عنوان: **طراحی گردش کاری با استفاده از State machines - قسمت اول**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۱۰ ۲۱:۱۰
آدرس: www.dotnettips.info
برچسب‌ها: Workflow, State machine

State machine چیست؟

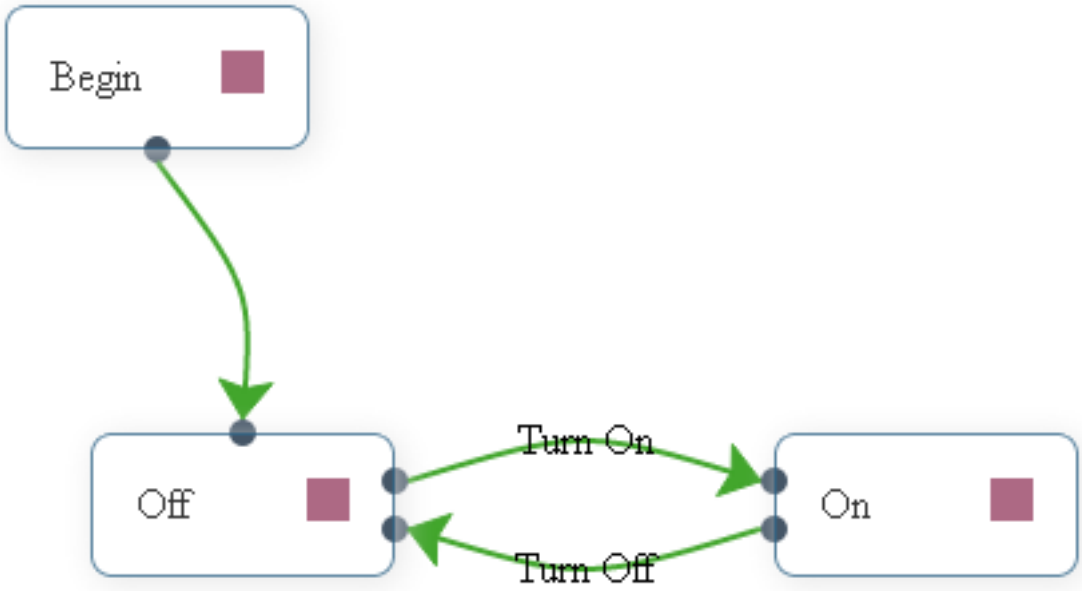
State machine مدلی است بیانگر نحوه واکنش سیستم به وقایع مختلف. یک ماشین حالت وضعیت جاری قسمتی از سیستم را نگهداری کرده و به ورودی‌های مختلف پاسخ می‌دهد. این ورودی‌ها در نهایت وضعیت سیستم را تغییر خواهند داد.
نحوه پاسخگویی یک ماشین حالت (State machine) را به رویدادی خاص، انتقال (Transition) می‌نامند. در یک انتقال مشخص می‌شود که ماشین حالت بر اساس وضعیت جاری خود، با دریافت یک رویداد، چه عکس العملی را باید بروز دهد. عموما (و نه همیشه) در حین پاسخگویی ماشین حالت به رویدادهای رسیده، وضعیت آن نیز تغییر خواهد کرد. در اینجا گاهی از اوقات پیش از انجام عملیاتی، نیاز است شرطی بررسی شده و سپس انتقالی رخ دهد. به این شرط، guard گفته می‌شود.
بنابراین به صورت خلاصه، یک ماشین حالت، مدلی است از رفتاری خاص، تشکیل شده از حالات، رویدادها، انتقالات، اعمال (actions) و شرطها (Guards). در اینجا:
- یک حالت (State)، شرطی منحصربفرد در طول عمر ماشین حالت است. در هر زمان مشخصی، ماشین حالت در یکی از حالات از پیش تعریف شده خود قرار خواهد داشت.
- یک رویداد (Event)، اتفاقی است که به ماشین حالت اعمال می‌شود؛ یا همان ورودی‌های سیستم.
- یک انتقال (Transition)، بیانگر نحوه رفتار ماشین حالت جهت پاسخگویی به رویداد وارده بر اساس وضعیت جاری خود می‌باشد. در طی یک انتقال، سیستم از یک حالت به حالتی دیگر منتقل خواهد شد.
- برای انجام یک انتقال، نیاز است یک شرط (Guard/Conditional Logic) بررسی شده و در صورت true بودن آن، انتقال صورت گیرد.
- یک عمل (Action)، بیانگر نحوه پاسخگویی ماشین حالت در طول دوره انتقال است.

چگونه می‌توان الگوی ماشین حالت را تشخیص داد؟

اکثر برنامه‌های وب، متشکل از پیاده سازی چندین ماشین حالت می‌باشند؛ مانند ثبت نام در سایت، درخواست یک کتاب از کتابخانه، ارسال درخواست‌ها و پاسخگویی به آن‌ها و یا حتی ارسال یک مطلب در سایت، تائید و انتشار آن.
البته عموما در حین طراحی برنامه‌ها، کمتر به این نوع مسایل به شکل یک ماشین حالت نگاه می‌شود. به همین جهت بهتر است معیارهایی را برای شناخت زود هنگام آن‌ها مدنظر داشته باشیم:
- آیا در جدول بانک اطلاعاتی خود فیلدهایی مانند State (حالت) یا Status (وضعیت)دارید؟ اگر بله، به این معنا است که در حال کار با یک ماشین حالت هستید.
- عموما فیلدهای Bit و Boolean، بیانگر حضور ماشین‌های حالت هستند. مانند IsPaid ، IsPublished و یا حتی داشتن یک فیلد timeStamp که می‌تواند NULL بپذیرد نیز بیانگر استفاده از ماشین حالت است؛ مانند فیلدهای paid_at، published_at و یا confirmed_at.
- داشتن رکوردهایی که تنها در طول یک بازه زمانی خاص، معتبر هستند. برای مثال آبونه شدن در یک سایت در طول یک بازه زمانی مشخص.
- اعمال چند مرحله‌ای؛ مانند ثبت نام در سایت و دریافت ایمیل فعال سازی. سپس فعال سازی اکانت از طریق ایمیل.

