Afstudeerverslag
Propacity Loadrunner Agent

7 februari 2011 t/m 6 juni 2011

Uitvoerende
Edwin Harteveld
07073720
edwin_harteveld@hotmail.com

Opdrachtgever
Michel Stol
mstol@computest.nl

DE HAAGSE HOGESCHOOL

Computest specialists in testing

Computest specialists in testing

Referaat

Auteur: Edwin Harteveld
Titel: Propacity Loadrunner Agent

Organisatie:
Computest
Postbus 653
2700 AR Zoetermeer
Nederland

Onderwijsinstelling:
De Haagse Hogeschool
Academie voor ICT & Media Zoetermeer
Bredewater 24
2715 CA Zoetermeer

Examinatoren:
Arno Nederend
Vincent Broeren

Voorwoord

In de hedendaagse wereld van het internet komen er elke dag nieuwe websites en webapplicaties bij met als doel informatie te verstrekken en/of een dienst te leveren.
De ontwikkeling hiervan wordt door een brede selectie van ontwikkelaars aangeboden van gespecialiseerde bedrijven tot eenmansbedrijven met verschillende prijs/kwaliteitsverhoudingen.

Ongeacht de kwaliteit van een dergelijke applicatie vergeet men vaak dat er een gevaar bestaat dat op het moment van oplevering (nog) niet zichtbaar is. Dit is het verlies van performance veroorzaakt wordt door een stijgend aantal bezoekers/gebruikers of een bottleneck in het netwerkverkeer. Meestal wordt dit gevaar onderschat totdat de applicatie niet meer naar behoren functioneert, wat in dat geval betekent dat het al te laat is om dit probleem te voorkomen en zal leiden tot ontevreden gebruikers en kosten die voorkomen hadden kunnen worden.
Een uitstekend praktijkvoorbeeld is de online kaartverkoop van evenementen. Deze raken vaak overbelast tijdens de verkoop waardoor liefhebbers vaak misgrijpen. Op het moment dat de problemen ontstaan, is reparatie al niet meer mogelijk.

In dit soort situaties is voorkomen beter dan genezen en is het verstandiger om een applicatie te testen op performance voordat deze gebruikt wordt.
Dit is precies wat Computest kan aanbieden. Het schrijft scripts voor de betreffende applicaties/websites en rapporteert daar over.

Echter bestaat er de kans dat er na oplevering, ondanks de performance tests die zijn uitgevoerd, toch problemen kunnen ontstaan. Hier speelt Computest op in door een applicatie te ontwikkelen die deze performance 'real-time' kan meten. Op deze manier kan een stijgend aantal gebruikers lang van tevoren worden herkend en kunnen er patronen van bottlenecks worden vastgesteld.

Tijdens dit afstudeertraject zal er aan een deel van deze applicatie gewerkt worden. De afstudeerperiode is begonnen op 7 februari 2011 en eindigde op 6 juni 2011.
In dit verslag beschrijf ik de werkzaamheden die ik verricht heb bij Computest en het bijhorende proces.

Mijn dank gaat uit naar de volgende mensen voor hun hulp en begeleiding:

Computest
- Michel Stol

De Haagse Hogeschool
- Arno Nederend
- Vincent Broeren

Edwin Harteveld
Zoetermeer

Inhoudsopgave

1.Inleiding 8

2.Organisatie 9

3.Opdracht 10

3.1 Algemeen 10

3.2 HP Loadrunner 10

3.3 Startsituatie 11

3.4 Requirements 12

3.4.1 Functioneel 12

3.4.2 Niet functioneel 12

3.4.3 Prioriteiten 13

4.Plan van Aanpak 14

4.1 Werkwijze 14

4.2 Talen & Technieken 14

4.3 Planning 15

4.4 Risico's & Complexiteit 17

5.Sprint 0: Prototype 18

5.1 Scripts 18

5.2 De data 19

5.2.1 Regular Expressions 19

5.3 Gebruik van de Loadrunner Engine 19

5.4 Het ontwerp 20

5.5 Evaluatie 21

6.Sprint 1: Data inladen 22

6.1 Hergebruik prototype 22

6.2 Opbouw van een logfile 22
6.3 Scheduling 23
6.4 Configuratie 24
6.5 Toevoegingen ontwerp 25
6.6 Multiplatform 25
6.7 Vergelijking 26
6.8 Evaluatie 26
7.Sprint 2: Data versturen 27
7.1 JSON 27
7.2 Data versturen 27
7.3 Ontwerp 28
7.4 Onverwachte mijlpaal 29
7.5 Evaluatie 30
8.Sprint 3: Unit Testing 31
8.1 Waarom unit tests? 31
8.2 Implementatie 31
8.3 Het testplan 32
8.4 De tests bouwen 32
8.5 Performance test 33
8.6 Evaluatie 33
9.Sprint 4: Opslag gegevens 34
9.1 SQLite 34
9.2 De Database 34
9.3 Integriteit & Concurrency 35
9.4 Ontwerp 35
9.5 Evaluatie 37

10.Sprint 5: Multithreading & Aansturing 38
10.1 Experiment 38
10.2 Ontwerp 38
10.3 Aansturing 39
10.4 Evaluatie 40
11.Evaluatie Procesgang 41
12.Evaluatie Producten 42
13.Verantwoording Competenties 43
Literatuur 44
Bijlagen 45
Begrippenlijst 45
Klassendiagram 46
Deployment Diagram & Sequence Diagram 47
Testplan Unit testing 51
Config 51
DataLoaderWin32 51
ExpressionParser 52
LogfileParser 52
Result 52
Schedule 52
Transaction 53
Script 54
ScriptManager 54
Functioneel Testplan 55
Configuratie 55
Executie 55

Resultaten 56

Input 56

Output 56

1.Inleiding

Dit verslag geeft inzicht in het proces dat gevolgd wordt tijdens het afstudeertraject met als doel te laten zien dat ik het HBO denkniveau bezit en dit kan toepassen op een zelfstandig project.
Als eerste wordt beschreven hoe de organisatie in elkaar zit. Ten tweede wat de opdracht inhoudt en wat de wensen zijn voor het eindresultaat. En tot slot wat er op dit moment al beschikbaar is dat hier aan kan bijdragen.
Per ontwikkelfase is er een hoofdstuk dat beschrijft wat de verwachtingen zijn en welke oplossingen beschikbaar zijn om aan deze verwachtingen te kunnen voldoen. Daarnaast wordt er beschreven welke keuzes tussen de oplossingen gemaakt zijn en wat de motivatie daarvoor is.

De eerste fase is de introductie. Deze zal gaan over de eerste week van het project. In deze fase worden de requirements opgesteld en wordt er kennis gemaakt met de benodigde tools en software. Vervolgens wordt het prototype van het product beschreven wat een beeld schept van het eindresultaat om op die manier een start te maken met het ontwikkelen.

In het hoofdstuk over de eerste sprint gaat het over het uitwerken van het prototype. De tweede sprint zal gaan over het versturen van data die verkregen is van het deel uit de eerste sprint.
In de derde sprint zal er gewerkt worden aan een tussentijdse oplevering van het product voor de 'Overheid en ICT beurs' waar Computest een demonstratie zal geven over de software die zij kunnen bieden.
De vierde sprint zal gaan over de tijdelijke opslag van data wat een rol zal spelen bij uitval van systemen of verbroken verbindingen.
De vijfde sprint beschrijft het gebruik van multi-threading en wat voor invloed dat heeft op de bestaande architectuur.

Bij elke fase wordt er inzicht gegeven over het tot stand komen van het ontwerp met motivatie van de keuzes van de verschillende componenten.
Zowel bij het beschrijven van de werkzaamheden als bij het ontwerpen zijn er afbeeldingen die de tekst begeleiden.

Tot slot zal er een evaluatie worden gegeven over de procesgang van het project. Er zal gekeken worden hoe de aanpak door middel van methodes en technieken invloed heeft gehad op het eindresultaat. Ook de producten zullen geëvalueerd worden.

2.Organisatie

Computest is opgericht in 2002 en bestaat uit een team van ongeveer 35 testconsultants en developers.
Het bedrijf is gespecialiseerd in het aanbieden van testdiensten op het gebied van functioneel, load- en performancetesten van (internet) applicaties in complexe omgevingen. Het bedrijf richt kwaliteitssystemen in, test software in alle aspecten en stadia, introduceert inspectie- en test methoden en helpt bij het verbeteren van de ontwikkelprocessen bij grote klanten zoals ING en de Rabobank.

De consultants krijgen opdrachten toegewezen die binnenkomen vanaf klanten en zullen veelal op locatie van de klant aanwezig zijn voor een vastgestelde periode.
De developers zijn echter aanwezig zijn bij Computest zelf om te werken aan de visie om eigen software te ontwikkelen om als extra service aan te bieden aan bestaande klanten, of op deze manier nieuwe klanten te werven die specifiek interesse hebben voor de software van Computest.

Binnen Computest is er een informele sfeer. Men kleedt zich voornamelijk casual met uitzondering van belangrijke gesprekken met klanten of bijvoorbeeld intake gesprekken. Ook de omgang tussen de werknemers is informeel en bestaat er op dat gebied geen onderscheid vanwege de functie die iemand heeft.
Deze sfeer wordt versterkt door maandelijks met het hele bedrijf deel te nemen aan activiteiten. Dit is voornamelijk ook omdat de meeste werknemers vaak fulltime bij klanten aanwezig zijn en niet op het kantoor van Computest zijn. Naast deze maandelijkse bijeenkomst maakt men ook gebruik van moderne middelen zoals e-mail en chat om dagelijks op de hoogte te blijven samen met collega’s.

3.Opdracht

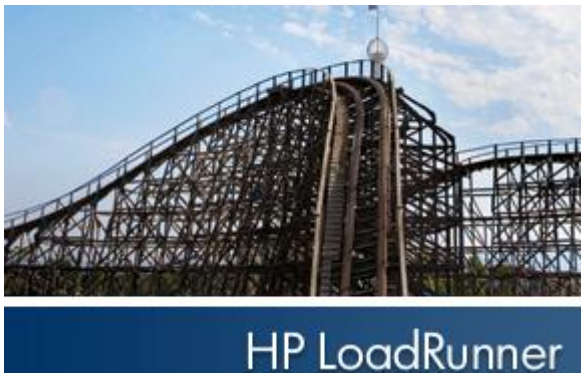
3.1 Algemeen

Dit project betreft de webapplicatie Propacity van Computest. Het wordt een monitoring tool die (indirect) de performance meet van een website en rapporteert op een interactieve manier.
De gebruiker zal grafieken en diagrammen kunnen opstellen die de performance van de website van de klant weergeeft over een ingestelde periode van tijd. Verder zullen er 'alerts' kunnen worden ingesteld die een melding kan versturen indien de performance drastisch verschilt met relatief normale waarden. Deze alerts houden in dat er een melding gestuurd kan worden via bijvoorbeeld e-mail of sms.

3.2 HP Loadrunner

Achter de schermen van Propacity zal er een zogenaamde 'agent' draaien die tijdens dit project wordt ontwikkeld. Deze agent maakt gebruik van de applicatie Loadrunner HP de marktleider op het gebied van testsoftware.
Loadrunner is een pakket wat gebruikt wordt voor het performancetesten en loadtesten van internet applicaties. Deze tests bestaan uit een script(die te schrijven zijn met de taal C) met handelingen die een gebruiker simuleren. Een script kan worden geschreven met behulp van de Virtual User Generator(VUGEN) wat onderdeel is van het Loadrunner pakket.

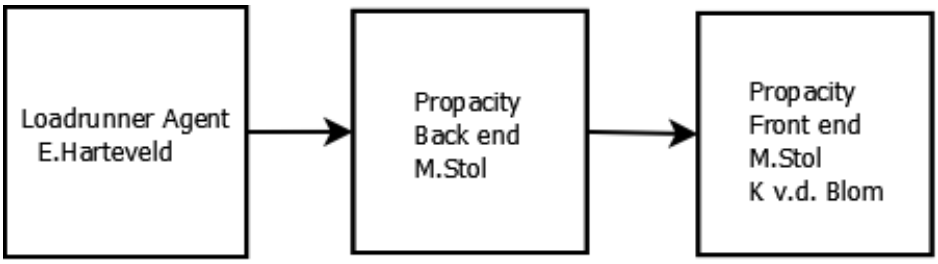
Binnen VUGEN kan dit script vervolgens worden uitgevoerd en zal dit output opleveren op het scherm betreft de responstijden van de handelingen. Verder wordt deze output weggeschreven in een tekstbestand per script. Aan de hand van deze responstijden is te zien hoe snel de onderdelen van een webapplicatie functioneren.



3.3 Startsituatie

Bij aanvang van het afstuderen is Propacity al in ontwikkeling. Het bestaat uit een fundament waar verder op gebouwd gaat worden door zowel mijzelf als de opdrachtgever.
Op dit moment is er één ontwikkelaar bezig met het project (de opdrachtgever), waar vervolgens ikzelf en een andere (tijdelijke) ontwikkelaar aan toegevoegd zijn.
Naast de Loadrunner Agent is er de zichtbare front end (look and feel van de webapplicatie). Dit zijn webpagina's die dynamisch zullen worden aangepast en zullen bewerken te zijn naar de wens van de gebruiker. Daar achter is de back end aanwezig. Deze regelt de data die dynamisch op het scherm te zien is en bevat een database met deze data.

Binnen Propacity zijn er drie delen te onderscheiden, de front end, de back end en de agent.
Op de onderstaande afbeelding is te zien wat de rolverdeling is binnen het Propacity projectteam.



De verantwoordelijkheden binnen het Propacity project.

3.4 Requirements

Mijn opdracht is gericht op de ontwikkeling van een Loadrunner agent. De afbakening zal op deze applicatie gericht zijn en zal er niet gewerkt worden aan de Propacity front end of back end.
Voor deze opdracht zijn er een aantal requirements opgesteld door de opdrachtgever.

3.4.1 Functioneel

De functionele requirements stellen dat de agent Loadrunner scripts kan uitvoeren, hun uitvoer kan inlezen en deze door kan sturen naar Propacity via het internet.
Het uitvoeren van deze scripts dient aangestuurd te worden door configuratie door middel van scheduling (bijvoorbeeld een starttijd en een wachttijd) en een configuratiebestand met informatie over het systeem en de Propacity server.

De uitvoer die deze scripts opleveren, dient uitgelezen worden en gefilterd te worden op de informatie die van belang is. Dit betreft de duurtijden van scripts en of deze fouten hebben opgeleverd.
Vervolgens zullen deze op het lokale systeem opgeslagen worden voordat deze worden verstuurd. Dit is voornamelijk bedoeld om data in de wacht te zetten voor verzending mocht de server niet bereikbaar zijn. De agent dient deze metingen uit de opslag te halen en door te sturen naar de server door middel van een beveiligde verbinding, een autorisatiesleutel en moet het mogelijk zijn om een proxy server te gebruiken. De server ontvangt data enkel in een bepaalde opmaak dat JSON heet en daarom zal de agent de data volgens deze opmaak opbouwen voordat deze het verstuurt.

Wat betreft het gebruik van de applicatie zal er zelden sprake zijn van menselijke handelingen en ligt de aandacht voor gebruikers bij de Propacity front-end.
De handelingen bij de agent zullen slechts het starten van de applicatie zijn en deze configureren voor het systeem waar deze op draait. Het afsluiten en opnieuw starten van de applicatie zal ook zelden voorkomen aangezien de agent bedoeld is om te draaien over een lange periode van tijd. Verder zal de gebruiker enkel een terugkoppeling kunnen verwachten van de agent als er problemen ontstaan. Bijvoorbeeld als Propacity niet bereikt kan worden.

Het proces wat data zal opleveren d.m.v. een script en deze zal verzenden naar Propacity.

3.4.2 Niet functioneel

De niet functionele requirements stellen dat de agent bestand is tegen een herstart van de agent zelf of het systeem waar deze op draait. Het dient een lange tijd(maanden, jaren) te draaien zonder dat het gebruik van geheugen, schijfruimte en processorkracht na verloop van tijd stijgt. Ook hoort de agent bestand te zijn tegen andere onverwachte omstandigheden zoals scripts met fouten of scripts die blijven hangen tijdens de uitvoer. Een andere belangrijke eis is dat de agent zo snel mogelijk moet kunnen draaien en een zo’n klein mogelijke impact op de performance heeft op het systeem waar het op draait.
Tot slot zal het ontwerp rekening moeten houden met uitbreiding naar andere platformen zoals Linux.

