طراحی و پیاده سازی یک موتور گردش فرایند

شهریور ، مهر، و آبان 95

این مستندات در دو بخش آماده شده است:

* بخش اول به مشخصات[[1]](#footnote-1) یک ورک فلوی سفارشی (مولفه های ورک فلو) می پردازد. به علاوه چندین سناریو که این ورک فلوی سفارشی قادر به پشتیبانی از آنها می باشد، بررسی شده است.
* بخش دوم یک طراحی مرجع برای پیاده سازی[[2]](#footnote-2) موتور این ورک فلوی سفارشی ارایه می کند. در پیاده سازی اینترفیس های این موتور ورک فلو زبان برنامه نویسی خاصی درنظر گرفته نشده است اما زبان های برنامه نویسی شی گرا نظیر C# ، C++ و Java گزینه های بسیار مناسبی هستند.

توجه: در حال حاضر (و به صورت موقت) به منظور شفاف سازی نام ورک فلوی سفارشی خود را star-workflow نام گذاری می کنیم تا از استاندارد BPMN2 قابل تفکیک باشد.

بخش اول : Specification یک ورک فلوی سفارشی

در طراحی star-workflow سعی شده است تا حد زیادی از اسپک BPMN2 اقتباس شود. دلایل این تصمیم عبارتند از:

1. BPMN2 یک مرجع استاندارد و کامل برای پیاده سازی
2. امکان تبدیل star-workflow به فرایندهای BPMN2 در آینده امکان پذیر باشد.

**چرا از BPMN2 استفاده نکنیم؟**

با وجود این که BPMN2 یک مرجع استاندارد برای پیاده سازی ورک فلوی فرایندها بوده و پیاده سازی های کاملی از آن موجود است چه دلیلی دارد که یک پیاده سازی سفارشی از ورک فلو داشته باشیم که تا حد زیادی هم شبیه به BPMN2 است؟

هدف ما پیاده سازی یک ورک فلوی ساده است که استفاده از آن برای توسعه دهندگان آسان[[3]](#footnote-3) باشد. در عین این که نمی خواهیم امکانات لازم و اساسی یک موتور گردش فرایند واقعی را از دست بدهیم. هم استاندارد BPMN2 و هم پیاده سازی های آن نظر بیشتری به طراحان و تحلیلگران داشته و استفاده از آنها برای توسعه دهندگان بسیار دشوار است. به نظر من این فریم ورک ها برای پروژه های کوچک و متوسط اصلا مناسب نیستند. این فریم ورک ها منحنی یادگیری با شیب تند داشته و یادگیری استفاده از آنها مستلزم صرف میزان زیادی زمان و انرژی است. به علاوه این که استفاده از آنها نیز نیاز به سطح پیشرفته ای از دانش و تجربه کدنویسی دارد. گذشته از انعطاف و امکانات بسیاری که این فریم ورک ها در اختیار می گذارند اما باز هم برنامه نویس را از نوشتن کد (گاها بسیار زیادو بسیار پیچیده) بی نیاز نمی کنند. به عقیده من "انعطاف و قدرت عمل این فریم ورک ها با پیچیدگی ها آنها مبادله شده است".

از طرفی ما سعی خواهیم کرد تا حد ممکن به استاندارد BPMN2 نزدیک باشیم تا امکان تبدیل دیاگرام ها star-workflow به BPMN2 برای سازمان ها ساده باشد.

**اهداف**

* ایجاد یک فریم ورک کوچک برای مدیریت چرخه های کاری که استفاده از آن برای برنامه نویسان ساده باشد و به صورت پیش فرض سرویس های مورد انتظار یک موتور گردش کار را فراهم آورد.
* یک زیرمجموعه[[4]](#footnote-4) از BPMN2 که صرفا چیزهای غیرپیچیده و بیشتر موردنیاز (ضروری) را داشته باشد.
* با امکان ارایه خدمات بر روی API و پیاده سازی ورک فلو بتوان پروژه های تجاری تعریف کرد. مثلا خدمات مشاوره یا پیاده سازی امکانات خاص و غیره
* امکان مجتمع شدن ورک فلو در پروژه های مختلف و استفاده از آن در کنار کتابخانه های utility دیگر به راحتی امکان پذیر باشد. مثلا شما نمی توانید به راحتی jBMP را به کتابخانه های دیگر در پروژه مجتمع کنید به دلیل عدم سازگاری.
* پروژه اوپن سورس بوده و مبتنی بر یکی از لایسنس های سازگار با پروژه های تجاری[[5]](#footnote-5) ارایه می شود. مثلا لایسنس BSDیا Apache که هم امکان ایجاد یک community حول پروژه و هم استفاده از آن در پروژه های تجاری و غیر اوپن سورس فراهم شود.

