می کنیم به ساختن تک تک مولفه ها. ابتدا یک تولید کننده پیام را می سازیم. دستور زیر این کار را انجام می دهد.

```
JobGenerator jobGen = new JobGenerator("JGen",evGen, q1);
```

JobGenerator کلاسی از بسته QueueingNetwork می باشد.jobGen شیء جدیدی است که از این کلاس نمونه گیری شده است. JGen نشاندهنده شناسه این شیء می باشد. هر تولید کننده پیامی باید یک تولید کننده رخداد[۳۴]

---

[31] JobGenerator
[32] Queue
[33] Dispatcher

داشته باشد، که در این مثال evGen این کار را انجام می دهد. همچنین محل مصرف این پیامها با q1 نشان داده شده است.evGen یک شیء از نوع کلاس پایه BaseXEventGenerator می باشد که از کلاس ExpTimeIntervalGenerator نمونه گیری شده است. این کار با دستور زیر صورت می گیرد:

```
BaseXEventGenerator evGen = new ExpTimeIntervalGenerator("EGen", 1 / arrivalRate,  1234);
```

یک کلاس پایه بنام ExpTimeIntervalGenerator وجود دارد که سه کلاس زیر از آن مشتق شده اند:
TimeIntervalGenerator, FixedTimeIntervalGenerator, ExpTimeIntervalGenerator
مامیتوانیم توزیعهای آماری دیگری رابه عنوان کلاس جدیداز نوع BaseTimeIntervalGenerator به کد برنامه اضافه کنیم. در این مثال از توزیع نمایی استفاده می کنیم. در توزیع نمایی نرخ ورود برابر λ است و عکس آن میانگین توزیع نمایی را نشان می دهد. بنابراین یک متغییر بنام arrivalRate برای نرخ توزیع نمایی در نظر می گیریم.
عدد ۱۲۳۴ که در این قسمت می بینید، عددی است که تولید کننده اعداد تصادفی طبق این عدد، اعداد تصادفی را ایجاد می کند. اگر این عدد عوض شود، اعداد تصادفی تولید شده نیز عوض می شوند. واضح است که اگر این عدد برای تولیدکننده های پیام یکی باشد، همگی آنها یک سری پیام را طی زمانهای یکسان تولید خواهند کرد که دور از واقعیت است. بنابراین عدد تولید اعداد تصادفی باید برای تولید کننده های پیام، متفاوت باشد.
نتیجه اجرای خط فوق تولید یکسری XEvent در فاصله های زمانی با توزیع نمایی خواهد بود. در واقع یک فرایند پوآسون خواهیم داشت.
در هنگام نمونه گیری از شیء jobGeb از q1 در تابع سازنده استفاده کردیم. قبل از بررسی q1 لازم است بدانید که دو نوع واسط[۳۵]بنامهای ObjectSource و ObjectSink وجود دارد.ObjectSource کلاسی است که متد GetObject() دارد و وظیفه آن تولید واحدهای انتقال[۳۶] می باشد. ObjectSink کلاسی است که متد PutObject() دارد و وظیفه آن دریافت واحدهای انتقال می باشد. با استفاده از این دو کلاس می توانیم براحتی مولفه ها را به همدیگر وصل کنیم.
در قسمت سوم از تابع سازنده شیء jobGen احتیاج به یک ObjectSink داریم تا پیام های تولید شده را دریافت کند. این شیء درواقع همان صف است که با q1 نشان دادیم. پس باید ابتدا q1 را بسازیم. اینکار با دستور زیر انجام می شود:

```
Xmulator.QueuingNetwork.Queue q1 = new Queue("q1", 100, warmup);
```

