Vrije Universiteit Brussel
Faculteit Wetenschappen - Departement Informatica
Academiejaar 2001 - 2002

Quality of Service voor Mobile Multi Agent Systemen

Cedric Vanrykel

Proefschrift ingediend met het oog op het behalen van de graad van Licentiaat in de Informatica

Promotor: Prof. Dr. Theo D'Hondt
Begeleider: Johan Fabry

Samenvatting

Quality of Service is een netwerk technologie waar men probeert bepaalde garanties te leveren over de kwaliteit van de diensten die een netwerk levert. Dit hebben we gecombineerd aan Borg, wat een Mobile Multi Agent Systeem is waarvan de bedoeling is om gedistribueerd programmeren te vereenvoudigen. In deze verhandeling leggen we deze verschillende componenten uit en proberen we aan de hand van een aantal experimenten na te gaan of deze combinatie wel of niet een nut heeft.

Dankbetuiging

Eerst en vooral zou ik professor Dr. Theo D'Hondt en de leden van zijn laboratorium willen bedanken om me de kans geven te hebben om mijn thesis op PROG te kunnen doen.
Dan zou ik mijn begeleider, Johan Fabry, willen bedanken, voor al de tijd die hij genomen heeft om mij te helpen en voor al de steun die hij me verleend heeft.
Ik dank ook nog Werner Van Belle, voor al de hulp dat hij me verleend heeft bij het begrijpen en het uitbreiden van de Borg router.
Verder dank ik nog Kris Gybels voor de tijd dat hij genomen heeft om dit document na te lezen, en voor de hulp dat ik van hem kreeg in het begin van mijn stage.
Ten slotte dank ik nog mijn medestudenten Dimitri Verspecht en Pieter Verheyden, voor alle hulp en steun dat ze me verleend hebben.

Inhoudsopgave

1 Inleiding **6**
1.1 Overzicht . . . 7
1.1.1 Hoofdstuk 2: Quality of Service . . . 7
1.1.2 Hoofdstuk 3: Borg . . . 7
1.1.3 Hoofdstuk 4: Quality of Service in Borg . . . 8
1.1.4 Hoofdstuk 5: Experimenten . . . 8
1.1.5 Hoofdstuk 6: Conclusies . . . 9

2 Quality of Service **10**
2.1 Inleiding . . . 10
2.2 Wat is Quality of Service . . . 10
2.3 Waarom Quality of Service gebruiken . . . 11
2.4 Welke Quality of Service levels zijn er . . . 12
2.4.1 Best Effort . . . 12
2.4.2 Differentiated Service . . . 12
2.4.3 Guaranteed Service . . . 13
2.4.4 Praktijk . . . 14
2.5 Wat maak Quality of Service . . . 15
2.5.1 Beschikbaarheid . . . 15
2.5.2 Overdracht . . . 16
2.5.3 Vertraging . . . 16
2.5.4 Vertragings variatie . . . 17
2.5.5 Verlies . . . 17
2.6 Hoe Quality of Service bezorgen . . . 18
2.7 En verder 20
2.8 Conclusie . . . 21

3 Borg **22**
3.1 Inleiding . . . 22
3.2 Mobile Multi Agent Systemen . . . 23
3.2.1 Agents . . . 23

3.2.2 Mobile Agents . . . 23
3.2.3 Mobile Multi Agent Systemen . . . 24
3.3 Wat is Borg . . . 24
3.4 Programmeren in Borg . . . 25
3.4.1 Maken van Agents . . . 25
3.4.2 Objecten in Borg . . . 27
3.4.3 Communicatie tussen agents . . . 29
3.4.4 Een agent laten migreren . . . 33
3.5 Routing in Borg . . . 35
3.5.1 Naming . . . 35
3.5.2 Routing . . . 36
3.5.3 Serializing . . . 39
3.6 Andere manieren van werken dan Borg . . . 40
3.6.1 CORBA . . . 40
3.6.2 Java RMI . . . 42
3.7 Voor en nadelen van Borg . . . 42
3.8 Conclusie . . . 44

4 Quality of Service in Borg **45**
4.1 Inleiding . . . 45
4.2 De Borg Router . . . 45
4.3 De Quality of Service Router . . . 48
4.3.1 Priority Queue . . . 49
4.3.2 Congestion Control . . . 53
4.3.3 In de praktijk . . . 55
4.4 Aspecten van Quality of Service in Borg . . . 56
4.4.1 Beschikbaarheid . . . 56
4.4.2 Overdracht . . . 57
4.4.3 Vertraging . . . 57
4.4.4 Vertragings Variaties . . . 58
4.4.5 Verlies . . . 58
4.5 Conclusie . . . 59

5 Experimenten **60**
5.1 Inleiding . . . 60
5.2 Algemeen . . . 61
5.3 Wall Street . . . 62
5.3.1 Principe . . . 62
5.3.2 Werking . . . 63
5.3.3 Doel . . . 66
5.4 Monitoring . . . 66

5.4.1 Principe 66
5.4.2 Werking 67
5.4.3 Doel 68
5.5 Andere applicaties 69
5.5.1 De Meetingplanner 69
5.5.2 De Routeplanner 70
5.5.3 De Mail Client 72
5.6 Testresultaten 74
5.6.1 Setup 74
5.6.2 Verwachtingen 74
5.6.3 Observaties 75
5.6.4 Uitleg bij de observaties 77
5.7 Gebruik van Quality of Service in Borg 78
5.8 Conclusies 79

6 Conclusies **81**
6.1 Introductie 81
6.2 Overzicht 81
6.3 Resultaten 82
6.4 Conclusie 84
6.5 Verder werk 84

Lijst van figuren

3.1 De hiërarchie van de routers in Borg 38
3.2 Werken met stubs 41

4.1 De architectuur van de Borg router 46
4.2 De evolutie van de Borg router queue 51

5.1 De structuur van de Wall Street applicatie 65

Hoofdstuk 1

Inleiding

Het internet breidt zich dagelijks uit. Met die uitbreidingen ontstaan nieuwe vormen van gebruik die andere eisen stellen aan het netwerk dan wat we tot nu toe gewoon waren. Er wordt dan ook geprobeerd om aan die eisen te voldoen om de gebruiker van het netwerk tevreden te stellen. Dat is waar Quality of Service voor staat.

We komen in het domein van netwerken meer en meer Quality of Service tegen. Quality of Service staat niet voor een specifieke technologie die ervoor zal zorgen dat de eisen van de gebruiker voldaan worden, maar is een overkoepeling van verschillende technieken waarmee geprobeerd wordt om de netwerkervaring van de gebruiker te verbeteren door bepaalde garanties te leveren op de diensten die aan de gebruiker aangeboden worden.

Naast Quality of Service vinden we nog andere rijzende technologieën. Mobile Multi Agent Systemen is daar een van, die actief is in het veld van de gedistribueerde applicaties. Wat hiermee bereikt wil worden is het maken van een nieuwe manier om software componenten die op verschillende computers zitten met elkaar te laten communiceren. De methodes die tot nu toe bestonden werken op een vrij laag niveau. Bij Mobile Multi Agent Systemen gebeurt de communicatie op een hoger en abstracter niveau.

Een voorbeeld van zo een Mobile Multi Agent Systeem is Borg. Het is een agent systeem omdat de verschillende onderdelen van een programma in een agent terecht komen. Een agent is een abstractie die men kan zien als een object met zijn eigen thread, en waar de overerving werd weggelaten. Borg is een multi agents systeem omdat de verschillende agents in het systeem met elkaar kunnen communiceren door een eenvoudige mechanisme van boodschappen die tussen de verschillende agents gestuurd kunnen worden. Daarbij komt in Borg nog eens dat de verschillende agents niet verplicht zijn om op dezelfde computer te staan om met elkaar de kunnen communiceren: Ze kunnen gedistribueerd zijn over het netwerk. Dit betekent dat de agents die in het systeem aanwezig zijn niet altijd op dezelfde plaats moeten blijven. Borg ondersteunt migratie en de agents kunnen zich dus in het systeem van de ene computer naar de andere verplaatsen.

Voor deze thesis hebben we de technologieën van Quality of Service en Mobile Multi Agent Systemen gecombineerd. Het Borg systeem vormt een abstract netwerk bovenop het fysiek netwerk. Er werd een poging gedaan om dat abstract netwerk uit te breiden met technieken die toelaten om Quality of Service aan te bieden. Hiervan verwachten we dat het een pluspunt zal brengen voor het verzekeren van bepaalde garanties aan de gebruiker van een Borg applicatie. Maar het is niet zeker, en dat is wat we in deze verhandeling zullen proberen na te gaan.

1.1 Overzicht

We geven hier een korte overzicht van de structuur van dit document.

1.1.1 Hoofdstuk 2: Quality of Service

We beginnen dit document met een uitleg over Quality of service (QoS). Wat Quality of Service is en waarvoor het dient wordt ten eerste uitgelegd. Dan komt er een analyse van de verschillende niveaus van Quality of Service dat men terugvindt, namelijk Best Effort, Differentiated Service en Guaranteed Service. Verder komt er in dit hoofdstuk een uiteenzetting van de verschillende onderdelen van een netwerkverbinding die een sterke invloed hebben op de kwaliteit van de diensten. Ze worden ook de dimensies van Quality of Service genoemd. Het zijn beschikbaarheid, overdracht, vertraging, vertragings variaties en verlies. We eindigen dit hoofdstuk met een uiteenzetting van verschillende technieken die gebruikt worden om Quality of Service aan te bieden en het wordt gevolgd met een reflectie over hoe het verder kan verlopen met het gebruik van Quality of Service.

1.1.2 Hoofdstuk 3: Borg

Het derde hoofdstuk van deze thesis handelt over Borg. Dit komt omdat er verder gewerkt wordt met een versie van Borg die uitgebreid werd met Quality of Service eigenschappen en dat Borg voor de experimenten ook gebruikt werd. Ten eerste wordt er uitgelegd wat een Mobile Multi Agents systeem is. Dit is om te begrijpen wat er met Borg bereikt wil worden. Verder kom er dan de uitleg over waar Borg echt voor staat. Het wordt gevolgd met een inleiding naar Borg. De manier waarop in Borg geprogrammeerd wordt is uitgelegd. Het werken met agents wordt besproken, samen de technieken die aanwezig zijn om berichten tussen de agens te sturen. Er zijn ook voorbeelden van het gebruik van Borg en de verschillende mogelijkheden die het aan de programmeur biedt. Vervolgens komt een uitleg van de routing in Borg. Het hoofdstuk wordt beëindigd met een analyse van Borg. De voor en nadelen van Borg worden besproken en vergeleken met twee andere systemen waarvan de bedoeling ook is om het programmeren van gedistribueerde applicaties te vereenvoudigen, namelijk CORBA en Java RMI.

1.1.3 Hoofdstuk 4: Quality of Service in Borg

We hebben een versie van Borg gemaakt die werd uitgebreid met eigenschappen die toelaten om Quality of Service te bieden. Deze wordt in hoofdstuk vier besproken. Het eerste deel van dit hoofdstuk bespreekt de originele Borg router. Het is het deel van Borg dat ervoor zorgt dat boodschappen tussen de verschillende Borg agents gestuurd kunnen worden. Het is deze router dat aangepast werd om Quality of Service te bieden. In het tweede deel worden de uitbreidingen die gemaakt werden uitgelegd. De voornamelijkste aanpassingen zijn het toevoegen van prioriteiten aan de berichten en het toevoegen van congestie controle. Verder wordt er in dat hoofdstuk nog besproken hoe de verschillende dimensies die Quality of Service beïnvloeden zich naar Borg vertalen. Het wordt vervolledigd met een theoretische conclusie over welke de toegevoegde waarde kan zijn die de met Quality of Service uitgebreide router aan de verschillende dimensies aanbied.

1.1.4 Hoofdstuk 5: Experimenten

De theoretische conclusies uit het vorig hoofdstuk worden in hoofdstuk vijf aan de tand gevoeld met een reeks experimenten die de eigenschappen van Quality of Service moeten testen in een Borg netwerk. Verschillende programma's die de toegevoegde eigenschappen moeten uitbuiten worden uitgelegd. Verder vinden we in dit hoofdstuk de uitleg bij de experimenten die uitgevoerd werden. Ze worden gevolgd met de bespreking van de resultaten die men verwachtte te vinden. Daarna komen de echte resultaten die gevonden werden aan bod. Het hoofdstuk eindigt met een uitleg over hoe het komt dat die resultaten gevonden werden.

1.1.5 Hoofdstuk 6: Conclusies

In het laatste hoofdstuk van deze thesis beginnen we door een samenvatting te maken van wat er allemaal gezegd werd doorheen het document. Het wordt gevolg door een herhaling van de conclusies die men in deze thesis heeft kunnen maken.

Hoofdstuk 2

Quality of Service

2.1 Inleiding

Quality of service is een term dat men meer en meer tegenkomt als men over netwerken bezig is. Het is ook een van de hoofdaspecten van deze verhandeling.

In dit hoofdstuk wordt er uitgelegd wat Quality of Service betekent en wat het nut ervan eigenlijk is. Dan worden drie verschillende levels van Quality of service uit de doeken gedaan: Best effort, Differentiated service en Guaranteed service . Er volgt nog een uiteenzetting van de verschillende factoren die Quality of Service beïnvloeden. Tenslotte worden er verschillende technieken om Quality of Service aan te bieden aan het licht gebracht.

2.2 Wat is Quality of Service

De dag van vandaag kunnen we opmerken dat communicatienetwerken convergeren naar een unieke netwerk infrastructuur die zowel data overbrengt als spraak, video en andere gegevens afkomstig van gelijkaardige applicaties. De verschillende soorten gebruik gaan over dezelfde datalijnen en moeten met elkaar coëxisteren. De verhuurders van die lijnen moeten met dat samenleven overweg kunnen zonder dat de gebruikers er iets van moeten weten. De term Quality of Service (QoS) [1] refereert naar de kwaliteit van de diensten die aan de netwerk gebruikers aangeboden worden en overkoepelt al de verwante technologieën in deze sector. Het doel van QoS is om een dienst te leveren die voldoet aan de eisen die de gebruikersapplicaties aan het netwerk stellen. QoS is niet gekoppeld aan een specifieke technologie maar groepeert alle bestaande en toekomstige technologieën zoals IP, Frame Relay, ATM, MPLS, Ethernet, en ook de nieuwe draadloze en mobile netwerken.

QoS verleent de nodige brandbreedte, gecontroleerde latency (wat staat voor de tijdsverschil tussen zending en ontvangst [2]) en jitter (wat het verschil in latency is [3]), vereist door bepaalde netwerk trafieken en verbetert de dataverlies karakteristieken van het netwerk. Quality of Service bereikt zijn doelen door een groot assortiment aan tools en protocollen aan te bieden voor het managen en voorkomen van netwerk congestie, bandbreedte reservatie en trafiek vormgeving en beheer. Verbeterde queueing tools kunnen de inkomende trafiek opsplitsen in verschillende queues om prioriteit te geven aan interactieve applicaties (bv. voice pakketten) en de kwaliteit bewaren die door de eindgebruiker waargenomen wordt. Andere QoS mechanismen kunnen congestie voorkomen door het vertragen van trafiek die veroorzaakt wordt door applicaties zoals file transfer. Vormgeving en beheer van data trafiek kan helpen bij het respecteren van de Service Level Agreement (SLA), of de overeengekomen kwaliteit van de diensten, tussen de gebruikers en de Netwerk Service Provider[4]. Bandbreedte reservatie over het netwerk geeft een gegarandeerde [5] end-to-end service voor constante bit rate service als voice, wat hun strikte latency en jitter eigenschappen bewaart. Ook betekent QoS netwerk-engineering criteria om al deze blokken samen te hangen.

2.3 Waarom Quality of Service gebruiken

De nood aan QoS stijgt omdat nieuwe netwerk applicaties veel resources opeisen. We denken hierbij vooral aan de voice, video en interactieve multimedia services. De resources moeten met zorg behandeld worden om aan de eindgebruiker een goede kwaliteit te leveren. Bandbreedte is nodig maar niet voldoende, de netwerken moeten een vorm van intelligentie vertonen om deze applicaties te ondersteunen. QoS tools en protocollen zijn de instrumenten die toelaten om deze applicaties te ondersteunen en daarmee de kwaliteit te garanderen in de communicatie netwerken [16].

Dat is niet het enige waarvoor quality of Service goed is. Providers hebben ook voordelen van het gebruik van Quality of Service. Het is zo dat in de communicatie sector de meeste opbrengst afkomstig is van hoge kwaliteit voice applicatie en dergelijke [9]. Daarnaast komt het data transport dat meer een meer groeit. Het transport van data is wat er tegenwoordig het meest voorkomt op de netwerken. De vraag naar interactive multimedia services is enorm gestegen en de providers hebben moeten volgen door meer een meer bandbreedte aan te bieden. Maar de prijzen zijn bijna evenredig gedaald, vooral doordat er meestal slechts een Best Effort netwerk aangeboden wordt en dat de mensen niet klaar zijn om veel te betalen voor een lage kwaliteit. Het gebruik van Quality of Service zou toelaten om nieuwe Service Level

Arrangementen te maken. Er kunnen dan, dankzij de garanties die gekoppeld zijn aan het netwerk, verschillen gemaakt worden naar gelang de QoS dat de gebruiker vraagt en waar hij voor betaald heeft. De gebruiker zal dan voor een hogere prijs ook een beter netwerk ervaren.

2.4 Welke Quality of Service levels zijn er

Niet alle netwerken bieden dezelfde soort Quality of Service. Ze verschillen naar gelang de technieken die het netwerk gebruikt. Er zijn voornamelijk drie levels van Quality of Service dat een netwerk kan aanbieden: Best Effort, Differentiated Service en Guaranteed Service [5] [17].

2.4.1 Best Effort

Best Effort service [6] is de eerste level van Quality of Service. De naam op zichzelf duid al aan dat deze geen garanties van welke soort dan ook geeft. Als de netwerk resources aanwezig zijn zal het Best Effort systeem de datastroom tot zijn doel brengen, anders kan het gebeuren dat de stroom ergens onder weg verloren loop. Het kan daarbij geen garantie geven voor een minimale data rate of een levering binnen bepaalde tijden. Best Effort service kan zonder probleem gebruikt worden voor het versturen van files, e-mail of nog andere Web pagina's en dergelijke. Het verlies van datastromen kan bij deze soort applicaties verholpen worden door het opnieuw sturen van de pakketten die hun doel niet bereikt hebben. De dag van vandaag is internet gebaseerd op Internet Protocol (IP) een goed voorbeeld van een Best effort netwerk die niet echt een betrouwbare oplossing is voor interactieve applicaties. Iedereen kan merken hoe laag de performanties zijn voor video en voice applicaties over het Web. Toch wordt er gewerkt aan het ontwerp van gestandaardiseerde Quality of Service protocollen om Internet Protocol QoS gevoelig te maken. Wanneer deze af zijn zal het Internet de mogelijkheid hebben om de meeste QoS eigenschappen aan te kunnen [6].

2.4.2 Differentiated Service

Differentiated Service [7], ook gekend als “lichte QoS”, is een intermediaire Quality of Service level. Verschillende types trafiek gaan over dezelfde paden door het netwerk, hier kunnen bijvoorbeeld tijdskritieke applicaties concurreren met minder tijdsafhankelijke applicaties voor de netwerk resources. In het Differentiated Service systeem wordt een deel van de trafiek beter behandeld dan de rest. Pakketten die het netwerk binnenkomen krijgen een etiket mee, naar gelang de applicatie waarvan ze afkomstig zijn, of de Service Level Agreement met de gebruiker. Dan krijgt het meest belangrijk verkeer naar gelang deze classificatie voorrang. Laten we een voorbeeld beschouwen van een netwerk waar video en voice applicaties runnen naast data applicaties zoals File Transfer, Web Browsing en Print Spooling. Om te voorkomen dat data verkeer in de weg van voice en video zou komen wordt de inkomende trafiek geclassificeerd naargelang zijn oorsprong.

Door het configureren van geavanceerde queueing mechanismen op de acces devices, wat het apparatuur is waar de netwerk trafiek het netwerk binnentreed, kan een netwerk manager hoge prioriteit geven aan tijdsafhankelijke trafiek. Dit betekent lage latency, gecontroleerde jitter en lage frame verlies ratio voor real-time applicaties waar de vertragingen op de gegevens een negatieve invloed hebben, en dat is een goede Quality of Service. Verkeer dat minder tijdsafhankelijke gegevens overbrengt wordt behandeld met een lagere prioriteit. Als de device queues vol zijn of als het netwerk overvol is, dan zal dit soort trafiek meer kans hebben om gedropt te worden en er zal dan een nieuwe zending gebeuren van de verloren pakketten. Vermits de applicaties die deze stromen aanmaken minder tijdsafhankelijk zijn zullen ze minder gevoelig zijn voor de vertraging die door het pakketverlies veroorzaakt worden.

2.4.3 Guaranteed Service

Voorrang geven aan netwerk trafiek is nog niet echt een veilige oplossing om Quality of Service te verzekeren. Het is meestal op basis van statistische gegevens gebouwd, en dus niet een strikte garantie. Bij een grote congestie periode kan alle verkeer, zelfs die met hoge prioriteit, de gevolgen van die congestie ondergaan en de kwaliteit van de overdracht van data voor een applicatie dat een hoge kwaliteit verwacht kan daar ook onder verminderd geraken. Om een strikte garantie van Quality of Service te bereiken is Differentiated Service niet genoeg en moet er een level hoger gegaan worden: de Guaranteed level.

De Guaranteed Service level [5], ook gekend als “sterke QoS”, is de hoogste level van Quality of Service architectuur waar er een absolute end-to-end reservatie is van netwerk resources voor een specifiek type verkeer. Net zoals bij Differentiated Service gebeurt er een classificatie van de inkomende trafiek en wordt deze opgesplitst in verschillende banen met verschillende voorrangs niveaus die met de eisen van de netwerk applicaties overeen stemmen. Sommige van deze banen kunnen gegarandeerde resources krijgen zoals een gedefinieerd percentage van de aanwezige bandbreedte op het kanaal. Bandbreedte wordt gebruikt als middel om aan de andere eisen van QoS te voldoen, zoals lage delay, jitter en data verlies limieten. Zo kunnen applicaties zoals voice en video in een Guaranteed Service netwerk altijd een snelle weg vinden door het netwerk, zelfs wanneer de congestie hoog is.

Bandbreedte reservatie kan in het algemeen zowel dynamisch als statisch gebeuren. In statische modus is de bandbreedte reservatie altijd actief. Dit impliceert constante allocatie van netwerk resources, zelfs indien er geen data verzonden wordt. Met de dynamische modus gebeurt bandbreedte reservatie via signalering mechanismen wat betekent dat netwerk resources enkel gebruikt worden wanneer data doorgezonden wordt.

2.4.4 Praktijk

Wanneer men een degelijke Quality of Service wil aanbieden is het gebruik van Guaranteed Service beter dan Differentiated Service maar het is moeilijker om een netwerk te bouwen dat aan Quality of Service doet gebruik makende van Guaranteed Service. Statische reservatie van netwerk resources is heel inefficiënt, en dynamische reservatie gebaseerd op signalering mechanismen is heel complex om te implementeren in een heterogeen netwerk [8]. Daarom is Differentiated Service Level hedendaags het meest gebruikte paradigma om een netwerk te bouwen dat Quality of Service aanbiedt.

