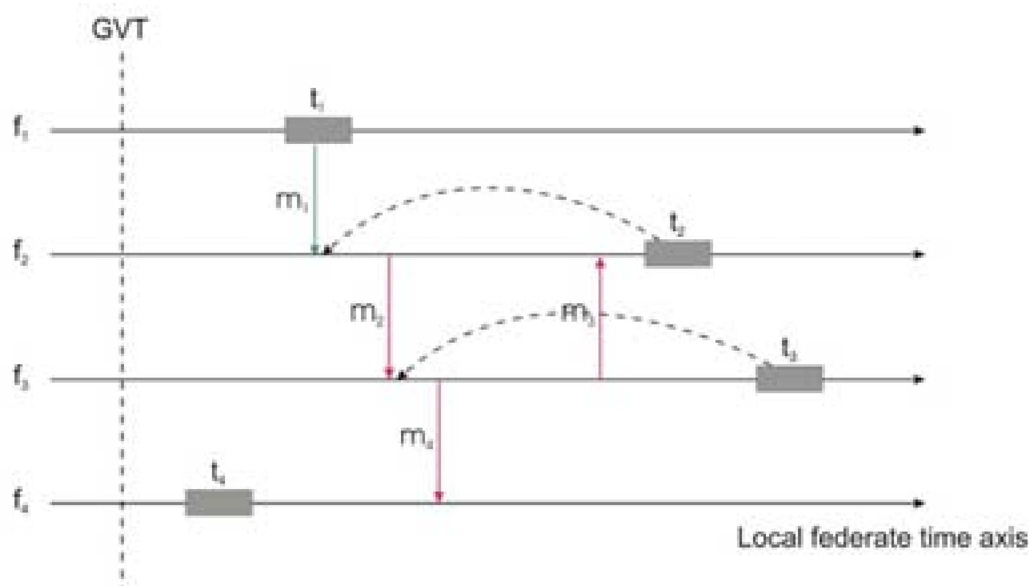
Ett mellanlager för effektiv hantering av tid och synkronisering i distribuerade simuleringar (HLA)

Jenny Ulriksson



FOI-R-- 1746 --SE
ISSN 1650-1942

Systemteknik
Metodrapport

Oktober 2005

FOI är en huvudsakligen uppdragsfinansierad myndighet under Försvarsdepartementet. Kärnverksamheten är forskning, metod- och teknikutveckling till nytta för försvar och säkerhet. Organisationen har cirka 1350 anställda varav ungefär 950 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, bedömningen av olika typer av hot, system för ledning och hantering av kriser, skydd mot hantering av farliga ämnen, IT-säkerhet och nya sensorers möjligheter.

FOI
Totalförsvarets forskningsinstitut
Systemteknik
164 90 Stockholm

Tel: 08-555 030 00
Fax: 08-555 031 00

www.foi.se

Ett mellanlager för effektiv hantering av tid och synkronisering i distribuerade simuleringar (HLA)

FOI-R--1746--SE
ISSN 1650-1942

Systemteknik
Metodrapport

Oktober 2005

Utgivare	Rapportnummer, ISRN	Klassificering
FOI - Totalförsvarets forskningsinstitut Systemteknik 164 90 Stockholm	FOI-R--1746--SE	Metodrapport
	Forskningsområde 2. Operationsanalys, modellering och simulering	
	Månad, år Oktober 2005	**Projektnummer** I6024
	Delområde 21 Modellering och simulering	
	Delområde 2	
Författare/redaktör Jenny Ulriksson	**Projektledare** Jenny Ulriksson	
	Godkänd av Monica Dahlén	
	Uppdragsgivare/kundbeteckning FOI	
	Tekniskt och/eller vetenskapligt ansvarig Jenny Ulriksson	

Rapportens titel

Ett mellanlager för effektiv hantering av tid och synkronisering i distribuerade simuleringar (HLA)

Sammanfattning (högst 200 ord)

Modellering och Simulering (M&S) utgör ett alltmer ovärderligt verktyg inom militära operationer och processer. Den standard som används för det idag inom Försvarsmakten är HLA (*the High Level Architecture*). HLA är ett ramverk för simuleringsutveckling och som tillhandahåller mjukvara och tjänster för distribuerad exekvering. Delvis på grund av sin stora komplexitet används inte ramverket till fullo. En av de tjänster som är särskilt komplexa att använda och där stöd kan behövas är tidhantering och synkronisering i HLA.

Denna rapport presenterar forskningen bakom, designen och utvecklingen av ett mellanlager till HLA, *HLAMiddleLayer*. Syftet med lagret är att separera tung HLA-logik från simuleringsutvecklingen för att på detta sätt underlätta utveckling i HLA och dessutom tillhandahålla stöd för mer avancerad användning av tjänster kring tidhantering. Synkroniseringsalgoritmer implementerade i HLA, såsom TimeWarp, framställs och presenteras i detalj. Experiment som har utförts avseende HLA-prestanda demonstreras och diskuteras, samt även experiment kring prestanda för de olika synkroniseringsalgoritmerna. Resultatet presenteras i en artikel som bifogas rapporten. Avslutningsvis diskuteras förslag till framtida vidareutveckling.

Nyckelord

Distribuerad simulering, Tidshantering, HLA, TimeWarp, Simuleringsutveckling, Synkronisering

Övriga bibliografiska uppgifter	Språk Svenska
ISSN 1650-1942	**Antal sidor:** 55 s.
Distribution enligt missiv	**Pris: Enligt prislista**

Issuing organization
FOI – Swedish Defence Research Agency
Systemteknik
164 90 Stockholm

Report number, ISRN
FOI-R--1746--SE

Report type
Methodology report

Programme Areas
2. Operational Research, Modelling and Simulation

Month year
Oktober 2005

Project no.
I6024

Subcategories
21 Modelling and Simulation

Subcategories 2

Author/s (editor/s)
Jenny Ulriksson

Project manager
Jenny Ulriksson

Approved by
Monica Dahlén

Sponsoring agency
FOI

Scientifically and technically responsible
Jenny Ulriksson

Report title (In translation)
A middlelayer for efficient management of time and synchronization in distributed simulations (HLA)

Abstract (not more than 200 words)

Modelling and simulation constitute an increasingly utilized tool within military operations and processes. The applied standard for M&S within the Swedish Armed Forces is *the High Level Architecture* (HLA). HLA is a framework for simulation development and that provides software and services for distributed execution. Partly due to its great complexity, the framework is unfortunately not fully utilized. One of the services that is particularly complex to use and where support can be necessary is time management and synchronization in HLA.

This report presents the research behind, the design and the development of a middlelayer for HLA, the *HLAMiddleLayer*. The target of developing the layer is to separate heavy HLA logic from simulation development, in order to facilitate utilization of HLA and in addition supply support for more advanced utilization of time management services. Synchronization algorithms implemented in HLA, such as TimeWarp, are developed and presented in detail. Experiments that have been conducted concerning HLA performance are demonstrated and discussed, as well as experiments concerning performance of the implemented synchronization algorithms. The result is presented in a paper that is attached. Finally proposals to further development are discussed.

Keywords
Distributed simulation, Time management, HLA, TimeWarp, Simulation development, Synchronization

Further bibliographic information

Language Swedish

ISSN 1650-1942

Pages 55 p.

Price acc. to pricelist

va 1.0. 2005-09-06

Innehållsförteckning

Sammanfattning 7

1 Introduktion 9

1.1 Bakgrund till projektet 9

1.2 Syfte 9

1.3 Nyttan för FM och omvärlden 9

1.4 Genomförande 10

1.5 Disposition 10

2 Distribuerad simulering 11

2.1 Bakgrund 11

2.2 Tidshantering 11

2.2.1 Tid i distribuerad simulering 11

2.2.2 Synkroniseringsalgoritmer och protokoll 12

2.2.3 TimeWarp 12

2.3 High Level Architecture (HLA) 13

2.3.1 Grunder i HLA 14

2.3.2 The HLA Runtime Infrastructure (RTI) 15

2.3.3 HLA Time Management 15

2.3.4 Icke-simuleringsrelaterad tillämpning av HLA 16

2.3.5 Avsaknader i HLA 16

2.4 Relaterat arbete 17

3 HLAMiddleLayer – Design och implementation 19

3.1 Krav 19

3.2 Design 19

3.3 Implementation 20

3.3.1 Tidshantering och synkroniseringsalgoritmer 21

3.3.2 Gränssnitt 21

3.3.3 TimeWarp 22

3.3.4 Testfederat med stöd för TimeWarp 23

3.4 Examensarbete inom projektet 24

4 Experiment och resultat 27

4.1 Prestandatest 27

4.1.1 Testplattform 27

4.1.2 Experiment 1 – Utvärdering av HLA jämfört med sockets 27

4.1.3 Experiment 2 – Utvärdering av olika synkroniseringsprotokoll 28

4.2 Experiment 3 – Praktisk utvärdering i befintlig applikation 29

4.2.1 NätSim-projektet och en redan befintlig applikation 29

4.2.2 Integrering och utvärdering 30
4.3 Summering av projektaktiviteter ... 30
5 Summering samt framtida arbete 31
5.1 Summering och slutsatser ... 31
5.2 Förslag till framtida arbete ... 31
6 Referenser 33
Appendix A – Förslag till examensarbete 35
Appendix B – Implementerade algoritmer i HLAMiddleLayer 39
Appendix C – Artikel 45

Sammanfattning

Modellering och simulering (M&S) utgör ett alltmer ovärderligt verktyg inom militära operationer och processer. En betydelsefull metod för M&S är distribuerad simulering, och den standard som används för detta idag inom Försvarsmakten (FM) är HLA (*the High Level Architecture*). HLA är ett ramverk för simuleringsutveckling och som tillhandahåller mjukvara och tjänster för distribuerad simulering. Trots att HLA stödjer avancerad utveckling och exekvering av simuleringar, så är ramverket komplext att använda och viss funktionalitet saknas eller stöds otillräckligt.

