مانیتورینگ سامانه‌های تولید و توزیع نفت و گاز در سیستم‌های کنترلی توزیع شده با به‌کارگیری الگوهای طراحی نرم‌افزار

بهناز دمهری*، رضا روانمهر، دانشگاه آزاد اسلامی، واحد تهران مرکزی

چکیده

زنجیره بالادستی تولید، توزیع و انتقال نفت و گاز گستره‌ای از فعالیت‌هاست که به‌منظور بهینه نمودن هر یک از آنها لازم است تا میلیون‌ها پارامتر به‌صورت پیوسته و مرتبط با یکدیگر بررسی و بهینه گردند. این امر در حال حاضر در شرکت‌های نفت و گاز به‌منظور بهینه نمودن شرایط تولید، توزیع و انتقال و کسب بالاترین سود اقتصادی در قالب سامانه‌های مانیتورینگ از درون‌چاهی تا تأسیسات تحویل، بررسی، پایش و نظارت، صورت می‌گیرد. از سویی، ارتباط دینامیکی مخزن با شرایط عملیاتی تأسیسات سطح‌الارضی، انتقال و توزیع و تحویل می‌تواند در بهبود مدیریت بهینه کل زنجیره، تعیین‌کننده باشد. البته سیستم‌های نظارت و کنترل که قابلیت مانیتور، محاسبه و اِعمال دستورات لازم در چنین حجم و گستره‌ای را داشته باشند، به‌دلیل سرعت و دقت عملیات، نیازمند استفاده از روش‌های کنترلی خاص و پیچیده می‌باشند که این روش‌ها در قالب سیستم‌های کنترلی توزیع شده، طبقه‌بندی می‌گردند که برای طراحی چنین سیستم‌هایی، استفاده از الگوهای طراحی توصیه می‌شود زیرا الگوهای طراحی بهترین راه‌حل برای مشکلات رایج طراحی نرم‌افزار هستند و استفاده از آنها در سیستم‌های کنترلی توزیع‌شده می‌تواند باعث بهبود قابلیت استفاده مجدد از برنامه‌ها، کاهش زمان توسعه مجدد، افزایش کارآمدی، بهبود ساختاری و کاهش پیچیدگی شود. در این مقاله، با استفاده از الگوهای طراحی دستور یک رویکرد جهت مانیتورنگ فعالیت‌های موجود در سیستم‌های توزیع شده ارائه شده است تا بتوان به بهترین شکل ممکن از وضعیت اجزای موجود در این‌گونه سیستم‌ها مطلع گردید. از آنجایی که اهداف مانیتورینگ در سیستم توزیع شده شامل تنظیم پیکربندی، نمایش شرایط محیطی، نمایش اطلاعات زیرسامانه‌ها و تولید گزارش می‌باشد، بنابراین با استفاده از آنها در زنجیره بالادستی می‌توان به تولید صیانتی نفت و گاز، جلوگیری از آسیب به مخزن و چاه‌ها، تولید نفت‌خام از لایه‌های منتخب، کاهش هزینه‌های تولید، فرآورش و ذخیره‌سازی و بهینه نمودن عملیات فرآورش اقدام نمود.

اطلاعات مقاله

تاریخ ارسال نویسنده: ۹۵/۱۱/۱۳
تاریخ ارسال به داور: ۹۵/۱۱/۱۳
تاریخ پذیرش داور: ۹۵/۱۱/۲۶

واژگان کلیدی:
سیستم‌های کنترلی توزیع شده؛ زنجیره تولید، توزیع و انتقال نفت و گاز؛ الگوهای طراحی نرم‌افزار؛ الگوی طراحی دستور؛ مانیتورینگ

مقدمه

نظارت و ثبت شرایط چاه و مخازن تحت‌الارضی نفت و گاز، اولین اقدام در شناخت بهتر شرایط رفتار دینامیکی مخازن و مدیریت بهینه تولید از آنها می‌باشد. این در حالی است که مدیریت بهینه تولید از مخازن، تحت تأثیر شرایط متغیر فرآورش، انتقال و توزیع و صادرات نفت و گاز می‌باشد و به واقع، صرفاً در صورت نظارت، کنترل و بهینه نمودن کل زنجیره می‌توان از رفتار بهینه هریک از اجزاء آن مطمئن بود. این امر نیازمند نه‌تنها سیستم‌های نظارت و مانیتورینگ محدود و تک‌بُعدی، که نیازمند سیستم‌های دینامیکی با قابلیت یکپارچه‌سازی کل زنجیره از درون چاه تا ایستگاه‌های تقلیل فشار شهری و کشتی‌ها و خطوط انتقال نفت‌خام و یا مخازن پالایشگاه‌هاست که به‌منظور مانیتورینگ این زنجیره، سیستم‌های نرم‌افزاری پیچیده به‌همراه بانک اطلاعاتی بسیار حجیم با قابلیت اطمینان بالا مورد نیاز است.