چون زبان C# خود کلاسی بنام Queue دارد، بنابراین برای اینکه اشتباه رخ ندهد و از کلاس مورد نظر استفاده کنیم، آنرا بصورت Xmulator.QueuingNetwork.Queue نشان می دهیم. عدد ۱۰۰ نشاندهنده ظرفیت صف است. متغییر warmup نشاندهنده مدت زمانی است که مولفه ها کار خود را شروع می کنند و به مقدار میانگین مورد نظر خود می رسند. مثلا یک صف ابتدای کار خالی است و مدتی طول می کشد تا طول صف به حد مورد انتظار خود برسد. به این مدت warmup می گویند.
حال می خواهیم یک توزیع کننده پیام بسازیم. وظیفه این توزیع کننده پیام، گرفتن پیام از ورودی (ObjectSource) و پخش پیامها به طور تصادفی در خروجیها(ObjectSink) می باشد. دستور زیر نشاندهنده این عمل است:

```
Dispatcher disp = new Dispatcher("disp.", q1, new IObjectSink[] {server});
```

---

[34] EventGenerator
[35] Interface
[36] Transfer Unit

با اجرای این دستور یک شیء بنام disp از کلاس dispatcher با شناسه disp نمونه گیری می شود. چون منبع این شیء صف است در قسمت ObjectSource آن نام صف تولید شده یعنی q1 را قرار می دهیم. قسمت ObjectSink آن شامل آرایه ای از خدمت دهنده ها است که چون در این مثال سیستم ما M/M/1 است، فقط یک خدمت دهنده بنام Server داریم.این خدمت دهنده از کلاس Server طبق دستور زیر نمونه گیری می شود:

```
Server server = new Server ("Ser.", 1, dist, warmup);
```

شناسه این شیء Ser است و عدد ۱ نشاندهنده ظرفیت خدمت دهنده است. متغییر warmup مانند قبل است. تنها نکته جدید تابع توزیع dist می باشد که خود طبق دستور زیر ساخته می شود:

```
RealDistribution dist = new ExpDistribution("dis", 1.0 / serviceRate, 234);
```

Dist یک توزیع نمایی است که از کلاس ExpDistribution نمونه گیری شده است. شناسه آن dist و میانگین آن عکس نرخ سرویس[۳۷] می باشد. عدد ۲۳۴ نیز مانند قبل تعیین کننده محدوده اعداد تصادفی تولید شده می باشد.

به این ترتیب همه مولفه ها ساخته شده و بهم وصل شده اند.حال باید دستوری بنویسیم که شبیه سازی شروع شود. دستور زیر اینکار را انجام می دهد:

```
XEngineFactory.XEngine.StartSimulation (-1, 1000000);
```

عدد ۱۰۰۰۰۰۰ معرف تعداد گامهایی[۳۸] است که شبیه ساز باید طی کند. حال اگر برنامه را اجرا کنیم، شبیه سازی با موفقیت انجام می شود. قبل از اینکه به نحوه نمایش خروجی بپردازیم ، لازم است توضیحی در مورد بسته log4net بیاوریم:

Log4net یک بسته برای گزارش گیری از برنامه های نوشته شده به زبان #C می باشد.(مشابه Log4J در زبان Java). با استفاده از این بسته می توانیم اولا سطوح گزارش گیری را تعیین کنیم، یعنی مشخص کنیم تا چه جزئیاتی در گزارش ثبت شود، ثانیا مکانهای گزارش گیری[۳۹] را مشخص کنیم. برای استفاده از این بسته باید در ابتدای کد اصلی برنامه با دستور

```
Using Log4net;
```

آنرا اعلام کنیم و یکبار در تابع ()main برنامه اصلی متد ()Configure آنرا اجرا کنیم.خطوط زیر را باید به ابتدای برنامه اضافه کنیم:

```
StaticSettings.DisableLogs = true;
log4net.Config.DOMConfigurator.Configure ();
```

خروجی های مختلفی از این مثال می توان گرفت. مثلا فرض کنید میخواهیم متوسط طول صف را ببینیم. اینکار با دستور زیر انجام می شود:

```
Console.WriteLine ("Average Queue Length = " + q1.probe.AvgQueueLengthW);
```

همانطور که ملاحظه می کنید متد AvgQueueLengthW متوسط طول صف (بعد از warm up) را محاسبه می کند. این متد یکی از متدهای کلاس probe می باشد.