مثالی ساده از یک ماشین حالت

یک کلید برق را در نظر بگیرید. این کلید دارای دو حالت (states) روشن و خاموش است. زمانی که خاموش است، با دریافت رخدادی (event)، به وضعیت (state/status) روشن، منتقل خواهد شد (Transition) و برعکس.



در اینجا حالات با مستطیل‌های گوشه گرد نمایش داده شده‌اند. انتقالات توسط فلش‌هایی انحناء دار که حالات را به یکدیگر متصل می‌کنند، مشخص گردیده‌اند. برچسب‌های هر فلش، مشخص کننده نام رویدادی است که سبب انتقال و تغییر حالت می‌گردد. با شروع یک ماشین حالت، این ماشین در یکی از وضعیت‌های از پیش تعیین شده‌اش قرار خواهد گرفت (initial state)؛ که در اینجا حالت خاموش است.
این نوع نمودارها می‌توانند شامل جزئیات بیشتری نیز باشند؛ مانند برچسب‌هایی که نمایانگر اعمال قابل انجام در طی یک انتقال هستند.

رسم ماشین‌های حالت در برنامه‌های وب، به کمک کتابخانه jsPlumb

کتابخانه‌های زیادی برای رسم فلوچارت، گردش‌های کاری، ماشین‌های حالت و امثال آن جهت برنامه‌های وب وجود دارند و یکی از معروف‌ترین‌های آن‌ها کتابخانه jsPlumb است. این کتابخانه به صورت یک افزونه jQuery طراحی شده است؛ اما به عنوان افزونه‌ای برای کتابخانه‌های MooTools و یا YUI3/Yahoo User Interface 3 نیز قابل استفاده می‌باشد. کتابخانه jsPlumb در مرورگرهای جدید از امکانات ترسیم SVG و یا HTML5 Canvas استفاده می‌کند. برای سازگاری با مرورگرهای قدیمی‌تر مانند IE8 به صورت خودکار به VML سوئیچ خواهد کرد. همچنین این کتابخانه امکانات ترسیم تعاملی قطعات به هم متصل شونده را نیز دارا است (شبیه به طراح یک گردش کاری). البته برای اضافه شدن امکاناتی مانند کشیدن و رها کردن در آن نیاز به jQuery-UI نیز خواهد داشت.
برای نمونه اگر بخواهیم مثال فوق را توسط jsPlumb ترسیم کنیم، روش کار به صورت زیر خواهد بود:

```
<!doctype html>
<html>
<head>
    <title>State Machine Demonstration</title>
    <style type="text/css">
        #opened
        {
            left: 10em;
            top: 5em;
        }

        #off
        {
            left: 12em;
            top: 15em;
        }
```

```
        #on
        {
            left: 28em;
            top: 15em;
        }

        .w
        {
            width: 5em;
            padding: 1em;
            position: absolute;
            border: 1px solid black;
            z-index: 4;
            border-radius: 1em;
            border: 1px solid #346789;
            box-shadow: 2px 2px 19px #e0e0e0;
            -o-box-shadow: 2px 2px 19px #e0e0e0;
            -webkit-box-shadow: 2px 2px 19px #e0e0e0;
            -moz-box-shadow: 2px 2px 19px #e0e0e0;
            -moz-border-radius: 0.5em;
            border-radius: 0.5em;
            opacity: 0.8;
            filter: alpha(opacity=80);
            cursor: move;
        }

        .ep
        {
            float: right;
            width: 1em;
            height: 1em;
            background-color: #994466;
            cursor: pointer;
        }

        .labelClass
        {
            font-size: 20pt;
        }
    </style>
    <script type="text/javascript" src="jquery.min.js"></script>
    <script type="text/javascript" src="jquery-ui.min.js"></script>
    <script type="text/javascript" src="jquery.jsPlumb-all-min.js"></script>
    <script type="text/javascript">
        $(document).ready(function () {

            jsPlumb.importDefaults({
                Endpoint: ["Dot", { radius: 5}],
                HoverPaintStyle: { strokeStyle: "blue", lineWidth: 2 },
                ConnectionOverlays: [
["Arrow", { location: 1, id: "arrow", length: 14, foldback: 0.8}]
]
            });

            jsPlumb.makeTarget($(".w"), {
                dropOptions: { hoverClass: "dragHover" },
                anchor: "Continuous"
            });

            $(".ep").each(function (i, e) {
                var p = $(e).parent();
                jsPlumb.makeSource($(e), {
                    parent: p,
                    anchor: "Continuous",
                    connector: ["StateMachine", { curviness: 20}],
                    connectorStyle: { strokeStyle: '#42a62c', lineWidth: 2 },
                    maxConnections: 2,
                    onMaxConnections: function (info, e) {
                        alert("Maximum connections (" + info.maxConnections + ") reached");
                    }
                });
            });

            jsPlumb.bind("connection", function (info) {
            });

            jsPlumb.draggable($(".w"));

            jsPlumb.connect({ source: "opened", target: "off" });
            jsPlumb.connect({ source: "off", target: "on", label: "Turn On" });
            jsPlumb.connect({ source: "on", target: "off", label: "Turn Off" });
```

```
        });
    </script>
</head>
<body>
    <div class="w" id="opened">
        Begin
        <div class="ep">
        </div>
    </div>
    <div class="w" id="off">
        Off
        <div class="ep">
        </div>
    </div>
    <div class="w" id="on">
        On
        <div class="ep">
        </div>
    </div>
</body>
</html>
```