Een kritisch punt voor alle netwerk technologieën en zeker voor Quality of Service onderdelen is interoperabiliteit. Echte Quality of Service is er slechts als de gebruikers aan elke uiteinde van een heterogeen netwerk dezelfde QoS kunnen krijgen. Netwerk services kunnen verschillende netwerken doorkruisen die gebaseerd zijn op andere technologieën die QoS op verschillende manieren ondersteunen. We kunnen als voorbeeld internet Protocol nemen dat ontworpen werd op basis van een Best Effort protocol en slechts recent QoS eigenschappen heeft beginnen incorporeren. Dat kunnen we dan vergelijken met een ATM (Asynchronous Transfer Mode) platform dat ontworpen werd met een ingeburgerd Quality of Service level [18]. ATM is tegenwoordig een van de beste netwerk technologie om QoS te ondersteunen [19], maar door zijn complexiteit en zijn cost heeft het vaak moeilijkheden om op te komen tegen het Ethernet dat we meestal in Locale Area Networks (LAN) terugvinden. Het werk om te verzekeren dat IP netwerken met ATM netwerken verbonden kunnen zijn is gedaan, maar er bestaat geen unieke standaard om de ATM classes van trafiek over te brengen in de IP QoS mogelijkheden [20]. Ook is het nodig, om de voordelen van Quality of Service te gebruiken, dat er een integratie gebeurt van de verschillende technologieën met verschillende QoS levels in een uniek netwerk met echte end-to-end Quality of Service levels. Dit is zeer belangrijk vermits het zeer onwaarschijnlijk is dat er in de toekomst slechts één netwerk technologie zal overblijven.

2.5 Wat maak Quality of Service

Om aan de Quality of Service noden van een applicatie te voldoen zijn er verschillende zaken die een rol spelen. Er zijn vijf basis dimensies [1] die een impact hebben op de QoS die de eindgebruiker zal ervaren:

- Beschikbaarheid (availability)
- Overdracht (throughput)
- Vertraging (delay)
- Vertragings variaties
- Verlies (loss)

Verder bespreken we ieder van die dimensies apart, samen met technieken die gebruikt kunnen worden om met de dimensies te werken.

2.5.1 Beschikbaarheid

Beschikbaarheid is een percentage tijd dat het netwerk werkt wanneer de gebruiker het nodig heeft. Het traditionele streefdoel is 99.999% [10], wat vertaalt tot ongeveer 5.25 minuten per jaar waar het netwerk niet werkt. Dit is echter een maat die niet bereikbaar is met de meeste huidige datacom uitrustingen.

Beschikbaarheid wordt bereikt door een combinatie van de betrouwbaarheid van de uitrusting en het overlevingsvermogen van het netwerk. Andere factoren spelen ook een rol zoals de stabiliteit van de gebruikte software en de mogelijkheid om de netwerk uit te breiden of te upgraden zonder het buiten dienst te brengen.

Verder moet het netwerk ook in staat zijn om zichzelf te herstellen wanneer er ergens een faling gebeurt. Alternatieve paden door het netwerk moeten zo snel mogelijk beschikbaar zijn zodat het verkeer niet moet lijden onder het verlies van een link of een knooppunt. Het netwerkprovisioneringssysteem moet op zo een manier geïmplementeerd worden dat het falen van een deel niet te snel de andere delen gaan overbelasten. Wanneer er een fout of een probleem voorkomt moet de service zo snel mogelijk opnieuw hersteld worden zodat de onderbreking voor de gebruiker onzichtbaar wordt.

2.5.2 Overdracht

Overdracht, of throughput, is de basis meettechniek om de hoeveelheid trafiek of bandbreedte geleverd over een bepaalde periode te meten. In het algemeen geldt hoe meer overdracht hoe beter. De gebruikers zijn tegenwoordig gewoon om bandbreedte te krijgen die berekend wordt in kilobit per seconde (kbps), en een T1 lijn (aan 1.544 megabit per seconde) wordt beschouwd als een hoge snelheid verbinding voor een huishoudelijk gebruik met applicaties die hoofdzakelijk aan file transfer doen, maar ook nog voor gebruik binnenin een bedrijf [11]. Maar de nieuwe technologieën en services gaan zo snel vooruit dat men binnenkort over megabit of zelfs gigabit per seconde zal beginnen praten. Natuurlijk heeft niet iedereen daar nood aan en is niet iedereen bereid om de prijs ervan te betalen. De priority queueing en trafiek controle technieken laten dan ook toe om het snelheid van een gebruiker te vertragen zodat deze ten minste krijgen wat ze voor betalen.

2.5.3 Vertraging

De vertraging (of ook delay of latency in het Engels) is het gemiddelde transittijd van een stuk data tussen twee punten in het netwerk . Vele services, en vooral real-time services zoals voice en video communicaties, zijn zeer gevoelig en hoogst intolerant voor vertraging [12]. Om deze soort service op een aanvaarbare manier toe te laten moet de netwerk apparatuur in staat zijn om lage vertragingen te garanderen.

Er zijn veel componenten bij delay, waaronder pakketisatie delay, queueing delay, switching delay en propagatie delay. Pakketisatie delay komt voor uit de feit dat de datastromen opgesplitst moeten worden in aparte pakketten die elk op hun beurt door het netwerk moeten reizen.

Queueing delay wordt opgelegd aan pakketten die aankomen op knooppunten die zich in een congestie toestand bevinden. Daar worden ze opgehouden en moeten ze op hun beurt wachten om verder verzonden te worden. Dit kom voor wanneer een knooppunt meer verkeer binnen krijgt dan het aankan. Een nieuw pakket dat aankomt zal niet onmiddellijk behandeld kunnen worden. Het zal moeten wachten dat de pakketten die voor hem aangekomen zijn verder geraakt zijn. ATM netwerken verhelpen dit door de pakketten in zo klein mogelijke cellen te plaatsen. Met het gebruik van prioriteit queues zorg dit ervoor dat pakketten met een hoge prioriteit minder lang moeten wachten. Dit is gebeurt omdat het verder doorsturen van een pakket minder tijd in beslag neemt. Daardoor zal een queue met hogere prioriteit sneller weer aan de beurt komen. Dit zorg ervoor dat de wachttijd verminderd voor hoge prioriteit pakketten. Maar op gigabit snelheid is de wachttijd voor hoge prioriteit pakketten zeer klein, zelfs onder de slechtste condities, vermits de pakketten altijd heel snel voort propageren. Deze eigenschap van snelle netwerken laat toe om applicaties zoals voice toe te laten zonder daarvoor complexe queueing mechanismen te implementeren.

Switching en propagatie delay heeft te maken met de tijd die informatie nodig heeft om een koper, fiber of draadloze link te doorkruisen, en is in functie van de snelheid van licht. Het heeft dus een grote invloed op de vertraging van een datastroom, vooral over lange afstanden. Het is daarom ook een zeer belangrijke factor in een netwerk systeem.

2.5.4 Vertragings variatie

Vertragings variatie (delay varaition) is het verschil in delay die ondervonden wordt op verschillende pakketten die deel uitmaken van de zelfde trafiek flow. Hoge frequentie delay variaties wordt ook gekend onder de naam jitter [3], en lage frequentie variaties worden wander genoemd [13].

Jitter wordt vooral veroorzaakt door verschillen in queue wachttijden voor opeenvolgende pakketten in een flow en is hetgeen dat het meest invloed heeft op Quality of Service. Sommige types van verkeer, en zeker real-time trafiek als voice en video, zijn zeer intolerant voor jitter. Verschillen in de aankomst van pakketten kan de oorzaak zijn van hakken en kappen in de stream van gegevens. Alle transport systemen veroorzaken jitter. Zolang de jitter binnen bepaalde getolereerde normen valt stoort het niet echt de Quality of Service. Overdadige jitter kan voorkomen worden door te bufferen, maar deze vergroot de vertraging, wat op zijn beurt voor problemen kan zorgen.

Wander is een probleem dat opkomt bij synchrone systemen. Dit kan overkomen worden door een strikte Stratum timing over het netwerk te hebben [14]. In asynchrone netwerken in wander meestal niet een probleem.

2.5.5 Verlies

Verlies, zij het bit errors of pakket verlies, heeft een groter invloed op pakket data services dan op real-time services. Als er gedurende een gesprek een bit of een pakket verloren geraakt gebeurt dat meestal zonder dat de gebruikers het zelfs merken. Bij video broadcast kan verlies zorgen voor een momentele storing op het scherm maar de video zal daarna verder gaan als het ervoor bezig was.

Zelfs een Transmision Control Protocol (TCP) data transfer kan data verlies behandelen. Het is zelfs zo dat een veelgebruikte congestie controle mechanisme, genaamd Random Early Discard (RED) [15], eigenlijk juist gewillig pakketten begint te droppen wanneer het verkeer een bepaalde voorgedefinieerde status bereikt zodat het aantal pakketten in een knoop zou dalen en de congestie zou verminderen. Maar als pakket verlies te hoog begint te worden, dan vermindert de kwaliteit van de transmissies.

Tabel 2.1 toont een reeks applicaties en in welke mate ze beïnvloed worden door de dimensies van Quality of Service.

Tabel 2.1: De invloed van de QoS dimensies op verschillende soorten applicaties

Applicatie	Beschikbaarheid	Overdracht	Delay	Jitter	Verlies
Circuit services	Hoog	Hoog	Hoog	Hoog	Hoog
Interactieve video	Hoog	Hoog	Hoog	Hoog	Medium
Voice telefonie	Hoog	Laag	Hoog	Hoog	Laag
Broadcast video	Hoog	Hoog	Laag	Medium	Medium
File transfer	Medium	Laag	Laag	Laag	Hoog
Email	Laag	Laag	Laag	Laag	Hoog

2.6 Hoe Quality of Service bezorgen

Er zijn verschillende manieren om Quality of Service te bezorgen. Het kan gaan door brute kracht of op een intelligente manier. De beste manier hangt af van de mogelijkheden die men ter beschikking heeft wat mankracht en apparatuur betreft voor het opstellen van een QoS ondersteunend netwerk.

Een van de manieren om Quality of Service te bezorgen is door overlappende netwerken te gebruiken. Elke type netwerkverkeer krijgt zijn eigen infrastructuur. Dit is een vorm van gebruik van brute kracht. Het is duur wat het apparatuur, gebruik van bedrading en management betreft. Toch is het tegenwoordig een van de meest toegepaste methoden bij providers [14].

Anderen geloven erin dat het toevoegen van genoeg bandbreedte in de verbindingen tussen de routers en de switches voldoende is om de nood aan Quality of Service mechanismen te verwijderen en dat dit geschikt is voor de controle van delay en jitter voor tijdsafhankelijke pakketten. Het is wel zo dat meer bandbreedte leveren, zonder dat er een equivalente groei van het gebruik ervan is, als resultaat zal hebben dat er een vermindering zal gebeuren van de jitter. Maar het is op zichzelf nog niet voldoende om een goede dienst kwaliteit te bezorgen en het verwijdert niet de nood naar eenvoudige QoS mechanismen zoals bijvoorbeeld het voorrang geven aan verkeer die gevoelig is voor tijdsverschillen.

Dat bandbreedte toevoegen niet genoeg is kunnen we aantonen door een klein voorbeeld. Laten we een netwerk beschouwen waar gewone data applicatie dezelfde netwerk apparaten gebruiken als een video applicatie. Een data applicatie stuurt een groot pakket door op het netwerk. Die komt aan juist voor een pakket van de video stroom. Het datapakket, die toevallig langs dezelfde weg moet als het videopakket, zal altijd voor blijven op het videopakket en zo zal de laatste altijd opgehouden worden door het eerste wanneer ze in een queue terecht komen. Daarbij is het nog mogelijk dat andere applicaties onderweg ook grote pakketten doorsturen die in een queue terecht komen voor het videopakket. Deze zullen ook voor vertraging zorgen. Al deze vertragingen zouden verdwijnen door een eenvoudige prioriteits algoritme aan de queue toe te voegen. Bandbreedte in overvloed toevoegen versnelt misschien wel het verkeer tussen te knooppunten maar niet erin. Daarom blijft het gebruik van zelfs eenvoudige Quality of Service mechanismen altijd een voordeel.

Daarnaast komt ook nog eens een economische factor kijken. Het is zo dat applicaties zoals voice tegenwoordig slechts een kleine percentage van het verkeer vertegenwoordigt tegenover de meer klassieke IP toepassingen. De overvloed aan bandbreedte die voor die soort applicaties voorbehouden worden zou dan slechts weinig gebruikt worden. Dit wil zeggen dat het netwerk van een provider niet op volle capaciteit zou werken, terwijl men bij een provider meestal streeft naar een volledig gebruik van de apparaten om de rendabiliteit te verhogen. Voor die mensen betekent dus meer bruikbare bandbreedte meer inkomen, wat ertoe leidt dat de volle bandbreedte meestal wel voor alle soorten applicaties gebruikt worden, zelfs diegene die een lagere prioriteit hebben.

Naast overlappende netwerken en bandbreedte toevoegen zijn er nog andere oplossingen om Quality of Service te leveren, zoals bij ATM [18]. De manier van werken dat bij ATM gebruikt wordt is om queueing en signalering te doen voor elke netwerk stroom in elk knooppunt. Langs een kant is dit een zeer goede manier om de behandeling van het verkeer te garanderen. Langs een andere kant is dat een zeer complexe en dure oplossing, want er moet veel opgevolgd en gecontroleerd worden, en de queues nemen veel geheugen in. Daarbij is het moeilijk om de status van elke stroom bij te houden, vermits er miljoenen stromen dagelijks over het internet gaan. Zelf al kan een goed Quality of Service management het geheel verlichten, toch moeten er queues en statussen onderhouden worden voor een zeer groot aantal stromen.

Quality of Service voor IP werd ontworpen om de complexiteit van ATM tegen te gaan. Internet Protocol is beginnen werken met een model van reservatie van de resources en de signalering protocollen om aan Quality of Service te kunnen doen. Daarbij komen nog technieken bij zoals random early discard (RED) om congestie van het netwerk te verhelpen en om een goede spreiding van de aanwezige bandbreedte te verzekeren. Hier wordt er van uitgegaan dat alle stomen van een zekere soort verkeer in eenzelfde grote queue kunnen. Dit laat toch nog een zekere graad van QoS toe, zonder dat het de complexiteit en de kost van een ATM netwerk heeft. Dit is effectief zo zolang er congestie management mechanismen aanwezig zijn.

Er is in het algemeen geen optimale oplossing die voor alle gevallen geldig is. Er gebeurt altijd een afweging tussen kost, complexiteit en functionaliteit. Belangrijk is dat er eerst geanalyseerd wordt hoe het netwerk gebruikt zal worden, en daaruit vertrekkende de nodige compromissen te maken. Het lijkt wel zo te zijn dat de oplossingen meer en meer complex worden, vermits er altijd geprobeerd wordt om het beste uit het netwerk te verkrijgen. Maar er komen meer en meer mogelijkheden door de steeds stijgende rekenkracht en bandbreedte, nieuwe algoritmen voor netwerk verkeer management, en het begrip van het gebruik van het netwerk onder realistische omstandigheden.

2.7 En verder . . .

De groei van het internet verandert veel: Data, Voice en Video netwerken convergeren naar een uniek netwerk. Mobile communicatie gaat ook in de richting van multimedia applicaties. Het resultaat zal een nieuwe netwerk omgeving zijn waar data, voice en video, vaste en mobile communicatie dezelfde infrastructuur zullen gebruiken. In dit scenario zal Quality of Service de meest belangrijke parameter zijn voor het beheer van dat netwerk. Het succes van toekomstige netwerk applicaties zal afhangen van het vermogen van de providers in het leveren van de beste Quality of Service die overeenkomt met de aanvraag van de gebruikers. Het uiteindelijk doel is om overdracht van informatie toe te laten met een zo laag mogelijke vertraging om de best mogelijke ervaringen aan te bieden.

2.8 Conclusie

Quality of Service is een term dat veel technologiën overkoepelt. Het zijn allemaal technieken waarvan de bedoeling is om aan de gebruiker een betere ervaring van het netwerk aan te bieden. We komen drie verschillende niveau's van QoS tegen: Best effort, dat de laagste level is vermits ze geen garantie levert. Differentiated service is de meest voorkomende. Het geeft een kwaliteit dat redelijk goed is maar nog niet verzekert is. Guaranteed service is de sterkste van de drie. Die level biedt echte garanties aan de gebruiker van het netwerk, maar is ook de minst eenvoudige om in de praktijk te hanteren.

Quality of Service wordt beïnvloed door vijf hoofdfactoren: de beschikbaarheid van het netwerk, of het percentage tijd dat het netwerk actief is wanneer de gebruiker het nodig heeft. De hoeveelheid overgedragen data over netwerk speelt ook een rol bij QoS. Een derde element dat invloed heeft op Quality of Service is de vertraging dat kan voorkomen op pakketten die door het netwerk heen moeten. Het verschil in vertraging speelt ook een grote rol vermits het voor sommige applicaties niet toegelaten wordt. Het laatste element dat QoS beïnvloed is het verlies van data gedurende een data overdracht.

Er zijn verschillende technieken die toelaten om Quality of Service te bezorgen aan de gebruiker. Maar wat we wel altijd zien terug komen is dat er een vorm van intelligente queueing moet gebeuren, om data met hogere prioriteit voorrang te geven. Deze technieken zullen in de toekomst nog verbeteren, want Quality of Service is iets dat steeds meer belangrijk wordt.

Hoofdstuk 3

Borg

3.1 Inleiding

In dit hoofdstuk wordt Borg uit de doeken gedaan. De reden waarom Borg beschreven wordt is omdat het uitgebreid werd met Quality of Service en dat we dit verder in het document zullen gebruiken. De test cases die in volgende hoofdstukken beschreven worden zijn allemaal in deze taal gemaakt. Borg is een Mobile Multi Agent Systeem ontworpen op het Laboratorium PROG van de Departement Informatica van de Vrije Universiteit Brussel.

In het eerste deel van dit hoofdstuk bespreken we nog niet onmiddellijk Borg. Eerst komt er een stuk dat uitlegt wat agents, mobile agents en mobile multi agent systemen zijn. Dit wordt in het tweede deel gevolgd door een uitleg over wat Borg is, wat de oorsprong ervan is en waarom het ontwikkeld werd.

Het derde deel van dit hoofdstuk dient om uit te leggen hoe men met Borg kan werken. Er wordt besproken hoe een programma in Borg te maken, maar ook de verschillende onderdelen en mogelijkheden die in die taal aanwezig zijn en die van Borg de unieke taal maken die het echt is.

Het vierde deel hanteert over de communicatie aspecten van Borg. De benaming van de Borg onderdelen wordt als eerste besproken. Dan komt de routing in Borg, of hoe een boodschap van de ene plaats in het netwerk naar de andere kan geraken. Dit deel wordt beëindigd met de uitleg over hoe men iets door het netwerk krijgt in Borg.

Deel vijf legt een paar alternatieven van Borg uit, namelijk CORBA en Java RMI. Deel zes past hierbij door te zeggen wat de voordelen en nadelen van Borg zijn, tegenover de twee andere methoden maar ook in het algemeen.

3.2 Mobile Multi Agent Systemen

Borg is een Mobile Multi Agent Systeem, een platform dat agents in een netwerk ondersteunt. Maar voor we verder gaan met Borg leggen we eerst uit wat agents, mobile agent en mobile multi agent systemen zijn.

3.2.1 Agents

Agents is een term die we in de computerwereld zien terugkomen in de gebieden van gedistribueerde applicaties en artificiële intelligentie. De twee zijn in hun gebruik sterk verschillend, maar hebben toch overeenkomstige eigenschappen. Agents zijn bedoeld om autonoom te zijn in een omgeving waar nog andere agents aan het werken zijn, en ze moeten met elkaar kunnen interageren. De meest voorkomende en meest bekende agents zijn die uit de artificiële intelligentie. Deze worden meestal geprogrammeerd in high level programmeertalen en draaien op een platform dat een gemeenschap van agents ondersteunt. De bedoeling van deze agents is om intelligentie aan te werven door middel van interactie met de omgeving. Het doel dat met deze agents bereikt wil worden is meestal een studie van de evolutie van een of andere eigenschap van de reële wereld, zoals bijvoorbeeld de evolutie van taal [21].

De agents uit de gedistribueerde applicaties omgeving zijn vooral bedoeld om de traditionele client/server architectuur te vervangen. Dit is het type agents dat in deze verhandeling besproken worden. Een agent wordt hier gezien als een thread die zijn eigen code bevat en zijn eigen executie stack heeft. Er kan naar een agent gerefereerd worden door gebruik te maken van zijn naam ,die uniek is in het netwerk waarop de agent zich bevindt. Dit is bedoeld om toe te laten dat er boodschappen gestuurd worden tussen de agents. We hebben hier weer een interactie van agents op een platform die ze ondersteunt. Agents worden hier meer bedoeld als een vorm van objecten, zoals in het object georiënteerd paradigma, waar een functie op afstand opgeroepen kan worden. Het is hier vooral de bedoeling dat twee of meerdere agents die met elkaar communiceren op verschillende fysieke machines zitten. We hebben hier dus een platform die verspreid is over verschillende computers en waarop agents naar elkaar boodschappen zitten te sturen en samenwerken.

3.2.2 Mobile Agents

In de vorige sectie hebben we gezien dat agents stukken software zijn die met elkaar communiceren over het netwerk. Nu kunnen we nog een stap verder gaan door mobile agents te beschouwen. Mobile agents zijn ook agents die verspreid zijn over een platform en die met elkaar communiceren door middel van boodschappen, maar ze zijn niet verplicht om altijd op de zelfde plaats te staan. Ze kunnen op elk moment van de ene plaats in het netwerk naar een andere plaats migreren.

Er zijn twee manieren om te migreren. Er is de zwakke migratie, waar als een agent een proces aan het executeren is, dit proces eerst moet afwerken en wanneer het gedaan heeft, dan pas van plaats kan verhuizen. Er is dan nog sterke migratie, waar een agent op ender welk moment van locatie kan veranderen, zelf middenin de uitvoering van een methode.

3.2.3 Mobile Multi Agent Systemen

Een mobile multi agent systeem, waarvoor men soms ook gewoon mobile agent systeem voor zegt, is het platform die de mobile agents ondersteunt. Het moet de communicatie van de agents in handen nemen, maar ook vooral het migreren van de agents verzorgen. Het doel van een mobile agent systeem is dat het verhuizen van een agent onzichtbaar gebeurt voor de andere agents. Wanneer een boodschap naar een agent gestuurd wordt die van locatie verandert is, dan moet het platform ervoor zorgen dat de boodschap nog steeds bij de bestemmeling aankomt.

3.3 Wat is Borg

Borg is een mobile multi agent systeem dat gebouwd werd bovenop de taal Pico [22]. Deze werd ontworpen door Professor Dr. Theo D'Hondt op de afdeling PROG van de Vrije Universiteit Brussel. Het doel van Pico was om een eenvoudige programmeertaal te hebben die gemakkelijk te hanteren was voor studenten die niet afkomstig zijn uit de computer wetenschappen. Het werd ontworpen met de taal Scheme (een dialect van Lisp) in het achterhoofd, en heeft er de expressiviteit van overgenomen.

