

Was SIMULA overbodig?

Het ontstaan van het objectgeoriënteerde
programmeren als bijproduct van de opkomst van
computersimulatie

H.T. de Beer

H.T.d.Beer@student.tue.nl

0491874

Eindhoven, 6 oktober 2004

Inhoudsopgave

0	Inleiding	2
0.0	Opzet van het document	3
0.1	Gebruikte literatuur	4
1	Simulatie	5
1.0	Inleiding	5
1.1	Het vergaren van informatie (0): experimenteren	5
1.2	Het vergaren van informatie (1): simulatie	6
1.3	Computersimulatie	7
1.4	<i>Discrete event</i> simulatieprogrammeertalen	11
2	SIMULA	12
2.0	Inleiding	12
2.1	Het ontstaan van SIMULA	12
2.2	ALGOL 60	14
2.3	<i>Processes</i> en <i>activities</i> : de start van OO?	15
2.4	Invloeden van andere simulatieprogrammeertalen	15
2.5	Conclusie	16
3	Univac en SIMULA	17
3.0	Inleiding	17
3.1	De overeenkomst UNIVAC - NCC	17
3.2	Motieven van UNIVAC	19
3.3	De verhouding tussen machine en software	21
3.4	Conclusie	22
4	Van Simula naar Simula 67	24
4.0	Inleiding	24
4.1	Van SIMULA naar SIMULA 67	24
4.2	Was SIMULA 67 een logische opvolger van SIMULA	26
4.3	Conclusie	28
5	Conclusie	29
6	Literatuurlijst	32

Hoofdstuk 0

Inleiding

‘Objectgeoriënteerd’ is een van de vele modewoorden waaraan de informatica rijk is. In de jaren negentig van de twintigste eeuw leek alles objectgeoriënteerd te moeten zijn, was het dat nog niet, dan kwam er wel een nieuwe objectgeoriënteerde versie. Programmeertalen die jarenlang de gebruiker tevreden stelden zonder objectgeoriënteerde mogelijkheden, werden in nieuwere versies uitgebreid met dergelijke mogelijkheden. Ook werden er nieuwe talen ontwikkeld, gebaseerd op het objectgeoriënteerde paradigma, zoals C++ en Java.

Toch is de idee van objectgeoriënteerde talen al ouder. In 1967 werden de eerste stappen gezet op het objectgeoriënteerde pad met het ontstaan van SIMULA 67, dertien jaar later was de eerste volledig objectgeoriënteerde programmeertaal een feit: Smalltalk 80¹.

Mijn eerste idee was een onderzoek te doen naar de wortels van het objectgeoriënteerd programmeren. Echter ik werd, natuurlijk terecht, er op gewezen dat zo’n onderzoek veel te groot zou worden. Verkleining van het onderwerp was noodzakelijk, en mijn docent maakte me attent op het feit dat de beide makers van de SIMULA-talen, Ole-Johan Dahl en Kristen Nygaard in 2002 gestorven waren. De geschiedenis van SIMULA en deze twee personen zou een goede beperking van het onderwerp zijn.

Tijdens het onderzoek in de literatuur over dit onderwerp, trof ik in ‘The Development of the SIMULA Languages’ van Ole-Johan Dahl en Kristen Nygaard een opsomming aan van een aantal redenen waarom men op het Noorse rekencentrum², waar zowel Dahl als Nygaard werkzaam waren, niet echt enthousiast was over de eerste ideeën voor SIMULA:

1. *There would be no use for such a language as SIMULA.*
2. *There would be use, but it had been done before.*
3. *Our ideas were not good enough, and we lacked in general the competence needed to embark upon such a project, which for these reasons never would be completed.*

¹Robert W. Sebasta, *Concepts of programming languages* (vijfde druk; Boston 2002), 458.

²In de literatuur vaak Norwegian Computer Center of kortweg NCC genoemd. In het Noors heet het *Norsk Regnesentral* (<http://www.nr.no>). Dit soort rekencentra werden rond 1950 in veel landen opgericht om in te spelen op de nieuwe ontwikkelingen in de wiskunde (en de informatica). In Nederland was ook een rekencentrum, het huidige Centrum voor Wiskunde en Informatica (<http://www.cwi.nl>).

4. *Work of this nature should be done in countries with large resources, and not in small and unimportant countries like Norway.*³

Dit zette me aan het denken. Aan de ene kant vonden de collega's van Dahl en Nygaard SIMULA overbodig. Aan de andere kant blijkt datzelfde SIMULA later wel de basis voor het objectgeoriënteerde programmeren dat in de jaren negentig op de informatica zo'n grote invloed zal hebben. Hiermee had ik ook een interessante beperking van het onderwerp gevonden, de vraagstelling van dit document is dan ook: *was SIMULA overbodig?*

Deze vraag mag zelf overbodig lijken, dat is ze echter niet. Mocht het antwoord op deze vraag positief zijn, dan zou blijken dat overbodig geachte ontwikkelingen toch van belangrijke betekenis kunnen zijn. Dat het opnieuw uitvinden van het wiel toch tot vernieuwende resultaten kan leiden. Kortom dat overbodigheid niet het belangrijkste criterium zou moeten zijn om iets wel of niet te doen.

Toch dient de vraagstelling enige verklaring, het woord 'overbodig' is niet eenduidig. Een programmeertaal kan overbodig zijn wat betreft vernieuwing, maar kan toch een gat in het aanbod van programmeertalen opvullen. De bedoeling in dit werkstuk is om 'overbodig' breed op te vatten, dus zowel de informatica-aspecten als de gebruiksaspecten van de programmeertaal komen aan bod.

Informatica-aspecten zijn die aspecten in de programmeertaal die de programmeertaal van toegevoegde waarde laten zijn voor de informatica als geheel. In het geval van SIMULA is dat bijvoorbeeld het object-klasse-idee.

Gebruiksaspecten zijn aspecten die het gebruik van de programmeertaal betreffen, zoals de omgeving waarin de programmeertaal functioneert. Denk bijvoorbeeld aan de computers waarvoor de taal beschikbaar is, of de geografische locatie waarin het gebruikt wordt.

0.0 Opzet van het document

Het antwoord op de vraagstelling wordt gezocht door middel van het stellen van een aantal deelvragen. In het volgende hoofdstuk wordt ingegaan op simulatie en simulatieprogrammeertalen. Voordat de zin en onzin van SIMULA besproken kan worden, moet duidelijk zijn wat simulatie precies is, waarom het er is, en waarom SIMULA in de jaren zestig tot ontstaan kwam. Tevens komen eigenschappen van *discrete event* simulatietalen aan bod, zodat duidelijk wordt wat voor soort taal SIMULA is.

Daarna, in hoofdstuk 2 wordt gekeken naar wat SIMULA toevoegt aan die simulatieprogrammeertalen besproken in hoofdstuk 1. Voegt SIMULA echt iets toe, of is ze eigenlijk overbodig?

Het volgende onderwerp, besproken in hoofdstuk 3 is een belangrijke overeenkomst in de totstandkoming van SIMULA: een UNIVAC computer voor weinig voor een SIMULA-compiler. Blijkt uit deze overeenkomst dat SIMULA overbodig genoemd kan worden?

Nadat SIMULA klaar was en ingezet werd, wilden Dahl en Nygaard SIMULA uitbreiden tot een algemeen toepasbare programmeertaal, wat leidde tot de totstandkoming van SIMULA 67. De vraag die gesteld kan worden is of deze stap

³Ole-Johan Dahl en Kristen Nygaard, 'The Development of the SIMULA Languages', *ACM SIGPLAN Notices* (1978) vol. 13 nr. 8, 245 - 272, aldaar 247.

van SIMULA naar SIMULA 67 aangeeft of SIMULA overbodig was en dat daarom zo veel veranderen moest, of dat de verandering om andere redenen gewenst was.

Tenslotte, in hoofdstuk 5, de conclusie, wordt de vraagstelling beantwoord nadat een korte samenvatting van het voorgaande is gegeven.

0.1 Gebruikte literatuur

Bij het maken van dit werkstuk zijn een aantal verschillende soorten van literatuur gebruikt. Voor het eerste hoofdstuk, wat over simulatie en simulatietalen in het algemeen gaat is veel gebruik gemaakt van Wikipedia-artikelen⁴ en een aantal overzichtsartikelen.

‘Computer Simulation — Discussion of the Technique and Comparison of Languages’ van Daniël Teichroew en John Francis Lubin is een artikel uit 1966 met de stand van zaken op dat moment. Dit artikel, of althans de tabel (zie pagina 21) met algemene kenmerken over simulatietalen, is ook in de andere hoofdstukken gebruikt.

Daarnaast is ‘A history of discrete event simulation programming languages’ van Richard E. Nance uit 1993 aanbevelenswaardig omdat het een volledig overzicht van de ontwikkeling van de simulatietalen geeft. Dit is meer een historisch perspectief op de ontwikkeling van *discrete event* simulatietalen.

Voor de andere hoofdstukken, waarin het betoog van dit werkstuk gemaakt wordt, geldt dat het uitgangspunt ‘The development of the SIMULA languages’ van Kristen Nygaard en Ole-Johan Dahl uit 1978 is. Dit werk is hét overzicht van de SIMULA-talen, achteraf geschreven door de makers van die talen zelf.

Nadeel is wel dat dit artikel ook door alle andere auteurs die artikelen over de SIMULA’s geschreven hebben, gebruikt is als het uitgangspunt. Tevens is het artikel niet objectief, de auteurs putten veel uit herinneringen en eigen archief, zodat de controleerbaarheid gering is.

Dat neemt niet weg dat er toch een aantal artikelen geschreven zijn die een historische verhandeling over de SIMULA’s bevatten. Deze zijn: ‘Compiling SIMULA: a historical study of technological genesis’ van Jan Rune Holmevik uit 1994; en ‘Simula and Smalltalk: a social and political history’ van Benedict Dugan, ook uit 1994.

Het artikel van Holmevik verdient extra lof omdat hij in staat is geweest meer primaire bronnen te gebruiken. Ook heeft hij veel mensen die bij de SIMULA-talen betrokken zijn, geïnterviewd. Daar een aantal van die bronnen, maar ook gebruikte literatuur, in het Noors gesteld is, en derhalve voor mij onbruikbaar, is zijn stuk extra waardevol geweest.

Tenslotte wil ik nog opmerken dat er meer artikelen en literatuur over SIMULA 67 zijn verschenen en te vinden dan over SIMULA. Misschien is deze constatering al een antwoord op de hoofdvraag van dit werkstuk.

⁴www.wikipedia.org.

Hoofdstuk 1

Simulatie

1.0 Inleiding

Zoals in hoofdstuk 0 al aangegeven is, moet, voordat de hoofdvraag over de overbodigheid van SIMULA kan worden beantwoord, eerst dieper ingegaan worden op simulatie in het algemeen. SIMULA is immers een simulatieprogrammeertaal, een *discrete event* simulatieprogrammeertaal om precies te zijn.

