Rijks*universiteit* Groningen **Faculteit der Wiskunde en Natuurwetenschappen**

Wiskunde en Informatica

PTS - cursus OOT

De homevox

Rein Smedinga

juli 2001

RuG

Inleiding

Als CASE-studie wordt in dit document de ontwikkeling van HomeVox programmatuur beschreven. Er wordt getracht zoveel mogelijk de OO-methodes te volgen zoals die op de cursus behandeld zijn. Omdat het een ingewikkeld probleem betreft, wordt gebruik gemaakt van 'incremental delivery'.

Requirements

Deze zijn beschreven in aparte documenten, die in hoofdstuk 1 zijn gebundeld. Ook over deze requirements kunnen al direct veel opmerkingen gemaakt worden, maar die proberen we uit te stellen totdat we tegen problemen aanlopen: op dat moment kunnen we om opheldering vragen.

Aanpak

Gevraagd wordt om de besturingsprogrammatuur van een HomeVox te ontwikkelen. Om de werking hiervan te tonen zonder over de bijbehorende hardware te beschikken, moeten we deze hardware simuleren. We proberen de gesimuleerde hardware zo te scheiden van de Homevox-besturing, dat we deze laatste eenvoudig naar de echte hardware kunnen overbrengen. Bovendien moeten we een (grafisch?) user interface maken om de simulator en besturing te laten werken.

We kunnen onze 'incremental delivery' op verschillende manieren aanpakken, bijvoorbeeld door met de hardware simulator te beginnen, of met het user interface. Om in een vroeg stadium al de mogelijke problemen in de requirements signaleren, is het waarschijnlijk verstandiger om met de functionaliteit van de HomeVox te beginnen. Dit zullen we doen op een hoog abstractieniveau, waarbij we ons bijvoorbeeld nog geen zorgen maken over zaken als de duur of frequentie van het belsignaal. Bovendien beginnen we met een erg eenvoudige HomeVox, zonder externe (PTT) aansluiting, en zonder faciliteiten als drieling-gesprekken, doorverbinden, enz. Geleidelijk voeren we de complexiteit op, totdat we de gehele functionaliteit beschreven hebben. Vervolgens proberen we de lagere abstractieniveaus te implementeren; een belangrijk probleem hierbij is de vormgeving van de simulatie van de hardware. Tenslotte zullen we ons richten op het user-interface.

In een omgeving met voldoende mankracht kan deze ontwikkeling voor een groot deel parallel plaatsvinden: een team houdt zich bezig met de functionaliteit op het hoogste niveau, een team met de simulatie van de hardware, en een team met het user-interface. Het is op een gegeven moment nodig om de interfaces tussen deze onderdelen vast te leggen, maar dit hoeft niet noodzakelijk vooraf plaats te vinden.

Taal

Voor de implementatie gebruiken we vanzelfsprekend Java.

Indeling

Eerst behandelen we de requirements en de abstracte ontwikkeling van het onderdeel interne gesprek. Verder behandelen we de externe gesprekken waarbij we een tweetal alternatieve analyses geven. Als practicumopgave dient het onderdeel met externe gesprekken verder te worden uitgewerkt.

0.1 Notatie

Vanzelfsprekend wordt zoveel mogelijk de UML notatie aangehouden. Voor de collaboration diagrammen is de afwijkende notatie uit de Fusion methode voor object interactie diagrammen aangehouden. De afwijkingen betreffen het feit dat volgnummer, conditie en identificatie bij een message niet achterelkaar, maar nu onder elkaar staan en voorts dat de verbindingen van een pijl zijn voorzien, in plaats van een seperate pijl achter de identificatie van de message.

Inhoudsopgave

1
HomeVox requirements

Inleiding

Bedrijf S heeft een ruime ervaring met electronische producten op de transmissie-markt. Men denkt een gat in de markt gevonden te hebben. Een kleine huis(telefoon)centrale voor de consumentenmarkt. De electronica is al geruime tijd in prototype aanwezig, maar het schrijven van het "programmaatje" levert de nodige problemen op. Na eerst zelf een poging gedaan te hebben heeft S het uitbesteed aan een "goedkoop" software-house dat pretendeerde alles van OOA (Object Oriented Assembly) af te weten. Een desillussie rijker, ziet het bedrijf nu al concurrenten op de markt komen en begint haast te krijgen. Daarom wordt nu Cursisten b.v. gevraagd, deze opdracht uit te voeren. Uit de ervaring wijs geworden vraagt S eerst om een prototype (= simulatie) op een PC. Als deze opdracht naar tevredenheid wordt uitgevoerd, kan eventueel de opdracht gegeven worden om de software echt te implementeren op de target hardware.

1.1 Eisen

In deze sectie worden de eisen opgesomd, zoals door bedrijf S gesteld. Cursief en in kleinere letter is zo hier en daar een opmerking bijgeschreven. Dit zijn vragen, opmerkingen, suggesties e.d. die bij een eerste doorlezen van de gestelde eisen opkwamen.

1. De HomeVox doet dienst als een kleine huiscentrale.

2. Gebruikers (maximaal 4) kunnen elkaar en ook naar buiten bellen.
Er zijn kennelijk verschillende soorten gesprekken (zie ook verderop). Kan een intern gesprek plaatsvinden tegelijk met een extern gesprek? Wat betekent dit voor faciliteiten als ruggespraak? (Tijdens ruggespraak PTT-intern afschakelen?)

3. Gebruikers zijn gelijkwaardig, hetgeen betekent dat bij binnenkomende gesprekken alle toestellen worden gewekt.

4. Het is wenselijk dat gebruikers aan het ringsignaal kunnen horen, of er intern of extern wordt gewekt.
Een toestel wekken = de bel over laten gaan.

Ook in dit geval moeten we voldoen aan de 25%-eis m.b.t. de duty-cycle van het ring-signaal (zie pagina 4).

5. De HomeVox moet zowel op moderne electronische centrales (DTMF – dual tone multi-frequency) als op oude mechanische (zowel registerhoudende als registerloze) centrales kunnen worden aangesloten.
De HomeVox bepaalt bij power-up of de openbare telefooncentrale een impuls- of een DTMF-centrale is.
Het begrip registerhoudend mag men koppelen aan moderne (DTMF) centrales en het begrip registerloos aan de "oude" mechanische centrales. Wat betekent dit onderscheid voor de HomeVox?

6. Ongeacht het type centrale moeten zowel puls- als DTMF-(toon)toestellen aangesloten kunnen worden.

7. Noodzakelijke faciliteiten:

- intern gesprek
- extern gesprek
- nummerherhaling extern gesprek
- drieling gesprek
- drieling gesprek buitenlijn
- ruggespraak
- gesprek doorgeven.

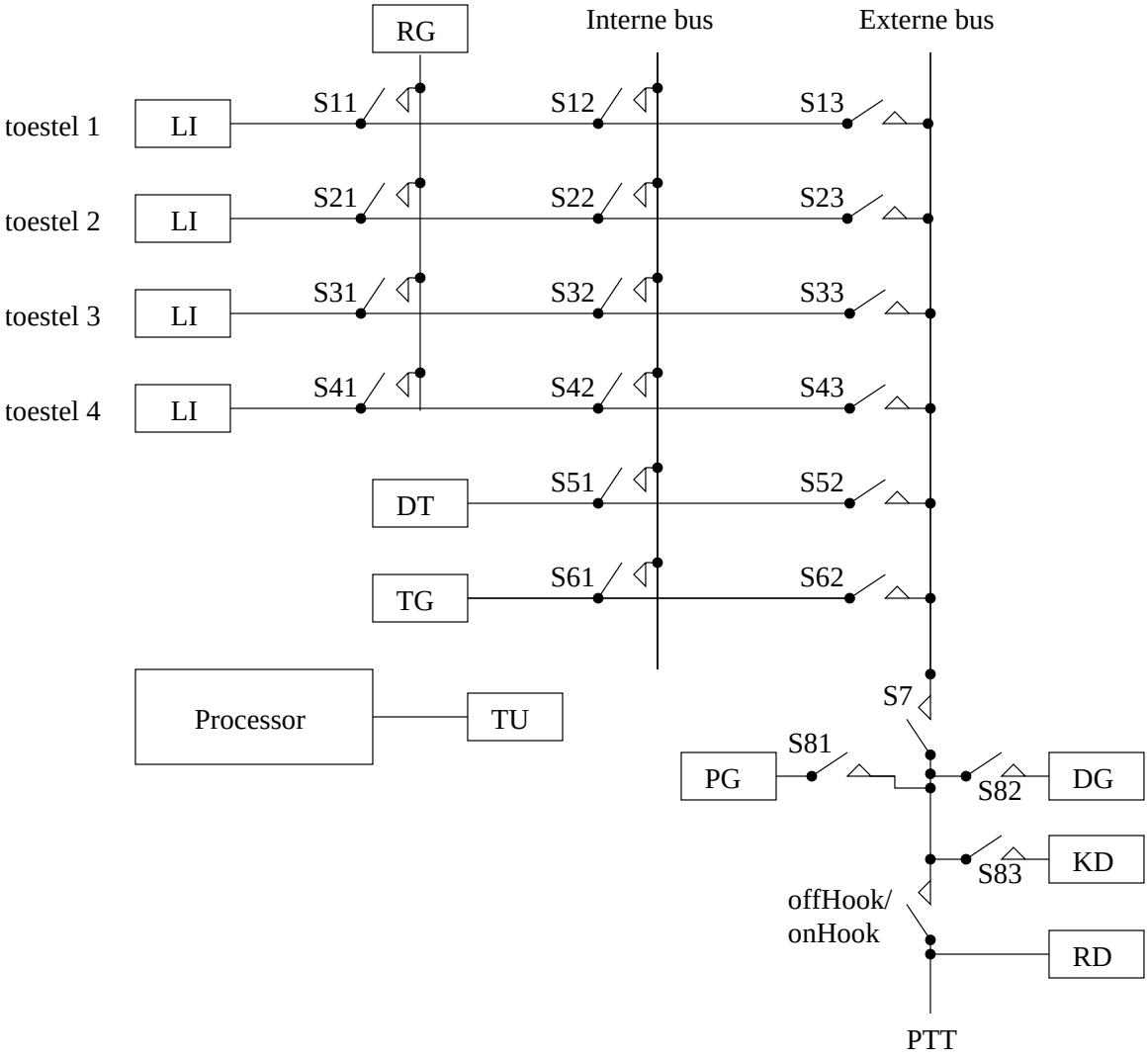
Kiespulsen (van kiesschijf of IDK-toestel) voor het oproepen van een intern toestel moeten worden onderdrukt naar de buitenlijn. DTMF signalen mogen in deze situatie wel tot de buitenlijn worden doorgelaten.
Gesprek binnenlijn: binnen 15 seconden na het laatste cijfer, worden pulsen/DTMF-signalen niet als HomeVox commando's beschouwd en normaal doorgegeven aan het openbare telefoonnet.
Meldt er zich tijdens een binnenlijn gesprek een buitenlijn (attentiesignaal + overgaan op de vrije toestellen) en wordt deze beantwoord op een vrij toestel, dan wordt het binnenlijn gesprek afgebroken.

8. Nummerplan:

0 – naar buiten bellen/uitschakelen ruggespraak (geldt voor beide partijen)[1]
1–2–3–4 – nummers toestellen
1–2–3–4 – ruggespraak
5 – herhalen laatste nummer (van toestel)
0–9 – extern bellen

9. Zoals uit het nummerplan blijkt, hebben sommige cijfers een dubbelrol. De actuele rol kan afgeleid worden uit de status van de centrale en/of een time-out.
Inschakelen drieling gesprek: vanuit de ruggespraak situatie het gekozen toestelnummer nogmaals kiezen.
Functie 0: naar buiten bellen/uitschakelen ruggespraak, neerleggen is tevens het doorgeven van het gesprek, in ruggespraak situatie geldt dit beide partijen.

[1] bij binnenkomend gesprek niet nodig



Figuur 1.1: Blockschema van de HomeVox hardware

1.2 HomeVox hardware

Een blokdiagram van de hardware is gegeven in figuur 1.1. Toelichting:

IDK – Impuls druktoets keuze
Toestel met kiesschijf of met druktoetsen en puls-generatie.

TDK – Toon druktoets keuze
Toestel met druktoetsen en toon-generatie.

LI – Line Interface.
Koppeling van de (gebruikers-)toestellen, geeft een interrupt aan de processor:
– onhook
– offhook
– start dialing digit (puls)
– digit dialed (digit) (puls)

De LI kan ook (indirect) kiestonen genereren, via het toestel.

De LI moet ook signalen (events) vanuit het de HomeVox kunnen verwerken, bijvoorbeeld: ‘ring’.

Voorzover ik iets van telefonie afweet, wordt voor ‘pulse dialing’ dezelfde signalering gebruikt als voor onhook/offhook. Hoe zit dat hier precies?

In de Req. is sprake van ‘start dialing digit’ en ‘digit dialed’; betekent dit laatste: ‘end dialing digit’? Is hier sprake van een compleet digit (zo ja: welk?), of is dit slechts ‘puls’ van een digit? Hebben we informatie nodig over de timing van pulsen?

Gezien de andere requirements moet per LI het laatst gebelde (externe) nummer bijgehouden worden.

RG – Ring Generator.

Genereert continue een ringsignaal (voor rinkelen toestel). Het gegenereerde vermogen is beperkt. Er kan dan ook maar één toestel tegelijkertijd gewekt worden.

In combinatie met de eis dat alle toestellen gewekt (=gebeld?) dienen te worden bij een externe oproep, betekent dit dat de duty-cycle van het externe oproep-signaal $< 25\%$ is, of komt het vermogen dan van de buitenlijn?

Processor.

Het “hart” van het systeem beschikt over verschillende interrupts.

Wat betekent dat: ‘beschikt over verschillende interrupts’?

DT – Dual Tone

DTMF detector kan DTMF-tonen detecteren. Geeft een interrupt op detectie met een indicatie welk DTMF woord (0–9) gekozen is. De DTMF detector negeert # en *.

Deze detector is in het blockschema via een schakelaar verbonden met de interne/externe bus; zou dit ook een permanente verbinding kunnen zijn?

Deze detector is ook verbonden met de externe bus; waarvoor? Ruggespraak tijdens extern gesprek? Is het mogelijk dat er een intern/extern gesprek aangevraagd wordt terwijl er een extern/intern gesprek bezig is?

TG.

Toongenerator kan genereren:

– kiestoon

– bezettoon

– vrijtoon (ander toestel belt)

– attentiesignaal (oproep van buitenlijn)

In de wachtstand wordt op de buitenlijn geen toon gezet.

Uit de genoemde tonen blijkt dat een intern gesprek ‘onderbroken’ kan worden door een attentiesignaal van een buitenlijn-oproep. (Moet de bel overgaan bij de toestellen die niet bij dit interne gesprek betrokken zijn?)

Wat is de ‘vrijtoon’? De toon die je als beller krijgt als indicatie dat de bel aan de andere kant overgaat?

Welke timing-eisen zijn er m.b.t. deze tonen? (bijv. de bezet-toon heeft normaliter een duty-cycle $< 100\%$.) Verzorgt de TG zelf deze timing, of moet deze aan- en uitgezet worden (resp. afgeschakeld)?

PG – Puls Generator.

Kan een cijfer met pulsen uitzenden, geeft een interrupt als hij klaar is.

‘cijfer’ en ‘digit’ zijn ’t zelfde, neem ik aan.

DG – DTMF-generator.

Kan een cijfer met DTMF-toon uitzenden. Moet aan- en uitgezet worden. De DTMF generator geeft minimaal een 55 msec toon – 55 msec pauze signaal, bijvoorbeeld bij nummerherhaling extern gesprek.

Bij het "aanzetten" opgeven welke toon gegenereerd moet worden?
Uitzetten is misschien overbodig: dit kan door de verbinding met de generator te verbreken, analoog aan de andere toon-generatoren.

CI – Centrale Interface
Is in de figuur opgeplitst in de onderdelen S83, KD (kiestoon detector), RD (ring detector) en offhook/onhook.
– Kan een lijn naar de centrale beleggen of niet.
– detectie ringsignaal (interrupt bij start- en interrupt bij stopsignaal)
– heeft dial-toon detector (interrupt wanneer gedetecteerd)
Het ringsignaal is waarschijnlijk afkomstig van de PTT-verbinding.
'dial-toon' is vermoedelijk: kiestoon (detectie van kiestoon afkomstig van de (PTT) centrale).

TU – Time Unit.
Elke 55 msec een interrupt.
Wat zijn de timing-eisen van het systeem? Is de TU de enige tijdsreferentie?
Je kunt deze timer mischien ook gebruiken om de duur van (externe) gesprekken bij te houden.

Interne bus.
Alle toestellen verbonden met deze bus kunnen met elkaar praten.
Gesuggereerde afkorting: IB (en EB, externe bus)

Externe bus.
Alle toestellen verbonden met deze bus kunnen met elkaar praten en eventueel ook met de buitenlijn.

Schakelaars
S1 t.e.m. S83 staan allemaal onder procesbesturing.

1.3 Simulatie-eisen

Voordat de software op de hardware wordt geïmplementeerd, is het wenselijk om een simulatie op een PC te maken.

Hierbij dienen minimaal 3 toestellen (naar keuze draaischijf of druktoetsen) elk in een (al of niet grafisch) window te worden geplaatst. De gebruiker kan met een muis de verschillende user actions uitvoeren en krijgt ook een normale terugmelding, zoals toonindicatie, ringsignaal, etc. Een extra window wordt gevormd door de centrale interface. Hier wordt de gebruiker de op elk moment mogelijke keuzes voorgeschoteld waaruit hij met de muis kan kiezen.

Optioneel: een window, waarin de toestand van alle schakelaars van de hardware is gegeven.

1.4 Algemene vragen en opmerkingen

Wat zijn de timing-requirements, en andere protocol-eisen, zowel mbt. de PTT verbinding, als mbt. de toestellen?

Voor simulatie, analyse, implementatie e.d. zijn we voornamelijk geïnteresseerd in modellering met behulp van events*. Voor de genoemde componenten moeten we eerst event-modellen maken, d.w.z. modellen die zich lenen voor 'discrete event simulation'. In sommige gevallen kan dit lastig zijn, bijvoorbeeld voor de bus: deze moeten 'bidirectioneel' events propageren naar de componenten die via schakelaars verbonden zijn (of naar de schakelaars, die verder de disrtibutie naar de component verzorgen?). We moeten misschien de component die de events aan de bus geleverd heeft uitzonderen!*

1.4.1 Met betrekking tot de simulatie

Voordat we een simulatie-model opstellen moeten we eerst het doel van de simulatie *duidelijk maken. Er zijn namelijk vele soorten simulatie mogelijk; het doel van de simulatie heeft een grote invloed op de modelvorming: in het algemeen proberen we een model zo eenvoudig mogelijk te maken. Voor deze vereenvoudiging zijn er tenminste twee redenen: ten eerste kost het opstellen en verifiëren van een eenvoudig model minder tijd; ten tweede kost het daadwerkelijk simuleren van een eenvoudig model meestal aanzienlijk minder resources dan een simulatie die alle details tot op het laagste niveau betreft. (Om een extreem voorbeeld te noemen: voor een performance-analyse van een computer-systeem op het niveau van het Operating System kunnen we dit computersysteem beter niet op het niveau van de analoge electronica (Spice) simuleren!)*

In ons geval is het doel van de simulatie: aantonen dat de besturing van een HomeVox centrale met behulp van Object-Oriented technieken gerealiseerd kan worden (en misschien ook: een demonstratie van de voordelen van deze aanpak). Uiteindelijk zal het bestruingsprogramma in plaats van een gesimuleerde HomeVox hardware, de echte HomeVox hardware moeten kunnen aansturen.

Een belangrijk aspect is dus dat we onderscheid maken tussen (i) de gesimuleerde hardware die uiteindelijk door de echte hardware vervangen gaat worden, en (ii) de besturingssoftware die we bij deze vervanging eigenlijk ongewijzigd over willen nemen. De gesimuleerde hardware moet zoveel mogelijk dezelfde interfaces hebben als de uiteindelijke hardware.

1.5 Afspraken betreffende specificatie

Later zijn, o.a. naar aanleiding van de hierboven vermelde opmerkingen en vragen, de volgende afspraken toegevoegd aan de specificatie:

1. In een ruggespraak situatie, met een call on-hold, wordt voor beide toestellen, het neerleggen geïnterpreteerd als call-forward. Het andere toestel wordt automatisch verbonden met de lijn die on-hold staat.
wat is hier call on-hold?

2. In een call hold situatie mag degene, die de ruggespraakverbinding maakt reeds neerleggen terwijl de, in ruggespraak opgeroepen partij, nog gebeld wordt. De party on-hold ontvangt dan weer de vrijtoon. Bij opnemen van de telefoon worden beide partijen verbonden.

3. Begrenzing van intern wekken: 60 seconden.
Begrenzing van extern wekken: zolang de ptt wekt, blijven wekken.
Ook hier: wekken = bellen, het toestel laten rinkelen.

4. Bij call hold van een externe verbinden en neerleggen van de ruggespraak initiator voordat de derde partij beantwoord heeft (zie 2), moet de vrijtoon naar de openbare centrale worden gezonden.
De derde partij wordt 60 seconden gewekt, daarna moeten allen weer gewekt worden. Totale begrenzing van het wekken door stoppen met wekken door ptt.

5. Initiatie van ruggespraak door kiezen van cijfer tijdens gesprek. DTMF toon mag wel naar buiten doorlopen, Het impulskiezen moet zo snel mogelijk worden afgestopt naar de buitenlijn: geen gekraak.

