Testen *voor* Testen

"Ontwikkeling van geautomatiseerde testmethodes voor de binnen Dare!! Instruments ontwikkelde EMC meetapparatuur (RadiCentre producten) en EMC testsoftware (RadiMation)"

Bachelor Scriptie

Frank Elshout

Testen *voor* Testen

"Ontwikkeling geautomatiseerde testmethodes voor de binnen Dare!! Instruments ontwikkelde EMC meetapparatuur (RadiCentre producten) en EMC testsoftware (RadiMation)"

Frank Elshout
Haagse Hogeschool – TIS Delft
Opleiding Elektrotechniek
studentnummer: 10046410
Woerden, 2014

Samenvatting

Een van de belangrijkste onderdelen van ieder project waarin een product wordt ontwikkeld is het testen. De testdichtheid en de kwaliteit van de test is bepalend voor de uiteindelijke productkwaliteit. Handmatige testprocessen zijn echter vaak tijdrovend en de complexiteit van de test neemt enorm toe naarmate het aantal mogelijkheden uitbreidt. De dekking en de kwaliteit van de test wordt hierdoor negatief beïnvloed. In veel gevallen is daarom een automatische testprocedure gewenst, waarin eenmalig gedefinieerd kan worden wat er getest moet worden en aan welke criteria het te testen product moet voldoen.

De kernactiviteiten van het bedrijf Dare!! Instruments zijn het ontwikkelen van EMC meet- en testapparatuur en EMC Software. Deze producten worden wereldwijd gebruikt bij vooraanstaande EMC testlaboratoria. De kwaliteit van de ontwikkelde producten is bepalend voor de kwaliteit van de EMC testen en is daarom van groot belang. Tegelijk neemt het aantal mogelijkheden van de ontwikkelde producten en software toe. Dit maakt dat binnen Dare!! Instruments de wens is ontstaan om de ontwikkelde producten geautomatiseerd te kunnen testen. Hieruit is het afstudeerproject ontstaan dat in deze scriptie is beschreven.

In de eerste fase van het afstudeerproject is een geautomatiseerde test ontwikkeld voor het testen van de door Dare!! Instruments ontwikkelde RadiCentre producten. Het RadiCentre is een centraal meetinstrument wat kan worden ingericht met de bijbehorende RadiCentre insteekkaarten. Alle RadiCentre producten kunnen remote worden aangestuurd en geconfigureerd. De ontwikkelde test bestaat uit een generiek testprogramma en een bijbehorende commandodatabase waarin alle specifieke commando's zijn opgenomen voor alle momenteel ondersteunde RadiCentre producten. Het testprogramma is dusdanig opgebouwd dat niet alleen individuele commando's worden getest, maar het ook mogelijk is om verschillende commando's in een gedefinieerde volgorde te testen. Dit maakt het mogelijk om uitgebreide testsequenties in de commandodatabase te definiëren. Dankzij deze opzet is de specifieke testfunctionaliteit van iedere RadiCentre product in de database gedefinieerd, waardoor specifieke testfunctionaliteit eenvoudig kan worden aangepast of uitgebreid via deze database, zonder dat hiervoor het generieke TestComplete testprogramma hoeft worden aangepast. Het succes van de ontwikkelde testmethode heeft zich tijdens de afstudeerperiode bewezen, doordat verschillende fouten en problemen zijn ontdekt met behulp van het ontwikkelde testprogramma.

In de tweede fase van het afstudeerproject is een geautomatiseerde test ontwikkeld voor het testen van de door Dare!! Instruments ontwikkelde RadiMation software. RadiMation kan worden gebruikt voor het volledig geautomatiseerd uitvoeren van EMC metingen, testen en kalibraties. Voor de test is gebruik gemaakt van een bestaand TestComplete testprogramma. Dit verouderde testprogramma was niet meer bruikbaar voor de meest recente versie van RadiMation. De belangrijkste testprocedures voor het in RadiMation uitvoeren van EMC metingen, testen en kalibraties zijn herzien en functioneel gemaakt voor de laatste RadiMation software versie. Daarnaast zijn een aantal uitbreidingen geïmplementeerd waarmee functionaliteit aan de test is toegevoegd. Ook voor deze test geldt dat verschillende tot dan toe onbekende problemen zijn gevonden dankzij het ontwikkelde testprogramma.

Voor beide projecten is een succevolle implemenatie ontwikkeld, welke voldoen aan de gestelde eisen. Dit wordt mede onderstreept door de waardevolle testresultaten die zijn verkregen met de ontwikkelde testmethodes. Hieruit kan worden gecontroludeerd dat de doelstellingen van het gehele project zijn behaald.

Naast deze conclusie zijn een aantal aanbevelingen gedaan voor het eventuele vervolg van de projecten.

Voor de RadiCentre test wordt aanbevolen om deze uit te breiden met een automatische detectie van het via USB aangesloten RadiCentre. Dit kan het testproces optimaliseren, met name wanneer grotere aantalen RadiCentre´s worden getest.

De belangrijkste aanbeveling voor de RadiMation test is om de bestaande testprocedures, bestaande uit de grote hoeveelheid scriptroutines, grondig te herstructureren en selecteren, om de onderhoudbaarheid en uitbreidbaarheid voor de toekomst te verbeteren.

Voorwoord

Testen is het kernwoord als het gaat om het vakgebied EMC, waar Dare!! Instruments in is gespecialiseerd. Testen is ook het kernwoord van het afstudeertraject waar deze scriptie over schrijft, waarin methodes zijn ontwikkeld voor het testen van de EMC apparatuur en EMC testsoftware van Dare!! Instruments. Testen *voor* Testen is daarom de titel van deze scriptie die ik met enige trots aan u presenteer. Deze scriptie is het resultaat van een succesvol afstudeertraject, uitgevoerd binnen het bedrijf Dare!! Instruments.

Dit afstudeertraject is de bekroning op mijn studie Elektrotechniek aan de Haagse Hogeschool in Delft. Deze studie heeft voor mij een belangrijke basis gevormd voor zowel het inhoudelijke kennisniveau als het algemene denkniveau wat ik tijdens de opleiding heb ontwikkeld. Bij deze wil ik daarom graag mijn waardering uitspreken voor de docenten van de opleiding Elektrotechniek, waarbij ik iedereen hartelijk bedank voor het professionele onderwijs en de waardevolle begeleiding tijdens de opleiding.

Ook wil ik van de gelegenheid gebruik maken om mijn dank uit te spreken voor iedereen die een bijdrage heeft geleverd aan het succes van deze afstudeerstage.

In de eerste plaats wil ik John de Rooij bedanken voor de begeleiding vanuit Dare!! Instruments tijdens de stage. Het succes van de afstudeerstage is voor een belangrijk deel te danken aan de waardevolle begeleiding en technisch inhoudelijke bijdrage tijdens de stage. Daarnaast heb ik de prettige samenwerking ook enorm gewaardeerd.

Daarnaast wil ik graag Harry Broeders bedanken voor de begeleiding als docent van de Haagse Hogeschool. De begeleiding en betrokkenheid bij het afstudeertraject zijn enorm gewaardeerd, evenals de inhoudelijke bijdrage voor de scriptie en altijd snelle reacties op de gestelde vragen.

Als laatste wil ik u veel plezier wensen bij het lezen van mijn scriptie.

Frank Elshout

20 maart 2014, Woerden

Inhoudsopgave

1 Inleiding

Deze scriptie beschrijft het afstudeerproject wat is uitgevoerd bij Dare!! Instruments, specialist op het gebiedt van EMC meet- en testapparatuur en EMC software. Binnen dit afstudeerproject zijn twee verschillende oplossingen ontwikkeld voor het geautomatiseerd testen van de door Dare!! Instruments ontwikkelde producten.

1.1 Dare!! Instruments

De kernactiviteiten van het in 1992 opgerichte bedrijf Dare!! draaien om het vakgebied EMC, wat staat voor ElektroMagnetische Compatibiliteit. Onder EMC wordt verstaan in hoeverre een apparaat gevoelig is voor elektromagnetische straling van de omgeving en in welke mate het apparaat zelf elektromagnetische straling afgeeft aan de omgeving.

In de loop der tijd zijn de bedrijfactiviteiten van Dare!! sterk uitgebreid en verdeeld in verschillende bedrijfsonderdelen. Voor alle bedrijfsonderdelen geldt nog steeds EMC als belangrijkste vakgebied voor de kernactiviteiten.

Eén van de huidige onderdelen is het bedrijf Dare!! Instruments. Binnen Dare!! Instruments wordt professionele EMC meetapparatuur en EMC testsoftware ontwikkeld, gericht op het volledig geautomatiseerd uitvoeren van EMC testen.

De belangrijkste onderdelen van het huidige productportfolio zijn het RadiCentre met bijbehorende insteekkaarten en de RadiMation software. Het RadiCentre wordt gebruikt als het centrale apparaat in een EMC testlaboratorium. De verschillende meetapparatuur kan via de insteekkaarten op het RadiCentre worden aangesloten en op deze manier worden bestuurd. Het RadiCentre en de aangesloten apparatuur kunnen vervolgens volledig worden aangestuurd vanuit de RadiMation EMC Test software. Met deze combinatie kunnen verschillende EMC metingen volledig geautomatiseerd worden uitgevoerd.

1.2 Opdracht

Een belangrijk onderdeel van alle project waarin producten worden ontwikkeld is het testen van de ontwikkelde producten. Dit testen kan op allerlei verschillende manieren worden vormgegeven. Wanneer het aantal mogelijkheden van een product toeneemt, is het vaak wenselijk om gebruik te kunnen maken van een geautomatiseerde testmethode. De dekking van deze methodes is al snel veel hoger dan wanneer handmatig getest wordt. Daarnaast kunnen geautomatiseerde testen in veel gevallen vele malen sneller worden uitgevoerd.

1.2.1 Aanleiding

De door Dare!! Instruments ontwikkelde meetapparatuur en software worden wereldwijd verkocht aan de meest vooraanstaande EMC laboratoria. De EMC metingen zijn van groot belang voor de ontwikkeling van elektronica in het algemeen. De kwaliteit van de door Dare!! Instruments ontwikkelde producten is daarmee automatisch ook zeer belangrijk. Om deze kwaliteit te kunnen handhaven wil Dare!! Instruments gebruik maken van geautomatiseerde testmethodes, zowel voor de ontwikkelde hardware als software. De testmethodes moeten inzetbaar zijn bij zowel productie als bij ontwikkeling van nieuwe of uitbreiding van bestaande producten.

Deze wens is de aanleiding geweest voor het afstudeerproject welke wordt beschreven in deze scriptie.

1.2.2 Doel

Het afstudeertraject is verdeeld in twee individuele projecten.

Het doel van het eerste project is het ontwikkelen van een volledig geautomatiseerde testmethode voor het testen van alle momenteel ondersteunde RadiCentre producten. Voor ieder RadiCentre product moeten hierin alle commando's minimaal één keer getest worden. Daarnaast moeten ook totale functionaliteiten van een product (meestal aangestuurd met meerdere commando's) getest kunnen worden. De test moet uitgevoerd kunnen worden via alle ondersteunde communicatiemethodes, bestaande uit GPIB, USB, LAN en RS232. Aan het einde van de test moeten de resultaten automatisch worden gerapporteerd.

Het doel van het tweede project is het herontwikkelen van een bestaande testmethode voor het geautomatiseerd testen van de RadiMation EMC Software. De bestaande methode is verouderd en niet meer toepasbaar voor de meest recente versie van RadiMation. De belangrijkste onderdelen van deze methode moeten worden herontwikkeld zodat deze gebruikt kunnen worden voor de laatste versie van RadiMation. Daarnaast zijn er een aantal specifieke wensen voor de uitbreiding van de testmethode.

Beide testmethodes moeten worden ontwikkeld met behulp van het programma TestComplete.

Specifieke details over de eisen en wensen van ieder project worden nader toegelicht in de betreffende hoofdstukken van de projecten.

1.3 TestComplete

Het programma TestComplete kan worden gebruikt voor het ontwikkelen en uitvoeren van geautomatiseerde testmethodes. TestComplete is in de eerste plaats gericht op het kunnen testen PC Software. Veel ingebouwde functies zijn daarom ook gericht op het kunnen bedienen en uitlezen van software applicaties. Het is in testcomplete mogelijk om een test te ontwikkelen met scripttalen, waaronder VBScript. Hierdoor is het dus mogelijk om geheel eigen testprogramma's te programmeren, onafhankelijk van de speciale testcomplete functies. Dit maakt dat de inzetbaarheid van TestComplete veel groter is dan alleen het testen van computerapplicaties. [1]

In deze paragraaf zullen een aantal specifieke TestComplete functies worden toegelicht, welke van belang zijn bij de beschrijving van de ontwikkelde testmethodes.

1.3.1 Testen .NET applicaties

Eén van de specialiteiten van TestComplete is het kunnen testen van zogeheten open applicaties, zoals .NET applicaties [2]. TestComplete heeft een breed assortiment aan ingebouwde functies, waardoor alle gebruikersinterfaceobjecten en zelfs specifieke klassen van een applicatie direct kunnen worden benaderd vanuit bijvoorbeeld een TestComplete script. Dit maakt het eenvoudig om de te testen Windows applicatie direct te besturen via de memberfuncties van deze gebruikersinterfaceobjecten. Het is dus niet nodig om interfaceobjecten te bedienen via bijvoorbeeld muisklik simulaties op exacte schermposities, objecten kunnen eenvoudig worden benaderd via de

instantienaam of weergave tekst. Onderstaand voorbeeld laat zien hoe een knop met de naam “MyButton” kan worden bediend in de applicatie met procesnaam “MyApplication”.

```
Sys.Process("MyApplication").WinFormsObject("MyButton").ClickButton
```

1.3.2 TestItems

Ieder testproject in TestComplete bestaat uit een aantal Test Items [3]. Deze Test Items kunnen worden gebruikt om aan te geven welke onderdelen van een testproject wel en niet moeten worden uitgevoerd. De TestItems vormen hiermee de interface voor de ontwikkelde test.

Wanneer de test is ontwikkeld met één van de beschikbare scripttalen wordt ieder TestItem gekoppeld aan exact één scriptroutine. Wanneer een TestItem is geselecteerd voor de test, zal deze gekoppelde routine worden uitgevoerd.

Een TestComplete testprogramma bestaat daardoor meestal niet uit één hoofdroutine, maar uit verschillende deelroutine die worden aangeroepen door de verschillende TestItems.

Wanneer een geparametriseerde routine is gekoppeld aan een TestItem, dan kunnen de waardes van deze parameters voor ieder testitem individueel worden ingesteld. [bron]

1.3.3 Project Variabelen

In ieder TestComplete project kunnen projectvariabelen worden ingesteld [4]. Deze variabelen en de waardes hiervan kunnen door de gebruiker worden ingesteld via de projectpagina van het TestComplete project. De waardes van deze projectvariabelen kunnen vervolgens eenvoudig worden uitgelezen door het ontwikkelde testprogramma. Projectvariabelen kunnen bijvoorbeeld worden gebruikt als gebruikersinterface voor geavanceerde testparameters.

1.3.4 Automatisch genereren van testrapport

Eén van de belangrijkste mogelijkheden van TestComplete is de automatische rapportgenerator [5]. In ieder testproject kan gedetailleerd worden gespecificeerd welke informatie op welke manier moet worden opgenomen in het rapport. In een scriptproject kan dit eenvoudig met de ingebouwde log functie. Hiermee kunnen onder andere checkpoints, waarschuwingen, foutmeldingen en informatieve berichten worden toegevoegd. Wanneer een computer applicatie wordt getest en hierbij gebruik gemaakt wordt van ingebouwde TestComplete functies, wordt veel informatie automatisch toegevoegd aan de log, bijvoorbeeld wanneer een knop wordt bediend of wanneer een onverwacht venster wordt gedetecteerd.

1.3.5 Storages.ini object voor werken met .ini bestanden

TestComplete heeft een ingebouwde functie voor het lezen van en schrijven naar .ini bestanden. Deze functie is vanuit een testscript te benaderen via het `storages.ini` object [6]. Een belangrijke voorwaarde voor dit object is dat het betreffende .ini bestand moet beschikken over een sectie met de naam”root”. Dit hoeft niet de enige of eerste sectie te zijn. Ook hoeft deze sectie geen items te bevatten. De enige voorwaarde is dat de sectie bestaat.

1.3.6 CLR-Bridge

In een TestComplete kan een DLL bibliotheek worden toegevoegd aan een project via de CLR-Bridge functie [7]. Componenten en bibliotheken die zijn toegevoegd via de CLR-bridge zijn in het script eenvoudig te benaderen als memberobject van het speciale `dotNET` object. De belangrijkste beperkingen van VBScript zijn hierin afgevangen. Zo wordt de constructor van een klasse (eventueel met parameters) automatisch als public memberfunctie toegevoegd met de naam `zctor`. Deze constructie maakt het onder andere mogelijk om constructors met parameters te gebruiken, wat standaard niet mogelijk is in VBScript.

2 RadiCentre commando test (fase 1)

In de eerste fase van het afstudeertraject is een methode ontwikkeld voor het geautomatiseerd testen van alle RadiCentre producten. In deze test worden alle aangesloten RadiCentre producten individueel getest. De kern van de test bestaat uit het testen van alle beschikbare commando's van de RadiCentre producten. Het basisprincipe van de test bestaat steeds uit:

- Het sturen van een specifiek commando (bijvoorbeeld: stel een frequentie in)
- Het controleren van het hierop gestuurde antwoord (bijvoorbeeld: OK)
- Het controleren van het resultaat (bijvoorbeeld: vraag de actuele frequentie)

Voor deze test is een project ontwikkeld in het programma TestComplete. Dit project dat kan worden gebruikt voor het geautomatiseerd uitvoeren en rapporteren van de test. TestComplete biedt verschillende methodes voor het ontwikkelen van geautomatiseerde testen. Voor dit project is gekozen om de test te ontwikkelen in een scripttaal. Deze methode biedt veruit de meeste mogelijkheden, en is daarnaast ook de meest generieke en TestComplete onafhankelijke methode.

TestComplete ondersteunt de talen VBScript, Microsoft JScript en DelphiScript. Er is gekozen voor ontwikkeling in VBScript, enerzijds omdat dit in TestComplete de standaard en daardoor best ondersteunde scripttaal is. Anderzijds omdat er binnen Dare!! Instruments veruit de meeste ervaring is met deze taal.

2.1 Functionele eisen

Voor dit deelproject zijn de volgende eisen en wensen opgesteld:

Testen via alle beschikbare communicatieprotocollen
De test moet kunnen worden uitgevoerd via alle communicatieprotocollen die het RadiCentre ondersteunt. Voor het meest recente RadiCentre's is dit GPIB, RS-232, USB en LAN.
De test engineer moet kunnen aangeven welke communicatie methodes wel en niet gebruikt moeten worden. (Omdat niet elke RadiCentre is uitgevoerd met alle communicatiemogelijkheden)

Automatische detectie van aangesloten insteekkaarten
Bij iedere test moet automatisch worden gedetecteerd welke insteekkaarten zijn aangesloten in het RadiCentre. Hierbij moet de typenaam, softwareversie en benodigde prefix worden bepaald met behulp van het generieke commando "*IDN?".