Daarom ligt in dit hoofdstuk de nadruk op deze *discrete event* simulatieprogrammeertalen. Voordat die programmeertalen besproken worden, zal echter simulatie, maar bovenal computersimulatie, besproken worden.

1.1 Het vergaren van informatie (0): experimenteren

Het doel van simulatie is het vergaren van informatie over een systeem. Het woord ‘systeem’ moet hier breed geïnterpreteerd worden, het is meer een verzamelterm. Systemen kunnen heel reëel zijn, denk aan een auto of een computer, maar ze kunnen ook erg abstract zijn, zoals groepsvorming, een voorbeeld uit de sociologie.¹

Systemen kunnen door mensen gemaakt zijn, maar evengoed kan het een biologisch of fysisch proces zijn. Zo kan een konijn een systeem genoemd worden, of een bekend voorbeeld: het weer. In de meteorologie wordt veel gebruik gemaakt van computersimulaties die de gedragingen van het weersysteem proberen te voorspellen.

In allerhande systemen bestaat interesse, en om een systeem goed te leren kennen is informatie over zo’n systeem nodig. Het vergaren van informatie is dus een belangrijke taak, en de meest gewenste methode om informatie te vergaren is door middel van experimenteren. De door experimenten verkregen informatie heeft immers betrekking op de werkelijkheid.

Toch zijn er experimenteren waar problemen aan kleven:

1. Gevaarlijke experimenten. Bepaalde experimenten kunnen gevaarlijk zijn en daardoor zo goed als ongewenst. Experimenteren met wapensystemen of kernenergie bijvoorbeeld zijn niet alleen erg gevaarlijk, vaak zijn ze ook

¹Bij het maken van deze paragraaf is gebruik gemaakt van Jaroslav Sklenar, ‘Simulation’, <http://staff.um.edu.mt/jskl1/simul.html>, laatst bezocht 17 augustus 2004.

ethisch onverantwoord. Denk echter ook aan grotere systemen als de ecologie, of kleinere milieus, experimenteren met dergelijke systemen kunnen leiden tot erg grote negatieve gevolgen.

2. Ethisch onverantwoorde experimenten. Vooral experimenten met mensen en in mindere mate met andere levende wezens, zullen vaak ethisch niet te tolereren zijn. Denk aan de vele protesten tegen vivisectie, of de door doctor Mengele uitgevoerde experimenten in de vernietigingskampen in de tweede wereldoorlog.
3. Dure experimenten. Zeker experimenten uitgevoerd door de industrie zullen vaak wegens de kosten ervan onwenselijk zijn. Het testen van de veiligheid van een auto is een dure zaak als de testen enkel met echte auto's uit gevoerd kunnen worden.
4. Onmogelijke experimenten. Experimenten van systemen in ontwikkeling zullen niet uitgevoerd kunnen worden omdat het systeem eenvoudigweg nog niet bestaat. Misschien dat wel onderdelen van het systeem bestaan, dan kunnen daar experimenten op uitgevoerd worden, maar het echte systeem kan pas getest worden als het bestaat.

Een andere categorie van experimenten die fysiek onmogelijk zijn, kunnen in de astronomie gevonden worden. Een ster onderzoeken zal voorlopig vanaf een afstand gedaan moeten worden.

Een laatste categorie onmogelijke experimenten betreffen experimenten met meer abstracte systemen zoals die in de sociale wetenschappen beschreven worden. Experimenteren met een natie, een bepaalde groep mensen of iets dergelijks is onmogelijk door de grootte ervan, laat staan of het gewenst is.

Het is dus duidelijk dat experimenteren niet altijd de beste methode is om informatie te vergaren over een systeem. Om toch informatie te verkrijgen worden dan meestal andere methoden ingezet: analyse en simulatie².

1.2 Het vergaren van informatie (1): simulatie

Als experimenteren niet mogelijk is, kan simulatie uitkomst bieden. Natuurlijk heeft experimenteren de voorkeur bij het vergaren van informatie over een systeem omdat simulatie de informatie over het systeem niet uit het systeem zelf haalt maar uit een model van het systeem.³

Dat model is dan ook een belangrijk aspect van de simulatie. Immers hoe beter het model de werkelijkheid benadert, hoe meer de resultaten van de simulatie de resultaten van de experimentele methode zullen benaderen. Zo'n model kan, net als het geval bij systemen was, zowel abstract als concreet zijn. Een prototype van een auto is een voorbeeld van een concreet model, eigenlijk net zo tastbaar en concreet als de auto zelf. Een wiskundig model van een warme golfstroom is daarentegen veel abstracter dan de golfstroom zelf.

²Sklenar, 'Simulation'. Hierin worden drie basismethoden voor het vergaren van objectieve informatie over een systeem genoemd: experimenteren, analyse en simulatie. Op experimenteren en analyse wordt niet meer ingegaan, het onderwerp is immers simulatie.

³Bij het maken van deze paragraaf is gebruik gemaakt van Sklenar, 'Simulation'.

Voor het maken van een model van een systeem is wel voldoende kennis over dat systeem nodig om het zo waarheidsgetrouw mogelijk te krijgen. Zeker bij een wiskundig model is het noodzakelijk te beschikken over voldoende kwantitatieve gegevens om een reëel model te kunnen creëren. Het zijn immers die gegevens, ook wel parameters genoemd, die het systeem beschrijven. Zijn er te weinig parameters, dan is het onwaarschijnlijk dat het model bruikbare gegevens over het systeem op kan leveren.

Modellen kunnen dus ‘echt’ zijn ofwel tastbaar, maar ook enkel op papier bestaan als een verzameling formules, parameters en afhankelijkheden. Een computermodel zou een tussenmodel genoemd kunnen worden. In wezen is het een wiskundig model, maar het model kan wel, althans virtueel, tastbaar gemaakt worden door een visualisatie. Computerspellen zijn hiervan een goed voorbeeld, de daarin aanwezige modellen lijken echt, soms bijna levensecht. Toch zijn het niet meer dan ‘eenvoudige’ wiskundige modellen. Computersimulaties zijn het onderwerp van de volgende paragraaf.

Eenmaal een model geconstrueerd, kan de simulatie beginnen. Bij concrete modellen zal dat gewoon een experimentele methode zijn, en is de grens tussen simulatie en experiment een stuk minder duidelijk dan bij abstracte, wiskundige modellen.

Het simuleren met een wiskundig model komt neer op het herhaaldelijk uitvoeren van een simulatiestap. Een zo’n stap behelst het berekenen van een nieuwe toestand van het model gegeven de oude toestand. Deze nieuwe toestand is dan weer de oude toestand voor de volgende simulatiestap. Het klinkt eenvoudig, maar dra het model complexer wordt zal het berekenen van zo’n simulatiestap een moeilijke taak worden, en een volledige simulatie berekenen wordt dan een hels karwei of, gezien de tijd die het kost, zo goed als onmogelijk.

Het is dit probleem van de rekentijd, en in mindere mate de complexiteit van de berekening, die door gebruik te maken van computers voor een groot deel opgelost kan worden. Computers zijn namelijk in staat om over langere tijd op hoge snelheid veel berekeningen uit te voeren.

1.3 Computersimulatie

Computersimulatie is simulatie met behulp van een computer. Van een (wiskundig) model van een systeem wordt een computerprogramma gemaakt. Het is dit computerprogramma dat gegeven een aantal parameters, de simulatie uitvoert en resultaten teruggeeft. Dit is de basis van computersimulatie. Op allerlei terreinen kan deze basis uitgebreid, bijvoorbeeld met een grotere rol voor menselijke interactie.⁴

Er zijn twee verschillende typen computersimulaties te onderscheiden, behoudens tussenvormen en aanpassingen van die twee typen:

1. Discrete of stochastische simulatie, ook wel digitale simulatie genoemd. Discrete simulaties heten discreet omdat de simulatie draait om *events* die op

⁴Bij het maken van deze paragraaf is gebruik gemaakt van Wikipedia, ‘Computer simulation’, http://en.wikipedia.org/wiki/Computer_simulation, laatst bezocht 18 augustus 2004; Shannon, ‘Introduction to simulation languages’ en Nance, ‘A history of discrete event simulation programming languages’.

een bepaald tijdstip, simulatietijd welteverstaan, plaatsvinden. Deze *events* worden geordend naar de tijd van uitvoeren geplaatst in een wachtrij⁵. Een voor een worden de *events* uitgevoerd, wat meestal weer zal resulteren in het toevoegen van nieuwe *events* aan de wachtrij.

Het is deze computersimulatievorm die het meest voorkomt⁶, en is uitermate geschikt voor de digitale computer, zelf ook een discrete machine. Opgemerkt moet worden dat simulatietijd en echte tijd in deze simulatievorm kan, en vaak ook zal, verschillen. Zo kunnen langdurige processen, bijvoorbeeld van systemen in de geologie, eenvoudig versneld gesimuleerd worden. Of juist het omgekeerde, kunnen erg kortedurende processen, bijvoorbeeld van systemen op atomair niveau, erg vertraagd uitgevoerd worden.

2. Analoge simulatie is eigenlijk een vreemde eend in de bijt, het betreft hier het uitvoeren van differentiaalvergelijkingen die periodiek uitgerekend worden. Voor de grote opkomst van de digitale (mini)computer in de jaren tachtig van de twintigste eeuw werden deze simulaties vaak uitgevoerd op zogenaamde analoge computers, waarbij de verschillende differentiaalvergelijkingen in elektronische componenten uitgedrukt en berekend werden. Tegenwoordig worden deze simulaties uitgevoerd op gewone digitale computers waarbij het analoge gedrag wordt nagebootst.

Analoge simulatie wordt veel toegepast in de electrotechniek om elektronische circuits te simuleren, bijvoorbeeld bij het ontwerpen van nieuwe chips, processors en andere hardware. Het ontwerp en implementeren van dergelijke hardware is een kostbare aangelegenheid, dus simulatie van dergelijke systemen om de kwaliteit ervan te kunnen beoordelen voordat het productieproces ervan begint werkt zeer kostenbesparend.

De reden dat computersimulatie eind jaren vijftig, begin jaren zestig zo populair werd, is voornamelijk te danken aan de ontwikkeling en opkomst van de computer. Welliswaar waren de computers nog steeds immense bakbeesten van machines, maar ze waren eind jaren vijftig een stuk betrouwbaarder en werkbaarder dan de machines van eind jaren veertig en begin jaren vijftig.

De computer kwam steeds meer in beeld in de grotere organisaties en computersimulatie werd dan ook op veel gebieden gezien als een bruikbaar hulpmiddel. Niet enkel bij de natuurwetenschappelijke of technische vakgebieden, ook de sociale wetenschappen zagen heil in simulatie.

