Beknopte uitleg designkeuzes
Iteratie 1: project Software-Ontwerp

Jakob Heirwegh, Martijn Leplae, Thibault van Win en Andreas Hinderyckx

Maart 2021

1 Domeinlaag

Om het design van de domeinlaag uit te bouwen, hebben we ons gebaseerd op het meegeleverde domeinmodel als basis. Hierbij maakten we gebruik van **GRASP-principles** (verder in het document steeds geformatteerd in boldface). Onderstaande uitleg en structuur kan steeds op het meegeleverde design model (`designModel.png`) gevolgd worden. We beginnen onderaan het designmodel en bouwen op naar boven. Het gegeven domeinmodel hebben we aldus uitgebreid door volgende zaken toe te voegen:

- `Document`-klasse:
  Deze klasse staat centraal in onze domeinlaag en gebruiken we als abstracte voorstelling van een document. Een `Document` wordt uniek bepaald door een `URL` en bevat precies één `ContentSpan` die de inhoud omschrijft.

- `ContentSpanBuilder`-klasse:
  Deze klasse erft over van `BrowsrDocumentValidator` en hergberuikt de logica uit deze laatste klasse om tokens die geparset worden, om te zetten naar een boomstructuur van `ContentSpan`'s die aan het `Document`-object wordt gelinkt.

  Wanneer een `Document`-object HTML-code wil inlezen, doet deze beroep op de `ContentSpanBuilder`-klasse. Deze heeft alle kennis van de HTML code die gelezen wordt, en fungeert bijgevolg als **Information Expert** wat betreft het aanmaken van de `ContentSpan`-structuur. Verder is de `ContentSpanBuilder`-klasse een voorbeeld van **Pure Fabrication**: het is een artificiële klasse die geen domeinconcept voorstelt, maar specifiek is aangemaakt om de **koppeling** laag te houden tussen de `Document`-klasse enerzijds en de `HtmlLexer`- en `BrowsrDocumentValidator`-klasse anderzijds. Daarnaast verhoogt het mede de **cohesie** van de `Document`-klasse door de verantwoordlijkheid van het aanmaken van de `ContentSpan`-structuur te verschuiven naar een aparte klasse.

- `UIController`-klasse:
  Deze klasse fungeert als **Controller** waarop de UI-laag beroep kan doen wanneer ze diensten uit de domeinlaag wilt oproepen. De **controller** verlaagt de koppeling tussen de domein- en UI-laag: alle oproepen van de UI-laag naar de domeinlaag moeten via deze klasse gebeuren. Hierdoor kan de UI-laag hergebruikt worden op verschillende domeinlagen.

- `DocumentListener`-Interface:
  Dit is een toepassing van het design patroon *Observer*: bij aanpassingen in het `Document`-object wordt een notification ge-broadcast en worden alle UI-elementen die hierbij actie moeten ondernemen op de hoogte gesteld. Elk UI-object dat hierbij actie moet ondernemen, zal namelijk als `listener` op dit evenement 'geabonneerd' zijn. Dit is op zijn beurt ook een voorbeeld van het **polymorphism**-principle, doordat DocumentListener door twee soorten listeners wordt geïmplementeerd:

  - enerzijds `urlListeners`, en
  - anderzijds `documentListeners`.

  Afhankelijk van het soort listener, zal de subject-methode `contentChanged` (waarop deze listeners geabonneerd zijn) ander gedrag vertonen.

  Ook in het licht van andere GRASP-principles is deze aanpak voordelig, aangezien we op deze manier ook gebruik maken van **indirectie**: de **koppeling** tussen de UI- en domeinlaag wordt verlaagd, waardoor de UI-laag kan losgekoppeld worden en bij verschillende domeinlagen kan hergebruikt worden.

Vervolgens zullen we onze design-keuzes in de UI-laag bespreken.

2 UI-laag

We starten uiterst rechts in de UI-laag, met de klasse `Browsr`.

- `Browsr`-klasse:
  Deze klasse stelt het ganse UI-venster voor dat in de opdracht dezelfde naam Browsr krijgt. Het erft over van de geleverde klasse `CanvasWindow`. Deze klasse voldoet aan het **creator**-principe omwille van twee redenen:

  - ze maakt twee klassen `DocumentArea` en `AddressBar` aan en aggregeert deze klassen doordat ze beide subklassen zijn van `Frame`. `Browsr` heeft immers steeds een connectie naar deze twee `Frame`s.
  - Daarnaast beschikt de klasse `Browsr` ook over de initialiserende data voor het correct aanmaken van deze vorige twee vernoemde klassen: `AddressBar` en `DocumentArea` moeten immers gelinkt worden aan hetzelfde `UIController`-object.

- `Frame`-klasse:
  Deze klasse is een abstracte voorstelling van de sub-onderdelen van het `Browsr`-object. De klasse laat toe om de gemeenschappelijke eigenschappen van de sub-onderdelen van het `Browsr`-object af te zonderen en in verdere uitbreidingen van het project eenvoudig nieuwe sub-onderdleen toe te voegen. Deze klasse vervult het principe van **protected variations**, doordat sub-onderdelen van de Browsr onderling kunnen uitgewisseld worden zonder een invloed te hebben op de gehele structuur van de Browsr.

