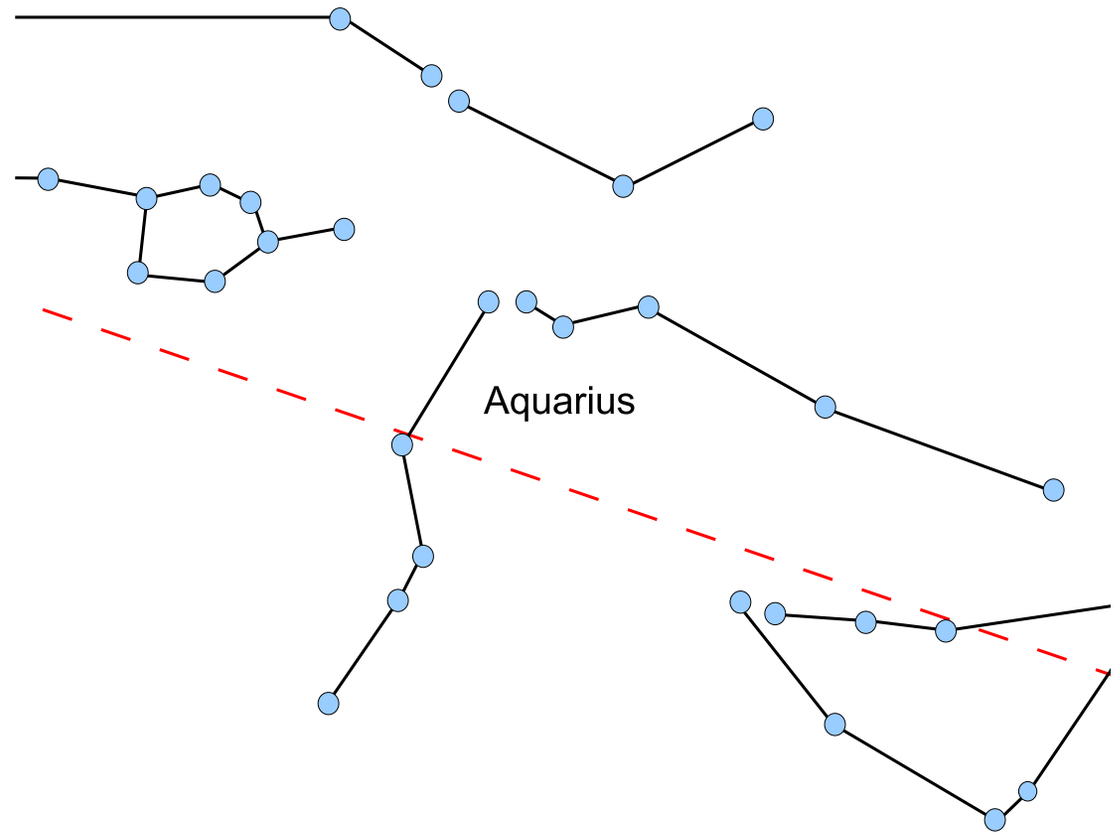
Aquarius, en simuleringsmiljö för ad hoc-nät

– Strategidokument

ULF STERNER



FOI är en huvudsakligen uppdragsfinansierad myndighet under Försvarsdepartementet. Kärnverksamheten är forskning, metod- och teknikutveckling till nytta för försvar och säkerhet. Organisationen har cirka 1000 anställda varav ungefär 800 är forskare. Detta gör organisationen till Sveriges största forskningsinstitut. FOI ger kunderna tillgång till ledande expertis inom ett stort antal tillämpningsområden såsom säkerhetspolitiska studier och analyser inom försvar och säkerhet, bedömning av olika typer av hot, system för ledning och hantering av kriser, skydd mot och hantering av farliga ämnen, IT-säkerhet och nya sensorers möjligheter.

FOI
Totalförsvarets forskningsinstitut
Ledningssystem
Box 1165
581 11 Linköping

Tel: 013-37 80 00
Fax: 013-37 81 00

www.foi.se

FOI-R--2776--SE
ISSN 1650-1942

Metodrapport
Juni 2009

Ledningssystem

Ulf Sterner

Aquarius, en simulerings-miljö för ad hoc-nät

– Strategidokument

Titel	Aquarius, en simuleringsmiljö för ad hoc-nät – Strategidokument
Title	Aquarius, a simulation tool for ad hoc networks – Strategy document
Rapportnr / Report No.	FOI-R–2776–SE
Rapporttyp	Metodrapport
Report Type	Methodology Report
Månad / Month	Juni / June
Utgivningsår / Year	2009
Antal sidor / Pages	31
ISSN	1650-1942
Kund / Customer	FM
Kompetenskloss	22 Samband och Telekom
Projektnr / Project No.	E53057
Godkänd av / Approved by	Anders Hansson

FOI, Totalförsvarets Forskningsinstitut
Avdelningeng för Ledningssystem
Box 1165
581 11 LINKÖPING

FOI, Swedish Defence Research Agency
Command and Control Systems
P.O. Box 1165
SE-581 11 LINKÖPING

Sammanfattning

Modellering och simulering är idag viktiga verktyg vid forskning av utveckling av kommunikationssystem. I denna rapport analyserar vi möjliga utvecklingslinjer för den egenutvecklade simuleringsmiljön Aquarius.
Tre olika utvecklingslinjer som analyseras är, produktifiering, vidmakthållande och avveckling. Olika för och nackdelar såsom ekonomiska aspekterna, externa samarbeten, utveckling av modeller, hanteringen av ramverket och simuleringskapacitet analyseras i rapporten för de olika utvecklingslinjerna.

Nyckelord: Aquarius, simuleringsmiljö, kommunikationssystem, ad hoc-nät

Abstract

Modeling and simulation is today important tool in research and development of communication systems. In this report we analyze strategies for Aquarius, an in house simulation tool for communication system.
Three different strategies are analyzed in the report, making a product of the simulation tool, maintain the simulation tool, and phasing out the simulation tool. Different pros and cons as economical aspects, external collaboration, development of models, maintenance of the framework, and simulation capacity are analyzed for the different strategies.

Keywords: Aquarius, simulation tool, communication system, ad hoc network

Innehållsförteckning

1 Inledning

Inom forskning och utveckling av kommunikationssystem är det idag vanligt att använda modellering och simulering (MoS) som stöd för att utveckla och evaluera olika systemlösningar. Möjliga användningsområden för MoS sträcker sig här från att skapa en djupare förståelse om enskilda algoritmer till övergripande utvärderingar av hela system. För att på ett kostnadseffektivt sätt genomföra modellering och simulering behövs ett simuleringsramverk med tillhörande modellbibliotek så att existerande modeller kan återanvändas.