Testen van alle beschikbare commando's aan de hand van een commandodatabase
Voor de test wordt een commandodatabase opgebouwd, met hierin alle beschikbare commando's van alle RadiCentre producten. De uitvoer van de test wordt bepaald door deze commandodatabase. Iedere insteekkaart wordt individueel getest. Per apparaat worden alle bijbehorende commando's uit de database uitgelezen en uitgevoerd. De test bepaald aan de hand van het verwachte antwoord (als reguliere expressie) of het verkregen antwoord voldoet.
Zoals beschreven is de volgorde van de database bepalend voor de uitvoer. Dit geldt echter voor ieder apparaat individueel. Wanneer er twee apparaten worden getest, worden eerst alle commando's van apparaat A opgezocht in de database en vervolgens getest. Vervolgens wordt de

database opnieuw doorzocht voor alle commando's voor apparaat B en worden deze in volgorde getest.

Optioneel kunnen testen van externe hardware

Extern aangesloten hardware (dus hardware die is aangesloten op een insteekkaart, bijvoorbeeld een meetprobe) kan als optie worden geselecteerd voor de test. Door de grote variëteit aan beschikbare externe hardware wordt deze niet automatisch herkend door de test. De test engineer dient het type en het exacte adres handmatig op te geven. Per insteekkaart kan maximaal één extern aangesloten apparaat worden opgenomen in de test.

Geautomatiseerd rapporteren van de testresultaten

Na het uitvoeren van de test wordt automatisch een testrapport gegenereerd. Dit testrapport bevat een overzicht van de gehele test en detail rapporten van de geteste onderdelen (gebruikte communicatie methodes en de geteste commando's). Het totale eindresultaat van de test is alleen "succesvol" als de testen van alle individueel geteste commando's zijn geslaagd.

Een van de belangrijkste eisen van de test is dat deze volledig geautomatiseerd kan worden uitgevoerd. Dit betekend dat het Use-Case diagram van de test relatief eenvoudig is. De testengineer heeft de mogelijkheid om te test te configureren in de TestComplete software. Dit is beschreven in Use-Case *Configureer Test*, Tabel 2-1. Na de configuratie kan de test worden uitgevoerd. Dit is beschreven in Use-Case *Start test*, Tabel 2-2. Het resulterende Use-Case diagram voor de test is weergegeven in Figuur 2-1.

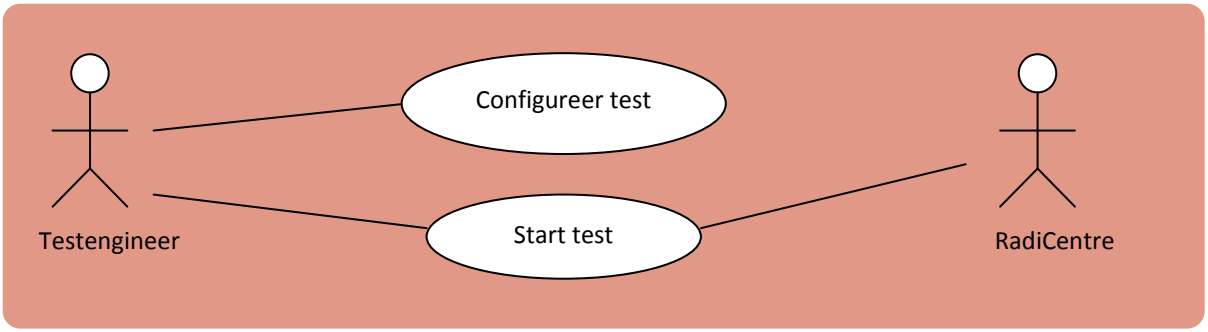
Tabel 2-1 **Use-Case *Configureer test***

Naam	Configureer test
Samenvatting	De testengineer geeft aan welke onderdelen wel en niet getest moeten worden
Actoren	Testengineer
Beschrijving	De testengineer geeft in de TestComplete software aan welke onderdelen van het RadiCentre getest moeten worden, bestaande uit: • Welke communicatiemethodes gebruikt moeten worden • Welke RadiCentre sloten (insteekkaarten) getest moeten worden • Eventueel welke externe hardware is aangesloten aan de RadiCentre producten (bijvoorbeeld meetprobes). Daarnaast kunnen er eventueel specifieke communicatie instellingen worden aangepast, zoals RS232 parameters, GPIB adres en IP adres
Resultaat	De test is geconfigureerd en gereed om te worden uitgevoerd

Tabel 2-2 **Use-Case *start test***

Naam	Start test
Samenvatting	De testengineer start de test
Actoren	Testengineer, RadiCentre (incl. aangesloten hardware
Beschrijving	De testengineer start de test. De test wordt geheel automatisch afgehandeld, gebaseerd op de opgegeven configuratie. Gedurende de test is er geen interactie met de testengineer. De test bestaat uit de volgende onderdelen: (1) Het openen communicatie met RadiCentre

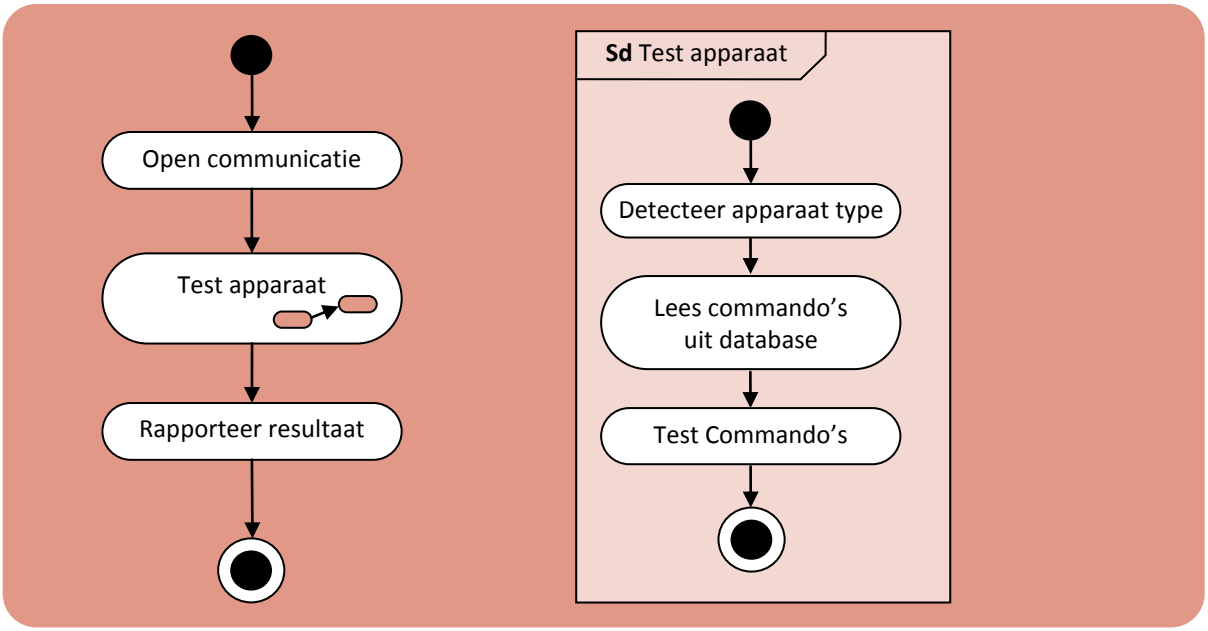
	(2) Het detecteren van aanwezige RadiCentre hardware (insteekkaarten) (3) Het uitlezen van alle beschikbare commando's uit de database (4) Het testen van alle commando's, bestaande uit het sturen van de commando's en het controleren van het verkregen antwoord (5) Het rapporteren van alle testresultaten
Resultaat	De test is uitgevoerd en een testrapport is gegenereerd



Figuur 2-1 **Use-Case diagram RadiCentre test**

2.2 Overzicht implementatie

De totale test bestaat uit een generieke testprocedure die voor alle individuele apparaten wordt uitgevoerd. Deze testprocedure bestaat uit een aantal kernactiviteiten. Deze verschillende activiteiten zijn weergegeven in Figuur 2-2**.**



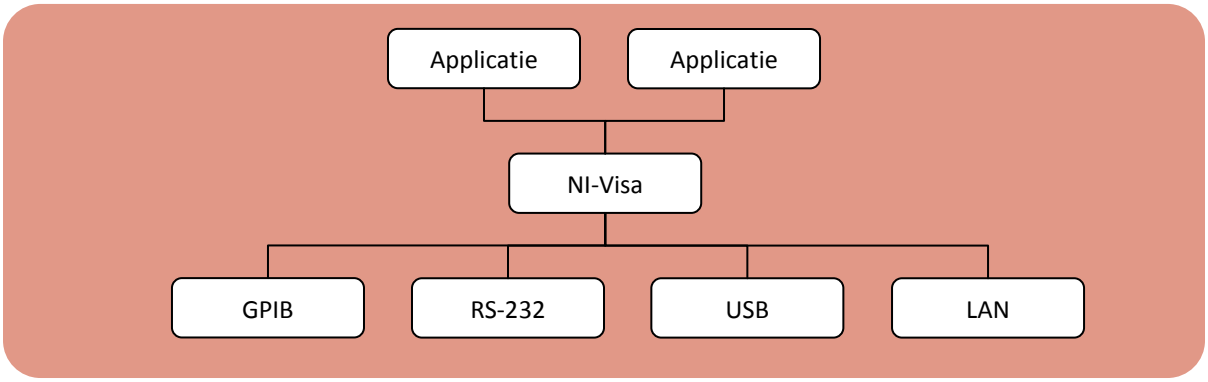
Figuur 2-2 **Activiteitsdiagram testprocedure**

Belangrijke basisonderdelen van de testprocedure zijn de communicatie en de commandodatabase. Deze onderdelen zullen eerst worden toegelicht in deze paragraaf. Vervolgens zal het ontwikkelde TestComplete programma worden besproken.

2.3 Communicatie met RadiCentre via National Instruments Visa

Voor de realisatie van deze communicatie is gebruik gemaakt van de softwarecomponent NI-Visa, ontwikkeld door National Instruments [8]. Deze softwarecomponent functioneert als extra laag tussen softwareapplicaties en de verschillende communicatiemethodes.

De NI-Visa softwarecomponent biedt een eenduidige interface voor de softwareapplicaties, waardoor de verschillende communicatieprotocollen op dezelfde manier kunnen worden aangestuurd vanuit de applicatie. Figuur 2-3 geeft hier een schematische weergave van.



Figuur 2-3 **Visa Interface als extra software laag tussen communicatie en applicatie**

Bij de Visa software van National Instruments wordt een dotNET component meegeleverd. Deze component kan gebruikt worden om een NI-Visa sessie te openen, en via deze sessie te communiceren met de aangesloten apparatuur.

De algemene procedure voor communicatie via de Visa laag is als volgt [9]:

Voor alle soorten communicatie moet eerst een sessie worden geopend via de Resource Manager. De Resource Manager zorgt voor het beheer van de sessies, onder andere om conflicten tussen verschillende sessies te voorkomen. NI-Visa ondersteunt verschillende soorten sessies. Voor communicatie met het RadiCentre is een zogeheten “Message Based” sessie nodig. Vanuit deze sessie kan vervolgens worden gecommuniceerd met het RadiCentre.

In de dotNET bibliotheek van NI-Visa is dit als volgt geïmplementeerd:

De Resource Manager is een member van de NI-Visa klasse en heeft een memberfunctie `open()` waarmee een nieuwe sessie kan worden geopend. In de deze functie wordt alleen de naam van de resource meegegeven. Deze `open()` functie retourneert een object van het algemene type `session`, welke is gedefinieerd in de NI-Visa dotNET bibliotheek.

Het volgende code voorbeeld laat zien hoe een sessie voor COM 1 geopend kan worden.

```
Set session = VisaNS.ResourceManager.GetLocalManager.Open("COM1")
```

De geretourneerde sessie is een object van het basis type `session`. Om de communicatie functies van de sessie te kunnen gebruiken, moet het object eerst worden geconverteerd naar een object van een specifiek type. Dit converteren moet met een handmatige typecasting functie worden gedaan.

Bij een nette object georiënteerde implementatie zou verwacht worden dat er op dusdanige manier gebruik gemaakt is van polymorfisme, dat handmatige typecasting niet nodig is. In dit geval zou dat

bijvoorbeeld betekenen dat bij het openen van een sessie kan worden aangegeven om wat voor soort sessie het gaat, zodat de `open()` functie direct het juiste type retourneert en er geen handmatige typecasting nodig is.

Tijdens ontwikkeling is gebleken dat het wel mogelijk is om direct een sessie te openen van een specifiek type, echter dit kan alleen buiten de resource manager om. De NI-Visa documentatie beschrijft echter altijd dat de resource manager moet worden gebruikt voor het openen van een sessie. Voor maximale betrouwbaarheid van een juiste werking is besloten om gebruik te maken van de generieke open functie en het basis type `session` object handmatig te converteren naar het juist specifieke type. Deze procedure van openen en handmatige typecasting wordt ook in een VB code voorbeeld van de NI-Visa documentatie gebruikt [10].

In Visual Basic kan een type conversie worden gedaan met de CType functie. Het volgende voorbeeld laat zien hoe een eerder verkregen `session` object geconverteerd kan worden naar een object van het type `MessageBasedSession`.

```
Set mbSession = CType(session, VisaNS.MessageBasedSession)
```

Het verkregen `MessageBasedSession` object kan vervolgens worden gebruikt voor de communicatie. In het volgende voorbeeld wordt het commando *IDN? Verstuurd via deze sessie

```
mbSession.Write("*IDN?")
```

2.3.1 Extra dotNET component voor benadering VISA laag

In het voorgaande voorbeeld is de typeconversie (Type Casting) vormgegeven met de functie `CType`. Dit is een specifieke functie van de taal Visual Basic en is niet beschikbaar in de scripttaal VBScript. In VBScript is ook geen andere vorm van type casting mogelijk.

Om het opzetten van een sessie via de hiervoor beschreven methode toch mogelijk te maken is besloten een speciaal dotNET component te ontwikkelen, die dient als extra laag tussen het script en de NI-Visa component.

Deze component is in de eerste plaats bedoeld voor het kunnen uitvoeren van de benodigde conversie. Er is besloten om daarnaast ook interfacefuncties te implementeren die gebruikt kunnen worden voor het opzetten en gebruiken van eenvoudige sessies. De conversie wordt door deze interfacefuncties worden uitgevoerd. De component dient op deze manier niet alleen als oplossing voor de ontbrekende conversie mogelijkheid, maar kan gebruikt worden als vervanger van het NI-Visa object. De communicatie wordt op deze manier nog steeds door één enkel object beheerd, wat ten goede komt aan de onderhoudbaarheid van de VBScript code.

Naast deze interface functies beschikt de component ook over een individuele functie voor het uitvoeren van de benodigde typeconversie. Mocht er in de toekomst bijvoorbeeld een geavanceerde Visa functie nodig zijn, dan kan de component eenvoudig als hulpobject worden gebruikt voor de conversie, en kan de geconverteerde sessie en de memberfuncties direct in de VBScript code worden benaderd. Hierdoor is het niet nodig de component aan te passen, en wordt de flexibiliteit vergroot.

Op basis hiervan zijn de volgende eisen voor de component opgesteld:

- De component moet beschikken over een vereenvoudigde interface voor de communicatie:

  - Generieke functies voor het openen en sluiten van sessies voor de verschillende NI-Visa resources
  - Generieke functies voor het lezen en schrijven via een geopende sessie
  - Specifieke functie voor het configureren van seriële communicatie sessies
- De daadwerkelijke NI-Visa sessie wordt door de component beheerd (als private datamember)
- De benodigde typeconversie wordt intern uitgevoerd wanneer de interface functies worden aangeroepen.
- De component moet onafhankelijk van de interfacefuncties gebruikt kunnen worden voor het converteren van een `session` object naar een `MessageBasedSession` object
- De component wordt als dotNET dll gecompileerd zodat deze via de CLR-Bridge functie van TestComplete kan worden toegevoegd. (zie hoofdstuk 1.3.6)

De component kan dus op twee verschillende manieren worden gebruikt. Voor beide mogelijkheden is een Use-Case opgesteld. In de eerste Use-Case (Tabel 2-3) wordt gebruik gemaakt van de vereenvoudigde interface functies. De ontwikkelde component vervangt het NI-Visa object in de VBScript code.

In de tweede case (

Tabel 2-4) wordt de ontwikkelde component alleen gebruikt voor de benodigde typeconversie van het sessie object. Het openen en beheren van de sessie wordt direct met de NI-Visa bibliotheek gedaan.

Tabel 2-3 **Use-Case 1: Communicatie met behulp van de interface functies**

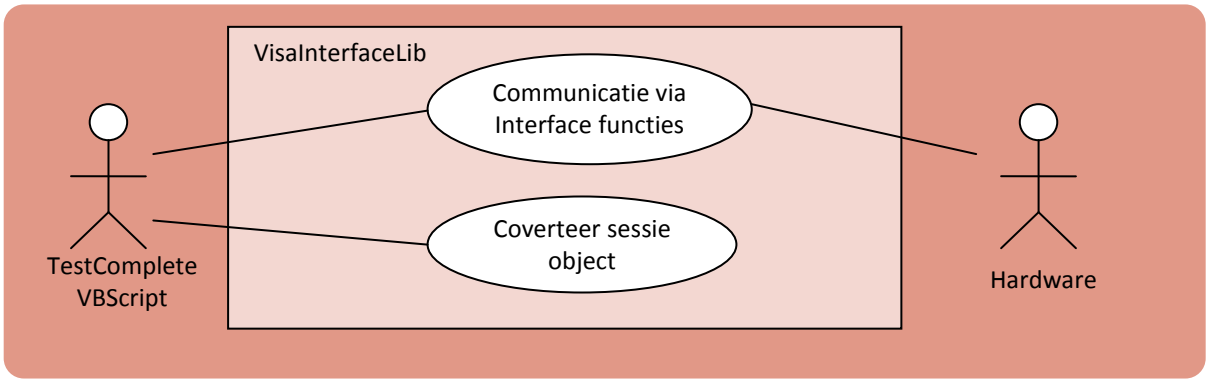
Naam	Sessie met vereenvoudigde interface
Samenvatting	Er wordt een sessie geopend en gebruikt met behulp van de toegevoegde interface functies
Actoren	TestComplete VBScript, Aangesloten hardware
Aannamen	De te openen resource is beschikbaar
Beschrijving	In het TestComplete VBScript wordt een object aangemaakt van de `VisaInterface` klasse. Met behulp van de interfacefuncties van dit object wordt een sessie geopend, gebruikt voor communicatie, en weer gesloten. Wanneer het om een seriële sessie gaat (zoals RS-232) kan ook de `ConfSerial` functie aangeroepen voor het instellen van de parameters.
Resultaat	De communicatie is uitgevoerd

Tabel 2-4 **Use-Case 2: VisaInterfaceLib als hulpobject voor de conversie**

Naam	VisaInterfaceLib als hulpobject voor conversie
Samenvatting	Met behulp van de NI-Visa bibliotheek is een sessie geopend. De VisaInterfaceLib wordt als hulpobject gebruikt voor de benodigde typeconversie.
Actoren	TestComplete VBScript

Aannamen	De te converteren sessie is succesvol geopend
Beschrijving	De te gebruiken sessie is direct via de NI-Visa bibliotheek geopend. Het resulterende algemene `session` object wordt geconverteerd met behulp van de `CastMBSession` functie van de `VisaInterface` klasse. De geconverteerde sessie kan vervolgens direct worden gebruikt voor de communicatie. In deze Use-Case wordt het openen, configureren en gebruiken van de sessie direct in het VBScript uitgevoerd. Deze acties vallen daarmee buiten de systeemgrenzen van de `VisaInterface` component.
Resultaat	De geopende sessie is geconverteerd en kan worden gebruikt voor communicatie

In Figuur 2-4 is het Use-Case diagram van de Use-Cases weergegeven. Voor dit diagram is de ontwikkelde `VisaInterface` component gekozen als systeemgrens. Het openen en gebruiken van een sessie gebeurd in de tweede Use-Case buiten de component om. Deze acties vallen daarmee buiten de gestelde systeemgrens. Ook is er voor de tweede Use-Case geen interactie met de hardware binnen de gestelde systeemgrens.