---

[37] Service Rate
[38] Step
[39] Provider

۶. بسته Meta K_ary N_Cube

از این بسته برای ساختن یک شبکه دلخواه و همچنین نوشتن الگوریتم مسیریابی استفاده می شود.

۶-۱- ساختن یک شبکه میان ارتباطی

در این قسمت نحوه ساختن یک شبکه میان ارتباطی کلی از نوع k_ary N_cube توسط کلاس KaryNcube توضیح داده می شود.

هر قسمت در شبکه یک MetaComponent در نظر گرفته می شود. برای ساخت شبکه از بالا به پایین شروع می کنیم و نام MetaComponent اصلی را network می گذاریم. پارامترهای ورودی که برای ساخت یک شبکه لازم است توسط یک نمونه از کلاس KaryNcubeParameterSet تامین می شوند. در این کلاس پارامترهای مربوط به گره پردازشی خود از کلاس SimpleNodeParameterSet بدست می آیند. در هنگام اجرای برنامه اگر در درایو C فایل run.xml وجود داشته باشد برنامه پارامترهای ورودی را از این فایل می خواند، در غیر اینصورت همان پارامترهای پیش فرض کلاسهای مذکور مورد استفاده قرار می گیرند. دستورات xml در فایل run.xml باید دقیقا طبق ترتیب کلاس KaryNcubeParameterSet ساخته شوند.

متد ()karyPres از کلاس KaryNcube وظیفه نمایش آدرس یک گره را به عهده دارد. هر آدرسی با توجه به تعداد بعدهای آن که همان n است، قسمت می شود و قسمتهای مختلف یک آدرس با علامت @ از هم جدا می شود.

متد اصلی در این کلاس متد ()BuildNetwork می باشد که وظیفه ساخت شبکه را به عهده دارد. نکته قابل توجه این است که اگر k برابر ۲ باشد درجه شبکه نیز ۲ خواهد بود و در این حالت خاص یک فوق مکعب[۴۰] ساخته خواهد شد. در سایر موارد درجه شبکه دو برابر n خواهد شد.در ادامه این متد و در یک حلقه به اندازه سایز شبکه گره ها ساخته می شوند. شناسه[۴۱] هر گره مشخص کننده ترتیب ساخته شدن آن در توالی بالا به پایین می باشد. به این صورت که ابتدا شناسه شبکه و سپس بهمراه یک نقطه نام گره و آدرس آن که به همان صورتی است که متد karyPres نشان می دهد، آورده می شود. این شیوه نامگذاری و جدا کردن اجزای سازنده یک مولفه به وسیله نقطه در سراسر کد شبیه ساز دیده می شود.به طور مثال nt.n_@2@3@1 نشاندهنده گره ای از یک شبکه ۳ بعدی با اندیسهای۲و۳و و۱شناسه nt می باشد بهنگام ساختن یک گره تمام مولفه های لازم آن مانند RoutingFunction ، MessageConsumer و ... نیز ساخته می شوند.

Mcp یک متغییر از نوع MessageConsumerProbe می باشد. همانطور که قبلا گفته شد، تمام گزارش های یک شبکه از متد Consume گرفته می شود. بنابراین mcp آمار کلی شبکه مانند میانگین تاخیر پیام و ... را نشان می دهد. این آمارها روی بلوکی از داده که msgBlockSize سایز آنرا مشخص می کند، اعمال می شود. گرفتن آمار بلوکی ما را در شناختن رفتار شبکه کمک می کند. اگر آمار تقریبا ثابت باشد یعنی شبکه در حالت عادی خود کار می کند ولی اگر آمار مرتبا رو به افزایش باشد نشان می دهد که یا شبکه هنوز به حالت پایدار خود نرسیده است و شبیه سازی باید کماکان ادامه داشته باشد، یا شبکه به اشباء رسیده است، که در این صورت شبیه سازی باید متوقف شود.

