چهاردهمین کنفرانس ملی سالانه انجمن کامپیوتر ایران
دانشگاه صنعتی امیرکبیر (پلی تکنیک تهران) ایران، تهران – ۲۰ و ۲۱ اسفندماه ۱۳۸۷

ارائه جایگاه جدیدی برای واسط کاربر در معماری مبتنی بر سرویس

جواد احمدی
دانشکده مهندسی برق، رایانه و فن‌آوری اطلاعات
دانشگاه آزاد اسلامی واحد قزوین
ahmadi_javad@yahoo.com

ابوالفضل طرقی حقیقت
دانشکده مهندسی برق، رایانه و فن‌آوری اطلاعات
دانشگاه آزاد اسلامی واحد قزوین
at_haghighat@yahoo.com

سید امیرهادی مینوفام
دانشکده مهندسی برق، رایانه و فن‌آوری اطلاعات
دانشگاه آزاد اسلامی واحد قزوین
minoofam@qazviniau.ac.ir

چکیده: واسط‌کاربر به عنوان تنها ابزار انتقال معنا میان برنامه کاربردی و کاربر، حوزه پژوهشی بسیار مهمی است. معماری مبتنی بر سرویس، رهیافتی است که کارکردهای نرم افزاری را در قالب سرویس ارائه می‌دهد. در این معماری، با توجه به ماهیت استفاده مجدد سرویس، تعریف جایگاه صحیحی برای واسط کاربر، گام مهمی در توسعه آن و همچنین نیل به اهداف طراحی تلقی می‌شود. در این مقاله، با استفاده از فناوری‌های موجود، تعامل میان سرویس و برنامه‌های کاربردی را توسعه داده و جایگاه جدیدی برای واسط کاربر در معماری مبتنی بر سرویس، تعریف می‌کنیم. این ساختار پیشنهادی، با قرار دادن سیاست‌های طراحی واسط‌کاربر در درون سرویس و کاهش هزینه‌های تغییر، مفهوم واسط کاربر مبتنی بر سرویس را ممکن می‌سازد. علاوه بر این، همگون سازی در طراحی را به دنبال دارد و موجب سادگی یادگیری و رضایت‌مندی کاربر می‌گردد.

واژه‌های کلیدی: معماری مبتنی برسرویس، واسط کاربر مبتنی بر سرویس، همگونی واسط کاربر، تعامل میان سرویس‌ها، فناوری XML.

۱- مقدمه

معماری مبتنی بر سرویس، طی سال‌های اخیر توسط دو شرکت IBM و Microsoft به‌وجود آمد، که هر دو از حامیان اصلی سرویس‌های وب نیز بوده‌اند. امروزه معماری‌های نرم افزاری موجود، به قابلیت‌های خوبی دست یافته‌اند. معماری مبتنی بر سرویس قدم تکاملی بعدی برای یاری سازمان‌ها در جهت مدیریت چالش‌های پیچیده است. معماری مبتنی بر سرویس، حالت تکامل یافته معماری مبتنی بر اجزا، طراحی شئ‌گرا و سیستم‌های توزیع شده است. سیستم توزیع شده، تعمیمی از معماری مبتنی بر اجزا است و به اجزایی که در موقعیت‌های فیزیکی مختلف وجود دارند، اشاره می‌کند. مهم‌ترین مزایای معماری مبتنی بر اجزا، سهولت در استفاده مجدد، تغییر هدف اجزای خاص و سهولت در امر نگهداری سیستم است. اهمیت واسط کاربر نیز به عنوان بخش مهمی در هر نرم افزار، بر هیچ‌کس پوشیده نیست. واسط کاربر، ابزار قدرتمندی در دست طراحان است که می‌تواند نقش مهمی در موفقیت یک پروژه نرم افزاری داشته باشد. از آنجایی‌که قدرت اصلی نرم‌افزار در راحتی انتقال معنا است، واضح است که عدم موفقیت در این امر، سایر قابلیت‌های نرم‌افزار را نیز به زیر سؤال خواهد برد. با توجه به مفهوم استفاده مجدد به عنوان یکی از مهم‌ترین قابلیت‌های معماری مبتنی بر سرویس، توسعه‌دهندگان برنامه‌های کاربردی به جای توسعه قابلیت‌های مورد نیاز خود، از سرویس‌دهنده‌های موجود بهره می‌گیرند. سرویس‌دهنده‌ها می‌توانند علاوه بر معرفی قابلیت‌های خود در قالب مجموعه‌های خوش تعریف، مجموعه‌ای از سیاست‌ها را نیز در اختیار برنامه‌های کاربردی قرار دهند. سیاست‌های طراحی واسط کاربر نیز از طریق سرویس‌دهنده می‌تواند در اختیار برنامه‌های کاربردی قرار گیرد. به عنوان مثال، سرویسی که وظیفه اصلی آن ثبت مشخصات مشتری است، می‌تواند از برنامه‌های کاربردی بخواهد کلیدهای میان‌بر خاصی را برای ذخیره یا اصلاح مشخصات مشتری استفاده کنند. در این صورت، هر برنامه کاربردی بکارگیرنده سرویس، در طراحی واسط کاربر خود از این کلید میان‌بر استفاده خواهد کرد. بدین ترتیب، حتی اگر توسعه‌دهندگان متفاوتی این برنامه‌ها را ایجاد کرده باشند نیز به همگونی واسط کاربر برنامه‌های کاربردی دست می‌یابیم. از مهم‌ترین ویژگی‌های واسط‌های کاربر، دارا بودن قابلیت‌هایی چون امنیت استفاده، سادگی در یادگیری و رضایت‌مندی کاربر است. از عوامل تأمین‌کننده این ویژگی‌ها، هم‌خوانی واسط کاربر قسمت‌هایی از برنامه است که کار یکسانی انجام می‌دهند. حصول قابلیت فوق، در سایه ایجاد ساختار لازم برای تعامل بیشتر سرویس‌دهنده و درخواست‌کننده سرویس است. نکته مهم دیگر، ایجاد مرکزیت در سیاست‌های طراحی واسط کاربر و دستیابی به مفهوم واسط کاربر مبتنی بر سرویس است. بدین معنی که سیاست‌های طراحی واسط کاربر باید به درون سرویس منتقل شوند. در صورتی که سیاست‌های سازمان به هر علت دست‌خوش تغییر گردد، کافی است این سیاست‌ها در سرویس مربوطه اعمال شوند تا این امر بصورت خودکار، در کلیه برنامه‌های کاربردی بکارگیرنده این سرویس انعکاس یابد. بدین ترتیب، سازمان نیازی به تغییر برنامه‌های کاربردی بکارگیرنده این سرویس خاص، نخواهد داشت. تکنولوژی مورد استفاده برای رسیدن به اهداف فوق از نکات بسیار حائز اهمیت است. عدم وابستگی به زیرساخت خاص، از جمله زبان برنامه‌نویسی و سیستم عامل نیز از ویژگی‌های معماری مبتنی بر سرویس است [۱۰][۱]. فناوری بکار رفته نباید اصول معماری مبتنی بر سرویس را نقض کند. ساختار

