# Afstudeerverslag

## Afstudeerverslag Dennis van Leeuwen

**12 augustus 2011**

Dennis van Leeuwen 07004222

# I. Voorwoord

Dit stageverslag is geschreven naar aanleiding van de afstudeerstage voor de opleiding Technische Informatica aan de HHS te Delft / Academie voor ICT & Media. Begin februari is er begonnen met een afstudeerstage. Gedurende deze afstudeeropdracht werd duidelijk dat het geen geschikte HBO afstudeeropdracht bleek, waardoor er begin april is besloten opnieuw te beginnen met een andere afstudeeropdracht. De nieuwe opdracht is om onderzoek te doen naar de visualisatie van een SystemC model en onderzoek te doen naar de mogelijkheid van het extraheren van TLM onderdelen uit een SystemC model.
Deze opdracht is een vervolg op de afstudeeropdracht van Harry Broeders. Hij heeft tijdens zijn afstudeeropdracht onderzoek gedaan naar de extractie van SystemC onderdelen uit een SystemC model. Zijn verslag is te vinden op http://shabe.sourceforge.net/.
Bij deze wil ik dan ook graag mijn opdrachtgever Harry Broeders bedanken voor het aanbieden van de opdracht en voor de begeleiding gedurende de afstudeerstage. Omdat de eerste afstudeerstage geen geschikte opdracht bleek te zijn, moest er op korte termijn een nieuwe stageopdracht worden verkregen. Door de snelle communicatie met Harry Broeders, kon er op korte termijn een nieuwe opdracht worden vastgesteld, waardoor er opnieuw kon worden begonnen met afstuderen.

Delft, 12 augustus 2011
Dennis van Leeuwen

# II. Samenvatting

Dit afstudeerverslag is het eindverslag van de afstudeerstage van Dennis van Leeuwen. De duur van de afstudeerstage is 17 weken. In dit document wordt beschreven hoe het proces van de afstudeerstage is verlopen en welke keuzes er tijdens de verschillende fases zijn gemaakt. De afstudeerstage is heeft plaatsgevonden  aan de HHS te Delft.
De afstudeerstage is in opdracht van Harry Broeders, docent Elektrotechniek, uitgevoerd. De opdracht bestaat uit twee delen:

- Onderzoek doen naar de mogelijkheid om de hiërarchie van een SystemC model te visualiseren.
- Onderzoek doen naar de mogelijkheid om TLM onderdelen uit een SystemC model te extraheren.

Het eerste deel van de opdracht heeft de eerste 5 weken plaatsgevonden. Tijdens dit onderzoek zijn verschillende visualisatie methodes met elkaar vergeleken om de hiërarchie van een SystemC model te visualiseren. Uit eindelijk is er gekozen om gebruik te maken van een bestaand tekenprogramma genaamd yEd. Om de hiërarchie van een SystemC model te kunnen visualiseren moest het formaat waarin de geextraheerde gegevens uit een SystemC model worden opgeslagen eerst worden omgezet naar een ander formaat. Om de conversie te kunnen verrichten is er een conversietool gemaakt waarmee het formaat waarin een SystemC model wordt opgeslagen om te zetten naar het formaat dat gebruikt kan worden door yEd. Uit het onderzoek kan worden geconcludeerd dat een SystemC model (nog) niet volledig kan worden gevisualiseerd. Dit komt door de beperkingen van de onderzochte programma's.

Tijdens het tweede deel van de afstudeeropdracht is er onderzoek gedaan naar de extractie van TLM onderdelen uit een SystemC model. Voordat er is begonnen met een proof of concept te maken is er onderzoek gedaan naar TLM, hoe TLM verschilt van SystemC en uit welke onderdelen TLM bestaat. Nadat dit is gedaan, kon er een tussentijdse conclusie worden getrokken ofdat het mogelijk is om TLM onderdelen te extraheren. De conclusie was dat de extractie mogelijk lijkt.
Door het maken van een proof of concept kan er een bewijs worden geleverd ofdat de extractie van TLM onderdelen ook praktisch mogelijk is. De ontwikkeling van het proof of concept genaamd SaTHE is opgedeeld in vijf incrementen. In ieder increment is SaTHE uitgebreid door de implementatie van extra TLM onderdelen toe te voegen.
Uit het onderzoek blijkt dat TLM onderdelen te extraheren zijn uit een SystemC model. Door beperkingen van GDB zijn er echter wel beperkingen aan het extraheren van een SystemC/TLM model.

# Inhoudsopgave

# 1 Inleiding

Iedere Technische Informatica student dient in het laatste gedeelte van zijn opleiding gedurende 17 weken een afstudeeropdracht uit te voeren. De opdracht die uitgevoerd wordt moet aansluiten bij kennis die is opgedaan tijdens de opleiding. Door het uitvoeren van deze opdracht kan de student bewijzen dat hij of zij genoeg kennis heeft opgedaan op het gebied Technische Informatica en de juiste keuzes kan maken en kan verantwoorden. Aan het eind van de afstudeerperiode wordt er een verslag opgeleverd waarin het proces van de afstudeerperiode is beschreven en waarin wordt uitgelegd welke keuzes er tijdens het uitvoeren van de afstudeeropdracht zijn gemaakt.

In dit verslag wordt de afstudeerperiode van Dennis van Leeuwen beschreven. De opdracht die wordt uitgevoerd tijdens de afstudeerperiode bestaat uit 2 onderdelen.
Tijdens het eerste gedeelte van de opdracht wordt er onderzoek gedaan naar de mogelijkheid om een SystemC model geëxtraheerd door SHaBE te visualiseren in een tekenprogramma.
Het tweede gedeelte en ook het grootste gedeelte van de opdracht is om te onderzoeken of TLM onderdelen uit een SystemC model kunnen worden geëxtraheerd door gebruik te maken van SHaBE.

Het doel van het uitvoeren van deze opdracht is om duidelijkheid te krijgen over de mogelijkheid om SystemC modellen te visualiseren en TLM onderdelen uit een SystemC model te extraheren. Het uitvoeren van deze opdracht is een vervolg op het onderzoek naar de extractie van SystemC uit een SystemC model, dat is uitgevoerd door Harry Broeders, zie http://shabe.sourceforge.net/
Tijdens zijn afstudeeropdracht heeft Harry Broeders zich verdiept in het dynamisch extraheren van een dynamisch opgebouwd SystemC programma.

# 2 Inleiding SystemC/Extractie SystemC

In de volgende hoofdstukken wordt er gesproken over verschillende begrippen. Omdat deze begrippen mogelijk onbekend of deels onbekend zijn wordt er in dit hoofdstuk aandacht aan deze begrippen besteed, zodat vast staat wat deze begrippen betekenen en er helder is wat er met de begrippen bedoeld wordt.

## 2.1 SystemC

SystemC is een C++ library waarmee een systeem beschreven kan worden. Zowel de software als de hardware kan worden beschreven in SystemC. Dit wordt gedaan door componenten te modelleren met bepaald gedrag. Er zijn verschillende componenten te beschrijven, zoals modules, ports en channels.

## 2.2 Modules(sc_module)

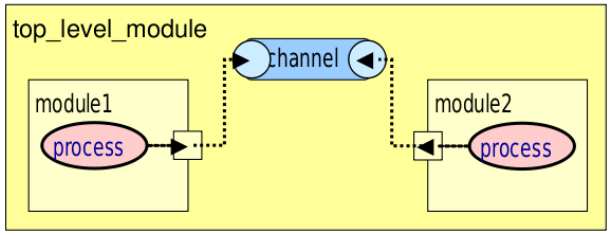
Een module kan worden gezien als een component. Een module kan berekeningen uitvoeren, dit wordt gedaan in processen. Een module kan zelf meerdere modules bevatten. Hierdoor ontstaat er in SystemC een hiërarchische structuur van modules.

## 2.3 Ports(sc_port & sc_export)

Om modules met elkaar te verbinden bevatten de modules ports. Deze ports kunnen aan elkaar worden gekoppeld. Tijdens het afstudeerproject zijn er 2 type SystemC ports van belang, sc_port en sc_export.

## 2.4 Channels

De ports van verschillende modules kunnen hierarchisch direct aan elkaar worden gekoppeld of kunnen via een channel aan elkaar worden gekoppeld. Channels kunnen gebruikt worden om bijvoorbeeld data te bufferen. Zie afbeelding 1.




*Afbeelding 1: 2 modules zijn via een channel gekoppeld*

## 2.5 ScObject

De klasse sc_object is de klasse waarvan de objecten, die er in de hiërarchie van een SystemC model gebruikt worden, zijn afgeleid. Wanneer er wordt gesproken over een ScObject betekent dit dat er wordt gesproken over een object dat wordt gebruikt in een SystemC model, een SystemC Object. De term ScObject wordt gebruikt om hiermee een object uit een SystemC model dat afgeleid is van sc_object aan te duiden.

## 2.6 TLM

TLM staat voor Transaction Level Modelling. TLM is een uitbreiding van de SystemC library. Met behulp van TLM kan een systeem op een standaard manier op een abstracter niveau dan RTL(Register Transfer Level) worden beschreven. SystemC biedt de mogelijkheid om een systeem op een abstracter niveau dan RTL niveau te beschrijven. Echter is het nadeel hiervan dat dit niet volgens een standaard manier kan worden gedaan. Hierdoor kan iedere SystemC gebruiker een systeem op een hogere abstractie niveau beschrijven, maar wel op zijn eigen manier. Dit zorgt voor een slechte interoperability wanneer de systeembeschrijvingen worden uitgewisseld tussen de SystemC ontwikkelaars.

Een van de belangrijkste aspecten van TLM ten opzichte van SystemC is dat TLM zorgt voor een hogere uitwisselbaarheid(interoperability) van de beschreven onderdelen. Dit komt omdat in de TLM standaard gestandariseerd is hoe een model op TLM niveau beschreven kan worden. Door volgens deze standaard te werken, wordt een model beter uitwisselbaar.

### 2.7 SCMDL

SCMDL is de afkorting voor SystemC Model Definition Language. SCMDL is het formaat waar de data in wordt opgeslagen welk uit een SystemC model door SHaBE wordt geëxtraheerd.

### 2.8 Elaboration fase

De elaboration fase is een fase die plaatsvindt wanneer het SystemC model wordt uitgevoerd. Tijdens deze fase wordt het te simuleren systeem opgebouwd. Dit betekent dat de verschillende SystemC onderdelen die gesimuleerd moeten worden, worden aangemaakt en worden geregistreerd in de SystemC simulatie kernel. In deze fase worden de gebruikte sc_object's met elkaar verbonden, zodat het te simuleren systeem opgebouwd wordt volgens de gedefinieerde beschrijving.

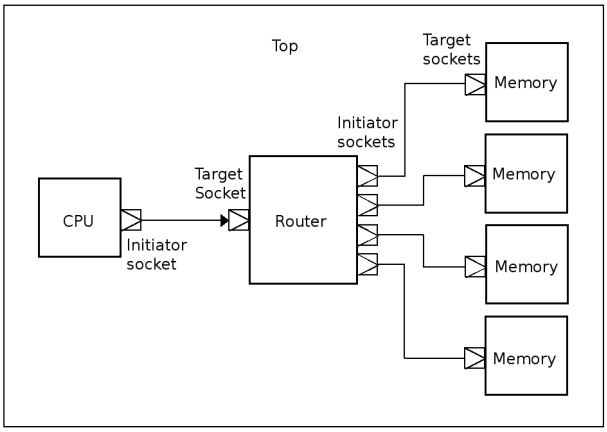
### 2.9 Dynamische extractie van een SystemC model

Tijdens de afstudeeropdracht van Harry Broeders heeft hij zich gericht op het dynamisch extraheren van een SystemC model en daarom wordt er ook gebruik gemaakt van dynamische extractie tijdens het uitvoeren van deze afstudeeropdracht. Er zijn drie verschillende aanpakken bekend om een SystemC model te extraheren.

1. Statische aanpak. Bij het kiezen van deze aanpak wordt het SystemC model al geëxtraheerd voordat deze uitgevoerd wordt. Een voorbeeld van het gebruik van deze aanpak is door een programma te ontwikkelen die de source code doorzoekt op zoek naar SystemC onderdelen. Deze onderdelen worden vervolgens geëxtraheerd.
2. Dynamische aanpak. Bij het gebruik van deze aanpak moet het SystemC model worden uitgevoerd. Dit betekent dat er een executable van het SystemC model nodig is, deze kan worden gemaakt door de source code van het SystemC model te compileren en te linken aan de SystemC library. Tijdens het uitvoeren van het SystemC model wordt onderzocht welke SystemC onderdelen er worden gebruikt in de simulatie.
3. Hybride aanpak. De hybride aanpak maakt gebruik van de dynamisch en de statische aanpak. Bij deze aanpak wordt de source code doorzocht en wordt het SystemC model uitgevoerd.

## 2.10 Voorbeeld SystemC/TLM model

Om het gebruik van een SystemC model en de hiërarchie die een SystemC model bevat verder toe te lichten, wordt er een schematische weergave van een TLM systeembeschrijving van Doulos gebruikt.(Zie http://www.doulos.com/knowhow/systemc/tlm2/tutorial__3/)
In SystemC/TLM worden er diverse modules gedefinieerd met in- en uitgangen. Deze in- en uitgangen worden aangemaakt door het gebruik van sockets. De schematische weergave van het voorbeeld is in afbeelding 2 te zien.



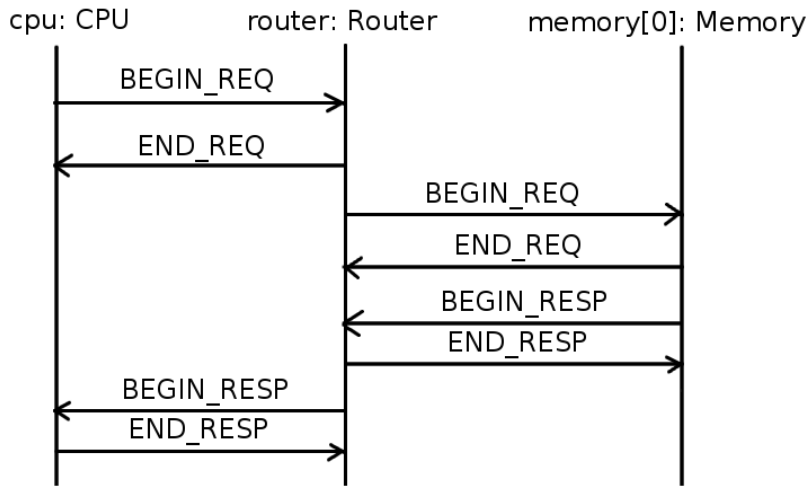

*Afbeelding 2: Schematische tekening van het SystemC/TLM model*

Omdat beide onderdelen van de opdracht zich richten op de hiërarchie van een SystemC model wordt er zo min mogelijk aandacht besteed aan het gedrag van het SystemC model. Omdat het echter noodzakelijk is om een simpel idee te geven van het mogelijke gedrag van dit model, wordt in dit voorbeeld ook aandacht besteed aan het gedrag. Hieronder wordt echter alleen de beschrijving van de hiërarchie van dit model in SystemC/TLM weergeven.

```
SC_MODULE(CPU){
public:
	tlm_utils::simple_initiator_socket<Initiator> socket;
	SC_CTOR(Initiator): socket("initiatorSocket"){}
};

SC_MODULE(Memory){
public:
	tlm_utils::simple_target_socket<Memory> socket;
	SC_CTOR(Memory): socket("memorySocket"){}
};

template<unsigned int TARGETS>
SC_MODULE(Router){
public:
	tlm_utils::simple_target_socket<Router> target_socket;
	tlm_utils::simple_initiator_socket_tagged<Router>* initiator_socket[TARGETS];

	SC_CTOR(Router): target_socket("target_socket"){
		for (unsigned int i = 0; i < TARGETS; i++){
			initiator_socket[i] = new
tlm_utils::simple_initiator_socket_tagged<Router>(("initiator" + int2string(i)).c_str());
		}
```

```
        }
        ~Router(){
                for (unsigned int i = 0; i < TARGETS; i++){
                        delete initiator_socket[i];
                }
        }
};

template <unsigned int N>
SC_MODULE(Top){
public:
        CPU cpu;
        Router<N> router;
        Memory*    memory[N];

        SC_CTOR(Top): initiator("initiator"), router("router"){
                for (unsigned int i = 0; i < N; i++){
                        memory[i] = new Memory(("memory_" + int2string(i)).c_str());
                        router.initiator_socket[i]->bind(memory[i]->socket);
                }
                cpu.socket.bind(router.target_socket);
        }
        ~Top(){
                for (unsigned int i = 0; i < N; i++)
                        delete memory[i];
        }
};

int sc_main( int argc, char **argv){
        Top<4> top("top");
        return 0;
}
```

Mogelijke gedrag dat aan dit model in de beschrijving kan worden toegevoegd is te zien in het sequence diagram in afbeelding 3. Hierin is een voorbeeld van het mogelijke gedrag te zien wanneer er een bericht vanaf de CPU naar een Memory moet worden gestuurd. De Router zorgt er in dit geval voor dat het bericht naar de juiste Memory module wordt gestuurd.




*Afbeelding 3: Versturen van berichten*

# 3 Opdracht

## 3.1 Probleemstelling

Harry Broeders heeft zich tijdens zijn afstudeeropdracht aan de TU Delft bezig gehouden met een onderzoek naar de extractie van SystemC onderdelen uit een dynamisch opgebouwd SystemC model. Een dynamisch opgebouwd SystemC model bestaat uit o.a. SystemC objecten die met de (C++) new operator zijn aangemaakt. Tijdens zijn onderzoek heeft hij het programma SHaBE ontwikkeld.

SHaBE extraheert SystemC onderdelen uit een SystemC model en slaat deze geëxtraheerde gegevens op in het SCMDL formaat. Het SCMDL formaat is gebaseerd op het XML formaat. Wanneer dit formaat wordt gebruikt om de geëxtraheerde gegevens door te geven aan een ander programma, is het SCMDL erg handig, omdat er op deze manier data volgens een gedefinieerd formaat kan worden uitgewisseld. Het probleem is echter dat wanneer een gebruiker het SCMDL bestand wil lezen en de gehele hiërarchie van het geëxtraheerde model wil achterhalen dit erg veel moeite kost, omdat er veel data in het formaat wordt opgeslagen en er veel tags aan elkaar gekoppeld kunnen worden.

Een ander probleem bij het extraheren van SystemC onderdelen uit een SystemC model is dat SHaBE geen ondersteuning geeft voor het extraheren van TLM onderdelen uit SystemC modellen, terwijl TLM wel kan worden gebruikt in SystemC modellen. Door het gebruik van TLM in SystemC modellen kan het systeem op een abtracter niveau worden beschreven. Omdat systemen die ontwikkeld worden in de loop der jaren steeds complexer worden is het nodig om het systeem op een hoger niveau te kunnen beschrijven en te testen. Door het beschrijven van het systeem op een abstracter niveau kan de werking van het systeem op een abstracter niveau worden geverifieerd, waardoor ontwerpfouten eerder in de ontwikkelfase ontdekt kunnen worden. Bij het gebruik van TLM kan er bijvoorbeeld puur worden getest of de communicatie tussen 2 modules wel correct is i.p.v. de implementatie van het ontvangen en het zenden van de berichten. SHaBE kan geen TLM onderdelen uit een SystemC model extraheren, waardoor er waardevolle informatie over een SystemC model verloren kan gaan.

## 3.2 Doelstelling

De afstudeeropdracht bestaat uit 2 opdrachten. Er worden daarom 2 doelstellingen gedefinieerd.

1. Het is onbekend hoe een SystemC model gevisualiseerd kan worden, zodat een gebruiker snel een goed overzicht heeft over de onderdelen die uit een SystemC model zijn geëxtraheerd. De doelstelling is daarom om onderzoek te doen naar het visualiseren van de hiërarchie van een SystemC model.
2. SHaBE biedt niet de mogelijkheid om een SystemC model te gebruiken tijdens de extractie om er vervolgens TLM onderdelen uit te extraheren. Omdat onbekend is of TLM onderdelen uit een SystemC model geëxtraheerd kunnen worden, wordt er onderzoek gedaan naar het extraheren van TLM onderdelen uit een SystemC model.

### 3.3 Op te leveren producten

Gedurende de afstudeeropdracht worden er verschillende producten opgeleverd. Dit zijn:

1. Onderzoeksrapport van het onderzoek naar de visualisatie van de hiërarchie van een SystemC model
2. Proof of concept van de visualisatie van de hiërarchie van een SystemC model
3. Onderzoeksrapport van het onderzoek naar de extractie van TLM onderdelen uit een SystemC model.
4. Proof of concept van de extractie van TLM onderdelen uit de hiërarchie van een SystemC model.

