CAM Programmeren voor dummies

"Licht in de duisternis van programmatuurontwikkeling voor CAMS en smartcards"

Versie 2.0

Inhoudsopgave

1. Inleiding

Ontvangst via een schotel en receiver was vier maanden geleden nog geheel nieuw voor mij. De eerste maanden ben ik bezig geweest met mijn decoder, een Manhattan Mx met geïntegreerde Matrix Reloaded CAM. Geweldig op gang geholpen door mijn leverancier kon ik al snel wat spelen met emulaties en titanium en fun kaarten.

Ik liep natuurlijk net als iedereen regelmatig tegen het merkwaardige fenomeen aan dat schermen zomaar ineens volledig zwart werden. De sleuteltjes tot de oplossing van dit probleem bleken vaak wel beschikbaar. Of deze echter tezamen met eventuele andere vereiste aanpassingen ook in emulaties of kaarten beschikbaar kwamen, bleek kennelijk een overwegend commerciële afweging te zijn. Voor de nodige euri kon de opvolger van een CAM of kaart worden aangeschaft omdat updates voor de voorganger niet meer verschenen. Terwijl het mij nooit echt duidelijk is geworden wat er in technologisch opzicht nou daadwerkelijk was verbeterd aan de opvolger.

De voor het ontwikkelen van een emulatie benodigde kennis lijkt schaars en voor zover ik kan zien wordt deze bewust schaars gehouden. Dit feit draagt natuurlijk ook niet bij aan het ontworstelen aan de wurggreep van de leveranciers. Terwijl zij zonder deze gedwongen verkoop veel meer gestimuleerd zouden worden tot het produceren van daadwerkelijk snellere en betere hardware met meer mogelijkheden.

Het lijkt mij een goede zaak dat meer mensen zich bezighouden met de ontwikkeling van software voor CAMS en kaarten. Natuurlijk schuilt hier het risico in van ondeugdelijke emulaties en smakeloze grappen, maar het voordeel van de onafhankelijkheid van producenten lijkt mij hier tegenop te wegen. Open source heeft zich uiteindelijk ook op andere vlakken bewezen. Bovendien zouden maatregelen mogelijk zijn zoals een “certificering” van een emulatie door een partij als sat4all.

Ik heb me daarom voorgenomen deze materie te onderzoeken en te kijken of ik enerzijds zelf tot een emulatie zou kunnen komen en anderzijds gaandeweg mijn bevindingen in een soort van tutorial zou kunnen vastleggen. Dit laatste in de hoop ook anderen te stimuleren zich meer in deze materie te verdiepen. Door de bevindingen gaandeweg te publiceren, hoop ik op terugkoppeling van anderen die verder gevorderd zijn dan ik, zodat ik dit weer in dit document kan verwerken. Hiermee kan het een document van ons allemaal worden.

Mijn persoonlijke achtergrond ligt weliswaar in software-ontwikkeling, maar dan voor financieel administratieve systemen. Navraag leerde mij dat de software voor CAM’s in de programmeertaal C en/of assembly kon worden geschreven. Nu heb ik in een grijs verleden wel eens een jaartje C geprogrammeerd en mijn kennis van assembly is puur theoretisch, maar ik heb wel het gevoel dat ik dit met de nodige studie zou moeten kunnen leren. Ik heb voor mijzelf een werkplan gemaakt dat mij de volgende kennis zou moeten opleveren:

- Algemene kennis van (gecodeerde) satelliet uitzendingen
- Algemene kennis van de werking van CAMS en kaarten
- Programmeerkennis van C en/of assembly
- Gedetailleerde kennis van een specifieke CAM
- Gedetailleerde kennis van minimaal één van de bekende coderingen (seca, conax, etc.)

Met deze kennis zou men uiteindelijk toch in staat moeten zijn om software voor een CAM te ontwikkelen.

Om deze kennis te verkrijgen bestudeer ik de nodige documenten en websites en val ik medemensen lastig. Wat ik daarvan begrijp en waarvan ik denk dat het zinvol is voor de doelstelling, neem ik in dit document op. Daarbij neem ik zoveel mogelijk verwijzingen op naar de sites met de achtergrondinformatie. Informatie die mij ook interessant lijkt, maar die niet direct vereist lijkt voor de doelstelling, neem ik op in appendices.

Het document komt dus in stappen tot stand en bij iedere afronding van een fase, of na het verkrijgen van input voor correcties zal een nieuwe versie worden gemaakt. De x.0 (punt nul) versie zal altijd het document zijn met de eerste poging tot het verstrekken van nieuwe informatie. In documenten met een hoger subnummer zullen fouten zijn gecorrigeerd of aangedragen verbeteringen zijn aangebracht.

Of het document ooit klaar en volledig zal zijn? Geen idee. Zal het gewenste effect worden bereikt? Geen idee. Maar het is in ieder geval wel leuk om eraan te werken!

Hermanator
3 oktober 2004

Met dank aan: Bommeltje, Duwgati, Pic-o-matic, MiLo, Ozzo, Wildcard, EnEmA, John43 en anderen.

2. Een globaal overzicht

2.1 MPEG-2

Een uitzending waarnaar je via de schotel kijkt is eigenlijk een **MPEG-2** bestand zoals ook de talloze filmpjes waarmee we elkaar bestoken in de mail. Het bestand wordt in dat geval uit je mail gehaald en opgeslagen op je harddisk waarna het met bijvoorbeeld de Mediaplayer van Windows wordt afgespeeld. Een gesimplificeerde voorstelling is dat nu dit bestand als "**datastream**" wordt verzonden vanaf de satelliet. De satellietontvanger speelt deze datastream direct tijdens het ontvangen al af op de televisie. Deze datastream kun je zien als een lange reeks bytes van het oorspronkelijke MPEG-2 bestand, netjes gegroepeerd in **packets,** met hierin toegevoegde controle-informatie om een correcte ontvangst te garanderen.

De werkelijkheid is vanzelfsprekend iets complexer. Een satelliet bestaat bijvoorbeeld uit verschillende transponders die meerdere transportstreams kunnen uitzenden met daarbinnen meerdere substreams (**PES, Packetized Elementary Stream**, geïdentificeerd met een PID nummer: de programma's, teletekst, EPG, etc.). De programma's kunnen weer over meerdere satellieten zijn gegroepeerd in een boeket, waarop men zich kan abonneren. Binnen een PES worden tussen de packets met de daadwerkelijke informatie (beeld, geluid, etc.) ook packets met besturingsgegevens tussengevoegd. Hierin bevinden zich gegevens over het uitgezonden programma, de codering, abonnementen, etc. De ontvanger filtert deze berichten uit de datastream voor verdere verwerking. Wat meer achtergrondinformatie hierover vinden we in Appendix 1.

2.2 Scrambling

Onder andere om ervoor te zorgen dat alleen "bevoegden" de uitzending kunnen zien, wordt de datastream **scrambled** (gecodeerd) verzonden. Dit uitzenden van MPEG-2 via de satelliet en de scrambling hiervan is in Europa gestandaardiseerd in de **DVB-s** specificaties (**Digital Video Broadcast via Satelliet**). De scrambling zelf vindt plaats met het gestandaardiseerde DVB Common Scrambling Algorithm (**CSA**).

Deze CSA scrambling voorziet in codering met het zogenaamde **controlword**. Aan de hand van dit controlword wordt een algoritmische bewerking op de datastream uitgevoerd, waardoor dit een onherkenbare reeks tekens wordt. In de ontvanger kan, mits dit controlword bekend is, via de omgekeerde bewerking de datastream weer worden omgezet naar een herkenbare MPEG-2 datastream. De ontvanger kan deze dan, feitelijk net als de Windows Mediaplayer, op de televisie laten zien.

2.3 ECM Entitlement Control Message

Het controlword wordt om de 2 tot 10 seconden vervangen (door een random controlword generator bij het uitzenden) om de beveiliging te verhogen. Alleen met het juiste controlword, op het juiste moment heb je dus (gedurende 2-10 seconden) beeld. Je ontvanger moet dus ook met diezelfde frequentie worden voorzien van het nieuwe controlword, om continu beeld te kunnen tonen. De controlwords worden hiertoe in de datastream in een speciaal soort berichten (in aparte packets) tussengevoegd: de **ECM** (**Entitlement Control Message**). Deze ECM's worden uit de datastream gefilterd en apart behandeld. Het controlword wordt uit een ECM bericht gehaald en hiermee kan weer een stukje van de datastream worden ontcijferd en op de televisie worden getoond. Tot het volgende ECM bericht wordt ontvangen met een nieuw controlword. Ook andere, op dit moment voor ons begrip nog niet relevante informatie over de uitzending wordt met ECM berichten verzonden.

2.4 Encryptie / Keys

Nu gaat het er om wie de controlwords uit de ECM's kunnen en mogen halen. Hier komen de coderingen **Seca, Conax, Irdeto**, etc. om de hoek kijken. Op het ECM bericht wordt bij een "beschermde" uitzending namelijk een **encryptie** losgelaten van de soort codering waarvoor de provider heeft gekozen. Alleen als een ontvanger in staat is het ECM bericht te decrypten, kan het controlword worden verkregen en kan de datastream correct worden descrambled. Er is dus een duidelijk verschil tussen scrambling en encryption:

- **Scrambling** is de standaard DVB-s CSA codering, waarbij alle uitzendingen worden versleuteld met controlwords.
- **Encryption** is een aparte codering om de meegezonden controlwords te beveiliging, zodat de uitzending alleen descrambled kan worden door hiertoe gerechtigde kijkers.

Als een provider uitzendt in meerdere coderingen (zoals bijvoorbeeld bij Nederlandse zenders Seca/Irdeto) dan worden gewoon voor beide coderingen ECM berichten verzonden met de juiste controlwords.

Omdat alle uitzendingen plaatsvinden volgens de DVB-s specificaties plaatsvinden, wordt een **FTA (Free to Air)** uitzending wél op dezelfde manier gescrambled, maar worden de ECM berichten met de controlwords níet encrypted. Hierdoor kan dus met iedere ontvanger de uitzending worden bekeken.

Voor "beschermde" uitzendingen moet een ontvanger dus in staat zijn om de ECM berichten hierin te kunnen decrypten. Het encryptie-algoritme op zich is niet voldoende, want dan zou iedere ontvanger alle uitzendingen in de hem bekende encryptie op de televisie kunnen toveren. De ECM berichten worden hierom encrypted met periodiek (bijvoorbeeld wekelijks of maandelijks) wijzigende **keys**. Deze keys zijn dan ook de gewilde sleuteltjes waar we altijd zo druk naar zoeken. Helaas wijzigt een provider ook nog wel eens andere onderdelen van het encryptieproces, en dan redden alleen de sleuteltjes ons niet.

