میان افزار جاوا سمفونی

مقدمه

ازمیان افزار JavaSymphony جهت ایجاد سیستم توزیع شده استفاده میشود . این ابزار یکی از کاراترین میان افزار های طراحی شده جهت ساخت سیستم های توزیع شده است که برنامه نویس را از پرداختن به بسیاری از جزئیات برنامه نویسی بی نیاز می کند.

در بخش2 به معرفی JavaSymphony پرداخته, امکانات و توانایی هایی که این ابزار جهت ساخت سیستم های توزیع شده و موازی در اختیار برنامه نویس قرار می دهد را معرفی کرده, نحوه استفاده از JavaSymphony برای برنامه نویسی و ساخت سیستم های توزیع شده به زبان جاوا, خصوصیات کلاس ها و توابع مختلف این ابزار و ویژگی های آنها بیان می شود. در بخش3 امکانات پیشرفته تر JavaSymphony که در نسخه های بعدی آن به منظور افزایش کارایی برنامه های ساخته شده توسط این ابزار به آن اضافه شده (مانند تکنیک های همگام سازی پروسه ها, ایجاد اشیا چند نخی, بسط معماری مجازی, تبدیل پویا اشیا و...) را بررسی خواهیم کرد. در بخش 4 نحوه نصب, راه اندازی و استفاده از JavaSymphony درکامپیوتر های مختلف شبکه برای ایجاد یک زیر ساخت ارتباطی و تعریف معماری فیزیکی سیستم توزیع شده را بررسی خواهیم کرد.

2. مفاهیم اصلی JavaSymphony

اغلب سیستمهای توزیع شده یا موازی که به وسیله جاوا پیاده سازی می شوند نیازمند پرداختن برنامه نویس به کارهای جزئی بسیار و خسته کننده هستندکه احتمالا منجر به اشتباه برنامه نویس نیز می شوند. JavaSymphony یک ابزار برنامه نویسی جهت پیاده سازی سیستم های توزیع شده و موازی است که دامنه وسیعی از سیستم های همگن از سیستم هایی با خوشه های کوچک تا سیستم های محاسباتی وسیع را پشتیبانی می کند. این ابزار تماما به زبان جاوا نوشته شده وبا تمام ماشینهای مجازی جاوا (JVM) سازگاری دارد. می توان گفت در مقایسه با دیگر ابزارهای موجود، JavaSymphony امکانات بهتر و قابل انعطاف تری را برای مکان یابی اشیاء و بارگذاری متعادل در اختیار برنامه نویس قرار می دهد.

JavaSymphony در واقع یک کتابخانه از کلاس های جاوا است که به برنامه نویس امکان کنترل Locality موازی سازی و بارگذاری متعادل را در سطح بالایی می دهد, و برنامه نویس را از پرداختن به مسائل جزئی مانند ایجاد و استفاده از Remot-Proxy ها, برنامه نویسی نخ ها, برنامه نویسی سوکت ها و پردازش خطا ها بی نیاز می کند. به کمک این ابزار یک معماری مجازی، ازگره های محاسباتی (کامپیوتر های شبکه) تعریف می شود. برنامه نویس می تواند با تعریف معماری مجازی دلخواه خود، سلسله مراتبی ازگره های محاسباتی فیزیکی برای سیستم توزیع شده خود بسازد. اشیاء برنامه می توانند به صورت پویا در هر کدام از مؤلفه های این معماری مجازی مستقرشده یا بین آنها حرکت کنند. مجمو عه ای ازگره های سطح بالا جهت کنترل پارامترهای مختلف نرم افزاری و سخت افزاری سیستم در دسترس است . اشیاء می توانند توسط فراخوانی متدهای یکدیگر به سه روش سنکرون، آسنکرون و یک طرفه با یکدیگر ارتباط برقرار کنند. بارگذاری کلاسهای برنامه به صورت دلخواه در گره های مختلف محاسباتی نیز می تواند موجب کاهش حافظه مورد نیاز در کل سیستم توزیع شده شود. بعلاوه می توان اشیاء را به طور دائم در حافظه های جانبی ذخیره, نگهداری و بازیابی کرد.

اغلب پروژهای تحقیقاتی که یک زیر ساخت نرم افزاری برای برنامه های توزیع شده وموازی ارائه می کنند معمولا مانع از کنترل مکان استقرار اشیاء توسط برنامه نویس می شوند. با توجه به اینکه معمولا برنامه نویس اطلاعات بیشتروکامل تری در مورد ساختار سیستم توزیع شده و نحوه توزیع شده گی مطلوب اشیاء بر روی شبکه را دارد از این رو اکثر سیستم های توزیع شده که توزیع اشیاء درگره های محاسباتی و حرکت بین آنها را به صورت اتوماتیک انجام می دهند موجب کاهش کارایی سیستم توزیع

شده خواهند شد. زیرا که از اطلاعات مهمی که برنامه نویس در مورد سیستم توزیع شده دارد استفاده نمی کنند و در نتیجه ممکن است ارتباط بین اشیاء توزیع شده در شبکه با یکدیگر زیاد شده و کارایی کل سیستم بسیار پایین بیاید.

2. 1. ویژگی های Java Symphony

JavaSymphony یکی از ابزارهای ساخت سیستمهای توزیع شده و موازی است که یک زیر ساخت مناسب جهت ساخت برنامه های توزیع شده و موازی ارائه می کند. همانطورکه گفته شد این ابزار توسط زبان جاوا پیاده سازی شده وبه صورت کتابخانه ای از کلاسهای جاوا (.class) موجوداست که می توان از این کلاس ها در هر برنامه نوشته شده به زبان جاوا استفاده کرد. برخی از خصوصیات مهم این ابزار عبارتند از:

- امکان تعریف یک معماری مجازی توزیع شده: برنامه نویس می تواند یک معماری مجازی برای سیستم توزیع شده خود تعریف کرده و به این ترتیب سلسله مراتبی از گره های محاسباتی فیزیکی را برای برنامه خود تشکیل دهد. معماری مجازی از مولفه های زیر تشکیل می شود: گره های محاسباتی، خوشه ها که مجموعه ای ازگره های محاسباتی هستند، سایت هاکه مجموعه ای از خوشه ها هستند و دامنه ها که مجموعه ای از سایتها هستند. در هنگام تعریف یک معماری مجازی می توان با استفاده از امکانات تعریف شرایط برای گره های محاسباتی، فقط از منابع محاسباتی مناسب یا دلخواه در شبکه، جهت ساخت سیستم توزیع شده استفاده کرد. (بعنوان مثال استفاده از کامپیوترهای بیکار در شبکه) معماری های مجازی چندگانه نیز قابل تعریف هستند که می توانند مولفه های خود را به اشتراک بگذارند.
- دسترسی به پارامترهای سیستم : JavaSymphony مجموعه ای از API های سطح بالا برای کنترل بسیاری از پارامترهای سیستم ارائه می کند از جمله می توان به پارامترهای CPU Load (میزان کار بارگزاری شده به CPU)، idle Times (درصد بیکاری)، میزان حافظه در دسترس، تعداد پروسه ها و نخهای موجود در سیستم, تاخیر شبکه و پهنای شبکه اشاره کرد. این پارامترها می توانند از سیستم زمان اجرای JavaSymphony یا JRS درخواست شده و برای کنترل استقرار اشیاء در مولفه های مختلف معماری توزیع شده, حرکت اشیاء بین مولفه ها و بارگذاری متعادل استفاده شوند.
- استقرار اشیاء در مولفه های محاسباتی بصورت اتوماتیک وکنترل شونده از طرف کاربر: برنامه نویس می تواند ایجاد و استقرار یک شی در یکی از مولفه های معماری مجازی را کنترل کند. استقرار یک شی در یک مولفه می تواند با توجه به مکان اشیا دیگر که با این شی در ارتباط خواهندبود, صورت گیرد. بعنوان مثال ممکن است مجموعه ای از اشیاء که با یکدیگر ارتباط زیادی دارند برروی مولفه هایی از معماری مجازی که نزدیک یکدیگر هستند مستقر شوند. اگر برنامه نویس دقیقا" محل یک شی را جهت اجراء در آن مکان مشخص نکند JRS بصورت اتوماتیک در مورد محل قرارگیری شی در بین گره های محاسباتی فیزیکی تصمیم گیری خواهد کرد.
- حرکت اشیاء بین مولفه های محاسباتی بصورت اتوماتیک و قابل کنترل از طرف کاربر:JavaSymphony از حرکت اشیاء بین گره های محاسباتی بصورت اتوماتیک و یا تحت کنترل کاربر پشتیبانی می کند.
- فراخوانی سنکرون, آسنکرون و یک طرفه متدها: همانطورکه می دانیم تمامی فراخوانی روالهای دور در جاوا بصورت سنکرون انجام می گیرد. علاوه بر این مکانیزم، JavaSymphony دو روش دیگر برای فراخوانی متدهای دور را در اختیار برنامه نویس قرار می دهد: 1- فراخوانی آسنکرون که در این روش یک Handle که برای بررسی آماده بودن نتیجه در آینده استفاده خواهد شد, بعنوان نتیجه فراخوانی روال برگشت داده می شود. 2- فراخوانی یک طرفه : که دراین روش طرف فراخوانی کننده متد، منتظر دریافت هیچ نتیجه ای از فراخوانی متد نمانده، بلکه متد دو را فراخوانی کرده و بدون اینکه بداند اجرای آن کی به اتمام خواهد رسید کارهای خود را دنبال می کند.

- بارگذاری کلاسهای دور بصورت دلخواه : به کمک JavaSymphony بجای آنکه لازم باشد تمام فایلهای حاوی کلاس های برنامه توزیع شده در تمامی گره های محاسباتی ذخیره شوند، این فایل ها می توانند در زمان نیاز در یک گره به آن گره بارگزاری شده و استفاده شوند. این خصوصیت می تواند باعث کاهش کل حافظه مورد نیاز برنامه توزیع شده شود.

بعلاوه JavaSymphony از اشیاء مانا نیز پشتیبانی می کند. یعنی به برنامه نویس اجازه میدهد اشیاء برنامه را در حافظه های جانبی ذخیره کرده و بازیابی کند. این ابزار نیازی به هیچ گونه تغییردر زبان جاوا یا JVM یا کامپایلر جاوا نداشته و بصورت کتابخانه ای از Class ها پیاده سازی شده و قابل استفاده است.JavaSymphony براساس سیستم های مبتنی برAgent ساخته شده و فعلا در حال ارزیابی است.

2. 2. معماری مجازی توزیع شده گی پویا (Dynamic Virtual Distributed Architectures)

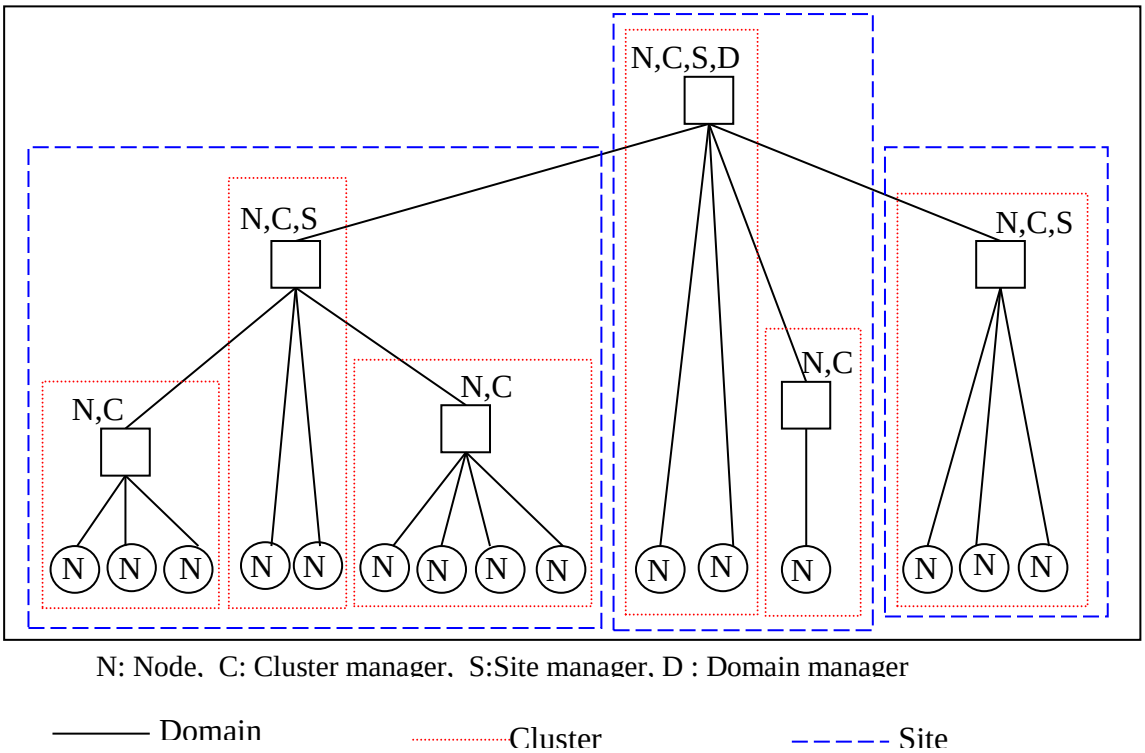
JavaSymphony از مکان یابی, بارگذاری متعادل و حرکت اشیاء بین گره های محاسباتی به صورت اتوماتیک پشتیبانی می کند (بدون اینکه برنامه نویس دخالتی داشته باشد) اما تجربه نشان داده که سیستمهای تمام اتوماتیک فعلی معمولا نمی توانند اطلاعات کافی (به اندازه برنامه نویس) در مورد ساختار سیستم توزیع شده, بدست آورندو در نتیجه کارایی ضعیفی را به دنبال دارند.