مولفه های گردش کار[[6]](#footnote-6)

مولفه های star-workflow بسیار شبیه به نودهای BPMN2 می باشند با این تفاوت که در این جا صرفا با تعداد بسیار محدودتری از المان ها سروکار داریم که در نهایت سادگی طراحی و پیاده سازی را به همراه خواهند داشت :

1. نود شروع
2. نود پایان
3. نود وظایف سرویس[[7]](#footnote-7)
4. نود وظایف انسانی[[8]](#footnote-8)
5. انواع نود های گیت وی برای فورک کردن و تصمیم گیری در فرایند
6. نود اتصال دهنده[[9]](#footnote-9)

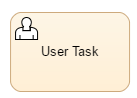
نودهای شروع و پایان



انواع گیت وی



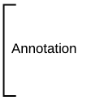
انواع تسک ها



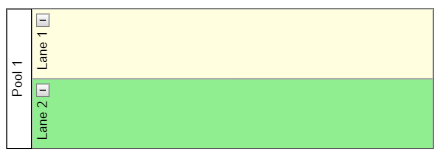
اتصال یا کانکشن



کامنت: برای نوشتن توضیحات و حاشیه نویسی درون دیاگرام



جداسازی منطقی درون دیاگرام



Pool ها و lane ها صرفا برای پارتیشن بندی منطقی درون دیاگرام ها به کار می روند و تاثیری در نحوه اجرای فرایند ندارند. مثلا lane ها می توانند دپارتمان های مختلف یک سازمان بزرگ و یا سازمان های بیرونی باشند.

مقایسه نوتیشن star-workflow با BPMN2

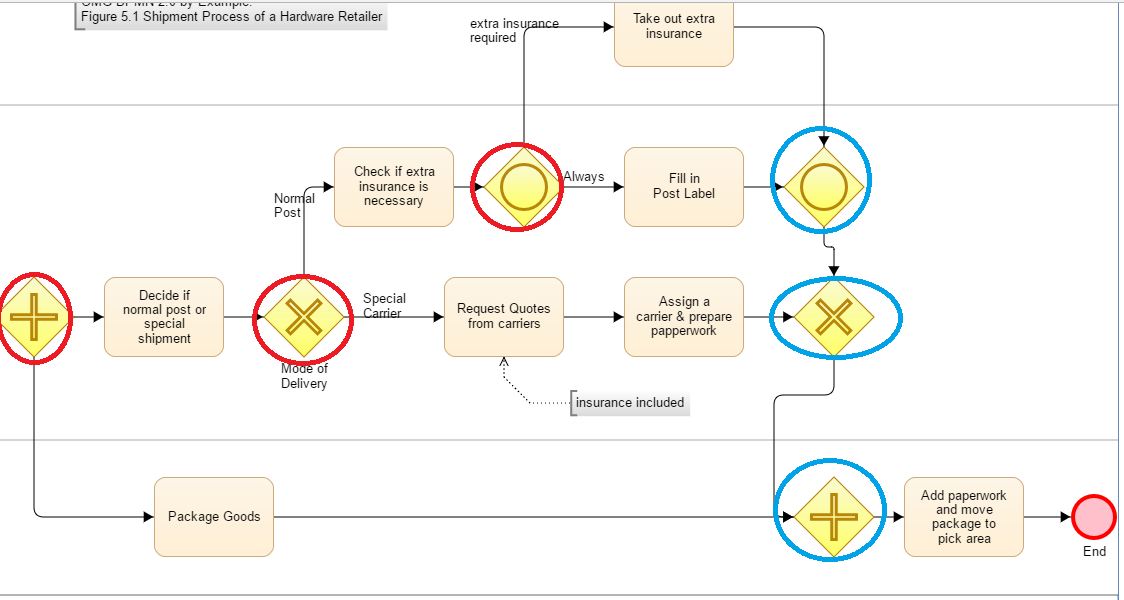
در ادامه به یکسری تفاوت ها بین قواعد دیاگرام BPMN2 و star-workflow می پردازیم:

1. **گیت وی ها**

ما صرفا برای فورک کردن فرایند از نودهای گیت وی استفاده می کنیم اما برخلاف BPMN2 از گیت وی ها برای join استفاده نمی کنیم بلکه صرفا کانکشن ها را به نودهای مقصد متصل می کنیم (نود های مبدا نودهایی هستند که اتصال از آنها شروع شده است).

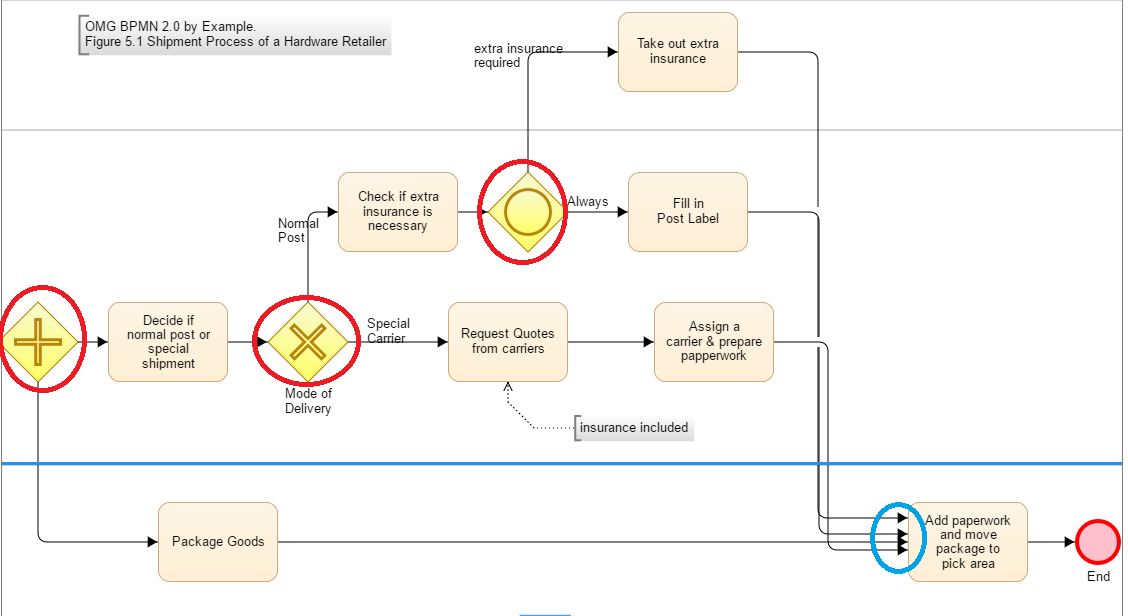
یعنی در star-workflow صرفا گیت وی های واگرا[[10]](#footnote-10) داریم به نظر من گیت وی های همگرا[[11]](#footnote-11) خوانایی دیاگرام را کاهش می دهند و باعث سردرگمی می شوند.

مثالی از نحوه استفاده گیت وی ها در BPMN2



گیت وی های مشخص شده با دایره قرمز واگرا و آنهایی که با رنگ آبی مشخص شده اند همگرا می باشند. در BPMN2 به ازای گیت وی های واگرا باید گیت وی همگرا از همان نوع داشته باشیم.

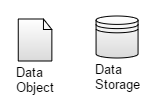
همین سناریو را در star-workflow بدین صورت بیان می کنیم