Borg breidt Pico uit met natives die de communicatie tussen agents toelaat en de mogelijkheid bieden tot het migreren van agents. De reden waarom Pico als basis gekozen werd is omdat deze werkt op basis van een stack machine, met twee interne stacks, de expressie stack en de continuatie stack, die de uitvoeringsstaat van een agent bevatten. Die Pico Virtuele Machine (VM) laat op elk moment toe om de interne status van een agent vast te nemen, zelf gedurende de executie van een programma. Dit zorgt ervoor dat Borg sterke migratie [27] ondersteunt, wat wil zeggen dat een Borg agent op eender welk moment van locatie kan veranderen.

Zowel Borg als Pico zijn experimentele programmeer omgevingen. Dit heeft natuurlijk zijn nadelen. Zo is het dat we nog regelmatig bugs tegenkomen en kan het wel eens regelmatig gebeuren dat het programma ten onder gaat. Ook is de error handling nog niet helemaal perfect, maar we vertrouwen erop dat dit allemaal in orde zal komen.

Origineel is Borg gemaakt in C. Ontwikkeling van Borg gebeurt vooral in Linux, maar de source code is zeer flexibel en laat toe om gecompileerd te worden voor verschillende platformen, waaronder Linux, Mac, Windows maar ook PalmOS. Voor deze platformen zijn ook verschillende gebruikersinterfaces beschikbaar, tekstueel of grafisch.

3.4 Programmeren in Borg

In deze sectie geven we een overzicht van hoe het programmeren in Pico en vooral in Borg gebeurt. Het is bedoeld voor de mensen die weinig of geen ervaring hebben met deze taal en kan nuttig zijn voor het verder begrip van dit document.

3.4.1 Maken van Agents

Een eerste vereiste om in borg te programmeren is om een agent aan te maken waar de code in komt. Dit is gelukkig niet te moeilijk. Laten we bijvoorbeeld uitleggen hoe het te werk gaat met de grafische Borg evaluator voor de KDE omgeving in Linux. Het aanmaken van een agent zal onmiddellijk gebeuren. Het venster dat opengaat stelt een agent voor. Om een nieuwe agent aan te maken kan men gewoon op de new knop drukken, en een tweede venster zal tevoorschijn komen dat zijn eigen agent zal voorstellen.

Nu dat we een agent aangemaakt hebben kunnen we beginnen programmeren. We bekijken een klein voorbeeld programma.

```
average(a,b)::
   {sum:a+b;
    avg:sum/2;
    avg}
```

Dit programma berekent het gemiddelde van twee getallen in Borg. We zien hier dat de taal dynamisch getypeerd is. Het declareren van een functie gebeurt via de operator '`::`'. De parameters van de functie komen in de haakjes voor de dubbele punten terecht. We zien dan dat de body van de functie tussen accolades komen. Dit is niet altijd verplicht. Wanneer de body slechts uit een lijn code bestaat mag die onmiddellijk na de dubbele punten komen.

De accolades betekenen dat er een blok code is, bestaande uit verschillende lijnen. De punt-komma's betekenen het einde van een lijn code. Het is niet zoals in C waar elke oproep moet eindigen met een '`;`'. Het moet enkel gebruikt worden in een blok code wanneer er nog een lijn achter komt. Zo zien we dat in de voorbeeld code de laatste lijn van de definitie geen punt-komma heeft.

Verder kunnen we nog de operator '`:`' merken. Deze dient voor de declaratie van een variabele. Een derde gelijkaardige operator (samen met '`::`' en '`:`') is '`:=`'. Deze dient om een variabele van waarde te doen veranderen.

Wat we in het voorbeeld niet kunnen zien maar dat wel belangrijk is, is de manier waarop we een oproep evalueert.

```
average(4,10)
->7
```

We hebben nu een klein programma gezien. Maar om iets interessantere zaken te kunnen doen hebben we controle structuren nodig. De volgende vinden we in Borg terug:

`if(cond, then, else)`

bijvoorbeeld

```
a : 5
b : 13
if(or(a>b,a=b), a, b)
->13
```

Andere zijn:

```
while(condition, expression)

until(condition, expression)

for(initialization, condition, step, expression)
```

Wat ook nog besproken moet worden is de container datastructuur die in Borg gebruikt wordt: de tabel. We kunnen er op de volgende manier mee werken:

Een table aanmaken

```
a[5]:0
->[0 0 0 0 0]
```

Een element veranderen

```
a[3]:=5
->[0 0 5 0 0]
```

Een element accesseren

```
a[3]
->5
```

Meer voorbeelden van programma's in Borg en Pico zijn te vinden op de Pico [22] en Borg [23] websites.

Soms willen we agents aanmaken die code bevatten die op file zit zonder daarvoor elke keer een grafisch venster te moeten opendoen. Die agents moeten gewoon in de achtergrond actief zijn omdat ze nuttig kunnen zijn of belangrijk zijn voor een of andere applicatie. Dit kan gelukkig ook. Er bestaat een primitieve `new(agentname, sourcecode)` in Borg die een nieuwe agent zal aanmaken. Het neemt als parameter de naam die we aan de agent willen geven en de source code die we erin willen hebben. Als we source code uit een file willen halen kunnen we dan de native functie `load(filename)` gebruiken, die de code uit de file dat het als arguments krijgt in de agent zal laden. Eens de oproep uitgevoerd zal de nieuwe agent actief zijn.

Nog een andere manier om een agent aan te maken is door gebruik te maken van de native functie `agentclone()`. Bij deze manier van werken zal er een nieuwe agent aangemaakt worden die een kopie zal krijgen van de code- en data-ruimte van de agent waarvan het een clone is.

Technisch gezien zal bij het aanmaken van een nieuwe agent een nieuwe stack aangemaakt worden en zal er geheugen gealloceerd worden voor code-ruimte en data-ruimte van de agent. De code van de agent zal dan ingeladen worden in de code-ruimte en zijn variabelen gaan in de data-ruimte.

3.4.2 Objecten in Borg

Borg werd niet ontworpen als een pure object georiënteerde taal. Maar dat houdt niet tegen dat we er wel mee met objecten kunnen werken. Het is zelfs simpel om met objecten te werken in Borg. De interne werking van de objecten is wel wat minder eenvoudig. Om deze uit te leggen moet het concept van de dictionaries worden bijgehaald. Een dictionary is eigenlijk de lokale omgeving van een agent. Het is een lijst van al de variabelen en de waarde die ze hebben. Bij het aanmaken van een object gaat de dictionary uitgebreid worden met de nieuwe variabelen en waarden die we in het object tegenkomen. Wanneer er dan een instantie van een object aanmaakt wordt moet er een referentie naar de dictionary van dat object in handen genomen worden. Dit gebeurt door op het einde van de definitie van een object de functie `clone()` te plakken. Dit zal een clone nemen van de dictionary die bij het object hoort en deze binden aan de variabele waaraan de instantie van het object gebonden moet worden.

Dit is kort de theorie achter de objecten. Nu volgt een kort voorbeeld van het gebruik ervan. We maken hier een eenvoudig counter object.

```
make_counter() ::
  { c:0;
    increment()::{c:=c+1};
    get()::{display(c)};
    clone()
  }
```

We zien dus dat de code om een object aan te maken een functie is die een object moet maken. Daarin vinden we de variabelen en de methodes dat het object kent. Om de methodes van het object op te roepen gebruiken de operator '.', zoals we kunnen zien in het vervolg van het voorbeeld, waar we met de counter gaan werken.

```
c:make_counter()
:<dictionary>

d:make_counter()
:<dictionary>

c.increment()
:1

c.increment()
```

```
:2

c.increment()
:3

d.get()
:0
```

Hier zien we goed dat elke instantie van een object zijn eigen data heeft en dat ze niet met elkaar kunnen interfereren.

Voor meer complexere voorbeelden kunnen de respectievelijke website van Borg [23] en Pico [22] geraadpleegd worden.

3.4.3 Communicatie tussen agents

De communicatie tussen de agents is in Borg een heel belangrijk onderdeel vermits het de bedoeling is om verschillende agents over het netwerk te laten samenwerken. Er zijn twee manieren om agents in Borg met elkaar te laten communiceren. De eerste is door gebruik te maken van remote message sends. Bij de tweede gaat er gebruik gemaakt worden van de send en receive primitieven die in Borg aanwezig zijn.

Remote Message Sends

Bij deze manier van werken gaan we eerst een referentie maken naar een agent. Dit gebeurt via de native functie `agent(agentname)` die als parameter de naam van een agent neemt. Het is natuurlijk best om die referentie aan een variabele te binden zodat we er later nog gebruik van kunnen maken. Verder wordt er dan gewerkt met de variabele die de referentie bevat. Dankzij de operator `->` kunnen we de methodes van de agent waarnaar we refereren gewoon oproepen, op een manier die lijkt op een methode invocatie in object gerichte talen. Het verschil met object gericht programmeren is dat de oproep hier asynchroon gebeurt, wat betekent dat er geen return waarde teruggestuurd wordt en dat het programma direct zal verder gaan. Als er een waarde terug moet komen uit de opgeroepen functie moet dit gebeuren via callbacks.

Om dit te illustreren volgt een voorbeeld van het gebruik van remote message sends.

We maken grafisch twee agents aan. In de eerste agent waar we als naam bijvoorbeeld “progpc/Alice”gebruiken laden we volgende code:

```
sendmessage(tekst)::
{
   a:agent(‘‘progpc/Bob’’);

   a->display(tekst)
}
```

In de tweede agent, die dan "progpc/Bob"zal noemen zetten we deze code:

```
sendmessage(tekst)::
{
   a:agent(‘‘progpc/Alice’’);

   a->display(tekst)
}
```

Nu komt een voorbeeld van de uitvoer van de vorige functies. In de eerste tabel komt de code dat in het venster van "progpc/Alice"moet komen en in de tweede van "progpc/Bob". Hoe dieper we in de code zakken, hoe verder het tijdsverloop is.

```
bob:agent(‘‘progpc/Bob’’)              alice:agent(‘‘progpc/Alice’’)
:<reference progpc/Bob>                :<reference progpc/Alice>

sendmessage(‘‘hallo’’)                 :hallo

:hoe gaat het                          sendmessage(‘‘hoe gaat het’’)

bob->sendmessage(‘‘goed’’)
:goed
```

Dit is natuurlijk slechts een simplistisch voorbeeld. Er kunnen met remote method invocation veel complexere systemen gebouwd worden. Andere voorbeelden kunnen op de Borg website gevonden worden [23].

Send en Receive

Naast de remote message sends is er nog een andere manier van boodschappen te sturen, namelijk door gebruik maken van de send en receive primitieven. Send zal ervoor zorgen dat een boodschap van de ene agent naar de andere gestuurd wordt. Send is asynchroon, wat betekend dat het sturen van een boodschap overeenkomt met het plaatsen van die boodschap op een leveringssysteem. De oproep keert onmiddellijk terug en wacht niet tot de boodschap bij de bestemmeling aangekomen is. Dit houdt tegen dat een programma vertraagt wordt door het wachten van het aankomen van een boodschap op lange afstanden. Maar hierdoor kan het wel moeilijker worden om fouten op te sporen.

Om te kunnen werken met send en receive hebben we weer de agent primitieve nodig die een referentie zal teruggeven naar de agent waarmee men wil communiceren. De oproep van send heeft twee parameters. De eerste is de referentie naar de agent waarvoor de boodschap bedoeld is. De tweede is de boodschap zelf. Deze laatste mag ook een data structuur bevatten zoals een lijst, wat toelaat om verschillende boodschappen tegelijk te kunnen sturen. Send laat ook toe een pattern in een boodschap te steken waar de ontvanger van de boodschap op kan zoeken.

De receive (`recv`) native neemt ook twee parameters mee. De eerste kan drie vormen aannemen:

1. Een referentie naar een agent.
2. Een wildcard
3. Een table van referenties naar agents.

Bij een wildcard zal de agent boodschappen krijgen van eender welke agent. Bij een tabel van referenties naar agenten zijn zal een boodschap enkel binnengehaald worden als de zender in die lijst aanwezig is.

De tweede parameter van de receive is een pattern waaraan de boodschap moet voldoen. Dit kan weer een wildcard zijn, maar er kan ook een structuur zijn. De ontvanger zal dan enkel de boodschappen binnenhaalt waar de inhoud aan de structuur voldoen. In elk geval krijg men bij een oproep van receive een boodschap terug als er een was en indien er geen was een native `<nomesg>`.

Ter illustratie volgt er een voorbeeld van het gebruik van send en receive. Links staat de code uit een agent "progpc/Alice" en rechts uit "progpc/Bob".

```
bob:agent(``progpc/Bob'')             alice:agent(``progpc/Alice'')
:<reference``progpc/Bob''>            :<reference ``progpc/Alice''>

send(bob, ``a message'')
:a message

                                      recv(any,any)
                                      :a message

send(bob, ``another message'')
:another message

                                      recv(alice, any)
                                      :another message

                                      recv(any,any)
                                      :<nomesg>

send(bob, [51, ``a pattern'')
:a pattern

                                      recv(alice,[51, any])
                                      :a pattern

send(bob, [52, ``another pattern'')
:another pattern

                                      recv(any,[51,any])
                                      :<nomesg>

                                      recv(any,any)
                                      :[52, another pattern]
```

Synchronisatie

De communicatie methodes tot nu toe besproken zijn allemaal asynchroon in hun werking. Maar soms kan het gebeuren dat men op een synchrone manier te werk wil gaan. Dit kan het geval zijn wanneer de zender van een boodschap moet wachten tot deze aangekomen is voor verder te kunnen werken.

Daarvoor is er in Borg de native `sync` aanwezig.

De parameters van de functie `sync` zijn dezelfde als die van de native receive. we kunnen weer meegeven met welke agents men synchroon wil gaan werken, ofwel door een referentie, een table van referenties of een wildcard. De tweede optionele parameter is weer een structuur waarmee vergeleken kan worden om te kunnen bepalen met welke boodschap men juist wil synchroniseren.

Met het behulp van `sync` kan men op de volgende manier een synchrone send (`ssend`) en receive (`srecv`) in Borg maken.

```
ssend(to,message):
{
   send(to,message);
   sync(to,message)
}

srecv(from,pattern):
{
   sync(from,pattern);
   recv(from,pattern)
}
```

Bij het gebruik van ssend moet er wel goed opgelet worden dat de ontvanger de overeenkomstige srecv gebruikt wordt. Als er een gewone recv gebruikt wordt zal de zender ongedefinieerd lang blijven wachten.

3.4.4 Een agent laten migreren

Een van de redenen waarvoor Borg ontworpen werd is om toe te laten dat een agent op elk moment van de ene virtuele machine (VM) naar een ander kan migreren zonder dat het invloed heeft op de executie van zijn programma. Dus, uiteindelijk, dat lopende code van de ene computer naar de andere kan gaan zonder dat er problemen mee zijn.

Zoals eerder gezegd gebeurt de executie van een programma in borg via twee stacks. Op elk moment kunnen deze vastgenomen worden. Van deze feature gaat er gebruik gemaakt worden wanneer een agent gaat migreren. De status van de stacks en de locale omgeving van de agent zullen omgevormd worden naar een structuur die over het netwerk kan. Ze zullen dan doorgestuurd worden en wanneer ze op hun nieuwe plaats zijn aangekomen zullen ze weer omgevormd worden zodat ze zonder probleem kunnen verder werken.

Dit is een vorm van sterke mobiliteit [27], want op elk moment kan het veranderen van werkplaats gebeuren. Het kan dus heel goed zijn dat een agent in het midden van een berekening zit wanneer de migratie plaats vind.

De native die in Borg migratie toelaat is `agentmove(naarWaar)`, met als argument de plaats naar waar de agent moet gaan. De agent waarin deze oproep gebeurt zal dan naar zijn bestemming gestuurd worden en daar weer geactiveerd worden om verder te werken. Maar om dit op een goede manier te laten gebeuren moet er opgepast worden dat de VM naar waar de agent moet migreren wel bestaat en actief is, want anders loop de agent het risico om onderweg verloren te geraken.

Met migratie ontstaat er een probleem met de uitvoer van een programma. Waar moet deze op het scherm afgedrukt worden wanneer een agent migreert. Er werd daarvoor gezorgd door de agents op te splitsen in twee delen: Een eerste deel zal zorgen voor de gebruikersinterface en een tweede voor de berekeningen. Wanneer een agent moet migreren zal de userinterface agent op de plaats blijven waar hij oorspronkelijk was en zal enkel het rekendeel verhuizen. De interactie met de gebruiker gebeurt dus steeds door het sturen van boodschappen tussen de rekenagent en de interface.

Ter illustratie van dit alles volgt een klein voorbeeld. Het is een functie dat een factorial van een getal gaat berekenen op verschillende computers. Bij elke nieuwe berekening zal het van computer veranderen. De tabel `machines` moet de namen bevatten van vier agents die actief zijn. De operator `router_placename()` gaat de locale computer teruggeven. Meer info over de benamingen van agenten komt in de sectie Routing in Borg.

```
{
    [
    agent(router_placename()+‘‘/Bob’’),
    agent(router_placename()+‘‘/Christine’’),
    agent(router_placename()+‘‘/Dirk’’),
    agent(router_placename()+‘‘/Erica’’)
    ];

fac(i)::
    {
    display(‘‘factorial ’’ + text(i));
    machine:machines[(abs(random())\\size(machines))+1];
    display(‘‘ moving to ’’+ text(machine)+eoln);
    agentmove(machine);
    if (i > 1, fac(i - 1)*i, 1)
```

```
    }
}
```

De uitvoer van `fac(5)` geeft dan bijvoorbeeld de volgende resultaat:

```
fac(5)
:factorial 5 moving moving to progpc/Bob
:factorial 4 moving moving to progpc/Christine
:factorial 3 moving moving to progpc/Bob
:factorial 2 moving moving to progpc/Erica
:factorial 1 moving moving to progpc/Dirk
:120
```

3.5 Routing in Borg

In dit deel van het hoofdstuk wordt uitgelegd hoe men het in Borg mogelijk maakt om boodschappen te versturen tussen verschillende agents. Er wordt begonnen met een uitleg over de benamingen van de agents in het netwerk. Dit wordt gevolgd door een uiteenzetting van de structuur die gebruikt wordt om een boodschap naar zijn bestemming te krijgen. Als slot komt de uitleg over hoe boodschappen gemaakt worden.

3.5.1 Naming

De communicatie tussen de agents is een belangrijk deel van Borg. Om toe te laten dat agents boodschappen naar elkaar sturen zijn bepaalde technieken nodig. Eerst moet er kunnen bepaald worden voor welke agent de boodschap bedoeld is. De agents moeten dus een unieke identificatie in het netwerk hebben. Zo kan er geen twijfel bestaan voor wie een boodschap bedoeld is. Er zijn verschillende computers in het netwerk die allemaal hun eigen gemeenschap agents kunnen dragen.

De manier van werken van de naming in Borg is als volgt. Elk computer waarop een Borg evaluator draait geeft een unieke naam aan elke nieuwe agent dat het aanmaakt. Dit zorgt ervoor dat er geen twee agents aangemaakt worden die dezelfde naam zouden dragen. Wanneer er geen naam meegedeeld wordt voor de nieuwe agent die aangemaakt moet worden, dan gaat het platform er zelf een kiezen. Het kiezen van de naam gebeurt niet random. De namen zitten in een file die Borg inleest bij het opstarten. Elke keer een nieuwe agent aangemaakt worden gaat de VM in de file kijken welke de volgende vrije naam is. Indien men een agent wel een naam geeft, en dat deze op de computer al in gebruik is, dan zal de evaluator de naam bewaren maar zal er een nummer aan toe voegen zodat de naam toch uniek blijft.

Nu hebben de agents allemaal unieke namen op een evaluator. Maar een Borg netwerk is gemaakt uit verschillende agentgemeenschappen. Toch moeten de benamingen uniek blijven. Dit gebeurt door voor de naam van elke agent de naam van de Virtuele Machine waar het aangemaakt werd te plakken. De naam van de VM komt overeen met de hostname van de computer waar de VM op draait. Dit heeft als voordeel dat we ook onmiddellijk weten waar een agent zich bevindt door naar zijn naam te kijken, toch zolang de agent niet gemigreerd heeft.

De uitleg over de naming is nog niet volledig. Er is nog een extra deel dat bij de naam van een agent komt om een problemen te vermijden. Want met de bovenstaande techniek is er nog een probleem als we twee VM's op eenzelfde machine willen laten werken. De tweede versie zal niet kunnen opstarten omdat de naam van de host al gebruikt wordt. Dit kan ook voorkomen wanneer toevallig twee computers dezelfde naam dragen. Net zoals de agents unieke namen hebben op een VM moeten de VM's unieke namen hebben in het netwerk. Dit wordt gecontroleerd via een nameserver. Wanneer een nieuwe Borg sessie geopend wordt zal er een router lokaal aangemaakt worden. De router zal zich dan binden aan een nameserver en zal deze een naam geven waaronder de router verder geregistreerd wil zijn. Als deze naam al ingenomen is door een andere router zal een nieuwe naam gegeven moeten worden. Het vinden van een vrije naam gebeurt door het toevoegen van een getal aan de hostname van de computer. Als we op de computer “progpc”werken, dan zal de eerste naam die gekozen wordt “progpc.0”zijn. Als deze al bezet is zal er “progpc.1”gekozen worden en zo verder. De eerste agent die dan op deze computer aangemaakt zal worden is “progpc.0/Alice”. De tweede zal “progpc.0/Bob”zijn. Als er een tweede evaluator op dezelfde machine komt zal zijn eerste agent dan “progpc.1/Alice”noemen en zo verder.

3.5.2 Routing

In het vorig onderdeel hebben we gezien hoe we unieke namen konden geven aan de agents in het systeem. We kunnen dus de bestemming van een boodschap op een unieke wijze bepalen. Nu weten we nog niet hoe een boodschap van zijn vertrekpunt tot zijn bestemming moet geraken.

Er werd in het vorig deel al het bestaan van routers en nameservers vermeld. Het systeem maakt hiervan gebruik om een forwarding structuur te bouwen waardoor de boodschappen moeten passeren. De gebruikte structuur die de Borg routers verbindt is een boom [26]. Wanneer een router in het netwerk komt zal deze in een configuratie file opzoeken of hij een nameserver kent. Hij gaat die dan in het netwerk opzoeken om te zien indien deze actief is. Als dit het geval is gaat hij zich bij die nameserver aanmelden. Zijn naam een locatie zullen dan bijgehouden worden in de nameserver. Wanneer er geen nameserver gevonden wordt zal de router zelf als een nameserver dienen. Een router die niet bij een andere nameserver geregistreerd is zal dan de top van een boom zijn. De routers die wel met die server verbonden zijn, zijn dan de kinderen van die server.