از این رو درJavaSymphony یک حالت نیمه اتوماتیک برای پیاده سازی و پیکربندی سیستم توزیع شده ارائه شده است. به این صورت که بسیاری از کارهای جزیی و خسته کننده و مسائل سطح پایین برای پیاده سازی سیستمهای توزیع شده (مانند ایجاد و استفاده از پروکسی ها ، برنامه نویسی سوکتها و...) توسط JavaSymphony انجام می شود اما تصمیم گیری های مهم و استراتژیک در مورد ساختار سیستم توزیع توسط برنامه نویس صورت می گیرد. برنامه نویس جهت تعریف ساختار سیستم توزیع شده کارهای زیر را انجام می دهد:

- تعریف یک معماری مجازی توزیع شده با مشخص کردن گره های محاسباتی، خوشه ها، سایتها ودامنه ها: این معماری برای اجرای برنامه کاربردی بصورت توزیع شده برروی شبکه استفاده خواهد شد. برنامه نویس میتواند برخی شرایط سیستمی برای گره های محاسباتی تعریف کند تا تنها کامپیوترهایی در شبکه که حائز آن شرایط هستند (چه نرم افزای و چه سخت افزاری) بعنوان منابع محاسباتی فیزیکی برای معماری تعریف شده, استفاده شوند.
- استقرار داده ها و اشیاء با توجه به ارتباط آنها با داده های دیگربر روی گره ها : بعنوان مثال اشیایی که ارتباط زیادی با هم دارند ممکن است برروی گره های محاسباتی نزدیک به هم و یا حتی برروی یک گره بارگذاری شوند.
- استقرار اشیاء یا داده ها برروی گره های محاسباتی با توجه به شرایط سیستمی آنها : بعنوان مثال استقرار اشیا در گره هایی با کمترین میزان کاری یا بالاترین حافظه در دسترس.
- بارگذاری کدهای مورد نیاز (کد کلاسها) در گره های محاسباتی هنگام نیاز به آنها: با استفاده از این ویژگی نیازی به ذخیره سازی کد تمام کلاسهای برنامه توزیع شده در تمام گره های محاسباتی نبوده و میتوان کد هر کلاس را هنگام نیاز به آن در یک گره خاص به آن بارگذاری کرد.

JavaSymphony مفهومی بنام Dynamic Virtual distributed architectures (که از این به بعد معماری مجازی می نامیم) ارائه می کند, که به برنامه نویس امکان تعریف ساختار شبکه ای از منابع محاسباتی را می دهد. پس از تعریف این ساختار می توان اشیاء مختلف موجود در برنامه را در مولفه های تشکیل دهنده این معماری مستقر کند یا اشیاء موجود درگره های محاسباتی را به گره های دیگر حرکت دهد. همچنین می تواند بین تعداد اشیاء بارگذاری شده در گره های مختلف موازنه ایجاد کند ونیز کدهای لازم را هنگام نیاز به آنها درهر یک ازگره ها بارگذاری کند. هر معماری مجازی در حقیقت دامنه

ای است که به سایتها، خوشه ها وگره هایی تقسیم شده است. (شکل 1) در پایین ترین سطح گره هایا گره های محاسباتی قرار دارند که در حقیقت معادل یک منبع محاسباتی فیزیکی مانند PC یا WorkStation میباشند. گره های مختلف می توانند با یکدیگر ترکیب شده و تشکیل یک خوشه را دهند که در حقیقت معادل یک شبکه محلی از چند PC یا WorkStation است سطح بعدی را سایت ها تشکیل می دهند که مجموعه ای از خوشه های متصل به هم مثلا به وسیله یک WAN یا LAN میباشند. در بالاترین سطح سایتها می توانند با یکدیگر ترکیب شده و تشکیل یک دامنه (domain) را دهند که در حقیقت یک شبکه محاسباتی بزرگ و توزیع شده است. توجه داشته با شید که هرگره متعلق به یک سه تایی یکتای (خوشه- سایت – دامنه) می باشد بطور مشابه هر خوشه یه یک دو تایی یکتای (سایت -دامنه) وهرسایت به یک دامنه خاص تعلق دارد.



شکل 1 . معماری مجازی

معماری مجازی می تواند به صورت پویا ایجاد شده یا تغییر داده شود هرمؤلفه ازیک معماری مجازی(گره ، سایت ، خوشه ، دامنه) توسط مدیری که قابل رویت برای برنامه نویس نبوده و به عنوان بخشی از JRS پیاده سازی شده، کنترل می شود.

2. 3. مدل برنامه نویسی Java Symphony

در حالت کلی هربرنامه کاربردی که از JavaSymphony استفاده می کند باید ابتدا خود رادر سیستم زمان اجرای JavaSymphony یا (JRS) ثبت یا Register کند. پس از آن می تواند معماری مجازی خود را تعریف کند. اشیاء می توانند هم در گره های محاسباتی محلی و هم درگره های دیگر به صورت دور ایجاد شده وبین گره های شبکه حرکت کنند. برای ارتباط بین اشیاء موجود در سیستم توزیع شده نیز JavaSymphony از سه مدل فراخوانی متدهای دورپشتیبانی می

کند (سنکرون، آسنکرون و یک طرفه). در نهایت برنامه کاربردی قبل از اتمام, خود را از وضعیت ثبت شده در JRS خارج می کند(unregister) این کار باعث آزاد شدن منابع JRS و حافظه تخصیص یافته به برنامه خواهد شد.

2. 3. 1. Unregisterو Register کردن برنامه کاربردی درjrs :

همانطور که ذکر شد, هر برنامه کاربردی که ازJavasymphony استفاده میکند ابتدا باید خود را درJRS ثبت یا Register کند تاJRS از وجود این برنامه مطلع شود.

```
…
register application with jrs//
JSRegistration reg=new JSRegistration();
…
//un- register application
reg.unregister();
```

هر برنامه ای هم که به زودی به پایان خواهد رسید باید JRS را با unregister کردن خود مطلع کند تا منابع تخصیص یافته به برنامه توسط JRS آزاد شود.

2. 3. 2.ایجاد معماری توزیع شدۀ مجازی

برای مشخص کردن مکان استقرار اشیاء برنامه در شبکه، JavaSymphony مفهومی به نام معماری مجازی ارائه می کند که برنامه نویس می تواند با استفاده از آن توپولوژی دلخواهی از یک شبکه ازگره های محاسباتی تعریف کند. این ساختار مرکب ازگره ها,خوشه ها,سایت ها ویک دامنه است امکاناتی جهت تعریف مجموعه ای از شرایط سیستمی به منظورکنترل بار گذاری اشیاء در گره های محاسباتی ارائه شده است. به این شکل که برنامه نویس مشخصات وپیش شرایط کامپیوتری که لازم دارد را مشخص می کندو JRS یک گره محاسباتی مناسب با آن ویژگی در شبکه را انتخاب کرده و شی را در آن مستقر می کند. برنامه نویس این شرایط را با استفاده ازپارامترهای استاتیک و دینامیک سیستم مشخص می کند پارامترهای استاتیک به پارامترهایی گفته می شود که حین اجرای برنامه در سیستم توزیع شده تغییری نمی کنند مثلا سیستم عامل ماشین،نام ماشین,سرعت cpu. پارامترهای دینامیک به پارامترهایی گفته می شود که ممکن است حین اجرای برنامه در سیستم توزیع شده تغییر یابد مانند میزان حافظه آزاد، میزان بیکاری سیستم، تعداد فراخوانی های سیستمی وتعداد نخ ها.

JavaSymphony امکان ایجاد شی ای از نوع jsConstraints را می دهد که برنامه نویس می تواند مجموعه ای از شرایط سیستمی را تعریف کرده و دراین شی ذخیره کند. شرایط، با فراخوانی روال setConstraints به این شی اضافه می شوند. ساختار این روال به شکل زیر است:

setConstraints(system-prameter, relational-operator, number/string)

که در آن relational-Operator یک عملگر رابطه ای دلخواه, Number/String یک داده عددی یا متنی و System – Parameter یکی از پارامترهای سیستمی است. به عنوان مثال کد زیر را در نظر بگیرید

```
JSConstrainst constr= new JSConstraints();
constr.setConstraints (JSConstants.NODE_NAME, "!=" :," milena");
constr.setConstraints(JSConstants.CPU_SYS_LOAD,"<=",10);
constr.setConstraints(JSConstants.IDLE,">=",50);
constr.setConstraints(JSConstants.AVAIL_MEM,">=",50);
constr.setConstraints(JSConstants.SWAP_SPACE_RATIO,">=",0.3);
```

مجموعه ای از شرایط سیستمی, درشئی به نام Constr جمع شده اند. این شرایط نشان می دهند که گره محاسباتی نباید دارای نام Melina بوده،کمتر از 10 درصد کار سیستمی داشته، بیشتر از 50 درصد زمان ها بیکار بوده و حداقل 50mb حافظه خالی داشته باشد.همچنین نرخ استفاده از فضای موقت این گره باید کمتر از 0.3 درصد باشد. در حالت کلی برنامه نویس می تواندبا استفاده ازحدود 40 پارامتر سیستمی شرایط گره دلخواه خود راتعریف کند. همانطورکه ذکرشد برنامه نویس می تواند با ایجاد گره ها، خوشه ها و سایت هایی که تشکیل یک دامنه را می دهند یک معماری مجازی را ایجاد می کند.در ادامه نحوه ایجاد هر یک از این مولفه ها بررسی می شود .

Nodes: گره ها می توانند به صورت زیر ایجاد شده یا آزاد شوند:

```
//request arbitrary node
Node n1 =new Node ();
//request node with name "rachel"
Node n2 =new Node ("rachel");
//request node for which constraints hold
Node n3 =new Node ( constr);
//determine the associated cluster,site,and domain  of   n1
Cluster c1= n1.getCluster();
Site s1= n1.getSite();
Domain d1= n1.getDomain();
n1.freeNode();       //release node n1 from application
```

گره n1 بدون مشخص کردن هیچ شرطی برای آن ایجاد شده است در چنین حالت هایی JRS گره های با کمترین کار سیستمی و مقدار معقولی از منابع را انتخاب می کند.گره n2 باید نامی برابر Rachel داشته باشد. برای گره n3 باید کامپیوتری که تمام شرایط تعریف شده در شی constr در آن صدق می کند انتخاب شود. همانطور که قبلا نیز ذکر شد برای هر گره یک سه تایی یکتای(سایت،خوشه ودامنه)وجود داردکه به ترتیب می توان هر یک از آنها را با متدهای getDomain, getSite, getCluster به دست آورد. همچنین می توان فضای تخصیص یافته برای هر گره را به وسیله متد FreeNode آزاد کرد.

خوشه ها: یک خوشه می تواند با مشخص کردن تعداد گرههای آن ایجاد شود. مجموعه ای از شرایط می تواند برای یک خوشه مشخص شود که در این صورت تمامی گره های آن خوشه حائز این شرایط خواهند بود. در مثال زیر خوشه c1 با 5 گره در خواست شده است. از نماد [...] برای بیان اختیاری بودن عبارت استفاده شده است. یک خوشه را همچنین می توان با اضافه کردن گره های مختلف به آن ساخت به عنوان مثال خوشه c2 با اضافه شدن گره های n3,n2,n1 که اشیایی از نوع Nodeهستند, به آن ساخته شده است.

```
node n1, n2, n3;
//allocate cluster with 5 nodes
Cluster c1 =new Cluster ( 5[, conster]);
// define individual cluster which contains nodes n1, n2, and n3
Cluster c2= new Cluster();
c2.addNode(n1);  c2.addNode(n2);   c2.addNode(n3);
determine current number of nodes in cluster//
c1.nrNodes();
//access node-3 in cluster
Node n3=c1.getNode(3);
// determine site of cluster
Site s1=c1.getSite();
```

```
// determine domain of cluster
Domain d1=c1.getDomain();
//release node n2 from cluster  c2
c2.freeNode(N2);
//release  node 2 from cluster c2
c2.freeNode(2);
//release cluster c2
c2.freeCluster();
```

متدnrNode() می تواند برای مشخص کردن تعداد گره های محاسباتی یک خوشه فراخوانی شود. شماره گره های هر خوشه از0 تا nrNode()-1 شماره گذاری می شوند.متد های getNode و getSite و getDomain برای به دست آوردن گره ها, سایت و دامنه، خوشه می توانند استفاده شوند.گره های یک خوشه را می توان با استفاده از متد freeNode آزاد کرد. خود خوشه را نیز می توان با استفاده از متد freeCluster آزاد کرد.