Figuur 2-4 **Use-Case 1: diagram voor het gebruik van de vereenvoudigde interface functies**

De dotNET component bestaat uit één klasse, de `VisaInterface`. Tabel 2-5 geeft een overzicht van alle public memberfuncties van de klasse.

Tabel 2-5 **Public memberfuncties van de VisaInterface klasse**

`Open`	Met deze functie kan een sessie worden geopend. De naam van de resource wordt meegeven als parameter. Wanneer het openen gelukt is wordt de geopende sessie geconverteerd en als private member opgeslagen. De functie retourneert een `boolean` om aan te geven of het openen gelukt is. Er worden geen exceptions gegenereerd. Hiervoor is bewust gekozen, omdat het mislukken van het openen niet per definitie een fout is. De resource kan 'legaal' bezet zijn. Deze functionaliteit is niet in de constructor opgenomen omdat VBScript geen parameters in de constructor accepteert[1].
`ConfSerial`	Met deze functie kunnen de parameters voor een seriële sessie (baudrate, aantal databits en aantal stopbits) worden aangepast. Deze functie kan worden gebruikt *na* het openen van een sessie, en is alleen van toepassing op seriële communicaties zoals RS-232. De functie hoeft alleen gebruikt te worden wanneer de parameters afwijken van de standaard Visa instellingen (geconfigureerd via de Visa software). Deze parameters zijn niet in de `Open` functie opgenomen, omdat VBScript geen

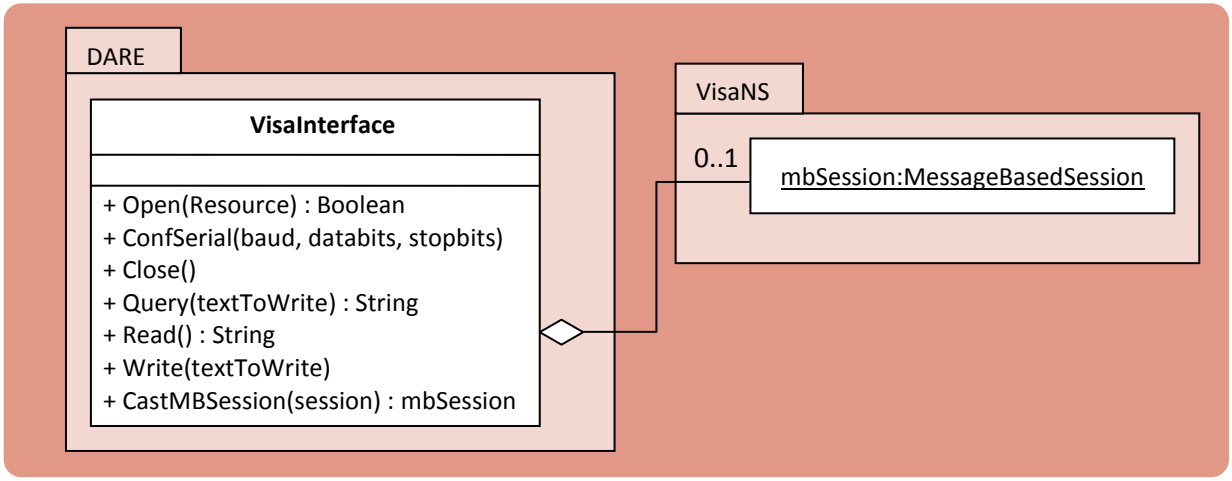
[1] Via de CLR-bridge functie van TestComplete is het wel mogelijk om geparametriseerde constructors aan te roepen, echter dit maakt de oplossing minder generiek.

	ondersteuning biedt voor *Function Name Overloading* of *optionele parameters*.
`Close`	Met deze functie wordt de geopende sessie (opgeslagen als private memberobject) gesloten en de bijbehorende resource weer vrijgegeven. Binnen een object van de klasse kan maximaal één sessie geopend worden. De close functie kan dus ook maar één sessie sluiten, en heeft daarom geen parameters nodig. Deze functie wordt ook altijd door de destructor van de klasse aangeroepen en hoeft dus niet altijd gebruikt te worden.
`Write`	Deze functie verstuurt het meegegeven commando naar de eerder geopende sessie. Er wordt een exception gegenereerd als het versturen mislukt.
`Read`	Deze functie leest het 'oudste' bericht uit welke door de aangesloten hardware is verstuurd. Als er geen antwoordt is ontvangen wordt een exception gegenereerd [2] .
`Query`	Deze functie is een combinatie van de hiervoor beschreven `Write` en `Read` functies. Het commando wordt verstuurd, en direct daarna wordt het antwoord uitgelezen. Geen antwoord resulteert in een exception [2].
`CastMBSession`	Met deze functie kan een generiek `session` object worden geconverteerd. Het `session` object kan als parameter worden meegegeven. De functie retourneert het geconverteerde `MessageBasedSession` object. Deze functie is individueel te gebruiken, en heeft geen enkele relatie met de datamember en de andere interface functies.

Zoals in Tabel 2-5 is weergeven zijn zowel de `Query` als de `Write` en `Read` functies opgenomen. Wanneer er op een verstuurd commando direct een antwoord wordt gegeven is de `Query` de eenvoudigste functie. Echter niet in alle gevallen wordt er (direct) een antwoordt gegeven op een verstuurd commando. Daarom zijn naast de `Query` functie ook de `Write` en `Read` functies als aparte functies geïmplementeerd.
In Figuur 2-5 is het klasse diagram van de `VisaInterface` weergeven. Wanneer gebruik gemaakt wordt van de interface functies, wordt de daadwerkelijke NI-Visa sessie een private memberobject van de `VisaInterface` klasse.

De dotNET component kan aan het TestComplete project worden toegevoegd via de hiervoor beschreven CLR-Bridge functie (zie hoofdstuk 1.3.6). De component is dan in het script benaderbaar als member van het dotNET object. Een instantie van het object kan worden gemaakt met de automatisch gegenereerde memberfunctie `zctor`.

[2] Bij het uitlezen wordt de standaard NI-Visa time-out gebruikt, ingesteld in de NI-Visa software



Figuur 2-5 **Klasse als interface tussen VBScript en Visa component**

Het volgende code fragment geeft een voorbeeld voor het openen en gebruiken van een RS-232 sessie via de `VisaInterface` klasse, zoals beschreven in Use-Case 1.

```
Set ViInterface = dotNET.DARE.VisaInterface.zctor
ViInterface.Open "COM1"                  'open serielle sessie
ViInterface.ConfSerial 57600, 8, 1       'RS-232 parameters
ViInterface.Write "REBOOT"               'stuur reboot commando
ViInterface.Close                        'Sluit de sessie
```

De `VisaInterface` klasse kan ook als hulpobject worden gebruikt voor alleen het uitvoeren van de conversie. Voordeel hiervan is dat het geconverteerde `session` object direct benaderbaar is in de VBScript code, waardoor alle geavanceerde NI-Visa memberfuncties te gebruiken zijn.

Het onderstaande code voorbeeld laat zien hoe een sessie kan worden opgezet en gebruikt met de `VisaInterface` klasse als hulpobject, zoals beschreven in Use-Case 2.

```
Set NI = dotNET.NationInstruments_VisaNS
Set session = NI.ResourceManager.GetLocalManager.Open("GPIB::6")
'GPIB is geopend, nu converteren naar MessageBasedSession
Set viHulpObject = dotNET.DARE.VisaInterface.zctor
set mbSession = viHulpObject.CastMBSession(session)
'mbSession kan nu direct worden gebruikt voor alle NI-Visa functies
mbSession.Write("REBOOT")          'stuur reboot commando
'Geavanceerde Visa functies kunnen nu direct worden aangeroepen
mbSession.Dispose                  'sluit de sessie
```

2.4 Excel commando database als input data

Een belangrijke eis van de ontwikkelde test is dat deze gebruikt kan worden voor alle beschikbare RadiCentre apparatuur. Dit betekent dat de test voor ieder apparaat type een specifieke set aan commando's moet kunnen sturen en deze moeten kunnen controleren op antwoord en uitvoer. Om dit te kunnen realiseren is een commando database opgebouwd. Deze database bevat alle

commando's van de hardware die door de test moeten worden uitgevoerd. Daarnaast is in de database ook voor ieder commando aangegeven welk antwoord er verwacht wordt. De database wordt in de ontwikkelde TestComplete test gebruikt om te bepalen welke commando's er getest moeten worden voor de aangesloten hardware. Het bijbehorende verwachte antwoord uit de database wordt vervolgens gebruikt om te bepalen of het ontvangen antwoordt correct is.

De database dient als belangrijkste input van de test, en is bepalend voor de specifieke inhoud van de test. De test zelf kan op deze manier meer generiek en apparaat onafhankelijk worden opgebouwd.

2.4.1 Eisen

Tijdens ontwikkeling zijn een aantal belangrijke eisen aan de database gesteld, welke bepalend zijn voor de functionaliteit van de test. Hier volgt een overzicht van de eisen met bijbehorende toelichting:

Eenvoudig aanpasbaar en uitbreidbaar
Dare!! Instruments is continue bezig met zowel het ontwikkelen van nieuwe apparatuur als het uitbreiden van huidige apparatuur. De lijst met beschikbare commando's is daarmee dynamisch en moet eenvoudig kunnen worden uitgebreid of aangepast.

Volgorde van commando's in database is bepalend
De volgorde waarin de commando's zijn opgenomen in de database moet gehandhaafd worden bij de uitvoer van de test. Veel commando's voeren slechts een klein deel uit van een bepaalde functionaliteit. Dit betekend dat meerdere commando's nodig kunnen zijn om bepaalde functionaliteit volledig te kunnen testen. De volgorde van deze commando's is daarbij van belang. Daarnaast is het op deze manier ook mogelijk om zowel "instel" als "uitlees" commando's te kunnen testen. Zo kan eerst een specifieke parameter worden ingesteld, en kan *daarna* deze parameter (met een ander commando) worden uitgelezen. Bij het uitlezen is dan exact bekend wat het resultaat zou moeten zijn. Deze eis maakt het mogelijk om in de database (per apparaat) een complete test sequentie te maken, waarin verschillende functies in een bepaalde volgorde worden uitgevoerd.

Minimale uitvoertijd als optionele parameter
Voor ieder commando kan een minimale uitvoertijd in milliseconden worden opgegeven. Veel commando's hebben een bepaalde uitvoertijd. De minimale uitvoertijd zorgt ervoor dat de test het opgegeven aantal milliseconden wacht alvorens door te gaan met het volgende commando.

Minimale en maximale software versie als optionele parameter
Voor ieder commando kan het minimale en/of het maximale versienummer worden opgegeven. Het commando mag dan alleen worden uitgevoerd als de software versie van het te testen apparaat hoger is dan de minimale en lager is dan de maximale software versie. Beide parameters zijn optioneel, en zijn zowel tegelijk als individueel van elkaar te gebruiken. Dit maakt het mogelijk om specifieke functies alleen te testen vanaf, of tot een bepaalde software versie. Dit is wenselijke wanneer er nieuwe functionaliteit wordt geïntroduceerd, of juist wanneer een functie niet meer wordt ondersteund vanaf een bepaalde software versie.

Het verwachte antwoord als reguliere expressie
Voor een aantal commando's is het verwachte antwoord dynamisch. De opbouw van een antwoord is veelal wel bekend, maar bijvoorbeeld opgevraagde waardes kunnen onbekend zijn. Wanneer

bijvoorbeeld een interne temperatuur wordt opgevraagd kan niet exact worden bepaald welke antwoord er wordt geretourneerd. Het is echter wel bekend in welke vorm het antwoord wordt geretourneerd (bijvoorbeeld: “TEMP dd.dd”, waarin d staat voor een willekeurig getal tussen 0 en 9). Hiervoor moet gebruik gemaakt worden van reguliere expressies [11]. Een reguliere expressie is een veelgebruikte generieke methode waarmee de mogelijke opbouw van een bepaalde karakterreeks gedetailleerd kan worden beschreven. Op deze manier kan worden bepaald of een bepaalde karakterreeks voldoet aan de expressie. In dit geval kan dus worden gecontroleerd of het werkelijke antwoord voldoet aan het verwachte antwoord.

Deze methode maakt het mogelijk om ook antwoorden met een dynamisch element zoals waardes te controleren.

2.4.2 Implementatie

TestComplete biedt verschillende oplossingen voor het benaderen van data lijsten of tabellen. Zo kan er een dataverzameling worden opgebouwd met behulp van TestComplete. Het is ook mogelijk data vanuit een externe bron of programma te benaderen. Er is voor gekozen om de database als spreadsheet te implementeren met het programma Microsoft Excel. De redenen hiervoor zijn:

- Microsoft Excel is bij alle engineers beschikbaar en is algemeen bekend in het gebruik
- De zekerheid van beschikbaarheid van Microsoft Excel op langere termijn is hoog
- De database kan onafhankelijk van TestComplete worden gegenereerd en bijgewerkt
- De data is eenvoudig in tabelvorm te structuren, waardoor Microsoft Excel geschikt is

Dit maakt de implementatie van de commandodatabase generiek en eenvoudig uitbreidbaar en aanpasbaar.

Zoals hiervoor beschreven wordt de specifieke testfunctionaliteit bepaald door de commandodatabase. De hierin opgenomen commando’s en de volgorde hiervan bepalen de uiteindelijke uitvoer en kwaliteit van de test. Bij het opbouwen van deze database zijn daarom een aantal regels opgesteld. Deze regels zijn bedoeld om een eenduidige teststrategie te creëren voor de verschillende apparaten die zijn opgenomen in de database. Deze regels zijn als volgt:

- alle beschikbare commando’s worden ten minsten één keer getest, in alle mogelijke schrijfvormen. Als bijvoorbeeld meerdere grootheden of eenheden met een parameter kunnen worden meegegeven moeten alle grootheden of eenheden getest worden.
- Voor alle geparametriseerde commando’s worden alle grenswaardes getest (bijvoorbeeld: laagst en hoogst instelbare frequentie).
- Voor alle functies die beschikken over zowel een instel- als een opvraag commando, wordt altijd de uitvoer van het instelcommando direct gecontroleerd met het opvraagcommando.
- Alle geparametriseerde commando’s worden minimaal één keer met een verkeerde parameter getest, om te controleren of de juiste foutmelding wordt geretourneerd. Na het foutief instellen wordt gecontroleerd of de waarde ook daadwerkelijk NIET is ingesteld.

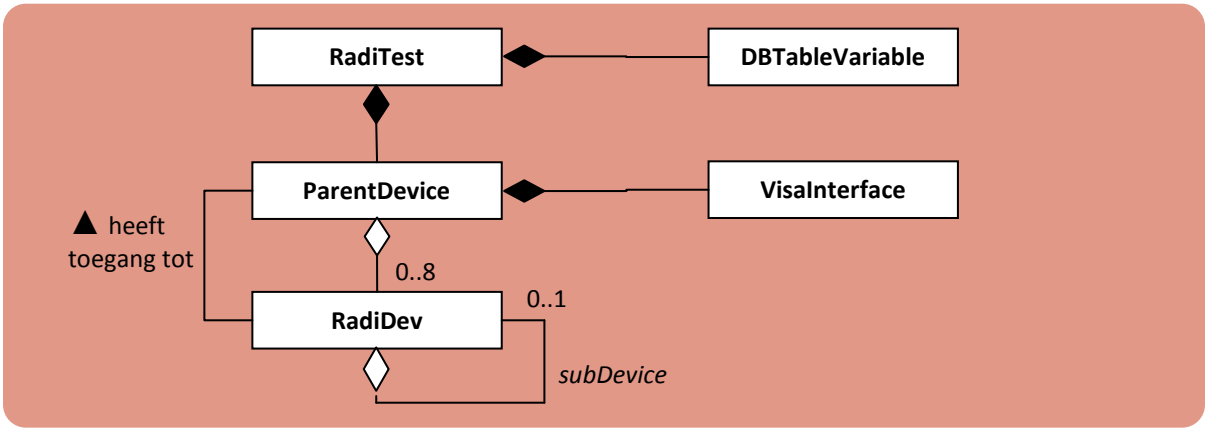
- Indien een apparaat of over een functie beschikt waarmee deze in een gedefinieerde staat komt (zoals een reset functie), worden altijd alle parameters opgevraagd om te controleren of deze op de standaard waarde staan, zoals vermeld in de producthandleiding.

2.5 Generiek Testprogramma

Het ontwikkelde TestComplete programma vormt het generieke onderdeel van de test. De kern van dit programma bestaat uit het uitvoeren van de verschillende acties, zoals beschreven in hoofdstuk 2.2. De specifieke uitvoer van het programma wordt bepaald door de aangesloten hardware en de gekoppelde commandodatabase. Voor deze structuur is bewust gekozen zodat de uitbreidingen of aanpassingen van de specifieke testfunctionaliteit gerealiseerd kunnen via de commandodatabase in Excel, zonder dat hiervoor het generieke testprogramma aangepast hoeft te worden.

Het programma is ontwikkeld in de taal VBScript en is zoveel mogelijk opgebouwd volgens een object georiënteerde structuur. Deze scripttaal is van nature geen object georiënteerde taal zoals C++ of VB.net. Er kunnen in VBScript echter wel klassen worden gedefinieerd die als objecten kunnen worden geïnstantieerd [12]. Dit maakt het wel mogelijk om een object georiënteerde programma structuur te ontwikkelen. De mogelijkheden zijn echter wel beperkt. Zo zijn geavanceerdere mechanismen als overerving en polymorfisme niet mogelijk in VBScript, waardoor het ontwikkelen van hiërarchische structuren onmogelijk is.

Voor deze test zijn een aantal verschillende klassen ontwikkeld, waarin specifieke taken en verantwoordelijkheden zoveel mogelijk gescheiden zijn. Dit ter bevordering van de onderhoudbaarheid en eventuele foutanalyse van het programma. Figuur 2-6 geeft een overzicht van de gebruikte klassen en de relatie hiertussen.



Figuur 2-6 **Eenvoudig klassediagram**

De `RadiTest` klasse is de hoofdsklasse van het programma. Deze klasse bevat de generieke testfunctionaliteit. Vanuit deze klasse wordt bepaald welke commando's er worden verstuurd en wat het testresultaat van ieder commando's is (wel of niet geslaagd). Daarnaast is deze klasse ook verantwoordelijk voor het rapporteren van de resultaten en het correct afhandelen van eventuele fouten. Bij de instantiëring van een `RadiTest` object wordt altijd een instantie van de `ParentDevice` klasse aangemaakt als private memberobject. Daarnaast wordt een private

referentie bijgehouden naar een `DBTabelVariable` object. Dit is een TestComplete object en wordt gebruikt voor het eenvoudig uitlezen van data uit een Excel bestand [13].

