مقدمه

امروزه در معماري نرم‌افزار الگوهاي معماري متنوعي پديد آمده‌اند كه بهبود فرآيند معماري نرم‌افزار كمك مينمايند. يكي از الگوهاي معماري رايج در معماري نرم‌افزار الگوي معماري رويداد محور [[1]](#footnote-1) مي‌باشد .در ادامه اين تحقيق ابتدا به تعريف اين معماري پرداخته مي‌شود . سپس اجزاء مختلف الگوي معماري رويداد محور مشاهده مي‌گردد . در بخش بعدي به كاربرد اين الگوي معماري پرداخته مي‌شود . سپس مزايا و معايب الگوي معماري رويداد محور توضيح داده شده است . و در بخش آخر نكاتي كه بايد درحين استفاده از اين الگوي معماري بايد مورد توجه قرار گيرد پرداخته مي‌شود .

1. فزار ‌ا

# الگوی معماری رویداد محور

## شناخت و تعریف الگوی معماری رویداد محور

برای آشنایی با الگوی معماری رویداد محور دو تعریف مختلف در ذیل این بخش آورده شده است. بخش بعد از تعاریف یک نیم نگاهی به کاربرد این الگوی معماری دارد که در بخش های بعد مفصل توضیح داده شده است .

## تعریف شماره 1

معماری رویداد محور یک الگوی طراحی نرم‌افزاری است که در آن برنامه‌های کاربردی جداشده می‌توانند به‌طور ناهمزمان رویدادها را از طریق یک واسطه رویداد (میان‌افزار مبتنی بر پیام‌رسانی مدرن) منتشر کرده و مشترک شوند. (سایت مدیوم)

## تعریف شماره 2

معماری رویداد محور یک معماری نرم افزاری و مدل برای طراحی نرم افزار است . با یک سیستم رویداد محور، ضبط، ارتباط، پردازش و تداوم رویدادها ساختار اصلی راه حل هستند.این متفاوت از مدل سنتی درخواست محور است . (سایت ردهت)

بسیاری از طراحی‌های اپلیکیشن‌های مدرن رویداد محور هستند ، مانند چارچوب‌های تعامل با مشتری که باید از داده‌های مشتری به صورت بلادرنگ استفاده کنند . اپلیکیشن های رویداد محور می توانند با تمام زبان‌های برنامه‌نویسی به وجود بیایند زیرا معماری رویداد محور یک رویکرد است نه یک زبان . معماری رویداد محور حداقل میزان دوگانگی را ممکن می‌سازد،که آن را به گزینه خوبی برای معماری‌های کاربردی مدرن و توزیع شده تبدیل می کند.

## رویداد چیست ؟

یک رویداد هر رخداد قابل توجه یا تغییر در وضعیت برای سیستم سخت افزاری یا نرم افزاری است . یک رویداد همانند اعلام رویداد نیست، که پیام یا اطلاعی بوسیله سیستم باشد، که برای اطلاع دادن به بخش دیگری از سیستم که یک رویداد صورت گرفته است ، ارسال شود .

منبع یک رویداد میتواند ناشی از ورودی‌های خروجی یا داخلی باشد . رویداد می‌تواند توسط یک کاربر به وجود آید ، برای مثال می‌تواند یک کلیک موس یا کیبورد ، یک منبع خارجی مانند سنسور خارجی ، یا از سیستم مانند لود شدن یک برنامه باشد .

معماری رویداد محور چگونه کار می‌کند؟

Event-driven architecture is made up of event producers and event consumers. An event producer detects or senses an event and represents the event as a message. It does not know the consumer of the event, or the outcome of an event.

معماری رویداد محور از تولید کنندگان رویداد و مصرف کنندگان رویداد تشکیل شده است . یک تولید کننده

رویداد یک رویداد را تشخیص و حس می‌کند و رویداد را تحت یک پیام نشان می‌دهند .

الگوی معماری رویداد محور از تولید‌کنندگان رویداد ، مصرف‌کنندگان رویداد و روتر رویداد

## چه زمان‌هايي از معماري رويداد محور استفاده مي‌شود؟

1. زيست‌بوم مصرف‌كننده غيرشفاف

مواردی که تولیدکنندگان عموماً از مصرف کنندگان بی اطلاع هستند. مورد دوم حتی ممکن است فرآیندهای زودگذری باشند که می توانند با اطلاع کوتاهی بیایند و بروند!

## مزایای معماری رویداد محور

1. قابلیت بافرکردن و تحمل خطا

رویداد‌ها ممکن است با نرخ متفاوتی نسبت تولیدشان مصرف شود و تولیید‌کنندگان نباید سرعت خود را کم کنند تا مصرف‌کنندگان عقب نمانند .

1. جداسازی تولییدکنندگان و مصرف‌کنندگان

افزودن تولییدکنندگان و مصرف‌کنندگان جدید به سیستم و تغییر در پیاده‌سازی تولییدکنندگان و مصرف‌کنندگان بسیار آسان است ، مشروط بر اینکه قرداد و طرح‌واره‌هایی که رکوردهای رویداد را محدود می‌کنند رعایت شود .