### 3.4 Aanpak

Voordat een aanpak gekozen kan worden, moet er een ontwikkelstrategie gekozen worden. Een analyse die bij de afstudeerder bekend is voor het onderzoeken welke ontwikkelstrategie het meest geschikt is voor een project, is door het uitvoeren van een contingentie analyse. Met een contingentie analyse kan op basis van bepaalde factoren een keuze gemaakt worden uit verschillende ontwikkelstrategieën. Deze keuze wordt gemaakt door de totale onzekerheid van het project te bepalen. De verschillende ontwikkelstrategieën zijn:

1. Acceptatie strategie.
2. Lineaire strategie.
3. Iteratieve strategie.
4. Incrementele strategie.

Aan de hand van de factoren projectgrootte, mate van gestructureerdheid, taakinzicht gebruikers en deskundigheid ontwikkelaars kan een geschikte ontwikkelstrategie bepaald worden. Deze factoren behoren tot de lijst van factoren waarmee een contingentieanalyse kan worden opgesteld. Hieronder een overzicht van de waardes van deze factoren en wat hun aandeel is in de totale onzekerheid. De manier waarop een contigentieanalyse moet worden uitgevoerd is te vinden in het boek "Management informatiesystemen".

| *Factor* | *Waarde* | *Aandeel totale onzekerheid* |
|---|---|---|
| Projectgrootte | Klein | - |
| Mate van gestructureerdheid | Redelijk gestructureerd | +/- |
| Taakinzicht gebruikers | Onvoldoende | + |
| Deskundigheid ontwikkelaars | Laag | + |

De verkalring van de ingevulde waardes is als volgt:

***Projectgrootte – klein***

Het project is in opzet klein. De afstudeerder werkt alleen aan de afstudeeropdracht en wordt daarbij door de opdrachtgever ondersteund.

***Mate van gestructureerdheid – redelijk gestructureerd***
De opdracht is redelijk gestructureerd, want als er vastgesteld kan worden hoe het proof of concept moet gaan werken kan de opdracht in de vorm van een programma beschreven worden.

***Taakinzicht gebruikers – onvoldoende***
Het taakinzicht van de gebruikers is onvoldoende. Dit komt omdat het onzeker is welke eisen er gesteld kunnen worden. Er wordt een onderzoek uitgevoerd, waardoor duidelijk wordt tot in hoeverre bepaalde wensen van de opdrachtgever kunnen worden geïmplementeerd.

***Deskundigheid ontwikkelaars – laag***
De afstudeerder heeft de basiskennis voor het ontwikkelen in C++ en heeft een minor gevolgd waar gewerkt is met SystemC. Er is tijdens de minor weinig ervaring met het werken met TLM opgedaan. De opdrachtgever daarentegen heeft veel meer ervaring op het gebied van SystemC en TLM.

**Conclusie**
Het risico van het project is gemiddeld (taakinzicht en deskundigheid onvoldoende). Er wordt gekozen om gebruik te maken van een incrementele ontwikkelmethode, omdat het risico van het project gemiddeld is en er verschillende onderdelen zijn die gevisualiseerd en geëxtraheerd moeten worden. Door dit in incrementen te doen, kan er elk increment worden bepaald wat er gedaan moet worden, waardoor er beter kan worden vastgesteld wanneer welke onderdelen gevisualiseerd en geëxtraheerd kunnen worden. Het grote voordeel van incrementeel ontwikkelen t.o.v. iteratief ontwikkelen is dat er aan het eind van ieder increment kan worden vastgesteld welke onderdelen wel en welke onderdelen niet kunnen worden gevisualiseerd of geëxtraheerd. Wanneer er voor een iteratieve ontwikkelmethode wordt gekozen vindt de oplevering van het project pas aan het eind plaats, waardoor er dan pas een moment is waarop kan worden vastgesteld of alle onderdelen kunnen worden gevisualiseerd of geëxtraheerd, omdat dan de extractie van ieder onderdeel pas af hoeft te zijn.

Het is belangrijk om veel kennisoverdracht met de opdrachtgever plaats te laten vinden. Het is voor de opdrachtgever ook moeilijk om alle informatie waarover hij beschikt in een keer over te dragen. Dit is wel de bedoeling wanneer er voor een acceptatie ontwikkelstrategie wordt gekozen. Tevens is het noodzakelijk dat er tussentijds wordt overlegd om te bekijken tot in hoeverre bepaalde wensen en eisen van de opdrachtgever geïmplementeerd kunnen worden.

Voor een lineaire ontwikkelstrategie is niet gekozen, omdat dan in één keer het heel het project moet worden opgeleverd. Daarnaast moet vooraf het eisenpakket volledig worden gespecificeerd en kunnen er problemen optreden als eisen gedurende het project veranderen. Door het project in onderdelen te ontwikkelen, is het risico kleiner dat het project mislukt. Omdat het ook om een onderzoek gaat is het van belang om het project op te splitsen in verschillende onderdelen. Zo kan er per onderdeel worden vastgesteld of de hoofdvraag

hierdoor falsificeerbaar is en of er verder moet worden gegaan met het onderzoek of het ontwikkelen van het proof of concept.

Er zijn veel verschillende soorten ontwikkelmethodes. Op school is er weinig aandacht besteedt aan iteratieve/incrementele ontwikkelmethodes (alleen RUP). Door de beperkte kennis en ervaring met ontwikkelmethodes, zijn RUP, IAD en XP als oplossing uit de zoektocht naar geschikte ontwikkelmethodes gekomen.
Er wordt gekozen om een Agile methode gebaseerd op XP te gebruiken omdat er veel contact nodig is met de opdrachtgever en er een verschillende onzekerheden zijn. Uit de onderzoeken kunnen diverse resultaten komen waardoor er een snelle aanpassing qua gestelde eisen moet worden ingevoerd. XP lijkt erg geschikt voor een project waarbij er een grote kans is dat de eisen tussentijds veranderen.

Om een keuze te maken voor een geschikte ontwikkelmethode is er gekeken naar drie ontwikkelmethodes: IAD,XP en RUP.
Er wordt niet voor IAD of RUP gekozen, omdat:

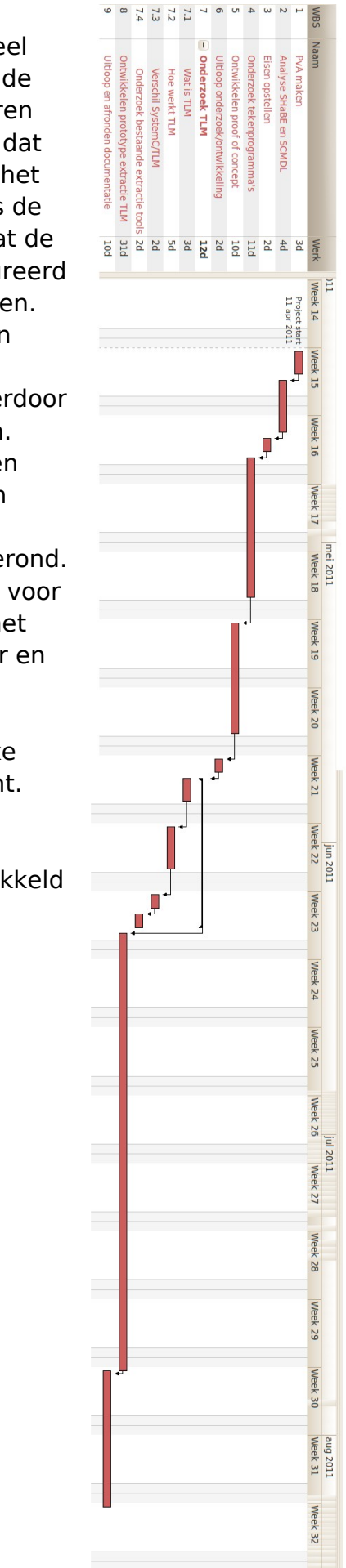
Een van de nadelen van de IAD methode is “Onervarenheid met de manier van werken kan leiden tot teleurstellingen en misverstanden.”. Omdat er nog nooit volgens de IAD methode is ontwikkeld en onervarenheid met deze methode kan leiden tot teleurstellingen en misverstanden wordt deze methode niet gekozen. Wanneer er wordt gekeken naar de fases die er doorlopen worden, dan is de ontwikkelmethode echter wel geschikt.

Nadeel van RUP kan zijn dat er teveel tijd aan de ontwerpfase wordt besteed. Tijdens het onderzoek is het juist van belang dat er een werkend product moet komen waarmee het antwoord op de hoofdvraag kan worden aangetoond. Het is natuurlijk van belang dat er volgens een goede aanpasbare structuur wordt gewerkt, het is echter van groter belang dat er een proof of concept komt. Hierdoor wordt de voorkeur als snel aan de Agile methode XP gegeven, omdat er bij deze methode meer gefixeerd wordt op het eindproduct en minder op het proces. Bij zowel IAD als XP is er een sterke betrokkenheid van de gebruiker, daarom wordt ervoor gekozen om de mening van de opdrachtgever mee te laten tellen. Wanneer er met de opdrachtgever wordt besproken volgens welke methode het project zal gaan verlopen geeft de opdrachtgever de voorkeur aan een Agile ontwikkelmethode gebaseerd op XP vanwege het gebruik van de Planning Game.

Een van de eigenschappen van XP kan niet worden gebruikt en dat is pair programming. Omdat er wel veel communicatie is met de opdrachtgever kan deze wel de genomen ontwerp/implementatie beslissingen valideren tijdens het overleg. Een andere eigenschap van XP is dat de code het belangrijkste product is. Er wordt tijdens het gebruik van XP niet vaak een model gemaakt. Tijdens de afstudeeropdracht wordt dit echter wel gedaan, omdat de afstudeerder weet dat hij software beter en gestructureerd kan ontwikkelen door voorafgaand een model te maken. Door dit model te maken, kunnen de afhankelijkheden beter in kaart worden gebracht en kan er op een abstracter niveau naar software gekeken worden. Hierdoor kunnen onnodige afhankelijkheden worden vermeden. Wanneer er nieuwe taken/te ontwikkelen eisen worden ontdekt, worden deze op een papiertje geschreven en wordt er verdergegaan met het werk waaraan werd gewerkt. Iedere 2 weken wordt er een increment afgerond. Bij het afronden van een increment, worden de taken voor het volgende increment vastgesteld. Dit proces van het samenstellen van taken samen met de opdrachtgever en wordt in Agile termen ook wel The Planning Game genoemd. Tijdens The Planning Game worden de papiertjes met taken gebruikt om vast te leggen welke taken er worden uitgevoerd in het volgende increment.

### 3.5 Planning

Nadat is vastgesteld volgens welke methode er ontwikkeld wordt kan er een planning worden opgesteld. Op de afbeelding aan de rechterkant van deze pagina is de planning te zien die aan het begin van de afstudeerperiode is gemaakt.

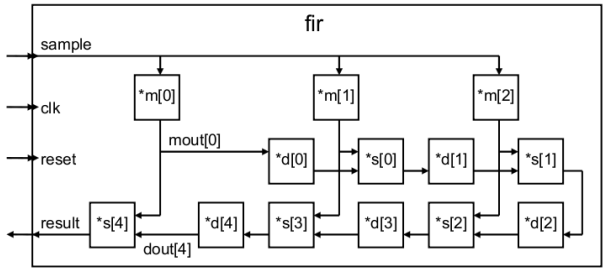


# 4 Onderzoek visualisatie SystemC model

Het eerste onderdeel van de afstudeeropdracht is om onderzoek te doen naar een manier om SystemC modellen te visualiseren. Dit onderzoek is te vinden in bijlage A. In de volgende hoofdstukken wordt beschreven hoe het onderzoek is verlopen en wordt er beschreven welke keuzes er zijn gemaakt en wat het uiteindelijke resultaat van het onderzoek is.

## 4.1 Eisen vaststellen

Om de eisen van de opdrachtgever in kaart te brengen is er gekozen om eenmalig een gesprek te houden waarbij de eisen, die er aan het begin van het onderzoek aan het visualiseren van een SystemC model worden gesteld, worden geformuleerd. Aan deze eisen zijn in eerste instantie geen prioriteiten toegewezen, omdat er tijdens het onderzoek alleen wordt vastgesteld met welke programma's en formaten er welke SystemC onderdelen kunnen worden gevisualiseerd. Alle eisen zijn indirect herleidt uit het SystemC model in afbeelding 4. Deze afbeelding dient als voorbeeld hoe de uiteindelijk gegenereerde afbeelding van de hiërarchie van het SystemC model eruit moet komen te zien. Uit deze afbeelding en het gesprek met de opdrachtgever zijn de volgende specificaties opgesteld.




*Afbeelding 4: SystemC model FIR filter*

Specificaties van het weergeven van een poort

- Tonen van de poort. (SPEC1)
- Tonen van het type van een poort. (SPEC2)
- Tonen van de naam van de poort. (SPEC3)

Specificaties van het weergeven van een signaal(channel)

- Tonen van het signaal (SPEC4)
- Tonen van de richting van een signaal. (SPEC5)
- Tonen van de naam van een signaal. (SPEC6)

Specificaties van het weergeven van een module

- Tonen van de module. (SPEC7)
- Tonen van de hiërarchie van een module. (SPEC8)
- Tonen van de naam van een module. (SPEC9)

Overige

- Bij voorkeur is het gebruikte programma open-source. (SPEC10)

Wanneer deze eisen zijn vastgesteld, kan er samen met de opdrachtgever worden bepaald welke manier van visualiseren het beste is en daarmee welke programma's en formaten nodig zijn om een SystemC model te visualiseren.

Tijdens het onderzoek zijn er diverse programma's gevonden waarmee SystemC modellen gevisualiseerd kunnen worden. Daarnaast gebruiken de meeste programma's een formaat waarin de visualisatie opgeslagen wordt. Omdat bijna

geen van de programma's specifiek bedoeld is voor het visualiseren van een SystemC model heeft ieder programma andere voor- en nadelen. Tijdens het onderzoek zijn deze voor- en nadelen met elkaar vergeleken, zodat er kan worden vastgesteld welk programma en eventueel bijbehorend formaat het meest geschikt is voor het visualiseren van een SystemC model.
In het onderzoek zijn niet alleen bestaande programma's maar ook andere visualisatie mogelijkheden onderzocht, zoals het ontwikkelen van een nieuw programma. Hieronder een overzicht van de belangrijkste en daarmee meest bruikbare oplossingen welke tijdens het onderzoek zijn onderzocht. De gehele lijst met onderzochte programma's en formaten is te vinden in het onderzoek in bijlage A.

### 4.2 Formaten
In dit hoofdstuk worden de formaten die zijn onderzocht besproken.

#### 4.2.1 GraphML
GraphML is een formaat gebaseerd op het XML en op het GML formaat. Net zoals de oprichters van het GML formaat vonden de oprichters van het GraphML formaat het nodig dat er een standaard formaat kwam waarmee grafen getoond kunnen worden. GraphML is een opvolger van het GML formaat. In GraphML is er rekening gehouden met eventuele uitbreidingen die nodig zijn om extra gegevens in het formaat op te slaan. Hiervoor kunnen extra tags kunnen worden gebruikt. Dit kan door zelf een uitbreiding van het formaat te definiëren in een DTD.

#### 4.2.2 IP-XACT
IP-XACT is een formaat waarmee hardware modules kunnen worden beschreven. IP-XACT is een open standaard formaat gebaseerd op XML(vastgesteld door de IEEE). In dit formaat kunnen de diverse componenten beschreven worden die er in een systeem worden gebruikt. Dit formaat is dus niet een formaat waarin een graaf wordt beschreven, maar een formaat waarin de eigenschappen van een hardware module wordt beschreven. Omdat deze eigenschappen allemaal in dit formaat worden beschreven is het mogelijk om dit formaat te gebruiken bij het genereren van modellen voor een simulatie van hardware.

### 4.3 Tekenprogramma's
In dit hoofdstuk worden de belangrijkste bestaande tekenpakketten besproken die onderzocht zijn.

#### 4.3.1 yEd
Met het programma yEd kan een GraphML bestand geopend worden. Het programma yEd is ontwikkeld in Java en zou dus op ieder platform moeten kunnen worden uitgevoerd. In yEd is een functie beschikbaar waarmee gebruikers zelf de graaf kunnen positioneren. yEd is gratis te gebruiken, maar het programma is niet open-source.

#### 4.3.2 SpiceVision
SpiceVision is een programma waarmee RTL blok diagrammen gemaakt kunnen worden. Ook is het mogelijk om op een hoger hiërarchisch niveau het blok

diagram te bekijken. Dit is een van de wensen van de opdrachtgever. Om SpiceVision te gebruiken is er wel een licentie nodig en het programma is niet open source. De kosten van deze licentie liggen rond de 3.200 euro per jaar. Voor academische doeleinden wordt er een korting gegeven van 50%.

### 4.4 Libraries

Naast het gebruiken van bestaande tekenpakketten is er ook gekeken of er libraries zijn waarmee de hiërarchie van een SystemC model kan worden gevisualiseerd. Hieronder wordt de belangrijkste library besproken.

#### 4.4.1 OGDF(Open Graph Drawing Framework)

Met gebruik van de OGDF library kan er een applicatie worden geschreven waarin alle wensen van de opdrachtgever vervuld kunnen worden. In het OGDF zijn diverse algoritmes aanwezig om de route van de pijlen en de plaats van de blokken te bepalen, zodat er geen overlappende pijlen en blokken in de tekening worden geplaatst. Door het gebruik van deze algoritmes blijft de tekening overzichtelijk.
In OGDF is ook een utility beschikbaar genaamd gml2pic. Door het gebruik van deze utility kan de graaf die met OGDF is gemaakt worden omgezet naar een afbeelding.

### 4.5 Bestaand SystemC extractie programma

Naast een onderzoek naar libraries die herbruikt kunnen worden is er ook gekozen om onderzoek te doen naar applicaties die al SystemC modellen kunnen visualiseren.

#### 4.5.1 gSysC

gSysC is een programma wat net als SHaBE gegevens uit een SystemC model kan extraheren. gSysC biedt echter niet de mogelijkheid om dynamisch de gegevens uit een SystemC model te extraheren.(Vandaar dat SHaBE is ontwikkeld.) gSysC kan de hiërarchie van het SystemC model tonen. Dit deel van gSysC zou kunnen worden hergebruikt om de gegevens die SHaBE uit het SystemC model haalt te tonen.
gSysC maakt gebruik van Qt3 om de hiërarchie te tekenen. In Qt3 zijn er algoritmes aanwezig om lijnen en blokken te tonen. In Qt3 is echter geen lay-out algoritme aanwezig, daarom heeft de auteur van het programma voor een alternatieve manier van het tonen van modules en signalen gekozen en dat is om de modules even groot weer te geven en in rijen evenwijdig aan elkaar. Vervolgens worden de signalen die tussen de modules lopen gebundeld. Doordat de signalen gebundeld zijn, wordt het gevisualiseerde model minder praktisch om te gebruiken, omdat het niet erg overzichtelijk is.

### 4.6 Conclusie

Het programma waarmee de hiërarchie van het SystemC model gevisualiseerd gaat worden is bij voorkeur open source, Magillem is dit echter niet. Omdat Magillem niet gratis te downloaden is, of proefversie te verkrijgen is, wordt ervoor gekozen om het IP-XACT formaat en daarmee het enige programma wat het IP-XACT formaat kan openen, Magillem, niet te gebruiken.

In GraphML kan alles worden gedefinieerd wat nodig is om de hiërarchie van een SystemC model te visualiseren. Het voordeel wat GraphML daarnaast biedt is dat het goed uitbreidbaar is, zodat extra eisen die in de toekomst aan het formaat worden gesteld aan het formaat toegevoegd kunnen worden. Dit kan door zelf een extensie van het GraphML formaat in het GraphML bestand toe te voegen.

Om GraphML weer te geven zijn er verschillende tools beschikbaar. Een van de tools die zowel de hiërarchie volgens in een in- en uitklap mechanisme kan weergeven en alle namen bij de juiste items kan zetten is yEd.
yEd is gratis te gebruiken en biedt het voordeel dat het te gebruiken is op alle platformen die JAVA ondersteunen.
yEd kan met verschillende formaten werken, waaronder GraphML. Omdat er tijdens het omzetten van de SCMDL naar GraphML geen coördinaten worden toegevoegd aan de GraphML is het van belang om een editor te gebruiken, die de lay-out van de graaf zelf kan opmaken. Dit algoritme voor het bepalen van de lay-out is in yEd aanwezig. Het enige nadeel van yEd is dat dit programma de poorten die in de GraphML gedefinieerd zijn niet toont. Het tonen van poorten wordt wel opgegeven als feature, er wordt daarom verwacht dat dit in de toekomst wel mogelijk wordt.

Tot slot is er nog de SystemC front-end genaamd gSysC die ook de geëxtraheerde data uit een SystemC beschrijving grafisch kan weergeven. Het grote nadeel van gSysC is dat deze de signalen/channels bundelt, waardoor het gevisualiseerd model snel onoverzichtelijk wordt. Om gSysC aan te passen om ook de signalen volgens een lay-out algoritme te tonen is hoogstwaarschijnlijk teveel werk, omdat de gehele implementatie van gSysC daardoor moet worden aangepast.

Het uiteindelijke formaat dat gekozen wordt, is GraphML. Dit formaat wordt gekozen omdat het alles biedt wat nodig is en daarbij ook uitbreidbaar is. Dit is erg handig als er in de toekomst extra wensen ontstaan, waardoor het formaat uitgebreid moet worden.
Als tekenprogramma wordt gekozen voor yEd. Dit wordt ten eerste gedaan omdat de opdrachtgever niet wenst zelf een tekenprogramma te ontwikkelen, ook al is dat in dit geval de beste optie, omdat geen van alle tekenprogramma's perfect passen bij de aangegeven eisen. Een van de redenen van het niet ontwikkelen van een tekenprogramma is dat dit hoogstwaarschijnlijk niet binnen de tijdsperiode van de afstudeerstage past. Mocht dit tekenprogramma in de toekomst worden ontwikkeld dan kan dit programma gebruik maken van onderdelen uit de volgende programma's: OGDF, Block diagram Editor en gSysC. Deze library en programma's zijn open source en bieden allemaal onderdelen die genodigd zijn in een tekenprogramma die aan de eisen van de opdrachtgever moet gaan voldoen.