Een object van de `ParentDevice` klasse functioneert als manager voor het RadiCentre en de hierop aangesloten hardware. Deze klasse beheert de communicatie met de aangesloten hardware, en dient als container voor de verschillende `RadiDev` objecten.

Een `RadiDev` object dient als beheerder voor een individueel RadiCentre product. Dit kan zijn: het RadiCentre zelf, een RadiCentre insteekkaart of een extern apparaat dat is aangesloten via een insteekkaart. Een `RadiDev` object is verantwoordelijk voor de automatische hardware detectie en het beheer van de specifieke productinformatie (zoals typenaam, versienummer en hardwareadres). De communicatie met een specifiek apparaat verloopt altijd via het bijbehorende `RadiDev` object. Het `RadiDev` object zorgt vervolgens voor een afhandeling hiervan via het `ParentDevice` object.

Wanneer een `RadiDev` object gebruikt wordt voor een extern apparaat, is dit object een memberobject van een ander `RadiDev` object, namelijk het `RadiDev` object van de insteekkaart waarop de externe hardware is aangesloten.

2.5.1 RadiDev klasse

Zoals hiervoor beschreven kan de `RadiDev` klasse worden gebruikt voor het beheer van een individueel RadiCentre product. De `RadiDev` klasse wordt gebruikt voor alle apparaten die worden getest, inclusief het RadiCentre zelf en hardware die is aangesloten op een insteekkaart. De reden dat hiervoor dezelfde klasse wordt gebruikt, is omdat de manier van benaderen en de gebruikte eigenschappen (zoals naam, softwareversie) nagenoeg overeenkomen voor zowel RadiCentre, insteekkaarten en externe hardware. Door het ontbreken van overerving in VBScript is het niet mogelijk om een specifieke afleiding van een implementatie te maken, zonder dat hierbij veel codeduplicatie moet worden gebruikt.

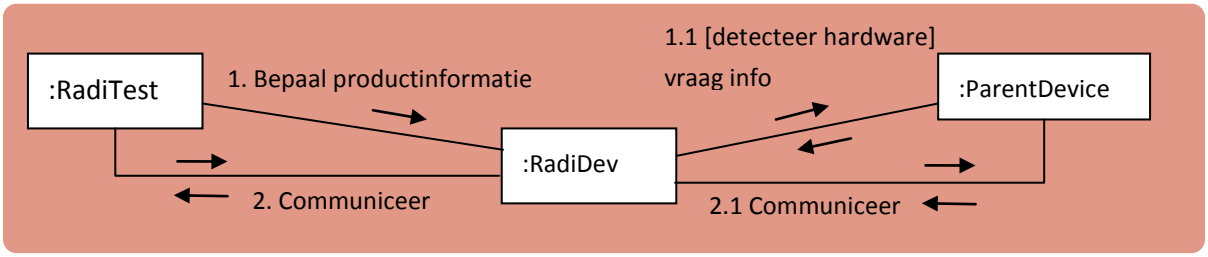
De `RadiDev` klasse wordt altijd gebruikt in combinatie met een `ParentDevice` object. Bij het aanmaken van een `RadiDev` instantie wordt deze altijd toegevoegd aan het `ParentDevice` object, zodat alle gebruikte `RadiDev` objecten via dit `ParentDevice` object kunnen worden benaderd. Ook wordt in ieder `RadiDev` object een referentie ingesteld naar het `ParentDevice` object, zodat alle toegevoegde `RadiDev` object gebruik kunnen maken van de (publieke) properties en methodes van het `ParentDevice` object. De `RadiDev` object kunnen op deze manier gebruik maken van de generieke communicatiefuncties van het `ParentDevice` object.

Het gebruik van de `RadiDev` klasse en de relatie met andere klassen is weergegeven in een communicatiediagram, Figuur 2-7.

Voordat de test wordt uitgevoerd, wordt eerst de productinformatie bepaald die bestaat uit typenaam, prefix (nodig voor communicatie), softwareversie en slotnummer. Dit proces wordt aangestuurd vanuit de `RadiTest` hoofdklasse en kan op twee manieren worden uitgevoerd. (1) De informatie wordt automatisch gedetecteerd door de `detectHardware()` functie van de `RadiDev` klasse. Deze functie wordt voor het RadiCentre en alle insteekkaarten worden gebruikt. (2) De productinformatie wordt expliciet gedefinieerd door de `RadiTest` hoofdklasse. Dit wordt gedaan voor alle extern aangesloten hardware, omdat de informatie hiervan niet kan worden

bepaald met een generiek commando. Bij veel externe hardware kan de informatie zelfs helemaal niet worden bepaald. De productinformatie kan worden opgegeven door de testengineer.

Wanneer de productinformatie is bepaald, kan het `RadiDev` object worden gebruikt voor het uitvoeren van de test. De `RadiTest` hoofdklasse bepaald welke commando's getest moeten worden en stuurt deze commando's naar het `RadiDev` object. Het `RadiDev` object voegt hier eventuele prefix informatie aan toe en stuurt het commando door naar het `ParentDevice` object. Het `ParentDevice` object zorgt voor verdere afhandeling van de communicatie.



Figuur 2-7 **Communicatiediagram `RadiDev` klasse**

De productinformatie wordt als private data opgeslagen in het `RadiDev` object. Deze informatie bestaat uit: de typenaam van het product, de actuele softwareversie van het product, het slotnummer van de insteekkaart (geldt ook voor externe hardware) en de hardwareprefix die nodig is voor de communicatie. De prefix bestaat uit het slotnummer, een eventuele letter voor het type hardware en een dubbelepunt. Voor het RadiCentre zelf wordt geen prefix gebruikt en als slotnummer wordt intern slotnummer 0 gebruikt. Door dit (niet bestaande) slotnummer te gebruiken kan het RadiCentre op dezelfde manier worden toegevoegd en gebruikt in de `ParentDevice` collectie, waardoor ook voor het RadiCentre de generieke implementatie kan worden gebruikt.

ParentDevice referentie als oplossing voor ontbrekende overerving

In de klasse wordt een referentie bijgehouden naar het `ParentDevice` object. Bij het ontwerp van de klassenstructuur is bewust gekozen voor het splitsen van verantwoordelijkheden. De `ParentDevice` klasse is verantwoordelijk voor de communicatie met de aangesloten hardware. De communicatiefuncties van de `RadiDev` klasse zijn alleen bedoeld voor een extra aanvulling op de bestaande communicatiefuncties van de `ParentDevice` klasse. Hiervoor zou een vorm van overerving wenselijk zijn. Dergelijke constructies zijn niet mogelijk met VBScript. Als oplossing voor dit specifieke probleem is het volgende geïmplementeerd: bij de initialisatie van een `RadiDev` object wordt een referentie van het `ParentDevice` object opgeslagen in het `RadiDev` object. Het `RadiDev` object kan daarmee de publieke memberfuncties van het `ParentDevice` object aanroepen, via deze referentie. Op deze manier wordt de generieke communicatiefunctionaliteit van de `ParentDevice` klasse hergebruikt en kan apparaatspecifieke functionaliteit worden toegevoegd in de daarvoor bestemde `RadiDev` klasse.

Extern aangesloten hardware als SubDevice

Op een aantal RadiCentre insteekkaarten kan externe hardware worden aangesloten, zoals een meetprobe. Een aantal van deze producten kunnen direct worden aangestuurd met een eigen commandoset. Om deze commando's te sturen is alleen een aangepaste prefix nodig. De `RadiDev` klasse kan dus ook voor deze hardware worden gebruikt. Er kan daarom aan een `RadiDev` object

een member worden toegevoegd van eveneens het type `RadiDev`. Het geïnstantieerde `RadiDev` object van de externe hardware kan op die manier als member worden toegevoegd aan het `RadiDev` object van de betreffende insteekkaart. De testprocedure kan via deze member eenvoudig controleren of en welke externe hardware is aangesloten op een insteekkaart. De `subDevice` referentie wordt alleen gebruikt bij insteekkaarten omdat dat de enige producten zijn waar externe hardware op kan worden aangesloten. In het `RadiDev` object van het subDevice zijn de `detecthardware` functie en de `subDevice` referentie dus feitelijk overbodig. (er is in de praktijk nooit een subDevice op een subDevice aangesloten). In deze situatie zou overerving dus wenselijk zijn, zodat er verschillende specifieke implementaties kunnen worden gebruikt voor insteekkaarten en externe hardware, die generiek kunnen worden gebruikt door de hoofdklasse (polymorfisme). Dit is echter niet mogelijk in VBScript. De enige manier om een specifieke implementatie te maken is door een geheel nieuwe klasse te definiëren. Voor dit geval zou dit echter zeer veel code duplicatie introduceren doordat de overige functionaliteit van de `RadiDev` klasse exact overeenkomt. Dit is onwenselijk, daarom is gekozen voor het hergebruik van de `RadiDev` klasse.

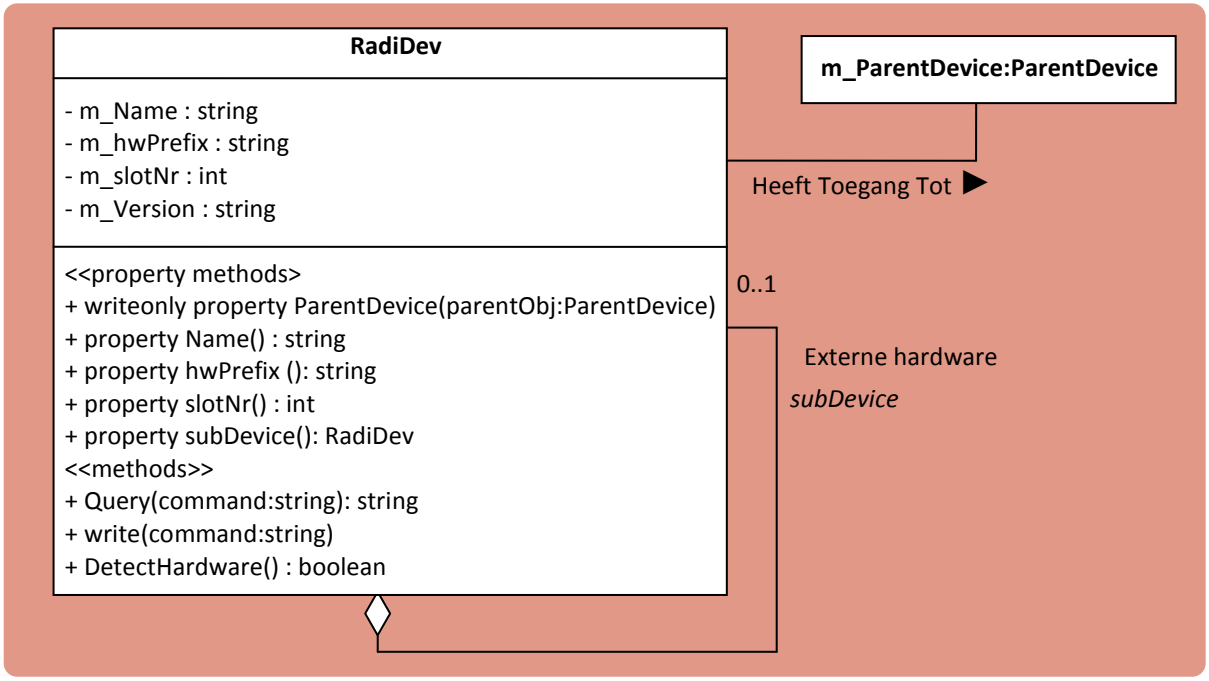
Alle memberfuncties en properties worden beschreven in Tabel 2-6**.** (Properties zijn speciale memberfuncties en worden op dezelfde manier benaderd als een public datamember. In werkelijkheid bestaat een property uit één of meerdere memberfuncties met dezelfde naam, met verschillende definities voor het lezen en schrijven van data.)

In Figuur 2-8 is een klassediagram weergegeven, specifiek voor de `RadiDev` klasse.

Tabel 2-6 Beschrijving van `RadiDev` memberfuncties en properties

Write-only Property ParentDevice	Deze property wordt gebruikt voor het instellen van de referentie naar het `ParentDevice` object. Deze referentie is voor intern gebruik, daarom is alleen de schrijfmogelijkheid van de property geïmplementeerd. Deze referentie moet altijd worden ingesteld. Omdat VBScript geen constructor parameters ondersteunt is deze functionaliteit als publieke property opgenomen. Er mag alleen een referentie naar een `ParentDevice` object worden opgegeven. Voor andere types wordt een error gegenereerd.
Property Name	Deze property wordt gebruikt voor het opvragen of handmatig instellen van de typenaam, welke leidend is voor de test. Bij het instellen wordt een error gegenereerd als er geen geldige `string` wordt opgegeven.
Property hwPrefix	Deze property wordt gebruikt voor zowel opvragen als instellen van de hardware prefix, gebruikt voor de communicatie. Bij het instellen wordt een error gegenereerd als er geen geldige `string` wordt opgegeven.
Property slotNr	Deze property wordt gebruikt voor het opvragen of instellen van het slotnummer. Bij het instellen van de property wordt gecontroleerd of de opgegeven waarde een getal is en of de waarde van het getal geldig is (dus groter of gelijk aan 0 en kleiner of gelijk aan het hoogste slotnummer, gedefinieerd als constante). Als de waarde ongeldig is wordt een error gegenereerd. Wanneer het slotnummer groter is dan 0 wordt een standaard prefix gegenereerd, bestaande uit het slotnummer en een dubbelepunt. Deze prefix kan later worden overschreven door de hardwaredetectie of de `hwPrefix` property.
Property Version	Deze property wordt gebruikt voor het opvragen of handmatig instellen van de software versie van het product. Bij het instellen wordt een error gegenereerd

	als er geen geldige `string` wordt opgegeven.
Property SubDevice	Deze property wordt gebruikt voor het opvragen of instellen van een referentie naar een ander `RadiDev` object. Deze constructie wordt gebruikt voor het toevoegen en testen van hardware die is aangesloten op het product waar de huidige instantie naar refereert. Deze “subdevice” constructie wordt alleen gebruikt voor hardware die is aangesloten op een insteekkaart.
Query(command)	De Query functie wordt gebruikt voor het versturen van een commando en het direct teruglezen van het antwoord. Deze implementatie roept de Query functie aan van de `ParentDevice` aan. Hierbij wordt de hardwareprefix toegevoegd aan het commando.
Write(command)	De `Write` functie wordt gebruikt voor alleen het versturen van een commando, zonder op antwoord te controleren. Deze implementatie roept de `Write` functie aan van de `ParentDevice`. Hierbij wordt de hardwareprefix toegevoegd aan het commando.
DetectHardware()	Deze functie wordt gebruikt voor het automatisch detecteren van de productinformatie. De functie stuurt het commando *IDN? en controleert welk antwoord wordt gestuurd. Zolang er “Error” wordt geretourneerd, worden alle mogelijke prefixen geprobeerd. Als het antwoord geen “error” bevat, wordt met een reguliere expressie de typenaam van het product en de softwareversie bepaald. De naam, softwareversie en laatst gebruikte prefix worden opgeslagen als datamember. Als met alle mogelijke prefixen “error” is geretourneerd wordt aangenomen dat er geen hardware is aangesloten op het betreffende slot. Geen hardware gevonden is geen foutconditie. Er wordt binnen deze functie alleen een error gegenereerd als er geen geldige naam of softwareversie kan worden bepaald uit een ‘geldig’ antwoord (een antwoord zonder “error”) De functie retourneert een `boolean` om aan te geven of er hardware is gevonden. Wanneer geen apparaat wordt gevonden, wordt geen error gegenereerd. Dit is omdat het niet vinden van een apparaat op dit niveau niet per definitie een foutconditie is.



Figuur 2-8 **Klassediagram `RadiDev` klasse**

2.5.2 ParentDevice klasse

De `ParentDevice` klasse is verantwoordelijk voor enerzijds het beheer van de communicatie via de `VisaInterface`, anderzijds voor het beheer van de verschillende `RadiDev` objecten. Dit zijn in de hoofdzaak twee afzonderlijke verantwoordelijkheden.

Voor de test is het wenselijk dat er één globaal object is voor het beheer van de communicatie met de verschillende RadiCentre producten. Op deze manier hoeft een sessie slechts eenmalig te worden opgezet en kan deze voor verschillende testen worden gebruikt. Dit zou opgelost kunnen worden door een `VisaInterface` object als globale variabele aan te maken in het testprogramma. Het nadeel hiervan is echter dat generieke functionaliteit steeds lokaal moet worden geïmplementeerd (zoals controleren of een sessie geopend is en het toevoegen of verwijderen van afsluit karakters).

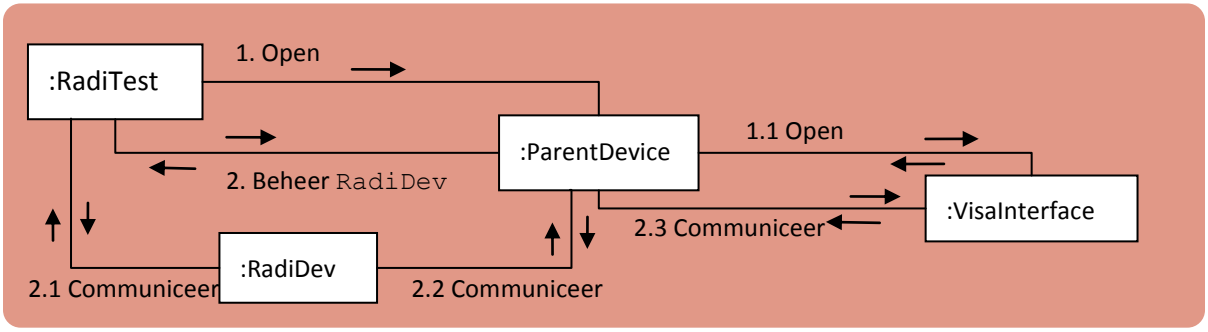
Naast de communicatie moeten ook de verschillende `RadiDev` objecten worden verzameld en kunnen worden benaderd door de generieke testklasse. Dit kan eenvoudig worden opgelost door deze objecten te verzamelen in een globale array. Generieke functionaliteit zoals het controleren van objecttypes of het aanvullen van objectinformatie moet echter altijd lokaal worden geïmplementeerd.

Deze twee verschillende wensen zijn gecombineerd geïmplementeerd in de `ParentDevice` klasse. De communicatiesessie wordt als private object in het `ParentDevice` object beheert. Alle communicatie moet daarom via de interfacefuncties van het `ParentDevice` object verlopen, wat het mogelijk maakt om generieke functionaliteit centraal te implementeren. De `RadiDev` objecten worden in dezelfde klasse beheerd. De `RadiDev` klasse is daarom zo ingericht dat deze gebruik maakt van een `ParentDevice` referentie voor de communicatie. In het `ParentDevice` object wordt er voor gezorgd dat voor alle toegevoegde `RadiDev` objecten de juiste referentie wordt ingesteld. Met deze oplossing is het zelfs mogelijk om meerdere sessies in verschillende `ParentDevice` objecten te beheren. De verschillende sessies en bijbehorende `RadiDev` objecten

kunnen dan volledig onafhankelijk van elkaar functioneren, ieder `RadiDev` object gebruikt altijd het goede `ParentDevice` object voor de communicatie.

Het gebruik van de `ParentDevice` klasse en directe relaties met andere klassen is weergegeven in een communicatiediagram, zie Figuur 2-9.