6. Het nummer herhalen geschiedt door het kiezen van de 5, direct na offhook.

7. Het kiezen van een cijfer na 15 seconden kiespauze wordt geïnterpreteerd als initiatie ruggespraak.

8. Er is slechts 1 gesprek tegelijkertijd mogelijk. Dit is een binnenlijn of een buitenlijngesprek.

9. Zijn er situaties dat niet alle gewenste tonen gegeven kunnen worden? Waarschijnlijk wel. Gevraagd is om die situaties in kaart te brengen en de dan gekozen oplossing te geven.

10. Dubbele wektoon is intern wekken.

11. Inkomende oproep tijdens intern gesprek.
Diegenen die in gesprek zijn ontvangen een attentiesignaal. De vrije toestellen worden gewekt. Degene die het eerst beantwoordt ontvangt de inkomende oproep. De anderen worden verbroken, zo mogelijk met een correcte toon.

12. Initiatie drieling gesprek door, in ruggespraak, nogmaals zelfde nummer te kiezen.

13. Drieling gesprek ontstaat altijd uit ruggespraak situatie.

14. Vanuit een interne verbinding mag geen ruggespraak of drieling naar de buitenlijn worden opgebouwd. Een drieling gesprek mag dus wel voor een inkomende buitenlijn verbinding.

15. Geen ingebouwde testprocedures voor de hardware.

16. DTMF detector geeft alle 16 mogelijke DTMF signalen af. Signaaloverdracht als een ASCII-code.

17. Als men on-hold staat, krijgt men geen toon.

18. Naar PTT wordt vanuit de homevox weer een vrijtoon gezonden als men bij call forward neerlegt voordat door de derde partij is beantwoord.

19. De toongenerator heeft voldoende vermogen om alle toestellen tegelijkertijd van een toon (dezelfde) te voorzien.

20. De toongenerator maakt zelf de benodigde pauzes voor de toonsignalen.

21. Uitzenden van DTMF signalen: duur 55 msec, duur pauzes tussen de signalen ook 55 msec.

22. DTMF generator hoeft niet afgeschakeld te worden i.v.m. demping.

23. Simulatie-eisen: getoond en bediend moeten worden:
– de aangesloten toestellen
– de PTT lijn
– de toestand van de centrale.

1.6 Aandachtspunten implementatie homevox

1.6.1 opstarten

Bij het opstarten moet gekeken worden of de centrale DTMF aan kan. Dit werkt als volgt:
– lijn beleggen (kiestoon-detector aan)
– wachten op kiestoon
– kiestoon detector uitzetten
– DTMF-toon zenden (bijvoorbeeld 0)
– kiestoon-detector aanzetten
– time-out zetten (1 seconde?)
– indien kiestoon detected: oude centrale (alleen puls)
– indien time-out: centrale herkent DTMF.

1.6.2 inkomend belsignaal

Het belsignaal is niet scherp begrensd. Neem aan dat indien langer dan 5 seconden geen refresh heeft plaatsgehad, de centrale ermee is gestopt. Alle toestellen gaan in tempo 1 seconde aan, 3 seconden uit. Alle toestellen afwisselend i.v.m. belasting.
Indien er toestellen in interne traffic zijn, krijgen zij het attentie-signaal (soort tikker) door hun gesprek heen. Er kan door hen alleen bantwoord worden door onhook/offhook.
Indien er toestellen in signalering zijn, krijgen zij de interne bus (dus eventueel onderlinge communicatie) gemengd met het attentie-signaal. Alle andere signalering wordt afgebroken.
Beantwoorden bij inkomend belsignaal: direct buitenlijn (geen 0 kiezen nodig).
Eventueel gekozen digits (1–4) worden als ruggespraak naar het betreffende toestel gezien (zie ook uitgaand bellen).
Vanuit ruggespraak kan de primaire partij het volgende doen:
– 0 kiezen: andere interne partij eraf gooien, terug naar buitenlijn.
– eigen toestel bellen: drieling gesprek maken (hierna is er geen primaire partij meer totdat er één onhook gaat!)
– onhook (derde partij offhook): gesprek doorgeven.
– onhook (derde partij al onhook): behandelen als inkomend bel-signaal (buitenlijn blijft belegt!). Dit is een tricky requirement. Je kunt hem eventueel weglaten.

1.6.3 intern bellen

Belsignaal met gat maken (verschil met buiten bel): 550 msec aan, 110 msec uit, 550 msec aan, 1100 msec uit.
Bij intern geïnitieerde call kan er geen ruggespraak met de buitenlijn gemaakt worden. Andere ruggespraak is wel mogelijk, analoog met inkomend belsignaal (ook drieling gesprek).

Indien er al een intern gesprek gaande is, wordt er bij een nieuwe initiator bij voorkeur de bezettoon gegeven. Indien dit i.v.m. resources niet mogelijk is: een dode lijn.

1.6.4 uitgaand bellen

Tijdens de call set-up wordt het nummer opgeslagen voor eventueel nummerherhaling (wissen bij het opnieuw kiezen van de buitenlijn). Dit geheugen wordt niet beïnvloed door eventueel intern bellen.
Bij uitgaand bellen wordt eventueel DTMF vertaald naar pulsen. Pulsen worden op cijferbasis geregenereerd.
Bij het opslaan van het nummer moeten bij puls bellen eventuele kiestonen ook gedetecteerd worden. Bij nummerherhaling dient op deze momenten gepauseerd te worden tot er opnieuw een kiestoon gedetecteerd wordt (i.v.m. registerloze centrales).
Tijdens call set-up worden de digits als kiesinformatie gezien. Na een digitloze periode van 15 seconden of meer, wordt digit 1–4 voor ruggespraak naar het betreffende toestel gezien.
Ruggespraak net als bij inkomend belsignaal.

1.6.5 nummerherhaling

Let op pauzes voor dial-tone.

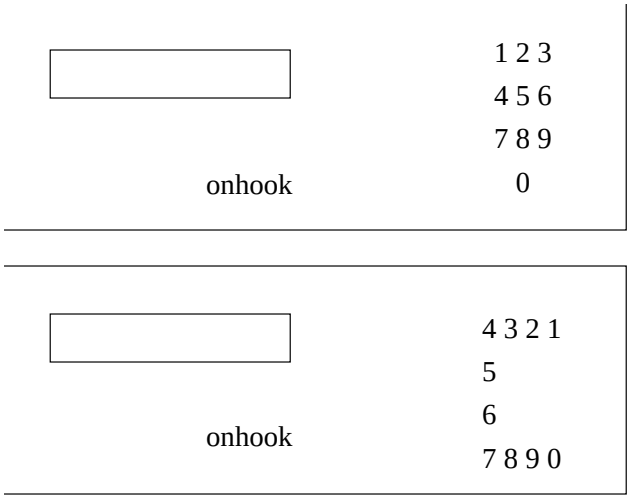
1.6.6 Simulatie

Een model voor simulatie kan alsvolgt zijn: verdeel het scherm in vier stukken: drie voor een toestel en één voor de centrale. Per toestel de mogelijkheid tot nummer keuze en onhook/offhook en daarnaast een kader waarin een systeemboodschap komt (bijvoorbeeld i-kiestoon, i-vrijtoon, i-bezettoon, i-attentiesignaal, i-ringsignaal, e-ringsignaal, e-kiestoon, e-bezettoon, cijfer kiezen, rust, i-verkeer (x,y) (verbonden met toestel x en y), e-verkeer; waarbij i = intern en e = extern). Via muis en/of keyboard kan onhook/offhook getoggeld worden en kunnen cijfers worden gekozen.

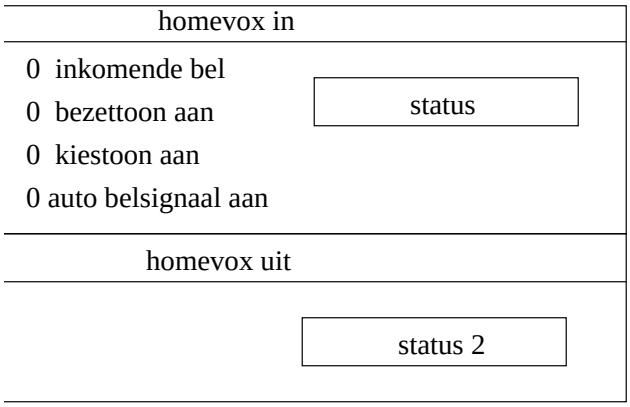
Daarnaast de centrale interface, waarbij 4 keuzes die met de muis aangeklikt kunnen worden:
– inkomende bel (1 ring van 1 sec.)
– bezet toon (toggle aan/uit)
– kiestoon (toggle aan/uit)
– auto belsignaal toggle (1 sec. aan, 3 sec. uit, etc.)
Daarnaast is er een status-venster, die het signaal aangeeft conform de gemaakte keuze.
Een tweede status-venster voor uitgaande signalen geeft de homevox toestand aan:
– beleggen
– vrijgeven/rust
– pauze (= wachten op kiestoon)
– verkeer
– kies cijfer y (bij uitkiezen)

1.6.7 Diversen

Als een gebruiker geen progress heeft, wordt hij er na ca. 15 seconden afgegooid (dode lijn).
55 msec timer-interrupt is gekozen i.v.m. overeenkomst IBM-PC. Hierbij is de timer interrupt ook 55 msec.



Figuur 1.2: Window voor DTMF-toestel (boven) en pulse toestel



Figuur 1.3: Window voor de centrale interface

Uit de beschrijving is het wellicht niet geheel duidelijk: maar ook de LF geeft bij een digit dialed interrupt het gekozen cijfer als parameter mee.

2
HomeVox1: interne gesprekken

Voor de eerste versie gebruiken we de volgende vereenvoudigingen:

hoog abstractieniveau Gebeurtenissen worden zoveel mogelijk, voor zover dit voor een beschrijving van de functionaliteit op een hoog niveau mogelijk is, als ondeelbaar beschouwd. Zo maken we ons niet druk over de tijdsduur van een belsignaal of een toon; ook het kiezen van een nummer beschouwen we als een ondeelbare gebeurtenis. (Later zullen we dit detailleren tot het kiezen via de afzonderlijke cijfers.)

eenvoudige, interne gesprekken We laten de PTT-aansluiting eerst buiten beschouwing. De interne gespekken mogen alleen enkelvoudig zijn: geen ruggespraak, geen doorverbinden, slechts gesprekken met twee personen tegelijk.

We gebruiken UML en de methode van Larman, waarbij we niet alle diagrammen en deelmodellen zullen meenemen. Aan de hand van use-cases zullen we proberen sequence diagrammen te maken en deze in de design fase m.b.v. collaboration diagrammen verder detailleren. De implementatie aan de hand van deze diagrammen is dan vrijwel rechttoe-rechtaan.

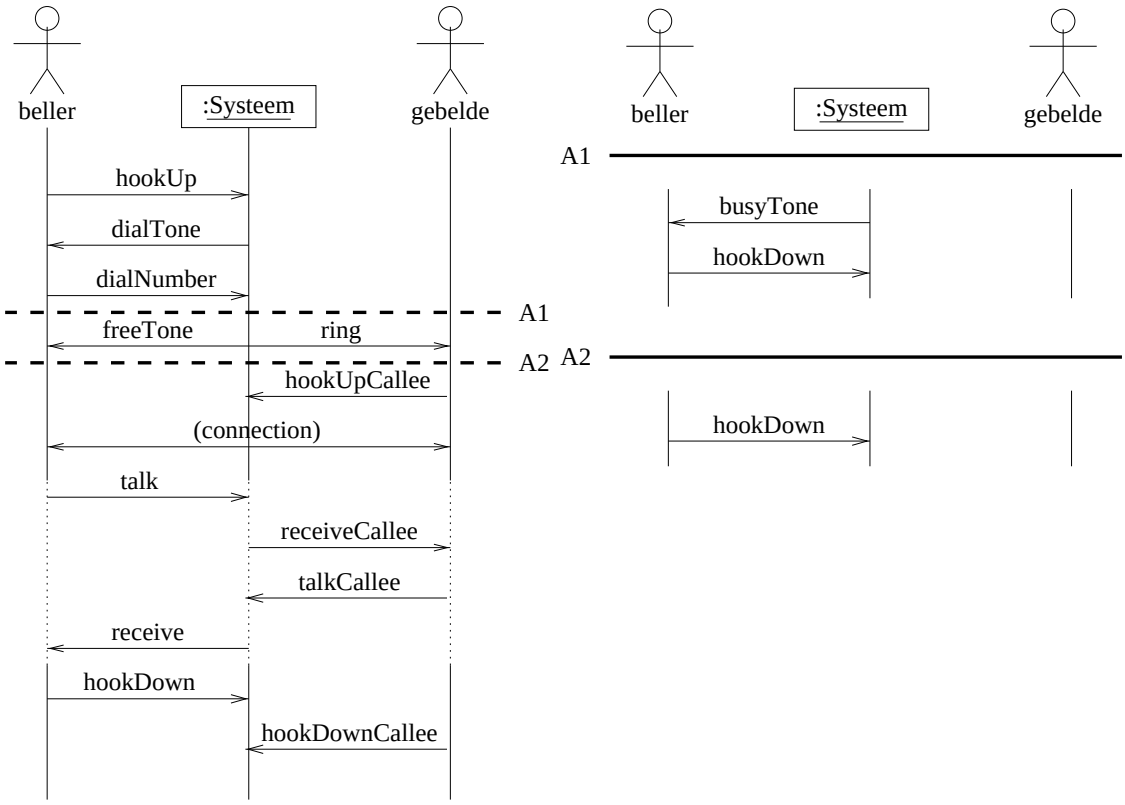
2.1 OOA

We proberen bij deze analyse eerst de relevante scenario's op te schrijven. Daaruit leiden we de objecten af, en vervolgens proberen we de scenario's te verfijnen, in dit geval tot aan de implementatie in Java. Het kan hierbij nodig zijn de oorspronkelijke analyse aan te passen, door het verkrijgen van nieuwe inzichten.

2.1.1 use case

Voor een eenvoudig gesprek gaan we uit van de onderstaande use-case. De 'actors' hierbij zijn: de beller (caller), de homevox, en de gebelde (callee). We kunnen de beller en de gebelde ook representeren door het overeenkomstige toestel.

eerste scenario. De beller (caller) neemt de hoorn van de haak. De homevox beantwoordt dit met de kiestoon. Daarop kiest de beller een nummer (mogelijk bestaand uit meerdere cijfers). De homevox gaat na of het overeenkomstige toestel vrij is;
(i) zo ja, dan krijgt de beller de vrijtoon, en het gebelde toestel het belsignaal. Als de gebelde hierop de hoorn van de haak neemt, wordt de verbinding tussen beide toestellen gelegd, en is een gesprek mogelijk. De verbinding wordt verbroken als de beller of de gebelde de hoorn op



Figuur 2.1: Sequence diagram voor interne gesprek

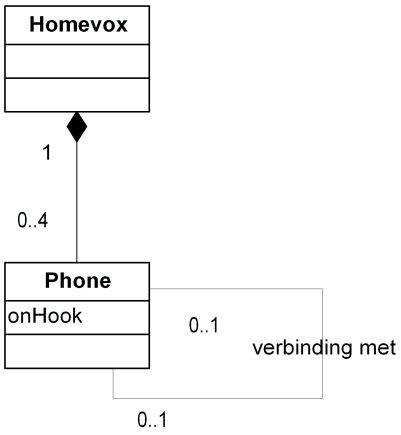
de haak legt.
(ii) zo nee, dan krijgt de beller de bezettoon. Hierop legt hij de hoorn op de haak.

We hebben hier de belangrijkste variaties gegeven. Er zijn nog andere mogelijkheden; de beller kan op elk moment de hoorn op de haak leggen, waardoor het gesprek (in opbouw) direct afgebroken wordt. Ook kan de gebelde nalaten de hoorn op te nemen, maar daarover maken we ons verder geen zorgen: we gaan ervan uit dat de beller dan binnen redelijke tijd de hoorn neerlegt.

Het scenario is in figuur 2.1 in de vorm van een sequence diagram gegeven. Met een vette stippellijn zijn plaatsen aangegeven waar alternatieve diagrammen kunnen beginnen. De overeenkomstige nummering (bijvoorbeeld “A1”) verwijst dan naar het alternatieve scenario, dat eronder is gegeven.

De operatie “connection” is tussen haakjes geplaatst: eigenlijk gebeurt er niets waardoor duidelijk wordt dat er verbinding is. Er valt een stilte aan beide zijden en daaruit dienen beller en gebelde op te maken dat er een verbinding is.[1] Zolang geen van beide iets zegt is echter niet duidelijk of de stilte veroorzaakt is door een dode lijn! Intern in de homevox zal deze operatie wel gevolgen hebben: de nodige schakelaars zullen goedgezet moeten worden.

[1] De overgang naar “connection” kan eventueel verduidelijkt worden door toevoeging van een extra actie hiervoor: “homevox geeft geen toon,” waarmee we aangeven dat de vrijtoon wordt verwijderd. We kunnen hieruit afleiden dat sommige acties een duidelijk begin en eind hebben, bijvoorbeeld een belsignaal dat stopt wanneer de hoorn van de haak wordt genomen.



Figuur 2.2: Eerste stap van de analyse

OO-Analyse, eerste stap. We hebben is elk geval twee classes: Phone (voor beller en gebelde), en Homevox:[2]

- class Phone

  attributen
  onHook: boolean[3]
  connection: Phone[4]

- class Homevox

  attributen
  thePhones[5]

We krijgen zo figuur 2.2.

OO Analyse: het gesprek. Tussen twee Phones kan een instance connection bestaan (ook letterlijk een verbinding), namelijk in het geval van een *gesprek*. Als we van deze connection gegevens willen bijhouden, zoals de duur van het gesprek, dan moeten we daarvoor een Class introduceren (‘Call’). Vervolgens kunnen we ons afvragen of we deze class alleen willen gebruiken voor de administratie van een gesprek ten behoeve van de Homevox, of dat we deze class een grotere ‘verantwoordelijkheid’ willen geven. Hier ligt dit laatste voor de hand: we kunnen deze class verantwoordelijk stellen voor de juiste afwikkeling van het gesprek, dat wil zeggen voor de ‘life cycle’ daarvan.

We komen zo tot de volgende taakverdeling:

[2]De hier vermelde services zijn deels afgeleid van de scenario’s en de daaruit afgeleide collaboration diagrammen in de OOD fase. Feitelijk is de lijst van bekende services op dit moment mogelijkerwijs een stuk kleiner, maar wordt dan in de OOD fase verder aangevuld. We kiezen ervoor om hier al een zo volledig mogelijke lijst te tonen.

[3]Pas in de implementatie wordt echt duidelijk, waarom dit attribuut nodig is: dan blijkt dat het belangrijk is te weten in welke toestand het toestel zich bevindt.

[4]Dit is een OOD-attribuut en nodig om te weten met welk toestel we een verbinding hebben.

[5]Een OOD-attribuut, nodig voor het bijhouden van de connecties.

Phone representeert het toestel en de toestand daarvan, en zorgt voor de juiste afhandeling van het toestel-protocol.

Call representeert het gesprek en de toestand daarvan; zorgt voor de juiste afwikkeling van het gesprek (gesprek life-cycle).

Homevox representeert de Homevox en de (globale) toestand daarvan; zorgt voor de globale coördinatie, o.a. van de resources.

Van een Call is het 'caller' attribuut altijd ingevuld (en ongelijk `null`); de 'callee' wordt pas ingevuld als deze bekend is, d.w.z. nadat de caller het nummer opgegeven heeft. Merk op dat dit nog niet wil zeggen dat de callee daadwerkelijk deelneemt: dat gebeurt pas op het moment dat deze de hoorn van de haak neemt. Deze relatie tussen de Call en de callee wordt alleen gelegd als de callee vrij is, d.w.z. deze is niet geclaimd voor een andere Call (d.i. het call-attribuut van het callee-object is ongelijk aan `null`). Een toestel kan door ten hoogste één Call geclaimd worden.

Bij het verbreken van de verbinding, doordat een van beide toestellen de hoorn op de haak legt, wordt de relatie van de Call met beide toestellen (eenzijdig) verbroken: de toestellen hebben geen relatie meer met het Call-object.[6] In het Call-object houden we, voor administratieve doeleinden, deze gegevens nog wel bij.[7]

Door het toevoegen van de class **Call** krijgen we figuur 2.3. De associations tussen class en phone-objecten zijn van namen voorzien aan weerskanten van de associatie. De namen gebruiken we in de rest van het verhaal om duidelijk te maken hoe het object "aan de andere kant van de associatie" moet worden aangeduid. Zo kunnen van uit het Call-object twee Phone-objecten worden aangewezen, die we zullen aanduiden met resp. "caller" en "callee" (beller en gebelde).