Vilken typ av simuleringsmiljö som är lämplig att använda beror på vilken typ av frågor som ska besvaras. Ska det fysiska lagret i ett kommunikationssystem utvärderas används andra typer av modeller än om ett routingprotokoll på nätlagret ska utvärderas. För den första typen av simuleringar behövs en modell där systemet modelleras på sampelnivå. För den andra används normalt en modell med högre abstraktionsnivå som modellerar systemet på paketnivå. För att höja abstraktionsnivån i modellen används förenklade modeller av vissa delsystem. Detta leder visserligen till lite större modelleringsfel men samtidigt blir det möjligt att studera exempelvis skalbarhetsfrågor utan att simulerings tiderna blir orimliga.

Målsättningen med denna rapport är att analysera möjliga framtidsstrategier för den egenutvecklade simuleringsmiljön Aquarius. Aquarius är en C++-baserad simuleringsmiljö som utvecklats med fokus på modellering och simulering av lager 2 och 3 i OSI-stacken. Fokus för utvecklingen av programmet har varit att erbjuda stöd för en omfattande modellabstraktion så att enskilda systemfunktioner enkelt ska kunna testas.

Det finns idag flera simuleringsmiljöer för kommunikationssystem att tillgå, både kommersiella och open source. Bland de kommersiella aktörerna är OPNET [1] idag dominerande medan ns2 [2] är dominerade bland open source alternativen. Vilken simuleringsmiljö som är lämpligast att använda beror på flera olika faktorer så som exempelvis inriktningen på de simuleringar som ska göras, samarbeten med andra och tillgång på kompetens.

Inom den svenska försvarsmakten (FM) och Försvarets Materielverk (FMV) används idag OPNET/JCSS [1, 3] som simuleringsmiljö för utbildning och övergripande evalueringar av kommunikationssystemslösningar. OPNET är en C/C++-baserad simuleringsmiljö med ett omfattande modellbibliotek med fokus på lager 3-5 i OSI-stacken. Simuleringsmiljön är i första hand framtagen i syfte att utnyttja det befintliga modellbiblioteket men stöd finns även för att utveckla egen modeller.

Inom FOI finns en flerårig erfarenhet att använda OPNET för att genomföra simulering och modellering av kommunikationssystem. Projektets bedömning

är att OPNET/JCSS är användbart för MoS av befintliga system eller specificerade system under utveckling och upphandling.

Inom FOI:s långsiktiga forskningen, såsom FoT, finns dock behov av att kunna experimentera med konceptuella idéer. För att kunna genomföra simuleringar av system som som ännu bara är på idé-stadiet krävs normalt att nya simuleringsmodeller utvecklas. Denna kravbild har lett till att FOI förutom OPNET även använder det egenutvecklade simuleringsmiljön Aquarius i de fall där det anses vara mest kostnadseffektivt.

I denna rapport analyseras de tre olika utvecklingslinjerna produktifiering, vidmakthållande och avveckling för simuleringsmiljön Aquarius. Dessa presenteras mera utförligt i kapitel 3. De olika utvecklingslinjerna analyseras sedan med avseende på ekonomi, samarbete, utveckling av modeller, hantering av modeller och simuleringskapacitet i kapitel 4. Resultatet av denna analys sammanfattas sedan i kapitel 5. Rapporten avslutas med ett appendix där en av utvecklingslinjerna fördjupas en del.

2 Aquarius idag

Aquarius är ett simuleringsramverk med tillhörande modellbibliotek som utvecklats för simulering av mobila ad-hoc nät. Målsättningen vid utvecklandet av Aquarius har varit att skapa en simuleringsmiljö som tillåter att delfunktioner i ett kommunikationssystem kan utvärderas utan krav på att alla funktioner är specificerade i detalj. De flesta av de genomförda simuleringarna med Aquarius bygger således på att vissa delfunktioner har implementerats mycket väl medan andra delfunktioner har mera principiella implementationer. Genom detta metodval kan utvecklingskostnaderna hållas nere samtidigt som möjligheten att testa delfunktioner under mera kontrollerbara förhållanden ökar. Detta ger i sin tur ökad förmåga att skapa förståelse för hur olika delfunktioner fungerar under olika förhållanden.

2.1 Modellbibliotek

Ramverket har framförallt använts till att studera frågeställningar rörande routing och resursfördelning på datalänknivå, det vill säga lager 2 och 3 i en OSI stack, se figur 2.1. Innehållet i Aquarius-modellbibliotek har styrts av behovet av modeller i de projekt som har använt verktyget. Det fokus som varit mot lager 2 och 3 i en OSI-stack återspeglas därför tydligt i modellbiblioteket. För att kunna testa de olika protokollen finns dessutom diverse applikationer samt ett enkelt transportlager.

På datalänklagret finns idag tre huvudprotokoll, dels en generisk IEEE 802.11 implementation, dels ett distribuerat STDMA protokoll och dels ett centraliserade TDMA protokoll. Genom att byta moduler i protokollen såsom kösystem och tidluckeallokerings metod kan protokollens beteende förändras i grunden utan att omfattande omprogrammering krävs.

På nätnivå finns ett antal routingprotokoll, både ideala, reaktiva och proaktiva. De ideala routingprotokollen bygger på att routerna fås från ett orakel vilket kan vara fördelaktigt om en kontrollerbar omgivning är önskvärd vid simulering av exempelvis ett datalänk protokoll.

Transportlagret i Aquarius erbjuder stöd för UDP liknande sessioner, med och utan svar från mottagaren. På applikationslagernivå finns idag tre huvudtyper av applikationer; sessionsbaserade, Poisson-baserade och en sensornätbaserade. De sessionsbaserade applikationerna används normalt för att modellera tal och filöverförings liknande applikationer. Trafiken som sensornätet genererar motsvarar ett sensornät med distribuerad dataaggregering.

Förutom modeller av kommunikationssystem innehåller också modellbibli-

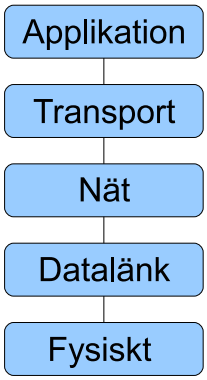


Bild 2.1: De i Aquarius modellerade delarna av en OSI-stack. Fokus har framförallt varit nät och datalänk-lagret.

oteket måttobjekt som kan användas för att estimera hur väll de olika protokollen fungerar. Måttobjekten i Aquarius är jämfört med motsvarande funktionalitet i många andra ramverk relativt avancerade. Det finns exempelvis måttobjekt som kan jämför ett routingprotokolls router med för en använd metrik optimal router medan andra objekt kan estimera hur väl en talförbindelse fungerar sett till paketförlust och fördröjning.

