

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

توصیف تابعی کارکردهای سامانه انتخاب واحد دانشگاه

نگارش

فاطمه علاقه‌بند

استاد راهنما

دکتر رامتین خسروی

پروژه نهایی مقطع کارشناسی رشته مهندسی کامپیوتر

تابستان ۱۴۰۰

تعهدنامه اصالت اثر

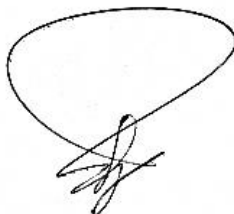
بسمه تعالی

اینجانب فاطمه علاقه‌بند تایید می‌کنم که مطالب مندرج در این پایان‌نامه حاصل تلاش اینجانب است و به دستاوردهای پژوهشی دیگران که در این نوشته از آن‌ها استفاده شده است مطابق مقررات ارجاع گردیده است. این پایان‌نامه قبلاً برای احراز هیچ مدرک هم سطح یا بالاتر ارائه نشده است. کلیه حقوق مادی و معنوی این اثر متعلق به دانشکده فنی دانشگاه تهران می‌باشد.

نام و نام خانوادگی دانشجو:

فاطمه علاقه‌بند

امضای دانشجو:



چکیده

آزمون نرم‌افزار امروزه بخش مهمی از فرآیند تولید نرم‌افزار را تشکیل می‌دهد. تا کنون روش‌های مختلفی برای تولید آزمون خودکار برای نرم‌افزار پیشنهاد شده است. یکی از حوزه‌هایی که به این مهم توجه دارد، حوزه آزمون بر مبنای توصیف است. در این روش‌ها توصیفی از سیستم به عنوان مدلی برای استخراج آزمون استفاده می‌شود. متأسفانه گرچه تحقیقات گسترده‌ای در این زمینه انجام شده است، استقبال کمی از این حوزه در صنعت نرم‌افزار صورت گرفته است. دلیل این امر عدم امکان استفاده از روش‌های پیشنهادی در نرم‌افزارهای در مقیاس صنعتی است. برای مثال در سرویس‌های نرم‌افزاری با توانمندی بالا معمولاً نیازمندی‌های کارکردی و غیرکارکردی زیادی وجود دارد که باعث می‌شود سیستم تولید شده دارای معماری بسیار پیچیده‌ای باشد. این امر باعث می‌شود خود سیستم به طور مستقیم قابلیت استفاده برای تولید آزمون خودکار را نداشته باشد. از سوی دیگر در اکثر روش‌های تولید آزمون مبتنی بر توصیف، نیاز است توصیف سیستم در قالب توصیف‌های رسمی صورت گیرد. استفاده از توصیف‌های رسمی منجر می‌شود که اجرای این روش‌ها توسط تیم‌های توسعه عملاً غیرممکن باشد زیرا برای تولید توصیف‌های رسمی نیاز به مهارت و تخصص بالایی است که عمده جامعه توسعه‌دهندگان از آن برخوردار نیستند.

در این تحقیق، به روش دیگری برای توصیف سامانه‌ها پرداخته شده که مشکلات ذکر شده را نداشته باشد و منطق دامنه یک سامانه جامع اطلاعاتی با زبان تابعی هسکل به طور کامل توصیف شده است. با توجه به ویژگی‌های مورد نیاز سامانه هدف و همچنین آشنایی با دامنه سامانه‌های دانشگاهی، سامانه انتخاب واحد بدین منظور انتخاب شده است. ارزیابی نهایی و مقایسه توصیف با پیاده‌سازی دیگری از همین سامانه به زبان جاوا و همچنین اصول و الگوهای به دست آمده برای توصیف یک سامانه در ابعاد بزرگ، نشانگر این است که زبان هسکل و به طور کل روش‌های تابعی برای مدل‌سازی و توصیف منطق چنین سامانه‌هایی، مناسب و مفید هستند. توصیف به دست آمده، خوانا، ساختارمند، با قابلیت گسترش‌پذیری و نگهداری بالا و دارای حجمی معقول و مناسب و بسیار کمتر از حجم پیاده‌سازی کامل یک سامانه است. این عوامل در کنار قابلیت اجرای توصیف و آزمون‌پذیری، آن را برای استفاده به عنوان مدل کاملی از سامانه در روش‌های تولید آزمون خودکار و همچنین پیشگوی آزمون مناسب می‌سازند.

کلمات کلیدی:

توصیف تابعی – طراحی دامنه محور – زبان برنامه نویسی هسکل – معماری مبتنی بر رویداد

فهرست مطالب

۱. مقدمه	۶
۱-۱ اهمیت و انگیزه تحقیق	۶
۱-۲ تعریف مسئله	۷
۱-۳ روش انجام پژوهش	۷
۱-۴ ارزیابی و دستاوردهای تحقیق	۷
۱-۵ خلاصه و ساختار فصل‌ها	۸
۲. پیش‌زمینه	۹
۲-۱ طراحی دامنه محور	۹
۲-۲ معماری مبتنی بر رویداد	۱۰
۳. روش پیشنهادی	۱۲
۳-۱ مقدمه	۱۲
۳-۲ توصیف مدل دامنه	۱۲
۳-۳ درخواست‌های سامانه	۱۸
۳-۴ معماری مبتنی بر رویداد	۲۱
۳-۵ اصول و الگوها	۲۳
۴. ارزیابی	۲۵
۵. جمع‌بندی و نتیجه‌گیری	۲۷
۶. مراجع	۲۹

۱. مقدمه

۱-۱ اهمیت و انگیزه تحقیق

امروزه سرویس‌های نرم افزاری بسیاری وجود دارند که به خاطر نیازمندی‌های کارکردی و غیرکارکردی، پیچیدگی‌های ساختاری بسیاری دارند. برای نمونه می‌توان از سرویس‌های تهیه تاکسی‌های آنلاین، هسته معاملات بورس، فروشگاه‌های آنلاین و سامانه آموزش دانشگاه نام برد. در این سیستم‌ها مجموعه بزرگی از نیازمندی‌های کارکردی برای سیستم تعریف شده است که بسیاری از آن‌ها منطق‌های پیچیده‌ای را پیاده‌سازی می‌کنند.

با توجه به اهمیت این گونه از سیستم‌ها، صحت عملکرد سیستم اهمیت بسیار بالایی دارد. گرچه در این سیستم‌ها معمولاً مجموعه قابل قبولی از آزمون‌های واحد برای هر قسمت از سیستم تولید می‌شود، نمی‌توان از درستی کارکرد سیستم به صورت یک مجموعه کامل اطمینان حاصل کرد. علی‌رغم اینکه در هر بخش به صورت مجزا اطمینان نسبی نسبت به صحت عملکرد وجود دارد ولی مشکلات بسیاری در تعامل این بخش‌ها با یکدیگر و همچنین مفروضات این بخش‌ها نسبت به یکدیگر می‌تواند وجود داشته باشد که مورد بررسی قرار نگرفته است. در نتیجه نیاز به مجموعه آزمون دیگری است که سیستم را به صورت یک مجموعه یک‌جا مورد بررسی قرار دهد. این مجموعه آزمون، آزمون‌های سرتاسری نام دارند.

از آنجا که در تولید آزمون‌های سرتاسری، هدف پوشش تمامی نیازمندی‌های کارکردی است، نیاز است روشی برای تولید و اطمینان از پوشش کامل مجموعه آزمون ایجاد شود. این نکته باعث ایجاد روش‌های آزمون مبتنی بر مدل شده است. در این روش‌ها، بر اساس نیازمندی‌های کارکردی سیستم، مدلی ساخته می‌شود که با پوشش کامل می‌توان اطمینان حاصل کرد که سیستم مورد آزمون از نظر تمامی نیازمندی‌های کارکردی مورد بررسی قرار گرفته و عملکرد مناسبی دارد. فایده دیگری که استفاده از روش‌های آزمون مبتنی بر مدل به همراه دارد، خودکارسازی فرآیند تولید آزمون است.

روش‌های مدل‌سازی بسیاری برای تولید مدلی که بتواند برای تولید خودکار آزمون مورد استفاده قرار گیرد، وجود دارد. در برخی از این روش‌ها سعی می‌شود تا سیستم با یک شیوه رسمی توصیف شود اما توسعه توصیف در این روش‌ها نیاز به تسلط بالا به روش‌های توصیف رسمی سیستم‌ها دارد که باعث شده است این روش‌های تولید آزمون با اقبال خوبی در صنعت مواجه نشوند. در این تحقیق، از مدل‌سازی و توصیف با استفاده از زبان‌های تابعی^۱ استفاده شده است. ماهیت این زبان‌ها به توصیفات رسمی نزدیک است و این نکته باعث می‌شود درستی‌یابی سیستم با این توصیف‌ها راحت‌تر باشد و همچنین به اندازه کافی راحت است تا نیاز به متخصصین برای توصیف سیستم نباشد و با مهارت و تسلط میانگین نیز بتوان توصیف را انجام داد.

در میان زبان‌های برنامه‌نویسی تابعی، زبان هسکل^۲ به عنوان زبان مورد استفاده برای توصیف سیستم انتخاب شده است. از دلایل این انتخاب می‌توان به سادگی توصیف سیستم با استفاده از این زبان و همچنین وجود منابع بیشتر در مورد این زبان، اشاره کرد.

¹ Unit Tests

² End to End Tests

³ Functional Programming Languages

⁴ Haskell

برای مثال در مرجع [۱] اشاره شده است که برای توسعه نسخه اولیه‌ای از یک سیستم که پیچیدگی‌های بسیاری نیز داشته است، چندین زبان مورد بررسی قرار گرفته است و از میان آن‌ها، برنامه توسعه داده شده در زبان هسکل با اختلاف قابل توجهی ساده‌تر و کوچک‌تر از سایر زبان‌های مورد استفاده بوده است.

۲-۱ تعریف مسئله

با توجه به اینکه زبان‌های تابعی مانند هسکل عموماً برای توسعه توصیف‌های سریع اما کامل از یک دامنه مورد استفاده قرار می‌گیرند، تجربه و نمونه‌های مشابهی که از هسکل برای توصیف یک سامانه جامع سازمانی - اطلاعاتی بهره گرفته باشند به ندرت یافت می‌شود. صورت مسئله اصلی که در این پژوهش مورد بررسی قرار می‌گیرد این است که آیا به طور کل، روش‌های مدل‌سازی با استفاده از زبان‌های تابعی و به طور خاص، زبان هسکل، انتخاب مناسبی برای توصیف سامانه‌های جامع و گسترده اطلاعاتی هستند یا خیر و برای رسیدن به توصیف کاملی از منطق چنین سامانه‌ای، چه الگوها و معماری‌هایی می‌توانند کمک‌کننده و مفید باشند. سیستمی که برای توصیف انتخاب می‌شود باید دارای ویژگی‌های زیر باشد:

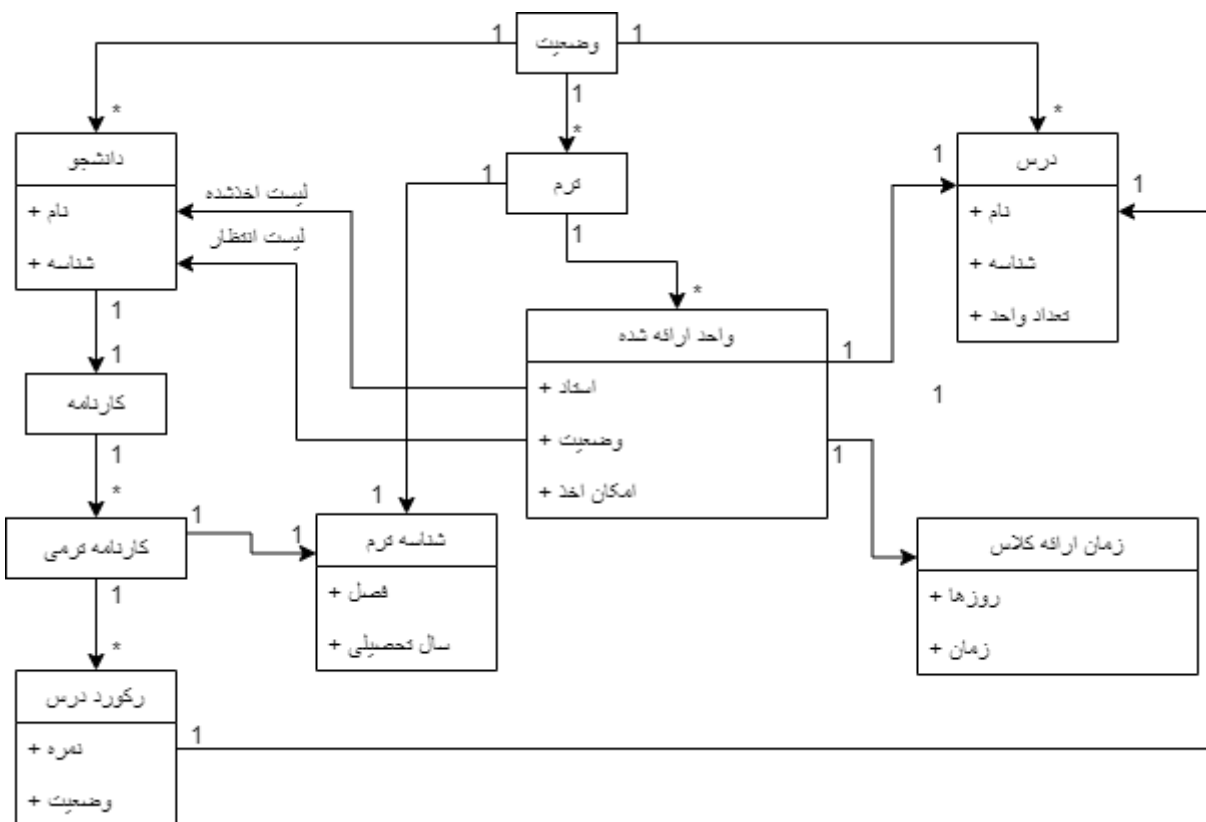
- سیستم از مجموعه‌ای از درخواست‌ها و پاسخ‌های مستقل از هم تشکیل شده باشد. پاسخ درخواست‌ها در اینگونه سیستم‌ها فقط به وضعیت سیستم در هنگام دریافت درخواست و پارامترهای درخواست وابسته است. همچنین سیستم مورد بررسی باید از چندین نوع درخواست تشکیل شده باشد.
- سیستم دارای منطق محاسباتی پیچیده‌ای باشد. یکی از مهم‌ترین ویژگی‌هایی که روش‌های موجود تولید آزمون خودکار را تحت‌تاثیر قرار می‌دهد، پیچیدگی منطق سامانه مورد بررسی است زیرا هرچه منطق برنامه پیچیده‌تر می‌شود، فضای حالت برنامه بزرگ‌تر شده و در نتیجه، بودجه و زمان بیشتری برای تولید آزمون مورد نیاز خواهد بود. در مقابل این سیستم‌ها، سیستم‌هایی قرار می‌گیرند که هدف آن‌ها برآورده‌سازی نیازهای داده‌ای کاربر است مثلاً سیستمی که امکان مشاهده و جست‌وجوی فایل‌ها را برای کاربر فراهم می‌کند که در دسته سیستم‌های هدف این تحقیق قرار نمی‌گیرد.
- معماری سیستم نسبتاً پیچیده باشد. گرچه روش پیشنهادی می‌تواند برای سیستم‌هایی که معماری پیچیده‌ای ندارند به درستی عمل کند، پیچیدگی معماری از دلایل توسعه روش جدیدی برای تولید آزمون است. در نتیجه هرچه معماری سیستم پیچیده‌تر باشد، بهتر می‌توان ارزش روش پیشنهادی و قابلیت استفاده از آن را درک کرد.

۳-۱ روش انجام پژوهش

روش ما در این پژوهش برای دستیابی به هدف طرح شده، مطالعه موردی روی سامانه‌ای دارای ویژگی‌های فوق بوده است. از بین انتخاب‌های متعدد برای سامانه هدف، سامانه آموزش دانشگاه به دلیل آشنایی و تسلط بیشتر بر دامنه و منطق آن برگزیده شده است. تصویر ۱ نشان‌دهنده نمودار دامنه سامانه است که در این پژوهش توسعه یافته و به تدریج گسترش داده شده است.

۴-۱ ارزیابی و دستاوردهای تحقیق

برای بررسی این موضوع که آیا توصیف تابعی چنین سیستمی در عمل قابل نگهداری، گسترش و استفاده برای تولید خودکار آزمون هست یا خیر، توصیف نهایی با زبان هسکل با پیاده‌سازی منطق همین سیستم به زبان جاوا مقایسه شده است. چند سناریوی تغییر که از قبل در مدل‌سازی توصیف پیش‌بینی نشده بودند، طراحی و در پیاده‌سازی سیستم با هر دو زبان اجرا شد و اثرات آن روی



تصویر ۱: نمودار دامنه سامانه دانشگاه

توصیف‌ها با معیارهای مختلفی مانند میزان تغییر کد و زمان لازم برای ایجاد این تغییر در سیستم مقایسه گردید. نتایج این مقایسه به تفصیل در بخش ۴ توضیح داده شده است اما نکته مهم در این قسمت این است که نتایج نهایی به ما نشان می‌دهد توصیف سامانه جامع اطلاعاتی با زبان‌های تابعی، ممکن و مفید است و عملاً فرق زیادی با زبان‌های رویه‌ای^۱ ندارد. همچنین باید توجه داشت که یک سامانه سازمانی جدای از منطق دامنه، حجم بسیار زیادی کد مربوط به تکنولوژی‌های مختلف اعم از دسترسی به داده‌ها، رابط کاربری و فراهم کردن نیازهای غیرکارکردی دارد و توصیف دامنه چنین سیستمی با یک زبان تابعی، حجم با اختلاف کمتری خواهد داشت که می‌تواند در تولید خودکار آزمون‌ها و خصوصاً آزمون‌های سرتاسری، بسیار کمک‌کننده باشد.

۵-۱ خلاصه و ساختار فصل‌ها

در فصل آتی، به شرح دو مفهوم کلیدی و پیش‌زمینه این پژوهش خواهیم پرداخت و اصول و مبنای کلی آن‌ها را توضیح خواهیم داد. سپس در فصل سوم به توضیح روش انجام پژوهش، مراحل پیشرفت و گسترش اجزای مختلف توصیف، تصمیم‌ها و انتخاب‌های طول مسیر و اصول و الگوهای به دست آمده بر اساس تجربه می‌پردازیم. فصل چهارم به بررسی روش ارزیابی توصیف و نتایج این ارزیابی اختصاص دارد. در فصل آخر نیز یک خلاصه کلی از هدف پروژه و تعریف مسئله ارائه خواهد شد و در مورد اصلی‌ترین نتایج به دست آمده و همچنین موضوعات پیشنهادی برای تحقیقات آتی سخن خواهیم گفت.

¹ Procedural

۲. پیش زمینه

۱-۲ طراحی دامنه محور^۱

طراحی دامنه محور یا DDD، رویکردی برای تولید و توسعه نرم‌افزارهای بزرگ با فرآیندها و قوانین متعدد، پیچیده و در حال تغییر می‌باشد. اصطلاح دامنه به حوزه و دامنه اصلی نرم افزار اطلاق می‌شود که نرم‌افزار برای پیاده‌سازی آن توسعه می‌یابد. هسته اصلی طراحی دامنه محور، مجموعه‌ای از مفاهیم و تکنیک‌ها است که برای تحلیل دامنه و ساخت یک مدل از روی آن (مدل دامنه) به کار برده می‌شود. توجه و تمرکز اصلی این رویکرد بر توسعه این مدل می‌باشد. تحلیل و طراحی مدل دامنه به طور خاص به منظور تولید نرم‌افزار در ابعاد Enterprise و با فرآیندهای زیاد و پیچیده مناسب است که پس از طراحی، باید با جزئیات و دقیق باشد و مفاهیم و قوانین کسب‌وکار به خوبی در آن پیاده‌سازی شده باشد.

طراحی دامنه محور بر زوایای دیگری از فرآیند توسعه نرم افزار نیز تاثیرگذار است. برای مثال تاکید زیادی بر ارتباط دوطرفه تیم توسعه نرم‌افزار و متخصصان دامنه دارد. از آنجا که ممکن است در این ارتباط دوطرفه، تیم توسعه نرم‌افزار در فهمیدن برخی مفاهیم و مسائل دچار اشتباه و دوگانگی شوند، لذا ایجاد یک زبان یکسان بین دو تیم در مورد مفاهیم دامنه امری الزامی است. این رویکرد همچنین راهکارهایی برای تقسیم نرم‌افزار به بخش‌هایی جدا و مستقل و همچنین ارتباط این بخش‌ها با یکدیگر ارائه می‌کند. این امر سبب می‌شود تا فرآیند توسعه نرم‌افزار به صورت موازی بین چند تیم انجام شود و همچنین معماران سیستم را قادر می‌سازد تا از معماری‌ها و تکنولوژی‌های مختلف در بخش‌های مختلف استفاده نمایند.

همانطور که مطرح شد، رویکرد دامنه محور برای نرم‌افزارهای بزرگ و پیچیده مناسب است لذا استفاده از آن در پروژه‌های کوچک و ساده یا پروژه‌هایی که صرفاً نیاز به ذخیره و خواندن اطلاعات دارند و بخش کسب‌وکار آن‌ها جدی و پیچیده نیست، ممکن است تنها هزینه و زمان پروژه را افزایش داده و مزیت خاصی به همراه نداشته باشد.

۱-۱-۲ مفهوم مدل دامنه

در اغلب پروژه‌های نرم‌افزاری، دغدغه اصلی تیم توسعه، طراحی دقیق و درست فرآیندها و قوانین نرم‌افزار می‌باشد. زمان و هزینه‌ای که صرف تحلیل، پیاده‌سازی و تست فرآیندها و منطق نرم‌افزار می‌شود بسیار بیشتر از زمان و هزینه‌ای است که صرف موارد فنی و زیرساخت نرم‌افزاری می‌شود. تیم توسعه نرم‌افزار می‌تواند با تمرکز بر روی دامنه و طراحی مدل دامنه از روی آن و همچنین با رعایت مفاهیم و تکنیک‌های طراحی دامنه محور از پیاده‌سازی درست این فرآیندها اطمینان پیدا کند.

¹ Domain Driven Design

² Domain

³ Domain Model

⁴ Business Rules

⁵ Ubiquitous Language

⁶ Bounded Context

مدل دامنه، مدلی انتزاعی از دامنه فعالیت نرم افزار است که توسط تیم توسعه نرم افزار در قالب یک مدل شیء گرا طراحی می شود. این مدل ممکن است در طول زمان و با تغییر فرآیندها تغییر یافته و تکمیل شود. مدل دامنه تنها یک ساختار داده ای نیست بلکه مجموعه ای از داده ها، رفتارها و قوانین کسب و کار است و از آن جا که مهم ترین چیز در طراحی مدل دامنه، اطمینان از صحت فرآیندها و منطق نرم افزار می باشد، هنگام طراحی از پرداختن به دغدغه ها و نکات فنی زیرساختی و مربوط به پایگاه داده یا رابط کاربری خودداری می شود.