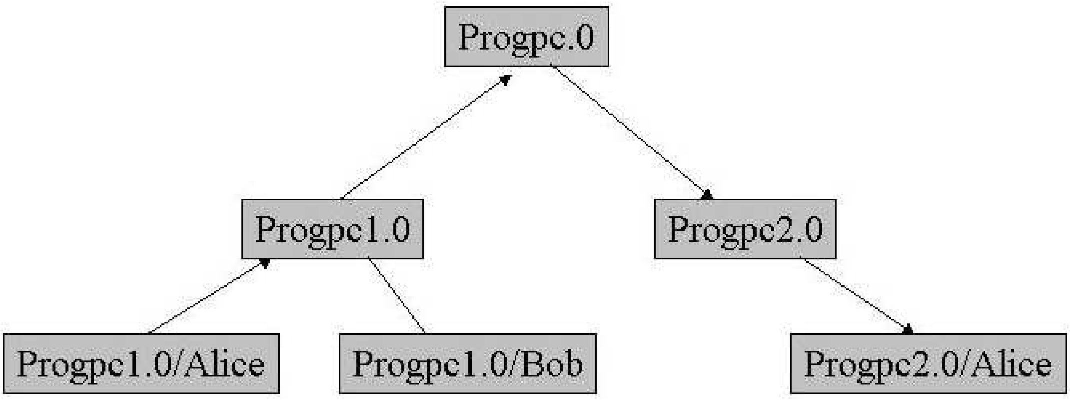
Bij het aanmaken van een nieuwe router moet deze niet speciaal gebonden worden aan de top node van de boom. Het kan evengoed aan een andere knooppunt binden. Zo kan een router die al een kind is van een node op zijn beurt een ouder worden en zo komt de boomstructuur tot stand.

We kennen nu de structuur van de connecties tussen de verschillende routers. Het volgende gebeurt wanneer een boodschap gestuurd wordt:

- Eerst komt die boodschap terecht in de queue van de router op de lokatie van de verzender.
- Die router zal dan kijken of de bestemming van de boodschap toevallig niet op dezelfde computer is.
- Als dat het geval is zal de boodschap onmiddellijk geleverd worden.
- Indien niet zal de router kijken of de agent waartoe het boodschap bestemd is niet afkomstig is van een router dat een afstammeling van hem is.
- Moest het zo zijn zal de boodschap dan naar dat kind gestuurd worden.
- Indien hij de bestemming niet kent stuurt de router de boodschap dan naar zijn ouder.
- De boodschap zal zo verder omhoog in de boom gaan totdat het op een knoop komt dat de bestemming als een van zijn kinderen heeft.
- Dan zal de boodschap in de gepaste richting de boom weer afzakken tot zijn bestemming.

In figuur 3.1 vindt men een grafische voorstelling van de architectuur van de Borg routers en hoe een boodschap tot zijn bestemming komt.

In dit systeem merken we ook dat niet enkel de router bij de nameserver moet registreren, maar ook elke nieuwe agent, en wanneer een router bij een nameserver bind moet de nameserver dat verder geven aan zijn ouder knoop.



Figuur 3.1: De hiërarchie van de routers in Borg

Dit systeem werkt goed maar moet nog uitgebreid worden, want Borg is niet enkel een communicatie systeem. Het ondersteunt ook nog migratie. Wanneer een agent migreert willen we dat deze verder nog zijn boodschappen blijft krijgen. Hiervoor wordt gezorgd met de volgende stappen:

- De router waar de agent was voor zijn vertrek moet zijn link naar de agent aanpassen naargelang de nieuwe locatie van de agent.
- Indien deze locatie in de kinderen van de router zitten zal de link aangepast worden naar het juiste kind.
- Indien het niet het geval is gaat de link verwijzen naar de ouder van de router.
- De agent zal alleszins ook naar de goede router verhuizen.
- Voor alle routers waar de agent een tak naar boven gaat volgen in de boom zal de link naar de agent vervangen worden door een link naar de ouderknoop.
- Wanneer de agent dan op een knoop geraakt is dat een ouder is van de lokatie naar waar hij moet gaan, zal de agent via de gepaste tak weer naar beneden dalen en zullen de links onderweg weer aangepast worden.

Een boodschap dat gestuurd wordt naar een agent dat gemigreerd heeft kan altijd de juiste weg terug vinden. Die boodschap is niet eens verplicht om naar de oorsprong van de agent te gaan. Het moet gewoon naar omhoog in de boom to het een router tegenkomt die de agent in een van zijn takken heeft zitten.

3.5.3 Serializing

In de vorige onderdelen werd er uitgelegd hoe elke agent een unieke naam kon krijgen en hoe we aan routing konden doen zodat elke boodschap altijd op zijn bestemming aankomt. Om compleet te zijn moet er nog wat uitleg komen over het bouwen van de boodschappen.

Een boodschap kan niet zomaar door het netwerk. Het moet eerst bewerkt worden. Dit is wat men serializing noemt. Dit moet gebeuren omdat de meeste boodschappen die door het netwerk gaan afkomstig zijn van remote message sends, waar een functie uit een agent wordt opgeroepen vanuit een andere agent. De parameters die meegegeven worden als argumenten zijn meestal Borg expressies. Deze hebben een structuur die niet altijd lineair is. Ze kunnen soms in boomstructuren zitten of zelf circulair zijn. Zo een expressie kan dan niet door het netwerk. Het moet eerst "platgedrukt"worden, serieel gemaakt worden. Serializing gaat daarvoor zorgen: Het gaat een Borg expressie omvormen naar een expressie die over het netwerk gestuurd kan worden. Wanneer een boodschap bij zijn bestemming aankomt moet de Borg expressie weer zijn originele vorm terug krijgen. Dit gebeurt in een deserializing proces.

Een Borg expressie kan interne referenties naar andere delen van het geheugen van de Borg agent bevatten. Deze pointers zijn na het sturen van het bericht niet meer geldig. De plaatsen naar waar de pointers refereren zullen bij de bestemmeling op een andere plaats zitten. De referenties moeten dan allemaal ook aangepast worden om met het nieuwe geheugen schema overeen te komen. Dit gebeurt ook in het deserializing proces.

Er is nog een type boodschap waarmee serializing extra rekening moet houden. Deze boodschappen zijn namelijk de agents zelf. Tot nu toe wisten we dat agents konden migreren maar er werd nog niet besproken hoe dat in zijn gang ging. Een agent die de functie heeft opgeroepen om de verhuizen zal door zijn lokale router geserialiseerd worden. Deze agent zal dan verder door het netwerk behandeld worden als een gewone boodschap, behalve dan dat de links naar de agent aangepast zullen worden. Eens de agent op zijn bestemming is aangekomen gaat de router deze boodschap als een agent identificeren. Het zal dan de deserialisatie proces erop uitvoeren die de agent terug zal opbouwen en toelaten om verder te werken alsof er niets gebeurd was.

3.6 Andere manieren van werken dan Borg

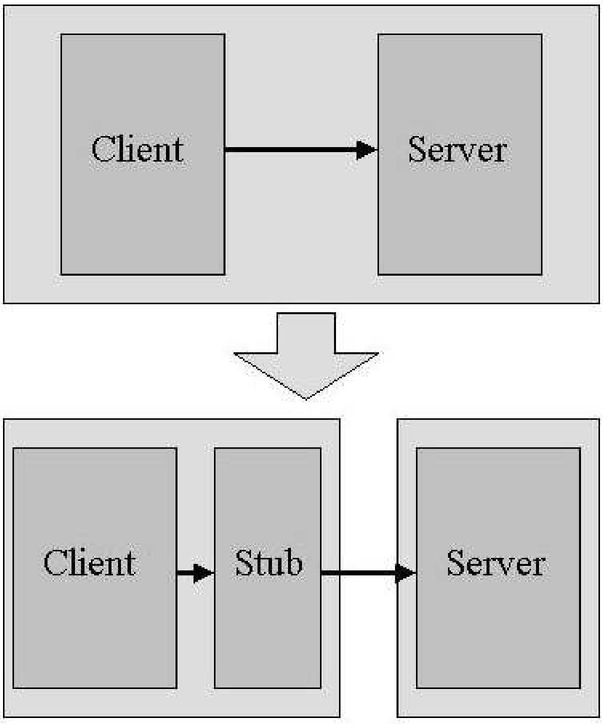
Borg wordt gebruikt om gedistribueerd te programmeren en om code over het netwerk te verhuizen om het op andere computers te laten runnen. Borg is niet het enige systeem dat deze eigenschappen aanbiedt. In deze sectie zullen een paar alternatieven van Borg uit de doeken worden gedaan.

3.6.1 CORBA

CORBA [24] staat voor het Engels "Common Object Request Broker Architecture". Het werd in 1991 door de Object Management Group (OMG) uitgebracht en het definieert een Interface Definition Language (IDL) en een Application Program Interface (API) die een client/server interactie toelaat binnenin een specifieke implementatie van een Object Request Broker (ORB).

De ORB is een programma die toelaat om een client/server communicatie te starten tussen verschillende objecten. Door gebruik te maken van de ORB kan een client transparant requests doorsturen naar een server object, onafhankelijk van de feit dat deze op de zelfde computer zitten of niet. De ORB zal de oproep van de client opvangen. Het zal dan op zoek gaan naar een server die de oproep implementeert en zal de argumenten van de oproep doorgeven en verder het resultaat van de oproep aan de client aanbieden. De client moet geen rekening houden met waar het object zich bevindt, noch met de programmeertaal waarin het object gemaakt werd, de operating system waarop het draait, of eender welke systeem specifieke details die geen deel uit maken de object interface. Het gebruik van een ORB laat dus een werking toe van verschillende objecten op verschillende computers in een heterogeen gedistribueerd systeem en geeft een interconnectie tussen verschillende object systemen.

CORBA is dus geen programmeertaal op zichzelf maar een specificatie voor het aanmaken en het werken met gedistribueerde objecten. Er zijn verschillende implementaties van deze standaard te vinden. De communicatie tussen de verschillende delen van een systeem worden door de ORB beheerd en gebeurt via stubs. Het werken met stubs gaat als volgt. Wanneer een object (een client) een ander object (een server) nodig heeft, die mogelijk op een andere computer is, gaat er een stub aangemaakt worden bij het eerste object die de berichten naar tweede zal doorsturen. Elke oproep van de client voor de server gaat eerst naar de stub opgestuurd worden. Deze zal de server in het netwerk opzoeken en de boodschap doorgeven. Het resultaat van de oproep zal dan aan de client aangeboden worden. De stubs worden gemaakt via de IDL en vormen een interface tussen de verschillende object systemen. Dit laat toe om elke component van het systeem te maken op de meest geschikte operating system met de meest geschikte programmeeromgeving en taal. Figuur 3.2 geeft een grafische voorstelling van het werken met stubs.



Figuur 3.2: Werken met stubs

Het gebruik van CORBA veroorzaakt dat er meer code geschreven moet worden. Naast de gewone client/server code moet er ook nog een IDL gemaakt worden voor de objecten. Er moet dan ook extra gewerkt worden om de verschillende delen met elkaar te kunnen linken. De IDL moet gecompileerd worden met een specifieke compiler die bij de ORB geleverd wordt. De zo gegenereerde code moet dan samen met de applicatie code gecompileerd worden. Er komt dus meer werk om de componenten met elkaar te kunnen laten werken.

Een ander nadeel heeft te maken met wat er als argument aan de functie oproepen mag meegegeven worden. Taal specifieke componenten kunnen niet doorgestuurd worden. De argument moeten voldoen aan algemene voorwaarden die elke verschillende taal uit het systeem kan ondersteunen.

Migratie is niet mogelijk onder CORBA. Het is ten eerste al goed mogelijk dat een component niet werkt op een andere computer. Een stuk code gecompileerd op computer A zal niet altijd werken op computer B. Daarbij hebben de gebruikte programmeertalen geen natives die migratie ondersteunt. Het zou dus de verantwoordelijkheid van de middleware CORBA worden, dat normaal onopgemerkt moet blijven bij het programmeren van de verschillende objecten. Met remote procedure call gebeurt de tussenkomst van CORBA zonder dat de client het merkt dankzij het gebruik van de stubs die de oproepen naar de server sullen opvangen.

3.6.2 Java RMI

Java RMI (Remote Method Invocation) [25] is een systeem die de communicatie toelaat tussen objecten die zich op verschillende virtuele machines bevinden, juist zoals CORBA. Het verschil ligt erin dat het enkel over Java objecten gaat.

De communicatie in een Java RMI proces is afkomsting van de client. Deze stuurt een identificatie door van het object waarmee het wil communiceren naar een RMI registry. Deze zal aan de client een stub teruggeven waarmee de client verder zal werken. Elke oproep naar de stub zal door de RMI architectuur omgevormd worden naar een oproep naar de server. De server bevat een thread die de inkomende berichten dispatcht naar de gepaste objecten. Eens het object zijn methode uitgevoerd heeft zal het antwoord naar de client doorgestuurd worden zodat deze verder kan werken.

Java code wordt gecompileerd naar een machine onafhankelijke vorm. De gecompileerde code, dat ook Byte-Code genoemd wordt, draait sneller dan de geïnterpreteerde source code. Toch moet de Byte-Code verder nog geïnterpreteerd worden door de Virtuele Machine. Dit heeft als voordeel dat de Java code op ender welke platform die een Java Virtuele Machine heeft kan draaien.

Uit de feit dat de code geïnterpreteerd wordt en dat het machine onafhankelijk is zou men kunnen denken dat het mogelijk is om deze code te migreren. Dit is niet het geval. Vermits we niet aan de runtime stack en aan de interne data van een object kunnen raken, kunnen we deze niet serialiseren en daarna verder laten werken. Om dit toe te laten moeten we onze eigen executie stack maken of de Virtuele Machine aanpassen.

3.7 Voor en nadelen van Borg

Borg heeft tegenover de programmeer methodes uit het vorig deel een groot voordeel, en dat is zijn mobiliteit. Het is iets dat weinig voorkomt maar die de mogelijkheid biedt tot het maken van illicite applicaties. Men kan bijvoorbeeld een applicatie maken waar een client van de ene plaats naar de andere gaat. Een gebruiker zou maar één client moeten hebben en deze zou de gebruiker mee volgen naar een andere computer indien hij van plaats zou veranderen. Men kan zo als voorbeeld een mail client nemen. We willen onze mail zowel op werk als thuis kunnen nakijken. Om geen twee verschillende clients te moeten hebben en problemen te krijgen van de aard dat een mail op een computer wel is maar op de andere niet, gebruiken we een Borg mail client, die, wanneer we van werk weggaan, ook naar een andere computer gaat en dan op de computer thuis zou terecht komen om ermee verder te kunnen werken.

Een ander voordeel van de mobiliteit van Borg merken we wanneer we met programma's werken die veel moeten berekenen. Soms heeft men die berekeningen vlug nodig maar de computer waarop men aan het werken is niet krachtig genoeg om die berekening snel uit te voeren. Met Borg kunnen we die berekening doorsturen naar een andere machine die meer rekenkracht heeft. Dit kan bijvoorbeeld nuttig zijn op een Palm device, die maar een beperkte rekenkracht heeft. Men kan van de mobiliteit van de Borg code ook gebruik maken wanneer we verschillende zware berekeningen aan het doen zijn. We kunnen de berekeningen heel gemakkelijk verdelen over verschillende computers in het netwerk.

Een ander gebruik voor de mobiliteit van Borg agent is een applicatie die gegevens moet verzamelen en vergelijken. Laten we het voorbeeld nemen van een agent die prijzen moet vergelijken voor vliegtuig tickets. Men zou die agent dan gewoon doen verhuizen naar de computers van de vliegmaatschappijen en daar de gegevens oppikken. Na langs de verschillende compagnieën te zijn geweest zou de agent terug naar zijn vertrekpunt keren om aan zijn gebruiker de verzamelde informatie te tonen.

Een ander voordeel van Borg ligt bij het Remote Message Sends. Tegenover CORBA en Java RMI, waar naast client en server code ook nog communicatie code moet geschreven worden, moet er in Borg enkel de client en de server geprogrammeerd worden. Het gebruik van de remote message sends gebeurt volledig asynchroon. Wanneer een boodschap gestuurd wordt komt er geen return waarde terug. Dit kan als een voordeel gezien worden, vermits een agent niet vastzit bij het versturen van een boodschap omdat langs de andere kant de ontvanger druk bezig is. Maar dit kan ook een nadeel zijn, vermits dat we om een resultaat terug te krijgen andere technieken moeten gebruiken, zoals callbacks. Dit leidt dan tot het maken van veel kleine functies en het gebruik van globale variabelen om te kunnen verder werken. Dit is niet altijd leuk om te doen en kan zorgen voor moeilijkheden wanneer we een programma wil debuggen.

Een ander nadeel van Borg dat we hier zullen bespreken is de veiligheid.

Het probleem is dat Borg er weinig of geen heeft. Maar vermits Borg gebruikt wordt voor onderzoek is dit meestal een voordeel, een zorg minder voor de programmeur. Op commercieel vlak zou dat natuurlijk wel een groot nadeel zijn. Ook zijn er problemen met error handling in Borg. Sommige fouten in een programma blijven ongemerkt. Dit maakt het debuggen van een applicatie vaak een ingewikkelde zaak. Er zijn daarbij nog foutsoorzaken zoals het migreren van een agent naar een niet bestaande computer die steeds ongemerkt blijven, wat meestal tot het verlies van die agent leidt. Ook het sturen van een boodschap naar een agent die niet bestaat of weggevallen is wordt door Borg niet opgenomen. Gedistribueerd werken met Borg is dus eenvoudig, maar er moet wel heel goed mee opgepast worden.

3.8 Conclusie

In dit hoofdstuk hebben we gezien wat Borg was. Het is een Mobile Multi Agent Systeem dat gebruikt wordt voor onderzoeks doeleinde. Het laat toe om vrij eenvoudig gedistribueerd te programmeren dankzij ingebakken natives die communicatie toelaat zonder dat de gebruiker er zich zorgen over moet maken. Het is ook vrij krachtig dankzij het toelaten van migratie van zijn agenten. Hiermee moet natuurlijk wel opgepast worden. Borg is slecht een experimentele taal en daarom kunnen soms zaken mislopen zonder dat de programmeur het echt opmerkt, wat het bouwen van applicaties soms een zware taak kan maken.

Borg werd hier uitgelegd omdat het verder in dit document gebruikt wordt als programmeertaal voor de test cases. Borg werd vooral gekozen omdat het een netwerk opbouwt met zijn eigen router. Op die manier konden Quality of Service eigenschappen uitgetest worden op software niveau en moesten we niet beschikken over een reeks hardware router voor het onderzoek.

Hoofdstuk 4

Quality of Service in Borg

4.1 Inleiding

In de vorige hoofdstukken werd besproken wat Quality of Service en Borg zijn. In dit hoofdstuk gaan we de twee combineren. De router van Borg kreeg een nieuwe versie die Quality of Service eigenschappen heeft. Deze uitgebreide router wordt in dit hoofdstuk besproken.

Voordat de aanpassingen die uitgevoerd werden besproken worden moet er een uitleg komen over de structuur van de originele router. Dit is wat in het eerste deel van dit hoofdstuk gebeurt. De verschillende delen van de router worden daar besproken.

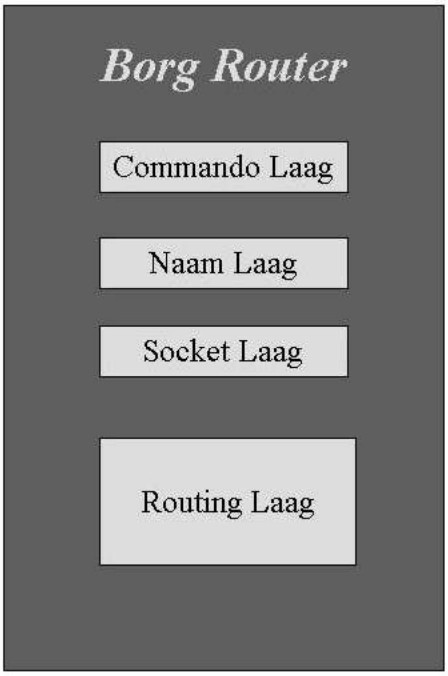
Het tweede deel hanteert dan over de aanpassingen. Deze zijn voornamelijk het vervangen van de router queue naar een priority queue en toevoegen van congestie controle in de router.

Het laatste deel behandelt de verschillende aspecten van Quality of Service. Ze krijgen allemaal eerst een eigen definitie die overeenkomt met de concepten van de Borg router. Vervolgens wordt er uitgelegd welke de toegevoegde waarde van de Quality of Service uitbreidingen voor hen is.

4.2 De Borg Router

Om quality of service te kunnen toevoegen aan Borg moet er gewerkt worden in de Borg router. Voor de aanpassingen te bespreken wordt er eerst uitgelegd hoe de router in zijn werk gaat. Dit is natuurlijk niet eenvoudig. Het is een file van meer dan duizend tweehonderd lijnen C code.

De code is opgesplitst in verschillende delen. Figuur 4.1 toont de verschillende onderdelen waaruit de router gemaakt is.



Figuur 4.1: De architectuur van de Borg router

Het eerste deel is een commando gedeelte. Commando is een ADT voor de structuur van de boodschappen dat tussen agents gestuurd worden. Alle velden van een boodschap komen terecht in deze structuur en deze wordt dan ook in de rest van de code gebruikt om boodschappen voor te stellen. Er zijn een paar speciale functies die erop worden gedefinieerd om de verschillende delen van het ADT te accesseren en te kunnen wijzigen. Er zijn dan ook nog een paar hulpfuncties om het geheel te beheren.

Het tweede deel is de naam laag. Daar wordt een abstract data type gebouwd om informatie bij te houden over elke agent die door de router gekend is. De gegevens die in deze laag bijgehouden worden zijn de laatste gekende locatie van de agent, het moment waarop de locatie het laatst nagekeken werd en het hoeveelste adres van de agent dat is. Dit wordt gebruikt om bij communicatie tussen de verschillende routers in het netwerk te kunnen bepalen welk adres het meest kans heeft om correct te zijn, wanneer er verschillende adressen voor eenzelfde agent gekend zijn. Er is nog een ander element dat bijgehouden wordt, en dat is de nummering van het aantal berichten gekregen van en verstuurd naar die specifieke agent. Die nummering wordt meegegeven samen met het bericht zodat de ontvanger kan beslissen welke de oorspronkelijke volgorde van de boodschappen was. Er zijn dan natuurlijk ook nog functies om dit geheel te initialiseren, om het te kunnen aanpassen en om ermee te kunnen werken.

Een derde laag is die van de sockets. De sockets zijn stukken software die poorten voorstellen op een computer. Dit laat toe om een verbinding te maken tussen verschillende computers door een kanaal op te zetten van een socket naar een andere. De data die dan tussen de computers moet verstuurd worden wordt naar de locale socket gestuurd en deze zal er dan voor zorgen dat de data over het netwerk geraakt. Het lezen van inkomende data gebeurt door bij de locale socket te kijken of er iets in zijn buffer aanwezig is en dit dan uit te lezen.

In de Borg router maakt men gebruikt van Borg sockets, een interface die ervoor zorgt dat de router zou kunnen werken op verschillende operating systems, want deze hebben verschillende manieren van werken en implementatie van de sockets. De router zal dan oproepen maken naar de interface die hoort bij het operating system en zal dan de gepaste manier van werken gebruiken. Deze manier van werken is verplicht om Borg platform onafhankelijk te maken.

