Afstudeerscriptie

Integratietest voor de ASF+SDF MetaEnvironment

Peter Heibrink

1 jarige Master Software Engineering

Afstudeerdocent:	Prof. Dr. P. Klint
Stagebegeleider:	Dr. M.G.J. van den Brand
Opdrachtgever:	Centrum voor Wiskunde en Informatica

Universiteit van Amsterdam, Vrije Universiteit, Hogeschool Amsterdam

Voorwoord

Dit verslag is geschreven in het kader van mijn afstudeeropdracht bij het Centrum voor Wiskunde en Informatica te Amsterdam.

Het is bestemd voor de medewerkers van het Centrum voor Wiskunde en Informatica en voor docenten en studenten van de Universiteit van Amsterdam, de Vrije Universiteit en de Hogeschool van Amsterdam.

Ten slotte wil ik de mensen van het Centrum voor Wiskunde en Informatica bedanken voor de mogelijkheid die zij mij gegeven hebben om mijn afstudeerproject daar uit te voeren. In het bijzonder wil ik prof. dr. P. Klint en dr. M.G.J. van den Brand bedanken voor het helpen op de momenten dat ik was vast gelopen.

Spijkenisse, augustus 2005

Inhoudsopgave

Samenvatting

Dit verslag behandelt het onderzoek naar geautomatiseerd integratietesten voor de ASF+SDF MetaEnvironment. De onderzoeksvraag die in dit verslag centraal staat is: "*Hoe is een automatische integratietest voor de ASF+SDF MetaEnvironment te ontwikkelen?*".

Om deze vraag te kunnen beantwoorden, wordt eerst uitgelegd hoe de ASF+SDF MetaEnvironment werkt. Een belangrijk punt hierbij is de ToolBus architectuur die hiervoor gebruikt is.

Vervolgens worden een aantal onderzoeken behandeld, die nodig waren om de tools voor de integratietest te kunnen ontwerpen. Hierbij worden de volgende aspecten behandeld:

- De reden dat integratietests uitgevoerd worden.
- De samenwerking tussen componenten binnen een ToolBus architectuur.
- Het parameteriseren van een integratietest.
- Het bepalen van equivalentie bij niet deterministische berichten.

Met de kennis, die hiermee is opgedaan, zijn vier tools ontwikkeld waarmee de integratietest uit te voeren is. Dit zijn:

- De metalogger
- De imitatortool
- De testgenerator
- De metaplaybacker

Met deze tools is het mogelijk om geautomatiseerde integratietests voor de ASF+SDF MetaEnvironment te maken. Zowel de functionaliteit als de manier waarop de tools gebruikt moeten worden, is in dit verslag terug te vinden.

Het verslag wordt afgesloten met de resultaten, die dit onderzoek hebben opgeleverd en een zelfevaluatie. De resultaten zijn:

- Het onderzoek naar de manier waarop automatische integratietests binnen de ASF+SDF MetaEnvironment te ontwikkelen zijn. Hiervoor is gekozen voor het record en playback principe.
- De vier tools, die de integratietest mogelijk maken.

1 Inleiding

In dit hoofdstuk wordt de achtergrond van de ASF+SDF MetaEnvironment beschreven en wordt uitgelegd waar deze voor wordt gebruikt. Daarna wordt de context van de opdracht uitgelegd en wordt de onderzoeksvraag gepresenteerd. Tot slot wordt er uitgelegd hoe dit verslag is opgebouwd.

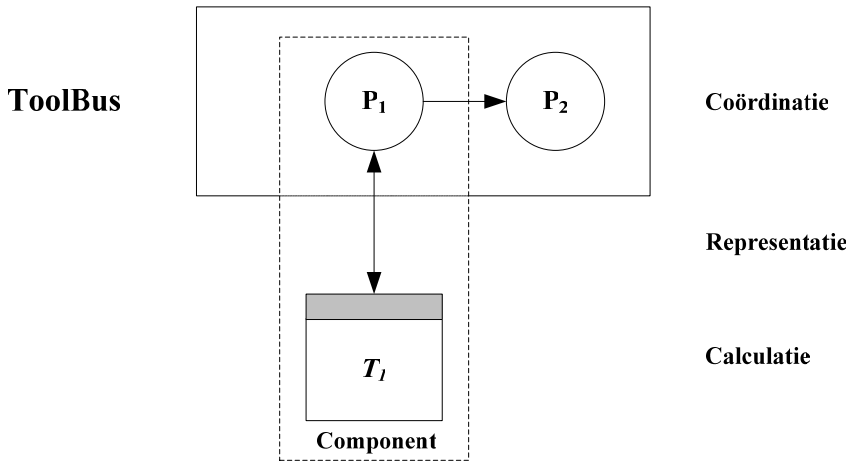
1.1 Achtergrond

De ASF+SDF MetaEnvironment is een interactieve omgeving, waarmee taalspecifieke tools ontwikkeld kunnen worden, die gebaseerd zijn op formele taal specificaties. Het is mogelijk om zowel de syntactische als semantische aspecten van talen met deze omgeving te definiëren. De omgeving bevat veel verschillende functionaliteiten, bijvoorbeeld typechecking en prettyprinting. De ASF+SDF MetaEnvironment is in de volgende omstandigheden te gebruiken [18]:

- Wanneer een formele specificatie voor een probleem opgesteld moet worden, waarbij interactieve ondersteuning nodig is.
- Wanneer een eigen taal ontwikkeld wordt en er een interactieve omgeving voor ontwikkeld moet worden.
- Wanneer programma's in een bepaalde programmeertaal geanalyseerd of getransformeerd moeten worden.

De ASF+SDF MetaEnvironment is geïmplementeerd op een component gebaseerde manier, waarbij de ToolBus wordt gebruikt als coördinatie architectuur om de samenwerking tussen de componenten te coördineren. De ToolBus koppelt verschillende componenten aan elkaar, waardoor één applicatie ontstaat. Het is ook mogelijk, dat verschillende componenten op verschillende computers worden geplaatst, waardoor een gedistribueerd systeem over een LAN gemaakt kan worden [14, 15, 16].

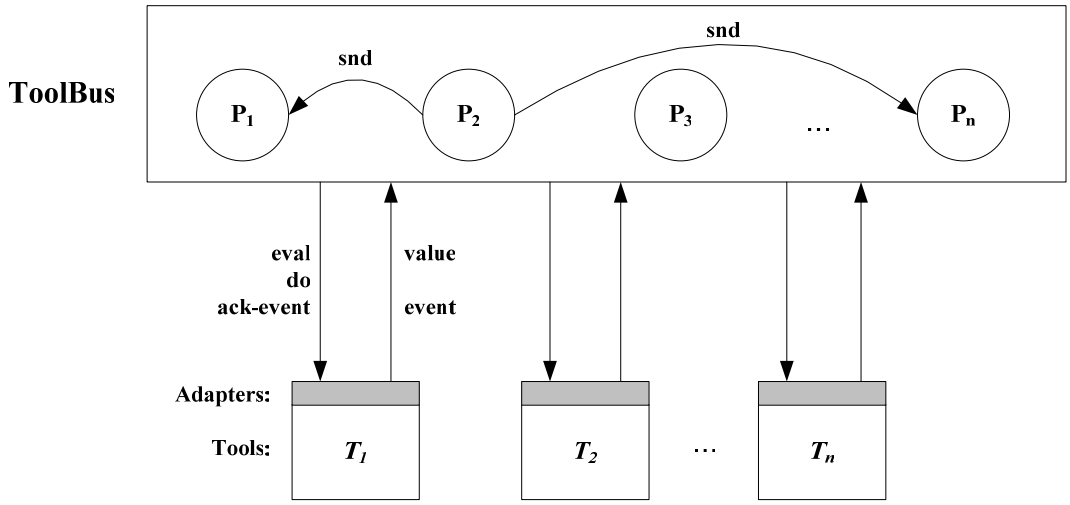
De ToolBus zelf is een Unix proces waarin T-scripts uitgevoerd worden. In deze T-scripts zijn de processen gedefinieerd, die binnen de applicatie uitgevoerd moeten worden. Deze processen kunnen onderling communiceren. Daarnaast kunnen de processen tools opstarten die vervolgens berekeningen voor hun rekening kunnen nemen of user interfaces kunnen weergeven.



Figuur 1: Opdeling component

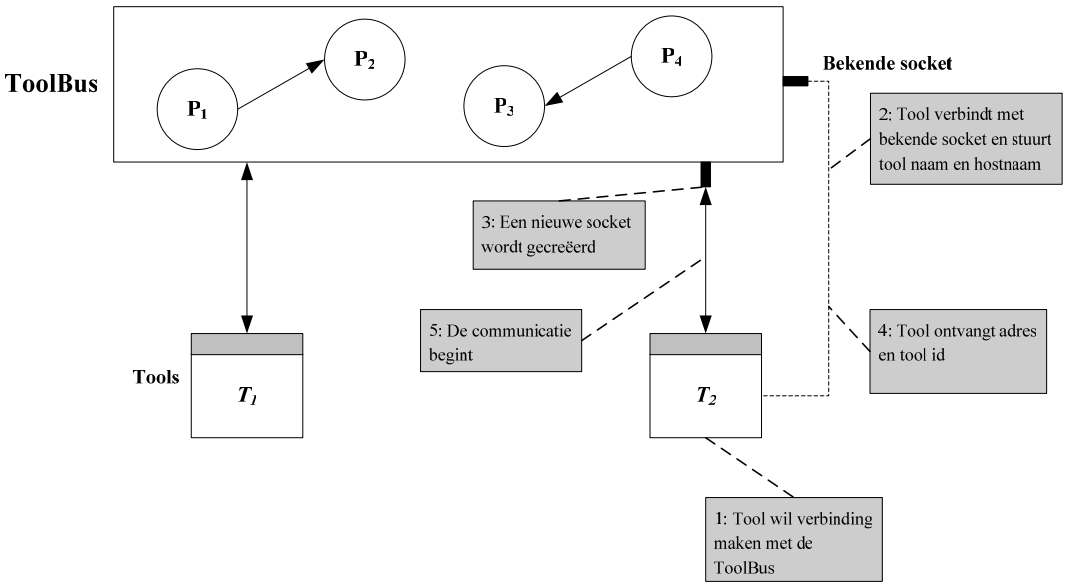
Een belangrijke scheiding die gemaakt wordt doormiddel van de ToolBus architectuur, is de scheiding tussen coördinatie en calculatie (zie figuur 1) [14]. Componenten worden hiervoor in

tweeën gedeeld, namelijk in een proces deel (T-script) en in een tool. De representatie van de data die hiertussen wordt verstuurd, wordt gedaan met ATermen [17]. Er zijn een aantal basisfuncties, die binnen T-scripts gebruikt kunnen worden, maar alle belangrijke verwerking gebeurt binnen de tools (de ToolBus functies staan uitgelegd in [16, 17]). Deze tools weten niets van de overige tools, die verbonden zijn met de ToolBus [14]. De processen zorgen ervoor, dat de geleverde data of events van een tool bij het proces terecht komt dat daar verder mee moet werken (zie figuur 2).



Figuur 2: ToolBus [1]

Tools kunnen op twee manieren opgestart worden. Ze kunnen worden opgestart door de T-scripts, waarbij alle benodigde argumenten meegegeven worden. (Een overzicht van alle argumenten staat in [16]). De tools kunnen echter ook afzonderlijk worden opgestart, waarna ze zelf contact zoeken met de ToolBus. Dit kan, doordat de ToolBus een bekend port nummer heeft waar de tool een port kan aanvragen. De tool krijgt vervolgens een portnummer terug en de tool is klaar om gebruikt te worden binnen de applicatie. Dit proces te zien in figuur 3.



Figuur 3: Verbinden met de ToolBus

Een volledige syntax definitie van de huidige versie van de ToolBus staat in [16].

