مدیریت نیازهای غیر عملیاتی در پارادایم سرویس گرا بر اساس ایده جنبه گرایی

مهرنوش باقری [*۱]، محمد یوسف نژاد[۲]، علی ریحانیان[۳]

۱- دانشگاه علامه طبرسی، قائمشهر، ایران، merbagheri@outlook.com

۲- دانشگاه هوا و فضای نانجینگ، نانجینگ، چین، myousefnezhad@nuaa.eud.cn

۳- دانشگاه تبریز، تبریز، ایران، areihanian@ustmb.ac.ir

خلاصه

پارادایم سرویس‌گرا با هدف توصیف یک سازمان بر اساس سه لایه ی کسب و کار، نرم افزار و زیر ساخت به صورت سرویس گرا معرفی شده است. از آنجایی که این چهارچوب دارای اعتبار بالایی در حوزه معماری سازمانی می باشد و می توان آن را شیوه معماری و مدیریت سازمان های در آینده دانست پس دور از ذهن نیست که این معماری روی شیوه تجزیه و تحلیل نیازهای نرم افزار های تولید شده در آینده تاثیر بسزایی بگذارد. یکی از عواملی که در مهندسی نرم افزار جهت تولید نرم افزارهای سازگار با پارادایم سرویس گرا باید مورد توجه قرار گیرد روش مستندسازی نیازها در چنین نرم افزارهایی است از میان نیازهای یک سیستم آن چیز که بیشتر باعث سازگاری یا عدم سازگاری یک نرم افراز با محیط تعریف شده معماری در پارادایم سرویس گرا می شود نیازهای غیر عملیاتی سیستم می باشد. از این رو ما بر اساس ایده مطرح شده در جنبه گرایی یک فرآیند استاندارد جهت جمع آوری، مدیریت و ایجاد توازن بین نیازهای غیر عملیاتی را مطرح می کنیم و ثابت خواهیم کرد این مدل با بالا بردن سطح برازندگی نیازهای غیر عملیاتی کمک به سزایی در بالا بردن کیفیت ، افزایش نرخ مدیریت و ایجاد توازن نیازهای غیر عملیاتی در سازمان خواهد کرد و علاوه بر آن می توان از محصول نهایی این فرآیند در معماری نرم افزار سازمان نیز استفاده نمود.

کلمات کلیدی: معماری سازمانی سرویس گرا، جنبه گرایی، نیازهای غیر عملیاتی، ماتریس نیازهای غیر عملیاتی

۱. مقدمه

معماری سازمانی[۱] مجموعه ای از ارائه های توصیفی (مدل) در ارتباط با تشریح یک سازمان می باشد به گونه ای که این مدل قابل مدیریت و در دوره حیات مفیدش قابل نگهداشت باشد. [۱, ۲]

معماری سرویس گرا سبکی از معماری سیستم های اطلاعاتی به شمار می آید که یکی از اهداف آن دستیابی به اتصال سست[۲] در ارتباطات بین مولفه های نرم افزاری است. [۳] و همچنین روشی برای طراحی و پیاده سازی نرم افزارهای گسترده سازمانی به وسیله ارتباط بین سرویس هائی که دارای خواص اتصال سست، دانه درشتی، پیمانه‌ای بودن[۳]، محصورسازی[۴] و قابل استفاده مجدد[۵] هستند. [۳, ۴, ۵]

در سال های اخیر، جریان های تحقیقاتی به سمت ترکیب این دو مفهوم با هم رفته اند. این جریان ها ابتدا به معرفی سازمان سرویس گرا پرداخته اند که تاثیرات آن شامل تغییراتی در لایه نرم افزارهای کاربردی می باشد[۶] و سپس این

* **Corresponding author**

ایده را مطرح کردند که سایر لایه ها (زیر ساخت و کسب و کار) نیز باید تحت تاثیر معماری سازمانی قرارگیرند و با ارائه پارادایم سرویس گرا[۶] به این منظور رسیدند. [۷]

پارادایم سرویس گرا در سه حوزه مطرح می شود : اول در لایه بالا در حوزه کسب و کار با عنوان سازمان سرویس گرا و دوم در لایه میانی در حوزه نرم افزار با عنوان معماری سرویس گرا و سوم در لایه پایین در حوزه زیر ساخت با عنوان زیر ساخت سرویس گرا. [۷]

در سال های اخیر یک مدل برنامه نویسی جهت پیاده سازی سریع تر و راحت تر کد های برنامه ارائه شده است که به مدل جنبه گرا معروف شده است این مدل با جداسازی دغدغه های تداخلی (که عمدتاً نیازهای غیر عملیاتی می باشند) از بدنه اصلی کد ، تمرکز برنامه نویس را روی الگریتم اصلی برنامه متمرکز می کند. [۸]

برنامه نویسی جنبه گرا علاوه بر اینکه یک متد جدید برنامه نویسی می باشد پتانسیل آن را دارد تا در حوزه تجزیه و تحلیل نرم افزار نیز به کار برده شود از این رو متدهای توسعه جنبه گرا برای مهندسی نرم افزارهای جنبه گرا توسعه یافته است. [۹]

ما در این مقاله با استفاده از ایده جنبه گرایی و بسط آن به حوزه پارادایم سرویس گرا روشی را ارائه می دهیم تا جمع آوری، مدیریت و ایجاد توازن بین نیازهای غیر عملیاتی در روند اجرای پارادایم سرویس گرا در یک سازمان آسان تر شود. این روش هم در اجرای فرآیند معماری سازمانی سرویس گرا مفید خواهد بود و کمک بسزایی در اجرای معماری سازمانی بر اساس مستندات تهیه شده خواهد کرد و هم در تدوین مستند نیازهای نرم افزارهایی که قرار است در سازمانی که بر اساس پارادایم سرویس گرا بنا شده کار کنند بکار خواهد رفت.