# 5 Ontwikkeling SCMDL2GraphML

## 5.1 Analyse

Naar aanleiding van het onderzoek en als praktische resultaat wordt er een programma genaamd SCMDL2GRAPHML ontwikkeld waarmee de SCMDL die SHaBE genereerd kan worden omgezet naar GraphML.

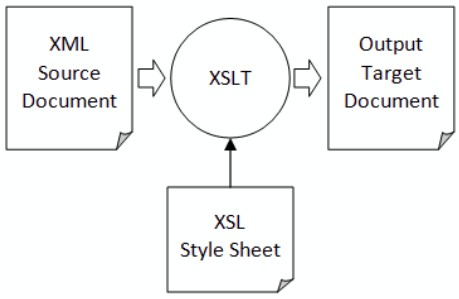
## 5.2 Ontwerp

SCMDL2GRAPHML maakt gebruik van XSLT conversie voor het converteren van de SCMDL naar GraphML. Er is voor XSLT conversie gekozen, omdat er gebruik wordt gemaakt van 2 formaten welke beide uit XML bestaan en naar elkaar moeten worden omgezet. XSLT is speciaal voor dit doeleinden gemaakt.
Alternatieven hadden kunnen zijn:

1. Een programma ontwikkelen welke de SCMDL inleest en vervolgens zelf alle tags omzet naar het GraphML formaat. Hier is niet voor gekozen, omdat deze oplossing omslachtig is, wanneer er ook XSLT kan worden gebruikt.
2. SHaBE aanpassen zodat SHaBE de hiërarchie niet alleen in het SCMDL formaat maar ook in het GraphML formaat kan opslaan. Hier wordt niet voor gekozen, omdat het SCMDL formaat er juist voor bedoeld is om alles uit een geëxtraheerd SystemC model in op te slaan. Er is 1 formaat waarmee andere ontwikkelaars kunnen doen wat zij willen. Anders zou SHaBE in de toekomst opnieuw aangepast moeten worden wanneer er andere formaten moeten worden ondersteund. Door een losstaand programma te ontwikkelen, wordt de visualisatie niet direct afhankelijk van SHaBE, maar slechts van het SCMDL formaat.

## 5.3 Implementatie

Tijdens de implementatie fase is er gekozen om gebruik te maken van de XalanC en XercesC library. Dit is gedaan, omdat deze libraries functionaliteiten van een XSLT processor bevatten, waardoor de XSLT transformatie vergemakkelijkt kan worden en de XSLT transformatie eigenlijk uitgevoerd kan worden door deze libraries. In afbeelding 5 is de conversie schematisch weergegeven.




*Afbeelding 5: XSLT transformatie*

## 5.4 Testen

Om te testen of de uitvoer van de conversie SCMDL2GraphML hoort bij de gegeven invoer wordt er gebruikt gemaakt van een unittest library, genaamd UnitTest++.
Met deze unittest library kan er worden getest of de GraphML uitvoer van SCMDL2GraphML is opgebouwd volgens de opbouw die benodigd is door yEd. Bij het uitvoeren van de tests wordt de hiërarchie van de XML elementen gecontroleerd. Alternatief zou kunnen zijn om een andere/uitgebreide testtechniek of een andere library te gebruiken.
Er is niet voor een andere testtechniek gekozen, omdat SCMDL een relatief klein programma is. Het zou raar zijn om een programma waar relatief weinig

ontwikkelingstijd in is gestopt enorm langdurig te testen. Hierdoor overschrijden de kosten voor het testen van het programma de kosten van het ontwikkelen van het programma, terwijl testen slechts een lage prioriteit heeft, omdat het slechts om een proof of concept gaat. Met dit onderzoek wordt alleen aangetoond dat er nu een manier is om SystemC modellen te visualiseren en hoe dit er dan uit komt te zien.
Er is ook niet voor een andere Unit test framework gekozen, omdat UnitTest++ een redelijk klein framework is. Voor de tests die uitgevoerd worden is er ook geen uitgebreid framework nodig. Het belangrijkste wat het framework moet kunnen is het bijhouden van geslaagde en niet geslaagde test en als een test geslaagd is wat de conditie is die niet gepasseerd kon worden. De UnitTest++ library is tot deze taken is staat en wordt daarom gekozen om te gebruiken tijdens het maken van de tests.

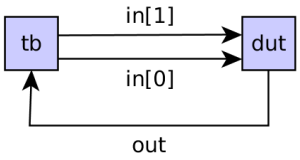
### 5.5 Resultaat

Om het resultaat dat is verkregen door het visualiseren van een SystemC model toe te lichten wordt het onderstaande SystemC model gebruikt. Hierin wordt een 2 tot de macht N AND-port beschreven waarbij N >= 0 is. Hierbij heeft de AND port een aantal ingangen van 2 tot de macht N. De implementatie van het gedrag van het model is niet beschreven.

```
template <unsigned int N>
struct Two_pow_N {
          static const unsigned int value = 2 * Two_pow_N<N-1>::value;
};

template <>
struct Two_pow_N<0> {
          static const unsigned int value = 1;
};

SC_MODULE(And2) {
          sc_in<bool> in[2];
          sc_out<bool> out;
          SC_CTOR(And2) {}
};

template <unsigned int N>
SC_MODULE(And2_N) {
          sc_in<bool> in[Two_pow_N<N>::value];
          sc_out<bool> out;
          SC_CTOR(And2_N): and0("and0"), and1("and1"), and2("and2") {
                    for (unsigned int i(0); i<Two_pow_N<N-1>::value; ++i) {
                              and0.in[i](in[i]);
                              and1.in[i](in[Two_pow_N<N-1>::value + i]);
                    }
                    and0.out(sig02);
                    and1.out(sig12);
                    and2.in[0](sig02);
                    and2.in[1](sig12);
                    and2.out(out);
          }
private:
          And2_N<N-1> and0, and1;
          And2 and2;
          sc_signal<bool> sig02, sig12;
};

template <>
struct And2_N<1>: public And2 {
          And2_N<1>(sc_module_name name): And2(name) {}
};

template <unsigned int N>
SC_MODULE(TB) {
public:
          sc_out<bool> stimulus[Two_pow_N<N>::value];
          sc_in<bool> response;
          SC_CTOR(TB){}
};

int sc_main(int argc, char* argv[]) {
```

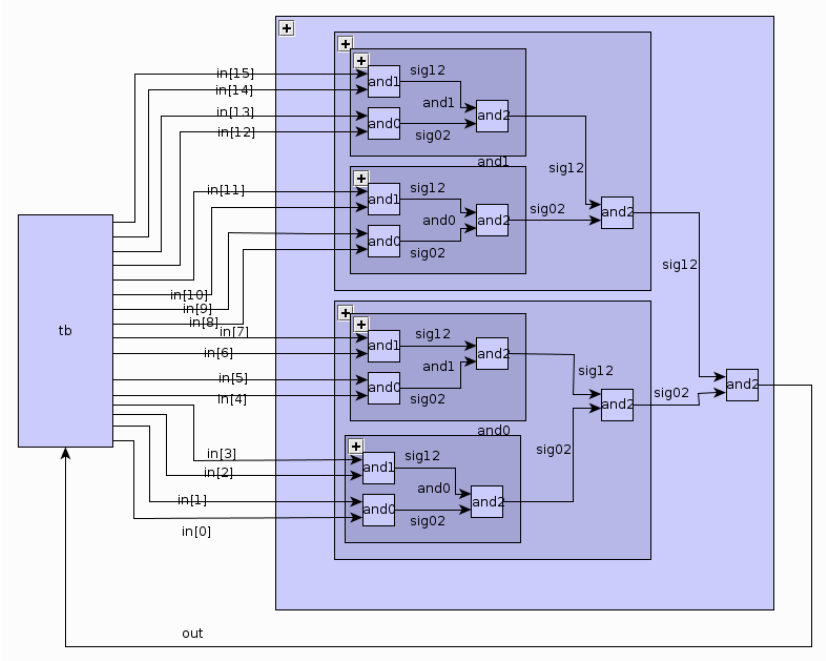
```
        const int N = 1;
        sc_signal<bool> in[Two_pow_N<N>::value], out;

        TB<N> tb("tb");
        And2_N<N> dut("dut");

        for (unsigned int i(0); i<Two_pow_N<N>::value; ++i) {
                tb.stimulus[i](in[i]);
                dut.in[i](in[i]);
        }
        dut.out(out);
        tb.response(out);

        sc_start(1000, SC_NS);
        return 0;
}
```

Wanneer dit voorbeeld wordt geëxtraheerd door SHaBE en de SCMDL uitvoer wordt omgezet naar GraphML, dan wordt dit model gevisualiseerd in yEd als het SystemC model in afbeelding 6.
Wanneer `const int N = 1;` wordt gebruikt in sc_main, komt de complexe implementatie van het voorbeeld niet volledig tot zijn recht, maar ontstaat er wel een simpel SystemC model en wordt er tijdens de extractie een simpel SCMDL bestand gegenereerd. Zie bijlage C.




*Afbeelding 6: SystemC model AND-Ports $2^N$ met N= 1*

Wanneer `const int N = 4;` wordt gebruikt, dan ontstaat er echter al een complexer SystemC model en wordt er duidelijker waarom de ingewikkelde implementatie van het voorbeeld nodig is. Zie afbeelding 7.
Wanneer het SCMDL van dit voorbeeld geanalyseerd wordt, blijkt het SCMDL bestand uit meer dan 430 regels te bestaan! Vanwege de grootte van het bestand is het dan ook veel makkelijker om de hiërarchie van het SystemC model in afbeelding 7 te bekijken i.p.v. in het SCMDL bestand.




*Afbeelding 7: SystemC model AND-Ports 2N met N= 4*

# 6 Onderzoek TLM Extractie

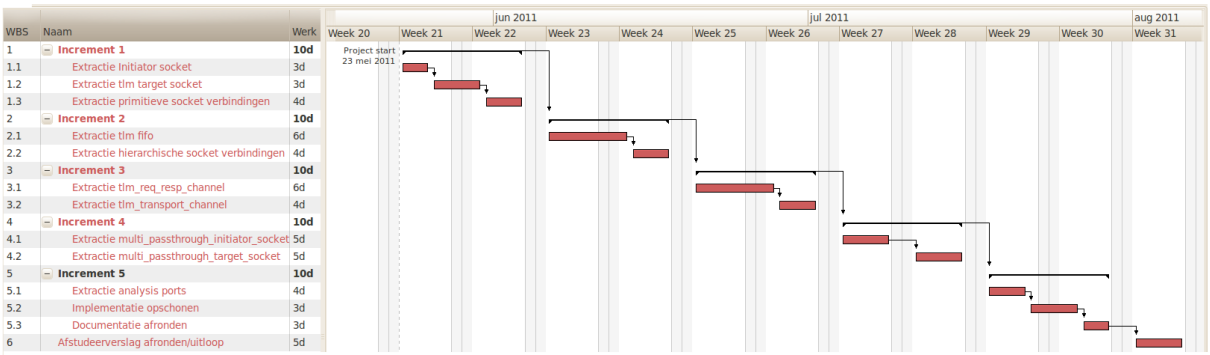
Nadat het onderzoek naar het visualiseren van een SystemC model is uitgevoerd, wordt er onderzoek gedaan naar het extraheren van TLM door SHaBE.
Dit onderzoek is te vinden in bijlage B.
Tijdens dit onderzoek zijn er diverse vragen gesteld om te onderzoeken of TLM geëxtraheerd kan worden. Hieronder een lijst van vragen die er voorafgaand aan het onderzoek zijn opgesteld:

- Wat is TLM?
- Wat is het verschil tussen SystemC en TLM?
- Uit welke onderdelen bestaat TLM?

Er is gekozen om deze vragen te beantwoorden zodat er een goed beeld kan worden gevormd bij wat TLM is. Het is namelijk beter om een groter deel van de tijd te besteden aan het onderzoeken wat TLM is, hoe het werkt en waar het uit bestaat en daardoor een kleiner gedeelte van de tijd aan de daadwerkelijk extractie, dan andersom.
Wanneer er meer tijd aan de analyse fase wordt besteed, is de kans dat er tijdens de ontwerp en constructie fase wordt ontdekt dat TLM niet geëxtraheerd kan worden namelijk kleiner, als visa versa. Wanneer er later in het ontwikkelproces pas ontdekt wordt dat TLM niet te extraheren valt, betekent dit dat er tijd is besteed aan de ontwikkeling terwijl dit voorkomen had kunnen worden. Tijdens het onderzoek wordt duidelijk dat TLM waarschijnlijk wel te extraheren valt. Dit wordt gedacht, omdat TLM klasses voor het grootste gedeelte afgeleide zijn van SystemC klasses en alle benodigde SystemC klasses kunnen al reeds worden geëxtraheerd door SHaBE. Om aan dit onderzoek een duidelijke conclusie te kunnen geven, is ervoor gekozen om een proof of concept te maken, waarmee kan worden aangetoond of TLM uit een SystemC model kan worden geëxtraheerd.

## 6.1 Planning

Het onderzoek naar de visualisatie van SystemC modellen is sneller afgerond dan werd verwacht, daarom wordt er een nieuwe planning gemaakt.
Er is gekozen om deze planning pas aan het begin van het TLM onderzoek te maken, omdat er tijdens dit gedeelte van het onderzoek wordt vastgesteld wat de extractie van TLM precies inhoud en welke TLM onderdelen er geëxtraheerd kunnen worden. Aan de hand van het aantal te extraheren onderdelen en de wijze waarop deze onderdelen afhankelijk zijn van elkaar kan er een nieuwe planning worden gemaakt.
Deze nieuwe planning geldt als globale planning voor de komende 11 weken. Tijdens deze 11 weken moeten er 7 voorbeelden worden geëxtraheerd en iedere 2 weken wordt er een increment afgerond.
Hierdoor is de onderstaande planning ontstaan.



| WBS | Naam | Werk |
|---|---|---|
| 1 | **Increment 1** | 10d |
| 1.1 | Extractie Initiator socket | 3d |
| 1.2 | Extractie tlm target socket | 3d |
| 1.3 | Extractie primitieve socket verbindingen | 4d |
| 2 | **Increment 2** | 10d |
| 2.1 | Extractie tlm fifo | 6d |
| 2.2 | Extractie hierarchische socket verbindingen | 4d |
| 3 | **Increment 3** | 10d |
| 3.1 | Extractie tlm_req_resp_channel | 6d |
| 3.2 | Extractie tlm_transport_channel | 4d |
| 4 | **Increment 4** | 10d |
| 4.1 | Extractie multi_passthrough_initiator_socket | 5d |
| 4.2 | Extractie multi_passthrough_target_socket | 5d |
| 5 | **Increment 5** | 10d |
| 5.1 | Extractie analysis ports | 4d |
| 5.2 | Implementatie opschonen | 3d |
| 5.3 | Documentatie afronden | 3d |
| 6 | Afstudeerverslag afronden/uitloop | 5d |



## 6.2 Resultaat onderzoek

Nadat de nieuwe planning is gemaakt, wordt er verdergegaan met het onderzoek naar TLM extractie. Hieronder een samenvatting van de vragen die vervolgens in het onderzoek zijn beantwoord.

### 6.2.1 Wat is TLM?

TLM staat voor Transaction Level Modelling. Door het gebruik van TLM kunnen er modellen op een abstracter niveau als bijvoorbeeld in RTL worden gemaakt. Bij een TLM model wordt de communicatie tussen verschillende onderdelen gemodelleerd m.b.v. transactions. Dit zijn berichten die tussen componenten worden verstuurd.

### 6.2.2 Wat is het verschil tussen SystemC en TLM?

Door het gebruik van TLM kan een SystemC model op een abstracter niveau als in SystemC en volgens een standaard manier van modelleren worden gemodelleerd. Op RTL niveau wordt er beschreven welke modules er gevoelig zijn voor welk signaal en hoe deze modules op dit signaal zullen reageren. Hierbij wordt het systeem al snel op pin niveau gemodelleerd. Op TLM niveau wordt er beschreven welke transactions er door verschillende modules gestuurd kunnen worden en welke channels welke protocollen gebruiken om die transacties te versturen. Wanneer een model op TLM niveau wordt gemodelleerd wordt er gemodelleerd welke data er tussen de modules wordt gestuurd i.p.v. hoe deze data tussen de modules wordt overgebracht.

### 6.2.3 Uit welke onderdelen bestaat TLM?

TLM bestaat uit verschillende onderdelen. Omdat er tijdens het onderzoek alleen wordt onderzocht of de hiërarchie van TLM modules kan worden geëxtraheerd, worden ook alleen deze onderdelen besproken. Hieronder een overzicht van deze onderdelen.

#### tlm_initiator_socket & tlm_target_socket

Door het gebruiken van 2 sockets kunnen 2 modules met elkaar communiceren. Een socket is een combinatie van een sc_port en een sc_export. Hierdoor is er tweerichtingsverkeer mogelijk tussen een initiator en target socket.

#### tlm_fifo

tlm_fifo is een channel. Een channel is de verbinding tussen 2 ports. tlm_fifo is een speciaal soort channel, deze kan namelijk de data die wordt verstuurd bufferen.

**tlm_req_resp_channel & tlm_transport_channel**
Dit zijn 2 channels waarmee tweerichtingsverkeer tussen ports kan worden gesimuleerd. Beide channels hebben hun eigen specifieke eigenschappen.

**multi_passthrough_initiator_socket en Multi_passthrough_target_socket**
multi_passthrough sockets zijn sockets die functioneren als multi ports in SystemC. multi_passthrough sockets zijn sockets die meerdere verbindingen aan kunnen gaan met andere sockets. Zo is er bijvoorbeeld maar een multi_passthrough_target_socket nodig waaraan twee tlm_initiator_socket's kunnen worden gekoppeld.

**analysis_port**
Door het gebruik van een analysis_port kan een module bijvoorbeeld log info naar een andere port in een andere module sturen. Deze port kan vervolgens geen data terugsturen naar de analyse_port.

### 6.3 Tussentijdse conclusie
TLM is bestudeerd en er wordt gedacht dat het mogelijk moet zijn om TLM onderdelen uit een SystemC model te extraheren. Dit wordt gedacht, omdat TLM klasses voor het grootste gedeelte afgeleide klasses zijn van SystemC klasses. Klasses die worden afgeleid van SystemC klasses kunnen al worden herkent. Deze worden namelijk ook aangemaakt en gebruikt wanneer er een SystemC model wordt gemaakt. Er wordt daarom besloten om een proof of concept te ontwikkelen waarmee kan worden aangetoond of de hoofdvraag “Kunnen TLM onderdelen uit een SystemC model worden geëxtraheerd?” falsificeerbaar is. Onder de term TLM onderdelen vallen de TLM onderdelen die zijn gevonden tijdens het beantwoorden van de vraag “Uit welke onderdelen bestaat TLM?”

### 6.4 Aanpak verdere verloop van het onderzoek
Zoals in de aanpak uit hoofdstuk 3.4 is besproken, wordt er volgens een Agile manier gewerkt. Omdat er in het onderzoek naar de extractie van TLM al is vastgesteld dat er een vast aantal onderdelen uit TLM moeten worden geëxtraheerd, heeft het niet veel zin om volgens een aanpak te ontwikkelen, waarbij er rekening gehouden moet worden met wisselende eisen. De eisen staan echter nu al vast en omdat het onderzoek is gericht op TLM 2.0 hoeft er geen rekening gehouden te worden met een eventuele nieuwe versie van TLM welke tussentijds kan worden uitgegeven. Omdat ook de volgorde van het implementeren van de onderdelen uit TLM redelijk vaststaat, door de afhankelijkheid van verschillende TLM onderdelen, moet er volgens een vaststaande aanpak worden ontwikkeld. Echter is het extraheren van TLM uit een SystemC model een vrij specialistische opdracht en omdat de opdrachtgever veel kennis heeft van het extraheren van gegevens uit een SystemC model en het modelleren van hardware in SystemC, wordt er gekozen om met de eerder gekozen Agile methode te blijven werken. Door het gebruik van deze aanpak blijft er veel contact met de opdrachtgever om ervoor te zorgen dat alle TLM onderdelen ook worden gebruikt en worden geëxtraheerd uit de voorbeelden zoals verwacht wordt. Problemen die de opdrachtgever bij het extraheren van onderdelen uit de SystemC library uit een SystemC model heeft gehad kunnen

worden gedeeld wanneer er tijdens het onderzoek tegen eenzelfde probleem wordt aangelopen.
Omdat er voordat er wordt begonnen met de ontwikkeling van de extractie van een voorbeeld al kan worden bepaald wat de uitvoer in SCMDL zou moeten zijn wordt er voor de testtechniek TestDriven development gekozen. Bepaalde aspecten zoals het opslaan van een module zijn al vastgelegd in het SCMDL formaat. Hierdoor is het mogelijk om voorafgaand aan de ontwikkeling van de extractie en nadat het voorbeeld is ontwikkeld te bepalen wat de geëxtraheerde gegevens in het SCMDL zouden moeten zijn. Nieuwe tags in het SCMDL formaat worden tijdens de ontwerpfase vastgelegd, waardoor het maken van een verwacht testresultaat mogelijk is voordat er wordt begonnen aan de implementatie.

