بررسی تحلیلی روش بهره‌گیری از میان‌افزار در معماری مبتنی بر سرویس

جواد احمدی
دانشجوی کارشناسی ارشد
رشته کامپیوتر نرم افزار
دانشگاه آزاد اسلامی واحد قزوین
ahmadi_javad@qazviniau.ac.ir

سید امیرهادی مینوفام
دانشجوی کارشناسی ارشد
رشته کامپیوتر نرم افزار
دانشگاه آزاد اسلامی واحد قزوین
minoofam@qazviniau.ac.ir

بهنام آزادی کناری
دانشجوی کارشناسی ارشد
رشته کامپیوتر نرم افزار
دانشگاه آزاد اسلامی واحد قزوین
behnam.azadi@gmail.com

چکیده – توسعه سریع فناوری‌های اطلاعاتی، نیازمندی‌های سازمان‌ها را به سرعت افزایش داده است. استفاده از معماری مناسب در توسعه و پیاده‌سازی نرم افزارها، راهکار مفیدی برای غلبه بر پیچیدگی‌های روز افزون سیستم‌های اطلاعاتی است. معماری مبتنی بر سرویس، رویکرد جدیدی در دنیای نرم افزار است که با رفع محدودیت‌ها و نواقص معماری‌های پیشین، به عنوان بهترین گزینه در این زمینه محسوب می‌گردد. معماری مبتنی بر سرویس، رهیافتی برای ساخت سیستم‌های توزیع شده است که کارکردهای نرم‌افزاری را در قالب سرویس ارائه می‌کند. ایجاد معماری مبتنی بر سرویس، به مجموعه‌ای از ارتباطات قابل توزیع و یک مسیر ارتباطی سریع و یکپارچه نیازمند است. میان‌افزار به‌عنوان لایه واسط در معماری نرم‌افزار، توسعه‌دهندگان را قادر می‌سازد کار خود را بدون اطلاع از پیچیدگی‌های لایه‌های زیرین انجام دهند. در این مقاله با ارائه تلفیق جدیدی از این دو واقعیت، به ساختاری نوین در طراحی می‌رسیم. این طراحی، افزایش کارایی و شفافیت مکانی و منطقی را به‌دنبال خواهد داشت و موجبات امنیت چند لایه را ممکن می‌سازد.

کلید واژه – معماری مبتنی بر سرویس، میان‌افزار، افزایش کارایی، امنیت چند لایه

۱ – مقدمه

معماری مبتنی بر سرویس مفهوم جدیدی نیست و از دهه ۹۰ وجود داشته است، ولی آنچه جدید است توانایی اجرا و عینیت بخشیدن به آن است که به کمک ابزارها و پروتکل‌های مربوطه میسر شده است. این معماری توسط دو شرکت IBM, Microsoft به‌وجود آمد، که هر دو شرکت طی سالهای اخیر از حامیان اصلی سرویس‌های وب و عامل بسیاری از ابداعات جدید در این حیطه بوده‌اند. سیستم‌های اطلاعاتی، به سرعت در حال رشد هستند و سازمان‌ها نیازمند پاسخگویی سریع به نیازمندی‌های جدید کسب‌وکار می‌باشند. این در حالی است که معماری‌های نرم‌افزاری موجود به حد نهایی قابلیت‌های خود رسیده‌اند. معماری مبتنی بر سرویس قدم تکاملی بعدی برای کمک به سازمان‌ها جهت مدیریت چالش‌های پیچیده است. معماری مبتنی بر سرویس، حالت تکامل یافته معماری مبتنی بر اجزا، طراحی مبتنی بر واسطه (شیءگرا) و سیستم‌های توزیع شده است. سیستم توزیع شده، تعمیمی از یک معماری مبتنی بر اجزا است و به اجزایی که در موقعیت‌های فیزیکی مختلف وجود دارند، اشاره می‌کند. مهم‌ترین مزیت معماری مبتنی بر اجزا، سهولت در استفاده مجدد و تغییر هدف اجزای خاص و سهولت در امر نگهداری سیستم است. استفاده مجدد و تغییر هدف، مهم‌ترین محرک‌های کسب‌وکار برای استفاده از این نوع معماری در دهه ۹۰ میلادی بوده است. میان‌افزار نیز به‌عنوان یک نرم‌افزار خاص، مانند یک مبدل و لایه انتقال عمل می‌کند و با

