



# Omgivningssimulator till försvars- och övervakningssystem

## An environment simulator for testing defence and surveillance systems

**Tobias Öbrink**

TRITA-NA-E9613

## Referat

Vad är definitionen på en omgivningssimulator? Går det att göra en omgivningssimulator som enkelt kan återanvändas mellan olika projekt? Dessa var de frågor som jag ställdes inför och förväntades svara på.

Definitionen av en omgivningssimulator varierar beroende av vad den är tänkt att användas till; utbildning / övning eller utveckling / testning. Denna rapport behandlar en omgivningssimulator avsedd för testning. Definitionen av en sådan kan kort sammanfattas som; en lägesbildsgenerator som ska generera indata till ett visst datorsystem enligt ett visst scenario.

Enkel återanvändning mellan projekt är en svårare fråga. Att göra en omgivningssimulator för ett specifikt projekt är enkelt eftersom man vet vad som skall simuleras och vilka funktioner som användarna önskar. Att försöka prediktera vad en framtida användare kan tänkas vilja testa hos ett framtida system med okänd konfiguration anser jag är meningslöst. Istället rekommenderar jag att man har en utvecklingsmiljö för simulatorer som tillhandahåller

1. En kunskapsbas med dokumentation och erfarenheter, typlösningar m m

2. Ett enkelt "bassystem"

3. En enkel modell för utbyggnad och modifiering av bassystemet.

I rapporten har jag givit ett förslag till punkterna 2 och 3. Kunskapsbasen har påbörjats i samband med en inventering av existerande simulatorer som jag genomförde som praktikjobb under sommaren innan examensarbetet påbörjades.



## Abstract

What is the definition of an environment simulator? Is it possible to make an environment simulator that easily can be reused by future projects? These were the questions that I was confronted by and that I was supposed to answer.

The definition varies according to the planned purpose of the environment simulator; education / training or development / testing. This report discusses an environment simulator for testing. A short definition for such an environment simulator is as follows; a plot generator that generate input data to a certain computer system according to a certain scenario.

Easy reusability between different projects is a little bit trickier to achieve. To design an environment simulator for a specific project is easy,

because you already know what kind of equipment that is to be simulated and which functions and what type of data that the users wants. To try to predict what a future user wants to test in his particular system, years from now, is in my opinion a pointless waste of time. Instead I recommend to develop a resource base consisting of

1. A reference library with documentation, manuals, solutions to typical problems, etc

2. A simple "base system"

3. A simple model for development and modification of the base system.

In this report there is a draft for no. 2 and 3. Some materials for the reference library I collected during an inventory of "simulators for external equipment" the summer 1995.

# Förord

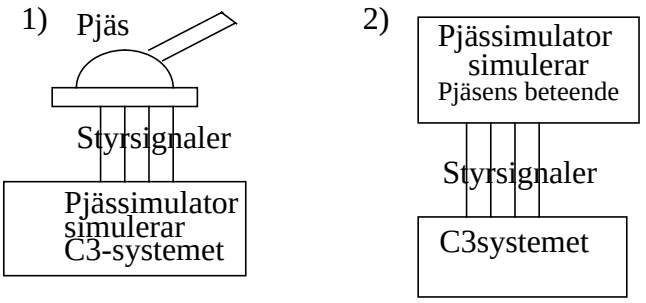
Detta dokument behandlar ett examensarbete i datalogi om 20 poäng vid institutionen för Numerisk analys och datalogi (Nada), Stockholms Universitet (SU).

Examensarbetet utfördes huvudsakligen på D-divisionen, CelsiusTech Systems AB. Ett stort tack till mina handledare; Henrik Eriksson (Nada) och Helena Schytt (Celsius) för att de ställde upp i tid och otid, för uppmuntran och stöd.

# Innehållsförteckning

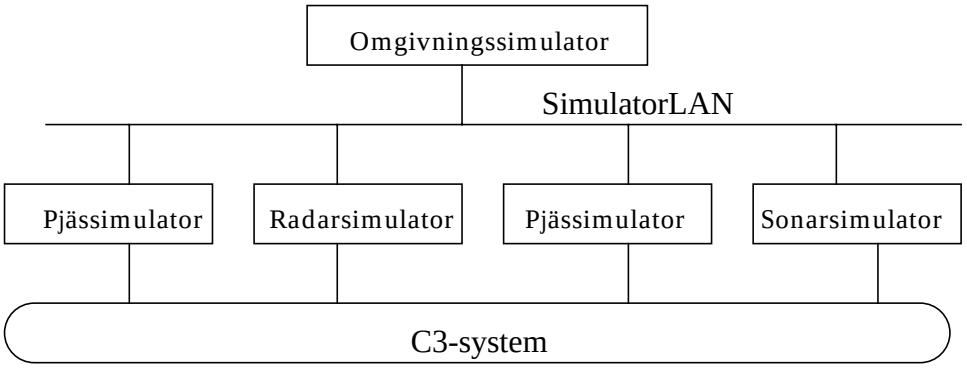
# 1 Inledning

Varför har man simulatorer? När det gäller utveckling av och övning med militära stridsledningss- och övervakningssystem är det både säkrare och billigare att använda simulatorer än skarp utrustning, t ex artilleripjäser, radaranläggningar. Samtidigt ställs stora krav på simulatorernas korrekthet. De måste uppföra sig exakt som den skarpa utrustningen och använda samma kommunikationskanaler och protokoll gentemot det system som ska testas. Därför måste simulatorn testas mot den skarpa utrustningen först innan den kan användas till att testa systemet. En utrustningssimulator är huvudsakligen till för att testa kommunikation.




*Figur 1. Utrustningssimulator*

Varför har man omgivningssimulatorer? Efter att utveckling och testning av enskilda systemkomponenter var för sig är avklarat så skall systemet monteras ihop (integreras) och testas. För att testa samarbete mellan olika systemkomponenter, tidsprestanda hos vissa operationer (systemets reaktionsförmåga), m m så vill man ofta köra vissa testscenarion. Dessa testscenarion genereras antingen manuellt genom att låta varje enskild utrustningssimulator att ge vissa data till systemet, eller genom att koppla ihop utrustningssimulatorerna med en omgivningssimulator som koordinerar de olika utrustningssimulatorernas beteende enligt en scenariobeskriving. Dessutom kan omgivningssimulatorn samla in felrapporter från utrustningssimulatorerna vilket underlättar felsökning.




*Figur 2. Omgivningssimulatorn ihopkopplad med utrustningssimulatorerna över ett LAN*

Nedan följer en kort bakgrund till ämnet för examensarbetet, dess mål, den metod jag använt och min slutsats.

# 2 Kort bakgrund

Under sommaren 1995 gjorde jag en inventering av existerande simulatorer för extern utrustning i projekt på Utvecklingsdivisionen vid CelsiusTech Systems och fick en inblick i hur och vad simulatorerna används till. Syftet var att man ville undersöka vilka av de redan existerande simulatorerna som kunde återanvändas i nya projekt.

Under denna inventering kom jag även i kontakt med omgivningssimulatorer, som dock inte ingick i inventeringen.

I flera projekt har utvecklats någon form av omgivningssimulator som använts för att testa de system man utvecklat. Främst då samordningen mellan olika systemkomponenter, MMI (Man-Machine-Interface) samt belastningstester.

Detta har gjorts genom att koppla ihop simulatorer för specifika, externa utrustningar, som t ex radarsimulatorer, så att de genererar samma bild av omgivningen och sedan köra ett eller flera indatascenarion.

Dessa omgivningssimulatorer har dock hittills inte återanvänts i senare projekt för att de varit

- projektspecifika, d v s specialdesignade för att testa en viss systemkonfiguration.
- mindre väl genomtänkta p g a tidsbrist (Detta för att de tillkommit i ett senare skede av projektet då behov uppstått).
- varit alltför tunga att sätta sig in i för nya projektgrupper, och därför förkastats. En viktig orsak till detta har varit bristande dokumentation.

Istället har nya omgivningssimulatorer tagits fram av nya projekt, samma problemställningar har lösts gång efter gång av olika personer. Mycket tid har lagts ned på att lösa redan lösta problem och upprepande av redan tidigare gjorda misstag. Så hur kan man undvika detta slöseri med tid och resurser? En lösning skulle kunna vara att ta fram en omgivningssimulator som skulle kunna återanvändas mellan olika projekt med ett minimum av modifikationer.

## 2.1 Mål

Målet med mitt examensarbete var uppdelat i två delmål.

### 2.1.1 Delmål 1

Att ta reda på om det går att definiera en omgivningssimulator som enkelt kan återanvändas mellan projekt, vad som krävs av en sådan, vilka problem som finns och förhoppningsvis finna lösningar till dessa problem.

### 2.1.2 Delmål 2

Om möjligt; ta fram ett eller flera designförslag, presentera och implementera (i den ordningen).

Om ej möjligt; släpp ett eller flera av kraven så att en design blir möjlig. Önskad prioritering är:

| Släpp kravet på | Till förmån för |
|---|---|
| Generalitet | Marin tillämpning |
| Run-time-ändring möjligt | ej möjligt |
| Plattformsoberoende | PC-miljö |

# 3 Metod

## 3.1 Definition

Första steget var att ta reda på vad en omgivningssimulator är, dess funktion och vilka krav som ställs av dess huvudsakliga användare, dvs verifierare och integratörer. Under inventeringen och under examensarbetesperioden kom jag i kontakt med många framtida användare vilket resulterade i en lång lista av krav. En sammanfattning av insamlade krav ges nedan. Utifrån denna kravlista ser man att en minimal definition av en omgivningssimulator är; en "svart låda" som genererar omgivningsdata enligt ett givet scenario via samma kommunikationskanaler som skarp utrustning använder.

### 3.1.1 Mjukvara

- Tillhandahålla indata till systemet motsvarande vad externa utrustningar ger. Alltså även temperatur, luftfuktighet, höjd / djup, vindstyrka, lufttryck och utrustningsstatus m m, om detta skulle begäras. (Generalitet.)
- Möjliggöra spelscenarion som kan återupprepas. (Spelfiler.)
- Möjliggöra spelscenarion som kan startas vid en given tidpunkt. (Snabbspolning.)
- Möjliggöra interaktion med simulerade farkoster eller baser genom simulerad kommunikationskanal. (Feedback.)
- Möjliggöra definition av flera "eget system"-objekt som kan interagera med varandra i den simulerade världen. (Flera användare.)
- Möjliggöra styrning av utrustningssimulatorerna. En slags "orderkanal" vid sidan av omvärldssimuleringen. T ex; en radar kan se ett mål medan en annan inte ser det. (Orderkanal.)
- Ge en konsistent bild av omgivningen som kan påverkas av egna aktioner. T ex Vänder eget skepp så skall relativ bäring till omgivande objekt ändras. Skjuter man ner flygplan eller torpederar båtar, så ska de försvinna från omgivningen. (Konsistent omvärldsbild, Feedback.)
- Tidsmarkera och logga felaktig respons från systemet. (Feedback.)
- Vara lätt att använda och sätta sig in i. (Användarvänlighet.)
- Vara väl dokumenterad. (Användarvänlighet.)
- Lätt att skapa nya scenarion. (Användarvänlighet.)
- Run-time ändringar i scenarion möjligt. (Run-time ändringar.)
- Ha åtminstone samma funktionalitet som OBT (On Board Training). (Användarvänlighet.)
- Robust användargränssnitt. (Användarvänlighet.)

### 3.1.2 Hårdvara

- Väl dokumenterad. (Användarvänlighet.)
- Hög realtidsprestanda. (Prestanda.)
- Kunna hantera många komponenter som kräver olika indata. (Prestanda.)
- Kunna hantera stora mängder data, komplexa scenarion och hög uppdateringshastighet för belastningstester av ett system. (Prestanda.)
- Plattformsoberoende eller åtminstone lätt att modifiera till nya plattformar. (Porterbar.)
- Lätt att lägga till och ta bort noder på simulator-LAN:et. (Modifierbar LAN-konfiguration.)

## 3.2 Utredning

Andra steget var att med utgångspunkt från dessa krav utreda om det är möjligt att utveckla en omgivningssimulator som motsvarar dessa krav. Jag har använt följande tillvägagångssätt:

- Genomgång av redan existerande omgivningssimulatorer och hårdvarukomponenter.
- Finna svagheter hos existerande omgivningssimulatorer och hårdvara / kombinationer av hårdvara och försöka finna lösningar till dessa problem. (Se *Appendix*, avsnittet *Omgivningssimulatorer*.)
- Jämföra funktionalitet hos omgivningssimulatorerna och hårdvaran med de behov som specificeras i Kraven ovan.
- Leta i litteraturen efter lösningar till funna problem, nya ideer och goda råd, metoder och förklaringar till oklara begrepp och förkortningar.
- Intervjua insatta personer om hur systemen, som omgivningssimulatorn skall interagera med, fungerar.
- Följa ett pågående utvecklingsprojekt.

## 3.3 Design

Parallellt med utredningen och med utgångspunkt från insamlad information togs fram en mer eller mindre generell design av en omgivningssimulator enligt Delmål 2. Det senast framtagna designförslaget finns under *Designförslag*.

## 3.4 Implementation

Vissa delar, som näthantering och gränssnitten mellan de olika delarna i designen, måste testas under utredningen. Dessa delar av designförslaget finns i skrivande stund som fungerande program för PC / DOS och till Sun / Unix.

# 4 Slutsats

## 4.1 Delmål 1

Min slutsats är att det inte är möjligt att göra en färdig produkt som går att återanvända direkt utan modifieringar. Varje nytt projekt har nya krav och förutsättningar, ny utrustning med nya funktioner utvecklas ständigt och skall integreras i systemet. Det är inte möjligt att ta fram en generell design för en godtycklig omgivningssimulator av "plug-in-and-go"-typ till ett godtyckligt system eftersom man inte kan förutsäga vilket slags komponenter som kommer att ingå i ett framtida projekt. Det som är möjligt och, vad jag tror, realiserbart är att ha en utvecklingsmiljö för simulatorer som tillhandahåller

1. En kunskapsbas med dokumentation och erfarenheter, typlösningar m m.

2. Ett enkelt "bassystem".

3. En enkel modell för utbyggnad och modifiering av bassystemet.

Generaliteten ligger då i utvecklingspotentialen. Enkel modifiering ger återanvändbarhet.