مستندات کامل jsPlumb را در سایت آن می‌توان ملاحظه نمود.
در مثال فوق، ابتدا css و فایل‌های js مورد نیاز ذکر شده‌اند. توسط css، مکان قرارگیری اولیه المان‌های متناظر با حالات، مشخص می‌شوند.
سپس زمانیکه اشیاء صفحه در دسترس هستند، تنظیمات jsPlumb انجام خواهد شد. برای مثال در اینجا نوع نمایشی Endpointها به نقطه تنظیم شده است. موارد دیگری مانند مستطیل نیز قابل تنظیم است. سپس نیاز است منبع و مقصدها به کتابخانه jsPlumb معرفی شوند. به کمک متد jsPlumb.makeTarget، تمام المان‌های دارای کلاس w به عنوان منبع و با شمارش divهایی با class=ep، مقصدهای قابل اتصال تعیین شده‌اند (jsPlumb.makeSource). متد jsPlumb.bind یک callback function است و هربار که اتصالی برقرار می‌شود، فراخوانی خواهد شد. متد jsPlumb.draggable تمام عناصر دارای کلاس w را قابل کشیدن و رها کردن می‌کند و در آخر توسط متدهای jsPlumb.connect، مقصد و منبع‌های مشخصی را هم متصل خواهیم کرد. نمونه نهایی تهیه شده برای بررسی بیشتر.

برای مطالعه بیشتر

Finite-state machine

UML state machine

UML 2 State Machine Diagrams

مثال‌هایی در این مورد

نظرات خوانندگان

نویسنده: omid
تاریخ: ۱۳۹۱/۱۰/۱۱ ۷:۷

سلام
با تشکر از مطلب بسیار جالبی که بیان کردید . می‌خواستم بدونم آیا این امکان وجود داره که سمت سرور بعد از طراحی ، گردش کار رو ذخیره کنیم ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۱۱ ۱۱:۷

jsPlumb یک سری callback function داره که زمان اتصال نودها و یا زمان قطع اتصالات فراخوانی خواهند شد:

```
jsPlumb.bind("jsPlumbConnection", function(connectionInfo) {
  // update your data model here.
});

jsPlumb.bind("jsPlumbConnectionDetached", function(connectionInfo) {
  // update your data model here.
});
```

در اینجا شما فرصت خواهید داشت اطلاعات مدل مورد نظر را به روز کنید.
connectionInfo دریافتی یک شیء جاوا اسکریپتی است شامل connection, source, sourceEndpoint, sourceId, target, targetEndpoint, targetId

نویسنده: علیرضا
تاریخ: ۱۳۹۱/۱۰/۱۱ ۱۲:۲۳

سلام،
آیا چنین افزونه یا تکنیکی برای پیاده سازی SM داخل برنامه‌های تحت ویندوز (.net) وجود داره؟
ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۱۱ ۱۲:۳۵

در قسمت بعد این مساله دنبال خواهد شد.

نویسنده: امیر بختیاری
تاریخ: ۱۳۹۱/۱۰/۱۱ ۱۳:۱۵

با سلام
با تشکر از مطلب خوبی که بیان کردید.
امکان تعریف WorkFlowهای پیچیده و با شرایط و تنظیمات پیچیده نیز وجود دارد؟
به غیر از این کامپوننت ، کامپوننت‌های دیگری هم وجود داره ؟ اگر ممکنه لیستی از این کامپوننت‌ها را قرار بدهید.
همچنین میشه بعد از رسم کامل workFlow ذخیره سازی را انجام بدیم و بعد دوباره به همان شکل لود کنیم ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۱۱ ۱۴:۱۶

- بله. در قسمت بعد این مساله با معرفی یک کتابخانه مدیریت ماشین‌های حالت، دنبال خواهد شد.

- موارد دیگری مانند WireIt ، (^) draw.io ، raphaeljs و jGraph هم برای رسم گراف هستند.
- بله. باید کمی به jQuery Ajax آشنا باشید. می‌تونید اشیایی رو که قرار هست در صفحه ترسیم بشن به صورت آرایه‌ای از اشیاء جاوا اسکریپتی تعریف کنید. هر شیء دارای source و target است به علاوه مختصات x و y. نهایتا برای ارسال آن به سرور از طریق jQuery Ajax خواهید داشت:

```
JSON.stringify(whole_object)
```

برای دریافت لیست اشیاء هم به صورت JSON از سرور و رسم آن در سمت کلاینت با JSON.decode می‌تونید شروع کنید.

عنوان: **طراحی گردش کاری با استفاده از State machines - قسمت دوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۱۱ ۲۳:۱۰
آدرس: www.dotnettips.info
گروه‌ها: WorkFlow Foundation, Workflow, State machine

معرفی کتابخانه stateless به عنوان جایگزین سبک وزنی برای Windows workflow foundation

کتابخانه سورس باز Stateless ، برای طراحی و پیاده سازی «ماشین‌های حالت گردش کاری مانند» تهیه شده و مزایای زیر را نسبت به Windows workflow foundation دارا است:

- جمعا 30 کیلوبایت است!
- تمام اجزای آن سورس باز است.
- دارای API روان و ساده‌ای است.
- امکان تبدیل UML state diagrams، به نمونه معادل Stateless بسیار ساده و سریع است.
- به دلیل code first بودن، کار کردن با آن برای برنامه نویس‌ها ساده‌تر بوده و افزودن یا تغییر اجزای آن با کدنویسی به سادگی میسر است.

دریافت کتابخانه Stateless از Google code و یا از NuGet

پیاده سازی مثال کلید برق با Stateless

در ادامه همان مثال ساده کلید برق قسمت قبل را با Stateless پیاده سازی خواهیم کرد:

```
using System;
using Stateless;

namespace StatelessTests
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                string on = "On", off = "Off";
                var space = ' ';

                var onOffSwitch = new StateMachine<string, char>(initialState: off);

                onOffSwitch.Configure(state: off).Permit(trigger: space, destinationState: on);
                onOffSwitch.Configure(state: on).Permit(trigger: space, destinationState: off);

                Console.WriteLine("Press <space> to toggle the switch. Any other key will raise an
error.");

                while (true)
                {
                    Console.WriteLine("Switch is in state: " + onOffSwitch.State);
                    var pressed = Console.ReadKey(true).KeyChar;
                    onOffSwitch.Fire(trigger: pressed);
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine("Exception: " + ex.Message);
                Console.WriteLine("Press any key to continue...");
                Console.ReadKey(true);
            }
        }
    }
}
```

کار با ایجاد یک وهله از ماشین حالت (new StateMachine) آغاز می‌شود. حالت آغازین آن (initialState) مطابق مثال قسمت قبل، مساوی off است.