ادامه مقاله به شرح زیر است. در بخش دوم، معماری مبتنی بر سرویس و ویژگی‌های اصلی آن بیان شده است. در بخش سوم، به شیوه طراحی در این معماری اشاره می‌کنیم. مثال گنجانده شده در این قسمت می‌تواند در سهولت درک مطلب، مفید باشد. گسترش تعامل میان سرویس دهنده و سرویس گیرنده و تأثیر آن بر واسط کاربر در بخش چهارم مورد بررسی قرار می‌گیرد. در بخش پنجم، به ساختار متاداده پیشنهادی و نحوه استفاده از آن برای دستیابی به مفهوم واسط کاربر مبتنی بر سرویس اشاره می‌شود. در بخش ششم نیز نمونه‌ای از این متاداده به همراه توضیح کامل آورده شده است. در بخش هفت نحوه پیاده‌سازی روش‌پیشنهادی گنجانده شده است. در انتها ضمن نتیجه‌گیری، درقسمت کار آینده اشاره کوتاهی به استفاده از میان‌افزار در معماری مبتنی بر سرویس خواهیم داشت.

۲- تعریف معماری مبتنی بر سرویس

تعاریف متنوعی برای معماری مبتنی بر سرویس، ارائه شده است که هر یک از دیدگاهی به تبیین خصوصیات آن پرداخته‌اند. برای درک بهتر این مفهوم و آگاهی از برداشت‌ها و دیدگاه‌های موجود، در این قسمت، به چند تعریف اشاره می‌کنیم: ۱- یک چارچوب راهبردی از فناوری که به تمام سیستم‌های داخل و خارج اجازه ارائه یا دریافت سرویس‌های خوش‌تعریف را می‌دهد [۱]. ۲- روشی برای طراحی و پیاده‌سازی نرم‌افزارهای گسترده سازمانی بوسیله ارتباط میان سرویس‌هایی که دارای خواص اتصال سست، دانه‌درشتی و قابلیت استفاده مجدد هستند [۲]. ۳- سبکی از معماری که از اتصال سست سرویس‌ها جهت انعطاف‌پذیری و تعامل پذیری حرفه استفاده می‌کند و بصورت مستقل از فناوری پشتیبانی می‌نماید. همچنین از ترکیب سرویس‌های مبتنی بر حرفه تشکیل شده است که انعطاف‌پذیری و پیکربندی پویا را برای فرایندها محقق می‌سازند [۴]. ۴- چارچوب وسیع و استانداردی که سرویس‌ها در آن ساخته شده، استقرار یافته و مدیریت می‌شوند و هدف آن، افزایش چابکی زیرساخت‌های فناوری اطلاعات در جهت واکنش سریع به تغییرات نیازهای کسب‌وکار می‌باشد [۵]. سرویس، فصل مشترک تمام این تعاریف است. سرویس را می‌توان برآورده کننده قابلیت یا مجموعه‌ای از قابلیت‌ها دانست. این قابلیت می‌تواند همانند تبدیل واحدهای ارزی، ساده یا مانند خرید بلیت خطوط هوایی، بسیار پیچیده باشد [۱۶]. البته باید توجه داشت که ممکن است برخی قابلیت‌ها در نتیجه تعامل چندین سرویس فراهم گردند. آنچه حائز اهمیت است، توانایی معماری مبتنی بر سرویس در به اشتراک گذاشتن قابلیت‌های از پیش فراهم شده بدون نیاز به تولید مجدد و بکارگیری فناوری خاص است برای انتخاب این رویکرد در معماری نرم‌افزار، می‌توان دلایل متعددی را برشمرد، در ادامه به برخی از ویژگی‌های معماری مبتنی بر سرویس اشاره می‌کنیم.