## 4.2 Delmål 2

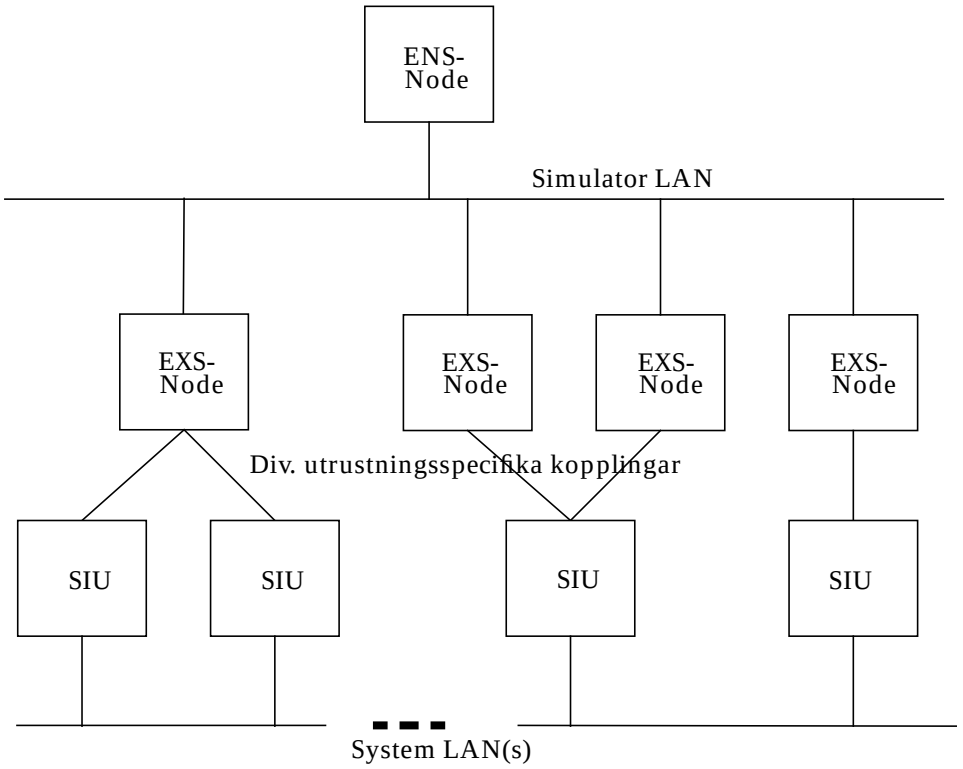
Kunskapsbasen har påbörjats i samband med den inventering av existerande simulatorer som jag utförde sommaren 1995, samt det material som jag insamlat om omgivningssimulatorer under examensarbetet hösten 1995.

För varje nytt system som skall testas kommer sannolikt konfigurationen av utrustningssimulatorer att vara ny, liksom vilka omgivningsdata som skall genereras och överföras. Vissa egenskaper återkommer dock i de omgivningssimulatorer jag tittat på. Dessa egenskaper är grunden till bassystemet.

Under kapitlet *Designförslag* har jag gett ett förslag till ett bassystem och en modell för utbyggnad baserat på de simulatorer som jag tittat på.

# Designförslag

Denna del av rapporten behandlar ett förslag till design av ett bassystem som enkelt ska kunna byggas ut till ett omgivningssimulatorsystem till ett visst projekt. Den innehåller också ett utkast till modell för hur detta bassystem ska kunna byggas ut utan att förlora de egenskaper som motsvarar de krav som inkommit (Se Kapitlet *Metod,* avsnitt *Definition*.)




*Figur 3. Översiktlig beskrivning av ett omgivningssimulatorsystem*

## Terminologi

**ESS :**

Environment-Simulator-System. Definieras som ett SimulatorLAN, en eller flera ENS-Noder, ett godtyckligt antal EXS-Noder samt diverse utrustningsspecifika kopplingar. (Se kapitlet *ESS*.)

**C3-system** :

Command, Control and Communication system. Består av ett System-LAN, ett antal SIU, m m. Ett datorsystem vars omgivning ESS ska simulera.

**ENS-Node** :

ENvironment-Simulator-Node. Den nod på SimulatorLANet som innehåller ENS och nätadministrationen. (Se kapitlet *ENS*.)

**EXS-Node** :

EXternal equipment-Simulator-Node. Den nod på SimulatorLANet som simulerar en viss extern utrustning (sensorer, m m) tillhörande ett visst C3-system. Kan innehålla flera EXS. (Se kapitlet *EXS*.)

**SIU** :

Standard Interface Unit. Ingår inte i ESS.

# 1 Designstrategi

För att försöka uppfylla kraven i Kapitlet *Metod,* avsnittet *Definition* ovan har jag valt följande lösningar.

Rubrikerna i detta kapitel har givits inom parentes efter de krav i Kapitlet *Metod,* avsnittet *Definition* som uppfylls av lösningarna givna under rubrikerna.

## 1.1 Generalitet

Kan approximeras med kombinationen utbyggbarhet och användarvänlighet. Svårigheten här ligger i att hitta en modell för utbyggnad som också är lättanvänd och begriplig. Införandet av nya EXS kan innebära att nya data måste kunna uppdateras i omvärlden och föras över till EXSen. Dessutom tillkommer hela tiden ytterligare krav på funktionalitet i samband med att med att man vill testa nya funktioner i C3-systemet. Hur kunna enkelt introducera denna data i omvärlden?

En ide är att använda en objektorienterad design av den simulerade omvärlden med en bas av klasser som är såpass generella att de alltid kan användas för att härleda en ny klass av objekt, som innehåller den nya informationen.

Sedan skapar man en ny Meddelandetyp (se kapitlet *Meddelandetyper*) med tillhörande extraktorfunktioner som kan överföra denna information till EXS. Detta medför att SIM- och NET-modulerna i ENS och EXS (de som berörs av ändringen) måste kompileras om.

Redan existerande Meddelandetyper får INTE modifieras, p.g.a att bakåtkompatibilitet är önskvärt om EXS:er som använder den gamla versionen ska återanvändas.

Skapande av nya Meddelandetyper kan leda till hög nätbelastning om man har en konfiguration av nya och gamla EXS:er, som begär olika varianter av i stort sett samma information.

## 1.2 Spelfiler

Ett scenario är tänkt att byggas med hjälp av ett grafiskt gränssnitt och sedan lagras på en fil. Scenariot kan sedan laddas in och köras ett godtyckligt antal gånger. Under definitionsfasen av ett scenario kan alltid ändringar göras och scenariot kan sparas.

## 1.3 Snabbspolning

Ett scenario kan "snabbspolas" från spelstart till önskad starttid genom att spelets tid uppdateras med en given faktor gånger reell tid. Detta innebär att snabbspolningshastigheten begränsas av scenariots uppdateringsintervall. Under snabbspolning utförs varken uppdatering av det grafiska gränssnittet eller sändning av data till EXS.

## 1.4 Feedback

C3-systemets påverkan på omvärlden via extern utrustning simuleras med hjälp av RPC / XDR-procedurer som arbetar direkt på den simulerade omvärlden. Felrapportering från EXS sker också med RPC / XDR.

## 1.5 Flera användare

För att möjliggöra testning av kommunikationsutrustning och rapporteringsfunktioner behövs ibland att två eller flera C3-system kan dela samma omgivning. Detta kräver att

1. Flera egna system måste kunna finnas i omvärlden.
2. Olika omgivningsdata måste kunna ges till olika C3-system
3. EXS måste veta till vilket eget-system-objekt de tillhör
4. Vilket objekt som helst måste kunna väljas som eget system
5. Objekt valda som eget system till något anslutet C3-system får ej kunna raderas
6. Måste hålla reda på till vilket objekt / eget-system som vissa omgivningsdata hör

## 1.6 Orderkanal

Det tillkommer alltid nya krav på ytterligare funktioner för styrning av simulerad utrustning, d v s EXS. T ex vid testning av C3-systemets reaktion på felaktig indata, utrustningsfel, m m. För att dessa "order" ska kunna läggas in vid en viss tidpunkt eller i en viss sekvens måste de ha en koppling till scenariohanteringen. En lösning är att ha en lista av tidsbestämda "globala" händelser samt en lista över anslutna EXS och deras adresser, för unicast av order till en viss EXS. Nya order kan alltid definieras enligt metoden beskriven under Generalitet ovan.

## 1.7 Konsistent omvärldsbild

Överföring av, för C3-systemets utrustning, relevant information från ENS till EXS sker med broadcast / multicast för att upprätthålla en konsistent bild av omvärlden gentemot C3-systemet. På så vis får alla EXSer samtidigt tillgång till data.

## 1.8 Användarvänlighet

Det finns två användarkategorier. Kategori ett är personer som vill använda ESS som ett testverktyg för att testa ett eller flera C3-systems funktioner, gränssnitt m m. Kategori två är personer som står inför utmaningen att modifiera ESS till att motsvara den 1:a kategorins krav. Kategori ett har ingen kunskap om hur ESS är uppbyggt. Kategori två har förhoppningsvis någon insikt i hur ESS fungerar inuti.

För kategori ett är det grafiska användargränssnittet och funktionaliteter som t ex hur och när man kan spara ett scenario, framåt- och bakåtspolning av ett scenario, vad den där lilla "blippen" i nedre vänstra hörnet är till för, m m av störst intresse.

För kategori två är det viktigt att snabbt kunna hitta och rätta fel (som t ex den där lilla "blippen" i nedre vänstra hörnet), skriva nya EXS, introducera nya data och funktionalitet i ENS och distribuera dessa data.

Således krävs användarmanual och teknisk beskrivning.

För att inte kategori ett ska tröttna bör användargränssnittet likna något som redan används, t ex OBT, om inte i utseende så åtminstone i funktionalitet och terminologi. Dessutom bör ESS vara robust.

För att inte kategori två ska tröttna så bör kod och funktionalitet delas upp i oberoende moduler med klart definierade gränssnitt, och plattformsberoende delar och "specialtricks" vara i egna moduler som ska vara så små som möjligt.

## 1.9 Prestanda

1. Endast relevanta data överförs.

EXS kan "prenumerera" på, för den utrustning som ska simuleras, intressant information från den simulerade omvärlden.

Multicast möjliggör att beslut om huruvida ett mottaget meddelande innehåller relevant information kan tas vid adresshanteringsnivå. Om man vill ha en viss sorts data lyssnar man på den multicastadress där denna data sänds.

Önskad information specificeras av EXS-programmeraren i en Indataspecifikationsfil. Denna fil avläses automatiskt då EXSen ansluter sig till ESS (OBS. En EXS-Nod kan innehålla flera EXS-SIM).

2. Viss uppdatering hanteras i EXS.

Detta är en avvägning mellan hur mycket processorkraft och minne som lokal uppdatering av den del av omvärlden som en EXS ser kostar jämfört med mottagning av motsvarande datamängd över nätet. Distribuerad uppdatering innebär också att mängden kontrollinformation som behövs för att upprätt-hålla en konsistent omvärldsbild och korrigera eventuella informationsförluster på nätet ökar drastiskt. Denna extra information bör man också ta med i beräkningen. Distribuerad uppdatering är dock att föredra om man behöver en hög uppdateringstakt. Ett exempel på en sådan beräkning finns i *Appendix*, avsnitt *Distribuerad uppdatering*.

För t ex radarmåldata är det en fördel om position uppdateras i EXS med avseende på hastighet och hastigheten uppdateras med avseende på maxaccelerationen. Eftersom huvuddelen av all data som hanteras är radarmåldata och dessa två funktioner kräver relativt lite jobb jämfört med näthanteringen. Radarmålrelaterade sändningar reduceras då till ändringar av hastighet, riktning, m fl. egenskaper utom position. Dock tillkommer positionssynkronisering för att förhindra att bilderna i olika radarsimulatorer glider isär med tiden och extrasändningar av ändringsmeddelanden för att parera förlust av paket på nätet samt meddelanden för borttagning av objekt i EXS.

## 1.10 Porterbar

Använder UDP / IP för kommunikation mellan ENS och EXS. Finns till de flesta datormärken. Designen är uppdelad i moduler med klart definierade gränssnitt och funktionalitet så att maskin- och operativsystemsberoende delar lätt kan modifieras eller bytas ut vid portering. Dessutom används XDR vid all överföring av data, så att ESS även fungerar med en blandad konfiguration av datorer och programspråk med olika datarepresentation.

## 1.11 Modifierbar LANkonfiguration

Använder en client-server modell för automatisk registrering av anslutna EXS:er hos ENS, baserad på RPC / XDR. Modellen innehåller dessutom funktioner för att meddela ENS vilka data som EXSen behöver från omvärlden, funktioner för att påverka objekt i omvärlden samt en felrapporteringsfunktion. Konfigurationer innehållande datorer av olika märken understödjs eftersom all information som överförs över SimulatorLANet konverteras till XDR.

# 2 ESS

## 2.1 Övergripande beskrivning

ESS består av ett SimulatorLAN, ENS- och EXS-noder. LANet bör understödja broadcastsändning. ENS-nod(erna) hanterar en allmän omvärldsbild definierad som ett scenario innehållande objekt av olika slag, med olika egenskaper, och händelser på dessa objekt samt också "globala" händelser som insättning / borttagning av objekt m m. Denna omvärldsbild förmedlas över LANet till EXS-noderna där informationen konverteras till ett utrustningsspecifikt format samt att information som inte skulle uppfattas av den "riktiga" utrustningen ej överförs till C3-systemet av EXS. En EXS-nod simulerar alltså en viss utrustning, t ex en sensor, och får information om den omvärld som den delar med övriga EXS från ENS.

Vissa utrustningar, t ex en kommunikationskanal, har möjlighet att påverka omgivningen. Dessutom kan central felhantering vara önskvärt för att slippa springa runt till varje enskild EXS-nod när något går fel. För att understödja detta ingår också en Client-Server-förbindelse mellan ENS och EXS. Denna är skriven i RPC / XDR och är sålunda enkelt utbyggbar. Denna Client-Server-förbindelse hanterar också viss nätadministration. Om inte RPC understödjs så kan motsvarande funktionalitet enkelt implementeras med något annat hjälpmedel.

ESS är tidssynkroniserat för att central felhantering ska fungera, samt för att möjliggöra distribuerad uppdatering (se *Appendix*, avsnitt *Distribuerad uppdatering*).

ESS är tänkt att fungera med varierande antal EXS-noder som simulerar godtyckliga utrustningar. En EXS-nod ska kunna kopplas in när som helst och ENS ska enkelt kunna modifieras att tillhandahålla minst den information som begärts av EXS-noderna. Om informationen redan genereras i ENS vid tiden för inkopplingen ska ingen modifiering av ENS behövas, utan ENS ska kunna börja överföra informationen utan att pågående scenario ska behöva avbrytas.

## 2.2 Kommunikation

Kommunikation är det som huvudsakligen definieras i detta designförslag. Jag har försökt designa en modell för effektiv överföring av komplexa datatyper samtidigt som nätlasten hålls så låg som möjligt.

Över SimulatorLANet körs UDP/IP. Kommunikationen mellan ENS- och EXS-processerna i ESS-systemet kan indelas i fyra delar.

### 2.2.1 Client-Server förbindelse mellan ENS och EXS

En RPC-baserad Client-Server-modell används för att ge EXSerna en möjlighet att påverka den simulerade omvärlden och rapportera fel, samt för administrativa sysslor som Connect, Disconnect och Subscribe. (Se *Client-Server gränssnitt*.)

