ARC-bus Plug-in

Ontwikkelen van een ARC-bus plug-in voor VDG Sense

Student: Aiken Brus (10007695)
Datum: 21-03-2016
Plaats: Zoetermeer
Afstudeerverslag Haagse Hogeschool
Opleiding: Technische informatica
Bedrijf: VD Security B.V.
Opdrachtgever: Timme Grijpink
Begeleidende examinator: Cobie van der Hoek
Tweede examinator: John Visser

Referaat

Titelbeschrijving

Aiken Brus, “ARC-bus Plug-in”. Afstudeerverslag opleiding Technische Informatica, Haagse Hogeschool, 2016.

Samenvatting

Dit document is een eindverslag gemaakt voor het afstuderen door Aiken Brus voor de opleiding Technische Informatica. Het afstuderen is gedaan bij het bedrijf VDG Security B.V. VDG Security is onderdeel van het technologie bedrijf TKH Group. VDG Security maakt een Video Management System (VMS) genaamd Sense. De opdracht is een simulatie tool te maken voor testen, C++ SDK en een ARC-bus plug-in te maken voor Sense. ARC-bus is een software matige bus waarmee berichten verstuurd kunnen worden tussen verschillende apparaten. Voor het ontwikkelen van de simulatie tool, C++ SDK en de plug-in is er gekozen voor de ontwikkelmethode Scrumban. Het ontwikkel proces is opgebouwd is negen sprints.

Descriptoren

Afstudeerverslag, ARC-bus, Plug-in, C++ SDK, simulatie tool.

Voorwoord

Mijn naam is Aiken Brus en ik heb dit verslag geschreven. Ik ben een student aan de Haagse hogeschool in Delft en volg de opleiding Technische Informatica. Dit verslag is gemaakt als een afstudeerverslag. Er is gekozen voor deze opdracht omdat het direct mijn interesse trok. Het onderwerp vind ik leuk en ik had de indruk dat ik hier veel van kon leren. Het is een software ontwikkel opdracht, waarin de software zelf geschreven moet worden. Vervolgens geïntegreerd moet worden met het bestaande systeem VDG Sense en dat ook nog via een software matige bus moet kunnen communiceren. Ik had nooit eerder aan een soortgelijke opdracht gewerkt. Dit zorgde voor veel leermogelijkheden.

Dit verslag is geschreven voor de examinatoren van het afstuderen. De examinatoren kunnen dit verslag gebruiken om te kijken of de student zijn afstudeerstage goed heeft afgerond. Dit verslag is ook nog geschreven voor de opdrachtgever en de developers van VDG Security B.V.

Ik zou graag de opdrachtgever en de algemeen directeur van VDG Security B.V. Timme Grijpink willen bedanken voor de mogelijkheid die hij me gegeven heeft om deze opdracht te kunnen doen. Daarnaast zou ik ook graag mijn bedrijf begeleider Robin Hermann willen bedanken voor de hulp die hij heeft gegeven tijdens mijn afstudeerstage. Ik wil ook alle andere collega's van Research and Development bedanken voor de leuke tijd die ik heb gehad tijdens mijn afstudeerstage. Als laatste zou ik graag Gianna Brus willen bedanken voor het door lezen van mijn verslagen en de hulp om een goed en grammaticaal correct verslag te schrijven. Deze hulp kon ik zeer goed gebruiken, aangezien ik een hoge graad van dyslectie heb en dus niet zozeer de inhoud maar de Nederlandse taal mijn grootste struikelblok is.

Aiken Brus
Zoetermeer
21-03-2016

Inhoudsopgave

1 Inleiding

Om de beelden van security camera's te bekijken en daar verschillende algoritmen op uit te voeren kan je gebruik maken van een video management system (VMS). VDG security B.V. maakt een VMS genaamd Sense. VDG is een onderdeel van het technologie bedrijf TKH Group. Onder de TKH Group vallen verschillende bedrijven die iets doen in de security branch. Keyprocessor maakt bijvoorbeeld slimme deursloten. Deze twee systemen kunnen aan elkaar gekoppeld worden door dat VDG een plug-in geschreven heeft. Een voorbeeld van de koppeling is, als iemand de deur probeert open te maken er een bericht van keyprocessor naar sense wordt gestuurd. Voor elk product dat gekoppeld zou worden aan Sense moet VDG een plug-in schrijven.

Om de samenwerking/integratie van verschillende producten te verbeteren heeft de TKH innovatieafdeling de ARC-bus edacht. De ARC-bus gaat gebruikt worden om de communicatie tussen de producten van de verschillende bedrijven te eneraliseren over één protocol. VDG wilt de software voorbereiden op de komst van de ARC-bus en dit integreren in hun nterne systeem VDG Sense. De ARC-bus is een nieuw concept en het is voor de VDG dan ook nog niet volledig duidelijk op elke wijze dit gedaan moet worden.

Een voorbeeld hoe de ARC-bus gebruikt gaat worden is als VDG, Sense en camera's levert en Keyprocessor slimme deursloten levert. Op het moment dat iemand de deur probeert open te maken wordt er een bericht van de key processor server niet naar Sense gestuurd maar naar de ARC-bus server. De ARC-bus server weet dat Sense dit soort berichten wilt ontvangen en stuurd deze dan door naar de Sense server. Sense kan met dit bericht bijvoorbeeld bepalen om de camera die op de deur gericht staat te laten zien aan de bewaker.

De doelstelling van de afstudeeropdracht is door middel van een onderzoek meer duidelijkheid te krijgen wat de ARC-bus is en wat deze kan. Met behulp van dit onderzoek worden requirements geïdentificeerd waarmee de ARC-bus geïntegreerd moet gaan worden in de VDG Sense software. Door middel van een plug-in moeten berichten over de ARC-bus uitgewisseld kunnen worden.

De hoofdvraag van dit project is: kan de ARC-bus gebruikt worden door VDG en kan de ARC-bus geïntegreerd worden in Sense.

In dit verslag wordt eerst verteld bij welk bedrijf het afstuderen gedaan wordt. In het volgende hoofdstuk wordt verteld wat de aanleiding was om dit project van start te gaan. Het probleem en doel bij dit project wordt ook verteld is dit hoofdstuk. Als laatst worden de resultaten beschreven. De ontwikkelmethode die is gekozen voor dit project is scrumban. Scrumban wordt ook gebruikt door VDG Security.

In sprint één oriëntatie werd er kennis gemaakt met het bedrijf en VDG Sense. In sprint twee is er onderzoek gedaan naar de ARC-bus. Hierin wordt behandeld wat de ARC-bus kan en hoe deze gebruikt kan worden? Uit dit onderzoek zijn er requirements ontstaan voor de plug-in. De derde sprint wordt gebruikt om het ontwerp te maken van de plug-in. Het ontwerp bestaat uit verschillende versies van klassendiagrammen en sequence diagrammen. Na het ontwerp kan er ontwikkeld worden. In sprint vier wordt de simulatie tool ontwikkeld. Dit is een tool die gebruikt gaat worden voor testen. In sprint vijf wordt de C++ SDK ontwikkeld. In sprint zes wordt de plug-in voor Sense ontwikkeld. Wanneer dit allemaal klaar is, kan er getest worden. Deze tests zijn onderdeel van sprint zeven. Er worden verschillende tests uitgevoerd. De unittests worden gedaan met GoogleTest. De integratie tests en systeem tests worden gedaan door middel van een scenario. Enkele van deze tests bleken onsuccesvol. In sprint acht wordt er gebruik gemaakt van de informatie die uit te test naar voren kwam. Zowel de successen als fallen zijn gebruikt om de code aan te passen zodat alle tests werkend werden. De laatste sprint is sprint negen. In deze sprint wordt alle documentatie afgemaakt. Het eindverslag afgerond, de user manual voor de C++ SDK en de bijlages.

Aan het eind van dit project is er een simulatie tool, een C++ SDK en een Sense plug-in gemaakt. Alles hiervan werkt. Alleen een deel van de koppeling tussen Sense en de plug-in is nog niet helemaal af. Dit is niet helemaal afgekomen omdat er een grote verandering gemaakt moest worden in Sense en deze verandering wou het bedrijf nu nog niet doorvoeren.

In het volgende hoofdstuk wordt nog iets meer informatie gegeven over VDG en TKG Group.

2 De organisatie

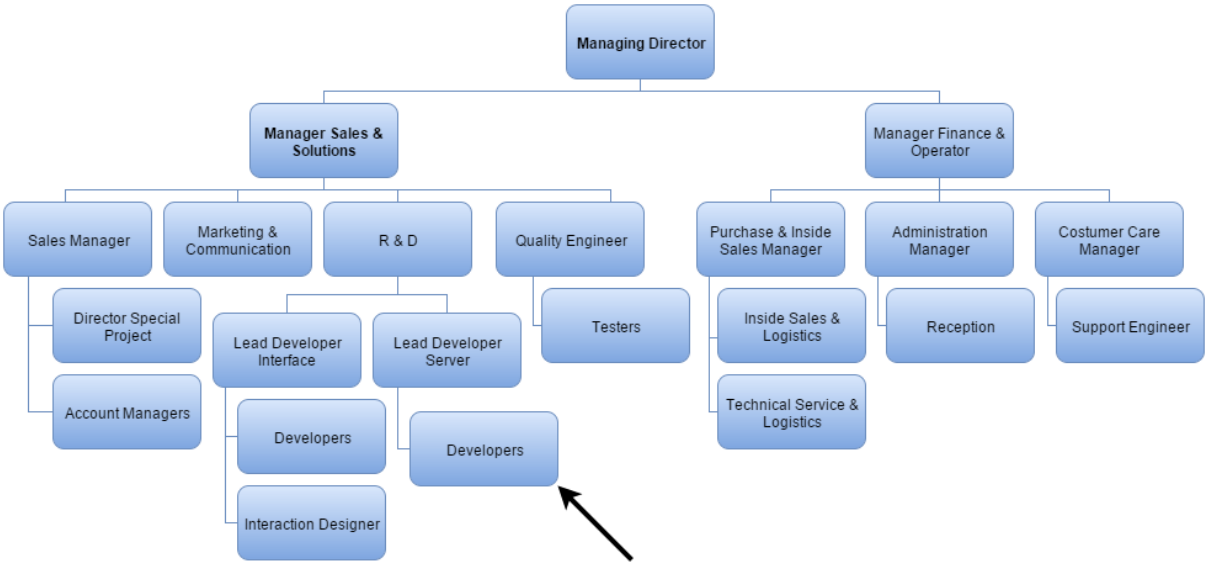
In dit hoofdstuk wordt beschreven bij welke organisatie de afstudeeropdracht is gedaan. Er wordt ook beschreven hoe deze organisatie in elkaar zit met afdelingen en hoeveel mensen er op elke afdeling werken. De opdracht wordt uitgevoerd in een van deze afdelingen en welke dat is wordt ook beschreven in dit hoofdstuk.

VDG Security B.V. staat voor Video Development Group Security B.V. VDG maakt innovatieve video management software oplossingen en heeft de applicatie VDG Sense ontwikkeld, dit is een Video Management System. Een Video Management System (VMS) wordt gebruikt om video's te monitoren, te analyseren en op te nemen. De focus van het bedrijf ligt voornamelijk op de security branch voor de verkeer infrastructuur, bouw en gezondheidszorg markt. VDG Security B.V. is redelijk plat en informeel. Je ziet wel wie de directeur is van het bedrijf maar hij gaat met iedereen hetzelfde om. Er is geen dress code.

VDG Security is een onderdeel van het technologie bedrijf TKH Group. De TKH Group bestaat uit een groot aantal bedrijven, met in totaal 5.337 werknemers. Een ander bedrijf dat onder TKH Group valt is KeyProcessor. De TKH Group bestaat uit de volgende drie hoofdgroepen: Telecom Solutions, Industrial Solution en Building Solutions.
VDG Security is werkzaam binnen Building Solutions. Binnen deze groep wordt nauw samengewerkt met de zuster bedrijven die ook actief zijn in de security branch. Waarbij zoveel mogelijk de technologie van de zuster bedrijven gecombineerd wordt met de video technologie van VDG Security. Het afstudeer project dat wordt uitgevoerd binnen het bedrijf VDG Security B.V. VDG bestaat uit verschillende afdelingen deze zijn te vinden in figuur 1 Organogram VDG Security B.V.

Het project is onderdeel van de afdeling Research and Development. In deze afdeling werken tien mensen. Deze afdeling is onderverdeeld in een groep van server- en client developers. Dit is de afdeling waar de afstudeerder werkzaam in is. Naast VDG Sense wordt er niets anders ontwikkeld door VDG Security B.V. Bij support werken vier man. Support levert telefonisch support en ook op locatie support. Bij verkoop werken vier man en in het magazijn werken drie man. VDG Security heeft drie testers in dienst. Deze testers testen VDG Sense. Elk onderdeel dat ontwikkeld is, wordt getest door één van deze testers. Naast VDG Sense verkoopt VDG Security B.V. ook nog verschillende hardware, waar bijvoorbeeld pc's, servers, encoders en camera's onder vallen. In figuur 1 vind je een organogram van VDG Security B.V. Deze organogram betreft alleen VDG Security en niet de hele concern TKH group. De stagiair doet de afstudeerstage in de afdeling waar het pijltje bij staat.

In het volgende hoofdstuk wordt beschreven hoe de opdracht ontstaan is en wat de opdracht inhoudt.



Figuur 1: Organogram VDG Security B.V.

3 De opdracht

In dit hoofdstuk wordt de opdracht geformuleerd waaruit dit afstudeeronderzoek ontstaan is. Tevens wordt de achtergrond en het ontstaan van de opdracht beschreven.

3.1 De aanleiding

Binnen de TKH group opereert een innovatieafdeling (TKH innovations). Deze houdt zich, onder andere, bezig met het bedenken van oplossingen waarmee de verschillende TKH ondernemingen en hun producten beter kunnen integreren. Projecten die door de TKH group uitgevoerd worden, bestaan in de meeste gevallen uit een samenstelling van één of meerdere TKH ondernemingen, die elk hun eigen product(en) leveren. Hierbij kan bijvoorbeeld aan een situatie gedacht worden waarbij VDG een camera-installatie met video analyse en opslag levert en KeyProcessor een toegangscontrole oplossing. Deze twee producten zullen vervolgens samen moeten gaan werken. Beide zijn een TKH onderneming die hun eigen hardware en software leveren.

Uit het verleden is gebleken dat het een intensieve klus is om de verschillende producten met elkaar te integreren. Hierbij komen een tweetal problemen met name naar voren:

- Elke TKH onderneming heeft in het verleden de vrije keuze gehad in welke techniek(en) en protocol(len) zij implementeerden in hun product(en) waarmee een extern systeem kan communiceren.
- De product- en softwareontwikkelingen staan niet stil en daarmee verandert dus ook regelmatig de wijze waarop met een product gecommuniceerd kan worden.

3.2 De probleemstelling

Om de samenwerking/integratie van verschillende producten op de middellange termijn te verbeteren, heeft de TKH innovatieafdeling de ARC-bus bedacht. Dit is een softwarebus waarmee het mogelijk gemaakt moet worden dat verschillende producten middels een netwerkverbinding, gegevens met elkaar kunnen uitwisselen op een uniforme manier. Een ontvanger kan zich abonneren op berichten die verstuurd worden over de ARC-bus en het is mogelijk om zelf berichten te versturen waar andere zich weer op kunnen abonneren. VDG wilt de software voorbereiden op de komst van de ARC-bus en dit integreren in hun interne systeem VDG Sense. De ARC-bus is een nieuw concept en het is voor de VDG dan ook nog niet volledig duidelijk op welke wijze dit gedaan moet worden.

3.3 De doelstelling

De doelstelling van de afstudeeropdracht is door middel van een onderzoek meer duidelijkheid te krijgen wat de ARC-bus is en wat deze kan. Met behulp van dit onderzoek worden requirements geïdentificeerd waarmee de ARC-bus geïntegreerd moet gaan worden in de VDG Sense software. Door middel van een plug-in moeten berichten over de ARC-bus uit gewisseld kunnen worden.

3.4 De resultaten

Een van de resultaten is een plug-in die als een middleware laag tussen de VDG Sense software en ARC-bus gaat fungeren. Naast de plug-in komt er een rapport waarin staat beschreven hoe de plug-in gebruikt kan worden. Via deze plug-in zal het mogelijk zijn om berichten over de ARC-bus uit te wisselen.

In het volgende hoofdstuk wordt beschreven welke ontwikkelmethode gekozen is om de opdracht te voltooien en hoe deze is gekozen.

4 De ontwikkelmethode

Om een project zo goed mogelijk uit te voeren is er gebruik gemaakt van een ontwikkelmethode. In dit hoofdstuk wordt uitgelegd welke ontwikkelmethode is gekozen en waarom deze is gekozen.

VDG maakt gebruik van de systeemontwikkelingsmethode Scrumban. Een projectopzet binnen VDG zou er als volgt uit kunnen zien: definitiestudie/analyse, basisontwerp, technisch ontwerp/detailontwerp, bouw, testen, integratie en beheer/onderhoud. Het lijkt op een watervalmethode en er wordt een Scrumban board gebruikt om het te visualiseren. Deze ontwikkelmethode zal ook worden gebruikt tijdens het project. Elke dag zal er een stand-up meeting gehouden worden met alle ontwikkelaars van VDG. Tijdens deze stand-up wordt besproken wat je de dag er voor hebt gedaan, wat je gaat doen vandaag en of je nog blokkades hebt.

Om te kijken welke ontwikkelmethode het beste is voor dit project zijn er eisen opgesteld waar de ontwikkelmethode aan moet voldoen. De eisen zijn opgesteld door te bestuderen wat de kenmerken van het project zijn. Tijdens het project worden er verschillende tussenproducten opgeleverd. De plug-in die ontwikkeld moet worden, moet geïntegreerd worden in de al bestaande software Sense. Het is nog niet zeker wat de ARC-bus allemaal kan. Hierdoor moet voor dit project een onderzoek gedaan worden naar de ARC-bus. De tussenproducten moeten getest worden. Het project duurt 20 weken. De eisen voor de ontwikkelmethode zijn:

- Het moet een softwareontwikkelmethode zijn.
- Er moet een mogelijkheid zijn om de product requirements tussentijds te kunnen aanpassen omdat het nog onzeker is hoe de ARC-bus gebruikt kan worden.
- De ontwikkel methode moet kunnen werken binnen de 20 weken die voor het project staan.
- Er moet een onderzoek element in zitten.

Tijdens het opstellen van de eisen is er gekeken naar de verschillende ontwikkelmethodes die bestaan en al bekend zijn bij de afstudeerder. Deze ontwikkelmethodes zijn:

- Rational Unified Process (RUP)
- De waterval methode
- Extreme programming (XP)
- Scrum
- Scrumban

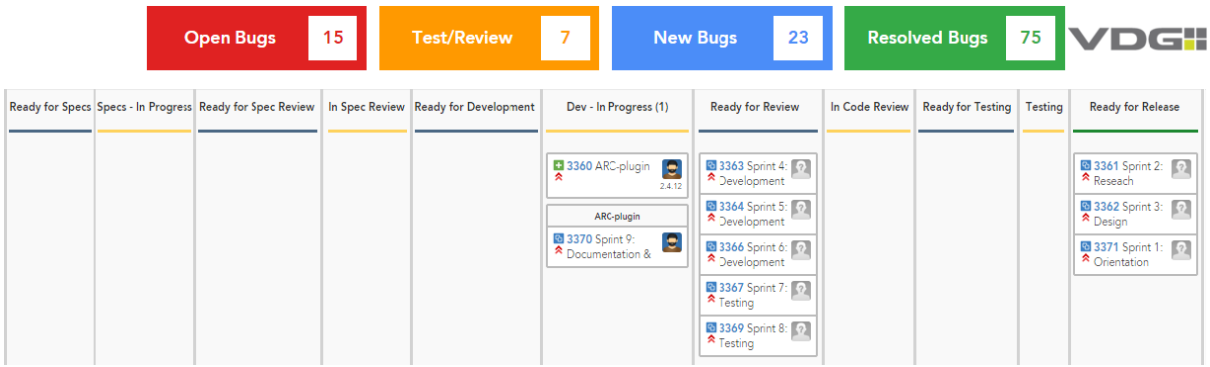
RUP is een ontwikkel methode voor systeemontwikkeling. Het maakt gebruik van iteraties en verschillende fases die je uitvoert in elke iteratie. Na elke iteratie is er een mijlpaal waar een onderdeel van het project af is. RUP is heel erg uitgebreid en is beter te gebruiken bij grote en langdurige projecten omdat RUP uitgaat van verschillende rollen binnen een project. RUP heeft voor verschillende onderdelen een 'best practice'. RUP maakt gebruik van vier fasen; inception, elaboration, consruction en transition. Elk van deze fasen kunnen onderverdeeld worden in één of meer iteraties. Om RUP te kunnen gebruiken voor dit project moet het eerst aangepast worden. In dit project kunnen sommige milestone documenten vergeten worden. In de inception fase kan de business case over geslagen worden. De business case is een document waarin beschreven wordt waarom het systeem gemaakt moet worden, wat het systeem moet gaan doen en er komt een financiëel overzicht in. Deze eerder genoemde onderdelen komen allemaal in het plan van aanpak te staan. Ook wordt het vision document niet gemaakt. In het vision document staat beschreven vanuit de ogen van de klant wat het product moet gaan doen en wat de risico's voor hun zijn. De prototypes van het architectuur kunnen nog niet gemaakt worden omdat er eerst onderzoek gedaan moet worden naar de ARC-bus. Tijdens de transition fase gaat er geen training gegeven worden over de werking van de plug-in. Het product wordt ook nog niet gedeployed. De verschillende disciplines worden in elke iteratie gedaan daar door kunnen de eisen van het project in elke iteratie worden aangepast.

De waterval methode is eenvoudig te gebruiken en te plannen. Alle eisen moeten in het begin bekent zijn en er is geen mogelijkheid om deze aan te passen later in het project. De waterval methode is ook niet iteratief.

Extreme programming is een iteratief ontwikkel methode. Tijdens deze ontwikkel methode is het mogelijk om de eisen aan te passen. Extreme programming heeft ook geen specificaties voor het ontwerpen van hardware.

Scrum maakt gebruik van sprints. Een sprint is een periode van een maand of minder en in deze periode wordt er aan een onderdeel van het project gewerkt dat aan het eind van de sprint af moet zijn. Scrum maakt gebruik van stand-up meetings. Dit betekent dat elke week of dag wordt besproken wat er gedaan is, wat er gedaan gaat worden en of er nog problemen ontstaan zijn. Tijdens scrum is het mogelijk om de eisen van het systeem aan te passen in het begin van elke sprint.

Scrumban is vergelijkbaar met Scrum. Dat betekent dat het ook gebruik maakt van sprints en stand-up meetings. Naast deScrum onderdelen maakt het ook gebruik van een 'kanban board'. Een 'kanban board' wordt gebruikt om alle statussen van de activiteiten bij te houden en dit zichtbaar te maken. figure 2 is de door VDG Security B.V. zelf ontwikkelde 'kanban board'.

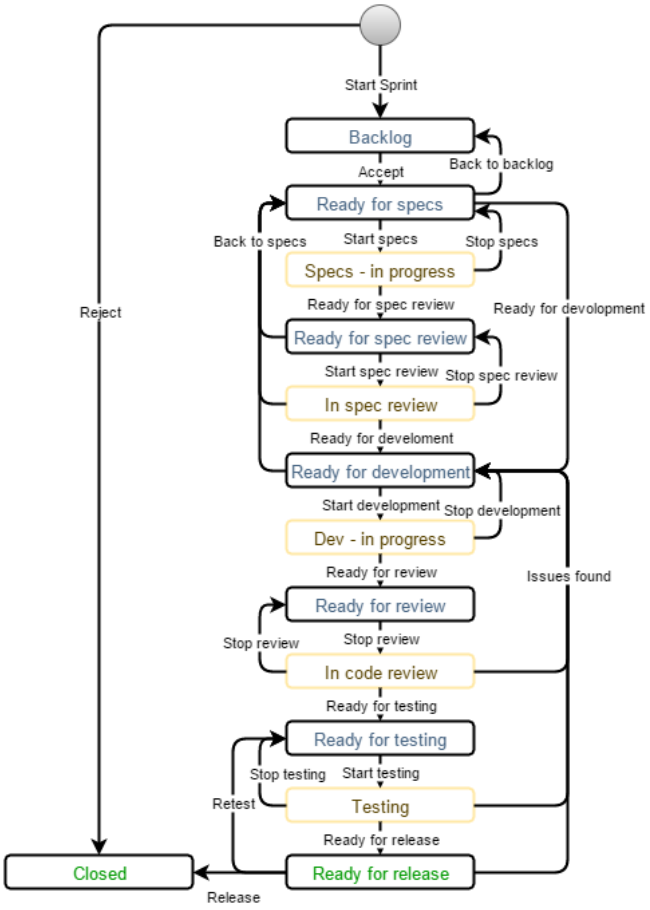


Figuur 2: Kanban board VDG Security B.V.[19]

De waterval methode valt af omdat deze niet flexibel is, aan het begin van de ontwikkeling moeten alle eisen bekend zijn en eerder kwam naar voren dat de eisen tussentijds aangepast moeten kunnen worden tijdens dit project. Er is geen mogelijkheid bij de waterval methode om een paar stappen terug te gaan en de eisen opnieuw te gaan verwoorden. Hiermee is deze methode dus niet geschikt. Extreme programming valt ook af omdat hier niet genoeg kennis van is. RUP, Scrum en Scrumban zijn alle drie goede mogelijkheden. Uiteindelijk is er gekozen om Scrumban te gebruiken want dit wordt ook gebruikt door VDG. Het project is onderverdeeld in negen sprints. Er is voor negen sprints gekozen omdat aan het eind van elke sprint tussen producten die onder de zelfde discipline vallen kan opleveren. Er is gekozen om de simulatie tool, C++ SDK en de plug-in in aparte sprints te doen omdat deze producten te groot zijn voor één sprint. Elke sprint is twee weken lang. Elke sprint volgt apart de workflow van figure 3 van start tot closed. In de backlog wordt van op geschreven wat er in de sprints gedaan gaat worden. Tijdens specs wordt worden de specificaties bepaald maar er wordt ook naar de al eerder gedachten specificaties gekeken. In spec review worden de specificaties door de bedrijfsbegeleider door genomen. Tijdens de stap Dev - in progress worden de tussenproducten van de sprint ontwikkelt. In code review wordt de gemaakte code gereviewd door de bedrijfsbegeleider. Bij de stap testing worden de tussenproducten getest. wanneer alles gedaan is komt het in ready for release. De workflow wordt ook gebruikt door het bedrijf.

Tijdens dit project worden er een paar standaard scrumban specificaties niet gebruikt. Er is geen WIP (work in progress) limit. WIP is de hoeveelheid taken die per activiteit gedaan kan worden. Er is geen WIP omdat het project door één persoon gedaan wordt en een persoon kan maar aan één taak tegelijkertijd werken. Scrumban kent twee verschillende meetings de dagelijkse stand-up en wekelijkse timeblock. In de dagelijkse stand-up verteld elke ontwikkelaar wat er de vorige dag gebeurt is, wat er vandaag gedaan gaat worden en of er nog blokkades zijn. De VDG Security developers hebben een retrospectieve na elke release.

In de eerste sprint is een oriëntatie sprint. In deze sprint wordt er ingewerkt en wordt het plan van aanpak gemaakt. In de tweede sprint wordt het onderzoek gedaan. Tijdens dit onderzoek wordt ook een 'proof of concept' ontwikkeld. Deze proof of concept wordt gebruikt om te onderzoeken wat de functionaliteit van de SDK is en hoe het is opgebouwd. SDK is een software development kit. In deze kit zitten de meeste hulpmiddelen om een programma te kunnen schrijven. Dit onderzoek geeft een onderzoeks-document als resultaat. In dit document staat beschreven waar de ARC-bus uit bestaat, hoe dit gebruikt kan worden en dit ook bruikbaar gaat zijn voor VDG Security B.V. Na het onderzoek van de ARC-bus is er een lijst met mogelijkheden van de ARC-bus deze mogelijkheden kunnen omgezet worden naar requirements voor de plug-in. In deze sprint wordt daarom ook nog een requirements document gemaakt. De derde sprint is de



Figuur 3: workflow sprints[18]

ontwerp sprint. Tijdens deze sprint wordt een design van de plug-in gemaakt. Na het ontwerpen wordt een testdocument geschreven. Vanaf sprint 4 word er ontwikkeld. Als eerste wordt er in sprint 4 een simulatie tool ontwikkeld. In dit testdocument komt te staan welke tests uitgevoerd gaan worden. Deze simulatie tool gaat alle onderdelen van de ARC-bus geïntegreerd krijgen. In sprint 5 en 6 gaat de plug-in ontwikkeld worden. Tegelijkertijd met het ontwikkelen gaan er unit test uitgevoerd worden. Sprint 7 gaat gebruikt worden om het hele product te testen met en zonder de simulatie tools. Na de test sprint kunnen er bugs te voorschijn zijn gekomen. In sprint 8 worden deze bugs weg ontwikkeld. De laatste Sprint is sprint 9, deze wordt gebruikt om aan het eindverslag te werken. Tijdens het project word er elke week aan het eindverslag gewerkt. In het volgende hoofdstuk zullen alle sprints uitgebreid toegelicht worden en beschreven worden hoe deze ingepast worden in dit onderzoek.