۲-۱ استفاده مجدد

این ویژگی، با استفاده مجدد از سرویس‌های موجود در راستای بکارگیری قابلیت‌های آنان تأمین می‌گردد. این ویژگی از دیرباز مورد توجه تولیدکنندگان برنامه‌های کاربردی بوده است و نقش مهمی در زمان و هزینه دارد. گرچه این ویژگی قبلاً مورد توجه بوده ولی آنچه مهم است نحوه بکارگیری مجدد آن می‌باشد، زیرا آنچه در معماری مهم است عدم وابستگی قابلیت به زیرساخت یا ابزار خاص است [۳] [۱].

۲-۲ قابلیت همکاری

در این رویکرد معماری، گیرندهٔ سرویس و ارائه کننده آن در تعامل گسترده‌ای شرکت می‌کنند و هیچ‌کدام به آگاهی از بستری که در آن درحال اجرا هستند، نیازی ندارند. بستر را می‌توان به زبان برنامه‌نویسی که هردوی آنها با آن تولید شده‌اند نیز تعمیم داد [۳][۱].

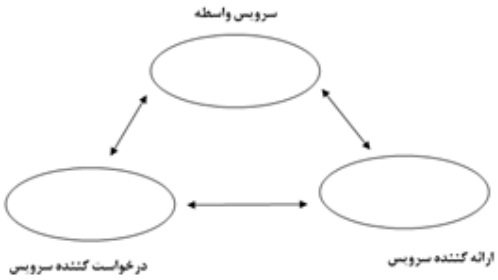
۲-۳ قابلیت انعطاف و کاهش هزینه‌ها

در این معماری، برنامه‌های کاربردی به راحتی قادر هستند قابلیت‌های خود را بهبود بخشیده و بر آنها بیفزایند.کاهش هزینه‌ها، بهره‌گیری از کلیه منابع سازمان‌ها در راستای اهداف مختلف و سهولت تغییر در شرایط مختلف، همواره مورد توجه بوده است. در صورت اتخاذ رویکرد دیگری در معماری نرم‌افزار، تحلیل و طراحی بخشی از نرم‌افزار در پی تغییر شرایط، همواره امری پرهزینه است. این در حالی است که در معماری مبتنی بر سرویس، جابجایی و استفاده از سرویس، امری ساده و بسیار کم هزینه می‌باشد [۳][۱].

۳- طراحی مبتنی بر سرویس

معماری مبتنی بر سرویس، سرویس‌هایی که سیستم از آنها تشکیل شده را تعریف می‌کند. این معماری، تعاملات لازم بین سرویس‌ها جهت بروز رفتار مشخص را توصیف می‌نماید و در نهایت آنها را به یک یا چند پیاده‌سازی در فناوری‌های خاص، تصویر می‌کند [۱۱]. در طراحی مبتنی بر سرویس، هر سرویس برای تعامل با سایرین باید به شکل استانداردی توسعه داده شده باشد. هر سرویس می‌تواند به عنوان یک نقطه پایانی نیز تلقی گردد. به عنوان مثالی از این تعریف می‌توان به دستگاه‌های خودپرداز که برای ارائه سرویس‌های مالی در موقعیت‌های جغرافیایی مختلفی نصب می‌شوند، اشاره نمود. دستگاه خودپرداز به عنوان یک نقطه پایانی در تعامل با کاربران بوده و توانایی لازم برای ارتباط با سرویس‌های دیگر به منظور انجام وظایف را دارا است [۱۳]. هر سرویس، امکان دسترسی به مجموعه خوش‌تعریفی از عملیات را می‌دهد. سیستم به عنوان یک کل، بصورت مجموعه‌ای از تعاملات میان این سرویس‌ها طراحی می‌شود. در صورت بهره‌گیری از این معماری در پایان تولید برنامه‌کاربردی، علاوه بر تولید برنامه مورد درخواست، امکان تولید سرویس‌های دیگری نیز وجود دارد که به هر کدام می‌توان به دید

یک محصول قابل ارائه نگاه کرد. در شکل(۱)، مدل مفهومی معماری مبتنی بر سرویس، ارائه شده است.