3.4.3 Prioriteiten

Binnen het project zal er ook een verdeling van prioriteiten zijn. Dit is gebaseerd op van wat er nodig is om een werkend product in te kunnen zetten bij klanten. Tijdens het project is het mogelijk dat deze prioriteiten verschuiven. De verdeling ziet er als volgt uit:

Must have:

- Script uitvoeren
- Resultaten uitlezen
- Resultaten opslaan indien er geen verbinding gemaakt kan worden
- Resultaten versturen
- Weerstand tegen onverwachte situaties

Should have:

- Scheduling
- Configuratie
- Proxy ondersteuning
- Stabiel gebruik van geheugen en processor

Could have:

- Authentication
- Scripts en schedules ontvangen van Propacity

4.Plan van Aanpak

4.1 Werkwijze

Het doel van het plan van aanpak is om de werkzaamheden van de opdracht te verdelen over de beschikbare tijd en vast te leggen welke technieken er gebruikt zullen worden.

Als eerste is er een ontwikkelmethode gekozen. Ik heb gekozen voor Scrum, omdat het korte sprints (2 a 3 weken) heeft waardoor de prioriteiten en specificaties regelmatig bijgesteld kunnen worden en er per sprint een werkend product aanwezig is.

Ook speelt de grootte van het Propacity team een rol. Omdat dit slechts drie man betreft dat met elkaar samenwerkt passen de korte iteraties en overlegmomenten van Scrum hier goed bij. Aangezien de aparte onderdelen van Propacity van elkaar afhankelijk zullen zijn, zijn deze overlegmomenten erg belangrijk en past de transparantie die Scrum biedt daar goed bij.

Een alternatieve methode had RUP kunnen zijn, maar bij deze methode zijn iteraties vaak langer dan bij Scrum. Dit zou als gevolg kunnen hebben dat het deelproject niet meer goed samen loopt met de rest van Propacity en dat er minder flexibiliteit zou zijn betreft de functionaliteiten.

Aan de hand van de eerder beschreven requirements die de opdrachtgever heeft aangeleverd, heb ik de afbakening bijgesteld. Deze betrof in het originele afstudeerplan Propacity in het geheel en de agent, wat nu enkel de agent zal betreffen. Dit zal betekenen dat er gewerkt zal worden aan een nieuw deelproject en niet aan de Propacity front end of de back end.
Ik heb gekozen voor sprints van twee weken met één keer per week een overleg over de voortgang en de planning en zal per sprint een backlog opgesteld worden met werkzaamheden.

4.2 Talen & Technieken

In het oorspronkelijke afstudeerplan is er beschreven dat er gebruikt gemaakt zou worden van de talen en technieken C++, PHP, MongoDB, Smarty, AJAX, Javascript, JSON, jQuery en Subversion.
Voor dit project zal er een slechts een deelselectie nodig zijn om de applicatie te ontwikkelen. Deze selectie bestaat uit C++, JSON en Subversion. De niet gekozen opties horen uitsluitend bij het ontwikkelen van een webapplicatie en dit valt niet binnen de afbakening.
Verder beschrijft het plan dat de applicatie onder Linux werkzaam dient te zijn, de requirements stellen echter dat er slechts rekening mee gehouden moet worden voor in de toekomst.

Voor het ontwikkelen van de agent heb ik gekozen om gebruik te maken van C++ binnen Visual Studio 2010 Express. Een alternatief had Java kunnen zijn, maar deze taal heeft (naar ervaring) het nadeel meer afhankelijkheden van andere software te hebben dan C++ en is mogelijk minder snel. De snelheid speelt een rol bij de te bouwen applicatie aangezien het een requirement is dat het zo snel mogelijk moet kunnen worden gedraaid. Vandaar dat C++ hier de beste keuze is.
De keuze voor Visual Studio Express heb ik genomen, omdat deze tool het meest gebruikt wordt voor ontwikkelen met C++ en de meeste documentatie en oplossingen voor problemen heeft.

Voor het maken van ontwerpen zal er UML aangehouden worden vanwege het feit dat de kennis en ervaring beschikbaar is.
Voor het tekenen van klassendiagrammen en use cases zal Dia gebruikt worden. Het maken van sequence diagrammen in Dia is niet praktisch genoeg, omdat het teveel tijd kost om deze tekenen en bij te werken. De tool die daarvoor gebruikt zal worden is Quick Sequence Diagram Editor.

4.3 Planning

Voor de planning van de agent heb ik besloten om een werkende versie in april op te leveren met een deel van de requirements die de basis vormen. Dit met oog op de 'Overheid en ICT' beurs waar Computest aanwezig zal zijn met een demonstratie van Propacity. Computest wil op deze beurs graag laten zien dat zij een product kunnen aanbieden wat goedkoper en beter is dan bestaande software en wil op deze manier klanten werven voor Propacity.
Dit zal voor de planning betekenen dat er tot die tijd gewerkt zal worden aan de benodigde requirements van de demo en daarom geldt er voor de sprint tot aan april dat er gewerkt zal worden aan het uitvoeren van scripts, het verzamelen en het versturen van data en te testen door middel van unit tests. Op die manier zal de agent naar wens functioneren, maar zal dan nog niet bestand zijn tegen onverwachte situaties en zal er bij uitval altijd sprake zijn van verlies van data die op dat moment niet verstuurd is.

Indien deze beurs niet aanwezig zou zijn, zou de planning zich eerst gericht hebben op het lineair bouwen van de requirements met de uitvoer van scripts, gevolgd door opslag van gegevens en daarna het sturen van data.
Na deze demo zal er gewerkt worden aan het opslaan van de data op het systeem met de mogelijkheid om deze later te versturen naar Propacity, gevolgd door het implementeren van multithreading.
Tijdens de periode vanaf de oplevering tot het einde van het project zullen er unit tests bijgewerkt en eventueel aangemaakt worden en zal er ook een functionele test uitgevoerd worden.

Week	**Data**	**Activiteit**
1 t/m 2	7-02 t/m 18-02	Prototype
3 t/m 4	21-02 t/m 04-03	Sprint 1
5	07-03 t/m 11-03	Sprint 2
6	14-03 t/m 18-03	Vakantie
7	21-03 t/m 25-03	Sprint 2
8 t/m 9	28-03 t/m 08-04	Sprint 3 Oplevering demo
10 t/m 11	11-04 t/m 22-04	Sprint 4
12 t/m 13	25-04 t/m 06-05	Sprint 5

Sprint 1 (2 weken)
- Configuration, het inladen van een tekstbestand met configuratie.
- Execution, het uitwerken van de uitvoer van scripts.
- Result Gathering, de vereiste metingen(vastgesteld in de requirements) in een soortgelijke structuur opslaan binnen de applicatie.

Sprint 2 (2 weken)
- Result Transmission.
- Het versturen van metingen naar de webservice door middel van een externe library.
- Configuration uitbreiden voor transmission.

Sprint 3 (2 weken)
- Input en Transmission optimaliseren voor de demo.
- Testrapporten opstellen voor de demo.
- Unit testing.

Sprint 4 (2 weken)
- Queueing
- Lokale opslag voor metingen
- Uitlezen van metingen uit opslag

Sprint 5 (2 weken)
- Multithreading
- Data ontvangen
- Scheduling

4.4 Risico's & Complexiteit

Tijdens het project zullen er een aantal risico's aanwezig zijn. Als eerste is er het aansturen van Loadrunner. De uitdaging hier is om een nieuwe manier te verzinnen om Loadrunner te gebruiken en mogelijk kunnen hier obstakels zich voordoen waardoor dit niet voldoet aan de eisen.

Bij het uitvoeren van een script is de eis dat daar controle over is door middel van configuratie en scheduling. Het risico is dat Loadrunner zich niet in de verwachtte mate laat instellen.
De uitvoer die scripts opleveren zal uitgelezen en gefilterd moeten worden op de informatie die van belang is voor Propacity. Het gevaar daarbij is dat de juiste informatie in sommige situaties niet gevonden kan worden of in gebreke.
De moeilijkheid daarbij is dat er gekeken moet worden op welke mogelijke manieren de uitvoer weggeschreven kan worden en hoe dit per situatie ingelezen kan worden.

Bij het tijdelijk opslaan van gegevens is er sprake van een langer traject dat de data zal volgen en is er het risico dat de integriteit wordt aangetast. De complexiteit hierbij is een manier te ontwikkelen om deze integriteit te waarborgen.
Om de gegevens vervolgens te versturen, zal er manier bedacht moeten worden om dit te kunnen doen. Dit zal veel tijd kosten indien dit zelf gebouwd moet worden in plaats van het gebruik van een library.

Tijdens het ontwerpen zal er gedacht moeten worden hoe elke functionaliteit per platform in het ontwerp past en zal er het risico zijn dat er in de toekomst zal blijken dat een functionaliteit niet volgens hetzelfde principe werkt als dat van een ander platform en het bestaande ontwerp aangepast moet worden.

De zware requirements zijn naar inschatting het inlezen van de data en deze op te slaan met integriteit, omdat het gebruiken van Loadrunner op deze manier niet gebruikelijk is en op maat gemaakt wordt. Het opslaan van de data zal veel aandacht vereisen, omdat de integriteit van de data cruciaal is voor Propacity aangezien de Loadrunner Agent het startpunt van het proces is.
De minder zware requirements zullen het versturen van de data zijn aangezien er al bekend is hoe de data er uit moet zien om verstuurd te worden en er binnen het team de kennis is over hoe Propacity in zijn geheel werkt. Deze kan echter wel zwaarder worden als blijkt dat een verbinding maken veel tijd zal kosten om te ontwikkelen.

Het waarborgen van de snelheid en performance zal ook niet zwaar zijn, omdat er goede software beschikbaar is om tijdens het ontwikkelen een verlies in performance te kunnen opsporen. Wat betreft de multiplatform requirement is het goed mogelijk dat er gebruik gemaakt zal worden van bestaande libraries. Deze beschikken meestal al de mogelijkheid om op andere platforms gebruikt te worden waardoor dit geen invloed zal hebben op het eigen ontwerp.

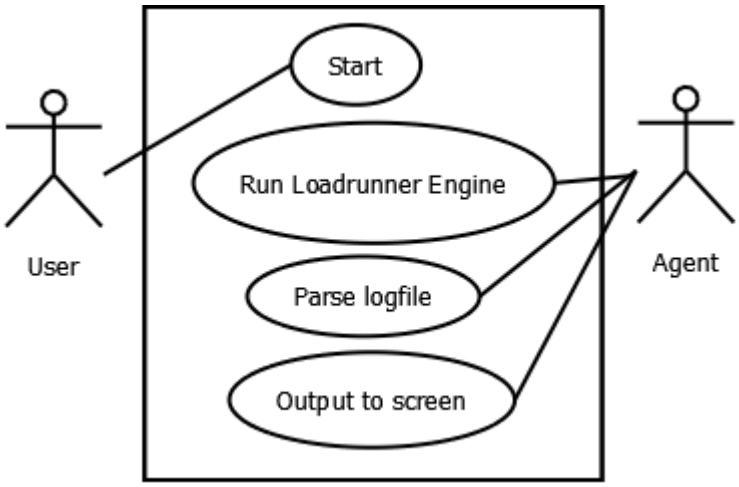
5.Sprint 0: Prototype

Om mijzelf bekend te maken met de werking van Loadrunner en de data die deze oplevert heb ik besloten een prototype te bouwen. Dit omdat zonder deze kennis niet een goed ontwerp gemaakt kan worden voor de daadwerkelijke applicatie.
De planning stelt dat er tot april gewerkt zal worden aan het inladen van gegevens en doorsturen hiervan. Aangezien er gegevens niet doorgestuurd kunnen worden als deze er niet zijn, is de meest logische keuze om te werken aan het uitvoeren van scripts en het inlezen van data.

5.1 Scripts

Om de gewenste gegevens te verkrijgen, wordt er gekeken naar hoe een Loadrunner script werkt. Het bevat een verzameling van bestanden waarin één bestand met het script zelf centraal staat. In dit bestand worden aanroepen beschreven op webpagina's en wordt de responstijd gemeten. Deze aanroepen en de resulterende metingen worden vervolgens in een tekstbestand opgeslagen als een 'transaction' per meting. Dit bestand wordt ook wel een logfile genoemd.
De belangrijke delen van een script voor de agent zijn het scriptbestand en de logfile. De resterende bestanden zullen verder niet genoemd worden.

Het doel is om deze metingen uit de logfile te kunnen lezen, maar voordat dit gedaan kan worden wordt er gekeken hoe dit script in de eerste plaats uitgevoerd kan worden aangezien één van de eisen is dat de agent dit zal gaan verzorgen.
Loadrunner blijkt een apart component te gebruiken wat een executable bestand is. Dit bestand wordt ook wel de Loadrunner Engine genoemd en wordt aangestuurd door het meegeven van parameters vanuit VUGEN (de tool waarmee scripts gebouwd en bewerkt kunnen worden).
Het is mogelijk om dit component te gebruiken zonder dat er VUGEN aan de pas komt. Daarom zal deze Loadrunner Engine aangeroepen worden vanuit de agent.



De use case van het prototype.
Er wordt een script uitgevoerd en de resultaten worden op het scherm getoond.

5.2 De data

Om te beginnen heb ik gekeken hoe een logfile ingelezen kan worden en deze doorzocht kan worden voor de metingen, voor het inladen en zoeken heb ik C++ functionaliteiten gebruikt. Voor het inladen werken deze goed (hoewel er uiteindelijk wel rekening gehouden zal moeten worden met multiplatform), maar voor het zoeken lijkt dit niet geschikt te zijn. Dit omdat er met de gebruikte functionaliteit slechts gezocht kan worden naar specifieke regels tekst. Echter de regels die deze metingen bevatten kunnen verschillen vanwege instellingen in het script en zal de zoekcriteria niet meer overeen komen met de tekst.
Daarom is er een manier nodig om bepaalde patronen te herkennen in een regel tekst. Daarvoor wordt er gekeken naar regular expressions.

5.2.1 Regular Expressions

Een regular expression is een manier om een patroon binnen een stuk tekst te herkennen. Het bestaat uit een syntax waarmee een patroon beschreven kan worden en sleutelwoorden waar op gezocht kan worden.
Om dit te gebruiken is er gezocht naar oplossingen van derde partijen en is de keuze gevallen op Boost, een verzameling van functionaliteiten voor C++. Dit omdat Boost bekend is, multiplatform is en uitgebreide documentatie heeft. Verder bleek dat het onderdeel voor regular expressions van Boost al was geïntegreerd binnen C++. Daarom heb ik vanwege de beschikbaarheid hiervan gekozen voor deze oplossing.

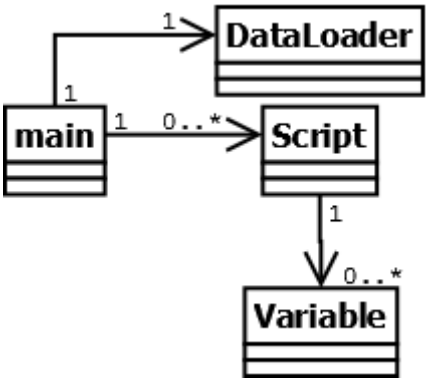
5.3 Gebruik van de Loadrunner Engine

Vervolgens is er een manier nodig om de Loadrunner Engine aan te kunnen roepen met parameters. Om dit te kunnen doen zijn er standaard functionaliteiten binnen C++ die dit mogelijk maken. Er kan een programma opgestart worden met de gewenste parameters en kan er controle over dit proces gehouden worden. Met de controle kan bijvoorbeeld het proces worden beëindigd als een script te lang duurt. Dit proces wordt per script uitvoer uitgevoerd en zal uit zichzelf afsluiten zodra een script is uitgevoerd.

5.4 Het ontwerp

Voor het ontwerpen van dit prototype is er nagedacht over de verantwoordelijkheden van de applicatie en de data die is er is.
Als eerste heb ik nagedacht over het inladen van de gegevens. Dit heb ik samen met het gebruik van regular expressions, ondergebracht in een 'data loader'. Op deze manier zijn alle functionaliteiten die met de logfile te maken hebben ondergebracht in een enkele class.