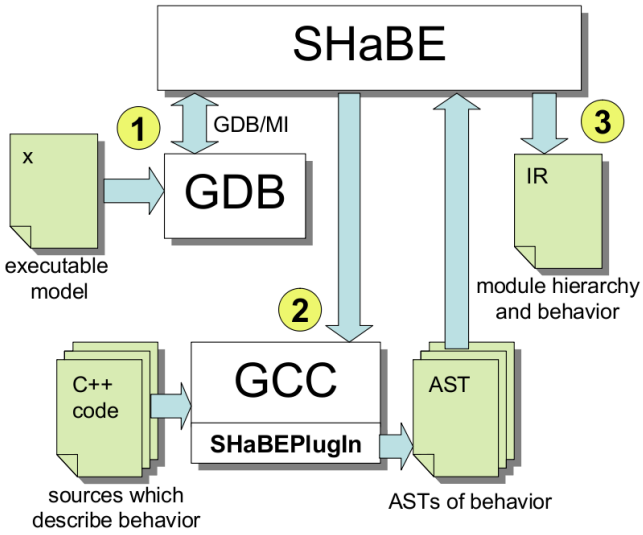
### 6.5 Werking SHaBE

Voorafgaand aan de ontwikkeling is er geanalyseerd hoe SHaBE verschillende SystemC onderdelen uit een SystemC model extraheert. Een grafische weergave van de globale werking van SHaBE is te zien in afbeelding 8.

Hierin zijn 3 stappen te onderscheiden:

1. Hiërarchische structuur van het SystemC model wordt geëxtraheerd uit de executable.
2. Gedrag van SystemC model wordt geëxtraheerd.
3. Hiërarchische structuur en gedrag worden aan elkaar gekoppeld en opgeslagen in het SCMDL formaat.

Tijdens het TLM onderzoek wordt er alleen op stap 1 en het opslaan in stap 3 gericht.




*Afbeelding 8: Globale werking van SHaBE*

# 7 Ontwikkeling TLM extractie

In de volgende hoofdstukken wordt het ontwikkelproces van de 5 incrementen beschreven. Zoals in de planning in het vorige hoofdstuk is te zien worden er 7 voorbeelden geëxtraheerd. Deze voorbeelden worden voorafgaand aan de incrementen ontwikkeld, zodat de afstudeerder zich beter in SystemC en TLM kan verdiepen voordat er aan het uitbreiden van SHaBE wordt begonnen. Door het vooraf ontwikkelen van de voorbeelden worden de te testen onderdelen uit TLM gedefinieerd. Deze voorbeelden worden zelf gemaakt. Dit heeft twee voordelen:

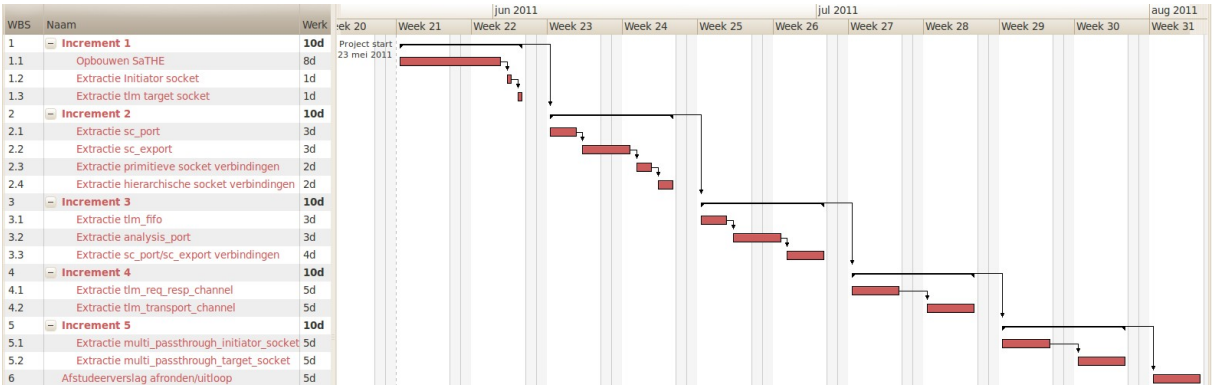
1. Voorbeelden die via internet beschikbaar zijn, zijn complexer dan nodig is en deze SystemC modellen bevatten veel gedrag. Er is voorafgaand aan het uitvoeren van de opdracht besloten dat het extraheren van gedrag buiten de scope van dit project valt, waardoor er dus veel meer informatie in een SystemC model wordt gebruikt als nodig is voor de extractie.
2. Door de voorbeelden zelf te ontwikkelen kan er beter in de functionaliteit van TLM worden verdiept en kan er per TLM onderdeel wat geëxtraheerd moet worden een nieuw voorbeeld worden gemaakt. Wanneer er voorbeelden herbruikt worden van internet zijn er diverse onderdelen samengevoegd in 1 voorbeeld, waardoor er moeilijk te testen is of het juiste TLM onderdeel wel wordt geëxtraheerd. Tevens kan er dan niet per increment worden getest of de juiste onderdelen zijn geëxtraheerd.

Het doel van deze aanpak is om op deze manier ook de voorbeelden te kunnen gebruiken voor de testfase. Er wordt volgens een Test Driven aanpak gewerkt waardoor het noodzakelijk is om voorafgaand aan de implementatie fase de test output te kunnen definiëren. Hierdoor blijft er gedurende de increment een duidelijk doel voor ogen en dat is om de noodzakelijke tests succesvol te kunnen uitvoeren.

# 8 Increment 1 – Extractie tlm sockets

## 8.1 Analyse

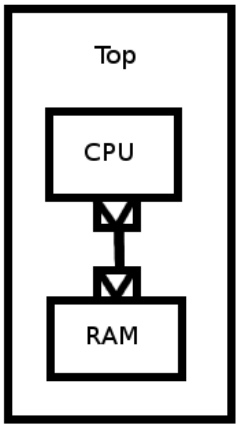
Tijdens de analyse van SHaBE wordt er ontdekt dat de structuur waarmee SHaBE is opgebouwd niet erg uitbreidbaar is, waardoor de keuze gemaakt wordt om een TLM versie van SHaBE te ontwikkelen. Hierbij wordt SHaBE niet aangepast, maar wordt er een zelfstandige versie van SHaBE maar dan op het gebied van TLM extractie ontwikkeld.
Alternatief is om op SHaBE verder te bouwen, maar omdat het programma niet erg onhoudbaar en uitbreidbaar is opgebouwd, is het onverstandig om op deze manier verder te gaan. Wanneer een volgende ontwikkelaar SHaBE wil uitbreiden zal deze persoon op hetzelfde probleem stuitte waardoor de drempel om SHaBE opnieuw op te bouwen nog hoger wordt. Wanneer er nu echter al een beter onderhoudbare en beter uitbreidbare versie van TLM en SystemC extractie wordt opgezet betekent dit dat SHaBE in de toekomst een betere structuur krijgt waardoor de verdere ontwikkeling van SHaBE op een gestructureerde manier kan worden voortgezet.
Deze keuze heeft erg veel invloed op de planning waardoor het nodig is een nieuwe planning te maken.
Door het opbouwen van een nieuwe applicatie, genaamd SaTHE(SystemC and TLM Hierarchy Extractor) moet er ook rekening worden gehouden dat er ook onderdelen die door SHaBE worden geëxtraheerd ook door SaTHE moeten worden geëxtraheerd, zoals een sc_module, sc_port en sc_export. Deze onderdelen worden daarom in de nieuwe planning opgenomen. Omdat er nu ook een sc_port en sc_export geëxtraheerd moeten gaan worden wordt ervoor gekozen om een extra voorbeeld te maken waarbij de sc_port en sc_export worden gebruikt. Hierdoor ontstaan er dus in totaal 8 voorbeelden.
De nieuwe planning ziet er als volgt uit:




| WBS | Naam | Werk |
|---|---|---|
| 1 | Increment 1 | 10d |
| 1.1 | Opbouwen SaTHE | 8d |
| 1.2 | Extractie Initiator socket | 1d |
| 1.3 | Extractie tlm target socket | 1d |
| 2 | Increment 2 | 10d |
| 2.1 | Extractie sc_port | 3d |
| 2.2 | Extractie sc_export | 3d |
| 2.3 | Extractie primitieve socket verbindingen | 2d |
| 2.4 | Extractie hierarchische socket verbindingen | 2d |
| 3 | Increment 3 | 10d |
| 3.1 | Extractie tlm_fifo | 3d |
| 3.2 | Extractie analysis_port | 3d |
| 3.3 | Extractie sc_port/sc_export verbindingen | 4d |
| 4 | Increment 4 | 10d |
| 4.1 | Extractie tlm_req_resp_channel | 5d |
| 4.2 | Extractie tlm_transport_channel | 5d |
| 5 | Increment 5 | 10d |
| 5.1 | Extractie multi_passthrough_initiator_socket | 5d |
| 5.2 | Extractie multi_passthrough_target_socket | 5d |
| 6 | Afstudeerverslag afronden/uitloop | 5d |

Er wordt gekozen om voorbeeld 7 te implementeren in increment 3, omdat er wordt geschat dat voorbeeld 7 qua implementatie niet 2 weken kost en tijdens increment 3 kan nu ook de extractie van sc_port en sc_export verbindingen worden geïmplementeerd.

Voorbeeld 1 wordt gebruikt voor de extractie in increment 1. Een schematische weergave van voorbeeld 1 is te zien in afbeelding 9. In dit SystemC model zijn 3 modules aanwezig, Top, CPU en RAM. De module Top is de topmodule en bevat 2 submodules, CPU en RAM. CPU bevat een initiator socket en RAM een target socket. Beide sockets zijn met elkaar verbonden, waardoor er informatie tussen de 2 modules kan worden uitgewisseld.



*Afbeelding 9: Hiërarchie uit voorbeeld 1*

### 8.2 Ontwerp
Tijdens het ontwerp en de implementatie zijn er verschillende keuzes gemaakt. In het onderzoek in bijlage B zijn alle keuzes beschreven. Hier volgt een overzicht van de belangrijkste keuzes.

#### 8.2.1 Benaming
Het programma SaTHE moet er voor zorgen dat er SystemC en TLM klasses worden geëxtraheerd uit een SystemC model. Om dit te doen wordt er gebruikt gemaakt van de GDB debugger, omdat deze ook door SHaBE wordt gebruikt. In SystemC en TLM zijn deze te extraheren onderdelen afgeleid van een sc_object klasse. In de extractie wordt er voor gekozen om ook alle te extraheren objecten af te leiden van een ScObject klasse zodat duidelijk is dat dit een geëxtraheerde SystemC of TLM klasse is. Ieder sc_object bevat een address, parentaddress en een name. Omdat ieder sc_object deze gegevens bevatten worden deze gegevens opgeslagen in een object van de klasse ScObject.
Iedere afgeleide van ScObject wordt door een afgeleide van de ScObjectExtractor geëxtraheerd. Door de namen van de afgeleide klasses van ScObject dezelfde ScObject naam te geven is er duidelijk te zien dat een bepaalde afgeleide van ScObjectExtractor een bepaalde afgeleide van ScObject extraheert.

#### 8.2.2 Down-Top benadering
Om SystemC en TLM onderdelen uit een SystemC model te extraheren met GDB is het nodig om breakpoints te plaatsen. Wanneer er bijvoorbeeld een breakpoint in de initiator socket constructor wordt geplaatst en deze wordt geraakt, dan is het duidelijk dat er een initiator socket wordt aangemaakt in de SystemC simulatie. Omdat bepaalde eigenschappen zoals de name en het parentaddress van de initiator socket in een base class “sc_object” van de initiator class worden opgeslagen, moet er omhoog in de klassenhiërarchie worden gezocht naar deze base class. Een object van deze klasse bevat namelijk de te extraheren name en parentaddress van het sc_object.
Bovenstaande beschrijving is de Down-Top benadering, omdat er vanuit de te extraheren afgeleide klasse naar sc_object, de 'most-base' klasse, wordt beredeneerd. De Top-Down benadering, die op dit moment in SHaBE wordt toegepast, heeft als voordeel dat er maar 1 breakpoint geplaatst hoeft te worden, namelijk in  sc_object. Wanneer dit breakpoint wordt geraakt kunnen gelijk de naam en het parentaddress worden geëxtraheerd, omdat deze zich in sc_object bevinden. Vervolgens kan er m.b.v. de callstack naar beneden worden gegaan in de klassenhiërarchie om te bepalen wat de 'most-derived' klasse van het sc_object is. Er is tijdens het ontwerpen van SaTHE gekozen om van de Down-Top benadering gebruik te maken. Een van de onderdelen die SHaBE

minder goed onderhoudbaar maakte was het gebruik van de Top-Down benadering. Door het gebruik van deze methode moet er wanneer er een sc_object geëxtraheerd moet worden, worden bepaald van welk type dit sc_object is. Hierdoor ontstaan er dus verschillende if/else ladders waarmee wordt bepaald wat voor type sc_object er aangemaakt moet worden in SHaBE. Door te kiezen voor een Down-Top benadering wordt er voor iedere te extraheren afgeleide klasse van sc_object een breakpoint geplaatst. Wanneer er een callback functie wordt aangeroepen wanneer dit breakpoint wordt geraakt, is het duidelijk welk type sc_object er in SaTHE moet worden aangemaakt. Een nadeel van deze methode is wel dat wanneer er extra sc_object's geëxtraheerd moet worden, dat SaTHE uitgebreid moet worden en extra breakpoints in het SystemC model moet plaatsen. Het voordeel van dit nadeel is wel dat er nu op een overzichtelijke manier onderscheid wordt gemaakt tussen het extraheren van de verschillende klasses, waardoor er makkelijk extra eigenschappen van een klasse kunnen worden geëxtraheerd die niet ieder ScObject heeft.

### 8.2.3 Aparte klasse voor aparte functionaliteiten

Wanneer er goed naar de opbouw van de extractie wordt gekeken zijn er diverse onderdelen te ontdekken. Dit zijn namelijk:

- Extractie van sc_objecten. Dit gebeurd wanneer de constructor van de afgeleide van sc_object worden aangeroepen.
- Extraheren van verbindingen tussen sc_object's. Dit gebeurd bij het aanroepen van functies waarmee een verbinding wordt aangemaakt.
- Verbinden van de sc_objecten volgens de geëxtraheerde verbindingen.
- Opbouwen van de hiërarchie van de sc_object's.
- Opslaan van de hiërarchie.

Voor ieder van deze functionaliteiten worden er aparte klasses ontworpen. Er wordt voor OO(Object Oriented) aanpak gekozen, omdat SHaBE ook volgens een OO manier is opgebouwd en de verschillende functionaliteiten op deze manier beter van elkaar worden gescheiden door ze in klasses op te splitsen. Zo heeft iedere klasse zijn eigen 'responsibility'. Hierdoor kunnen de bovenstaande functionaliteiten losser van elkaar worden ontwikkeld, dan wanneer dit principe niet wordt toegepast.
Door het toepassen van dit mechanisme hoeven de onderdelen:

- Extraheren van verbindingen tussen sc_object's. Dit gebeurd bij het aanroepen van functies waarmee een verbinding wordt aangemaakt.
- Verbinden van de sc_objecten volgens de geëxtraheerde verbindingen.

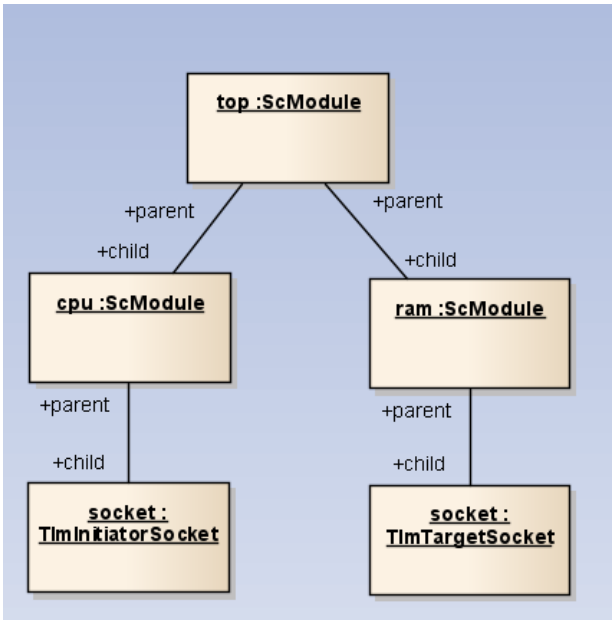
nog niet te worden ontwikkeld in dit increment en kan de ontwikkeling worden verplaatst naar het volgende increment, zodat ieder increment 2 weken aan ontwikkeltijd in beslag neemt en increment 1 op deze manier minder kans heeft om meer dan 2 weken tijd in beslag te nemen. In increment 1 moet namelijk een hele opzet van SaTHE worden ontwikkeld. De structuur die wordt ontwikkeld moet goed uitbreidbaar en overzichtelijk blijven zodat er een goed uitbreidbare eerste versie van SaTHE ontstaat, zodat er in de volgende incrementen op SaTHE verder kan worden ontwikkeld. De afstudeerder schat dat hij daar veel tijd aan kwijt is en daarom is het goed om van te voren alvast vast te stellen dat er specificaties naar een volgend increment moeten worden verschoven.

### 8.2.4 Callback mechanisme

In SHaBE waren diverse onderdelen te vinden welke niet erg goed uitbreidbaar en overzichtelijk waren. Zo ook het mechanisme waarmee wordt bepaald welk breakpoint is geraakt tijdens het debuggen van het SystemC model. Dit is in SHaBE geïmplementeerd als een grote if/else ladder waarin wordt gecontroleerd welk breakpoint wordt geraakt. Om hier verandering in aan te brengen is er tijdens het ontwerp van SaTHE gekozen om een BreakPointHandler klasse te implementeren. Door het aanroepen van de register methode van deze klasse kunnen er BreakPoint's worden geregistreerd. Ieder breakpoint heeft een methode handle. Hierin wordt het opgegeven breakpoint afgehandeld. Maar als er voor iedere breakpoint een andere handle methode moet worden geimplemeteerd, wordt het het programma onoverzichtelijk i.v.m. met het vele malen overerven van een klasse, terwijl er een betere optie beschikbaar is. Daarom wordt ervoor gekozen om een CustomFunctorBreakPoint aan te maken. Aan dit object kan een functor, een functie van een object, worden meegegeven. Zodra de opgegeven breakpoint wordt geraakt, wordt deze functie als callback functie aangeroepen. CustomFunctorBreakPoint is afgeleid van een abstracte klasse BreakPoint, zodat er later bijvoorbeeld nog een nieuw type BreakPoint kan worden aangemaakt die als callback een gewone functie/C-functie aanroept. Door het gebruik maken van dit callback mechanisme kunnen alle afgeleide klasses van ScObjectExtractor welke zorgen voor het zetten van de breakpoints, ook zelf callback functies definiëren, welke worden aangeroepen als een door hun gezet breakpoint wordt geraakt. Hierdoor is er per te extraheren ScObject een afgeleide van de ScObjectExtractor klasse. Hierin worden zowel de breakpoints als de callback functies geïmplementeerd. Hierdoor blijft het overzichtelijk in welke klasses welke functionaliteit is geïmplementeerd. Een afgeleide van een ScObjectExtractor zorgt namelijk nu voor de gehele extractie van één afgeleide klasse van ScObject.

### 8.2.5 Opbouw van de hiërarchie

Om die hiërarchie van geëxtraheerde ScObject's op te slaan wordt er gekozen voor een boomstructuur. Zoals in afbeelding 10 is te zien kan voorbeeld 1 in de vorm van een boom worden weergegeven. Dit komt doordat ieder ScObject een address en een parentaddress heeft. Door deze te koppelen kan er een boomstructuur worden aangemaakt. Tijdens de implementatie waren er 2 mogelijkheden om deze boomstructuur te implementeren.



*Afbeelding 10: Opgebouwde hiërarchie van voorbeeld 1*

1. Het ScObject object bevat een lijst met alle ScObject's welke een child zijn van het ScObject.
2. Er wordt een TreeNode klasse ontwikkeld die een verwijzing bevat naar een ScObject en een lijst van verwijzingen naar andere TreeNodes welke verwijzingen bevatten naar ScObject's die child zijn van het ScObject.