امضای کلاس StateMachine را در ذیل مشاهده می‌کنید؛ جهت توضیح آرگومان‌های جنریک string و char معرفی شده در مثال:

```
public class StateMachine<TState, TTrigger>
```

که اولی بیانگر نوع حالات قابل تعریف است و دومی نوع رویداد قابل دریافت را مشخص می‌کند.
برای مثال در اینجا حالات روشن و خاموش، با رشته‌های on و off مشخص شده‌اند و رویداد قابل قبول دریافتی، کاراکتر فاصله است.
سپس نیاز است این ماشین حالت را برای معرفی رویدادهایی (trigger در اینجا) که سبب تغییر حالت آن می‌شوند، تنظیم کنیم.
اینکار توسط متدهای Configure و Permit انجام خواهد شد. متد Configure، یکی از حالات از پیش تعیین شده را جهت تنظیم، مشخص می‌کند و سپس در متد Permit تعیین خواهیم کرد که بر اساس رخدادی مشخص (برای مثال در اینجا فشرده شدن کلید space) وضعیت حالت جاری، به وضعیت جدیدی (destinationState) منتقل شود.
نهایتا این ماشین حالت در یک حلقه بی‌نهایت مشغول به کار خواهد شد. برای نمونه یک Thread پس زمینه (BackgroundWorker) نیز می‌تواند همین کار را در برنامه‌های ویندوزی انجام دهد.

یک نکته

علاوه بر روش‌های یاد شده‌ی تشخیص الگوی ماشین حالت که در قسمت قبل بررسی شدند، مورد refactoring انبوهی از if و elseها و یا switchهای بسیار طولانی را نیز می‌توان به این لیست افزود.

استفاده از Stateless Designer برای تولید کدهای ماشین حالت

کتابخانه Stateless دارای یک طراح و Code generator بصری سورس باز است که آن‌را به شکل افزونه‌ای برای VS.NET می‌توانید در سایت Codeplex دریافت کنید . این طراح از کتابخانه GLEE برای رسم گراف استفاده می‌کند.

کار مقدماتی با آن به نحو زیر است:
الف) فایل StatelessDesignerPackage.vsix را از سایت کدپلکس دریافت و نصب کنید. البته نگارش فعلی آن فقط با VS 2012 سازگار است.
ب) ارجاعی را به اسمبلی stateless به پروژه خود اضافه نمائید (به یک پروژه جدید یا از پیش موجود).
ج) از منوی پروژه، گزینه Add new item را انتخاب کرده و سپس در صفحه ظاهر شده، گزینه جدید Stateless state machine را انتخاب و به پروژه اضافه نمائید.
کار با این طراح، با ادیت XML آن شروع می‌شود. برای مثال گردش کاری ارسال و تائید یک مطلب جدید را در بلاگی فرضی، به نحو زیر وارد نمائید:

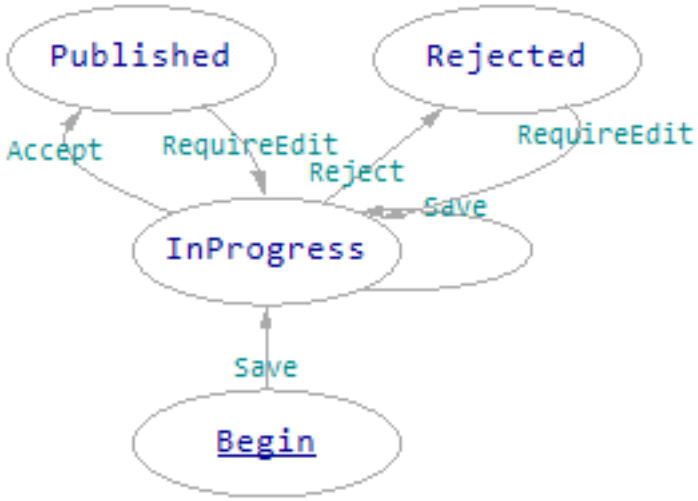
```
<statemachine xmlns="http://statelessdesigner.codeplex.com/Schema">
  <settings>
    <itemname>BlogPostStateMachine</itemname>
    <namespace>StatelessTests</namespace>
    <class>public</class>
  </settings>
  <triggers>
    <trigger>Save</trigger>
    <trigger>RequireEdit</trigger>
    <trigger>Accept</trigger>
    <trigger>Reject</trigger>
  </triggers>
  <states>
    <state start="yes">Begin</state>
    <state>InProgress</state>
    <state>Published</state>
    <state>Rejected</state>
  </states>
  <transitions>
    <transition trigger="Save" from="Begin" to="InProgress" />

    <transition trigger="Accept" from="InProgress" to="Published" />
    <transition trigger="Reject" from="InProgress" to="Rejected" />

    <transition trigger="Save" from="InProgress" to="InProgress" />
```

```
    <transition trigger="RequireEdit" from="Published" to="InProgress" />
    <transition trigger="RequireEdit" from="Rejected" to="InProgress" />
  </transitions>
</statemachine>
```

حاصل آن گراف زیر خواهد بود:



به علاوه کدهای زیر که به صورت خودکار تولید شده‌اند:

```
using Stateless;

namespace StatelessTests
{
  public class BlogPostStateMachine
  {
    public delegate void UnhandledTriggerDelegate(State state, Trigger trigger);
    public delegate void EntryExitDelegate();
    public delegate bool GuardClauseDelegate();

    public enum Trigger
    {
      Save,
      RequireEdit,
      Accept,
      Reject,
    }

    public enum State
    {
      Begin,
      InProgress,
      Published,
      Rejected,
    }

    private readonly StateMachine<State, Trigger> stateMachine = null;

    public EntryExitDelegate OnBeginEntry = null;
    public EntryExitDelegate OnBeginExit = null;
    public EntryExitDelegate OnInProgressEntry = null;
    public EntryExitDelegate OnInProgressExit = null;
    public EntryExitDelegate OnPublishedEntry = null;
    public EntryExitDelegate OnPublishedExit = null;
    public EntryExitDelegate OnRejectedEntry = null;
    public EntryExitDelegate OnRejectedExit = null;
    public GuardClauseDelegate GuardClauseFromBeginToInProgressUsingTriggerSave = null;
```

```
    public GuardClauseDelegate GuardClauseFromInProgressToPublishedUsingTriggerAccept = null;
    public GuardClauseDelegate GuardClauseFromInProgressToRejectedUsingTriggerReject = null;
    public GuardClauseDelegate GuardClauseFromInProgressToInProgressUsingTriggerSave = null;
    public GuardClauseDelegate GuardClauseFromPublishedToInProgressUsingTriggerRequireEdit = null;
    public GuardClauseDelegate GuardClauseFromRejectedToInProgressUsingTriggerRequireEdit = null;
    public UnhandledTriggerDelegate OnUnhandledTrigger = null;

    public BlogPost()
    {
      stateMachine = new StateMachine<State, Trigger>(State.Begin);
      stateMachine.Configure(State.Begin)
        .OnEntry(() => { if (OnBeginEntry != null) OnBeginEntry(); })
        .OnExit(() => { if (OnBeginExit != null) OnBeginExit(); })
        .PermitIf(Trigger.Save, State.InProgress , () => { if
(GuardClauseFromBeginToInProgressUsingTriggerSave != null) return
GuardClauseFromBeginToInProgressUsingTriggerSave(); return true; } )
      ;
      stateMachine.Configure(State.InProgress)
        .OnEntry(() => { if (OnInProgressEntry != null) OnInProgressEntry(); })
        .OnExit(() => { if (OnInProgressExit != null) OnInProgressExit(); })
        .PermitIf(Trigger.Accept, State.Published , () => { if
(GuardClauseFromInProgressToPublishedUsingTriggerAccept != null) return
GuardClauseFromInProgressToPublishedUsingTriggerAccept(); return true; } )
        .PermitIf(Trigger.Reject, State.Rejected , () => { if
(GuardClauseFromInProgressToRejectedUsingTriggerReject != null) return
GuardClauseFromInProgressToRejectedUsingTriggerReject(); return true; } )
        .PermitReentryIf(Trigger.Save , () => { if
(GuardClauseFromInProgressToInProgressUsingTriggerSave != null) return
GuardClauseFromInProgressToInProgressUsingTriggerSave(); return true; } )
      ;
      stateMachine.Configure(State.Published)
        .OnEntry(() => { if (OnPublishedEntry != null) OnPublishedEntry(); })
        .OnExit(() => { if (OnPublishedExit != null) OnPublishedExit(); })
        .PermitIf(Trigger.RequireEdit, State.InProgress , () => { if
(GuardClauseFromPublishedToInProgressUsingTriggerRequireEdit != null) return
GuardClauseFromPublishedToInProgressUsingTriggerRequireEdit(); return true; } )
      ;
      stateMachine.Configure(State.Rejected)
        .OnEntry(() => { if (OnRejectedEntry != null) OnRejectedEntry(); })
        .OnExit(() => { if (OnRejectedExit != null) OnRejectedExit(); })
        .PermitIf(Trigger.RequireEdit, State.InProgress , () => { if
(GuardClauseFromRejectedToInProgressUsingTriggerRequireEdit != null) return
GuardClauseFromRejectedToInProgressUsingTriggerRequireEdit(); return true; } )
      ;
      stateMachine.OnUnhandledTrigger((state, trigger) => { if (OnUnhandledTrigger != null)
OnUnhandledTrigger(state, trigger); });
    }

    public bool TryFireTrigger(Trigger trigger)
    {
      if (!stateMachine.CanFire(trigger))
      {
        return false;
      }
      stateMachine.Fire(trigger);
      return true;
    }

    public State GetState
    {
      get
      {
        return stateMachine.State;
      }
    }
  }
}
```

توضیحات:

ماشین حالت فوق دارای چهار حالت شروع، در حال بررسی، منتشر شده و رد شده است. معمول است که این چهار حالت را به شکل یک enum معرفی کنند که در کدهای تولیدی فوق نیز به همین نحو عمل گردیده و public enum State معرف چهار حالت ذکر شده است. همچنین رویدادهای ذخیره، نیاز به ویرایش، ویرایش، تائید و رد نیز توسط public enum Trigger معرفی شده‌اند.
در قسمت Transitions، بر اساس یک رویداد (Trigger در اینجا)، انتقال از یک حالت به حالتی دیگر را سبب خواهیم شد.
تعاریف اصلی تنظیمات ماشین حالت، در سازنده کلاس BlogPostStateMachine انجام شده است. این تعاریف نیز بسیار ساده

هستند. به ازای هر حالت، یک Configure داریم. در متدهای OnEntry و OnExit هر حالت، یک سری callback function فراخوانی خواهند شد. برای مثال در حالت Rejected یا Approved می‌توان ایمیلی را به ارسال کننده مطلب جهت یادآوری وضعیت رخ داده، ارسال نمود.
متدهای PermitIf سبب انتقال شرطی، به حالتی دیگر خواهند شد. برای مثال رد یا تائید یک مطلب نیاز به دسترسی مدیریتی خواهد داشت. این نوع موارد را توسط delgateهای Guard ایی که برای مدیریت شرطها ایجاد کرده است، می‌توان تنظیم کرد.
PermitReentryIf سبب بازگشت مجدد به همان حالت می‌گردد. برای مثال ویرایش و ذخیره یک مطلب در حال انتشار، سبب تائید یا رد آن نخواهد شد؛ صرفا عملیات ذخیره صورت گرفته و ماشین حالت مجددا در همان مرحله باقی خواهد ماند.