سایت ها : (Sites) سایت ها را نیز می توان به طریق مشابهی مانند خوشه ها ایجاد کرد. در تکه کد زیر یک سایت با 3 خوشه ایجاد شده است. اگر شرایطی هنگام ایجاد یک سایت مشخص شود, تمام خوشه ها و گره های موجود در سایت دارای این شرایط خواهند بود. همچنین یک سایت را می توان با ایجاد یک نمونه خالی از کلاس Site و اضافه کردن خوشه های موجود به آن با استفاده از متد addCluster ایجاد کرد.

```
int[] SiteNodes = {2,4,5 };
// request for site with 3 clusters with 2, 4
// and 5 nodes, respectively
Site s1 = new Site(SiteNodes [,constr]);
// define individual site which contains cluster c1 and c2
Site s2 = new Site();
s2.addCluster(c1);  s2.addCluster(c2);
// determine current number of clusters and nodes in the site
s1.nrClusters();
s1.nrNodes();
1 in site // access cluster
Cluster c1 = s1.getCluster(1);
1 2 of site s1: alternative 1 in cluster // access node
Node n1 = s1.getCluster(2).getNode(1);
2 2 of site s1: alternative 1 in cluster // access node
Node n1 = s1.getNode(2,1);
// determine domain of site
Domain d1 = s1.getDomain();
1 2 of site s1: alternative 1 from cluster // release node
s1.freeNode(2,1);
2 // release node-1 from cluster-2 of site s1: alternative
s1.getCluster(2).freeNode(1);
// release cluster 1 of site s1
s1.freeCluster(1);
// release cluster c2 of site s1
s1.freeCluster(c2);
// release site s1
s1.freeSite();
```

تعداد گره ها و خوشه های موجود در یک سایت را به ترتیب می توان با استفاده از متد های nrNodes وnrClusters به دست آورد. به هر یک از خوشه های سایت نیز می توان با استفاده از متد getCluster(int clusterID) دست یافت. دو روش برای دستیابی به یک گره از یک سایت وجود دارد. روش اول به دست آوردن خوشه وسپس به دست آوردن گره مورد نظر ازآن خوشه, و روش دوم استفاده از متد getNode(int ClusterID, int NodeID) می باشد. دامنه یک سایت را نیز می توان با استفاده از متد getDomain() به دست آورد. متد های متنوعی نیز برای آزاد سازی هر یک از گره ها, خوشه ها و خود سایت نیز برای هر سایت در دسترس است که در تکه کد فوق نشان داده شده اند.

دامنه ها (Domains) : دامنه ها را نیز می توان همانند خوشه ها و سایت ها به کمک یک آرایه چند بعدی ایجاد کرد. در تکه کد زیر یک دامنه با دو سایت ایجاد شده است که هر یک از سایت ها به ترتیب 3و2 خوشه و هر یک از خوشه ها به ترتیب 1و3و5و6و4 گره دارند.

```
int[][] DomainNodes = {{1,3,5}, {6,4} };
// request for domain with 2 sites
// site-1 with 3 clusters with 1, 3, and 5 nodes, respectively
// site-2 with 2 clusters with 6 and 4 nodes, respectively
Domain d1 = new Domain(DomainNodes [,constr]);
// define individual domain which contains site s1 and s2
Domain d2 = new Domain();
d2.addSite(s1);   d2.addSite(s2);
// determine current number of sites, clusters and nodes
// in the domain
d1.nrSites();
d1.nrClusters();
d1.nrNodes();
// access site-2 in domain
Site s2 = d1.getSite(2);
// access node-3 in cluster-2 of site-1 in domain : alternative-1
Node n1 = d1.getSite(1).getCluster(2).getNode(3);
// access node-3 in cluster-2 of site-1 in domain : alternative-2
Node n1 = s1.getNode(1,2,3);
// release node-3 from cluster-2 of site-1: alternative-1
d1.freeNode(1,2,3);
// release node-3 from cluster-2 of site s1: alternative-2
d1.getSite(1).getCluster(2).freeNode(3);
// release cluster-2 of site-1: alternative-1
d1.freeCluster(1,2);
// release cluster-2 of site-1: alternative-2
d1.getSite(1).freeCluster(2);
// release site-1 of domain d1
d1.freeSite(1);
// release site s1 of domain d1
d1.freeSite(s1);
// release domain d1
d1.freeDomain();
```

برای یک دامنه نیز می توان پیش شرایطی تعریف کرد که در این صورت تمام گره های موجود در سایت ها و خوشه های آن دارای این شرایط خواهند بود. می توان با استفاده از متد addSite(Site s)، سایت های از قبل ایجاد شده را به یک دامنه اضافه کرد. همچنین می توان با استفاده از متد های freeSite، freeCluster و freeNodes یک دامنه ایجاد شده را تغییر داده و سایت ها , خوشه ها و گره های دلخواه را از آن حذف کرد. نحوه استفاده از این متد ها در تکه کد فوق نشان داده شده است. تعداد گره ها, سایت ها و خوشه های موجود در یک دامنه نیز به ترتیب با استفاده از توابع nrNodes()، nrSites() وnrClusters() قابل به دست آوردن است.

2. 3. 3. بار گذاری کلاس ها به صورت پویا :

JavaSymphony به برنامه نویس اجازه می دهد تا اشیا برنامه خود را هم به صورت محلی و هم در یک گره دور ایجاد کند. همانطور که می دانیم قبل از اینکه شئی ایجاد شود باید فایل Class آن در مسیر CLASSPATH جاری ویا در یک URL مشخص قرار داشته باشد. JavaSymphony فرض می کند که هنگام ایجاد یک شی فایل Class آن در همان ماشین در دسترس است. این عمل موجب کاهش حجم داده های تبادلی هنگام ایجاد یک شی در شبکه خواهد شد. برای این منظور JavaSymphony امکان ساخت یک codebase از کد های جاوا را می دهد که پس از ساخته شدن به عنوان یک آرشیو کد های جاوا (مانند فایل های jar) قابل تبادل بین مولفه های مختلف موجود در معماری توزیع شده خواهد بود. به این ترتیب فقط مولفه هایی از معماری مجازی که نیاز به یک فایل class خاص دارند می توانند آن را بارگذاری کرده و استفاده کنند و نیازی به ذخیره کردن همه فایل های Class برنامه، در تمام گره های محاسباتی نخواهد بود.

```
Node node; Cluster cluster; Site site; Domain domain;
// initialize a codebase
JSCodebase codebase = new JSCodebase();
// a Java archive or class file is added to the codebase
codebase.add("../classes.jar");
codebase.add("../testclasses.class");
// Java archive or class file is fetched from URL and added to the codebase
URL classURL = new URL("http;//www.par.univie.ac.at/JS/test/file.class");
codebase.add(classURL);
// load codebase to a node of a virtual architecture
codebase.load(node);
// load codebase to all nodes of a cluster, site, or domain.
codebase.load(cluster);
codebase.load(site);
codebase.load(domain);
// free codebase
codebase.free();
```

2. 3. 4. ایجاد ، استقرار و از بین بردن اشیا

با فرض اینکه فایل های class مورد نیاز در تمام مولفه های معماری مجازی وجود دارد اشیا برنامه توزیع شده را می توان با ایجاد نمونه هایی از شی JSObj ساخت.(JSObj یکی از کلاس های کتابخانه javaSumphony است). اولین پارامتر سازنده این کلاس که در دستور new باید مشخص شود نام کلاسی است که باید شئی از نوع آن در برنامه ایجاد شود. پارامتر دوم اختیاری بوده و نشان دهنده مکان ایجاد شی در سیستم توزیع شده است. برای این پارامتر می توان یک گره، خوشه، سایت، یا دامنه را به عنوان آرگومان در نظر گرفت. در سه حالت اخیر JRS یکی از گره های سایت یا خوشه را که دارای حداقل کار

بارگذاری سیستمی و مقدار معقولی از منابع باشد را انتخاب خواهد کرد. (چنانچه مقداری برای این پارامتر مشخص نشود نیز همین اتفاق خواهد افتاد) برای مشخص کردن ویژگی های الزامی کامپیوتر مقصد می توان مجموعه ای از شرایط، را به کمک یک شی JSConstraints تعریف کرده و به عنوان پارامتر سوم برای سازنده این کلاس فرستاد. یکی از متد های مفید دیگر که توسط JavaSymphony ارائه شده است تابع getLocalNode() می باشد که برای به دست آوردن گرهی که برنامه جاری در آن در حال اجرا است استفاده می شود.

```
// get node on which this application is being executed
Node local = JS.getLocalNode(); JSConstraints constr;
Node node; Cluster cluster; Site site; Domain domain;
// generate an object of class “class name” at
// a node decided by JRS or restricted to constraints
JSObj obj1 = new JSObj(“class name” [, constr]);
// generate object on the local node
JSObj obj1 = new JSOBj(“class name”,local);
// generate object on a specific node
JSObj obj1 = new JSOBj(“class name”,node);
// generate object on an arbitrary node of a cluster, site,
// or domain decided by JRS or restricted to contraints
JSObj obj1 = new JSOBj(“class name” [,cluster jsite jdomain,constr]);
// generate obj1 on the same node
// where obj2 has been generated
JSObj obj1 = new JSOBj(“class name”,obj2.getNode());
// generate obj1 on the same cluster, site,
// or domain where obj2 has been generated
JSObj obj1 = new JSOBj(“class name”,obj2.getCluster() [,constr]);
JSObj obj1 = new JSOBj(“class name”,obj2.getSite() [,constr]);
JSObj obj1 = new JSOBj(“class name”,obj2.getDomain() [,constr]);
// free object
obj1.free();
```

همانطور که در کد های فوق نشان داده شده است برنامه نویس می تواند شی جدید را در گره, خوشه, سایت یا دامنه ای که شی دیگری در آن در حال اجرا است، با استفاده از آن شی, ایجاد کند. اشیائی که دیگر نیازی به آنها نباشد باید توسط متد free آزاد شوند.

2. 3. 5. فراخوانی متدها

تکنیک RMI جاوا, فراخوانی متد های دور به صورت blocking را امکان پذیر می کند. در این روش از انجام محاسبات محلی هنگام انتظار برای دریافت نتیجه فراخوانی متد جلوگیری شده و فراخواننده متددور تا هنگام دریافت نتیجه آن بلوکه میشود. JavaSymphony علاوه بر روش فوق دو روش دیگر برای فراخوانی متد های دور را در اختیار می گذارد که عبارتند از فراخوانی آسنکرون (non-blocking) و فراخوانی یک طرفه.

فراخوانی سنکرون متد ها: با فراخوانی سنکرون متدی طرف فراخوانی کننده تا هنگام رسیدن نتیجه فراخوانی بلوکه خواهد شد. فراخوانی سنکرون متد ها با استفاده از تابع sinvoke صورت می گیرد. پارامتر ها به صورت آرایه ای از Object هابه این تابع ارسال شده و نتیجه نیز همیشه به صورت شئی از نوع Object برگشت داده می شود. که باید به نوع داده مورد نظر نگاشت

شود. در تکه کد زیر متدی به نام methodname با دو پارامتر param1() وparam2() از شی obj به روش سنکرون فراخوانی می شود.

```
JSObj obj = new JSOBj("class name");
…
Object[] params = {new Param1(), new Param2()};
ResultClass result = (ResultClass)obj.sinvoke("method name",params);
…
```

فراخوانی آسنکرون : با فراخوانی آسنکرون متد ها می توان نوعی موازی سازی در انجام محاسبات در گره های محاسباتی تشکیل دهنده سیستم توزیع شده پیاده سازی کرد. فراخوانی آسنکرون توسط تابع ainvoke صورت گرفته و پارامتر ها به صورت آرایه ای از Object هابه این تابع ارسال می شوند. فراخوانی متد دور به این روش موجب بلوکه شدن طرف فراخواننده نمی شود بلکه بلافاصله یک handle به عنوان نتیجه برگشت داده شده و اجرای بقیه دستورات برنامه ادامه می یابد. handle برگشت داده شده دارای متدی به نام isReady() است که در صورت آماده بودن نتیجه فراخوانی Trueو در غیر اینصورت False برگشت می دهد. اگر طرف فراخواننده بنا به هر دلیل بخواهد تا زمان رسیدن نتیجه بلوکه شود (مثلا دستور محاسباتی دیگری برای اجرا نداشته باشد) می تواند تابع handle.getResult() را فراخوانی کند. این تابع نیز شئی از نوع Objectرا به عنوان نتیجه بر گشت می دهد,که باید به نوع مورد نظر نگاشت شود.

```
// invoke remote method with parameters;  a handle is returned
// to refer to the method's result in the future
Object[] params =  {new Param1(), new Param2()};
ResultHandle handle = obj.ainvoke("method name",params);
...
// verify whether result is available
if (handle.isReady()) {
// wait for result to arrive in blocking mode
ResultClass result = (ResultClass)handle.getResult();
}
...
// wait for result to arrive in blocking mode, without checking for available result
ResultClass result = (ResultClass)handle.getResult();
```

فراخوانی یک طرفه : فراخوانی یک طرفه متد ها زمانی انجام می گیرد که طرف فراخوانی کننده متد نیازی به دریافت نتیجه از شی اجرا کننده یا اطمینان از اجرای کامل متددور، نداشته باشد. فراخوانی یک طرفه متد به وسیله تابع oinvoke صورت گرفته وپارامتر ها همانند دو روش قبل به صورت آرایه ای از Objectها به این تابع ارسال می شود. این نوع فراخوانی می تواند موجب افزایش کارایی سیستم توزیع شده شود زیرا در این روش نیازی به ارسال نتیجه از گره مقصد به گره مبدا نخواهد بود.

```
Object[] params = {new Param1(), new Param2()};
obj.oinvoke("method",params);
```

2. 3. 6. حرکت پویای اشیا (dynamicly migrate object)

اشیا می توانند حین اجرای یک برنامه از یک گره به گره دیگر حرکت کنند. البته قبل از حرکت باید اجرای تمام متد های آنها در گره جاری به پایان برسد. اگر متدی از شی در حال اجرا باشد مهاجرت آن شی به یک گره دیگر تا زمان اتمام اجرای متد آن به تاخیر خواهد افتاد.JavaSymhony دو نوع حرکت برای اشیا را پشتیبانی می کند.

1- حرکت اتوماتیک و 2- حرکت, تحت کنترل برنامه نویس

حرکت اتوماتیک اشیا به وسیله JS-Shell انجام می شود. در این حالت JRS به طور متناوب بررسی می کند که آیا گره حاوی یک شی همچنان دارای شرایط تعیین شده از طرف کاربر برای نگهداری آن شی است یا خیر. (اگر شرایطی از طرف برنامه نویس تعیین نشده باشد, شرایط تعیین شده از طرف JS_Shell بررسی خواهد شد). اگر گره مذکور حاوی این شرایط نباشد شی مستقر در آن گره به گره دیگری منتقل خواهد شد.