شکل (۱) مدل مفهومی معماری مبتنی بر سرویس

ارائه‌کننده سرویس، سرویس‌هایی را برای عموم ارائه می‌کند و این سرویس‌ها را برای دردسترس قرار گرفتن، در سرویس واسطه ثبت می‌کند. سرویس واسطه، شامل اطلاعات سرویس‌ها و خدمات آنها می‌باشد و دارای مخزن بزرگی شامل این اطلاعات است. درخواست‌کننده سرویس برای در اختیار داشتن قابلیتی 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 ارائه کننده این سرویس خاص ختم می‌شود [۷]. یک سرویس ساده می‌تواند توسط تنها یک سرویس دهنده ارائه گردد، در صورتی که یک سرویس پیچیده ممکن است در نتیجه تعامل چند سرویس‌دهنده فراهم شود [۱۲]. آنچه مهم است، راحتی دسترسی سرویس‌گیرنده است، بدون اینکه نیازمند هیچ اطلاعات اضافی در مورد سرویس دهنده یا زبان برنامه‌نویسی و بستر اجرایی آن باشد. شایان ذکر است که سرویس‌گیرنده و سرویس‌دهنده نیازمند یک زبان مشترک برای تعامل باهم و درک نیازهای یکدیگر هستند. این زبان مشترک باید برای هر دو طرف قابل درک باشد. در قسمت بعد، به نمونه‌ای از زبان مشترک و سرویس واسطه اشاره می‌کنیم.

۳-۱ سرویس‌های Web

به عنوان نمونه‌ای از این زبان مشترک، می‌توان به WDSL اشاره کرد. WSDL زبان توصیفی در سرویس‌های Web است که از شمای مستندات XML برای توصیف این سرویس‌ها بهره می‌گیرد. این مستند توصیفی، اطلاعات سرویس را در قالب مجموعه‌ای انتزاعی به نام درگاه توصیف می‌کند [۱۶][۹]. هر قابلیت سرویس دارای درگاهی انتزاعی در این مجموعه است که علاوه بر معرفی آن قابلیت، اطلاعات لازم درباره داده‌های مورد نیاز برای در اختیار گرفتن این قابلیت را دارا می‌باشد. از اطلاعات مهم دیگر درگاه می‌توان به آدرس شبکه‌ای سرویس اشاره‌کرد. برای بهره‌گیری از سرویس‌های Web، درخواست‌کننده سرویس برای بدست آوردن توصیف کامل- WDSL - آن سرویس دهنده خاص به UDDI مراجعه می‌کند که نقش سرویس واسطه را برعهده دارد. این سرویس دارای مخزن بزرگی است که اطلاعات WSDL ارائه‌کننده‌های سرویس در آن نگهداری می‌شود. مثالی که می‌تواند به خوبی UDDI را توصیف کند دفاتر تلفن همگانی‌است [۱۰]. در این دفاتر با مراجعه به صفحات زرد می‌توان اطلاعات کلی فرد مورد جستجو را یافت، ولی برای دسترسی به اطلاعات جامع‌تر از جمله آدرس فرد بایستی با توجه به شماره صفحه، به قسمت اطلاعات فرد مراجعه کنیم. صفحات زرد در این سرویس، اطلاعات کلی از یک سرویس‌دهنده است. این اطلاعات کلی، توصیفی انتزاعی است از آنچه توسط سرویس ارائه می‌شود. شماره صفحه را می‌توان آدرس توصیف کامل سرویس در مخزن WSDL تلقی کرد [۹].