Sprint	Naam	Resultaat
Sprint 1	Oriëntatie	Plan van aanpak
Sprint 2	Onderzoek naar de ARC-Bus	• Onderzoeks-document • Requirements document
Sprint 3	Ontwerp simulatie tool, ontwerp C++ SDK en ontwerp plug-in	• Achitectuur • Testdocument
Sprint 4	Ontwikkelen simulatie tool	Simulatie tool
Sprint 5	Ontwikkelen C++ SDK	C++ SDK
Sprint 6	Ontwikkelen plug-in	Plug-in
Sprint 7	Testen	Test resultaten
Sprint 8	Bugs verhelpen en testen	Eindproduct
Sprint 9	documentatie en acceptatie	Eindverslag

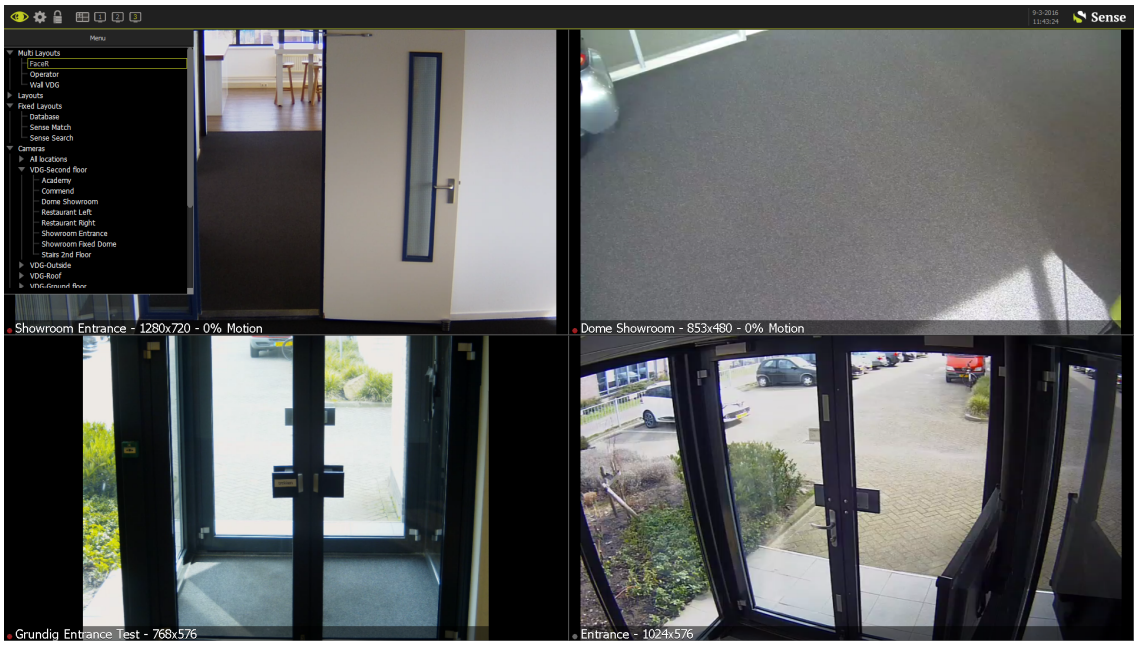
Tabel 1: sprints met naam en resultaten

Het volgende hoofdstuk is de oriëntatie. In de oriëntatie wordt sprint 1 beschreven. Tijdens sprint 1 is er kennis gemaakt met VDG Security B.V., het programma Sense dat door VDG ontwikkeld wordt. Tijdens deze sprint is er ook een plan van aanpak geschreven.

5 De oriëntatie (sprint 1)

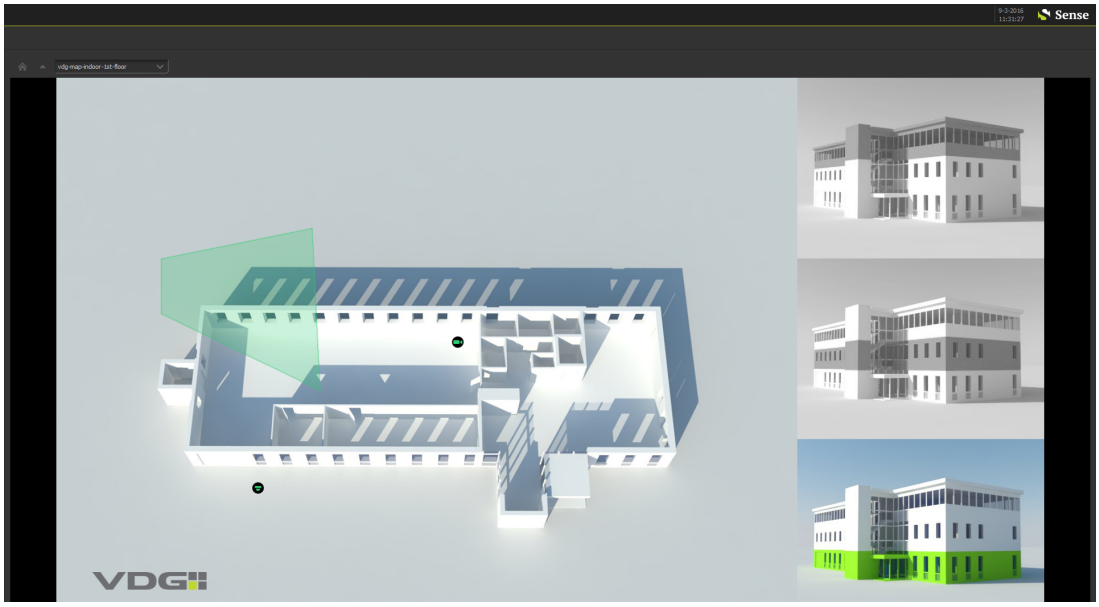
Sprint 1 is bedoeld om een beter inzicht te krijgen in het project. Aan het einde van sprint 1 moet een plan van aanpak gemaakt zijn. Tijdens de oriëntatie is er eerst ingewerkt. Tijdens het inwerken is er kennis gemaakt met de collega's bij VDG Security B.V. en is de computer geïnstalleerd met de benodigde software. Er is ook naar VDG Sense gekeken. Hierbij is niet naar de code gekeken maar naar het programma zelf. Nadat er gekeken was naar het programma zelf is al een beetje gekeken naar de architectuur van VDG Sense. VDG Sense is een video management system met een groot aantal functionaliteiten

Er zijn twee verschillende opties live modes of settings. In figure 4 is een simpele twee bij twee view. Links boven is de treeview. Met de treeview kunnen verschillende camera's op panels van layouts gestopt worden en kan je tussen layouts switchen. Een layout is de verzamel naam van alle panels op één beeldscherm.



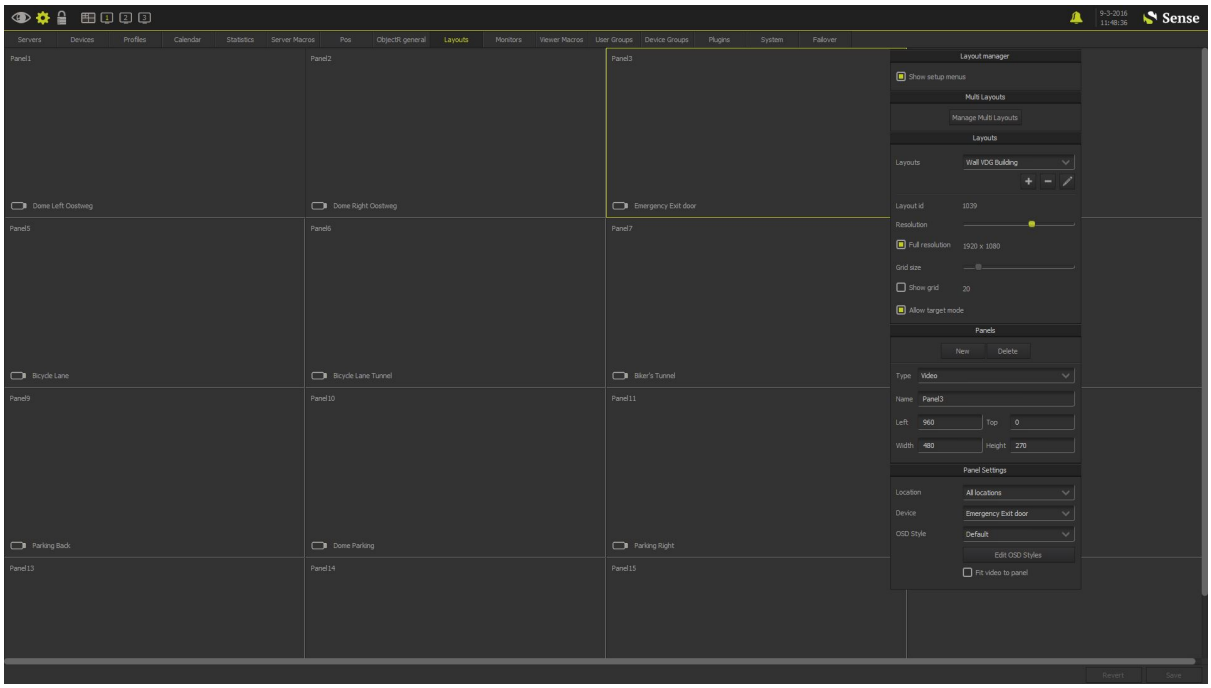
Figuur 4: live modes Sense[19]

Naast Panels met camera's er op is er ook een map panel. De map panel wordt bijvoorbeeld gebruikt om te laten zien waar de camera's in het gebouw staan. De camera's op de map panel kunnen gesleept worden naar een video panel en dan komt het beeld van die camera op de video panel.



Figuur 5: Map panel[19]

Naast beelden te laten zien, beschikt Sense ook nog over verschillende functionaliteit om videobeelden te analyseren (Video Content Analysis/VCA). Sense beschikt bijvoorbeeld over CarR om kentekenplaten van auto's te herkennen en FaceR om gezichten van personen te herkennen. ObjectR herkent objecten die bijvoorbeeld over een ingestelde lijn gaan of in bepaalde zones. Hier mee kan bijvoorbeeld herkent worden of een persoon bij het hek van het terrein is waar in een normale situatie niemand komt. SceneR herkent of het beeld van de camera niet veranderd is. Zodat er herkent kan worden of iemand de camera weg draait.
Door middel van het layout systeem kan de gebruiker zelf dynamisch bepalen hoe de layouts er uit komen te zien.



Figuur 6: Create layout menu[19]

Sense beschikt over een macrosysteem om automatisch acties uit te voeren op basis van door de gebruiker geconfigureerde condities. Een macro heeft minstens één conditie en één actie nodig om uitgevoerd te worden. Verder kan het over meerdere condities en acties beschikken. Wanneer een macro over meerdere condities beschikt, kan er in gesteld worden of de macro uitgevoerd moeten worden als alle condities voldaan zijn of als één van de condities waar zijn. Wanneer een macro over meerdere acties beschikt, worden alle acties van de macro uitgevoerd. De gebruiker kan macro's aanmaken in het macro systeem.

Tijdens het maken van het plan van aanpak is er een planning gemaakt met de sprints die genoemd worden in tabel 1. Elk van deze sprint duurt twee weken. Sprint één begint op 14 september. De inlever datum van het eindverslag is 21 maart 2015. Door Sense te bestuderen tijdens de oriëntatie kon er gekeken worden of er nog extra tijd nodig was om een mini cursus te krijgen hoe Sense werkt.

In het volgende hoofdstuk wordt sprint 2 besproken. In sprint 2 is er een onderzoek gedaan naar de ARC-bus. In dit hoofdstuk staat wat er onderzocht gaat worden en wat de uitkomst van het onderzoek is.

6 Het onderzoek naar de ARC-Bus (sprint 2)

Tijdens sprint 2 is er onderzoek gedaan naar de werking van de ARC-bus. Na het onderzoek kunnen de eisen van het product opgesteld worden. Aan het einde van sprint 2 zijn er twee tussenproducten die opgeleverd kunnen worden. Deze zijn: een onderzoeksrapport en een requirements document. Het onderzoeksrapport kan gevonden worden als bijlage 4 en het requirements document kan gevonden worden als bijlage 5.
Voor dat het onderzoek begon zijn er een paar vragen opgesteld waarvan het antwoord uit het onderzoek zou moeten komen. Deze vragen zijn opgesteld door te overleggen met de begeleider wat er allemaal onderzocht zou moeten worden. De vragen die opgesteld zijn:

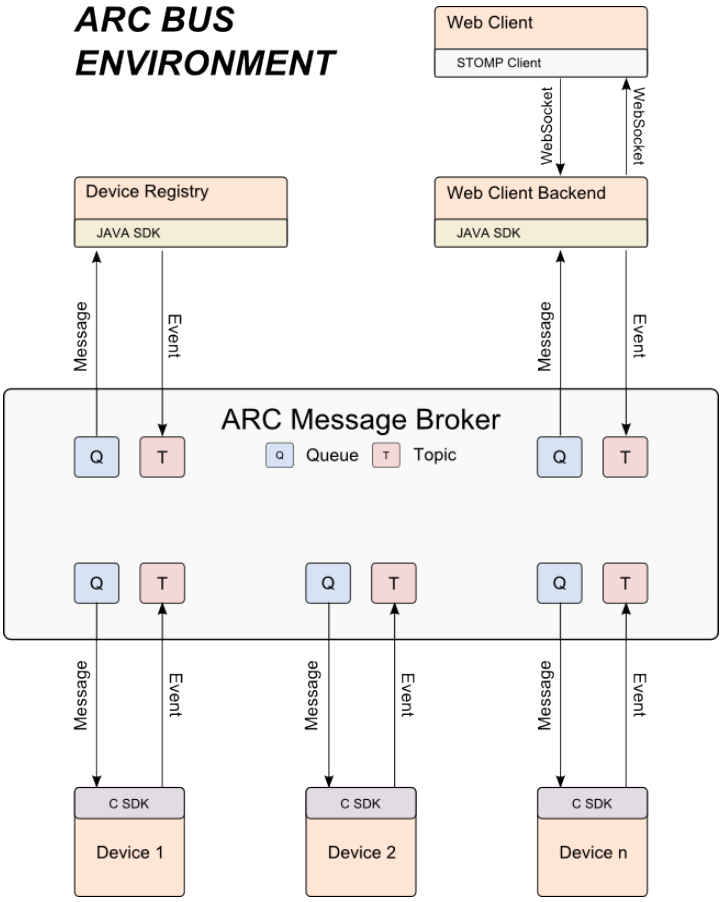
- Wat is de ARC-bus?
- Welke protocol wordt er gebruikt?
- Uit welke onderdelen bestaat de ARC-bus
- Hoe kan de ARC-bus gebruikt worden
- Wat zijn de toekomst plannen voor de ARC-bus
- Kan de ARC-bus gebruikt worden door VDG Security B.V.

Voor het onderzoek is er gekeken naar verschillende aspecten. Als eerste naar de meegeleverde documentatie en als tweedewordt er gekeken naar de code van de C SDK. In deze documentatie staat welke protocollen en libraries worden gebruiken door de ARC-bus en de SDK.

6.1 Message Broker

De ARC-bus is een software architectuur model waar gedeelde communicatie is tussen verschillende devices. De software architectuur wordt geïmplementeerd door via de message broker de communicatie te laten gaan. Deze message broker bepaald voor elk binnengekomen bericht naar welke devices het gaat. figuur 7: is een schematische afbeelding van de ARC-bus architectuur. De message broker is geschreven in Java. De ARC message broker is een wrapper om ActiveMQ[4] van het bedrijf Apache. De ARC message broker gebruikt dus de functionaliteiten van ActiveMQ. Voordat TKH Innovations gekozen heeft voor ActiveMQ is er eerst naar andere message brokers gekeken zijn.
Een paar specificaties van ActiveMQ zijn:

- Cross platfrom
- Cross language voor clients
  - Java
  - C
  - C++
  - C#
  - Ruby
  - Perl
  - Python
  - PHP
- ondersteund 4 verschillende protocols
  - Openwire
  - Stomp
  - AMQP
  - MQTT



Figuur 7: ARC-bus architectuur[6]

6.2 SDK

Als message protocol wordt AMQP gebruikt. AMQP is een application layer protocol voor message-oriented middleware. application layer protocol is een protocol dat gebruikt wordt voor de application layer in het iso model. AMQP is een open standard. De developers van TKH innovation hebben gekozen om Qpid Proton te gebruiken voor hun SDK. Qpid Proton[5] is een library waarmee AMQP berichten kunnen worden verzonden en ontvangen. Ook hierbij is er eerst gekeken naar andere protocollen.

Op het moment is er alleen een SDK in C en Java. De volgende functies zitten al in de SDK. SDK oftewel Software Development Kit is een set van software tools die nodig zijn om een bepaalde applicatie te kunnen ontwikkelen[20].

- Bericht verzenden
- Bericht ontvangen
- Registeren op requests
- Registeren op events
- Data aan een bericht toevoegen
- Een bericht uitlezen

De hier boven benoemde functies zijn getest door middel van een ‘proof of concept’ te maken. De proof of concept is een product dat de boven genoemde functies kan uitvoeren. Dit is getest door het product twee keer op te starten en dan berichten naar elkaar te sturen.

Nadat er verschillende punten onderzocht waren is er een beter idee gekregen van hoe de ARC-bus en de C SDK werkten. Daarna was de lead developer van de ARC-bus langs gekomen bij VDG om in zijn woorden te vertellen wat de ARC-bus op dit moment is en zijn visie voor de toekomst. Een paar onderdelen die nog ontwikkeld worden in de toekomst zijn:

- Message brokers aan elkaar koppelen
- C# SDK
- UPnP[17]
- Een filter bij de Get devices methode

UPnP is een set van netwerkprotocollen die netwerkapparaten mogelijk maakt elkaars aanwezigheid te ontdekken op het netwerk en stellen een netwerk service in voor het delen van gegevens, communicatie en entertainment. UPnP kan gebruikt worden in de ARC-bus door de message broker en devices elkaar automatisch te laten vinden binnen een netwerk.

Nadat dit onderzoek besproken was met de begeleider had deze nog een paar vragen die nog niet beantwoord waren in het document. De antwoorden van deze vragen stonden ook niet in de documentatie van de ARC-bus.
De vragen die gesteld waren door de begeleider zijn:

- Is het mogelijk om meerdere message brokers aan elkaar te koppelen?
- Is er al een installer om de ARC-bus message broker mee te installeren?
- Hoe zit het met de DeviceRegistry, waar gaat de DeviceRegistry voor gebruikt worden?

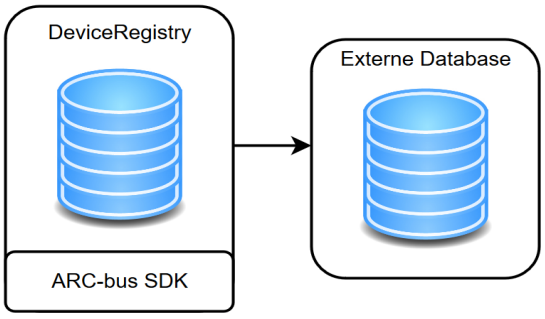
Op het moment is het nog niet mogelijk om meerdere message brokers aan elkaar te koppelen. TKH innovation is wel van plan om dit in de toekomst mogelijk te maken.

Op het moment is er nog geen installer voor de ARC-bus message broker en op het moment stond het nog niet in de planning om een installer te maken, maar als er één door VDG gemaakt zou worden zouden ze die graag willen hebben. Een installer is nodig zodat als de sense server geïnstalleerd wordt ook gelijk de message broker geïnstalleerd kan worden.

6.3 DeviceRegistry

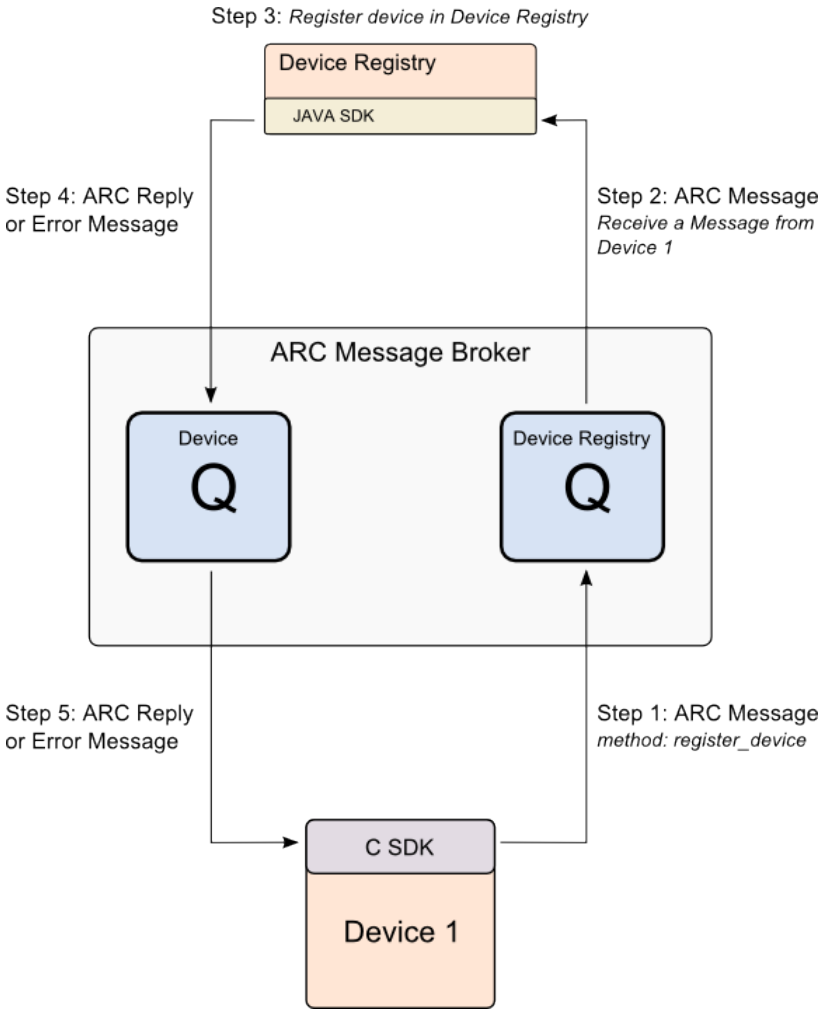
De DeviceRegistry is een aparte server die naast de message broker draait. DeviceRegistry is een java applicatie die een database in zich heeft en een externe database kan gebruiken. In de database van de DeviceRegistry komen alleen de devices te staan die zichzelf hebben geregistreerd bij de DeviceRegistry. In figuur 8 wordt de DeviceRegistry schematisch afgebeeld. De DeviceRegistry heeft verschillende functionaliteiten die aangeroepen kunnen worden. Deze zijn:

- Device registreren in de database.
- Device uit de database halen.
- Alle devices opvragen die in de database zitten.
- De informatie van het device update in de database.
- Een uuid op vragen.
- De persistent value op true zetten.

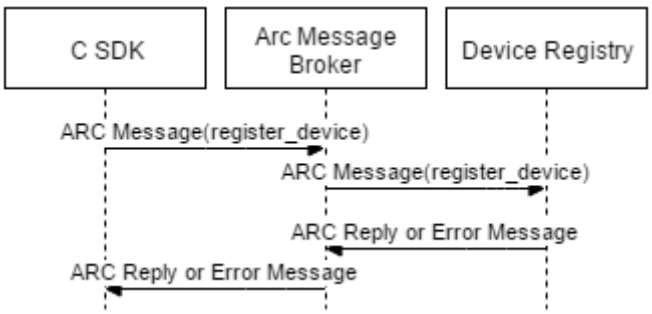


Figuur 8: schematisch afbeelding DeviceRegistry

Figuur 9 toont hoe de bericht verloop gaat bij het registreren via de C SDK. Als een apparaat zich registeert bij de DeviceRegistry dan komen zijn settings zoals uuid, IP-adressen en naam in de database te staan. Eerst wordt de message naar de ARC message broker gestuurd. De message broker ziet dat het voor de deviceregistry is en stuurt het door. De deviceregistry verwerkt het bericht en stuurt een reply als het goed is gegaan of een error als het fout is gegaan naar de message broker. De message broker stuurt het weer door naar device 1. In figuur 10 is een sequence diagram voor het versturen van een register message.



Figuur 9: Register device[6]



Figuur 10: Sequence diagram register DeviceRegistry

Aan het eind van het onderzoek bleek de ARC-bus redelijk goed te werken met de proof of concept. Met de proof of concept is het opstellen van berichten, registeren bij de "DeviceRegistry", verzenden en ontvangen van berichten getest. Deze tests zijn uitgevoerd omdat dit de hoofd functionaliteit van de ARC-bus is. Er waren nog een paar functies die niet werkten. De ARC-bus kan gebruikt worden voor de communicatie tussen de verschillende Sense servers. Er moet alleen nog wat verder ontwikkeld worden aan de ARC-bus omdat zoals eerder vermeld een paar functies het niet deden.

6.4 De requirements

De eisen zijn verzameld door in gesprek te gaan met de lead developer van VDG security, developer van de ARC-bus en door een onderzoek te doen naar de werking en specificaties van de ARC-bus. Deze eisen zijn SMART opgesteld. De requirements zijn onderverdeeld in verschillende categorieën. Deze categorieën zijn:

- Functionele eisen, dit zijn de functies waar het product aan moet voldoen.
- Usability eisen, dit zijn de eisen om het voor de gebruiker makkelijker te maken om de plug-in te gebruiken.
- Technische eisen, dit zijn de eisen waar het product aan moet voldoen op een technisch vlak.

Een paar belangrijke functionele eisen zijn:

- De plug-in moet request berichten kunnen verzenden naar de ARC-bus.
- De plug-in moet event berichten kunnen verzenden naar de ARC-bus.
- De plug-in moet ontvangen berichten kunnen converteren naar een Sense bericht.
- De plug-in moet zich bij de ARC-bus kunnen abonneren op event berichten van elk apparaat apart.
- De plug-in moet zich registeren bij de DeviceRegistry na het aanzetten van de plug-in.

De technische eis is belangrijk. Deze is:

- De plug-in moet asynchrone berichten kunnen verzenden en ontvangen.

Aan het eind van deze sprint zijn inderdaad de eerder vernoemde documenten opgeleverd. Meer informatie over het onderzoek kan je vinden in bijlage 4. Tijdens het onderzoek zijn alle onderzoeksvragen beantwoord. Voor de rest van de requirements kan er gekeken worden in het requirements document in de bijlage 5.

In hoofdstuk 7 wordt het ontwerp van de simulatie tool, C++ SDK en de plug-in besproken

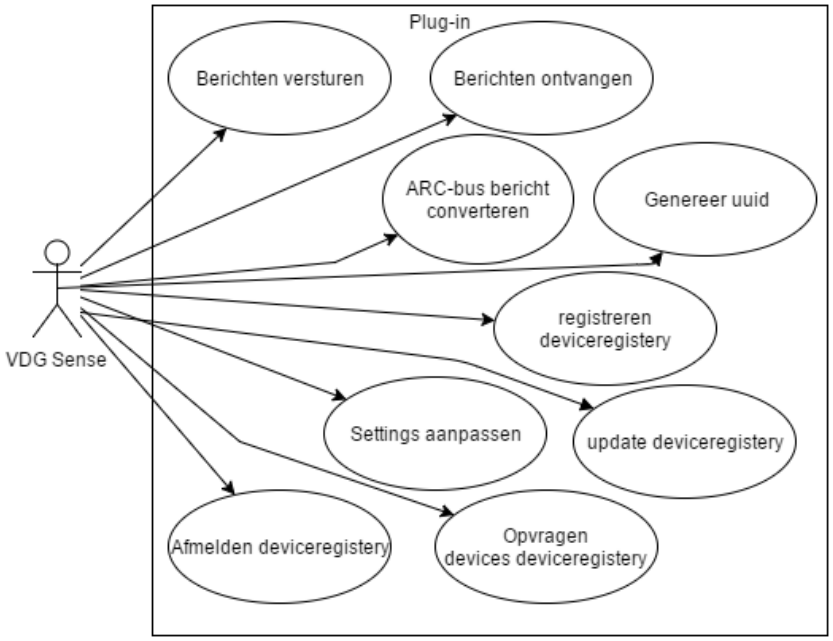
7 Het ontwerp (sprint 3)

Deze sprint is net als elke andere sprint onderverdeeld in twee weken. Tijdens de ontwerpfase wordt de architectuur van het systeem bedacht en opgeschreven. Voor het ontwerpen van de architectuur word UML gebruikt.

Als eerste worden er use-cases gemaakt. De architectuur van de plug-in wordt uitgeschreven door middel van een klassendiagram. Als het design klassendiagram gemaakt is, kunnen er sequence diagrammen gemaakt worden. Een sequence diagram toont de object interacties. Als laatste in de ontwerpfase wordt er een testdocument gemaakt. Deze wordt als laatst gemaakt omdat de UML modellen hiervoor nodig zijn. Aan het eind van deze sprint moeten de UML diagrammen en de testdocument gemaakt zijn. In dit test document staan de tests om alle functionele eisen en niet functionele eisen te testen.

