GENERIEK ACCOUNTING−FRAMEWORK

Arthur de Jong

afstudeerverslag

2001−01−30

West Consulting BV
Delftechpark 5
2628 XJ Delft
Postbus 3318
2601 DH Delft
015 219 1600
http://www.west.nl/
info@west.nl

Technische Universiteit Delft
Faculteit Informatietechnologie en Systemen
Afdeling Information Systems & Software Engineering
Groep Software Engineering, Programmeren, Programmeertalen & Compilers
http://sepc.twi.tudelft.nl/

Afstudeergegevens

Titel:	Generiek accounting−framework
Schrijver:	Arthur de Jong
Afstudeercommissie:	ir. F. Ververs (TUDelft)
	ir. G.J. van Oosten (West Consulting BV)
	ir. H.J.A.M. Geers (TUDelft)
	prof.dr.ir. J.L.G. Dietz (TUDelft)

Korte samenvatting:

Accounting is het vakgebied dat betrekking heeft op het verzamelen en transporteren van gegevens over het gebruik van electronische dienstverlening. Accounting−gegevens worden met behulp van een accounting−protocol getransporteerd naar een accounting−server waar de gegevens worden verwerkt. Er is een sterke groei te constateren in zowel de toepassing van accounting als in de verschillende vormen van dienstverlening. Als verschillende organisaties dezelfde accounting−gegevens moeten kunnen gebruiken, is het belangrijk dat accounting−gegevens uitwisselbaar zijn. Hierbij zouden alle organisaties hetzelfde protocol kunnen gebruiken. Veel accounting−protocollen zijn echter beperkt tot één specifieke toepassing. Een andere mogelijkheid is het ontwerpen van een framework dat meerdere accounting−protocollen ondersteunt. In dit verslag is een ontwerp gemaakt van een dergelijk framework. Aan de hand van dit ontwerp is een implementatie gemaakt die wordt toegelicht. Een groot deel van het ontworpen framework is herbruikbaar voor andere gerelateerde toepassingen zoals authenticatie en autorisatie. In het framework kunnen verschillende accounting−protocollen tegelijk worden ondersteund, zodat gegevens in verschillende formaten kunnen worden verwerkt. Tevens is van een aantal protocollen voor het framework een implementatie gemaakt.

Voorwoord

Binnen West Consulting ben ik in het kader van een afstudeeropdracht bezig geweest met een onderzoek naar accounting−protocollen en accounting−systemen. In eerste instantie is een onderzoeksverslag [acc−prot] gemaakt over accounting−protocollen en de requirements die bij een algemeen toepasbaar accounting−protocol een rol spelen. Dit afstudeerverslag en de bijbehorende implementatie zijn het logisch gevolg van de conclusies uit het onderzoeksverslag.

In dit verslag ligt de nadruk op accounting−systemen en wordt uiteindelijk een implementatie gepresenteerd van een generiek accounting−framework dat voor meerdere vormen van dienstverlening toepasbaar is.

Voor meer informatie over mijn afstudeerwerk, inclusief het onderzoeksverslag, sheets van presentaties, alle Java−code en bijbehorende documentatie is een Internet−pagina [aaapage] beschikbaar.

Graag wil ik iedereen bedanken die met feedback voor dit verslag is gekomen, zodat het een verslag in de Nederlandse taal is geworden. Daarnaast wil ik graag Hella nog speciaal bedanken voor haar steun en het herhaaldelijk doorlezen van mijn verslag.

2001−01−30

Arthur

Samenvatting

Accounting is het vakgebied dat betrekking heeft op het verzamelen en transporteren van gegevens over het gebruik van bepaalde vormen van electronische dienstverlening. Accounting wordt vaak toegepast om klanten een gespecificeerde rekening te kunnen sturen voor het gebruik van aangeboden diensten. Accounting is echter ook interessant voor trend−analyse en auditing−toepassingen. Het wordt toegepast bij allerlei vormen van electronische dienstverlening, zoals het aanbieden van een Internetverbinding, telefonie, application hosting en nog vele anderen. Er vindt een sterke groei plaats in deze vormen van dienstverlening wat het een interesant vakgebied maakt.

Accounting−gegevens worden van de plek waar ze worden gegenereerd naar een centrale accounting−server getransporteerd waar ze worden verwerkt. Dit transport gebeurt met behulp van een accounting−protocol. Hierbij vind het transport vaak binnen dezelfde organisatie plaats, omdat één bedrijf zowel de dienstverlening als de billing doet. Steeds vaker worden accounting−gegevens tussen verschillende organisaties uitgewisseld. Dit gebeurt bijvoorbeeld als een organisatie diensten aanbiedt voor de klanten van een aantal andere organisaties. Bij deze uitwisseling van accounting−gegevens spelen zaken als beveiliging van de gegevens en het gebruik van een standaard accounting−protocol voor de uitwisseling een grote rol.

Omdat verschillende toepassingen sterk uiteenlopende eisen aan accounting−protocollen stellen, is de ontwikkeling van een enkel standaard accounting−protocol dat voor alle toepassingen goed bruikbaar is niet waarschijnlijk. Een oplossing waarbij verschillende deelprotocollen in een framework kunnen worden gecombineerd om aan de verschillende eisen te kunnen voldoen heeft daarbij interesante mogelijkheden. Als het framework bruikbaar is voor bestaande en toekomstige accounting−protocollen is het mogelijk om accounting−gegevens van een brede verzameling van toepassingen te verwerken.

Voor dit afstudeerwerk is een framework ontworpen en geïmplementeerd waarin eenvoudig meerdere accounting−protocollen kunnen worden gebruikt. Voor het ontwerpen van het framework zijn requirements opgesteld waaraan een dergelijk accounting−systeem moet voldoen. Om de structuur van het systeem aan te geven is een model geschetst waarin zaken als accounting−protocol en accounting−server zijn uitgewerkt. Een accounting−protocol wordt opgedeelt in een transportprotocol, een recordformat en een verzameling berichten. Voor de verzameling berichten wordt een aantal modellen gepresenteerd waarmee accounting kan worden uitgevoerd.

Aan de hand van de requirements en het model zijn een ontwerp en een implementatie gemaakt. Het is mogelijk voor een applicatie om met een klein aantal regels code het framework te configureren en te gebruiken. Het is mogelijk om meerdere accounting−protocollen tegelijk te ondersteunen en meerdere berichtenmodellen te gebruiken. Hiermee is het framework toepasbaar in verschillende omgevingen waar uiteenlopende eisen aan een accounting−systeem zijn gesteld.

Met het voor dit afstudeerwerk ontworpen en geïmplementeerde framework kunnen meerdere accounting−protocollen worden geïntegreerd. Met de binnen dit framework gedefinieerde modulen is het mogelijk accounting−gegevens op een generieke manier te transporteren. Voor een uiteindelijke toepassing van het systeem hoeft het framework alleen nog maar te worden aangevuld met de juiste modulen voor de verschillende protocollen en toepassingen.

Inhoudsopgave

1. Inleiding

Dit is het afstudeerverslag behorende bij het afstudeerwerk van Arthur de Jong uitgevoerd bij West Consulting BV. Dit verslag behandelt een generiek model voor het uitvoeren van accounting. Voor steeds meer vormen van electronische dienstverlening is het nodig om gedetailleerde gegevens van gebruik van de dienstverlening bij te houden. Het grootschalig aanbieden van Internet−verbindingen, de toename in mobiele dienstverlening, een groei in application−hosting en vele andere nieuwe vormen van electronische dienstverlening maken accounting een steeds interessanter en breder toegepast vakgebied. Ook wordt er steeds vaker een totaalpakket van verschillende gerelateerde diensten aangeboden.

Bij deze nieuwe vormen van dienstverlening spelen financiële belangen een grote rol. Het is belangrijk om inzicht te hebben in de afname van de diensten, al was het maar om gespecificeerde rekeningen te kunnen sturen. Met een uitgebreide toepassing van accounting in verschillende vormen van dienstverlening wordt het steeds belangrijker dat verschillende systemen onderling probleemloos gegevens kunnen uitwisselen.

Als verschillende bedrijven samen een dienst moeten verlenen voor een gezamenlijke klant en daarvoor gegevens moeten uitwisselen is dit een gecompliceerde zaak. Hierbij is het belangrijk om onderling gegevens uit te wisselen in een gestandaardiseerd formaat of een systeem te hebben waarmee in veel verschillende formaten gegevens eenvoudig zijn uit te wisselen. Voor deze problemen zal in dit verslag een oplossing worden gezocht.

Om inzicht te krijgen in de aspecten die bij accounting een rol spelen en om een inleiding tot accounting te geven zal eerst worden beschreven wat accounting is. In hoofdstuk twee komen doelen, toepassingen en eigenschappen van accounting−protocollen en accounting−systemen aan de orde. Verschillende modellen voor het versturen van accounting−gegevens worden uitgelegd, alsmede beveiligingsaspecten van accounting−transport.

Na de inleiding tot accounting en accounting−problematiek zal het probleem waar dit verslag zich op concentreert worden geformuleerd. In hoofdstuk drie komt ook de doelstelling van dit verslag in meer detail aan de orde. De probleemstelling en de doelstelling zullen in de rest van het verslag als basis dienen voor het uiteindelijke resultaat.

In hoofdstuk vier zullen, naar aanleiding van de probleemstelling en aspecten die bij accounting een rol spelen, eisen die aan een oplossing worden gesteld aan de orde komen. Deze eisen vloeien voor een groot deel voort uit toepassingen en modellen die in het tweede hoofdstuk zijn behandeld.

Aan de hand van de doelstelling en de eisen wordt in hoofdstuk vijf een model beschreven dat het gestelde probleem moet kunnen oplossen. Het model beschrijft de structuur van de oplossing, welke elementen er een rol in spelen en geeft een definitie van de gebruikte structuren. Het model geeft een globaal overzicht van de oplossing.

Dit model wordt in hoofdstuk zes verder uitgewerkt tot een ontwerp. Hierbij zullen ook de eerdergenoemde eisen worden meegenomen. In het ontwerp zullen de te implementeren software−modulen worden gedefinieerd, hoe deze modulen samenhangen en welke gegevens zij onderling uitwisselen. Het ontwerp geeft aan hoe de uiteindelijke oplossing kan worden gerealiseerd.

In hoofdstuk zeven wordt beschreven hoe, vanuit de software−modulen in het ontwerp, in Java een implementatie in klassen is gemaakt. Het geeft de globale structuur van de software weer, zonder te veel in detail te treden. Het geeft een overzicht van de gebruikte klassen en hoe uitbreidingen aan het framework kunnen plaatsvinden.

In hoofdstuk acht zal worden aangegeven wat de resultaten van de gekozen oplossing zijn. Er zal worden toegelicht in welke mate het framework aan de doelstelling en de requirements voldoet en tekortkomingen van deze aanpak zullen worden behandeld. Voor enkele tekortkomingen zal worden aangegeven hoe deze kunnen worden verholpen.

Tenslotte zullen in hoofdstuk negen de conclusies van dit project worden gepresenteerd. Aan de hand van de resultaten zullen aanbevelingen voor uitbreidingen en verder onderzoek worden gedaan en de toepasbaarheid van het framework zal worden toegelicht.

2. Accounting

In dit hoofdstuk wordt een inleiding tot accounting gegeven. Het beschrijft de context waarin accounting wordt toegepast, de functie van accounting−protocollen binnen een accounting−systeem, verschillende modellen van berichtuitwisseling, toepassingen van accounting−systemen en beveiligingsaspecten van accounting−gegevens.

Accounting betreft het verzamelen en transporteren van gegevens over gebruik van geleverde (electronische) diensten. Bij het aanbieden van een dienst zijn vaak authenticatie, autorisatie en accounting (AAA) van belang. Bij authenticatie wordt de identiteit van een client (gebruiker van de dienst) vastgesteld. Autorisatie is het bepalen of een client toegang heeft tot een dienst en onder welke voorwaarden (parameters van de dienstverlening). Met behulp van accounting wordt tijdens het verlenen van de dienst inzicht verschaft in het gebruik van de geleverde dienst. Aan autorisatie gaat in het algemeen authenticatie vooraf. Aan accounting gaat vaak autorisatie en authenticatie vooraf.

Een dienst wordt aangeboden door een service−element. Het aanbieden van een enkele dienst aan een client wordt een sessie genoemd. Accounting houdt zich bezig met het verzamelen en transporteren van gegevens over sessies. Deze gegevens worden binnen accounting−berichten in accounting−attributen vastgelegd en opgeslagen. Het gaat hier onder andere om gegevens over de duur van de sessie, de identiteit van de client, de kwaliteit van de geleverde dienst en andere zaken die van belang kunnen zijn. Bij accounting wordt er niet direct een geldelijke waarde aan de attributen toegekend, dit gebeurt pas in een later stadium. Er worden alleen direct gemeten gegevens zoals vertsuurde en ontvangen hoeveelheid bytes getransporteerd.

Het doel van accounting is het kunnen maken van trendanalyses, het uitvoeren van auditing, het sturen van rekeningen (billing) of het toewijzen van kosten (cost allocation). Hiermee wordt inzicht verschaft in het gebruik van diensten en wordt het aanbieden beheersbaar gemaakt.

Trendanalyse

Met trendanalyse wordt inzicht verschaft in gebruik en ontwikkelingen in gebruik van diensten. Met trendanalyse wordt een voorspelling van het gebruik van een bepaald service−element in de toekomst gemaakt. Dit is nodig voor capaciteitsplanning op de lange termijn of eventuele signalering van problemen of tekorten op de korte termijn. Trendanalyse legt niet al te strikte eisen aan accounting, omdat deze meestal niet real−time hoeft te worden uitgevoerd en omdat er slechts een algemeen beeld van de gegevens nodig is. Hiervoor kan vaak een eenvoudige vorm van logging al volstaan. Voor veel organisaties is het doen van trendanalyse van groot belang.

Auditing

Auditing heeft betrekking op het volgen van activiteiten van gebruikers. Met auditing kunnen bijvoorbeeld misbruik van middelen en diensten of zwakheden in de beveiliging worden opgespoord. Met auditing kan worden nagegaan of gebruikers zich aan de voorwaarden van hun gebruik houden. Er kunnen met behulp van policies grenzen worden gesteld aan gebruik van diensten, zowel over langere termijn als op korte termijn (piekbelasting). Verslaglegging alleen is niet voldoende om naleving van de policies af te dwingen. Er moet een (real−time) terugkoppeling plaatsvinden naar het autorisatie−gedeelte om gebruik van de dienst te kunnen beperken.

Billing

Billing is het opstellen van een rekening voor een client voor het gebruik van de gemeten dienst. Als er sprake is van een rekening die op basis van gebruik wordt vastgesteld (usage−sensitive billing) is het belangrijk om betrouwbare gegevens over het gebruik te hebben. Er moet ook tegenover de client te verantwoorden zijn dat de accounting−gegevens waarop de rekening is gebaseerd betrouwbaar zijn. Soms zijn er real−time billing−gegevens nodig als er sprake is van een beperkt krediet. De accounting−gegevens zelf bevatten echter geen prijsgegevens, omdat deze meestal niet bij het service−element beschikbaar zijn. Billing−gegevens zijn vaak afhankelijk van gegevens van de client en kunnen bijvoorbeeld met speciale kortingen of piek− en daltarieven werken. Omdat accounting−gegevens zich direct laten vertalen in opbrengsten stelt billing strenge eisen aan accounting met betrekking tot betrouwbaarheid, beveiliging en fraudegevoeligheid. Ook als er met een vast tarief (flat−rate billing) wordt gewerkt, kan accounting van belang zijn, om bijvoorbeeld te controleren of een gebruiker binnen de afgesproken limieten blijft.

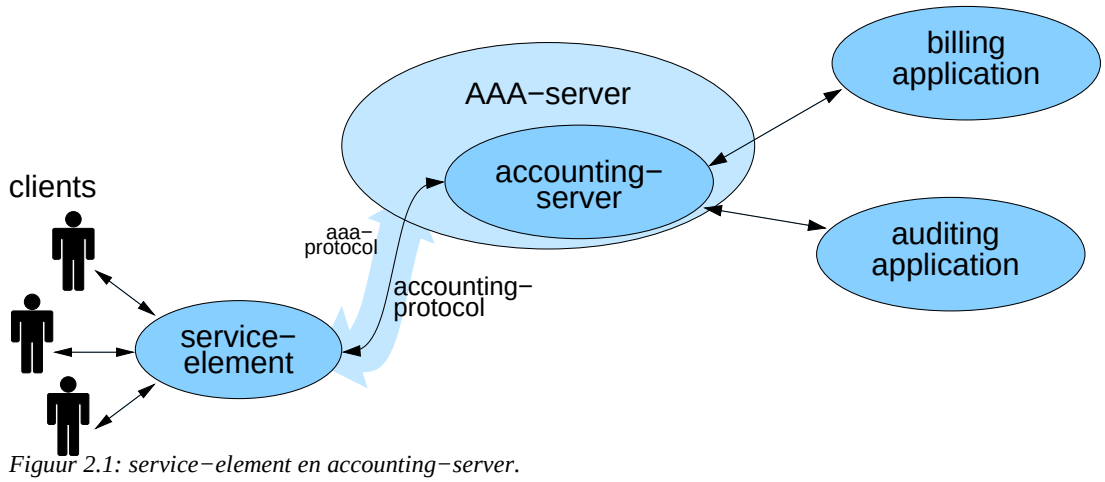
Cost allocation

Bij cost allocation worden accounting−gegevens gebruikt om gemaakte kosten te verdelen over projecten, afdelingen of zakenpartners die een gemeenschappelijke dienst gebruiken. Hierbij kunnen bijvoorbeeld de kosten voor telefoonverkeer worden verdeeld op basis van gemeten gebruik. Bij cost allocation is er, net als bij billing, een financieel belang, dat er voor zorgt dat er eisen moeten worden gesteld aan de fraudegevoeligheid. Cost allocation onderscheidt zich van billing, doordat hier geen sprake is van opbrengsten van het leveren van een dienst, maar van het intern verdelen van de kosten van gebruikte diensten. Usage−sensitive cost allocation kan tot een efficiënter gebruik van middelen leiden doordat kosten reëel worden teruggekoppeld.

Als er een zekere betrouwbaarheid nodig is van de accounting−gegevens, bijvoorbeeld als er een rekening op wordt gebaseerd is, voorafgaand aan de sessie, authenticatie van de gebruiker en autorisatie van de sessie wenselijk. Als er met beperkte kredieten wordt gewerkt, is er een directe terugkoppeling van de accounting naar de autorisatie nodig. Met deze terugkoppeling kan worden geregeld dat een client binnen een afgesproken gebruik van bepaalde middelen blijft.

2.1 Accounting−protocol

Accounting−gegevens worden met behulp van een accounting−protocol door het service−element aan een accounting−server doorgegeven. De accounting−server kan trendanalyse, auditing, billing of cost allocation uitvoeren of dit aan een gespecialiseerde server overlaten. De accounting−server is vaak onderdeel van een AAA−server die ook authenticatie en autorisatie uitvoert. Het accounting−protocol is daarom meestal onderdeel van een AAA−protocol.



Figuur 2.1: service−element en accounting−server.

De accounting−server heeft een verzamelende taak. Service−elements genereren in het algemeen bij begin en eind van het aanbieden van de dienst accounting−berichten, maar ook tijdens het aanbieden van de dienst kunnen zogenaamde interim accounting−berichten worden verstuurd. Accounting−servers kunnen start−, stop− en interim−berichten die tot dezelfde sessie behoren samennemen om accounting−records te maken. Zo kunnen bepaalde accounting−attributen worden gesommeerd.

Aan de hand van de accounting−records kan trendanalyse, auditing, billing en cost allocation worden uitgevoerd. Hiermee wordt inzicht gegeven in het gebruik en eventuele kosten. Deze gegevens kunnen ook voor de gebruiker van de dienst van belang zijn. Die kan bepalen in welke mate veranderingen in zijn structuur van invloed zijn op het gebruik van de dienst. Zo kan hij de kosten terugbrengen. De accounting−records kunnen dan verder worden opgeslagen of getransporteerd. Accounting−records worden bijvoorbeeld door een billing server gebruikt om rekeningen op te stellen.

Een accounting−protocol is in het algemeen op te delen in een transportprotocol, een recordformat en een verzameling mogelijke berichten.

Transportprotocol

Een transportprotocol draagt zorg voor het vervoer van accounting−berichten over een netwerk. Hierbij spelen zaken als betrouwbaarheid en beveiliging van de gegevens vaak een belangrijke rol. Het transportprotocol kan een verbinding tussen service−element en accounting−server opzetten of enkele berichten doorgeven tussen beide.

Recordformat

Een recordformat bepaalt op welke manier de accounting−gegevens in berichten worden vastgelegd. Het recordformat regelt de vertaling tussen een bericht en binaire data. Deze vertaalde berichten kunnen dan met behulp van het transportprotocol over een netwerk worden vervoerd of op een medium worden opgeslagen.

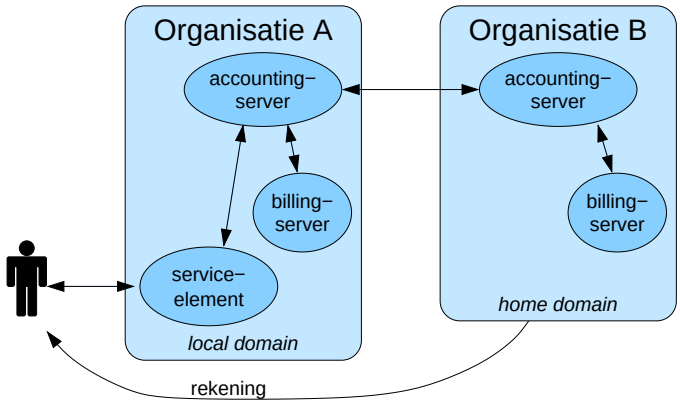
Verzameling berichten

De verzameling berichten definieert de verschillende soorten accounting−berichten die kunnen worden verstuurd. Het gaat hier dan om berichten die het begin of het einde van de afname van een dienst signaleren, bevestigingen van deze berichten en dergelijke. Er wordt ook gedefinieerd hoe berichten worden uitgewisseld en hoe antwoorden op berichten dienen te worden verstuurd.

Het accounting−protocol als geheel bevindt zich binnen het OSI−model op de applicatie−laag. Binnen het accounting−protocol definieert het transportprotocol welke eisen aan de transportlaag in het OSI−model worden gesteld. Zo wordt een accounting−protocol opgebouwd uit een aantal deelprotocollen. De scheiding in trasnportprotocol, recordformat en verzameling berichten is meestal wel terug te vinden in de bestaande accounting−protocollen, maar wordt vaak niet zo expliciet aangegeven. Deze opdeling is niet alleen van toepassing voor accounting, maar kan in het algemeen ook worden gemaakt voor het gehele AAA−protocol, waarvan het accounting−protocol een onderdeel is. Bij authenticatie en autorisatie zal dan in het algemeen hetzelfde transportprotocol en recordformat worden gebruikt, terwijl elk onderdeel zijn eigen verzameling berichten heeft.

2.2 Inter−domain accounting

Bij het aanbieden van diensten is het voor de aanbieder vaak belangrijk om accounting toe te passen. Als er echter nog een organisatie belang heeft bij de dienst kan het voorkomen dat ook deze inzicht wil hebben in de accounting−gegevens. Dit gebeurt bijvoorbeeld als verschillende organisaties samenwerken om een dienst aan te bieden. Een client kan dan diensten afnemen van een local domain (zie *Figuur 2.2*), terwijl hij een zakelijke relatie heeft met zijn home domain. Hierbij levert het local domain de dienst voor het home domain aan de client. De client krijgt uiteindelijk de rekening van de organisatie die het home domain beheert. Zo lijkt het voor de client alsof hij direct van zijn home domain diensten afneemt. In het voorbeeld in *Figuur 2.2* speelt organisatie A de rol van local domain en organisatie B de rol van home domain.



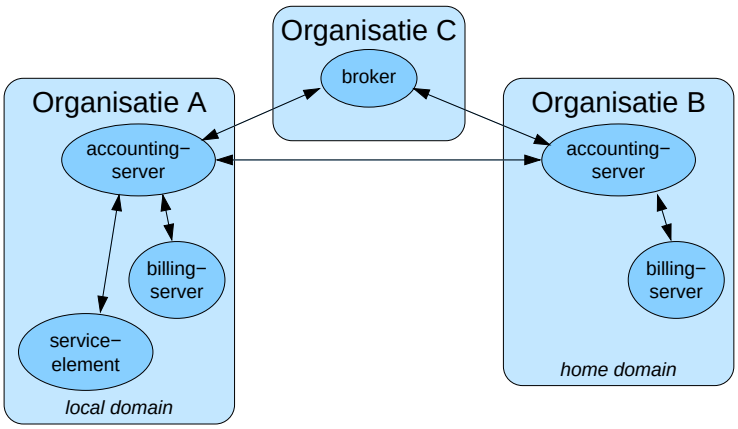
Figuur 2.2: inter−domain accounting.

Het local domain heeft over het algemeen niet voldoende directe gegevens over een client om te weten bij welk home domain deze client behoort. De client moet in de authenticatie fase genoeg informatie verschaffen, zodat de AAA−server van het local domain weet waar de AAA−berichten heen moeten. Dit kan bijvoorbeeld gebeuren met behulp van de Network Access Identifier (NAI) [rfc2486] die de gebruiker en het home domain identificeert.

Bij inter−domain accounting worden accounting−gegevens tussen verschillende organisaties uitgewisseld. Bij intra−domain accounting blijven de accounting−gegevens binnen dezelfde organisatie. De accounting−gegevens worden bij inter−domain accounting via een accounting−server in het local domain doorgespeeld naar de accounting−server in het home domain. De accounting−server in het local domain werkt dan als proxy voor het service−element. Een proxy kan zowel als client als als server optreden. Als de proxy in de rol van server een bericht ontvangt, kan het dit in de rol van client aan een andere server doorsturen. Een proxy kan op deze manier gebruikt worden om accounting−berichten te routeren door een netwerk. Als proxies gebruikt worden, hoeft het service−element alleen op de hoogte te zijn van de lokale accounting−server. De informatie die nodig is om een accounting−pakket te bezorgen hoeft dan niet in het service−element aanwezig te zijn.

Het kan nodig zijn dat de accounting−server van het local domain extra informatie, zoals extra kosten voor het inter−domain gebruik, aan de accounting−gegevens toevoegt. Ook blijven de accounting−gegevens interessant binnen het local domain, als organisatie A bijvoorbeeld een rekening aan organisatie B wil sturen voor het gebruik door klanten van organisatie B.

Omdat accounting−gegevens mogelijk over een open netwerk gaan of door meerdere proxies worden doorgegeven, stelt dit extra eisen aan de beveiliging. Het mag niet zo zijn dat een tussenliggend element kan frauderen door gegevens te veranderen. Ook is het meestal nodig om de gegevens tegen afluisteren te beschermen. Om de communicatie tussen de twee organisaties te beveiligen moet encryptie worden toegepast. Om het uitwisselen van de sleutels die voor de encryptie nodig zijn te vereenvoudigen, kan gebruik worden gemaakt van een broker. Deze broker (zie *Figuur 2.3*), die bij een trusted third party behoort, bemiddelt in de communicatie tussen de twee accounting−servers. De gehele communicatie kan via de broker verlopen, waarbij de broker dus als proxy optreedt. De broker kan ook slechts bemiddelen in het opzetten van een beveiligd kanaal tussen de twee accounting−servers, bijvoorbeeld door het uitwisselen van sleutels voor encryptie. De broker wordt als onafhankelijke partij in de bemiddeling gebruikt.



Figuur 2.3: gebruik van een broker.

Door gebruik te maken van een broker hoeven de individuele organisaties niet meer allemaal onderling overeenkomsten hebben, maar kan elke organisatie één overeenkomst met een gezamenlijke broker hebben. Dit vereenvoudigt de administratie van de organisaties, zodat ze over een groot gebied hun diensten kunnen leveren.

Als inter−domain accounting wordt toegepast stelt dit extra eisen aan het transportprotocol. Intra−domain accounting−berichten worden over het algemeen over een klein lokaal beheerd netwerk getransporteerd, terwijl inter−domain berichten over een groter netwerk (bijvoorbeeld het Internet) kunnen lopen. Daarbij zijn verschillende organisaties verantwoordelijk voor het beheer. Voor het transport zijn andere timeout−parameters nodig, er moet rekening worden gehouden met een aanzienlijke packet−loss en de doorlooptijden van berichten kunnen sterk fluctueren. Ook moet flow−control worden geregeld, zodat het lokale netwerk niet wordt overspoeld met berichten die het achterliggende netwerk niet kan verwerken.

2.3 Accounting−records

Accounting−berichten worden per sessie gegroepeerd tot accounting−records. In een accounting−record staat een overzicht van de zaken die tijdens de dienst zijn gemeten, zoals identiteit van de client, duur van de sessie, getransporteerde hoeveelheid gegevens, etc. Deze accounting−records bevatten alle relevante informatie voor billing, auditing en andere doeleinden.

Er zijn verschillende vormen van dienstverlening mogelijk. Als eerste kan dienstverlening bestaan uit een enkele gebeurtenis. Een voorbeeld hiervan is het opvragen van een www−pagina. Hierbij wordt per gebeurtenis over het algemeen één accounting−record gegenereerd. Ook kan de dienstverlening bestaan uit een sessie van een bepaalde duur. Het bieden van een inbelverbinding is een voorbeeld van een sessie. Een sessie heeft een begin− en eindtijdstip en een (beperkte) duur. Per sessie wordt in het algemeen één enkel accounting−record gegenereerd. Een sessie kan ook erg lang duren. Als bijvoorbeeld een vaste verbinding wordt gemaakt kan deze verbinding in principe oneindig lang duren. Om billing op basis van accounting−records uit te kunnen voeren, moeten dan ook voor delen van de sessie accounting−records beschikbaar zijn.

Bij eenvoudige dienstverlening is het dus voldoende om een enkel accounting−record per sessie te genereren. Als de dienstverlening of de prijs van de dienstverlening tijdens de sessie verandert, bijvoorbeeld omdat er sprake is van piek− en daluren, dan is het nodig om het gebruik in deze afzonderlijke periodes in aparte accounting−records vast te leggen. Het is dan bijvoorbeeld nodig om te weten hoeveel bytes verzonden zijn in de piekuren en hoeveel in de daluren om een juiste rekening op te kunnen stellen.

Ook als de gebruiker van de dienst, tijdens een enkele sessie, gebruik maakt van verschillende service−elements, waarvoor verschillende tarieven gelden, kan het nodig zijn om hiervoor verschillende accounting−records te genereren. Dit alles zorgt ervoor dat het nodig kan zijn om een enkele sessie met meerdere accounting−records te beschrijven.

2.4 Berichtenmodel

Accounting−berichten kunnen op verschillende manieren van een service−element naar een accounting−server worden getransporteerd. Accounting−berichten worden over het algemeen gegenereerd op basis van events. Deze events treden bijvoorbeeld op bij het begin en einde van een sessie, maar ook gedurende de sessie, bijvoorbeeld na het aflopen van een timer, kan een event optreden. Het model dat wordt gebruikt om de berichten daadwerkelijk te transporteren hangt af van de eisen die aan het accounting−systeem worden gesteld.

