دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

گزارش کتبی سمینار

درس مقدمه‌ای بر الگوهای طراحی نرم‌افزار شئ‌گرا

عنوان

مروری بر پژوهش‌های انجام‌شده در حوزه الگوهای طراحی نرم‌افزار شئ‌گرا و الگوهای معماری سرویس‌گرا

نگارش

سید محمدمهدی احمدپناه

۹۶۱۳۱۹۱۳

استاد راهنما

دکتر سید مجید نورحسینی

بهمن ۱۳۹۶

چکیده

در این گزارش، مروری بر پژوهش‌های انجام‌شده در زمینه الگوهای توسعه و طراحی نرم‌افزار انجام می‌شود. ابتدا به معرفی پژوهش‌های انجام‌شده پرداخته‌ایم و سپس انواع دسته‌ها و حوزه‌های تحقیقاتی فعال در این زمینه معرفی شده‌اند.

همچنین، به عنوان یکی از محورهای پژوهشی الگوهای توسعه و طراحی نرم‌افزار، مروری بر الگوهای مربوط به معماری سرویس‌گرا صورت گرفت. تعدادی از الگوهای معروف در قسمت‌های معماری سرویس‌گرا، الگوهای پایه‌ای سرویس، الگوهای امنیت سرویس، الگوهای قرارداد سرویس و الگوهای مدیریت سرویس مطرح می‌شود و درباره انواع هر یک از آن‌ها بحث شده است.

واژه‌های کلیدی:

الگوهای توسعه و طراحی نرم‌افزار، پژوهش‌های حوزه الگوهای طراحی نرم‌افزار شی‌گرا، الگوهای معماری سرویس‌گرا

فهرست عناوین

صفحه

فهرست شکل‌ها **صفحه**

فصل اول

مروری بر پژوهش‌های انجام‌شده در حوزه الگوهای طراحی نرم‌افزار شیءگرا

مروری بر پژوهش‌های انجام‌شده در حوزه الگوهای طراحی نرم‌افزار شئ‌گرا

الگوهای نرم‌افزار راه‌حل‌هایی را برای مسائلی عنوان می‌کنند که هنگام توسعه یک نرم‌افزار در یک زمینه خاص پدید می‌آیند [1] . به این ترتیب، می‌توان هر الگو را جفتی از مسئله-راه‌حل برای یک زمینه در نظر گرفت. به بیان دیگر، الگوهای نرم‌افزار برای بازاستفاده معماری و طراحی نرم‌افزار مطرح می‌شوند. در ادامه، مروری بر روی انواع دسته‌بندی‌های الگوهای در فازهای گوناگون توسعه نرم‌افزار می‌شود.

۱.۱ معرفی الگوهای توسعه نرم‌افزار در بخش‌های مختلف توسعه نرم‌افزار

در سال ۱۹۹۰ Gang of Four کار جمع‌آوری الگوهای طراحی را در قالب یک کاتالوگ آغاز کردند که منجر به چاپ کتاب Design Patterns در سال ۱۹۹۴ شد. اما پیش از آن و در سال ۱۹۹۲ Peter Coad [2] الگوهایی برای نرم‌افزارهای شئ‌گرا ارائه کرد که این الگوهای هفت‌گانه عبارتند از:

- Item Description
- Time Association
- Event Logging
- Roles Played
- State over a Collection
- Behavior over a Collection
- Broadcast

در کتاب Design Patterns که به Gang of Four نیز معروف است، ۲۳ الگوی طراحی نرم‌افزار شی‌گرا در قالب سه دسته Creational، Structural و Behavioral معرفی می‌شود که کماکان به عنوان مهم‌ترین مرجع برای الگوهای طراحی شی‌گرایی نرم‌افزار محسوب می‌شود.

در سال ۱۹۹۸ Larman [3] الگوهای نرم‌افزار General Responsibility Assignment (GRASP) را مطرح کرد که در آن‌ها مفاهیم و اصول شی‌گرایی در قالب الگوهایی در نظر گرفته شده‌اند. در ویرایش سوم این کتاب در سال ۲۰۰۴ [4]، ۹ الگوی مرتبط با نحوه انتساب مناسب مسئولیت‌ها در طراحی شی معرفی شده است. این الگوها عبارتند از:

- Information Expert
- Creator

- Low Coupling
- High Cohesion
- Controller
- Polymorphism
- Indirection
- Pure Fabrication
- Protected Variations

همچنین، در سال ۱۹۹۶ معماری نرم‌افزار مبتنی بر الگو توسط Gang of Five پیشنهاد شد [5] و الگوها را در سه دسته Architectural، Design و Idiom تقسیم‌بندی کرده‌اند. از الگوهای معرفی‌شده برای الگوهای معماری می‌توان به موارد زیر اشاره کرد:

- Layers
- Pipes and Filters
- Blackboard
- Broker
- Model-View-Control (برای سیستم‌های تعاملی)
- Presentation-Abstraction-Control (برای سیستم‌های تعاملی)
- Reflection
- Microkernel (برای سسیتم‌های انطباق‌پذیر)

علاوه بر این، الگوهای معرفی‌شده برای دسته Design نیز شامل موارد زیر است:

- Whole-Part برای تجزیه اجزای مختلف یک سیستم
- Master-Slave برای سازمان‌دهی ارتباط بین اجزا برای حل یک مسئله پیچیده بزرگ
- Proxy برای کنترل دسترسی
- Command Processor و View Handler برای مدیریت اشیا، خدمات و اجزای سیستم
- Forward-Receiver، Client-Dispatcher-Server و Publisher-Subscriber برای سامان‌دهی ارتباط بین اجزا

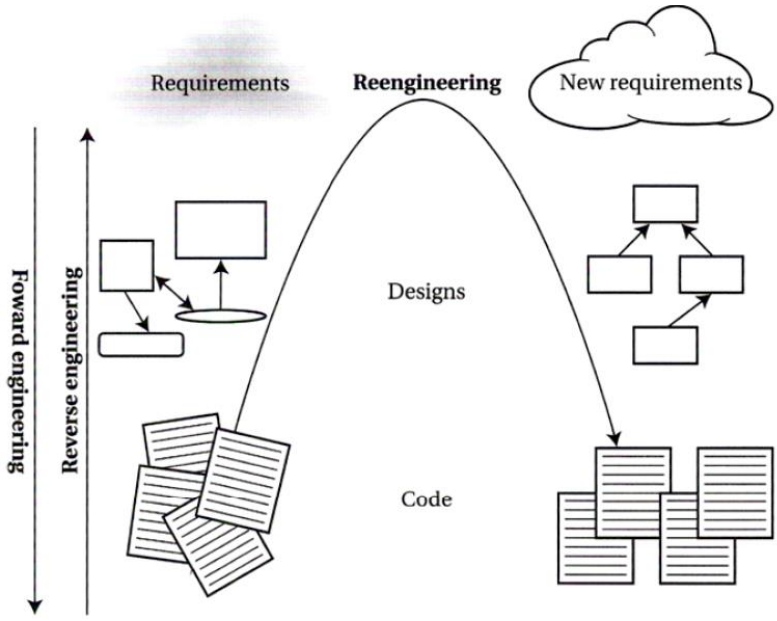
همچنین لازم به ذکر است که الگوهایی برای Refacroring نرم‌افزارها پیشنهاد شده است [6]. منظور از Refactoring تغییر در ساختار داخلی نرم‌افزار برای راحت‌تر کردن فرایند اعمال تغییرات بعدی در نرم‌افزار بر اساس نیازمندی‌های جدید و آزمون سریع‌تر آن است. معمولا هنگامی که یک عمل مشابه

بیش از دوبار در کد برنامه وجود داشته باشد، یا نیاز به ایجاد یک عملکرد جدید در نرم‌افزار مطرح شده باشد، و یا در حین بررسی کد و آزمون کارکرد نرم‌افزار، عملیات Refactoring صورت می‌پذیرد.

از جمله دسته‌بندی‌های الگوهای Refactoring باید به موارد زیر اشاره کرد:

- Composing Methods
- Moving Features Between Objects
- Organizing Data
- Simplifying Conditional Expressions
- Making Method Calls Simpler
- Dealing with Generalization
- Big Refactorings

نوع دیگری از الگوها، الگوهای بازمهندسی[1] در توسعه نرم‌افزار است [7]. هدف از بازمهندسی، کاهش کاهش پیچیدگی یک سیستم میراثی[2] است به طوری که بتواند نیازمندی‌های جدید و به‌روز را با هزینه قابل قبولی برآورده سازد.



شکل ۱ – نمایی از مفهوم فرایند بازمهندسی

[1] Reengineering

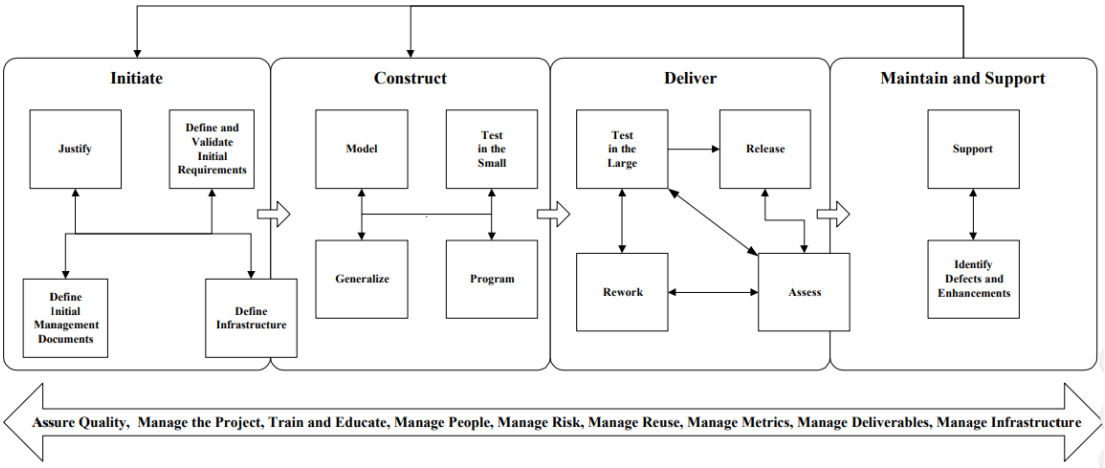
[2] Legacy System

دسته‌بندی الگوهای مطرح‌شده در این زمینه به صورت زیر است:

- Setting Direction
- First Contract
- Initial Understanding
- Detailed Model Capture
- Tests
- Migration Strategies
- Detecting Duplicated Code
- Redistribute Responsibilities
- Transform Conditionals to Polymorphism

علاوه بر الگوهایی که تا اینجا عنوان شد، الگوهایی برای فرایندهای تولید نرم‌افزار مطرح هستند [8]، که در شکل ۲ نیز نمایی از فرایند تولید نرم‌افزار شی‌گرا قابل مشاهده است. همچنین، الگوهایی برای تحلیل نرم‌افزار نیز ارائه شده است [۹]. دسته‌های مختلف الگوهای تحلیل عبارتند از:

- Accountability
- Observations and Measurements
- Referring to Objects
- Inventory and Accounting
- Planning
- Trading



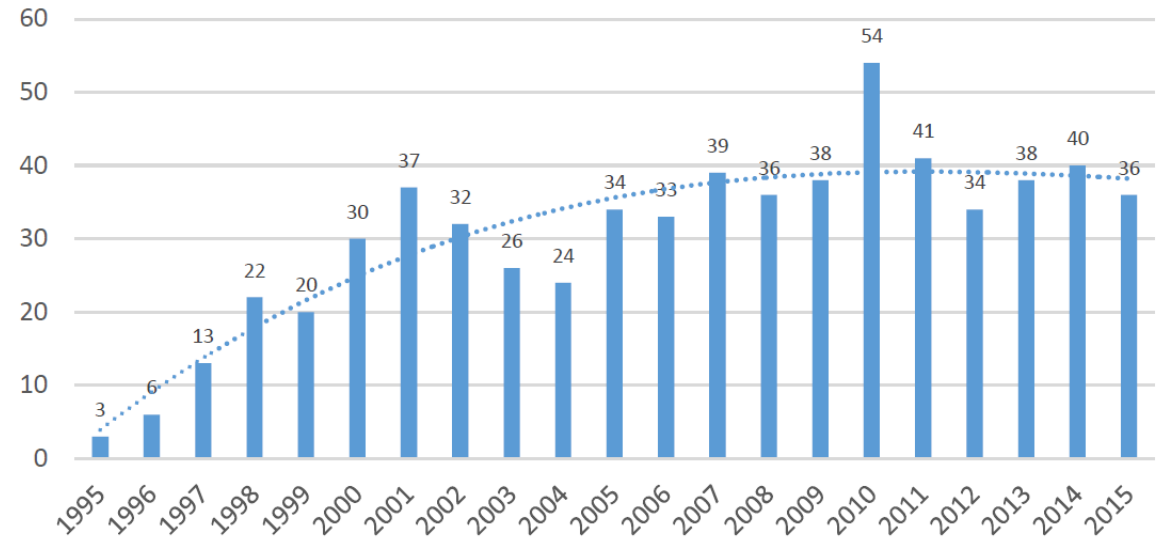
شکل ۲ – فرایند نرم‌افزار شی‌گرا [۸]

۲.۱ مرور نظام‌مند پژوهش‌های صورت‌گرفته در حوزه الگوهای طراحی نرم‌افزار

تا این‌جا به طور مختصر و موردی، انواع الگوهای مختلف در مراحل مختلف توسعه نرم‌افزار عنوان شد که می‌توان به هر یک از آن‌ها به تفصیل پرداخت که خارج از محدوده این گزارش است. علاوه بر موارد مطرح‌شده، برای کاربردهای خاص نیز الگوهایی معرفی شده‌اند. به عنوان نمونه می‌توان به الگوهایی برای سیستم‌های موبایل و برنامه‌های کاربردی آن‌ها [۱۰]–[۱۵]، الگوهای آزمون [۱۶]–[۲۱]، الگوهای طراحی و توسعه وب [۲۲]–[۲۶]، الگوهای مورد استفاده در رایانش ابری [۲۷]–[۳۳]، الگوهای مورد استفاده برای اینترنت اشیا [۳۴]–[۳۷] ، الگوهای توسعه مبتنی بر مدل [۳۸]–[۴۱]، الگوهای معماری سرویس‌گرا [۴۲]–[۴۷]، الگوهای معماری سازمانی [۴۸]–[۵۴]، الگوهای مورد استفاده برای سیستم‌های توزیع‌شده [۵۵]–[۶۲] و الگوهای امنیتی [۶۳]–[۷۴] اشاره کرد.

