FOI-R--0701--SE
December 2002
ISSN 1650-1942
Metodrapport

David Steen

Hermes II, Prototypdesign

TOTALFÖRSVARETS FORSKNINGSINSTITUT

Ledningssystem
Box 1165
581 11 Linköping

FOI-R--0701--SE

December 2002

ISSN 1650-1942

Metodrapport

David Steen

Hermes II, Prototypdesign

Utgivare Totalförsvarets Forskningsinstitut – FOI Ledningssystem Box 1165 581 11 Linköping	**Rapportnummer, ISRN** FOI-R--0701--SE	**Klassificering** Metodrapport
	Forskningsområde 6. Telekrig	
	Månad, år December 2002	**Projektnummer** E70170
	Verksamhetsgren 5. Uppdragsfinansierad verksamhet	
	Delområde 61 Telekrigföring med EM-vapen och skydd	
Författare/redaktör David Steen	**Projektledare** Jan Arnsby	
	Godkänd av	
	Uppdragsgivare/kundbeteckning	
	Tekniskt och/eller vetenskapligt ansvarig	

Rapportens titel

Hermes II, Prototypdesign

Sammanfattning (högst 200 ord)

Hermes II (HEteRogen Meddelandedistribution för EW-System) är tänkt att knyta samman enheterna i ett distribuerat signalspanings- och störsystem, med hjälp av de befintliga kommunikationssystem som finns tillgängliga vid varje tillfälle, och på så vis skapa ett nätverk som tillhandahåller den kommunikationskapacitet som är nödvändig mellan de olika enheterna. Rapporten behandlar två grundläggande problem: Distribution av information i nätverket, och upprätthållande av nätverket. Flooding pekas ut som ett lämpligt distributionsprotokoll, men ingen konkret lösning på hur nätverket kan upprätthållas har hittats. Dessutom presenteras en arkitektur för systemet, och en prototyp som utvecklats inom denna arkitektur.

Nyckelord

nbf, routing, mobilt nätverk, adhoc, flooding

Övriga bibliografiska uppgifter	**Språk** Svenska
ISSN 1650-1942	**Antal sidor:** 192 s.
Distribution enligt missiv	**Pris: Enligt prislista**

Issuing organization FOI – Swedish Defence Research Agency Command and Control Systems P.O. Box 1165 SE-581 11 Linköping	**Report number, ISRN** FOI-R--0701--SE	**Report type** Methodology report
	Programme Areas 6. Electronic Warfare	
	Month year December 2002	**Project no.** E70170
	General Research Areas 5. Commissioned Research	
	Subcategories 61 Electronic Warfare including Electromagnetic Weapons and Protection	
Author/s (editor/s) David Steen	**Project manager** Jan Arnsby	
	Approved by	
	Sponsoring agency	
	Scientifically and technically responsible	
Report title (In translation) Hermes II, Prototype design		

Abstract (not more than 200 words)

Hermes II (HEteRogeneous Message distribution for EW-Systems) aims to connect the units in a distributed electronic warfare system, using the in-place communication systems that are available at any instant, and thus creating a network that provides the communication capacity that the units require. The report considers two problems: The distribution of information in the network, and the maintenance of the connectivity of the network. Flooding is found to be a suitable information distribution protocol. No solution is found to the problem of maintaining the connectivity, but some of the points that need to be considered are discussed. Additionally a software architecture for Hermes II, and a prototype built on this architecture is presented.

Keywords routing, mobile network, adhoc, flooding	
Further bibliographic information	**Language** Swedish
ISSN 1650-1942	**Pages** 192 p.
	Price acc. to pricelist

Innehållsförteckning

1 Inledning

1.1 Bakgrund

För att kunna uppnå en konsistent och detaljerad omvärldsuppfattning på stridsfältet krävs det att det finns en digital infrastruktur som möjliggör effektiv och robust kommunikation mellan enheter. Många kommunikationssätt finns redan tillgängliga, i form av radio(länk), lokala nätverk, telefonledningar, etc. För att kunna kommunicera mellan alla enheter på det sättet som krävs för ett nätverksbaserat försvar, måste dessa kommunikationskanaler knytas samman i ett nätverk så att rätt information finns på rätt plats vid rätt tidpunkt. Hermes (HEteRogent MEddelandeSystem) är ett sådant system.

Omfattande forskning har gjorts på heterogena fasta nätverk, samt homogena mobila ad-hoc nätverk, men kombinationen heterogena mobila ad-hoc nätverk har inte behandlats. Det finns därför en del obesvarade fundamentala frågor när det gäller att bygga ett sådant nätverk. I [1] beskrivs ett antal frågeställningar som är relevanta för det system som skall byggas. Det arbete som beskrivs i denna rapport bygger vidare på en tidigare prototyp som använde ett nätverk av mobiltelefoner. Denna tidigare prototyp fungerade delvis, men flera problem upptäcktes, bl.a. är det svårt att ansluta nya noder till nätverket, dessutom är den inte uppdelad i moduler, vilket gör det svårt att ansluta nya kommunikationssätt. Den nya prototypdesign som beskrivs i denna rapport strävar efter att lösa de problem som upptäckts i den tidigare prototypen, samt att vara flexibel för framtida förändringar.

1.2 Rapportöversikt

Rapporten beskriver arbetet som lagts ned på att ta fram en plattform för utvärdering av Hermesprototyper. En inledande analysdel, där de frågeställningar som är relevanta för plattformen och för en enkel prototyp behandlas, följs av en designdel där det framtagna systemet beskrivs. Sist finns en beskrivning av själva implementationen, i form av källkoder för hela systemet.

De två huvudfrågor som har behandlats är upprätthållandet av nätverket, och distribuering av information i nätverket.

2 Systemanalys

2.1 Inledning

För att lösa många av de problem som identifierats är det nödvändigt att kunna experimentera med olika lösningsförslag för att se hur väl de fungerar i ett verkligt nätverk. Denna systemanalys kommer att fokusera på de frågeställningar som måste lösas för att kunna bygga en plattform för prototyputveckling. Dessutom ges förslag på hur en första prototyp lämpligen kan se ut.

2.2 Krav på prototypplattformen

En prototypplattform har till uppgift att underlätta prototyputveckling. Dessutom måste den vara flexibel så att de erfarenheter som görs under prototyputprovningen enkelt kan resultera i en förändrad prototyp.

För att hantera kravet på att det skall vara enkelt att bygga en prototyp är det lämpligt att modularisera systemet, samt definiera ett antal gränssnitt mellan modulerna. Detta gör att det blir enkelt att bygga en prototyp av en modul, eftersom man endast behöver koncentrera sig på den modul som skall byggas och implementera de gränssnitt som modulen måste stödja enligt plattformen. Dessutom underlättas testning av delsystem genom att modulerna kan testas separat.

Modulariseringen gör även att det är lättare att förändra prototypen, eftersom det förenklar modifiering av ett delsystem. För att ytterligare underlätta förändringar är det lämpligt att i plattformen endast specificera ett minimalt antal gränssnitt, samt att göra dessa så små som möjligt.

Ovanstående egenskaper är även önskvärda för ett skarpt system, eftersom det måste underhållas, och nya kommunikationssätt kommer att byggas in. Designen av prototypplattformen är därför inget bortkastat arbete, utan arkitekturen är tänkt att gradvis få fastare form och resultera i ett skarpt system. Frågor som prestanda och robusthet kommer därför att beaktas senare när mer grundläggande frågor är lösta.

2.2.1 Kommunikationsmedia

Arkitekturens bör konstrueras för att underlätta anslutandet av nya kommunikationsmedia. De kommunikationssätt som är lämpliga att börja med är:

- Mobiltelefon (modem point-to-point)
- Fast telefon (modem point-to-point)
- TCP/IP

Att de är lämpliga beror dels på att de är enkla att testa i en labmiljö, ingen svårtillgänglig hårdvara krävs, dels att de är exempel på tre distinkta kommunikationssätt som kan tänkas förekomma i ett verkligt nätverk.

Det är vanligt med små LAN, där trafiken vanligtvis går över ett snabbt och säkert IP-nätverk. Dessa LAN kan sedan tänkas kopplas samman med betydligt långsammare och osäkra (i den meningen att kommunikationen kan brytas när som helst) länkar i form av tråd(lös) kommunikation mellan enheter.

2.2.2 Användare (nodtyper)

Systemet kommer att användas av olika användare som delas in i olika klasser beroende på vilken funktion de har, dock har ingen sådan klassificering gjorts för denna prototyp. Alla användare har samma egenskaper och rättigheter i nätverket. Senare när ett antal olika applikationer har definierats är det sannolikt lämpligt att klassificera olika användare för att på förhand veta vilken information som önskas av varje användare och vilken information den kan tillhandahålla. De olika nodtyperna kommer då att ha olika egenskaper. För att hantera olika egenskaper har attribut införts. En nod kan ha ett antal attribut som beskriver dess egenskaper.

2.2.3 Attribut

En viktig del av den information som kommer att skickas i nätverket är olika egenskaper hos de enheter till vilka noderna hör. T.ex. geografisk position och typ av enhet. Dessa attribut hos noderna kan hanteras av applikationen och behöver i så fall inte vara en del av Hermes. Dock är attribut av en så grundläggande karaktär att alla noder kommer att ha vissa attribut. T.ex. så har alla noder en identitet. För att inte behöva implementera attributhantering i varje applikation som använder Hermes, är det därför bättre att lägga attributhanteringen nere i nätverkshanteringen. Detta upplägg gör det även möjligt att på ett standardiserat sätt prenumerera på attribut hos någon annan nod. T.ex. kan en ledningsenhet prenumerera på ett antal attribut hos de underlydande enheterna, som t.ex. geografisk position, och beredskapsläge, och har då hela tiden en uppdaterad bild av förbandet.

Man kan tänka sig att följande attribut är intressanta för alla typer av noder:

- Nodidentitet, t.ex. namnet på enheten.
- Nodtyp, t.ex. enhetstyp.
- Kommunikationsmedia, d.v.s. vilka olika kommunikationssätt som finns i denna nod.
- Geografisk position.

Det finns också ett antal metaattribut, d.v.s. egenskaper som beskriver attributen. T.ex. så följande metaattribut viktiga:

- Uppdateringsfrekvens, hur ofta detta attribut uppdateras från källan, t.ex. var 10:e minut.
- Ändringsbart, kan denna nod modifiera detta attribut? Ett exempel på ett attribut som kan ändras är en störtabell. Ett exempel på ett attribut som inte kan ändras är nodidentiteten.
- Prenumeranter, de noder som prenumererar på detta attribut. Detta är nödvändigt om källnoden är ansvarig för att uppdatera attributen. Om prenume-

ranterna själva håller reda på när och varifrån detta attribut ska uppdateras, och då används metaattributet källnod istället.

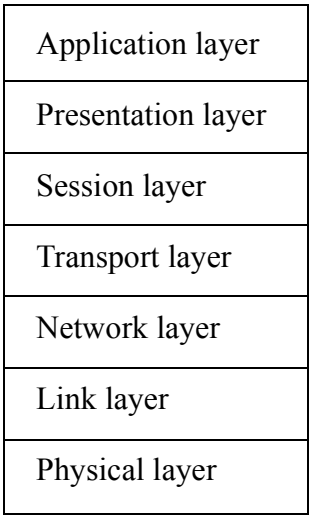
2.3 Problembeskrivning

HERMESsystemets uppgift är att distribuera information i ett nätverk bestående av ett antal olika noder (användare) som är anslutna med varandra genom kommunikationslänkar. Vidare är kommunikationslänkarna av olika typ, t.ex. mobiltelefon, eller TCP/IP.

2.3.1 OSImodellen

För att bestämma vilken modularisering som är lämplig och vilka gränssnitt som behövs så är det nödvändigt att till viss del undersöka vilken funktionalitet som systemet skall ha. En standardmodell för modularisering av nätverksprotokoll, OSI (Open systems interconnection) har tagits fram av ISO (International Standards Organization), se Fig. 2-1. Den består av ett antal olika skikt, där varje skikt har en viss uppgift [2]. Varje skikt kommunicerar endast med de skikt som det angränsar till.

Fig. 2-1. OSI ISO modellen



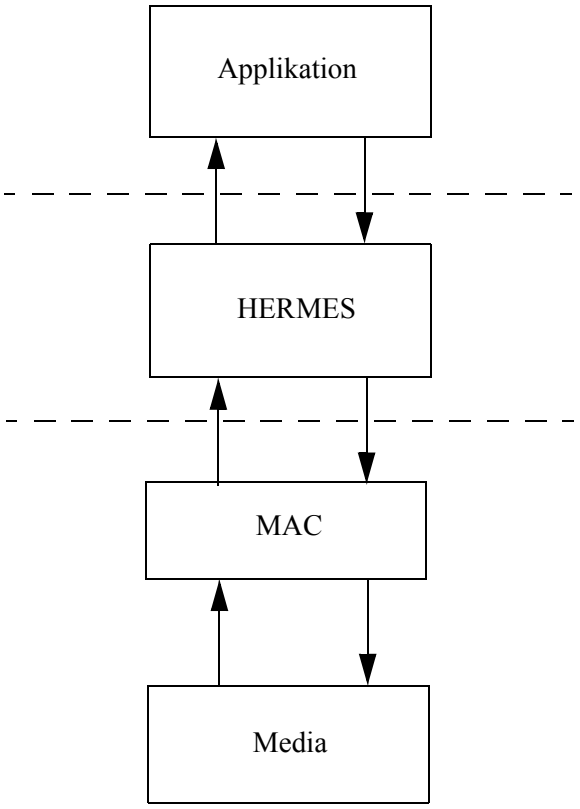
OSImodellen ger även en bra bild av de problem som måste lösas för att bygga ett nätverk. Skikten och deras funktionalitet beskrivs kortfattat nedan:

- *Application layer (AL)* är det skiktet som applikationen pratar med. Det implementerar högnivåprotokoll för olika applikationer, t.ex. FTP implementeras. Dessutom brukar kryptering ske på denna nivå, t.ex. SSH.
- *Presentation layer (PL)* handhar syntaxtransformering, t.ex. om Unix pratar med en Windows, så kan CR-LF behöva översättas.
- *Session layer (SL)* sköter dialoghantering och synkronisering. T.ex. om TCP används som är en uppkopplingsorienterad anslutning, så handhar SL själva initieringen och nedkopplingen av en TCP-kanal.
- *Transport layer (TL)* sköter felkontroll, fragmentering, och flödeskontroll.

- *Network layer (NL)* sköter routing, adressering, och uppkoppling.
- *Link layer (LL)* hanterar en datalänk; datatransparans, fel- och pakethantering, samt mediaaccess (MAC).
- *Physical layer (PL)* är det mekaniska och elektriska specifikationerna för nätverket.

I denna rapport är utgångspunkten att allt upp till MACprotokollet redan finns tillgängligt, vilket är sant för de kommunikationsmedia som är aktuella för tillfället. HERMESsystemet befinner sig således mellan MACskiktet och applikationen, se Fig. 2-2.

Fig. 2-2. HERMESsystemets gränssnitt.



Problemet med den tidigare prototypen är framför allt att den är låst till ett ringnätverk av mobiltelefoner. Detta medför att man inte kan koppla in nya noder i ett redan upprättat nätverk. Dessutom skickas information bara åt ett håll i nätverket vilket resulterar i att delar av nätverket blir onåbara om kommunikationen förloras i en länk, eller om en nod slås ut.

Denna rapport fokuserar på två huvudproblem som måste lösas innan systemet blir användbart;

- Distribuering av information, och
- upprätthållande av nätverket.

Dessa problem undersöks närmare i de följande två avsnitten.

2.3.2 Distribuering av information

För att två noder skall kunna kommunicera med varandra krävs det att informationen som skickas från en nod kommer fram till den avsedda mottagaren. Denna distribution av information sker på olika nivåer beroende på vilken information det är frågan om. På en låg nivå distribueras information genom att paket skickas från någon källnod till någon destinationsnod. På en högre nivå sker informationsdistribution genom att en applikation pratar med en eller flera andra applikationer. Den applikation som hittills använts är en meddelandeapplikation i vilken en nod kan skicka ett meddelande till en annan nod. Förutom denna applikation kommer endast informationsdistribution på en lägre nivå att behandlas i denna rapport.

Man kan urskilja ett antal typer av information som skall distribueras i nätverket;

- Administrativ information, som innehåller detaljer om nätverket som sådant.
- Nod till nod meddelande, som innehåller ett meddelande som endast destinationsnoden bedöms vara intresserad av.
- Allmän information, dvs information som alla eller åtminstone flera noder kan tänkas vara intresserad av.

Några saker att beakta vid distribution av information:

- Push och/eller pull, d.v.s. ska mottagaren begära information (pull), eller skall sändaren på eget bevåg sända data (push).
- Mellanlagring, skall paket sparas i mellanliggande noder? Hur länge?
- Routing, till vilka noder skall ett paket skickas för att nå fram till mottagaren?

Nätverksstrukturen har mycket gemensamt med ett MANET (Mobile Ad-hoc NETwork) [1]. Att det är ett mobilt nätverk betyder att upprättade länkar ofta bryts och att felsannolikheten i överföringarna är relativt hög. Med Ad-hoc menas att nätverket automatiskt kan hantera omstruktureringen som är nödvändig när enheterna rör sig och kommunikationsmöjligheterna förändras. Det pågår en intensiv forskning på bl.a. informationsdistribution i MANET. En gemensam nämnare för de flesta routingprotokoll som konstruerats för MANET är att Flooding används som en grundkomponent [3]. Anledningen till detta är framförallt att Flooding inte kräver någon topologisk information om nätverket, och är det mest robusta protokollet. Om det finns en väg från sändaren till mottagaren, så kommer Flooding garanterat att hitta den. Enda nackdelen med Floodingprotokollet är att det använder mycket bandbredd.

Med bakgrund av detta så framstår Flooding som ett naturligt val för det första routingprotokollet att implementera. Om det senare är önskvärt att implementera något mer bandbreddssnålt routingprotokoll är det lämpligt att göra en ny undersökning då, eftersom utvecklingen går ganska snabbt på detta område. I [1] påpekas att det kan vara önskvärt att spara all information som passerar genom en nod, för att minska trafiken i nätverket och för att göra nätverket mer robust. Denna lösning kan vara speciellt bra i kombination med Flooding, eftersom Flooding sprider information bättre än andra routingprotokoll, och därigenom gör nätverket ännu mer robust. Dessutom kan detta sätt att spara allt som går

genom noden motverka Floodingprotokollets egenskap att kräva mycket bandbredd. Med denna strategi så går det även snabbare för en nyansluten nod att bli uppdaterad med information som finns spridd i nätverket. T.ex. kan en nod som ansluter snabbt få information om andra kända enheters läge.

Till att börja med får den sändande noden ta ansvar för att den mottagande noden får ett meddelande. D.v.s. Cachning sker endast hos den sändande noden, och nätverket behöver inte innehålla någon felkorrigering.

Push och/eller pull kommer att bestämmas av applikationerna, och ingen vidare utredning om detta kommer att göras. Först när olika applikationer som använder systemet är definierade är det meningsfullt att ta ställning till vilken teknik som skall användas. Dock kan det nämnas att push-teknik kan vara olämplig i ett system där det är stor sannolikhet att mottagaren blir onåbar, antingen genom att kommunikationen bryts tillfälligt, eller att noden blir förstörd. De paket som då pushas ut i nätverket, kommer bara att belasta kapaciteten på nätverket och försämra prestandan för de noder som fortfarande är anslutna.

2.3.2.1 Flooding

Flooding har identifierats som det mest lämpliga protokollet att implementera i en första prototyp. I detta avsnitt beskrivs flooding översiktligt.

Floodingprotokollet är mycket enkelt och kan sammanfattas i dessa två punkter:

1. Notera alla paket som passerat noden
2. Skicka vidare paket som inte passerat noden tidigare till alla grannar (utom till den grannen från vilken paketet kom ifrån).

De två specialfallen är sändarnoden och mottagarnoden. Sändarnoden tar inte emot något paket utan skickar bara ut paketet till alla sina grannar. Mottagarnoden skickar inte paketet vidare utan skickar upp den information som paketet innehåller till den användare som är ansluten till noden.

I Fig. 2-3. visas ett exempel på hur ett paket skickas från nod A till nod C i ett litet nätverk. Det är uppenbart att paketet kommer att nå fram till mottagaren om det finns någon väg till den. Det är även uppenbart att paketet skickas betydligt många fler gånger än vad som är nödvändigt för att få fram det till mottagaren.

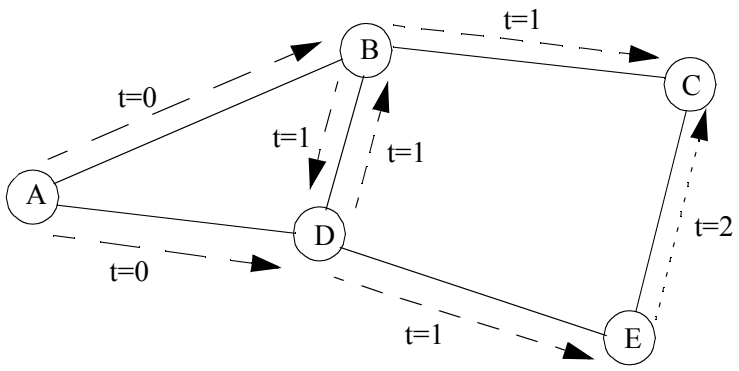


Fig. 2-3. Exempel på Flooding för paket som skickas från A till C.

För att paket inte skall skickas runt i evighet i nätverket så måste noderna hålla reda på vilka paket som redan passerat noden. Det finns två olika sätt att göra detta på; antingen kan varje paket förses med en unik identitet och varje nod underhåller en lista på vilka paket som passerat, eller kan varje paket innehålla en lista på de noder det passerat. Fördelen med att låta noden hålla reda på passerade paket är att paketen inte växer i storlek, som de gör om varje nod ska haka på sin identitet. Nackdelen är att noderna behöver mer minne för att hålla listan med passerade paket. Dessutom är det nödvändigt att definiera en tid under vilken paketidentiteterna sparas i listan, annars kommer listan att växa mot oändligheten. Det är enklare och robustare att låta paketen hålla reda på noderna eftersom man undviker det svåra problemet att välja hur länge paketidentiteterna ska lagras. Vilken metod som är lämplig att använda beror dels på hur stort nätverket är och dels på hur kritisk bandbredden är. Ett stort nätverk ger potentiellt långa vägar (många noder måste passeras) vilket gör att paketen kan bli väldigt stora om nodidentiteterna sparas i paketet, därför är det lämpligt att låta noderna hålla reda på paketen. Ett litet nätverk har inte detta problem, därför är det lämpligt att låta paketen hålla reda på noderna. Om bandbredden är kritisk så bör man låta noderna hålla reda på paketen eftersom det går åt mer bandbredd när paketen växer. Dessutom kommer paket i vissa fall att skickas om fler gånger när paketen håller reda på noderna, eftersom samma paket som når en nod genom olika vägar ses som olika paket och därför kommer att skickas om.

Eftersom de nätverk som uppstår när man kopplar samman några telekrigkomponenter blir relativt små, kommer metoden att låta paketen hålla reda på noderna att användas. Om det visar sig att bandbreddsproblem uppstår kan man senare gå över till metoden att låta noderna hålla reda på paketen, och kanske implementera ett annat bandbreddssnålt protokoll ovanpå Flooding.

2.3.3 Upprätthållande av nätverket

Att upprätthålla nätverket, betyder att se till att det alltid, när så är möjligt, finns en väg genom nätverket från en godtycklig nod till varje annan nod. D.v.s. att alla enheter kan prata med varandra. Eftersom länkarna kan störas ut, eller avbrytas på annat sätt kan det hända att nätverket faller sönder i två eller flera delnät som inte kan kommunicera med varandra. Detta är naturligtvis önskvärt

att undvika, och nätverket bör därför förses med en mekanism som kan läka ihop delnäten.

Tyvärr kan inte forskning inom MANET tillämpas utan modifiering när det gäller upprätthållandet av nätverket, eftersom de flesta använder den utgångspunkten att nätverket är byggt på ett broadcastmedium. Om nätverket faller sönder i sådan topologi kan man inte göra något för att reparera det, och upprätthållandet av nätverket är trivialt. När ett nätverk består av punkt-till-punktförbindelser kan ett nät som sönderfallit i två delar repareras genom att en nod i varje del upprättar en förbindelse, men det är oklart vilka noder som skall upprätta förbindelsen, samt om och hur det är möjligt att detektera att ett nätverk har sönderfallit i flera delnät.

Upprätthållande av nätverket sker på en relativt låg nivå i systemet, typiskt från network layer och neråt. Det finns två övergripande sätt att kommunicera i ett nätverk. Dels i uppkopplat läge (normally connected mode) och dels i nedkopplat läge (normally disconnected mode). I uppkopplat läge är alla länkar i systemet uppkopplade hela tiden, om en länk är nerkopplad så anses den inte vara delaktig i nätverket. En förutsättning för att kunna ha ett nätverk i uppkopplat läge är att de länkar som används i nätverket kan känna av att de är anslutna till nätverket. När nätverket arbetar i nedkopplat läge vet inte noderna om de har kontakt med nätverket eller inte förrän kommunikationen inleds. Därefter är det inte heller känt om nätverkskommunikationen fungerar förrän ett svar fås på den kommunikation som har inletts. Om nätverket opererar i uppkopplat eller nedkopplat läge, bestäms oftast av vilket kommunikationsmedia som används, och därför kommer vissa länkar i ett Hermessystem att operera i uppkopplat läge och vissa i nedkopplat läge. Ett medium som t.ex. en mobiltelefon kan tänkas operera i båda lägena, dock bara ett åt gången. Om mobiltelefonen används i nedkopplat läge, så görs en uppringning till mottagaren varje gång information skall överföras. När överföringen är klar bryts förbindelsen. I uppkopplat läge skapas en fast förbindelse mellan två mobiltelefoner, som bibehålls under lång tid. När information skall skickas är länken redan öppen och informationen skickas genast vidare till den nod som finns i andra änden av förbindelsen. Uppkopplat läge har uppenbarligen den fördelen att kommunikationen kan startas genast, samt att det är känt redan när informationen skickas att länken till mottagaren fungerar. I fallet med mobiltelefoner, så är en uppenbar nackdel att man bara kan skicka information till den nod som finns i andra änden av länken. Om informationen är avsedd för någon annan nod, måste informationen skickas vidare till rätt mottagare. Ett annat problem med nedkopplat läge är att information om hur uppkopplingen mot den nya noden skall ske måste anges för varje informationspaket som skall skickas. I fallet med mobiltelefoner, måste typiskt telefonnumret anges varje gång någon information skall skickas. I ett uppkopplat nätverk behöver denna information bara anges när nätverket startas. Eftersom både uppkopplade- och nerkopplade länkar troligtvis kommer att samexistera måste båda varianterna hanteras, och lämpligtvis bör systemet ta hänsyn till deras respektive för- och nackdelar när topologin väljs. Det är dock oklart exakt hur en lämplig topologi väljs utgående ifrån en mängd länkar med olika egenskaper.

För att hitta en optimal topologi, måste man bland annat veta hur mycket trafik det förekommer mellan de olika noderna. Detta kan skötas automatiskt genom att man samlar in statistik över belastningen, och omorganiserar nätet dynamiskt.

Ett annat sätt är att användarna manuellt får ange en sannolik belastning, eller vilka noder som behöver kunna prata med varandra på ett snabbt sätt. Man kan urskilja två olika prestandamått på kommunikationen mellan två noder; dels bandbredd, dvs hur mycket data per tidsenhet som kan överföras, dels fördröjning, dvs hur lång tid det tar för ett meddelande överföras från källnoden till destinationsnoden. Ingen automatisk lösning på hur nätverkets topologi skall underhållas har hittats. Tills vidare får det skötas för hand av användarna, men för att få ett effektivt system måste det kunna hantera sönderfall av nätverket.

De frågor som måste beaktas är åtminstone

- anslutning av en ny nod,
- avsiktlig borttagning av en nod,
- oavsiktligt bortfall av en nod, och
- bortfall av en länk.

Det finns två sätt att minska de negativa effekterna av bortfall av noder och länkar; dels kan en robust topologi väljas så att bortfall av enstaka noder och/eller länkar inte betyder att nätverket sönderfaller, dels kan nätverket ha självläkande egenskaper. Även om en robust topologi väljs är det fördelaktigt att ha en läkningsmekanism i systemet, utom i det fallet när varje nod har kontakt med alla andra noder.

2.3.3.1 Robust topologi

För att göra nätet så robust som möjligt är det önskvärt att ingen nod blir onåbar när en länk bryts, dvs man vill ha så många kopplingar som möjligt från varje nod till resten av nätverket. Detta skulle innebära att alla kommunikationslänkar används vilket gör att det blir svårt eller omöjligt att ansluta nya noder. Om ett broadcastmedium används finns inte denna begränsning eftersom det alltid är möjligt att sända till en nod så länge den befinner sig inom räckhåll. Detta pekar på att det är lämpligast att använda ett punkt-till-punktmedium i nedkopplat läge när det är möjligt, om inte uppkopplingsfördröjningen är oacceptabel. Detta resulterar i ett maximalt antal potentiella länkar, men då är det oklart hur flooding skall implementeras. Om flooding implementeras strikt, så skall ny information skickas ut på varje potentiell länk. I fallet med mobiltelefoner, så innebär detta att var och en av de telefoner som finns i alla andra noder rings upp i tur och ordning. Ett sådant förfaringssätt blir väldigt långsamt om många mobiltelefoner finns i nätet, vilket antyder att det är bättre att använda punkt-till-punktmedia i uppkopplat läge. Sammanfattningsvis är det oklart hur en robust topologi skall byggas upp och underhållas.

2.3.3.2 Självläkande topologi

För att nätverket skall kunna självläka topologin måste det först och främst vara möjligt att detektera att nätverket har sönderfallit i två eller flera delar. Ett tecken på att nätet kan ha sönderfallit i två delnät är att nod A inte får något svar på ett paket som skickas till nod B. Om frånvaron av svar beror på ett trasigt nätverk så skulle detta kunna lösas genom att A försöker upprätta en länk till B. Om A inte har något kommunikationssätt som kan upprätta en direktförbindelse med B måste nätverket repareras på något annat sätt. Detta tyder på att det är lämpligt

att varje nod känner till grannarna till de noder som de kommunicerar med, detta för att utöka möjligheterna till självläkning. Det bästa är naturligtvis om alla noder har all information om alla andra noders kommunikationsmedia. Eftersom nya media relativt sällan ansluts till en nod är det inte omöjligt att underhålla en sådan databas i varje nod, men även i detta fall är det inte klart hur självläkning kan genomföras automatiskt, eftersom information om vilka punkt-till-punktmedier är kopplade till varandra, dessutom är det i allmänhet okänt vilka noder som ligger inom räckvidden för nodernas broadcastmedia. Om denna kunskap finns så kan varje nod inse att nätverket har sönderfallit och också beräkna någon lämplig åtgärd. Att hela tiden distribuera information om varje länks status till varje nod i nätverket kräver stor bandbredd av nätverket och är knappast användbart utom i väldigt små nätverk. Ett ytterligare problem är att om förändringarna i nätverket sker alltför snabbt så kommer noderna att innehålla inaktuell information. Dessutom kan information naturligtvis inte distribueras till alla noder i ett nätverk om det har sönderfallit i flera delar.

3 Designbeskrivning

3.1 Inledning

Systemdesignen utgår ifrån de krav som framkom under systemanalysen. Dessutom eftersträvas enkelhet och minimalism i systemet. Det är ingen mening med att designa och implementera detaljer som ändå kommer att ändras när systemet börjar utprovas. Därför tas endast den funktionalitet som bedöms nödvändig för att en första prototyp av systemet skall fungera med i designen.

3.2 Utvecklingsverktyg och teknologival

Ett krav är att systemet ska kunna köras på Microsoft Windows 2000, inga krav på att använda andra operativsystem har ställts, därför har ingen större hänsyn tagits till portabilitet när utvecklingsmiljö och implementationsteknologi valts. För att utveckla de moduler som systemet består av har COM (Component Object Model) valts som ramverk. COM är Microsofts standard för komponentutveckling och Windows har bra stöd för COM, dessutom stödjs det av de allra flesta utvecklingsverktyg.

Borland Delphi 6.0 har valts som utvecklingsverktyg, huvudsakligen för att det använts tidigare i projektet. Dock gör valet av COM som teknologi att det är fullt möjligt att utveckla moduler med något annat verktyg, som t.ex. Microsoft Visual Studio, utan att det blir några problem att koppla samman modulerna.

3.3 Skiktindelning

För att organisera systemet delas det in i ett antal skikt, liknande OSImodellen, se avsnitt 2.3.1. Hermes har delats in i fyra skikt, se Fig. 3-1. Regelskiktet innehåller regler och data som är specifika för varje nod, d.v.s. attributhanteringen finns i detta skikt. Topologiskiktet hanterar upprätthållandet av nätverket, se avsnitt 2.3.3, och distributionskiktet hanterar datadistributionen, se avsnitt 2.3.2. Topologiskiktet och distributionsskiktet utgör tillsammans motorskiktet.

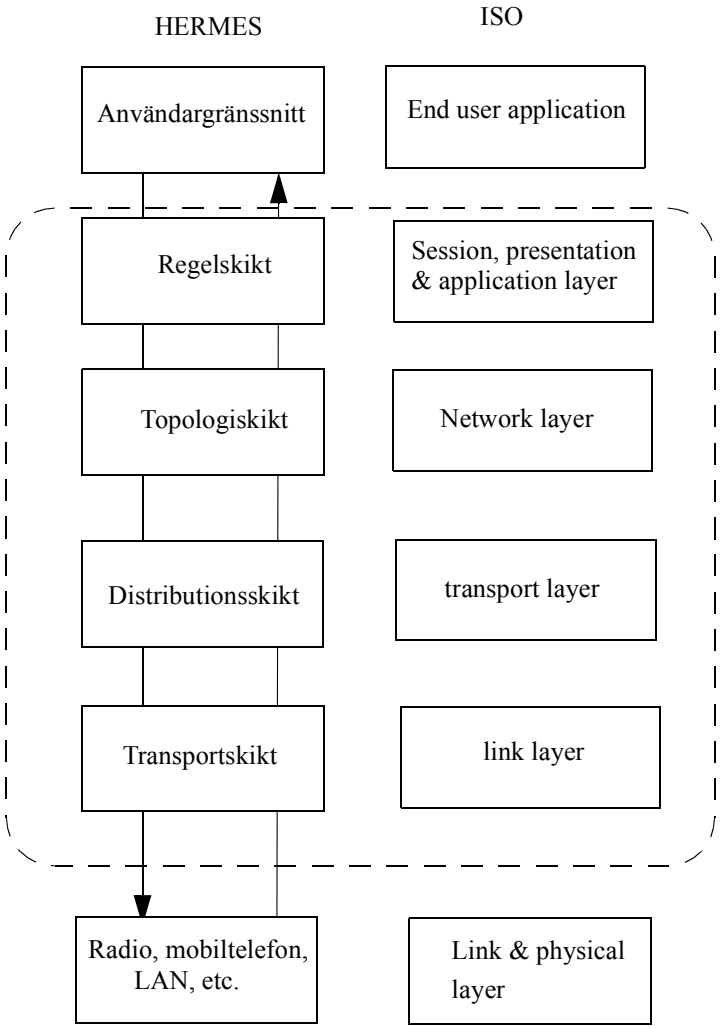


Fig. 3-1. Skiktindelning för Hermes i relation till OSI-modellen

3.3.1 Användargränssnitt

I ett första läge är användargränssnittet ett grafiskt gränssnitt mot en operatör. Gränssnittet presenteras på en vanlig datorskärm under Windows, och åskådliggör funktionaliteten i regelskiktet.

I nästa steg kan man tänka sig att autonoma, eller automatiska enheter ansluts till nätverket. Dessa kommer då att styras av särskilda kontrollmeddelanden. Gränssnittet måste då anpassas dels för den autonoma enheten, så att kontrollmeddelandena kan tolkas av enheten. Dels för operatörerna, så att kontrollmeddelanden kan skickas på ett enkelt sätt, utan att svårbegripliga kontrollkoder måste skrivas in för hand.

Användargränssnittslagret har följande funktionalitet:

- Presentera information för användaren,
- visa mottagna meddelanden,

- skicka meddelande,
- koppla upp media,
- koppla ner media,
- ändra egna attribut, och
- prenumerera på attribut hos andra användare.

När användargränssnittet startas visas först inloggningsformuläret, se Fig. 3-2. För att komma vidare skrivs nodens identitet in, och knappen *Login* klickas. När detta görs skapas ett regellager som kopplas till denna instans av användargränssnittet. Om det finns en startad instans av motorlagret på datorn, kopplas regellagret till denna, annars startas en instans av motorlagret.

Fig. 3-2. Formuläret för inloggning.

När inloggningen är klar visas huvudformuläret, se Fig. 3-3. I huvudformuläret kan mottagna meddelanden läsas, och nodens attribut visas. Dessutom finns knappar för att skicka meddelanden, hantera attributprenumerationer, och hantera mediaanslutningar.

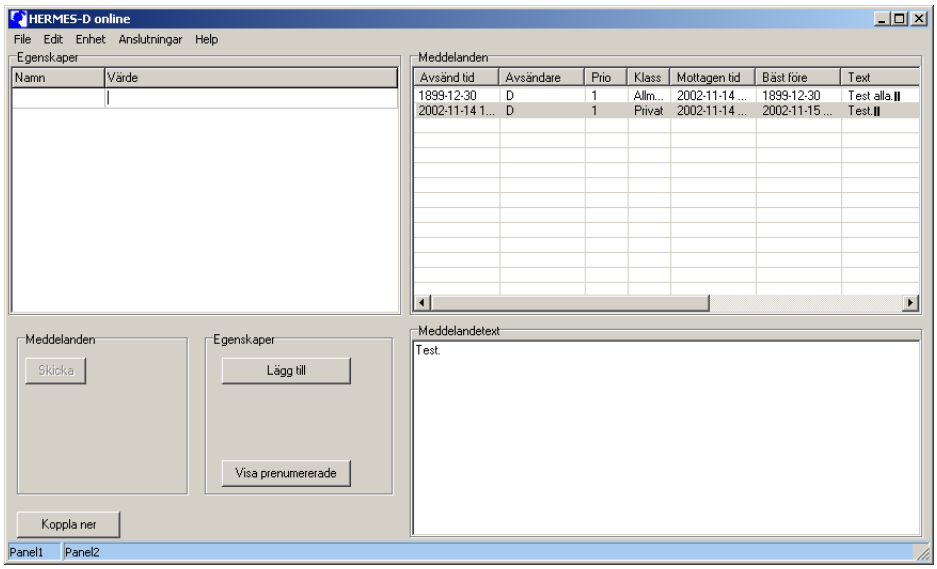
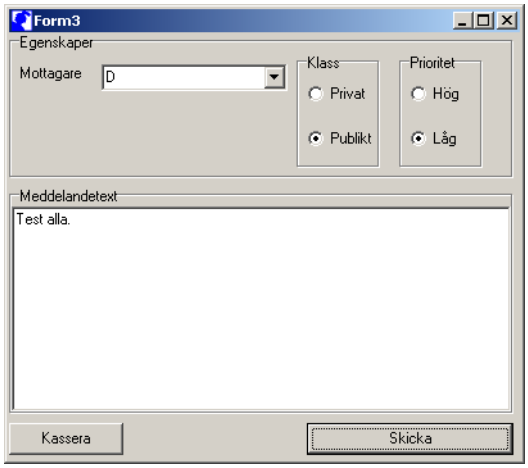


Fig. 3-3. Huvudformuläret innehåller egna attribut och mottagna meddelanden.

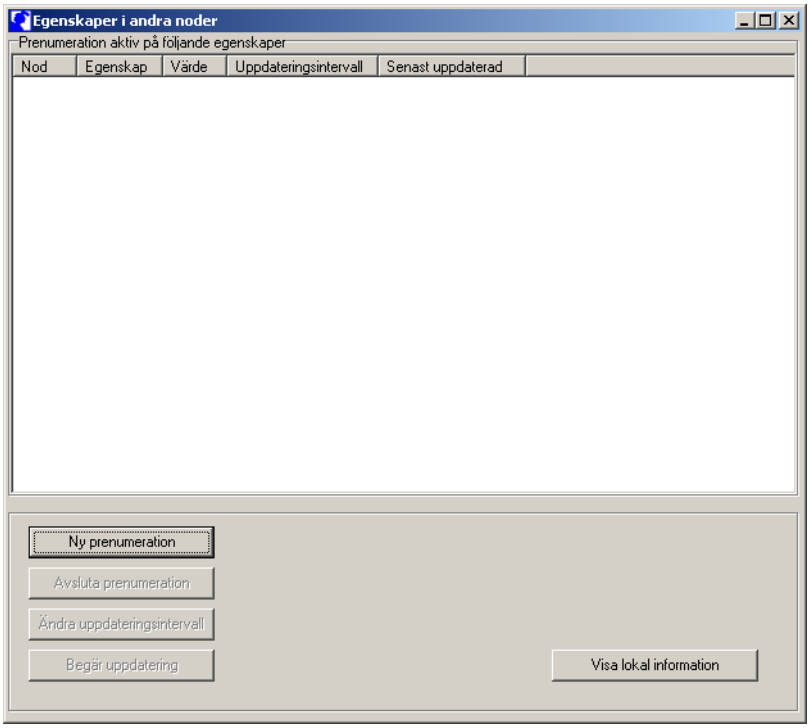
Om knappen *Skicka* klickas i huvudformuläret öppnas formuläret skicka meddelande, se Fig. 3-4. I detta formulär kan ett meddelande skapas. Meddelandetexten kan skrivas in och mottagare anges. Dessutom kan prioriteten på meddelandet anges, men det finns ännu inget stöd i resten av systemet för att hantera meddelanden av olika prioritet.

Fig. 3-4. Formulär för att skicka meddelande.



När knappen *Visa prenumererade* klickas öppnas attributformuläret, se Fig. 3-5. Härifrån kan man prenumerera på attribut i andra noder och avläsa vilka värden och uppdateringsintervall de har.

Fig. 3-5. Formulär för att prenumerera på attribut (egenskaper) hos andra noder.



Via anslutningsformuläret kan de kommunikationslänkar som noden har, kopplas upp mot andra noders kommunikationslänkar. Alla länkar tillsammans med den adress de är anslutna till visas tillsammans med den status som de har (upp-, eller nerkopplad), se Fig. 3-6..

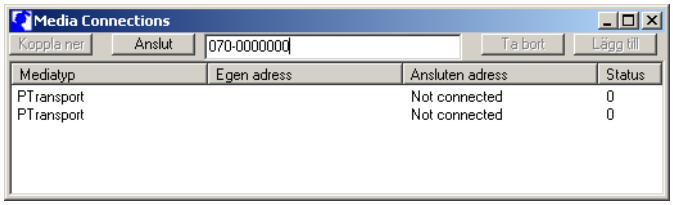


Fig. 3-6. Formulär för att hantera kommunikationslänkarna hos en nod.

3.3.2 Regelskiktet

Regelskiktet handhar information som beror på vilken typ den aktuella noden är av. Noden måste i utgångsläget endast känna sitt eget namn, samt ett sätt att kontakta noden till vilken den vill ansluta. Av denna nod får den nya noden information om alla noderna i nätverket (förutsatt att den kontaktade noden känner till alla enheter, annars fås bara en partiell uppfattning om nätverket).

Regellagret har följande funktionalitet:

- Innehåller uppgifter om vilken information som är intressant för denna nod.
- Innehåller information om speciella önskemål förknippade med denna nod.
- Innehåller information om nodens attribut och hur dessa skall distribueras.

3.3.2.1 Gränssnitt

Regelskiktets gränssnitt stödjer kommunikationen mot användargränssnittet, t.ex. att sända ett meddelande till en nod, och koppla upp mot nätverket. Följande funktionalitet finns i skiktet:

- Sänd meddelande till nodnamn,
- sänd meddelande publikt (till alla noder),
- ta emot meddelande,
- begära QoS (Quality of Service),
- utforska QoS,
- anslut till nätverket,
- koppla ner från nätverket,
- meddela om förlorad nätverkskontakt,
- meddela om förlorad länkkontakt,
- meddela om framgångsrikt levererat meddelande,
- meddela om misslyckad eller fördröjd leverans av meddelande,
- anslut media, och
- meddela maximal storlek på meddelande,

QoS innehåller information om bandbredden och fördröjningen för kommunikation mot en viss nod. Även tillförlitlighetsstatistik och liknande kan ingå i QoS.