7.1 Ontwerp plug-in

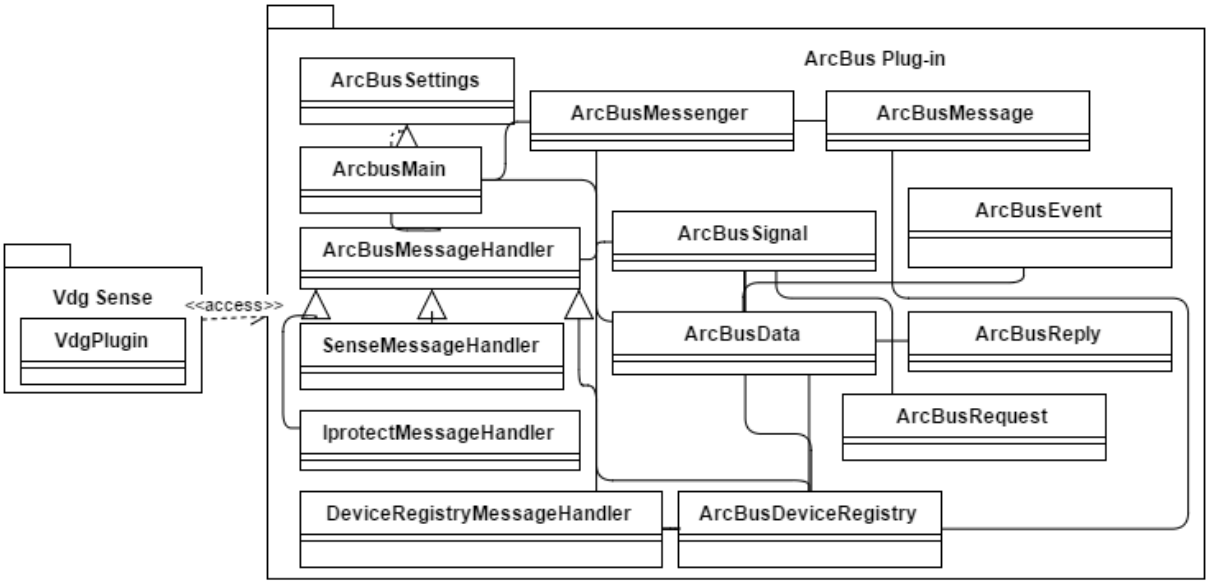
Ook bij het ontwerp van de C++ SDK is eerst de use-case gemaakt. figuur 11 is de use-case diagram die gemaakt is voor de plug-in. Er is geen use-case gemaakt voor elk verschillend bericht die de plug-in kan verzenden, omdat de functie voor VDG Sense en de gebruikers van de simulatie tool niet uitmaakt. In tabel 2 vind je een use-case beschrijving van de use-case "Berichten versturen". De rest van de use-case beschrijvingen kunnen gevonden worden in de bijlage requirements document.



Figuur 11: Use-case diagram plug-in.

Naam	Berichten versturen
Aanleiding	VDG Sense wilt een event sturen naar de ARC-bus
Actoren	VDG Sense
Doel	Een bericht verzenden naar de ARC-bus
Beschrijving	1. Request of Event Bericht wordt gemaakt. 2. Inhoud wordt toegevoegd aan het bericht. 3. Bericht wordt verzonden.
Resultaat	Een bericht is gestuurd naar de ARC-bus

Tabel 2: use-case beschrijving Berichten versturen



Figuur 12: globale klassendiagram plug-in versie 1

De volgende stap van het ontwerp van de plug-in was een klassendiagram maken. Dit klassendiagram is er voor om een beter en dieper inzicht te krijgen van de architectuur van het systeem en de bij behorende modules. Dit klassen diagram is ook gemaakt met de UML standaard voor de compatibiliteit met de andere diagrammen. Er zijn twee versies van dit diagram. Een globale klassendiagram van versie één kan gevonden worden als figuur 12 en een volledige klassendiagram is te vinden als bijlage 1.1. Dit klassendiagram is onderverdeeld in vier packages. Er is gekozen voor deze packages omdat de eerst 3 punten al ontwikkeld waren. De plug-in wordt een aparte package omdat dit in de form van een library ingeladen wordt in Sense.

- QpidProton, dit is een library van een derde partij.
- ARC-bus C SDK, dit is een SDK van TKH innovations
- VDG Sense, dit is het product waar de plug-in voor geschreven moet worden.
- ARC-bus plug-in, deze plug-in moet ontwikkelt worden voor het afstuderen.

Deze eerste versie is gereviewd met de bedrijfsbegeleider. Uit dit gesprek kwamen nog een paar punten.

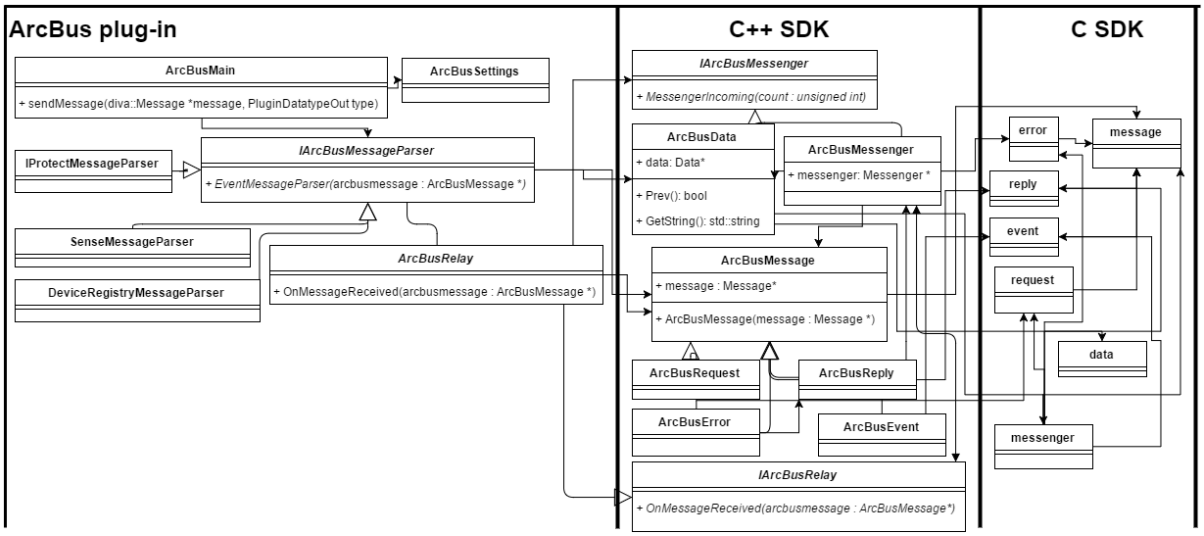
De plug-in moest in tweeën gesplitst worden. Het eerste deel moest een C++ SDK worden die de huidige ARC-bus C SDK implementeert. Het andere gedeelte moest de plug-in worden. De package moet opgesplitst worden, zodat de C++ SDK hergebruikt kan worden als een library door andere bedrijven binnen de TKG group.

Er moet een interface komen van de ArcBusMessenger. Zodat deze gebruikt kan worden door de producten die de C++ SDK implementeren.

De ArcBusMessageHandler kan beter hernoemd worden, want de klassen handelt niet de berichten, maar doet alleen iets met de inhoud van de berichten.

De ArcBusSignal klassen kon weg en moest vervangen worden door een interface in de C++ SDK genaamt IArcBusRelay. In de implementatie van deze klassen kan bepaald worden welk bericht naar welke parser moet.

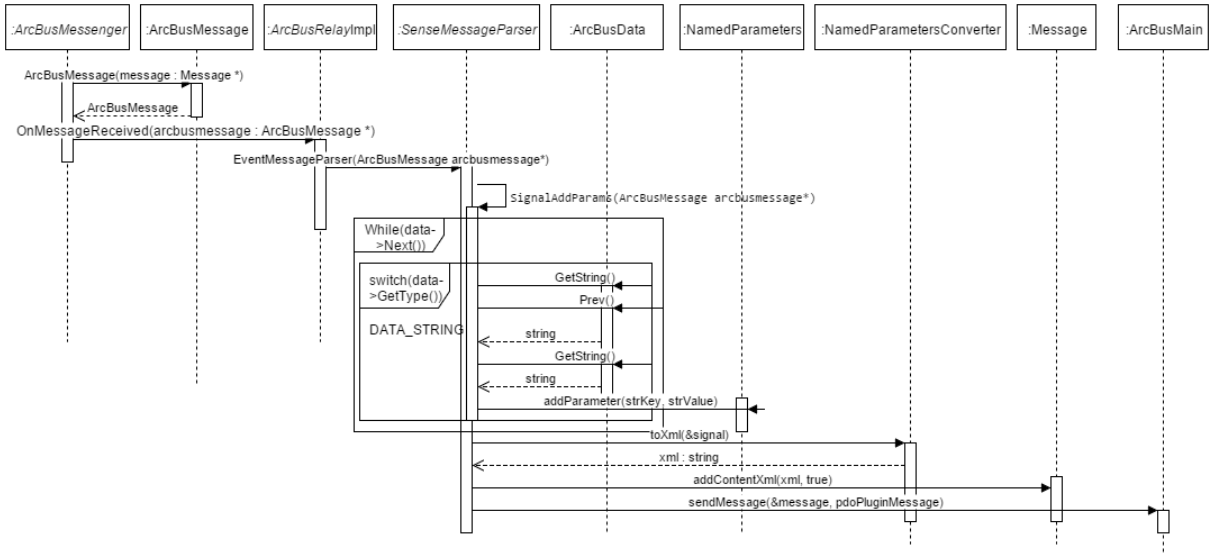
Uit de review kwam naar voren dat er naast de plug-in ook een C++ SDK gemaakt moesten worden, zodat dit hergebruikt kon worden door andere bedrijven binnen de TKH group. Er moet ook een simulatie tool ontwikkeld worden waarmee gelijk een gedeelte van de C++ SDK getest kon worden. Deze simulatie tool wordt vrijgegeven aan de ARC-bus developers. De simulatie tool werkt gelijk als een voorbeeld voor het integreren van de C++ SDK. Dit zijn nieuwe eisen geworden voor het afstuderen.



Figuur 13: globale klassendiagram plug-in versie 2

Nadat de tweede versie van het klassendiagram af was, zijn er sequentiediagrammen gemaakt. Er is gekozen om niet voor elke use-case een sequence diagram te maken, omdat sommige functie aanroepen heel erg op elkaar lijken waardoor je anders een paar sequence diagrammen krijgt die er hetzelfde uitzien. Voor de sequence diagram van berichten verzenden en berichten ontvangen is er gekozen om alleen voor een event bericht een diagram te maken. De sequence diagrammen kan je vinden in bijlage 1.4 tot en met bijlage 1.5.

Figuur 14 is een sequentie diagram voor het ontvangen van een event bericht. In deze sequentie diagram is het stukje van de C SDK niet ontworpen omdat dit gedeelte niet ontwikkelt wordt tijdens deze afstudeer opdracht. Op het moment dat de "ArcBusMessenger" een bericht binnen krijgt is dit een C bericht deze zet om naar een C++ "ArcBusMessage". Er wordt een functie call gedaan op OnMessageReceived met de C++ "ArcBusMessage" als parameter. In deze functie wordt bepaald wat voor bericht het is en naar welke parser het moet. In de parser wordt het bericht omgezet naar een XML bericht zodat het naar Sense gestuurd kan worden.



Figuur 14: Plug-in Receive event message versie 2

7.2 Ontwerp C++ SDK

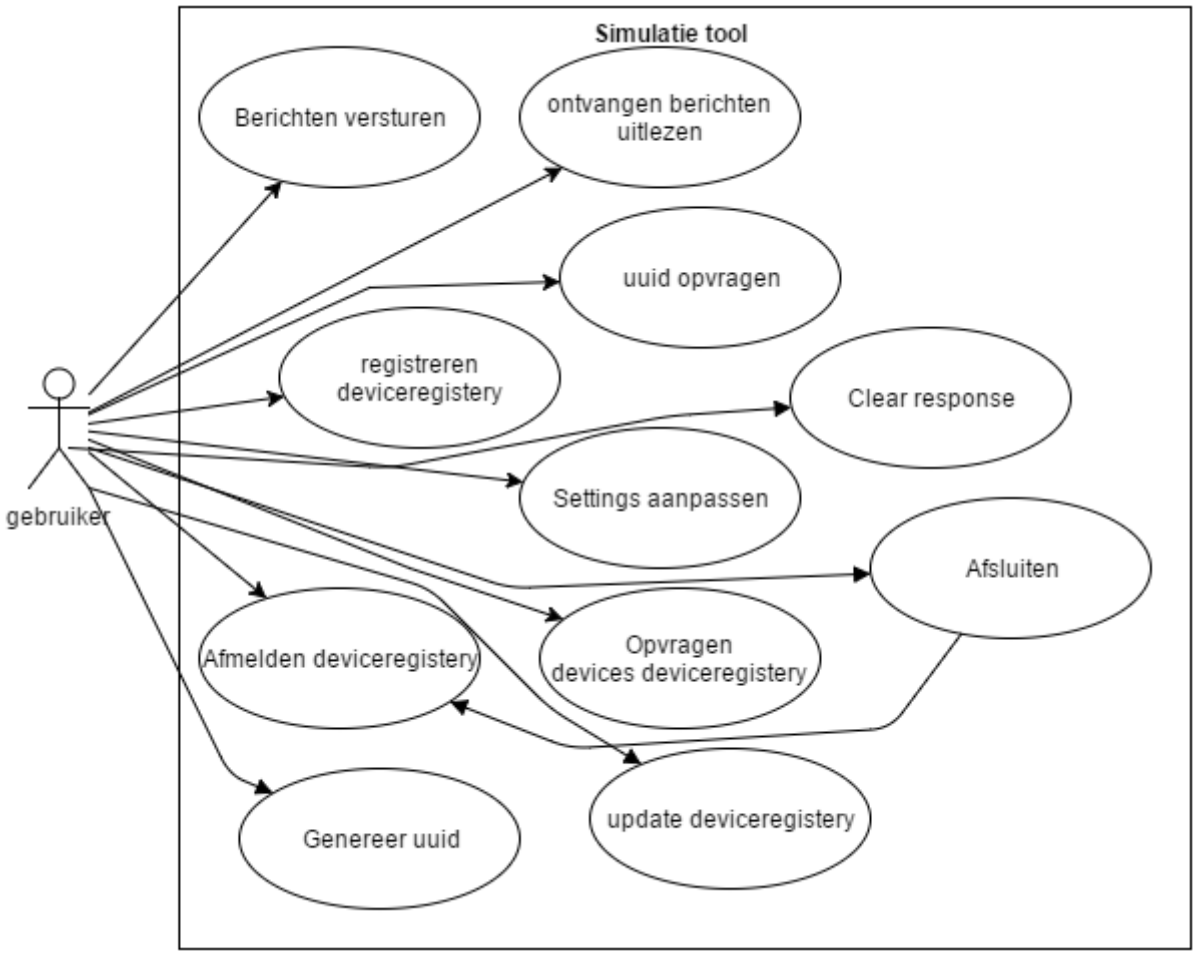
Uit de punten die naar boven zijn gekomen tijdens het bespreken van de eerst versie van het klassendiagram is versie twee ontstaan. In deze versie gebruikt de C++ SDK de C SDK functies om berichten te kunnen verzenden. De C++ SDK zorgt er voor dat de C code in een object georiënteerde manier gebruikt kan worden. Hier door ontstaan de klassen om een request, event, error of reply message te maken. In de C SDK bestaat er maar één soort message. Er zijn wel functies gemaakt die bedoeld zijn voor de verschillende messages en functies die gebruikt worden door elke message. In C++ kan dit vertaald worden naar een base klassen ArcBusmessage en vier afgeleiden klassen ArcBusRequest, ArcBusEvent, ArcBusError en ArcBusReply. Om de C++ SDK te gebruik moet er voor de ArcBusReplay interface een implementatie geschreven worden in de plug-in. Deze classe registeer je dan bij de ArcBusMessenger er dan wordt de functie "OnMessageReceived" aangeroepen als er een bericht binnen komt. In de plug-in kan het bericht uitgelezen worden en kan het omgezet worden naar een signal die Sense kan gebruiken. Versie twee kan gevonden worden als bijlage 1.2. Een globale klassendiagram van versie 2 is te vinden als 13. De ArcBusMessenger klassen maakt gebruik van de singleton design pattern. Dit design pattern word gebruikt zo dat er maar één object van de klassen gemaakt kan worden. Hierdoor worden de berichten maar van uit één klasse verzonden en ontvangen hierdoor is de bericht verwerking eenvoudiger te gebruiken.

Voor het ontwerp van de C++ SDK zijn er geen sequentie diagrammen gemaakt omdat de functie aanroepen in de sequentie diagrammen van de plug-in zitten.

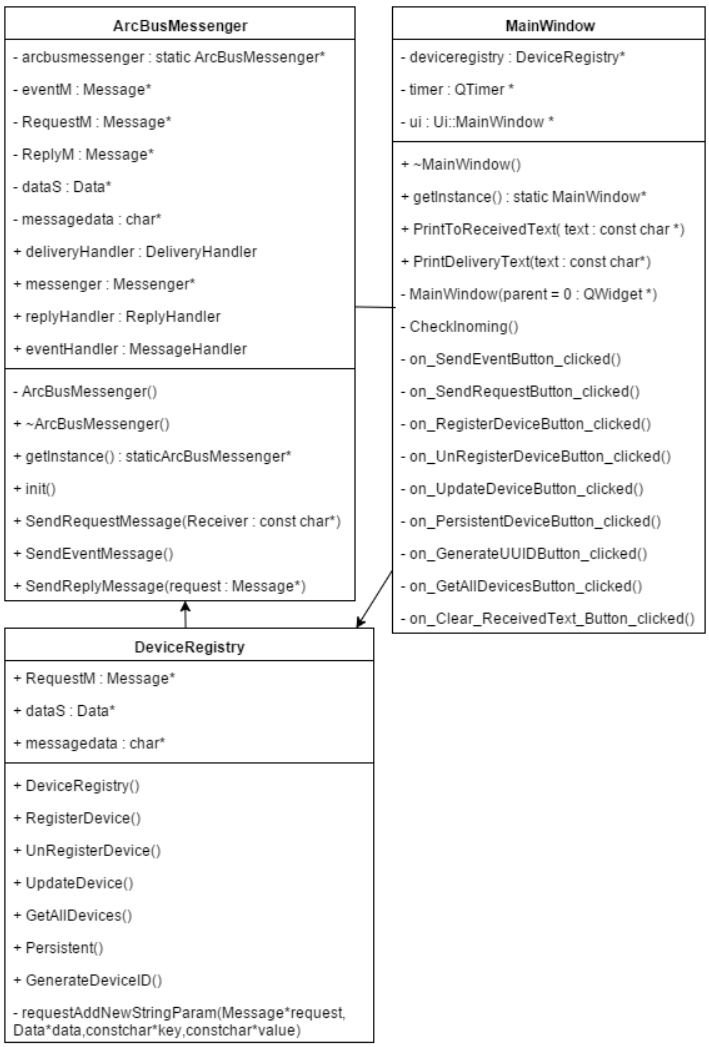
7.3 Ontwerp simulatie tool

De eerste stap van een ontwerp is het maken van een use-case. Een use-case diagram wordt gebruikt om per gebruiker te laten zien welke systeemfunctie de gebruiker gebruikt. De use-case diagrammen die gemaakt zijn voor de simulatie tool kunnen gevonden worden als figuur 15. Er is geen use-case gemaakt voor elk verschillend bericht die de simulatie tool kan verzenden, omdat de functie voor de gebruikers van de simulatie tool niet uitmaakt. In tabel 2 vind je een use-case beschrijving van de use-case "Berichten versturen". De rest van de use-case beschrijvingen kunnen gevonden worden in de bijlage requirements document.

Figuur 16 is een klassendiagram van de eerste versie van de simulatie tool. Deze maakt gebruik van de C SDK. Versie 1 maakt gebruik van een simpel design. De klasse "ArcBusMessenger" maakt de berichten en zorgt dat de C SDK ze verzendt en verwerkt de ontvangen berichten. De klasse DeviceRegistery maakt de berichten die verzonden kunnen worden naar de deviceregistery. De klasse "MainWindow" is een klasse die een window met button en invoer velden maakt. In deze klasse wordt het event dat ontstaat als iemand op een knop drukt opgevangen. In het klassen diagram is gebruik gemaakt van één design pattern. Dit design pattern is "singleton". Dit design pattern is gebruikt in de klasse "ArcBusMessenger". Het design pattern is gebruikt omdat je maar één instantie wilt van deze klasse. Dit komt omdat de ArcBusMessenger klasse een connectie maakt met de ARC-bus messagebroker. Je kan maar één keer met een bepaald UUID connecten met de messagebroker.

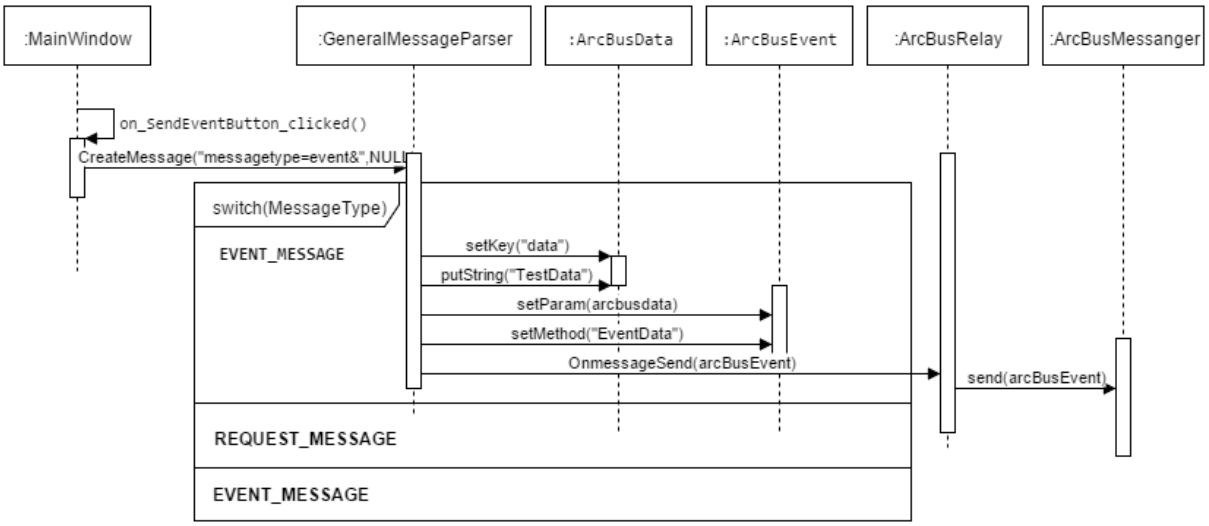


Figuur 15: Use-case diagram simulatie tool.



Figuur 16: klassendiagram simulatie tool versie1.

Figuur 17 is het sequence diagram de use-case "berichten verzenden". Als op de button "SendEventButton" wordt gedrukt, wordt de "on_SendEventButton_clicked" uitgevoerd. Dan wordt de CreateMessage functie van GeneralMessageParser aangeroepen. In de GeneralMessageParser wordt de data gemaakt en toegevoegd aan de ArcBusEvent message. Als die gemaakt is, wordt hij via de ArcBusRelay klassen doorgegeven aan de ArcBusMessenger die het bericht verzendt.



Figuur 17: Send event message simulatie tool versie 2.

7.4 Het opstellen van een testdocument

Op basis van alle diagrammen is er een testdocument geschreven. In dit document staan verschillende testen beschreven om de eisen uit het requirementsdocument te kunnen testen. Een testdocument is nodig om er voor te zorgen dat het systeem compleet werkt volgens de opgestelde eisen. Doel van het testen zelf is om te testen of het product werkt volgens de opgestelde requirements. In het testdocument worden drie verschillende testen beschreven. Unit-test: dit is het testen van een zo klein mogelijk stuk in de code. Bijvoorbeeld classes of functies. Deze tests worden uitgevoerd door scripts die gemaakt zijn met de framework "GoogleTest"[8]. Dit framework wordt gebruikt omdat VDG dit gebruikt en na verder ingelezen te hebben, bleek dit een goede keuze te zijn, omdat dit cross platform gebruikt kon worden. Het heeft een ingebouwde test discovery en een uitgebreide assertions methode. Dit is een nieuw requirement geworden. Dit framework is geschreven voor C++ en kan gebruikt worden onder Linux, Windows en Mac. In dit framework schrijf je voor elke functie die getest moet worden een test. Op het einde van de test wordt door een opgestelde bewering bepaald of de test gehaald is of niet. Voor het testen van de interfaces wordt de test framework "GoogleMock"[10] gebruikt omdat dit goed samen werkt met "GoogleTest" en het bedrijf wilde dit in de toekomst ook gaan gebruiken en zag daarom graag dat dit project een voorbeeld werd. Het gebruiken van "GoogleMock"[10] werd een nieuwe requirement. Mocks zijn objecten die voorgeprogrammeerd zijn met verwachtingen, deze verwachtingen vormen de specificaties van class functies. Als de mock class wordt aangeroepen moeten de verwachtingen overeen komen met de daadwerkelijke uitkomst. Deze mocks kunnen gebruikt worden in het testen. Bij deze tests word getest of de functie verwachtingen die opgesteld zijn in de mock ook in de werkelijkheid zo voor komen. De testscripts kunnen gevonden worden in het testdocument.

De tweede test die uitgevoerd gaat worden is een integratie test. In deze test wordt gebruik gemaakt van de simulatie tools. In de integratie test wordt de hele plug-in getest. Deze tests maken gebruik van een 'Specification-based testing'. Dit betekent dat de stappen die uitgevoerd moeten worden beschreven staan in een test case. Een paar belangrijke cases uit het testdocument zijn die van een bericht versturen en ontvangen.

Test Case	Bericht versturen
Beschrijving	Deze test is om te kijken of er een ARC-bus message gestuurd kan worden naar de ARC-bus broker
Benodigdheden	Sense, web simulatie tool, ARC-bus plug-in, QT simulatie tool
preconditie	Sense en ARC-bus plug-in hebben al goede settings en simulatie tool is gestart
Uitgevoerd door	
Datum van uitvoering	

Test2 1.1	Verwachte uitkomst	Officiële uitkomst
Druk op de knop 'SendEvent' of 'SendRequest	Bij de QT simulatie tool word het ontvangen bericht ontvangen en geprint op het scherm	

Test Case	Bericht ontvangen
Beschrijving	Deze test is om te kijken of een bericht ontvangen kan worden en de juiste data uit het bericht gelezen kan worden
Benodigdheden	Sense, ARC-bus plug-in, QT simulatie tool
preconditie	Sense en ARC-bus plug-in hebben al goede settings en simulatie tool is gestart met de goede settings
Uitgevoerd door	
Datum van uitvoering	

Test2 2.1	Verwachte uitkomst	Officiële uitkomst
Druk op de knop 'SendEvent' of 'SendRequest' van de simulatie tool	In de console van de SensePluginManager wordt een melding gegeven wanneer het bericht is ontvangen	

Naast dat er een integration test is uitgevoerd met cases, zijn er ook integration tests geschreven met de google test framework. In deze tests worden een deel van de test cases geautomatiseerd.

Als laatste wordt er een 'System test' uitgevoerd. Deze test maakt ook gebruik van 'Specification-based testing'. De 'System test' wordt uitgevoerd door een tester van VDG en niet door de developer. Deze tests worden niet uitgevoerd met de simulatie tool maar met VDG Sense zelf. Doordat deze tests worden uitgevoerd door de tester en niet de developer zijn deze tests black box tests. Een black box test is een test waar de tester niet weet wat er binnen het systeem gebeurt. De tester heeft alleen invloed op de input en de output van het systeem. Met deze test wordt de functionele en niet functionele eisen getest. Als de 'Specification-based' tests gehaald zijn dan gaat de tester proberen om het systeem te slopen door waardes in te vullen die niet verwerkt kunnen worden.

Na het opstellen van het ontwerp wordt dit ontwerp geïmplementeerd. Hoofdstuk 8 is het volgende hoofdstuk en in dit hoofdstuk wordt besproken hoe de simulatie tool is geïmplementeerd.