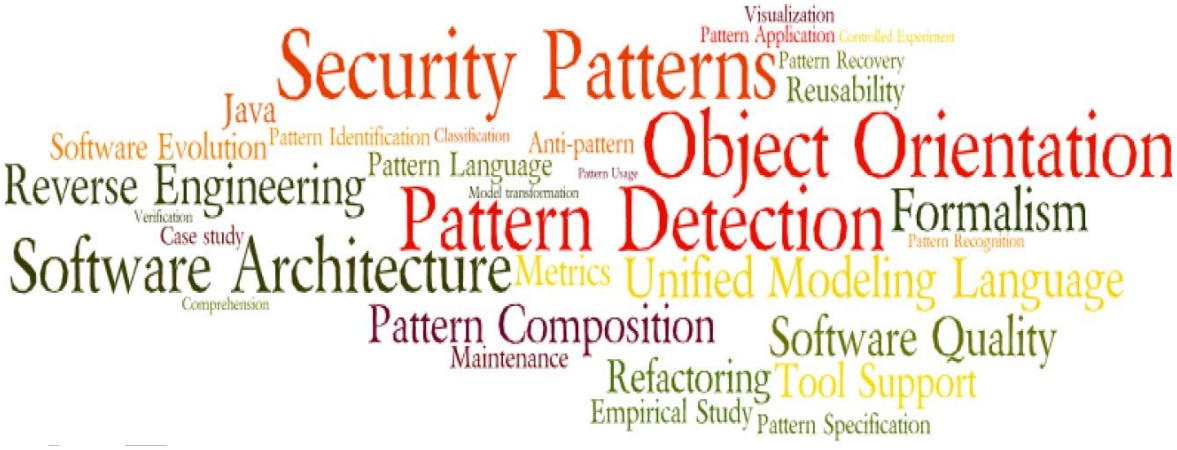
از مرورهای انجام‌شده درباره الگوهای طراحی نرم‌افزار می‌توان به مقاله‌ای اشاره کرد که به طور نظام‌مند درباره فقط الگوهای مطرح‌شده توسط Gang of Four بحث کرده است [۷۵]. گرچه مقالات دیگری [۷۶]–[۷۸] نیز تلاش‌هایی برای انجام یک پژوهش مروری در این حوزه انجام داده‌اند اما همگی محدود به الگوهای معروف ارائه‌شده توسط Gang of Four هستند.

یکی دیگر از مرورهای انجام‌شده در حوزه الگوهای طراحی نرم‌افزار شی‌گرا [۷۹] به بررسی نظام‌مند درباره تلاش‌های صورت‌گرفته در این حوزه می‌پردازد و همه الگوهای موجود در فضای الگوهای نرم‌افزار را بررسی می‌کند. در این مقاله، موضوعات پژوهشی طراحی الگوهای نرم‌افزار مشخص و اهمیت هر یک از آن‌ها را نیز تعیین کرده است و درباره همه الگوهای طراحی بحث شده است. در شکل ۳ می‌توان تعدد مقالات منتشرشده در این حوزه را بر اساس سال میلادی مشاهده کرد.



شکل ۳ – تعداد مقالات منتشرشده در حوزه الگوهای نرم‌افزار در بازه سال‌های ۱۹۹۵ تا ۲۰۱۵ [۷۹]

همچنین در شکل ۴، کلیدواژه‌ها و عبارات مطرح در این زمینه تحقیقاتی به طور گرافیکی قابل مشاهده است که از بین آن‌ها عبارات Security Patterns، Object Orientation و Pattern Detection از پرتکرارترین‌ها به شمار می‌روند.



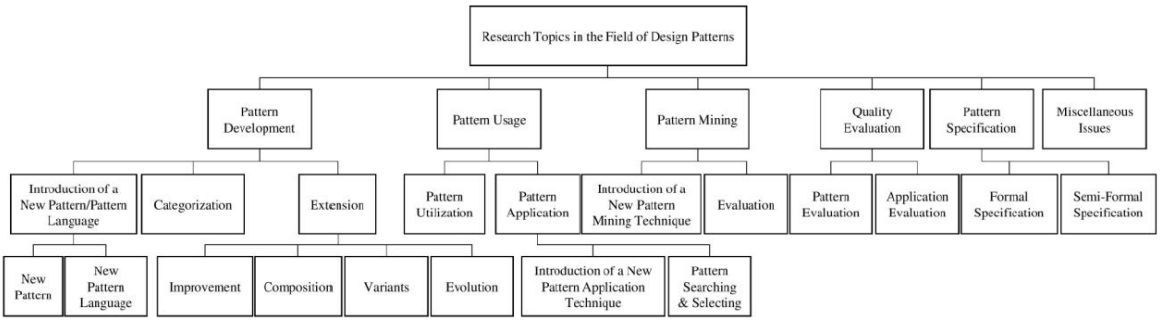
شکل ۴ – عبارات و کلیدواژه‌های پرتکرار در زمینه‌های تحقیقاتی مرتبط با الگوهای نرم‌افزار [۷۹]

به طور کلی می‌توان الگوهای نرم‌افزار بر اساس فاز توسعه نرم‌افزار به دسته‌های الگوهای پیاده‌سازی، الگوهای آزمون، الگوهای تحلیل و الگوهای طراحی دسته‌بندی کرد. از طرفی، موضوعات پژوهشی مرتبط

با الگوهای نرم‌افزار را می‌توان به شش حوزه اصلی تقسیم‌بندی کرد که به ترتیب فعال‌بودن پژوهش‌ها عبارتند از:

- Pattern Development
- Pattern Mining
- Pattern Usage
- Quality Evaluation
- Pattern Specification
- Miscellaneous Issues

در شکل ۵، این دسته‌بندی به همراه جزئیات بیشتری قابل مشاهده است.



شکل ۵ – دسته‌بندی موضوعات پژوهشی در حوزه الگوهای نرم‌افزار [۷۹]

۱.۲.۱ حوزه Pattern Development

هر گونه توسعه در پژوهش الگوهای طراحی نرم‌افزار در این حوزه قرار می‌گیرد و شامل موارد زیر است:

- معرفی یک الگوی جدید یا یک زبان الگوی جدید: یک زبان الگو مجموعه‌ای از الگوهای طراحی مرتبط با یکدیگر است که برای سرشت‌نمایی یک متدلوژی طراحی در کنار یکدیگر قرار گرفته‌اند. ارائه یک زبان الگو چیزی فراتر از معرفی الگوها و ساختار آن‌ها است و در آن راهنمایی‌ها و نکاتی برای توسعه‌دهندگان وجود دارد که بتوانند با استفاده از الگوها، سیستم‌های پیچیده‌تری را طراحی کنند.
- بحث درباره گونه‌های مختلف یک الگو و تکامل الگو: منظور از گونه‌های مختلف یک الگو، تغییراتی است که می‌تواند در درون یک الگوی طراحی ایجاد شود تا بتواند علی‌رغم نگه‌داری قسمت‌های اصلی الگو، با پیاده‌سازی در شرایط خاصی سازگار شود. همچنین منظور از تکامل الگو، در نظر گرفتن شرایط آتی در یک الگو برای بهبود آن بسته به نیازمندی‌های بعدی نرم‌افزار

است. به این ترتیب، می‌توان الگوهای موجود را به نحوی بهبود داد که تا حد امکان نقایص فعلی آن را برطرف کرد.

- تقسیم‌بندی الگوهای طراحی موجود در زمینه‌های گوناگون مانند برنامه‌های موبایل و امنیت از دیگر نکات این دسته به شمار می‌رود.

۲.۲.۱ حوزه Pattern Specification

این حوزه شامل پژوهش‌هایی است که از روش‌ها و نمادگذاری‌های متفاوت برای بیان الگوها استفاده کرده‌اند. طرح‌های مختلفی برای بیان الگوهای طراحی نرم‌افزار وجود دارد که تا حد امکان از توصیف مبهم و نادقیق یک الگو به زبان طبیعی جلوگیری شود. این طرح‌ها عبارتند از:

- طرح‌های توصیف صوری: از آن‌جایی که توصیف رفتار یک برنامه با روش‌های ریاضی و زبان‌های صوری بدون ابهام هستند، می‌توان از یک زبان صوری برای بیان الگوهای طراحی نیز استفاده کرد. بنابراین، جنبه‌های ساختاری و رفتاری یک الگو به کمک نحو و معناشناسی یک زبان صوری به طور دقیق بیان می‌شود.
- طرح‌های توصیف نیمه صوری: منظور از یک زبان نیمه صوری زبانی است که نحو و واژگان مورد استفاده در آن مشخص است اما معناشناسی دقیقی در آن ارائه نمی‌شود.

۳.۲.۱ حوزه Pattern Usage

در این حوزه، پژوهش‌های مرتبط با به‌کارگیری الگوها در فرایند توسعه نرم‌افزار قرار می‌گیرند. بر همین اساس می‌توان این حوزه را به دو دسته اصلی تقسیم‌بندی کرد:

- استفاده از الگوها: ارائه روش‌های مبتنی بر الگو یا طراحی نرم‌افزارهای خاص‌منظوره به کمک الگوهای طراحی در این دسته قرار می‌گیرند. همان‌طور که پیشتر نیز مطرح شد، الگوهای طراحی را می‌توان در زمینه‌های مختلف مانند امنیت، مدل‌های طراحی خودکار با کامپیوتر و برنامه‌نویسی شی‌گرایی به کار بست. به این ترتیب، نحوه استفاده از الگوها کمک می‌کند تا استانداردهای مربوط به مفاهیم طراحی و چالش‌های استفاده از یک الگوریتم به طور دقیق مشخص شود.

- کاربرد الگو: پژوهش‌های مربوط به استفاده از الگوهای طراحی در پیاده‌سازی و طراحی نرم‌افزار در این دسته قرار داده می‌شود. نمونه‌ای از این کارها را می‌توان در مستنداتی که نحوه جستجو و انتخاب یک الگو از میان الگوهای مختلف برای طراحی و پیاده‌سازی یک نرم‌افزار بیان می‌کنند، مشاهده کرد.

۴.۲.۱ حوزه Quality Evaluation

یکی از اصلی‌ترین دغدغه‌هایی که توسعه‌دهندگان نرم‌افزار در هنگام استفاده از یک الگو دارند، بحث درباره کیفیت الگو و تأثیر استفاده از آن الگو بر روی سیستم است. پژوهش‌های انجام‌شده در این حوزه در دو دسته مطرح می‌شوند:

- تکامل الگو؛ به معنای ارزیابی کیفیت الگوها
- تکامل به‌کارگیری؛ به معنای سنجش تاثیرات به‌کارگیری الگوهای طراحی در کیفیت یک سیستم نرم‌افزاری است.

بررسی تکامل الگوهای طراحی به شیوه عددی (معیارهای محاسبه کد) و تجربی (نظرات متخصصان) انجام‌پذیر است.

۵.۲.۱ حوزه Pattern Mining

نوع دیگری از پژوهش‌ها به بررسی چالش‌ها و ارائه راه‌حل‌هایی برای یافتن الگوهای استفاده‌شده در یک سیستم نرم‌افزاری می‌پردازد. در صورتی که بتوان الگوهای استفاده‌شده در طراحی یک نرم‌افزار را تشخیص داد، می‌توان فرایندهای refactoring، بازمهندسی و نگه‌داری سیستم را سریع‌تر و راحت‌تر انجام داد. از طرفی این کار ارزیابی کیفیت نرم‌افزار، درک از برنامه و بهبود مستندسازی برنامه را تسهیل می‌کند. دو گروه اصلی در این دسته از پژوهش‌ها عبارتند از:

- معرفی روش‌های جدید برای تشخیص الگوهای طراحی نرم‌افزار
- ارزیابی تکنیک‌های موجود در کاوش الگوهای استفاده‌شده در سیستم‌های نرم‌افزاری

در سال‌های اخیر بیشتر مقالات و پژوهش‌های انجام‌شده در زمینه Pattern Mining، Pattern Develoment و Pattern Usage بوده است به طوری که در سال ۲۰۱۵، Pattern Mining از سایر حوزه‌ها تعداد مقالات بیشتری را به خود اختصاص داده است.

در این فصل به طور اجمالی حوزه‌های پژوهشی مختلف و الگوهای موجود معرفی شدند. همان‌طور که پیشتر نیز اشاره شد، الگوهایی مخصوص کاربردهای خاص مطرح شده‌اند که در فصل بعدی به بررسی الگوهای معماری سرویس‌گرا می‌پردازیم.

فصل دوم

مروری بر الگوهای معماری سرویس‌گرا

۱.۲ مقدمه

معماری سرویس‌گرا[۳] (SOA) یک طراحی برای متصل‌کردن منابع تجاری و پردازشی (سازمان-ها، نرم‌افزارها و داده‌ها) بر اساس تقاضا به منظور دستیابی به نتایج مورد نیاز سرویس‌گیرندگان است. سرویس‌گیرندگان ممکن است کاربران نهایی یا سرویس‌های دیگری باشند. معماری سرویس‌گرا این‌گونه قابل تعریف است [80]:

«یک الگو برای سازماندهی و استفاده از توانایی‌های توزیع‌شده که ممکن است در کنترل حوزه-های مالکیتی متفاوتی باشند. معماری سرویس‌گرا ابزاری یکپارچه برای بیان، شناسایی، تعامل و استفاده از توانایی‌ها برای ایجاد اثراتی منطبق با پیش‌شرط‌ها و انتظارات قابل اندازه‌گیری ارائه می‌کند. معماری سرویس‌گرا شامل تکامل معماری مبتنی بر مولفه[۴]، طراحی مبتنی بر واسط[۵] و سیستم‌های توزیع شده[۶] است.»