En särskilt krävande fråga är tidshantering och det finns ett tydligt behov av att tillhandahålla bättre stöd för HLA-baserad utveckling. En algoritm som är särskilt komplex är TimeWarp, som stöds bristfälligt av ramverket. Med en mjukvara som stödjer dessa tjänster kan förhoppningsvis mer komplex logik utnyttjas och återanvändas som dessutom kan varieras enkelt, samt mer kvalitativa simuleringsmodeller kan utvecklas. Därutöver underlättas användning och spridning av standarden.

Den här rapporten presenterar resultatet av ett projekt som har utförts på Institutionen för Systemmodellering, under tidsperioden våren 2004 till hösten 2005, finansierat av Innovationsrådet på FOI. De två huvudmålen med projektet har varit att dels implementera ett transparent mellanlager som tillhandahåller avancerad tidshantering i HLA för att komma tillrätta med problemen ovan, samt dels att formulera och sammanställa en algoritm för implementering av synkroniseringsprotokollet TimeWarp och specifikt i HLA. Rapporten presenterar även HLA och tidshantering i HLA i detalj, samt ger handledning i implementering av TimeWarp samt strikt synkronisering i HLA.

I arbetet har ett mellanlager för tidshantering i HLA, *HLAMiddleLayer* designats och utvecklats. Detta separerar HLA-specifik logik från simuleringsutvecklingen och bistår med stöd för tidshantering. Ett antal protokoll för synkronisering har implementerats i lagret och en algoritm för implementering av TimeWarp i HLA har formulerats och implementerats. De olika funktionerna kan användas genom att ange olika moder då lagret initieras och till dags dato har tre huvudmoder implementerats: *relaxed*, *strict* och *optimistic*. Moderna ger utvecklaren automatiskt stöd för att använda de funktioner för tidshantering som lagret tillhandahåller.

Designen som valdes är federatspecifik, dvs. varje federat är värd åt sitt eget lager istället för att utnyttja ett gemensamt lager mellan federaterna och RTI. Detta medför att lagret kan konfigureras efter federatens behov, samt att lösningen blir skalbar och mer generell än ett gemensamt lager skulle vara. Den design som utvecklades innebär att HLAMiddleLayer ärver HLA-klassen RTIAmbassador. Detta för federaten på ett smidigt sätt får tillgång till all logik som mellanlagret erbjuder, samtidigt som federaten kan använda RTI-funktionalitet på dess ursprungliga sätt, vilket visade sig vara en fördelaktig lösning.

För att utvärdera nyttan av lagret genomfördes tre olika experiment. Det första var en praktisk utvärdering där HLAMiddleLayer integrerades i en befintlig applikation, detta för att utvärdera användarvänligheten samt vilken extra funktionalitet som lagret kunde tänkas tillhandahålla. Dessutom utfördes två prestandaexperiment, varav det första utgjordes av en utvärdering av RTI som kommunikationsarkitektur jämfört med ett system enbart baserat på sockets. Därefter gjordes en utvärdering av prestanda för olika synkroniseringsprotokoll, för att kunna utgöra underlag för utvecklare vid val av tidshanteringsmod.

Resultatet visade att implementeringen av ett lager såsom HLAMiddleLayer är fördelaktigt och en slutsats var att mer liknande stöd för HLA-utveckling borde finnas. Prestandatesterna visade att RTI fungerar dugligt som inte bara simuleringsarkitektur, utan generellt som kommunikationsarkitektur för distribuerade system. Testerna för synkroniseringsmoderna gav bra underlag till val av algoritm för utveckling och tidshantering. Dessutom identifierades bristen av ett HLA-forum, såväl internationellt som nationellt, och ett förslag ges till att ett sådant skapas, åtminstone inom Försvarsmakten. Slutligen presenterades även delar av resultatet i en artikel som antogs för presentation på en konferens och som ska presenteras under hösten.

1 Introduktion

Den här rapporten presenterar resultatet av ett projekt som har utförts på Institutionen för Systemmodellering. Projektet finansierades av Innovationsrådet på FOI och genomfördes från våren 2004 till hösten 2005.

1.1 Bakgrund till projektet

Modellering och simulering (M&S) utgör ett alltmer ovärderligt verktyg inom militära operationer och processer. Det används för militär träning, övning, för logistikplanering, utvärdering av utförda operationer, utveckling av nya system och som beslutsstöd. En betydelsefull metod för M&S är distribuerad simulering (DS) DS möjliggör för simuleringar att delas upp i och utvecklas som fristående komponenter som exekveras tillsammans och distribuerat, snarare än att utvecklas som enskilda resurskrävande simuleringar. Detta gör att simuleringar enklare kan återanvändas och valideras, samt att de kan exekveras mer effektivt. En nödvändig faktor för denna typ av M&S är standarder, för att bl.a. stödja interoperabilitet och säkring av kvalitet.

Den simuleringsstandard som idag används inom Försvarsmakten (FM) är HLA (*the High Level Architecture*). HLA är ett ramverk för simuleringsutveckling och som tillhandahåller mjukvara och tjänster för distribuerad exekvering. Trots att HLA stödjer avancerad utveckling och exekvering av simuleringar, så är ramverket komplext att använda och viss funktionalitet saknas eller stöds bristfälligt. Detta gör att många utvecklare undviker HLA, vilket medför bland annat mindre återanvändbarhet och samarbete inom FM och med andra organisationer. Detta hindrar även spridningen av standarden till den civila marknaden. Dessutom är vissa HLA-tjänster så komplexa att utvecklare skyr HLA-logiken och implementerar enklare alternativ, istället för de metoder som är mest lämpade för simuleringen. En särskilt krävande fråga är tidshantering. För detta tillhandahåller HLA mycket stöd, men fordrar även kunskap från utvecklaren. En mycket vanlig metod för synkronisering är TimeWarp, ett synkroniseringsprotokoll som visserligen stöds av HLA, men bristfälligt. Resultatet blir att utvecklare implementerar sina egna versioner av algoritmen, utan att verifiera funktionaliteten och utan att återanvända redan implementerade lösningar.

Det finns ett tydligt behov av att separera tung HLA-logik från själva simuleringen och av att tillhandahålla bättre stöd för utveckling baserad på HLA. Några av de tjänster som fördelaktigt skulle kunna stödjas är tidshantering och implementerade algoritmer för synkronisering. Med en mjukvara som stödjer dessa tjänster kan förhoppningsvis mer komplex logik utnyttjas och återanvändas, och mer kvalitativa simuleringsmodeller utvecklas. Dessutom underlättas användning och spridning av standarden.

1.2 Syfte

Detta projekt ämnar studera funktionaliteten i HLA och tjänsterna som implementeras enligt HLA-specifikationen, med särskilt fokus på tidshantering och synkroniseringsalgoritmer. En av de algoritmer som kommer att studeras särskilt, och som idag inte helt stöds av ramverket, är TimeWarp. Ett mellanlager, *HLAMiddleLayer*, som transparent för utvecklaren tillhandahåller varierande former av tidshantering och synkronisering kommer att utvecklas. Syftet med mellanlagret är att separera HLA-specifik logik från utvecklingen, och främst med fokus på att stödja varierande mekanismer för tidshantering och synkronisering. Dessutom ges utvecklaren möjligheter att utnyttja avancerad tidshantering, utan att själv besitta sådan kunskap och utan att behöva identifiera hur just den funktionaliteten kan implementeras med hjälp av HLA. Detta underlättar för simuleringsutvecklare att använda den typ av tidshantering som är mest lämpad för simuleringen.

1.3 Nyttan för FM och omvärlden

Sedan ett par år tillbaka är HLA anmodad försvarsstandard för M&S. Trots detta är fortfarande år 2005 långt ifrån alla simuleringsmodeller inom FM baserade på HLA. En av anledningarna till detta kan vara att HLA-standarden är omfattande och komplex, vilket gör den alltför tidskrävande för att projekt ska anamma standarden. Det som behövs är utvecklingsstöd och mjukvara för att underlätta användningen

av HLA. Detta skulle sprida användningen av standarden, samt medföra att återanvändningen av försvarsrelaterad simulering ökar betydligt. Dessutom skulle samarbete med industrin främjas om båda parter utnyttjar standarden. Slutligen kan med en gemensam standard och återanvändbar simuleringslogik kvalitén hos simuleringen bättre säkerställas och processer som Validering och Verifiering (VV) underlättas och möjliggöras.

1.4 Genomförande

Projektet delades huvudsakligen upp i fyra aktiviteter:

- *Forskning:* Forskningen i projektet behandlade HLA och undersökning av relaterade aktiviteter i omvärlden. Särskilt fokus var algoritmen för TimeWarp och hur denna skulle kunna implementeras i HLA. I arbetet ingick även att producera en artikel för en lämplig konferens.

- *Utveckling:* En testmiljö sattes upp som ett mindre datorlabb och med ett antal lämpliga testfederater som skapades. Mellanlagret implementerades med tre synkroniseringsmekanismer och delvis i samarbete med ett examensarbete för 2 personer som utformades inom ramarna för projektet.

- *Experiment:* Ett antal olika experimentsessioner genomfördes för mellanlagret både under utveckling och av slutprodukten, och delvis i samarbete med NätSim-projektet. Resultatet gav input till vidareutveckling och nyutveckling av lagret, samt utvärdering av slutresultatet. Dessutom påvisades brister i HLA och RTI.

- *Samarbete:* Samarbete med NätSim-projektet initierades och mellanlagret integrerades i en av projektets applikationer. Detta för att kunna tillämpa mellanlagret praktiskt och ge feedback i form av praktisk utvärdering och förslag till vidareutveckling.