Vanuit de `RadiTest` hoofdklasse kan een sessie worden geopend. Het `ParentDevice` object zorgt voor de afhandeling hiervan en beheert de geopende sessie als private `VisaInterface` object. Daarnaast kunnen vanuit de `RadiTest` hoofdklasse `RadiDev` objecten worden toegevoegd en later direct worden benaderd voor de test. De testklasse kan hierdoor commando's direct naar het betreffende `RadiDev` object sturen. Verder afhandeling hiervan verloopt vervolgens via het `ParentDevice` object en de geopende sessie van het `VisaInterface` object. (zie hoofdstuk 2.3.1 voor gedetailleerde informatie over de `VisaInterface` klasse). De verschillende `RadiDev` objecten worden bijgehouden in een array, waarvan de lengte wordt vastgesteld bij instantiering van het `ParentDevice` object. Alle elementen van deze array worden geïnitialiseerd met een `Nothing` object, zodat de elementen altijd als object kunnen worden benaderd.



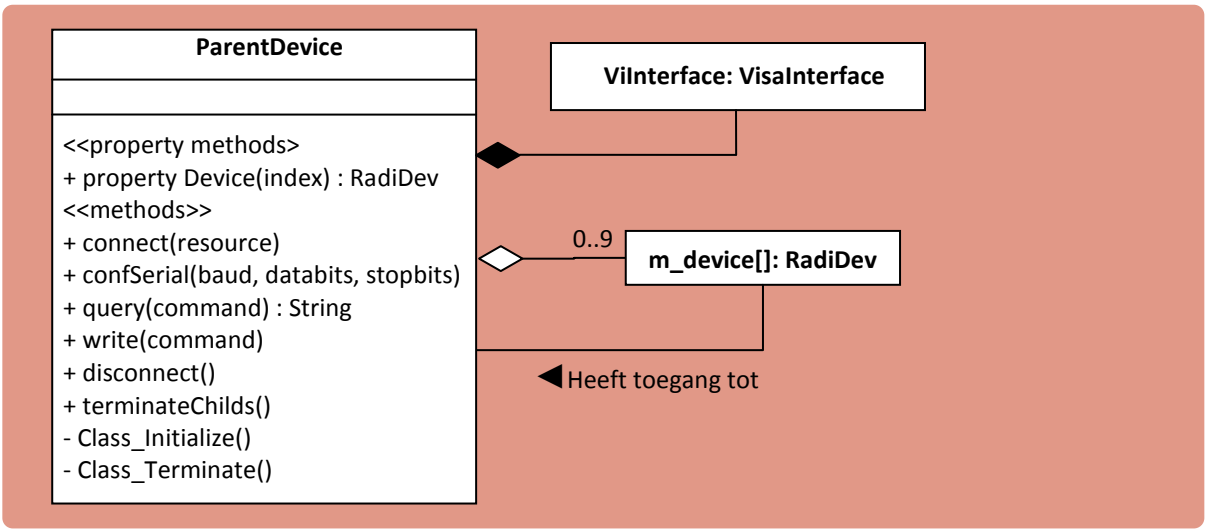
Figuur 2-9 **Communicatiediagram `ParentDevice` klasse**

Alle properties en memberfuncties van de klasse worden besproken in Tabel 2-7. Het klassediagram van de `ParentDevice` klasse is weergegeven in Figuur 2-10.

Tabel 2-7 **Properties en methodes van `ParentDevice` klasse**

Property device(index)	Deze property wordt gebruikt voor het toevoegen of benaderen van een `RadiDev` object. Als index wordt het slotnummer gebruikt. Bij het toevoegen van een `RadiDev` object wordt direct de `ParentDevice` referentie ingesteld voor het betreffende `RadiDev` object. Het toegevoegde object mag alleen een `RadiDev` of `Nothing` object zijn. Bij een ongeldig object type of een ongeldige index wordt een error gegenereerd.
Connect(resource)	Deze functie wordt gebruikt voor het openen van een communicatie sessie. Bij het uitvoeren van de test moet de resource beschikbaar zijn. Daarom wordt een error gegenereerd als het openen van de sessie mislukt.
ConfSerial(baud, databits, stopbits)	Deze functie wordt gebruikt voor het instellen van parameters voor seriële sessies. De functie controleert de geldigheid van de parameters, en roept daarna direct de `ConfSerial` functie van het `VisaInterface` object aan.
Query(command)	Deze functie voert de `Query` functie van het `VisaInterface` object uit, en voegt hier specifieke functionaliteit aan toe. Aan ieder commando wordt het new-line karakter toegevoegd, vereist voor de uitvoer van een RadiCentre commando. Bij het

	verkregen antwoordt wordt new-line karakter verwijderd. Het resultaat wordt retourneert. Wanneer een NI-Visa error is gedetecteerd, wordt één nieuwe poging gedaan, na een korte pauze van de uitvoer. Bij een tweede fout wordt een error gegenereerd. Ook wordt een error gegenereerd bij een ongeldige parameter of als het `VisaInterface` object ongeldig is.
Write(command)	Deze functie voert de `Write` functie van het `VisaInterface` object uit. Ook in deze functie wordt het new-line karakter aan het commando toegevoegd. Er wordt een error gegenereerd bij een ongeldige parameter of als het `VisaInterface` object ongeldig is.
Disconnect()	Deze functie wordt gebruikt voor het sluiten van een geopende sessie. De functie roept de Close functie van het `VisaInterface` memberobject aan.
TerminateChilds()	Deze functie verwijderd alle `RadiDev` memberobjecten en moet worden aangeroepen vóór het verwijderen van het `ParentDevice` object. De reden hiervoor is als volgt: Er is in VBScript geen onderscheid tussen objecten en referenties naar objecten. Elke objectvariabele is in werkelijkheid een referentie naar het betreffende object. Belangrijk hierbij is dat een object pas daadwerkelijk wordt opgeruimd als **alle** referenties naar dit object zijn opgeruimd. Wanneer dus in de hoofdklasse van dit programma de referentie naar het `ParentDevice` object wordt opgeruimd, dan bestaan er nog referenties naar ditzelfde object bij de `RadiDev` objecten. De objecten houde elkaar in stand: De `ParentDevice` refereert naar de `RadiDev` objecten, en de `RadiDev` objecten naar het `ParentDevice` object. Belangrijk is dus dat eerst alle `RadiDev` objecten worden opgeruimd. De oorspronkelijke variabele is dan weer de enige referentie, waardoor het object kan worden opgeruimd
Class_initialize	In de constructor wordt een instantie van de `VisaInterface` klasse toegekend als private member. Ook wordt het aantal elementen van de `RadiDev` array vastgesteld en worden alle elementen geïnitialiseerd met een `Nothing` object. Deze initialisatie wordt in de constructor gedaan om het in VBScript niet mogelijk is om een waarde of typedefinitie te bepalen bij de definitie van een variabele of memberobject. Het is wel mogelijk om de lengte van een array op te geven bij declaratie, echter daar kan dan geen constante voor worden gebruikt. De lengte van de array stelt het maximale aantal sloten voor en is voor eenvoudige aanpasbaarheid als constante gedefinieerd
Class_terminate	In de destructor wordt de `Disconnect` functie van de `VisaInterface` aangeroepen, om er zeker van te zijn dat de sessie altijd is gesloten na de uitvoer



Figuur 2-10 **Klassediagram ParentDevice**

2.5.3 RadiTest hoofdklasse

De generieke testfunctionaliteit van het programma is geïmplementeerd in de `RadiTest` klasse. Deze klasse vormt hiermee de hoofdklasse van het programma. Deze generieke functionaliteit had ook in globale functies geïmplementeerd kunnen worden. Het voordeel van het gebruik van een hoofdklasse is echter dat alle hoofdfunctionaliteit vanuit één centraal punt wordt uitgevoerd, namelijk het `RadiTest` object. Ook is er in de klasse gebruik gemaakt van private dataobjecten en private memberfuncties. Hierdoor wordt de interface van de klasse en dus het hoofdprogramma eenvoudiger, wat ten goede komt aan de leesbaarheid en onderhoudbaarheid van het programma.

De kernfunctionaliteiten en de relatie met andere objecten is weergegeven in het communicatiediagram van Figuur 2-11.

Het testprogramma wordt aangestuurd vanuit de verschillende hoofdroutines van TestComplete. Een belangrijk gegeven hierbij is dat ieder Test Item van een project is gekoppeld aan een globale routine (zie hoofdstuk 1.3.2 en bron [3]). Het hoogste niveau van de test bestaat hierdoor niet uit één hoofdroutine van waaruit het programma wordt uitgevoerd, maar uit een aantal geheel individuele routines die afzonderlijk worden uitgevoerd. Belangrijk hierbij is dat globale variabelen beschikbaar blijven voor na de uitvoer van een hoofdroutine.

Voor dit project wordt gebruik gemaakt van deze Test Items (gekoppeld aan globale routines) als gebruikersinput om aan te geven welke onderdelen (communicatiemethodes en slotnummers) getest moeten worden.

Aan het begin van een test wordt de klasse gebruikt voor initialisatie. De `ParentDevice` en `RadiDev` objecten worden geïnstantieerd en de communicatie wordt via het `ParentDevice` object geopend. Na de initialisatie wordt de klasse gebruikt voor het uitvoeren van de generieke test. Deze test omvat de gehele procedure voor het testen van een slot. In deze procedure wordt als eerste gedetecteerd welke hardware is aangesloten. Met deze informatie worden alle commando's voor het betreffende apparaat uitgelezen uit de commandodatabase. Alle commando's worden in de juiste volgorde naar de `RadiTest` klasse gestuurd. De verkregen antwoorden worden gecontroleerd en aan de hand daarvan worden de testresultaten bepaald en toegevoegd aan het testrapport.

Voor het rapporteren van de resultaten wordt gebruik gemaakt van de in TestComplete ingebouwde log functies (zie hoofdstuk 1.3.4). Bij iedere relevante actie kan eenvoudig een item aan de log worden toegevoegd met daarin de specifieke actuele informatie. Het rapporteren van de resultaten is daarmee geen aparte taak die aan het einde wordt uitgevoerd, maar is in de testprocedure verweven en wordt automatisch door TestComplete afgehandeld. Het rapporteren is duidelijk geen taak van de `ParentDevice` of `RadiDev` klassen, daarom is er wel bewust voor gekozen om alleen in de `RadiTest` klasse gebruik te maken van de log functies. Het gebruik van de log functies blijft daarmee overzichtelijk en eenvoudig aanpasbaar.

Error afhandeling

Een andere belangrijke verantwoordelijkheid van de `RadiTest` klasse is het afhandelen van fouten. Het afvangen van fouten gebeurt in de verschillende procedures van de `RadiTest` klasse. De `RadiTest` klasse heeft een generieke methode voor het afhandelen van onbekende fouten. Afhankelijk van het niveau van de fout wordt ofwel de uitvoer van de test gestopt ofwel een foutmelding aan het testrapport toegevoegd. Onbekende externe fouten (bijvoorbeeld van de

`VisaInterface` klasse of in Excel) hebben standaard het hoogste niveau, wat resulteert in het op een correcte manier afbreken van de test. Bij "eigen" fouten (gedefinieerd in één van de ontwikkelde klassen) wordt het niveau van de fout bepaald bij het genereren van de fout. VBScript heeft geen geavanceerd exception mechanisme zoals C++ of VB.NET. De enige manier om fouten af te vangen of te genereren is via het VBScript Error object. Bij dit object kan gebruik gemaakt worden van de error beschrijving en een (generiek of uniek) error nummer. In het hele testprogramma wordt bij het genereren van errors gebruik gemaakt van verschillende unieke nummers om het niveau van de error aan te geven. Het afvangen van errors kan in VBScript met de statements `On Error Resume Next` en `On Error Goto 0`. het volgende codefragment geeft een voorbeeld van hoe de errorchecker functie (binnen de `RadiTest` klasse) kan worden gebruikt.

```
On Error Resume Next        'vervolg lokale uitvoer wanneer error optreed
functieWaarErrorKanOptreden()     'voer een speciaal statement uit
ErrorChecker()                    'controleer of een error is gegenereerd
On Error Goto 0                   'stop met (lokaal) afvangen van errors
```

Het is met deze klasse ook mogelijk om extra referenties naar andere VBScript error objecten toe te voegen. De reden hiervoor is als volgt:

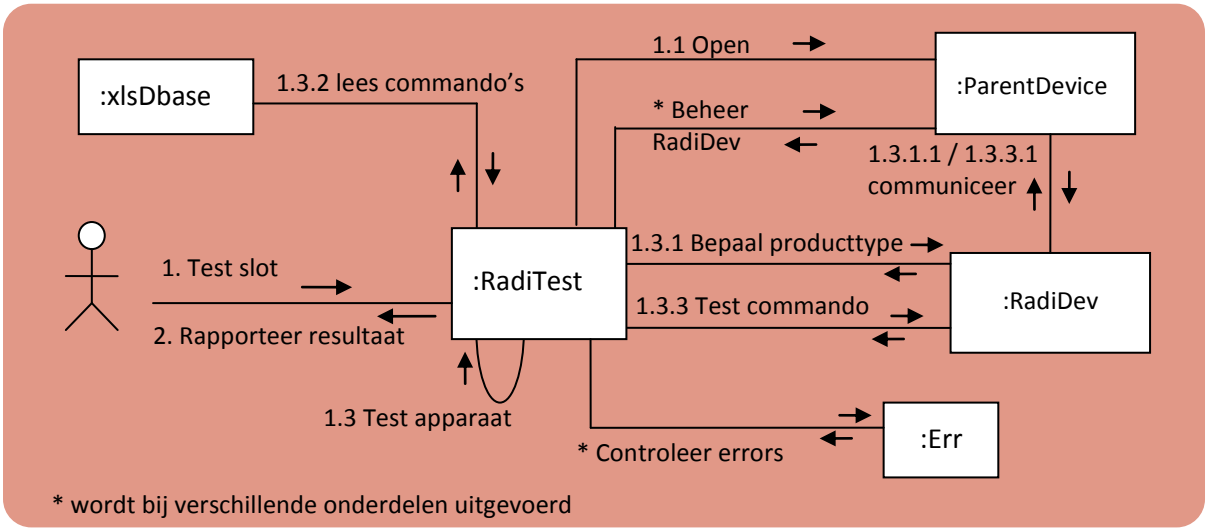
VBScript biedt geen ondersteuning voor het gebruiken van meerdere scriptbestanden. Het is dus niet mogelijk om een routine van een ander bestand aan te roepen. TestComplete heeft hier een speciale oplossing voor gemaakt. Met behulp van het UseUnit statement is het wel mogelijk om routines van een ander scriptbestand aan te roepen, zolang dit scriptbestand in het zelfde TestComplete project is opgenomen [14]. Deze functie heeft echter een aantal beperkingen.

1) het is alleen mogelijk *routines* van het andere bestand aan te roepen. Het is dus bijvoorbeeld niet mogelijk om met het keyword **`new`** een instantie van een klasse te maken welke is gedefinieerd in een ander scriptbestand.
2) Het globale VBScript error object is globaal voor de hele uitvoer, maar is lokaal per bestand. Wanneer dus in bestand B een error wordt gegenereerd, dan kan deze alleen worden uitgelezen met het globale error object van bestand B, niet van bestand A

De `RadiTest` klasse biedt daarom de mogelijkheid om een referentie naar een ander error object op te geven, zodat deze kan worden gebruikt bij de error afhandeling.

Om een object aan te kunnen maken van de klasse in een ander script bestand is een globale functie gemaakt naast de `RadiTest` klasse definitie. Deze functie retourneert een nieuwe instantie van de `RadiTest` klasse. Deze functie kan wel worden aangeroepen in een ander bestand, waardoor het mogelijk is om een nieuwe instantie van de klasse te verkrijgen in een ander bestand.

Van deze beperking is gebruikt gemaakt voor het error object. Aan de globale functie is een parameter toegevoegd waarin een referentie naar een error object moet worden meegegeven. Deze referentie wordt aan het nieuwe object toegevoegd voordat deze wordt geretourneerd. Voordeel van deze oplossing is dat er in een ander scriptbestand dus altijd een referentie naar het "lokale" error object moet worden meegegeven om een instantie van het `RadiTest` object te kunnen maken.



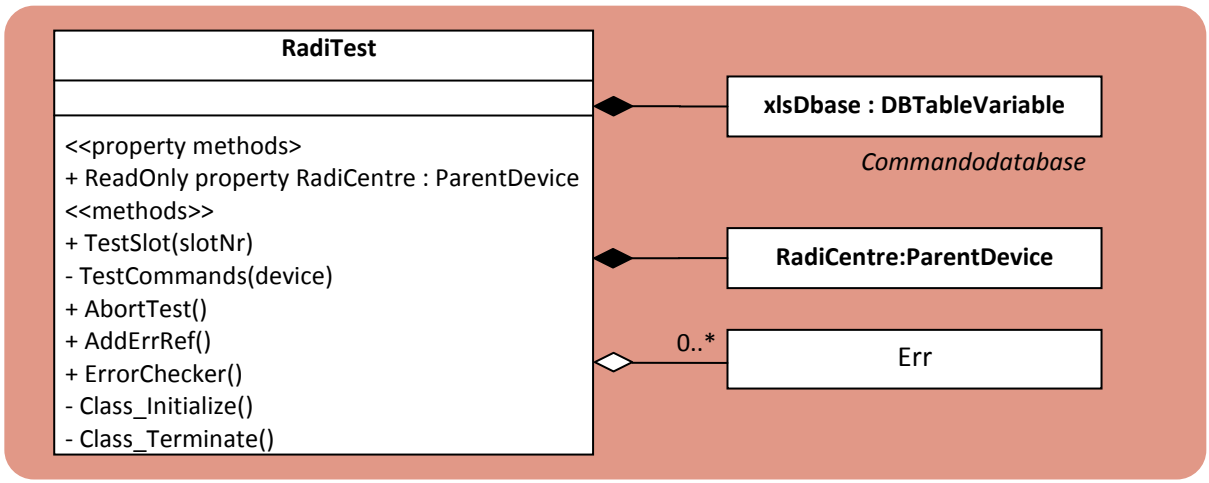
Figuur 2-11 **Communicatiediagram `RadiTest` klasse**

Tabel 2-8 geeft een overzicht van de memberfuncties en properties van de `RadiTest` klasse.