حرکت صریح اشیا نیز توسط برنامه نویس امکانپذیر است. برای این منظور JavaSymphony به برنامه نویس اجازه میدهد تا به پارمتر های سیستمی سایت ها, خوشه و گره ها دسترسی داشته باشد. تابع getSysParam می تواند برای به دست آوردن خصوصیات هر یک از مولفه های معماری مجازی فراخوانی شود. همچنین با استفاده از تابع constrHold میتوان بررسی کرد که آیا گرهی حائز مجموعه ای از شرایط است یا خیر. این دو تابع دارای پارامتر های مشابهی با توابع درخواست مولفه های معماری مجازی (مطرح شده در بخش 2. 3) می باشند. در تکه کد زیر بررسی می شود که یک گره دارای متوسط زمان بیکاری 50 درصد است یا خیر. اگر نباشد شی مستقر شده در آن با استفاده از تابع migrate به یک گره دیگر مهاجرت خواهد کرد. اگر تابع migrate بدون پارامتر ذکر شود JRS تصمیم می گیرد که این شی به کجا منتقل شود.

```
JSConstraints constr;
Node node;  Cluster cluster;  Site site; Domain domain;
JSObj obj;
Node n1 = object.getNode();
// node on which object resides has less than 50 % idle time
// set of constraints constr hold for node n1
if (n1.getSysParam(JSConstants.IDLE) < 50) || n1.constrHold(constr)) {
// migrate object to a node destined by JRS
obj.migrate();
// migrate object to a node according to a set of constraints
obj.migrate(constr);
// migrate object to a specific node
obj.migrate(node);
// migrate object to a node of a cluster, site, or domain
// to be destined by JRS or optionally based on constraints
obj.migrate(cluster | site | domain [,constr]);
```

تابع migrate می تواند با استفاده از پارامتر های زیرنیز فراخوانی شود

پارامتری از نوع Node: که گرهی که باید شی به آنجا منتقل شود را نشان می دهد.

پارامتری از نوع Cluster یا Site یا Domain: که در این حالت JRS تصمیم می گیرد که کدام یک از گره های دامنه، سایت، یا خوشه باید میزبان شی باشد. اگر به همراه یکی از این سه نوع پارامتر, پارامتر دیگری از نوع Constraints به تابع migrate ارسال شود JRS در دامنه یا سایت یا خوشه مشخص شده به دنبال گرهی که حائز شرایط ذکر شده در پارامتر دوم باشد, خواهد گشت.

پارامتری از نوع Constraints : در این حال JRS به دنبال گرهی در معماری مجازی که حائز شرایط ذکر شده در پارامتر باشد گشته و شی را به آن گره منتقل خواهد کرد.

2. 3. 7. اشیا ما نا (persistent objets)

JavaSymphony ذخیره سازی اشیا موجود در سیستم توزیع شده را در حافظه های جانبی, امکانپذیر می کند یک شی زمانی می تواند در حافظه جانبی ذخیره شود که هیچ یک از متد های آن در حال اجرا نباشد. یک رشته یکتا توسط JRS برای شی ذخیره شده در حافظه جانبی برگشت داده می شود که از آن می توان برای بارگذاری مجدد شی به یکی از گره ها استفاده کرد.

```
JSObj obj; String str;
// save object on external storage
str = obj.store();
...
// load object from external storage
JSObj obj = new JSObj(str);
```

اشیا می توانند با استفاده از فراخوانی یکی از سازنده های کلاس JSObj که یک رشته را به عنوان آرگومان دریافت می کند، از حافظه جانبی بارگذاری شوند.

3. مفاهیم پیشرفته تر JavaSymphony برای ساخت سیستم های توزیع شده

در این بخش برخی مفاهیم پیشرفته تر JavaSymphony که برای ساخت راحت تر سیستم های توزیع شده ارائه شده اند بررسی می شود.

3. 1. افزایش کارایی Java Symphony

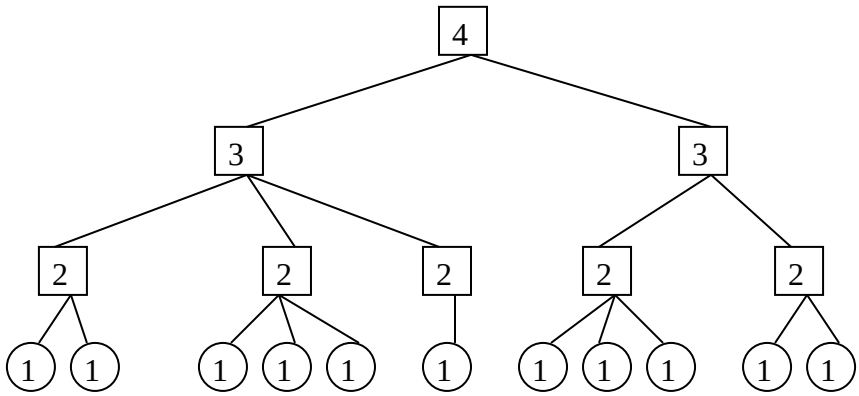
استفاده از زبان برنامه نویسی جاوا برای ساخت برنامه هایی با کارایی بالا یکی از بحث های رایج علوم کامپیوتر است. از یک طرف جاوا امکاناتی از قبیل انتقال کد، شی گرایی، قابلیت حمل، اجرای چند نخی، همزمان سازی وAPI های مختلفی برای برقراری ارتباطات ارائه می کند که همه این موارد برای ساخت یک سیستم توزیع شده یا موازی بسیار مفید هستند. اما از طرف دیگر تقریبا تمام برنامه های نوشته شده توسط جاوا مفسری بوده و باعث پایین آمدن کارایی برنامه در زمان اجرا می شود. در سال های اخیر تلاش های زیادی برای بهبود کارایی برنامه های جاوا شده است. بهینه سازی هایی در ماشین مجازی جاوا صورت گرفته است تا برنامه ها را با سرعت بالاتری اجرا کند. تحقیقات اخیر نشان می دهد که کارایی کد های بهینه شده جاوا قابل مقایسه با کارایی برنامه های Cیا فرترن است.

پروژه های تحقیقاتی متعددی نیز, کتابخانه ها یی از Classها و یا بسط هایی در زبان جاوا ایجاد کرده اند که از مسائلی مانند Locality، موازی سازی و بارگذاری متعادل پشتیبانی می کنند اما اکثر این روش ها تحت کنترل سیستم زمان اجرا بوده و از لحاظ کارایی با مشکل مواجه هستند. در بخش قبل ابزاری به نام JavaSymphony معرفی شده و توانایی های آن جهت ساخت سیستم های توزیع شده بررسی شد.

در این بخش بسط های که درJavaSymphony داده شده است بررسی می شود این بسط ها شامل توسعه معماری مجازی به منظور ایجاد توانایی برای پیاده سازی معماری های مختلف توزیع شده و موازی، افزایش و کنترل تعداد نخ هایی که متدی از یک شی را اجرا می کنند, تبدیل اشیا معمولی جاوا به صورت پویا به اشیا JavaSymphony, و همگام سازی ارتباطات و پردازش ها می شود. تکنیک های ارائه شده برای همگام سازی، BarrierSynchronization و همگام سازی در فراخوانی آسنکرون متد ها است.

3. 2. شکل جدیدی از Dynamic Virtual Disteributed Architecturs

همانطور که قبلا بحث شد, JavaSymphony مفهومی به نام معماری توزیع شده مجازی ارائه می کندکه در ادامه آن را VAمی نامیم. این معماری به برنامه نویس امکان تعریف ساختار یک شبکه همگن از منابع محاسباتی را می دهد. همچنین به کمک این معماری می توان نحوه مهاجرت اشیا، بارگذاری متعادل و مکان یابی کدها را کنترل کرد. VAدامنه وسیعی از سیستم های همگن را پوشش می دهد. در این بخش علاوه بر ساختار قبلی, شکل جدیدی از معماری مجازی تعریف می شود، که توسط آن می توان معماری های متنوع تری برای منابع محاسباتی یک سیستم توزیع شده طراحی کرد. (شکل 2) دراین حالتVAشامل مجموعه ای از مولفه هاست که هر کدام متناظر یک سطح در کل معماری مجازی هستند.



شکل 2 . معماری مجازی

سطح یک معادل با یک گره محاسباتی مانند یک PCیا WorkSattionیا یک سیستم چند پردازنده ای است. سطح 2 از معماری VAنشان دهنده خوشه ای ازگره های محاسباتی (سطح 1) است. سطح i که در آن i>=2 است نشان دهنده خوشه ای از سطوح i-1 است که دارای یک معماری توزیع شده دلخواه GRID است.

JavaSymphony امکان ایجاد و تغییر VAها به صورت پویا و قابل انعطاف را می دهد. VAها می توانند به صورت Top-Down در یک خط کد, (شکل 1 را ببنید)و یا به صورت Bottom-Up، با ترکیب سطوح پایینی در یک سطح بالای ایجاد شوند. در تکه کد زیر این روش ها نشان داده شده است :

```
JSConstraints constr;
// request level 1 VA
VA v1 = new VA( 1) ;
// request level 1 VA for which constraints hold
VA v2 = new VA( 1, constr ) ;
// bottom-up request for level 2 VA
// by adding existing level-1 VAs to it
VA v3 = new VA( 2) ;
v3.addVA(v1);
v3.addVA(v2);
// top-down request for level 4 VA (Fig.1) with 2 level 3 VA's:
// first level 3 VA with 3 level 2 VAs with 2, 3,
// and 1 level 1 VAs, respectively
```

// second level 3 VA with 2 level 2 VAs with 3
// and 2 level 1 VAs, respectively
VA v4 = new VA(4, new int[][] {{2,3,1 }, { 3,2 }});

JRS برای هر VA ایجاد شده یک Handle برمیگرداند. اینHandle ها درحقیت اشیا موجود در اولین مرتبه VAهستند که می توانند به صورت پارامتر به دیگر متد ها ارسال شوند. هر نخی که دارای یک Handle به یک VAباشد می تواند به آن دسترسی داشته, آن را تغییر داده و حتی حافظه آن را آزاد کند. می توان از تغییر همزمان یک VAتوسط چند نخ به کمک مکانیزم قفل ارائه شده توسط JS جلو گیری کرد. هدف از این ایده در JS, گرد آوریی مولفه های محاسباتی با معماری های مختلف در یک VAمی باشد که هر کدام از این مولفه ها با استفاده از پارامتر های Satatic وDynamic طبق خواسته ها و دستورات کاربر فعالیت می کنند. از طرف دیگر شرط های تعیین شده از طرف کاربر، می توانند برای بررسی خصوصیات منابع فیزیکی موجود در سیستم استفاده شوند.

3. 3. تبدیل اشیا جاوا به به اشیا JavaSymphony به صورت پویا

همانطور که قبلا ذکر شد، برای استفاده ازJavaSymphony به منظور توزیع اشیا در معماری مجازی لازم است که ابتدا هر کدام از اشیا برنامه دریک شی JS کپسوله شوند. علاوه براینکه می توان از کلاس JSObj برای ساخت اشیا JavaSamphony استفاده کرد، اشیا JS را می توان با ایجاد نمونه هایی از کلاس JSobject نیز ساخت. این کلاس نیز جزئی از کتابخانه کلاس های JavaSymphony بوده و دارای سازنده هایی است که امکان مشخص سازی نام کلاس اصلی (کلاسی که باید در شی JSکپسوله شود)، آرگومان های سازنده این کلاس، تک نخی یا چند نخی بودن شی JS و مکان شی JS به همراه شروط توصیف کننده خصوصیات آن مکان، را میسر می کنند. می توان با مشخص کردن سطح یک مناسبی در VA مکان مناسبی برای شی JS تدارک دید. چنانچه یکی ازسطوح بالای موجود در VAمثلا v(که شماره سطح آن بزرگتر یا مساوی 2 است)به عنوان مکان شی JS مشخص شود،JRS تلاش می کند خود، سطح یک مناسبی که تمام شروط ذکر شده در آن صدق می کند از لایه های پایینی Vپیدا کند. چنانچه فقط شروط مکان مورد نظر ذکر شده و خود مکان مشخص نشود آنگاه JRS از کل VAبه دنبال مکانی که تمام شروط در آن صدق می کند خواهد گشت. اگر هیچ یک از پارامتر های مکان و شروط مکان مشخص نشده باشد, JRS از محلی پیش فرض بر مبنای شروط پیکر بندی استفاده خواهد کرد. شروط پیکر بندی در JS_Shellتنظیم می شود.

VA v1 = new VA(1); // allocate level-1 VA
VA v2 = new VA(4,....); // allocate level-4 VA
VA vLocal = VA.getLocalNode(); // get local level-1 VA
JSConstraints constr;
// parameters for the new object
] args = new Object[ ] {... } ; Object[
// create object obj1 of class "ClassName" at a VA decided by
// the JRS, restricted to constraints or at the local level-1 VA
JSObject obj1 = new JSObject("ClassName"[, args] [, constr] [, vLocal]);
// create object obj1 on a higher level VA; JRS decides
// on which level-1 VA of v2, the obj1 will be generated
JSObject obj1 = new JSObject("ClassName",[args,] v2);
// create object on a specific VA v1
JSObject obj1 = new JSObject("ClassName",[args,] v1);
// create obj1 on the same level-1 VA

// where obj2 has been created
JSObject obj1 = new JSObject("ClassName",obj2.getVA());

اشیا JSمی توانند پس از ایجاد نیز، اشیا معمولی موجود در برنامه جاوا را کپسوله کنند. این عمل می تواند توسط متد ConvertToJSObject انجام شود. اولین پارامتر این متد مشخص کننده شی معمولی جاوا و دومین پارامتر مشخص کننده نوع شی ایجاد شده از لحاظ تک نخی یا چند نخی بودن آن است. با استفاده از این روش اشیا معمولی جاوا می توانند با فراخوانی متد هایشان از راه دور دستیابی شوند. مهاجرت یک شی JS مانند obj2که از یک شی معمولی جاوامانند obj1 ساخته شده است نیز امکان پذیر است.