پنهان‌سازی پیچیدگی‌های لایه‌های زیرین، پیاده‌ساز را از درگیر شدن با این پیچیدگی‌ها بی‌نیاز می‌سازد. امروزه میان‌افزارهای بسیاری به منظور اجرا در ساختارهای متفاوت تهیه می‌گردند. به عنوان مثال، میان‌افزار پیغام، با فراهم کردن واسط مناسب، امکان تبادل پیغام میان دو برنامه کاربردی را فراهم می‌سازد. در معماری مبتنی بر سرویس، کلیه برنامه‌های کاربردی به منظور دسترسی به سرویس مورد نیاز، از سرویسی بهره می‌گیرند که نقش واسطه را بازی می‌کند. در صورت تعامل برنامه‌های کاربردی با میان‌افزار، تعامل با سرویس واسطه می‌تواند بر عهده میان‌افزار باشد. در این صورت خود میان‌افزار، نیازمند اطلاع دقیق از سرویس‌ها و محدودیت‌های آنها است. در این مقاله چارچوبی برای میان‌افزار ارائه می‌کنیم تا ضمن ارتباط با سرویس واسطه، تأمین‌کننده شفافیت سرویس باشد. همچنین ساختار ارائه شده انعطاف لازم برای تغییرات مورد نیاز را داراست و می‌تواند بستر مناسبی برای توسعه‌های بعدی باشد.

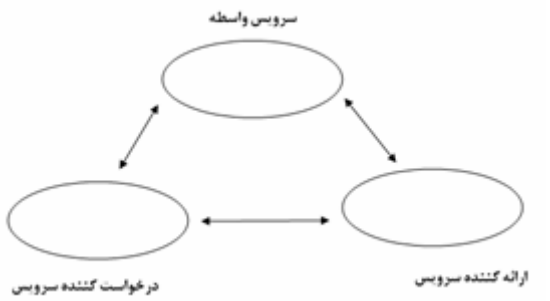
ساختار ادامه مقاله بدین شرح است. در بخش۲، به مهم‌ترین تعاریف معماری مبتنی بر سرویس اشاره می‌کنیم. در بخش ۳، طراحی مبتنی بر سرویس را ارائه می‌نماییم. در بخش ۴، ساختار متاداده سرویس را بررسی خواهیم نمود. در بخش ۵، کنترل دنباله دسترسی را بیان می‌کنیم. در بخش ۶، پیاده‌سازی میان‌افزار را شرح می‌دهیم و سرانجام در بخش ۷، به نتیجه‌گیری می‌پردازیم.

۲-تعریف معماری مبتنی بر سرویس

برای معماری مبتنی بر سرویس، تعاریف مختلفی ارائه شده که هر کدام از نگاهی به تبیین خصوصیات آن پرداخته‌اند. برای درک بهتر این مفهوم و آگاهی از کلیه برداشت‌ها و نگاه‌های موجود، در این قسمت به چند تعریف از معماری مبتنی بر سرویس، اشاره می‌کنیم. ۱- یک چارچوب راهبردی از فناوری که به تمام سیستم‌های داخل و خارج اجازه ارائه یا دریافت سرویس‌های خوش‌تعریف را می‌دهد [۱][۸]. ۲- روشی برای طراحی و پیاده‌سازی نرم‌افزارهای گسترده سازمانی بوسیله ارتباط میان سرویس‌هایی که دارای خواص اتصال سست، دانه درشتی و قابلیت استفاده مجدّد هستند [۲]. ۳- سبکی از معماری که از اتصال سست سرویس‌ها جهت انعطاف‌پذیری و تعامل‌پذیری حرفه و بصورت مستقل از فناوری پشتیبانی می‌کند و از ترکیب سرویس‌های مبتنی بر حرفه تشکیل شده است. این سرویس‌ها، انعطاف‌پذیری و همچنین پیکربندی پویا را برای فرآیندها محقق می‌سازند [۳][۱۳]. ۴- چارچوبی وسیع و استاندارد که سرویس‌ها در آن ساخته، استقرار و مدیریت می‌شوند و هدف آن، افزایش چابکی زیرساخت‌های فناوری اطلاعات در جهت واکنش سریع به تغییرات در نیازهای کسب و کار می‌باشد [۴][۵].

