Afstudeerrapport

Visual Code editing voor Smart Contracts

David Doorn
Afstudeerder
442351@student.saxion.nl

Anthony van den Berg
Afstudeerbegeleider
a.r.vandenberg@saxion.nl

Steven Verkuil
Bedrijfsbegeleider namens Coinversable
steven@coinversable.com

Inhoudsopgave

I. Inleiding

De omgevingswet gaat over onze leefomgeving, en hoe die wordt ingericht. Met de invoering van de omgevingswet worden verschillende plannen, aanvragen en methodes vereenvoudigd. Hiermee komen ook veel verantwoordelijkheden te liggen bij de lokale overheden. Hier bieden verschillende blockchain technieken mogelijkheden voor een simpelere, snellere, en accuratere afhandeling van dit proces.[1]

Ten eerste biedt blockchain technologie een duidelijkere scheiding van verantwoordelijkheden en informatiestromen. Op deze manier is de regelgever (in dit geval de lokale overheid) verantwoordelijk voor het aanleveren van de regelgeving waar de afstudeeropdracht zich op richt, maar kan de verdere informatie die wordt gebruikt in die regelgeving uit andere bronnen worden gehaald.

Daarnaast is er mogelijkheid om historische gegevens makkelijk in kaart te brengen dankzij de persistentie van informatie op de blockchain. Zo kan er, indien er twijfels zijn over waarom bepaalde aanvragen op een bepaalde wijze zijn afgehandeld, altijd gekeken worden wat de regelgeving op het moment van afhandeling was.

Ten slotte biedt de blockchain meer consistentie dan de handmatige afhandeling van een aanvraag. Omdat er gebruik wordt gemaakt van Smart Contracts kan er met het uitsluiten van menselijke fouten gegarandeerd worden dat gelijkwaardige aanvragen altijd op gelijke wijze worden afgehandeld, waardoor er significant minder kans is op onterechte goedkeuring of afwijzing van een aanvraag, en is er meer duidelijkheid over waar dat door komt.

In dit afstudeerproject wordt er gekeken naar de verschillende mogelijkheden om een op blockchain technologieën gebaseerd systeem toegankelijker te maken voor regelgevende partijen die daarmee gemakkelijker Smart Contracts kunnen genereren zonder daarbij programmeerkennis nodig te hebben. Hierbij wordt er aangesloten op een prototype dat reeds door Coinversable ontwikkeld is, waarbij een aanvraag kan worden opgesteld en aan de hand van bestaande Smart Contracts kan worden bepaald of die aanvraag kan worden goedgekeurd. Het opstellen van de aanvraag vindt plaats door middel van het tekenen van het bijgebouw op de kaart, en het bijvoegen van aanvullende gegevens (zoals de bouw- en goothoogte van het bijgebouw).

[1]Meer informatie over de toepassing van blockchain en de omgevingswet staat in een blogpost op de website van Coinversable

II. Opdrachtgever

Coinversable werkt in opdracht van de gemeente Deventer aan een oplossing om aanvraagprocessen rondom de Omgevingswet te versimpelen en versnellen.

Coinversable is een softwarebedrijf uit Deventer dat zich in eerste instantie primair richtte op het vernieuwen van bestaande processen met behulp van blockchain technologieën door het bouwen van maatwerkoplossingen voor onder andere de onderwijsbranche. Tegenwoordig zijn daar meer algemene projecten aan toegevoegd.

Het is een relatief klein bedrijf met 10-15 medewerkers die zich bezighouden met verschillende projecten en disciplines. Toch wordt er onderling veel samengewerkt en gebruik gemaakt van elkaars expertises.

Voor de blockchain-toepassingen die Coinversable voor haar klanten ontwikkelt wordt gebruik gemaakt van het zelfontwikkelde Validana [1] framework. Met behulp van Validana kunnen klanten profiteren van bijvoorbeeld de schaalbaarheid en betrouwbaarheid van de blockchain.

III. Opdrachtomschrijving

De opdracht omvat het onderzoeken van mogelijke benaderingen om een visuele editor te bouwen voor Smart Contracts die zich richt op niet-programmeurs, en het realiseren van een prototype gebaseerd op de uitkomsten van dit onderzoek.

Probleemstelling

Smart Contracts voor de Validana-Blockchain worden geschreven in TypeScript [2]. Voor ontwikkelaars binnen Coinversable en voor andere technisch onderlegde gebruikers van Validana is dit erg handig, maar voor gebruikers zonder kennis van programmeren is dit een erg ontoegankelijk systeem.

Een voorbeeld van een situatie waarin het onhandig is dat contracten geschreven zijn in TypeScript is de casus van de omgevingswet. Coinversable werkt aan een systeem waarin aanvragen met betrekking tot de omgevingswet deels automatisch kunnen worden afgehandeld aan de hand van Smart Contracts. Het is echter onwenselijk om alle wetgeving en beleidsplannen door programmeurs in contracten om te laten zetten, want indien er iets verandert zou dit betekenen dat het niet direct door een jurist kan worden aangepast.

Doelstelling

Coinversable wil graag een platform ontwikkelen waarbij het schrijven van Smart Contracts meer toegankelijk wordt gemaakt voor gebruikers met weinig tot geen kennis van programmeren. Dit platform moet het mogelijk maken om in een grafische omgeving regels in elkaar te klikken, die omgezet worden in TypeScript-contracten en naar een Validana-blockchain kunnen worden geüpload.

Projectgrenzen

Het project wordt afgebakend aan de hand van een aantal projectgrenzen. Deze grenzen geven aan welke eisen er aan het afstudeerproduct worden gesteld en aan welke gebieden extra aandacht moet worden besteed.

- Het afstudeerproduct bevat een onderzoek, een specificatie en een prototype
- Het onderzoek richt zich op het analyseren van verschillende benaderingen voor het prototype, en hun toepasbaarheid voor de situatie van Coinversable
- Hoewel het afstudeerproduct de basis moet vormen van een mogelijke implementatie voor verschillende casussen beperkt de opdracht zich tot de omgevingswet. Wel worden er enkele voorstellen gedaan over de verbreding van de applicatie naar meerdere casussen
- De afstudeerperiode loopt van 8 februari 2021 tot en met 2 juli 2021
- Het prototype is in staat om in TypeScript geschreven Smart Contracts voor de Validana-blockchain te genereren
- Het prototype wordt waar mogelijk gebouwd met technologieën en frameworks die reeds binnen het bedrijf worden toegepast

IV. Uitvoering

Fasen

Het afstudeerproject kan in 4 fasen worden opgesplitst:

Onderzoek

Als onderdeel van dit onderzoek wordt er een vooronderzoek gedaan naar welke opties er in het onderzoek kunnen worden meegenomen. Hiervoor wordt er gezocht naar bestaande low-code toepassingen en modelleertechnieken, en wordt er van elke mogelijke oplossing beoordeeld of het zinvol is om hier uitgebreider onderzoek naar te doen.

In de tweede fase van het onderzoek wordt er bij elk van de oplossingen die uit het eerste deel komen een korte pilotproef gedaan, waarin er een set aan regels uit de omgevingswet wordt genomen[2] en wordt gepoogd om deze regels te implementeren waarbij gebruik wordt gemaakt van die oplossing. Deze implementatie is puur visueel en hoeft geen code te genereren, maar geeft een duidelijk beeld van hoe een afstudeerproduct gebaseerd op die oplossing eruit zou komen te zien. Deze implementaties worden intern behandeld en besproken, en één van de stakeholders.

Tijdens het onderzoek wordt er gekeken naar de verschillende low code oplossingen die reeds worden toegepast in vergelijkbare situaties. Uit het onderzoek volgt de keuze voor een bepaalde benadering of toepassing. Hierbij hoort ook een specificatie die omschrijft welke concepten die in het onderzoek naar voren zijn gekomen worden toegepast in het afstudeerproduct. Ten tijde van het onderzoek kent het afstudeerproduct nog geen concrete requirements, anders dan dat het afstudeerproduct zelfstandig door de beoogde gebruikersgroep moet kunnen worden bediend, en dat het afstudeerproduct bruikbaar moet zijn in combinatie met andere binnen het bedrijf bestaande projecten.

Specificatie

In de specificatiefase wordt er een specificatie opgesteld die omschrijft hoe het systeem moet worden geïmplementeerd. Door een specificatie aan te houden kan er al voorafgaand aan de realisatiefase worden bepaald hoe het project er uit moet komen te zien en welke functionaliteiten er moeten worden geboden. Ook biedt de specificatie richtlijnen voor verdere uitbreiding van het prototype of andere, op het afstudeerproduct gebaseerde oplossingen.

De specificatiefase ligt tussen de realisatie- en onderzoeksfase in, maar overlapt ook met beiden. Zo wordt er tijdens de onderzoeksfase al inspiratie opgedaan voor de specificatie, maar kunnen richtlijnen tijdens de implementatie nog worden toegevoegd of aangepast.

Realisatie

Tijdens de realisatiefase zal er op basis van de conclusie van de onderzoeksfase en de opgestelde specificatie een afstudeerproduct worden gebouwd. Afhankelijk van de uitkomst van het onderzoek zal er een systeem worden gebouwd dat gericht is op makkelijke integratie met de Validana-blockchain en dat gebruik maakt van dezelfde technieken en frameworks als andere projecten in het bedrijf, zodat er eventueel op het prototype kan worden doorontwikkeld. Hierbij speelt ook de documentatie van het project een grote rol: het project moet overdraagbaar worden opgeleverd, zodat (onderdelen ervan) het product kan worden gebruikt in andere projecten.

