Foutregistratie in Silverlight business

Fouten belanden niet zomaar bij de applicatiebeheerder

Reinhard Brongers en Loek Duys

Bij het maken van business applicaties op het windows- of webplatform, is het gebruik van logging en exception handling gemeengoed. Door componenten als de Enterprise Library te gebruiken, kan in allerlei loggingbehoeften worden voorzien. Nu ook Silverlight als client platform voor business applicaties in opkomst raakt, is het ook daar belangrijk inzicht te krijgen in het verloop van de applicatieprocessen en welke fouten daarbij zijn opgetreden.

Omdat de code van de applicatie in een browser bij de eindgebruiker uitgevoerd wordt, belanden fouten hierin niet zomaar bij een applicatie beheerder. In dit artikel wordt voor dit probleem een mogelijke oplossing beschreven, aan de hand van een voorbeeld applicatie. Aan het einde wordt kort geschetst hoe deze oplossing kan doorgroeien tot een generiek inzetbare logging faciliteit over processen en applicaties heen.

De voorbeeld applicatie

In dit artikel wordt een webshop als voorbeeld genomen, voorzien van een product catalogus en een winkelmandje. Deze webshop bestaat uit een Silverlight applicatie die gebruik maakt van services om gegevens uit een database te kunnen ophalen, of daarin te kunnen wegschrijven. In Figuur 1 worden de lagen binnen de webshop weergegeven.

- Laag 1 'Webshop Silverlight' bevat de proceslogica, user interface componenten en gegenereerde service proxies. De code hierin wordt uitgevoerd in een browser. Deze laag maakt gebruik van services in laag 2.
- Laag 2 'Webshop Services' bevat de services die de webshop gegevens ontsluit. Deze laag gebruikt de data in laag 3.
- Laag 3 'Webshop Data' bevat alle gegevens die nodig zijn binnen de webshop, zoals het assortiment en de klantgegevens.

FIGUUR 1: WEBSHOP OVERZICHT.

De problemen

Tijdens het winkelen kunnen op de client fouten optreden. Bijvoorbeeld doordat een service tijdelijk niet beschikbaar is door een overbelaste database, of door bugs in de code. Als hiermee niets gedaan wordt, zal de browser de fout aan de eindgebruiker kenbaar maken. In het geval van Internet Explorer, verschijnt bijvoorbeeld een foutmelding icoontje. De eindgebruiker kan waarschijnlijk niets doen om de fout op te lossen en koopt zijn product wellicht ergens anders. De beheerder van de webshop zal de foutmelding niet te zien krijgen. Hierdoor kan ongemerkt omzet verloren gaan.

Ook kan het nodig zijn om fouten in de applicatie die niet leiden tot het afbreken ervan, de zogenaamde warnings, centraal te verzamelen. Tenslotte kan het voor ontwikkelaars handig zijn om in specifieke gevallen het verloop van de applicatie te kunnen traceren.

Een oplossing

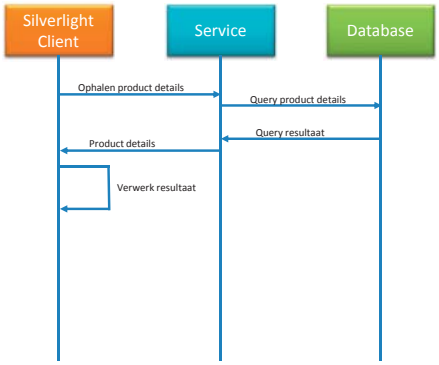
Fouten die optreden tijdens het uitvoeren van code in de browser, moeten dus bekend worden bij de beheerder van de webshop. De enige manier om vanuit Silverlight informatie naar een centrale omgeving te sturen, is via services. Voor foutregistratie zal dus ook een service nodig zijn. De bestaande servicelaag kan met weinig inspanning worden uitgebreid voor dit doel. Vanuit een 'catch block' in de code kan deze service worden aangeroepen. De situatie ziet er dan uit, zoals afgebeeld in Figuur 2. In laag twee is een service toegevoegd en in laag drie een database waarin de fouten centraal worden opgeslagen.