۴- گسترش تعامل سرویس‌دهنده با گیرندهٔ سرویس

آنچه در قسمت قبل قابل توجه است، این است که سرویس‌دهنده‌ها می‌توانند علاوه بر ارائه قابلیت‌های خود در قالب مجموعه‌های خوش تعریف، اطلاعات مفید دیگری نیز در اختیار برنامه‌های کاربردی قرار دهند. سیاست‌های مربوط به طراحی واسط کاربر در این دسته اطلاعات است. اولین مزیت انتقال سیاست‌های طراحی واسط کاربر از طریق سرویس به برنامه‌های کاربردی که از آن بهره می‌گیرد، یک همگونی در طراحی واسط کاربر است. در قسمت بعد، ضمن بیان ویژگی‌های واسط کاربر قابل قبول، به مزایای همگونی نیز اشاره می‌کنیم. مهمترین مزیت انتقال سیاست‌های طراحی واسط کاربر به سرویس، رسیدن به مفهوم واسط کاربر مبتنی بر سرویس است. زیرا با انتقال این سیاست‌ها به درون سرویس، در صورت تغییر سیاست‌های حاکم بر سازمان، تنها این سیاست‌ها در درون سرویس به‌روزرسانی می‌شود و برنامه‌های کاربردی نیازمند هیچ تغییری نیستند. این امر می‌تواند به لحاظ زمان و هزینه بسیار مقرون به صرفه باشد. برای برآورده کردن چنین قابلیت‌هایی نیازمند ارتقای تعامل سرویس‌دهنده و سرویس‌گیرنده هستیم. همان طور که در قسمت قبل اشاره شد، در معماری مبتنی بر سرویس، سرویس‌گیرنده نیازمند اطلاعات خاصی اعم از زبان برنامه‌نویسی، یا زیرساخت اجرایی سرویس‌دهنده نیست. بنابراین در ارتقای این تعامل باید از فناوری استفاده شود که ناقض این اصل مهم نباشد. روش پیشنهادی این مقاله، بهره‌گیری از متاداده برای هر سرویس است. این متاداده، حاوی سیاست‌های طراحی واسط کاربر است. کلیه قابلیت‌های برنامه‌های کاربردی در نتیجه تعامل و خواست کاربر نمود پیدا می‌کند. در صورت استفاده از معماری مبتنی بر سرویس، خواست کاربر منجر به فراخوانی قابلیتی از سرویسی می‌شود که برنامه کاربردی به آن مراجعه کرده است. حال در صورت استفاده از روش پیشنهادی این مقاله، این خود سرویس است که سیاست طراحی آن بخش از واسط کاربر را به برنامه‌کاربردی عرضه می‌کند. به عنوان مثال، ساده‌ترین اطلاعی که

سرویس می‌تواند به برنامه‌کاربردی منتقل کند کلید میان‌بری است، که کاربر با زدن آن می‌تواند باعث فراخوانی آن قابلیت خاص از سرویس شود. در قسمت پنجم به ساختار این متداده و روش پیشنهادی برای رسیدن به چنین ساختاری اشاره خواهیم کرد. حال، در صورتی که این سرویس در سایر برنامه‌های کاربردی در آن سازمان استفاده شود، و حتی توسعه آن قسمت از برنامه بر عهده تیم دیگری باشد نیز این کلید میان‌بر ثابت خواهد بود که بیانگر همگونی است . درصورتی که به هر علتی این کلید میان‌بر تغییر کند، تغییر سرویس کافی خواهد بود و برنامه‌های کاربردی که از این سرویس بهره می‌گیرند، نیازمند هیچ تغییری نیستند. البته تعامل صحیح برنامه کاربردی با سرویس و اعمال سیاست‌های واسط کاربری در صورت هرگونه تغییر احتمالی، الزامی است.

۱-۴ مزیت همگونی در واسط کاربر

هرچند استاندارد مدوّنی در طراحی واسط کاربر وجود ندارد ولی رعایت برخی نکات مهم و ظریف، باعث برتری یک واسط کاربر بر دیگری می‌گردد [۱۴]. طراحی واسط کاربر، فصل مشترک بسیاری از علوم از جمله علوم رفتار شناختی برای رسیدن به تعامل بهتر انسان و کامپیوتر است. باتوجه به گسترش بهره‌گیری از برنامه‌های کاربردی در سازمان‌ها و هزینه‌بر بودن خطاهای انسانی، ایجاد تعامل بهتر، حائز اهمیت است [۱۵]. در صورت بهره‌گیری از معماری مبتنی بر سرویس، بسیار محتمل است که سرویسی در چند برنامه‌کاربردی در یک سازمان مورد استفاده قرار گیرد. به عنوان مثال، ممکن است سرویسی که مسؤول بازیابی و به‌روز رسانی اطلاعات مشتری است هم در برنامه کاربردی فروش و هم در برنامه کاربردی پرداخت، مورد استفاده قرار گیرد. یکی از قابلیت‌های واسط کاربر قابل قبول، یکپارچگی برخی سیاست‌های واسط کاربر در تمام قسمت‌هایی است که برنامه قرار است وظیفه یکسانی را انجام دهد. این یکپارچگی، باعث ایجاد رفتار، کاهش خطای انسانی، ایجاد امنیت و رضایت‌مندی کاربر می‌گردد. شایان ذکر است که این امر، با افزایش سرعت یادگیری واسط کاربر توسط کاربر نهایی، احتمال موفقیت برنامه را افزایش می‌دهد.