Vervolgens is er gekeken naar de informatie over het script en de metingen. Ik heb gekozen om in een script class de naam en parameters op te nemen zodat de agent de Loadrunner Engine deze informatie kan gebruiken om het daadwerkelijke script uit te voeren. De resultaten worden na uitvoer in een 'variable' object opgeslagen en aan het bijhorende script object toegevoegd.
Met dit ontwerp is er te zien hoe de agent zich samen met de Loadrunner Engine gedraagt.

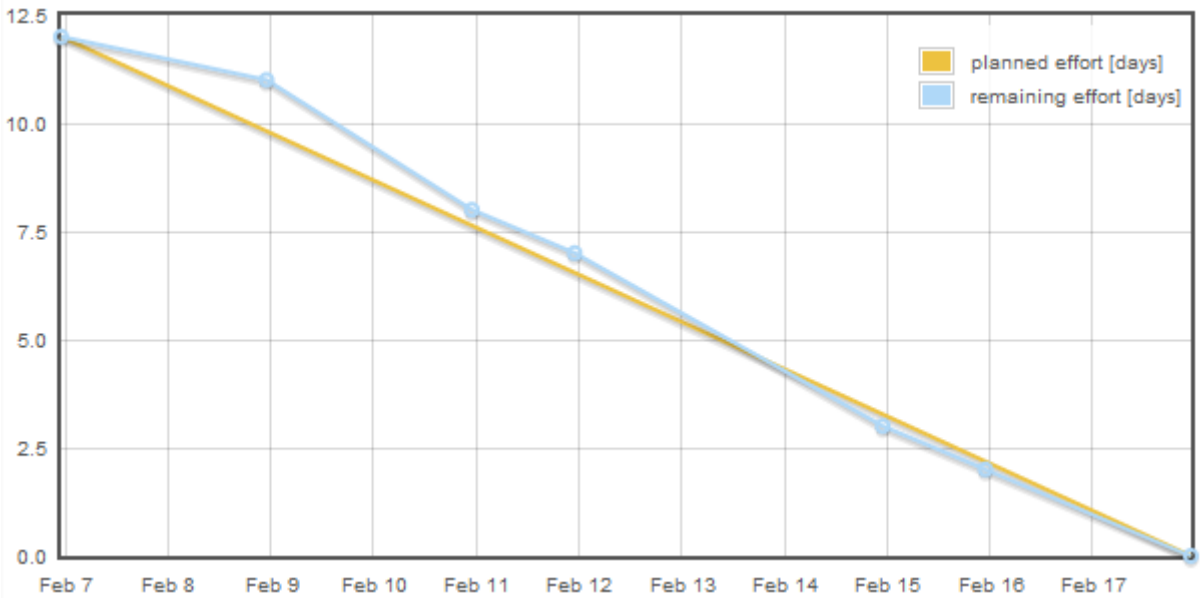


Het ontwerp voor het prototype van de Loadrunner Agent. Hier is de requirement van het inlezen van een logfile uitgewerkt in een simpele vorm.

Binnen het ontwerp van het prototype is er nog geen rekening gehouden met *portability* en daarom zal de applicatie enkel op Windows besturingssystemen werken op dit moment, omdat er een Windows specifieke functionaliteit gebruikt wordt om de Loadrunner Engine te starten. Uiteindelijk is het ook de bedoeling dat dit multiplatform kan.

5.5 Evaluatie

Aan het eind van deze sprint is er op de onderstaande burndown chart te zien dat er aan het begin weinig voortgang was, omdat de regular expressions functionaliteit binnen de applicatie mogelijk gemaakt moest worden en deze leren te gebruiken.
Daarna is de voortgang stabiel gebleven. Deze is zelfs iets sneller gegaan, omdat het opzetten van een configuratie en het aanroepen van externe applicaties minder tijd heeft gekost dan verwacht.



6.Sprint 1: Data inladen

6.1 Hergebruik prototype

Binnen deze sprint zal ik in eerste instantie kijken hoe delen van het prototype gebruikt kunnen worden om de daadwerkelijke applicatie een basis te geven.
De al ontwikkelde functionaliteiten zullen opnieuw worden ingedeeld met oog op cohesion. Dit houdt in dat bestaande classes gescheiden en ingedeeld worden met functionaliteiten die bij elkaar horen. Er wordt hier gestreefd naar 'strong cohesion'. De klassen doen enkel wat hun naam en documentatie beschrijven en zijn begrijpbaar.
Naast cohesion wordt er ook nagedacht over coupling. Dit betekent hoe afhankelijk klassen van elkaar zijn. Er zal gekozen worden voor 'loose coupling'. De klassen zullen worden ontworpen zodat deze niet teveel afhankelijk van elkaar zijn. Het scheiden van functionaliteiten is in een lichte vorm aanwezig in het prototype, maar er is geen rekening gehouden met cohesion en coupling.

6.2 Opbouw van een logfile

Als eerste kijk ik naar de opbouw van de logfile van een script en wat voor structuur deze heeft
Deze structuur bestaat uit iterations welke actions hebben en actions hebben weer transactions. Dit is een structuur met meerdere niveaus en ziet er als volgt uit:

```
Iterations
{
        Actions
        {
                Transactions
                {
                }
        }
}
```

```
Action.c(4): Found resource "http://www.google.nl/images/mgyhp_sm.png" in HTML "http://www.
Action.c(4): web_url("google.nl") was successful, 8648 body bytes, 1838 header bytes     [Ms
Action.c(12): Notify: Transaction "00_Website" ended with "Pass" status (Duration: 0,1823).
Ending action Action.    [MsgId: MMSG-15918]
```

*Een deel van de logfile van een script met een **transaction.** De informatie die van belang is, zijn de **naam** van de transaction (in dit geval "00_Website"), de **status**("Pass") en de **duration.** Een andere belangrijke waarde is "**Error**" welke niet voorkomt in dit voorbeeld.*

Er moet een keuze gemaakt worden in welke mate deze structuur van data wordt overgenomen. Door te kijken naar wat Propacity opslaat qua metingen, heb ik gekozen om enkel het laagste niveau (de transactions) over te nemen. Verder verwacht Propacity de metingen binnen te krijgen als een verzameling die gelijk staat tot een enkele scriptuitvoer.
Voor deze verzameling zal er ook enkele algemene informatie worden opgenomen, zoals de starttijd van het script, de globale duurtijd, of het script fouten heeft en of het script succesvol is uitgevoerd.

```
------------------------------------
************************************
------------------------------------
Start Result
  -Start:
  -End:
  -Duration: 182ms
  -Successful: 1

        Start Transaction 0:
          -Name: 00_Website
          -Succesful: 1
          -Duration: 182.3ms
          -Error Message:
        End Transaction 0
End Result
------------------------------------
************************************
------------------------------------
```

*Een scriptuitvoer binnen de agent als tekst gevisualiseerd. De verzameling bevat een enkele transaction met de informatie over de **naam**, of de transaction **geslaagd** is, de **duurtijd** en een eventuele **foutmelding**.*

*Voor de verzameling in zijn geheel is er ook een **duurtijd** opgenomen en of alle transactions gezamenlijk **geslaagd** zijn.*

6.3 Scheduling

Vervolgens wordt er een basis gemaakt voor *scheduling*. Dit houdt in dat scripts op vooraf ingestelde tijden uitgevoerd worden. De requirements stellen dat dit schedule een start datum, een starttijd, een duurtijd en een wachttijd hoort te bevatten.

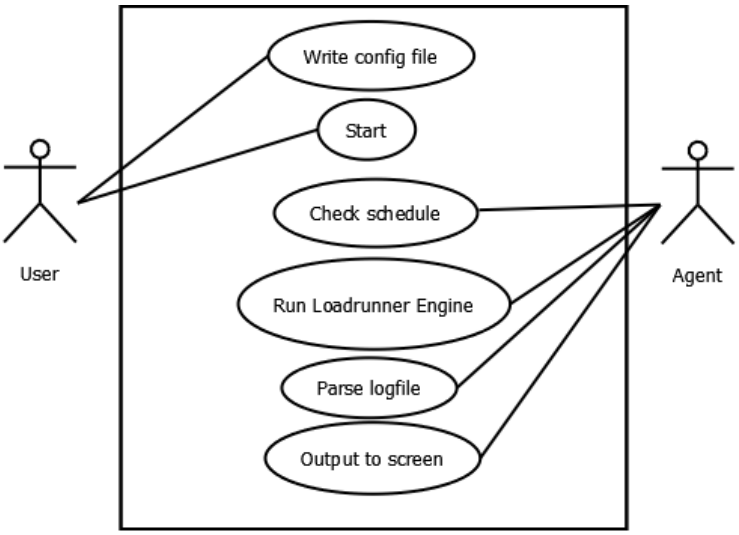
Binnen deze sprint heb ik gekozen om slechts een deel van de requirements voor het schedule te implementeren. Dit omdat de (zoals eerder beschreven) functionaliteiten slechts in een basisvorm zouden worden uitgewerkt voor de demonstratie in april.
Uiteindelijk is de bedoeling dat een schedule wordt opgevraagd via Propacity en deze aan een script gekoppeld wordt.

Er is een klasse gemaakt wat alle informatie betreft het schedule bevat. Het bestaat uit een ‘interval’ wat het aantal seconden wachttijd tussen de uitvoer aangeeft. En er wordt een waarde bijgehouden wanneer het laatste uitvoer moment was. Op deze manier kan er berekend worden of er al een nieuwe uitvoer plaats mag vinden of niet.
Verder is er de mogelijkheid om een starttijd op te geven, waarna deze wordt vergeleken met de huidige tijd. Bijhorende waarde is een duurtijd. De starttijd plus de duurtijd is de tijdsperiode waarin het script mag uitgevoerd worden.

6.4 Configuratie

Tot slot wordt er in deze sprint aandacht besteed aan de configuratie van de agent. Dit houdt in dat er een tekstbestand wordt opgesteld met informatie van het systeem waarop de agent draait. Deze informatie bevat onder andere locaties van de installatie van Loadrunner en van de scripts die uitgevoerd gaan worden. Het tekstbestand wordt ingeladen in de agent en de gegevens worden gezocht met regular expressions.

De keuze voor deze manier van configuratie heb ik gemaakt, omdat er op deze manier makkelijker systeem specifieke veranderingen kunnen worden doorgegeven aan de agent. Indien dit niet was geïmplementeerd zou de agent intern moeten worden aangepast per systeem en opnieuw worden gecompileerd en geïmplementeerd. Dit zou nou niet aansluiten met de requirement dat de agent met geen tot weinig onderhoud operationeel kan blijven, omdat er sprake zou zijn van een onderbreking in de stroom van data.

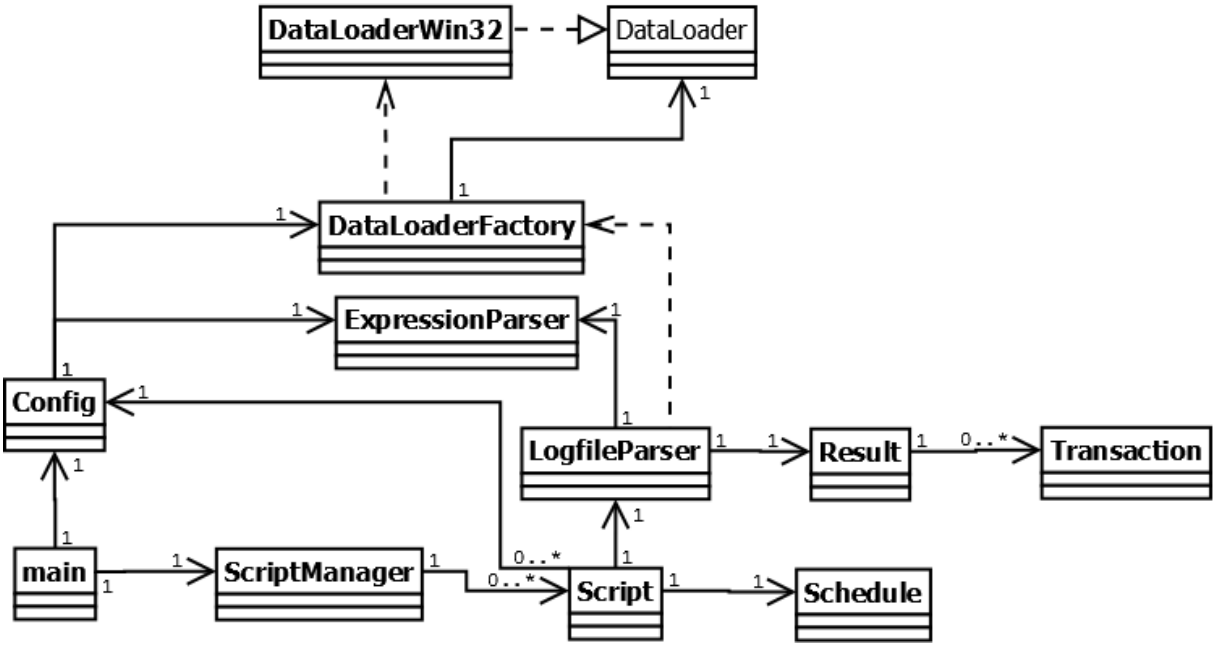


De use case laat zien dat de configuratie wordt ingesteld en het schedule wordt gecontroleerd.

6.5 Toevoegingen ontwerp

Aan het einde van deze sprint zijn er een aantal toevoegingen te zien binnen het ontwerp en vormen de basis van de Loadrunner Agent.

- Een structuur voor de ingeladen data bestaande uit een Result met Transaction objecten.
- Splitsing van de functionaliteiten van het inladen van bestanden, het uitvoeren van scripts en het gebruik van regular expressions naar aparte onderdelen.
- Een basis voor de Schedule klasse.
- Config klasse die de configuratie gegevens beheert.



Op dit diagram is te zien hoe het ontwerp er op het moment uit ziet.
Alle scripts worden beheert in een script manager die elk script uitvoert aan de hand van het schedule wat een script bevat. Op deze manier blijven deze op een centraal punt.

6.6 Multiplatform

Na uitvoer zal de logfile parser het bij het script horende logbestand inlezen met de eerder beschreven data loader.
In het diagram wordt deze klasse aangegeven als een Factory. Een design pattern dat gebruikt is om deze klasse multiplatform te gebruiken in de toekomst. Als de applicatie data wil gaan inlezen zal deze een dataloader object aanvragen aan deze Factory, die zal vervolgens kijken onder welk besturingssysteem de applicatie op dat moment draait en aan de hand daarvan een type dataloader meegeven dat code bevat bedoeld voor het huidige besturingssysteem.

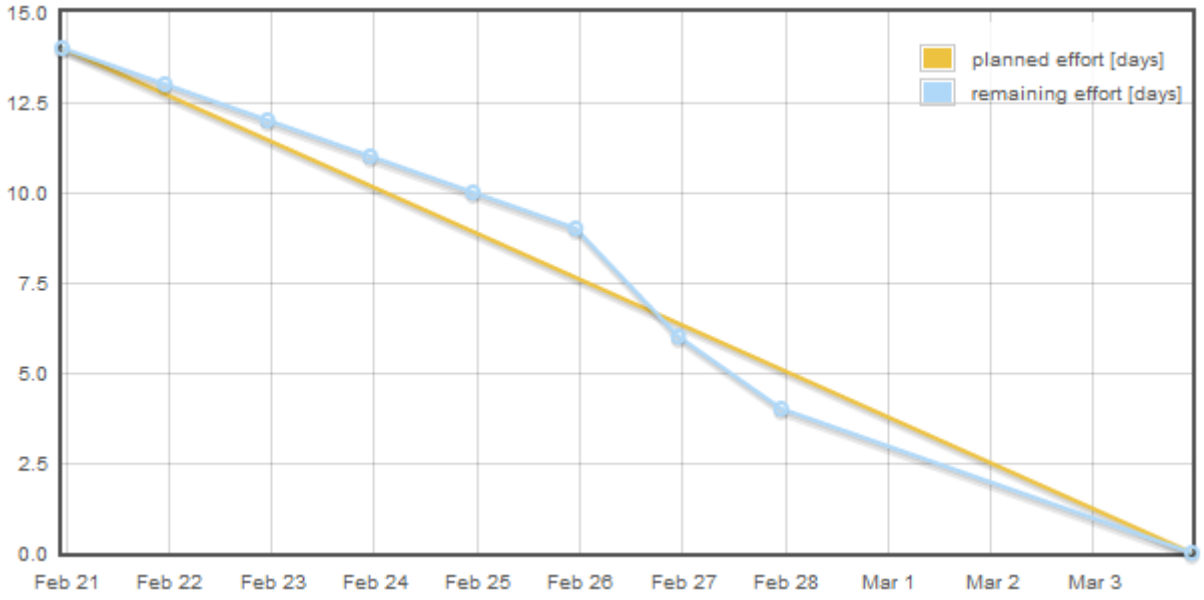
Vervolgens wordt de expression parser klasse gebruikt om variabelen te zoeken in de data en daarna wordt de structuur van de logfile opgeslagen in een Result object met Transaction objecten. Het Result komt tot slot terug bij de script manager aan.