در این شرایط، سیستم‌های نرم‌افزاری روزبه‌روز پیچیده‌تر شده و فرآیند تولید آنها نیز به‌همین روند دچار پیچیدگی و صرف هزینه و زمان بیشتری می‌گردد، به‌نحوی‌که برنامه‌نویسان مجبور به درگیر شدن با جزئیات بسیاری در حین پیاده‌سازی یک نرم‌افزار می‌باشند. همین امر موجب خطاهای برنامه‌نویسی بیشتر شده و برنامه‌نویسی سیستم‌های نرم‌افزاری را به یک کار سخت و طاقت‌فرسا تبدیل کرده است. افزایش پیچیدگی در طراحی و تولید سیستم‌های نرم‌افزاری، علاوه‌بر بالا رفتن هزینه و زمان، موجب تولید سیستم‌هایی باکیفیت پایین و خطاهای زمان اجرای زیاد، می‌شود که برای غلبه بر مشکل پیچیدگی سیستم‌های نرم‌افزاری و نیز کاهش خطا در تولید کد، استفاده از الگوهای طراحی پیشنهاد می‌شود[۱].

الگوهای طراحی به‌دنبال ارائه یک راه‌حل، جهت ایجاد ارتباط میان مشکلات شناخته شده و رویکرد موفق هستند. در واقع، هر الگو بیانگر یک مسئله و مشکل (که می‌تواند بارها و بارها روی بدهد)، همراه با راه‌حل آن مسئله است[۲] و سایرین می‌توانند از این راه‌حل برای میلیون‌ها بار، بدون نیاز به پیدا کردن مجدد راه‌حل، استفاده کنند. نظریه الگوی طراحی سبب می‌گردد تا نرم‌افزار، انعطاف‌پذیرتر و توابع آن توسعه‌پذیر باشد[۲, ۳].

امروزه سیستم‌های توزیع‌شده به‌سرعت در حال گسترش به مقیاس‌های بسیار بزرگ هستند که نیاز به ارائه خدمات و قابلیت اطمینان بالایی

* نویسنده‌ی عهده‌دار مکاتبات (beh.demehri.eng@iauctb.ac.ir)

دارند. این روزها سیستم‌های توزیع‌شده می‌توانند به هزاران گره[۱] متصل شوند[۴, ۵] که درنتیجه این پیچیدگی، نیاز به برنامه‌های توزیع‌شده به‌سرعت در حال افزایش است.

سیستم توزیع شده، مجموعه‌ای از کامپیوترهای مستقل است که از دیدگاه کاربران مانند یک سیستم منسجم عمل می‌کند. این تعریف دو جنبه مهم دارد؛ اولین جنبه این است که سیستم توزیع‌شده شامل قطعاتی است که خودمختار هستند و جنبه دوم این است که کاربران برنامه یا انسان‌ها، تصور می‌کنند که آنها تنها با یک سیستم سروکار دارند[۶].

سیستم‌عامل توزیع‌شده در یک محیط شبکه‌ای اجرا می‌شود. در این سیستم قسمت‌های مختلف برنامه کاربر بدون آنکه خود او متوجه شود، می‌توانند هم‌زمان در چند کامپیوتر مجزا اجرا شده و سپس، نتایج نهایی به کامپیوتر اصلی کاربر برگردند. کاربران نباید از این موضوع آگاه شوند که برنامه آنها در کجا به اجرا درمی‌آید و یا فایل‌های آنها در کجای شبکه قرار دارد و همه این کارها باید توسط سیستم‌عامل به‌صورت خودکار انجام گیرد. به‌عبارت‌دیگر، سیستم باید از دید کاربر شفاف باشد. یکی از مزایای مهم سیستم‌های توزیع‌شده، سرعت بالای اجرای برنامه‌هاست؛ چراکه یک برنامه هم‌زمان می‌تواند از چندین کامپیوتر برای اجرا شدنش استفاده نماید. همچنین به علت توزیع شدن اطلاعات، بانک‌های اطلاعاتی حجیم می‌توانند روی چندین کامپیوتر شبکه شده، قرار گیرند و نیازی نیست تا تمام اطلاعات به یک کامپیوتر مرکزی فرستاده شود. از دیگر مزایای سیستم‌های توزیع شده، افزایش قابلیت اطمینان، کارایی و قابلیت گسترش آنهاست[۶, ۷].

