Beknopte uitleg designkeuzes
Iteratie 1 & 2: project Software-Ontwerp

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

Maart 2021

1 Iteratie 1

1.1 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 designpatroon *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.

1.2 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.

1.3 System operations

1.3.1 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.

1.3.2 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.

1.3.3 (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`.

2 Iteratie 2

2.1 Domeinlaag

In de domeinlaag hebben we de klassen `TextInputField`, `SubmitButton` en `Form` toegevoegd zoals gespecifieerd in de opgave.

Verder hebben we een klasse `BookmarksURLKeeper` toegevoegd met als doel om de URL's bijbehorend bij door de gebruiker aangemaakte bookmarks bij te houden. Deze klasse is nodig omdat we niet alle informatie van de bookmarks willen bijhouden in de UI-laag. Om dit probleem op te lossen, hebben we besloten om bij het aanmaken van een bookmark enkel de naam in de UI-laag bij te houden en de bijbehorende URL op te slaan in de domeinlaag, in de klasse `BookmarksURLKeeper`. De motivatie achter deze aanpak zijn de volgende elementen:

- We houden kennis vanuit de domeinlaag zoveel mogelijk gescheiden van de UI-laag. We hebben echter besloten om de namen van de bookmarks wel in de UI-laag bij te houden zodat elke bookmark gelinkt kan worden aan een bijbehorende sectie van de UI om het renderen van deze bookmarks in de `BookmarksBar` (cfr. infra) mogelijk te maken.

- Indien er later extra attributen aan een bookmark moeten kunnen worden toegevoged, zoals bijvoorbeeld functionaliteit voor een beschrijving, commentaar, `tags` etc., kunnen deze eenvoudigweg aan de klasse `BookmakrsURLKeeper` worden toegevoegd zonder dat deze in de UI-laag moeten worden opgeslagen en de **koppeling** tussen UI- en domeinlaag laag blijft.

2.2 UI-laag

- **BookmarksBar**
  Zoals vermeld in de vorige paragraaf, hebben we een klasse genaamd `BookmarksBar` toegevoegd die een grafische versie van de toegevoegde bookmarks weergeeft en van de bookmarks enkel en alleen de ingegeven naam bijhoudt. Indien op een bookmark geklikt wordt, kan dit gedetecteerd worden door het grafische gebied (i.e. een instantie van de klasse `Frame`, zie Iteratie 1) gelinkt van de naam van de bookmark te raadplegen. De bijhorende URL van de bookmark wordt vervolgens uit de domeinlaag opgevraagd via de `UIController`. Om dit idee te

implementeren maken we gebruik van een beperktere vorm van een hyperlink, die enkel de naam maar niet de URL zelf omvat, genaamd een `UITextHyperlink` (cfr. infra).

Deze werkwijze houdt de **koppeling** tussen de domein- en UI-laag zo laag mogelijk en leidt tot een **hoge cohesie** van de `BookmarksBar`-klasse.

- **UI-tegenhangers van de domeinobjecten**
  Om de uitgebreide structuur van de domeinlaag te weerspiegelen in de UI-laag, hebben we volgende tegenhangers van deze nieuwe objecten in de UI-laag toegevoegd:

  - **UItextInputField**
    Dit is het UI-equivalent van het domeinobject `InputField`. Deze klasse omvat alle functionaliteit wat betreft tekst-invoer van de gebruiker en is bijgevolg **information expert** op dit vlak. De klasse `AddressBar` laten we van deze klassen overerven, zodat we **code duplicatie beperken**. Deze zaken leiden ertoe dat de klasse `UITextInputField` een **hoge cohesie** heeft en de klasse `AddressBar` ook zijn hoge cohesie verder behoudt.
  - **UISubmitButton**
    Deze klasse is de tegenhanger van het domeinobject `SubmitButton` en omvat alle functionaliteit wat het grafisch weergeven van knoppen en de benodigde acties die hierbij moeten worden genomen. Om de verschillende grafische weergaves van een knop te implementeren met **minimale koppeling**, hebben we gebruik gemaakt van het *State* designpatroon. Meer specifiek: beide mogelijke toestanden waarin een knop zich kan bevinden zijn geïmplementeerd d.m.v. een overeenkomstige geneste klasse. Zo zijn er geneste klasses `Pressed` en `NotPressed` om de correcte grafische weergave en nodige acties van een druk op een ingedrukte respectievelijk niet-ingedrukte knop weer te geven en af te handelen.
  - **UIForm** Deze klasse is de tegenhanger van het domeinobject `Form`. Net zoals gespecifieerd in de domeinlaag, kan een `UIForm` meerdere `DocumentCell`'s (UI tegenhanger van een `ContentSpan`: zie Iteratie 1) bevatten om de gewenste structuur te verkrijgen. Een `UIForm` is gekenmerkt door een `action` die bij het aanklikken

van een bijhorende `UISubmitButton` specifieert hoe de URL moet worden samengesteld.