6.7 Vergelijking

Na het hergebruik van het prototype is er te zien dat er een aantal verschillen zijn ten opzichte van het huidige ontwerp en dat van het prototype. De dataloader is nu alleen nog maar verantwoordelijk van het lezen van tekstbestanden en de functionaliteit voor regular expressions is een aparte class geworden. Het doorzoeken van een logfile gebeurt nu in de logfile parser die gebruik maakt van vorige twee genoemde classes.
In het prototype werden variabelen met hun waarde, regular expression en uitkomst binnen één class bewaard. Dit is nu uitgesplitst naar een result met transaction en bevinden de regular expressions zich in de logfile parser welke een logfile omzet naar result.
Elk script bevindt zich nu in de scriptmanager class welke bepaald wanneer een script uitgevoerd mag worden. Dit in tegenstelling tot het prototype waar geen enkele controle was over de uitvoer van scripts.

6.8 Evaluatie

Aan het einde van de sprint is te zien dat er in de eerste week iets minder voortgang is geboekt tijdens het ontwerpen en herstructureren van de functionaliteiten van het prototype. Vervolgens is deze achterstand weer ingehaald vanwege het feit dat het ontwikkelen minder tijd heeft gekost dan verwacht, omdat ik over het ontwerp goed heb nagedacht en geen noemenswaardige problemen ben tegengekomen daardoor.



7.Sprint 2: Data versturen

In deze sprint gaat de meeste aandacht naar het oversturen van data naar de Propacity webservice. Dit is de tweede stap voor het bouwen van functionaliteiten die nodig zijn voor de demo in april.

7.1 JSON

Om de metingen door te sturen naar Propacity zullen deze omgezet worden naar een specifieke opmaak van tekst (JSON, een manier om data te representeren) zodat de gegevens ontvangen kunnen worden. Dit is overigens de enige manier waarop Propacity de metingen kan ontvangen en zal andere manieren niet accepteren. De criteria zijn dat de data omgezet kan worden naar een enkele regel tekst die zich aan het JSON formaat houdt.
Er is een afweging gemaakt tussen het zelf schrijven van een functie die de omzet naar JSON doet of een bestaande library te gebruiken. Het gebruik van een bestaande library zou betekenen dat deze eerste onderzocht en geïmplementeerd zou moeten worden. Wat mogelijk meer tijd zou kosten dan het zelf bouwen van deze functionaliteit en zou het mogelijk niet zou voldoen aan de criteria. Daarom heb ik gekozen om zelf deze functionaliteit te bouwen.

Deze functionaliteit bevindt zich in de Result klasse zelf, waardoor deze met een functie aanroep zichzelf omzet in een JSON string. Een andere manier had kunnen zijn dat er een nieuwe of bestaande klasse deze verantwoordelijkheid op zich nam, maar dit zou de cohesion niet ten goede komen.

7.2 Data versturen

Vervolgens wordt er een basis ontwikkeld voor het doorsturen van de data. Van tevoren was er al duidelijk dat dit via het HTTP/HTTPS protocol gedaan zal worden, omdat dit protocol wordt gebruikt voor communicatie via het internet en daarom ook voor Propacity.
Om de communicatie op te zetten, is er behoefte aan een functionaliteit binnen C++ die via HTTP gegevens kan sturen.
De requirements stellen dat dit via een beveiligde verbinding dient te kunnen evenals het gebruik van een proxy. Daarnaast zal dit ook multiplatform horen te werken. De afweging is hier de tijd dat het zal kosten om iets te bouwen/implementeren.
Het gebruik van standaard functionaliteiten zou betekenen dat deze mogelijk niet uitgebreid genoeg zijn en ook niet multiplatform te gebruiken zullen zijn. Dit zou een abstractie opleveren waar later voor andere platforms een aparte implementatie voor gebouwd zal moeten worden.
Een andere keuze zou zijn om een library te implementeren die de gewenste functionaliteiten bezit en multiplatform te gebruiken is. De enige library die aan deze criteria voldoet is cURLpp.

7.3 Ontwerp

Om de output functionaliteiten af te scheiden van de rest van het ontwerp zal er een onderdeel ontwikkelt worden die de verantwoordelijkheid op zich neemt van het beheren van metingen Dit zal de result manager worden genoemd. Dit onderdeel ontvangt de metingen en slaat ze op in het geheugen en heeft op deze manier een eenvoudige manier van opslag gerealiseerd met oog op de demo. Deze manier van opslag zal later in het project echter wel sterk uitgebreid en/of vervangen worden.

Vervolgens is cURLpp geïmplementeerd binnen deze result manager zodat er metingen verstuurd kunnen worden vanuit het geheugen. Om deze library aan te sturen wordt er in de configuratie gedefinieerd wat het adres en de port van de Propacity server is en of er een eventuele proxy is die gebruikt zal worden. Door cURLpp in te stellen met deze waarden en de metingen in het JSON formaat mee te geven, kan er data over worden gestuurd naar Propacity.

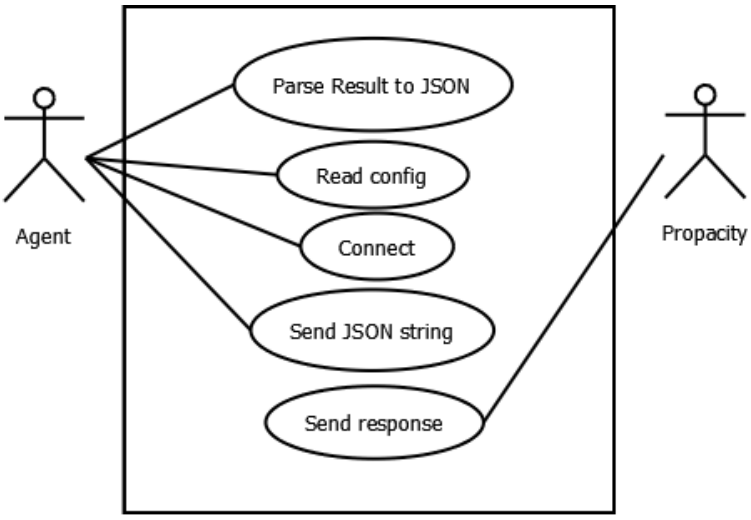
Vervolgens wordt de functionaliteit om een verbinding te maken gebouwd. Er wordt rekening gehouden met de configuratie welke aangeeft wat bijvoorbeeld het adres van de server is en op welke port er verbonden kan worden. Ook wordt er rekening gehouden met de mogelijkheid om via een proxy te verbinden, zowel als in de configuratie als tijdens het verbinden.
Uiteindelijk kan de eerder beschreven JSON data met metingen meegestuurd worden naar de Propacity webservice zodat deze gebruikt kunnen worden voor visualisaties.

```
< HTTP/1.1 100 Continue
< HTTP/1.1 200 OK
< Date: Thu, 24 Mar 2011 10:25:46 GMT
< Server: Apache/2.2.16 (Win32) PHP/5.3.3
< X-Powered-By: PHP/5.3.3
< Content-Length: 114
< Content-Type: text/html; charset=UTF-8
<
{"webservice":"Measurements\/Submit","error":false,"error_message":"No error.","
payload":{"measurements_added":7}}* Connection #0 to host 192.168.11.49 left int
act
* Closing connection #0
```

Een response van de Propacity webservice. De tekst "measurements_added" is een indicatie van een geslaagde overdracht van metingen.

De al bestaande script manager heeft een result manager object wat Result objecten kan ontvangen. Aangezien de script manager de results als eerste binnen krijgt van de scripts, stuurt deze ze door naar de result manager. Ik heb hiervoor gekozen, omdat de scripts en hun uitvoer aparte onderdelen zijn die voorkomen in verschillende momenten van het proces binnen de applicatie.
Deze bevat een queue (een lijst met elementen gesorteerd van oud naar nieuw) met Result objecten die met de originele volgorde verzonden worden naar Propacity door middel van cURLpp.

Propacity verwacht zoals eerder beschreven een JSON string. Daarom wordt de eerder beschreven functie in de Result klasse gebruikt hiervoor.



Aan de hand van de configuratie wordt er een verbinding gemaakt en worden de metingen verstuurd.

7.4 Onverwachte mijlpaal

Tijdens de sprint is er vanaf de opdrachtgever de vraag gekomen om de huidige versie van de agent te implementeren op een server met als doel een maand lang realistische metingen te genereren. Dit omdat op dit moment nog gebruik wordt gemaakt van fictieve data om Propacity te demonstreren. Daardoor is er afgeweken van de planning door een niet van tevoren geplande mijlpaal op te leveren binnen korte tijd.

Voor de implementatie hiervan wordt er een tijdelijke manier van opslag gekozen zonder enig onderzoek of ontwerp. De metingen worden in een Excel bestand bijgeschreven zodat deze later met de hand in Propacity geladen kunnen worden.

```
"computest";1299859463;"Bol_110311";1;"UC2_01_Homepage";0.6674;""
"computest";1299859463;"Bol_110311";1;"UC2_02_Inlog_Pagina";0.3348;""
"computest";1299859464;"Bol_110311";1;"UC2_03_Login";0.1841;""
"computest";1299859464;"Bol_110311";1;"UC2_04_Alles_Zoeken";0.4183;""
"computest";1299859464;"Bol_110311";1;"UC2_05_Boek_in_Winkelwagen";0.355;""
"computest";1299859465;"Bol_110311";1;"UC2_06_DVD_Zoeken";0.3743;""
                                                                  645;""
"   1          2          3      4            5          6    7   9;""
"computest";1299859466;"Bol_110311";1;"UC2_08_Verwijder_Item2";0.1189;""
"computest";1299859466;"Bol_110311";0;"UC2_09_Uitloggen";0.1864;"Error -263
"computest";1299859468;"CompuForum_110311";1;"UC3_01_Homepage";0.2142;""
"computest";1299859469;"CompuForum_110311";1;"UC3_02_Login_Pagina";0.0493;"
"computest";1299859469;"CompuForum_110311";1;"UC3_03_login";0.3745;""
```

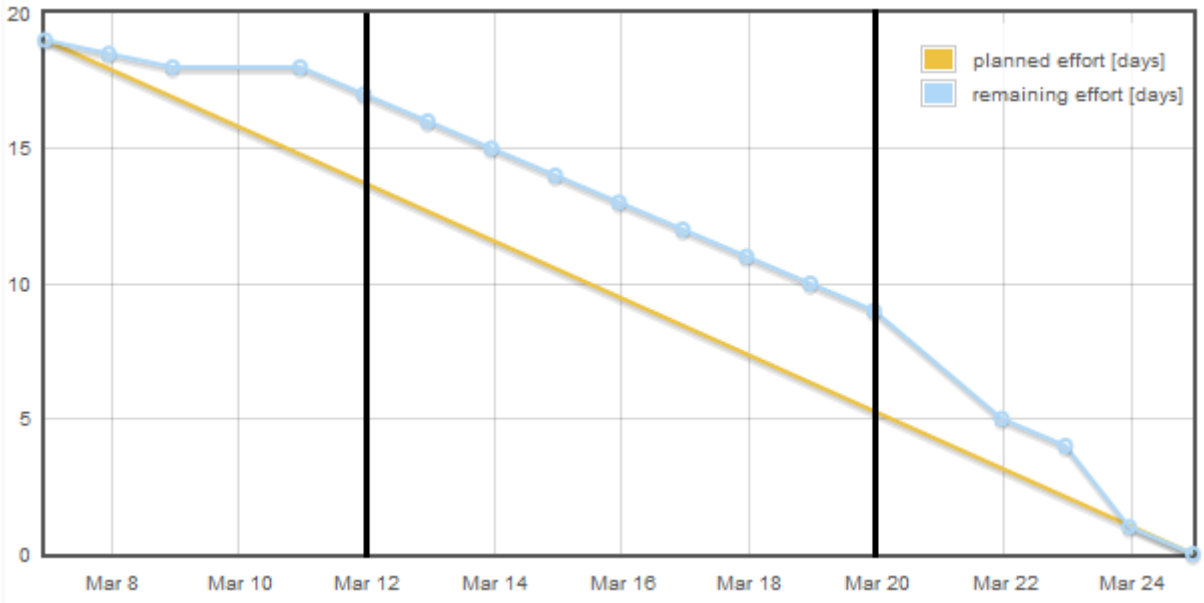
*Metingen in het Excel bestand. Er wordt per regel een transaction weggeschreven met de **klantnaam(1)**, de **timestamp (starttijd in UTC) (2)**, de **scriptnaam (3)**, of de transaction **geslaagd (4)** is, de **transactionnaam (5)** en **hoe lang (6)** de transaction heeft geduurd. Tot slot is er de **error (7)** waarde die in dit voorbeeld bij elke transaction leeg is.*

Er wordt voor het eerst gebruik gemaakt van Loadrunner scripts die Computest zelf heeft geschreven in tegenstelling tot eerdere zelfgemaakte scripts die slechts dienden om de invoer van metingen te ontwikkelen. Een probleem wat hierbij optreedt, is dat een script twee mogelijke manieren van wegschrijven heeft, namelijk een eenvoudige of een uitgebreide.
De Computest scripts zijn ingesteld om uitgebreid een logfile te genereren, waardoor de agent deze niet kan uitlezen. Om dit tijdelijk op te lossen zijn de scripts zelf aangepast en zal de Loadrunner Engine in de toekomst een extra parameter mee krijgen wat de script in de eenvoudige stand zal uitvoeren.

Achteraf blijkt dat de metingen in het Excel bestand een afwijking hebben in de UTC timestamp (de tijd waarop een meting is verricht in seconden sinds 1 januari 1970). In deze timestamp heb ik een correctie gedaan voor de tijdzone vanwege de aanname dat Propacity metingen aan nam van die timestamps bevatten met GMT+1. Echter hoort dit GMT+0 te zijn en is de correctie overbodig.
Het omzetten van metingen vanuit Excel naar Propacity vereist ook enig handwerk om tot een geschikt formaat te kunnen komen.

7.5 Evaluatie

Op de onderstaande grafiek is te zien dat er in de eerste week de voortgang moeizaam verloopt. Dit komt vanwege het implementeren van cURLpp. Dit bleek een grotere uitdaging dan verwacht, omdat deze library uit meerdere onderdelen bestaat. Vervolgens is er in het midden van de grafiek te zien dat er een periode geen voortgang is geboekt wegens een vakantie. Daarna is de voortgang sterk gestegen richting de optimale planning vanwege het feit dat cURLpp goed bruikbaar is en geen problemen vertoonde nadat deze library geïmplementeerd was. Het opbouwen van de JSON structuur met metingen ging ook sneller dan verwacht, omdat het formaat niet erg complex is.



8.Sprint 3: Unit Testing

Het doel van deze sprint is om de agent klaar te maken voor de 'Overheid & ICT beurs'. Dit houdt in dat er getest wordt of de gewenste functionaliteiten werken naar behoren.
Deze functionaliteiten zijn het uitvoeren van scripts en de resultaten direct doorsturen naar Propacity.

8.1 Waarom unit tests?

Om de code te testen heb ik gekozen om unit tests te schrijven zodat er zekerheid is over de integriteit van de data tijdens elk moment van de weg die deze volgt. De keuze voor unit tests is gemaakt, omdat er te allen tijde zekerheid moet zijn over de data en dit is niet met een andere testmethode mogelijk of zou deze meer tijd in beslag nemen.
Een unit test is het schrijven van een stuk code dat een functionaliteit test door de invoer en verwachte uitkomst van tevoren vaststelt. Wijkt deze uitkomst af, dan is de test niet geslaagd.
Op deze manier kan er constant getest worden of functionaliteiten nog voldoen aan hun verwachtingen en kan dit automatisch getest worden in plaats van een handmatige test.