2.5 EMM (Entitlement Management Message)

Hoe komen deze periodiek wijzigende keys dan weer in de ontvangers van de abonnees? Hiervoor is een nieuwe soort berichten bedacht: de **EMM (Entitlement Management Message)** berichten. Ook deze berichten worden met dezelfde encryptie (seca, irdeto, etc.) versleuteld en in de datastream tussengevoegd. Maar aangezien de EMM bedoeld is om alleen diegenen met een geldig abonnement te voorzien van o.a. nieuwe keys, moeten deze met een per abonnement unieke key worden versleuteld.

Een abonnee heeft een **smartcard** met abonnementsgegevens, waaronder een unieke **masterkey**. Een EMM bericht met onder andere de nieuwe keys wordt nu encrypted met deze masterkey. Een EMM bericht kan dus uitsluitend met behulp van deze ene specifieke smartcard worden ontsleuteld. De nieuwe keys in het EMM bericht worden bijgewerkt op de smartcard. Met de EMM's worden nog andere kenmerken van het abonnement en het algoritme verzonden en bijgewerkt op de smartcard. De codering kent daarom naast het algoritme ook commando's om de kaart te bewerken. Samenvattend:

- Iedere **datastream** wordt ge**scrambled** met **controlwords;**
- Die controlwords worden in de datastream meegezonden in **ECM berichten;**
- Deze ECM berichten kunnen worden **encrypted** met **keys;**
- Deze keys worden periodiek verzonden in een **EMM bericht** per abonnement;
- Een EMM bericht wordt encrypted met een per **smartcard** unieke **masterkey;**
- Op de smartcard staat deze masterkey en worden de keys bijgewerkt.

2.6 Cam (Conditional Access Module)

Wat is nu de plaats van de CAM in dit hele proces? Bij oudere ontvangers zit alle genoemde functionaliteit in de ontvanger. Deze kent dan ook meestal maar één encryptie zoals bijvoorbeeld de door Canal Digitaal "goedgekeurde" ontvangers. Het enige dat dit type ontvanger nog nodig heeft is een geldige smartcard. Met dit type ontvanger kunnen we dan ook uitsluitend FTA uitzendingen bekijken en de uitzendingen die zijn gecodeerd in de encryptie die de ontvanger kent.

Een modernere ontvanger heeft één of meerdere zogenaamde **CI slots (Common Interface)**. Een CI slot is een sleuf in de ontvanger waarin een zogenaamde **CAM (Conditional Access Module)** kan worden gestoken. Een CAM heeft de vorm van een PCM/CIA module met een sleuf waarin een smartcard kan worden geschoven. De benodigde functionaliteit om de encryptie aan te kunnen zit nu in de CAM. Zo zou je bijvoorbeeld een Conax Cam of een Seca Cam kunnen aanschaffen. Hierdoor hoef je voor iedere encryptie in ieder geval geen aparte ontvanger meer te hebben. Je verwisselt gewoon de CAM. Het enige dat je dan nog nodig hebt is een smartcard met een abonnement voor de programma's die je wilt kunnen zien.

Om er onder andere voor te zorgen dat fabrikanten van ontvangers en fabrikanten van CAM's onafhankelijk van elkaar producten kunnen ontwikkelen die toch met elkaar samenwerken is het uitwisselingsprotocol (de interface) tussen de ontvanger en de CAM gestandaardiseerd. Dit zijn de specificaties voor het CI slot en deze zijn vastgelegd in "*EN 50221 Common Interface Specification for Conditional Access and other Digital Video Broadcasting Decoder Applications*". Omdat dit de specificaties zijn van hoe de CAM communiceert met de ontvanger, zijn dit voor onze studie ongelooflijk belangrijke specificaties.

2.7 UCAS CAM

Nog fraaier is de **UCAS (Universal Common Access System) CAM** die meerdere coderingen aankan**,** zoals bijvoorbeeld de Magic Module, de Matrix, de Xcam en de Dragon. Nu hebben we in principe nog maar één CAM nodig in onze ontvanger. En om het "ongemak" van geldige abonnementskaarten te voorkomen, hebben deze CAM's software aan boord, die naar de ontvanger toe net doen alsof ze een smartcard in de sleuf hebben. Deze software noemt men een **emulatie**. Het is deze emulatie die we downloaden van internet en in de CAM laden met onze CAS, Multipro, etc. De sleuteltjes die normaal van de kaart worden gehaald, zijn nu in het geheugen van de CAM geladen in de emulatie. Deze sleuteltjes kunnen in de meeste emulaties ook met de afstandsbediening van de ontvanger worden bewerkt. En als de software in de CAM er zelf niet uitkomt, kan hij altijd nog te rade gaan bij een titaniumkaart, funcard of andere all purpose card met gegevens van diverse providers.

2.8 En nu verder…

In dit hoofdstuk hebben we de algemene werking van (gecodeerde) uitzendingen via de satelliet in kaart gebracht en de plaats van de CAM hierin bepaald. In het volgende hoofdstuk gaan we de interne werking van de CAM en de wijze waarop de CAM met de ontvanger communiceert, nader onderzoeken.

Literatuurverwijzingen:

http://www.duwgati.com
http://www.satinfo.org/archive/Documentation/CAS-model.pdf
http://www.nhk.or.jp/strl/publica/bt/en/le0012.pdf
http://users.pandora.be/satelliet/mpegtrans.pdf
http://www.videoaudioreport.nl/index.php?action=1&catno=57&artno=1079&category=2004-april

3. De CAM: het model

3.1 Inleiding

In het voorgaande hoofdstuk hebben we een globaal inzicht gekregen in de wijze waarop door de providers de digitale uitzendingen via de satelliet worden gescrambled, encrypted en verzonden. Tevens hebben we gezien wat de plaats en de functie van de CAM hierin is. Nu is de tijd gekomen om ons vergrootglas op de CAM te richten en de onderdelen en de werking van de CAM nauwkeurig in kaart te brengen.

Zoals we in het voorgaande hoofdstuk hebben gezien is de CI interface, het slot waar we de CAM in doen, gestandaardiseerd. Het document met deze beschrijving, "*EN 50221 Common Interface Specification for Conditional Access and other Digital Video Broadcasting Decoder Applications*" is dan ook het uitgangspunt voor dit onderdeel van onze studie. Het overnemen van alle details uit EN50221 zou niet alleen overbodig zijn, maar zou ook niet bijdragen aan de leesbaarheid van dit rapport. Aan bestudering van EN50221 naast dit rapport, kan dus helaas niet worden ontkomen! Hopelijk zal dit rapport de bestudering van EN50221 wel vergemakkelijken.

3.2 De structuur

In EN50221 vinden we deze structuurweergave van de CAM en de CI interface.

Links zien we de **host** (receiver) en rechts de **module** (CAM) die met elkaar communiceren over de CI interface in het midden (**PC card interface**)

De MPEG-2 **transportstream** komt over de CI-interface de CAM in en wordt -eventueel (gedeeltelijk) descrambled- teruggeven aan de host (de receiver).

Besturingscommunicatie tussen de Module (cam) en de Host (ontvanger) loopt over de **Command Interface**.

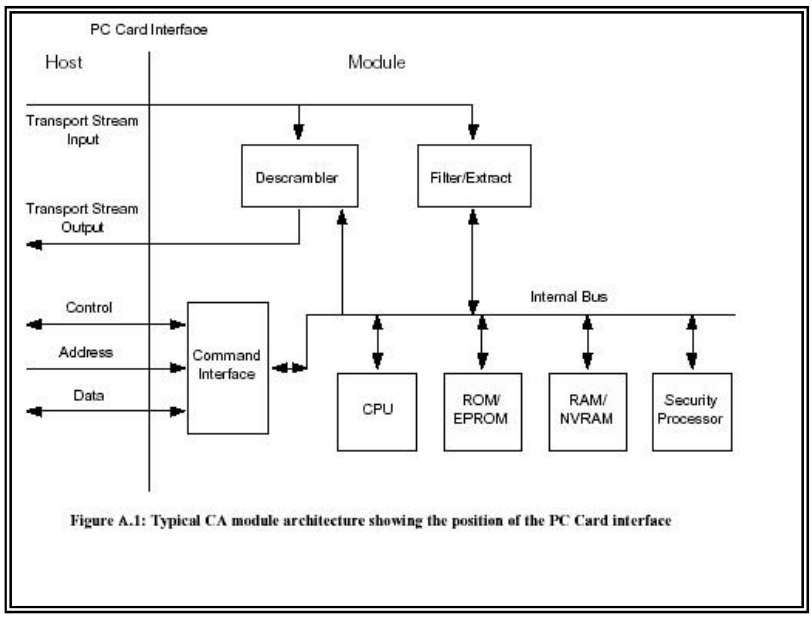


Figure A.1: Typical CA module architecture showing the position of the PC Card interface

Descrambler	De descrambler decodeert selectief (delen van) de MPEG-2 transportstream. Dit gebeurt door de CPU periodiek controlwords te laten laden in de descrambler.
Filter/extract	Dit circuit filtert de besturingsinfo (ECM en EMM) uit de datastream.
CPU	Deze processor draait de applicatie(s) van het CA proces (onze emulatie) en bestuurt de dataflow door de module en over de CI interface.
ROM/EPROM & RAM/NVRAM	In dit geheugen bevindt zich de programmatuur van het CA proces (onze emulatie dus) en de hiervoor benodigde data (gegevens).
Security processor	Een aparte processor met een hogere beveiliging dan de CPU. Deze voert beveiligingsfuncties uit zoals decryptie, en beheert beveiligingsgegevens zoals keys, en entitlements. Hij kan zich in de module zelf bevinden, maar ook daarbuiten, op een aparte smartcard.

3.3 Algemene werking

Er wordt gesproken over host en module. Een **host** is het apparaat dat gebruik maakt van de module en betreft in ons geval dus de ontvanger. Een **module** is een apparaat dat niet zelfstandig werkt, maar in samenwerking met een host taken kan vervullen, in ons geval is dit dus de CAM.

De CI interface, het slot waar de CAM in de ontvanger wordt gestoken, is gedefinieerd op basis van het concept van applicaties die gebruik maken van resources. Een **applicatie** is een programma dat draait op een module en biedt de gebruiker aanvullende functionaliteit als aanvulling op de functionaliteit van de host (bijvoorbeeld descrambling). Een **resource** is een verzameling functionaliteit die zowel kan worden aangeboden door host als de module en die door een applicatie kan worden gebruikt. Enkele voorbeelden van resources zijn de MMI (Man machine interface), de Conditional Access Support en de Smart Card Reader Resource.