FIGUUR 2: FOUT REGISTRATIESERVICE.

Veel functionaliteit van de webshop wordt uitgevoerd op de client, bijvoorbeeld het samenstellen van een bestelling in de vorm van een winkelmandje. Wanneer onverwacht een fout optreedt tijden het samenstellen van het winkelmandje, kan deze di-

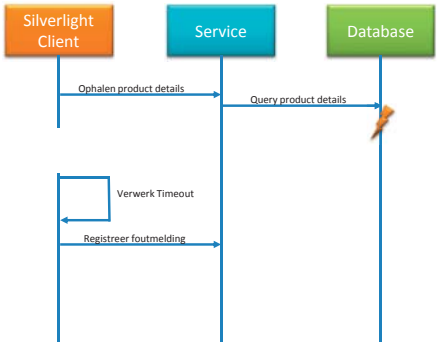
rect naar de registratieservice verstuurd worden.

Een ander veel voorkomend scenario zal bijvoorbeeld het ophalen van product details zijn. Hierbij verstuurt de client een bericht naar de service, de service leest gegevens uit de database en stuurt deze weer terug naar de client. De client verwerkt tenslotte de gegevens, door deze op het scherm weer te geven. Dit proces is weergegeven in Figuur 3.



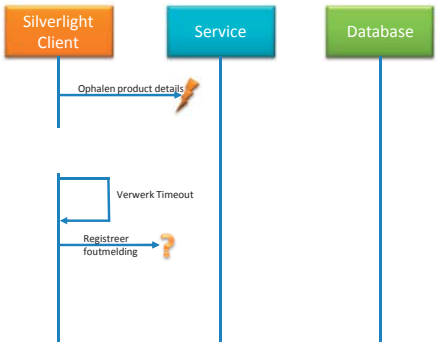
FIGUUR 3: SEQUENCE DIAGRAM SERVICE AANROEP.

Maar tijdens het ophalen van gegevens kan er van alles misgaan, de database kan door omstandigheden overbelast zijn, waardoor nooit of te laat een antwoord retour komt. Dit scenario wordt in Figuur 4 getoond. De Silverlight client zal met dit probleem moeten kunnen omgaan en de fout moeten kunnen verwerken. Hij doet dit door een foutmelding te registreren bij de foutregistratie service.



FIGUUR 4: TIMEOUT BIJ OPHALEN PRODUCT DETAILS.

Tot nu toe zijn we uitgegaan van de situatie dat de services benaderbaar zijn. Maar een andere soort fout die kan optreden is het mislukken van het aanroepen van een service doordat de internetverbinding (tijdelijk) is weggevallen. In dit geval heeft het aanroepen van de foutregistratie service natuurlijk geen zin. Deze situatie is gevisualiseerd in Figuur 5. Er is dus een risico dat foutmeldingen alsnog verloren gaan en dat is onaanvaardbaar.



FIGUUR 5: SERVICE ONBEREIKBAAR.

Om de betrouwbaarheid van foutafhandeling groter te maken, moet het afhandelen van fouten niet direct afhankelijk zijn van de foutregistratie service. Daarvoor maken we een nieuwe component in de Silverlight client, in Figuur 6 is deze terug te vinden onder de naam 'Logging', in laag een. Deze component gaat fouten bufferen en automatisch proberen te verzenden naar de foutregistratie service.
Het bufferen van fouten kan gedaan worden met behulp van een Queue object. Het nadeel hiervan is dat de buffer dan alleen in het geheugen leeft, en verloren gaat bij een 'herstart' van de applicatie.
Om het bufferen betrouwbaarder te maken, is het nodig deze op de harde schijf van de gebruiker te schrijven. Om een Silverlight applicatie gegevens te laten opslaan zonder tussenkomst van de gebruiker, moet 'Isolated Storage' gebruikt worden.