Det fullständiga gränssnittet till regellagret beskrivet i MIDL (Microsoft Interface Definition Language):

```
[
  uuid(5836DADE-10C5-11D6-9231-00008637B362),
```

```
  version(1.0),
  helpstring("HermesLib Library")

]
library HermesLib
{

  importlib("stdole2.tlb");
  importlib("stdvcl40.dll");

  [
    uuid(8FA11B77-DB32-4DA1-B44D-0AB8673C17B6),
    version(1.0),
    helpstring("Dispatch interface for HermesControl Object"),
    dual,
    oleautomation
  ]
   interface IHermesControl: IDispatch
  {
    [
    id(0x00000001)
    ]
    HRESULT _stdcall MessageSend([in] VARIANT TheMessage );
    [
    id(0x00000002)
    ]
    HRESULT _stdcall MessageReceive([out] VARIANT * TheMessage, [out] long * Count
);
    [
    id(0x00000003)
    ]
    HRESULT _stdcall QOSRequest( void );
    [
    id(0x00000004)
    ]
    HRESULT _stdcall QOSExplore( void );
    [
    id(0x00000007)
    ]
    HRESULT _stdcall QueueGetTransmitSize( void );
    [
    id(0x00000008)
    ]
    HRESULT _stdcall QueueGetReceiveSize( void );
    [
    id(0x00000009)
    ]
    HRESULT _stdcall QueueGetTransmitMaxSize( void );
    [
    id(0x0000000A)
    ]
    HRESULT _stdcall QueueGetReceiveMaxSize( void );
    [
    id(0x0000000B)
    ]
    HRESULT _stdcall QueueSetTransmitMaxSize( void );
    [
    id(0x0000000C)
    ]
    HRESULT _stdcall QueueSetReceiveMaxSize( void );
    [
```

```
id(0x00000005)
]
HRESULT _stdcall AttributeAdd([in] BSTR Key, [in] BSTR Value );
[
id(0x00000006)
]
HRESULT _stdcall AttributeRemove([in] BSTR Key );
[
id(0x0000000D)
]
HRESULT _stdcall AttributeRead([in] BSTR Key, [out] BSTR * Value );
[
id(0x0000000E)
]
HRESULT _stdcall AttributeChange([in] BSTR Key, [in] BSTR Value );
[
id(0x0000000F)
]
HRESULT _stdcall AttributeSubscriptionStart([in] BSTR Key );
[
id(0x00000010)
]
HRESULT _stdcall AttributeSubscriptionEnd([in] BSTR Key );
[
id(0x00000012)
]
HRESULT _stdcall LogOut( void );
[
id(0x00000013)
]
HRESULT _stdcall NodeAdd([in] BSTR Node );
[
id(0x00000014)
]
HRESULT _stdcall NodeRemove([in] BSTR Node );
[
id(0x00000015)
]
HRESULT _stdcall NodeMediaAdd([in] BSTR Node, [in] BSTR MediaType, [in] BSTR
Address, [in] BSTR Port );
[
id(0x00000016)
]
HRESULT _stdcall NodeMediaRemove([in] BSTR Node, [in] BSTR MediaType, [in]
BSTR Address );
[
id(0x00000017)
]
HRESULT _stdcall NodeMediaQuery( void );
[
id(0x00000018)
]
HRESULT _stdcall NodeConnectedQuery( void );
[
id(0x0000001A)
]
HRESULT _stdcall LogIn([in] BSTR Node, [in] BSTR Password );
[
id(0x00000011)
]
HRESULT _stdcall MediaInventory([out] VARIANT * MediaList );
```

```
    [
    id(0x00000019)
    ]
    HRESULT _stdcall MediaStatus([in] BSTR MediaType, [in] BSTR MediaAddress, [out]
int * MediaStatus );
    [
    id(0x0000001B)
    ]
    HRESULT _stdcall MediaConnectionAddress([in] BSTR MediaType, [in] BSTR Medi-
aAddress, [out] BSTR * ConnectionAddress );
    [
    id(0x0000001C)
    ]
    HRESULT _stdcall MediaConnect([in] BSTR MediaType, [in] BSTR MediaAddress, [in]
BSTR ConnectionAddress );
    [
    id(0x0000001D)
    ]
    HRESULT _stdcall MediaDisconnect([in] BSTR MediaType, [in] BSTR MediaAddress );
  };

  [
    uuid(DD425DB2-0061-4CD5-AEC6-5A8B4C36CA92),
    version(1.0),
    helpstring("Events interface for HermesControl Object")
  ]
   dispinterface IHermesControlEvents
  {
    properties:
    methods:
    [
    id(0x00000001)
    ]
    HRESULT OnMessageReceived( void );
    [
    id(0x00000004)
    ]
    HRESULT OnMessageAck( void );
    [
    id(0x00000005)
    ]
    HRESULT OnMessageNak( void );
    [
    id(0x00000006)
    ]
    HRESULT OnLinkLost( void );
    [
    id(0x00000007)
    ]
    HRESULT OnContactLost( void );
    [
    id(0x00000008)
    ]
    HRESULT OnLinkRequest( void );
    [
    id(0x00000002)
    ]
    HRESULT OnNodeAdd( void );
    [
    id(0x00000003)
    ]
```

```
  HRESULT OnNodeDelete( void );
 };

 [
  uuid(2550F702-C223-4E52-9484-67C5B67D2565),
  version(1.0),
  helpstring("HermesControl Object")
 ]
 coclass HermesControl
 {
  [default] interface IHermesControl;
  [default, source] dispinterface IHermesControlEvents;
 };

};
```

3.3.3 Motorskiktet

I motorskiktet monteras paket samman till hela meddelanden, och meddelanden delas upp i paket. Motorskiktet har delats in i två skikt som handhar informationsdistribution respektive nätverkstopologi.

3.3.3.1 Distribution

Distributionslagret har följande funktionalitet:

- Hittar en lämplig väg genom nätverket
- Bestämmer om och hur data skall replikeras.
- Paketerar data.

3.3.3.2 Topologi

Topologilagret har följande funktionalitet:

- Bestämmer topologin på nätverket.
- Ser till att nätverket är sammanhängande.
- Sköter uppkoppling och nedkoppling av noder.

3.3.3.3 Gränssnitt

Följande funktionalitet finns i gränssnittet till motorlagret:

- Skicka meddelande privat,
- skicka meddelande publikt,
- ta emot meddelande,
- meddela inkommande meddelande,
- begära QoS,
- utforska QoS,
- anslut till nätverket,
- koppla ner från nätverket,
- meddela avbruten nätverkskommunikation,
- meddela avbruten länkkommunikation,
- sätt ny maxlängd på mottagarkön,

- efterfråga maxlängd på mottagarkön,
- sätt ny maxlängd på sändkön,
- efterfråga maxlängd på sändkön,
- efterfråga längd på sändkön,
- efterfråga längd på mottagarkön,
- efterfråga maxlängd på meddelande,
- sätt ny maxlängd på meddelande,
- anslut media.

Motorskiktet hanterar en mängt olika köer vilkas storlek bestämmer hur mycket data som kan buffras i noden. Dessutom finns en lista med alla regelmoduler och en lista med alla mediamoduler som är anslutna.

Det fullständiga gränssnittet till motorskiktet beskrivet i MIDL:

```
[
  uuid(1E4D5EFF-9BE5-425D-99F8-98A9C9660CC1),
  version(1.0),
  helpstring("PEngine Library")

]
library PEngine
{

  importlib("stdole2.tlb");
  importlib("stdvcl40.dll");

  [
    uuid(0AFD8346-5645-456D-AFD7-156E204C3CFA),
    version(1.0),
    helpstring("Dispatch interface for HermesSession Object"),
    dual,
    oleautomation
  ]
   interface IHermesSession: IDispatch
  {
    HRESULT _stdcall MediaStart([in] BSTR Config );
    [
    id(0x00000002)
    ]
    HRESULT _stdcall MediaStop( void );
    [
    id(0x00000003)
    ]
    HRESULT _stdcall MessageSend([in] VARIANT TheMessage );
    [
    id(0x00000004)
    ]
    HRESULT _stdcall MessageReceive([out] VARIANT * TheMessage, [out] long *
NoMessages );
    [
    id(0x00000006)
    ]
    HRESULT _stdcall QueueGetTransmitSize( void );
    [
    id(0x00000007)
    ]
    HRESULT _stdcall QueueGetReceiveSize( void );
```

```
[
id(0x00000008)
]
HRESULT _stdcall QueueGetTransmitMaxSize( void );
[
id(0x00000009)
]
HRESULT _stdcall QueueGetReceiveMaxSize( void );
[
id(0x0000000A)
]
HRESULT _stdcall QueueSetTransmitMaxSize( void );
[
id(0x0000000B)
]
HRESULT _stdcall QueueSetReceiveMaxSize( void );
[
id(0x00000005),
helpstring("Adds a new node. ")
]
HRESULT _stdcall NodeAdd([in] BSTR Node );
[
id(0x0000000C)
]
HRESULT _stdcall NodeRemove([in] BSTR Node );
[
id(0x0000000D)
]
HRESULT _stdcall NodeMediaAdd([in] BSTR Node, [in] BSTR MediaType, [in] BSTR
Address, [in] BSTR Port );
[
id(0x0000000E)
]
HRESULT _stdcall NodeMediaRemove([in] BSTR Node, [in] BSTR MediaType, [in]
BSTR Address );
[
id(0x0000000F)
]
HRESULT _stdcall Open([in] BSTR Node );
[
id(0x00000011)
]
HRESULT _stdcall Close( void );
[
id(0x00000001)
]
HRESULT _stdcall MediaStatus([in] BSTR MediaType, [in] BSTR MediaAdress, [out]
int * MediaStatus );
[
id(0x00000010)
]
HRESULT _stdcall MediaConnectionAddress([in] BSTR MediaType, [in] BSTR Medi-
aAddress, [out] BSTR * ConnectionAddress );
[
id(0x00000012)
]
HRESULT _stdcall MediaConnect([in] BSTR MediaType, [in] BSTR MediaAddress, [in]
BSTR ConnectionAddress );
[
id(0x00000013)
]
```

```
  HRESULT _stdcall MediaDisconnect([in] BSTR MediaType, [in] BSTR MediaAddress );
  [
  id(0x00000014)
  ]
  HRESULT _stdcall MediaInventory([out] VARIANT * MediaList );
};

[
  uuid(F40D541A-117B-47C7-99CE-47286DFB228E),
  version(1.0),
  helpstring("Events interface for HermesSession Object")
]
 dispinterface IHermesSessionEvents
{
  properties:
  methods:
  [
  id(0x00000001)
  ]
  HRESULT OnMessageAck( void );
  [
  id(0x00000002)
  ]
  HRESULT OnMessageNak( void );
  [
  id(0x00000003)
  ]
  HRESULT OnMessageReceived( void );
  [
  id(0x00000004)
  ]
  HRESULT OnContactLost( void );
  [
  id(0x00000005)
  ]
  HRESULT OnLinkLost( void );
 };

 [
  uuid(545A3789-5233-44C5-9EFF-3BEB0A23BD50),
  version(1.0),
  helpstring("HermesSession Object")
 ]
 coclass HermesSession
 {
  [default] interface IHermesSession;
  [default, source] dispinterface IHermesSessionEvents;
 };

};
```

3.3.4 Transportskiktet

Transportskiktet har följande funktionalitet:

- Är en adapter för mediamoduler.
- Sköter upp- och nerkoppling av länkar.
- Felkontroll av länkkommunikationen.

Det är önskvärt med ett enhetligt gränssnitt mot transportlagret, men eftersom flera olika media ska kunna användas, är det svårt att samla all nödvändig funktionalitet i ett enda gränssnitt. Därför används en konfigurationsfil till varje mediamodul som innehåller specifik konfigureringsinformation. Transportskiktet implementeras som ett gränssnitt som måste stödjas av varje mediamodul.

3.3.4.1 Gränssnitt

Transportskiktets gränssnitt stödjer följande funktionalitet:

- Koppla ner förbindelsen,
- koppla upp en ny förbindelse,
- meddela avbrott på förbindelsen,
- meddela QoS på förbindelsen,
- meddela maximal paketlängd på förbindelsen,
- meddela mediatyp på förbindelsen,
- ta emot ett paket för sändning,
- meddela att mottaget paket finns att hämta,
- meddela längd på sändkön,
- meddela maxlängd på sändkön,
- sätt ny maxlängd på sändkön,
- meddela längd på mottagningskön,
- meddela maxlängd på mottagningskön,
- sätt ny maxlängd på mottagningskön, och
- ladda media.

Det fullständiga gränssnittet för transportskiktet i MIDL:

```
[
 uuid(3B781756-6433-4D4B-9802-C83E95F8B165),
 version(1.0),
 helpstring("PTransport Library")

]
library PTransport
{

 importlib("stdole2.tlb");
 importlib("STDVCL40.DLL");

  [
   uuid(6FCEF03D-48FC-4BD0-ABDA-B71692237A36),
   version(1.0),
   helpstring("Dispatch interface for Media Object"),
   dual,
   oleautomation
  ]
   interface IMedia: IDispatch
  {
   [
   helpstring("Connect to  another node, contact info is for example telephone number or
IP address, depending upon the media")
   ]
   HRESULT _stdcall ConnectionOpen([in] VARIANT ContactInfo );
   [
```

```
    id(0x00000002),
    helpstring("Disconnects the current communication link.")
    ]
    HRESULT _stdcall ConnectionClose( void );
    [
    id(0x00000003),
    helpstring("Inserts a packet into the transmit queue.")
    ]
    HRESULT _stdcall PacketTransmit([in] VARIANT ThePacket );
    [
    id(0x00000004),
    helpstring("Fetches a packet from the receive queue.")
    ]
    HRESULT _stdcall PacketReceive([out] VARIANT * ThePacket, [out] long * QueueL-
ength );
    [
    id(0x00000005),
    helpstring("Returns the size of the transmit queue")
    ]
    HRESULT _stdcall QueueGetTransmitSize([out] long * Size );
    [
    id(0x00000006),
    helpstring("Returns the size of the receive queue.")
    ]
    HRESULT _stdcall QueueGetReceiveSize([out] long * Size );
    [
    id(0x00000007),
    helpstring("Returns the maximum size of he transmit queue.")
    ]
    HRESULT _stdcall QueueGetTransmitMaxSize([out] long * Size );
    [
    id(0x00000008),
    helpstring("Returns the maximum size (number of packets) of the receive queue.")
    ]
    HRESULT _stdcall QueueGetReceiveMaxSize([out] long * Size );
    [
    id(0x00000009),
    helpstring("Sets the maxiimum size (number of packets) of the transmit queue.")
    ]
    HRESULT _stdcall QueueSetTransmitMaxSize([in] long * Size );
    [
    id(0x0000000A),
    helpstring("Sets the maximum size (number of packets) of the receive queue.")
    ]
    HRESULT _stdcall QueueSetReceiveMaxSize([in] long * Size );
    [
    id(0x0000000B),
    helpstring("Returns the maximum size (in bytes) of a packet for this media.")
    ]
    HRESULT _stdcall PacketGetMaxSize([out] long * Size );
    [
    id(0x0000000C),
    helpstring("Returns the contact info (as used by the ConnectionOpen method).")
    ]
    HRESULT _stdcall ConnectionGetAddress([out] VARIANT * Info );
    [
    id(0x0000000D)
    ]
    HRESULT _stdcall ConnectionGetStatus( void );
    [
    id(0x0000000E)
```

```
  ]
  HRESULT _stdcall Start([in] BSTR ConfigFile );
  [
  id(0x0000000F)
  ]
  HRESULT _stdcall Stop( void );
};

[
  uuid(3A7790DA-4F32-4B0C-A8AA-9236B11E9995),
  version(1.0),
  helpstring("Events interface for Media Object")
]
 dispinterface IMediaEvents
{
  properties:
  methods:
  [
  id(0x00000001),
  helpstring("This event occurs when a packet is inserted in the receive queue.")
  ]
  HRESULT OnPacketReceived( void );
  [
  id(0x00000002),
  helpstring("Called when the contact is lost over an open connection.")
  ]
  HRESULT OnLinkLost( void );
  [
  id(0x00000003),
  helpstring("Occurs when another node is trying to connect to this media (perhaps by
calling the mobile phone).")
  ]
  HRESULT OnLinkRequest( void );
};

[
  uuid(129D97CC-B5AC-4D55-B6F1-2B6EC10BD394),
  version(1.0),
  helpstring("Media Object")
]
coclass Media
{
  [default] interface IMedia;
  [default, source] dispinterface IMediaEvents;
};

};
```

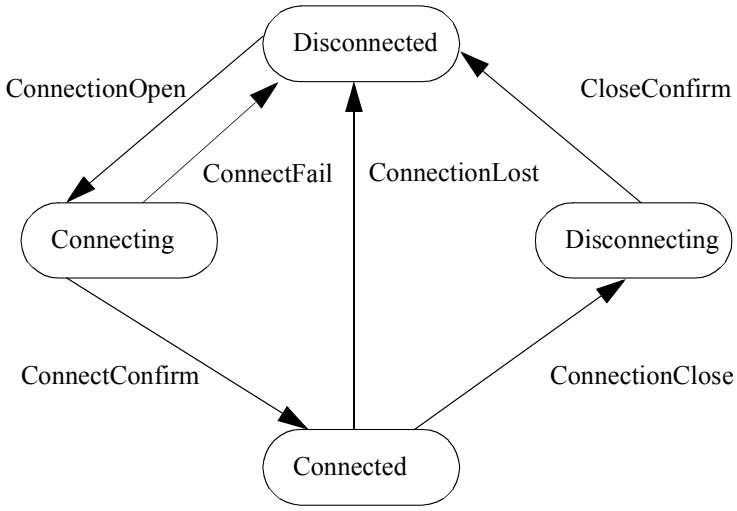
3.3.4.2 Mediamoduler

Styrningen av de olika länkarna varierar beroende på vilket medium som används för kommunikationen. Det är därför nödvändigt att ha en adapter eller drivrutin för varje medium som skall användas i nätverket. Mediamodulerna används till detta och har ett standardiserat interface mot resten av kommunikationslagret, vilket är gemensamt för alla medier.

Eftersom mediamodulerna är starkt knutna till transportskiktet implementeras dessa som en enhet, d.v.s. varje mediamodul implementerar transportskiktets gränssnitt och innehåller den funktionalitet som behövs för detta.

Varje mediamodul implementeras som en tillståndsmaskin, se Fig. 3-7. Tillståndsmaskinen kan bli mer komplicerad om mediet kräver detta. Loopbackmodulen implementerar dock denna enkla tillståndsmaskin, eftersom Loopbackmodulen endast skickar tillbaks de paket som skickas in i den från motorskiktet. Av denna anledning kallas Loopbackmodulen för PTransport.exe, se A.4 PTransport.exe på sidan 151.

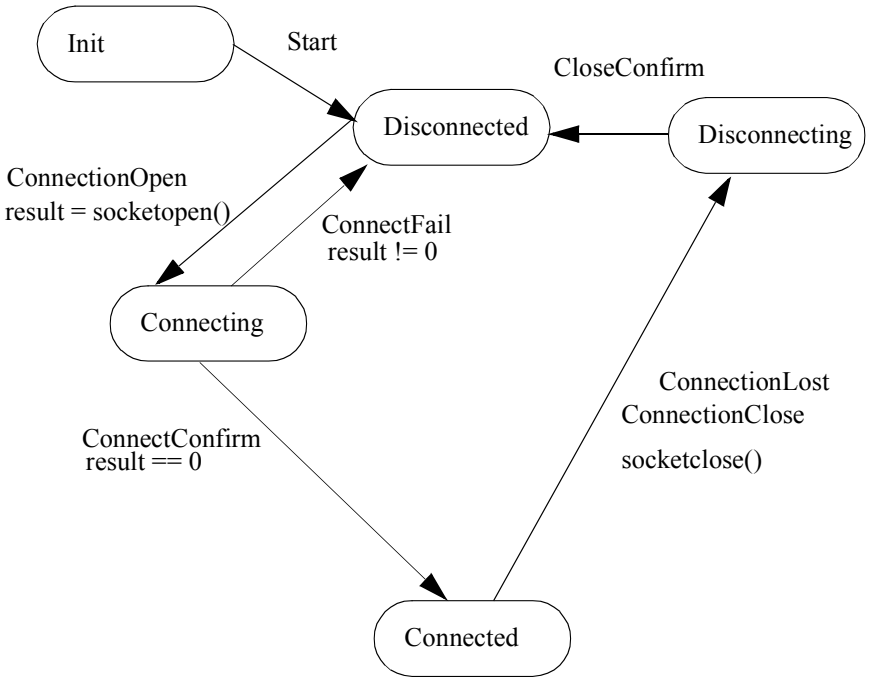
Fig. 3-7. Tillståndsmaskin för allmän mediamodul.



3.3.4.3 UDP

UDP använder inte uppkopplingar, utan skickar bara ut paketen på nätet. Felhantering och fragmentering sker på en högre nivå. Detta gör att UDP kräver att felhantering sker i transportskiktet.

Fig. 3-8. Tillståndsmaskin för UDP mediamodul.



3.3.4.4 TCP

TCP använder portar för att koppla upp sig i en client-serverkonfiguration. Detta innebär att punkt-till-punktförbindelser används även här. En TCPanslutning identifieras av ett portnummer. Varje anslutning kan uppträda antingen som server eller klient. En server ligger och väntar på anslutningsbegäran från klienter. Därför måste en mediamodul först starta en server för att detektera anslutningsbegäran. Om en anslutningsbegäran inkommer till servern skickas en händelse uppåt i systemet och berättar detta. Om den godkänns så går mediamodulen över i tillståndet ServerKoppling. TCP innehåller själv fel- och flödeskontroll, vilket gör att man slipper implementera detta i mediamodulen, vilket är anledningen till att TCP valts framför UDP i prototypen. Implementationen av mediamodulen för TCP finns i A.6 PMediaTCP.exe på sidan 165.

3.4 Systemarkitektur

De olika objekten har implementerats som COMkomponenter i Delphi, vilket bl.a. betyder att de olika komponenterna är implementationsspråksoberoende.

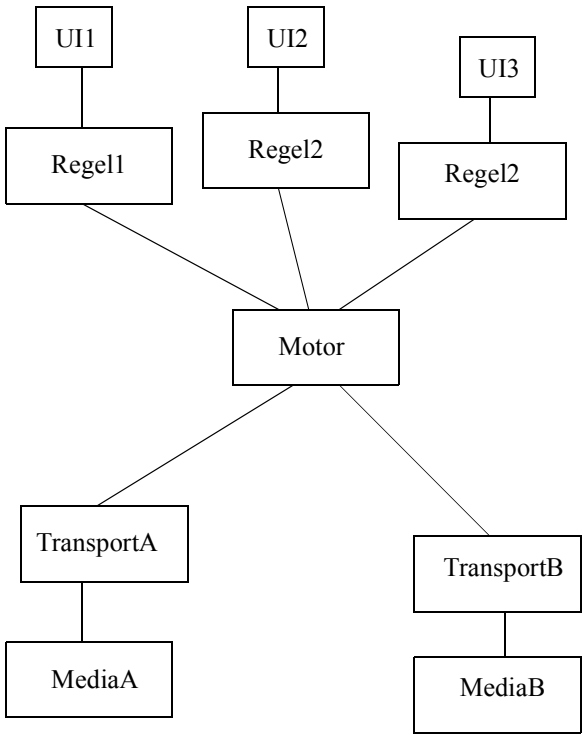
Det kan krävas flera instanser av regelskiktet om flera användare vill logga in på samma maskin. Ett regelskikt motsvarar en nod. Därför har regelskiktet implementerats som en COM komponent som skapar en ny instans av sig själv varje gång någon försöker använda den. Vidare har regelskiktet lagts i ett bibliotek (dll) för att det ska vara enkelt att använda från användargränssnittet. För att ytterligare stödja utveckling av gränssnitt så har regelskiktet implementerats som en ActiveX-komponent, så att det enkelt kan importeras i en grafisk utvecklingsmiljö. Implementationen av regelskiktet visas i A.1 hermeslib.dll på sidan 48.

Det krävs flera instanser av transportskiktet eftersom det finns flera olika kommunikationsmedia. Därför har transportskiktet implementerats som en COM komponent som skapar en ny instans av sig själv varje gång någon försöker använda den. Implementationen av transportskiktet är endast ett gränssnitt som varje mediamodul måste stödja.

Det krävs endast en instans av motorskiktet, då detta endast har till uppgift att upprätthålla kommunikationen med övriga motorskikt. Det är sannolikt olämpligt att ha flera instanser av motorskiktet, eftersom dessa i så fall måste kommunicera intensivt med varandra. Därför har motorskiktet implementerats som en komponent som endast kan existera i en instans i systemet.

Denna design ger upphov till en systemarkitektur enligt Fig. 3-9. Det finns ett godtyckligt antal användargränssnitt på varje maskin, och varje användargränssnitt är kopplat till var sitt regellager. Samtliga instanser av regelskiktet kommunicerar med den enda instansen av motorskiktet, som i sin tur styr de mediamoduler som är kopplade till maskinen.

Fig. 3-9. HERMES systemarkitektur.



3.5 Meddelande

Den information som ett meddelande innehåller är:

- Mottagare,
- avsändare,

- prioritet,
- datum/tid,
- datum/tid mottaget,
- livslängd,
- datastorlek,
- meddelandekropp (En mängd text, eller binär data).

Denna information skickas in till HERMES genom argumenten i ett metodanrop, när ett meddelande skickas.

3.6 Paket

Meddelandet fragmenteras till paket i motorskiktet, eftersom det är olämpligt att skicka stora meddelanden i en lång dataström (felsannolikheten ökar, och trafik med högre prioritet kan inte komma in emellan). I den gamla prototypen används Z-modem och varje meddelande överförs som en fil. Tyvärr har z-modem en del egenskaper, som inte är lämpliga, bl.a. måste hela filen överföras punkt-till-punkt, innan meddelandet kan skickas vidare. För stora meddelanden ställer det högre krav på buffertminne, och resulterar i längre fördröjning vid multihopp-kommunikation. Z-modem kan heller inte användas för paketöverföring, utan att mellanlagra paketen som en fil.

Varje paket måste förutom data innehålla destinationsnoden, samt routinginformation och sekvensnummer (för att kunna montera ihop ett paket till ett meddelande).

Transportlagret är ansvarigt för datagenomskinlighet och checksummekontroll. Motorlagret är ansvarigt för routinginformation och fragmentering av meddelandet till lagom stora paket för att transportlagret skall kunna hantera dem.

Två typer av paket har definierats. Ett datapaket och ett ACKpaket. Datapaketet innehåller en del av ett meddelande och skickas från sändaren till mottagaren. Ett ACKpaket är ett svar på att mottagaren har fått ett datapaket, och skickas därför från mottagaren till avsändaren.

Ett datapaket som hanteras i motorlagret för flooding har följande information, se Fig. 3-10.

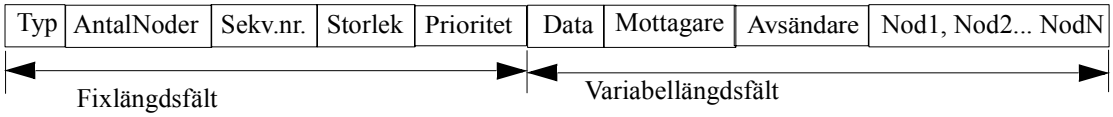
- Typ := Flood data (Word),
- prioritet (byte),
- mottagare (sträng),
- sekvensnummer (Long),
- avsändare (sträng),
- antal besökta noder (Word),
- besökt nod1, nod2... nodN (strängar),
- storlek (long), och
- data (storlek antal bytes).

Datatyperna är:

- Long är ett 32 bitars heltal,

- word är ett 16 bitars heltal,
- byte är ett 8 bitars heltal, och
- sträng är en följd av byte, som avslutas med byte 0.

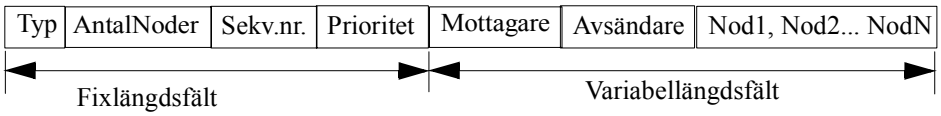
Fig. 3-10. Paketformat datapaket



Ett Flooding ACKpaket har följande utseende, se Fig. 3-11.

- Typ := Flood ACK (word)
- Prioritet (byte)
- Mottagare (sträng)
- ACKnummer (Long)
- Avsändare (sträng)
- Antal besökta noder (Word)
- Besökt nod1, nod2... nod N (Strängar)

Fig. 3-11. Paketformat ACKpaket



Ett paket som sänds över en länk innehåller följande information:

- Startflagga (byte),
- motorpaket,
- checksumma, och
- slutflagga (byte).

Om medium som används sköter detta, som för t.ex. TCP, behöver ingen information läggas till i mediamodulen. Eftersom TCP är det enda protokoll som implementerats hittills, så har inget mediapaketformat bestämts.

3.7 Testbänk

Det är lämpligt att skriva en testbänk för kommunikationsmotorn, där funktionaliteten testas mot fiktiva regel- och transportlager. (regellagren kan vara skarpa, men det är lämpligt att konstruera stubbar för transportlagren). Dessa kan sedan knytas samman med ett virtuellt nätverk, flera olika typer av stubbar kan skrivas för att testa heterogenitetseffekter. Detta kräver att man kan köra flera separata

instanser av Hermes på samma dator. Med en sådan uppsättning kan brott på länkar, samt korrupt kommunikation enkelt simuleras.

3.8 Installering av utvecklingsmiljön

För att installera utvecklingsmiljön krävs följande:

- Microsoft Windows 2000 professional,
- källkoder för Hermes,
- Borland Delphi 6.0,
- Async Professional 4.0.

Andra versioner av programmen kan fungera, men har inte testats.

Följande steg är nödvändiga för att använda miljön:

1. Installera Delphi
2. Installera Async Professional
3. Packa upp zip-filen till rotkatalogen på C:
4. Öppna C:\projects\hermes2.bpg (*File -> Open Project...*)
5. aktivera projektet HermesLib.dll genom att dubbelklicka på det i Project Manager (välj *View -> Project Manager* om den inte visas för tillfället).
6. kompilera HermesLib.dll (*Project -> Build HermesLib*)
7. registrera HermesLib.dll (*Run -> Register ActiveX Server*)
8. skapa hermescontrol (*Project -> Import Type Library...*, välj HermesLib ur listan)
9. Registrera PEngine.exe, PTransport.exe, och PMediaTCP.exe genaom att i tur och ordning aktivera projekten (dubbelklicka på dem i Project Manager) och köra dem (*Run -> Run*). Stäng sedan programmen direkt efter att de startat.
10. Testa utvecklingsmiljön genom att aktivera GUI.exe och välja *Run -> Run*.

3.9 Installering av körbart system

Ett installationspaket har gjorts med Installshield. Tyvärr gör inte installationsprogrammet allt som behövs för att kunna köra prototypen. Följande måste göras för att kunna köra prototypen på ett system (det system som prototypen har testats på är Microsoft Windows 2000 professional):

1. Kör installationpaketet (setup.exe)
2. Flytta filen hermesconfig.hme från installationskatalogen, till C:\WINNT\System32\
3. Editera filen hermesconfig.hme för att passa den hårdvara som finns.
4. Editera eventuellt de mediakonfigurationsfiler som behöver ändras.

4 Slutsatser

Flooding har funnits vara ett lämpligt routingprotokoll att använda som bas för inledande försök och vidare utveckling, dels för att det är ett mycket robust och lättimplementerat protokoll, och dels för att det är en bas i de flesta andra routingprotokoll för MANET.

Hermes kan sägas vara ett heterogent MANET, och därför är det av stort intresse att bevaka forskningen inom detta område. Dock ställer heterogeniteten ytterligare krav på Hermessystemet som för närvarande inte hanteras av MANET och det finns många problem kvar att lösa.

Ett olöst grundläggande problem är hur nätverket ska upprätthållas på bästa sätt. Hur väljs en robust topologi? Hur kan nätverket läka sig själv efter sönderfall? Detta måste utredas närmare innan det är möjligt att bygga ett väl fungerande nätverk.

För att kunna testa de effekter som heterogeniteten ger på nätverket måste prototypen utökas med fler kommunikationssätt, t.ex. mobiltelefon och fast telefon. Ett alternativ och bra komplement är att bygga en testbänk för att simulera ett heterogent nätverk, antingen över ett homogent nätverk, eller på en maskin.

För att kunna avgöra om nätverket fungerar bra för de applikationer som det är avsett för måste man också utreda vilken information som kommer att skickas i nätverket och vilka krav detta ställer på systemet.

Referenser

[1] M. Brage, “Hermes II - En inledande systembeskrivning”, Linköping, FOI 2001, 17 s. FOI-RH--0073--SE (Underlagsrapport).

[2] F. Halsall, “Data Communications, Computer Networks and Open Systems”, fourth edition. Addison-Wesley, ISBN 0-201-42293-X, 1996.

[3] C. Perkins, E. Belding-Royer, and S. Das. “IP Flooding in Ad Hoc Mobile Networks”, Mobile Ad Hoc Networking Working Group Memo, November 2001.

Appendix AProgramlistningar

A.1 hermeslib.dll

A.1.1 C:\projects\hermes\HermesLib.dpr

```
1: library HermesLib;
2:
3: uses
4: ComServ,
5: HermesLib_TLB in 'HermesLib_TLB.pas',
6: HermesMessage in 'HermesMessage.pas',
7: HermesAttribute in 'HermesAttribute.pas',
8: HermesTypes in 'HermesTypes.pas',
9: ControlImpl in 'ControlImpl.pas' {HermesControl: CoClass};
10:
11: {$E dll}
12:
13: exports
14: DllGetClassObject,
15: DllCanUnloadNow,
16: DllRegisterServer,
17: DllUnregisterServer;
18:
19: {$R *.TLB}
20:
21: {$R *.RES}
22:
23: begin
24: end.
```

A.1.2 C:\projects\hermes\ControlImpl.pas

```
1: unit ControlImpl;
2:
3: {$WARN SYMBOL_PLATFORM OFF}
4:
5: interface
6:
7: uses
8: HermesAttribute, PEngine_TLB, SysUtils,
9: ComObj, ActiveX, AxCtrls, Classes, HermesLib_TLB, StdVcl;
10:
11: type
12: THermesControl = class(TAutoObject, IConnectionPointContainer, IHermesControl)
13:
14: private
15: { Private declarations }
16: FConnectionPoints: TConnectionPoints;
17: FConnectionPoint: TConnectionPoint;
18: FEvents: IHermesControlEvents;
19: { note: FEvents maintains a *single* event sink. For access to more
20: than one event sink, use FConnectionPoint.SinkList, and iterate
21: through the list of sinks. }
22: //FEngine: IHermesSession;
23: //FSession: TDataModule1;
24: FAttributeList: TAttributeList;
25: FSession: THermesSession;
26:
```

```
27: function CheckPassword(const Node, Password: WideString): Boolean;
28: public
29: procedure Initialize; override;
30: protected
31: { Protected declarations }
32: property ConnectionPoints: TConnectionPoints read FConnectionPoints
33: implements IConnectionPointContainer;
34: procedure EventSinkChanged(const EventSink: IUnknown); override;
35: procedure AttributeAdd(const Key, Value: WideString); safecall;
36: procedure AttributeChange(const Key, Value: WideString); safecall;
37: procedure AttributeRead(const Key: WideString; out Value: WideString);
38: safecall;
39: procedure AttributeRemove(const Key: WideString); safecall;
40: procedure AttributeSubscriptionEnd(const Key: WideString); safecall;
41: procedure AttributeSubscriptionStart(const Key: WideString); safecall;
42: procedure LogIn(const Node, Password: WideString); safecall;
43: procedure LogOut; safecall;
44: procedure MessageReceive(out TheMessage: OleVariant; out Count: Integer);
45: safecall;
46: procedure MessageSend(TheMessage: OleVariant); safecall;
47: procedure NodeAdd(const Node: WideString); safecall;
48: procedure NodeConnectedQuery; safecall;
49: procedure NodeMediaAdd(const Node, Type_, Address, Port: WideString);
50: safecall;
51: procedure NodeMediaQuery; safecall;
52: procedure NodeMediaRemove(const Node, Type_, Address: WideString);
53: safecall;
54: procedure NodeRemove(const Node: WideString); safecall;
55: procedure QOSExplore; safecall;
56: procedure QOSRequest; safecall;
57: procedure QueueGetReceiveMaxSize; safecall;
58: procedure QueueGetReceiveSize; safecall;
59: procedure QueueGetTransmitMaxSize; safecall;
60: procedure QueueGetTransmitSize; safecall;
61: procedure QueueSetReceiveMaxSize; safecall;
62: procedure QueueSetTransmitMaxSize; safecall;
63:
64: procedure OnMessageAck(Sender: TObject);
65: procedure OnMessageNak(Sender: TObject);
66: procedure OnMessageReceived(Sender: TObject);
67: procedure OnContactLost(Sender: TObject);
68: procedure OnLinkLost(Sender: TObject);
69: procedure MediaConnect(const MediaType, MediaAddress,
70: ConnectionAddress: WideString); safecall;
71: procedure MediaConnectionAddress(const MediaType, MediaAddress: WideString;
72: out ConnectionAddress: WideString); safecall;
73: procedure MediaDisconnect(const MediaType, MediaAddress: WideString);
74: safecall;
75: procedure MediaInventory(out MediaList: OleVariant); safecall;
76: procedure MediaStatus(const MediaType, MediaAddress: WideString;
77: out MediaStatus: SYSINT); safecall;
78:
79:
80: end;
81:
82:
83: implementation
84:
85: uses ComServ;
86:
87: procedure THermesControl.OnMessageReceived(Sender: TObject);
```

```
88: begin
89: FEvents.OnMessageReceived;
90: end;
91:
92: procedure THermesControl.OnMessageAck(Sender: TObject);
93: begin
94: end;
95: procedure THermesControl.OnMessageNak(Sender: TObject);
96: begin
97: end;
98: procedure THermesControl.OnContactLost(Sender: TObject);
99: begin
100: end;
101: procedure THermesControl.OnLinkLost(Sender: TObject);
102: begin
103: end;
104:
105:
106: procedure THermesControl.EventSinkChanged(const EventSink: IUnknown);
107: begin
108: FEvents := EventSink as IHermesControlEvents;
109: end;
110:
111: procedure THermesControl.Initialize;
112: begin
113: inherited Initialize;
114: FConnectionPoints := TConnectionPoints.Create(Self);
115: if AutoFactory.EventTypeInfo <> nil then
116: FConnectionPoint := FConnectionPoints.CreateConnectionPoint(
117: AutoFactory.EventIID, ckSingle, EventConnect)
118: else FConnectionPoint := nil;
119: //if not Assigned(FEngine) then
120: // FEngine := CoHermesSession.Create;
121: FAttributeList := TAttributeList.Create;
122: //FSession := TDataModule1.Create(nil);
123:
124:
125: if not Assigned(FSession) then
126: FSession := THermesSession.Create(nil);
127: if not Assigned(FSession) then
128: //ShowMessage('Could not start HermesEngine')
129: else begin
130: FSession.Connect;
131: FSession.OnMessageReceived := Self.OnMessageReceived;
132: end;
133:
134: end;
135:
136:
137: procedure THermesControl.AttributeAdd(const Key, Value: WideString);
138: begin
139: FAttributeList.Add(Key, Value);
140: end;
141:
142: procedure THermesControl.AttributeChange(const Key, Value: WideString);
143: begin
144:
145: end;
146:
147: procedure THermesControl.AttributeRead(const Key: WideString;
148: out Value: WideString);
```

```
149: begin
150:
151: end;
152:
153: procedure THermesControl.AttributeRemove(const Key: WideString);
154: begin
155:
156: end;
157:
158: procedure THermesControl.AttributeSubscriptionEnd(const Key: WideString);
159: begin
160:
161: end;
162:
163: procedure THermesControl.AttributeSubscriptionStart(const Key: WideString);
164: begin
165:
166: end;
167:
168: function THermesControl.CheckPassword(const Node, Password: WideString):
Boolean;
169: begin
170: CheckPassword := true;
171: end;
172:
173:
174: procedure THermesControl.LogIn(const Node, Password: WideString);
175: begin
176: if CheckPassword(Node, Password) then begin
177:
178:
179: // Get the Engine
180: //if not Assigned(FEngine) then
181: // FEngine := CoHermesSession.Create;
182: //if not Assigned(FEngine) then
183: // ShowMessage('Could not start HermesEngine')
184: //else begin
185: FAttributeList.Add('Name', Node);
186: FSession.Open(Node);
187: //TheController := self;
188: //FEngine.Open(Node);
189: end else begin
190: // Wrong password!
191: end;
192: end;
193:
194: procedure THermesControl.LogOut;
195: var Name: WideString;
196: begin
197: Name := FAttributeList.Read('Name');
198: //FEngine.NodeRemove(Name);
199: FAttributeList.Clear;
200: FSession.Close;
201: end;
202:
203: procedure THermesControl.MessageReceive(out TheMessage: OleVariant;
204: out Count: Integer);
205: var
206: MessageVariant: OleVariant;
207: MessageCount: Integer;
208: begin
```

```
209: FSession.MessageReceive(MessageVariant, MessageCount);
210: if MessageCount > 0 then
211: begin
212: TheMessage := MessageVariant;
213: Count := MessageCount;
214: end
215: else
216: begin
217: Count := 0;
218: end;
219: end;
220:
221: procedure THermesControl.MessageSend(TheMessage: OleVariant);
222: begin
223: FSession.MessageSend(TheMessage);
224: end;
225:
226: procedure THermesControl.NodeAdd(const Node: WideString);
227: begin
228:
229: end;
230:
231: procedure THermesControl.NodeConnectedQuery;
232: begin
233:
234: end;
235:
236: procedure THermesControl.NodeMediaAdd(const Node, Type_, Address,
237: Port: WideString);
238: begin
239:
240: end;
241:
242: procedure THermesControl.NodeMediaQuery;
243: begin
244:
245: end;
246:
247: procedure THermesControl.NodeMediaRemove(const Node, Type_,
248: Address: WideString);
249: begin
250:
251: end;
252:
253: procedure THermesControl.NodeRemove(const Node: WideString);
254: begin
255:
256: end;
257:
258: procedure THermesControl.QOSExplore;
259: begin
260:
261: end;
262:
263: procedure THermesControl.QOSRequest;
264: begin
265:
266: end;
267:
268: procedure THermesControl.QueueGetReceiveMaxSize;
269: begin
```

```
270:
271: end;
272:
273: procedure THermesControl.QueueGetReceiveSize;
274: begin
275:
276: end;
277:
278: procedure THermesControl.QueueGetTransmitMaxSize;
279: begin
280:
281: end;
282:
283: procedure THermesControl.QueueGetTransmitSize;
284: begin
285:
286: end;
287:
288: procedure THermesControl.QueueSetReceiveMaxSize;
289: begin
290:
291: end;
292:
293: procedure THermesControl.QueueSetTransmitMaxSize;
294: begin
295:
296: end;
297:
298: procedure THermesControl.MediaConnect(const MediaType, MediaAddress,
299: ConnectionAddress: WideString);
300: begin
301: FSession.MediaConnect(MediaType, MediaAddress, ConnectionAddress);
302: end;
303:
304: procedure THermesControl.MediaConnectionAddress(const MediaType,
305: MediaAddress: WideString; out ConnectionAddress: WideString);
306: begin
307: FSession.MediaConnectionAddress(MediaType, MediaAddress, Connectio-
nAddress);
308: end;
309:
310: procedure THermesControl.MediaDisconnect(const MediaType,
311: MediaAddress: WideString);
312: begin
313: FSession.MediaDisconnect(MediaType, MediaAddress);
314: end;
315:
316: procedure THermesControl.MediaInventory(out MediaList: OleVariant);
317: begin
318: FSession.MediaInventory(MediaList);
319: end;
320:
321: procedure THermesControl.MediaStatus(const MediaType,
322: MediaAddress: WideString; out MediaStatus: SYSINT);
323: begin
324: FSession.MediaStatus(MediaType, MediaAddress, MediaStatus);
325: end;
326:
327: initialization
328: TAutoObjectFactory.Create(ComServer, THermesControl, Class_HermesControl,
329: ciSingleInstance, tmFree);
```

```
330:
331: end.
```

A.1.3 C:\projects\hermes\HermesAttribute.pas

```
1: unit HermesAttribute;
2:
3: interface
4:
5: uses
6: Classes, HermesTypes, IniFiles;
7:
8: type
9:
10: TAttributeValue = WideString;
11: TAttributeKey = WideString;
12: PAttributeKey = ^TAttributeKey;
13: PAttributeValue = ^TAttributeValue;
14:
15: TAttribute = Class(TObject)
16: private
17: TheKey : TAttributeKey;
18: TheValue : TAttributeValue;
19: Subscribers : TStringList;
20: protected
21: public
22: constructor Create(Key: TAttributeKey; Value: TAttributeValue);
23: procedure SetKey(Key: TAttributeKey);
24: function GetKey: TAttributeKey;
25: procedure SetValue(Value: TAttributeValue);
26: function GetValue: TAttributeValue;
27: procedure SubscriptionStart(Node: TNodeID);
28: procedure SubscriptionEnd(Node: TNodeID);
29: end;
30:
31: PAttribute = ^TAttribute;
32:
33: TAttributeList = Class(THashedStringList)
34: private
35: protected
36: public
37: procedure Update(Key: TAttributeKey; Value: TAttributeValue);
38: procedure SubscriptionStart(Key: TAttributeKey; Node: TNodeID);
39: procedure SubscriptionEnd(Key: TAttributeKey; Node: TNodeID);
40: procedure Add(Key: TAttributeKey; Value: TAttributeValue);
41: function Read(Key: TAttributeKey): TAttributeValue;
42: constructor Create;
43: end;
44:
45: implementation
46:
47: constructor TAttribute.Create(Key: TAttributeKey; Value: TAttributeValue);
48: begin
49: TheKey := Key;
50: TheValue := Value;
51: end;
52:
53: procedure TAttribute.SetKey(Key: TAttributeKey);
54: begin
55: TheKey := Key;
56: end;
```

```
57:
58: function TAttribute.GetKey: TAttributeKey;
59: begin
60: GetKey := TheKey;
61: end;
62:
63: procedure TAttribute.SetValue(Value: TAttributeValue);
64: begin
65: TheValue := Value;
66: end;
67:
68: function TAttribute.GetValue: TAttributeValue;
69: begin
70: GetValue := TheValue;
71: end;
72:
73: procedure TAttribute.SubscriptionStart(Node: TNodeID);
74: var Index: Integer;
75: begin
76: Index := Subscribers.IndexOf(Node);
77: if Index <> -1 then
78: begin
79: // Error handling for Node already subscriber
80: end
81: else
82: begin
83: Subscribers.Add(Node);
84: end;
85: end;
86:
87: procedure TAttribute.SubscriptionEnd(Node: TNodeID);
88: var Index: Integer;
89: begin
90: Index := Subscribers.IndexOf(Node);
91: if Index <> -1 then
92: begin
93: Subscribers.Delete(Index);
94: end
95: else
96: begin
97: // Error handling for Node not subscriber
98: end;
99: end;
100:
101: constructor TAttributeList.Create;
102: begin
103: inherited Create;
104: Sorted := true;
105: Duplicates := dupError;
106: end;
107:
108: procedure TAttributeList.Add(Key: TAttributeKey; Value: TAttributeValue);
109: var Attribute: TAttribute;
110: begin
111: Attribute := TAttribute.Create(Key, Value);
112: AddObject(Key ,Attribute);
113: end;
114:
115: procedure TAttributeList.Update(Key: TAttributeKey; Value: TAttributeValue);
116: var Index: Integer;
117: begin
```

```
118: Index := IndexOf(Key);
119: if Index <> -1 then
120: begin
121: TAttribute(Objects[Index]).SetValue(Value);
122: end
123: else
124: begin
125: // TODO: Error handling for non existing record.
126: end;
127: end;
128:
129: procedure TAttributeList.SubscriptionStart(Key: TAttributeKey; Node: TNodeID);
130: var Index: Integer;
131: begin
132: Index := IndexOf(Key);
133: if Index <> -1 then
134: begin
135: TAttribute(Objects[Index]).SubscriptionStart(Node);
136: end
137: else
138: begin
139: // TODO: Error handling for non existing record.
140: end;
141: end;
142:
143: procedure TAttributeList.SubscriptionEnd(Key: TAttributeKey; Node: TNodeID);
144: var Index: Integer;
145: begin
146: Index := IndexOf(Key);
147: if Index <> -1 then
148: begin
149: TAttribute(Objects[Index]).SubscriptionEnd(Node);
150: end
151: else
152: begin
153: // TODO: Error handling for non existing record.
154: end;
155: end;
156:
157: function TAttributeList.Read(Key: TAttributeKey): TAttributeValue;
158: var Index: Integer;
159: begin
160: Index := IndexOf(Key);
161: if Index <> -1 then
162: begin
163: Read := TAttribute(Objects[Index]).GetValue;
164: end
165: else
166: begin
167: // TODO: Error handling for non existing record.
168: end;
169: end;
170:
171: end.
```