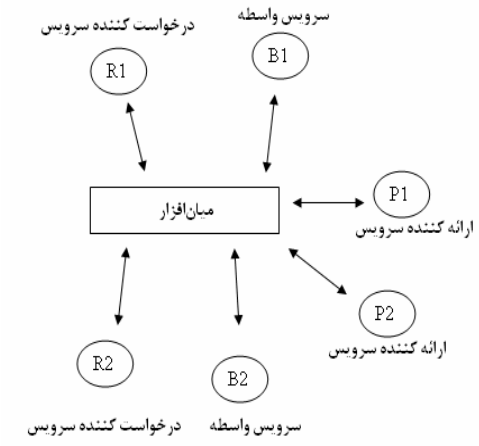
۳ – طراحی مبتنی بر سرویس

معماری مبتنی بر سرویس، سرویس‌های تشکیل‌دهنده سیستم را تعریف می‌کند و تعامل بین سرویس‌ها جهت ارائه رفتار مشخص را توصیف می‌نماید و در نهایت سرویس‌ها را به یک یا چند پیاده‌سازی در فناوری‌های خاص تصویر می‌کند. در طراحی مبتنی بر سرویس، هر سرویس برای تعامل با سرویس‌های دیگر، باید به شکل استاندارد توسعه داده شده باشد. یک سرویس می‌تواند به عنوان یک نقطه پایانی نیز تلقی گردد. به عنوان مثالی از این تعریف، به دستگاه‌های خودپرداز اشاره می‌کنیم که برای ارائه سرویس‌های مالی، در موقعیت‌های مختلف جغرافیایی نصب می‌گردند. دستگاه خودپرداز به عنوان یک نقطه پایانی در تعامل با کاربران بوده و توانایی لازم برای ارتباط با سرویس‌های دیگر برای انجام وظایف مربوطه را دارا است. هر سرویس، امکان دسترسی به مجموعه خوش تعریفی از عملیات را می‌دهد. سیستم به عنوان یک کلّ، بصورت مجموعه‌ای از تعاملات میان این سرویس‌ها طراحی می‌شود. در شکل(۱) یک مدل مفهومی برای معماری مبتنی بر سرویس ارائه شده است.



شکل (۱). مدل مفهومی معماری مبتنی بر سرویس[۹]

ارائه کننده سرویس، سرویس‌هایی را برای عموم ارائه می‌کند و این سرویس‌ها را برای در دسترس قرار گرفتن، در سرویس واسطه ثبت می‌کند. سرویس واسطه، سرویسی شامل اطلاعات سرویس‌ها و خدمات آنها است. سرویس واسطه دارای مخزن بزرگی است که دربردارنده این اطلاعات می‌باشد. درخواست‌کننده سرویس برای در اختیار داشتن یک سرویس خاص، به سرویس واسطه مراجعه می‌کند و با ارائه نام یا کلیدواژه خاصی باعث جستجو در این مخزن می‌شود. نتیجه این جستجو در سرویس واسطه، آدرسی است که به ارائه‌کننده این سرویس خاص ختم می‌شود [۱۴]. یک سرویس ساده می‌تواند بوسیله یک سرویس‌دهنده ارائه گردد؛ در حالی که یک سرویس پیچیده ممکن است از تعامل چند سرویس‌دهنده حاصل شود [۱۲][۱۵]. در این صورت است که دنباله دسترسی به سرویس‌دهنده‌های لازم برای دسترسی به یک سرویس خاص، می‌تواند محل چالش باشد. زیرا در صورتی که به دلایل طراحی یا سیاست‌های سازمانی، دنباله دسترسی به سرویس‌هنده‌ها تغییر کند، درخواست‌کننده این سرویس دچار مشکل خواهد شد. به عنوان مثال، فرض کنید که درخواست‌کنندهٔ R1 برای دراختیار گرفتن سرویس خاصی، سراغ سرویس‌دهنده‌های P1 و P2 می‌رود. حال اگر این توالی به هر دلیلی معکوس شود یا P3 نیز وارد این توالی شود، تکلیف سرویس‌گیرنده چیست؟ آیا سرویس‌گیرنده باید دارای دو کد جدا برای دراختیار گرفتن این سرویس خاص باشد؟ برای حل مشکلاتی از این قبیل، میان‌افزاری را پیشنهاد می‌کنیم که وظیفه اصلی آن تعامل با واسطه‌های سرویس و شفاف‌سازی سرویس است. شکل (۲)، نشان‌دهنده حالت ساده‌ای از این پیشنهاد است.