اصلی‌ترین انگیزه در سیستم‌های مبتنی بر این معماری، تسهیل رشد سیستم‌های سازمانی بزرگ، تسهیل تامین و استفاده از سرویس‌ها در مقیاس بزرگ اینترنت، و کاهش هزینه همکاری‌های بین سازمانی است. ارزش معماری سرویس‌گرا به این است که الگویی ساده و مقیاس‌پذیر[۷] برای سازماندهی شبکه بزرگی از سیستم‌هایی است که برای تحقق بخشیدن به ارزش ذاتی هر یک از مولفه‌ها، نیاز به همکاری با یکدیگر دارند [80]. معماری سرویس‌گرا مقیاس‌پذیر است؛ چون کمترین فرضیات ممکن را نسبت به شبکه دارد و فرضیات مربوط به اعتماد به سیستم‌ها را در حداقل ممکن نسبت به سیستم‌های با مقیاس کوچکتر در نظر می‌گیرد.

---

[3] Service Oriented Architecture

[4] Component Based Architecture

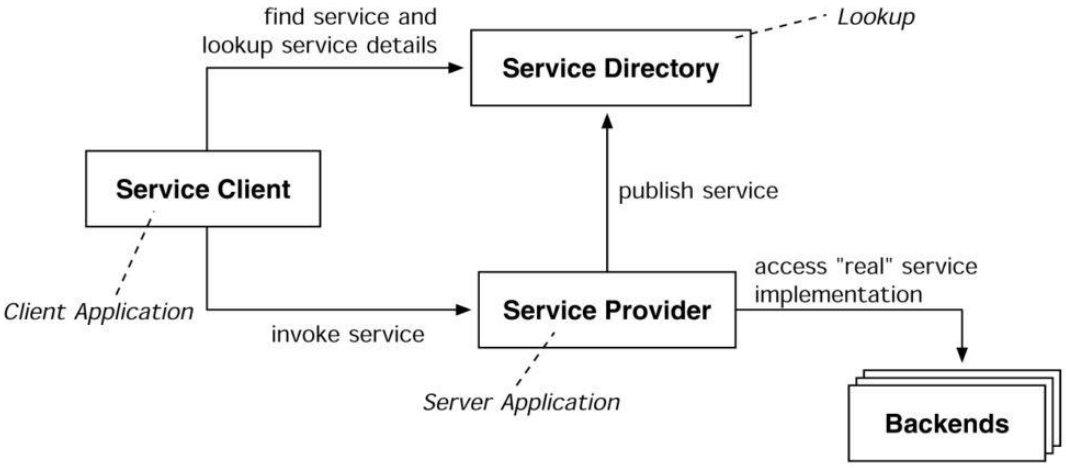
[5] Interface Based Design

[6] Distributed Systems

[7] Scalable

۲.۲ ساختار کلی معماری سرویسگرا

یک سرویس، از طریق یک واسط از راه دور[8] ارائه می‌شود و توصیف واسطی[9] که به خوبی آن را توصیف می‌کند، از خود ارائه می‌دهد. این توصیف، همه جزئیات واسط سرویس مانند امضای عملیات‌ها[10] و چگونگی دسترسی به این عملیات‌های از راه دور را در بر می‌گیرد. سرویس خود را در یک سرویس مرکزی که سرویس جستجو[11] نامیده می‌شود، ثبت‌نام و معرفی می‌کند. دیگر نرم‌افزارها، از این پس می‌توانند سرویس ثبت‌شده را با نام یا مشخصات آن جستجو کرده، از جزئیات نحوه بهره‌برداری از آن مطلع شده و از آن سرویس بگیرند. معمولا یک سرویس به طور کامل توسط خود ارائه‌کننده سرویس پیاده‌سازی نمی‌شود و در تعدادی backend مانند پایگاه‌داده، دیگر معماری‌های سرویس‌گرا، سیستم‌های ERP و سیستم‌های میراثی پیاده‌سازی شده‌اند. شکل ۶ معماری ساده‌ای از این سیستم‌ها را نمایش می‌دهد.



شکل ۶- معماری ساده SOA [81]

[8] Remote Interface

[9] Interface Description

[10] Operation Signatures

[11] Lookup service

۳.۲ مروری بر کارهای پیشین

مقالات و کتب متعددی در زمینه الگوها در معماری سرویس‌گرا نوشته شده‌اند. در یک نگاه کلی می‌توان گفت برخی از آنها مانند [44] و [81] به دنبال تفکیک معماری سرویس‌گرا و مشخص‌کردن الگوهای ریزدانه طراحی و معماری به‌کاررفته در هر یک از مولفه‌ها هستند. این گونه مقالات، به دنبال ایجاد درک عمیق از ماهیت و عملکرد این معماری با یک دید بالا به پایین[۱۲] هستند.

برخی دیگر مانند [85]–[82][85] الگوهای مورد استفاده در صنعت برای پیاده‌سازی این معماری در شرایط گوناگون را جمع‌آوری کرده و به شکل اصولی الگوها (مسئله، زمینه، راه حل) ارائه کرده‌اند. الگوهای بررسی‌شده در این کتاب‌ها و مقالات، به شکل یک کاتولوگ الگو[۱۳] توصیف شده و ارتباطات و وابستگی آن‌ها به یکدیگر نیز به تشریح بررسی شده‌اند.

پژوهش‌هایی به ارتباط ویژگی‌های کیفی همچون کارایی و امنیت در این معماری پرداخته و الگوهای مورد استفاده در این معماری را از این حیث تحلیل کرده [47] و [86] و یا به چگونگی اعمال الگوهای مربوط به این ویژگی‌ها در معماری سرویس‌گرا پرداخته‌اند [43] و [87] . در [42]، روشی برای ترکیب الگوهای مختلف این معماری ارائه شده است که به دنبال جلوگیری از استفاده‌های خارج از زمینه و اشتباه در ترکیب الگوهاست.

تحقیقاتی نیز در زمینه پادالگوهای[۱۴] مرتبط با این معماری صورت گرفته است. تلاش‌هایی برای جمع‌آوری پادالگوهای این معماری صورت گرفته است که [88] و [89] نمونه‌هایی از آن‌ها محسوب می‌شود. همچنین تحقیقاتی مانند [90] با هدف شناسایی این پادالگوها در سیستم‌های مبتنی بر معماری سرویس‌گرا انجام شده‌اند.

---

[12] Top-Down Prespective

[13] Pattern Catalogue

[14] Anti pattern

با توجه به اینکه این معماری به صورت گسترده در سازمان‌های بزرگ مورد استفاده قرار می-گیرد، ارتباطات تنگاتنگی بین معماری سرویس‌گرا و فرایندهای تجاری[۱۵] ایجاد شده است وبا توجه به اهمیت این مقولات در صنعت، کتاب‌هایی مانند [45] با هدف ارائه معماری سازمانی مبتنی بر این دو مقوله نوشته شده‌اند و الگوهایی نیز برای پیاده‌سازی آنها ارائه کرده‌اند. مقالاتی مانند [91] و [92] نیز در مورد الگوها در این زمینه صورت گرفته‌اند.

با توجه به ویژگی‌های مناسب این معماری در مواجهه با شرایط متغیر در شبکه متشکل از سیستم‌های در حال تعامل، روش‌هایی برای استفاده از گونه‌هایی از این معماری در زمینه‌های دیگر نیز صورت گرفته است. در [46]، با هدف بهره‌گیری از دستگاه‌ها توسط سیستم‌های مختلف، الگوهای معماری سرویس‌گرا تحلیل شده و الگوهای قابل اعمال در این زمینه شناسایی شده‌اند. همچنین چند الگوی جدید برای معماری سرویس‌گرا متناسب با این شرایط پیشنهاد شده‌اند.

در سال‌های اخیر، معماری میکروسرویس[۱۶] به عنوان گونه خاصی از معماری سرویس‌گرا ارائه و مورد استقبال صنعت نرم‌افزار قرار گرفته است. هرچند هنوز تحقیقات گسترده‌ای در مورد الگوهای این معماری صورت نگرفته است، در [93] الگوهایی برای پیاده‌سازی این معماری ارائه شده است. همچنین [94]، مجموعه الگویی برای مهاجرت یک سیستم با معماری مشتری-سرویس‌دهنده[۱۷] به معماری میکروسرویس ارائه داده است.

---

[۱۵] Business Process

[16] Micro Services Architecture

[17] Client - Server

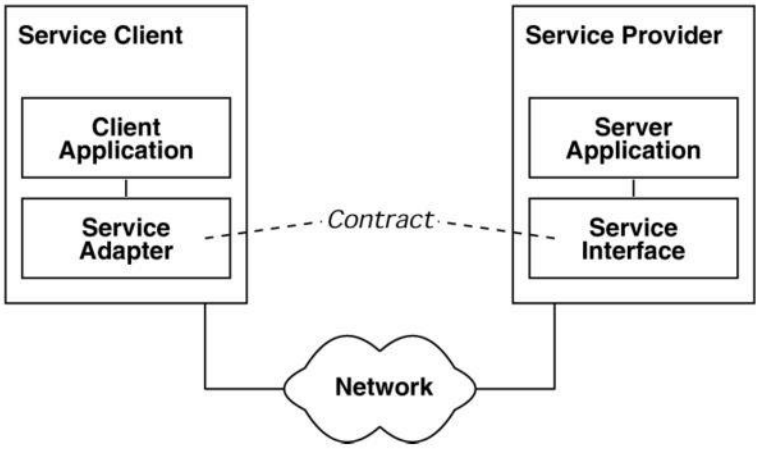
۴.۲ بررسی مبتنی بر الگوی معماری سرویس‌گرا

در [81]، معماری پایه‌ای برای سیستم‌های سرویس‌گرا ارائه شده است که در شکل ۶ نیز دیده شد. در این معماری، الگوی Lookup [95] و [96] نقش مهمی برای جستجوی سرویس‌ها بر اساس شناسه یا مشخصات آنها ایفا می‌کند. توسعه‌دهندگان سرویس‌ها را در یک فهرست سرویس[۱۸] منتشر کرده و با انتساب شناسه‌ای از نوع Object ID یا Absolute Object Reference [95] به هر یک، آنها را قابل شناسایی می‌کنند.

در این مقاله، به این موضوع اشاره شده است که هر سرویس در واقع یک قرارداد بین سرویس-گیرنده و سرویس‌دهنده را، مشابه آنچه در مورد Design by Contract رخ می‌دهد، مشخص می‌کند. در این راستا، الگوی Interface Description [95] برای اعلان ویژگی‌های سرویس‌ها مورد استفاده قرار می‌گیرد و جستجو نیز روی جزئیاتی از سرویس که در آن مشخص شده است، قابل انجام است. برای نظارت بر تحقق برخی از مفاد قرارداد مانند کیفیت سرویس، می‌توان از Invocation Interceptor [95] یا Observer [1] در سرویس‌گیرنده یا سرویس‌دهنده استفاده کرد.

معمولا معماری سرویس‌گرا در نرم افزارهای مشتری-سرویس‌دهنده بزرگی استفاده می‌شوند و SOA فقط برای مقاصد یکپارچه‌سازی با دیگر سامانه‌ها مورد استفاده قرار می‌گیرد. در این صورت،[81] پیشنهاد کرده است که از Service Interface در سمت سرویس‌دهنده و Service Adapter در سمت سرویس‌گیرنده استفاده شود تا مسائل مربوط به ارتباطات را در خود کپسوله‌بندی کنند که این موضوع در شکل۷ دیده می‌شود. Service Adapter با استفاده از الگوی Remote Proxy [1] قابل تحقق است.

[18] Service Directory

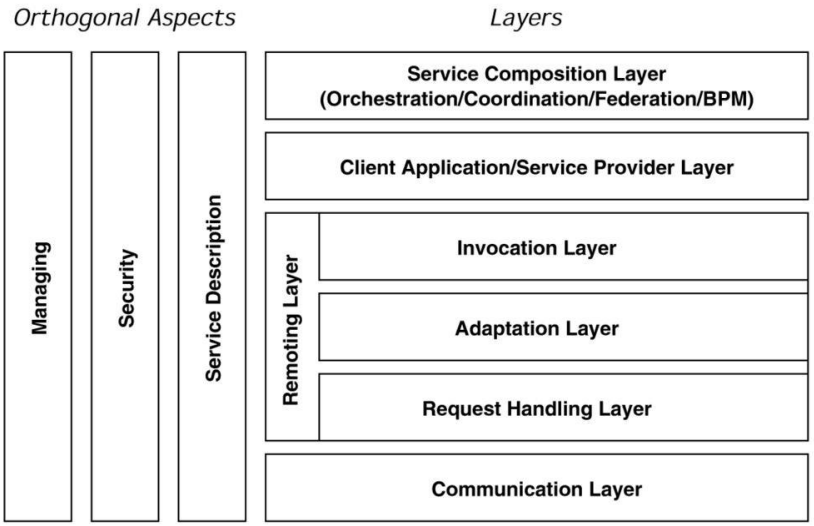


شکل ۷- service Interface و Adapter [81]

شکل ۸ معماری لایه‌ای یک سیستم مبتنی بر SOA را نمایش می‌دهد [81]. نگاه عمیقی به لایه Remoting که عملکردهای میان‌افزاری[19] مانند چارچوب[20]های وب‌سرویس در SOA را در بر می‌گیرد، داشته و الگوهای مورد استفاده در آن را بر شمرده است. به طور کلی، معماری Broker [5] در این لایه، جزئیات سمت سرویس‌گیرنده و سرویس‌دهنده را مخفی می‌کند و همه ارتباطات بین بخش‌های مختلف سیستم را بر عهده می‌گیرد. لایه Invocation که مسئول marshalling و multiplexing درخواست‌ها است، با استفاده از الگوهای Client Proxy، Requester و Invoker [95] پیاده‌سازی می‌شود. در لایه Adaptation، الگوی Invocation Interceptor [95] مسئول انطباق‌دادن فراخوانی‌ها و پاسخ‌ها در جریان پیام‌هاست. در انتها نیز لایه Request Handling، با برقراری الگوهای Client/Server Request Handler [95]، مسئول برقراری ارتباط بین سرویس‌دهنده و مشتری و انتقال پیام‌ها بین این دو است.

