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!

## سبك‌هاي مختلف الگوي معماري رويداد محور (رفرنس وب رد هت 3 ماکروسافت 1 وب سایت مدیوم 4 )

سبك هاي مختلف الگوي معماري رويداد محور براساس نحوه عملكرد خود بر دو مدل مختلف تقسيم مي‌شوند كه در ادامه به آنها پرداخته خواهد شد[1] :

1. مدل انتشار اشتراك

اين يك زير ساخت پيام‌رساني مبتني بر اشتراك در جريان ‌رويداد است [3]. در این مدل هنگامی که یک رویداد منتشر می شود، رویداد را برای هر مشترک ارسال می کند. پس از دریافت یک رویداد، نمی توان آن را دوباره پخش کرد و مشترکین جدید رویداد را نمی بینند[1].

1. مدل جريان رويداد

رویدادها در یک گزارش نوشته می شوند. رویدادها کاملاً سفارشی (در یک پارتیشن) و بادوام هستند. مشتریان در جریان مشترک نمی شوند، در عوض مشتری می تواند از هر بخشی از جریان بخواند. مشتری مسئول پیشبرد موقعیت خود در جریان است. این بدان معناست که یک مشتری می تواند در هر زمان بپیوندد و می تواند رویدادها را دوباره پخش کند[1] .

مدل جريان رويداد نيز خود بر حسب نوع جريان رويداد به سه گونه ديگر تقسيم مي‌گردد كه عبارتند از:

1. پردازش رويداد گسترده

از یک پلت فرم جریان داده، مانند Azure IOT Hub یا Apache Kafka، به عنوان خط لوله برای جذب رویدادها و تغذیه آنها به پردازشگرها استفاده کنید. پردازشگرهای جریان برای پردازش یا تبدیل جریان عمل می کنند. ممکن است چندین پردازنده جریانی برای زیرسیستم های مختلف برنامه وجود داشته باشد. این رویکرد برای حجم کاری اینترنت اشیا مناسب است [1] . . یک مثال خوب ممکن است پیوند تغییرات در یک نهاد تجاری باشد . یک مصرف کننده ممکن است این تغییرات را طبق دستور تولیید کننده اعمال کند تا یک نسخه از موجودیت را در پایگاه داده محلی خود ذخیره کند . پردازش این سوابق تغییر به طور مجزا ممکن است آن را قطع نکند، زیرا سفارش مهم است. مشتریها نیاز به پرهیز از شرایط رقابتی دارند، . مصرف کنندگان همچنین باید از شرایط مسابقه اجتناب کنند، به طوری که چندین نمونه مصرف کننده ممکن است سعی کنند به طور همزمان تغییراتی را در یک رکورد در یک پایگاه داده اعمال کنند، که منجر به ناسازگاری داده ها به دلیل به روز رسانی های خارج از دستور می شود [4] .

1. پردازش رويداد ساده (وب سايت رد هت )

پردازش رویداد ساده زمانی است که یک رویداد بلافاصله اقدامی را در مصرف کننده رویداد ایجاد می کند [3]. برای مثال، می‌توانید از توابع Azure با یک تریگر Service Bus استفاده کنید، به طوری که هر زمان که پیامی در یک موضوع Service Bus منتشر می‌شود، یک تابع اجرا می‌شود [1] .

1. پردازش رويداد پيچيده (وب سايت مديوم)

در پردازش رویداد پیچیده یک مصرف کننده با استفاده از فناوری هایی مانند Azure Stream Analytics یا Apache Storm، مجموعه ای از رویدادها را پردازش می کند و به دنبال الگوهایی در داده های رویداد می گردد[1]. يك مثال از پردازش رويداد پيچيده ممكن است كه نظارت بر يك گروه از سنسورهاي دما و دود در يك ساختمان براي پی بردن به وقوع آتش سوزی و پیشرفت آن باشد . تغییرات یک سنور دما برای نواختن زنگ هشدار کافی نیست ; با این حال، خوشه‌بندی نوسانات دما و نرخ تغییر ممکن است بینش‌های معناداری بیشتری ارائه دهد که در نهایت می‌تواند جان انسان‌ها را نجات دهد [4].