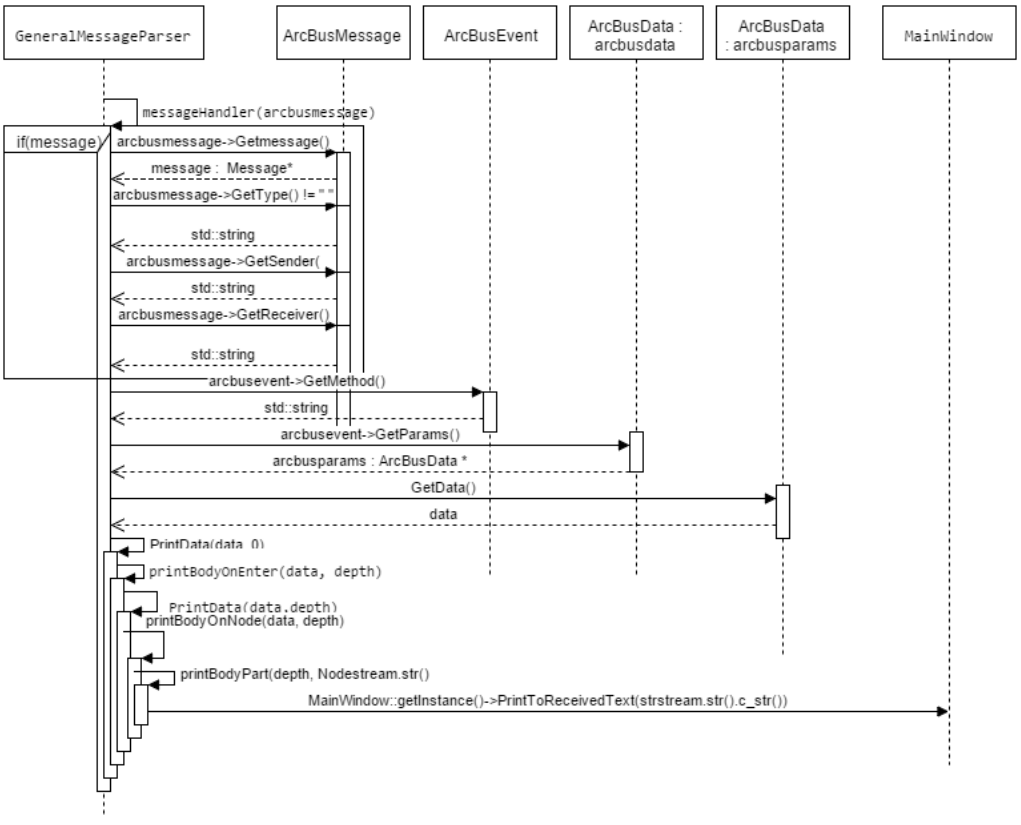
8 Ontwikkelen van simulatie tool (sprint 4)

Sprint 4 duurt twee weken lang. Aan het eind van deze sprint kan de simulatie tool opgeleverd worden. Tijdens het ontwikkelen zijn er twee versies gemaakt van de simulatie tool. De eerste versie maakt gebruik van de C SDK en de tweede maakt gebruik van de C++ SDK. Versie 1 is gemaakt om de verschillende functies van de C SDK te kunnen testen en om later versie 2 met de C++ SDK te kunnen testen. Het klassendiagram van versie 1 kan je vinden in figuur 12.

Voordat versie 2 van de simulatie tool gemaakt kon worden, moest eerst de C++ SDK gemaakt worden. De C++ SDK is gemaakt volgens de klassendiagram en sequence diagrammen uit sprint 3. Tijdens het maken van de C++ SDK zijn er een paar problemen gevonden in de C SDK, waardoor de code niet werkt. Deze zijn gecoördineerd met de developers van de C SDK. Eén van deze fouten was dat sommige functie namen in de header file niet overeen kwamen met de functie in de source file. Een functie die mist was er één om naar de volgende node te kunnen stappen. Deze functie is zelf toegevoegd aan de C SDK.

De functie voor het uit elkaar halen van een message kon hergebruikt worden. Er zijn wel een paar aanpassingen gemaakt om hem om te zetten naar beter gebruik van C++. De ArcBusParser klasse is overgenomen van de ARC-bus plug-in klassendiagram om te testen of het omzetten van de berichten op deze manier gaat werken. Dit is terug te vinden in figure 14: Send event sequence diagram simulatie tool

Het ontvangen van een event bericht en het verwerken is ook anders dan de sequence diagram van de plug-in. Deze kan je vinden als figuur 18. Aan het eind van deze sprint is een werkende simulatie tool opgeleverd met versie 2.



Figuur 18: sequece diagram use-case berichten ontvangen

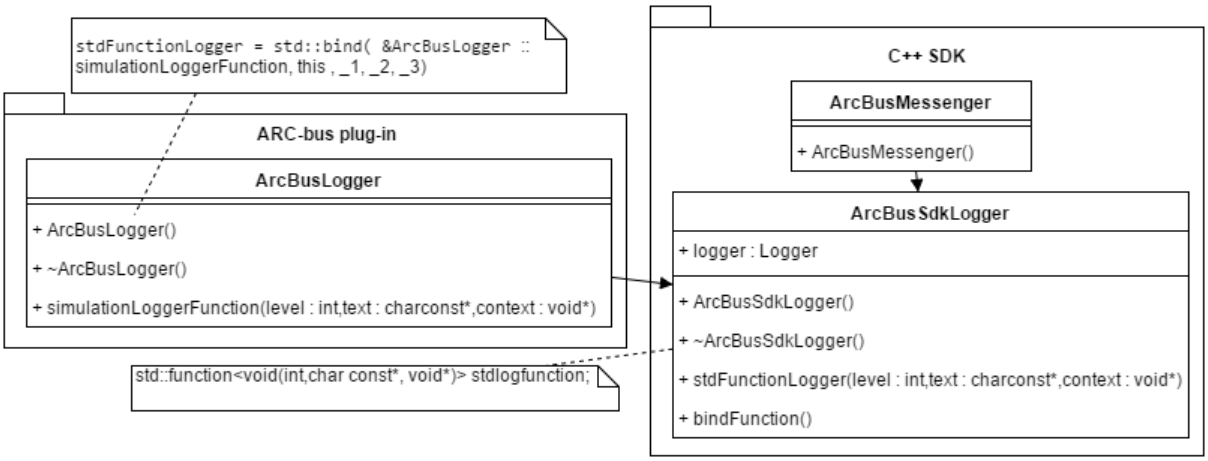
Aan het eind van deze sprint heeft de bedrijfsbegeleider de code gereviewed van de simulatie tool versie 2. Hierbij kwamen nog een paar puntjes naar boven. Deze worden in sprint 5 genoemd en verbeterd. Sprint 5 wordt besproken in het volgende hoofdstuk.

9 Ontwikkelen van C++ SDK (sprint 5)

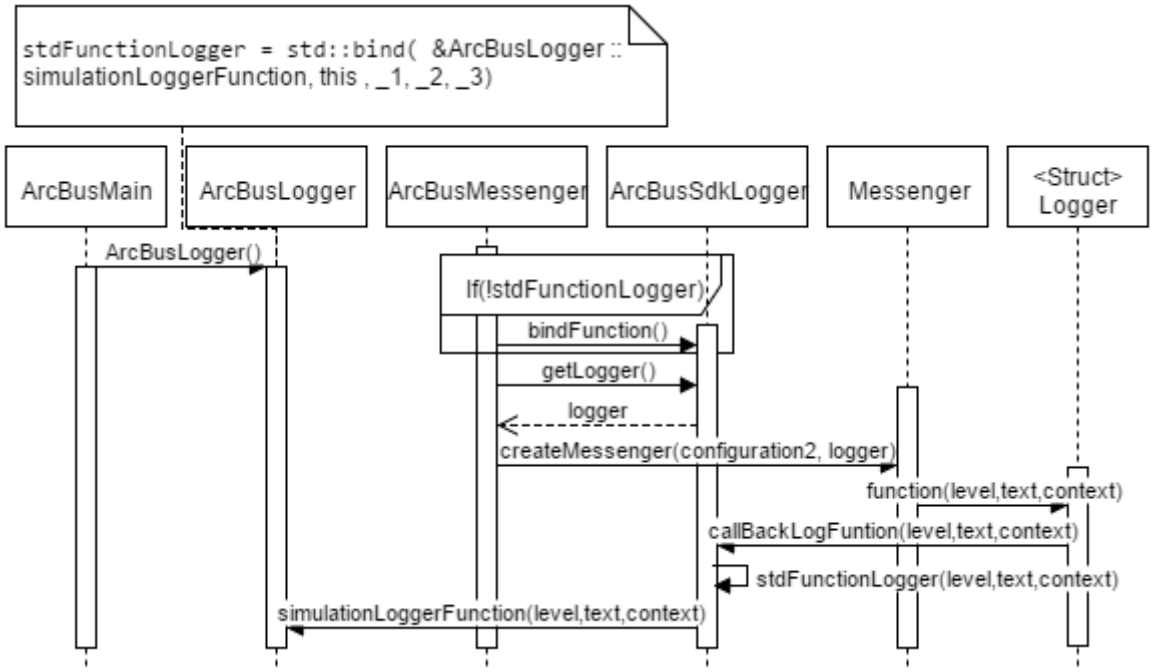
Aan het eind van sprint 5 zou de C++ SDK werkend moeten zijn in de plug-in en is er begonnen aan de plug-in.

Uit de review van de simulatie tool versie 2 waren er enkele punten naar boven gekomen. Hierdoor moest tijdens deze sprint het design van de C++ SDK en de simulatie tool refactored worden. Dit heeft ongeveer een week geduurd. Het kan zijn dat een parser alleen maar een event binnen wil krijgen. In het huidige klassen diagram heeft de parser nog steeds functies voor de andere berichten. Elke keer als er een parser gemaakt moet worden moet de OnMessageReceived functie bij gewerkt worden omdat er in OnMessageReceived wordt bepaald met bijvoorbeeld een switch welke parser aangeroepen moet worden. Het eerste probleem is gefixt door van de IArcBusMessageParser vier interfaces te maken. Deze interfaces konden in de C++ SDK komen. Als een parser nu event berichten wil ontvangen moet hij van de interface IEventListener overerven. Het tweede probleem is verholpen door de IArcBusRelay interface en de implementatie klassen ArcBusRelay van de interface te verwijderen. In plaats van deze klassen zijn vier vectoren toegevoegd aan de ArcBusMessenger klassen. Ook functies om de vier listeners toe te kunnen voegen en te verwijderen. Dit kan gezien worden als observer design pattern. Op het moment dat er een bericht ontvangen wordt van elk object in de vectoren, wordt de parse message aangeroepen. Als je nu een extra parser maak hoeft die zich alleen maar te registeren bij de ArcBusMessenger. Door de functies "registerEventListener", "registerRequestListener", "registerReplyListener" of "registerErrorListener" aangeroepen. Voor de functie om de parser te kunnen registeren, wordt er gebruik gemaakt van een factory. Een factory is een design pattern die gebruikt kan worden om instanties van klassen aan te maken. Voor klassendiagram versie 2 zie bijlage 1.2 en voor versie 3 van het klassendiagram zie bijlage 1.6.

In deze versie van de C++ SDK is ook een logger klasse toegevoegd. Figuur 20 is een sequence van de eerste oplossing om te loggen in de plug-in en SDK. figuur 19 is de bijbehorende klassendiagram. De C SDK heeft een struct genaamd logger met daar in een function pointer. Deze function pointer word aangeroepen in de C SDK. Tijdens het initialiseren van de messenger moet er een struct logger mee gegeven worden. De functie waarmee je de messenger initialiseert is "createMessenger" In de logger klasse van de C++ SDK wordt de struct aan gemaakt met een pointer naar een niet member functie callBackLogFuntion. In de functie callBackLogFuntion wordt een std::function aangeroepen. std::function is te vergelijken met een container waar je functies in kan opslaan en die functies kan je dan weer aanroepen via de std::function. De std::function word gebonden als je niet zelf een functie bind. Met de bind functie kan je een functie in de container std::function stoppen. Deze standaard logger functie print alleen naar de console, als je dit anders wilt dan kan je een eigen logger maken en deze binden aan de std::function. In de sequence diagram wordt een externe logger gemaakt en die bind zijn functie in de std::function stdFunctionLogger.

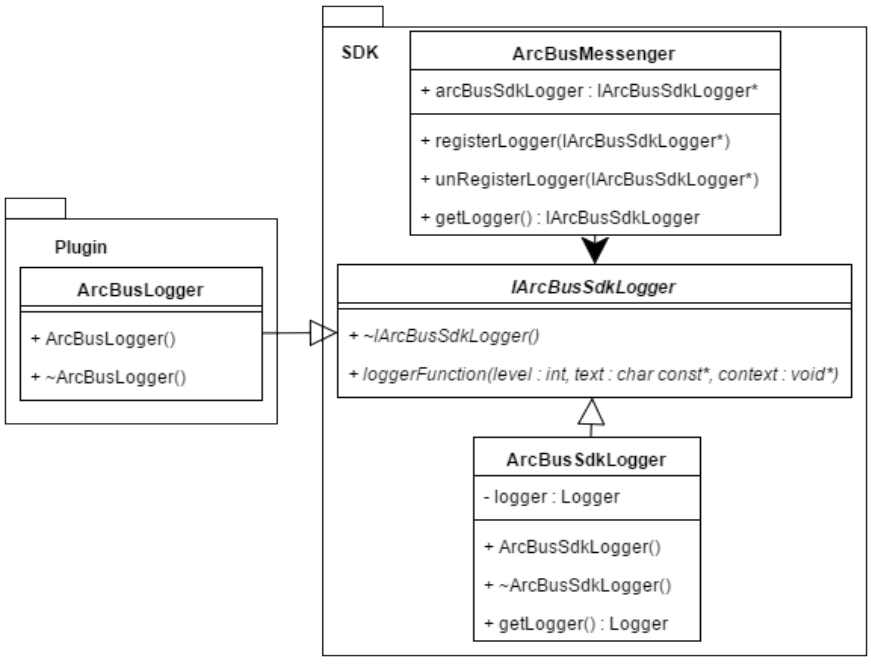


Figuur 19: Klassendiagram oplossing 1 logging

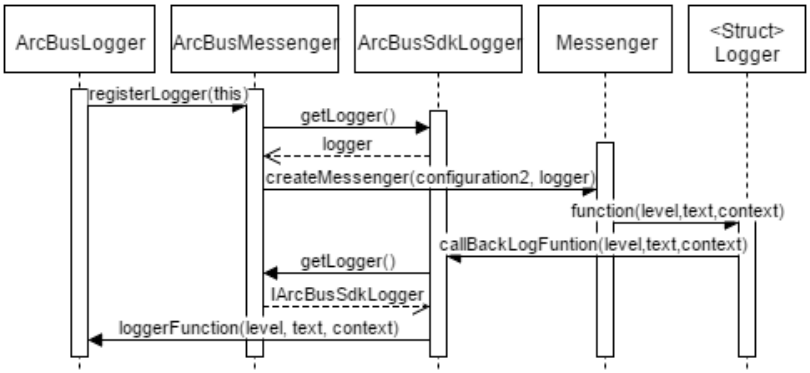


Figuur 20: oplossing 1 logging

Een andere manier om de logging te doen is door middel van een interface klassen. Een interface beschrijft het bedrag en de mogelijkheden van een klassen zonder deze te implementeren. De klassendiagram van het log stukje is te vinden als figuur 21. De bijbehorende sequence is te vinden als figuur 22. In deze versie wordt de struct Logger nog steeds gebruikt om mee te loggen in de C SDK. De functie pointer wijst nog steeds naar de functie "callBackLogFunction". Een functie pointer is een variabele die het adres in het geheugen van de functie opslaat. Zo kan via de functie pointer de functie aangeroepen worden. Alleen wordt in in de functie "callBackLogFunction" nu niet een sdk::function aangeroepen maar wordt aan de "ArcBusMessenger" de Logger opgevraagd. De "ArcBusMessenger" klassen returned de pointer naar een "IarcBusSdkLogger". Hiervan wordt de functie "loggerFunction" aangeroepen. Je kan zelf een implementatie maken van de "IArcBusSdkLogger" interface en deze registeren. Als je niks registreert dan wordt de standaard logger klasse van de C++ SDK geregistreerd en wordt de functie "loggerFunctie" van hem aangeroepen.



Figuur 21: klassendiagram logger versie 2



Figuur 22: Sequence diagram logger versie 2

Er is gekozen voor de eerst versie 1 van het loggen. Allebei de oplossingen zijn niet ingewikkeld om te implementeren. Alleen zorgt versie 1 voor minder code dan in de messenger. De messenger hoort niet te bepalen welke logger gebruikt wordt in de plug-in. De messenger hoort berichten te kunnen verzenden en ontvangen. Het loggen van de eerste versie is makkelijker te gebruiken. Je roept de std::function aan als je wilt loggen en dan maakt het niet uit welke class method gebonden is aan die functie. In versie 2 moet je eerst de logger opvragen aan de messenger en dan van die logger zijn logger functie aanroepen.

Toen de nieuwe versie van de C++ SDK af was kon de C++ SDK geïntegreerd worden in de simulatie tool. Voordat dit gedaan kon worden zijn eerst de klassen diagrammen en de sequence diagrammen aangepast naar de nieuwste versie van de C++ SDK.

De klassen "IArcBusMessageParser" en ArcBusRelay moesten verwijderd worden. Bij de afgeleiden klassen van "IArcBusMessageParser" was het alleen nodig om de base klassen te veranderen. Er kwam nog een factory klassen bij. In deze klassen worden de parsers aangemaakt en geregistreerd bij de ArcBusMessenger. Er werd ook een klasse "SimulationLogger" aangemaakt.

Bij deze versie is er ook een optie bij gekomen om de settings van de "ArcBusMessenger" te veranderen via de applicatie. Dit kan gedaan worden in een aparte dialoog om het overzichtelijker te maken. De settings zelf worden niet bijgehouden in de "ArcBusMessenger" klassen maar in de "ArcBusSettings" klassen.

Nu kon er gekeken worden of het verzenden en het ontvangen met de nieuwe C++ SDK werkt. Uit de tests bleek dit te werken. De tests die uitgevoerd zijn kan je vinden in bijlage 6 testdocument. Daarna kon de plug-in geïmplementeerd worden.

Stap één was om de "ArcBusMain" klassen aan te maken om te kijken of de plug-in geladen kon worden door de "SensePluginManager". Door de structuur die VDG al had opgezet was dit makkelijk te doen.

De volgende klasse die gemaakt moest worden was de "ArcBusSettings". Deze klasse wordt gebruikt om de settings voor de messenger bij te houden en te veranderen als dit gebeurt in de client. De client vraagt via de plug-in manager aan de plug-in welke settings hij heeft, zodat de client run time de settings window kan aanpassen aan de gebruikte settings.

Aan het eind van deze sprint is een werkende simulatie tool versie 3 en een C++ SDK versie 2 opgeleverd. In het volgende hoofdstuk wordt beschreven hoe de plug-in ontwikkeld is.

10 Ontwikkelen van plug-in (sprint 6)

Aan het einde van sprint 6 moet de plug-in af zijn. Voordat de plug-in af is, moeten de parsers gemaakt worden. Ook moeten de huidige klassen aangepast worden, zodat ze de parsers gebruiken.

De DeviceRegistry parser kon voor een deel overgenomen worden. Alleen waar de waardes in de berichten vandaan komen is veranderd. VDG had al een klassen waarmee het IP-adres opgevraagd werd aan Windows. Voor het opvragen van het MAC-adres was er nog geen code. Deze methode is toegevoegd aan de huidige klassen in de VDG software, zodat dit later ook nog hergebruikt kan worden door VDG.

De "SenseMessageParser" is één van de belangrijkste klasse van de plug-in. Deze klasse convert het sense xml bericht naar een ARC-bus bericht. In de sequence diagram is te vinden hoe deze patch loopt. Een bericht van Sense komt binnen in de "ArcBusMain". Hier wordt eerst gekeken naar wat voor type bericht het is. Als het een "pdiPluginMessage" is dan wordt de "messageHandle" functie aangeroepen. Hier wordt gekeken of dit een bericht is dat de settings verandert, het een bericht is voor de deviceregistry of een signal of action is. Bij een deviceregistry wordt de "createMessage" functie van de deviceregistry aan geroepen en bij een signal of een action wordt de createmessage van de SenseMessageParser aangeroepen. In de "createMessage" functie wordt de xml eerst omgezet naar een Sense signal of action. Daarna wordt er met een iterator door de parameters gelopen en in de ArcBusMessage gestopt. De reden dat het op deze manier gedaan wordt, is omdat het signal en action al van xml geconverteerd kan worden. Als dat gedaan is kan er eenvoudig door de parameters gelopen worden met een for loop.

Het omzetten van een "ArcBusMessage" naar een Sense message is relatief eenvoudig. De data in een "ArcBusMessage" kan er ongeveer uitzien als in tabel 3. Het bericht bestaat uit een map structuur en bestaat als eerste uit een key en als tweede uit de waarde. Er is gekozen om het bericht er zo uit te laten zien, omdat de parameters van de signal en action ook bestaan uit een key en waarde. Er kan door deze map heen gelopen worden alsof het nodes zijn. Door middel van een switch wordt bepaald welke get functie aangroepen moet worden. Tijdens het lopen langs de nodes worden de keys en de bijbehorende waardes in een aan de parameters van een signal of action toegevoegd. In deze parser wordt alleen de map, string, integer en bool gesupport omdat dit de enige types zijn die in het bericht gestopt worden en sense signal en action parameters bestaan alleen uit strings, integers en bools.

map "ipv4" (string) "172.21.243.86" (string) "_id" (string) {283add21-5412-4b5a-9e99-dce566661111} (uuid) "online" (string) "true" (string) "integer" (string) "55" (int) }

Tabel 3: simpele weergave van een ArcBusData

Toen de parsers klaar waren kon de "ArcBusFactory" aangepast worden zodat de klasse "ArcBusMain" via de factory de twee parsers kan aanmaken. Het stukje code om de "createMessage" functies aan te roepen.

Aan het eind van deze sprint zijn de volgende klassen gemaakt of afgemaakt:

- "SenseMessageParser"
- "DeviceRegistryMessageParser"
- "ArcBusFactory"
- "ArcBusMain"

Er is ook nog een code clean-up gedaan op de andere gemaakte klassen. Met de boven genoemde klassen gemaakt of afgemaakt is er een plug-in gereed die getest kan worden. Dit testen gaat gebeuren in de volgende sprint. Meer informatie over het testen kan gevonden worden in het volgende hoofdstuk.

11 Het Testen (sprint 7)

In deze sprint zijn de testen die beschreven staan in het test document uitgevoerd. Het test document is niet geschreven in deze sprint maar in sprint 3 in de ontwerp fase. Daarom is een samenvatting van het testdocument beschreven in paragraaf 3.4. Aan het eind van deze sprint zouden alle test uitgevoerd moeten zijn en zou er een volledige werkend product klaar zijn of er komen een paar kleine bugs uit die in sprint 8 gefixt worden en gelijk weer getest worden.

De eerst tests die werden uitgevoerd zijn de unittests. Uit deze test bleek een paar dingen niet volledig te werken. Een aar van deze unit tests van de SDK zien er als volgt uit.

```
TEST(ArcBusEvent, GetParams) {
 ArcBusEvent arcBusEvent = ArcBusEvent();
 ArcBusData arcBusData = ArcBusData();
 arcBusData.setKey("event");

 arcBusData.putString("message");

arcBusEvent.setParam(&arcBusData);

 ArcBusData* arcBusDataResult = arcBusEvent.getParams();
 ArcBusData* arcBusDataString = arcBusDataResult->getMapValue("event");

 EXPECT_EQ(arcBusData.getKey(), "event");
 EXPECT_EQ(arcBusDataString->getString(), "message"); }
```

Tabel 4: Test ArcBusEvent GetParams

In dit testje wordt zoals in het test document staat, gebruik gemaakt van googletest. Elke unittest moet beginnen met TEST. De eerste parameter wordt gezien als test naam. Tweede parameter is wat de rest doet. In deze test wordt een functie van de ArcBusEvent getest die de funtie GetParams test. Met "EXPECT_EQ" wordt geweerd dat de waarde links van de komma gelijk is met de waarde aan de rechter kant. De functies die na het testen niet bleken te werken waren die om uit een container te stappen. Dit ging bijvoorbeeld fout wanneer je maar één container had. De functie returnde true als het uit de container stappen en naar de volgende node stappen goed ging. Door dat je maar één container gebruikte kon er niet naar een volgende node gestapt worden dus krijg je altijd false terug. Toen dit getest werd met drie containers door in de eerste container te stappen en er uit te stappen en dan vervolgens in de volgende te stappen. Door dit te testen bleek de tweede container over geslagen te worden.

Ook zijn de "GoogleMock" tests uitgevoerd. Er zijn twee mock classes gemaakt. Één voor de ArcBusMessenger interface en één voor de IEventListener, IRequestListener, IReplyListener, IErrorListener. Voor de IArcBusMessenger mock class is niet voor elke functie een test geschreven, omdat die functie rechtstreeks worden aan geroepen en niet via een andere functie. Er is een test geschreven voor de belangrijkste functies en dit zijn de functies die worden gebruikt om de messenger de initialiseren. De mock class voor de listeners ziet er als volgt uit:

```
class MockParser : public IEventListener, public IRequestListener,
                   public IReplyListener, public IErrorListener
{
 public:
 MOCK_METHOD1(eventMessageParser,void(ArcBusMessage*arcBusMessage))
 MOCK_METHOD1(requestMessageParser,void(ArcBusMessage*arcBusMessage));
 MOCK_METHOD1 (replyMessageParser,void(ArcBusMessage*arcBusMessage));
 MOCK_METHOD1(errorMessageParser,void(ArcBusMessage*arcBusMessage))
};
```

Tabel 5: MockParser

De eerste regel met MOCK_METHOD1 betekent dat het een functie is in een mock class en dat het één functie parameter eeft. Functie naam is "eventMessageParser". De functie geeft niks terug en de functie parameter is "eventMessageParser".

```
TEST(MockParser, Event)
{
 MockParser parser;
 EXPECT_CALL(parser, eventMessageParser(A<ArcBusMessage*>())).Times(1);

 IArcBusMessenger* arcBusMsg = ArcBusMessenger::getArcBusMessenger();
 bool result = false;
 result = arcBusMsg->init("amqp"
                          ,"127.0.0.1"
                         ,"5672"
                         ,"tkh.vdg.device.mock.event.1"
                         ,"283add21-5412-4b5a-9e99-dce566666666"
                         ,"certificate.der"
                         ,"privatekey.der");
 EXPECT_TRUE(result);
 EXPECT_TRUE(arcBusMsg->start());

 while(!arcBusMsg->isConnected());
 EXPECT_TRUE(arcBusMsg->isConnected());

 arcBusMsg->registerEventListener(&parser,">");
 arcBusMsg->send(&arcBusEvent);

 Sleep(10);
 DWORD waitResult = WaitForSingleObject(arcBusMsg->getSynchronizationHandle(), INFINITE);
 if(waitResult == WAIT_OBJECT_0)
   arcBusMsg->dispatchMessages();
 arcBusMsg->unRegisterEventListener(&parser);
 delete arcBusMsg;
}
```

Tabel 6: MockParser test eventMessageParser

De test om de mock functie "eventMessageParser" te kunnen testen ziet er als volgt uit:
De vierde regel is voor deze test het belangrijkst. Met deze regel wordt beweerd dat de functie "eventMessageParser" van parser één keer wordt aangeroepen. De rest van deze code in de test is er om te zorgen dat de functie aangeroepen wordt. Aan het eind van deze test bleek de "errorMessageParser" niet werd aangeroepen. Na wat onderzoek bleek deze fout te zitten en de C SDK van TKH innovations. Deze fout is aan hun gemeld. Ze zouden dit zo snel mogelijk aanpassen.

Een uitgevoerde test van de simulatie tool is bijvoorbeeld die van berichten ontvangen. In deze test zie je dat de simulatie tool alle event berichten wil ontvangen die naar de message broker worden gestuurd. Deze test was ook succesvol.

Test Case	Berichten kunnen ontvangen van devices zonder dat ze een specifieke naam moeten hebben.
Beschrijving	Verstuur een request bericht over de ARC-bus naar een niet bestaande naam. use-case Berichten ontvangen.
Benodigdheden	Twee simulatie tools, message broker, deviceregistry
preconditie	Instellingen zijn ingesteld
Uitgevoerd door	Aiken Brus
Datum van uitvoering	12-07-2015

Tabel 7: Test Case

Tijdens het testen zijn ook een paar testen foutief uitgekomen. Dat is bijvoorbeeld wanneer je de simulatie tool afsluit dat hij zich afmeld bij de deviceregistry of als de value Persistent op true staat dan komt de simulatie tool op offline te staan. Dit is getest door twee simulatie tools te gebruiken. In de database stonden twee simulatie tools ingeschreven. Nadat er één van de twee afgesloten was stonden ze allebei nog steeds in de database. De rest van de testen gingen wel goed.

Test2.1 11.2	Verwachte uitkomst	Officiële uitkomst
1. Druk op de knop "Settings" 2. Verander de Device Name naar een willekeurige naam bijvoorbeeld:"sdhjwdf" 3. Druk op de knop "Sendevent"	In de ouput komt de volgende message te staan: <messagetype> event </messagetype> <timesend> (datum en tijd)</timesend> <sender> sdhjwdf </sender> <method> EventData </method> <Body> { "data" (string) "TestData" (string) } </Body>	<messagetype> event </messagetype> <timesend> 12/07/15 10:32:45 </timesend> <sender> sdhjwdf </sender> <method> EventData </method> <Body> { "data" (string) "TestData" (string) } </Body>

Tabel 8: Test scenario

Nadat de simulatie tool getest was kon de plug-in getest worden. Voor de plug-in zijn er test cases geschreven. Er zijn tests cases geschreven voor zowel de integratie test als voor de System test.

