Aanwezig: Broes, Peter, Bart, Tommy, Johan
Verontschuldigd:

MVC versus webforms

Toevoegingen aan het overzichtsdocument

- Peter voegt toe dat webforms oorspronkelijk bedoeld waren voor 'windowsformsprogrammeurs', zodat die hun manier van werken konden blijven gebruiken voor webapplicaties.
- Johan hecht veel belang aan 'separation of concerns'. Bij MVC heb je dat out of the box. Als je dat voor webforms wil doen, dan moet je zelf iets a la MVP implementeren.
- We zijn het erover eens dat de code voor de views van MVC niet elegant is.
- Tommy merkt op dat je met webforms heel snel weg bent, maar dat je naarmate je project complexer wordt, je tegen de beperkingen stoot. Bij MVC is de leercurve dan weer een pak steiler.
- Om te vermijden dat een webformsapplicatie onbeheersbaar wordt, is het belangrijk de page life cycle en de event handling ('bubbling') goed te begrijpen.
- Johan heeft voor de huidige applicatie met MVC gewerkt, en vond dat een aangename manier van werken.

Verschillen in 'flow'

Bijvoorbeeld: wijzigen van gegevens

- MVC
    - Request komt toe
    - Controller haalt gegevens uit backend, en steekt die in het model
    - Vervolgens wordt de view getoond, op basis van het model
    - Gebruiker maakt wijzigingen, en post
    - De controller wordt aangeroepen, en doet de binding (op basis van ingevulde gegevens en hidden fields worden objecten gemaakt)
    - Deze gegevens gaan naar de backend[1]
    - Typisch gebeurt er dan een redirect
    - Er worden opnieuw gegevens opgehaald, en in een model gestoken
    - Een nieuwe view wordt getoond op basis van het model
- Webforms
    - Request komt toe
    - Geen postback, dus de gegevens worden uit de database gehaald
    - Web controls worden 'gebind' aan gegevens
    - Gebruiker maakt wijzigingen, en post
    - Het 'OnClick'-event van de 'postknop' stuurt de gegevens naar de backend
    - Via wat er in de viewstate zit, worden de velden van het formulier automatisch opnieuw opgevuld; de backend hiervoor terug consulteren is niet nodig.

MVC zal vaker de backend aanspreken dan webforms, omdat MVC geen viewstate heeft om gegevens in te stockeren.

Conclusie

We houden het bij MVC, op voorwaarde dat we

- Het op een elegante manier kunnen opvangen als er in de backend iets mis loopt (bijv: de inhoud van reeds ingevulde formvelden moet behouden blijven)
- Het aantal hidden fields nodig voor een webform beperkt kunnen houden

[1] Op de vergadering werd gezegd dat na een post eerst het originele object uit de backend wordt opgevraagd, dan de wijzigingen worden toegepast, en dan het object opnieuw naar de backend gaat. Dat is in de testapplicatie echter niet het geval. (Het uit de database halen van het bestaand object, wijzigen, en opnieuw posten, gebeurt steeds allemaal in de backend, onafhankelijk van het gebruikte UI-pattern.)

State

In de verschillende 'tiers' van onze applicatie kan state bewaard blijven:

- BROWSER
  - Cookies
  - Viewstate
  - Hidden fields
  - URL (niet doen)
  - DOM (niet doen)
- ASP.NET WEBAPP
  - Sessie
  - Cache
  - Application wide state (global.asax; liever niet)
  - Files (niet doen)
- WCF SERVICE
  - Is gebouwd op ASP.NET, dus alle opties boven zijn mogelijk.
- DATABASE
  - De database van de app
  - Temporary tables (sessiegebaseerd, of 'definitief[2]')
  - Globale cursors en variabelen (niet doen)

Nadelen van sessions:

1. Niet type-safe (iedere keer boxing en unboxing)
2. Het is niet duidelijk wie verantwoordelijk is voor welk sessie-object
   a) Je weet niet welke pagina een object in de sessie stak
   b) Het is niet duidelijk wie verantwoordelijk is voor de cleanup
3. Een sessie heeft een timeout (standaard 20 minuten)
4. Wordt in-memory bewaard
5. Apart voor elke user (mogelijk zit er in 10 verschillende sessies dezelfde postcodelijst)

Gebruik liever cache (hiervoor is standaard iets voorzien in .net). Caching kan globaal of op sessieniveau gebeuren. Wanneer een gecachet object 'expiret' is per object instelbaar. Problemen (1), (2a), (3) en (4) blijven. Ivm (3) is het wel zo dat je bij een cache minder gauw vergeet dat het zou kunnen dat een object expiret – van sessieobjecten neem je nogal gauw aan dat ze 'nog wel zullen bestaan'.

Als toch sessies gebruikt worden, dan voor een kleine hoeveelheid informatie die niet op een gemakkelijke manier op te vragen is.

Programmeerdag

Op 22 mei (de dag na Hemelvaart) plannen we tentatief een programmeerdag op Kipdorp[3]. De bedoeling is om niet te discussiëren, maar te 'coden'. Er zal dus wel wat voorbereiding nodig zijn ivm flow, requirements, ui.

Tommy kan deze datum nog niet bevestigen – dat hangt af van zijn vrouw. Maar voorlopig houden we deze datum vrij.

Varia

- De publieke dao-attributen in de workersklasses breken het lagenmodel. Dit moet veranderen.
- Ipv BasicHttpBinding te gebruiken voor de webservice, kunnen we ook NetHttpBinding gebruiken. Dit is wat performanter, maar werkt enkel als alle clients .NET-clients zijn. (Wat tot nader order nog steeds het geval is.) Aks dat achteraf nodig blijkt, kunnen we altijd nog BasicHttpBinding als alternatief aanbieden.
- Om een proxy te genereren voor de webservice zullen we een ChannelFactory gebruiken (ipv svcutil, wat de IDE standaard doet).

Verdere planning

- Johan kijkt hoe we in MVC
  - Errors in de backend na een postback kunnen opvangen

[2] Worden pas verwijderd bij sql server restart

[3] Hans liet al weten dat we broodjes/pizza's/... mogen bestellen op kosten van de firma

    - Het aantal hidden fields in forms kunnen vermijden
- Peter werkt een gedetailleerde use case uit, op basis van de schetsen over de UI die hij in augustus maakte (zie https://develop.chiro.be/trac/cg2/wiki/Gui)
- Tommy stuurt ons documentatie over ChannelFactory