شکل (۲). مدل ساده‌ای از میان‌افزار پیشنهادی

میان‌افزاری که قادر باشد مشکلاتی از این دست را حل کند، باید تأمین‌کنندهٔ قابلیت‌هایی مانند: ارائه دنباله‌های تغییرپذیر از سرویس‌ها و ایجاد شفافیت سرویس برای برنامه‌های کاربردی باشد. در بخش ۴، به منظور برآورده ساختن چنین قابلیت‌هایی، برای هر سرویس یک متاداده پیشنهاد می‌کنیم. از ساختار XML برای ساخت این متاداده بهره گرفته‌ایم. علت اصلی استفاده از آن در قابلیت‌های XML نهفته است که عبارتند از: ۱- بوسیله W3C پشتیبانی می‌شود [۵][۱۰]. ۲- دارای ساختار خوش‌تعریفی است که توسط معروف‌ترین زیرساخت‌های تولید برنامه‌های توزیع‌شده یعنی Sun Java و Net Framework. پشتیبانی می‌شود [۶][۱۱]. ۳- از قوانین ارتباطی HTTP بهره می‌گیرد که یکی از معروف‌ترین قوانین ارتباطی است.

۴ – ساختارمتاداده سرویس

در این بخش، ساختاری را برای متاداده پیشنهاد می‌کنیم که می‌تواند میان‌افزار را در تعامل هرچه بهتر با سرویس‌ها وایجاد شفافیت سرویس یاری دهد. در صورت تحقق چنین قابلیتی، حتی تغییر در توالی دسترسی به یک سرویس خاص، نیازمند تغییر دربرنامه کاربردی نیست. در این طراحی، هر سرویس

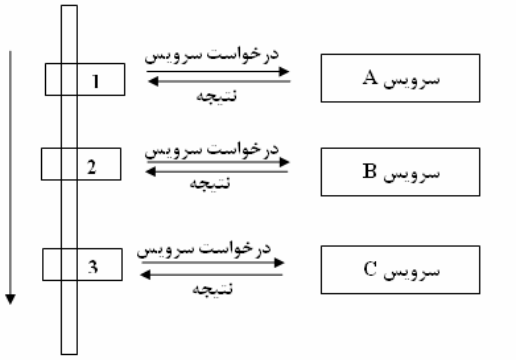
به معرفی قابلیت‌های خود در قالب واسط‌های خوش تعریف می‌پردازد. همچنین سایر ویژگی‌هایی را که می‌تواند در نحوه استفاده از این سرویس مهم باشد، در فرمت XML بیان می‌کند و این متاداده را در اختیار سرویس واسط قرار می‌دهد. اکنون، مخزن سرویس واسط دارای اطلاعات مفیدی راجع به هر سرویس است که در قالب تگ‌های XML از هم جدا شده‌اند. از آنجایی که هر سرویس می‌تواند خود به تنهایی ارائه‌کننده چندین سرویس باشد، در این متاداده، هر سرویس می‌تواند مجموعه‌ای از سرویس‌ها را در قالب تگ‌های XML در قسمت ProvidingService ارائه کند. به عنوان مثال، اگر دستگاه ATM به عنوان یک سرویس تلقی شود، می‌تواند ارائه‌دهنده چندین سرویس مالی برای کاربران باشد. نکته مهم در این متاداده که برروی آن تأکید داریم ControlSequence است. ارائه‌دهندهٔ سرویس، کلیه ملاحظاتی که مورد نیاز استفاده‌کننده است را در این قسمت قرار می‌دهد. هرگونه تغییر در سیاست‌های استفاده و الزامات سازمانی می‌تواند در این قسمت منعکس گردد. متاداده در هر سرویس، ایجاد شده و برای ذخیره در مخزن در اختیار سرویس واسط قرار می‌گیرد. سپس میان‌افزار با اتصال به سرویس واسط، اقدام به درخواست سرویس‌های مورد نیاز برنامه‌های کاربردی می‌کند. بدین ترتیب، برنامه‌های کاربردی، به تعامل با سرویس واسط نیازی نخواهند داشت. شکل(۳) نمایشی از متاداده یک سرویس می‌باشد.

```
<Services>
Name:  Name
Description: Description
    <ProvidingService>
        <Service>
            Name: Name
            Description: Description
            Interface: Interface
            <UserInterFacePolicies>
            Name: Name
            Shortcut: Shortcut
            Validation Message: Validation
        Message
            <! -- Collection of policy -->
            </UserInterFacePolicies>
        </Service>
    </ProvidingService>
<Services>
```