Communicatie tussen een applicatie en een resource vindt plaats in een session waarin objecten worden uitgewisseld die door de resource zijn gedefinieerd. Een **session** is een “gesprek” met een duidelijk begin en einde tussen een applicatie en een resource.

Een **object** is een bericht, bestaande uit een reeks bytes (tekens) die op een gestructureerde manier is ingedeeld: een **tag** (label) die aangeeft wat voor soort object het is, een lengteveld waarin staat hoeveel databytes er volgen en de daadwerkelijke databytes (de inhoud van het “bericht”). De uitwisseling van objecten binnen een session vindt plaats volgens een protocol. Een **protocol** is een voorgeschreven wijze waarop de objecten moeten worden uitgewisseld, zoals wie welke objecten mag verzenden en welk object een antwoord is op welk object. Zo zijn er bijvoorbeeld objecten om een session te openen en te sluiten.

De CI interface bestaat uit twee logische componenten: de **Transport Stream Interface** en de **Command Interface**. Beide logische componenten delen **dezelfde fysieke interface**: de PC-card interface.

- Over de **Transport Stream Interface** loopt de MPEG-2 transport stream in twee richtingen, de module in en uit.
- Over de **Command Interface** communiceren de host en de module hun commando’s d.m.v. objecten.

In de MPEG-2 transportstream kunnen meerdere **PES** stromen (Packetized Elementary Stream) zitten welke bij elkaar gegroepeerd kunnen zijn in een **Service** en samen een televisieprogramma vormen (beeld, geluid, ondertiteling, etc.)

Als de MPEG-2 transportstream gescrambelde pakketten bevat én de module toegang kan verlenen tot deze service én de host heeft deze service geselecteerd, dan zullen de packets van deze service descrambled worden teruggegeven aan de host. Zo niet, dan worden de packets ongewijzigd retour gegeven. De module haalt zelf de voor descrambling benodigde elementen uit de transportstream zoals ECM en EMM messages.

Het is mogelijk om meerdere modules in één host te plaatsen. Het uitgangspunt van de CI interface is dat deze modules als het ware serieel geschakeld worden. De transportstream loopt de eerste module in en terug naar de host. Van de host naar de volgende module en weer terug. De transportstream wordt dus als het ware door alle modules geleid.

Het is aan te raden het bovenstaande nog eens aandachtig door te lezen. Een goed begrip van deze basiskennis is onmisbaar om de nu volgende beschrijving en EN50221 goed te kunnen begrijpen!

3.4 Objecten en Protocollen (Command Interface)

Alvorens de beschikbare resources in beeld te brengen waar we met onze applicatie(s) gebruik van kunnen maken, gaan we eerst nog wat dieper in op het principe van objecten en protocollen. Dit is uiteindelijk de taal waarin we met de ontvanger zullen gaan praten, dus zullen we dit tot op "bitniveau" moeten begrijpen.

Stel, we willen een session starten met de Resource Manager (zie 3.5.1 De Resource Manager). We stellen hiertoe volgens het hiervoor geldende protocol een **Open_session_request** object samen:

Open_session_request		
Open_session_request_tag	**Length field**	**Resource identifier**
91	04	00010041
soort object	*omdat er 4 bytes volgen*	*id van de Resource Manager, uit tabel*

Wat we dus verzenden aan de host is de volgende reeks van 6 bytes (hexadecimaal weergegeven): "**910400010041**".

Volgens het protocol gaat de host ons nu beantwoorden met **Open_Session_Response** object. Dit object gaat ons vertellen of de host een session met ons heeft geopend en zo ja wat het nummer van deze session is. Uiteindelijk kunnen en zullen we namelijk meerdere sessions tegelijkertijd geopend hebben. Wij ontvangen dus van de host bijvoorbeeld het volgende antwoord:

Open_session_response				
Open_session_response_tag	**Length field**	**Session status**	**Resource Identifier**	**Session_nb**
92	07	00	00010041	0001
Soort object	*7 bytes volgend*	*session opened*	*Id Resource Mngr*	*Sessienummer*

We krijgen dus de volgende reeks van 9 bytes terug van de host (hexadecimaal weergegeven): "**920700000100410001**"

Hierdoor weten we nu dat de session is geopend onder sessionnumber 0001. Nu zijn we dus in gesprek met de Application Manager!

Het veld Session status zou ook bijvoorbeeld F0 kunnen bevatten, in plaats van 00. In een tabel in EN50221 kunnen we de betekenis hiervan vinden: "*Session not opened, resource non existent*". Dan hebben we dus een probleem. Gelukkig is in een DVB ontvanger de Resource Manager altijd aanwezig en beschikbaar.

Als we het bovenstaande goed begrijpen dan is het dus een kwestie van goed in kaart brengen welke resources er beschikbaar zijn en hoe we hiermee kunnen en communiceren: welke objecten zijn er beschikbaar en volgens welk protocol wisselen we deze objecten uit?

In de nu volgende paragrafen gaan we de standaard beschikbare resources en nog wat optioneel aanwezige resources bekijken. Zoals eerder aangegeven beperken we ons uit oogmerk van overzichtelijkheid tot een globale omschrijving. In EN50221 is een gedetailleerde beschrijving van ieder object, veld en protocol te vinden. Zoals gezegd, er valt echt niet te ontsnappen aan EN50221!

3.5 De Resources (Command Interface)

3.5.1 De Resource Manager

De **Resource Manager** is een resource op de host en vormt de manager van alle beschikbare resources op zowel de host als de aanwezige module(s). Er is een protocol van objecten waarmee de resource manager met de applications en de resource providers kan communiceren over beschikbare resources.

Het eerste dat een application of resource provider doet wanneer de module in de host wordt geplaatst, of wanneer de host wordt aangezet, is een session openen met de Resource Manager. De Resource Manager stuurt een **Profile Enquiry** object aan alle applications en/of resource providers die deze beantwoorden met een **Profile Reply** waarin de beschikbare resources (indien aanwezig) worden opgegeven. Nadat de Resource Manager alle resources in kaart heeft gebracht stuurt hij een **Profile Change** object aan alle applications en resource providers.

Na het ontvangen van het Profile Change object kan een application of resource provider, indien gewenst, met een **Profile Enquiry** object bij de Resource Manager informeren naar alle beschikbare resources. De Resource Manager antwoordt met een **Profile Reply** met alle beschikbare resources.

Pas na het ontvangen van het eerste Profile Change object staat het een application of resource provider vrij om sessions te openen of te accepteren. De oorspronkelijke session met de Resource Manager blijft open om eventuele wijzigingen in de beschikbare resources te kunnen ontvangen van de Resource Manager met een Profile Change object. Als de application of resource provider zelf een wijziging in de beschikbare resources wil doorgeven stuurt hij een Profile Change object naar de Resource Manager. Deze beantwoordt met een Profile Enquiry, waarop de application of resource provider een nieuwe Profile Reply zendt met de gewijzigde resource lijst. Indien dit de resourcelist in de Resource Manager wijzigt, dan wordt een Profile Change gestuurd aan alle applications en resource providers. Deze kunnen dan met een Profile Enquiry weer informeren naar de nieuwe resourcelijst.

3.5.2 Application Information Resource

De **Application Information Resource** is een resource van de host, vergelijkbaar met de Resource Manager. Waar de Resource Manager de manager van de Resources is, is de Application Information Resource de manager van de applications.

Iedere application zal na de Profile Enquiry initialisatie fase een session openen naar de Application Information Resource in de host. De host stuurt vervolgens een **Application Info Inquiry** object naar de application, die deze beantwoordt met een **Application Info** object. In dit object vinden we informatie zoals Application type (b.v. “01” = Common Access) en de toplevel (hoogste) menukeuze van de application. Deze menukeuze wordt door de host naar eigen inzicht opgenomen ergens in haar eigen menustructuur.

De session wordt open gehouden, zodat de host op ieder moment een **Enter Menu** object kan sturen waarna de application direct een MMI (Man Machine Interface) session zal starten met het hoofdmenu van de application. Dit gebeurt natuurlijk wanneer een gebruiker in de host naar de menustructuur van de host gaat, en hier de toplevel menukeuze van de application kiest.

3.5.3 Conditional Access Support resource

De Conditional Access Support is een resource op de host om Conditional Access applicaties te ondersteunen. Alle CA applications openen een session naar deze resource zodra de Application Info Inquiry is voldaan. De host verzendt een **CA Info Inquiry** object naar de application en de application beantwoordt deze met een **CA Info** object met informatie over de CA System ID's (Provider ID's) die de application ondersteunt. De host weet hierdoor welke application welk CA system kan decoderen. De session blijft vervolgens geopend om de host in staat te stellen gebruik te maken van deze ondersteuning door middel van de onderstaande CA PMT en CA PMT Reply objecten.

Als een gebruiker een programma selecteert, dan verstuurt de host aan één of meerdere CA applications een **CA PMT object** met hierin informatie over het geselecteerde programma, zoals de PES streams waaruit het programma bestaat en aanwijzingen hoe de ECM's gevonden kunnen worden. Als een gebruiker meerdere programma's heeft gekozen wordt voor ieder programma een CA PMT object verzonden. Het CA PMT bevat alle -maar ook alleen maar de- **CA descriptors** voor het geselecteerde programma uit de Program Map Table (PMT) in de MPEG-2 transportstream (zie: 3.7 MPEG-2 transportstream (Transport Interface))

De applications antwoorden met een **CA PMT Reply** object, waarna de host de application kan selecteren die de descrambling zal uitvoeren voor het geselecteerde programma. In het CA PMT Reply object is hiervoor het veld **ca_pmt_cmd_id** aanwezig. Hiermee geeft de host aan welke actie van de application wordt verwacht, zoals bijvoorbeeld direct descramble of dat eerst nog een MMI (Man Machine Interface) session moet worden gestart. Bijvoorbeeld om eerst kijkrechten aan te schaffen.

3.5.4 Host Control en Information Resources

Naast de bovenstaande belangrijke resources staan er nog een aantal andere resources ter beschikking.

Zo geeft de **DVB Host Control** resource de module de mogelijkheid om tijdelijk de besturing van de host over te nemen. Met het **Tune** object worden overgeschakeld naar een andere service (programma). In het object bevinden zich de hiervoor benodigde parameters (Network ID, Original Network ID, Transport Stream ID en service_id). Met **Replace, Clear Replace** en **Ask Replace** kan tijdelijk worden overgeschakeld (bijvoorbeeld voor een reclameboodschap).