۵- ساختارمتاداده سرویس

روشی که در این مقاله پیشنهاد می‌کنیم بهره‌گیری از ساختار مستندات XML برای ساخت این متاداده است. با توجه به استفاده از XML برای ساخت متاداده به هیچ وجه اصل مهم عدم وابستگی نقض نمی‌شود. علت اصلی استفاده از آن را می‌توان در قابلیت‌های XML دانست که مهم‌ترین آن قابلیت‌ها عبارتند از: ۱- XML به وسیله W3C پشتیبانی می‌شود[۵]. همچنین ساختار خوش‌تعریفی دارد که توسط معروف‌ترین زیرساختارهای تولید برنامه‌های توزیع‌شده از جمله Sun Java و Net Framework. پشتیبانی می‌شود [۶]. ۲- XML از قوانین ارتباطی HTTP بهره می‌گیرد که یکی از معروفترین قوانین ارتباطی است. در این طراحی، هر سرویس علاوه بر معرفی قابلیت‌های خود در قالب واسط‌های خوش‌تعریف، سایر ویژگی‌هایی را که می‌تواند در نحوه استفاده از این سرویس مهم باشد را در فرمت XML قرار می‌دهد. هر سرویس، هنگام معرفی خود در سرویس واسط، اطلاعات بیشتری در قالب تگ‌های XML در اختیار سرویس واسط قرار می‌دهد. هر کدام از قابلیت‌هایی که یک سرویس ارائه می‌کند دارای یک تگ با نام service در قسمت ProvidingServies است. به عبارت بهتر، ProvidingServies مجموعه‌ای است از تمام قابلیت‌هایی که یک سرویس ارائه می‌کند. بدین ترتیب، مخزن سرویس واسط، شامل اطلاعات مفیدی از هر سرویس است که این اطلاعات در قالب تگ‌های XML از هم جدا شده‌اند. شکل ۲ می‌تواند نمایشی از متاداده یک سرویس باشد. از آنجایی که یک سرویس می‌تواند خود به تنهایی ارائه کننده چندین سرویس باشد، در این متاداده نیز هر سرویس می‌تواند مجموعه‌ای از سرویس‌ها را درقالب تگ‌های XML در قسمت ProvidingServies ارائه نماید.

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

شکل (۲)ساختار متاداده سرویس

نکته مهم در این متاداده که بر روی آن تأکید داریم UserIterfacePolicy است، که می‌تواند ما را به اهداف یکسان‌سازی و مبتنی بر سرویس بودن رهنمون کند. در UserIterfacePolicy، ارائه‌دهنده سرویس، کلیه ملاحظاتی که استفاده کننده از سرویس در طراحی واسط کاربر باید به آنها توجه کند را قرار می‌دهد. هرگونه تغییر در سیاست‌های طراحی واسط کاربر و الزامات سازمانی، می‌تواند در این قسمت منعکس گردد. برای تغییرات آینده، نیازمند هیچ تغییری در کد برنامه‌های کاربردی نیستیم. متادادهٔ هر سرویس، پس از ایجاد، برای ذخیره در مخزن در اختیار سرویس واسط قرار می‌گیرد. برنامه کاربردی با اتصال به سرویس واسط اقدام به درخواست سرویس‌های مورد نیاز خود می‌کند. در این شیوه جدید علاوه بر دریافت آدرسی که به ارائه دهنده سرویس ختم می‌شود، مجموعه‌ای از ملاحظاتی را دریافت می‌کند که می‌تواند در طراحی واسط خود لحاظ کند. از آنجایی که این اطلاعات در فرمتی قابل فهم است، استفاده از آن بسیار راحت خواهد بود.