1.5 Disposition

Kapitel 2: För en läsare som är ny inom området ger kapitel 2 en bakgrund till området distribuerad simulering samt särskilt ämnet tidshantering. M&S-standarden HLA presenteras även detaljerat, samt relaterat arbete i omvärlden.

Kapitel 3: Detta kapitel presenterar designen av det mellanlager, HLAMiddleLayer, som har implementerats, samt även implementering och specifikt avseende de synkroniseringsalgoritmer som har utnyttjats. Algoritmen TimeWarp är särskilt prioriterad.

Kapitel 4: Detta kapitel presenterar konkreta implementeringsfrågor, testbädd för experiment, samt vilka test som har utförts. Därefter diskuteras testresultat och slutsatser.

Kapitel 5: För den läsare som endast vill orientera sig i resultatet av arbetet, bidrar Kapitel 5 med slutsatser och diskussion kring arbetet som avhandlas i denna rapport. Dessutom diskuteras förslag framtida arbete.

2 Distribuerad simulering

En viktig metod för M&S är konceptet distribuerad simulering. Detta kapitel presenterar grunderna i konceptet och ämnet tidshantering ingående. Dessutom gås HLA igenom detaljerat.

2.1 Bakgrund

Distribuerad simulering (DS) innebär att en simulering delas upp i delkomponenter som sig kan exekveras på enskilda noder distribuerade över ett nätverk [Fujimoto 2000]. Delkomponenterna koordineras för att tillsammans utgöra en simulering. Detta möjliggör för processer och simuleringar att utnyttja datorkapacitet från flera datorer i ett nätverk, vilket exempelvis tillåter även mycket tunga processer att exekveras. Dessutom tillåter det att fasta simuleringar, såsom simulatorer, kan exekveras tillsammans med andra simuleringar och nätverk. En besläktad metod är *parallell simulering*, där istället komponenterna exekveras på samma dator och processorkraften härmed utnyttjas på bästa sätt. Hastigheten hos exekveringen kan härmed ökas betydligt, s.k. *speed-up*, vilket möjliggör för exekvering av mycket tidskrävande processer, såsom vädersimuleringar där mycket data behandlas. Att på detta sätt dela upp en simulering i delkomponenter och koordinera dem över ett nätverk kräver gemensamma gränssnitt och gemensamma standarder. Detta för att komponenterna dels ska kunna kommunicera, dels för att bistå med eventuella nödvändiga tjänster såsom tidshantering och koordination av kommunikationen.

2.2 Tidshantering

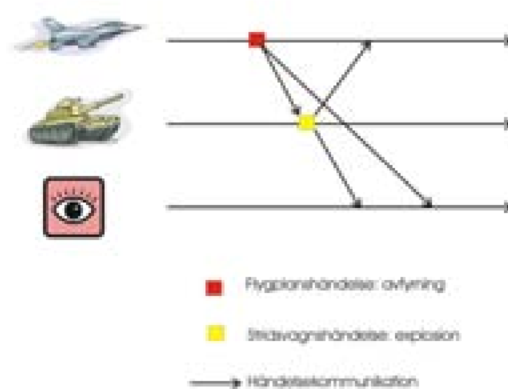
2.2.1 Tid i distribuerad simulering

En av de viktiga beståndsdelarna i distribuerad simulering är tiden och hanteringen av tid. Tidshantering i DS är tjänsten som ser till att exekveringen av distribuerade simuleringar är korrekt synkroniserad. *Tid* i distribuerad simulering delas ofta upp i två typer, *Lokal Tid* (LT) som är den lokala tiden för simuleringen, samt *Global Virtuell Tid* (GVT) som utgörs av den tid som simuleringen gemensamt har kommit fram till och som LT synkroniseras mot. Mekanismerna för tidshantering kan även utnyttjas för att garantera att händelser i en simulering tas emot och skickas i händelseordning. På detta sätt kan det säkerställas att simuleringar som mottar exakt likadan input även producerar exakt samma resultat. Synkroniseringen kan ske med hjälp av olika algoritmer, såsom presenteras i Avsnitt 3.2.2. Som ett exempel på hur tiden kan ha en avgörande betydelse ges ett exempel nedan, Exempel 1.

Exempel 1:

Antag att ett scenario består av ett spaningsplan som söker efter en stridsvagn. En observatör beskådar simuleringen (representerad med ett öga i figuren bredvid). De två simuleringskomponenterna har varsin tidsaxel för sin lokala tid. Observatörens tidsaxel motsvarar realtid.

Då flygplanet ser stridsvagnen avfyrar den en missil (händelse markerad med en röd fyrkant i figuren). När stridsvagnen mottar denna händelse genererar den i sin tur en händelse som motsvarar en explosion eftersom den har blivit träffad. Spaningsplanet mottar händelsen och vet härmed att operationen är lyckad. Händelsen från stridsvagnen har däremot försenats på sin väg till observatören, vilket gör att det som presenteras för observatören är först hur en explosion uppstår då stridsvagnen träffas, *därefter* avfyrar planet sin missil!



Detta ologiska händelseschema hade kunnat undvikas med mekanismer för tidshantering.

2.2.2 Synkroniseringsalgoritmer och protokoll

Synkronisering i DS avser mekanismerna för att koordinera de distribuerade processerna. De regler som gäller för en viss mekanism beskrivs av en särskild algoritm och implementeringen av en algoritm utgör ett protokoll. Olika protokoll medför olika korrekthet vad gäller ordning av händelser samt ordning av tidstämplar. De olika algoritmerna medför även olika mycket koordination, vilket leder till olika overhead och tidsfördröjningar som just den synkroniseringsmekanismen medför. Overhead och förseningar kan göra simuleringen ineffektiv eller till och med ogenomförbar. Därför är synkroniseringen av stor vikt för många simuleringar, särskilt de där människor medverkar, eftersom en människa som interagerar med systemet inte accepterar ologiska hack, förskjutningar eller fördröjningar i simuleringen. Såsom inses lätt behöver därför olika synkroniseringsmekanismer användas vid olika tillfällen, och det måste alltså finnas stöd för att kunna variera dessa, något som stöds dåligt i existerande simuleringssystem och –arkitekturer.

Algoritmer för synkronisering delas traditionellt in i två typer; *konservativa* och *optimistiska* [Fujimoto 1998]. Den konservativa är enkel att implementera, men innebär ofta tidsfördröjningar och overhead. Den optimistiska är i många fall mer effektiv, men komplicerad och har andra nackdelar.

Konservativa algoritmer uppfyller kravet att inga distribuerade processer, även kallade logiska processer (LP), tar emot information från någon annan LP som har en tid som är mindre än den tid som den mottagande LPn har. Dvs. om *m* motsvarar ett specifikt meddelande så gäller att de meddelanden som kan tas emot har en tidstämpel med: $t_m > t_{LP}$. Det stora problemet med konservativa protokoll är därför att undvika *deadlocks* och att garantera att simuleringen fortlöper stabilt framåt. *Deadlock* är när exekveringen fastnar och inte kan avancera tiden framåt. Det kan uppstå om t.ex. samtliga processer innehar exakt samma lokala tidpunkt, och således inte kan motta något meddelande från en annan LP och alltså inte heller kan avancera tiden framåt. När tiden avanceras fram kontinuerligt i lika stora steg benämns synkroniseringen tidsstegad, eller synkron. Synkrona protokoll ser således till att alla distribuerade simuleringskomponenter precis hela tiden är korrekt synkroniserade, en metod som kräver mycket overhead.

Till skillnad från konservativa algoritmer avancerar optimistiska algoritmer tid och händelser utan respekt till andra enheter och med risken att något kan gå fel. Detta kräver att algoritmen måste bistå med mekanismer för att kunna upptäcka och korrigera (återställa) för fel. Ett felaktigt meddelande är till exempel om händelseordningen mellan meddelanden inte stämmer, eller om en process upptäcker att tidstämpeln på ett mottaget meddelande är en tid som den LPn redan har passerat.

Två olika typer av händelser kan behöva bli korrigerade:

1. det felaktiga meddelandet har förändrat status hos processen, vilket måste åtgärdas
2. processen kan som resultat av det felaktiga meddelandet ha skickat ytterligare meddelanden till andra processer. Mottagande federater måste därför notifieras om att meddelandena var felaktiga.

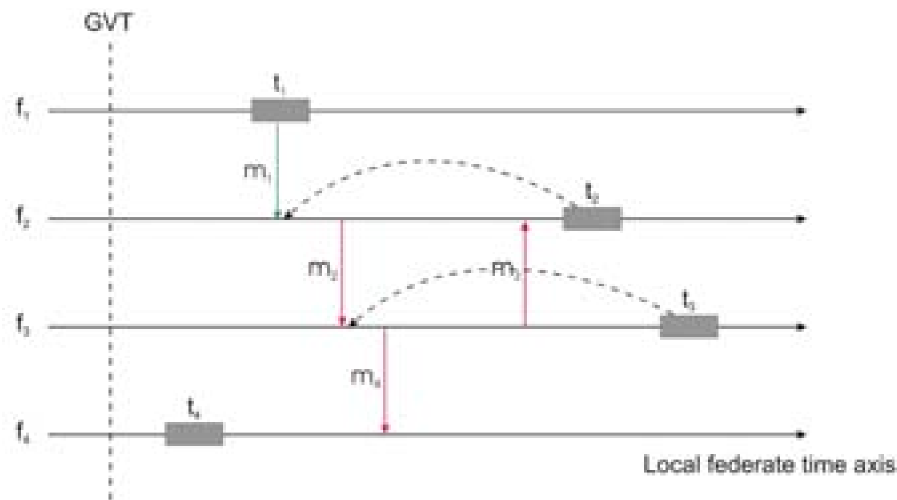
2.2.3 TimeWarp

Jefferson var den första att formulera ett protokoll för optimistisk tidssynkronisering. Han kallade det *TimeWarp* (TW) och är det mest kända protokollet för denna typ av tidshantering. Jefferson beskriver att om en LP mottar ett meddelande från en föregångare som har en tidstämpel mindre än den lokala klockan, här kallad Lo*cal Virtual Time* (LVT), rollar processen tillbaka i simuleringstid och gör simuleringen ogjord tillbaka till den tid i simuleringens förgångna som den rollar tillbaka till. Ett sådant ”försenat” meddelande kallas *straggler*, och i och med att processen rollar tillbaka kan den ta emot och processa straggler-meddelandet på den tidpunkt det egentligen var tänkt. Rollback kräver periodisk lagring av processernas status. Dessa perioder bestäms gentemot den globala tiden GVT (*Global Virtual Time*), som är ett minimum av alla LVTs. Att göra ett meddelande osänt görs genom att skicka ett s.k. negativt meddelande, ett anti-meddelande.