[19] Middleware

[20] Framework



شکل ۸-لایه‌های معماری سرویس‌گرا [81]

۵.۲ الگوهای معماری سرویس‌گرا

در [82]، معماری سرویس‌گرا به ۴ محدوده پیاده‌سازی تقسیم شده است:

۱) معماری سرویس[۲۱] که به طراحی و پیاده‌سازی داخلی هر سرویس می‌پردازد.

۲) معماری ترکیب سرویس[۲۲] که بر معماری لازم برای سرهم‌کردن[۲۳] سرویس‌ها در راستای تشکیل سرویس‌های مرکب تمرکز دارد.

۳) معماری فهرست سرویس[۲۴] که به معماری لازم برای پشتیبانی از مجموعه‌ای از سرویس-های مرتبط می‌پردازد که هر یک جداگانه استانداردسازی و مدیریت می‌شوند.

۴) معماری سازمان سرویس‌گرا[۲۵] که درگیر با معماری سازمانی است که برای توسعه نرم-افزارهای آن چنین معماری را برگزیده است.

---

[21] Service Architecture

[22] Service Composition Architectuer

[23] Assemble

[24] Service Inventory Architecture

در ادامه این بخش، الگوهای موجود برای معماری سرویس در معماری سرویس‌گرا را بررسی می‌کنیم.

۶.۲ الگوهای پایه‌ای سرویس

در این بخش، الگوهای پایه ای برای طراحی، تقسیم وظایف و میزبانی سرویس‌ها مطرح می‌شوند.

۱.۶.۲ Functional Decomposition

اکثر وظایف و فرایندهای سازمانی، مسائل بزرگی هستند. پیش از همه‌گیر شدن سیستم‌های توزیع‌شده، همه سیستم‌های نرم‌افزاری به صورت stand alone پیاده‌سازی می‌شدند و هر یک، همه وظایف عملکردی لازم در خود را جداگانه پیاده‌سازی می‌کردند. با برقراری اصل Seperation of Concerns درسطح سازمان، وظایف بزرگ به بخش‌های کوچک شکسته شده و می‌توان به مجموعه‌های عملکردی هم‌بسته‌ای دست یافت که لزوم تکرار برخی عملکردهای پرتکرار را در هر یک از سیستم‌ها از بین می‌برد. با توسعه سرویس‌های مرتبط به هریک از این مجموعه‌ها طبق معماری سرویس‌گرا، این سرویس‌ها با همکاری با یکدیگر نیازهای کلی سازمان را برطرف خواهند کرد. [۸۳]

۲.۶.۲ Service Encapsulation

منطق‌ها و عملکردهایی که در هریک از سیستم‌های مرتبط پیاده‌سازی شده‌اند، برای استفاده در دیگر سیستم‌ها محدودیت دارند. این الگو به دنبال بسته‌بندی منطق‌ها و عملکردهای قابل استفاده مجدد به صورت سرویس‌هایی است که قابل دسترسی و استفاده توسط کل مجموعه باشند. هر یک از این مجموعه سیستم‌ها، منابع سازمانی محسوب می‌شوند. هر چند، پس از شناسایی و استخراج این بخش‌ها از سیستم، نیاز به اعمال دیگر الگوها در راستای برقراری

[25] Service Oriented Enterprise Architecture

شرایطی خواهیم داشت که سرویس‌های استخراج‌شده بتوانند همکاری مناسبی با دیگر سیستم‌ها داشته باشند. [۸۳]

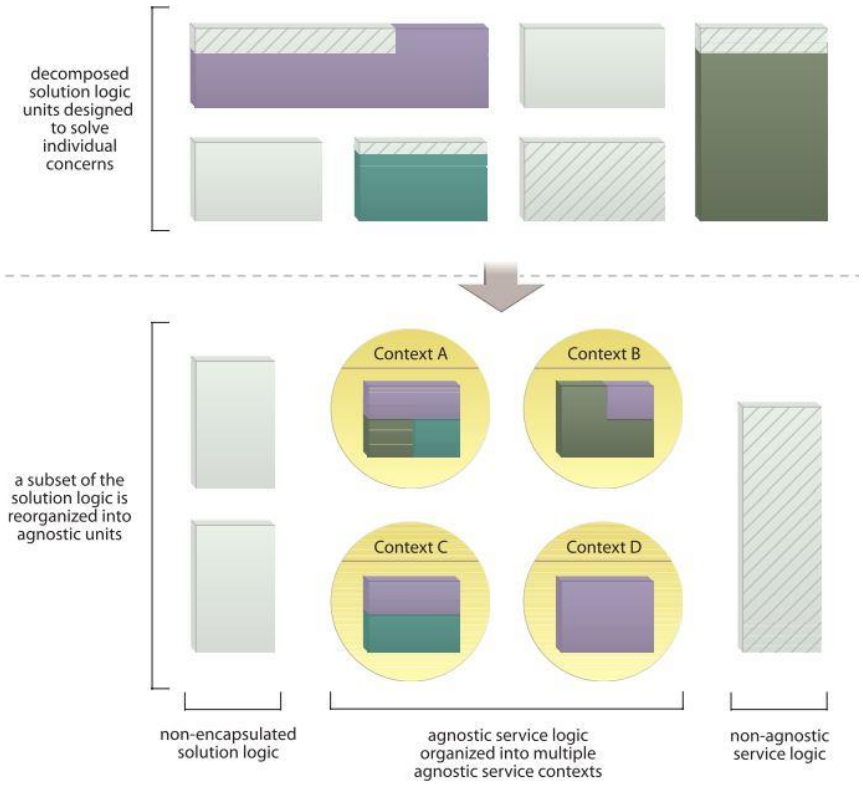
۳.۶.۲ Service Host

هر سرویس، برای انتشار و سرویس‌دهی نیازمند فعالیت‌ها و روندهای مشترکی است. فعالیت‌هایی مانند برنامه‌ریزی listener ها برای دریافت پیام‌ها و درخواست‌ها، فعال‌سازی و انجام تنظیمات زمان اجرای سرویس از جمله این فعالیت‌های مشترک است. برای جلوگیری از تکرار و بازنویسی این گونه فعالیت‌ها در همه سرویس‌هایی که نوشته می‌شوند، این الگو ساختن یک چارچوب یا مولفه ای به عنوان میزبان سرویس[۲۶] را پیشنهاد می‌کند که یک container برای سرویس‌ها خواهد بود و وظایف و قابلیت‌های مورد نیاز برای تنظیمات اولیه، ارسال و دریافت پیام‌ها و تنظیمات زمان اجرا را به سرویس‌ها ارائه می‌دهد. چارچوب‌های ارائه‌شده برای وب‌سرویس‌ها در زبان‌های مختلف به همراه سیستم‌های تجاری و متن باز میزبانی وب، نمونه‌هایی از این الگو هستند. [۸۵]

۴.۶.۲ Agnostic Context

منطق پیاده‌سازی‌شده برای حل یک مسئله خاص، معمولا شامل بخش‌هایی خواهد شد که قابل استفاده برای حل مسائل دیگری نیز هستند. گروه‌بندی بخش‌های تک منظوره با بخش‌های چند منظوره این منطق، امکان استفاده مجدد از منطق چندمنظوره را از بین می‌برد. این الگو با جداسازی بخش‌های چندمنظوره سرویس‌ها و تبدیل آن‌ها به سرویس‌های جداگانه، امکان استفاده مجدد را فراهم می‌آورد. منطق تک منظوره، یکی از مشتریان سرویس جدید خواهد بود. هرچند قابلیت استفاده مجدد در این الگو افزایش می یابد، معرفی سرویس جدید، پیچیدگی کل سیستم را افزایش خواهد داد. [۸۳]

[۲۶] Service Host



شکل ۹- الگوی Agnostic Context [۸۳]

۵.۶.۲ Non-Agnostic Context

بخش‌هایی از وظایف سیستم‌ها که مرتبط به زمینه خاصی هستند و چندان قابل استفاده مجدد نیستند، معمولا از معماری سرویس‌گرا بیرون می‌افتند و بر عهده مشتریان نهایی سیستم گذاشته می‌شوند. چون این بخش‌ها، ارتباطات زیادی با بخش‌های دیگر سازمان دارند که در قالب معماری سرویس‌گرا جا گرفته‌اند، استفاده از آن‌ها خارج از این مجموعه، ماهیت اصلی و ویژگی‌های مثبتی که این معماری ایجاد کرده را زیر سوال خواهند برد و از تبدیل این گونه وظایف به منابع سازمانی که در آینده استفاده مجدد خواهند شد، جلوگیری می‌کند. بنابراین، طبق این الگو، این گونه وظایف توسط سرویس‌های جدیدی بسته‌بندی می‌شوند و آنها نیز بخشی از منابع سازمانی به شمار می‌روند و قابل استفاده مجدد در شرایط پیش‌بینی‌نشده بعدی خواهند بود. [۸۳]

۶.۶.۲ Agnostic Capability

وظایف سرویس‌هایی که از دل مسائل خاصی بیرون آمده‌اند، ممکن است آن‌گونه که در ابتدا تصور شده بودند، چندان برای سرویس‌گیرندگان متعدد سودمند نباشند و شرایط استفاده خاصی شبیه به مسئله اولیه خود را لازم داشته باشند. طبق این الگو، وظایف سرویس‌های چندمنظوره به وظایف ریزدانه‌تری شکسته شده و هر یک از آن‌ها در قالب سرویس مجزایی در اختیار کل سازمان قرار می‌گیرند. این شکسته‌شدن سرویس‌ها در صورت امکان، به صورت بازگشتی برای سرویس‌های جدید نیز اعمال می‌شود. به این ترتیب، هر یک از سرویس‌های ریزدانه جدید، مسئله خاصی را که *واقعا* مشترک و همه‌گیر است را حل می‌کند و با سرویس‌گیری از مجموعه‌ای از آن‌ها، مسائل متنوع‌تر و بیشتری قابل حل خواهند بود. [۸۳]

۷.۶.۲ Service Façade

هر سرویس، بدنه‌ای شامل منطق اصلی خود است که وظایف سرویس را بر عهده دارد. وقتی سرویس تحت تاثیر تغییرات در قراردادها یا پیاده‌سازی‌های دیگر منابع سازمانی (دیگر سرویس‌ها) قرار گرفته و نیازمند تغییرات و تکامل می‌شود، ممکن است تغییرات را به سرویس‌گیرندگان خود نیز منتقل کرده و آنها را تحت تأثیر منفی قرار دهد. در این موارد، Facade برای برقراری یک یا چند سطح انتزاع به معماری سرویس اضافه می‌شود که می‌تواند تغییرات آتی در قراردادها، منطق سرویس و پیاده‌سازی سرویس را از دید سرویس‌گیرندگان مخفی نگه دارد. در معماری جدید، منطق هسته سرویس مسئول ارائه مجموعه عملیات‌های لازم برای انجام وظایف توصیف‌شده در قرارداد سرویس است و منطق Facade پردازش‌های تکمیلی و میانی برای پشتیبانی از منطق هسته خواهد بود. [۸۳]

۸.۶.۲ Workflodize

تغییرات در نیازمندی‌های تجاری در اکثر سیستم‌ها امری معمول است. این الگو به دنبال به حداقل رساندن هزینه اِعمال تغییرات مربوط به فرایندهای تجاری در سیستم‌ها است. طبق الگو، یک موتور Workflow به سرویس اضافه می‌شود تا فرایند تجاری را پیاده‌سازی کند. استفاده از

این موتور، تطبیق پذیری زیادی به سرویس اضافه کرده و هزینه اعمال تغییرات را که معمولا مکررا لازم می‌شود، به حداقل خواهد رساند. [۸۵]

۹.۶.۲ Redundant Implementation

سرویس‌های چندمنظوره‌ای که قابلیت استفاده مجدد بالایی دارند، معمولا توسط سرویس‌های مختلفی در ترکیب‌های زیادی از وظایف مورد استفاده قرار می‌گیرند. بدین ترتیب، در صورت بروز مشکلات در این سرویس‌ها، تبدیل به نقطه شکست یگانه[۲۷] در همه این ترکیبات خواهند شد. با توجه به اینکه ترکیبات سرویس‌ها، معمولا از تعداد قابل توجهی از این سرویس‌های پراستفاده تشکیل می‌شوند، هر ترکیب می‌تواند شامل چندین نقطه شکست باشد. بنابراین، طبق این الگو، برای این گونه سرویس‌های پراستفاده، نمونه‌های متعدد از هر یک می‌تواند پیاده‌سازی و بارگذاری شود تا قابلیت دسترسی و اطمینان در این نقاط حساس افزایش یابد و در صورت بروز مشکلی در هر یک از این نمونه‌های یکسان، دیگری در دسترس مجموعه قرار داشته باشد. البته همگام‌سازی نمونه‌های تکراری این سرویس‌ها، نیازمند صرف هزینه‌های مدیریتی و نظارتی در سیستم خواهد بود. [۸۳]