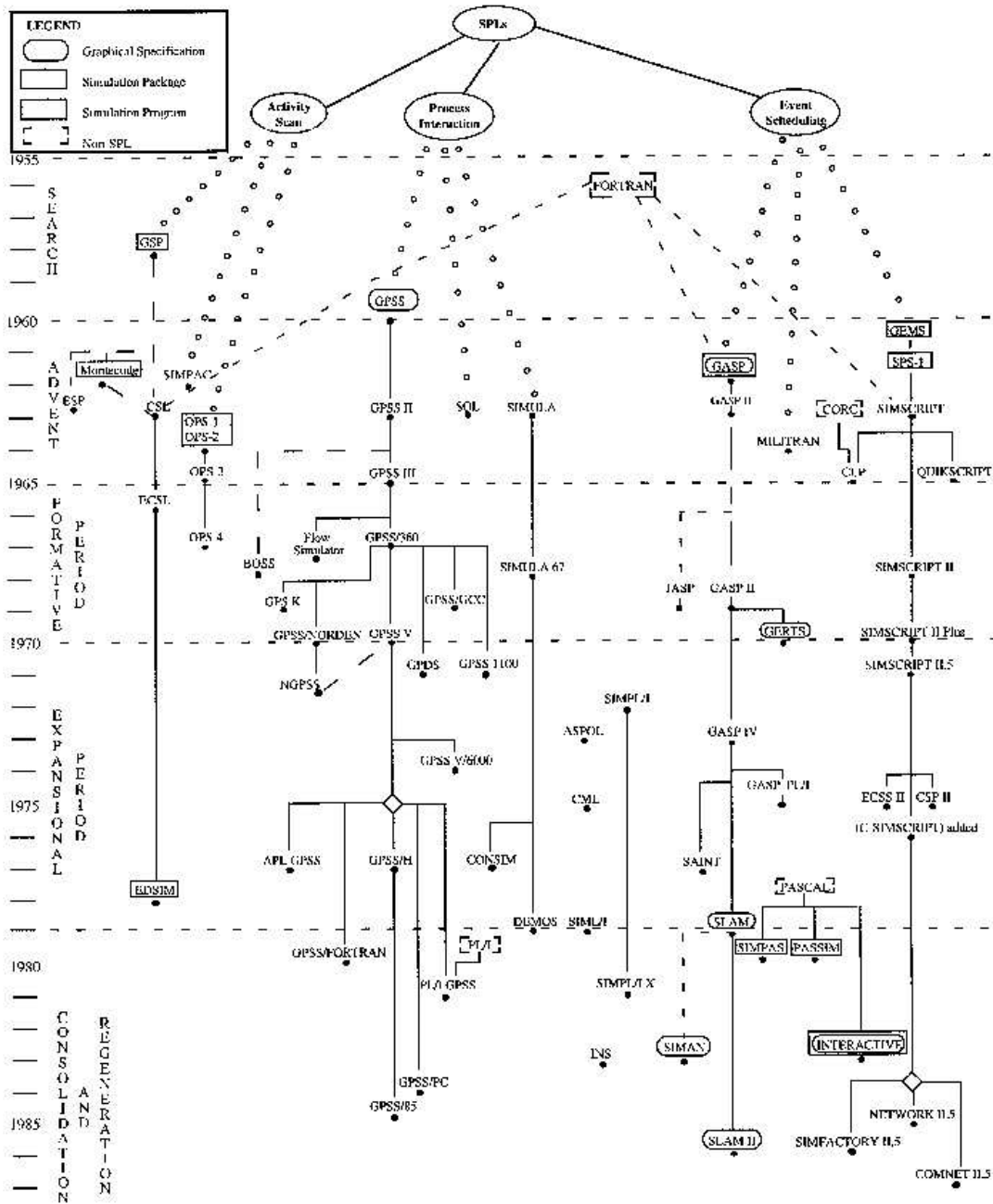
Als eerder gezegd, is een computersimulatie in wezen niet anders dan een computerprogramma dat op een computer uitgevoerd wordt. Over het belang van het type computer bij computersimulatie in de jaren vijftig en zestig van de twintigste eeuw wordt in hoofdstuk 3 (pagina 17) dieper ingegaan. Voor nu is de aandacht voor de computerprogramma's.

Computerprogramma's worden gemaakt met behulp van programmeertalen, met computersimulatie is dat niet anders. De keuze voor een programmeertaal voor simulatie is meestal tussen een speciale simulatieprogrammeertaal of een algemene programmeertaal. Betekende voor de uitvinding van FORTRAN dat

⁵In de informatica wordt een wachtrij vaak een queue genoemd.

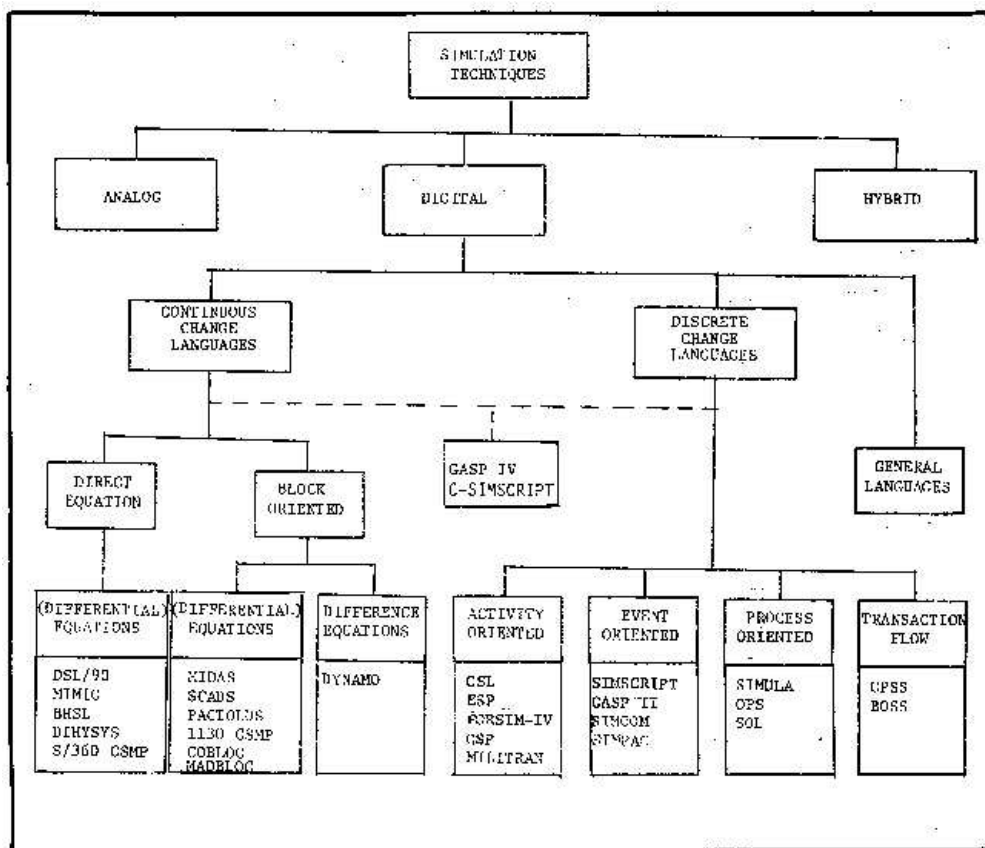
⁶Wikipedia, 'Computer simulation'.

Figuur 1.1: Afkomstboom van *discrete event* simulatieprogrammeertalen



Figuur overgenomen uit: Richard E. Nance, 'A history of discrete event simulation programming languages', *ACM Sigplan Notices* (maart 1993) vol. 28 nr. 3 149 - 175, aldaar pagina 6.

Figuur 1.2: Onderverdeling in simulatiesoorten en simulatietalen



Figuur overgenomen uit: Robert E. Shannon, 'Introduction to simulation languages', *Winter Simulation Conference* (december 5-7 1977), 14 - 20, aldaar 19.

simulatieprogramma's in assembleertaal geschreven werden, na FORTRAN kwamen andere hogere niveau programmeertalen zoals ALGOL, zodat het schrijven van simulatieprogramma's vergemakkelijkt werd.

Naast algemene programmeertalen zijn er ook speciale simulatietalen ontwikkeld, vooral in de jaren zestig en zeventig, zie ook figuur 1.3 (pagina 9) waarin een afkomstboom gemaakt is van *discrete event* simulatieprogrammeertalen.

De speciale simulatieprogrammeertalen kunnen onderverdeeld worden in een tweetal typen, dit wordt ook samengevat in figuur 1.3 (pagina 10):

1. *Continuous event* simulatietalen, ook wel *continuous change* simulatietalen genoemd, zijn bedoeld voor die modellen waar tijd en toestanden van het model niet altijd een precies gedefiniëerde relatie hebben. Vaak komt dit voor in fysieke systemen, zoals in de ecologie en economie.
2. *discrete event* simulatietalen of *discrete change* simulatietalen genoemd, maken gebruik van een wiskundig model van een systeem waar de toestand van het model op vaste punten in de tijd verandert. Een veelgebruikt voorbeeld is dat van reizigersstromen op een vliegveld.

Deze simulatietalen zijn vaak gemaakt als uitbreiding van bestaande talen, in de jaren zestig veelal FORTRAN en soms ALGOL, bijvoorbeeld als een preprocessor, een programma dat de programmatekst van de simulatietaal vertaald in

de taal waarop de simulatietaal een uitbreiding is. Of als een pakket van een aantal procedures in de taal waarvan de simulatietaal een uitbreiding is. Daarnaast zijn er simulatietalen die als volledig nieuwe programmeertalen zijn gemaakt, en dus niet gebouwd op bestaande en bekende programmeertalen.

1.4 *Discrete event* simulatieprogrammeertalen

Over *continuous event* simulatietalen zal in dit document weinig aandacht worden geschonken. Het hoofddoel is immers de vraag te beantwoorden of SIMULA overbodig was. En aangezien SIMULA een representant is van de *discrete event* simulatietalen, zal in dit document en bovenal in deze paragraaf de nadruk ook liggen bij deze smaak van simulatietalen.

In figuur 1.3 (pagina 9) zijn al een groot aantal *discrete event* simulatietalen te zien, de meeste stammen af van drie talen: GPSS, GASP en GEMS, van die laatste is vooral SIMSCRIPT een bekende en veelgebruikte simulatietaal.

In ‘A history of discrete event simulation programming languages’ van Richard E. Nance worden een aantal eisen opgesomt waaraan een *discrete event* simulatietaal in meer of mindere mate moet voldoen⁷. Deze eisen zijn:

1. Het genereren van willekeurige getallen⁸ om onzekerheid te kunnen representeren. Onzekerheid die in veel simulaties voor moet komen, omdat veel systemen beschreven kunnen worden met stochastische processen.
2. Procesomzettere die de gegenereerde willekeurige getallen om kunnen zetten naar stochastische processen.
3. Mogelijkheden om lijsten te manipuleren, de events in de simulaties zijn verzameld in lijsten en op die lijsten moeten regelmatig allerhande operaties uitgevoerd worden.
4. Procedures voor statistische analyse om het gedrag van het systeem (in het model) te kunnen weergeven en analyseren.
5. Generatie van rapporten over de simulatie. Door de simulatie worden veel gegevens gegenereerd, en deze gegevens moeten op een goede en duidelijke manier gepresenteerd kunnen worden.
6. Een tijdproces om de tijd explicite te kunnen representeren.

⁷Nance, ‘A history of discrete event simulation programming languages’, 150.