TW-protokollet har varit ett ämne för intensiv forskning inom simuleringsvärlden och åtskilliga optimeringsansatser av protokollet har skapats. En av de faktorer som har ägnats mest tid åt är hur

perioderna ska väljas för lagring av processers status. Om perioderna är för stora skapas icke-hanterbara mängder data, medan om de är för små kan inte processerna rolla tillbaka längre än tiden för den perioden.

Nedan presenteras ett exempel på en TimeWarp, Exempel 2.



Exempel 2:

I figuren representeras meddelanden av m, de som är röda visar antimeddelanden och de som är gröna är ursprungliga meddelanden. f representerar en federat och således finns det fyra olika federater med i figuren ovan, där deras respektive tidsaxlar ses. De svarta streckade pilarna motsvarar rollback-perioden. I exemplet är GVT = 12; $t_1 = 16$; $t_2 = 20$; $t_3 = 24$ samt $t_4 = 14$.

Händelseförlopp:

m_1: f_2 mottar ett meddelande från federat f_1 med tidstämpel: $t_{m1} = 16 < t_2 = 24$. Detta tvingar f_2 att rolla tillbaka till $t_{rollback\ f2} = 16$. Rollback!

m_2: Under rollback-perioden för f_2 har federaten ett meddelande att göra ogjort: m_2 med $t_{m2} = 17$. Den skickar därför ett antimeddelande, m_2, till respektive federat, f_3. Detta påtvingar i sin tur f_3 att rolla tillbaka till tiden för m_2: $t_{m2} \Rightarrow t_{rollback\ f3} = 17$. Rollback!

m_3: Under rollback-perioden för f_3 måste två meddelanden göras ogjorda och två antimeddelanden skickas. Det första är m_3 med tiden $t_{m3} = 20$ som skickas till federat f_2. Men eftersom trollback $t_{m3} > t_{rollbackf2}$ blir antimeddelandet m_3 ignorerat av f_2, antimeddelandet gäller ju ett meddelande som f_2 har raderat. Ingen rollback!

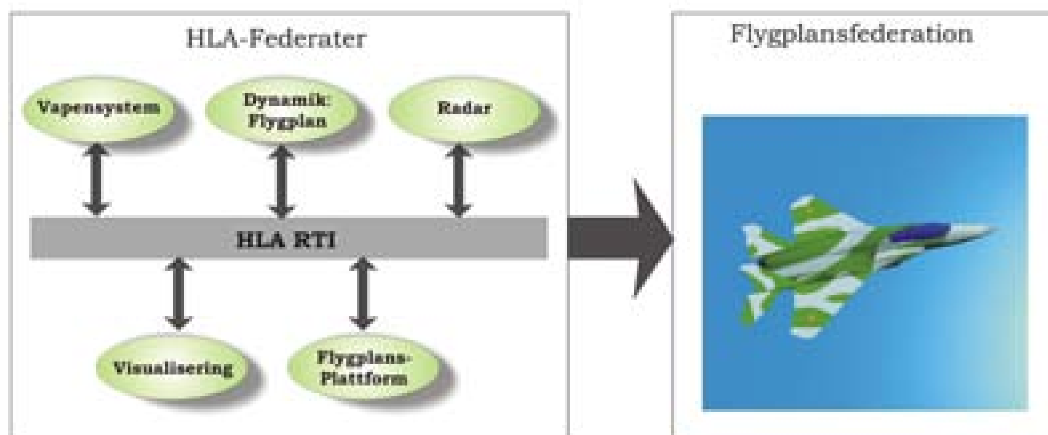
m_4: Det andra meddelandet som f_3 måste göra ogjort är m_4 och för tiden $t_{m4} = 19$. Eftersom $t_4 < t_{m4}$ har inte f_4 processat det meddelandet än, utan raderar originalmeddelandet från sin inkö med meddanden. Ingen rollback!

2.3 High Level Architecture (HLA)

I mitten av 80-talet uppdagades ett behov av att standardisera kommunikationen mellan geografiskt spridda simulatorer inom amerikanska försvaret. Man insåg att för att kunna utnyttja simulatorerna, och även andra simuleringar, tillsammans istället för enbart separat på deras originella platser, behövdes gemensamma gränssnitt mellan simuleringskomponenterna. Som ett resultat av flera års forskning utvecklade amerikanska *Defence Modelling and Simulation* Office (DMSO) den första standarden för syftet, *Distributed Interactive Simulation* (DIS) [Kuhl et al.]. DIS tillfredsställde de krav som ställdes för det specifika syftet, men det var utvecklat för realtidssimulering och dessutom eftersöktes en standard som stödde komponentbaserad utveckling på ett bättre sätt. Arbetet fortsatte därför och en vidareutveckling och förbättring av DIS kom 1995, the High Level Architecture (HLA). HLA IEEE-standardiserades 2000 (IEEE 1516) och vidareutvecklas än idag, men trots det finns DIS kvar och används i simuleringar där DIS är mer lämplig.

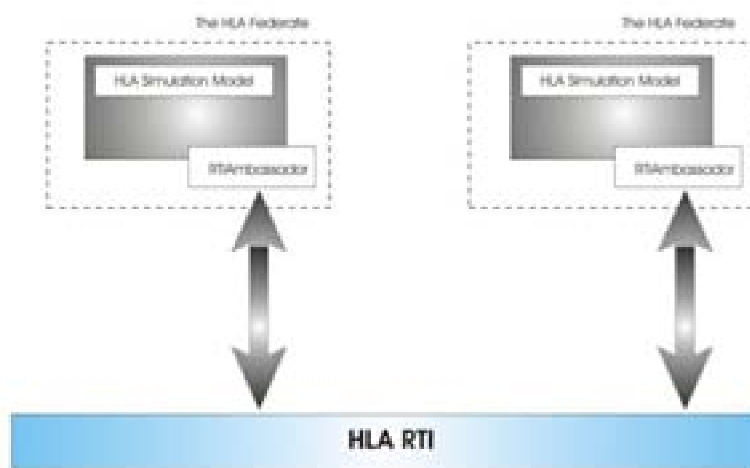
2.3.1 Grunder i HLA

HLA är ett ramverk för distribuerad simulering, med regler och mjukvara för att stödja komponentbaserad simuleringsutveckling och –exekvering. Tekniken medför att återanvändbara simuleringskomponenter kan skapas, i HLA så kallade *federater*, som kan sättas samman i konstellationer de ursprungligen inte var ämnade för, se figur 2.1. Detta innebär att federaterna till exempel kan lagras i bibliotek och delas mellan organisationer, för att kunna återanvändas istället för kostsam nyutveckling. Federaterna kan i sig vara utvecklade i olika språk och i olika miljöer och HLA-gränssnittet möjliggör interoperabilitet mellan dem. Det som sammanbinder federaterna till en simulering, i HLA en så kallad *federation*, är en implementering av de tjänster som HLA specificerar; *HLA Interface Specification*. Denna implementering kallas HLA *Runtime Infastructure* (RTI) och är beskriven vidare i Avsnitt 3.3.2.



Figur 2.1: *Återanvändbara HLA-komponenter sätts samman i en flygplanssimulering. Flygplanet kan i sig vara del i en större simulering.*

För att en federat ska kunna kommunicera med andra federater och kunna erhålla information och tjänster från RTI, måste federaten ha en lokal så kallad *RTIAmbassador*. Ambassadören utnyttjas för att ta emot och sända meddelanden och håller även reda på specifik vilken information som federaten är intresserad av att få veta från resten av simuleringen (dvs. det den *prenumererar på*) och vilken information som federaten kan publicera. Ambassadören är således federatens handtag ut i simuleringen, och det är egentligen de olika komponenternas ambassadörer som kommunicerar med varandra, se Figur 2.2.



Figur 2.2: RTIAmbassador *möjliggör för kommunikation med* RTI, *det är egentligen den som "utgör" själva HLA-federaten, det vill säga det är den som är "synlig" för federationen.*

2.3.2 The HLA Runtime Infrastructure (RTI)

RTI är den mjukvara som stödjer ihopkoppling av federater samt kommunikationen mellan dem. RTI är ett distribuerat operativsystem som förutom att det tillhandahåller ett gränssnitt mellan federaterna, även bistår med avancerade tjänster för distribuerad simulering. De sex huvudsakliga tjänster som stöds är:

- *Federation management*: Funktionalitet för hantering av federationer. Detta inkluderar till exempel skapande, modifiering och borttagning av simuleringar.
- *Object management*: Bistår med tjänster för federater att skapa, modifiera och radera objekt och interaktioner.
- *Declaration management*: Denna tjänst tillåter federater att registrera sina intressen, dvs. att prenumerera på den information de är intresserad av att bli uppdaterade med och registrera vilken information de själva kan bistå med.
- *Time management*: Avancerad tidshantering och tjänster för att koordinera händelser mellan federater.
- *Ownership management*: Denna tjänst gör det möjligt för federater att utbyta äganderätten till objektattribut mm.
- *Data distribution management*: Tjänsten förkortas DDM och är en mekanism för att effektivt transportera och filtrera information mellan federater.