- `AddressBar`-klasse:
  Dit is een specialisatie van de `Frame`-klasse die de adres-bar van de Browsr voorstelt. Ze staat in verbinding met de domeinlaag via de controller, wat haar in staat stelt om zaken die domeinkennis vereisen zoals het laden van documenten, het doorgeven van gebruikersinvoer e.d. te vervolledigen. In de omgekeerde richting, vloeit er informatie van de domein- naar de UI-laag doordat deze klasse het `DocumentListener`-interface implementeert en zo bij veranderingen van de informatie uit de domeinlaag - bv. de URL bijbehordend bij een nieuw ingeladen `Document` - hier op een gespecialiseerde manier kan op ingaan.

- `DocumentArea`-klasse:
  Dit is eveneens een specialisatie van de `Frame`-klasse die het gedeelte van de Browsr voorstelt waarin de inhoud van documenten moet worden weergegeven. Ze vertoont een gelijkaardige link met de domeinlaag zoals deze bij de bovenvermelde `AddressBar`-klasse.

  Wanneer een document moet gerenderd worden, wordt deze taak doorgeschoven naar de onderliggende sub-onderdleen van `DocumentArea`, namelijk de `DocumentCell` (zie verder). Deze manier van aanpak steunt op het **information expert** principe in die zin dat elk sub-onderdeel verantwoordelijkheid neemt om zichzelf te renderen. Verder komt hier ook het **creator**-principe naar voren: de klasse `DocumentArea` is verantwoordelijk voor het aanmaken van een `DocumentCell` die de juiste structuur bevat.

  Om elementen uit de domeinlaag te kunnen weergeven in de UI-laag, bevat de `DocumentArea`-klasse een link naar een `DocumentCell`-object, wat op zich ook een specialisatie van de `Frame`-klasse is.

- `DocumentCell`-klasse:
  Deze klasse is eveneens een specialisatie van de `Frame`-klasse en fungeert als algemeen object om domeinelementen onafhankelijk van de structuur van de domeinlaag naar UI-elementen te kunnen omzetten om deze te kunnen renderen. Door gebruik te maken van een overkoepelend en abstract UI-element zoals `DocumentCell`, verlagen we de **koppeling** tussen specifiekere UI-elementen (zie onderstannde drie klassen) en `DocumentArea`.

  Verder is deze structuur van `DocumentCell`'s ontworpen volgens het *composite*-designpatroon

- `UIHyperlink`-, `UITextField`- en `UITable`-klassen: dit zijn alledrie specialisaties van de `DocumentCell`-klasse die in staat zijn om de individuele domeinobjecten voor te stellen.

Ten slotte bespreken we de uitwerking van enkele system operations in ons design.

3 System operations

3.1 Document Loading

Wanneer een document moet geladen worden, wordt een `loadDocument()`-oproep gedaan vanuit de UI-laag naar de `UIController`. Deze delegeert de oproep verder naar de `Document`-klasse, die op zijn beurt een beroep doet op de `ContentSpanBuilder` om het document succesvol in een `ContentSpan` om te zetten. Dit triggert een event, namelijk `contentChanged` uit het `DocumentListener`-interface, waardoor de elementen uit de UI-laag die zich hierdoor moeten aanpassen, op de hoogte gebracht worden. Hierdoor wordt zowel de `DocumentArea` als de `AddressBar` op de benodigde manier geüpdated. De nodige listeners om dit te realiseren worden bij de aanmaak van het `UIController`-object in de `Browsr`-klasse ineens mee toegevoegd.

3.2 URL clicking

Wanneer een gebruiker een URL aanklikt, wordt in de `DocumentArea`-klasse uit de UI-laag elke `Frame` overlopen om te bepalen op welke `DocumentCell` deze klik juist was. Het afhandelen van de benodigde actie bij deze klik,

wordt dan gedaan door deze bepaalde `DocumentCell`, die via `DocumentArea` en de controller `UIController` beroep kan doen op de nodige diensten uit de domeinlaag.

3.3 (Malformed) URL handling

Wanneer een gebruiker een URL in de `AddressBar` ingeeft, wordt deze als `String` opgeslagen in de UI-laag en doorgegeven aan de `UIController`, dewelke het eerste object in de domeinlaag is. Pas wanneer de URL hier aankomt, wordt geprobeerd om de URL van een `String` naar een effectief `URL`-object om te zetten. Hierbij kunnen we twee scenario's onderscheiden:

- De URL is correct:
  De `loadDocument`-methode uit `Document` wordt opgeroepen vanuit de controller en deze bepaalt verder welk document moet geladen worden. Dit triggert een `contentChanged()`-event in het `DocumentListener`-interface, waardoor de betrokken UI-elementen worden gealarmeerd en zich naargelang kunnen updaten.

- De URL is malformed:
  In de `UIController` zal een `MalformedURLException` optreden. Deze wordt *defensief* afgehandeld door een nieuw `Document`-object aan te maken met een vastgelegde 'error-URL'. `contentChanged` wordt getriggerd, maar door de vastgelegde error-URL, zullen enkel de `DocumentListeners` en niet de `urlListeners` zichzelf updaten. Hierdoor blijft de URL die door de gebruiker is ingegeven in de `AddressBar` staan, maar kan wel een error-Document worden weergegeven in de `DocumentArea`.