شکل(۳) ساختار متاداده سرویس

۵ – کنترل دنباله دسترسی

در صورتی که یک درخواست‌کننده متقاضی سرویسی باشد که با تعامل چند سرویس‌دهنده ممکن می‌شود، نیازمند دسترسی به دنباله‌ای از سرویس‌ها است، همانند آنچه در شکل (۴) دیده می‌شود. حال اگر از میان‌افراز استفاده شود، ارائه این توالی بر عهده میان‌افزار است و درصورتی که به هر علتی ترتیب توالی تغییر کند، میان‌افزار به‌گونه‌ای عمل می‌کند که برنامه کاربردی از هرگونه تغییر، بی‌نیاز باشد. ControlSequence باعث می‌شود میان‌افزار از تغیرات توالی دسترسی‌ها مطلع شود. میان‌افزار باید دارای قسمتی باشد که مسوول پیگیری و بررسی توالی است. این قسمت با بررسی متاداده سرویس‌ها و بررسی ControlSequence از این تغییرات مطلع می‌شود. به عنوان مثال، درصورتی که به هر علت، توالی نشان داده شده در شکل (۴) تغییر کند، تنها چیزی که باید تغییر کند، توالی فراخوانی بر اساس سیاست جدید است و برنامه کاربردی نیاز به هیچ تغییری ندارد و می‌تواند نتیجه مطلوب را تولید نماید.



شکل (۴) توالی دسترسی به سرویس‌ها[۷]

۶ –پیاده‌سازی میان‌افزار

در این قسمت به بررسی نکاتی می‌پردازیم که می‌توانند در طراحی میان‌افزار مفید باشند. در نظر گرفتن قسمت‌های مفید در کنار اجزای لازم، مفید خواهد بود. مهمترین قسمت در این میان‌افزار بخشی است که وظیفه اصلی آن اتصال و ارتباط با سرویس واسطه است. از قسمت‌های مهم دیگر، انجام وظایف مربوط به کنترل توالی است که به وظایف آن در بخش قبل اشاره شد. قسمت بعدی، قسمت ثبت اقدامات به منظور پیگیری‌های بعدی است. در صورتی که نیازمند باشیم در صورت بروز خطا در میان‌افزار، مدیر سیستم را مطلع کنیم، می‌توانیم قسمتی را در میان‌افزار بگنجانیم که وظیفه هشدار و اطلاع‌رسانی را برعهده گیرد. به عنوان مثال، اگر برنامه کاربردی خاصی نیازمند یک سرویس خاص باشد، این سرویس را از میان‌افزار درخواست می‌کند. درخواست برنامه کاربردی در میان‌افزار ثبت می‌گردد و در صورتی که برنامه‌کاربردی محدودیت خاصی را نقض کند، این تخلف به مدیر سیستم اطلاع داده می‌شود.