RTI tillhandahålls från flera olika leverantörer. Som exempel kan nämnas Mäk RTI från Mäk Technologies[1] samt pRTI från svenska företaget Pitch[2]. De olika systemen är mer eller mindre distribuerade. I pRTI sker till exempel all initiering centralt, varefter kommunikationen kan ske direkt mellan federaternas RTIAmbassador

2.3.3 HLA Time Management

Tjänsterna för tidshantering i HLA, *HLA Time Management services* (HLA TM), utnyttjas för att kontrollera avancemanget av tid under exekveringen av en federation, och inkluderar andra mekanismer för att för att garantera transporten av meddelanden. Detta innebär att HLA TM omfattar:

- Transporttjänster, dvs. Ordning av meddelanden, tillförlitlighet vad gäller transport av meddelanden etc.
- Olika typer av tjänster för tidsavancering

RTI tillhandahåller huvudsakligen två typer av policys för tid: *time-regulating* (TR) och *time-constrained* (TC) [Vardanega & Maziero]. Om en federat är TR producerar den *time-regulating* (tidsreglerande) händelser, och om en federat är TC konsumerar den *time-stamped* (tidsstämplade) händelser, det vill säga händelser som har producerats av TR-federater. Avancemanget av en federats LT sker explicit genom en förfrågan från federaten till RTI och när RTI har godkänt avancemanget, detta görs genom ett anrop till metoden *timeAdvanceGrant*.

HLA stödjer interoperabilitet mellan federater som har olika mekanismer för tidshantering. Detta betyder att federater kan kommunicera trots att de har olika krav på händelseordning, såsom till exempel att de är händelsedrivna eller tidsstegade, eller använder olika eller en blandning av transporttjänster, såsom reliable eller unreliable kommunikation [Fujimoto 1998].

---

[1] Mäk hemsida. Nås via: http://www.mak.com. Senast besökt augusti 2005.

[2] Pitch hemsida. Nås via: http://www.pitch.se. Senast besökt augusti 2005.

2.3.4 Icke-simuleringsrelaterad tillämpning av HLA

Sedan HLA blev en IEEE-standard 2000 har användningen av HLA sträckt sig till den civila marknaden och även till applikationsområden utanför den militära domänen. Nedan presenteras tre exempel på icke-militär, icke-simuleringsbaserad tillämpning av HLA:

- *Virtuellt shoppingcenter*: Shen et al. har implementerat ett virtuellt shoppingcenter för datorbaserad samverkan [Shen et al.]. Användare av 3D-miljön representeras av virtuella 3D-personligheter, så kallade *avatars*, som är implementerade som HLA-federater. Shoppingsessionerna hanteras som HLA-federationer och kommunikationen går genom HLA RTI.

- *Distribuerad interaktiv träningsmiljö*: För att kunna koppla kunder och experter närmare utvecklade Blümel et al. en distribuerad virtuell träningsmiljö i 3D [Blümel et al.]. De baserade implementeringen på HLA och en av anledningarna till att de valde just denna arkitektur var att HLA till skillnad från andra traditionella distribuerade teknologier, såsom t.ex. CORBA[3], tillhandahåller avancerad tidshantering.

- *Online multiplayer-spel*: Det av dessa tre mest mogna exemplet av HLA-tillämpning är Vuong et al., som har implementerat ett ramverk för multiplayer online-spel som baseras på HLA [Vuong et al.]. Ramverket tillhandahåller server-baserade spel-lounger som spelare kan logga in på och hämta spel i, och de kan använda plattformar såsom personliga datorer, handdatorer och även mobiltelefoner. Klientapplikationerna har delvis implementerats som HLA-federater och spelsessionerna representeras delvis av HLA-federationer.

2.3.5 Avsaknader i HLA

Under åren har ett flertal brister och avsaknader identifierats i samband med HLA och RTI. Några av problemen med RTI har adresserats till varierande prestanda och icke-effektiv implementation av tjänster. Författaren till denna rapport deltog t.ex. i implementeringen av en lösning för att stödja hierarkisk utveckling av HLA-federationer för att lösa vissa av de tjänster HLA inte stödjer [Ulriksson et al.]. När bristerna i HLA diskuteras är det dock viktigt att komma ihåg att själva *implementeringen* av de tjänster som HLA specificerar, RTI, skiljer mellan olika leverantörer. Även om de följer specifikationen kan de således presentera vitt skild prestanda, varför konceptet HLA inte bör utdömas enbart på grund av lågpresterande implementeringar av RTI.

En annan brist med HLA är att HLA-specifikationen i vissa fall inte är heltäckande. Vissa funktioner som implementeras och följer specifikationen fullt ut och korrekt, kan ändå uppvisa brister, något som diskuteras av Vuong et al. där några specifika brister kring *HLA Object Management* presenteras och åtgärdas [Vuong et al.]. Ett relaterat ämne som flera har ifrågasatt är det koncept som HLA tillhandahåller för komponentbaserad utveckling. Radeski et al. är några att ifrågasätta detta och de presenterar ett förslag till en utökning av HLA för att komplettera de brister som identifierades [Radeski et al.].

En tjänst som särskilt har ifrågasatts är tidshanteringen i HLA. Trots att HLA tillhandahåller betydligt mer avancerad tidshantering än många andra simuleringsramverk är många av funktionerna implementerade på en mycket låg nivå och svåra att använda. Problemet har identifierats av flera, t.ex. [Vardanega & Maziero], [Yan et al.] och [Huang et al.], och särskilt har algoritmen TimeWarp utpekats som dåligt understödd. Såsom beskrevs i Avsnitt 2.2.2 skapas simuleringar för många olika syften, varvid olika simuleringar ställer olika krav på simuleringsprotokollet som används, och det är inte helt lätt att variera mellan dessa i HLA idag. Det direkta stöd HLA ger för synkronisering är tidsstegad synkronisering, men utvecklare bör ha bättre utvecklingsstöd även för mer avancerade former av synkronisering.

---

[3] CORBA, *Common Object Request Broker Architecture*, är en arkitektur för att hantera distribuerade program och objekt i nätverk. Det utvecklades i ett konsortium genom Object Management Group (OMG) och används över hela världen i distribuerade system.

2.4 Relaterat arbete

Eftersom tidshantering har identifierats av flera som en svår fråga, finns det en del aktiviteter som har utförts inom området.

Turner et al. implementerade till exempel ett mellanlager för att tillägga ytterligare en mekanism, en optimerad sådan, för händelsehantering [Turner et al.]. Mellanlagret var lovande men till författarens kännedom inte utbyggbart och utvecklas inte vidare. Dessutom skapades det som så många andra lösningar enbart för den enda orsaken som diskuteras i artikeln.

Ett annat arbete, och som direkt relaterar till implementering av TimeWarp i HLA, är Yan et al. som presenterar ett helt koncept för att implementera TimeWarp i HLA [Yan et al.]. De implementerade ingenting och inget fortsatt arbete har setts till, men några som däremot utvecklade något på riktigt var Vardanega och Maziero [Vardanega & Maziero]. De föreslog vad de kallar en *"Rollback Manager"* som ska hantera frågeställningar relaterade till just TimeWarp och optimistisk tidshantering. En prototyp till konceptet utvecklades, men sedan dess har inget nytt om det publicerats. "Rollback Manager" var dessutom strikt reglerad till att hantera TimeWarp och på ett begränsat sätt.

Det kanske mest relaterade arbetet till forskningen i denna rapport är det mellanlager som har implementerats av [Huang et al.]. Detta utvecklades som en utökning till HLA och för att tillhandahålla simuleringsutvecklare med ett enat gränssnitt för tidshantering mot HLA. Det tillhandahåller både stöd för konservativ, synkron och optimistisk tidshantering. Men även om arbetet verkar omfattande är det svårt att utröna hur mycket som verkligen är implementerat, och hur mycket som är koncept. Författaren till denna rapport har tagit kontakt med forskargruppen och de är intresserade av ett eventuellt samarbete om vidareutveckling, men verkar inte ha producerat någon konkret produkt själva.

3 HLAMiddleLayer – Design och implementation

De två huvudmålen med projektet är dels att implementera ett transparent lager för utvecklare av HLA-simuleringar med stöd för avancerade synkroniseringsalgoritmer, och dels att formulera och sammanställa en algoritm för implementering av TimeWarp i HLA. Detta kapitel beskriver kraven på lagret, den generella designen som utformades, TimeWarp-algoritmen som utvecklades, samt de implementeringsval som gjordes.

3.1 Krav

De största kraven på lagret avsåg användarvänlighet och att avlasta utvecklaren från komplex logik och sammanfattas i Tabell 3.1:

Tabell 3.1: *De övergripande kraven på HLAMiddleLayer.*

Krav	Beskrivning
Användartransparens	HLAMiddleLayer ska utgöra ett transparent mellanlager för användaren, det vill säga användaren ska inte tro eller belastas av att det finns ett hinder mellan simuleringen och RTI. Detta innebär att utvecklaren ska kunna använda även ursprunglig HLA-funktionalitet.
RTI-transparens	Samtliga funktioner som *RTIAmbassador* tillhandahåller måste kunna användas såsom ursprungligen tänkt, detta som ett led mot målet att lagret ska vara transparent. En utvecklare måste kunna välja om den vill använda lagerspecifika funktioner eller traditionella HLA-metoder, ett val som inte tillhandahålls av till exempel det lager som utvecklades av [Huang et al.].
Synkroniseringsalgoritmer	Lagret ska tillhandahålla ett flertal synkroniseringsalgoritmer som enkelt går att ändra med hjälp av olika moder i lagret. Dessa ska även kunna ändras under exekveringens förlopp.
Enlighet med HLA	Mjukvaran som utvecklas ska följa HLA-specifikationerna. Detta innebär att RTI inte får modifieras.
Interoperabilitet	Federater som använder mjukvaran måste vara interoperabla med federater som inte använder den. Det vill säga det ska vara transparent för andra federater att en federat använder HLAMiddleLayer.
Overhead	Användning av HLAMiddleLayer ska inte införa onödig overhead i exekveringen, det vill säga onödig exekveringstid eller ökad mängd kommunikation för de federater som använder lagret.
Dokumentation	Lagret ska bistå med lämplig dokumentation och användarhandledning för att göra det användarvänligt.