⁸In de informatica wordt dit vaak een random number generator genoemd.

Hoofdstuk 2

SIMULA

2.0 Inleiding

De punten van kritiek die Kristen Nygaard en Ole-Johan Dahl kregen van hun collega's bij het Norwegian Computing Center zijn in het eerste hoofdstuk gegeven (pagina 2). Vooral de eerste twee punten zijn onderwerp van dit hoofdstuk.

De vraag die gesteld wordt, is: *In hoeverre voegde SIMULA iets toe aan de al bestaande of in ontwikkeling zijnde discrete event simulatietaalen?* Om deze vraag te kunnen beantwoorden is nodig de bepalende eigenschappen van SIMULA te bespreken. Deze bepalende punten zijn het gebruik van ALGOL 60 en de notie van *processes* en *activities*, wat een eerste aanzet tot objecten en klassen genoemd kan worden. Dit wordt besproken in respectievelijk paragraaf 2.2 en paragraaf 2.3.

Een andere belangrijke eigenschap is de machine waarvoor SIMULA ontwikkeld werd, maar dat is onderwerp van het volgende hoofdstuk. Voordat echter de hier genoemde eigenschappen besproken worden, wordt eerste een korte beschrijving gegeven van de ontwikkeling van SIMULA, zodat duidelijk is waar deze eigenschappen opkomen.

Een ander belangrijk punt dat besproken wordt, is de invloed die andere simulatietaalen gehad hebben op SIMULA. Zijn er aanwijzingen dat SIMULA erg beïnvloed is door een andere taal, of juist helemaal niet? Tenslotte wordt in de conclusie de vraagstelling van dit hoofdstuk beantwoord.

2.1 Het ontstaan van SIMULA

De ideeën die aan SIMULA ten grondslag liggen kwamen voort uit het werk van Kristen Nygaard in de jaren vijftig en begin jaren zestig van de vorige eeuw. Hij was eerst werkzaam bij de Norwegian Defence Research Establishment (NDRE) bij het onderdeel *operational research*. Later vertrekt hij naar het Noorse rekencentrum (NCC) om een informaticainstituut op te bouwen met als hoofddoel *operational research*.

Veel van de daar uit te voeren taken bleken voort te komen uit vergelijkbare methodologische problemen: 'the necessity of using simulation, the need of concepts and a language for system description, lack of tools for generating simulation programs.'¹ Kortom een simulatietaal was een noodzakelijk hulpmiddel.

¹Dahl, 'The Development of the SIMULA Languages', 246.

Figuur 2.1: Nygaard (tweede van rechts) en Dahl (rechts) voor een SIMULA-bord op het Noorse Rekencentrum



Figuur afkomstig van http://www.nrk.no/programmer/radio/verdt_a_vite/-2720488.html met onderschrift: 'Dahl (th) og Nygaard (ved siden av) ved "Sumula-tavlaï Norsk regnesentral (foto:NR)'

Ook de achtergrond van Ole-Johan Dahl ligt bij het NDRE. Toch zal hij pas met Nygaard kennismaken als Nygaard zijn ideeën daadwerkelijk wilde gaan verwezenlijken: Nygaard had daarvoor een goede programmeur nodig. Op 6 januari 1962 ontmoetten zij elkaar om over de simulatietaal te praten², deze ontmoeting zou uitgroeien in een jarenlange samenwerking. In mei 1962 was de taal zover doorontwikkeld dat zij aan de buitenwereld gepresenteerd kon worden. Tijdens deze korte periode was Dahl zo veel betrokken geraakt bij het proces dat ook hij naar het NCC vertrok.³

Datzelfde NCC droeg de ideeën voor SIMULA geen echt goed hart toe, maar de leiding gaf een fiat aan SIMULA, mede door een overeenkomst met UNIVAC. NCC zou de krachtige en financieel onhaalbare UNIVAC-computer goedkoop kunnen krijgen als NCC op haar beurt weer voor een SIMULA-compiler voor diezelfde machine zou zorgen⁴. Over deze overeenkomst wordt in het volgende hoofdstuk dieper ingegaan (pagina 17).

Nygaard en Dahl verdelen de ontwikkeling van SIMULA in in vier fasen⁵. De eerste fase begint in de zomer van 1961 en duurt tot de herfst van het daaropvolgende jaar. In deze fase werden de eerste ideeën op papier gezet.

De tweede fase, ook ongeveer een jaar, liep tot september van het jaar 1963. Het resultaat van deze fase was de beslissing om SIMULA als een preprocessor en subroutineepakket van ALGOL 60 te maken, waardoor er een grotere flexibiliteit mogelijk werd.

ALGOL 60 bleek toch niet flexibel genoeg omdat het *single-stack* regime niet geschikt was om de simulatiemogelijkheden van SIMULA zoals Nygaard en Dahl die wilden, te ondersteunen.⁶ In de derde fase, van september 1963 tot maart

²Ibidem 246.

³Ibidem 246.

⁴Ibidem 247.

⁵Ibidem 247.

⁶Jan Rune Holmevik, 'Compiling SIMULA: A historical study of technological genesis', *IEEE Annals of the History of Computing* (1994) vol. 16 nr. 4, 25 - 37, aldaar 31

1964, ontwikkelde Ole-Johan Dahl een nieuw manier van geheugenbeheer, gebaseerd op een ‘twodimensional free area list’. Hierdoor verviel ook het preprocessoridee en werd de SIMULA-compiler bereikt door een uitbreiding en aanpassing van de al bestaande ALGOL-compiler.

Hierna, in de vierde fase, werd SIMULA geïmplementeerd. In december 1964 was de compiler af, SIMULA was klaar voor gebruik.

2.2 ALGOL 60

De eerste opvallende eigenschap van SIMULA is het feit dat de taal gebaseerd was op de programmeertaal ALGOL 60. ALGOL was ontwikkeld in de late jaren vijftig als reactie op FORTRAN. FORTRAN was de eerste, op grote schaal gebruikte gecompileerde taal, maar voornamelijk beschikbaar op IBM machines (te beginnen met de IBM 704). Destijds was FORTRAN zelfs eigendom van IBM⁷.

Het idee voor ALGOL was om een universele taal te ontwikkelen, dat werd ALGOL 58. Twee jaar later in 1960 werd de tweede ALGOL conferentie gehouden waarvan de uitkomst ALGOL 60 was. Alhoewel ALGOL niet veel gebruikt werd in de Verenigde Staten, werd ze wel de *dè facto* standaard discussietaal voor algoritmen in de literatuur⁸.

Omdat ALGOL niet veel gebruikt werd in de Verenigde Staten, is het ook niet vreemd dat FORTRAN steeds meer gebruikt werd. Gezien de grote invloed van de Verenigde Staten op de computerindustrie, denk aan IBM, is het niet verwonderlijk dat veel simulatietalen die in de jaren vijftig en zestig ontwikkeld werden, niet op ALGOL gebaseerd waren.

Op pagina 21 is een tabel opgenomen uit 1966⁹ waarin duidelijk te zien is dat ALGOL enkel als basis voor vier simulatieprogrammeertalen gebruikt is: ESP, SIMON, SIMULA en SOL. Van deze vier was enkel SIMULA in 1966 ‘in preparation’, de andere waren niet eens beschikbaar.

Aan de andere kant zijn er wel acht talen gebaseerd op FORTRAN, waarvan GASP en SIMSCRIPT de bekendste en meest gebruikte simulatietalen zijn. Waarschijnlijk is het feit dat SIMULA op ALGOL gebaseerd was een van de redenen waarom SIMULA niet echt doorbrak.

Volgens Ole-Johan Dahl werd de keuze voor ALGOL al in de eerste onwikkelfase gemaakt, voornamelijk vanwege de blokstructuur, de goede programmeerbaarheid en Europese patriotisme¹⁰. Deze keuze had ook zijn voordelen. ALGOL’s geheugenbeheer bleek namelijk ongeschikt voor simulatie zoals Nygaard en Dahl dat wilden hebben. Hierdoor verwierpen zij het idee van een preprocessor, en besloten zij SIMULA te verwezenlijken door een verandering aan te brengen aan de ALGOL-compiler¹¹.

Dit bleek een gouden ontwikkeling, hierdoor kon het *process* in SIMULA opgenomen worden. Een model in SIMULA werd nu een verzameling van *processes*

⁷Sebasta, *Concepts of programming languages*, 55.

⁸Ibidem 59, 60.

⁹Teichroew, ‘Computer Simulation’, 726.

¹⁰Ole-Johan Dahl, ‘The birth of object orientation: the Simula languages’, <http://heim.ifi.uio.no/olejohan/birth-of-oo.pdf>, 2.

¹¹Holmevik, ‘Compiling SIMULA’, 31.

die semiparallel uitgevoerd werden. En het waren deze *processes* en de onderliggende *activities* dat de basis voor het objectgeoriënteerde programmeren in SIMULA 67 zou gaan vormen.

2.3 *Processes en activities: de start van OO?*

Processes en *activities* dus, het is interessant te lezen hoe Dahl en Nygaard deze taalconstructies beschrijven in hun artikel *SIMULA — an ALGOL-based Simulation Language*¹² uit 1966:

‘The process concept is intended as an aid for decomposing a discrete event system into components, which are separately describable. In general, a process has two aspects: it is a data carrier and it will execute actions.’

In het voorgaande citaat is duidelijk het idee van een object te zien. Er zijn eigenschappen ofwel de data, en er is gedrag, ofwel acties op diezelfde data, ‘(...) combined into a block called a *process block*’, oftewel gecombineerd tot een object.

‘The description of a process is called an *activity* declaration.’ Deze activities kunnen in dit licht gezien worden als de klassen die de objecten ofwel de processes beschrijven. Dat zien Nygaard en Dahl zelf ook: ‘The concept of an “activity”, which is a class of processes described by the same declaration, is distinguished from the the concept of a ”process”, which is one dynamic instance of an activity declaration.’

Alhoewel de taal voor de verdere rest nog niet veel objectgeoriënteerde ideeën ondersteunde, was de basis ervoor wel gelegd. Hiermee onderscheidde SIMULA zich duidelijk van de andere simulatietalen, maar het geeft de taal niet echt andere of betere simulatiefunctionaliteit.

2.4 Invloeden van andere simulatieprogrammeertalen

Omdat SIMULA wat betreft simulatiefunctionaliteit niet echt afwijkt van andere talen, er kan min of meer hetzelfde uitgedrukt worden, is het zinvol te kijken naar eventuele invloeden van andere simulatieprogrammeertalen op SIMULA.

Nygaard en Dahl zelf gaven aan dat SIMSCRIPT de enige simulatietaal was waarmee ze goed bekend waren tijdens de ontwikkeling van SIMULA¹³. Het is dan ook niet verwonderlijk dat die taal een invloed heeft gehad op SIMULA.