اکثر مباحث مطرح شده در طراحی دامنه محور در راستای تحلیل و توسعه مدل دامنه می باشد که در قالب مفاهیمی مانند Entity, Value Object, Aggregate و... پیاده سازی می شود.

در این پژوهش نیز با توجه به اینکه تمرکز اصلی بر توصیف منطق و کارکردهای سامانه منتخب است، طراحی دامنه محور به عنوان رویکرد اصلی پیاده سازی در نظر گرفته شده است که نمودار شکل ۱، موجودیت های اصلی مدل دامنه و رابطه بین آن ها را نشان می دهد.

۲-۲ معماری مبتنی بر رویداد^۱

معماری مبتنی بر رویداد یکی از معماری های رایج نرم افزار است که بخش اصلی آن را دریافت، پردازش و ارتباط رویدادها تشکیل می دهد که این اصلی ترین تفاوت آن با معماری مبتنی بر درخواست-پاسخ^۲ است. بسیاری از اپلیکیشن ها و سایت ها امروزه از این معماری استفاده می کنند. این معماری با هر زبان برنامه نویسی قابل پیاده سازی است (به عبارت دیگر، رویکرد وابسته به نوع برنامه نویسی است نه زبان آن) و کمترین میزان وابستگی^۳ بین موجودیت های مختلف را به وجود می آورد که باعث می شود این معماری انتخاب مناسبی برای سیستم های توزیع شده باشد. معماری مبتنی بر رویداد در حالت کلی از تولیدکننده های رویداد^۴ و مصرف کننده ها^۵ تشکیل می شود و علت وابستگی پایین بین موجودیت ها این است که تولیدکننده های هر رویداد، از اینکه کدام مصرف کننده ها از رویداد استفاده می کنند اطلاعی ندارند و خود رویداد نیز از نتایجی که در هر مصرف کننده ایجاد می کند، آگاه نیست.

در تعریف رویداد می توان گفت شامل هر اتفاق مشخص یا تغییری در وضعیت سیستم می شود. منشا ایجاد یک رویداد می تواند داخلی یا خارجی باشد مثلاً کاربری که درخواستی به سیستم می دهد یا فشرده شدن یک کلید یا ورودی یک سنسور همه می توانند منشا یک رویداد باشند.

از فواید اصلی معماری مبتنی بر رویداد می توان به دستیابی به سیستم انعطاف پذیرتری اشاره کرد که می تواند بسته به تغییرات مورد نیاز، راحت تر سازگاری یابد زیرا احتمالاً این تغییرات به اضافه کردن چند رویداد یا اطلاع رسانی این رویدادها به مصرف کننده های جدید منجر خواهد شد که کلیت آن پیش از این در سیستم وجود داشته است. این معماری با دو مدل کلی پیاده سازی می شود:

- مدل جریان و انتشار عمومی رویدادها؛^۶ در این مدل، رویدادها در کانال های جداگانه ای منتشر نمی شوند بلکه همه کنار هم در یک فایل (جریان) ثبت می شوند و مصرف کننده ها می توانند از هر جای این فایل رویدادها را بخوانند یا هر زمانی که خواستند این جریان را ترک کنند.

¹ Even-Driven

² Request-Response Architecture

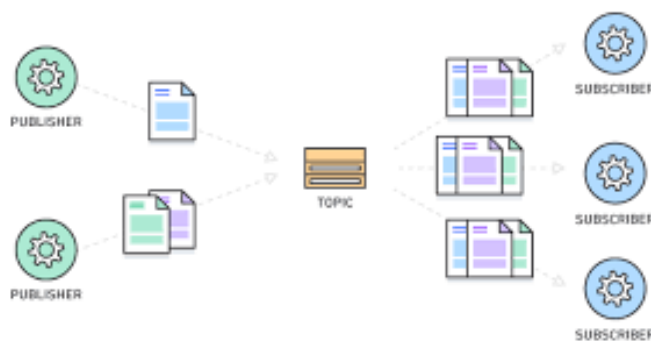
³ Coupling

⁴ Event Producers

⁵ Event Consumers

⁶ Event Streaming Model

- مدل انتشار/عضویت^۱ در این مدل، موجودیت‌ها بسته به نیاز خود می‌توانند در کانال انتشار هر رویداد عضو شوند و به محض رخ دادن رویداد و منتشر شدن آن در این کانال، از آن آگاه شده و متناسب با آن اقدام کنند. به عبارت دیگر، رویدادها در این مدل به صورت عمومی منتشر نمی‌شوند اما تولیدکننده‌های رویداد از اینکه چه کسانی از رخ دادن این رویداد مطلع خواهند شد، آگاه نیستند. تصویر ۲ شمای کلی این مدل را نمایش می‌دهد.



تصویر ۲: مدل انتشار/عضویت در معماری مبتنی بر رویداد

پیاده‌سازی توصیف سیستم آموزش دانشگاه در این پژوهش در دو مرحله انجام شده است. در مرحله اول، توصیف با ارتباط ساده و خالص موجودیت‌ها با هم طراحی و پیاده‌سازی شد و در مرحله دوم، معماری مبتنی بر رویداد به سیستم اضافه گردید و درخواست‌های مختلف سیستم به صورت رویدادهایی تعریف شدند که بین موجودیت‌های مختلف، تبادل شده و اقدامات متناسب با آن انجام می‌گیرد. توضیح بیشتر هر دو مرحله در فصل ۳ به تفصیل ارائه خواهد شد.

¹ Pub/Sub Model

۳. روش پیشنهادی

۳-۱ مقدمه

پیاده‌سازی توصیف سامانه آموزش دانشگاه در این پژوهش، در دو مرحله کلی انجام گرفته است. در مرحله اول، پس از طراحی نمودار UML از موجودیت‌های اصلی و نحوه ارتباط آن‌ها، کل سامانه به سه ماژول اصلی شکسته شد که هریک شامل موجودیت‌های مرتبط‌تر به هم و توابع کمکی مورد نیاز برای هر کدام از آن‌ها می‌شد. در این مرحله به طور کلی برای ذخیره و نمایش مفاهیم اصلی و موجودیت‌ها (مانند درس، دانشجو، ترم، کارنامه و غیره)، نوع داده‌ای آن‌ها با استفاده از امکان زبان هسکل برای تعریف رکوردها پیاده‌سازی شده است. همچنین برای ذخیره‌سازی وضعیت سیستم، مفهوم جدید وضعیت علاوه بر مفاهیم آشنای این سامانه تعریف و پیاده‌سازی شده است. درخواست‌های سامانه با پارامترهای خاص خود، از طریق این مفهوم وضعیت در سیستم اعمال می‌شوند.

در مرحله بعدی، معماری مبتنی بر رویداد با کمک مرجع [۲] به سیستم اضافه گردید و درخواست‌های مختلف سیستم به صورت رویدادهایی تعریف شدند که بین موجودیت‌های مختلف، تبادل شده و اقدامات متناسب با آن انجام می‌گیرد (شروع درخواست همچنان از مفهوم وضعیت اتفاق می‌افتد).

در این فصل، پس از توضیح چگونگی مدل‌سازی و توصیف مدل دامنه، به نحوه پیاده‌سازی درخواست‌های سامانه در مرحله اول و فرآیند اضافه کردن معماری مبتنی بر رویداد به سیستم و تأثیرات آن بر توصیف خواهیم پرداخت. در نهایت در مورد اصول کلی و الگوهای اصلی که در توصیف یک سامانه جامع اطلاعاتی با یک زبان تابعی به دست آمده است، صحبت خواهیم کرد.

۳-۲ توصیف مدل دامنه

همانطور که در بخش مقدمه ذکر شد، سامانه در این مرحله از پیاده‌سازی به سه ماژول کلی شکسته شده است: ماژول درس، ماژول دانشجو و ماژول اصلی. در ادامه در مورد هر یک از این ماژول‌ها و موجودیت‌ها و توابع کمکی مربوط به هر کدام توضیح داده خواهد شد.

۳-۲-۱ ماژول درس^۱

این ماژول شامل چهار نوع داده ای اصلی برای مفاهیم درس، رابطه پیش‌نیازی، زمان کلاس و واحد ارائه شده است.

➤ درس

این مفهوم در فضای سامانه انتخاب واحد، دارای نام و کد درس، تعداد واحد مشخص و لیستی از پیش‌نیازهاست. پیاده‌سازی معادل آن نیز شامل همین فیلدهاست (قطعه کد ۱). کد درس حکم شناسه آن را دارد و برای مقایسه دو درس استفاده می‌شود.

¹ Data type

² Course

³ Offering

در حالت کلی این کد یک رشته^۱ در نظر گرفته می‌شود و در این پیاده‌سازی برای خوانایی بیشتر کد، یک نوع داده معادل با نام **Code** برای آن تعریف شده‌است.

لیست پیش‌نیازها بدیتهای لیستی از درس‌های دیگر است (و نه واحد ارائه شده) که در برنامه‌نویسی شیء‌گرا از اشاره‌گرها برای پیاده‌سازی آن استفاده می‌شود. در برنامه‌نویسی تابعی چون عملیات متغیرها تغییر داده نمی‌شوند و نسخه جدیدی از آن‌ها ساخته می‌شود، وجود اشاره‌گرها بی‌معنی است. برای پیاده‌سازی چنین فیلدی دو راه پیش روی ما قرار دارد:

- (۱) لیستی از خود نوع داده ای درس داشته باشیم. این پیاده‌سازی منجر به تکرار و داشتن چند نسخه از یک درس در قسمت‌های مختلف خواهد شد (به طور مثال اگر درس الف پیش‌نیاز ۵ درس باشد، در لیست هر ۵ درس نسخه‌ای از آن وجود خواهد داشت). اگر درس‌ها و همچنین لیست پیش‌نیازها هیچوقت تغییر نمی‌کردند این انتخاب مشکلی ایجاد نمی‌کرد اما ما در این سیستم درخواست‌های تغییر درس، حذف درس، حذف یا اضافه پیش‌نیازها خواهیم داشت و با این نوع از پیاده‌سازی، تمام نسخه‌های درس در هر قسمتی که باشند باید تغییر داده شوند که معقول به نظر نمی‌رسد.
- (۲) لیستی از شناسه درس داشته باشیم. گرچه کمی کدنویسی بیشتری برای پیاده‌سازی این روش لازم است، تقریباً مفهوم اشاره‌گر را منتقل می‌کند و مشکل داده‌های تکراری را ندارد. این راه در این پروژه انتخاب شده‌است.

```
data Course = Course { cName :: String
                      , cCode :: Code
                      , cPrerequisites :: [Code]
                      , unitCount :: Float
                      } deriving (Show)
```

قطعه کد ۱: نوع داده ای درس

➤ رابطه پیش‌نیازی

رابطه پیش‌نیازی بین دو درس تعریف می‌شود. بنابراین در این مفهوم درس الف یک درس وابسته و درس ب یک درس مستقل است (به عبارت دیگر، درس ب پیش‌نیاز درس الف است). بدیهی است که درس ب، خود می‌تواند در رابطه‌های پیش‌نیازی دیگر، وابسته نیز باشد. پیاده‌سازی این مفهوم نیز با همان دو انتخابی که در مفهوم درس شرح دادیم مواجه است چون باید به دو درس اشاره کند و با همان استدلال، راه دوم انتخاب شده است (قطعه کد ۲).