2.2 Utvecklingsmiljö

Verktyget har utvecklats enligt filosofin att finns det ett program som kan lösa en uppgift så ska inget nytt utvecklas. Aquarius sköter således enbart själva simuleringen av systemet medan scenariobyggandet och analysen av resultaten sker i andra verktyg, normalt Matlab [4]. På samma sätt innehåller Aquarius ingen egen utvecklingsmiljö för kod eller egen debugger utan utvecklingen sker i det verktyg som användaren föredrar. För närvarande stöds kompilatorerna Microsoft Visual Studio [5] och GCC [6].

För att framtagning av nya modeller till Aquarius ska kunna ske på ett effektivt och strukturerat sätt innehåller ramverket ett omfattande stöd för att skriva tester av modellerna. Det nuvarande testramverket erbjuder stöd för tester av både enskilda klassers medlemsfunktioner och tester av hela scenarion.

3 Möjliga utvecklingslinjer för Aquarius

Projektet bedömer att FM och FMV även fortsättningsvis kommer att utnyttja OPNET för kommunikationsnätssimuleringar. Detta gör att FOI har ett behov av att kunna stödja FM och FMV med simuleringskompetens för OPNET. Troligen finns även ett mindre behov av kompetens för modellutveckling i OPNET även om FOI:s roll här troligen är mera som expertstöd vid upphandling av modeller från extern part.

Den frågeställning som denna rapport främst fokuserar på framtiden är för det egenutvecklade ramverket Aquarius. De huvudalternativ som behandlas i rapporten är en produktifiering av Aquarius, ett långsiktigt vidmakthållande av Aquarius samt en avveckling av Aquarius. FM:s och FMV:s behov av OPNET-stöd gör att som alternativ till Aquarius är det svår att se något annat än OPNET. I fallet med en avveckling antas således att all kommunikationsnätssimulering i framtiden sker i OPNET. I de två övriga fallen antas att FOI behåller kompetens för att stödja FM och FMV i OPNET-frågor men att Aquarius kvarstår som huvudverktyg för den långsiktiga forskningsverksamheten inom området.

3.1 Produktifiering

Målsättningen här är att vidareutveckla Aquarius så att programmet kan användas av en extern part. Med en extern part avses här både ett annat projekt inom FOI eller en extern organisation. Möjligheten att ta betalt för program torde dock vara mycket begränsade sett till marknadens struktur med god tillgång till både kommersiella och open source alternativ. Möjligen skulle programmet kunna användas som bytesvara vid projektsamarbeten. FOI tillhandahåller då programmet mot att samarbetsparterna deltar i modellutvecklingen av gemensamma modeller för projektet.

Det skulle också kunna vara tänkbart att andra FOI projekt har behov av en kommunikationsnätssimulator för att simulera kommunikationen i mera övergripande simuleringar. Aquarius har redan idag använts för sådana ändamål inom projektet Interaktiva Adaptiva Marksensorer. Som återföring till Aquarius blev det i det speciella fallet en applikationsmodell för sensordatafusion.

3.2 Vidmakthållande

Målsättningen här är att skapa ett väl testat och dokumenterat program som nya programmerar enkelt kan ta till sig. Programmet kommer även fortsättningsvis

främst att vara avsett för simuleringar av lager 2 och 3 i en OSI-stack. Någon målsättning att möjliggöra användning av programmet för externa användare utanför kompetensgruppen finns inte här utan dokumentationen och testerna är bara till för internt bruk.

3.3 Avveckling

Målsättningen här är att avveckla Aquarius på ett ordnat sätt. De modeller som finns i Aquarius och som är viktiga för den fortsatta verksamheten bör här portas till OPNET. Genom att genomföra en avveckling av Aquarius och koncentrera resurserna på en simulatormiljö bör kunskapen om OPNET på sikt öka. Eftersom OPNET är det mest troliga simuleringsmiljön vid ett extern samarbete så bör en sådan utveckling även vara positivt ur samarbetssynpunkt.

4 Analys

De i kapitel 3 föreslagna utvecklingslinjerna har alla olika för och nackdelar. I detta kapitel försöker vi att belysa de viktigaste. Vi börjar med att titta på de ekonomiska aspekterna, varefter vi studerar hur vår förmåga till externa samarbeten påverkas. Vi går sedan vidare med utveckling av modeller samt hanteringen av ramverket varefter vi avslutar med frågan om simuleringskapacitet.

4.1 Ekonomi

Ur ett ekonomiskt perspektiv innebär produktifieringslinjen att vi initialt måste satsa avsevärda resurser på utveckling av dagens Aquarius. Sett till marknadsläget är dock möjligheten att sälja programmet till en extern kund mycket begränsad. Den möjlighet som finns att få tillbaka investerade resurser ligger i att använda programmet som bytesvara vid samarbeten. Vi tillhandahåller då programmet mot att samarbetsparterna deltar i modellutvecklingen av gemensamma modeller för projektet. Vidare innebär en produktifiering att vi långsiktigt riskerar ta på oss kostnader för underhåll och support. Dessa kan bli svåra att långsiktigt hantera om vårt behov av programmet minskar och ansvariga utvecklare byter uppgifter.

Väljer vi att istället satsa på vidmakthållandelinjen krävs även då resurser för att dokumentera och utveckla ramverk. Vidare krävs att det finns projekt som är intresserade av att använda ramverket så att vi inte får ett tomt ramverk utan modeller. Denna utvecklingslinje kräver dock mindre resurser än produktifieringslinjen eftersom enbart kärnan beaktas. Vidare kan utvecklingen spridas över än längre tid.

Utvecklingslinjen avveckling innebär visserligen att vi inte behöver investera mera i Aquarius, samtidigt drar vi dock på oss kostnader för att portera vissa modeller från Aquarius till OPNET. Vidare krävs sannolikt investeringar i fråga om både ny hårdvara att köra OPNET på samt flera licenser. Behovet av nya utvecklingslicenser är dock troligen lågt utan i första hand handlar det om att införskaffa några licenser som enbart stödjer simulering, vilka är billigare. Initialt är kostnaden här stor men på sikt är det enbart årliga licenskostnader vi behöver betala. Om dessa kommer att överstiga kostnaden för att långsiktigt underhålla Aquarius beror på hur många licenser och hur mycket hårdvara vi anser att vi har behov av. Avgörande är också hur många modeller utanför OPNET:s modellbibliotek vi använder eftersom dessa måste underhållas av oss själva. Att bara använda OPNET som ett simuleringsramverk med huvudsakligen egna modeller är knappast ett ekonomisk fördelaktigt alternativ.