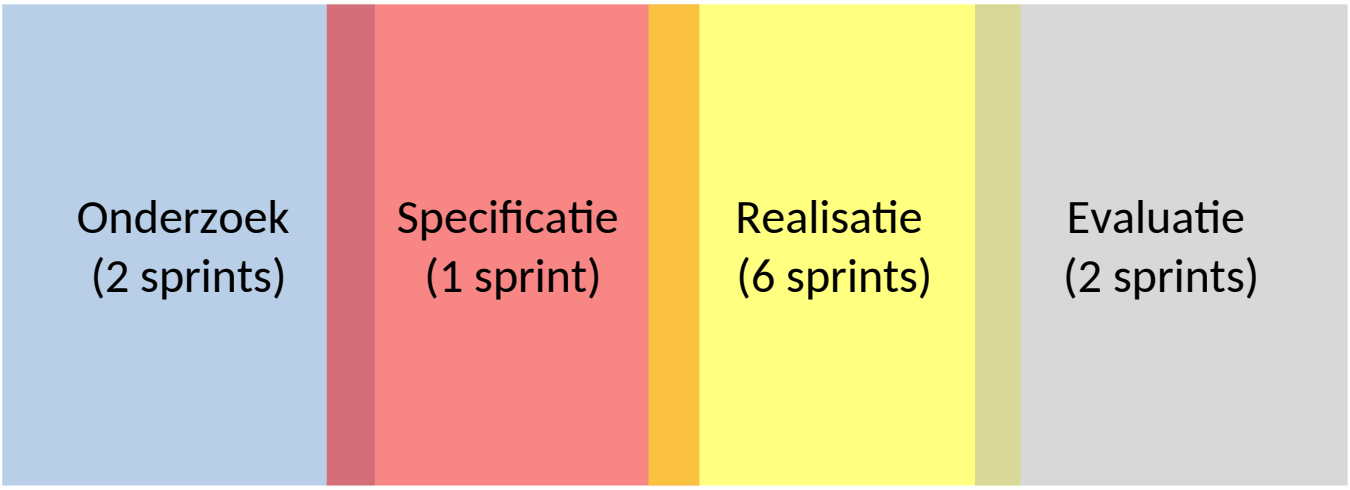
[2] Zie ook Bijlage A

Evaluatie

Gedurende de evaluatiefase wordt er in samenwerking met testpersonen gekeken naar de uitkomst van het afstudeerproject en welke mogelijke verbeteringen er nog kunnen worden gedaan. Zo wordt er gekeken naar de gebruiksvriendelijkheid van het prototype voor niet-programmeurs, en wordt er overwogen welke functionaliteiten van belang zijn voor een volledige afronding van het systeem. Hierbij kunnen er helaas geen medewerkers van de gemeente als testpersoon fungeren, maar zal er getest worden met niet-programmeurs binnen het bedrijf.

Planning

Tijdens dit project zal worden gewerkt volgens de SCRUM-methode. Binnen Coinversable is het gebruikelijk om te werken in sprints van twee weken. Het afstudeerproject zal verlopen volgens dezelfde planning, waar de sprints beginnen en eindigen om dezelfde momenten als elders binnen het bedrijf. De afstudeerperiode bevat tussen de aanvang van de afstudeerperiode (8 februari) en het begin van de presentatieweek (28 juni) precies 11 sprints. Aan het begin van elke sprint wordt vastgesteld welke taken er die sprint worden opgepakt. Wanneer aan het einde van de sprint er nog taken open staan wordt er heroverwogen of de taken naar de volgende sprint worden doorgeschoven, of dat deze niet urgent genoeg zijn om nog meer tijd in te laten nemen. In het laatste geval wordt dus de scope van de opdracht aangepast. De volgende planning zal worden gehandhaafd:



Figuur 1.1: Een schematische weergave van de projectplanning

V. Onderzoeksfase

De onderzoeksfase omvat het verkennen en beoordelen van verschillende implementatie-mogelijkheden. Hiervoor wordt gekeken naar huidige gangbare low-code methoden die enige vorm van code generation toepassen, of aangepast zouden kunnen worden om code te genereren. Hierbij worden de volgende onderzoeksvragen besproken voor elke mogelijke benadering:

1. Wat zijn de voor- en nadelen van deze oplossing op het gebied van gebruiksvriendelijkheid?
2. Hoe aanpasbaar en beschikbaar is de oplossing met betrekking tot aansluiting op reeds toegepaste technieken en systemen?

In het geval van onderzoeksvraag 1 wordt er ook specifiek gekeken naar de use case van de omgevingswet. In eerste instantie bestaat de beoogde gebruikersgroep uit juristen die zeer weinig tot geen voorkennis van dergelijke systemen hebben, zodoende kan niet worden aangenomen dat low-code oplossingen die bijvoorbeeld reeds in bedrijfskundige context worden toegepast direct bruikbaar zijn voor dit project.

De volgende implementatiemogelijkheden worden in dit onderzoek overwogen:

- Workflow Languages
- TurboBPMN
- Activity Diagrams
- Blockly
- And-Or Trees

Workflow Languages

Workflow Languages worden reeds breed toegepast binnen de bedrijfskunde om processen te automatiseren en te verbinden. Op deze manier worden resultaten van processen verwerkt en gebruikt als invoer voor andere processen. Er bestaat een groot aantal verschillende specificaties waaruit gekozen kan worden, zoals YAWL [3] en WDL [4]. Voor dit onderzoek wordt er uitgegaan van de Common Workflow Language [5], omdat hier een overwegend grote community achter zit die reeds een aantal verschillende tools heeft ontwikkeld. Eén van deze tools is Rabix [6] [7], een visuele editor voor de CWL. Aan de hand hiervan zullen de onderzoeksvragen worden behandeld.

Gebruiksgemak

Rabix blijkt overwegend complex in gebruik. Er wordt uitgegaan van voor gedefinieerde applicaties die zorgen voor invoer van gegevens, maar het is niet duidelijk hoe die aangemaakt kunnen worden. Ook is het specifiek modelleren van het onderliggende proces vrij ingewikkeld, en hiervoor zou redelijk veel kennis van Workflow Languages benodigd zijn.

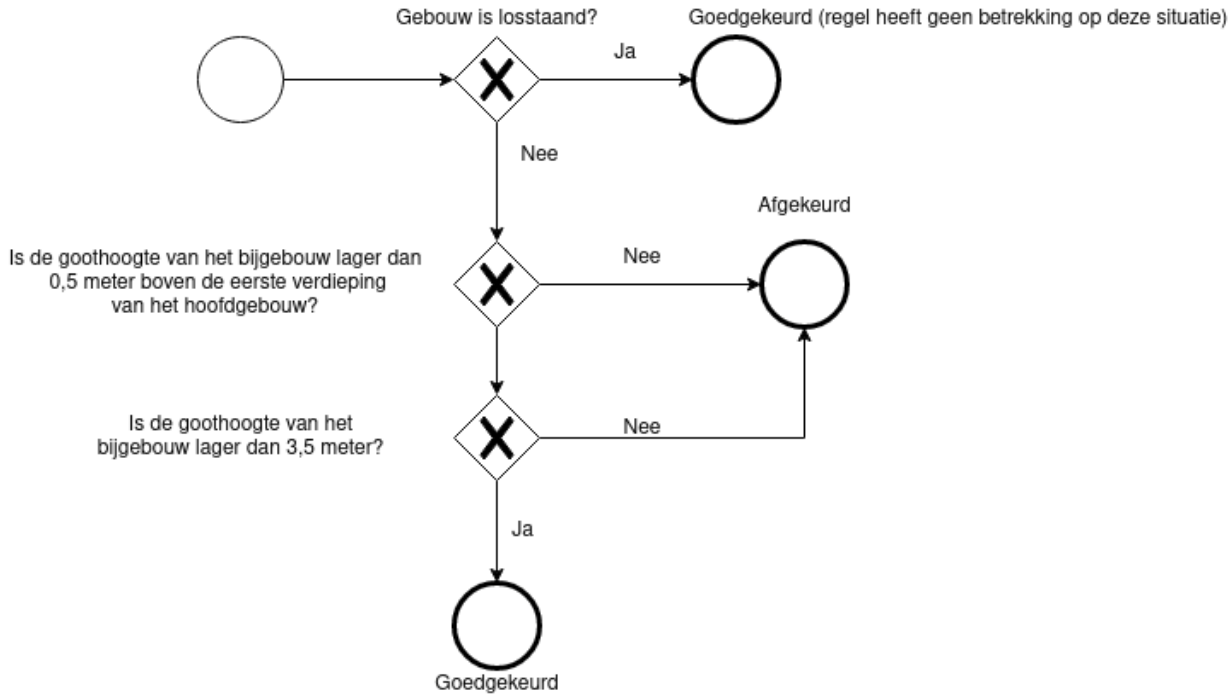
Aanpasbaarheid

Rabix is in principe open source software, en zou, wanneer er gekozen wordt voor het gebruik van workflow languages, kunnen worden aangepast voor de use case van dit project. Van zichzelf is Rabix echter niet zeer bruikbaar voor deze use case, omdat er geen mogelijkheid is om het als los component in andere applicaties op te nemen. Zodoende zou er

redelijk veel aangepast moeten worden om Rabix te gebruiken, of moet de contract-builder bij elke project waar deze wordt gebruikt als losse applicatie worden meegeleverd.

TurboBPMN

TurboBPMN [8] is een project dat erop gericht is om Smart Contracts voor de Etherium blockchain te schrijven aan de hand van Business Process Model and Notation (of BPMN) [9]. BPMN is een procesmodelleringstechniek waarmee bedrijfsprocessen kunnen worden gemodelleerd.