یکی از درخواست‌های سیستم انتخاب واحد، اضافه کردن یک درس به لیست پیش‌نیازهای یک درس دیگر است بنابراین در این قسمت یک تابع کمکی نیاز خواهیم داشت. این تابع یک درس و یک رابطه پیش‌نیازی گرفته و یک درس به عنوان خروجی می‌دهد. اگر شناسه درس وابسته در رابطه پیش‌نیازی برابر با شناسه درس ورودی باشد، شناسه درس مستقل به لیست پیش‌نیازهای آن اضافه می‌شود و در غیر این صورت، درس بدون تغییر باقی خواهد ماند. همچنین تابع مشابهی برای حذف یکی از پیش‌نیازهای یک درس مورد نیاز است. از این دو تابع در پیاده‌سازی درخواست‌ها در ماژول اصلی استفاده خواهد شد.

```
data Prerequisite = Prerequisite { dependent :: Code,
                                  dependee :: Code
                                } deriving (Show, Eq)
```

قطعه کد ۲: نوع داده ای رابطه پیش‌نیازی

¹ String

² Type Synonym

➤ زمان ارائه کلاس

این مفهوم یک مفهوم کمکی مورد استفاده در واحد ارائه شده است. با توجه به اینکه معیار طراحی ما سیستم انتخاب واحد دانشگاه تهران (سیستم گلستان) می‌باشد، زمان ارائه کلاس از ساعت، دقیقه و روز هفته تشکیل می‌شود. برای پیاده‌سازی روز هفته می‌توان از اعداد یا رشته‌ها استفاده کرد ولی برای خوانایی بیشتر، یک نوع داده ای کمکی با نام `Day` مطابق قطعه کد ۳ تعریف شده است. این مفهوم می‌تواند یکی از مقادیر ۷ روز هفته را داشته باشد.

```
data Day = Saturday | Sunday | Monday | Tuesday | Wednesday | Thursday | Friday
    deriving (Eq, Ord, Show, Read, Bounded, Enum)
```

قطعه کد ۳: نوع داده ای روز

اما با توجه به اینکه برخی کلاس‌ها یک روز در هفته و برخی دیگر دو روز در هفته تشکیل می‌شوند، برای انعطاف‌پذیر بودن پیاده‌سازی یک نوع داده ای معادل با نام `ClassDay` تعریف شده‌است که یا معادل با یک روز است و یا یک زوج از روز. به کمک این مفهوم، زمان ارائه کلاس سه فیلد مطابق قطعه کد ۴ خواهد داشت. تابع کمکی خاصی در این قسمت مورد نیاز نیست.

```
data ClassDay = Single Day | Double (Day, Day)
    deriving (Show, Eq)
```

```
data ClassTime = ClassTime { days :: ClassDay,
    hour :: Int,
    minute :: Int
    } deriving (Show, Eq)
```

قطعه کد ۴: نوع داده ای روز کلاس و زمان کلاس

➤ واحد ارائه شده

این مفهوم معادل ارائه یک درس توسط یک استاد در یک ساعت خاص با ظرفیت مشخص است که لیستی از دانشجویانی که آن را اخذ کرده‌اند و لیستی از دانشجویان در انتظار دارد (قطعه کد ۵). برای ذخیره درس مطابق با استدلالی که بالاتر ذکر شده‌بود از شناسه آن استفاده می‌شود. یک رشته به عنوان نام استاد استفاده می‌شود، ظرفیت یک عدد صحیح است و زمان ارائه از جنس نوع داده ای زمان کلاس است. دقت شود که در این قسمت خود زمان کلاس را می‌گذاریم چون مختص به همین واحد ارائه شده است و نیازی نیست یک نسخه از آن در کل سیستم موجود باشد.

همانطور که گفته شد به دو لیست از دانشجویان در این پیاده‌سازی احتیاج داریم. اینجا هم با همان انتخاب بین نگه داشتن نسخه‌ای از خود دانشجو و یا شناسه آن مواجه هستیم که با استدلال مشابه، لیستی از شناسه دانشجویان را ذخیره می‌کنیم (شناسه دانشجو یک رشته است و برای خوانایی بیشتر یک نوع داده ای معادل برای آن به نام `SID` تعریف شده است).

```
data Offering = Offering { course :: Code,
    teacher :: String,
    time :: ClassTime,
    capacity :: Int,
    takenList :: [SID],
    waitingList :: [SID],
    canTake :: Bool
    } deriving (Show)
```

قطعه کد ۵: نوع داده ای واحد ارائه شده

دو درخواست اصلی در سیستم با این مفهوم سر و کار دارد: اخذ واحد ارائه شده توسط دانشجو و اضافه شدن دانشجو به لیست انتظار. بنابراین دو تابع کمکی در اینجا نوشته می‌شود که شناسه دانشجو و یک واحد ارائه شده را گرفته و یک واحد ارائه شده به عنوان خروجی می‌دهند و دانشجو را به لیست مناسب اضافه می‌کنند.

تابع دیگری نیز برای بررسی ظرفیت باقی مانده واحد ارائه شده مورد نیاز است که مقدار ظرفیت را با تعداد دانشجویانی که در لیست اخذ کرده‌ها وجود دارند مقایسه می‌کند. اینکه با توجه به ظرفیت باقی مانده از واحد ارائه شده کدام یک از توابع صدا زده شوند و همچنین بقیه بررسی‌های لازم برای اخذ یک درس توسط دانشجو، به وضعیت سیستم احتیاج دارد و در مازول اصلی به آن پرداخته خواهد شد.

از درخواست‌های دیگر سامانه که با این مفهوم مرتبط است می‌توان به حذف درس، ترک لیست انتظار توسط دانشجو، تغییر ظرفیت واحد ارائه شده و بستن یا باز کردن امکان اخذ واحد ارائه شده اشاره کرد. برای حذف درس و یا ترک لیست انتظار، نیاز است دو تابع کمکی تعریف شود که شناسه دانشجو و یک واحد ارائه شده را گرفته و در صورتی که شناسه دانشجو در لیست مربوطه (لیست اخذشدگان و یا لیست انتظار) یافت شود، آن را حذف کند.

درخواست بستن یا باز کردن امکان اخذ واحد ارائه شده تنها نیازمند تغییر فیلد مربوطه در واحد ارائه شده است و اما برای درخواست تغییر ظرفیت، باید تابعی تعریف شود که پس از تغییر ظرفیت بررسی کند اگر دانشجویی در لیست انتظار است، به تعداد ممکن آن‌ها را به لیست اخذشدگان منتقل کند (و اقدامات بعدی اخذ درس که در مازول دانشجو توضیح داده خواهد شد).

۲-۲-۳ مازول دانشجو^۱

این مازول شامل پنج نوع داده‌ای اصلی برای نام ترم، رکورد اخذ درس و نمره، کارنامه ترمی، کارنامه و دانشجو می‌باشد.

➤ نام ترم

این نوع داده‌ای برای شناسایی یک ترم مشخص در کارنامه ترمی (و همچنین در مفهوم ترم در مازول اصلی) استفاده می‌شود. مطابق ترم‌های دانشگاه تهران، برای شناسایی هر ترم دانستن سال آن و فصل کافی است (به طور مثال در سال اخیر ترم‌های تابستان ۱۳۹۹، پاییز ۱۳۹۹ و بهار ۱۴۰۰). بنابراین نام ترم مطابق قطعه کد ۶، یک زوج از سال (معادل یک عدد صحیح با نوع داده‌ای معادل) و فصل (یک نوع داده‌ای که می‌تواند یکی از سه مقدار بهار، تابستان یا پاییز را داشته باشد) است.

```
data Season = Spring | Summer | Fall
    deriving (Eq, Show, Ord, Read, Enum, Bounded)
```

```
type Year = Int
```

```
type TermName = (Year, Season)
```

قطعه کد ۶: نوع داده‌ای فصل، سال و نام ترم

➤ رکورد اخذ درس

پس از اخذ یک درس، یک رکورد متناسب با آن به کارنامه ترمی دانشجو اضافه خواهد شد. این رکورد شامل درس، نمره و وضعیت است. برای ذخیره درس از شناسه آن استفاده شده است و نمره یک عدد اعشاری است.

¹ Student

فیلد وضعیت یک نوع داده ای کمکی است که می تواند مطابق قطعه کد ۷، یکی از ۴ مقدار تمام نشده، پاس شده، افتاده و یا بی تاثیر داشته باشد. رکوردهای تمام نشده رکوردهایی هستند که برای تشخیص واحدهای اخذ شده در این ترم و یا واحدهای تمدید شده برای ترم بعد (که نمره آن ها در معدل محاسبه نمی شود) استفاده می شوند. رکوردهای پاس شده هم در تعداد واحد گذرانده و هم در معدل موثرند، رکوردهای افتاده فقط در معدل موثرند و در تعداد واحد گذرانده نمی آیند و رکوردهای بی تاثیر، در هیچ یک از معدل و یا تعداد واحد گذرانده موثر نمی باشند.

```
data Status = Unfinished | Passed | Failed | Ineffective
    deriving (Eq, Show, Read)
```

```
data Record = Record { course :: C.Code,
    grade :: Maybe Float,
    status :: Status
} deriving (Show)
```

قطعه کد ۷: نوع داده ای وضعیت رکورد و رکورد اخذ درس

از روی این رکورد می توانیم تشخیص بدهیم که آیا واحد پاس شده (با توجه به نمره و وضعیت) و اینکه آیا واحد در معدل موثر است یا خیر. دو تابع کمکی برای این دو منظور نوشته شده است که در توابع محاسبه معدل استفاده می شوند. هرچند نمره موثر یک رکورد برابر با حاصل ضرب نمره آن در تعداد واحد درس است (در صورت موثر بودن در معدل)، اما با توجه به اینکه در رکورد فقط شناسه درس ذخیره می شود برای به دست آوردن تعداد واحد آن باید درس را از ریپازیتوری اصلی سیستم (مفهوم وضعیت) پیدا کنیم که در مازول اصلی به آن پرداخته خواهد شد.

➤ کارنامه ترمی

کارنامه ترمی هر دانشجو متشکل از نام ترم (برای شناسایی) و لیستی از رکوردهای اخذ درس متعلق به آن ترم است. با توجه به اینکه نام ترم مختص به همین کارنامه ترمی است و نیازی نیست تنها یک نسخه از آن در سیستم موجود باشد، خود آن ذخیره خواهد شد و تنها محل ذخیره رکوردهای این ترم برای این دانشجو نیز همین قسمت است (قطعه کد ۸).

```
data TermTranscript = TermTranscript { records :: [Record],
    termName :: TermName
} deriving (Show)
```

قطعه کد ۸: نوع داده ای کارنامه ترمی

در صورتی که دانشجو در این ترم درسی را اخذ کند، باید رکورد جدیدی ساخته شده و به کارنامه ترمی آن اضافه شود. بنابراین یک تابع کمکی در این قسمت نیاز داریم که یک واحد ارائه شده، یک نام ترم و یک کارنامه ترمی را بگیرد و یک کارنامه ترمی به عنوان خروجی بدهد. در صورتی که کارنامه ترمی متعلق به همین ترم بود (از روی مقایسه نام ترمها)، رکورد جدیدی با وضعیت ناتمام و شناسه درس ذخیره شده در واحد ارائه شده ساخته شده و به لیست رکوردهای ترم اضافه می شود و در غیر این صورت، کارنامه ترمی بی تغییر می ماند.

محاسبه معدل ترم از نمرات رکوردها انجام می شود که طبق توضیحات قسمت قبل احتیاج به پیدا کردن درس از روی شناسه آن در ریپازیتوری سیستم دارد و در مازول اصلی به آن پرداخته خواهد شد.