4.2 Samarbete

En produktifiering av Aquarius gör att vi lättare skulle kunna använda programmet som en plattform vid samarbeten. Sedan tidigare har det förekommit interna samarbeten där Aquarius har använts. Vid dessa har dock själva arbetet med Aquarius skötts av personal från kompetensgruppen. Denna form av samarbete får nog anses som troligaste formen när det gäller interna samarbeten vilket gör att möjligheten till interna samarbeten knappast påverkas av en produktifiering av Aquarius. För att det långsiktigt ska gå att använda Aquarius i interna samarbeten måste dock åtminstone vidmakthållandelinjen genomföras så att programmets långsiktiga överlevnad hanteras mera systematiska. Att internt använda OPNET vid samarbeten skulle kunna vara ett alternativ i de flesta fall. I alla fall så länge ramverket inte ska integreras med andra simulatorer. Visserligen stödjer OPNET detta via exempelvis HLA men eftersom frågan om tillgång på OPNET licenser måste lösas riskerar det hela att stupa på licenskostnaderna.

Vid samarbeten med externa organisationer där det finns behov av att gemensamt genomföra kommunikationsnätssimulering är OPNET det mest troliga valet av verktyg. Detta gäller både i fallet med industriella samarbetspartner och andra forskningsinstitut. I alla fall sett till den nisch av kommunikationsforskningsvärlden som vi huvudsakligen har ett utbyte med. Att utnyttja en produktifierad version av Aquarius vid samarbeten skull kunna vara möjligt om samarbetspartner saknar simuleringsramverk. Denna situation är troligast vid samarbeten med universitet och högskolor. Eftersom OPNET erbjuder universitet och högskolor bra rabatter bör dock även OPNET kunna vara ett alternativ här.

Att avveckla Aquarius och helt satsa på OPNET skulle öka vår förmåga till samarbeten eftersom vår kunskap om OPNET på sikt skulle öka. Det är dock viktigt att beakta att val av verktyg för ett gemensamt projekt inte räcker för att möjliggöra gemensamma simuleringar eftersom olika versioner av verktyget och tillhörande modeller inte behöver vara kompatibla. Risk finns alltså att olika policy rörande versionshantering av program och modeller ställer till problem vid samarbeten även om verktygen kommer från samma leverantör.

4.3 Modellutveckling

Med den nuvarande inriktningen på verksamheten finns ett behov av att utveckla nya modeller till ramverket. Genomförs produktifieringslinjen eller vidmakthållandelinjen antas detta behov kvarstå. Genomförs avvecklingslinjen är det

dock möjligt att arbetsmetodiken kommer ändras för att bättre passa till OPNET's styrkor och svagheter. Vi antar dock fortsättningsvis i detta kapitel att behovet att utveckla modeller kvarstår.

En produktifiering av Aquarius skulle, givet att någon använder produkten, kunna leda till att antalet modeller i ramverket ökar. Detta bygger dock på att användare är beredda att dela med sig av sina modeller. Eftersom den mest sannolika affärsmodellen vid en produktifiering förefaller att vara tillgång till Aquarius mot att vi får tillgång till framtagna modeller så bör dock en spridning av programmet inverka positivt på modellutvecklingen.

Genomförs vidmakthållandelinjen bör Aquarius kärna bli mera lättillgänglig vilket bör underlätta för ny utvecklar som ska börja arbeta med programmet. Detta bör i sin tur kunna leda till att flera projekt väljer att göra sina simuleringar i Aquarius istället för i exempelvis Matlab. En sådan utveckling är önskvärd eftersom ett ramverk utan användare och modeller är meningslöst.

En avveckling av Aquarius till förmån för OPNET leder visserligen till att OPNETS stora modellbibliotek utnyttjas i högre grad. Modellbiblioteket är dock främst välutvecklat när det gäller fast kommunikationsutrustning. Men de vanligast routing och access protokollen som vi använder finns implementerade. OPNET's stora svaghet är dock stödet för nyutveckling av modeller. De största problemen här är:

- C orienterat programmeringsspråk med många makron
- Många råa pekare inklusive många void pekare
- Egen mycket bristfällig utvecklingsmiljö
- Svagt stöd för versionshantering
- Inget stöd för testning av koden

Dessa faktorer gör att även om utvecklaren kan OPNET väl finns ett antal riskfaktorer som gör att predikterbarheten i kostnaden för kodutveckling riskerar att bli dålig.

4.4 Hantering av ramverk

Aquarius saknar idag ett integrerat grafiskt gränssnitt. I stället används Matlab både för att skapa scenarier till Aquarius och för att analyser resultat från Aquarius. Själva körningen av Aquarius sker normalt i ett skalfönster. Designfilosofin har här varit att finns det ett fungerade program som kan lösa en uppgift

så ska inte ett nytt utvecklas. Om denna designfilosofi behålls vid en produktifiering handlar en produktifiering främst om dokumentation och testning av kärnan och modeller. En möjlig utveckling skulle kunna vara att bygga ett grafiskt gränssnitt i Matlab för att köra programmet.

Att ta fram ett helt fristående grafiskt gränssnitt som stödjer både scenarioframtagning, simulering och analys behöver inte heller vara önskvärt. Speciellt med tanke på erfarenheten av OPNET:s grafiska gränssnitt. Vid en första anblick verkar gränssnittet bra. Vid större simuleringar har det dock ofta visat sig lämpligast att både bygga sceneriet och analysera resultaten i andra program.

De långsiktigt sett största konsekvensen av att produktifiera Aquarius är dock sannolikt att vi förbinder oss att underhålla programmet. Även om antalet plattformar som vi stödjer är mycket begränsat så ska inte kostnaden för detta underskattas. Vidare kan en produktifiering innebära krav på att kunna ge support till eventuella användare. För att kunna uppfylla dessa krav krävs att antalet personer som kan underhålla ramverket och ge support ökar för att minska problemen vid personalförändringar.

Ska ramverket långsiktigt underhållas så krävs att dagens personberoende försvinner. För att detta ska vara möjligt måste dokumentationen av kärnan förbättras. Vidare måste tester skrivas för huvuddelen av kärnan så att förändringar kan ske på ett säkert sett. För en mera utförlig beskrivning av vad som behöver göras med kärnan givet att produktifierings- eller vidmakthållandelinjen väljs se appendix A.