Archival accounting

Bij archival accounting gaat het om het verzamelen van alle accounting−gegevens. Bij een systeemfout moeten alle accounting−gegevens zo goed mogelijk te reconstrueren zijn. Verder worden accounting−gegevens ook nadat ze zijn verwerkt voor bijvoorbeeld billing gearchiveerd voor een bepaalde tijd. Juridische en financiële belangen kunnen archival accounting voorschrijven en opleggen dat accounting−gegevens vertrouwelijk worden behandeld. Dit kan bijvoorbeeld optreden bij usage−sensitive billing, waarbij het verlies van accounting−gegevens in het algemeen verlies van inkomsten betekent.

Ook bij uitval van service−elements, accounting−servers, netwerkstructuur of andere onderdelen van het accounting−systeem, moeten geen of zo min mogelijk accounting−gegevens verloren gaan. Dit betekent dat er op deze elementen bufferruimte aanwezig moet zijn waarop ook na stroomuitval gegevens bewaard blijven. Hierin moeten de accounting−gegevens tijdelijk worden opgeslagen. Archival accounting legt dus erg strikte eisen aan een accounting−systeem.

Real−time accounting

Bij real−time accounting is het doel accounting−gegevens binnen een bepaald tijdsbestek te verwerken. Deze tijdslimiet wordt in het algemeen opgelegd om financiële risico’s te vermijden of om real−time auditing uit te kunnen voeren. Een snelle terugkoppeling van gegevens over gebruik kan bijvoorbeeld nodig zijn om, bij overschrijding van een kredietlimiet de dienstverlening snel te staken.

Real−time accounting eist dat het transportprotocol de berichten snel kan versturen, dat de accounting−server snel accounting−gegevens aan een billing of auditing server doorstuurt en dat de billing of auditing server deze gegevens snel verwerkt en hier eventueel actie op onderneemt.

Batch accounting

Soms is het nodig om een grote hoeveelheid accounting−gegevens in één keer te versturen. Dit kan zo zijn als accounting−gegevens voor een off−line audit worden getransporteerd, maar ook als het service−element accounting−gegevens buffert. Redenen voor buffering kunnen zijn het niet bereikbaar zijn van de accounting−server of een efficiënter gebruik van netwerkbandbreedte. Met gebufferde gegevens kan een betere compressie worden behaald en buffering vermindert overhead.

Aan de hand van benodigde eigenschappen en eisen van een accounting−systeem kan een model voor berichtbezorging worden gekozen. Het versturen van accounting−berichten kan op een aantal manieren gebeuren. De eerste mogelijkheid is dat bij het optreden van een event accounting−gegevens vanuit het service−element aan de accounting−server worden doorgegeven. De tweede is dat een accounting−server met enige regelmaat alle service−elements vraagt of er nog accounting−gegevens zijn te versturen. Ook is het mogelijk om beide methoden te combineren. Elk heeft zo zijn voor− en nadelen.

Event−driven model

In het eenvoudigste geval worden de accounting−gegevens door het service−element aan de accounting−server verstuurd als deze beschikbaar zijn. Er treed een push van accounting−gegevens op. Met dit model zijn accounting−gegevens het snelst op de plaats van bestemming. Als er sprake moet zijn van real−time accounting is dit in veel gevallen de beste oplossing.

Op het service−element is weinig bufferruimte nodig, omdat accounting−gegevens maar erg kort (totdat er een bevestiging is teruggekomen) hoeven te worden bewaard. Een probleem is dat er voor deze vorm relatief veel netwerkbandbreedte nodig is en dat het gebruik van de netwerkbandbreedte moeilijk te voorspellen is. De benodigde hoeveelheid verbindingen, en daarmee de bandbreedte, schaalt met het aantal events dat optreedt.

Binnen deze aanpak is het mogelijk om bandbreedte efficiënter te gebruiken door batching toe te passen. Er wordt dan gewacht tot een buffer voor een bepaalde hoeveelheid is gevuld. Deze variant wordt ook wel event−driven−batching genoemd.

Polling model

Het is ook mogelijk om de accounting−server alle aangesloten service−elements af te laten gaan om te vragen of er accounting−gegevens beschikbaar zijn. Dit wordt ook wel polling genoemd. Het voordeel van deze aanpak is dat er geen verbinding per event wordt opgezet, maar één verbinding per service−element. Ook is het mogelijk om polling zo in te delen dat het netwerk niet in één keer wordt overbelast met allemaal accounting−berichten. Verder leent dit model zich beter voor batching, waarmee de efficiëntie van het transport kan worden verhoogd. Een nadeel van deze aanpak is dat er op het service−element een aanzienlijke hoeveelheid bufferruimte nodig is om de accounting−gegevens op te slaan. Verder is het toepassen van polling in een netwerk met veel service−elements die niet allemaal regelmatig accounting−gegevens produceren inefficiënt. Zo is deze oplossing praktisch niet toe te passen bij inter−domain accounting.

Event−driven polling model

Het is ook mogelijk om een hybride oplossing te kiezen: event−driven polling. Hierbij stuurt een service−element één keer een klein bericht aan een accounting−server dat het accounting−gegevens beschikbaar heeft, waarna de accounting−server deze gegevens in zijn eigen tempo ophaalt. Deze aanpak heeft de voordelen van polling waarbij scheduling van accounting−transport kan worden toegepast, terwijl alleen die service−elements die accounting−gegevens beschikbaar hebben hoeven te worden gepolled. Het is efficiënter dan het event−driven model, omdat het service−element maar één keer een bericht aan de accounting−server hoeft te sturen. Event−driven polling is in dit geval wel mogelijk in een inter−domain accounting omgeving. Voor real−time accounting blijft echter vaak een pure event−driven aanpak nodig, omdat daarmee de gegevens het snelst op de plaats van bestemming zijn.

2.5 Beveiliging

Bij het transport van accounting−gegevens over een netwerk komt beveiliging kijken. Accounting−gegevens kunnen gevoelige informatie bevatten, waar financiële belangen mee zijn gemoeid. Ook kan het zijn dat accounting−berichten privacy−gevoelige informatie bevatten. Zeker als accounting−berichten over een groter netwerk lopen is het gevaar van het onderscheppen of zelfs wijzigen van accounting−berichten groot. In het algemeen moet een accounting−protocol bescherming bieden tegen:

- het veranderen van accounting−gegevens,
- het verwijderen van accounting−gegevens,
- het toevoegen van valse accounting−gegevens en
- het ongeoorloofd lezen van accounting−gegevens.

Door het veranderen van de gegevens over de sessie ontstaat een verkeerde indruk van het verloop van de sessie. Ook als een gebruiker kan zorgen dat accountig berichten die bij zijn sessie horen niet aankomen, levert dit voor de aanbieder van de dienst een verkeerd beeld op. Verder kan bijvoorbeeld door het genereren van valse accounting−berichten een te hoge rekening aan gebruikers worden gestuurd.

Om in de beveiliging van accounting−gegevens te voorzien, moeten in een accounting−systeem bepaalde voorzorgsmaatregelen worden getroffen. Deze voorzieningen zijn integrity protection, authenticatie, confidentiality protection, replay protection en non−repudiation. Deze zaken worden in het accounting−protocol geregeld met behulp van encryptie.

Integrity protection

Er moet kunnen worden geverifieerd of het accounting−bericht ongewijzigd is aangekomen. Deze beveiliging moet beter zijn dan de beveiliging tegen bitfouten die in het transportprotocol aanwezig is. Ook opzettelijk veranderde accounting−gegevens moeten gedetecteerd kunnen worden. Als accounting−gegevens via proxies of brokers worden doorgegeven moet de uiteindelijke ontvanger kunnen nagaan welke gegevens van welke instantie afkomstig zijn en of ze daarna niet ongewenst zijn veranderd.

Authenticatie

Beide partijen moeten zeker zijn van wie een bericht afkomstig is en wie het ontvangen heeft. Daarom is het nodig om wederzijdse authenticatie uit te voeren. Voor de accounting−server is het belangrijk te weten dat de accounting−gegevens van een legitieme client (service−element) afkomstig zijn en voor het service−element is het belangrijk te weten of de accounting−berichten door een bekende accounting−server worden verwerkt.

Confidentiality protection

Confidentiality protection houdt in dat berichten die zijn bestemd voor een bepaalde ontvanger niet door anderen kunnen worden gelezen. Sommige gegevens in accounting−berichten zijn van belang voor proxies om bijvoorbeeld de goede ontvanger van het accounting−bericht te bepalen. Andere gegevens moeten dusdanig worden beschermd dat alleen de uiteindelijke ontvanger ze kan lezen.

Replay protection

Het mag niet mogelijk zijn dat een eerder verstuurd accounting−bericht of een aanpassing daarvan nogmaals wordt geaccepteerd. Dit beveiligt tegen het opnieuw genereren van accounting−gegevens, dat dubbele billing zou kunnen veroorzaken.

Non−repudiation

Non−repudiation wil zeggen dat geen van de partijen betrokken in de communicatie later kan ontkennen dat een bepaald bericht is verzonden of ontvangen. Het service−element kan later niet ontkennen dat het accounting−bericht is verstuurd en de accounting−server kan later niet ontkennen dat hij het bericht heeft ontvangen. Dit is vooral van belang bij inter−domain accounting.

De beveiliging kan end−to−end en hop−by−hop worden toegepast. Bij end−to−end security worden berichten beveiligd tussen het element dat de berichten genereert (het service−element, eventueel accounting−server in het local domain) en het element dat de berichten uiteindelijk ontvangt en verwerkt (accounting−server in het home domain). By hop−by−hop security wordt de beveiliging alleen geregeld tussen de elementen in de communicatie onderling. Dit betekent dat proxies ook accounting−gegevens kunnen inzien en eventueel aanpassen. Het is duidelijk dat bepaalde delen van accounting−berichten end−to−end beveiliging vereisen en andere hop−by−hop.

Het is nodig om de verschillende niveaus van beveiliging per attribuut te kunnen gebruiken. Het is bijvoorbeeld mogelijk dat bepaalde accounting−gegevens alleen voor lokaal gebruik zijn bestemd en niet voor een home server van een ander domein. Ook is het mogelijk dat berichten via untrusted proxies gaan, die de accounting−berichten alleen hoeven te bezorgen. De bestemming van het accounting−bericht moet wel door de proxy zijn te bepalen.

Accounting−berichten hoeven niet altijd te worden gecodeerd. Als er geen strikte eisen zijn aan de beveiliging vanuit een oogpunt van financieel risico dan kan encryptie achterwege blijven. Dit is bijvoorbeeld het geval als accounting−gegevens alleen over een lokaal afgeschermd netwerk gaan en alleen worden gebruikt voor trendanalyse.

2.6 Toepassingen

Hier zullen enkele toepassingen van accounting worden behandeld met als doel het inzicht geven in de soorten diensten waarbij accounting wordt toegepast. Veel van de hier behandelde toepassingen stellen verschillende eisen aan accounting. Vaak is er sprake van een financieel belang, dat er voor zorgt dat accounting zeer zorgvuldig moet gebeuren. Niet voor alle toepassingen zijn trendanalyse, auditing, billing en cost allocation mogelijk of vanzelfsprekend. In dit verband wordt logging gezien als een eenvoudige vorm van accounting. In veel systemen is het niet vanzelfsprekend om een logging−systeem te vervangen door een volledig accounting−systeem, maar als deze gegevens, eventueel gecombineerd met andere gegevens, voor billing−doeleinden worden gebruikt kan een accounting−systeem wel voordelen opleveren.

2.6.1 Inbellen

Internet Service Providers (ISP’s), ook wel Internet Access Providers (IAP’s) genoemd, bieden inbelfaciliteiten aan. Hiermee kan via de telefoon of kabel een verbinding met het Internet worden gemaakt. Voor het leveren van inbelfaciliteiten is een overvloed aan producten (Network Access Servers) te koop. De meeste van deze producten ondersteunen het RADIUS (Remote Authentication Dial In User Service) protocol [rfc2865]. Daarnaast worden vaak diverse andere protocollen ondersteund. Bij inbellen zijn authenticatie en autorisatie van groot belang. De meeste van de protocollen voor Network Access Servers zijn dan ook authenticatie en autorisatie protocollen waaraan later een accounting−deel is toegevoegd.

Bij inbellen is accounting zeer belangrijk. Bij een aantal ISP’s wordt usage−sensitive billing toegepast, maar ook bij de ISP’s die met flat−rate billing werken, is het belangrijk trendanalyse en auditing uit te voeren. Inbellen (network access) is momenteel het belangrijkste aandachtsgebied waarvoor AAA−applicaties worden ontwikkeld.

2.6.2 Telefonie

In de telefonie wordt accounting al erg lang toegepast. De rekening die een telecomprovider maakt is gebaseerd op specifiek gebruik van telefoonnummers in bepaalde regio’s. Vroeger werden gegevens over gevoerde telefoongesprekken via papier en later via magneetbanden vervoerd, waarna ze werden verwerkt. Tegenwoordig is het proces geheel geautomatiseerd.

In de telefonie worden vaak ingewikkelde billing strategieën gebruikt, met bijvoorbeeld een aantal door de klant gekozen nummers waarmee tegen een gereduceerd tarief kan worden gebeld. Ook is het mogelijk om een zeer gespecificeerd verslag te maken met een overzicht van welke nummers wanneer en hoe lang zijn gebeld. Voor dit alles is een zeer betrouwbaar accounting−systeem nodig.

Bij mobiele telefonie wordt het verzamelen van accounting−gegevens ingewikkelder, omdat de client kan bewegen en van verschillende service−elements gebruik kan maken om een enkele sessie (gesprek) te voeren. Dit gebeurt als een client van een cel naar een andere cel beweegt.

2.6.3 Telefooncentrale

Veel bedrijven hebben intern een PBX (Private Branch Exchange). Deze kan worden gebruikt voor intern telefoonverkeer, maar ook voor telefoonverkeer naar buiten. Om na te gaan of de buitenlijn niet wordt misbruikt, om kosten van gebruik van het systeem op de juiste afdeling of op het juiste project te krijgen of om de rekening van de telecomaanbieder te controleren [billaudit] kan hier accounting worden toegepast.

Veel telefooncentrales kunnen Call Detail Records genereren en die met SMDR (Station Message Detail Recording) transporteren. Hiermee kunnen van inkomende en uitgaande telefoongesprekken gegevens worden bijgehouden. Bijgehouden gegevens zijn onder andere het nummer waar vandaan wordt gebeld en waar naartoe wordt gebeld, datum, tijd en duur van het gesprek en eventueel informatie om het betrokken project of de klant te identificeren. Met behulp van een PC kunnen deze gegevens dan worden verwerkt om overzichten te genereren.

2.6.4 Interne netwerkelementen

Van interne netwerkelementen zoals routers, firewalls en dergelijke is het vaak belangrijk inzicht te krijgen in het gebruik ervan [rfc1272]. Het gaat hier echter vaak om grote hoeveelheden enkele events. Vaak is het onmogelijk om hier erg gedetailleerde accounting toe te passen, omdat dan voor elk binnenkomend bericht een accounting−bericht zou moeten worden gegenereerd. Dit betekent bijvoorbeeld bij routers een verdubbeling van het netwerkverkeer. Bij routers is het wel belangrijk om een globaal inzicht te hebben in het netwerkverkeer, dit met het oog op trendanalyses voor de netwerkstructuur. Hier worden dan vaak totalen van verzonden en ontvangen pakketjes en bytes geteld. Bij routers en andere netwerkelementen wordt meestal gebruik gemaakt van SNMP [snmpintro] om de accounting−gegevens te transporteren.

Bij firewalls wordt vaak van een beperkte hoeveelheid transacties relatief uitvoerig bericht. Omdat firewalls voor beveiliging van een netwerk zorgen is auditing van dit verkeer natuurlijk belangrijk. Hetzelfde geldt voor border−routers die twee netwerken van verschillende organisaties met elkaar verbinden.

2.6.5 Hosting

Hosting is het aanbieden van computercapaciteit voor bedrijven die het systeembeheer willen uitbesteden. Bedrijven die zelf geen snelle Internet−verbinding hebben of het beheer van systemen buiten de eigen organisatie willen houden kunnen gebruik maken van hosting. Hierbij kan worden gedacht aan het draaien van databases of het bieden van specifieke Internet−diensten. Bij hosting komen ook zaken kijken als dag tot dag systeembeheer, installatie van systeemupdates, beveiliging, onderhoud aan databases of andere zaken waarbij specifieke technische kennis of middelen nodig zijn.

Bij hosting wordt over het algemeen een afgesproken hoeveelheid middelen gereserveerd en verhuurd. Dit kan bijvoorbeeld door het plaatsen van een computersysteem van de klant in het netwerk van de aanbieder. De klant kan dan met het systeem de aanwezige resources van het netwerk gebruiken.

Aan hosting hangt natuurlijk een prijskaartje. Het is daarom wenselijk om inzicht te krijgen in het gebruik van de aangeboden middelen en de kosten van onderhoud. Ook bij flat−rate billing is het belangrijk om dit inzicht te hebben, zodat met behulp van trendanalyse een capaciteitsplanning kan worden uitgevoerd. Ook kan het nodig zijn om auditing uit te voeren om te controleren of aan de afgesproken voorwaarden voor het gebruik wordt voldaan.

Deze vorm van dienstverlening vereist een andere methode van accounting dan bijvoorbeeld bij inbellen wordt gebruikt. Er is geen sprake van korte sessies waarover wordt afgerekend. De sessies bij hosting duren in het algemeen erg lang (in de orde van maanden of jaren) en hoeven niet te eindigen. Bij hosting en soortgelijke dienstverlening is het dus nodig om op basis van delen van de sessie billing uit te voeren. Een bijkomend probleem is dat er bij hosting over het algemeen veel verschillende soorten resources worden gebruikt. Om het gebruik van al deze resources te kunnen combineren moet het accounting−systeem daar ingewikkelde accounting−records voor kunnen ondersteunen.

2.6.6 Website hosting

Website hosting is een voorbeeld van hosting. Bij website hosting wordt schijfruimte op een webserver verhuurd voor het plaatsen van www−pagina’s. Bedrijven die zelf geen snelle Internet−verbinding hebben of het beheer buiten de eigen organisatie willen houden kunnen hiervan gebruik maken. Ook is deze dienst vaak beschikbaar voor particulieren die een persoonlijke homepage willen publiceren. Internet−providers bieden hun klanten, naast een Internet−verbinding vaak ruimte op een webserver aan.

Bij website hosting is het vaak belangrijk inzicht te krijgen in de frequentie waarmee de pagina's worden opgevraagd en in de totale hoeveelheid data die wordt verstuurd. Voor de klant kan het belangrijk zijn om te weten welke onderdelen van de website regelmatig worden opgevraagd. Dit is het geval als op websites reclame wordt gemaakt waarmee, afhankelijk van de hoeveelheid mensen die de pagina bezoeken, geld wordt verdiend.

De logfiles van de webserver geven over het algemeen voldoende inzicht in de benodigde gegevens. Het is ook mogelijk om de webserver logs met behulp van een accounting−systeem te verwerken. Dit heeft als voordeel dat de gegevens eenvoudiger kunnen worden gekoppeld met andere gegevens die met behulp van accounting worden verzameld.

2.6.7 Roaming

Roaming [rfc2194] is het gebruik van diensten bij een andere provider dan waarmee een overeenkomst bestaat. Dit kan bijvoorbeeld gebeuren bij inbellen waarbij een client is aangesloten bij een Internet−provider (home ISP) en voor het inbellen gebruik maakt van de diensten van een lokale Internet−provider (local ISP). Zo kunnen verschillende Internet−providers samenwerken om gezamenlijk een groter gebied te bestrijken. Deze afspraak wordt een roaming−agreement genoemd. Bij roaming is sprake van inter−domain accounting.

Bij roaming worden de gegevens over de gebruiker in het algemeen bij de home ISP opgeslagen. Bij het inloggen (bij de local ISP) wordt de authenticatie en autorisatie mede door de home ISP geregeld. Accounting−gegevens zijn voor zowel home ISP als local ISP belangrijk. Voor de home ISP worden accounting−gegevens gebruikt om klanten rekeningen te kunnen sturen en voor de local ISP worden de accounting−gegevens gebruikt om de home ISP eventueel een rekening te sturen.

Bij roaming worden accounting−gegevens over grote afstand getransporteerd. Omdat hierbij financiële belangen een rol spelen vereist deze vorm van accounting een grote betrouwbaarheid. Accounting−gegevens moeten bijvoorbeeld niet door de local ISP kunnen worden vervalst. Omdat de gegevens door de local ISP worden gegenereerd moet de home ISP de uiteindelijke gebruiker kunnen authenticeren en er zeker van zijn dat de accounting−gegevens echt zijn. De local ISP moet aan de andere kant de zekerheid hebben dat de home ISP de kosten voor de verbinding accepteert. Beide ISPs moeten later niet kunnen ontkennen dat de accounting−gegevens daadwerkelijk zijn verstuurd.

Omdat het niet haalbaar is alle ISP's met alle andere ISP's roaming agreements te laten maken, kunnen brokers worden gebruikt waarmee een roaming consortium kan worden gevormd. De brokers handelen dan als onafhankelijk tussenpersoon en kunnen zaken als beveiliging regelen.

2.6.8 Mobile IP

Met mobile IP [mobiptut] worden uitbreidingen aan bestaande methoden voor routering gespecificeerd, waarmee het mogelijk is om IP verkeer te routeren tussen bewegende hosts en statische hosts. Een bewegende host blijft geïdentificeerd met een vast adres, terwijl de huidige positie en de huidige verbinding met het Internet kan veranderen.

Hiermee is het mogelijk om een mobile Internetverbinding op te zetten, zonder dat de netwerkconfiguratie van de mobile host of de statische host aangepast hoeven te worden.

Mobile IP is bij uitstek geschikt om te combineren met roaming. Hierbij is accounting natuurlijk ook van belang. De accounting−gegevens van een enkele sessie kunnen dan van verschillende local domains afkomstig zijn.

2.6.9 Resource reservation

RSVP (Resource ReserVation Protocol) kan worden gebruikt om Quality of Service (QoS) verbindingen te leveren op Internet. Internet is een best−effort netwerk waarbij geen garanties kunnen worden gegeven over de beschikbare bandbreedte en response−tijd voor een verbinding. Met RSVP kan een hoeveelheid bandbreedte of nodige response−tijd van het netwerk worden gereserveerd. Multimedia−applicaties kunnen dit bijvoorbeeld nodig hebben voor streaming. RSVP is gedefinieerd in [rfc2205].

Bij RSVP wordt vanuit de ontvanger een verzoek gedaan om bandbreedte te reserveren vanaf een bepaalde zender [rsvpwww]. Onderweg worden verschillende reserveringen samengevoegd. RSVP gebruikt de aanwezige routering in Internet en verzorgt zelf geen transport. Het verzorgt alleen de reservering. Het is ontworpen om multicast verbindingen te ondersteunen, zoals radio uitzendingen vanuit een enkele bron naar een groot aantal ontvangers.

Bij het bieden van een gegarandeerde bandbreedte komt accounting vanzelfsprekend om de hoek kijken. Diensten met een gegarandeerde kwaliteit hebben over het algemeen een prijskaartje. In [rsvpacc] wordt een opzet voor het gebruik van accounting in RSVP beschreven. Hierin wordt echter geen accounting−protocol gepresenteerd waarmee dit is te verwezenlijken. De beschreven methode is gebaseerd op eenvoudige afspraken tussen netwerk−providers waarbij verkeer tussen de twee partijen wordt verrekend, zoals dat ook bij best effort verkeer gebruikelijk is. Gebruikers worden dan afgerekend op een deel best effort verkeer en een deel gereserveerd verkeer over het netwerk. Providers krijgen op hun beurt weer rekeningen in dezelfde vorm van de ISP waar het verkeer in eerste instantie naar toe gaat.

2.6.10 Overig

Naast de eerdergenoemde toepassingen zijn er nog een groot aantal gebieden waar accounting wordt of kan worden toegepast. Bij het gebruik van processortijd op een mainframe, printen via een centrale printerserver, gebruik van een fileserver waarbij schijfruimte wordt verhuurd, teleconferencing, pay−television, business email, fax en telefoon via IP en nog vele andere zaken kan accounting worden gebruikt.

Bij teleconferencing is het denkbaar dat er via verschillende media door verschillende personen en organisaties een sessie tot stand wordt gebracht. Deze verschillende media produceren naar alle waarschijnlijkheid voor alle aangesloten gebruikers (met verschillende geografische locaties) accounting−gegevens. Het is wenselijk om alle gegevens te kunnen groeperen en daar één rekening voor te sturen.

Ook zou een accounting−protocol wellicht kunnen worden gebruikt om structuur te brengen in logging bij FTP servers en andere soortgelijke diensten. Deze servers produceren grote logfiles met erg veel informatie. Om deze informatie op een eenvoudige manier te kunnen verwerken en te transporteren voor bijvoorbeeld remote auditing zou een accounting−protocol kunnen worden gebruikt. Ook zouden logfiles van verschillende servers centraal kunnen worden bijgehouden.

2.7 Overhead

Het genereren, transporteren en verwerken van accounting−gegevens behoort niet tot de dienstverlening zelf. Voor het uitvoeren van accounting worden ook resources gebruikt. Deze resources bestaan voornamelijk uit gebruik van bandbreedte op het netwerk, het gebruik van processortijd op service−elements en accounting−servers, gebruik van bufferruimte voor tijdelijke opslag van accounting−gegevens en het gebruik van permanente opslagruimte van accounting−gegevens voor het naderhand verwerken of het uitvoeren van auditing [ietf−aaa−acct]. Deze resources zijn overhead bij de dienstverlening.

Voor het transporteren van accounting−gegevens worden verbindingen op het netwerk tot stand gebracht. De hoeveelheid benodigde bandbreedte is evenredig met de hoeveelheid getransporteerde accounting−gegevens vermeerderd met de overhead voor transport. Omdat enkele accounting−berichten over het algemeen erg klein zijn loopt de overhead voor transport, als accounting−berichten individueel worden verzonden, aardig op. Het transporteren gebruikt niet alleen netwerkbandbreedte, maar er zijn voor een verbinding ook buffers en handles nodig. Het uitvoeren van batching en het gebruik van compressie kunnen de hoeveelheid gebruikte netwerkbandbreedte verminderen.

Het opslaan van accounting−gegevens die wachten op transport kost een hoeveelheid geheugenruimte op het service−element. De hoeveelheid ruimte hangt af van de transportmethode. Ook op de accounting−server is geheugenruimte nodig om ontvangen gegevens op te slaan. Ook als de accounting−server de gegevens direct doorstuurt, is er een bufferruimte nodig om gegevens te bewaren totdat ze zijn getransporteerd. Voor de uiteindelijke opslag van accounting−gegevens in een database of logfile is een grote hoeveelheid geheugenruimte nodig. Vaak moeten accounting−gegevens voor langere perioden worden bewaard. Het gebruik van compressie bij opslag van accounting−gegevens kan hierin een aanzienlijke besparing opleveren. Ook het in een vroeg stadium elimineren van dubbele gegevens (bijvoorbeeld het samennemen van bij elkaar horende interim berichten) kan een aardige besparing opleveren.

Het genereren en verwerken van accounting−gegevens op service−elements, accounting−servers en proxies kost een zekere hoeveelheid processortijd. Als compressie van accounting−gegevens wordt gebruikt kost dit eveneens processortijd. Om start−, stop− en interimberichten bij elkaar te zoeken is een hoeveelheid administratie van sessies op de accounting−server nodig. Accounting−gegevens moeten vaak worden beschermd tegen misbruik. Hiervoor worden beveiligingstechnieken toegepast. Het coderen van gegevens kost over het algemeen een aanzienlijke hoeveelheid resources. Tevens is het vaak nodig om sleutels tussen de verschillende partijen te distribueren.

Bij het gebruik van accounting moet rekening worden gehouden met de overhead die accounting met zich meebrengt. De hoeveelheid overhead hangt in het algemeen af van de detaillering en de betrouwbaarheid van de verslaglegging. Bij service−elements die snel veel diensten moeten kunnen leveren, zoals routers, kan het berekenen en vastleggen van detailinformatie aanzienlijke rekenkracht en netwerkbandbreedte vergen.

Bij het gebruik van accounting moet dus een afweging worden gemaakt tussen de opbrengsten van accounting (billing, auditing, etc.) en de kosten ervan (overhead). Bij deze afweging moet worden bepaald tot op welk niveau detailgegevens moeten worden bijgehouden. Het is mogelijk om, na inspectie, bepaalde detailgegevens te sommeren en slechts een beperkt aantal gegevens te bewaren.

3. Probleemstelling

In dit hoofdstuk wordt ingegaan op de tekortkomingen die er op het gebied van accounting momenteel zijn en waarmee deze tekortkomingen deels kunnen worden verholpen. Aangegeven wordt wat de behoeften zijn bij accounting−toepassingen en waar deze behoeften nog niet kunnen worden vervuld. Voor de diverse tekortkomingen in het gebied van accounting zal een doel geformuleerd worden waar naartoe gewerkt zal worden.

3.1 Probleemstelling

Er is een behoefte aan accounting−systemen en accounting−protocollen die in verschillende omgevingen toepasbaar zijn. Een algemeen toepasbaar accounting−systeem is bruikbaar voor meerdere vormen van dienstverlening. Door gebruik te maken van een algemeen geaccepteerd standaardprotocol voor het uitwisselen van accounting−gegevens is het mogelijk om deze gegevens van verschillende soorten diensten tussen verschillende organisaties uit te wisselen. Het is dan mogelijk om in zowel een inbelomgeving als in een application−hosting omgeving hetzelfde accounting−systeem en accounting−protocol te gebruiken.

Door de ontwikkeling van een algemeen toepasbaar accounting−systeem of de beschikbaarheid van een algemeen geaccepteerd standaard−protocol gaan de kosten voor beheer en ontwerp van accounting−systemen omlaag en wordt de flexibiliteit en portabiliteit van accounting−gegevens verhoogd. Het gebruik van een algemeen accounting−protocol of een uniform toepasbaar accounting−systeem maakt het bijvoorbeeld mogelijk om van verschillende vormen van dienstverlening eenvoudig een gecombineerde rekening op te stellen.

Accounting−systemen zijn momenteel vaak opgezet met een specifieke toepassing in gedachten. Deze accounting−systemen zijn dan slechts toepasbaar binnen dit voorgedefinieerde domein van omgevingen. Deze systemen zijn over het algemeen weinig flexibel, zodat het koppelen van accounting gegevens uit deze systemen met andere gegevens niet eenvoudig is. Het zou praktisch zijn als voor verschillende toepassingen eenzelfde accounting−systeem zou kunnen worden gebruikt. Een generieke manier om accounting−gegevens vast te leggen en te versturen zou ook voor billing−systemen erg praktisch zijn.

Bij inbel−toepassingen worden vaak RADIUS−systemen gebruikt, waarin het RADIUS−protocol wordt gebruikt om accounting−gegevens uit te wisselen. Voor deze toepassing is RADIUS in de praktijk als standaard geaccepteerd. Hierdoor zijn inbel−gegevens van verschillende organisaties over het algemeen redelijk uitwisselbaar. Ook kan RADIUS worden gebruikt bij andere vormen van remote login. Een probleem van RADIUS is dat leverancier−specifieke attributen niet zijn gestandaardiseerd. Hierbij komt dat door de beperkte verzameling van attributen en berichten RADIUS over het algemeen niet goed geschikt is voor andere vormen van dienstverlening. Ook schiet RADIUS tekort in een inter−domain omgeving, omdat het niet aan de hiervoor benodigde beveiligingsvoorzieningen voldoet.