Tabel 2-8 **Memberfuncties en properties `RadiTest` klasse**

ReadOnly Property RadiCentre	Deze publieke readonly property geeft een referentie naar het `ParentDevice` memberobject. In het huidige programma wordt deze referentie gebruikt zodat er extern een `RadiDev` object kan worden toegevoegd aan een bestaand `RadiDev` object in de `ParentDevice` collectie. Dit wordt gebruikt voor het toevoegen van externe hardware. Het aanmaken van een `ParentDevice` objecten gebeurt alleen binnen de `RadiTest` hoofdklasse. Deze property kan daarom niet gebruikt worden voor het (over)schrijven van het `ParentDevice` object. Readonly geldt alleen voor de property zelf. Publieke write properties van het verkregen `ParentDevice` object kunnen gewoon worden gebruikt.
TestSlot(slotNr)	Deze functie wordt gebruikt voor het starten van een test voor een specifiek slot. De functie roept de `detectHardware` functie aan, van het betreffende `RadiDev` object. Als er hardware is gedetecteerd wordt het gevonden apparaat getest met de private `TestCommands` functie. Als het `RadiDev` object een subDevice heeft, wordt ook dit subDevice getest met dezelfde `testCommands` functie. Als geen hardware is gevonden door de `detecthardware` functie wordt dit als informatieve melding toegevoegd aan het TestComplete rapport
TestCommands(device)	Deze functie bevat de generieke testprocedure die voor ieder apparaat gebruikt kan worden voor het testen van alle beschikbare commando's. Deze procedure bestaat uit: (1) Het uitlezen van een commando uit de database waarvan het type en de versie overeenkomt met die van het betreffende apparaat (2) Het sturen van het commando en eventueel het lezen van het antwoord (3) Controleren of het verkregen antwoord overeenkomt met de reguliere expressie uit de commandodatabase (4) Het loggen van het testresultaat (5) Het wachten van de in de database gespecificeerde delay tijd (6) Herhalen van stap 1 t/m 5 zolang het einde van de database niet is bereikt
abortTest()	Deze functie wordt gebruikt voor het afronden van de test. In deze functie worden de `RadiDev` objecten en het `ParentDevice` object opgeruimd, en daarmee de communicatie resource vrijgegeven. Deze functie wordt altijd aangeroepen door de

	destructor van de `RadiTest` klasse en kan worden aangeroepen door de `errorchecker` functie, wanneer een kritieke error wordt afgehandeld en daarom de test wordt afgebroken.
AddErrRef()	Met deze functie kan een referentie naar een VBScript Error object worden toegevoegd aan de `RadiTest` klasse. Het toegevoegde error object wordt bewaard in een dynamische array, als private datamember van de klasse. Er is gebruik gemaakt van een dynamische array, zodat het aantal error objecten (ofwel het aantal scriptbestanden wat op errors gecontroleerd wordt) eenvoudig kan worden uitgebreid. De `errorchecker` functie kan eenvoudig alle elementen van de array langslopen om op errors te controleren.
ErrorChecker()	Met deze functie wordt gecontroleerd of er een error is opgetreden welke nog niet is afgehandeld. De functie controleert de errornummers van alle toegevoegde errorobjectreferenties welke zijn toegevoegd via de `AddErrRef`() functie. Wanneer één van deze objecten een error bevat wordt er "lokaal" een nieuwe error gegenereerd. Voor deze nieuwe error worden de gegevens van de originele error gekopieerd (nummer, beschrijving en bron). Hierna wordt het lokale error object gecontroleerd op errors. Wanneer een error is gevonden wordt de errorinformatie wordt toegevoegd aan het testrapport. Aan de hand van het errornummer wordt bepaald of de test al dan niet wordt afgebroken.
Class_initialize	Het aanmaken van een `RadiTest` object gebeurd altijd aan het begin van de test. De constructor van deze klasse bevat daarom een deel van de initialisatiefunctionaliteit. Er wordt een nieuw `ParentDevice` object aangemaakt als private member. Vervolgens wordt een nieuw `RadiDev` object toegevoegd aan alle elementen van de `ParentDevice` collectie. De reden hievoor is dat deze collectie na het aanmaken van een `RadiTest` object direct kunnen worden gebruikt om een subDevice aan toe te voegen, al voordat de hardware in het betreffende slot is gedetecteerd.
Class_terminate	In de destructor van de klasse wordt de `abortTest` functie aangeroepen. Op deze manier kan altijd worden gegarandeerd dat de memberobjecten zijn opgeruimd en belangrijke resources (zoals de Visa communicatie sessie) zijn vrijgegeven.



Figuur 2-12 **klassediagram `RadiTest` klasse**

2.5.4 Hoofdroutines

Het testprogramma wordt op het hoogste niveau aangestuurd door de hoofdroutines welke zijn gekoppeld aan de Test Items van het project (zie hoofdstuk 1.3.2). Vanuit deze routine worden de `RadiTest` klasse en de bijbehorende `ParentDevice` klasse en `RadiDev` klassen geïnstantieerd. Deze hoofdroutines zijn direct gekoppeld aan de Test Items van het project. Zoals eerder beschreven wordt de test niet aangestuurd vanuit één hoofdroutine, maar worden de verschillende hoofdroutines individueel uitgevoerd. Globale objecten en variabelen blijven wel behouden tussen de uitvoer van de verschillende routines. Daarom is ervoor gekozen om de instantie van het `RadiTest` object globaal te definiëren. De verschillende routines kunnen hierdoor allemaal gebruik maken van hetzelfde object, waardoor deze niet voor ieder testitem opnieuw hoeft te worden geïnstantieerd en geïnitialiseerd (wat ongunstig zou zijn voor bijvoorbeeld het beheer van de communicatiesessie).

Het project bevat vier verschillende hoofdroutines. De uitvoer hiervan wordt bepaald door de parameters, ingesteld in de gekoppelde Test Items. De routines worden beschreven in Tabel 2-9.

Tabel 2-9 Overzicht van de verschillende hoofdroutines, van waaruit de test wordt aangestuurd

configureSubDevice(slot, devName, hwPrefix)	Deze functie wordt gebruikt voor het “handmatig” specificeren van de aangesloten externe hardware. Met de paramters wordt opgegeven op welke slot de hardware is aangesloten, wat de typenaam is (de naam die in de commandodatabase wordt gebruikt) en welke hwPrefix moet worden gebruikt voor de communicatie. De procedure slaat de informatie op in een nieuw `RadiDev` object. Dit `RadiDev` object wordt in een globale array opgeslagen. De TestSlot procedure gebruikt deze globale array om het `RadiDev` object daadwerkelijk toe te voegen aan de `ParentDevice` instantie van het `RadiTest` object.
initConnection(source)	Deze functie is wordt gebruikt voor de initialisatie van een test. In het gekoppelde test item kan de naam van de communicatie resource worden opgegeven. Deze functie instantieert het globale `RadiTest` object en opent een sessie voor de opgegeven resource.
testDevice(slotNr)	Deze functie wordt gebruikt voor het starten van de test voor één specifiek slot. Deze functie is gekoppeld aan de testitems waarmee kan worden geselecteerd welke sloten wel en niet getest moeten worden. Bij de aanroep van deze functie is al een `RadiTest` object geïnitialiseerd en een sessie geopend. (dit wordt gedaan door het bovenliggende testitem, waarmee de initConnection fucntie wordt aangeroepen).
Finalize()	Deze functie is gekoppeld aan de verplichte Finalize test items. Hiermee wordt de geopende communicatiesessie gesloten en het globale `RadiTest` object opgeruimd.

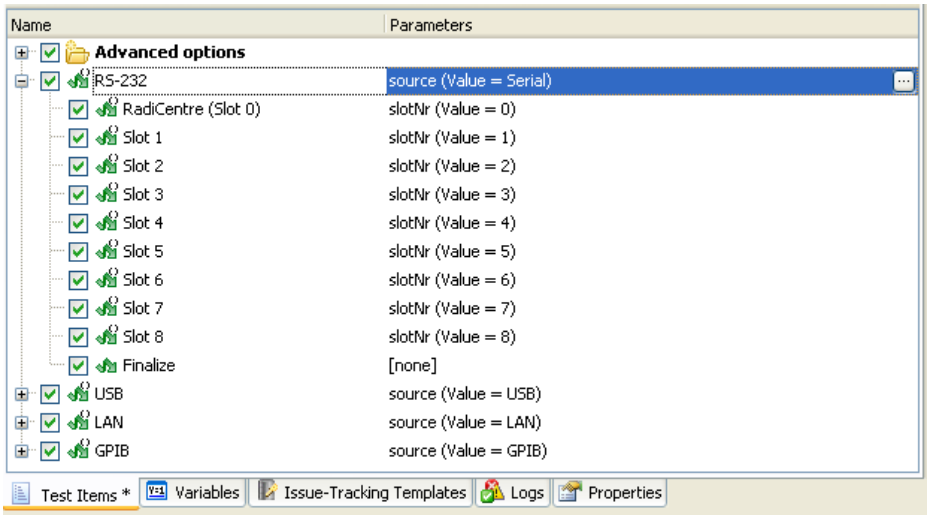
2.6 Gebruikersconfiguratie van de test

De testengineer heeft de mogelijkheid om de test te configureren voordat deze wordt gestart. Voor de implementatie hiervan is gebruikt gemaakt van de standaard testcomplete project configuratie mogelijkheden. De interface is hiermee eenvoudig aanpasbaar en uitbreidbaar, zonder dat hiervoor direct het testprogramma moet worden herontwikkeld.

Op de projectpagina kunnen Test Items worden geselecteerd voor de test. Deze Test Items zijn hiërarchisch gestructureerd en kunnen gebruikt worden om aan te geven:

- welke communicatie methodes getest moeten worden
- welke insteekkaarten (slot nummers) er moeten worden getest
- of en welke externe hardware is aangesloten op een specifiek slot. Het type hardware moet hier handmatig worden opgegeven via de parameters van het Test Item

In Figuur 2-13 is een weergave van de projectpagina te zien waarin de Test Items kunnen worden geselecteerd.



Figuur 2-13 **Voorbeeld van de projectpagina waarop Test Items geselecteerd kunnen worden**

Als geavanceerde optie is het mogelijk om aan te geven welke externe hardware er moet worden getest. De test engineer kan hier per slot handmatig invoeren welke hardware er is aangesloten.

Naast de Test Items kunnen ook geavanceerde parameters worden opgegeven voor de test. Deze zijn geïmplementeerd als projectvariabelen. Deze projectvariabelen zijn net als de Test Items te configureren via de projectpagina. Configureerbare parameters voor dit project zijn: GPIB adres, RSR232 parameters (baudrate, aantal start- en stopbits en poortnummer), USB poortnummer en IP adres.

Er is gekozen om gebruik te maken van deze Test Items omdat dit de onderhoudbaarheid en uitbreidbaarheid van het programma sterk vergroot. Iedere gebruiker die bekend is met TestComplete is in staat om de volgorde van de Test Items aan te passen, of Test Items uit te breiden zonder dat hiervoor aanpassingen gemaakt hoeven te worden in de code. Een alternatieve oplossing zou kunnen zijn om een gebruikersinterface te ontwikkelen (eveneens mogelijk met behulp van TestComplete). Voordeel is dat er dan meer mogelijkheden zijn voor wat betreft de structuur van het hoofdprogramma. Er zou dan een hoofdroutine gemaakt kunnen worden voor de uitvoer van het gehele programma, in plaats van dat gebruik gemaakt moet worden van verschillende individuele deelroutines die gekoppeld zijn aan een Test Item. Nadeel hiervan is echter dat een dergelijke interface minder eenvoudig aanpasbaar is. Uitbreiding of aanpassing van de te testen onderdelen moet altijd door een ontwikkelaar met ruime TestComplete kennis en ervaring worden gerealiseerd.

2.7 Testresultaten

Tijdens de afstudeerperiode zijn verschillende belangrijke testresultaten gegenereerd met behulp van de ontwikkelde testmethode. In de eerste plaats is dit geweest tijdens de ontwikkeling van de test, in het bijzonder bij de opbouw van de commandodatabase. Bij het opbouwen van de commandodatabase zijn steeds de gebruikershandleidingen van de betreffende producten gebruikt om te bepalen welke functionaliteiten en commando's getest moeten worden.

Gedurende de afstudeerperiode is de ontwikkelde test in gebruik genomen door de Dare!! Instruments engineers. De test is hierbij gebruikt voor zowel productietest, als laatste controle voor uitlevering van de producten, als voor verificatie methode voor het testen van nieuwe apparaat software.

Een overzicht van de testresultaten is weergegeven in Tabel 2-10.

Deze resultaten laten zien dat de implementatie van de test is geslaagd en het beoogde doel is bereikt. Daarnaast laten deze testresultaten ook het belang en de meerwaarde van dergelijke testmethodes zien. Ontwikkelingsfouten worden eenvoudiger ontdekt en kunnen worden opgelost voor het vrijgeven van de software. Productiefouten worden op tijd ontdekt en kunnen worden opgelost voordat het product wordt uitgeleverd. Problemen bij de klant worden hiermee voorkomen.

Tabel 2-10 **Overzicht testresultaten**

Probleem gevonden tijdens:	Omschrijving:
Opbouwen commandodatabase	Bij het opbouwen van de commandodatabase zijn verschillende onbekende problemen ontdekt. Zo bleek voor verschillende producten dat bij enkele commando's een ongeldige parameterwaarde kan worden ingesteld, zonder dat hierop een foutmelding wordt gegeven. Ook zijn er een aantal commando's ontdekt waar een ongeldig antwoordt wordt geretourneerd, of waar teveel antwoorden worden gestuurd op een vraagcommando. Alle gevonden problemen zijn gemeld bij de betreffende engineers.
Productietest	Na de ontwikkeling van de test zijn verschillende geassembleerde RadiCentre's getest, als laatste controle voor het uitleveren van producten. Hierbij zijn een aantal problemen afgevangen. Zo bleek bij enkele RadiCentre's een configuratie verkeerd te zijn ingesteld waardoor de communicatie niet naar behoren werkte. In een ander geval bleken de RadiCentre's te beschikken over een oudere softwareversie, waarin een specifiek probleem nog niet was opgelost. In beide gevallen konden de problemen eenvoudig worden opgelost en konden de producten alsnog worden uitgeleverd
Testen nieuwe softwareversie	De testmethode is verschillende keren gebruikt voor het testen van nieuwe softwareversies voor specifieke producten. Hierbij is de commandodatabase uitgebreid met nieuwe of aangepaste commando's. Ook is in een aantal gevallen gebleken dat nog niet alle commando's de verwachte response retourneren. De betreffende engineer heeft de problemen kunnen oplossen voordat de nieuwe softwareversie werd vrijgegeven.

3 RadiMation software test (fase 2)

In de tweede fase van het afstudeertraject is een testmethode he ontwikkeld en uitgebreid voor het geautomatiseerd testen van de door Dare!! ontwikkelde RadiMation EMC Test software. Deze testmethode is eveneens ontwikkeld in het programma TestComplete. Hiervoor is gebruik gemaakt van een bestaand testprogramma, welke sterk verouderd was en niet meer bruikbaar voor de meest recente versie van de RadiMation software. De uiteindelijke einddoelen van deze testmethode zijn als volgt:

- het testen van alle beschikbare gebruikersinterface elementen
  hierbij worden eenvoudig alle elementen bediend en gecontroleerd of de uitvoer voldoet aan de verwachtingen. Het testen van alle elementen staat los van het testen van complete RadiMation procedures en functionaliteiten
- Het geautomatiseerd kunnen uitvoeren van alle mogelijke kalibraties, EMC metingen en EMC testen. Hierbij worden niet alle besturingselementen getest, maar wordt een gebruikelijke procedure voor het configureren en uitvoeren van kalibraties en metingen getest. Belangrijkste testcriteria hierin zijn voornamelijk de benodigde tijdsduur voor iedere test en het controleren op bijvoorbeeld onverwachte vensters.

Het realiseren van deze doelen voor alle onderdelen van RadiMation is niet haalbaar in de betreffende afstudeerperiode. Daarom zijn een aantal specifieke eisen en wensen opgesteld, specifiek voor dit onderdeel van het afstudeertraject.

3.1 Functionele eisen

De specifieke eisen en wensen voor dit onderdeel van het afstudeertraject zijn weergegeven in Tabel 3-1. Deze eisen hebben een verschillende prioriteit, welke zijn weergegeven met behulp van de MoSCoW methode [15].

Tabel 3-1 **Doelen RadiMation software testproject**

Must	her ontwikkelen / aanpassen bestaande procedures zodat deze operationeel zijn voor de meest recente versie van RadiMation. De belangrijkste procedures zijn: • Het automatisch kunnen starten en stoppen van RadiMation • Het automatisch kunnen configureren van virtuele meet- en testapparatuur die gebruikt wordt kalibraties en EMC testen • Het automatisch kunnen uitvoeren van kalibraties en EMC Testen, met behulp van Virtuele meet- en testapparatuur • Het automatisch toevoegen van alle beschikbare RadiMation drivers en controleren of deze geconfigureerd kunnen worden
Should	• Uitbreiden van de test met een methode waarmee de tijdsduur van het uitvoeren van een kalibratie of EMC test automatisch wordt gemeten en opgeslagen in een Excel bestand • Uitbreiden van de test met een methode waarmee het mogelijk is om een tijdelijke configuratie aan te maken voor de RadiMation software die gebruikt wordt voor de TestComplete test. Na uitvoer van de test moet de originele RadiMation configuratie weer worden hersteld • Uitbreiden van de test met de mogelijkheid om de bestaande procedures voor het uitvoeren van EMC testen te kunnen gebruiken met fysiek aangesloten meetapparatuur (in plaats van

	virtuele apparatuur)
Could	• her ontwikkelen van bestaande procedures voor het uitbereid testen van alle beschikbare schermelementen (knoppen, tekstvelden enz.)
Won't	• uitbreiden van het TestComplete project met aparte procedures voor het testen van alle beschikbare gebruikersinterface elementen.

3.2 Overzicht testmethode

Het bestaande TestComplete project bestaat uit een zeer uitgebreid aantal procedures en algemene routines voor het testen van de RadiMation software. De test is ontwikkeld in de scripttaal VBScript. Hierin is in de bestaande test nauwelijks tot geen gebruik gemaakt van een object georiënteerde structuur. Wel wordt er in het bestaande programma veel gebruik gemaakt van generieke procedures voor bijvoorbeeld het vinden van specifieke elementen in een RadiMation venster. Voordeel hiervan is dat vaak alleen de generieke procedures aangepast hoeven te worden, en dat de procedure waarin specifieke testfunctionaliteit is beschreven voor een groot deel kan worden hergebruikt.

De bestaande testmethode kan worden verdeeld in een aantal kernactiviteiten, namelijk het:

- Starten en stoppen van RadiMation
- Configureren van RadiMation
- Configureren van de te gebruiken (virtuele) testapparatuur
- Configureren en uitvoeren van alle beschikbare kalibraties en EMC testen

3.3 Herzien bestaande testprocedures

Een groot deel van de tijd is besteed aan het herzien van de bestaande procedures. Deze zijn geschreven voor veel oudere versies van RadiMation. In de tussenliggende periode zijn een aantal belangrijke wijzigingen aangebracht in zowel RadiMation als TestComplete. Dit maakt dat veel procedures niet meer werken.

Het belangrijkste doel is het weer functioneel maken van de procedures voor de laatste versie van RadiMation. De nadruk ligt hierbij op de procedures die nodig zijn voor het geautomatiseerd configureren en uitvoeren van kalibraties en EMC testen.

Bij het herzien van deze procedures zijn veelal dezelfde stappen gevolgd:

Analyseren van de bestaande procedure
Het eerste doel is om de bestaande procedures bruikbaar te maken voor de laatste RadiMation versie. In de eerste instantie wordt dus uitgegaan van een hergebruik van de bestaande procedures. Allereerst moet daarom worden geanalyseerd wat de oorspronkelijke uitvoer van de procedure zou moeten zijn.

Analyseren en aanbrengen van de benodigde wijzigen in de bestaande procedure
Wanneer bekend is wat de uitvoer van de procedure moet zijn, wordt geanalyseerd wat er voor nodig is om de procedure bruikbaar te maken voor de laatste RadiMation versie. De functionaliteiten zijn

vaak in grote lijnen hetzelfde gebleven. Vaak zijn de gebruikte interfaceobjecten gewijzigd, of is functionaliteit verplaatst of uitgebreid.