// convert a non-JSObject obj to a JS object obj2
ClassName obj = new ClassName(...);
JSObject obj2 = JSObject.convertToJSObject(obj [,multiThreaded]);

در ادامه برخی از خصوصیات اشیا JSتوضیح داده می شود :

امکان قفل بر روی اشیا JS : اشیا JS از طریق Handle هایی دستیابی می شوند که همان اشیا موجود در اولین مرتبه VAهستند. آنها می توانند به متد های دیگر ارسال شده و در نتیجه در کل VAتوزیع شوند. هر نخی با یک Handle به شی JS می تواند به این شی دسترسی داشته و متد های آن را فراخوانی کند به منظور فراهم کردن روشی برای سازگاری در هنگام تغییرات و دسترسی انحصاری به اشیا JS مکانیزمی به نام قفل در JS ارائه شده است. اگر نخ t دارای Handle ای به یک شی JS بوده و آن را قفل کند آنگاه هیچ نخ دیگری تا زمانی که t قفل خود را از آن شی بردارد به آن شی دسترسی نخواهد داشت. عمل قفل کردن یک شی تا زمان اتمام اجرای متد های فراخوانی شده از آن شی به تعویق می افتد.

فراخوانی متد ها : JAVA/RMI فراخوانی روال دور به صورت Blocking را ارائه می کند. این تکنیک از انجام محاسبات محلی در هنگام انتظار برای دریافت نتیجه فراخوانی یک متد دور جلوگیری می کند(sinvoke). علاوه بر فراخوانی متد های دور به صورت سنکرون(با استفاده از RMI) JavaSymphony از فراخوانی متد های دور به صورت آسنکرون (NonBlocking) یا (ainvoke) و RMI یک طرفه(که در آن نتیجه ای برگشت داده نمی شود) نیز پشتیبانی می کند(oinvoke) هر سه نوع فراخوانی متد های دور دارای امضای یکسان هستند: نام متد به همراه لیست پارامتر ها و احتمالا نوع پارامتر ها مشخص می شود. اما در برگشت نتیجه فراخوانی متفاوت عمل می کنند, در فراخوانی سنکرون شئی که حاوی نتیجه اجرای متد است برگشت داده می شود. در فراخوانی آسنکرون یک resultHandle به عنوان نتیجه برگشت داده می شود که می توان نتیجه فراخوانی متد را بعدا از طریق این resultHandle به دست آورد. در فراخوانیRMI یک طرفه نیز نتیجه ای برگشت داده نمی شود.

مهاجرت اشیا : اشیا می توانند در حین اجرای برنامه حرکت کنند البته قبل از حرکت یک شی JRS بررسی می کند که آیا متدی از آن شی در حال اجرا است یا خیر ؟, اگر بود مهاجرت شی تا هنگام اتمام اجرای متد در حال اجرای آن به تعویق می افتد.در غیر این صورت شی مورد نظر به سرعت منتقل می شود.JavaSymphony دو نوع مهاجرت مختلف برای اشیا ارائه کرده است : 1-مهاجرت اتوماتیک که توسط JRSکنترل می شود. 2- مهاجرت صریح که توسط خود برنامه نویس کنترل می شود. علاوه بر این برنامه نویس می تواند مقصد مهاجرت شی در VA، شرط های لازم برای مقصدو چگونگی ترجمه کد ها در مقصد را مشخص کند.

3. 4. اشیا تک نخی و چند نخی

هر شی JSمی تواند در دو حالت تک نخی و چند نخی ایجاد شود.اشیا تک نخی فقط دارای یک نخ اجرایی بوده و تمام متدهای این شی فقط توسط همین نخ اجرا می شوند. این خصوصیت باعث می شود تا یک ترتیب متوالی در فراخوانی تمام متد های دور

این شی اعمال شود. در مقابل اشیا چند نخی می توانند دارای چندین نخ اجرایی باشند که JRS همه آنها را به صورت موازی و همزمان اجرا می کند. حتی یک متد از یک شی چند نخی می تواند توسط نخ های مختلف به صورت موازی اجرا شود

```
// generate a multi-threaded object in a node of
// a higher level VA v that honors a set of constraints
boolean multiThreaded = true;
JSObject obj1 = new JSObject(multiThreaded,"ClassName" [,args] [,constr][,v]);
// objects can be made single- or multi-threaded at runtime
obj1.singleThreaded();
obj1.multiThreaded();
```

3. 5. همگام سازی فراخوانی آسنکرون متدها

در سیستم های توزیع شده یا موازی معمولا برنامه نویسان اجرای نخ ها و پروسه ها را هماهنگ می کنند در فراخوانی آسنکرون متد ها به حالاتی بر می خوریم که در آن مجموعه ای از نخ ها که متد هایی را به صورت موازی (و احتمالا بر روی کامپیوتر های مختلف) اجرا می کنند باید در یک نقطه مشخص هم گام شوند. برای این منظور JavaSymphony برنامه نویس را قادر می سازد تا مجموعه ای از resultHandleها را(که هر کدام مربوط به فراخوانی آسنکرون یک روال دور هستند) گروه بندی کند. این عمل توسط کلاس ResultHandleGroup انجام می شود. این کلاس دارای متد های مختلفی است که برای بلوکه کردن یا بررسی وضعیت تعداد معینی از نخ ها و یا تمام آنها (از لحاظ اینکه آیا اجرای متد های خود را به اتمام رسانده اند یا نه)استفاده می شود. لازم به ذکر است فراخوانی هیچ کدام از متد های این کلاس موجب بلوکه شدن برنامه نخواهد شد.

بعلاوه می دانیم برای پیاده سازی بارگذاری متعادل متد های اشیا گوناگون (که در کامپیوتر های مختلف مستقر هستند)باید به صورت موازی اجرا شوند. با استفاده از مکانیزم همگام سازی به راحتی می توانیم تشخیص دهیم که کدام یک از اشیا اجرای متد هایش را تمام کرده و فعلا بیکار است. در تکه کد زیر نحوه همگام سازی فراخوانی متد های دور نشان داده شده است :

```
JSObject obj[n];
ResultHandleSet rhs;
ResultHandle rh;
] params; Object[
...
for(i=0; i < n; i++) {
ResultHandle Set rhs // add a ResultHandle and index i (optional) to the
].ainvoke("run", params), i); rhs.add(obj[ i
}
...
//non-blocking test if at least 5 methods are finished
if( rhs.isReady(5) ) {... }
 //non-blocking test if all methods are finished
if( rhs.isAllReady() ) {... }
//block until at least 5 methods are finished
if( rhs.waitReady(5) ) {... }
 // block until all methods are finished
if( rhs.waitAll () ) {... }
// block until the first method has returned results
rh = rhs.getFirstReady();
```

```
// get results one by one without specific order;
while(rh ! = null) { // get the results
ResultClass result = (ResultClass)rh.getResult();
... // process results
// get index of idle object
index = rhs.getIndex(rh);
// invoke next method on idle object for load balancing
// and add ResultHandle in ResultHandleSet again
rhs.add( obj[index].ainvoke("run", params), index);
// get ResultHandle of a method that finishes next;
// block until results returned
rh = rhs.getNextReady(); }
```

3. 6. Barrier Synchronization(همگام سازی متوقف کننده)

JavaSymphony می تواند برای متد های مختلف فراخوانی شده از اشیا JS یک نقطه توقف مشخص کند به این صورت که جریان اجرای مجموعه ای از نخ ها که به این نقطه رسیده اند را تا زمانی که تمام نخ های موجود در مجموعه به این نقطه برسند متوقف کند. بعد از اینکه تمام نخ ها به نقطه توقف رسیدند دو باره می توانند اجرای خود را به صورت موازی با شروع از بعد از نقطه توقف از سر گیرند.این مکانیزم بسیار ساده است اما استفاده ناصحیح از آن می تواند موجب کاهش کارایی, یا حتی بن بست شود. نقاط توقف یا Barrierها را می توان با استفاده از متد newBarrier دربرنامه قرار داد. newBarrier قسمتی از کلاس JSRegistery می باشد.برای هر نقطه توقف باید یک شناسه یکتاو تعداد نخ هایی که باید در این نقطه منتظر بمانند مشخص شود.یک Barrier از طرف تمام اشیا برنامه قابل رویت است. اجرای نخی که به یک Barrier رسیده باشد، تا زمان رسیدن nنخ دیگر به این نقطه متوقف خواهد شد. پس از رسیدن اجرای تمام نخ ها به این نقطه, اجرای همگی آنها با شروع از بعد از نقطه توقف از سر گرفته می شود. تکه کد زیر نحوه استفاده از نقاط توقف برای هم گام سازی را نشان می دهد :

```
// a barrierId defines a unique synchronization point
int barrierId = 17;
// define a barrier for two (remote) threads.
JSRegistry.newBarrier(2, barrierId);
obj1.oinvoke("runThread1", params);
obj2.oinvoke("runThread2", params);
...
// inside runThread1
int barrierId = 17;
// suspend execution until runThread2
// reaches the synchronization point
JSRegistry.barrier(barrierId);
...
// inside runThread2
int barrierId = 17;
// suspend execution until runThread1
// reaches the synchronization point
JSRegistry.barrier(barrierId);
...
```

3. 7. رویداد های JavaSymphony (JavaSymphony Events)

برنامه هایی که عکس العمل های اشیا مختلف را در مقابل رویداد های اتفاق افتاده در همان مکان یا در Nodeهای محاسباتی دیگر ثبت و پردازش می کنند، هم برای سیستم های توزیع شده و هم برای سیستم های موازی لازم هستند. در حالت کلی یک عمل از طرف کاربر یا سیستم به عنوان رویدادی برای دیگر اشیا برنامه محسوب می شود. همچنین استفاده از رویداد ها روشی برای برقراری ارتباطات آسنکرون می باشد. جاوا دارای چند مدل برای پردازش رویداد ها است. همه این مدل ها رویدادهایی رابرای پاسخ به برخی تغییرات اتفاق افتاده در سیستم (چه در داخل شی و چه در خارج از آن و در مکان های دور) تولید می کنند.در اکثر حالات یک مصرف کننده رویداد, با رویداد خاصی متناظر شده و در هنگام وقوع آن رویداد، فراخوانی هایی به متد های مشخص از مصرف کننده انجام می شود.

JS نیز از این مدل کلی پیروی می کند که در آن اشیا می توانند به عنوان مصرف کننده برای یک رویداد ثبت شوند. رویداد هایی با انواع مختلف قابل ایجاد بوده و اشیائی که به عنوان مصرف کننده این رویداد ثبت شده اند از وقوع این رویدادها با خبر می شوند.مصرف کننده یک رویداد با فراخوانی متد های مناسب , به رویداد اتفاق افتاده پاسخ می دهد.یکی از مزیت های JSدر مورد مکانیزم رویداد ها این است که هیچ محدودیتی برای نوع اشیائی که می توانند رویداد ها را تولید یا آنها را مصرف کنند وجود ندارد. هر شی جاوا که توسط JavaSymphony توزیع شده باشد می تواند بدون پیاده سازی کردن رابط خاص و یا ارث بردن از کلاس خاصی به تولید یا مصرف رویداد ها بپردازد. JavaSymphony سه نوع از رویداد ها را پشتیبانی می کند :

رویداد های تعریف شده از طرف کاربر(user defined events) : این نوع رویداد ها صریحا از طرف کاربر تولید میشوند. از این نوع رویداد ها برای پشتیبانی کردن از ارتباطات آسنکرون و تعامل بین اشیا مختلف جاوا استفاده میشود.(اشیائی که به اشیا JSتبدیل نشده اند) برنامه نویس باید کد های لازم برای تولید کننده رویدادو نیز متد هایی که باید توسط مصرف کننده در هنگام وقوع رویداد فراخوانی شوند را مشخص کند.

رویداد های میان افزار (middleware events) : این رویداد ها توسط JRS در مواردی از قبیل در دسترس نبودن VA، ثبت شدن یا از ثبت خارج شدن یک برنامه درJRS, قفل شدن اشیا یا VAها و... تولید می شوند. در این حالت کاربر فقط وظیفه مشخص کردن متدی که در هنگام رخداد این نوع رویداد باید اجرا شود را بر عهده دارد تولید این رویداد ها برعهده JRS است.

رویداد های سیستمی (System events) : این نوع رویداد ها زمانی اتفاق می افتند که برخی از پارامتر های سیستم مانند زمان بیکاری, میزان حافظه در دسترس و... به صورت پویا تغییر کنند. تمامی این پارامتر ها از طریق JS API های مربوط به پارامتر های staticوdynamic در دسترس کاربر قرار دارند.(بخش اول) برای یک شی مصرف کننده رویداد ها، مجموعه ای از شروط, یک ثابت که تولید رویداد را کنترل می کندو متدی که در هنگام رخداد این رویداد باید اجرا شود به صورت پارامتر های سازنده مشخص می شوند. JRS به طور متناوب منابع موجود در سیستم را بررسی کرده و مشخص می کند که چه نوع رویدادی باید تولید شود.