Isolated Storage is beschikbaar in .NET WinForms applicaties, maar is ook de manier in Silverlight om gegevens op de client te bewaren. Het is vergelijkbaar met het cookie mechanisme dat bij websites wordt gebruikt. Het voornaamste verschil met een cookie is dat een willekeurige soort informatie kan worden opgeslagen in de vorm van een dictionary (i.e. key/value pairs). De omvang en beschikbaarheid van de isolated storage kan door de gebruiker zelf per applicatie worden ingesteld en de inhoud kan handmatig worden gewist (maar niet worden bekeken).

FIGUUR 6: SILVERLIGHT LOGGING COMPONENT.

De logging component

Wanneer nu tijdens de uitvoering van de Silverlight applicatie een fout optreedt, wordt deze aan de logging component aangeboden. De ontwikkelaar van de applicatie heeft bepaald of deze fout fataal is, of niet. In het eerste geval zal de applicatie worden afgebroken, in het laatste geval kan het proces doorgaan, en spreken we van een warning. In de logging component kan worden ingesteld of alleen fouten, of ook warnings moeten worden opgeslagen. De opgeslagen meldingen zullen verzonden worden naar de logging service. Wanneer het verzenden niet lukt, moet even gewacht worden voordat het opnieuw geprobeerd wordt. Voor deze geconfigureerde intervallen is een timer aanwezig.
De logging component moet dus geconfigureerd kunnen worden. Het is handig als dit centraal gedaan kan worden. Ook dit bereiken we door de service te gebruiken. Hiermee krijgt elke gebruiker automatisch dezelfde instellingen.

Code van de Logging Service

De service kent een tweetal datacontracten, weergegeven in Listing 1. Er is een contract voor configuratie data en een voor de definitie van foutmeldingen.
De configuratie data bestaat uit een minimaal logniveau, 'MinimumErrorLevel' en een interval dat de wachttijd tussen twee verstuurpogingen specificeert. De foutmelding definitie heeft drie velden. De eerste 'ClientApplication' bevat de naam van de applicatie waarin de fout is opgetreden en gelogd. Door deze als parameter mee te geven, is het mogelijk meerdere applicaties via de service te laten loggen. Het veld 'ErrorLevel' geeft aan welk niveau van toepassing is op de betreffende fout. Het laatste veld, 'Message' bevat de daadwerkelijke foutmelding. In principe is het mogelijk om hier alleen een foutboodschap in op te nemen.

Om zeker te zijn dat alle relevante informatie bij de service belandt, kan dit bericht worden uitgebreid met aparte velden zoals tijdstip van foutmelding, stacktrace, innerexception, etc. In dit voorbeeld is gekozen voor een eenvoudige opzet en wordt de 'ToString()' method van Exception gebruikt voor de volledige foutmelding.

Tenslotte is er nog een Enum aanwezig met de definitie van de beschikbare logniveau's.

```
public enum ErrorLevel
  {
    Critical = 3,
    Error = 2,
    Warning = 1,
    Informational = 0,
  }

  [DataContract]
  public class Configuration
  {
    [DataMember]
    public ErrorLevel MinimumErrorLevel {
get; set; }

    [DataMember]
    public int SendErrorIntervalSeconds {
get; set; }
  }

  [DataContract]
  public class ExceptionMessage
  {
    [DataMember]
    public string ClientApplication { get;
set; }

    [DataMember]
    public ErrorLevel ErrorLevel { get;
set; }

    [DataMember]
    public string Message { get; set; }
  }
```

LISTING 1: LOGGINGSERVICE: DATACONTRACTS.

Het servicecontract, Listing 2, bestaat uit het kunnen ophalen van de gewenste configuratie en het kunnen versturen van een lijst van meldingen. De service implementatie zou de gemelde fouten weg kunnen schrijven met bijvoorbeeld het Exception Handling building block van Enterprise Library.