Bij de **Date-Time resource** kunnen met het **Date-Time Enquiry** object de datum en tijd van de host worden opgevraagd.

3.5.5 Man Machine Interface Resource

Deze resource stelt de module in staat om met de gebruiker van de host te communiceren. Het geeft de module de controle over de display en kan toetsaanslagen van de gebruiker ontvangen. Een toepassing hiervan is vanzelfsprekend de menustructuur van onze CAM.

Er kan op twee manieren met de MMI resource worden gecommuniceerd: **Low-level MMI** en **High-level MMI**. Met Low-level MMI heeft de module de absolute controle over de graphics van de display en kunnen toetsaanslagen van de gebruiker rechtstreeks worden ontvangen. Bij High-level MMI zijn een aantal objecten gedefinieerd, waarin menu's en lijsten kunnen worden gecommuniceerd. De host bepaalt in dit geval de look-and-feel van de display.

Een MMI session wordt bijvoorbeeld opgestart als het **Enter menu** object in de session met de Application Information Resource wordt ontvangen (zie 3.5.2 Application Information Resource). Een MMI session kan door de host en de module worden beëindigd door verzending van het **Close_MMI** object. Om menu's tussen applications te kunnen laten wisselen zonder het "flitsen" van het "onderliggende" programma, kan een **delay** parameter worden meegegeven.

Met het **Display control** object kunnen zowel de karakteristieken van de display worden opgevraagd als de gewenste modus worden ingesteld. Er zijn drie mogelijkheden: Bitmap graphics door de huidige uitzending, Bit map graphics in plaats van de huidige uitzending of character based High-level modus. Gezien de complexiteit van de Low-level modus en onze doelstelling beperken wij ons hier tot de eenvoudigste: de character based high-level modus.

Op het display control object antwoordt de host met een **Display reply** object. Voor onze high-level mode is in het reply object alleen de tabel met ondersteunde karakterset van enig belang en de bevestiging van de geselecteerde modus.

In High-level MMI kan met het **Text** object een blok tekst door de module naar de host worden verzonden om op de display te worden getoond. Het is mogelijk hier wat controlcodes in op te nemen om de tekst enigszins op te maken. Met het **Enq** object kan een vraag op de display worden gesteld. Het door de gebruiker ingegeven antwoord wordt door de host met een **Answ** object teruggestuurd. Hierin bevindt zich de parameter **answ_id**, die de waarde 00 heeft, als de gebruiker op cancel heeft gedrukt.

Met het **Menu** object kan een menu op het scherm worden gezet waaruit de gebruiker een keuze kan maken. In het object kunnen en menutitel, -subtitel en -ondertitel worden opgenomen, tezamen met een aantal keuzes. In deze High-level MMI mode bepaalt de host natuurlijk hoe dit menu wordt getoond en op welke wijze de gebruiker een keuze kan maken. De gemaakte keuze wordt met een Answ object retour gezonden. Hierin bevindt zich het veld **Choice_ref**, die de waarde 01 heeft voor de eerste keuze, de waarde 02 voor de tweede keuze, etc. Een waarde 00 geeft aan dat de gebruiker het menu zonder keuze heeft beëindigd (escape).

Het kan echter ook wenselijk zijn een lijst af te drukken, zonder dat hier een keuze uit hoeft of kan worden gemaakt. Dit is mogelijk met het List object, dat een bijna gelijke structuur heeft als het Menu object, maar dan zonder Choice_ref veld natuurlijk.

3.5.6 Overige resources

Daarnaast is er nog een aantal andere resources die we hier voor de volledigheid kort in beeld brengen. Als we deze in het kader van onze studie nodig blijken te hebben, zullen we ze later verder uitwerken.

Zo is er de **Low-speed communication class** die ons in staat stelt om bijvoorbeeld over een modem of kabel systeem te communiceren. Met de optionele **Authentication resource** kan een autorisatie worden geregeld, waardoor een module alleen bevoegd is bepaalde signalen te verwerken. Met de **EBU Teletext display resource** kan informatie via de teletekst functie van de host op de display worden afgedrukt. De **Smart Card Reader Resource class** kan zowel door door de host beschikbaar worden gesteld als door (een andere) module. Met de objecten **Smart Card Cmd** en **Smart Card Reply** kunnen beheer- en antwoord commando's worden uitgewisseld. **Smart Card Send** en **Smart Card Reply** verzenden en ontvangen gegevens van de smart card. Met de optionele **DVB EPG Future Event Support Class** wordt het mogelijk voor een EPG (Electronic Program Guide) applicatie in de host te communiceren met een CA applicatie. Zo kan de host in de EPG lijst van de komende programma's direct tonen of de kijker gerechtigd is het programma te zien, of dat hier bijvoorbeeld eerst een aparte handeling (zoals aankoop van kijkrechten) voor moet plaatsvinden.

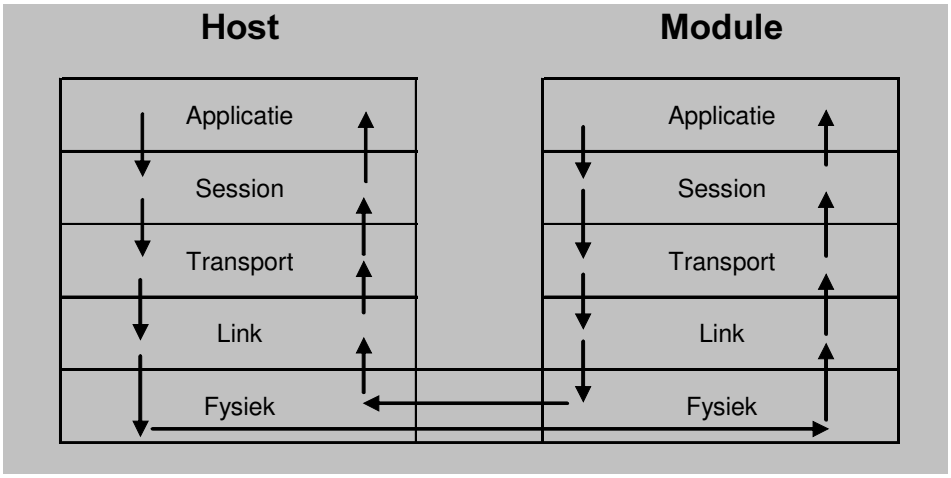
3.6 Communicatie tussen Host en Module

3.6.1 Het principe van layering

Je zou gezien de algemene beschrijving denken dat een application rechtstreeks “praat” met de resource. Een application zou dan een object (de reeks tekens) samenstellen en dit rechtstreeks zenden aan de resource. De resource zou de reeks tekens inlezen en reageren. Helaas is de werkelijkheid natuurlijk weer ingewikkelder. De communicatie moet namelijk uiteindelijk over een fysieke interface (de pc-card interface) naar de andere kant en hier komt een transportmechanisme om de hoek kijken, met componenten zoals buffers, registers, spanningswisselingen op pinnen, etc. Het is onder andere hierom dat tussen de application en de resource in zowel de host als de module een aantal “lagen”, de zogenaamde **layers,** zitten. Aan de verzendende kant praat iedere layer met de eronder liggende layer, die de boodschap weer doorgeeft aan de layer daaronder tot uiteindelijk de fysieke layer is bereikt. Hier wordt de data via de pinnen van de interface naar de andere kant “overgeseind”. Aan de andere kant wordt de boodschap weer door de fysieke layer aan de layer erboven doorgegeven, die deze boodschap weer aan de layer daarboven doorgeeft, tot deze bij de bestemming (de applicatie of de resource) aankomt. Zij die met het zogenaamde *OSI model* op de hoogte zijn, zullen geen moeite met dit concept hebben.

Dit veel toegepaste gelaagde transportmechanisme zorgt ervoor dat bijvoorbeeld een internet applicatie kan worden ontwikkeld zonder kennis van een specifieke netwerkkaart. Een specifieke laag, bijvoorbeeld een driver voor een netwerkkaart, kan worden ontwikkeld door mensen met de hiervoor benodigde specifieke kennis. De laag er bovenop, die bijvoorbeeld een transportconnectie opent met een andere computer op internet, kan worden ontwikkeld zonder specifieke kennis van netwerkkaarten en drivers hiervoor. Als je maar weet hoe je de driver moet aanroepen met de juiste parameters om de data over te dragen. Voordeel is ook dat bijvoorbeeld de onderste laag, de netwerkkaart, makkelijk kan worden vervangen door een andere kaart, met een andere driver, mits die maar op dezelfde wijze kan worden aangeroepen. De applicatie op het hogere niveau hoeft hiervoor dus niet te worden aangepast. Dit biedt dus de nodige flexibiliteit. Nog een ander voordeel is dat wanneer op een lager niveau reeds een transportconnectie met een andere computer is gemaakt, een andere applicatie die ook wil communiceren met een applicatie op die andere computer, er gebruik kan worden gemaakt van dezelfde transportconnectie. Er behoeft dan geen nieuwe transportconnectie te worden gemaakt.

In feite communiceert iedere layer met zijn soortgenoot aan de andere kant. Een layer accepteert een object van een bovenliggende layer, stopt deze objecten in zijn eigen object(en), voegt adressering toe voor de corresponderende layer aan de andere kant en draagt het object over aan de layer eronder. Ook voor de communicatie tussen de host en de module zijn (erg) globaal op deze wijze layers gedefinieerd:



3.6.2 De layers

De in de voorafgaande paragraaf zeer globaal benoemde layers worden nu verder uitgewerkt. Voor de Transport Stream Interface en de Command Interface zijn namelijk aparte layers gedefinieerd:

<table>
<tr><th>Transport Stream Interface</th><th colspan="4">Command Interface</th></tr>
<tr><td rowspan="2">Upper layers

(MPEG-2 specs ISO 13818)</td><td colspan="4">Application layer
Resources</td></tr>
<tr><td>User interface</td><td>Low speed communications</td><td>System</td><td>Optional extensions</td></tr>
<tr><td></td><td colspan="4">Session layer</td></tr>
<tr><td rowspan="2">Transport Layer
(MPEG-2 specs ISO 13818)</td><td colspan="4">Generic transport sublayer</td></tr>
<tr><td colspan="4">PC Card transport sublayer</td></tr>
<tr><td colspan="5">PC Card Link layer</td></tr>
<tr><td colspan="5">PC Card physical layer</td></tr>
</table>

1. De application verzoekt de session layer om een nieuwe session te openen met een resource. Nadat de session door de session layer is geopend, kan de application beginnen de te verzenden data in objecten te stoppen: (**Application Protocol Data Unit = APDU**).
2. De applicatie geeft de APDU's door aan de session layer.
3. De session layer stopt één of meer van deze APDU's in een **Session Protocol Data Unit (SPDU)** en draagt deze SPDU's over aan de Transport Layer.
4. De transport layer voegt één of meerdere SPDU's in één of meer **Transport Protocol Data Units (TPDU)** samen en draagt deze over aan de PC-card link layer.
5. Op de PC-card Link Layer wordt de TPDU's ingedeeld in **Link Protocol Data Units (LPDU)** die zijn afgestemd op de in de Physical Layer beschikbare buffer. Hierbij kunnen TPDU's van meerdere transport connections door elkaar worden verzonden. Een Link Connection wordt automatisch tijdens de initialisatie van host of module tot stand gebracht.
6. Op de Physical Layer wordt de data daadwerkelijk overgebracht naar het andere systeem. De bits worden door middel van wisseling van elektrische spanning op verschillende pinnen met een bepaalde snelheid gezet en gelezen door het andere systeem. De specificaties van dit niveau zijn dan ook fysiek van aard, zoals betekenis van pinnen, spanningshoogte, bitrate, etc. In ons geval betreft dit dus de specificaties van een PC-card (PCM CIA module).
7. In de host loopt deze informatiestroom weer naar boven via de Physical Layer, de Link Layer, de Session Layer naar de resource. Het antwoord volgt natuurlijk weer de omgekeerde weg.