### 2.2.2 Unicast

Central styrning av specifika EXS:er sker med hjälp av Order till en viss adress. (Se kapitlet *Sändningstyper*.)

### 2.2.3 Broadcast

Hanterar tidssynkronisering, överföring av globala konstanter. (Se kapitlet *Sändningstyper*.)

### 2.2.4 Multicast

En Multicast-modell används för att förmedla den omvärldsbild som EXSerna beställt i sina Indata-specifikationer. (Se kapitlen *Sändningstyper* och *Indataspecifikation*.)

## 2.3 Tidssynkronisering

Systemets alla noder är tidssynkroniserade. Ett tidssynkroniseringsmeddelande sänds med broadcast av ENS till alla EXS med ett visst tidsintervall som är sättbart. Varje konfiguration av noder, LAN-typ och datortyp kräver olika tidssynkroniseringar för att uppfylla noggrannhetskrav, begränsningar i LAN-kapacitet och datorkapacitet.Tidssynkroniseringsmeddelanden tas automatiskt omhand så snart som möjligt i EXS-NET.

## 2.4 Riktlinjer

Det är viktigt att man håller isär simulering av omgivningen och simulering av en viss utrustning. När man återanvänder simulatorer och skriver nya så är det lätt hänt att man för enkelhets skull, för att inte belasta nätet i onödan, m fl. ursäkter placerar utrustningsspecifik information och beräkningar på denna i ENS. Resultatet blir förstås att modulariteten upplöses och att man till slut inte har en aning om var i koden saker och ting händer.

Nedan följer några allmänna riktlinjer om var viss information och funktionalitet bör placeras i ESS.

1. Typexempel på information som bör hanteras endast i EXS-Nod (Ej i ENS.)

   Variabler:

   - Egna radarsystems räckvidd, mode och horisont
   - Plot resolution, clutter density och position variance
   - Egna vapensystems räckvidd
   - Missile kill radius
   - Vilka track som skall raporteras vidare
   - Utrustningsstatus, som pjäsvinklar / elevationer, m m

   Funktioner:

   - Ballistikberäkningar
   - Beräkning och utplacering av "splash plots"
   - Träff / miss med SAM / SSM / ASM,ASW, IR, m m
   - Track init / delete / fail
   - Weapon designation, Target allocation / deallocation / fail
   - Uppdatera utrustningsstatus
   - Roll och Pitch (hårdvara). Min / max vinkel fås av ENS
   - Avstånd till botten.

2. Typexempel på information som bör hanteras i ENS-Nod
   - Egenskaper hos objekt i omvärlden, t ex position (x, y, z), hastighet (dx, dy, dz), storlek, m m
   - Händelser på objekten, t ex kurs- och hastighetsändring (newdx, newdy, newdz) vid en viss tidpunkt eller position, på- och avslagna emitteregenskaper, m m.
   - Globala händelser, t ex insättning och borttagning av objekt i omvärlden, väderomslag, order till specifika EXS, m m.
   - Administration av SimulatorLAN, t ex tidssynkronisering, antal säkerhetssändningar vid distribuerad uppdatering, sändningsfrekvens för olika Meddelandetyper, m m.

## 2.5 Införande av ny data i simulering

I nuvarande version av ESS-implementationen krävs att man utför nedanstående punkter när man vill simulera någon ny företeelse i omvärlden, som t ex ESM-egenskaper hos objekt, el. Dyl. Denna lista kommer att bli kortare.

1. Skapa en ny subklass, som innehåller önskad info, till ngn klass som redan ingår i simuleringen.
   Om det redan finns en klass som innehåller en del av den information som önskas så gör man en subklass till den, annars så gör man en subklass till Something.

2. Uppdatera defines.h
   Ny klassidentifikator införs och NO_CLASSES inkrementeras.

3. Uppdatera game.h
   Inkludera filen med den nya klassen.

4. Skapa en ny Meddelandetyp till klassen.
   Uppdatera ens.x med en ny Meddelandetyp i XDR enligt samma mönster som de exempel jag gjort.

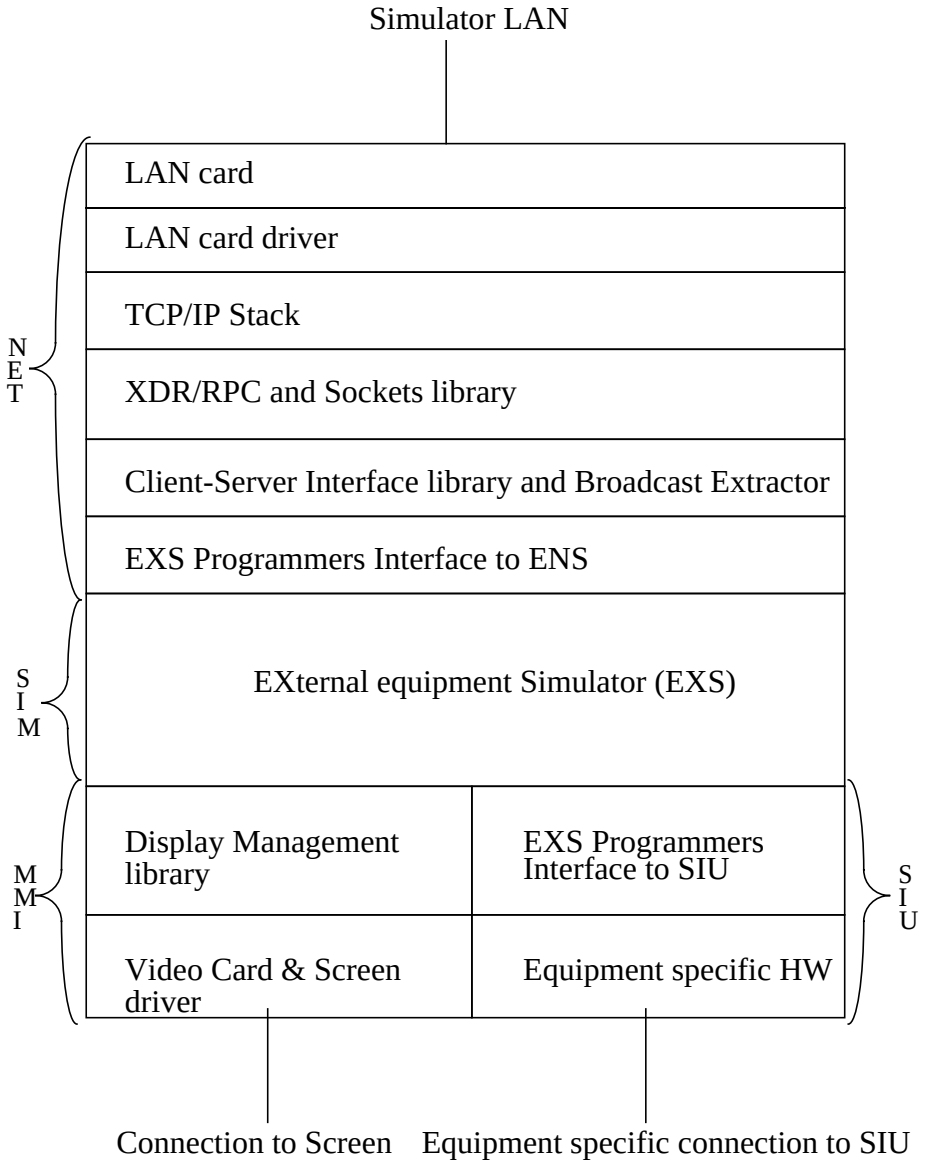
5. Uppdatera ClassID, InfoType, FieldName och FieldValue i Client-Server gränssnittet.
   Även dessa ska finnas i ens.x. Lägg till klassidentifikator (samma som i defines.h) i ClassID, din nya Meddelandetyps namn i InfoType, namn på de fält som finns i din nya Meddelandetyp i FieldName och fältens typer i FieldValue.

6. Uppdatera Game::load_scenario, SNI::send_object, SNI::send_object_event, MMI::update i ENS.

7. Uppdatera NET::receive_object i EXS.

# 3 EXS

En EXS-nod består av en nätkommunikationsdel (EXS-NET), en eller flera simulatordelar (EXS-SIM), en MMI-del (EXS-MMI) och en del som hanterar ett SIU-gränssnitt som är specifikt för den utrustning som skall simuleras. NET-, MMI- och SIU-delarnas implementation beror av underliggande hårdvara, operativsystem, m m. SIM-delen kan implementeras i något plattformsoberoende högnivåspråk. EXS simulerar en specifik utrustning, så funktionalitet och användargränssnittskiftar mellan varje specifik EXS. Därför är beskrivningen av EXS-SIM, EXS-MMI och EXS-SIU mycket allmän.




*Figur 4. Ett exempel på en EXS-Nod*

## 3.1 Nätkommunikationsdelen i en EXS-nod

Kallas kort för EXS-NET. Implementeras i C / C++ med BSD Sockets, Sun RPC och XDR för att få så portabel programvara som möjligt. Följande funktionalitet understödjs.

### 3.1.1 EXS-NET's gränssnitt mot EXS-SIM

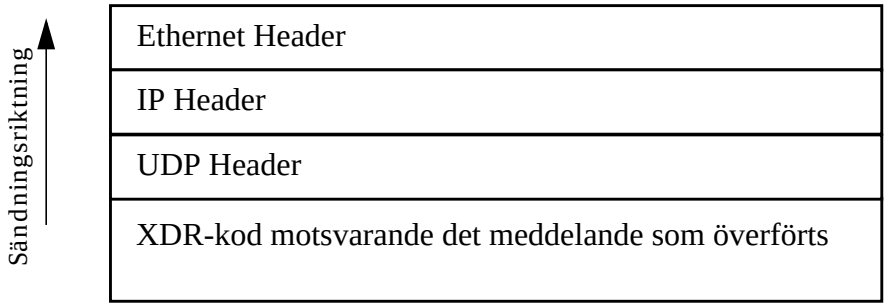
Bibliotek av funktioner för åtkomst av meddelanden i en array av listor samt de funktioner som RPC-klienten understödjer (se *Client-Server gränssnitt*). En startupprocedur som tar hand om uppkopplingen mot ENS, "prenumeration" på information, skapande av arrayen av listor, m fl initieringsrutiner bör finnas för att inte EXS-programmeraren ska behöva sätta sig in i detalj hur EXS-NET fungerar.

### 3.1.2 EXS-NET's gränssnitt mot ENS-NET

Se kapitel *ESS*, avsnitt *Kommunikation*. En avbrottshanterare eller daemon tar hand om inkommande meddelanden enligt prioritetsordningen: Broadcast, Unicast, Multicast, RPC-Server reply.

### 3.1.3 Extrahering av data

Extraktorn i EXS-NET får den del av datagrammet som är under UDP-headern (se bild nedan). Denna del avkodas och sedan uppdateras datat i en array i EXS-NET.




*Figur 5. Ett datagram som det ser ut på SimulatorLANet (Ethernet)*

### 3.1.4 Uppbyggnad

- Avbrottshanterare eller liknande som tar hand om inkommande meddelanden.
- Startupprocedur som sköter uppkoppling mot ENS vid omkoppling till Remote mode i EXS-SIM (se *Körningsexempel, Start-Up*.)
- RPC-klient (se *Client-Server gränssnitt*.)
- Procedurer för avkodning av Meddelandetyper (Objectdata, m fl.)
- Array av länkade listor, vars ordning i arrayen samt innehåll bestäms av indataspecifikationen

### 3.1.5 Datahantering

- Indata lagras i en array av listor uppbyggd enligt indataspecifikationen. Vid kommunikation med ENS används XDR.

### 3.1.6 Ex: Indata till en sonar- eller radarsimulator:

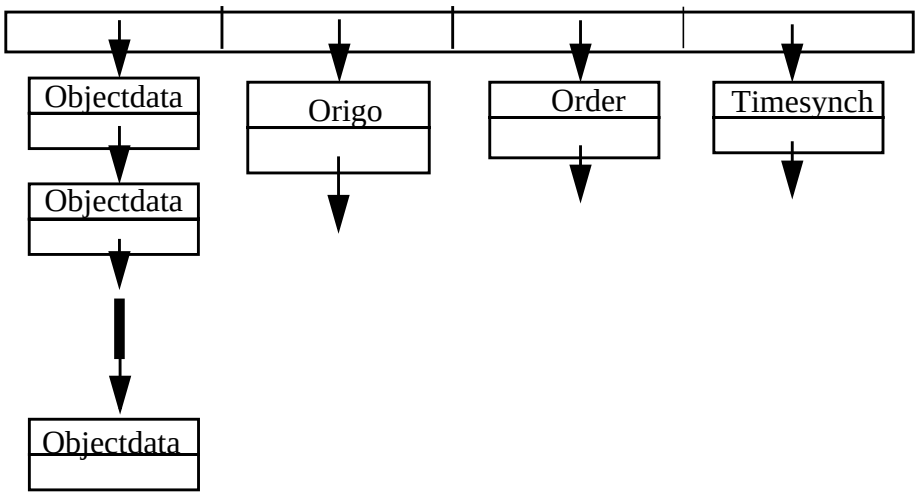
I Indataspecifikation:

Objectdata

Origo

Order

Ger en array med detta innehåll:




*Figur 6. Indataarray i EXS-nod*

Timesynch sänds alltid till alla EXS-noder och läggs alltid i slutet av arrayen.

## 3.2 Simulatordelen i EXS-Nod

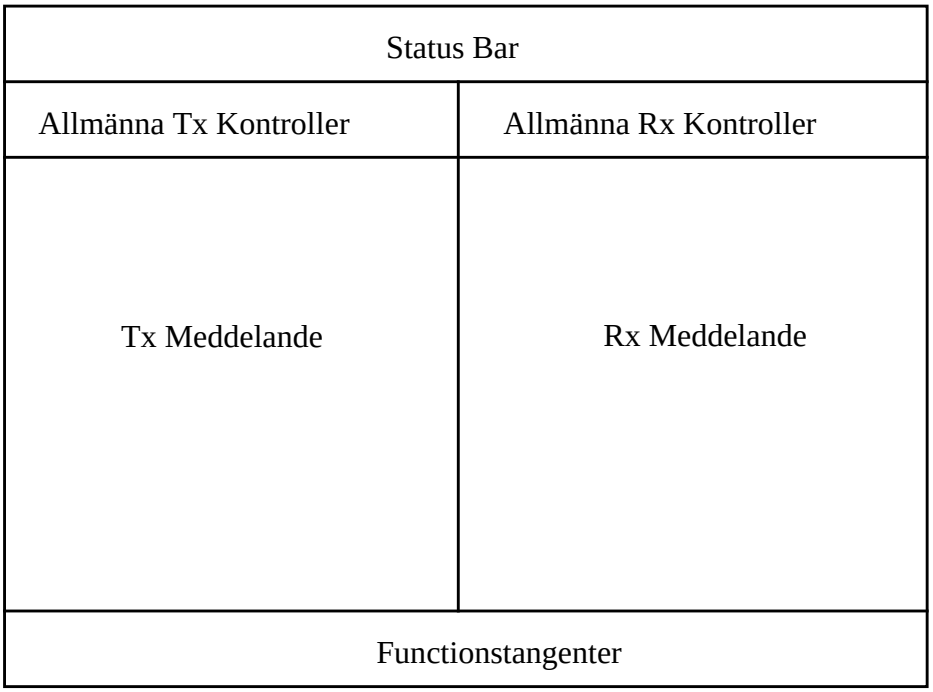
Kallas kort för EXS-SIM. Baserad på Tekniker-Terminalen, en testutrustning. Simulatorn kan simulera extern utrustning mot C3-systemet, och C3-systemet mot extern utrustning. Huvudsaklig användning är att testa gränssnittet mellan C3-systemet och extern utrustning på meddelandenivå. Det finns möjlighet att titta på och modifiera ankomna och avsända meddelanden till/från C3-systemet och utrustningen i hexadecimal form för att simulera fel på bitnivå.

