LOCO

kompositie mikrowerelden in Logo[1],[2]

Peter Desain en Henkjan Honing

1 INLEIDING

Een ideaal systeem voor komputermuziek stelt een komponist in staat om op een direkte manier zijn of haar ideeën te realiseren. De meeste muzieksystemen beperken zich tot een bepaalde stijl en/of esthetika van muziek. Daardoor zijn deze systemen niet erg bruikbaar voor kreatieve doeleinden. Het is natuurlijk onmogelijk alle bestaande muzikale kriterea en regels te implementeren, zelfs wanneer alle huidige en toekomstige behoeften van te voren bekend zijn. Maar als men zich zich voorneemt om een *kompositie*systeem te maken, ipv een *muziek*systeem, wordt het geheel al een stuk eenvoudiger. Zo'n kompositiesysteem zou niet een implicite kijk op muziek moeten hebben, laat staan deze de gebruiker opdringen. In plaats hiervan zou de gebruiker zelf moeten kunnen bepalen welke technieken of stijlen gebruikt worden. Dit stelt de volgende eisen aan een ideaal systeem voor kompositie:

- *uitbreidbaar*: de gebruiker moet nieuwe funkties kunnen definiëren die dezelfde status en mogelijkheden hebben als de reeds bestaande, ingebouwde funkties.
- *modulair*: funkties moeten van elkaar geïsoleerd en beschermd zijn en moeten los van elkaar bestudeerd en gebruikt kunnen worden.
- *orthogonaal*: funkties moeten zo gemaakt zijn dat elke toekomstige uitbreiding steeds in het systeem past en gebruik kan maken van alle mogelijkheden.
- *naamgeving*: het koderen van muzikale parameters en objekten moet zo dicht mogelijk bij het normale gebruik zijn: bijvoorbeeld fortissimo, -1 dB of keihard ipv 255.
- *abstraktie*: er moeten families van gerelateerde objekten gedefiniëerd kunnen worden, die slechts verschillen in één of meer aspekten.

Deze ontwerp kriterea gelden voor elke algemene programmeertaal. En inderdaad, een goede programmeertaal zou toereikend kunnen zijn. Maar aangezien niet elke komponist een programmeur is, of wil worden, is dit niet erg realistisch. De huidige generatie van komputerkomponisten zal moeten toegeven dat het realiseren van ideeën in programma's zeer indirekt en onhandig is. Vooral met programmeertalen die gekozen zijn uit gewoonte of om hun snelheid, zoals Basic en FORTRAN. Maar er is een andere benadering mogelijk. Door het gebruik van een set krachtige gereedschappen in een geïntegreerde kompositieomgeving is de komponist verlost van ad hoc oplossingen. In speciale gevallen moet het echter wel mogelijk zijn de kracht van de programmeertaal te gebruiken, maar ook dan kunnen de gereedschappen een grote hulp zijn. Op deze wijze verdwijnt het onderscheid tussen programmeren en het gebruiken van al bestaande programma's. Om dit te kunnen realiseren hebben we een symbolische programmeertaal nodig zoals LISP (de voertaal van het moderne Artificiële Intelligentie onderzoek). Logo, een afgeleide van LISP, voldoet eveneens. Maar deze taal heeft daarintegen het voordeel dat het eenvoudig is om te leren, wat komponisten in staat stelt om direkt in Logo te programmeren. Een ander voordeel is dat het beschikbaar is op vele kleine komputers en dus een veel grotere gebruikersgroep kent. Een derde reden om Logo te kiezen was dat het ons in staat stelt zowel LISP-machines als Personal Computers te gebruiken. Terwijl het ontwikkelwerk gedaan kan

---

[1] Gepubliceerd als Desain, P., & Honing, H. (1987). LOCO, kompositie mikrowerelden in Logo. In V. Haars (ed.), *Leren met Logo*. 181-200. Nijmegen: Stichting Leren met Logo.

[2] Voor geluidsvoorbeelden zie http://www.hum.uva.nl/mmm/loco/

worden binnen de programmeeromgeving van de LISP-machine, kunnen de programma's daarna naar goedkopere komputers overgezet worden.

Onze programma's zijn niet in een objekt-georiënteerde taal geschreven, maar in een eigen gemaakte objekt-georiënteerde aanvulling op Logo. Er waren nog geen objekt-georiënteerde Logo's beschikbaar. Met de komst van dit soort Logo's zouden de programma's nog eleganter geschreven kunnen worden. We hopen tevens aan te tonen dat Logo niet alleen voor kinderen is.

2 ARCHITEKTUUR VAN KOMPOSITIESYTEMEN

Het doel van het opzetten van welke systeemarchitektuur dan ook is het opsplitsen van komplexe problemen in eenvoudiger delen. Zelfs al zijn systemen in de realiteit amper opdeelbaar, een systeemontwerper moet altijd zoeken naar plekken waar een kunstmatige verdeling of opsplitsing zo min mogelijk kwaad kan.

In het algemeen kan de produktie van muziek van twee kanten bekeken worden: de kant van de komponist en de kant van de interpretator. Of, in andere woorden, de kompositie en het instrument. De komponist maakt een kompositie aan de hand van regels, konventies, intuïties, en/of persoonlijke voorkeuren. Een kompositie kan resulteren in een partituur die bedoeld als de beste of één van de mogelijke representaties van zijn of haar kompositionele werk. Een partituur kan alles zijn wat overgebleven is van de kompositie fase, variërend van kodes om een synthesiser aan te sturen tot een traditionele gedrukte partituur. Deze partituur wordt, met een zekere vrijheid, gedekodeerd door de interpretator tot een hoorbare vorm,. In komputermuziek blijkt zo'n verdeling in drieën (kompositie, partituur en instrument) zeer handig en daardoor ook vaak toegepast. Er kunnen dan drie talen gedefiniëerd worden. De taal van het kompositiesysteem, de taal waarin de resulterende partituur is beschreven en de beschrijvingstaal van het instrument. Deze talen kunnen variëren tussen een algemene programmeertaal en eenvoudige gegevens representatie (bv het notenschrift).

3 ARCHITEKTUUR VAN BESTAANDE SYSTEMEN

Eind jaren 50, begin jaren 60 werd de eerste komputermuziek geproduceerd. Het vernieuwende van de eerste experimenten was het geven van komponeertaken aan de komputer. Dit was geheel nieuw in vergelijking met de al wat langere traditie van de elektronische muziek. De door de komputer geproduceerde partituur werd door musici uitgevoerd. Maar al snel werden de komputers krachtig genoeg om het geluid zelf te genereren. Dit was zo'n fascinerend aspekt van komputermuziek dat het zeker tien jaar duurde voor er weer een zekere interesse kwam in het kompositieproces (natuurlijk enige opvallende uitzonderingen daargelaten). Profiterend van de ontwikkeling van hogere en meer abstrakte programmeertalen werden er kompositiesystemen ontworpen. Deze kommuniceerden via de partituur met de hierboven genoemde klankopwekkende systemen.