1.2 Context

De ASF+SDF MetaEnvironment is nog steeds in ontwikkeling. Dit betekent dat alle aanpassingen, die hierop gedaan worden, goed moeten worden getest. Hiervoor zijn voor alle componenten unittests gespecificeerd om de componenten te valideren. Hiermee worden elke nacht automatisch de laatste versies van de bronbestanden getest. Nu is er echter geen automatische integratietest, die de samenwerking van de verschillende componenten valideert. In plaats daarvan wordt deze integratietest nu handmatig uitgevoerd wat tijd kost. Deze test is wel belangrijk, want het controleert of een aanpassing in één component de werking van een ander component aantast. Als de integratietest geautomatiseerd is, kan deze veel vaker uitgevoerd worden.

1.3 Onderzoeksvraag

De onderzoeksvraag van de afstudeeropdracht luidt als volgt:

> *"Hoe is een automatische integratietest voor de ASF+SDF MetaEnvironment te ontwikkelen?"*

De bedoeling van de afstudeeropdracht is, dat een integratietestomgeving ontwikkeld wordt voor de ASF+SDF MetaEnvironment. Deze moet de samenwerking van de verschillende componenten testen.

Deze onderzoeksvraag is op te delen in een aantal subonderzoeksvragen:

1. Hoe kan een integratietest voor de ASF+SDF MetaEnvironment uitgevoerd worden?
2. Hoe werken de verschillende componenten binnen de ToolBus samen?
3. Hoe is de integratietest te parameteriseren om de gewenste processen in de test te gebruiken?
4. Hoe is de equivalentie van niet deterministische berichten te bepalen tijdens testen?
5. Hoe kan de integratietest geïmplementeerd worden?

In de rest van dit verslag zullen de subonderzoeksvragen beantwoord worden. Hierdoor kan uiteindelijk ook de onderzoeksvraag worden beantwoord.

1.4 Leeswijzer

In hoofdstuk twee wordt het plan van aanpak beschreven. Hierin staat het stappenplan dat is gevolgd tijdens dit project. Daarna wordt in hoofdstuk drie het onderzoek beschreven dat nodig is geweest, voordat de integratietest ontworpen kon worden. Hierbij wordt gekeken naar wat een integratietest is, op welke manier de communicatie tussen de componenten binnen de ToolBus gebeurt, hoe een integratietest te parameteriseren is en hoe de equivalentie bij niet deterministische berichten te bepalen is.

Aan de hand van deze informatie zijn een viertal tools ontwikkeld, namelijk de metalogger, de imitatortool, de testgenerator en de metaplaybacker. De functionaliteit van deze tools wordt in hoofdstuk vier beschreven. Hoofdstuk vijf behandelt vervolgens het maken van een integratietest met deze tools. Dit moet in drie fases gebeuren: de opnamefase, het genereren van een testomgeving en een playbackfase.

In hoofdstuk zes zijn de resultaten van het onderzoek terug te vinden, waarna hoofdstuk zeven een zelfevaluatie behandelt die na het onderzoek is gemaakt.

2 Plan van aanpak

Tijdens de afstudeerperiode zijn er een aantal stappen geweest die ik heb uitgevoerd. Dit zijn:

1. Onderzoek naar de ToolBus en de bijbehorende ToolBus logger functionaliteit.
2. Ontwerp en ontwikkeling van de basisfunctionaliteit van de metalogger.
3. Ontwerp en ontwikkeling van de basisfunctionaliteit van de metaplaybacker.
4. Ontwerp en ontwikkeling van de basisfunctionaliteit van de toolimitator.
5. Ontwerp en ontwikkeling van de basisfunctionaliteit van de testgenerator.
6. Beginnen met het schrijven van de scriptie.
7. Afmaken van metalogger, toolimitator, testgenerator en metaplaybacker.
8. Afmaken van de scriptie.

In de eerste fase heb ik onderzocht hoe de ToolBus met de bijbehorende logger werkt. Veel kennis van de ToolBus was tijdens het vooronderzoek al opgedaan, maar tijdens deze fase is de werking van de ToolBus logger helemaal getest. Dit omdat hier geen documentatie van beschikbaar is.

Zodra ik door had hoe de logger werkte, ben ik begonnen met het ontwerpen en ontwikkelen van de opnamefunctionaliteit. Hierbij heb ik een filter ontwikkeld om alleen de gewenste informatie te verwerken. Voor de opnamefunctionaliteit is de metalogger ontwikkeld.

Hierna heb ik de metaplaybacker ontwikkeld. Deze logger applicatie controleert tijdens de playbackfase of alle berichten, die tussen de componenten worden verstuurd, tijdens de opname zijn opgeslagen. Met deze applicatie is de integratietest al uit te voeren. Het is dan alleen wel zo, dat de tester zelf tijdens de test dezelfde input moet geven als tijdens de opname.

Tijdens het ontwikkelen van de metaplaybacker, kwam ik erachter dat er meer nodig was dan een record en een playback tool. Dit omdat ik niet zomaar events vanuit een logger applicatie kan laten genereren. In eerste instantie was ik van plan de metaplaybacker alle events en evaluatieverzoeken te laten afhandelen. De ToolBus accepteert deze opzet echter niet. De berichten moeten namelijk vanuit de tools zelf verstuurd worden. Om de test toch automatisch te kunnen laten verlopen, zijn er vervangende tools nodig, die de originele tools imiteren.

Na de metaplaybacker heb ik een imitatortool ontwikkeld. Deze heb ik in eerste instantie gemaakt voor de grafische user interface. De imitatortool gebruikt de opgeslagen communicatie van tijdens de metalogger, om automatisch dezelfde berichten naar de ToolBus te sturen als tijdens de opnamefase. Zodra deze applicatie goed functioneerde, is er een meer generieke tool van gemaakt. Om eenvoudig imitatortools te maken voor verschillende tools, heb ik een testgenerator ontwikkeld. Deze analyseert de opname en genereert specifieke imitatortools voor de tools, die vervangen moeten worden. Hierbij worden de bridge bestanden apart gegenereerd, zodat de ToolBus de signatures accepteert.

Tijdens het ontwikkelen van de imitatortool heb ik veel aanpassingen gemaakt aan de metaplaybacker. Dit omdat veel functionaliteit van de playbackfase in zowel de metaplaybacker als in de imitatortools te vinden is en deze moesten op elkaar afgestemd worden.

Zodra de basisfunctionaliteit van de testomgeving was gemaakt, ben ik begonnen aan de scriptie. Dit om zoveel mogelijk van het werk alvast te documenteren. De metalogger, testgenerator en de metaplaybacker konden later namelijk uitgebreid worden. Hierdoor was beter in te schatten hoeveel tijd er nog beschikbaar is voor het ontwikkelen van de verschillende tools.

Nadat de scriptie voor zover mogelijk geschreven was, zijn de verschillende tools uitgebreid, waarna de scriptie afgerond kon worden.

3 Onderzoek

Dit hoofdstuk behandelt vier subonderzoeken, die tijdens de afstudeerperiode verricht zijn. Deze kennis is nodig voordat de integratietest voor de ASF+SDF MetaEnvironment ontwikkeld kan worden. Als eerste wordt er gekeken naar integratietesten. Hierbij wordt gekeken wat integratietesten inhoud, hoe record en playback toegepast kan worden en welke teststrategieën er te gebruiken zijn.

Vervolgens wordt er gekeken hoe de communicatie binnen een ToolBus applicatie werkt. Hier kan bij het opzetten van een integratietest gebruik van gemaakt worden. Als derde deel wordt de mogelijkheid van het parameteriseren van de integratietest bekeken. Dit is nodig om te kunnen kiezen welke componenten in de integratietest meegenomen moeten worden. Tot slot wordt er gekeken naar de equivalentie bij niet deterministische berichten. De volgorde van het communiceren van componenten binnen de ASF+SDF MetaEnvironment staat niet vast. Er moet daarom niet zozeer gekeken worden naar de volgorde van de berichten, die binnen de ToolBus worden verzonden, maar wel naar de berichten zelf.

3.1 Integratietest

In dit hoofdstuk worden de mogelijkheden van integratietesten voor de ASF+SDF MetaEnvironment bekeken. Hiervoor wordt gekeken naar de verschillende vormen van testen, welke vorm van integratietest voor de ASF+SDF MetaEnvironment gekozen is en met welke teststrategie de testgevallen bepaald kunnen worden.

3.1.1 Testen

Testen is nodig om de kwaliteit van een software product te bewaken [1, 2, 4, 9]. Het is dan ook het belangrijkste middel om te controleren, of het product doet wat het zou moeten doen. Dit moet zowel tijdens de ontwikkeling, als tijdens de levensduur van het product gebeuren.

Volgens [2, 5, 8] zijn er drie soorten tests die minimaal moeten worden verricht, dit zijn:

- Unittests: Hierbij worden de individuele componenten getest.
- Integratietests: Waarbij het subsysteem dat ontstaan door de integratie van individueel geteste componenten, wordt getest als een entiteit [10].
- Systeemtests: Waarbij het systeem dat gevormd wordt door de geteste subsystemen, wordt getest als een entiteit.

Dit onderzoek richt zich uitsluitend op de integratietest. In [6] is te lezen, dat integratietesten de manier is om tests uit te voeren, waarbij de componenten worden geïntegreerd in het systeem. Voor integratietesten kan zowel gekozen worden voor whitebox als voor blackbox testen (of beide). Dit hangt vooral van de grote van de subsystemen af. Meestal wordt de nadruk gelegd op de interface code, aangezien de code van de individuele componenten, die worden geïntegreerd, al getest is met unittests [2, 5].

Op dit moment worden alle componenten van de ASF+SDF MetaEnvironment getest met behulp van unittests. Dit is op dit moment de enige soort test, die is geautomatiseerd. Om beter en sneller te kunnen testen is een geautomatiseerde integratietest nodig. Binnen de ASF+SDF MetaEnvironment kan de communicatie tussen de componenten gebruikt worden om te controleren, of er geen fouten in de onderlinge samenwerking zit.

Een van de hoofdproblemen voor een kostenefficiënte integratie van componenten is het minimaliseren van het aantal stubs en drivers, die geschreven moeten worden [6]. Om het opzetten van een testomgeving in een zo kort mogelijke tijd te kunnen doen, moeten deze voor de

integratietest zoveel mogelijk automatisch gegenereerd kunnen worden, waardoor de tester zelf geen stubs en drivers hoeft te ontwikkelen.

3.1.2 Record en playback

Zoals hiervoor vermeld is, wordt er bij integratietesten vooral gekeken naar de interfaces van de verschillende componenten. Door de ToolBus architectuur van de ASF+SDF MetaEnvironment kunnen de berichten, tussen de verschillende componenten eenvoudig opgevangen worden. Hierdoor kan bepaald worden of de berichten, die verstuurd worden, gelijk zijn aan wat ze zouden moeten zijn.

Om deze controle te kunnen automatiseren, is er gekozen om het record en playback principe toe te passen. Hierbij worden de berichten, die tussen de verschillende componenten verstuurd worden opgeslagen. Zodra er aanpassingen op de ASF+SDF MetaEnvironment zijn gemaakt, kunnen deze berichten via het playbackprincipe herhaald worden. Op dat moment kan er gecontroleerd worden, of het systeem nog op dezelfde manier reageert op de herhaalde berichten. Doordat er bij de integratietest gekeken wordt naar de berichten tussen de verschillende componenten, hoeft de record- en playbackfunctionaliteit niet de gebruikersinteractie met het systeem op te slaan, zoals bij veel “commerciële” testrobots gebeurt [13]. Hierdoor blijven de opgenomen testgevallen ook bruikbaar als de lay-out van de user interface aangepast wordt.