در بخش دوم، تعدادی از پُرکاربردترین الگوهای طراحی سیستم‌های کنترلی توزیع‌شده را معرفی نموده و در بخش سوم، به ارائه یک رویکرد جهت مانیتورنگ فعالیت‌های موجود در سیستم‌های توزیع شده پرداخته شده تا بتوان به بهترین شکل ممکن از وضعیت اجزای موجود در سیستم توزیع شده به موقع مطلع گردید.

۱- الگوهای طراحی برای سیستم‌های توزیع شده

۱-۱- الگوی طراحی دستور[۲]

الگوی طراحی دستور یک ابزار مفید جهت رفتار با اشیاء می‌باشد. با ایجاد درخواست برای یک شیء، دستور در شیء Invoker ذخیره شده و می‌توان به تغییر یا حفظ سوابق اقدامات مختلف انجام گرفته بر روی آن شیء پرداخت[۸].

الگوی طراحی دستور اجازه می‌دهد تا درخواست‌های ارسالی به یک شیء، خود به‌عنوان یک شیء وجود داشته باشد و این بدان معناست که اگر درخواستی برای توابعی از یک شیء ارسال شود، شیء Com-mand می‌تواند آن درخواست را درون شیء جای دهد. هنگامی‌که درخواستی ارسال می‌گردد، آن دستور در شیء ذخیره می‌شود. در آینده درصورتی‌که نیاز به دسترسی به همان درخواست باشد یا برخی از متدهای آن درخواست موردنظر باشند، می‌توان از شیء درخواست به‌جای فراخوانی متد شیء به‌صورت مستقیم استفاده نمود. در واقع، یک درخواست به‌صورت یک شیء کپسوله‌سازی می‌شود. بنابراین، این امکان را فراهم می‌کند تا مشتری‌ها را با درخواست‌های متفاوت پارامتردهی کرده، درخواست‌ها را صف‌بندی یا ثبت کرده و اعمال قابل برگشت را فراهم نمود. به‌طور خلاصه می‌توان عملکرد الگوی طراحی دستور را کپسوله‌سازی یک درخواست برای یک شیء و درنتیجه پارامتری نمودن مشتریان[۳] با درخواست‌های متفاوت، صف‌بندی یا ثبت[۴] نمودن درخواست‌ها و حمایت از عملیات لغو، دانست[۸].

یکی از اهداف مهم الگوی دستور این است که رابط کاربر از اعمالی که باید انجام دهد، مجزا گردد. به بیانی دیگر، این اشیاء برنامه باید کاملاً از یکدیگر جدا شده و نباید بدانند که سایر اشیاء چگونه کار می‌کنند[۹]. از این الگو می‌توان در نظارت و کنترل سیستم‌های مجزا و در عین حال اِعمال دستورات بالا به پایین استفاده نمود.

برخی کاربردهای الگوی دستور عبارت است از[۱۰]:

- ایجاد قابلیت‌های Undo و Redo

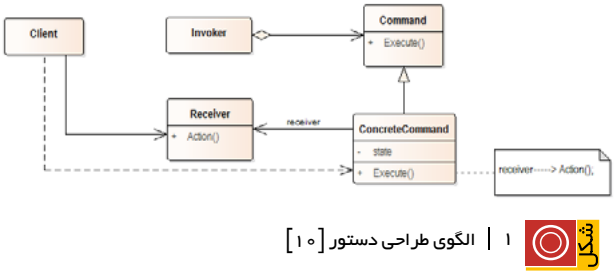
می‌توان Commandهای اجرا شده را در یک پشته[۵] قرار داد و در صورت لزوم و به درخواست کاربر Undo نمود.

- در نظر گرفتن یک کلاس جداگانه به‌عنوان گیرنده تا درخواست از عمل جدا گردد.
- ایجاد صف درخواست برای اجرای در زمان دیرتر.