Eventueel uitbreiden van de bestaande procedures
In een aantal gevallen is een specifieke RadiMation functionaliteit uitgebreid. Deze uitbreiding is opgenomen in de betreffende procedure wanneer deze uitbreiding een belangrijke rol speelt in de betreffende functionaliteit. Voor het bepalen van het belang hiervan is gebruik gemaakt van de gestelde prioriteiten (zie paragraaf 3.1)

Eventueel vereenvoudigen en / of optimaliseren van bestaande procedures
In een aantal gevallen is in de bestaande procedure gebruik gemaakt van complexe methodes voor het besturen van interfaceobjecten. Reden hiervoor is vaak omdat de voorheen gebruikte objecten lastiger controleerbaar waren, of minder goed ondersteund door de toenmalige gebruikte versie van TestComplete. In dergelijke gevallen zijn de procedures vereenvoudigd wat vaak de onderhoudbaarheid en de herbruikbaarheid vergroot of de benodigde uitvoertijd verkleind.

3.3.1 Generieke procedure voor het starten van kalibraties of EMC testen

Eén van de belangrijkste bestaande procedures welke binnen dit project zijn her ontwikkeld, zijn de procedures voor het uitvoeren van de verschillende kalibraties en EMC testen. De kern van deze procedures bestaat uit het laden van een specifiek configuratiebestand, het starten van de kalibratie of test en het wachten tot de test voltooid is. De manier waarop dit moet worden uitgevoerd komt in grote lijnen overeen voor de verschillende kalibraties en testen. Een aantal details verschillen echter per test. Om die reden waren er in het bestaande testprogramma verschillende routines opgenomen voor het uitvoeren van deze verschillende testen.

Bij de herontwikkeling van deze procedures is besloten om één hoofdroutine te ontwikkelen die kan worden gebruikt als generieke functie voor het uitvoeren van de test. Het doel hiervan is om een de onderhoudbaarheid en de uitbreidbaarheid te verbeteren, door een duidelijkere en eenvoudigere structuur in de verschillende routines aan te brengen. Een andere belangrijke aanleiding voor de ontwikkeling van één hoofdroutine voor alle verschillende testen, is de wens om timing gegevens van alle uitgevoerde kalibraties en testen te kunnen meten en exporteren. Wanneer met één hoofdroutine wordt gewerkt hoeft deze functionaliteit ook maar eenmalig te worden geïmplementeerd. Ook dit komt ten goede aan de uitbreidbaarheid van het programma. Wanneer in de toekomst de test wordt uitgebreid kan van dezelfde hoofdroutine gebruik worden gemaakt, en hoeft hierbij de timing functionaliteit niet opnieuw worden toegevoegd.

3.4 Geïmplementeerde uitbreidingen

Na het herzien van de bestaande procedures zijn een aantal uitbreidingen aangebracht in het TestComplete project. In dit hoofdstuk worden de gemaakte implementaties van de belangrijkste uitbreidingen toegelicht.

3.4.1 Aanmaken tijdelijke RadiMation configuratie

Met de huidige testmethode kunnen alle kalibraties en EMC testen worden uitgevoerd, gebruik makend van virtuele meetapparatuur (deze virtuele meetapparatuur is standaard beschikbaar in

RadiMation). Eén van de belangrijkste wensen voor het testproject is dat dezelfde testen ook uitgevoerd kunnen worden met fysiek aangesloten meetapparatuur (zie paragraaf 3.1).

In de praktijk zal dit betekenen dat de test zal worden uitgevoerd in een bestaande installatie, namelijk die van een testlaboratorium. In dit geval is het uiteraard wenselijk dat de bestaande RadiMation configuratie ongewijzigd blijft na de uitvoer van de test. Als voorbereiding hierop is daarom een procedure ontwikkeld waarmee de bestaande configuratie wordt gekopieerd naar een tijdelijke locatie, welke gebruikt kan worden voor de TestComplete test. Na uitvoer van de test kan dan de originele RadiMation configuratie weer worden hersteld.

RadiMation maakt gebruik van verschillende .ini bestanden voor het opslaan en uitlezen van de configuratie. De belangrijkste daarvan is het bestand radimation.ini. Dit bestand is staat standaard in de “application data” map van het “all users” profiel (een generieke Windows locatie voor het opslaan van specifiek applicatie data). In het radimation.ini bestand is de bestandslocatie gedefinieerd van de RadiMation configuratie map. De specifieke configuratie bestanden bevinden zich in deze map. De locatie van deze configuratie map kan wel gewijzigd worden.

De ontwikkelde procedure is als volgt:

- Er wordt een kopie van radimation.ini gemaakt, die kan worden hersteld na de test
- De huidige bestandlocatie van de configuratie map wordt uitgelezen uit radimation.ini, zodat deze map gekopieerd kan worden naar de tijdelijke locatie
- Er wordt een nieuwe (tijdelijke) bestandslocatie van de configuratie map opgeslagen in radimation.ini zodat RadiMation deze (nog te maken) kopie van de configuratie zal gebruiken tijdens test
- De originele configuratie map wordt gekopieerd naar een tijdelijke locatie
- Het settings.ini bestand uit de gekopieerde configuratie map wordt geopend en alle bestandlocaties van de RadiMation (gebruikers) mappen worden uitgelezen
- Al deze gebruikersmappen worden gekopieerd naar een tijdelijke locatie
- De locaties van de gemaakte kopieën worden opgeslagen in het settings.ini bestand

Voor het lezen en schrijven van waardes uit een .ini bestand wordt een object gebruikt van de speciaal ontwikkelde generieke klasse `IniFile`. Deze klasse is beschreven in hoofdstuk 3.4.2.

Er wordt altijd gecontroleerd of er al een kopie bestaat van het radimation.ini bestand. Mocht de originele configuratie in een voorgaande test niet hersteld zijn, dan wordt hiermee voorkomen dat het originele radimation.ini bestand wordt overschreven met een door TestComplete aangepaste versie van het bestand.

De tijdelijke locatie waar alle mappen en bestanden naar gekopieerd worden is gedefinieerd in de projectvariabelen van het testproject. De tijdelijke locatie is hierdoor eenvoudig aanpasbaar (voor projectvariabelen, zie hoofdstuk 1.3.3)

Het uitlezen en kopiëren van de RadiMation gebruikersmappen wordt gedaan door altijd alle beschikbare items uit te lezen uit de “Dirs” sectie van het settings.ini bestand. Door alle items uit te lezen, zonder de itemnamen van het .ini bestand expliciet te definiëren, blijft ontwikkelde procedure

compatibel wanneer het aantal mappen zou worden uitgebreid in een toekomstige versie van RadiMation.

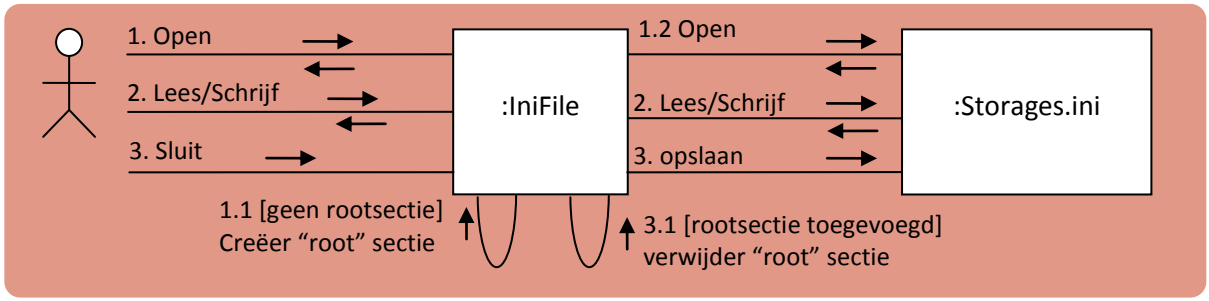
3.4.2 Generieke klasse voor werken met .ini bestanden

Het testproject is uitgebreid met een generieke klasse voor het lezen en schrijven van .ini bestanden. Een belangrijke vermelding hierbij dat deze oplossing gebruik maakt van TestComplete functies, en dus alleen in TestComplete projecten is te gebruiken.

Onder andere voor de in hoofdstuk 3.4.1 beschreven procedure is een oplossing nodig voor het vanuit script kunnen lezen van en schrijven naar .ini bestanden. TestComplete heeft een standaard ingebouwd object wat kan worden gebruikt voor het lezen van en schrijven naar .ini bestanden. Dit object heeft echter één belangrijke beperking. Om secties en items uit het .ini bestand te kunnen benaderen moet het betreffende .ini bestand een sectie bevatten met de naam “root”. (Voor meer informatie over dit object zie hoofdstuk 1.3.5)

Geen van de gebruikte .ini bestanden heeft standaard een “root” sectie. Daarom is een klasse ontwikkeld waarin een oplossing is geïmplementeerd voor dit probleem. Deze klasse voegt simpelweg een “root” sectie toe alvorens het bestand te openen met speciale de TestComplete functie. Bij het sluiten van het bestand wordt de “root” sectie weer verwijderd. De klasse dient hiermee als soort “wrapper” klasse voor het TestComplete `storages.ini` object, voor het werken met .ini bestanden. De oplossing had ook geïmplementeerd kunnen worden in een generieke functie. Er is echter bewust gekozen voor de implementatie in een klasse. Het voordeel hiervan is dat er publieke interface functies geïmplementeerd kunnen worden waardoor alleen de gebruikelijk open, lees/schrijf en sluit functies aangeroepen hoeven te worden. Ook kan doormiddel van private memberfuncties en de destructor gegarandeerd worden dat een eventueel toegevoegde “root” sectie altijd wordt verwijderd na het gebruik van een object van deze klasse.

Het gebruik van de klasse is weergegeven in het communicatiediagram van Figuur 3-1.



Figuur 3-1 **Communicatiediagram van de IniFile klasse**

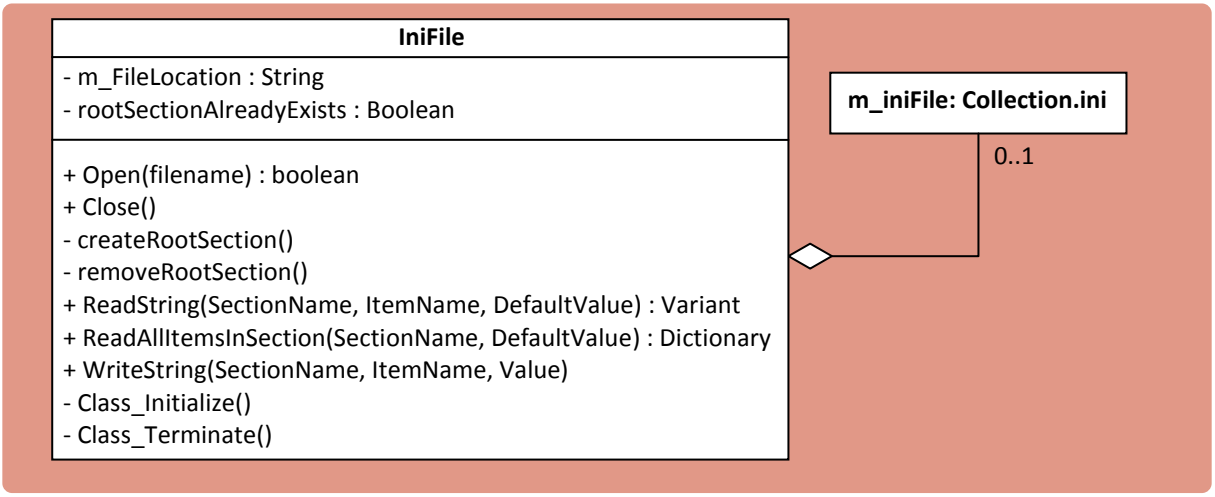
De verschillende memberfuncties zijn weergegeven in Tabel 3-2. Het bijbehorende klassediagram is weergegeven in Figuur 3-2.

Tabel 3-2 **Memberfuncties van de IniFile klasse**

Open(filename)	Met deze functie wordt een .ini bestand geopend. De functie controleert eerst of het betreffende bestand bestaat. Zo ja, dan wordt eerst de private `CreateRootSection` aangeroepen waarna het bestand wordt geopend met het `storage.ini` object. De functie retourneert een boolean om aan te geven

	of het openen gelukt is.
CreateRootSection()	Deze functie opent het betreffende .ini bestand als tekstbestand. Vervolgens controleert de functie of er al een root sectie bestaat, door te zoeken naar de tekst "[root]". Als deze tekst niet is gevonden wordt deze als nieuwe lijn toegevoegd. Hierna wordt het als tekstbestand geopende bestand gesloten. Er wordt een private member bijgehouden om aan te geven of de rootsectie al bestond voor het toevoegen. Als dit zo is moet de root sectie uiteraard niet worden verwijderd na gebruik van de klasse.
ReadString(Sectionname, Itemname, DefaultValue)	Met deze functie kan de waarde van een specifiek item uit het .ini bestand worden uitgelezen. In de parameters van de functie wordt de naam van de sectie en het betreffende item opgegeven. De functie maakt gebruik van het storage.ini object om de waarde van het item uit te lezen. Als het betreffende item niet bestaat wordt de als parameter meegegeven DefaultValue geretourneerd.
ReadAllItemsInSection(SectionName, DefaultValue)	Met deze functie kunnen alle items uit een sectie worden uitgelezen. Deze functie is toegevoegd om het mogelijk te maken om alle beschikbare items uit een sectie uit te lezen, zonder dat hiervoor een item naam gespecificeerd hoeft te worden. De functie maakt hiervoor een `dictionary` object aan. Dit is een generieke datastructuur uit de scripting bibliotheek van Windows waarmee het mogelijk is om waardes op te slaan of te benaderen aan de hand van een unieke sleutelnaam (beter bekend als map structuur) [16]. De functie opent een sectie met behulp van het `storage.ini` object. Vervolgens worden alle items van deze sectie toegevoegd aan het `dictionary` object, waarbij de item naam wordt gebruikt als unieke keynaam voor het `dictionary` element. Er is gekozen voor het gebruiken van deze datastructuur omdat in één `dictionary` item zowel de naam als de bijbehorende waarde kan worden opgeslagen. Ook kunnen alle elementen van deze datastructuur eenvoudig worden benaderd met het VBScript For Each statement. Wanneer een item van het .ini bestand geen waarde bevat wordt de DefaultValue van de parameter gebruikt. De functie retourneert het de al dan niet gevulde `dictionary` datastructuur.
WriteString(SectionName, ItemName, Value)	Met deze functie kan een waarde worden weggeschreven in het .ini bestand. In de parameters van de functie moet de naam van de sectie, de naam van het item en de weg te schrijven waarde worden meegegeven. De functie gebruikt het `storage.ini` object om de waardes in het .ini bestand weg te schrijven. De waardes worden pas daadwerkelijk in het .ini bestand opgeslagen als de memberfunctie `Close` wordt aangeroepen.
removeRootSection()	Deze functie kan na het gebruik van het .ini bestand worden gebruikt om de toegevoegde "root" sectie te verwijderen. De functie opent hiervoor het .ini bestand als tekstbestand, zoekt naar de tekst "[root]" en verwijderd deze tekst.
Close()	Met deze functie wordt het geopende .ini bestand gesloten. Vanuit deze functie wordt ook de memberfunctie `save` van het `storage.ini` object aangeroepen. Hiermee worden de gemaakte wijzigingen definitief opgeslagen in het .ini bestand. Voor deze constructie is gekozen om te kunnen garanderen dat gemaakte wijzigingen op een gespecificeerd moment worden opgeslagen. Wanneer bijvoorbeeld de uitvoer het programma onverwacht wordt afgebroken voordat de close functie is aangeroepen, zijn de gemaakte (mogelijk nog niet complete) wijzigingen niet opgeslagen. Wanneer de private member `rootSectionAlreadyExists` de waarde `False` heeft, wordt de `RemoveRootSection` functie aangeroepen. De `RemoveRootSection` wordt vervolgens op `False` gezet om te voorkomen dat de destructor opnieuw zal proberen de root sectie te verwijderen.

Class_initialize	In de constructor van de klasse wordt het private memberobject voor de later gebruikte `storage.ini` object geïnitialiseerd met het VBScript `Nothing` object. Hierdoor is de variabele altijd als object te benaderen en kan eenvoudig worden gecontroleerd of het memberobject al dan niet is geïnitialiseerd. De private member `rootSectionAlreadyExists`, wordt geïnitialiseerd met de waarde `True`. Deze member wordt gebruikt als vlag om aan te geven of de rootsectie moet worden verwijderd na gebruik van de klasse. Op deze manier wordt gegarandeerd dat de eventuele rootsectie niet wordt verwijderd als nog niet is bepaald of het .ini bestand een root sectie heeft. Deze situatie komt voor als het `IniFile` object wordt verwijderd vóór dat de `CreateRootSection` functie is aangeroepen.
Class_Terminate	In de destructor van de klasse wordt de private member voor het `storage.ini` object verwijderd, zodat gegarandeerd wordt dat het .ini bestand is gesloten na het opruimen van een instantie van deze `IniFile` klasse. Wanneer de private member `rootSectionAlreadyExists` de waarde `False` heeft, wordt de `RemoveRootSection` functie aangeroepen, zodat gegarandeerd wordt dat de gecreëerde root sectie altijd is verwijderd na het opruimen van het `IniFile` object



Figuur 3-2 **Klassediagram voor IniFile klasse**

3.4.3 Generieke klasse voor exporteren van data naar Excel

TestComplete biedt verschillende mogelijkheden voor het lezen van Excel data vanuit een script. Er is echter geen methode voor het schrijven van data naar een Excel bestand. Hiervoor adviseert TestComplete om gebruik te maken van een COM/OLE object [17]. Binnen dit project is deze functionaliteit wenselijk voor het bijhouden en opslaan van timing gegevens van de in RadiMation uitgevoerde kalibraties en EMC testen. Voor het schrijven naar een Excel bestand kan gebruik gemaakt worden van het Excel.Application COM object. Dit COM kan direct in VBScript worden gecreëerd, waarna alle door Excel gedefinieerde methodes, properties en events kunnen worden gebruikt [18].

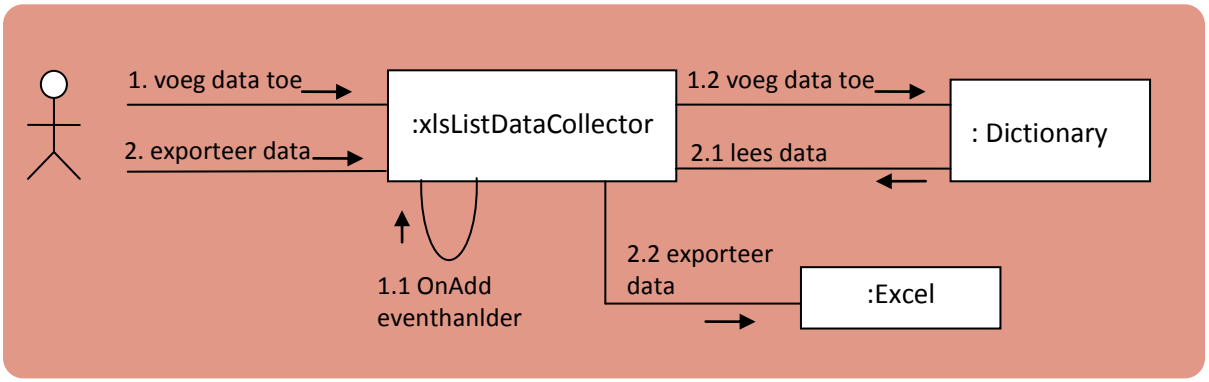
Deze functionaliteit had eenvoudig in de betreffende procedure geïntegreerd kunnen worden. Echter omdat het niet onwaarschijnlijk is dat dergelijke functionaliteit in de toekomst zal worden hergebruikt is besloten een klasse te ontwikkelen waarmee een eenvoudige datacollectie kan worden

bijgehouden. Deze data kan vervolgens met een memberfunctie naar een Excel bestand worden geëxporteerd.