A.1.4 C:\projects\hermes\HermesLib_TLB.pas

```
1: unit HermesLib_TLB;
2:
3: // ************************************************************************ //
4: // WARNING
```

```
5: // -------
6: // The types declared in this file were generated from data read from a
7: // Type Library. If this type library is explicitly or indirectly (via
8: // another type library referring to this type library) re-imported, or the
9: // 'Refresh' command of the Type Library Editor activated while editing the
10: // Type Library, the contents of this file will be regenerated and all
11: // manual modifications will be lost.
12: // ************************************************************************ //
13:
14: // PASTLWTR : $Revision: 1.130 $
15: // File generated on 2002-04-24 13:06:58 from Type Library described below.
16:
17: // ************************************************************************ //
18: // Type Lib: C:\projects\hermes\HermesLib.dll (1)
19: // LIBID: {5836DADE-10C5-11D6-9231-00008637B362}
20: // LCID: 0
21: // Helpfile:
22: // DepndLst:
23: // (1) v2.0 stdole, (C:\WINNT\System32\stdole2.tlb)
24: // (2) v4.0 StdVCL, (C:\WINNT\System32\stdvcl40.dll)
25: // ************************************************************************ //
26: // ************************************************************************//
27: // NOTE:
28: // Items guarded by $IFDEF_LIVE_SERVER_AT_DESIGN_TIME are used by properties
29: // which return objects that may need to be explicitly created via a function
30: // call prior to any access via the property. These items have been disabled
31: // in order to prevent accidental use from within the object inspector. You
32: // may enable them by defining LIVE_SERVER_AT_DESIGN_TIME or by selectively
33: // removing them from the $IFDEF blocks. However, such items must still be
34: // programmatically created via a method of the appropriate CoClass before
35: // they can be used.
36: {$TYPEDADDRESS OFF} // Unit must be compiled without type-checked pointers.
37: {$WARN SYMBOL_PLATFORM OFF}
38: {$WRITEABLECONST ON}
39:
40: interface
41:
42: uses ActiveX, Classes, Graphics, OleServer, StdVCL, Variants, Windows;
43:
44:
45:
46: // ************************************************************************//
47: // GUIDS declared in the TypeLibrary. Following prefixes are used:
48: // Type Libraries : LIBID_xxxx
49: // CoClasses : CLASS_xxxx
50: // DISPInterfaces : DIID_xxxx
51: // Non-DISP interfaces: IID_xxxx
52: // ************************************************************************//
53: const
54: // TypeLibrary Major and minor versions
55: HermesLibMajorVersion = 1;
56: HermesLibMinorVersion = 0;
57:
58: LIBID_HermesLib: TGUID = '{5836DADE-10C5-11D6-9231-00008637B362}';
59:
60: IID_IHermesControl: TGUID = '{8FA11B77-DB32-4DA1-B44D-0AB8673C17B6}';
61: DIID_IHermesControlEvents: TGUID = '{DD425DB2-0061-4CD5-AEC6-
5A8B4C36CA92}';
62: CLASS_HermesControl: TGUID = '{2550F702-C223-4E52-9484-67C5B67D2565}';
63: type
64:
```

```
65: // ************************************************************************//
66: // Forward declaration of types defined in TypeLibrary
67: // ************************************************************************//
68: IHermesControl = interface;
69: IHermesControlDisp = dispinterface;
70: IHermesControlEvents = dispinterface;
71:
72: // ************************************************************************//
73: // Declaration of CoClasses defined in Type Library
74: // (NOTE: Here we map each CoClass to its Default Interface)
75: // ************************************************************************//
76: HermesControl = IHermesControl;
77:
78:
79: // ************************************************************************//
80: // Interface: IHermesControl
81: // Flags: (4416) Dual OleAutomation Dispatchable
82: // GUID: {8FA11B77-DB32-4DA1-B44D-0AB8673C17B6}
83: // ************************************************************************//
84: IHermesControl = interface(IDispatch)
85: ['{8FA11B77-DB32-4DA1-B44D-0AB8673C17B6}']
86: procedure MessageSend(TheMessage: OleVariant); safecall;
87: procedure MessageReceive(out TheMessage: OleVariant; out Count: Integer); safe-
call;
88: procedure QOSRequest; safecall;
89: procedure QOSExplore; safecall;
90: procedure QueueGetTransmitSize; safecall;
91: procedure QueueGetReceiveSize; safecall;
92: procedure QueueGetTransmitMaxSize; safecall;
93: procedure QueueGetReceiveMaxSize; safecall;
94: procedure QueueSetTransmitMaxSize; safecall;
95: procedure QueueSetReceiveMaxSize; safecall;
96: procedure AttributeAdd(const Key: WideString; const Value: WideString); safecall;
97: procedure AttributeRemove(const Key: WideString); safecall;
98: procedure AttributeRead(const Key: WideString; out Value: WideString); safecall;
99: procedure AttributeChange(const Key: WideString; const Value: WideString); safe-
call;
100: procedure AttributeSubscriptionStart(const Key: WideString); safecall;
101: procedure AttributeSubscriptionEnd(const Key: WideString); safecall;
102: procedure LogOut; safecall;
103: procedure NodeAdd(const Node: WideString); safecall;
104: procedure NodeRemove(const Node: WideString); safecall;
105: procedure NodeMediaAdd(const Node: WideString; const MediaType: WideString;
106: const Address: WideString; const Port: WideString); safecall;
107: procedure NodeMediaRemove(const Node: WideString; const MediaType:
WideString;
108: const Address: WideString); safecall;
109: procedure NodeMediaQuery; safecall;
110: procedure NodeConnectedQuery; safecall;
111: procedure LogIn(const Node: WideString; const Password: WideString); safecall;
112: procedure MediaInventory(out MediaList: OleVariant); safecall;
113: procedure MediaStatus(const MediaType: WideString; const MediaAddress:
WideString;
114: out MediaStatus: SYSINT); safecall;
115: procedure MediaConnectionAddress(const MediaType: WideString; const Medi-
aAddress: WideString;
116: out ConnectionAddress: WideString); safecall;
117: procedure MediaConnect(const MediaType: WideString; const MediaAddress:
WideString;
118: const ConnectionAddress: WideString); safecall;
```

```
119: procedure MediaDisconnect(const MediaType: WideString; const MediaAddress:
WideString); safecall;
120: end;
121:
122: // ************************************************************//
123: // DispIntf: IHermesControlDisp
124: // Flags: (4416) Dual OleAutomation Dispatchable
125: // GUID: {8FA11B77-DB32-4DA1-B44D-0AB8673C17B6}
126: // ************************************************************//
127: IHermesControlDisp = dispinterface
128: ['{8FA11B77-DB32-4DA1-B44D-0AB8673C17B6}']
129: procedure MessageSend(TheMessage: OleVariant); dispid 1;
130: procedure MessageReceive(out TheMessage: OleVariant; out Count: Integer); dispid
2;
131: procedure QOSRequest; dispid 3;
132: procedure QOSExplore; dispid 4;
133: procedure QueueGetTransmitSize; dispid 7;
134: procedure QueueGetReceiveSize; dispid 8;
135: procedure QueueGetTransmitMaxSize; dispid 9;
136: procedure QueueGetReceiveMaxSize; dispid 10;
137: procedure QueueSetTransmitMaxSize; dispid 11;
138: procedure QueueSetReceiveMaxSize; dispid 12;
139: procedure AttributeAdd(const Key: WideString; const Value: WideString); dispid 5;
140: procedure AttributeRemove(const Key: WideString); dispid 6;
141: procedure AttributeRead(const Key: WideString; out Value: WideString); dispid 13;
142: procedure AttributeChange(const Key: WideString; const Value: WideString); dis-
pid 14;
143: procedure AttributeSubscriptionStart(const Key: WideString); dispid 15;
144: procedure AttributeSubscriptionEnd(const Key: WideString); dispid 16;
145: procedure LogOut; dispid 18;
146: procedure NodeAdd(const Node: WideString); dispid 19;
147: procedure NodeRemove(const Node: WideString); dispid 20;
148: procedure NodeMediaAdd(const Node: WideString; const MediaType: WideString;
149: const Address: WideString; const Port: WideString); dispid 21;
150: procedure NodeMediaRemove(const Node: WideString; const MediaType:
WideString;
151: const Address: WideString); dispid 22;
152: procedure NodeMediaQuery; dispid 23;
153: procedure NodeConnectedQuery; dispid 24;
154: procedure LogIn(const Node: WideString; const Password: WideString); dispid 26;
155: procedure MediaInventory(out MediaList: OleVariant); dispid 17;
156: procedure MediaStatus(const MediaType: WideString; const MediaAddress:
WideString;
157: out MediaStatus: SYSINT); dispid 25;
158: procedure MediaConnectionAddress(const MediaType: WideString; const Media-
aAddress: WideString;
159: out ConnectionAddress: WideString); dispid 27;
160: procedure MediaConnect(const MediaType: WideString; const MediaAddress:
WideString;
161: const ConnectionAddress: WideString); dispid 28;
162: procedure MediaDisconnect(const MediaType: WideString; const MediaAddress:
WideString); dispid 29;
163: end;
164:
165: // ************************************************************//
166: // DispIntf: IHermesControlEvents
167: // Flags: (4096) Dispatchable
168: // GUID: {DD425DB2-0061-4CD5-AEC6-5A8B4C36CA92}
169: // ************************************************************//
170: IHermesControlEvents = dispinterface
171: ['{DD425DB2-0061-4CD5-AEC6-5A8B4C36CA92}']
```

```
172: procedure OnMessageReceived; dispid 1;
173: procedure OnMessageAck; dispid 4;
174: procedure OnMessageNak; dispid 5;
175: procedure OnLinkLost; dispid 6;
176: procedure OnContactLost; dispid 7;
177: procedure OnLinkRequest; dispid 8;
178: procedure OnNodeAdd; dispid 2;
179: procedure OnNodeDelete; dispid 3;
180: end;
181:
182: // *********************************************************************//
183: // The Class CoHermesControl provides a Create and CreateRemote method to
184: // create instances of the default interface IHermesControl exposed by
185: // the CoClass HermesControl. The functions are intended to be used by
186: // clients wishing to automate the CoClass objects exposed by the
187: // server of this typelibrary.
188: // *********************************************************************//
189: CoHermesControl = class
190: class function Create: IHermesControl;
191: class function CreateRemote(const MachineName: string): IHermesControl;
192: end;
193:
194:
195: // *********************************************************************//
196: // OLE Server Proxy class declaration
197: // Server Object : THermesControl
198: // Help String : HermesControl Object
199: // Default Interface: IHermesControl
200: // Def. Intf. DISP? : No
201: // Event Interface: IHermesControlEvents
202: // TypeFlags : (2) CanCreate
203: // *********************************************************************//
204: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
205: THermesControlProperties= class;
206: {$ENDIF}
207: THermesControl = class(TOleServer)
208: private
209: FOnMessageReceived: TNotifyEvent;
210: FOnMessageAck: TNotifyEvent;
211: FOnMessageNak: TNotifyEvent;
212: FOnLinkLost: TNotifyEvent;
213: FOnContactLost: TNotifyEvent;
214: FOnLinkRequest: TNotifyEvent;
215: FOnNodeAdd: TNotifyEvent;
216: FOnNodeDelete: TNotifyEvent;
217: FIntf: IHermesControl;
218: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
219: FProps: THermesControlProperties;
220: function GetServerProperties: THermesControlProperties;
221: {$ENDIF}
222: function GetDefaultInterface: IHermesControl;
223: protected
224: procedure InitServerData; override;
225: procedure InvokeEvent(DispID: TDispID; var Params: TVariantArray); override;
226: public
227: constructor Create(AOwner: TComponent); override;
228: destructor Destroy; override;
229: procedure Connect; override;
230: procedure ConnectTo(svrIntf: IHermesControl);
231: procedure Disconnect; override;
232: procedure MessageSend(TheMessage: OleVariant);
```

```
233: procedure MessageReceive(out TheMessage: OleVariant; out Count: Integer);
234: procedure QOSRequest;
235: procedure QOSExplore;
236: procedure QueueGetTransmitSize;
237: procedure QueueGetReceiveSize;
238: procedure QueueGetTransmitMaxSize;
239: procedure QueueGetReceiveMaxSize;
240: procedure QueueSetTransmitMaxSize;
241: procedure QueueSetReceiveMaxSize;
242: procedure AttributeAdd(const Key: WideString; const Value: WideString);
243: procedure AttributeRemove(const Key: WideString);
244: procedure AttributeRead(const Key: WideString; out Value: WideString);
245: procedure AttributeChange(const Key: WideString; const Value: WideString);
246: procedure AttributeSubscriptionStart(const Key: WideString);
247: procedure AttributeSubscriptionEnd(const Key: WideString);
248: procedure LogOut;
249: procedure NodeAdd(const Node: WideString);
250: procedure NodeRemove(const Node: WideString);
251: procedure NodeMediaAdd(const Node: WideString; const MediaType: WideString;
252: const Address: WideString; const Port: WideString);
253: procedure NodeMediaRemove(const Node: WideString; const MediaType:
WideString;
254: const Address: WideString);
255: procedure NodeMediaQuery;
256: procedure NodeConnectedQuery;
257: procedure LogIn(const Node: WideString; const Password: WideString);
258: procedure MediaInventory(out MediaList: OleVariant);
259: procedure MediaStatus(const MediaType: WideString; const MediaAddress:
WideString;
260: out MediaStatus: SYSINT);
261: procedure MediaConnectionAddress(const MediaType: WideString; const Medi-
aAddress: WideString;
262: out ConnectionAddress: WideString);
263: procedure MediaConnect(const MediaType: WideString; const MediaAddress:
WideString;
264: const ConnectionAddress: WideString);
265: procedure MediaDisconnect(const MediaType: WideString; const MediaAddress:
WideString);
266: property DefaultInterface: IHermesControl read GetDefaultInterface;
267: published
268: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
269: property Server: THermesControlProperties read GetServerProperties;
270: {$ENDIF}
271: property OnMessageReceived: TNotifyEvent read FOnMessageReceived write FOn-
MessageReceived;
272: property OnMessageAck: TNotifyEvent read FOnMessageAck write FOnMessa-
geAck;
273: property OnMessageNak: TNotifyEvent read FOnMessageNak write FOnMessa-
geNak;
274: property OnLinkLost: TNotifyEvent read FOnLinkLost write FOnLinkLost;
275: property OnContactLost: TNotifyEvent read FOnContactLost write FOnContact-
Lost;
276: property OnLinkRequest: TNotifyEvent read FOnLinkRequest write FOnLinkRe-
quest;
277: property OnNodeAdd: TNotifyEvent read FOnNodeAdd write FOnNodeAdd;
278: property OnNodeDelete: TNotifyEvent read FOnNodeDelete write FOnNodeDelete;
279: end;
280:
281: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
282: // ************************************************************//
283: // OLE Server Properties Proxy Class
```

```
284: // Server Object : THermesControl
285: // (This object is used by the IDE's Property Inspector to allow editing
286: // of the properties of this server)
287: // *********************************************************************//
288: THermesControlProperties = class(TPersistent)
289: private
290: FServer: THermesControl;
291: function GetDefaultInterface: IHermesControl;
292: constructor Create(AServer: THermesControl);
293: protected
294: public
295: property DefaultInterface: IHermesControl read GetDefaultInterface;
296: published
297: end;
298: {$ENDIF}
299:
300:
301: procedure Register;
302:
303: resourcestring
304: dtlServerPage = 'Hermes';
305:
306: implementation
307:
308: uses ComObj;
309:
310: class function CoHermesControl.Create: IHermesControl;
311: begin
312: Result := CreateComObject(CLASS_HermesControl) as IHermesControl;
313: end;
314:
315: class function CoHermesControl.CreateRemote(const MachineName: string): IHer-
mesControl;
316: begin
317: Result := CreateRemoteComObject(MachineName, CLASS_HermesControl) as IHer-
mesControl;
318: end;
319:
320: procedure THermesControl.InitServerData;
321: const
322: CServerData: TServerData = (
323: ClassID: '{2550F702-C223-4E52-9484-67C5B67D2565}';
324: IntfIID: '{8FA11B77-DB32-4DA1-B44D-0AB8673C17B6}';
325: EventIID: '{DD425DB2-0061-4CD5-AEC6-5A8B4C36CA92}';
326: LicenseKey: nil;
327: Version: 500);
328: begin
329: ServerData := @CServerData;
330: end;
331:
332: procedure THermesControl.Connect;
333: var
334: punk: IUnknown;
335: begin
336: if FIntf = nil then
337: begin
338: punk := GetServer;
339: ConnectEvents(punk);
340: Fintf:= punk as IHermesControl;
341: end;
342: end;
```

```
343:
344: procedure THermesControl.ConnectTo(svrIntf: IHermesControl);
345: begin
346: Disconnect;
347: FIntf := svrIntf;
348: ConnectEvents(FIntf);
349: end;
350:
351: procedure THermesControl.DisConnect;
352: begin
353: if Fintf <> nil then
354: begin
355: DisconnectEvents(FIntf);
356: FIntf := nil;
357: end;
358: end;
359:
360: function THermesControl.GetDefaultInterface: IHermesControl;
361: begin
362: if FIntf = nil then
363: Connect;
364: Assert(FIntf <> nil, 'DefaultInterface is NULL. Component is not connected to Server. You must call "Connect" or "ConnectTo" before this operation');
365: Result := FIntf;
366: end;
367:
368: constructor THermesControl.Create(AOwner: TComponent);
369: begin
370: inherited Create(AOwner);
371: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
372: FProps := THermesControlProperties.Create(Self);
373: {$ENDIF}
374: end;
375:
376: destructor THermesControl.Destroy;
377: begin
378: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
379: FProps.Free;
380: {$ENDIF}
381: inherited Destroy;
382: end;
383:
384: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
385: function THermesControl.GetServerProperties: THermesControlProperties;
386: begin
387: Result := FProps;
388: end;
389: {$ENDIF}
390:
391: procedure THermesControl.InvokeEvent(DispID: TDispID; var Params: TVariantArray);
392: begin
393: case DispID of
394: -1: Exit; // DISPID_UNKNOWN
395: 1: if Assigned(FOnMessageReceived) then
396: FOnMessageReceived(Self);
397: 4: if Assigned(FOnMessageAck) then
398: FOnMessageAck(Self);
399: 5: if Assigned(FOnMessageNak) then
400: FOnMessageNak(Self);
401: 6: if Assigned(FOnLinkLost) then
```

```
402: FOnLinkLost(Self);
403: 7: if Assigned(FOnContactLost) then
404: FOnContactLost(Self);
405: 8: if Assigned(FOnLinkRequest) then
406: FOnLinkRequest(Self);
407: 2: if Assigned(FOnNodeAdd) then
408: FOnNodeAdd(Self);
409: 3: if Assigned(FOnNodeDelete) then
410: FOnNodeDelete(Self);
411: end; {case DispID}
412: end;
413:
414: procedure THermesControl.MessageSend(TheMessage: OleVariant);
415: begin
416: DefaultInterface.MessageSend(TheMessage);
417: end;
418:
419: procedure THermesControl.MessageReceive(out TheMessage: OleVariant; out
Count: Integer);
420: begin
421: DefaultInterface.MessageReceive(TheMessage, Count);
422: end;
423:
424: procedure THermesControl.QOSRequest;
425: begin
426: DefaultInterface.QOSRequest;
427: end;
428:
429: procedure THermesControl.QOSExplore;
430: begin
431: DefaultInterface.QOSExplore;
432: end;
433:
434: procedure THermesControl.QueueGetTransmitSize;
435: begin
436: DefaultInterface.QueueGetTransmitSize;
437: end;
438:
439: procedure THermesControl.QueueGetReceiveSize;
440: begin
441: DefaultInterface.QueueGetReceiveSize;
442: end;
443:
444: procedure THermesControl.QueueGetTransmitMaxSize;
445: begin
446: DefaultInterface.QueueGetTransmitMaxSize;
447: end;
448:
449: procedure THermesControl.QueueGetReceiveMaxSize;
450: begin
451: DefaultInterface.QueueGetReceiveMaxSize;
452: end;
453:
454: procedure THermesControl.QueueSetTransmitMaxSize;
455: begin
456: DefaultInterface.QueueSetTransmitMaxSize;
457: end;
458:
459: procedure THermesControl.QueueSetReceiveMaxSize;
460: begin
461: DefaultInterface.QueueSetReceiveMaxSize;
```

```
462: end;
463:
464: procedure THermesControl.AttributeAdd(const Key: WideString; const Value:
WideString);
465: begin
466: DefaultInterface.AttributeAdd(Key, Value);
467: end;
468:
469: procedure THermesControl.AttributeRemove(const Key: WideString);
470: begin
471: DefaultInterface.AttributeRemove(Key);
472: end;
473:
474: procedure THermesControl.AttributeRead(const Key: WideString; out Value:
WideString);
475: begin
476: DefaultInterface.AttributeRead(Key, Value);
477: end;
478:
479: procedure THermesControl.AttributeChange(const Key: WideString; const Value:
WideString);
480: begin
481: DefaultInterface.AttributeChange(Key, Value);
482: end;
483:
484: procedure THermesControl.AttributeSubscriptionStart(const Key: WideString);
485: begin
486: DefaultInterface.AttributeSubscriptionStart(Key);
487: end;
488:
489: procedure THermesControl.AttributeSubscriptionEnd(const Key: WideString);
490: begin
491: DefaultInterface.AttributeSubscriptionEnd(Key);
492: end;
493:
494: procedure THermesControl.LogOut;
495: begin
496: DefaultInterface.LogOut;
497: end;
498:
499: procedure THermesControl.NodeAdd(const Node: WideString);
500: begin
501: DefaultInterface.NodeAdd(Node);
502: end;
503:
504: procedure THermesControl.NodeRemove(const Node: WideString);
505: begin
506: DefaultInterface.NodeRemove(Node);
507: end;
508:
509: procedure THermesControl.NodeMediaAdd(const Node: WideString; const Media-
Type: WideString;
510: const Address: WideString; const Port: WideString);
511: begin
512: DefaultInterface.NodeMediaAdd(Node, MediaType, Address, Port);
513: end;
514:
515: procedure THermesControl.NodeMediaRemove(const Node: WideString; const
MediaType: WideString;
516: const Address: WideString);
517: begin
```

```
518: DefaultInterface.NodeMediaRemove(Node, MediaType, Address);
519: end;
520:
521: procedure THermesControl.NodeMediaQuery;
522: begin
523: DefaultInterface.NodeMediaQuery;
524: end;
525:
526: procedure THermesControl.NodeConnectedQuery;
527: begin
528: DefaultInterface.NodeConnectedQuery;
529: end;
530:
531: procedure THermesControl.LogIn(const Node: WideString; const Password:
WideString);
532: begin
533: DefaultInterface.LogIn(Node, Password);
534: end;
535:
536: procedure THermesControl.MediaInventory(out MediaList: OleVariant);
537: begin
538: DefaultInterface.MediaInventory(MediaList);
539: end;
540:
541: procedure THermesControl.MediaStatus(const MediaType: WideString; const Medi-
aAddress: WideString;
542: out MediaStatus: SYSINT);
543: begin
544: DefaultInterface.MediaStatus(MediaType, MediaAddress, MediaStatus);
545: end;
546:
547: procedure THermesControl.MediaConnectionAddress(const MediaType:
WideString;
548: const MediaAddress: WideString;
549: out ConnectionAddress: WideString);
550: begin
551: DefaultInterface.MediaConnectionAddress(MediaType, MediaAddress, Connectio-
nAddress);
552: end;
553:
554: procedure THermesControl.MediaConnect(const MediaType: WideString; const
MediaAddress: WideString;
555: const ConnectionAddress: WideString);
556: begin
557: DefaultInterface.MediaConnect(MediaType, MediaAddress, ConnectionAddress);
558: end;
559:
560: procedure THermesControl.MediaDisconnect(const MediaType: WideString; const
MediaAddress: WideString);
561: begin
562: DefaultInterface.MediaDisconnect(MediaType, MediaAddress);
563: end;
564:
565: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
566: constructor THermesControlProperties.Create(AServer: THermesControl);
567: begin
568: inherited Create;
569: FServer := AServer;
570: end;
571:
572: function THermesControlProperties.GetDefaultInterface: IHermesControl;
```

```
573: begin
574: Result := FServer.DefaultInterface;
575: end;
576:
577: {$ENDIF}
578:
579: procedure Register;
580: begin
581: RegisterComponents(dtlServerPage, [THermesControl]);
582: end;
583:
584: end.
```

A.1.5 C:\projects\hermes\HermesMessage.pas

```
1: unit HermesMessage;
2:
3: interface
4:
5: uses
6: ActiveX, HermesTypes, Variants, Contnrs;
7:
8: // A message is an OleVariant to make it easy to pass it
9: // to other components. The OleVariant is wrapped in the
10: // Class TMessage, which provides methods to extract
11: // Information such as sender, and receiver.
12:
13: // An empty message may be created by calling the constructor
14: // Without any arguments.
15:
16:
17: type
18:
19: TMessageData = Array of Byte;
20: TPMessageData = ^TMessageData;
21: THermesMessage = class(TObject)
22: private
23: // Internal structure is an SafeArray with 7 fields as follows
24: // Data[0], the NodeID of the receiver (BSTR)
25: // Data[1], the NodeID of the transmitter (BSTR)
26: // Data[2], priority (Byte)
27: // Data[3], TimeOfCreation (TDateTime)
28: // Data[4], BestBefore (TDateTime)
29: // Data[5], DataSize (Integer)
30: // Data[6], The actual data (SafeArray with <DataSize>
31: // numeber of (Byte).
32:
33: TheMessage: OleVariant;
34:
35: protected
36: public
37: constructor Create;
38: function GetTheMessage: OleVariant;
39: procedure SetTheMessage(NewMessage: OleVariant);
40: procedure SetData(NewData: TMessageData);
41: procedure AddData(NewData: TMessageData);
42: function GetData: TMessageData;
43: function ConsumeData(Size: Integer): TMessageData;
44: procedure SetSender(NewSender: TNodeID);
45: function GetSender: TNodeID;
46: procedure SetReceiver(NewReceiver: TNodeID);
```

```
47: function GetReceiver: TNodeID;
48: procedure SetPriority(NewPriority: TPriority);
49: function GetPriority: TPriority;
50: procedure SetCreateTime(NewTime: TDateTime);
51: function GetCreateTime: TDateTime;
52: procedure SetValidTime(NewTime: TDateTime);
53: function GetValidTime: TDateTime;
54: end;
55:
56: TPHermesMessage = ^THermesMessage;
57:
58: THermesMessageQueue = Class(TObjectQueue)
59: private
60: protected
61: public
62: function Peek: THermesMessage;
63: function Pop: THermesMessage;
64: procedure Push(AMessage: THermesMessage);
65: end;
66:
67:
68: function MessageDataToWideString(TheData: TMessageData): WideString;
69: function WideStringToMessageData(TheString: WideString): TMessageData;
70:
71:
72: implementation
73:
74: function MessageDataToWideString(TheData: TMessageData): WideString;
75: var
76: DataIndex: Integer;
77: begin
78: result := '';
79: for DataIndex := low(TheData) to high(TheData) do
80: begin
81: result := result + WideChar(Chr(TheData[DataIndex]));
82: end;
83: end;
84:
85: function WideStringToMessageData(TheString: WideString): TMessageData;
86: var
87: DataIndex: Integer;
88: StringIndex: Integer;
89: begin
90: SetLength(result, length(TheString));
91: DataIndex := 0;
92: for StringIndex := 1 to length(TheString) do
93: begin
94: result[DataIndex] := Byte(Ord(TheString[StringIndex]));
95: Inc(DataIndex, 1);
96: end;
97: end;
98:
99: function THermesMessageQueue.Peek: THermesMessage;
100: begin
101: Peek := THermesMessage(inherited Peek);
102: end;
103:
104: function THermesMessageQueue.Pop: THermesMessage;
105: begin
106: Pop := THermesMessage(inherited Pop);
107: end;
```

```
108:
109: procedure THermesMessageQueue.Push(AMessage: THermesMessage);
110: begin
111: inherited Push(TObject(AMessage));
112: end;
113:
114:
115: constructor THermesMessage.Create;
116: begin
117: inherited Create;
118: TheMessage := VarArrayCreate([0, 6], varVariant);
119: end;
120:
121: function THermesMessage.GetTheMessage: OleVariant;
122: begin
123: GetTheMessage := TheMessage;
124: end;
125:
126: procedure THermesMessage.SetTheMessage(NewMessage: OleVariant);
127: begin
128: TheMessage := NewMessage;
129: end;
130:
131: procedure THermesMessage.SetData(NewData: TMessageData);
132: begin
133: TheMessage[6] := NewData;
134: end;
135:
136: function THermesMessage.GetData : TMessageData;
137: begin
138: result := TMessageData(TheMessage[6]);
139: end;
140:
141: procedure THermesMessage.AddData(NewData: TMessageData);
142: var
143: Index: Integer;
144: Count: Integer;
145: TheData: TMessageData;
146: begin
147: TheData := TheMessage[6];
148: Index := High(TheData) + 1;
149: SetLength(TheData, Length(TheData) + Length(NewData));
150: For Count := Low(NewData) to High(NewData) do
151: begin
152: TheData[Index] := NewData[Count];
153: Inc(Index, 1);
154: end;
155: TheMessage[6] := TheData;
156: end;
157:
158: function THermesMessage.ConsumeData(Size: Integer): TMessageData;
159: begin
160: end;
161:
162: procedure THermesMessage.SetSender(NewSender: TNodeID);
163: begin
164: TheMessage[1] := NewSender;
165: end;
166:
167: function THermesMessage.GetSender : TNodeID;
168: begin
```

```
169: GetSender := TheMessage[1];
170: end;
171:
172: procedure THermesMessage.SetReceiver(NewReceiver : TNodeID);
173: begin
174: TheMessage[0] := NewReceiver;
175: end;
176:
177: function THermesMessage.GetReceiver : TNodeID;
178: begin
179: GetReceiver := TheMessage[0];
180: end;
181:
182: procedure THermesMessage.SetPriority(NewPriority : TPriority);
183: begin
184: TheMessage[2] := NewPriority;
185: end;
186:
187: function THermesMessage.GetPriority : TPriority;
188: begin
189: GetPriority := TheMessage[2];
190: end;
191:
192: procedure THermesMessage.SetCreateTime(NewTime : TDateTime);
193: begin
194: TheMessage[3] := NewTime;
195: end;
196:
197: function THermesMessage.GetCreateTime : TDateTime;
198: begin
199: GetCreateTime := TheMessage[3];
200: end;
201:
202: procedure THermesMessage.SetValidTime(NewTime : TDateTime);
203: begin
204: TheMessage[4] := NewTime;
205: end;
206:
207: function THermesMessage.GetValidTime : TDateTime;
208: begin
209: GetValidTime := TheMessage[4];
210: end;
211:
212: end.
```

4.0.1 C:\projects\hermes\HermesTypes.pas

```
1: unit HermesTypes;
2:
3: interface
4: type
5: TTransportList = class(TObject);
6: TNodeID = WideString;
7: TPriority = Byte;
8: PNodeID = ^TNodeID;
9: PPriority = ^TPriority;
10: TMediaType = WideString;
11: TMediaAddress = WideString;
12:
13: TPacketArray = Array of Byte;
14: TPPacketArray = ^TPacketArray;
```

```
15: TMediaState = (SUnconfigured, SRunningServer, SRunningClient, SDisconnected,
SConnecting, SDisconnecting);
16:
17:
18: implementation
19:
20: end.
```

A.2 GUI.exe

A.2.1 C:\projects\hermesGUI\GUI.dpr

```
1: program GUI;
2:
3: uses
4: Forms,
5: HermesGUI in 'HermesGUI.pas' {Form1},
6: HermesAttribute in '..\hermes\HermesAttribute.pas',
7: HermesTypes in '..\hermes\HermesTypes.pas',
8: mainform in 'mainform.pas' {Form2},
9: MessageSender in 'MessageSender.pas' {Form3},
10: AttributeList in 'AttributeList.pas' {Form4},
11: NodeAdd in 'NodeAdd.pas' {Form5},
12: HermesMessage in '..\hermes\HermesMessage.pas',
13: GUILog in '..\hermes\GUILog.pas' {FormLog},
14: MediaConnect in 'MediaConnect.pas' {FormConnect};
15:
16: {$R *.res}
17:
18: begin
19: Application.Initialize;
20: Application.CreateForm(TForm1, Form1);
21: Application.CreateForm(TForm2, Form2);
22: Application.CreateForm(TForm3, Form3);
23: Application.CreateForm(TForm4, Form4);
24: Application.CreateForm(TForm5, Form5);
25: Application.CreateForm(TFormLog, FormLog);
26: Application.CreateForm(TFormConnect, FormConnect);
27: Application.Run;
28: end.
```

A.2.2 C:\projects\hermesGUI\AttributeList.pas

```
1: unit AttributeList;
2:
3: interface
4:
5: uses
6: Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
7: Dialogs, StdCtrls, ComCtrls;
8:
9: type
10: TForm4 = class(TForm)
11: GroupBox1: TGroupBox;
12: ListView1: TListView;
13: GroupBox2: TGroupBox;
14: Button1: TButton;
15: Button2: TButton;
16: Button3: TButton;
17: Button4: TButton;
```

```
18: Button5: TButton;
19: procedure Button5Click(Sender: TObject);
20: private
21: { Private declarations }
22: public
23: { Public declarations }
24: end;
25:
26: var
27: Form4: TForm4;
28:
29: implementation
30:
31: uses mainform;
32:
33: {$R *.dfm}
34:
35: procedure TForm4.Button5Click(Sender: TObject);
36: begin
37: Form2.Show;
38: end;
39:
40: end.
```

A.2.3 C:\projects\hermesGUI\AttributeList.dfm

```
1: object Form4: TForm4
2: Left = 605
3: Top = 185
4: Width = 627
5: Height = 561
6: Caption = 'Egenskaper i andra noder'
7: Color = clBtnFace
8: Font.Charset = DEFAULT_CHARSET
9: Font.Color = clWindowText
10: Font.Height = -11
11: Font.Name = 'MS Sans Serif'
12: Font.Style = []
13: OldCreateOrder = False
14: PixelsPerInch = 96
15: TextHeight = 13
16: object GroupBox1: TGroupBox
17: Left = 0
18: Top = 0
19: Width = 617
20: Height = 361
21: Caption = 'Prenumeration aktiv p'#229' f'#246'ljande egenskaper'
22: TabOrder = 0
23: object ListView1: TListView
24: Left = 2
25: Top = 15
26: Width = 613
27: Height = 344
28: Align = alClient
29: Columns = <
30: item
31: Caption = 'Nod'
32: end
33: item
34: Caption = 'Egenskap'
35: end
```

```
36: item
37: Caption = 'V'#228'rde'
38: end
39: item
40: Caption = 'Uppdateringsintervall'
41: end
42: item
43: Caption = 'Senast uppdaterad'
44: end>
45: SortType = stText
46: TabOrder = 0
47: ViewStyle = vsReport
48: end
49: end
50: object GroupBox2: TGroupBox
51: Left = 0
52: Top = 368
53: Width = 617
54: Height = 161
55: TabOrder = 1
56: object Button1: TButton
57: Left = 16
58: Top = 16
59: Width = 145
60: Height = 25
61: Caption = 'Ny prenumeration'
62: TabOrder = 0
63: end
64: object Button2: TButton
65: Left = 16
66: Top = 48
67: Width = 145
68: Height = 25
69: Caption = 'Avsluta prenumeration'
70: Enabled = False
71: TabOrder = 1
72: end
73: object Button3: TButton
74: Left = 16
75: Top = 80
76: Width = 145
77: Height = 25
78: Caption = #196'ndra uppdateringsintervall'
79: Enabled = False
80: TabOrder = 2
81: end
82: object Button4: TButton
83: Left = 16
84: Top = 112
85: Width = 145
86: Height = 25
87: Caption = 'Beg'#228'r uppdatering'
88: Enabled = False
89: TabOrder = 3
90: end
91: object Button5: TButton
92: Left = 424
93: Top = 112
94: Width = 161
95: Height = 25
96: Caption = 'Visa lokal information'
```

```
97: TabOrder = 4
98: OnClick = Button5Click
99: end
100: end
101: end
```

A.2.4 C:\projects\hermes\GUILog.pas

```
1: unit GUILog;
2:
3: interface
4:
5: uses
6: Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
7: Dialogs, StdCtrls;
8:
9: type
10: TFormLog = class(TForm)
11: Memo1: TMemo;
12: private
13: { Private declarations }
14: public
15: { Public declarations }
16: end;
17:
18: var
19: FormLog: TFormLog;
20:
21: implementation
22:
23: {$R *.dfm}
24:
25: end.
```

A.2.5 C:\projects\hermes\GUILog.dfm

```
1: object FormLog: TFormLog
2: Left = 0
3: Top = 843
4: Width = 183
5: Height = 150
6: Caption = 'GUI Log'
7: Color = clBtnFace
8: Font.Charset = DEFAULT_CHARSET
9: Font.Color = clWindowText
10: Font.Height = -11
11: Font.Name = 'MS Sans Serif'
12: Font.Style = []
13: OldCreateOrder = False
14: PixelsPerInch = 96
15: TextHeight = 13
16: object Memo1: TMemo
17: Left = 0
18: Top = 0
19: Width = 175
20: Height = 123
21: Align = alClient
22: TabOrder = 0
23: end
24: end
```

A.2.6 C:\projects\hermes\HermesAttribute.pas

Se A.1.3 C:\projects\hermes\HermesAttribute.pas på sidan 54

A.2.7 C:\projects\hermesGUI\HermesGUI.pas

```
1: unit HermesGUI;
2:
3: interface
4:
5: uses
6: Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
7: HermesTypes, Dialogs, StdCtrls, HermesLib_TLB, OleServer,
8: HermesAttribute, Menus, Grids, PEngine_TLB;
9:
10: type
11: TForm1 = class(TForm)
12: Edit1: TEdit;
13: Label1: TLabel;
14: Button1: TButton;
15: HermesController: THermesControl;
16: procedure Button1Click(Sender: TObject);
17: procedure FormCreate(Sender: TObject);
18: procedure HermesControllerMessageReceived(Sender: TObject);
19: procedure HermesControllerContactLost(Sender: TObject);
20: private
21: { Private declarations }
22: FAttributes: TAttributeList;
23: public
24: { Public declarations }
25: //HermesController: IHermesControl;
26: end;
27:
28: var
29: NodeID: TNodeID;
30: Form1: TForm1;
31:
32: implementation
33:
34: uses
35: HermesMessage, mainform, GUILog, ComCtrls, MediaConnect;
36:
37:
38: {$R *.dfm}
39:
40: procedure TForm1.Button1Click(Sender: TObject);
41: begin
42: if Assigned(HermesController) then begin
43: Button1.Enabled := False;
44: NodeID := Edit1.Text;
45: HermesController.LogIn(WideString(NodeID), WideString('NoPassword'));
46: Edit1.ReadOnly := True;
47: Form2.Show;
48: Form2.Caption := 'HERMES-' + NodeID + ' online';
49: Form2.Button1.Enabled := True;
50: Form1.Hide;
51: FormLog.Show;
52: HermesController.MediaInventory(FormConnect.MediaList);
53: FormConnect.UpdateMediaList;
54: FormConnect.Show;
```

```
55: end
56: else
57: ShowMessage('Error Controller not started');
58: end;
59:
60: procedure TForm1.FormCreate(Sender: TObject);
61: begin
62: FAttributes := TAttributeList.create;
63: // if not Assigned(HermesController) then
64: // HermesController := CoHermesControl.Create;
65: end;
66:
67: procedure TForm1.HermesControllerMessageReceived(Sender: TObject);
68: var
69: MessageData: OleVariant;
70: MessageCount: Integer;
71: AMessage: THermesMessage;
72: begin
73: FormLog.Memo1.Lines.Add('Message received');
74: HermesController.MessageReceive(MessageData, MessageCount);
75: while MessageCount > 0 do begin
76: AMessage := THermesMessage.Create;
77: AMessage.SetTheMessage(MessageData);
78: Form2.AddMessage(AMessage);
79: HermesController.MessageReceive(MessageData, MessageCount);
80: end;
81: end;
82:
83: procedure TForm1.HermesControllerContactLost(Sender: TObject);
84: begin
85: //ShowMessage('Contact lost');
86: FormLog.Memo1.Lines.Add('Contact lost');
87: end;
88:
89: end.
```

A.2.8 C:\projects\hermesGUI\HermesGUI.dfm

```
1: object Form1: TForm1
2: Left = 356
3: Top = 149
4: Width = 346
5: Height = 73
6: Caption = 'HERMES-offline'
7: Color = clBtnFace
8: Font.Charset = DEFAULT_CHARSET
9: Font.Color = clWindowText
10: Font.Height = -13
11: Font.Name = 'MS Sans Serif'
12: Font.Style = []
13: OldCreateOrder = False
14: OnCreate = FormCreate
15: PixelsPerInch = 96
16: TextHeight = 16
17: object Label1: TLabel
18: Left = 0
19: Top = 0
20: Width = 46
21: Height = 13
22: Caption = 'Nodnamn'
23: Font.Charset = DEFAULT_CHARSET
```

```
24: Font.Color = clWindowText
25: Font.Height = -11
26: Font.Name = 'MS Sans Serif'
27: Font.Style = []
28: ParentFont = False
29: end
30: object Edit1: TEdit
31: Left = 56
32: Top = 0
33: Width = 177
34: Height = 21
35: Font.Charset = DEFAULT_CHARSET
36: Font.Color = clWindowText
37: Font.Height = -11
38: Font.Name = 'MS Sans Serif'
39: Font.Style = []
40: ParentFont = False
41: TabOrder = 0
42: Text = 'Edit1'
43: end
44: object Button1: TButton
45: Left = 240
46: Top = 0
47: Width = 89
48: Height = 25
49: Caption = 'LogIn'
50: Default = True
51: Font.Charset = DEFAULT_CHARSET
52: Font.Color = clWindowText
53: Font.Height = -11
54: Font.Name = 'MS Sans Serif'
55: Font.Style = []
56: ParentFont = False
57: TabOrder = 1
58: OnClick = Button1Click
59: end
60: object HermesController: THermesControl
61: AutoConnect = True
62: ConnectKind = ckRunningOrNew
63: OnMessageReceived = HermesControllerMessageReceived
64: OnContactLost = HermesControllerContactLost
65: Top = 16
66: end
67: end
```

A.2.9 C:\projects\hermes\HermesMessage.pas

Se A.1.5 C:\projects\hermes\HermesMessage.pas på sidan 67.