- class Homevox

  attributen
  thePhones

- class Call

  attributen
  caller: Phone
  callee: Phone

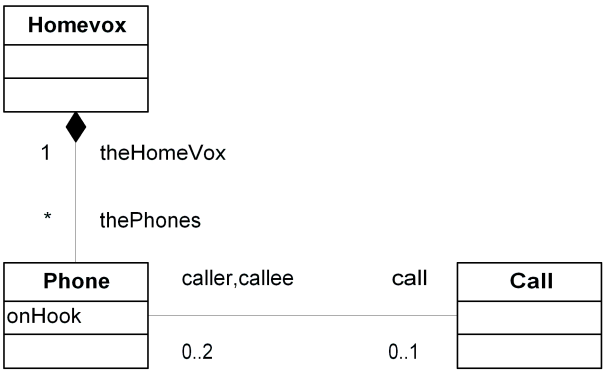
- class Phone

  attributen
  call: Call
  number: int

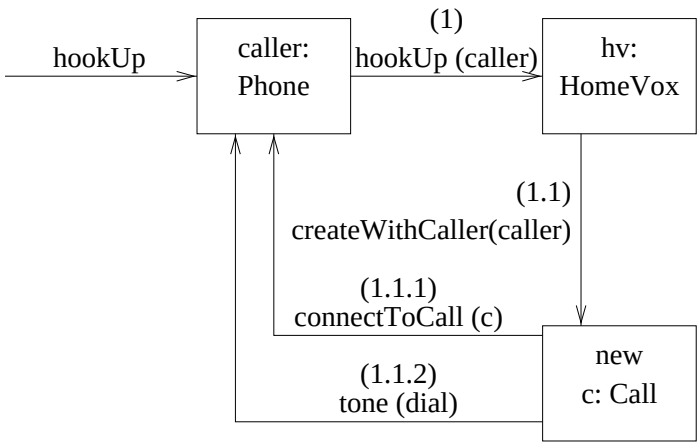
In het zo geformuleerde analysemodel worden de Call-objecten niet verder "bewaard" voor administratieve doeleinden. Dit is voor de homevox ook niet nodig. Voor een centrale waarbij de administratie wel een rol speelt, kunnen we een extra instance connection leggen tussen HomeVox en Call, die bij initiëren van een Call-object wordt gelegd, waardoor de HomeVox steeds de mogelijkheid heeft de administratie bij te werken en de benodigde gegevens van alle Call's op te vragen.

[6] Desgewenst kunnen we het gesprek pas afbreken als de caller de hoorn op de haak plaatst; met de gekozen representatie is dat geen groot probleem.

[7] Misschien moet dat later anders: scheiden van administratie en de actuele toestand. We zouden in elk geval in de toestand van de Call kunnen aangeven dat dit geen actueel gesprek betreft.



Figuur 2.3: Tweede stap van de analyse: uitbreiding met Call



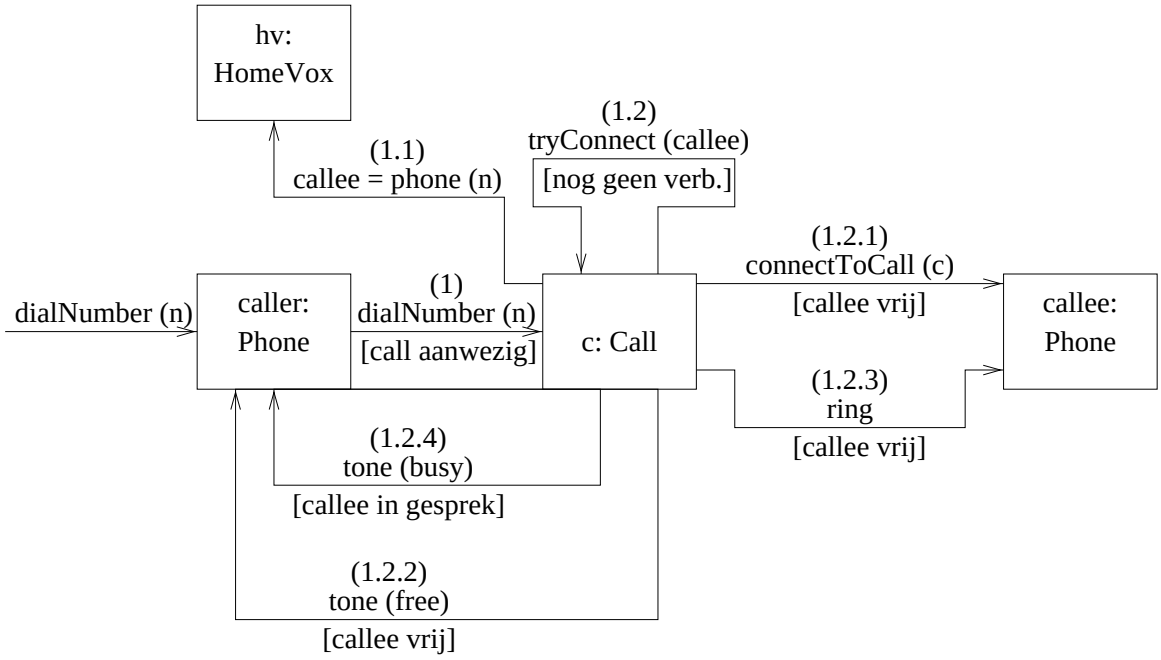
1. meld de homevox dat caller de hoorn van de haakt neemt
1.1. creëer een nieuw Call-object
1.1.1. verbind call met caller
1.1.2. dial-tone naar caller

Figuur 2.4: Eerste aanzet voor een collaboration diagram voor de message "hookUp"

2.2 OOD

Als eerste stap zullen we proberen uit de analyse de benodigde collaboration diagrammen te maken. We maken deze diagrammen voor alle input-events. We beginnen met het inputevent "hookUp." Eerst kijken we welke objecten een rol spelen voor dit event, dan zoeken we een geschikte controller (het object dat deze message ontvangt), waardoor de overige objecten automatisch de rol van collaborator krijgen. In geval van de message "hookUp" is het object "caller" (van class "Phone") een geschikte controller. In figuur 2.4 staat vervolgens de gedetailleerde uitwerking van deze message beschreven. In feite is de enige actie (aannemende dat "hookUp" het van de haak halen van een toestel is met de bedoeling te gaan bellen) het aanmaken en initialiseren van een nieuw "Call"-object.

Opmerking: men zou geneigd kunnen zijn om als conditie bij actie (1) op te nemen dat de



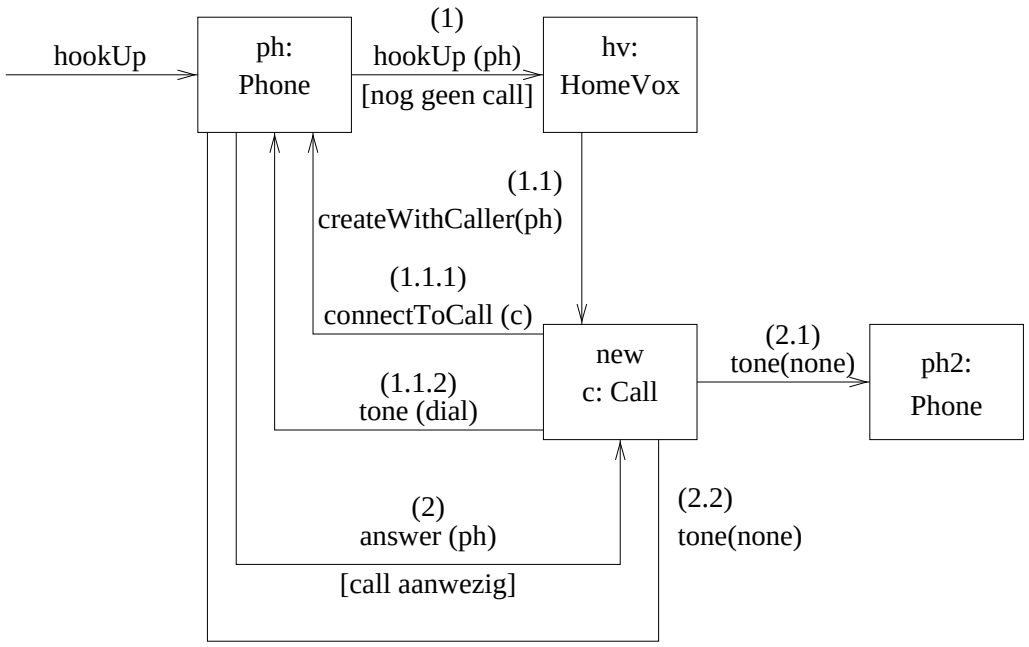
1. meld Call-object dat caller een nummer heeft gedraaid
1.1. bepaal de door dit nummer aangeduide callee
1.2. probeer verbinding te maken met callee
Gekozen toestel vrij:
1.2.1. maak de connectie
1.2.2. geef vrijtoon aan caller
1.2.3. geef belsignaal aan callee
Gekozen toestel in gesprek:
1.2.4. geef bezettoon aan caller

Figuur 2.5: Collaboration diagram voor de message "dialNumber"

telefoon zich bevindt in de toestand waarbij oorspronkelijk de hoorn nog op de haak ligt. Dit is echter ongewenst. Zou een event "hookUp" kunnen optreden indien de hoorn al van de haak is, dan is dit een ontwerpfout in het interface! In ons geval een fout in the telefoontoestel, dan wel in ons gesimuleerde toestel (het window op het scherm).[8]

Het volgende input-event in de life-cycle is "dialNumber." We zullen deze message van een argument voorzien (namelijk het gedraaide of ingetoetste nummer). Wederom is de "caller" de meest voor de hand liggende controller. We krijgen het diagram als in figuur 2.5. Merk op, dat we in plaats van verschillende messages zoals "toneFree" en "toneBusy" gekozen hebben voor één message "tone" met een argument om aan te geven welke toon gegenereerd moet worden. De conditie "callee vrij" volgt uit het feit dat het callee-object nog niet aan een Call-object refereert, ofwel `call=null`.

[8] Met moderne toestellen waarbij de "haak" is vervangen door een knopje, moet de vervolgactie van het indrukken van het knopje worden gerelateerd aan de toestand van het toestel. Is al eerder op het knopje gedrukt (om "de hoorn van de haak te nemen" dan moet een volgende druk op de knop resulteren in het weer neerleggen van het toestel. Het kan handig zijn hiervoor de toestand van een bijbehorende eindige automaat mee te nemen. Eén en ander is in de huidige implementatie ook als zodanig opgelost, nl. door een attribuut `onHook` te introduceren, die de toestand van de hoorn bijhoudt.



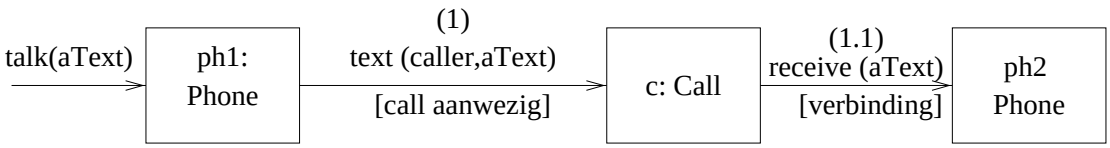
Toestel ph is nog niet in een verbinding verwikkeld:
1. meld de homevox dat callee de hoorn van de haakt neemt
1.1. creëer een nieuw Call-object
1.1.1. verbind call met caller
1.1.2. dial-tone naar caller
Ph heeft al een verbinding:
2. meld call-object dat ph gesprek beantwoordt
2.1. annuleer tone bij caller
2.2. annuleer ring bij callee

Figuur 2.6: Collaboration diagram voor de message "hookUp"

Het input-event "hookUpCallee" is in feite een vergelijkbaar event, als het event "hookUp" van de caller. Het Phone-object zelf is niet in staat om (zonder in zijn toestand te kijken) te zien of het opnemen van de hoorn gebeurt om te gaan bellen, of om te beantwoorden. We kiezen er daarom hier voor, één event "hookUp" te beschouwen, wat zowel voor de caller als de callee moet kunnen werken. We zullen daarom het diagram uit figuur 2.4 herzien. We verkrijgen dan figuur 2.6. Hierin is bovendien meegenomen, dat het beantwoorden van een gesprek tevens inhoudt, dat de aanwezige toon aan de kant van de beller dient te worden opgeheven. We noteren dit in eerste instantie door een nieuwe toon te genereren, de none-toon.

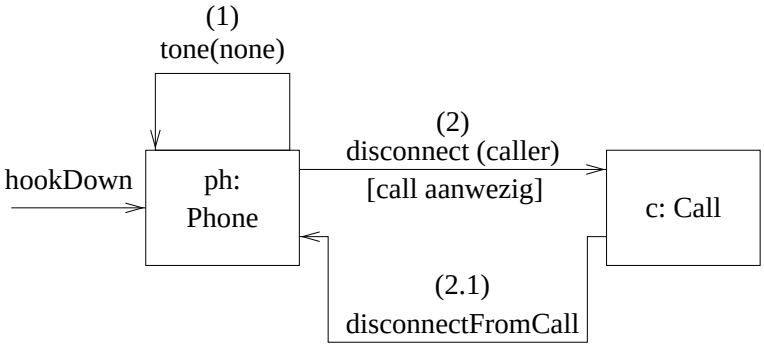
Zonder verder commentaar geven we vervolgens de collaboration diagrammen voor de events "talk" (voor "talkCallee" nemen we hetzelfde event om gelijke redenen als boven) en "hookDown" (en "hookDownCallee" zullen we eveneens gelijknemen met "hookDown"), zie figuren 2.7 en 2.8. Een disconnect van Phone en Call zullen we hier eerst eenzijdig laten plaatsvinden, d.w.z. alleen het toestel dat oplegt wordt gedisconnect.[9]

[9] Bij de introductie van externe gesprekken zullen we dit anders doen.



1. meld call-object ingesproken tekst en van wie dit komt
1.1. zend tekst naar andere gesprekspartner

Figuur 2.7: Collaboration diagram voor de message "talk"



1. annuleer eventuele toon

(nog) verbinding aanwezig:

2. meld call-object beëindiging van verbinding
2.1. meld caller beëindiging van verbinding

Figuur 2.8: Collaboration diagram voor de message "hookDown"

2.3 OOP

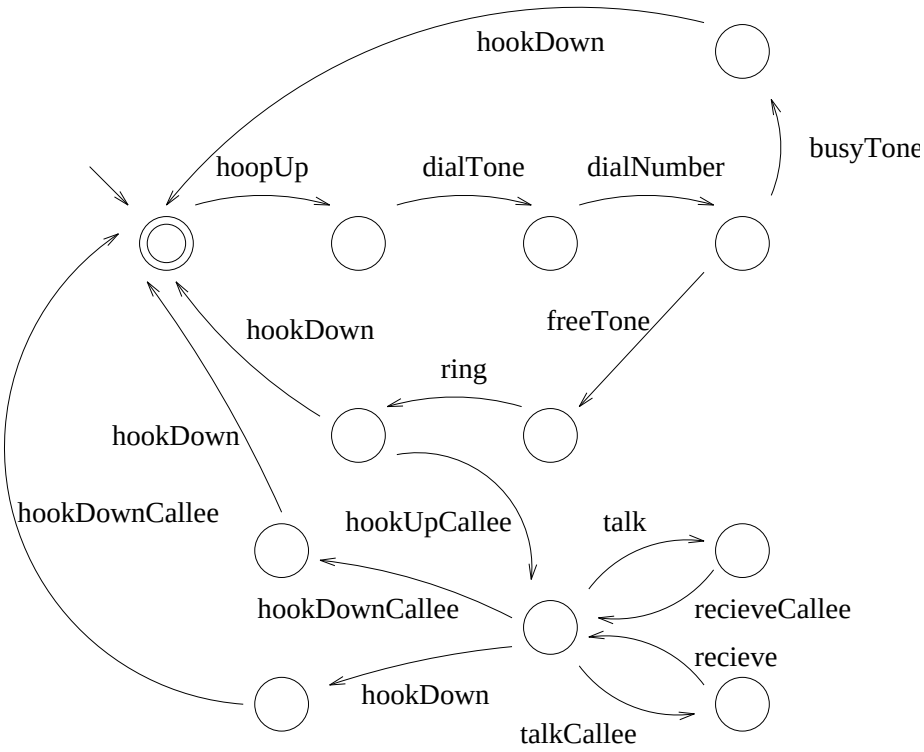
Tot slot zullen een implementatie geven. Merk op dat input-events zoals "hookUp," "talk" e.d. niet op elk moment kunnen worden uitgevoerd. Het life-cycle model geeft precies aan wanneer welk input-event kan optreden.

Om de leesbaarheid te bevorderen (en om toestanden te verkrijgen, waarin het systeem zich kan bevinden tijdens de life cycle) creëren we eerst een eindige automaat aan de hand van het sequence diagram.

We kunnen nu de toestanden nummeren en in ons systeem bijhouden in welke toestand we ons bevinden. Alleen indien het systeem zich in een toestand bevindt van waaruit in de automaat een pijl vertrekt gelabeled met de naam van een zeker event, mag dit event ook daadwerkelijk worden uitgevoerd.

We kunnen een aantal vereenvoudigingen uitvoeren. Zo kan, volgens deze automaat, een "hookDown" slechts plaatsvinden na een "hookUp" (en evenzo een "hookDownCallee" na een "hookUpCallee"). Het is dus voldoende bij te houden in welke toestand een toestel zich bevindt. Dit is gedaan door in de Phone-objecten een attribuut "hookState" op te nemen.

Voorts kan een nummer alleen worden gedraaid, indien de haak van het toestel genomen is, en kan alleen een gesprek worden doorgegeven, indien er verbinding is. Dit laatste bijvoorbeeld, is te controleren, door te kijken of het call-object volledig is ingevuld (dus zowel "caller" als



Figuur 2.9: Eindige automaat voor interne gesprekken

"callee" attribuut ingevuld heeft).[10]

Kortom: het is niet noodzakelijk een volledige toestandsbeschrijving bij te houden. Wel kunnen we uit de eindige automaat goed aflezen, wanneer welk event mag optreden. Tot slot spreken we af, dat een input event op een niet toegestaan moment wordt genegeerd.

[10] Spreken kan in feite zodra de hoorn van de haak is. Het gesprek wordt slechts doorgegeven, indien er een verbinding tot stand is gekomen.

3
Homevox2: externe gesprekken

In dit hoofdstuk behandelen we de externe gesprekken. We zullen dezelfde methode hanteren als in hoofdstuk 2, maar niet altijd even gedetailleerd op alle aspecten ingaan.

Bij de externe gesprekken onderscheiden we twee varianten: inkomende gesprekken en uitgaande gesprekken. Bij deze gesprekken is steeds, naast een lokaal toestel, ook het externe telefoonnet (‘de ptt’) betrokken.

3.1 OOA: inkomend gesprek

3.1.1 use case

Eerst geven we de belangrijkste use-case van een inkomend gesprek in informele vorm:

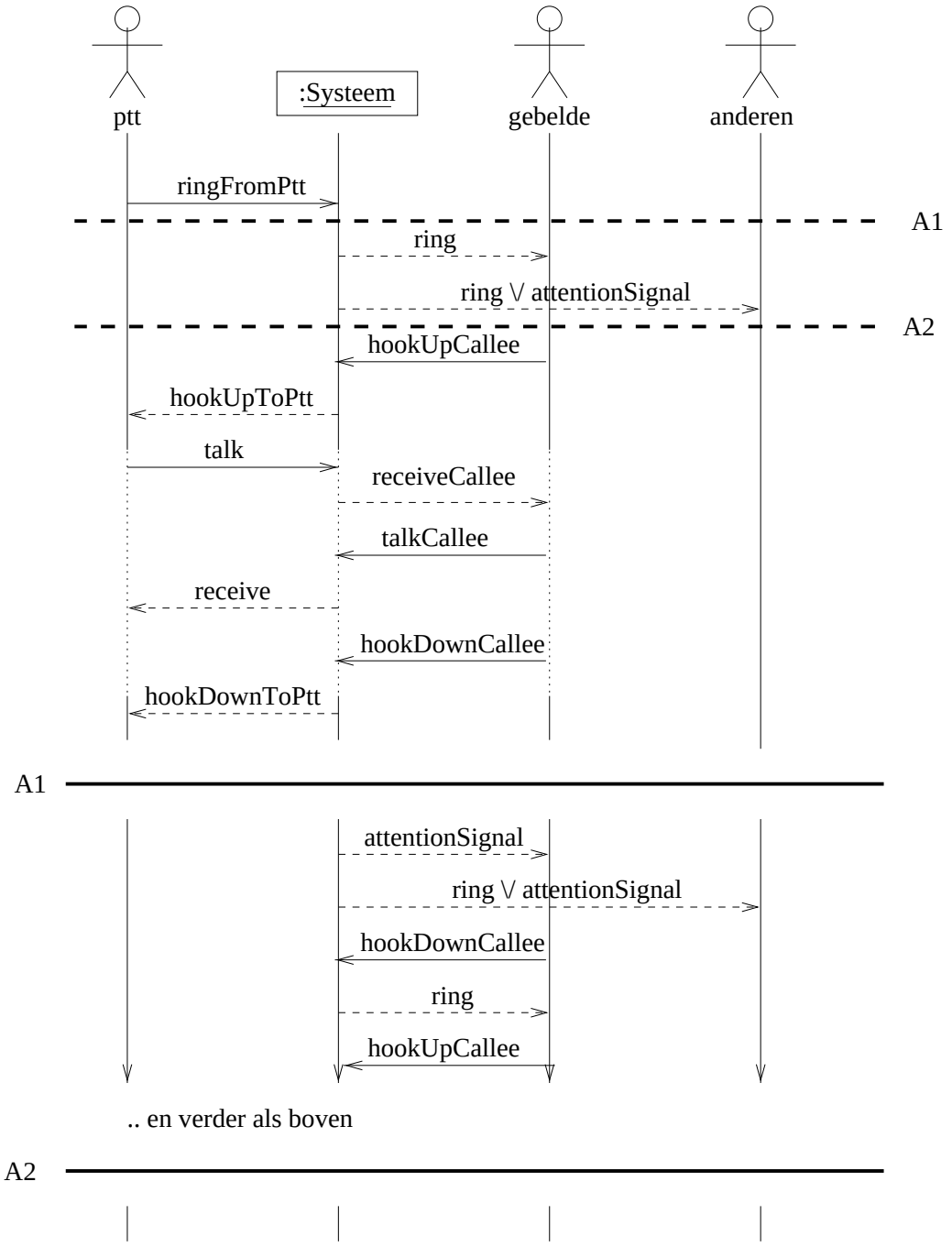
Het externe telefoonnet (‘de ptt’) geeft een belsignaal aan de homevox. De homevox geeft dit belsignaal door aan alle vrije toestellen, en geeft een attentiesignaal aan de toestellen die niet vrij zijn (d.w.z. in een gesprek verwikkeld). Het eerste toestel dat vervolgens de hoorn van de haak neemt, wordt verbonden met het externe telefoonnet. (Bij alle toestellen wordt het belsignaal verwijderd.) Een eventueel intern gesprek wordt afgebroken.[1] Vervolgens vindt het gesprek plaats. De verbinding wordt verbroken wanneer het lokale toestel de hoorn op de haak plaatst.