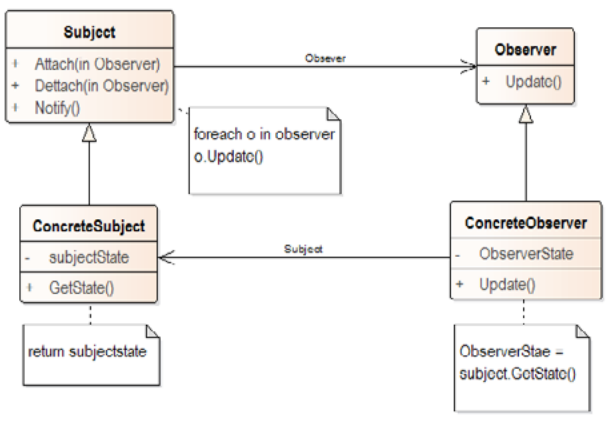
۱-۲- الگوی طراحی ناظر[۶]

یک رابطه را زمانی یک به چند بین اشیاء[۷] می‌نامند که درصورت تغییر آن شیء، سایر اشیاء مرتبط با آن بلافاصله متوجه شده و وضعیت خود را به‌روزرسانی کنند. در این الگو، مجموعه اشیاء وابسته را Observer و شیئی را که دیگر اشیاء به آن وابسته هستند، Subject می‌نامند[۱۱].

الگوی ناظر به این موضوع خاص که چگونه می‌توان مجموعه‌ای از مشتریان را به‌موقع از تغییر یک مقدار، آگاه نمود، اشاره دارد؛



شکل ۱ | الگوی طراحی دستور [۱۰]



شکل ۲ | الگوی طراحی ناظر [۸]

به‌خصوص زمانی‌که فرایند اطلاع‌رسانی به‌صورت نسبتاً طولانی تکرار شونده باشد. راه‌حل ارائه شده توسط الگوی ناظر این است که مشتریان، حالت Subscriber را نسبت به server داشته باشند تا از ارزش سؤال با توجه به برخی سیاست‌ها آگاهی یابند. این روش باعث می‌شود تا تلاش محاسباتی برای اطلاع‌رسانی مشتریان و گذرگاه ارتباطی سراسری و به‌حداقل رساندن پهنای باند موردنیاز، برای اطلاع‌رسانی مشتریان مناسب به حداقل برسد[۱۲].

از مزایای الگوی طراحی ناظر این است که باعث کم کردن وابستگی Observer و Subject به یکدیگر می‌شود و در اصطلاح این دو موجودیت در برنامه Loose couple خواهند بود[۱۱].

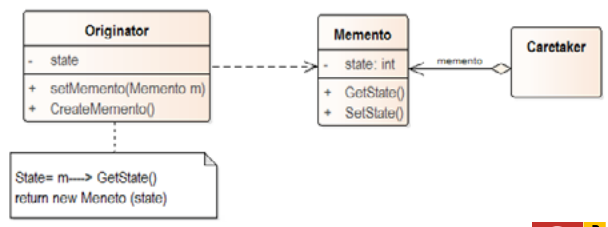
برخی کاربردهای الگوی ناظر عبارت است از[۱۲]:

- هنگامی‌که یک شیء می‌خواهد اطلاعاتش را منتشر سازد و همچنین بسیاری از اشیاء نیاز خواهند داشت تا آن اطلاعات را دریافت کنند.
- استفاده در برنامه‌های کاربردی نظارتی[۸] و مانیتورینگ شبکه‌های تولید، انتقال و توزیع در سطح وسیع که نمونه آن را می‌توان در اتاق‌های مانیتورینگ صنعت نفت دید.

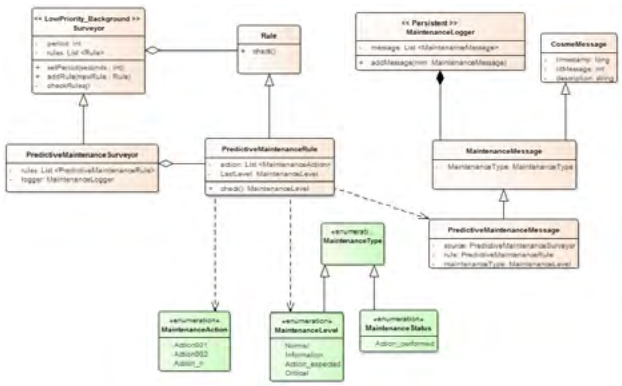
۱-۳- الگوی طراحی Memento

الگوی Memento به ذخیره‌سازی موقت حالت موجود اشیاء و سپس بازیابی آن در صورت نیاز می‌پردازد.