## مزایای معماری رویداد محور

1. قابلیت تحمل خطا و بافرکردن

رویداد ناهمزمان است، که استنباط می‌کند که رویدادها به ‌موقع و زمانی که اتفاق می‌افتند منتشر می‌شوند. سرویس یا مؤلفه مصرف کننده رویداد در این رویدادها مشترک می شود. بنابراین وقتی هر مصرف کننده ای از کار بیفتد، برنامه در غیاب آن به کار خود ادامه می دهد. رویدادها در کارگزار صف‌بندی می‌شوند تا زمانی که مصرف کننده از شکست بهبود یافت، بتواند رویدادهای معلق را دریافت کند [5].

1. اتصال سست یا جداسازی تولید‌کنندگان و مصرف کنندگان

سیستم‌هایی که از معماری رویداد محوراستفاده می‌کنند دارای اجزایی هستند که به‌طور سست متصل شده‌اند. این جداسازی به تفکیک منطقی تولید و مصرف رویدادها کمک می کند. تولیدکنندگان نگران نحوه مصرف رویدادها نیستند، در حالی که مصرف کنندگان نیز نگران تولید رویدادها نیستند. به دلیل این اتصال سست، اجزا یا میکروسرویس های مختلف را می توان به زبان های دیگر توسعه داد یا از فناوری های مختلف استفاده کرد. همچنین به مقیاس بندی برنامه کمک می کند. تولیدکنندگان یا مصرف‌کنندگان را می توان بر اساس الزامات بدون تغییر منطقی در سایر اجزاء اضافه یا حذف کرد [5] .

1. مقرون به صرفه

معماری رویداد محور بر اساس نشاندن پیام توسط تولید کننده در صف و دریافت آن توسط مصرف کننده انجام می شود. به این ترتیب نمونه‌برداری مداوم برای بررسی یک رویداد مورد نیاز نیست. پردازنده کمتر مورد استفاده قرار می‌گیرد و استفاده از پهنای باند شبکه کاهش می یابد [5].

1. قابلیت مقیاس‌پذیری زیاد

مقیاس پذیری یکی از جذاب ترین ویژگی های پشته بدون سرور است. با افزایش ترافیک به صورت پویا مقیاس می شود و همچنین تعداد عملکردهای همزمان را افزایش می دهد. این امکان پذیر است زیرا عملیات ها کوچکتر هستند و می توانند در هر یک از مجری های رایگان موجود بارگذاری شوند. در مقابل، این کار قبلاً دشوار بود زیرا کل برنامه باید روی حافظه بارگذاری شود [5].

5-تجربیات کاربری بلادرنگ بهتر

رابطهای برنامه نویسی کاربردی رویداد محور با حذف بسیاری از مسئولیت‌هایی که قبلاً به کاربران محول شده بود، منجر به تجربه تعاملی بهتری برای کاربر نهایی با نیازهای تعاملی مدرن می‌شوند. البته، این ممکن است پیچیدگی را در تولید کننده افزایش دهد، اما خروجی کار ارزش آن را دارد [5].

## معایب معماری رویداد محور

1-محدود به پردازش ناهمزمان

اگرچه معماری رویداد محور یک الگوی قوی برای جداسازی سیستم‌هاست ، اما کاربرد آن به پردازش‌ رویداد‌های ناهمزمان محدود می‌شود. معماری رویداد محور در جایی که آغاز کننده باید قبل از ادامه به کار باید منتظر پاسخ باشد، نمی‌تواند به خوبی به عنوان جایگزینی برای تعاملات درخواست-پاسخ کار کند [4] .

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 [4].

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 ازآن آگاه باشند [4].

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[4].

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[4].

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[4].

مراجع:

ماکروسافت 1

آمازون 2

رد هت 3

مدیوم 4

نیلل 5

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