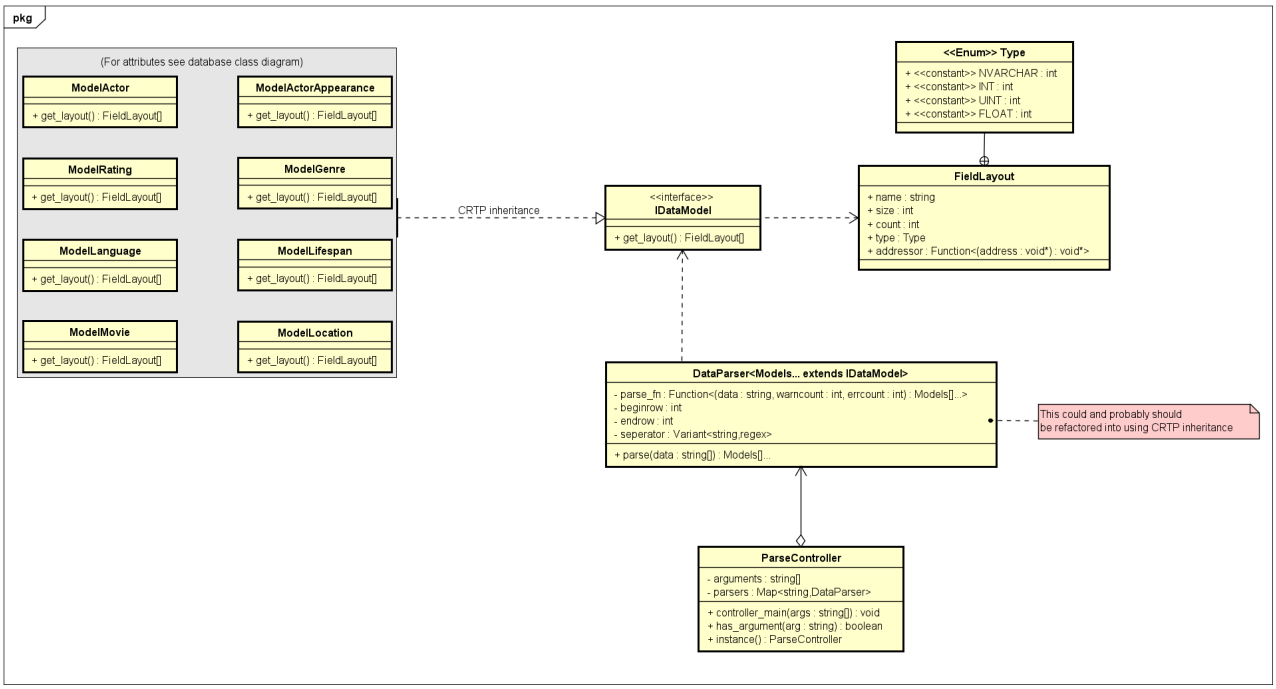
Ontwerp Parser

Voorafgaande aan het maken van de parser, is gekeken naar de layout van de IMDB bestanden. Deze bleken veel verschillende datalayouts te bevatten. Hierdoor is er voor gekozen om een systeem te creëren waarbij voor ieder bestand een aparte parser bestaat:



Iedere parser produceert arrays van een of meer typen models. Iedere modelklasse heeft een descriptor waarmee de inhoud van objecten van die klasse geserialiseerd kan worden. De controller laadt één voor één ieder bestand, roept de parser aan en schrijft het resultaat weer terug als een nieuw bestand.

Gebruikte Design Patterns

Voor de ParseController is gebruik gemaakt van het Singleton pattern: er is maar één controller in het programma. Deze draait de parsers een voor een en stuurt hun uitvoer naar de CSVWriter. Er is een static methode om de instance van deze controller te verkrijgen en de constructor is private gemaakt, zodat er geen extra instances gecreëerd kunnen worden.
De ParseController is tevens een voorbeeld van het Mediator pattern: communicatie tussen de bestandsreader, de parser en de bestandsschrijver gebeuren niet direct via de parser, maar worden gedaan door de controller.

Bij de verschillende typen models is gebruik gemaakt van het Curiously Recurring Template Pattern. (CRTP) Hierbij wordt door via een baseklasse de methoden van de deriverende klasse aan te roepen het implementeren van een interface gegarandeerd, zonder dat hiervoor virtual methoden met runtime overhead moeten worden gebruikt.

Op verschillende plaatsen in de code is std::variant gebruikt. Om een variant object te gebruiken moet een visitor gebruikt worden. Deze pattern is dan ook hier aanwezig.
Daarnaast is de klasse Visitor geschreven. Hiermee kunnen snel nieuwe visitors gegenereerd worden uit een verzameling van callables. (Doorgaans lambdas)

Op verschillende plaatsen is gebruik gemaakt van het Servant pattern. In plaats van utilityklassen zijn echter vrije methoden in een namespace gebruikt. Dit heeft echter in de praktijk hetzelfde effect. Voorbeelden hiervan zijn de methoden in ParserUtils die veelgebruikte operaties op modelklassen implementeren.

Reflectie

Tijdens de initiële analyse van de IMDB bestanden leek het alsof voor ieder bestand een compleet nieuwe parser nodig zou zijn. Na verloop van tijd bleek dat er extra bestanden nodig zouden zijn voor het beantwoorden van de vragen en moesten er extra parsers geschreven worden.
Bij het analyseren van deze bestanden bleek echter dat er toch veel overeenkomst tussen de syntax in de bestanden was: het overgrote merendeel van de bestanden volgde de *movies.list* layout of de *actors.list* layout.
Acteraf gezien was het wellicht handig geweest als er een systeem was geschreven waarbij verschillende deelparsers samengevoegd hadden kunnen worden tot een grotere parser. Vanwege tijdsgebrek is het echter gebleven bij enkele utilitymethodes om de meestvoorkomende secties te parsen.

Daarnaast was het handig geweest om de verschillende parsertypes te implementeren door middel van CRTP inheritance, in plaats van het huidige systeem waarbij function pointers worden gebruikt. De initiële reden hiervoor was het hergebruiken van verschillende parsers voor verschillende bestanden: bij CRTP inheritance kan een concrete klasse niet verder gederiveerd worden, omdat deze geen template parameters meer heeft.
Echter was het wel mogelijk geweest om hiervoor een verdere abstracte klasse te gebruiken. Dit is wederom vanwege tijdsgebrek niet gebeurd.