A.2.10 C:\projects\hermes\HermesTypes.pas

```
1: unit HermesTypes;
2:
3: interface
4: type
5: TTransportList = class(TObject);
6: TNodeID = WideString;
7: TPriority = Byte;
8: PNodeID = ^TNodeID;
9: PPriority = ^TPriority;
```

```
10: TMediaType = WideString;
11: TMediaAddress = WideString;
12:
13: TPacketArray = Array of Byte;
14: TPPacketArray = ^TPacketArray;
15: TMediaState = (SUnconfigured, SRunningServer, SRunningClient, SDisconnected,
SConnecting, SDisconnecting);
16:
17:
18: implementation
19:
20: end.
```

A.2.11 C:\projects\hermesGUI\mainform.pas

```
1: unit mainform;
2:
3: interface
4:
5: uses
6: HermesMessage, Menus, StdCtrls, ComCtrls, Controls, Grids, ValEdit,
7: Windows, Messages, SysUtils, Variants, Classes, Graphics, Forms,
8: Dialogs;
9:
10: type
11: TForm2 = class(TForm)
12: GroupBox1: TGroupBox;
13: GroupBox2: TGroupBox;
14: GroupBox3: TGroupBox;
15: ValueListEditor1: TValueListEditor;
16: ListView1: TListView;
17: StatusBar1: TStatusBar;
18: MainMenu1: TMainMenu;
19: Memo1: TMemo;
20: File1: TMenuItem;
21: Exit1: TMenuItem;
22: N1: TMenuItem;
23: PrintSetup1: TMenuItem;
24: Print1: TMenuItem;
25: N2: TMenuItem;
26: SaveAs1: TMenuItem;
27: Save1: TMenuItem;
28: Open1: TMenuItem;
29: New1: TMenuItem;
30: Edit1: TMenuItem;
31: Object1: TMenuItem;
32: Links1: TMenuItem;
33: N3: TMenuItem;
34: GoTo1: TMenuItem;
35: Replace1: TMenuItem;
36: Find1: TMenuItem;
37: N4: TMenuItem;
38: PasteSpecial1: TMenuItem;
39: Paste1: TMenuItem;
40: Copy1: TMenuItem;
41: Cut1: TMenuItem;
42: N5: TMenuItem;
43: Repeatcommand1: TMenuItem;
44: Undo1: TMenuItem;
45: Help1: TMenuItem;
46: About1: TMenuItem;
```

```
47: HowtoUseHelp1: TMenuItem;
48: SearchforHelpOn1: TMenuItem;
49: Contents1: TMenuItem;
50: Nod1: TMenuItem;
51: Lggtill1: TMenuItem;
52: abort1: TMenuItem;
53: Anslutningar1: TMenuItem;
54: Lggtill2: TMenuItem;
55: abort2: TMenuItem;
56: ndra1: TMenuItem;
57: ndra2: TMenuItem;
58: Button1: TButton;
59: GroupBox4: TGroupBox;
60: GroupBox5: TGroupBox;
61: ButtonSendMessage: TButton;
62: Button3: TButton;
63: Button4: TButton;
64: procedure Button1Click(Sender: TObject);
65: procedure FormClose(Sender: TObject; var Action: TCloseAction);
66: procedure ButtonSendMessageClick(Sender: TObject);
67: procedure Button4Click(Sender: TObject);
68: procedure ListView1SelectItem(Sender: TObject; Item: TListItem;
69: Selected: Boolean);
70: private
71: { Private declarations }
72: public
73: Procedure AddMessage(TheMessage: THermesMessage);
74: { Public declarations }
75: end;
76:
77: var
78: Form2: TForm2;
79:
80: implementation
81:
82: uses HermesGUI, MessageSender, AttributeList;
83:
84: {$R *.dfm}
85:
86:
87: Procedure TForm2.AddMessage(TheMessage: THermesMessage);
88: var
89: NewItem: TListItem;
90: SystemTime: TSystemTime;
91: Receiver: WideString;
92: begin
93: GetLocalTime(SystemTime);
94: NewItem := Form2.ListView1.Items.Add;
95: Form2.Memo1.Text := MessageDataToWideString(TheMessage.GetData);
96: NewItem.Caption := DateTimeToStr(TheMessage.GetCreateTime);
97: NewItem.SubItems.Add(TheMessage.GetSender);
98: NewItem.SubItems.Add(IntToStr(TheMessage.GetPriority));
99: Receiver := TheMessage.GetReceiver;
100: if Receiver = 'Alla' then begin
101: NewItem.SubItems.Add('Allmänt');
102: end else begin
103: NewItem.SubItems.Add('Privat');
104: end;
105: NewItem.SubItems.Add(DateTimeToStr(SystemTimeToDateTime(SystemTime)));
106: NewItem.SubItems.Add(DateTimeToStr(TheMessage.GetValidTime));
107: NewItem.SubItems.Add(MessageDataToWideString(TheMessage.GetData));
```

```
108: end;
109:
110:
111: procedure TForm2.Button1Click(Sender: TObject);
112: begin
113: Form2.Button1.Enabled := False;
114: Form1.HermesController.LogOut;
115: Form1.Caption := 'HERMES-offline';
116: Form1.Edit1.ReadOnly := False;
117: Form1.Button1.Enabled := True;
118: Form2.Hide;
119: Form1.Show;
120: end;
121:
122: procedure TForm2.FormClose(Sender: TObject; var Action: TCloseAction);
123: begin
124: Button1.Click;
125: end;
126:
127: procedure TForm2.ButtonSendMessageClick(Sender: TObject);
128: begin
129: Form2.ButtonSendMessage.Enabled := False;
130: Form3.Show;
131: end;
132:
133: procedure TForm2.Button4Click(Sender: TObject);
134: begin
135: Form4.Show;
136: end;
137:
138: procedure TForm2.ListView1SelectItem(Sender: TObject; Item: TListItem;
139: Selected: Boolean);
140: begin
141: Form2.Memo1.Text := Item.SubItems.Strings[5];
142: end;
143:
144: end.
```

A.2.12 C:\projects\hermesGUI\mainform.dfm

```
1: object Form2: TForm2
2: Left = 269
3: Top = 176
4: Width = 866
5: Height = 524
6: Caption = 'HERMES - Nodnamn'
7: Color = clBtnFace
8: Font.Charset = DEFAULT_CHARSET
9: Font.Color = clWindowText
10: Font.Height = -11
11: Font.Name = 'MS Sans Serif'
12: Font.Style = []
13: Menu = MainMenu1
14: OldCreateOrder = False
15: OnClose = FormClose
16: PixelsPerInch = 96
17: TextHeight = 13
18: object GroupBox1: TGroupBox
19: Left = 0
20: Top = 0
21: Width = 369
```

```
22: Height = 249
23: Caption = 'Egenskaper'
24: TabOrder = 0
25: object ValueListEditor1: TValueListEditor
26: Left = 2
27: Top = 15
28: Width = 365
29: Height = 232
30: Align = alClient
31: TabOrder = 0
32: TitleCaptions.Strings = (
33: 'Namn'
34: 'V'#228'rde')
35: ColWidths = (
36: 86
37: 273)
38: end
39: end
40: object GroupBox2: TGroupBox
41: Left = 376
42: Top = 0
43: Width = 481
44: Height = 249
45: Caption = 'Meddelanden'
46: TabOrder = 1
47: object ListView1: TListView
48: Left = 2
49: Top = 15
50: Width = 477
51: Height = 232
52: Columns = <
53: item
54: Caption = 'Avs'#228'nd tid'
55: MinWidth = 40
56: Width = 80
57: end
58: item
59: Caption = 'Avs'#228'ndare'
60: MinWidth = 40
61: Width = 80
62: end
63: item
64: Caption = 'Prio'
65: MinWidth = 20
66: Width = 40
67: end
68: item
69: Caption = 'Klass'
70: MinWidth = 20
71: Width = 40
72: end
73: item
74: Caption = 'Mottagen tid'
75: MinWidth = 40
76: Width = 80
77: end
78: item
79: Caption = 'B'#228'st f'#246're'
80: MinWidth = 40
81: Width = 80
82: end
```

```
83: item
84: Caption = 'Text'
85: MinWidth = 10
86: Width = 400
87: end>
88: DragMode = dmAutomatic
89: GridLines = True
90: HideSelection = False
91: ReadOnly = True
92: RowSelect = True
93: ShowWorkAreas = True
94: SortType = stText
95: TabOrder = 0
96: ViewStyle = vsReport
97: OnSelectItem = ListView1SelectItem
98: end
99: end
100: object GroupBox3: TGroupBox
101: Left = 376
102: Top = 256
103: Width = 481
104: Height = 201
105: Caption = 'Meddelandetext'
106: TabOrder = 2
107: object Memo1: TMemo
108: Left = 2
109: Top = 15
110: Width = 477
111: Height = 184
112: Align = alClient
113: Lines.Strings = (
114: 'Memo1')
115: TabOrder = 0
116: end
117: end
118: object StatusBar1: TStatusBar
119: Left = 0
120: Top = 459
121: Width = 858
122: Height = 19
123: Color = clGradientActiveCaption
124: Panels = <
125: item
126: Text = 'Panel1'
127: Width = 50
128: end
129: item
130: Text = 'Panel2'
131: Width = 50
132: end>
133: SimplePanel = False
134: end
135: object Button1: TButton
136: Left = 8
137: Top = 432
138: Width = 97
139: Height = 25
140: Caption = 'Koppla ner'
141: TabOrder = 4
142: OnClick = Button1Click
143: end
```

```
144: object GroupBox4: TGroupBox
145: Left = 184
146: Top = 264
147: Width = 177
148: Height = 153
149: Caption = 'Egenskaper'
150: TabOrder = 5
151: object Button3: TButton
152: Left = 16
153: Top = 24
154: Width = 121
155: Height = 25
156: Caption = 'L'#228'gg till'
157: TabOrder = 0
158: end
159: object Button4: TButton
160: Left = 16
161: Top = 120
162: Width = 121
163: Height = 25
164: Caption = 'Visa prenumererade'
165: TabOrder = 1
166: OnClick = Button4Click
167: end
168: end
169: object GroupBox5: TGroupBox
170: Left = 8
171: Top = 264
172: Width = 161
173: Height = 153
174: Caption = 'Meddelanden'
175: TabOrder = 6
176: object ButtonSendMessage: TButton
177: Left = 8
178: Top = 24
179: Width = 57
180: Height = 25
181: Caption = 'Skicka'
182: TabOrder = 0
183: OnClick = ButtonSendMessageClick
184: end
185: end
186: object MainMenu1: TMainMenu
187: Left = 424
188: Top = 80
189: object File1: TMenuItem
190: Caption = '&File'
191: object New1: TMenuItem
192: Caption = '&New'
193: end
194: object Open1: TMenuItem
195: Caption = '&Open...'
196: end
197: object Save1: TMenuItem
198: Caption = '&Save'
199: end
200: object SaveAs1: TMenuItem
201: Caption = 'Save &As...'
202: end
203: object N2: TMenuItem
204: Caption = '-'
```

```
205: end
206: object Print1: TMenuItem
207: Caption = '&Print...'
208: end
209: object PrintSetup1: TMenuItem
210: Caption = 'P&rint Setup...'
211: end
212: object N1: TMenuItem
213: Caption = '-'
214: end
215: object Exit1: TMenuItem
216: Caption = 'E&xit'
217: end
218: end
219: object Edit1: TMenuItem
220: Caption = '&Edit'
221: object Undo1: TMenuItem
222: Caption = '&Undo'
223: ShortCut = 16474
224: end
225: object Repeatcommand1: TMenuItem
226: Caption = '&Repeat <command>'
227: end
228: object N5: TMenuItem
229: Caption = '-'
230: end
231: object Cut1: TMenuItem
232: Caption = 'Cu&t'
233: ShortCut = 16472
234: end
235: object Copy1: TMenuItem
236: Caption = '&Copy'
237: ShortCut = 16451
238: end
239: object Paste1: TMenuItem
240: Caption = '&Paste'
241: ShortCut = 16470
242: end
243: object PasteSpecial1: TMenuItem
244: Caption = 'Paste &Special...'
245: end
246: object N4: TMenuItem
247: Caption = '-'
248: end
249: object Find1: TMenuItem
250: Caption = '&Find...'
251: end
252: object Replace1: TMenuItem
253: Caption = 'R&eplace...'
254: end
255: object GoTo1: TMenuItem
256: Caption = '&Go To...'
257: end
258: object N3: TMenuItem
259: Caption = '-'
260: end
261: object Links1: TMenuItem
262: Caption = 'Lin&ks...'
263: end
264: object Object1: TMenuItem
265: Caption = '&Object'
```

```
266: end
267: end
268: object Nod1: TMenuItem
269: Caption = 'Enhet'
270: object Lggtill1: TMenuItem
271: Caption = 'L'#228'gg till'
272: end
273: object abort1: TMenuItem
274: Caption = 'Ta bort'
275: end
276: object ndra2: TMenuItem
277: Caption = #196'ndra'
278: end
279: end
280: object Anslutningar1: TMenuItem
281: Caption = 'Anslutningar'
282: object Lggtill2: TMenuItem
283: Caption = 'L'#228'gg till'
284: end
285: object abort2: TMenuItem
286: Caption = 'Ta bort'
287: end
288: object ndra1: TMenuItem
289: Caption = #196'ndra'
290: end
291: end
292: object Help1: TMenuItem
293: Caption = '&Help'
294: object Contents1: TMenuItem
295: Caption = '&Contents'
296: end
297: object SearchforHelpOn1: TMenuItem
298: Caption = '&Search for Help On...'
299: end
300: object HowtoUseHelp1: TMenuItem
301: Caption = '&How to Use Help'
302: end
303: object About1: TMenuItem
304: Caption = '&About...'
305: end
306: end
307: end
308: end
```

A.2.13 C:\projects\hermesGUI\MediaConnect.pas

```
1: unit MediaConnect;
2:
3: interface
4:
5: uses
6: Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
7: Dialogs, ComCtrls, StdCtrls;
8:
9: type
10: TFormConnect = class(TForm)
11: ListView1: TListView;
12: ButtonAdd: TButton;
13: ButtonRemove: TButton;
14: ButtonDisconnect: TButton;
15: ButtonConnect: TButton;
```

```
16: EditConnect: TEdit;
17: procedure ListView1SelectItem(Sender: TObject; Item: TListItem;
18: Selected: Boolean);
19: procedure ButtonConnectClick(Sender: TObject);
20: private
21: { Private declarations }
22: public
23: { Public declarations }
24: MediaList: OleVariant;
25: EditItem: TListItem;
26: procedure UpdateMediaList;
27: end;
28:
29: var
30: FormConnect: TFormConnect;
31:
32: implementation
33:
34: uses
35: GUILog, HermesTypes, HermesGUI;
36:
37: {$R *.dfm}
38:
39: function StringHead(AString: AnsiString): AnsiString;
40: var
41: breakpoint: Integer;
42: begin
43: breakpoint := Pos('=', AString);
44: result := Copy(AString, 0, breakpoint - 1);
45: end;
46:
47: function StringTail(AString: AnsiString): AnsiString;
48: var
49: breakpoint: Integer;
50: begin
51: breakpoint := Pos('=', AString);
52: result := Copy(AString, breakpoint + 1, 65536);
53: end;
54:
55:
56: procedure TFormConnect.UpdateMediaList;
57: var
58: Index: Integer;
59: NewItem: TListItem;
60: MediaType: TMediaType;
61: MediaAddress: TMediaAddress;
62: MediaStatus: Integer;
63: ConnectionAddress: WideString;
64: StatusString: WideString;
65: begin
66: ListView1.Items.Clear;
67: for Index := VarArrayLowBound(MediaList, 1) to VarArrayHighBound(MediaList, 1)
do
68: begin
69: NewItem := ListView1.Items.Add;
70: MediaType := StringHead(MediaList[Index]);
71: MediaAddress := StringTail(MediaList[Index]);
72: NewItem.Caption := MediaType;
73: NewItem.SubItems.Add(MediaAddress);
74: Form1.HermesController.MediaConnectionAddress(MediaType, MediaAddress,
ConnectionAddress);
```

```
75: NewItem.SubItems.Add(ConnectionAddress);
76: Form1.HermesController.MediaStatus(MediaType, MediaAddress, MediaStatus);
77: Str(MediaStatus, StatusString);
78: NewItem.SubItems.Add(StatusString);
79: end;
80: end;
81:
82:
83: procedure TFormConnect.ListView1SelectItem(Sender: TObject;
84: Item: TListItem; Selected: Boolean);
85: begin
86: FormLog.Memo1.Lines.Add('OnSelsectItem');
87: EditItem := Item;
88: EditConnect.Text := Item.SubItems.Strings[1];
89: if Item.SubItems.Strings[1] = 'Not connected' then
90: begin
91: ButtonConnect.Enabled := True;
92: end
93: else
94: begin
95: ButtonConnect.Enabled := False;
96: end;
97: end;
98:
99: procedure TFormConnect.ButtonConnectClick(Sender: TObject);
100: begin
101: Form1.HermesController.MediaConnect(EditItem.Caption, EditItem.SubI-
tems.Strings[0], EditConnect.Text);
102: end;
103:
104: end.
```

A.2.14 C:\projects\hermesGUI\MediaConnect.dfm

```
1: object FormConnect: TFormConnect
2: Left = 440
3: Top = 492
4: Width = 519
5: Height = 153
6: Caption = 'Media Connections'
7: Color = clBtnFace
8: Font.Charset = DEFAULT_CHARSET
9: Font.Color = clWindowText
10: Font.Height = -11
11: Font.Name = 'MS Sans Serif'
12: Font.Style = []
13: OldCreateOrder = False
14: PixelsPerInch = 96
15: TextHeight = 13
16: object ListView1: TListView
17: Left = 0
18: Top = 21
19: Width = 511
20: Height = 105
21: Align = alBottom
22: Columns = <
23: item
24: AutoSize = True
25: Caption = 'Mediatyp'
26: MinWidth = 90
27: end
```

```
28: item
29: AutoSize = True
30: Caption = 'Egen adress'
31: MinWidth = 90
32: end
33: item
34: AutoSize = True
35: Caption = 'Ansluten adress'
36: MinWidth = 90
37: end
38: item
39: Caption = 'Status'
40: end>
41: ReadOnly = True
42: RowSelect = True
43: TabOrder = 0
44: ViewStyle = vsReport
45: OnSelectItem = ListView1SelectItem
46: end
47: object ButtonAdd: TButton
48: Left = 440
49: Top = 0
50: Width = 65
51: Height = 17
52: Caption = 'L'#228'gg till'
53: Enabled = False
54: TabOrder = 1
55: end
56: object ButtonRemove: TButton
57: Left = 368
58: Top = 0
59: Width = 65
60: Height = 17
61: Caption = 'Ta bort'
62: Enabled = False
63: TabOrder = 2
64: end
65: object ButtonDisconnect: TButton
66: Left = 0
67: Top = 0
68: Width = 65
69: Height = 17
70: Caption = 'Koppla ner'
71: Enabled = False
72: TabOrder = 3
73: end
74: object ButtonConnect: TButton
75: Left = 80
76: Top = 0
77: Width = 65
78: Height = 17
79: Caption = 'Anslut'
80: Enabled = False
81: TabOrder = 4
82: OnClick = ButtonConnectClick
83: end
84: object EditConnect: TEdit
85: Left = 152
86: Top = 0
87: Width = 201
88: Height = 21
```

```
89: TabOrder = 5
90: end
91: end
```

A.2.15 C:\projects\hermesGUI\MessageSender.pas

```
1: unit MessageSender;
2:
3: interface
4:
5: uses
6: Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
7: HermesMessage, Dialogs, ExtCtrls, StdCtrls;
8:
9: type
10: TForm3 = class(TForm)
11: GroupBox1: TGroupBox;
12: GroupBox2: TGroupBox;
13: ButtonMessageSend: TButton;
14: Button2: TButton;
15: Memo1: TMemo;
16: Label1: TLabel;
17: ComboBox1: TComboBox;
18: RadioGroup1: TRadioGroup;
19: RadioGroup2: TRadioGroup;
20: procedure ButtonMessageSendClick(Sender: TObject);
21: procedure Button2Click(Sender: TObject);
22: procedure FormClose(Sender: TObject; var Action: TCloseAction);
23: private
24: { Private declarations }
25: public
26: { Public declarations }
27: end;
28:
29: var
30: Form3: TForm3;
31:
32: implementation
33:
34: uses mainform, HermesGUI;
35:
36: {$R *.dfm}
37:
38: procedure TForm3.ButtonMessageSendClick(Sender: TObject);
39: var
40: NewMessage: THermesMessage;
41: SystemTime: TSystemTime;
42: DateTime: TDateTime;
43: begin
44: Form3.Hide;
45: Form2.ButtonSendMessage.Enabled := True;
46:
47: NewMessage := THermesMessage.Create;
48: NewMessage.SetSender(NodeID);
49: if RadioGroup2.ItemIndex = 0 then begin
50: NewMessage.SetReceiver(Form3.ComboBox1.Text);
51: end else begin
52: NewMessage.SetReceiver('Alla');
53: end;
54: GetLocalTime(SystemTime);
55: DateTime := SystemTimeToDateTime(SystemTime);
```

```
56: NewMessage.SetCreateTime(DateTime);
57: NewMessage.SetValidTime(DateTime + 1.0);
58: NewMessage.SetPriority(RadioGroup1.ItemIndex);
59: NewMessage.SetData(WideStringToMessageData(Form3.Memo1.Text));
60: Form1.HermesController.MessageSend(NewMessage.GetTheMessage);
61:
62: end;
63:
64: procedure TForm3.Button2Click(Sender: TObject);
65: begin
66: Form3.Hide;
67: Form2.ButtonSendMessage.Enabled := True;
68: end;
69:
70: procedure TForm3.FormClose(Sender: TObject; var Action: TCloseAction);
71: begin
72: Form3.Button2.Click;
73: end;
74:
75: end.
```

A.2.16 C:\projects\hermesGUI\MessageSender.dfm

```
1: object Form3: TForm3
2: Left = 870
3: Top = 111
4: Width = 406
5: Height = 359
6: Caption = 'Form3'
7: Color = clBtnFace
8: Font.Charset = DEFAULT_CHARSET
9: Font.Color = clWindowText
10: Font.Height = -11
11: Font.Name = 'MS Sans Serif'
12: Font.Style = []
13: OldCreateOrder = False
14: OnClose = FormClose
15: PixelsPerInch = 96
16: TextHeight = 13
17: object GroupBox1: TGroupBox
18: Left = 0
19: Top = 0
20: Width = 393
21: Height = 113
22: Caption = 'Egenskaper'
23: TabOrder = 0
24: object Label1: TLabel
25: Left = 8
26: Top = 24
27: Width = 48
28: Height = 13
29: Caption = 'Mottagare'
30: end
31: object ComboBox1: TComboBox
32: Left = 72
33: Top = 24
34: Width = 145
35: Height = 21
36: ItemHeight = 13
37: TabOrder = 0
38: Text = 'ComboBox1'
```

```
39: end
40: object RadioGroup1: TRadioGroup
41: Left = 304
42: Top = 16
43: Width = 65
44: Height = 89
45: Caption = 'Prioritet'
46: ItemIndex = 1
47: Items.Strings = (
48: 'H'#246'g'
49: 'L'#229'g')
50: TabOrder = 1
51: end
52: object RadioGroup2: TRadioGroup
53: Left = 224
54: Top = 16
55: Width = 65
56: Height = 89
57: Caption = 'Klass'
58: ItemIndex = 0
59: Items.Strings = (
60: 'Privat'
61: 'Publikt')
62: TabOrder = 2
63: end
64: end
65: object GroupBox2: TGroupBox
66: Left = 0
67: Top = 120
68: Width = 393
69: Height = 177
70: Caption = 'Meddelandetext'
71: TabOrder = 1
72: object Memo1: TMemo
73: Left = 2
74: Top = 15
75: Width = 389
76: Height = 160
77: Align = alClient
78: Lines.Strings = (
79: 'Memo1')
80: TabOrder = 0
81: end
82: end
83: object ButtonMessageSend: TButton
84: Left = 232
85: Top = 304
86: Width = 161
87: Height = 25
88: Caption = 'Skicka'
89: TabOrder = 2
90: OnClick = ButtonMessageSendClick
91: end
92: object Button2: TButton
93: Left = 0
94: Top = 304
95: Width = 89
96: Height = 25
97: Caption = 'Kassera'
98: TabOrder = 3
99: OnClick = Button2Click
```

```
100: end
101: end
```

A.2.17 C:\projects\hermesGUI\NodeAdd.pas

```
1: unit NodeAdd;
2:
3: interface
4:
5: uses
6: Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
7: Dialogs, StdCtrls, Grids, ValEdit, ExtCtrls;
8:
9: type
10: TForm5 = class(TForm)
11: Enhetsbeteckning: TLabeledEdit;
12: GroupBox1: TGroupBox;
13: ValueListEditor1: TValueListEditor;
14: Button1: TButton;
15: Button2: TButton;
16: private
17: { Private declarations }
18: public
19: { Public declarations }
20: end;
21:
22: var
23: Form5: TForm5;
24:
25: implementation
26:
27: {$R *.dfm}
28:
29: end.
```

A.2.18 C:\projects\hermesGUI\NodeAdd.dfm

```
1: object Form5: TForm5
2: Left = 339
3: Top = 260
4: Width = 495
5: Height = 274
6: Caption = 'L'#228'gg till en enhet'
7: Color = clBtnFace
8: Font.Charset = DEFAULT_CHARSET
9: Font.Color = clWindowText
10: Font.Height = -11
11: Font.Name = 'MS Sans Serif'
12: Font.Style = []
13: OldCreateOrder = False
14: PixelsPerInch = 96
15: TextHeight = 13
16: object Enhetsbeteckning: TLabeledEdit
17: Left = 96
18: Top = 8
19: Width = 169
20: Height = 21
21: EditLabel.Width = 86
22: EditLabel.Height = 13
23: EditLabel.Caption = 'Enhetsbeteckning'
24: LabelPosition = lpLeft
```

```
25: LabelSpacing = 3
26: TabOrder = 0
27: end
28: object GroupBox1: TGroupBox
29: Left = 0
30: Top = 40
31: Width = 481
32: Height = 201
33: Caption = 'Kommunikationss'#228'tt'
34: TabOrder = 1
35: object ValueListEditor1: TValueListEditor
36: Left = 2
37: Top = 15
38: Width = 477
39: Height = 184
40: Align = alClient
41: TabOrder = 0
42: TitleCaptions.Strings = (
43: 'Kommunikationss'#228'tt'
44: 'Adress')
45: ColWidths = (
46: 150
47: 321)
48: end
49: end
50: object Button1: TButton
51: Left = 280
52: Top = 8
53: Width = 89
54: Height = 25
55: Caption = 'L'#228'gg till'
56: TabOrder = 2
57: end
58: object Button2: TButton
59: Left = 376
60: Top = 8
61: Width = 89
62: Height = 25
63: Caption = 'Avbryt'
64: TabOrder = 3
65: end
66: end
```

A.3 PEngine.exe

A.3.1 C:\projects\hermes\PEngine.dpr

```
1: program PEngine;
2:
3: uses
4: Forms,
5: Inifiles,
6: SysUtils,
7: ComObj,
8: HermesEngine in 'HermesEngine.pas' {Form6},
9: PEngine_TLB in 'PEngine_TLB.pas',
10: EngineImpl in 'EngineImpl.pas' {HermesSession: CoClass},
11: AddressList in '..\hermesGUI\AddressList.pas',
12: HermesMedia in 'HermesMedia.pas',
13: HermesNode in 'HermesNode.pas',
```

```
14: TransportList in 'TransportList.pas',
15: EngineGlobals in 'EngineGlobals.pas',
16: Routing in 'Routing.pas',
17: Fragments in 'Fragments.pas',
18: HermesPacket in 'HermesPacket.pas',
19: HermesConnection in 'HermesConnection.pas',
20: MediaModule_TLB in '..\HermesMedia\MediaModule_TLB.pas',
21: HermesVariants in 'HermesVariants.pas';
22:
23: {$R *.TLB}
24:
25: {$R *.res}
26:
27: var
28: Index: Integer;
29: begin
30: Application.Initialize;
31: Application.CreateForm(TForm6, Form6);
32: EngineHasIdentity := False;
33: PacketSize := 16000; // Perhaps dynamic later...
34: FNodeList := TNodeInfoList.Create;
35: FConfig := THashedStringList.Create;
36: FMediaList := TMediaList.Create;
37: FConfig.LoadFromFile('HermesConfig.hme');
38: Index2 := 0;
39: Index := FConfig.IndexOfName('Media');
40: while (Index > -1) and (Index2 < 20) do begin
41: MediaFile := Copy(FConfig.Strings[Index], 7, 255);
42: FConfig.Delete(Index);
43: FMediaList.AddMediaInfo(MediaFile);
44: Index2 := Index2 + 1;
45: Index := FConfig.IndexOfName('Media');
46: Form6.Memo1.Lines.Add('Starting media nr:' + IntToStr(Index2));
47: end;
48:
49: SystemRouter := TRouter.Create;
50:
51: Application.Run;
52: end.
```

A.3.2 C:\projects\hermesGUI\AddressList.pas

```
1: unit AddressList;
2:
3: interface
4: uses
5: HermesTypes, Contnrs;
6:
7: type
8: TMediaLocator = class(TObject)
9: public
10: MediaType: TMediaType;
11: MediaAddress: TMediaAddress;
12: constructor Create(TheType: TMediaType; TheAddress: TMediaAddress);
13: end;
14:
15: TLocatorList = class(TObjectList)
16: public
17: procedure AddLocator(Locator: TMediaLocator); overload;
18: procedure AddLocator(TheType: TMediaType; TheAddress: TMediaAddress); over-
load;
```

```
19: end;
20:
21: implementation
22:
23: constructor TMediaLocator.Create(TheType: TMediaType; TheAddress: TMedi-
aAddress);
24: begin
25: inherited create;
26: MediaType := TheType;
27: MediaAddress := TheAddress;
28: end;
29:
30: procedure TLocatorList.AddLocator(Locator: TMediaLocator);
31: begin
32: Add(Locator);
33: end;
34:
35: procedure TLocatorList.AddLocator(TheType: TMediaType; TheAddress: TMedi-
aAddress);
36: begin
37: Add(TMediaLocator.Create(TheType, TheAddress));
38: end;
39:
40: end.
```

A.3.3 C:\projects\hermes\EngineGlobals.pas

```
1: unit EngineGlobals;
2:
3: interface
4:
5: uses
6: Routing, HermesTypes, HermesMedia, HermesNode, IniFiles;
7:
8:
9: var
10: FMediaList: TMediaList; // List of Communication Medias connected to this Engine
11: FNodeList: TNodeInfoList; // List of nodes known by this engine
12: FConfig: THashedStringList; // Configuration (read from file, key=value entries
13: Index, Index2: Integer; // Temporary vars
14: MediaFile: WideString; // Temporary var
15: EngineIdentity: WideString;
16: EngineHasIdentity: Boolean;
17: SystemRouter: TRouter;
18: PacketSize: Integer; // The Maximum size of packets in this engine
19:
20: function IsLocalNode(TheNode: TNodeID): Boolean;
21: function IsKnownNode(TheNode: TNodeID): Boolean;
22:
23: implementation
24:
25:
26: function IsLocalNode(TheNode: TNodeID): Boolean;
27: var
28: Index: Integer;
29: begin
30: Result := False;
31: Index := FNodeList.IndexOf(TheNode);
32: If Index >= 0 then
33: Result := FNodeList.Objects[Index] is TLocalNodeInfo;
34: end;
```

```
35:
36: function IsKnownNode(TheNode: TNodeID): Boolean;
37: begin
38: Result := (FNodeList.IndexOf(TheNode) >= 0);
39: end;
40:
41:
42: end.
```

A.3.4 C:\projects\hermes\EngineImpl.pas

```
1: unit EngineImpl;
2:
3: {$WARN SYMBOL_PLATFORM OFF}
4:
5: interface
6:
7: uses
8: Routing, HermesConnection, HermesPacket,
9: HermesNode, Inifiles, HermesMedia, PTransport_TLB, HermesMessage, HermesTy-
pes,
10: ComObj, ActiveX, AxCtrls, Classes, PEngine_TLB, StdVcl, TransportList;
11:
12: type
13: THermesSession = class(TAutoObject, IConnectionPointContainer, IHermesSession)
14: private
15: { Private declarations }
16: FConnectionPoints: TConnectionPoints;
17: FConnectionPoint: TConnectionPoint;
18: FEvents: IHermesSessionEvents;
19: { note: FEvents maintains a *single* event sink. For access to more
20: than one event sink, use FConnectionPoint.SinkList, and iterate
21: through the list of sinks. }
22: FNodeInfo: TNodeInfo;
23: IncomingQueue: THermesMessageQueue;
24: OutgoingQueue: THermesMessageQueue;
25: ConnectionList: THermesConnectionList;
26: public
27: procedure Initialize; override;
28: procedure FireOnMessageReceived;
29: procedure MessageEnqueue(TheMessage: THermesMessage);
30: Procedure MessageDispatch(TheMessage: THermesMessage; TheSender: Pointer);
31: Procedure NewPacket(ThePacket: THermesPacket);
32: protected
33: { Protected declarations }
34: property ConnectionPoints: TConnectionPoints read FConnectionPoints
35: implements IConnectionPointContainer;
36: procedure EventSinkChanged(const EventSink: IUnknown); override;
37: procedure MediaStart(const Config: WideString); safecall;
38: procedure MediaStop; safecall;
39: procedure MessageReceive(out TheMessage: OleVariant;
40: out NoMessages: Integer); safecall;
41: procedure MessageSend(TheMessage: OleVariant); safecall;
42: procedure NodeAdd(const Node: WideString); safecall;
43: procedure NodeMediaAdd(const Node, MediaType, Address, Port: WideString);
44: safecall;
45: procedure NodeMediaRemove(const Node, MediaType, Address: WideString);
46: safecall;
47: procedure NodeRemove(const Node: WideString); safecall;
48: procedure QueueGetReceiveMaxSize; safecall;
49: procedure QueueGetReceiveSize; safecall;
```

```
50: procedure QueueGetTransmitMaxSize; safecall;
51: procedure QueueGetTransmitSize; safecall;
52: procedure QueueSetReceiveMaxSize; safecall;
53: procedure QueueSetTransmitMaxSize; safecall;
54: procedure Close; safecall;
55: procedure Open(const Node: WideString); safecall;
56: procedure MediaConnect(const MediaType, MediaAddress,
57: ConnectionAddress: WideString); safecall;
58: procedure MediaConnectionAddress(const MediaType, MediaAddress: WideString;
59: out ConnectionAddress: WideString); safecall;
60: procedure MediaDisconnect(const MediaType, MediaAddress: WideString);
61: safecall;
62: procedure MediaInventory(out MediaList: OleVariant); safecall;
63: procedure MediaStatus(const MediaType, MediaAdress: WideString;
64: out MediaStatus: SYSINT); safecall;
65: end;
66:
67:
68:
69:
70: implementation
71:
72: uses
73: Variants, ComServ, HermesEngine, Sysutils, EngineGlobals;
74:
75: Procedure THermesSession.NewPacket(ThePacket: THermesPacket);
76: var
77: Engines: TStringList;
78: Index: Integer;
79: begin
80:
81: {$IFDEF DEBUG}
82: Form6.Memo1.Lines.Add('<At node:' + FNodeInfo.Name);
83: Form6.Memo1.Lines.Add('New packet coming in to:' + ThePacket.GetReceiver);
84: Form6.Memo1.Lines.Add('From:' + ThePacket.GetSender);
85: Form6.Memo1.Lines.Add('Type:' + IntToStr(Integer(ThePacket.GetType)));
86: Form6.Memo1.Lines.Add('Priority:' + IntToStr(ThePacket.GetPriority));
87: Form6.Memo1.Lines.Add('Has visited the following engines:');
88: Engines := ThePacket.GetEngines;
89: Index := Engines.Count - 1;
90: while Index >= 0 do
91: begin
92: Form6.Memo1.Text := Form6.Memo1.Text + Engines.Strings[Index];
93: Dec(Index, 1);
94: end;
95: {$ENDIF}
96:
97:
98: {
99: Here a message from "A" to "alla", will be treated the same as a message
100: from "A" to any node.
101:
102: Fix:
103: 1. Treat packets to "alla" as connectionless.
104: Drawback: Impossible to request retransmission.
105: 2. Treat "alla" as a sender.
106: Drawback: Ack - packets will be sent back to "alla".
107: 3. Let the Engine handle a speciel defragmenter for Messages coming
108: in to "alla".
109:
110: Solution 3. is selected since none of the drawbacks of 1 and 2 are
```

```
111: acceptable.
112: }
113:
114: if LowerCase(ThePacket.GetReceiver) = 'alla' then
115: begin
116: ConnectionList.Find(ThePacket.GetReceiver).DeFragmenter.HandleFragment(The-
Packet);
117: end
118: else
119: begin
120: if not Assigned(ConnectionList.Find(ThePacket.GetSender)) then
121: begin
122:
123: {$IFDEF DEBUG}
124: Form6.Memo1.Lines.Add('New sender, adding a new connection');
125: {$ENDIF}
126:
127: ConnectionList.Add(THermesConnection.Create(ThePacket.GetSender, Self, Sys-
temRouter));
128: end;
129: ConnectionList.Find(ThePacket.GetSender).DeFragmenter.HandleFragment(The-
Packet);
130:
131: {$IFDEF DEBUG}
132: Form6.Memo1.Lines.Add('>');
133: {$ENDIF}
134:
135: end;
136: end;
137:
138: procedure THermesSession.MessageEnqueue(TheMessage: THermesMessage);
139: begin
140: IncomingQueue.Push(TheMessage);
141: end;
142:
143: procedure THermesSession.FireOnMessageReceived;
144: var
145: I: Integer;
146: EventSinkList: TList;
147: EventSink: IHermesSessionEvents;
148: begin
149: if FConnectionPoint <> nil then
150: begin
151: EventSinkList :=FConnectionPoint.SinkList; {get the list of client sinks }
152: for I := 0 to EventSinkList.Count - 1 do
153: begin
154: EventSink := IUnknown(EventSinkList.Items[I]) as IHermesSessionEvents;
155: //Form6.Memo1.Lines.Add('Firing OnMessageReceived on client ' + IntToStr(I));
156: EventSink.OnContactLost;
157: EventSink.OnMessageReceived;
158: end;
159: end;
160: end;
161:
162: //begin
163: // Form6.Memo1.Lines.Add('Firing OnMessageReceived.');
164: // FEvents.OnMessageReceived;
165: //end;
166:
167:
168:
```

```
169: Procedure THermesSession.MessageDispatch(TheMessage: THermesMessage; The-
Sender: Pointer);
170: var
171: Index: Integer;
172: TheReceiver: TObject;
173: //LocalReceiver: THermesSession;
174: begin
175:
176: {$IFDEF DEBUG}
177: Form6.Memo1.Lines.Add('Receiver:' + TheMessage.GetReceiver);
178: {$ENDIF}
179:
180: If TheMessage.GetReceiver = 'Alla' then begin
181: // Distribute message to all local nodes and send over all Media
182:
183: {$IFDEF DEBUG}
184: Form6.Memo1.Lines.Add('Public message, send to all.');
185: {$ENDIF}
186:
187: {
188: for Index := 0 to (FNodeList.Count - 1) do begin
189: TheReceiver := FNodeList.Objects[Index];
190: If TheReceiver is TLocalNodeInfo then begin
191: with TheReceiver as TLocalNodeInfo do begin
192: if Rule <> TheSender then begin
193: NewMessage(TheMessage);
194: end;
195: end;
196: end;
197: end;
198: }
199:
200: ConnectionList.Find('Alla').Fragmenter.HandleMessage(TheMessage);
201:
202: // Send message over media also. (at startup, add an default node that
203: // Has the name all, and contains all media?).
204:
205: end else begin
206:
207: // Send to specified node only
208: Index := FNodeList.IndexOf(TheMessage.GetReceiver);
209:
210: {$IFDEF DEBUG}
211: Form6.Memo1.Lines.Add('Receiver found at NodeList[' + IntToStr(Index) + '].');
212: {$ENDIF}
213:
214: If Index >= 0 then begin
215: TheReceiver := FNodeList.Objects[Index];
216: If Assigned(TheReceiver) then if TheReceiver is TLocalNodeInfo then with TheRe-
ceiver as TLocalNodeInfo do
217: begin
218: // Send message to local node
219:
220: {$IFDEF DEBUG}
221: Form6.Memo1.Lines.Add('Sending to local node.');
222: {$ENDIF}
223:
224: NewMessage(TheMessage);
225: end
226: else
227: begin
```

```
228: // Send message to remote node
229:
230: {$IFDEF DEBUG}
231: Form6.Memo1.Lines.Add('Sending to remote node.');
232: {$ENDIF}
233:
234: ConnectionList.Find(TheMessage.GetReceiver).Fragmenter.HandleMessage(The-
Message);
235: end;
236: end else begin
237:
238: {$IFDEF DEBUG}
239: Form6.Memo1.Lines.Add('Unknown receiver. Adding remote node.');
240: {$ENDIF}
241:
242: // Add new remote node
243: FNodeList.Add(TNodeInfo(TRemoteNodeInfo.Create(TheMessage.GetReceiver)));
244: // Resend on all media except for incoming
245:
246: {$IFDEF DEBUG}
247: Form6.Memo1.Lines.Add('Sending to remote node.');
248: {$ENDIF}
249:
250: ConnectionList.Find(TheMessage.GetReceiver).Fragmenter.HandleMessage(The-
Message);
251:
252: end;
253: end;
254: end;
255:
256:
257: procedure THermesSession.EventSinkChanged(const EventSink: IUnknown);
258: begin
259: FEvents := EventSink as IHermesSessionEvents;
260: end;
261:
262: procedure THermesSession.Initialize;
263: begin
264: inherited Initialize;
265: FConnectionPoints := TConnectionPoints.Create(Self);
266: if AutoFactory.EventTypeInfo <> nil then
267: FConnectionPoint := FConnectionPoints.CreateConnectionPoint(
268: AutoFactory.EventIID, ckSingle, EventConnect)
269: else FConnectionPoint := nil;
270: end;
271:
272:
273: procedure THermesSession.MediaStart(const Config: WideString);
274: begin
275:
276: end;
277:
278: procedure THermesSession.MediaStop;
279: begin
280:
281: end;
282:
283: procedure THermesSession.MessageReceive(out TheMessage: OleVariant;
284: out NoMessages: Integer);
285: var
286: AMessage: THermesMessage;
```

```
287: begin
288:
289: {$IFDEF DEBUG}
290: Form6.Memo1.Lines.Add('Message receive called.');
291: {$ENDIF}
292:
293: NoMessages := IncomingQueue.Count;
294:
295: {$IFDEF DEBUG}
296: Form6.Memo1.Lines.Add(IntToStr(NoMessages) + ' Messages in queue.');
297: {$ENDIF}
298:
299: if NoMessages > 0 then begin
300: AMessage := IncomingQueue.Pop;
301:
302: {$IFDEF DEBUG}
303: Form6.Memo1.Lines.Add('Receiving the following message:');
304: Form6.Memo1.Lines.Add('Sender:' + AMessage.GetSender);
305: Form6.Memo1.Lines.Add('Receiver:' + AMessage.GetReceiver);
306: Form6.Memo1.Lines.Add('Data:' + AnsiString(AMessage.GetData));
307: Form6.Memo1.Lines.Add('Priority:' + IntToStr(AMessage.GetPriority));
308: Form6.Memo1.Lines.Add('Creation time:' + DateTimeToStr(AMessage.GetCreate-
Time));
309: Form6.Memo1.Lines.Add('Best before time:' + DateTimeToStr(AMessage.GetValid-
Time));
310: {$ENDIF}
311:
312: TheMessage := AMessage.GetTheMessage;
313: end
314: else
315: begin
316: //TheMessage := nil;
317: // No Message to receive
318: end;
319: end;
320:
321: procedure THermesSession.MessageSend(TheMessage: OleVariant);
322: var
323: FMessage: THermesMessage;
324: begin
325:
326: {$IFDEF DEBUG}
327: Form6.Memo1.Lines.Add('Message received for transmission');
328: {$ENDIF}
329:
330: FMessage := THermesMessage.Create;
331: FMessage.SetTheMessage(TheMessage);
332:
333:
334: {$IFDEF DEBUG}
335: Form6.Memo1.Lines.Add('Sender:' + FMessage.GetSender);
336: Form6.Memo1.Lines.Add('Receiver:' + FMessage.GetReceiver);
337: Form6.Memo1.Lines.Add('Data:' + AnsiString(FMessage.GetData));
338: Form6.Memo1.Lines.Add('Priority:' + IntToStr(FMessage.GetPriority));
339: Form6.Memo1.Lines.Add('Creation time:' + DateTimeToStr(FMessage.GetCreate-
Time));
340: Form6.Memo1.Lines.Add('Best before time:' + DateTimeToStr(FMessage.GetValid-
Time));
341: {$ENDIF}
342:
343: if not Assigned(ConnectionList.Find(FMessage.GetReceiver)) then
```

```
344: begin
345:
346: {$IFDEF DEBUG}
347: Form6.Memo1.Lines.Add('New receiver, adding a new connection');
348: {$ENDIF}
349:
350: ConnectionList.Add(THermesConnection.Create(FMessage.GetReceiver, Self, Sys-
temRouter));
351: end;
352: MessageDispatch(FMessage, self);
353: end;
354:
355: procedure THermesSession.NodeAdd(const Node: WideString);
356: begin
357:
358: end;
359:
360: procedure THermesSession.NodeMediaAdd(const Node, MediaType, Address,
361: Port: WideString);
362: begin
363:
364: end;
365:
366: procedure THermesSession.NodeMediaRemove(const Node, MediaType,
367: Address: WideString);
368: begin
369:
370: end;
371:
372: procedure THermesSession.NodeRemove(const Node: WideString);
373: begin
374:
375: end;
376:
377: procedure THermesSession.QueueGetReceiveMaxSize;
378: begin
379:
380: end;
381:
382: procedure THermesSession.QueueGetReceiveSize;
383: begin
384:
385: end;
386:
387: procedure THermesSession.QueueGetTransmitMaxSize;
388: begin
389:
390: end;
391:
392: procedure THermesSession.QueueGetTransmitSize;
393: begin
394:
395: end;
396:
397: procedure THermesSession.QueueSetReceiveMaxSize;
398: begin
399:
400: end;
401:
402: procedure THermesSession.QueueSetTransmitMaxSize;
403: begin
```

```
404:
405: end;
406:
407: procedure THermesSession.Close;
408: var
409: Index: Integer;
410: begin
411:
412: {$IFDEF DEBUG}
413: Form6.Memo1.Lines.Add('Closing session for ' + FNodeInfo.Name + '.');
414: {$ENDIF}
415:
416: Index := FNodeList.IndexOfObject(FNodeInfo);
417: FNodeList.Delete(Index);
418: FNodeInfo.Destroy;
419: end;
420:
421: procedure THermesSession.Open(const Node: WideString);
422: begin
423:
424: {$IFDEF DEBUG}
425: Form6.Memo1.Lines.Add('Opening session for ' + Node + '.');
426: {$ENDIF}
427:
428: If not EngineHasIdentity then begin
429: EngineIdentity := Node;
430: EngineHasIdentity := True;
431: end;
432:
433: {$IFDEF DEBUG}
434: Form6.Memo1.Lines.Add('EngineID:' + EngineIdentity);
435: {$ENDIF}
436:
437: FNodeInfo:= TLocalNodeInfo.Create(Node, self);
438: FNodeList.Add(FNodeInfo);
439: IncomingQueue := THermesMessageQueue.Create;
440: OutgoingQueue := THermesMessageQueue.Create;
441: ConnectionList := THermesConnectionList.Create;
442: ConnectionList.Add(THermesConnection.Create(Node, Self, SystemRouter));
443: if not Assigned(ConnectionList.Find('Alla')) then
444: begin
445:
446: {$IFDEF DEBUG}
447: Form6.Memo1.Lines.Add('Adding node Alla');
448: {$ENDIF}
449:
450: ConnectionList.Add(THermesConnection.Create('Alla', self, SystemRouter));
451: end;
452: end;
453:
454: procedure THermesSession.MediaConnect(const MediaType, MediaAddress,
455: ConnectionAddress: WideString);
456: var
457: Media: TMediaInfo;
458: begin
459: Media := FMediaList.FindMediaInfo(MediaType, MediaAddress);
460: If Assigned(Media) then
461: begin
462: Media.Transport.ConnectionOpen(ConnectionAddress);
463: end;
464: end;
```

```
465:
466: procedure THermesSession.MediaConnectionAddress(const MediaType,
467: MediaAddress: WideString; out ConnectionAddress: WideString);
468: var
469: Media: TMediaInfo;
470: begin
471: Media := FMediaList.FindMediaInfo(MediaType, MediaAddress);
472: If Assigned(Media) then
473: begin
474: ConnectionAddress := Media.ConnectAddress;
475: end;
476: end;
477:
478: procedure THermesSession.MediaDisconnect(const MediaType,
479: MediaAddress: WideString);
480: var
481: Media: TMediaInfo;
482: begin
483: Media := FMediaList.FindMediaInfo(MediaType, MediaAddress);
484: If Assigned(Media) then
485: begin
486: Media.Transport.ConnectionClose;
487: end;
488: end;
489:
490: procedure THermesSession.MediaInventory(out MediaList: OleVariant);
491: var
492: Index: Integer;
493: Count: Integer;
494: begin
495: MediaList := VarArrayCreate([0, FMediaList.Count - 1], VT_BSTR);
496: Count := 0;
497: for Index := 0 to FMediaList.Count - 1 do
498: begin
499: if Assigned(FMediaList.Items[Index]) then
500: begin
501: MediaList[Count] := TMediaInfo(FMediaList.Items[Index]).MediaType + '=' +
502: TMediaInfo(FMediaList.Items[Index]).Address;
503: Inc(Count, 1);
504: end;
505: end;
506: if Count < FMediaList.Count - 1 then
507: begin
508: VarArrayRedim(MediaList, Count - 1);
509: end;
510: end;
511:
512: procedure THermesSession.MediaStatus(const MediaType,
513: MediaAdress: WideString; out MediaStatus: SYSINT);
514: var
515: Media: TMediaInfo;
516: begin
517: Media := FMediaList.FindMediaInfo(MediaType, MediaAdress);
518: If Assigned(Media) then
519: begin
520: Media.Transport.ConnectionGetStatus(MediaStatus);
521: end;
522: end;
523:
524: initialization
525: TAutoObjectFactory.Create(ComServer, THermesSession, Class_HermesSession,
```

```
526: ciMultiInstance, tmFree);
527: end.
528:
```