Bij dit scenario zijn de volgende objecten betrokken: (i) het externe telefoonnet (ptt); (ii) alle lokale toestellen (voor de signalering); (iii) het lokale toestel (dat uiteindelijk verbonden wordt met het telefoonnet, deze zullen we aangeven met callee); (iv) de homevox.

In figuur 3.1 is het scenario in diagramvorm gegeven. Merk op dat de “hookDown” aan de Ptt-kant niet door de homevox als bijzonder event wordt opgevat: als de beller ophangt zorgt de ptt-centrale voor een bepaalde toon op de lijn. In feite is het ophangen door de beller niets anders dan een bijzondere “talk” over een lijn. Er is dan ook geen speciaal event “hookDown” voor het Ptt-object.

varianten. Dit scenario heeft een paar varianten: (a) de gebelde kan in een gesprek verwikkeld zijn: in dat geval moet de hoorn eerst neergelegd worden (om het lokale gesprek af te breken). (b) de gebelde neemt de hoorn niet op: dit is eigenlijk alleen te detecteren door een time-out. of door het wegvallen van het belsignaal. (Voor het einde van een signaal hebben we tot nu toe nog geen acties ingevoerd; in dit geval is dat wel nodig!) Na verloop van tijd hangt de ptt weer op.

[1] Er zijn niet genoeg hardware-resources om tegelijkertijd een intern en een extern gesprek te voeren.



Figuur 3.1: Sequence diagram voor een inkomend gesprek

Hoe is met de huidige analyse in geval (a) bekend wat het externe gesprek is? Het call-attribuut van een gesprek betreft het *interne* gesprek. Op het moment dat de hoorn neergelegd wordt, moet dit toestel alsnog geclaimd worden door het externe gesprek; het toestel krijgt dan ook het belsignaal. Het probleem is dat we het ‘call’ attribuut eigenlijk voor twee doelen gebruiken: (i) om aan te geven aan welk gesprek het toestel *actief deelneemt*; (ii) om aan te geven

welk gesprek dit toestel *claimt*. Het is waarschijnlijk beter om voor deze twee aspecten twee attributen te gebruiken: `activeCall` voor het eerste geval, `claimingCall` voor het tweede geval.

Hoe gebruiken we de call-attributen? Een gesprek kan een toestel *claimen* om dit actief in het gesprek te betrekken. Dit betreft meestal het toestel dat gebeld wordt (vanuit een intern gesprek of vanuit een binnenkomend extern gesprek). Door de hoorn op te nemen kan de gebelde deze claim omzetten in een actieve deelname. De beller kan de claim opheffen door de hoorn neer te leggen.[2][3]

Opmerking: Bovenstaande uitbreiding van Call-objecten maakt het noodzakelijk het ontwerp en de implementatie van de interne gesprekken aan te passen. Aangezien de input events in dit hoofdstuk voornamelijk uitbredingen zijn van de events uit het vorige hoofdstuk, zullen we de aanpassingen gelijktijdig met de uitbreidingen aan bod laten komen.

We willen nu de detaillering van het inkomend gesprek behandelen. We gebruiken hierbij weer de splitsing in Homevox en Call. Daarnaast introduceren we het 'Ptt' object, als representatie van het externe netwerk. We zullen Ptt en Phone voorzien van een gemeenschappelijke generatie: de Service, aangezien beide attributen en services gemeen hebben.

Probleem met het Call-object Bij de detaillering van het inputevent "hookUp"[4] stuiten we op een probleem: indien de hoorn wordt opgenomen als antwoord op een extern geprek van buiten, dienen de attentie- of belsignalen naar de andere toestellen te worden opgeheven. Indien echter de hoorn wordt opgenomen als antwoord op een intern gesprek is dit niet nodig. We dienen dus externe en interne gesprekken te kunnen onderscheiden.

Een mogelijke oplossing voor dit probleem is verschillende soorten Call-objecten te introduceren door specialisaties van de class Call te maken. We stellen voor (met in ons achterhoofd ook al het externe uitgaande gesprek) de volgende specialisaties te introduceren:

- InitialCall – zolang niet bekend is of we een intern of extern gesprek gaan beginnen.
- OriginatingCall – extern uitgaand gesprek
- InternalCall – intern gesprek
- AnswerCall – extern inkomend gesprek.

Het zal duidelijk zijn dat een aantal attributen en services voor elk van deze subclasses van Call gelijk zullen zijn. Echter: services, zoals "dialNumber" kunnen mogelijk verschillend zijn, omdat het initiëren van een gesprek verschillend is voor de verschillende specialisaties. Daarom hebben we tenslotte deze specialisaties ook ingevoerd. Uiteindelijk vinden we het OOA-diagram, zoals in figuur 3.2.

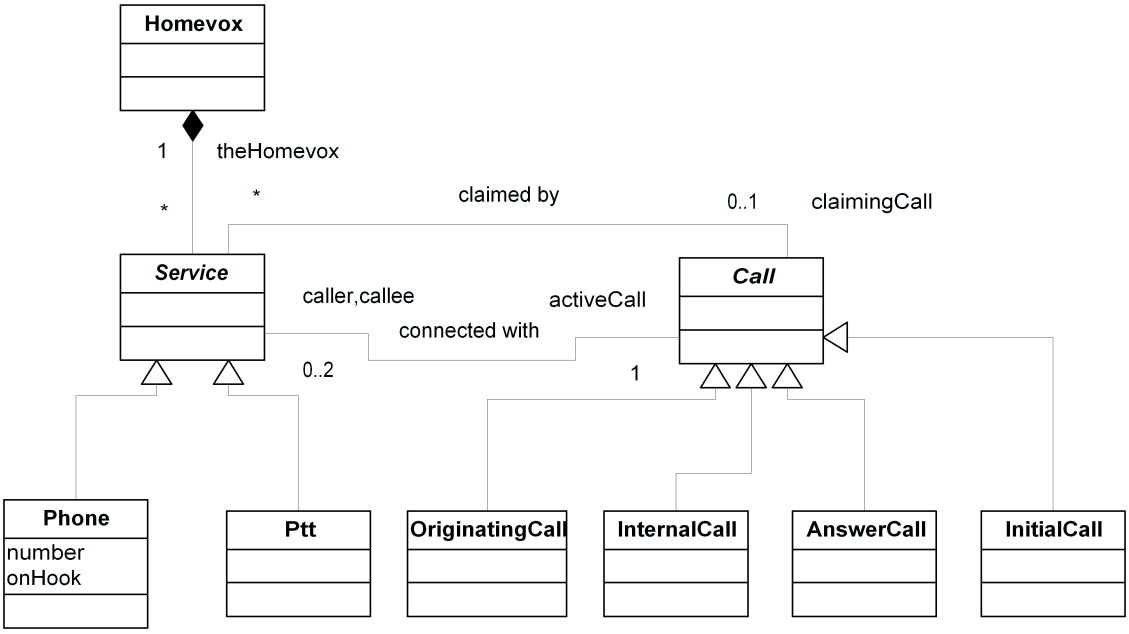
3.2 OOD: detailering inkomend gesprek

We zullen deze detaillering onder andere gebruiken om de eigenschappen van de nieuwe classes te bepalen. We beginnen met het event "ringFromPtt." Collaborators hierbij zijn het ptt-object, de homevox, de (answer)call, en alle Phone-objecten. Het Ptt-object kan het best dienen als

[2] De Homevox kan de claim ook opheffen door een time-out...

[3] We leggen hier de verantwoordelijkheid voor het afhandelen van de claim geheel bij de gebelde; een andere mogelijkheid is om de beller de toestand van het gebelde toestel steeds te laten inspecteren, maar dat lijkt om meerdere redenen minder gewenst.

[4] vanwege "hookUpCallee" in de use-case.



Figuur 3.2: Derde stap van de analyse: specialisaties van Call en toevoegen van Ptt en Phone als specialisaties van Service.

controller. Het gevolg van het input event is dat eerst een AnswerCall-object wordt gemaakt en dat daarna de noodzakelijke signalen naar de verschillende toestellen wordt gestuurd. Zie figuur 3.3.

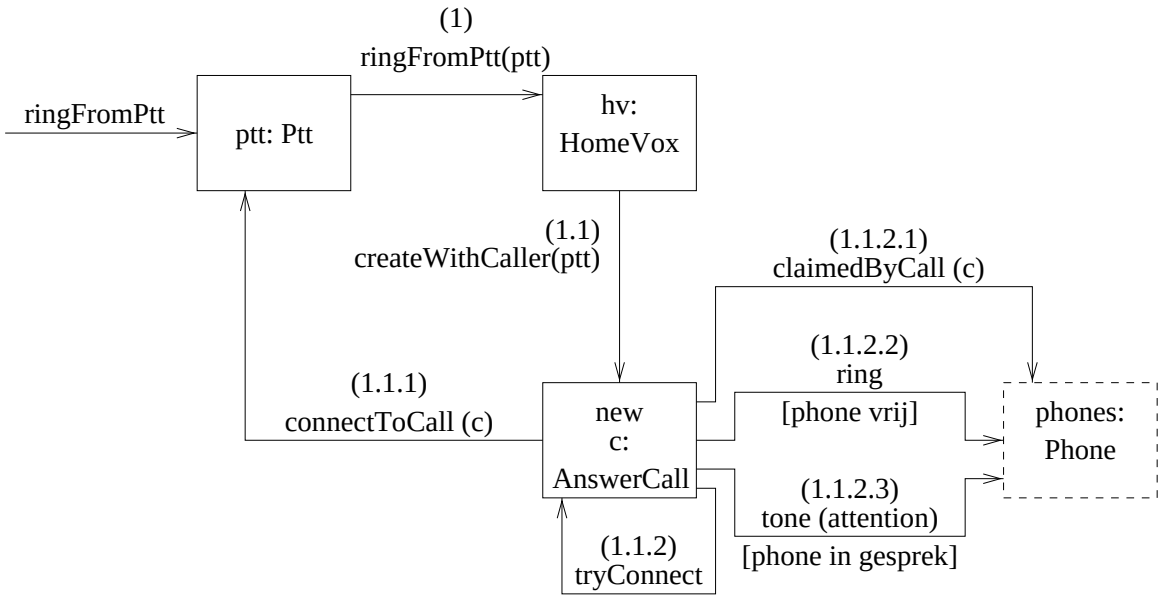
Het inputevent “hookUpCallee” hebben we voor de interne gesprekken ook al gehad. Het beantwoorden van een intern gesprek en een van buiten komend gesprek is echter analoog. Door het invoeren van verschillende typen call-objecten is ook steeds duidelijk wanneer wat gedaan moet worden.

Figuur 3.4 toont de situatie, zoals die ook (na introductie van claiming en active Calls en introductie van de vier verschillende Call-types) geldig is voor interne gesprekken.

Eén van de mogelijke alternatieven is dat de callee in eerste instantie een intern gesprek voert: de callee moet dan eerst dit gesprek beëindigen door de hoorn op de haak te leggen, waarna de bel overgaat en hij het inkomende gesprek kan nemen. De aangepaste “hookDown” staat in figuur 3.5.

3.3 OOA: uitgaand gesprek

use case Het belangrijkste scenario voor een uitgaand gesprek: De beller (een lokaal toestel) neemt de hoorn van de haak. De homevox geeft vervolgens dit toestel een kiestoon. Het lokale toestel kiest een 0. De homevox neemt de hoorn op van het externe netwerk, en verbindt het lokale toestel met het externe netwerk. Het externe netwerk geeft een kiestoon; het lokale toestel kiest een nummer; dit wordt door de homevox doorgegeven aan het externe netwerk. Het externe netwerk geeft vervolgens een vrijtoon (ringback tone). Daarna neemt het externe netwerk ‘de hoorn van de haak’, en maakt verbinding met de homevox. Nu vindt het gesprek plaats. Om het



1. meld homevox dat buitenlijn belt
1.1. creëer een nieuw answerCall-object
1.1.1. verbind call-object met ptt-object
1.1.2. probeer verbinding te maken met elk van de toestellen uit de collectie
1.1.2.1. claim elk toestel
1.1.2.2. en laat de bel overgaan voor elk vrij toestel
1.1.2.3. en een attentiesignaal voor elk bezet toestel

Figuur 3.3: Collaboration diagram voor de message "ringFromPtt"

gesprek te beëindigen legt het lokale toestel de hoorn op de haak; daarna legt de homevox de hoorn van het externe netwerk neer.

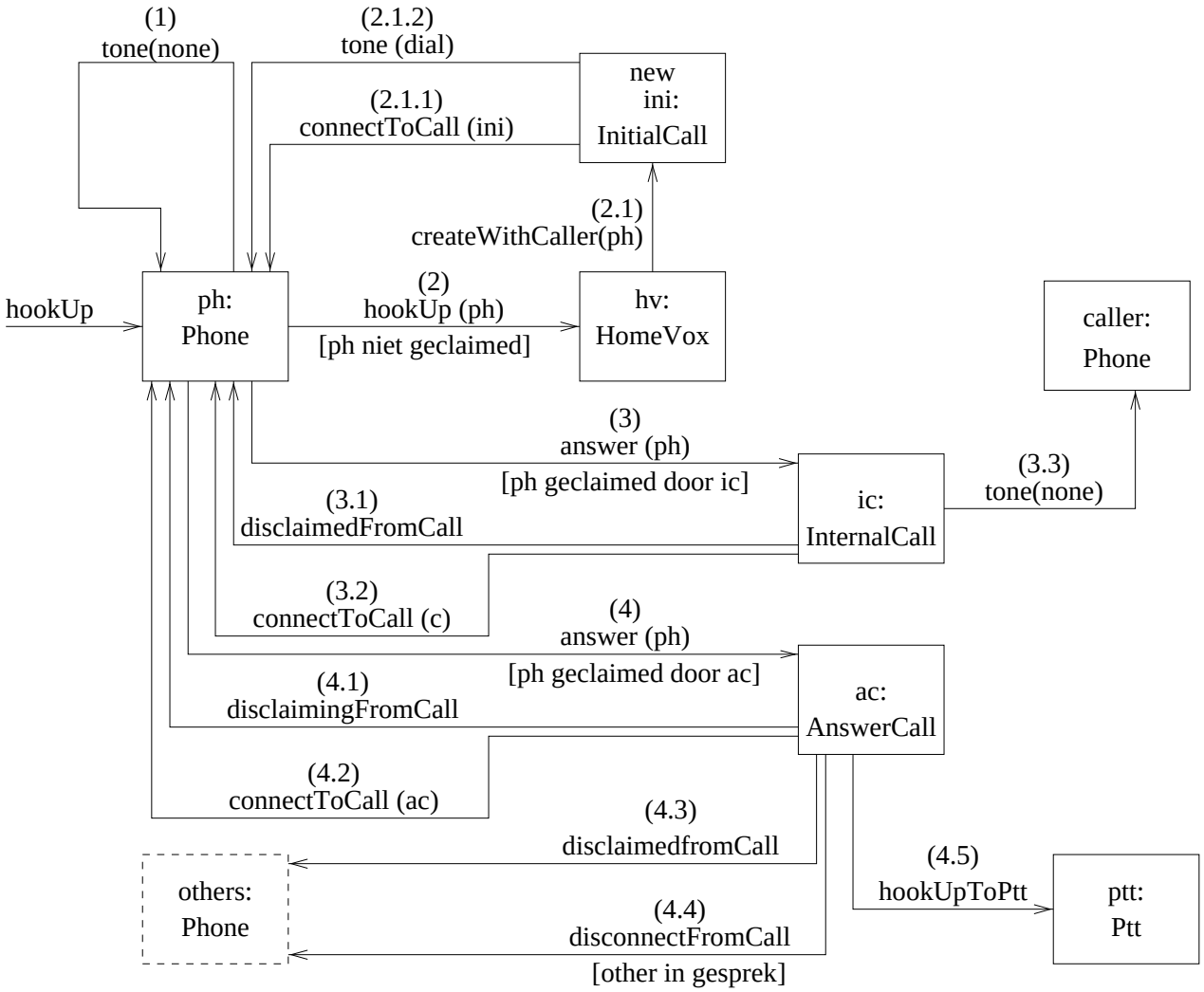
Ook dit scenario kunnen we weer in een diagram representeren, zie figuur 3.6. Ook nu geldt dat het opnemen van de hoorn aan de andere kant (de buitenlijn) geen bijzonder event is, maar uitsluitend door de beller kan worden begrepen, doordat hetgeen hij in de hoorn hoort verandert. Het opnemen van de hoorn door de gebelde is ook hier weer een event uit de serie "talk-receive."

Merk op dat het begin van een uitgaand gesprek identiek is aan dat van een lokaal gesprek. We kunnen pas concluderen dat het om een extern gesprek gaat na het kiezen van de '0'. We kunnen dit oplossen door het interne te vervangen door het externe gesprek (d.w.z.: in de administratie van het toestel, de referentie naar het actieve gesprek te veranderen.)

3.4 OOD: detaillering uitgaand gesprek

Het collaboration diagram van "hookUp" is al gegeven, zie figuur 3.5. Het volgende diagram is voor "dialNumber" en is gegeven in figuur 3.7.

In dit geval geven we de status van het gesprek dat aan een toestel verbonden is, *impliciet* weer door de class van dit gesprek: als de status verandert, moet het toestel ook met een ander gesprek-object verwijzen. Een alternatief is om de status expliciet te maken; bij het afhandelen van diensten als 'dialNumber' moeten we dan steeds deze toestand inspecteren, en de eigenlijke



1. annuleer vorige toon

ph heeft geen claiming call:

2. meldt homevox dat ph wil bellen
- 2.1. creëer InitialCall object
  - 2.1.1. verbind ph met InitialCall-object
  - 2.1.2. en geef de kiestoon

claiming call van ph is internalCall:

3. bevestig beantwoorden van gesprek
   - 3.1. annuleer claim bij ph
   - 3.2. en verbind ph met internalCall-object
   - 3.3. annuleer vrijtoon

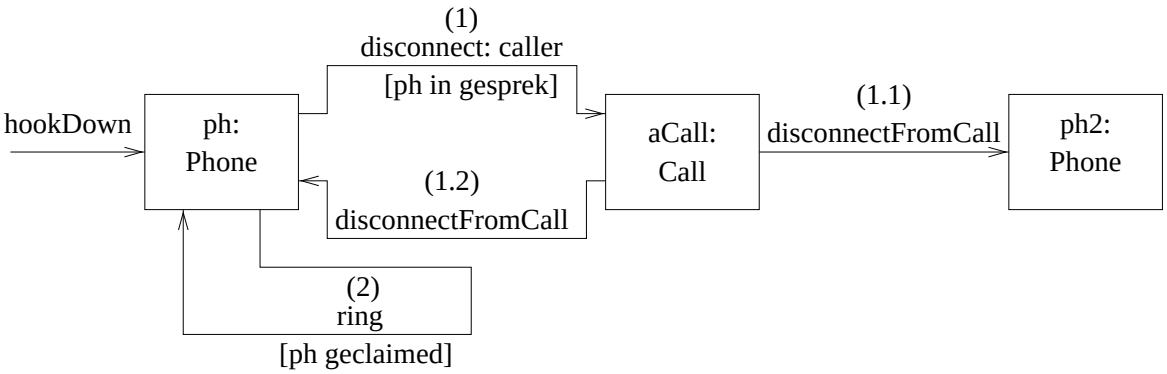
claiming call van ph is answerCall:

4. bevestig beantwoorden van gesprek
   - 4.1. annuleer claim bij ph
   - 4.2. en verbindt ph met answerCall-object
   - 4.3. annuleer claims bij andere toestellen
   - 4.4. verbreek overige gesprekken
   - 4.5. meld ptt aannemen van gesprek

Figuur 3.4: Collaboration diagram voor de message "hookUp" met aanpassingen t.g.v. introductie van claiming en active calls en de verschillende Call-specialisaties.