هر شئی که می خواهد مصرف کننده ای برای یک رویداد تعریف شده از طرف کاربر یا یک رویداد میان افزار باشد شئی از نوع JSEventConsumer ایجاد می کند و به وسیله آن نوع (کاربر یا میان افزار) و خصوصیات رویداد مورد نظر خود را مشخص می سازد. برای رویداد های سیستمی می توان از کلاسی به نام SystemEventConsumer استفاده کرد. هنگام ساختن شئی از کلاس JSEventConsumer, برنامه نویس اطلاعات زیر را مشخص می کند : یک ارجاع به شی ای که رویداد را مصرف خواهد کرد, یک شناسه یکتا برای نوع رویداد مورد نظر, و متدی که در هنگام رخداد رویداد باید فراخوانی شود. رویدادها می توانند توسط پارامترهای ویژه ای که به سازنده این کلاس ارسال می شود فیلتر شوند. به عنوان مثال می توان رویداد ها را بر اساس تولید کننده ها ی خاص و یا تولید کننده هایی که در مکان خاصی قرار دارند محدودکرد. مجموعه ای از ثابت ها برای انواع رویداد ها در کلاس JSConstans تعریف شده است برخی از آنها عبارت اند از : C_USER_TYPE نشان دهنده یک

رویداد تعریف شده از طرف کاربر است. C_APP_REGISTERED نشان دهنده رویدادی است که هنگام ثبت شدن یک برنامه در JRS تولید می شود. C_SYSTEM_EVENT معادل یک رویداد سیستمی است.ثابت C_ANY_LOCATION نشان دهنده این است,که رویداد ها ی اتفاق افتاده از طرف تولید کننده هایی در هر مکان از VA توسط مصرف کننده قبول شود.C_LIST_EVENT رویداد های دریافت شده را به لیست مشخصی از VAها محدود می کند. C_LIST_JSOBJECT_EVENT رویداد های دریافت شده را به لیست مشخصی از JSObject ها محدود می کند.

برای یکJSSystemEventConsumer یک شی JSConstraints شرط های لازم برای دریافت رویداد ها را کپسوله می کند. پارامتر های اضافی دیگری نیز تولید یک رویداد را کنترل می کنند مثلا اگر شرط ها معتبر باشند (JS_CONSTRAINTS_HOLD)یا اگر نامعتبر باشند (JS_COSTRAINT_NOT_HOLD)یا اگر وضعیت آنها تغییر کند (JS_CONSTRAINTS_CHANGE). یک مصرف کننده با فراخوانی متد subscrib ازکلاس JSConsumerEvent خود را به عنوان مصرف کننده یک رویداد ثبت می کند. اگر مصرف کننده ای دیگر نیاز به دریافت پیام هنگام وقوع رویدادی نداشته باشد می تواند از متد unsubscribe استفاده کند. تنها رویداد های تعریف شده توسط کاربر می توانند به صورت صریح از طریق شی JSEventProducer تولید شوند. اولین پارامتر سازنده این کلاس, شی تولید کننده رویداد را مشخص می کند, دومین پارامتر نیز مرجعی به شناسه یکتای رویداد تولید شده است که باید با پارامتر دوم JSEventConsumer مطابقت داشته باشد.علاوه بر این لیست یکسانی از ثابت های تعریف شده در JSConstans میتواند برای محدود کردن مصرف کننده گان یک رویداد استفاده می شود. رویداد هایی که از طرف کاربر تعریف میشوند باید به طور صریح و با فراخوانی متد ProduceEvent از کلاس JSEventProducer تولید شوند. پارامتر های ارسال شده به این متد توسط JRSبه متد مصرف کننده ها ارسال خواهد شد. تکه کد زیر نحوه استفاده از رویداد های JSرا نشان می دهد.

```
// ****** Code for Event Consumer ********
...
// define types for user defined and middleware events
int userEvType = JSConstants.CUSERTYPE + 1;
int middleEvType = JSConstants.CAPPREGISTERED;
]=.....; // list of remotes objects JSObject listObj[
]=.....; // list of VAs VA listVAs[
JSConstraints constr1;
// subscribe for a user defined event; no restriction on
// event producers; handleMethod will handle events.
JSEventConsumer(this,userEvType, JSEventConsumer cEv1 = new
JSConstants.CANYLOCATION,"handleMethod");
// events can be produced only on VAs in listVAs
userEvType,  JSEventConsumer cEv2 = new JSEventConsumer(this,
JSConstants.CLISTVAEVENT,listVAs, "handleMethod");
// event can be produced only by JSObjects in listObj
JSEventConsumer cEv3 = new JSEventConsumer(this,userEvType,
JSConstants.CLISTJSOBJECT EVENT,listObj, "handleMethod");
// subscribe for a middleware event which can be produced
// anywhere;the event is generated when a new application registers with JS
JSEventConsumer cEv4 = new JSEventConsumer(this,
middleEvType,JSConstants.CANYLOCATION,"handleMethod");
// subscribe for a system event which can be produced
```

```
// only by the VA va when the validity for a set of constraints constr changes
JSSystemEventConsumer cEvSystem =new JSSystemEventConsumer(this,
JSConstants.C VA EVENT, va, ”handleMethod”,constr,
JSConstants.JSCONSTRAINTSCHANGE);
...
// subscribe for event cEv1
cEv1.subscribe()
...
// unsubscribe for event cEv1
cEv1.unsubscribe()
...
// ****** Code for Producer of User-Defined Events ********
...
int userEvType = JSConstants.CUSERTYPE + 1;
Object listObj[]=.....; // list of remotes objects
Object listVAs[]=.....; // list of VAS
// produces a user-defined event of type userEvType
// no restriction on event consumers
JSEventProducer pEv1 = new JSEventProducer (this, userEvType,
JSConstants.CANYLOCATION);
// notify only those consumers registered on VAs in listVAs
JSEventProducer pEv2 = new JSEventProducer (this, userEvType,
JSConstants.CLISTVAEVENT, listVAs);
// notify only those consumers in listObj
JSEventProducer pEv3 = new JSEventProducer (this, userEvType,
JSConstants.CLISTJSOBJECTEVENT, listObj);
...
// produce a user-defined event; parameters will be
// transmitted to the handleMethod of matching consumer
Object params[]=.....;
pEv1.produceEvent(params);
```

4. نصب, راه اندازی و استفاده از JavaSymphony

در این بخش نحوه نصب و پیکر بندی JavaSymphony برای آماده سازی سیستم جهت اجرای برنامه توزیع شده وبرقراری ارتباط بین کامپیوتر های شبکه توضیح داده می شود. JavaSymphony یک سیستم مبتنی بر Agentهاست . به طور خلاصه برای اینکه برنامه های نوشته شده توسط JavaSymphony بر روی شبکه ای از کامپیوتر ها قابل اجرا باشند باید برنامه های کوچکی به نام NetworkAgentها بر روی هر یک ازکامپیوتر های شبکه اجرا شوندتا این کامپیوتر ها بتوانند با استفاده از توابع کتابخانه ای JavaSymphony با همدیگر ارتباط بر قرار کرده و برای یکدیگر قابل شناسایی باشند. برنامه لازم دیگری نیز که همراه کتابخانه JavaSymphony ارائه می شودبرنامه JS_Shell می باشد که برای تعریف و پیکر بندی یک معماری فیزیکی از ساختار سیستم توزیع شده در کامپیوتر های شبکه می باشد. معماری مجازی تعریف شده در برنامه های کاربردی, بر روی این معماری فیزیکی نگاشت خواهدشد.

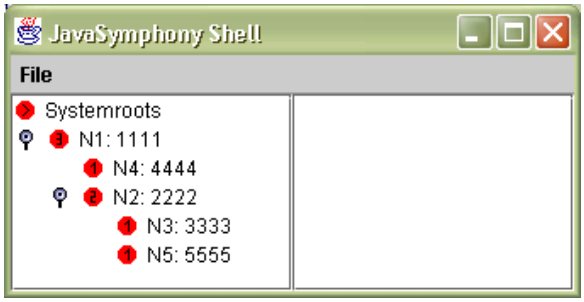
4. 1. Network Agent

به منظور اجرای برنامه های کاربردی(که از JavaSymphony استفاده می کنند) بر روی شبکه ای از کامپیوتر ها یا منابع محاسباتی ابتدا باید بر روی هر یک از کامپیوتر های موجود در شبکه برنامه ای به نام NetworkAgent اجرا شود تا این کامپیوتر ها بتوانند با یکدیگر ارتباط بر قرار کنند برای اجرای NetwokAgent در یک کامپیوتر کافیست در دایرکتوری JS دستور زیر صادر شود :

Run_na.bat [port]

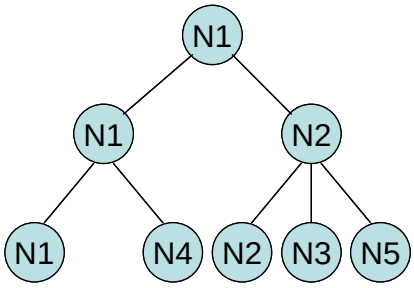
شماره پورتی که NetworkAgent به صورت پیش فرض از آن استفاده می کند 5678 می باشد. می توان شماره پورت دلخواه خود که می خواهیم NetworkAgent از آن استفاده را به عنوان آرگومان خط فرمان به برنامه ارسال کنیم. پس از اجرای NetworkAgent بر روی کامپیوتر ها باید یک معماری فیزیکی از ساختار سیستم توزیع شده ایجاد کنیم به عبارت بهتر به صورت فیزیکی ساختار درختی گره ها, خوشه ها, سایت ها و دامنه ها را تعریف کنیم. این کار نیز توسط برنامهJS_Shell انجام می گیرد که در قسمت بعد به آن خواهیم پرداخت.

حال ببینیم یک معماری مجازی تعریف شده در برنامه توزیع شده, چگونه در یک معماری فیزیکی پیدا می شود. به عنوان مثال به معماری فیزیکی زیر دقت کنید.



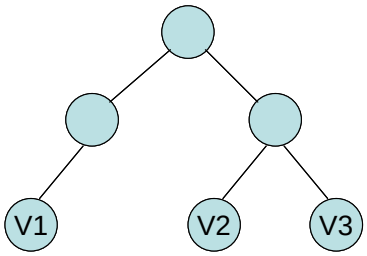
شکل 3. یک معماری فیزیکی نمونه

به خاطر دارید که که هر گره در سطح 3 می تواند بچه هایی در سطوح 2و1داشته باشد. بنابر این یک گره با سطح 3 گرهی با سطح 2 یا 1 نیز می باشد.شکل زیر این مثال را نشان می دهد :



شکل 4 . نمونه یک معماری مجازی

با این نوع نمایش درخت بهتر می توان فهمید چه معماری های مجازی قابل پیاده سازی در یک معماری فیزیکی می باشند. یک معماری مجازی درختی است که تنها برگ های آن به عنوان گره های کاری محسوب می شود. به مثال زیر از یک معماری مجازی توجه کنید :



شکل 5. گره های محاسباتی در یک معماری مجازی

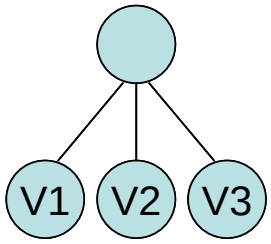
به روش های مختلفی می توان این معماری مجازی را به معماری فیزیکی شکل قبلی نگاشت کرد. بدون در نظر گرفتن پیش شرط های تعریف شده برای گره ها, جواب های زیر برای این مساله وجود خواهد داشت :

V1 = N1; V2 = N2; V3 = N3
V1 = N4; V2 = N5; V3 = N2
V1 = N5; V2 = N1; V3 = N4
…

همانطور که مشاهده می کنید جواب های مختلفی می توان برای این نگاشت پیدا کرد. البته حالت های زیر نیز درست نیست :

V1 = N2; V2 = N3; V3 = N5
…

ترتیب گره های موجود در معماری فیزیک برای یافتن یک معماری مجازی در آن اهمیتی ندارد بلکه فقط کافیست ساختاری مشابه ساختار معماری مجازی در معماری فیزیکی پیدا شود. سطح گره ریشه در معماری های مجازی و فیزیکی نیز می تواند یکسان نباشد. اما سطح گره ریشه در معماری فیزیکی همیشه باید بزرگتر یا برابر سطح گره ریشه در معماری مجازی باشد. معماری مجازی زیر را نیز می توان در معماری فیزیکی شکل فوق پیدا کرد :



شکل 6 . یک معماری مجازی ساده

جواب های ممکنه عبارت اند از :

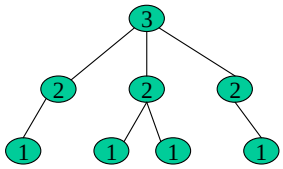
V1 = N2; V2 = N3; V3 = N5
V1 = N3; V2 = N5; V3 = N2
…

JavaSymhony Shell .2 .4

JS_Shell قسمتی از کتابخانه کلاس های JavaSymphony بوده و ابزاری گرافیکی برای طراحی معماری های فیزیکی از کامپیوتر های شبکه است که Network Agent بر روی آنها اجرا شده است. معماری مجازی توسط برنامه های ساخته شده با JavaSymphony استفاده می شود. معماری مجازی تعریف شده در این برنامه ها برای ایجاد یک سیستم توزیع شده بر روی این معماری فیزیکی نگاشت خواهد شد.در حقیقت کامپیوتر های تعریف شده در معماری فیزیکی, نقش گره های محاسباتی در معماری مجازی را بر عهده خواهند گرفت.