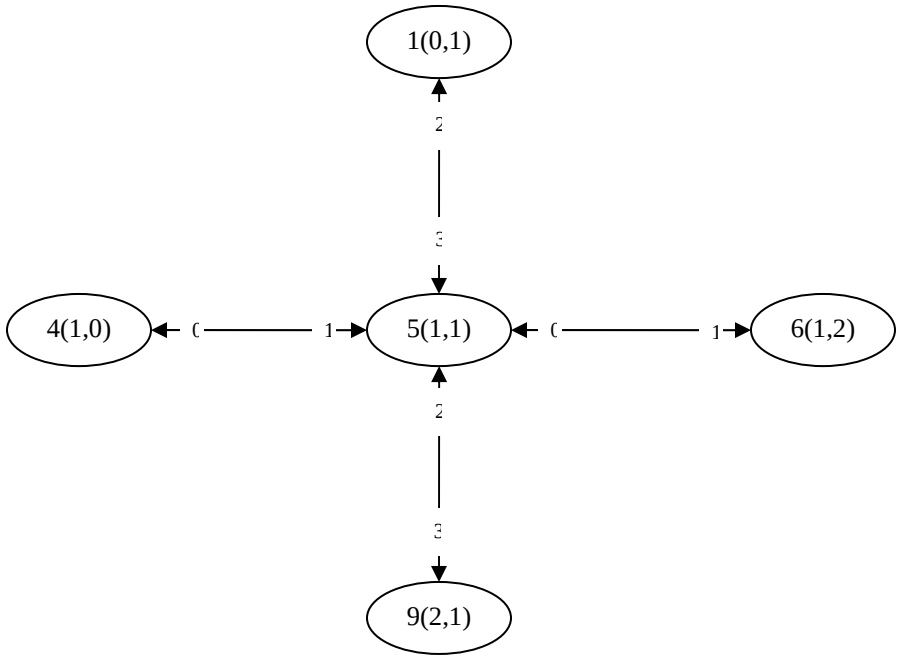
پس از اینکه مولفه های یک شبکه همگی ساخته شدند، در حلقه بعدی بهم متصل می شوند. برای بدست آوردن یک مولفه در هنگام اتصال کافی است که نام MetaComponent اصلی را آورده و داخل کروشه شناسه مولفه مورد نظر را به همان ترتیب شکل گیری آن از بالا به پایین ذکر کنیم. به طور مثال اگر بخواهیم به گره مثال قبلی دسترسی داشته

---

[40] Hypercube
[41] Id

باشیم می توانیم با کد network[nt.n_@2@3@1] اینکار را انجام دهیم. اگر k برابر ۲ باشد، آدرس همسایه هر گره از XOR کردن آدرس گره با عدد یک که به اندازه بعد فعلی شیفت داده شده است بدست می آید. به طور مثال همسایه گره ۳ از یک فوق مکعب ۳ بعدی گره های ۲،۱،۷ می باشند. پس از پیدا کردن گره همسایه با استفاده از متد AddAndConnectLink یک اتصال[۴۲] بین کانال فیزیکی فرستنده[۴۳] از گره مبدا و کانال فیزیکی گیرنده[۴۴] از گره همسایه ایجاد می کنیم. برای مثال اتصال اول بین گره ۳ و گره ۲ از یک فوق مکعب سه بعدی با شناسه NT.l_@0@1@1,@0@1@0 بین گره ۳ با مختصات nMeta[NT.n_@0@1@1.p_pcs_0] و گره ۲ با مختصات neiMeta[NT.n_@0@1@0.p_pcr_0] بدست می آید.