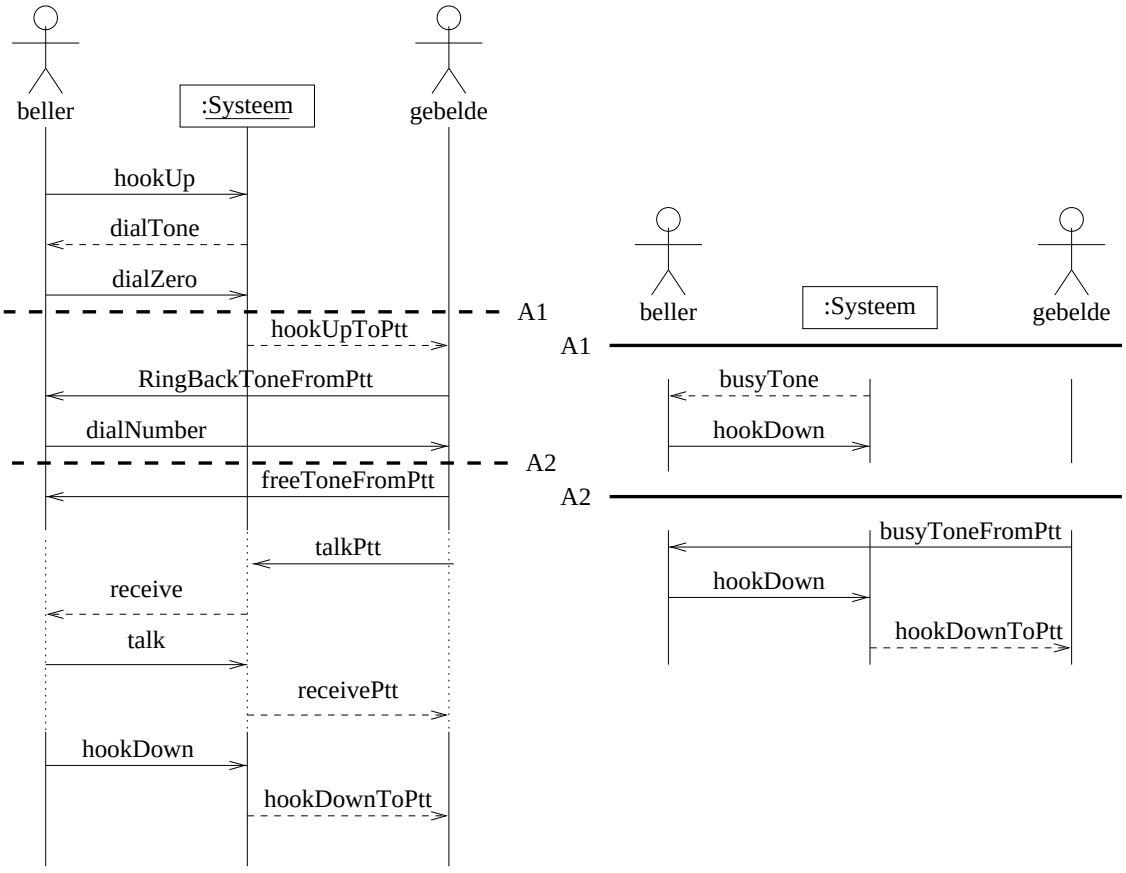
actie daarvan af laten hangen.

Bij de directe verbinding tussen 'beller' en 'ptt' kunnen de ptt-tonen direct doorgegeven worden naar de beller. We kunnen hier ook een tussenstap via de Homevox (Call) introduceren: dit heeft alleen zin als de Homevox op de een of andere manier iets met dit signaal doet,

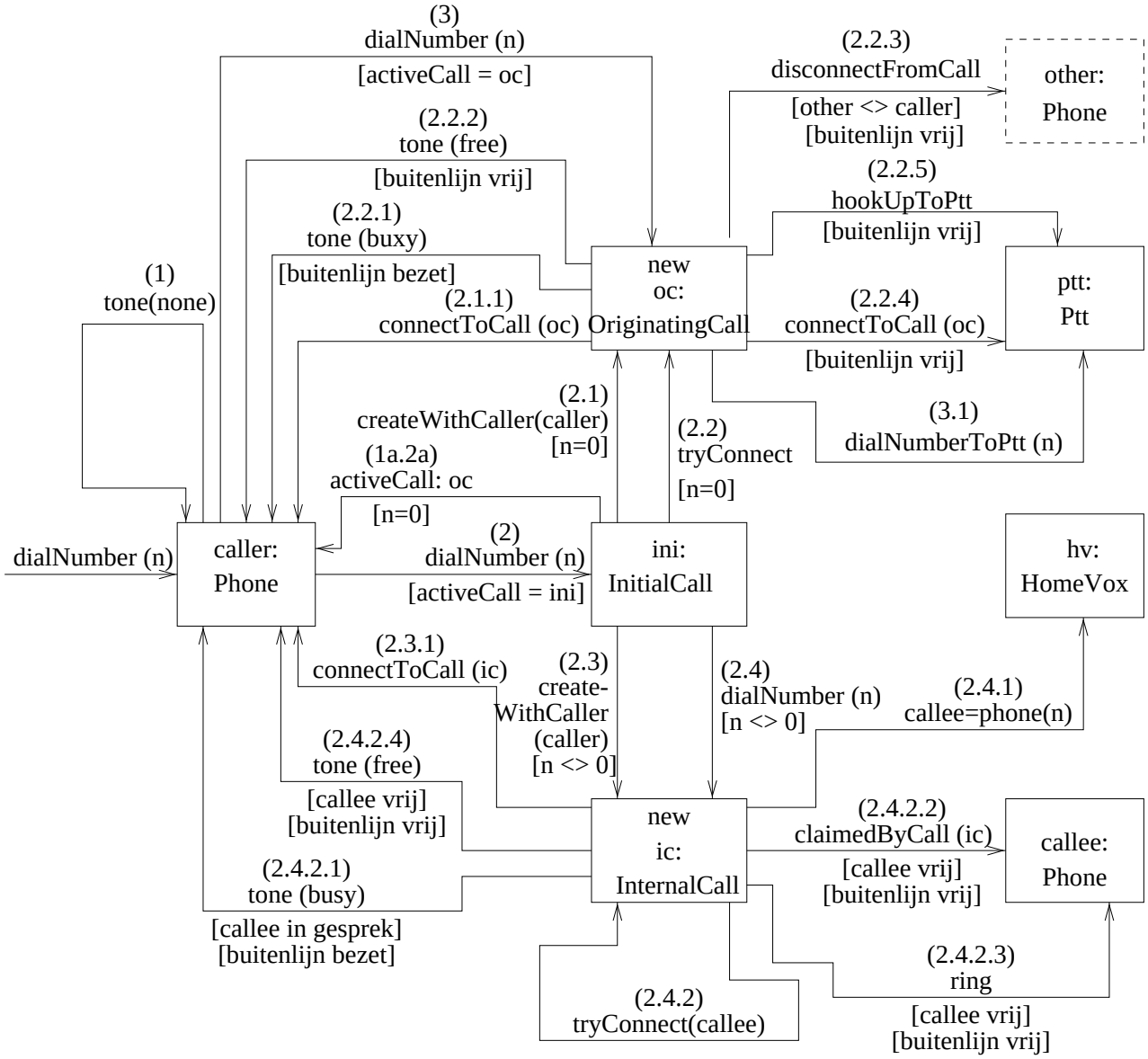


ph is actief bij gesprek betrokken:
1. meld call-object beëindiging van verbinding
1.1. meld callee beëindiging van verbinding
1.2. meld caller beëindiging van verbinding
ph is geclaimed:
2. laat de bel overgaan

Figuur 3.5: Collaboration diagram voor "hookDown" met aanpassing voor beantwoording attentiesignaal



Figuur 3.6: Scenario voor uitgaand gesprek



1. annuleer vorige toon.

activeCall van caller is een InitialCall:

2. meld ini dat nummer n is gedraaid

$n = 0$:

2.1. creëer OriginatingCall-object

2.1.1. verbind activeCall van caller met oc

2.2. probeer verbinding te maken met de buitenlijn

buitenlijn bezet:

2.2.1. geef bezettoon aan caller

buitenlijn vrij:

2.2.2. geef vrijtoon aan caller

2.2.3. verbreek verbinding met andere toestellen

2.2.4. maak verbinding met ptt

2.2.5. meld opnemen hoorn aan buitenlijn

$n \neq 0$:

2.3. creëer InternalCall-object

2.3.1. verbindt met caller

2.4. probeer verbinding te maken

2.4.1. zoek bij nummer horend object

2.4.2. en probeer hiermee verbinding te maken

callee is bezet of buitenlijn is bezet

2.4.2.1. geef bezettoon aan caller.

callee is vrij en geen buitenlijn:

2.4.2.2. claim de callee

2.4.2.3. geef belsignaal aan callee

2.4.2.4. geef vrijtoon aan caller

activeCall van caller is een OriginatingCall:

3. meldt oc het gedraaide cijfer

3.1. geef cijfer door aan buitenlijn.

Figuur 3.7: Collaboration diagram voor "dialNumber"

bijvoorbeeld een conversie (zoals bij de gekozen nummers).

3.4.1 Slotopmerkingen

In plaats van een enkel Call-Class&Object onderscheiden we nu: InitialCall, InternalCall, AnswerCall, OriginatingCall. Dit zijn alle specialisaties van Call; gemeenschappelijke eigenschappen zijn de instance connections (caller en callee), en operaties als `talk` en `receive`. De subclasses verschillen bijvoorbeeld in de manier waarop 'dialNumber' afgehandeld wordt. Voorts zijn Ptt en Phone als specialisaties opgenomen van de Abstract-Class Service. Een schematische representatie is terug te vinden in figuur 3.2.

De hierboven gegeven uitwerking is niet voldoende consequent met betrekking tot de vraag welk object nu precies verantwoordelijk is voor het verzorgen van het belsignaal en andere tonen. Een mogelijke juiste verdeling van de verantwoordelijkheden zou kunnen zijn dat de Call-objecten de tonen verzorgen en de Phone-objecten de belsignalen.

Eigenlijk is de hier uiteindelijk gevonden benadering niet meer geheel in overeenstemming met onze spelregels: het zou beter geweest zijn de Call-objecten via een rollenpatroon te definiëren. Onze werkwijze heeft ons echter uiteindelijk tot de nu gegeven oplossing gedreven. Er staan nu twee mogelijkheden open: (1) we laten het zoals het is (het werkt immers!),[5] of (2) we beschouwen wat we nu hebben als een prototype en passen het geheel aan in de volgende versie.[6]

[5] Althans dat hopen we na implementatie van het hier gepresenteerde ontwerp.

[6] In ons geval: in het volgende hoofdstuk.

4
Opnieuw: externe gesprekken

Zoals aan het eind van vorig hoofdstuk opgemerkt, verdient de analyse daar niet de schoonheidsprijs. Belangrijkste argument is dat de specialisatie van Call-objecten in een aantal soorten (intern, extern-inkomend en extern-uitgaand) slechts mogelijk is, door nog een vierde type te onderscheiden (initieel). Dit vierde type is noodzakelijk omdat bij een intern of een extern-uitgaand gesprek de eerste stappen gelijk zijn en pas na het intoetsen van het eerste cijfer duidelijk is welke van de twee gesprekken gevoerd zal worden. Op zich is een dergelijke overeenkomst nog niet zo erg, maar in het ontwerp blijkt dat dit eigenlijk alleen goed op te lossen is door na het intoetsen van dat eerste cijfer alsnog het goede Call-object aan te maken. Er is zelfs een complete copieeractie nodig van de gegevens in het IntialCall-object naar het InternalCall- of OriginatingCall-object.

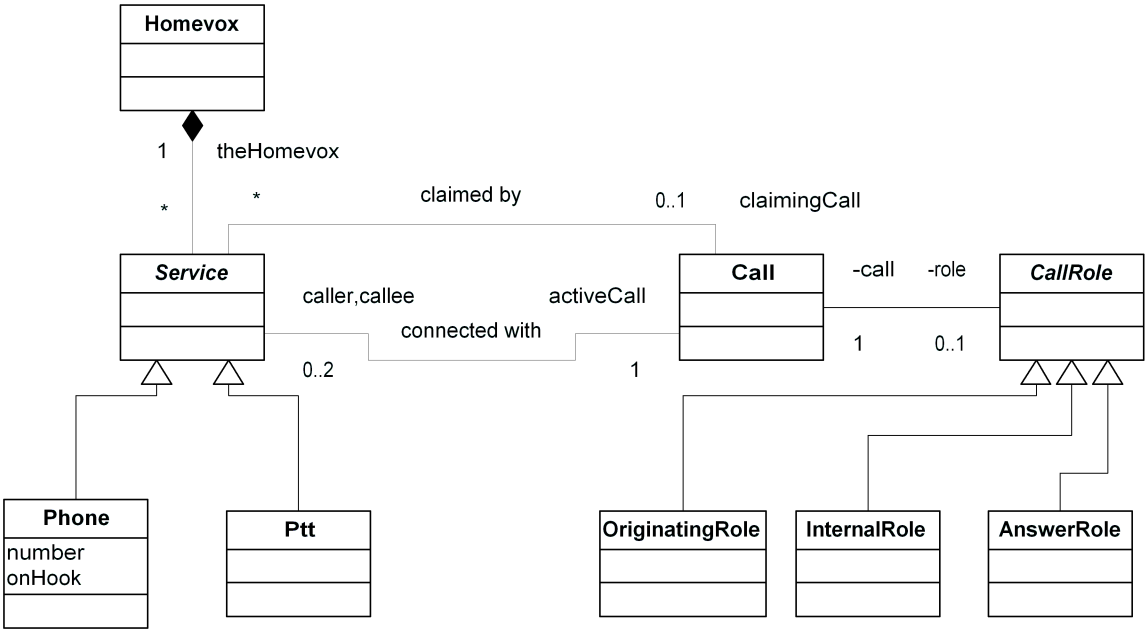
Copieren is in OO in zekere zin een doodzonde! Maar er is een alternatief. In feite gebruikten we de specialisaties van het Call-object in de vorige analyse om aan te geven dat sommige acties voor het ene soort Call anders zouden moeten zijn dan voor het andere soort (een claim is voor bijvoorbeeld een AnswerCall anders dan voor een InternalCall enz.). De attributen van al dit soort Call-objecten is wel steeds gelijk. Of, om het anders te formuleren: alle soorten Call-objecten delen de verantwoordelijkheid om de verbinding te administreren, maar verdelen de verantwoordelijkheden voor het leggen van zo'n verbinding. Dat laatste is afhankelijk van het type van de Call.

Voor de hand liggend in dergelijke gevallen is te kiezen voor het rollenpatroon. De administratie bevindt zich dan in de "centrale" en algemene Call-objecten. De per rol afwijkende handelingen stoppen we in de verschillende rol-objecten. Ziehier het ontstaan van de alternatieve analyse, zoals gegeven in figuur 4.1. Bijkomend voordeel blijkt nu te zijn, dat zolang nog niet duidelijk is of een Call-object de rol van intern of van originating zal spelen, het centrale Call-object alle handelingen zelf kan verrichten. Daarnaast zijn er geen copieeracties nodig, zodra de rol bekend is. Er behoeft slechts een rol-object te worden gecreeerd en dit rol-object kan vervolgens de specifiek voor die rol noodzakelijke of afwijkende acties voor zijn rekening nemen.

4.1 Aangepaste collaboration diagrammen

De zojuist geïntroduceerde aangepaste analyse i.v.m. het rollenpatroon voor de Call resulteert vanzelfsprekend in een aantal aangepaste collaboration diagrammen. De belangrijkste geven we in dit hoofdstuk opnieuw weer.

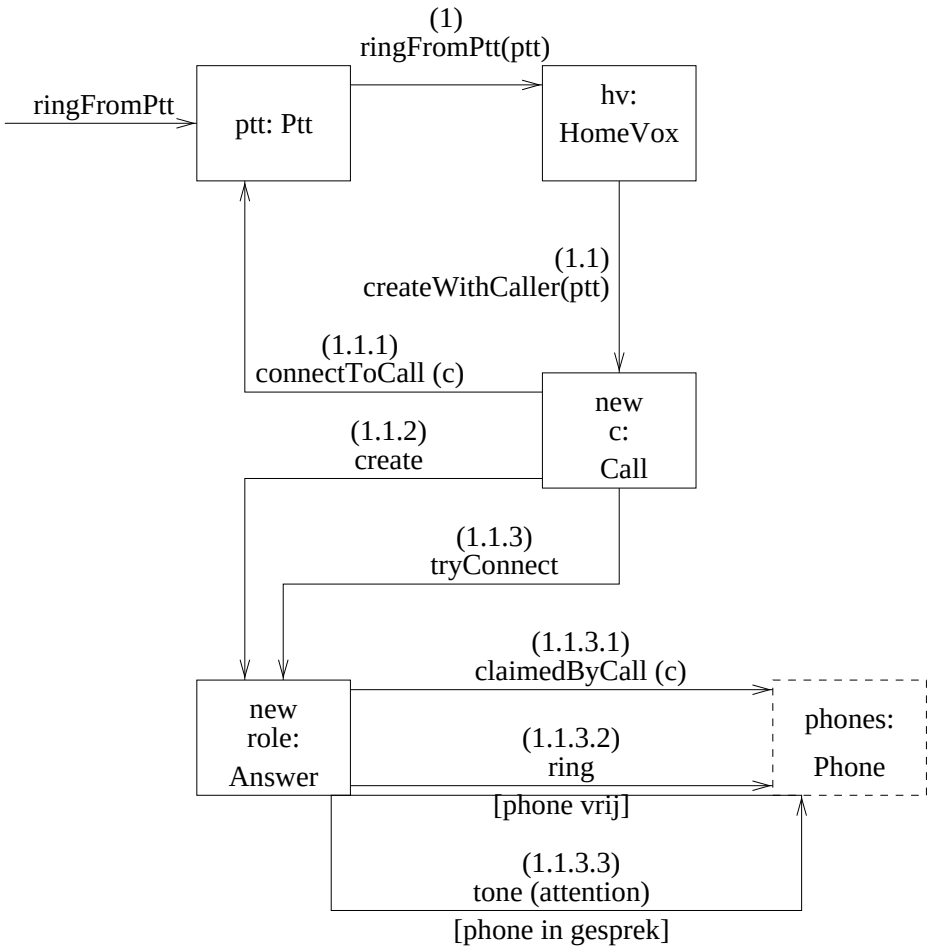
Opvallend is, dat de collaboration diagrammen door introductie van het rollenpatroon in het algemeen wat eenvoudiger worden. Nadeel is dat sommige objecten wat minder gemakkelijk



Figuur 4.1: Alternatieve analyse van de homevox door introductie van rollenpatroon voor het Call-object

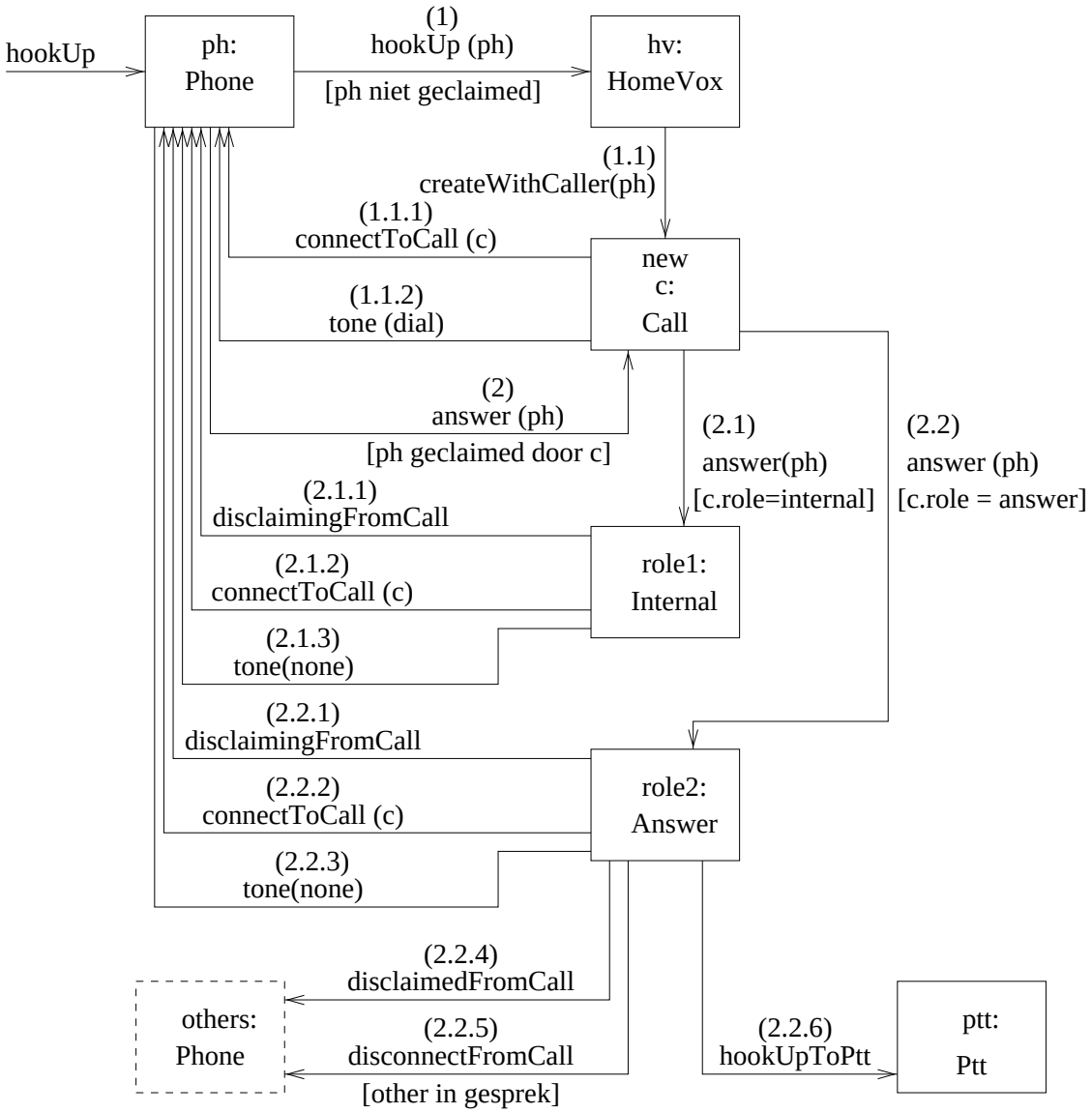
zijn terug te vinden. Zo dient een AnswerCall-Role-object het ptt-object terug te vinden via het Call-object, waar de rol bijhoort, terwijl deze op zijn beurt weer het homeVox-object nodig heeft om het ptt-object te kunnen vinden. In het algemeen verkrijgen we soms een extra indirectie door het introduceren van rollen.