A.3.5 C:\projects\hermes\Fragments.pas

```
1: unit Fragments;
2:
3: interface
4:
5: uses
6: Math, HermesMessage, HermesTypes, HermesPacket;
7:
8: type
9: TFragmenter = class(TObject)
10: private
11: protected
12: FragmentSize: Integer;
13: Router: TObject;
14: MessageQueue: THermesMessageQueue;
15: AckQueue: THermesPacketQueue;
16: CurrentMessage: THermesMessage;
17: CurrentSize: Integer;
18: CurrentPosition: Integer;
19: CurrentSequence: LongWord;
20: Function NextPacket: TPHermesPacket;
21: Procedure SetCurrentMessage(const TheMessage: THermesMessage);
22: public
23: Procedure HandleMessage(const TheMessage: THermesMessage);
24: Procedure GetFragment(out ThePacket: THermesPacket);
25: Procedure Ack(const SequenceNumber: LongWord);
26: Procedure Nak(const SequenceNumber: LongWord);
27: Constructor Create(TheRouter: TObject);
28: end;
29:
30: TDeFragmenter = class(TObject)
31: private
32: protected
33: Session: TObject;
34: Router: TObject;
35: MessageQueue: THermesMessageQueue;
36: CurrentMessage: THermesMessage;
37: CurrentSize: Integer;
38: CurrentPosition: Integer;
39: CurrentSequence: LongWord;
40: Function CheckSequence(const ThePacket: THermesPacket): Boolean;
41: public
42: Fragmenter: TFragmenter;
43: Procedure HandleFragment(const ThePacket: THermesPacket);
44: Procedure GetMessage(out TheMessage: THermesMessage);
45: Constructor Create(TheSession: TObject; TheRouter: TObject);
46: end;
47:
48:
49: implementation
50: uses
51: Routing, EngineGlobals, EngineImpl;
52:
53: Procedure TFragmenter.Ack(const SequenceNumber: LongWord);
54: //var
55: // APacket: THermesPacket;
```

```
56: begin
57: //SequenceNumber := ThePacket.GetSequenceNumber;
58: //Assert(AckQueue.AtLeast(1), 'AckQueue empty when trying to Ack');
59: if AckQueue.AtLeast(1) then
60: begin
61: if AckQueue.Peek.GetSequenceNumber = SequenceNumber then
62: begin
63: AckQueue.Pop; // Throw away Acked packet
64: end;
65: end;
66: end;
67:
68: Procedure TFragmenter.Nak(const SequenceNumber: LongWord);
69: var
70: APacket: THermesPacket;
71: FirstSeq: LongWord;
72: begin
73: //Assert(AckQueue.AtLeast(1), 'AckQueue empty when trying to Nak');
74: //SequenceNumber := ThePacket.GetSequenceNumber;
75: if AckQueue.AtLeast(1) then
76: begin
77: if AckQueue.Peek.GetSequenceNumber = SequenceNumber then
78: begin
79: AckQueue.Pop;
80: end;
81: end;
82: // TODO: Resend All packets that are left in the queue
83: APacket := AckQueue.Pop;
84: Assert(Assigned(APacket), 'Ack queue empty.');
85: FirstSeq := APacket.GetSequenceNumber;
86: TRouter(Router).PacketDispatch(APacket, nil);
87: AckQueue.Push(APacket);
88: While AckQueue.Peek.GetSequenceNumber <> FirstSeq do
89: begin
90: APacket := AckQueue.Pop;
91: TRouter(Router).PacketDispatch(APacket, nil);
92: AckQueue.Push(APacket);
93: end;
94:
95: end;
96:
97: Procedure TFragmenter.SetCurrentMessage(const TheMessage: THermesMessage);
98: begin
99: CurrentMessage := TheMessage;
100: CurrentSize := Length(TheMessage.GetData);
101: CurrentPosition := 0;
102: end;
103:
104: Procedure TFragmenter.HandleMessage(const TheMessage: THermesMessage);
105: var
106: APacket: TPHermesPacket;
107: begin
108: if not Assigned(CurrentMessage) then
109: begin
110: SetCurrentMessage(TheMessage);
111: // Send all fragments to router
112: While Assigned(CurrentMessage) do
113: begin
114: APacket := NextPacket;
115: AckQueue.Push(APacket^);
116: SystemRouter.PacketDispatch(APacket^, nil);
```

```
117: end;
118: end
119: else
120: begin
121: MessageQueue.Push(TheMessage);
122: end;
123: end;
124:
125: Function TFragmenter.NextPacket: TPHermesPacket;
126: var
127: NextData: TPacketArray;
128: Index, DataLength: Integer;
129: TheType: ThermesPacketType;
130: begin
131: if (CurrentSize - CurrentPosition) <= FragmentSize then
132: begin
133: TheType := FloodDataStop;
134: end
135: else
136: if CurrentPosition = 0 then
137: begin
138: TheType := FloodDataStart;
139: end
140: else
141: begin
142: TheType := FloodData;
143: end;
144:
145: DataLength := min(CurrentSize-CurrentPosition, FragmentSize);
146: SetLength(NextData, DataLength);
147:
148: For Index := Low(NextData) to High(NextData) do
149: begin
150: NextData[Index] := (CurrentMessage.GetData)[CurrentPosition];
151: Inc(CurrentPosition, 1);
152: end;
153:
154: new(result);
155: result^ := THermesPacket.Create(TheType, CurrentMessage.GetReceiver, Current-
Message.GetSender);
156: result^.SetData(@NextData);
157: result^.SetPriority(CurrentMessage.GetPriority);
158: result^.SetSequenceNumber(CurrentSequence);
159: Inc(CurrentSequence, 1);
160: //result.AddEngine(EngineIdentity);
161:
162: if CurrentPosition >= CurrentSize then
163: begin
164: CurrentMessage := nil;
165: end;
166: end;
167:
168:
169: Procedure TFragmenter.GetFragment(out ThePacket: THermesPacket);
170: begin
171: if Assigned(CurrentMessage) then
172: begin
173: // Send a part of the message
174: ThePacket := NextPacket^;
175: end
176: else if MessageQueue.Count > 0 then
```

```
177: begin
178: CurrentMessage := MessageQueue.Pop;
179: CurrentSize := SizeOf(CurrentMessage.GetData);
180: CurrentPosition := 0;
181: // Send first part of the message
182: ThePacket := NextPacket^;
183: end
184: else
185: begin
186: // Nothing to fragment, Error?
187: ThePacket := nil;
188: end;
189: end;
190:
191: Constructor TFragmenter.Create(TheRouter: TObject);
192: begin
193: FragmentSize := 8;
194: MessageQueue := THermesMessageQueue.Create;
195: AckQueue := THermesPacketQueue.Create;
196: CurrentMessage := nil;
197: CurrentSize := 0;
198: CurrentPosition := 0;
199: CurrentSequence := 0;
200: Router := TheRouter;
201: end;
202:
203:
204: Function TDeFragmenter.CheckSequence(const ThePacket: THermesPacket): Boo-
lean;
205: var
206: //PacketType: THermesPacketType;
207: PReplyPacket: TPHermesPacket;
208: begin
209: result := false;
210: if ThePacket.GetSequenceNumber = CurrentSequence then
211: begin
212: // The packet that we have been waiting for, send ACK and
213: New(PReplyPacket);
214: PReplyPacket^ := THermesPacket.Create(FloodAck, WideString(ThePacket.GetSen-
der), WideString(ThePacket.GetReceiver));
215: PReplyPacket^.SetSequenceNumber(CurrentSequence);
216: TRouter(Router).PacketDispatch(PReplyPacket^, Session);
217: // increment current sequence
218: Inc(CurrentSequence);
219: result := true; // The sequence number is OK
220: end
221: else if ThePacket.GetSequenceNumber > CurrentSequence then
222: begin
223: // We have missed a packet, send NAK
224: {
225: New(PReplyPacket);
226: PReplyPacket^ := THermesPacket.Create(FloodNak, WideString(ThePacket.GetSender),
WideString(ThePacket.GetReceiver));
227: Dec(CurrentSequence, 1);
228: PReplyPacket^.SetSequenceNumber(CurrentSequence);
229: Inc(CurrentSequence, 1);
230: TRouter(Router).PacketDispatch(PReplyPacket^, Session);
231: }
232: end;
233: end;
234:
```

```
235: Procedure TDeFragmenter.HandleFragment(const ThePacket: THermesPacket);
236: var
237: PacketType: THermesPacketType;
238: NewData: TPacketArray;
239: begin
240: PacketType := ThePacket.GetType;
241: case PacketType of
242:
243: FloodNak:
244: begin
245: Assert(Assigned(Fragmenter), 'No fragmenter to handle frame!');
246: // NAK to fragmenter
247: Fragmenter.Nak(ThePacket.GetSequenceNumber);
248: end;
249:
250: FloodAck:
251: begin
252: Assert(Assigned(Fragmenter), 'No fragmenter to handle frame!');
253: // ACK to fragmenter
254: Fragmenter.Ack(ThePacket.GetSequenceNumber);
255: end;
256:
257: FloodDataStart:
258: begin
259: if CheckSequence(ThePacket) then
260: begin
261: // Deliver current message if present
262: if Assigned(CurrentMessage) then
263: begin
264: THermesSession(Session).MessageEnqueue(CurrentMessage);
265: THermesSession(Session).FireOnMessageReceived;
266: CurrentMessage := nil;
267: end;
268: // Start on a new message
269: CurrentMessage := THermesMessage.Create;
270: CurrentMessage.SetSender(ThePacket.GetSender);
271: CurrentMessage.SetReceiver(ThePacket.GetReceiver);
272: CurrentMessage.SetPriority(ThePacket.GetPriority);
273: CurrentMessage.SetCreateTime(0);
274: CurrentMessage.SetValidTime(0);
275: // Add fragment to current message
276: ThePacket.GetData(@NewData);
277: CurrentMessage.SetData(TMessageData(NewData));
278: end;
279: end;
280:
281: FloodData:
282: begin
283: if CheckSequence(ThePacket) then
284: begin
285: Assert(Assigned(CurrentMessage));
286: // Add fragment to current message
287: ThePacket.GetData(@NewData);
288: CurrentMessage.AddData(TMessageData(NewData));
289: end;
290: end;
291:
292: FloodDataStop:
293: begin
294: if CheckSequence(ThePacket) then
295: begin
```

```
296: ThePacket.GetData(@NewData);
297: if not Assigned(CurrentMessage) then
298: begin
299: // Start on a new message
300: CurrentMessage := THermesMessage.Create;
301: CurrentMessage.SetSender(ThePacket.GetSender);
302: CurrentMessage.SetReceiver(ThePacket.GetReceiver);
303: CurrentMessage.SetPriority(ThePacket.GetPriority);
304: CurrentMessage.SetCreateTime(0);
305: CurrentMessage.SetValidTime(0);
306: CurrentMessage.SetData(TMessageData(NewData));
307: end
308: else
309: begin
310: // Add fragment to current message
311: CurrentMessage.AddData(TMessageData(NewData));
312: end;
313: // Deliver current message
314: THermesSession(Session).MessageEnqueue(CurrentMessage);
315: THermesSession(Session).FireOnMessageReceived;
316: // Set current message to nil
317: CurrentMessage := nil;
318: end;
319: end;
320: else
321: begin
322: Assert(false);
323: end;
324: end;
325: end;
326:
327: Procedure TDeFragmenter.GetMessage(out TheMessage: THermesMessage);
328: begin
329: end;
330:
331: Constructor TDeFragmenter.Create(TheSession: TObject; TheRouter: TObject);
332: begin
333: inherited Create;
334: Session := TheSession;
335: Router := TheRouter;
336: MessageQueue := THermesMessageQueue.Create;
337: CurrentMessage := nil;
338: CurrentSize := 0;
339: CurrentPosition := 0;
340: CurrentSequence := 0;
341: end;
342:
343:
344:
345: end.
```

A.3.6 C:\projects\hermes\HermesConnection.pas

```
1: unit HermesConnection;
2:
3: interface
4:
5: uses
6: Contnrs, Fragments, HermesTypes;
7:
8: Type
```

```
9: THermesConnection = Class(TObject)
10: private
11: protected
12: public
13: NodeID: TNodeID;
14: Fragmenter: TFragmenter;
15: DeFragmenter: TDeFragmenter;
16: constructor Create(TheNode: TNodeID; TheSession: Pointer; TheRouter: Pointer);
17: end;
18:
19: THermesConnectionList = Class(TObjectList)
20: private
21: protected
22: public
23: procedure Add(AConnection: THermesConnection);
24: function Find(ANode: TNodeID): ThermesConnection;
25: end;
26:
27: implementation
28: procedure THermesConnectionList.Add(AConnection: THermesConnection);
29: begin
30: inherited Add(TObject(AConnection));
31: end;
32:
33: function THermesConnectionList.Find(ANode: TNodeID): ThermesConnection;
34: var
35: Index: Integer;
36: begin
37: result := nil;
38: for Index := 0 to Count - 1 do
39: begin
40: if Assigned (Items[Index]) then
41: begin
42: if THermesConnection(Items[Index]).NodeID = ANode then
43: begin
44: result := THermesConnection(Items[Index]);
45: end;
46: end;
47: end;
48: end;
49:
50:
51: constructor THermesConnection.Create(TheNode: TNodeID; TheSession: Pointer;
TheRouter: Pointer);
52: begin
53: NodeId := TheNode;
54: Fragmenter := TFragmenter.Create(TheRouter);
55: DeFragmenter := TDeFragmenter.Create(TheSession, TheRouter);
56: DeFragmenter.Fragmenter := Fragmenter;
57: end;
58:
59: end.
```

A.3.7 C:\projects\hermes\HermesEngine.pas

```
1: unit HermesEngine;
2:
3: interface
4:
5: uses
6: TransportList, HermesNode, Inifiles, HermesMedia, PTransport_TLB,
```

```
7: Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
8: Dialogs, StdCtrls;
9:
10: type
11: TForm6 = class(TForm)
12: Memo1: TMemo;
13: private
14: { Private declarations }
15: public
16: { Public declarations }
17: end;
18:
19: var
20: Form6: TForm6;
21:
22: implementation
23:
24: {$R *.dfm}
25:
26:
27:
28: end.
```

A.3.8 C:\projects\hermes\HermesEngine.dfm

```
1: object Form6: TForm6
2: Left = 877
3: Top = 158
4: Width = 361
5: Height = 388
6: Caption = 'Hermes Engine'
7: Color = clBtnFace
8: Font.Charset = DEFAULT_CHARSET
9: Font.Color = clWindowText
10: Font.Height = -11
11: Font.Name = 'MS Sans Serif'
12: Font.Style = []
13: OldCreateOrder = False
14: PixelsPerInch = 96
15: TextHeight = 13
16: object Memo1: TMemo
17: Left = 0
18: Top = 0
19: Width = 353
20: Height = 361
21: Align = alClient
22: TabOrder = 0
23: end
24: end
```

A.3.9 C:\projects\hermes\HermesMedia.pas

```
1: unit HermesMedia;
2:
3: interface
4: uses
5: HermesTypes, Contnrs, Inifiles, ActiveX, PTransport_TLB, Classes;
6:
7: type
8: TMediaInfo = class(THashedStringList, IUnknown, IDispatch)
9: private
```

```
10: protected
11: { IUnknown }
12: function QueryInterface(const IID: TGUID; out Obj): HRESULT; stdcall;
13: function _AddRef: Integer; stdcall;
14: function _Release: Integer; stdcall;
15: { IDispatch }
16: function GetTypeInfoCount(out Count: Integer): HRESULT; stdcall;
17: function GetTypeInfo(Index, LocaleID: Integer; out TypeInfo): HRESULT; stdcall;
18: function GetIDsOfNames(const IID: TGUID; Names: Pointer;
19: NameCount, LocaleID: Integer; DispIDs: Pointer): HRESULT; stdcall;
20: function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer;
21: Flags: Word; var Params; VarResult, ExcepInfo, ArgErr: Pointer): HRESULT; stdcall;
22:
23: public
24: MediaType: TMediaType;
25: Address: TMediaAddress;
26: ConnectAddress: TMediaAddress;
27: Transport: OleVariant;
28: //TransportEvents: OleVariant;
29: TransportEventsConnection: Integer;
30: Running: Boolean;
31: Router: Pointer;
32: constructor Create(TheType: TMediaType; TheAddress: TMediaAddress); overload;
33: constructor Create(SetupFile: WideString); overload;
34: destructor Destroy; override;
35: procedure OnPacketReceived(Sender: TObject);
36: procedure OnLinkLost(Sender: TObject);
37: procedure OnLinkRequest(Sender: TObject);
38: end;
39:
40: TMediaList = class(TObjectList)
41: private
42: //Array of TMediaInfo;
43: protected
44: public
45: procedure AddMediaInfo(SetupFile: WideString); overload;
46: procedure RemoveMediaInfo(MediaType: TMediaType; Address: TMediaAddress);
47: procedure DeliverTo(PacketArray: TPPacketArray; Receiver: AnsiString);
48: procedure DeliverToAll(PacketArray: TPPacketArray);
49: function FindMediaInfo(TheType: TMediaType; TheAddress: TMediaAddress): TMe-
diaInfo;
50: constructor Create; overload;
51: end;
52:
53: function IsMediaCompatible(MediaA, MediaB: TMediaInfo): Boolean;
54:
55: implementation
56:
57: uses
58:
59: {$IFDEF DEBUG}
60: HermesEngine,
61: {$ENDIF}
62:
63: SysUtils,
64: HermesPacket, Variants, HermesVariants, Routing, MediaModule_TLB, comobj;
65:
66: function TMediaList.FindMediaInfo(TheType: TMediaType; TheAddress: TMedi-
aAddress): TMediaInfo;
67: var
68: Index: integer;
```

```
69: begin
70: result := nil;
71: For Index := 0 to Count - 1 do
72: begin
73: if Assigned(Items[Index]) then
74: begin
75: with Items[Index] as TMediaInfo do
76: begin
77: if (CompareText(WideString(TheType), WideString(MediaType)) = 0) and (Compare-
Text(WideString(TheAddress), WideString(Address)) = 0) then
78: begin
79: result := TMediaInfo(Items[Index]);
80: end;
81: end;
82: end;
83: end;
84: end;
85:
86: procedure TMediaList.DeliverTo(PacketArray: TPPacketArray; Receiver: AnsiString);
87: begin
88: end;
89:
90: procedure TMediaList.DeliverToAll(PacketArray: TPPacketArray);
91: var
92: Index: Integer;
93: PacketVarArray: OleVariant;
94: begin
95: ArrayToVariant(PacketArray^, PacketVarArray);
96: for Index := 0 to Count - 1 do
97: begin
98: if Assigned(Items[Index]) then
99: begin
100: TMediaInfo(Items[Index]).Transport.PacketTransmit(PacketVarArray);
101: end;
102: end;
103: end;
104:
105: procedure TMediaList.AddMediaInfo(SetupFile: WideString);
106: begin
107: Add(TMediaInfo.Create(SetupFile));
108: end;
109:
110: procedure TMediaList.RemoveMediaInfo(MediaType: TMediaType; Address: TMe-
diaAddress);
111: begin
112: end;
113:
114: constructor TMediaList.Create;
115: begin
116: inherited Create;
117: end;
118:
119:
120: function IsMediaCompatible(MediaA, MediaB: TMediaInfo): Boolean;
121: begin
122: IsMediaCompatible := (MediaA.MediaType = MediaB.MediaType);
123: end;
124:
125:
126: // IUnknown
127: // ********************************
```

```
128: function TMediaInfo.QueryInterface(const IID: TGUID; out Obj): HRESULT;
129: begin
130: // We need to return the two event interfaces when they're asked for
131: Result := E_NOINTERFACE;
132: if GetInterface(IID,Obj) then
133: Result := S_OK;
134: if IsEqualGUID(IID,DIID_IMediaEvents) and GetInterface(IDispatch,Obj) then
135: Result := S_OK;
136: end;
137:
138: function TMediaInfo._AddRef: Integer;
139: begin
140: // Skeleton implementation
141: Result := 2;
142: end;
143:
144: function TMediaInfo._Release: Integer;
145: begin
146: // Skeleton implementation
147: Result := 1;
148: end;
149:
150: // IDispatch
151: // ********************
152:
153: function TMediaInfo.GetTypeInfoCount(out Count: Integer): HRESULT;
154: begin
155: // Skeleton implementation
156: Count := 0;
157: Result := S_OK;
158: end;
159:
160: function TMediaInfo.GetTypeInfo(Index, LocaleID: Integer; out TypeInfo): HRE-
SULT;
161: begin
162: // Skeleton implementation
163: Result := E_NOTIMPL;
164: end;
165:
166: function TMediaInfo.GetIDsOfNames(const IID: TGUID; Names: Pointer;
167: NameCount, LocaleID: Integer; DispIDs: Pointer): HRESULT;
168: begin
169: // Skeleton implementation
170: Result := E_NOTIMPL;
171: end;
172:
173: function TMediaInfo.Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer;
174: Flags: Word; var Params; VarResult, ExcepInfo, ArgErr: Pointer): HRESULT;
175: begin
176: // Fire the different event handlers when
177: // the different event methods are invoked
178: case DispID of
179: 1 : OnPacketReceived(nil);
180: 2 : OnLinkLost(nil);
181: 3 : OnLinkRequest(nil);
182: end;
183: Result := S_OK;
184: end;
185:
186: procedure TMediaInfo.OnPacketReceived(Sender: TObject);
187: var
```

```
188: PacketVariant: OleVariant;
189: APacketArray: TPacketArray;
190: PacketsLeft: Integer;
191: begin
192:
193: {$IFDEF DEBUG}
194: Form6.Memo1.Lines.Add('OnPacketReceived');
195: {$ENDIF}
196:
197: PacketsLeft := 1;
198: while PacketsLeft > 0 do
199: begin
200: Dec(PacketsLeft, 1);
201:
202: {$IFDEF DEBUG}
203: Form6.Memo1.Lines.Add('Trying to receive packet.');
204: {$ENDIF}
205:
206: Transport.PacketReceive(PacketVariant, PacketsLeft);
207:
208: {$IFDEF DEBUG}
209: Form6.Memo1.Lines.Add('Packet received, converting to Array.');
210: {$ENDIF}
211:
212: ArrayFromVariant(APacketArray, PacketVariant);
213:
214: {$IFDEF DEBUG}
215: Form6.Memo1.Lines.Add('Conversion done, dispatching packet');
216: {$ENDIF}
217:
218: TRouter(Router).PacketDispatch(THermesPacket.Create(@APacketArray), self);
219:
220: {$IFDEF DEBUG}
221: Form6.Memo1.Lines.Add('Dispatching done, exiting event handler.');
222: {$ENDIF}
223:
224: end;
225: end;
226:
227: procedure TMediaInfo.OnLinkLost(Sender: TObject);
228: begin
229: end;
230:
231: procedure TMediaInfo.OnLinkRequest(Sender: TObject);
232: begin
233: end;
234:
235: constructor TMediaInfo.Create(TheType: TMediaType; TheAddress: TMedi-
aAddress);
236: begin
237: inherited Create;
238: Running := False;
239: Router := TRouter.Create;
240: MediaType := TheType;
241: Address := TheAddress;
242: ConnectAddress := 'Not connected';
243: Add('MediaType=' + TheType);
244: Add('Adress=' + TheAddress);
245: end;
246:
247: constructor TMediaInfo.Create(SetupFile: WideString);
```

```
248: var
249: {$IFDEF DEBUG}
250: DummyPacketVariant: OleVariant;
251: DummyPacketsLeft: Integer;
252: {$ENDIF}
253:
254: {
255: DummyPacket: THermesPacket;
256: DummyData: TPacketArray;
257: DummyArray: TPacketArray;
258: DummyVariant: OleVariant;
259: }
260: Index: Integer;
261: //AnEvent: TNotifyEvent;
262: begin
263: inherited Create;
264: Running := False;
265: Router := TRouter.Create;
266: ConnectAddress := 'Not connected';
267: LoadFromFile(SetupFile);
268: Index := IndexOfName('MediaType');
269: if Index > -1 then begin
270: MediaType := Copy(Strings[Index], 11, 255);
271: end;
272: Index := IndexOfName('Adress');
273: if Index > -1 then begin
274: Address := Copy(Strings[Index], 8, 255);
275: end;
276:
277: Transport := CreateOleObject(MediaType + '.Media');
278:
279: // Hook the sink up to the automation server (TransportLayer)
280: InterfaceConnect(Transport, DIID_IMediaEvents, Self, TransportEventsConnection);
281:
282: Transport.Start(SetupFile);
283:
284: // Do a test with a dummy packet
285: {
286: SetLength(DummyData, 1);
287: DummyData[0] := Ord('D');
288: DummyPacket := THermesPacket.Create(FloodDataStop, 'TestReceiver', 'TestSender');
289: DummyPacket.SetData(@DummyData);
290: DummyPacket.Concretize(@DummyArray);
291: ArrayToVariant(DummyArray, DummyVariant);
292: Transport.PacketTransmit(DummyVariant);
293: Transport.PacketReceive(VarArrayRef(DummyVariant), Index);
294: }
295:
296: {$IFDEF DEBUG}
297: Form6.Memo1.Lines.Add('Trying to receive packet');
298: Transport.PacketReceive(DummyPacketVariant, DummyPacketsLeft);
299: {$ENDIF}
300:
301: Running := True;
302: end;
303:
304: destructor TMediaInfo.Destroy;
305: begin
306: // Unhook the sink from the automation server (IMediaEvents)
307: InterfaceDisconnect(Transport,DIID_IMediaEvents,TransportEventsConnection);
308: inherited Destroy;
```

```
309: end;
310:
311: end.
312:
```

A.3.10 C:\projects\hermes\HermesNode.pas

```
1: unit HermesNode;
2:
3: interface
4:
5: uses
6: HermesPacket,
7: Fragments, HermesTypes, IniFiles, HermesMedia, AddressList, HermesMessage;
8:
9: Type
10:
11: TLinkState = (UN, UP, DN, DN_B, UN_W, A_BR, NULL);
12:
13: TNodeInfo = class(TObject)
14: private
15: protected
16: public
17: Name: TNodeID;
18:
19: // AddressList is a list of media units that the node supports. It is empty
20: // (nil) for a local node, since local nodes use the medialist in the
21: // Engine layer.
22: Addresses: TLocatorList;
23: //Medias: TMediaList;
24:
25: // The following are specific to the actual routing protocol used
26: // SEQ: Integer; // The sequence number for outgoing packets to this node
27: // ACK: Integer; // The highest frame acknowledged for this node
28: //QS: Integer; // LMR
29: //LS: TLinkState; // LMR
30: procedure AddAddress(MediaType: TMediaType; Address: TMediaAddress);
31: Procedure NewMessage(TheMessage: THermesMessage); virtual; abstract;
32: Procedure NewPacket(ThePacket: THermesPacket); virtual; abstract;
33: end;
34:
35: TLocalNodeInfo = class(TNodeInfo)
36: private
37: protected
38: public
39: // Rule is the pointer to the rule-layer (control layer)
40: // or nil if the node is remote. Local messages can therefore be distributed
41: // in the engine layer. For test purposes it can however be advantageous to
42: // also implement that the messages are sent over a local transport instance.
43: Rule: TObject;
44:
45: constructor Create(TheName: TNodeID; TheRule: Pointer);
46: Procedure NewMessage(TheMessage: THermesMessage); override;
47: Procedure NewPacket(ThePacket: THermesPacket); override;
48: end;
49:
50: TRemoteNodeInfo = class(TNodeInfo)
51: private
52: protected
53: // Frag contains a Fragmenter that splits messages into packets.
54: Frag: TFragmenter;
```

```
55: // DeFrag contains an defragmenter that merges packets into messages
56: DeFrag: TDeFragmenter;
57: public
58: constructor Create(TheName: TNodeID);
59: Procedure NewMessage(TheMessage: THermesMessage); override;
60: Procedure NewPacket(ThePacket: THermesPacket); override;
61: end;
62:
63:
64: TNodeInfoList = class(THashedStringList)
65: private
66: protected
67: public
68: procedure Add(NodeInfo: TNodeInfo);
69: //procedure PacketToAll(ThePacket: TPPacketArray);
70: procedure DispatchPacket(ThePacket: TPPacketArray; Receiver: TNodeID);
71: procedure DispatchMessage(ThePacket: THermesPacket);
72: procedure PacketToAll(ThePacket: THermesPacket);
73: end;
74:
75: implementation
76:
77: uses
78: EngineImpl;
79:
80: procedure TNodeInfoList.DispatchPacket(ThePacket: TPPacketArray; Receiver: TNo-
deID);
81: var
82: Index: Integer;
83: begin
84: Index := IndexOf(Receiver);
85: If Index >= 0 then
86: begin
87: TNodeInfo(Objects[Index]).NewPacket(THermesPacket.Create(ThePacket));
88: end;
89: end;
90:
91: procedure TNodeInfoList.PacketToAll(ThePacket: THermesPacket);
92: var
93: Index: Integer;
94: begin
95: for Index := 0 to count - 1 do
96: begin
97: If Assigned(Objects[Index]) then
98: begin
99: TNodeInfo(Objects[Index]).NewPacket(ThePacket);
100: end;
101: end;
102: end;
103:
104: procedure TNodeInfoList.DispatchMessage(ThePacket: THermesPacket);
105: begin
106: end;
107:
108:
109: Procedure TLocalNodeInfo.NewMessage(TheMessage: THermesMessage);
110: begin
111: if Assigned(Rule) then begin
112: // send message to rule layer
113: with Rule as THermesSession do begin
114: MessageEnqueue(TheMessage);
```

```
115: FireOnMessageReceived;
116: end;
117: end else if Assigned(Addresses) then begin
118: // send message on media
119: end else begin
120: // Error node is not local, and has no assigned media
121: end;
122: end;
123:
124: Procedure TLocalNodeInfo.NewPacket(ThePacket: THermesPacket);
125: begin
126: if Assigned(Rule) then begin
127: // Defragment packet
128: //self.
129: with Rule as THermesSession do begin
130: NewPacket(ThePacket);
131: //MessageEnqueue(TheMessage);
132: //FireOnMessageReceived;
133: end;
134: end else if Assigned(Addresses) then begin
135: // send message on media
136: end else begin
137: // Error node is not local, and has no assigned media
138: end;
139: end;
140:
141: Procedure TRemoteNodeInfo.NewMessage(TheMessage: THermesMessage);
142: begin
143: {
144: if Assigned(Rule) then begin
145: // send message to rule layer
146: with Rule as THermesSession do begin
147: MessageEnqueue(TheMessage);
148: FireOnMessageReceived;
149: end;
150: end else if Assigned(Addresses) then begin
151: // send message on media
152: end else begin
153: // Error node is not local, and has no assigned media
154: end;
155: }
156: end;
157:
158: Procedure TRemoteNodeInfo.NewPacket(ThePacket: THermesPacket);
159: begin
160: { if Assigned(Rule) then begin
161: // Defragment packet
162:
163: with Rule as THermesSession do begin
164: MessageEnqueue(TheMessage);
165: FireOnMessageReceived;
166: end;
167:
168: end else if Assigned(Addresses) then begin
169: // send message on media
170: end else begin
171: // Error node is not local, and has no assigned media
172: end;
173: }
174: end;
175:
```

```
176: procedure TNodeInfo.AddAddress(MediaType: TMediaType; Address: TMedi-
aAddress);
177: //var
178: //Done: Boolean;
179: //Index, ListMax: Integer;
180: begin
181: Addresses.AddLocator(MediaType, Address);
182: //Done := false;
183: //ListMax := High(Medias);
184: //Index := Low(Medias);
185: //while (Index <= ListMax) and not Done do begin
186: // if not Assigned(Medias[Index]) then begin
187: // Medias[Index] := TheMedia;
188: // Done := True;
189: // end;
190: // Index := Index + 1;
191: //end;
192: //if not Done then begin
193: // SetLength(Medias, Length(Medias) + 20);
194: // Medias[Index] := TheMedia;
195: //end;
196: end;
197:
198: procedure TNodeInfoList.Add(NodeInfo: TNodeInfo);
199: begin
200: AddObject(NodeInfo.Name, NodeInfo);
201: end;
202:
203: constructor TLocalNodeInfo.Create(TheName: TNodeID; TheRule: Pointer);
204: begin
205: inherited create;
206: Name := TheName;
207: Rule := TheRule;
208: //Medias := TMediaList.Create;
209: Addresses := TLocatorList.Create;
210: end;
211:
212: constructor TRemoteNodeInfo.Create(TheName: TNodeID);
213: begin
214: inherited create;
215: Frag := TFragmenter.Create(nil);
216: DeFrag := TDeFragmenter.Create(nil, nil);
217: Name := TheName;
218: Addresses := TLocatorList.Create;
219: end;
220:
221:
222: end.
```