Het record en playback principe is al vaker gebruikt voor integratietesten. Hierdoor zijn de veel gemaakte fouten uit het verleden in de literatuur terug te vinden. Er zijn gevoeligheden van record en playback waar rekening mee gehouden moet worden [11, 13]. Dit zijn:

Gedragsgevoeligheid

Dit treedt op, als er veranderingen in het systeem optreden door aanpassingen, bijvoorbeeld als de requirements veranderen. Door deze veranderingen zou een deel van de opgenomen testgevallen kunnen falen. Wat echter ook gebeurt, is dat het systeem niet in de goede staat voor de komende tests achtergelaten wordt, waardoor alle vervolgtests ook falen.

Interfacegevoeligheid

Dit probleem treedt op bij “commerciële” testrobots. Deze werken veelal via de user interface. Als de lay-out hiervan aangepast wordt, kunnen opgenomen tests al falen, terwijl dit niet zou gebeuren, als de tests door een persoon uitgevoerd zouden worden.

Datagevoeligheid

Alle tests hangen op de een of andere manier af van precondities. In informatiesystemen is dit de data, die reeds in het systeem aanwezig is. Mocht deze data veranderen, dan moeten alle testgevallen, die daarmee te maken hebben, aangepast worden.

Contextgevoeligheid

Soms hangt het gedrag van systemen af van de status van de context, alles buiten het systeem. Dit kunnen apparaten, andere applicaties of zelfs de systeemklok zijn.

3.1.3 Teststrategieën

In de literatuur zijn er niet veel teststrategieën te vinden, die voor integratietesten te gebruiken zijn [3, 8]. Veel literatuur gaat over principes als: incrementele integratie, big bang integratie, top down en bottom up integratie [11], het onderzoeken van de code, of het ontwerp van de applicatie.

In [3] wordt een teststrategie aangedragen, die erg geschikt is om toe te passen op de ASF+SDF MetaEnvironment. Deze paper beschrijft een E2E (end-to-end) methode. Dit is een methode van integratietesten, waarbij alleen gefocust wordt op het gebruikerspunt. Hierbij wordt gekeken hoe de gebruiker met het systeem zou omgaan en welke acties de gebruiker zou verrichten. De functies die de gebruiker daadwerkelijk gebruikt, worden hiervoor in thin threads geplaatst. Een thin thread is een basisfunctie binnen het systeem en vormt een testscenario, bijvoorbeeld het parseren van een term.

Van een thin thread kan een testgeval gegenereerd worden door aan de condities variabelen toe te kennen. Deze variabelen kunnen door verschillende technieken gekozen worden, zoals equivalentieklas testen, boundary testen of random testen.

Door de verschillende functies, die een gebruiker binnen de ASF+SDF MetaEnvironment kan verrichten te specificeren, ontstaat een lijst met acties, die kunnen worden opgenomen. De condities, die gespecificeerd worden voor de verschillende acties kunnen tijdens het opnemen worden opgegeven doormiddel van de acties van de tester. Naderhand is tijdens de playbackfase de output van het systeem te vergelijken met de output die verwacht wordt.

Veel andere teststrategieën, zoals onder andere die beschreven zijn in [7, 8, 12], gaan er van uit dat de statecharts, architectuur beschrijvingen of de volledige systeemanalyse beschikbaar is. Dit is niet het geval en deze teststrategieën zijn dan ook niet bruikbaar.

3.2 ToolBus communicatie

Binnen een ToolBus applicatie zijn verschillende manieren van communicatie mogelijk (zie figuur 2). Dit valt op te delen in communicatie tussen processen en communicatie tussen een proces en een tool. De communicatie tussen processen kan zowel synchroon als asynchroon plaatsvinden [1]. Bij de synchrone communicatie moet er rekening gehouden worden met het feit, dat de processen en tools, die met de communicatie te maken hebben, wel klaar moeten staan om het bericht te ontvangen. Als dat niet het geval is, kan een deadlock situatie optreden.

In de voorbeelden hieronder van de aanroepen van de communicatiefuncties, worden de termen Tid en ATerm gebruikt. Tid is de tool identifier en ATerm is het datatype, dat gebruikt wordt om data te versturen.

3.2.1 Communicatie tussen processen

De communicatie tussen processen kan zowel synchroon als asynchroon plaatsvinden [1].

Message passing

De synchrone communicatie vindt plaats door het versturen van berichten. Hierbij stuurt een proces een bericht naar een ander proces, dat daar op dat moment op staat te wachten. Door middel van matching controleert het ontvangende proces, of het bericht daadwerkelijk voor hem bedoeld is. Hierbij wordt het inkomende bericht vergeleken met een patroon, dat gedefinieerd staat in het T-script.

Deze manier van communicatie wordt verwezenlijkt door de volgende twee functies:

Bericht	Uitleg	Locatie
`snd-msg(ATerm)`	Het versturen van een bericht	T-script
`rec-msg(ATerm)`	Het ontvangen van een bericht	T-script

Broadcasting

De asynchrone communicatie vindt plaats doormiddel van de publish subscribe methode. Hierbij geven de processen aan op welke diensten ze zich willen inschrijven. Zodra een proces een bericht heeft, dat voor meerdere processen bedoelt kan zijn, wordt dit bericht via broadcasting naar alle processen gestuurd die zich voor het bericht hebben ingeschreven. Deze worden vervolgens in de queue van het ontvangende proces geplaatst. Zodra het proces klaar is met zijn verwerking, wordt gecontroleerd of er nog notes in de queue staan. Als dat het geval is, wordt de note verwerkt.

Bericht	Uitleg	Locatie
`subscribe(patroon)`	Het inschrijven voor een dienst	T-script
`snd-note(ATerm)`	Het versturen van een broadcastbericht	T-script
`Rec-note(ATerm)`	Het ontvangen van een broadcastbericht	T-script

3.2.2 Communicatie tussen proces en tool

De communicatie tussen een proces en een tool is altijd synchroon. Er zijn hierbij drie verschillende manieren van communicatie mogelijk.

Evaluatie

Bij deze manier van communicatie wordt een evaluatieaanvraag van het proces naar de tool verstuurd. Deze wordt beantwoord door de tool met een waarde. In de tussenliggende tijd kan de tool zijn verwerking verrichten. Bij deze manier van communicatie wordt altijd het initiatief genomen door het proces, waarbij het proces een antwoord terugkrijgt van de tool.

Bericht	Uitleg	Locatie
`snd-eval(Tid, ATerm)`	Verstuurt verzoek voor een berekening	T-script
`rec-eval(ATerm)`	Ontvangt verzoek voor een berekening	Tool
`snd-value(ATerm)`	Verstuurt de feedback op een evaluatieverzoek	Tool
`rec-value(Tid, ATerm)`	Ontvangt de feedback op een evaluatieverzoek	T-script

Do

Bij deze communicatievorm wordt een uitvoer commando door het proces naar de tool verzonden. Hierbij geeft het proces de tool een opdracht om een actie uit te voeren, waarbij er geen antwoord wordt terug verwacht.

Bericht	Uitleg	Locatie
`snd-do(Tid, ATerm)`	Verstuurt een uitvoeropdracht	T-script
`rec-do(ATerm)`	Ontvangt een uitvoeropdracht	Tool

Event

De tools kunnen events genereren (bijvoorbeeld een druk op een knop in een user interface). Hier wordt het initiatief genomen door de tool, die het proces van een actie op de hoogte stelt. Het proces geeft vervolgens een bevestiging van ontvangst.

Bericht	Uitleg	Locatie
`snd-event(ATerm)`	Verstuurt een event	Tool
`rec-event(Tid, ATerm)`	Ontvangt een event	T-script
`snd-ack-event(Tid, ATerm)`	Verstuurt een bevestiging van een event	T-script
`rec-ack-event(ATerm)`	Ontvangt een bevestiging van een event	Tool

3.2.3 Het monitoren van berichten

Het is mogelijk om een logger applicatie aan een ToolBus applicatie te koppelen. Zodra deze aan de ToolBus is verbonden, worden alle berichten die tussen de componenten verstuurd worden, ook naar de logger gestuurd. Dit zijn zowel de berichten tussen processen, als tussen processen en tools. Deze berichten komen binnen met de ATerm logpoint().

In de logpoint wordt verschillende informatie opgeslagen, onder andere de proces id van de verstuurder, het proces of de tool waar het bericht naartoe gestuurd moet worden, het patroon van het bericht, de waardes van alle variabelen, die binnen het T-script van de verstuurder gebruikt worden, bestandsnaam en coördinaten waar het T-script zich op de computer bevindt.

Voor de integratietest is slechts een fractie van dit bericht van belang. Alleen het patroon van het bericht met de waardes van de variabelen, die hierin worden gebruikt, is nodig. Hiermee kan het bericht gevormd worden, dat tijdens het imiteren van tools gebruikt moet worden. Ook kan dit bericht tijdens de playbackfase gebruikt worden, om te controleren of in het bericht, dat dan verstuurd wordt, de variabelen dezelfde waardes hebben.

Zodra de logger een logpoint heeft ontvangen, kan deze hier bewerkingen op uitvoeren. Het is echter niet mogelijk om het bericht te wijzigen en vervolgens naar de ontvanger te sturen. Zodra de logger klaar is met zijn bewerkingen op het bericht, moet de logger een bericht naar de ToolBus sturen, zodat deze weet dat het verder kan.

Het is voor de integratietest de bedoeling, dat alleen de communicatie van een aantal componenten gecontroleerd wordt. Het is mogelijk om de communicatie van alle componenten te controleren, maar dit zorgt voor een enorm performance verlies. Daarnaast is deze test ook veel preciezer dan nodig. Als alle componenten gecontroleerd worden, moet zodra er maar iets binnen de ASF+SDF MetaEnvironment aangepast wordt, verschillende testgevallen opnieuw opgenomen worden.

Voor de testopstelling die is gemaakt, is gekozen om in eerste instantie alleen de berichten van een drietal tools met hun processen te monitoren. Dit zijn de editor-hive (hieraan worden bestaande editors gekoppeld), de input / output (hiermee worden alle lees en schrijf acties naar bestanden gedaan) en de GUI (de grafische interface van de ASF+SDF MetaEnvironment). Dit zijn de punten waar de gebruiker zelf merkt, wat het systeem doet en op deze punten kan een gebruiker dan ook fouten constateren.

BAF berichten

Tijdens het opnemen van de test is het mogelijk dat er berichten met ATermen in binair formaat ontvangen worden (Binary ATerm Format). De verschillende onderdelen van de integratietest zijn ontwikkeld in JAVA. De JAVA implementatie van de ATerm ondersteunt echter niet het BAF formaat. Daarom moeten de BAF gedeeltes uit de berichten gefilterd worden. Deze zullen vervolgens vervangen worden door hun hashwaarde, zodat uit de berichten nog wel af te leiden is, of er BAF gedeeltes in hebben gezeten.

Dit gebeurt door in berichten het gedeelte:

```
_baf-encoded_(“binaire informatie”)
```

te vervangen door

```
_baf-recoded_(“hashwaarde binaire informatie”)
```

Vervolgens worden er bestanden gecreëerd met als naam de hashwaarde van het bericht en met de extensie ".baf". Zodra deze berichten in de playbackfase nodig zijn bij het versturen van events of het beantwoorden van evaluatiebezoeken, kan de BAF waarde weer in het bericht teruggeplaatst worden.

Secundaire communicatie tussen de metaplaybacker en de imitatortools

Tijdens de playbackfase moet de metaplaybacker bijhouden, of er events gegenereerd moeten worden. De imitatortools kunnen dit zelf niet, aangezien deze geen kennis hebben van de communicatie tussen de overige componenten. De metaplaybacker kan echter geen berichten versturen naar de imitatortools. Om toch te communiceren met de imitatortools, wordt er gebruik gemaakt van een gemeenschappelijk bestand. De metaplaybacker zet hierin de event die gegenereerd moeten worden of de evaluatiefeedback die opgestuurd moet worden.