همان‌طور که اشاره شد، این الگو راه‌حلی جهتِ نیاز به ذخیره و بازیابی اطلاعات یک شیء را فراهم می‌کند. این شیء با نام Originator در این الگو مشخص می‌گردد. چگونگی ذخیره‌سازی وضعیت شیء موجود بدین‌گونه است که زمانی که برنامه Client ذخیره‌سازی را از شیء Originator درخواست می‌نماید، این شیء (Originator) تمام



شکل ۳ | الگوی طراحی Memento [۸]



شکل ۴ | الگوی طراحی ارزیاب پیش‌بینی‌کننده تعمیر و نگهداری [۱۴]

صفاتی را که جهت بازیابی وضعیتش موردنیاز است در یک شیء دیگر به نام Memento قرار می‌دهد و آن را به Client ارسال می‌نماید. در واقع، بنا به نمودار فوق، مقدار صفت state از شیء Originator، در مقدار صفت state شیء Memento قرار می‌گیرد. بنابراین، نیاز کنونی آن است تا اشیاء از نوع Memento را مدیریت و نگهداری نماییم. جهت رسیدن به این هدف از کلاسی به نام Caretaker استفاده می‌گردد. زمانی‌که یک شیء Memento ایجاد می‌گردد، آن شیء به مجموعه اشیاء Caretaker اضافه می‌شود. زمانی‌که یک عمل Undo انجام پذیرد، شیء Caretaker با یک شیء دیگر (Client) همکاری نموده تا یک شیء Memento انتخاب گردد. بعد از انتخاب شئ Memento، آن شیء متد SetMemento شیء Originator را فراخوانی می‌نماید تا حالت انتخاب شده را بازیابی نماید[۱۳].

برخی کاربردهای الگوی Memento عبارت است از [۹]:

- Undo/Redo کردن یک مشخصه[۹].
- ساخت تصاویر لحظه‌ای از یک وضعیت برای یک شیء و یا کل زنجیره.
- اجازه داده شود تا برخی از اطلاعات در یک شیء، توسط شیء دیگری در دسترس باشد.

۱-۴- الگوی طراحی ارزیاب پیش‌بینی‌کننده تعمیر و نگهداری[۱۰]

نرم‌افزار موجود در برنامه‌های کنترلی مدرن، به‌طور فزاینده در حال پیچیده‌تر شدن و درنتیجه نیازمند توانمندسازی جهت مسئولیت‌های جدیدتر و پیچیده‌تر است. این الگوی طراحی جهت مدیریت[۱۱] تعمیر و نگهداری پیش‌گیرانه[۱۲]، پیشنهاد شده است. این الگو در مورد ابزار و ماشین‌آلاتی که شامل بخش‌هایی هستند که از زوال تدریجی رنج می‌برند، استفاده می‌شود. الگوی Predictive Maintenance Survey-or به شناسایی ابزار فوق می‌پردازد[۱۴]. به‌این‌ترتیب، اقدامات تعمیر و نگهداری پیش‌گیرانه را می‌توان برای پایش وضعیت تجهیز و ارایه برنامه پیش‌گیرانه تعمیرات و نگهداری و جلوگیری از بُروز آسیب‌های ناخواسته به تجهیزات و همچنین عدم انجام تعمیرات غیرضروری برنامه‌ریزی نمود. این الگوی طراحی در نرم‌افزارهای کنترلی جهت موارد یاد شده استفاده می‌شود.