Kan köras i Local- eller Remote-mode. När EXS-SIM befinner sig i Local-mode är den ej ansluten till ESS och man kan editera och lagra meddelanden samt sända cykliskt och manuellt. När den befinner sig i Remote-mode fås innehållet i meddelandena från ENS-SIM. Dessa meddelanden kan fortfarande editeras, men beteendet i detta fall är ospecificerat.

Lokal uppdatering av meddelanden mellan sändningar av data från ENS kan förekomma. Hur denna går till är upp till EXS-programmeraren.

### 3.2.1 Funktion

:




Tx = Sänd (Transmit)
Rx = Ta emot (Receive)

*Figur 7. EXSs (teknikerterminalens) användargränssnitt*

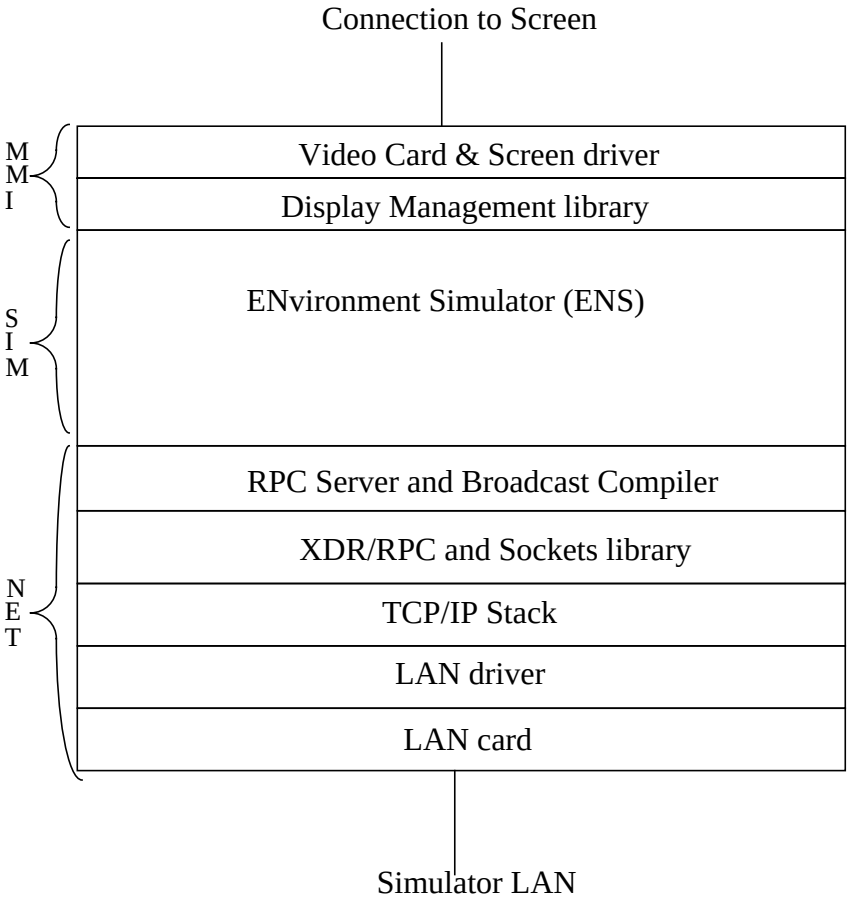
Inmatad information (I vänstra fönsterhalvan) läggs direkt in på rätt plats i meddelandet som skall sändas till SIU (eller till utrustningen vid testning av denna). Meddelandena sparas i en lista. Meddelanden från SIU (eller utrustningen) lagras i en annan lista och kan visas i högra fönsterhalvan. När EXS-SIM är i Remote mode så läser den värden i en array av listor i EXS-NET istället för på skärmen (eller både och). Dessa värden behöver i de flesta fall konverteras eller att flera värden från ENS används för att beräkna ett värde till ett fält i ett meddelande till SIU.

### 3.2.2 Datahantering

- I simulatorkärnan ligger meddelandena i en länkad lista efter sändningsordning.
- Data kan loggas på en fil. Flera uppsättningar av meddelanden kan sparas och användas senare.

# 4 ENS

En ENS-nod består av en nätkommunikationsdel (ENS-NET), en simulatordel (ENS-SIM) och en MMI-del (ENS-MMI). NET- och MMI-delarnas implementation beror av underliggande hårdvara, operativsystem, m m. SIM-delen kan implementeras i något plattformsoberoende högnivåspråk. ENS genererar en allmän omvärldsbild och sänder en delmängd av genererad information vidare till EXS.




*Figur 8. Ett exempel på en ENS-nod*

## 4.1 Nätkommunikationsdelen i ENS-Nod

Kallas kort för ENS-NET. Implementeras i C / C++ med BSD Sockets, Sun RPC och XDR för att få så portabel programvara som möjligt.

Följande funktionalitet understödjs:

### 4.1.1 ENS-NET's gränssnitt mot ENS-SIM

Bibliotek av funktioner för olika typer av sändningar som anropas av tidshanteringen i ENS-SIM:

- void send_object (Element* object)
  Om meddelanden av motsvarande typ som objektet (se avsnittet *Objekt i simulerade världen* nedan) beställts av någon EXS så konverteras objektet till en Meddelandetyp som sedan sänds.
- void time_synch()
  Sänder ett Timesynch-meddelande med broadcast.

### 4.1.2 ENS-NET's gränssnitt mot EXS-NET

Se kapitel *ESS*, avsnitt *Kommunikation*. ENS-NET innehåller en RPC-server, som lyssnar efter kommandon från EXS-NET. Kommandona utförs och resultatet returneras (se kapitlet *Client-Server gränssnitt*).

Använder XDR och BSD Sockets för överföring av omgivningsdata, order, tidssynkronisering m m. XDR tar hand om konvertering av byteordning och överföring av complexa datatyper. (Se kapitlet *Meddelandetyper*.)

### 4.1.3 Uppbyggnad

- RPC-server (se kapitlet *Client-Server gränssnitt*).
- Bibliotek av funktioner som hanterar XDR-kodning och sändning av information.

### 4.1.4 Datahantering

Funktionerna som specificerats i Client-Server gränssnittet opererar direkt på ENS-SIM.

Vid kommunikation med EXS används XDR. (Se kapitlet *Meddelandetyper*.)

## 4.2 Simulatordelen i ENS-Nod

Kallas kort för ENS-SIM. ENS-SIM hanterar scenarion och spel. Scenarion (spelfiler) kan editeras direkt, men detta rekommenderas inte. Istället rekommenderas att man skapar ett spel, sätter det i speldesignsmode och startar det. När man är nöjd med designen kan man spara det.

### 4.2.1 Scenario

Är en abstrakt beskrivning av vad som ska simuleras, d v s en spelfil.Innehåller beskrivningar av en mängd objekt, händelser på dessa objekt, globala händelser och order till EXS

### 4.2.2 Spel

Hanterar simulering av omvärlden enligt något fördefinierat scenario. Kan hantera grupperingar av objekt på samma sätt som enstaka objekt.

Ett spel kan befinna sig i ett av fyra tillstånd:

1. Speldesign
   Under speldesign är ENS frånkopplat från nätet. När som helst kan spelet "frysas" (se *Fryst spel* nedan) eller avslutas (se *Stoppat spel* nedan), objekt kan sättas in, modifieras och tas bort, och händelser på objekt kan sättas in, modifieras och tas bort, liksom globala händelser.

2. Fryst spel
   När spelet är fryst kan det sparas på disk, objekt kan sättas in, modifieras och tas bort, och händelser på objekt kan sättas in, modifieras och tas bort, liksom globala händelser. Ingen uppdatering av objekt, speltid eller händelser sker då spelet är fryst. Ett fryst spel kan fortsätta vid den speltid där det frystes, eller avslutas. Speltiden relativt riktig tid kan ändras.

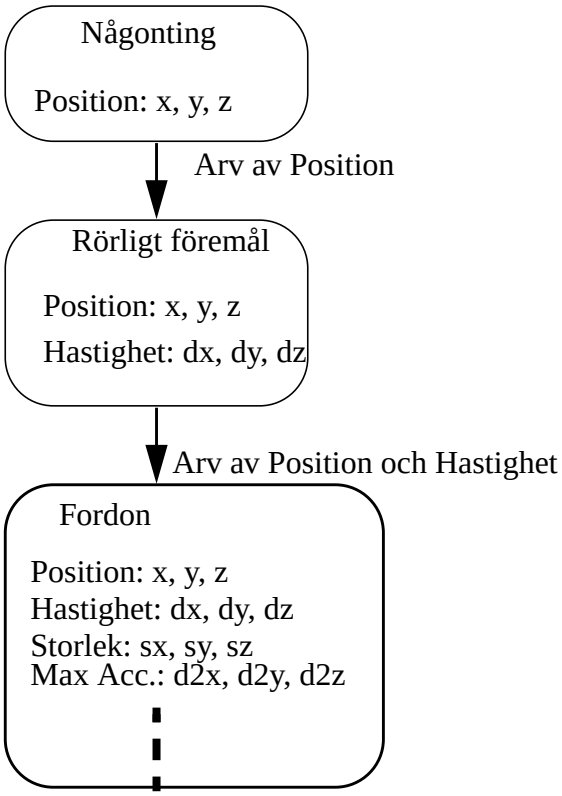
3. Spelkörning
   Under spelkörning är ENS kopplat till nätet. Spelet kan avslutas (se *Stoppat spel* nedan). Objekt kan sättas in och tas bort, och händelser på objekt kan sättas in, modifieras och tas bort, men man bör vara medveten om att detta kan innebära försening av händelser och uppdateringar av objekt.

4. Stoppat spel
   De objekt som fanns i spelet när det stoppades finns kvar, liksom de händelser som inte hunnit utföras. Objekt kan sättas in, modifieras och tas bort, och händelser på objekt kan sättas in, modifieras och tas bort. Spelet kan sparas på disk och / eller ett nytt scenario kan läsas in från disk. Spelet kan startas med samma speltid som då det avslutades. Spelets mode kan ändras.

### 4.2.3 Objekt i Simulerade världen, Basklasser:

Ett spel innehåller en lista med objekt. Objekt kan skapas, raderas och modifieras. Ett objekts egenskapsvärden kan ändras genom direkt modifiering eller genom tidsbaserade händelser. Direkt modifiering av ett objekts egenskapsvärden medför att dess händelselista raderas. Ett objekt bör ha defaultvärden för sina egenskaper. På så sätt kan objekt enkelt skapas med ett kommando som är gemensamt för alla objektklasser. Varje objekt innehåller två identifikatorer; ett som bestämmer klasstillhörighet och ett spelunikt ID-nummer som tilldelas av spelet då objektet skapas. Kopiering av objekt planerat att införas.

- Mobila objekt
  Innehåller medlemsvariabler motsvarande innehållet i Meddelandetypen Objectdata + åtkomst- och modifierings- funktioner, en uppdateringsfunktion, en lista av händelser.
- Stationära objekt
  Innehåller medlemsvariabler motsvarande innehållet i Meddelandetypen Objectdata + åtkomst- och modifierings- funktioner, en uppdateringsfunktion, en lista av händelser. Hastighet och kurs = 0.
- Omgivningsobjekt
  Kan inte raderas. Representerar omgivningen för ett Eget system. Innehåller medlemsvariabler motsvarande innehållet i Meddelandetypen Environmentdata + åtkomst-, modifierings- funktioner, en uppdateringsfunktion, en lista av händelser.




*Figur 9. Arv används för enkel utbyggnad av simulerade omvärlden*

### 4.2.4 Händelsehantering

En händelse sker vid en viss tidpunkt relativt spelstart. Det finns två typer av händelser:

1. Ena varianten är knuten till ett visst objekt eller en grupp av objekt. Exempel på denna typ av händelser är kurs- och fartändringar, slå av eller på vissa egenskaper eller sändare, m m. Denna typ av händelser hanteras på objektsnivå med hjälp av en händelselista i varje objekt.

2. Andra varianten är "globala" händelser, som t ex insättning och borttagning av objekt vid en viss tidpunkt, order som ska till EXS vid en viss tidpunkt, m m. Denna typ av händelser hanteras på spelnivå med hjälp av en global händelselista.

Modifikation, skapande och borttagning av objektshändelser och listor av objektshändelser är för tillfället knutet till det / de objekt som händelserna tillhör. Det har framförts önskemål om möjlighet att kunna kopiera en lista av händelser mellan objekt. Det finns också planer på att införa positionsbaserade händelser.

### 4.2.5 Tidshantering

All simulering sker relativt realtid. Uppdatering av objekt i omvärlden, uppdatering av skärm samt kontrollsändningar och tidssynkronisering sker med olika intervall givna av användaren, vid spelstart eller interaktivt. All tidshantering bygger på en speltid (currTime) som är reell tid multiplicerad med en faktor given av användaren vid spelstart eller då spelet är fruset. Om denna faktor är större än ett rekommenderas att sändningar till EXS och skärmuppdateringar upphör. Detta för att kunna uppnå samma noggrannhet i uppdateringarna som vid normal speltid.