Voor de integratie test wordt de simulatie tool gebruikt om de plug-in te kunnen testen. De test case voor het ontvangen van berichten is te vinden in tabel 9 en tabel 10.

Test Case	Bericht ontvangen
Beschrijving	Deze test is om te kijken of een bericht ontvangen kan worden en de juiste data uit het bericht gelezen kan worden
Benodigdheden	VDG sense, ARC-bus plug-in, simulatie tool, web plug-in tool
preconditie	Sense en ARC-bus plug-zijn hebben al goede settings en simulatie tool is gestart met de goede settings
Uitgevoerd door	Aiken Brus
Datum van uitvoering	7-12-2015

Tabel 9: Test case Bericht ontvangen

Test2 2.1	Verwachte uitkomst	Officiële uitkomst
Druk op de knop 'SendEvent' van de simulatie tool van de web plug-in tool. Ala je op 'SendRequest' drukt met de naam van Sense in de invoerveld gebeurt er niks	In de console en de log file van de sensepluginmanager wordt een melding gegeven wanneer het bericht is ontvangen. Als de console niet draait dan kan ik het log bestand van de plug-in terug gevonden worden dat er een bericht ontvangen is.	in de consle en log file staat SenseMessageParser event ontvangen from tkh.vdg.device.simulation.1 with method EventData en als op 'SendRequest' drukt gebeurt er niks.

Tabel 10: Tests voor test case bericht ontvangen

Bij de system test wordt er gebruik gemaakt van twee servers. De bedoeling van deze tests is om te kijken of signals en actions verstuurd kunnen worden via de ARC-bus. De signals en actions moeten ook worden verwerkt. Een signal is een soort event die getriggerd is. Het ligt aan de instellingen of er een actie wordt uitgevoerd voor de signals. Een action is daadwerkelijk een actie die uitgevoerd moet worden. Een actie kan bijvoorbeeld een layout switchen. De test case voor het layout switchen is te vinden tabel 9 en 10. Sommige van deze test zijn automatisch gemaakt door middel van "GoogleTest". In plaats van op een knop drukken waardoor er een bericht gemaakt wordt, is het bericht al standaard ingesteld en wordt deze verstuurd als een event op de ARC-bus.

Test Case	Switch layout 1 van systeem twee
Beschrijving	use-case bericht verzenden
Benodigdheden	twee systemen met VDG Sense geïnstalleerd, één message broker.
preconditie	Allebei de VDG Sense systemen hebben de ARC-Bus settings van de message broker, twee layouts één met de naam Firstlayout, de tweede systeem is live modus met de andere layout zichtbaar
Uitgevoerd door	Aiken Brus
Datum van uitvoering	7-12-2015

Tabel 11: Switch layout 1

Test2 2.1	Verwachte uitkomst	Officiële uitkomst
1. Open de link http://127.0.0.1/ArcBus/test.html in je browser 2. Druk op de knop "VMswitchlayout1"	De secondlayout moet zichtbaar zijn in de live modus.	De secondlayout is zichtbaar op de live modus.

Tabel 12: Switch layout 1 tests

In het volgende hoofdstuk wordt besproken hoe de bugs die tijdens deze sprint ontdekt zijn verholpen zijn.

12 Bugs verhelpen en testen (sprint 8)

In deze sprint moeten de bugs, die gevonden zijn tijdens het testen in sprint 7, opgelost worden en dan weer opnieuw getest worden.

De eerste bug die verholpen was is die van het uit een container stappen tijdens het uitlezen van een AMQP bericht. Dit kan makkelijk werkend worden gemaakt door de next functie uit de vergelijking te halen. De tests die met deze functies te maken hadden die in sprint 7 waren uitgevoerd zijn weer opnieuw uitgevoerd. De unittest kwamen nu wel positief uit en het bericht werd ook volledig uitgeprint in de simulatie tool.

De tweede bug was dat de errormessages niet in de "errorMessageParser" functie kwam. Na wat uitzoeken en debuggen van de C SDK bleek het wanneer er een bericht binnen komt er nooit gekeken wordt of het een error message is, dus nooit de errormessagehandler callback aangeroepen.

De laatste bug die ontdekt was is dat als je Sense of de simulatie tool afsloot het device niet afgemeld werd bij de database. Dit kwam omdat de unregister functie niet wordt aangeroepen als Sense of de simulatie tool wordt afgesloten. De functie wordt aangeroepen in de destructor van de factory. De factory wordt gebruikt om alle pointers naar de parsers te verwijderen.

Nadat de bugs behalve de tweede verholpen waren kon alles weer getest worden. De test waren allemaal positief verlopen. Het volgende hoofdstuk is de laatste sprint van dit project. Tijdens deze sprint wordt alle documentatie verder uitgewerkt.

13 De documentatie en acceptatie (Sprint 9)

Sprint 9 is de laatste sprint. In deze sprint wordt de documentatie afgemaakt. Tijdens de andere sprints was er al aan de verschillende documenten gewerkt. In de onderzoek sprint is het onderzoeks-document geschreven en het requirements document. In de ontwerp sprint werd het testdocument gemaakt. Naar deze documenten is in sprint negen opnieuw gekeken en doorgelezen en waar nodig extra data aan toe gevoegd.

In sprint 9 is ook een user manual gemaakt. In deze user manual staat hoe de C++ SDK geïntegreerd kan worden. Door het hele project is er één dag per week gewerkt aan het eindverslag. Dit was niet goed genoeg om hem helemaal af te krijgen. Daardoor is ook tijdens sprint negen gewerkt aan het eindverslag.

Conclusie / aanbevelingen Tijdens dit project is er een onderzoek gedaan naar de ARC-bus. Uit dit onderzoek kwam naar voren dat de ARC-bus gebruikt kon gaan worden voor Sense. De ARC-bus is alleen nog niet helemaal af. De message brokers kunnen bijvoorbeeld nog niet aan elkaar gekoppeld kunnen worden. Op het eind van dit project zijn de simulatie tool, C++ SDK en de plug-in ontwikkeld. Deze drie producten werken volgens de requirements. In Sense is er alleen een kleine aanpassing gemaakt die misschien wat onderdelen van Sense gesloopt zouden kunnen hebben.

De aanbeveling voor dit project is om de aanpassing in Sense nog een keertje goed door te kijken en misschien een betere te bedenken. Ook nog even wachten totdat de ARC-bus volledig ontwikkeld is en er nog wat meer bedrijven binnen TKH Group zijn die de ARC-bus geïmplementeerd te worden.

Het volgende hoofdstuk is nummer 14. In dit hoofdstuk wordt de kwaliteit van de gemaakte producten en het proces geëvalueerd, of het doel is gehaald is en of het probleem is opgelost.

14 Evaluatie

In dit hoofdstuk worden de producten en het proces op kwaliteit beoordeeld. Voor elk product wordt eerst beschreven hoe de kwaliteit gewaarborgd zou moeten zijn. Daarna wordt gekeken of het product ook echt de kwaliteit heeft.

14.1 Producten

Plan van aanpak Het plan van aanpak zou gemaakt moeten worden volgens de checklist van Roel Grit[1]. Dit is ook zo gedaan. De planning is niet helemaal volledig gevolgd doordat er na een sprint review puntjes naar boven kwamen die eerst verbetert zouden moeten worden voordat er verder gewerkt kon worden. Tijdens het schrijven van het plan van aanpak zelf zijn er geen problemen opgekomen. Plan van aanpak kan je vinden als bijlage 3.

Onderzoeksrapport De kwaliteit van het onderzoeksrapport kon gewaarborgd worden door eerst de onderzoeks vragen op te stellen. Dit onderzoeksrapport gaat gelezen worden door de begeleider zodat onderdelen die nog niet duidelijk zijn, onderzocht kunnen worden. Het enige probleem dat opgekomen was tijdens het onderzoek is dat de documentatie van de ARC-bus niet volledig is. Dit probleem kon verholpen worden door naar de source code te kijken of te mailen met de lead developer. Het onderzoeksrapport kan je vinden als bijlage 4.

Requirements document De requirements document moet gereviewd worden door de bedrijfsbegeleider. Na de review zag de bedrijfsbegeleider dat er nog een paar niet goed SMART waren. Requirements document kan je vinden als bijlage 5.

Testdocument Dit testdocument is gemaakt met de toetscriteria van de beroepstaak D17. Alle puntjes in de toetscriteria zijn volledig uitgevoerd in het testdocument. Tijdens het testen waren er een paar tests die fout gingen. Sommige van deze konden in de code verholpen worden. De test waar een error message bericht ontvangen werd ging fout. Dit probleem zat hem in de C SDK waardoor het niet verholpen kon worden. Het Testdocument kan je vinden als bijlage 6.

Use-case diagram De use-case diagram moet gemaakt worden met de specificaties van UML. De use-case diagram wordt besproken met de bedrijfsbegeleider. Dit is inderdaad gedaan en het use-case diagram is goedgekeurd. Use-case diagrammen kan je vinden in hoofdstuk 7.1 en 7.4 figuur 11 en figuur 15.

UML klassendiagrammen De klassendiagrammen moeten gemaakt worden met de specificaties van UML. De klassendiagram wordt gecontroleerd door de bedrijfsbegeleider. Dit is inderdaad gedaan, maar versie twee was pas gecontroleerd aan het eind van de sprint waardoor de C++ SDK al geschreven was via dit ontwerp en de simulatie tool ook. Toen er aanpassingen kwamen in het ontwerp van de C++ SDK moest dit weer opnieuw geschreven worden. Ook moest er een nieuwe versie van de sequence diagrammen komen. Klassendiagrammen kan je vinden in bijlage 1.

UML sequence diagrammen De sequence diagrammen moet gemaakt worden met de specificaties van UML. De sequence diagrammen komen voort uit scenario's van de use-cases. De functie namen volgen uit de klassendiagram. De sequence diagrammen volgen inderdaad de scenario's van de use-case en de functie namen zijn gelijk aan die in de klassendiagram. De sequence diagrammen zijn de scenario's die worden uitgevoerd in de code. sequence diagrammen kan je vinden in bijlage 1.

C++ SDK De code moet worden geschreven in de google code style. Er moeten tests geschreven worden voor de C++ SDK. Er zijn unittests geschreven met de "GoogleTest" framework. De C++ SDK moest na een review verbeterd worden. Het is nu een werkende SDK. Voor het procesbeschrijving kan je naar hoofdstuk 9 gaan.

Simulatie tool De simulatie tool wordt ook geschreven in de google code style. Er moeten tests geschreven worden. Aan het eind van sprint vier wordt de simulatie tool gereviewd. Na de eerst review van de C++ SDK bleek deze niet goed te zijn waardoor de eerste versie van de simulatie tool ook niet goed was. Toen de laatste versie getest werd, kwam er maar één bug naar boven. De twee de review van de simulatie tool was wel goed gekeurd. Voor het process beschrijving kan je naar hoofdstuk 8 gaan.

Plug-in De plug-in wordt ook geschreven in de google code style. Er moeten tests geschreven worden. Aan het eind van sprint zes wordt de simulatie tool gereviewd. Dit was goed gevonden door de begeleider. Alleen

de test voor het unregisteren als Sense afgesloten wordt ging fout. Voor het process beschrijving kan je naar hoofdstuk 10 gaan.

Afstudeerverslag Het afstudeer verslag wordt gelezen en zo nodig verbetert of commentaar geleverd door de bedrijfsbegeleider. De grammatica wordt gecontroleerd door Gianna Brus. Dit is inderdaad allebei gedaan. Een concept van het verslag is ook in week 15 ingeleverd bij Cobie van der Hoek. Hier zijn een paar puntjes uitgekomen waar nog naar gekeken moest worden.

14.2 Beroepstaken

In dit paragraaf worden de beroepstaken geëvalueerd die in het afstudeer plan (bijlage 2) besproken zijn.
A1 Analyseren van het probleemdomein Voor deze beroepstaak is er zoals in het afstudeer plan staat een onderzoek rapport gemaakt deze kan je vinden als bijlage 4.

A5 Opstellen van systeemeisen (requirements) Deze eisen zijn succesvol opgesteld samen met de bedrijfsbegeleider. Tijdens het project zijn er een paar eisen bijgekomen. Deze eisen kunnen gevonden worden in het requirements document als bijlage 5.

C8 Ontwerpen van een technisch informatie systeem Voor dit project zijn er verschillende klassendiagrammen en sequence diagrammen gemaakt. Deze kan je vinden in bijlage 1. Van sommige van de diagrammen zijn er meerdere versies doordat er na een review een paar verbeter puntjes opgekomen waren.

D17 Testen van software systemen Voor dit beroepstaak is er inderdaad een testdocument gemaakt waar alle eisen getest worden. Daarnaast is er voor elke functie een unit test gemaakt. Naast deze twee tests die in het afstudeer plan staat zijn er ook nog geautomatiseerde test gemaakt volgens de test cases van het testdocument.

G1 Praktische aspecten hanteren in (internationale) projecten Voor dit project is deze beroepstaak uitgevoerd. Er is een plan van aanpak gemaakt met de genoemde hoofdstukken uit de beroepstaak.

14.3 Proces

Het proces is niet helemaal verlopen volgens de planning. Dit kwam doordat na een paar reviews onderdelen weer opnieuw ontworpen of ontwikkeld moesten worden.

14.4 Doel

De doelstelling van dit project is: "De doelstelling van de afstudeeropdracht is door middel van een onderzoek meer duidelijkheid te krijgen wat de ARC-bus is en wat deze kan. Met behulp van dit onderzoek worden requirements geïdentificeerd waarmee de ARC-bus geïntegreerd moet gaan worden in de VDG Sense software. Door middel van een plug-in moeten berichten over de ARC-bus uit gewisseld kunnen worden."

Het doel is gehaald. Er is een onderzoek naar de ARC-bus gedaan. Er zijn requirements opgesteld (zie bijlage 5: requirements document). De plug-in is gemaakt en er kunnen berichten van de ene Sense server naar de andere gestuurd worden.

14.5 Probleem

De probleemstelling van dit project is: "Om de samenwerking/integratie van verschillende producten op de middellange termijn te verbeteren, heeft de TKH innovatieafdeling de ARC-bus bedacht. Dit is een softwarebus waarmee het mogelijk gemaakt moet worden dat verschillende producten middels een netwerkverbinding, gegevens met elkaar kunnen uitwisselen op een uniforme manier. Een ontvanger kan zich abonneren op berichten die verstuurd worden over de ARC-bus en het is mogelijk om zelf berichten te versturen waar andere zich weer op kunnen abonneren. VDG wilt de software voorbereiden op de komst van de ARC-bus en dit integreren in hun interne systeem VDG Sense. De ARC-bus is een nieuw concept en het is voor de VDG dan ook nog niet volledig duidelijk op welke wijze dit gedaan moet worden."

Het grootste deel van het probleem is verholpen. De huidige release van Sense is nog niet klaar voor de ARC-bus plug-in. De ARC-bus build van Sense heeft aanpassingen in de code om dit wel werkend te krijgen. Dit was snel gemaakt en tijdens het ontwikkelen van de plug-in zijn hier verschillende problemen naar boven gekomen waardoor dit onderdeel nog niet leverbaar is. VDG zou nog een keer goed naar dat onderdeel willen

kijken. Er is één ander bedrijf dat de ARC-bus al geïntegreerd heeft maar die berichten zijn binair en worden alleen intern gebruikt. Voor de rest is er nog niet een ander product naast sense waarmee dit getest kan worden.

Literatuurlijst

[1] R Grit. *Projectmanagement: Projectmatig Werken in De Praktijk*. 2008. URL: http://hoadd.noordhoff.nl/sites/7720/_assets/7720d24.pdf (bezocht op 21-09-2015).

[2] Google. *Google C++ Style Guide*. URL: https://google-styleguide.googlecode.com/svn/trunk/cppguide.html#Ownership_and_Smart_Pointers (bezocht op 15-09-2015).

[3] *Bitbucket*. URL: https://bitbucket.org/ (bezocht op 16-09-2015).

[4] Apache. *ActiveMQ*. URL: http://activemq.apache.org/ (bezocht op 29-09-2015).

[5] Apache. *Qpid proton*. URL: https://qpid.apache.org/proton/ (bezocht op 29-09-2015).

[6] TKH innovations. *ARC-bus documentatie*. URL: https://tkhinnovations.atlassian.net/wiki/display/EB/Architecture (bezocht op 28-09-2015).

[7] TKH innovations. *arc-bus-sdk git*. URL: https://git.tkhinnovations.com/development-center/arc-bus-sdk (bezocht op 28-09-2015).

[8] Google. *Google Test*. URL: https://github.com/google/googletest (bezocht op 04-11-2015).

[9] Google. *Introduction: Why Google C++ Testing Framework?* URL: https://github.com/google/googletest/blob/master/googletest/docs/Primer.md (bezocht op 04-11-2015).

[10] Google. *What Is Google C++ Mocking Framework?* URL: https://github.com/google/googletest/blob/master/googlemock/docs/ForDummies.md (bezocht op 07-12-2015).

[11] Google. *CheatSheet*. URL: https://github.com/google/googletest/blob/master/googlemock/docs/CheatSheet.md (bezocht op 07-12-2015).

[12] J Toebes. *Scrumban*. 2014. URL: http://blog.xebia.com/scrumban/ (bezocht op 21-01-2016).

[13] P Kruchten. *The rational unified process an introduction. (3e druk)*. Boston: Pearson Education, inc, 2004.

[14] uml-diagrams. *UML Association*. URL: http://www.uml-diagrams.org/association.html?context=class-diagrams (bezocht op 07-01-2016).

[15] wikipedia. *Waterfall model*. URL: https://en.wikipedia.org/wiki/Waterfall_model (bezocht op 22-09-2015).

[16] K Beck en C Andres. *Extreme Programming Explained. (2e druk)*. Upper Saddle River: Pearson Education, inc, 2004.

[17] Wikipedia. *Universal Plug and Play*. URL: https://en.wikipedia.org/wiki/Universal_Plug_and_Play (bezocht op 30-09-2015).

[18] Atlassian. *Jira*. URL: https://vdgsecurity.atlassian.net/ (bezocht op 21-09-2015).

[19] VDG. *VDG Security B.V.* URL: https://vdgsecurity.atlassian.net/ (bezocht op 21-09-2015).

[20] Wikipedia. *Software development kit*. URL: https://en.wikipedia.org/wiki/Software_development_kit (bezocht op 22-12-2015).

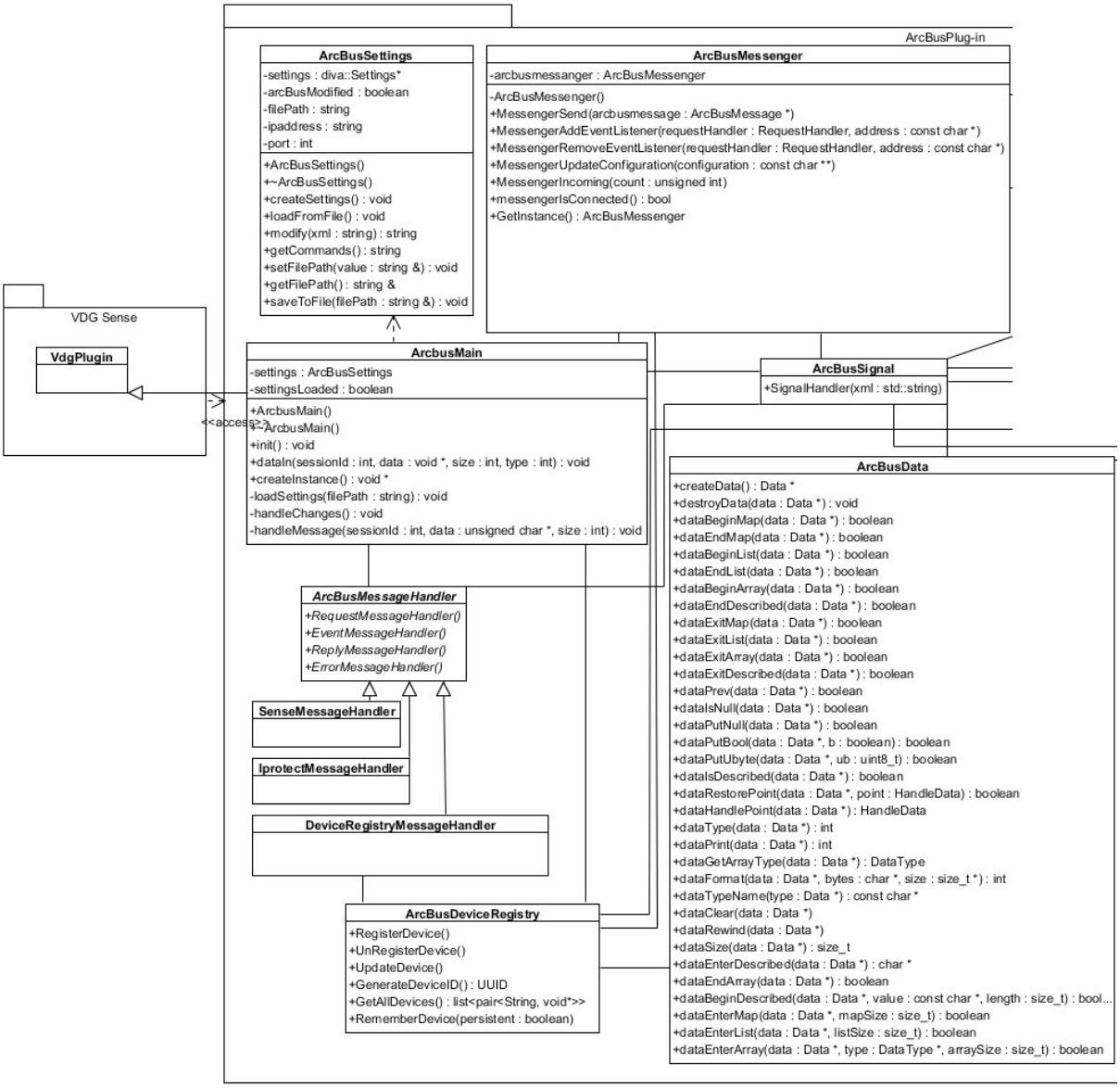
Begrippenlijst

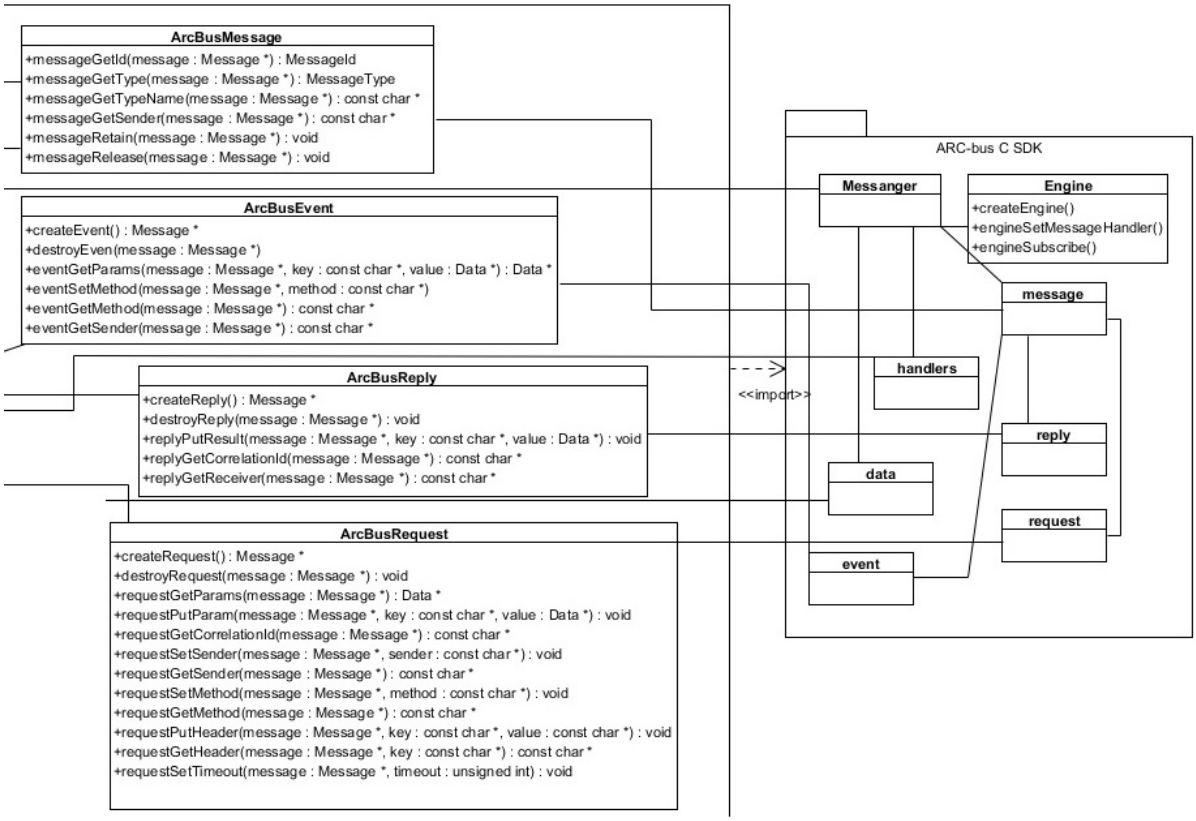
Iteratie: Een project kan één of meer iteratie hebben. Een iteratie is een ontwikkelcyclus. In elke cyclus worden onderdelen van het ontwikkel proces herhaalt.
UPnP: Een set van netwerkprotocollen die netwerkapparaten, zoals pc's, printers, Internet-gateways, Wi-Fi toegangspunten en mobiele apparaten mogelijk maakt naadloos elkaars aanwezigheid ontdekken op het netwerk en stellen een netwerk service in voor het delen van gegevens, communicatie en entertainment.