➤ کارنامه

این مفهوم تنها متشکل از لیست کارنامه‌های ترمی دانشجو است و چون تنها محل ذخیره آن‌ها این قسمت است، از خود آن‌ها استفاده می‌شود (قطعه کد ۹).

```
data Transcript = Transcript {termTranscripts :: [TermTranscript]} deriving(Show)
```

قطعه کد ۹: نوع داده ای کارنامه

➤ دانشجو (Student)

مفهوم دانشجو برای ساده‌سازی، متشکل از نام (یک رشته)، شناسه و کارنامه در نظر گرفته شده است و سایر اطلاعات شناسایی حذف شده‌اند (قطعه کد ۱۰).

```
data Student = Student { sName :: String,
                        sId :: C.SID,
                        sTranscript :: Transcript
                      } deriving(Show)
```

قطعه کد ۱۰: نوع داده ای دانشجو

تابع کمکی این قسمت نیز برای اخذ درس است. این تابع یک واحد ارائه شده، یک نام ترم و یک دانشجو را می‌گیرد و یک دانشجو به عنوان خروجی می‌دهد. در این تابع وظیفه دارد تابع کمکی اخذ درس در کارنامه ترمی را روی لیست کارنامه‌های ترمی دانشجو (با استفاده از کارنامه او) اجرا کند. نتیجه احتمالی این خواهد بود که به یکی از کارنامه‌های ترمی یک رکورد ناتمام از این درس اضافه می‌شود و سایر کارنامه‌های ترمی بی‌تغییر می‌مانند. محاسبه معدل کل دانشجو با استفاده از نمرات رکوردهای داخل کارنامه‌های ترمی انجام می‌شود که مطابق توضیحات قبلی، احتیاج به پیدا کردن درس از روی شناسه آن در ریزپایتوری سیستم دارد و در مازول اصلی به آن پرداخته خواهد شد.

۳-۲-۳ مازول اصلی^۱

این مازول شامل دو نوع داده ای اصلی برای مفاهیم ترم و وضعیت سیستم و همچنین تمام توابع لازم سیستم برای پیاده‌سازی درخواست‌ها می‌باشد.

➤ ترم

مفهوم ترم با یک شناسه (از جنس نام ترم) شناخته شده و دارای یک لیست از واحدهای ارائه شده است (قطعه کد ۱۱).

```
data Term = Term { offerings :: [C.Offering],
                  termName :: S.TermName
                } deriving (Show)
```

قطعه کد ۱۱: نوع داده ای ترم

¹ Main

➤ وضعیت سیستم

وضعیت کنونی سیستم در حکم ریپازیتوری سیستم و مرجع ما برای یافتن داده‌ها یا تغییر دادن آن‌هاست. وضعیت سیستم در هر لحظه متشکل از لیستی از دروس تعریف شده (از جنس درس است نه شناسه آن چون نسخه اصلی آن در این ریپازیتوری نگهداری می‌شود)، لیستی از دانشجویان (از جنس دانشجو و نه شناسه به دلیل مشابه) و لیستی از ترم‌هاست (قطعه کد ۱۲).

```
data State = State { courses :: [C.Course],
                    terms :: [Term],
                    students :: [S.Student]
                  } deriving(Show)
```

قطعه کد ۱۲: نوع داده ای وضعیت

- تعدادی تابع کمکی برای به‌روزرسانی وضعیت سیستم و اضافه کردن یا تغییر فیلدهای آن مورد نیاز است از جمله:
- اضافه کردن درس به سیستم: یک درس و وضعیت سیستم را می‌گیرد و یک وضعیت خروجی می‌دهد.
 - اضافه کردن ترم به سیستم: یک ترم و وضعیت سیستم را می‌گیرد و یک وضعیت خروجی می‌دهد.
 - اضافه کردن دانشجو به سیستم: یک دانشجو و وضعیت سیستم را می‌گیرد و یک وضعیت خروجی می‌دهد.
 - اضافه کردن رابطه پیش‌نیازی به سیستم: یک رابطه پیش‌نیازی و یک وضعیت سیستم می‌گیرد و یک وضعیت خروجی می‌دهد. این تابع کافی است از تابع کمکی که در مازول درس برای اضافه کردن رابطه پیش‌نیازی به درس تعریف کرده بودیم، روی کل دروس سیستم استفاده کند (تابع کمکی ذکر شده در صورتی که شناسه درس وابسته در رابطه پیش‌نیازی برابر با شناسه درس نباشد، آن را تغییر نمی‌دهد).
 - اضافه کردن واحد ارائه شده به سیستم: این تابع یک واحد ارائه شده، یک نام ترم و یک وضعیت سیستم را می‌گیرد و یک وضعیت به عنوان خروجی می‌دهد. برای پیاده‌سازی کافی است نام ترم ورودی را با نام ترم‌های سیستم مقایسه کند، هر کدام برابر نبود را بی‌تغییر بگذارد و به هرکدام برابر بود، واحد ارائه شده را اضافه نماید.
 - به‌روزرسانی اطلاعات یک دانشجو در سیستم: این تابع یک دانشجو و وضعیت سیستم را می‌گیرد، دانشجو را با شناسه آن پیدا می‌کند و اطلاعات جدید را جایگزین می‌کند و سپس یک وضعیت به عنوان خروجی می‌دهد.
 - به‌روزرسانی یک واحد ارائه شده در سیستم: این تابع یک واحد ارائه شده، یک نام ترم و وضعیت سیستم را می‌گیرد، واحد ارائه شده را با درس و استاد و زمان کلاس در ترمی با نام ترم داده شده پیدا می‌کند و اطلاعات جدید را جایگزین می‌کند و سپس یک وضعیت به عنوان خروجی می‌دهد. از این تابع وقتی استفاده می‌شود که درسی توسط دانشجو اخذ، حذف، وارد انتظار و یا خارج از انتظار شود و بخواهیم لیست دانشجویانی که این واحد را اخذ کرده‌اند (و یا لیست دانشجویان در انتظار) را تغییر دهیم.

۳-۳ درخواست‌های سامانه

وظیفه اصلی سیستم انتخاب واحد، پاسخگویی به درخواست‌ها و تغییر وضعیت سیستم متناسب با نتیجه درخواست است. درخواست‌های مختلف سیستم به شرح زیر پیاده‌سازی شده‌اند:

➤ درخواست محاسبه معدل کل یا معدل ترمی دانشجو

هر دوی این درخواست‌ها به روش مشابهی پیاده‌سازی می‌شوند و یک دانشجو و یک وضعیت به عنوان ورودی می‌گیرند و یک وضعیت خروجی می‌دهند (برای محاسبه معدل یک ترم خاص باید نام ترم نیز به ورودی‌ها اضافه شود). برای محاسبه معدل باید قدم‌های زیر را طی کرد:

- (۱) رکوردهای دروس را از کارنامه‌های ترمی پیدا می‌کنیم.
 - (۲) باید جمع نمرات ضریب‌دار این رکوردها را بیابیم. برای رکوردهایی که در معدل موثر هستند، حاصل ضرب نمره رکورد در تعداد واحد درس برگردانده می‌شود و برای سایر رکوردها صفر برگردانده می‌شود تا در جمع کل نمرات بی‌تاثیر باشند. اما همانطور که در طراحی رکورد ذکر کردیم، رکورد فقط شناسه درس را ذخیره می‌کند و برای یافتن تعداد واحد درس باید با استفاده از این شناسه، درس را از ریپازیتوری (وضعیت سیستم) پیدا کنیم و سپس تعداد واحد آن را برگردانیم. (یک تابع کمکی برای یافتن تعداد واحد درس با استفاده از شناسه آن و وضعیت سیستم در این قسمت برای خوانایی بیشتر کد اضافه شده است).
 - (۳) تعداد واحد موثر در معدل برابر با جمع تعداد واحدهای رکوردهای موثر در معدل است که با استفاده از تابع کمکی یافتن تعداد واحد درس از وضعیت سیستم محاسبه می‌شود.
 - (۴) در نهایت از تقسیم جمع نمرات ضریب‌دار بر تعداد واحد موثر، معدل محاسبه می‌شود.
- دقت شود که در محاسبه معدل، داده‌ای از سیستم تغییر داده نمی‌شود. از توابع پیاده‌سازی شده برای این درخواست در درخواست اخذ درس (برای بررسی شرط معدل و سقف واحد) استفاده می‌شود.

➤ درخواست اخذ واحد

این درخواست برای اخذ یک درس توسط یک دانشجو در یک ترم خاص مطرح می‌شود بنابراین ورودی‌های تابع، دانشجو، نام ترم، واحد ارائه شده و وضعیت سیستم است و یک وضعیت سیستم نیز به عنوان خروجی داده می‌شود. برای اجرا شدن این درخواست، ابتدا برقراری شروط زیر بررسی می‌شود:

- (۱) دانشجو این درس را پیش از این پاس نکرده باشد. برای بررسی این شرط، تمامی رکوردهای کارنامه‌های ترمی دانشجو که شناسه‌ای برابر با شناسه این درس دارند بررسی می‌شوند تا وضعیت پاس شده نداشته باشند.
- (۲) دانشجو تمام پیش‌نیازهای این درس را گذرانده باشد. برای بررسی این شرط، ابتدا باید پیش‌نیازهای درس پیدا شوند. با استفاده از شناسه درس که در واحد ارائه وجود دارد، درس و لیست پیش‌نیازهای آن را از ریپازیتوری (وضعیت سیستم) می‌گیریم. سپس مشابه شرط قبلی، تمامی رکوردهای کارنامه‌های ترمی دانشجو بررسی می‌شوند تا به ازای هر درس پیش‌نیاز، حداقل یک رکورد پاس شده موجود باشد.
- (۳) سقف تعداد واحد قابل اخذ با معدل دانشجو مغایر نباشد. برای بررسی این شرط، ابتدا معدل دانشجو را با توابع درخواست قبلی محاسبه می‌کنیم. برای یافتن تعداد واحد اخذ شده این ترم، رکوردهای ناتمام در کارنامه ترمی با این نام ترم را یافته و سپس با استفاده از شناسه درس در آن رکوردها، درس‌ها و تعداد واحدهای آن‌ها را از ریپازیتوری گرفته و حاصل جمع آن را محاسبه می‌کنیم. در صورتی که معدل دانشجو بالای ۱۷ باشد، دانشجو مجاز به اخذ ۲۴ واحد و در غیر این صورت، مجاز به اخذ ۲۰ واحد است.

۴) امکان اخذ این واحد ارائه شده وجود داشته باشد.

در صورتی که حداقل یکی از شروط فوق برقرار نباشد، دانشجو شرایط اخذ درس را ندارد و وضعیت سیستم بدون تغییر می ماند. و در غیر این صورت باید بررسی شود که آیا واحد ارائه شده ظرفیت خالی دارد یا خیر. اگر واحد ارائه شده دارای ظرفیت خالی نباشد، دانشجو وارد لیست انتظار این واحد می شود بنابراین واحد ارائه شده، ترم و وضعیت سیستم به روزرسانی می شوند اما چون رکوردی به کارنامه دانشجو اضافه نمی شود، دانشجو بی تغییر باقی می ماند. و اگر واحد ارائه شده دارای ظرفیت باشد، شناسه دانشجو به لیست دانشجویان اخذ کرده اضافه می شود و رکورد جدیدی با وضعیت ناتمام و شناسه این درس به کارنامه ترمی دانشجو (متعلق به این نام ترم) اضافه می شود. بنابراین واحد ارائه شده، ترم، دانشجو و وضعیت سیستم به روزرسانی می شوند. برای به روزرسانی این داده ها از توابع کمکی که بالاتر توضیح داده شد استفاده می شود.