Momenteel wordt een opvolger van RADIUS ontwikkeld, DIAMETER [diameter] genaamd. Dit protocol is beter uitgerust voor inter−domain accounting onder andere door betere beveiliging. Ook de verzamelingen attributen en berichten zijn beter uitbreidbaar. DIAMETER is echter nog steeds erg specifiek toegesneden op inbel−achtige omgevingen. Het protocol is wel beter uit te breiden, zodat het voor andere toepassingen kan worden gebruikt, maar hierop ligt binnen de ontwikkeling zeker niet de nadruk.

Er is een behoefte aan een eenvoudige manier om accounting−gegevens van verschillende toepassingen te transporteren. Hierbij is het van belang dat deze gegevens uitwisselbaar zijn tussen verschillende partijen. Voor dit probleem zijn verschillende oplossingen denkbaar. Het meest voor de hand ligt het ontwikkelen van een enkel gestandaardiseerd algemeen geaccepteerd accounting−protocol dat voor meerdere toepassingen bruikbaar is.

Een probleem bij het ontwikkelen van een accounting−protocol dat voor meerdere toepassingen bruikbaar is, is dat de requirements voor de verschillende toepassingen nogal uit elkaar liggen. Voor sommige toepassingen is real−time accounting nodig waarbij veel relatief eenvoudige accounting gegevens moeten worden getransporteerd. In andere situaties is er behoefte aan een accounting−systeem dat complexe accounting records kan verwerken, waarbij het een minder tijdkritische applicatie betreft. Dit maakt de ontwikkeling en acceptatie van een enkel algemeen toepasbaar accounting−protocol niet waarschijnlijk.

3.2 Doelstelling

Een accounting−framework, waarin verschillende accounting−protocollen kunnen worden gecombineerd zou een goed alternatief kunnen vormen voor een algemeen geaccepteerd standaardprotocol. Als het mogelijk is om diverse accounting−protocollen door elkaar te gebruiken, zonder dat voor een nieuw protocol veel aanpassingen aan het systeem nodig zijn, is het mogelijk om accounting−informatie van verschillende bronnen in een enkel systeem te verwerken. Afhankelijk van de toepassing en de daaraan gekoppelde eisen, kan uit een verzameling beschikbare protocollen het juiste protocol worden gekozen.

Bij eenvoudige diensten waarbij real−time accounting belangrijk is kan een DIAMETER−achtig protocol worden gebruikt. Bij complexere diensten, waarbij meer details over sessies nodig zijn, kunnen accounting−gegevens worden vastgelegd in een XML[xmlw3c]−recordformat en over een HTTP verbinding worden getransporteerd. Een framework met een dergelijke opzet is, door gebruik van de juiste deelprotocollen zowel toepasbaar bij real−time inter−domain usage−sensitive billing als bij offline intra−domain trendanalyse.

Het framework kan een eenvoudige interface bieden voor diverse accounting−toepassingen, waarbij implementatie−kwesties van het gebruikte accounting−protocol losgekoppeld kunnen worden van de uiteindelijke toepassing. Hierbij geschiedt het transport van accounting−gegevens op transparante manier.

Binnen het kader van dit afstudeerproject wordt er naar gestreefd, voor het ontwikkelen van een algemeen toepasbaar accounting−framework waarbinnen verschillende accounting−protocollen kunnen worden geïntegreerd. Het framework als geheel zou dan toepasbaar moeten zijn voor verschillende vormen van dienstverlening en het moet mogelijk zijn om meerdere accounting−protocollen te gebruiken. Er zal aandacht worden besteed aan beveiliging, aan de structuur van service−elements en accounting−servers en tevens aan algemene eigenschappen van accounting−protocollen en de uitwisseling van accounting−gegevens. Om het framework te testen zullen enkele accounting−protocollen worden geïmplementeerd die in het framework in te passen zijn.

4. Requirements

Het doel van het opstellen van requirements is het stellen van eisen aan de oplossing van het probleem. Bij het maken van requirements wordt het probleem verder geanalyseerd en wordt vastgesteld binnen welke grenzen het framework het gestelde probleem moet oplossen. De requirements bieden een leidraad bij het ontwerpen en implementeren van het framework en kunnen worden gebruikt om het uiteindelijke resultaat van zowel het ontwerp als de implementatie te toetsen.

In de requirements wordt aangegeven wat de functionaliteit van het framework moet zijn en in welke omgeving en met welke randvoorwaarden het framework moet functioneren.

4.1 Functionele requirements

De functionele requirements geven inzicht in de eisen die aan het algemeen functioneren van het framework zijn verbonden. Het geeft de belangrijkste eigenschappen van het framework weer zonder al te veel in detail te treden over de plaats van het framework.

Het doel van het framework is een eenvoudige interface te bieden waarmee, op een voor de gebruiker transparante manier, accounting−gegevens getransporteerd kunnen worden. Hierbij moeten bestaande en nieuwe accounting−protocollen kunnen worden gebruikt voor het transporteren van accounting−gegevens.

Verschillende diensten

Het framework moet toepasbaar zijn voor meerdere vormen van dienstverlening. Accounting moet mogelijk zijn bij dienstverlening waarbij sprake is van sessies, zoals inbellen, en het verzorgen van tijdelijke netwerkverbindingen. Ook bij dienstverlening waarbij sprake is van events, zoals het doen van bestellingen of het opvragen van een enkele www−pagina moet het framework toepasbaar zijn. Bij dienstverlening waarbij over een langere periode accounting−gegevens verzameld worden, zoals het verzorgen van een vaste verbinding of een vorm van hosting moet het ook mogelijk zijn om accounting−gegevens te verzamelen. Het framework moet dus niet specifiek zijn toegespitst op één specifieke vorm van dienstverlening en moet uitbreidbaar zijn voor nieuwe vormen van dienstverlening.

Transport onafhankelijk van accounting−protocol

Het gebruik van het framework moet zo veel mogelijk onafhankelijk zijn van het gebruikte accounting−protocol. Bij een verandering van accounting−protocol moeten er voor de gebruikers van het framework zo min mogelijk veranderingen optreden. Verschillende protocollen gebruiken natuurlijk wel verschillende attributen, die ervoor zorgen dat informatie op een andere manier wordt aangeboden.

Meerdere protocollen tegelijk

Het moet mogelijk zijn om in een enkel element van het accounting−systeem, bijvoorbeeld een accounting−server, meerdere accounting−protocollen tegelijk te ondersteunen. Om tegelijkertijd voor meerdere toepassingen, die verschillende protocollen ondersteunen, bruikbaar te zijn is het nodig dat een accounting−server tegelijkertijd bijvoorbeeld RADIUS−berichten en MSIX−berichten kan verwerken. Hierdoor is het mogelijk om accounting−berichten tussen de twee gebruikte protocollen te vertalen.

4.2 Toepasbaarheid

De toepasbaarheid schetst de omgeving waarin het framework toepasbaar moet zijn. Het geeft randvoorwaarden aan voor het functioneren in deze omgeving.

Het accounting−framework moet zo breed mogelijk toepasbaar zijn. Hierbij zijn zaken als platvorm−onafhankelijkheid en uitbreidbaarheid van belang.

Toepasbaar voor verschillende elementen

Het framework moet aangeven hoe het gebruikt kan worden in zowel accounting−servers, accounting−proxies als in service−elements. Het framework moet toepasbaar zijn in het hele proces van transport van accounting−gegevens. Het framework moet dus toepasbaar zijn in zowel een service−element als een accounting−server.

Toepasbaar in bestaande omgevingen

Het framework moet toepasbaar zijn in bestaande accounting−omgevingen door gebruik te maken van bestaande accounting−protocollen. Het moet bijvoorbeeld mogelijk zijn om het framework te gebruiken bij de communicatie met network−access−servers die alleen RADIUS of TACACS+ ondersteunen.

Meerdere berichtenmodellen

Het moet voor het bezorgen van accounting−gegevens mogelijk zijn om zowel een event−driven model, een polling model als een event−driven polling model toe te passen. Hieruit vloeien onder andere eisen ten aanzien van buffering en het ondersteunen van bidirectionele communicatie voort. Veel accounting−protocollen ondersteunen echter alleen unidirectionele communicatie of een te beperkte verzameling berichten en kunnen dan slechts beperkt worden toegepast. Het framework moet echter in alle berichtenmodellen toe te passen zijn. Het moet ook mogelijk zijn om accounting−protocollen die slechts in één model toepasbaar zijn, te gebruiken.

Uitbreidbaar

Het accounting−framework moet uitbreidbaar zijn met nieuwe accounting−protocollen en vormen van dienstverlening. Er moeten zo min mogelijk eisen worden gesteld aan nieuwe protocollen, zodat integratie in het framework eenvoudig kan geschieden. Uitbreidbaarheid is met name belangrijk, omdat er nieuwe accounting−protocollen in ontwikkeling zijn en telkens tal van nieuwe vormen van dienstverlening ontstaan. Voor een uitbreiding met een nieuw accounting−protocol of voor een nieuwe toepassing moeten zo min mogelijk aanpassingen in het systeem nodig zijn.

Onderhoudbaarheid

In het framework moet het eenvoudig zijn om in de toekomst ook andere zaken dan alleen accounting te regelen. Hierbij valt te denken aan geschiktheid voor bijvoorbeeld authenticatie en autorisatie. Het framework moet zo zijn opgezet dat uitbreidingen in het framework zelf eenvoudig te realiseren zijn.

4.3 Interface requirements

De interface requirements geven aan hoe de gebruiker van het framework, in dit geval een applicatie, het framework gebruikt. Het geeft aan welke taken een gebruiker met het framework kan uitvoeren.

Het framework biedt geen user−interface, maar een interface voor een applicatielaag. Deze moet eenvoudig en consistent zijn. De interface biedt de volgende functionaliteit:

Configuratie

Het moet mogelijk zijn om het systeem dynamisch te configureren. Bij de configuratie worden zaken als gebruikte accounting−protocollen, accounting−servers en transport−parameters ingesteld. Zo veel mogelijk instellingen van transport en verwerken van accounting−berichten moeten configureerbaar zijn.

Verzenden accounting−gegevens

Het framework moet de mogelijkheid bieden om accounting−gegevens te versturen op basis van eerder geconfigureerde parameters. Hierbij wordt het framework gebruikt op een service−element waarbij aangeboden accounting−gegevens worden verstuurd.

Ontvangen accounting−gegevens

Als het framework wordt gebruikt op een accounting−server, moet het bruikbaar zijn voor het eenvoudig ontvangen en bevestigen van ingekomen accounting−gegevens.

4.4 Schaalbaarheid en betrouwbaarheid

De schaalbaarheid en betrouwbaarheid van een systeem geven aan in welke mate het systeem bruikbaar moet zijn onder een belasting en welke voorzieningen het systeem hiervoor moet bieden.

Beperkte complexiteit

Het framework moet zo min mogelijk complexiteit tussen de applicatielaag, die de accounting−gegevens aanbiedt en verwerkt, en het daadwerkelijke transport aanbrengen. Hierdoor introduceert het weinig problemen voor de schaalbaarheid van een systeem en hangt dit voornamelijk af van het gebruikte accounting−protocol en het achterliggende verwerkende systeem.

Buffering

Het framework moet accounting−gegevens kunnen bufferen. Dit is nodig in een polling berichtenmodel, maar ook voor het opslaan van accounting−gegevens die nog niet zijn bevestigd. Op een accounting−server is buffering nodig van ingekomen berichten die nog niet zijn verwerkt.

Fail−over

Het moet mogelijk zijn om voor een service−element meerdere accounting−servers te ondersteunen, zodat als een primaire server niet bereikbaar is, kan worden overgegaan op het versturen van accounting−gegevens naar een tweede server. Ook is het hiermee mogelijk om load−balancing uit te voeren.

Performance

Het framework moet geschikt zijn om toegepast te worden in een omgeving waar een grote hoeveelheid accounting−gegevens real−time en simultaan moet worden verwerkt. Het is belangrijk dat het framework kan worden opgeschaald en geen beperkingen hiervoor bevat.

Zaken als schaalbaarheid en betrouwbaarheid hangen voor een groot deel af van de gebruikte accounting−protocollen. In een accounting−protocol moeten natuurlijk zaken als flow−control, integriteitsgarantie en dergelijke zijn geregeld.

4.5 Security requirements

De security requirements geven aan welke voorzieningen het systeem moet treffen om misbruik van het systeem te voorkomen. Ze geven aan hoe de integriteit en eventueel vertrouwelijkheid van het systeem en de gegevens blijft gewaarborgd.

In het framework moeten voldoende voorzieningen aanwezig zijn om encryptie van transport te regelen. Hierbij moet een end−to−end model kunnen worden gebruikt, maar ook beveiliging op hop−by−hop basis moet mogelijk zijn.

Het framework zelf hoeft geen encryptie te regelen, omdat deze voorzieningen in de betreffende accounting−protocollen aanwezig dient te zijn. Het moet wel voldoende mogelijkheden bieden om encryptie op een eenvoudige manier te implementeren.

5. Model

Het model geeft een definitie van de structuur van een accounting−systeem zoals het ontworpen gaat worden. Als basis hiervoor dienen de requirements. In het model worden zaken als accounting−protocol en accounting−bericht nader toegelicht en gedefinieerd. Ook wordt aangegeven welke elementen in een accounting−systeem berichten uitwisselen en op welke manier dit gebeurt.

In het framework moet het mogelijk zijn om eenvoudig verschillende accounting−protocollen te combineren, zodat er bij de communicatie van verschillende protocollen gebruik van kan worden gemaakt. Het uitgangspunt is dat eenvoudig nieuwe accounting−protocollen toegevoegd kunnen worden, zonder dat hiervoor uitgebreide aanpassingen nodig zijn in de rest van het framework.

Het model beschrijft de datastructuren die worden gebruikt, elementen die in het systeem zijn te onderscheiden, hoe deze gegevens uitwisselen en welke aannamen er betreft elementen zijn gemaakt. Het model beschrijft de architectuur van het accounting−framework.

5.1 Accounting−protocol

In het framework is het mogelijk om verschillende accounting−protocollen te gebruiken Om het gebruik van verschillende accounting−protocollen te vereenvoudigen, wordt een accounting−protocol opgedeeld in een transportprotocol, een recordformat en een berichtenlaag. Aan elk van deze onderdelen worden specifieke eisen gesteld.

Het transportprotocol

Het transportprotocol zorgt voor het "doorgeven van de bitjes". Hierin worden gecodeerde berichten doorgegeven. Een transportprotocol zou encryptie (integrity protection, authenticatie, confidentiality, replay protection en non−repudiation) kunnen bieden, maar dit zou ook op één van de andere lagen kunnen worden geregeld. Het transportprotocol regelt zaken als retransmissions, flow−control en buffering. Voorbeelden van transportprotocollen in dit verband zijn TCP, UDP en SCTP [rfc2960], maar ook protocollen als FTP en HTTP die eerdergenoemde protocollen gebruiken, kunnen als transportprotocol worden gebruikt.

Een transportprotocol dat goed toepasbaar is voor het bezorgen van accounting−berichten zorgt voor betrouwbaar transport van kleine en grote berichten, snelle foutdetectie zodat snel van servers kan worden veranderd, bidirectionele communicatie voor berichten van het service−element naar de accounting−server en viceversa en flow−control, zodat een burst van accounting−gegevens het netwerk en de accounting−server niet overspoelt.

Het recordformat

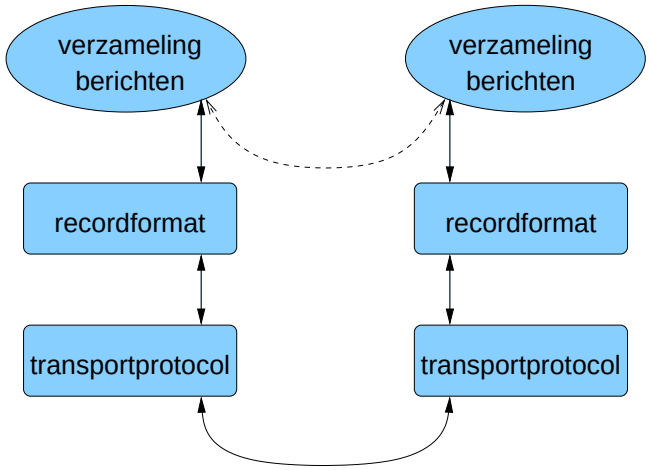
Het recordformat zorgt voor de transformatie vanuit een bericht met attributen en waarden naar een specifiek gecodeerde bitstream die kan worden verstuurd en viceversa. In het recordformat moet het mogelijk zijn om alle mogelijke soorten berichten vast te leggen die de berichtenlaag wil versturen. Het moet mogelijk zijn om alle mogelijk denkbare attributen op te slaan met gebruikmaking van standaard−datatypen als integer en string. Ook in deze laag zou encryptie kunnen worden toegepast. Vooral als er voor verschillende attributen verschillende niveaus van beveiliging worden geëist, is het recordformat hiervoor de aangewezen plaats. Recordformats kunnen gebaseerd zijn op ASN.1, XML, MIME, maar ook een eigen formaat als RADIUS is natuurlijk mogelijk. Het verdient aanbeveling om, zeker voor de vaste opslag van accounting−gegevens, een algemeen bekend of tekst formaat te gebruiken. Hiermee wordt debugging eenvoudiger en zijn accounting−gegevens eenvoudiger door andere programma’s te verwerken.

De berichtenlaag

De berichtenlaag definieert de mogelijke berichten en bevestigingen die kunnen worden verstuurd. Voor accounting zijn dit over het algemeen een "accounting request" bericht, waarin de accounting−gegevens worden verstuurd en een bijbehorende "accounting reply" waarin de ontvangst en verwerking van de accounting−gegevens wordt bevestigd. De requests zijn eventueel onder te verdelen in "start", "stop" en "interim" berichten, afhankelijk van de huidige toestand van de dienstverlening. Als een berichtenmodel wordt gebruikt waarin polling nodig is, zal ook een "accounting poll" bericht moeten worden gebruikt. Met dit bericht kunnen klaarstaande berichten op het service−element worden opgevraagd. Als een event−driven polling berichtenmodel wordt gebruikt, zijn ook "accounting indication" berichten nodig om de accounting server te laten weten dat er accounting−berichten klaar staan om te worden bezorgd. In het framework wordt ervan uitgegaan dat accounting−berichten onder te verdelen zijn in bovengenoemde berichtsoorten.

De berichtenlaag en de vorm van de geleverde dienst bepalen welke attributen in welke berichten moeten worden gebruikt. De functionaliteit van de berichtenlaag zal in het algemeen afhangen van de geleverde dienst.

Deze drie delen worden voor het te ontwerpen framework als afzonderlijk implementeerbaar beschouwd. Voor veel bestaande protocollen kan dat zonder veel problemen, temeer omdat een aantal protocollen maar één van de bovengenoemde lagen definieert. Voor een enkeling moeten er extra aanpassingen worden gemaakt, omdat de verschillende lagen verweven zijn (bijvoorbeeld transporteigenschappen naar aanleiding van de inhoud van het bericht). Als de afzonderlijke delen van het accounting−protocol goed scheidbaar zijn, kan een willekeurig recordformat met een willekeurig transportprotocol worden gecombineerd. Als recordformat, transportprotocol en berichtenlaag niet goed te scheiden zijn, zoals bij RADIUS, is het accounting−systeem maar voor beperkte toepassingen bruikbaar. RADIUS is bijvoorbeeld vrijwel alleen bruikbaar in een inbel−omgeving.



Figuur 5.1: verschillende lagen in accounting−protocol.

Bij scheiding van berichtenverzameling, recordformat en transportprotocol ontstaat een gelaagd model (zie *Figuur 5.1*) waarin twee berichtenverzameling−lagen onderling berichten uitwisselen door gebruik te maken van een recordformat en een transportprotocol. Twee elementen kunnen communiceren als ze hetzelfde transportprotocol en recordformat gebruiken en op de berichtenlaag geldige berichten uitwisselen.

5.2 Berichten

In de berichtenlaag wordt een abstracte representatie van een bericht gebruikt. Dit bericht bestaat uit een berichtsoort en een verzameling attributen met waarden. Het soort bericht kan een "accounting request", "accounting reply", "accounting poll" of "accounting indication" zijn.

accounting request

Met een "accounting request" wordt accounting−informatie van een client naar een accounting−server getransporteerd.

accounting reply

Als een accounting−server accounting−informatie ontvangt en verwerkt, wordt deze informatie bevestigd met een "accounting reply" bericht.

accounting poll

Een accounting−server kan een service−element vragen om alle gebufferde accounting−gegevens te versturen. Dit gebeurt met een "accounting poll" bericht. Het service−element zal dit over het algemeen beantwoorden met een of meer "accounting requests".

accounting indication

Een service−element kan, in plaats van het direct versturen van de accounting−gegevens, ook een signaal aan de accounting−server geven dat accounting−gegevens op het service−element beschikbaar zijn. Dit gebeurt met een "accounting indication" bericht. De accounting−server zal dit over het algemeen, eventueel na verloop van tijd, beantwoorden met een "accounting poll" bericht.

Het "accounting indication" bericht zal worden gebruikt in een event−driven polling berichtenmodel. De andere berichten zullen in dat model ook worden gebruikt. In het polling model zullen de "accounting poll"," accounting request" en "accounting reply" berichten worden gebruikt. In een event−driven berichtenmodel zullen alleen de "accounting request" en "accounting reply" berichten worden gebruikt.

De verzameling gebruikte attributen in een accounting−bericht is afhankelijk van de toepassing en van het accounting−systeem waarin het wordt gebruikt. In een systeem waarin RADIUS voor network access wordt gebruikt zullen attributen als "NAS−IP−Address", "Acct−Input−Octets" en "Acct−Output−Packets" worden gebruikt, terwijl bij accounting van webserver−logs attributen als "httpd_host" en "httpd_url" worden gebruikt. Over het algemeen zal er altijd een attribuut aanwezig zijn die een sessie uniek identificeert en er zal vaak een attribuut aanwezig zijn dat een al dan niet geauthenticeerde gebruiker van de dienst aangeeft.

Alle attributen hebben een naam en een waarde. De naam bestaat uit een rij van letters en/of cijfers. De waarde kan van verschillende types zijn, afhankelijk van de betekenis. Over het algemeen moet een waarde zijn te schrijven als een lijst van verschillende karakters.

Binnen het "accounting request" bericht valt bij veel accounting−protocollen onderscheid te maken tussen een start−, stop− en interim−bericht om aan te geven of het het begin, einde of een update van de sessie betreft. Ook is het mogelijk om, als er sprake is van dienstverlening in de vorm van events, het "accounting request" niet onder te verdelen. In het accounting−bericht wordt een eventuele onderscheid in het soort accounting−bericht kenbaar gemaakt door middel van één of meer attributen in het bericht.

5.3 Structuur

Alle elementen die aan het transport van accounting−berichten deelnemen hebben een gelijksoortige structuur. Dat wil zeggen dat het transportprotocol en het recordformat van zowel het service−element als de accounting−server dezelfde eigenschappen hebben. Met deze opzet kan in alle elementen dezelfde basisimplementatie worden gebruikt.

De verschillen tussen service−element, accounting−server, proxy en broker zit hem vooral in de functionaliteit op het niveau van de berichtenlaag. Een service−element zal over het algemeen accounting−berichten versturen en alleen reageren op inkomende polling−berichten. Een accounting−server zal inkomende accounting−berichten accepteren en bevestigen en eventueel polling berichten versturen. Een proxy en een broker sturen over het algemeen alleen ontvangen berichten door.

Het service−element

Het service−element verstuurt "accounting requests" aan een accounting−server of −proxy en verwerkt de bevestigingen hiervan. Het service−element kan polling berichten ontvangen en zal daarop nog niet verstuurde accounting−berichten versturen en eventueel voor de lopende sessies accounting−berichten genereren en versturen. Als een event−driven polling model wordt gebruikt, zal het service−element "accounting indication" berichten aan een accounting−server of proxy versturen om aan te geven dat accounting−berichten klaar zijn om te worden verstuurd. De accounting−server kan deze dan via polling ophalen.

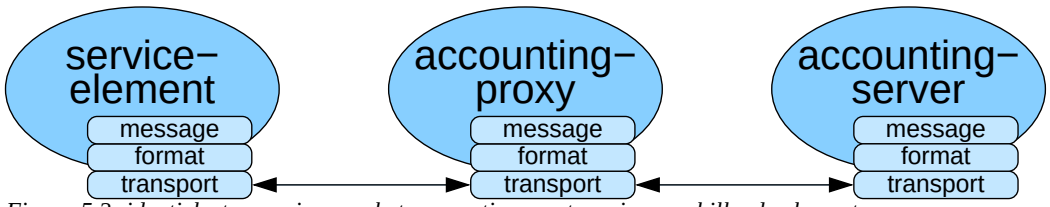
De accounting−server

Een accounting−server zal inkomende "accounting requests" verwerken en bevestigen. Een accounting−server kan polling berichten versturen aan "aangemelde" service−elements. Deze aanmelding kan bij het configureren van de server gebeuren. De aanmelding kan ook dynamisch geschieden, als een service−element met een "accounting indication"−bericht meldt dat het accounting−berichten beschikbaar heeft. Dit gebeurt bij event−driven polling. De accounting−server moet dus ook "accounting indications" kunnen verwerken.

De accounting−proxy/broker

Een accounting−proxy of −broker moet van inkomende accounting−berichten (en antwoorden) de juiste bestemming weten te bepalen en het bericht doorsturen. Eventueel kunnen extra attributen worden toegevoegd als dit noodzakelijk is. Ook kan een broker de gebruikte encryptiemethode aanpassen als dit nodig is. Als in de berichten encryptie is gebruikt, moet de proxy minstens het deel kunnen ontcijferen waaruit valt af te leiden waar het accounting−bericht heen moet. De proxy moet natuurlijk wel in staat zijn om gecodeerde delen van het bericht, die het zelf niet kan lezen, door te sturen.

Uitgangspunt is dat in alle elementen dezelfde basisimplementatie kan worden gebruikt om accounting−berichten uit te wisselen. De overeenkomsten zitten in een gelijk gebruik van tranportprotocol en recordformat, terwijl de verschillen zich voordoen op de het niveau van de aansturende laag die met de berichtenlaag is verweven. In *Figuur 5.2* is schematisch aangegeven waar de verschillen en overeenkomsten in de verschillende soorten elementen zijn te vinden.



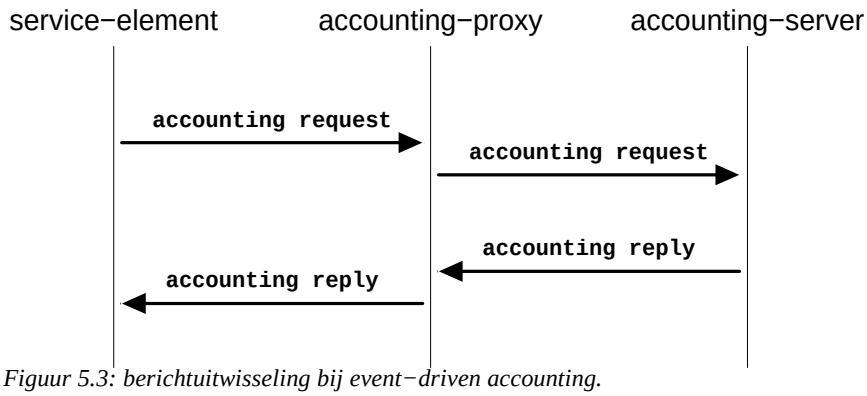
Figuur 5.2: identieke toepassing van het accounting−systeem in verschillende elementen.

Zowel het service−element, de accounting−proxy als de accounting−server gebruiken hetzelfde transportprotocol en recordformat. Ook worden gelijksoortige berichten uitgewisseld. De verschillen zitten in de manier waarop de inkomende berichten worden verwerkt en wanneer berichten worden verstuurd.

5.4 Berichtuitwisseling

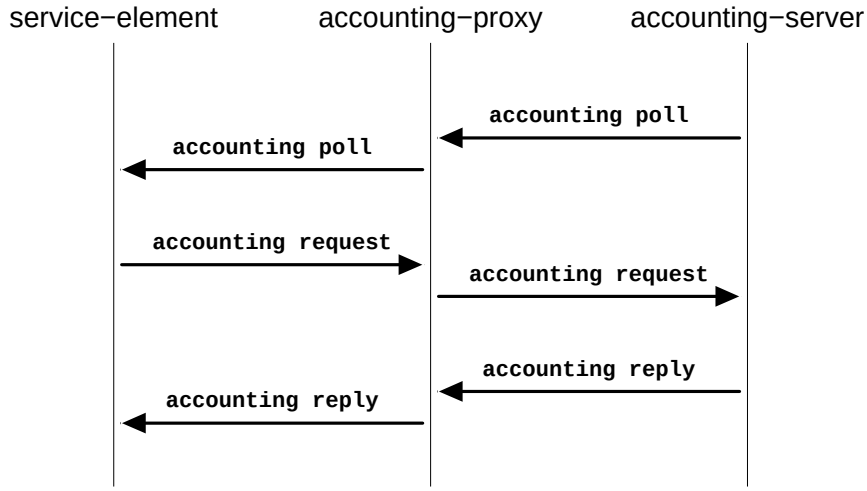
Het service−element, de eventueel aanwezige accounting−proxy en de accounting−server wisselen accounting−berichten uit. De manier waarop dit gebeurt en welke berichten worden uitgewisseld hangen af van het gehanteerde berichtenmodel.

In het eenvoudigste model, event−driven berichtbezorging, stuurt het service element "accounting request"−berichten naar een accounting−server of accounting−proxy. Uiteindelijk wordt dit beantwoord met een "accounting reply". In *Figuur 5.3* is het uitwisselen van berichten bij event−driven berichtbezorging geïllustreerd.



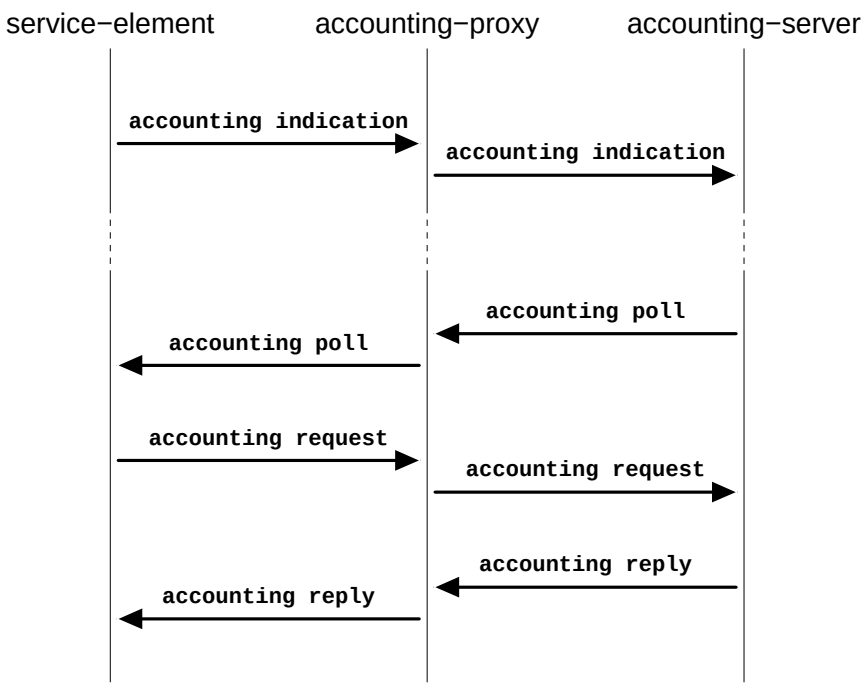
Figuur 5.3: berichtuitwisseling bij event−driven accounting.