Wij geven enkele voorbeelden van bestaande komputermuziek systemen en hoe zij vanuit dit gezichtspunt gezien kunnen worden:

- *Music V* was het eerste muzieksysteem dat een instrument beschrijvingstaal had. De sterke kant van Music V is dan ook deze taal en de partituur. Eén van de nadelen is dat de gebruiker verplicht is te werken met koncepten als noot, tempo, etc. Er is geen kompositiesysteem beschikbaar (Matthews 1969).
- *Project 2* is een komplex kompositieprogramma met het aksent op seriële kompositiemethodes. Het is een gesloten muzieksysteem dat niet aan persoonlijke voorkeuren aangepast kan worden. Het resultaat van het kompositionele werk wordt gepresenteerd in een partituur op noot nivo (Koenig 1970).
- *PLA* heeft veel meer vrijheid wat kompositietechnieken betreft en heeft ook de mogelijkheid om instrumenten te definiëren. Maar het hele systeem is geschreven rond het koncept van een noot en genereert een partituur op een zeer laag noot en instrument instruktie nivo (Schottsteadt 1983).

- *Pile* negeert het gegeven van een partituur. Ten eerste omdat het bedoeld is als een real-time systeem. Ten tweede omdat het ontworpen is als een "control language". De kompositietaal staat dicht bij de komputer (Berg 1979).

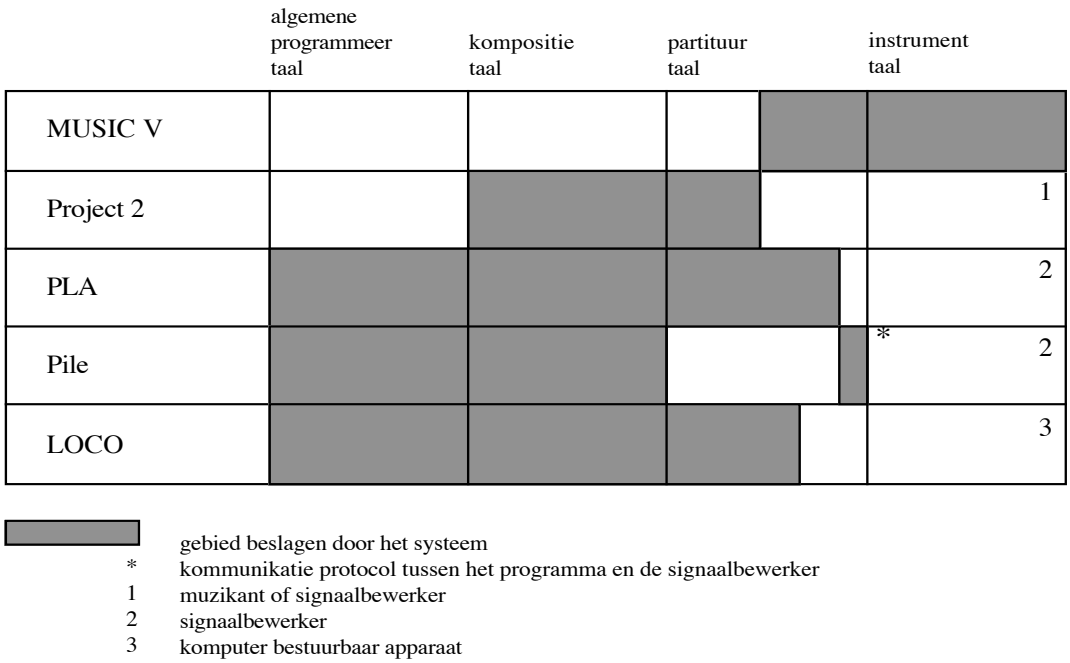


Fig.1. Verschillende muziek/kompositie systemen met elkaar vergeleken.

4 ARCHITEKTUUR VAN HET LOCO SYSTEEM

4.1 DE PARTITUUR

Bij het ontwerpen van LOCO was dan ook onze eerste taak het maken van een flexibele partituurtaal als kommunikatiemiddel tussen kompositie- en instrumentsysteem. Als de partituurtaal eenmaal krachtig genoeg is kan het gebruikt worden om er tussentijdse kompositie resultaten in uit te drukken.

Zoals gewoonlijk bestaat deze taal uit een reeks van primitieven (basis bouwblokken, woorden van de taal) en manieren om van kleinere bouwblokken grotere samen te stellen (de grammatikaregels). Bij het opzetten van een taal wordt men vaak verleid tot het definiëren van grote aantallen primitieven. Dit blijkt uit de gigantische lijsten van "features" die vaak genoemd worden in advertenties van programmeertalen. Maar juist het konstruëren van objekten uit kleinere objekten, op een algemene en krachtige manier, is essentiëel.

4.1.1 PRIMITIEVEN

Alles wat een instrumentsysteem zelf kan afhandelen (zoals bv een noot) is in principe een primitief muzikaal objekt.

Eerst zullen we wat voorbeelden geven van interessante instrumenten en hun primitieven.

Als het instrumentsysteem een interface is naar een mechanische slagwerkinstallatie moet het weten van zaken als materiaal, toonhoogte, plaats en kan het een primitief objekt als het volgende aangeboden krijgen:

[SLAG "kwart "metaal "derde.rek "hoog]

Dit lijkt op een subroutine aanroep (in prefix notatie) met als naam SLAG en een reeks argumenten die hier konstanten zijn (aangegeven met het Logo aanhalingsteken " dat staat voor: neem het volgende letterlijk, behandel het niet als een programma). Dit voorbeeld laat het gebruik van één van de primitieven zien die we gebruikten voor het instrumentsysteem dat *Ringo* bestuurde: een slagwerkinstallatie van Floris van Manen en Trimpin (Manen &Trimpin 1986). Het enige andere primitieve objekt dat nodig was is RUST:

```
[RUST "hele]
```

Voor andere instrumentsystemen zullen andere primitieve objekten gemaakt moeten worden. Om nog een paar voorbeelden te geven:

```
[FONEEM "kort "ooaah  "laag "luid]
[NOOT "achtste "gis  "staccato]
[BANDREKORDER "A.77  "neem.op  db -10]
[NOOT gepunkteerd "kwart  reine.stemming "c3]
```

De eerste zou gebruikt kunnen worden voor een foneemsynthesizer, de tweede voor een programma dat partituren afdrukt, de derde voor een afstand bestuurde geluidsstudio en de laatste voor een digitaal gestuurde oscilator. Let op het afwezig zijn van willekeurige kodering van de parameters. Het schalen van parameters en dekoderen kan zowel in de primitieven zelf (zoals in de eerste twee voorbeelden) als met de Logo funktieaanroep (zoals in de laatste twee voorbeelden) gedaan worden.