Er wordt voor optie 2 gekozen, omdat bij optie 2 de objecten van de klasse ScObject puur gebruikt blijven worden om de gegevens die geëxtraheerd zijn in op te slaan. De klasse TreeNode is een klasse die een verwijzing naar child TreeNode's kan bevatten zoals in een boomstructuur. Op dit moment is een module het enige ScObject wat andere ScObject's kan bevatten, dus is het raar als een ScObject een lijst met child ScObject's gaat bevatten, terwijl maar een van de afgeleide klasses genaamd ScModule verschillende child ScObject's kan bevatten. Wanneer ScObject namelijk een lijst gaat bevatten met andere ScObject's, dan moet ScObject ook een functie krijgen waarmee er ScObject's worden toegevoegd aan deze lijst.
Wanneer dit wordt gedaan, wordt het OO Liskov Substitution principe geschonden, omdat afgeleide klasses hierdoor juist minder kunnen als in de base klasse wordt gesuggereerd. Niet alle afgeleide klasses van ScObject bevatten namelijk andere ScObject's tijdens de SystemC simulatie en deze objecten hoeven daarom geen functie te krijgen waarmee er child ScObject's aan hun lijst kunnen worden toegevoegd.
Andere optie is om alleen de klasse ScModule een lijst met child ScObject's te geven. Hier wordt niet voor gekozen, omdat het dan nodig is om ieder ScObject te proberen te casten naar een ScModule om te kijken of dit object een ScModule is, als dit een ScModule is kan vervolgens de lijst met de child ScObject's worden opgehaald. Er wordt dus ook niet voor dit alternatief gekozen, omdat casten in dit geval niet nodig hoeft te zijn door het gebruik maken van een TreeNode in het ontwerp.

### 8.2.6 SCMDL

In het SCMDL formaat worden diverse gegevens opgeslagen, om goed uit te leggen welke gegevens er uit een SystemC model worden geëxtraheerd, wordt dit uitgelegd aan de hand van het SystemC model hieronder.

```
class Top: public sc_module{
public:
	Top( const sc_module_name& nm ): sc_module(nm){}
};

int sc_main( int argc , char **argv ){
	Top topVar("TopNAME");
	sc_start();
	return 0;
}
```

Uit het voorbeeld worden er 6 gegevens geëxtraheerd waarvan er 5 in het SCMDL formaat worden opgeslagen.

1. name(naam van de variabele waarin de afgeleide van sc_object is opgeslagen. In het voorbeeld is dat topVar)
2. type(most-derived klasse van het object. In het voorbeeld is dat Top)
3. systemc-name(hiërarchische SystemC naam. In het voorbeeld is dat TopNAME)
4. systemc-type(systemc type. In het voorbeeld is dat sc_module)
5. address(adres van het object. Deze wordt gegenereerd tijdens de elaboration fase, maar is gelijk aan de this variabele in de constructor van de base class sc_object)

6. parentaddress(address van de bovenliggende sc_object, in het voorbeeld heeft Top parentaddress 0x0 omdat Top geen parent sc_object heeft. Het parentaddress wordt niet opgeslagen in de SCMDL, maar wordt wel gebruikt om te bepalen hoe de hiërarchie van het model is opgebouwd.)

Tijdens dit increment zijn alleen de systemc-name, address en het parentaddress nodig om een eerste versie van SaTHE te kunnen opzetten. Er wordt daarom gekozen om alleen deze 3 gegevens te extraheren. Dit wordt gedaan zodat er niet achter op planning wordt geraakt, omdat er in dit increment veel ontwikkeld moet worden.

### 8.3 Implementatie

Tijdens de implementatie fase heeft zich een groot probleem voortgedaan, hieronder wordt dit probleem beschreven en hoe ermee is omgegaan.

#### 8.3.1 Template methodes

Verschillende afgeleide klasses van sc_object die in de SystemC library aanwezig zijn gebruiken template methodes en template klasses. Om breakpoints te kunnen plaatsen in deze methodes van deze klasses kan de naam van de template klasse en de naam van de template functie niet worden gebruikt om er een breakpoint op te zetten. De compiler maakt namelijk tijdens het compileren van alle template klasses en methodes nieuwe klasses en methodes aan waarin de meegegeven template argumenten worden ingevuld en wanneer de template argumenten worden genegeerd, kan er geen breakpoint worden geplaatst.
In het onderstaande stuk code is een template klasse Help met de template methode **`returnMeAValue`** gedefinieerd.
Wanneer er een breakpoint op de functie **`returnMeAValue`** moet worden geplaatst kan dit niet door een breakpoint op de volgende regel te plaatsen **`Help:: returnMeAValue()`**.

```
template<typename T>
class Help{
public:
	T returnMeAValue();
};
```

Dit komt door het volgende:
Wanneer onderstaande main functie gecompileerd wordt, maakt de compiler 2 nieuwe klasses aan. Allebei genaamd Help, maar met 2 verschillende template argumenten.

```
void main(){
	Help<int> a;
	Help<std::string> b;
}
```

Tijdens het compileren maakt de compiler de volgende klasses aan. Deze klasses worden opgeslagen in de debuginformatie en zijn daarom met de GDB debugger te achterhalen.

```
class Help<int>{
public:
	int returnMeAValue();
};

class Help<std::string>{
public:
	std::string returnMeAValue();
};
```

Voor dit probleem zijn er 2 mogelijke oplossingen:

1. Elke klasse en elke methode hebben een regelnummer. Deze kunnen worden opgezocht in de source code. Door dit regelnummer en het sourcecode bestand op te geven kan er een breakpoint op de juiste regel worden geplaatst.
2. Alle template klasses met ingevulde template parameters, zoals in bovenstaande stuk code, worden opgezocht in de debug informatie. In ieder mogelijke template methode van de template klasse wordt vervolgens een breakpoint geplaatst.

Er wordt voor optie 2 gekozen, omdat er door het implementeren van optie 2 een beter onderhoudbaar programma ontstaat. Optie 2 is namelijk minder gevoelig voor wijzigingen in de TLM source code. De functies die gebruikt worden in TLM zijn vastgelegd in de TLM standaard. De regelnummers waarop de functies beginnen zijn niet vastgelegd in deze standaard. Hierdoor wordt er geconcludeerd, dat regelnummers sneller kunnen veranderen, waardoor er door het implementeren van optie 1 verwacht wordt dat SaTHE sneller aangepast moet worden, omdat het bij een volgende minor release misschien nodig is geweest voor het SystemC ontwikkelteam om bepaalde code in de TLM of SystemC library aan te passen, waardoor de regelnummers zijn veranderd.

### 8.4 Testen

Om te testen of SaTHE werkt zoals verwacht wordt ervoor gekozen om alleen een zogenaamde blackbox test en memoryleak test uit te voeren. SaTHE is afhankelijk van een ingevoerd SystemC model en kan niet zonder de invoer van een SystemC model worden uitgevoerd. Een SystemC model kan op een oneindig aantal manieren worden opgebouwd, omdat SystemC een systeem beschrijvingingstaal is en een systeem kan uit een oneindig aantal manieren zijn opgebouwd. Er wordt daarom gekozen om alleen een blackbox test en geen unit test uit te voeren, omdat de gegevens die worden geëxtraheerd uit het SystemC model direct te vinden zijn in de SCMDL uitvoer van SaTHE. Deze gegevens worden gecontroleerd op juistheid m.b.v. de blackbox test. Wanneer er daarnaast ook nog eens een unit test gemaakt zou worden, is de meerwaarde van de unit test alleen dat er bepaalde delen in de code los van elkaar kunnen worden getest om te controleren of ieder deel wel doet wat het zou moeten doen. Wanneer echter een deel van SaTHE niet werkt zoals zou moeten, dan is dit in de meeste gevallen direct te zien aan de uitvoer van het SaTHE, want dan worden ook de onjuiste gegevens in de SCMDL gespecificeerd.

Om de blackbox tests uit te voeren wordt er tijdens de testfase een testprogramma gemaakt waarmee getest wordt of de juiste onderdelen tijdens

de extractie worden geëxtraheerd en opgeslagen in de SCMDL. Dit wordt gedaan door voorafgaand aan de test een SCMDL bestand aan te maken waarin de verwachte SCMDL wordt gedefinieerd. Door de extractie uit te voeren en de uitvoer te vergelijken met de verwachte resultaten kan worden bepaald of de test is geslaagd.
SaTHE is ontwikkeld in de programmeertaal C++. Het ontwikkelen in C++ brengt het voordeel met zich mee dat de programmeur zelf het geheugen kan beheren. Dit is tevens ook een nadeel, want de programmeur kan vergeten om bepaalde resources in zijn programma weer vrij te geven. Om deze reden wordt er gekozen om gebruik te maken van het programma genaamd Valgrind waar zogenaamde memory leaks mee opgespoord kunnen worden. Er is voor Valgrind gekozen, omdat dit programma gratis te gebruiken is en aangeeft op welke plekken er zich een (mogelijk) memory leak bevindt.
Een alternatief zou kunnen zijn om bijvoorbeeld Leak Tracer te gebruiken. Het nadeel van dit programma is, dat het als library is geïmplementeerd, waardoor het nodig is om iedere keer een nieuw testprogramma te bouwen welke gelinkt moet worden aan SaTHE en Leak Tracer. Valgrind is daarentegen een compleet losstaand gebouwd programma welke vanaf de command line gebruikt kan worden. De te testen executable kan bij Valgrind als parameter worden meegegeven. Hierdoor wordt het mogelijk om het testen van verschillende modellen en SaTHE in Valgrind automatisch te laten verlopen door in een test programma Valgrind via de commandline aan te roepen en te controleren of Valgrind memory leaks detecteert.

Tijdens de testfase is er ontdekt dat het design pattern Singleton zoals in het boek “Design Patterns” is beschreven als volgt moet worden geïmplementeerd:

```
BreakPointHandler* BreakPointHandler::getInstance(){
	if(_instance == NULL)
		_instance = new BreakPointHandler();
	return _instance;
}
```

Na nader onderzoek en uit resultaten van de test met Valgrind, blijkt dit geen slimme implementatie te zijn, omdat er op deze manier een memory leak ontstaat.
Dit komt omdat de BreakPointHandler die aangemaakt wordt in de getInstance methode niet wordt vrijgegeven. Om dit design pattern op een memory leak vrije manier toe te passen, zijn er 2 mogelijkheden:

1. Extra methode toevoegen waarmee het geheugen van het object in de variabele _instance wordt vrijgegeven. Deze zou dan bijvoorbeeld vanuit de main functie moeten worden aangeroepen. Omdat de main functie niet verantwoordelijk is voor het vrijgeven van de BreakPointHandler, omdat deze functie ook het geheugen niet heeft ingenomen is deze oplossing erg foutgevoelig, omdat de programmeur kan vergeten deze functie aan te roepen of niet weten dat deze functie moet worden aangeroepen.

2. Het geheugen niet dynamisch alloceren. Door het geheugen niet dynamische te alloceren, maar statisch en door de variabele lokaal en statisch in de getInstance methode te declareren, zorgt de compiler er tijdens het compileren voor dat het geheugen van de statische variabele aan het eind van het programma wordt vrijgegeven. Er wordt voor deze methode gekozen, omdat deze methode niet foutgevoelig is, omdat de programmeur op deze manier niks van de implementatie van de BreakPointHandler hoeft te weten. De implementatie ziet er dan als volgt uit.

```
BreakPointHandler* BreakPointHandler::getInstance(){
	static BreakPointHandler brk;
	return &brk;
}
```

# 9 Increment 2 – Extractie SystemC ports en verbindingen tussen tlm sockets

## 9.1 Analyse

In increment 2 wordt de extractie van verbindingen(Bindings) tussen tlm sockets ontwikkeld. Er is tijdens de analysefase gekeken welke verbindingen er mogelijk zijn tussen tlm sockets. In SystemC/TLM zijn er 2 mogelijke verbindingen.

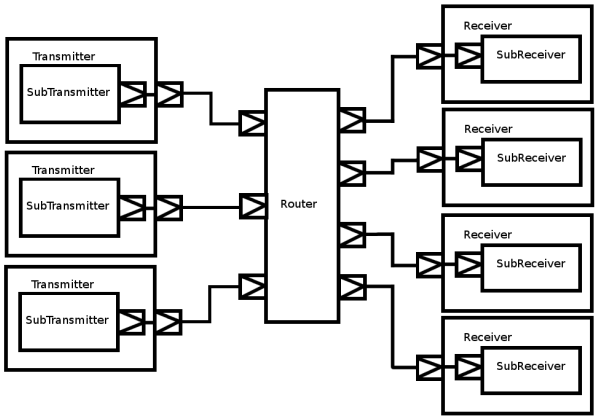
### *Primitieve verbinding*

Primitieve bindingen zijn verbindingen tussen initiator en target socket.

### *Hiërarchische verbinding*

Hiërarchische verbindingen zijn bindingen die worden gemaakt tussen initiator en initiator of target en target socket op een verschillend diepte niveau in de hiërarchie.

In afbeelding 11 is voorbeeld 2 te zien waarin gebruik wordt gemaakt van een hiërarchische verbinding tussen de socket in Receiver en de socket in SubReceiver. Een voorbeeld van een primitieve verbinding is te zien tussen de socket van Transmitter en de socket van Router.




*Afbeelding 11: TLM voorbeeld 2*

In increment 2 worden ook de sc_port en sc_export geëxtraheerd. Deze twee SystemC poorten worden al door SHaBE geëxtraheerd, maar zijn nodig om met tlm_fifo en andere tlm objecten te werken en de verbindingen tussen deze objecten te extraheren. Daarom wordt ervoor gekozen om deze ook de sc_port en sc_export extractie te implementeren in SaTHE. De extractie van een tlm_fifo wordt in het volgende increment geïmplementeerd, zodat er een goede verdeling is qua taken per increment. Alternatief had kunnen zijn om alleen de tlm onderdelen te extraheren, zonder de sc_export en sc_port. Dit zou betekenen dat de sc_port en de sc_export niet worden geëxtraheerd door SaTHE. Dit heeft dan als gevolg dat de verbindingen met bijvoorbeeld tlm_fifo ook niet geëxtraheerd kunnen worden omdat een tlm_fifo alleen kan worden verbonden aan een sc_port.

## 9.2 Ontwerp

Zoals in de analyse hierboven is besproken wordt de extractie van verbindingen tussen sockets en de extractie van onderdelen zoals sc_port en sc_export in dit increment ontwikkeld.
Wanneer er zo abstract mogelijk naar een verbinding wordt gekeken, is een verbinding een koppeling tussen twee adressen van ScObject's. Tussen deze twee adressen is dan een Binding aanwezig, dat betekent dat deze adressen aan elkaar zijn gekoppeld. Bepaalde ScObject's kunnen worden gebonden aan andere ScObject's. Omdat deze ScObject's speciale eigenschappen hebben, wordt er een

extra afgeleide van ScObject genaamd ScBindedObject gemaakt. Een afgeleide van deze klasse kan bindingen aangaan met andere ScObject's.
Voor het maken van de relatie tussen een ScBindedObject en andere objecten wordt er het design pattern Composite gebruikt. Het design pattern is geschikt voor de implementatie van dit stuk software, omdat een ScBindedObject hierdoor afhankelijk wordt van de klasse ScObject en het voor de ScBindedObject object onbekend blijft met welke object van welke afgeleide klasse hij is verbonden. Hierdoor kunnen de ScBindedObject's gemakkelijker aan toekomstige ScObject's worden verbonden, omdat de ScBindedObject afhankelijk is van de super klasse.
Om een Binding te extraheren worden er zogenaamde BindingExtractor's aangemaakt. Omdat Bindings in SystemC en TLM geen afgeleide zijn van een ScObject en niet ieder ScObject bindingen kan aangaan, wordt er in SaTHE voor gekozen, om een nieuwe abstracte Extractor te maken genaamd BindingExtractor. Om een Binding te kunnen extraheren die bijvoorbeeld een Binding tussen een tlm initiator socket en een tlm target socket voorsteld wordt er een TlmInitiatorSocketBindingExtractor klasse ontwikkeld. Deze klasse zorgt voor het extraheren van hiërarchische en primitieve bindings die een tlm initiator socket kan maken. De adressen die een BindingExtractor extraheert kan van ieder mogelijk geëxtraheerd ScObject zijn, waardoor er tijdens het extraheren geen onderscheid gemaakt hoeft te worden tussen het aanmaken van een hiërarchische Binding en tussen een primitieve Binding. Om een Binding te extraheren wordt ervoor gekozen om een breakpoint te zetten in de meest abstracte klasse waarin zich een bind functie bevindt. Dit wordt gedaan, omdat deze functie van deze klasse indirect of direct altijd wordt aangeroepen wanneer er ScObject's worden verbonden. Wanneer er voor wordt gekozen om een afgeleide klasse te gebruiken om breakpoints in te plaatsen, is er een kans dat er Bindings worden gemaakt tijdens de elaboration fase, die niet worden geëxtraheerd. Wanneer deze bind functie wordt aangeroepen kan zowel het adres van het sc_object van het begin van de binding  als aan het eind van de binding worden geëxtraheerd.
Om de geëxtraheerde Bindings aan de geëxtraheerde ScObject's te koppelen wordt er een nieuwe klasse genaamd Binder gemaakt. Een object van deze klasse zorgt voor het verbinden van de juiste ScBindedObject's aan de juiste ScObject's door het doorzoeken van de lijst met alle geëxtraheerde Bindings. De Binder vergelijkt het adres van het begin van de Binding met het adres van alle geëxtraheerde ScBindedObject's en het adres van het eind van de Binding met alle geëxtraheerde ScObject's.

### 9.3 Implementatie

Tijdens de implementatie is de extractie van bindingen tussen ScObject's ontwikkeld. Hieronder een stukje code uit de TLM broncode, waar een Binding tussen een tlm_initiator_socket en tlm_target_socket wordt gemaakt. tlm_base_initiator_socket_b is hier gedeclareerd als super klasse van tlm_initiator_socket.

```
class tlm_base_initiator_socket_b{
…
  // Bind initiator socket to target socket
```

```
  void bind(base_target_socket_type& s){ … }
…
};
```

Wanneer er een breakpoint op de functie bind wordt gezet, dan kan op het moment dat de breakpoint wordt geraakt het adres van het this object en het adres van variabele s worden geëxtraheerd door het adres van de objecten op te vragen. Er kan nu een Binding worden aangemaakt van het adres van this naar het adres van s.

### 9.4 Testen

Omdat in het eerste increment een goed en herbruikbare blackbox test is gemaakt, kan dit programma worden gebruikt om de vernieuwde extractie te testen. Om dit te doen hoeven alleen de nieuwe voorbeelden aan de test toegevoegd te worden en wordt er een nieuw SCMDL bestand gemaakt waarin de te verwachte resultaten staan.
Tijdens het testen is echter wel gebleken dat het steeds meer werk kost om iedere test handmatig uit te voeren. Er wordt daarom gekozen om alle tests uit te voeren in een testomgeving, genaamd UnitTest++. Hierdoor wordt de tijd die er wordt verloren door de tests elke keer handmatig te starten verminderd. Hier tegenover staat wel dat er eenmalig een programma ontwikkeld moet worden waarmee alle tests in een keer uitgevoerd kunnen worden.
Om te controleren of de memoryleak test m.b.v. Valgrind succesvol zijn verlopen moeten er commandline commando's worden uitgevoerd. Om de uitvoer van deze commando's te gebruiken in de test, moeten de resultaten via een pipe terug naar het testprogramma worden geschreven. Vervolgens kan het resultaat worden uitgelezen. Wanneer de test nu meerdere malen snel uitgevoerd kan worden, blijft deze op willekeurig momenten hangen. Wanneer de TLMSocketExtractors uitgezet worden, is het probleem verholpen. Om het probleem op te lossen wordt er gekozen om met de debugger het programma te debuggen, wanneer deze vastloopt.
De fout blijkt te ontstaan, wanneer de functie van GDB wordt aangeroepen waarmee er wordt gewacht totdat de methode volledig is uitgevoerd. Omdat de fout zich op willekeurige momenten voordoet is het erg moeilijk om te onderzoeken waar de fout vandaan komt.
Omdat deze klasse herbruikt is uit SHaBE wordt de fout doorgegeven aan de programmeur van deze klasse(de opdrachtgever) en wordt er geen aandacht meer aan dit probleem besteed. Wanneer het probleem optreedt, wordt het genegeerd.
Wanneer wordt overlegd met de opdrachtgever is er een tijdelijke oplossing beschikbaar. Het probleem zit in het tegelijkertijd uitlezen van data die wordt ontvangen door GDB en input die de gebruiker kan invoeren. Deze input kan bijvoorbeeld nodig zijn om tijdens het uitvoeren van het SystemC model de gebruiker het aantal te simuleren modules te laten bepalen. Omdat er (nog) geen voorbeeld programma's worden gebruikt waarbij invoer door een gebruiker nodig is, wordt er door de opdrachtgever een tijdelijke oplossing aangegeven, waardoor invoer van gebruiker wordt genegeerd.

# 10 Increment 3 – Extractie tlm analyse port, tlm fifo en verbindingen tussen SystemC ports

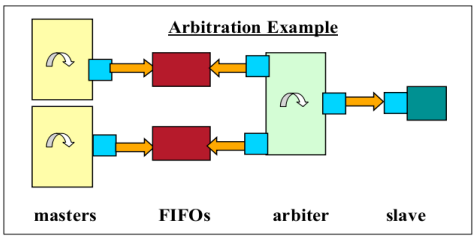
## 10.1 Analyse

In de analysefase van increment 2 is uitgelegd dat sc_port's en sc_export's nodig zijn om de bindingen met tlm_fifo's te maken. Wanneer voorbeeld 3 gebruikt wordt tijdens de extractie door SaTHE is dit te zien.
De tlm_fifo die in voorbeeld 3 wordt gebruikt, wordt niet geëxtraheerd, ook niet als een ander sc_object. Dit komt omdat een tlm_fifo een sc_prim_channel(primary channel) is en sc_prim_channel's worden (nog) niet door SaTHE geëxtraheerd.
Omdat er een kleine kans is dat SHaBE dit voorbeeld wel zou kunnen extraheren, omdat SHaBE sc_port's, sc_export's en primary channels kan extraheren, wordt dit gecontroleerd. Er blijkt dan echter dat SHaBE de tlm_fifo niet kan extraheren. Dit komt waarschijnlijk doordat SHaBE het extraheren van sc_object's die gebruik maken van multiple inherintance niet ondersteund. Omdat SHaBE geen tlm_fifo's kan extraheren, wordt de extractie van tlm_fifo's door SaTHE ontwikkeld.
Het tlm_analysis_port object kan niet door SaTHE worden geëxtraheerd, omdat er na nader onderzoek blijkt dat een tlm_analysis_port een afgeleide is van sc_object en om dit soort objecten te extraheren moet er een nieuwe afgeleide klasse van de klasse ScObjectExtractor worden ontwikkeld.

