Dokumentation des Assistantmoduls "External Communication"

FKE

6. November 2018

Inhaltsverzeichnis

1 Beschreibung

External Communication dient als Komponente Senden und Empfangen von SignalR Nachrichten. Der XClient als die Schnittstelle zum Benutzer, verbindet sich via SignalR auf den von External Communication bereitgestellten Endpunkt. Über die bereitgestellten Endpunkte ist es dem Benutzer möglich, die KPUs in einer generischen, modularen Weise zu steuern, je nach dem welche Schnittstellen, Modelle und Ansichten diese selbst bereitstellen. Dazu kommuniziert External Communication aktiv mit den folgenden Diensten:

- Core Service
  - Activ: Interaktion mit den ‚dahinter' liegenden KPUs
  - Passiv: Rückgabe von angefragten KPU-Packages die zum ‚Anzeigen einer KPU' im Client benötigt werden
- Security Service
  - Aktiv: zur Authentifizierung des Benutzers und zur immanenten Berechtigungsüberprüfung bei jeder Benutzeranfrage
- Presenter Service
  - Aktiv: Anmeldung einer Verbindung für die Model Updates eines bestimmten Models
  - Passiv: Entgegennahme und Weitergabe von paketierten Model Updates an Clients

2 Verwendung

External Communication stellt die einzige Komponente dar, die von außerhalb des Systems direkt angesprochen werden kann. Als Technologie für die Kommunikation kommt dabei SignalR Core zum Einsatz, welches netstandard-kompatibel ist und sowohl in .Net Framework, als auch .Net Core Applikationen eingesetzt werden kann. Die Schnittstellen Client -> Server und Server -> Client, die für External Communication zum Einsatz kommen, sind im Projekt „AssistantViewerInterfaces" der Assistant-Solution definiert.

- - IAssistantViewer: Dies ist die Schnittstellendefinition für die Methoden, die im Client zur Verfügung stehen müssen. Zum tatsächlichen Ansprechen ist etwas mehr nötig, als nur das Interface zu implementieren. Dazu in Folge mehr.

- - IExternalCommunication: Dies ist das Interface für Methodenaufrufe mit denen ein Client beim Server Anfragen absetzt. Die wichtigsten Methoden hierbei sind:
  - Login: Anmelden eines registrierten Benutzers
  - RequestExecute:: Ausführen einer Aktion bei einer KPU
  - ReceiveViewRequest: Anfrage für ein Paket zur Instanziierung im Client für die spätere Anzeige einer KPU.
  - ReceiveSubscribeRequest / ReceiveUnsubscribeRequest:Anfrage für die Registrierung auf Model Updates einer bestimmten KPU

Um im Client „Nachrichten“ vom Server zu erhalten ist es nötig, einerseits die entsprechenden Methoden zu implementieren, andererseits diese dem SignalR-Clientobjekt bekannt zu geben. In Listing 1 wird gezeigt, wie man in einer selbst geschriebenen Klasse „SignalRClient“ die entsprechenden Konfiguration und Registrierung vornimmt.

```
1  // missing code
2  using Microsoft.AspNetCore.SignalR.Client;
3  // missing code
4  class SignalRClient
5  {
6  private HubConnection Hub;
7
8  public async Task CreateConnection()
9  {
10 var connectionString = "http://127.0.0.1:4242/ecom";
11 Hub = new HubConnectionBuilder()
12 .WithUrl(connectionString)
13 .Build();
14 RegisterMethods();
15 await Hub?.StartAsync();
16 }
17 private void RegisterMethods()
18 {
19 Hub?.On(nameof(OnReceiveBadLogin), async () => await
       OnReceiveBadLogin());
20 Hub?.On(nameof(OnReceiveGoodLogin), async () => await
       OnReceiveGoodLogin());
21 Hub?.On(nameof(OnBatchDataReceived), async (string viewId,
       string[] items, string[] data, DateTime[] dateTimes) => await
        OnBatchDataReceived(viewId, items, data, dateTimes));
22 Hub?.On(nameof(OnMessageReceived), async (string message, string
       [] buttons) => await OnMessageReceived(message, buttons));
23 Hub?.On(nameof(OnPackageReceived), async (string viewId, string
       data) => await OnPackageReceived(viewId, data));
24 Hub?.On(nameof(OnDisconnect), async () => await OnDisconnect());
25 }
26 // missing code
27 }
```