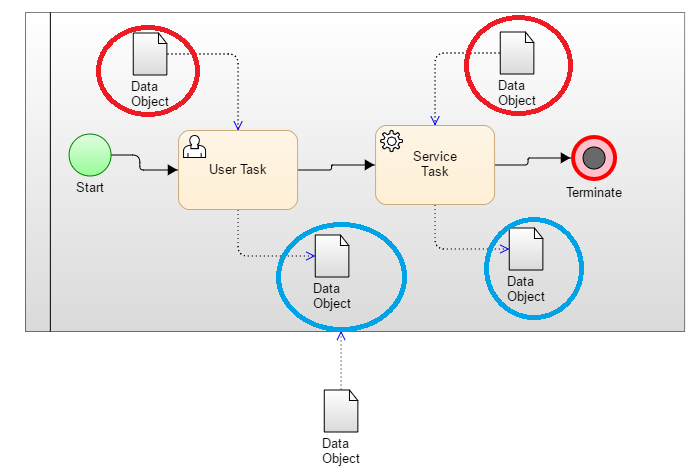
همان طور که ملاحضه می کنیم ما صرفا گیت وی های واگرا را درنظر می گیریم همانند نوتیشن BPMN2 (با رنگ قرمز مشخص شده اند) اما از گیت وی های همگرا استفاده نمی کنیم که باعث فهم ساده تر دیاگرام می شوند.

1. **نودهای دیتا**

بر خلاف BPMN2 ما از انواع نودهای دیتا اعم از ورودی-خروجی، Data Storeو غیره پشتیبانی نمی کنیم اما رویکرد مشابه و بسیار ساده تری داریم.



مثالی ازاستفاده از نودهای دیتا در BPMN2



دیتای ورودی با رنگ قرمز و خروجی با رنگ آبی مشخص شده است همچنین کل ورک فلو نیز می تواند نود (های) دیتا داشته باشد.

نکته حايز توجه این است که نودهای دیتا در دیاگرام بیشتر منطقی می باشند و اتصال آنها با سایر نودها با رنگ خط چین مشخص شده است.

مثلا یک سرویس تسک می تواند یکسری پروپرتی به عنوان ورودی بپذیرد و یکسری پروپروتی های دیگر را در خروجی خود بدهد که برای آن تسک در دیتابیس فرایند ذخیره می شود. این روش انعطاف بسیار زیادی را در بر دارد اما باعث افزایش پیچیدگی می شود. مثلا هر تسک می تواند پروپرتی های ورودی - خروجی خود را داشته باشد.

نودهای Data Storage نیز همانند نودهای دیتا منطقی (نمایشی) هستند و به معنای آن است یک نود (مثلا سرویس تسک) با یک دیتابیس در ارتباط است. طبیعی است که پیاد سازی این ارتباط صرفا در کد معنا می یابد و وجود Data Storage در دیاگرام صرفا نمایشی است.

**روش star-workflow برای پشتیبانی از نودهای دیتا**

در star-workflow به فرایند و تسک ها دیتا تایپ اختصاص می یابد. دیتای فرایند شامل پروپرتی های تعریف شده برای فرایند است که از تمامی اکشن های تسک ها و نودهای دیگر (گیت وی) قابل دسترسی است. دیتای تسک ها نیز دارای یک تایپ مشخص است که از اکشن های تسک ها قابل دسترسی است با این تفاوت که این دیتا تایپ از قبل مشخص بوده و به ازای هر تسک متفاوت نیست و همچنین دارای دو نوع input و output نیست.

در بخش طراحی و پیاده سازی فرایند در مورد این دیتا تایپ ها بیشتر صحبت خواهیم کرد.

1. **نودهای رویداد**

ما نودهای رویداد را به صورت صریح پشتیبانی نمی کنیم اما در عوض از یک روش بسیار ساده تر و ملموس تر استفاده می کنیم.

**رویدادها[[12]](#footnote-12)**

در ابتدا سعی شده بود که از پشتیبانی رویداد ها در ورک فلوی سفارشی صرفنظر شود اما با توجه به این که رویدادها جز المان های اساسی برخی از فرایندها بوده و امکان دور زدن آنها در بعضی موارد ممکن نیست، سعی بر آن شد که یک مدل غیرپیچیده برای پشتیبانی از رویدادها درنظر گرفته شود که فاقد پیچیدگی ها رویدادهای BPMN2 باشد.

قبل از تشریح روش خودمان برای پیاده سازی رویدادها ابتدا اجازه دهید به این موضوع بپردازیم که رویدادها در فرایندهای مبتنی بر BPMN2 چگونه عمل می کنند؟

در BPMN2 رویدادها از یک منظر به 3 دسته starting - intermediate- ending و از منظری دیگر به دو دسته throwing و catching تقسیم بندی می شوند که در ادامه توضیح خواهیم داد. در BPMN2 تقریبا تمامی عناصر یک فرایند (البته به جز اتصالات) قابلیت تبدیل شدن به نود رویداد را دارند.