### 4.2.6 ENS-SIM's gränssnitt mot ENS-NET och ENS-MMI

Bibliotek av funktioner för modifikation av den simulerade omvärlden och för spelhantering

**Accessorer**

- int mode()
  Returnerar spelets nuvarande tillstånd.
- void mode (int m)
  Sätter spelets tillstånd till m
- int origolat()
  Returnerar koordinatsystemets origos Latitud
- void origolat(int)
  Sätter koordinatsystemets origos Latitud
- int origolong()
  Returnerar koordinatsystemets origos Longitud

- void origolong(int)
  Sätter koordinatsystemets origos Longitud
- double timespeed()
  Returnerar spelets tidsfaktor
- void timespeed(double)
  Sätter spelets tidsfaktor
- double updatespeed()
  Returnerar spelets uppdateringsintervall
- void updatespeed(double)
  Sätter spelets uppdateringsintervall
- double sendspeed()
  Returnerar spelets sändintervall
- void sendspeed(double)
  Sätter spelets sändintervall
- double screenspeed()
  Returnerar spelets skärmuppdateringsintervall
- void screenspeed(double)
  Sätter spelets skärmuppdateringsintervall
- List* objectlist()
  Returnerar en lista med alla objekt i omgivningen
- List* globalevents()
  Returnerar en lista med alla globala händelser i spelet
- double starttime()
  Returnerar spelets starttid
- double currtime()
  Returnerar nuvarande speltid

**Funktioner**

- int save_game()
  Sparar spelets nuvarande tillstånd
- int load_scenario (char* filename)
  Slänger det gamla scenariot och läser in en scenariofil. Endast då spelet är stoppat.
- int run()
  Startar spelet
- void freeze()
  Pausar spelets uppdatering och sändning, men inte MMI
- void stop()
  Avslutar spelet
- int insert_object_event (int objectNr, EventPoint* event)
  Sätter in den givna händelsen i ett objekt med det givna objekt-nummret

- EventPoint* remove_object_event (int objectNr, EventPoint* event)
  Tar bort den givna händelsen ur ett objekt med det givna objektnummret
- int insert_object (Something* object)
  Sätter in det givna objektet i omvärlden
- Something* remove_object (int objectNr)
  Tar bort ett objekt med det givna objektnummret ur omvärlden
- int insert_global_event (GlobalEvent* event)
  Sätter in den givna globala händelsen i den globala händelselistan
- GlobalEvent* remove_global_event (GlobalEvent* event)
  Tar bort den givna globala händelsen ur den globala händelselistan

## 4.3 ENS-MMI

Användargränssnittet är beroende av plattformens grafikunderstöd och programmerarens intresse. Ett av mig fastslaget användargränssnitt skulle givetvis också begränsa utrymmet för utbyggnad och modifikation av systemet. Därför rekommenderar jag endast ett användargränssnitt som i stort liknar OBT vad gäller funktion och utseende, eftersom dess funktioner i många fall är likartade och många användare redan kan det gränssnittet.

Det finns heller inget som säger att MMI't måste ligga i ENS. Det kan lika gärna ligga på en annan maskin på SimulatorLANet och operera på ENS via Client-Server gränssnittet och hantera Order (se kapitlet *Meddelandetyper*) och filhantering utan att gå via ENS-SIM.

### 4.3.1 ENS-MMI's gränssnitt mot ENS-SIM

Följande funktioner för I/O mot användare anropas av tidshanteringen i ENS-SIM och måste således finnas, åtminstone som tomma funktioner, i ENS-MMI. Hur dessa funktioner fungerar och vilken funktionalitet som finns i user_input() är implementationsberoende.

- void update (Game* game)
  Uppdaterar MMI-representationen av alla objekt och händelser i omvärlden. Används vid spelstart och kan anropas från user_input närhelst användaren vill ha en korrekt lägesbild.
- void update (Element* object)
  Uppdaterar MMI-representationen av ett objekt eller en händelse i omvärlden. Används under spelkörning.
- int user_input (Game* game)
  Tar emot och utför ett kommando givet av användaren på det givna spelet.

# 5 Sändningstyper

I detta designförslag avhandlas huvudsakligen den kommunikation-som behövs mellan de olika noderna i ESS. Det finns olika typer av kommunikation, sändningstyper. En sändningtyp har ett visst användningsområde, en prioritet och använder en viss typ av adresser.

## 5.1 Unicast

Data som gäller en viss EXS-nod sänds med unicast. Unicast har näst högst prioritet vid sändning och mottagning. Vid sändning måste mottagarens adress ges. T ex Meddelandetypen Order sänds med unicast.

### 5.1.1 Adresser

Varje EXS-nod har en IP-adress och ett alias för denna adress. Varje Meddelandetyp är förknippad med en UDP-port (se kapitlet *Meddelandetyper*). Till denna port sänds endast denna typ av information.

## 5.2 Broadcast

Data som skall förmedlas till alla EXS, som tidssynkronisering, felmeddelanden som gäller hela ESS, m m sänds med broadcast. Broadcastmeddelanden har alltid högst prioritet vid sändning och mottagning. T ex Meddelandetypen Timesynch sänds med broadcast.

### 5.2.1 Adresser

INET_BROADCAST är den fördefinierade broadcastadressen i IP. Varje Meddelandetyp är förknippad med en UDP-port (se kapitlet *Meddelandetyper*). Till denna port sänds endast denna typ av information.

## 5.3 Multicast

Meddelandetyper som EXS kan prenumerera på, d v s är av intresse för en grupp av EXSer men ej alla har en egen multicastadress. EXS lyssnar på de adresser där intressanta data sänds. T ex så lyssnar en radarsimulator på de multicastadresser där ENS sänder Objectdata och Environmentdata.

Fördelen med Multicast är att data av en viss typ blir tillgänglig samtidigt för alla EXS:er som vill ha den så att inte de EXSer (t.ex. olika radar-

typer) som använder samma typ av indata har olika omvärldsbilder vid något tillfälle.

Samtidigt möjliggör Multicast att man kan flytta beslutet om mottagen information är intressant eller ej, till adressavläsningsnivå. D v s hårdvarunivå för de nätverkskort som kan hantera flera adresser annars drivernivå för de drivers som kan hantera flera adresser och i värsta fall i TCP/IP-kärnan. Alla dessa alternativ är dock att föredra mot att vara tvungen att sköta allt på applikationsnivå.

Förutsatt att inte alla EXS begär olika indata ger Multicast/Broadcast också god skalbarhet, då nätbelastningen inte ökar proportionellt med antalet anslutna EXS:er.

### 5.3.1 Prenumeration

Endast den information som någon EXS "prenumererat" på sänds över nätet. Varje klass av objekt är associerad med någon Meddelandetyp. När ett objekt av en viss klass uppdaterats eller ska kontrollsändas undersöks först om den Meddelandetyp som associeras med objektets klass har begärts av någon EXS.

Fördelen med detta är att ingen redundant information sänds (Information som inte används av någon EXS).

### 5.3.2 Adresser

Varje Meddelandetyp är förknippad med en multicastadress och en UDP-port (se kapitlet *Meddelandetyper*). På denna adress/port-kombination sänds endast denna typ av information.

## 5.4 Client-Server gränssnitt

Ett internt klientID används vid kommunikation mellan klienterna och servern. En mängd felkoder av UNIXmodell används. Skrivet i RPC/XDR. Se kapitlet *Meddelandetyper* för definition av sammansatta datatyper (typer som börjar med stor bokstav).

Dessa funktioner anropas i klienten i EXS-NET men utförs av servern i ENS-NET. Funktionerna arbetar direkt på ENS-SIM. Ett anrop av en av nedanstående funktioner kan ta godtyckligt lång tid, eftersom dessa meddelanden har lägst prioritet vid mottagning och sändning. Se kapitlet *Körningsexempel* för exempel på användning av nedanstående funktioner.

### 5.4.1 Run-time:

- int init_object (ClassID classID)
  Skapar ett nytt objekt med alla värden = default. Sätts inte in i omvärlden.
- int insert_object (int objectNr)
  Sätter in ett nytt objekt i omvärlden.
- int delete_object (int objectNr)
  Tar bort ett objekt ur omvärlden.
- int change_attr (int objectNr, FieldValue value)
  Ändrar ett objekts eller en händelses attributvärde. Om objekt så raderas dess händelselista.
- int disconnect (int clientID)
  Avregistrerar klienten hos servern. Hindrar dock inte EXS-NET från att lyssna på broadcast.
- int init_object_event (int objectNr, int type)
  Skapar en händelse till givet objekt med alla värden = objektets värden. Sätts inte in i objektets händelselista.
- int insert_object_event (int objectNr, int TPNr)
  Sätter in en händelse i ett objekts händelselista.
- int get_object_event (int objectNr, int place)
  Returnerar unikt ID för en händelse på den givna platsen i det givna objektets händelselista. place=0 returnerar nästa händelse.
- int delete_object_event (int objectNr, int TPNr)
  Tar bort en händelse på den givna platsen i det givna objektets händelselista.
- int report_error (int clientID, int errorCode, Timedata time)
  Logga klientID, felkod och tidpunkt för felet i ENS och ev. meddela användaren.

### 5.4.2 Start-up:

- int connect ()
  Registrerar klienten hos servern. Unikt klientId returneras.
- int subscribe (InfoType infoType)
  Klienten prenumererar på en viss informationstyp.
- int end_client_specification (int clientID)
  Klienten meddelar att den är redo att ta emot omvärldsinformation

# 6 Meddelandetyper

I ENS genererad omgivningsdata som skall överföras till EXS-noderna kan delas upp i olika typer av meddelanden beroende av innehåll. Varje enskild meddelandetyp hör ihop med vilken sändningstyp som används för överföringen av meddelandetypen. Meddelandetyperna indelas här efter sändningssätt och innehåll.

En meddelandetyp definieras av en sammansatt datatyp i XDR. XDR är plattformsoberoende och kan användas för beskrivning av -, och överföring av sammansatta datatyper på ett datornätverk.

Sun RPC innehåller ett program som heter rpcgen. Detta program läser XDR-kod och genererar kod i något målspråk (oftast C) för konvertering av datatyper mellan XDR och målspråkets datarepresentation på den dator på vilken målspråket är implementerat. Detta ger snabbt korrekt kod utan att kräva djupare kunskaper i målspråket och minneshanteringen på målmaskinen.

Meddelandetyperna i ESS definieras lämpligen i en separat fil med suffixet ".x" som måste finnas i både ENS- och EXS-noderna. Innehållet i meddelanden som överförs kan då relativt snabbt och enkelt modifieras genom att editera ".x"-filen, generera kod med rpcgen, och kompilera om NET-delen i ENS och den(de) berörd(a) EXS-noden(erna).

Nedan finns några exempel på Meddelandetyper baserade på innehållet i *Appendix*, avsnitt *Data i meddelanden*.

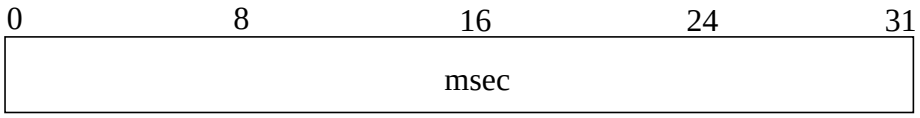
## 6.1 XDR-definition av Meddelandetyper, Broadcast

Dessa Meddelandetyper innehåller information som gäller alla EXS i ESS och är oftast också brådskande.

### 6.1.1 Timesynch

Innehåller antalet millisekunder från midnatt. Sänds med en viss, sättbar frekvens.

```
const TIME_ADDR = INADDR_BROADCAST;

const TIME_PORT = 401;

struct Timesynch

{

    int msec;

};
```

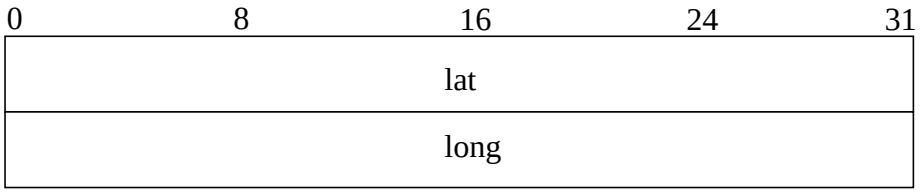



*Figur 10. Meddelandetypen Timesynch*

### 6.1.2 Origo

Innehåller koordinatsystemets origo i Latitud och Longitud.

```
const ORIGO_ADDR = INADDR_BROADCAST;

const ORIGO_PORT = 402;

struct Origo

{

    int lat;

    int long;

};
```




*Figur 11. Meddelandetypen Origo*

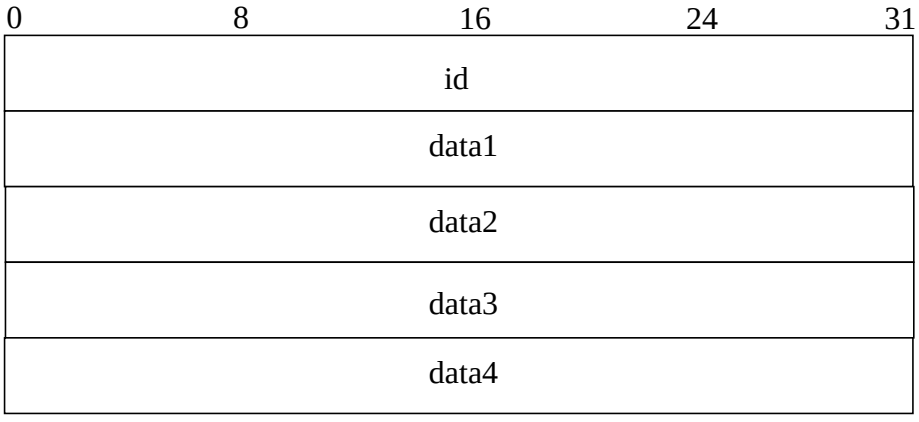
## 6.2 XDR-definition av Meddelandetyper, Unicast

Dessa datatyper innehåller information för styrning av en viss EXS's beteende, eller annan information som måste tolkas i EXS-SIM. En bakdörr för att tillåta central styrning av EXS, som kan vara användbar t ex vid testning av C3-systemets reaktion på olika indata från sensorer med likartade funktioner (olika radartyper eller sonartyper), en s.k. planerad icke-konsistent omvärldsbild. Adressen är sättbar för alla Unicast-Meddelandetyper.

### 6.2.1 Order

Detta är ett exempel på en Meddelandetyp bestående av fem fält: Ett ID-fält och fyra datafält. ID-fältet avkodas av EXS och den operation koden står för utförs. De fyra datafälten kan innehålla indata till operationen. Detta är en dålig lösning eftersom ID-fältet måste avkodas av applikationen innan man vet om man kan utföra operationen. En bättre lösning skulle kunna vara att ha en Meddelandetyp för varje operation.

```
string ORDER_ADDRESS[12];
const ORDER_PORT = 400;
struct Order
{
   int id;
   int data1;
   int data2;
   int data3;
   int data4;
};
```




*Figur 12. Meddelandetypen Order*

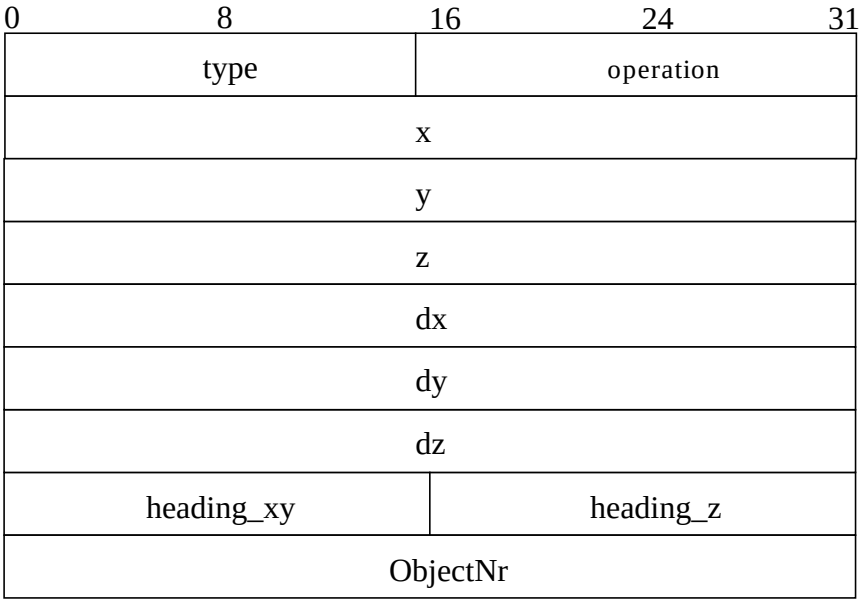
## 6.3 XDR-definition av Meddelandetyper, Multicast

Dessa datatyper innehåller information som berör en avgränsad del i omvärlden.