Zodra de metaplaybacker gedurende een vooraf bepaald aantal seconden, geen berichten binnen heeft gekregen, wordt er gecontroleerd of er een event gegenereerd moet worden. Is dit het geval, dan wordt dat event in het gemeenschappelijke bestand genoteerd. De imitatortools controleren dit bestand elke seconde. Mocht een imitatortool zien, dat het bericht voor hem bedoeld is, dan genereert deze het event, dat verwacht wordt.

Mocht de metaplaybacker een verzoek voor een evaluatie voor een imitatortool binnenkrijgen, dan wordt direct in het gemeenschappelijke bestand aangegeven welke feedback er teruggestuurd moet worden.

Er is gekozen om niet de communicatie tussen componenten te gebruiken om de momenten te bepalen waarop events verstuurd moeten worden. Hierdoor is het genereren van events onafhankelijk van de communicatie tussen componenten, waarvan de berichten niet worden opgeslagen. Het voordeel hiervan, is dat bij het aanpassen van T-scripts van componenten, die geen deel uitmaken van de integratietest, de integratietest niet aangepast hoeft te worden. Dit zou anders wel nodig geweest zijn om er voor te zorgen, dat de integratietest de nieuwe momenten weet waarop events gegenereerd moeten worden.

3.3 Het parameteriseren van de integratietest

Bij het bepalen van de onderdelen, die getest moeten worden door de integratietest, is het van belang om aan te kunnen geven tussen welke processen en tools de communicatie moet worden gecontroleerd. Om dit mogelijk te maken moet de logger de ontvangen berichten kunnen filteren.

Alle berichten zijn voorzien van een proces id. Dit is een unieke identifier en hierop kan goed gefilterd worden. Het probleem is echter dat de proces id kan veranderen zodra de componenten van een ToolBus in een andere volgorde worden opgestart, of als er componenten worden toegevoegd of verwijderd. Wat niet verandert zijn de procesnamen. Tijdens het opstarten van de ToolBus applicatie kunnen de logpoints, die de "create" berichten bevatten, geanalyseerd worden. Hierin staat zowel de procesnaam als de proces id vermeld. Op die manier kan de gebruiker parameteriseren op procesnaam, terwijl intern de proces id gebruikt kan worden.

Op het moment, dat een opname van de ASF+SDF MetaEnvironment gemaakt wordt, kunnen voordat de opname wordt gestart, de processen opgegeven worden die moeten worden bijgehouden. Hierbij moet er wel op gelet worden, dat deze lijst hoofdlettergevoelig is. Om te voorkomen, dat er fouten gemaakt worden tijdens het intypen, worden de standaard processen weergegeven op het scherm. De gebruiker kan vervolgens de gewenste processen selecteren. Daarnaast is er een invoerveld om eventuele nieuwe componenten op te nemen. De lijst met geselecteerde processen wordt opgeslagen bij de informatie van de opname. Tijdens het genereren van een testomgeving en het afspelen van de test kunnen daardoor dezelfde filters weer gebruikt worden.

3.4 Het bepalen van equivalentie bij niet deterministische berichten

Bij een systeem met niet deterministische berichten, is de verwerking van het systeem onafhankelijk van de volgorde van de intern verstuurde berichten. Binnen de ToolBus applicaties werken alle componenten onafhankelijk. Daardoor kan het zijn dat de volgorde van de berichten bij het voor de tweede maal uitvoeren van dezelfde handelingen anders is.

Voor de te ontwikkelen integratietest heeft dit als gevolg dat de ontvangen berichten tijdens de playbacksessie niet in dezelfde volgorde hoeven te staan als de ontvangen berichten tijdens de eerdere opnamesessie. Tijdens de playbackfase zullen alle berichten, die nog niet gemarkeerd zijn als “correct herhaald”, vergeleken moeten worden met het inkomende bericht. Hierdoor is het na het uitvoeren van een test mogelijk om te bepalen of alle berichten daadwerkelijk herhaald zijn en of er geen andere berichten zijn verstuurd.

Er zijn echter een aantal berichten, die veranderen als ze op een ander moment verstuurd worden. Dit is goed te zien in de volgende berichten van de user interface:

- add-statusf(49,”Opening %s”,[“SL-verwerking”])
- end-status(49)
- add-status(51,”Opening modules”)

Door het niet deterministische karakter van de ToolBus, kunnen modules in een andere volgorde geopend worden als tijdens de opname. De identifier van het bericht kan in dat geval anders zijn voor hetzelfde bericht. Zo kan *add-statusf(49,”Opening %s”,[“SL-verwerking”])* tijdens de playbackfase terugkomen als *add-statusf(50,”Opening %s”,[“SL-verwerking”])*. Om deze berichten toch te kunnen goedkeuren, zal de filter van de metaplaybacker dit soort berichten moeten herkennen. Dit kan door patronen te definiëren waar deze veranderlijke berichten aan moeten voldoen.

4 De ontwikkelde tools

In dit hoofdstuk worden de tools beschreven, die nodig zijn voor de integratietest voor de ASF+SDF MetaEnvironment. Voor deze integratietest zijn vier verschillende applicaties nodig:

1. De metalogger
2. De imitatortool
3. De testgenerator
4. De metaplaybacker

De metalogger is een logger applicatie, die de communicatie tijdens de opnamefase opslaat. Hierbij kan de gebruiker de processen opgeven, die tijdens de opname opgeslagen moeten worden. Deze informatie wordt vervolgens gebruikt om imitatortools te genereren. Deze tools worden gebruikt om de originele tools te vervangen. Hierdoor wordt het mogelijk om de integratietest zonder gebruikersinteractie uit te voeren.

Tijdens de playbackfase wordt de communicatie, die binnen de ASF+SDF MetaEnvironment plaatsvindt, door de metaplaybacker vergeleken met de eerder opgenomen berichten. Aan de hand van deze gegevens kan er bepaald worden, of de samenwerking tussen de componenten tijdens de playbackfase nog hetzelfde is als tijdens de opnamefase.

Hieronder wordt elke tool afzonderlijk besproken. In het volgende hoofdstuk is vervolgens te lezen hoe hiermee de integratietest uit te voeren is.

4.1 De metalogger

De metalogger is een logger applicatie voor de ASF+SDF MetaEnvironment, die de berichten tussen de geselecteerde processen en de bijbehorende tools opslaat. Deze applicatie wordt als eerste onderdeel van de integratietest uitgevoerd en vervult de opnamefase van de test.

4.1.1 De functionaliteit

De metalogger filtert de inkomende berichten en slaat de berichten, die nodig zijn voor de integratietest, op. Zoals in 3.2.3 vermeld staat, is het niet de bedoeling, dat de communicatie met alle tools bijgehouden wordt, maar alleen van de geselecteerde tools. De metalogger begint daarom met een scherm waarmee het filter in te stellen is. Deze zorgt ervoor, dat alleen de communicatie van de gekozen tools opgeslagen wordt.

De metalogger slaat de volgende informatie op:

1. Hoofdbestand
2. Filterlijst
3. Lijst met events en evaluatieverzoeken
4. Communicatie per proces
5. BAF berichten

In het hoofdbestand wordt een lijst bijgehouden, waarin beschreven staat welke bestanden er nodig zijn voor de integratietest. Hierdoor kunnen de testgenerator en de metaplaybacker eenvoudig achterhalen welke bestanden er gebruikt worden door de integratietest.

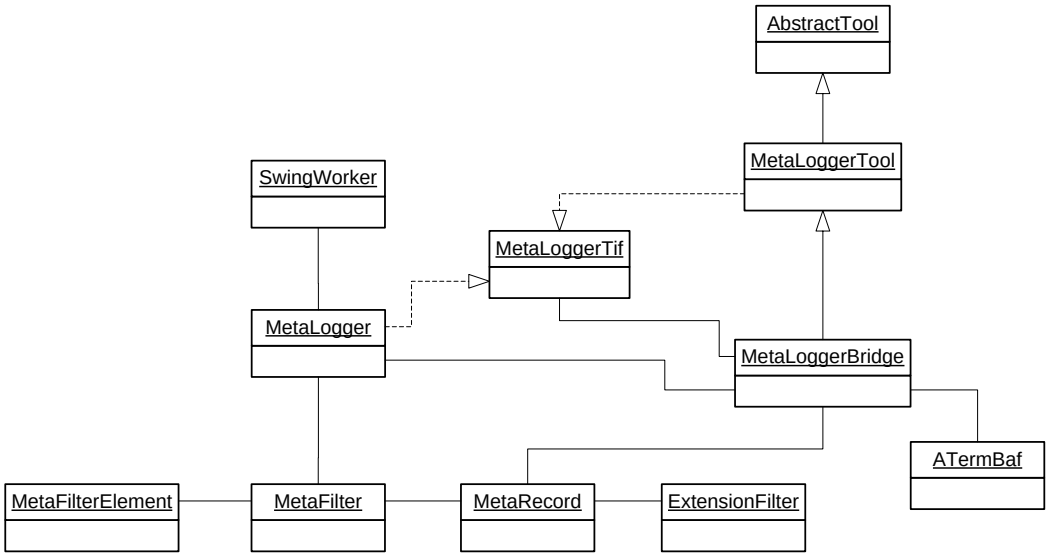
De filterlijst wordt bijgehouden en opgeslagen om er voor te zorgen, dat de juiste processen tijdens de playbackfase worden gecontroleerd. Daarnaast wordt deze lijst ook gebruikt tijdens het genereren van de testomgeving. Aan de hand van deze lijst kan namelijk bepaald worden welke tools vervangen moeten worden door imitatortools.

De lijst met events en evaluatieverzoeken kan tijdens de playbackfase gebruikt worden om op het juiste moment een event te laten plaatsvinden. De evaluatieverzoeken worden bijgehouden om de imitatortools het correcte antwoord op een verzoek te laten terugsturen. Aan de hand van de lijst worden deze gebeurtenissen op de juiste volgorde uitgevoerd.

De communicatie van elk proces met de bijbehorende tool wordt opgeslagen in een apart bestand. Tijdens de playbackfase worden de op dat moment ontvangen berichten vergeleken met de berichten uit deze bestanden. De imitatortools gebruiken op dat moment de informatie uit deze bestanden om de events en de feedback te genereren, die de ASF+SDF MetaEnvironment verwacht. De gedeeltes van berichten, die langskomen in BAF formaat, worden als losse bestanden opgeslagen.

4.1.2 De implementatie

Het klassendiagram van de metalogger ziet er als volgt uit:



Figuur 4: Klassendiagram van de metalogger

De mainklasse van de metalogger is MetaLogger. Deze heeft de standaard bridge bestanden MetaLoggerTif, MetaLoggerTool en MetaLoggerBridge, die nodig zijn voor het communiceren met de ToolBus [16]. Voordat er begonnen wordt met een opname, is het mogelijk om de processen op te geven die gefilterd moeten worden. Deze processen worden vervolgens in de filter (MetaFilter) als filterelementen (MetaFilterElement) opgeslagen .

De metalogger kan maar twee verschillende soorten berichten ontvangen. Dit zijn de rec-terminate(ATerm) en de rec-monitor(ATerm). Zodra de rec-terminate binnenkomt wordt de applicatie afgesloten. De rec-monitor ontvangt alle berichten, die in de ToolBus verstuurd worden, in de vorm van logpoints. Als dit berichten zijn van tools, die in de filter voorkomen en als de berichten tussen een proces en de bijbehorende tool verstuurd worden, wordt het oorspronkelijke bericht gereconstrueerd (3.2.3) en opgeslagen. Dit gebeurt door de MetaRecord klasse. Deze heeft tevens toegang tot de bridge om eventuele BAF berichten op te halen, die zijn vervangen door een hashwaarde.

Zodra de bridge opgestart wordt met de run() functie, komt de bridge in een oneindige loop. Er wordt dan continue gecontroleerd of er een nieuw bericht ontvangen wordt. Om de logger functionaliteit van de metalogger nog te kunnen gebruiken, moeten deze twee processen in aparte threads uitgevoerd worden. De SwingWorker zorgt ervoor, dat meerdere threads opgestart kunnen worden binnen de metalogger.