در BPMN2 رویدادها با دایره نشان داده می شوند. مثلا نود شروع و پایان جز رویدادها هستند. بسته به اینکه یک نود رویداد در ابتدا، میانه یا انتهای فرایند به کار رفته باشد starting - intermediate یا ending خوانده می شود. اگر نود رویداد یک event را پرتاب کند throwing و اگر آن را بگیرد catching خوانده می شود.

تسک ها نیز می توانند یک رویداد را بگیرند و یا حتی آن را پرتاب کنند.

در پشتیبانی ما از رویدادها یک نود جداگانه برای رویداد درنظر گرفته نمی شود. یک تسک یا گیت وی می توانند رویدادگرا باشند و یک رویداد را انتشار دهند. زمانی که موتور ورک فلو به این نودها برسد منتظر می ماند تا از درون برنامه (کد برنامه نویس) رویداد مورد نظر منتشر شود، سپس فرایند ادامه می یابد. بنابراین هیچ نود اضافی برای catch کردن رویدادها نیز نیاز نیست.

ولی حتی از منظر BPMN2 هم رویدادها در بسیاری از موارد صرفا منطقی بوده و موتور فرایند قادر به تفکیک آنها نیست. مثلا به هر رویداد باید یک id اختصاص داده شود و برنامه نویس از درون کد و توسط API ورک فلوآن رویداد را فایر کند تا موتور فرایند آن را دریافت کرده و فرایند ادامه یابد. مثلا اگر درون فرایند دو رویداد دریافت نامه و منقضی شدن تایمر داشته باشیم از منظر موتور BPMN2 این رویدادها صرفا جنبه نشانه گذاری داشته و در نهایت به عهده برنامه نویس است که این رویدادها را در کد پیاده سازی کرده و فایر شدن آنها را به موتور ورک فلو اطلاع دهد تا فرایند ادامه یابد.

به همین دلیل ما برای سادگی اولا از ایجاد نود جداگانه برای رویدادها پرهیز کرده و در ثانی تفکیک منطقی برای رویداد درون ورک فلو نداریم. مثلا یک رویداد مجزا برای دریافت پیام و یک رویداد دیگر برای تایمر نداشته و همه را با یک تایپ رویداد می شناسیم که صرفا می تواند به تسک ها یا گیت وی اختصاص[[13]](#footnote-13) یابد.

**منابعی برای مطالعه بیشتر**

1. XPDL یا زبان تعریف فرایند یک DSL قدیمی برای تعریف فرایندها بود. حرف X به منزله کلی بودن این Specification است مثلا jPDL پیاده سازی آن به زبان جاوا بود که تا jBPM 4.4 مورد استفاده قرار می گرفت. این زبان یا DSL دارای پیچیدگی ها ناخواسته زیادی بود و اجرای برخی سناریوها با آن عملا ناممکن بود، مثل سناریوی پیاده سازی رویدادها.

<https://en.wikipedia.org/wiki/XPDL>

<https://sourceforge.net/projects/yxe/?source=recommended>

یک دیزاینر رایگان و اوپن سورس برای XPDL می باشد.

1. یکسری پیاده سازی ها از BPMN در پلاتفرم های مختلف

<https://en.wikipedia.org/wiki/List_of_BPMN_2.0_engines>

<https://github.com/adrobisch/bpmn.js>

<https://github.com/bpmn-io>

<https://en.wikipedia.org/wiki/JBPM>

<https://en.wikipedia.org/wiki/Activiti_(software)>

<https://sourceforge.net/projects/bpmn/>

~~یک دیزاینر رایگان و اوپن سورس برای BPMN2، صرفا دیزاینر بوده و فاقد موتور ورک فلو است. نسخه سفارشی شده از آن را می توانیم برای پروژه خود استفاده کنیم.~~

<https://sourceforge.net/projects/processmaker/?source=recommended>

1. BPEL نیز یک زبان دیگر برای پیاده سازی فرایندها می باشد که بنده با آن آشنایی کمی دارم. Oracle BPEL از پیاده سازی های معروف است. در ویکی زیر لیستی از انجین هایی که آن را پیاده سازی نموده اند، قابل مشاهده است:

<https://en.wikipedia.org/wiki/List_of_BPEL_engines>

نمودارهای این بخش با GoJS طراحی شده اند که یک دیزاینر تحت وب و جاوا اسریپتی برای BPMN2 است.

http://gojs.net/latest/extensions/BPMN.html#

توجه: این بخش هنوز آماده نشده است.