1. مقیاس پذیری زیاد

اغلب می‌توان جریان‌های رویداد را به زیرجریان‌های نامرتبط تقسیم کرد و آنها را به صورت موازی پردازش کرد. همچنین می‌توانیم تعداد مصرف‌کنندگان را برای برآورده کردن نیازهای بار، درصورت افزایش رویدادهای عقب‌افتاده ، مقیاس کنیم . پلتفرم‌هایی مانند Kafka پردازش رویدادها را به ترتیب دقیق امکان‌پذیر می‌سارند و در عین حال موازی‌سازی عظیم در سراسر جریان را فراهم می‌کند .

## معایب معماری رویداد محور

1-محدود به پردازش ناهمزمان

اگرچن معماری رویداد محور یک الگوی قوی برای جداسازی سیستم‌هاست ، اما کاربرد آن به پردازش‌های رویداد‌های ناهمزمان محدود می‌شود. معماری رویداد محور به خوبی به عنوان جایگزینی برای تعاملات درخواست –پاسخ کار ‌نمی‌کند ، جایی که باید قبل از ادامه دادن منتظر پاسخ باشد .

2-ایجاد پیچیدگی اضافی

درجایی که روش سنتی محاسبات مشتری-سرور و درخواست-پاسخ فقط دوطرف را شامل می‌شود ، پذیرش معماری رویداد محور به یک کارگزار برای میانجیگری تعاملات بین تولییدکنندگان و مصرف‌کنندگان رویداد نیاز است .

3-پوشش شکست

این یک مورد عجیب است زیرا به نظر می‌رسد برخلاف اصل سیستم‌های جداسازی است . هنگامی که سیستم های به شدت جفت می‌شوند ، یک خطا دریک سیستم به سرعت منتشر می‌شود و اغلب به طرق ناراحت کننده‌‌اي در خط مقدم توجه ما قرار مي‌گيرد . در بيشتر موارد ، اين چيزي است كه ما دوست داريم از آن اجتناب كنيم : خرابي يك جزء بايد كمترين تاثير ممكن را بر سايرين داشته باشد . آنسوي ديگر پوشش شكست اين است كه به طور ناخواسته مشكلاتي را پنهان مي‌كند كه در غير اينصورت بايد به ما توجه شود . اين با افزودن نظارت و ثبت بي‌درنگ به هر مؤلفه رويداد محور حل مي‌شود، اما اين با پيچيدگي بيشتري همراه است.

## نكاتي كه بايد هنگام استفاده از معماري رويداد محور بايد مورد توجه قرار داد؟

معماري رويداد محور يك نوشدارو نيست و مانند هر ابزار قدرتمند ديگري مستعد استفاده نادرست است . فهرست زير نبايد به عنوان معايب آشكار معماري رويداد محور خوانده شود ، بلكه بيشتر به عنوان مجموعه اي از مشكلات كه توسعه‌دهندگان و معماران بايد هنگام طراحي و اجراي سيستم‌هاي رويداد محور از آن آگاه باشند .

1-رقص پيچيده

با مولفه‌هایی که به‌طور ضعیف جفت شده‌اند، می‌توان وارد موقعیتی شد که معماری ممکن است شبیه ماشین روب گلدبورگ باشد، که در آن کل منطق کسب‌وکار به‌عنوان مجموعه‌ای از اثرات جانبی که به‌عنوان رویدادها پنهان می‌شوند پیاده‌سازی می‌شوند: یک جزء ممکن است رویدادی را ایجاد کند که باعث ایجاد یک رویداد شود. پاسخ در مؤلفه دیگری که رویداد دیگری را ایجاد می کند، مؤلفه دیگری را تحریک می کند و غیره. درک و استدلال این سبک از تعامل بین اجزا به سرعت دشوار می شود.

2- پنهان كردن دستورات به عنوان رويداد

یک رویداد تصویری خام از چیزی است که اتفاق افتاده است. نحوه رسیدگی به رویداد را تعیین نمی کند. از سوی دیگر، یک فرمان یک دستور مستقیم است که به یک جزء خاص خطاب می شود. از آنجایی که دستورات و رویدادها هر دو پیام هایی هستند، به راحتی می توان یک فرمان را به عنوان یک رویداد نادرست معرفی کرد.

3-ناشناس ماندن مصرف كنندگان

رویدادها باید ویژگی‌های مرتبط را به گونه‌ای دربر گیرند که نحوه پردازش آن رویدادها را محدود نکنند. گفتن این کار آسان تر از انجام آن است. گاهی اوقات ممکن است اطلاعات بیشتری را در اختیار داشته باشیم که در تئوری می‌تواند به یک رکورد رویداد اضافه شود، اما مشخص نیست که افزودن آن اطلاعات به رکورد مفید است یا اینکه منجر به نفخ بی‌فایده می‌شود.

1. Event Driven Architecture Pattern(EDAP) [↑](#footnote-ref-1)