- **UITexeHyperlink** Deze klasse gelijkaardig aan de klasse `UIHyperlink` uit Iteratie 1, met de uitzondering dat ze het URL-veld van de hyperlink niet bevat. Deze klasse maakt het mogelijk de hogerop beschreven werking van de domeinklasse `BookmarksURLKeeper` en UI-klasse `BookmarksBar` te implementeren.
- **UIHyperlink** Deze klasse erft over van de klasse `UITextHyperlink` en voegt een URL attribuut toe aan de superklasse. Dit stelt ons in staat om volledige hyperlinks weer te geven en bij te houden in de UI-laag daar waar dit nodig is, zoals in `UIForm`'s en `DocumentCell`s in het algemeen. Op deze manier kunnen we garanderen dat enkel die objecten die volledige kennis moeten hebben over de hyperlinks (zoals `UIForm`'s e.d.) ook effectief deze kennis bevatten, en klassen zoals `BookmarksBar` enkel een minimale subset van deze domeinkennis omvatten.

Ten slotte hebben we een drietal klassen toegevoegd teneinde de `Save` en `Save Bookmark`-functionaliteit te implementeren, namelijk;

- `GenericDialogScreen`
  Dit is een abstracte superklasse om het abstract idee van een dialoogscherm voor te stellen. De klasse bevat alle gedeelde functionaliteit van de klassen `BookmarksDialog` en `SaveDialog` (cfr. infra) om **code duplicatie** tot een minimum te **beperken**.
- **BookmarksDialog**
  Deze klasse stelt het concept van een venster voor dat weergegeven wordt bij het opslaan van een bookmark. Het bevat een `UIForm` met twee knoppen om de gevraagde layout weer te geven. De URL - die zich op het moment dat de toetsencombinatie `Ctrl + d` wordt ingedrukt in de `AddressBar` bevindt - wordt via de link met de klasse `Browsr` van de superklasse `GenericDialogScreen` opgevraagd en ingevuld in het 'URL'-veld uit de `UIForm`.

  Door gebruik van een superklasse te maken, vermijden we dat de klasse `BookmarksDialog` een rechtstreekse link met de klasse `Browsr` moet hebben. Dit probleem lijkt verschoven te worden naar de superklasse, maar dit is onze ogen een meer modulair

design. Hierdoor moeten de connecties met de `Browsr`-klasse namelijk niet voor elke specifieke instantie van een dialoogscherm worden voorzien: in dit geval zou de klasse `Browsr` namelijk kennis zou moeten hebben van de verschillende soorten dialoogschermen. Dit verhoogt de **cohesie** van zowel de klasse `Browsr` als de dialoogscherm-klassen, en verlaagt opnieuw de koppeling tussen deze klassen.

Ten slotte bevat de klasse `BookmarksDialogScreen` een connectie met de klasse `BookmarksBar` om de functionaliteit van het toevoegen van bookmarks te implementeren. Deze associatie **verhoogt de koppeling niet** naar onze mening, doordat de functionaliteit van een `BookmarksDialog` rechtsreeks in verband staat met de `BookmarksBar`, en met geen enkele andere klasse. Bijgevolg modelleert deze associatie modelleert dit idee en veroorzaakt ze geen onnodige, extra koppeling.

– **SaveDialog**
Deze klasse stelt het dialoogscherm voor dat weergegeven wordt wanneer de toetsencombinatie `Ctrl + s` wordt ingeduwd. Ze erft over van de klasse `GenericDialogScreen` en deelt bijgevolg degemeenschappelijke functionaliteit met `BookmarksDialog`. Door deze overervingsstructuur, heeft de klasse `SaveDialog` geen associatie met de klasse `BookmakrsDialog`, wat onnodige **koppeling vermijdt**.

Tot slot hebben we enkele toevoegingen gemaakt aan de klasse `Browsr` tegenover de versie die we hebben ingediend bij Iteratie 1. De voornaamste aanpassing is de toevoeging van geneste klassen die de huidige layout van het programma weergeven, i.e. of er al dan niet een dialoogscherm wordt weergegeven en hoe input afgehandeld moet worden indien dit al dan niet het geval is. Deze geneste klassen noemen we `Layouts`, omdat ze de huidige layout van het programma weergeven; ze zijn dan ook een rechtstreekse implementatie van het *State* designpatroon.

Bij aanvang bevindt `Browsr` zich in een `RegularLayout`. Wanneer de toetsencombinatie `Ctrl + d` of `Ctrl + s` wordt ingeduwd, wordt de layout geüpdated naar respectievelijk een `BookmarksBarLayout` of een `SaveDialogLayout`. Elke layout heeft een eigen implementatie van de uit `CanvasWindow` overgeërfde methoden `handleMouseEvent`, `handleKeyEvent` en `Render`. Op deze manier kunnen we en juiste afhandeling van deze methoden impmlementeren

en houden we ons design modulair wat het eenvoudig maakt om later eventueel extra dialoogschermen toe te voegen: dit vereist namleijk enkel een toevoegingn van een overeenkomstige `Layout` en implementatie van diens overgeërfde methodes.

Daarnaast hebben we ook overwogen om het *memento* design pattern te implementeren om de vorige staten van de `Regular-`, `BookmarksDialog-` of `SaveDialog`-layout te herstellen. Een implementatie van dit designpatroon bracht echter een aanzienlijke hoeveelheid overhead met zich mee vanwege de nood aan zogenoemde 'memento-klassen' voor elk object dat de mogelijkheid moest hebben om hersteld te worden na het wisselen van een layout. Deze overhead vermijden indien we voor elk van deze layouts de benodigde informatie in de hoger vermelde klassen bijhouden en via het gebruik van de zonet vernoemde layouts tussen deze scenario's wisselen vanuit de `Browsr`-klasse. Bovendien gebeurt dit alles met slechts een instantie van de `CanvasWindow`-klasse, zoals ook gevraagd is geweest.