## 10.2 Ontwerp

In dit increment worden de volgende 3 onderdelen geëxtraheerd:

- sc_port & sc_export bindingen
- tlm_fifo
- tlm_analysis_port



*Afbeelding 12: tlm_fifo en sc_port's voorbeeld*

In afbeelding 12 is te zien dat er tussen de masters en de arbiter een fifo wordt gebruikt. Dit wordt gedaan omdat de arbiter bijvoorbeeld niet op hetzelfde moment een bericht van beide masters kan afhandelen. Door het gebruik van een fifo kan de data gebufferd worden. De fifo kan worden gebruikt door de porten van de masters en de arbiter aan de fifo te binden. Een tlm_analysis_port is niet afgebeeld, maar kan worden vergeleken met een sc_port.

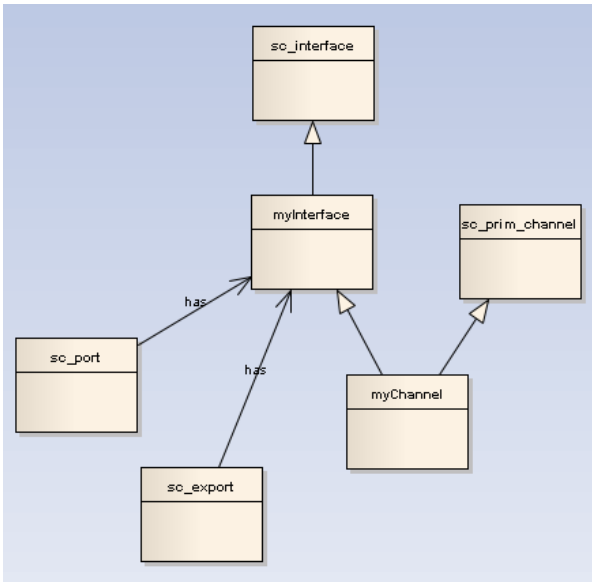
Om de extractie van al deze onderdelen aan SaTHE toe te voegen wordt er een ontwerp gemaakt. Omdat SaTHE in eerste instantie al zo is opgebouwd zodat er makkelijk extra te extraheren ScObject's kunnen worden toegevoegd is het makkelijk om de tlm_fifo en tlm_analysis_port te extraheren. Omdat de tlm_fifo eigenlijk een primary channel is, wordt ervoor gekozen om een nieuwe afgeleide van ScObject te maken, genaamd ScPrimitiveChannel. Een tlm_fifo wordt van deze ScPrimitiveChannel afgeleid. Dit voegt op dit moment nog niet veel waarde toe aan SaTHE, echter wanneer er later nieuwe channels moeten worden geëxtraheerd, kunnen deze worden geextraheerd door een nieuwe afgeleide van ScPrimitiveChannel aan te maken. 1 ding hebben zij namelijk allemaal gemeen en dat is dat zij een koppeling zijn tussen 2 SystemC poorten.

Om een tlm_analysis_port te extraheren wordt er een nieuwe afgeleide van de klasse ScObjectExtractor aangemaakt. Deze afgeleide zorgt voor de extractie van een tlm_analysis_port.
Een binding tussen SystemC poorten extraheren is een veel complexer onderdeel. Een van de problemen die verbindingen tussen SystemC poorten met zich meebrengen is dat SystemC poorten anders aan elkaar worden gekoppeld dan TLM sockets. SystemC poorten bevatten namelijk een interface. Deze interface definieert volgens wat voor functies de SystemC poorten met elkaar kunnen communiceren.
Om te kunnen bepalen hoe de binding precies aangemaakt wordt, is er gekozen om met de debugger het voorbeeld te debuggen zodat er een goed beeld kan worden gevormd bij het proces van het binden van de interfaces met de ports. Hieruit blijkt dat wanneer een sc_port aan een sc_export wordt verbonden, eigenlijk de interface die de sc_export bevat wordt doorgegeven aan de sc_port en de sc_port wordt dan verbonden aan de interface.
Wanneer een sc_port en sc_export met elkaar verbonden zijn, zijn zij aan dezelfde interface gebonden en zijn de SystemC poorten dus verbonden aan hetzelfde interface object met hetzelfde adres.

### 10.3 Implementatie

Tijdens het implementeren van voorbeeld 8 waarin een verbinding wordt gemaakt tussen een sc_port en een sc_export wordt er een sc_prim_channel gebruikt. Dit wordt gedaan, omdat de tlm_fifo die in een volgend voorbeeld in dit increment wordt geëxtraheerd een afgeleide is van een sc_prim_channel. Een sc_prim_channel wordt al door SHaBE geëxtraheerd, maar omdat de tlm_fifo een speciale soort sc_prim_channel is, is het nodig om tijdens de extractie onderscheid te kunnen maken tussen een tlm_fifo en een sc_prim_channel. Er wordt daarom gekozen om ook de sc_prim_channel te extraheren. Een sc_prim_channel is een afgeleide van sc_object, waardoor er voor het extraheren van een sc_prim_channel een nieuwe afgeleide van ScObjectExtractor moet worden aangemaakt. In SaTHE wordt de geëxtraheerde sc_prim_channel als ScPrimitiveChannel opgeslagen.

Tijdens het extraheren van de bindingen tussen de SystemC poorten ontstaat er een nieuw probleem. Een sc_port en een sc_export zijn in SystemC model aan elkaar gekoppeld doordat zij dezelfde interface delen. In voorbeeld 8 wordt de interface genaamd myInterface gebruikt. Voorbeeld 8 is schematisch te zien in afbeelding 13. De sc_port en sc_export worden in voorbeeld 8 met elkaar verbonden, doordat de export is verbonden aan een channel en de sc_export geeft de interface van het channel door aan de sc_port bij het binden, wanneer de export aan de



*Afbeelding 13: Gedeelte van de klassenstructuur voorbeeld 8*

channel is verbonden en de sc_port aan de sc_export wordt verbonden. De channel, myChannel, overerft zowel over van de sc_prim_channel als van de interface, myif. Tijdens de extractie moet worden achterhaald met welk afgeleide klasse van sc_prim_channel de SystemC poorten zijn verbonden, in dit geval myChannel, omdat dit the most derived base klasse is, maar wanneer de bind functie van een sc_port wordt aangeroepen waarmee een koppeling wordt gemaakt tussen een sc_port en een sc_export, wordt er altijd een sc_port aan een interface gekoppeld. Wanneer de sc_port dus wordt gebonden aan de sc_export wordt dit gedaan zoals in onderstaande code. Hierin zijn 2 modules gedefinieerd. Een Start en een End. Een Start module heeft een sc_port en een End module heeft een export. Deze sc_port en sc_export moeten aan elkaar worden verbonden.

```
class myif: virtual public sc_interface
{
public:
    virtual void run() = 0;
};

class mychannel: public sc_prim_channel, public myif
{
public:
    SC_CTOR(mychannel){}
    virtual void run(){}
};

SC_MODULE(End)
{
public:
        sc_export<myif> exportPort;
        SC_CTOR(End):exportPort("MyExport"), chan("channel"){
                exportPort.bind(chan); //Stap 1. Export krijgt eerst een interface van chan!
        }
private:
        mychannel chan;
};

SC_MODULE(Start)
{
public:
        sc_port<myif> portPort;
        SC_CTOR(Start):portPort("portPort"){}
};

SC_MODULE(Top)
{
public:
        Start start;
        End end;

        SC_CTOR(Top):
        start("Start"),
        end("End")
        {
                start.portPort.bind(end.exportPort);//2. Nu wordt de interface van exportPort
aan portPort doorgegeven. En portPort is aan de interface gebonden.
        }
};

int sc_main( int argc , char **argv )
{
        Top top("Top");
```

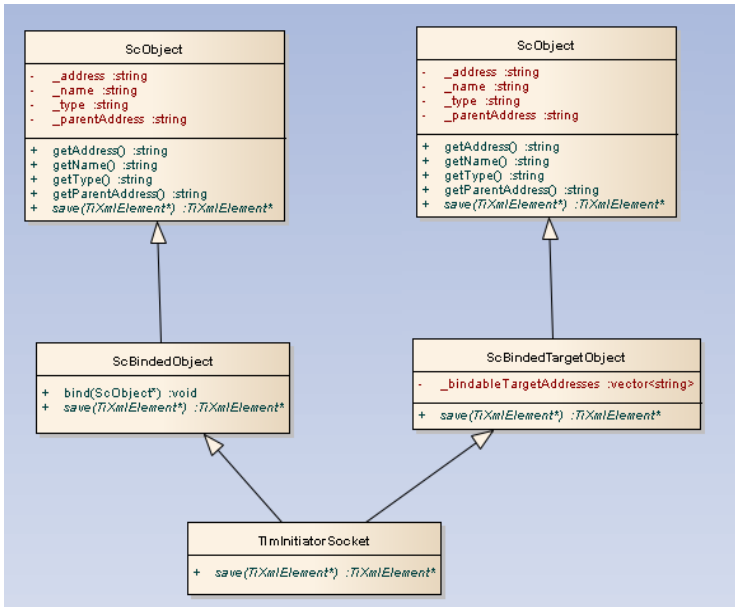
```
        sc_start( 100 , SC_NS );
        return 0;
}
```

Op de regel `exportPort.bind(chan);` wordt de export gebonden aan de channel, omdat de exportPort een interface moet bevatten voordat de sc_port kan worden gebonden aan een sc_export. Maar met de member functie van de sc_port wordt alleen een parameter van het type myif& meegegeven. Het addres van de base klasse myif wordt nu dus gebruikt. Dit betekent dat de sc_export gebonden is aan een interface, myif, en niet aan de channel.
Bij de regel: `start.portPort.bind(end.exportPort);` wordt de sc_port portPort aan de interface, waaraan de exportPort is verbonden, verbonden, waardoor de sc_port en sc_export nu via een interface aan elkaar zijn gekoppeld. Want ook de parameter van het memberfunctie bind van een sc_port is van het type myif&. Wanneer er nu gekeken wordt aan welk adres zowel de sc_port als de sc_export zijn gekoppeld dan is dat het adres van de base klasse myif van de mychannel. Doordat er multiple inheritance wordt gebruikt in de mychannel klasse hoeft het adres van een object van een afgeleide klasse(in dit geval mychannel) niet overeen te komen met het adres van iedere base klasse(myif en sc_prim_channel). Hierdoor is het dus niet mogelijk om tijdens het maken van de verbinding tussen een sc_port en sc_export in SaTHE alleen het adres van de afgeleide klasses te vergelijken met het eindadres van een geëxtraheerde binding.
Dit probleem kan worden omzeild door in de voorbeelden de gebruikte klasse mychannel altijd als eerst te laten overerven van myif, zoals:

```
class mychannel: public myif, public sc_prim_channel {};
```

Wanneer er een mychannel object wordt aangemaakt, dan krijgt dit object altijd hetzelfde adres als dat het myif base klasse object heeft. Omdat niet in de SystemC standaard of TLM standaard is gespecificeerd dat SystemC ontwikkelaars volgens bovenstaande structuur moeten werken, wordt de keuze gemaakt om van iedere SystemC onderdeel dat afgeleid is van een sc_object en een sc_interface alle adressen van alle base klasses te extraheren die afgeleid zijn van sc_interface. Hierdoor worden de adressen van alle interfaces die kunnen worden meegegeven als parameter met de bind functie geëxtraheerd. Door deze adressen op te slaan in het geëxtraheerde ScObject kan tijdens het binden van de geëxtraheerde ScObject's in SaTHE worden gecontroleerd tussen welke ScObject's er een Binding moet worden aangemaakt en of deze binding wordt gemaakt tussen een interface.

Doordat er nu ook bindingen tussen SystemC poorten en interfaces toegestaan moeten worden, moet het programma gedeeltelijk gerefactored worden. De eerder bedachte ScBindedObject klasse kan in deze structuur niet worden gebruikt, omdat dit zou zorgen voor redundante data in een afgeleide klasse van ScObject. Hieronder wordt toegelicht waarom.



*Afbeelding 14: Onderscheid maken tussen ScObject's die bindingen ontvangen of aangaan*

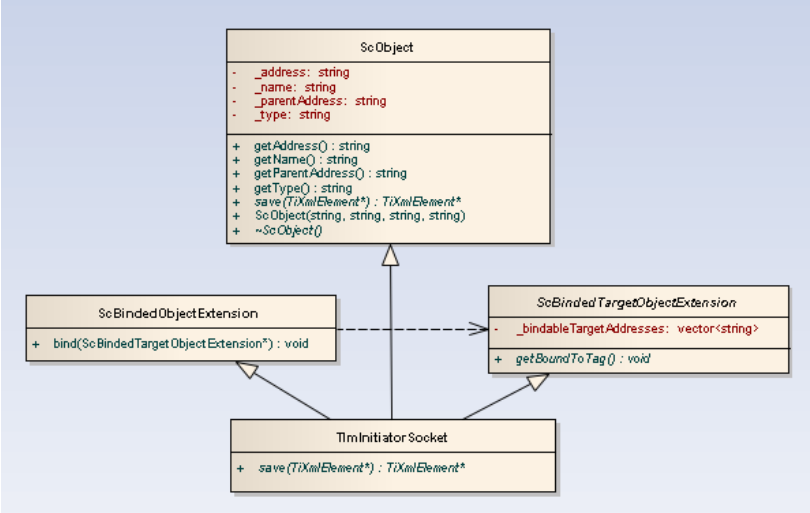
In het nieuwe ontwerp wordt er een zogenaamde ScBindedTargetObject toegevoegd. Een ScBindedTargetObject is een klasse waarin de adressen van de sc_interface's van een sc_object in worden opgeslagen, omdat ScObject's die verbindingen kunnen ontvangen, interfaces kunnen bevatten. Een ScBindedTargetObject is een klasse voor ScObject's die verbindingen kunnen ontvangen. Wanneer een ScObject nu zowel een ScBindedObject als een ScBindedTargetObject is, zoals een TlmInitiatiatorSocket dan ontstaat er een probleem. Zie afbeelding 14. Wanneer de klasse TlmInitiatorSocket wordt gebruikt, dan moet deze zowel verbonden met een ander TlmTargetSocket kunnen worden (bij een primitieve binding) als dat een andere TlmInitiatorSocket verbindt met de TlmInitiatorSocket(bij een hiërarchische binding). Hierdoor moet een TlmInitiatorSocket overerven van zowel een ScBindedObject als een ScBindedTargetObject. Door de multiple inherintance krijgt de TlmInitiatorSocket indirect twee keer een base klasse ScObject , waardoor dus ook tweemaal per TlmInitiatorSocket wordt gespecificeerd wat het address, name en parentAddress van dit ScObject is. Wanneer de getName methode wordt aangeroepen dan is deze ambigious. Dit komt doordat er in een TlmInitiatorSocket 2 keer een getName functie is gespecificeerd vanwege de dubbele base klasses. Om dit probleem op te lossen heeft de afstudeerder verschillende oplossingen bedacht, waarmee het ontwerp aangepast kan worden zodat er onderscheid wordt gemaakt tussen ScObject's die met met andere ScObject's verbinden, ScObject's waarmee andere ScObject's kunnen verbinden, of ScObject's die beide kunnen:

1. TlmInitiatorSocket overerft van een ScObject en twee andere klasses. Deze twee andere klasses bevatten een gedeelte van de implementatie van de huidige ScBindedObject en ScBindedTargetObject en deze klasses overerven beide niet van ScObject. Hierdoor wordt de data die in TlmInitiatorSockt wordt opgeslagen maar in 1 ScObject base object opgeslagen.
2. De volgende klasses worden toegevoegd: een klasse voor ScObject's die bindingen kunnen aangaan met andere ScObject's, een klasse voor ScObject's die verbindingen kan ontvangen en een klasse voor ScObject's die zowel verbindingen kan ontvangen als verbindingen kan aangaan. Door een interface te definiëren voor ScBindedObject's waarin de bind functie

wordt gedefinieerd, wordt een constructie gemaakt waarin geen redundante data wordt opgeslagen.

Om deze opties een beter toelichting te geven worden ze hieronder uitgebreid beschreven.

In het ontwerp van optie 1, in afbeelding 15, is te zien dat ieder te extraheren sc_object wordt afgeleid van een ScObject. De ScBindedObject en ScBindedTargetObject klasses worden omgezet naar de klasses ScBindedObjectExtension en ScBindedTargetObjectExtension, omdat deze naam beter aangeeft, dat zij zelf geen ScObject zijn, maar een uitbreiding zijn voor een afgeleide van ScObject. Wanneer een sc_object bindingen aan kan gaan, erft deze over van ScBindedObjectExtension, wanneer deze bindingen kan ontvangen erft deze over van ScBindedTargetObjectExtension en wanneer deze zowel kan verbinden als verbonden kan worden erft het sc_object over van beide Extention klasses.

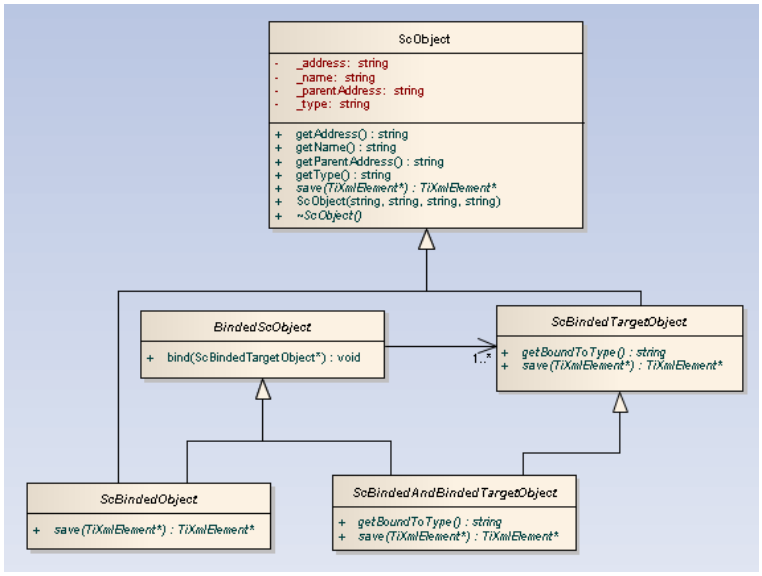



*Afbeelding 15: Ontwerp optie 1*

Er wordt niet voor optie 1 gekozen omdat er op deze manier klasses kunnen worden gedefinieerd door een volgende ontwikkelaar die wel afgeleid kunnen zijn van een ScBindedObjectExtension of ScBindedTargetObjectExtension, en dus wel bindingen kunnen aangaan of ontvangen, maar niet afgeleid kunnen zijn van de ScObject klasse. Deze vrijheid mag niet worden gegeven, omdat een dergelijke constructie nooit voor kan komen. Wanneer een binding wordt opgeslagen in SCMDL dan moet daarin bijvoorbeeld de systemc-name van het sc_object worden gespecificeerd. Alleen afgeleide klasses van ScObject bevatten een systemc-name waardoor het onmogelijk moet zijn een afgeleide van ScBindedObjectExtension of ScBindedTargetObjectExtension aan te maken zonder dat dit een ScObject is.

Om deze reden wordt er voor optie 2 gekozen. Hierin kan de programmeur niet 'per ongeluk' een verkeerd afgeleide maken. Het diagram hiervan is in afbeelding 16 afgebeeld.

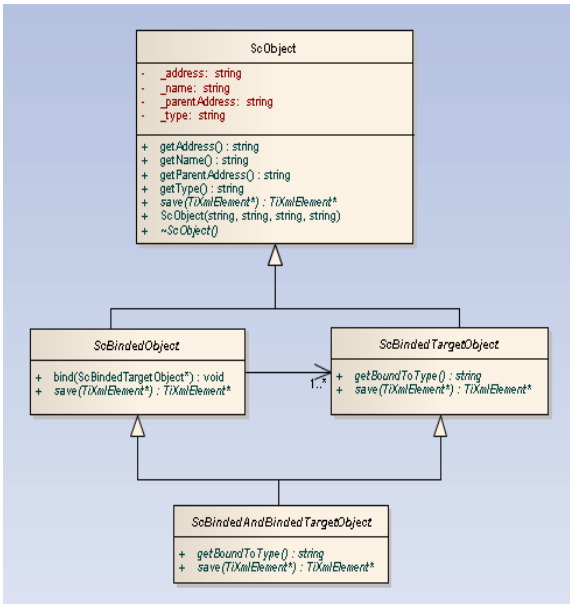
Door de te extraheren ScObject's zoals een TlmInitiatorSocket af te leiden van een ScBindedAndBindedTargetObject




*Afbeelding 16: Gekozen ontwerp*

kan er een ScObject worden gemaakt die zowel verbindingen kan maken als kan ontvangen. Een ScExport kan alleen nog maar bindingen aangaan en kan daardoor worden afgeleid van een ScBindedObject en een ScPrimitiveChannel kan alleen bindingen kan ontvangen, daarom wordt een ScPrimitiveChannel afgeleid van een ScBindedTargetObject. Door de member functie bind alleen een parameter van het type ScBindedTargetObject te laten accepteren wordt er voorkomen dat ScObject's die geen verbindingen kunnen ontvangen kunnen worden verbonden.