Door deze extra indirectie komen sommige objecten “wel erg ver weg te liggen.” Het is dan ook het overwegen waard, extra attributen in de classes op te nemen die direct naar de betreffende objecten wijzen. Zo hoeven we minder vaak lange paden af te lopen om een bepaald object terug te vinden. Zouden we visibility diagrammen maken naar aanleiding van de onderstaande collaboration diagrammen dan zouden we tot dezelfde conclusie komen.



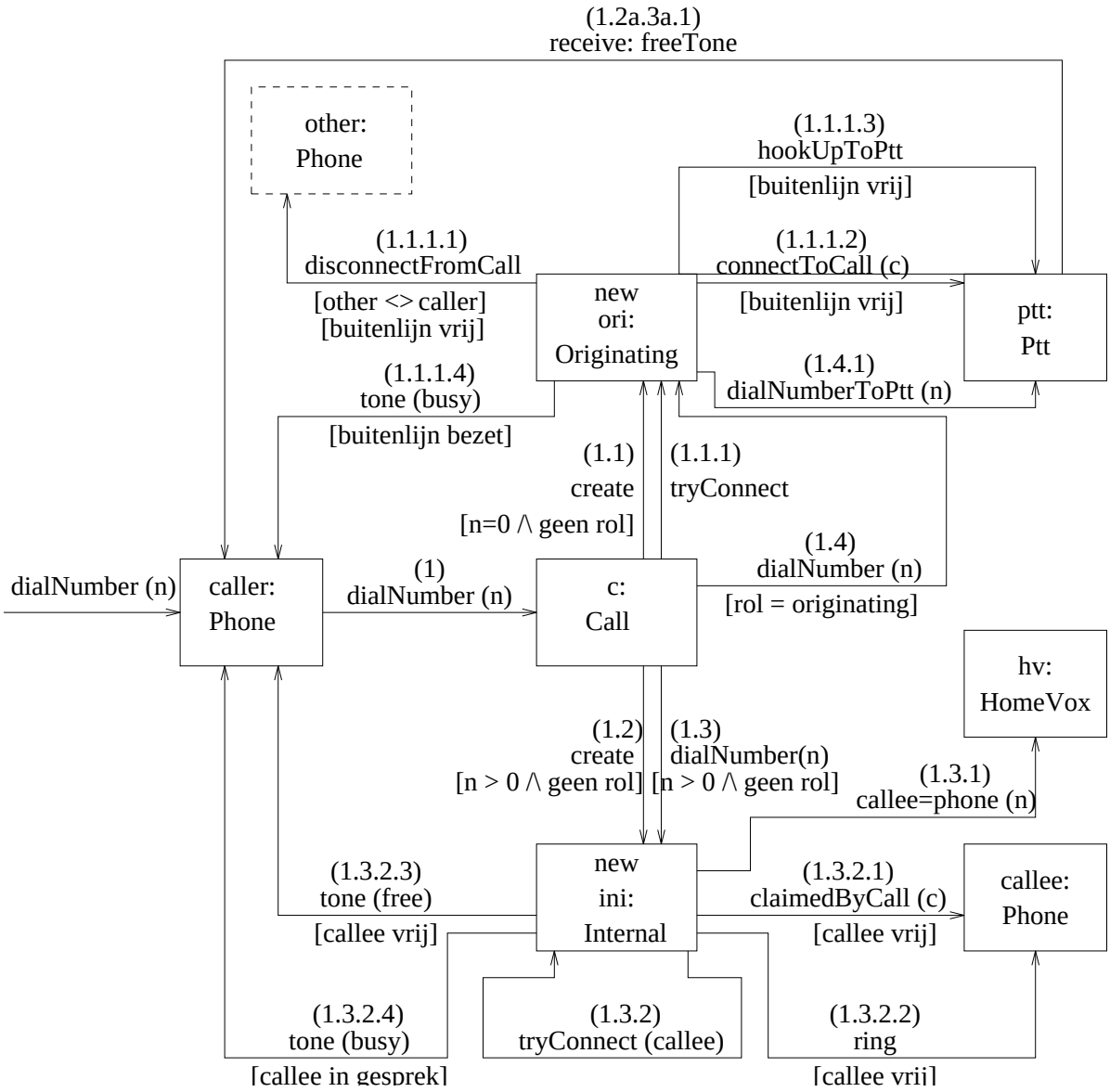
1. meld homevox dat buitenlijn belt
1.1. creëer een nieuw Call-object
1.1.1 verbind call-object met ptt-object
1.1.2. creëer Answer-object voor rol
1.1.3. probeer verbinding te maken met elk van de toestellen uit de collectie
1.1.3.1. claim elk toestel
1.1.3.2. en laat de bel overgaan voor elk vrij toestel
1.1.3.3. en een attentiesignaal voor elk bezet toestel

Figuur 4.2: Collaboration diagram voor de message "ringFromPtt"



ph heeft geen claiming call:
1. meldt homevox dat ph wil bellen
1.1. creëer Call object
1.1.1. verbind ph met InitialCall-object
1.1.2. en geef de kiestoon
ph is geclaimed:
2. bevestig beantwoorden van gesprek
2.$\frac{1}{2}$. bevestig beantwoorden van het gesprek
2.$\frac{1}{2}$.1. annuleer de claim op ph
2.$\frac{1}{2}$.2. verbind ph met c
2.$\frac{1}{2}$.3. annuleer toon
rol van claimedCall is answer:
2.2.4 annuleer alle andere claims
2.2.5 verbreek nog lopende gesprekken
2.2.6 meld ptt aannemen van gesprek

Figuur 4.3: Collaboration diagram voor de message "hookUp."



1. meld call dat nummer n is gedraaid

$n = 0$ en call heeft nog geen rol:

1.1. creëer Originating-object

1.1.1. en probeer verbinding met buitenlijn te maken

buitenlijn vrij:

1.1.1.1. verbreek overige verbindingen

1.1.1.2. verbind buitenlijn met caller

1.1.1.3. meld buitenlijn opnemen van de hoorn

buitenlijn in gesprek:

1.1.1.4 geef bezettoon aan caller

$n \neq 0$ en call heeft nog geen rol:

1.2. creëer Internal-object

1.3. en geef nummer door aan Internal-role

1.3.1. bepaal callee-object

1.3.2. en probeer verbinding te maken met callee

callee is vrij:

1.3.2.1. claim callee

1.3.2.2. geef belsignaal aan callee

1.3.2.3. geef vrijtoon aan caller (direct via buitenlijn)

callee is in gesprek

1.3.2.4. geef bezettoon aan caller

call heeft een rol

1.4. stuur nummer door naar rol-object

1.4.1. stuur nummer door naar Ptt

Figuur 4.4: Collaboration diagram voor "dialNumber"

5
Verdere uitwerking

Bij dit practicum is het de bedoeling om de Homevox-software te construeren conform de (afzonderlijk) gegeven requirements (zie hoofdstuk 1). Omdat deze opdracht nogal omvangrijk is, vindt de uitwerking plaats volgens de methode van 'incremental delivery'.

5.1 Homevox besturing

Bij dit onderdeel staat de besturing van de Homevox hardware centraal. In eerste instantie is ook dit een erg omvangrijk probleem; we proberen dit op te lossen door een vereenvoudigde versie van het probleem eerst op een hoog abstractieniveau op te lossen. Vervolgens kunnen we op ditzelfde niveau een completere (complexere) versie van het probleem aanpakken; tenslotte kunnen we het abstractieniveau verlagen tot het niveau van de hardware-aansturing.

5.1.1 Homevox1: alleen interne gesprekken

Dit gedeelte is gegeven en besproken in hoofdstuk 2. De bij het practicum behorende sources van deze uitwerking in de directory

```
intern\
```

Naast de in hoofdstuk 2 globaal beschreven sources zijn als extra toegevoegd de volgende files:

```
intern\HomeVoxContainer.java
intern\PhoneView.java
```

Door `HomeVoxContainer` te vertalen en uit te voeren ontstaan vier windows (beschreven in `PhoneView`), elk een telefoon voorstellend. Hiermee zijn eenvoudig allerlei testscenario's te proberen. Zoals je uit de namen zou kunnen afleiden, is een object uit de class `PhoneView` een view die hoort bij het model-object `Phone`, zoals beschreven in het MVC-concept. Aan de op deze manier geïmplementeerde HIC behoeft verder niets te worden aangepast en/of toegevoegd.

5.1.2 Homevox2: uitbreiding met een extern gesprek

Geef een uitbreiding van Homevox1 waarbij ook externe gesprekken (via de PTT-lijn) mogelijk zijn: zowel inkomende als uitgaande.

Mogelijk scenario's voor de beide mogelijkheden, alsmede een analyse zijn reeds gegeven en besproken in hoofdstuk 4.

Een implementatie volgens het diagram in figuur 4.1 is te vinden in de directory

`extern\`

De HIC is nu ook uitgebreid met een `PttView`, waarmee de buitenlijn kan worden gesimuleerd. Alles hangt weer aanelkaar via de class `HomeVoxContainer`. In zowel de class `Call` als in `Service` is een methode `show` beschikbaar, die berichten op de uitvoer schrijft en zonodig ook via het MVC-mechanisme verstuurt naar de overeenkomstige view. In de subclasses van `CallRole` kan deze methode eenvoudig worden gebruikt via

```
call.show("some message") ;
```

Naast `show` bestaat ook `testshow`, die de in de parameter meegegeven mededeling eveneens afdrukt, maar niet naar de bijbehorende view stuurt.

5.1.3 Homevox3: uitbreiding met ruggespraak

Uitbreiding: ruggespraak tijdens extern gesprek, met ander lokaal toestel. Zie de requirements voor de juiste gebruikersinterface.

5.1.4 Homevox4: volledige functionaliteit

Zie de requirements voor de juiste gebruikersinterface.

5.1.5 Homevox1a: gedetailleerde versie van Homevox1

Geef een gedetailleerde versie van Homevox1, waarmee rechtstreeks de Homevox hardware (-simulator) aangestuurd wordt. (De hardware kan in deze versie nog bestaan uit dummy objecten, die slechts melden dat bepaalde acties met ze zijn uitgevoerd (bijvoorbeeld een schakelaar die slechts meldt aan of uitgezet te zijn, enz.)

5.1.6 Homevox hardware simulator

Om de Homevox besturings-software te kunnen testen moet een simulator van de Homevox hardware geconstrueerd worden. Deze is beschikbaar en in het volgende hoofdstuk nader beschreven.

5.2 De eindopdracht

Als eindopdracht bij de cursus OO-technieken is het volgende de bedoeling:

1. maak een analyse en ontwerp voor HomeVox 3 en
2. implementeer dit

Zie verder het aparte opgaveblad voor wijze van inleveren e.d.

6
HomeVox hardware simulator

In dit gedeelte proberen we een oplossing te vinden voor de simulatie van de HomeVox hardware. Bij deze simulatie spelen drie soorten interacties een rol:

- met het interface met de 'Human Interaction Component': om de toestand van de hardware zichtbaar te maken en om de hardware te manipuleren.
- met het interface met de Processor (ofwel, met de besturingssoftware daarop).
- tussen de hardware-componenten onderling.

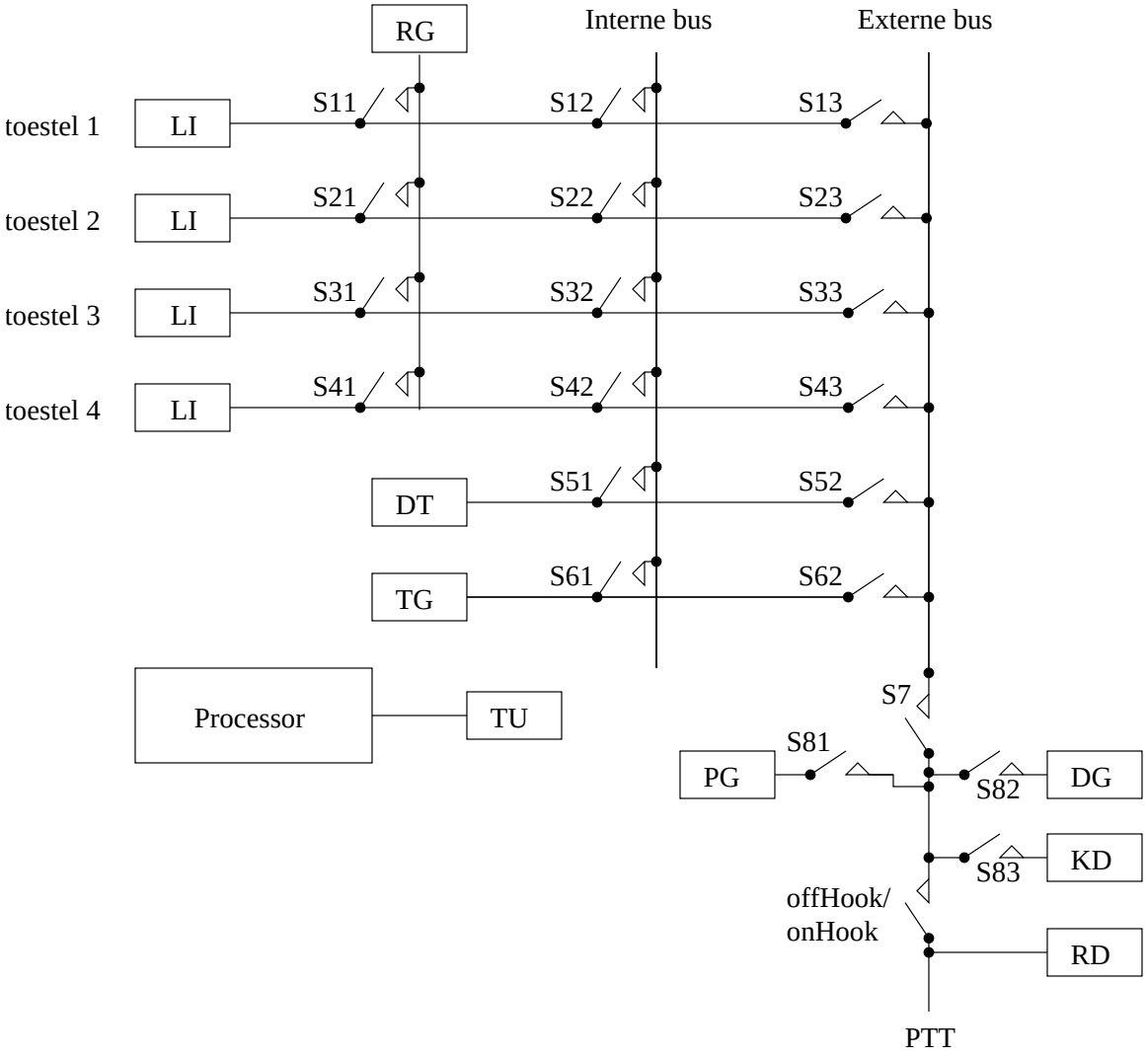
De interacties met de buitenwereld bestaan uit specifieke messages (events) die geassocieerd zijn met specifieke hardware-componenten. Detectie van het van de haak nemen van de hoorn, bijvoorbeeld dient te resulteren in het sturen van de message `offHook` naar het overeenkomstige `Phone`-object. In dit geval kunnen we het normale 'message' mechanisme gebruiken (als in Smalltalk).

De interacties tussen de componenten onderling spelen zich af via het netwerk. Dit netwerk kan signalen bevatten (signalen op één of meerdere draden). We zullen signalen op een draad voorstellen door middel van events, die het begin en het eind van zo'n signaal voorstellen. Bijvoorbeeld een belsignaal op de draad wordt gerepresenteerd door een event start-bel en een event eind-bel. Beide events geven een verandering aan de toestand aan: resp. een belsignaal komt op de draad en een belsignaal gaat weer van de draad. In discrete simulatie is dit de gebruikelijke manier om gedurige acties weer te geven, namelijk door het starten en het stoppen van zo'n actie als discrete gebeurtenis te zien.

Omdat via het hardware-netwerk allerlei events verspreid kunnen worden, kunnen we niet gebruik maken van specifieke messages. Bovendien worden de messages niet direct gericht aan een bepaalde receiver: het netwerk verzorgt eigenlijk een broadcast. Dit betekent dat we een soort 'generic message' of 'generic event' moeten introduceren, en alle componenten met een handler voor dit soort messages of events moeten uitrusten: de interpretatie van deze messages moet dan dynamisch plaatsvinden.

Bij deze interpretatie zijn er drie mogelijkheden: (i) de receiver heeft een specifieke reactie, (ii) de receiver negeert de message, of (iii) de receiver reageert met een foutmelding.

Er zijn twee manieren om dit soort events te genereren: (i) een component genereert een event, en 'broadcast' deze via het netwerk naar de verbonden andere componenten; (ii) een schakelaar wordt omgezet, waardoor voor sommige componenten het begin of het einde van een signaal (event) kan ontstaan. In beide gevallen moet het netwerk zorgen voor de verdere distributie van de events, wat uiteindelijk resulteert in het ontvangen hiervan door de componenten die momenteel verbonden zijn (in de hierna beschreven implementatie is een deel van deze distributie door de afzonderlijke componenten overgenomen).



Figuur 6.1: Blockschema van de HomeVox hardware

We onderzoeken een aantal mechanismen voor de generatie en distributie van deze events:

1. het signaal-model: in dit geval associëren we met een draad (‘net’) een signaal; eigenlijk is dit de toestand van de net. Dit signaal wordt veroorzaakt door events die het begin of het einde van een signaal aangeven. Eenzelfde net kan tegelijkertijd meerdere signalen bevatten (denk bijvoorbeeld aan modulatie). In sommige gevallen (niet van belang voor de HomeVox) mogen bepaalde signalen niet tegelijkertijd aanwezig zijn.
2. het connectie-model: in dit geval staat de verbindingsfunctie van het netwerk voorop. Het omzetten van een schakelaar resulteert uiteindelijk in het koppelen (of loskoppelen) van twee componenten (of, algemener, van twee verzamelingen componenten). Deze componenten wisselen vervolgens zelf eventueel hun events uit. De toestand van een draad speelt hier geen rol.

6.1 het signaal-model

In dit geval speelt de draad een belangrijke rol. Deze weet welk signaal er op de draad staat (en dat kunnen er meerdere zijn). De toestand van een draad laat zich beschrijven door de signalen op de draad. Een generator kan een signaal aanzetten (d.i., op de draad zetten) en uitzetten (d.i., er weer af halen). Elke verandering aan de draad wordt door de eraan hangende componenten waargenomen (de draad doet een broadcast van elke toestandsverandering aan alle aan de draad hangende componenten). Detector-componenten kunnen eventueel op de broadcast reageren.
Schakelaars spelen hier een belangrijke rol. Bij openen en sluiten van schakelaars kunnen signalen op een draad worden gestopt en gestart. Een op een draad aanwezig signaal wordt, bij sluiten van een schakelaar die met zo'n draad verbonden is, doorgegeven aan de draad van het andere aansluitpunt van de schakelaar. Deze laatste draad moet dan alsnog een broadcast doen van dit signaal naar alle componenten aan deze draad. Evenzo vindt een broadcast plaats van een beëindigd signaal als een schakelaar wordt geopend en de draad een signaal bevat dat afkomstig is van een generator van de andere, nu losgekoppelde, draad.
Een draad dient dus weet te hebben van zijn toestand (signalen op de draad), van alle met deze draad verbonden componenten en of een signaal op de draad door een genererende component aan die draad veroorzaakt is (dit laatste is van belang bij openen van een schakelaar).

Problemen bij dit model zijn vergelijkbaar met die bij het connectie model. We zullen dan ook slechts van deze laatste een volledige uitwerking geven.

6.2 Het connectie-model

In dit geval houdt elke draad bij welke componenten ermee verboden zijn, zodat wanneer een component een bepaalde event genereert (als begin of einde van een signaal), deze direct bij de relevante componenten afgeleverd kan worden. De componenten hebben de verantwoordelijkheid van distribueren van events aan andere componenten.

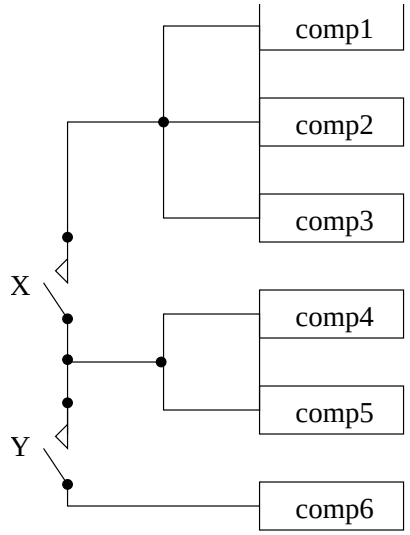
Het omzetten van een schakelaar resulteert uiteindelijk in het koppelen (of ontkoppelen) van twee (verzamelingen) componenten; als reactie op `connectTo: x` genereert een component een event, alleen afhankelijk van de toestand van de component, niet van de receiver. Vervolgens wordt dit event gestuurd naar de ontvanger. Hierbij moeten we erom denken dat het koppelen van twee componenten een symmetrisch gebeuren is: zowel `a connectTo: b` als `b connectTo: a` is nodig.