4.5 Simuleringskapacitet

Den idag använda arbetsmetodiken vid nätsimuleringar innebär ofta omfattande simuleringar för att skapa förståelse för hur enskilda systemfunktioner påverkar radiosystemet. Denna arbetsmetodik ställer relativt höga krav på simuleringskapaciteten. För att skapa simuleringskapacitet används idag relativt många datorer av varierande kapacitet. Eftersom antalet Aquarius licenser inte är begränsat fungerar detta bra. Den nuvarande simuleringskapaciteten kommer inte att påverkas i någon högre grad om produktifierings- eller vidmakthållande linjen genomförs. Visserligen kommer kärnan att effektiviseras en del men totalt sett blir förändringen ändå begränsad.

Genomförs avvecklingslinjen förändras dock läget eftersom antalet OPNET-licenser är begränsat, i nuläget till två. En metod att hantera detta är att ändra forskningsinriktning till en mindre simuleringsresurskrävande. Alternativet är att köpa flera licenser eller bättre hårdvara. Ska licenserna bara användas för att köra simuleringar finns det idag billigare licenstyper att tillgå. Kostnaden är

dock fortfarande relativt hög så initialt är antagligen bättre hårdvara ett mera ekonomisk val.

5 Slutsats

Målsättningen med denna rapport har inte varit att rekommendera vilken utvecklingslinje som ska väljas för Aquarius utan bara att analysera effekterna av olika val. Hur olika utvecklingslinjer ska viktas mot varandra beror också på hur olika aspekter såsom exempelvis ekonomi och möjligheten till samarbete viktas relativt varandra. Som slutsats av rapporten presenterar vi därför en sammanfattning av konsekvenserna av de olika utvecklingslinjerna i tabell 5.1. Vid denna jämförelse har vi valt att uttrycka konsekvenserna relativt dagens situation. Ett grundantagande som gäller för samtliga utvecklingslinjer är att förmågan att använda simuleringsmiljön OPNET antas kvarstå.

	Produktifiering	**Vidmakthållande**	**Avveckling**
Ekonomi	Initiala investeringar med långsiktigt tveksam avkastning	Visst långsiktigt resurstillskott för underhåll och dokumentation	Nya OPNET licenser samt underhåll av befintliga krävs
Samarbete	Möjligen bättre men det finns gott om alternativa simuleringsmiljöer	Oförändrat	Klar förbättring eftersom OPNET är det troligaste verktygetsvalet vid ett samarbete
Utveckling av modeller	Egen utveckling underlättas samtidigt som möjlighet att få tillgång till externt producerade modeller ökar	Underlättas eftersom dokumentationen och gränssnitt blir mera lättanvända	Bättre tillgång till externa modeller men egen utveckling försvåras pga arbetskrävande programmeringsmetod
Underhåll av ramverk	Långsiktigt underhåll krävs men bättre dokumentation och enklare gränssnitt bör underlätta	Bättre dokumentation och enklare gränssnitt bör underlätta	Hanteras av OPNET
Simuleringskapacitet	Oförändrat	Oförändrat	Begränsas av antal OPNET licenser

Tabell 5.1: Konsekvenser av olika strategier.

Av de tre utvecklingslinjerna är produktifieringslinjen den mest osäkra då det är svårt att se en potentiell köpare. Vilken linje som är bäst av de två övriga beror på inriktningen och omfattningen hos den långsiktiga forskningen. Av-

vecklingslinjen ger visserligen ökade samarbetsmöjligheter med externa forskningsorganisationer då OPNET är ett relativt vanligt förekommande verktyg. Detta ska dock vägas mot att kostnaderna för vidmakthållandelinjens långsiktiga underhåll av Aquarius bedöms var klart lägre än de ökade licenskostnaderna för OPNET om avvecklingslinjen genomförs. Vidare är kostnaderna för att ta fram nya modeller lägre och mera predikterbara med Aquarius än med OPNET.

A Utveckling Aquarius

Av de tre föreslagna utvecklingslinjerna för Aquarius innebär produktifierings- och vidmakthållandealternativen att dokumentering och en viss utveckling av Aquarius måste ske. I detta kapitel presenteras den dokumentering och utveckling som på sikt behöver ske om vidmakthållandelinjen väljs. Aquarius består dag av två huvuddelar, dels kärnan och dels olika modeller av diverse kommunikationssystemskomponenter. Väljs vidmakthållandelinjen berör dokumenteringen och utvecklingen i huvudsak kärnan. Väljs produktifieringslinjen behöver dock alla delar av programmet som långsiktigt ska bevaras dokumenteras och testas.

Vid denna process är det viktigt att beakta de förändringsförslag av kärnan som föreslås nedan eftersom dessa påverkar det som ska dokumenteras. Så även om det viktigaste troligen är att dokumentera kärnan först bör de föreslagna förändringarna av kärnan införas så att dokumenteringen inte måste göras om. De nedan föreslagna förändringarna kommer dock främst att påverka kärnan internt så steg 1 ovan bör gå att genomföra relativt oberoende av förändringsarbetet.

A.1 Aquarius struktur

Aquarius struktur kan betraktas ur flera vyer. Vi väljer här att dels betraktar de olika klassernas inbördes beroenden samt dels betrakta hur instanser av olika klasser sättas samman så att de representera ett kommunikationssystem.

A.1.1 Klasstruktur

Klasstrukturen i Aquarius kärna är idag relativt komplex med många beroenden mellan olika klasser. För en principskiss av strukturen hos de klasser som används vid framtagandet av nya modeller kan vi betrakta figur A.1. Alla klasser har här ett beroende av klassbiblioteket Boost [7]. Vidare antas användarna bara direkt utnyttjar klasserna I1 och I2 medan klasserna C1-C4 utgör rena implementationsklasser för klasserna I1 och I2. Beroendena mellan klasserna C1-C4 samt det faktum att gränssnitten I1 och I2 har ett komplext beroende av implementationsklasserna C1-C4 gör att kostnaderna för att testa och underhålla kärnan riskerar att bli onödigt höga.

En mera lämplig struktur för en framtida kärna är den som visas i figur A.2. Funktionalitet som är gemensam för flera gränssnitt har här samlats i Base på

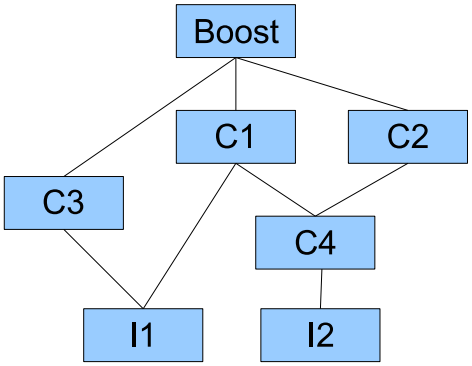


Bild A.1: Principskiss av dagens klassberoenden i kärnan. Karakteriserande för designen är de korsvisa kopplingarna mellan olika implementationsklasser C1-C4 och gränssnitten I1 och I2.