De enige vooronderstellingen die we zullen maken wat betreft primitieven is hun gedrag ten opzichte van de tijd. Zij hebben een bepaalde tijdsduur, maar er wordt niet gezegd *wanneer* zij plaatsvinden. De begintijd is geen parameter van het basis muzikale objekt. Een geluid nu, en hetzelfde geluid een tijd later beschouwen we als hetzelfde muzikale objekt. De tijdsduur van een primitieve is echter van te voren vastgesteld en bij zijn aanroep te bepalen.

4.1.2 TIJDSTRUKTURERING

In LOCO is een partituur een verzameling muzikale objekten plus hun tijdsrelaties. Om deze tijdsrelaties vast te stellen zijn er slechts twee tijdsordeningen nodig: twee dingen (muzikale objekten) gebeuren na elkaar of tegelijkertijd. De eerste tijdsordening noemen we S(equentiëel) de tweede P(arallel). Een partituur kan nu uitgedrukt worden worden als een hiërarchische struktuur van parallele en sequentiele objekten. Een voorbeeld:

```
[S thema improvisatie thema coda]
```

betekent dat er muzikale objekten zijn die *thema*, *improvisatie* en *coda* heten en die gekombineerd in een sequentie een stuk vormen. We kunnen een klein Logo programma maken om dit nieuwe muzikale objekt een naam te geven:

```
LEER jammen
 GEEFTERUG [S thema improvisatie thema coda]
EIND
```

Hier koppelt LEER de programma naam *jammen* aan het programma dat, wanneer het gestart wordt, het muzikale objekt [S thema improvisatie thema coda] oplevert. Op zijn beurt kunnen we nu thema definiëren als een gelaagde muzikale struktuur:

```
LEER thema
 GEEFTERUG [P baspartij pianopartij saxofoonpartij]
EIND
```

Dit maakt dat *thema* bestaat uit een bas-, piano-, en saxofoonpartij die allen op dezelfde tijd beginnen. Op deze manier kunnen partijen weer gedefiniëerd worden als opeenvolgende akkoorden, akkoorden als parallelle noten, enz. Als noten primitieve objekten zijn (bekend bij het instrumentsysteem) dan kunnen we hier stoppen. Een grafische representatie van het gedefiniëerde objekt *jammen* is weergegeven in fig. 2. In het partituursysteem kunnen S en P gezien worden als funkties, programma's die een tijdstruktuur kreëren. Zij kunnen echter tevens gezien worden als gegevens, die de tijdsrelaties tussen verschillende delen aangeeft. Deze aanpak, kennis representeren als programma en als gegevens, geeft het beste van de procedurele en de deklaratieve aanpak. Alleen met LISP-achtige talen is dit mogelijk. Het "process scheduling mechanism" van Formes (Rodet 1984) gebruikt dezelfde tijdstruktur erende funkties, maar zij kunnen niet als gegevens (data) bekeken worden. Het maken van transformaties op deze strukturen is dan ook onmogelijk.

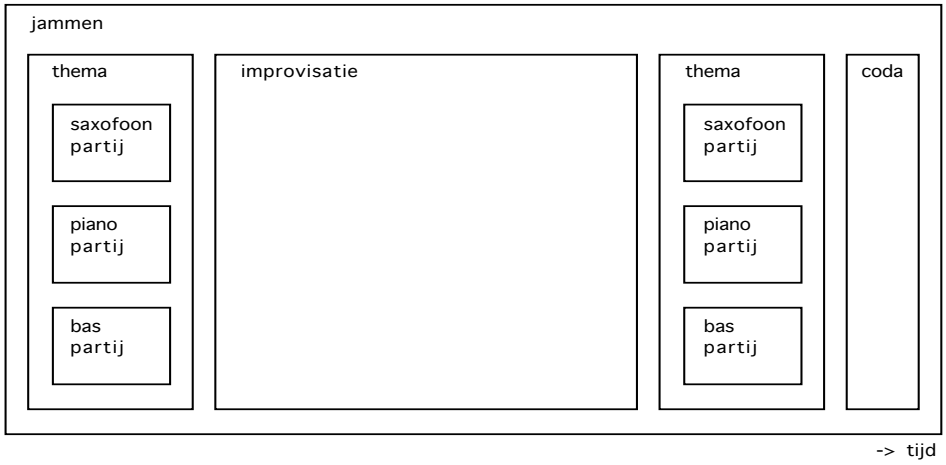


Fig.2. Tijdstrukturing van het objekt *jammen*.

Een kleine opsomming van de partituurtaal:

- er worden geen primitieven vastgelegd
- de tijdstrukturering wordt gemaakt met geneste S en P objekten.
- modulariteit en naamgeving worden gerealiseerd met Logo's funktie definitie mechanisme

4.2 DE KOMPONIST

Een kompositiesysteem moet gèèn implicite kijk op kompositie hebben. Het systeem zou de mogelijkheid moeten geven om gebruik te maken van verschillende technieken en stijlen van komponeren. Wij vermijden het maken van de keuze van wat de beste manier is om te komponeren. In plaats hiervan stellen we een reeks van kompositietechnieken beschikbaar die anders alleen in verschillende programma's en instituten te vinden zouden zijn. De gebruiker kan nu zelf bepalen waar hij of zij mee wil experimenten en kan dan eventueel zelf de keuze maken.

Zoals uit de inleiding blijkt zijn er vele manieren van komponeren. Het zou verwarring scheppen om ze allemaal naast elkaar te behandelen. Daarom is het beter om een aantal brede gebieden af te bakenen. Elk gebied is dan in een apart programmapakket onder te brengen, een verzameling gereedschappen waar met een bepaalde kompositieaanpak kan worden geëxperimenteerd. Omdat deze gereedschappen een min of meer kompleet of gesloten geheel vormen spreken we ook wel van

mikrowereld. Een voorbeeld van een mikrowereld, waar de meeste komponisten gebruik van maken, is de set gereedschappen pen, papier en gum.

4.2.1"*KOMPONEREN IS KEUZES MAKEN*"

Keuzes kunnen zowel door de komponist als door de komputer gemaakt worden. In deze mikrowereld kan de mate waarin de komponist kontrole heeft op het muzikale materiaal tussen brede grenzen worden gevariëerd. De komponist kan de komputer instruëren om volgens bepaalde regels keuzes te maken. Er kan gekomponeerd worden met mechanismen uit de stochastische en seriële muziek. LOCO laat zich niet uit over de muzikale waarde van de keuzes, dit moeten de komponist en de luisteraar beslissen.

4.2.1.1 DISKRETE KEUZES

In dit gedeelte wordt het maken van keuzes uit een afgebakende verzameling van mogelijkheden behandeld. Diskrete of gekwantiseerde keuzes reflekteren het koncept van schalen of ladders (zoals in dynamiek, toonhoogte of duur gebruikt worden) waar niet alle waardes toegestaan zijn.