De socket functies die men in de router terugvindt zijn aanwezig om het commando ADT dat de boodschappen in de router voorstelt op te sturen door het netwerk en om ze verder ook terug te kunnen inlezen. Zo kan alles altijd in de goede volgorde gebeuren, want de volgorde is bij socket communicatie heel belangrijk. Als de ontvanger een bepaalde volgorde verwacht en dat de zender er een andere gebruikt, dan kan het geheel bij de ontvanger op een foute pad gaan.

Het vierde deel van de router is het echte routing gedeelte. Daar bevindt zich eerst en vooral de code voor de router queue. Alle berichten, die in commando vorm zijn, komen in deze router queue terecht. Dit geld zowel voor de berichten die in de router binnenkomen door middel van communicatie met een andere router, als voor de berichten die afkomstig zijn van lokale agents en die de router verder moet sturen. Naast nog een paar hulpfuncties komen we bij het behandelen van de boodschappen. Een eerste functie gaat kijken of er op de queue wel berichten aanwezig zijn en, als dit het geval is, gaat het een tweede functie oproepen die dan de eerste boodschap zal nemen en kijken wat ermee gedaan moet worden. Als het een lokale boodschap is die verzonden moet worden, dan wordt het bericht aan de gepaste agent afgeleverd. Indien het een bericht is dat verder gestuurd moet worden, dus een boodschap die van een andere router kwam maar waarvan de ontvanger niet op de lokale computer is, dan wordt het verder gestuurd naar de geschikte router. Indien het bericht bestemd is voor de lokale router of een van de agents die erop aanwezig zijn, dan zal dit bericht verder behandeld worden naar gelang het type boodschap.

Alle boodschappen die men in de router terugvindt dienen niet allemaal gewoon voor communicatie tussen verschillende agents. We weten uit het deel die over de communicatie handelt in het Borg hoofdstuk dat de migratie van agents ook via berichten tussen de verschillende routers gebeurt. Daarbij zijn er dan ook nog berichten die zorgen voor acknowledgements, wat een router op de hoogte stelt dat een eerder gestuurd bericht op zijn bestemming aangekomen is. Er zijn ook nog boodschappen die gestuurd worden om een router te verwittigen dat een bepaalde agent van plaats gemigreerd heeft. Naast de berichten tussen routers zijn er ook nog die tussen een router en de nameserver waarmee de router gebonden is. Er zijn de berichten om de router te binden of te ontbinden van een nameserver, dus te verwittigen dat ze actief zijn of niet, of ook gewoon om te vragen of de nameserver nog actief is. Er zijn ook boodschappen afkomstig van de nameserver om te zien of een agent op de router nog leeft of niet.

Een onderdeel van de router die we niet vermeld hebben is de interfacing tussen de Borg router en de lokale computer. Dit deel staat in voor de initialisatie van de router. De naam en het IP nummer van de locale machine worden opgezocht, zodat de Borg evaluator zijn naam kan bepalen. Er wordt dan een file opengemaakt die de lokatie van de nameserver bevat waar de router zich mee moet binden. De nameserver wordt opgezocht en indien deze actief is wordt de router ermee gebonden. Moest de nameserver niet reageren, dan zou de locale router zichzelf als nameserver opstellen.

4.3 De Quality of Service Router

De Borg router, zoals in de vorige sectie besproken, ondersteunde oorspronkelijk geen Quality of Service. De uitbreiding om Quality of Service aan te bieden gebeurde gedurende mijn stage periode op het laboratorium PROG. De router met de Quality of Service uitbreiding is geen vervanging voor de originele router, maar een alternatief erop. De gewone router blijft dus nog bestaan en werkt samen met de nieuwe versie.

Nu is het zo dat als we kijken naar Quality of service zoals het in hoofdstuk 2 uitgelegd staat, het opvalt dat het iets is dat bepaald wordt op hardware niveau, ter hoogte van de netwerk routers. Daaraan kunnen we weinig doen vanuit Borg. Borg is op zichzelf een netwerk die zich bovenop het gewone netwerk bevindt. Daarbij komt nog dat er in plaats van op verschillende netwerk pakketten te werken we met boodschappen beginnen werken. Alle communicatie tussen de Borg routers gebeurt namelijk via boodschappen, die allemaal dezelfde structuur hebben en op dezelfde manier door het netwerk moeten. Een boodschap die door het netwerk moet is op zichzelf opgebouwd uit verschillende pakketten, namelijk een voor elk veld die we in het commando ADT uit het vorige onderdeel terug vinden. Daarom werd Quality of Service voor dit onderzoek een level hoger geplaatst. Het is op dat niveau dat er in Borg Quality of Service werd toegevoegd.

Het was al vermeld geweest dat we de Borg router zouden moeten aanpassen. De eerste stap om dat te doen was door te kijken welke de belangrijkste aanpassingen zouden zijn die kunnen leiden tot het aanbieden van Quality of Service. Wat uit het hoofdstuk over Quality of Service kan afgeleid worden is dat er een grote behoefte is aan een vorm van prioriteiten in de router queue en een systeem dat de congestie controleert. Dit is dus wat er toegevoegd moest worden.

4.3.1 Priority Queue

Wanneer een boodschap in de queue van een router terecht komt, dan moet de router kunnen zeggen welke prioriteit die boodschap heeft en er op een correcte manier mee kunnen handelen. Een router moet dus eerst kunnen bepalen wat de prioriteit van een bericht is. Vermits elke bericht een vast ADT volgt kan dit gewoonweg door het ADT uit te breiden. Het commando ADT uit de vorige sectie krijgt een veld meer dat de prioriteit van de boodschap bevat. Deze zal dan, samen met de rest van het bericht, mee over het netwerk gestuurd worden.

Nu dat de router de prioriteit van een boodschap kan bepalen moet het nog kunnen bepalen wat het ermee moet doen. Daarvoor moet een vorm van protocol opgebouwd worden die de prioriteiten vastlegt. Hetgeen gekozen werd is om met vijf prioriteiten te werken, vijf zijnde de hoogste prioriteit, één de laagste. Een boodschap met hoge prioriteit zal dan voorrang moeten krijgen op een boodschap met een lagere prioriteit. Zo zal een bericht dat een hoge prioriteit heeft nooit vast blijven zitten achter een reeks boodschappen met lage prioriteit. Standaard bepalen we dat een boodschap een prioriteit 3 heeft. Zo kan een boodschap versneld worden, maar ook vertraagd wanneer het afkomstig is van een applicatie waar de snelheid niet van belang is.

Eén Queue

Er zijn verschillende manieren om nu verder te werken met de prioriteiten. In de vorige sectie hebben we getoond dat de Borg router intern met een queue werkt, waar alle boodschappen in terecht komen. Een eerste manier om dit uit te breiden zou zijn om deze queue om te vormen naar een klassieke priority queue. Wanneer een nieuwe boodschap aankomt zal er dan gezocht worden naar welke plaats die in de queue moet innemen en zal de queue aangepast worden. Wanneer de router een nieuwe boodschap moet behandelen zal die dan gewoon het eerste element uit de queue halen.

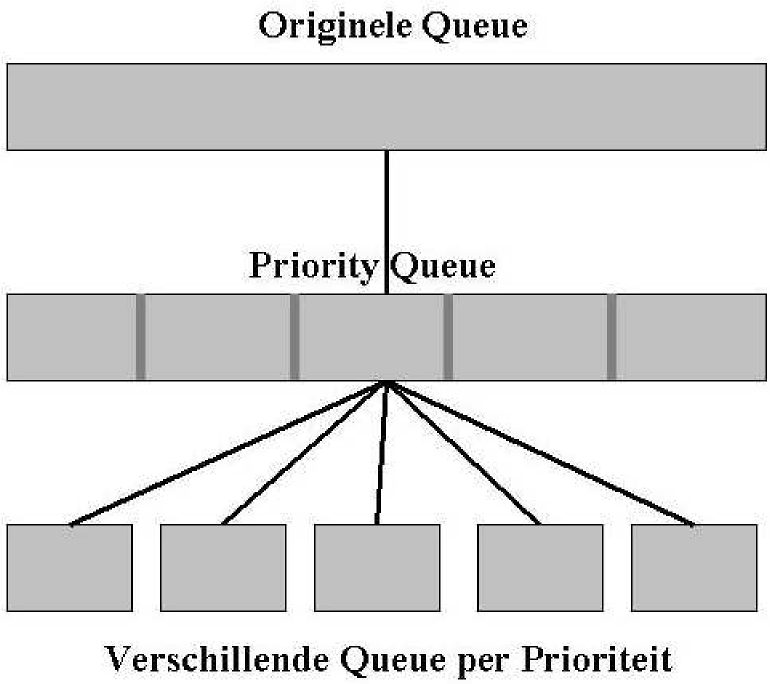
Deze manier van werken heeft een serieus nadeel. Het toevoegen van een element aan de queue gebeurde in de oorspronkelijke versie van de router in grootte orde O(1). Het was dus een operatie die in een constante en korte tijd gebeurde. Maar met de priority queue is dit heel anders. Om te kunnen bepalen op welke plaats een element moet komen moet de queue eerst eens doorlopen worden. Dit duurt langer naar gelang de queue groter wordt. De efficiëntie van het algoritme gaat dus een serieuze stap achteruit.

Een ander probleem dat tevoorschijn komt is die van de oudere boodschappen die afkomstig zijn van dezelfde agent en die dezelfde bestemming hebben. Als de oude boodschappen een lagere prioriteit hebben zullen deze moeten wachten dat de nieuwe boodschap doorgestuurd word. In eerste plaats zal dit voor jitter [3] zorgen vermits de boodschappen niet in de volgorde zullen aankomen waarin ze verstuurd werden. Die jitter zal een vertraging veroorzaken op de boodschap die een hoge prioriteit heeft. Het is zo dat alle boodschappen aan de ontvangstzijde op de juiste volgorde behandeld moeten worden. Als de boodschappen met een lage prioriteit tegengehouden worden in het netwerk door boodschappen met hogere prioriteiten, dan zal de boodschap die snel behandeld moest worden heel lang wachten en zal zijn effectieve prioriteit gereduceerd worden naar de laagste prioriteit van de boodschappen die voor hem gestuurd werd en die nog niet op hun bestemming waren aangekomen. Als we nu een constante stroom van berichten tegenkomen die een grote prioriteit hebben en die berichten volgen waarvan de prioriteit laag is, dan zou het resultaat catastrofaal kunnen zijn. De berichten mogen steeds in een router voor de oude berichten de tijd hebben om behandeld te worden. Door de stroom van hoge prioriteiten geraken de oude berichten nooit behandeld, en dus zullen de nieuwe berichten, wanneer die op hun bestemming aankomen, ook niet behandeld kunnen worden. We komen dan in een deadlock en in een status van hoge congestie terecht die kan leiden tot het falen van een Borg router.

Voor dit probleem werd er gelukkig een oplossing gevonden. Voordat een nieuwe boodschap in de priority queue gestoken wordt gaat men eerst nagaan of de queue niet al boodschappen bevat met een lagere prioriteit die van dezelfde agent afkomstig zijn en dezelfde bestemming hebben. Als dit het geval is worden die elementen uit de queue gehaald. Men geeft hen dan een nieuwe prioriteit die overeenkomt met die van het nieuwe bericht en steekt ze terug in de queue. Dit zal dan zorgen dat alle oude boodschappen de prioriteit krijgen van de nieuwe, wat we liever zien gebeuren dan het omgekeerde. Het nadeel is dan dat het invoegen van een nieuw element in de queue nog veel langer kan duren vermits de queue eerst doorzocht moet worden en dat elke oude boodschap ook nog eens in de queue op de juiste plaats moet gestoken worden. De efficiëntie gaat dus sterk achter.

Meerdere Queues

Een oplossing op dit efficiëntie probleem vindt men door de priority queue op een andere manier te bekijken. De router queue hoeft niet per se één queue te zijn, het kan evengoed in verschillende stukken opgedeeld worden. We kunnen hier een nieuwe queue nemen voor elke verschillende prioriteit. Dan hebben we hier vijf queues in de router. Bij het invoegen van een element hoeft men dan gewoon te kijken naar de prioriteit dat die het nieuwe element heeft en het aan de juiste queue toe te voegen. Het toevoegen van een element gebeurt dan weer in een constante tijd. Het bepalen van welke boodschap eerst behandeld zal wordt gebeurt door steeds met de hoogste queue te beginnen en dan te zakken naar die met lagere prioriteit, en regelmatig weer eens de hogere prioriteit queues na te kijken. Figuur 4.2 toont de evolutie van de router queue.



Figuur 4.2: De evolutie van de Borg router queue

We zitten dan nog altijd met het probleem van de oude boodschappen met een lagere prioriteit die nog in een queue zitten te wachten. Daar kan men ook een oplossing voor vinden. We kunnen bijvoorbeeld, wanneer een nieuwe bericht binnenkomt, elke queue met een prioriteit lager dan die van de nieuwe boodschap doorzoeken naar voorgangers van die boodschap, maar het kan ook beter. Als we terug kijken naar de sectie over de originele Borg router, dan zien we dat elke router een structuur bijhoudt met informatie voor elke agent die hij kent. We kunnen hiervan gebruik maken door die name structuur uit te breiden met een extra veld. Deze zal de laagste prioriteit bijhouden van een boodschap die afkomstig is van die agent en die in zijn queue aanwezig is. Met andere woorden, wanneer een boodschap aankomt wordt er naar de prioriteit ervan gekeken. Indien deze lager is dan de waarde die de router in zijn geheugen heeft voor de zender, dan gaat de waarde van de boodschap de nieuwe referentie in de router worden. Indien het gelijk is gebeurt er niets. Indien de nieuwe waarde hoger is, dan gaan de queues vanaf diegene overeenkomstig met de oude waarde tot diegene overeenkomstig met één waarde minder dan de prioriteit van het nieuwe bericht doorzocht worden op boodschappen die eerder gestuurd werden. Die boodschappen zullen dan naar de juiste queue verplaatst worden en de referentie waarde zal dan vervangen worden door de nieuwe prioriteit. Om dit te verduidelijken komt er een kort voorbeeld.

We beginnen dus met een eerste boodschap afkomstig van agent Alice. Deze boodschap komt in de router af en heeft een prioriteit 3, de standaard prioriteit. De router zal dan als referentie waarde voor die agent 3 bijhouden. Alice stuurt dan een reeks nieuwe boodschappen met prioriteit 3. Niets specials gebeurt vermits dit dezelfde waarde is als de referentie waarde. De boodschappen worden in een constante tijd aan de queue 3 toegevoegd en wachten op hun beurt om behandeld te worden. Dan stuurt Alice een boodschap met lage prioriteit, namelijk 2. De router zal dan zijn referentie verlagen naar 2 en de boodschap zal in queue 2 geplaatst worden. Tot nu toe gebeurt alles dus in een constante tijd. Als Alice nu een boodschap met prioriteit 5 stuurt, dan zal de router merken dat dit hoger is dan de referentie waarde die op 2 staat. Op dit moment gaat de router zijn queues 2, 3 en 4 doorlopen om te zien of er nog oude berichten afkomstig van Alice in aanwezig zijn. Het is niet nodig om queue 1 na te kijken vermits er daar geen boodschappen van Alice kunnen zijn, anders zou de referentie waarde op 1 staan. Wanneer de queues afgelopen zijn worden alle oude boodschappen die gevonden werden op queue 5 gezet. De referentie waarde zal dan op 5 gezet worden. Als de volgende boodschap van Alice weer een prioriteit 5 heeft gaat die boodschap gewoon op de queue geplaatst worden, allemaal weer in een constante tijd. Met die oplossing gaan de queues nog altijd afgelopen worden om op zoek te gaan naar oudere berichten, maar dit zal enkel gebeuren wanneer een boodschap getuurd worden met een hogere prioriteit dan de vorige berichten, en niet bij elke boodschap. Als een agent steeds boodschappen stuurt van dezelfde prioriteit, dan zal het toevoegen van de boodschappen in de verschillende routers die het op zijn weg tegenkomt altijd op een constante tijd gebeuren.

4.3.2 Congestion Control

Met het toevoegen van een priority queue zijn onze problemen nog niet opgelost. Een router kan maar een eindig aantal berichten tegelijk in zijn queue bevatten. Als de queue gedurende een bepaalde tijdsduur meer berichten binnenkrijgt dan dat het er kan verwerken, dan zal de queue blijven groter worden. Dit veroorzaakt meer en meer vertragingen en kan er zelf zorgen dat de router het na een bepaalde tijd laat afweten, wanneer het geheugen voorbehouden voor de router vol geraakt. Ook kan een grote stroom van hoge prioriteit boodschappen het goede verloop van de andere berichten, met lagere prioriteit, tegenwerken. Daarom is er congestie controle nodig.

Een eerste manier om aan congestie controle te doen is werken met een bovengrens voor het aantal elementen in de queue. Wanneer de limiet bereikt is, worden nieuwe berichten niet meer in de queue toegelaten en worden er slechts weer boodschappen aanvaard wanneer het aantal aanwezige elementen in de queue gedaald is. Het verwijderen van boodschappen zal opgevangen worden door het oorspronkelijk systeem van acknowledgements dat de Borg router bevat. Wanneer een router na een zekere tijd geen bericht terug krijgt dat zijn oorspronkelijke zending is aangekomen, dan stuur hij deze opnieuw op.

Deze manier van werken heeft als nadeel dat het niet fair is. We nemen een voorbeeld om dit te illustreren. We hebben een agent Alice en een agent Bob. Agent Alice stuurt aan een hoog ritme boodschappen door het netwerk terwijl agent Bob slechts heel af een toe een boodschap stuurt. Met de congestie controle die tot nu toe beschreven is zullen bij het vollopen van een router geen enkele boodschappen meer aanvaard worden. Maar wat we weten is dat het vollopen van de queue de fout is van agent Alice. Toch zullen de berichten van agent Bob tegengehouden worden, terwijl deze slechts heel weinig in aantal zijn en niets te maken hebben met het vollopen van het netwerk.

We kunnen de router uitbreiden door de referenties die het van de agents bewaart in de naam laag aan te vullen met een waarde die het aantal berichten afkomstig van die agent en aanwezig in de router queue bijhoudt, net zoals er daar al de laagste prioriteit wordt bijgehouden. De router blijft nog altijd een maximum waarde voor het aantal aanwezige elementen hebben, maar deze ligt nu lager. Wanneer deze keer de waarde overschreden wordt, dan zal er gekeken worden wie de zender is van een nieuwe boodschap. Indien deze al zijn quota boodschappen overschreden heeft, dan wordt het nieuwe bericht niet aanvaard. Indien het niet het geval is kan de boodschap gewoon aan de queue toegevoegd worden. Met deze manier van werken heeft elke agent recht op een aantal boodschappen. De reden waarom de globale waarde voor het aantal boodschappen in de router nog gehouden wordt is omdat een agent op die manier toch een hele hoop boodschappen kan sturen, zelf als het over zijn quota gaat. Zolang het geen congestie veroorzaakt zijn nieuwe berichten toegelaten. Als men enkel met de individuele waarden zou werken, dan zouden er een heleboel boodschappen geweigerd kunnen worden wanneer een agent alleen bezig is en dat de router toch de capaciteit heeft om de stroom te behandelen. Met deze oplossing zal er altijd gestreefd worden naar een optimaal gebruik van de router en er is toch een eerlijke verdeling van het verlies van de boodschappen wanneer congestie voorkomt.

Bij de Quality of Service router voor Borg werd dit zelf nog een stap verder gezet. Het aantal toegelaten boodschappen in een queue is hier ook afhankelijk van de prioriteit van de boodschappen. Er werd gekozen om de prioriteit te nemen die ook gebruikt wordt voor het bepalen in welke queue een boodschap moet komen. Een andere oplossing had geweest om een tweede waarde te gebruiken die de gebruiker apart kan instellen. Een nadeel is dan dat er gekozen kan worden voor een hoge prioriteit en een groot aantal toegelaten boodschappen. Met de actuele oplossing hangen de twee samen. Een agent die berichten met een hoge prioriteit stuurt heeft recht op slechts een beperkt aantal elementen tegelijkertijd in de queue. Hoe lager de prioriteit, hoe meer boodschappen toegelaten worden. Zo moet er gekozen worden tussen snel maar met weinig boodschappen of traag en veel boodschappen. Dit heeft als gevolg dat een stroom van hoge prioriteit berichten de lagere prioriteiten minder zullen tegenwerken.

Een extra uitbreiding werd hieraan nog toegevoegd. Het heeft te maken met de extremen in de prioriteiten, namelijk 1 en 5. Bij 1 is het zo dat het aantal berichten onbeperkt is. Er mogen dus zo veel berichten in de queue aanwezig zijn als toegelaten door de hoeveelheid geheugen. De bedoeling hiervan is dat een bericht van prioriteit 1 nooit verwijdert wordt, dus dat het zeker aankomt. Op die manier worden vertragingen door het opnieuw te moeten zenden van het bericht vermeden. Dit is dus een gepaste prioriteit voor een bericht dat absoluut moet aankomen. De aanpassing voor 5 is juist het omgekeerde. Die prioriteit dient voor berichten die mogen aankomen, maar waar het geen drama is als het niet bij zijn bestemming toekomt. Het is namelijk zo dat bij deze prioriteit de acknowledgements wegvallen. Wanneer een bericht van prioriteit 5 gestuurd wordt, dan verwacht de zender geen boodschap terug dat hem verwittigt dat het goed aangekomen is. Het kan dus goed mogelijk zijn dat een bericht onderweg kwijtgespeeld werd. Dit is geschrikt voor stromen van data waar niet elke frame moet aankomen.

4.3.3 In de praktijk

Verschillende praktische onderdelen moeten nog even besproken worden. Ten eerste, hoe kan men de toegevoegde eigenschappen bij het programmeren van een Borg applicatie oproepen. Hiervoor zijn verschillende oplossingen mogelijk. Een eerste had geweest om bij elke boodschap dat gestuurd moet worden in de Borg code een extra argument toe te voegen dat de prioriteit van de boodschap moet meedelen. Een eerste nadeel hiervan is dat, als we bestaande code willen aanpassen aan de Quality of Service eigenschappen, dat elke functie die staat voor het opsturen van een bericht opgezocht en aangepast moet worden. Een tweede nadeel is dat wanneer we gewone Borg code die geen Quality of Service wil gebruiken op een met Quality of Service uitgebreide router wil laten draaien, dat we niet aan alle oproepen een prioriteit willen toevoegen. De oorspronkelijke code, zonder extra argument, moet dus ook nog blijven werken. Omgekeerd moet de gewone router ook nog aangepast worden om met het extra argument te kunnen werken, wanneer het er oorspronkelijk niet voor gemaakt is. Een laatste nadeel is dat de meeste boodschappen tussen agenten getuurd worden door middel van remote message sends. Het aantal argumenten die daaraan meegegeven worden komen overeen met het aantal argumenten die de functie verwacht. Als we daaraan nog een extra argument moeten toevoegen wordt het heel onduidelijk om te volgen wanneer men wel of niet de prioriteit wil meegeven, vooral dat de evaluator nooit echt weet hoeveel argumenten een functie juist verwacht vermits die mogelijk op een andere virtuele machine zit.