Bijlage

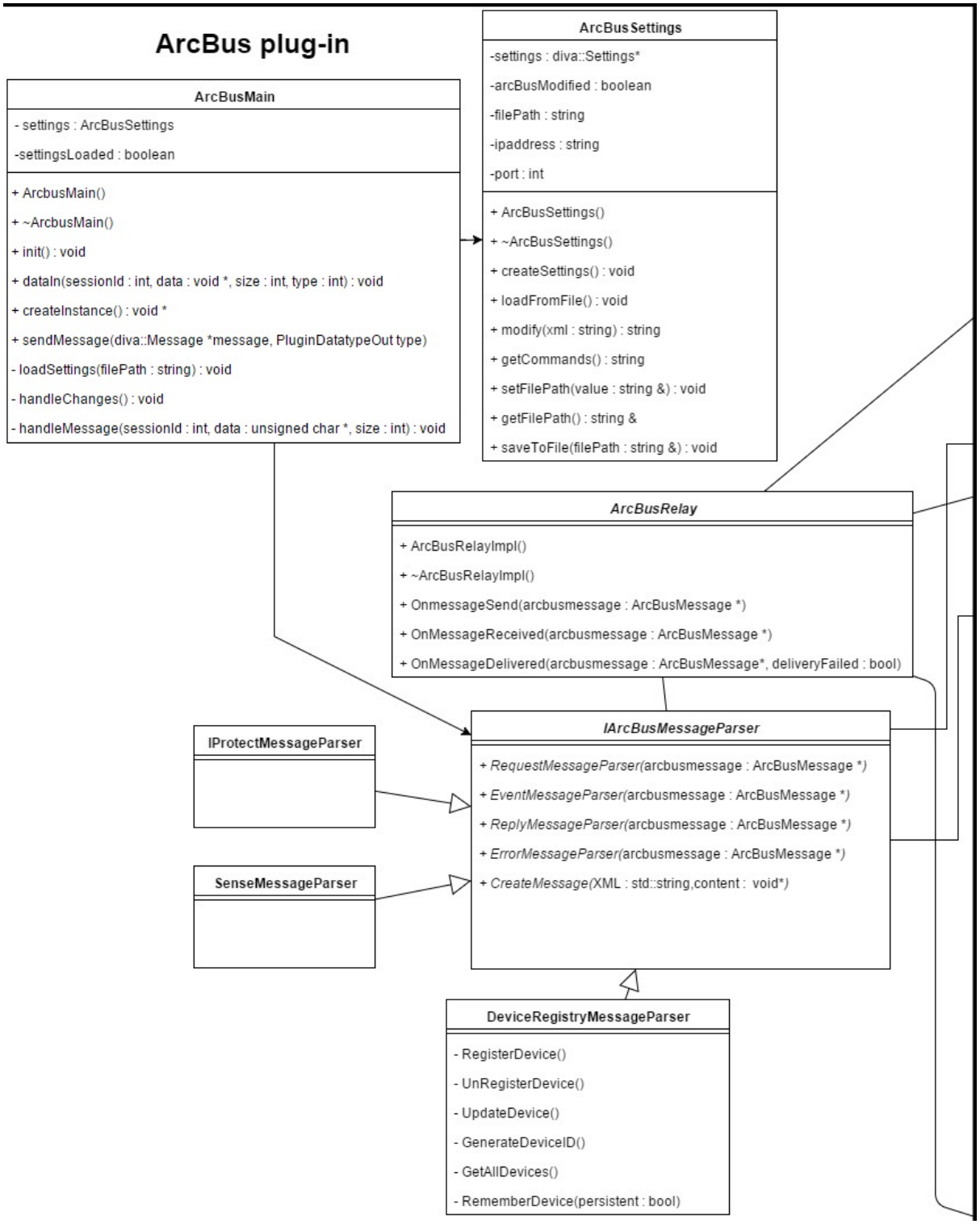
UML diagrammen

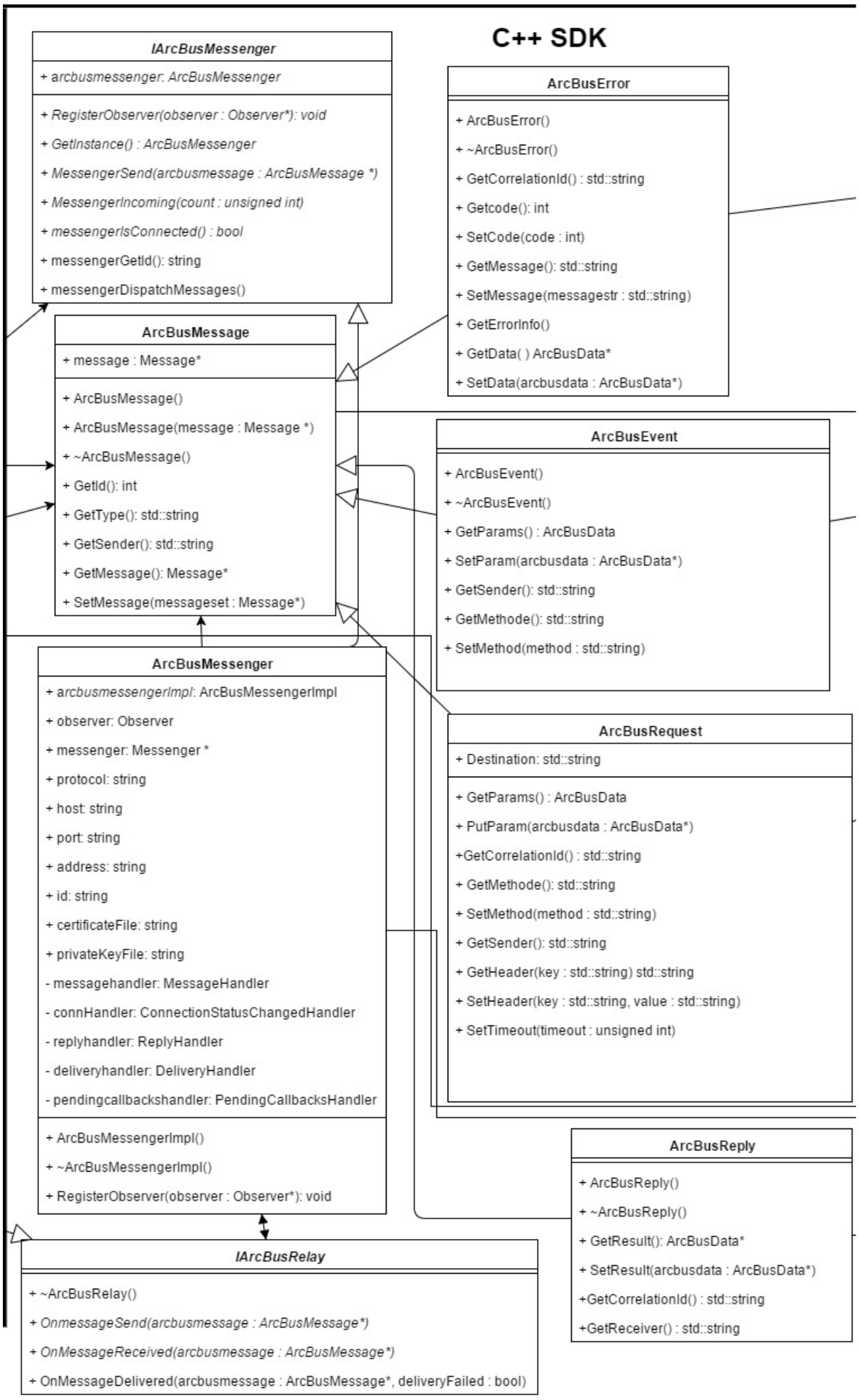
Plug-in klassendiagram versie 1



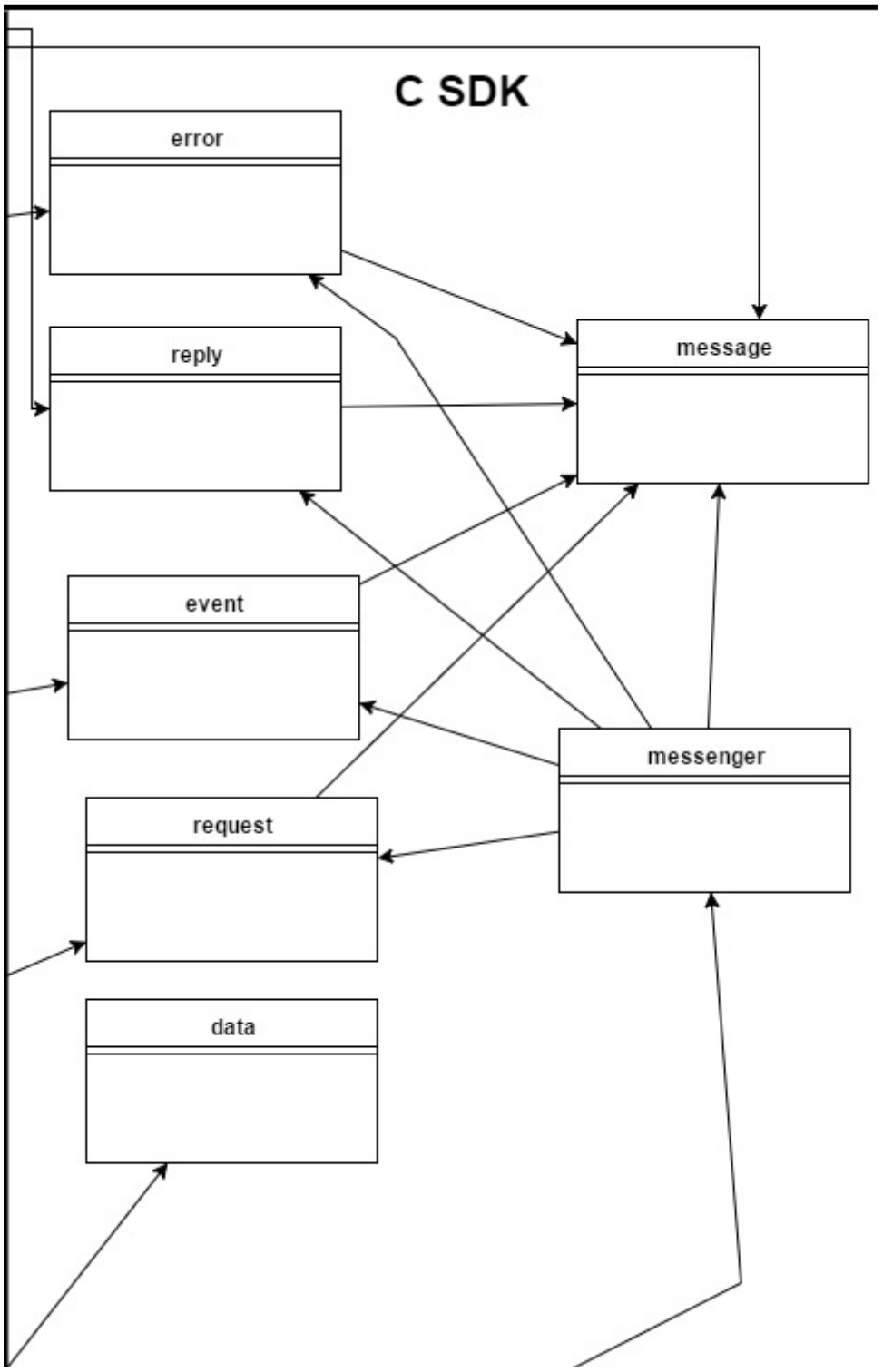

ArcBusMessage
+messageGetId(message : Message *) : MessageId
+messageGetType(message : Message *) : MessageType
+messageGetTypeName(message : Message *) : const char *
+messageGetSender(message : Message *) : const char *
+messageRetain(message : Message *) : void
+messageRelease(message : Message *) : void
ArcBusEvent
+createEvent() : Message *
+destroyEven(message : Message *)
+eventGetParams(message : Message *, key : const char *, value : Data *) : Data *
+eventSetMethod(message : Message *, method : const char *)
+eventGetMethod(message : Message *) : const char *
+eventGetSender(message : Message *) : const char *
ArcBusReply
+createReply() : Message *
+destroyReply(message : Message *) : void
+replyPutResult(message : Message *, key : const char *, value : Data *) : void
+replyGetCorrelationId(message : Message *) : const char *
+replyGetReceiver(message : Message *) : const char *
ArcBusRequest
+createRequest() : Message *
+destroyRequest(message : Message *) : void
+requestGetParams(message : Message *) : Data *
+requestPutParam(message : Message *, key : const char *, value : Data *) : void
+requestGetCorrelationId(message : Message *) : const char *
+requestSetSender(message : Message *, sender : const char *) : void
+requestGetSender(message : Message *) : const char *
+requestSetMethod(message : Message *, method : const char *) : void
+requestGetMethod(message : Message *) : const char *
+requestPutHeader(message : Message *, key : const char *, value : const char *) : void
+requestGetHeader(message : Message *, key : const char *) : const char *
+requestSetTimeout(message : Message *, timeout : unsigned int) : void
<<import>>
ARC-bus C SDK
Messanger
Engine
+createEngine()
+engineSetMessageHandler()
+engineSubscribe()
message
handlers
reply
data
request
event

Plug-in klassendiagram versie 2

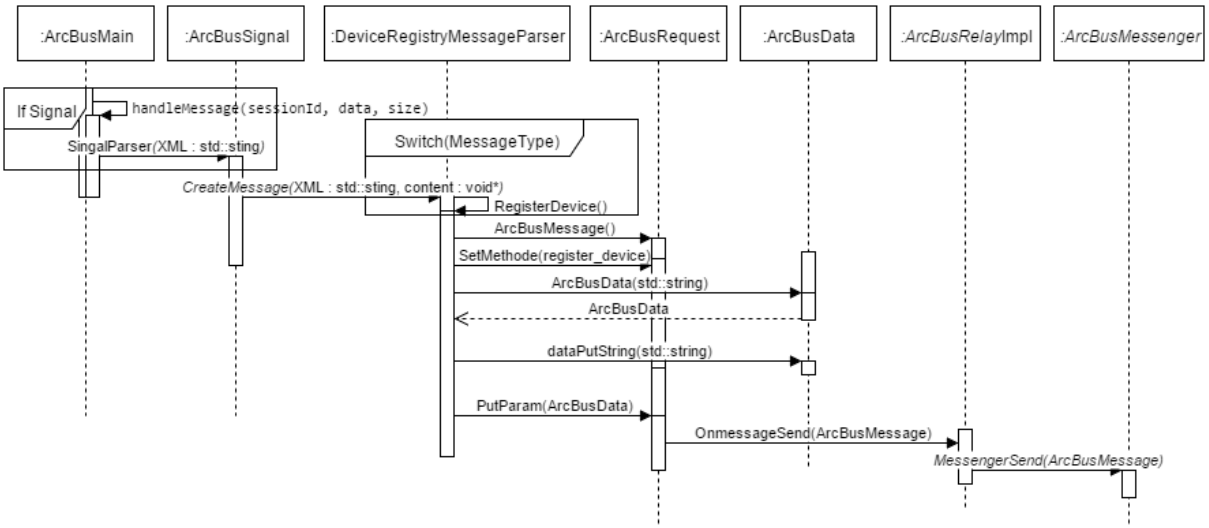



C++ SDK
IArcBusMessenger
+ arcbusmessenger: ArcBusMessenger
+ RegisterObserver(observer : Observer*): void
+ GetInstance() : ArcBusMessenger
+ MessengerSend(arcbusmessage : ArcBusMessage *)
+ MessengerIncoming(count : unsigned int)
+ messengerIsConnected() : bool
+ messengerGetId(): string
+ messengerDispatchMessages()
ArcBusError
+ ArcBusError()
+ ~ArcBusError()
+ GetCorrelationId() : std::string
+ Getcode(): int
+ SetCode(code : int)
+ GetMessage(): std::string
+ SetMessage(messagestr : std::string)
+ GetErrorInfo()
+ GetData( ) ArcBusData*
+ SetData(arcbusdata : ArcBusData*)
ArcBusMessage
+ message : Message*
+ ArcBusMessage()
+ ArcBusMessage(message : Message *)
+ ~ArcBusMessage()
+ GetId(): int
+ GetType(): std::string
+ GetSender(): std::string
+ GetMessage(): Message*
+ SetMessage(messageset : Message*)
ArcBusEvent
+ ArcBusEvent()
+ ~ArcBusEvent()
+ GetParams() : ArcBusData
+ SetParam(arcbusdata : ArcBusData*)
+ GetSender(): std::string
+ GetMethode(): std::string
+ SetMethod(method : std::string)
ArcBusMessenger
+ arcbusmessengerImpl: ArcBusMessengerImpl
+ observer: Observer
+ messenger: Messenger *
+ protocol: string
+ host: string
+ port: string
+ address: string
+ id: string
+ certificateFile: string
+ privateKeyFile: string
- messagehandler: MessageHandler
- connHandler: ConnectionStatusChangedHandler
- replyhandler: ReplyHandler
- deliveryhandler: DeliveryHandler
- pendingcallbackshandler: PendingCallbacksHandler
+ ArcBusMessengerImpl()
+ ~ArcBusMessengerImpl()
+ RegisterObserver(observer : Observer*): void
ArcBusRequest
+ Destination: std::string
+ GetParams() : ArcBusData
+ PutParam(arcbusdata : ArcBusData*)
+GetCorrelationId() : std::string
+ GetMethode(): std::string
+ SetMethod(method : std::string)
+ GetSender(): std::string
+ GetHeader(key : std::string) std::string
+ SetHeader(key : std::string, value : std::string)
+ SetTimeout(timeout : unsigned int)
ArcBusReply
+ ArcBusReply()
+ ~ArcBusReply()
+ GetResult(): ArcBusData*
+ SetResult(arcbusdata : ArcBusData*)
+GetCorrelationId() : std::string
+GetReceiver() : std::string
IArcBusRelay
+ ~ArcBusRelay()
+ OnmessageSend(arcbusmessage : ArcBusMessage*)
+ OnMessageReceived(arcbusmessage : ArcBusMessage*)
+ OnMessageDelivered(arcbusmessage : ArcBusMessage*, deliveryFailed : bool)

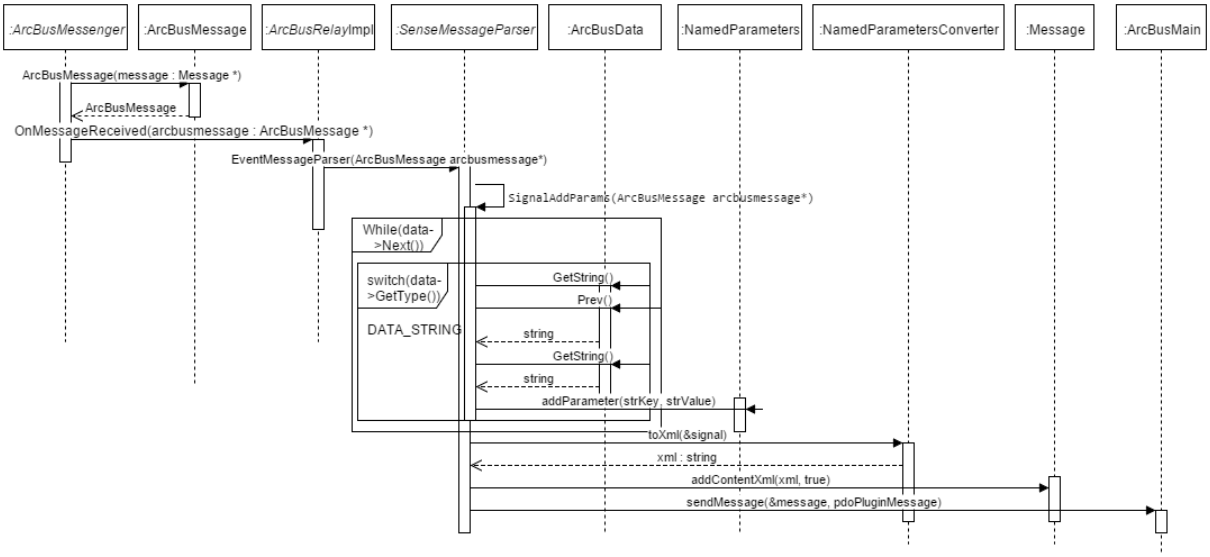

ArcBusData
+ data: Data*
+ key: std::String
+ point: HandleData*
+ ArcBusData()
+ ~ArcBusData()
+ BeginMap(): bool
+ EndMap(): bool
+ BeginList(): bool
+ EndList(): bool
+ BeginArray(): bool
+ EndArray(): bool
+ BeginDescribed(value : std::string , length : size_t): bool
+ EndDescribed(): bool
+ EnterMap(mapSize : size_t*): bool
+ ExitMap(): bool
+ EnterList(listSize : size_t*): bool
+ ExitList(): bool
+ EnterArray(type : DataType*, arraySize : size_t*): bool
+ ExitArray(): bool
+ EnterDescribed(): std::string
+ ExitDescribed(): bool
+ Clear()
+ Size(): size_t
+ Rewind()
+ Prev(): bool
+ IsNull(): bool
+ GetType(): int
+ GetTypeName(): std::string
+ IsDescribed(): bool
+ HandlePoint(): Handle
+ RestorePoint(HandleData point): bool
+ PutNull(): bool
+ PutBool(boolvalue : bool): bool
+ PutUbyte(ubyte : unsigned char): bool
+ PutByte(byte : char): bool
+ PutUshort(ushort : unsigned short): bool
+ PutShort(Short : short) : bool
+ PutUint(uint : unsigned int) : bool
+ PutInt(intvalue : int): bool
+ PutChar(char32bit : unsigned int): bool
+ PutUlong(ulong : uint64_t): bool
+ PutLong(longvalue : int64_t) : bool
+ PutTimestamp(timestamp : uint64_t) : bool
+ PutFloat(floatvalue : float): bool
+ PutDouble(doublevalue : double): bool
+ PutDecimal32(decimal32 : unsigned int) : bool
+ PutDecimal64(decimal64 : uint64_t): bool
+ PutDecimal128(decimal128 : DataDecimal128): bool
+ PutUuid(uuid : DataUuid *): bool
+ PutBinary(value : std::string, length : size_t): bool
+ PutString(value : std::string): bool
+ PutSymbol(value : std::string, length : size_t): bool
+ GetBool(): bool
+ GetUbyte(): unsigned char
+ GetByte(): char
+ GetUshort(): unsigned short
+ GetShort(): short
+ GetUint(): unsigned int
+ GetInt(): int
+ GetChar(): unsigned char
+ GetUlong(): uint64_t
+ GetLong(): int64_t
+ GetTimestamp(): int64_t
+ GetFloat(): float
+ GetDouble(): double
+ GetDecimal32(): unsigned int
+ GetDecimal64(): uint64_t
+ GetDecimal128(): DataDecimal128
+ GetUuid(): DataUuid
+ GetBinary(length : size_t*): std::string
+ GetString(): std::string
+ GetSymbol(length : size_t*): std::string
+ GetBytes(length : size_t*): std::string
+ GetMapValue(key : std::string): ArcBusData


C SDK
error
reply
message
event
messenger
request
data

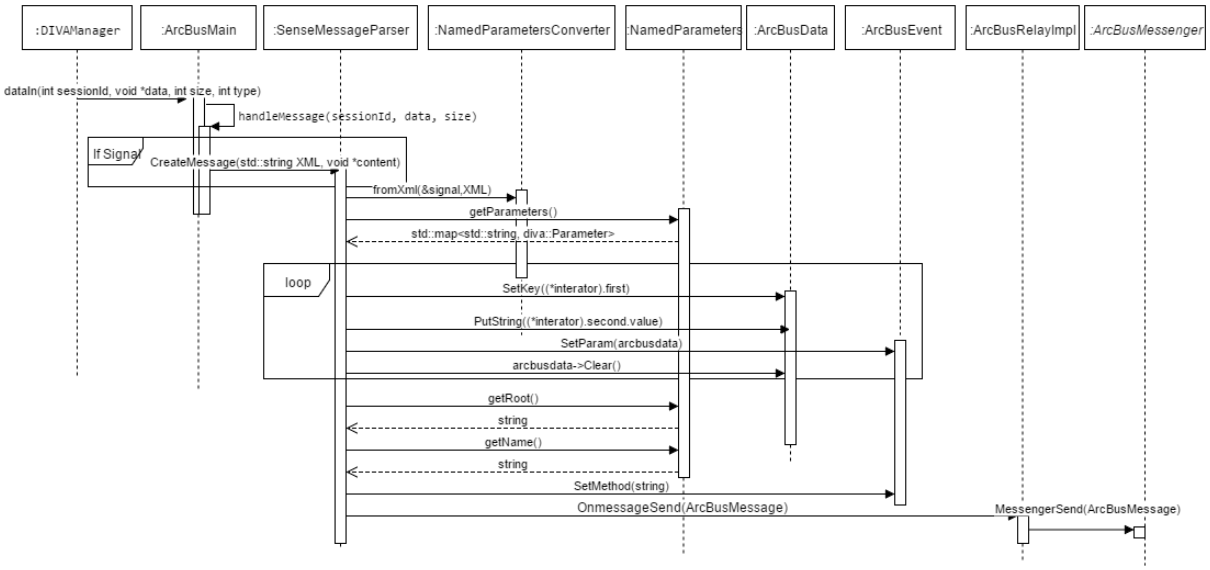
Plug-in registreren deviceregistery versie 2