نحوه استفاده از ماشین حالت تولیدی:
همانطور که عنوان شد، حداقل استفاده از ماشین‌های حالت، refactoing انبوهی از if و elseها است که در حالت مدیریت یک چنین گردش‌های کاری باید تدارک دید.

```
namespace StatelessTests
{
    public class BlogPostManager
    {
        private BlogPostStateMachine _stateMachine;
        public BlogPostManager()
        {
            configureWorkflow();
        }

        private void configureWorkflow()
        {
            _stateMachine = new BlogPostStateMachine();

            _stateMachine.GuardClauseFromBeginToInProgressUsingTriggerSave = () => { return
UserCanPost; };
            _stateMachine.OnBeginExit = () => { /* save data + save state + send an email to admin */
};

            _stateMachine.GuardClauseFromInProgressToPublishedUsingTriggerAccept = () => { return
UserIsAdmin; };
            _stateMachine.GuardClauseFromInProgressToRejectedUsingTriggerReject = () => { return
UserIsAdmin; };
            _stateMachine.GuardClauseFromInProgressToInProgressUsingTriggerSave = () => { return
UserHasEditRights; };
            _stateMachine.OnInProgressExit = () => { /* save data + save state + send an email to user
*/ };

            _stateMachine.OnPublishedExit = () => { /* save data + save state + send an email to admin
*/ };
            _stateMachine.GuardClauseFromPublishedToInProgressUsingTriggerRequireEdit = () => { return
UserHasEditRights; };

            _stateMachine.OnRejectedExit = () => { /* save data + save state + send an email to admin
*/ };
            _stateMachine.GuardClauseFromRejectedToInProgressUsingTriggerRequireEdit = () => { return
UserHasEditRights; };
        }

        public bool UserIsAdmin
        {
            get
            {
                return true; // TODO: Evaluate if user is an admin.
            }
        }

        public bool UserCanPost
        {
            get
            {
                return true; // TODO: Evaluate if user is authenticated
            }
        }

        public bool UserHasEditRights
        {
            get
            {
                return true; // TODO: Evaluate if user is owner or admin
            }
```

```
        }

        // User actions
        public void Save() { _stateMachine.TryFireTrigger(BlogPostStateMachine.Trigger.Save); }
        public void RequireEdit() {
_stateMachine.TryFireTrigger(BlogPostStateMachine.Trigger.RequireEdit); }

        // Admin actions
        public void Accept() { _stateMachine.TryFireTrigger(BlogPostStateMachine.Trigger.Accept); }
        public void Reject() { _stateMachine.TryFireTrigger(BlogPostStateMachine.Trigger.Reject); }
    }
}
```

در کلاس فوق، نحوه استفاده از ماشین حالت تولیدی را مشاهده می‌کنید. در delegateهای Guard، سطوح دسترسی انجام عملیات بررسی خواهند شد. برای مثال، از بانک اطلاعاتی بر اساس اطلاعات کاربر جاری وارد شده به سیستم اخذ می‌گردند. در متدهای Exit هر مرحله، کارهای ذخیره سازی اطلاعات در بانک اطلاعاتی، ذخیره سازی حالت (مثلا در یک فیلد که بعدا قابل بازیابی باشد) صورت می‌گیرد و در صورت نیاز ایمیلی به اشخاص مختلف ارسال خواهد شد.
برای به حرکت درآوردن این ماشین، نیاز به یک سری اکشن متد نیز می‌باشد. تعدادی از این موارد را در انتهای کلاس فوق ملاحظه می‌کنید. کد نویسی آن‌ها در حد فراخوانی متد TryFireTrigger ماشین حالت است.

یک نکته:
ماشین حالت تولیدی به صورت پیش فرض در حالت State.Begin قرار دارد. می‌توان این مورد را از بانک اطلاعاتی خواند و سپس مقدار دهی نمود تا با هر بار وهله سازی ماشین حالت دقیقا مشخص باشد که در چه مرحله‌ای قرار داریم و TryFireTrigger بتواند بر این اساس تصمیم‌گیری کند که آیا مجاز است عملیاتی را انجام دهد یا خیر.

نظرات خوانندگان

نویسنده: برنامه نویس
تاریخ: ۱۳۹۱/۱۰/۱۴ ۱۰:۲۵

استفاده از یک Dictionary از نوع string و Action، چه مشکلی نسبت به این روش داره؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۱۴ ۱۰:۴۶

مشکلی نداره. شما در هر زمانی می‌تونید دست به اختراع مجدد چرخ بزنید. با یک Dictionary از نوع string و Action فقط قسمت حالات و رویدادها رو طراحی کردید. مابقی قسمت‌ها مانند انتقال‌ها رو هم که اضافه کنید می‌شود کتابخانه Stateless.

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۳/۰۶/۱۶ ۲۳:۲۶

طبق کد زیر:

```
_stateMachine.OnPublishedExit = () => { /* save data + save state + send an email to admin */ };
_stateMachine.GuardClauseFromPublishedToInProgressUsingTriggerRequireEdit = () => { return
UserHasEditRights };
```

آیا درست متوجه شدم که: بازهم ابتدا سطح دسترسی بررسی می‌شود و سپس عملیات ذخیره سازی صورت می‌پذیرد؟!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۶/۱۶ ۲۳:۴۲

این‌ها فقط یک سری تنظیم اولیه سیستم هستند. مهم نیست ترتیب معرفی آن‌ها به چه صورتی است. اجرای آن‌ها اینجا انجام نمی‌شود.

نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۳/۰۶/۱۷ ۰:۱۶

ظاهرا در کلاس BlogPostStateMachine، متد سازنده آن به اشتباه BlogPost درج شده است.
برای تغییر حالت و مقدار دهی آن از بانک اطلاعاتی، باید کد کلاس BlogPostStateMachine را مانند زیر تغییر دهیم:

```
public BlogPostStateMachine(State state)
    {
      stateMachine = new StateMachine<State, Trigger>(state);
```

عنوان: **طراحی گردش کاری با استفاده از State machines - قسمت سوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۱۲ ۱۵:۵
آدرس: www.dotnettips.info
گروه‌ها: Workflow, State machine

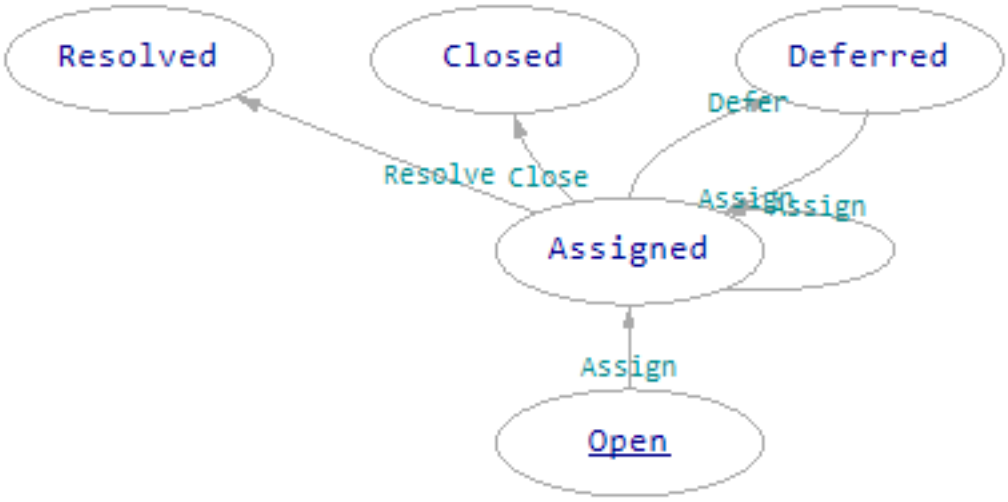
در این قسمت، یک سری مثال گردش کاری سازگار با Stateless Designer را بررسی خواهیم کرد. خروجی‌های XML زیر را می‌توانید در Stateless Designer وارد کرده و تبدیل به کدهای معادل کنید. اگر نمونه‌ای را هم خودتان طراحی کرده‌اید می‌توانید در قسمت نظرات مطلب جاری به اشتراک بگذارید.

الف) طراحی گردش کاری یک سیستم ردیابی خطاها (Bug tracking system)

در ادامه رویدادها، حالات و انتقالات یک ماشین حالت ردیابی خطاها را مشاهده می‌کنید:

```
<statemachine xmlns="http://statelessdesigner.codeplex.com/Schema">
  <settings>
    <itemname>BugTrackingStateMachine</itemname>
    <namespace>StatelessTests</namespace>
    <class>public</class>
  </settings>
  <triggers>
    <trigger>Assign</trigger>
    <trigger>Defer</trigger>
    <trigger>Resolve</trigger>
    <trigger>Close</trigger>
  </triggers>
  <states>
    <state start="yes">Open</state>
    <state>Assigned</state>
    <state>Deferred</state>
    <state>Resolved</state>
    <state>Closed</state>
  </states>
  <transitions>
    <transition trigger="Assign" from="Open" to="Assigned" />
    <transition trigger="Assign" from="Assigned" to="Assigned" />
    <transition trigger="Close" from="Assigned" to="Closed" />
    <transition trigger="Defer" from="Assigned" to="Deferred" />
    <transition trigger="Assign" from="Deferred" to="Assigned" />
    <transition trigger="Resolve" from="Assigned" to="Resolved" />
  </transitions>
</statemachine>
```

با گرافی معادل:



توضیحات:
یک گزارش خطا حداقل پنج حالت آغاز (Open)، انتساب به شخص، جهت رفع مشکل (Assign)، به تاخیر افتادن/درحال بررسی (Deffered)، برطرف شده (Resolved) و خاتمه یافته/برطرف نخواهد شد (Closed) را می‌تواند داشته باشد.
برای حرکت (Transition) از هر حالت به حالتی دیگر نیاز به یک سری رویداد (Trigger) است که لیست آن‌ها را در بالا مشاهده می‌کنید.
در ابتدا سیستم در حالت انتساب به شخص قرار می‌گیرد. سپس در همین حالت شخص می‌تواند یکی از سه حالت رفع شده، بستن موضوع و یا ارجاع به زمانی دیگر را انتخاب کند. حتی در حالت ارجاع به شخص، شخص می‌تواند مساله را به شخصی دیگر ارجاع دهد. یا در حالت به تاخیر افتادن حل مساله، می‌توان مشکل را به شخصی دیگر انتساب داد.

ب) طراحی گردش کاری درخواست ارتقاء در یک شرکت

مراحل درخواست ارتقاء شغلی را در یک سازمان فرضی، در ذیل مشاهده می‌کنید:

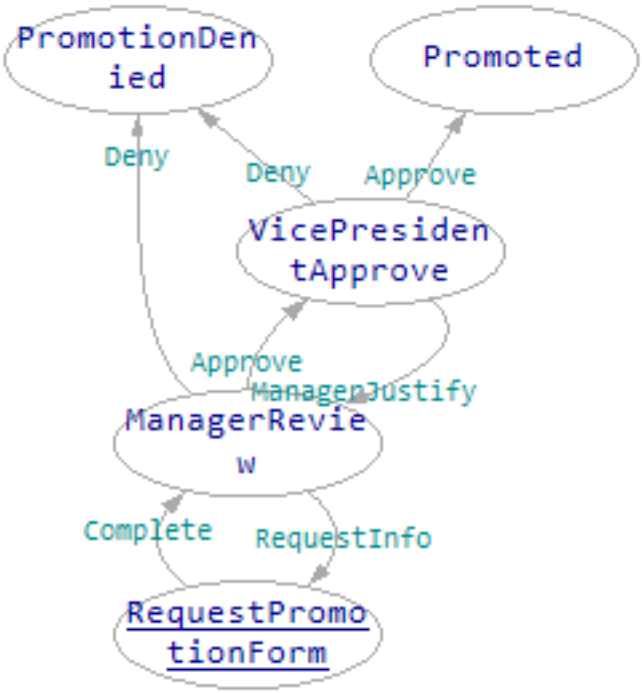
```
<statemachine xmlns="http://statelessdesigner.codeplex.com/Schema">
  <settings>
    <itemname>RequestPromotionStateMachine</itemname>
    <namespace>StatelessTests</namespace>
    <class>public</class>
  </settings>
  <triggers>
    <trigger>Complete</trigger>
    <trigger>RequestInfo</trigger>
    <trigger>Deny</trigger>
    <trigger>Approve</trigger>
    <trigger>ManagerJustify</trigger>
  </triggers>
  <states>
    <state start="yes">RequestPromotionForm</state>
    <state>ManagerReview</state>
    <state>PromotionDenied</state>
    <state>VicePresidentApprove</state>
    <state>Promoted</state>
  </states>
  <transitions>
    <transition trigger="Complete" from="RequestPromotionForm" to="ManagerReview" />

    <transition trigger="RequestInfo" from="ManagerReview" to="RequestPromotionForm" />
    <transition trigger="Deny" from="ManagerReview" to="PromotionDenied" />
    <transition trigger="Approve" from="ManagerReview" to="VicePresidentApprove" />

    <transition trigger="ManagerJustify" from="VicePresidentApprove" to="ManagerReview" />
    <transition trigger="Deny" from="VicePresidentApprove" to="PromotionDenied" />
    <transition trigger="Approve" from="VicePresidentApprove" to="Promoted" />
  </transitions>
```

```
</statemachine>
```