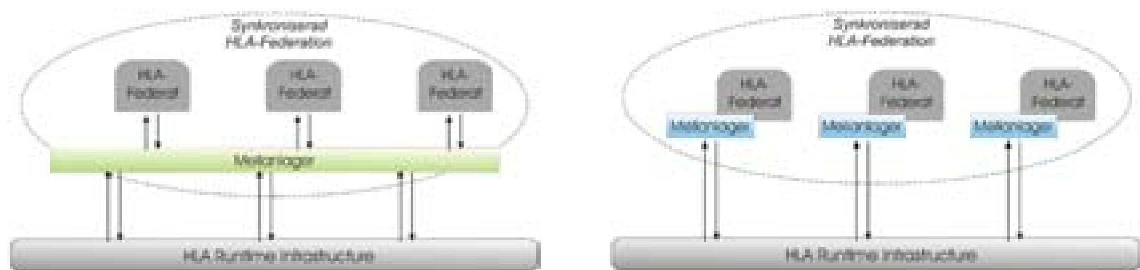
3.2 Design

Med grund i kraven för mellanlagret uppkom tidigt i designfasen ett vägval. Valet gällde om lagret skulle implementeras som ett mellanlager ovanför RTI, bilden till vänster i Figur 3.1 nedan, eller det skulle implementeras som en separat komponent som varje federat ensam var värd till, högra bilden i Figur 3.1.

Alternativ 1: Ett gemensamt mellanlager är enklare att implementera och kan stödja funktioner för hela simuleringen och även hålla reda på federationsgemensamma faktorer såsom meddelandeköer etc.

Dessutom blir det mer transparent för federaterna, då lagret på detta sätt skulle kunna implementeras så att minsta möjliga modifikation till den ursprungliga federaten behöver genomföras. Då ett av kraven är att implementeringen inte får inkräkta på HLA-specifikationen, dvs. RTI får inte påverkas, måste mellanlagret utvecklas som en separat komponent mellan RTI och federaterna. Detta tvingar kommunikationen mellan federater och RTI att gå en extra väg genom mellanlagret, vilket totalt sett ökar kommunikationen i systemet, jämför bilderna i Figur 3.1. Detta leder till en overhead som enligt kraven inte accepteras.

Alternativ 2: Den andra lösningen baseras på att varje federat är värd åt sitt eget mellanlager. På detta sätt krävs aningen mer modifikation till federaten, men designen medför ingen onödig overhead samtidigt som lösningen blir mer generisk. Varje federat kan härmed enkelt konfigurera sin egen federat att använda synkroniseringsalgoritmer. Detta val bidrar även till en mer skalbar lösning, då endast varje federat belastas med sina egna funktioner för att lagra status etc., istället för att skapa en gemensam knutpunkt för kommunikation och funktionalitet som Alternativ 1 medför.

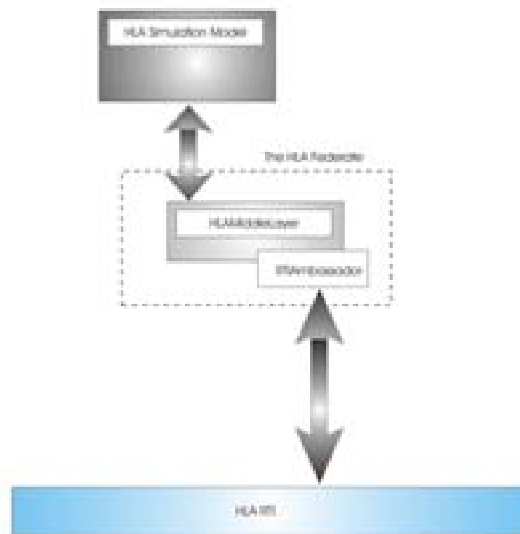


Figur 3.1: *Schematisk figur av lager med funktionalitet för HLA. Till vänster ses en barriärlösning med ett gemensamt mellanlager för HLA-federater. Till höger visas en designlösning där istället varje federat har ett enskilt lager.*

Eftersom alternativ 2 medför skalbarhet och en mer generisk lösning, samt följer kraven bättre, valdes denna design som grund för implementeringen av mellanlagret. Mer detaljerad implementering beskrivs i Avsnitt 3.3.

3.3 Implementation

På grund av det stöd för nätverksprogrammering och interoperabilitet som Java medför, valdes all utveckling till att utföras i detta programmeringsspråk. En klass implementerades som innehöll samtlig logik för mellanlagret, klassen HLAMiddleLayer. För att kunna få ett för federaten transparent mellanlager fick HLAMiddleLayer ärva klassen RTIAmbassador (se Figur 3.2 nedan). Detta är också den lösning Turner et al. valde för sitt mellanlager och som de ansåg vara en effektiv lösning. Denna design medför att den enda modifikation som behövs göras till federaten är att den istället för att initiera en RTIAmbassador, initierar ett HLAMiddleLayer. På detta sätt får federaten tillgång till all logik som mellanlagret erbjuder, samtidigt som den kan använda RTI-funktionalitet på dess ursprungliga sätt. För att kunna göra detta måste även HLAMiddleLayer implementera och ersätta de metoder som finns i RTIAmbassador. När federaten initierar HLAMiddleLayer kan den ange en tidshanteringsmod, som anvisar vilken tidshantering och vilken konfiguration på meddelanden etc. som federaten önskar. Denna mod ger utvecklaren automatiskt stöd för att använda de funktioner för tidshantering som lagret tillhandahåller. Hur tidshanteringen specifikt sköts i lagret diskuteras i Avsnitt 3.3.1 nedan, medan gränssnittet mot HLAMiddleLayer presenteras i Avsnitt 3.3.2.



Figur 3.2: *HLAMiddleLayer implementerades som en transparent komponent och ärver av HLA-klassen RTIAmbassador. På detta sätt sker ingen onödig kommunikation utom den ursprungliga som sker mellan RTIAmbasssador, RTI och federaten.*

3.3.1 Tidshantering och synkroniseringsalgoritmer

Tre olika moder för tidshantering implementerades; *strict* (strikt konservativ), *relaxed* (helt obunden tidshantering), samt *optimistic* (med algoritmen TimeWarp). Moden anges antingen då HLAMiddleLayer initieras, eller efter initiering, och kan ändras i körtid. De tre moderna kan även konfigureras för användning. Till exempel kan TimeWarp konfigureras för att avancera tiden i federaten endast i vissa intervall och intervallet bestäms av en parameter som kan sättas. Oavsett om en specifik mod är angiven eller ej kan federaten utnyttja både HLA-specifika och lagerspecifika metoder. De tre moderna implementerades övergripande enligt följande i HLA:

- *Relaxed:* helt ”relaxerad” synkroniseringsmod, det vill säga den tillhandahåller inga synkroniseringsmekanismer alls och den är inte tidsbunden. Den har implementerats som icke-tidsstyrd, det vill säga varken TR eller TC.
- *Strict:* strikt tidshantering såsom definierad i [Greenhalgh & Vaghi], det vill säga helt strikt tidshantering implementerad som TC och TR i HLA, och strikt händelsestyrd, konservativ synkronisering, det som även kallas kontinuerlig synkronisering.
- *Optimistic:* optimistisk tidshantering och synkroniseringsalgoritmen TimeWarp implementerad enligt beskrivet i Avsnitt 3.3.2. Tidshanteringen är angiven som TR och TC i HLA.

Implementeringen av de tre olika moderna och respektive implementerade algoritmer är presenterade i mer detalj och i kodform i Appendix B. Här ses även tydligt den stora skillnaden mellan att implementera de olika algoritmerna, och inses lätt att stöd för att implementera TimeWarp är önskvärd.

3.3.2 Gränssnitt

Om en användare av HLAMiddleLayer initierar en federat utifrån en specifik mod utförs alla de specifika funktioner som är nödvändiga per automatik. Om istället en mod inte används måste utvecklaren själv se till att specifika inställningar initieras och att somliga metoder anropas och status etc. lagras då det ska. Utvecklaren kan ändå använda de metoder som tillhandahålls av lagret. Dock finns det då ingen kontroll att samtlig funktionalitet som behövs för korrekt tidshantering är tillgodosedd. Tabell 3.2 nedan presenterar en sammanställning av, inte samtliga, men de huvudsakliga metoder som HLAMiddleLayer tillhandahåller och som en utvecklare kan använda, utöver de som ersätter RTIAmbassador. För komplett metodbeskrivning hänvisas till Java-dokumentationen av lagret.