De manier van werken waarvoor gekozen werd is het gebruiken van een vorm van globale variabele, die de prioriteit voorstelt. Door middel van een native oproep kan deze veranderd worden. De waarde is eigen aan elke agent en wordt dan gebruikt als prioriteit voor de berichten die van deze agent vertrekken. De native die hiervoor gebruikt wordt is:

`messagepriority(newPriority)`

Het aanpassen van oude code gebeurt door het toevoegen van die oproep op de geschikte plaats. Indien gewone code gebruikt wordt is de oproep niet nodig. De prioriteit van de berichten zullen dan standaard op 3 gezet worden. Op een gewone router zal die oproep de waarde false teruggeven, terwijl het op de met Quality of Service uitgebreide router true weergeeft. Hiermee kan een agent tijdens zijn uitvoer bepalen op welke soort router hij aan het werken is.

Verder is het de bedoeling is dat de twee verschillende types router zonder probleem met elkaar kunnen blijven werken. Daarom kreeg het commando ADT uit de originele router ook een extra veld voor de prioriteit. Deze wordt gewoon niet gebruikt. Alle boodschappen vertrekken met een waarde 3 in dat veld, zelf indien de Borg source code een native oproep bevatte die dat had moeten veranderen. Als er een boodschap binnen komt wordt er ook niet gelet op de prioriteit van deze. Op die manier gebeurt de communicatie feilloos tussen de verschillende soorten Borg routers aanwezig in het netwerk.

4.4 Aspecten van Quality of Service in Borg

In het hoofdstuk over Quality of service werden de verschillende elementen uitgelegd die een invloed hadden op de kwaliteit van een verbinding. We kunnen deze ook naar Borg overbrengen en bestuderen hoe Borg met Quality of Service presteert.

Wat we eerst al kunnen zeggen is welke level van Quality of Service Borg aanbied. Binnenin het Borg netwerk is dit een Differentiated Service. We werken via prioriteiten en niet via reservatie doorheen het netwerk van een bepaalde capaciteit waarop een applicatie recht heeft. Een harde garantie van de kwaliteit is er ook niet. Dit is nog eens versterkt door de feit dat het Borg netwerk draait bovenop een Best Effort netwerk, namelijk het klassieke TCP/IP netwerk.

Verder worden in deze sectie de verschillende dimensies van Quality of Service [1] behandeld. Bij elk apart komt er uitleg van hoe ze bij Borg overgenomen worden en wat de Quality of Service eventueel als plus bijgebracht heeft voor dit onderdeel.

4.4.1 Beschikbaarheid

Beschikbaarheid is de tijd dat het netwerk actief is op de momenten dat de gebruiker het nodig heeft. In klassieke netwerken is het streefdoel om een gemiddelde van 99.999% uptime te hebben [10] , wat overeenkomt met iets meer dan vijf minuten down per jaar.

Dit is in Borg niet echt realiseerbaar. Zoals al eerder vermeld is Borg een experimentele taal die gebruikt wordt voor onderzoek doeleinden. Het is dus niet een netwerk dat de hele tijd actief blijft. Desalniettemin is het een goed idee om eens te kijken wat de Quality of Service eigenschappen aan de beschikbaarheid kunnen veranderen.

De manier om een degelijke beschikbaarheid te bezorgen is om de betrouwbaarheid van de elementen van het netwerk te verbeteren. Bij Borg kunnen we weinig doen aan de hardware waarop men aan het werken is en moet er dus vooral gewerkt worden om de software te verbeteren. De stabiliteit van Borg wordt dus van groot belang.

Hier kunnen we zeggen dat de met Quality of Service uitgebreide router een verschil kan uit maakt. Dankzij de congestie controle gaat een router proberen te voorkomen dat zijn geheugen vol geraakt met berichten. Dit zou de stabiliteit van de router moeten verbeteren. Het is namelijk zo dat wanneer een klassieke Borg router te veel boodschappen in een keer moet behandelen, dat dit leiden tot het falen van de router. Dan moet deze weer opnieuw opgestart worden, wat de beschikbaarheid vermindert. Het verschil dat Quality of Service biedt is misschien niet enorm, maar het maakt wel een verschil uit om te weten dat wanneer er te veel boodschappen naar een router gestuurd worden, deze dat kan behandelen en toch wel verder zal kunnen werken zonder te falen.

4.4.2 Overdracht

De overdracht is de hoeveel data dat door het netwerk gestuurd werd gedurende een bepaalde periode. In Borg spreekt men niet echt van hoeveelheden data, waar men kan wel een transitie maken tussen de hoeveelheid data dat doorgezonden wordt en het aantal berichten dat gestuurd werden in Borg. Zo kan men dan de overdracht voor een Borg agent laten overeenstemmen met het aantal berichten dat de agent gestuurd heeft.

Als we de zaken op die manier bekijken, dan kan er gezegd worden dat het toevoegen van Quality of Service eigenschappen aan de Borg router een verschil kan uitmaken. Dit gebeurt door middel van de congestie controle structuur. Door te melden met welke prioriteit hij wil werken zegt een agent aan de router hoeveel berichten hij minimaal overgedragen wil zien. Wanneer de router de mogelijkheid heeft om meer toe te laten zal deze dit ook doen. Zo kan een agent meer krijgen dan wat hij verwacht. Eens het netwerk veel gebruikt wordt en in een status geraakt dat congestie kan veroorzaken, dan zal het aantal toegelaten berichten verminderen naar hetgeen oorspronkelijk overeen gekomen was met de agent, en dus zal de agent altijd minimum krijgen wat het gevraagd heeft.

4.4.3 Vertraging

De vertraging die te merken is in een netwerk is het verschil dat een stuk data ondervindt tussen twee punten in het netwerk. De vertraging is in Borg het verschil tussen het moment waarop de zender een boodschap opstuurt en het moment waar die boodschap bij zijn bestemming aankomt. De vertraging kan afkomstig zijn van van twee lagen. De eerste laag is die van het netwerk waarop Borg draait, dus de TCP/IP laag. Zoals alle trafiek zullen boodschappen gestuurd in Borg ook vertragingen kennen op dit niveau. De tweede laag waar vertragingen kunnen opduiken is die van de Borg router. Een boodschap kan in die router ook blijven wachten op zijn beurt. Voor het eerste soort vertraging kan er in Borg weinig worden gedaan, maar op de tweede kan er wel gewerkt worden.

Dit is dan ook wat er gebeurt in de Quality of Service router. De vertragingen opgelopen door een boodschap zullen verschillend zijn naar gelang de agent die de boodschap stuurde een hoge of een lage prioriteit aangevraagd heeft. Het verschil in vertraging zal slechts klein zijn wanneer het Borg netwerk niet zwaar gebruikt wordt, maar het verschil wordt groter wanneer meer een meer boodschappen gestuurd worden en dat de router queues beginnen groter te worden. In dat geval zal een bericht met hoge prioriteit minder vertraging oplopen dan een bericht met lage prioriteit.

4.4.4 Vertragings Variaties

De vertragings variaties, of jitter [3], is het verschil in vertraging opgedaan door op elkaar volgende pakketten. Voor Borg hoeven we de pakketten gewoon te vervangen door berichten en de definitie blijft geldig.

Jitter komt dus voor wanneer een boodschap voor zijn voorganger aankomt. Vermits de volgorde van aankomst dan verschillend is van die van het opsturen kan dit voor problemen zorgen. Dit probleem werd al besproken in het deel over de priority queue, en de oplossing hiervoor ook.

4.4.5 Verlies

Wanneer we over data verlies praten in Borg dan gaat het enkel over het verlies van berichten onderweg of in een router. In de oorspronkelijke router wordt er geen data verlies veroorzaakt. Daar wordt er nooit een bericht verwijdert uit de queue bij congestie. Maar toch kunnen er af en toe een paar berichten verdwijnen. Dit is het geval wanneer bijvoorbeeld een router wegvalt en dat deze nog berichten in zijn queue had. Er zijn ook nog factoren die extern zijn aan het Borg netwerk dat kunnen zorgen voor verlies van data.

Met de toegevoegde congestie controle kan het verlies van boodschappen wel veroorzaakt worden door de Borg router zelf. In beide versies van de router zal het verlies van een boodschap als enige gevolg hebben dat er wat vertragingen gebeuren, vermits een niet aangekomen boodschap na een bepaalde tijd opnieuw opgestuurd wordt. Dit is enkel niet het geval wanneer men met prioriteiten werkt en als de hoogste prioriteit gekozen wordt. Daar worden de berichten niet meer opnieuw gestuurd, maar dat is iets waar de programmeur van de agent rekening mee moet houden.

4.5 Conclusie

Voor deze verhandeling moet er onderzocht worden of Quality of Service toevoegen aan een systeem zoals Borg een verschil uitmaakt. Hiervoor moest de Borg router eerst uitgebreid worden. Daarom moet de werking van de originele router goed duidelijk zijn. Dan kan er gezocht worden naar wat we eraan kunnen toevoegen zodat de router Quality of Service eigenschappen zou vertonen. Dit is wat we in het eerste deel van dit hoofdstuk besproken hebben.

In het tweede deel werden de verschillende aanpassingen aan de router uitgelegd. Deze zijn voornamelijk het toevoegen van congestie controle en het gebruiken van een priority queue voor het bewaren van de boodschappen in de router.

Het hoofdstuk eindigt door een kijkje te nemen naar de verschillende dimensies van Quality of Service. Wat ons dan vooral interesseert is hoe deze in Borg bekeken kunnen worden. We eindigen dan door te zeggen welke toegevoegde waarde de aangepaste router aan die dimensies kan brengen.

Hoofdstuk 5

Experimenten

5.1 Inleiding

In het vorig hoofdstuk werd de Quality of Service router voor Borg in grote lijnen uitgelegd. Een paar theoretische conclusies kwamen uit de analyse van de verschillende toegevoegde onderdelen. Deze eerste conclusies moeten nu nog in de praktijk nagekeken worden. Dit is wat er in dit hoofdstuk gebeurt. Een paar testprogramma's werden in Borg ontwikkeld waarmee de Quality of Service mogelijkheden aan de tand gevoeld kunnen worden.

Het eerste programma dat gemaakt werd is de Wall Street applicatie. Het is een vorm van beurs applicatie waar een gebruiker aandelen kan kopen en verkopen terwijl hij aan de lopende band informatie krijg over de verschillende aandelen die door het systeem beheerd worden.

Als tweede programma wordt de Monitoring applicatie gebruikt. In deze case wordt een sensor geobserveerd door twee agents, één die constant de laatste sensorwaarde weergeeft en één die alle verschillen van de waarde bewaart. De bedoeling van de laatste is om te kunnen nagaan op welk tijdstip een bepaalde verandering is gebeurt.

Daarna komen nog een paar applicaties aan bod die nog in andere onderzoek gebruikt worden en die ook de Quality of Service eigenschappen van de router kunnen gebruiken. Deze zijn een Mail Client, een Router Planner en een Meetingplanner.

Het hoofdstuk wordt beëindigd met de uitleg van de experimenten die gemaakt worden. Hun context wordt uitgelegd samen met wat men van het experiment verwacht. Dit wordt dan aangevuld met de resultaten die uit de experimenten volgden en de uitleg van hoe men aan die resultaten kon geraken.

5.2 Algemeen

De bedoeling van dit hoofdstuk is om Quality of Service uit te testen. Dit gebeurt niet met eender de welke programma. Eerst moet er bepaald worden welke soort applicaties men in Borg zou kunnen gebruiken die Quality of Service zouden kunnen gebruiken.

Er zijn twee manieren om die applicaties te vinden. De eerste is door typische Quality of Service voorbeelden te nemen en deze naar Borg over te brengen. De tweede manier is om Borg applicaties te nemen en ze uit te breiden met het gebruik van Quality of Service.

Het volgen van de eerste manier levert spijtig genoeg weinig op. Wat we veel terugvinden in een onderzoek naar Quality of Service zijn applicaties die veel bandbreedte nodig hebben. We vinden voornamelijk interactieve media applicaties zoals video en voice over IP. Het zijn dus programma's die een constante stroom van beelden en geluid produceren die over het netwerk heen moeten. Dit is iets waar we weinig mee kunnen doen in dit onderzoek. Borg bevat namelijk geen eigenschappen die deze soort applicaties zou kunnen ondersteunen. Het is ook niet de bedoeling van Borg om applicaties te maken die beeld en geluid weergeven. Daarbij is het niet zeker dat een Mobile Multi Agent Systeem zoals Borg echt de meest geschikte omgeving is om zulke applicaties in te ontwikkelen. Het zijn applicaties waar de data snel moet weergegeven worden. De beste manier is om te werken met een directe connectie tussen zender en ontvanger zodat de stroom van gegevens het best doorkomt. Borg daarentegen werkt met boodschappen, en het ondersteunt geen vaste verbinding tussen twee agents. De stroom van gegevens zou dus eerst nog in aparte boodschappen opgesplitst moeten worden en verder weer in elkaar gestoken worden. Ook kunnen deze applicaties niet echt gebruik maken van de voordelen die Borg biedt, zoals de mobiliteit van de code. Er moeten dus best andere soorten programma's gevonden worden die nood hebben aan Quality of Service.

We moeten dus eerder de tweede manier van werken volgen, namelijk het vertrekken van applicaties die in Borg een natuurlijke omgeving vinden en die te voorzien met Quality of Service eigenschappen. We kunnen daarvoor ook een eerste eigenschap voor deze programma's opmerken uit het zoeken naar applicaties voor Quality of Service, en dat is dat we programma's moeten hebben die veel trafiek veroorzaken in het systeem. Als we applicaties zouden gebruiken die slechts weinig netwerk verkeer veroorzaken, dan zouden de routers niet intensief gebruikt worden en zou men slechts moeilijk kunnen opmerken of Quality of Service toevoegen wel degelijk een verschil uitmaakt.

Dit wil niet zeggen dat we niets zijn met applicaties die weinig verkeer veroorzaken. We kunnen er ook gebruik van maken door ze te laten werken in een systeem waar andere applicaties bezig zijn en waar er in het algemeen veel berichten door het netwerk moeten, die daarvoor niet allemaal afkomstig moeten zijn van dezelfde applicaties. De boodschappen die van een applicaties komen die weinig trafiek veroorzaakt moeten misschien ook aan bepaalde kwaliteiten voldoen, en de feit dat het netwerk toch intensief in gebruik is kan Quality of Service ook een voordeel worden voor die applicaties.

De beste manier om de router intensief te laten werken blijft toch om applicaties te maken die veel boodschappen doorsturen. Het is met die gedacht in het hoofd dat de Wall Street en de Monitoring applicaties ontworpen werden. Ze gaan in de eerste plaats zorgen voor het aanmaken van veel netwerk trafiek en gaan dan daarmee in parallel een reeks boodschappen zenden die niet tot een grote stroom behoren, zodat we verschillende types boodschappen in het netwerk terugvinden. De andere applicaties die nog besproken zullen worden gaan dan uitgetest worden in een netwerk waar de twee eerste programma's al actief zullen zijn.

5.3 Wall Street

In deze sectie wordt er uitgelegd wat de Wall Street applicatie is, hoe het in zijn werk gaat en wat het doel ervan is. De verwachtingen van de resultaten van het experiment met deze applicatie zijn de vinden in de sectie 5.6.1. De resultaten zelf zijn te vinden in sectie 5.6.2.

5.3.1 Principe

De Wall Street case is een vorm van beurs applicatie. Het is geïnspireerd van de televisie, waar we sommige kanalen hebben die onderaan het scherm een band laten lopen die de recentste waarden van bepaalde stock opties weergeeft.

Qua architectuur is dit een Client/Server applicatie. Er draait een server op een welbepaalde computer, waar clients op kunnen inloggen. Eens ze dat gedaan hebben beginnen ze een stroom van boodschappen te krijgen die hen de recentste waarden van de opties die de server beheert weergeeft. De client kan ook specifiek vragen achter de waarde van een wel bepaalde optie. De client heeft dan ook de mogelijkheid om aan de server te vragen welke zijn eigen status is. Dit betekent dat hij gaat nakijken welke opties hij zelf bezit en in welke hoeveelheden. Er zijn dan ook operaties mogelijk die een client toelaat om nieuwe opties te kopen of om zijn eigen bezittingen weer te verkopen.

Het gebruik van Borg past goed bij deze applicatie voor verschillende reden. De server, die normaal op een vaste plaatst moet staan zodat de clients hem terug kunnen vinden, hoeft in Borg niet echt altijd op die wel gedefinieerde plaats aanwezig te zijn. Het enige wat hoeft is dat de server daar opgestart wordt, en dat het een agent aanmaakt die de correcte naam draagt. De clients werken dan met de naam van de server, en niet zijn lokatie, wat de server toelaat om te migreren naar andere computers. Borg zal er voor zorgen dat de server zijn berichten blijft krijgen.

Een andere reden die van Borg een geschikte omgeving maakt is dat de client zelf ook van plaats kan veranderen. Wanneer een client met de server wil communiceren, dan moet de client gewoon een referentie naar zijn agent geven zodat de server verder met hem kan werken. De server moet dus niet weten welke de locatie van de client is. Indien de client dan van computer verandert moet de server daar niet van op de hoogte gebracht worden. Het Borg systeem zal er automatisch voor zorgen dat de boodschappen voor de client van zelf naar de nieuwe locatie gaan.

5.3.2 Werking

We gaan nu verder de meer technische aspecten van de Wall Street applicaties uitdiepen. We beginnen hiervoor met de client. Dit is het meest eenvoudige deel van het systeem. De client moet niet veel bijhouden. Het heeft gewoon een referentie naar de agent waar de server op aangemaakt werd en het houdt de gebruikersnaam in het systeem van de persoon die de client gebruikt. De eerste functie die gedefinieerd wordt is het inloggen op de server. Daar geeft de client de gebruikersnaam en een referentie naar zichzelf weer. Dan komt er een bijhorende functie logout, die de server verwittigt dat de client niet meer in gebruik is. Er zijn dan een paar functies die toelaten om de status van een gebruiker op te vragen en aan te passen. Het opvragen van de status kan ook gebeuren voor een optie. Bij deze operaties is er altijd ook een callback functie zodat de server de resultaten steeds kan meedelen. Een laatste functie die men in de client terugvindt is diegene die zorgt voor het opvangen van de stroom van gegevens die door de server doorgestuurd wordt. Deze kan dan op verschillende manieren aan de gebruiker gepresenteerd worden. Dit kan gebeuren via een printout naar de command line, maar het kan evengoed grafisch gebeuren. Momenteel worden de gegevens opgeslagen in een tabel.

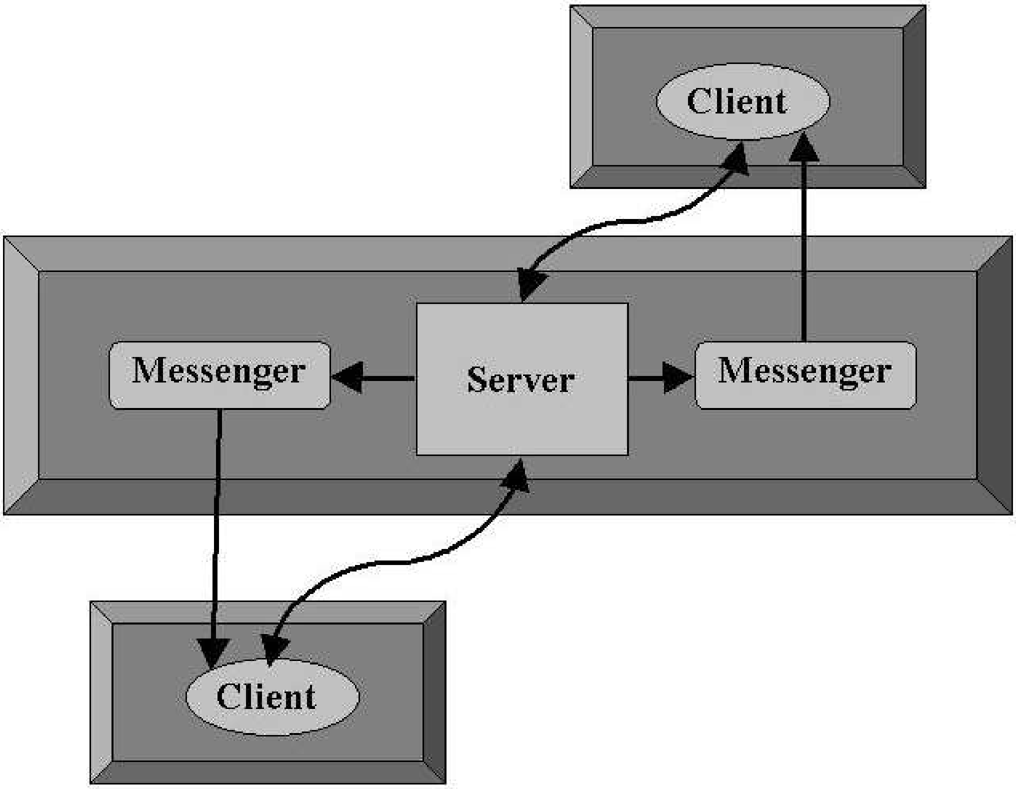
Het tweede deel van de Wall Street case is de server. Deze is opgesplitst in verschillende delen. Het eerste deel in een datastructuur, de voidtable. Deze laat toe om op een eenvoudige manier elementen toe te voegen, te verwijderen, te vergelijken, zonder rekening te moeten houden met de vorm van de tabel.

Verder zijn er in dit deel van de server nog de objecten gedefinieerd die de verschillende soorten elementen van een voidtable voorstellen. Deze zijn nodig omdat de server verschillende voidtables bijhoudt voor verschillende soorten gegevens. Een eerste tabel houdt bij welke gebruikers ingelogd zijn in het systeem en welke de agent is waarop ze actief zijn. Een tweede tabel houdt bij wat de opties zijn die de server beheert en wat hun waarde is. Een derde soort table is diegene die de bezettingen van de gebruikers bevat. Deze werkt intern nog eens met een voidtable voor elke gebruiker. Deze laatste tabel heeft een element voor elke soort bezitting en houdt de hoeveelheid dat de gebruiker ervan bezit bij.

Tot zover het eerste deel van de server. De rest van de server wordt opgesplitst over twee agents. Deze zijn de actuele server en een messenger deel. De server is de agent waarmee de client zal communiceren. Alle vragen die een client moet stellen komen terecht bij de server en deze laatste zal de vragen ook beantwoorden. Het is juist het stellen van de vragen aan de server die het messenger gedeelte nodig maken. Het sturen van de stroom van gegevens is een thread die de server continu bezig houdt. De responstijd voor de vragen van de clients zou dan sterk vergroten. Daarom laat de server het sturen van de stroom door de messenger uitvoeren. Wanneer een client zich bij de server aanmeld zal deze laatste een nieuwe messenger agent aanmaken voor de client. Die messenger krijgt een referentie naar de client en de gebruikersnaam mee. Het zal dan de gegevens die het moet doorsturen aan de server opvragen en de stroom starten. Het zal regelmatig aan de server vragen of zijn gebruiker nog actief is en een nieuwe versie van de data meenemen. Wanneer de gebruiker van de messenger niet meer actief is zal de messenger agent van deze client verwijderd worden.

De structuur van de messenger bestaan uit functies die zorgen voor het sturen van de stroom van gegevens en van communicatie functies met de server, om een update van de gegevens te vragen. De structuur van de server zelf is gemaakt van functies die de vragen van de client beantwoorden en de nodige informatie aan de messenger teruggeven.