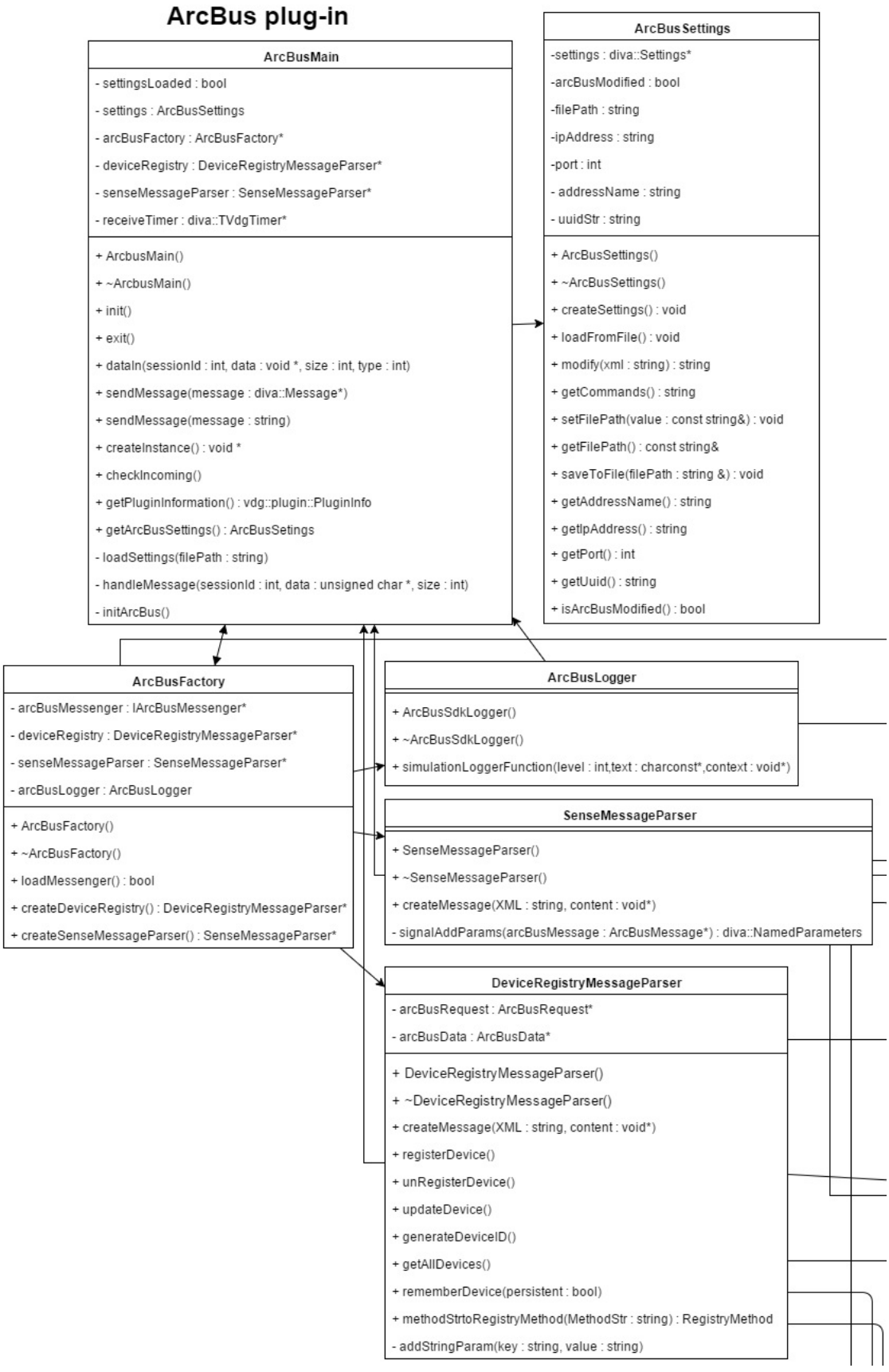
Plug-in sequence receive event versie 2

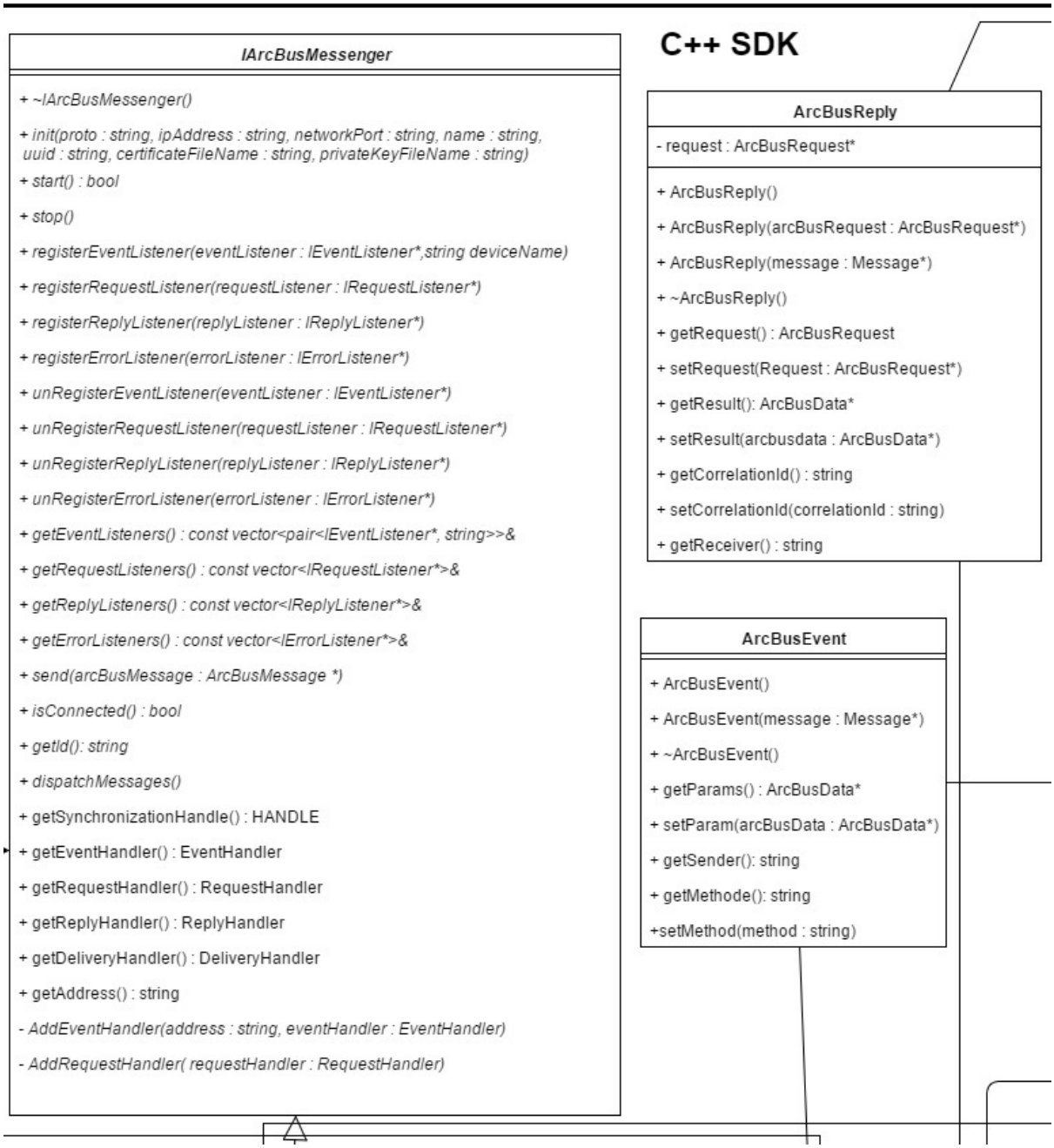


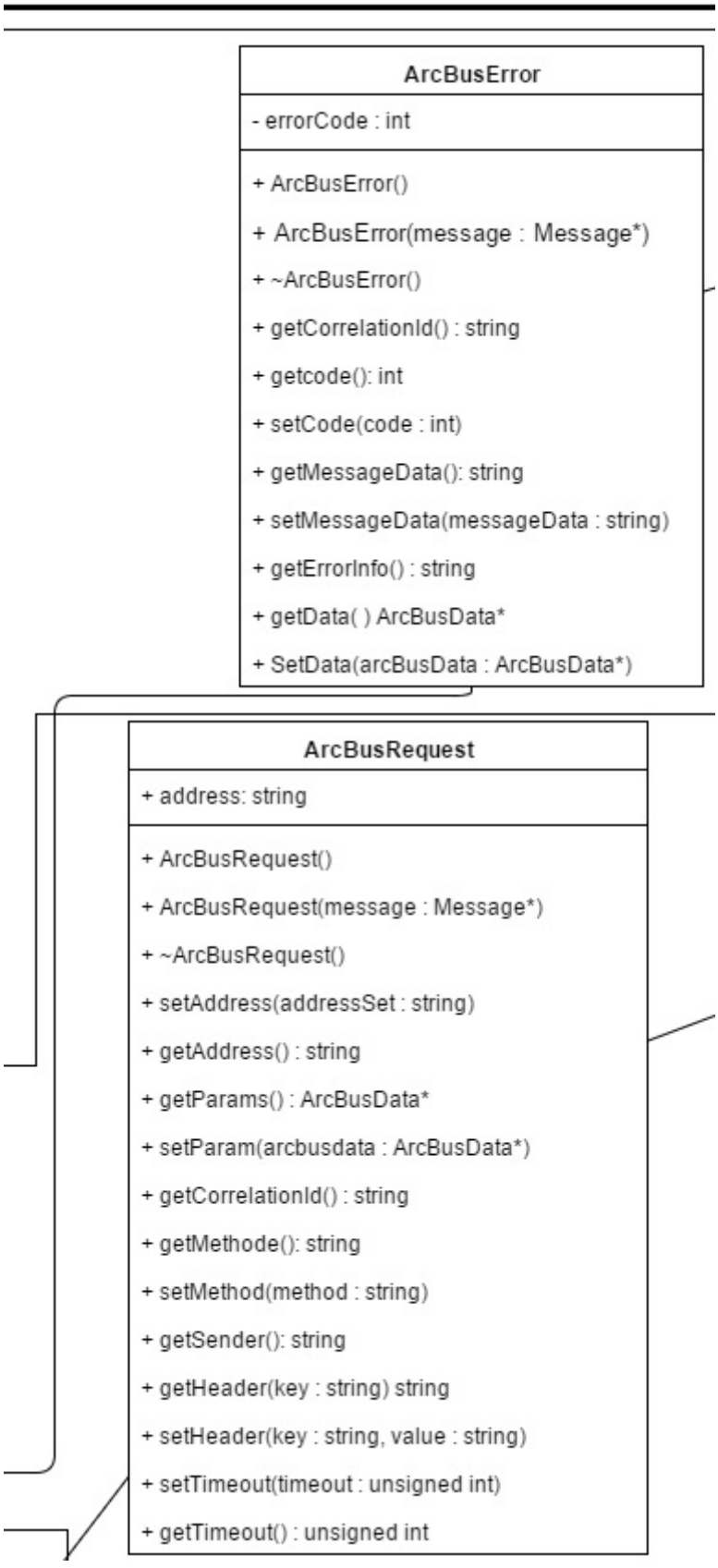
Plug-in sequence send event versie 2

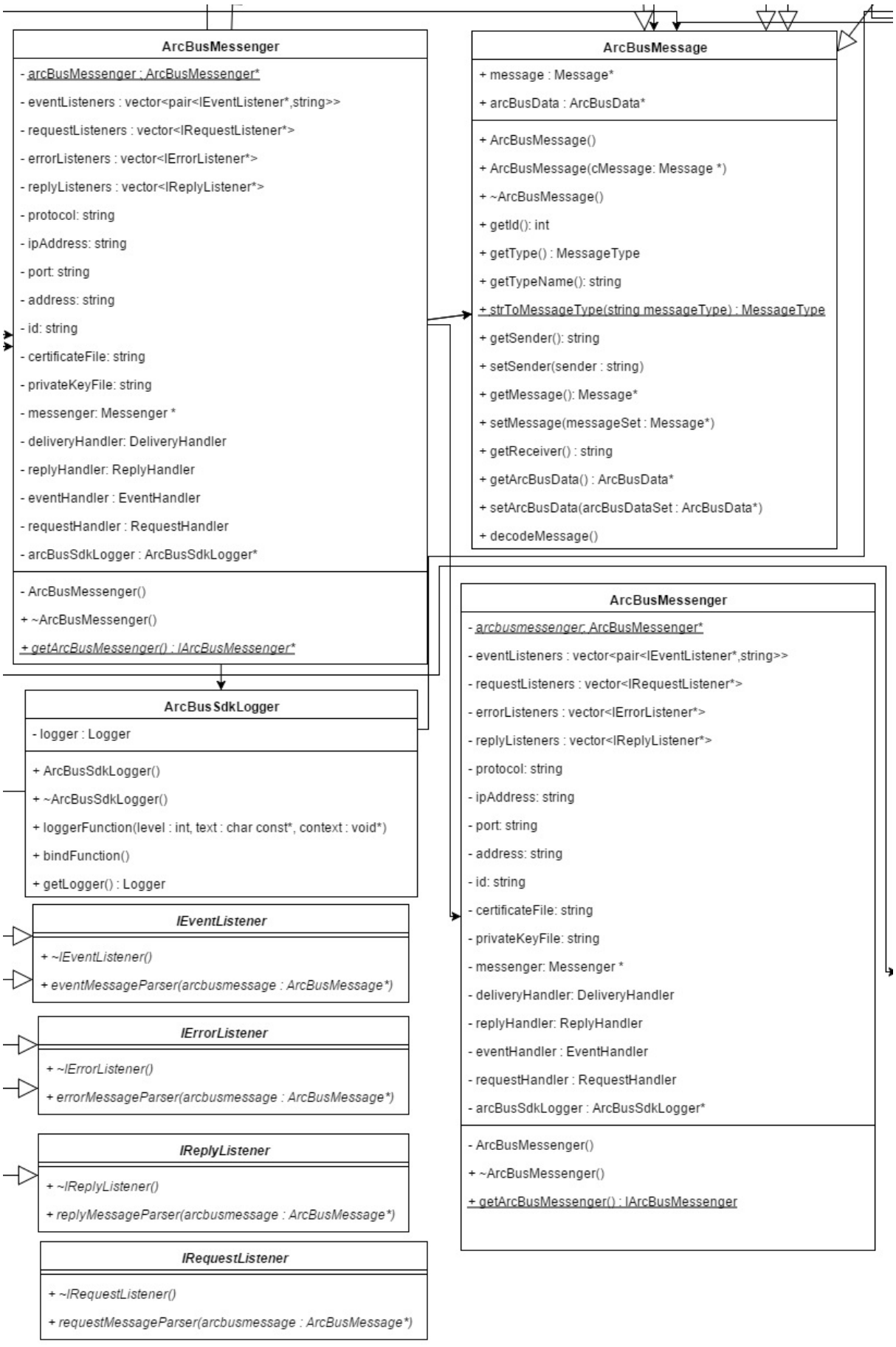


Plug-in klassendiagram versie 3

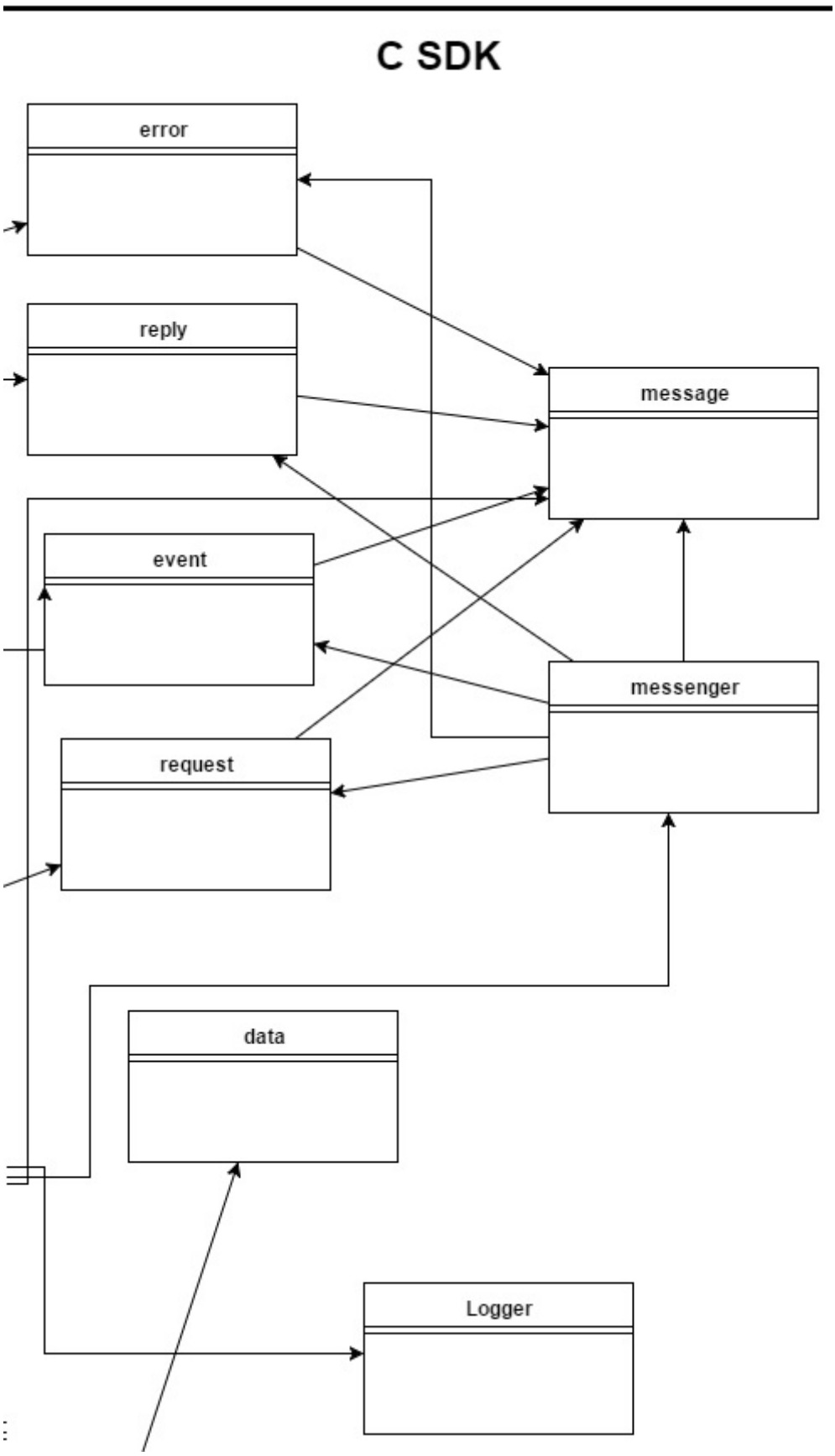



C++ SDK
IArcBusMessenger
+ ~IArcBusMessenger()
+ init(proto : string, ipAddress : string, networkPort : string, name : string, uuid : string, certificateFileName : string, privateKeyFileName : string)
+ start() : bool
+ stop()
+ registerEventListener(eventListener : IEventListener*,string deviceName)
+ registerRequestListener(requestListener : IRequestListener*)
+ registerReplyListener(replyListener : IReplyListener*)
+ registerErrorListener(errorListener : IErrorListener*)
+ unRegisterEventListener(eventListener : IEventListener*)
+ unRegisterRequestListener(requestListener : IRequestListener*)
+ unRegisterReplyListener(replyListener : IReplyListener*)
+ unRegisterErrorListener(errorListener : IErrorListener*)
+ getEventListeners() : const vector<pair<IEventListener*, string>>&
+ getRequestListeners() : const vector<IRequestListener*>&
+ getReplyListeners() : const vector<IReplyListener*>&
+ getErrorListeners() : const vector<IErrorListener*>&
+ send(arcBusMessage : ArcBusMessage *)
+ isConnected() : bool
+ getId(): string
+ dispatchMessages()
+ getSynchronizationHandle() : HANDLE
+ getEventHandler() : EventHandler
+ getRequestHandler() : RequestHandler
+ getReplyHandler() : ReplyHandler
+ getDeliveryHandler() : DeliveryHandler
+ getAddress() : string
- AddEventHandler(address : string, eventHandler : EventHandler)
- AddRequestHandler(requestHandler : RequestHandler)
ArcBusReply
- request : ArcBusRequest*
+ ArcBusReply()
+ ArcBusReply(arcBusRequest : ArcBusRequest*)
+ ArcBusReply(message : Message*)
+ ~ArcBusReply()
+ getRequest() : ArcBusRequest
+ setRequest(Request : ArcBusRequest*)
+ getResult(): ArcBusData*
+ setResult(arcbusdata : ArcBusData*)
+ getCorrelationId() : string
+ setCorrelationId(correlationId : string)
+ getReceiver() : string
ArcBusEvent
+ ArcBusEvent()
+ ArcBusEvent(message : Message*)
+ ~ArcBusEvent()
+ getParams() : ArcBusData*
+ setParam(arcBusData : ArcBusData*)
+ getSender(): string
+ getMethode(): string
+setMethod(method : string)

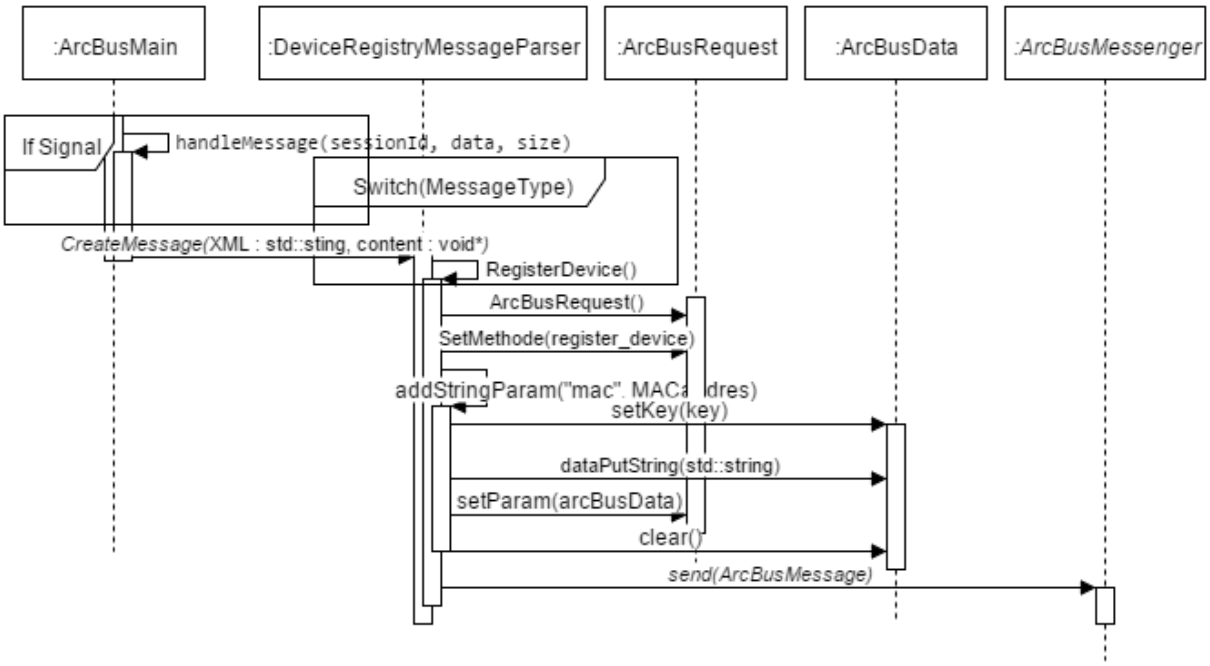

ArcBusError
- errorCode : int
+ ArcBusError()
+ ArcBusError(message : Message*)
+ ~ArcBusError()
+ getCorrelationId() : string
+ getcode(): int
+ setCode(code : int)
+ getMessageData(): string
+ setMessageData(messageData : string)
+ getErrorInfo() : string
+ getData( ) ArcBusData*
+ SetData(arcBusData : ArcBusData*)
ArcBusRequest
+ address: string
+ ArcBusRequest()
+ ArcBusRequest(message : Message*)
+ ~ArcBusRequest()
+ setAddress(addressSet : string)
+ getAddress() : string
+ getParams() : ArcBusData*
+ setParam(arcbusdata : ArcBusData*)
+ getCorrelationId() : string
+ getMethode(): string
+ setMethod(method : string)
+ getSender(): string
+ getHeader(key : string) string
+ setHeader(key : string, value : string)
+ setTimeout(timeout : unsigned int)
+ getTimeout() : unsigned int


ArcBusMessenger
- arcBusMessenger : ArcBusMessenger*
- eventListeners : vector<pair<IEventListener*,string>>
- requestListeners : vector<IRequestListener*>
- errorListeners : vector<IErrorListener*>
- replyListeners : vector<IReplyListener*>
- protocol: string
- ipAddress: string
- port: string
- address: string
- id: string
- certificateFile: string
- privateKeyFile: string
- messenger: Messenger *
- deliveryHandler: DeliveryHandler
- replyHandler: ReplyHandler
- eventHandler : EventHandler
- requestHandler : RequestHandler
- arcBusSdkLogger : ArcBusSdkLogger*
- ArcBusMessenger()
+ ~ArcBusMessenger()
+ getArcBusMessenger() : IArcBusMessenger*
ArcBusMessage
+ message : Message*
+ arcBusData : ArcBusData*
+ ArcBusMessage()
+ ArcBusMessage(cMessage: Message *)
+ ~ArcBusMessage()
+ getId(): int
+ getType() : MessageType
+ getTypeName(): string
+ strToMessageType(string messageType) : MessageType
+ getSender(): string
+ setSender(sender : string)
+ getMessage(): Message*
+ setMessage(messageSet : Message*)
+ getReceiver() : string
+ getArcBusData() : ArcBusData*
+ setArcBusData(arcBusDataSet : ArcBusData*)
+ decodeMessage()
ArcBusSdkLogger
- logger : Logger
+ ArcBusSdkLogger()
+ ~ArcBusSdkLogger()
+ loggerFunction(level : int, text : char const*, context : void*)
+ bindFunction()
+ getLogger() : Logger
ArcBusMessenger
- arcbusmessenger: ArcBusMessenger*
- eventListeners : vector<pair<IEventListener*,string>>
- requestListeners : vector<IRequestListener*>
- errorListeners : vector<IErrorListener*>
- replyListeners : vector<IReplyListener*>
- protocol: string
- ipAddress: string
- port: string
- address: string
- id: string
- certificateFile: string
- privateKeyFile: string
- messenger: Messenger *
- deliveryHandler: DeliveryHandler
- replyHandler: ReplyHandler
- eventHandler : EventHandler
- requestHandler : RequestHandler
- arcBusSdkLogger : ArcBusSdkLogger*
- ArcBusMessenger()
+ ~ArcBusMessenger()
+ getArcBusMessenger() : IArcBusMessenger
IEventListener
+ ~IEventListener()
+ eventMessageParser(arcbusmessage : ArcBusMessage*)
IErrorListener
+ ~IErrorListener()
+ errorMessageParser(arcbusmessage : ArcBusMessage*)
IReplyListener
+ ~IReplyListener()
+ replyMessageParser(arcbusmessage : ArcBusMessage*)
IRequestListener
+ ~IRequestListener()
+ requestMessageParser(arcbusmessage : ArcBusMessage*)

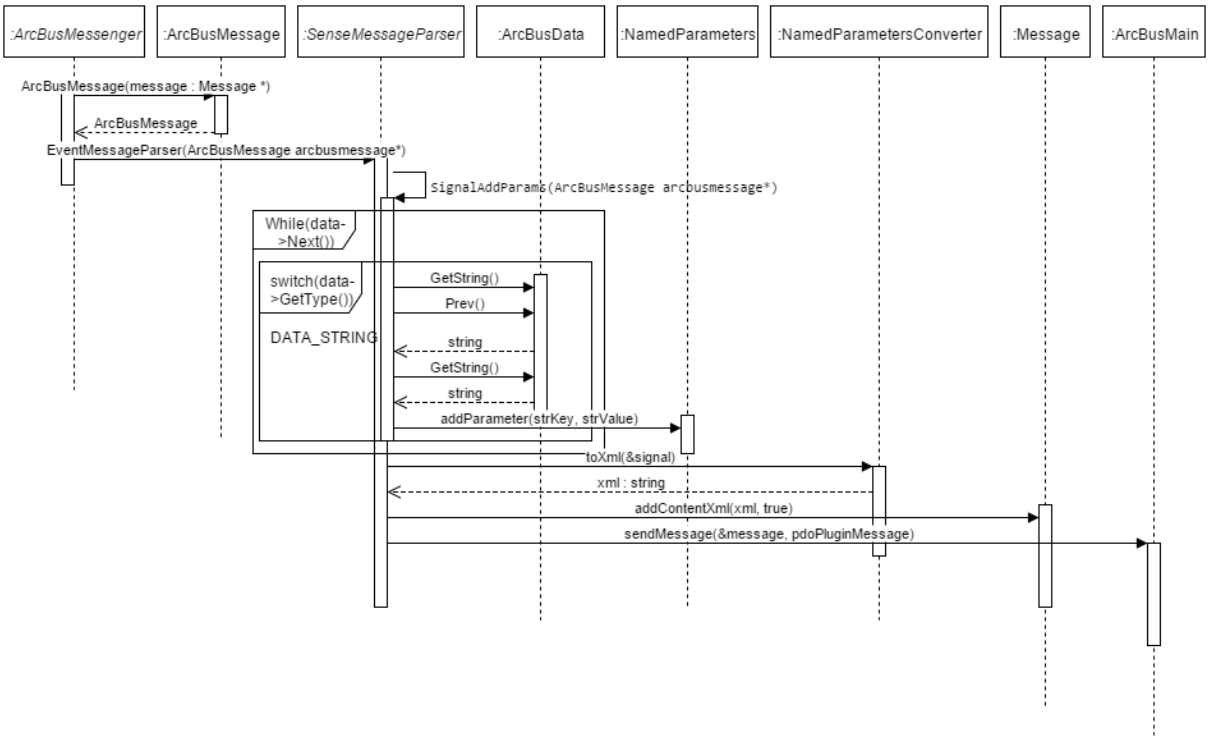

ArcBusData
- data: Data*
- key: String
- point: HandleData*
+ ArcBusData()
+ ArcBusData(datanew : Data*)
+ ArcBusData(datanew : Data*, keynew : string)
+ ~ArcBusData()
+ getKey() : const string
+ setKey(keySet : string)
+ getData() : Data*
+ beginMap(): bool
+ endMap(): bool
+ enterMap(mapSize : size_t*): bool
+ exitMap(): bool
+ beginList(): bool
+ endList(): bool
+ enterList(listSize : size_t*): bool
+ exitList(): bool
+ beginArray(type : DataType): bool
+ endArray(): bool
+ enterArray(type : DataType*, arraySize : size_t*): bool
+ exitArray(): bool
+ beginDescribed(value : string , length : size_t): bool
+ endDescribed(): bool
+ enterDescribed(): string
+ exitDescribed(): bool
+ clear()
+ size(): size_t
+ rewind()
+ prev(): bool
+ next() : bool
+ isNull(): bool
+ getType(): int
+ getTypeName(): string
+ isDescribed(): bool
+ handlePoint(): Handle
+ restorePoint(): bool
+ putNull(): bool
+ putBool(boolvalue : bool): bool
+ putUByte(ubyte : unsigned char): bool
+ putByte(byte : char): bool
+ putUShort(ushort : unsigned short): bool
+ putShort(shortValue : short) : bool
+ putUInt(uint : unsigned int) : bool
+ putInt(intValue : int): bool
+ putChar(charValue : unsigned int): bool
+ putULong(ulong : uint64_t): bool
+ putLong(longValue : int64_t) : bool
+ putTimestamp(timestamp : int64_t) : bool
+ putFloat(floatValue : float): bool
+ putDouble(doubleValue : double): bool
+ putDecimal32(decimal32 : unsigned int) : bool
+ putDecimal64(decimal64 : uint64_t): bool
+ putDecimal128(decimal128 : DataDecimal128): bool
+ putUuid(uuid : DataUuid *): bool
+ putBinary(value : string): bool
+ putString(value : string): bool
+ putSymbol(value : string): bool
+ getBool(): const bool
+ getUByte(): const unsigned char
+ getByte(): const char
+ getUShort() : const unsigned short
+ getShort(): const short
+ getUInt(): const unsigned int
+ getInt(): const int
+ getChar(): const unsigned char
+ getULong(): const uint64_t
+ getLong(): const int64_t
+ getTimestamp(): const int64_t
+ getFloat(): const float
+ getDouble(): const double
+ getDecimal32(): const unsigned int
+ getDecimal64(): const uint64_t
+ getDecimal128(): const DataDecimal128
+ getUuid(): const DataUuid
+ getBinary(length : size_t*): const string
+ getString(): const string
+ getSymbol(length : size_t*): const string
+ getBytes(length : size_t*): const char*
+ getMapValue(key : string): ArcBusData*


C SDK
error
reply
message
event
messenger
request
data
Logger

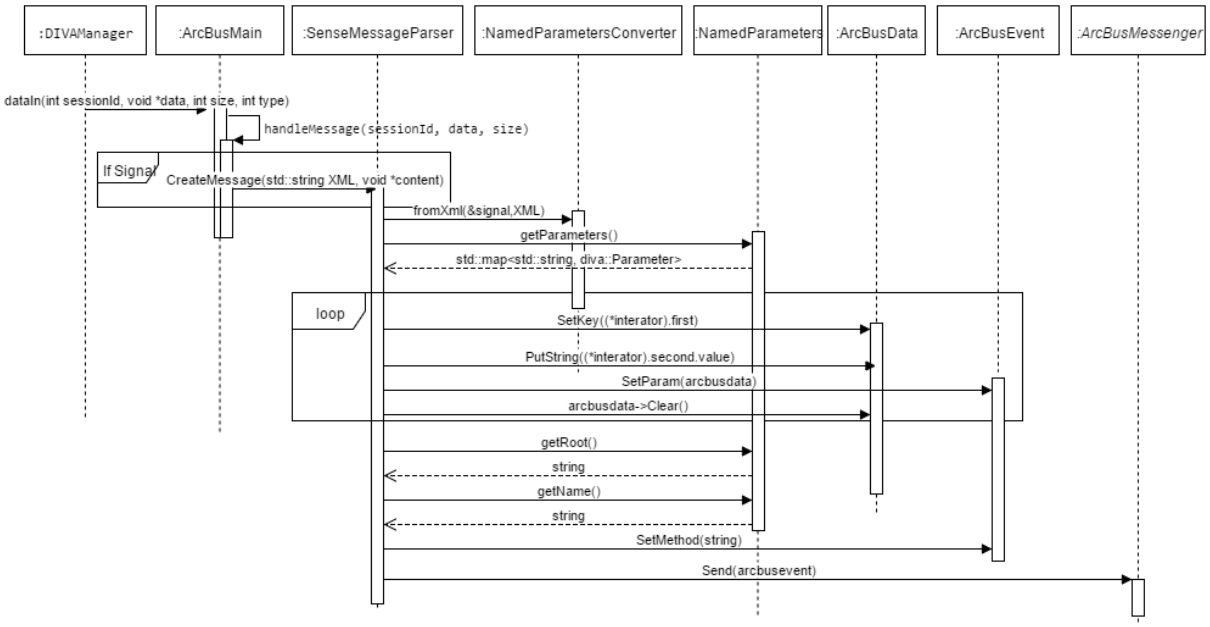
Plug-in registreren deviceregistery versie 3