در ادامه این مقاله ما ابتدا در بخش دوم به بررسی پیش زمینه های مورد نیاز می پردازیم و در بخش سوم کارهای انجام شده در این زمینه را بیان می کنیم و سپس در بخش چهارم به ارائه مدل پیشنهادی این مقاله می پردازیم و در بخش پنجم به ارزیابی و بررسی فواید و مشکلات مدل پیشنهادی خواهیم پرداخت و در نهایت در بخش ششم نتایج ارائه این مقاله و خط و مشی کارهای آتی را بیان می کنیم.

۲ -پیش زمینه

۲-۱- پارادایم سرویس گرا

معماری سازمانی مجموعه ای از ارائه های توصیفی (مدل) در ارتباط با تشریح یک سازمان می باشد به گونه ای که این مدل قابل مدیریت و در دوره حیات مفیدش قابل نگهداشت باشد. [۱, ۲]

معماری سازمانی رهیافتی بالا به پائین است که نگرشی کلان به ماموریت ها و وظایف سازمانی، موجودیت های اطلاعاتی، فرآیندهای کاری، شبکه های ارتباطی ، سلسله مراتب و ترتیب انجام کارها در یک سازمان که با هدف ایجاد سامانه های اطلاعاتی یکپارچه و کارآمد، را با توجه به نیازهای کسب و کار (اهداف، استراتژی ها، نیازها و ..) در یک سازمان فراهم می کند.[۱, ۲, ۶, ۷, ۱۰]

معماری سرویس گرا سبکی از معماری سیستم های اطلاعاتی به شمار می آید که یکی از اهداف آن دستیابی به اتصال سست در ارتباطات بین مولفه های نرم افزاری است. [۳] و همچنین روشی برای طراحی و پیاده سازی نرم افزارهای گسترده سازمانی به وسیله ارتباط بین سرویس هائی که دارای خواص اتصال سست، دانه درشتی، پیمانه ای بودن ، محصورسازی و قابل استفاده مجدد هستند. [۳, ۴, ۵] معماری سرویس گرا رویکردی پایین به بالا است که عموماً در حوزه پیاده سازی و تهیه نرم افزار کاربرد دارد و بیشتر به تشریح مسائل تکنیکی جهت پیاده سازی می پردازد تا به مسائلی همانند مسائل کسب و کار. تقریباً از سال ۲۰۰۰ به بعد یک سری جریان های تحقیقاتی و مقالات باعث شد تا ایده ترکیب

معماری سازمانی و معماری سرویس گرا مطرح شود. که نتیجه این تحقیقات معماری سازمانی سرویس گرا[۷] بوده است. [۶, ۷, ۱۰]

چگونگی رابطه و اشتراک معماری سازمانی و سرویس گرا و امکان ترکیب این دو رویکرد در معماری سازمانی سرویس گرا یکی از موضوعات پرطرفدار سال های اخیر بوده است ولی تقریباً در تمامی این مستندات تا قبل از سال ۲۰۰۸ تاثیر معماری سازمانی سرویس گرا را شامل تغییراتی در لایه نرم افزاری دیده اند و به تاثیرات آن در لایه زیر ساخت و کسب و کار پرداخته نشده است. در نیمه دوم سال ۲۰۰۸ شورای مدیران ارشد اطلاعاتی دولت ایالت متحده آمریکا نتایج مطالعات و بررسی های چند ساله خود را در حوزه معماری سازمانی با عنوان "راهنمای کاربردی برای معماری سرویس گرای فدرال" منتشر کرد. در این مطالعات حوزه تاثیرات معماری سازمانی سرویس گرا را در سه لایه ۱-کسب و کار ۲- نرم افزار ۳- زیرساخت پرداخته است به این ایده پارادایم سرویس گرا می گویند. [۶, ۷]

شکل ۱ –لایه های معماری در پارادایم سرویس گرا [۷]

پارادایم سرویس گرا یک سازمان را به سه حوزه تقسیم می کند و در حوزه کسب و کار، سازمان سرویس گرا[۸] و در حوزه نرم افزار، معماری سرویس گرا[۹] و در حوزه زیر ساخت، زیر ساخت سرویس گرا[۱۰] را مطرح می کند شکل (۱) نمایانگر این دیدگاه است.[۷]

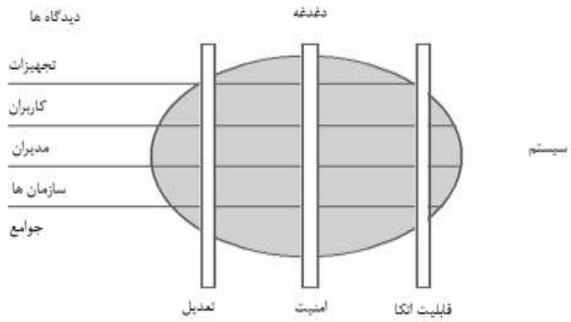
۲-۲- جنبه گرایی

درجهان امروزی، استفاده ی صحیح و کاربردی یک سیستم نرم افزاری می تواند بسیار حائز اهمیت باشد. وظیفه مندی[۱۱]، انعطاف پذیری[۱۲]، دسترسی مداوم[۱۳] و عملکرد صحیح[۱۴]، عواملی هستند که به سودمندی سیستم های نرم افزاری مربوط می شوند. معمولاً تغییر در طول عمر یک سیستم رخ داده و حمایت عوامل فوق را لازم دارد. برای مثال، ممکن است تغییرات در راستای افزایش عملکرد، بهبود وظیفه مندی یا رسیدگی به خطاهای کشف شده در سیستم باشد.