یک معماری مجازی، مجموعه ای از گره های محاسباتی است. این گره های محاسباتی در واحد هایی گروه بندی می شوند که به هریک از این گروه ها یک خوشه گفته می شود. به گروهی از خوشه ها نیز سایت و به گروهی از سایت ها نیز دامنه گفته می شود و این سلسله مراتب تا 9 سطح می تواند ادامه داشته باشد. مدیران خوشه ها, سایت ها و دامنه ها, همان گره های

معمولی هستند که علاوه بر وظایف گره های دیگر وظیفه مدیریت گره های موجود در خوشه, سایت یا دامنه را نیز بر عهده دارند. به جای استفاده از نام برای گره ها از شماره سطح برای آنها استفاده شده به این ترتیب که گره با سطح 1 یک گره محاسباتی معمولی, گره با سطح 2 مدیر یک خوشه,گره با سطح 3 مدیر سایت و... در شکل 8 یک گره با سطح 3 (مدیر سایت)، دو گره با سطح 2 (مدیران خوشه ها) و تعداد یکسانی گره محاسباتی در هر خوشه رسم شده است. توجه داشته باشید که این سیستم توزیع شده از هشت Network Agentتشکیل شده است.



شکل 8 : یک معماری مجازی با 8 گره

4. 3. اجرای JS_Shell

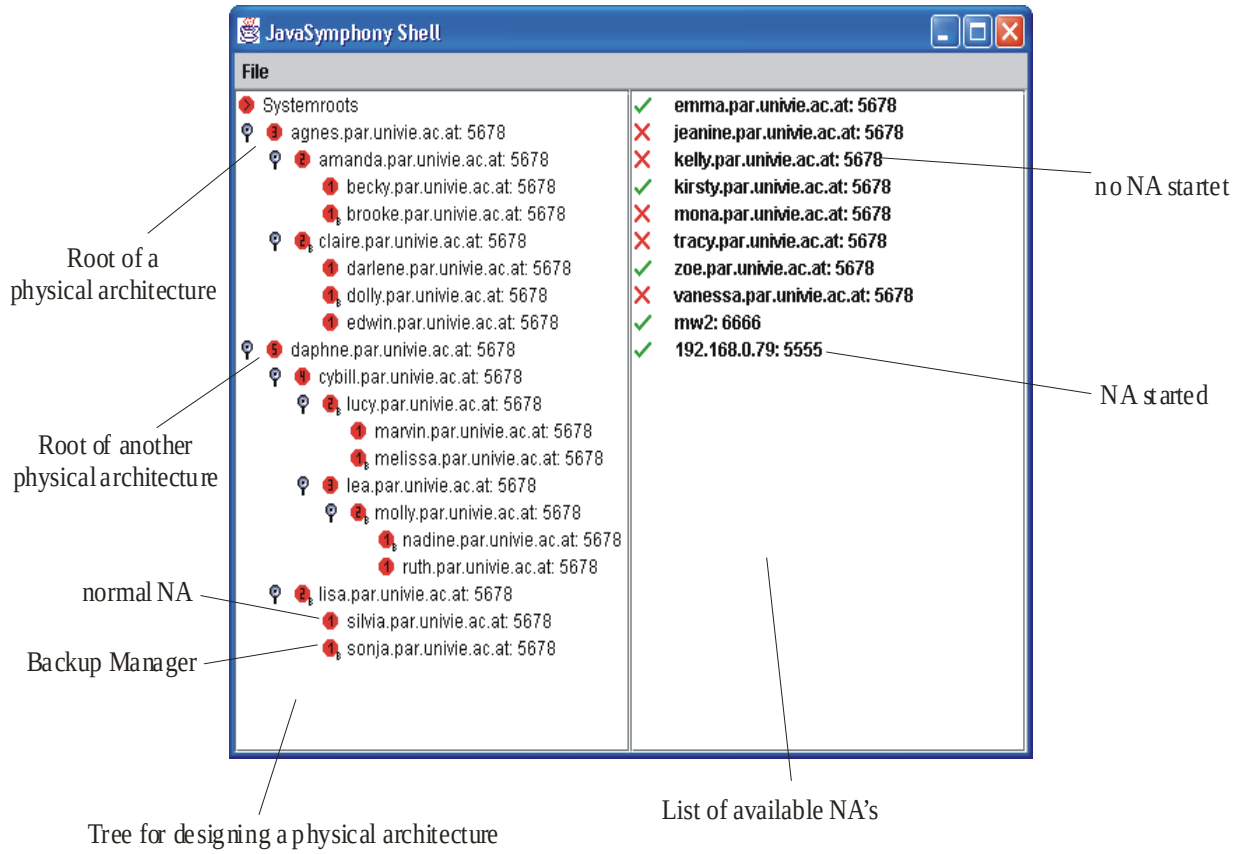
می توان برنامه JS_Shell رابا اجرای فایل run_shell.bat اجراکرد. این فایل دردایرکتوری اصلی JavaSymphony قرار دارد. با اجرای این فایل یک رابط کاربری گرافیکی نمایش داده می شود که می توان با استفاده از آن یک معماری فیزیکی برای سیستم های توزیع شده, از کامپیوتر های موجود در شبکه ساخت.اگر قبلا یک معماری فیزیکی ساخته و آن را ذخیره کرده باشید (قسمت Save Configuration), می توانید به کمک JS_Shell آن را مجددا بارگذاری نمایید. برای این کار نیازی به اجرای رابط کاربری گرافیکی نبوده و می توان در خط فرمان دستور زیر را صادر کرد :

run_shell [filename]

به این ترتیب معماری فیزیکی ذخیره شده در فایل بارگذاری شده و در صورت بروز خطا در راه اندازی آن, این خطا ها گزارش داده می شوند. پس از بارگذاری کامل معماری فیزیکی نیز برنامه JS_Shell خاتمه می یابد.

4. 4. پنجره اصلی JS_Shell

نمونه ای از پنجره اصلی JS_Shellدر شکل 9 نشان داده شده است



شکل 9 :

نمونه ای از پنجره اصلی JS_Shell

پنجره اصلی به دو بخش مجزا تفکیک شده است.در سمت راست لیستی از Network Agentهای در دسترس نمایش داده می شودکه می توانند به ساختار معماری فیزیکی درسمت چپ اضافه شوند. هر Network Agent با نام یا IP کامپیوتری که برنامه Network Agentبر روی آن اجرا شده و به وسیله شبکه ای به این کامپیوتر وصل است، نمایش داده می شود. همچنین شماره پورتی که کامپیوتر در آن پورت Listeningمی کندنیزمقابل نام هرکدام از NAها نمایش داده می شود (computer name : port)هنگام راه اندازی JS_Shell لیست پیش فرض ذخیره شده در فایل Serverlist.ini بار گذاری شده ودر سمت چپ نمایش داده می شود. همچنین پیکربندی ذخیره شده در فایل config.txt بار گذاری شده و در سمت راست به صورت درختی از NAها نمایش داده می شود. البته کاربر می تواند لیست ها و پیکر بندی های ذخیره شده دلخواه را بار گذاری کرده و نمایش دهد.(بخش load configurationوload NA-list)

لیست *NA*ها : در لیست NAها که در سمت راست پنجره اصلی نشان داده می شود NAهایی که هم اکنون در دسترس بوده و قابل استفاده هستند با یک چک مارک سبز در ابتدایشان مشخص می شوند. همچنین کامپیوتر هایی که Network Agentبر روی آنها اجرا نشده و یا فعلا در دسترس نیستند نیز با یک علا مت ضربدر قرمز در ابتدایشان مشخص می شوند. NAهایی نیز که هنوز علامتی ندارند, وضعیت نا معلوم داشته و طی چند ثانیه بعدی وضعیتشان مشخص خواهد شد. کاربر می تواند NAهایی را به لیست اضافه کرده یا از آن حذف کند. همچنین می تواند لیست را در فایلی ذخیره کرده و یا لیست ذخیره شده قبلی را بار گذاری کند. برای اضافه کردن یک NA به ساختار معماری مجازی می توان آن را به مکان دلخواه دردرخت (در سمت چپ پنجره اصلی) Drag کرد.

درخت معماری فیزیکی : قسمت سمت چپ پنجره اصلی, امکان طراحی یک معماری فیزیکی به صورت درختی از NAهای در دسترس را به کاربر می دهد. کاربر می تواند NA مورد نظر خود را با Drag از لیست و انداختن به مکان دلخواه از درخت، به معماری فیزیکی اضافه کند. برای حذف یک NA از معماری فیزیکی نیز کافیست آن را از قسمت درخت به قسمت لیست

Drag کرد. درخت طراحی شده برای معماری فیزیکی را نیز می توان در فایلی ذخیره و بعدا بازیابی کرد. بعد از طراحی معماری فیزیکی با انتخاب گزینه Make Configuration از منو، معماری فیزیکی عملا ساخته شده و هر یک از گره های طراحی شده در درخت معماری فیزیکی با توجه به ساختار درخت واقعا به یکدیگر متصل خواهند شد. درختی که در شکل فوق نمایش داده شده، نشان دهنده دو معماری فیزیکی مستقل است. ریشه یکی agnes.par.univie.ac.at و ریشه دیگری daphne.par.univie.ac.atمی باشد گرهی به نام Systemrootsصرفا یک گره مجازی بوده و یک NAواقعی نیست.اولین درخت (agnes)یک معماری فیزیکی با 3سطح است که دارای 2 NAدر سطح 2 (amanda,claire)می باشد. NAاول (amanda) دارای 2 NAدر سطح 1 است (becky,brooke). NA دوم نیز دارای 3 NAدر سطح یک است (darlene,dolly,edwin). دومین درخت(daphne) نیز یک معماری فیزیکی با پنج سطح است که دارای دو NAدر سطح چهار به نام cybill و lisa و ... می باشد. گره sonja.par.univie.ac.at یک مدیر پشتیبان یا backup manager است که به وسیله یک B کوچک مشخص شده است. گرهی مانند silvia.par.univie.ac.atنیز یک NAمعمولی است. NAهای در سطح یک را گره, در سطح دو را خوشه, در سطح سه را سایت و در سطح چهار را دامنه می نامیم.(البته می توان درختی تا نه سطح را ایجاد کرد).

منوی اصلی:منوی اصلی نیز به دو قسمت تقسیم شده است. قسمت اول مربوط به درخت معماری, و قسمت دوم مربوط به لیست NAها است

Load Configuration, Save Configuration, Save Configuration as : این 3 گزینه برای ذخیره سازی و بارگذاری درخت طراحی شده برای معماری فیزیکی می باشند.هنگام اجرای JS_shell پیکربندی ذخیره شده در فایل Config.txt به صورت اتوماتیک بارگذاری ونمایش داده می شود. می توان برای ایجاد یک معماری فیزیکی, که قبلا طراحی و ذخیره شده است بدون اجرای GUI و از خط فرمان, نام فایل مورد نظر را به عنوان آگومان خط فرمان به JS_Shell داد.

Make Configuration : بعد از طراحی معماری فیزیکی با استفاده از این گزینه می توان معماری طراحی شده را بر روی شبکه ساخت. با اجرای این گزینه ارتباط تمامی NAها، طبق سلسله مراتبی که دردرخت مشخص شده است به صورت فیزیکی بر قرار می شود. پارامتر های پشتیبانی و نیز گره هایی که باید بچه های خود را مدیریت کنند نیز پیکر بندی شده و شروع به کار می کنند. اگر پیکر بندی معماری فیزیکی با موفقیت انجام گیرد پیامی نمایش داده خواهد شد در غیر این صورت یک پیام خطا ظاهر خواهد شد.

Load NA-List, Save NA-List, Save NA-List as : این سه گزینه نیز کاربر را قادر می سازد تا لیست NAهای موجود در سمت راست پنجره اصلی را در فایلی ذخیره کرده و یا بارگذاری کند. هنگام راه اندازی JS_Shellلیست موجود در فایل serverliste.ini به صورت اتوماتیک بارگذاری می شود.

محتویات منوی *popup* برای قسمت طراحی درخت : با راست کلیک کردن در قسمت سمت چپ پنجره اصلی(درخت طراحی) یک پنجره popupنمایش داده می شود. این پنجره حاوی همان دستوراتی است که در قسمت اول منوی اصلی نیز وجود دارند.

محتویات منوی *popup* برای *NA*های موجود در درخت: با راست کلیک کردن در روی هریک از NAهای موجود در درخت طراحی شده برای معماری فیزیکی یک پنجره popup نمایش داده می شود. این پنجره حاوی برخی از دستورات موجود در منوی اصلی و چند دستور دیگر است که در زیر توضیح داده می شوند.

Set Level : از این دستور برای تنظیم صریح سطح یک NA استفاده می شود. شماره سطح یک NA باید بیشتر از تعداد سطوح بچه های آن باشد. به عنوان مثال اگر گرهی دارای بچه هایی در سطح پایینی خود باشد نمی تواند شماره سطح 1 داشته باشد. اگر کاربر شماره سطح یک NA را کوچکتر از مقدار فعلی آن کند، JS_Shell تلاش می کند به صورت بازگشتی شماره سطح تمام بچه های آن گره را کاهش دهد. (البته تا زمانی که شماره سطح گره ها بزرگتر از صفر باشد).

Set Rate : با انتخاب این گزینه دیالوگی نمایش داده می شود که می توان با آن نرخ دوره تناوب بررسی بچه ها توسط یک گره پدر (برای اینکه ببنید هنوز در دسترس هستند یا خیر) را تغییر داد.. مقدار پیش فرض این نرخ 10000می باشد.

Toggle Backup Manager : یک مدیر پشتیبان یا backup manager گرهی است که به طور متناوب (با دوره تناوبی که توسط set Backup Rate مشخص می شود) گره پدر خود را بررسی می کند که آیا در دسترس بوده و فعال است یا خیر. اگر نبود وظیفه وی را این گره بر عهده خواهد گرفت. یک NA را زمانی می توان توسط این گزینه به عنوان یک گره پشتیبان معرفی کرد که وی حد االقل دارای یک گره پدر باشد هر گره مدیر نیز فقط می تواند یک مدیر پشتیبان داشته باشد.بنابراین اگر قبلا مدیر پشتیبانی برای یک گره مشخص شده باشد با انتخاب مدیر پشتیبان جدید, پشتیبان قبلی به یک گره معمولی تبدیل می شود. مدیر پشتیبان توسط یک Bکوچک در درخت طراحی از دیگر گره ها متمایز می شود. تغییر دادن مدیر پشتیبان تا اجرای گزینه Make Configuration بر روی معماری فیزیکی اعمال نخواهد شد.