Het spreekt bijna vanzelf dat op elke layer een protocol is benoemd volgens welke deze layer communiceert met de corresponderende layer aan de andere kant. Zo moeten bijvoorbeeld een session en een transport connection kunnen worden geopend en gesloten. Daar er op enig moment meerdere sessions en meerdere transport connections geopend kunnen zijn, zullen er identificaties moeten zijn om deze stromen "uit elkaar" te houden. Een session wordt daarom geïdentificeerd met een **Sessionnumber** en een transport connection met een **Transport Connection Identifier**.

Nu we deze problematiek globaal in kaart hebben, gaan we de objecten en de protocollen per layer nader bekijken. Voor ons doel, het ontwikkelen van software voor de CAM, is een goed en nauwkeurig begrip van deze communicatie natuurlijk van essentieel belang.

3.6.3 PC-card layers

De specificaties van deze onderste lagen komen overeen met de standaard definities voor een PC-Card interface (PCM-CIA module). Op het onderste niveau, de Physical Layer, worden zaken gedefinieerd als pinbezetting, spanning, data transfer rates, etc. In Appendix A van EN50221 vinden we de specificaties hiervan. Op deze Physical Layer worden de gegevens daadwerkelijk fysiek overgedragen met spanningswisselingen op de pinnen naar de ontvangende partij.

De PC-Card Link Layer heeft van het bovenliggende Transport Layer TPDU objecten ontvangen ter verzending en maakt hiervoor LPDU objecten, afgestemd op de op de Physical Layer beschikbare buffer. Een Link connection komt automatisch tot stand bij de initialisatie die plaatsvindt zodra er op fysiek niveau contact is gemaakt. Bij deze initialisatie wordt de Card Information Structure gelezen en wordt de card in de juiste modus geconfigureerd. Hierbij wordt ook de buffergrootte onderhandeld. Ieder LPDU object bestaat uit een header van twee bytes en een deel van een TPDU, het geheel niet groter dan de buffergrootte. De eerste byte is het nummer van de transport connection waartoe het deel van de TPDU behoort. Het tweede byte heeft in het most significant bit (linker bitje) een “1” als er nog meer fragmenten van de TPDU volgen en een “0” als dit niet zo is. De andere 7 bits zijn gereserveerd en hebben de waarde “0”. Elke LPDU heeft maar de gegevens van één (deel van een) TPDU aan boord. Als er meerdere transport connecties tegelijkertijd over de link Layer lopen, dan worden fragmenten van beide TPDU’s door elkaar verzonden om een eerlijke verdeling van de beschikbare bandbreedte te verkrijgen.

3.6.4 Generic Transport Layer

De functie van de Transport Layer is de door de hogere Session Layer aangeleverde Session Protocol Data Units te transporteren naar de Transport Layer aan de andere kant, en de van de andere kant ontvangen Session Protocol Data units aan de eigen Session Layer aan te leveren. De Transport layer is dus het transport mechanisme van een session.

Communicatie op Transport Layer nivo vindt plaats volgens uitwisseling van **Command Objects**. De host verzendt een Command Transport Protocol Data Unit **C_PTDU**. De module antwoordt met een Response Transport Protocol Data Unit **R_PTDU**. De module kan geen transport beginnen en moet wachten tot de host begint. Er zijn in totaal 11 soorten **Transport Layer objects**, waarvan sommigen alleen door de host kunnen worden verzonden, sommigen alleen door de module en sommigen door beiden.

Een C_PTDU van de host bevat slechts één Transport Protocol object. Een R_PTDU van de module kan één of twee Transport Protocol Objecten bevatten. Het enige of het tweede object in een R_PTDU van de module is altijd een status object (T_SB).

Iedere transport connection heeft een **transport connection identifier** van 1 byte. Omdat 0 is gereserveerd, kan de host over alle modules tegelijkertijd 255 connections geopend hebben. Volgens EN50221 specificaties minimaal 16 per module, maar het liefst 255 verdeeld over de in de host aanwezige modules. De identifier wordt bepaald door de host.

De Objects

Nu gaan we de 11 soorten objecten bekijken en het protocol volgens welke ze worden uitgewisseld.

Transport Layer objects	
1. Create_T_C	Deze creëert een nieuwe connectie en bevat de connection identifier. (Alleen door host).
2. C_T_C_Reply	Antwoord hierop van de module met de connection identifier
3. Delete_T_C	Wist een transport connectie en bevat als parameter de te wissen connectie. Host en module kunnen hem verzenden, maar module alleen als antwoord op een poll (*) of data van de host
4. D_T_C_Reply	Het antwoord hierop. Omdat dit soms niet aankomt heeft Delete_T_C een timeout. Als die is bereikt kunnen de acties worden genomen die normaal o.g.v. de reply worden genomen.
5. Request_T_C	Een verzoek aan de host om een nieuwe transport connectie. Wordt met een bestaand connectienummer verzonden en alleen als antwoord op een poll (*) of data van de host.
6. New_T_C	Antwoord op Request_T_C. Wordt dezelfde connectie verzonden als de request en bevat de nieuwe connection identifier. New_T_C wordt direct gevolgd door een Create_T_C om de nieuwe connection te maken.
7. T_C_Error	Wordt verzonden met daarin 1 byte met het errornummer. In deze versie wordt deze alleen als antwoord op een Request_T_C verzonden dat er geen connecties meer mogelijk zijn.
8. T_SB	Wordt als antwoord op alle objecten van de host verzonden, eraan vast geplakt of apart en geeft in één byte aan of de module nog data te verzenden heeft
9. T_RCV	Wordt verzonden door host als antwoord op een T_SB met het verzoek de data aan de host te sturen.
10. T_Data_more 11. T_Data_last	Deze bevatten de daadwerkelijke data van of naar de module. De module mag allen verzenden op verzoek met een T_RCV. T_data_more wordt gebruikt als een Protocol Data Unit van een hoger nivo moet worden gesplitst omdat het te groot is om in één object te worden verzonden (door externe beperking). T_data_last bevat het laatste fragment van een gesplitste PDU of het enige fragment van een PDU. Bij meerdere segmenten, moet ieder datapacket wachten op een aparte T_RCV.

(*) Pollen gebeurt met een lege "T_Data_last"

Het protocol

Als we de objecten goed bestuderen, dan ligt het protocol voor de uitwisseling van deze objecten min of meer voor de hand.

Als een host een transport connection wil starten naar een module (en alleen de host kan beginnen!) dan stuurt de host een Create_T_C object naar de module. De module antwoordt direct met een C_T_C_Reply. Als deze reply niet binnen een time-out periode wordt ontvangen dan is de transport connecion mislukt, en zal het transport connectionnumber opnieuw worden gebruikt, bijvoorbeeld om opnieuw een connectie te proberen op te bouwen met de module.

Na de C_T_C_reply kan de host data aan de module gaan verzenden met T_data_last (de host verstuurt altijd maar één Protocol Data Object). Is er geen data te verzenden, dan poll't de host de module of die data te verzenden heeft met een leeg T_data_last object. De module antwoordt de host met een T_sb object waarin met één byte wordt aangegeven of de module data te verzenden heeft. Is dit het geval, dan antwoordt de host met een T_RCV object waarop de module met T_data_more of T_data_last de data verzendt. Indien er meerdere data objecten te verzenden zijn, dan wordt de data verzonden met T_data_more. Het laatste pakket wordt verzonden met T_data_last. Voor iedere T_data_more of T_data_last moet de module wachten op een T_RCV van de host.

Indien een module een nieuwe transport connection wil starten dan kan het antwoord van de module ook een Request_T_C zijn waarop de host antwoord met een T_C_error als er geen connections meer beschikbaar zijn, of een New_T_C met het nieuwe connectionnumber. De module antwoordt hierop met een C_T_C_reply, en na het ontvangen van de T_RCV van de host kan de module de data over deze connectie weer gaan verzenden met T_data_more en T_Data_last.

Zowel de host als de module kan vervolgens de connection sluiten door het verzenden van een Delete_T_C object en na ontvangst van de bevestiging hiervan met een D_T_C_Reply object is de connection gesloten.

3.6.5 Session Layer

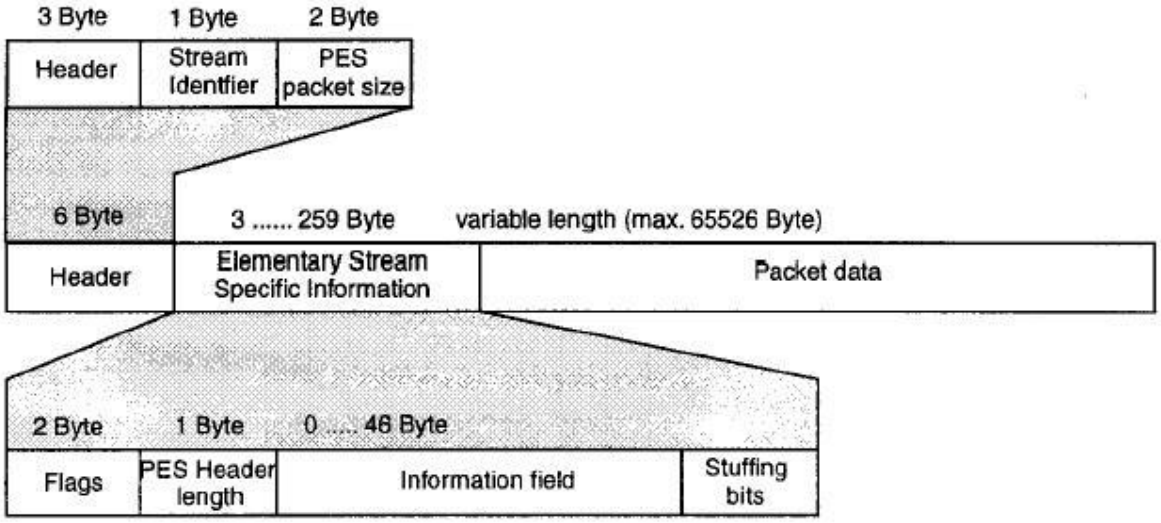
De session layer voorziet in het mechanisme waarmee applications gebruik kunnen maken van de resources. Een resource kan zich op de host of de module bevinden. Een module kan ook via de host gebruik maken van een resource op een andere module.