A.3.11 C:\projects\hermes\HermesPacket.pas

```
1: unit HermesPacket;
2:
3: interface
4: uses
5: Classes, HermesTypes, Contnrs;
6:
7: type
8: {$Z2}
9: THermesPacketType = (FloodData, FloodAck, FloodDataStart, FloodDataStop, Flood-
Nak);
```

```
10: {$Z1}
11: THermesPacket = Class(TObject)
12: public
13: //function GetGUID(in out Index: Integer): TGUID;
14: //procedure PutReceiver(Receiver: AnsiString; PacketArray: TPPacketArray; in out Index:
Integer);
15: function GetReceiver: AnsiString;
16: function GetSender: AnsiString;
17: function GetPriority: Byte;
18: procedure GetData(TheData: TPPacketArray);
19: function GetType: THermesPacketType;
20: function GetSequenceNumber: LongWord;
21: procedure SetReceiver(TheReceiver: AnsiString);
22: procedure SetSender(TheSender: AnsiString);
23: procedure SetPriority(ThePriority: Byte);
24: procedure SetData(TheData: TPPacketArray);
25: procedure SetType(TheType: THermesPacketType);
26: procedure SetSequenceNumber(TheNumber: LongWord);
27: function GetEngines: TStringList;
28: procedure AddEngine(Engine: AnsiString);
29: function FindEngine(Engine: AnsiString): Boolean;
30: procedure Concretize(TheArray: TPPacketArray);
31: constructor Create(TheArray: TPPacketArray); overload;
32: constructor Create(TheType: THermesPacketType); overload;
33: constructor Create(TheType: THermesPacketType; TheReceiver, TheSender: TNo-
deID); overload;
34: protected
35: Data: TPacketArray;
36: Receiver: TNodeID;
37: Sender: TNodeID;
38: Engines: TStringList;
39: SequenceNumber: LongWord;
40: Priority: TPriority;
41: PacketType: THermesPacketType;
42:
43: function GetByteB(TheArray: TPPacketArray; out Index: Integer): Byte;
44: function GetWordB(TheArray: TPPacketArray; out Index: Integer): Word;
45: function GetLongB(TheArray: TPPacketArray; out Index: Integer): LongWord;
46: //procedure StringCopyB(out Index: Integer): AnsiString;
47: //procedure ByteCopyB(out Index: Integer; Length: Integer): TPacketArray;
48:
49: procedure PutByteB(TheByte: Byte; TheArray: TPPacketArray; out Index: Integer);
50: procedure PutWordB(TheWord: Word; TheArray: TPPacketArray; out Index: Inte-
ger);
51: procedure PutLongB(TheLong: LongWord; TheArray: TPPacketArray; out Index:
Integer);
52: //procedure PutStringB(TheString: AnsiString; out Index: Integer);
53: //procedure PutArrayB(TheData: PByte; out Index: Integer; Length: Integer);
54:
55: function GetByteF(TheArray: TPPacketArray; out Index: Integer): Byte;
56: function GetWordF(TheArray: TPPacketArray; out Index: Integer): Word;
57: function GetLongF(TheArray: TPPacketArray; out Index: Integer): LongWord;
58: //function GetStringF(out Index: Integer): AnsiString;
59: //function GetArrayF(out Index: Integer; Length: Integer): TPacketArray;
60:
61: procedure PutByteF(TheByte: Byte; TheArray: TPPacketArray; out Index: Integer);
62: procedure PutWordF(TheWord: Word; TheArray: TPPacketArray; out Index: Inte-
ger);
63: procedure PutLongF(TheLong: LongWord; TheArray: TPPacketArray; out Index:
Integer);
64: //procedure PutStringF(TheString: AnsiString; out Index: Integer);
```

```
65: //procedure PutArrayF(TheData: PByte; out Index: Integer; Length: Integer);
66:
67: {
68: procedure ByteCopyB(PBlockA, PBlockB: PByte; Count: Integer);
69: procedure StringCopyB(PBlockA, PBlockB: PByte);
70: }
71:
72: procedure ByteCopyF(PBlockA, PBlockB: PByte; Count: Integer);
73:
74: {
75: procedure StringCopyF(PBlockA, PBlockB: PByte);
76: }
77:
78: function GetStringF(TheArray: TPPacketArray; out Index: Integer): AnsiString;
79: procedure PutStringF(TheString: AnsiString; TheArray: TPPacketArray; out Index:
Integer);
80:
81: //procedure PutGUID(TheGUID: TGUID; in out Index: Integer);
82: //procedure SendToAllLocal(PacketArray: TPPacketArray);
83: protected
84: private
85: end;
86:
87: TPHermesPacket = ^THermesPacket;
88:
89: THermesPacketQueue = Class(TObjectQueue)
90: private
91: protected
92: public
93: function Peek: THermesPacket;
94: function Pop: THermesPacket;
95: procedure Push(APacket: THermesPacket);
96: end;
97:
98:
99: implementation
100:
101: uses Variants;
102:
103: function THermesPacketQueue.Peek: THermesPacket;
104: begin
105: Peek := THermesPacket(inherited Peek);
106: end;
107:
108: function THermesPacketQueue.Pop: THermesPacket;
109: begin
110: Pop := THermesPacket(inherited Pop);
111: end;
112:
113: procedure THermesPacketQueue.Push(APacket: THermesPacket);
114: begin
115: inherited Push(TObject(APacket));
116: end;
117:
118:
119: function THermesPacket.GetReceiver: AnsiString;
120: begin
121: result := Receiver;
122: end;
123:
124: function THermesPacket.GetSender: AnsiString;
```

```
125: begin
126: result := Sender;
127: end;
128:
129: function THermesPacket.GetPriority: Byte;
130: begin
131: result := Priority;
132: end;
133:
134: procedure THermesPacket.GetData(TheData: TPPacketArray);
135: var
136: Index: Integer;
137: begin
138: SetLength(TheData^, Length(Data));
139: For Index := 0 to Length(Data) - 1 do
140: begin
141: TheData^[Index] := Data[Index];
142: end;
143: end;
144:
145: function THermesPacket.GetType: THermesPacketType;
146: begin
147: result := PacketType;
148: end;
149:
150: function THermesPacket.GetSequenceNumber: LongWord;
151: begin
152: result := SequenceNumber;
153: end;
154:
155: procedure THermesPacket.SetReceiver(TheReceiver: AnsiString);
156: begin
157: Receiver := TheReceiver;
158: end;
159:
160: procedure THermesPacket.SetSender(TheSender: AnsiString);
161: begin
162: Sender := TheSender;
163: end;
164:
165: procedure THermesPacket.SetPriority(ThePriority: Byte);
166: begin
167: Priority := ThePriority;
168: end;
169:
170: procedure THermesPacket.SetData(TheData: TPPacketArray);
171: var
172: Index: Integer;
173: begin
174: SetLength(Data, Length(TheData^));
175: For Index := 0 to Length(TheData^) - 1 do
176: begin
177: Data[Index] := TheData^[Index];
178: end;
179: end;
180:
181: procedure THermesPacket.SetType(TheType: THermesPacketType);
182: begin
183: PacketType := TheType;
184: end;
185:
```

```
186: procedure THermesPacket.SetSequenceNumber(TheNumber: LongWord);
187: begin
188: SequenceNumber := TheNumber;
189: end;
190:
191: function THermesPacket.GetEngines: TStringList;
192: begin
193: result := Engines;
194: end;
195:
196: procedure THermesPacket.AddEngine(Engine: AnsiString);
197: begin
198: Engines.Add(Engine);
199: end;
200:
201: function THermesPacket.FindEngine(Engine: AnsiString): Boolean;
202: begin
203: result := Engines.IndexOf(Engine) >= 0;
204: end;
205:
206: procedure THermesPacket.Concretize(TheArray: TPPacketArray);
207: var
208: Index: Integer;
209: NoEngines: Word;
210: DataSize: LongWord;
211: begin
212: DataSize := 0;
213:
214: // THermesPacketType = (FloodData, FloodAck, FloodDataStart, FloodDataStop, FloodNak);
215:
216: case PacketType of
217: FloodData, FloodDataStart, FloodDataStop:
218: begin
219: DataSize := 13;
220: Inc(DataSize, Length(Data));
221: end;
222: FloodAck, FloodNak:
223: begin
224: DataSize := 9;
225: end;
226: else
227: Assert(false);
228: end;
229: inc(DataSize, (Length(Receiver) + 1) + (Length(Sender) + 1));
230: NoEngines := 0;
231: While NoEngines < Engines.Count do
232: begin
233: inc(DataSize, Length(Engines.Strings[NoEngines]) + 1);
234: Inc(NoEngines, 1);
235: end;
236:
237: SetLength(TheArray^, DataSize);
238:
239: Index := 0;
240: PutWordF(Word(PacketType), TheArray, Index);
241: PutWordF(Engines.Count, TheArray, Index);
242: PutLongF(SequenceNumber, TheArray, Index);
243: case PacketType of
244: FloodData, FloodDataStart, FloodDataStop:
245: begin
246: Assert(Length(Data) > 0);
```

```
247: PutLongF(Length(Data), TheArray, Index);
248: PutByteF(Priority, TheArray, Index);
249: ByteCopyF(@(Data[0]), @(TheArray^[Index]), Length(Data));
250: inc(Index, Length(Data));
251: end;
252: FloodAck, FloodNak:
253: begin
254: PutByteF(Priority, TheArray, Index);
255: end;
256: else
257: Assert(false);
258: end;
259:
260: PutStringF(Receiver, TheArray, Index);
261: PutStringF(Sender, TheArray, Index);
262:
263: NoEngines := 0;
264: While NoEngines < Engines.Count do
265: begin
266: PutStringF(Engines.Strings[NoEngines], TheArray, Index);
267: Inc(NoEngines, 1);
268: end;
269: end;
270:
271: constructor THermesPacket.Create(TheArray: TPPacketArray);
272: var
273: Index: Integer;
274: NoEngines: Word;
275: DataSize: LongWord;
276: begin
277: inherited Create;
278: Engines := TStringList.Create;
279: // Read the fixed size records
280: Index := 0;
281: PacketType := THermesPacketType(GetWordF(TheArray, Index));
282: NoEngines := GetWordF(TheArray, Index);
283: SequenceNumber := GetLongF(TheArray, Index);
284: case PacketType of
285: FloodData, FloodDataStart, FloodDataStop:
286: begin
287: DataSize := GetLongF(TheArray, Index);
288: Priority := TPriority(GetByteF(TheArray, Index));
289: // Read Data
290: SetLength(Data, DataSize);
291: ByteCopyF(@(TheArray^[Index]), @(Data[0]), DataSize);
292: inc(Index, DataSize);
293: end;
294: FloodAck, FloodNak:
295: begin
296: //DataSize := 0;
297: SetLength(Data, 0);
298: Priority := TPriority(GetByteF(TheArray, Index));
299: end;
300: else
301: Assert(false);
302: end;
303:
304: // Get sender & receiver
305: Receiver := GetStringF(TheArray, Index);
306: Sender := GetStringF(TheArray, Index);
307:
```

```
308: // Get all visited nodes
309: while NoEngines > 0 do
310: begin
311: Engines.Add(GetStringF(TheArray, Index));
312: dec(NoEngines, 1);
313: end;
314: end;
315:
316: constructor THermesPacket.Create(TheType: THermesPacketType);
317: begin
318: inherited Create;
319: SetLength(Data, 0);
320: Receiver := '';
321: Sender := '';
322: Engines := TStringList.Create;
323: SequenceNumber := 0;
324: Priority := 0;
325: PacketType := TheType;
326: end;
327:
328: constructor THermesPacket.Create(TheType: THermesPacketType; TheReceiver,
TheSender: TNodeID);
329: begin
330: inherited Create;
331: SetLength(Data, 0);
332: Receiver := TheReceiver;
333: Sender := TheSender;
334: Engines := TStringList.Create;
335: SequenceNumber := 0;
336: Priority := 0;
337: PacketType := TheType;
338: end;
339:
340:
341:
342: function THermesPacket.GetByteB(TheArray: TPPacketArray; out Index: Integer):
Byte;
343: begin
344: if Low(TheArray^) <= (Index - 1) then begin
345: result := TheArray^[Index];
346: Dec(Index, 1);
347: end else begin
348: result := 0;
349: end;
350: end;
351:
352: function THermesPacket.GetWordB(TheArray: TPPacketArray; out Index: Integer):
Word;
353: begin
354: if Low(TheArray^) <= (Index - 2) then begin
355: result := 0;
356: result := result OR TheArray^[Index];
357: result := result SHL 8;
358: result := result OR TheArray^[Index - 1];
359: Dec(Index, 2);
360: end else begin
361: result := 0;
362: end;
363: end;
364:
```

```
365: function THermesPacket.GetLongB(TheArray: TPPacketArray; out Index: Integer):
LongWord;
366: begin
367: if Low(TheArray^) <= (Index - 4) then begin
368: result := 0;
369: result := result OR TheArray^[Index];
370: result := result SHL 8;
371: result := result OR TheArray^[Index - 1];
372: result := result SHL 8;
373: result := result OR TheArray^[Index - 2];
374: result := result SHL 8;
375: result := result OR TheArray^[Index - 3];
376: Dec(Index, 4);
377: end else begin
378: result := 0;
379: end;
380: end;
381:
382: function THermesPacket.GetByteF(TheArray: TPPacketArray; out Index: Integer):
Byte;
383: begin
384: if High(TheArray^) >= (Index + 1) then begin
385: result := TheArray^[Index];
386: Inc(Index, 1);
387: end else begin
388: result := 0;
389: end;
390: end;
391:
392: function THermesPacket.GetWordF(TheArray: TPPacketArray; out Index: Integer):
Word;
393: begin
394: if High(TheArray^) >= (Index + 2) then begin
395: result := 0;
396: result := result OR TheArray^[Index];
397: result := result SHL 8;
398: result := result OR TheArray^[Index + 1];
399: Inc(Index, 2);
400: end else begin
401: result := 0;
402: end;
403: end;
404:
405: function THermesPacket.GetLongF(TheArray: TPPacketArray; out Index: Integer):
LongWord;
406: begin
407: if High(TheArray^) >= (Index + 4) then begin
408: result := 0;
409: result := result OR TheArray^[Index];
410: result := result SHL 8;
411: result := result OR TheArray^[Index + 1];
412: result := result SHL 8;
413: result := result OR TheArray^[Index + 2];
414: result := result SHL 8;
415: result := result OR TheArray^[Index + 3];
416: Inc(Index, 4);
417: end else begin
418: result := 0;
419: end;
420: end;
421:
```

```
422:
423:
424:
425: procedure THermesPacket.PutByteB(TheByte: Byte; TheArray: TPPacketArray; out
Index: Integer);
426: begin
427: TheArray^[Index] := TheByte;
428: dec(Index, 1);
429: end;
430:
431: procedure THermesPacket.PutWordB(TheWord: Word; TheArray: TPPacketArray;
out Index: Integer);
432: begin
433: TheArray^[Index - 1] := TheWord and $FF;
434: TheArray^[Index - 0] := (TheWord SHR 8) and $FF;
435: dec(Index, 2);
436: end;
437:
438: procedure THermesPacket.PutLongB(TheLong: LongWord; TheArray: TPPacketAr-
ray; out Index: Integer);
439: begin
440: TheArray^[Index - 3] := TheLong and $FF;
441: TheLong := TheLong SHR 8;
442: TheArray^[Index - 2] := TheLong and $FF;
443: TheLong := TheLong SHR 8;
444: TheArray^[Index - 1] := TheLong and $FF;
445: TheLong := TheLong SHR 8;
446: TheArray^[Index - 0] := TheLong and $FF;
447: dec(Index, 4);
448: end;
449:
450:
451: procedure THermesPacket.PutByteF(TheByte: Byte; TheArray: TPPacketArray; out
Index: Integer);
452: begin
453: TheArray^[Index] := TheByte;
454: inc(Index, 1);
455: end;
456:
457: procedure THermesPacket.PutWordF(TheWord: Word; TheArray: TPPacketArray;
out Index: Integer);
458: begin
459: TheArray^[Index + 1] := TheWord and $FF;
460: TheArray^[Index + 0] := (TheWord SHR 8) and $FF;
461: inc(Index, 2);
462: end;
463:
464: procedure THermesPacket.PutLongF(TheLong: LongWord; TheArray: TPPacketAr-
ray; out Index: Integer);
465: begin
466: TheArray^[Index + 3] := TheLong and $FF;
467: TheLong := TheLong SHR 8;
468: TheArray^[Index + 2] := TheLong and $FF;
469: TheLong := TheLong SHR 8;
470: TheArray^[Index + 1] := TheLong and $FF;
471: TheLong := TheLong SHR 8;
472: TheArray^[Index + 0] := TheLong and $FF;
473: inc(Index, 4);
474: end;
475:
```

```
476: function THermesPacket.GetStringF(TheArray: TPPacketArray; out Index: Integer):
AnsiString;
477: var
478: AByte: Byte;
479: begin
480: result := '';
481: AByte := TheArray^[Index];
482: while AByte <> 0 do
483: begin
484: result := result + Char(AByte);
485: inc(Index, 1);
486: AByte := TheArray^[Index];
487: end;
488: inc(Index, 1);
489: end;
490:
491: procedure THermesPacket.PutStringF(TheString: AnsiString; TheArray: TPPacke-
tArray; out Index: Integer);
492: var
493: StrIndex: Integer;
494: begin
495: for StrIndex := 1 to Length(TheString) do
496: begin
497: TheArray^[Index] := Byte(TheString[StrIndex]);
498: inc(Index, 1);
499: end;
500: TheArray^[Index] := 0;
501: inc(Index, 1);
502: end;
503:
504:
505: {
506: procedure THermesPacket.ByteCopyB(PBlockA, PBlockB: PByte; Count: Integer);
507: begin
508: while Count > 0 do begin
509: PBlockB^ := PBlockA^;
510: dec(PBlockB, 1);
511: inc(PBlockA, 1);
512: dec(Count, 1);
513: end;
514: end;
515:
516: procedure THermesPacket.StringCopyB(PBlockA, PBlockB: PByte);
517: var
518: AByte: Byte;
519: begin
520: AByte := 1; // Dummyvärde skilt från 0
521: while AByte <> 0 do begin
522: AByte := PBlockA^;
523: PBlockB^ := AByte;
524: dec(PBlockB, 1);
525: inc(PBlockA, 1);
526: end;
527: end;
528:
529: }
530:
531:
532: procedure THermesPacket.ByteCopyF(PBlockA, PBlockB: PByte; Count: Integer);
533: begin
534: while Count > 0 do begin
```

```
535: PBlockB^ := PBlockA^;
536: inc(PBlockB, 1);
537: inc(PBlockA, 1);
538: dec(Count, 1);
539: end;
540: end;
541:
542: {
543: procedure THermesPacket.StringCopyF(PBlockA, PBlockB: PByte);
544: var
545: AByte: Byte;
546: begin
547: AByte := 1; // Dummyvärde skilt från 0
548: while AByte <> 0 do begin
549: AByte := PBlockA^;
550: PBlockB^ := AByte;
551: inc(PBlockB, 1);
552: inc(PBlockA, 1);
553: end;
554: end;
555:
556: }
557:
558: //end.
559:
560:
561:
562: {
563: uses
564: EngineGlobals, SysUtils, Classes, HermesTypes;
565:
566:
567: type
568: }
569:
570: {
571: procedure ReverseStringCopy(PBlockA, PBlockB: PByte);
572: procedure ReverseByteCopy(PBlockA, PBlockB: PByte; Count: Integer);
573:
574:
575: implementation
576: procedure THermesPacket.SendToAllLocal(PacketArray: TPPacketArray);
577: begin
578: end;
579:
580: function THermesPacket.IsFirstVisit(Engine: AnsiString; EngineList: TStrings): Boolean;
581: begin
582: Result := (EngineList.IndexOf(Engine) >= 0);
583: end;
584:
585: function THermesPacket.GetReceiver(PacketArray: TPPacketArray; out Index: Integer):
AnsiString;
586: begin
587: while (PacketArray^[Index] <> 0) and not (Index < Low(PacketArray^)) do begin
588: Result := Result + Chr(PacketArray^[Index]);
589: Dec(Index, 1);
590: end;
591: end;
592:
593: procedure THermesPacket.PutReceiver(Receiver: AnsiString; PacketArray: TPPacketArray;
out Index: Integer);
```

```
594: var
595: StrIndex: Integer;
596: begin
597: For StrIndex := 0 to Length(Receiver) - 1 do begin
598: PacketArray^[Index] := Byte(Receiver[StrIndex]);
599: Dec(Index, 1);
600: end;
601: end;
602:
603: procedure THermesPacket.ReverseByteCopy(PBlockA, PBlockB: PByte; Count: Integer);
604: begin
605: while Count > 0 do begin
606: PBlockB^ := PBlockA^;
607: dec(PBlockB, 1);
608: dec(PBlockA, 1);
609: dec(Count, 1);
610: end;
611: end;
612:
613: procedure THermesPacket.ReverseStringCopy(PBlockA, PBlockB: PByte);
614: var
615: AByte: Byte;
616: begin
617: AByte := 1; // Dummyvärde skilt från 0
618: while AByte <> 0 do begin
619: AByte := PBlockA^;
620: PBlockB^ := AByte;
621: dec(PBlockB, 1);
622: dec(PBlockA, 1);
623: end;
624: end;
625:
626: procedure THermesPacket.AddEngine(PacketArray: TPPacketArray; Engine: AnsiString);
627: var
628: Index: Integer;
629: NoEngines: Word;
630: Receiver: AnsiString;
631: begin
632: Index := High(PacketArray^);
633: Receiver := GetReceiver(PacketArray, Index);
634: NoEngines := GetWord(PacketArray, Index);
635: SetLength(PacketArray^, Length(PacketArray^) + Length(Engine));
636:
637: Index := High(PacketArray^);
638: PutReceiver(Receiver, PacketArray, Index);
639: Inc(NoEngines, 1);
640: PutWord(NoEngines, PacketArray, Index);
641: PutReceiver(Engine, PacketArray, Index);
642: end;
643:
644: function THermesPacket.GetEngines(PacketArray: TPPacketArray; out Index: Integer):
TStringList;
645: var
646: NoEngines: Word;
647: begin
648: Result := TStringList.Create;
649: if Low(PacketArray^) >= (Index - 2) then begin
650: NoEngines := GetWord(PacketArray, Index);
651: while (NoEngines > 0) and (Index >= Low(PacketArray^)) do begin
652: Result.Add(GetReceiver(PacketArray, Index));
653: Dec(NoEngines, 1);
```

```
654: end;
655: end;
656: end;
657: }
658:
659: {
660: function THermesPacket.GetGUID(PacketArray: TPPacketArray; out Index: Integer):
TGUID;
661: var
662: AGUID: TGUID;
663: begin
664: AGUID.D1 := 0;
665: if Low(PacketArray^) >= Index - 16 then begin
666: AGUID.D1 := GetLong(PacketArray, Index);
667:
668: AGUID.D2 := GetWord(PacketArray, Index);
669:
670: AGUID.D3 := GetWord(PacketArray, Index);
671:
672: AGUID.D4[0] := PacketArray^[Index - 8];
673: AGUID.D4[1] := PacketArray^[Index - 9];
674: AGUID.D4[2] := PacketArray^[Index - 10];
675: AGUID.D4[3] := PacketArray^[Index - 11];
676: AGUID.D4[4] := PacketArray^[Index - 12];
677: AGUID.D4[5] := PacketArray^[Index - 13];
678: AGUID.D4[6] := PacketArray^[Index - 14];
679: AGUID.D4[7] := PacketArray^[Index - 15];
680: Dec(Index, 16);
681: end;
682: result := AGUID;
683: end;
684: }
685:
686: {
687: procedure THermesPacket.PutGUID(TheGUID: TGUID; PacketArray: TPPacketArray; out
Index: Integer);
688: begin
689: if Low(PacketArray^) >= Index - 16 then begin
690: PutLong(TheGUID.D1, PacketArray, Index);
691: PutWord(TheGUID.D2, PacketArray, Index);
692: PutWord(TheGUID.D3, PacketArray, Index);
693: PacketArray^[Index - 0] := TheGUID.D4[0];
694: PacketArray^[Index - 1] := TheGUID.D4[1];
695: PacketArray^[Index - 2] := TheGUID.D4[2];
696: PacketArray^[Index - 3] := TheGUID.D4[3];
697: PacketArray^[Index - 4] := TheGUID.D4[4];
698: PacketArray^[Index - 5] := TheGUID.D4[5];
699: PacketArray^[Index - 6] := TheGUID.D4[6];
700: PacketArray^[Index - 7] := TheGUID.D4[7];
701: Dec(Index, 8);
702: end;
703: end;
704: }
705:
706:
707:
708: {
709: procedure THermesPacket.PacketDispatch(ThePacket: OleVariant; TheMedia: Pointer);
710: var
711: Receiver: TNodeID;
712: Index: Integer;
```

```
713: EngineList: TStrings;
714: PacketArray: TPacketArray;
715: begin
716: // 1. check if the packets destination is connected to this engine
717: PacketArray := ThePacket;
718: Index := High(PacketArray);
719: Receiver := GetReceiver(@PacketArray, Index);
720: if IsLocalNode(Receiver) then begin
721: // Deliver packet to local fragmenter
722: FMediaList.DeliverTo(@PacketArray, Receiver);
723: end else if Receiver = 'Alla' then begin
724: // Deliver packet to all local nodes
725: FMediaList.DeliverToAll(@PacketArray);
726: // and resend over all media, but incoming
727: SendToAllLocal(@PacketArray);
728: end else begin
729: // Find List of visited engines
730: EngineList := GetEngines(@PacketArray, Index);
731: // 2. check if the packet has been to this engine before
732: if IsFirstVisit(EngineIdentity, EngineList) then begin
733: // 3. Append this Engine to the packet, and
734: // resend the packet on all media, but the incoming.
735: AddEngine(@PacketArray, EngineIdentity);
736: SendToAllLocal(@PacketArray);
737: end else begin
738: // The packet has been to this node before. Just ignore it.
739: end;
740: EngineList.Destroy; // Dispose of the EngineList
741: end;
742: end;
743: }
744:
745: end.
746:
```

A.3.12 C:\projects\hermes\HermesVariants.pas

```
1: unit HermesVariants;
2:
3: interface
4:
5: uses
6: HermesTypes, Variants;
7:
8: procedure ArrayFromVariant(var AnArray: TPacketArray; const AVariant: OleVari-
ant);
9: procedure ArrayToVariant(const AnArray: TPacketArray; var AVariant: OleVariant);
10:
11: implementation
12:
13:
14: procedure ArrayToVariant(const AnArray: TPacketArray; var AVariant: OleVariant);
15: var
16: ArrHigh: Integer;
17: Index: Integer;
18: begin
19: ArrHigh := High(AnArray);
20: AVariant := VarArrayCreate([0, ArrHigh], varByte);
21: For Index := 0 to ArrHigh do
22: begin
23: AVariant[Index] := AnArray[Index];
```

```
24: end;
25: end;
26:
27: procedure ArrayFromVariant(var AnArray: TPacketArray; const AVariant: OleVari-
ant);
28: var
29: VarHigh, VarLow: Integer;
30: Index: Integer;
31: begin
32: VarLow := VarArrayLowBound(AVariant, 1);
33: VarHigh := VarArrayHighBound(AVariant, 1);
34: SetLength(AnArray, VarHigh - VarLow + 1);
35: Index := 0;
36: while VarLow <= VarHigh do
37: begin
38: AnArray[Index] := AVariant[VarLow];
39: Inc(VarLow, 1);
40: Inc(Index, 1);
41: end;
42: end;
43:
44:
45: end.
```

A.3.13 C:\projects\HermesMedia\MediaModule_TLB.pas

```
1: unit MediaModule_TLB;
2:
3: // ************************************************************************ //
4: // WARNING
5: // -------
6: // The types declared in this file were generated from data read from a
7: // Type Library. If this type library is explicitly or indirectly (via
8: // another type library referring to this type library) re-imported, or the
9: // 'Refresh' command of the Type Library Editor activated while editing the
10: // Type Library, the contents of this file will be regenerated and all
11: // manual modifications will be lost.
12: // ************************************************************************ //
13:
14: // PASTLWTR : $Revision: 1.130 $
15: // File generated on 2002-11-13 13:13:54 from Type Library described below.
16:
17: // ************************************************************************ //
18: // Type Lib: C:\Jobb\hermes\projects\HermesMedia\mediamodule.tlb (1)
19: // LIBID: {C7B14B80-E09A-42C6-9C80-320C86755D43}
20: // LCID: 0
21: // Helpfile:
22: // DepndLst:
23: // (1) v2.0 stdole, (C:\WINNT\System32\Stdole2.tlb)
24: // Parent TypeLibrary:
25: // (0) v1.0 PMediaTCP, (C:\Jobb\hermes\projects\HermesMedia\PMediaTCP.tlb)
26: // ************************************************************************ //
27: {$TYPEDADDRESS OFF} // Unit must be compiled without type-checked pointers.
28: {$WARN SYMBOL_PLATFORM OFF}
29: {$WRITEABLECONST ON}
30:
31: interface
32:
33: uses ActiveX, Classes, Graphics, StdVCL, Variants, Windows;
34:
35:
```

```
36: // ************************************************************************//
37: // GUIDS declared in the TypeLibrary. Following prefixes are used:
38: // Type Libraries : LIBID_xxxx
39: // CoClasses : CLASS_xxxx
40: // DISPInterfaces : DIID_xxxx
41: // Non-DISP interfaces: IID_xxxx
42: // ************************************************************************//
43: const
44: // TypeLibrary Major and minor versions
45: MediaModuleMajorVersion = 1;
46: MediaModuleMinorVersion = 0;
47:
48: LIBID_MediaModule: TGUID = '{C7B14B80-E09A-42C6-9C80-320C86755D43}';
49:
50: IID_IMedia: TGUID = '{3E4422B5-937C-4367-84ED-C64B7CAEA907}';
51: DIID_IMediaEvents: TGUID = '{A92E67BB-F3C6-4497-996F-6F61DD891E99}';
52: type
53:
54: // ************************************************************************//
55: // Forward declaration of types defined in TypeLibrary
56: // ************************************************************************//
57: IMedia = interface;
58: IMediaDisp = dispinterface;
59: IMediaEvents = dispinterface;
60:
61: // ************************************************************************//
62: // Declaration of structures, unions and aliases.
63: // ************************************************************************//
64: PInteger1 = ^Integer; {*}
65:
66:
67: // ************************************************************************//
68: // Interface: IMedia
69: // Flags: (4416) Dual OleAutomation Dispatchable
70: // GUID: {3E4422B5-937C-4367-84ED-C64B7CAEA907}
71: // ************************************************************************//
72: IMedia = interface(IDispatch)
73: ['{3E4422B5-937C-4367-84ED-C64B7CAEA907}']
74: procedure ConnectionOpen(ContactInfo: OleVariant); safecall;
75: procedure ConnectionClose; safecall;
76: procedure PacketTransmit(ThePacket: OleVariant); safecall;
77: procedure PacketReceive(out ThePacket: OleVariant; out QueueLength: Integer);
safecall;
78: procedure QueueGetTransmitSize(out Size: Integer); safecall;
79: procedure QueueGetReceiveSize(out Size: Integer); safecall;
80: procedure QueueGetTransmitMaxSize(out Size: Integer); safecall;
81: procedure QueueGetReceiveMaxSize(out Size: Integer); safecall;
82: procedure QueueSetTransmitMaxSize(var Size: Integer); safecall;
83: procedure QueueSetReceiveMaxSize(var Size: Integer); safecall;
84: procedure PacketGetMaxSize(out Size: Integer); safecall;
85: procedure ConnectionGetAddress(out Info: OleVariant); safecall;
86: procedure ConnectionGetStatus(out ConnectionStatus: Integer); safecall;
87: procedure Start(const ConfigFile: WideString); safecall;
88: procedure Stop; safecall;
89: end;
90:
91: // ************************************************************************//
92: // DispIntf: IMediaDisp
93: // Flags: (4416) Dual OleAutomation Dispatchable
94: // GUID: {3E4422B5-937C-4367-84ED-C64B7CAEA907}
95: // ************************************************************************//
```

```
96: IMediaDisp = dispinterface
97: ['{3E4422B5-937C-4367-84ED-C64B7CAEA907}']
98: procedure ConnectionOpen(ContactInfo: OleVariant); dispid 1610743808;
99: procedure ConnectionClose; dispid 2;
100: procedure PacketTransmit(ThePacket: OleVariant); dispid 3;
101: procedure PacketReceive(out ThePacket: OleVariant; out QueueLength: Integer);
dispid 4;
102: procedure QueueGetTransmitSize(out Size: Integer); dispid 5;
103: procedure QueueGetReceiveSize(out Size: Integer); dispid 6;
104: procedure QueueGetTransmitMaxSize(out Size: Integer); dispid 7;
105: procedure QueueGetReceiveMaxSize(out Size: Integer); dispid 8;
106: procedure QueueSetTransmitMaxSize(var Size: Integer); dispid 9;
107: procedure QueueSetReceiveMaxSize(var Size: Integer); dispid 10;
108: procedure PacketGetMaxSize(out Size: Integer); dispid 11;
109: procedure ConnectionGetAddress(out Info: OleVariant); dispid 12;
110: procedure ConnectionGetStatus(out ConnectionStatus: Integer); dispid 13;
111: procedure Start(const ConfigFile: WideString); dispid 14;
112: procedure Stop; dispid 15;
113: end;
114:
115: // ************************************************************************//
116: // DispIntf: IMediaEvents
117: // Flags: (4096) Dispatchable
118: // GUID: {A92E67BB-F3C6-4497-996F-6F61DD891E99}
119: // ************************************************************************//
120: IMediaEvents = dispinterface
121: ['{A92E67BB-F3C6-4497-996F-6F61DD891E99}']
122: procedure OnPacketReceived; dispid 1;
123: procedure OnLinkLost; dispid 2;
124: procedure OnLinkRequest; dispid 3;
125: end;
126:
127: implementation
128:
129: uses ComObj;
130:
131: end.
```

A.3.14 C:\projects\hermes\PEngine_TLB.pas

```
1: unit PEngine_TLB;
2:
3: // ************************************************************************ //
4: // WARNING
5: // -------
6: // The types declared in this file were generated from data read from a
7: // Type Library. If this type library is explicitly or indirectly (via
8: // another type library referring to this type library) re-imported, or the
9: // 'Refresh' command of the Type Library Editor activated while editing the
10: // Type Library, the contents of this file will be regenerated and all
11: // manual modifications will be lost.
12: // ************************************************************************ //
13:
14: // PASTLWTR : $Revision: 1.130 $
15: // File generated on 2002-04-19 12:18:43 from Type Library described below.
16:
17: // ************************************************************************ //
18: // Type Lib: C:\projects\hermes\PEngine.exe (1)
19: // LIBID: {1E4D5EFF-9BE5-425D-99F8-98A9C9660CC1}
20: // LCID: 0
21: // Helpfile:
```

```
22: // DepndLst:
23: // (1) v2.0 stdole, (C:\WINNT\System32\stdole2.tlb)
24: // (2) v4.0 StdVCL, (C:\WINNT\System32\STDVCL40.DLL)
25: // ************************************************************************ //
26: // ************************************************************************//
27: // NOTE:
28: // Items guarded by $IFDEF_LIVE_SERVER_AT_DESIGN_TIME are used by properties
29: // which return objects that may need to be explicitly created via a function
30: // call prior to any access via the property. These items have been disabled
31: // in order to prevent accidental use from within the object inspector. You
32: // may enable them by defining LIVE_SERVER_AT_DESIGN_TIME or by selectively
33: // removing them from the $IFDEF blocks. However, such items must still be
34: // programmatically created via a method of the appropriate CoClass before
35: // they can be used.
36: {$TYPEDADDRESS OFF} // Unit must be compiled without type-checked pointers.
37: {$WARN SYMBOL_PLATFORM OFF}
38: {$WRITEABLECONST ON}
39:
40: interface
41:
42: uses ActiveX, Classes, Graphics, OleServer, StdVCL, Variants, Windows;
43:
44:
45:
46: // ************************************************************************//
47: // GUIDS declared in the TypeLibrary. Following prefixes are used:
48: // Type Libraries : LIBID_xxxx
49: // CoClasses : CLASS_xxxx
50: // DISPInterfaces : DIID_xxxx
51: // Non-DISP interfaces: IID_xxxx
52: // ************************************************************************//
53: const
54: // TypeLibrary Major and minor versions
55: PEngineMajorVersion = 1;
56: PEngineMinorVersion = 0;
57:
58: LIBID_PEngine: TGUID = '{1E4D5EFF-9BE5-425D-99F8-98A9C9660CC1}';
59:
60: IID_IHermesSession: TGUID = '{0AFD8346-5645-456D-AFD7-156E204C3CFA}';
61: DIID_IHermesSessionEvents: TGUID = '{F40D541A-117B-47C7-99CE-
47286DFB228E}';
62: CLASS_HermesSession: TGUID = '{545A3789-5233-44C5-9EFF-3BEB0A23BD50}';
63: type
64:
65: // ************************************************************************//
66: // Forward declaration of types defined in TypeLibrary
67: // ************************************************************************//
68: IHermesSession = interface;
69: IHermesSessionDisp = dispinterface;
70: IHermesSessionEvents = dispinterface;
71:
72: // ************************************************************************//
73: // Declaration of CoClasses defined in Type Library
74: // (NOTE: Here we map each CoClass to its Default Interface)
75: // ************************************************************************//
76: HermesSession = IHermesSession;
77:
78:
79: // ************************************************************************//
80: // Interface: IHermesSession
81: // Flags: (4416) Dual OleAutomation Dispatchable
```

```
82: // GUID: {0AFD8346-5645-456D-AFD7-156E204C3CFA}
83: // *********************************************************************//
84: IHermesSession = interface(IDispatch)
85: ['{0AFD8346-5645-456D-AFD7-156E204C3CFA}']
86: procedure MediaStart(const Config: WideString); safecall;
87: procedure MediaStop; safecall;
88: procedure MessageSend(TheMessage: OleVariant); safecall;
89: procedure MessageReceive(out TheMessage: OleVariant; out NoMessages: Integer);
safecall;
90: procedure QueueGetTransmitSize; safecall;
91: procedure QueueGetReceiveSize; safecall;
92: procedure QueueGetTransmitMaxSize; safecall;
93: procedure QueueGetReceiveMaxSize; safecall;
94: procedure QueueSetTransmitMaxSize; safecall;
95: procedure QueueSetReceiveMaxSize; safecall;
96: procedure NodeAdd(const Node: WideString); safecall;
97: procedure NodeRemove(const Node: WideString); safecall;
98: procedure NodeMediaAdd(const Node: WideString; const MediaType: WideString;
99: const Address: WideString; const Port: WideString); safecall;
100: procedure NodeMediaRemove(const Node: WideString; const MediaType:
WideString;
101: const Address: WideString); safecall;
102: procedure Open(const Node: WideString); safecall;
103: procedure Close; safecall;
104: procedure MediaStatus(const MediaType: WideString; const MediaAdress:
WideString;
105: out MediaStatus: SYSINT); safecall;
106: procedure MediaConnectionAddress(const MediaType: WideString; const Medi-
aAddress: WideString;
107: out ConnectionAddress: WideString); safecall;
108: procedure MediaConnect(const MediaType: WideString; const MediaAddress:
WideString;
109: const ConnectionAddress: WideString); safecall;
110: procedure MediaDisconnect(const MediaType: WideString; const MediaAddress:
WideString); safecall;
111: procedure MediaInventory(out MediaList: OleVariant); safecall;
112: end;
113:
114: // *********************************************************************//
115: // DispIntf: IHermesSessionDisp
116: // Flags: (4416) Dual OleAutomation Dispatchable
117: // GUID: {0AFD8346-5645-456D-AFD7-156E204C3CFA}
118: // *********************************************************************//
119: IHermesSessionDisp = dispinterface
120: ['{0AFD8346-5645-456D-AFD7-156E204C3CFA}']
121: procedure MediaStart(const Config: WideString); dispid 1610743808;
122: procedure MediaStop; dispid 2;
123: procedure MessageSend(TheMessage: OleVariant); dispid 3;
124: procedure MessageReceive(out TheMessage: OleVariant; out NoMessages: Integer);
dispid 4;
125: procedure QueueGetTransmitSize; dispid 6;
126: procedure QueueGetReceiveSize; dispid 7;
127: procedure QueueGetTransmitMaxSize; dispid 8;
128: procedure QueueGetReceiveMaxSize; dispid 9;
129: procedure QueueSetTransmitMaxSize; dispid 10;
130: procedure QueueSetReceiveMaxSize; dispid 11;
131: procedure NodeAdd(const Node: WideString); dispid 5;
132: procedure NodeRemove(const Node: WideString); dispid 12;
133: procedure NodeMediaAdd(const Node: WideString; const MediaType: WideString;
134: const Address: WideString; const Port: WideString); dispid 13;
```

```
135: procedure NodeMediaRemove(const Node: WideString; const MediaType:
WideString;
136: const Address: WideString); dispid 14;
137: procedure Open(const Node: WideString); dispid 15;
138: procedure Close; dispid 17;
139: procedure MediaStatus(const MediaType: WideString; const MediaAdress:
WideString;
140: out MediaStatus: SYSINT); dispid 1;
141: procedure MediaConnectionAddress(const MediaType: WideString; const Medi-
aAddress: WideString;
142: out ConnectionAddress: WideString); dispid 16;
143: procedure MediaConnect(const MediaType: WideString; const MediaAddress:
WideString;
144: const ConnectionAddress: WideString); dispid 18;
145: procedure MediaDisconnect(const MediaType: WideString; const MediaAddress:
WideString); dispid 19;
146: procedure MediaInventory(out MediaList: OleVariant); dispid 20;
147: end;
148:
149: // ************************************************************************//
150: // DispIntf: IHermesSessionEvents
151: // Flags: (4096) Dispatchable
152: // GUID: {F40D541A-117B-47C7-99CE-47286DFB228E}
153: // ************************************************************************//
154: IHermesSessionEvents = dispinterface
155: ['{F40D541A-117B-47C7-99CE-47286DFB228E}']
156: procedure OnMessageAck; dispid 1;
157: procedure OnMessageNak; dispid 2;
158: procedure OnMessageReceived; dispid 3;
159: procedure OnContactLost; dispid 4;
160: procedure OnLinkLost; dispid 5;
161: end;
162:
163: // ************************************************************************//
164: // The Class CoHermesSession provides a Create and CreateRemote method to
165: // create instances of the default interface IHermesSession exposed by
166: // the CoClass HermesSession. The functions are intended to be used by
167: // clients wishing to automate the CoClass objects exposed by the
168: // server of this typelibrary.
169: // ************************************************************************//
170: CoHermesSession = class
171: class function Create: IHermesSession;
172: class function CreateRemote(const MachineName: string): IHermesSession;
173: end;
174:
175:
176: // ************************************************************************//
177: // OLE Server Proxy class declaration
178: // Server Object : THermesSession
179: // Help String : HermesSession Object
180: // Default Interface: IHermesSession
181: // Def. Intf. DISP? : No
182: // Event Interface: IHermesSessionEvents
183: // TypeFlags : (2) CanCreate
184: // ************************************************************************//
185: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
186: THermesSessionProperties= class;
187: {$ENDIF}
188: THermesSession = class(TOleServer)
189: private
190: FOnMessageAck: TNotifyEvent;
```

```
191: FOnMessageNak: TNotifyEvent;
192: FOnMessageReceived: TNotifyEvent;
193: FOnContactLost: TNotifyEvent;
194: FOnLinkLost: TNotifyEvent;
195: FIntf: IHermesSession;
196: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
197: FProps: THermesSessionProperties;
198: function GetServerProperties: THermesSessionProperties;
199: {$ENDIF}
200: function GetDefaultInterface: IHermesSession;
201: protected
202: procedure InitServerData; override;
203: procedure InvokeEvent(DispID: TDispID; var Params: TVariantArray); override;
204: public
205: constructor Create(AOwner: TComponent); override;
206: destructor Destroy; override;
207: procedure Connect; override;
208: procedure ConnectTo(svrIntf: IHermesSession);
209: procedure Disconnect; override;
210: procedure MediaStart(const Config: WideString);
211: procedure MediaStop;
212: procedure MessageSend(TheMessage: OleVariant);
213: procedure MessageReceive(out TheMessage: OleVariant; out NoMessages: Integer);
214: procedure QueueGetTransmitSize;
215: procedure QueueGetReceiveSize;
216: procedure QueueGetTransmitMaxSize;
217: procedure QueueGetReceiveMaxSize;
218: procedure QueueSetTransmitMaxSize;
219: procedure QueueSetReceiveMaxSize;
220: procedure NodeAdd(const Node: WideString);
221: procedure NodeRemove(const Node: WideString);
222: procedure NodeMediaAdd(const Node: WideString; const MediaType: WideString;
223: const Address: WideString; const Port: WideString);
224: procedure NodeMediaRemove(const Node: WideString; const MediaType:
WideString;
225: const Address: WideString);
226: procedure Open(const Node: WideString);
227: procedure Close;
228: procedure MediaStatus(const MediaType: WideString; const MediaAdress:
WideString;
229: out MediaStatus: SYSINT);
230: procedure MediaConnectionAddress(const MediaType: WideString; const Media-
aAddress: WideString;
231: out ConnectionAddress: WideString);
232: procedure MediaConnect(const MediaType: WideString; const MediaAddress:
WideString;
233: const ConnectionAddress: WideString);
234: procedure MediaDisconnect(const MediaType: WideString; const MediaAddress:
WideString);
235: procedure MediaInventory(out MediaList: OleVariant);
236: property DefaultInterface: IHermesSession read GetDefaultInterface;
237: published
238: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
239: property Server: THermesSessionProperties read GetServerProperties;
240: {$ENDIF}
241: property OnMessageAck: TNotifyEvent read FOnMessageAck write FOnMessa-
geAck;
242: property OnMessageNak: TNotifyEvent read FOnMessageNak write FOnMessa-
geNak;
243: property OnMessageReceived: TNotifyEvent read FOnMessageReceived write FOn-
MessageReceived;
```

```
244: property OnContactLost: TNotifyEvent read FOnContactLost write FOnContact-
Lost;
245: property OnLinkLost: TNotifyEvent read FOnLinkLost write FOnLinkLost;
246: end;
247:
248: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
249: // *********************************************************************//
250: // OLE Server Properties Proxy Class
251: // Server Object : THermesSession
252: // (This object is used by the IDE's Property Inspector to allow editing
253: // of the properties of this server)
254: // *********************************************************************//
255: THermesSessionProperties = class(TPersistent)
256: private
257: FServer: THermesSession;
258: function GetDefaultInterface: IHermesSession;
259: constructor Create(AServer: THermesSession);
260: protected
261: public
262: property DefaultInterface: IHermesSession read GetDefaultInterface;
263: published
264: end;
265: {$ENDIF}
266:
267:
268: procedure Register;
269:
270: resourcestring
271: dtlServerPage = 'Hermes';
272:
273: implementation
274:
275: uses ComObj;
276:
277: class function CoHermesSession.Create: IHermesSession;
278: begin
279: Result := CreateComObject(CLASS_HermesSession) as IHermesSession;
280: end;
281:
282: class function CoHermesSession.CreateRemote(const MachineName: string): IHer-
mesSession;
283: begin
284: Result := CreateRemoteComObject(MachineName, CLASS_HermesSession) as IHer-
mesSession;
285: end;
286:
287: procedure THermesSession.InitServerData;
288: const
289: CServerData: TServerData = (
290: ClassID: '{545A3789-5233-44C5-9EFF-3BEB0A23BD50}';
291: IntfIID: '{0AFD8346-5645-456D-AFD7-156E204C3CFA}';
292: EventIID: '{F40D541A-117B-47C7-99CE-47286DFB228E}';
293: LicenseKey: nil;
294: Version: 500);
295: begin
296: ServerData := @CServerData;
297: end;
298:
299: procedure THermesSession.Connect;
300: var
301: punk: IUnknown;
```

```
302: begin
303: if FIntf = nil then
304: begin
305: punk := GetServer;
306: ConnectEvents(punk);
307: Fintf:= punk as IHermesSession;
308: end;
309: end;
310:
311: procedure THermesSession.ConnectTo(svrIntf: IHermesSession);
312: begin
313: Disconnect;
314: FIntf := svrIntf;
315: ConnectEvents(FIntf);
316: end;
317:
318: procedure THermesSession.DisConnect;
319: begin
320: if Fintf <> nil then
321: begin
322: DisconnectEvents(FIntf);
323: FIntf := nil;
324: end;
325: end;
326:
327: function THermesSession.GetDefaultInterface: IHermesSession;
328: begin
329: if FIntf = nil then
330: Connect;
331: Assert(FIntf <> nil, 'DefaultInterface is NULL. Component is not connected to Server. You must call ''Connect'' or ''ConnectTo'' before this operation');
332: Result := FIntf;
333: end;
334:
335: constructor THermesSession.Create(AOwner: TComponent);
336: begin
337: inherited Create(AOwner);
338: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
339: FProps := THermesSessionProperties.Create(Self);
340: {$ENDIF}
341: end;
342:
343: destructor THermesSession.Destroy;
344: begin
345: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
346: FProps.Free;
347: {$ENDIF}
348: inherited Destroy;
349: end;
350:
351: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
352: function THermesSession.GetServerProperties: THermesSessionProperties;
353: begin
354: Result := FProps;
355: end;
356: {$ENDIF}
357:
358: procedure THermesSession.InvokeEvent(DispID: TDispID; var Params: TVariantArray);
359: begin
360: case DispID of
```

```
361: -1: Exit; // DISPID_UNKNOWN
362: 1: if Assigned(FOnMessageAck) then
363: FOnMessageAck(Self);
364: 2: if Assigned(FOnMessageNak) then
365: FOnMessageNak(Self);
366: 3: if Assigned(FOnMessageReceived) then
367: FOnMessageReceived(Self);
368: 4: if Assigned(FOnContactLost) then
369: FOnContactLost(Self);
370: 5: if Assigned(FOnLinkLost) then
371: FOnLinkLost(Self);
372: end; {case DispID}
373: end;
374:
375: procedure THermesSession.MediaStart(const Config: WideString);
376: begin
377: DefaultInterface.MediaStart(Config);
378: end;
379:
380: procedure THermesSession.MediaStop;
381: begin
382: DefaultInterface.MediaStop;
383: end;
384:
385: procedure THermesSession.MessageSend(TheMessage: OleVariant);
386: begin
387: DefaultInterface.MessageSend(TheMessage);
388: end;
389:
390: procedure THermesSession.MessageReceive(out TheMessage: OleVariant; out
NoMessages: Integer);
391: begin
392: DefaultInterface.MessageReceive(TheMessage, NoMessages);
393: end;
394:
395: procedure THermesSession.QueueGetTransmitSize;
396: begin
397: DefaultInterface.QueueGetTransmitSize;
398: end;
399:
400: procedure THermesSession.QueueGetReceiveSize;
401: begin
402: DefaultInterface.QueueGetReceiveSize;
403: end;
404:
405: procedure THermesSession.QueueGetTransmitMaxSize;
406: begin
407: DefaultInterface.QueueGetTransmitMaxSize;
408: end;
409:
410: procedure THermesSession.QueueGetReceiveMaxSize;
411: begin
412: DefaultInterface.QueueGetReceiveMaxSize;
413: end;
414:
415: procedure THermesSession.QueueSetTransmitMaxSize;
416: begin
417: DefaultInterface.QueueSetTransmitMaxSize;
418: end;
419:
420: procedure THermesSession.QueueSetReceiveMaxSize;
```

```
421: begin
422: DefaultInterface.QueueSetReceiveMaxSize;
423: end;
424:
425: procedure THermesSession.NodeAdd(const Node: WideString);
426: begin
427: DefaultInterface.NodeAdd(Node);
428: end;
429:
430: procedure THermesSession.NodeRemove(const Node: WideString);
431: begin
432: DefaultInterface.NodeRemove(Node);
433: end;
434:
435: procedure THermesSession.NodeMediaAdd(const Node: WideString; const Media-
Type: WideString;
436: const Address: WideString; const Port: WideString);
437: begin
438: DefaultInterface.NodeMediaAdd(Node, MediaType, Address, Port);
439: end;
440:
441: procedure THermesSession.NodeMediaRemove(const Node: WideString; const
MediaType: WideString;
442: const Address: WideString);
443: begin
444: DefaultInterface.NodeMediaRemove(Node, MediaType, Address);
445: end;
446:
447: procedure THermesSession.Open(const Node: WideString);
448: begin
449: DefaultInterface.Open(Node);
450: end;
451:
452: procedure THermesSession.Close;
453: begin
454: DefaultInterface.Close;
455: end;
456:
457: procedure THermesSession.MediaStatus(const MediaType: WideString; const Medi-
aAdress: WideString;
458: out MediaStatus: SYSINT);
459: begin
460: DefaultInterface.MediaStatus(MediaType, MediaAdress, MediaStatus);
461: end;
462:
463: procedure THermesSession.MediaConnectionAddress(const MediaType:
WideString;
464: const MediaAddress: WideString;
465: out ConnectionAddress: WideString);
466: begin
467: DefaultInterface.MediaConnectionAddress(MediaType, MediaAddress, Connectio-
nAddress);
468: end;
469:
470: procedure THermesSession.MediaConnect(const MediaType: WideString; const
MediaAddress: WideString;
471: const ConnectionAddress: WideString);
472: begin
473: DefaultInterface.MediaConnect(MediaType, MediaAddress, ConnectionAddress);
474: end;
475:
```

```
476: procedure THermesSession.MediaDisconnect(const MediaType: WideString; const
MediaAddress: WideString);
477: begin
478: DefaultInterface.MediaDisconnect(MediaType, MediaAddress);
479: end;
480:
481: procedure THermesSession.MediaInventory(out MediaList: OleVariant);
482: begin
483: DefaultInterface.MediaInventory(MediaList);
484: end;
485:
486: {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
487: constructor THermesSessionProperties.Create(AServer: THermesSession);
488: begin
489: inherited Create;
490: FServer := AServer;
491: end;
492:
493: function THermesSessionProperties.GetDefaultInterface: IHermesSession;
494: begin
495: Result := FServer.DefaultInterface;
496: end;
497:
498: {$ENDIF}
499:
500: procedure Register;
501: begin
502: RegisterComponents(dtlServerPage, [THermesSession]);
503: end;
504:
505: end.
```