همانطور که می دانید می توان نیازمندی های یک نرم افزار را به نیازمندی های عملیاتی[۱۵] و نیاز مندی های غیر عملیاتی[۱۶] تقیسم کرد. کار جنبه گرایی مدیریت نیازمندهای غیر عملیاتی در برنامه توسط جنبه ها[۱۷] می باشد. [۸, ۱۱]

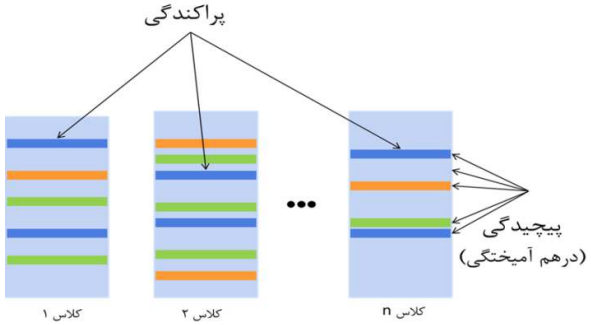
یکی از مفاهیم مهم در جنبه گرایی دغدغه می باشد که نگرشی متفاوت به نیازهای غیر عملیاتی می باشد [۱۲]. دغدغه یک نیازمندی و شاخص ضروری در یک سیستم است که با استفاده از یک "ساختار کد"[۱۸] پیاده سازی می شود [۱۱] این تعریف به ما این اجازه را می دهد که یک دغدغه محدود به سیستم های شی گرا نباشد و یک سیستم ساخت یافته نیز بتواند چنین مفهومی داشته باشد. در اصل ما با شناسایی دغدغه ها و پیاده سازی آنها در جنبه ها کد اصلی را مستقل از تاثیرات آن بر روی نیازهای غیر عملیاتی پیاده سازی می کنیم و برای ارتباط بین کد اصلی برنامه که در اصل

بخشی از یک نیاز عملیاتی را فراهم می کند و یک یا چند جنبه که در اصل اشاره به نیاز های غیر عملیاتی دارد از مفاهیم و تکنیک های جنبه گرایی استفاده می کنیم.[۸, ۱۱]



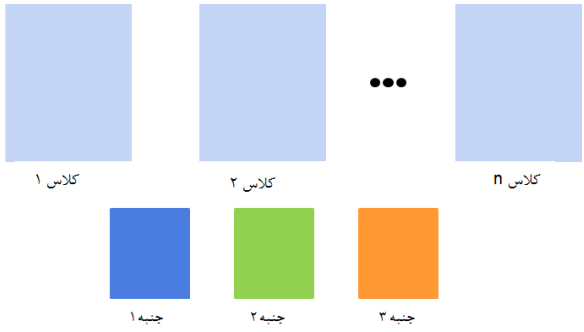
شکل 2 –دغدغه های سیستم نرم افزار در معماری جنبه گرا[۱۱]

شکل (۲) یک مثال از دغدغه های تداخلی می باشد. همانطور که می بینید مفاهیم دغدغه و نیازهای غیرعملیاتی بسیار به هم نزدیک می باشند. در تحلیل جنبه گرا ما دنبال این هستیم که هر کدام از این دغدغه ها را در یک جنبه پیاده سازی کنیم و هر گاه در کد اصلی نیاز به آن جنبه خاص بود همانند امنیت یک ارجاع به آن جنبه داشته باشیم.



شکل 3 –کلاس های برنامه نویسی در شی گرایی[۸]

شکل (۳) کلاس های یک برنامه بر اساس متد شی گرایی می باشد در این شکل بلوک های رنگی داخل کلاس ها جنبه های برنامه می باشد.



شکل 4–کلاس های برنامه نویسی در جنبه گرایی [۸]

شکل (۴) مدل کلاس های برنامه در متد جنبه گرایی است همان طور که می بینید کلاس های برنامه فقط شامل کد هایی که نیازهای عملیاتی را پیاده سازی می کنند می باشد و جنبه های زیر شکل (۴) در اصل جهت پیاده سازی نیازهای غیر عملیاتی می باشند که هر جا از کلاس های کد اصلی به این جنبه ها نیاز داشته باشد به آنها رجوع می کند. [۸, ۱۱]

اگر بخواهیم مفاهیم جنبه گرایی را به صورت یک ایده مطرح کنیم می توان آن را به این گونه بیان کرد : روشی جهت مدیریت و پیاده سازی نیازهای غیر وظیفه مند در قالب جنبه ها به طوری که مستقل از کد اصلی باشند و تغییرات روی آنها یا روی کد اصلی تاثیر مستقیم و ناخواسته بر روی یک دیگر نداشته باشد و بعد از مرحله پیاده سازی به صورت اتوماتیک قبل از مرحله کامپایل توسط زبان برنامه نویسی جنبه گرا این دو کد با هم ادغام می شود این فرآیند توسط بافنده[۱۹] انجام می شود.

۳- کارهای انجام شده

همان طور که پیشتر اشاره گردید در حوزه معماری سازمانی و پارادایم سرویس گرا هیچ اقدامی جهت مدیریت نیازهای غیر عملیاتی صورت نگرفته است ولی می توان فرآیند مدیریت این نیازها را در حوزه مهندسی نرم افزار بیان کرد.