Sommige resources kunnen meer dan één sessie aan sommigen niet. Zo kan bijvoorbeeld de display wellicht slechts één session aan. Heeft de display windows, dan kunnen meer sessies worden geopend, anders niet. Dan wordt een Resource_busy reply object verzonden. De volgende objecten zijn gedefinieerd op de sessionlayer.

Open_session_request	Wordt verzonden door een applicatie over zijn transport connection met verzoek om gebruik van een resource. De host kan de resource direct beschikbaar stellen of een nieuwe transport connection maken naar een module die de resource beschikbaar heeft.
Open_session_response	Dit verzendt de host naar de applicatie om een sessienummer toe te kennen of een melding dat de resource niet beschikbaar is.
Create_session	Wordt door een Host verstuurd aan een module die een resource beschikbaar heeft om aan een session_request van een andere module te kunnen voldoen over een nieuwe transport connection.
Create_session_response	Is het antwoord van een resource provider in een module aan de host zodat de host aan de aanvragende module kan aangeven of de session kan worden geopend.
Close_session_request	Wordt door host of module verzonden om een session te beëindigen
Close_session_response	Wordt door host of module verzonden om de beëindiging van een session te bevestigen.
Session_number	Een Session_number gaat altijd vooraf aan een Session Protocol Data Unit (SPDU) die een Application Protocol Data Unit (Application Protocol Data Unit) bevat.
Session_nb(n, data)	De daadwerkelijke data, voorafgegaan door het session number.

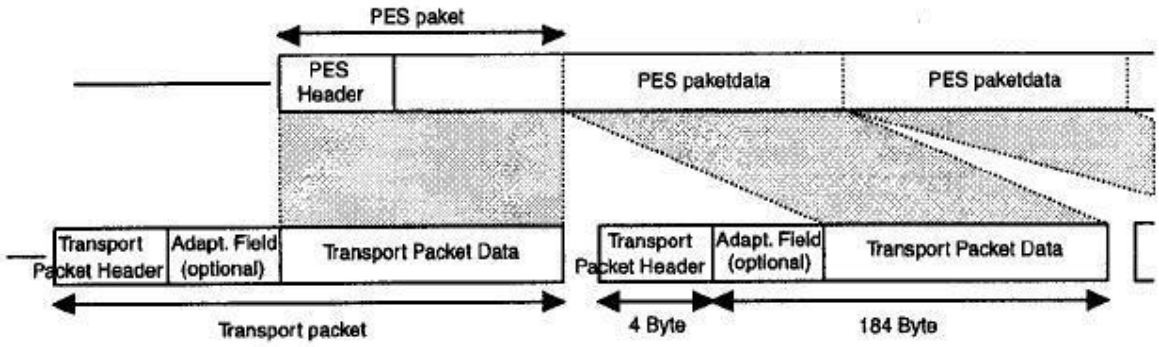
3.7 MPEG-2 transportstream (Transport Interface)

Zoals we hebben gezien communiceren de *applicaties* en de *resources* op de module over de *Command Interface*. De **MPEG-2 transportstream** komt echter over de *Transport Interface* de module binnen en verlaat deze ook weer via de transport interface. De MPEG-2 transportstream bestaat uit meerdere **Packetized Elementary Streams (PES),** en iedere PES bevat één elementaire stroom data, zoals beeld, geluid, ondertiteling, besturingsinformatie, etc. Een PES stroom zelf bestaat uit elkaar opvolgende packets met een (lange) variabele lengte. Elk packet begint met een "header" van 6 bytes. Het volgende dataveld bevat hoofdzakelijk stuurinformatie (Elementary Stream Specific Information) en is maximaal 259 bytes. Als laatste volgt de eigenlijke data met een variabele lengte van maximaal 65526 bytes. Het eerste veld, Flags, bevat statusbits voor bijvoorbeeld scrambling, copywright, etc.



Structuur van de PES-stroom.

De **MPEG-2 transport stream** is opgebouwd uit transport packets met een (korte) vaste lengte van 188 bytes. De **transport packet header** is 4 bytes lang. De overige 184 bytes zijn voor de daadwerkelijke data. De PES packets worden in stukjes gehakt en verdeeld over meerdere transport packets. In onderstaand figuur wordt de relatie weergegeven tussen de transportstream en de PES stream.



Samenhangende structuur van transportstroom en PES-stroom.

De PES header volgt altijd direct de Transport header tenzij een 'adaptation field' wordt meegezonden. In dat geval volgt de PES header direct daarna. Aan het einde van een PES packet kunnen een aantal 'stuffing bytes worden toegevoegd, die geen informatie bevatten.

Een PES wordt geïdentificeerd met een **PID** nummer, dus PES packets met hetzelfde PID nummer behoren tot dezelfde PES stroom. Hierdoor kan de ontvanger de verschillende PES stromen uit elkaar houden en selecteren. De transport packet header bestaat uit 32 bits (4 bytes). De PID vinden we hierin op bit 12 tot en met bit 24 (13 bits).

Verschillende PES stromen vormen samen een programma, geïdentificeerd met een programmanummer van 2 bytes. Voor informatie over programma's wordt de **Program Specific Information (PSI)** gebruikt. Er zijn in de PSI een aantal tabellen gedefinieerd, waarvan onderstaand de belangrijkste:

Table	Locatie	Inhoud
Program Association Table (PAT)	PID = 0	Voor ieder programmanummer in de transportstream de PID van de *Program Map Table* van het programma. (Daarnaast in programma 0 de PID voor de Network Information Table)
Program Map Table (PMT)	PID per PMT	Voor ieder programma in de transportstream de PID nummers en descriptors van het programma.
Network Information Table (NIT)	Programma 0 in PAT	Karakteristieken van het transmissienetwerk zoals frequentieband, centrale frequentie, kanaalbandbreedte, transponder, etc.
Conditional Access Table (CAT)	PID = 1	Geeft voor iedere CA provider de PID voor de EMM messages en eventuele parameters.

Een ontvanger haalt uit de Program Association Table (PID=0) de programma's die zich in de transportstream bevinden. In het geval dat de gebruiker programma x kiest, kan de ontvanger uit de Program Association Table bij programma(x) de PID halen voor de Program Map Table van dit programma. In deze Program Map Table vindt de ontvanger de PID's waaruit het programma bestaat (beeld, geluid, etc.) en de eventuele descriptors van het programma, zoals de CA-descriptor benodigd voor descrambling.

Een descriptor is een reeks tekens die informatie over het betreffende onderwerp bevat en we komen op verschillende plekken verschillende soorten descriptors tegen. De CA-descriptor bijvoorbeeld bevat de informatie die we nodig hebben om de ECM's en EMM's te vinden. De ECM's en EMM's worden namelijk met aparte PID nummers verzonden. We kunnen een CA descriptor op twee plaatsen tegenkomen:

- In de **Program Map Table** geeft de CA-descriptor de PID van de **ECM**'s van een programma;
- In de **Conditional Access Table** geeft de CA-descriptor de PID van de **EMM**'s van een provider.

Naast de door MPEG-2 gedefinieerde tabellen zin er in DVB nog een aantal extra tabellen gedefinieerd zoals bijvoorbeeld de **Bouquet Associaton Table (BAT)** en de **Event Information Table (EIT)**. Voor iedere tabel is een PID gedefinieerd waarmee deze wordt verzonden, waarbij sommige tabellen met dezelfde PID worden verzonden. Het onderscheid tussen de verschillende tabellen wordt gemaakt met een **Table_id**. Geen van de tabellen mag scrambled worden verzonden, op de EIT (Event Information Table) na, maar de data mag wel scrambled worden verzonden. Er vindt dan meestal 'stuffing' plaats om de overgang tussen gescrambelde en niet gescrambelde data te laten plaatsvinden.

Eén van onze uitdagingen zal zijn de ECM's te filteren en de hierin verstopte controlwords te vinden. De structuur van ECM's en EMM's is in ISO13818-1 slechts globaal voorgeschreven in een algemeen PES packet format:

Syntax	Aantal bits	Mnemonic
Packet_start_code_prefix	24	Bslbf
Stream_id	8	Uimsbf
PES_packet_length	16	Uimsbf
Databytes (aantal = PES_packet_length)	8	bslbf

- **packet_start_code_prefix**: The packet_start_code_prefix is een 24-bits code. Samen met de stream_id erna, vormt het een packet start code die het begin van een packet identificeert. De packet_start_code_prefix is de volgende reeks bits: '**0000 0000 0000 0000 0000 0001**' (0x000001).
- **Stream_id:** geeft de soort data weer in het packet en hiervoor is in ISO13818.1 een tabel gedefinieerd. Bij een ECM zal deze waarde '**1111 0000**' (0xF0) zijn en bij een EMM '**1111 0001**' (0xF1).
- **PES_packet_length**: Een 16 bits veld dat het aantal bytes in het PES packet aangeeft dat na dit veld volgt. (De waarde 0 geeft aan dat de PES Packet length niet is gespecificeerd en niet begrensd is. Dit kan alleen vookomen in PES packets waarvan de data een video elementary stream is die in Transport Stream packets is verzonden.)

Voor de databytes in een ECM/EMM wordt in ISO13818 geen structuur voorgeschreven. Dit zal waarschijnlijk het onderzoeksonderwerp worden bij de bestudering van de diverse coderingen zoals Irdeto, Seca, etc.

Deze informatie over de transportstream is afkomstig uit de ISO13818.1 specificaties. Er is hier slechts een globale beschrijving met enkele details opgenomen om de werking te doorgronden en een beeld te vormen van wat ons te wachten staat. Voor de details van alle packets, descriptors, tables, etc. kan ISO13818-1 worden geraadpleegd.