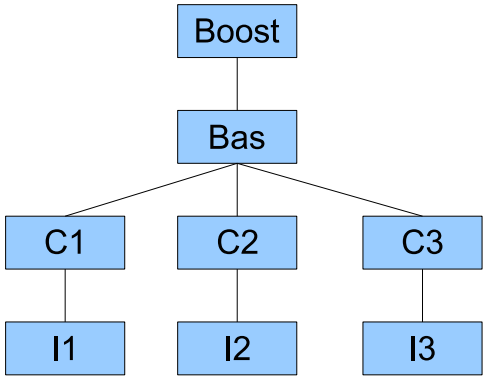


Bild A.2: Principskiss av framtida klassberoenden i kärnan. Karakteriserande för denna struktur är att gemensamma funktionalitet lyfts till en tydlig basnivå samt de flera men enklare gränssnitten I1-I3. Vidare försvinner kopplingen mellan de olika implementationsklasserna C1-C3.

en egen nivå som en påbyggnad till Boost. De olika gränssnitten har här också renodlats och blivit flera men mindre. Vidare är implementation av de olika gränssnitten helt oberoende av varandra förutom den funktionalitet som ligger i Boost och Base.

A.1.2 Objektstruktur

Betraktar vi den objektstruktur som skapas då en simulering körs i Aquarius så kommer en kommunikationsnod representeras av ett antal objekt som modellerar de olika delarna i en protokollstack. Förutom de objekt som representerar olika protokoll i noderna finns ett extra objekt för varje protokoll. Detta objekt

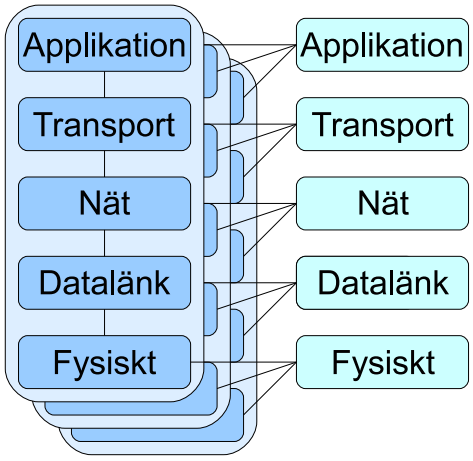


Bild A.3: Exempel på objektstruktur med tillhörande bindningar i Aquarius. Objektet som representerar nodernas protokollinstanser är grupperade medan de gröna objekten representerar de centrala instanserna.

är gemensamt för alla objekt på en viss protokollnivå och kan användas till att implementera funktionalitet som lämpligt bör vara centraliserad, se figur A.3. Exempel på sådan funktionalitet är en dubbelriktad tal applikation eftersom dessa ofta beskrivs som en tillståndsmodell som är gemensam för de båda deltagarna.

För att förbättra programstrukturen och öka återutnyttjadegraden av koden har många objekt idag implementerats med hjälp av dynamiskt allokerade underobjekt. Alla paketköer är idag exempelvis implementerade på detta sätt vilket möjliggör dels ett effektivt kodutnyttjade och dels att olika kötyper på ett enkelt sätt kan testas i alla modeller. Denna utveckling har dock gjort att den dagens strukturen med extra objekt för centraliserad funktionalitet har blivit otillräcklig och behöver bytas ut. I flera fall har nämligen behovet att även underobjekt är centraliserade uppstått. Detta går att hantera med dagens system men eftersom konstruktionen kräver att en hel del stödfunktionalitet skrivs explicit för varje fall är detta inte en bra lösning på lång sikt. Istället bör dagens system med ett centralt objekt för varje protokoll ersättas med ett system där godtyckligt objekt kan allokeras som centralt.

I figur A.4 visas ett exempel på dagens objektstruktur. Noderna N1-N3 innehåller här objekten O samt underobjekt q. För att åstadkomma ett centralt underobjekt till q måste även ett central objekt motsvarande O skapas vilket ger strukturen i figur A.4. En bättre objektstruktur för situationen där D.O i figur A.4 enbart finns för att möjliggöra objektet D.O.q är den som visas i figur A.5. Här binds objektet D.O.q direkt till N1.O-N3.O objekten.

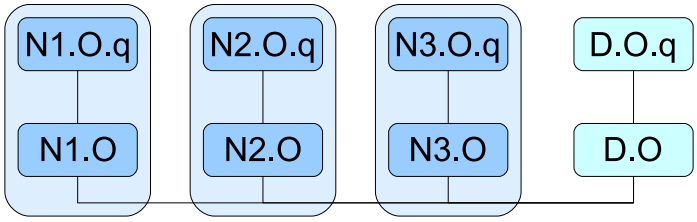


Bild A.4: Exempel på dagens objektstruktur med noderna N1-N3 innehållande objektet O och underobjekten O.q samt ett central objekt D.0 med underobjektet D.O.q.

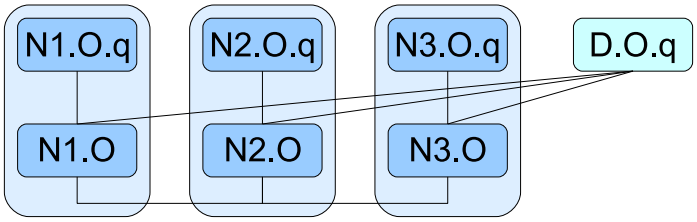


Bild A.5: Exempel på önska objektstruktur. Jämfört med figur A.4 behövs här inget central objekt D utan D.O.q kan bindas direkt till N1.O-N3.O.

A.1.3 Adresser

Alla adresser som används för att sända packet till rätt mottagare representeras idag av en `int`. I de flesta protokollen sätts både den logiska adressen på nätlagret och den fysiska adressen på datalänklagret till samma värde. För att underlätta en mera generell adresshantering bör alla dagens adresser ersättas med klasser. Genom att låta adressklasserna få en storlek som inparameter kan dessutom den använda adresstorleken variera på ett enkelt sätt via inparametrarna till programmet.