۶ – نمونه‌ای از واسط کاربر مبتنی بر سرویس

در این قسمت، در شکل ۳ متاداده سرویس‌دهنده‌ای را می‌بینیم که دو سرویس برای ثبت اطلاعات مشتری و بازیابی این اطلاعات ارائه می‌کند. با توجه به سیاست‌های خاص از میان‌بر ALT + S برای ثبت اطلاعات مشتری و میان‌بر F5 برای بازیابی اطلاعات استفاده می‌شود. حال با توجه به نگرش سرویسی، این سیاست‌ها به جای گنجانده شدن در برنامه کاربردی، توسط سرویس ارائه می‌شوند و برنامه کاربردی ملزم به رعایت این سیاست‌ها است. تغییر میان‌بر ALT + S به میان‌بر ALT + R نیازمند هیچ تغییری در برنامه کاربردی نبوده و کافی است این سیاست در درون سرویس به‌هنگام شود. از طرفی، سیاست یکسانی در تمام برنامه‌های کاربردی استفاده کننده از این سرویس، حاکم خواهد بود. علاوه بر کلیدهای میان‌بر که به آن‌ها اشاره شد، مورد دیگری که می‌تواند برای توسعه‌دهندگان برنامه‌های کاربردی بسیار مفید باشد، صدور پیغام‌های مناسب از طرف برنامه کاربردی است. به عنوان یک توصیه مهم، در ساخت این متاداده می‌توان این پیغام‌ها را نیز گنجانید. در این صورت، کاربر در هر برنامه کاربردی اعم از فروش یا پرداخت که از این سرویس بهره می‌گیرد، پیغام یکسانی در راستای قابلیتی که از سرویس فراخوانی می‌کند، دریافت خواهد نمود. در صورتی که از این توصیه استفاده شود، تولید برنامه‌هایی که واسط کاربر آنها قادر است چندین زبان را پشتیبانی کند، بسیار آسان و کم‌هزینه خواهد بود. چراکه به عنوان مثال، یک پیغام می‌تواند هم به زبان فارسی و هم به زبان انگلیسی در متاداده گنجانده شود، و برنامه‌کاربردی بسته به زبان کاربر نهایی، پیغام مناسب را نمایش می‌دهد. با توجه به بهره‌گیری از فرمت مستندات XML، اضافه کردن زبان‌های دیگر نیز به راحتی ممکن است.

```
<Services>
<Name> CustomerInfo</ Name>
<Description>Save customer information and retrieve
persisted data</ Description>
    <ProvidingService>
        <Service>
            <Name>SaveCustomerInfo</ Name>
            <Description>Save customer information<
        Description>
            <Interface>void Save (int id, string name,
        string lastName, string address) </ Interface>
            <UserInterFacePolicies>
            <Name>Save</ Name>
            <Shortcut>ALT + S< /Shortcut>
            <Validation Message>Error: Duplicate
        information< /Validation Message>
            </UserInterFacePolicies>
        </Service>
        <Service>
            <Name>RetrieveCustomerInfo</ Name>
            <Description>Retrieve customer information<
        Description>
            <Interface>CustomerInfo[ ]  Retrieve ( ) </
        Interface>
            <UserInterFacePolicies>
            <Name>Refresh</ Name>
            <Shortcut>F5< /Shortcut>
            <Validation Message>Error: Connection Fail<
        /Validation Message>
            </UserInterFacePolicies>
        </Service>
    </ProvidingService>
<Services>
```

شکل (۳) نمونه‌ای از متاداده سرویس

۷- پیاده‌سازی روش پیشنهادی

با توجه با اینکه سیاست‌های طراحی واسط کاربر در درون سرویس گنجانده شده است، هر توسعه‌دهنده برنامه کاربردی بایستی سیاست‌های طراحی واسط کاربر را به شکل پویا طراحی نماید. به عنوان مثالی از طراحی پویا می‌توان به عدم قرار دادن کلیدهای میان‌بر ثابت در فرم‌ها اشاره کرد. در این طراحی، سیاست‌های واسط کاربر در زمان اجراء بر اساس متاداده سرویس به‌هنگام می‌شوند. پیشنهاد می‌شود تمامی فرم‌ها interface مشترکی را توسعه دهند تا ملزم به به‌هنگام کردن واسط کاربر در زمان اجراء باشند. قطعاً وجود تابعی در این interface که در پارامترهای ورودی خود متاداده سرویس را درخواست کند الزامی به نظر می‌رسد.

۸- نتیجه‌گیری

در این مقاله، جایگاه جدیدی برای واسط کاربر، در معماری مبتنی بر سرویس تعریف کردیم. با تمرکز بر تعامل سرویس‌گیرنده و سرویس‌دهنده و توسعه این تعامل به یک مدل عملی و قابل پیاده‌سازی رسیده‌ایم. در این معماری، اصل عدم وابستگی سرویس‌گیرنده به سرویس‌دهنده از لحاظ زبان برنامه‌نویسی و زیرساخت اجرایی، الزامی است. از این رو از ساختار مستندات XML برای ایجاد این تعامل بهره‌گرفتیم. زیرا ساختار مستندات XML دارای کلیه قابلیت‌های مورد نیاز، از جمله همه‌فهم بودن و انعطاف بسیار بالا است و امکان توسعه و تغییر برحسب نیاز را بسیار راحت‌تر می‌سازد. در این شیوه پیشنهادی، با قرار دادن سیاست‌های طراحی واسط کاربر در درون سرویس، واسط کاربر مبتنی بر سرویس را ارائه نمودیم. مزایای این شیوه طراحی عبارتند از : ۱-باعث تمرکز سیاست‌های طراحی واسط کاربر و سایر الزامات سازمانی در درون سرویس می‌شود. ۲- در صورتی که این سیاست‌ها به هر دلیل تغییر کند، برنامه کاربردی از هر تغییری بی‌نیاز خواهد بود. این امر به لحاظ هزینه و زمان به‌روزرسانی برنامه‌های کاربردی، بسیار قابل توجه است. ۳- وجود سیاست‌های طراحی واسط کاربر در درون سرویس، باعث ایجاد همگونی در واسط کاربر برنامه‌های استفاده کننده از آن سرویس می‌گردد. یکی از قابلیت‌های واسط کاربر مناسب، همگون بودن سیاست‌ها در تمام قسمت‌هایی است که برنامه، وظیفه یکسانی انجام می‌دهد. در نهایت این همگونی، باعث ایجاد رفتار، کاهش خطا، ایجاد امنیت و رضایت‌مندی کاربر می‌گردد. همچنین این امر، سرعت یادگیری واسط کاربر توسط کاربر نهایی را افزایش داده و احتمال موفقیت برنامه را نیز افزایش می‌دهد. ۴- این شیوه پیشنهادی، طراحان برنامه کاربردی را قادر می‌سازد بسته به خلاقیت خود انعطاف بیشتری را در طراحی متاداده آن ایجاد کنند. به عنوان مثالی از این انعطاف، توضیح داده شد که با بهره‌گیری از واسط کاربر مبتنی بر سرویس، تولید برنامه‌های کاربردی که واسط کاربر آنها قادر به پشتیبانی چندین زبان باشد، بسیار راحت و کم‌هزینه است.