۱۰.۶.۲ Service Data Replication

قدم‌های زیادی برای خودمختار کردن سرویس‌ها در این معماری صورت می‌گیرد. تا حد امکان سعی می‌شود سرویس‌ها به صورت کاملا استقرار یابند و بتوانند مستقل از یکدیگر کار کنند. با این حال، معمولا حتی ایزوله‌ترین سرویس‌ها نیز نیاز دارند با یک پایگاه‌داده مرکزی برای دسترسی و بروز رسانی داده‌های تجاری در ارتباط باشند. این داده‌ها معمولا بین بخش‌های بزرگی از سیستم به اشتراک گذاشته شده‌اند. برای حل این مشکل، با حفظ استقلال سرویس‌ها تا حد ممکن، این الگو پیشنهاد می‌کند هر یک از سرویس‌ها پایگاه داده اختصاصی خود را برای نگهداری داده‌های منحصر به فرد خود داشته باشد و داده‌های مرکزی نیز از طریق تکرار[۲۸] به

[27] Single Point of Failure

[28] Replication

واسطه همین پایگاه‌داده در اختیار و دسترس سرویس گذاشته شوند. استفاده از این الگو در سطح گسترده در معماری سیستم، کانال‌های متعدد تکرار ایجاد خواهد کرد که منابع زیرساختی مضاعف را به سیستم تحمیل می‌کنند و مدیریت آنها سخت خواهد شد. [۸۳]

۱۱.۶.۲ Active Service

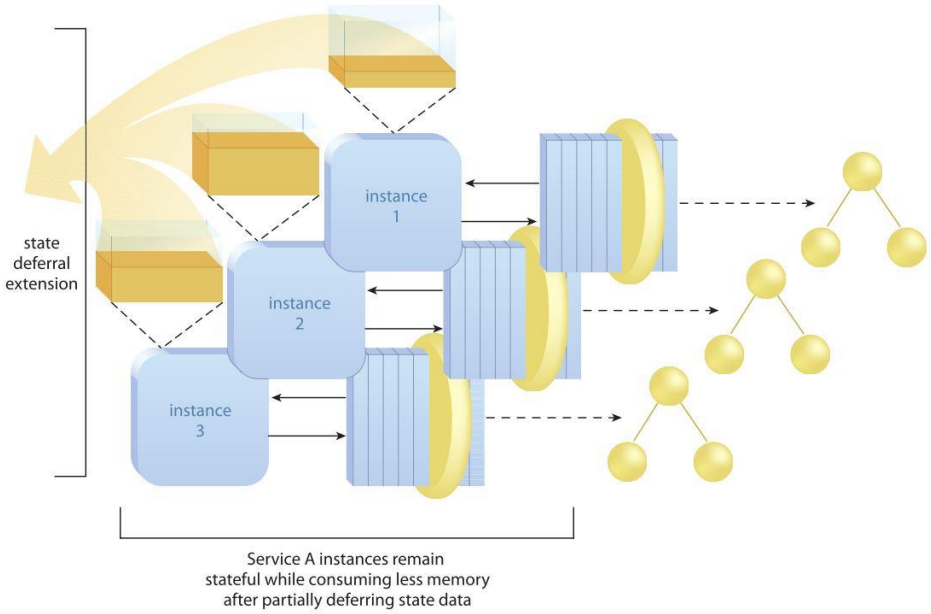
سرویس‌ها معمولا برای سرویس‌دهی نیازمند همکاری با یکدیگر هستند. زمانی که یک سرویس فراخوانی می‌شود، ممکن است برای پاسخگویی نیاز به داده‌ها و اطلاعاتی داشته باشد که توسط سرویس‌های دیگری تامین می‌شوند. در صورتی که سرویس دوم، در لحظه نیاز، در دسترسی نباشد یا با بار زیادی مواجه باشد، پاسخگویی سرویس اول به کلی به مشکلی خورده یا به تعویق خواهد افتاد. برای حل مشکلاتی از این دست، این الگو با به کارگیری مفهوم Active Class در طراحی شی‌گرا پیشنهاد می‌دهد هر سرویس، شامل حداقل یک کلاس فعال باشد که وظیفه جمع آوری اطلاعاتی از این دست که از سرویس‌های دیگر باید گرفته‌شوند را بر عهده دارد. این کلاس، فارغ از فراخوانی سرویس فعال است و در یک ریسه کارگر[۲۹] به صورت دوره‌ای در حال فراخوانی سرویس‌های ثانویه و جمع‌آوری داده‌های مورد نیاز سرویس اصلی است. به این ترتیب، در زمان فراخوانی سرویس اصلی، نگرانی‌ای از بابت دسترسی به داده‌های دیگر سرویس‌ها وجود ندارد و سرویس داده‌های پیش‌آماده را استفاده کرده و خودمختار خواهد بود. [۸۵]

۱۲.۶.۲ Partial State Deferal

در مواقعی که فراخوانی سرویس به عنوان جزئی از یک فرایند بزرگ صورت می‌گیرد، معمولا نیاز می‌شود سرویس در حین انجام فعالیت‌ها توسط دیگر عناصر سیستم، stateful باشد و وضعیت خود را حفظ کرده و فعال باقی بماند. اگر داده‌های مربوط به وضعیت سنگین بوده و تعدد درخواست‌های فعال در یک سرویس بالا رود، سرویس با مشکلات حافظه درگیر شده و از سرویس‌دهی خارج خواهد شد. با این‌حال، لزومی ندارد سرویس همه داده‌های وضعیت را در

[29] Worker Thread

حافظه خودش نگه دارد. طبق این الگو، سرویس مسئولیت نگهداری و مدیریت زیرمجموعه‌ای از داده‌های وضعیت خود را بر عهده بخش دیگری از سیستم (State Repository) می‌گذارد. بنابراین، سرویس با صرف منابع سیستمی کمتری، stateful باقی می‌ماند و داده‌های وضعیتی را که واگذار کرده، در مواقع لزوم باز پس خواهد گرفت. البته پیاده‌سازی این الگو، باعث وابستگی عمیق بین سرویس و بخش‌های دیگر سیستم می‌شود که سرویس را از حالت خودمختار خارج کرده و سربارهای مدیریتی و انتشار تغییرات را باعث می‌شود. [۸۳]



شکل ۱۰- پیاده‌سازی الگوی Partial State Deferal [۸۳]

۱۳.۶.۲ Partial Validation

وظایف عمومی که بر عهده سرویس‌های چندمنظوره قرار داده می‌شود، معمولا قراردادهایی را با مشتریان ایجاب می‌کنند که داده‌ها و اعتبارسنجی‌های بی‌مورد را به آنها تحمیل خواهند کرد. به طور مثال، وقتی سرویسی خیلی عام‌منظوره نوشته شده است و مجموعه بزرگی از داده را برمی‌گرداند، لزوما همه این مجموعه، مورد نیاز همه مشتریانش نخواهد بود و احتمالا تعدادی از مشتریان فقط به بخش کوچکی از این داده‌ها نیاز دارند. با این حال، طبق قرارداد سرویس، مشتری همه این داده‌ها را دریافت کرده و ملزم به اعتبار سنجی همه داده، حتی آن بخشی که

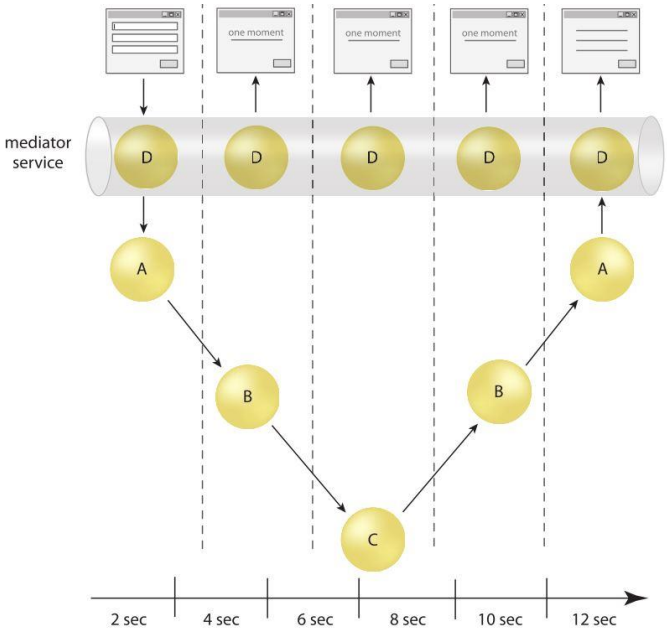
اصلا مورد استفاده‌اش نخواهد بود، می‌شود. این الگو به مشتری امکان می‌دهد به صورت خودآگاه، به طور کامل قرارداد سرویس را برآورده نکند و صرفا به اعتبارسنجی بخشی از داده بپردازد که مورد استفاده‌اش است و بخش‌های بی‌استفاده را رها کند. به این ترتیب، نیازهای پردازشی سرویس‌گیرنده کاهش یافته و وابستگی او به بخش‌های نامربوط قرارداد سرویس از بین می‌رود. [۸۳]

UI Mediator ۱۴.۶.۲

راهکارهای سرویس‌گرا، به طور معمول به صورت ترکیبی از سرویس‌هایی با رفتارهای زمان اجرا و نیازمندی‌های پردازشی متفاوت طراحی می‌شوند. اگر فرایند سرویس‌گرا، توسط کاربر نهایی از طریق یک رابط کاربری فعال شود، کیفیت تجربه کاربری[۳۰] پایین خواهد بود. به دلیل این‌که احتمالا برخی از این سرویس‌ها قادر به پاسخ همگام و در زمان قابل قبول نیستند. این الگو، با واسط قراردادن یک Mediator بین رابط کاربری و سرویس‌های مورد استفاده، سعی در برطرف‌کردن این مسئله دارد. این واسط مسئول ارائه بازخورد مداوم از مراحل فرایند و نحوه انجام کار به کاربر بوده و شرایط مختلفی را که در زمان اجرا ممکن است پیش بیایند، مدیریت کرده و اجازه نمی‌دهد این تغییرات در شرایط، تجربه کاربری را تحت تاثیر قرار دهند. این الگو به دو روش قابل پیاده‌سازی است: یکی با تعریف یک سرویس جدید برای Mediator و دیگری تبدیل Mediator به یک عامل[۳۱] سرویس با کمک گرفتن از معماری‌های Agent Oriented. [۸۳]

[30] User Experience (UX)

[31] Agent



شکل ۱۱- نقش UI Mediator [۸۳]

۷.۲ الگوهای امنیت سرویس

۱.۷.۲ Exception Shielding

در صورت بروز شرایط استثنا[۳۲] در پیاده‌سازی سرویس، معمولا سرویس یک پیام در پاسخ فراخوانی به سرویس‌گیرنده ارسال می‌کند تا بروز مشکل را اطلاع دهد. این پیام ممکن است سهواً حاوی اطلاعات محرمانه‌ای باشد که بتواند برای حمله به سرویس و محیط پیرامون آن مورد استفاده قرار گیرد. برای جلوگیری از این مشکلات، اطلاعات مربوط به مورد استثنا تصفیه[۳۳] شده و بخش‌هایی از اطلاعات که حاوی اطلاعات محرمانه هستند، با اطلاعاتی که امن هستند، جایگزین می‌شوند. نمونه‌های این گونه اطلاعات داده‌های حساس و حیاتی، Stack Trace نرم افزار هستند که افشای آنها اطلاعات زیادی در مورد ساختار درونی سیستم را افشا

[32] Exception

[33] Sanitize

می‌کند. هرچند، این الگو با حذف امکان ردگیری خطا از بیرون از سرویس و توسط سرویس‌گیرنده، کار را برای حل مشکلات سرویس دشوار می‌کند و برای حل این مشکل، پیشنهاد می‌شود امکان فعال‌سازی و غیر فعال کردن این ویژگی در زمان اجرا برای سرویس‌ها وجود داشته باشد. [۸۳]

۲.۷.۲ Message Screening

ممکن است پیام‌های ارسال شده به سرویس، حاوی داده‌های نامعتبر و مخرب باشند که باعث رفتار نامطلوبی در سرویس یا سیستم‌های زیرینی شود که پیام‌های دریافتی را پردازش می‌کنند. این داده ممکن است به طور اشتباه توسط سرویس‌گیرنده یا عمداً توسط یک مهاجم برای حمله به سرویس فرستاده شده باشد. طبق این الگو، سرویس‌ها باید همه پیام‌های دریافتی را خطرناک و نامعتبر تلقی کنند، مگر اینکه خلاف این موضوع مشخص شود. توابع و روال‌های مختص نظارت بر پیام‌های دریافتی به سرویس اضافه می‌شوند تا قبل از انجام هر عمل پردازشی توسط سرویس، صحت و اعتبار پیام دریافتی را بررسی کنند. این روال‌های نظارتی، برای هر نوع داده دریافتی، فرایندهای مختلفی را شامل می‌شود که ممکن است مانند بررسی داده‌های باینری مثل فایل‌ها، پیچیدگی زیادی داشته باشند. [۸۳]

۳.۷.۲ Trusted Subsystem

اگر یک مصرف‌کننده، به صورت مستقیم به منابع backend یک سرویس مانند پایگاه‌داده آن دسترسی داشته باشد، صحت[۳۴] آن منبع را زیر سوال برده و همچنین وابستگی‌های نامطلوبی بین او و این منابع ایجاد خواهد کرد. طبق این الگو، دسترسی به منابع زیرین سرویس‌ها فقط از طریق خود سرویس امکان‌پذیر است و سرویس فقط از گواهی[۳۵]های خود برای کار با منابع خود استفاده می‌کند، و نه از گواهی‌های مصرف‌کنندگان. سرویس تنها مسیر و ابزار دسترسی به