### 6.3.1 Objectdata

Ett objekt av en viss typ, med en position, en hastighet, en kurs och en unik identitet. En enkel bastyp som enkelt kan byggas ut till ett radarmål genom att lägga till radarreflektion, IFFkod m m eller till ett sonarmål genom att lägga till ljudstyrka och vad mer man önskar. Type innehåller information om objektets identitet och kategori, t ex fientligt flyg, jättekompis ubåt, m m. Operation anger vilken operation som skall utföras på objektet av EXS, t ex delete vid distribuerad uppdatering.

```
const OBJECT_ADDR = "224.0.1.10";
const OBJECT_PORT = 403;
struct Objectdata
{
   int typeNop;
   int x;
   int y;
   int z;
   int dx;
   int dy;
   int dz;
   int headingxyNheadingz;
   int objectNr;
};
```

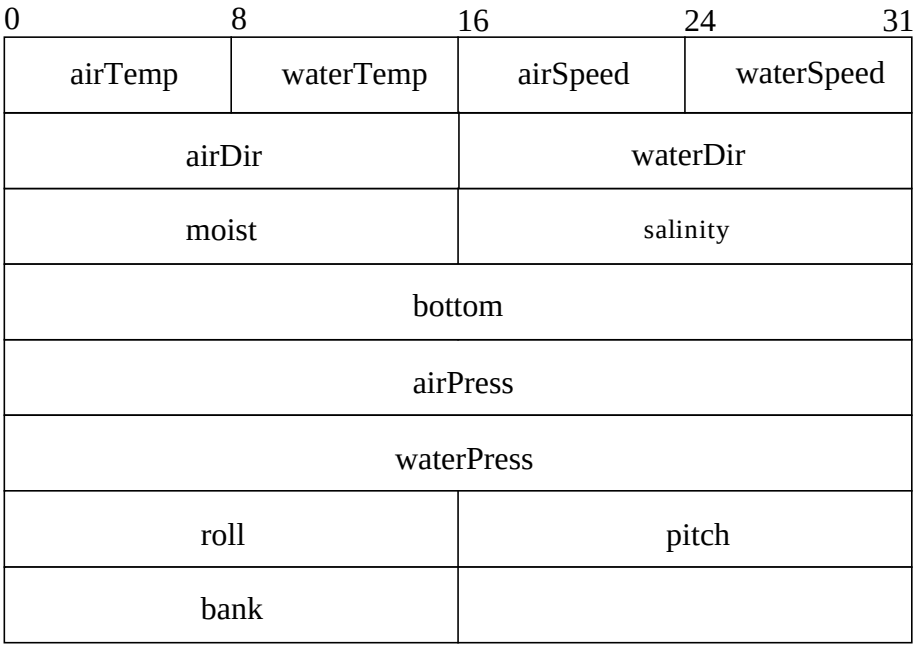



*Figur 13. Meddelandetypen Objectdata*

### 6.3.2 Environmentdata

Omgivningen till eget system innehåller en del variabler som kan avläsas med diverse sensorer. Moist är luftfuktigheten, som kan vara av intresse vid t ex ballistikberäkningar och sikt. Salinity, salthalten i vattnet, är huvudsakligen av intresse för ubåtssystem.Roll är maxvinkeln för roll-rörelsen utgående från bank-vinkeln. Pitch är maxvinkeln för pitch-rörelsen utgående från headZ för eget system. Dessa används huvudsakligen för fartygssystem.

```
const ENV_ADDR = "224.0.1.11";
const ENV_PORT = 404;
struct Environmentdata
{
   int tempsNspeeds;
   int dirs;
   int moistNsalinity;
   int bottom;
   int airPress;
   int waterPress;
   int rollNpitch;
   int bank;
};
```




*Figur 14. Meddelandetypen Environmentdata*

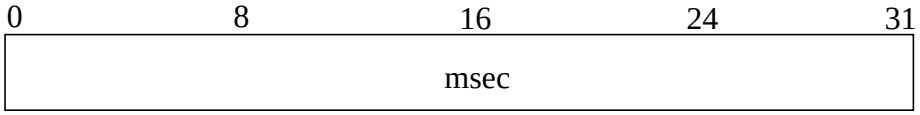
## 6.4 XDR-definition av sammansatta datatyper i Client-Server gränssnittet

I RPC löses typbestämningen av ett meddelande med hjälp av procedurmallar och unika procedurID som är tillgängliga för både sändare och mottagare. Meddelandetyperna är alltså inte associerade med någon viss IP-adress, utan med i vilken funktion de används.

### 6.4.1 Timesynch

Innehåller antalet millisekunder från midnatt. Används i samband med felrapportering från EXS till ENS.

```
struct Timesynch
{
   int msec;
};
```

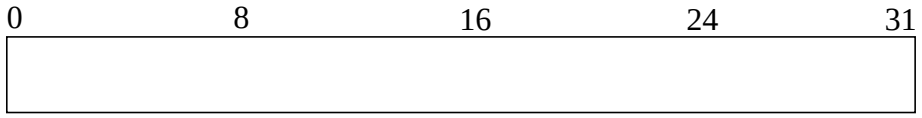



*Figur 15. Client-Server datatypen Timesynch*

### 6.4.2 ClassID

När en EXS vill skapa ett objekt i omvärlden, t ex en avfyrad missil med funktionen init_object(ClassID) måste man ange vilken klass objektet ska tillhöra. ClassID speglar alltså vilka slags objekt som kan skapas utifrån.

```
enum ClassID
{
   ELEMENT = 1,
   SOMETHING = 2,
   MOVING = 3,
   EVENTPOINT = 4,
   VEHICLE = 5
};
```

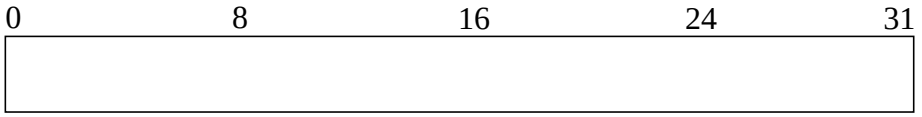



*Figur 16. Client-Server datatypen ClassID*

### 6.4.3 InfoType

När en EXS ska "prenumerera" på information från ENS med funktionen subscribe(InfoType) måste man ange vilken Meddelandetyp som önskas. InfoType speglar alltså vilka Meddelandetyper som man kan prenumerera på.

```
enum InfoType
{
   OBJECTDATA = 1,
   ENVIRONMENTDATA = 2,
   ORDER = 3,
   ORIGO = 4
};
```

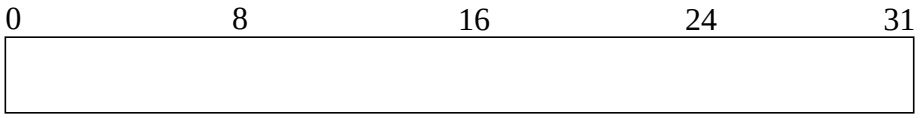



*Figur 17. Client-Server datatypen InfoType*

### 6.4.4 FieldName och FieldValue

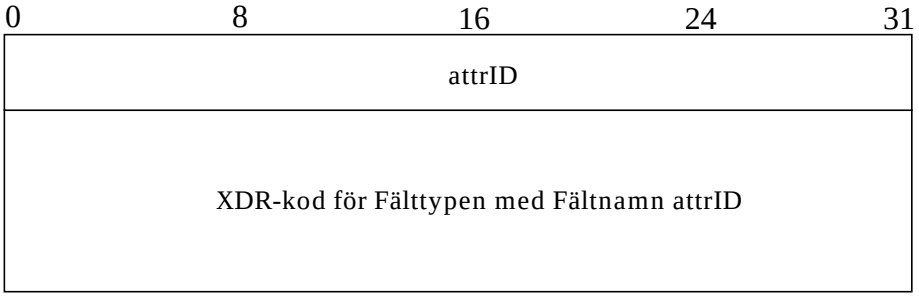
När man vill ändra ett värde i ett objekt i omvärlden, t ex styra sin nyss avskjutna missil mot målet, med funktionen change_attr (int, FieldValue) måste man först skapa ett FieldValue (se *FieldValue* nedan). Ett FieldValue består av två delar. Dels en kod (FieldName) som beskriver vad Meddelandetypen innehåller och dels själva innehållet(FieldValue).

```
enum FieldName
{
   TYPE = 1,
   SOUNDSTRENGTH = 2,
       :
   o.s.v för alla värden som kan ändras i objekt i omvärlden
       :
};
```




*Figur 18. Client-Server datatypen FieldName*

```
union FieldValue switch (FieldName attrID)
{
   case TYPE : int type;
   case SOUNDSTRENGTH : int soundstrength;
        :
   o.s.v för alla Fälttyper
        :
};
```




*Figur 19. Client-Server datatypen FieldValue*

# 7 Indataspecifikation

Indataspecifikationen ligger i EXS-NET, antingen som en separat fil som läses in då EXS-noden loggar in sig på ESS eller hårdkodat i ENS-NET's källkod. Det är upp till EXS-implementatören. Endast funktion och innehåll specificeras här.

## 7.1 Funktion

Används vid startup. Påverkar i vilken ordning en EXS's indata skall ligga i indataarrayen samt vilken information som EXS-programmeraren vill "prenumerera" på. Meddelandetypen Timesynch skickas alltid till alla EXS.

### 7.1.1 Innehåll

Önskat indata.(Se kapitlet *Meddelandetyper.*)

### 7.1.2 Exempel på Indataspecifikation:

Objectdata

Environmentdata

Origo

Order

# 8 Körningsexempel

Hypotetiska körningsexempel för att visa omgivningssimulatorsystemets funktioner. Felhantering utesluten. Timeout; Om mer än 5 timeout, meddela användaren.

## 8.1 Start-Up

1. ENS startar
   - Starta Portmap och RPC-server

2. EXS startar
   - Starta Portmap.

3. EXS byter till Remote mode
   - myID= connect ()
   - Läs av myID, om fel försök igen.

4. i ENS::connect ()
   - Registrera klienten.
   - Returnera ID.

5. i EXS: om EXS::connect() != fel
   - Så länge Indataspecifikationen inte är slut;
     1) status = subscribe (infoType)
     2) Läs av status, om fel försök igen.
   - status = end_client_specification (myID)
   - Läs av status, om fel försök igen
   - Skapa indataarray
   - Meddela användaren OK.

6. i ENS::subscribe(infoType)
   - Om den klient som meddelandet kommer ifrån inte är registrerad eller har rapporterat klar; returnera felkod.
   - Om Meddelandetypen ej beställts tidigare; möjliggör sändning av Meddelandetypen.
   - Återsänd resultat.

7. i ENS::end_client_specification(klientID)
   - Märk klienten som klar.
   - Om alla klienter är klara, meddela användaren att ESS klart för körning.

## 8.2 Avfyring av missil

1. i EXS
   - objectNr = init_object (MISSILE)
   - status = change_attr (objectNr, ownPos.x)
   - :
   - TPNr = init_object_event (objectNr, POSITION_BASED)
   - status = change_attr (TPNr, targetPos.x)
   - :
   - status = insert_object_event (objectNr, TPNr)
   - status = insert_object (objectNr)

2. i ENS::init_objekt (classID) där classID i detta exempel == MISSILE
   - Skapa ett objekt av klassen Missile.
   - Returnera objektets unika IDnummer.

3. i ENS::change_attr (objectNr, value)
   - Hitta objektet och anropa motsvarande medlemsfunktion.
   - Returnera resultatet.

4. i ENS::init_object_event (objectNr, type)
   - Hitta objektet och skapa en händelse till objektet med alla värden = objektets värden. Denna händelse sätts inte in i objektets händelselista.
   - Returnera unikt objektID för händelsen.

5. i ENS::insert_object_event (objectNr, TPNr, place)
   - Hitta händelse-objektet som skapats tidigare. (TPobj)
   - Hitta objektet som händelsen tillhör och anropa game::insert_object_event (ObjectNr, TPobj)

6. i ENS::insert_object (objectNr)
   - Lägg in objektet med givet ID i den simulerade världen.
   - Returnera resultatet.
   - Sänd objektet med multicast

7. i EXS
   - Jämför avstånd mellan målet och missilen för att se om inom "killing range".
   - Om inom "killing range", räkna ut skada på målet, radera missilen, och eventuellt simulera HPI hos målet och radera målet.
   - Om ej inom "killing range" styr missilen enligt produktspecifikt mönster m h a change_attr(...) eller händelser.

8. i annan EXS-nod
   - Eventuellt initiera siktesföljning.

## 8.3 Avfyring av pjäs

1. i EXS
   - Inriktning av pjäs, målföljning, laddning, avfyring, och träff / miss tas omhand i EXS.
   - Vid träff, räkna ut skada på målet och ev. simulera HPI och / eller radera målet.

## 8.4 Simulering av HPI (Hit Pattern Indikation)

1. i EXS
   - Räkna ut HPI m h a anslagsvinkel och målets hastighetsvektor.
   - Gör en plötslig kurs-och fartändring hos målet m h a change_attr (...)

# 9 Framtiden

Det finns mycket som behöver undersökas närmare och vidareutvecklas i designen / implementationen. För den händelse att någon skulle vilja fortsätta designprocessen från det tillstånd den nu befinner sig i, har jag sammanställt en lista över de brister och möjliga förbättringar som jag plågas av i nuläget.