Ons eerste mechanisme beschrijft de totale kontrole van de komponist. Dit keuzemechanisme heet KONSTANT en is gedegenereerd tot het maken van geen keuze.

```
KONSTANT "instrument "sitar
```

zal een programma *instrument* maken dat, als het gestart wordt, het woord *sitar* teruggeeft.

Een volgend keuzeprincipe is een eenvoudige aleatorische keuze, zoals vaak door komponisten gebruikt wordt.

```
TOEVAL "klankleur [hol vet transparant ruimtelijk]
```

De programma generator TOEVAL wordt gebruikt met twee argumenten. Het eerste argument is de naam van het te maken programma en het tweede is een lijst met mogelijkheden van de keuze, ook wel kansruimte genoemd. Nadat het programma *klankkleur* gegenereerd is kan het overal in een Logo programma gebruikt worden en zal het een toevallige keuze doen uit de aangeven klankkleuren. Natuurlijk kan de Logo interpretator gebruikt worden om de werking te testen.

```
? TOEVAL "klankkleur [hol vet transparant ruimtelijk]
? klankkleur
transparant
? klankkleur
hol
```

(het resultaat is kursief aangegeven en het vraagteken is de Logo prompt)

Het is belangrijk te weten dat noch TOEVAL, noch het programma *klankkleur* iets weten van de genoemde klankkleuren. Zij zien ze slechts als woorden, woorden die pas later worden geïnterpreteerd. Hierdoor is het mogelijk hetzelfde keuzeprincipe voor verschillende typen muzikale objekten te gebruiken.

```
TOEVAL "kompositie [deel1 deel2 deel3 deel4]
```

Het programma *kompositie* zal toevallig gekozen delen teruggeven.

Het gebruik van aleatorische keuzes voor toonhoogte in een melodie zal neigen naar wat genoemd wordt een witte melodie. Het ontbreekt aan struktuur of voorspelbaarheid, net als witte ruis. Een natuurlijke uitbreiding op TOEVAL is de mogelijkheid om verschillende kansen of weegfaktoren te koppelen aan de elementen van de lijst van mogelijkheden (kansruimte), zoals bijvoorbeeld bij een valse dobbelsteen (die meer kans heeft om een zes te geven dan één van de andere mogelijkheden). De programmagenerator WEEG voldoet aan deze beschrijving. Het heeft dezelfde argumenten volgorde als TOEVAL, behalve dat de elementen van de lijst van mogelijkheden samen met hun keuze kans in een lijst gegroepeerd zijn.

WEEG 'instrument [[kleine.trom 0,5][pauken 0,2][cymbal 0,3]]

In 50 procent van de gevallen zal *instrument kleine.trom* teruggeven. In 20 procent van de gevallen *pauken* en voor de rest het woord *cymbal*.

Soms is er een groot aantal numerieke elementen nodig op vaste afstanden, waaruit gekozen moet worden. Het zou erg onhandig zijn als ze allemaal in een lijst gezet moesten worden. Hiervoor is er de generator SCHAAL.

SCHAAL "puls 1/4 2 1/8

Het programma *puls* zal waardes in een schaal tussen 1/4 en 2 opleveren met een resolutie (raster) van 1/8. Bijvoorbeeld *0,375*(3/8), *0,5*(1/2), *0,75*(3/4), etc.

De keuzesprincipes tot nu toe hadden geen geheugen, dwz het resultaat van de keuzes werd niet beïnvloed door de reeds gedane keuzes. Nu zullen we keuze processen introduceren die dat wel doen. In plaats van de term "stochastische variabele" als referentie naar het resultaat van een keuze, moeten we nu de term "stochastisch proces" gebruiken. Ons eerste en bekende stochastisch proces heet SERIEEL. Elke keer als een keuze gedaan wordt wordt de lijst van mogelijkheden gekort met het gekozen element. Bij een volgende keuze wordt dit element uigesloten van deelname. Pas op het moment dat alle mogelijkheden uit de opgegeven lijst aan bod zijn geweest wordt de lijst van mogelijkheden weer in zijn oude toestand teruggezet. Dit is een generalisatie van het bekende twaalftoons keuzeprincipe.

Een voorbeeld:

SERIEEL "duur [achtste kwart zestiende zestiende]

Een ritmische struktuur gemaakt met dit programma staat afgebeeld in fig. 3.

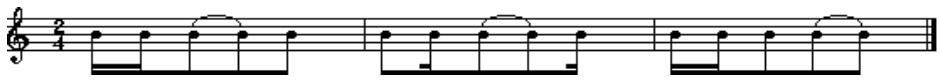


Fig.3. Struktuur gegenereerd met het programma *duur*.

Soms wil een komponist komplete kontrole hebben over de te maken keuzes. In dat geval kan OPVOLGEND gebruikt worden. De elementen van z'n kansruimte zullen altijd in een vaste volgorde teruggegeven worden.

OPVOLGEND "aksent [hard zacht licht geen]

zal opleveren: *hard zacht licht geen hard zacht* ... tot in het oneindige.

Er zijn verschillende manieren waarop primitieven, keuzes, en tijdsordeningen gekombineerd kunnen worden. We geven een paar voorbeelden. De argumenten van primitieven kunnen het resultaat zijn van een keuze programma.

SERIEEL "duur [hele halve kwart kwart]
TOEVAL "toonhoogte [c d f g a]
KONSTANT "dynamiek "pp

Het primitieve muzikale objekt

[NOOT duur toonhoogte dynamiek]

zal resulteren in een noot met een toevalige toonhoogte (uit de aangegeven pentatonische toonladder) en een toevallige duur (die deel uitmaakt van een seriële struktuur) in een konstant pianissimo.
Keuzes kunnen ook in andere keuzes besloten zitten zoals in het volgende voorbeeld:

OPVOLGEND "element [[NOOT duur toonhoogte dynamiek][RUST duur]]

Dit zal een elkaar afwisselende sequens opleveren van de hierboven gedefiniëerde noten en rusten.
Ook kunnen tijdsordeningen in keuzes besloten zitten en vice versa:

TOEVAL "struktuur [[P element element][S element element]]

De resultaten van keuzes kunnen bijvoorbeeld gebruikt worden in berekeningen als de volgende:

SERIEEL "afwijking [klein.positief.getal nul klein.negatief.getal]
[NOOT duur + afwijking ... ...]

Dit zou een eerste experiment kunnen zijn in het maken van een rubato.
Omdat bij het interpreteren van muzikale objekten steeds doorgeëvalueerd wordt tot er zinvolle resultaten zijn kunnen we keuzeprincipes laten kiezen uit keuzeprincipes, zoals in het volgende voorbeeld.
Een eerste stap is het maken van drie waarde generators.

TOEVAL "hoog [100 106 109]
TOEVAL "midden [10 15 9]
TOEVAL "laag [0 1 3]

Als we de te gebruiken generator uit deze programma's willen laten kiezen kunnen we het volgende doen:

TOEVAL "woord [hoog midden laag]
EVALUEER "waarde "woord

Het programma *waarde* zal het resultaat van *woord* evalueren. Het zal dus niet, zoals *woord hoog*, *laag* of *midden* opleveren, maar de waarde van één van de drie woorden. *Waarde* zal dus een getal als resultaat geven.