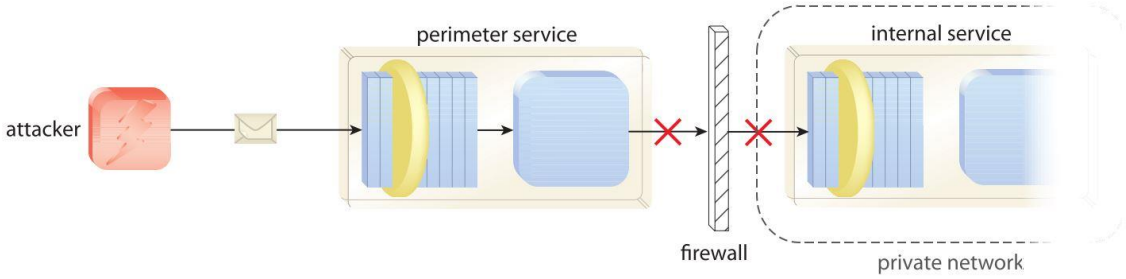
[34] Integrity

[35] Credential

منابع زیرساختی خود خواهد بود. اشکال اصلی این الگو آن است که به خاطر دسترسی زیاد این سرویس‌ها، در صورت حمله مهاجمین یا دسترسی‌های غیرمجاز، مهاجم به حجم وسیعی از منابع سرویس دسترسی پیدا خواهد کرد. [۸۳]

۴.۷.۲ Service Perimiter Guard

گاهی نیاز است سرویسی که در یک شبکه خصوصی قرار دارد، به دیگر سرویس‌های خارج از این شبکه سرویس‌دهی کند. در صورتی که قرار باشد ارتباط مستقیم بین مصرف‌کنندگان و این سرویس برقرار شود، مصرف‌کنندگان باید به شبکه محلی این سرویس دسترسی پیدا کنند و این باعث می‌شود دیگر سرویس‌ها و سیستم‌های موجود در شبکه، در معرض حملات و دسترسی‌های غیر مجاز قرار گیرند. طبق این الگو، یک سرویس میانجی، در مرز شبکه خصوصی قرار می‌گیرد که نقطه اتصال یگانه‌ای بین مصرف‌کنندگان خارجی و سرویس‌های داخلی شبکه خواهد بود. [۸۳]



شکل ۱۲- اجرای الگوی Service Perimiter [۸۳]

۸.۲ الگوهای قرارداد سرویس

۱.۸.۲ Decoupled Contract

سرویس‌ها با استفاده از تکنولوژی‌های توسعه مولفه‌محور مثل java و .NET. پیاده‌سازی می‌شوند. این تکنولوژی‌ها معمولا نیازمند این هستند که قرارداد فنی[۳۶] سرویس، به صورت فیزیکی به منطق زیرین سرویس متصل باشد. این باعث می‌شود قرارداد سرویس، با همان تکنولوژی‌هایی که سرویس با آنها پیاده‌سازی شده توصیف شوند و در نتیجه، فقط مصرف‌کنندگانی که با آن تکنولوژی سازگارند، قادر به استفاده از سرویس خواهند بود. این مسئله، قابلیت استفاده مجدد سرویس را به شدت محدود می‌کند و وابستگی مربوط به تکنولوژی بین مصرف‌کننده و سرویس ایجاد می‌کند. طبق این الگو، قرارداد سرویس به صورت فیزیکی جدا از هسته سرویس پیاده‌سازی می‌شود. این کار، وابستگی قرارداد به پیاده‌سازی سرویس را حذف کرده و اجازه می‌دهد به صورت مستقل طراحی و مدیریت شود. [۸۳]

۲.۸.۲ Contract Centralization

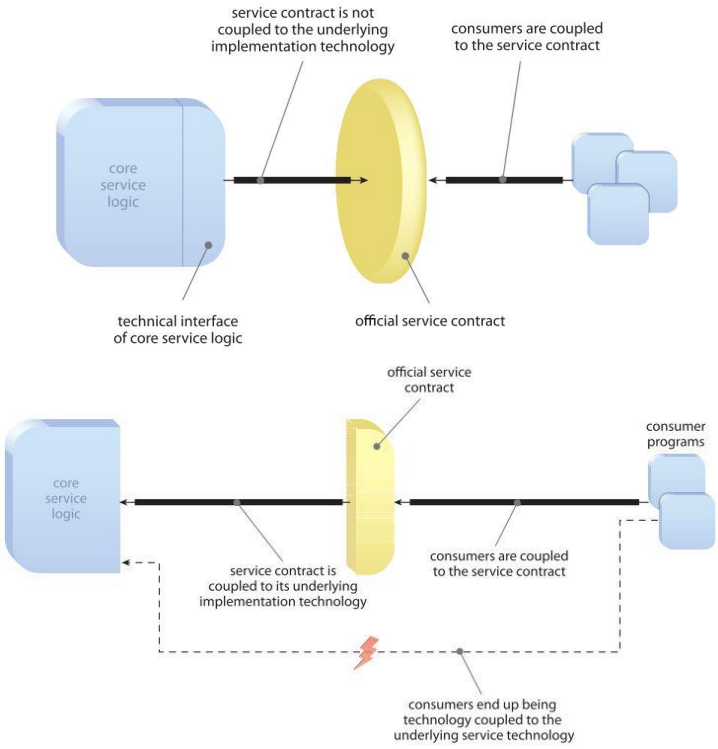
حتی در مواقعی که سرویس‌ها در یک سازمان به همراه قراردادهای استاندارد منتشر می‌شوند، افرادی که برنامه‌های مصرف‌کننده را طراحی می‌کنند، همواره به دنبال نقاط ورود [۳۷] جایگزینی برای دسترسی به منطق هسته سرویس هستند. چرا که ممکن است دسترسی از این طریق برای آنها ساده‌تر یا سریع‌تر باشد. در صورتی که این اتفاق روی دهد، قرارداد سرویس عملا ماهیت خود را از دست داده و وابستگی‌های زیادی به طور مستقیم با سرویس به وجود می‌آید که تکامل و مدیریت سرویس را تحت اثرات منفی قرار داده و اهداف اصلی معماری سرویسگرا را زیر

---

[۳۶] Technical Contract

[37] Entry Point

سوال می‌برد. این الگو، یک استاندارد طراحی را به سازمان دیکته می‌کند که طبق آن، قرارداد سرویس، تنها نقطه ورود به هسته منطق سرویس خواهد بود. [۸۳]



شکل ۱۳- قبل و بعد از اجرای الگوی Decoupled Contract [۸۳]

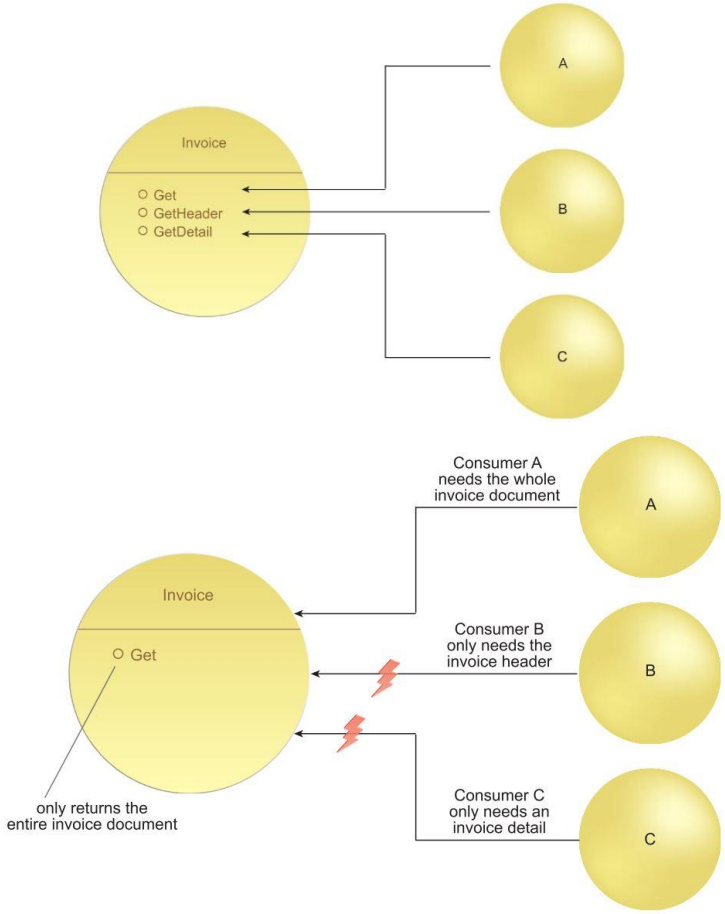
۳.۸.۲ Contract Denormalization

سرویس‌ها می‌توانند در زمینه‌های مختلف و متفاوت ترکیب و مورد استفاده قرار گیرند. به همین دلیل، این‌که همه سرویس‌ها به گونه‌ای توصیف شوند که برای همه مصرف‌کنندگان آنها مناسب باشد، دشوار است. به طور مثال مانند آنچه در شکل ۱۴ دیده می‌شود، ممکن است یک مصرف‌کننده، کل داده‌های مربوط به یک موجودیت را نیاز داشته باشد و دیگری فقط به بخش خاصی از آن نیازمند باشد. در صورتی که فقط یک وظیفه و نقطه ورود برای سرویس‌ها در نظر گرفته شود، برای پاسخگویی به بسیاری از مصرف‌کنندگان پردازش‌های بیهوده‌ای هم در سمت سرویس‌دهنده و هم مشتری صورت می‌گیرد که باعث هدررفت منابع پردازشی سازمان می‌شود. در این الگو، بر این مسئله تاکید شده است که لزومی ندارد هر سرویس دقیقا یک وظیفه و

عملکرد داشته باشد و فقط یک نقطه دسترسی در اختیار مشتریان خود قرار دهد؛ بلکه می‌تواند برای نیازمندی‌های متفاوتی که از سوی مشتریان وجود دارد، در سطوح مختلف ریزدانگی نیازهای آن‌ها، وظایف مختلفی ارائه دهد. بدین ترتیب پردازش‌های بیهوده برای مشتریانی که نیاز به بخش خاصی از عملکرد سرویس دارند، صورت نخواهد گرفت. البته استفاده بیش از حد از این الگو در یک قرارداد سرویس خاص، باعث وسعت بیش از حد سرویس شده و مدیریت و نگهداری آن را پرهزینه و پیچیده می‌کند. [۸۳]

۴.۸.۲ Concurrent Contracts

به صورت پیش‌فرض، قرارداد یک سرویس، تمام قابلیت‌ها و وظایف سرویس را توصیف می‌کند. اما طراحی قراردادی که برای همه انواع مصرف‌کنندگان سرویس مناسب باشد، کار سختی است. ممکن است لازم باشد برخی قابلیت‌های سرویس، صرفا برای مصرف‌کنندگان مورد اعتماد خاصی قابل بهره‌برداری باشد، یا سرویس‌دهی به برخی مصرف‌کنندگان نیازمند سیاست‌های خاصی باشد. این الگو، برای حل این دغدغه‌ها، قراردادهای متعددی برای یک سرویس خاص متصور شده است. به این ترتیب، هر قرارداد توسط مجموعه خاصی از مشتریان مورد استفاده قرار گرفته و نیازهای آن مجموعه را متناسب با سیاست‌های سرویس برطرف می‌کند. هر قرارداد نیز به طور جداگانه قابل نظارت خواهد بود. هر قرارداد جدید، یک endpoint به مجموعه اضافه می‌کند که سربار نظارتی آنها از نقاط ضعف این الگو محسوب می‌شود. [۸۳]

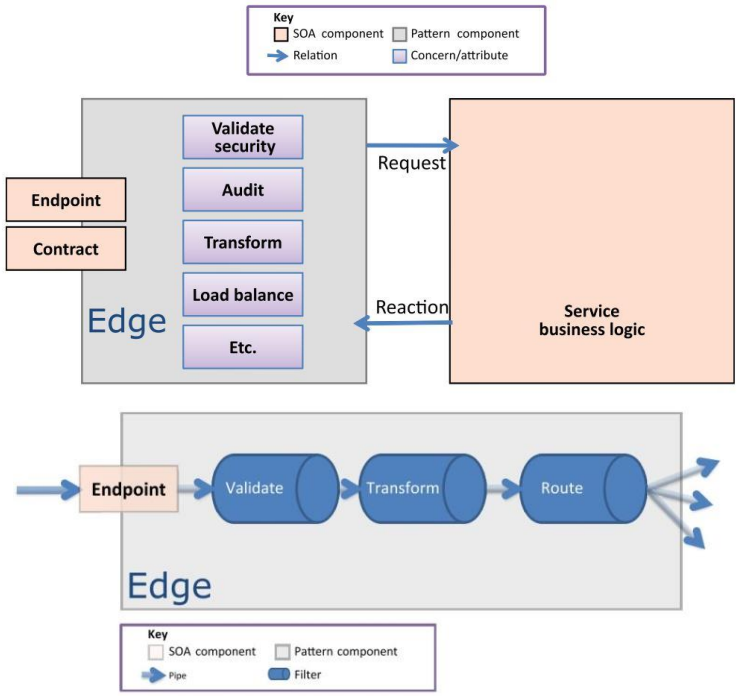


شکل ۱۴- قبل و بعد از پیاده‌سازی الگوی Contract Denormaliztion [۸۳]

۵.۸.۲ Edge Component

در برخی موارد لازم است یک منطق یکسان سرویس، از طریق پروتکل‌ها و تکنولوژی‌های متنوعی مورد استفاده قرار گیرد. همین‌طور ممکن است پیاده‌سازی یک سرویس، بدون تغییرات در واسط ارائه شده‌اش به مصرف کنندگان تغییر یابد. گاهی نیز برخی عملکردهای عمومی مانند اعتبارسنجی درخواست، یا لاگ‌گرفتن از پیام‌ها در همه سرویس‌های یک مجموعه پیاده‌سازی شوند. در این گونه موارد، برای ایجاد امکان تکامل جداگانه ویژگی‌های تجاری سرویس، دغدغه‌های مرتبط با تکنولوژی و عملکردهای عمومی مورد انتظار در همه سرویس‌ها، با کمک اصل Seperation of Concerns در طراحی شی‌گرا، دغدغه‌های فراتر از منطق اصلی خود سرویس، بر عهده Edge Component