Figuur 2.1: Een voorbeeld van 199.3D (goothoogte van losstaande bijgebouwen) als BPMN-diagram

Gebruiksgemak

Het grote voordeel van het gebruiken van BPMN is de leesbaarheid van het diagram. BMPN-diagrammen zijn qua niveau vergelijkbaar met flowcharts. Er is echter een aantal zaken die de theoretische toepasbaarheid in de weg kunnen zitten, waaronder:

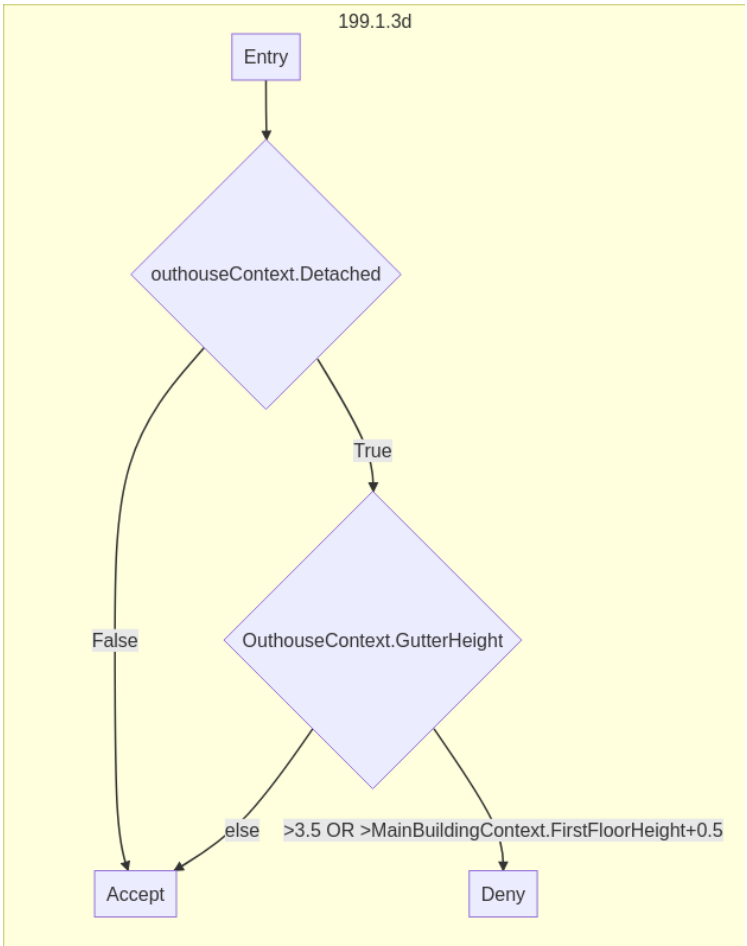
- In figuur 1.1 wordt slechts een kleine subset van de mogelijkheden van BPMN toegepast, namelijk begin, eind en exclusiviteitssymbolen. De BPMN-specificatie is veel breder, en is ook voornamelijk gericht op gebeurtenissen en acties. Daarmee zou BPMN een te brede (en complexe) toepassing zijn voor deze casus.
- BPMN kent geen variabelen of inputs van bepaalde typen, zoals strings, booleans of integers. Er is ook geen beperkte set aan operaties die kunnen worden omgezet in code.
- BPMN kent geen state of uitvoer (naast de acties binnen het proces), de omschrijvingen van de eindpunten in figuur 1.1 zijn puur illustratief en komen niet voor binnen de specificatie.
- De volgorde waarin voorwaarden worden gecontroleerd hoeft niet altijd uit te maken. In Figuur 1.1 wordt er een expliciete volgorde aangegeven waarin de voorwaarden worden afgegaan, maar dat is lang niet altijd relevant. Toch is er geen logische manier waarop meerdere voorwaarden gegroepeerd worden.

Aanpasbaarheid

TurboBPMN is een project dat niet actief wordt bijgehouden, en momenteel werkt het niet om een ontwikkelomgeving op te zetten om het project uit te testen. Er zou dus veel moeten worden doorontwikkeld aan TurboBPMN om de bruikbaarheid ervan te kunnen beoordelen, en dat terwijl er momenteel geen voorbeelden zijn van hoe het project werkt. Daarmee is het in de huidige fase van het project te risicovol om dieper te duiken in de mogelijke toepassing.

Activity Diagram

Activity Diagrams [10] zijn een andere vorm van procesmodellering onderdeel uitmaakt van UML. Binnen de softwareontwikkeling worden deze al veel ingezet om de logica van een programma grafisch in beeld te brengen. Voor het maken van Activity Diagrams kan in zekere zin gebruik worden gemaakt van Mermaid [11], een tekstmarkeertaal voor diagrammen. Mermaid heeft helaas geen ingebouwde ondersteuning voor Activity Diagrams, maar er kan wel gebruik worden gemaakt van de flowchart ondersteuning van Mermaid om diagrammen te maken die op een zeer vergelijkbare manier zijn gestructureerd, en die dezelfde principes volgen, zoals het figuur hieronder.



Figuur 2.2: Regel 199.1.3.D als Activity Diagram

Gebruiksgemak

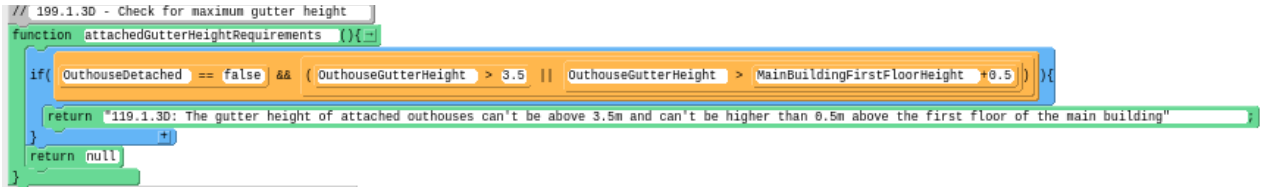
Hoewel Mermaid een visuele editor heeft (dat wil zeggen: aanpassingen in de broncode zijn direct zichtbaar in het diagram) is de leercurve overwegend steil. De syntax is enigszins intuïtief, zo kan een pijlverbinding worden genoteerd als `--->`. Men kan echter niet verwachten dat een nieuwe gebruiker zonder uitgebreide toelichting meteen met Mermaid aan de slag kan.

Aanpasbaarheid

Op het gebied van toepassing heeft Mermaid een aantal voordelen. Zo kan de huidige visuele editor in de vorm van een library met weinig inspanning in een andere applicatie worden ingebouwd. Ook heeft Mermaid al een eigen syntax die geparset zou kunnen worden en kan worden omgezet in TypeScript. Wel zouden er nog aanpassingen moeten worden gedaan op het gebied van gebruiksgemak om Mermaid te kunnen gebruiken voor dit project.

Blockly

Blockly [12] is een library ontwikkeld door Google voor het bouwen van visuele code-editors. De library kan worden gebruikt in webapplicaties om gebruikers zelf code in elkaar te laten klikken. De toepassing van deze library ligt vooral in het toegankelijk maken van programmeervaardigheden, zoals bij Code.org [13] en Scratch [14].



Figuur 2.3: een voorbeeld van regel 199.3.D in Blockly (code.org)

Gebruiksgemak

Het grote voordeel van Blockly is dat de complexiteit van de applicatie bij de ontwikkelaar ligt. Een ontwikkelaar zou een heel eigen dialect en een hele eigen manier van programmeren kunnen opzetten en daar een Blockly-implementatie omheen kunnen schrijven. In essentie zijn de enige soorten blokken die Blockly kent expressies, statements en velden waar expressies in kunnen worden geplaatst. Deze concepten kunnen gecombineerd worden tot een eigen stijl van programmeren, en daarmee toegankelijker en toepasbaarder worden gemaakt voor een specifieke use case.

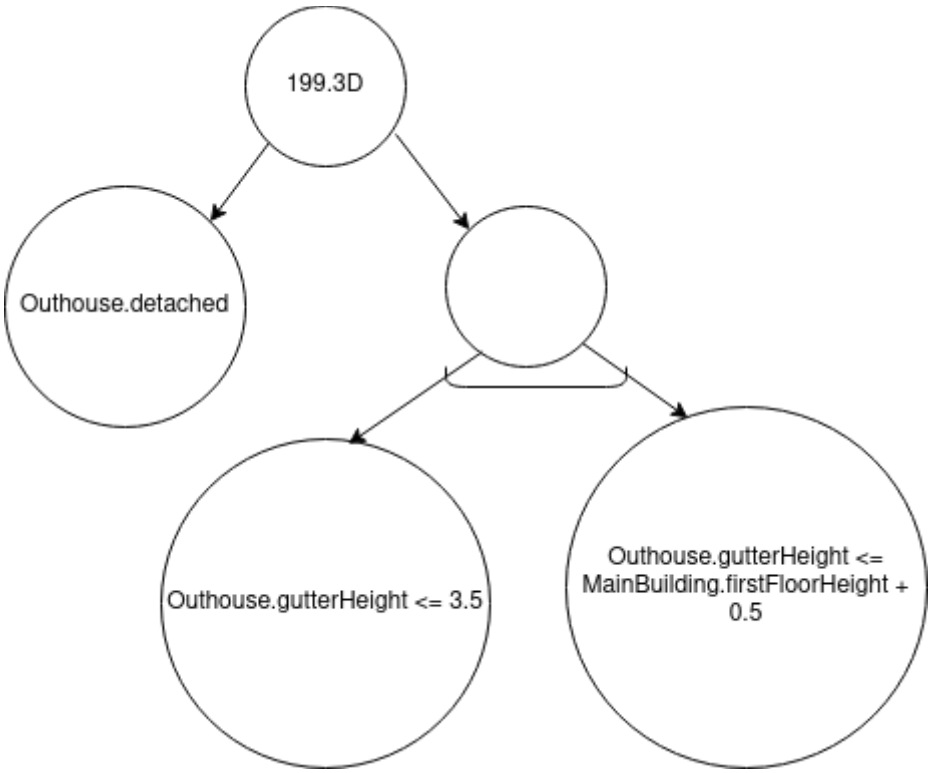
Aanpasbaarheid

Blockly kan op verschillende vlakken worden aangepast, maar het voornaamste onderdeel daarvan zijn de zelfgeschreven blokken. Deze blokken kunnen een aantal inputs en connectoren bevatten om andere blokken erop aan te sluiten. Zo kan een blok dus een input voor statements bevatten, terwijl het zelf ook onderdeel kan uitmaken van een aantal statements. Deze blokken hebben ook allemaal een eigen code generator, waarmee de velden, expressies, en statements gebruikt kunnen worden om en blok code uit te genereren.