همان طور که در مستندات معماری سازمانی ذکر شده است یکی از کاربردهای محصولات معماری سازمانی در مستندسازی مهندسی نرم افزار است که توسط معمار نرم افزار[۲۰] این محصولات استفاده می شود.[۱, ۲, ۱۰] دو کاربرد این محصولات اول در تدوین و کشف نیازهای عملیاتی می باشد و دوم در جمع آوری، مدیریت و ایجاد توازن بین نیازهای غیر عملیاتی می باشد که در بیشتر اوقات از آنها با عنوان نیازهای کیفی نرم افزار نیز یاد می شود.[۱۳, ۱۴]

حال این سوال مطرح می شود که وقتی قرار است ما در تدوین نرم افزار برای یک سازمان به شناسایی نیازهای غیر عملیاتی بپردازیم چرا این کار را در فاز معماری سازمانی انجام ندهیم؟ قبل از ارائه پارادایم سرویس گرایی این دیدگاه کمی عجیب به نظر می رسید ولی حالا که بر اساس پارادایم سرویس گرا ما سازمان را در سه سطح می بینیم اگر شناسایی و مدیریت نیازها در این فاز انجام شود علاوه بر اینکه ما می توانیم دیدگاه سطح نرم افزار را در تاثیر بر این نیازها ببینیم بلکه می توانیم دیدگاه سطح زیر ساخت و کسب و کار را نیز در این تحلیل تاثیر دهیم که این کاملاً منطبق با پارادایم سرویس گرا و معماری نرم افزار می باشد. [۷, ۱۳, ۱۴]

هدف از این مقاله ارائه روشی جهت مدیریت و سازمان دهی نیازهای غیر وظیفه مند در پارادایم سرویس گرا است تا هم جمع آوری و مدیریت این نیازها بر اساس سه دیدگاه حاکم در پارادایم سرویس گرا (دیدگاه کسب و کار، نرم افزار و زیرساخت) باشد و هم محصول این فرآیند را بتوان در ساخت و تولید نرم افزار به عنوان نقشه نیازهای غیر عملیاتی سازمان به کاربرد.

۴- مدیریت نیازهای غیر عملیاتی

مدلی که در این مقاله به عنوان راهکاری جهت مدیریت نیازهای غیر عملیاتی مطرح شده است بر اساس ایده جنبه گرایی می باشد. این مدل برای مدیریت نیازهای غیر عملیاتی در پارادایم سرویس گرا می خواهد آنها را جهت مدیریت و کنترل راحت تر در یک جا متمرکز کند و سپس از طریق اجرای فرآیند نرمال سازی بین این نیازها ایجاد توازن کند و باعث حفظ یکپارچگی آن شود. مدل پیشنهادی این مقاله در چهار فاز اجرای می شود که عبارتند از :

۱- تشکیل ماتریس نیازهای غیر عملیاتی سازمان
۲- ایجاد گراف نیازهای غیر عملیاتی سازمان
۳- نرمال سازی گراف نیازهای غیر عملیاتی سازمان
۴- استخراج قوانین

۴-۱- ماتریس نیازهای غیر عملیاتی

مهمترین بخش در اجرای مدل مدیریت نیازهای غیر عملیاتی فراهم کردن ماتریس نیازهای غیر عملیاتی می باشد.

	رابطه	روش پیاده سازی	تاثیر	اثر تاثیر
نیاز غیر عملیاتی ۱	SOE		SOE	
	SOA		SOA	
	SOI		SOI	
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮
نیاز غیر عملیاتی N	SOE		SOE	
	SOA		SOA	
	SOI		SOI	

شکل 5– ماتریس نیازهای غیر عملیاتی

شکل (۵) ماتریس نیازهای غیر عملیاتی می باشد در این ماتریس لیستی از نیازهای غیر عملیاتی که در سازمان مورد نظر شناسایی شده تهیه می شود سپس رابطه بین این نیاز با سه حوزه کسب و کار و نرم افزار و زیرساخت معلوم می شود و روش دقیق پیاده سازی آن در آن حوزه نیز بیان می شود. توجه داشته باشید که لزوماً یک نیازمندی خاص نباید در هر سه حوزه وجود داشته باشد. علاوه بر این اطلاعات به بیان اینکه تاثیر این نیاز غیر عملیاتی روی چه حوزه هایی می باشد می پردازیم و همچنین اینکه این نیاز عملیاتی با این روش پیاده سازی روی کدام یک از نیازهای غیر عملیاتی دیگر تاثیرگذار می باشد را بیان می کنیم.

روش جمع آوری این اطلاعات به علت اینکه نیاز به شناخت کامل سازمان دارد کار بسیار دشواری می باشد و معمولاً باید از ترکیب روش هایی همانند مشاهده و تهیه پرسش نامه از پرسنل سازمان و گرفتن مشاوره از خبرگان و صاحبان نظران و مطالعه اسناد سازمان همانند مستندات برنامه ریزی استراتژیک و ... تهیه شود.

	رابطه	روش پیاده سازی	تاثیر	اثر تاثیر
Security			SOE	باعث قابلیت اتکا و اطمینان می شود
	SOA	با اعمال حقوق دسترسی و احراز هویت		
	SOI	طراحی و پیاده سازی IDS و Firewall		
Readability			SOE	باعث تسریع در کارها می شود
	SOA	با انتخاب زبان برنامه نویسی و الگوریتم های مناسب		
Accessibility	SOE	طراحی فرآیند های کسب کار بر اساس سیستم های مشتری محور		
	SOA	پیاده سازی بر اساس پلت فرم تحت وب	SOA	روی امنیت این لایه تاثیر بسزایی دارد
			SOI	زیرساخت باید کارایی و امنیت لازم را داشته باشد

شکل 6 –نمونه ای از ماتریس نیازهای غیر عملیاتی

شکل (۶) یک نمونه از سه نیاز غیر عملیاتی در یک سازمان باشد که در ماتریس نیازهای غیر عملیاتی وارد شده است. آن چیز که باید به آن توجه ویژه کرد این است که مطالب داخل این ماتریس برآیند کل فرآیند جمع آوری اطلاعات می باشد که بسیار مشابه همان کاری است که ما برای طراحی یک نرم افزار در مهندسی نرم افزار فاز تحلیل نیازها انجام

می دهیم با این تفاوت که ما در اینجا با هدف معماری سازمان این کار را انجام می دهیم و فقط دنبال نیازهای غیر عملیاتی در سطح کلان سازمان می باشیم.