Voor de verder rest, geven ze aan niet met andere talen genoeg bekend te zijn geweest. Niet met GPSS, maar later merkten ze wel dat ‘the ”transactions” of GPSS could in fact be looked upon as processes in quasi-parallel.’¹⁴

Over SOL, dat in juli 1964 uitkwam, wat ze een heel mooie taal vonden, schreven ze dat ‘others before us had had the idea of quasi-parallel processes in an ALGOL-like setting.’¹⁵

¹²Ole-Johan Dahl en Kristen Nygaard, ‘SIMULA — an ALGOL-based simulation language’, *Communications of the ACM* ed: D.E. Knuth (September 1966) vol. 9 nr. 9, 671 - 678, aldaar 672.

¹³Dahl, ‘The Development of the SIMULA Languages’, 253.

¹⁴Ibidem 253.

¹⁵Ibidem 253.

Kortom een van SIMULA's opvallende eigenschappen, het quasi-parallel uitvoeren van processen, was eerder een kind van de tijd waarin SIMULA ontwikkeld werd dan een specifiek nieuw iets. Zelfs het gebruiken van ALGOL als basistaal was iets waarmee SIMULA niet uniek was.

Het doet voorkomen alsof Dahl en Nygaard niet genoeg kennis hadden van andermans werk. In de jaren zestig ontstonden wel een groot aantal simulatietaalen, maar het is eerder meer van hetzelfde met verschillen op de details dan een vernieuwende ontwikkeling.

2.5 Conclusie

Duidelijk is gebleken dat SIMULA wel een aantal opvallende punten kende, denk aan het gebruik van ALGOL 60 en de eerste aanzetten tot het objectgeoriënteerd programmeren, maar dat SIMULA zelf geen vernieuwende simulatiefunctionaliteit bracht.

SIMULA mag met recht een kind van de tijd genoemd worden, zoals er meerdere talen waren die terzelfder tijd ontwikkeld werden. Jammer voor SIMULA is dat ALGOL een minder gebruikte programmeertaal was in de Verenigde Staten, waardoor SIMULA niet veel gebruikt werd.

Hoofdstuk 3

Univac en SIMULA

3.0 Inleiding

In het vorige hoofdstuk is het ontstaan van SIMULA besproken waarbij de overeenkomst tussen het Noorse rekencentrum (NCC) met UNIVAC genoemd werd. Voor een nadere verklaring werd naar dit hoofdstuk verwezen, en deze overeenkomst is dan ook het onderwerp van dit hoofdstuk.

Het was deze overeenkomst die de ontwikkeling van SIMULA mogelijk maakte bij het NCC. De overeenkomst was dus een belangrijke gebeurtenis voor de totstandkoming van SIMULA. Interessant is daarom het waarom van deze overeenkomst, zeker in het kader van de vraagstelling van dit document. Want het waarom van de overeenkomst kan een indicatie geven over de overbodigheid van SIMULA.

De vraagstelling van dit hoofdstuk is: *Wat waren de motieven achter de UNIVAC-overeenkomst?*. Deze vraag wordt beantwoord door een aantal deelvragen. Eerst wordt de overeenkomst zelf uitvoerig besproken. Daarna komen de motieven van UNIVAC voor deze overeenkomst aan bod, en wordt een aannemelijkheid voor al deze motieven bepaald.

Hierna wordt een belangrijk gegeven van de de informatica in de jaren zestig besproken: de machinegeïntereerdheid. De meeste software werd speciaal voor een bepaalde machine geschreven, en de machine werd speciaal voor de software die ervoor beschikbaar was gekocht. Deze verhouding tussen machine en software, zeker een belangrijk motief in de UNIVAC-overeenkomst, verdient een diepere benadering.

Tenslotte, in de conclusie, wordt de vraagstelling beantwoord.

3.1 De overeenkomst UNIVAC - NCC een overzicht

De overeenkomst tussen UNIVAC en het NCC moet gezien worden in het licht van de hardware, oftewel de computers die het NCC in de jaren vijftig gebruikte. Vanaf 1954 was dat een NUSSE-machine, maar drie jaar later komt er in Noorwegen een krachtigere computer (de Ferranti Mercury) zodat de NUSSE, en daarmee de taak van het NCC, steeds minder belangrijk voor Noorwegen werd¹.

¹Holmevik, 'Compiling SIMULA', 27.

In 1958 schafte het Centraal Bureau voor de Statistiek in Noorwegen zich een Deuce computer aan, en het NCC krijgt de mooie taak toebedeeld om deze machine te onderhouden. Twee jaar later blijkt dat het NCC dit niet meer kan volbrengen, de organisatie had namelijk veel meer werk gekregen². De Deuce-machine bleek ook niet meer te voldoen.

Er werd dus gezocht naar nieuwe hardware, naar een nieuwe computer voor het NCC. Tezeldertijd kwam er een Scandinavisch initiatief van de grond om een computernetwerk te bouwen waarvan ook het NCC deel van zou kunnen uitmaken. Hiertoe werd een GIER-computer in Denemarken besteld, bij het Deense Rekencentrum. De GIER was een middelgrote computer, die eigenlijk niet aan de eisen van het NCC voldeed. Het NCC wilde liever een echt groot mainframe, maar de financiële positie van het NCC leidde ertoe dat de GIER het hoogst haalbare was. In februari 1962 werd zo'n GIER officieel besteld³.

Tezeldertijd was Sperry Rand Corporation bezig met het promoten van hun nieuwe UNIVAC III en UNIVAC 1107. Daartoe nodigde het bedrijf enige topmannen van mogelijke Europese klanten uit, om in Amerika te kijken naar enkele in gebruik zijnde UNIVAC machines, waaronder Nygaard voor het NCC. Nygaard maakte van de mogelijkheid gebruik om SIMULA aan de Amerikanen te laten zien⁴.

James W. Nickitas, assistent van de directeur van UNIVAC Europa, was enthousiast over SIMULA en arrangeerde een ontmoeting tussen hem, Nygaard, Alvin M. Paster (manager van systems research), Robert W. Bremer (de baas van Nickitas, dus de directeur van UNIVAC Europa) en William R. Lonergan (de baas van Bremer). Bij deze ontmoeting bleken de vertegenwoordigers van UNIVAC serieus geïnteresseerd in SIMULA en een pakket voor lineair programmeren, dat ook in ontwikkeling bij het NCC was⁵.

Op 29 mei 1962 doet Nickitas Nygaard een informeel aanbod in een Griekse nachtclub, terwijl ze zaten te kijken naar een mooie buikdanseres⁶. Het aanbod hield in dat als NCC SIMULA en het pakket voor lineair programmeren voor UNIVAC zou maken, dat UNIVAC het NCC een UNIVAC 1107 tegen een gereduceerde prijs zou leveren.

Terug op het NCC wilde niemand Nygaard geloven, maar als eind juni Luther Harr, Nickitas en Walstam (directeur UNIVAC Scandinavië) hetzelfde aanbod officieel aan het NCC presenteerden, bleek het toch een reële mogelijkheid voor het NCC, te meer omdat de ontwikkeling van SIMULA en het pakket voor lineair programmeren die deel zouden uitmaken van de overeenkomst, nu als losse softwarecontracten aan het NCC aangeboden werden⁷.

Helaas was er nog een GIER-computer in bestelling, maar het aanbod van UNIVAC was te aantrekkelijk om te laten liggen. De UNIVAC-overeenkomst werd officieel ondertekend (zie ook figuur 3.2 op pagina 20) op 24 oktober 1962. Toch waren er een aantal problemen. Het afzeggen van de GIER bleek echter wat problemen op te leveren. In de lente van 1963, na een aantal onderhandelingen,

²Ibidem 28

³Ibidem 28

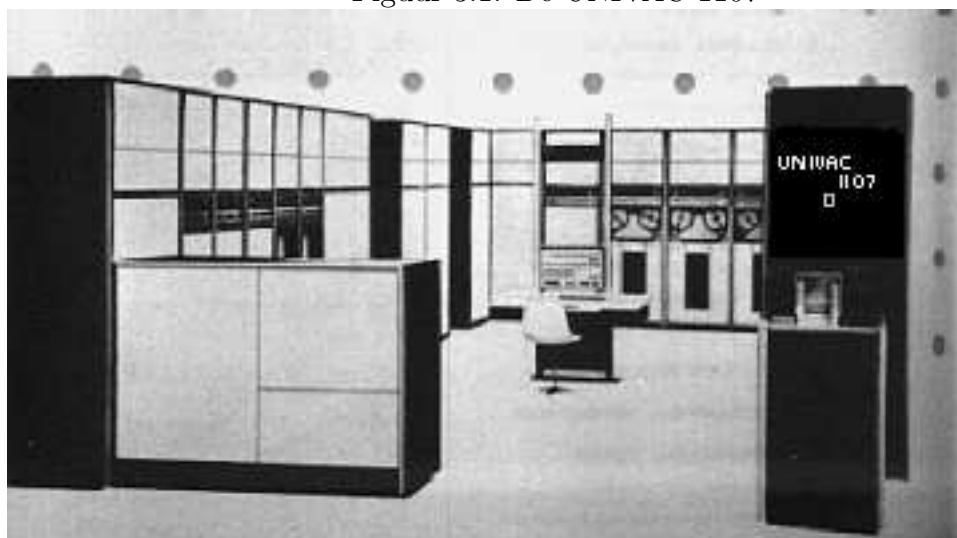
⁴Dahl, 'The Development of the SIMULA Languages', 254.

⁵Ibidem 254

⁶Ibidem 254

⁷Ibidem 255

Figuur 3.1: De UNIVAC 1107



Figuur afkomstig van: <http://www.computermuseum.li/Testpage/Univac-1107.gif>.

werden die problemen opgelost. Een ander probleem was het feit dat de software voor de UNIVAC 1107 nog niet gereed was in 1962, dat kon na wat uitstellen pas in augustus 1963 geleverd worden⁸.

De geleverde software voldeed echter niet aan NCC's eisen omtrend de kwaliteit ervan, zodat ze UNIVAC hiervoor een claim oplegden. Als in juni 1964 het NCC de software goed bevond, wilde UNIVAC de claim betalen met een hardwareupgrade. Resultaat was dat het NCC een zeer goede computer kreeg voor erg weinig geld⁹

3.2 Motieven van UNIVAC voor de overeenkomst met het NCC

In de vorige paragraaf is een beschrijving gegeven van het verloop van de overeenkomst tussen het NCC en UNIVAC. Het resultaat van die overeenkomst was een goede en goedkope computer voor het NCC en de ontwikkeling van de taal SIMULA. Maar het aan het waarom van de overeenkomst is nog weinig aandacht besteed.

Dat waarom, de motieven van UNIVAC om deze overeenkomst te sluiten en daarmee SIMULA het levenslicht te laten zien, is het onderwerp van deze paragraaf.

Zoals van een commerciële organisatie als Sperry Rand Corporation verwacht kan worden, waren de motieven voornamelijk commercieel georiënteerd. De reden waarom Nygaard, of beter gezegd het NCC, uitgenodigd werd voor de tour in Amerika langs UNIVAC-gebruikers, was het promoten van de UNIVAC computers in Europa.

⁸Holmevik, 'Compiling SIMULA', 30.

⁹Ibidem 30.