De AbstractTool, die normaal gesproken gebruikt wordt bij ToolBus applicaties, is vervangen. Dit om het filteren van BAF berichten mogelijk te maken. De ontvangen BAF berichten worden in een lijst opgeslagen (ATermBaf).

De ExtensionFilter zorgt ervoor dat bij het openen van het hoofdbestand, alleen de bestanden met de extensie "*.*rp*" zichtbaar zijn.

4.2 De imitatortool

De imitatortools worden gebruikt om tools, waarvan de communicatie tijdens de opname opgeslagen is, te imiteren. Hierdoor is het mogelijk om events automatisch te laten genereren, zonder dat hier interactie met de gebruiker voor nodig is. Dit maakt het mogelijk om de integratietest tijdens de DailyBuild automatisch uit te voeren.

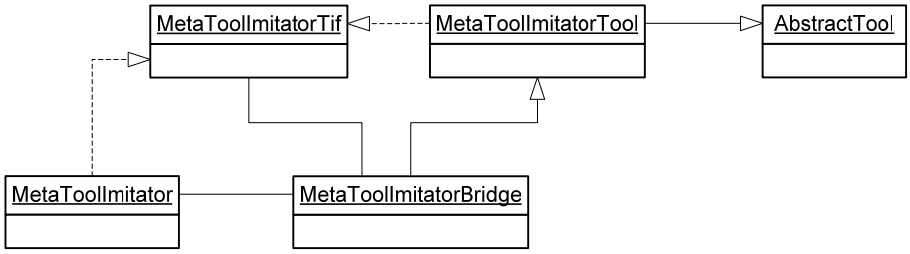
4.2.1 De functionaliteit

Tijdens de playbackfase is het de bedoeling dat de test automatisch wordt herhaald, zonder dat gebruikersinteractie met het systeem nodig is. Om dit mogelijk te maken, worden er tijdens het genereren van de testomgeving een aantal imitatortools gecreëerd.

Zodra een imitatortool wordt opgestart, leest deze een bestand uit, waarin alle instellingen van de imitatortool zijn vastgelegd. Hierdoor weet de imitatortool waar het gemeenschappelijke bestand opgeslagen is om te kunnen communiceren met de metaplaybacker en welke tool het moet imiteren.

Om te bepalen of de imitatortool een event moet genereren, wordt om de seconde het gemeenschappelijke bestand gecontroleerd, waarin de metaplaybacker aangeeft wat er moet gebeuren. In dit bestand wordt ook de feedback op evaluatie aanvragen opgeslagen.

4.2.2 De implementatie



Figuur 5: Klassendiagram van de imitatortool

De imitatortool is een heel eenvoudige applicatie. Het enige wat gebeurt, is dat de mainklasse MetaToolImitator om de seconde controleert of er een event gegenereerd moet worden, of dat er een evaluatiebericht verzonden moet worden. Daarnaast maakt het gebruik van de standaard gegenereerde bridge bestanden, MetaToolImitatorTif, MetaToolImitatorTool en MetaToolImitatorBridge om de communicatie met de ToolBus mogelijk te maken [16].

De AbstractTool, die normaal gesproken gebruikt wordt bij ToolBus applicaties, is vervangen. Dit om het filteren van BAF berichten mogelijk te maken.

4.3 De testgenerator

Om de integratietest uit te kunnen voeren is een testomgeving nodig. Hiervoor moeten de imitatortools, die hiervoor beschreven zijn, gegenereerd worden. Dit proces gebeurt automatisch, zodat het eenvoudig te herhalen is op het moment, dat een T-script van een tool wordt aangepast.

4.3.1 De functionaliteit

Tijdens de playbackfase moeten de verschillende tools, die deel uitmaken van de integratietest, geïmiteerd worden. Dit om ervoor te zorgen dat de tools, die events genereren door gebruikersinteractie, deze events automatisch versturen. De imitatortools zijn allemaal nagenoeg hetzelfde, met een paar kleine verschillen in de standaard gegenereerde tif, tool en bridge bestanden [16]. Hierom is er ook gekozen om deze tools automatisch te genereren. Daarnaast zorgt dit ervoor, dat een aanpassing in een T-script er niet voor zorgt, dat de imitatortools aangepast moeten worden. In plaats daarvan kunnen nieuwe imitatortools gegenereerd worden.

Het generatieproces heeft een aantal fases:

1. Het bepalen van de namen van de tools.
2. Het filteren van de T-scripts.
3. Het genereren van de bridge bestanden.
4. Het samenstellen van de applicatie.
5. Het genereren van het instellingenbestand.
6. Het genereren van het startbestand.

Om de ASF+SDF MetaEnvironment tijdens de playbackfase de imitatortools te laten opstarten in plaats van de originele tools, moet de PATH variabele aangepast worden. Hierin moeten de imitatortools opgenomen worden, welke dezelfde naam moeten hebben als de originele tools. Deze worden dan in plaats van de originele tools opgestart. Een andere optie is om de T-scripts van de ASF+SDF MetaEnvironment zo aan te passen, dat de tooldeclaraties een ander bestand aanroepen. Dit moet dan vervolgens na de playbackfase weer ongedaan worden gemaakt. Het is eenvoudiger om de imitatortools dezelfde namen te geven als de originele tools.

Elke imitatortool moet een bridge hebben conform het T-script van het proces wat het imiteert. Dit om ervoor te zorgen dat de tool geaccepteerd wordt tijdens het controleren van de signatures. Om deze bridge bestanden te genereren moeten deze vanuit het T-script van het te imiteren proces gegenereerd worden. Om de nodige T-scripts te vinden, worden vanuit het T-script waarin de Meta Studio wordt gestart, alle geïmporteerde T-scripts geanalyseerd, die onderdeel uitmaken van de ASF+SDF MetaEnvironment.

Zodra de imitatortool specifieke bridge is gegenereerd, kan de applicatie worden samengesteld met de standaard imitator bestanden en deze kan vervolgens gebruikt worden.

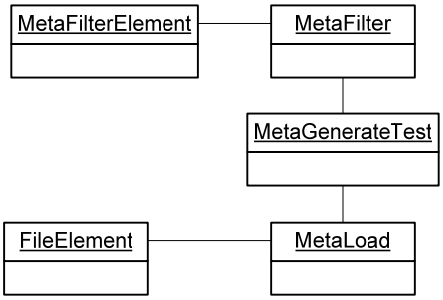
De imitatortool heeft daarnaast nog een aantal gegevens nodig voordat het kan worden gebruikt. Dit zijn:

- De naam van het proces dat geïmiteerd moet worden.
- Het bestand dat door de metaplaybacker gebruikt wordt om te communiceren met de imitatortools.

Deze gegevens worden opgeslagen in een instellingenbestand, die bij de toolimitator wordt opgeslagen.

Voor het starten van de integratietest is een nieuw opstartbestand nodig. Dit bestand moet de PATH variabele aanpassen, zodat de juiste tools opgestart worden.

4.3.2 De implementatie



Figuur 6: Klassendiagram van de testgenerator

De mainklasse van de testgenerator is MetaGenerateTest. Deze klasse doet de meeste verwerking van het generatieproces, zoals hiervoor beschreven is. Het maakt gebruik van de MetaLoad klasse om alle bestanden in te laden, die tijdens de integratietest gebruikt worden. De gegevens van de bestanden worden opgeslagen in een lijst (FileElement). Daarnaast maakt de mainklasse gebruik van een filter (MetaFilter) met filterelementen (MetaFilterElement), waarin bijgehouden wordt welke filters er gebruikt zijn tijdens de opnamefase. Hierdoor kan bepaald worden welke imitatortools gegenereerd moeten worden. Hierbij wordt gekeken, welke tools events genereren. De tools die geen events genereren, hoeven niet vervangen te worden door imitatortools. Hier kunnen de originele tools gebruikt worden.

4.4 De metaplaybacker

De metaplaybacker is een logger applicatie, die tijdens de playbackfase gebruikt wordt. Het controleert, of de communicatie tijdens deze fase nog hetzelfde is als tijdens de opnamefase. Dit is dan ook de tool, die de daadwerkelijke integratie van de ASF+SDF MetaEnvironment test.

4.4.1 De functionaliteit

Zodra een integratietest gestart wordt, moet er gecontroleerd worden of dezelfde berichten binnen de ASF+SDF MetaEnvironment rondgestuurd worden als tijdens de opname. Hiervoor is de metaplaybacker ontwikkeld. Deze applicatie is, net zoals de metalogger, een ToolBus logger. Deze slaat echter niet alle informatie op in verschillende bestanden, maar controleert of de communicatie identiek is aan de communicatie tijdens de opname.

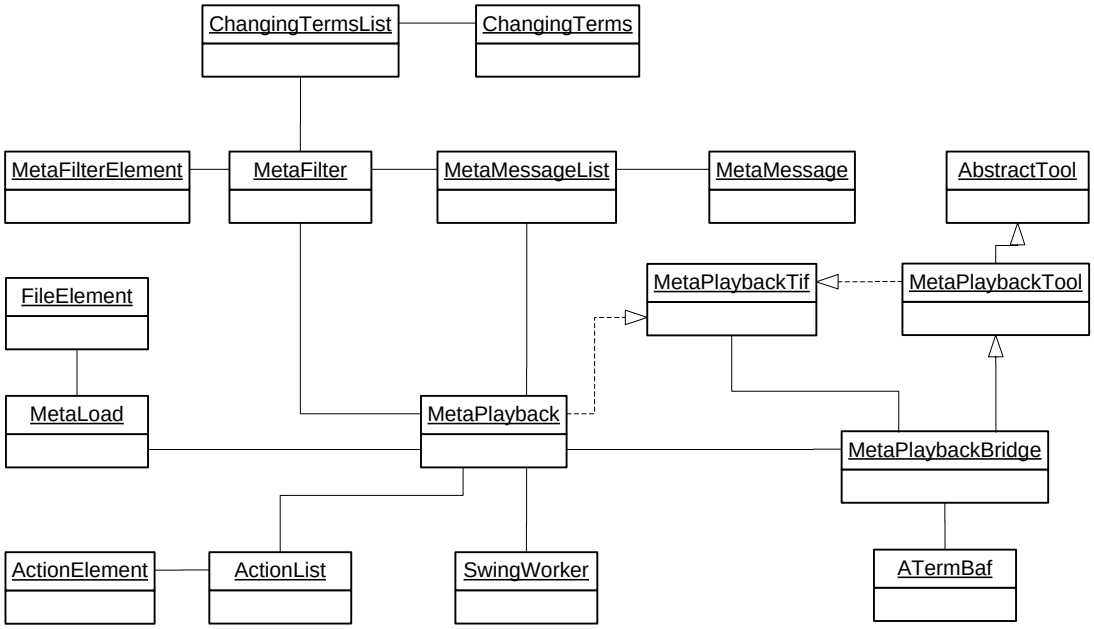
Net zoals in de metalogger zit in de metaplaybacker een filter, die controleert of de ontvangen berichten van toepassing zijn voor de integratietest. De processen, die gecontroleerd worden, worden hier echter niet bij het starten van de test opgegeven, zoals bij de metalogger, maar worden uitgelezen uit het filterbestand dat tijdens de opname gemaakt is. Komt het bericht van een van de componenten, die in de filterlijst voorkomt, dan wordt het originele bericht gereconstrueerd uit de ontvangen logpoint (zie 3.2.3).

Zodra een bericht overeenkomt met een bericht dat tijdens de opname is ontvangen, wordt het gemarkeerd als correct. De berichten die niet direct als correct gemarkeerd kunnen worden, worden verder gecontroleerd. Hierbij wordt gekeken of het geen veranderlijke berichten zijn, zoals in 3.4 is beschreven. Hiervoor worden patronen gebruikt waaraan de berichten moeten voldoen.