قرار می‌گیرند. منطق تجاری سرویس نیز بر عهده مولفه دیگری است که تنها همین دغدغه را داراست. در واقع، Edge، مانند Facade یا Proxy بر روی پیاده‌سازی سرویس عمل می‌کند. این مولفه، همه وظایف عمومی را بر عهده می‌گیرد، مانند نظارت و لاگ گرفتن، انواع خاص Endpoint و قراردادهای متفاوت که جزئی از منطق سرویس نیستند. [۸۳]



شکل ۱۵- عملکرد و پیاده سازی الگوی Edge Component [۸۳]

۶.۸.۲ Validation Abstraction

با توصیف منطق اعتبارسنجی وب‌سرویس‌ها با استفاده از امکانات قوی‌ای که شِماهای XML و WS-Policy در اختیار می‌گذارند، سرویس‌ها از دغدغه‌های مربوط به اعتبارسنجی انتزاع شده و نیازی به بررسی پیام‌های دریافتی نخواهند داشت. اما زمانی که قوانین تجاری که پایه و اساس این‌گونه اعتبارسنجی‌ها بوده‌اند، تغییر کنند، امکان منطبق‌شدن قرارداد با منطق‌های جدید بدون انتشار نسخه جدید قرارداد وجود ندارد. تغییرات در قرارداد، بر مصرف‌کنندگان فعلی سرویس تاثیرگذار بوده و نیازمند هزینه‌های سازمانی خواهد بود. این الگو، به دنبال کاهش درشت‌دانگی محدودیت‌های تعریف شده در قرارداد سرویس‌ها است. بررسی محدودیت‌های

برداشته‌شده از قرارداد بر عهده دیگر اجزای سیستم (مانند هسته اصلی سرویس، یک سرویس دیگر یا یک عامل سرویس) قرار می‌گیرد تا ریسک تغییر در قرارداد به واسطه تغییرات قوانین تجاری کاهش پیدا کند. هر چند با این کار، منطق اعتبارسنجی از متمرکزبودن خارج شده و در بخش‌های متفاوتی گسترده خواهد شد. [۸۳]

۹.۲ الگوهای مدیریت سرویس

۱.۹.۲ Compatible Change

پس از انتشار یک سرویس، سرویس به عنوان یک منبع سازمانی در اختیار کل سازمان قرار می‌گیرد و مصرف‌کنندگانی شروع به استفاده از آن می‌کنند. بنابراین وابستگی بین قرارداد سرویس و مصرف‌کنندگان ایجاد خواهد شد. وقتی قرارداد سرویس نیازمند تغییرات شود، این تغییرات مصرف‌کنندگان فعلی را که به قرارداد قبلی وابسته هستند را در معرض ریسک تغییرات قرار می‌دهد. طبق این الگو، سعی می‌شود در مواقع ایجاد تغییر در قرارداد سرویس‌ها، تا حد امکان سازگاری با قبل[۳۸] حفظ شود تا مصرف‌کنندگان نسخه قبلی، تحت تاثیر قرار نگیرند. این امر در صورت تغییر نام قابلیت قبلی، یا ایجاد قابلیت جدید با اضافه کردن این موارد به قرارداد به سادگی قابل دستیابی است. هر چند در مواردی مانند حذف یک قابلیت، دستیابی به آن غیر ممکن است یا پیچیده‌تر می‌شود. انطباق با نسخه‌های پیشین، زحمات مدیریت سیستم را افزایش می‌دهد و در صورتی که مکرراً تکرار شود، طراحی اصلی سیستم را مورد تهدید قرار خواهد داد.

[38] Backward Compatibility

۲.۹.۲ Version Identification

زمانی که قرارداد یک سرویس تغییر می‌کند، هر تغییری در محتویاتی از آن که منتشر می‌شود، نسخه جدیدی از قرارداد ایجاد می‌کند. بدون وجود ابزاری برای مرتبط‌ساختن نسخه‌های قرارداد با تغییرات، هماهنگی بین سرویس و مصرف‌کنندگان فعلی و آتی آن مورد تهدید قرار می‌گیرد. طبق این الگو، قرارداد سرویس به گونه‌ای طراحی می‌شود که شامل شناسه نسخه باشد تا مصرف‌کننده بتواند متوجه شود که با نسخه فعلی سرویس هماهنگ است یا خیر. استفاده از این شناسه، پیاده‌سازی قراردادهای موازی به صورت همزمان برای سرویس را نیز پشتیبانی می‌کند. در وب سرویس ها، این شناسه به صورت عددی در حاشیه‌های قرارداد سرویس قرار می‌گیرد. محدودیت عمده این الگو این است که لازم است طراحان سیستم‌های مصرف‌کننده، با دایره لغاتی که برای شماره‌گذاری نسخه‌ها استفاده می‌شود، از پیش آشنا باشند.

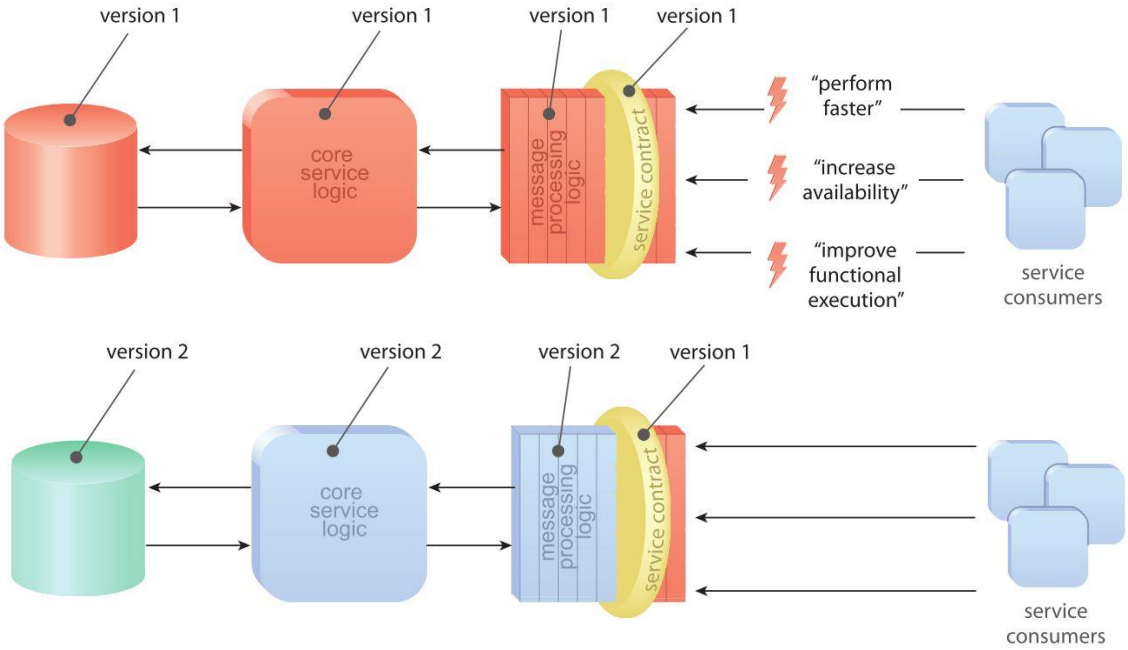
۳.۹.۲ Termination Notification

با تکامل سرویس در طول زمان، شرایط مختلف باعث نیاز به فسخ یک قرارداد سرویس، بخشی از آن یا کل سرویس شود. ممکن است این نیاز در اثر نیاز به تغییراتی باشد که سازگار با قبل نیستند، یا تغییر ریزدانگی سرویس‌های ارائه شده باشد یا به کلی به خاطر تغییر روند تجاری انجام کارها در سازمان، سرویس به کلی دیگر قابل استفاده نباشد. در این موارد، اطلاع این گونه تغییرات به مصرف‌کنندگان سرویس‌ها، به خصوص زمانی که سرویس در معرض عموم قرار گرفته باشد، بسیار زمان‌بر و هزینه‌بر خواهد بود. از طرفی، در صورت عدم اطلاع قبلی، پس از اعمال تغییرات مصرف‌کنندگان با خطاهای زمان اجرا مواجه خواهند شد. طبق این الگو، اطلاعات مربوط به پایان قرارداد، به قرارداد سرویس اضافه می‌شود تا مصرف‌کنندگان از قبل از فسخ قرارداد مطلع باشند.

۴.۹.۲ Service Refactoring

منطق یا تکنولوژی پیاده‌سازی سرویس‌ها ممکن است در طول زمان با مشخص شدن نیازمندی‌های مربوط به کارایی یا نیازمندی‌های تجاری که سرویس دیگر توانایی برقراری آنها را ندارد، نیازمند تغییر شوند. با این حال، جایگزینی کامل یک سرویس توسط دیگری، با توجه به

وابستگی مصرف‌کنندگان فعلی به سرویس قابل قبول نیست. این الگو، با انجام راهکارهای refactoring نرم‌افزار، پیاده‌سازی زیرین سرویس را بدون تغییر در رفتار بیرونی آن تغییر می‌دهد. با به کارگیری این الگو در سازمان، سرویس‌ها بدون درگیر کردن مصرف‌کنندگان خود، انعطاف‌پذیری بالایی در تغییر خواهند داشت. این الگو، قرارداد سرویس را که نقطه وابستگی مصرف‌کنندگان است، بدون تغییر باقی می‌گذارد و تغییرات در منطق سرویس از دید بیرون مخفی می‌شود.



شکل ۱۶ – استفاده از الگوی Service Refactoring و نحوه انتزاع مصرف‌کنندگان از تغییرات [۸۳]

فصل سوم

جمع‌بندی

۱.۳ جمع‌بندی

در این گزارش، مروری بر پژوهش‌های انجام‌شده در زمینه الگوهای توسعه و طراحی نرم‌افزار انجام شد. ابتدا به معرفی پژوهش‌های انجام‌شده پرداختیم و انواع دسته‌ها و حوزه‌های تحقیقاتی فعال در این زمینه معرفی شدند.

در فصل دوم نیز مروری بر الگوهای مربوط به معماری سرویس‌گرا، به عنوان یکی از محورهای پژوهشی الگوهای توسعه و طراحی نرم‌افزار، صورت گرفت. همچنین تعدادی از الگوهای معروف در قسمت‌های معماری سرویس‌گرا، پایه، امنیت، قرارداد و مدیریت سرویس مطرح و هر یک از انواع آن‌ها شرح داده شد.

منابع و مراجع

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of*. Addison-Wesley, 1994.

[2] P. Coad, "Object-oriented patterns," *Commun. ACM*, vol. 35 ,no. 9 ,pp. 152─159 ,1992.

[3] C. Larman, *Applying UML and Patterns: An Introduction to Oblect-oriented Analysis and Design*. Prentice Hall, 1998.

[4] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd edition. Prentice-Hall, 2004.

[5] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *A system of patterns: Pattern-oriented software architecture*. Wiley New York, 1996.

[6] M. Fowler and K. Beck, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[7] S. Demeyer, S. Ducasse, and O. Nierstrasz, *Object-oriented reengineering patterns*. Elsevier, 2002.

[8] S. W. Ambler, *Process patterns: building large-scale systems using object technology*. Cambridge University Press, 1998.

[9] M. Fowler, *Analysis patterns: reusable object models*. Addison-Wesley Professional, 1997.

[10] S. R. Humayoun, S. Hess, F. Kiefer, and A. Ebert, "Patterns for Designing Scalable Mobile App User Interfaces for Multiple Platforms," in *Proceedings of the 28th International BCS Human Computer Interaction Conference on HCI 2014-Sand, Sea and Sky-Holiday HCI*, 2014, pp 317─322.

[11] A. Neyem, S. F. Ochoa, and J. A. Pino, "A patterns system to coordinate mobile collaborative applications," *Gr. Decis. Negot.*, vol 20 ,no 5 ,pp 563─592 ,2011.

[12] A. Neyem, S. F. Ochoa, J. A. Pino, and R. D. Franco, "A reusable structural design for mobile collaborative applications," *J. Syst. Softw.*, vol 85 ,no 33 ,pp 511─524 ,2012.

[13] B.-K. White, "Visualizing mobile design pattern relationships," in *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services companion*, 2012, pp 71─76.

[14] Z. McCormick and D. C. Schmidt, "Data synchronization patterns in mobile application design," in *Proceedings of the 19th Conference on Pattern Languages of Programs*, 2012, p 12.

[15] M. M. Ali, A. F. Elsharkawi, M. G. El Said, and M. Zaki, "Design patterns for multimedia mobile applications," *J. Comput. Sci.*, vol 1 ,no 2 ,2014.

[16] N. Soundarajan, J. O. Hallstrom, A. Delibas, and G. Shu, "Testing Patterns," in *31st IEEE Software Engineering Workshop (SEW 2007)*, 2007, pp 109─120.

[17] N. Soundarajan, J. O. Hallstrom, G. Shu, and A. Delibas, "Patterns: from system design to software testing," *Innov. Syst. Softw. Eng.*, vol 4 ,no 1 ,pp 71─85 ,2008.

[18] L. Rising and others, "System test pattern language." 2000.

[19] B. Falah, M. Akour, and N. El Marchoum, "Testing Patterns in Action: Designing a Test-Pattern-Based Suite," *Comput. Softw.*, p 489 ,2015.

[20] P. Hamill, *Unit test frameworks: tools for high-quality software development*. " O'Reilly Media, Inc.," 2004.

[21] G. Meszaros, *xUnit test patterns: Refactoring test code*. Pearson Education, 2007.