In figuur 5.1 vinden we de structuur van de applicatie in beeld terug. De donkere vierhoeken met dubbele randen stellen verschillende computers voor. De lichtere objecten met een naam stellen de verschillende agents voor. De bidirectionele pijlen stellen een gewone communicatie voor tussen de agents, terwijl de unidirectionele pijlen een stroom van informatie voorstellen. De figuur stelt een voorbeeld voor waar twee clients met de server geconnecteerd zijn.



Figuur 5.1: De structuur van de Wall Street applicatie

5.3.3 Doel

Wat er met deze case voornamelijk bereikt wil worden is aan te tonen dat er een verschil bestaat tussen de gewone en de uitgebreide router. Daarom zijn er twee versies van de applicatie. Een maakt gebruik van de Quality of Service native en draait op de uitgebreide router. De ander probeert even goed te presteren op de originele router. Op die manier proberen we te zien of er wel degelijk een verschil bestaat tussen de twee routers.

5.4 Monitoring

In deze sectie wordt er uitgelegd wat de Monitoring applicatie is, hoe het in zijn werk gaat en wat het doel ervan is. De verwachtingen van de resultaten van het experiment met deze applicatie zijn de vinden in de sectie 5.6.1. De resultaten zelf zijn te vinden in sectie 5.6.2.

5.4.1 Principe

Als tweede case wordt een monitoring applicatie gebruikt. De bedoeling hiervan is om de waarde van een sensor in het oog te houden. De applicatie bestaat uit drie belangrijke componenten:

- een sensor
- een monitor
- een tracker

De sensor is het deel die de waarde weergeeft van een bepaalde bron. Het kan een lichtsensor zijn die weergeeft hoeveel licht er in een ruimte aanwezig is, of een warmtesensor die de temperatuur van een bepaald onderdeel uit een systeem meet. Wat de sensor juist meet is niet echt van groot belang, het is gewoon een stuk programma dat een waarde levert.

Het tweede deel, de monitor, is het onderdeel dat instaat voor het weergeven van de waarde van de sensor aan de gebruiker van het systeem. Het is bijvoorbeeld een display waar de meest recente waarde op afgebeeld staat. De bedoeling van de monitor is om de gebruiker in real time de staat van de sensor te laten kennen met zo weinig mogelijk vertraging.

Het laatste onderdeel, de tracker, is een deel dat gebruikt wordt om alle veranderingen van waarde van de sensor bij te houden. Alle waarden die de sensor aangenomen heeft komen daar terecht en kunnen weer opgehaald worden. De bedoeling van de tracker is om het tijdstip op te sporen wanneer er bij de sensor een verandering is gebeurt.

In dit systeem gaat de sensor zijn waarde steeds sturen naar de monitor en de tracker. Een voorbeeld is die van een warmte detector in een kernreactor. De temperatuur wordt steeds nagekeken en wanneer een probleem voorvalt kunnen de gebruikers van het systeem altijd terugvinden waar het is beginnen mislopen en wat er juist gebeurt is met de waarde van de sensor.

Borg wordt voor deze applicatie gebruikt omdat het de communicatie tussen de verschillende onderdelen vergemakkelijkt. De sensor moet niet op voorhand weten wie zijn monitor en tracker worden. Bij het opstarten moeten een monitor en een tracker zich gewoon aanmelden bij de sensor en de communicatie gebeurt voor de rest gewoon door boodschappen op te sturen. De mobiliteit van de Borg agents kan ook gebruikt worden. Als de persoon die achter de monitor staat voor een of andere reden van plaats moet veranderen, dan kan de monitor doorgestuurd worden naar een andere computer om door iemand anders nagekeken te worden, of zelf gewoon naar een palm computer die de gebruiker altijd bij zich kan houden. Ook de tracker kan van computer verandert worden, wanneer deze computer bijvoorbeeld problemen begint te vertonen en dat met dan de tracker van plaats verandert zodat het toch kan blijven verder werken.

5.4.2 Werking

Technisch gesproken vinden we drie onderdelen terug in deze applicatie die op onafhankelijke manieren geprogrammeerd kunnen worden. Het enige wat overeen moet stemmen is de registratie van een tracker of monitor bij een sensor en de functie die voor het sturen van de sensor waarden moeten instaan.

Het eerste deel dat besproken wordt is de tracker. Deze bestaat uit een enkele agent. Intern bevat deze een voidtable, zoals bij de Wall Street applicatie. Wat we nog in de tracker vinden zijn functies om te registreren en te deregistreren bij een sensor en om de waarde die de sensor doorstuurt binnen te krijgen. Een laatste deel van de tracker is gemaakt uit functies die toelaten om de voidtable na te gaan en kijken wat die allemaal als informatie bevat.

De monitor is niet veel verschillend van de tracker. Het bevat ook de registratie functies met de sensor en de functies om de waarden binnen te krijgen. Het verschil ligt in wat er met de waarde gebeurt. Hier gaat de waarde niet in een tabel bijgehouden worden maar aan de gebruiker van het systeem weergegeven worden. Dit weergeven kan op verschillende manieren gebeuren, grafisch of tekstueel. Dit kan afhangen van hoe de nieuwe weergegeven waarde bekomen wordt. Als elke nieuwe waarde die binnenkomt getoond wordt, dan is het misschien beter om met een grafisch venster te werken waar de nieuwe waarde altijd de oude zal overschrijven. Bij een tekstuele uitvoer gaat dit zorgen voor een grote stroom van getallen die mogelijk altijd dezelfde zijn. Als we daarentegen de waarde die getoond wordt enkel vernieuwen wanneer de binnengekomen waarde een verschil toont met de oude waarde, dan kan een tekstuele oplossing gebruikt worden. Dit is ook wat in de gebruikte versie van de monitor gebeurt.

Het laatste deel van de applicatie is de sensor zelf. Deze is verdeeld over twee agents. De eerste agent bevat de code voor de sensor zelf. De tweede is een dummy agent. De dummy zorgt voor het verbreken van de thread van de sensor, zodat het niet altijd met dezelfde functie bezig is. Zo krijgt de sensor de tijd om nieuwe boodschappen, afkomstig van een andere agent, ook te behandelen en niet in een loop te zitten waar het de sensorwaarde zit te versturen. De sensor zelf bevat code om een tracker en monitor te registreren. Wanneer beide aangemeld zijn gaat de sensor beginnen zijn waarde op te halen en zal deze doorsturen naar de twee andere agents. Dan geeft het de controle over aan de dummy agent. Deze zal de controle teruggeven aan de sensor die dan een nieuwe waarde zal lezen en verder doorsturen naar de tracker ene monitor. Het gebruik van de dummy dient vooral om de loop van de sensor te onderbreken, zodat deze kan merken wanneer een van de andere agents niet meer geregistreerd willen zijn. De dummy kan verder ook gebruikt worden om de interval te bepalen tussen het lezen van twee opeenvolgende waarden van de sensor. Het lezen van opeenvolgende waarden kan snel gebeuren, maar met de dummy agent kan er dus gespeeld worden en kunnen de intervallen voor het lezen van een nieuwe waarde groter gemaakt worden.

5.4.3 Doel

De bedoeling van deze test case is om na te gaan of het gebruik van verschillende prioriteiten in het systeem een verschil gaat geven of niet. We kunnen twee verschillende noden van Quality of Service opmerken in deze applicatie. Voor de monitor is er nodig dat de nieuwe waarde zo snel mogelijk aankomt zodat de gebruiker steeds op de hoogste is van de nieuwste sensorwaarde. Dit kunnen we identificeren als een prioriteit van waarde 5 als we het systeem uit hoofdstuk 4 volgen. Als er af en toe een waarde niet aankomt is het nog niet een ramp, maar we willen wel zo snel mogelijk op de hoogte zijn van de nieuwe waarde.

Voor de tracker hebben we een andere eis wat de prioriteit betreft. Dat de waarden traag aankomen is geen probleem. De tracker wordt vooral gebruikt om naderhand na te gaan waar de verschillen in sensorwaarden zijn voorgekomen. Hiervoor kunnen we een prioriteit 1 gebruiken, om zeker te zijn dat elke verschillende waarde aankomt bij de tracker.

De verschil in prioriteiten maken we in de sensor, bij het opsturen van de waarden. De waarde zelf die in deze case gebruikt wordt is een dummy waarde.

Om de resultaten van het experiment te kunnen bepalen zijn er nog een paar aanpassingen gebeurt in de monitor. Deze kreeg ook een voidtable om de verschillende waarden bij te houden. Dit dient vooral om de verschillen tussen de berichten gekregen door de tracker en de monitor te kunnen vastleggen. Er kwamen dan een paar functies om de tabellen te kunnen vergelijken.

5.5 Andere applicaties

Ander programma's werden nog gebruikt om het systeem uit te testen. Ze kregen ook lichte aanpassingen om prioriteiten te bevatten en om te testen of de aangebrachte verschillen merkbaar zijn. Die applicaties worden ook gebruikt door andere studenten voor hun eigen onderzoek, voornamelijk door Dimitri Verspecht in zijn onderzoek naar het gebruik van Borg voor Palm computers [28]. In deze sectie worden de verschillende applicatie uitgelegd. De verwachtingen van de resultaten van het experiment met deze applicaties zijn de vinden in de sectie 5.6.1. De resultaten zelf zijn te vinden in sectie 5.6.2.

De applicaties zijn:

- een meetingplanner
- een routeplanner
- een mail client.

5.5.1 De Meetingplanner

De meetingplanner is een systeem dat dient om afspraken te coördineren. Een gebruiker kan zeggen waar en wanneer hij een vergadering wil organiseren en het systeem zal proberen om met zoveel mogelijk andere gebruikers een afspraak te maken zodat ze aanwezig zouden zijn op de vergadering. De gebruiker die de vergadering wil organiseren kan ook meegeven hoeveel percent van de andere gebruikers aanwezig moeten zijn. Als er niet genoeg mensen aanwezig kunnen zijn dan kan de gebruiker beslissen om de vergadering af te lassen. Het systeem zal ervoor zorgen dat dit allemaal zonder complicaties kan gebeuren.

Het programma is gebouwd uit drie verschillende soorten agents. Deze drie zijn:

- MP-Agent
- GUI-Agent
- Meeting-Agent

De MP-Agent stelt een gebruiker of een locatie voor. Van zo een agent gaat de aanvraag komen om een meeting te organiseren. Het bevat ook de gegevens van een gebruiker. Wanneer een samenkomst gepland moet worden dan zal er aan de MP-Agent van een gebruiker gevraagd worden of de gebruiker vrij is op dat moment en indien het zo is zal de MP-Agent die tijdslot invullen met de gegevens van de vergadering. Een locatie heeft ook een MP-Agent zodat er geen twee samenkomsten tegelijk plaatsvinden op dezelfde locatie.

De GUI-Agent is de Grafische User Interface. Dit is het deel waar de gebruiker echt mee werkt. Het vraagt aan de gebruiker welke de gegevens zijn voor de vergadering en stuurt die door naar de MP-Agent die dan voor de organisatie ervan zal zorgen. De GUI-Agent laat aan een gebruiker ook toe om zijn planning na te kijken. De GUI-Agent en de MP-Agent hoeven niet op dezelfde computer aanwezig te zijn. Dit kan nuttig zijn wanneer we het programma willen gebruiken op een computer die weinig geheugen heeft, zoals een Palm.

De laatste soort agent, de Meeting-Agent, stelt een meeting voor. Het wordt aangemaakt door een MP-Agent en zal dan zorgen voor het in orde brengen van de vergadering. Wanneer de vergadering voorbij is of dat deze afgelast wordt zal de Meeting-Agent door de MP-Agent die hem aangemaakt heeft verwijdert worden.

De meetingplanner werd ontworpen door Pieter Verheyden [29]. Het heeft niet echt een nood aan Quality of Service. Het wordt hier gebruikt om te testen hoe een normaal programma zich zal gedragen in een Quality of Service netwerk. Het zal werken samen met andere programma's die wel de QoS eigenschappen gebruiken en die veel trafiek genereren. Er zal dan nagegaan worden of het een verschil maakt als de twee soorten applicaties in parallel gedraaid worden in een systeem met en een zonder Quality of Service eigenschappen.

5.5.2 De Routeplanner

De routeplanner is een applicatie dat gebruikt wordt om een weg te vinden tussen twee locaties met behulp van een kaart. Het werd ontworpen om een vorm van GPS navigatie systeem te maken die gebruikt zou kunnen worden op draagbare computers zoals een Palm. Het principe is dat een gebruiker een startpunt en een bestemming invoert. De applicatie zal dan een database van wegen raadplegen en de kortste weg proberen op te bouwen.

De Borg routeplanner bestaat uit verschillende onderdelen die samen moeten werken en verspreid zijn over verschillende agents. De verschillende soorten agents zijn:

- een Routeplanner-Agent
- een GUI-Agent
- een Monitor-Agent
- een Map-Agent

De Routeplanner-Agent is het deel van het systeem dat een weg gaat proberen vinden. Het krijg een begin en een eindpunt en gaat de kaart doorzoeken naar een verbinding tussen de twee. Wanneer het een weg gevonden heeft gaat het verder blijven zoeken of het geen betere weg kan vinden, want de eerste weg die gevonden wordt in zo een systeem is meestal niet de beste die en bestaat. Door bepaalde technieken te gebruiken kan men het programma dan dwingen om een betere weg te vinden tot het een redelijke weg, of zelf de beste weg, gevonden heeft.

De GUI-Agent is de grafische user interface. Deze zal zorgen voor de communicatie tussen de gebruiker en de Routeplanner-Agent. De GUI zal alle invoer nemen en doorsturen. Het zal de planner dan opstarten. Wanneer de planner een weg gevonden heeft zal die deze dan naar de GUI doorsturen zodat deze die aan de gebruiker kan weergeven.

De Monitor-Agent heeft als taak het observeren van de Routeplanner-Agent. De Monitor-Agent gaat kijken hoe lang de planner zit te werken en hoeveel geheugen hij in beslag neemt. Wanneer het plannen niet meer op een efficiënte manier gebeurt kan de monitor de beslissing nemen om de Routeplanner-Agent te laten migreren naar een andere computer waar hij mogelijk beter zal kunnen presteren.

De Map-Agent is de plaats waar de kaart bijgehouden wordt. Hij bevat tabellen met locaties en verbindingen tussen deze locaties. Wanneer de routeplanner de weg die hij opbouwt moet uitbreiden zal hij aan de Map-Agent vragen welke wegen genomen kunnen worden uit de laatste locatie waar de planner aangekomen is.

Deze applicatie werd ontwikkeld door Dimitri Verspecht [28]. Hij gebruikt dit als test case voor zijn onderzoek dat hanteert over het gebruik van Borg voor Palm toestellen. Dit verklaart het bestaan van de Monitor-Agent. Vermits een Palm toestel slechts weinig geheugen en rekenkracht bezit kan het berekenen van een route een langdurig proces zijn. De monitor kan dan de beslissing nemen om de routeplanner van de Palm naar een krachtigere machine te laten migreren. Daarom ook worden de GUI en de planner zelf in verschillende agents gestoken. De GUI kan op de oorspronkelijke device blijven terwijl de berekeningen verplaatst worden. Wanneer een resultaat gevonden wordt is de routeplanner niet verplicht om terug te keren naar zijn oorsprong, het hoeft gewoon een bericht de sturen naar de gebruikersinterface.

De routeplanner is een programma dat veel boodschappen kan sturen. Dit is omdat de planner steeds met de database moet communiceren, en dat de twee op verschillende agents zitten. Er moeten dus steeds boodschappen gestuurd worden van de ene naar de andere. Dit is een eigenschap dat we ook terugvinden bij de Quality of Service applicaties. We kunnen dit dus aanpassen zodat er ook gebruik gemaakt wordt van de uitgebreide Borg router.

5.5.3 De Mail Client

De laatste applicatie die we bespreken is de Mail Client. Hier hebben we te maken met een programma dat mail zal afhalen van een email server, zoals bijvoorbeeld de pop mail van de VUB. Het systeem laat toe om emails te downloaden en te bekijken, om ze te beheren en om erin op te zoeken. Het verschil met een gewone email client is dat de mails zich hier niet in een database bevinden. Een mail komt terecht in een agent die alle informatie van de mail bevat. De database bevat dan gewoon een referentie naar de mailagent. Bij de gebruiker bevindt zich dan een stuk van de applicatie die aan de mailserver de lijst van mails zal vragen. Wanneer er een specifieke mail bekeken moet worden zal de agent die ermee overeenkomt naar de virtuele machine van de gebruiker verplaatst worden, zodat deze ook gelezen kan worden indien de netwerk connectie onderbroken wordt.

Het systeem bestaat uit vijf verschillende soorten agents:

- een Mailagent
- een Mailserveragent
- een Usermailagent
- een GUI Agent
- een Mailserver

De eerste soort agent, de mailagent, moet een mail voorstellen. Deze agent wordt gebruikt als een object waar de verschillende velden van de email in opgeslagen worden. Men kan ze dan één voor één weer opvragen en het is mogelijk om erin te zoeken naar een welbepaalde tekst.

De mailserveragent is het onderdeel dat emails zal downloaden en ze ontbinden in verschillende velden die dan in een mailagent kunnen gestoken worden. Hij zal voor de gebruiker ook een tabel opmaken van de verschillende mails die in de mailbox zitten en een referentie bijhouden naar de agent waarin een mail zich bevindt.

De usermailagent is het gedeelte van de applicatie dat zich op de computer van de gebruiker bevindt. Het heeft de verantwoordelijkheid voor het beheren en het tonen van de mails. Het communiceert met de mailserveragent die hem een lijst teruggeeft van de verschillende mails die voor de gebruiker zijn. Het zal dan die mails terugvinden en naar de locale computer verhuizen wanneer de gebruiker deze wil zien.

De GUI agent draagt de code die een grafische voorstelling toelaat van het systeem. Het geeft de commando's die de gebruiker uitgevoerd wil zien door aan de usermailagent. Het zal verder ook de mails structureren voor het weergeven op het scherm van de gebruiker.

De mailserver is een onderdeel die toegevoegd werd om een gewone email server, zoals een pop server, te simuleren. Netwerk communicatie met componenten die geen Borg agents zijn is mogelijk in Borg. Hiervoor bestaat een socket API die natives in Borg definieert die de communicatie met een email server toelaten. Toch blijft het werken met sockets iets dat niet eenvoudig is en het protocol om met een pop server te interageren moet bestudeerd worden. Dit is geen primordiaal onderdeel van de applicatie vermits het voornamelijk dient voor een onderzoek naar het gedrag van agents. Daarom gaat de mailserver een simulatie zijn van de echte mail server. Het enige wat de mailserver doet is het uploaden van emails naar de mailserveragent.

Deze applicatie werd ook door Dimitri Verspecht ontworpen en dient voor zijn onderzoek naar het gebruik van Borg op draagbare devices zoals een Palm [28]. Het wordt gebruikt in ons onderzoek om te zien hoe het zich zal gedragen in een systeem dat gebruikt wordt door applicaties die veel netwerk trafiek veroorzaken. Er wordt dan nagegaan of Quality of Service wel of niet een invloed heeft gehad op het programma.

5.6 Testresultaten

5.6.1 Setup

De verschillende experimenten die uitgevoerd werden gebeurden steeds op een enkele computer. Hierop werden verschillende versies van de Borg router opgestart die met elkaar verbonden waren. De gebruikte router was niet enkel de met Quality of Service uitgebreide router, maar ook de klassieke router, om te testen hoe een experiment op die versie verliep.

De communicatie tussen de verschillende routers moest tijdens de experimenten nooit over een IP netwerk gaan. Dit zorgt voor minder vertragingen voor de boodschappen, wat als resultaat heeft dat de router sneller de berichten zal binnenkrijgen en ze dus ook sneller zal moeten behandelen. Op die manier kan het uithoudingsvermogen van een router gemakkelijker getest worden.

5.6.2 Verwachtingen

Experiment 1: Wall Street

Het eerste experiment die uitgevoerd werd was die met de Wall Street applicatie. Om te beginnen werd het programma uitgetest op een gewone router. Daarna werd het uitgevoerd op de met Quality of Service uitgebreide router. We verwachtten hier als resultaat dat de router met Quality of Service beter zou presteren dan de gewone router. Door de congestie controle had de uitgebreide router het langer moeten uithouden en zouden berichten met een hoge prioriteit afkomstig van de Wall Street messenger af en toe of zelf regelmatig verwijderd moeten zijn uit het systeem. We hadden verwacht dat er bij de gewone versie na een tijdje meer en meer vertragingen zouden optreden doordat de router queue groter en groter zou beginnen worden en dat de berichten dus langer zouden moeten wachten om behandeld te worden, terwijl dat minder het geval zou zijn voor die versie met Quality of Service ingebouwd.

Experiment 2: Monitoring

Het tweede experiment was die met de Monitoring applicatie. We wilden hier zien of het gebruik van verschillende soorten prioriteiten een verschil zouden geven bij de monitor en de tracker. Hiervoor werd de monitor aangepast om een lijst van waarden op te bouwen zoals de tracker. De tracker en de monitor gingen zich dan alle twee registreren bij de sensor. De monitor houdt dan een teller bij voor het aantal waarden dat het al gekregen heeft. Wanneer er 5000 waarden geweest waren ging de monitor zich deregistreren, wat leidt tot het stoppen van de stroom van berichten voor de monitor, maar ook voor de tracker, vermits beide geregistreerd moeten zijn om de stroom te laten vloeien. De lijsten van de tracker en de monitor worden dan onderling vergeleken.

Het experiment werd eens uitgevoerd met enkel één sensor in het netwerk. Daarna werd het uitgevoerd met in het netwerk nog andere programma's, onder meer de Wall Street applicatie dat werkte met steeds meer clients. Wat verwacht was is dat de lijsten verschillen zouden tonen. Deze zouden klein of zelfs onbestaand zijn in het begin, wanneer er enkel die applicatie, in een enkele versie, aan het werken is, maar zou groter worden naar gelang er meer een meer verkeer zou komen in het netwerk.

Experiment 3: The Big Mixup

Het derde experiment is een waar de verschillende soorten programma's door elkaar moeten werken in een Borg netwerk. We gebruiken daar naast de Wall Street en de Monitor applicatie ook de Meetingplanner, de Routerplanner en de Mail Client. Wat we hier mee willen bereiken is zien of Quality of Service een invloed zal hebben op de uitvoer van de verschillende programma's. Daarvoor voeren we het experiment eerst uit met de gewone Borg router en dan eens met de uitgebreide router. Wat we hopen te zien is dat het experiment met de aangepaste router beter zal verlopen dan op die de originele router.

5.6.3 Observaties

De praktijk is soms anders dan wat de theorie verwacht, en dit is een van die theorieën die met de praktijk niet helemaal overeen komen. De observatie die gemerkt werden waren niet diegene die verwacht werden.

Wall Street

We hadden verwacht dat de congestie controle een verschil zou uitmaken. Dit gebeurde niet echt. Het experiment met de klassieke router toonde dat deze niet meer op de juiste manier te werk ging nadat er twintig clients met de server geconnecteerd waren. De router waar de server op zat te werken begaf het en na een korte tijd viel die evaluator uit. De andere routers uit het systeem konden wel nog verder werken, hoewel zo ook moeilijkheden hadden. Wanneer een server aangemaakt werd op de Borg nameserver, dan had de router moeilijkheden om te weten of nieuwe agents, en dus ook nieuwe clients, wel of niet geregistreerd waren bij de nameserver, toch totdat het door had dat de nameserver dood was.