Blockly heeft meegeleverde ondersteuning voor Javascript, Python, PHP, Lua en Dart. Daarmee is er geen meegeleverde ondersteuning voor TypeScript, waar de contracten voor de Validana Blockchain in zijn geschreven. Echter zijn TypeScript en Javascript zeer verwant aan elkaar, en zou met een paar aanpassingen de Javascript-generator kunnen worden gebruikt om alsnog TypeScript contracten te maken.

And-Or Tree

Een and-or tree [15] is een beslisboom waarmee beslissingen met een binaire uitkomst kunnen worden gemaakt. De boom bestaat uit knopen die ieder *waar* of *onwaar* kunnen zijn. Als een knoop meerdere subknopen heeft is die knoop *waar* als ten minste één van de subknopen *waar* is. Indien meerdere subknopen onderling met elkaar verbonden zijn, is de knoop alleen *waar* als alle verbonden subknopen *waar* zijn.



Figuur 2.4: Regel 199.3.D als and-or tree

Gebruiksgemak

Er bestaat nog geen implementatie voor code generators gebaseerd op and-or trees. Dat betekent dat hiervoor een volledig nieuwe oplossing moet worden gebouwd. Hiermee moet ook rekening worden gehouden met de stellingen in de knopen. In de schematische weergave in figuur 1.3 zijn de stellingen vrij lang, en bij een complexere regel kan dit een onoverzichtelijk diagram opleveren.

Aanpasbaarheid

Aangezien er een volledig nieuwe oplossing moet worden geschreven zou een dergelijke oplossing volledig kunnen worden gericht op dit project. Wel zou er hiervoor vanaf de grond af aan een nieuwe applicatie moeten worden geschreven, en dit kan ten koste gaan van de functionaliteit. Ook biedt deze oplossing weinig ruimte voor uitbreiding, zoals bijvoorbeeld complexere output.

Conclusie

De aanpasbaarheid, flexibiliteit en uitgebreide basis maken Blockly een goede keuze voor dit project. Er is ruimte voor een volledig eigen implementatie, maar er kan ook gebruik worden gemaakt van bestaande blokken. Op deze manier kan er een uitgebreide, goed werkende en passende oplossing worden gebouwd.

Met betrekking tot de gebruiksvriendelijkheid is Blockly ook een goede optie. Het framework is specifiek ontworpen zodat het makkelijk op te pakken is voor beginnende programmeurs. Mede door de beperking in het aantal verschillende concepten kenmerkt Blockly zich in simplisme, waardoor er een kleinere kans is dat een gebruiker het overzicht kwijtraakt.

VI. Realisatiefase

Requirements

Naar aanleiding van het onderzoek zijn de volgende requirements opgesteld:

- De applicatie zal worden gebouwd in Angular [16]
- De applicatie maakt gebruik van Blockly
- Een gebruiker kan Contexts en Inputs toevoegen
- Een gebruiker kan condities stellen rondom de waarde van Inputs
- Een gebruiker kan condities groeperen in een en/of structuur
- Een gebruiker kan groepen en condities samenvoegen in een Regel
- Een gebruiker kan controleren of de Blockly-code voldoet aan bepaalde standaarden
- Een gebruiker kan code exporteren naar Typescript contracten
- Een gebruiker kan code tussentijds opslaan

Specificatie

Voor de applicatie is een specificatie opgesteld die omschrijft hoe een contract er (visueel) uit ziet, en welke mogelijkheden het systeem bevat. De specificatie is dynamisch document dat kan worden aangevuld met nieuwe concepten die bijvoorbeeld betrekking hebben op het aanpassen van informatie op de blockchain. Aangezien het systeem momenteel uitsluitend een controlerende functie heeft zijn er ook alleen functionaliteiten die daarop betrekking hebben opgenomen.

De specificatie bestaat momenteel uit de volgende onderdelen:

Inputs en context

De gebruiker kan specificeren welke informatie er voor het systeem gebruikt kan worden in de contractcode. Een bijgebouw kan bijvoorbeeld losstaan, en heeft een nok- en goothoogte. In het contract is dan een context Bijgebouw beschikbaar met een boolean waarde genaamd *Losstaand* en getalswaarden voor *Nokhoogte* en *Goothoogte*.

Regels

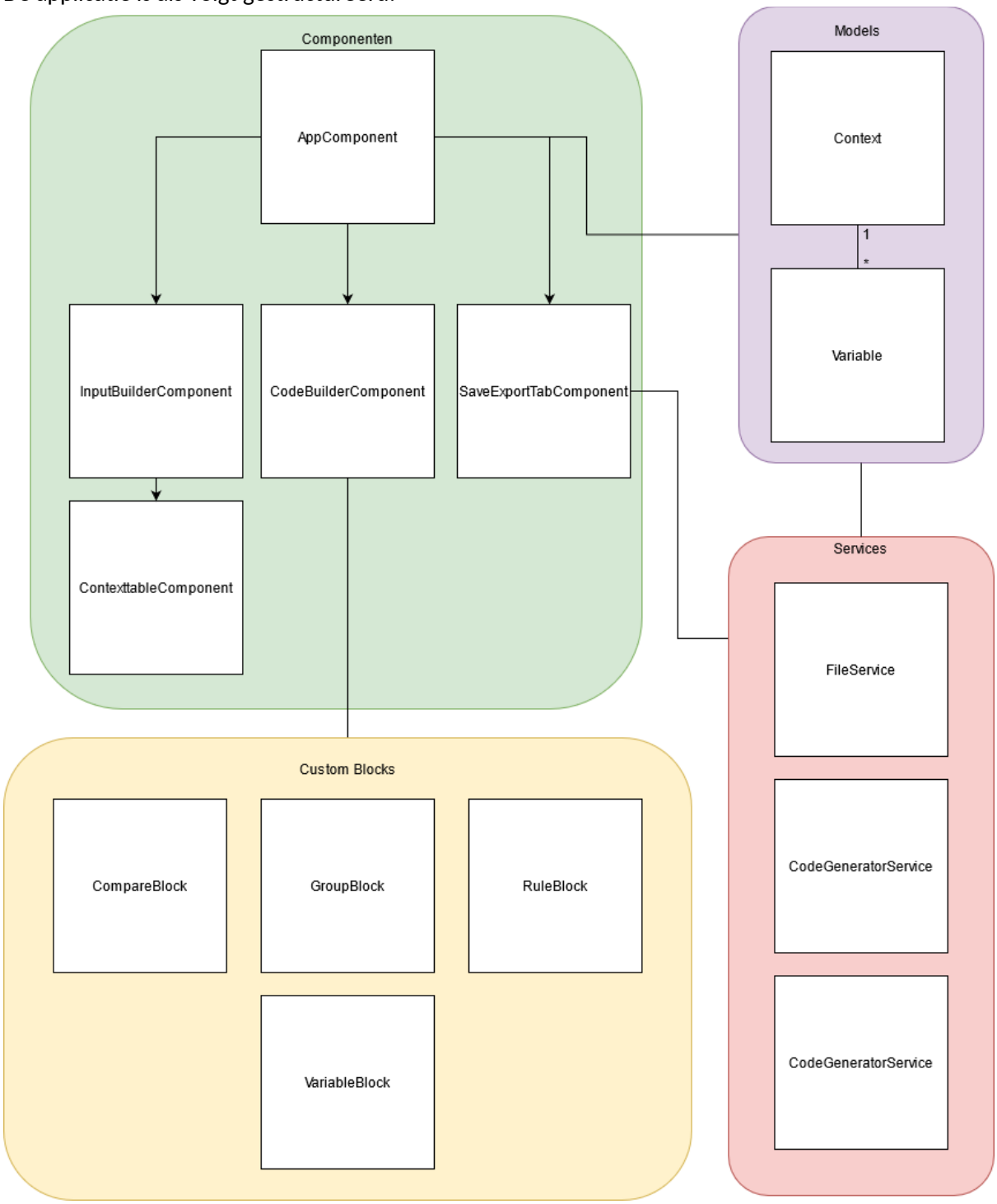
Een regel is een onderdeel van een contract. Een regel bestaat uit een omschrijving en uit meerdere condities die samen een positief of negatief resultaat kunnen teruggeven. Daarmee moet elke regel zelfstandig kunnen worden uitgevoerd. Wanneer een regel geen betrekking heeft op een bepaalde situatie (bijvoorbeeld omdat deze alleen geldt voor losstaande bijgebouwen en niet voor aan- of uitbouw) komt er een positief resultaat uit. Bij het uitvoeren van een contract worden alle regels uitgevoerd. Wanneer alle regels een positief resultaat teruggeven komt het contract positief terug. Wanneer er regels zijn waarbij niet aan de voorwaarden wordt voldaan is het resultaat van het contract negatief.

Applicatie[3]

De applicatie bestaat uit een aantal tabbladen waarin een gebruiker verschillende onderdelen van het contract kan aanpassen.

[3] Schermafbeeldingen van de applicatie zijn te vinden in Bijlage B

De applicatie is als volgt gestructureerd:



Figuur 3.1: Een schematische weergave van de applicatiestructuur.

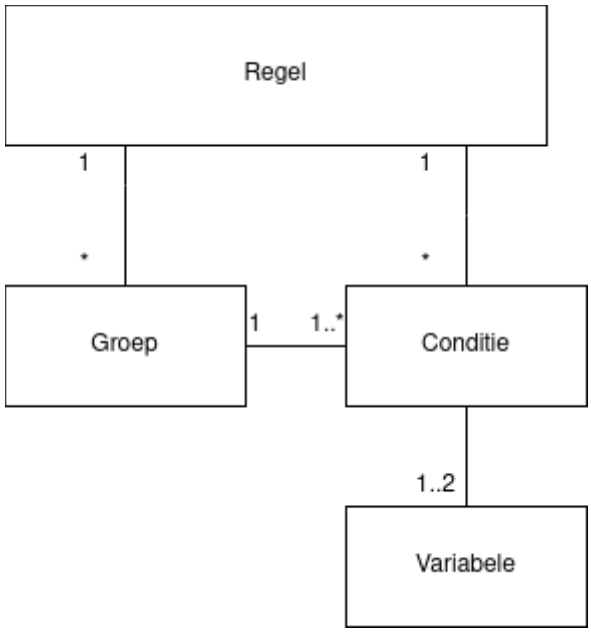
De applicatie bestaat uit drie tabbladen, waarin de gebruiker respectievelijk contexts met inputs kan aanmaken, code kan schrijven en het project kan laden, opslaan, valideren en exporteren. Het SaveExportTabComponent maakt gebruik van de services om code te valideren en om te zetten in TypeScript.