Op deze manier wordt bijgehouden welke berichten wel worden verwacht, maar niet binnen zijn gekomen. Daarnaast worden ook alle berichten opgeslagen, die binnen zijn gekomen, maar niet in de opname voorkomen.

De metaplaybacker communiceert via een gemeenschappelijk bestand met de imitatortools. Dit is nodig om de imitatortools de juiste feedback te laten versturen op een evaluatieverzoek. Daarnaast wordt, als er voor een bepaald aantal seconden geen bericht via de ToolBus is verstuurd, gecontroleerd of er een event gegenereerd moet worden. Als dit het geval is, wordt dit in het gemeenschappelijke bestand genoteerd. Een alternatief zou zijn om tijdens de opname een analyse te maken van de communicatie tussen alle componenten, om aan de hand van deze gegevens te bepalen wanneer een event gegenereerd zou moeten worden. Het nadeel hiervan is, dat de opgenomen integratietest op dat moment afhankelijk is van componenten, die buiten de test vallen. Mochten die componenten aangepast worden, dan moet de testset worden vernieuwd.

4.4.2 De implementatie



Figuur 7: Klassendiagram van de metaplaybacker

De mainklasse van de metaplaybacker is MetaPlayback. Deze heeft de drie standaard bestanden om de connectie met de ToolBus mogelijk te maken. Dit zijn de MetaPlaybackTif, MetaPlaybackTool en de MetaPlaybackBridge.

Zodra een test gestart wordt, worden alle gegevens geladen, die voor de test van toepassing zijn. Dit wordt gedaan door de klasse MetaLoad. Deze klasse slaat de informatie van de gebruikte bestanden vervolgens op in een lijst van het type FileElement.

Zodra de bestanden, die gebruikt moeten worden zijn ingeladen, wordt de filterlijst (MetaFilter) gevuld met filterelementen (MetaFilterElement), de berichtenlijst (MetaMessageList) wordt gevuld met berichten, die tijdens de recordfase zijn opgeslagen (MetaMessage) en de lijst met acties (ActionList) wordt met acties gevuld (ActionElement). De filter maakt gebruik van een tweede berichtenlijst. Hierin worden alle berichten, die niet tijdens de opname voorkwamen, opgeslagen. Daarnaast maakt de filter ook gebruik van een verzameling patronen (ChangingTermList met ChangingTerms) om te controleren of een afgekeurd bericht voldoet aan de voorwaarden van een veranderlijk bericht, zoals is uitgelegd in 3.4.

Net als bij de metalogger, moeten de logger functionaliteit en de bridge in aparte threads worden uitgevoerd. Hiervoor wordt de SwingWorker klasse gebruikt.

Wederom wordt een aangepaste AbstractTool gebruikt, zodat de BAF gedeeltes gefilterd kunnen worden.

5 Integratietest voor de ASF+SDF MetaEnvironment

Nadat de tools die in hoofdstuk 4 zijn beschreven, zijn ontwikkeld, kan een integratietest uitgevoerd worden. Het ontwikkelen van een integratietest voor de ASF+SDF MetaEnvironment gebeurt in drie stappen.

Als eerste wordt een opname gemaakt van de communicatie tussen de gekozen componenten van de ASF+SDF MetaEnvironment met behulp van de metalogger. Vervolgens wordt de testomgeving gegenereerd. In deze omgeving zijn tools van de ASF+SDF MetaEnvironment, die events genereren vervangen door imitatortools. Dit om ervoor te zorgen dat gebruikersinteractie tijdens de integratietest niet nodig is.

Zodra een testomgeving is gegenereerd, kan een playbacksessie gedaan worden met behulp van de metaplaybacker. Deze controleert of de communicatie tussen de geselecteerde processen en tools nog hetzelfde is als tijdens de opname.

5.1 De testsituatie

De eerste testsituatie, waarvoor een integratietest gewenst is, is een integratietest die de grafische user interface, de editor-hive en de input / output tool controleert. Deze drie tools hebben de meeste interactie met de gebruiker en ze ontvangen daarom veel informatie over de verwerking, die binnen de ASF+SDF MetaEnvironment plaatsvindt. Daarnaast worden veel events vanuit de grafische user interface en de editor-hive verstuurt.

Door deze drie tools in een integratietest te betrekken, is er voor een groot deel van de acties automatisch te controleren of de integratie van de componenten correct is. Het is ook mogelijk om de communicatie van alle tools op te slaan, maar dit is een te precieze test. Bij elke kleine aanpassing binnen de ASF+SDF MetaEnvironment zou dan een nieuwe testset opgenomen moeten worden. Om te demonstreren hoe dit in zijn werk gaat, zal het proces van het maken van een integratietest voor deze drie tools beschreven worden.

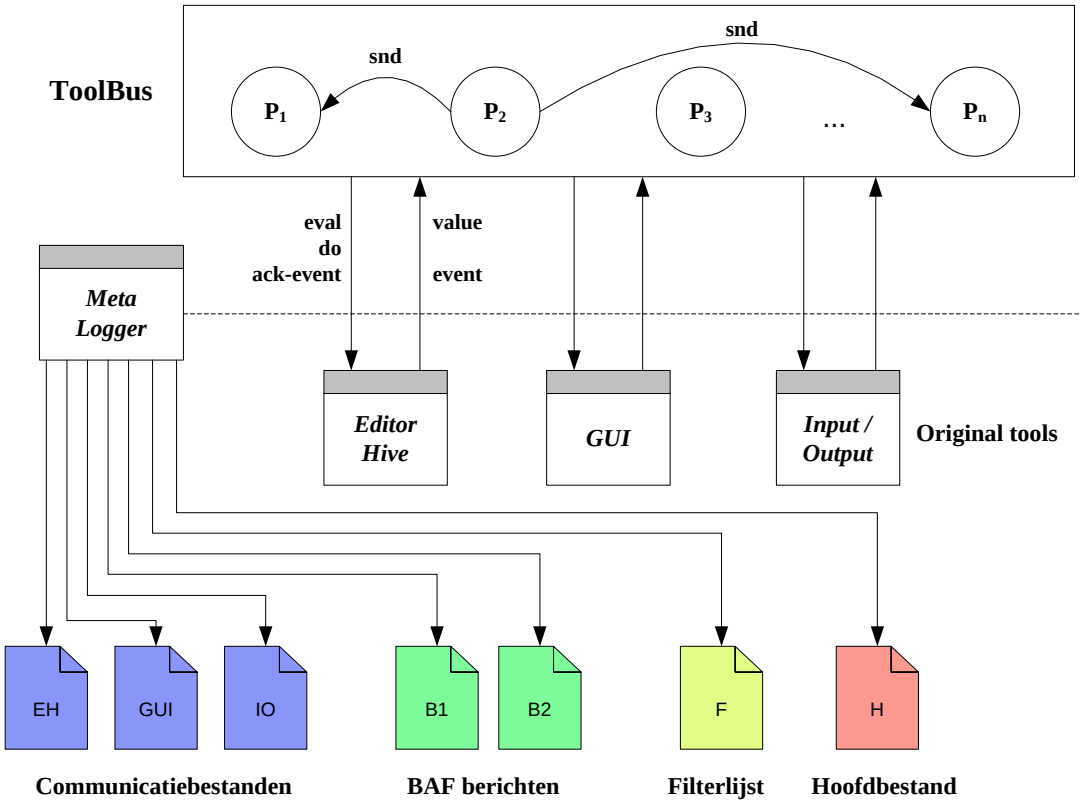
5.2 Het opnemen van een integratietest

Tijdens het opnemen van een integratietest, wordt door de metalogger de communicatie tussen de tools (in dit geval de editor-hive, grafische user interface en de input / output) en de bijbehorende processen opgeslagen in verschillende communicatiebestanden, waarbij in elk communicatiebestand de volledige communicatie van één tool wordt opgeslagen. Meer over de metalogger is te lezen in hoofdstuk 4.1.

Daarnaast controleert de metalogger of de berichten BAF gedeeltes bevatten. Deze BAF gedeeltes moeten worden opgeslagen, zodat deze tijdens het uitvoeren van de test weer gebruikt kunnen worden.

Tijdens het opnemen van de integratietest, wordt de lijst met processen die gefilterd moeten worden, opgeslagen. Deze informatie wordt gebruikt bij het maken van de testomgeving en bij het uitvoeren van de test.

Tenslotte wordt er in een bestand bijgehouden welke bestanden er gebruikt moeten worden voor de test en welk type informatie deze bestanden bevatten. Deze informatie is zowel nodig bij het genereren van de testomgeving als bij het herhalen van de test. Dit proces is te zien in figuur 8.



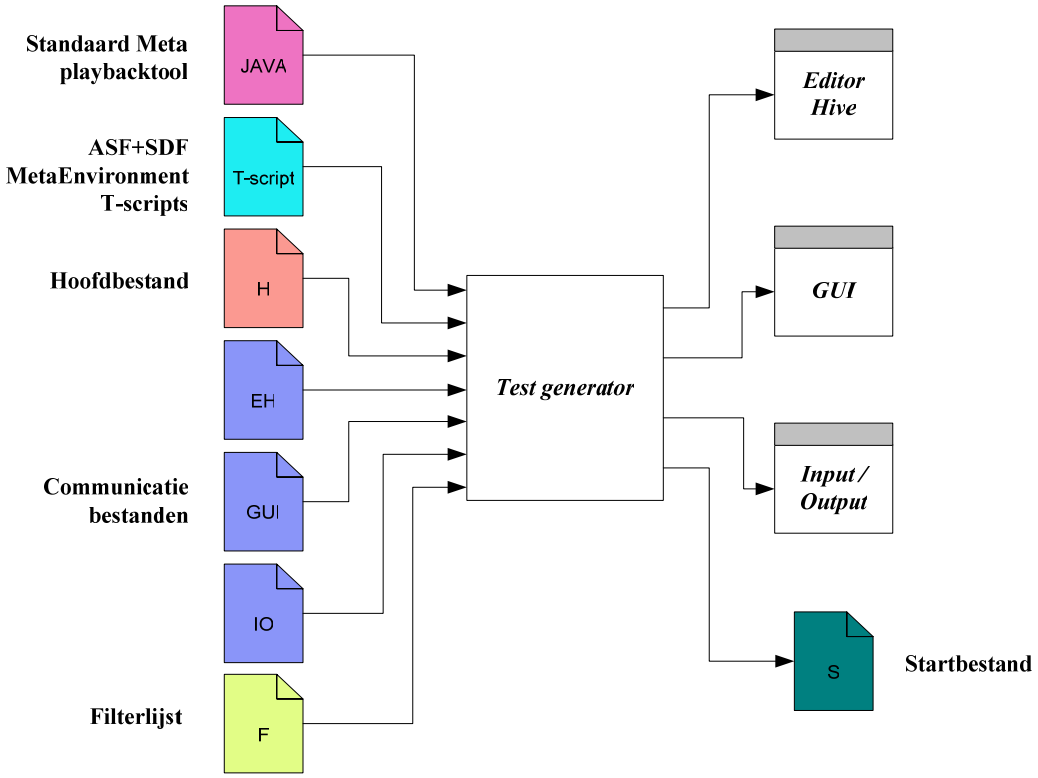
Figuur 8: Opnemen van een integratietest

5.3 Het creëren van een testomgeving

Zodra een integratietest is opgenomen, kan de bijbehorende testomgeving worden gecreëerd om de test uit te voeren. Dit proces is te zien in figuur 9. Dit is nodig omdat de originele tools vervangen moeten worden door imitatortools, zodat de test automatisch kan verlopen.