We hadden dan verwacht dat het beter ging gaan met de met Quality of Service uitgebreide router. Vermits er congestie controle was zou die het te groot aantal boodschappen moeten kunnen opnemen en toch verder moeten blijven werken. We hadden wel verwacht dat de router het niet eindeloos zou volhouden, maar toch wel een tijdje langer dan de originele router. Wat bleek is dat er weinig of geen verschil was. Het is bij het zelfde aantal clients, namelijk twintig, dat de uitgebreide router het begaf. De oorzaak van deze observatie wordt besproken in sectie 5.6.4.

Monitoring

Bij dit experiment is het weer eens zo dat we niet wat we verwachten observeren. We dachten dat de twee tabellen verschillen gingen vertonen naar gelang het netwerk meer en meer belast werd. Dit was niet het geval. Wanneer er enkel een sensor met zijn monitor en tracker in het netwerk aanwezig zijn, dan zijn de twee tabellen gelijk, maar dat was wel te verwachten. Maar zelfs wanneer er meer en meer netwerk verkeer is blijven de twee gelijk. Wanneer de Wall Street applicatie opgestart wordt om het netwerk nog meer te belasten, dan loopt het experiment snel af, want dan gaat de router het begeven na een zekere tijd omdat het te veel berichten moet behandelen.

The Big Mixup

Zoals we nu al uit de vorige observaties kunnen verwachten maakt het toevoegen van Quality of Service in dit experiment weinig uit. Bij de originele router werken de applicaties op een degelijke manier tot er een van de routers overbelast wordt en het moet begeven. Waar we hadden verwacht dat de Quality of Service router betere resultaten zou leveren was het weer niet het geval. De routers gaven het ook op na een tijd dat even langer is dan in het geval van de originele router. Verschillen in vertragingen vielen ook niet op. Bij de twee versies van de router hadden de applicaties ongeveer hetzelfde gedrag, voor zolang het geheel toch nog degelijk werkte.

5.6.4 Uitleg bij de observaties

De experimenten verliepen niet zoals verwacht. De oorzaak hiervan moeten we proberen op te zoeken zodat er verder gewerkt kan worden aan een oplossing die wel zoals verwacht zal werken.

Een eerste zaak dat we kunnen opmerken en dat misschien ook wel veel kan uitleggen is de snelheid waarop een boodschap behandeld wordt. De router is snel, en een boodschap dat binnenkomt blijft nooit lang in de router queue aanwezig. Men kan zelfs zeggen dat de snelheid waarmee een boodschap behandeld wordt bijna even groot is als die waarmee een boodschap gestuurd wordt. Hieruit kunnen we afleiden dat een agent nooit veel boodschappen in een queue zal bevatten.

Hieruit kunnen we de observaties dat we tegen zijn gekomen mogelijk wel afleiden. Als we om te beginnen de reden bekijken waarom de router het begeeft. Het gaat ten onder omdat het aantal berichten in de queue te groot wordt. We hebben congestie controle toegevoegd dus zou dat het moeten verhelpen. Maar zoals in het vorig hoofdstuk uitgelegd staat is die congestie controle actief op de individuele agents, niet op het totaal aantal berichten in de router. Dus wanneer de queue te veel berichten begint te krijgen gaat de congestie controle de berichten tegenhouden van de agents die al te veel berichten in de queue hebben. Maar vermits de berichten snel verwerkt worden heeft een agent nooit echt veel berichten in de queue aanwezig. Wat er gebeurt is dat er te veel agents zijn die op de router actief zijn. Dit is iets dat voorkomt in de gevallen met en zonder uitgebreide router.

De reden waarom de congestie controle niet op een correcte manier werkt kan dus zijn omdat de restrictie van een aantal berichten voor de individuele agents niet genoeg is. Er zou ook een limiet moeten zitten op het totaal aantal berichten in de router queue, waarna geen enkel nieuw bericht meer aanvaard wordt. Dus gebruiken we het totaal aantal berichten niet meer enkel om te bepalen wanneer een agent geen berichten in de queue mag toevoegen, maar ook in het algemeen om te zeggen dat geen enkel nieuw bericht nog toegelaten is.

De snelheid waarop de berichten behandeld worden kan misschien ook uitleggen waarom de prioriteiten niet veel verschil uitmaken. Wanneer een bericht met hoge prioriteit in de router queue aankomt zal die behandeld worden voor die met een lagere prioriteit. Maar vermits het behandelen van de berichten snel gebeurt komen de berichten met lagere prioriteit ook snel aan de beurt. Daarom kan er weinig of geen verschil opgemerkt worden tussen de verschillende gebruikte prioriteiten.

Dit was een eerste mogelijke uitleg. Een andere factor die een invloed kon hebben is de manier waarop de experimenten gebeurt zijn. De experimenten gebeurden steeds op slechts één computer. De verschillende routers zaten dus op dezelfde computer. Er was wel een Borg netwerk maar deze lag niet over een fysiek IP netwerk. Dit kon een verschil geven op de vertragingen die berichten opliepen tijdens het versturen. De echte negatieve invloed hiervan op de experimenten is niet zeker.

Het is natuurlijk ook mogelijk dat het probleem ergens anders ligt. Het kan zijn dat de verschillen die opgemerkt moesten worden niet zijn opgetreden omdat de router het begaf voor dat die aanpassingen hun werking konden doen. Dus dat een ander probleem het falen van de router veroorzaakt zodat de experimenten niet kunnen aantonen wat van hen verwacht wordt. Een andere reden waardoor de experimenten niet de verwachte resultaat hadden kan te maken hebben met de snelheid van de computer waarmee gewerkt werd. Indien deze trager had geweest, dan had het behandelen van de boodschappen minder snel gebeurd. Dan had het misschien wel gekund dat een agent meer berichten in de router queue bezat. Maar vermits de snelheid van de computers maar blijft stijgen en dat de gebruikte computer traag was tegenover wat men tegenwoordig op de markt vindt, zal hieruit geen oplossing gevonden worden voor toekomstige aanpassingen aan de router.

De uitleg die we hier proberen te maken blijft theoretisch. Sommige zaken kunnen wel opgemerkt worden maar het zoeken wat er allemaal echt misgaat is niet een eenvoudige zaak. Wat wel gezegd kan worden is dat het nodig zal zijn om de Quality of Service router nog eens goed te herdenken en een paar aanpassingen maken, met de hoop dat het dan wel een verschil zal uitmaken.

5.7 Gebruik van Quality of Service in Borg

Werken met Quality of Service in Borg is eenvoudig wanneer we de uitgebreide router gebruiken. Er moet in de source code van en agent enkel een oproep naar de `messageprioritiy()` native gebeuren die de prioriteit van de boodschappen die door de agent gestuurd zullen worden zal bepalen. Indien de router het verwachte gedrag vertoont zal dit zorgen voor het beheren van prioriteiten voor de berichten en voor congestie controle. Maar indien we gebruik maken van de originele versie van de router wordt het minder eenvoudig om Quality of Service te gebruiken.

Het beheren van de prioriteiten voor de berichten in de router kan gewoonweg niet bij de originele versie. Een agent kan niet aan de message queue van de router. We zouden dus een ander systeem moeten vinden, zoals bijvoorbeeld een agent als router gebruiken en de communicatie nog eens een niveau hoger te plaatsen. Hiervoor moeten we dan een router maken die in Borg zelf geschreven is.

Voor de congestie controle kan er een oplossing gevonden worden waar we niet verplicht zijn om de een router in Borg te maken. We zouden bijvoorbeeld kunnen werken met delay's. Wanneer een agent een stroom van berichten moet sturen kan men dan instellen dat er tussen het opsturen van opeenvolgende berichten een zekere tijd gewacht wordt, waardoor er minder berichten tegelijkertijd in het netwerk zullen geraken.

Maar zelfs als we het gebruik van Quality of Service in Borg kunnen garanderen blijven we nog altijd met de vraag of er wel een nut is aan Quality of Service in Borg. We zagen in het hoofdstuk over Quality of Service en in het begin van dit hoofdstuk dat de applicaties die echt gebruik maken van zulke eigenschappen vooral programma's zijn waar beeld en geluid continue over het netwerk moeten worden gestuurd. Dit zijn applicaties die men in Borg niet terugvindt. Af en toe kan men wel een programma bedenken die een voordeel zou vinden in het gebruik van Quality of Service, maar voor de meeste Borg applicaties is het gebruik van Quality of Service niet echt een noodzaak.

5.8 Conclusies

In dit hoofdstuk wou er nagegaan worden of de theoretische conclusies getrokken in het vorig hoofdstuk zich in de praktijk zouden verder zetten. Hiervoor werden verschillende voorbeeldapplicaties gemaakt die de eigenschappen van Quality of service moeten testen. De twee voornamelijkste zijn de Wall Street en de Monitoring applicatie. De eerste stelt een beurs applicatie voor die aan de gebruiker een continue stroom van informatie aanbied over de verschillende opties die het systeem beheert. Daarnaast kan de gebruiker informatie opvragen en opties kopen of verkopen.

Bij de tweede applicatie hebben we een sensor die zijn waarde doorstuurt naar ten eerste een monitor, die de waarde aan de gebruiker toont, en ten tweede een tracker, die de verschillen in de waarde opslaat om ze naderhand weer te kunnen opzoeken. Nog een paar andere programma's worden gebruikt, namelijk een Routeplanner, een Meetingplanner en een Mail Client.

Het doel van de verschillende programma's was om enerzijds te testen of er wel een verschil was tussen de originele router en de met Quality of Service uitgebreide router, en anderzijds om te testen of het gebruik van verschillende prioriteiten ook in de praktijk een verschil zou gegeven. Met de toegevoegde priority queue en congestie controle hadden we verwacht om op te merken dat de stabiliteit van de aangepaste router beter zou zijn en dat het verschil in prioriteit verschillen zou geven wanneer het netwerk sterk gebruikt wordt.

De voorspellingen die gemaakt waren bleken niet correct te zijn. De toegevoegde waarde die men verwacht had werd niet teruggevonden. De experimenten toonden geen verschil in stabiliteit uit en het gebruik van de prioriteiten deed niets uit aan het verloop van de programma's. De redenen die aan de oorzaak liggen van de observaties die gemaakt worden zijn divers en niet eenvoudig te bepalen. Wat men wel zeker kan zeggen is dat de router uitgebreid met Quality of Service nog meer aangepast moet worden indien men echt een verschil wil merken bij het gebruik ervan.

Hoofdstuk 6

Conclusies

6.1 Introductie

Het doel van deze thesis was om na te gaan of het gebruik van Quality of Service aspecten in een Mobile Multi Agent Systeem zoals Borg een verschil zou uitmaken voor de gebruiker van het systeem. Om dat te kunnen doen werd er een uitbreiding toegevoegd aan de Borg router die verschillende technieken implementeerde waarvan er gedacht werd dat ze toegevoegde waarden zouden brengen aan het systeem. Er werden dan verschillende applicaties in Borg geschreven waarmee er getest kon worden of het toevoegen van de Quality of Service eigenschappen de verwachte werking zou hebben.

6.2 Overzicht

Deze verhandeling werd begonnen met een uitleg over wat Quality of Service is. De verschillende niveau's dat we ervan tegenkomen werden besproken. Het zijn de Best Effort, de Differentiated Service en de Guaranteed Service. Nadien werd er getoond waarom men Quality of Service gebruikt. Er kwam dan een uiteenzetting van de verschillende onderdelen van een netwerk verbinding die een invloed spelen op de kwaliteit van de diensten ondervonden door de gebruiker. Deze zijn de beschikbaarheid van het netwerk, de overdracht die het netwerk biedt, de vertragingen die pakketten oplopen in het netwerk, de verschillen in vertragingen voor opeenvolgende pakketten en het verlies van data. Het hoofdstuk eindigde met een analyse van verschillende methodes om Quality of Service aan te bieden.

In het volgende hoofdstuk werd de programmeertaal gebruikt voor het onderzoek besproken. Deze is Borg; een Mobile Multi Agent Systeem. Dit betekent dat de programmeercode verdeeld wordt over verschillende agents.

Deze agents kunnen verspreid zijn over verschillende computers en met elkaar communiceren door middel van berichten, om de mogelijkheid te geven om op een eenvoudige manier gedistribueerd te programmeren. Een agent heeft dan ook de mogelijkheid om te migreren, wat betekent dat hij de beslissing kan nemen om op een andere computer verder te werken. Dit werd allemaal in hoofdstuk 3 uitgelegd, samen met een paar voorbeelden van het gebruik van Borg. Er kwam dan nog een technische bespreking van de methoden gebruikt om de boodschappen die tussen de agents gestuurd worden door het netwerk te transporteren. Om het hoofdstuk te eindigen kwam er een vergelijking tussen Borg en andere systemen die toelaten om op een vereenvoudigde manier gedistribueerde applicaties te bouwen, namelijk CORBA en Java RMI, waaruit we konden afleiden dat de communicatie in Borg eenvoudiger was en dat de migratie van de agents die Borg toelaat een voordeel voor het systeem is.

Het vierde hoofdstuk werd besteed aan de aanpassingen die gedaan werden om Quality of Service te bieden in Borg. Hiervoor was een uitleg over de werking van de originele Borg router nodig. Dit werd dan gevolgd door de bespreking van de aanpassingen zelf. Deze zijn voornamelijk het toevoegen van een systeem die prioriteiten toelaat voor de berichten en congestie controle. Dit liet ons toe om de verschillende aspecten die Quality of Service beïnvloeden over te brengen naar concepten die beter bij Borg horen. We keken hoe de aspecten zich vertaalden en probeerden al eerste conclusies te trekken over welke verschillen de Quality of Service in Borg ging veroorzaken.

Vertrekkende van de theoretische conclusies uit hoofdstuk vier gingen we in hoofdstuk vijf de experimenten bespreken die gingen bepalen of deze conclusie wel of niet overeenkwamen met wat we in het systeem uiteindelijk tegenkwamen. Verschillende applicaties werden uit de doeken gedaan waarmee we de elementen toegevoegd aan de Borg router konden uittesten. De experimenten dat met deze applicaties gebeurden werden dan behandeld samen met de verwachte observaties. Uiteindelijk kwamen de resultaten zelf, die bleken niet overeen te komen met wat er verwacht was, en we gaan dan ook verder met de uitleg waarom het niet overeenkwam.

6.3 Resultaten

De eerste conclusies over de met Quality of Service uitgebreide Borg waren theoretisch. Ze volgden in hoofdstuk vier uit de verwachtingen die vloeiden uit het uitbreiden van het systeem. We dachten dat de congestie controle en de prioriteiten op de berichten een verschil zouden bieden tegenover het oorspronkelijk systeem.

Een eerst zaak die toen werd afgeleid was dat het systeem meer stabiliteit zou vertonen. Dit kwam uit de feit dat congestie controle ervoor zou zorgen dat de Borg router niet overspoeld zou worden met boodschappen en daarom minder snel zou beginnen falen.

Er werd dan ook gedacht dat het gebruik van verschillende prioriteiten voor de berichten zou zorgen dat sommige berichten sneller zouden aankomen dan andere. Daarbij dachten we ook nog dat de prioriteiten, gekoppeld aan de congestie controle als gevolg zouden hebben dat het aantal berichten die in het netwerk gestuurd werden niet overeen zouden komen met het aantal die uiteindelijk op hun bestemming aankomen. Dit zou gekomen zijn doordat er bij de hoogste prioriteit niet meer gekeken wordt of het bericht wel aangekomen is, terwijl het bij de andere prioriteiten wel het geval is.

Dat waren conclusies die we dachten terug te vinden bij de resultaten van de experimenten. Dit was niet het geval. Uit de resultaten bleek dat het verwachte verschil tussen de gewone Borg en de met Quality of Service uitgebreide Borg niet echt aanwezig was. Bij het experiment waar het de bedoeling was om te observeren of de stabiliteit van de uitgebreide router beter was of niet dan die van de originele router bleek dat de beide versies op ongeveer het zelfde ogenblik gingen beginnen falen. Het werk dat de congestie controle moest uitrichten gaf blijkbaar geen verschil.

In een volgend experiment waar het gebruik van verschillende prioriteiten moest getest worden zagen we geen verschil tussen de berichten gestuurd met hoge prioriteit en lage prioriteit. Hier hadden we weer eens verwacht dat de congestie controle een rol ging spelen wanneer het netwerk druk bezet was maar weer maakte het weinig uit. Ook het experiment waar applicaties die niet echt gebruik moesten maken van de aanwezige uitbreidingen samen op het systeem moesten werken met applicaties die ze wel gebruikte toonden geen verschillen aan.

Hieruit kunnen we afleiden dat de aanpassingen die aan de router gebracht werden weinig effectief waren. Een mogelijke oorzaak hiervan ligt in de feit dat een bericht die in de router aankomt snel behandeld wordt, en dat die snelheid als gevolg heeft dat er nooit veel berichten van eenzelfde agent aanwezig in de router queue zijn. Vermits de congestie controle inwerkt op het aantal berichten aanwezig in de queue en afkomstig van eenzelfde agent heeft de snelheid van het systeem tot gevolg dat de congestie controle weinige veranderingen veroorzaakt. Voor dezelfde reden speelden de verschillende prioriteiten slecht een kleine rol. Ook is het mogelijk dat de condities waarin de experimenten gebeurden een rol gespeeld hebben.

6.4 Conclusie

We wilden voor deze thesis nagaan of Quality of Service gebruiken in een Mobile Multi Agent Systeem zoals Bog een verschil zou geven bij het gebruik van het systeem. De aanpassingen die hiervoor aan de Borg router werden gemaakt bleken bij de experimenten weinig effectief te zijn. De vergelijking tussen de oorspronkelijke versie van de router en de uitgebreide router toonde aan dat de twee bij het gebruik equivalent waren. Daarbij komt nog eens dat applicaties vinden in Borg die gebruik maken van de Quality of Service eigenschappen niet altijd voor de hand liggen en dat de meeste programma's dat we in Borg tegenkomen weinig nut hebben aan Quality of Service.

6.5 Verder werk

Indien de interesse voor het gebruik van Quality of Service in Borg blijft bestaan, dan kunnen de resultaten van deze onderzoek wel dienen. Het was een eerste poging die niet op de verwachte manier is afgelopen. Dit betekent niet dat het onmogelijk is om Quality of Service aan te bieden in Borg.

Om verder te kunnen werken moet er eerst gekeken worden naar wat er hier mis gelopen is. Er moet een grondig onderzoek gebeuren naar wat er juist verkeerd gaat met het actueel systeem en daarmee nieuwe uitbreidingen bedenken die misschien wel een effect zullen hebben op de kwaliteit van de diensten die Borg aanbiedt. We denken hier dan vooral aan een betere congestie controle die ook rekening zou houden met het totale aantal berichten aanwezig in de router.

Bibliografie

[1] Nortel networks: *Introduction to Quality of Service (QoS)*, September 2001,
http://www.nortelnetworks.com

[2] *Webopedia: Online Dictionary for Computer en Internet tems*,
http://www.webopedia.com/TERM/l/latency.html

[3] Y. Mansour en B. Patt-Shamir: *Jitter Control in QoS Networks*, Dept. of Computer Science en Dept. of Electrical Engeneering, Tel-Aviv University, December 20, 1998.

[4] Network Magazine, The Indian Edition: *Examin the SLA*,
http://www.networkmagazineindia.com/200110/focus2.htm

[5] A. Agarwal: *Quality of Service (QoS) in the New Public Network Architecture*, Concordia University, Montreal, QC, IEEE Canadian Review - Fall / Automne 2000

[6] G. Hansen: *Quality of Service (QoS)*,Object Services and Consulting, Inc.
http://www.objs.com/survey/QoS.htm

[7] S. Blake, D. Black, EMC Corporation, M. Carlson, Sun Microsystems, E. Davies, Nortel UK, Z. Wang, Bell Labs Lucent Technologies, W. Weiss, Lucent Technologies, December 1998: *An Architecture for Differentiated Services*.
http://www.ietf.org/rfc/rfc2475.txt?number=2475

[8] A. Pezzuto: *What is Quality of Service? A brief introduction to "Quality of Service"concepts.*, QoSmagazine.net, Januari 2002.
http://www.qosmagazine.net/What_is_Quality_of_Service_1.asp

[9] J. Tiberghein: *Tele-informatica*, Dept. Toegepaste Informatica, VUB,
http://info.vub.ac.be

[10] Nokia Networks Broadband Systems: *The Five Nines in IP Network*, N9137_NN_US-1101_1.0,
http://www.nokia.com

[11] Broadband DSLReport: *High Speed*,
http://www.dslreports.com/information/kb/high+speed

[12] C. Bajorek: *Real-time QoS Monitoring For VoIP*,CT Labs,
http://www.cconvergence.com/article/CTM20010202S0004

[13] Anritsu corporation: *Technical Note: What is Wander*
http://www1.anritsu.co.jp/MPB/Products/pdf_e/Wander_EE1100.pdf

[14] Network Luminous: *Quality of Service*, 2001 Luminus Networks, Inc, Maart 2001, 00-024MC

[15] Cisco Systems: *Per-Virtual Circuit (Distributed), Weighted Random Early Discard (Per-VC DWRED)*,
http://www.cisco.com/warp/public/121/pervc-dwred.html

[16] Stardust.com: *The Need for QoS*, stadust.com, Inc, QoSForum.com, Juli 1999

[17] Stardust.com: *QoS protocols and architectures*, stadust.com, Inc, QoSForum.com, Juli 1999

[18] J. Kershaw: *What is ATM*, Nextep Broadband, 221-003-BS-085-1-0 Rev 1.0, Mei 2001.
http://www.nextep.com.au/pdf/What_is_ATM.PDF

[19] A. Marshall: *ATM Asynchronous Transfer Mode: An Overview*, Department of Electrical and Electronic Engineering, The Queen's University of Belfast,
http://www.pcc.qub.ac.uk/tec/courses/network/ATM/ATM.html

[20] Sitaraman VR: *IP QOS OVER ATM*,
http://www.netlab.ohio-state.edu/j̃ain/cis788-99/ip_qos_atm/

[21] F. Van Nieuwenhuyzen: *Zinsproductie met behhulp van categoriale grammatica*, AI-Lab, DINF, VUB,
http://arti.vub.ac.be.

[22] T. D'Hondt: *The Pico Page*, PROG, DINF, VUB,
http://pico.vub.ac.be

[23] W. Van Belle: *Welcome To Borg*, PROG, DINF, VUB, http://borg.rave.org

[24] The OMG: *What is CORBA*, http://www.omg.org/corba/whatiscorba.html

[25] Java Remote Method Invocation, http://java.sun.com/products/jdk/rmi

[26] T. D'Hondt, W. Van Belle, K. Verelst: *Location Transparent Routing in Mobile Agent Systems Merging Name Lookup with Routing*, PROG, DINF, VUB.

[27] W. Van Belle: *Agent Mobility and Reification of Computational State*, PROG, DINF, VUB.

[28] D. Verspecht: *Mobile agents on Mobile Platform*, PROG, DINF, VUB.

[29] P. Verheyden: *Ontwikkeling van een Failsafe Systeem voor Mobile Multi Agenrt Systemen*, PROG, DINF, VUB.