Het CodeBuilderComponent maakt gebruik van de Custom Blocks, die extensies zijn van de CustomBlock klasse uit de *ngx-blockly* [17] library, een voor Angular gebouwde wrapper om de Blockly JavaScript-library.

Inputs

De gebruiker kan in de eerste tab contexts en inputs aanmaken. Een context bevat gegroepeerde informatie in de vorm van verschillende inputs. Een bijgebouw heeft bijvoorbeeld een oppervlakte, nokhoogte, goothoogte, enzovoort. Dit concept is vergelijkbaar met hoe klassen werken in object georiënteerd programmeren. In eerdere interne verslaggeving van Coinversable wordt er voor dit project al gesproken over 'object georiënteerde tekst'. Dat wil zeggen:

- Onderwerpen en onderdelen binnen het wetsartikel worden omschreven in een aantal eigenschappen
- Die eigenschappen worden gebruikt in voorwaarden (kleinere onderdelen van een regel)
- De uitkomst van de combinatie van één of meerdere voorwaarden vormt de uitkomst van de regel



Figuur 3.2: Een schematische weergave van hoe de blokken zich tot elkaar verhouden

Code-Editor en Custom Blocks

De tweede tab bestaat uit een code-editor in de vorm van een Blockly-frame. In dit frame kan met verschillende blokken een contract worden opgebouwd. Om de complexiteit van de applicatie zo laag mogelijk te houden, is er maar weinig gebruik gemaakt van de standaard blokken die met Blockly worden meegeleverd. In plaats daarvan is er een aantal zelfgeschreven blokken toegevoegd die zijn gebaseerd op de specificatie.

De standaard blokken die zijn opgenomen zijn blokken die wiskundige functies bevatten, omdat deze mogelijk gebruikt kunnen worden in de regelgeving. Ook de blokken met logische functies worden gebruikt, voornamelijk het true/false-blok. De custom blokken bieden de volgende mogelijkheden:

Regels

Regels zijn de voornaamste blokken van een contract. Alle andere blokken zijn onderdeel van een regel. Een regel heeft input voor de naam en ruimte voor groepen of voorwaarden. Een regel geeft *waar* terug op het moment dat er aan alle voorwaarden in de regel wordt voldaan.

Een contract bestaat uit meerdere regels, die allemaal worden uitgevoerd. Wanneer alle regels *waar* teruggeven is het resultaat van het contract ook *waar*. In gevallen waar één of meerdere regels *niet waar* teruggeven geeft het contract ook *niet waar* terug, met daarbij de regels waaraan niet wordt voldaan.

Groepen

Groepen worden gebruikt om één of meerdere voorwaarden te groeperen. Een groep bestaat in twee soorten: *all* en *one*. In het geval van een *all* groep moet aan alle voorwaarden binnen die groep worden voldaan, in het geval van een *one* groep moet aan slechts één van de voorwaarden worden voldaan. Een groep kan naast voorwaarden ook andere groepen bevatten.

Voorwaarden

Een voorwaarde is het kleinste onderdeel van een regel. De voorwaarden van een regel zijn vaak rechtstreeks uit het wetsartikel te halen. In regel 199.3.D staat bijvoorbeeld:

> *De goothoogte van aanbouwen, uitbouwen en aangebouwde bijgebouwen mag niet meer dan 3,5 m, of 0,50 m boven de vloer van de eerste verdieping van het hoofdgebouw, bedragen;*

Hieruit zijn de volgende voorwaarden te halen:

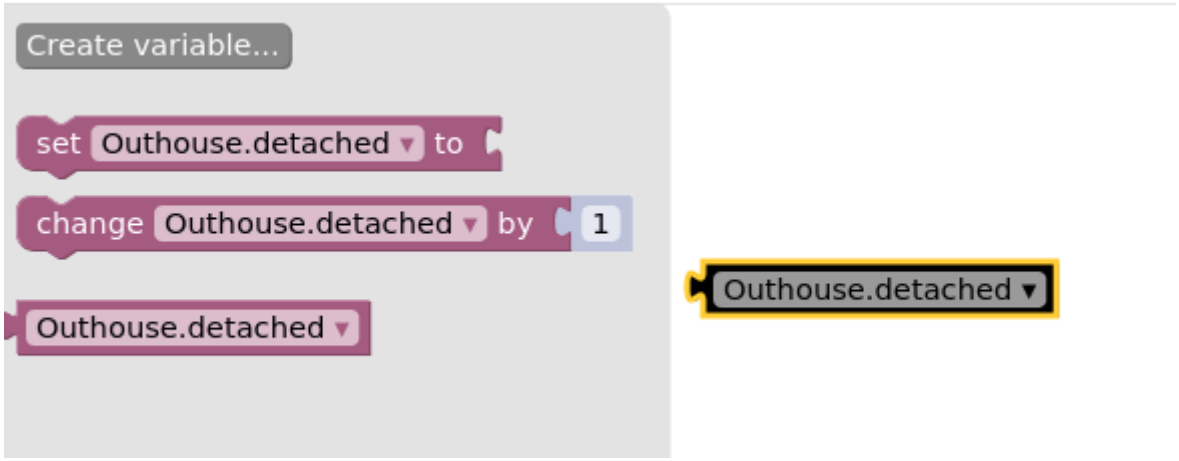
- Het bijgebouw is vrijstaand, OF
    - De goothoogte van het bijgebouw mag niet hoger zijn dan 3,5 meter EN
    - De goothoogte mag niet hoger zijn dan 0,5 meter boven de vloer van de eerste verdieping van het hoofdgebouw

Een voorwaarde bestaat uit twee value inputs[4] en een operator waarmee de twee waarden kunnen worden vergeleken. De volgende operators worden ondersteund:

- Gelijk aan
- Ongelijk aan
- Groter dan
- Groter dan of gelijk aan
- Kleiner dan
- Kleiner dan of gelijk aan

Inputs

[4] In Blockly is een value input een plaats waar een expressie kan worden ingevoegd. Dit kan bijvoorbeeld een functie-aanroep of een variabele zijn



Figuur 3.3: Aan de linkerkant de ingebouwde variabele-functionaliteit, aan de rechterkant de zelfgebouwde variant

Input blokken zijn er ter vervanging van de ingebouwde variabele blokken. Deze twee verschillen op een aantal vlakken:

Ten eerste kan de waarde van een input blok niet aangepast worden. De invoer van een contract hoeft niet te worden overschreven binnen de scope van het contract, om de integriteit van de gegevens tussen regels onderling te behouden.

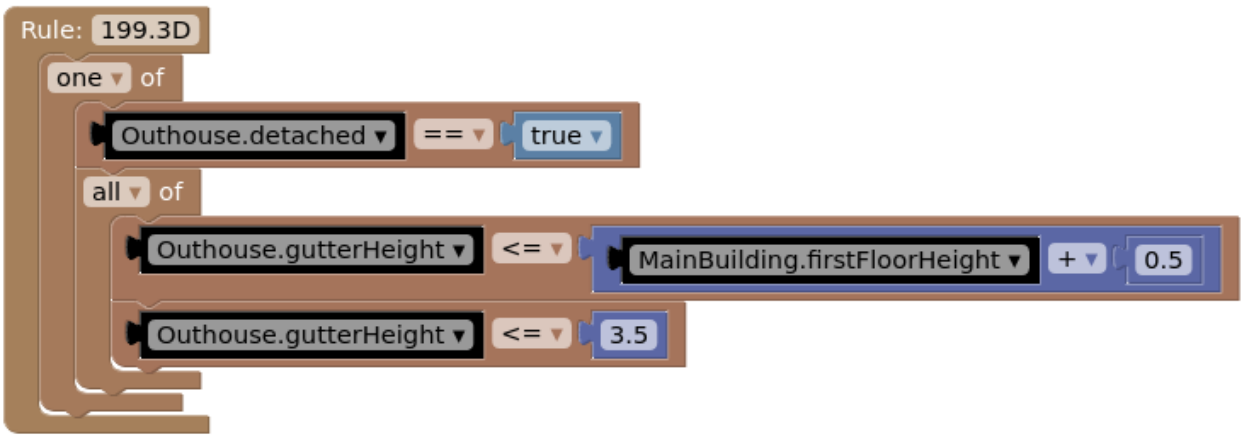
Ten tweede is het input blok gekoppeld aan de input tab van de applicatie. Hierdoor hoeven deze blokken niet meer handmatig te worden aangemaakt in Blockly, maar bevat het input blok een dropdown waarmee de input kan worden geselecteerd. Bij gebruik van de standaard variabele-functionaliteit bestaat deze informatie uitsluitend in de context van de Blockly-workspace, en kan deze niet worden worden aangepast.

Ten slotte is er door de koppeling met de input tab meer informatie beschikbaar over de gebruikte input. Variabelen in Blockly hebben van zichzelf geen type, waardoor de validator niet aan type checking zou kunnen doen. Daarnaast kan de gebruiker een Input verder toelichten bij het aanmaken van de contexts. Deze toelichting kan dan weergegeven worden tijdens het programmeren[5].

Conditions as statements

De applicatie maakt gebruik van een op propositielogica gebaseerd paradigma dat omschreven kan worden als *conditions as statements*. Dat wil zeggen dat de code primair bestaat uit voorwaarden die gecombineerd de state van een regel (geaccepteerd of afgewezen) bepalen. Dit concept vergroot de leesbaarheid van een contract, en vergemakkelijkt het programmeerproces. In plaats van *if statements* met daarin blokken code worden regels, net als in natuurlijke taal, uitgedrukt in één of meerdere voorwaarden. Hierdoor hoeft de gebruiker geen rekening te houden met principes die in de meeste programmeertalen worden toegepast en een steilere leercurve hebben, zoals variabelen en *return statements*.