Figuur 3.2: Kristen Nygaard (tweede van rechts) ondertekend de overeenkomst voor de UNIVAC 1107



Figuur afkomstig van http://www.nrk.no/programmer/radio/verdt_a_vite/-2720488.html met onderschrift: ‘Kristen Nygaard (nr. 2 fra v) undertegner avtalen om innkjøp av Univac 1107’.

Voor dezelfde reden, het promoten van UNIVAC in Europa, wilde UNIVAC snel een demonstratieplaats creëren in Europa. Nygaard heeft UNIVAC ervan weten te overtuigen dat het NCC de rol van zo’n demonstratieplaats goed zou kunnen vervullen¹⁰. Dit leidde tot het aanbod van een UNIVAC 1107 voor de halve prijs plus het implementeren van SIMULA en een lineair programmeerpakket voor de UNIVAC.

Daar komt bij dat UNIVAC op de hoogte was van het feit dat het NCC een GIER besteld had, en wilde maar wat graag dat het NCC een UNIVAC zou aanschaffen. Het SIMULA-gedeelte van het aanbod leverde UNIVAC in het NCC een goede bondgenoot op, namelijk Nygaard¹¹. Dat Nygaard, als hem later opdracht gegeven werd een rapport te maken over de keuze tussen de UNIVAC en de GIER, voor de UNIVAC kiest¹², mag dan ook geen verwondering opwekken.

Toch waren er een aantal andere motieven van UNIVAC om de overeenkomst te willen sluiten, om de beschikking over SIMULA te krijgen. Een ervan is dat UNIVAC SIMULA het meest geschikt achtte ‘to establish a strong systems simulation capability for our own use here in St Paul.’¹³ Hieruit blijkt een interesse voor SIMULA wegens de simulatiemogelijkheden van de taal.

Een andere motief komt van de directeur systems programming van UNIVAC, Robert Bremer. Hij was een liefhebber van ALGOL 60 en wilde het gebruik ervan stimuleren ten opzichte van FORTRAN¹⁴. Aangezien SIMULA gebaseerd zou worden op ALGOL 60, zag hij SIMULA een goede troef in zijn strijd voor acceptatie van ALGOL 60.

Een tweede voordeel dat het ALGOL 60 gebaseerd zijn van SIMULA UNIVAC bracht, was het feit dat in Europa ALGOL een veel gebruikte taal was. Om UNIVAC’s marktaandeel in Europa te vergroten zou SIMULA een goede

¹⁰Holmevik, ‘Compiling SIMULA’, 29.

¹¹Ibidem 29

¹²Ibidem 30

¹³Ibidem 30

¹⁴Ibidem 29

uitbreiding van het softwarepakket van de UNIVAC zijn.

Maar voor het aanschaffen van SIMULA door UNIVAC zijn een aantal personen op de juiste plaats van groot belang geweest. De al eerder genoemde heer Bremer had als hoofd *systems programming* de mogelijkheid om vijf procent van acht miljoen dollar te besteden naar eigen goeddunken, en dus ook voor SIMULA¹⁵.

Later blijkt eens te meer dat de wil om SIMULA te verkrijgen voor SIMULA van enkele personen afhing. In de loop van de jaren zestig werd de communicatie tussen UNIVAC en het NCC steeds slechter¹⁶ Toen SIMULA klaar was en volgens het contract in januari 1965 opgeleverd werd UNIVAC hiervan op de hoogte gesteld, ‘but we [Nygaard en het NCC] got no reaction’¹⁷. De brief waarin UNIVAC op de hoogte werd gesteld bleek ergens in een ongebruikt bureau te liggen. Uiteindelijk, na persoonlijk contact tussen Nygaard, Dahl en Ira A. Clark en W.J. Raymond van UNIVAC, werd SIMULA op 10 januari geaccepteerd door UNIVAC.

3.3 De verhouding tussen machine en software

Tenslotte is er nog een andere reden aan te wijzen waarom UNIVAC geïnteresseerd kon zijn in SIMULA, in een simulatietaal. In onderstaande tabel (pagina 21) is duidelijk te zien dat de meeste simulatietaalen, en zeker de populairste, zoals GASP, GPSS en SIMSCRIPT, voornamelijk voor IBM machines geïmplementeerd waren.

General Characteristics of Various Contemporary Discrete-Change Simulation Packages				
Simulation Package	Computer Language	Originating Organisation	Machines Implemented	Availability
CLP	CORC	Cornell U.	CDC 1604	Current
CSL	FORTRAN	Esso, Ltd. & IBM U.K.	IBM 7090	Current
CSL 2	FORTRAN	IBM U.K.	IBM 7090/7094	In preparation
ESP	ALGOL	Elliott	Elliott 503 & 803	—
FORSIM IV	FORTRAN	MITRE	IBM 7030 & others with FORTRAN compilers	Current
GASP	FORTRAN	U.S. Steel Corp.	IBM 7040/7044, 7090/7090, 1620, 7070707 4, CDC G20	Current
GPSS	FAP	IBM	IBM 7090	Current
GPSS II	FAP	IBM	IBM 7090/94, IBM 7040/44	Current
	FORTRAN	UNIVAC	UNIVAC 1107	Current
GPSS III	MAP	IBM	IBM 7090/7094, IBM 7040/7044	Current
	—		IBM S/360	In preparation
GSP 2	—	U.S. Steel Co. Ltd.	Ferranti Pegasus	Current
			Elliott 503	In preparation
Job Shop Simulator	—	IBM-GE	IBM 7090	Current
MILTRAN	—	Systems Research Group for Office of Naval Research	IBM 7090/7094	Current
OPS	—	MIT	IBM 7090/7094	Current
QUICKSCRIPT	—	Carnegie Inst. of Technology	CDC G20	Current
SILLY	—	U.S. Steel Corp.	—	In preparation
SIMON	ALGOL	Bristol College of Sci. & Tech.	Elliott 503 & 803	—
SIMPAC	SCAT	Systems Development Corp.	IBM 7090	Notavailable
SIMSCRIPT	FORTRAN	RAND (SHARE)	IBM 7090/7094, 7040/44	Current
	FORTRAN	California Analysis Corp.	IBM 7090, 7090/7094, 7040/7044 CDC3600, 3800, 6400, 6600, 6800	Current
			Philco 210, 211, 212; UNIVAC 490, 1107, 1108	
		Digitek	GE 625/635	Current
SIMTRAN	FORTRAN	MITRE	IBM 7030	Current
SIMULA	ALGOL	Norwegian Computing Center	UNIVAC 1107	In preparation
SOL	ALGOL	Burroughs, Case Inst. of Tech.	Burroughs B5000/5500	—
			UNIVAC 1107	Current
UNISIM	FAP	Bell Labs.	IBM 7090/7094	Current

Tabel: Algemene karakteristieken van verschillende discrete event simulatieprogrammeertalen rond 1966.¹⁸

¹⁵Ibidem 30

¹⁶Dahl, ‘The Development of the SIMULA Languages’, 256.

¹⁷Ibidem 256

¹⁸Integraal overgenomen uit Daniel Teichroew en John Francis Lubint, ‘Computer Simulation - Discussion of the Technique and Comparison of Languages’, *Communications of the ACM* (october 1966) vol. 9 nr. 10., 723 - 741, aldaar 726. ‘Current’ in de tabel is 1966.

In de tabel zijn maar vier talen te vinden die voor de UNIVAC geïmplementeerd zijn, te weten GPSS II, SIMSCRIPT, SIMULA en SOL. Voor UNIVAC was het dus interessant om een simulatietaal te hebben die potentiële klanten over zou kunnen halen om een UNIVAC aan te schaffen. Zeker in Europa zou SIMULA, maar ook SOL, beide op ALGOL gebaseerd, een goede indruk maken, en dus ook de UNIVAC.

Het feit dat talen en software speciaal voor bepaalde computers gemaakt werd, is een belangrijk gegeven. Hierdoor werd software en hardware een afhankelijke eenheid. Bij de redenen om bepaalde hardware aan te schaffen speelde de meegeleverde software een belangrijke rol. Maar ook andersom, bij de redenen om bepaalde software aan te schaffen speelde de hardware ervoor ook een belangrijke rol.

Deze verhouding tussen hardware en software kan een van de motieven zijn geweest voor UNIVAC om SIMULA te laten ontwikkelen. Nadeel is wel dat het gebruik van SIMULA voor een groot deel afhankelijk was van het bezit van een UNIVAC 1107 computer. Naast UNIVAC machines werden ook de URAL 16 machines van Sowjet makelij platformen waar SIMULA op draaide. Er werd voor die machine door de Sowjets een compiler geschreven en gebruikt¹⁹ Zelfs de handleiding van SIMULA werd vertaald naar het Russisch.

Toch bleef de verspreiding van SIMULA door de innige verhouding tussen hardware en software beperkt. Misschien is dit een indicatie dat de simulatietaal niet echt aansloeg. De grote populaire talen, en voornamelijk SIMSCRIPT kende namelijk veel implementaties op verschillende machines. Waarom de taal niet aansloeg kan meerdere redenen hebben. Een ervan zou de basis ALGOL 60 kunnen zijn. Een ander de implementatie op een UNIVAC machine, een implementatie voor IBM zou misschien de gebruikersgroep sterk vergroot hebben.

3.4 Conclusie

De hoofdvraag van dit hoofdstuk, wat zijn de motieven achter de UNIVAC-overeenkomst, is beantwoord nadat er eerst een korte beschrijving van de tot standkoming van de overeenkomst is geschets.

De motieven van UNIVAC lagen vooral op het commerciële vlak. De verkoop van UNIVAC computers in Europa moest gestimuleerd worden, en een Europese op ALGOL 60 gebaseerde taal zou daarbij kunnen helpen.

Daarbij wilde UNIVAC graag een demonstratieplaats in Europa en NCC leek daarvoor erg geschikt. Een ander motief was het feit dat UNIVAC SIMULA het meest geschikt achtte voor eigen gebruik.

Enkele personen bij UNIVAC speelden een belangrijke rol, zeker de directeur *systems programming*, die vijf procent van acht miljoen dollar kon besteden naar eigen goeddunken, was erg gecharmeerd van SIMULA. Hij wilde namelijk het gebruik van ALGOL 60 stimuleren, en SIMULA daartoe bijdragen, dacht hij.

Later bleek dat UNIVAC minder interesse in SIMULA stelde, er waren andere personen aan het bewind gekomen in UNIVAC, en de communicatie tussen NCC en UNIVAC verliep slecht. Het opleveren van SIMULA werd zelfs over het hoofd gezien.

¹⁹Dahl, 'The Development of the SIMULA Languages', 257.

Tenslotte kan een motief gezocht worden in de verhouding van software en hardware. In de jaren zestig van de vorige eeuw was die verhouding erg innig, software werd speciaal voor bepaalde hardware geschreven, en zodoende kan de keus voor SIMULA van UNIVAC gezien worden in dat licht: een eigen nieuwe veelbelovende simulatietaal voor de UNIVAC zou een reden voor potentiële computerkopers kunnen zijn om voor de UNIVAC te kiezen.