Tabell 3.2: *De huvudsakliga metoder som en utvecklare kan använda i HLAMiddleLayer, förutom de som ersätter ursprungliga metoder i RTIAmbassador.*

Metod i HLAMiddleLayer	Beskrivning
advanceTime	Enhetlig metod som används för att avancera tiden i en federat till en önskad tidpunkt. För en konservativ federat anropas t.ex. metoden *timeAdvanceRequest*, medan för en optimistisk federat metoden *flushQueueRequest* anropas istället. Initierar även de specifika funktioner för de olika moderna som är nödvändiga.
sendMessage	Enhetlig metod för samtliga moder att skicka ett meddelande. Flera olika varianter på metoden finns för att passa de objekt federaten vill skicka. Initierar även de specifika funktioner för de olika moderna som är nödvändiga.
createAndJoinFederation	En grundläggande metod för att skapa och gå med i en federation. Initierar de specifika funktioner för de olika moderna som är nödvändiga, samt sätter de inställningar som behövs för federaten och för just den valda moden.
saveState	En federat kan även egeninitiera lagring av ett status. Status kan skickas i olika former, t.ex. som ett *Java Object*, eller som ett för syftet särskilt utvecklat lämpligt objekt: *HLAObject,* som bifogas HLAMiddleLayer. Denna behöver inte användas då en mod är aktiv.
rollback	Metod som hanterar alla funktioner som måste utföras i samband med en rollback. Om denna metod används utan en mod måste status samt in- och utmeddelanden för federaten ha sparats manuellt.
retractMessages	Metod som skickar antimeddelanden för alla meddelanden som har skickats under en viss angiven period. Om denna metod används utan en mod måste utmeddelanden för federaten ha sparats manuellt.

3.3.3 TimeWarp

Optimistisk tidshantering stöds visserligen av HLA, men själva funktionaliteten för TimeWarp saknas såsom flera har påpekat [Vardanega & Maziero], [Yan et al.] och [Huang et al.]. De tjänster som behövs för att genomföra en korrekt TimeWarp måste utvecklaren själv skapa och det finns inte kompletta riktlinjer för detta i HLA-specifikationen. Fujimoto, som är en av de mer aktiva forskarna inom HLA-området och DS generellt, utformade en teoretisk algoritm för att implementera TimeWarp i HLA [Fujimoto 1998]. Denna var, såsom författaren själv kommenterade, endast ett förslag. Den var inte komplett och demonstrerade inte den verkliga implementeringen i HLA, utan gav endast teoretiska riktlinjer. Arbetet med att implementera TimeWarp inleddes därför med att komplettera algoritmen och därefter fylla på den med HLA-specifik funktionalitet. Nedan presenteras i detalj den algoritm som implementerades i HLAMiddleLayer.

TimeWarp i HLA

Rollback: Rollback implementeras med metoder enligt beskrivet i Avsnitt 2.2.3 i denna rapport. Två händelser kan orsaka rollback här:

1. Federaten mottar ett meddelande med en tidstämpel som är mindre än den logiska processens tidsstämpel: $t_m < t_{LP}$.
2. Federaten mottar ett antimeddelande. Detta görs genom ett anrop från RTI till metoden *requestRetract*.

Tidshantering: den optimistiska federaten måste vara både TC och TR. Denna åtgärd kan först verka besynnerlig, eftersom den innebär att federaten åläggs med de strängaste tidskraven, något som just optimistisk tidshantering försöker undvika. Anledningen till att detta görs i HLA är för att den optimistiska federaten ska kunna fungera i en federation med även konservativa federater. På detta sätt kan konservativa federater både skicka och ta emot tidstämplade meddelanden från den optimistiska federaten, som även får möjlighet att påverka avancemang av tid. Den optimistiska federaten kan även ta emot tidstämplade meddelanden, för att kunna avgöra om den får in meddelanden med ”gammal” tidstämpel.

Ta emot meddelanden: För att meddelanden ska kunna levereras till federaten utan att RTI tar hänsyn till vilken tidstämpel meddelandena har, anropas *flushQueueRequest* med den önskade tidstämpeln $t_{request}$ för federaten. Härmed levereras samtliga meddelanden som finns och är möjliga till federaten, oavsett tidstämpel, och kan alltså ha en tid lägre än fedeatens lokala tid.

Avancera tid: I anropet till flushQueueRequest anger federaten vilken önskad tid federaten vill avancera till, $t_{request}$. Efter att ha levererat samtliga meddelanden anropar RTI *timeAdvanceGrant* i federatens RTIAmbassdor med den tid som federaten kan avancera till, $t_{granted}$. Denna tid motsvarar den lägsta tid i federationen vilken RTI kan garantera att inga meddelanden kommer att produceras med lägre tid än, vilken är detsamma som GVT, och det gäller att: $t_{granted} \leq t_{request}$

Antimeddelanden: Ett antimeddeland görs genom ett anrop till metoden *retract* i RTI och med ett handtag till det meddelandet. Handtagen lagras varje gång en federat skickar ett meddelande (lagras i en UT-vektor) och varje gång federaten får in ett meddelande (lagras i en IN-vektor).

fossileCollection: *fossileCollection* kallas den metod som används för att radera gamla status och meddelanden som lagras ifall eventuell rollback inträffar. Att lagra dessa data genom hela simuleringen skapar onödig overhead och dessutom är det inte säkert att de får plats i minnet. Därför måste gammal data raderas. Denna funktion måste federaten själv implementera men implementeras istället i HLAMiddleLayer för att federaten ska slippa uppgiften. *fossileCollection* utförs efter varje *timeAdvanceGrant* och med $t_{granted}$ som utgångspunkt. All data äldre än $t_{granted}$ raderas, inklusive UT- och IN-vektorerna.

stateSave: Status lagras genom att antingen kopiera en hel instans av federaten (se även lösning nedan i Avsnitt 3.3.3) eller genom att lagra en lämplig objektrepresentation av samlad status för federaten vid en viss tid. I HLAMiddleLayer anropas en implementerad metod i federaten, *retrieveState*, som lämnar sitt status till lagret, men StateSave kan implementeras på flera sätt än angivet här.

3.3.4 Testfederat med stöd för TimeWarp

En testfederat utvecklades för att verifiera mellanlagret och för att kunna användas i experiment. Den utformades som ett tomt skal med mycket enkelt beteende, och med avsikt att kunna vidareutvecklas genom att addera simuleringslogik i efterhand. Federaten kunde utifrån inlästa scenarios producera olika händelser och efter varierande händelsescheman. HLAMiddleLayer integrerades i testfederaten, vilket gav värdefull feedback avseende vad som krävdes för att integrera lagret i en befintlig federat.

Det som visade sig vara komplext i arbetet, men som inte hade något att göra med mellanlagret, var att implementera stöd för TimeWarp-algoritmen. För att en simuleringskomponent ska kunna stödja processer såsom rollback, förväntas federaten bistå med två metoder: `stateSave()` samt `restoreState()`. `stateSave()` är en metod som samlar processens aktuella status, för att kunna lagra det inför framtida eventuell återställning. Denna metod kan vara omfattande om

processens status är komplext att representera, men går att lösa på flera sätt. `restoreState()` kan däremot ställa till med svårigheter. Beroende på simuleringsprocessens natur kan det vara svårt att backa tillbaka processen till ett tidigare status. Det kan till och med vara så att processen behöver exekveras om från början för att nå tidpunkten för rollback. Men det främsta problemet med att implementera `restoreState` är HLA-relaterat. Ett enkelt sätt att implementera metoden är att radera den gamla processen och initiera en ny process utifrån det status från vilken federaten ska starta från rollback. Detta fungerar däremot inte så bra i HLA-sammanhang. Då en federat går ur en federation och går in igen (som ju blir fallet här), får den ett nytt ID-nummer av RTI. Detta medför att den för RTI blir en helt ny federat, utan en historia av eventuella meddelanden som väntar på att levereras, och utan att en eventuell *retract* skulle kunna genomföras eftersom inga gamla retract-meddelanden finns lagrade som förknippas med den ”nya” federaten. För att undvika detta problem är det bättre att inte radera den gamla processen, utan att istället få den att återgå till ett gammalt status för hela processen, vilket inte är helt trivialt.

En lösning till problemet implementerades i testfederaten. En övre klass skapades som bara har som ansvar att initiera en instans av den riktiga simuleringskomponenten, alltså där den riktiga simuleringslogiken finns, och som initieras som en inre klass till testfederaten. Dessutom har övre klassen som ansvaret att vid eventuell rollback återstarta den inre klassen, det vill säga själva simuleringen, utifrån det status som ska återskapas. Detta är en smidig lösning på problemet och demonstreras i Figur 3.3 nedan. Det måste dock uppmärksammas att denna lösning endast fungerar om rollback uppstår på grund av de orsaker som har diskuterats här. Om orsaken istället är att en dator havererar, och alltså likaså federaten på den, och hela processen för federaten går förlorad, så uppstår ändå problemet som diskuterades ovan.

Figur 3.3: *För att angripa problemet med att återställa status för en federat, men bibehålla identifikationen gentemot RTI, skapades en testfederat med följande struktur; en övre klass är tom förutom att den initierar själva simuleringen och dessutom kan återstarta simuleringen från ett givet status vid rollback, själva simuleringslogiken tillhandahålls i en inre klass.*

Förutom värdefull input till utvecklingen så fungerar testfederaten som en exempelfederat och underlättar för utvecklare, oavsett användning av HLAMiddleLayer eller inte, och kommer att bifogas slutresultatet.

3.4 Examensarbete inom projektet

För att effektivisera projektarbetet skapades ett förslag till ett examensarbete redan i den första fasen av projektet. Uppgiften utformades för att passa inom ramen för projektet och omfattade två personer á 20 poäng. Två examensarbetare anställdes för uppgiften och startade våren 2004. Målet med uppgiften var att använda det skal till mellanlagret som redan hade skapats inom projektet, för att använda och implementera vidare i då de angrep en specifik frågeställning. Frågeställningen var TimeWarp och hur denna skulle implementeras med hjälp av HLA. En första prototyp till synkroniseringsalgoritmen implementerades i samarbete med författaren till denna rapport. Prototypen blev delvis lyckad och gav värdefulla erfarenheter, men producerade ingen komplett TimeWarp-algoritm. Funktioner som t.ex.

inte implementerades korrekt var antimeddelanden och komplett rollback. Däremot var det en god första implementation och hjälpte projektarbetet framåt. Förslaget till examensarbetet presenteras i sin helhet i Appendix A.