8.2 Implementatie

Om dit te kunnen doen is er eerst gekeken naar de functionaliteit van Visual Studio die dit verzorgt, maar deze is uitgeschakeld onder de Express versie die gebruikt wordt voor dit project. Dit had geen probleem opgeleverd als er een andere versie van Visual Studio gebruikt zou zijn en de Express versie blijkt nu een minder goede keuze te zijn geweest dan ik dacht.
Daarom heb ik nagedacht over alternatieven. Het zelf schrijven van een unit test functionaliteit is geen optie, omdat dit een project op zichzelf zou zijn waar veel kennis over unit testing voor nodig zou zijn. Vandaar dat er uitgeweken wordt naar het gebruik van een library, de criteria hier voor zijn dat er per class tests geschreven kunnen worden die bij elke compilatie van de applicatie worden uitgevoerd. Verder dient de library multiplatform te werken.

Boost biedt ook een library binnen het pakket en hier is ook mee geëxperimenteerd. Dit bleek naar wens te werken en is uiteindelijk de keuze geworden voor unit testing. Wat ook meespeelt is dat Boost al binnen het project aanwezig is en al enige ervaring is om de verschillende libraries daarvan te implementeren.

Naast de voorgaande genoemde oplossingen zijn ook enkele kleinere alternatieven bekeken, maar deze bevatten weinig tot geen documentatie en zijn in de meeste gevallen zelf nog in ontwikkeling. Uiteindelijk heb ik gekozen voor Boost, zoals eerder beschreven is dit pakket in de eerste plaats multiplatform en biedt het unit testing aan die aan de criteria voldoen. Een bijkomend voordeel is dat er binnen de applicatie geen afhankelijkheid is bijgekomen, omdat Boost al aanwezig is.

8.3 Het testplan

Voordat de tests geschreven worden, wordt er eerst een test plan (zie bijlagen) opgesteld.
Hierin wordt beschreven per klasse wat voor data er gebruikt wordt en wat de uitvoer is als er een verwerking plaats heeft gevonden.

```
testconfig.txt
propacity_hostname "192.168.11.49"
propacity_port "80"
propacity_root "/propacity"
loadrunner_path "C:\Program Files (x86)\HP\LoadRunner\"
script_path "C:\Users\Edwin\Desktop\LoadrunnerAgent\Scripts\"
```

Function	Passed variable(s)	Expected
Config	"testconfig.txt"	-
GetHostName	-	"192.168.11.49"
GetPort	-	80
GetRoot	-	"/propacity"
GetMDRVPath	-	"C:\Program Files (x86)\HP\LoadRunner\"
GetScriptPath	-	"C:\Users\Edwin\Desktop\LoadrunnerAgent\Scripts\"

Een unit test binnen het testplan voor de class die configuratie bevat.

In de afbeelding staat de test voor het inladen van een configuratiebestand beschreven. In de tabel staan de waarden die worden verwacht als ze worden opgehaald uit de class.
Binnen deze test wordt het bestand aangemaakt met vaste waarden zodat er zekerheid is dat het bestand deze informatie bevat.

8.4 De tests bouwen

Na het beschrijven van de test cases per class wordt er overgegaan naar het schrijven van de daadwerkelijke tests.
Ik heb een aparte applicatie opgezet die de onderdelen van de agent laadt nadat deze gecompileerd zijn. En voor elke klasse wordt er een bijhorende unit test klasse geschreven.

Deze tests worden elke keer gedraaid zodra er wijzigingen in de agent of de tests zelf zijn en er wordt gecompileerd. Mochten de uitkomsten anders zijn dan de verwachtingen dan zal er tijdens het compileren een foutmelding weergeven worden.

```
3>  Running 5 test cases...
3>  }Parsing logfile "testlog.log"
3>  c:/users/edwin/desktop/afstuderen/sprint 3/loadrunner agent/
3>
3>  *** 1 failure detected in test suite "Master Test Suite"
```

Tijdens het compileren van de code wordt er aangegeven of een unit test slaagt of niet.
Op deze afbeelding is te zien dat er een test niet is geslaagd.

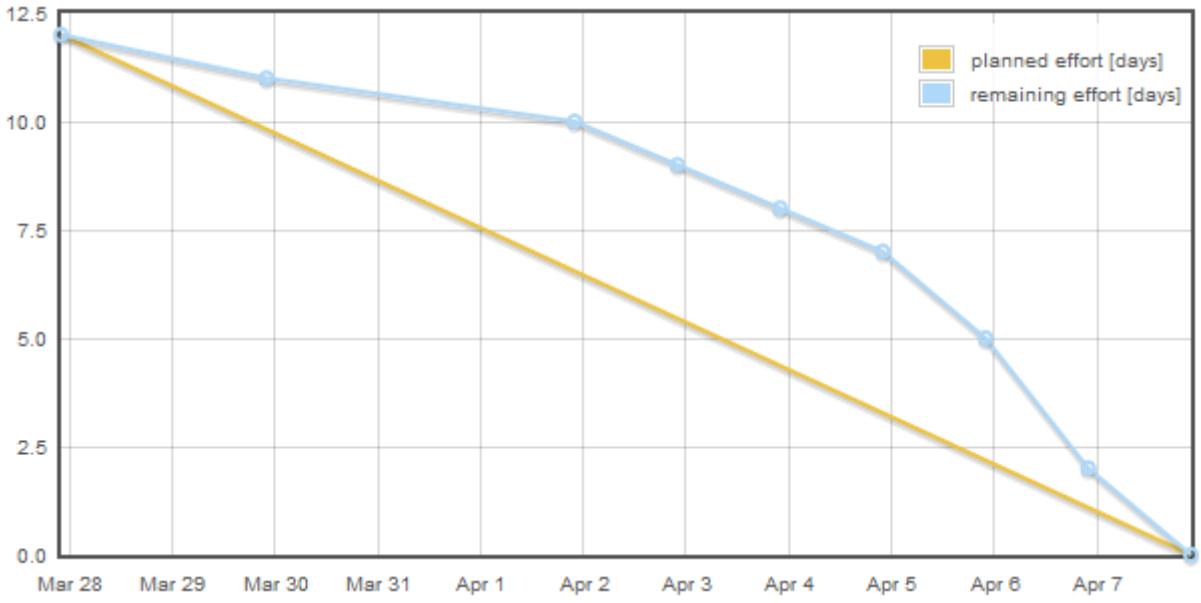
8.5 Performance test

Tot slot wordt de performance getest. Belangrijke test cases in het testplan (zie bijlagen) zijn de weerstand van de agent tegen onverwachte situaties (wat een niet functionele requirement is) zoals een foutief script, het plotseling afbreken van de uitvoering van het script en als het script bleef hangen. Deze situaties herkennen en afhandelen zijn onderdeel van de niet functionele requirements betreft de weerstand tegen onverwachte omstandigheden.
Daaruit blijkt dat de agent enkel om kan gaan met de situatie dat een script succesvol wordt uitgevoerd en dat het script geen fouten bevat. Scripts die bijvoorbeeld blijven hangen of fouten bevatten worden niet afgevangen.

Om dit te verbeteren wordt de zogenaamde 'exit code' bekeken van het proces van de Loadrunner Engine dat tijdens het uitvoeren van een script wordt aangemaakt.
Deze code geeft een indicatie of de Loadrunner Engine geslaagd is om een script te draaien of niet.
Ook is er een maximale wachttijd ingesteld om hangende scripts af te kunnen breken.
Het performance testplan aan het einde van het project nogmaals worden uitgevoerd, maar zal niet nogmaals worden beschreven.

8.6 Evaluatie

In de burndown chart voor deze sprint is duidelijk te zien dat het ontbreken van unit testing in Visual Studio Express invoed heeft gehad op de voortgang. Ik heb oorspronkelijk de aanname gedaan dat dit wel mogelijk was en op basis daar van de sprint ingedeeld.
Vervolgens is er te zien dat het gebruik van Boost unit testing makkelijker was dan verwacht.
Hetgene wat de voorgaande daling in voortgang opheft.



9.Sprint 4: Opslag gegevens

Nadat de uitvoer van scripts en het versturen van metingen is gebouwd, wordt er gewerkt aan de opslag van gegevens. De requirements stellen dat metingen lokaal worden opgeslagen voordat ze verstuurd worden zodat deze niet verloren gaan als Propacity niet bereikbaar is of als de agent of het systeem waar het op draait vast lopen. De opslag van metingen zijn een functionele requirement en sluit ook aan met de niet functionele requirement voor weerstand tegen onverwachte situaties

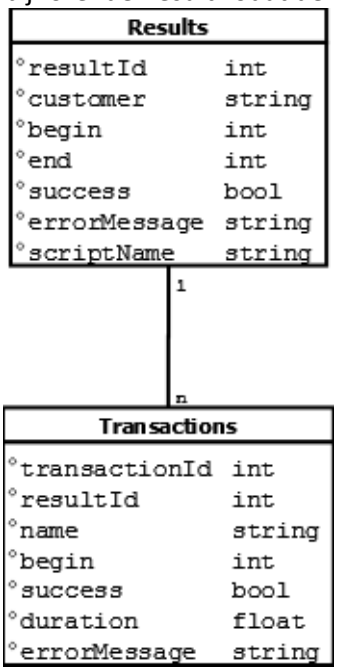
9.1 SQLite

Eerder werd al beschreven dat er een soortgelijke functionaliteit was geïmplementeerd, maar deze voldeed niet aan de eisen en was slechts tijdelijk.
Bijkomende eisen zijn dat de integriteit van de metingen goed hoort te zijn en dat de kans op verlies van data zo min mogelijk is. Beschikbare oplossingen voor deze situatie zijn zeer beperkt. De enige library die aan deze criteria voldoet is SQLite en ik heb deze daarom gekozen om te gebruiken.
Deze library maakt het mogelijk om een database te beheren binnen de applicatie waar het geïmplementeerd is met behulp van slechts een enkel lokaal bestand waarin de gegevens staan opgeslagen.

Het verbinden naar externe systemen komt daarom ook (in tegenstelling tot hoe databases doorgaans gebruikt worden) niet aan de pas. Ook is er geen sprake van enige installatie op het systeem waar het op gebruikt wordt.

9.2 De Database

Om de metingen in de database in te voeren, heb ik een ontwerp gemaakt dat twee tabellen bevat. Een tabel **Results** (informatie over de uitvoer van een script) en **Transactions** (de transactions binnen het script met de duurtijd). Een verzameling metingen wordt binnen de agent in delen de database ingevoerd door één query voor de tabel Result op te bouwen en een query per meting voor Transactions. Elk record in de Transactions tabel krijgt vervolgens de primaire sleutel mee van het bijhorende Result zodat de verzameling van metingen later weer bij elkaar kan worden gevonden.



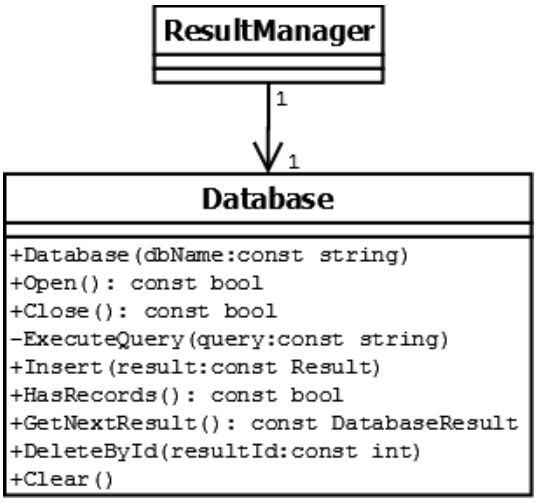
*De SQLite database voor de opslag bevat 2 tabellen, een **Results** tabel en **Transactions** met transactions en hun bijhorende result.*

9.3 Integriteit & Concurrency

Vervolgens wordt er gekeken naar de integriteit van de data. Twee aspecten die hier aan bijdragen, zijn locking en transactions (niet te verwarren met de Transactions binnen in de agent).
Een SQLite database voorziet uit zichzelf voor locking als er een transaction gedaan wordt, een transaction wordt per query gestart en gestopt.
Om zo'n transaction te beheren wordt er handmatig een transaction gestart en gestopt zodat alle benodigde queries binnen een enkele transaction worden uitgevoerd.

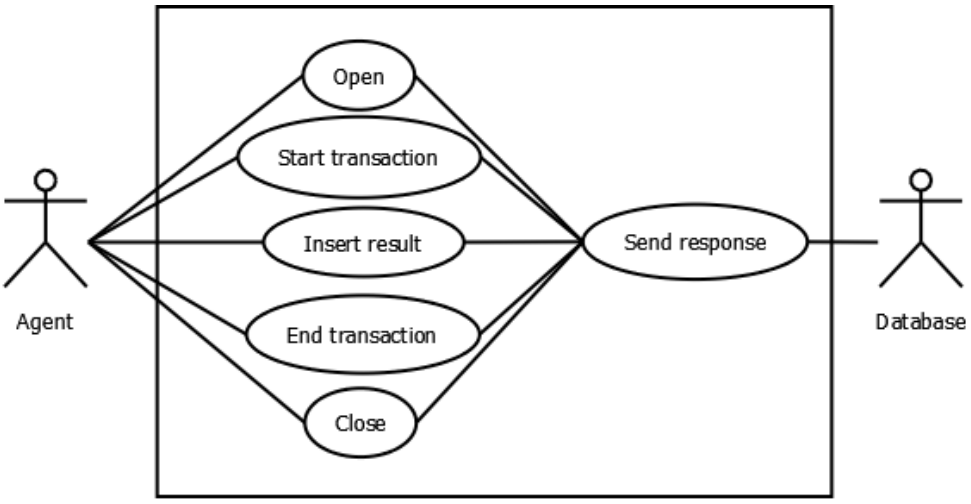
9.4 Ontwerp

Qua ontwerp heb ik gekozen voor een Database klasse die de communicatie met SQLite beheert. Dit wordt ook wel een interface genoemd. De result manager bevat een database object waar deze results naar stuurt. Als eerste wordt de database geopend, hiermee wordt er een lock verkregen op het database bestand waardoor er wordt voorkomen dat een andere applicatie of thread (zie hoofdstuk 10) tussentijds ook data leest of schrijft.
Vervolgens worden alle queries uitgevoerd die nodig zijn om een Result object in de database weg te schrijven en wordt de database weer gesloten.



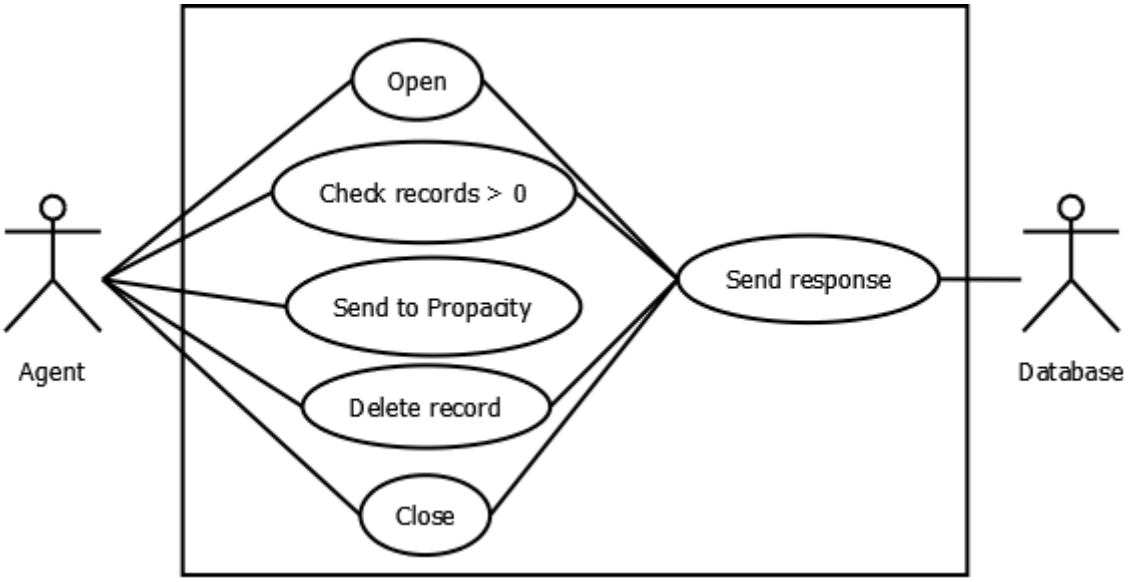
De result manager stuurt en ontvangt results met de Database class.