Het netwerk geeft eigenlijk aan welke componenten op een bepaald moment onderling verbonden zijn: dit is een N-M relatie. Door het omzetten van een schakelaar verandert deze relatie; wij zijn ook geïnteresseerd in de resulterende veranderingen in de 'verbindings'-relatie.

Het netwerk bestaat uit een aantal schakelaars en een aantal draden (nets); aan dit netwerk zijn de componenten verbonden. Enerzijds is er de *vaste structuur* van het netwerk: welke onderdelen *direct* met welke andere onderdelen verbonden zijn. Anderzijds is er de *huidige toestand* van het netwerk, gegeven door de stand van de schakelaars. Twee direct verbonden punten in de structuur heten 'aangesloten' (attached). Door het sluiten van een schakelaar kunnen componenten 'gekoppeld' (connected) worden, of 'ontkoppeld' (disconnected).

6.2.1 Het opbouwen van het netwerk

Voor het opbouwen van het netwerk, gegeven de componenten, moeten we de nets creëren. Vervolgens sluiten we de componenten aan op de nets.
Bijvoorbeeld het netwerk uit figuur 6.2 kunnen we alsvolgt opbouwen:



Figuur 6.2: Een voorbeeld netwerk

```
Net A = new Net () ;
Net B = new Net () ;
Net C = new Net () ;
A.attach(comp1) ; A.attach(comp2) ; A.attach(comp3) ;
B.attach(comp4) ; B.attach(comp4) ;
C.attach(comp6) ;
```

6.2.2 Schakelaar

Een schakelaar heeft twee aansluitpunten, te noemen L en R. Een aansluitpunt is aangesloten op een enkel net (draad); op een net kan een *verzameling* componenten en schakelaars aangesloten zijn.

Op een schakelaar kunnen we functies definiëren om de aangesloten component(en) van een aansluitpunt op te vragen, en om de gekoppelde componenten van dat punt op te vragen. Daarnaast heeft een schakelaar natuurlijk operaties om de stand van de schakelaar te veranderen (en misschien om deze op te vragen).

De aangesloten componenten (en schakelaars) van een aansluitpunt volgen via het aangesloten net. De momenteel gekoppelde componenten van een aansluitpunt (d.w.z. de componenten die ‘door de schakelaar heen’ te zien zijn) volgen via de schakelaars en componenten aangesloten op het andere aansluitpunt.

Uit bovenstaande volgt dat we een schakelaar kunnen opvatten als twee componenten (beide van class `Switch`). De ene component is aansluitpunt L, de andere R. Als extra attribuut voegen we aan zo’n paar schakelaars een verwijzing toe naar de partner. In feite introduceren we hier dus een soort tweeling-objecten (of twin-objects): elke schakelaar wordt gerepresenteerd door twee objecten, één voor elke deel van de schakelaar, waarbij het ene deel niet kan bestaan zonder het andere. Een object uit de class `Switch` heeft twee attributen: een verwijzing naar de andere helft van het twin-object: `partner` en een attribuut dat aangeeft of de schakelaar gesloten is, dat wil zeggen of de twin-objecten met elkaar verbonden zijn, `connected`.

Voor bovenstaande figuur krijgen we nu:

```
Switch xL = new Switch () ;
Switch xR = new Switch () ;
xR.setPartner (xL) ; xL.setPartner (xR) ;
Switch yL = new Switch () ;
Switch yR = new Switch () ;
yR.setPartner (yL) ; yL.setPartner (yR) ;
A.attach(xL) ;
B.attach(xR) ; B.attach(yL) ;
C.attach(yR) ;
```

In plaats van de omslachtige methode van het creëren van twee nieuwe objecten en het leggen van verwijzingen naar elkaar is het verstandiger één constructor te maken met als parameters beide netten, die alle initialisaties doet:

```
Switch x = new Switch (A, B) ;
Switch y = new Switch (B, C) ;
```

De constructor ziet er alsvolgt uit:[1]

```
public Switch (Net aNet, Net bNet) {
   partner = new Switch() ;
   partner.setPartner (this) ;
   theNet = aNet ;
   aNet.attach (this) ;
   partner.setNet (bNet) ;
   bNet.attach (partner) ;
   id = "switch-"+aNet.getId() ;
   partner.setId ("switch-"+bNet.getId()) ;
}
```

We kunnen nu eenvoudig schakelaars openen en sluiten, mits we intern steeds de overeenkomstige acties uitvoeren voor het ontvangende object (bijvoorbeeld `x`) alsook voor zijn partner.

6.3 Een analyse

Na bovenstaande uiteenzetting bespreken we nog een keer, en nu meer precies, de verschillende onderdelen van de hardware.[2]

6.3.1 Net

Een net (draad) bestaat uit een verzameling permanent aangesloten componenten en schakelaars. (In sommige contexten heet zoiets ook wel een *bus*.) Er zijn operaties om de structuur van het netwerk op te bouwen; voor een net betekent dit: het invullen van de verzameling aangesloten componenten. Deze verzameling wordt alleen geïnitialiseerd: tijdens de simulatie verandert

[1] Het attribuut `connected` wordt bij declaratie al `false` gemaakt.

[2] Naast de analyse zullen we zo hier en daar al ontwerpbeslissingen meenemen, bijvoorbeeld door de attributen behorende bij de connecties te definiëren, en attributen, die het rekenwerk vergemakkelijken.

deze niet meer. We onderscheiden daarom twee attributen: `attachedComponents` – de componenten die vast aan dit net gekoppeld zijn en `connectedComponents` – de componenten die door dichtgezette schakelaars (tijdelijk) aan dit net verbonden zijn. Dit laatste attribuut is redundant (de verzameling verbonden componenten kan eenvoudig elke keer opnieuw worden bepaald), maar zal in de ontwerpfase worden toegevoegd om de berekeningen efficienter te maken.

Methoden Een net heeft een methode nodig om een component aan het net te kunnen toevoegen, `attach`, alsmede om een component of een verzameling componenten tijdelijk aan een net te kunnen toevoegen, indien een schakelaar wordt gesloten, `connectTo`, alsook om die verzameling weer te kunnen ontkoppelen, indien de schakelaar weer open wordt gezet, `disconnectFrom`.
Daarnaast is het noodzakelijk dat een net signalen kan verspreiden, die één van de aangesloten componenten kan opstarten. We hebben daarom nodig de methoden `startSignal` en `endSignal`. Een signaal dat één van de aangesloten componenten nu begint, of eindigt, kan via deze methoden worden doorgegeven aan elke andere aangesloten component. Dit is het idee van broadcasting, zoals hiervoor aangegeven. Op net-niveau worden slechts symbolen doorgeven. Op componentniveau worden ontvangen symbolen omgezet in messages, of ze worden genegeerd.

6.3.2 Hardware component

Een component[3] heeft 1 aansluitpunt. (Dit is voor ons geval voldoende; desnoods voegen we extra componenten toe die alleen als aansluitpunt voor andere componenten fungeren.) Dit aansluitpunt is aangesloten op een net. (Hier is weer sprake van een relatie; in dit geval 1-N.) Een component is verantwoordelijk voor het gedrag en de interfacing van de hardware. Het gedrag bestaat uit connecting en disconnecting van/met een andere component en zenden en ontvangen van signalen. Een component is permanent verbonden met een net. Een schakelaar wordt als een bijzondere component gezien, met de mogelijkheid netten met elkaar te verbinden en weer te ontkoppelen. Als attributen voor een algemene component is nodig het net waarmee deze component verbonden is: `theNet`, en een identificatie: `id`.

Methoden Een component moet met een net verbonden kunnen worden (gesoldeerd), hiervoor is `attachToNet`. Componenten kunnen met elkaar verbonden worden en deze verbinding kan weer teniet gedaan worden via `connectTo` en `disconnectFrom`. Tot slot kan een component signalen ontvangen via `startSignal` en `endSignal`. Deze methoden worden op specialisatie-niveau geherdefinieerd.

6.3.3 Switch

De attributen van een schakelaar zijn al eerder ter sprake gekomen.

Methoden Belangrijke methoden voor een schakelaar zijn `open` en `close` (zie onder). Bij sluiten van een schakelaar dienen eventueel nog op het net aanwezige signalen alsnog verder verspreid te worden. Aangezien het net zelf geen weet heeft van deze signalen, moet, bij het

[3] In de tekst zullen we steeds spreken over componenten, omdat `Component` in Java als class al bestaat, wordt in de implementatie gewerkt met `HardwareComponent`.

verbinden van het ene net met het andere, een component, die een gedurig signaal kan produceren, deze alsnog doorgeven aan het andere net. We leggen hier de verantwoordelijkheid dus bij de componenten. Worden door het sluiten van een schakelaar twee netten met elkaar verbonden, dan zullen we in het bijzonder elke component van het ene net doorverbinden met elke component van het andere net via de `Net`-methode `connectToComponents`. Voorzover een component nog een signaal heeft staan, wordt dit zo doorgegeven aan alle andere zojuist verbonden componenten.

Aangezien een schakelaar noch een detector noch een generator is, worden de methoden `connectTo` en `disconnectFrom` geherdefinieerd tot een no-op (distributie van signalen vindt alleen plaats bij openen of sluiten van schakelaars). Voor een switch is het ontvangen van signalen niet van belang. We maken van de signaal-detectie methoden, zoals `startSignal` daarom no-op's.[4]

6.3.4 Openen en sluiten van een schakelaar

Openen of sluiten van een schakelaar heeft niet altijd effect. Indien een tweetal netten bijvoorbeeld via twee schakelaars met elkaar verbonden zijn en beide zijn dicht dan heeft openen van één van de schakelaars geen effect. Eveneens heeft sluiten geen effect als er al één schakelaar gesloten was. We beschouwen het sluiten van een schakelaar:

```
public void close () {
....
}
```

Essentieel bij sluiten van een schakelaar is dat twee netten via `connectTo` met elkaar verbonden worden. Deze `Net`-methode berekent opnieuw wat voor elk van de netten de verzameling verbonden componenten is. Alleen indien hierin verandering optreedt, dient elke component werkelijk met elke andere verbonden te worden met als effect het eventueel verder verspreiden van signalen. Dit laatste effect komt later aan de orde. Hier bestuderen we alleen het bepalen van de verzameling verbonden componenten bij verbinden van twee netten. Noem die netten `A` en `B`. We bestuderen het effect van `A.connectTo (B)`. Eerst bepalen we van elk der netten de verzameling verbonden netten. We hebben hiervoor een `netSet` nodig, een verzameling `Net`-objecten:

```
NetSet sA = new NetSet () ;
NetSet aSet = aNet.connectedNets (sA) ;
```

`sA` vervult hier de rol van gemarkeerde netten in de verbindingsboom van `A`. We beginnen met een lege verzameling `sA` en vullen deze geleidelijk aan via de `Net`-methode:

```
public NetSet connectedNets (NetSet nSet) {
    if (nSet.contains (this)) {
       return nSet ;
    } else {
       NetSet aSet = new NetSet () ;
       aSet.addAll (nSet) ;
```

[4] Een schakelaar is een beetje een buitenbeentje binnen het geheel van componenten. In feite is een schakelaar dan ook niet echt een specialisatie van een component. Niettemin beschouwen we een schakelaar wel als zodanig, waarbij we een aantal methoden wegstrepen door er no-op's van te maken en een aantal andere methoden herdefinieren door het partner-object erbij te betrekken.

```
            aSet.addElement(this) ;
            for (int i=0 ; i < attachedComponents.size() ; i++) {
               aSet =
                  ((HardwareComponent)attachedComponents.elementAt(i)).
                      connectedNets (aSet) ;
            }
            return aSet ;
         }
      }
```

Essentieel hierin is dat elke vast verbonden component via `connectedNets` aangeeft met welke netten deze verbonden is. Voor bijna alle componenten is deze methode erg simpel: een gewone component is slechts met één net verbonden. In de class `HardwareComponent` ziet deze methode er dan uit alsvolgt:

```
public NetSet connectedNets (NetSet aSet) {
   return aSet ;
}
```

Alleen voor schakelaars moeten we iets bijzonders doen: is de schakelaar gesloten, dan moet ook gekeken worden met welke netten de andere kant van de schakelaar verbonden is. In de class `Switch` is de methode herschreven alsvolgt:

```
public NetSet connectedNets (NetSet aSet) {
   if (connected) {
      return partner.theNet.connectedNets (aSet) ;
   } else {
      return aSet ;
   }
}
```

Uiteindelijk vinden we langs deze weg in `aSet` de verzameling met net `A` verbonden netten en evenzo in `bSet` de verzameling met net `B` verbonden netten. Alleen indien deze verzamelingen disjunct zijn, heeft het sluiten van een schakelaar werkelijk zin. `aSet` en `bSet` zijn steeds ofwel volledig disjunct, ofwel volledig aan elkaar gelijk. Het is dan ook voldoende te controleren of een element uit de ene set in de andere voorkomt. De test `aSet.contains(B)` (of `bSet.contains(A)`) is voldoende om deze controle uit te voeren. Alleen indien de verzamelingen disjunct zijn, gaan we door en heeft sluiten van een schakelaar werkelijk effect.

Via de methode `attachedNormalComponents` kunnen we dan uit de verzamelingen `aSet` en `bSet` de verzameling vast verbonden componenten halen (waarbij we de schakelaars niet meenemen, vandaar “normal”). Deze verzameling kunnen we nu toekennen aan het attribuut `connectedComponents` in het overeenkomstige `Net`-attribuut en gebruiken om de losse componenten wederzijds met elkaar te verbinden.

Openen van een schakelaar gaat volkomen analoog.

6.3.5 Generator

Een generator is een speciale component, bedoelt om tonen of belsignalen te generen. Als extra attribuut is nodig te weten of de generator aan is. Voor een toongenerator is het soort toon dat wordt gegenereerd van belang. Als extra heeft elke generator de verantwoordelijkheid om bij

sluiten van een schakelaar eventuele toon- of belsignalen alsnog door te geven aan de dynamisch verbonden componenten.

Methoden Een generator moet kunnen worden aan- en uitgezet, dus: `start` en `stop`. Een toongenerator heeft daarnaast een methode nodig om het soort toon te kunnen zetten: `tone` Voor het maken van verbindingen via een schakelaar en het daarvoor noodzakelijke verspreiden van signalen zijn daarnaast `connectTo` en `disconnectFrom` geherdefinieerd, waarin eventuele signalen alsnog worden doorgegeven bij dichtzetten van een schakelaar, c.q., het signaal wordt onderbroken bij het openen van een schakelaar.
Een voor de hand liggende methode van een toongenerator is nu:

```
public void connectTo (HardwareComponent aComp) {
   super.connectTo (aComp) ;
   if (isOn) aComp.startSignal (tone) ;
}
```

6.3.6 Detector

Een detector detecteert een bepaalt signaal. Bij detectie moeten zekere acties worden uitgevoerd. Deze acties worden in een interrupt-methode ondergebracht. In Java[5] kunnen geen methoden worden toegekend aan variabelen. Er bestaat geen methoden-class.[6] Toch willen we graag later kunnen besluiten welke methode we aan als interruptmethode gebruiken. Daarvoor definieren we in Java het volgende interface:[7]

```
interface DetectorInterrupt  {

   public void runInterrupt (String value) ;
}
```

en voegen in de class `Detector` een attribuut toe van de vorm:

```
DetectorInterrupt interrupt ;
```

Als default-interrupt definieren we:

```
class DefaultInterrupt implements DetectorInterrupt {
   String id = "--" ;

   public DefaultInterrupt (String anId) {
      id = anId ;
   }
   public void runInterrupt (String value) {
      MyConsole.println(id +
        " is running interrupt with value: " + value) ;
   }
}
```

[5] D.w.z. in Java 1.0. In Java 1.1 bestaan locale classes en zijn meer mogelijkheden om het hier geschetse probleem op te lossen.

[6] In Smalltalk bestaat hiervoor de class `block`, als beschrijving voor naamloze methoden met nul of meer parameters.

[7] Zie het Design Patterns boek. Vergelijk deze aanpak met bijvoorbeeld de Template Method.

Zodat we het interrupt-attribuut van een Detector-object alsvolgt kunnen initieren:

```
interrupt = new DefaultInterrupt (this.getId());
```

Door een subclass van `DefaultInterrupt` te definieren met eigen ingevulde `runInterrupt`-methode hebben we onze flexibiliteit weer terug. Een detector voert zijn interrupt-methode nu uit middels aanroep van

```
interrupt.runInterrupt (value) ;
```

Een line interface wordt ook als detector gezien. Er vindt niet alleen detectie plaats van signalen op het net, maar vooral ook van de toestand van het aangesloten toestel: de toestand van de hoorn en de toestand tijdens het draaien/drukken van een cijfer. Deze detectie vindt van buiten af plaats. Voor simulatie zijn daarom methoden nodig, die deze detectie aangeven (zie onder). Als attributen van een line interface zijn nog nodig de toestand van de haak (`isOffhook`) en met welk toestel de interface verbonden is (`thePhone`). Tijdens testen kunnen we het `thePhone`-attribuut nog na eigen believen invullen en kunnen we `isOffhook` gebruiken voor het bijhouden van de toestand van de haak. Bij verbinding van hardware en software kan ook via het `thePhone`-attribuut de toestand van de haak worden opgevraagd.

Methoden Detectie van signalen verloopt via de methoden `startSignal` en `endSignal` (die in twee versies beschikbaar zijn, met en zonder een extra parameter, die de waarde aangeeft). Een detectie methode wordt opgestart door een generator, indien deze met een detector wordt verbonden.
Voor een willekeurige component ziet de methode `startSignal` er nu als volgt uit:

```
public void startSignal (String signal, int i) {
   message("receives start of " + signal + " (" + i + ")") ;
}
public void startSignal (String signal) {
   message("receives start of " + signal) ;
}
```

In de class `DtmfDetector` wordt de eerste geherdefinieerd als:

```
public void startSignal (String signal, int value) {
   super.startSignal (signal, value) ;
   if (signal.equals(HardwareComponent.Dtmf)) {
      interrupt.runInterrupt (Integer.toString (value)) ;
   }
}
```

De detector detecteert een dtmf-signaal, dan wordt de interrupt van de detector uitgevoerd met als parameter de waarde!. Deze parameter stelt het cijfer voor, dat door het dtmf-signaal wordt voorgesteld.

De subclass `LineInterface` detecteert de toestand van de hoorn van het toestel en het begin en eind van een dtmf digit. Om dit te simuleren hebben we methoden nodig die deze detectie aangeven: `startDtmfDigit` en `endDtmfDigit` (om begin en eind van een dtmf digit te detecteren) en `onHook` en `offHook` (voor de toestand van de haak). Bedoeling van detectie is, deze detectie verder over het net te sturen. Voor begin en eind van een dtmf digit moeten we deze signalen verder over het net zenden, bijvoorbeeld:

```
   public void startDtmfDigit (int aDigit) {
      Vector comps = theNet.getConnectedComponents() ;
      for (int i =0 ; i < comps.size() ; i++) {
         ((HardwareComponent)comps.elementAt(i)).
              startSignal (HardwareComponent.Dtmf, aDigit) ;
      }
   }
   public void endDtmfDigit (int aDigit) {
      Vector comps = theNet.getConnectedComponents() ;
      for (int i =0 ; i < comps.size() ; i++) {
         ((HardwareComponent)comps.elementAt(i)).
              endSignal (HardwareComponent.Dtmf, aDigit) ;
      }
   }
```

Na detectie van een verandering aan de toestand van de hoorn dient onze software hier op te reageren. Dit is mogelijk door de interrupt dusdanig te definiëren, bijvoorbeeld:

```
class LineInterfaceInterrupt implements DetectorInterrupt  {
   String id ;
   int number ;
   Phone phone ;
   public LineInterfaceInterrupt (LineInterface li, Phone ph, int i) {
      id = li.getId() ;
      phone = ph ;
      number = i ;
   }
   public void runInterrupt (String x) {
      if (x.equals ("offHook")) {
         phone.hookUp();
      }
   }
}
```

Waarbij de initialisatie er alsvolgt uit zou kunnen zien:

```
      LineInterface li1 = new LineInterface ("LI1") ;
      LineInterfaceInterrupt pi1 =
         new LineInterfaceInterrupt (li1,phone1,1) ;
```

6.3.7 Distributie van signalen

Verspreiden van signalen over het net gaat nu alsvolgt: een genererende component bevat start en stop methoden om generatie van signalen te kunnen starten en stoppen. Bijvoorbeeld in de class `RingGenerator`:

```
   public void start () {
      if (!isOn) {
         isOn = true ;
         theNet.startSignal(HardwareComponent.Ring,this) ;
```

```
    }
  }
```

Belangrijkste onderdeel van deze methoden is het sturen van de message `startSignal` dan wel `endSignal` naar het net, waartoe de component zich bevindt. In de class `Net`:

```
public void startSignal (String signal, HardwareComponent c) {
    for (int i=0 ; i < connectedComponents.size() ; i ++) {
       HardwareComponent x =
           (HardwareComponent) connectedComponents.elementAt (i) ;
       if ( x != c) {
          x.startSignal (signal) ;
       }
    }
}
```

Deze net-methoden sturen eenzelfde message naar elk van de componenten. In detecterende componenten bevat deze methode, voorzover van toepassing, een afhandeling, bijvoorbeeld in `LineInterface`:

```
public void startSignal (String signal) {
   message("receives start of " + signal) ;
   status = "start ‘‘ + signal ;
}
```

De laatste regel heeft betrekking op het toestel dat aan de line interface gekoppeld is: dit toestel begint te rinkelen.