➤ درخواست حذف واحد

این درخواست برای حذف یک واحد توسط یک دانشجو در یک ترم خاص مطرح می شود بنابراین ورودی های تابع، دانشجو، نام ترم، واحد ارائه شده و وضعیت سیستم است و یک وضعیت سیستم نیز به عنوان خروجی داده می شود. برای اجرا شدن این درخواست، ابتدا برقراری شروط زیر بررسی می شود:

۱) دانشجو این واحد را اخذ کرده باشد. یعنی رکوردی با وضعیت ناتمام و با شناسه این درس این واحد ارائه شده در کارنامه ترمی (متعلق به این ترم) موجود باشد.

۲) با حذف این درس، تعداد واحد اخذ شده دانشجو در این ترم کمتر از ۱۲ واحد نشود.

در صورت برقراری هر دو شرط فوق، رکورد اخذ این واحد از کارنامه دانشجو حذف شده و شناسه دانشجو از لیست دانشجویانی که این واحد ارائه شده را اخذ کرده اند، حذف می شود. در صورتی که کسی در لیست انتظار درس باشد، باید از لیست انتظار حذف شده و درس برای او اخذ شود بنابراین اولین شناسه دانشجو از لیست انتظار حذف شده، دانشجو از ریزبازرسی سیستم پیدا می شود و سپس فرآیند اخذ واحد همانطور که در قسمت قبل ذکر شد تکرار می شود.

➤ درخواست ترک انتظار

این درخواست برای خروج از لیست انتظار یک واحد ارائه شده توسط یک دانشجو در یک ترم خاص مطرح می شود بنابراین ورودی های تابع، دانشجو، نام ترم، واحد ارائه شده و وضعیت سیستم است و یک وضعیت سیستم نیز به عنوان خروجی داده می شود. این تابع صرفاً وظیفه دارد در صورتی که شناسه دانشجو در لیست انتظار این واحد ارائه شده موجود بود، آن را حذف کند و سپس این واحد ارائه شده، ترم و وضعیت سیستم به روزرسانی می شوند.

➤ درخواست تغییر ظرفیت واحد ارائه شده

این درخواست یک واحد ارائه شده، یک عدد برای ظرفیت جدید، یک نام ترم و وضعیت سیستم را دریافت می کند و یک وضعیت به عنوان خروجی می دهد. برای اجرای این درخواست کافی است ظرفیت واحد ارائه شده با عدد جدید به روزرسانی شود و سپس واحد ارائه شده، ترم و وضعیت سیستم به روزرسانی می شوند (با توجه به منطق پیاده سازی سیستم گلستان، نیازی به بررسی تعداد دانشجویانی که پیش از این درس را اخذ کرده اند نیست و ممکن است تعداد آن ها بیشتر از ظرفیت جدید باشد. در صورتی که ظرفیت افزایش پیدا کند و دانشجویانی در لیست انتظار باشند، باید درس به طور خودکار برای آن ها اخذ شود).

➤ بستن یا باز کردن امکان اخذ واحد

این درخواست تنها وظیفه دارد فیلد `canTake` از واحد ارائه شده را ست یا ریست کند. ورودی‌های تابع یک واحد ارائه شده، یک نام ترم، و وضعیت سیستم است که با به‌روزرسانی واحد ارائه شده و ترم و وضعیت، یک وضعیت جدید به عنوان خروجی می‌دهد.

۳-۴ معماری مبتنی بر رویداد

در این معماری، تلاش اصلی بر این است که تغییرات مربوط به هر موجودیت، توسط خود آن بررسی و انجام شود و نه توسط مفهوم وضعیت. برای این منظور، با الگوبری از مرجع [۲]، موجودیت‌های گسترده‌تر که هر یک شامل تعدادی خصیصه و تابع می‌شوند و پیش از این تغییرات آن‌ها به صورت غیرمستقیم با صدا زدن توابعی در مفهوم وضعیت انجام می‌شد، به صورت نمونه‌ای از کلاس `Aggregate` تعریف می‌کنیم.

کلاس `Aggregate` که آن را طبق قطعه کد ۱۳ تعریف می‌کنیم، خاصیت‌های زیر را به موجودیت‌ها اضافه می‌کند:

- (۱) نوع داده‌ای با نام دستور به ازای این موجودیت تعریف می‌شود. این رویداد درجه ارتباط ما با این موجودیت است. به عبارت دیگر، اگر می‌خواهیم تغییری در خصیصه‌های موجودیت اعمال کنیم، به جای اینکه اعتبارسنجی‌ها و اعمال تغییر مستقیماً توسط ما انجام شود، با فراخوانی این دستور روی موجودیت، تمام این وظایف را به خودش می‌سپاریم.
- (۲) نوع داده‌ای با نام رویداد به ازای این موجودیت تعریف می‌شود. وقتی دستوری روی این موجودیت فراخوانی می‌شود، با اجرای تابع `execute`، اعتبارسنجی‌های لازم انجام می‌شود. در صورتی که دستور از شرایط لازم برخوردار باشد، یک رویداد روی این موجودیت ساخته می‌شود که با تابع `apply`، تغییرات هدف را روی موجودیت اعمال می‌کند.
- (۳) نوع داده‌ای با نام خطا به ازای این موجودیت تعریف می‌شود. در صورتی که در مرحله بررسی دستور، یک یا چندتا از مراحل اعتبارسنجی موفقیت‌آمیز نباشند، دستور به مرحله ساخت رویداد و اجرای تغییرات نمی‌رسد و یک خطا (از جنس خطاهای مربوط به این موجودیت) ساخته می‌شود.
- (۴) دو تابع برای بررسی دستور و اجرای رویدادها تعریف می‌شود که باید به ازای هر رویداد یا دستوری که روی این موجودیت تعریف کرده‌ایم، نسخه متناظر این تابع را پیاده‌سازی کنیم. تابع دیگری نیز برای ساختن یک موجودیت پایه (و مقداره‌ی اولیه خصیصه‌ها) با نام `seed` تعریف می‌شود.

```
class Aggregate s where
  data Error s :: *
  data Command s :: *
  data Event s :: *

  execute :: s -> Command s -> Either (Error s) (Event s)
  apply :: s -> Event s -> s
  seed :: s
```

قطعه کد ۱۳: تعریف کلاس `Aggregate`

در این مرحله، هر یک از موجودیت‌ها که ویژگی‌های لازم و گستردگی قابل قبولی دارند و می‌توان برای آن‌ها رویداد و دستوری متصور شد، به عنوان نمونه‌ای از این کلاس تعریف می‌کنیم. با توجه به نحوه تعریف مفهوم وضعیت سیستم که در بخش ۳-۲-۳ به آن پرداخته شد، وضعیت در این توصیف، در حکم درجه اعمال تغییرات در سامانه و همچنین ریپازیتوری دریافت و ذخیره

موجودیت‌ها عمل می‌کند. سه موجودیت اصلی که از طریق وضعیت به آن‌ها دسترسی داریم و تغییرات اصلی روی آن‌ها اعمال می‌شوند عبارت‌اند از موجودیت‌های درس، دانشجو و ترم. علاوه بر این سه موجودیت، خود وضعیت را نیز به عنوان نمونه‌ای از این کلاس تعریف می‌کنیم تا درخواست‌های سامانه به صورت یک دستور روی موجودیت وضعیت اعمال شوند. بنابراین فرآیند انجام یک درخواست در سامانه به صورت زیر تغییر می‌یابد:

- (۱) دستوری روی موجودیت وضعیت با پارامترهای لازم برای درخواست، فراخوانده می‌شود.
- (۲) در تابع `execute`، اعتبارسنجی‌های لازم و برقراری شرط‌های انجام دستور بررسی می‌شود.
- (۳) در صورتی که یک یا چندتا از شروط برقرار نباشند، خطای مناسب ایجاد شده و فرآیند متوقف می‌شود (لازم به ذکر است که برای هر اعتبارسنجی می‌توان نوع خاصی از خطا را ایجاد کرد تا پیام و علت دقیق خطا مشخص باشد).
- (۴) در صورتی که تمام شروط برقرار باشند، رویداد متناظر با پارامترهای لازم ساخته می‌شود.
- (۵) در فرآیند اجرای رویداد، ممکن است درخواست در حد اضافه کردن یک درس به سیستم باشد که می‌تواند به صورت مستقیم انجام شود و نیازی به مرحله دیگری ندارد. در صورتی که درخواست پیچیده‌تر باشد (مانند اخذ واحد که باید هم تغییراتی در دانشجو و رکوردهای اخذ درس آن صورت گیرد و هم واحد ارائه شده تغییر کند و شناسه دانشجو به لیست اخذشدگان اضافه شود)، می‌توان آن را به صورت دستورهای مختلفی روی هر موجودیت فراخوانی کرد و ادامه بررسی‌ها و اعمال تغییرات نهایی را به خود موجودیت سپرد.

به عنوان مثال، یک نمونه از تعریف موجودیت‌ها به عنوان یک `Aggregate` را بررسی می‌کنیم. در قطعه کد ۱۴، می‌بینیم که تنها درخواست و تغییر مربوط به موجودیت درس (به غیر از اضافه کردن یک درس به سیستم که مستقیماً در موجودیت وضعیت انجام می‌شود و ارتباطی به داخل موجودیت درس ندارد)، اضافه کردن یک رابطه پیش‌نیازی است. دستور اضافه کردن رابطه پیش‌نیازی دو پارامتر کد درس وابسته و کد درس پیش‌نیاز دارد که در مرحله اعتبارسنجی، باید ببینیم آیا شناسه این موجودیت برابر با کد درس وابسته هست یا خیر. در صورتی که برابر نباشد، خطای متناسب برای آن تعریف شده است و در صورتی که برابر باشد، رویداد متناظر با پارامترهای درخواست ساخته می‌شود و در تابع مربوطه، تغییرات روی موجودیت انجام می‌گیرد (کد درس پیش‌نیاز به لیست پیش‌نیازی‌های این درس اضافه می‌گردد).

```
instance Aggregate C.Course where
  data Error C.Course = NotForThisCourse
    deriving(Show)
  data Event C.Course = C_PrerequisiteAdded C.Code C.Code
    deriving(Show)
  data Command C.Course = C_AddPrerequisite C.Code C.Code
    deriving(Show)

  course `execute` C_AddPrerequisite dependee dependent = C_PrerequisiteAdded dependee
    <$> validate (C.cCode course ==) NotForThisCourse dependent

  course `apply` C_PrerequisiteAdded dependee dependent = C.Course { C.cName = C.cName cours
e, C.cCode = C.cCode course, C.cPrerequisites = dependee:C.cPrerequisites course}

seed = C.Course { C.cName = "", C.cCode = "", C.cPrerequisites = [], C.unitCount = 1}
-- قطعه کد ۱۴: تعریف موجودیت درس به عنوان نمونه‌ای از کلاس Aggregate
```

از تأثیرات اضافه کردن معماری مبتنی بر رویداد به توصیف سامانه، می‌توان به موارد زیر اشاره کرد:

- (۱) همانطور که در تشریح فرآیند اجرای یک درخواست مشخص است، کپسوله‌سازی^۱ و سطح دسترسی در کل موجودیت‌ها بسیار بهبود یافت. در این معماری، کسی جز خود موجودیت مسئول بررسی درخواست‌ها و انجام دادن تغییرات روی خصیصه‌ها نیست و مفهوم وضعیت به جای اینکه خودش تمام فرآیند اجرای درخواست را طی کند، پس از بررسی شروط اولیه و در سطح خودش، فرآیند را به فراخوانی چند دستور روی موجودیت‌های مختلف می‌شکند و به عبارتی، نقش توزیع‌کننده رویدادها را بازی می‌کند.
- (۲) فرآیند اجرای یک دستور بسیار خواناتر و قابل ردیابی‌تر در کل کد است. پیش از اضافه کردن این معماری به سیستم، ماژول‌ها و درخواست‌ها متشکل از تعداد بالایی تابع بودند که با فراخوانی‌های متعدد، در نهایت تغییر مورد نظر در سیستم انجام می‌گرفت (و یا اگر شرط‌ها برقرار نبودند، انجام نمی‌گرفت). اما اکنون قدم‌های بررسی شروط، اجرای دستور و توزیع دستور روی موجودیت‌های دیگر کاملاً مشخص است.
- (۳) گسترده‌تر کردن این سامانه، اضافه کردن درخواست‌های جدید، حذف درخواست‌های قبلی یا تغییر شرایط آن‌ها، به دلیل مشابه قبلی بسیار راحت‌تر شده است. حتی اگر شخص جدیدی وارد این پروژه شود که با جزئیات تک تک موجودیت‌ها و توابع تعریف شده آشنا نباشد، با کمی بررسی در کد می‌تواند بفهمد قدم اول اضافه کردن یک درخواست، اضافه کردن دستور و رویداد آن به موجودیت وضعیت است و قدم‌های بعدی به ترتیب پیاده‌سازی تابعی برای اعتبارسنجی درخواست و توزیع رویدادهای مناسب به سایر موجودیت‌ها می‌باشد.
- (۴) حجم کد نسبت به پیش از اضافه کردن این معماری افزایش یافته است که در نگاه اول ممکن است گیج‌کننده و زیاد به نظر برسد اما با توجه به ساختارمندی اضافه شده به توصیف و فواید دیگری که ذکر شد، این افزایش حجم منطقی و قابل قبول است.