Bij polling neemt niet het service−element, maar de accounting−server het initiatief om accounting−data te transporteren. De accounting−server stuurt een "accounting poll" bericht, waarop het service−element eventueel gebufferde accounting−berichten zal versturen. De accounting−server zal deze berichten op de gebruikelijke manier bevestigen. Het uitwisselen van accounting−berichten bij een polling model staat toegelicht in *Figuur 5.4*.



Figuur 5.4: berichtuitwisseling bij polling.

Bij event−driven polling ligt het initiatief van het daadwerkelijk versturen van de accounting−berichten bij de accounting−server. Het service−element stelt de accounting−server echter met een "accounting indication" bericht op de hoogte als het accounting−berichten klaar heeft staan om verzonden te worden. Dit is in *Figuur 5.5* toegelicht.



Figuur 5.5: berichtuitwisseling bij event−driven polling.

In alle gevallen speelt de accounting−proxy de rol van accounting−server ten opzichte van het service−element en de rol van service−element ten opzichte van de accounting−server. Voor het uitwisselen van berichten maakt het niet uit of er een accounting−proxy in de communicatie betrokken is. De accounting−proxy kan natuurlijk wel de berichten op attribuut−niveau veranderen en doorgeven of routering−beslissingen nemen, afhankelijk van de functie en werking van de proxy.

5.5 Beveiliging

Beveiliging kan op een aantal niveaus worden geregeld. Het kan op het transportprotocol−niveau plaatsvinden, waardoor een hop−by−hop security model tot stand komt, maar het kan ook op het recordformat−niveau worden geregeld, waarmee een end−to−end beveiliging kan worden opgezet.

Als op transportniveau, bijvoorbeeld met behulp van SSL, TLS of IPSec een beveiligde verbinding wordt geregeld, gebeurt dit tussen de elementen in het accounting−proces onderling. De toepassing op deze laag is relatief eenvoudig en hier zijn standaarden (zoals de eerdergenoemde methoden) voor beschikbaar. Toepassing op deze laag is echter weinig flexibel. De hele datastroom wordt beveiligd en in meer ingewikkelde beveiligingsaspecten als non−repudiation is meestal niet voorzien.

Als op het niveau van recordformat encryptie wordt gebruikt, zal dit over het algemeen via een specifieke implementatie moeten verlopen. Er zijn wel encryptiemethoden als S/MIME en PGP beschikbaar. Deze werken echter over het algemeen alleen op specifieke formaten. Als er encryptie nodig is van losse attributen met verschillende niveaus van beveiliging moet een eigen implementatie worden gemaakt. Deze methode is over het algemeen recordformat−specifiek en kan dus moeilijk tussen twee verschillende recordformats worden vertaald.

Het is ook mogelijk om de encryptie van bepaalde attributen niet in de AAA−laag te regelen, maar om deze gegevens in encrypte vorm aan de AAA−laag aan te bieden. Het versturen van passwords voor authenticatie is hier een voorbeeld van. In geval van wachtwoorden moeten eigenlijk alleen de client en de home AAA−server deze kunnen lezen. Het is niet wenselijk dat service−elements en proxy−servers beschikking hebben over deze gegevens.

Voor accounting zal in veel gevallen hop−by−hop security voldoende zijn. Dit kan dan in het transportprotocol worden geregeld. Eventuele uitbreidingen, zoals cryptografisch gesigneerde bevestigingen, zoals bij non−repudiation nodig is, moeten in het recordformat worden geregeld.

6. Ontwerp

In het ontwerp wordt de structuur van de software beschreven. Het ontwerp wordt gemaakt op basis van het model met inachtneming van de requirements. Het ontwerp geeft een overzicht van alle software−componenten, welke eigenschappen en methoden deze hebben, hoe zij zich onderling tot elkaar verhouden en hoe de interactie tussen de verschillende componenten plaatsvindt.

Het ontwerp kan direct als basis worden gebruikt voor de implementatie. Er is voor gekozen de implementatie in Java te maken. Java applicaties zijn eenvoudig uitbreidbaar en lenen zich goed voor het dynamisch toevoegen van objecten. Java heeft echter enkele beperkingen, met name in de performance. Het creëren van Java−objecten is een relatief tijdsintensieve operatie en hier dient dus bij de implementatie rekening mee te worden gehouden. Java is platvorm−onafhankelijk en daarom voor een dergelijk framework goed toepasbaar.

Met de keuze voor de implementatie−taal is in het ontwerp al rekening gehouden door uit te gaan van een object−georiënteerd model met een opdeling in klassen. Het probleem blijkt zich goed te lenen voor een object−georiënteerde aanpak, omdat nieuwe software−componenten, waarvan maar een deel van de functionaliteit bekend is, moeten kunnen worden ingevoegd.

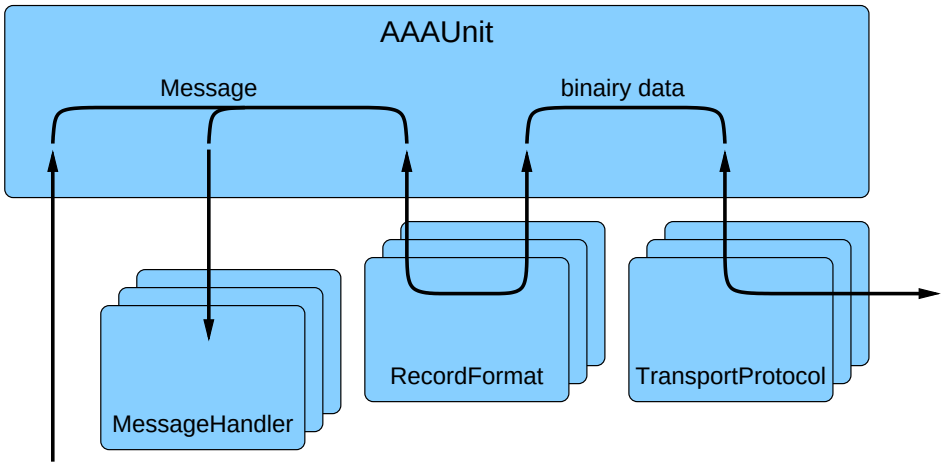
In dit hoofdstuk zullen van alle software−onderdelen de functie en de bijgehouden datastructuren worden beschreven. Gebruikte datatypen zullen worden gedefinieerd en de relatie met andere software−componenten zal worden toegelicht. Van alle specifieke functies van de onderdelen zal de invoer, uitvoer en de relatie tussen beide worden beschreven. Van alle bijgehouden gegevens zullen type en structuur worden gegeven.

Naast de definitie van de verschillende modulen zal de uitwisseling van berichten worden toegelicht, uitgelegd zal worden hoe de invulling op de berichtenlaag wordt geregeld en tot slot zullen alle modulen worden samengevat in een klassendiagram.

6.1 AAAUnit

De module die transportprotocollen, recordformats en berichtenafhandeling koppelt is de AAAUnit. De AAAUnit zorgt dat uitgaande berichten in het juiste recordformat worden gecodeerd en met het juiste transportprotocol worden verstuurd. Tevens zorgt het dat inkomende berichten worden gedecodeerd en worden aangeboden om te worden verwerkt. Dit proces is schematisch weergegeven in *Figuur 6.1*.

In de AAAUnit kunnen verschillende transportprotocollen worden gecombineerd met verschillende recordformats. Bij eenvoudige dienstverlening zou een DIAMETER/RADIUS/TACACS−achtig recordformat kunnen worden gebruikt (klein en efficiënt), terwijl bij complexere diensten een XML−gebaseerd recordformat kan worden gebruikt. Het gebruikte transportprotocol kan afhangen van de onderliggende netwerkstructuur en de eisen ten aanzien van de bezorging (real−time accounting). Het integreren van een applicatie met een dergelijke opzet in een bestaande omgeving kan relatief eenvoudig verlopen, doordat het reeds gebruikte protocol kan worden geïntegreerd in het framework.



Figuur 6.1: AAAUnit

Met een dergelijke opzet kunnen berichten worden verstuurd door gebruik te maken van een willekeurig recordformat en transportprotocol. De aanroepende laag, de berichtenlaag, hoeft alleen maar te werken met een abstracte representatie van een bericht.

De taken van de AAAUnit bestaan uit:

- het versturen van berichten met het juiste recordformat/transportprotocol,
- het decoderen en afhandelen van inkomende berichten en
- het bijhouden van recordformats, transporprotocollen en messagehandlers die kunnen worden gebruikt.

Versturen van uitgaande berichten

invoer: bericht in abstracte vorm, bestemming

Als een bericht voor versturen aan de AAAUnit wordt aangeboden kunnen het te gebruiken RecordFormat en TransportProtocol worden opgegeven. Als er geen voorkeur wordt opgegeven, kiest de AAAUnit het recordformat en transportprotocol die als default zijn opgegeven. Een bericht wordt vastgelegd met behulp van een recordformat, waarna het wordt verstuurd met behulp van het transportprotocol. Als er een fout optreedt bij het versturen, wordt dit doorgegeven middels een exceptie.

Decoderen en afhandelen van inkomende berichten

invoer: bericht in gecodeerde vorm, afzender

Inkomende berichten worden door het TransportProtocol aan de AAAUnit doorgegeven, waarna deze alle geregistreerde RecordFormats nagaat en bepaalt welk RecordFormat is gebruikt. Op deze manier wordt het verstuurde bericht gereconstrueerd. Als het bericht niet kan worden gedecodeerd, wordt het weggegooid.

Na het decoderen wordt het ingekomen bericht voor verwerking aan een geregistreerde MessageHandler aangeboden. Als er geen enkele MessageHandler is die het bericht kan verwerken wordt het bericht genegeerd.

Toevoegen RecordFormat, TransportProtocol of MessageHandler

invoer: RecordFormat, TransportProtcol of MessageHandler

De betreffende module wordt toegevoegd aan de verzameling van geregistreerde modules. Hierna is het direct bruikbaar voor het verwerken van inkomende en uitgaande berichten.

Verwijderen RecordFormat, TransportProtocol of MessageHandler

invoer: RecordFormat, TransportProtocol of MessageHandler

De betreffende module wordt verwijderd uit de verzameling van geregistreerde modules. Na deze operatie is de betreffende module niet meer bruikbaar voor het versturen of ontvangen van berichten.

Om een nieuw recordformat of transportprotocol te kunnen gebruiken hoeft deze alleen te worden geïmplementeerd. Het invoegen van een nieuw accounting−protocol in het framework kan dan over het algemeen zonder aanpassingen in de berichtenlaag geschieden.

Een dergelijke algemene opzet voor het versturen en ontvangen van berichten is in meerdere systemen bruikbaar voor het versturen van berichten die bestaan uit een berichtsoort en een verzameling attributen. De AAAUnit is ontworpen om ook toepasbaar te zijn bij het versturen en ontvangen van autorisatie− en authenticatie−berichten, die eenzelfde structuur hebben.

6.2 TransportProtocol

De module TransportProtocol definieert de minimale eisen waaraan de implementatie van een transportprotocol moet voldoen. Omdat er veel verschillende transportprotocollen mogelijk zijn, valt voor een TransportProtocol alleen een algemene omschrijvingen van taken te geven.

Een TransportProtocol zorgt voor het overbrengen van de berichten. Het is mogelijk om verschillende transportprotocollen te gebruiken, afhankelijk van de toepassing. Een TransportProtocol kan een betrouwbare dienst leveren, waarbij op het niveau van het transportprotocol bevestigingen worden verstuurd. Als de toepassing dit niet vereist, bijvoorbeeld doordat het verkeer alleen over een lokaal netwerk gaat, kunnen de bevestigingen op het niveau van de berichtenlaag voldoende zijn.

Het TransportProtocol heeft als taken:

- het versturen van uitgaande, gecodeerde berichten en
- het doorgeven van inkomende berichten aan de AAAUnit.

Versturen uitgaand bericht

invoer: gecodeerd bericht, bestemming

Het aangeboden bericht wordt aan de bestemming verzonden. Als het TransportProtocol een fout detecteert, bijvoorbeeld het onbereikbaar zijn van de bestemming, zal het dit aan de aanroepende laag doorgeven. Als het transportprotocol een onbevestigd transport levert, zijn niet alle fouten te detecteren en het uitblijven van een foutmelding biedt dus, afhankelijk van het gebruikte transportprotocol, geen garantie voor het daadwerkelijk succesvol ontvangen van het bericht. De berichtenlaag moet uiteindelijk de ontvangst en het verwerken van het bericht controleren aan de hand van een verstuurd bevestigingsbericht.

Als het transportprotocol een inkomende verbinding of een inkomend bericht signaleert, moet het dit aan de AAAUnit waarbij het is geregistreerd doorgeven. Als het een enkel bericht betreft, zal het gecodeerde bericht in zijn geheel aan de AAAUnit worden doorgegeven. Als het een stroom betreft, bijvoorbeeld bij een verbindingsgericht transportprotocol, wordt de stroom waaruit de gecodeerde berichten gelezen kunnen worden aan de AAAUnit opgegeven. Als het TransportProtocol dit kan afleiden zal het ook een referentie naar de afzender van de gegevens aan de AAAUnit doorgeven.

Een TransportProtocol houdt, afhankelijk van het soort transport dat plaatsvindt, gegevens over lopende verbindingen en uitstaande berichten bij.

Vanaf het moment dat het transportprotocol bij de AAAUnit wordt aangemeld zorgt het ervoor dat alle berichten en verbindingen die via het TransportProtocol binnenkomen aan de AAAUnit worden doorgespeeld.

6.3 Message

De abstracte representatie van een bericht, zoals deze in de berichtenlaag wordt gebruikt, is geïmplementeerd in de module Message. Een bericht bestaat uit:

- een berichttype en
- een verzameling attributen.

Het berichttype geeft het soort bericht aan ("accounting request", "accounting reply", etc). De verzameling attributen vormen de inhoud van het bericht. Een attribuut bestaat uit een naam en een waarde.

Naast de inhoud van een bericht kunnen de volgende zaken van een bericht van belang zijn:

- de afzender,
- de ontvanger,
- de in het recordformat gecodeerde versie van het bericht,
- een verwijzing naar het gebruikte of te gebruiken RecordFormat en
- een verwijzing naar het gebruikte of te gebruiken TransportProtocol.

Deze zaken kunnen van belang zijn voor het bepalen van wat er met het bericht moet gebeuren. Ze hoeven natuurlijk niet altijd bij elk bericht te identificeren zijn. Als een bericht nog niet is verzonden, kan het zijn dat de uiteindelijke ontvanger nog niet bekend is. Het betreffende veld wordt dan als leeg gedefinieerd.

Gebruikelijke operaties met een bericht zijn:

- het coderen van de inhoud van het bericht om het te versturen,
- het aanmaken van een nieuw bericht,
- het doorsturen van een bericht aan een andere ontvanger en
- het maken van een antwoord op een ontvangen bericht.

Het coderen van het bericht tot binaire data gebeurt door het RecordFormat. Bij de overige operaties wordt een nieuw bericht gegenereerd, eventueel op basis van een bestaand bericht.

Aanmaken nieuw bericht

invoer: berichttype, verzameling attributen

uitvoer: een nieuw bericht

Er wordt een nieuw bericht aangemaakt met het opgegeven berichttype en de opgegeven attributen. De overige gegevens van het bericht, zoals afzender, gebruikt RecordFormat en dergelijke, worden nog niet gedefinieerd. Het bericht wordt nog niet direct verstuurd, alleen aangemaakt.

Doorsturen bericht

invoer: bericht, nieuwe bestemming

uitvoer: een nieuw bericht

Er wordt een nieuw bericht aangemaakt op basis van een bestaand bericht. Alle gegevens uit het oorspronkelijke bericht worden overgenomen, waarbij alleen de bestemming wordt aangepast met een nieuwe waarde. Het bericht wordt nog niet direct doorgestuurd, het wordt alleen aangemaakt.

Maken van een antwoord

invoer: bericht, berichttype, verzameling attributen

uitvoer: een nieuw bericht

Er wordt een nieuw bericht aangemaakt met het opgegeven berichttype en de opgegeven attributen. De overige gegevens worden bepaald op basis van de gegevens in het oorspronkelijke bericht. Het bericht wordt nog niet direct verstuurd, alleen aangemaakt. Als op deze manier een antwoord−bericht wordt geconstrueerd, zal de AAAUnit, tenzij anders wordt opgegeven, bij het versturen van het antwoord hetzelfde RecordFormat en TransportProtocol gebruiken als die van het oorspronkelijk ontvangen bericht.

6.4 RecordFormat

De module RecordFormat definieert de manier waarop berichten worden gecodeerd, zodat ze kunnen worden verstuurd. Het RecordFormat zorgt ook voor het reconstrueren van attribuutnamen en attribuutwaarden uit gecodeerde berichten. De taken van een RecordFormat zijn dus:

- het coderen van uitgaande berichten en
- het decoderen inkomende berichten.

De exacte invulling van het coderen en decoderen hangt sterk af van het gebruikte recordformat. Als het recordformat bijvoorbeeld encryptie definieert, moeten sleutels hiervoor zijn uitgewisseld.

Coderen uitgaand bericht

invoer: bericht (abstracte vorm)

uitvoer: gecodeerd bericht (binaire vorm)

Het aangeboden bericht wordt vastgelegd in een binair formaat, dat geschikt is voor versturen. In het recordformat wordt het berichttype en de verzameling attributen vastgelegd, zodat het oorspronkelijke bericht later weer kan worden gereconstrueerd.

Als encryptie wordt gebruikt om het bericht te vormen, kan aan de hand van de in het bericht aanwezige gegevens, zoals ontvanger, worden bepaald welke sleutels moeten worden gebruikt.

Decoderen inkomend bericht

invoer: gecodeerd bericht (binair formaat), afzender

uitvoer: bericht (abstracte vorm)

Het inkomende gecodeerde bericht kan in zijn geheel worden aangeboden of als stroom van berichten. Als het RecordFormat het formaat van het inkomende bericht herkent, wordt het gedecodeerd en geretourneerd. Als een RecordFormat de gebruikte codering van een inkomend bericht niet herkent, wordt dit aan de AAAUnit doorgegeven, zodat deze een ander recordformat kan proberen. Als er een duidelijke fout wordt gedetecteerd in een herkend bericht, moet dit worden doorgeven middels een exceptie, zodat de AAAUnit geen verdere RecordFormats hoeft te proberen om het inkomende bericht te decoderen.

Als er sprake is van een stroom van berichten wordt het eerste bericht, als het leesbaar is, uit de stroom gelezen. Als er geen bericht in het begin van de stroom herkenbaar is, moet de stroom onveranderd blijven, zodat een volgend RecordFormat het kan proberen te lezen. De AAAUnit zal door herhaald uit de stroom te lezen alle berichten uit de stroom lezen.

De afzender wordt meegegeven zodat het RecordFormat de juiste encryptiesleutels voor het bericht kan gebruiken om het te decoderen.

Attributen die niet kunnen worden gedecodeerd, bijvoorbeeld omdat het element de juiste encryptiesleutels niet heeft, moeten wel in een bepaalde vorm in het bericht beschikbaar zijn. Het kan nodig zijn om de gecodeerde attributen door te sturen, bijvoorbeeld omdat er sprake is van een accounting−proxy. Een manier om deze attributen te behouden is bijvoorbeeld door alle attributen die niet zijn te decoderen te vatten in één attribuut met als naam "encrypted_data", of door per attribuut een attribuut met een aangepaste naam op te nemen, zoals "encrypted_uid" in plaats van "uid".

6.5 MessageHandler

Inkomende gedecodeerde berichten worden afgehandeld door een MessageHandler. Van gedecodeerde berichten in abstracte vorm besluit de MessageHandler of en hoe het bericht wordt afgehandeld. Het is mogelijk om verschillende MessageHandlers te definiëren, met elk een eigen taak. Zo is het bijvoorbeeld mogelijk om een MessageHandler te maken die inkomende "accounting request"−berichten afhandelt en bevestigt.

De taak van een MessageHandler bestaat dus uit:

- het afhandelen van inkomende berichten.

Afhandelen inkomend bericht

invoer: bericht

uitvoer: succes/geen succes

Een MessageHandler bepaalt aan de hand van het bericht en de taak van de MessageHandler of deze het inkomende bericht kan afhandelen. Als het bericht kan worden afgehandeld moet dit direct gebeuren. Het kan bijvoorbeeld nodig zijn om andere berichten te versturen.

Als een MessageHandler een inkomend bericht niet kan afhandelen, wordt dit aan de AAAUnit doorgegeven. Deze kan dan een andere MessageHandler proberen.

Een service−element zal in het algemeen weinig MessageHandlers gebruiken, omdat het alleen inkomende reacties op verstuurde berichten hoeft te verwerken en, als er sprake is van een "polling model", "polling requests". Een accounting−server zal in het algemeen alleen "accounting requests" en "accounting indications" (bij event−driven polling) ontvangen.

6.6 MessageHandlerContainer

Een bijzondere vorm van een MessageHandler is een MessageHandlerContainer. Deze verwerkt zelf geen berichten, maar delegeert dit aan geregistreerde MessageHandlers. Als een MessageHandlerContainer een bericht voor afhandeling ontvangt, probeert het één voor één de geregistreerde MessageHandlers. Als één van de MessageHandlers doorgeeft dat het bericht succesvol is verwerkt, zal de MessageHandlerContainer dit weer aan de aanroeper ervan doorgeven. Als geen van de geregistreerde MessageHandlers het bericht kan verwerken, zal de MessageHandlerContainer doorgeven dat het bericht niet kan worden verwerkt.

De taken van een MessageHandlerContainer bestaat uit:

- het delegeren van inkomende berichten en
- het bijhouden welke MessageHandlers zijn geregistreerd.

Delegeren inkomend bericht

invoer: bericht

uitvoer: succes/geen succes

Een MessageHandlerContainer delegeert het daadwerkelijke afhandelen aan de geregistreerde MessageHandlers. Als een van de MessageHandlers het bericht kan afhandelen zal dit als succes worden gesignaleerd.

MessageHandler toevoegen

invoer: MessageHandler

De MessageHandler wordt toegevoegd aan het eind van de lijst met MessageHandlers die gebruikt kunnen worden.

MessageHandler verwijderen

invoer: MessageHandler

Een geregistreerde MessageHandler wordt verwijderd uit de lijst met geregistreerde MessageHandlers.

Een afgeleide van de MessageHandlerContainer kan eventueel voor het proberen van de geregistreerde MessageHandlers een beslissing maken op basis van de inhoud van het bericht. Het is mogelijk om hiermee een filterende functie in te bouwen. Het is denkbaar een MessageHandlerContainer te maken die alleen berichten afhandelt, die zijn bestemd voor een bepaald domein.

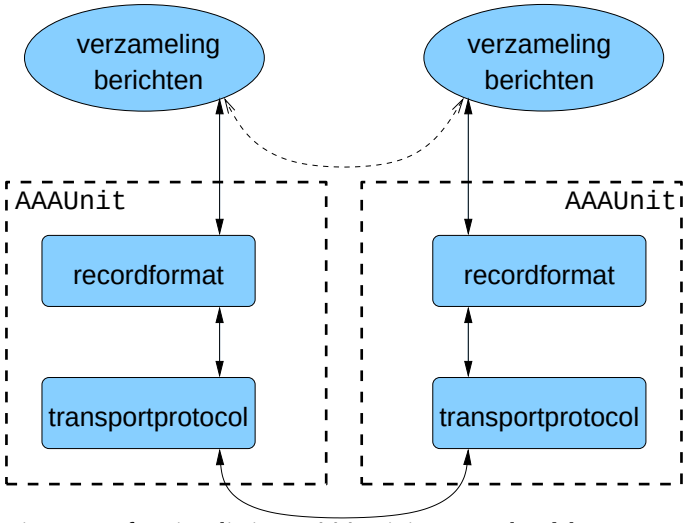
De AAAUnit is zelf ook een MessageHandlerContainer die af te handelen berichten delegeert aan geregistreerde MessageHandlers.

6.7 Berichtuitwisseling

Met gebruikmaking van AAAUnit, TransportProtocol, Message, RecordFormat, MessageHandler en MessageHandlerContainer is het mogelijk om berichten met verschillende protocollen tussen verschillende elementen uit te wisselen. Het is mogelijk om inkomende berichten op een generieke manier af te handelen door gebruik te maken van MessageHandlers en MessageHandlerContainers. Hiermee is het ook mogelijk om inkomende berichten in hiërarchische categorieën onder te verdelen.

Uitgaande berichten kunnen aan de AAAUnit worden aangeboden voor verzenden. Deze gebruikt een RecordFormat voor coderen en verstuurt het bericht dan met een TransportProtocol. Als een TransportProtocol een bericht ontvangt, wordt dit bericht aan de AAAUnit aangeboden. De AAAUnit decodeert het bericht door gebruik te maken van een geregistreerd RecordFormat. Daarna wordt het bericht verwerkt door de MessageHandlers.

Met de functionaliteit van de AAAUnit is de communicatie op de laag van recordformat en transportprotocol geregeld. Dit is geïllustreerd in *Figuur 6.2*. Tevens is, middels de MessageHandlers, een opzet aanwezig die kan worden gebruikt in de berichtenlaag.



Figuur 6.2: functionaliteit van AAAUnit in protocolmodel.

De AAAUnit verzorgt alleen het bezorgen en ontvangen van berichten. Hiermee wordt een eenvoudig event−driven berichtenmodel verzorgd. Omdat er bij het transport van accounting−gegevens niet alleen sprake is van een transportprotocol en een recordformat, is er ook enige functionaliteit op berichtenlaag−niveau nodig. Er moet kunnen worden gedefinieerd op welke manier wat voor soort berichten kunnen worden uitgewisseld. Op de berichtenlaag moeten voorzieningen komen waarmee kan worden bepaald wat er met inkomende berichten moet worden gedaan, welke berichten moeten worden bevestigd, op welke manier gegevens worden gebufferd, etc.

6.8 AccountingSender

Om het verzenden van accounting−gegevens te regelen is de AccountingSender ontworpen. De AccountingSender zorgt voor het verzenden van accounting−gegevens met gebruikmaking van een bepaald berichtenmodel. Aan de hand van zaken als berichtenmodel (event−driven, polling, event−driven polling), maximale bufferruimte en timeout−parameters kunnen aangeboden accounting−gegevens worden verzonden.

De AccountingSender gebruikt de AAAUnit voor het verzenden van accounting−gegevens middels "accounting request" en eventueel "accounting indication"−berichten. Voor het ontvangen van "accounting reply" en "accounting poll"−berichten wordt een MessageHandler bij de AAAUnit geregistreerd.

De AccountingSender zorgt ook voor het beheren van een verzameling van accounting−servers, die voor de communicatie kunnen worden gebruikt. Als accounting−berichten niet binnen een opgegeven tijdslimiet worden bevestigd of er bij het verzenden een fout optreedt, dan kan een andere accounting−server worden geprobeerd.

De taken van de AccountingSender bestaan uit:

- het transporteren van accounting−gegevens naar een accounting−server in het juiste berichtenmodel,
- het beheren van de verzameling accounting−servers,
- het afhandelen van "accounting reply"−berichten welke antwoorden zijn op eerder verstuurde "accounting request"−berichten en
- het afhandelen van en reageren op "accounting poll"−berichten.

Hiertoe worden in de AccountingSender de volgende gegevens bijgehouden:

- gegevens over het gebruikte berichtenmodel,
- een lijst van geregistreerde accounting−servers,
- een buffer van berichten die wachten op verzenden en
- een buffer van berichten die wachten op bevestiging.

Hiernaast zijn er nog enkele parameters die worden gebruikt om binnen het gebruikte berichtenmodel de vorm van communicatie te bepalen:

- de maximale buffergrootte voor te verzenden berichten,
- de maximale tijd dat berichten worden gebufferd,
- een flag die aangeeft of een "accounting indication" moet worden verstuurd,
- een timeout−waarde waarna een "accounting indication" als verloren wordt beschouwd en
- een timeout−waarde waarna een "accounting request" als verloren wordt beschouwd.

De AccountingSender is bedoeld om te worden gebruikt op een service−element voor het versturen van gegenereerde accounting−gegevens. De aangeboden accounting−gegevens bestaan uit attributen en bijbehorende waarden die voldoende informatie bevatten om bij aankomst te worden verwerkt. Tevens wordt ervan uitgegaan dat er voldoende attributen aanwezig zijn om een geldig accounting−bericht te vormen.

Verwerken accounting−gegevens

invoer: verzameling attributen

De aangeboden attributen worden, afhankelijk van het gehanteerde berichtenmodel, gebuffered om te worden verstuurd. Als er sprake is van event−driven polling zal een "accounting indication"−bericht worden verzonden om aan te geven dat accounting−gegevens beschikbaar zijn, mits dit nog niet eerder is gebeurd.

Verzenden "accounting request"−berichten

Bij het ontvangen van een "accounting poll"−bericht, het overlopen van de buffer of direct als er sprake is van event−driven accounting, zullen de accounting−gegevens in "accounting request"−berichten worden verstuurd. Verzonden berichten zullen in een speciale buffer worden geplaatst in afwachting van bevestiging door de accounting−server. Als een bevestiging op het bericht uitblijft zal het opnieuw worden verstuurd, zo mogelijk naar een andere accounting−server.

Verwerken "accounting poll"

Als een "accounting poll"−bericht wordt ontvangen, zullen alle gebufferde accounting−berichten die wachten om te worden verzonden, worden verstuurd.

Verwerken "accounting reply"−berichten

Inkomende "accounting reply"−berichten die een antwoord zijn op eerder verstuurde "accounting request"−berichten zullen worden verwerkt. Hierbij wordt het betreffende "accounting request"−bericht uit de buffer van berichten die wachten op bevestiging gehaald.

Toevoegen accounting−server

invoer: referentie accounting−server

De accounting−server wordt toegevoegd aan het eind van de lijst met mogelijke accounting−servers. Als bij het versturen van een bericht de accounting−server niet binnen een bepaalde termijn reageert, of als er bij de communicatie een fout optreedt, wordt een volgende accounting−server geprobeerd.

Verwijderen accounting−server

invoer: referentie accounting−server

De opgegeven accounting−server wordt verwijderd uit de lijst met mogelijke accounting−servers en er zullen geen berichten meer aan deze accounting−server worden verstuurd.

Instellingen berichtenmodel aanpassen

invoer: nieuwe instellingen

Bij initialisatie van de AccountingSender worden aan hand van het opgegeven berichtenmodel parameters voor buffering en transport opgegeven. Deze parameters zijn de maximale buffergrootte, de maximale tijd dat berichten worden gebufferd, of "accounting indication"−berichten moeten worden verstuurd en de timeout−waarden voor "accounting indication" en "accounting request"−berichten.

Omdat de AccountingSender ook moet reageren op inkomende berichten zal het een MessageHandler bij de AAAUnit registreren. De AccountingSender is echter zelf geen MessageHandler, omdat het MessageHandler−gedeelte alleen nut heeft in samenwerking met het verzendgedeelte.

6.9 AccountingHandler

De AccountingHandler is een MessageHandler die inkomende "accounting request"− en "accounting indication"−berichten opvangt en verwerkt. De AccountingHandler houdt een lijst van service−elements bij die een "accounting indication"−bericht hebben verstuurd en van service−elements die sowieso moeten worden gepolled.

Omdat het verwerken van accounting−gegevens applicatie−specifiek is, wordt het daadwerkelijk verwerken aan een afgeleide klasse overgelaten. Deze afgeleide klasse kan bijvoorbeeld de accounting−gegevens in een database verwerken, auditing uitvoeren, terugkoppelen naar een autorisatie−gedeelte of welke specifieke taak dan ook die in de applicatie nodig is.