1. Vidareutveckling av speltidsuppdatering relativt reell tid, som t ex bakåtspolning.
2. Djupare analys av följderna av införandet av flera egna system i omvärldssimuleringen och implementation.
3. Förbättra kopplingen mellan klasser av simulerade objekt och Meddelandetyper, så att införande av ny data i simuleringen blir enklare.
4. Djupare analys av följderna av distribuerad uppdatering, som t ex praktiska prestandautvärderingar, identifikation och lösning av problem som uppstår vid implementation, var distribuerad uppdatering bör hanteras i EXS-noden (Mitt förslag är i EXS-NET; flera ENS-SIM kan finnas i samma maskin men delar alltid samma omvärld).
5. Djupare analys av följderna av införande av uppdelningen av ENS's funktionalitet på flera noder, t ex vilken vidareutveckling av Client-Server gränssnittet som krävs, prestandautvärdering.
6. Förbättra gränssnittet mellan EXS-NET och EXS-SIM så att EXS-SIM blir friare.
7. Förbättra designen av ENS-MMI och förbättra kopplingen mellan MMI och klasser av simulerade objekt i omvärlden.
8. Förbättra filhantering i ENS. Optimera spelfiler, undersöka manuell editering av spelfiler.
9. Indataspecifikationens vara eller icke vara. Om samma information skall hårdkodas så bör den vara samlad på en specifik plats.
10. Implementation av EXS-NET och Client-Server gränssnittet, tidssynkronisering, m m.
11. Order. Behövs de? Jag lade in Order som ett wildcard för att skydda mot sista-minuten-krav och oförutsedda problem.
12. Hur hantera flera EXS-SIM i samma EXS-nod? t ex Order (Unicast) sänds till en maskinspecifik adress, hur hantera flera indataspecifikationer?
13. Hur hantera olika prioritet på sändningstyper?

# Litteratur

Litteratur som ligger till grund för detta dokument.

**Kurslitteratur**

- STA94 Data and Computer Communications, 4:th ed.; Stallings, William; Prentice Hall, 1994.
- TAN92 Modern Operating Systems; Tannenbaum, Andrew, S; Prentice Hall, 1992.

**Handböcker**

- MAM89 LAN Troubleshooting Handbook; Miller, Mark A; M&T Publishing Inc, 1989.
- Technical Handbooks; Bofors Electronics, 1990.

**CelsiusTech dokument**

- Simulatorplattformen SIMPL Beskrivning; DIP-AP2 S-91:214 Rev A, CelsiusTech,1992.
- Design Specification SSIM; PM 217 552, CelsiusTech,1995.
- Sensorsimulator A19; PM 194 647, CelsiusTech, 1995.
- Users Guide for Target Simulator TASI; SFUG-100 560, CelsiusTech, 1993.
- Product Specification SS2000 On Board Training; Ritzen, Benny; PM 183 366, CelsiusTech, 1994.
- 9LV200 Mk2 / 2.5 Simulator System Description; PM 172 372, CelsiusTech, 1993.
- 9LV200 Mk2 / 2.5 Simulator Interface Specification; PM 172 371, CelsiusTech, 1993.
- Exercising using PC-simulators in IMPACT; PM 257 780, CelsiusTech, 1995.
- Omgivningssimulator, Kravspecifikation; PMxxxxxx, Bofors Electronics AB, 1989

**Internet, IP m m**

- Omnicom Coursebook TCP / IP; Omnicom PPI Ltd, 1992 + 6 st videoband.
- HCL88 Introduction to Administration of an Internet-based Local Network; Hedrick, Charles, L; Rutgers University, 1988.
- RFC 791 Internet Protocol 1981
- RFC 768 User Datagram Protocol 1980
- RFC 793 Transmission Control Protocol 1981
- RFC 1340 Assigned Numbers 1992
- RFC 1014 eXternal Data Representation 1987
- RFC 1057 Remote Procedure Call 1988
- RFC 1001 Protocol Standard for a NetBIOS Service on a TCP / UDP Transport: Concepts and methods 1987

- RFC 1002 Protocol Standard for a NetBIOS Service on a TCP / UDP Transport: Detailed Specifications 1987
- IEN-212 IP-LAN Addressing Issues 1982
- RFC 919 Broadcasting Internet Datagrams 1984
- RFC 1122 Requirements for Internet Hosts -- Communication Layers 1989
- RFC 791 Internet Addressing
- RFC 1112 Host Extensions for IP Multicasting 1989
- RFC 1301 Multicast Transport Protocol 1992

**Distribuerad Simulering**

- Protocols for Distributed Interactive Simulation Applications; IEEE Std 1278-1993, IEEE Computer Society, 1993.

**Dokumentation: Inköpt programvara och hårdvara**

- Distinct TCP / IP for Windows + Run Time Installation and Configuration Guide; Distinct Corporation, 1995.
- PC / TCP Network Software and PC / TCP OnNet Developer's Toolkit 3.1 for DOS and Windows + API's; FTP Software Inc., 1995.
- EtherLink III Parallell Tasking 16-Bit ISA and 32-Bit EISA Adapters; 3Com Corporation, 1993.

## Bibliografi

Litteratur som rekommenderats mig men som jag inte läst själv.


- Distributed Computing and Client-Server Systems; Amjad, Umar; Prentice Hall, 1993.
- Human-Computer-Interaction; Preece, Jenny; Addison Wesley, The Open University, 1994.
- Client / Server LAN Programming; Nance, Barry; Que Computer Publishing Ltd, 1994.
- Novells Guide to Implementing Client / Server Applications; Becker, Eric, B; Sybex US, 1995.
- Power Programing with RPC; Bloomer, John; O'Reilly & Assoc. Inc., 1991. ISBN: 0-937175-77-3
- Computer Networking for Systems Programmers; Cole, Gerald D; John Wiley and sons, 1990. ISBN: 0-471-51057-2
- Internetworking with TCP / IP Volume I; Comer, Douglas; Prentice Hall, 1991. ISBN: 0-13-468505-9
- Internetworking with TCP / IP Volume II; Comer, Douglas and Stevens, David L; Prentice Hall, 1991. ISBN: 0-13-472242-6
- Internetworking with TCP / IP Volume III; Comer, Douglas and Stevens, David L; Prentice Hall, 1993. ISBN: 0-13-474222-2
- Programmers Guide to NetBIOS; Schwaderer, David W; Howard W Sams & Co, 1988. ISBN: 0-672-22638-3
- UNIX Network Programming; Stevens, W Richard; Prentice Hall, 1990. ISBN: 0-13-949876-1

# Appendix

## 1 Terminologi

### 1.1 ARP

Står för "Address Resolution Protocol" och är ett protokoll för översättning av IP-adresser till LAN-adresser.

### 1.2 Broadcast

Sändning till alla noder på ett nät. På LAN av ethernet-typ (IEEE 802.3 el.dyl) sker all kommunikation med broadcast, men alla noder lyssnar inte på alla adresser. På andra typer av LAN (token-ring, FDDI m fl) kan broadcast emuleras.[RFC 1122]

### 1.3 C3-system

Står för "Command, Control and Communication system". Ett datorsystem vars omgivning ESS ska simulera.

### 1.4 Client-Server

En kommunikationsmodell som bygger på att man har en "Server" som hanterar förfrågningar från ett flertal klienter. Klienten frågar och Servern svarar. Den enkla modellen möjliggör ett enkelt och effektivt kommunikationsprotokoll och data och funktionalitet kan delas upp i oberoende moduler vilket underlättar modifikation och felrätttning.[TAN92]

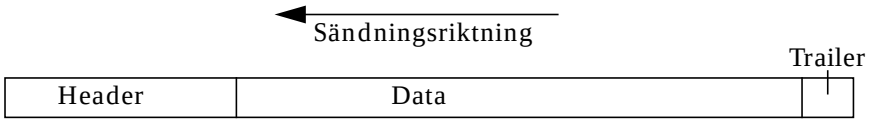
## 1.5 Datagram, Paket (Packet), Ram (Frame)

En logisk enhet av data för överföring via något media bestående av en "header"-del, en datadel och ibland en "trailer"-del.

Header-delen innehåller information som behövs för överföringen. T ex en IP-header innehåller avsändar- och mottagaradress, information som används vid fragmentering av meddelanden, vilket Transport-protokoll som skickat meddelandet m m.

Datadelen innehåller den information som användaren vill överföra.

Trailer-delen innehåller ytterligare information som används vid överföringen av datagrammet. [TAN92, STA94]




## 1.6 ENS-MMI

Hanterar användargränssnittet till en ENS-nod.

## 1.7 ENS-NET

Nätkommunikationsdelen i en ENS-nod.

## 1.8 ENS-Node

Står för "ENvironment-Simulator-Node". Den (De) nod(er) på SimulatorLANet som innehåller ENS-SIM och nätadministrationen. (Se ENS-specifikation).

## 1.9 ENS-SIM

Simulatordelen av en ENS-nod.

## 1.10 ESS

Står för "Environment-Simulator-System". Definieras som ett SimulatorLAN, en eller flera ENS-Noder, ett godtyckligt antal EXS-Noder samt diverse utrustningsspecifika kopplingar till SIU.

## 1.11 EXS-MMI

Hanterar användargränssnittet till en EXS-nod.

## 1.12 EXS-NET

Nätkommunikationsdelen i en EXS-nod.

## 1.13 EXS-Node

Står för "EXternal equipment-Simulator-Node". Den nod på Simulator-LANet som simulerar en viss extern utrustning tillhörande ett visst C3-system. Kan innehålla flera EXS-SIM. (Se EXS-specifikation).

## 1.14 EXS-SIM

Simulatordelen av en EXS-nod.

## 1.15 EXS-SIU

Hanterar den utrustningsspecifika kommunikationen med SIU

## 1.16 Header

Den del av ett datagram som sänds först ut på nätet. Innehåller information som behövs för överföringen av datagrammet, såsom destinationsadress m m.

## 1.17 ICMP

Står för "Internet Control Message Protocol" och är en del av IP. Tar hand om felhantering och routehantering.(Connection error, No destination machine, Timeout, Clogged gateway, Find shortest route). [RFC 791]

## 1.18 Internet

Ett kommunikationsnät som innefattar ett flertal WAN och LAN.

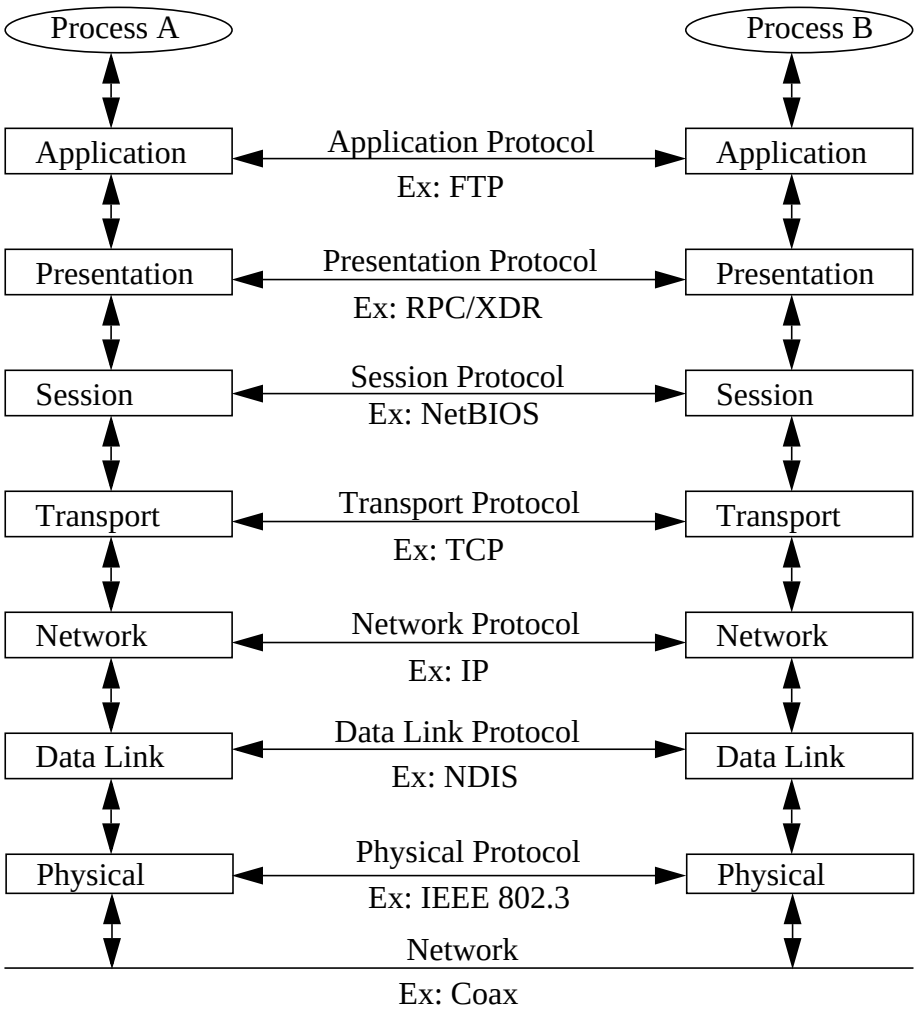
## 1.19 IP

Står för "Internet Protocol" och är ett protokoll för att sända datagram över flera ihopkopplade datornätverk, d v s datagrammet kan plockas isär (fragmenteras) i mindre delar för överföring över nät med mindre paketstorlek och sedan sättas ihop igen. Innehåller ingen kontroll av att datagrammet kom fram eller att mottagaren ens existerar. Mängden data som kan överföras i ett datagram begränsas av nätets MTU. [RFC 791]

## 1.20 ISO/OSI-modellen

(International Standards Organization / Open Systems Interconnection reference model)

En ISO-standard för beskrivning av datorkommunikationssystem genom definition av en "stack" av skikt där varje skikt karakteriseras av den funktion i kommunikationen som skiktet utför. [TAN92, STA94]




*Figur 20. ISO/OSI-modellen*

## 1.21 LAN

Står för "Local Area Network" och är ett kommunikationsnät som innefattar en eller ett fåtal byggnader, eller en maximal sträckning i hundratal meter.

## 1.22 MMI

Står för "Man Machine Interface" och är en applikations användargränssnitt.

## 1.23 MTU

Står för "Maximum Transmission Unit" och är den största mängd data som kan överföras i ett enda datagram på ett visst nätverk.[STA94]

## 1.24 Multicast

Sändning till en adress som kan delas av flera noder. Se RFC 1340 Assigned numbers, Multicast addresses för att se vilka multicastadresser som finns och vilka som används. [RFC 1112, RFC 1122, RFC 1340]

## 1.25 OBT

Står för "On Board Training" och är en simuleringsfunktion som är inbyggd i C3-systemet avsedd för träning av operatörer.

## 1.26 Protokoll

En samling regler för kommunikation.

## 1.27 Protokollstack, TCP/IP-stack

En grupp protokoll som beskriver gränssnitten mellan de olika lagren i ISO / OSI-modellen.[TAN92, STA94]

## 1.28 RARP

Står för "Reverse ARP" och är ett protokoll för översättning av LAN-adresser till IP-adresser.