De line interface detecteert, zoals eerder opgemerkt, ook handelingen aan het toestel, bijvoorbeeld begin en eind van een dtmf-signaal:

```
public void startDtmfDigit (int aDigit) {
   Vector comps = theNet.getConnectedComponents() ;
   for (int i =0 ; i < comps.size() ; i++) {
      ((HardwareComponent)comps.elementAt(i)).
           startSignal (HardwareComponent.Dtmf, aDigit) ;
   }
}
```

waarop, opdezelfde manier als boven beschreven, de Dtmf-generator een reactie geeft via:

```
public void startSignal (String signal, int value) {
   super.startSignal (signal, value) ;
   if (signal.equals(HardwareComponent.Dtmf)) {
      interrupt.runInterrupt (Integer.toString (value)) ;
   }
}
```

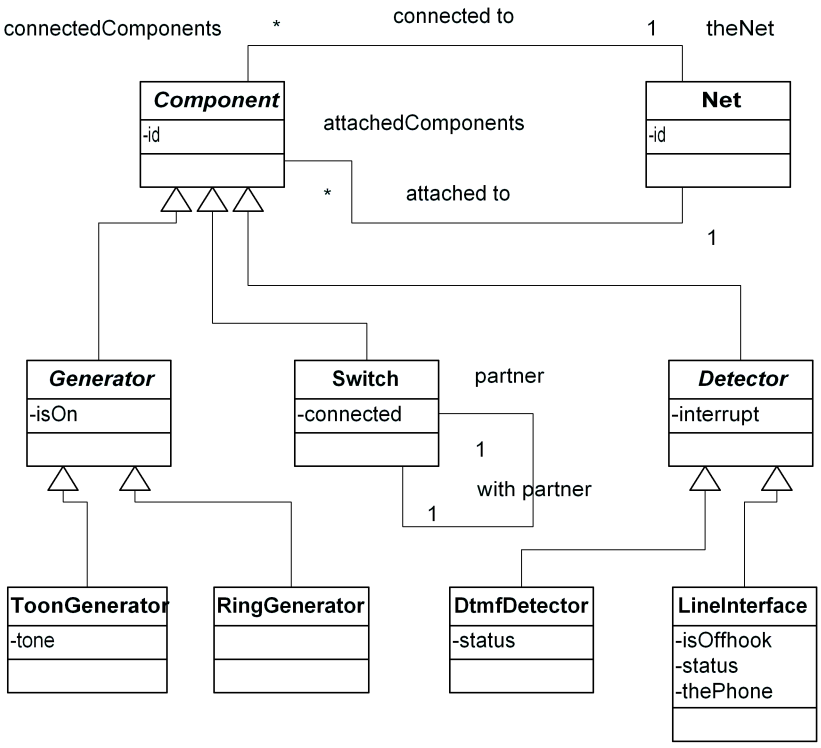
6.3.8 Conclusies

Een nadeel van het connectiemodel is dat we geen uitspraken kunnen doen over de signalen die zich op de draden (nets) bevinden. Een uitbreiding in die richting is erg eenvoudig: we

verbinden daartoe gewoon een ‘scope’ als component aan zo’n draad. Via het beschreven mechanisme ontvangt deze component alle events die via de draad verspreid worden. Eigenlijk is het `status`-attribuut in `LineInterface` als zo’n scope.

We zouden hiervoor ook een tussenoplossing kunnen vinden, als een component bij het genereren van een event, deze gewoon doorgeeft aan het netwerk. We komen dan weer in de richting van het signaal model.

We kunnen nu de analyse opmaken zoals in figuur 6.3 (waarbij we de connecties e.d. ook al via attributen hebben meegenomen, hetgeen eigenlijk pas in de ontwerp-fase gebeurt).



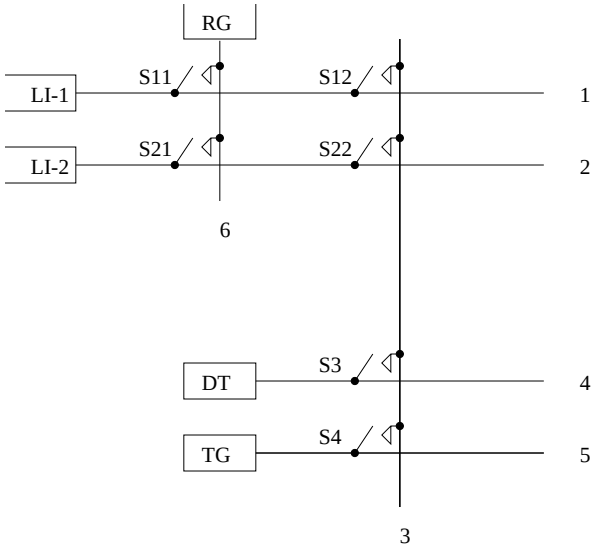
Figuur 6.3: Eerste analyse van hardware simulatie

7

HomeVox1a: gedetailleerde versie van HomeVox1

In dit hoofdstuk beschrijven we hoe we een gedetailleerde versie van HomeVox1 kunnen krijgen, waarmee rechtstreeks de HomeVox hardware (-simulator) aangestuurd kan worden.

We beschouwen daarbij het volgende, sterk vereenvoudigd, schema:



We zullen eerst het net moeten opbouwen. De elementen:

```
SimpleHomeVox homeVox = new SimpleHomeVox () ;
SimplePhone ph1 = new SimplePhone (homeVox) ;
SimplePhone ph2 = new SimplePhone (homeVox) ;
Net net1 = new Net ("1") ;
Net net2 = new Net ("2") ;
Net net3 = new Net ("3") ;
Net net4 = new Net ("4") ;
Net net5 = new Net ("5") ;
Net net6 = new Net ("6") ;
RingGenerator rg = new RingGenerator ("RG") ;
ToneGenerator tg = new ToneGenerator ("TG") ;
LineInterface li1 = new LineInterface ("LI1") ;
LineInterface li2 = new LineInterface ("LI2") ;
DtmfDetector dt = new DtmfDetector ("DT") ;
Switch s11 = new Switch (net1,net3, "S11") ;
```

```
      Switch s12 = new Switch (net1,net6, "S12") ;
      Switch s21 = new Switch (net2,net3, "S21") ;
      Switch s22 = new Switch (net2,net6, "S22") ;
      Switch s3 = new Switch (net6,net4, "S3") ;
      Switch s4 = new Switch (net6,net5, "S4") ;
```

Het aanelkaar solderen van de elementen:

```
      net3.attach(rg) ;
      net5.attach(tg) ;
      net1.attach (li1) ;
      net2.attach (li2) ;
      net4.attach (dt) ;
      LineInterfaceInterrupt pi1 =
         new LineInterfaceInterrupt (li1,ph1,1) ;
      LineInterfaceInterrupt pi2 =
         new LineInterfaceInterrupt (li2,ph2,2) ;
      DtmfInterrupt di = new DtmfInterrupt (dt, homeVox)  ;
      li1.setInterrupt (pi1) ;
      li2.setInterrupt (pi2) ;
      dt.setInterrupt (di) ;
```

Voor het zetten van de interrrupts hebben we bovendien gedeclareerd:

```
class LineInterfaceInterrupt implements DetectorInterrupt  {
   String id ;
   int number ;
   SimplePhone phone ;
   public LineInterfaceInterrupt (LineInterface li,
        SimplePhone ph, int i) {
      id = li.getId() ;
      phone = ph ;
      number = i ;
   }
   public void runInterrupt (String x) {
      MyConsole.println("interrupt running for "+id ) ;
      if (x.equals ("offHook")) {
         phone.getToon (number);
      }
   }
}

class DtmfInterrupt implements DetectorInterrupt {
   String id ;
   SimpleHomeVox homeVox ;
   public DtmfInterrupt (Detector dt, SimpleHomeVox hv) {
      id = dt.getId() ;
      homeVox = hv ;
   }
   public void runInterrupt (String x) {
      MyConsole.println("interrupt running for " + id
         + " with value: --" + x + "--") ;
      if (x.equals ("1")) {
         homeVox.connect (1);
      } else if (x.equals ("2")) {
         homeVox.connect (2) ;
```

```
      }
   }
}
```

De LineInterface-interrupt wordt gebruikt als van het toestel de hoorn van de haak gaat. De methode `getToon` komt verderop ter sprake. De bedoeling is dat deze methode uiteindelijk automatisch zorgt voor een toon in de hoorn. De DtmfDetector-interrupt wordt gebruikt om de verbinding tot stand te brengen met het toestel dat hoort bij het ingetoetste cijfer. De overeenkomstige methode `connect` komt eveneens verderop aan de orde.

Na het uitvoeren van:

```
net1.show () ;
net2.show () ;
net3.show () ;
net4.show () ;
net5.show () ;
net6.show () ;
```

verschijnt er nu op de uitvoer:

```
Net: 1:
attached:
  S11-1(1)  S12-1(1)  LI1(1)
connected:
  S11-1(1)  S12-1(1)  LI1(1)
Net: 2:
attached:
  S21-2(2)  S22-2(2)  LI2(2)
connected:
  S21-2(2)  S22-2(2)  LI2(2)
Net: 3:
attached:
  S11-3(3)  S21-3(3)  RG(3)
connected:
  S11-3(3)  S21-3(3)  RG(3)
Net: 4:
attached:
  S3-4(4)  DT(4)
connected:
  S3-4(4)  DT(4)
Net: 5:
attached:
  S4-5(5)  TG(5)
connected:
  S4-5(5)  TG(5)
Net: 6:
attached:
  S12-6(6)  S22-6(6)  S3-6(6)  S4-6(6)
connected:
  S12-6(6)  S22-6(6)  S3-6(6)  S4-6(6)
```

Door het activeren van `offHook` van de line interface die hoort bij toestel 1 wordt de LineInterface-interrupt aangesproken en via deze de methode `getToon` uit `SimplePhone`, die er alsvolgt uitziet:

```
   public void getToon(int i) {
      if (i==1) {
         homeVox.s12.close() ;
      } else if (i==2) {
         homeVox.s22.close() ;
      }
      homeVox.s4.close() ;
      homeVox.s3.close() ;
      MyConsole.test("start dialtone from tg");
      homeVox.tg.tone(HardwareComponent.Dial) ; homeVox.tg.start() ;
   }
```

Een detectie van het van de haak gaan van de hoorn resulteert dan in de volgende uitvoer:

```
..(offHook detected)..
interrupt running for LI1
..(S12-1 closing S12-1--S12-6)..
..(S4-6 closing S4-6--S4-5)..
..(TG connected to LI1)..
..(LI1 connected to TG)..
..(S3-6 closing S3-6--S3-4)..
..(DT connected to LI1)..
..(DT connected to TG)..
..(LI1 connected to DT)..
..(TG connected to DT)..
..(start dialtone from tg)..
..(LI1 receives start of dialTone)..
..(DT receives start of dialTone)..
```

Met behulp van

```
li1.startDtmfDigit(2) ; li1.endDtmfDigit(2) ;
```

simuleren we, dat de line interface detecteert dat het cijfer 2 wordt gedrukt. Doordat van de Dtmf detector een interrupt gezet is, zoals boven, resulteert detectie van dit Dtmf-signaal in het proberen te verbinden van toestel 2 met toestel 1. Dit laatste gaat via de methode `connect` uit `SimpleHomeVox`:

```
   public void connect (int i) {
      tg.stop() ; s4.open() ; s3.open() ;
      if (i==1) s11.close() ;
      else if (i==2)  s21.close() ;
      rg.start() ; rg.stop() ;
   }
```

Het gevolg van dit alles is, dat de bel overgaat bij toestel 2:

```
..(LI1 receives start of dtmfTone (2))..
..(TG receives start of dtmfTone (2))..
..(DT receives start of dtmfTone (2))..
interrupt running for DT with value: --2--
..(LI1 receives end of dialTone)..
```

```
..(DT receives end of dialTone)..
..(S4-6 opening S4-6--S4-5)..
..(TG disconnected from LI1)..
..(TG disconnected from DT)..
..(LI1 disconnected from TG)..
..(DT disconnected from TG)..
..(S3-6 opening S3-6--S3-4)..
..(DT disconnected from LI1)..
..(LI1 disconnected from DT)..
..(S21-2 closing S21-2--S21-3)..
..(RG connected to LI2)..
..(LI2 connected to RG)..
..(LI2 receives start of ring)..
..(LI2 receives end of ring)..
..(LI1 receives end of dtmfTone (2))..
```

Tot slot kan deze opnemen, waardoor de juiste verbinding voor spreken moet ontstaan:[1]

```
s21.open() ; s12.close() ; s22.close() ;
```

En ziehier:

```
..(S21-2 opening S21-2--S21-3)..
..(RG disconnected from LI2)..
..(LI2 disconnected from RG)..
..(S22-2 closing S22-2--S22-6)..
..(LI1 connected to LI2)..
..(LI2 connected to LI1)..
```

Het hierboven gegeven scenario is slechts een test-scenario. We zullen uiteindelijk de software uit de eerste hoofdstukken (bijvoorbeeld die uit hoofdstuk 3) moeten uitbreiden, zodat ook de hardware erbij betrokken wordt. In feite zijn slechts een beperkt aantal wijzigingen noodzakelijk. We noemen:

- een extra attribuut `theHardware` in `HomeVox`
- aanpassing van de `HomeVox`-constructor met een `initializeWithHardware`-methode.
- Koppelen van `LineInterface` en `Phone` middels een extra `theLineInterface`-attribuut in `Phone`.
- Aanpassing van `dialDigit` in `Phone`
- Aanpassing vn `ring`, `startDial` en `tone` in `Phone`

Het wordt aan de lezer overgelaten, de uiteindelijke implementatie te realiseren.

Implementatie in Java

Hieronder de sources van bovenstaande test.[2]

[1] De attente lezer moet het zijn opgevallen dat schakelaar 12 inmiddels al dicht is.

[2] De test is deels via de HIC te realiseren. Gezien het feit dat de volledige administratie van onze abstracte realisatie van interne gesprekken niet is meegenomen in deze test, zal het “beantwoorden van een gesprek” door het

7.0.9 SimpleHomeVox.java

```
 1  import java.util.* ;
 2  import java.lang.* ;
 3  import corejava.Console; // special from Core Java book
 4
 5  /** A special DetectorInterrupt for the LineInterface
 6  */
 7  class LineInterfaceInterrupt implements DetectorInterrupt  {
 8     String id ;
 9     int number ;
10     SimplePhone phone ;
11     public LineInterfaceInterrupt (LineInterface li,
12           SimplePhone ph, int i) {
13        id = li.getId() ;
14        phone = ph ;
15        number = i ;
16     }
17     public void runInterrupt (String x) {
18        MyConsole.println("interrupt running for "+id ) ;
19        if (x.equals ("offHook")) {
20           phone.getToon (number);
21        }
22     }
23  }
24  /** A special DetectorInterrupt for a DtmfDetector
25  */
26  class DtmfInterrupt implements DetectorInterrupt {
27     String id ;
28     SimpleHomeVox homeVox ;
29     public DtmfInterrupt (Detector dt, SimpleHomeVox hv) {
30     id = dt.getId() ;
31     homeVox = hv ;
32     }
33      public void runInterrupt (String x) {
34        MyConsole.println("interrupt running for " + id
35           + " with value: --" + x + "--") ;
36        if (x.equals ("1")) {
37           homeVox.connect (1);
38        } else if (x.equals ("2")) {
39           homeVox.connect (2) ;
40        }
41     }
42  }
43  /** A simple homevox, according to the hardware as in chapter 7.
44  */
45  class SimpleHomeVox {
46
47        Net net1 = new Net ("1") ;
48        Net net2 = new Net ("2") ;
```

tweede toestel nog via het handmatig zetten van schakelaars dienen te gebeuren. De bijbehorende HIC is zodanig ingericht, dat niet alleen bijvoorbeeld de stand van de schakelaars kan worden gezet, maar ook worden bekeken in welke stand de schakelaar zich bevindt. De lezer wordt van harte uitgenodigd de hardware-implementatie uit te proberen en zonodig aan te passen.

```
 49        Net net3 = new Net ("3") ;
 50        Net net4 = new Net ("4") ;
 51        Net net5 = new Net ("5") ;
 52        Net net6 = new Net ("6") ;
 53        RingGenerator rg = new RingGenerator ("RG") ;
 54        GeneratorView rgv = new GeneratorView (rg) ;
 55        ToneGenerator tg = new ToneGenerator ("TG") ;
 56        ToneGeneratorView tgv = new ToneGeneratorView (tg) ;
 57        LineInterface li1 = new LineInterface ("LI1") ;
 58        LineInterfaceView li1v = new LineInterfaceView  (li1) ;
 59        LineInterface li2 = new LineInterface ("LI2") ;
 60        LineInterfaceView li2v = new LineInterfaceView  (li2) ;
 61        DtmfDetector dt = new DtmfDetector ("DT") ;
 62        DetectorView dtv = new DetectorView (dt) ;
 63        Switch s11 = new Switch (net1,net3, "S11") ;
 64        SwitchView s11v = new SwitchView (s11) ;
 65        Switch s12 = new Switch (net1,net6, "S12") ;
 66        SwitchView s12v = new SwitchView (s12) ;
 67        Switch s21 = new Switch (net2,net3, "S21") ;
 68        SwitchView s21v = new SwitchView (s21) ;
 69        Switch s22 = new Switch (net2,net6, "S22") ;
 70        SwitchView s22v = new SwitchView (s22) ;
 71        Switch s3 = new Switch (net6,net4, "S3") ;
 72        SwitchView s3v = new SwitchView (s3) ;
 73        Switch s4 = new Switch (net6,net5, "S4") ;
 74        SwitchView s4v = new SwitchView (s4) ;
 75
 76     /** show the complete hardware configuration
 77     */
 78     public void showAll () {
 79        net1.show () ;
 80        net2.show () ;
 81        net3.show () ;
 82        net4.show () ;
 83        net5.show () ;
 84        net6.show () ;
 85     }
 86     /** simulate connection towards phone i
 87     */
 88     public void connect (int i) {
 89        tg.stop() ; s4.open() ; s3.open() ;
 90        if (i==1) s11.close() ;
 91        else if (i==2)  s21.close() ;
 92        rg.start() ; rg.stop() ;
 93     }
 94     /** test method,
 95     * The only thing that works is one hookup followed by one digit dialing
 96     * by the same phone.
 97     * Note that this is test configuration
 98     * The outline in this class should be reproduced in the real HomeVox
 and
 99     * integrated in what we already have made there.
100     */
101     public static void main (String[] argv) {
```

```
102        MyConsole.testing() ;
103        SimpleHomeVox homeVox = new SimpleHomeVox () ;
104        SimplePhone ph1 = new SimplePhone (homeVox) ;
105        SimplePhone ph2 = new SimplePhone (homeVox) ;
106        homeVox.net3.attach(homeVox.rg) ;
107        homeVox.net5.attach(homeVox.tg) ;
108        homeVox.net1.attach (homeVox.li1) ;
109        homeVox.net2.attach (homeVox.li2) ;
110        homeVox.net4.attach (homeVox.dt) ;
111        LineInterfaceInterrupt pi1 =
112           new LineInterfaceInterrupt (homeVox.li1,ph1,1) ;
113        LineInterfaceInterrupt pi2 =
114           new LineInterfaceInterrupt (homeVox.li2,ph2,2) ;
115        DtmfInterrupt di = new DtmfInterrupt (homeVox.dt, homeVox)  ;
116        homeVox.li1.setInterrupt (pi1) ;
117        homeVox.li2.setInterrupt (pi2) ;
118        homeVox.dt.setInterrupt (di) ;
119        homeVox.showAll() ;
120        MyConsole.noCounting() ;
121     }
122  }
```

7.0.10 SimplePhone.java

```
 1  import java.util.* ;
 2  import java.lang.* ;
 3  import corejava.Console; // special from Core Java book
 4
 5  /** This SimplePhone class is a subsitude for the real Phone class and
 6  * must only be used for test purposes during testing of the hardware.
 7  * It should be replaced by the original Phone-class from the HomeVox
 8  * package.
 9  *
10  * It comes together with SimpleHomeVox
11  * See chapter 7 of the HomeVox lecture notes for more details.
12  *
13  */
14  class SimplePhone {
15
16     SimpleHomeVox homeVox ;
17     /** constructor
18     */
19     public SimplePhone (SimpleHomeVox hv) {
20       homeVox = hv ;
21     }
22     /** phone i needs to get a tone
23     */
24     public void getToon(int i) {
25        if (i==1) {
26           homeVox.s12.close() ;
27        } else if (i==2) {
28           homeVox.s22.close() ;
29        }
30        homeVox.s4.close() ;
31        homeVox.s3.close() ;
```

```
32         MyConsole.test("start dialtone from tg");
33         homeVox.tg.tone(HardwareComponent.Dial) ; homeVox.tg.start() ;
34      }
35      /** phone i goes off hook
36      */
37      public void offHook (int i) {
38         if (i == 2) {
39            homeVox.s21.open() ;
40         } else if (i==1) {
41            homeVox.s11.open() ;
42         }
43         homeVox.s12.close() ;
44         homeVox.s22.close() ;
45      }
46   }
```