[5] Deze functionaliteit is momenteel nog niet geïmplementeerd



Figuur 3.4: Een implementatie van regel 199.3.D in Conditions as Statements, opvallend is de overeenkomst met de regel zoals deze omschreven is onder *Voorwaarden*

Contracten

Validana-contracten [18] zijn geschreven in Typescript en volgen een specifieke structuur. Ten eerste bevat elk contract een template interface waarin alle informatie staat die nodig is om het contract uit te voeren. Een instantie van deze interface wordt meegegeven bij de uitvoering van het contract.

Een contractbestand exporteert een standaardklasse die een extensie is van een standaard contractklasse. Deze bevat een functie *code*, die teruggeeft of aan de voorwaarden wordt voldaan, en die de code bevat die deze voorwaarden controleert.

Indien niet aan de voorwaarden wordt voldaan, of wanneer er onduidelijkheid is of aan de voorwaarden kan worden voldaan (dit kan voorkomen wanneer er nog informatie ontbreekt, of wanneer er sprake is van onnauwkeurige informatie), bevat het resultaat van het contract toelichting met de specifieke regels en onderdelen waaraan (wellicht) niet wordt voldaan.

Code validator

Onder de *Save and Export* tab kan een gebruiker een project opslaan of exporteren, en kan er een validator worden uitgevoerd die controleert of de geschreven code correct is. De validator parset de xml-code met *fast-xml-parser* [19], en doorloopt elk element. Hier wordt een aantal zaken gecontroleerd:

Ten eerste of alle blokken die hiërarchisch bovenaan staan regelblokken zijn. Andere blokken, zoals groepen en voorwaarden, maar ook wiskundige operaties kunnen alleen bestaan binnen de context van een regel.

Ten tweede of alle blokken in een regel valide zijn. Dit kunnen alleen groepen of voorwaarden zijn.

Ten derde of types in voorwaarde blokken valide zijn. Bij een is (niet) gelijk aan operator moeten de types van beide kanten gelijk zijn. Bij een kleiner dan, kleiner dan of gelijk aan, groter dan en groter dan of gelijk aan operator moeten de expressies aan beide kanten een getal bevatten.

Code Generator

De code generator is een service die gebruik maakt van de bestaande functionaliteit van Blockly om code te genereren. Van elk blok bevat de onderliggende klasse functies om de code van dat blok om te zetten in één van de ondersteunde talen. In dit project wordt de

functie *toJavaScriptCode()* gebruikt om Type- en Javascriptcode te genereren. Van de buitenste (of root-) blokken wordt de code generator aangeroepen, deze blokken doorlopen recursief hun eigen inhoud, en geven een string met code terug.

De CodeGeneratorService maakt gebruik van een template contract met een aantal onderdelen:

- De template interface bevat informatie over de gegevens die in het contract gebruikt worden. Deze interface wordt gebaseerd op de contexts en inputs. Een object van deze interface wordt als payload aan het contract meegegeven.
- Twee helper functies die de logica van de *all* en *one* blokken bevatten. Het return type van deze functies is boolean, en deze kunnen worden aangeroepen met één of meerdere boolean expressies (dat zijn andere blokken) of functiedefinities die een boolean als return type hebben (dat zijn voorwaarden).
- Een Rule klasse die de logica van een regel bevat
- De contract code, waarin van elke regel een Rule object wordt gemaakt en wordt gecontroleerd of aan alle regels wordt voldaan.
  De code die uit de Blockly-workspace komt wordt door de service opgesplist en ingevuld op de verschillende plekken in het contract-template.

Contracten in XML en TypeScript

Een contract kan tussentijds als xml-bestand worden opgeslagen en geopend. De keuze voor XML is gebaseerd op het feit dat Blockly al onderwater gebruik maakt van XML om de blokken te structureren. De FileService-service bevat een functie om de XML van de context weer om te zetten naar Context en Variable objecten. Hierdoor kan een gebruiker zijn voortgang opslaan en later verder gaan zonder het contract helemaal opnieuw te moeten schrijven. Op het moment dat de gebruiker het contract af heeft kan het met behulp van de code generator een TypeScript contract worden geëxporteerd.

Deployments

Momenteel kan de gebruiker een contract handmatig exporteren als TypeScript-bestand. Dit bestand kan in bestaande tools binnen Coinversable gebruikt worden om dit contract naar een blockchain te uploaden, en om het contract te testen met voorbeeld-waarden. Ook zou dit het proces van het later aanpassen van contracten die al op de blockchain staan significant versimpelen.

In de laatste fase van dit project zal er nog gekeken worden naar het integreren van een client library voor Validana zodat een gebruiker vanuit de applicatie het contract direct kan publiceren.

VII. Evaluatiefase

In de evaluatiefase wordt een gebruikerstest van het prototype uitgevoerd. Een medewerker met weinig kennis van programmeren zal hiervoor proberen een wetsartikel te implementeren.

Gebruiker

De Business Developer van Coinversable heeft eerder bij een woningcorporatie gewerkt en is daarmee bekend met de in de wetgeving gebruikte begrippen. Hij heeft beperkte ervaring met andere low-code systemen, waaronder Mendix [20], en heeft eerder geprogrammeerd in Python.

Testmethode

Tijdens de eerste test zal de gebruiker regel 199.1.3F implementeren, zonder daarbij verdere toelichting te hebben gehad over het systeem. Wanneer de gebruiker vast lijkt te lopen wordt er kort toegelicht waar de knelpunten zitten, en gaat de gebruiker verder met de afronding. Hierna wordt de applicatie verder toegelicht en besproken, waarna de gebruiker nogmaals een (ietwat complexere) regel zal implementeren, namelijk 199.1.3D.

Observaties eerste fase

De gebruiker koppelt in eerste instantie de programmeercontext en programmeerlogica aan elkaar. In plaats van de applicatie alleen te voorzien van de benodigde context en input, maakt de gebruiker slechts één input aan waar hij in het toelichtingsveld de tekst van de regel plakt. Nadat aan de gebruiker wordt uitgelegd dat hier alleen moet worden opgesteld welke informatie er voor deze regel beschikbaar moet zijn, kan de gebruiker zelfstandig bepalen welke inputs dit zijn, en onder welke context.

Ook is het de gebruiker onduidelijk waartoe de applicatie dient. In eerste instantie gaat hij ervan uit dat niet alleen moet worden aangeleverd welke informatie beschikbaar moet zijn, maar ook de informatie zelf. Er wordt toegelicht dat het hier alleen gaat om hoe de informatie verwerkt wordt, en niet om hoe de informatie verkregen wordt.

Het is voor de gebruiker onduidelijk of het verplicht is om iets in te vullen bij de toelichting op de input, en wat dat dan zou moeten zijn.

De gebruiker heeft moeite met het afvangen van de conditie waarin de regel geen betrekking heeft op de situatie. Dat wil zeggen: er moest worden toegelicht dat, indien het gebouw **niet** vrijstaand is, de regel een positief resultaat moet teruggeven, en dat dat kan door gebruik te maken van een OF-groep.

Evaluatie eerste fase

De gebruiker geeft aan dat het prettig zou zijn wanneer de applicatie de condities ook in natuurlijk taalgebruik zou weergeven.

Er is veel aan te merken op de gebruikerservaring van de applicatie, zo is het niet mogelijk om fouten die zijn gemaakt in het definiëren van de inputs te corrigeren zonder de applicatie te herladen. Ook is het niet duidelijk welke informatie verplicht moet worden ingevoerd, waardoor de gebruiker niet meer wist wat in te vullen bij de toelichting op de variabelen.

Observaties tweede fase

Na de verdere uitleg weet de gebruiker zelf te bepalen welke variabelen er nodig zijn, en maakt de gebruiker hier zelfstandig contexts en inputs voor aan. Wel maakt de gebruiker gebruik van spaties in de naam van de input, en daar is geen ondersteuning voor. Dit wordt ook niet afgevangen door de applicatie.

De gebruiker probeert in eerste instantie alle condities in dezelfde groep te plaatsen, maar komt op een gegeven moment tot de conclusie dat dat niet tot het gewenste resultaat leidt. Na de hint dat groepen ook andere groepen kunnen bevatten komt de gebruiker wel zelfstandig tot de oplossing.

Ook het opsplitsen van eisen aan een enkele eigenschap in meerdere condities gaat niet vanzelf. De eis *goothoogte (...) mag niet meer dan 3,5 m, of 0,50 m boven de vloer van de eerste verdieping van het hoofdgebouw*, *bedragen* bestaat uit twee condities, maar de gebruiker probeert deze in één conditie te verwerken.

Evaluatie tweede fase

De gebruiker stelt voor om de applicatie uit te breiden met een standaardbibliotheek voor inputs. Op deze manier kunnen bepaalde eigenschappen hergebruikt worden in verschillende contracten, en is er consistentie in hoe deze omschreven worden.

De gebruiker ziet veel toegevoegde waarde in het systeem. Het lijkt hem een goede aanvulling of vervanger van bestaande oplossingen, zoals Building Information Modeling [21]. De applicatie werd ervaren als overwegend makkelijk om te leren, en de programmeerlogica was snel uitgelegd en makkelijk om op te pakken.

VIII. Adviezen en uitbreidingen

Dit hoofdstuk omschrijft mogelijke uitbreidingen van het prototype die geen prioriteit hebben gekregen gedurende de afstudeerperiode, maar die wel van toegevoegde waarde zouden kunnen zijn aan het project.