## 1.29 RPC

Står för "Remote Procedure Call" och är ett protokoll för anrop av procedurer och program som exekveras på en annan fysisk maskin, med i stort sett samma syntax som för Local Procedure Call (LPC). [RFC 1057]

## 1.30 SIU

Står för "Standard Interface Unit".

## 1.31 TCP

Står för "Transmission Control Protocol" och är byggt på IP och ICMP. TCP upprättar en pålitlig förbindelselänk mellan två processer. TCP garanterar att överförd information kommer fram inom en viss tid, och om det misslyckas meddelas användaren. TCP är dock ganska omständligt. Det sänds tre meddelanden vid varje uppkoppling och nedkoppling av en förbindelse och dess "header" innehåller information för route- och sekvenshantering på paketnivå, checksumma över meddelandets innehåll, meddelandets prioritet m m. TCP understödjer inte IP's broadcast eller multicast. För att få sådan funktionalitet måste applikationen upprätta en separat TCP-förbindelse till var och en av broadcast / multicast-medlemmarna i sekvens. TCP används med fördel för överföring av större datamängder över Internet och för de fall då det är viktigare att informationen kommer fram än att ha hög överföringshastighet. [RFC 793]

## 1.32 TSR

Står för "Terminate and Stay Resident" och är ett program som terminerats med MSDOS-systemanropet KEEP_PROG. Programmet ligger kvar i minnet och dess procedurer kan anropas från andra program som t ex interrupthanterare. Om dess procedurer anropas av MSDOS eller om MSDOS någonstans i anropskedjan som leder till att en procedur i ett TSR-program anropats, så kan TSR-programmet i sin tur inte använda systemanrop. MSDOS kan bara hantera ett systemanrop i taget. [TAN92]

## 1.33 UDP

Står för "User Datagram Protocol" och är i stort sett IP plus en checksumma över innehållet i datagrammet. Används ofta i Multicast / Broadcast-sammanhang och kommunikation över LAN där risken för att tappa bort paket är mycket liten och där ingen routing-relaterad funktionalitet behövs. [RFC 768]

## 1.34 UTC

Står för "Universal Time Coordinated" och motsvarar Greenwinch Mean Time (GMT).

## 1.35 WAN

Står för "Wide Area Network" och är ett kommunikationsnät som innefattar ett flertal LAN eller ett flertal maskiner utspridda över ett stort område.

## 1.36 XDR

Står för "eXternal Data Representation" och är ett protokoll för maskinoberoende datarepresentation. Protokollet hanterar konvertering av byteordning, alignment, kodning och avkodning av data för transport över nät m m.[RFC 1014]

# 2 Distribuerad uppdatering

## 2.1 Varianter

1. Skicka endast ändringar och tidpunkt för ändringar i ett meddelande till EXS, som sedan tar hand om uppdatering av objekten fram till det är dags för nästa ändring

   x = En godtycklig tidssynkronisering.

   Händelser i tidsintervallet ] x-1, x] rapporteras under tidsintervallet ] x-2, x-1].

   Uppdatering av objekt sker både i EXS och ENS, fast med olika uppdateringshastigheter.

   Hantering av händelselistor och tidssynkronisering. hanteras i ENS.

2. Alla händelser överförs innan spelstart tillsammans med alla objekt.

   Om alla händelser överförs innan spelstart har vi överfört allt arbete på EXS. ENS sköter då dynamiska förändringar i omvärlden efter spelstart, samt tids- och egenskapssynkronisering. ENS's tid och omvärldsbild är prioriterad gentemot EXSerna.

   Hur sköter vi i detta fall nyintroducerade objekt med tillhörande händelselista? Ska alltihop överföras på en gång?

3. När en händelse sker på ett objekt sänds dess nya status till EXS.

   Uppdatering av objekt sker både i EXS och ENS, fast med olika uppdateringshastigheter. Hantering av händelselistor och tidssynkronisering hanteras i ENS.

   I EXS uppdateras objekten efter någon extrapolationsalgoritm mellan statussändningar. Detta innebär att om ESS innehåller en EXS som kräver hög upplösning och samtidigt inte accepterar alltför stora "hopp" vid statussändningar (T.ex en målföljningsradar i ett sikte) krävs en hög frekvens av statussändningar.

## 2.2 Fördelar med distribuerad uppdatering

- Minskad nätbelastning
- Noggrannare tidsupplösning i EXS
- Minskad belastning på EXS för näthantering

## 2.3 Nackdelar

- Förlust av paket kan resultera i olika lägesbilder under resten av ett scenario. Lösning: sänd extra
- Olika tidsupplösning kan resultera i olika avrundningsfel och lägesbilder som går isär med tiden. Lösning: Statussynkronisering. från ENS med olika tidsintervall.
- Om många objekt ska ändras samtidigt kan ESS bli överbelastat och vissa ändringar hamnar på efterkälken. Hur långt i förväg ska ändringar överföras?
- Hur hantera multipla förändringar på ett och samma objekt samtidigt? Ytterligare lagring av och hantering av händelser i EXS eller använd endast ett objekt innehållande alla förändringar vid en viss tidpunkt.
- Hur hantera borttagning av objekt i ett distribuerat system? Ett borttagningsmeddelande som kräver ack måste införas.

## 2.4 Diskussion

För en RADAR gäller

vinst_per_paket = (Tidsåtgång_beräkning_ny_hast.&pos * #objekt_i_paket) / (Tidsåtgång_näthantering / paket)

vinst = (Max_#_objekt / #objekt_i_paket) * vinst_per_paket

events = (Totalt_#_events / intervall_mellan_uppdateringar) * (#extra_sändningar + 1) * (Tidsåtgång_näthantering / paket)

sync = (Max_#_objekt / #objekt_ i_broadcastpaket) * (Tidsåtgång_näthantering / broadcastpaket) *

(#positionssync. / intervall_mellan_uppdateringar)

Total_vinst_på_en_världsuppdatering = vinst - events - sync

Är detta > 0 så tjänar man på det

## 2.5 Max nätlast för olika distributionsmodeller

Det som avgör "till syvende og sidst" är hur mycket data som kan överföras på SimulatorLANet. Begränsande faktorer är då bandbredd och anslutna EXS-noders mottagningskapacitet (som i sin tur beror på hårdvara och mjukvara. Kan alltså bara bestämmas empiriskt). Nedan används de Meddelandetyper som beskrivits i Designförslaget.

1) **All simulering i ENS**.

   Statusinformation om varje objekt i omvärlden ska skickas över nätet till EXS där det skall lagras. Då blir det som mest

   1 Objectdata*#objekt + 1 Environmentdata*#egna + 1 Timesynch + 1 Order + 1 Origo =

   288*#objekt + 256*#egna + 32 + (32 + #data) + 64 = 288*#objekt + 256*#egna + (32 + #data) + 86 bitar per uppdatering. Client-Server och Headers ej inräknat.

   Sätt #objekt = 400, #egna system = 1, #data = 4, #uppdateringar / sec = 11

   så får vi en nättrafik i storleksordningen 1.2Mbits / sec. Client-Server och Headers ej inräknat. Nätlasten i genomsnitt blir dock något lägre eftersom Environmentdata, Timesynch, Origo och Order sänds mer sällan.

   Bandbredden ligger på 10Mbits / sec så det kan fungera. Begränsande faktor blir då ENS's prestanda. Om vi lyfter ut det grafiska användargränssnittet på en fristående maskin som använder client-server gränssnittet för att kommunicera med ENS så avlastar det en hel del. (Detta koncept används i SIMPL.)

   En annan variant är att ha flera ENS som hanterar var sin del av multicastgrupperna.

2) **Uppdatering av object m a p tid i EXS**.

   Enligt variant 3 ovan.

   Händelsehantering, felhantering, tidssynkronisering,filhantering och Client-Server-gränssnitt i ENS. Tillkommer objektstatussynkronisering, meddelanden för införande av objekt, modifiering av objekt samt borttagning av objekt i EXS.

   Antag högst en ändring av objektsstatus per tidscykel och objekt. Då blir det som mest

   1 (Objectdata | Environmentdata)*#statussynk + 1 Timesynch*#timesynch + 1 Order*#order + 1 Objectdata*#objekt bitar per tidscykel.

   Sätt (Objectdata | Environmentdata) = 270, #statussynk = 10, #order = 1, #objekt = 400, #tids-cykler / sekund = 1, #övrigt = 4, #timesynch = 0.1

   så får vi en nättrafik i storleksordningen
   270*10+32*0.1+(32+4)*1+288*400 = 118 k bits / sec. Client-Server och Headers ej inräknat.

Bandbredden ligger på 10Mbits / sec. Begränsande faktor blir då EXS prestanda.

I genomsnitt blir dock nätlasten betydligt mindre. Att alla objekt skulle ändras samtidigt är mycket osannolikt.

3) **All händelsehantering sker i EXS**.

Objekt i världen vid spelstart samt deras händelselistor skickas över till EXS från ENS före spelstart. Tillkommer objektstatussynkronisering, meddelanden för införande av objekt, modifiering av objekt samt borttagning av objekt i EXS. Blir samma värsta fall som för 2) ovan, men i genomsnitt lägre nätlast än 2). Beräkningslast på EXS blir dock ännu större.

# 3 Omgivningssimulatorer

Nedan följer en kort presentation av de olika omgivningssimulatorer jag tittat på, kommentarer och vilka delar som jag återanvänt i designen.

## 3.1 Target Simulator (TASI)

Skriven i Ada. I stort sett ett förenklat bassystem i D85-miljö. Textbaserat användargränssnitt. Jag har återanvänt innehåll i meddelanden och behovet av en kommandokanal. Behovet av ett grafiskt gränssnitt insåg jag snabbt efter att ha försökt att sätta mig in i TASI's språksyntax.

## 3.2 Sensor Simulator (SSIM)

Inte riktigt en omgivningssimulator eftersom den endast kan simulera en del av omgivningen till systemet i taget. Skriven i C. PC-baserad. Använder seriell kommunikation. Omgivnings- och utrustningssimulatorer i samma maskin. Tidshantering, djup och sonar gav många ideer. Uppdelning av programmet i moduler hjälpte en del. Det är bara synd att inte modulerna återspeglar var saker och ting utförs i koden.

## 3.3 SimulatorPlattform (SIMPL)

Kallas också för TASS, FANCY,... . Används i många projekt i någon form. Delvis skriven i Ada, delvis i C. Tillåter användaren att definiera sitt eget beskrivningsspråk för uppbyggnad av sin simulator. Har ett grafiskt gränssnitt på Sun-versionen, tyvärr ganska buggigt. Också tyvärr är det f n. ingen som kan koden och det var svårt att få tag på dokumentation. Jag har försökt förenkla den något virriga objektdesignen i SIMPL och återanvänt den i ENS.

## 3.4 IMPACT Simulatorer

Skrivna i Turbo Pascal 7.0. PC-baserad. Använder Ethernet och TCP / IP för kommunikation mellan omgivningssimulator och utrustningssimulatorer. Beskrivningen av EXS i denna rapport baseras i stort på dess utrustningssimulatorer.

## 3.5 On Board Training (OBT)

Tränings- och utbildningssimulator som ingår i fartygssystemet. Skrivet i Ada. Ideer om att flytta ut detta system av simulatorer på ett omgivande simulator-LAN har förekommit. Dock innehåller det mycket direktkommunikation med användagränssnitt i utrustningssimulatorerna och innehåller vissa slumpfaktorer. Huvudproblemet med dessa ideertror jag personligen ligger i att OBT är avsett för utbildning, inte för testning. Dock har OBTs simulering av målbanor m m varit en källa till inspiration vid design av ENS. Önskemål om likartat användargränssnitt för ENS har också mottagits.

## 3.6 Omgivningssimulator till fartygssystem

Ett förslag till omgivningssimulator till fartygssystem som fastnade på planeringsstadiet. Skulle ha vidareutvecklats från specifikt för ett visst fartygssystem till ett allmänt omgivningssimulatorsystem för fartygssystem.

# 4 Data i meddelanden

En sammanfattning av innehållet i de meddelanden som används i existerande fartygs- och landbaserade omgivningssimulatorsystem som jag tittat på. Namn på variabler, enheter och defaultvärden är olika i olika system. De som här används är de namn, enheter och defaultvärden som används i ENS.

Nedanstående data är uppbyggda enligt följande mall:

<Beskrivande namn>; <enhet>. (<variabelnamn>) [Default; <defaultvärde>]

## 4.1 Måldata

- Lat; radianer*0.01. (x) [Default; 0]
- Long; radianer*0.01. (y) [Default; 0]
- Höjd / Djup; meter. (z) [Default; 0]
- Hastighet i x-led; meter / sekund. (dx) [Default; 0]
- Hastighet i y-led; meter / sekund. (dy) [Default; 0]
- Hastighet i z-led; meter / sekund. (dz) [Default; 0]
- Heading i xy-led; grader (headingxy) [Default; 0]
- Heading i z-led; grader (headingz) [Default; 0]
- Typ; heltal. (type) [Default; UNKNOWN]
- Objektnummer; heltal. (ObjectNr)
- Radar Cross Section; $m^2$ * 0.01. (radarReflection) [Default; 0]
- Ljudstyrka; mB. (soundStrength) [Default; 0]

## 4.2 Eget skeppsdata

- Lat; radianer*0.01. (x) [Default; 0]
- Long; radianer*0.01. (y) [Default; 0]
- Höjd / Djup; meter. (z) [Default; 0]
- Hastighet i x-led; meter / sekund. (dx) [Default; 0]
- Hastighet i y-led; meter / sekund. (dy) [Default; 0]
- Hastighet i z-led; meter / sekund. (dz) [Default; 0]
- Heading i xy-led; grader (headingxy) [Default; 0]
- Heading i z-led; grader (headingz) [Default; 0]
- Roll; grader (roll). [Default; 0]
- Pitch; grader (pitch). [Default; 0]
- Bank angle; grader (bank). [Default; 0]

## 4.3 Omgivningsdata

- Lufttemperatur; grader C. (airTemp) [Default; 20]
- Luftfuktighet; procent (moist) [Default; 30]
- Lufttryck; mPa (airPress) [Default; 101320]
- Vindriktning; grader. (airDir) [Default; 0]
- Vindhastighet; meter / sekund. (airSpeed) [Default; 0]
- Vattentemperatur; grader C. (waterTemp) [Default; 20]
- Vattentryck; mPa (waterPress) [Default; 101320]
- Havsström riktning; grader. (waterDir) [Default; 0]
- Havsström hastighet; meter / sekund. (waterSpeed) [Default 0]
- Bottendjup; meter. (bottom) [Default; 50]
- Salthalt; procent. (salinity) [Default; 0]
- Origo, Lat; radianer*0.01. (origoLat) [Default; 0]
- Origo, Long; radianer*0.01. (origoLong) [Default; 0]

## 4.4 Tidssynkronisering

- Millisekunder från midnatt, Lokal maskintid; heltal. (msec)

## 4.5 Utrustningsstatus

- Available; boolean. (avail) [Default; true]