Wanneer er overlegd wordt met de opdrachtgever wordt duidelijk dat er nog een mogelijke oplossing is. Deze oplossing maakt gebruik van een “virtual base class”. Omdat het gebruik van een virtual base class onbekend was bij de afstudeerder, is deze optie in eerste instantie niet meegenomen in de lijst met mogelijke oplossingen. Door gebruik te maken van een virtual base class kunnen de relaties die zijn gespecificeerd in afbeelding 14 worden geïmplementeerd zoals afgebeeld, zonder de redundante data uit dezelfde base klasses. Hierdoor kan het ontwerp van optie 2 worden vereenvoudigd naar het ontwerp te zien in afbeelding 17. Een van de aspecten van Agile is om alles zo simpel mogelijk te houden. Door voor het ontwerp te kiezen uit afbeelding 17 wordt het programma eenvoudiger opgebouwd en kan het zijn originele structuur deels behouden. Om de zogenaamd diamond structuur van de structuur te testen en de werking van de virtual base klasses is er een simpel testprogramma gemaakt om te controleren of de virtual base klasses ook werken zoals verwacht wordt. Dit programma is in bijlage D te zien. Doordat dit programma kan worden gecompileerd en slechts eenmaal de uitvoer '42' geeft en alle constructors worden slechts 1 keer aangeroepen wordt aangetoond, dat de virtual base class gebruikt kan worden. Er wordt gekozen om eerst een voorbeeld programma te maken om te kijken hoe de huidige structuur kan worden uitgebreid. De huidige structuur is namelijk van veel andere klasses afhankelijk waardoor er fouten worden ontdekt tijdens het implementeren die misschien helemaal niks met het gebruik van de virtual base klasses te maken hebben. Het gebruik van een virtual base klasse was onbekend, waardoor er verwacht wordt dat er sneller fouten worden gemaakt in de implementatie.



*Afbeelding 17: Virtual base class ontwerp*

In increment 1 is de factory method geïmplementeerd in iedere ScObjectExtractor, zodat er onafhankelijk van de ScObjectExtractor een afgeleide van ScObject kan worden aangemaakt. Tijdens de implementatiefase van increment 3 wordt er besloten om geen gebruik meer te gaan maken van de factory method, omdat deze niet goed toegepast is/kon worden. De bedoeling van de factory methode was dat deze functie aangeroepen wordt vanuit de base klasse en dat in de afgeleide klasses wordt gespecificeerd wat deze methode precies doet. Omdat de methode uiteindelijk niet vanuit een base klasse wordt

aangeroepen, wordt er besloten de createScObject functie te verwijderen. De abstractie die er met de createScObject functie bereikt wilde worden is wel gekregen, maar niet door het gebruik van de createScObject functie. Dit komt omdat iedere afgeleide van ScObjectExtractor alleen ScObject's teruggeeft en niet specificeerd wat de afgeleide klasse van het ScObject is. Hierdoor is de HierarchyObjectExtractor alleen afhankelijk van ScObject en niet van de afgeleide klasses.

Daarnaast is er in increment 1 voor gekozen om alleen het adres, parent sc_object adres en de systemc-name te extraheren, omdat deze nodig zijn om de hiërarchie op te bouwen. Tijdens dit increment wordt ervoor gekozen om ook de in SCMDL opgeslagen name en type te extraheren. Dit wordt gedaan, omdat de adressen van de interfaces van een sc_object zover er bedacht kan worden alleen kunnen worden verkregen door het most-derived klasse object te casten naar objecten van zijn base klasse, omdat de interfaces base klasses van de most derived klasse zijn.  Wanneer er multiple inherintance wordt gebruikt en het adres van de afgeleide klasse wordt vergeleken met het adres van de base klasses dan kunnen deze ongelijk zijn, waardoor het nodig is om het adres van ieder base class object te achterhalen. Wanneer met GDB het 'this' object wordt geprobeerd te casten naar zijn base klasse dan treedt er een GDB fout op waardoor het nodig is om de variabele waarin het object is opgeslagen te casten naar zijn base klasse. Hierdoor is het noodzakelijk om de naam van de variabele te achterhalen waarin het object is opgeslagen. Om de variabelenaam te achterhalen moet het type worden geëxtraheerd. Omdat de variabelenaam en het type ook gespecificeerd worden in het SCMDL formaat worden deze ook gelijk worden opgeslagen. Door de variabele op te slaan in de SCMDL kan deze ook gelijk worden getest in de test fase. Om de variabele te casten naar de base klasse objecten kunnen er verschillende cast methodes worden gebruikt:

1. (C-style) Cast
2. Static cast
3. Dynamic cast
4. Reinterpret cast

De enige cast methode welke geschikt is voor het casten van de variabele is dynamisch casten, omdat er bij een c-style cast een object altijd naar een ander type wordt gecast ofdat het object nu van type is waarnaar het gecast wordt of niet. Bij een reinterpret cast gebeurd precies hetzelfde. Mbv een statische cast wordt er statisch(dus voor het uitvoeren van het programma) gecontroleerd of als er een base klasse pointer naar een derived klasse pointer wordt gecast, ofdat deze derived klasse wel een derived klasse is. De geschikte methode voor het verkrijgen van het adres van een interface klasse is het gebruiken van een dynamic cast, omdat een variabele bij deze methode alleen wordt gecast naar het gewenste type als er een geldig en volledig object kan worden verkregen. Voorbeeld:

```
class A{};
class B: public A{};
class C: public B{};
class D{};
```

In dit geval kunnen zowel objecten van klasse B als C naar een object van de klasse A worden gecast met een dynamic_cast. Een object van klasse D kan echter niet naar een object van klasse A worden gecast.
Wanneer een pointer naar een object niet gecast kan worden, dan wordt het adres 0x0 teruggeven.

Gedurende de implementatiefase wordt duidelijk dat het extraheren van het type en de variabele naam te veel tijd kost. Omdat er aan het eind van dit increment toch een werkend product moet komen, wordt ervoor gekozen om de extractie van de variabelenaam en het type van sc_object's die geen interfaces bevatten uit te stellen.

### 10.4 Testen

Tijdens de testfase van increment 3 zijn er verschillende SCMDL bestanden gemaakt om de geëxtraheerd gegevens mee te vergelijken.
Omdat er besloten is om in increment 3 ook de name en het type welke in het SCMDL gespecificeerd zijn uit een SystemC model te extraheren, worden alle SCMDL test bestanden uit de vorige incrementen aangepast, zodat ook de name en het type in de tests worden gecontroleerd.
Omdat er tijdens de implementatie fase is gekozen om het extraheren van de naam van de variabele en het type door te schuiven naar een volgend increment moet een deel van de tests worden aangepast, zodat de geëxtraheerde variabele naam en type nog niet worden getest.

# 11 Increment 4 – Extractie tlm_req_resp_channel en tlm_transport_channel

## 11.1 Analyse

In increment 4 wordt er onderzocht of de TLM onderdelen tlm_req_resp_channel en tlm_transport_channel kunnen worden geëxtraheerd. Voordat er aan de extractie wordt begonnen, wordt er in de TLM source code gezocht of deze twee onderdelen uit andere TLM of SystemC onderdelen bestaan, zodat er gekeken kan worden of deze twee onderdelen als losstaande onderdelen moeten worden geëxtraheerd. Wanneer er in de TLM source code wordt gekeken zijn beide channels afgeleide van een sc_module en bevat een tlm_req_rsp_channel 2 tlm_fifo's en verschillende sc_export's en een  tlm_transport_channel bevat een tlm_req_resp_channel en verschillende sc_export's. Een van de eigenschappen van een tlm_transport_channel is dat deze channel altijd een maximum buffergrootte heeft van 1. Een tlm_req_rsp_channel heeft een vooraf te definiëren maximum grootte.
Wanneer de voorbeelden 3 en 4 worden gebruikt voor de extractie door SaTHE zijn er een aantal merkwaardige dingen gebeurd tijdens de extractie.
Bij voorbeeld 3 heeft SaTHE namelijk de sc_export's en tlm_fifo's die zich in de tlm_req_rsp_channel bevinden buiten de tlm_req_rsp_channel geplaatst, omdat de parentaddressen waarschijnlijk fout zijn geëxtraheerd.  tlm_req_rsp_channel wordt als module geëxtraheerd en dit klopt ook, want een tlm_req_rsp_channel is een afgeleide van sc_module. Wanneer voorbeeld 4 wordt geëxtraheerd, treedt er een GDB fout op genaamd “`virtual baseclass botch`”. Omdat beide channels niet goed geëxtraheerd kunnen worden, wordt SaTHE uitgebreid om ook deze channels te extraheren.

## 11.2 Ontwerp

Zowel de tlm_transport_channel als de tlm_req_resp_channel zijn channels. In increment 3 is er al voor gekozen om de channel tlm_fifo te extraheren als afgeleide van ScPrimitiveChannel. Dit werd gedaan om onderscheid te kunnen maken tussen channels en increment sc_object's. Wanneer de implementatie van een tlm_req_rsp_channel en een tlm_transport_channel wordt bekeken, dan blijkt dat zij geïmplementeerd zijn als afgeleide van sc_module. Om te kunnen verbinden met een van de channels, moet er worden verbonden met een sc_export uit de channel. Wanneer normale primitieve channels worden gebruikt, dan moet er met het primitieve channel object worden verbonden of met een interface van het object en niet met een object uit het primitieve channel object. Bij het gebruik van tlm_transport_channel en tlm_req_resp_channel wordt er juist verbonden met een van de sc_export's(ScObject's in de channel). Omdat een tlm_transport_channel en tlm_req_resp_channel op deze manier werken, wordt ervoor gekozen om de channels niet als afgeleide van ScPrimitiveChannel te extraheren, maar als nieuwe afgeleide van ScObject.

## 11.3 Implementatie

In het vorige increment heeft zich een probleem voortgedaan waardoor er voortijdig gestopt moest worden met het extraheren van de naam van de

variabele waarin het sc_object wordt opgeslagen en het type van het sc_object. Tijdens increment 4 wordt er verder gegaan met het extraheren van de naam van de variabele. Het probleem is namelijk dat de naam van de variabele waarin een afgeleide van bijvoorbeeld sc_module wordt opgeslagen niet altijd kan worden geëxtraheerd op het moment dat het breakpoint in de constructor van sc_module geraakt wordt. Hieronder een voorbeeld:

```
SC_MODULE(Top){
public:
        SC_CTOR(Top){}
};

int sc_main( int argc , char **argv ){
        Top* top = new Top("TopNAME");
        Top* top2 = new Top("TopNAME2");
        sc_start();
        return 0;
}
```

Het probleem hierbij is dat wanneer het breakpoint wordt geraakt in de constructor van sc_module wanneer er een Top wordt aangemaakt, dat op dat moment nog niet bekend is wat de waarde van top of top2 is, dus welk adres er aan de pointer is toegewezen. Hierdoor valt er op het moment dat de breakpoint is geraakt niet af te leiden welk ScObject er in welke variabele opgeslagen is.
In SHaBE is er een mechanisme geïmplementeerd om te achterhalen wanneer er een pointer van een mogelijke variabele van waarde is veranderd. Vervolgens kan er worden gecontroleerd ofdat het nieuwe adres overeenkomt met een van de eerder geëxtraheerd 'this' pointer adressen. Als deze overeenkomen is, dan de variabele waarin het ScObject is opgeslagen gevonden.
In eerste instantie is er gekozen om een gelijksoortig mechanisme in SaTHE te implementeren. Wanneer er een ScObject wordt geëxtraheerd, dan wordt er een GDB variabele op het this object van het ScObject gezet om te kijken wat het adres is. Daarnaast wordt er ook een GDB variabele op ieder mogelijke variabele in het parent ScObject gezet. In het parent ScObject zijn namelijk alle variabele van alle mogelijke ScObject's gedefinieerd(met uitzondering van bijvoorbeeld lokale variabele die in sc_main zijn aangemaakt). Vervolgens kan het SystemC model tot het eind van de elaboration fase worden gerund om vervolgens te kijken of het adres van de GDB variabele inmiddels zijn veranderd en of een van de variabele nu de waarde van het adres van een van de eerder geëxtraheerde this adressen bevat. Er moet tot eind van de elaboration fase worden gerund, omdat tot die tijd het adres van het ScObject nog in een variabele opgeslagen kan worden.
Wanneer er een GDB variabele op het parent ScObject wordt gezet ipv op de membervariabele van het parent ScObject en het SystemC model wordt tot het eind van de elaboration fase gerunt, dan blijken de adressen van de member variabele in het parent object niet meer opvraagbaar te zijn met GDB!
Daarom moet er wel op ieder mogelijke variabele een GDB variabele worden gezet. Omdat deze aanpak niet snel te implementeren blijkt, omdat er dan ook weer gecontroleerd moet worden op objecten die worden opgeslagen in variabele van base klasse pointers, wordt er met de opdrachtgever overlegd.

Nadat er met de opdrachtgever is overlegd, wordt er besloten niet opnieuw een aanpak te implementeren, omdat de opdrachtgever een hogere prioriteit geeft aan het extraheren van de simple_sockets en het onderscheid maken tussen simple sockets en tlm sockets, waarvan de extractie in het volgende increment worden ontwikkeld. Er wordt daarom gestopt met het oplossen van dit probleem, omdat er zoals gezegd in SHaBE al een mechanisme is geïmplementeerd waarmee pointers geëxtraheerd kunnen worden en er al aangetoond is dat pointers ook kunnen worden geëxtraheerd, als het parent ScObject en het adres van het ScObject bekend zijn. Daarom wordt er op dit moment besloten dat alleen variabele die statisch zijn aangemaakt geëxtraheerd worden.

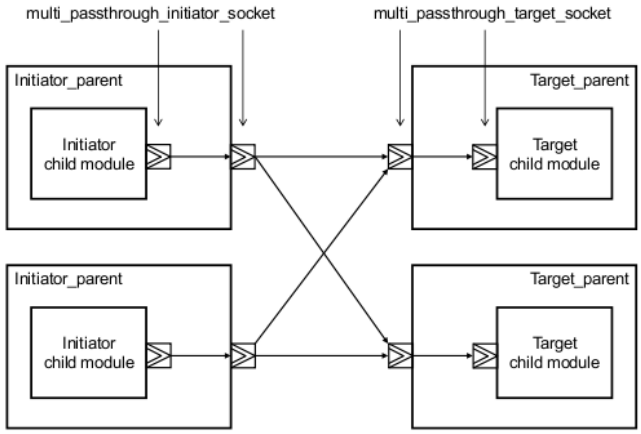
### 11.4 Testen

Tijdens de testfase zijn verschillende voorbeelden aangepast, omdat het extraheren van een pointer niet met zekerheid kan worden gedaan. Wanneer er maar 1 membervariabele in het parent ScObject is te vinden, dan moet dit wel het ScObject zijn wat geëxtraheerd wordt en wanneer het geen base klasse pointer is naar een object van een afgeleide klasse, kan er aan de hand van het type worden bepaald of het de juiste C++ variabele is die wordt geëxtraheerd. Daarom worden de voorbeelden zo aangepast, zodat er zo min mogelijk met pointers wordt gewerkt.

# 12 Increment 5 – Extractie multi passthrough sockets en simple sockets

## 12.1 Analyse

In increment 5 worden de tlm multi passthrough sockets geëxtraheerd. Een tlm multi passthrough socket kan zoals de naam al aangeeft meerdere verbindingen aangaan of ontvangen. Een voorbeeld van tlm multi passthrough sockets is te zien in de schematische weergave van voorbeeld 6 in afbeelding 18.




*Afbeelding 18: Schematische weergave voorbeeld 6*

Daarnaast heeft de opdrachtgever liever dat de simple_sockets, die in increment 1 en 2 gebruikt zijn, worden geëxtraheerd als simple_sockets en niet als tlm_sockets.
Tlm sockets zijn sockets die gedefinieerd zijn in TLM om er zelf gedrag aan toe te voegen. De simple sockets zijn in TLM aanwezig omdat er in deze sockets al standaard gedrag is toegevoegd. Deze zijn dus eenvoudiger om te gebruiken, tenminste dat wordt gesuggereerd door de naam. Aan het begin van het onderzoek werd gedacht dat alle sockets in 2 groepen, tlm sockets en multi sockets, waren te verdelen. Na overleg met de opdrachtgever wenst hij dat er wel onderscheid tussen iedere sockets moet worden gemaakt. Er wordt daarom besloten om naast de multi sockets ook de volgende sockets apart te extraheren van de tlm sockets:

- simple_initiator_socket
- simple_target_socket
- simple_initiator_socket_tagged
- simple_target_socket_tagged
- passthrough_target_socket
- passthrough_target_socket_tagged
- multi_passthrough_initiator_socket
- multi_passthrough_target_socket

Deze worden dan niet als TlmInitiatorSocket of TlmTargetSocket geëxtraheerd, maar krijgen ieder een eigen afgeleide klasse van ScObject, vanwege de verschillende eigenschappen van iedere sockets. Tijdens de analyse fase wordt gedacht dat de extractie van de bovenstaande sockets binnen de deadline van het laatste increment kunnen worden ontwikkeld, omdat SaTHE zo is opgebouwd dat er gemakkelijk extra te extraheren ScObject's kunnen worden toegevoegd.

## 12.2 Ontwerp

Op pagina 127 van de TLM standaard staat beschreven welke bindingen er tussen deze sockets gemaakt kunnen worden. Deze bindingen moeten daarom ook worden geëxtraheerd. Daarom worden er extra voorbeelden gemaakt waarin 2 modules worden gebruikt die met elkaar worden verbonden door het gebruik van alle bovenstaande sockets. In dit increment worden voorbeeld 7, 9, 10 en 11 gebruikt voor de extractie.

| To / From | tlm-init | simple-init | multi-init | | tlm-targ | simple-targ | multi-targ |
|---|---|---|---|---|---|---|---|
| tlm-init | 1 | | | | 1 | 1 | N:1 |
| simple-init | 1 | | | | 1 | 1 | N:1 |
| multi-init | | | 1 | | 1:M | 1:M | N:M |
| | | | | | | | |
| tlm-targ | 1* | 1* | | | 1 | 1 | |
| simple-targ | 1* | 1* | | | | | |
| multi-targ | | | | | | | 1 |

The above table is organized into four quarters as follows:

| | |
|---|---|
| Hierarchical child-to-parent binding | Initiator-to-target binding |
| Reverse binding operators | Hierarchical parent-to-child binding |

| Key | |
|---|---|
| tlm-init | tlm_initiator_socket |
| simple-init | simple_initiator_socket or passthrough_initiator_socket |
| multi-init | multi_passthrough_initiator_socket |
| tlm-targ | tlm_target_socket |
| simple-targ | simple_target_socket or simple_target_socket_tagged or passthrough_target_socket or passthrough_target_socket_tagged |
| multi-targ | multi_passthrough_target_socket |
| 1* | The binding is from inititiator to target, despite the method call being in the direction target.bind(initiator) |

*Afbeelding 19: Mogelijke bindingen van sockets uit de TLM standaard*

Voor iedere te extraheren socket wordt een afgeleide klasse van ScObjectExtractor en een afgeleide klasse van ScObject aangemaakt. Omdat de verschillende sockets gebruik maken van de bind functie uit een base klasse van de socket(zo is de bind functie van een simple_initiator_socket gelijk aan de bind functie van een tlm_initiator_socket omdat simple_initiator_socket een afgeleide is van tlm_initiator_socket en deze erft de bind functie over van de simple_initiator_socket), kan dezelfde bind functie gebruikt kan worden om de binding uit te extraheren.

Helaas is de TLM standaard niet volledig. Er vallen een paar aspecten niet uit de tabel uit de TLM standaard, zie afbeelding 19, af te leiden. Dat zijn:

- Wat een passthrough_initiator_socket is
- Met welke sockets een simple_initiator_socket_tagged bindingen kan aangaan of van welke sockets een simple_initiator_socket_tagged bindingen kan ontvangen.