Hoofdstuk 4

Van Simula naar Simula 67

4.0 Inleiding

1965, SIMULA is geïmplementeerd en wordt steeds meer gebruikt. Tijdens het eerste levensjaar van SIMULA besteden Kristen Nygaard en Ole-Johan Dahl hun meeste tijd met het onderwijzen van hun taal. Gaandeweg dat jaar raakten beiden ook steeds meer geïnteresseerd in de toepasbaarheid van SIMULA als een algemeen toepasbare programmeertaal¹.

Een andere reden om de overstap te maken van een speciale simulatieprogrammeertaal naar een algemeene programmeertalen werd door de beide heren niet gegeven. Nietemin gingen ze al snel aan de slag met de eerste ideeën voor wat later SIMULA 67 zou gaan heten.

Om te bepalen of SIMULA overbodig was, is interessant te kijken of deze verandering van SIMULA naar SIMULA 67 een logische stap was. Dat wil zeggen, betekende de overstap van een speciale simulatietaal naar een algemene programmeertaal dat de ideeën achter SIMULA overboord gegooid werden of was het een voortbouwen op die ideeën?

Voordat die vraag beantwoord gaat worden, komt eerst de ontwikkeling van SIMULA naar SIMULA 67 aan bod in een korte beschrijving van dat proces. Uiteindelijk, in de conclusie, wordt de vraag van dit hoofdstuk beantwoord.

4.1 Van SIMULA naar SIMULA 67

De eerste verwijzing naar de nieuwe SIMULA dateerde al van september 1965, maar al daarvoor werd gebrainstormd over de zwakke punten van SIMULA. Nygaard en Dahl noemden deze ideeën nog academisch, omdat ze voornamelijk gezien moesten worden als: ‘What should be revised if a revision was to be made.’²

Die herziening werd echter al snel een reële optie. De technische universiteit van Noorwegen in Trondheim nam in 1965 contact op met het Noorse rekencentrum over de mogelijkheid om een nieuwe ALGOL 60 te bouwen, geschikt voor de UNIVAC 1100 reeks. Deze nieuwe compiler zou van meet af aan ontwikkeld worden met de ondersteuning van SIMULA in het achterhoofd.³

¹Dahl, ‘The Development of the SIMULA Languages’, 257.

²Ibidem 258.

³Ibidem 258.

Dit leek een goede gelegenheid om de nieuwe ideeën over SIMULA werkelijkheid te laten worden. Te meer daar een van de zwakke punten van SIMULA de gebrekkige prestatie van de ALGOL 60 compiler van UNIVAC was, als het aankwam op grote aantallen processen. Deze compiler bleek dat wat betreft het geheugenbeheer niet efficiënt op te kunnen lossen. Omdat zulke grote aantallen processen bij serieuze simulaties gebruikelijk was, verminderde dit de bruikbaarheid van SIMULA.⁴

De andere zwakke punten van SIMULA, waarvan de meeste betrekking hadden op wat later objectgeoriënteerde aspecten genoemd zouden worden, zouden gezichtsbepalend voor SIMULA 67 worden. Nygaard en Dahl werden enorm geholpen door een artikel van C.A.R. Hoare, ‘Record handling’, waarin voorgesteld werd om records met recordklassen te beschrijven. Tevens introduceerde Hoare het idee van subklassen.⁵ Dahl en Nygaard hernoemden en breidden de *processes* en *activities* uit tot objecten en klassen.

Toch waren Hoare’s ideeën te strict, subklassen moesten binnen het blok van de hoofdklasse gedefiniëerd. Aan het einde van 1966 hadden Dahl en Nygaard hiervoor een oplossing gevonden: klassen en subklassen zouden onafhankelijk van elkaar gedefiniëerd moeten kunnen worden. Door middel van prefixing zouden zulke relaties tussen klassen duidelijk gemaakt kunnen worden⁶. Prefixing in dit verband is het aangeven van de voorouderstructuur door alle voorouders gescheiden door punten voor de huidige klassenaam te plaatsen.

Pas nadat dit concept van prefixing goed uitgekristalliseerd was, namen Dahl en Nygaard de beslissing om een volledig nieuwe algemeen toepasbare programmeertaal te ontwikkelen gebaseerd op prefixing, klassen, subklassen en objecten. Hierna was de tijd rijp voor een officiële uitwerking van deze ideeën.⁷

Deze uitwerking werd nog een haastklus omdat Ole-Johan Dahl in 1965 voorgesteld had om twee jaar later de IFIP⁸-conferentie over simulatietalen in Oslo te houden. Nygaard en hij wilde graag hun nieuwe ideeën presenteren op die conferentie, maar daarvoor moest het document wel voor eind maart verspreid zijn, wat ook lukte⁹.

Al in maart werd met Control Data onderhandeld over de implementatie van SIMULA 67, zoals de taal officieel na de conferentie genoemd werd. Op 23 mei 1967 was het zover, het Noorse rekencentrum en Control Data kwamen overeen dat Control Data zelf een SIMULA 67 compiler zou ontwikkelen voor de 6000 reeks en het voor de 3000 reeks zou een klant van Control Data, Kjeller Computer Installation zorgen.

Verder zou er een SIMULA 67 Common Base taaldefinitie komen, waarvoor in juni al een conferentie gehouden werd. Daarnaast zou er een nieuwe organisatie gecreëerd worden, de SIMULA 67 Standards Group. Tenslotte zou het rekencentrum de implementaties van Control Data begeleiden.¹⁰

Op de Common Base conferentie van juni 1967 werd de Common Base vast-

⁴Ibidem 258.

⁵Stein Krogdahl, ‘The birth of Simula’, *Proceedings of the HiNC 1 Conference in Trondheim* (juni 2003), 3.

⁶Ibidem 3

⁷Dahl, ‘The Development of the SIMULA Languages’, 259.

⁸International Federation for Information Processing

⁹Ibidem 259.

¹⁰Ibidem 266.

gesteld en de ontwikkeling ervan bevroren op de invoer/uitvoer- en stringbewerkingsmogelijkheden na. Op 10 februari 1986 werd de volledige SIMULA 67 Common Base Language vastgesteld en goedgekeurd door het eerder opgerichte SIMULA 67 Standaards Group¹¹.

Hierop kon de implementatie van de compilers beginnen. Al in de lente van 1969 waren de compilers van Control Data af. Omdat Dahl en Nygaard SIMULA 67 een overlevende taal wilden laten zijn, was het van belang dat SIMULA 67 beschikbaar zou zijn op alle belangrijke machines van die tijd, voor de UNIVAC 1100 reeks, de IBM 360/370 reeks en de PDP-10 van Digital Equipment werden daarom compilers vervaardigd. Begin jaren zeventig van de vorige eeuw werden ze voltooid, en met succes¹².

4.2 Was SIMULA 67 een logische opvolger van SIMULA

Een korte beschrijving van SIMULA 67 is in de vorige paragraaf gegeven, en duidelijk is geworden dat SIMULA 67 een taal is geworden van een heel andere orde dan SIMULA. Het is een algemeen toepasbare programmeertaal geworden, geschikt voor alle grote computers van eind jaren zestig, begin jaren zeventig. Heel anders dan SIMULA dat specifiek voor simulatie bedoeld was, en voornamelijk op de UNIVAC 1100 reeks draaide (alhoewel er later ook voor de Burroughs B5500 en URAL 16 compilers beschikbaar kwamen¹³).

Dat er grote verschillen tussen SIMULA en SIMULA 67 bestaan, behoeft geen verdere uitleg. Centraal in deze paragraaf staat de meer interessante vraag of SIMULA 67 een logische opvolger van SIMULA was. Een logische opvolger van een taal A is een taal die als basis A heeft, en daarop voortborduurde. Met andere woorden een taal die de basistaal verbeterd zonder daarbij goede functionaliteit te verliezen of te breken met veel conventies uit de basistaal.

In het geval van SIMULA 67 valt te betwijfelen of het een logische opvolger van SIMULA was. Al voordat daadwerkelijk met de revisie van SIMULA begonnen werd, in 1965, realiseerden Ole-Johan Dahl en Kristen Nygaard dat ‘SIMULA I’s simulation facilities were too heavy a burden to carry for a general purpose programming language.’¹⁴

Later, als het prefixing en object-klasse-idee uitgewerkt zijn, wordt besloten dat ‘We [Kristen en Dahl] would design a new general programming language, in terms of which an improved SIMULA I could be expressed.’¹⁵ Met andere woorden, SIMULA wordt volledig overboord gegooid om een algemeen toepasbare programmeertaal te maken. Nietemin blijft SIMULA 67 in staat om SIMULA te simuleren, zagezegd.

Aan de andere kant kan SIMULA 67 weer wel gezien worden als een vervolg van een opvallende eigenschap van SIMULA, namelijk de *processes* en *activities*. Alhoewel deze concepten op SIMULA 67 hernoemd worden tot objecten en klas-

¹¹Krogdahl, ‘The birth of Simula’, 3.

¹²Holmevik, ‘Compiling SIMULA’, 33, 34.

¹³Dahl, ‘The Development of the SIMULA Languages’, 264.

¹⁴Ibidem 258.

¹⁵Ibidem 259.

sen, is het in wezen wel hetzelfde idee: dat van een object met data en operaties op die data, geïnstantieerd op basis van een klasse.

SIMULA 67 gaat verder op het objectgeoriënteerde pad, overerving middels subklassen en prefixing werd geïntroduceerd. Achteraf, vanuit het standpunt van de huidige objectgeoriënteerde talen, lijkt dit een logisch vervolg. Net als het geval is met virtuele routines, dat zijn ontegenzeggelijk elementen van het objectgeoriënteerde paradigma.

Toch zijn het concepten die in SIMULA 67 voor het eerst in een taal verwerkt werden, maar zijn ze een logisch vervolg van SIMULA's *processes* en *activities* concept? Het antwoord is ja, Dahl en Nygaard zagen als een van de minpunten van SIMULA de observatie dat bij het maken van simulatieprogramma's *processes* vaak een aantal dezelfde eigenschappen en procedures hadden¹⁶.

Toch waren zulke *processes*, oftewel objecten, verschillende entiteiten waartussen onderscheid gemaakt moest kunnen worden. Voor veel klassehiërarchieën die in de huidige objectgeoriënteerde talen gemaakt worden geldt hetzelfde. Er is onderscheid tussen al die klassen, maar ze hebben toch ook weer een deel gemeen. Om met Nygaards en Dahls woorden te spreken: '(..) the idea of subclasses, somehow extended to apply to processes, might prove useful.'¹⁷

Blijft staan het feit dat SIMULA een simulatietaal was, en SIMULA 67 een algemeen toepasbare programmeertaal. Op zich is dat een onlogische stap, vanuit het perspectief van SIMULA gezien. Immers het idee van SIMULA was het maken van een taal om simulatieprogramma's en systeembeschrijvingen te maken, niet om allerhande programmeerproblemen op te lossen.

Daarentegen moet gezegd dat het idee om een algemeen toepasbare programmeertaal te maken pas kwam nadat bleek dat SIMULA daarvoor in de praktijk geschikt zou kunnen zijn. Dan is het niet vreemd, niet onlogisch, dat Nygaard en Dahl hun taal breder inzetbaar willen maken dan voor simulatie alleen.