۹- کارهای آینده

میان‌افزار به‌عنوان لایه واسط در معماری نرم‌افزار، توسعه‌دهندگان را قادر می‌سازد کار خود را بدون اطلاع از پیچیدگی‌های لایه‌های زیرین انجام دهند [۷]. در صورتی که در معماری مبتنی بر سرویس از میان‌افزار استفاده شود، وظیفه اصلی آن تعامل با سرویس واسطه خواهد بود [۶]. مزیت مهم این طراحی ایجاد شفاف‌سازی در تعامل برنامه‌های کاربردی با سرویس‌ها است. زیرا برنامه‌های کاربردی به جای تعامل مستقیم با سرویس واسطه با میان‌افزار تعامل دارند. در صورتی که انجام یک قابلیت در نتیجه تعامل با چندین سرویس باشد، میان‌افزار، مسؤول مدیریت این توالی خواهد بود [۱۶][۶]. در صورت گسترش وظایف میان‌افزار، مفهوم پیشنهادی واسط کاربر مبتنی بر سرویس می‌تواند به مفهوم به‌کارگیری میان‌افزار در این معماری اضافه گردد. در این ایده جدید، میان افزار می‌تواند با دریافت متاداده هر سرویس از سرویس واسطه، مسؤول تفسیر سیاست‌های واسط کاربر و الزامات سازمانی برای برنامه کاربردی باشد. انعطاف موجود در هردو ایده، امکان بسیاری از تغییرات و ابداعات را گسترده‌تر می‌سازد.

مراجع

[1] Erl, T., *SOA Principles of Service Design*, edition 1 , USA, by prentice hall 2007.
[2] Josuttis, N., M., *SOA in Practice: The Art of Distributed System Design*, edition1, O'Reilly Media,Inc, USA, 2007.
[3] Erl, T., SOA *Design Patterns*, USA, by 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, by John Wiley & Sons, May 2008.
[5] Brown, P. C., *Implementing SOA: Total Architecture in Practice*, USA, Addison-Wesley Professional 2008.
[6] Lee, Y., C., Ma, C., Chou, S. C., "A Service-Oriented Architecture for Design and Development of Middleware, Software Engineering", Conference, APSEC apos05. 12th Asia-Pacific Vol.2, 2005.
[7] Kirkham,, Savio,D. Smith, H., Harrison, Monfared,R.P., Phaithoonbuathong, P.,"SOA middleware and automation: Services, applications and architectures", 6th IEEE International Conference, pp. 1419 – 1424, 2008.
[8] Raghu Anantharangachar,, K. Krishna Gorur N. Shrinivas "A Flexible Framework for Tools Integration using Service Oriented Architecture", IEEE International Conference on Services Computing, pp. 522, 2006.
[9] Lee, Y., Yeoam, G., "A Research for Web Service Quality Presentation Methodology for SOA Framework", International Conference on Advanced Language Processing, pp. 434-439, 2007
[10] 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.
[11] S., Anand,S. Padmanabunid, J. Ganesh, "Perspectives on Service Oriented Architecture", IEEE International Conference on Services Computing, pp 17, 2005.
[12] R. Pablo,Z. Tari, "Software adaptation for service-oriented systems", Proceedings of the 1st workshop on Middleware for Service Oriented Computing, pp, 12-17, 2005
[13] L. Xudong W. Jiancheng , S. Univ., Shandong "User Interface Design Model", Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007.
[14] Kulkarni, N. Dwivedi, "The Role of Service Granularity in a Successful SOA Realization A Case Study", Congress on Services, SERVICES '08. IEEE, pp.423 - 430, 2008.
[15] Ingolf H. Krüger ,Reena Mathew, Michael Meisinger , "Efficient exploration of service-oriented architectures using aspects Proceedings", 28th international conference on Software engineering, pp. 62-71, 2006
[16] Zhang, Liang-Jie, "SOA Solution Reference Architecture Web Services", ICWS IEEE International Conference, pp. xxxvi-xxxvi, 2007.