Het verbinden van keuzeprincipes in een netwerk kan gedaan worden met de OVERGANG programmagenerator. Eerst geven we de definitie van enkele keuzeprogramma's die de naam van een ander keuzeprogramma opleveren:

```
TOEVAL "janice [bedoelde zei wilde]
TOEVAL "bedoelde [dat]
TOEVAL "zei [dat]
TOEVAL "wilde [dat]
TOEVAL "dat [johan hans janice]
TOEVAL "johan [bedoelde zei]
TOEVAL "hans [zei wilde]

OVERGANG "uitleg "janice
```

OVERGANG heeft als eerste argument de naam van het te genereren programma. Het tweede argument is de start toestand. Elke keer als *uitleg* aangeroepen wordt zal het zijn huidige toestand teruggeven, beginnend met *janice*, en gebruikt die toestand om zijn volgende toestand te berekenen. Dus het aantal maal aanroepen van uitleg zal iets als het volgende opleveren: j*anice zei dat johan bedoelde dat hans wilde dat janice zei* ... Deze struktuur wordt ook wel een overgangsnetwerk genoemd. Het ziet er bekender uit in z'n grafische representatie (fig.4).

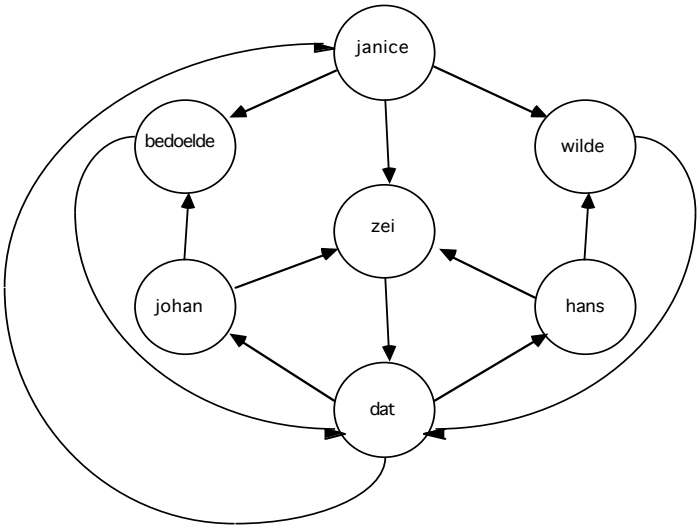


Fig.4. Overgangsnetwerk van het objekt *uitleg*.

Door het gebruik van WEEG ipv TOEVAL als de basis van het overgangsnetwerk kunnen de bekende, en in muziek zeer bruikbare Markov-ketens samengesteld worden. Kansen worden niet aan de kansruimte gekoppeld, maar aan de overgang van de ene gebeurtenis naar de andere. Dit krachtige gereedschap, waarmee alle diskrete stochastische processen zijn samen te stellen, kan weergegeven worden door het geven van kansen aan de verschillende overgangen. Op deze manier kunnen Markov-ketens in handzame gedeeltes opgesplitst worden. Dit in tegenstelling tot het opslaan van Markov-ketens in enorme matrixen, wat normaal gebeurt. In LOCO zijn dit soort overgangs netwerken echter zeer overzichtelijk toe te passen en kunnen er bijvoorbeeld ook andere keuzeprincipes (als OPVOLGEND en SERIEEL) gebruikt worden. Dit levert materiaal voor veel nieuwe muzikale experimenten.

Een manier om van een stochastische variabele een stochastisch proces te maken is het gebruik van de generator AKKUMULEER. Deze generator akkumuleert of integreert zijn vorige waardes.

```
TOEVAL "interval [1 2 3 -6]
AKKUMULEER "toonhoogte "interval 60
```

Op deze manier wordt een programma *toonhoogte* gemaakt dat de eerste keer dat het gebruikt wordt 60 zal teruggeven. Elke volgende keer zal het een waarde opleveren dat gevormd wordt door het optellen van 1, 2, 3 of -6 bij de voorgaande waarde. Met zo'n reeks kan een zgn bruine melodie gemaakt worden. Een bruine melodie kan vergeleken worden met een witte melodie. Ze zijn makkelijk van elkaar te onderscheiden. Er kan geëxperimenteerd worden met verschillende van deze "random walks" door de argumenten van AKKUMULEER te veranderen.
AKKUMULEER kan ook gebruikt worden voor het produceren van waardes die lineair veranderen.

```
AKKUMULEER "aftellen -1 10
```

aftellen zal inderdaad vanaf 10 aftellen.
Omdat in muziek vaak herhalingen gebruikt worden hebben we de programma generator ITEREER gemaakt.

```
ITEREER "herhalend "waarde 3
```

Als de gebruiker een programma genereerd of gedefiniëerd heeft met de naam *waarde* dan zal het programma *herhalend* dezelfde resultaten teruggeven als *waarde* alleen zal dan elke waarde drie keer herhaald worden voordat een volgende waarde aan bod komt. Met dit mechanisme is eenvoudig een Voss programma te maken dat een 1/f melodie genereert (Voss 1978).

```
SCHAAL "sprong 20 26 1
ITEREER "veel.sprongen "sprong 1
ITEREER "minder.sprongen "sprong 2
ITEREER "weinig.sprongen "sprong 4
```

Terwijl *veel.sprongen* elke ronde een nieuwe waarde (tussen 20 en 26) zal geven, geeft *minder.sprongen* elke twee rondes een nieuwe waarde (met hetzelfde bereik). *Weinig.sprongen* zal slechts om de vier rondes een nieuwe waarde opleveren.
We zullen de drie waardes optellen en ze als toonhoogte getal gebruiken.

```
[NOOT ... veel.sprongen + minder.sprongen + weinig.sprongen ...]
```

Eens in de zoveel tijd zal er een grote sprong in toonhoogte zijn omdat dan alle drie de waardes (in dezelfde richting) veranderen. Als er twee waardes veranderen, wat vaker gebeurt, zal er een kleinere sprong zijn. Maar het grootste deel van de tijd zal er een kleine sprong gemaakt worden omdat slechts één waarde verandert (fig.5). Een één-over-f melodie valt in het gebied tussen een witte en een bruine melodie, en wordt ook wel wel roze ruis genoemd. Het experimenteren en vergelijken met de andere processen wordt door het gebruik van onze open en simpele implementatie een stuk eenvoudiger en duidelijker dan bij het gebruik van Basic programma's (Dodge 1986). De mogelijkheden die onstaan door het maken van veranderingen en/of uitbreidingen zijn enorm. Bijvoorbeeld de herhalingswaardes 2 en 4 kunnen berekend worden, of ergens anders gebruikt

worden om een ritmische struktuur te maken die gerelateerd is aan de melodie. De SCHAAL generator kan door andere processen vervangen worden, etc.