۷– نتیجه‌گیری

در این مقاله کوشیدیم با کنار هم قرار دادن دو فناوری، به شکل جدیدی در طراحی سیستم‌های توزیع‌شده برسیم. تلاش‌های دیگری نیز برای ادغام این دو فناوری صورت گرفته است ولی در این مقاله با تمرکز بر روی فناوری موجود و استفاده از XML به عنوان وسیله تعاملی، روشی مفید و عملی را ترسیم نمودیم. نوع طراحی، امکان هرگونه تغییر براساس نیاز را ممکن می‌سازد. هر طراح می‌تواند با توسعه اجزای تعاملی، اطلاعات بیشتری را در قالب XML بگنجاند. میان‌افزار پیشنهادی در این مقاله می‌تواند شفافیت را بطور کامل ایجاد نماید و شفافیت سطح سرویس را ارائه کند. بدین ترتیب، برنامه‌های کاربردی از پیچیدگی ارتباط با سرویس واسط رهایی می‌یابند. همچنین با در نظر گرفتن ControlSequence در متاداده سرویس و بخشی برای مدیریت این قسمت در میان‌افزار، در صورت هرگونه تغییر در دنباله دسترسی به سرویس‌دهنده‌ها، برنامه کاربردی هیچ تغییری نخواهد کرد. میان‌افزار، خودش این تغییرات را برعهده می‌گیرد و به گونه‌ای عمل می‌کند که برنامه‌های کاربردی نیاز به اطلاع از/ به هیچ جزئیاتی را نداشته باشند.

مراجع

[1] Erl, T., SOA Principles of Service Design, 1st ed., USA, prentice hall, 2007.
[2] Josuttis, N., M., SOA in Practice: The Art of Distributed System Design (Theory in Practice) edition1, O'Reilly Media Inc., USA, 2007.
[3] Erl, T., SOA Design Patterns, USA, prentice hall, 2008.
[4] Rosen, M., Smith, K., T. , Balcer, M., J. , Rosen , M , Smith, K., T. and Balcer, M., J., Applied SOA: Service-Oriented Architecture and Design Strategies 1, USA, John Wiley & Sons, 2008.
[5] Brown, P. C., Implementing SOA: Total Architecture in Practice, USA, Addison-Wesley Professional, 2008.
[6] Kirkham,, Savio,D. Smith, H., Harrison, Monfared, R.P., Phaithoon buathong, P., "SOA middleware and automation: Services, applications and architectures", Industrial Informatics, 6th IEEE International Conference, pp. 1419 – 1424, 2008.
[7] Raghu Anantharangachar, K. Krishna Gorur N. Shrinivas, "A Flexible

Framework for Tools Integration using Service Oriented Architecture", IEEE International Conference on Services Computing, Page: 522, 2006.

[8] Lee, Y., Yeoam, G., "A Research for Web Service Quality Presentation Methodology for SOA Framework", Sixth International Conference on Advanced Language Processing and Web Information Technology, pp. 434-439, 2007

[9] Basu , S., Casati, F., Daniel , F., "Toward Web Service Dependency Discovery for SOA Management " IEEE International Conference on Services Computing, Vol.2 , pp. 422-429, 2008.

[10] Anand,S.,, Padmanabunid , S.,JaiG., "Perspectives on Service Oriented Architecture", IEEE International Conference on Services Computing, Vol.5, pp. 17, 2005.

[11] Rossi,P., Zahir, T., "Software adaptation for service-oriented systems", 1st workshop on Middleware for Service Oriented Computing, pp. 12-17,2005.

[12] Tierney,D., A.Chang, E Curtin Bus. Sch., "a user adaptable user interface model to support ubiquitous user access to EIS style applications", Computer Software and Applications Conference, 29th Annual International Publication Vol. 5, pp.365-369, 2005.

[13] Zhang, L.-J., "SOA Solution Reference Architecture", Web Services, IEEE International Conference, PP: xxxvi -xxxvi, 2007.

[14] Liang-J. Z.," SOA and Web Services, Services Computing," 2006. SCC '06. IEEE International Conference, pp.: xxxvi - xxxvi, 2006.