Set Backup Rate : با اجرای این گزینه دیالوگی نمایش داده می شود که می توان در آن نرخ دوره تناوب بررسی گره پدر، توسط مدیر پشتیبان (جهت کنترل اینکه هنوز فعال و در دسترس است یا خیر) را تغییر داد. مقدار پیش فرض آن 10000میلی ثانیه بوده و تغییر دادن آن تا اجرای گزینه Make Configuration برروی معماری فیزیکی اعمال نخواهد شد.

Show Info : اگر معماری فیزیکی قبلا با اجرای دستور Make Configuration ساخته شده و یک برنامه کاربردی (JS Application)نیز بر روی آن در حال اجرا باشد با اجرای این گزینه بر روی NA اطلاعاتی در مورد برنامه توزیع شده و اشیائی که بر روی این گره مستقر شده اند نمایش داده می شود.

Event Properties : کاربر می تواند با استفاده از این گزینه تغییراتی در مکانیزم رویداد های Event Agent اجرا شده در گره بدهد (برای اطلاعات بیشتر به مستندات Object Agent System وساختار پیاده سازی JavaSymphonyمراجعه شود).

Threadpoolsize (PubOA): با اجرای این گزینه دیالوگی نمایش داده می شود که توسط آن می توان حجم نخ های قابل اجرا در Public Object Agent اجرا شده در این NAرا مشاهده کرده یا تغییر داد. (برای اطلاعات بیشتر به مستندات Object Agent System و ساختار پیاده سازی JavaSymphony مراجعه شود). تغییرات انجام گرفته بلافاصله اعمال خواهند شد.

محتویات منوی *popup* برای قسمت لیست *NA* ها : با کلیک راست در قسمت سمت چپ پنجره اصلی (لیست NA ها)، یک پنجره popup نمایش داده می شود که شامل برخی از دستورات موجود در قسمت دوم منوی اصلی و چند دستور دیگر است که در ادامه توضیح داده می شود.

Remove All : با اجرای این گزینه تمام NA های موجود در لیست از آن حذف می شوند.

Add NA : اجرای این گزینه موجب باز شدن دیالوگ جدیدی می شود که می توان توسط آن NA جدیدی به لیست اضافه کرد. برای این کار باید نام و شماره پورت کامپیوتری از شبکه که برنامه Network Agent در آن کامپیوتر و در آن پورت در حال اجرا است وارد شود.می توان به جای نام کامپیوتر, آدرس IP آن را وارد کرد (اگرچه این کار توصیه نمی شود).

محتویات منوی *popup* برای *NA* های موجود در قسمت لیست *NA* ها: با کلیک راست بر روی NAهای موجود در قسمت سمت چپ پنجره اصلی (لیست NA ها), یک پنجره popup نمایش داده می شود که شامل برخی از دستورات موجود در قسمت دوم منوی اصلی و چند دستور دیگر است که در ادامه توضیح داده می شود.

Get current Configuration : با انتخاب این گزینه اگر NA قسمتی از یک معماری فیزیکی فعال باشد, این معماری به صورت یک درخت در سمت چپ نمایش داده خواهد شد.

Ping: این برنامه تلاش می کند تا با NA مورد نظر ارتباط برقرار کند. اگر موفق شود NA با یک چک مارک سبز علامت گذاری می شود. در غیر این صورت یک ضربدر قرمز در کنار NA نشان داده می شود.هر NA ای که در لیست نمایش داده شده خود هر 30ثانیه برنامه Ping را اجرا می کند.

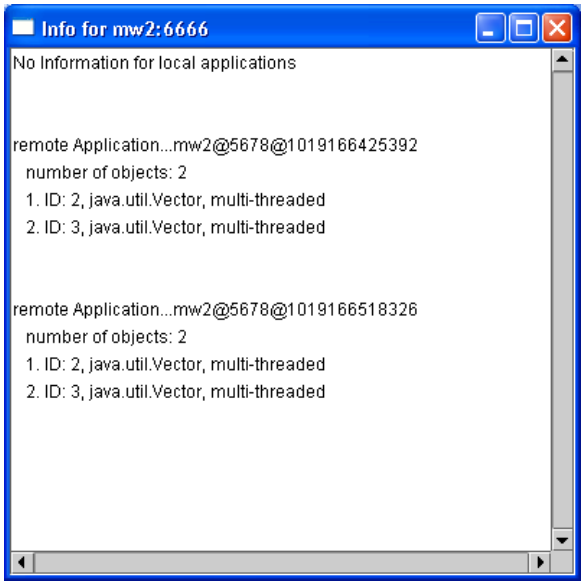
Show Info : همان اطلاعاتی که هنگام انتخاب گزینه Show info در روی NAهای سمت چپ(درخت طراحی) نشان داده می شود را نمایش می دهد .

Event Properties : همان پنجره ای که هنگام انتخاب گزینه Event Properties درروی NAهای سمت چپ(درخت طراحی) نشان داده می شود را نمایش می دهد .

Threadpoolsize (PubOA) : همان پنجره ای که هنگام انتخاب گزینه Threadpoolsizeدر روی NAهای سمت چپ(درخت طراحی) نشان داده می شود را نمایش می دهد

Remove: NAانتخاب شده را از لیست NAها حذف می کند.

پنجره اطلاعات: می توان با انتخاب گزینه Show Info از منوی popup ظاهر شده بر روی NAها، اطلاعاتی در مورد برنامه توزیع شده در حال اجرا، و اشیائی که بر روی NAها مستقر شده اند را به دست آورد.نمونه ای از این پنجره در شکل زیر نمایش داده شده است.



شکل 10 : نمونه ای از پنجره خصوصیات گره

این پنجره اطلاعاتی در مورد یکNAبا نام و شماره پورت mw2:6666 نمایش می دهد. هیچ برنامه محلی بر روی این گره در حال اجرا نیست. اما دو برنامه دور که از این NA استفاده می کنند وجود دارد. هر دو برنامه اشیائی را در این NAمستقر کرده اند. هر برنامه کاربردی در کل سیستم به کمک یک شناسه یکتا شناسایی می شود این شناسه از نام و شماره پورت NAای که برنامه روی آن در حال اجراست بعلاوه یک شماره یکتای درون NA تشکیل می شود.

حرکت اشیا: می توان اشیا مستقر شده در یک NAرا با Dragکردن و انداختن آن شی از پنجره اطلاعات NAبه پنجره اطلاعات یک NAدیگر در سیستم توزیع شده حرکت داد.برای مهاجرت یک شی به NAدیگر باید برنامه کاربردی یکسانی بر روی آن NA نیزدر حال اجرا باشد.

تغییرمیزان نخ های قابل اجرا در *AppOA*: میتوان میزان نخ های قابل اجرا در Application Object Agent(AppOA) را توسط پنجره اطلاعات تغییر داد. برای این منظور می توان بر روی متن local application (اگر برنامه محلی باشد) کلیک راست کرده و از آنجا مقدار جدید را وارد کرد. تغییرات داده شده بلافاصله اعمال خواهند شد. (برای اطلاعات بیشتر به مستندات Object Agent Systemو ساختار پیاده سازی JavaSymphonyمراجعه شود).

نحوه ایجاد یک معماری فیزیکی جدید : مراحل زیر برای ایجاد یک معماری فیزیکی جدید با استفاده از JS_Shell باید پیموده شود.

1- اگر درخت طراحی شده ای در قسمت چپ پنجره اصلی وجود دارد با Dragکردن تمام گره های آن به قسمت لیست NA ها (سمت راست) درخت را پاک کنید.(ریشه مجازی درخت یا Systemrootحذف نمی شود زیرا یک NAواقعی نیست).

2-NAهای مورد نظر خود را جهت ایجاد معماری فیزیکی به سمت چپ (لیست NAها) اضافه کنید.

3-با Drag کردن و انداختن NAها از سمت راست پنجره اصلی, به سمت چپ آن و ایجاد یک سلسله مراتب درختی از NA ها, معماری فیزیکی دلخواه خود را طراحی کنید. برای انجام این کار فقط باید از NA هایی که با چک مارک سبز علامت گذاری شده اند استفاده کرد. (زیرا بقیه فعلا در دسترس نبوده, یا فعال نبوده و یا ارتباط با آنها بر قرار نشده است). همچنین میتوانید NAها را از مکان های مختلف درخت به مکان های دیگرآن با Dragکردن، منتقل کنید. می توانید شماره سطح گره ها را صریحا با استفاده از دستور Set Level نیزمشخص کنید (البته ممکن است شماره مشخص شده برای یک سطح قابل اعمال نباشد).

به ازای هر گره در درخت که دارای بچه باشد باید یکی از بچه های آن را به عنوان Backup Managerیا مدیر پشتیبان، مشخص کرد. برای یک مدیر پشتیبان نیز می توان مقدار Bakup Rateرا تغییر داد. می توان نرخ دوره تناوب بررسی بچه ها توسط پدر (جهت کنترل فعال و در دسترس بودن) را نیز توسط دستور Set Rateتغییر داد. پس از اتمام طراحی، معماری فیزیکی با اجرای دستور Make Configuration ساخته می شود. اگر معماری فیزیکی با موفقیت ساخته شود برنامه های کاربردی JavaSypmhony می توانند با تعریف معماری های مجازی که از این معماری فیزیکی استخراج خواهند شد, سیستم توزیع شده دلخواه خود را ایجاد کنند. در صورت لزوم می توانید برخی از ویژگی های رویداد ها و میزان نخ های قابل اجرا در NAها را تغییر دهید. این کار می تواند در هر زمانی و بعد از شروع به کار برنامه کاربردی صورت گیرد.

پس از اینکه برنامه کاربردی که از JavaSymphony استفاده می کند در سیستم شروع به کار کرد می توانید نحوه اجرای آن، محل استقرار اشیا، و حرکت اشیا بین NAهای مختلف را به کمک پنجره نمایش اطلاعات مشاهده کرده و کنترل نمایید.

اگر قصد استفاده مجدد از معماری فیزیکی طراحی شده را داشته باشید می توانید به کمک دستورات ذخیره و بارگذاری، معماری فیزیکی طراحی شده و لیست NA های خود را ذخیره کنید.

[1] A. Alexandrov, M. Ibel, K. Schauser, and C. Scheiman. SuperWeb: Towards a global web-based parallel computing infrastructure. In *The 11 th IEEE International Parallel Processing Symposium (IPPS)*, pages 100–106, 1997.

[2] R. Aversa, B. D. Martino, N. Mazzocca, M. Rak, and S. Venticinque. Integration of mobile agents and openmp for programming clusters of shared memory processors: a case study. accepted for publication in proc. of PaCT 2001 Conference, 8-12 Sept. 2001, Barcelona, Spain, 2001.

[3] B. O. Christiansen, P. Cappello, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu. Javelin: Internet-based parallel computing using Java. *Concurrency: Practice and Experience*, 9(11):1139–1160, Nov. 1997. Special Issue: Java for computational science and engineering — simulation and modeling II.

[4] P. Ciancarini and D. Rossi. Jada - Coordination and Communication for Java Agents. In J. Vitek and C. Tschudin, editors, *Mobile Object Systems: Towards the Programmable*

Internet, volume 1222 of *Lecture Notes in Computer Science*, pages 213–228. Springer-Verlag: Heidelberg, Germany, Apr. 1997.
[5] T. Fahringer. JavaSymphony: A System for Development of Locality-Oriented Distributed and Parallel Java Applications. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER 2000), www.par.univie.ac.at/project/javasymphony*, Chemnitz, Germany, 2000. IEEE Computer Society.
[6] T. Fahringer and A. Jugravu. JavaSymphony: New Directives to Control Locality, Parallelism, and Load Balancing for Cluster and Grid-Computing. Technical Report TR2002-11, Institute for Software Science, University of Vienna, February 2002.
[7] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces principles, patterns, and practice*. Addison-Wesley, Reading, MA, USA, 1999.
[8] H.Moritsch and G.Ch.Pflug. Java Implementation of Asynchronous Parallel Nested Optimization Algorithms. In *Proceedings of the 3rd Workshop on Java for High Performance Computing, Sorrento, Italy*, June 2001.
[9] T. Kielmann, P. Hatcher, L. Bouge, and H. Bal. Enabling Java for High-Performance Computing. *Communications of the ACM*, 44(10):110–117, October 2001.
[10] E. Laure and H. Moritsch. Portable Parallel Portfolio Optimization in the Aurora Financial Management System. In *Proceedings of SPIE ITCom 2001 Conference: Commercial Applications for High-Performance Computing*, Denver, Colorado, August 2001.
[11] J. Moreira, S. Midkiff, M. Gupta, P. Artigas, P. Wu, and G. Almasi. The NINJA Project. *Communications of the ACM*, 44(10):102–109, October 2001.
[12] M. Philippsen and M. Zenger. JavaParty — transparent remote objects in Java. *Concurrency: Practice and Experience*, 9(11):1225–1242, Nov. 1997.
[13] Javaparty homepage: http://wwwipd.ira.uka.de/javaparty/.
[14] G. Stefanescu. “Interactive systems”: - from folklore to mathematics. In *Proc. 6th International Workshop on Relational Methods in Computer Science*, pages 208–221, Oisterwijk (near Tilburg),The Netherlands, 2001. Also Springer