Als eerste leest de testgenerator het hoofdbestand van de opname uit. Hierdoor weet de generator welke bestanden er gebruikt worden. De testgenerator leest vervolgens de filterlijst uit. Met de gegevens hieruit worden de namen van de te genereren tools opgezocht binnen de T-scripts, die door de ASF+SDF MetaEnvironment worden gebruikt. Vervolgens zoekt de testgenerator uit welke tools events kunnen genereren. Hierdoor weet de testgenerator dat er voor de editor-hive en de grafische user interface imitatortools gegenereerd moeten worden, maar dat de input / output niet vervangen hoeft te worden. (De testgenerator en de imitatortool worden in hoofdstuk 4 beschreven).

Daarna worden de delen van de T-scripts, die bij de grafische user interface en de editor-hive horen, gekopieerd. Deze worden gebruikt om de standaard MetaImitatorTif, MetaImitatorTool en MetaImitatorBridge te genereren voor de imitatortools, zodat deze met de ToolBus kunnen communiceren. Deze bestanden worden vervolgens gecombineerd met de standaard imitatortool bestanden. Deze worden gezamenlijk in een map opgeslagen, waarna een instellingenbestand wordt gegenereerd voor elke imitatortool. Hierin staat de locatie van de communicatie en BAF bestanden en welk proces de imitatortool moet imiteren. Als laatste stap wordt er een opstartbestand gemaakt voor de imitatortool en voor de testomgeving. Bij het starten van de testomgeving wordt de PATH variabele zodanig ingesteld dat de imitatortools worden gebruikt in plaats van de originele tools.



Figuur 9: Genereren van een testomgeving

5.4 Het uitvoeren van een integratietest

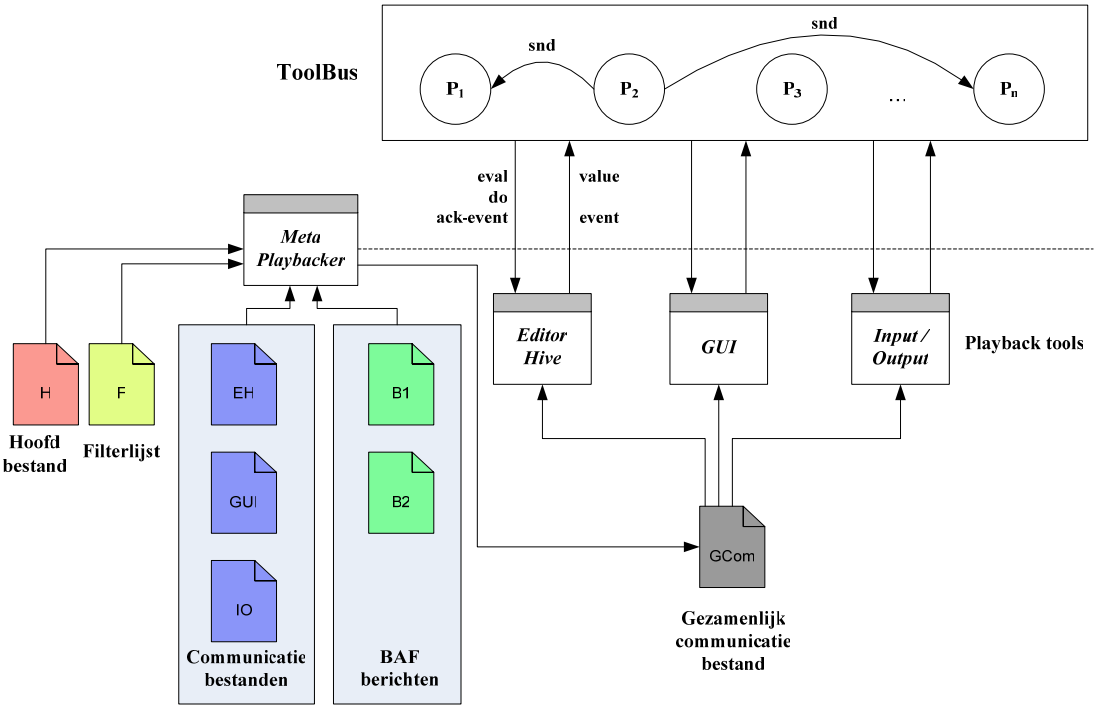
Nadat de testomgeving voor de integratietest gegenereerd is, kan de test uitgevoerd worden. De metaplaybacker (beschreven in hoofdstuk 4.4) opent het hoofdbestand, welke tijdens de opname is gecreëerd. Het hoofdbestand geeft aan welke bestanden er uitgelezen moeten worden. Dit zijn de lijst met filters (zodat de metaplaybacker weet dat de editor-hive, grafische user interface en de input / output gecontroleerd moeten worden), de verschillende communicatiebestanden en de BAF berichten. Dit proces is te zien in figuur 10.

Zodra alle informatie is ingelezen, kan de playbacksessie beginnen. De editor-hive en de grafische user interface worden vervangen door imitatortools. Op het moment dat de communicatie in de ToolBus stilvalt, controleert de metaplaybacker of een event door één van de imitatortools verstuurd moet worden. Als dit het geval is, wordt de naam van dat proces, met de boodschap dat een event gegenereerd moet worden en het event zelf in het gemeenschappelijke communicatiebestand opgeslagen. Vervolgens verstuurt de imitatortool, die dit event moet genereren, het event naar de ToolBus.

Er wordt bij het bepalen van het tijdstip waarop een event gegenereerd moet worden, expres niet de communicatie tussen componenten gebruikt. Dit om een abstractie te behouden, waarbij de integratietest geen gegevens heeft over de communicatie van tools, die buiten de integratietest vallen. Hierdoor kan de integratietest nog gebruikt worden nadat een component, die buiten de integratietest valt, wordt aangepast. Dit kan vooral handig zijn bij het refactoren van de ASF+SDF MetaEnvironment.

Als de metaplaybacker een evaluatieverzoek ontvangt voor één van de drie imitatortools, zal deze dit direct in het gemeenschappelijke communicatiebestand vermelden met de feedback, die terug naar de ToolBus gestuurd moet worden. De imitatortool zal deze feedback vervolgens naar de ToolBus sturen.

De metaplaybacker kijkt tijdens de communicatie of de berichten, die ontvangen worden, hetzelfde zijn als tijdens de opname, of dat deze voldoen aan het patroon van een veranderlijk bericht (zie 3.4). Mocht dit niet het geval zijn, dan wordt dit aan de tester gemeld. Er kunnen berichten gewijzigd zijn, maar het is ook mogelijk dat er berichten ontvangen worden, die tijdens de opnamefase niet verstuurd zijn.



Figuur 10: Uitvoeren van een integratietest

5.5 Een grotere test

Bij het maken van een grotere testomgeving blijkt dat er maar een minimum aantal imitatortools nodig zijn. Ook al worden er meerdere processen getest, de evens worden alleen door de grafische user interface en de editor-hive verstuurd. Hierdoor kunnen veel van de originele tools gebruikt worden in de test, waardoor de verwerking van deze tools in de test meegenomen wordt.

Nadat de test die hiervoor beschreven is, was uitgevoerd, is er een grotere test gemaakt. Hierbij werden alle processen bijgehouden. De acties die zijn uitgevoerd zijn:

- Syntax bestand geopend.
- Syntax definitie geparseerd.
- Syntax definitie gecontroleerd.
- Vergelijkingen bestand geopend.
- Vergelijkingen geparseerd.
- Vergelijkingen gecontroleerd.
- Term bestand geopend.
- Term geparseerd.
- Term weergegeven als boom.
- Term uitgevoerd.
- Uitgevoerde term weergegeven als boom.

De test is bijna helemaal goed te herhalen, op twee berichttypes na. Dit zijn:

- Het bericht sdf-definition van het proces SdfModules.
- Het bericht renamed-module van het proces SDFRenaming

Deze hebben tijdens de herhaling ander BAF berichten, doordat de informatie hierin in een andere volgorde kan binnenkomen. Hierdoor verandert het BAF bericht. Omdat de metaplaybacker geen BAF ondersteuning heeft, kan dit bericht niet verder gecontroleerd worden.

Op die twee berichten na, kan de test echter goed uitgevoerd worden. Om te voorkomen dat deze berichten foutmeldingen veroorzaken, kunnen de berichten als veranderlijk gekenmerkt worden. Deze berichttypes zullen in dat geval geen foutmelding meer veroorzaken.

6 Resultaten

Het onderzoek naar integratietesten voor de ASF+SDF MetaEnvironment heeft een aantal resultaten opgeleverd:

1. Inzicht in automatisch integratietesten met behulp van record en playback
2. Ontwerp en ontwikkeling van de metalogger
3. Ontwerp en ontwikkeling van de imitatortool
4. Ontwerp en ontwikkeling van de testgenerator
5. Ontwerp en ontwikkeling van de metaplaybacker

Als eerste geeft het onderzoek een goed inzicht in het automatiseren van de integratietest voor de ASF+SDF MetaEnvironment. De ToolBus architectuur biedt de mogelijkheid om de communicatie tussen componenten te gebruiken voor record en playback testen. Daarbij kan de tester zelf kiezen welke componenten in de test meegenomen moeten worden.

Daarnaast is er ook onderzocht wat er nodig is voor een testomgeving voor integratietesten. Naast een record- en een playbackapplicatie, zijn er ook tools nodig, die componenten kunnen vervangen. Deze moeten worden gebruikt, als bij de originele tools gebruikersinteractie nodig is. Als de test automatisch herhaald wordt, moeten die acties geautomatiseerd worden. Om ervoor te zorgen dat er bij het aanpassen van een tool, die gebruikersinteractie nodig heeft, geen nieuwe imitatortool ontwikkeld hoeft te worden, is een generator nodig die dit proces automatiseert.

Het tweede resultaat van dit onderzoek is een applicatie om een opname te maken van de communicatie tussen de verschillende componenten. Deze applicatie slaat alle informatie op, die nodig is om een testomgeving te genereren en voor het uitvoeren van de daadwerkelijke test.

Een gebruiker kan met deze applicatie eenvoudig opgeven welke processen tijdens de test gecontroleerd moeten worden. Vervolgens kunnen in de ASF+SDF MetaEnvironment alle acties worden uitgevoerd, die moeten worden getest. Hierdoor kan op een eenvoudige manier testgevallen opgenomen worden, die op een later tijdstip gebruikt kunnen worden bij de integratietest.

Het derde resultaat is de imitatortool. Deze tool is een generieke tool die elke tool van de ASF+SDF MetaEnvironment kan vervangen. Dit maakt het mogelijk om de test volledig geautomatiseerd uit te voeren. Anders moeten alle gebruikersinteractie tijdens de test ook plaats vinden, waarbij de gebruiker elke keer precies dezelfde acties moet uitvoeren.

De testgenerator is het vierde resultaat en genereert specifieke imitatortools aan de hand van het T-script, dat bij de te imiteren tool hoort. Hierdoor zijn de signatures precies hetzelfde als in de originele tool, waardoor de ToolBus geen verschil ziet.

Ten slotte is het vijfde resultaat de metaplaybacker. Deze applicatie zorgt ervoor dat de juiste imitatortools opgestart worden tijdens de integratietest. Daarnaast controleert het of de communicatie, op het moment dat de integratietest wordt uitgevoerd, nog hetzelfde is als tijdens de opname. Hiervoor wordt de ASF+SDF MetaEnvironment opgestart met de bijbehorende imitatortools en worden de acties herhaald, die tijdens de opname door de gebruiker zijn voorgedaan. Zodra de acties uitgevoerd zijn, wordt er onderzocht of de integratietest succesvol is verlopen, of dat er berichten veranderd zijn. Mochten er berichten anders zijn, dan kan de gebruiker aan de hand van deze informatie bepalen of er fouten in de applicatie zit, of dat de test vernieuwd moet worden als de veranderingen ook daadwerkelijk bedoeld zijn.

7 Evaluatie

In dit hoofdstuk staat de zelfevaluatie na het voltooien van het onderzoek.