A.2 In- och utdata

Dagens system för inmatning och utmattning av data till Aquarius fungerar relativt väl. Användargränssnittet för att lägga till ny måttobjekt för att samla in utdata är visserligen relativt komplext men eftersom det knappast är möjligt att skriva ett enkelt samtidigt som styrkan i dagens mått kvarstår bör systemet få vara kvar relativt oförändrat. Vissa förändringar rörande inmatning av data skulle dock kunna vara lämpliga att genomföra för att dels förenkla körandet av simuleringar samt dels öka integrationen med Matlab samtidigt som möjligheten till andra front-ends förbättras.

Ett första steg in en sådan anpassning är att byta indataformat. För att sätta attributet `size` till `5` för nod 0 till nod 11 används idag syntaxen

```
N0_N11.size = 5
```

En syntax som skulle vara mera konform med Matlab är

```
for i = 1:12
    N(i).size = 5
end
```

Denna syntax fungerar också bra som utformat. Observeras bör dock att nod 0 i Aquarius heter nod 1 i Matlab på grund av de olika indexsystemen i C och Fortran. Detta problem finns dock redan idag till viss del eftersom utdata i många fall lagras i matriser. För öka kompabiliteten mot XML samt öka läsbarheten bör toppnamn införas samtidigt som `N` skrivs ut så att `N.size` blir `net.node.size`.

Parallellt med denna förändring bör även namnsystemet för att sätta upp mått ändras så att mått av samma typ men med olika inparametrar, typ sampelintervall, hanteras på ett enklare sätt en dagens där användaren explicit måste ange att de har olika inparametrar. Genom införandet av namnsyntaxen ovan bör denna förändring underlättas avsevärt.

A.3 Framdrivning

För att driva simuleringen framåt används dels en händelse hanterare dels direkt kommunikation mellan de i simuleringen ingående objekten. Vidare finns även ett antal vanliga slumptalsgeneratorer implementerade. Funktionsmässigt fungerar dagens struktur väl, även om ett visst behov av enklare gränssnitt och effektivare kod dock kan ses.

A.3.1 Händelsehanterare

Den idag använda händelsehanteraren har i stor fungerat bra. Det gränssnitt som erbjuds för att schemalägga fria händelser och händelser med hanterare skiljer sig dock åt och bör ensas. Vidare bör minneshantering stramas upp så enbart `boost::weak_ptr` kan schemaläggas. I enlighet med den ovan föreslagna strukturella förändringarna bör också den direkta kopplingen mellan händelsehanteraren och inmatningen av data tas bort. Parametrar till händelsehanteraren bör istället sättas via gränssnittet till denna.

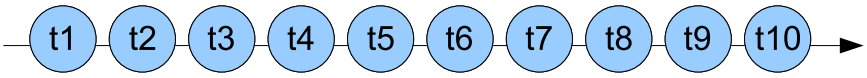


Bild A.6: Dagens händelsehanterare där alla händelser läggs på en gemensam kö.

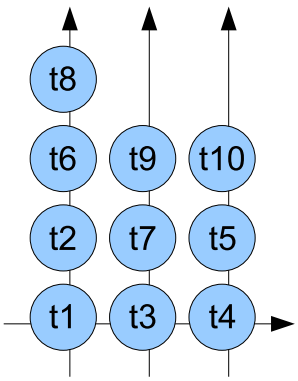


Bild A.7: Framtida händelsehanterare där de lokala köerna $\{t1, t2, t6, t8\}$, $\{t3, t7, t9\}$, $\{t4, t5, t10\}$ köas på i en gemensam kö.

Förutom dessa mera kosmetiska åtgärder finns det dock ett behov att effektivisera händelsehanteraren för vissa specifika fall. Vissa protokoll schemalägger idag relativ många händelser parallellt. Exempelvis kommer vissa applikationer att schemalägga minst 1 händelse för varje session som är igång. Eftersom antalet parallella sessioner kan bli stort innebär det att händelsekön kan bli lång vilket gör att kostnaden för att sorterar in nya händelser blir stor, se figur A.6. En metod att hantera detta på är att låta alla händelse som ett visst objekt schemalägger läggas i en lokal händelsehanterare i objektet. Denna schemalägger sedan en händelse i den globala händelsehanteraren. Antalet händelser som schemaläggs ökar visserligen men eftersom de enskilda köerna blir kortare finns mycket tid att spara vid insättning av händelserna. Vidare tenderar många protokoll att schemalägga nya händelser tidsmässigt efter den av protokollet senast schemalagda händelsen. Detta gör att insättningen i den lokala kön kan bli mycket billig eftersom de flesta händelser ska stoppas in sist. I figur A.7 visas ett exempel på en händelsekö med bestående av 3 lokala köer.

En möjlig vidareutveckling av detta är att låta grupper av objekt, exempelvis alla instanser av en viss protokolltyp, använda en gemensam lokal händelsehanterare. Detta skulle troligen i många fall vara effektivt eftersom många av händelserna skulle sättas in sist i den lokala kön. En lämplig vidareutveckling

av dagens händelsehanterare skulle därför kunna vara att inför tre alternativ genom en mall parameter

Parent: Använder förälderns hanterare – default – Lämpligt för små dynamiskt allokerade objekt

Sibling: Använder syskons hanterare – Bra om en viss protokollnivå totalt sätt kan anses skapa händelser som kan schemaläggas som O(1)

Local: Använder en lokal hanterare – Bra om objekt (eller barn) schemalägger många händelser, typiskt minst proportionellt mot antal noder

A.3.2 Intern kommunikation mellan objekt

Kommunikationen mellan objekt i en nod består dels av de informationspaket som ska sändas mellan olika noder och dels intern kommunikation mellan objekt i samma nod. Den senare typen av kommunikation består av två delar dels direkta frågor mellan olika objekt och dels utnyttjade av observatörer[1] för att kommunicera tillståndsförändringar i objekten.

Meddelandestruktur för internkommunikation

För att olika objekt ska kunna kommunicera med varandra utnyttjas idag objektens namn tillsammans med kunskap om protokollstrukturen. Så länge behov finns att enbart kommunicera med objekt som ligger intill varandra i protokollstacken har detta system fungera tillfredsställande. I flera fall har det dock uppstått ett behov för mera avlägsna protokoll att utbyta information direkt. Detta är fullt möjligt men kräver att användaren har koll på att den eftersökta informationen inte behöver filtreras av några mellan liggande protokoll. Ett exempel på detta finns i figur A.8. Nätlagret vill här få information om vilka grannoder vi kan kommunicera med från det fysiska lagret. Vart nätlagret ska vända sig efter denna information beror dock på vilket datalänklager som används. Om datalänklagret kan besvara frågan bör nätlagret lämpligen vända sig till datalänklagret men om datalänklagret inte kan besvara frågan måste nätlagret fråga det fysiska lagret direkt.