۴-۲- گراف نیازهای غیر عملیاتی

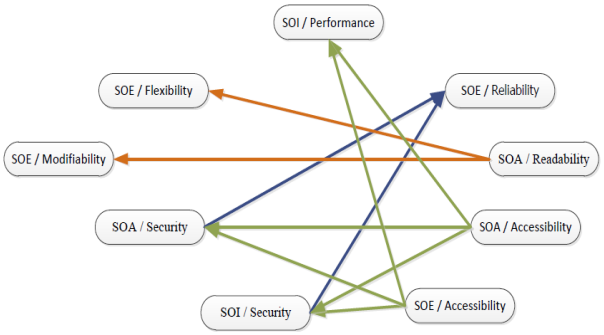
جهت ایجاد توازن بین نیازمندی ها و کشف تضاد[۲۱] بین نیازمندی ها و داشتن یک دید کلان به کل نیازمندی ها و حتی ارزیابی این نیازمندی ها می توان ماتریس نیازهای غیر عملیاتی را به یک گراف جهت دار و وزن دار تبدیل کرد. این کار در دو فاز انجام می شود :

۱- ایجاد گراف بدون وزن جهت دار
۲- تخصیص وزن به گراف نیازها

۴-۲-۱- ایجاد گراف بدون وزن جهت دار

شکل 7 –یک گره از گراف بدون وزن جهت دار

جهت ایجاد یک گراف بدون وزن جهت دار ما هر گره در گراف را به صورت ترکیبی از حوزه معماری و نیازمندی غیر عملیاتی فرض می کنیم. شکل (۷) گره مربوط به امنیت در حوزه معماری سرویس گرا می باشد. یال ها در این گراف از نیاز موثر به نیاز تحت تاثیر کشیده می شوند.



شکل 8 – گراف جهت دار بدون وزن

شکل (۸) گراف بدون وزن جهت دار معادل جدول شکل (۶) می باشد.

۴-۲-۲- تخصیص وزن به گراف نیازهای غیرعملیاتی

تخصیص وزن به نیازها در راستای ایجاد توازن بین نیازهای غیر عملیاتی صورت می گیرد این کار به خاطر اهمیت بالایی که در نتیجه فرآیند دارد توسط معمار سازمان انجام می شود. عموماً یا نیازها هم راستای هم می باشند و یا مغایر با هم می باشند و یا هیچ رابطه خاصی با هم ندارند معمار سازمان این اطلاعات را می تواند از یال های گراف در یابد و همچنین می تواند وابستگی ها و تقدم و تاخر نیازها را از روی این گراف تشخیص دهد. جهت تخصیص وزن معمار تمامی اسناد تهیه شده جهت تدوین نیازهای را مطالعه کرده و بر اساس نوع سازمان به ارتباط بین هر نیاز با نیاز دیگر (یال ها) در گراف از صفر به عنوان کمترین وزن تا چهار به عنوان بیشترین وزن را تخصیص می دهید و علاوه بر این کار در صورتی

که ارتباط در راستای آن نیاز مندی باشد به آن علامت مثبت در صورتی که ارتباط مغایر با آن نیازمندی و در جهت کاهش تاثیر آن نیازمندی باشد به آن منفی تخصیص داده می شود. از آنجا که ما سازمان را در سه سطح می بینیم معمار باید به هر نیاز در هر سطح، منحصر به فردی نگاه کند و بر اساس وزن های یال های ورودی گره مبدا به آن یال وزن تخصیص دهد. یکی از بهترین روش هایی که می تواند به معمار سازمان در تصمیم گیری جهت تخصیص وزن به یال ها کمک کند تعداد یال های ورودی گره مبدا و وزن آنها می باشد و راستای اهداف سازمان و ارتباطی که این یال ایجاد می کند.

۴-۳- نرمال سازی گراف نیازهای غیرعملیاتی

هدف از نرمال سازی گراف نیازهای غیرعملیاتی در سه موضوع است اول در ساده سازی گراف و توازن بین نیازها دوم در به حداقل رساندن یال ها با وزن های صفر و یک (ما در این فاز مقدار وزن ها را بدون علامت مورد مطالعه قرار می دهیم) و سوم به حداقل رساندن تاثیر گره هایی که یال هایی با وزن بالا دارند ولی علامتشان یکی نیست. این کار را می توان با تکرار فازهای چهارگانه زیر انجام داد :

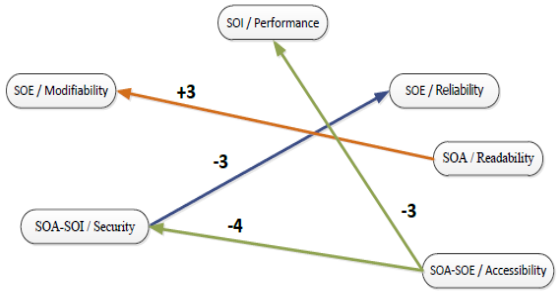
۱- حذف یال هایی که گره های مبدا آنها ورودی های کمی دارند یا وزن یال های ورودی آنها کم است و یا خود یال دارای وزن کمی می باشد (برای مثال صفر) یا ترکیبی از موراد ذکر شده. قابل ذکر است که سنجش کمی وزن های یک یال بدون علامت صورت می گیرد یعنی -۴ و +۴ دارای ارزش یک سان و در خلاف جهت هم می باشد.

۲- حذف یال هایی که می توان از نظر معنایی[۲۲] اهداف آنها را از طریق یال های موجود برآورده کرد یا به عبارتی ارتباطات بدیهی که در ذات سایر ارتباطات نیازها پیاده سازی خواهد شد.

۳- از طریق برگزاری جلسات و مشاوره با خبرگان در زمینه‌های مورد مناقشه و در صورتی که ارتباط بین نیازهای مورد بحث یا وزن یال مورد اختلاف در دو حوزه مختلف باشد می توان با تکنیک هایی مانند طوفان ذهنی بین خبرگان آن حوزه ها به نتیجه قابل قبولی رسید.