اما اگر k>2 باشد در اینصورت هر گره در هر بعد دو همسایه خواهد داشت که آدرس همسایه در هر بعد با افزایش و کاهش یک واحد از اندیس آدرس گره در آن بعد بدست می آید. در این حالت تعداد کانالهای فیزیکی فرستنده و گیرنده هر گره دو برابر تعداد ابعاد شبکه (2n) خواهد بود. طبق قرارداد از اعداد زوج برای نشان دادن کانالهای فیزیکی افزایشی و از اعداد فرد برای نشان دادن کانالهای فیزیکی کاهشی استفاده خواهیم کرد. برای مثال در یک 4_ary 2_cube گره شماره ۵ با اندیس (۱و۱) با گره با اندیسهای (۱و۰)،(۱و۲)،(۰و۱)،(۲و۱) همسایه خواهد بود. هر گره در این شبکه ۴ کانال فیزیکی فرستنده و ۴ کانال فیزیکی گیرنده دارد و برای نمونه اتصال بین گره شماره ۵ و گره شماره ۱ از طریق کانال فیزیکی فرستنده شماره ۳ از اولی و کانال فیزیکی گیرنده شماره ۲ از دومی برقرار می شود. در شکل زیر چگونگی این ارتباط را مشاهده می کنید.



شکل ۱۱

[42] Link
[43] PhysicalChannelSender
[44] PhysicalChannelReceiver

۶-۲- الگوریتم مسیریابی E_cube

در هنگام ساخت یک شبکه که در قسمت قبلی گفته شد، می بایست از یک الگوریتم مسیریابی استفاده می کردیم. الگوریتم مسیریابی E_cube از ابتدایی ترین الگوریتم های مسیریابی قطعی[۴۵] می باشد. برای پیاده سازی این الگوریتم کلاس EcubeRoutingFunctionKary نوشته شده است. متد اصلی این کلاس متد Route() می باشد که وظیفه آن پیدا کردن یک کانال فیزیکی و همچنین شماره کانال مجازی آن کانال فیزیکی می باشد. در ابتدای این متد آدرس گره فرستنده و آدرس گره گیرنده را به طریق نوشته شده در کد بدست می آوریم. اگر این دو آدرس با هم برابر باشند یعنی به مقصد رسیده ایم، در غیر اینصورت در شبکه های فوق مکعب (k=2) حاصل XOR این دو آدرس و در شبکه های غیر فوق مکعب (k>2) تفاضل اندیسهای آدرس گره مبدا و مقصد که در آرایه off نگهداری می شود جهت دهنده انتخاب کانال فیزیکی فرستنده می باشد. پس از انتخاب کانال فیزیکی فرستنده یکی از کانالهای مجازی آن که مشغول نیست به ترتیب شماره آن کانال مجازی انتخاب می شود.

۶-۳- الگوریتم مسیریابی Duato

پایه و اساس این الگوریتم همان الگوریتم E_cube می باشد. تفاوت این دو در این است که در الگوریتم Duato کانالهای مجازی به دو کلاس تقسیم می شوند که کانالهای مجازی کلاس اول به هر صورت دلخواه انتخاب می شوند و در صورت عدم انتخاب کانال مجازی از کلاس اول کانالهای مجازی کلاس دوم انتخاب خواهند شد. در کد همانطور که می بینید کل کانالهای مجازی به جز آخری در کلاس اول قرار گرفته اند و فقط کانال مجازی آخر در کلاس دوم قرار گرفته است. بنابراین اگر یک یا بیشتر از کانالهای مجازی کلاس اول در دسترس باشد، یکی از آنها به طور تصادفی انتخاب خواهد شد و الگوریتم خاتمه می یابد. ولی اگر هیچکدام از کانالهای مجازی کلاس اول در دسترس نباشند آنگاه کانال مجازی کلاس دوم مورد استفاده قرار خواهد گرفت.

---

[45] Deterministic