7.1 Positief

Na het verrichten van het onderzoek zijn er een aantal punten welke ik zelf erg positief bevonden heb:

- Zelf bezig geweest met integratietesten.
- Zelf bezig geweest met component gebaseerde architectuur.
- Functioneel werkt de integratietest

Tijdens de opleiding lees je in de literatuur dat testen een zeer belangrijk onderdeel is van het ontwikkelproces. Het is cruciaal om de kwaliteit van software te waarborgen. Waar veel literatuur, die ik gelezen heb, over gaat en het vak Software Testing vooral op gericht was, was het bepalen en uitvoeren van unittests.

Ikzelf vond het heel interessant om een keer naar testen op een ander niveau te kijken. Integratietesten is een belangrijk onderdeel van het testproces en ik wilde hier dan ook graag een keer kennis mee maken. Zowel om te kijken welke mogelijkheden er zijn, als het zelf uitvoeren / ontwikkelen van een integratietest. Tijdens het onderzoek heb ik dit beide kunnen doen.

Een ander punt waar veel in de literatuur over te lezen is, is het gebruik van componenten in grote software systemen. Hier wilde ik ook graag meer ervaring mee opdoen, om te kijken hoe je hier nu het best mee kan omgaan. Tijdens het vak Software Construction hebben we wel in het klein met componenten gewerkt, maar tijdens het onderzoek ging het om veel meer werk met componenten en de samenwerking hiertussen.

Een ander positief punt dat is dat de integratietest daadwerkelijk werkt. Ondanks dat het meer werk was om de testomgeving te ontwikkelen, werkt de integratietest toch binnen de periode die voor het afstudeeronderzoek stond.

7.2 Negatief

Naast de positieve ervaringen, die ik heb opgedaan tijdens het onderzoek, waren er ook een aantal negatieve ervaringen:

- Probleem met de JAVA ATerm.
- Niet gedocumenteerde ToolBus logger.
- Programmeren was veel meer als verwacht.
- Imitatortool en testgenerator niet verwacht.

Het werd bij het begin van het onderzoek al redelijk snel duidelijk, dat er meer met de C implementatie, als met de JAVA implementatie van de ATerm wordt gewerkt. Ik ben veel tijd kwijtgeraakt om erachter te komen waarom sommige berichten niet geaccepteerd werden binnen de metalogger.

Het eerste probleem dat ik hierbij had, was dat de JAVA versie geen ATerm constructies ondersteunt waarin het $ teken gebruikt wordt. Dit teken wordt echter door het logpoint bericht veelvuldig gebruikt.

Het tweede probleem was dat de JAVA versie van de ATerm geen BAF ondersteuning biedt. Hierdoor moest ik een constructie verzinnen, die ervoor zorgt dat alle BAF berichten toch

verwerkt worden. Dit heeft me veel tijd gekost, terwijl het naar mijn mening weinig toegevoegde waarde heeft voor het onderzoek zelf.

Daarnaast was het lastig, dat de loggerfunctie van de ToolBus ongedocumenteerd is. Hierdoor moest ik zelf diep de code van de ToolBus induiken om de precieze werking te achterhalen. Op die manier heb ik ook moeten achterhalen waarom de ToolBus steeds vastliep. Hierbij ging het om een fout in het genereren van de bridge, tool en tif bestanden voor de logger. Volgens de gegenereerde bestanden hoeft er na het ontvangen van een logpoint door een loggerapplicatie, geen bericht teruggestuurd te worden. Er moet echter wel een snd-continue bericht verstuurd worden. Ook de berichten, die een logger ontvangt, heb ik zelf moeten analyseren om erachter te komen welk gedeeltes ik nodig had om tijdens de integratietest te gebruiken.

Het programmeergehalte van de opdracht heb ik in het begin onderschat. Ik had verwacht dat ik veel sneller de applicaties voor de integratietest had kunnen maken. Grotendeels kwam dit door het feit dat ik er van uit ging dat ik met een playback applicatie eenvoudig met één tool alle events kon genereren. Hiervoor had ik echter de imitatortools nodig en een testomgeving generator.

7.3 Tevredenheid opdrachtgever

Aan het einde van de afstudeerperiode heb ik de integratietest aan vier mensen van het Centrum van Wiskunde en Informatica gepresenteerd. Deze waren allemaal erg positief over het resultaat. De test wordt gezien als belangrijk hulpmiddel bij het refactoren van de broncode.

Daarnaast werd er positief gereageerd op het feit dat alleen de communicatie van de opgegeven componenten wordt opgeslagen. Hierdoor wordt er bij het uitvoeren van de test niet gekeken naar de communicatie tussen andere componenten bij het genereren van events. Mochten er aanpassingen gedaan worden in de T-scripts van componenten, die niet aan de test meedoen, dan kan dezelfde test nog steeds gebruikt worden.

Naast het feit dat de applicatie goed te gebruiken is tijdens het refactoren, werd er ook gesteld dat deze integratietest een goede aanvulling is voor de DailyBuild. De regressietest kan hier goed mee worden uitgebreid.

7.4 Reflectie op onderzoeksaanpak

Terugkijkend op het onderzoek zijn er een aantal punten welke ik in een volgend project anders zou doen:

- Te snel in het diepe, daardoor verwacht dat loggers ook events konden imiteren
- Te laat de precieze opbouw van de logpoint onderzocht
- Eerder overschakelen van metalogger naar metaplaybacker en imitatortool

Bij de eerste twee punten was ik te snel begonnen met werken. Ik wilde te snel beginnen met het ontwikkelen van de benodigde applicaties, om er zeker van te zijn dat ik binnen de tijdslimiet alles af zou krijgen. Hierdoor miste ik echter belangrijke kennis, wat er voor zorgde dat ik werk opnieuw moest doen, omdat mijn aannames niet klopte. Dit heeft onnodig veel tijd gekost

Daarnaast ben ik te lang op de eerste applicatie doorgegaan. De metalogger voor de opname van testgevallen wilde ik helemaal goed afhebben voordat ik verder ging. Toen ik met de metaplaybacker en de imitatortool begon, kwam ik er echter achter dat ik verkeerde aannames had gemaakt tijdens de ontwikkeling van de metalogger. Hierdoor moest ik de metalogger aanpassen. Ook dit heeft mij tijd gekost wat niet nodig was.

Daarnaast waren ook punten in mijn aanpak die wel goed gingen:

- Veel geleerd door zelfdoen en onderzoeken hoe dingen werken
- Snel een goede oplossing gevonden om het probleem van het gebrek aan imitators te verhelpen doormiddel van imitatortools.

Doordat ik redelijk zelfstandig kon werken, heb ik alle onderdelen zelf uit kunnen pluizen. Hierdoor heb ik een goed beeld in mijn hoofd gekregen van de manier om record en playback toe te passen voor de integratietest. Daarnaast heb ik hierdoor een goed beeld gekregen hoe de ToolBus met de verschillende tools omgaat. Hierdoor had ik snel een oplossing gevonden voor het probleem dat een logger applicatie geen events kan sturen naar de ToolBus.

Bijlage A: Literatuurlijst

Testen

[1] M. J. Harrold, Testing: A Roadmap, Proceedings of the Conference on The Future of Software Engineering, pp. 61 - 72, 2000, http://portal.acm.org/citation.cfm?id=336532&coll=Portal&dl=ACM&CFID=45866231&CFTOKEN=36494076

[2] E. J. Weyuker, Testing Component based Software: A Cautionary Tale, AT&T Labs, 1998, http://csdl.computer.org/dl/mags/so/1998/05/s5054.pdf

[3] W. T. Tsai, X Bai, R. Paul, W.Shao, V. Agarwal, End-To-End Integration Testing Design, Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development, pp. 166 – 171, 2001, http://csdl.computer.org/dl/proceedings/compsac/2001/1372/00/13720166.pdf

[4] S. Kirani, W. T. Tsai, Specification and verification of object-oriented programs, dec. 1994, http://asusrl.eas.asu.edu/cse494/content/oo/svooprg.pdf

[5] P. C. Jorgensen, C. Erickson, Object-Oriented Integration Testing, Communications of the ACM Volume 37, Issue 9, pp. 30 – 38, 1994, http://portal.acm.org/citation.cfm?id=182989&coll=portal&dl=ACM&CFID=47204886&CFTOKEN=92038754

[6] Y. Le Traon, T. Jérom, J. M. Jézéquel, P. Morel, Efficient Object-Oriented Integration and Regression Testing, IEEE Transactions on Reliability, 2000, http://www.xpeqt.com/support/pdf/test_papers/00855533.PDF

[7] J. Hartmann, C. Imobordorf, M. Meisinger, UML-Based Integration Testing, Siemens, Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis, pp. 60 – 70, 2000, http://portal.acm.org/citation.cfm?id=348872&coll=Portal&dl=ACM&CFID=45866231&CFTOKEN=36494076

[8] A. Bertolino, P. Inverardi, H. Muccini, A. Rosetti, An Approach to Integration Testing Based on Architectural Descriptions, Proceedings of the Third IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '97), pp. 77, 1997, http://csdl.computer.org/dl/proceedings/iceccs/1997/8126/00/81260077.pdf

[9] N. S. Eickelmann, D. J. Richardson, An Evaluation of Software Test Environment Architectures, Proceedings of the 18th international conference on Software engineering, pp. 353 – 364, 1996, http://portal.acm.org/citation.cfm?id=227798&coll=GUIDE&dl=ACM&CFID=45869909&CFTOKEN=89336710

[10] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, C. J. Colbourn, Constructing Test Suites for Interaction Testing, Proceedings of the 25th International Conference on Software Engineering, pp. 38 – 48, 2003, http://portal.acm.org/citation.cfm?id=776822&coll=GUIDE&dl=ACM&CFID=45870550&CFTOKEN=21993510

[11] D. J. Schultz, A Case Study in System Integration Using The Build Approach, Proceedings of the 1979 annual conference, pp. 153 – 151, 1979, http://portal.acm.org/citation.cfm?id=810050&dl=ACM&coll=portal

[12] Y. Kim, C. R. Carlson, Scenario Based Integration Testing for Object-Oriented Software Development, Proceedings of the 8th Asian Test Symposium, pp. 283, 1999, http://csdl.computer.org/dl/proceedings/ats/1999/0315/00/03150283.pdf

[13] G. Meszaros, Agile Regression Testing Using Record & Playback, Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pp. 353 – 360, 2003, http://portal.acm.org/citation.cfm?id=949344.949442

ASF+SDF MetaEnvironment

[14] J.A. Bergstra, P. Klint, “The Discrete Time ToolBus – a software coordination architecture”, Science of Computer Programming, pp. 205-229,, 1998, http://homepages.cwi.nl/~paulk/publications/CC01.pdf

[15] J.A. Bergstra, P. Klint, “The ToolBus – a component interconnection architecture”, 1994, http://homepages.cwi.nl/~paulk/publications/P9408.pdf

[16] P. Klint, “A Guide to ToolBus Programming”, 1996, http://www.cwi.nl/themes/sen1/twiki/pub/SEN1/ToolBus/toolbus-guide.ps

[17] H. de Jong, P. Olivier, “ATerm Library User Manual”, 1999, http://www.cwi.nl/projects/MetaEnv/aterm/doc/manual.ps.gz

[18] M. G. J. van den Brand, A. van Deursen, J. Heering, H. A. de Jong, M. de Jonge, T. Kuipers, P. Klint, L. Moonen, P. A. Olivier, Scheerder, J.J. Vinju, E. Visser, J. Visser, “The ASF+SDF Meta-Environment: a Component Based Language Development Environment”, Proceedings of the 10th International Conference on Compiler Construction, pp. 365 – 370, 2001, http://homepages.cwi.nl/~paulk/publications/CC01.pdf

Bijlage B: CD met bronbestanden

Op de cd staan de bronbestanden, die nodig zijn voor de integratietest voor de ASF+SDF MetaEnvironment. Tevens staat er een PDF document op de cd, waarin uitgelegd wordt hoe de bestanden geïnstalleerd dienen te worden.