شکل ۹ –گره ترکیبی از دو لایه

۴- ترکیب یک نیازمندی در لایه های مختلف در صورتی که روابط آن نیازمندی در لایه های تحت تاثیر یکی باشد برای مثال در شکل (۸) نیازمندی Accessibility در لایه های نرم افزار و کسب و کار این شرایط را دارد. آنگاه این گره را مطابق شکل (۹) نشان می دهیم.



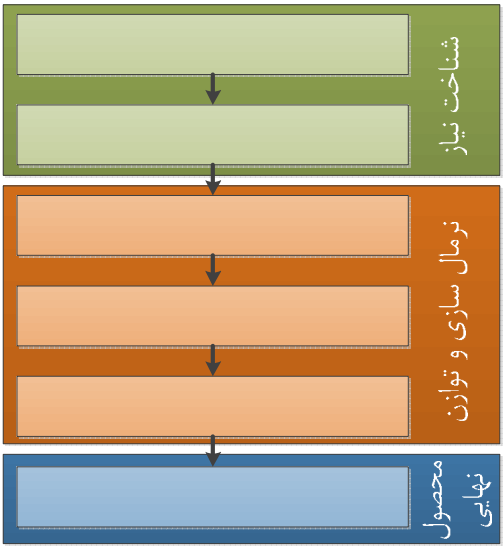
شکل ۱۰ – گراف جهت دار وزن دار نرمال شده

شکل (۱۰) همان گراف شکل (۸) بعد از تخصیص وزن و نرمال سازی می باشد. همانطور که می بینید اگر این دو مرحله که از نظر کاری وقت گیر می باشند و نیاز به دقت بسزایی دارند درست انجام شود تاثیر آن بر روی گراف کاملاً مشهود است.

۴-۴- استخراج قوانین

شکل ۱۱ – گره در گراف نیازهای غیرعملیاتی

اگر بخواهیم استخراج قوانین را در یک جمله تعریف کنیم می توان آن را تحلیل ورودی و خروجی گره های گراف دانست. محصول استخراج قوانین فصل آخر مستند نیازهای غیر عملیاتی سازمان می باشد. برای مثل شکل (۱۱) یک گره گراف را نشان می دهد و هر یال ورودی یا خروجی یک رابطه این نیازمندی با سایر نیازمندی ها را نشان می دهند. در استخراج قوانین ما ابتدا گره های مربوط به یک نیازمندی خاص را در نظر گرفته و سپس بر اساس ورودی و خروجی و وزن یال های گره ها وضعیت نیازمندی مورد نظر را در هر یک از سه حوزه سازمان تحلیل می کنیم. باید توجه داشته باشیم که در این فاز ما دیگر دید خرد به موضوع نداریم و هر نیازمندی را به صورت کلان در هر سه حوزه می بینیم و تحلیل می کنیم. برای مثال اگر در این مورد ما در مورد امنیت صحبت می کنیم باید هر سه گره مربوط به امنیت در سطوح کسب و کار، نرم افزار و زیرساخت مورد مطالعه و تحلیل قرار گیرند. خروجی این بخش در اصل همان خصوصیات[۳۳] سازمان مورد مطالعه ما می باشد که از جمع آوری، مطالعه و ایجاد توازن بین نیازهای غیر عملیاتی ما حاصل شده است. مجموع دو فاز نرمال سازی و استخراج قوانین معادل کاری است که در جنبه گرایی بافنده انجام می دهد. شکل (۱۲) مدل پیشنهادی مطرح شده در مقاله می باشد.



شکل ۱۲ – مدل پیشنهادی مقاله

۵- ارزیابی مدل

جهت ارزیابی این مدل بر اساس روشی که داودی و همکاران [۱۵, ۱۶] در تحلیل ویژگی های کیفی معماری سازمانی بر اساس میزان برازندگی نیازهای غیر عملیاتی ارائه داده اند عمل می کنیم. فرض کنید از خبرگان مختلف در مورد اولویت بندی نیازهای غیر عملیاتی سازمان نظر سنجی کرده ایم. iPEAV[۲۴] ماتریس اولویت بندی هر یک از خبرگان خواهد بود که در رابطه (۱) نشان داده شده است:

$$iPEAV_k = \begin{bmatrix} P_1 \\ \vdots \\ P_n \end{bmatrix} \quad (1)$$

حال فرض کنید تعداد خبرگان ما $k = 1,2,\ldots,K$ باشد و با استفاده از بکارگیری تکنیک میانگین هندسی iPQA[۲۵] بردار ویژه هر یک از ماتریس ها باشد که اولویت بندی نیازهای غیر عملیاتی سازمان نسبت به یکدیگر از نظر یک خبره را مشخص می کند. [15, 16] در نهایت بر اساس رابطه (۲) PQA_i ها را محاسبه خواهیم کرد. در رابطه PQA_i اندیس i نشان دهنده شماره نیاز غیر عملیاتی i ام می باشد که با فرض اینکه ما m تا نیاز غیر عملیاتی داریم $i = 1,2,\ldots,m$ خواهد بود. میانگین هندسی PQA_i را PQA می نامیم.

$$PQA_i = \left(\prod_{k=1}^{k} iPQA_{ik} \right)^{1/k} \quad (2)$$

حال فرض کنید $WCQA_x$[۲۶] مطلوب ترین وزن نیاز مندی x و $iWCQA_x$ وزنی باشد که هر یک از خبرگان به آن نیاز مندی تخصیص می دهند و با فرض بر اینکه ما m تا نیازمندی داریم رابطه (۳) وزن نرمال هر نیاز مندی خواهد بود.

$$NW_x = \frac{WCQA_i}{\sum_{i=1}^{m} WCQA_i} \quad (3)$$

حال ما بر اساس رابطه (۴) ماتریس PCQA را تشکیل می دهیم:

$$PCQA_{QA_x} = (NW_1 \quad NW_2 \quad \ldots \quad NW_m) \begin{bmatrix} p_{11} & \cdots & p_{1k} \\ \vdots & \ddots & \vdots \\ p_{m1} & \cdots & p_{mk} \end{bmatrix} \quad (4)$$