یکی از کاربردهای الگوی ارزیاب پیش‌بینی‌کننده تعمیر و نگهداری عبارت است از[۱۴]:

- استفاده در نرم‌افزارهای تعمیر و نگهداری

۲- الگوی طراحی ارائه شده جهت مانیتورینگ

با توجه به اینکه سیستم‌های توزیع شده از زیرسامانه‌های مختلفی تشکیل شده‌اند، نظارت و سرکشی به زیرسامانه‌ها جهت بررسی حوادث و رخدادهایی که در آنها اتفاق افتاده، امری ضروری است تا بتوان در سامانه اصلی، تصمیمات صحیح، سریع و به موقع را لحاظ نمود. زیرسامانه نظارت (مانیتورینگ)، بر روی تمام اشیاء داده‌ای[۱۳] که در حال انتقال در زیرسامانه‌های نرم‌افزاری می‌باشد، نظارت داشته و انتقال داده و همچنین اطلاعات محیطی را از پایگاه داده، دریافت می‌نماید. همچنین ممکن است به تنظیم برخی از

پیکربندی‌ها[۱۴] به‌منظور تغییر فرکانس نظارت بپردازد.

۲-۱- فعالیت‌ها

برای زیرسامانه مانیتورینگ، فعالیت‌ها به شرح زیر در نظر گرفته و اعمال می‌شوند:

تنظیم پیکربندی مانیتورینگ: یک پانل برای نظارت بر زیرسامانه‌ها در صفحه کنترل، وجود دارد. کاربر می‌تواند از طریق آن پانل به تنظیم برخی از پارامترها مانند فرکانس نظارت[۱۵] و غیره برای نظارت بر زیرسامانه‌ها بپردازد. این پارامترها در پایگاه داده ذخیره و بلافاصله در سخت‌افزار و یا زیرسامانه‌های نرم‌افزاری اعمال خواهد شد.

نمایش نتیجه مانیتورینگ محیط: این بخش آخرین شرایط محیطی را از پایگاه داده بازیابی و به کاربر ارایه می‌کند.

نمایش زیرسامانه مانیتورینگ: کاربر می‌تواند زیرسامانه موردنظر خود را انتخاب و بر اطلاعات آن نظارت نماید. این اطلاعات از سخت‌افزار و زیرسامانه‌های نرم‌افزاری دریافت و در پانل نشان داده می‌شود.

تولید گزارش مانیتورینگ: برای کاربر این امکان وجود دارد که از سامانه نظارت گزارش درخواست می‌کند. کاربر، پارامترهای مربوط به گزارش را مشخص نموده و سپس، سامانه مانیتورینگ، اطلاعات مربوطه را از پایگاه داده دریافت و به کاربر نشان می‌دهد.

۲-۲- فعالیت‌ها

برای طراحی این زیرسامانه از الگوی دستور(که در ۱-۱ توضیح داده شد) استفاده شده است زیرا به‌صورت دستوری به انجام دریافت یا ارسال اطلاعات می‌پردازد. به‌طورکلی در زیرسامانه نظارت، Client User Interface می‌باشد. زیرا در UI است که پانل‌ها مستقربوده و از آنجا می‌توان به تمام زیرسامانه‌ها دسترسی داشت و آنها را مدیریت نمود. کلاس Invoker نیز به‌عنوان یک مؤلفه رابط بین دستورات (ConcreteCommand) و دریافت‌کننده (Receiver) عمل می‌کند. به‌طورکلی وظیفه‌اش مدیریت اجرای Command ها می‌باشد. کلاس Command نیز رابطی است که عملیات قابل اجرا را مشخص می‌نماید. از آنجایی که این سه مورد ثابت هستند، جهت جلوگیری از تکرار مکررات در مراحل بعدی به بررسی ConcreteCommand و Receiver در هر مورد پرداخته خواهد شد.

Receiver: یک مؤلفه است که به‌محض دریافت درخواست، اجرا می‌شود(به انجام عملیات مربوطه می‌پردازد) و جزئیات پیاده‌سازی اجرای Command در این کلاس پیاده شده است.

ConcreteCommand: وظیفه گسترش رابط کاربری و پیاده‌سازی متد Execute با استناد[۱۶] بر عملیات مربوط به Receiver می‌باشد. در واقع، رابطی میان Receiver و یک فعالیت[۱۷] می‌باشد.

تنظیم پیکربندی مانیتورینگ

کلاس MS_Monitoring: Monitoring به‌عنوان کلاس Concrete-Command در نظر گرفته شده است. زیرا در این کلاس دستورات مربوط به تنظیم برخی از پارامترها، برای نظارت بر زیرسامانه‌ها وجود دارد. همچنین کلاس‌های Subsystem:ISoftwareSubsystem و Sub-system: IComponent و DB_ManageRequest چون دریافت‌کننده دستور هستند، به‌عنوان Reciever در نظر گرفته شده‌اند.