Zou het echter niet mogelijk zijn geweest om toch SIMULA te verbeteren zonder het zijn van een pure simulatietaal op te geven? Terugkerend naar de kritiekpunten op SIMULA van Nygaard en Dahl, blijkt dat er enkel een punt is, namelijk het feit dat SIMULA's simulatiemogelijkheden het onmogelijk maakten dat SIMULA algemeen toepasbaar was.

Hieruit kan opgemaakt worden dat de keuze om SIMULA een algemeen toepasbare programmeertaal te maken er een was die niet noodzakelijk was voor het functioneren van SIMULA. Met andere woorden, SIMULA 67 volgt niet uit de redenen van SIMULA's bestaan.

Was SIMULA een oplossing voor een probleem dat Kristen Nygaard had in zijn werk in Operations Research. Namelijk een taal voor het beschrijven van systemen en het maken van simulatieprogramma's, een taal die Nygaard ontbeerde en daarom zelf ontwikkelde.

SIMULA 67 daarentegen was niet een oplossing, maar eerder een vervolg op de ontwikkeling van SIMULA. Nygaard en Dahl hadden door het ontwikkelen van SIMULA ideeën ontwikkeld voor een nieuwe programmeertaal, ideeën die uit zouden monden in SIMULA 67. SIMULA was een bijdrage aan *operational research*, en SIMULA 67 een bijdrage aan de informatica.

¹⁶Ibidem 258.

¹⁷Ibidem 258.

4.3 Conclusie

Op de vraag of SIMULA 67 een logische opvolger van SIMULA was, is het antwoord tweërlei. Aan de ene kant is SIMULA 67 een volledig nieuwe taal, waar SIMULA's mogelijkheden nog wel beschikbaar gesteld worden. Aan de andere kant zijn de vernieuwende mogelijkheden van SIMULA 67, klassen, objecten, prefixing en subklassen, wel een logische opvolger van het *processes enactivities* concept van SIMULA.

Met andere woorden, SIMULA 67 is geen logische opvolger van de simulatietaal SIMULA, maar wel van de taal SIMULA.

Hoofdstuk 5

Conclusie

De hoofdvraag van dit document is: *was SIMULA overbodig?* In de inleiding is aangegeven dat het antwoord op deze vraag gevonden kan worden door het beantwoorden van een aantal deelvragen. Voordat naar antwoorden op die deelvragen gezocht kon worden, is eerst een algemene introductie gegeven over simulatie, simulatieprogrammeertalen en *discrete event* simulatietalen in het bijzonder.

Simulatie is niets meer of minder dan het vergaren van informatie over een systeem. Niet door het systeem zelf te gebruiken, als bij experimenteren het geval is, maar door een model van het systeem te gebruiken. Simulatie is vooral bruikbaar wanneer experimenteren niet mogelijk is, bijvoorbeeld bij wapensystemen of systemen met levende wezens.

Computersimulatie is simulatie met behulp van een computerprogramma. Zo'n programma wordt geschreven in een programmeertaal, soms in speciale simulatieprogrammeertalen, soms in algemeen toepasbare programmeertalen. Dergelijke speciale simulatieprogrammeertalen kunnen volledig nieuw zijn, maar ook gebaseerd op al bestaande talen, bijvoorbeeld als een preprocessor of als een pakket van subroutines.

Er zijn twee verschillende soorten van computersimulatie te onderscheiden: analoge computersimulatie en digitale computersimulatie. De speciale simulatieprogrammeertalen voor digitale computersimulatie zijn onder te verdelen in twee klassen: *discrete event* simulatietalen en *continuous event* simulatietalen. SIMULA valt onder de *discrete event* simulatieprogrammeertalen, waarvan de bekendste GASP, GPSS en SIMSCRIPT zijn.

Na deze korte introductie in simulatie kwam de eerste deelvraag aan bod: in hoeverre voegde SIMULA iets toe aan de al bestaande of in ontwikkeling zijnde *discrete event* simulatietalen? De beantwoording van deze vraag is, nadat de ontwikkeling van SIMULA beschreven is, gegeven door de opvallende eigenschappen van SIMULA te bespreken.

Een van die opvallende eigenschappen is het feit dat SIMULA gebaseerd was op ALGOL, een taal die vooral in Europa populair was. Hiermee was SIMULA een van de vier simulatietalen die op die taal gebaseerd was, overigens was SIMULA hiervan de enige taal die in 1966 daadwerkelijk geïmplementeerd was.

Een ander opvallend punt was het concept van *processes* en *activities*, de voorloper van het concept van objecten en klassen. Dergelijke processen werden in quasiparallel uitgevoerd, een idee dat destijds bij meerdere wetenschappers opkwam. Ook het gebruik van ALGOL maakte SIMULA niet uniek. Verder gaven de makers van SIMULA zelf aan dat SIMULA beïnvloed was door SIMSCRIPT.

Met andere woorden enkel het *processes* en *activities* concept kan als echt vernieuwend gezien worden, wat betref simulatiefunctionaliteit was SIMULA eerder overbodig dan vernieuwend.

In het volgende hoofdstuk kwam de belangrijke overeenkomst tussen het NCC en UNIVAC aan bod. Door deze overeenkomst, goedkoop een UNIVAC 1107 voor implementatie van SIMULA, kon SIMULA werkelijkheid worden.

De redenen voor deze overeenkomst moeten vooral gezocht worden in commerciële motieven van de kant van UNIVAC. Univac wilde namelijk in Europa een demonstratieplaats creëren, en dachten het NCC dat toe. Verder wilden ze hun marktaandeel in Europa vergroten, met een taal gebaseerd op Algol, een in Europa populaire taal, dachten ze een goede zet te doen.

Verder wilde UNIVAC SIMULA ook voor eigen gebruik, ze vonden dat deze taal de meeste mogelijkheden bood die ze zelf nodig hadden. Tenslotte moet opgemerkt dat enkele personen aan het hoofd bij UNIVAC SIMULA en Algol een goed hart toedroegen en daardoor SIMULA konden laten ontwikkelen.

Al met al kan niet gezegd worden dat SIMULA nu door UNIVAC gekozen is om haar vernieuwende kwaliteiten. De keus voor SIMULA werd eerder ingegeven doordat er in Europa weinig simulatietalen waren ontwikkeld. SIMULA was nieuw, op ALGOL gebaseerd en van Europese origine, wat dat betreft was de taal dus niet overbodig.

De laatste deelvraag betreft de ontwikkeling van SIMULA naar SIMULA 67. Van een simulatietaal naar een algemeen toepasbare taal is een grote stap, en de vraag is dan ook of die grote stap veroorzaakt werd doordat SIMULA eigenlijk overbodig was.

Opvallend is wel dat enkel de mogelijkheid dat SIMULA algemeen inzetbaar zou kunnen zijn, Nygaard en Dahgl aanzette tot de ontwikkeling van SIMULA 67. Het grootste obstakel van SIMULA om algemeen toepasbaar te worden, was het feit dat het simulatiegericht was. Dit doet dan weer vermoeden dat SIMULA als simulatietaal eigenlijk overbodig was, maar dat de ideeën in de taal goed waren. Die ideeën werden dan ook verder uitgewerkt en zo ontstond de eerste object georiënteerde taal, voortbordurend op SIMULA, SIMULA 67.

Na de deelvragen behandeld te hebben, kan de hoofdvraag beantwoord. Was SIMULA overbodig? Aan de ene kant, jazeker, SIMULA voegde niets tot weinig toe aan de al bestaande simulatietalen wat betreft simulatie zelf. SIMULA kwam tot stand in een tijd waar vergelijkbare ideeën ook elders uitgewerkt werden. Verder waren de motieven van UNIVAC commerciëel van aard, niet omdat SIMULA nu zo'n mooie en vernieuwende taal was.

Verder wilden de makers van SIMULA zelf ook van de simulatie-aspecten in de taal af, ze wilden een algemeen toepasbare taal maken, en daarvoor waren die simulatie-aspecten een te grote ballast.

Aan de andere kant was er geen simulatietaal uit Europa gebaseerd op ALGOL, dus wast dat betreft stapte SIMULA in het gat dat daar was. Andere simulatietalen waren vaak gebaseerd op FORTRAN. Dit was dan ook een van de redenen van UNIVAC om voor SIMULA te kiezen, een van de punten waarop het dus niet overbodig was, de ALGOL-basis.

Toch is de conclusie dat SIMULA overbodig was niet ongerechtvaardigd, de achterliggende motivatie voor SIMULA was het maken van een simulatietaal. Dat

is gelukt, doch dergelijke talen bestonden al. De makers zelf zagen later ook meer in een algemene programmeertaal dan in een simulatietaal. De motivatie voor SIMULA was dus blijbaar al voor een groot deel verdwenen.

Dat neemt niet weg dat overbodig werk toch tot goede resultaten kan leiden. Door de simulatiehype van de jaren zestig ontstonden allerhande simulatietalen, waaronder SIMULA, die niet allemaal even nodig waren. Toch ontstond uit SIMULA het objectgeoriënteerde programmeren, als het ware als een bijproduct van de opkomst van computersimulatie.

Hoofdstuk 6

Literatuurlijst

- Dahl, Ole-Johan, 'The birth of object orientation: the Simula languages', <http://heim.ifi.uio.no/olejohan/birth-of-oo.pdf>.
- Dahl, Ole-Johan en Kristen Nygaard, 'The Development of the SIMULA Languages', *ACM SIGPLAN Notices* (1978) vol. 13 nr. 8, 245 - 272.
- Dahl, Ole-Johan en Kristen Nygaard, 'SIMULA — an ALGOL-based simulation language', *Communications of the ACM* ed: D.E. Knuth (September 1966) vol. 9 nr. 9, 671 - 678.
- Holmevik, Jan Rune, 'Compiling SIMULA: A historical study of technological genesis', *IEEE Annals of the History of Computing* (1994) vol. 16 nr. 4, 25 - 37.
- Krogdahl, Stein, 'The birth of Simula', *Proceedings of the HiNC 1 Conference in Trondheim* (juni 2003).
- Nance, Richard E., 'A history of discrete event simulation programming languages', *ACM Sigplan Notices* (maart 1993) vol. 28 nr. 3, 149 - 175.
- Sebasta, Robert W., *Concepts of programming languages* (vijfde druk; Boston 2002).
- Shannon, Robert E. , 'Introduction to simulation languages', *Winter Simulation Conference* (december 5-7 1977), 14 - 20
- Sklenar, Jaroslav, 'Simulation', <http://staff.um.edu.mt/jskl1/simul.html>, laatst bezocht 17 augustus 2004.
- Teichroew, Daniel en John Francis Lubint, 'Computer Simulation - Discussion of the Technique and Comparison of Languages', *Communications of the ACM* (october 1966) vol. 9 nr. 10, 723 - 741.
- Wikipedia, 'Computer simulation', http://en.wikipedia.org/wiki/Computer_simulation, laatst bezocht 18 augustus 2004.