Uitbreiding en toepassing prototype

Het prototype kan los gebruikt worden om contracten te bouwen, en kan worden uitgebreid met meer blokken. Ook kunnen sommige onderdelen van het project, zoals de code validator en code generator, worden hergebruikt of herschreven om in andere applicaties te worden toegepast. Wel is het belangrijk dat in de ontwikkeling van het prototype weinig aandacht geschonken is aan de algemene gebruikerservaring, en dat aanpassingen hierin cruciaal zijn voor prettig gebruik van het product. Zie hiervoor ook de opmerkingen in het vorige hoofdstuk.

Alternatieve uitkomsten voor regels

In het de conceptapplicatie van Coinversable van de omgevingswet wordt er reeds gebruik gemaakt van een 'stoplicht'-model. Dat model heeft naast de reguliere acceptatie- en afwijzingsmogelijkheid nog een derde optie. In dat geval kan er op basis van de ingevoerde gegevens, in verband met de nauwkeurigheid van die gegevens, niet worden bepaald of de aanvraag kan worden geaccepteerd Dit hangt echter af van onnauwkeurige brondata in de invoer van het contract[6], en niet van het contract zelf. Zodoende is deze situatie niet meegenomen in het prototype.

Het kan echter voorkomen dat het wenselijk is om, naast de naam van de regel, ook terug te geven *waarom* er aan een specifieke regel wel of niet wordt voldaan (bijvoorbeeld omdat die regel niet op de situatie van toepassing is). Dit zou een goede toevoeging kunnen zijn aan het prototype om het resultaat van een aanvraag uitgebreider toe te lichten.

Testing

Binnen dit project zouden twee mogelijke scenario's omtrent testen relevant zijn:

Ten eerste is er de mogelijkheid om de applicatie zelf te testen. Gezien het afstudeerproject een prototype omvat, is er geen prioriteit gegeven aan het schrijven van bijvoorbeeld unit tests. Het zou echter wel relevant kunnen zijn om losse onderdelen van de applicatie (zoals bijvoorbeeld de code generator) te kunnen testen om geautomatiseerd te bepalen of de applicatie naar behoren werkt.

Een andere mogelijkheid is dat een gebruiker wil testen of het geschreven contract de juiste uitkomst biedt. In de casus van de omgevingswet is het zeker denkbaar dat een gebruiker het contract uittest met de parameters van een eerder behandeld verzoek, om te bepalen of de uitkomst van het contract overeenkomt met hoe het verzoek handmatig is afgehandeld.

Opslag van broncode en versiebeheer

In de huidige opzet van de applicatie slaat de gebruiker lokaal de XML-bestanden van een contract op, en wordt de gegenereerde versie van het contract op een blockchain geplaatst. Echter is hiermee de visueel versimpelde versie van het contract niet toegankelijk voor andere gebruikers, en is er ook geen garantie dat die versie zorgvuldig wordt bewaard om in

[6]Gegevens die afkomstig zijn uit (bijvoorbeeld) het kadaster kunnen bepaalde afwijkingen bevatten, zoals waar de erfgrens ligt. Het kan dus voorkomen dat er aan de hand van de beschikbare gegevens niet kan worden bepaald of een bijgebouw te dicht bij de erfgrens staat of niet

de toekomst aan te passen. Het zou dus zeker voordelig zijn als er een vorm van versiebeheer zou zijn waarmee XML-versies van contracten kunnen worden bewaard, hergebruikt en aangepast.

Integratie van de client in het systeem

In het huidige prototype slaat een gebruiker een contract lokaal op, om deze vervolgens handmatig naar de Validana-blockchain te uploaden. Validana heeft een client die gemakkelijk in webapplicaties kan worden toegepast om direct te communiceren met een blockchain. Dit kan zodoende worden toegepast om contracten direct op de blockchain te publiceren, maar ook om contracten op de blockchain uit te testen, zodat er naast het schrijven van contracten ook kan worden bepaald of de contracten naar behoren functioneren.

Acties

Er is bewust voor gekozen om contracten die in het prototype worden gebouwd een uitsluitend controlerende functie te geven, voornamelijk omdat dat het beste bij de casus paste. Echter kan het voorkomen dat in een andere toepassing er ook informatie moet worden weggeschreven naar de Validana-blockchain. In dat geval zou het handig zijn als een gebruiker ook in een contract kan inbouwen dat, indien er aan bepaalde voorwaarden is voldaan, er acties kunnen worden uitgevoerd. Idealiter komt deze functionaliteit los te staan van de controlerende functie van de applicatie, om de modulariteit hiervan te behouden en voor de gebruiker het overzicht te bewaren.

IX. Reflectie op het afstudeerproces

Mede door de pandemie heb ik de afstudeeropdracht vrijwel volledig vanuit huis uitgevoerd. Dit was een factor die erg demotiverend werkte, en dit heeft voor mij het belang van een gestructureerde werkplek met andere collega's in de buurt bevestigd. Wel heb ik op een aantal momenten online met een medestudent samengewerkt, wat ons beiden erg goed heeft geholpen.

Hoewel er maar weinig collega's binnen Coinversable echt op de hoogte waren van wat mijn afstudeerproject inhield, werkten de momenten die ik met hen had erg motiveren om elke dag weer met de opdracht aan de slag te gaan. Het gebrek aan motivatie dat bij mij ontstond door het grotendeels thuiswerken, werd opgelost doordat er een vast moment aan het begin van de ochtend was waar ik even met collega's sprak, waarna ik aan het werk ging.

Ook heb ik in het bijzonder veel gehad aan de wekelijkse momenten met Erwin. Hierin bespraken we mijn voortgang, en kon ik er zeker van zijn dat ik met betrekking tot mijn onderzoek en benaderingen nog op het juiste spoor zat. Ook met betrekking tot de werking van het prototype heb ik veel gehad aan zijn input en feedback, zodat ik nu tevreden kan zijn over de toegankelijkheid daarvan.

Tegen het einde van de afstudeerperiode aan stuitte ik op een andere low code implementatie die op juridische casussen gericht is, genaamd Blawx [22]. Ik betreur het dat ik dit project pas zo laat ontdekt heb, want het had een gedeelte van het proces kunnen versnellen en inspiratie kunnen leveren voor de specificatie. Toch ben ik er ook van overtuigd dat de huidige implementatie net zo goed dekkend is voor de casus, en tegelijkertijd toegankelijker is in gebruik. Blawx maakt namelijk gebruik van een vorm van logica die een stuk complexer geformuleerd is, en waarbij het definiëren van de context door het gebruiken van de context heen gebeurt, wat enige verwarring zou kunnen opleveren. Wel zouden concepten die in Blawx voorkomen ook kunnen worden toegepast in mogelijke uitbreidingen van het prototype, daarmee kan het als nog van toegevoegde waarde zijn.

Ondanks de omstandigheden kijk ik positief terug op het proces en het resultaat. Ik ben ervan overtuigd dat ik een systeem heb kunnen neerzetten dat bijdraagt aan het versimpelen van het schrijven van Smart Contracts, zonder dat daarbij expliciete kennis nodig is van zowel programmeren als wetgeving. Wel lijkt het me goed als in verdere doorontwikkeling van het prototype beoogde eindgebruikers (voornamelijk juristen) zijn betrokken om de gebruiksvriendelijkheid te waarborgen.

Ik vond het erg leerzaam om een overweging te maken van verschillende oplossingen, en deze op het gebied van gebruiksgemak te beoordelen. Wel vond ik het lastig om dit op een strikt wetenschappelijke manier te benaderen.

Tijdens de onderzoeksfase heb ik mijn conclusies vooral getrokken uit mijn eigen inschattingen en die van andere mensen in het team. Daarmee kon ik in eerste instantie niet met volledige zekerheid stellen dat het resultaat ook daadwerkelijk bruikbaar was voor de beoogde gebruikersgroep. In het vervolg zou ik dan ook niet tevreden zijn met een dergelijk onderzoeksresultaat zonder dat daarbij ook een potentiële gebruiker is geconsulteerd.

Ik heb wel gemerkt dat het van toegevoegde waarde was om concreet op te maken hoe het product eruit komt te zien wanneer een bepaalde oplossing wordt toegepast. In een volgend vergelijkend onderzoek zou ik die methode ook zeker opnieuw gebruiken.

X. Referenties

[1] „Validana - Homepage," [Online]. Available: https://validana.io/.

[2] „Typescript - Home," [Online]. Available: https://www.typescriptlang.org/.

[3] „YAWL BPM," [Online]. Available: https://yawlfoundation.github.io/.

[4] „OpenWDL," [Online]. Available: https://openwdl.org/.

[5] „Common Workflow Language - Home," [Online]. Available: https://www.commonwl.org/.

[6] „Rabix - Home," [Online]. Available: https://rabix.io/.

[7] „Rabix Composer - Documentation," [Online]. Available: http://docs.rabix.io/rabix-composer-home.

[8] „silkchain/TurboBpmn - Github," [Online]. Available: https://github.com/silkchain/TurboBpmn.

[9] „BPMN Specification," [Online]. Available: https://www.bpmn.org/.

[10] „Activity Diagrams - UML Diagrams," [Online]. Available: https://www.uml-diagrams.org/activity-diagrams.html.

[11] „Mermaid - Home," [Online]. Available: https://mermaid-js.github.io/mermaid/.

[12] „Blockly | Google Developers," [Online]. Available: https://developers.google.com/blockly.

[13] K. Apone, „Why does Code.org use Blockly, a visual programming language, for its elementary-level courses?," 18 09 2019. [Online]. Available: https://support.code.org/hc/en-us/articles/202518363-Why-does-Code-org-use-Blockly-a-visual-programming-language-for-its-elementary-level-courses-.

[14] „Scratch - Home," [Online]. Available: https://scratch.mit.edu/.

[15] „AI Materials: And-Or Graph," [Online]. Available: http://aimaterials.blogspot.com/p/and-or-graph.html.

[16] „Angular - Home," [Online]. Available: https://angular.io/.

[17] „roroettng/ngx-blockly - Github," [Online]. Available: https://github.com/roroettg/ngx-blockly.