زیرا DB_ManageRequest، پایگاه داده است و پارامترهای مربوط به نظارت بر زیرسامانه‌ها در پایگاه داده ذخیره خواهند شد.

Subsystem:ISoftwareSubsystem در واقع زیرسامانه‌های نرم‌افزاری و سخت‌افزار هستند که پارامترها بر روی آنها اعمال خواهند شد.

نمایش نتیجه مانیتورینگ محیط

کلاس MS_Monitoring: Monitoring به‌عنوان کلاس Concrete-Command در نظر گرفته شده است. زیرا در این کلاس است که دستورات مربوط به دریافت شرایط محیطی، وجود دارد. همچنین کلاس DB_ManageRequest (پایگاه داده) چون دریافت‌کننده دستور هستند و اطلاعات محیطی را باید از آن بازیابی نمود، به‌عنوان Reciever در نظر گرفته شده‌اند.

نمایش زیرسامانه مانیتورینگ

کلاس MS_Monitoring: Monitoring به‌عنوان کلاس Concrete-Command در نظر گرفته شده است. زیرا در این کلاس است که دستورات مربوط به انتخاب زیرسامانه موردنظر و درخواست برای نظارت بر آن، وجود دارد.

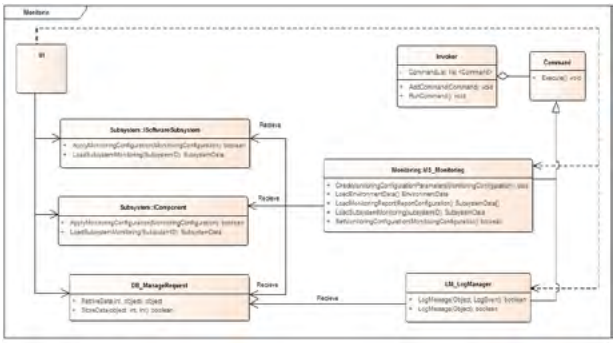
همچنین کلاس‌های ISoftwareSubsystem: Subsystem و Subsystem: IComponent چون دریافت‌کننده دستور هستند و اطلاعات مربوطه از Comemenent (سخت‌افزار) و زیرسامانه‌های نرم‌افزاری دریافت می‌شود، به‌عنوان Reciever در نظر گرفته شده‌اند.

تولید گزارش مانیتورینگ

کلاس MS_Monitoring: Monitoring به‌عنوان کلاس Concrete-Command در نظر گرفته شده است. زیرا در این کلاس است که دستورات مربوط به درخواست گزارش، وجود دارد. همچنین کلاس DB_ManageRequest (پایگاه داده) چون دریافت‌کننده دستور هستند و اطلاعات موردنیاز را باید از آن بازیابی نمود، به‌عنوان Reciever در نظر گرفته شده‌اند. در نهایت نیز به ثبت وقایع در پایگاه داده پرداخته شده است. در شکل-۵ طراحی زیرسامانه نظارت بر اساس الگوهای طراحی نمایش داده شده است.

نتیجه‌گیری

به‌دلیل گستردگی صنایع و پیچیده شدن فرآیندهای زنجیره



شکل ۵ | طراحی زیرسامانه نظارت

تولید، توزیع و انتقال نفت و گاز و لزوم بهینه نمودن هزینه‌ها و بهبود فرآیندهای عملیاتی و دستیابی به تولید با بهینه‌ترین پارامترهای فنی- اقتصادی، ضرورت دارد تا سیستم‌های نرم‌افزاری مانیتورینگ و کنترلی نیز از پیچیدگی و گستردگی بیشتری برخوردار شود که همین امر فرآیند تولید آنها را متعاقباً دچار پیچیدگی بیشتری می‌کند و موجب کاهش کیفیت سیستم و افزایش هزینه و پیچیدگی در طراحی خواهد شد. از این رو به‌منظور مانیتورینگ و نظارت برخط بر کل زنجیره و افزایش سرعت و دقت در تعیین و اعمال پارامترهای تصمیم‌گیری، افزایش کارامدی و بهبود ساختار و کاهش پیچیدگی این سیستم‌ها، می‌توان از الگوهای طراحی که یک مکانیسم برای بیان تجربه در طراحی و ارائه یک راه‌حل مناسب، توسط افراد خبره است، سود برد. در واقع، این الگوها راهبردهایی هستند که می‌توانند در هسته طراحی گنجانده شوند تا باعث افزایش قابلیت انعطاف و نگهداری بالای سیستم‌ها شوند. الگوهای طراحی که براساس نمونه‌های موفق انتخاب می‌شوند، به‌کمک کنترل سیستم‌های توزیع شده آمده‌اند تا به‌عنوان نمونه بتوان اطلاعات موردنیاز را مانیتور و براساس شرایط و رفتار کل زنجیره، وضعیت هر پارمتر و شیء را بهینه کرد.