A.3.15 C:\projects\hermes\Routing.pas

```
1: // The router determines wheter packets should be retransmitted on other media
2: // or if they should be assembled into a message at this node.
3:
4: unit Routing;
5:
6: interface
7:
8: uses
9: HermesPacket, SysUtils, Classes, HermesTypes;
10:
11:
12: type
13:
14: TRouter = Class(TObject)
15: public
16: procedure PacketDispatch(ThePacket: THermesPacket; TheMedia: Pointer);
17: protected
18: function IsFirstVisit(Engine: AnsiString; EngineList: TStrings): Boolean;
19: {
20: function GetGUID(PacketArray: TPPacketArray; out Index: Integer): TGUID;
21: function GetReceiver(PacketArray: TPPacketArray; out Index: Integer): AnsiString;
22: procedure PutReceiver(Receiver: AnsiString; PacketArray: TPPacketArray; out Index: Inte-
ger);
23: function GetEngines(PacketArray: TPPacketArray; out Index: Integer): TStringList;
24: procedure AddEngine(PacketArray: TPPacketArray; Engine: AnsiString);
25: function GetWord(PacketArray: TPPacketArray; out Index: Integer): Word;
26: function GetLong(PacketArray: TPPacketArray; out Index: Integer): LongWord;
```

```
27: procedure PutWord(TheWord: Word; PacketArray: TPPacketArray; out Index: Integer);
28: procedure PutLong(TheLong: LongWord; PacketArray: TPPacketArray; out Index: Integer);
29: procedure PutGUID(TheGUID: TGUID; PacketArray: TPPacketArray; out Index: Integer);
30: }
31: procedure SendToAllLocal(ThePacket: THermesPacket);
32: private
33: end;
34:
35: {
36: procedure ReverseStringCopy(PBlockA, PBlockB: PByte);
37: procedure ReverseByteCopy(PBlockA, PBlockB: PByte; Count: Integer);
38: }
39:
40: implementation
41:
42: uses
43: EngineGlobals;
44:
45: procedure TRouter.SendToAllLocal(ThePacket: THermesPacket);
46: begin
47: FNodeList.PacketToAll(ThePacket);
48: end;
49:
50: function TRouter.IsFirstVisit(Engine: AnsiString; EngineList: TStrings): Boolean;
51: begin
52: Result := (EngineList.IndexOf(Engine) < 0);
53: end;
54: {
55: function TRouter.GetReceiver(PacketArray: TPPacketArray; out Index: Integer): AnsiString;
56: begin
57: while (PacketArray^[Index] <> 0) and not (Index < Low(PacketArray^)) do begin
58: Result := Result + Chr(PacketArray^[Index]);
59: Dec(Index, 1);
60: end;
61: end;
62:
63: procedure TRouter.PutReceiver(Receiver: AnsiString; PacketArray: TPPacketArray; out
Index: Integer);
64: var
65: StrIndex: Integer;
66: begin
67: For StrIndex := 0 to Length(Receiver) - 1 do begin
68: PacketArray^[Index] := Byte(Receiver[StrIndex]);
69: Dec(Index, 1);
70: end;
71: end;
72:
73: procedure ReverseByteCopy(PBlockA, PBlockB: PByte; Count: Integer);
74: begin
75: while Count > 0 do begin
76: PBlockB^ := PBlockA^;
77: dec(PBlockB, 1);
78: dec(PBlockA, 1);
79: dec(Count, 1);
80: end;
81: end;
82:
83: procedure ReverseStringCopy(PBlockA, PBlockB: PByte);
84: var
85: AByte: Byte;
86: begin
```

```
87: AByte := 1; // Dummyvärde skilt från 0
88: while AByte <> 0 do begin
89: AByte := PBlockA^;
90: PBlockB^ := AByte;
91: dec(PBlockB, 1);
92: dec(PBlockA, 1);
93: end;
94: end;
95:
96: procedure TRouter.AddEngine(PacketArray: TPPacketArray; Engine: AnsiString);
97: var
98: Index: Integer;
99: NoEngines: Word;
100: Receiver: AnsiString;
101: begin
102: Index := High(PacketArray^);
103: Receiver := GetReceiver(PacketArray, Index);
104: NoEngines := GetWord(PacketArray, Index);
105: SetLength(PacketArray^, Length(PacketArray^) + Length(Engine));
106:
107: Index := High(PacketArray^);
108: PutReceiver(Receiver, PacketArray, Index);
109: Inc(NoEngines, 1);
110: PutWord(NoEngines, PacketArray, Index);
111: PutReceiver(Engine, PacketArray, Index);
112: end;
113:
114: function TRouter.GetEngines(PacketArray: TPPacketArray; out Index: Integer): TString-
List;
115: var
116: NoEngines: Word;
117: begin
118: Result := TStringList.Create;
119: if Low(PacketArray^) >= (Index - 2) then begin
120: NoEngines := GetWord(PacketArray, Index);
121: while (NoEngines > 0) and (Index >= Low(PacketArray^)) do begin
122: Result.Add(GetReceiver(PacketArray, Index));
123: Dec(NoEngines, 1);
124: end;
125: end;
126: end;
127:
128:
129:
130: function TRouter.GetGUID(PacketArray: TPPacketArray; out Index: Integer): TGUID;
131: var
132: AGUID: TGUID;
133: begin
134: AGUID.D1 := 0;
135: if Low(PacketArray^) >= Index - 16 then begin
136: AGUID.D1 := GetLong(PacketArray, Index);
137:
138: AGUID.D2 := GetWord(PacketArray, Index);
139:
140: AGUID.D3 := GetWord(PacketArray, Index);
141:
142: AGUID.D4[0] := PacketArray^[Index - 8];
143: AGUID.D4[1] := PacketArray^[Index - 9];
144: AGUID.D4[2] := PacketArray^[Index - 10];
145: AGUID.D4[3] := PacketArray^[Index - 11];
146: AGUID.D4[4] := PacketArray^[Index - 12];
```

```
147: AGUID.D4[5] := PacketArray^[Index - 13];
148: AGUID.D4[6] := PacketArray^[Index - 14];
149: AGUID.D4[7] := PacketArray^[Index - 15];
150: Dec(Index, 16);
151: end;
152: result := AGUID;
153: end;
154:
155: procedure TRouter.PutGUID(TheGUID: TGUID; PacketArray: TPPacketArray; out Index:
Integer);
156: begin
157: if Low(PacketArray^) >= Index - 16 then begin
158: PutLong(TheGUID.D1, PacketArray, Index);
159: PutWord(TheGUID.D2, PacketArray, Index);
160: PutWord(TheGUID.D3, PacketArray, Index);
161: PacketArray^[Index - 0] := TheGUID.D4[0];
162: PacketArray^[Index - 1] := TheGUID.D4[1];
163: PacketArray^[Index - 2] := TheGUID.D4[2];
164: PacketArray^[Index - 3] := TheGUID.D4[3];
165: PacketArray^[Index - 4] := TheGUID.D4[4];
166: PacketArray^[Index - 5] := TheGUID.D4[5];
167: PacketArray^[Index - 6] := TheGUID.D4[6];
168: PacketArray^[Index - 7] := TheGUID.D4[7];
169: Dec(Index, 8);
170: end;
171: end;
172:
173: function TRouter.GetWord(PacketArray: TPPacketArray; out Index: Integer): Word;
174: begin
175: if Low(PacketArray^) >= (Index - 2) then begin
176: result := 0;
177: result := result OR PacketArray^[Index];
178: result := result SHL 8;
179: result := result OR PacketArray^[Index - 1];
180: Dec(Index, 2);
181: end else begin
182: result := 0;
183: end;
184: end;
185:
186: function TRouter.GetLong(PacketArray: TPPacketArray; out Index: Integer): LongWord;
187: begin
188: if Low(PacketArray^) >= (Index - 4) then begin
189: result := 0;
190: result := result OR PacketArray^[Index];
191: result := result SHL 8;
192: result := result OR PacketArray^[Index - 1];
193: result := result SHL 8;
194: result := result OR PacketArray^[Index - 2];
195: result := result SHL 8;
196: result := result OR PacketArray^[Index - 3];
197: Dec(Index, 4);
198: end else begin
199: result := 0;
200: end;
201: end;
202:
203: procedure TRouter.PutWord(TheWord: Word; PacketArray: TPPacketArray; out Index:
Integer);
204: begin
205: PacketArray^[Index - 1] := TheWord and $FF;
```

```
206: PacketArray^[Index - 0] := (TheWord SHL 8) and $FF;
207: dec(Index, 2);
208: end;
209:
210: procedure TRouter.PutLong(TheLong: LongWord; PacketArray: TPPacketArray; out Index:
Integer);
211: begin
212: PacketArray^[Index - 3] := TheLong and $FF;
213: TheLong := TheLong SHL 8;
214: PacketArray^[Index - 2] := TheLong and $FF;
215: TheLong := TheLong SHL 8;
216: PacketArray^[Index - 1] := TheLong and $FF;
217: TheLong := TheLong SHL 8;
218: PacketArray^[Index - 0] := TheLong and $FF;
219: dec(Index, 4);
220: end;
221:
222: }
223:
224:
225: procedure TRouter.PacketDispatch(ThePacket: THermesPacket; TheMedia: Pointer);
226: var
227: Receiver: TNodeID;
228: // Index: Integer;
229: EngineList: TStrings;
230: PacketArray: TPacketArray;
231: begin
232: // 1. check if the packets destination is connected to this engine
233: //PacketArray := ThePacket;
234: //Index := High(PacketArray);
235: ThePacket.Concretize(@PacketArray);
236: Receiver := ThePacket.GetReceiver; // GetReceiver(@PacketArray, Index);
237: if IsLocalNode(Receiver) then
238: begin
239: // Deliver packet to local fragmenter
240: FNodeList.DispatchPacket(@PacketArray, Receiver);
241: end
242: else
243: begin
244: // Find List of visited engines
245: EngineList := ThePacket.GetEngines; // GetEngines(@PacketArray, Index);
246: // 2. check if the packet has been to this engine before
247: if IsFirstVisit(EngineIdentity, EngineList) then
248: begin
249: // 3. Append this Engine to the packet, and
250: // resend the packet on all media, but the incoming.
251: ThePacket.AddEngine(EngineIdentity); // AddEngine(@PacketArray, EngineIdentity);
252: ThePacket.Concretize(@PacketArray);
253: FMediaList.DeliverToAll(@PacketArray);
254: if Receiver = 'Alla' then
255: begin
256: SendToAllLocal(ThePacket);
257: end;
258: end else begin
259: // The packet has been to this node before. Just ignore it.
260: end;
261: EngineList.Destroy; // Dispose of the EngineList
262: end;
263: end;
264: end.
```

A.3.16 C:\projects\hermes\TransportList.pas

```
1: unit TransportList;
2:
3: interface
4: uses
5: PTransport_TLB, Contnrs;
6: //type TTransportList = Array of TMedia;
7:
8: implementation
9:
10: end.
```

A.4 PTransport.exe

A.4.1 C:\projects\HermesMedia\PTransport.dpr

```
1: program PTransport;
2:
3: uses
4: Forms,
5: HermesTransport in 'HermesTransport.pas' {Form8},
6: PTransport_TLB in 'PTransport_TLB.pas',
7: TransportImpl in 'TransportImpl.pas' {Media: CoClass},
8: MediaModule_TLB in 'MediaModule_TLB.pas',
9: HermesBinaryPacket in 'HermesBinaryPacket.pas';
10:
11: {$R *.TLB}
12:
13: {$R *.res}
14:
15: begin
16: Application.Initialize;
17: Application.CreateForm(TForm8, Form8);
18: Application.Run;
19: end.
```

A.4.2 C:\projects\HermesMedia\HermesBinaryPacket.pas

```
1: unit HermesBinaryPacket;
2:
3: interface
4:
5: uses
6: HermesTypes, contnrs;
7:
8: type
9:
10: THermesBinaryPacketQueue = Class(TObjectQueue)
11: private
12: protected
13: public
14: function Peek: TPacketArray;
15: function Pop: TPacketArray;
16: procedure Push(APacket: TPacketArray);
17: end;
18:
19:
20: implementation
21:
22:
```

```
23: function THermesBinaryPacketQueue.Peek: TPacketArray;
24: begin
25: Peek := TPacketArray(inherited Peek);
26: end;
27:
28: function THermesBinaryPacketQueue.Pop: TPacketArray;
29: begin
30: Pop := TPacketArray(inherited Pop);
31: end;
32:
33: procedure THermesBinaryPacketQueue.Push(APacket: TPacketArray);
34: begin
35: inherited Push(TObject(APacket));
36: end;
37:
38:
39:
40: end.
```

A.4.3 C:\projects\HermesMedia\HermesTransport.pas

```
1: unit HermesTransport;
2:
3: interface
4:
5: uses
6: Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
7: Dialogs, StdCtrls;
8:
9: type
10: TForm8 = class(TForm)
11: Memo1: TMemo;
12: private
13: { Private declarations }
14: public
15: { Public declarations }
16: end;
17:
18: var
19: Form8: TForm8;
20:
21: implementation
22:
23: {$R *.dfm}
24:
25: end.
```

A.4.4 C:\projects\HermesMedia\HermesTransport.dfm

```
1: object Form8: TForm8
2: Left = 984
3: Top = 806
4: Width = 222
5: Height = 176
6: Caption = 'Form8'
7: Color = clBtnFace
8: Font.Charset = DEFAULT_CHARSET
9: Font.Color = clWindowText
10: Font.Height = -11
11: Font.Name = 'MS Sans Serif'
12: Font.Style = []
```

```
13: OldCreateOrder = False
14: PixelsPerInch = 96
15: TextHeight = 13
16: object Memo1: TMemo
17: Left = 0
18: Top = 0
19: Width = 214
20: Height = 149
21: Align = alClient
22: TabOrder = 0
23: end
24: end
```

A.4.5 C:\projects\HermesMedia\MediaModule_TLB.pas

Se A.3.13 C:\projects\HermesMedia\MediaModule_TLB.pas på sidan 135.

A.4.6 C:\projects\HermesMedia\PTransport_TLB.pas

```
1: unit PTransport_TLB;
2:
3: // ************************************************************************ //
4: // WARNING
5: // -------
6: // The types declared in this file were generated from data read from a
7: // Type Library. If this type library is explicitly or indirectly (via
8: // another type library referring to this type library) re-imported, or the
9: // 'Refresh' command of the Type Library Editor activated while editing the
10: // Type Library, the contents of this file will be regenerated and all
11: // manual modifications will be lost.
12: // ************************************************************************ //
13:
14: // PASTLWTR : $Revision: 1.130 $
15: // File generated on 2002-11-13 13:13:41 from Type Library described below.
16:
17: // ************************************************************************ //
18: // Type Lib: C:\Jobb\hermes\projects\HermesMedia\PTransport.tlb (1)
19: // LIBID: {3B781756-6433-4D4B-9802-C83E95F8B165}
20: // LCID: 0
21: // Helpfile:
22: // DepndLst:
23: // (1) v1.0 MediaModule, (C:\Jobb\hermes\projects\HermesMedia\mediamodule.tlb)
24: // (2) v2.0 stdole, (C:\WINNT\System32\Stdole2.tlb)
25: // (3) v4.0 StdVCL, (C:\WINNT\System32\stdvcl40.dll)
26: // ************************************************************************ //
27: {$TYPEDADDRESS OFF} // Unit must be compiled without type-checked pointers.
28: {$WARN SYMBOL_PLATFORM OFF}
29: {$WRITEABLECONST ON}
30:
31: interface
32:
33: uses ActiveX, Classes, Graphics, MediaModule_TLB, StdVCL, Variants, Windows;
34:
35:
36:
37: // ************************************************************************//
38: // GUIDS declared in the TypeLibrary. Following prefixes are used:
39: // Type Libraries : LIBID_xxxx
40: // CoClasses : CLASS_xxxx
41: // DISPInterfaces : DIID_xxxx
```

```
42: // Non-DISP interfaces: IID_xxxx
43: // *********************************************************************//
44: const
45: // TypeLibrary Major and minor versions
46: PTransportMajorVersion = 1;
47: PTransportMinorVersion = 0;
48:
49: LIBID_PTransport: TGUID = '{3B781756-6433-4D4B-9802-C83E95F8B165}';
50:
51: CLASS_Media: TGUID = '{129D97CC-B5AC-4D55-B6F1-2B6EC10BD394}';
52: type
53:
54: // *********************************************************************//
55: // Declaration of CoClasses defined in Type Library
56: // (NOTE: Here we map each CoClass to its Default Interface)
57: // *********************************************************************//
58: Media = IMedia;
59:
60:
61: // *********************************************************************//
62: // The Class CoMedia provides a Create and CreateRemote method to
63: // create instances of the default interface IMedia exposed by
64: // the CoClass Media. The functions are intended to be used by
65: // clients wishing to automate the CoClass objects exposed by the
66: // server of this typelibrary.
67: // *********************************************************************//
68: CoMedia = class
69: class function Create: IMedia;
70: class function CreateRemote(const MachineName: string): IMedia;
71: end;
72:
73: implementation
74:
75: uses ComObj;
76:
77: class function CoMedia.Create: IMedia;
78: begin
79: Result := CreateComObject(CLASS_Media) as IMedia;
80: end;
81:
82: class function CoMedia.CreateRemote(const MachineName: string): IMedia;
83: begin
84: Result := CreateRemoteComObject(MachineName, CLASS_Media) as IMedia;
85: end;
86:
87: end.
```

A.4.7 C:\projects\HermesMedia\TransportImpl.pas

```
1: unit TransportImpl;
2:
3: {$WARN SYMBOL_PLATFORM OFF}
4:
5: interface
6:
7: uses
8: HermesVariants,
9: IniFiles, HermesTypes, HermesMessage, HermesBinaryPacket, MediaModule_TLB,
10: ComObj, ActiveX, AxCtrls, Classes, PTransport_TLB, StdVcl;
11:
12: type
```

```
13: TMedia = class(TAutoObject, IConnectionPointContainer, IMedia)
14: private
15: { Private declarations }
16: FConnectionPoints: TConnectionPoints;
17: FConnectionPoint: TConnectionPoint;
18: FEvents: IMediaEvents;
19: { note: FEvents maintains a *single* event sink. For access to more
20: than one event sink, use FConnectionPoint.SinkList, and iterate
21: through the list of sinks. }
22: MediaType: TMediaType;
23: Address: TMediaAddress;
24: Config: THashedStringList;
25: Port: WideString;
26: Init: WideSTring;
27: State: TMediaState;
28: public
29: procedure Initialize; override;
30: protected
31: { Protected declarations }
32: InQueue: THermesBinaryPacketQueue;
33: OutQueue: THermesBinaryPacketQueue;
34: property ConnectionPoints: TConnectionPoints read FConnectionPoints
35: implements IConnectionPointContainer;
36: procedure EventSinkChanged(const EventSink: IUnknown); override;
37: procedure ConnectionClose; safecall;
38: procedure ConnectionGetAddress(out Info: WideString); safecall;
39: procedure ConnectionGetStatus(out ConnectionStatus: Integer); safecall;
40: procedure ConnectionOpen(const ContactInfo: WideString); safecall;
41: procedure PacketGetMaxSize(out Size: Integer); safecall;
42: procedure PacketReceive(out ThePacket: POleVariant;
43: out QueueLength: Integer); safecall;
44: procedure PacketTransmit(const ThePacket: Variant); safecall;
45: procedure QueueGetReceiveMaxSize(out Size: Integer); safecall;
46: procedure QueueGetReceiveSize(out Size: Integer); safecall;
47: procedure QueueGetTransmitMaxSize(out Size: Integer); safecall;
48: procedure QueueGetTransmitSize(out Size: Integer); safecall;
49: procedure QueueSetReceiveMaxSize(var Size: Integer); safecall;
50: procedure QueueSetTransmitMaxSize(var Size: Integer); safecall;
51: procedure Start(const ConfigFile: WideString); safecall;
52: procedure Stop; safecall;
53: function OnLinkLost: HResult; stdcall;
54: function OnLinkRequest: HResult; stdcall;
55: function OnPacketReceived: HResult; stdcall;
56: procedure IMedia.ConnectionGetAddress = IMedia_ConnectionGetAddress;
57: procedure IMedia.ConnectionOpen = IMedia_ConnectionOpen;
58: procedure IMedia.PacketReceive = IMedia_PacketReceive;
59: procedure IMedia.PacketTransmit = IMedia_PacketTransmit;
60:
61: procedure IMedia_ConnectionGetAddress(out Info: OleVariant); safecall;
62: procedure IMedia_ConnectionOpen(ContactInfo: OleVariant); safecall;
63: procedure IMedia_PacketReceive(out ThePacket: OleVariant;
64: out QueueLength: Integer); safecall;
65: procedure IMedia_PacketTransmit(ThePacket: OleVariant); safecall;
66: end;
67:
68:
69: implementation
70:
71: uses Variants, ComServ, HermesTransport, SysUtils, Windows;
72:
73: procedure TMedia.EventSinkChanged(const EventSink: IUnknown);
```

```
74: begin
75: FEvents := EventSink as IMediaEvents;
76: end;
77:
78: procedure TMedia.Initialize;
79: begin
80: inherited Initialize;
81: FConnectionPoints := TConnectionPoints.Create(Self);
82: if AutoFactory.EventTypeInfo <> nil then
83: FConnectionPoint := FConnectionPoints.CreateConnectionPoint(
84: AutoFactory.EventIID, ckSingle, EventConnect)
85: else FConnectionPoint := nil;
86: end;
87:
88:
89: procedure TMedia.ConnectionClose;
90: begin
91: Form8.Memo1.Lines.Add('Connection closed');
92: end;
93:
94: procedure TMedia.ConnectionGetAddress(out Info: WideString);
95: begin
96:
97: end;
98:
99: procedure TMedia.ConnectionGetStatus(out ConnectionStatus: Integer);
100: begin
101:
102: end;
103:
104: procedure TMedia.ConnectionOpen(const ContactInfo: WideString);
105: begin
106: end;
107:
108: procedure TMedia.PacketGetMaxSize(out Size: Integer);
109: begin
110: Size := 1024;
111: Form8.Memo1.Lines.Add('PacketGetMaxSize' + IntToStr(Size));
112: end;
113:
114: procedure TMedia.PacketReceive(out ThePacket: POleVariant;
115: out QueueLength: Integer);
116: var
117: APacket: TPacketArray;
118: begin
119: Form8.Memo1.Lines.Add('Receive Packet');
120: if OutQueue.Count > 0 then
121: begin
122: APacket := OutQueue.Pop;
123: ArrayToVariant(APacket, ThePacket^);
124: end
125: else
126: begin
127: Assert(false);
128: APacket := nil;
129: end;
130: QueueLength := OutQueue.Count;
131: end;
132:
133: procedure TMedia.PacketTransmit(const ThePacket: Variant);
134: //var
```

```
135: //PacketArray: TPacketArray;
136: //TestArray: Pointer;
137: begin
138: //PacketArray := ThePacket;
139: //Form8.Memo1.Lines.Add('Transmitting Packet');
140: //ArrayFromVariant(PacketArray, ThePacket);
141: //OutQueue.Push(THermesPacket.Create(@PacketArray));
142: //Form8.Memo1.Lines.Add('Sending Packet Received event.');
143: //FEvents.OnPacketReceived;
144: end;
145:
146: procedure TMedia.QueueGetReceiveMaxSize(out Size: Integer);
147: begin
148: Form8.Memo1.Lines.Add('QueueGetReceiveMaxSize');
149: Size := 10;
150: end;
151:
152: procedure TMedia.QueueGetReceiveSize(out Size: Integer);
153: begin
154:
155: end;
156:
157: procedure TMedia.QueueGetTransmitMaxSize(out Size: Integer);
158: begin
159:
160: end;
161:
162: procedure TMedia.QueueGetTransmitSize(out Size: Integer);
163: begin
164:
165: end;
166:
167: procedure TMedia.QueueSetReceiveMaxSize(var Size: Integer);
168: begin
169:
170: end;
171:
172: procedure TMedia.QueueSetTransmitMaxSize(var Size: Integer);
173: begin
174:
175: end;
176:
177: procedure TMedia.Start(const ConfigFile: WideString);
178: var
179: Index: Integer;
180: begin
181:
182: InQueue := THermesBinaryPacketQueue.Create;
183: OutQueue := THermesBinaryPacketQueue.Create;
184:
185: Config := THashedStringList.Create;
186: Config.LoadFromFile(ConfigFile);
187: Index := Config.IndexOfName('MediaType');
188: if Index > -1 then begin
189: MediaType := Copy(Config.Strings[Index], 11, 255);
190: Form8.Memo1.Lines.Add('Media type = ' + MediaType);
191: end;
192: Index := Config.IndexOfName('Adress');
193: if Index > -1 then begin
194: Address := Copy(Config.Strings[Index], 8, 255);
195: Form8.Memo1.Lines.Add('Address = ' + Address);
```

```
196: end;
197: Index := Config.IndexOfName('Port');
198: if Index > -1 then begin
199: Port := Copy(Config.Strings[Index], 6, 255);
200: Form8.Memo1.Lines.Add('Port = ' + Port);
201: end;
202: Index := Config.IndexOfName('Init');
203: if Index > -1 then begin
204: Init := Copy(Config.Strings[Index], 6, 255);
205: Form8.Memo1.Lines.Add('Init = ' + Init);
206: end;
207: end;
208:
209: procedure TMedia.Stop;
210: begin
211: Config.Destroy;
212: InQueue.Destroy;
213: OutQueue.Destroy;
214: Form8.Memo1.Lines.Add('Media Stopped!');
215: end;
216:
217: function TMedia.OnLinkLost: HResult;
218: begin
219: result := HResult(0);
220: end;
221:
222: function TMedia.OnLinkRequest: HResult;
223: begin
224: result := HResult(0);
225: end;
226:
227: function TMedia.OnPacketReceived: HResult;
228: begin
229: result := HResult(0);
230: end;
231:
232: procedure TMedia.IMedia_ConnectionGetAddress(out Info: OleVariant);
233: begin
234: end;
235:
236: procedure TMedia.IMedia_ConnectionOpen(ContactInfo: OleVariant);
237: begin
238: Form8.Memo1.Lines.Add('Connection open with contact info:');
239: Form8.Memo1.Lines.Add(WideString(ContactInfo));
240: end;
241:
242: procedure TMedia.IMedia_PacketReceive(out ThePacket: OleVariant;
243: out QueueLength: Integer);
244: var
245: APacket: TPacketArray;
246: begin
247: Form8.Memo1.Lines.Add('Receive Packet');
248: if OutQueue.Count > 0 then
249: begin
250: APacket := OutQueue.Pop;
251: ArrayToVariant(APacket, ThePacket);
252: end
253: else
254: begin
255: Assert(false, 'Empty packet queue, while trying to receive');
256: APacket := nil;
```

```
257: end;
258: QueueLength := OutQueue.Count;
259: end;
260:
261: procedure TMedia.IMedia_PacketTransmit(ThePacket: OleVariant);
262: var
263: PacketArray: TPacketArray;
264: //TestArray: Pointer;
265: begin
266: //PacketArray := ThePacket;
267: Form8.Memo1.Lines.Add('Converting Packet');
268: ArrayFromVariant(PacketArray, ThePacket);
269: Form8.Memo1.Lines.Add('Queueing Packet');
270: OutQueue.Push(PacketArray);
271: Form8.Memo1.Lines.Add('Sending Packet Received event.');
272: FEvents.OnPacketReceived;
273: end;
274:
275: initialization
276: TAutoObjectFactory.Create(ComServer, TMedia, Class_Media,
277: ciSingleInstance, tmFree);
278: end.
279:
```

A.5 TestProgs.exe

A.5.1 C:\projects\tests\TestProgs.dpr

```
1: program TestProgs;
2:
3: uses
4: Forms,
5: TestPacket in 'TestPacket.pas' {Form7};
6:
7: {$R *.res}
8:
9: begin
10: Application.Initialize;
11: Application.CreateForm(TForm7, Form7);
12: Application.Run;
13: end.
```

A.5.2 C:\projects\tests\TestPacket.pas

```
1: unit TestPacket;
2:
3: interface
4:
5: uses
6: HermesPacket, HermesTypes,
7: Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
8: Dialogs, StdCtrls, ExtCtrls;
9:
10: type
11: TForm7 = class(TForm)
12: NewPacket: TButton;
13: Receiver: TLabeledEdit;
14: Sender: TLabeledEdit;
15: PacketType: TLabeledEdit;
16: Concretize: TButton;
```

```
17: ReadBack: TButton;
18: Memo1: TMemo;
19: OSender: TLabel;
20: OReceiver: TLabel;
21: OPriority: TLabel;
22: ODataSize: TLabel;
23: NoNodes: TLabel;
24: SeqNr: TLabel;
25: ListBox1: TListBox;
26: Memo2: TMemo;
27: Label1: TLabel;
28: Label2: TLabel;
29: OPacketType: TLabel;
30: NewEngine: TLabeledEdit;
31: AddEngine: TButton;
32: FindEngine: TButton;
33: EngineFound: TCheckBox;
34: procedure NewPacketClick(Sender: TObject);
35: procedure ReadBackClick(Sender: TObject);
36: procedure ConcretizeClick(Sender: TObject);
37: procedure AddEngineClick(Sender: TObject);
38: procedure FindEngineClick(Sender: TObject);
39: private
40: { Private declarations }
41: public
42: { Public declarations }
43: end;
44:
45: var
46: Form7: TForm7;
47:
48: Packet1, Packet2: THermesPacket;
49: DataArray, TheArray, DummyArray: TPacketArray;
50:
51: implementation
52:
53: {$R *.dfm}
54:
55: procedure TForm7.NewPacketClick(Sender: TObject);
56: var
57: Index: Integer;
58: begin
59: Packet1 := THermesPacket.Create(THermesPacketType(StrToInt(String(Form7.Pack-
etType.Text))), AnsiString(Form7.Receiver.Text), AnsiString(Form7.Sender.Text));
60: Packet1.SetPriority(2);
61: SetLength(DummyArray, 8);
62: For Index := 0 to 7 do
63: begin
64: DummyArray[Index] := Index;
65: end;
66: Packet1.SetData(@DummyArray);
67: Packet1.SetSequenceNumber($ABCDEF01);
68: //Packet1.AddEngine('En motor');
69: end;
70:
71: procedure TForm7.ReadBackClick(Sender: TObject);
72: var
73: Index: Integer;
74: begin
75: Packet2 := THermesPacket.Create(@TheArray);
76: Form7.OSender.Caption := 'Sender=' + Packet2.GetSender;
```

```
77: Form7.OReceiver.Caption := 'Receiver=' + Packet2.GetReceiver;
78: Form7.OPriority.Caption := 'Priority=' + IntToStr(Packet2.GetPriority);
79: Packet2.GetData(@DataArray);
80: Form7.ODataSize.Caption := 'Data Size = ' + IntToStr(Length(DataArray));
81: Form7.SeqNr.Caption := 'Seq.nr.=' + IntToStr(Packet2.GetSequenceNumber);
82: Form7.NoNodes.Caption := 'No.Nodes=' + IntToStr(Packet2.GetEngines.Count);
83: Form7.OPacketType.Caption := 'PacketType=' + IntToStr(Integer(Packet2.GetType));
84: Form7.ListBox1.Items := Packet2.GetEngines;
85: Memo2.Text := '';
86: for Index := Low(DataArray) to High(DataArray) do
87: begin
88: Memo2.Text := Memo2.Text + IntToStr(DataArray[Index]) + ',';
89: end;
90: end;
91:
92: procedure TForm7.ConcretizeClick(Sender: TObject);
93: var
94: Index: Integer;
95: begin
96: Packet1.Concretize(@TheArray);
97: Memo1.Text := '';
98: for Index := Low(TheArray) to High(TheArray) do
99: begin
100: Memo1.Text := Memo1.Text + IntToStr(TheArray[Index]) + ',';
101: end;
102: end;
103:
104: procedure TForm7.AddEngineClick(Sender: TObject);
105: begin
106: Packet1.AddEngine(WideString(NewEngine.Text));
107: end;
108:
109: procedure TForm7.FindEngineClick(Sender: TObject);
110: begin
111: EngineFound.Checked := Packet2.FindEngine(WideString(NewEngine.Text));
112: end;
113:
114: end.
```

A.5.3 C:\projects\tests\TestPacket.dfm

```
1: object Form7: TForm7
2: Left = 166
3: Top = 314
4: Width = 660
5: Height = 530
6: Caption = 'Form7'
7: Color = clBtnFace
8: Font.Charset = DEFAULT_CHARSET
9: Font.Color = clWindowText
10: Font.Height = -11
11: Font.Name = 'MS Sans Serif'
12: Font.Style = []
13: OldCreateOrder = False
14: PixelsPerInch = 96
15: TextHeight = 13
16: object OSender: TLabel
17: Left = 304
18: Top = 200
19: Width = 40
20: Height = 13
```

```
21: Caption = 'Sender='
22: end
23: object OReceiver: TLabel
24: Left = 304
25: Top = 216
26: Width = 49
27: Height = 13
28: Caption = 'Receiver='
29: end
30: object OPriority: TLabel
31: Left = 304
32: Top = 232
33: Width = 37
34: Height = 13
35: Caption = 'Priority='
36: end
37: object ODataSize: TLabel
38: Left = 304
39: Top = 264
40: Width = 49
41: Height = 13
42: Caption = 'DataSize='
43: end
44: object NoNodes: TLabel
45: Left = 304
46: Top = 280
47: Width = 51
48: Height = 13
49: Caption = 'NoNodes='
50: end
51: object SeqNr: TLabel
52: Left = 304
53: Top = 296
54: Width = 40
55: Height = 13
56: Caption = 'Seq.nr.='
57: end
58: object Label1: TLabel
59: Left = 304
60: Top = 360
61: Width = 26
62: Height = 13
63: Caption = 'Data:'
64: end
65: object Label2: TLabel
66: Left = 496
67: Top = 176
68: Width = 75
69: Height = 13
70: Caption = 'Visited Engines:'
71: end
72: object OPacketType: TLabel
73: Left = 304
74: Top = 248
75: Width = 64
76: Height = 13
77: Caption = 'PacketType='
78: end
79: object NewPacket: TButton
80: Left = 144
81: Top = 64
```

```
82: Width = 89
83: Height = 25
84: Caption = 'NewPacket'
85: TabOrder = 0
86: OnClick = NewPacketClick
87: end
88: object Receiver: TLabeledEdit
89: Left = 8
90: Top = 32
91: Width = 113
92: Height = 21
93: EditLabel.Width = 43
94: EditLabel.Height = 13
95: EditLabel.Caption = 'Receiver'
96: LabelPosition = lpAbove
97: LabelSpacing = 3
98: TabOrder = 1
99: end
100: object Sender: TLabeledEdit
101: Left = 144
102: Top = 32
103: Width = 113
104: Height = 21
105: EditLabel.Width = 34
106: EditLabel.Height = 13
107: EditLabel.Caption = 'Sender'
108: LabelPosition = lpAbove
109: LabelSpacing = 3
110: TabOrder = 2
111: end
112: object PacketType: TLabeledEdit
113: Left = 8
114: Top = 72
115: Width = 49
116: Height = 21
117: EditLabel.Width = 58
118: EditLabel.Height = 13
119: EditLabel.Caption = 'PacketType'
120: LabelPosition = lpAbove
121: LabelSpacing = 3
122: TabOrder = 3
123: end
124: object Concretize: TButton
125: Left = 8
126: Top = 160
127: Width = 89
128: Height = 25
129: Caption = 'Concretize'
130: TabOrder = 4
131: OnClick = ConcretizeClick
132: end
133: object ReadBack: TButton
134: Left = 304
135: Top = 160
136: Width = 89
137: Height = 25
138: Caption = 'ReadBack'
139: TabOrder = 5
140: OnClick = ReadBackClick
141: end
142: object Memo1: TMemo
```

```
143: Left = 8
144: Top = 192
145: Width = 193
146: Height = 241
147: Lines.Strings = (
148: 'Memo1')
149: TabOrder = 6
150: end
151: object ListBox1: TListBox
152: Left = 488
153: Top = 200
154: Width = 145
155: Height = 145
156: ItemHeight = 13
157: TabOrder = 7
158: end
159: object Memo2: TMemo
160: Left = 296
161: Top = 376
162: Width = 193
163: Height = 121
164: Lines.Strings = (
165: 'Memo2')
166: TabOrder = 8
167: end
168: object NewEngine: TLabeledEdit
169: Left = 336
170: Top = 32
171: Width = 145
172: Height = 21
173: EditLabel.Width = 55
174: EditLabel.Height = 13
175: EditLabel.Caption = 'NewEngine'
176: LabelPosition = lpAbove
177: LabelSpacing = 3
178: TabOrder = 9
179: end
180: object AddEngine: TButton
181: Left = 504
182: Top = 32
183: Width = 73
184: Height = 25
185: Caption = 'AddEngine'
186: TabOrder = 10
187: OnClick = AddEngineClick
188: end
189: object FindEngine: TButton
190: Left = 504
191: Top = 72
192: Width = 73
193: Height = 25
194: Caption = 'FindEngine'
195: TabOrder = 11
196: OnClick = FindEngineClick
197: end
198: object EngineFound: TCheckBox
199: Left = 384
200: Top = 80
201: Width = 97
202: Height = 17
203: Caption = 'EngineFound'
```

```
204: TabOrder = 12
205: end
206: end
```

A.6 PMediaTCP.exe

A.6.1 C:\projects\HermesMedia\PMediaTCP.dpr

```
1: program PMediaTCP;
2:
3: uses
4: Forms,
5: MediaTCPApp in 'MediaTCPApp.pas' {Form9},
6: PMediaTCP_TLB in 'PMediaTCP_TLB.pas',
7: MediaTCPImpl in 'MediaTCPImpl.pas' {Media: CoClass},
8: WinSockUnit in 'WinSockUnit.pas' {WinSockModule: TDataModule},
9: MediaRoutines in 'MediaRoutines.pas';
10:
11: {$R *.TLB}
12:
13: {$R *.res}
14:
15: begin
16: Application.Initialize;
17: Application.CreateForm(TForm9, Form9);
18: Application.CreateForm(TWinSockModule, WinSockModule);
19: Application.Run;
20: end.
```

A.6.2 C:\projects\HermesMedia\MediaRoutines.pas

```
1: unit MediaRoutines;
2:
3:
4: interface
5:
6: uses
7: HermesTypes;
8:
9:
10: procedure UnStuff(const InBuf: Array of Byte; InIndex: Integer; InLength: Integer; var
OutBuf: Array of Byte; var OutIndex: Integer; const EscapeByte, FlipByte: Byte);
11: procedure Stuff(const InBuf: Array of Byte; InIndex: Integer; InLength: Integer; var
OutBuf: Array of Byte; var OutIndex: Integer; const StuffByte, EscapeByte, FlipByte:
Byte);
12: procedure ArrayCopy(const InBuf: Array of Byte; InIndex: Integer; Length: Integer;
var OutBuf: Array of byte; var OutIndex: Integer);
13:
14: function PacketArrayToPrintString(const InBuf: Array of Byte; const InLength: Inte-
ger): WideString;
15:
16: implementation
17:
18: procedure ArrayCopy(const InBuf: Array of Byte; InIndex: Integer; Length: Integer;
var OutBuf: Array of byte; var OutIndex: Integer);
19: begin
20: while (Length > 0) and (InIndex <= High(InBuf)) and (OutIndex <= High(OutBuf)) do
21: begin
22: OutBuf[OutIndex] := InBuf[InIndex];
23: Inc(OutIndex, 1);
```

```
24: Inc(InIndex, 1);
25: Inc(Length, 1);
26: end;
27: end;
28:
29: function PacketArrayToPrintString(const InBuf: Array of Byte; const InLength: Integer): WideString;
30: var
31: Index: Integer;
32: ByteString: WideString;
33: ValuesString: WideString;
34: begin
35: ValuesString := '';
36: for Index := 0 to InLength - 1 do
37: begin
38: Str(InBuf[Index], ByteString);
39: ValuesString := ValuesString + ByteString + ', ';
40: end;
41: ValuesString := ValuesString + 'STOP';
42: Result := ValuesString;
43: end;
44:
45: procedure Stuff(const InBuf: Array of Byte; InIndex: Integer; InLength: Integer; var OutBuf: Array of Byte; var OutIndex: Integer; const StuffByte, EscapeByte, FlipByte: Byte);
46: begin
47: while (InLength > 0) do
48: begin
49: if ((InBuf[InIndex] = StuffByte) or (InBuf[InIndex] = EscapeByte)) then
50: begin
51: OutBuf[OutIndex] := EscapeByte;
52: Inc(OutIndex, 1);
53: OutBuf[OutIndex] := (InBuf[InIndex] xor FlipByte);
54: end
55: else
56: begin
57: OutBuf[OutIndex] := InBuf[InIndex];
58: end;
59: Inc(OutIndex, 1);
60: Inc(InIndex, 1);
61: Dec(InLength, 1);
62: end;
63: end;
64:
65: procedure UnStuff(const InBuf: Array of Byte; InIndex: Integer; InLength: Integer; var OutBuf: Array of Byte; var OutIndex: Integer; const EscapeByte, FlipByte: Byte);
66: Var
67: Flip: Byte;
68: begin
69: Flip := 0;
70: while (InLength > 0) do
71: begin
72: if (InBuf[InIndex] = EscapeByte) then
73: begin
74: Flip := FlipByte;
75: end
76: else
77: begin
78: OutBuf[OutIndex] := InBuf[InIndex] xor Flip;
79: Flip := 0;
80: Inc(OutIndex, 1);
```

```
81: end;
82: Inc(InIndex, 1);
83: Dec(InLength, 1);
84: end;
85: end;
86:
87: end.
```

A.6.3 C:\projects\HermesMedia\MediaTCPApp.pas

```
1: unit MediaTCPApp;
2:
3: interface
4:
5: uses
6: Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
7: Dialogs, StdCtrls, ScktComp, Sockets, ExtCtrls;
8:
9: type
10: TForm9 = class(TForm)
11: Memo1: TMemo;
12: private
13: { Private declarations }
14: public
15: { Public declarations }
16: end;
17:
18: var
19: Form9: TForm9;
20:
21: implementation
22:
23: {$R *.dfm}
24:
25:
26: end.
```

A.6.4 C:\projects\HermesMedia\MediaTCPApp.dfm

```
1: object Form9: TForm9
2: Left = 967
3: Top = 666
4: Width = 241
5: Height = 293
6: Caption = 'Form9'
7: Color = clBtnFace
8: Font.Charset = DEFAULT_CHARSET
9: Font.Color = clWindowText
10: Font.Height = -11
11: Font.Name = 'MS Sans Serif'
12: Font.Style = []
13: OldCreateOrder = False
14: PixelsPerInch = 96
15: TextHeight = 13
16: object Memo1: TMemo
17: Left = 0
18: Top = 0
19: Width = 233
20: Height = 266
21: Align = alClient
22: TabOrder = 0
```

```
23: end
24: end
```