۵-۳ اصول و الگوها

در طول پیاده‌سازی توصیف سامانه آموزش دانشگاه در این پژوهش، اصول خاصی رعایت شده و الگوهای متعددی بنا به تجربه و آزمون و خطا به دست آمده است که می‌تواند به عنوان راهنمایی‌هایی برای کسی که می‌خواهد در این راه قدم بگذارد و یک سامانه جامع سازمانی - اطلاعاتی را با زبان‌های تابعی (به طور خاص زبان هسکل) توصیف کند مفید باشد. این بخش از گزارش به جمع‌آوری و تشریح این اصول و الگوها اختصاص دارد.

- اولین و مهم‌ترین قدمی که باید پیش از شروع به پیاده‌سازی توصیف (به هر زبانی) انجام داد، طراحی و مدل‌سازی مدل دامنه و تعیین موجودیت‌ها، خصیصه‌های آن‌ها، دامنه گسترش هر یک از موجودیت‌ها و روابط بین آن‌هاست. هرچند ممکن است در طول پیاده‌سازی، مدل دامنه اولیه تغییر کند یا گسترش یابد، طراحی و فکر روی آن پیش از شروع کدنویسی باعث می‌شود پیش‌زمینه ذهنی نسبتاً کاملی از مسیر پیش رو داشته باشید و از قبل در مورد مسائلی مثل نحوه ارتباط موجودیت‌ها، وابستگی‌های آن‌ها به هم، ماژول‌های مورد نیاز و غیره اندیشیده باشید.
- در قدم بعدی، پیاده‌سازی را از پایه‌ترین موجودیت‌ها که احتمالاً وابستگی‌ای به موجودیت دیگری ندارند، شروع کنید. در سامانه‌های بزرگ و پیچیده اطلاعاتی، داشتن مفهومی مانند مفهوم وضعیت در این پروژه کمک می‌کند تا یک نقطه دسترسی به سامانه برای دریافت درخواست‌ها وجود داشته باشد و همچنین نسخه اصلی موجودیت‌های مختلف در این

¹ Encapsulation

² Event Dispatcher

مفهوم به عنوان یک ریپازیتوری مرجع وجود داشته باشند اما این موجودیت به احتمال زیاد بسیار بزرگ، گسترده، پیچیده و وابسته به تعداد زیادی از موجودیت‌های دیگر خواهد بود بنابراین با اینکه طراحی این مفهوم، مفید و حتی ضروری به نظر می‌رسد و می‌توان آن را یکی از الگوها و دستاوردهای این پروژه دانست، انتخاب این موجودیت به عنوان نقطه شروع پیاده‌سازی چندان منطقی به نظر نمی‌رسد. بهتر است در مرحله مدل‌سازی به اینکه چه موجودیت‌هایی باید در ریپازیتوری ذخیره شوند (و به بقیه موجودیت‌ها از طریق آن‌ها دسترسی داشته باشیم) فکر کرد اما این مفهوم احتمالا یکی از آخرین موجودیت‌هایی است که باید پیاده‌سازی شود.

- یکی از چالش‌هایی که در توصیف سامانه با آن مواجه خواهید بود، نحوه ارتباط موجودیت‌های وابسته به هم است (به طور مثال واحد ارائه شده و درس در این پروژه). همانطور که در بخش ۱-۲-۳ توضیح داده شد، دو انتخاب در مواجهه با این چالش وجود دارد. یا باید یک نسخه کپی از موجودیت را در مفهوم وابسته نگه داریم و یا شناسه‌ای از آن را. ذخیره یک نسخه کامل از موجودیت باعث تکرار و وجود داشتن چند نسخه از یک موجودیت واحد در کل سیستم می‌شود. افزودن حاصل از این انتخاب منجر به این می‌شود که برای هر تغییر آتی در این موجودیت، مجبور شویم تمام نسخه‌های کپی آن را در سیستم پیدا کنیم و همه آن‌ها را نیز تغییر دهیم که منطقی به نظر نمی‌رسد. بنابراین با انتخاب دوم، شناسه موجودیت را در مفهوم وابسته ذخیره می‌کنیم و اگر جایی نیاز به نسخه اصلی و یا خصیصه‌های آن داشتیم، از ریپازیتوری برای دریافت نسخه اصلی به کمک شناسه استفاده می‌کنیم.
- زبان هسکل عموماً برای قطعه کدهای کوچک و توصیف سریع استفاده می‌شود. بنابراین در منابع و نمونه‌هایی که در مراجع و اینترنت پیدا می‌کنید، خواهید دید که نام‌گذاری توابع چندان مسئله مهمی نیست. اما در توصیف سامانه بزرگی مانند سامانه آموزش دانشگاه، هر یک از ماژول‌ها و موجودیت‌های مربوط به آن‌ها به تعداد زیادی تابع احتیاج خواهند داشت تا کل سیستم، درخواست‌ها، اعتبارسنجی‌ها و پاسخ‌ها توصیف شود. بنابراین توصیه می‌شود در این توصیف به نام‌گذاری توابع اهمیت داده شده و اسمی متناسب با هدف و وظیفه آن تابع خاص انتخاب شود.
- از لحظه شروع کدنویسی و پیاده‌سازی اولین موجودیت، بنا را بر تعریف توابع کوچک (به احتمال زیاد یک خطی) که هر کدام تنها یک کار انجام می‌دهند بگذارید. درک‌پذیری و خوانایی سیستم با این شیوه بسیار افزایش پیدا می‌کند و از طرفی امکان اینکه توابع کوچک را در موارد مختلفی در توابع دیگر استفاده کنید، بیشتر است.
- گرچه در این پروژه اضافه کردن معماری مبتنی بر رویداد به توصیف، اثرات مثبت قابل توجهی به همراه داشت اما لزوماً این انتخاب برای هر پروژه‌ای نمی‌تواند مناسب باشد. روی تعدادی از معماری‌ها و ساختارهای ممکن برای پروژه خود مطالعه کنید و سپس بهترین راه را برگزینید.
- در زبان هسکل برای خیلی از اهدافی که در ذهن دارید، احتمالاً یک تابع میانبر یک خطی پیش‌بینی و تعریف شده است بنابراین قبل از پیاده‌سازی یک تابع جدید، مطمئن شوید هسکل برای آن امکانی در نظر نگرفته باشد.

۴. ارزیابی

هدف اصلی این پروژه، توصیف تابعی منطق یک سامانه جامع اطلاعاتی است. برای ارزیابی اینکه توصیف تابعی با الگوها و روش‌های ذکرشده چقدر موفق بوده و چقدر با پیاده‌سازی‌های دیگر (مانند زبان‌های شیء‌گرا) تفاوت دارد یا نسبت به آن‌ها ارزشمند است، از مقایسه توصیف نهایی (پس از اضافه کردن معماری مبتنی بر رویداد) با یک پیاده‌سازی ساده سامانه آموزش به زبان جاوا استفاده کردیم. به این منظور، ابتدا باید دو پیاده‌سازی را به مرحله یکسانی برسانیم.

با توجه به اینکه توصیف تابعی سامانه کامل‌تر از پیاده‌سازی به زبان جاوا بود، تغییرات زیر در کد جاوا اعمال شد تا دو پیاده‌سازی در یک سطح از پیچیدگی و درخواست‌ها قرار داشته باشند:

(۱) خصیصه نمره در رکورد اخذ درس در کد جاوا تنها به صورت یک عدد اعشاری تعریف شده بود. بنابراین خصیصه وضعیت نمره که در توصیف تابعی داشتیم (شامل چهار وضعیت پاس شده، افتاده، بی‌تاثیر و ناتمام) را به آن اضافه کردیم. در نتیجه باید توابعی که برای محاسبه معدل از این رکوردها استفاده می‌کردند را به گونه‌ای تغییر دهیم که فقط رکوردهای موثر در معدل را در نظر بگیرد.

(۲) امکان بستن و باز کردن اخذ یک واحد ارائه شده به صورت یک خصیصه در توصیف تابعی تعریف شده است که آن را به کد جاوا اضافه کردیم. همچنین یک شرط دیگر برای بررسی امکان اخذ واحد (در کنترل‌کننده ثبت نام در کد جاوا) باید اضافه شود که باز بودن واحد ارائه شده را چک کند.

(۳) واحد ارائه شده در کد جاوا، خصیصه‌های ظرفیت، لیست اخذشدگان و لیست انتظار نداشت. پس از اضافه کردن این خصیصه‌ها، می‌توان هنگام اخذ واحد بررسی کرد که اگر ظرفیت واحد ارائه شده پر است، دانشجو به لیست انتظار اضافه شود.

(۴) سه درخواست حذف درس، ترک لیست انتظار و تغییر ظرفیت واحد ارائه شده (و در نتیجه آن انتقال دانشجویان از لیست انتظار به لیست اخذشدگان) نیز در کد جاوا پیاده‌سازی نشده بودند که به آن اضافه شدند.

پس از اینکه دو پیاده‌سازی به سطح یکسانی رسیدند، تعدادی سناریوی تغییر سامانه که پیش از این در مدل‌سازی پیش‌بینی نشده بودند، طراحی شد و اثرات اجرای آن‌ها در هر دو پیاده‌سازی مقایسه گردید. سناریوهای تغییر به شرح زیر تعریف شدند:

(۱) درخواست ثبت نمره برای یک دانشجو. برای این تغییر تنها کافی است یک درخواست به سامانه اضافه شود که پارامترهای آن نمره، دانشجو، درس هدف و ترم (شناسه ترم) هستند. ۲۴ خط در کد هسکل در ۲۰ دقیقه و ۳۵ خط در کد جاوا در ۱۰ دقیقه در اثر اجرای این سناریو، تغییر یافت.