ماتریس مفروض در رابطه (۴) از کنار هم قرار دادن ماتریس های iPEAV تشکیل شده است که در آن k تعداد خبرگان و m تعداد نیازهای غیر عملیاتی سازمان خواهد بود. [۱۵] داودی و همکاران ثابت کرده اند که بر اساس رابطه (۵) می توان سطح برازندگی نیازهای غیر عملیاتی در سازمان را محاسبه کرد.

$$LOS_j = \sum_i (PQA_i * PCQA_{i,j}) \quad (5)$$

در رابطه (۵) j نمایانگر j امین انتخاب از کل حالت های گزینش شده از نظرات خبراگان خواهد بود هر چه LOC بیشتر باشد به معنی این است که گزینش ما بهتر و بهینه[۲۷]تر خواهد بود.

راه حلی که این مقاله ارائه داده است بر این اساس است که روش های بهینه سازی و نرمال سازی را طوری تغییر دهیم که بر روی پارامتر های موثر بر LOC تاثیر مثبت قرار دهد. جهت این کار روش پیشنهادی این مقاله بر روی رابطه میان نیازهای غیر عملیاتی (یال های گراف) کار کرده و از آنجایی که این رابطه ها، اثر مستقیم بر PQA_i خواهند داشت با بهینه سازی یال ها در فاز نرمال سازی مقدار PQA_i را افزایش می دهیم و با انتخاب وزن ها در فاز تخصیص وزن به گراف توسط معمار سازمان به واسطه پالایش نظرات خبرگان NW_x را افزایش می دهیم و این باعث بالا رفتن مقادیر پارامتر ها

در ماتریس PCQA خواهد شد و در نهایت LOC یا همان سطح برازندگی نیازهای غیر عملیاتی در سازمان افزایش یافته و بهینه خواهد شد. در بخش بعد به جمع بندی و بررسی نتایج و چالش های طرح پیشنهادی مقاله می پردازیم.

۶- نتیجه گیری

در این مقاله، ما بر اساس ایده جنبه گرایی و بسط آن به حوزه پارادایم سرویس گرا یک مدل جهت مدیریت نیازهای غیر وظیفه مند در یک سازمان ارائه دادیم. این مقاله دو هدف کلی را دنبال می کند اول اینکه یک روش جهت مدیریت نیازهای غیر وظیفه مند با دیدگاه پارادایم سرویس گرا ارائه دهد و دوم اینکه خروجی نهایی این فرآیند را می توان در تهیه نرم افزارهای سازمان توسط معمار نرم افزار استفاده کرد. از این رو ما با سه دیدگاه به بررسی نتایج طرح پیشنهادی مقاله می پردازیم.

از دیدگاه یکپارچه سازی سازمان یکی از مهمترین بحث ها، در طراحی نیازهای غیرعملیاتی روشی است که از طریق آن بتوان وابستگی ها و تداخل ها چک شود و همچنین توازن بین نیازها، درست برقرار شود. مدل پیشنهادی این مقاله با تبدیل نیازها و رابطه بین آنها در سه حوزه سازمان به گراف نیازمندی، باعث می شود که اولاً معمار یک دیدگاه جامع و یکپارچه به نیازها داشته باشد و ثانیاً در فاز نرمال سازی بتواند با حفظ این یکپارچگی تداخل ها را راحت تر حل کند و مشکلات را راحت شناسایی کند علاوه بر این چون در فاز استخراج قوانین ما همزمان یک نیازمندی را به صورت یکپارچه در سه حوزه سازمان بررسی و تحلیل می کنیم این یکپارچگی هم در سطح ارتباط این نیازمندی با نیازمندی های دیگر حفظ می شود و هم در ارتباط با خود نیازمندی در لایه های مختلف سازمان حفظ خواهد شد.

از دیدگاه مدل سازی نیازهای یک سازمان، مهمترین چالش پیش روی هر معمار مدل سازی مستندات است تا قبل از اجرای بتواند آنها را با واقعیت تطبیق دهد.[۱۴] مدل سازی هر چقدر بر پایه ریاضیات باشد می تواند قابلیت اطمینان بیشتری برای معمار نرم افزا فراهم کند. از این رو روش های رسمی[۲۸] و نیمه رسمی[۲۹] در سال های اخیر برای مدل سازی نرم افزار توسعه پیدا کرده اند که قالباً پیشتوانه ریاضی دارند. روش پیشنهادی این پروژه به خاطر تبدیل نیازمندی های و روابط آنها به گراف یک راه آسان جهت تبدیل نیازمندی های غیر عملیاتی به روش های مدل سازی همچون شبکه های بیضی[۳۰] یا شبکه های پتری[۳۱] یا بررسی مدل[۳۲] فراهم می کند. چون خود گراف نیز بر پشتوانه ریاضی دارد پس دیگر نیاز به استفاده از هیچ واسطی جهت تبدیل آن به یک روش مدل سازی نیست و می توان با تعاریف ریاضی بر اساس نیاز آن را به یک روش مدل سازی تبدیل کرد.

از دیدگاه معمار نرم افزار سازمان، همانطور که پیشتر نیز به آن اشاره کردیم مرحله بعد از معماری سازمانی توسعه نرم افزار های سازمان می باشد و تجزیه و تحلیل مستندات و تایید اینکه، آیا مستندات تهیه شده برای نرم افزار کامل و درست می باشد بر عهده معمار نرم افزار می باشد و همچنین ذکر شد که معمولاً در این فاز بحث بر روی کیفیت نرم افزار است که با بررسی نیازهای غیر عملیاتی این کار صورت می گیرد و عملیاتی مشابه با آن چیز که ما در این مقاله در فرآیند اجرای پارادایم سرویس گرا انجام دادیم صورت می گیرد. در اصل این مقاله با انتقال این فرآیند به معماری سازمانی دو هدف عمده را دنبال می کند اول اینکه اجرای این فرآیند تحت تاثیر هر سه حوزه سازمان (کسب و کار، نرم افزار و زیرساخت) قرار گیرد و دوم اینکه محصول این فرآیند بتواند نیازهای معمار نرم افزار را پوشش دهد که هدف بخش اول با تشکیل گراف نیازها برآورده می شود و بخش دوم با استخراج قوانین و تولید مستند نیازهای غیر عملیاتی سازمان برآورده می شود.