A.6.5 C:\projects\HermesMedia\MediaTCPImpl.pas

```
1: unit MediaTCPImpl;
2:
3: {$WARN SYMBOL_PLATFORM OFF}
4:
5: interface
6:
7: uses
8: SyncObjs,
9: HermesBinaryPacket, HermesTypes, IniFiles, ScktComp, Windows, Messages, Sockets,
10: ComObj, ActiveX, AxCtrls, Classes, PMediaTCP_TLB, StdVcl, MediaModule_TLB;
11:
12:
13:
14: const STUFF_FLIP = $02;
15: const STUFF_ESCAPE = $E7;
16: const STUFF_TERM = $7E;
17: const BUFFERSIZE = 64*1024;
18:
19: type
20:
21: TEventThread = class(TThread)
22: private
23: protected
24: procedure Execute; override;
25: end;
26:
27: TMedia = class(TAutoObject, IConnectionPointContainer, IMedia)
28: private
29: { Private declarations }
30: FConnectionPoints: TConnectionPoints;
31: FConnectionPoint: TConnectionPoint;
32: FEvents: IMediaEvents;
33: { note: FEvents maintains a *single* event sink. For access to more
34: than one event sink, use FConnectionPoint.SinkList, and iterate
35: through the list of sinks. }
36: InQueue, OutQueue: THermesBinaryPacketQueue;
37: InputPacket: TPPacketArray;
38: OutputPacket: TPacketArray;
39: InputPacketPos: Integer;
40: OutputPacketPos: Integer;
41: InputBuf: Array[0..BUFFERSIZE] of Byte;
42: OutputBuf: TPPacketArray;
43: InputBufPos: Integer;
44: OutputBufPos: Integer;
45: MediaType: TMediaType;
46: Address: TMediaAddress;
47: ConnectAddress: TMediaAddress;
48: Config: THashedStringList;
49: PortString: WideString;
50: TcpPort: Integer;
51: Init: WideSTring;
52: State: TMediaState;
53: EventThread: TEventThread;
54: //ClientSocket: TClientSocket;
55:
56: public
```

```
57: procedure Initialize; override;
58:
59: // Client Events BEGIN
60: procedure ClientConnect(Sender: TObject;
61: Socket: TCustomWinSocket);
62: procedure ClientConnecting(Sender: TObject;
63: Socket: TCustomWinSocket);
64: procedure ClientDisconnect(Sender: TObject;
65: Socket: TCustomWinSocket);
66: procedure ClientError(Sender: TObject; Socket: TCustomWinSocket;
67: ErrorEvent: TErrorEvent; var ErrorCode: Integer);
68: procedure ClientLookup(Sender: TObject;
69: Socket: TCustomWinSocket);
70: procedure ClientRead(Sender: TObject; Socket: TCustomWinSocket);
71: procedure ClientWrite(Sender: TObject; Socket: TCustomWinSocket);
72: // Client events END
73:
74: // Server events BEGIN
75: procedure ServerSocketAccept(Sender: TObject;
76: Socket: TCustomWinSocket);
77: procedure ServerSocketClientConnect(Sender: TObject;
78: Socket: TCustomWinSocket);
79: procedure ServerSocketClientDisconnect(Sender: TObject;
80: Socket: TCustomWinSocket);
81: procedure ServerSocketClientError(Sender: TObject;
82: Socket: TCustomWinSocket; ErrorEvent: TErrorEvent;
83: var ErrorCode: Integer);
84: procedure ServerSocketClientRead(Sender: TObject;
85: Socket: TCustomWinSocket);
86: procedure ServerSocketClientWrite(Sender: TObject;
87: Socket: TCustomWinSocket);
88: procedure ServerSocketGetSocket(Sender: TObject; Socket: Integer;
89: var ClientSocket: TServerClientWinSocket);
90: procedure ServerSocketGetThread(Sender: TObject;
91: ClientSocket: TServerClientWinSocket;
92: var SocketThread: TServerClientThread);
93: procedure ServerSocketListen(Sender: TObject;
94: Socket: TCustomWinSocket);
95: procedure ServerSocketThreadEnd(Sender: TObject;
96: Thread: TServerClientThread);
97: procedure ServerSocketThreadStart(Sender: TObject;
98: Thread: TServerClientThread);
99: // Server events END
100:
101: // IdleHandler
102: procedure IdleHandler(Sender: TObject; var Done: Boolean);
103:
104: procedure WakePacketIn;
105: procedure WakePacketOut;
106:
107: protected
108: { Protected declarations }
109: property ConnectionPoints: TConnectionPoints read FConnectionPoints
110: implements IConnectionPointContainer;
111: procedure EventSinkChanged(const EventSink: IUnknown); override;
112: function OnLinkLost: HResult; stdcall;
113: function OnLinkRequest: HResult; stdcall;
114: function OnPacketReceived: HResult; stdcall;
115: procedure ConnectionClose; safecall;
116: procedure ConnectionGetAddress(out Info: OleVariant); safecall;
117: procedure ConnectionGetStatus(out ConnectionStatus: Integer); safecall;
```

```
118: procedure ConnectionOpen(ContactInfo: OleVariant); safecall;
119: procedure PacketGetMaxSize(out Size: Integer); safecall;
120: procedure PacketReceive(out ThePacket: OleVariant;
121: out QueueLength: Integer); safecall;
122: procedure PacketTransmit(ThePacket: OleVariant); safecall;
123: procedure QueueGetReceiveMaxSize(out Size: Integer); safecall;
124: procedure QueueGetReceiveSize(out Size: Integer); safecall;
125: procedure QueueGetTransmitMaxSize(out Size: Integer); safecall;
126: procedure QueueGetTransmitSize(out Size: Integer); safecall;
127: procedure QueueSetReceiveMaxSize(var Size: Integer); safecall;
128: procedure QueueSetTransmitMaxSize(var Size: Integer); safecall;
129: procedure Start(const ConfigFile: WideString); safecall;
130: procedure Stop; safecall;
131:
132: procedure PacketTransmitInternal;
133: function Transmit(Socket: TCustomWinSocket): Integer;
134: function Receive(Sender: TObject; Socket: TCustomWinSocket): Integer;
135: procedure ReloadOutput;
136: end;
137:
138: var
139: TheMedia: TMedia;
140: PacketFlagsMutex: TCriticalSection;
141: NewPacketIn: Boolean;
142: NewPacketOut: Boolean;
143:
144:
145: implementation
146:
147: uses
148: Forms,
149: HermesVariants, MediaRoutines, WinSockUnit, SysUtils, MediaTCPApp, ComServ;
150:
151:
152: procedure TEventThread.Execute;
153: var
154: LocalInFlag: Boolean;
155: LocalOutFlag: Boolean;
156: begin
157: While True do
158: begin
159: PacketFlagsMutex.Acquire;
160: LocalInFlag := NewPacketIn;
161: LocalOutFlag := NewPacketOut;
162: NewPacketIn := False;
163: NewPacketOut := False;
164: PacketFlagsMutex.Release;
165:
166: {
167: if LocalInFlag then
168: begin
169: TheMedia.WakePacketIn;
170: end;
171: }
172:
173: if LocalOutFlag then
174: begin
175: TheMedia.WakePacketOut;
176: end;
177: self.Suspend;
178: end;
```

```
179: end;
180:
181: procedure TMedia.WakePacketIn;
182: begin
183: PacketTransmitInternal;
184: end;
185:
186: procedure TMedia.WakePacketOut;
187: begin
188: FEvents.OnPacketReceived;
189: end;
190:
191: procedure TMedia.IdleHandler(Sender: TObject; var Done: Boolean);
192: var
193:
194: {$IFDEF DEBUG}
195: TempString: WideSTring;
196: {$ENDIF}
197:
198: InLength: Integer;
199: OutLength: Integer;
200: begin
201:
202: InLength := InQueue.Count;
203: OutLength := OutQueue.Count;
204:
205: PacketFlagsMutex.Acquire;
206:
207: if NewPacketIn then
208: begin
209:
210: {$IFDEF DEBUG}
211: Str(InLength, TempString);
212: Form9.Memo1.Lines.Add('New InputPacket, InQueue length is ' + TempString + '
packets.');
213: {$ENDIF}
214:
215: end;
216:
217: if NewPacketOut then
218: begin
219:
220: {$IFDEF DEBUG}
221: Str(OutLength, TempString);
222: Form9.Memo1.Lines.Add('New OutputPacket, OutQueue length is ' + TempString +
' packets.');
223: {$ENDIF}
224:
225: end;
226:
227: PacketFlagsMutex.Release;
228:
229: EventThread.Resume;
230:
231: end;
232:
233:
234: procedure TMedia.EventSinkChanged(const EventSink: IUnknown);
235: begin
236: FEvents := EventSink as IMediaEvents;
237: end;
```

```
238:
239: procedure TMedia.Initialize;
240: begin
241: inherited Initialize;
242: FConnectionPoints := TConnectionPoints.Create(Self);
243: if AutoFactory.EventTypeInfo <> nil then
244: FConnectionPoint := FConnectionPoints.CreateConnectionPoint(
245: AutoFactory.EventIID, ckSingle, EventConnect)
246: else FConnectionPoint := nil;
247: TheMedia := self;
248: State := SUnconfigured;
249: PacketFlagsMutex := TCriticalSection.Create;
250: PacketFlagsMutex.Acquire;
251: NewPacketIn := False;
252: NewPacketOut := False;
253: PacketFlagsMutex.Release;
254: EventThread := TEventThread.Create(True);
255: end;
256:
257:
258: function TMedia.OnLinkLost: HResult;
259: begin
260: result := HResult(0);
261: end;
262:
263: function TMedia.OnLinkRequest: HResult;
264: begin
265: result := HResult(0);
266: end;
267:
268: function TMedia.OnPacketReceived: HResult;
269: begin
270: result := HResult(0);
271: end;
272:
273: procedure TMedia.ConnectionClose;
274: begin
275: {$IFDEF DEBUG}
276: Form9.Memo1.Lines.Add('Closing connection.');
277: {$ENDIF}
278: ConnectAddress := '';
279: State := SDisconnecting;
280: Case State of
281: SRunningClient:
282: begin
283: WinSockModule.ClientSocket.Close;
284: //State := SDisconnected;
285: end;
286: else
287: begin
288: WinSockModule.ServerSocket.Close;
289: //State := SDisconnected;
290: end;
291: end;
292: end;
293:
294: procedure TMedia.ConnectionGetAddress(out Info: OleVariant);
295: var
296: PortString: WideString;
297: begin
298: {$IFDEF DEBUG}
```

```
299: Form9.Memo1.Lines.Add('Connection get address.');
300: {$ENDIF}
301: Case State of
302: SRunningClient:
303: begin
304: //Str(Form9.TcpClient.RePort, PortString);
305: PortString := WinSockModule.ClientSocket.Address;
306: Info := WideString(WinSockModule.ClientSocket.Address) + '=' + PortString;
307: end;
308: else
309: begin
310: //Str(Form9.ServerSocket1.Port, PortString);
311: //Info := Form9.ServerSocket1.Socket.RemoteHost + '=' + PortString;
312: end;
313: end;
314: // Info := ConnectAddress;
315: end;
316:
317: procedure TMedia.ConnectionGetStatus(out ConnectionStatus: Integer);
318: begin
319: {$IFDEF DEBUG}
320: Form9.Memo1.Lines.Add('Connection get status.');
321: {$ENDIF}
322: ConnectionStatus := Ord(State);
323: end;
324:
325: procedure TMedia.ConnectionOpen(ContactInfo: OleVariant);
326: begin
327: {$IFDEF DEBUG}
328: Form9.Memo1.Lines.Add('Opening connection:' + String(ContactInfo));
329: {$ENDIF}
330: Case State of
331: SDisconnected:
332: begin
333: State := SConnecting;
334: ConnectAddress := ContactInfo;
335:
336: //Form9.Memo1.Lines.Add('Closing server');
337: //Form9.ServerSocket1.Active := false;
338: {$IFDEF DEBUG}
339: Form9.Memo1.Lines.Add('Connect to: ' + ConnectAddress);
340: {$ENDIF}
341: WinSockModule.ClientSocket.Address := ConnectAddress;
342: // Form9.ClientSocket.Open;
343: WinSockModule.ConnectTimer.Enabled := true;
344: {$IFDEF DEBUG}
345: Form9.Memo1.Lines.Add('Setting to client mode');
346: {$ENDIF}
347: // State := SRunningClient;
348: end;
349: end;
350: end;
351:
352: procedure TMedia.PacketGetMaxSize(out Size: Integer);
353: begin
354: {$IFDEF DEBUG}
355: Form9.Memo1.Lines.Add('Packet get max size.');
356: {$ENDIF}
357: end;
358:
359: procedure TMedia.PacketReceive(out ThePacket: OleVariant;
```

```
360: out QueueLength: Integer);
361: var
362: PacketToSend: TPacketArray;
363: begin
364:
365: {$IFDEF DEBUG}
366: Form9.Memo1.Lines.Add('Packet receive.');
367: {$ENDIF}
368:
369:
370: if OutQueue.AtLeast(1) then
371: begin
372: PacketToSend := OutQueue.Pop;
373: {$IFDEF DEBUG}
374: Form9.Memo1.Lines.Add('Converting to variant.');
375: {$ENDIF}
376: ArrayToVariant(PacketToSend, ThePacket);
377: QueueLength := OutQueue.Count;
378: end
379: else
380: begin
381: // No packet to return
382: //Assert(False, 'No packet in outqueue');
383: QueueLength := 0;
384: end;
385: {$IFDEF DEBUG}
386: Form9.Memo1.Lines.Add('Exiting packet receive.');
387: {$ENDIF}
388: end;
389:
390: procedure TMedia.PacketTransmit(ThePacket: OleVariant);
391: var
392: //OutPacket: Array of Byte;
393: //Index: Integer;
394: TransmitPacket: TPacketArray;
395: //OutIndex,
396: {$IFDEF DEBUG}
397: TempString: WideString;
398: {$ENDIF}
399: begin
400: SetLength(TransmitPacket, Length(ThePacket));
401: //TransmitPacket := ThePacket;
402: //for Index := 0 to Length(ThePacket) - 1 do
403: //begin
404: // TransmitPacket[Index] := ThePacket[Index];
405: //end;
406: ArrayFromVariant(TransmitPacket, ThePacket);
407:
408: {$IFDEF DEBUG}
409: Form9.Memo1.Lines.Add('Packet transmit.');
410: Str(Length(ThePacket), TempString);
411: Form9.Memo1.Lines.Add('Packet size:' + TempString);
412: Str(Length(TransmitPacket), TempString);
413: Form9.Memo1.Lines.Add('Bytestuffing packet. TransmitLength: ' + TempString);
414: {$ENDIF}
415:
416: New(OutputBuf);
417: SetLength(OutputBuf^, Length(TransmitPacket) * 2 + 2);
418: OutputBuf^[0] := STUFF_TERM;
419: OutputBufPos := 1;
```

```
420: Stuff(TransmitPacket, 0, Length(TransmitPacket), OutputBuf^, OutputBufPos,
STUFF_TERM, STUFF_ESCAPE, STUFF_FLIP);
421: OutputBuf^[OutputBufPos] := STUFF_TERM;
422: Inc(OutputBufPos, 1);
423:
424: {$IFDEF DEBUG}
425: Str(OutputBufPos, TempString);
426: Form9.Memo1.Lines.Add('Bytestuffing done, new length: ' + TempString);
427: {$ENDIF}
428:
429: SetLength(OutputBuf^, OutputBufPos);
430:
431: {$IFDEF DEBUG}
432: Str(Length(OutputBuf^), TempString);
433: Form9.Memo1.Lines.Add('Queueing packet to send with ' + TempString + ' bytes as
follows.');
434: Form9.Memo1.Lines.Add(PacketArrayToPrintString(OutputBuf^, Length(Output-
Buf^)));
435: {$ENDIF}
436:
437: InQueue.Push(OutputBuf^);
438:
439: {$IFDEF DEBUG}
440: Form9.Memo1.Lines.Add('Packet enqueued.');
441: {$ENDIF}
442:
443: OutputBuf := nil;
444: {
445: PacketFlagsMutex.Acquire;
446: NewPacketIn := True;
447: PacketFlagsMutex.Release;
448: EventThread.Resume;
449: }
450: PacketTransmitInternal;
451: end;
452:
453: procedure TMedia.PacketTransmitInternal;
454: var
455: SentBytes: Integer;
456:
457: {$IFDEF DEBUG}
458: TempString: WideString;
459: {$ENDIF}
460:
461: begin
462: Case State of
463: SRunningClient:
464: begin
465:
466: {$IFDEF DEBUG}
467: Form9.Memo1.Lines.Add('Sending over client.');
468: {$ENDIF}
469:
470: Assert(WinSockModule.ClientSocket.Socket.Connected, 'Client not connected');
471: SentBytes := Transmit(WinSockModule.ClientSocket.Socket);
472:
473: {$IFDEF DEBUG}
474: Str(SentBytes, TempString);
475: Form9.Memo1.Lines.Add('Number of bytes sent over client:' + TempString);
476: {$ENDIF}
477:
```

```
478: end;
479: SRunningServer:
480: begin
481:
482: {$IFDEF DEBUG}
483: Form9.Memo1.Lines.Add('Sending over server.');
484: {$ENDIF}
485:
486: Assert(WinSockModule.ServerSocket.Socket.Connections[0].Connected, 'Server not
connected');
487: SentBytes := Transmit(WinSockModule.ServerSocket.Socket.Connections[0]);
488:
489: {$IFDEF DEBUG}
490: Str(SentBytes, TempString);
491: Form9.Memo1.Lines.Add('Number of bytes sent over server:' + TempString);
492: {$ENDIF}
493:
494: end;
495: else
496: begin
497:
498: {$IFDEF DEBUG}
499: Form9.Memo1.Lines.Add('No connection running!');
500: {$ENDIF}
501:
502: end;
503: end;
504: end;
505:
506: procedure TMedia.QueueGetReceiveMaxSize(out Size: Integer);
507: begin
508:
509: end;
510:
511: procedure TMedia.QueueGetReceiveSize(out Size: Integer);
512: begin
513:
514: end;
515:
516: procedure TMedia.QueueGetTransmitMaxSize(out Size: Integer);
517: begin
518:
519: end;
520:
521: procedure TMedia.QueueGetTransmitSize(out Size: Integer);
522: begin
523:
524: end;
525:
526: procedure TMedia.QueueSetReceiveMaxSize(var Size: Integer);
527: begin
528:
529: end;
530:
531: procedure TMedia.QueueSetTransmitMaxSize(var Size: Integer);
532: begin
533:
534: end;
535:
536: procedure TMedia.Start(const ConfigFile: WideString);
537: var
```

```
538: Index: Integer;
539: ValCode: Integer;
540: begin
541:
542: {$IFDEF DEBUG}
543: Form9.Memo1.Lines.Add('Starting Media');
544: {$ENDIF}
545:
546: WinSockModule.MediaModule := Self;
547:
548: InQueue := THermesBinaryPacketQueue.Create;
549: OutQueue := THermesBinaryPacketQueue.Create;
550: Application.OnIdle := IdleHandler;
551: New(InputPacket);
552: InputPacketPos := 0;
553: OutputPacketPos := -1;
554: InputBufPos := 0;
555: OutputBufPos := 0;
556:
557:
558: Config := THashedStringList.Create;
559: Config.LoadFromFile(ConfigFile);
560: Index := Config.IndexOfName('MediaType');
561: if Index > -1 then begin
562: MediaType := Copy(Config.Strings[Index], 11, 255);
563:
564: {$IFDEF DEBUG}
565: Form9.Memo1.Lines.Add('Media type = ' + MediaType);
566: {$ENDIF}
567:
568: end;
569: Index := Config.IndexOfName('Adress');
570: if Index > -1 then begin
571: Address := Copy(Config.Strings[Index], 8, 255);
572:
573: {$IFDEF DEBUG}
574: Form9.Memo1.Lines.Add('Address = ' + Address);
575: {$ENDIF}
576:
577: end;
578: Index := Config.IndexOfName('Port');
579: if Index > -1 then begin
580: PortString := Copy(Config.Strings[Index], 6, 255);
581:
582: {$IFDEF DEBUG}
583: Form9.Memo1.Lines.Add('Port = ' + PortString);
584: {$ENDIF}
585:
586: val(PortString, TcpPort, ValCode);
587: Assert(ValCode = 0);
588: WinSockModule.ClientSocket.Port := TcpPort;
589: WinSockModule.ServerSocket.Port := TcpPort;
590: // Form9.ServerSocket.Active := true;
591: WinSockModule.ServerTimer.Enabled := True;
592:
593: {$IFDEF DEBUG}
594: Form9.Memo1.Lines.Add('TCP Port = ' + PortString);
595: Form9.Memo1.Lines.Add('Setting to server mode');
596: {$ENDIF}
597:
598: end;
```

```
599: Index := Config.IndexOfName('Init');
600: if Index > -1 then begin
601: Init := Copy(Config.Strings[Index], 6, 255);
602: Form9.Memo1.Lines.Add('Init = ' + Init);
603: end;
604:
605: State := SDisconnected;
606:
607: Index := Config.IndexOfName('AutoConnect');
608: if Index > -1 then begin
609:
610: {$IFDEF DEBUG}
611: Form9.Memo1.Lines.Add('Closing server');
612: {$ENDIF}
613:
614: WinSockModule.ServerSocket.Active := false;
615: ConnectAddress := Copy(Config.Strings[Index], 13, 255);
616:
617: {$IFDEF DEBUG}
618: Form9.Memo1.Lines.Add('AutoConnect = ' + ConnectAddress);
619: {$ENDIF}
620:
621: WinSockModule.ClientSocket.Address := ConnectAddress;
622: WinSockModule.ConnectTimer.Enabled := true;
623:
624: {$IFDEF DEBUG}
625: Form9.Memo1.Lines.Add('Setting to client mode');
626: {$ENDIF}
627:
628: //State := SRunningClient;
629: end;
630:
631: end;
632:
633: procedure TMedia.Stop;
634: begin
635:
636: {$IFDEF DEBUG}
637: Form9.Memo1.Lines.Add('Stopping Media');
638: {$ENDIF}
639:
640: InQueue.Destroy;
641: OutQueue.Destroy;
642:
643: {$IFDEF DEBUG}
644: Form9.Memo1.Lines.Add('State change: State = SDisconnected');
645: {$ENDIF}
646:
647: State := SDisconnected;
648: end;
649:
650: // Client Events start
651: procedure TMedia.ClientConnect(Sender: TObject; Socket: TCustomWinSocket);
652: begin
653:
654: {$IFDEF DEBUG}
655: Form9.Memo1.Lines.Add('State change: State = SRunningClient');
656: {$ENDIF}
657:
658: State := SRunningClient;
659: end;
```

```
660:
661: procedure TMedia.ClientConnecting(Sender: TObject; Socket: TCustomWinSocket);
662: begin
663:
664: {$IFDEF DEBUG}
665: Form9.Memo1.Lines.Add('State change: State = SConnecting');
666: {$ENDIF}
667:
668: State := SConnecting;
669: end;
670:
671: procedure TMedia.ClientDisconnect(Sender: TObject; Socket: TCustomWinSocket);
672: begin
673: State := SDisconnected;
674:
675: {$IFDEF DEBUG}
676: Form9.Memo1.Lines.Add('State change: State = SDisconnected');
677: {$ENDIF}
678:
679: FEvents.OnLinkLost;
680: end;
681:
682: procedure TMedia.ClientError(Sender: TObject; Socket: TCustomWinSocket; ErrorE-
vent: TErrorEvent; var ErrorCode: Integer);
683: begin
684: end;
685:
686: procedure TMedia.ClientLookup(Sender: TObject; Socket: TCustomWinSocket);
687: begin
688: end;
689:
690: procedure TMedia.ClientRead(Sender: TObject; Socket: TCustomWinSocket);
691: var
692: InSize: Integer;
693:
694: {$IFDEF DEBUG}
695: TempString: WideString;
696: {$ENDIF}
697:
698: begin
699: InSize := Receive(Sender, Socket);
700:
701: {$IFDEF DEBUG}
702: Str(InSize, TempString);
703: Form9.Memo1.Lines.Add('Client processed ' + TempString + ' bytes.');
704: {$ENDIF}
705: end;
706:
707: function TMedia.Receive(Sender: TObject; Socket: TCustomWinSocket): Integer;
708: var
709: Index, TempIndex, InSize: Integer;
710: //PInput: Pointer;
711:
712: {$IFDEF DEBUG}
713: TempString: WideString;
714: {$ENDIF}
715:
716: begin
717: InSize := Socket.ReceiveLength;
718: result := InSize;
719:
```

```
720: {$IFDEF DEBUG}
721: Str(InSize, TempString);
722: Form9.Memo1.Lines.Add('Input packet size = ' + TempString);
723: {$ENDIF}
724:
725: //SetLength(InputBuf, InSize);
726:
727: {$IFDEF DEBUG}
728: Str(Length(InputBuf), TempString);
729: Form9.Memo1.Lines.Add('InputBuf is ' + TempString + ' bytes long before receive.');
730: {$ENDIF}
731:
732: //PInput := InputBuf^;
733: InSize := Socket.ReceiveBuf(InputBuf, BUFFERSIZE);
734:
735: {$IFDEF DEBUG}
736: Str(Length(InputBuf), TempString);
737: Form9.Memo1.Lines.Add('InputBuf is ' + TempString + ' bytes long after receive.');
738: {$ENDIF}
739:
740: if (InSize > 0) then
741: begin
742:
743: {$IFDEF DEBUG}
744: Str(InSize, TempString);
745: Form9.Memo1.Lines.Add('Received ' + TempString + ' bytes as follows:');
746: Form9.Memo1.Lines.Add(PacketArrayToPrintString(InputBuf, InSize));
747: {$ENDIF}
748:
749: if Assigned(InputPacket) then
750: begin
751: if Assigned(InputPacket^) then
752: begin
753: SetLength(InputPacket^, Length(InputPacket^) + InSize);
754: end
755: else
756: begin
757: SetLength(InputPacket^, InSize);
758: end;
759: end
760: else
761: begin
762: New(InputPacket);
763: SetLength(InputPacket^, InSize);
764: end;
765:
766: {$IFDEF DEBUG}
767: Str(Length(InputPacket^), TempString);
768: Form9.Memo1.Lines.Add('Reserved ' + TempString + ' bytes for input packet.');
769: {$ENDIF}
770:
771:
772: For Index := Low(InputBuf) to Low(InputBuf) + InSize - 1 do
773: begin
774: {$IFDEF DEBUG}
775: {$IFDEF NADA}
776: Str(Index, TempString);
777: Form9.Memo1.Lines.Add('Checking byte ' + TempString + '.');
778: Str(InputBuf[Index], TempString);
779: Form9.Memo1.Lines.Add('Byte value = ' + TempString);
780: {$ENDIF}
```

```
781: {$ENDIF}
782: if InputBuf[Index] = STUFF_TERM then
783: begin
784: if InputPacketPos = 0 then
785: begin
786: // A flag byte, but nothing to send => Start flag, or erroneous transmission
787:
788: {$IFDEF DEBUG}
789: Form9.Memo1.Lines.Add('Flag byte detected, empty packet.');
790: {$ENDIF}
791:
792: end
793: else
794: begin
795: // Here we have some data. Create a packet, and insert it in the input queue
796: TempIndex := 0;
797:
798: {$IFDEF DEBUG}
799: Form9.Memo1.Lines.Add('Flag byte detected, unstuffing.');
800: {$ENDIF}
801:
802: UnStuff(InputPacket^, 0, InputPacketPos, InputPacket^, TempIndex,
STUFF_ESCAPE, STUFF_FLIP);
803:
804: {$IFDEF DEBUG}
805: Str(TempIndex, TempString);
806: Form9.Memo1.Lines.Add('Unstuffing done, new size: ' + TempString);
807: {$ENDIF}
808:
809: SetLength(InputPacket^, TempIndex);
810:
811: {$IFDEF DEBUG}
812: Form9.Memo1.Lines.Add('Enqueueing packet');
813: {$ENDIF}
814:
815: OutQueue.Push(InputPacket^);
816: InputPacket := nil;
817: New(InputPacket);
818: InputPacketPos := 0;
819:
820: PacketFlagsMutex.Acquire;
821: NewPacketOut := True;
822: PacketFlagsMutex.Release;
823:
824: //FEvents.OnPacketReceived;
825: end;
826: end
827: else
828: begin
829: {$IFDEF DEBUG}
830: Str(InputPacketPos, TempString);
831: Form9.Memo1.Lines.Add('Received byte at InputPacketPos = ' + TempString);
832: {$ENDIF}
833: InputPacket^[InputPacketPos] := InputBuf[Index];
834: Inc(InputPacketPos, 1);
835: end;
836: end;
837: end
838: else
839: begin
840: // Received no characters
```

```
841: end;
842: EventThread.Resume;
843: end;
844:
845: procedure TMedia.ClientWrite(Sender: TObject; Socket: TCustomWinSocket);
846: var
847: SentBytes: Integer;
848:
849: {$IFDEF DEBUG}
850: TempString: WideString;
851: {$ENDIF}
852:
853: begin
854: SentBytes := Transmit(Socket);
855:
856: {$IFDEF DEBUG}
857: Str(SentBytes, TempString);
858: Form9.Memo1.Lines.Add('Bytes sent over Client connection:' + TempString)
859: {$ENDIF}
860:
861: end;
862:
863: procedure TMedia.ReloadOutput;
864:
865: {$IFDEF DEBUG}
866: var
867: TempString: WideString;
868: {$ENDIF}
869:
870: begin
871: if InQueue.AtLeast(1) and (OutputPacketPos < 0) then
872: begin
873: OutputPacket := InQueue.Pop;
874:
875: {$IFDEF DEBUG}
876: Str(Length(OutputPacket), TempString);
877: Form9.Memo1.Lines.Add('Popped packet to send with ' + TempString + ' bytes as
follows:');
878: Form9.Memo1.Lines.Add(PacketArrayToPrintString(OutputPacket, Length(Output-
Packet)));
879: {$ENDIF}
880:
881: OutputPacketPos := 0;
882: end;
883: end;
884:
885: function TMedia.Transmit(Socket: TCustomWinSocket): Integer;
886: var
887: SentBytes: Integer;
888: OutIndex: Integer;
889: SendBuf: Array[1..BUFFERSIZE] of Byte;
890:
891: {$IFDEF DEBUG}
892: TempInt: Integer;
893: TempString: WideString;
894: {$ENDIF}
895:
896: begin
897: // Check for bytes left in current packet
898: ReloadOutput;
899: Result := 0;
```

```
900: // SentBytes := 1;
901: if OutputPacketPos >= 0 then
902: begin
903: // Send as many bytes as possible
904:
905: {$IFDEF DEBUG}
906: TempInt := Length(OutputPacket);
907: Str(TempInt, TempString);
908: Form9.Memo1.Lines.Add('Trying to send ' + TempString + ' bytes.');
909: {$ENDIF}
910:
911: OutIndex := 0;
912: ArrayCopy(OutputPacket, OutputPacketPos, Length(OutputPacket), SendBuf, Out-
Index);
913: //SendBuf := Copy(OutputPacket, OutputPacketPos, Length(OutputPacket));
914: SentBytes := Socket.SendBuf(SendBuf, Length(OutputPacket) - OutputPacketPos);
915:
916: {$IFDEF DEBUG}
917: Str(SentBytes, TempString);
918: Form9.Memo1.Lines.Add('Succeeded to send ' + TempString + ' bytes.');
919: {$ENDIF}
920:
921: if SentBytes > 0 then
922: begin
923: Inc(OutputPacketPos, SentBytes);
924: Inc(Result, SentBytes);
925: if OutputPacketPos >= Length(OutputPacket) then
926: begin
927: // Whole packet sent, reload, and send again
928:
929: {$IFDEF DEBUG}
930: Form9.Memo1.Lines.Add('Whole packet sent.');
931: {$ENDIF}
932:
933: OutputPacketPos := -1;
934: //ReloadOutput;
935: end;
936: end;
937: end;
938: end;
939:
940:
941: // Client events END
942:
943: //Server events BEGIN
944: procedure TMedia.ServerSocketAccept(Sender: TObject;
945: Socket: TCustomWinSocket);
946: begin
947: {$IFDEF DEBUG}
948: Form9.Memo1.Lines.Add('State change to SRunningServer.');
949: {$ENDIF}
950: State := SRunningServer;
951: end;
952:
953: procedure TMedia.ServerSocketClientConnect(Sender: TObject;
954: Socket: TCustomWinSocket);
955: begin
956: {$IFDEF DEBUG}
957: Form9.Memo1.Lines.Add('ClientConnect.');
958: {$ENDIF}
959: end;
```

```
960:
961: procedure TMedia.ServerSocketClientDisconnect(Sender: TObject;
962: Socket: TCustomWinSocket);
963: begin
964: {$IFDEF DEBUG}
965: Form9.Memo1.Lines.Add('State change to SDisconnected.');
966: {$ENDIF}
967: State := SDisconnected;
968: end;
969:
970: procedure TMedia.ServerSocketClientError(Sender: TObject;
971: Socket: TCustomWinSocket; ErrorEvent: TErrorEvent;
972: var ErrorCode: Integer);
973: begin
974: {$IFDEF DEBUG}
975: Form9.Memo1.Lines.Add('ClientError.');
976: {$ENDIF}
977: end;
978:
979: procedure TMedia.ServerSocketClientRead(Sender: TObject;
980: Socket: TCustomWinSocket);
981: var
982: InSize: Integer;
983:
984: {$IFDEF DEBUG}
985: TempString: WideString;
986: {$ENDIF}
987:
988: begin
989: InSize := Receive(Sender, Socket);
990:
991: {$IFDEF DEBUG}
992: Str(InSize, TempString);
993: Form9.Memo1.Lines.Add('Server processed ' + TempString + ' bytes.');
994: {$ENDIF}
995:
996: end;
997:
998: procedure TMedia.ServerSocketClientWrite(Sender: TObject;
999: Socket: TCustomWinSocket);
1000: var
1001: SentBytes: Integer;
1002:
1003: {$IFDEF DEBUG}
1004: TempString: WideString;
1005: {$ENDIF}
1006:
1007: begin
1008: SentBytes := Transmit(Socket);
1009:
1010: {$IFDEF DEBUG}
1011: Str(SentBytes, TempString);
1012: Form9.Memo1.Lines.Add('Bytes sent over Client connection:' + TempString)
1013: {$ENDIF}
1014:
1015: end;
1016:
1017: procedure TMedia.ServerSocketGetSocket(Sender: TObject;
1018: Socket: Integer; var ClientSocket: TServerClientWinSocket);
1019: begin
1020:
```

```
1021: end;
1022:
1023: procedure TMedia.ServerSocketGetThread(Sender: TObject;
1024: ClientSocket: TServerClientWinSocket;
1025: var SocketThread: TServerClientThread);
1026: begin
1027:
1028: end;
1029:
1030: procedure TMedia.ServerSocketListen(Sender: TObject;
1031: Socket: TCustomWinSocket);
1032: begin
1033: {$IFDEF DEBUG}
1034: Form9.Memo1.Lines.Add('Server Listening.');
1035: {$ENDIF}
1036: end;
1037:
1038: procedure TMedia.ServerSocketThreadEnd(Sender: TObject;
1039: Thread: TServerClientThread);
1040: begin
1041:
1042: end;
1043:
1044: procedure TMedia.ServerSocketThreadStart(Sender: TObject;
1045: Thread: TServerClientThread);
1046: begin
1047:
1048: end;
1049:
1050: // Server events END
1051:
1052:
1053: initialization
1054: TAutoObjectFactory.Create(ComServer, TMedia, Class_Media,
1055: ciSingleInstance, tmFree);
1056: end.
```

A.6.6 C:\projects\HermesMedia\PMediaTCP_TLB.pas

```
1: unit PMediaTCP_TLB;
2:
3: // ************************************************************************ //
4: // WARNING
5: // -------
6: // The types declared in this file were generated from data read from a
7: // Type Library. If this type library is explicitly or indirectly (via
8: // another type library referring to this type library) re-imported, or the
9: // 'Refresh' command of the Type Library Editor activated while editing the
10: // Type Library, the contents of this file will be regenerated and all
11: // manual modifications will be lost.
12: // ************************************************************************ //
13:
14: // PASTLWTR : $Revision: 1.130 $
15: // File generated on 2002-04-24 16:06:09 from Type Library described below.
16:
17: // ************************************************************************ //
18: // Type Lib: C:\projects\HermesMedia\PMediaTCP.tlb (1)
19: // LIBID: {4BF948C1-9EE5-4DBE-9CAA-79ECDAD3C4EB}
20: // LCID: 0
21: // Helpfile:
22: // DepndLst:
```

```
23: // (1) v1.0 MediaModule, (C:\projects\HermesMedia\MediaModule.tlb)
24: // (2) v2.0 stdole, (C:\WINNT\System32\stdole2.tlb)
25: // (3) v4.0 StdVCL, (C:\WINNT\System32\stdvcl40.dll)
26: // ************************************************************************ //
27: {$TYPEDADDRESS OFF} // Unit must be compiled without type-checked pointers.
28: {$WARN SYMBOL_PLATFORM OFF}
29: {$WRITEABLECONST ON}
30:
31: interface
32:
33: uses ActiveX, Classes, Graphics, MediaModule_TLB, StdVCL, Variants, Windows;
34:
35:
36:
37: // ************************************************************************//
38: // GUIDS declared in the TypeLibrary. Following prefixes are used:
39: // Type Libraries : LIBID_xxxx
40: // CoClasses : CLASS_xxxx
41: // DISPInterfaces : DIID_xxxx
42: // Non-DISP interfaces: IID_xxxx
43: // ************************************************************************//
44: const
45: // TypeLibrary Major and minor versions
46: PMediaTCPMajorVersion = 1;
47: PMediaTCPMinorVersion = 0;
48:
49: LIBID_PMediaTCP: TGUID = '{4BF948C1-9EE5-4DBE-9CAA-79ECDAD3C4EB}';
50:
51: CLASS_Media: TGUID = '{5D3BBABF-DD15-4240-B6C6-8D584F8E3696}';
52: type
53:
54: // ************************************************************************//
55: // Declaration of CoClasses defined in Type Library
56: // (NOTE: Here we map each CoClass to its Default Interface)
57: // ************************************************************************//
58: Media = IMedia;
59:
60:
61: // ************************************************************************//
62: // The Class CoMedia provides a Create and CreateRemote method to
63: // create instances of the default interface IMedia exposed by
64: // the CoClass Media. The functions are intended to be used by
65: // clients wishing to automate the CoClass objects exposed by the
66: // server of this typelibrary.
67: // ************************************************************************//
68: CoMedia = class
69: class function Create: IMedia;
70: class function CreateRemote(const MachineName: string): IMedia;
71: end;
72:
73: implementation
74:
75: uses ComObj;
76:
77: class function CoMedia.Create: IMedia;
78: begin
79: Result := CreateComObject(CLASS_Media) as IMedia;
80: end;
81:
82: class function CoMedia.CreateRemote(const MachineName: string): IMedia;
83: begin
```

```
84: Result := CreateRemoteComObject(MachineName, CLASS_Media) as IMedia;
85: end;
86:
87: end.
```

A.6.7 C:\projects\HermesMedia\WinSockUnit.pas

```
1: unit WinSockUnit;
2:
3: interface
4:
5: uses
6: ScktComp,
7: MediaTCPApp, SysUtils, Classes, OoMisc, AdPort, AdWnPort, Psock, Sockets,
8: ExtCtrls, MediaTCPImpl;
9:
10: type
11: TWinSockModule = class(TDataModule)
12: ConnectTimer: TTimer;
13: ClientSocket: TClientSocket;
14: ServerSocket: TServerSocket;
15: ServerTimer: TTimer;
16:
17: procedure ConnectTimerTimer(Sender: TObject);
18:
19: procedure ClientSocketConnect(Sender: TObject;
20: Socket: TCustomWinSocket);
21: procedure ClientSocketConnecting(Sender: TObject;
22: Socket: TCustomWinSocket);
23: procedure ClientSocketDisconnect(Sender: TObject;
24: Socket: TCustomWinSocket);
25: procedure ClientSocketError(Sender: TObject; Socket: TCustomWinSocket;
26: ErrorEvent: TErrorEvent; var ErrorCode: Integer);
27: procedure ClientSocketLookup(Sender: TObject;
28: Socket: TCustomWinSocket);
29: procedure ClientSocketRead(Sender: TObject; Socket: TCustomWinSocket);
30: procedure ClientSocketWrite(Sender: TObject; Socket: TCustomWinSocket);
31: procedure ServerTimerTimer(Sender: TObject);
32: procedure ServerSocketAccept(Sender: TObject;
33: Socket: TCustomWinSocket);
34: procedure ServerSocketClientConnect(Sender: TObject;
35: Socket: TCustomWinSocket);
36: procedure ServerSocketClientDisconnect(Sender: TObject;
37: Socket: TCustomWinSocket);
38: procedure ServerSocketClientError(Sender: TObject;
39: Socket: TCustomWinSocket; ErrorEvent: TErrorEvent;
40: var ErrorCode: Integer);
41: procedure ServerSocketClientRead(Sender: TObject;
42: Socket: TCustomWinSocket);
43: procedure ServerSocketClientWrite(Sender: TObject;
44: Socket: TCustomWinSocket);
45: procedure ServerSocketGetSocket(Sender: TObject; Socket: Integer;
46: var ClientSocket: TServerClientWinSocket);
47: procedure ServerSocketGetThread(Sender: TObject;
48: ClientSocket: TServerClientWinSocket;
49: var SocketThread: TServerClientThread);
50: procedure ServerSocketListen(Sender: TObject;
51: Socket: TCustomWinSocket);
52: procedure ServerSocketThreadEnd(Sender: TObject;
53: Thread: TServerClientThread);
54: procedure ServerSocketThreadStart(Sender: TObject;
```

```
55: Thread: TServerClientThread);
56:
57: private
58: { Private declarations }
59: public
60: { Public declarations }
61: MediaModule: TMedia;
62: end;
63:
64: var
65: WinSockModule: TWinSockModule;
66:
67: implementation
68:
69: {$R *.dfm}
70:
71: procedure TWinSockModule.ConnectTimerTimer(Sender: TObject);
72: begin
73: ConnectTimer.Enabled := false;
74: ServerSocket.Active := False;
75: Form9.Memo1.Lines.Add('Timer1 timeout');
76: ClientSocket.Open;
77: end;
78:
79: procedure TWinSockModule.ClientSocketConnect(Sender: TObject;
80: Socket: TCustomWinSocket);
81: begin
82: {$IFDEF DEBUG}
83: Form9.Memo1.Lines.Add('Client: OnConnect');
84: {$ENDIF}
85: MediaModule.ClientConnect(Sender, Socket);
86: end;
87:
88: procedure TWinSockModule.ClientSocketConnecting(Sender: TObject;
89: Socket: TCustomWinSocket);
90: begin
91: {$IFDEF DEBUG}
92: Form9.Memo1.Lines.Add('Client: OnConnectING');
93: {$ENDIF}
94: MediaModule.ClientConnecting(Sender, Socket);
95: end;
96:
97: procedure TWinSockModule.ClientSocketDisconnect(Sender: TObject;
98: Socket: TCustomWinSocket);
99: begin
100: {$IFDEF DEBUG}
101: Form9.Memo1.Lines.Add('Client: OnDisconnect');
102: {$ENDIF}
103: MediaModule.ClientDisconnect(Sender, Socket);
104: end;
105:
106: procedure TWinSockModule.ClientSocketError(Sender: TObject;
107: Socket: TCustomWinSocket; ErrorEvent: TErrorEvent;
108: var ErrorCode: Integer);
109: begin
110: {$IFDEF DEBUG}
111: Form9.Memo1.Lines.Add('Client: OnError');
112: {$ENDIF}
113: MediaModule.ClientError(Sender, Socket, ErrorEvent, ErrorCode);
114: end;
115:
```

```
116: procedure TWinSockModule.ClientSocketLookup(Sender: TObject;
117: Socket: TCustomWinSocket);
118: begin
119: {$IFDEF DEBUG}
120: Form9.Memo1.Lines.Add('Client: OnLookup');
121: {$ENDIF}
122: MediaModule.ClientLookup(Sender, Socket);
123: end;
124:
125: procedure TWinSockModule.ClientSocketRead(Sender: TObject;
126: Socket: TCustomWinSocket);
127: begin
128: {$IFDEF DEBUG}
129: Form9.Memo1.Lines.Add('Client: OnRead');
130: {$ENDIF}
131: MediaModule.ClientRead(Sender, Socket);
132: end;
133:
134: procedure TWinSockModule.ClientSocketWrite(Sender: TObject;
135: Socket: TCustomWinSocket);
136: begin
137: {$IFDEF DEBUG}
138: Form9.Memo1.Lines.Add('Client: OnWrite');
139: {$ENDIF}
140: MediaModule.ClientWrite(Sender, Socket);
141: end;
142:
143:
144: procedure TWinSockModule.ServerTimerTimer(Sender: TObject);
145: begin
146: ServerTimer.Enabled := false;
147: Form9.Memo1.Lines.Add('Server timer timed out.');
148: ServerSocket.Active := true;
149: end;
150:
151: procedure TWinSockModule.ServerSocketAccept(Sender: TObject;
152: Socket: TCustomWinSocket);
153: begin
154: {$IFDEF DEBUG}
155: Form9.Memo1.Lines.Add('Server: OnAccept');
156: {$ENDIF}
157: MediaModule.ServerSocketAccept(Sender, Socket);
158: end;
159:
160: procedure TWinSockModule.ServerSocketClientConnect(Sender: TObject;
161: Socket: TCustomWinSocket);
162: begin
163: {$IFDEF DEBUG}
164: Form9.Memo1.Lines.Add('Server: OnClientConnect');
165: {$ENDIF}
166: MediaModule.ServerSocketClientConnect(Sender, Socket);
167: end;
168:
169: procedure TWinSockModule.ServerSocketClientDisconnect(Sender: TObject;
170: Socket: TCustomWinSocket);
171: begin
172: {$IFDEF DEBUG}
173: Form9.Memo1.Lines.Add('Server: OnClientDisconnect');
174: {$ENDIF}
175: MediaModule.ServerSocketClientDisconnect(Sender, Socket);
176: end;
```

```
177:
178: procedure TWinSockModule.ServerSocketClientError(Sender: TObject;
179: Socket: TCustomWinSocket; ErrorEvent: TErrorEvent;
180: var ErrorCode: Integer);
181: begin
182: {$IFDEF DEBUG}
183: Form9.Memo1.Lines.Add('Server: OnClientError');
184: {$ENDIF}
185: MediaModule.ServerSocketClientError(Sender, Socket, ErrorEvent, ErrorCode);
186: end;
187:
188: procedure TWinSockModule.ServerSocketClientRead(Sender: TObject;
189: Socket: TCustomWinSocket);
190: begin
191: {$IFDEF DEBUG}
192: Form9.Memo1.Lines.Add('Server: OnClientRead');
193: {$ENDIF}
194: MediaModule.ServerSocketClientRead(Sender, Socket);
195: end;
196:
197: procedure TWinSockModule.ServerSocketClientWrite(Sender: TObject;
198: Socket: TCustomWinSocket);
199: begin
200: {$IFDEF DEBUG}
201: Form9.Memo1.Lines.Add('Server: OnClientWrite');
202: {$ENDIF}
203: MediaModule.ServerSocketClientWrite(Sender, Socket);
204: end;
205:
206: procedure TWinSockModule.ServerSocketGetSocket(Sender: TObject;
207: Socket: Integer; var ClientSocket: TServerClientWinSocket);
208: begin
209: {$IFDEF DEBUG}
210: Form9.Memo1.Lines.Add('Server: OnGetSocket');
211: {$ENDIF}
212: MediaModule.ServerSocketGetSocket(Sender, Socket, ClientSocket);
213: end;
214:
215: procedure TWinSockModule.ServerSocketGetThread(Sender: TObject;
216: ClientSocket: TServerClientWinSocket;
217: var SocketThread: TServerClientThread);
218: begin
219: {$IFDEF DEBUG}
220: Form9.Memo1.Lines.Add('Server: OnGetThread');
221: {$ENDIF}
222: MediaModule.ServerSocketGetThread(Sender, ClientSocket, SocketThread);
223: end;
224:
225: procedure TWinSockModule.ServerSocketListen(Sender: TObject;
226: Socket: TCustomWinSocket);
227: begin
228: {$IFDEF DEBUG}
229: Form9.Memo1.Lines.Add('Server: OnListen');
230: {$ENDIF}
231: MediaModule.ServerSocketListen(Sender, Socket);
232: end;
233:
234: procedure TWinSockModule.ServerSocketThreadEnd(Sender: TObject;
235: Thread: TServerClientThread);
236: begin
237: {$IFDEF DEBUG}
```

```
238: Form9.Memo1.Lines.Add('Server: OnThreadEnd');
239: {$ENDIF}
240: MediaModule.ServerSocketThreadEnd(Sender, Thread);
241: end;
242:
243: procedure TWinSockModule.ServerSocketThreadStart(Sender: TObject;
244: Thread: TServerClientThread);
245: begin
246: {$IFDEF DEBUG}
247: Form9.Memo1.Lines.Add('Server: OnThreadStart');
248: {$ENDIF}
249: MediaModule.ServerSocketThreadStart(Sender, Thread);
250: end;
251:
252: end.
```

A.6.8 C:\projects\HermesMedia\WinSockUnit.dfm

```
1: object WinSockModule: TWinSockModule
2: OldCreateOrder = False
3: Left = 222
4: Top = 858
5: Height = 81
6: Width = 303
7: object ConnectTimer: TTimer
8: Enabled = False
9: Interval = 500
10: OnTimer = ConnectTimerTimer
11: Left = 160
12: end
13: object ClientSocket: TClientSocket
14: Active = False
15: ClientType = ctNonBlocking
16: Port = 0
17: OnLookup = ClientSocketLookup
18: OnConnecting = ClientSocketConnecting
19: OnConnect = ClientSocketConnect
20: OnDisconnect = ClientSocketDisconnect
21: OnRead = ClientSocketRead
22: OnWrite = ClientSocketWrite
23: OnError = ClientSocketError
24: Left = 16
25: end
26: object ServerSocket: TServerSocket
27: Active = False
28: Port = 0
29: ServerType = stNonBlocking
30: OnListen = ServerSocketListen
31: OnAccept = ServerSocketAccept
32: OnGetThread = ServerSocketGetThread
33: OnGetSocket = ServerSocketGetSocket
34: OnThreadStart = ServerSocketThreadStart
35: OnThreadEnd = ServerSocketThreadEnd
36: OnClientConnect = ServerSocketClientConnect
37: OnClientDisconnect = ServerSocketClientDisconnect
38: OnClientRead = ServerSocketClientRead
39: OnClientWrite = ServerSocketClientWrite
40: OnClientError = ServerSocketClientError
41: Left = 88
42: end
43: object ServerTimer: TTimer
```

```
44: Enabled = False
45: Interval = 200
46: OnTimer = ServerTimerTimer
47: Left = 232
48: end
49: end
```