De taken van de AccountingHandler bestaan uit:

- het bieden van opzet voor het verwerken van inkomende accounting−gegevens,
- het verwerken van "accounting indication"−berichten en
- het pollen van service−elements.

Hiervoor worden de volgende datastructuren bijgehouden:

- een lijst met service−elements die regelmatig moeten worden gepolled en
- een lijst met service−elements die een "accounting indication"−bericht hebben verstuurd.

Polling−berichten worden met bepaalde tussenpozen verstuurd, zodat het netwerk niet wordt overbelast door service−elements die tegelijkertijd reageren. Hiertoe wordt een eenvoudig scheduling−mechanisme gebruikt, waarbij

- een minimum polling−interval wordt bijgehouden.

Eventueel is het mogelijk om in een subklasse een andere vorm van scheduling van de polling−berichten te regelen, waarbij bijvoorbeeld ook aan de hand van de hoeveelheid inkomende "accounting request"−berichten polling kan worden uitgevoerd.

Toevoegen service−element voor polling

invoer: referentie service−element, polling−interval

Het opgegeven service−element wordt opgenomen in de lijst met te pollen service−elements. Het polling−interval geeft aan met welke tussenpozen het element moet worden gepolled.

Verwijderen service−element voor polling

invoer: referentie service−element

Het opgegeven service−element wordt verwijderd uit de lijst met te pollen service−elements. Het service−element zal niet meer met enige regelmaat worden gepolled.

Verwerken "accounting indication"−bericht

invoer: bericht, afzender

Van een inkomend "accounting indication"−bericht moet de afzender binnen een op te geven tijd worden gepolled. Het pollen wordt geregeld in de scheduling van de service−elements die met enige regelmaat moeten worden gepolled.

Verwerken "accounting request"

invoer: bericht, afzender

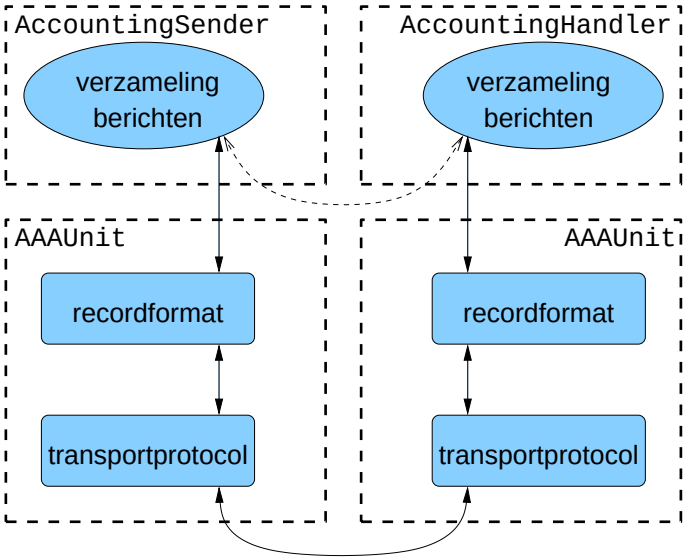
De accounting−gegevens in het bericht worden aangeboden om te worden verwerkt. Dit daadwerkelijk verwerken gebeurt in een subklasse. De subklasse moet er ook zorg voor dragen dat een "accounting reply"−bericht als reactie wordt gestuurd.

In de AccountingHandler wordt een eenvoudig scheduling−algoritme gebruikt. De AccountingHandler zal in elk geval minstens een opgeven minimumtijd wachten tussen het sturen van twee "accounting poll"−berichten. Aan service−elements die met enige regelmaat moeten worden gepolled, wordt zo snel mogelijk na het verlopen van het opgegeven polling−interval een "accounting poll"−bericht gestuurd. Aan service−elements die een "accounting indication"−bericht hebben verstuurd, wordt zo snel mogelijk een "accounting poll"−bericht gestuurd. Het service−element dat het langst geleden een "accounting indication" gestuurd heeft of waarbij het langst geleden het polling−interval is verlopen wordt als eerste een "accounting poll" gestuurd.

Een subklasse van de AccountingHandler is bedoeld om te worden toegepast op een accounting−server. Er moet dan nog wel in de subklasse worden gedefinieerd hoe inkomende accounting−gegevens moeten worden verwerkt.

6.10 Berichtenlaag

Met de AccountingSender en AccountingHandler is een verdere invulling aan de berichtenlaag gegeven. In *Figuur 6.3* is hiervan een symbolische representatie gegeven.



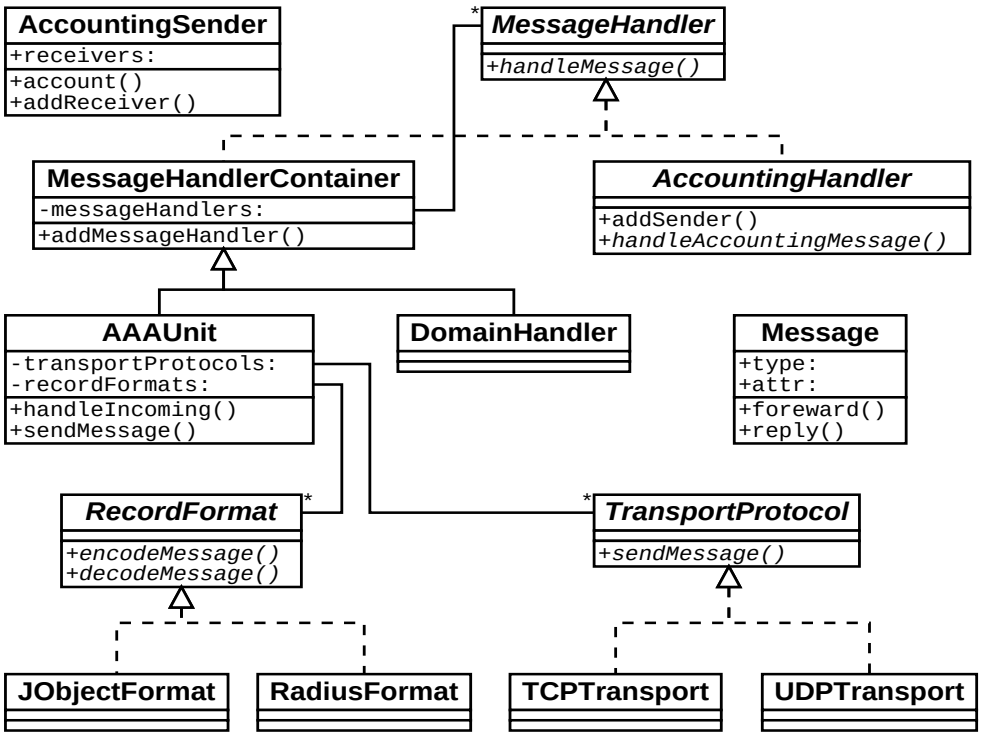
Figuur 6.3: plaats van modulen in lagenopdeling.

Omdat de AAAUnit alleen transport van berichten regelt is dit deel ook bruikbaar voor andere vormen van het versturen van bericht, zoals authenticatie en autorisatie. Als ook authenticatie en autorisatie moeten worden geregeld, kunnen deze worden geïmplementeerd door hun eigen verzender en handler.

De AccountingSender is te gebruiken om accounting−gegevens te versturen vanaf een service−element. De AccountingHandler om toegezonden accounting−gegevens te verwerken. Het daadwerkelijk genereren en verwerken van de accounting−gegevens valt echter buiten het bestek van dit framework en zal in een applicatie moeten gebeuren.

6.11 Klassendiagram

Alle genoemde software−modulen zullen hier nog eenmaal worden opgesomd en hun onderlinge relatie zal worden toegelicht. In *Figuur 6.4* zijn alle gebruikte klassen opgesomd en de inheritence aangegeven. Om het figuur eenvoudig te houden zijn niet alle methoden en attributen van alle klassen opgenomen.



Figuur 6.4: klassendiagram.

In het figuur worden abstracte klassen en methoden aangegeven met een cursief lettertype. Een gestippelde inheritence−relatie wil zeggen dat de ouder van de relatie een interface is waarvan de methoden worden geïmplementeerd. In het figuur zijn enkele voorbeelden van klassen opgenomen die in een implementatie kunnen worden gebruikt. Als voorbeeld−RecordFormat zijn het JObjectFormat en het RadiusFormat opgenomen. Als voorbeeld−TransportProtocol zijn het TCPTransport en het UDPTransport opgenomen. Verder is een DomainHandler opgenomen, die zou kunnen worden gebruikt om inkomende berichten te filteren.

klasse **AAAUnit** uitbreiding op MessageHandlerContainer	
transportProtocols	lijst met alle gebruikte transportprotocollen
recordFormats	lijst met alle gebruikte recordfromats
[messageHandlers]	van MessageHandlerContainer
handleIncoming()	decodering en afhandeling van inkomende connecties en berichten
sendMessage()	versturen van bericht
addTransportProtocol()	transportprotocol toevoegen
removeTransportProtocol()	transportprotocol verwijderen
addRecordFormat()	recordformat toevoegen
removeRecordFormat()	recordformat verwijderen
[addMessageHandler()]	van MessageHandlerContainer
[removeMessageHandler()]	van MessageHandlerContainer
[handleMessage()]	van MessageHandler

interface ***RecordFormat***	
encodeMessage()	transformatie van Message in binair formaat
decodeMessage()	transformatie van binair formaat naar Message

interface ***TransportProtocol***	
sendMessage()	verzenden van binair gecodeerd bericht

klasse **Message**	
type	berichttype
attr	lijst van attributen
sender	de afzender van het bericht
receiver	de ontvanger van het bericht
data	het bericht in binair formaat
recordFormat	het gebruikte RecordFormat
transportProtocol	het gebruikte TransportProtocol
forward()	nieuw bericht om door te sturen
reply()	nieuw bericht als antwoord

interface ***MessageHandler***	
handleMessage()	handel het ingekomen bericht af

klasse **MessageHandlerContainer** implementatie van MessageHandler	
messageHandlers	lijst van geregistreerde MessageHandlers
[handleMessage()]	van MessageHandler
addMessageHandler()	voeg een MessageHandler toe
removeMessageHandler()	verwijder een MessageHandler

abstracte klasse ***AccountingHandler*** implementatie van MessageHandler	
pollingQueue	lijst van service−elements die regelmatig moeten worden gepolled
indicationQueue	lijst van service−elements die een indication−bericht hebben gestuurd
minimumPollingInterval	minimumtijd tussen twee polling−berichten
[handleMessage()]	van MessageHandler: handel alleen "accounting request"− en ":accounting indication"−berichten af
handleIndication()	handel een "accounting indication"−bericht af
handleRequest()	handel een "accounting request"−bericht af (abstract)
addForPolling()	voeg een service−element toe aan de polling lijst
removeForPolling()	verwijder een service−element

klasse **AccountingSender**	
receivers	lijst van geregistreerde accounting−servers
awaitingSend	buffer van berichten die op verzenden wachten
awaitingReply	buffer van berichten die op antwoord wachten
maxBufferedSize	maximum aantal berichten wachtend op verzenden
maxBufferedTime	maximum tijd dat berichten moeten wachten op verzenden
sendIndication	flag om aan te geven of indication−bericht moet worden verzonden
requestTimeout	tijd waarna accounting−bericht verloren wordt beschouwd
indicationTimeout	tijd waarna indication−bericht als verloren wordt beschouwd
account()	aangeboden accounting−gegevens worden gebufferd of verzonden
addReceiver()	voeg een accounting−server toe
removeReceiver()	verwijder een accounting−server
setMaxBufferedSize()	stel de maximum buffergrootte in
setMaxBufferedTime()	stel de maximumtijd voor gebufferde berichten in
setSendIndication()	stel in of een indication bericht gestuurd moet worden
setRequestTimeout()	stel de timeout−waarde voor een "accounting request" in
setIndicationTimeout()	stel de timeout−waarde voor een "accounting indication" in

7. Implementatie

Er is een implementatie gemaakt in Java van het eerder beschreven framework. De implementatie bestaat uit het framework met een aantal geïmplementeerde transportprotocollen, recordformats en messagehandlers. In dit hoofdstuk wordt toegelicht hoe alle in het ontwerp beschreven software−modulen in de implementatie als klassen zijn geïmplementeerd.

Van alle gemaakte klassen wordt kort de signatuur kort toegelicht en de aanwezige methoden behandeld. In principe wordt alleen het publieke deel van de klassen en methoden behandeld. Voor enkele klassen zal ook worden ingegaan op de daadwerkelijke implementatie.

Verder zal worden uitgelegd hoe het verzenden en ontvangen van berichten in de AAAUnit plaatsvindt. Er zal worden aangegeven welke threads in het systeem worden gebruikt. Er zijn naast het hoofdsysteem met daarop de AccountingSender en AccountingHandler een aantal recordformats, transportprotocollen en messagehandlers geïmplementeerd die bij de toepassing van het framework van nut kunnen zijn. Deze hulpmodulen worden ook beschreven. Naast deze hulpmodulen is er ook een implementatie gemaakt van het RADIUS−protocol, zowel op transport−laag als op recordformat−laag. Dit zal uitvoeriger worden beschreven.

De gehele sourcecode, inclusief alle documentatie, is te vinden op de Internet−pagina van dit afstudeerwerk [aaapage].

7.1 nl.west.aaa.AAAUnit

De belangrijkste klasse in het geheel is de `AAAUnit`. Deze klasse combineert alle andere klassen en zorgt dat de communicatie hiertussen goed verloopt. Aan een `AAAUnit` kunnen transportprotocollen, recordformats en messagehandlers worden geregistreerd.

```
public class AAAUnit
    extends MessageHandlerContainer
```

Bij het aanmaken van een nieuwe `AAAUnit` wordt een `Identifier` meegegeven, die de `AAAUnit` in een netwerk identificeert. Hiermee is de `AAAUnit` op het netwerk bereikbaar. De `Identifier` is over het algemeen de naam van de host waarop de `AAAUnit` draait.

```
public AAAUnit(Identifier)
```

Aan de `AAAUnit` worden transportprotocollen toegevoegd met `addTransportProtocol()`. Op dezelfde manier kunnen recordformats en messagehandlers worden toegevoegd. Door het toevoegen worden ze bij de AAAUnit geregistreerd en kunnen ze direct worden gebruikt voor het verzenden en ontvangen van berichten.

```
public void addTransportProtocol(TransportProtocol)
public void addRecordFormat(RecordFormat)
public void addMessageHandler(MessageHandler)
```

Tevens zijn de methoden `removeTransportProtocol()`, `removeRecordFormat()` en `removeMessageHandler()` beschikbaar om aangemelde objecten af te melden.

Voor het versturen van een bericht wordt de `sendMessage()` methode gebruikt. Hierbij wordt een `Message` meegegeven en een bestemming. Als er bij het transport of het coderen een exceptie optreedt, wordt deze doorgegeven. Hiermee kan, afhankelijk van de exceptie, worden aangegeven dat een element niet bereikbaar is.

```
public void sendMessage(Message,Identifier)
    throws java.io.IOException
```

De in de `AAAUnit` gedefinieerde `handleIncoming()` methoden worden gebruikt om inkomende berichten en connecties van het `TransportProtocol` af te handelen. Deze inkomende berichten worden uiteindelijk door de `MessageHandlers` afgehandeld.

```
public void handleIncoming(byte[],Identifier,TransportProtocol)
public void handleIncoming(java.io.InputStream,Identifier,TransportProtocol)
```

Details over het gebruik van deze methoden is in de paragraaf over berichtafhandeling terug te vinden.

7.2 nl.west.aaa.Message

Deze klasse representeert een bericht in de berichtenlaag.

```
public class Message
```

Een bericht bestaat uit een berichttype en een verzameling attributen. De verschillende typen berichten staan vermeld in *Tabel 7.1*. Daarbij staat ook vermeld tussen welke elementen de berichten worden uitgewisseld.

naam	*van−>naar*	*omschrijving*
ACCOUNTING_REQUEST	client−>server	verzoek tot verwerken accounting−gegevens
ACCOUNTING_REPLY	server−>client	bevestiging van accounting−bericht
ACCOUNTING_POLL	server−>client	server vraagt alle accounting−berichten te versturen
ACCOUNTING_INDICATION	client−>server	indicatie dat accounting gegevens beschikbaar zijn

Tabel 7.1: geïmplementeerde berichttypen

Attributen van een bericht bestaan uit een naam en een waarde. De naam van een attribuut is van type `java.lang.String` en de waarde kan elke subklasse van `java.lang.Object` zijn. Over het algemeen zal de waarde een `String` zijn of van en naar een `String` te converteren zijn.

Een nieuw `Message` object kan worden aangemaakt door het type van het bericht en een verzameling attributen op te geven. De verzameling attributen wordt opgegeven met een `Hashtable`, waarin de attribuutnaam als `key` optreedt en de attribuutwaarde als `value`.

```
public Message(int,java.util.Hashtable)
```

Een bericht kan ook worden gegenereerd op basis van een ander bericht, bijvoorbeeld als het een antwoord is op een ontvangen bericht,

```
public Message reply(int,java.util.Hashtable)
public Message forward(Identifier)
```

Verder zijn er nog een aantal methoden aanwezig, waarmee informatie over het bericht kan worden ingewonnen. Het type bericht en de attributen kunnen worden opgevraagd en het is mogelijk om te bepalen of een bericht een antwoord is op een ander bericht.

```
public int getMessageType()
public java.util.Hashtable getAttributes()
public java.lang.Object getAttribute(java.lang.String)
public boolean isReplyTo(Message)
```

Ook zijn er nog enkele attributen aanwezig die bij de communicatie gebruikt kunnen worden.

```
public byte[] data;
public RecordFormat recordFormat;
public TransportProtocol transportProtocol;
public Identifier sender;
public Identifier receiver;
```

Het attribuut `data` verwijst naar de gecodeerde variant van het bericht, `recordFormat` naar het gebruikte of te gebruiken `RecordFormat`, `transportProtocol` naar het gebruikte of te gebruiken `TransportProtocol`, `sender` naar de afzender van het bericht en `receiver` naar de ontvanger van het bericht.

7.3 nl.west.aaa.TransportProtocol

Het `TransportProtocol` wordt gebruikt voor het versturen en ontvangen van berichten. Alle transportprotocollen die kunnen worden gebruikt, moeten het interface `TransportProtocol` implementeren.

```
public interface TransportProtocol
```

Een `TransportProtocol` kan worden gebruikt voor het versturen van een gecodeerd bericht naar een ontvanger. Een implementatie zal in haar documentatie moeten specificeren of het transportprotocol de ontvangst van een bericht bevestigt. Als het transportprotocol een fout signaleert zal het dit middels een `IOException` melden.

```
public void sendMessage(byte[],Identifier)
    throws java.io.IOException
```

Het `TransportProtocol` wacht tevens op inkomende berichten en stuurt deze naar de `AAAUnit`. Als het `TransportProtocol` bij de `AAAUnit` wordt geregistreerd, roept de `AAAUnit` de `setReceiver()` methode van het `TransportProtocol` aan om aan te geven naar welke `AAAUnit` inkomende berichten gestuurd moeten worden.

```
public void setReceiver(AAAUnit receiver)
```

Voor het doorgeven van inkomende verbindingen zijn twee methoden `handleIncoming()` in de `AAAUnit` aanwezig. De ene geeft een buffer met gelezen bytes door en kan worden gebruikt als er een afgerond bericht binnenkomt. Deze methode kan worden gebruikt bij datagram gebaseerde protocollen. Bij de andere methode wordt een `java.io.InputStream` opgegeven waaruit de berichten kunnen worden gelezen. Deze methode zal meer voor de hand liggen bij verbindingsgerichte protocollen. In beide gevallen wordt ook een referentie naar de afzender en het protocol meegegeven.

In de stream variant zal de `AAAUnit` berichten uit de stream lezen zolang deze gegevens bevat. Als er geen berichten meer in de stream aanwezig zijn (de stream is gesloten) zal deze methode terugkeren. De andere methode zal terugkeren zodra alle data tot berichten zijn verwerkt. Deze methoden wachten niet op het verwerken van de inkomende berichten, alleen het decoderen zal hier worden uitgevoerd. Als de berichten zijn gedecodeerd, worden ze gequeued om te worden verwerkt. De byte−buffer−variant zal dus in het algemeen snel terugkeren.

Klassen die `TransportProtocol` momenteel implementeren:

```
public class TCPTransport
    implements TransportProtocol

public class UDPTransport
    implements TransportProtocol

public class RadiusTransport
    extends UDPTransport
    implements TransportProtocol
```

`TCPTransport` levert een eenvoudig verbindingsgericht transportprotocol waarbij bidirectioneel verkeer over een enkele verbinding plaatsvindt. `UDPTransport` zorgt voor een datagramverbinding waarbij elk bericht in een enkel UDP−pakket wordt verpakt. Doordat UDP−pakketten een maximale afmeting hebben kan dit voor problemen zorgen bij erg grote berichten.

7.4 nl.west.aaa.RecordFormat

Het `RecordFormat` wordt gebruikt voor het coderen en decoderen van berichten. Net als het `TransportProtocol` is ook het `RecordFormat` een interface.

```
public interface RecordFormat
```

Als de `AAAUnit` een bericht wil versturen, wordt het eerst aan een `RecordFormat` aangeboden. Het `RecordFormat` vertaalt het bericht in een voor dat `RecordFormat` specifiek formaat.

```
public byte[] encodeMessage(Message)
```

Hierbij kunnen encryptie, compressie en dergelijke worden gebruikt. Via het `TransportProtocol` kan het bericht nu worden verstuurd. Bij het ontvangen van een bericht of verbinding wordt de `decodeMessage()` methode gebruikt.

```
public Message decodeMessage(java.io.InputStream,Identifier)
    throws java.io.IOException
public Message decodeMessage(byte[],Identifier)
```

Hierbij wordt aan het `RecordFormat` gevraagd, het ingekomen bericht te reconstrueren. Een bericht kan gecodeerd in een stream aanwezig zijn of opgeslagen zitten in een reeks bytes. Een `Identifier`, die de afzender van het bericht identificeert, wordt meegegeven om eventuele encryptiesleutels op te zoeken. Als het `RecordFormat` een geldig bericht kan reconstrueren, moet het dit bericht retourneren, als het dit niet kan moet de methode de `null`−waarde teruggeven, zodat de `AAAUnit` een ander `RecordFormat` kan proberen. Een exceptie kan worden gebruikt als er een fout optreedt bij het lezen van de gegevens. Een exceptie mag alleen worden gebruikt als duidelijk is dat de gegevens het huidige recordformat betreffen.

Klassen die `RecordFormat` momenteel implementeren:

```
public class JObjectFormat
    implements RecordFormat

public class RadiusFormat
    implements RecordFormat
```

Het `JObjectFormat` gebruikt "Java Object Serialization" om objecten te transporteren en `RadiusFormat` gebruikt het RADIUS formaat om berichten vast te leggen.

7.5 nl.west.aaa.MessageHandler

De `MessageHandler` zorgt voor het afhandelen van inkomende berichten. Deze berichten kunnen inkomende requests zijn of antwoorden zijn op eerder verzonden berichten. Klassen die inkomende berichten afhandelen implementeren `MessageHandler`.

```
public interface MessageHandler
```

Inkomende berichten worden door de `AAAUnit` middels de `handleMessage()` methode aan de `MessageHandler` aangeboden. De methode geeft aan of het inkomende bericht juist is afgehandeld.

```
public boolean handleMessage(Message,AAAUnit)
```

Als een `MessageHandler` het bericht niet kan afhandelen, zal het dit doorgeven door `false` terug te geven. De `AAAUnit` kan dan een andere `MessageHandler` proberen. Aan de `MessageHandler` wordt de `AAAUnit` die het bericht ontvangen heeft meegegeven om eventuele antwoorden op het bericht te kunnen versturen.

7.6 nl.west.aaa.MessageHandlerContainer

Er kan een boomstructuur van `MessageHandlers` worden gemaakt door de `MessageHandlerContainer`, of een subklasse daarvan te gebruiken.

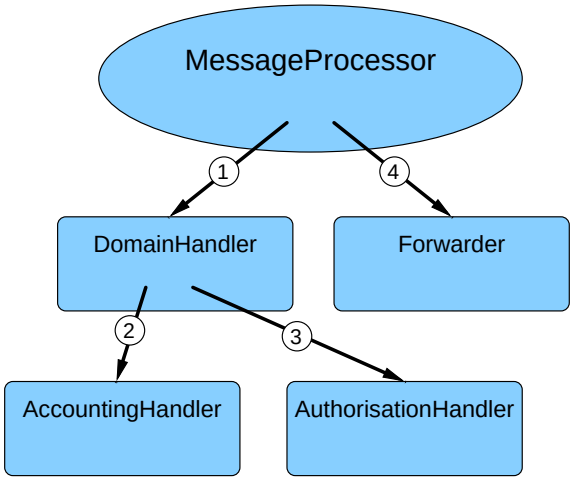
```
public class MessageHandlerContainer
    implements MessageHandler
```

Bij deze klasse kan een `MessageHandler` worden geregistreerd via de `addMessageHandler()` methode.

```
public void addMessageHandler(MessageHandler)
```

Bij het afhandelen van een inkomend bericht, via de `handleMessage()` methode, zal deze klasse één voor één de opgegeven handlers proberen totdat een handler positief reageert. Een subklasse hiervan kan vóór het proberen van de geregistreerde handlers het bericht filteren, zodat een beslissingsboom voor inkomende berichten ontstaat zoals geïllustreerd in *Figuur 7.1*.

In *Figuur 7.1* zal de `MessageProcessor` (een subklasse van `MessageHandlerContainer`) als eerste het bericht aanbieden aan de `DomainHandler` (1). De `DomainHandler` zal testen of het bericht voor een bepaald domijn bestemd is. Als dit het geval is zal de `DomainHandler` het bericht aan de `AccountingHandler` (2) en eventueel de `AuthorisationHandler` doorspelen. Als het bericht niet voor het opgegeven domein bestemd is, of als zowel de `AccountingHandler` als de `AuthorisationHandler` het bericht niet kunnen afhandelen, zal de `DomainHandler` aan de `MessageProcessor` doorgeven dat hij het bericht niet kan verwerken. De `MessageProcessor` zal dan de `Forwarder` het bericht laten afhandelen (4).



Figuur 7.1: afhandelingsboom van ingekomen berichten

7.7 nl.west.aaa.MessageProcessor

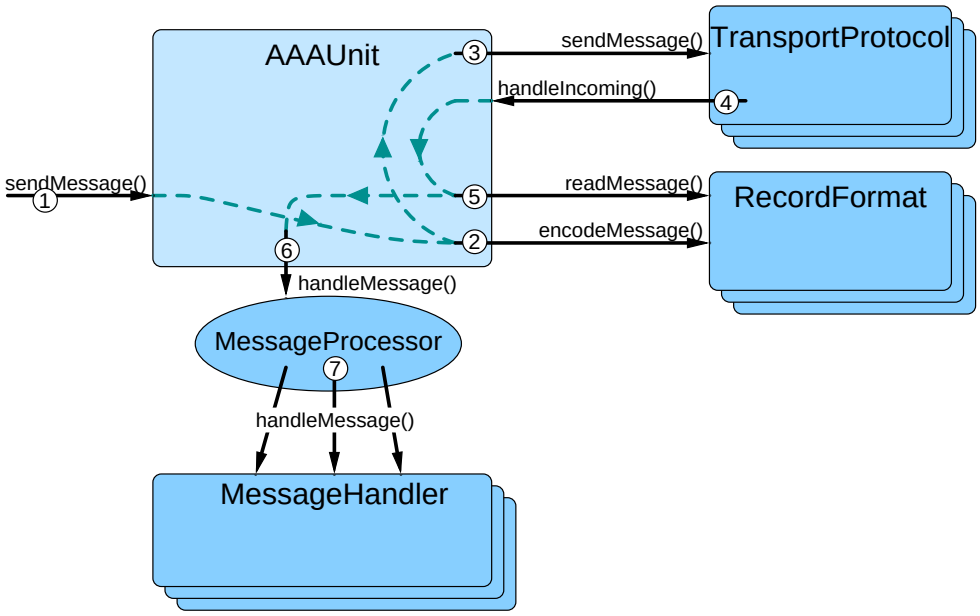
De `MessageProcessor` klasse wordt intern door de `AAAUnit` gebruikt om berichten in een aparte thread af te handelen.

```
class MessageProcessor
    extends MessageHandlerContainer
```

Deze klasse gedraagt zich precies als een gewone `MessageHandlerContainer`, behalve dat de `handleMessage()` methode direct terugkeert met `true` als waarde. Het ingekomen bericht wordt gebufferd en in een eigen thread afgehandeld.

7.8 Berichtafhandeling

In *Figuur 7.2* staat schematisch weergegeven hoe het versturen en ontvangen van berichten in de `AAAUnit` wordt afgehandeld.



Figuur 7.2: versturen en ontvangen van berichten in de AAAUnit

Versturen van bericht:

Bij het aanbieden van een nieuw bericht voor verzenden wordt de `sendMessage()` methode (1) van de `AAAUnit` aangeroepen. Het bericht wordt met behulp van het `RecordFormat`, dat in de Message is opgegeven, vertaald in binaire data (2). Als in de `Message` geen `RecordFormat` is gedefinieerd, wordt een opgegeven default gebruikt. Hierna wordt het gecodeerde bericht verstuurd met een `TransportProtocol` (3) waarvoor hetzelfde geldt. De methode retourneert zogauw het bericht is verstuurd. Er is, afhankelijk van het gebruikte `TransportProtocol`, nog geen garantie dat het bericht door de ontvanger is ontvangen. Hiervoor is het nodig het bericht met een antwoord bevestigd te krijgen.

Ontvangen van bericht:

Bij het ontvangen van een bericht meldt het `TransportProtocol` dit aan bij de `AAAUnit` (4). De `AAAUnit` decodeert het bericht door gebruik te maken van de geregistreerde `Recordformats` (5). Daarna wordt het gedecodeerde bericht in de `MessageProcessor` (6) in een queue opgeslagen, waarna de methode retourneert. Het bericht is dan klaar om door de thread in de `MessageProcessor` te worden verwerkt. Deze aparte thread zorgt ervoor dat het verwerken van inkomende berichten allemaal in dezelfde thread plaatsvindt en dat de `handleIncoming()` methode in de `AAAUnit` snel retourneert, zodat er niet voor elk inkomend bericht een aparte thread hoeft te worden opgestart.

Afhandelen van ingekomen bericht:

De `MessageProcessor` draait in een eigen thread, zodat inkomende connecties snel kunnen worden afgehandeld. De `MessageProcessor` biedt alle berichten die in de queue aanwezig zijn aan de geregistreerde `MessageHandlers` aan (7).

7.9 nl.west.aaa.AccountingSender

De `AccountingSender` werkt als een wrapper om de `AAAUnit` heen. De `AccountingSender` kan buffering regelen, indication berichten versturen en inkomende polling requests afhandelen. De `AccountingSender` is bedoeld om op het service−element te gebruiken.

```
public class AccountingSender
```

De AccountingSender kan in een aantal modellen opereren.

`AccountingSender.EVENT_DRIVEN`

In dit model worden accounting−berichten direct verzonden zogauw ze binnenkomen.

`AccountingSender.EVENT_DRIVEN_BATCHING`

In dit model worden berichten opgeslagen totdat een bepaald maximum is bereikt. Als het maximum is bereikt, worden alle opgeslagen berichten verstuurd.