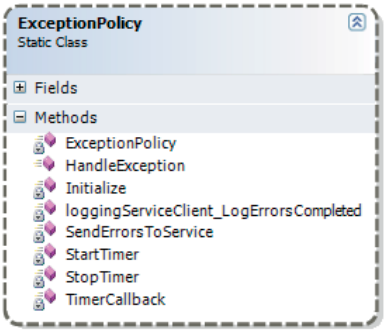
```
[ServiceContract]
 public interface ILoggingService
 {
   [OperationContract]
   Configuration GetConfiguration();

   [OperationContract]
   bool LogErrors(List<ExceptionMessage>
messages);
 }
```

LISTING 2: LOGGINGSERVICE: SERVICECONTRACT.

Code van de Silverlight Logging Component

De Silverlight kant is interessanter. Hier zijn de types uit de service definitie ook aanwezig, als onderdeel van de gegeneerde proxy. Het hart van de logging component bestaat uit de ExceptionPolicy class, die door de gehele webshop gebruikt wordt om de fouten aan te leveren. De functies van deze static class staan in Figuur 7.



FIGUUR 7: DE EXCEPTIONPOLICY CLASS.

De static initializer in Listing 3 zorgt ervoor dat de logging service client wordt geïnstantieerd, en dat de configuratie wordt opgehaald en bewaard. Tenslotte wordt de timer gestart, zodat niet verzonden meldingen van de vorige keer dat de applicatie werd gebruikt, verstuurd zullen worden.

```
static ExceptionPolicy()
{
  Initialize();
}

private static void Initialize()
{
  loggingServiceClient = new LoggingSer-
viceClient();
  store = new Storage();
  configurationManager = new Configurati-
```

```
onManager();
  StartTimer();
}
```

LISTING 3: EXCEPTIONPOLICY INITIALISATIE.

```
public static void HandleException(string
applicationName,
ErrorLevel level, Exception exception)
{
  ExceptionMessage em = new ExceptionMes-
sage
  {
    ClientApplication = applicationName,
    ErrorLevel = level,
    Message = exception.ToString(),
  };
  store.AddError(em);
  StartTimer();
}
```

LISTING 4: EXCEPTIONPOLICY.HANDLEEXPTION().

De enige publieke methode is HandleException, zie Listing 4. Hieraan wordt de opgetreden fout meegegeven, samen met het door de ontwikkelaar vastgestelde foutniveau en de applicatienaam. Deze parameters worden vertaald naar het door de service verwachte type (ExceptionMessage). Op de exception wordt ToString() aangeroepen, waardoor foutinformatie (de tekstuele message en de stacktrace) wordt overgenomen. Na iedere toevoeging van een ExceptionMessage aan de store, wordt de timer weer gestart. Dit zorgt ervoor dat de melding naar de service wordt verstuurd.

Een bericht wordt dus direct naar de service gestuurd. Om berichtverkeer te verminderen zouden berichten kunnen worden opgespaard in de store. Het nadeel hiervan is dat de applicatie mogelijk snel na een fout wordt afgesloten en zodoende een belangrijke foutmelding (voorlopig) niet bij de service belandt. Daarom is dit opsparen slechts voorbehouden aan die situaties waarbij versturen onmogelijk is.

```
private static void TimerCallback(object
state)
{
  StopTimer();
  SendErrorsToService();
}

private static void SendErrorsToService()
{
  try
  {
var errorMessages = new
ObservableCollection<ExceptionMessage>();
   foreach (var item in store.Errors)
   {
     errorMessages.Add(item);
   }

loggingServiceClient.LogErrorsCompleted +=
new
EventHandler<LogErrorsCompletedEventArgs>(
loggingServiceClient_LogErrorsCompleted);
loggingServiceClient.
LogErrorsAsync(errorMessages,
  errorMessages);
  }
  catch
  {
    StartTimer();
  }
}

private static void loggingServiceClient_
LogErrorsCompleted(object sender,
  LogErrorsCompletedEventArgs e)
{
  if (e.Error != null)
  {
    StartTimer();
  }
  else
  {
store.ClearErrors((IEnumerable<ExceptionMe
ssage>)e.UserState);
  }
}
```

LISTING 5: EXCEPTIONPOLICY: ERRORS VERZENDEN.