با تمام مزایا بیان شده در این بخش نباید چالش ها و مشکلات پیش رو را فراموش کرد اولین چالش طراحی و ایجاد این مدل عدم وجود مدل مشابه به این مدل در پارادایم سرویس گرا می باشد ولی همان طور که دربخش های قبل به دلایل آن اشاره شد ایجاد این مدل در فاز معماری سازمانی باعث کنترل بیشتر بر روی نیازهای غیر عملیاتی یک سازمان می شود ولی چون این مدل در مراحل تحقیق می باشد فاقد تجربه عملی در یک سازمان می باشد و تنها دلیل اعتبار آن

دلایل تئوری اجرای این مدل می باشد. چالش بعدی در این مدل مسئله ایجاد و نرمال سازی گراف می باشد اگر بخواهیم اجرای این مدل را به صورت دستی برای یک سازمان انجام دهیم به علت بالا بودن نیازها و بررسی هر نیاز در سه لایه با تعداد بالایی از گره ها و روابط سر و کار داریم و این باعث کندی کار و احتمال بروز خطا می شود و از طرفی نرمال سازی چنین گرافی کار بسیار دشواری است ولی چون مدل پیشنهادی این مقاله بر مبنای یک الگریتم مدون بنا شده می توان یک نرم افزار را به راحتی جهت مدیریت این مدل طراحی و توسعه داد. با توسعه یک نرم افزار جهت اجرای این مدل خیلی از بخش های تشکیل گراف و نرمال سازی آن می تواند به راحتی به صورت اتوماتیک انجام شود.

مراجع

1. Zachman, John A., "A Framework for Information Systems Architecture", IBM Systems Journal, Vol. 26, No. 3, 1987.

2. Zachman, J.A., Sowa, J.F. 1992, "Extending and Formalizing the Framework for Information Systems Architecture", IBM Systems Journal, vol. 31, no. 3.

3. Oasis: SOA Adoption Blueprint, 2006, Available: http://www.oasis-open.org/committees/download.php/17616/06-04-00002.000.doc

4. Linthicum, D. 2004, What Level Is Your SOA? Choose for what you need and maybe a little better, Available: http://webservices.sys-con.com/read/47277.htm

5. Borges, B., Holley, K. and Arsanjani, A. 2004, Service-oriented Architecture, Available:http://searchwebservices.techtarget.com/originalContent/0,289142,sid26_gci1006206,00.html?topic=299037

6. CIO Council, "A Practical Guide to Federal Enterprise Architecture Version 1.0", CIO, 2001.

7. CIO Council, "A Practical Guide to Federal Service Oriented Architecture", CIO, 2008.

8. Kiczales, Gregor; John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin, "Aspect-Oriented Programming", 1997, Proceedings of the European Conference on Object-Oriented Programming, vol.1241. pp. 220–242. The paper generally considered to be the authoritative reference for AOP.

9. Filman, Robert E.; Tzilla Elrad, Siobhán Clarke, and Mehmet Aksit, "Aspect-Oriented Software Development", 2004, ISBN 0-321-21976-7.

10. CIO Council, "Federal Enterprise Architecture Framework Version 1.1", CIO, Sep 1999.

11. Joseph D. Gradecki, Nicholas Lesiecki, "Mastering AspectJ: Aspect-Oriented Programming in Java", IEEE Comput Soc, pp. 199-202, 2002.

12. IBM Research Team, "Concern Space", 2008, Available: http://www.research.ibm.com/hyperspace/ConcernSpaces.htm.

13. Boehm B., Hoh, "Identifying Quality Requirement Conflicts", IEEE Software, pp.740-745, 1996.

14. Kitchenham B., Pfleeger S. L., "SOFTWARE QUALITY: THE ELUSIVE TARGET", IEEE Software, Vol. 13, No. 1, pp. 12-21, Jan 1996.

15. Davoudi M. R., "A New Framework for EA Quality Attribute Analysis", Master Thesis in Islamic Azad University Science and Research Branch, 2010

16. Davoudi M. R., Shams F. A., Badie K., "An AHP-based Approach towards Enterprise Architecture Analysis based on Enterprise Architecture Quality Attributes", Knowledge and Information System Journal, Springer, DOI 10.1007/s10115-010-0312-1.

زیرنویس ها

[1] Enterprise Architecture
[2] Loosely Coupled
[3] Modularity
[4] Encapsulation
[5] Reusable
[6] Service Orientation
[7] Service Oriented Enterprise Architecture (SOEA)
[8] Service Oriented Enterprise (SOE)
[9] Service Oriented Architecture (SOA)
[10] Service Oriented Infrastructure (SOI)
[11] Functionality
[12] Flexibility
[13] Continuous Availability
[14] Correct Operation
[15] Functional Requirement
[16] Non-Functional Requirement
[17] Aspect
[18] Code Structure
[19] Weaver
[20] Software Architecture
[21] Conflict
[22] Semantic
[23] Specification
[24] Individual Prioritized list of EA View
[25] Individual Prioritized list of Quality Attributes
[26] Weight list of sub-Criteria of each Quality Attribute
[27] Optimize
[28] Formal
[29] Semi-Formal
[30] Bayesian Network
[31] Petri Net
[32] Model Checking