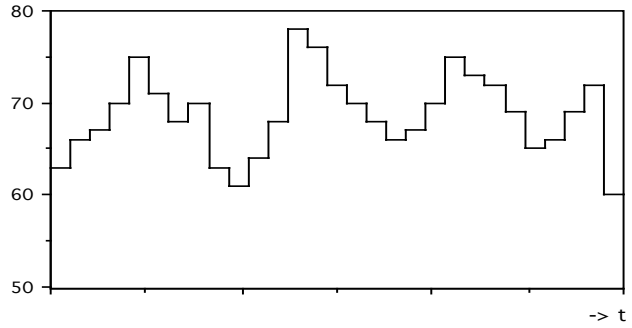


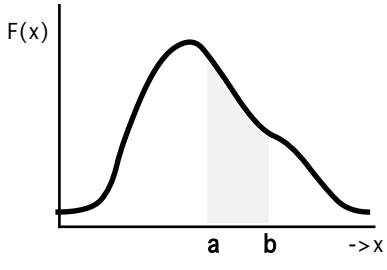
Fig.5. Toonhoogten geproduceerd met een Voss algoritme.

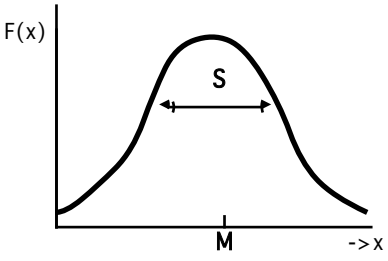
4.2.1.2 KONTINUE KEUZES

Schalen en toonladders werden (worden) zo veel gebruikt in traditionele muziek dat het voor sommige komponisten een harnas werd waar men vanaf moest zien te komen. In de elektronische muziek kon de vrijheid van het beschrijven van toonhoogte in Herz of Cents gebruikt worden. In de komputermuziek was het uitdrukken van parameters op een kontinue schaal nog eenvoudiger. Bij kansverdelingen van kontinue variabelen zijn een reeks van mathematische modellen beschikbaar. Zij zijn geïnspireerd op een nieuwe tak van muziek maken die "stochastische muziek" genoemd wordt. Iannis Xenakis beschrijft zijn werk in Formalised music (Xenakis 1971). Het belangrijkste gereedschap in het beschrijven van stochastische variabelen is de kansdichtheidsfunktie. Het laat de relatie zien tussen een interval en de kans dat een variabele binnen dit interval ligt. In fig 6.1 is een willekeurige kansdichtheidsfunktie getekend. De kans dat x tussen a en b valt is het (gearceerde) gebied onder F tussen a en b. Dit betekent dat alle kansdichtheidfunkties niet-negatieve funkties zullen zijn met een oppervlakte van 1. Een vaak gebruikte kansdichtheid is de Normaal of Gaussiaanse verdeling (fig. 6.2). Het wordt bestuurt door de gemiddelde waarde M (de positie van de piek in de funktie) en een afwijking S die aangeeft hoe breed de piek is. Er is ook een Uniforme distributie (fig. 6.3) waarin elke waarde in het bereik tussen een aangeven minimum en maximum dezelfde kans heeft voor te komen. In onze mikrowereld hebben we de programmagenerators NORMAAL en BEREIK geïmplementeerd die, gegeven deze kontinue distributies, een programma maakt dat waardes op levert volgens deze distributie.

NORMAAL "frekwentie.a 1000 200
NORMAAL "frekwentie.b 1000 2
BEREIK "dynamiek -80 0

Als het programma *frekwentie.a* aangeroepen wordt (in de Logo interpretator getypt wordt) zal het een waarde terug geven. De kansdichtheid van deze waarde is de Normaal verdeling met als gemiddelde 1000 Hz en als afwijking 200. Het programma *frekwentie.b* geeft waardes rond 1 kHz terug. Zij zullen echter meer in de buurt van de 1000 zijn als die geproduceerd door het programma frekwentie.a, dit komt door z'n kleinere afwijking. Het gegenereerde programma dynamiek zal waardes tussen -80 en 0 produceren. Zij hebben binnen het gebied van -80 tot 0 een gelijke kans om gekozen te worden.





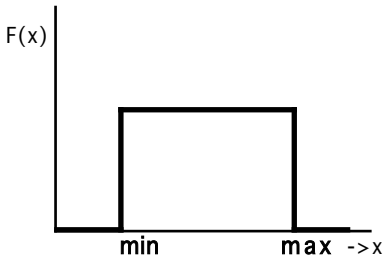


Fig.6.1 Willekeurige distributie funktie. Fig.6.2 Gaussiaanse verdeling. Fig.6.3 Uniforme verdeling.

Frekwentie.a en *frekwentie.b* zijn onafhankelijk van elkaar, dwz een waarde van de ene zal niet de keuze van een waarde van een andere beïnvloeden. Soms is dat iets wat we misschien wel willen. Bijvoorbeeld bij het produceren van toevallige toonhoogtes op twee instrumenten, waar als het éne instrument een hoge noot krijgt, de andere ook een tendens heeft om een hoge noot toebedeeld te krijgen. Wat we dan nodig hebben, om de relatie tussen twee stochastische variabelen vast te stellen, is een twee dimensionale distributiefunktie. We zullen er één beschrijven. De twee dimensionale Normaal verdeling heeft vijf parameters. Een gemiddelde voor de eerste variabele en een gemiddelde voor de tweede variabele. Ook zijn er twee afwijkingen nodig, voor elke variabele één. De laatste parameter, kovariatie, geeft aan hoeveel de variabelen met elkaar gekorreleerd moeten worden. Als de kovariatie hoog is kan de tweede waarde redelijk voorspeld worden als de eerste waarde bekend is. De tekeningen in fig.7 werden gemaakt door de waardes te tekenen die gegeven werden door de volgende programma's:

NORMAAL.2D "onafhankelijk 0 1 0 1 0
NORMAAL.2D "kleine.korrelatie 0 1 0 1 0,3
NORMAAL.2D "grote.korrelatie 0 1 0 1 0,9

Het is eenvoudig om dit proces te veralgemenen naar andere distributiefunkties en naar meerdere dimensies.