Genom att implementera ett intern meddelandesystem i alla noder som låter olika protokoll sända ut frågor om information de vill ha, typ aktuella länkar bör

[1] Om objekt A vill veta när en viss tillståndsförändring sker B registrerar A en observatör i B. När en förändringen sker meddelar B A's observatör som i sin tur meddelar A.

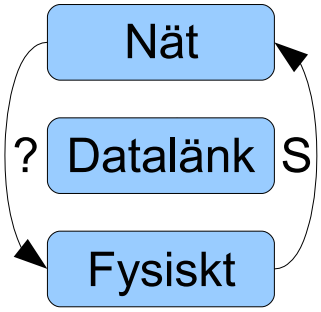


Bild A.8: Nätlagret vill ha information från det fysiska lagret och hämtar informationen direkt från fysiska lagret. Om datalänklagret hade kunnat besvara frågan skulle kommunikationen skett direkt med datalänklagret. För att nätlagret ska vända sig till rätt objekt krävs idag sidoinformation från användaren.

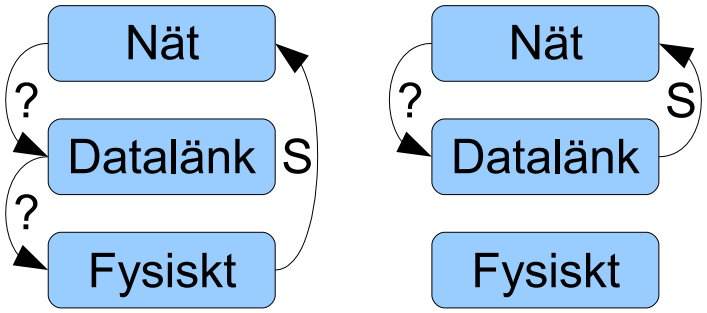
dock kravet på att användaren vet vad olika protokoll kan svara på att minska. I fallet ovan sänder således nätlagret en fråga till datalänklagret. Om datalänklagret inte kan besvara den frågan sänder datalänklagret lagret vidare den, i detta fall till fysikalagret, se figur A.9. Genom att införa ett sådant system bör kraven på användarnas kunskap minska samtidigt som alternativa protokollstrukturer lättare kan testas.

Interna observatörer

Aquarius erbjuder idag ett omfattande stöd för observatörer via `Interrupt` klasserna med tillhörande container klasser. Dessa fungerar idag bra. Genom att utnyttja Boost biblioteket `funktion` samt `signals` skulle det dock gå att generalisera dagens implementation samtidigt som kodmängden minskar vilket underlättar framtida underhåll. Det finns dock behov av att även fortsättningsvis kunna hantera nollställning av alla observatörsregister centralt vilket sannolikt leder till att en fullständig övergång till `signals` biblioteket inte är möjligt utan att en kombination av dagens kod, `funktion` och `signals` är den mest lämpliga vägen.

A.3.3 Slumptalsgenerering

Aquarius använder idag en extern implementation av slumptalsgenerator Mersenne twister för att generera likafördelade slumptal. För att få fram olika fördelningar används sedan Aquarius intern kod. All denna kod borde ersättas av motsvarande kod i Boost. Eventuellt borde denna kod fixas till så att olika lager



(a) Datalänklagret kan inte svara utan sänder vidare frågan till fysiska lagret som svarar till nätlagret.

(b) Datalänklagret besvarar frågan direkt till nätlagret.

Bild A.9: Intern kommunikation med hjälp av frågor via mellanliggande lager. Frågor som inte kan besvaras sänds vidare till nästas lager. Eventuelle svar sänds direkt till frågeställaren.

i protokollstacken kan få egna slumptalsgenerator vid behov. Detta skulle göra att samma trafiklast kan testats på olika protokoll.

A.4 Dokumentation, testning och debuggning

Aquarius dokumentation är idag bristfällig både ifråga om övergripande strukturella beskrivningar och direkta kommenterar i koden. Detta gäller både kärnan och flertalet av de modeller som finns implementerade. Vidare har bara delar av koden medföljande enhetstester. För att programmet långsiktigt ska kunna överleva finns ett stor behov av att öka dokumentationsgraden samt mängden enhetstester. Eftersom arbetet är omfattande är det viktig att prioritera vad som ska dokumenteras och eventuellt testats samt i vilken ordning detta ska ske. Arbetet kompliceras dock en del av att vissa delar i kärnan behöver åtgärdas vilket gör att eventuell befintlig dokumentation om dessa riskerar bli inaktuell. För att undvika detta bör därför eventuella arbeten på kärnan som påverkar dokumentation ske innan dokumentationen sker.

Fokus för dokumentationen bör inledningsvis vara att ta fram ett dokument som beskriver kärnans yttre gränssnitt som modellutvecklaren använder. Dokumentet bör även innehålla exempel på hur gränssnitten är tänkt att användas. För att öka antalet personer som kan underhålla kärnan bör sedan ett dokument som beskriver kärnans interna struktur tas fram. För att dessa båda dokument ska kunna hållas på en övergripande nivå som inte fastnar i djuplodande beskrivningar av kärnans klasser är det viktigt att de klasser som behandlas i do-

kumenten parallellt dokumenteras i koden. Vidare är det viktigt att de klasser som förändras i kärnan tillförs enhetstester så att deras funktionalitet kan garanteras. Långsiktigt bör här även äldre befintliga klasser tillföras enhetstester eftersom detta underlättar framtida underhåll av kärnan.

Det finns idag ett omfattande stöd i Aquarius för att skriva enhetstester. För tester som utnyttjar kärnan funktionalitet för att läsa in- och utdata, skapa objektstrukturer eller driva testerna framåt i tiden finns det dock ett behov av att förenkla utnyttjandet av kärnan genom att införa diverse hjälpfunktionalitet. Detta skulle kunna ske genom att en eller möjligen flera ny bas klasser tas fram för tester av ovanstående typ. Vidare bör möjligheten för att visualisera interna klasstrukturer i Microsoft Visual Studio’s debugger undersökas.

Referenser

[1] OPNET Technologies, Inc. www.opnet.com.

[2] ns-2. www.isi.edu/nsnam/ns.

[3] Joint Communication Simulation System. www.disa.mil/jcss.

[4] The mathworks, Inc. www.mathworks.com.

[5] Microsoft Corporation. www.microsoft.com.

[6] GNU Compiler Collection. gcc.gnu.org.

[7] Boost. www.boost.org.