بخش دوم: طراحی و پیاده سازی موتور ورک فلو

**استراتژی طراحی**

- باید سناریوهای مختلف از BPMN‌ را به عنوان نمونه انجام دهیم.

- کانتکس اجرای ورک فلو درون خود آن اتفاق می افتد. یعنی کلاینت ورک فلو از نحوه گرداندن فرایندها مطلع نخواهد بود و صرفا باید با API‌ کار کند.

**ملاحضات پیاده سازی**

**روش های تعریف یک ورک فلو**

باید بتوان یک فلو را به دو صورت زیر ایجاد کرد:

1. با استفاده از Fluent API و از تعریف دینامیک از درون برنامه
2. تعریف در قالب XML و پاس دادن آن به انجین

**روش های تعریف اکشن ها**

اکشن های کدهای تعریف شده در برنامه هستند که باید ازای تسک ها (مثلا ServiceTask ها یا HumanTask ها) اجرا شوند. همچنین در موارد دیگری از اجرای ورک فلو می توان اکشن تعریف کرد که در بخش اول مورد بررسی قرار گرفته است.

دو روش کلی برای تعریف اکشن ها در نظر گرفته شده است. باید بتوان اکشن های تعریف شده را هم به صورت meta programming (مثلا انوتیشن ها در زبان جاوا) و هم به صورت پیاده سازی اینترفیس (مثلا به صورت inner anonymous class یا عبارات لامبدا) پاس دهیم.

روش اول به منظور سادگی، ایجاد dynamicity در نظر گرفته شده است و روش پیاده سازی اینترفیس و عبارات لامبدا این امکان را می دهد که بتوان موتور ورک فلو را در زبان های دیگر مانند C++11 که امکانات meta programming ضعیفی دارند، پیاده سازی کرد.

در پیاده سازی جاوایی ورک فلو باید از Java 6 استفاده شود تا امکان استفاده از آن با انواع فریم ورک ها و پروژه های قدیمی فراهم باشد.

- تایمرها و بقیه event‌های BPMN2 پیاده سازی نمی کنیم. امکان پیاده سازی آن ها توسط اکشن های خودد کلاینت امکان پذیر است.

Workflow API

Workflow : شامل تعریف ورک فلو

WorkflowBuilder

WorkflowManager: استارت و مدیریت نمونه های ورک فلو باید صرفا با این کلاس انجام شود.

WorkflowInstance: شامل اطلاعات زمان اجرای ورک فلو

Node

StartNode

EndNode

ServiceNode

TaskNode

Gateway

OrGateway

ANdGateway

برای سرویس ها

ورژن Meta Programming

@Service (id = 'service\_node\_id', type = BLOCKING یا NON\_BLOCKING)

void sendMailAction(WorkflowData, ServiceData)

ورژن پیاده سازی کلاس

interface Service {

void doAction(WorkflowData, ServiceData);

}

class SendMailAction implements Service {

@Override

void doAction(WorflowData data, ServiceData) {

}

}

- نکته: سرویس ها مانند تسک ها دو زمان برای invoke‌ شدن (on-entry یا on-exit) ندارند. زیرا سرویس ها باید به یکباره اجرا شوند.

برای وظایف انسانی

ورژن Meta Programming

@Task (id = 'human-task-id', type = BLOCKING یا NON\_BLOCKING, invokeOn = ON\_ENTRY یا ON\_EXIT)

void onTaskSendMail(WowrkflowData , TaskData)

ورژن پیاده سازی کلاس ها

interface Task {

void taskPointCut(WorkflowData , TaskData)

}

class BeforeSendMailTask implements OnEntryNonBlockingTask {

@Override

void doTask(WorkdlowData, TaskData)

}

آیا ‌Blocking‌یا Non-Blocking‌ بودن اکشن ها ضرورتی دارد؟ بله - این مسيله شاید ربطی به کلاینت نداشته باشد و در اصل اجرای ورک فلو را تحت تاثیر قرار می دهد.