`AccountingSender.POLLING`

In dit model worden berichten opgeslagen totdat de accounting server met een `Message.ACCOUNTING_POLL` bericht aangeeft dat de gebufferde gegevens moeten worden verstuurd.

`AccountingSender.EVENT_DRIVEN_POLLING`

Dit model breidt het polling−model uit door het sturen van `Message.ACCOUNTING_INDICATION` berichten als accounting gegevens klaar staan om te worden verstuurd. Een indication−bericht wordt alleen verzonden als er nog niet recentelijk een ander indication−bericht is verzonden.

Het te gebruiken model wordt opgegeven bij het aanmaken van een `AccountingSender`−object.

```
public AccountingSender(int,AAAUnit)
```

De opgegeven `int` geeft het model aan, terwijl de `AAAUnit` wordt gebruikt voor het versturen en ontvangen van de berichten. Bij de `AAAUnit` wordt een `MessageHandler` geregistreerd die antwoorden op verstuurde berichten en inkomende polling−berichten opvangt.

Het is mogelijk om na de creatie van het AccountingSender−object de transport−parameters aan te passen met de setMaxBufferedSize(), setMaxBufferedTime(), setSendIndication(), setRequestTimeout() en setIndicationTimeout() methoden.

```
public void setMaxBufferSize(int)
public void setMaxBufferTime(long)
public void setSendIndication(boolean)
public void setRequestTimeout(long)
public void setIndicationTimeout(long)
```

Om deze waarden op te vragen zijn de getMaxBufferSize(), getMaxBufferTime(), getSendIndication(), getRequestTimeout() en getIndicationTimeout() methoden beschikbaar.

Om accounting−informatie te versturen is de account() methode beschikbaar. Hierbij worden te versturen accounting−gegevens als een verzameling attributen meegegeven. Deze attributen worden gebruikt om het accounting−bericht te vormen.

```
public void account(java.util.Hashtable)
```

De gegevens worden, afhankelijk van de opgegeven parameters, gebuffered en verstuurd in een Message.ACCOUNTING_REQUEST. Als dat aangegeven is, wordt ook nog een Message.ACCOUNTING_INDICATION bericht gestuurd om aan te geven dat er gegevens beschikbaar zijn. De berichten worden gestuurd naar één van de opgegeven accounting−servers. Een accounting−server kan worden opgegeven met de addReceiver() methode en uit de lijst worden gehaald met de removeReceiver() methode.

```
public void addReceiver(Identifier)
public void removeReceiver(Identifier)
```

Als een server binnen de opgegeven tijd niet reageert op een bericht, of als er een fout optreedt bij de communicatie met een server, zal een volgende van de opgegeven servers worden gebruikt om berichten te versturen. Bij het binnenkomen van een Message.ACCOUNTING_POLL bericht zal de standaard−server worden ingesteld op de server waar het poll−bericht van afkomstig is.

7.10 nl.west.aaa.AccountingHandler

De AccountingHandler is een MessageHander die inkomende Message.ACCOUNTING_REQUEST en Message.ACCOUNTING_INDICATION berichten opvangt en verwerkt. Omdat het daadwerkelijk afhandelen van de gegevens applicatie−specifiek is en dus nog niet in deze klasse kan worden gedefinieerd, is de klasse abstract.

```
public abstract class AccountingHandler
    implements MessageHandler
```

Omdat er ook polling van service−elements nodig is, wordt er bij het creëren van een nieuw AccountingHandler een referentie naar de AAAUnit opgegeven.

```
public AccountingHandler(AAAUnit)
```

Inkomende berichten worden afgehandeld door de handleIncoming() methode. Als het een Message.ACCOUNTING_REQUEST bericht betreft, wordt dit doorgestuurd aan de handleRequest() methode.

```
protected abstract boolean handleRequest(Message,AAAUnit)
public boolean handleMessage(Message,AAAUnit)
```

De handleRequest() methode moet in een subklasse zijn gedefinieerd. De subklasse moet op het binnenkomende bericht een antwoord versturen.

Met de addForPolling() en removeForPolling() methoden wordt een service−element opgegeven, danwel afgemeld om regelmatig te worden gepolled.

```
void addForPolling(Identifier,long)
void removeForPolling(Identifier)
```

Het daadwerkelijk pollen gebeurt in een aparte thread. In deze thread worden ook ingekomen `Message.ACCOUNTING_INDICATION` berichten beantwoord.

7.11 Threads

Omdat meerdere connecties tegelijk moeten worden afgehandeld, er met enige regelmaat moet worden gepolled en er moet worden gewacht op antwoorden, is het gebruik van threads noodzakelijk. Behalve de thread die de AAAUnit en andere objecten initieert zijn er de volgende threads:

MesageProcessor.Processor

Deze thread zorgt dat ingekomen en gedecodeerde berichten door de `MesageHandlers` worden verwerkt.

AccountingHandler.Poller

Deze thread zorgt dat opgegeven service−elements en service−elements die een `Message.ACCOUNTING_INDICATION` bericht hebben gestuurd worden gepolled.

AccountingSender.Poller

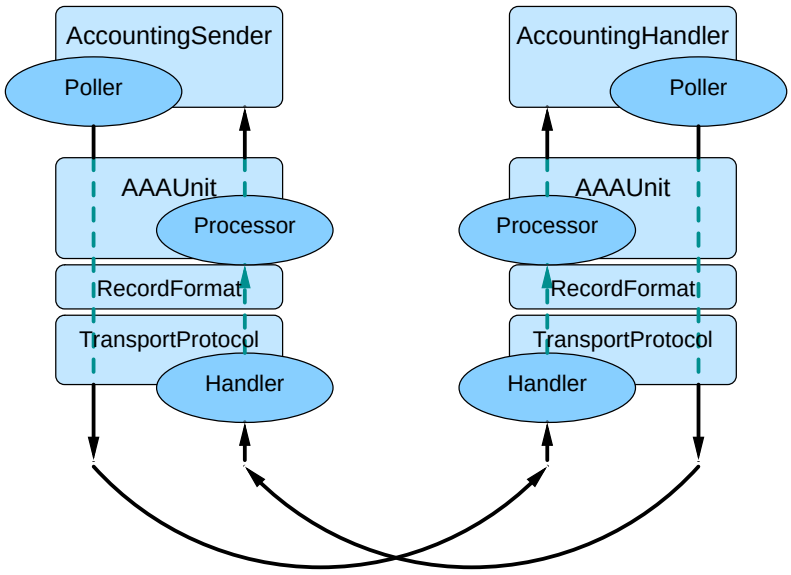
Deze thread zorgt ervoor dat gebufferde gegevens worden verstuurd en dat timeout−waarden worden gecontroleerd.

TransportProtocol.Handler

In een TransportProtocol is over het algemeen een thread aanwezig, die inkomende connecties afhandelt. Voor transportprotocollen die een stream−verbinding tot stand brengen is het in veel gevallen wenselijk om per inkomende verbinding een nieuwe thread op te starten, zodat de oorspronkelijke thread beschikbaar blijft voor nieuwe verbindingen.

Het opstarten van een nieuwe thread kost relatief veel resources en het is dus in verband met schaalbaarheid wenselijk om het starten van nieuwe threads tot een minimum te beperken. De threads die in het TransportProtocol worden gebruikt om per connectie de berichten te lezen zijn de enige threads die regelmatig worden gecreëerd. Hiervoor is het mogelijk om een vaste verzameling van threads te starten, die bij een inkomende connectie kunnen worden geactiveerd. Dit heeft als bijkomend voordeel dat de hoeveelheid inkomende connecties, die tegelijkertijd kan worden afgehandeld, beperkt is. Hiermee kan een denial−of−service−attack deels worden voorkomen.

In Figuur 7.3 zijn de verschillende threads schematisch weergegeven. Voor de duidelijkheid is de MessageProcessor als aparte klasse weggelaten en ondergebracht bij de AAAUnit.



Figuur 7.3: verschillende threads in het accounting−proces.

7.12 Hulpmodulen

Om het framework beter toepasbaar te maken, zijn er als voorbeeld een aantal eenvoudige klassen geïmplementeerd. Als eerste zijn met `TCPTransport` en `UDPTransport` twee eenvoudige transportprotocollen geïmplementeerd.

```
public class TCPTransport
    implements TransportProtocol

public class UDPTransport
    implements TransportProtocol
```

Deze klassen worden gebruikt om berichten via een TCP− respectievelijk UDP−protocol te versturen. Bij het creëren worden poortnummers opgegeven, waarover de communicatie moet verlopen.

```
public TCPTransport(int,int)
    throws java.io.IOException

public UDPTransport(int,int)
    throws SocketException
```

Bij het creëren worden bij deze protocollen twee poorten opgegeven. De eerste poort geeft aan op welke poort naar binnenkomende berichten of verbindingen moet worden geluisterd. De tweede poort geeft aan naar welke poort de berichten moeten worden verstuurd.

Als eenvoudig `RecordFormat` is `JObjectFormat` gemaakt. Hierbij worden berichten gecodeerd door gebruik te maken van "Java Object Serialization". Het recordformat bestaat uit een eenvoudige header (een identificerend `java.lang.String` object), waarna het berichttype dat als `int` wordt geschreven, gevolgd door de attributenverzameling die als een `java.util.Hashtable` object wordt geschreven.

```
public class JObjectFormat
    implements RecordFormat
```

Het `JObjectFormat` regelt geen encryptie of andere zaken en is bedoeld als snelle implementatie van een eenvoudig `RecordFormat` voor test−doeleinden.

Hiernaast zijn nog enkele `MessageHandlers` geïmplementeerd. `DebugHandler` is voor debug−doeleinden, de `DomainHandler` is een `MessageHandlerContainer` waarmee een filter wordt gelegd op berichtinhoud en de `Replyhandler` is bedoeld om antwoorden van verstuurde berichten te vinden.

```
public class DebugHandler
    implements MessageHandler
```

De `handleMessage()` methode van de `DebugHandler` geeft van alle inkomende berichten een melding in de uitvoer van het programma. De `handleMessage()` methode geeft aan de `AAAUnit` door dat deze het bericht niet heeft kunnen afhandelen. De `AAAUnit` probeert dan de volgende `MessageHandler`, zodat de DebugHandler de functionaliteit van het systeem niet beïnvloedt. De `DebugHandler` kan, als hij als eerste bij de `AAAUnit` wordt geregistreerd, gebruikt worden om alle inkomende berichten af te drukken.

```
public class DomainHandler
    extends MessageHandlerContainer
    implements MessageHandler

public void addDomain(String)
```

De `DomainHandler` zal van alle inkomende berichten eerst controleren of het domein dat in het "uid"−attribuut (user−id) vermeld staat vermeld, zich in de verzameling van opgegeven domeinen bevindt. Er wordt van uitgegaan dat het "uid"−attribuut aan een "user@domain"−formaat voldoet. Als het domein een opgegeven domein betreft, zal het bericht aan alle geregistreerde `MessageHandlers` worden gestuurd. Als dit niet het geval is, zal worden aangegeven dat de `DomainHandler` het bericht niet kan verwerken.

```
public class ReplyHandler
    implements MessageHandler
```

```
public void prepareForReply(Message)

public Message waitForReply(Message,long)
```

Met de `Replyhandler` is een `MessageHandler` gedefinieerd waarmee antwoorden van verzonden berichten kunnen worden afgewacht. Voordat een bericht wordt verzonden, moet het worden aangemeld met de `prePareForReply()` methode. Met de `waitForReply()` methode wordt het antwoord geretourneerd. Er kan worden opgegeven hoe lang er op het bericht moet worden gewacht.

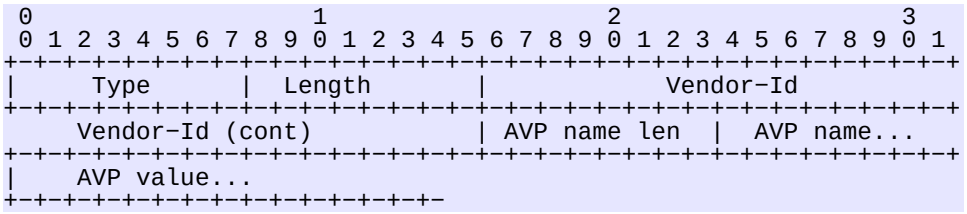
7.13 Radius

Om de mogelijkheden van het framework te testen is een implementatie gemaakt van het RADIUS−protocol. RADIUS is de standaard voor gebruik in inbel−omgevingen. Daarom is het logisch om RADIUS in een accounting−server te ondersteunen.

Bij RADIUS zijn transport en recordformat niet goed scheidbaar, doordat autorisatie−berichten naar een andere UDP−poort worden verstuurd dan accounting−berichten. Dit heeft te maken met het feit dat accounting pas later aan RADIUS is toegevoegd. Hierdoor is het in de implementatie van het `TransportProtocol` nodig om naar de inhoud van het te versturen bericht te kijken. De klasse `RadiusTransport`, die is afgeleid van de `UDPTransport` klasse, is dus ook alleen bruikbaar als de `RadiusFormat` klasse als recordformat wordt gebruikt. Andersom is het overigens wel mogelijk om berichten die met `RadiusFormat` zijn vastgelegd met een ander transportprotocol te versturen.

RADIUS heeft een beperkte verzameling berichten waardoor het niet toepasbaar is in een polling of event−driven polling model. RADIUS heeft standaard ook een beperkte verzameling attributen. Hiervoor is een oplossing gevonden door gebruik te maken van leverancier−specifieke attributen. Er wordt hierbij gebruik gemaakt van de Vendor−Specific AVP in RADIUS. Doordat het gebruik van de leverancier−specifieke attributen niet is gestandaardiseerd, zijn gegevens die hierin zijn vastgelegd, niet in andere RADIUS implementaties bruikbaar. Attributen die in RADIUS aanwezig zijn kunnen wel in andere systemen worden gebruikt.

In *Figuur 7.4* is het gebruik van de Vendor−Specific AVP aangegeven.



Figuur 7.4: Uitbreiding van attributen

Het Type−veld is 26 om Vendor−Specific aan te geven. Het Length−veld geeft de volledige lengte van de AVP aan. Als Vendor−Id is "west" gebruikt. Deze waarde is echter niet vastgelegd als geldige leverancier. Hierna volgt de lengte van de naam van het attribuut, gevolgd door de attribuutnaam in UTF−8 formaat. De waarde van het attribuut is als string in UTF−8 codering vastgelegd.

Het recordformat van RADIUS is vastgelegd in de `RadiusFormat` klasse.

```
public class RadiusFormat
    implements RecordFormat
```

De `RadiusFormat` klasse implementeert de verplichte `decodeMessage()` en `encodeMessage()` methoden.

```
public Message decodeMessage(byte[],Identifier)
public Message decodeMessage(java.io.InputStream,Identifier)
public byte[] encodeMessage(Message)
```

Hiernaast zijn er nog enkele methoden beschikbaar om het genereren van geldige RADIUS−berichten te vereenvoudigen.

```
public static int getAVPCode(java.lang.String)
public static java.lang.String getAVPName(int)
public static int getAVPType(int)
public static int getMessageType(int)
public static int getRadiusMessageCode(int)
public static byte[]newAuthenticator()
```

De `getAVPCode()`, `getAVPName()`, `getAVPType()`, `getMessageType()` en `getRadiusMessageCode()` methoden worden gebruikt om de vertaling tussen RADIUS−attributen en berichten te regelen. De `newAuthenticator()` methode kan gebruikt worden om een nieuwe authenticator in een RADIUS−bericht te maken. Verder zijn de mogelijke RADIUS−attributen en berichttypen als constanten opgenomen.

Het RadiusFormat gebruikt intern een RadiusMessageBuffer om een bericht op te bouwen. Deze klasse vormt een eenvoudige interface om een binair RADIUS−bericht op te bouwen.

```
class RadiusMessageBuffer
```

Deze klasse kan worden gebruikt om ingekomen berichten te lezen en nieuwe berichten te versturen. Het biedt methoden voor het lezen en schrijven van berichttype, authenticator, message−id en de AVP's in de verschillende formaten.

Het transport van RADIUS−berichten wordt door de klasse `RadiusTransport` geregeld. Opdat RADIUS transport over UDP verloopt, is het een eenvoudige uitbreiding op de `UDPTransport` klasse.

```
public class RadiusTransport
    extends UDPTransport
    implements TransportProtocol
```

Zoals gezegd is `RadiusTransport` alleen bruikbaar voor het transport van RADIUS−berichten. Omdat accounting−berichten naar een andere poort gaan, wordt in de `sendMessage()` methode gecontroleerd of het om een accounting−bericht of een autorisatie−bericht gaat.

```
public void sendMessage(byte[],Identifier)
    throws java.io.IOException
```

`RadiusTransport` is dus niet te gebruiken voor transport van berichten in andere recordformats. `RadiusFormat` is echter wel geschikt om berichten te coderen die op een andere manier worden verstuurd.

8. Resultaten

In dit hoofdstuk worden de resultaten van het onderzoek gepresenteerd en wordt de gekozen oplossing kort toegelicht. Er wordt toegelicht in welke mate het framework aan de doelstelling en de requirements voldoet. Ook tekortkomingen aan het framework worden behandeld. Voor enkele tekortkomingen wordt aangegeven hoe deze kunnen worden verholpen.

Het doel van het onderzoek is het ontwerpen en ontwikkelen van een accounting−framework waarmee het mogelijk is om verschillende accounting−protocollen te combineren, zodat het framework voor meerdere accounting−toepassingen bruikbaar is. Met deze implemantatie is een framework gedefinieerd waarbinnen verschillende accounting−protocollen kunnen worden geïntegreerd. Hierdoor is het framework in te passen in meerdere omgevingen.

Het framework bestaat uit twee onderdelen. Het eerste deel regelt het versturen en ontvangen van berichten met behulp van de verschillende protocollen. Het splitst een accounting−protocol in eerste instantie in een recordformat en transportprotocol. De AAAUnit zorgt dat berichten, die zijn opgebouwd uit attributen, met een recordformat worden vastgelegd en met een transportprotocol worden verstuurd. Inkomende verbindingen en ontvangen berichten van het transportprotocol worden met behulp van het juiste recordformat gedecodeerd. De AAAUnit biedt hulpmiddelen voor het verwerken van inkomende berichten. Hiermee is de AAAUnit bruikbaar als basis voor andere berichtuitwisselende systemen voor gerelateerde toepassingen zoals authenticatie en autorisatie. Door het gebruiken van een abstracte representatie van berichten kan het transport onafhankelijk van het gebruikte accounting−protocol geschieden.

Het tweede deel, dat het uitwisselen van accounting−berichten op een berichten−niveau regelt, is specifiek voor accounting ontwikkeld. In dit deel worden de verschillende uit te wisselen berichten vastgelegd. Accounting−gegevens worden hier uitgewisseld met behulp van een event−driven, polling of event−driven polling model. Tevens zijn hier zaken als batching, fail−over, en retry−mechanismes gedefinieerd. Dit deel is verder uit te breiden, waarbij het bijvoorbeeld mogelijk is om andere vormen van scheduling te gebruiken.

Deze delen kunnen in een toepassing worden gecombineerd om een volledig accounting−systeem te vormen. Het framework is toepasbaar aan zowel de client−kant als de server−kant. Ook is het mogelijk om het framework in een proxy−omgeving te gebruiken. Aan de client−kant worden accounting−gegevens aangeboden en aan de server−kant komen de accounting−gegevens weer beschikbaar om te worden verwerkt. De AAAUnit is op alle elementen op dezelfde manier te gebruiken, terwijl er op de accounting−server en op het service−element een specifieke module is die zorgt voor de uitwisseling op het berichtenniveau. Uiteindelijk is het gebruik van het accounting−protocol en het gebruikte berichtenmodel transparant voor de applicatie.

Het is mogelijk om verschillende accounting−protocollen in het framework te integreren. Dit is in het vorige hoofdstuk geïllustreerd door het maken van een implementatie van RADIUS voor het framework. Bij RADIUS zijn recordformat en transportprotocol niet direct eenvoudig te scheiden en daarom maakt het een interessante test−case. Het RADIUS−transportprotocol kan alleen in combinatie met het RADIUS−recordformat worden gebruikt, terwijl het RADIUS−recordformat wel in combinatie met andere transportprotocollen is te gebruiken. Voor het testen zijn verder nog enkele eenvoudige recordformats en transportprotocollen geïmplementeerd.

Het framework is getest door het in een eenvoudige omgeving te plaatsen waarin een simpel service−element eenvoudige dummy−berichten produceert. De accounting−server ontvangt deze berichten, drukt het resultaat af op het scherm en verstuurt een bevestiging van de ontvangen gegevens. Het service−element verwerkt deze bevestiging door het bericht uit de buffers te verwijderen. Dit is getest voor de verschillende berichtenmodellen, recordformats en transportprotocollen voor zover de combinatie is ondersteund. Zo is bij RADIUS bijvoorbeeld alleen een event−driven berichtenmodel mogelijk.

Naast deze eenvoudige opstelling is het framework ook getest door het integreren van het client−gedeelte in een webserver om daar de logging−informatie naar een accounting−server te transporteren. Er is een eenvoudige klasse gemaakt die vanuit de Apache webserver wordt aangeroepen. De klasse leest de logging−gegevens en maakt per opgevraagde pagina een accounting−bericht dat met behulp van de AAAUnit wordt verstuurd. Aan de accounting−server−kant is een eenvoudig billing programma gemaakt dat de opgevraagde pagina’s in verschillende categorieën verdeelt en aan de hand daarvan statistieken laat zien. Meer informatie hierover is te vinden op de Internet−pagina van dit afstudeerwerk [aaapage].

Aan de hand van de implementatie van RADIUS en de verschillende testomgevingen is gekeken in welke mate het framework aan de gestelde requirements voldoet. Dit zal in de volgende paragraaf worden beschreven. Bij het implementeren zijn ook enkele beperkingen en punten voor verder onderzoek naar voren gekomen die daarna zullen worden behandeld.

8.1 Toetsing requirements

In deze paragraaf zal van het geïmplementeerde framework worden aangegeven in welke mate het voldoet aan de in hoofdstuk vier genoemde requirements. Over het algemeen blijkt dat het framework goed aan de requirements voldoet. Er blijven van enkele zaken wensen over die in de volgende paragraaf zullen worden besproken.

8.1.1 Functionele requirements

Het framework biedt, met behulp van de AccountingSender, AccountingHandler en AAAUnit een eenvoudige interface voor accounting−applicaties. De AAAUnit wordt gebruikt om transportprotocollen en recordformats te koppelen, waarbij op een server een AccountingHandler en op een service−element een AccountingSender wordt gebruikt.

Het framework is toegespitst op het verzenden en ontvangen van accounting−gegevens. Er worden geen aannamen over sessies of accounting−records gedaan. Dit valt buiten het bestek van het framework. Binnen het framework wordt alleen onderscheid gemaakt tussen "accounting request"− en "accounting reply"−berichten en niet tussen "start"−, "stop"− en "interim"−berichten. Hiermee is het framework goed toepasbaar voor verschillende vormen van dienstverlening.

Het aanbieden en verwerken van accounting−gegevens is voor een groot deel onafhankelijk van het gebruikte transport−protocol. Verschillende record−formats en daarmee accounting−protocollen kunnen verschillende attributen opnemen in berichten. Hierbij komt wel dat er recordformats zijn die bepaalde attributen verplichten. In veel accounting−protocollen wordt een session−id gebruikt en is maar een beperkte verzameling attributen te gebruiken. Voor de implementatie van RADIUS is daar een workaround voor gemaakt. Ook kan het zijn dat een bepaald model van berichtentransport niet door het protocol wordt ondersteund. Deze beperkingen zijn beperkingen in de protocollen en niet zozeer in het geïmplementeerde systeem.

Het framework ondersteunt het simultaan gebruik van verschillende protocollen voor het transporteren van accounting−gegevens. Zo is het voor een accounting−server mogelijk om tegelijk berichten te verwerken die met verschillende accounting−protocollen zijn verstuurd. Van inkomende berichten wordt automatisch bepaald, met welk accounting−protocol ze zijn verstuurd. Bij uitgaande berichten kan worden aangegeven welk recordformat en welk transportprotocol moet worden gebruikt. Antwoorden op ingekomen berichten worden automatisch verstuurd met het recordformat en transportprotocol van het ingekomen bericht.

8.1.2 Toepasbaarheid

Omdat Java is gebruikt voor het maken van de implementatie, is het framework in principe platform−onafhankelijk. Op alle systemen waar een "Java virtual machine" kan draaien is het mogelijk om dit framework toe te passen. Java gebruikt relatief veel resources dus een directe vervanging op service−elements met dedicated hardware is niet waarschijnlijk. Java komt echter steeds vaker beschikbaar voor eenvoudige hardware en is zeker op service−elements die een complexe dienstverlening bieden en op accounting−servers goed toepasbaar.

Voor het verzorgen van accounting op zowel service−elements als accounting−servers zijn binnen het framework modulen beschikbaar. Door een kleine uitbreiding is het ook mogelijk om voor een accounting−proxy een dergelijk module te maken. Voor al deze omgevingen wordt de AAAUnit gebruikt voor het berichtentransport en een omgeving−specifieke module voor een specifieke toepassing.

Doordat eenvoudig protocollen aan het framework kunnen worden toegevoegd, is het mogelijk het framework in bestaande accounting−omgevingen in te passen. Er is bijvoorbeeld een implementatie gemaakt van RADIUS die in een RADIUS−omgeving zou kunnen worden gebruikt. In deze implementatie wordt gebruik gemaakt van leverancier−specifieke attributen voor het vastleggen van gegevens die niet in RADIUS zijn gedefinieerd. Dit zorgt ervoor dat deze attributen alleen leesbaar zijn voor systemen die op deze implementatie van RADIUS zijn gebaseerd.

In de AccountingSender zijn zowel het event−driven (met en zonder batching), als het polling en het event−driven polling model beschikbaar voor het transport van accounting−gegevens. Deze modellen kunnen alleen worden toegepast als de onderliggende protocollen deze berichten ook ondersteunen. Zo is bij gebruik van RADIUS alleen het event−driven model beschikbaar.

Voor het framework kunnen verschillende accounting−protocollen worden gedefinieerd. Voor een nieuw te definiëren transportprotocol of recordformat hoeft alleen de betreffende interface in Java te worden geïmplementeerd. Het framework legt verder geen eisen aan de vorm van de accounting−gegevens. In principe kunnen alle attributen, die door het recordformat worden ondersteund, worden verzonden.

Vrijwel alle aspecten van het framework zijn te beïnvloeden door een in het framework gegeven interface te implementeren of een subklasse te definiëren. De functionaliteit van het framework is duidelijk gespreid over de verschillende modulen. Een eventuele uitbreiding voor toepassing van autorisatie of authenticatie kan worden gerealiseerd door de implementatie van klassen vergelijkbaar met de AccountingHandler en AccountingSender.

8.1.3 Interface requirements

Het framework biedt een eenvoudige application−interface. Het framework wordt geconfigureerd door het aanmaken van een AAAUnit en daar TransportProtocollen, RecordFormats en MessageHandlers bij te registreren. Afhankelijk van of het om een server of een service−element gaat kan een AccountingSender of een subklasse van de AccountingHandler worden geregistreerd.

Hiermee is een flexibel systeem geïmplementeerd, waarbij, ook terwijl het systeem in gebruik is, transportprotocollen en recordformats kunnen worden toegevoegd en verwijderd. Ook is het mogelijk om dynamisch de instellingen van de AccountingSender en AccountingHandler aan te passen, zodat een ander model voor berichtenbezorging kan worden gebruikt.

De configuratie−mogelijkheden van het transportprotocol en het recordformat hangen af van de implementatie hiervan. Bij de huidige TCPtransport en UDPtransport kunnen alleen bij het initiëren van het protocol parameters worden opgegeven. Het is echter geen probleem om protocollen te implementeren die dynamischer zijn te configureren.

Het verzenden van accounting−gegevens gebeurt door het aanroepen van de account−methode in de AccountingSender. De opgegeven accounting−gegevens worden dan aan de hand van het ingestelde model en met het op te geven recordformat en transportprotocol verzonden. Het verwerken van ontvangen accounting−gegevens gebeurt door het definiëren van een methode in een subklasse van de AccountingHandler.

8.1.4 Schaalbaarheid en betrouwbaarheid

De schaalbaarheid en betrouwbaarheid van het gedefinieerde accounting−framework hangen sterk af van de implementatie van het gebruikte transportprotocol en recordformat. De schaalbaarheid wordt verbeterd door het gebruik van buffers en threads in het geïmplementeerde systeem. Doordat de afhandeling van ingekomen berichten is losgekoppeld van de decodering en het ontvangen van accounting−berichten, staat het framework steeds snel klaar voor het ontvangen van nieuwe berichten. Het huidige systeem biedt weinig overhead en is prima geschikt om grote hoeveelheden accounting−gegevens te verwerken.

Veel van de aspecten die de performance van het framework kunnen beïnvloeden zijn afzonderlijk te implementeren. Door het maken van een subklasse van de AccountingSender en AccountingHandler zijn deze zaken in meer detail te bepalen. Hierbij moet worden aangemerkt dat methoden voor het afhandelen van berichten zo geïmplementeerd dienen te worden dat ze snel retourneren, zodat de thread snel weer beschikbaar is voor het verwerken van nieuwe berichten. Het eventueel wachten op een antwoord−bericht moet dus in een aparte thread plaatsvinden.

In het framework worden uitgaande en ingekomen accounting−berichten gebufferd. Ontvangen en gedecodeerde berichten, berichten die nog verstuurd moeten worden en verstuurde berichten die wachten op een bevestiging worden in een buffer opgeslagen. Binnen het framework gebeurt dit in het werkgeheugen.

Er is in de AccountingSender een vorm van fail−over geïmplementeerd. Zogauw wordt gesignaleerd dat een server niet bereikbaar is, door een foutmelding van het transportprotocol of een time−out, wordt overgeschakeld naar een andere accounting−server. Door aan verschillende service−elements verschillende primaire accounting−servers op te geven kan een eenvoudige vorm van load−balancing worden gerealiseerd.

8.1.5 Security requirements

In het framework is het mogelijk om encryptie te regelen in het transportprotocol en/of in het recordformat. Het framework verzorgt zelf geen encryptie maar het recordformat en transportprotocol zijn flexibel genoeg om in deze zaken te voorzien. Hierin is het mogelijk om de encryptie zowel end−to−end als hop−by−hop toe te passen. In de voorbeeldprotocollen is geen encryptie geïmplementeerd.

8.2 Beperkingen

Tijdens het implementeren en testen is een aantal beperkingen in het framework gevonden. Deze zijn voor het grootste deel teruggekoppeld in het ontwerp van het framework en opgelost. Zo zijn de implementatie van de AccountingHandler en AccountingSender vrij laat nog aangepast om in het framework ruimte te bieden voor de polling en de event−driven polling aanpak. Er is echter nog een aantal beperkingen in het framework over. Er zijn ook nog enkele uitbreidingen aan het framework mogelijk waardoor de toepasbaarheid wordt verbeterd. Deze zaken worden hier beschreven.

Om het framework in meer omgevingen in te kunnen passen is het nodig om meer accounting−protocollen voor het framework te implementeren. Momenteel zijn naast de implementatie van RADIUS, alleen TCPTransport en UDPTransport als transportprotocol en JObjectFormat als recordformat gedefinieerd. Momenteel wordt bij geen enkel transportprotocol of recordformat een effectieve vorm van encryptie geïmplementeerd. Ook zijn de TCPTransport en UDPTransport gevoelig voor een "denial of service attack". Het verdient dus aanbeveling om andere protocollen te implementeren waarmee het framework in meer omgevingen kan worden gebruikt. Met name met de implementatie van nieuwe protocollen, zoals DIAMETER is het framework gebaat. Een beperking hierbij is dat het te implementeren protocol wel te splitsen moet zijn in een recordformat en een transportprotocol en dat op de berichtenlaag met een request−reply mechanisme gewerkt moet kunnen worden.