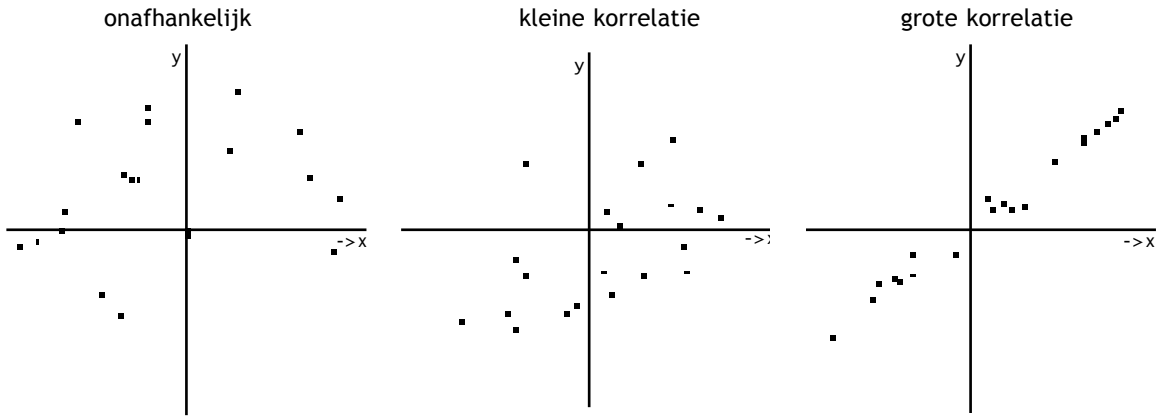


Fig.7. Het gebruik van twee dimensionale distributiefunkties met verschillende korrelaties.

Voordat we verder gaan met een diskussie over het stapelen van de verschillende kontinue keuzes, geven we een klein voorbeeld van het gebruik van een door een gebruiker op te geven kansdichtheidstabel.

TABEL "driehoek [[-1 0][0 1][1 0]]

De kansdichtheidsfunktie die door TABEL verkregen wordt door lineaire interpolatie is afgebeeld in fig. 8.1.
De kontinue distributies kunnen op verschillende wijzen gekombineerd worden met andere objekten. We kunnen bijvoorbeeld de parameters van een distributie variëren.

TOEVAL "gemiddelde [1 2 5]
NORMAAL "rond gemiddelde 1

De resulterende distributie van *rond* is afgebeeld in fig.8.2. Elke keer dat het programma *rond* gerund wordt zal het eerst waarde uit de kansruimte van *gemiddelde* opleveren. Deze waarde wordt dan op zijn beurt gebruikt in de Normaal verdeling van *rond*. Het zal resulteren in een waarde die rond 1, 2 of 5 ligt. Dit stapelen van toevallige keuzes maakt dat op een konceptueel eenvoudige manier stochastische variabelen gemaakt kunnen worden met zeer gesophisticeerde kansdichtheden.
Dezelfde methode kan gebruikt worden om in een paar regels de traditionele "tendency masks" te konstruëren.

AKKUMULEER "min -10 1
AKKUMULEER "max 20 -2
BEREIK "waarde min max

Het programma *waarde* zal waardes opleveren uit het toegestane gebied (van -20 tot 20) dat zich stap voor stap versmalt (fig.8.3). Het kombineren van de hiervoor beschreven bouwblokken geeft een enorme hoeveelheid aan mogelijkheden ("tendency masks" met interne, in de tijd veranderende distributies, distributies met in de tijd veranderende parameters, etc). Als we "tendency masks" op de traditionele manier gemaakt hadden (een vaste funktie met een aantal argumenten) dan waren alle andere mogelijkheden die met deze opzet mogelijk zijn uitgesloten en hadden we ons vaste komcept opgedrongen. In dit artikel proberen we aan te tonen dat juist het kombineren van kleine, maar krachtige mechanismen in elke mogelijk denkbare manier een veel betere aanpak is.

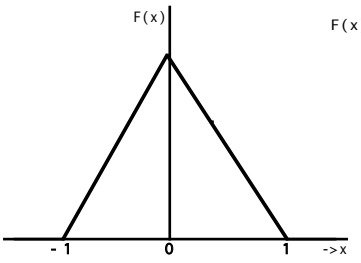


Fig.8.1. Distributie van *driehoek*.

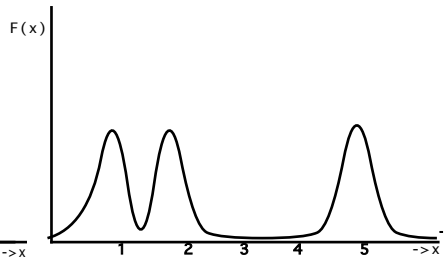


Fig.8.2. Distributie van *rond*.

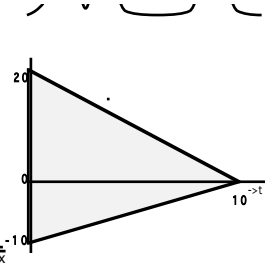


Fig.8.3. "Tendency mask" door *waarde* gegenereerd.

4.2.2 "*KOMPONEREN IS REGELS OPZETTEN*"

De theorieën van Chomsky hebben de linguistiek een enorme opleving gegeven. Nadat formele talen en grammatika's veel aandacht kregen in de musicologie werden ze gebruikt als een model om verschillende muzikale bereiken te beschrijven of om gebruikt te worden als een mechanisme voor nieuwe (komputer)komposities (Roads 1985). Toen we begonnen met het opzetten van een mikrowereld waarin met grammatika's geëxperimenteerd kon worden bleek dat we de kracht van ons systeem onderschat hadden. Het definieëren van grammatika regels bleek reeds mogelijk met LOCO.

Net zoals keuzeprincipes op elkaar gestapeld kunnen worden kunnen ze ook naar zichzelf refereren (Desain/Honing 1987).

TOEVAL "antwoord [lage.noot [S noot antwoord]]

Het programma *antwoord* zal een lage noot produceren of een noot gevolgd door een antwoord. Dit antwoord zal op zijn beurt wederom een lage noot produceren of een noot gevolgd door een antwoord. Op deze manier, als het systeem gevraagd wordt een antwoord te spelen, zal het een opeenvolging van noten spelen afgesloten met een lage noot. Merk op dat er een soort diepe evaluatie gebruikt wordt bij het interpreteren van muzikale objekten. We breiden de grammatika uit:

TOEVAL "dialoog [niets [S vraag antwoord dialoog]]
TOEVAL "vraag [hoge.noot [S noot antwoord]]
TOEVAL "antwoord [lage.noot [S noot antwoord]]

Deze kontekst-vrije grammatika zal een reeks van vraag-antwoord paren genereren. Natuurlijk moeten de programma's *noot*, *hoge.noot*, l*age.noot* en *niets* beschreven zijn. Deze programma's funktioneren als eindsymbool van de grammatika (zij worden niet verder door de grammatika geëxpandeerd).
Door het veranderen van het keuzeprincipe TOEVAL in WEEG, bij het opzetten van grammatika's, kan een zogenaamde gepogrammeerde grammatika opgezet worden. De individuele keuzekansen die aan de regels gegeven kunnen worden kunnen worden gebruikt om de gemiddelde grootte en het aantal keren dat de substrukturen geproduceerd worden te kontroleren. Het geven van een hoge keuzekans aan produktieregels met veel terminals zal kleine objekten opleveren, enz.
Nu een voorbeeld van en grammatika voor een ritmische struktuur die opgebouwd is uit inversie's.