Het invoeren van een result in de database in een use case afgebeeld.



Een result kan ook weer worden opgevraagd op volgorde van oud naar nieuw. Indien dit resultaat met succes wordt verstuurd naar Propacity met de result manager wordt de data uit de database verwijderd.

Tijdens elke handeling met de database wordt er een result code bekeken die SQLite genereert op basis van of de query is geslaagd.
Als deze code afwijkt van degene die een succesvolle query aangeeft zal het Database object altijd een waarde meegeven aan de result manager dat de query geslaagd is.

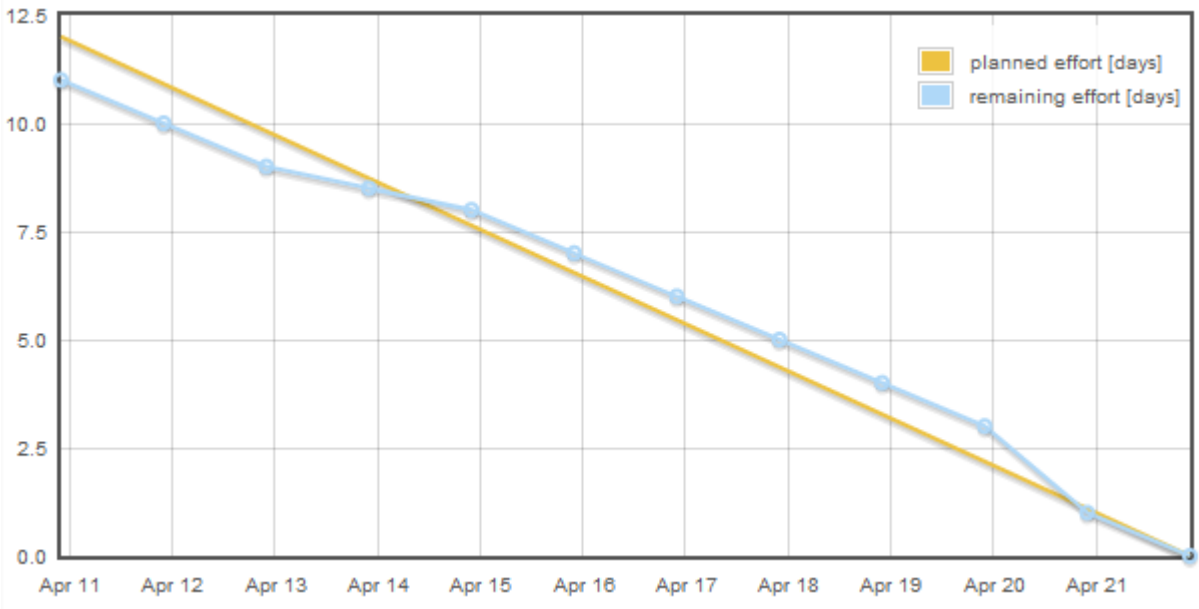


De use case voor het ophalen van een result.
Als deze succesvol wordt verstuurd kan het bijhorende record worden verwijderd.

Door het gebruik van een interface wordt er een makkelijkere manier geïmplementeerd om de database te gebruiken. Indien er gekozen was om herhaaldelijk functies van SQLite zelf te gebruiken, zou dit betekenen dat er onnodig veel code opnieuw wordt geschreven. Hierbij kan gedacht worden aan bijvoorbeeld het bekijken van de result code.
Ook een punt is de beveiliging. In plaats van dat er vanaf de agent elk type query uitgevoerd kan worden op een database is er een selectie gemaakt. Hiermee wordt voorkomen dat er queries worden uitgevoerd vanuit de agent die niet noodzakelijk zijn voor het proces.

9.5 Evaluatie

Deze sprint is gestart met een flinke voorsprong. Dit omdat er in de voortgaande sprint tijd over is gebleven om met SQLite aan de slag te gaan. Dit is daarna gelijk getrokken, omdat een deel van het eerder gebouwde SQLite stuk hergestructureerd moest worden om aan het ontwerp te voldoen. Later in de sprint komt de voortgang weer gelijk met de planning, omdat de zogenaamde 'queueing' van data minder tijd gekost heeft dan verwacht.



10.Sprint 5: Multithreading & Aansturing

Nu de belangrijkste functionaliteiten geïmplementeerd zijn kan er gekeken worden naar het gebruik van multithreading. Dit houdt in dat er stukken code van de agent tegelijkertijd uitgevoerd kunnen worden in plaats van een lineaire volgorde. Deze functionaliteit zal er voor zorgen dat het proces niet zal blijven hangen als scripts te lang duren of als de hoeveelheid scripts zo groot is dat de uitvoer langer duurt dan de wachttijd tussen de uitvoer. Ook wordt er gekeken hoe de agent aangestuurd kan worden door Propacity door middel van het downloaden van scripts en schedules.

10.1 Experiment

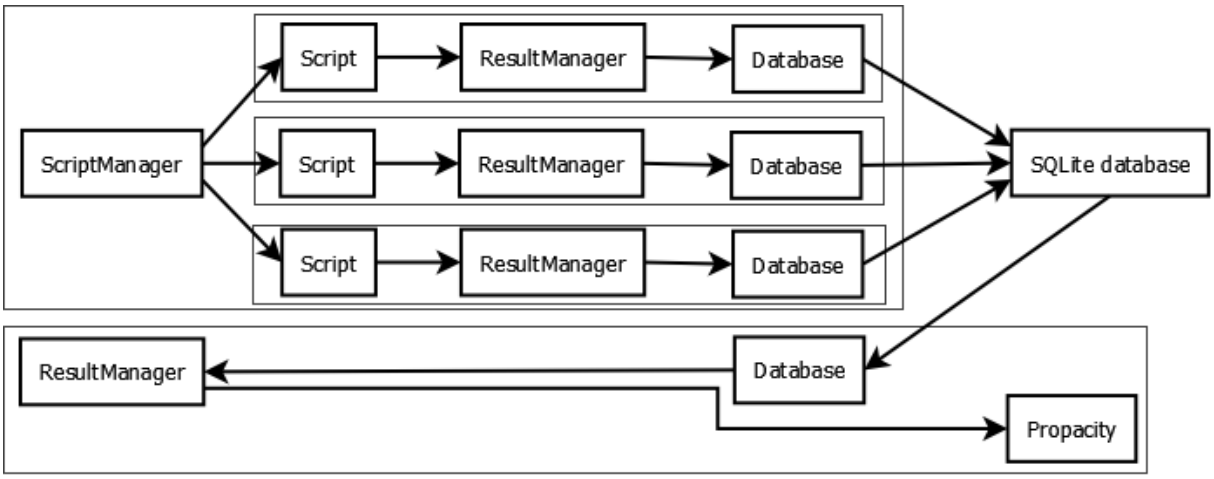
Als eerste wordt er nagedacht op welke manier deze functionaliteit geïmplementeerd kan worden. Er is een tijdelijke ontwikkelomgeving opgezet zodat er geëxperimenteerd kan worden met het gebruik van multithreading met functies van Windows om zo een idee te krijgen van hoe threads te implementeren zijn.
Na dit experiment ligt er de keuze om de Windows multithreading functionaliteit te implementeren met een abstractie voor multiplatform of dat er een andere oplossing hiervoor te gebruiken is. Hierbij komt wederom Boost bij kijken. Het bevat een library die multiplatform multithreading verzorgt.

De keuze valt op Boost omdat (zoals eerder beschreven) dit pakket al aanwezig is binnen het project. En ook omdat het bouwen van een abstractie meer tijd zou kosten dan het gebruik van de Boost library. Ook zou een andere library een extra afhankelijkheid opleveren wat met Boost niet het geval is.

10.2 Ontwerp

Vervolgens wordt er nagedacht waar de verschillende threads uitgevoerd zullen worden. Ik heb gekozen om in de script manager een thread aan te maken per script uitvoer. Deze thread doet hetzelfde als er al gebeurde met uitzondering van een nieuwe manier hoe de result manager gebruikt wordt.

Oorspronkelijk bevatte de script manager een result manager die de resultaten van elke scriptuitvoer beheerde. Nu is het zo veranderd dat elk script thread een result manager bevat, omdat gegevens binnen een thread lastig naar een ander deel van de code te versturen is wegens mogelijk problemen met concurrency. Daarom heb ik gekozen deze gegevens binnen de thread weg te schrijven naar de database.
Bijkomend voordeel daarbij is dat de SQLite database uit zichzelf is voorzien van locking. Op deze manier zal elke script thread moeten wachten tot de database een lock vrij geeft voordat deze mag wegschrijven. Hierdoor hoeft er geen gebruik gemaakt te worden van een zogenaamde mutex (mutual exclusion). Dit houdt in dat de database door meerdere threads wordt gedeeld en zou een mutex beheren welke thread de toegang heeft tot de database zodat er geen transacties door elkaar heen lopen.



Multithreading binnen de agent.Er zijn twee 'hoofd' threads die constant draaien. Binnen de script manager thread kunnen er meerdere 'sub' threads draaien.

Naast deze script threads draait er een andere thread met wederom een result manager. Deze is echter verantwoordelijk voor het ophalen van de results en deze te sturen naar Propacity.
Deze result manager haalt uit de database de results op van oud naar nieuw, op deze manier beinvloed deze thread ook niet de script uitvoer aangezien deze nieuwe records toevoegd aan een tabel. Op deze manier ontstaat er een queue.

Er is de mogelijkheid om het ontwerp enigzins aan te passen om rekening te houden met threading. Dit betreft de result manager. Deze wordt sinds het implenteren van threading voor de helft gebruikt door script uitvoer en voor de andere helft voor het vesturen van metingen.
Een optie is om de result manager op te splitsen, maar dit zou de cohesion verlagen en er is daarom ook niet voor gekozen.

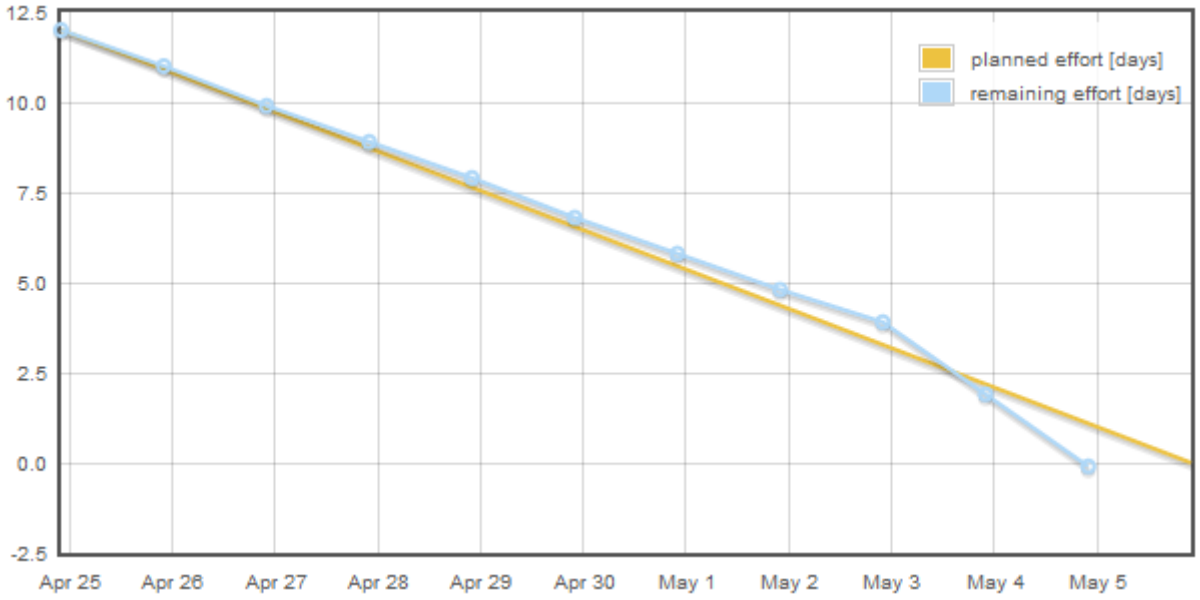
10.3 Aansturing

Tot slot wordt er gekeken naar het aansturen van de agent vanaf Propacity. Dit houdt in dat de schedules en scripts opgevraagd/gedownload worden van de front end.
Echter is er op het moment nog geen functionaliteit beschikbaar binnen Propacity om dit te kunnen implementeren, omdat er verwacht wordt dat het globale project zodanig werkt dat het bij klanten geïmplementeerd kan worden als eerste release en er daarom vanwege een deadline, de werkzaamheden bij andere functionaliteiten liggen.

Als alternatief voor deze requirement heb ik gekozen om per script een schedule configuratiebestand aan te maken. Dit omdat vaste waarden voor een schedule zou betekenen dat deze alleen veranderd kunnen worden door een wijziging in de code waarna er compilatie en het opnieuw implementeren plaats zou vinden.
Een eventuele verbinding naar een soortgelijke tijdelijk webservice zou betekenen dat er mogelijk veel tijd wordt besteed aan een oplossing die uiteindelijk op een andere manier in Propacity gebouwd zou worden zonder dat de huidige oplossing hergebruikt wordt.
Door middel van een bestand met de benodigde informatie is er op zekere manier toch nog controle over de scripts.

10.4 Evaluatie

Deze sprint is voorspoedig verlopen. Het gebruik van multithreading werd door Boost makkelijker gemaakt dan verwacht aangezien er zonder Boost een eigen implementatie gebouwd had moeten worden. Ook vanwege het feit dat SQLite is voorzien van locking is er geen behoefte geweest voor het gebruik van een mutex. Hierdoor is er minder tijd nodig geweest voor het implementeren van multithreading.
Ook is er te zien dat er aan het einde van de sprint dat de voortgang hoog is. Dit omdat een deel van de geplande requirements niet konden worden geïmplementeerd, omdat deze afhankelijk zijn van Propacity.



11.Evaluatie Procesgang

Het proces is naar gevoel zonder grote problemen verlopen. Er is gestart met het opstellen van de requirements welke min of meer in zijn geheel al waren opgeleverd door de opdrachtgever.
Op deze manier is er geen sprake geweest van miscommunicatie over wat er uiteindelijk verwacht werd.

Vanaf het begin was er bij de opdrachtgever al een duidelijk beeld hoe het uiteindelijke product er uit moest komen te zien. Dit omdat deze zelf een rol speelt binnen het globale Propacity project.
Er was dan ook voldoende kennis en expertise aanwezig binnen het bedrijf om mij aan te sturen.

Per week is er overleg geweest over de voortgang van het project en dat van Propacity in het geheel. Er is daarbij gekeken hoe de deelprojecten aansluiten aan elkaar en wat daar eventueel nodig voor was.
Verder werd er ook rekening gehouden met het belang van het bedrijf met betrekking tot het product. Met name de planning die vanaf het begin zo opgesteld is om aan de wensen te voldoen voor de 'Overheid & ICT' beurs.
Met betrekking tot Scrum is er niet sprake geweest van een dagelijkse ‘stand up’, omdat er per dag voldoende informatie werd gecommuniceerd tussen mij en de opdrachtgever waardoor een dagelijks overleg geen toegevoegde waarde zou hebben gehad.

De meest opvallende afwijking van de planning was de implementatie van de agent in de tweede sprint om metingen te verzamelen. Deze wens kwam onverwacht, maar is goed opgevangen en binnen de verwachtte tijd voltooid. Ik zou zoiets voortaan beter kunnen aanpakken door een haastige implementatie op zijn minst functioneel te testen vanwege de beschreven afwijking in de timestamps.

Tijdens elke sprint zijn er ook ontwerpen bijgewerkt om op deze manier het overzicht te behouden en om functionaliteiten eerst te overleggen voordat deze geïmplementeerd werden.
Oorspronkelijk werd er in het afstudeerplan beschreven dat het product uit drie afzonderlijke onderdelen moest bestaan (input, storage en output) en was de aanpak dat er per onderdeel een ontwerp beschikbaar zou zijn.

Echter maken deze zodanig gebruik van elkaar dat er eigenlijk niet meer gesproken kon worden over losse onderdelen, maar één samenhangende verzameling.
Daarom zijn de ontwerpen zo ingericht dat het over het totaalplaatje van het project gaat. De ervaring die ik daar mee heb opgedaan is dat componenten binnen een enkele applicatie nooit echt onafhankelijk kunnen zijn en zou ik die splitsing ook niet meer maken bij toekomstige projecten.