In de huidige implementatie van accounting−berichten worden de attributen opgeslagen in een verzameling. Dit betekent dat elk attribuut maar één keer voor mag komen en dat er aan de attributen geen volgorde wordt gegeven. Er zijn echter accounting−protocollen waarbij attributen vaker mogen voorkomen en waar de volgorde van attributen wel degelijk uitmaakt. Hierdoor is het voor een aantal protocollen niet eenvoudig een implementatie voor het framework te maken. Verder zijn momenteel geen voorzieningen getroffen om structuur aan attributen te geven door ze te groeperen. Het verdient aanbeveling deze beperkingen in het framework op te lossen door een uitbreiding op de definitie van een bericht in het framework.

In het huidige framework is het mogelijk om het recordformat en transportprotocol van een accounting−protocol te vervangen door andere protocollen. Het framework stelt weinig eisen aan de structuur van beide. In het framework wordt er echter wel van uitgegaan dat een te implementeren accounting−protocol in één van de gedefinieerde berichtenmodellen past. Zo is het in het framework momenteel alleen mogelijk om een accounting−protocol te integreren dat ook met "accounting request" en "accounting reply"−berichten werkt. In de AAAUnit is het echter geen probleem om ook andere soorten berichten te kunnen versturen. Als er accounting−protocollen zijn met andere berichtenmodellen zouden deze modellen in de AccountingSender en AccountingHandler geïmplementeerd moeten worden.

De Identifier−klasse, die een element in het netwerk identificeert, doet dit op basis van de naam van de host in het netwerk. Hiermee wordt ervan uitgegaan dat het service−element en de accounting−server zich op verschillende hosts bevinden. Veel accounting−protocollen, waaronder bijvoorbeeld RADIUS, gaan van dezelfde aanname uit. Dit hoeft echter niet noodzakelijkerwijs zo te zijn. In een webserver−omgeving is het bijvoorbeeld goed mogelijk dat de accounting−server zich op dezelfde machine bevindt als de webserver. Het blijkt dat deze aanname, afhankelijk van het transportprotocol, geen directe beperking hoeft te zijn als het transportprotocol op de client een andere poort voor de communicatie gebruikt dan op de server.

Het framework genereert op enkele plaatsen accounting−berichten waarbij de aanroepende applicatie geen volledige controle heeft over welke attributen precies in deze berichten aanwezig moeten zijn. De AccountingSender en AccountingHandler genereren bijvoorbeeld "accounting poll"− en "accounting indication"−berichten. Ook gebeuren er andere handelingen met berichten, zoals het controleren of een bericht een antwoord is op een ander bericht, waarbij meer invloed van de applicatie is gewenst. Het zou de toepasbaarheid van het framework ten goede komen als er een klasse beschikbaar zou komen, waarmee deze handelingen kunnen worden uitgevoerd. Deze klasse zou dan door de applicatie kunnen worden gedefinieerd of aangepast, zodat de applicatie meer grip krijgt op de uiteindelijke vorm van de berichten. Een dergelijke factory−constructie zou de implementatie van accounting−protocollen waarbij bijzondere attributen aanwezig moeten zijn kunnen vereenvoudigen.

Momenteel worden op verschillende plaatsen in het framework buffers van berichten bijgehouden. Het is wenselijk om deze buffers in een enkele klasse te implementeren, zodat er eenvoudig uitbreidingen aan de buffers kunnen worden gedaan. Momenteel gebeurt buffering bijvoorbeeld volledig in het werkgeheugen. In geval van uitval van het element zijn de gebufferde gegevens dan verloren. Het is mogelijk om een buffer te implementeren die de gebufferde gegevens naar disk wegschrijft, zodat deze gegevens niet verloren gaan bij een uitval.

Er is in de AccountingSender een vorm van fail−over geïmplementeerd. Zogauw wordt gesignaleerd dat een server niet bereikbaar is (door een foutmelding van het transportprotocol of door een time−out) wordt overgeschakeld naar een andere accounting−server. Als de AAAUnit, naast accounting, ook bijvoorbeeld voor authenticatie en autorisatie berichten uitwisselt, is het wenselijk dat deze allemaal hetzelfde fail−over mechanisme gebruiken. Hierdoor is het mogelijk om de functionaliteit van het sturen van berichten aan meerdere servers in een enkele klasse onder te brengen. Hierin zou ook een load−balancing mechanisme kunnen worden ondergebracht. Het gebruik van een enkele aparte klasse maakt het implementeren van een nieuw fail−over mechanisme ook eenvoudiger.

In de implementatie ontbreekt momenteel een module voor het gebruik van het framework in een accounting−proxy. Deze module, analoog aan de AccountingHandler en AccountingSender zou eenvoudig kunnen worden geïmplementeerd op basis van genoemde modulen. Hierbij kunnen de via de AccountingHandler ontvangen gegevens weer aan een AccountingSender worden aangeboden. Het is ook mogelijk om een proxy te implementeren die alleen de ingekomen berichten doorstuurt naar een andere server en de antwoorden daarop weer aan het service−element doorgeeft. De proxy zou dan op basis van een ontvanger eventueel een ander transportprotocol en recordformat kunnen gebruiken voor het doorsturen.

9. Conclusies en aanbevelingen

In dit hoofdstuk worden aan de hand van de resultaten van het onderzoek conclusies getrokken. Er worden suggesties aangedragen voor uitbreidingen en verbeteringen aan het framework en er zullen aanbevelingen worden gedaan voor toepassing en verder onderzoek.

Er is een framework ontworpen en geïmplementeerd, waarmee transport van accounting−gegevens kan worden geregeld, onafhankelijk van het gebruikte accounting−protocol. Hiermee is het framework toepasbaar in verschillende omgevingen, waar verschillende vormen van accounting nodig zijn en verschillende protocollen worden gebruikt. Het vormt een generiek model voor het uitvoeren van accounting−gerelateerde taken. Het framework is hierdoor bruikbaar om accounting−gegevens van verschillende bronnen in verschillende formaten centraal te verwerken. Het is mogelijk om accounting−gegevens in veel verschillende formaten te ontvangen en te versturen. Hierbij kan het framework bijvoorbeeld in een proxy worden toegepast om accounting−gegevens tussen verschillende protocollen te converteren.

Door het gebruik van verschillende protocollen is het framework in verschillende accounting−omgevingen toepasbaar. Als er relatief eenvoudige accounting−gegevens moeten worden getransporteerd, zou een RADIUS−achtig protocol kunnen worden gebruikt, terwijl bij complexere dienstverlening XML−geformatteerde gegevens over een SSL−verbinding kunnen worden getransporteerd. Met het gebruik van verschillende protocollen is het framework erg flexibel.

Het framework bestaat uit twee onderdelen. De AAAUnit regelt het versturen en ontvangen van berichten in verschillende formaten. Hiermee is de AAAUnit ook bruikbaar als basis voor andere berichtuitwisselende systemen, zoals bij authenticatie en autorisatie. Het tweede deel, dat het uitwisselen van accounting−berichten op een berichten−niveau regelt, is specifiek voor accounting ontwikkeld. Het definieert een model voor het transporteren van accounting−gegevens.

Deze twee delen kunnen in een toepassing worden gebruikt om het accounting−deel voor een dienstverlenend systeem te regelen. Het framework is toepasbaar aan zowel de client−kant als de server−kant. Aan de client−kant worden accounting−gegevens aangeboden en aan de server−kant komen de accounting−gegevens weer beschikbaar om te worden verwerkt. Hierbij is het gebruik van een accounting−protocol transparant voor de toepassing.

Gedurende de ontwikkeling van het framework zijn verscheidene grote en minder grote aanpassingen aan het model en het ontwerp teruggekoppeld. Er blijven echter natuurlijk altijd punten waar het huidige framework kan worden uitgebreid of waar verder onderzoek nodig is.

Het is mogelijk het huidige framework uit te breiden, zodat het ook voor authenticatie en autorisatie bruikbaar is. Deze uitbreiding is in principe relatief eenvoudig te realiseren, omdat bij deze vormen van berichtuitwisseling eigenlijk alleen een event−driven aanpak nodig is. Hiertoe zou een aantal modulen, vergelijkbaar met de AccountingSender en AccountingHandler moeten worden gemaakt. Hiervoor is echter nog wel extra onderzoek nodig om na te gaan welke requirements hierbij een rol spelen.

Als het framework ook voor authenticatie en autorisatie gaat worden gebruikt, is het waarschijnlijk handig om bepaalde functionaliteit, die nu in de AccountingSender en AccountingHandler is gelokaliseerd, in een aparte module onder te brengen. Voorbeelden van dergelijk functionaliteit zijn fail−over mechanismes zoals deze nu zijn gedefinieerd in de AccountingSender en polling−functionaliteit zoals die nu in de AccountingHandler is gedefinieerd. Ook zou een generieke manier van het bufferen van berichten kunnen worden gedefinieerd. Met een scheiding in aparte modulen is het ook denkbaar om zaken als fail−over en load−balancing centraal te regelen.

Momenteel heeft de applicatie die het framework gebruikt geen directe invloed op de vorm van bepaalde berichten. De AccountingSender en AccountingHandler bijvoorbeeld genereren "accounting poll"− en "accounting indication"−berichten, zonder dat een applicatie kan aangeven welke attributen in deze berichten aanwezig moeten zijn. Met een aanpassing in het framework is het mogelijk Message−objecten te genereren met behulp van een factory−constructie. Deze factory−klasse zou door de applicatie kunnen worden gedefinieerd en het daadwerkelijk genereren van Message−objecten op basis van een aantal parameters kunnen regelen. Met een dergelijke constructie kan de applicatie meer grip krijgen op de uiteindelijke vorm van berichten.

Om het framework goed toepasbaar te maken is het nodig om meerdere accounting−protocollen te implementeren. Met name met de implementatie van nieuwe protocollen, zoals DIAMETER is het framework gebaat. Hierbij zouden protocollen hiërarchisch kunnen worden geïmplementeerd, waarbij het protocol wordt gedefinieerd op basis van een ander, eerder geïmplementeerd protocol. Zo is in de huidige implementatie RADIUS gedefinieerd op basis van het UDP transportprotocol. Ook voor recordformats is een dergelijke structuur denkbaar waarbij MSIX bijvoorbeeld een afgeleide is van een algemeen XML−gebaseerd recordformat.

Met het voor dit afstudeerwerk ontworpen en geïmplementeerde systeem is een framework opgezet waarin meerdere accounting−protocollen kunnen worden geïntegreerd. Met de binnen dit framework gedefinieerde modulen is het mogelijk accounting−gegevens op een generieke manier te transporteren. Voor een uiteindelijke toepassing van het systeem hoeft het framework alleen nog maar te worden aangevuld met de juiste modulen voor de verschillende protocollen en toepassingen.

Literatuurlijst

[aaapage] A. de Jong, "Authentication Authorisation Accounting − AAA", http://ch.twi.tudelft.nl/~arthur/aaa/

[acc−prot] A. de Jong, "Accounting en Accounting protocollen", onderzoeksverslag, februari 2000.

[billaudit] Infozech: Telephone Bill Audit and Overcharge Recovery Service, http://www.infozech.com/audit.html

[bacct−msix] A. Blount, D. Young, "Metered Service Information eXchange Protocol Specification version 1.2", Internet draft (work in progress*), draft−blount−acct−msix−00.txt, July 1999.

[diameter] P.R. Calhoun, A.C. Rubens, H. Akhtar, E. Guttman, "DIAMETER Base Protocol", Internet draft (work in progress*), draft−calhoun−diameter−17.txt, September 2000.

[diameter−accounting] J. Arkko, P.R. Calhoun, P. Patel, G. Zorn, "DIAMETER Accounting Extension", Internet draft (work in progress*), draft−calhoun−diameter−accounting−08.txt, September 2000.

[diameter−framework] P. R. Calhoun, G. Zorn, P. Pan, H. Akhtar, "DIAMETER Framework Document", Internet draft (work in progress*), draft−calhoun−diameter−framework−08.txt, June 2000.

[etsiwww] ETSI (European Telecommunications Standards Institute) Home Page, http://www.etsi.org/

[grant−tacacs] D. Carrel, L. Grant, "The TACACS+ Protocol Version 1.78", Internet draft (work in progress*), draft−grant−tacacs−02.txt, January 1997.

[ietf−aaa−acct] B. Aboba, J. Arkko, D. Harrington, "Introduction to Accounting Management", Internet draft (work in progress*), draft−ietf−acct−06.txt, June 2000.

[ietf−rap−cops] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, A. Sastry, "The COPS (Common Open Policy Service) Protocol", Internet draft (work in progress*), draft−ietf−rap−cops−08.txt, November 1999.

[roamops−actng] B. Aboba, D. Lidyard, "The Accounting Data Interchange Format (ADIF)", Internet draft (work in progress*), draft−ietf−roamops−actng−07.txt, April 2000.

[ofxwww] OFX: Home Page, http://www.ofx.net/

[mobiptut] C.E. Perkins, "Tutorial: Mobile Networking Through Mobile IP", http://www.computer.org/internet/v2n1/perkins.htm

[msixwww] MSIX (Metered Service Information eXchange) homepage, http://www.msix.org/

[rfc1521] N. Borenstein, N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, December 1993.

[rfc1272] C. Mills, D. Hirsh, G. Ruth, "Internet Accounting: Background", RFC 1272, November 1991.

[rfc2194] B. Aboba, J. Lu, J. Alsop, J. Ding, W. Wang, "Review of Roaming Implementations", RFC 2194, September 1997.

[rfc2205] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) −− Version 1 Functional Specification", RFC 2205, September 1997.

[rfc2486] B. Aboba, M. Beadles, "The Network Access Identifier", RFC 2486, January 1999.

[rfc2865] C. Rigney, S. Willens, A. Rubens, W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[rfc2866] C. Rigney, "RADIUS Accounting", RFC 2866, June 2000.

* Internet−Drafts zijn werkddocumenten van de Internet Engineering Task Force (IETF), haar werkgebieden en haar werkgroepen. Internet−Drafts zijn tijdelijke documenten met een maximale geldigheidsduur van zes maanden die op elk moment kunnen worden vervangen, aangepast of verwijderd. Internet−Drafts horen niet gebruikt te worden als permanent referentiemateriaal en zouden altijd moeten worden aangeduid als "work in progress". Omdat accounting bij dit schrijven nog een dynamisch vakgebied is, wordt hier toch gerefereerd aan Internet−Drafts, omdat er nog geen betere bron voor handen is. Een lijst met Internet−Drafts is te vinden op http://www.ietf.org/ietf/1id−abstracts.txt .

[rfc2960] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

[rsvpacc] Accounting model based on integrated RSVP/intserv and diffserv architecture, http://ing.ctit.utwente.nl/WU5/ongoing/qosmodel/qos−model.html

[rsvpwww] RSVP Project, http://www.isi.edu/div7/rsvp/

[ts101321] "Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON); Inter−domain pricing, authorization, and usage exchange", TS 101 321 V1.4.2, December 1998.

[snmpintro] "An Introductory Overview of SNMP", Diversified Data Resources Inc., http://www.ddri.com/, 1999.

[xmlw3c] T. Bray, J. Paoli, C. Sperberg−McQueen, "Extensible Markup Language (XML) 1.0", W3C Recommendation, February 1998.

Index

S

T

U

V

X

Appendix A. Protocollen

In deze appendix zal worden ingegaan op een aantal meest voorkomende accounting−protocollen. Ook zullen enkele accounting−protocollen in ontwikkeling worden behandeld. Veel van de hier behandelde protocollen zijn AAA−protocollen, maar bij het behandelen zal de nadruk worden gelegd op het accounting−gedeelte. Ook zullen enkele protocollen worden behandeld die (tot op heden) geen accounting−functionaliteit kennen, maar wel aan accounting zijn gerelateerd.

Van de protocollen zullen het transportprotocol, recordformat en mogelijke berichten en attributen behandeld worden. Ook zal worden aangegeven in welke mate het protocol toepasbaar is en welke beperkingen het kent.

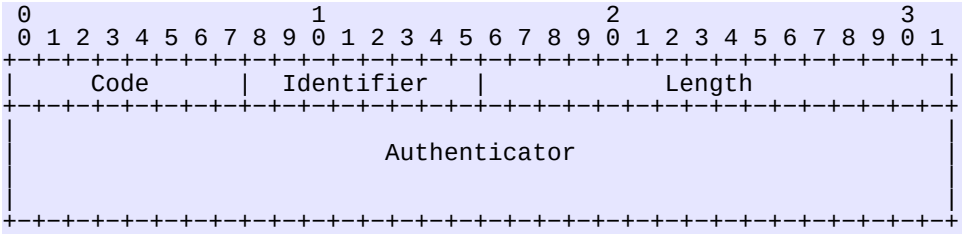
A.1. RADIUS

RADIUS (Remote Authentication Dial In User Service) is een protocol dat in [rfc2865] wordt beschreven. Dit protocol wordt gebruikt om authenticatie−, autorisatie− en configuratiegegevens te transporteren tussen een Network Access Server (NAS) en een RADIUS−server. RADIUS is ontwikkeld door Livingston Enterprises voor het besturen van hun PortMaster Access Servers en wordt tegenwoordig in veel inbelproducten ondersteund. Er is veel software voor beschikbaar. Aan RADIUS is een accounting−extensie toegevoegd [rfc2866].

De NAS (het service−element) vraagt aan de RADIUS−server of het een dienst aan een bepaalde gebruiker mag aanbieden. De RADIUS−server heeft een centrale database met gebruikersgegevens, wachtwoorden en configuratiegegevens en beantwoordt verzoeken van de NAS. De NAS zorgt voor het aanleveren van inloggegevens aan de server en volgt de bevelen van de server op. De server zorgt voor het authenticeren van de gebruiker (aan de hand van gegevens van de NAS) en het leveren van configuratiegegevens voor de sessie.

RADIUS gebruikt UDP als transportprotocol. Een RADIUS−bericht wordt in één UDP−pakket verstuurd, waardoor niet al te grote berichten kunnen worden verstuurd. Voor complexe berichten met erg veel attributen is RADIUS dus niet geschikt. Omdat er met UDP geen verbinding wordt gemaakt, definieert RADIUS zelf een retry−mechanisme, waarbij eventueel kan worden teruggevallen op een tweede server. De invulling van het retry−mechanisme wordt echter aan de implementatie overgelaten. Dit zorgt voor kleine incompatibiliteiten tussen verschillende implementaties.

RADIUS gebruikt een eenvoudig binair formaat om zijn gegevens te transporteren. Dat zorgt ervoor dat het eenvoudig is te genereren door simpele hardware. RADIUS−berichten bestaan uit een header gevolgd door Attribute−Length−Value 3−tuples. De RADIUS−header wordt beschreven in *Figuur 10.1*. RADIUS−transacties worden geauthenticeerd met behulp van een shared secret. User passwords worden versluierd tussen de NAS en de RADIUS−server verstuurd. Overige gegevens worden ongecodeerd verzonden en zijn dus gewoon leesbaar.



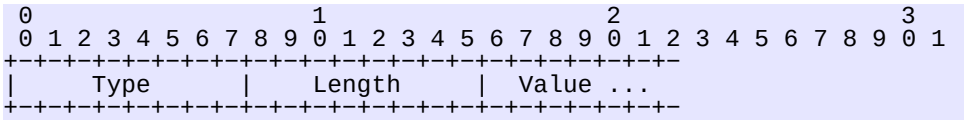
Figuur 10.1: RADIUS−header

Het Code−veld in de RADIUS−header geeft het type bericht aan. De mogelijke soorten berichten staan vermeld in *Tabel 10.1*. Accounting−berichten worden verpakt in een Accounting−Request. Deze worden in het algemeen door de RADIUS−server beantwoord met een Accounting−Response. Het Identifier−veld in de RADIUS−header wordt gebruikt om requests en replies bij elkaar te zoeken. Het Length−veld geeft de totale lengte van het RADIUS−bericht aan (header en attributen). Het Authenticator−veld wordt gebruikt om antwoorden van de RADIUS−server te authenticeren en om wachtwoorden te versluieren.

Code	*type of packet*
1	Access−Request
2	Access−Accept
3	Access−Reject
4	Accounting−Request
5	Accounting−Response
11	Access−Challenge
12	Status−Server (experimental)
13	Status−Client (experimental)
255	Reserved

Tabel 10.1: RADIUS Codes. De soorten berichten die in RADIUS zijn gedefinieerd.

RADIUS−attributen bevatten informatie die nodig is voor de authenticatie, autorisatie of accounting. Alle attributen hebben een naam, een lengte en een waarde. De Attributen hebben het volgende formaat:



Figuur 10.2: RADIUS attribute format.

Het Type−veld geeft de naam van het attribuut (en daarmee het type van het Value−veld). Het Length−veld geeft de lengte van dit attribuut (Type, Length en Value velden) aan. Het Value−veld geeft de waarde van het attribuut aan. Het type van de waarde wordt bepaald aan de hand van de Type− en Length−velden. In RADIUS zijn de datatypen string (ascii), text (utf8), address (32−bit), integer (32−bit unsigned) en time (32−bit) gedefinieerd. In *Tabel 10.2* staan de belangrijkste attributen die in RADIUS zijn gedefinieerd.

Code	*Attribute name*	*Type*	*In accounting*
2	User−Password	string	0
3	CHAP−Password	CHAP Ident+string	0
4	NAS−IP−Address	address	0−1
5	NAS−Port	integer	0−1
6	Service−Type	integer	0−1
7	Framed−Protocol	integer	0−1
8	Framed−IP−Address	address	0−1
9	Framed−IP−Netmask	address	0−1
10	Framed−Routing	integer	0−1
11	Filter−Id	text	0
12	Framed−MTU	integer	0−1
13	Framed−Compression	integer	0
14	Login−IP−Host	address	0
15	Login−Service	integer	0−1
16	Login−TCP−Port	integer	0−1
18	Reply−Message	text	0
19	Callback−Number	string	0−1
20	Callback−Id	string	0−1
26	Vendor−Specific	vendor−id+string	0
27	Session−Timeout	integer	0−1

Code	*Attribute name*	*Type*	*In accounting*
28	Idle−Timeout	integer	0−1
29	Termination−Action	integer	0−1
30	Called−Station−Id	string	0−1
31	Calling−Station−Id	string	0−1
32	NAS−Identifier	string	0−1
40	Acct−Status−Type	integer	1
41	Acct−Delay−Time	integer	0−1
42	Acct−Input−Octets	integer	0−1
43	Acct−Output−Octets	integer	0−1
44	Acct−Session−Id	text	1
45	Acct−Authentic	integer	0−1
46	Acct−Session−Time	integer	0−1
47	Acct−Input−Packets	integer	0−1
48	Acct−Output−Packets	integer	0−1
49	Acct−Terminate−Cause	integer	0−1
50	Acct−Multi−Session−Id	string	0
51	Acct−Link−Count	integer	0
60	CHAP−Challenge	string	0
85	Acct−Interim−Interval	integer	0−1

Tabel 10.2: RADIUS attributen. De kolom 'In accounting' geeft aan hoe vaak het attribuut in een Accounting−Request mag voorkomen:

0 Het attribuut mag niet worden gebruikt in een Accounting−Request
0−1 Het attribuut mag nul of een keer voorkomen
1 Het attribuut moet precies een keer voorkomen

Daarnaast zijn er diverse andere attributen voor bijvoorbeeld tunneling of leverancier−specifieke attributen gedefinieerd. Met het Acct−Status−Type attribuut wordt het soort accounting−bericht aangegeven. In *Tabel 10.3* staan de mogelijke soorten accounting−berichten die met RADIUS kunnen worden verstuurd.

Value	*type of message*
1	Start
2	Stop
3	Interim Update
7	Accounting On
8	Accounting Off

Tabel 10.3: RADIUS Acct−Status−Type attribuutwaarden.

Als de NAS begint met het leveren van een dienst wordt een Start−bericht aan de RADIUS accounting−server verstuurd met daarin de gegevens over de te leveren dienst en de gebruiker van de dienst. Aan het einde van het bieden van de dienst verstuurt de NAS een Stop−pakket met statistieken over de gebruikte dienst. Tijdens de dienstverlening kunnen Interim−Updates worden verstuurd.

RADIUS is ontworpen als AAA−protocol voor het regelen van inbelverbindingen. Hiervoor wordt RADIUS vaak toegepast en is het in feite de standaard. RADIUS mist echter een aantal eigenschappen om geschikt te zijn als algemeen accounting−protocol voor meerdere toepassingen. Bij RADIUS wordt verondersteld dat alle accounting−gegevens real−time worden verstuurd. Batching en polling behoren dus niet tot de mogelijkheden. Ook worden de berichten in RADIUS ongecodeerd verstuurd waardoor ze door iedereen leesbaar zijn wat vooral problemen oplevert bij inter−domain accounting. RADIUS is ook niet voldoende uitbreidbaar om voor algemene accounting−toepassingen te kunnen worden gebruikt. De berichten− en attributenverzameling is

beperkt en de mogelijkheid om leverancier−specifieke attributen te gebruiken is onvoldoende gestandaardiseerd. Hierdoor is RADIUS eigenlijk alleen te gebruiken voor inbellen en soortgelijke diensten. Het transport−gedeelte van RADIUS kent geen bidirectionele communicatie en flow−control, waarmee voorkomen kan worden dat netwerken overbelast raken.

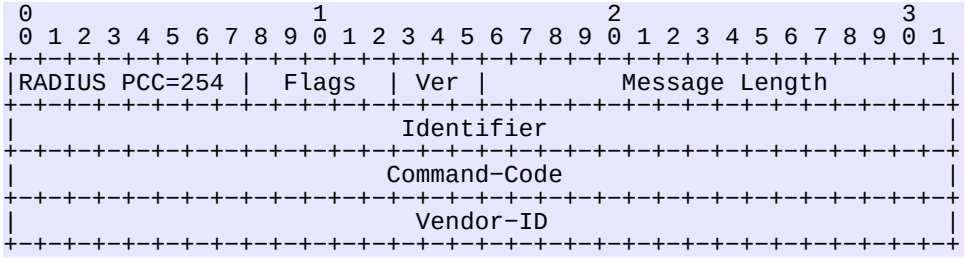
A.2. DIAMETER

Diameter is ontworpen om RADIUS te vervangen. DIAMETER is op het moment van dit schrijven nog volop in ontwikkeling en deze paragraaf geeft dus waarschijnlijk geen actueel beeld. De ontwikkeling vindt momenteel plaats in de AAA−werkgroep van de IETF. Het DIAMETER framework is gedefinieerd in een Internet−draft [diameter−framework] en geeft een beschrijving van het ontwerp van DIAMETER. Hierin worden de problemen en tekortkomingen van RADIUS opgesomd en wordt aangegeven hoe deze met DIAMETER worden opgelost. Het gebruikte recordformat, het transportprotocol en de mogelijke berichten zijn gedefinieerd in [diameter]. Een toepassing van DIAMETER wordt in een apart document als extensie van DIAMETER gedefinieerd. Accounting met DIAMETER is gedefinieerd in [diameter−accounting]. Verder zijn er tal van documenten die extensies van DIAMETER beschrijven.

DIAMETER is ontworpen om goed te functioneren in een omgeving met proxies. Het is dan ook goed toepasbaar in een inter−domain omgeving. DIAMETER biedt hop−by−hop beveiliging, maar het is ook mogelijk om door middel van een extensie van DIAMETER end−to−end beveiliging op attributen toe te passen.

Transport van DIAMETER−berichten verloopt via SCTP [rfc2960] dat een betrouwbaar transport met snelle retransmissie en configureerbare timeout−parameters biedt. Met DIAMETER wordt een fail−over mechanisme gedefinieerd waarmee kan worden teruggevallen op een tweede server indien de eerste niet bereikbaar is. Met SCTP is ook flow−control van inkomende berichten geregeld, waarmee inkomende berichten over meerdere servers kunnen worden verdeeld.

DIAMETER berichten bestaan uit een header beschreven in *Figuur 10.3*. Het RADIUS Packet Compatibility Code (PCC) is opgenomen om DIAMETER pakketten van RADIUS pakketten te kunnen onderscheiden. Zo kunnen RADIUS en DIAMETER door elkaar worden gebruikt. Met Flags worden eigenschappen van het transport aangegeven, Message Length geeft de lengte van het bericht aan en Identifier is bedoeld om requests by replies te vinden. Met de Command−Code wordt het soort bericht aangegeven. In *Tabel 10.4* staan de berichttypen die voor accounting relevant zijn. Als er een leverancier−specifieke Command−Code wordt gebruikt staat in Vendor−ID een verwijzing naar de betreffende leverancier.

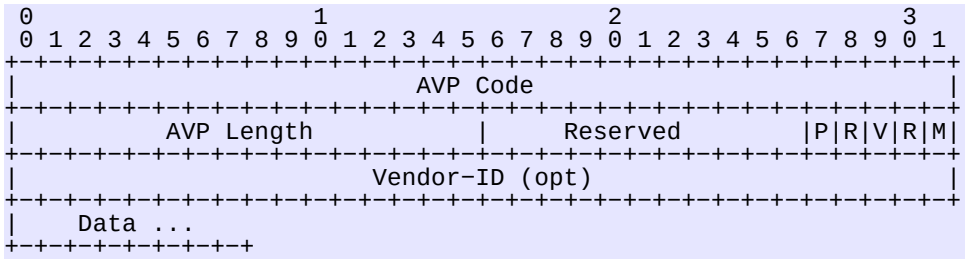


Figuur 10.3: DIAMETER header.

Value	***Name***	
271	Accounting−Request (ACR)	versturen accounting−gegevens
272	Accounting−Answer (ACA)	antwoord op Accounting−Request
273	Accounting−Poll−Ind (ACP)	polling−request door server
279	Accounting−Status−Ind (ASI)	client kan zijn status aan server doorgeven (up/down)

Tabel 10.4: DIAMETER accounting−berichten.

Na deze header volgen AVP's zoals beschreven in *Figuur 10.4*. Deze bevatten attributen die afhankelijk zijn van de toepassing. De AVP Code geeft het attribuut aan. Dit kan een leverancier−specifiek attribuut zijn, al naar gelang dat in de Flags is aangegeven. De eerste 256 attributen zijn gereserveerd voor RADIUS compatibiliteit en moeten dus worden geïnterpreteerd zoals ze in de RADIUS documenten zijn gedefinieerd.



Figuur 10.4: DIAMETER AVP format.

De AVP Length geeft de lengte van de AVP aan en met de Flags worden zaken aangegeven over het attribuut, zoals of het om een leverancier−specifiek attribuut gaat. Met het Vendor−ID wordt een eventuele leverancier aangeduid, die het attribuut heeft gedefinieerd. Hierna volgt de data. In *Tabel 10.5* staan de mogelijke datatypen opgesomd. In *Tabel 10.6* wordt een aantal attributen die in DIAMETER−berichten kunnen voorkomen opgesomd.

Type	***Omschrijving***
Data	binaire data in een willekeurig formaat
String	NULL−terminated UTF−8 string
Address	32−bit (IPv4) of 128−bit (IPv6) adres, afhankelijk van het AVP Length veld
Integer32	32−bit waarde
Integer64	64−bit waarde
Time	32−bit unsigned geeft het aantal verstreken seconden aan sinds 00:00:00 GMT, 1 Januari 1900
Complex	complexe datatypen

Tabel 10.5: DIAMETER datatypen.