Listing 1: SignalR Hub creation

In diesem Beispiel werden zwei Methoden des SignalRClient gezeigt, „Connect“ und „RegisterMethods“. „Connect“ versucht dabei, eine Verbindung zu einem SignalR Server aufzubauen und speichert das Verbindungsobjekt über das mit dem Server kommuniziert werden kann im Property „Hub“. Mit „RegisterMethods“ wird dem Hubobjekt mitgeteilt, wie es mit vom Server hereinkommenden Anfragen umgehen soll. Die Anfragen bei SignalR entsprechen in beiden Richtungen Methodenaufrufen denen Parameter mitgegeben werden können, die Daten dafür werden allerdings nur serialisiert übertragen. D.h., dass der Methodenaufruf vom Server beim Client für die Methode „OnReceiveGoodLogin“, effektiv nur den Namen der Methode als Text überträgt (vereinfacht gesagt aber für das Verständnis der Thematik hilfreich). Um zur Laufzeit im Client zu wissen, was mit dem Text vom Server „OnReceiveGoodLogin“ angefangen werden soll, muss dies beim Hub hinterlegt werden, so zu sehen in Listing 2

```
1 Hub?.On(nameof(OnReceiveGoodLogin), async ()=> await
      OnReceiveGoodLogin());
```

Listing 2: SignalR server callback method registration

Die Methode mit dem gleichlautenden Namen wird hier über den lambda-Ausdruck verwendet, um das Verhalten beim hereinkommenden Aufruf abzubilden. Wie die Methode dann aussieht, ist Implementierungsdetail des Clients, denkbar wäre ein Wechsel der Anzeige vom Login-Bildschirm zu einer nenügeführten Darstellung der verfügbaren KPUs. Auch ist es Implementierungsdetail, dass die implementierte Methode genauso heißt wie in der registrierten Schnittstelle, es ist hier allerdings hilfreich um den „nameof“-Operator verwenden zu können der den Methodennamen zur Compile-Zeit durch einen gleichlautenden String ersetzt. Die andere Richtung der Nachrichtenübertragung, also vom Client zum Server, erfolgt ebenfalls über den Hub, wie Listing 3 zeigt.

```
1 public async Task LoginAsync(string user, string password)
2 {
3     await Hub.InvokeAsync("Login", user, password);
4 }
```

Listing 3: SignalR client-side server method invocation

Hier werden lediglich die Parameter Username und Passwort an den Hub für den Aufruf der serverseitigen Methode „Login“ übergeben.

Anm.: Für Methoden mit Rückgabewerten sind Hub.On und Hub.InvokeAsync generisch geschrieben, sodass nahezu beliebige Methoden / Lambdas als Empfänger von verwendet werden können.

3 Installation

External Communication wird auf einem Knoten eines Service Fabric Application – Clusters in der Service Fabric Application "Assistant" installiert.

4 Konfiguration

Zur Konfiguration des Loggings kann die Nlog.xml angepasst werden um dessen Layout bzw. Ziel zu ändern.[1]

5 Abhängigkeiten

Sämtliche anderen Dienste der Service Fabric Application „Assistant" müssen im Cluster mit installiert werden. Die Service Fabric Application „Access Control" muss auf dem Cluster installiert und erreichbar sein. External Communication benötigt folgende Bibliotheken die nicht vom Standard-Nugetserver geladen werden können:

- BreanosConnectors.Kpu.Communication.Common

6 Anforderungen

Zum sinnvollen Einsatz von External Communication muss der Port 4042 des Knotenrechners, auf dem External Communication installiert ist, erreichbar sein. External Communication muss derzeit auf genau einem Knoten installiert werden, da der Dienst als „zustandsloser Dienst" in der Service Fabric Application konzipiert ist und die Logik zur Model Update – Weitergabe vom Presenter Service aus sonst nicht funktionieren würde. Logging kann über NLog mit einer konfigurierbaren Datenbankverbindung erfolgen und benötigt dafür dann ein erreichbares DBMS mit beschreibbarer Datenbank.

[1] https://github.com/nlog/NLog/wiki/Configuration-file

Listings