- می توان به ازای تایپ سرویس (کلاس سرویس) یک نوع ServiceData ست کرد. (در صورت نیاز به flexibility‌بیشتر)

Condition

کاندیشن ها چگونه چک می شوند؟ بر اساس دیتای ورک فلو که به آنها پاس می شود.

- اسکریپت نویسی نداریم. یعنی هیچ کدی درون دیاگرام نباید باشد. صرفا برای صریح بودن و سادگی طراحی و پیاده سازی.

WorkflowInstance

getStatus() :

- NOT\_STARTED

- RUNNING

- FINISHED

- FAILED

- CANCELED : وقتی کاربری تسک خود را متوقف کرده باشد که ورک فلو از هیچ جای دیگری امکان ادامه نداشته باشد.

- Gateway های join یا converging نداریم. صرفا diverging‌به منظور کاهش پیچیدگی

\* ساختمان داده WorkflowInstance

- در آن واحد می توان چند نود جاری (در حال اجرا) داشت به منظور پشتیبانی از امکان فورک کردن فرایندها

- صرفا نودهای از نوع gateway‌می توانند چندین خروجی داشته باشند و بقیه نودها باید صرفا یک خروجی داشته باشند.

- دیتای تسک و ورک فلو به صورت JSON‌در جداول ذخیره سازی شوند تا امکان ویرایش راحت را داشته باشند.

سناریوی شماره ۱

تخصیص تسک ها توسط مدیر

مدیر می خواهد یک تسک را به کارتابل کاربری خاص بفرستد.

مدیر باید ابتدا تسک را ببیند.

user یا group = manager

سپس مدیر دیتای تسک را بروز می کند بدون این که آن تسک را استارت کند یعنی

user یا group = scott

یعنی کاربر تسک را از مدیر به اسکات تغییر می دهد.

سناریوی شماره ۲

تایمر

کاربر باید مثلا یک سرویس تسک قبل از شرط بگذارد که تایمری را ست کند و دیتای ورک فلو را آپدیت کند و موقع رسیدن به شرط آن تایمر را چک کند.

اگر لازم بود باید تایمر بیرون کد ست شود و مثلا به صورت دوره ای دیتای ورک فلو بررسی شود. به همین دلیل نیاز به یک اصلاحیه بر روی condition داریم.

زمان تعریف کاندیشن باید بگوییم آن را تا برقرار شدن یکی از شرط ها به صورت دوره ای چک کند و یا خیر.

addCondition(node\_id, from\_node\_id, to\_node\_id)

addPeriodicCondition(node\_id, from\_node\_id, to\_node\_id, periodic\_time, time\_unit)

time\_unit: Second, Minute, Hour, Week, Month, Year

WorkflowManager

startWorkflowInstance(instance\_id)

stopWorkflowInstance(instance\_id)

startup() : از روی دیتابیس ورک فلوهای فعال را به جریان می اندازد

shutdown(): تمامی ورک فلوهای در حال اجرا را متوقف می کند. احتمالا اپلیکیشن هنگام متوقف شدن این متد را فراخوانی می کند تا data instability پیش نیاید.

restartWorkflow() : شروع مجدد ورک فلو

proceedWorkflow(instance\_id, node\_id): یک instance از ورک فلو را به نود خاص می برد.

createInstance(workflow\_id) : WorkflowInstance

1. specification [↑](#footnote-ref-1)
2. implementation [↑](#footnote-ref-2)
3. developer friendly [↑](#footnote-ref-3)
4. subset [↑](#footnote-ref-4)
5. commercial [↑](#footnote-ref-5)
6. Workflow Elements [↑](#footnote-ref-6)
7. Service Based Tasks [↑](#footnote-ref-7)
8. Human Tasks [↑](#footnote-ref-8)
9. Connections [↑](#footnote-ref-9)
10. Diverging [↑](#footnote-ref-10)
11. Converging [↑](#footnote-ref-11)
12. Events [↑](#footnote-ref-12)
13. Assign [↑](#footnote-ref-13)