3.8 En nu verder…

We weten nu de plaats van onze emulator (de applicatie) in de cam (de module) en hoe deze zal communiceren met de ontvanger. We weten welke functies de emulator moet vervullen en ook bijvoorbeeld hoe we een menu hierin kunnen aanbrengen. Wat we echter nog niet weten is welke onderdelen van het gehele proces wij zelf zullen moeten ontwikkelen. De application, session en transport Layer zullen we ongetwijfeld moeten ontwerpen en programmeren. Maar moeten we bijvoorbeeld ook de link layer zelf ontwikkelen? En zo ja, hoe spreken we de physical Layer dan aan? En als de link layer wél reeds beschikbaar is, hoe spreken we hem dan aan?

De Command Interface mag dan met de geleerde informatie, op de bovenstaande vragen na, redelijk nauwkeurig in beeld zijn gebracht, maar hoe krijgen wij toegang tot de MPEG-2 transportstream over de Transport Interface? Uiteindelijk zullen we daar de ECM en EMM packtets uit moeten filteren, om de controlwords voor de descrambling te verkrijgen. En hoe ziet de data in de ECM's en de EMM's er uit? Met het uit de ECM's halen van het controlword zullen we waarschijnlijk het gebied betreden van de encrypties zoals Seca, Conax, Irdeto, etc. Ook zal uitgezocht moeten worden hoe we met de descrambler kunnen communiceren om het controlword aan te leveren, conform de algemene beschrijving.

Dit is min of meer de situatie waarin we ons nu bevinden, en beantwoording van bovenstaande vragen is onze doelstelling voor het volgende hoofdstuk. Gezien het feit dat de link layer en het aanspreken van een descrambler hardware afhankelijk zullen zijn, lijkt het een goed plan om ons nu eens te gaan verdiepen in een specifieke CAM. We gaan hierbij kijken of we de geleerde theorie in verband kunnen brengen met de fysieke componenten van deze CAM en op welke wijze wij deze componenten in onze emulator kunnen gaan gebruiken.

Literatuurverwijzingen:

- http://www.bjpace.com.cn/data/tec/tec-DVB/DVB%20BlueBooks%20Standards/Specifications%20and%20Standards/interfacing/dvb-ci/EN50221.PDF
- http://neuron2.net/library/mpeg2/iso138181.doc
- http://users.pandora.be/satelliet/mpegnorm.pdf

4. Matrix CAM

5. Programmeerkennis

6. Coderingen

7. Emulatiesoftware

8. Smartcards

Appendix 1: DVB Algemeen

(Overgenomen van : http://mccb.be.eu.org/leden/krbonne/sat-tv.belgie.html#D4)

4. DVB-S: Digital Video Broadcasting - Satellite

DVB-S is een systeem voor digitale uitzendingen van audio, video en data via satelliet. Het vormt eveneens de basis voor DVB-MS, de variant van DVB over LMDS/MVDS. Hier volgt een beschrijving over enkele belangrijkste elementen van digitale uitzendingen en DVB in het algemeen en DVB-S in het bijzonder.

4.1 Streams, kanalen en boeketten

Het eerste belangrijke verschil tussen analoge en digitale uitzendingen is dat het verband tussen een transponder en een TV-programma volledig verdwijnt. Bij analoge uitzendingen bestaat er een duidelijk één-op-één verband tussen een 'kanaal' (een transponder op een satelliet) en een TV-programma. Eén kanaal komt overeen met één programma en visa-versa. Bij digitale uitzendingen is dit helemaal niet meer van toepassing.

- Een satelliet-transponder kan één of meerdere 'transport-streams' bevatten.
- Een transport-stream kan één of meerdere TV-, radio- of data-kanalen bevatten.
- Een TV- of radio-programma bestaat uit een combinatie van een aantal audio- en/of video- en/of data-kanalen.
- Anderzijds worden verschillende programma's (mogelijk verspreid over verschillende transponders op één of meerdere satellieten) gebundeld tot één enkel boeket.

4.1.1. De transport-stream

Een 'transport-stream' is, zoals de naam het zegt, de 'transportlaag' van digitale TV. Het is een stroom van bits, uitgezonden door de satelliet, met een bepaalde snelheid, op een bepaalde frequentie en polariteit van een satelliet-transponder. Ze komt overeen met 'zoveel miljoen bits per seconde'.

Een ander verschil met analoge uitzendingen is dat een transport-stream NIET altijd de gehele transponder van de satelliet in beslag neemt. Soms kan een transport-stream slechts een deel van het spectrum van een transponder in beslag nemen, waardoor één transponder op een satelliet meerdere transport-streams kan bevatten. Soms bevindt zich op een transponder zowel een analoog TV-kanaal als een digitale transport-stream.

4.1.2 De 'PES': Packetised Elementary Stream

- Binnenin de transport-stream bevindt zich één of meerdere 'substreams'; de zogenaamde 'PES' (Packetised Elementary Stream).
- Elke PES bevat één enkele uitzending: een TV-videosignaal, een geluidskanaal, teletekstinformatie, ondertitelinginformatie, of 'pure data'.
- Elke PES kan een verschillend bitrate hebben naar gelang het type informatie dat in de sub-stream opgeslagen zit. Het spreekt voor zich dat een videosignaal meer informatie moet bevatten dan een stream die enkel ondertitelings-informatie bevat.
- Elke PES wordt geïdentificeerd aan de hand van een 'PID' (PES Id).

4.1.3 Het boeket

Een boeket is een groepering van programma's van een bepaalde aanbieder (bv. een betaal TV-aanbieder). Een boeket kan zelfs verspreid zijn over verschillende transponders van een satelliet of zelfs over meerdere satellieten op dezelfde positie.

4.2: SRs, FECs: de beschrijving van een transport-stream

Een transport-stream wordt bepaald door vier waarden:

1. Satelliet-positie (bv. 'Astra 1' op 19,2 graden oost).
2. Frequentie en polarisatie: (bv. 12,574 GHz, horizontale polarisatie)
3. SR (Symbol Rate): zie hieronder
4. FEC (Forward Error correction), zie hieronder.

4.2.1. SR: Symbol rate

Indien men een transportstream op 'technisch' niveau bekijkt, dan is dat eigenlijk een radio-draaggolf die een vast aantal keer per seconde van fase verandert. Elke faseverandering noemt men een 'symbool', en omdat men bij DVB-S gebruik maakt van QPSK-modulatie, vertegenwoordigt elk symbool 2 bits.

De symbol-rate is het aantal keer per seconde dat de transport-stream van fase verandert. Het bepaalt dus effectief de hoeveelheid informatie per seconde wordt verstuurd door de totale transport-stream.

Eén van de eigenschappen van radio-communicatie bepaalt dat de totale hoeveelheid van het radio-spectrum (de zg. bandbreedte) dat een radio-signaal heeft rechtstreeks verband houdt met het aantal maal dat het radiosignaal per second verandert.
Voor digitale uitzendingen op satelliet betekent dit dat de bandbreedte die een transport-stream nodig heeft recht evenredig is met de symbol-rate van de transportstream (want de SR is net de maat voor het aantal veranderingen per second van de draaggolf).

Veel gebruikte symbol-rates zijn 27500 of 22000 Ksymbols/s omdat de benodigde bandbreedte voor zo'n signaal net overeenkomen met wat beschikbaar is op één volledige satelliet-transponder van bepaalde satellieten. Deze SR komt tegen 2 bits per symbol overeen met 55,5 of 44 MBps aan 'pure' bitrate.

4.2.2 Fout-correctie: Solomon-Reed en FEC

Een radioverbinding is nooit perfect, en ook bij satelliet-verbindingen is het altijd mogelijk dat er fouten optreden tijdens het oversturen van een signaal.

Omdat een omroepsysteem slechts in één richting werkt (van satelliet/zender naar de ontvanger) wordt hier gebruik gemaakt van "FEC" (ofwel "Forward Error Correction"). Bij dit soort systeem wordt reeds bij het uitzenden van het radiosignaal extra informatie meegestuurd zodat -indien er fouten optreden bij het oversturen van de bits van de zender/satelliet naar de ontvanger- de ontvanger dit kan detecteren en indien mogelijk ook de fouten kan corrigeren. De error-correctie bits worden dus vooraf doorgestuurd, vandaar de naam "forward error correction".

Een transport-stream van een satelliet-verbinding wordt 'beschermd' door twee verschillende error correctie-technieken: de 'outer-coding' (meestal aangeduid met de naam 'Reed Salomon') en de 'inner-coding' (gewoon aangeduidt met 'FEC').

De eerste (Reed-Salomon) neemt een vast percentage van de transport-stream in beslag (8%) en kan niet worden gewijzigd. De tweede (FEC) is wel instelbaar en wordt aangeduid met een breuk. Een FEC van bv. '3/4' betekent dat er per 3 bits 'echte' gegevens er een 4de bit meegestuurd wordt voor error-correctie. Veel gebruikte FEC-waarden zijn 1/2, 3/4, 5/6 of 7/8.

4.2.3 Een voorbeeld: BVN
Bon, voorbeelden zijn altijd een stuk duidelijker dan saaie getallen.
Dit zijn de gegevens voor de transport-stream waarin de TV-zender 'BVN' wordt uitgezonden. (BVN = "Beste van Vlaanderen en Nederland", een zender uitgebaat door de VRT en de openbare omroepen uit Nederland)

Positie: Astra 1 (op 19,2 graden oost)
Transponder: 12,574 GHz, horizontale polarisatie
transport-stream: Symbol rate 22000 (Ksymbols/s), FEC 5/6

4.3 De PES: binnenin de transport-stream
Een transport-stream is echter maar de pure 'drager' van de binaire informatie. De eigenlijk informatie die men wenst te bekijken of te beluisteren bevindt zich 'binnenin' de transport-stream in de verschillende 'substreams' of (in het juiste vakjargon) de 'PES' (Packetised Elementry Stream) in de transport-stream.

Even herhalen wat reeds kort werd besproken in 4.1.2.

- Elke PES bevat één stroom met 'informatie'. Deze informatie kan verschillend zijn van aard. Bijvoorbeeld:
  - beeld (video)
  - geluid (mono, stereo, surround sound, ...),
  - teletext-informatie,
  - ondertitel-informatie,
  - 'pure data' (bv. internet-data).
- Elke PES wordt aangeduid via een nummer: de 'PID'.