Verder had ik beter onderzoek kunnen doen naar de gebruikte versie van Visual Studio. Deze bleek geen unit testing te bevatten wat grote invloed had op de planning. In de toekomst zou ik voortaan niet blind vertrouwen op een bekend product, maar eerst onderzoek doen of een tool te bieden wat de requirements vragen.
Met het bouwen van de storage is er ook tijd verloren gegaan aan het herschrijven van een deel code dat gebouwd was om SQLite te begrijpen. Hoewel het leerzaam was om dit stuk te bouwen had het tijd kunnen besparen als ik eerst na had gedacht over een ontwerp.

12.Evaluatie Producten

Het eerste product was het plan van aanpak. Dit werd opgesteld aan de hand van de verkregen requirements. Er werd een planning gemaakt door de requirements op te delen in sprints.
Op deze manier konden de backlogs per sprint goed worden opgesteld.

Vervolgens is er het prototype. Het was een eenvoudige applicatie met slechts het laden van gegevens en het gebruik van regular expressions zonder dat er nagedacht was over het ontwerp hiervan met betrekking tot cohesion, coupling en beveiliging.
Dit heeft een goede basis opgeleverd voor het eindresultaat vanwege het feit dat er stukken hergebruikt zijn binnen het ontwerp van het eindproduct.

De tussentijdse implementatie was ook goed gelukt ondanks de krappe planning hiervoor. Het heeft minstens een maand lang aan data gegenereerd voor het gebruik met Propacity. Gedurende die tijd is de applicatie stabiel gebleven, maar is de kwaliteit van data minder goed gebleken dan gehoopt vanwege een afwijking in de timestamps en de manier van opslaan van de metingen.
Dit had voorkomen kunnen worden als er tijd beschikbaar was geweest voor het testen van de integriteit van de data.

De demo voor de 'Overheid en ICT beurs' is met succes opgeleverd. Het was in staat te functioneren zoals het uiteindelijk zou moeten. Het grote verschil tussen de demo en het uiteindelijke product is dat het alleen goed functioneerde onder ideale omstandigheden.
Daarbij kan gedacht worden aan uitval van het systeem en het geen verbinding kunnen maken met Propacity. Desondanks was het wel mogelijk om realtime data te versturen zodat Propacity hier grafieken van kon opbouwen.

Uiteindelijk is er van de demo gewerkt naar het eindproduct, wat bestand is tegen uitval van verbindingen en systemen en op die manier lange tijd blijft functioneren zonder onderhoud.
Met deze weerstand tegen onverwachte situaties in de applicatie is de belangrijkste niet functionele requirement van het project bereikt naast het verwerken van metingen.

Deze bestendigheid tegen uitval werd onder andere voor een deel ook mogelijk gemaakt door het testen van de applicatie.
De unit tests hebben gezorgd dat de integriteit van de data niet ongemerkt omlaag kon gaan aangezien deze tests bij elke wijziging van de code werden gedraaid.
Het performance testen heeft gezorgd dat de agent onverwachte situaties kan opvangen en deze situaties ook kan verwerken in de gegevens die naar Propacity gestuurd worden.

13.Verantwoording Competenties

In het afstudeerplan zijn er een aantal competenties beschreven die ik in dit project heb willen bewijzen.

1. Uitvoeren analyse door definitie van requirements
2. Ontwerpen softwarearchitectuur
3. Ontwerpen systeemdeel
4. Bouwen applicatie
5. Initiëren en plannen van het testproces
6. Uitvoeren van en rapporteren over het testproces.

Voor de analyse van de requirements heb ik de door de opdrachtgever opgeleverde requirements bestudeerd en opgedeeld in drie grote onderdelen, namelijk de input, de storage en de output.
Om de requirements in de planning te verwerken heb ik rekening gehouden met de mijlpaal van April waar er een werkende versie opgeleverd moest worden en vanaf dat punt heb ik de blokken verdeeld over de beschikbare tijd.
Het stuk van de software architectuur heb ik gevisualiseerd (zie bijlagen) in een deployment diagram om met deze een overzicht te hebben van het totaalbeeld van Propacity.
Eveneens diagrammen zijn het klassediagram en het sequentiediagram (zie bijlagen) voor het 'ontwerpen van een systeemdeel'.

Per sprint heb ik gekeken hoe de requirements vertaald konden worden naar functionaliteiten en heb deze incrementeel verwerkt in beide diagrammen.
Vervolgens voor het bouwen van de applicatie heb ik de ontwerpen uitgewerkt tot een applicatie inclusief documentatie voor de code zodat het onderhoudbaar is.
Het testproces heb ik aangepakt door unit tests te schrijven die voor elke functionaliteit een verwachte uitkomst vergelijkt met een werkelijke uitkomst. Mocht er tijdens het ontwikkelen iets veranderen waardoor er een functionaliteit niet meer werkt, dan zou dit een fout opleveren binnen Visual Studio. Deze zal deze opgelost moeten worden voordat de applicatie zich weer in werkende staat kan bevinden.
Voor dit testproces is er van tevoren eerst een document opgesteld waarin deze unit tests beschreven worden met in te voeren data en verwachte data. Ook het testen van de performance is van tevoren beschreven in een testplan met wat voor script er gebruikt wordt in de agent en welke data daar uit hoort te komen.

Literatuur

TR1 Regex tutorial
http://www.codeguru.com/cpp/cpp/cpp_mfc/stl/article.php/c15339

CreateProcess documentatie
http://msdn.microsoft.com/en-us/library/ms682425(v=vs.85).aspx

cURLpp
http://curlpp.org

Exploring the C++ Unit Testing Framework Jungle
http://gamesfromwithin.com/exploring-the-c-unit-testing-framework-jungle

Boost.Test
http://www.boost.org/doc/libs/1_46_1/libs/test/doc/html/index.html

Boost.Test tutorials
http://www.beroux.com/english/articles/boost_unit_testing/
http://www.alittlemadness.com/2009/03/31/c-unit-testing-with-boosttest/

TestDriven.Net
http://testdriven.net/overview.aspx

SQLite
http://www.sqlite.org/

Boost.Thread
http://www.boost.org/doc/libs/1_46_1/doc/html/thread.html

Bijlagen

Begrippenlijst

Class: Een onderdeel binnen het ontwerp van een applicatie dat een verantwoordelijkheid op zich neemt en de bijhorende data bevat.

Front end: De 'voorkant' van een applicatie. Bestaat meestal uit een grafische interface waarmee de gebruiker de applicatie kan gebruiken.

Back end: De 'achterkant' van een applicatie. Bestaat uit het grootste deel van de applicatie, maar heeft vaak geen visuele uitvoer. Verzorgt de front end van informatie.

Agent: Een back end onderdeel van Propacity dat tests uitvoert en resultaten doorstuurt naar de Propacity front end.

Regular Expression: Een manier om patronen in tekst te herkennen.

JSON (Javascript Object Notation): Een manier om data te representeren en een alternatief voor XML.

Cohesion: Hoeveel verantwoordelijkheid een onderdeel heeft.

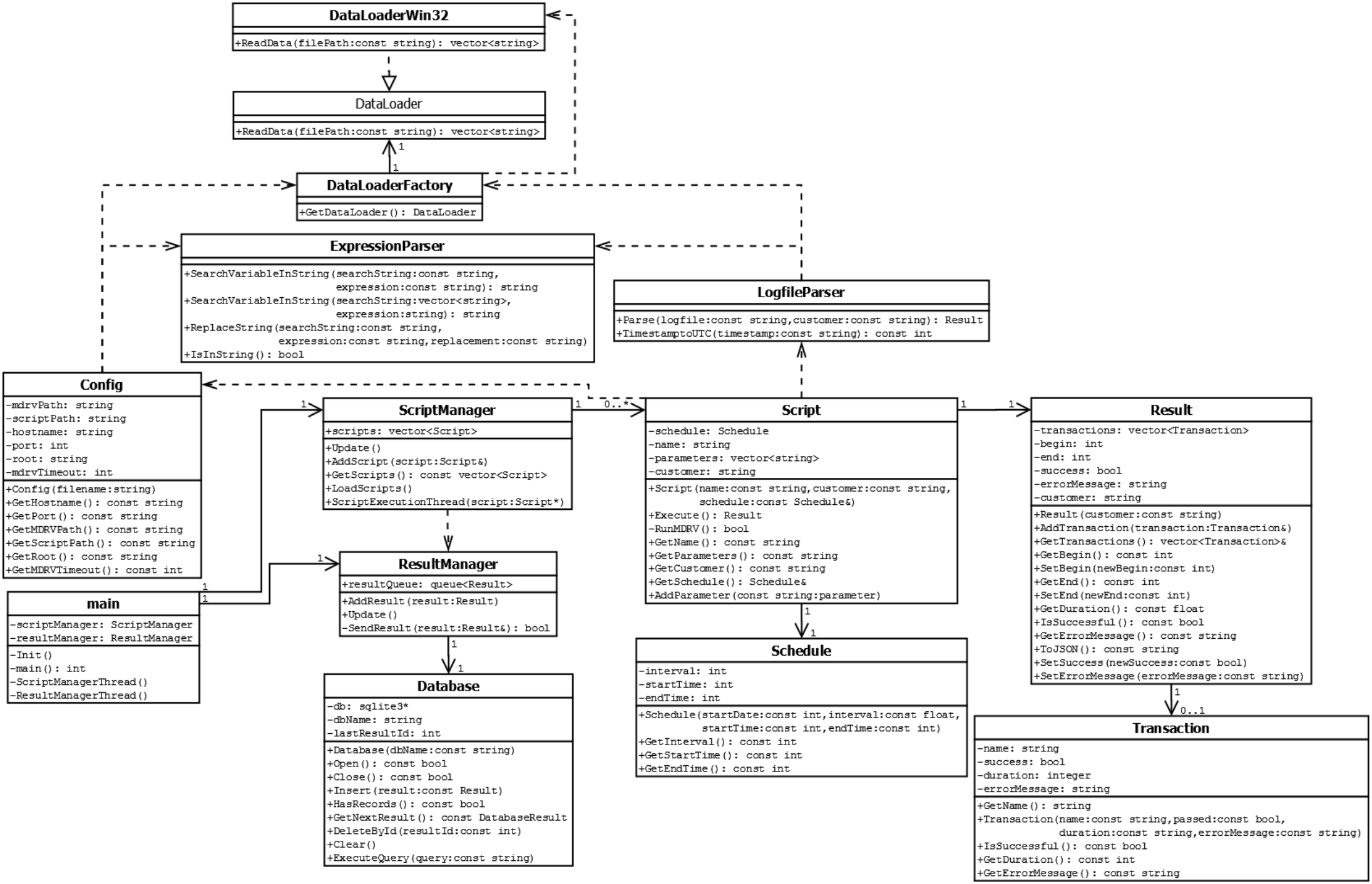
Coupling: Hoe afhankelijk een onderdeel van een ander is.

Webservice: Software die zich bezig houdt met communicatie binnen een netwerk.

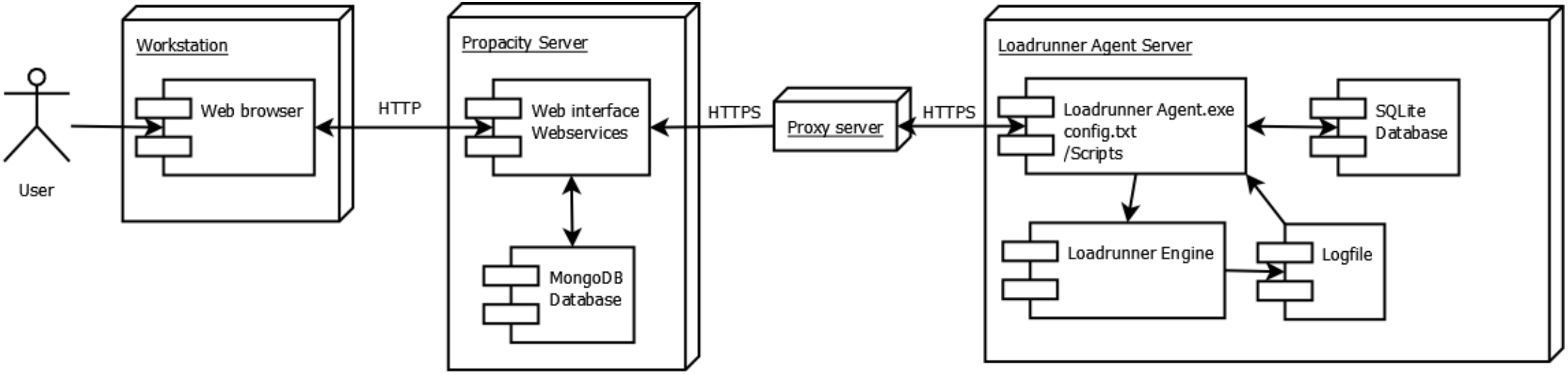
Unit test: Een test waarbij een onderdeel wordt voorzien van gegevens waarvan de uitkomst al bekend is. Als de uitkomst niet overeenkomt met de verwachting faalt de test.

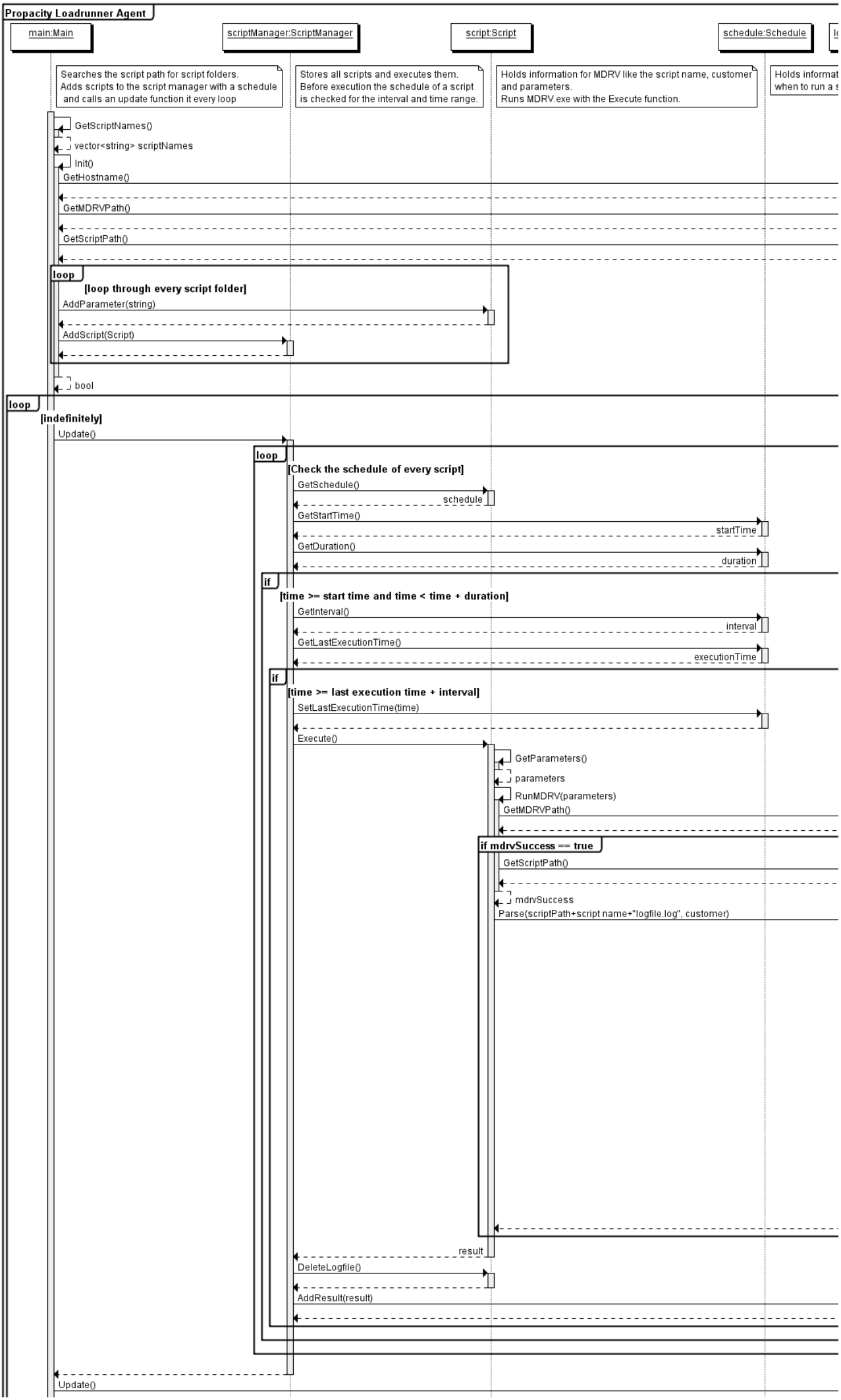
Multithreading: Stukken code parallel uitvoeren op het systeem.