[18] W. d. Bakker, „Writing Smart Contracts - Validana Wiki," 28 02 2020. [Online]. Available: https://github.com/Coinversable/validana-processor/wiki/Writing-Smart-Contracts.

[19] „hamravesh/fast-xml-parser - Github," [Online]. Available: https://github.com/hamravesh/fast-xml-parser.

[20] „Mendix - Home," [Online]. Available: https://www.mendix.com/.

[21] „Building Information Modeling - Autodesk,” [Online]. Available: https://www.autodesk.nl/solutions/bim.

[22] „Blawx - Home,” [Online]. Available: https://www.blawx.com/.

XI. Bijlagen

Bijlage A: Toegepast fragment bestemmingsplan

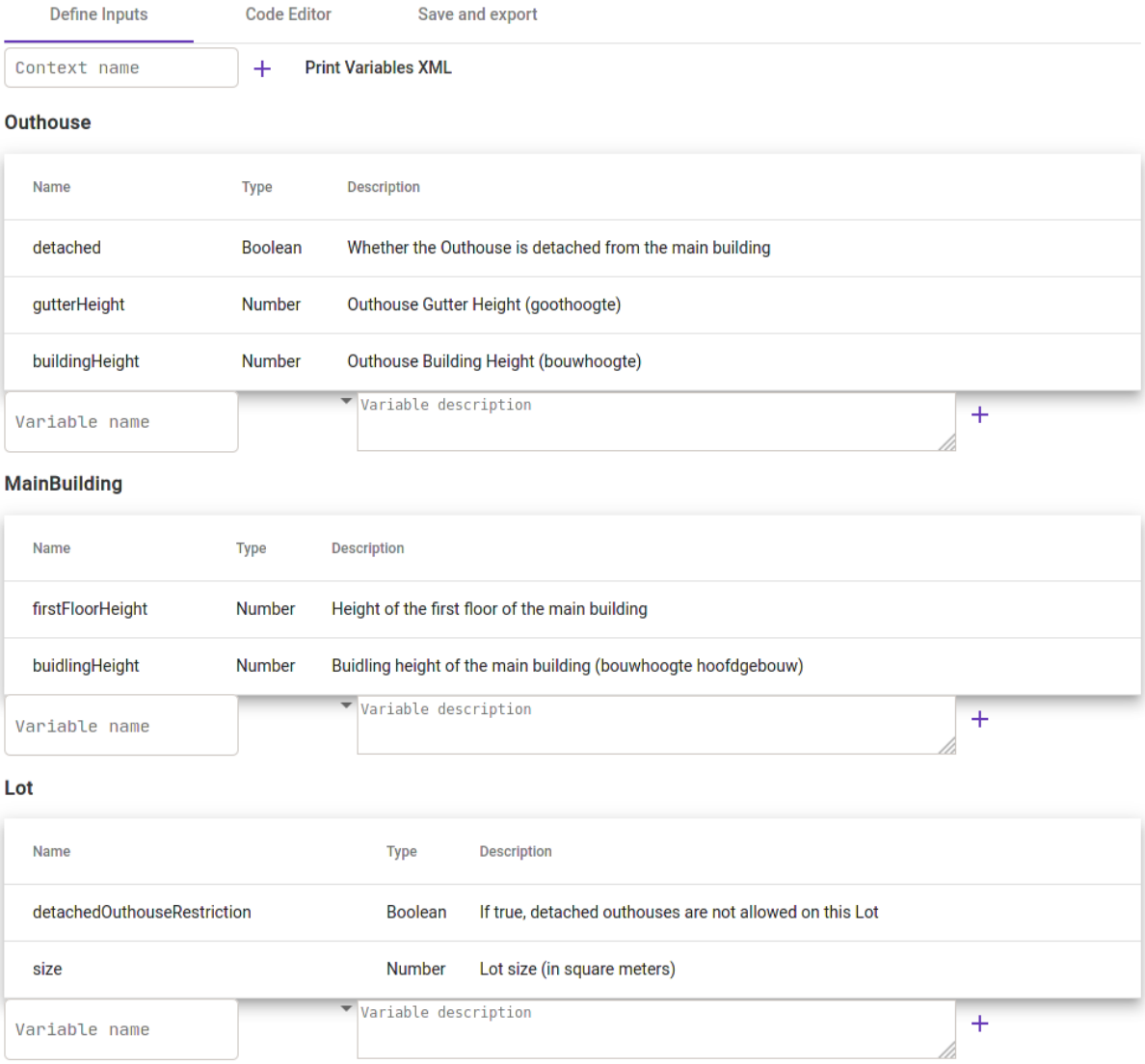
Het onderzoek en het ontwerp en de realisatie van het prototype zijn gedaan aan de hand van een fragment uit Chw bestemmingsplan Deventer, stad en dorpen deel C[7]. Hiernaar wordt ook verwezen in het verslag. De tekst van dit fragment is als volgt:

199.1.3 Aanvullende bouwregels aanbouwen, uitbouwen en bijgebouwen

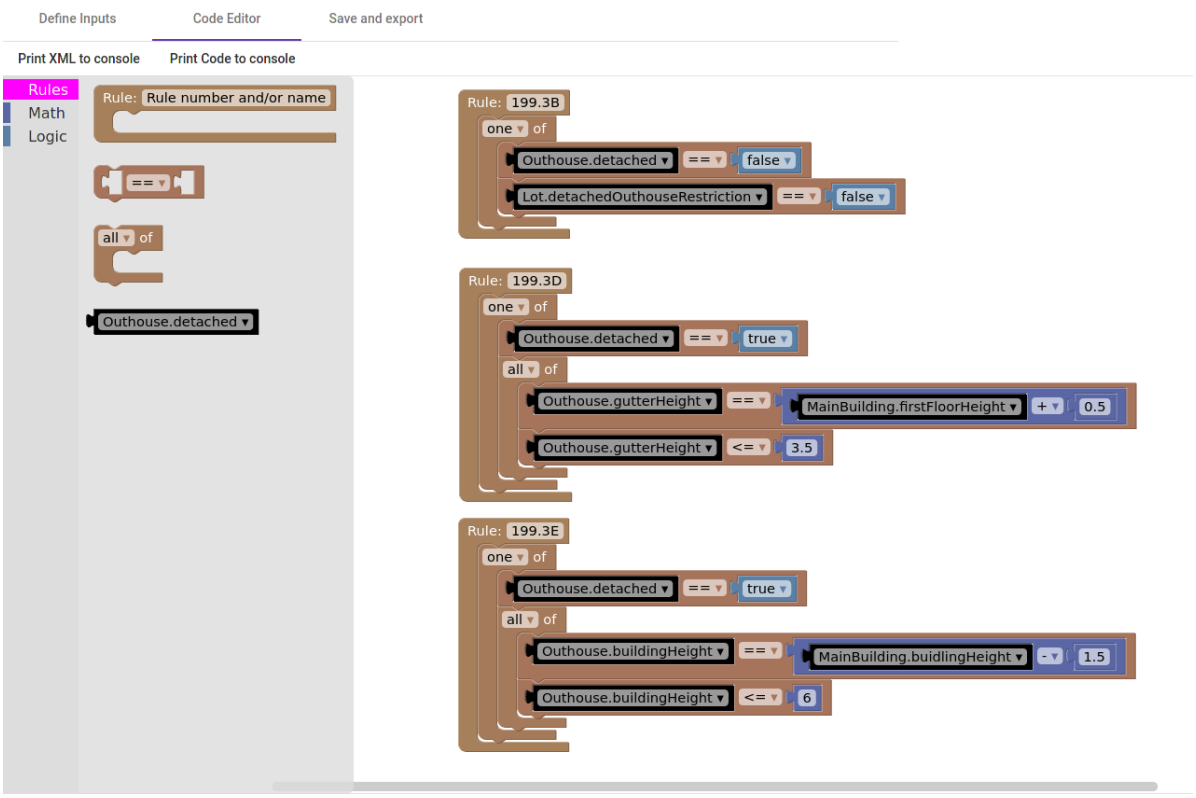
A. Aanbouwen, uitbouwen en bijgebouwen mogen uitsluitend worden gesitueerd ter plaatse van de aanduiding 'bouwvlak' en/of ter plaatse van de aanduiding 'bijgebouwen';
B. Ter plaatse van de aanduiding 'bijgebouwen uitgesloten' zijn geen vrijstaande bijgebouwen toegestaan;
C. Ter plaatse van de gebiedsaanduiding 'milieuzone - geluidsgevoelige functie' mogen geen aan-, uit- of bijgebouwen worden gerealiseerd ten behoeve van geluidgevoelige functies;
D. De goothoogte van aanbouwen, uitbouwen en aangebouwde bijgebouwen mag niet meer dan 3,5 m, of 0,50 m boven de vloer van de eerste verdieping van het hoofdgebouw, bedragen;
E. De bouwhoogte van aanbouwen, uitbouwen en aangebouwde bijgebouwen mag niet meer dan 6 m bedragen, met dien verstande dat deze ten minste 1,5 m is gelegen onder de bouwhoogte van het hoofdgebouw;
F. De goothoogte van vrijstaande bijgebouwen mag niet meer dan 3 m bedragen;
G. De bouwhoogte van vrijstaande bijgebouwen mag niet meer dan 5 m bedragen;
H. De afstand van aanbouwen, uitbouwen en bijgebouwen tot de onbebouwde zijdelingse bouwperceelgrens mag:
    1. Op bouwpercelen met een oppervlakte tot 300 m2, 0 m bedragen;
    2. Op bouwpercelen met een oppervlakte vanaf 300 m2 niet minder dan 1 m bedragen

[7] Chw bestemmingsplan Deventer, stad en dorpen deel C, artikel 199 – Bouwregel 16, te vinden op https://deventer.tercera-go.nl/mapviewer/default.aspx?id=NLIMRO0150Chw001C-VG01, geraadpleegd op 15 februari 2021

Bijlage B: Screenshots van het prototype



Figuur 4.1: Contexts voor het bijgebouw, hoofdgebouw en het perceel



Figuur 4.2: Implementatie van regels B, D en E