با گرافی معادل:



توضیحات:
کارمند فرم درخواست ارتقاء را تکمیل می‌کند. این فرم به مسئول او ارسال می‌شود. مسئول می‌تواند درخواست را یک ضرب رد کند؛ یا تائید کند که سپس برای مدیرعامل شرکت ارسال می‌شود و یا مجددا به شخص برای تکمیل نواقص فرم ارجاع دهد.
مدیرعامل شرکت می‌تواند درخواست را تائید کند که در اینجا کار خاتمه می‌یابد و شخص ارتقاء خواهد یافت. یا می‌تواند درخواست را رد کند و یا برای بررسی بیشتر مجددا به مسئول شخص ارجاع دهد.

تمرین! توضیحات زیر را تبدیل به یک State machine کنید!

چند سال قبل به اداره‌ی بیمه تامین اجتماعی منطقه مراجعه کردم. جهت دریافت ریز سوابق و انتقال آن‌ها به این مرکز ابتدا یک برگه دریافت شد. پر شد، بعد به صورت دستی (توسط بنده) به یک نفر دیگر ارجاع شد تا امضاء کند. سپس به صورت دستی به مسئول قسمت ارجاع شد تا امضاء کند. مجددا به صورت دستی به مدیر کل مجموعه ارجاع شد تا امضاء کند. سپس به صورت دستی به دبیرخانه برای پیگیری ارجاع شد. قرار است ظرف یک ماه تا 45 روز این سوابق از یک واحد دیگر به این واحد منتقل شوند!
بعد از 45 روز:
مراجعه به دبیرخانه: دریافت شماره پرونده رسیده.
گفته شد که به قسمت دریافت شماره مراجعه کنید. مراجعه شد، گفتند برو پرونده‌ات را بگیر بیار. رفتم زیر زمین، گفت که ما اینطوری پرونده نمی‌دیم! برو فرمش رو هم پر کن بیار. مراجعه شد به کارمند مربوطه، ایشان پس از مشورت با سایر همکاران به این نتیجه رسیدند که در این مرحله نیازی به مراجعه به زیر زمین نبوده! و باید به قسمت ثبت نام مجدد مراجعه کنید! چشم!
اینجا هم مجددا فرم پر شد،ارجاع داده شده به معاون قسمت، امضاء کرد گفت برو دبیرخانه شماره بگیر. شماره گرفته شده بعد مجددا به همان قسمت ثبت نام مراجعه کردم، گفتند برو پرونده‌ات را از زیر زمین بگیر بیار! بعد از آوردن پرونده، ارجاع شد به

صورت دستی به یک قسمت دیگر که سوابق وارد سیستم شود (هنوز نشده بود!). بعد از ثبت (نیم ساعت یا بیشتر ...)، مجددا به همان قسمت ثبت نام مراجعه کردم، گفت حالا برو یک شماره بگیر بیار. شماره گرفته شد از قسمتی دیگر و مراجعه مجدد به قسمت ثبت نام، یک نامه دیگر تهیه کرد، به سه نفر دیگر + دبیرخانه برای امضاء و شماره گرفتن ارجاع داده شد. اینجا تمام شد. بعد این موارد ارجاع شد به قسمت دیگری از شهر برای دریافت قبض پرداخت بیمه.

مطلب مرتبط

نظرات خوانندگان

نویسنده: javad
تاریخ: ۱۳۹۱/۱۰/۱۴ ۱۰:۳۷

با سلام؛
میخواستم بدانم مورد استفاده این طراحی گردش کاری کجاست؟
آیا می‌خواهیم یک سری نمودار رسم کنیم تا بصورت خودکار کتابخانه مربوطه کدهای مورد نیاز مار را برای استفاده در برنامه تولید کند؟
یا اینکه با استفاده از کتابخانه مربوطه می‌خواهیم با استفاده از یکسری کد به نمودار لازم برسیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۱۴ ۱۲:۲۲

- مورد استفاده در هر شرکتی با بیش از یک نفر کارمند.
مانند گردش کاری درخواست:

مساعده
مرخصی
ماموریت
تائید ساعات کاری
درخواست و تائید تشکیل یک جلسه

پر کردن و تائید تایم شیت
و ... تمام کارهای یک سازمان یا شرکت
- هدف آشنایی بصری شما با نحوه حل مسایل چند مرحله‌ای که در ابتدا ساده به نظر می‌رسند، اما 10 مرحله که به آن اضافه شود، مدیریت آن به روش‌های متداول طاقت فرسا خواهد شد.
- وجود این تصاویر، تولید کننده کد و امثال آن صرفا برای ساده کردن توضیح انبوهی متن، به همراه روشی برای حل آن بود.