TOEVAL "noot [niets
[S halve noot halve]
[S kwart noot kwart]
[S achtste noot achtste]
[S noot hele.rust noot]]

Dit programma kan bijvoorbeeld het ritme uit fig.9 genereren.

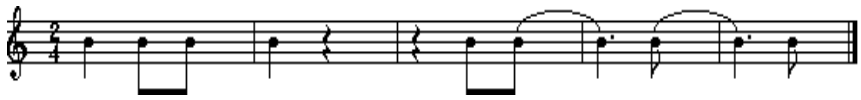


Fig.9. Ritme gegeneerd door het programma *noot*.

Grammatika's zullen over het algemeen zeer gestruktureerde muziek opleveren en daarom ook het suksesvolst als ze hiervoor gebruikt worden (Een voorbeeld van een grammatika voor het genereren van akkoord opéénvolgingen voor blues wordt beschreven in Steedman 1984). Omdat de mogelijkheid tot het maken van grammatika's verweven zit in de hele struktuur van LOCO kan het op allerlei manieren met de andere mechanismen gekombineerd worden. Merk op dat deze mikrowereld zich beperkt tot kontekst-vrije grammatika's

4.2.3 MEER MIKROWERELDEN

Als muzikale objekten in een uniforme en konsistente manier gerepresenteerd worden dan kunnen alle transformaties op alle vormen van muzikale objekten gebruikt worden. Wij zijn bezig met de

ontwikkeling van een algemeen transformatie mechanisme dat gebruikt kan worden voor alle kontekst-vrije transformaties op muzikale objekten. Het maakt efficiënt gebruik van funktionele argumenten, wederom een algemeen gebruik bij het programmeren in LISP (zie Henderson 1980). De mikrowereld rond dit mechanisme heet "Komponeren is het maken van variaties".

Patroonherkennings technieken lijken een opleving te hebben ondergaan sinds PROLOG het unifikatie algoritme (een soort patroonherkenning naar twee kanten) als zijn primaire interne mechanisme opnam. Wij ontwikkelen een mikrowereld gemaakt om te experimenteren met patroontransformatie technieken. Het heet "Komponeren is het transformeren van patronen" en kan gebruikt worden voor kontekst-gevoelige transformaties.

4.3 INSTRUMENT

De instrumentkant van het systeem kan niet (behalve voor bijvoorbeeld voor een grafisch apparaat dat een soort partituur kan produceren) geïmplementeerd worden in Logo door z'n real time vereisten. Voor de machines waar Logo nu op draait hebben we een instrument gedeelte ontworpen (in assembler, gelinkt aan Logo) dat zich gedraagt als een MIDI-recorder (of sequencer). Op deze manier kunnen MIDI gegegevens op de langzame produktiesnelheid worden opgenomen, om daarna op de vereiste snelheid te worden afgespeeld.

Een algemene aanpak voor het konstruëren van instrumenten zou een informatiestroom- of simulatietaal kunnen zijn. Het eerste programma van dit type was MUSIC-V. Maar programmeertechnieken van nu maken een veel gebruikersvriendelijke (modulair, grafisch) systeem mogelijk dat verbonden aan een reeks van van randapparaten, elk met zijn eigen kapaciteiten wat betreft snelheid en sophisticatie, zoals geluidschips in komputers, signaalbewerkers of DA omzetters. Dit valt echter buiten het onderwerp van dit artikel (zie Desain 1986).

5 HUIDIG ONDERZOEK

Er zijn nu implementaties van LOCO voor de Yamaha CX5-MII, gekoppeld aan de Nederlandse MSX-LOGO (Philips/LCSI) die de interne FM geluidschip gebruikt en een tweede versie is voor de Apple IIe, die gebruik maakt van Terrapin Logo (MIT) en een MIDI-interface (o.a. Roland en Jellinghaus). Een versie voor Macintosh LSCI logo is voorjaar 1988 gereed.

Verdere plannen zijn het implementeren van LOCO in LISP en het ontwikkelen van een grafische interface op LOCO, waardoor grafisch komponeren mogelijk wordt.

6 REFERENTIES

Allen, J.R. 1984. Thinking about Logo, a graphic look at computing with ideas. CBS. College Printing. New York.

Berg, P. 1979. Pile. Reprint in Foundations of computer music. Ed. C. Roads and J. Strawn, 1985. Cambridge, Massachusetts: MIT Press.

Desain, P. and H. Honing 1986. LOCO: Composition microworlds in Logo. Proceedings of the 1986 Computer Music Conference. ed. P. Berg. Computer Music Association, P.O.Box 1634, San Francisco, CA 94101.

Desain, P. en H. Honing 1987. LOCO: kompositie mikrowerelden in Logo. Leren met Logo, Nijmegen.

Desain, P. en H. Honing 1987. LOCO: een gebruikers handleiding. St. LOCO, postbus 1037, Utrecht.

Desain, P. 1986. Graphical programming in computer music, a proposal. Proceedings of the 1986 Computer Music Conference. Ed. P. Berg. Computer Music Association, P.O.Box 1634, San Francisco, CA 94101.

Dodge, G. and C. Bahn 1986. Musical fractals. Byte (6):185-196.

Fry, C. 1984. Flavors Band: A language for specifying musical style. Computer Music Journal 8(4):20-34.

Henderson, P. 1980. Functional programming: application and implementation. London: Prentice Hall.

Jones, K. 1981. Compositional applications of stochastic processes. Computer Music Journal 5(2):45-61.

Koenig, G.M. 1970. Project two. Electronic Music Reports 2. Utrecht: Institute of Sonology.

Manen, F. van, Trimpin. 1986 .Ringo: a percussion installation. Proceedings of the 1986 Computer Music Conference. Ed. P. Berg. Computer Music Association, P.O.Box 1634, San Francisco, CA 94101.

Mathews, M. 1969. The technology of computer music. Cambridge, Massachusetts: MIT press.

Papert, S. 1980. Mindstorms, children computers and powerful ideas. Harvester Press.

Roads, C. 1985. Grammars as representations for music. In: Fundamentals of computer music. Ed. C. Roads and J. Strawn, 1985. Cambridge, Massachusetts: MIT press.

Rodet, X and P. Cointe. 1984. FORMES: Composition and scheduling of processes. Computer Music Journal 8(3):32-50.

Schottsteadt, B. 1983. PLA: A composer's idea of a language. Computer Music Journal 7(1):11-20.

Steedman, M.J. 1984. A generative grammar for Jazz chord sequences. Music Perception (2) 1:53-77.

Voss, R.F. and J. Clarke. 1970. "1/F Noise" in music: Music from 1/F noise. J.A.S.A., 63(1):258-263.

Winston, P.H. and B.K.P. Horn. 1981. LISP. Addison Wesley Reading.

Xenakis, I. 1971. Formalised music. Bloomington, Indiana University Press.

241087 hh