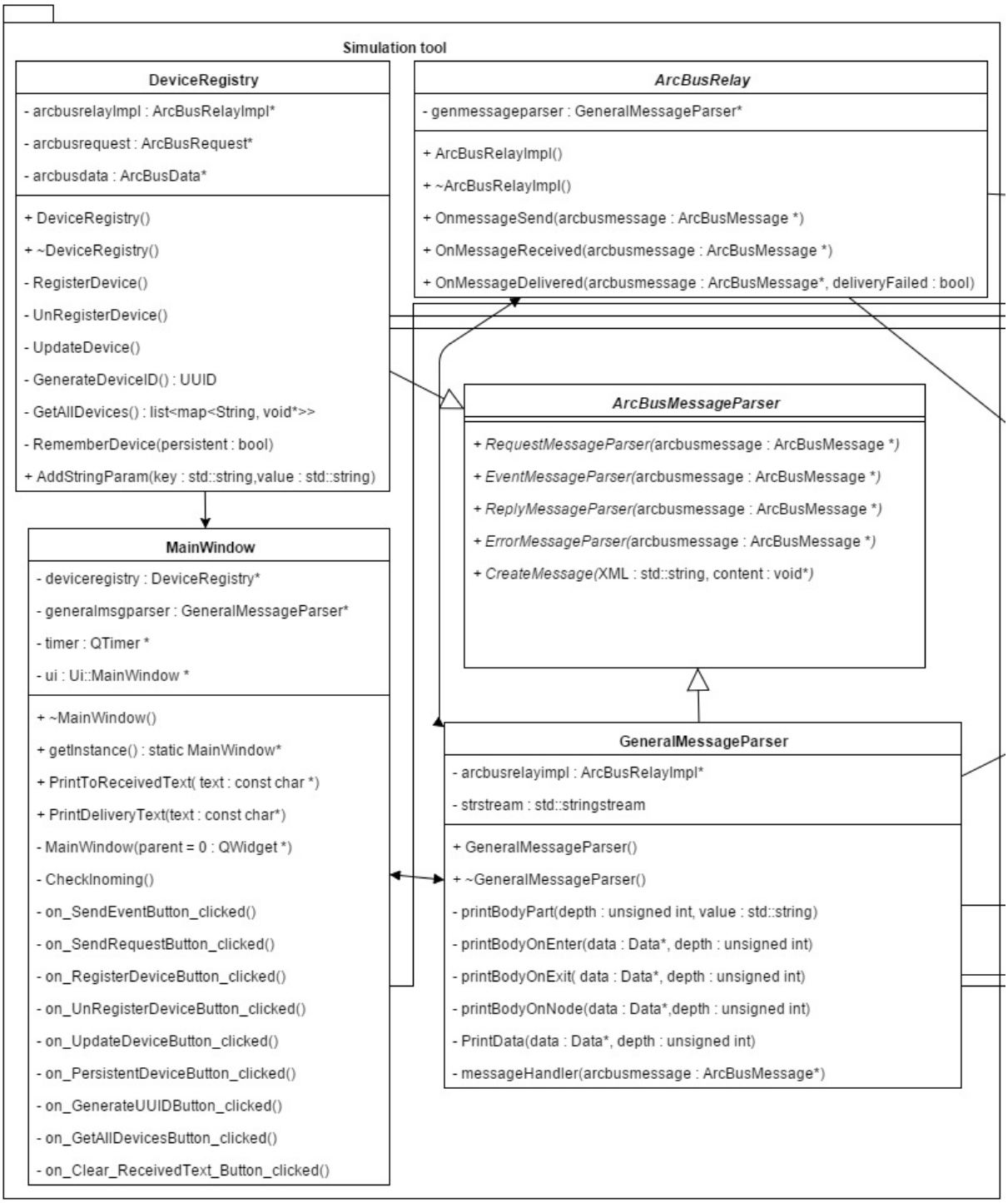
Plug-in sequence receive event versie 3

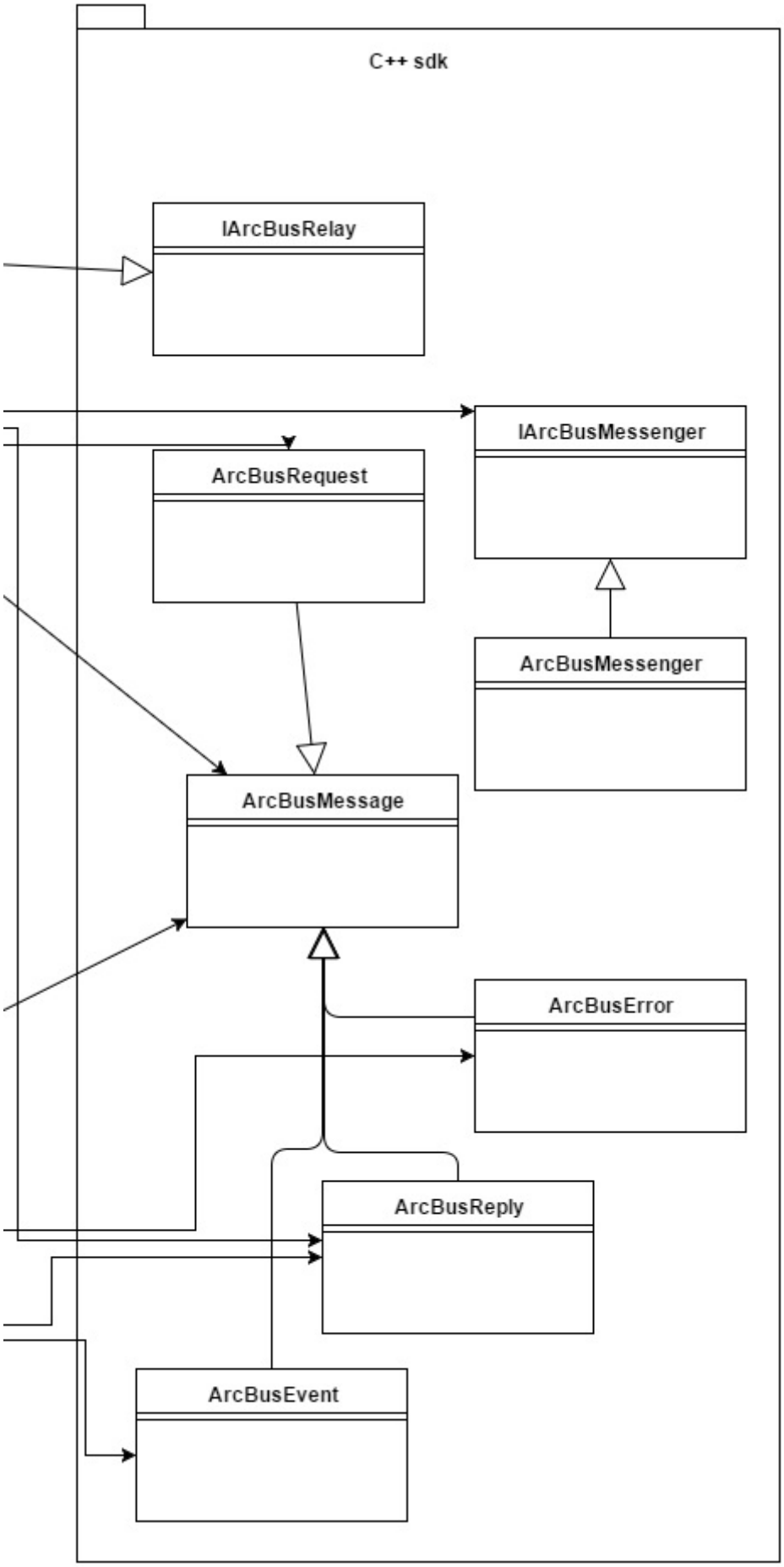


Plug-in sequence send event versie 3

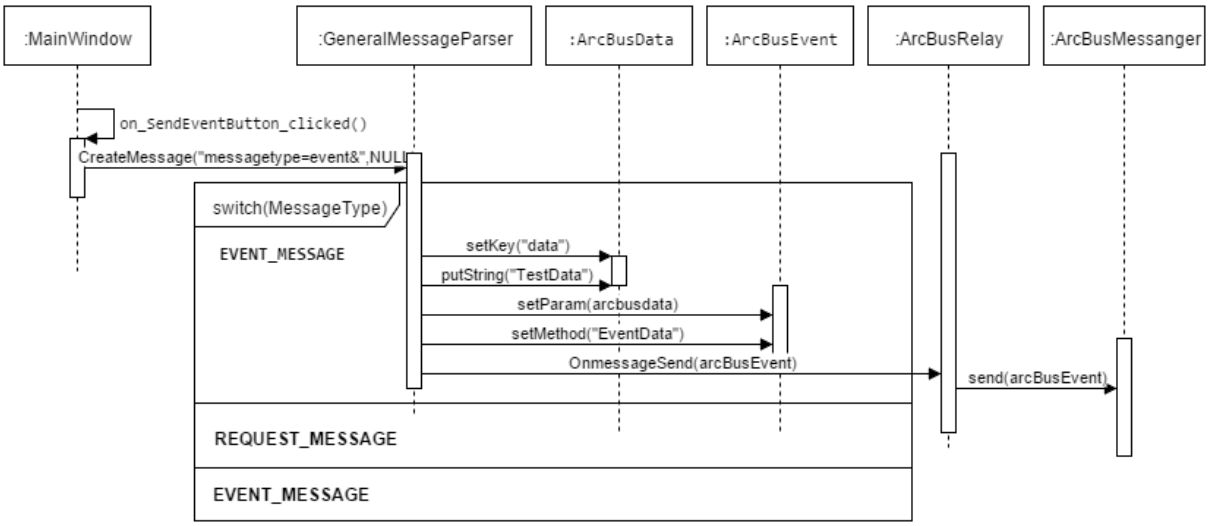


Simulatie tool klassendiagram versie 2

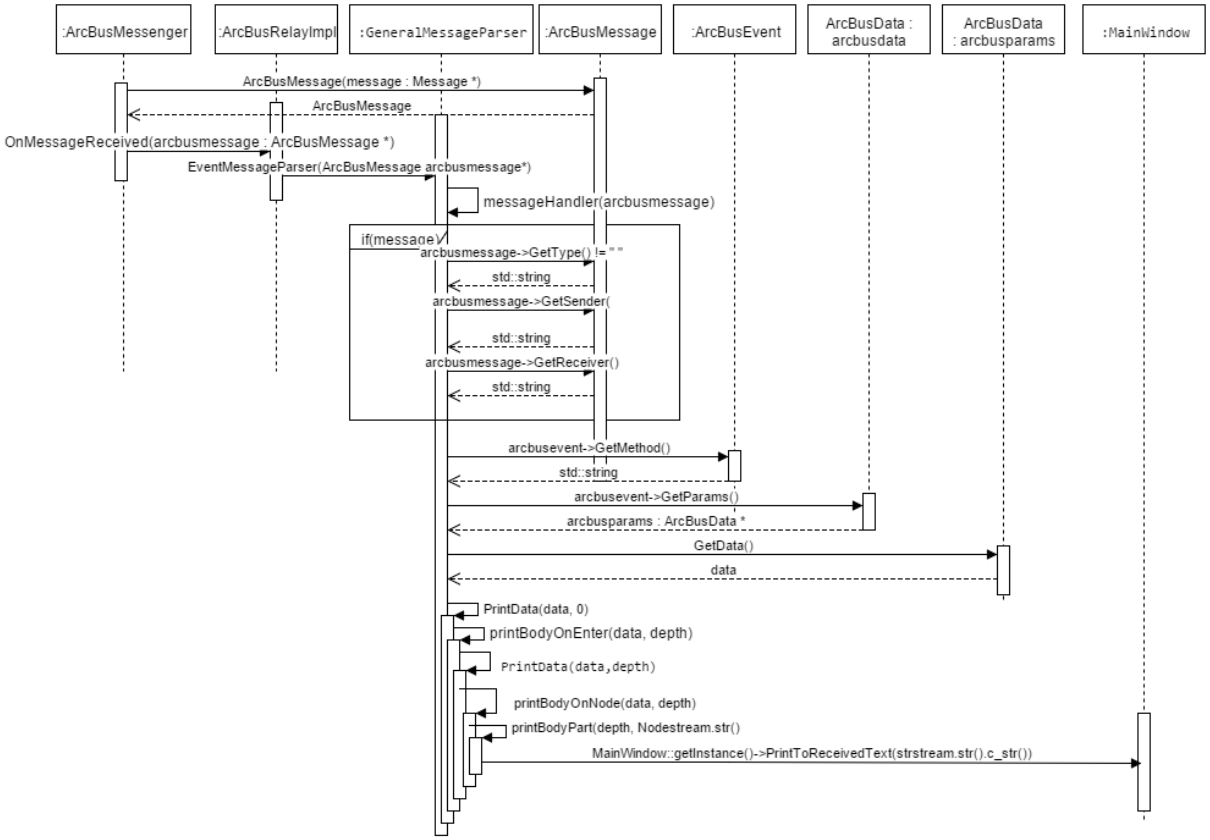



C++ sdk
IArcBusRelay
IArcBusMessenger
ArcBusRequest
ArcBusMessenger
ArcBusMessage
ArcBusError
ArcBusReply
ArcBusEvent

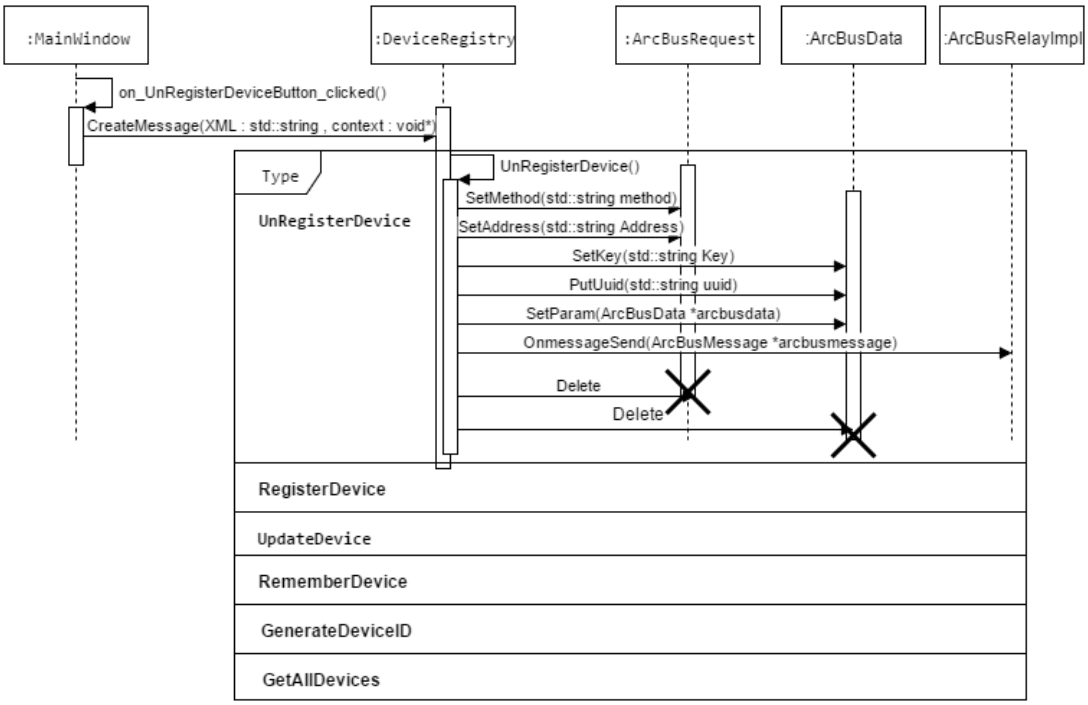
Simulatie tool sequence event send versie 2



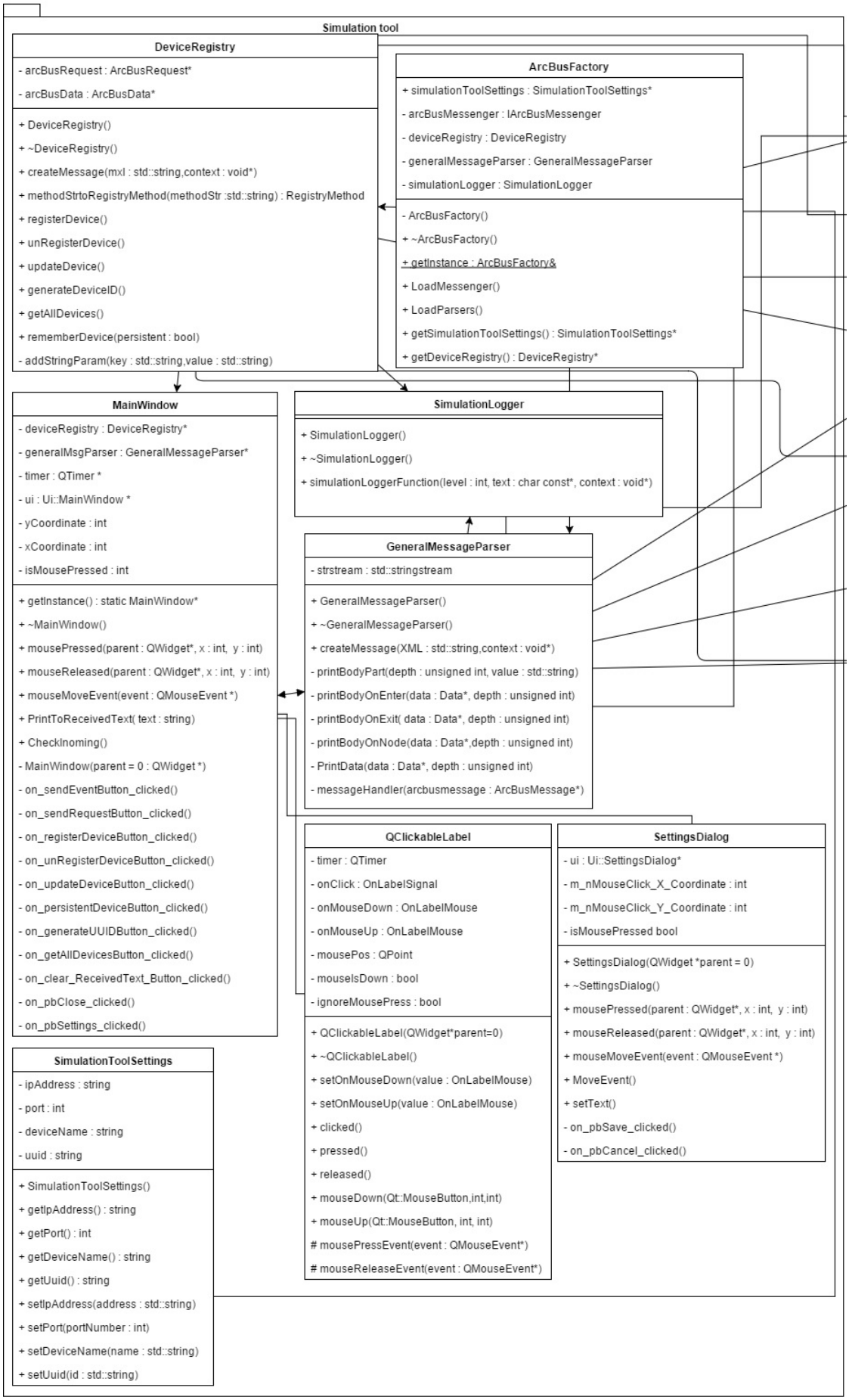
Simulatie tool sequence receive event versie 2

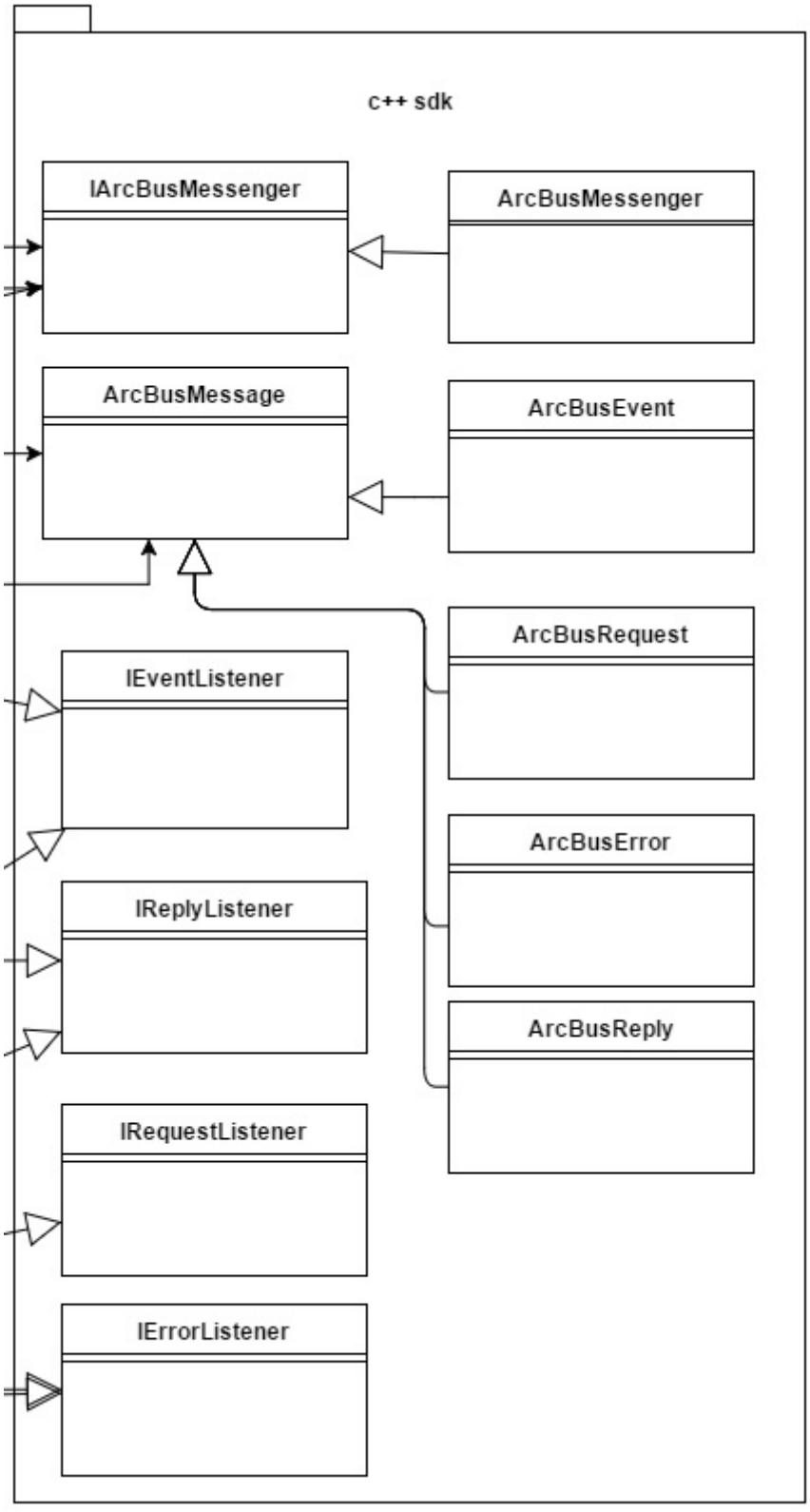


Simulatie tool unregisterdevice versie 2

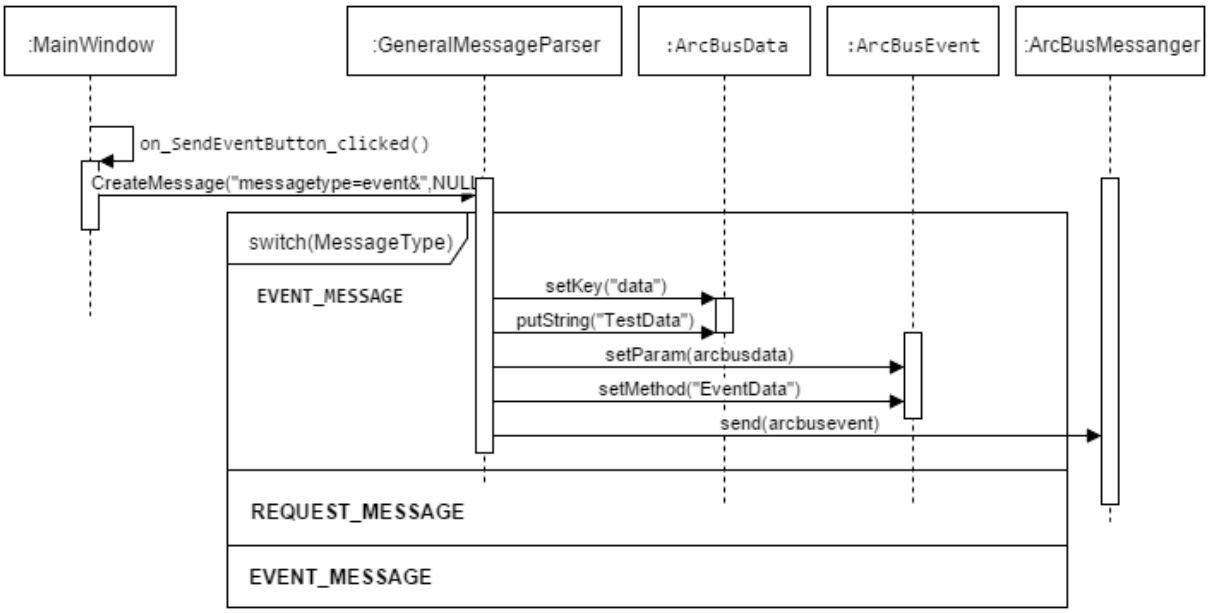


Simulatie tool klassendiagram versie 3

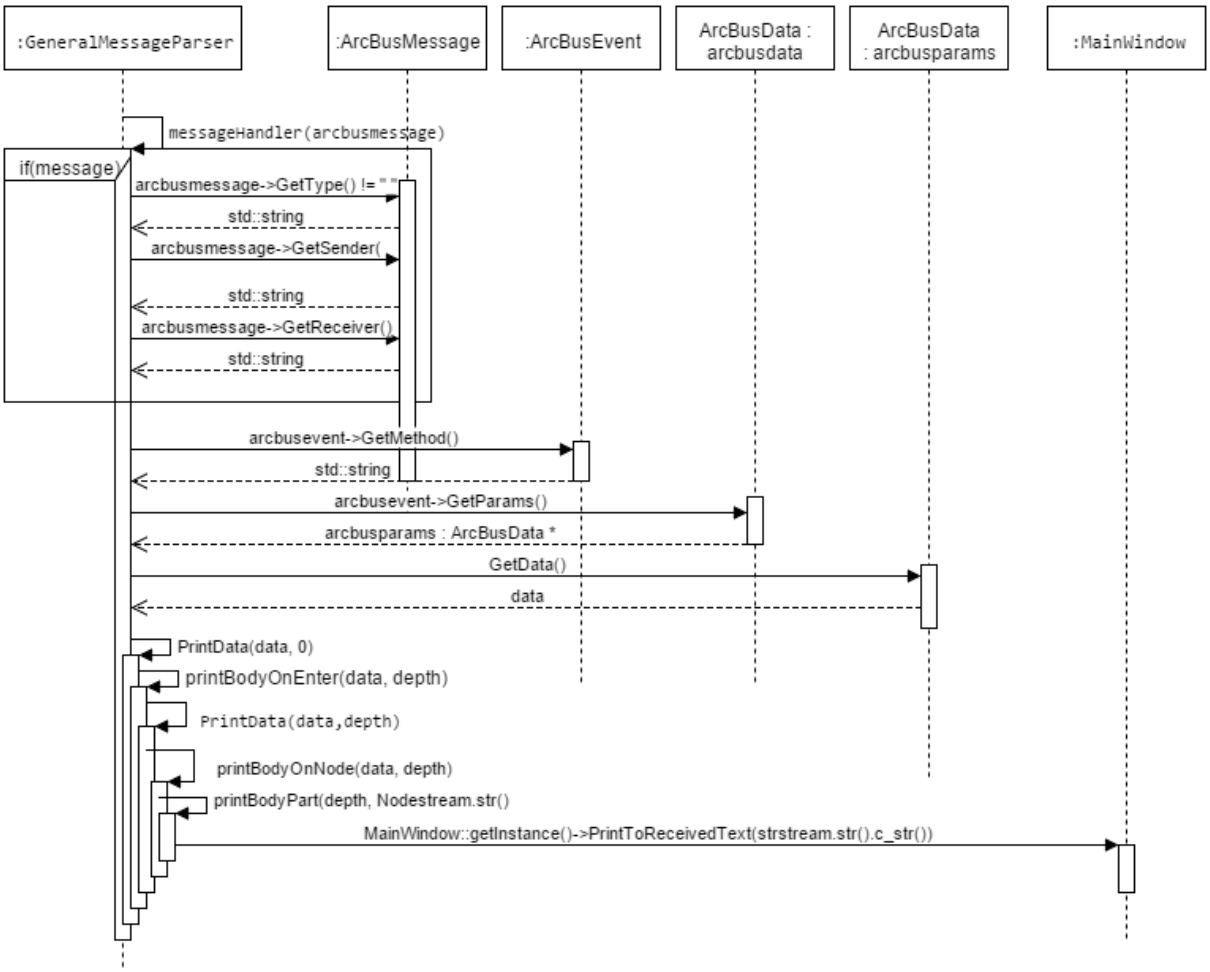



c++ sdk
IArcBusMessenger
ArcBusMessenger
ArcBusMessage
ArcBusEvent
ArcBusRequest
IEventListener
ArcBusError
IReplyListener
ArcBusReply
IRequestListener
IErrorListener

Simulatie tool sequence event send versie 3



Simulatie tool sequence receive event versie 3



Simulatie tool unregisterdevice versie 3