(۲) اضافه کردن موجودیت برنامه به پیاده‌سازی. برنامه به معنی موجودیتی است که شامل تعدادی درس و تعدادی رابطه پیش‌نیازی می‌باشد و هر دانشجو، درس‌های خود را مطابق یکی از این برنامه‌ها اخذ می‌کند (تقریباً معادل مفهوم رشته

مهندسی کامپیوتر یا مهندسی برق). برای این تغییر، موجودیت برنامه نیز باید به موجودیت‌هایی که در مفهوم وضعیت ذخیره می‌شوند اضافه گردد و هر دانشجو، شناسه برنامه خود را نیز ذخیره داشته باشد. برای اخذ هر واحد دانشجو به جای اینکه درس هدف و پیش‌نیازی‌های آن را مستقیماً از ریپازیتوری پیدا کنیم، با توجه به شناسه برنامه دانشجو، ابتدا برنامه را از ریپازیتوری پیدا می‌کنیم و سپس درس و پیش‌نیازی‌های آن را از برنامه گرفته و مطابق آن شرایط اخذ واحد را بررسی می‌کنیم. با توجه به اینکه این سناریو منجر به تغییر در مدل‌سازی می‌شود و صرفاً محدود به یک درخواست نیست، طبیعتاً منجر به تغییرات بیشتری نسبت به سناریوی قبل می‌گردد. ۷۳ خط در کد هسکل در ۴۰ دقیقه و ۱۰۲ خط در کد جاوا در ۳۰ دقیقه در اثر اجرای این سناریو، تغییر یافت.

در دو تجربه سناریوی تغییر فوق، مشاهده شد که تغییر یکسان در دو پیاده‌سازی، با صرف اندک زمان بیشتر در کد هسکل و حجم تغییر بسیار کمتر نسبت به کد جاوا همراه بود. البته باید توجه داشت که کد جاوایی که مورد استفاده قرار گرفت، صرفاً شامل پیاده‌سازی منطق سامانه بود و در حالت عادی، این پیاده‌سازی شامل بخش‌های مربوط به رابط کاربری، لایه مربوط به پایگاه داده، تکنولوژی‌های مختلف و بخش‌های دیگر نیز می‌شود که حجم کد را بسیار افزایش خواهد داد. در نتیجه احتمال دارد این تغییرات در کد جاوا در صورتی که صرف پیاده‌سازی منطق نباشد، حتی از این تعداد خط تغییر هم به تغییرات بیشتری نیاز داشته باشد و تفاوت توصیف تابعی بیشتر مشخص شود. هر دو پیاده‌سازی هسکل و جاوا، پیش و پس از این تغییرات به این گزارش پیوست شده است.

سناریوهای تغییر دیگری نیز می‌توان برای مقایسه این دو پیاده‌سازی متصور بود از جمله مفهوم هم‌نیازی، مقطع (کارشناسی و تحصیلات تکمیلی) و گسترده‌تر کردن مفهوم برنامه به صورتی که تمرکزها، درس‌های عمومی، اختصاصی و سایر دسته‌بندی‌های دروس را به صورت جداگانه دربرداشته باشد و بتوان شرط فارغ التحصیلی دانشجو را با استفاده از آن بررسی کرد. در این پژوهش به دلیل محدودیت زمان، ارزیابی و مقایسه پیاده‌سازی تابعی و شیء‌گرا با دو سناریوی فوق انجام شده است و سایر سناریوها و بررسی‌های بیشتر به تحقیقات آتی سپرده می‌شود.

۵. جمع‌بندی و نتیجه‌گیری

همانطور که در بخش مقدمه و تعریف مسئله ذکر شد، هدف اصلی و انگیزه آغاز این پژوهش، دستیابی به روشی برای مدل‌سازی یک سامانه جامع اطلاعاتی بود به نحوی که بتوان در تولید آزمون‌های خودکار از آن استفاده نمود. روش‌های رسمی برای مدل‌سازی نقاط ضعف خود را داشتند و انتخاب بعدی برای امکان‌سنجی و بررسی مدل نهایی، روش‌های مدل‌سازی با زبان‌های تابعی بودند.

با توجه به اینکه زبان‌های تابعی مانند هسکل عموماً برای توسعه توصیف‌های سریع اما کامل از یک دامنه مورد استفاده قرار می‌گیرند، تجربه و نمونه‌های مشابهی که از هسکل برای توصیف یک سامانه جامع سازمانی - اطلاعاتی بهره گرفته باشند به ندرت یافت می‌شود. بنابراین صورت مسئله اصلی این پژوهش به این صورت تعریف شد که آیا به طور کل، روش‌های مدل‌سازی با استفاده از زبان‌های تابعی و به طور خاص، زبان هسکل، انتخاب مناسبی برای توصیف سامانه‌های جامع و گسترده اطلاعاتی هستند یا خیر و برای رسیدن به توصیف کاملی از منطق چنین سامانه‌ای، چه الگوها و معماری‌هایی می‌توانند کمک‌کننده و مفید باشند.

به این منظور، گام‌های مشخصی برای پیاده‌سازی یک مدل جامع طی شد از جمله طراحی مدل دامنه، پیاده‌سازی موجودیت‌ها و ماژول‌ها و روابط‌ها و افزودن معماری مبتنی بر رویداد که به تفصیل در فصل ۳ شرح داده شد. در این مسیر، انتخاب‌ها و آزمون و خطاهای متعدد منجر به شکل‌گیری تجربه‌ها، اصول و الگوهای برای مدل‌سازی‌های مشابه گردید. از رئوس نتایج این پژوهش که در قسمت‌های مختلفی از فصول به آن‌ها اشاره شد، می‌توان موارد زیر را برشمرد:

- دستیابی به مدل نهایی و جامعی از مدل دامنه و توصیف منطق سامانه انتخاب واحد با تعداد قابل قبولی از درخواست‌ها و پاسخ‌ها و امکان گسترش سامانه با هزینه زمانی پایین (با توجه به بخش ارزیابی) نشان‌دهنده این است که زبان تابعی هسکل را می‌توان برای توصیف منطق دامنه در یک سامانه اطلاعاتی معمول و متداول (با ویژگی‌های سیستم انتخابی که در بخش ۱-۲ ذکر شد)، مناسب و قابل استفاده دانست.
- در طول فرآیند توصیف و پیاده‌سازی مدل دامنه، اگر کدنویسی طبق توصیه‌های برنامه‌نویسی تمیز (پیش‌برود، خروجی نهایی توصیف، خوانایی و قابلیت نگهداری و گسترش بالایی خواهد داشت. از جمله مواردی که در این زمینه در توصیف سامانه انتخاب واحد در این پژوهش رعایت شده‌اند می‌توان به توجه به نام‌گذاری معنی‌دار توابع، پیاده‌سازی منطق با توابع کوچک با یک وظیفه، حفظ کپسوله‌سازی با ساخت ماژول‌ها و اضافه کردن معماری مبتنی بر رویداد، و عدم نوشتن کد تکراری اشاره کرد.
- نکته قابل توجه این است که در دستیابی به توصیفی کامل و ساختارمند با قابلیت خوانایی و نگهداری بالا، می‌توان از پیچیدگی‌های مدل‌های تابعی نیز پرهیز کرد. مواردی مانند منادها اگرچه در منابع و توصیف‌های تابعی در دسترس، بسیار مورد استفاده قرار می‌گیرند اما خوانایی و درک عملکرد آن‌ها مشکل است و در عین حال می‌توان با راهکارهای دیگری (مانند معماری مبتنی بر رویداد در این توصیف) به عملکرد مورد انتظار دست یافت.
- حجم توصیف نهایی سامانه با توجه به پیاده‌سازی کامل مدل دامنه و منطق هدف، معقول و متناسب به نظر می‌رسد. باید توجه داشت که توصیف تابعی سامانه تنها منطق را شامل می‌شود و اگر چنین سامانه‌ای یا هر نوع سیستم اطلاعاتی دیگری

¹ Clean Coding Practices

² Monad

در ابعاد Enterprise پیاده‌سازی شود، حجم کد زیادی مربوط به بخش‌های مختلف از جمله دسترسی به اطلاعات پایگاه داده و کدهای بهینه‌سازی این دسترسی، واقع‌نگاری‌ها، احراز هویت و دسترسی‌ها^۲ لایه نمایش و رابط کاربری، موجودیت‌هایی برای جابه‌جایی داده‌آیین لایه‌های نمایش و دامنه و بسیاری بخش‌های دیگر به سامانه افزوده خواهد شد که مدل‌سازی منطق دامنه اصلی در این میان تقریباً گم و محو می‌شود.

- از نکاتی که باید در تولید آزمون سراسری به آن توجه شود، نحوه تعیین خروجی مناسب برای هر مجموعه ورودی سیستم است. این مسئله که به مسئله پیشگوی آزمون معروف است در بسیاری از تحقیقات مورد توجه قرار گرفته است. یکی از ساده‌ترین پیشگوهای آزمون استفاده از متخصصان دامنه سیستم است ولی این مسئله با دشواری‌های متعددی همراه است. مشکل اول هزینه بالای استفاده از پیشگوی انسانی است. پیشگوی انسانی باید هر مورد آزمون را به صورت مستقل مورد بررسی قرار دهد و با در نظر گرفتن سناریوی اجرایی آن، پاسخ مناسب را برای آزمون مشخص کند. از سوی دیگر خطاهایی که پیشگوی انسانی می‌تواند انجام دهد ممکن است اعتبار کلی مجموعه آزمون را زیر سوال ببرد. از همین رو در بسیاری از تحقیقات سعی می‌شود تا از پیشگوهای خودکار برای تعیین پاسخ درست یک مجموعه آزمون استفاده شود. یکی از منابعی که می‌تواند به عنوان پیشگوی آزمون مورد استفاده قرار گیرد، مدلی است که برای تولید آزمون استفاده می‌شود. البته این نکته باید در تولید مدل نیز مورد توجه قرار گیرد زیرا روش‌های مدل‌سازی بسیاری هستند که می‌شود از آن‌ها برای تولید خودکار آزمون استفاده کرد ولی خود مدل نمی‌تواند به عنوان پیشگوی آزمون استفاده شود. توصیف نهایی به دست آمده در این پژوهش، توصیفی قابل اجرا و قابل آزمون با قابلیت استفاده برای تولید آزمون خودکار است که همچنین می‌تواند به عنوان پیشگوی آزمون نیز مورد استفاده قرار گیرد (مرجع [۴]). به عبارت دیگر می‌توانیم به طور خودکار آزمایش‌های سامانه را با استفاده از این مدل تولید کنیم و خروجی‌ها را نیز از کد هسکل دریافت کنیم.

تحقیقات آتی و کارهای آینده در ادامه این پژوهش می‌تواند حول موضوعات زیر تعریف شود:

- انجام پژوهشی کامل‌تر برای ارزیابی کیفیت توصیف هسکل از لحاظ خوانایی و انعطاف‌پذیری در برابر تغییرات
- پیاده‌سازی الگوهای دیگر طراحی مبتنی بر دامنه در هسکل به طور مثال الگوی Decorator
- پیاده‌سازی توصیف تابعی در ابزار کوک (Coq) و به کارگیری روش‌های اثبات برای درستی‌سنجی توصیف
- تولید خودکار آزمایش برای مدل هسکل توسط ابزار QuickCheck

¹ Logging

² Authentication and Authorization

³ Data Transfer Objects between layers

⁴ Test Oracle

۶. مراجع

- [1] P. Hudak and M. P. Jones, "Haskell vs. ada vs. c++ vs. awk vs.... an experiment in software prototyping productivity," *tech. rep.*, Citeseer, 1994.
- [2] Fristi, 2015, *Aggregate*, accessed 20 May 2021, <<https://gist.github.com/Fristi/7327904>>
- [3] A. Avram and F. Marinescu, "Domain Driven Design Quickly", *InfoQ*, 2006
- [4] A. Zakeriyan, R.Khosravi, H. Safari, E. Khamespanah. 2020. "Towards Automatic Test Case Generation for Industrial Software Systems Based on Functional Specifications". *FSEN Conference*, Tehran, Iran