De klasse kan in de eerste instantie worden gebruikt voor het verzamelen van data. Deze data wordt opgeslagen in een private memberobject van het type `Dictionary`. Dit is een standaard datastructuur van de Windows Scripting `dictionary` en kan worden gebruikt als een map datastructuur. Ieder element in de datastructuur bestaat uit een unieke keynaam en een bijbehorende waarde [16]. Hiervoor is gekozen omdat de waardes die moeten worden bijgehouden unieke waardes moeten zijn, en bovendien moeten worden opgeslagen met een bijbehorende naam. De interfacefuncties voor het beheren van de data zijn gebaseerd op de interfacefuncties van het `dictionary` object. In feite bevat deze klasse een enigszins aangepaste functionaliteit van het `dictionary` object, met daaraan toegevoegd de functionaliteit voor het exporteren van de data. Voor deze oplossing zou het overerven van het `Dictionary` object daarom wenselijk zijn. Overerving is echter niet mogelijk in VBScript, daarom is gekozen voor het opnieuw implementeren van de interfacefuncties.

In de klasse is de mogelijkheid geïmplementeerd om een eventhandler functie op te geven voor de memberfunctie `Add()`. Deze mogelijkheid is voor de volgende specifieke situatie geïmplementeerd:

Deze klasse is in de eerste instantie ontwikkeld voor het kunnen verzamelen en exporten van timing gegevens van de in RadiMation uitgevoerde kalibraties en EMC testen. Een speciale wens hierbij is dat naast de timing gegevens ook één element wordt toegevoegd met hierin de actuele softwareversie van RadiMation. Dit versienummer moet worden toegevoegd zodra de eerste timing waarde wordt toegevoegd. Om de ontwikkelde klasse generiek te houden is besloten om de specifieke functionaliteit van het opvragen en opslaan van het versienummer, in een externe functie te implementeren. De eventhandler van de ontwikkelde klasse kan dan gebruikt worden om de externe functie te koppelen aan het `OnAdd` event, zodat altijd het versienummer wordt opgeslagen zodra een item wordt toegevoegd. Het gebruik van de klasse is weergegeven in het communicatiediagram van Figuur 3-3.



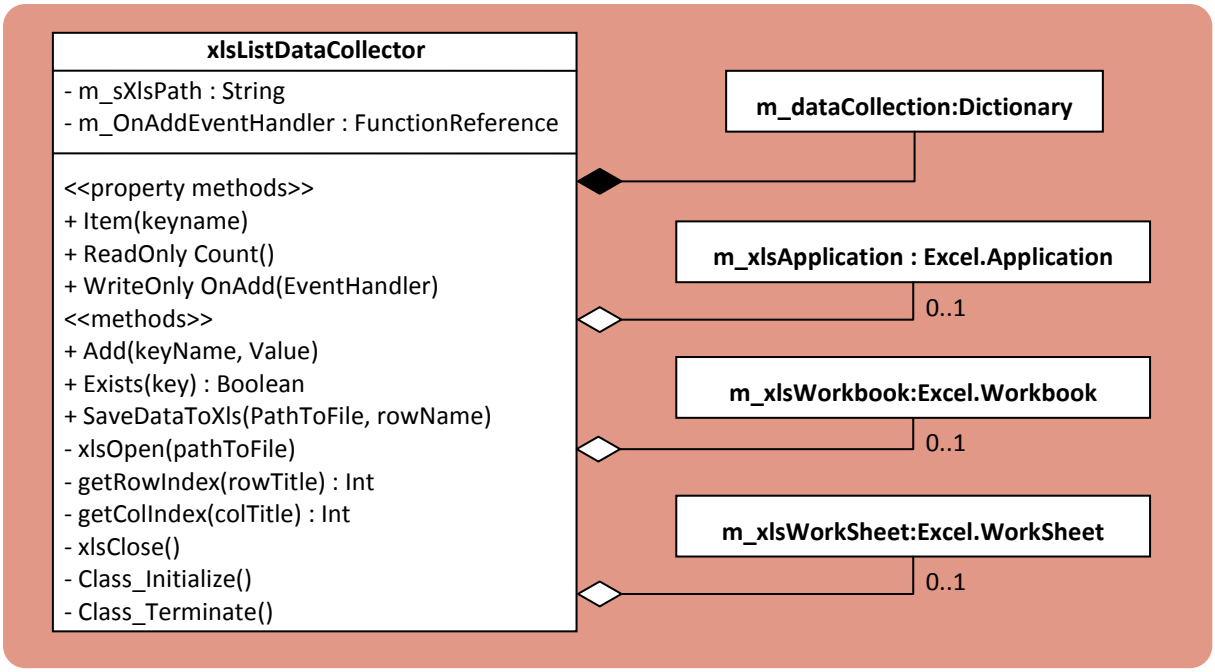
Figuur 3-3 **communicatiediagram van de xlsListDataCollector klasse**

Een overzicht van de verschillende methodes en properties van de klasse zijn weergegeven in Tabel 3-3. Het bijbehorende klassediagram is weergegeven in Figuur 3-4.

Tabel 3-3 **Methodes en properties van de xlsListDataCollector klasse**

Property Item(keyname)	Met deze property methode kan de waarde van een specifiek item worden uitgelezen of aangepast. Wanneer het item met de meegegeven keyname niet bestaat, wordt bij het opvragen een lege variable geretourneerd. Bij het instellen wordt een niet bestaand item als nieuw item opgeslagen.
ReadOnly property Count	Deze property retourneerd het actuele aantal elementen
WriteOnly OnAdd(eventHandler)	Met deze functie kan een referentie naar een functie worden opgegeven. Deze functie wordt uitgevoerd zodra de memberfunctie `Add()` wordt aangeroepen. De eventhandler kan worden uitgeschakeld door een `Nothing` object in te stellen voor deze property
Add(keyName, value)	Met deze functie wordt een nieuwe waarde toegevoegd aan de private `dictionary` datastructuur. Hiervoor worden de meegegeven keyname en value gebruikt. Wanneer er een eventhandler is ingesteld voor het OnAdd event, wordt de opgegeven functiereferentie aangeroepen. Deze functie wordt aangeroepen vóór dat de meegegeven waarde is toegevoegd aan de `dictionary`.
SaveDataToXls(pathToFile, rowName)	Met deze functie wordt alle verzamelde data geëxporteerd naar het Excel bestand waarvan het pad is opgegeven in de parameter. De worksheet wordt geopend met de private functie `xlsOpen`. Vervolgens wordt de data weggeschreven in kolommen. Iedere keynaam uit de dictionary wordt een nieuwe kolom. De waardes worden allemaal in dezelfde rij geschreven. De naam van de rij wordt als parameter meegegeven aan deze functie. Voor het wegschrijven van de data wordt eerst de rij index bepaald met de `getRowIndex` functie. Vervolgens wordt voor ieder data element de kolom index bepaald met de `getColIndex` functie. Vervolgens worden alle waardes weggeschreven met de verkregen rij en kolom index. Als alle elementen zijn geëxporteerd wordt de `xlsClose` functie aangeroepen.
xlsOpen(filename)	De functie controleert of het opgegeven bestand bestaat en bepaald aan de hand daarvan of een bestaand bestand moet worden geopend, of dat een nieuw bestand moet worden aangemaakt. Er wordt altijd gebruik gemaakt van de eerste worksheet van het bestand. De geopende excel applicatie, workbook en worksheet worden als private memberobjecten opgeslagen, zodat deze door de andere memberfuncties kunnen worden gebruikt.
getRowIndex(rowTitle)	Deze functie retourneert ofwel de index van de bestaande rij, of voegt de betreffende rij onderaan toe, en retourneert hier de index van. De functie zoekt hiervoor naar de rij waarvan de inhoud van de eerste kolum exact overeenkomt met de meegegeven parameterwaarde. Als deze rij wordt gevonden wordt de index van deze rij geretourneerd. Als deze rij niet wordt gevonden, wordt de index gegeven van de rij direct na de allerlaatste niet lege rij. Hiervoor wordt alleen in de eerste kolom gekeken. In dit geval wordt de waarde van de parameter direct ingevuld in de eerste kolom van de nieuwe rij.
getColIndex(colTitle)	Deze functie retourneert ofwel de index van de bestaande kolom, of voegt de betreffende kolom na de laatste kolom toe en retourneert hier de index van. De functie zoekt hiervoor naar de kolom waarvan de inhoud van de eerste rij exact overeenkomt met de meegegeven parameterwaarde. Als deze kolom wordt gevonden wordt de index van deze kolom geretourneerd. Als deze kolom niet wordt gevonden, wordt de index gegeven van de kolom

	direct na de allerlaatste niet lege kolom. Hiervoor wordt alleen in de eerste rij gekeken. In dit geval wordt de waarde van de parameter direct ingevuld in de eerste rij van de nieuwe kolom.
xlsClose()	Met deze functie worden de aangebrachte wijzingen opgeslagen. Vervolgens wordt het bestand en de geopende excel applicatie gesloten.
Class_Initialize	In de constructor van de klasse wordt een `dictionary` object geïnstantieerd. De overige members worden geïnitialiseerd met een `Nothing` object. Dit wordt in de constructor gedaan omdat het in VBScript niet mogelijk is om een standaard waarde mee te geven bij definitie van private members.
Class_terminate	De destructor van de klasse bevat geen extra functionaliteit. Memberobjecten worden standaard al opgeruimd wanneer de instantie van een klasse wordt opgeruimd.



Figuur 3-4 **klassediagram voor xlsListDataCollector klasse**

3.5 Testresultaten

Doordat deze testmethode in de tweede en dus laatste fase van het afstudeertraject is uitgevoerd zijn geen resultaten bekend van na de ontwikkelding, zoals het wel het geval is bij de ontwikkelde RadiCentre test. Echter, tijdens de (her)ontwikkeling van de verschillende testprocedures zijn de betreffende onderdelen steeds uitgevoerd om de werking hiervan te controleren. Resultaat is dat hiermee automatisch een groot deel van de steeds gebruikte RadiMation versie is getest. Hierbij zijn een aantal tot dan toe onbekende fouten ontdekt, en is meer gedetailleerde informatie ontdekt van bekende fouten. Een belangrijk voorbeeld hiervan is de uitgevoerde device driver test. Deze her-ontwikkelde test voegt alle beschikbare RadiMation drivers toe, en controleert vervolgens voor

iedere driver of het “Advanced” scherm geopend kan worden. Door de grote hoeveelheid drivers zou het handmatig uitvoeren van de test enkele weken in beslag nemen. Met behulp van de geautomatiseerde test was na 4 uur bekend dat er problemen waren voor een zeer beperkt aantal drivers. De oorzaak en details van de problemen konden eenvoudig achterhaald worden aan de hand van het testrapport.

4 Conclusie

Het doel van het afstudeertraject wat in deze scriptie wordt beschreven is het ontwikkelen van twee verschillende testmethodes voor het geautomatiseerd testen van de door Dare!! Instruments ontwikkelde EMC test producten.

De doelstelling voor de eerste testmethode is het kunnen testen van alle RadiCentre producten, door het testen van alle beschikbare commando's via alle mogelijke communicatiemethodes. Voor de realisatie van deze doelstelling is een nieuwe testmethode ontwikkeld, bestaande uit een generiek testprogramma ontwikkeld in het programma TestComplete en een bijbehorende commandodatabase waarin de specifieke testfunctionaliteit voor de verschillende RadiCentre producten is gedefinieerd. De implementatie van de methode is volledig afgerond en de testmethode is gedurende de afstudeerperiode in gebruik genomen. Hieruit zijn een aantal bijzonder waardevolle testresultaten voortgekomen. Aan de ene kant geven deze testresultaten weer dat de ontwikkeling van de testmethode is geslaagd. Tegelijk laten deze testresultaten ook de meerwaarde en het belang van de ontwikkelde test zien. Mede op basis van de behaalde testresultaten kan worden geconcludeerd dat deze doelstelling is behaald.

De doelstelling voor de tweede testmethode is het kunnen testen van de belangrijkste onderdelen van de RadiMation EMC test software. Een specifiek doel hierin is het herontwikkelen van de belangrijkste procedures van een bestaande testmethode, namelijk die voor het testen van kalibraties en EMC testen in RadiMation, zodat deze kunnen worden gebruikt voor de laatste RadiMation versie. Specifieke wensen daarbij zijn dat de test wordt uitgebreid met een tijdsmeting voor de kalibraties en EMC testen en de mogelijkheid om kalibraties en EMC testen te kunnen uitvoeren met zowel virtuele als fysieke meetapparatuur. Voor de realisatie van deze doelstelling zijn de testprocedures van de genoemde onderdelen herontwikkeld, waardoor deze onderdelen van de testmethode gebruikt kunnen worden voor de laatste versie van de RadiMation software. Daarnaast zijn een aantal uitbreidingen geïmplementeerd waaronder een automatische tijdsmeting voor de uitgevoerde kalibraties en EMC metingen. Ook zijn uitbreidingen geïmplementeerd als voorbereiding op het kunnen uitvoeren van kalibraties en EMC metingen waarbij gebruik gemaakt wordt van fysieke meetapparatuur. In de ontwikkelingsfase van deze testmethode zijn verschillende specifieke onderdelen van de test uitgevoerd, waarin opnieuw waardevolle testresultaten zijn gegenereerd. Alle voor dit project als "must" gespecificeerde eisen zijn behaald. Daarnaast zijn ook vrijwel alle should en could doelen verwezenlijkt. Het "should" doel om de test te kunnen gebruiken voor fysieke meetapparatuur is niet volledig geimplementeerd en getest vanwege de beperkte beschikbaarheid van het hiervoor beoogde testlaboratorium en de beperkte tijd van het project. Deze functionaliteit is voor het grootste gedeelte voorbereid waardoor definitieve implementatie eenvoudig is. Op basis hiervan en op basis van de behaalde testresultaten kan ook hier worden gesteld dat de doelstelling is behaald.

Als eindconclusie voor het gehele afstudeertraject kan worden gesteld dat alle doelstellingen zijn behaald. De behaalde testresultaten geven het grote belang en de meerwaarde van de ontwikkelde testmethodes aan.

5 Aanbevelingen

In deze paragraaf worden een aantal specifieke aanbevelingen gegeven voor de ontwikkelde testmethodes.

5.1 Detectie USB poortnummer

Bij het toepassen van de ontwikkelde RadiCentre testmethode is gebleken dat optimalisatie mogelijk is voor het testen van grotere aantallen. Wanneer een nieuw RadiCentre via USB wordt aangesloten aan de testcomputer, wordt deze altijd gezien als nieuw product waar de drivers voor geïnstalleerd moeten worden. Dit USB apparaat is in werkelijkheid een USB-RS232 omvormer. Resultaat is dat voor ieder nieuw aangesloten RadiCentre USB apparaat een nieuw com-poort nummer wordt toegewezen.

Voor het testen van de RadiCentre's betekent dit momenteel dat voor ieder RadiCentre ofwel het toegekende poortnummer moet worden aangepast, ofwel het nieuwe poortnummer moet worden gewijzigd in het testprogramma. Wanneer de testmethode gebruikt gaat worden voor het testen van grote aantalen RadiCentre's via USB is dit een tijdrovend proces. Daarom wordt aanbevolen om een procedure te ontwikkelen waarmee het USB apparaat van het aangesloten RadiCentre automatisch kan worden gedetecteerd.

5.2 Opbouw RadiMation test

In het huidige TestComplete project van de RadiMation testmethode is geen verschil aangebracht tussen procedures voor het testen van een totale functionaliteit (zoals het uitvoeren van EMC metingen) en het individueel testen van alle gebruikersinterface elementen. In een aantal gevallen is het testen van gebruikerselementen zelfs nog verweven in procedures voor het testen van een complete functionaliteit. Door "onnodig" bedienen van alle gebruikersinterface elementen kan minder waardevolle informatie worden gehaald uit het resultaat van de test. De benodigde tijdsduur van een specifieke functionaliteit is bijvoorbeeld nietszeggend als de uitvoer sterk wordt vertraagd door het onnodig bedienen van gebruikersinterface elementen. Bij het vervolg van de herontwikkeling wordt daarom aanbevolen om een duidelijke scheiding te maken tussen procedures voor het testen van totale functionaliteiten en procedures voor het simpelweg testen van alle gebruikersinterface elementen.

5.3 Structuur scriptcode RadiMation test

De huidige scriptcode van het RadiMation testproject bevat een grote hoeveelheid dubieuze routines. Veel routines worden bijvoorbeeld niet meer gebruikt, bevatten sterk vergelijkbare functionaliteit met andere routines of zijn dusdanig specifiek dat deze niet meer kunnen worden hergebruikt. Ook is de relatie met andere functies in een aantal gevallen buitengewoon complex. Naar alle waarschijnlijkheid is de belangrijkste reden voor deze punten het feit dat het programma verschillende keren is aangepast en uitgebreid. Dit maakt de uitbreidbaarheid en vooral de onderhoudbaarheid van dit programma complex en tijdrovend. Daarom wordt aanbevolen om bij het vervolg van dit hervormingsproject te investeren in het volledig herstructureren van de bestaande code en het opruimen van dubieuze of ongebruikte code. Hierbij wordt ook het gebruik van een object georiënteerde structuur sterk aanbevolen.

6 Bronnen

[1] : over TestComplete http://support.smartbear.com/viewarticle/55657/

[2] : Open applicaties http://support.smartbear.com/viewarticle/55256/
.NET aplicaties http://smartbear.com/products/qa-tools/automated-testing-tools/windows-desktop-testing/testing-dotnet-applications/

[3] : Test Items http://support.smartbear.com/viewarticle/55015/

[4] : project variabelen http://support.smartbear.com/viewarticle/55666/

[5] : test rapport http://support.smartbear.com/viewarticle/55125/

[6] : storages.ini http://support.smartbear.com/viewarticle/55684/

[7] : CLR-Bridge http://support.smartbear.com/viewarticle/55447/

[8] : NI-Visa http://sine.ni.com/nips/cds/view/p/lang/nl/nid/12145

[9] : NI-Visa help documentatie, geintegreerd in de NI-Visa software

[10] : NI-Visa “Simple I/O Example”, onderdeel van NI-Visa help documentatie

[11] : Reguliere Expressies http://msdn.microsoft.com/en-us/library/6wzad2b2.aspx

[12] : VBScript klasse http://msdn.microsoft.com/en-us/library/4ah5852c%28v=vs.84%29.aspx

[13] : TestComplete DBTableVariable http://support.smartbear.com/viewarticle/55656/

[14] : TestComplete UseUnit statement http://support.smartbear.com/viewarticle/56539/

[15] : MoSCoW methode http://nl.wikipedia.org/wiki/MoSCoW-methode

[16] : Dictionary object http://msdn.microsoft.com/en-us/library/x4k5wbx4%28v=vs.84%29.aspx

[17] : TestComplete schrijven in excel http://support.smartbear.com/viewarticle/20878/

[18] : Excel.application COM object http://msdn.microsoft.com/en-us/library/office/ff194565%28v=office.15%29.aspx

Bijlagen