Wanneer de timer afgaat, worden opgeslagen meldingen daadwerkelijk verstuurd. De code hiervoor staat in Listing 5. Service calls zijn in Silverlight altijd asynchroon, dus de gebruiker hoeft nooit te wachten op dit proces. Wanneer het verzenden niet lukt, wordt de timer weer gestart om het later nog eens te proberen. Het starten en stoppen van de timer is wat ingewikkelder dan een simpel Start/Stop mechanisme. Daarom zijn hiervoor twee helper methodes aanwezig in Listing 6.

```
private static void StartTimer()
{
if (timer== null)
  {
    timer = new Timer(TimerCallback, null,
TimeSpan.FromSeconds(0),
  TimeSpan.FromSeconds(
configurationManager.Configuration.SendEr-
rorIntervalSeconds
  ));
  }
  else
  {
    timer.Change(TimeSpan.FromSeconds(0),
TimeSpan.FromSeconds(
configurationManager.Configuration.SendEr-
rorIntervalSeconds));
}
}

private static void StopTimer()
{
  if (timer != null)
  {
    timer.Change(Timeout.Infinite, Time-
out.Infinite);
  }
}
```

LISTING 6: EXCEPTIONPOLICY: TIMER START EN STOP.

Geavanceerde features

Zoeken

De logging service bevat al de applicatienaam om voor verschillende applicaties te kunnen loggen. Maar hoe kan de service onderscheid maken tussen 'instanties' van deze applicatie? Met andere woorden, hoe kan de webshop beheerder de fouten van een specifieke gebruiker of sessie terugvinden?

Voor deze vraag kan de service simpelweg naar het ip-adres van de aanvrager kijken. Wanneer gebruikers geauthenticeerd worden bij de service, kan ook zijn gebruikersnaam en evt. aanvullende informatie worden bepaald.

Ketenlogging

Wanneer het ExceptionMessage object wordt uitgebreid met extra velden, kunnen deze extra mogelijkheden bieden bij het maken van overzichten. Een mooi voorbeeld hiervan is het toevoegen van attributen (of nog beter: een SOAP Header) waardoor het mogelijk wordt om het loggen van meldingen van verschillende onderdelen of services te combineren tot een soort 'super-trace'. We noemen dit ketenlogging. De attributen hebben dan dezelfde waarde voor verschillende onderdelen die gelogd hebben tijdens het proces. Deze onafhankelijke logacties kunnen op die manier gecorreleerd worden tot een geheel. Dit maakt het mogelijk om bijvoorbeeld proces timing meetbaar te maken.

Client-side logviewer

Voor de ontwikkelaar is het handig als er live in de isolated storage gekeken kan worden welke meldingen klaarstaan voor verzending. Vooral als het synchroniseren mislukt, is een dergelijke functionaliteit redelijk onmisbaar. Op de logcomponent kan daarvoor een collectie worden ontsloten, die simpelweg in Silverlight als datasource van een datagrid wordt gezet.

Reinhard Brongers, is werkzaam als architect en technisch consultant bij VX Company. In die hoedanigheid geeft hij onder meer advies over het opzetten van (SO) architecturen en het inrichten van ontwikkelstraten en –processen. Hij is te bereiken via rbrongers@vxcompany.com.

Loek Duys, werkt als .Net architect en ontwikkelaar bij VX Company. Hij is te bereiken via lduys@vxcompany.com.