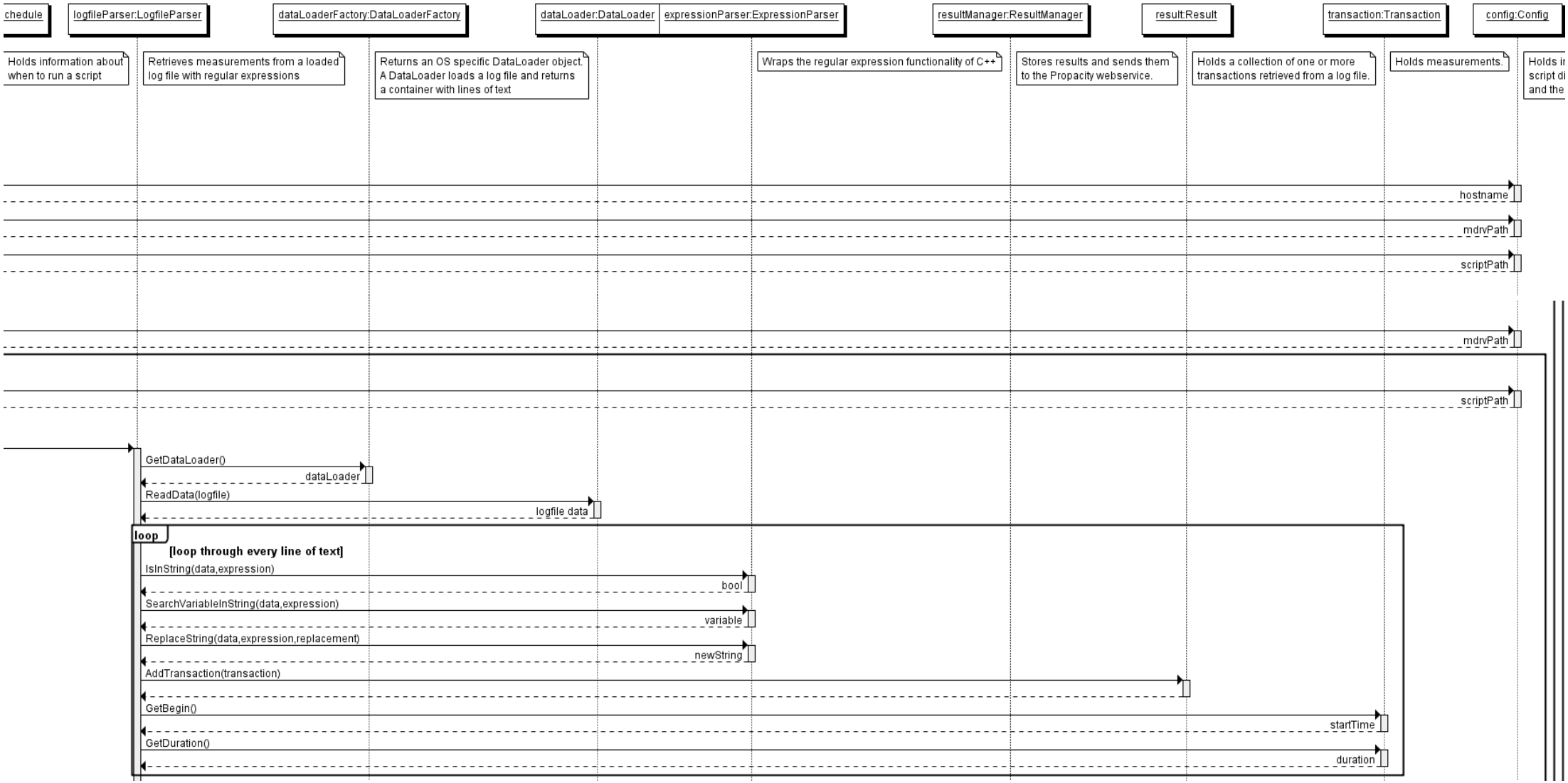
Klassendiagram

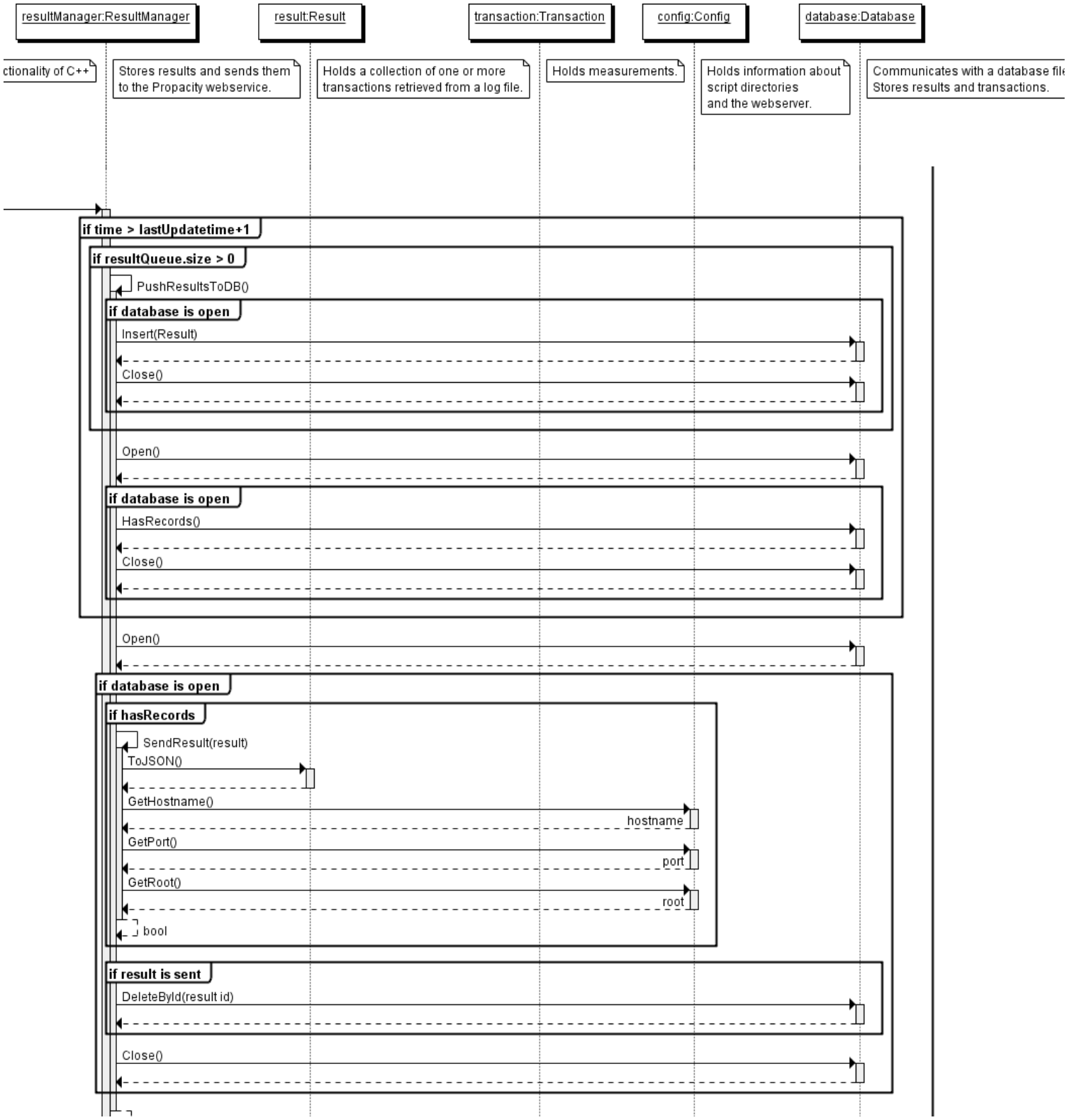


Deployment Diagram & Sequence Diagram




Propacity Loadrunner Agent
main:Main
scriptManager:ScriptManager
script:Script
schedule:Schedule
Searches the script path for script folders.
Adds scripts to the script manager with a schedule
and calls an update function it every loop
Stores all scripts and executes them.
Before execution the schedule of a script
is checked for the interval and time range.
Holds information for MDRV like the script name, customer
and parameters.
Runs MDRV.exe with the Execute function.
GetScriptNames()
vector<string> scriptNames
Init()
GetHostname()
GetMDRVPath()
GetScriptPath()
loop
[loop through every script folder]
AddParameter(string)
AddScript(Script)
bool
loop
[indefinitely]
Update()
loop
[Check the schedule of every script]
GetSchedule()
schedule
GetStartTime()
startTime
GetDuration()
duration
if
[time >= start time and time < time + duration]
GetInterval()
interval
GetLastExecutionTime()
executionTime
if
[time >= last execution time + interval]
SetLastExecutionTime(time)
Execute()
GetParameters()
parameters
RunMDRV(parameters)
GetMDRVPath()
if mdrvSuccess == true
GetScriptPath()
mdrvSuccess
Parse(scriptPath+script name+"logfile.log", customer)
result
DeleteLogfile()
AddResult(result)
Update()


chedule
logfileParser:LogfileParser
dataLoaderFactory:DataLoaderFactory
dataLoader:DataLoader
expressionParser:ExpressionParser
resultManager:ResultManager
result:Result
transaction:Transaction
config:Config
Holds information about when to run a script
Retrieves measurements from a loaded log file with regular expressions
Returns an OS specific DataLoader object. A DataLoader loads a log file and returns a container with lines of text
Wraps the regular expression functionality of C++
Stores results and sends them to the Propacity webservice.
Holds a collection of one or more transactions retrieved from a log file.
Holds measurements.
Holds in script di and the
hostname
mdrvPath
scriptPath
mdrvPath
scriptPath
GetDataLoader()
dataLoader
ReadData(logfile)
logfile data
loop
[loop through every line of text]
IsInString(data,expression)
bool
SearchVariableInString(data,expression)
variable
ReplaceString(data,expression,replacement)
newString
AddTransaction(transaction)
GetBegin()
startTime
GetDuration()
duration


resultManager:ResultManager
result:Result
transaction:Transaction
config:Config
database:Database
ctionality of C++
Stores results and sends them to the Propacity webservice.
Holds a collection of one or more transactions retrieved from a log file.
Holds measurements.
Holds information about script directories and the webserver.
Communicates with a database file Stores results and transactions.
if time > lastUpdatetime+1
if resultQueue.size > 0
PushResultsToDB()
if database is open
Insert(Result)
Close()
Open()
if database is open
HasRecords()
Close()
Open()
if database is open
if hasRecords
SendResult(result)
ToJSON()
GetHostname()
hostname
GetPort()
port
GetRoot()
root
bool
if result is sent
DeleteById(result id)
Close()

Testplan Unit testing

Config

testconfig.txt
propacity_hostname "192.168.11.49"
propacity_port "80"
propacity_root "/propacity"
loadrunner_path "C:\Program Files (x86)\HP\LoadRunner\"
script_path "C:\Users\Edwin\Desktop\LoadrunnerAgent\Scripts\"

Function	**Passed variable(s)**	**Expected**
Config	“testconfig.txt”	-
GetHostName	-	“192.168.11.49”
GetPort	-	80
GetRoot	-	“/propacity”
GetMDRVPath	-	“C:\Program Files (x86)\HP\LoadRunner\"
GetScriptPath	-	“C:\Users\Edwin\Desktop\LoadrunnerAgent\Scripts\”

DataLoaderWin32

Function	**Passed variable(s)**	**Expected**
ReadData	Logfile of the first script found in folder	Vector size not 0

ExpressionParser

Function	Passed variable(s)	Expected
SearchVariableInString	“test string test”, “()(string)()”	“string”
SearchVariableInString	Vector with strings, expression	“string”
ReplaceString	"1,678",",","."	“1.678”

LogfileParser

Function	Passed variable(s)	Expected
Parse	“testlog.log”	Result object which is not empty.

Result

Function	Passed variable(s)	Expected
Result	True, 10, 20, “test error”, “computest”	-
AddTransaction	2 transactions	-
GetTransactions	-	Vector with 2 transactions.
GetBegin	-	Equals the start time of transaction 1.
GetDuration	-	Equals durations of t1 and t2.
IsSuccessful	-	Is not true if any transaction is false.
GetErrorMessage	-	“test error”
ToJSON	-	Not an empty string.

Schedule

Function	Passed variable(s)	Expected
Schedule	10,30,10,2	-
SetLastExecutionTime	1	-

GetLastExecutionTime	-	1
GetInterval	-	2
GetDuration	-	10

Transaction

Function	**Passed variable(s)**	**Expected**
Transaction	“testTransaction”,true, 5, 0, “testerror”	-
GetName	-	“testTransaction”
GetBegin	-	0
IsSuccessful	-	true
GetDuration	-	5
GetErrorMessage	-	“testerror”

Script

Function	Passed variable(s)	Expected
Script	“Testscript,”computest”,Schedule(0,0,10,2))	-
Execute	-	Default result with error.
Script	Existing script, “computest”,Schedule(0,0,10,2)	-
AddParameters	Various parameters	-
GetParameters	-	Not an empty string
Execute	-	Filled result set.
GetName	-	Script name used in previous constructor.
GetCustomer	-	“computest”
GetSchedule	-	A schedule with the same values used in the constructor

ScriptManager

Function	Passed variable(s)	Expected
AddScript	Script	-
GetScripts	-	Vector size 1.

Functioneel Testplan

Configuratie

-De agent bevat een executable met een Scripts folder en een configuratiebestand genaamd config.txt.
-De agent kan niet worden uitgevoerd met een ontbrekend of leeg configuratiebestand.
-Er zijn standaardvariabelen ingesteld voor de waarden port, proxy_type, proxy_port en root.
-De volgende variabelen zijn in te stellen zonder dat er fouten optreden, als er een variabele wordt ingesteld zonder waarde moet dit een default waarde opleveren :
propacity_hostname
propacity_port
auth_key
proxy_enabled
proxy_type
proxy_hostname
proxy_port
proxy_username
proxy_password
loadrunner_path
script_path
mdrv_timeout

-De agent laadt de scripts volgens de structuur /Scripts/[klantnaam]/[componentnaam]/[scriptnaam]
-Zonder scripts moet de agent geen foutenopleveren.
-In de script folder kan een .schedule bestand zich bevinden met dezelfde naam als het script. Dit schedule zal niet geladen worden onder een andere naam binnen dit script.
-In het schedule bestand kunnen de volgende waarden worden ingesteld:
active
interval
duration

Als het schedule bestand ontbreekt moet de agent dit detecteren en zal deze het script niet inladen.

Executie

-Scripts die te lang duren moeten worden afgebroken na de ingestelde timeout variabele in de configuratie, mocht het script een fout bevatten dan wordt dat ook gedaan.
-Er moeten meerdere scripts tegelijk uitgevoerd kunnen worden.
-Tijdens dat de agent draait kunnen er nieuwe scripts worden toegevoegd en worden deze gedecteerd en uitgevoerd.
-Als mdrv.exe wordt afgebroken tijdens de uitvoer van een script, zal de agent dit als een error afvangen.
-Als mdrv.exe niet beschikbaar is zal de agent dit detecteren.

Resultaten

-Na het uitvoeren van tenminste 1 script zal er een nieuw bestand in de root staan genaamd measurements.sql3.
-De metingen die worden verzonden naar het vooraf ingestelde ip met port en proxy.
-Als de verbinding wordt verbroken met het internet, zullen de metingen met terugwerkende kracht worden verzonden.

Input

De volgende invoer wordt voor deze test gebruikt, het betreft een configuratiebestand en één script. Het configuratiebestand zal worden ingesteld met alle mogelijke variabelen die eerder beschreven zijn. Het betreffende script is 'Computest_110311' en bevat 6 transacties, voor dit script wordt een schedule aangemaakt met de waarden active 1 en interval 60.

Output

De output zal 6 transacties bevatten en zich in Propacity bevinden.