Value	*Name*	
1	User−Name	de gebruiker van de dienst
6	Service−Type	de soort dienstverlening
257	Host−IP−Address	het adres van de gebruiker
259	Integrity−Check−Value	hash van bericht voor integrity protection
260	Encrypted−Payload	gebruikt om attributen in te pakken
262	Timestamp	tijd van verzenden op replay−protection te bieden
263	Session−Id	identificatie van sessie
280	Grouped−AVP	hiermee kunnen AVPs worden gegroepeerd
480	Accounting−Record−Type	het soort accounting−bericht
481	ADIF−Record	accounting−gegevens
482	Accounting−Interim−Interval	hoe vaak interim−berichten moeten worden verstuurd
483	Accounting−Delivery−Max−Batch	maximaal gebufferde accounting−berichten
484	Accounting−Delivery−Max−Delay	maximale tijd dat gebufferde accounting−berichten kunnen wachten
485	Accounting−Record−Number	id binnen het session−id
486	Accounting−State	enabled/disabled

Tabel 10.6: DIAMETER AVP's.

Accounting bij DIAMETER wordt uitgevoerd door de accounting−gegevens in een ADIF−record vast te leggen. Met het Accounting−Record−Type attribuut wordt dan het soort accounting−bericht aangegeven. De soorten berichten staan gegeven in *Tabel 10.7*.

Value	*Name*	
1	EVENT_RECORD	ondeelbare event
2	START_RECORD	start van dienstverlening
3	INTERIM_RECORD	tijdens de dienstverlening
4	STOP_RECORD	einde van de dienstverlening

Tabel 10.7: DIAMETER verschillende berichten.

De DIAMETER−server kan aan de client (het service−element) aangeven hoe vaak interim berichten moeten worden verstuurd en hoeveel accounting−gegevens moeten worden gebufferd. Tijdens de autorisatie van de dienstverlening kan met het Accounting−Interim−Interval attribuut de frequentie van de interimberichten, met Delivery−Max−Batch de maximum hoeveelheid van de gebufferde accounting−gegevens en met Delivery−Max−Delay de maximale wachttijd voor accounting−berichten worden aangegeven.

Omdat DIAMETER is ontworpen op basis van de problemen met RADIUS, zijn de security problemen die in RADIUS aanwezig zijn verholpen. Met uitbreidingen op DIAMETER kan beveiliging end−to−end geschieden of er kan voor een hop−by−hop aanpak worden gekozen. Ook zijn de tekortkomingen van het transportgedeelte opgelost door in plaats van UDP SCTP te gebruiken.

Het blijft een protocol met hetzelfde uitgangspunt als RADIUS: een AAA−protocol voor inbellen. Het is dus ook toegespitst op inbellen, hoewel het wel meer faciliteiten voor extensies en leverancier−specifieke attributen biedt. De mogelijkheid om ADIF records in DIAMETER in te bedden is een methode die accounting van meerdere soorten diensten mogelijk maakt.

DIAMETER, in combinatie met ADIF, is een goede kandidaat om RADIUS te vervangen. DIAMETER ondersteunt echter geen event−driven polling, accounting−gegevens en berichten zijn deels in een binair formaat, deels in ADIF−formaat opgeslagen en voor de definitie van diensten moet er een aanpassing aan de standaard worden geschreven.

DIAMETER is nog vol in ontwikkeling en wordt nog niet toegepast in systemen. Vooral het accounting−gedeelte is nog niet zo oud. Dit alles zorgt ervoor dat er momenteel nog geen eenduidige implementatie kan worden gemaakt.

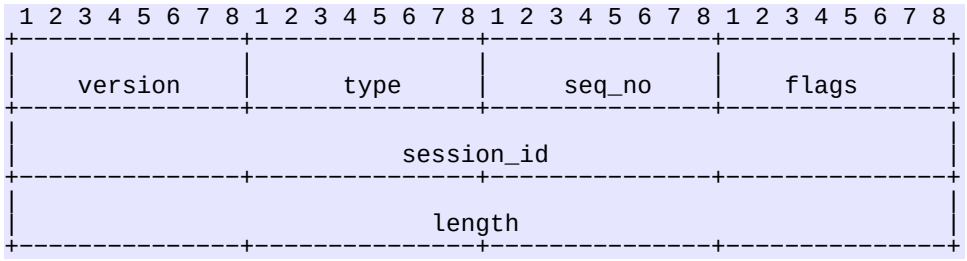
A.3. TACACS+

TACACS+ is de meest recente opvolger van TACACS. TACACS staat voor Terminal Access Controller Access Control System. TACACS+ versie 1.78 is gedefinieerd in [grant−tacacs].

TACACS is een eenvoudig UDP−gebaseerd protocol om access control te bieden. TACACS is oorspronkelijk ontwikkeld door BBN voor het MILNET. Cisco heeft een aantal verbeteringen aan TACACS aangebracht. De implementatie van Cisco wordt ook wel XTACACS genoemd.

In TACACS+ worden de authenticatie−, autorisatie− en accounting−functies gescheiden. Tevens wordt al het verkeer tussen de NAS (service−element) en de AAA−server gecodeerd verzonden. Het ondersteunt ook grotere berichten en meerdere vormen van authenticatie. Het is mogelijk om eigen attributen te gebruiken en uitbreiding in de toekomst te ondersteunen. Het gebruikt TCP in plaats van UDP als onderliggend transportprotocol om de betrouwbaarheid te verhogen. TACACS+ ondersteunt zowel de klassieke user/password authenticatie als one−time passwords en challenge−response authenticatie en kan bij de autorisatie specifieke configuratiegegevens doorgeven. Een voorbeeld van beperkte autorisatie is het instellen van een tijdbeperking op de verbinding.

TACACS+ heeft een binair recordformat. Alle TACACS+ berichten worden voorafgegaan door de packet−header die is beschreven in *Figuur 10.5*. De header zelf wordt niet gecodeerd verzonden en beschrijft de rest van het bericht. Met het version−veld wordt de gebruikte versie van TACACS+ aangegeven. Met het type−veld wordt aangegeven of het om een authenticatie−, autorisatie− of accounting−bericht gaat. TACACS+ sessies zijn enkele gevallen van het uitvoeren van authenticatie, een autorisatie request of het uitwisselen van een accounting−bericht. Met het seq_no−veld worden berichten die tot dezelfde TACACS+ sessie behoren oplopend genummerd. Met de flags wordt aangegeven of encryptie en multiplexing wordt toegepast. Het session_id geeft een uniek nummer aan de TACACS+ sessie. Het length−veld geeft de lengte van het TACACS+ bericht aan.

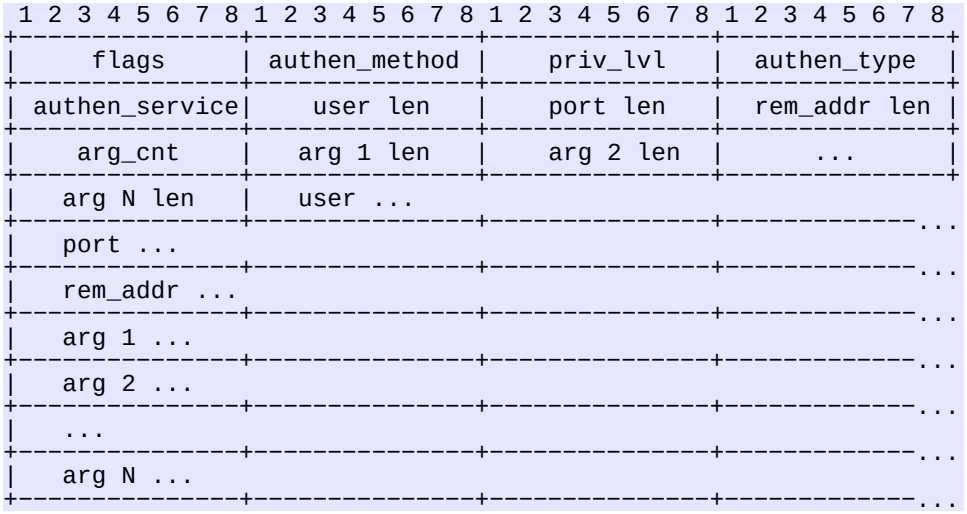


Figuur 10.5: TACACS+ packet−header.

De inhoud van een TACACS+ bericht wordt beveiligd door MD5−hashes over de header en een sleutel te gebruiken als een pseudo−random string. De gemeenschappelijke sleutel (secret key) is alleen bij zender en ontvanger bekend.

In TACACS+ bestaan accounting−berichten uit accounting−requests en accounting−replies. De body van het accounting−request formaat staat beschreven in *Figuur 10.6*, die van de reply in *Figuur 10.7*. Met flags wordt het soort bericht aangegeven (start, stop of watchdog), het authen_method−veld geeft de gebruikte authenticatie−methode aan en het priv_lvl−veld geeft de huidige gebruikersprivileges aan. Het authen_service−veld geeft de vorm van dienstverlening aan (zie *Tabel 10.8*). De len−velden geven de lengte van de volgende

attributen aan. Het user−veld geeft de gebruikersnaam aan en het port−veld de naam van de poort waarop de verbinding plaatsvindt. Daarna volgen de attributen. Deze zijn afkomstig uit het autorisatie−gedeelte, aangevuld met enkele specifieke attributen. Een aantal van de mogelijke attributen staan vermeld in *Tabel 10.9*. Een TACACS+ accounting reply bevat een status (success, error of follow), een bericht van de server voor de gebruiker en een bericht voor een eventuele systeembeheerder.



Figuur 10.6: TACACS+ accounting−request packet body.

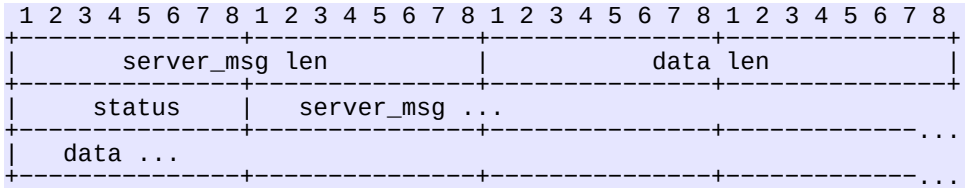
Value	***Name***	
0x00	TAC_PLUS_AUTHEN_SVC_NONE	onbekende dienst
0x01	TAC_PLUS_AUTHEN_SVC_LOGIN	
0x02	TAC_PLUS_AUTHEN_SVC_ENABLE	
0x03	TAC_PLUS_AUTHEN_SVC_PPP	
0x04	TAC_PLUS_AUTHEN_SVC_ARAP	
0x05	TAC_PLUS_AUTHEN_SVC_PT	
0x06	TAC_PLUS_AUTHEN_SVC_RCMD	
0x07	TAC_PLUS_AUTHEN_SVC_X25	
0x08	TAC_PLUS_AUTHEN_SVC_NASI	
0x09	TAC_PLUS_AUTHEN_SVC_FWPROXY	

Tabel 10.8: TACACS+ verschillende diensten.

Attribute name	***Omschrijving***
service	soort dienst
protocol	het gebruikte protocol in de dienstverlening
cmd	het uitgevoerde commando
cmd−arg	argumenten bij het commando
addr	netwerk adres
timeout	gebruikte timeout
idletime	gebruikte idletime
autocmd	commando dat in een eventuele shell wordt uitgevoerd
remote_user	

Attribute name	*Omschrijving*
remote_host	
task_id	de id die bij de sessie hoort
start_time	begin van de sessie
stop_time	de tijd dat de sessie eindigde
elapsed_time	de verlopen tijd in seconden
bytes	het aantal bytes dat bij de sessie is getransporteerd
bytes_in	
bytes_out	
paks	het aantal packets dat bij de sessie is getransporteerd
paks_in	
paks_out	
status	geeft status aan
err_msg	foutmelding

Tabel 10.9: aantal TACACS+ attributen.



Figuur 10.7: TACACS+ accounting−reply packet body.

TACACS+ is wat opzet en functionaliteit te vergelijken met RADIUS en wordt dan ook vaak als alternatief voor RADIUS gebruikt. Verschillen zijn het gebruik van TCP in plaats van UDP en het bieden van confidentiality. Ook TACACS+ is geen algemeen accounting−protocol en is in het ontwerp toegespitst op inbellen met alleen hop−by−hop security.

Verder zijn veel van de beperkingen van RADIUS ook van toepassing op TACACS+. In TACACS+ wordt security hop−by−hop toegepast, zodat het niet goed toepasbaar is bij untrusted proxies.

A.4. COPS

Het COPS (Common Open Policy Service) protocol is een eenvoudig vraag−en−antwoord−protocol in een client/server omgeving, dat wordt gebruikt om informatie over policies uit te wisselen tussen een policy server en haar clients. COPS is ontworpen om autorisatie te regelen bij RSVP resource requests in netwerken die Intserv ondersteunen. Het protocol is ontworpen om toegang tot algemene resources (diensten) te kunnen regelen, maar heeft op het moment geen accounting deel gedefinieerd. COPS is beschreven in [ietf−rap−cops].

Een Policy Decision Point (PDP) fungeert als policy server. Een Policy Enforcement Point (PEP) bevindt zich in het service−element en doet autorisatie=requests bij een PDP. De PEP voert dan de beslissingen van de PDP uit. Hierbij kan de PEP een statusbericht aan de PDP sturen, met daarin informatie over de uitgevoerde besluiten. Deze statusgegevens zouden voor accounting−toepassingen kunnen worden gebruikt.

Het protocol gebruikt TCP als transportprotocol en voorziet in beveiliging op berichtenniveau met authenticatie, replay−protection en integrity−protection.

A.5. SNMP

SNMP (Simple Network Management Protocol) is een wijdverbreid communicatieprotocol dat wordt gebruikt voor het beheer van allerlei, voornamelijk TCP/IP−gebaseerde, applicaties [snmpintro]. SNMP is gedefinieerd in een groot aantal RFC’s. SNMP wordt uitgebreid met MIB’s (Management Information Base), waar definities van objecten en hun betekenis in staan. Er zijn zeer veel van deze MIB’s voorhanden voor allerlei verschillende apparatuur.

SNMP kan over meerdere transportprotocollen worden getransporteerd, maar wordt in het algemeen over UDP getransporteerd. Het is niet ontworpen als accounting−protocol, maar kan daarvoor, binnen bepaalde grenzen, wel worden gebruikt. SNMP kent vier soorten berichten:

- een get−request,
- een get−next−request,
- een set−request en
- een trap−message.

Met een get−request kan een waarde uit een SNMP−object worden opgevraagd. Met behulp van een get−next−request kunnen alle meetbare waarden in een SNMP−object worden nagelopen. Het set−request wordt gebruik om instellingen te veranderen. Een trap−message kan door een object worden gegenereerd als een bepaalde vooraf gedefinieerde toestand optreedt. Accounting−informatie zou met polling via get−requests of een trap−mechanisme kunnen worden verstuurd. Het trap−mechanisme is echter eerder ontworpen voor foutmeldingen dan voor het transporteren van data.

SNMP gebruikt ASN.1 als formaat om gegevens te representeren. ASN.1 is een gestandaardiseerd binair formaat om berichten mee vast te leggen. Al met al is SNMP een niet erg simpel netwerk management protocol en geen accounting−protocol. Het kan in enkele gevallen wel als accounting−protocol worden gebruikt. Het mist toch enkele eigenschappen zoals end−to−end security, hoewel security in latere versies van SNMP beter is geregeld. Ook het ontbreken van een goede mogelijkheid om event−driven−accounting uit te voeren en het onbevestigde karakter van SNMP berichten zorgt voor beperkte toepasbaarheid. Verder is SNMP een protocol waarmee een grote hoeveelheid aan management−specifieke taken kan worden uitgevoerd. Een protocol dat zo algemeen is, is moeilijk in te passen op een plek waarbij zulke specifieke requirements spelen als bij accounting.

A.6. MSIX

Metered Service Information eXchange (MSIX) (spreek uit als em siks) is een protocol dat is ontwikkeld om gegevens over het gebruik van communicatie−diensten (bijvoorbeeld inbelfaciliteiten van een Internet−provider) te beschrijven en te transporteren. MSIX is echter niet beperkt tot een bepaalde dienst of een beperkte verzameling van diensten.

MSIX is ontwikkeld door NetCentric en Compaq met hulp van zo'n twintig andere bedrijven [msixwww]. De eerste revisie is gepresenteerd op 1 juni 1997. In 1999 is het werk aan MSIX opgepakt door MatraTech en onder de hoede van de Internet Engineering Task Force (IETF) gebracht. MSIX is momenteel beschreven in een Internet draft [bacct−msix].

MSIX is gebaseerd op XML [xmlw3c] (Extensible Markup Language) en is dus ook human−readable. XML is wel een formaat dat veel tekst gebruikt om relatief eenvoudige dingen te beschrijven, maar door gebruik van compressie valt de grootte flink te reduceren. MSIX is een recordformat dat voorziet in het definiëren van services en het versturen van gebruik van deze services. Het heeft geen authenticatie− en autorisatie−gedeelten en is dus geen algemeen AAA−protocol.

Een belangrijke eigenschap van MSIX is dat er 'compound' services en sessies mee kunnen worden gedefinieerd. Hierin kan een parent−service worden gedefinieerd waaronder verschillende child−services hangen. Een voorbeeld hiervan is teleconferencing waarbij verschillende verbindingen worden gebruikt. Deze kunnen dan onder één noemer worden bijgehouden.

Bij MSIX is geen transportprotocol gedefinieerd, maar wordt het gebruik van een HTTP/SSL/TCP/IP−stack gesuggereerd. De basis van het MSIX−bericht wordt in *Figuur 10.8* beschreven. De timestamp geeft het moment aan waarop het bericht is gegenereerd, de uid wordt gebruikt om een unieke identificatie aan het bericht te geven.

```
<?xml version=1.0?>
<msix version="1.2" timestamp="2001−01−29T09:33:25Z" uid="[uid]">

   ...

</msix>
```

Figuur 10.8: MSIX bericht.

De MSIX−berichten die zijn gedefinieerd, staan vermeld in *Tabel 10.10*. Alle berichten worden door het service−element geïnitieerd. De MSIX server beantwoordt deze berichten. Het status−bericht kan aan een bericht (in het algemeen een response van de server) worden toegevoegd om aan te geven of het request correct is ontvangen en afgehandeld. Met deze berichten worden diensten gedefinieerd en accounting−berichten beschreven.

Message	
defineservice	definitie van een dienst
beginsession	geeft aan dat dienst is begonnen
updatesession	geeft update over dienst
commitsession	geeft het einde van de sessie aan
abortsession	maak sessie ongedaan (bewaar geen gegevens)
getversions	vraag ondersteunde MSIX versies op
relateservices	breng een parent−child relatie tussen gedefinieerde diensten aan.
status	geef status (bijvoorbeeld foutmelding) van request aan
defineservicers	antwoord op defineservice
beginsessionrs	antwoord op beginsession
updatesessionrs	antwoord op updatesession
commitsessionrs	antwoord op commitsession
abortsessionrs	antwoord op abortsession
relateservicesrs	antwoord op relateservices
getversionrs	antwoord op getversions

Tabel 10.10: MSIX−berichten.

Een voorbeeld van een definitie van een dienst is beschreven in *Figuur 10.9*. Hierin wordt een telefoondienst beschreven. Dit bericht wordt verstuurd in een MSIX−bericht zoals in *Figuur 10.8* is beschreven. Bij het versturen van accounting−gegevens wordt een referentie opgenomen naar de gedefinieerde dienst (server.net/FoneCall in ons voorbeeld). In *Figuur 10.10* wordt een beginsession bericht gestuurd voor de gedefinieerde sessie. Dit is een start−bericht. De MSIX−server beantwoordt deze met een bericht als beschreven in *Figuur 10.11*.

```
<defineservice>
   <dn>server.net/Fonecall</dn>
   <version>7.3</version>
   <description>Internet to PSTN telephone call</description>
   <ptype>
      <dn>AccountId</dn>
      <type>STRING</type>
   </ptype>
   <ptype>
      <dn>DialedNumber</dn>
      <type>STRING</type>
   </ptype>
   <ptype>
      <dn>Duration<dn>
      <type>INT32</type>
   </ptype>
   <ptype>
      <dn>StartTime</dn>
      <type>TIMESTAMP</type>
   </ptype>
</defineservice>
```

Figuur 10.9: MSIX service definitie.

```
<beginsession>
   <uid>[sessionid]</uid>
   <dn>server.net/FoneCall</dn>
   <property>
      <dn>AccountId</dn>
      <value>324955</value>
   </property>
   <property>
      <dn>DialedNumber</dn>
      <value>+16177205200</value>
   </property>
   <property>
      <dn>Duration</dn>
      <value>280</value>
   </property>
   <property>
      <dn>StartTime</dn>
      <value>2001-01-29T09:35:32Z</value>
   </property>
</beginsession>
```

Figuur 10.10: eenvoudige MSIX accounting−request.

```
<beginsessionrs>
   <status>
      <code>msix.org/200</code>
   </status>
   <uid>[sessionid]</uid>
</beginsessionrs>
```

Figuur 10.11: eenvoudige MSIX accounting−response.

MSIX is gebaseerd op XML en gebruikt dus relatief veel karakters om een accounting−bericht te coderen. MSIX is echter, in tegenstelling tot andere accounting−protocollen, wel ontworpen om accounting−gegevens over verschillende soorten diensten te transporteren. Met MSIX kunnen diensten worden gedefinieerd en relaties tussen verschillende diensten worden aangegeven.

Bij MSIX wordt HTTP als transportprotocol gesuggereerd, dat voor een aanzienlijke overhead zorgt. HTTP is allesbehalve eenvoudig en biedt veel meer functionaliteit dan voor een accounting−protocol nodig is. Het is niet waarschijnlijk dat, met gebruikmaking van XML en HTTP, accounting−berichten binnen een erg kort tijdsbestek kunnen worden geproduceerd, verstuurd en verwerkt. Een ander nadeel van HTTP is dat de accounting−server geen verbindingen kan initiëren naar de client voor het doen van polling.

Tijdens het transport worden gegevens via de SSL−laag gecodeerd. Dit betekent dat de encryptie hop−by−hop plaatsvindt. Er wordt geen ondersteuning geboden voor end−to−end codering.

Momenteel ligt de ontwikkeling en het gebruik van MSIX vrijwel stil. Er zijn niet of nauwelijks toepassingen beschikbaar waarin het wordt gebruikt.

A.7. ADIF

ADIF staat voor Accounting Data Interchange Format en is momenteel gedefinieerd in een Internet draft [roamops−actng]. ADIF is gebaseerd op MIME [rfc1521] en is human−readable. Het is ontworpen om accounting−gegevens van een bestaand protocol op een compacte manier in tekst weer te geven. ADIF is dus geen volledig accounting−protocol, maar een recordformat om accounting−data uit bestaande protocollen weer te geven.

Een ADIF bericht bestaat uit een header met algemene informatie over de records in het bericht, gevolgd door een aantal records die worden gescheiden door een scheidingsteken. In de header staat het versienummer, de naam of omschrijving van het service−element en een timestamp die het begin van het verzamelen van gegevens aangeeft. Optioneel wordt een protocol aangegeven van waaruit de attributen worden gebruikt.

Elk record bestaat uit een of meerdere regels. Zoals in MIME gebruikelijk is kunnen grote strings op de volgende regel worden voortgezet door te beginnen met een spatie of tab. Regels die met het ’#’ symbool beginnen worden als commentaar opgevat. ADIF ondersteunt het gebruik van attributen uit elk ander protocol. Dit gebeurt door attributen uit andere protocollen te gebruiken. Attributen worden aangegeven door de naam

van het protocol waaruit ze afkomstig zijn, gevolgd door het nummer van het attribuut. Zo geeft `radius//46` het `Acct−Session−Time` attribuut (attribuut 46) uit het RADIUS protocol aan. Als in de header een default protocol is aangegeven kan worden volstaan met het nummer van het attribuut. ADIF voorziet ook in attributen met sub−attributen en het gebruik van aliassen voor delen van lange object−identifiers.

```
version: 1
device: server3
descripton: Accounting Server 3
date: 29 Jan 2001 10:40:12 +0100
defaultProtocol: radius

rdate: 29 Jan 2001 10:41:17 +0100
#NAS−IP−Address
4: 204.45.34.12
#NAS−Port
5: 12
#NAS−Port−Type
61: 2
#User−Name
1: fred@bigco.com
#Acct−Status−Type
40: 2
#Acct−Delay−Time
41: 14
#Acct−Input−Octets
42: 234732
#Acct−Output−Octets
43: 15439
#Acct−Session−Id
44: 185
#Acct−Authentic
45: 1
#Acct−Session−Time
46: 1238
#Acct−Input−Packets
47: 153
#Acct−Output−Packets
48: 148
#Acct−Terminate−Cause
49: 11
#Acct−Multi−Session−Id
50: 73
#Acct−Link−Count
51: 2
```

Figuur 10.12: voorbeeld ADIF file.

ADIF is geen accounting−protocol, maar een manier om accounting−gegevens op te slaan. Daarom kan het ook niet echt worden vergeleken met de andere protocollen. ADIF is gebaseerd op MIME, dat human−readable is, terwijl het behoorlijk compact blijft. Met ADIF worden alleen volledige accounting−records opgeslagen en er is dus ook geen sprake van accounting−berichten. ADIF kent geen datatypen, alles wordt als tekst opgeslagen.

ADIF is echter goed te gebruiken als algemeen recordformat, omdat het de attributen van andere protocollen gebruikt en op die manier uitstekend uitbreidbaar is. ADIF wordt bijvoorbeeld ook gebruikt om accounting−gegevens in DIAMETER vast te leggen.

A.8. TIPHON

TIPHON (Telecommunications and Internet Protocol Harmonization Over Networks) is een initiatief van de ETSI (European Telecommunications Standards Institute) [etsiwww] om telefoonverkeer over IP en over geschakelde netwerken te combineren. Zo is telefonie mogelijk tussen gebruikers die telefoneren via IP−netwerken en gebruikers die aan een geschakeld netwerk zijn aangesloten, zoals PSTN, ISDN of GSM.

Om gegevens over inter−domain pricing, autorisatie en gebruik uit te wisselen is in [ts101321] een op XML−gebaseerd protocol beschreven. Gegevens worden via een HTTP−verbinding uitgewisseld met gebruikmaking van SSL− of TLS−beveiliging. TIPHON is echter niet toepasbaar als algemeen accounting−protocol, omdat het specifiek is toegespitst op telefoongegevens.

In *Figuren 10.13* en *10.14* is een voorbeeld van een Usage Exchange gegeven. De client geeft aan dat een telefoonsessie 10 minuten heeft geduurd. De server genereert hierop een bevestiging.

```
<?xml version=1.0?>
<Message messagId="234565432" random="87654321">
   <UsageIndication componentID="13579990">
      <Timestamp>
         2001−01−29T09:44:45Z
      </Timestamp>
      ...
      <SourceInfo type="e164">
         81458811202
      </SourceInfo>
      <DestinationInfo type="e164">
         4766841360
      </DestinationInfo>
      ...
      <UsageDetail>
         <Amount>
            10
         </Amount>
         <Increment>
            60
         </Increment>
         <Unit>
            s
         </Unit>
      </UsageDetail>
   </UsageIndication>
</Message>
```

Figuur 10.13: vereenvoudigd TIPHON UsageIndication bericht.

```
<?xml version=1.0?>
<Message messagId="234565432" random="87654321">
   <UsageConfirmation componentID="13579990">
      <Timestamp>
         2001−01−29T09:45:35Z
      </Timestamp>
      <Status>
         <Code>
            201
         </Code>
         <Description>
            new usage information created
         </Description>
      </Status>
   </UsageConfirmation>
</Message>
```

Figuur 10.14: vereenvoudigd TIPHON UsageConfimation bericht.

A.9. OFX

OFX staat voor Open Financial eXchange en is in 1997 ontwikkeld door Intuit, Microsoft en Checkfree. OFX is een protocol om financiële gegevens te transporteren en is bedoeld om on−line bankieren en e−commerce dienstverlening over het Internet mogelijk te maken. Met OFX kunnen financiële gegevens tussen financiële instellingen, ondernemingen en klanten worden uitgewisseld. De OFX specificatie is beschikbaar op de Internet−site [ofxwww].

Bij gebruik van OFX wordt direct een prijskaartje aan gebruik toegekend en is dus niet algemeen geschikt als accounting−protocol. OFX kan wel worden gebruikt voor on−line billing als accounting wordt toegepast. OFX ondersteunt een grote diversiteit aan financiële taken, waaronder electronic banking voor kleine en middelgrote klanten, bill payment voor kleine klanten en bedrijven en het sturen van rekeningen.

Ook OFX gebruikt XML om gegevens vast te leggen. In *Figuur 10.15* is een voorbeeld gegeven waarbij de gebruiker Jan vraagt welke rekeningen sinds 2000−12−05 zijn binnengekomen. In *Figuur 10.16* is een sterk vereenvoudigd antwoord op deze vraag weergegeven.

```
<OFX>
   <SIGNONMSGSRQV2>
      <SONRQ>
         <DTCLIENT>20010129095712    <!--Current date -->
         <USERID>123-45-6789         <!--Jan's user id-->
         <USERPASS>JansPassword
         <LANGUAGE>ENG
         <APPID>EndUserApp
         <APPVER>0700
      </SONRQ>
   </SIGNONMSGSRQV2>
   <PRESDLVMSGSRQV1>
      <PRESLISTTRNRQ>
         <TRNUID>12345
         <PRESLISTRQ>
            <BILLPUB>ABillPublisher
            <DTSTART>20001205000000 <!--Get bills since-->
            <NOTIFYWILLING>Y
            <INCLUDEDETAIL>Y
         </PRESLISTRQ>
      </PRESLISTTRNRQ>
   </PRESDLVMSGSRQV1>
</OFX>
```

Figuur 10.15: OFX bill request..

```
<OFX>
   <PRESDLVMSGSRSV1>
      <PRESLISTTRNRS>
         <PRESLISTRS>
            <BILLPUB>ABillPublisher
            <USERID>123-45-6789
            <DTSTART>20001208000000
            <DTEND>20010109000000
            <PRESLIST>
               <PRESBILLINFO>
                  <AMTDUE>124.24      <!--to pay $124.24-->
                  <DTPMTDUE>20010201  <!--by 1/2/01 -->
                  <BILLDETAILTABLE>
                     <TABLENAME>usage
                     <BILLDETAILTABLETYPE>x_Power_usage
                     <BILLDETAILROW>
                        <C>elec     <!--Consumable-->
                        <C>20001208 <!--Date meter start->
                        <C>65543    <!--Meter reading-->
                        <C>20010109 <!--Date meter end-->
                        <C>65643    <!--Meter reading-->
                        <C>100      <!--Difference-->
                        <C>KWH      <!--Units-->
                        <C>.8934    <!--Rate-->
                        <C>89.34    <!--Charge-->
                     </BILLDETAILROW>
                     <BILLDETAILROW>
                        ... another detail row of the bill
                     </BILLDETAILROW>
                  </BILLDETAILTABLE>
               </PRESBILLINFO>
               <PRESBILLINFO>
                  ... another bill
               </PRESBILLINFO>
            </PRESLIST>
         </PRESLISTRS>
      </PRESLISTTRNRS>
   </PRESDLVMSGSRSV1>
</OFX>
```

Figuur 10.16: vereenvoudigde OFX bill response