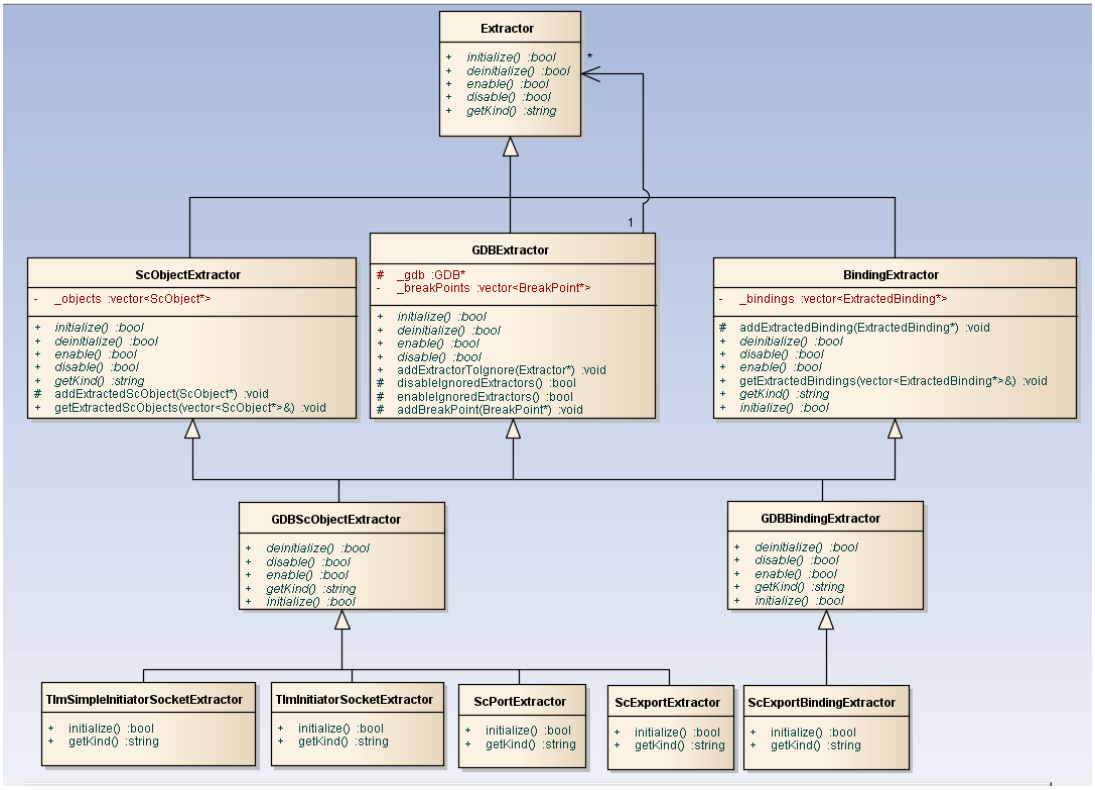
Om toch verder te kunnen gaan met het onderzoek, wordt er overlegd met de opdrachtgever. Na overleg met de opdrachtgever wordt duidelijk dat een passthrough_initiator_socket helemaal niet bestaat. De term passthrough_initiator_socket moet worden vervangen door de simple_initiator_socket_tagged. Hierdoor wordt duidelijk dat een simple_initiator_socket_tagged onder de groep 'simple-init' valt.

In de vorige incrementen is er gebruik gemaakt van een ScBindedObject, een ScBindedTargetObject en een ScBindedAndBindedTargetObject om onderscheid te maken tussen ScObject's die bindingen aan kunnen gaan, bindingen kunnen ontvangen en beide. Uit bovenstaande tabel kan worden beredeneerd of een te extraheren socket van een ScBindedObject, een ScBindedTargetObject of een ScBindedAndBindedTargetObject moet worden afgeleid.

In de TLM standaard maken sockets gebruik van een callback mechanisme, wanneer er namelijk een transaction vanuit socket A naar socket B wordt gestuurd, dan moet socket B hierop reageren. De transaction die socket B heeft ontvangen moet op de een of andere manier worden afgehandeld. Dit wordt in de simple_sockets gedaan doordat er een proces in de sockets is gedefinieerd. Zodra er een transaction wordt ontvangen, wordt dit in het proces afgehandeld. Dit afhandelen vindt plaats door het aanroepen van bepaalde callback functies. Deze callback functies zijn verkregen door over te erven van een tlm interface. Een tlm socket erft niet over van deze interface, maar het afhandelen van een transaction kan alleen worden gedaan in een callback functie. Daarom moet de gebruiker zelf een klasse van de tlm interface afleiden. In dit object moeten dan de callback functies worden geïmplementeerd. Vervolgens moet de socket aan een object van de klasse die de interface implementeerd worden verbonden. Een voorbeeld hiervan is te zien in het volgende hoofdstuk *Implementatie*.
Omdat er nu tijdens de extractie onderscheid gemaakt moet worden tussen een simple socket en een tlm socket, wordt er met de opdrachtgever besloten om een extra voorbeeld te maken. In dit voorbeeld wordt gebruik wordt gemaakt van 2 modules met ieder een tlm socket. De callback functies van de socket worden nu in de modules geïmplementeerd. Dit voorbeeld ziet er dan schematische hetzelfde uit als voorbeeld 1, zie afbeelding 9.

Een simple socket heeft zoals beschreven een proces die de interface implementeert, waardoor het niet nodig is om de simple socket nog van een interface te laten overerven. Echter intern wordt de simple socket wel verbonden aan het proces, dit binden gebeurd namelijk in de constructor van de simple socket.  Omdat er tijdens het uitvoeren van de constructor nu verschillende breakpoints geraakt worden en er eigenlijk breakpoints tijdelijk moeten worden uitgeschakeld, wordt er in dit increment een mechanisme ontwikkeld waarmee dit kan worden gedaan. Hierdoor wordt het niet alleen mogelijk om de bindingen die simple sockets intern maken niet te extraheren, maar daardoor kan er ook voor worden gezorgd dat als er bijvoorbeeld een simple socket geëxtraheerd wordt, dat dan niet alle breakpoints tijdelijk uitgeschakeld hoeven te worden maar alleen de breakpoints van de tlm initiator socket en sc_port. Dit omdat een simple initiator socket een afgeleide klasse is van tlm initiator socket en tlm initiator socket weer een afgeleide klasse is van sc_port.
In het model in afbeelding 20 is te zien dat er verschillende Extractors zijn. Door gebruik te maken van de multiple inherintance zoals in afbeelding 20 is te zien, kan er voor worden gezorgd dat de deinitialize, enable en disable functie alleen door de GDBExtractor klasse worden geïmplementeerd. Door het gebruik van een GDBExtractor klasse wordt er voorkomen dat zowel in de ScBindingExtractor als in de ScObjectExtractor dezelfde implementatie wordt gebruikt voor het in- en uitschakelen van Extractors. In SaTHE worden alle Extractors die gebruikt worden afgeleide van GDBScObjectExtractor en GDBBindingExtractor. Mocht er ooit een andere debugger worden gebruikt i.p.v. GDB, dan kunnen de klasses ScObjectExtractor en BindingExtractor blijven zoals deze zijn.
Om een bepaalde Extractor uit te schakelen kan de memberfunctie disable worden aangeroepen. Wanneer de memberfunctie disable van een afgeleide van GDBExtractor wordt aangeroepen wordt ook de memberfunctie disable van alle

Extractors uit de lijst van GDBExtractor aangeroepen, omdat deze ook moeten worden uitgeschakeld. Door het gebruik van het Composite Design pattern kan dit mechanisme worden ontwikkeld, zonder dat GDBExtractor afhankelijk is van de implementatie van de memberfunctie disable van de afgeleide klasses.




*Afbeelding 20: Uitschakelmechanisme in de klassenhierarchie van de afgeleide van Extractor*

## 12.3 Implementatie

In increment 2 is ervoor gekozen om alleen de bindingen die een tlm socket met andere tlm sockets kan maken te extraheren. Een tlm socket kan namelijk ook een binding aangaan met een interface. Dit wordt gedaan door de socket te binden aan een object van een klasse die overerft van de benodigde interface. In SHaBE wordt dit een hiërarchische channel genoemd. Hieronder een stuk van voorbeeld 12 waarin er een hiërarchische channel wordt gebruikt.

```
using namespace tlm;
using namespace sc_core;
class RAM: public tlm_fw_transport_if<>, public sc_module{
public:
	tlm_target_socket<> socket;
	SC_CTOR(RAM): socket("RAMSocketNAME"){
		socket.bind(*this);
	}
	virtual ~RAM();

	void b_transport(tlm_generic_payload& trans, sc_time& t);
	tlm_sync_enum nb_transport_fw(tlm_generic_payload& trans, tlm_phase& p, sc_time& t);
	bool get_direct_mem_ptr(tlm_generic_payload& trans, tlm_dmi&  dmi_data);
	unsigned int transport_dbg(tlm_generic_payload& trans);
};
```

Op de regel `socket.bind(*this);` is te zien dat een tlm socket een binding aangaat met een object van een klasse die is afgeleid van een tlm interface en van een

sc_object. Omdat het object waarmee verbonden wordt overerft van de juiste interface, wordt dit gezien als de implementatie van de callback functies van de tlm socket. De klasse RAM zorgt dan ook voor het afhandelen van de transactie.

Tijdens de implementatie van de extractie van van de verschillende voorbeelden zijn er verschillende problemen ontstaan. Hieronder een overzicht van de problemen.

Zoals te zien in het stuk code hierboven erft de klasse RAM over van 2 klasses, tlm_fw_transport_if en sc_module.

```
class RAM: public tlm_fw_transport_if<>, public sc_module
```

Wanneer RAM volgens bovenstaande regel wordt afgeleid van deze klasses ontstaat er geen probleem, want wanneer de constructor van RAM wordt aangeroepen wordt eerst de base klasse interface geïnitialiseerd en daarna wordt de base klasse sc_module geïnitialiseerd. Wanneer er een sc_module wordt aangemaakt, wordt er een breakpoint geraakt, omdat dan de constructor van sc_module wordt aangeroepen. Wanneer de RAM module wordt geëxtraheerd wordt er gekeken wat de most-derived klasse is en in welke variabele of membervariabele deze is opgeslagen.
Vervolgens wordt er gekeken of RAM is afgeleid van een afgeleide van sc_interface en wordt de variabele gecast naar alle base klasses van RAM die een afgeleid van sc_interface zijn. Zodra RAM overerft van een `public virtual tlm_fw_transport_if<>` dan blijkt dat het opvragen van alle base klasses van tlm_fw_transport_if niet meer kan en GDB geeft dan de volgende foutmelding “`virtual baseclass botch`”. Hierdoor is het onmogelijk geworden om de interfaces van de module te extraheren, omdat er niet meer achterhaald kan worden of tlm_fw_transport_if in dit geval een afgeleide van sc_interface is, waardoor het hiërarchische channel niet altijd kan worden geëxtraheerd.

Een ander probleem is dat wanneer RAM overerft van de tlm interface en de sc_module dat het overerven in deze volgorde moet `class RAM: public tlm_fw_transport_if<>, public sc_module`. Zodra de interface en sc_module worden omgedraaid en de tlm interfaces worden geextraheerd, dan wordt er ook een foutmelding “`virtual baseclass botch`” door GDB teruggegeven. Hoe dit kan is onduidelijk, omdat er nu helemaal geen virtuele klasses gebruikt worden in het voorbeeld, alleen in de TLM source code. Omdat de foutmelding nu al 2 keer is voorgekomen en het een redelijk kritische fout is voor SaTHE wordt er contact opgenomen met het GDB ontwikkelteam om de GDB fout te proberen op te lossen.

### 12.4 Testen

Zoals in de implementatie fase is beschreven kunnen de interfaces van een ScObject dat gebruik maakt van een virtual base klasse niet worden geëxtraheerd. Tijdens de test fase wordt dit nogmaals duidelijk. De tests waarbij er gebruik wordt gemaakt van een virtual base klasse slagen niet. De voorbeelden worden daarom aangepast, zodat de base klasses in een andere volgorde staan en er geen virtuele overerving wordt gebruikt.

# 13 Conclusie TLM onderzoek

Uit het onderzoek naar de extractie van TLM kan worden geconcludeerd dat TLM kan worden geëxtraheerd uit een SystemC model wanneer er gebruik wordt gemaakt van de dynamische aanpak m.b.v. GDB. Aan de extractie van TLM door SaTHE zit op dit moment wel een beperking, omdat GDB op dit moment de virtuele base klasses van een C++ klasse nog niet altijd kan extraheren. Hierdoor kunnen bijvoorbeeld de hiërarchische channels tussen een tlm socket en een sc_module niet in alle gevallen worden geëxtraheerd.

# 14 Conclusie

De afstudeerder vind dat de opdracht erg uitdagend was. Het ging om twee onderzoeken en bij een onderzoek is het altijd onverwachts wat het resultaat van het onderzoek gaat worden, anders hoeft er geen onderzoek te worden gedaan. Tijdens het onderzoek stuit je op allerlei problemen. Het was de taak van de afstudeerder om goed met deze problemen om te gaan en de juiste beslissingen te nemen. Soms bleken deze beslissingen onjuist en moest er achteraf bijvoorbeeld een stuk code van het proof of concept worden gewijzigd.
Uit eindelijk is het gelukt om beide onderzoeken succesvol af te ronden. Aan de twee proof of concept's die ontwikkeld zijn, zitten wel bepaalde beperkingen. Deze beperkingen zijn ontstaan, doordat er programma's van derde in de proof of concept's zijn gebruikt die bepaalde functionaliteiten van de proof of concept's (nog) niet volledig ondersteunen.

# 15 Evaluatie

In dit hoofdstuk worden de punten toegelicht de afstudeerder juist goed of slecht vond gaan tijdens de afstudeerstage. Tevens wordt er teruggekeken op het verloop van het proces.

## 15.1 Proces

Voorafgaand aan de afstudeerstage is er gekozen om volgens de Agile ontwikkelmethode XP(eXtreme Programming) te werken. Een van de aspecten van XP die gekozen is om uit te voeren is The Planning Game. Gedurende het project is deze Planning Game geen enkele keer uitgevoerd. Dit is vooral gekomen omdat de eisen voor de visualisatie van een SystemC model eenmalig zijn vastgesteld en de eisen voor de extractie van TLM onderdelen uit een SystemC model zijn afgeleid uit de resultaten van het onderzoek naar TLM. Uiteindelijk is er wel gebruik gemaakt van een Agile methode, maar niet helemaal van de ontwikkelmethode XP. Het is uiteindelijk meer een combinatie geworden van TDD(Test Driven Development), XP en FDD(Feature Driven Development). Omdat er tijdens de ontwikkeling van de TLM extractie vooral gefocust werd op het ontwikkelen van de te extraheren onderdelen. Dit is gedaan door vooraf tests te definiëren waarmee de feature getest kan worden.

## 15.2 Producten

Een van de meest positieve punten is dat de opdracht erg uitdagend was, vooral omdat de opdracht zich bevond op redelijk onbekend gebied. Er was voorafgaand aan de afstudeeropdracht al een beetje kennis van SystemC opgedaan in een minor, maar tijdens de afstudeerstage is de werking van SystemC en TLM duidelijker geworden.
De afstudeerder is vooral trots op de manier waarop SaTHE is opgebouwd. Het product is naar zijn mening goed en overzichtelijk opgebouwd. Daarbij is er zoveel mogelijk rekening gehouden met de uitbreidbaarheid van het programma.

# 16 Competenties

In dit hoofdstuk worden de competenties beschreven die tijdens het uitvoeren van de afstudeeropdracht zijn aangetoond.

### *A3 Achterhalen van behoeften van belanghebbende*

De behoeften van de belanghebbende zijn achterhaald door een gesprek met de opdrachtgever te voeren. Uit dit gesprek zijn de eisen vastgesteld. Om zeker te zijn over de juistheid van de eisen is de lijst met eisen door de opdrachtgever goedgekeurd.

### *A4 Kiezen van een ontwikkelstrategie en ontwikkelmethodiek*

Het aantonen van deze competentie is gedaan door het onderbouwen van de keuze van de gekozen ontwikkelstrategie in het afstudeerverslag.

### *C8 Ontwerpen van een technisch informatie systeem*

Deze competentie wordt aangetoond door het toelichten van de bepaalde keuzes die zijn gemaakt tijdens het ontwikkelen van de architectuur van SaTHE. Bij iedere belangrijke uitbreiding van de software is beargumenteerd waarom dit nodig was en wat de alternatieven waren.

### *D16 Het ontwerpen van software*

Deze competenties wordt aangetoond door het gebruik van juist commentaar in de broncode en het gebruik van de rijkheid van C++. Zo is er gebruik gemaakt van functors, OO ontwikkelen, template klasses, multiple inherintance en dynamic casten.

### *D17 Testen van software systemen*

Door het kiezen van een geschikte teststrategie en door het uitvoeren van deze strategie wordt deze competentie aangetoond.

# 17 Bronnen

## 17.1 Literatuur

“Design Patterns” – Erich Gamma, Richard Helm, Ralph Johnson en John Vlissides
“Beginning XML” – David Hunter, Jeff Rafter, Joe Fawcett, Eric van der Vlist, Danny Ayers, Jon Duckett, Andrew Watt en Linda McKinnon
“SystemC: From the Ground Up” – David C. Black, Jack Donovan, Bill Bunton en Anna Keist
“Management informatiesystemen” – G.B. Davis en M.H. Olson

## 17.2 Websites

SystemC reference en standaard
http://www.systemc.org/home/

BreakPoint probleem met templates
http://tdistler.com/2008/11/13/debugging-c-templates-brekpoints-and-gdb

SCMDL definities
http://shabe.sourceforge.net/

SystemC & TLM examples
http://www.doulos.com/knowhow/systemc/tlm2/
http://ens.ewi.tudelft.nl/Education/courses/et4351/SystemC-TLM.pdf

TinyXML
http://www.grinninglizard.com/tinyxmldocs/tutorial0.html

XSLT
http://www.w3schools.com/xsl/

Virtual base class
http://www.deitel.com/articles/cplusplus_tutorials/20060225/virtualBaseClass/

Templates
http://www.cplusplus.com/doc/tutorial/templates/

Casting
http://www.cplusplus.com/doc/tutorial/typecasting/

# 18 Bijlagen

Bijlage A Onderzoek Tekenpakket - Onderzoek Tekenpakket.pdf

Bijlage B Onderzoek TLM extractie - Onderzoek TLM.pdf

Bijlage C SCMDL bij de invoer van N =1

```
<?xml version="1.0" encoding="UTF-8" ?>
<systemc-model xmlns="http://shabe.sourceforge.net/systemc-model"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://shabe.sourceforge.net/systemc-model http://shabe.sourceforge.net/systemc-
model/systemc-model.xsd"
name="/home/dennis/Bureaublad/workspace/TLMExample01/Debug/TLMExample01">
   <hierarchy>
      <primitive-channel name="in[0]" type="sc_core::sc_signal&lt;bool&gt;" systemc-name="signal_0"
systemc-type="sc_signal" address="0xbffff6ec" />
      <primitive-channel name="in[1]" type="sc_core::sc_signal&lt;bool&gt;" systemc-name="signal_1"
systemc-type="sc_signal" address="0xbffff738" />
      <primitive-channel name="out" type="sc_core::sc_signal&lt;bool&gt;" systemc-name="signal_2" systemc-
type="sc_signal" address="0xbffff784" />
      <module name="tb" type="TB&lt;1u&gt;" systemc-name="tb" systemc-type="sc_module"
address="0xbffff4d8">
         <port name="stimulus[0]" type="sc_core::sc_out&lt;bool&gt;" systemc-name="tb.port_0" systemc-
type="sc_out" address="0xbffff534">
            <bound-to to="primitive-channel" name="in[0]" systemc-name="signal_0" />
         </port>
         <port name="stimulus[1]" type="sc_core::sc_out&lt;bool&gt;" systemc-name="tb.port_1" systemc-
type="sc_out" address="0xbffff570">
            <bound-to to="primitive-channel" name="in[1]" systemc-name="signal_1" />
         </port>
         <port name="response" type="sc_core::sc_in&lt;bool&gt;" systemc-name="tb.port_2" systemc-
type="sc_in" address="0xbffff5ac">
            <bound-to to="primitive-channel" name="out" systemc-name="signal_2" />
         </port>
      </module>
      <module name="dut" type="And2_N&lt;1u&gt;" systemc-name="dut" systemc-type="sc_module"
address="0xbffff5e4">
         <port name="in[0]" type="sc_core::sc_in&lt;bool&gt;" systemc-name="dut.port_0" systemc-
type="sc_in" address="0xbffff640">
            <bound-to to="primitive-channel" name="in[0]" systemc-name="signal_0" />
         </port>
         <port name="in[1]" type="sc_core::sc_in&lt;bool&gt;" systemc-name="dut.port_1" systemc-
type="sc_in" address="0xbffff678">
            <bound-to to="primitive-channel" name="in[1]" systemc-name="signal_1" />
         </port>
         <port name="out" type="sc_core::sc_out&lt;bool&gt;" systemc-name="dut.port_2" systemc-
type="sc_out" address="0xbffff6b0">
            <bound-to to="primitive-channel" name="out" systemc-name="signal_2" />
         </port>
      </module>
   </hierarchy>
   <behavior />
</systemc-model>
```

Bijlage D Test virtual base class(Diamond structuur)

```
class A{
public:
    A(int p): _i(p){ cout << "calling A(int)" << endl; }
    int getMyInt(){ return this->_i; }
private:
    int _i;
};

class B: virtual public A{
public:
    B(int i): A(i){ cout << "calling B(int)" << endl; }
};

class C: virtual public A{
public:
    C(int i): A(i){ cout << "calling C(int)" << endl; }
};

class D: public B, public C{
public:
    D(int i): A(i), B(i),  C(i){ cout << "calling D(int)" << endl; }
};

int main(){
    D d(42);
    cout << d.getMyInt() << endl;
    return 0;
}
```