[22] A. Syromiatnikov and D. Weyns, "A journey through the land of model-view-design patterns," in *Software Architecture (WICSA), 2014 IEEE/IFIP Conference on*, 2014, pp 21─30.

[23] T. Dennis, D. Hong, M. Marks, and K. Snow, "UC Berkeley Web Design Patterns Library," *Sch. Information, Univ. California, Berkeley*, 2006.

[24] F. Montero, M. Lozano, P. González, and I. Ramos, "Designing websites by using patterns," in *Second Latin American conference on Pattern Languages of Programming. SugarLoafPLoP02. Itaipava. Rio de Janeiro. Brasil. ISBN*, 2002, pp 85─87.

[25] M. Taleb, A. Seffah, and A. Abran, "Patterns-oriented design applied to cross-platform web-based interactive systems," in *Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on*, 2007, pp 122─127.

[26] D. Parsons, "Evolving architectural patterns for web applications," *PACIS 2007 Proc.*, p 56 ,2007.

[27] C. Fehling, F. Leymann, J. Rütschlin, and D. Schumm, "Pattern-based development and management of cloud applications," *Futur. Internet*, vol 4 ,no 1 ,pp 110─141 ,2012.

[28] S. Strauch, V. Andrikopoulos, U. Breitenbuecher, O. Kopp, and F. Leyrnann, "Non-functional data layer patterns for Cloud applications," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, 2012, pp 601─605.

[29] C. Fehling, F. Leymann, R. Retter, D. Schumm, and W. Schupeck, "An architectural pattern language of cloud-based applications," in *Proceedings of the 18th Conference on Pattern Languages of Programs*, 2011, p 2.

[30] C. Fehling, "Cloud computing patterns: identification, design, and application," 2015.

[31] C. Fehling, T. Ewald, F. Leymann, M. Pauly, J. Rütschlin, and D. Schumm, "Capturing cloud computing knowledge and experience in patterns," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012, pp 726─733.

[32] E. B. Fernandez, N. Yoshioka, and H. Washizaki, "Patterns for security and privacy in cloud ecosystems," in *Evolving Security and Privacy Requirements Engineering (ESPRE), 2015 IEEE 2nd Workshop on*, 2015, pp 13─18.

[33] C. Fehling, F. Leymann, R. Mietzner, and W. Schupeck, "A collection of patterns for cloud types, cloud service models, and cloud-based application architectures," *Univ. Stuttgart, Fac. Comput. Sci. Electr. Eng. Inf. Technol. Ger. Univ. Stuttgart, Inst. Archit. Appl. Syst. Tech. Rep. Comput. Sci.*, vol5 ,2011.

[34] L. Reinfurt, U. Breitenbücher, M. Falkenthal, F. Leymann, and A. Riegg, "Internet of Things Patterns," in *Proceedings of the 21st European Conference on Pattern Languages of Programs*, 2016, p 5:1--5:22.

[35] L. Reinfurt, U. Breitenbücher, M. Falkenthal, F. Leymann, and A. Riegg, "Internet of Things Patterns for Devices," in *Ninth international Conferences on Pervasive Patterns and Applications (PATTERNS) 2017*, 2017, pp 117─126.

[36] S. Qanbari *and et al.*, "IoT design patterns: computational constructs to design, build and engineer edge applications," in *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on*, 2016, pp 277─282.

[37] G. S. CHANDRA, "Pattern language for IoT applications," in *PATTERN LANGUAGES OF PROGRAMS CONFERENCE*, 2016.

[38] S. O. F. T. WORK, "Patterns for Model-Driven Software-Development," 2004.

[39] P. Ruben and S. Vjeran, "Framework for Using Patterns in Model-Driven Development," in *Information Systems Development*, Springer, 2009, pp 309─317.

[40] H. Ergin, E. Syriani, and J. Gray, "Design pattern oriented development of model transformations," *Comput. Lang. Syst. Struct.*, vol 46 ,pp 106─139 ,2016.

[41] K. Lano and S. Kolahdouz-Rahimi, "Model-transformation design patterns," *IEEE Trans. Softw. Eng.*, vol 40 , no 12 ,pp 1224─1259 ,2014.

[42] I. Tounsi, M. H. Kacem, A. H. Kacem, and K. Drira, "An Approach for SOA Design Patterns Composition," *Proc. - 2015 IEEE 8th Int. Conf. Serv. Comput. Appl. SOCA 2015*, pp 219─226 ,2016.

[43] A. K. Dwivedi and S. K. Rath, "Incorporating Security Features in Service-Oriented Architecture using Security Patterns," *ACM Softw. Eng. Notes*, vol 40 ,no 1 ,pp 1─6 ,2015.

[44] C. Pahl and R. Barrett, "Pattern-Based Software Architecture for Service-Oriented Software Systems," vol 4 , no 1 ,2010.

[45] M. Endrei *and et al*, "Patterns: Service- Oriented Architecture and Web Services," *Contract*, p 364 ,2006.

[46] C. Mauro, J. M. Leimeister, and H. Krcemar, "Service Oriented Device Integration – An Analysis of SOA

Design Patterns," *Hawaii Int. Conf. Syst. Sci.*, pp 1─10 ,2010.

[47] N. Mani, D. C. Petriu, and M. Woodside, "Studying the Impact of Design Patterns on the Performance Analysis of Service Oriented Architecture," *2011 37th EUROMICRO Conf. Softw. Eng. Adv. Appl.*, pp 12─19 ,2011.

[48] C. Moser, S. Junginger, M. Brückmann, and K.-M. Schöne, "Some Process Patterns for Enterprise Architecture Management.," in *Software Engineering (Workshops)*, 2009, pp 19─30.

[49] M. Taleb and O. Cherkaoui, "Pattern-oriented approach for enterprise architecture: TOGAF framework," *Des. Enterp. Archit. Fram.*, p 99 ,2012.

[50] S. Buckl, A. M. Ernst, F. Matthes, R. Ramacher, and C. M. Schweda, "Using enterprise architecture management patterns to complement TOGAF," in *Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International*, 2009, pp 34─41.

[51] S. Buckl, A. M. Ernst, J. Lankes, F. Matthes, and C. M. Schweda, "Enterprise architecture management patterns--exemplifying the approach," in *Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE*, 2008, pp 393─402.

[52] S. Buckl, A. M. Ernst, F. Matthes, and C. M. Schweda, "Enterprise Architecture Management Patterns for Enterprise Architecture Visioning.," in *EuroPLoP*, 2009.

[53] M. Fowler, *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[54] P. Aleatrati Khosroshahi, M. Hauder, and F. Matthes, "Analyzing the Evolution and Usage of Enterprise Architecture Management Patterns," 2016.

[55] A. Aarsten, G. Elia, and G. Menga, "G++: a pattern language for the object oriented design of concurrent and distributed information systems, with applications to computer integrated manufacturing," in *Design of Concurrent and Distributed Information Systems, with Applications to Computer Integrated Manufacturing. In Pattern Languages of Program Design. Coplien*, 1995.

[56] A. R. Silva, F. Hayes, F. Mota, N. Torres, and P. Santos, "A Pattern Language for the Perception, Design and Implementation of Distributed Application Partitioning," in *Workshop on Methodologies for Distributed Objects*, 1996.

[57] N. Delessy, E. B. Fernandez, M. M. Larrondo-Petrie, and J. Wu, "Patterns for access control in distributed systems," in *Proceedings of the 14th Conference on Pattern Languages of Programs*, 2007, p 3.

[58] K. Brown, P. Eskelin, and N. Pryce, "A mini-pattern language for distributed component design," in *Pattern Languages of Programs (PLoP) Conference*, 1999.

[59] A. H. Karp and K. Smathers, "Three design patterns for secure distributed systems," *Inf. Secur. Bull.*, pp 49─54 ,2003.

[60] R. Barrett and C. Pahl, "Distribution pattern-driven development of service architectures," *J. Univers. Comput. Sci.*, vol 15 ,no 11 ,pp 2166─2195 ,2009.

[61] F. Buschmann, K. Henney, and D. C. Schmidt, "Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing (v. 4)," 2007.

[62] F. J. da S. e Silva, F. Kon, J. Yoder, and R. Johnson, "A pattern language for adaptive distributed systems," in *Proceedings of the 5th Latin American Conference on Pattern Languages of Programming*, 2005, pp 19─48.

[63] Jeremiah Y. Dangler, "Categorization of Security Design Patterns," East Tennessee State University, 2013.

[64] P. Anand, J. Ryoo, and R. Kazman, "Vulnerability-based security pattern categorization in search of missing patterns," in *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on*, 2014, pp 476─483.

[65] P. Ponde, S. Shirwaikar, and C. Kreiner, "An analytical study of security patterns," in *Proceedings of the 21st European Conference on Pattern Languages of Programs*, 2016, p 33.

[66] R. Dove and L. Shirey, "On discovery and display of agile security patterns," in *Conf Syst Eng Res, Stevens Institute of Technology, Hoboken, NJ*, 2010.

[67] R. Scandariato, K. Yskout, T. Heyman, and W. Joosen, "Architecting software with security patterns," 2008.

[68] M. Bunke, R. Koschke, and K. Sohr, "Application-domain classification for security patterns," in *Proceedings*

of the International Conferences on Pervasive Patterns and Applications, IARIA Conferences. XPS (Xpert Publishing Services), 2011, pp 138─143.

[69] M. Hafiz and P. Adamczyk, "The nature of order: from security patterns to a pattern language," in *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, 2012, pp 75─76.

[70] E. B. Fernandez, "Security patterns and secure systems design," in *ACM Southeast Regional Conference*, 2007, p 510.

[71] A. Kumar and E. B. Fernandez, "Security patterns for intrusion detection systems," in*1st LACCEI International Symposium on Software Architecture and Patterns (LACCEI-ISAP-MiniPLoP'2012), Panama City, Panama*, 2012.

[72] A. Wiesauer and J. Sametinger, "A Security Design Pattern Taxonomy based on Attack Patterns," in *International Joint Conference on e-Business and Telecommunications*, 2009, pp 387─394.

[73] N. Yoshioka, H. Washizaki, and K. Maruyama, "A survey on security patterns," *Prog. informatics*, vol 5 ,no 5 , pp 35─47 ,2008.

[74] E. B. Fernandez, N. Yoshioka, H. Washizaki, J. Jürjens, M. VanHilst, and G. Pernul, "Using security patterns to develop secure systems," *IGI Glob.*, pp 16─31 ,2011.

[75] A. Ampatzoglou, S. Charalampidou, and I. Stamelos, "Research state of the art on GoF design patterns: A mapping study," *J. Syst. Softw.*, vol 89 ,no 7 ,pp 1945─1964 ,2013.

[76] M. Riaz, T. Breaux, and L. Williams, "How have we evaluated software pattern application? A systematic mapping study of research design practices," *Inf. Softw. Technol.*, vol 65 ,pp 14─38 ,2015.

[77] C. Zhang and D. Budgen, "A survey of experienced user perceptions about software design patterns," *Inf. Softw. Technol.*, vol 55 ,no 5 ,pp 822─835 ,2013.

[78] C. Zhang and D. Budgen, "What do we know about the effectiveness of software design patterns?," *IEEE Trans. Softw. Eng.*, vol 38 ,no 5 ,pp 1213─1231 ,2012.

[79] B. B. Mayvan, A. Rasoolzadegan, and Z. G. Yazdi, "The state of the art on design patterns: A systematic mapping of the literature," *J. Syst. Softw.*, vol 125 ,pp 93─118 ,2017.

[80] M. H. Valipour, B. Amirzafari, K. N. Maleki, and N. Daneshpour, "A brief survey of software architecture concepts and service oriented architecture," *Comput. Sci. Inf. Technol. 2009. ICCSIT 2009. 2nd IEEE Int. Conf.*, pp 34─38 ,2009.

[81] U. Zdun, C. Hentrich, and W. M. P. Van Der Aalst, "A survey of patterns for service-oriented architectures," *Int. J. Internet Protoc. Technol.*, vol 1 ,no 3 ,pp 132─143 ,2006.

[82] T. Erl, "Introducing SOA Design Patterns," *SOA World Mag.*, pp 2─7 ,2008.

[83] T. Erl, *SOA Design Patterns*, vol 47 ,no February. 2009.

[84] "SOA Patterns." .

[85] A. Rotem-Gal-Oz, E. Bruno, and U. Dahan, *SOA Patterns*. Manning, 2012.

[86] M. K. Khan, *Performance Measurement of Design Patterns for Service-Oriented Architecture*, no January. 2013.

[87] E. B. Fernandez, O. Ajaj, I. Buckley, N. Delessy-Gassant, K. Hashizume, and M. M. Larrondo-Petrie, "A Survey of Patterns for Web Services Security and Reliability Standards," *Futur. Internet*, vol 4 ,pp 430─450 , 2012.

[88] J. Steve, "SOA anti-patterns." .

[89] T. Modi, "SOA Antipatterns." .

[90] N. Moha *and et al*, "Specification and Detection of SOA Antipatterns," pp 1─16 ,2012.

[91] D. Androcec, D. Kermek, T. Hunjak, S. Lovrenoc, and I. Tomicic, "Useful Patterns for BPEL Developers," *23rd Cent. Eur. Conf. Inf. Intell. Syst.*, pp 457─461 ,2012.

[92] C. Hentrich and U. Zdun, “A pattern language for process execution and integration design in service-oriented architectures,” pp 136─191 ,2009.

[93] L. Krause, “Microservices: patterns and applications,” vol 1 ,2015.

[94] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices Migration Patterns,” no 1 ,pp 1─21 ,2015.

[95] M. Voelter, *Object Oriented Remoting: Foundations of Realtime Internet and Enterprise Distribution Infrastructures*. John Wiley \& Sons, 2004.

[96] M. Kircher and P. Jain, *Pattern-Oriented Software Architecture, Patterns for Resource Management*. John Wiley \& Sons, 2013.