Daarnaast bevat een transport-stream echter ook nog een hoeveelheid administratieve informatie:

- Benaming en parameters van de individuele substreams.
- Het beeldkanaal en het geluidskanaal die moeten worden gekoppeld voor het verkrijgen van een 'programma'.
- Informatie over andere transport-streams.
- 'Klok'-informatie, nodig om het geluid van een film synchroon te houden met het beeld: de PCR-stream.
- Informatie of een programma al dan niet geëncrypteerd is.
- De 'EPG' ('electronic program guide', de elektronische programmagids)

4.4 En uiteindelijk: het programma

4.4.1 Een TV-programma zoals we het nu kennen
De laatste stap die nog moet gebeuren is het opbouwen van een 'programma': datgene waar wij als kijker naar kijken. Dat verkrijgt men gewoon door verschillende substreams te combineren. Voor een TV-kanaal is dat minimaal één video-signaal samen met één geluidskanaal.

Verder moet ook worden vermeld dat er -onzichtbaar voor de gebruiker- altijd nog een 'PCR' (klok) signaal wordt meegestuurd. Dit is nodig om de verschillende datastromen synchroon met elkaar te laten lopen. Indien dit niet klopt, bv. waarbij het geluid voor of achter loopt t.o.v. het beeld, dan spreekt men over 'lip sync' problemen.

4.4.2 Een TV-programma zoals het kan zijn

Echter, één van de voordelen van digitale TV, is dat men in principe alles met alles kan combineren. Hoewel het samenvoegen van één videosignaal met één geluidssignaal de meest logische combinatie is, zijn er veel meer mogelijkheden. Enkele voorbeelden:

Een 'uitgebreid' TV-kanaal. Dit bevat naast de video-stream (het beeld) en de audio (het geluid) eveneens 'teletekst', ondertitels en de elektronische programma gids (EPG).
Soms bevat een TV-kanaal meerdere geluidskanalen: bv. verschillende talen (zoals EbS, Eurosport of Arte) of een combinatie van een 'gewoon' audio-kanaal met een 'Surround'-geluidskanaal.
Het is ook denkbaar om meerdere videokanalen samen te voegen met één enkel audiokanaal. (Bv. een sportwedstrijd bekeken vanuit verschillende camerastandpunten).
Een andere mogelijkheid: een 'onderwijs' TV-kanaal, bestaande uit beeld, geluid en een 'data'-kanaal waarlangs de slides van de cursus worden doorgestuurd.

4.4.3 Het programma: de gegevens

Al deze extra gegegevens (Audio-PID, Video-PID, enz.) bepalen samen met de gegevens van de transport-stream het TV-programma. Nemen we opnieuw het programma 'BVN' (zie 4.2.3), dan krijgen we de volgende gegevens:

De transport-stream:

- Astra 1 (op 19,2 graden oost),
- 12,574 GHz, horizontale polarisatie,
- SR 22000, FEC 5/6

De gegevens van het programma 'BVN' binnenin deze transport-stream:

- Video-PID 516
- Audio-PID 690

Appendix 2: Program Map Table (PMT) (ISO13818-1)

The Program Map Table provides the mappings between program numbers and the program elements that comprise them. A single instance of such a mapping is referred to as a "program definition." The program map table is the complete collection of all program definitions for a Transport Stream. This table shall be transmitted in packets, the PID values of which are selected by the encoder. More than one PID value may be used, if desired. The table may be segmented into one or more sections, before insertion into Transport Stream packets, with the following syntax. In each section the section number field shall be set to zero. Sections are identified by the program_number field.

Table 2-29 -- Transport Stream program map section

Syntax	No. of bits	Mnemonic
TS_program_map_section() {		
table_id	**8**	**uimsbf**
section_syntax_indicator	**1**	**bslbf**
'0'	**1**	**bslbf**
Reserved	**2**	**bslbf**
section_length	**12**	**uimsbf**
program_number	**16**	**uimsbf**
Reserved	**2**	**bslbf**
version_number	**5**	**uimsbf**
current_next_indicator	**1**	**bslbf**
section_number	**8**	**uimsbf**
last_section_number	**8**	**uimsbf**
Reserved	**3**	**bslbf**
PCR_PID	**13**	**uimsbf**
Reserved	**4**	**bslbf**
program_info_length	**12**	**uimsbf**
for (i=0; i<N; i++) {		
Descriptor()		
}		
for (i=0;i<N1;i++) {		
stream_type	**8**	**uimsbf**
Reserved	**3**	**bslbf**
elementary_PID	**13**	**uimsnf**
Reserved	**4**	**bslbf**
ES_info_length	**12**	**uimsbf**
for (i=0; i<N2; i++) {		
descriptor()		
}		
}		
CRC_32	**32**	**rpchof**
}		

table_id -- This is an 8 bit field, which in the case of a TS_program_map_section shall be always set to 0x02 as shown in table 2-27 on page 47 above.

section_syntax_indicator -- The section_syntax_indicator is a 1 bit field which shall be set to '1'.

section_length -- This is a 12 bit field, the first two bits of which shall be '00'. It specifies the number of bytes of the section starting immediately following the section_length field, and including the CRC. The value in this field shall not exceed 1021.

program_number -- program_number is a 16 bit field. It specifies the program to which the program_map_PID is applicable. One program definition shall be carried within only one TS_program_map_section. This implies that a program definition is never longer than 1016 bytes. See Informative Annex C for ways to deal with the cases when that length is not sufficient. The program_number may be used as a designation for a broadcast channel, for example. By describing the different program elements belonging to a program, data from different sources (e.g. sequential events) can be concatenated together to form a continuous set of streams using a program_number. For examples of applications refer to Annex C.

version_number -- This 5 bit field is the version number of the TS_program_map_section. The version number shall be incremented by 1 modulo 32 when a change in the information carried within the section occurs. Version number refers to the definition of a single program, and therefore to a single section. When the current_next_indicator is set to '1', then the version_number shall be that of the currently applicable TS_program_map_section. When the current_next_indicator is set to '0', then the version_number shall be that of the next applicable TS_program_map_section.

current_next_indicator -- A 1 bit field, which when set to '1' indicates that the TS_program_map_section sent is currently applicable. When the bit is set to '0', it indicates that the TS_program_map_section sent is not yet applicable and shall be the next TS_program_map_section to become valid.

section_number -- The value of this 8 bit field shall be always 0x00.

last_section_number -- The value of this 8 bit field shall be always 0x00.

PCR_PID -- This is a 13 bit field indicating the PID of the Transport Stream packets which shall contain the PCR (Program Clock Reference) fields valid for the program specified by program_number. If no PCR is associated with a program definition for private streams then this field shall take the value of 0x1FFF. Refer to the semantic definition of PCR in 2.4.3.5 on page 25 for restrictions on the choice of PCR_PID value.

program_info_length -- This is a 12 bit field, the first two bits of which shall be '00'. It specifies the number of bytes of the descriptors immediately following the program_info_length field.

stream_type -- This is an 8 bit field specifying the type of program element carried within the packets with the PID whose value is specified by the elementary_PID. The values of stream_type are specified in table 2-36 on page 64.

elementary_PID -- This is a 13 bit field specifying the PID of the Transport Stream packets which carry the associated program element.

ES_info_length -- This is a 12 bit field, the first two bits of which shall be '00'. It specifies the number of bytes of the descriptors of the associated program element immediately following the ES_info_length field.

CRC_32 -- This is a 32 bit field that contains the CRC value that gives a zero output of the registers in the decoder defined in Annex B after processing the entire Transport Stream program map section.

Appendix 3: Conditional Access Descriptor (ISO13818-1)

The conditional access descriptor is used to specify both system-wide conditional access management information such as EMMs and elementary stream-specific information such as ECMs. It may be used in both the TS_program_map_section and the program_stream_map. If any elementary stream is scrambled, a CA descriptor shall be present for the program containing that elementary stream. If any system-wide conditional access management information exists within a Transport Stream, a CA descriptor shall be present in the conditional access table.

When the CA descriptor is found in the TS_program_map_section (table_id = 0x02), the CA_PID points to packets containing program related access control information, such as ECMs. Its presence as program information indicates applicability to the entire program. In the same case, its presence as extended ES information indicates applicability to the associated program element. Provision is also made for private data.

When the CA descriptor is found in the CA_section (table_id = 0x01), the CA_PID points to packets containing system-wide and/or access control management information, such as EMMs.

The contents of the Transport Stream packets containing conditional access information are privately defined.

Table 2-52 -- Conditional access descriptor

Syntax	No. of bits	Mnemonic
CA_descriptor() {		
descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
CA_system_ID	**16**	**uimsbf**
reserved	**3**	**bslbf**
CA_PID	**13**	**uimsbf**
for (i=0; i<N; i++) {		
private_data_byte	**8**	**uimsbf**
}		
}		

CA_system_ID -- This is an 16 bit field indicating the type of CA system applicable for either the associated ECM and/or EMM streams. The coding of this is privately defined and is not specified by ITU-T | ISO/IEC.

CA_PID -- This is an 13 bit field indicating the PID of the Transport Stream packets which shall contain either ECM or EMM information for the CA systems as specified with the associated CA_system_ID. The contents (ECM or EMM) of the packets indicated by the CA_PID is determined from the context in which the CA_PID is found, i.e. a TS_program_map_section or the CA table in the Transport Stream, or the stream_id field in the Program Stream.

Appendix 4: Releasenotes

Versie 1.0 (3 oktober 2004)
Document hoofdstukindeling gemaakt en hoofdstukken 1.”Inleiding” en 2.”Een globaal overzicht” geschreven. Appendix 1 gekopieerd van internet en appendix 2 met de releasenotes gemaakt. In deze versie zitten nog een aantal aannames van mij, waarvan ik hoop dat deze door anderen bevestigd of tegengesproken worden. Kleun ik er compleet naast, hoor ik het ook graag. Verder is de smartcard is momenteel nog een wat onderbelicht onderwerp. Misschien moet ik als gevolg hiervan de hoofdstukindeling in de toekomst herzien. Ook kan gedurende de studie blijken dat een andere volgorde wellicht handiger is. Ik ga vooralsnog verder met hoofdstuk 3. “UCAS CAM” door, in afwachting van commentaar op deze versie. Dit commentaar zal ik verwerken in versie 1.1 en hoger. In versie 2.0 zal nieuwe informatie zijn opgenomen.

Versie 2.0 (26 oktober 2004)
Geen inhoudelijke feedback ontvangen. Hoofdstuk 2 is dus ongewijzigd gebleven. Alleen het woord substream in PES veranderd. Hoofdstuk 3 “De CAM: het model” geschreven en appendix 2 en 3 tussengevoegd. Eventueel commentaar zal ik in versie 2.1 bijwerken en ondertussen ga ik verder met hoofdstuk 4.