در این مقاله با معرفی چند الگوی طراحی توزیع شده، به ارائه یک الگوی طراحی جهت مانیتورینگ در این‌گونه سیستم‌ها پرداخته شده است تا به افزایش قابلیت استفاده مجدد از برنامه‌ها، کاهش زمان توسعه مجدد و مقیاس‌پذیر بودن آن کمک نماید. با استفاده از این الگوی طراحی در سیستم‌های مانیتورینگ صنعت نفت و گاز از درون چاه‌های نفت و گاز تا پایانه‌های بارگیری نفت‌خام، مخازن ذخیره پالایشگاه‌ها و ایستگاه‌های توزیع گاز را می‌توان نظارت و شرایط محیطی همچون دبی نفت، دبی آب، دبی گاز، دمای سیال و فشار جریانی هر چاه و همچنین شرایط عملیاتی (دما، فشار و دبی) واحدهای فرآورش، شرایط سیستم‌های انتقال و توزیع و مبادی صادراتی و مخازن ذخیره‌سازی را مانیتور نمود. درنهایت، براساس شاخص‌های فنی و اقتصادی، ارتباط فی‌مابین کلیه حلقه‌های زنجیره تولید را تنظیم و بهینه‌ترین رفتار را در کل زنجیره پیاده نمود.

پانویس‌ها

1. Node
2. Command
3. Clients
4. Log
5. Stack
6. Observer
7. Object
8. Monitoring application
9. Features
10. Predictive Maintenance Surveyor
11. Handle
12. Predictive
13. Data objects
14. Configurations
15. Frequency of monitoring
16. Invoking
17. Action

منابع

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

[2] M. J. Pont, "Control system design using real-time design patterns," International Conference on Control(UKACC), pp. 1078-1083, 1998.

[3] H. Bibin, L. Qionghui, W. Qiankun, X. Guohui, K. Weizheng, W. Xiaolu, et al., "Application of design patterns in power system transient simulator," IEEE International Conference in Computer Science and Automation Engineering (CSAE), pp. 465-469, 2012.

[4] S. Bagchi, "Self-adaptive and reconfigurable distributed computing systems," Applied Soft Computing, vol. 12, pp. 3023-3033, 2012.

[5] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, et al., "Design patterns from biology for distributed computing," ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 1, pp. 26-66, 2006.

[6] A. S. Tanenbaum and M. Van Steen, Distributed systems: Prentice-Hall, pp.1-40 2007.

[7] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, Software architecture: foundations, theory, and practice: Wiley Publishing, 2009.

[8] J. Vlissides, R. Helm, R. Johnson, and E. Gamma, "Design patterns: Elements of reusable object-oriented software," Reading: Addison-Wesley, vol. 49, pp.1-120, 1995.

[9] D. C. Schmidt and C. O'Ryan, "Patterns and performance of distributed real-time and embedded publisher/subscriber architectures," Journal of Systems and Software, vol. 66, pp. 213-223, 2003.

[10] B. James and L. Lalonde, "Design Patterns," in Pro XAML with C#, ed: Springer, pp. 37-55, 2015.

[11] E. Freeman, E. Robson, B. Bates, and K. Sierra, Head first design patterns: " O'Reilly Media, Inc.", 2004.

[12] B. P. Douglass, Real-time design patterns: robust scalable architecture for real-time systems vol. 1: Addison-Wesley Professional, 2003.

[13] V. Sarcar, "Memento Patterns," in Java Design Patterns, ed: Springer, pp. 77-81, 2016.

[14] F. Serna, C. Catalán, A. Blesa, J. M. Colom, and J. M. Rams, ""Predictive maintenance surveyor" design pattern for machine tools control software applications," 16th Conference on Emerging Technologies & Factory Automation (ETFA), pp. 1-7, 2011.