4 Experiment och resultat

Detta kapitel summerar arbetet som har genomförts i projektet och presenterar även de experiment som utfördes. Experimenten var uppdelade dels i prestandatest och dels i en mer praktisk utvärdering då HLAMiddleLayer integrerades i en befintlig applikation. Resultatet från experimenten presenteras dels här och dels i en artikel som skrevs i arbetet, se Appendix C. Slutligen presenteras en summering av de aktiviteter som har utförts i projektet.

4.1 Prestandatest

Två olika typer av prestandatest utfördes. Det första avsåg prestandatest av HLA och RTI generellt. Målet med testet var att svara på om HLA är en lämplig arkitektur för att utnyttjas som kommunikationsarkitektur vid utveckling av distribuerade system. Det andra testet gjordes med målet att jämföra prestanda för tre olika synkroniserinsprotokoll som implementerats. Detta för att kunna stödja en användare i valet av protokoll vid utvecklingen av simuleringar. Valet av designlösning (se Avsnitt 3.2) gjorde att overhead vid användning av lagret undviks, och därför ansågs inte eventuell overhead-mätning, som HLAMiddleLayer skulle kunna medföra, behövas.

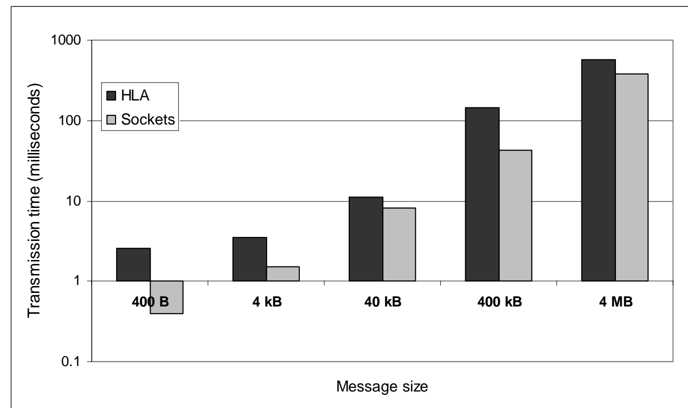
4.1.1 Testplattform

Ett datorlabb sattes upp för experiment bestående av särskilt formaterade datorer och som vardera exekverade varsin testfederat. Labbet utgjordes främst av två likadant konfigurerade PC, sammankopplade med en 100Mbit switch i ett slutet LAN. Prestanda hos datorerna var Pentium III 1 GHz med 256 Mb RAM och som körde operativsystemet Windows XP. Även 4 likadant konfigurerade datorer utnyttjades, men för att förenkla testen användes endast så många för att verifiera att testresultaten som utfördes med endast två datorer var korrekta. Samtliga applikationer implementerades i Java och det RTI som användes vid experimenten var det svenskutvecklade pRTI från företaget Pitch.

4.1.2 Experiment 1 – Utvärdering av HLA jämfört med sockets

För att få ett mått på generell prestanda hos HLA och RTI så jämfördes kommunikationen i M&S-arkitekturen med kommunikationen i ett lättviktssystem. Experiment utfördes för att mäta transmissionstiden för meddelanden i först ett system helt baserat på sockets[4], och sedan jämfört med ett som var helt baserat på HLA. Resultatet presenteras i detalj i den artikel som bifogas i Appendix C. I Figur 4.1 nedan visas ett övergripande diagram av resultatet. Av diagrammet framgår tydligt att socket-lösningen presterade bättre vid alla tillfällen, men inte överdrivet mycket bättre. Slutsatserna var övergripande att RTI presterar dugligt jämfört med andra distribuerade kommunikationsarkitekturer och att den som använder HLA får väga fördelarna med att utnyttja HLA-konceptet, mot den prestandaförsämring RTI medför.

[4] En *socket* är en mjukvarurepresentation av en kommunikationskanal. Vad som avses här är kommunikationskanaler mellan datorer på låg nivå och utan ovanliggande lager, till skillnad från HLA som har lager ovanpå kommunikationen och för synkronisering, meddelandehantering, adressering etc.



Figur 4.1: *Figuren visar prestandatester av sändningstider för varierande storlek på meddelanden i dels en kommunikationslösning baserad på sockets (grå staplar) och dels en HLA-baserad (svarta staplar). Notera att skalan på Y-axeln är logaritmisk.*

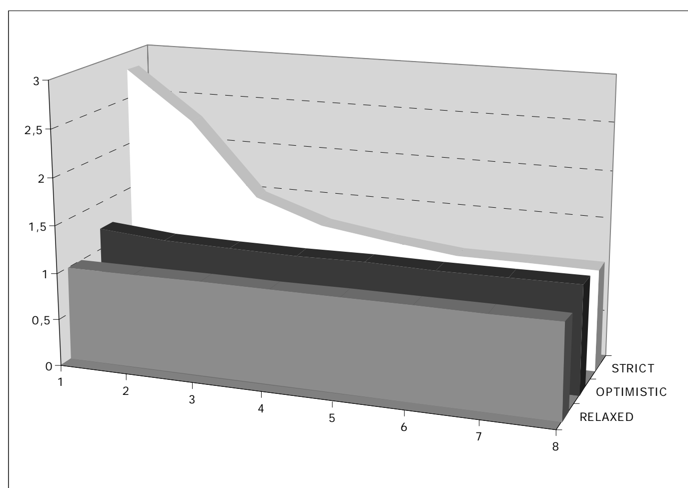
4.1.3 Experiment 2 – Utvärdering av olika synkroniseringsprotokoll

För att underlätta för användning av de olika tidshanteringsmoderna i mellanlagret utfördes även prestandatester på de huvudsakliga moder och synkroniseringsprotokoll som HLAMiddleLayer tillhandahåller. Med grund i dessa resultat kan en utvecklare enklare välja vilket protokoll som ska användas för en specifik situation. Således underlättar experimentresultatet för effektiv användning av lämpliga protokoll och den flexibilitet som HLAMiddleLayer tillför.

Ett antal olika scenarios sattes upp. Två klientapplikationer skapades som kommunicerade med varandra över RTI (testfederaten som beskrivs i Avsnitt 3.3.3. användes). Antalet meddelanden per minut (meddelandehastighet), och även tidsperioden mellan varje meddelande, slumpades fram genom likformig fördelning. Samma meddelandeschema och med samma värden användes därefter för var och en av de tre moderna (*relaxed*, *strict* och *optimistic*). De olika värden som användes för experimenten visas i Tabell 4.1 nedan, där I motsvarar gränsvärdena för tidsperioderna och anges i millisekunder. Moden *relaxed* användes som normalvärde, vilket de andra två moderna (*strict* och *optimistic*) normaliserades mot. Det normaliserade värdet, ε, användes för att kunna jämföra prestanda hos moderna.

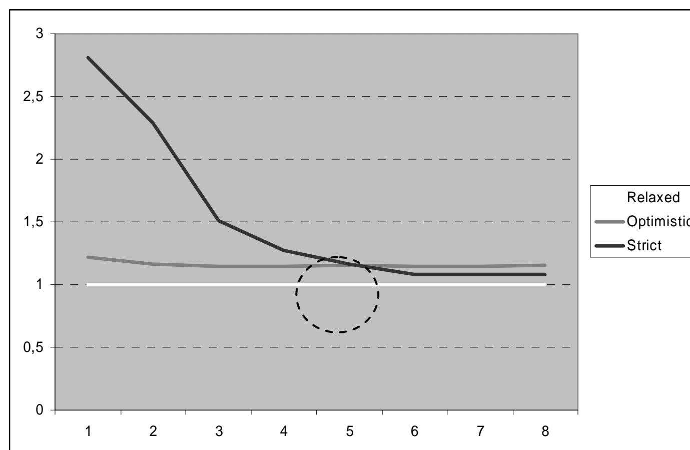
Tabell 4.1: *Tabellen visar de gränsvärden som användes för slumpvist valda tidsperioder mellan skickade meddelanden. Värdena anges i millisekunder [ms].*

Period	Värden [ms]
1.	[5 < I < 10];
2.	[10 < I < 20];
3.	[25 < I < 50];
4.	[50 < I < 100];
5.	[100 < I < 200];
6.	[500 < I < 1000];
7.	[1000 < I < 2000];
8.	[5000 < I < 10000];



Figur 4.2: *Visar en jämförelse av prestanda för de tre olika moderna. Det normaliserade värdet, ε, visas på y-axeln. På x-axeln finns de respektive experimentperioderna såsom beskrivs i Tabell 4.1.*

Experimenten visade tydligt på de olika prestanda hos synkroniseringsprotokollen och att den helt obundna moden, *relaxed*, såsom kan förväntas innebar klart mindre overhead för alla perioder. Den optimistiska moden uppvisade bättre prestanda än den strikta för vissa perioder (högfrekventa perioder, se de till vänster i Figur 4.2), men däremot påvisades en brytpunkt där overheaden plötsligt blir mindre för den strikta moden (se Figur 4.3 nedan). Resultatet presenteras i detalj i artikeln som bifogas i Appendix C. Nedan visas i Figur 4.2 ett diagram över prestanda hos de olika moderna.



Figur 4.3: *Ytterligare en figur av desamma värden som i Figur 4.2. I denna visas diagrammet från en annan vy, där vi ser en brytpunkt i prestanda markerat med en streckad cirkel. För situationer som motsvarar värden till höger om brytpunkten medför den strikta moden mindre overhead än den optimistiska.*

4.2 Experiment 3 – Praktisk utvärdering i befintlig applikation

Ett mer praktiskt experiment utfördes, för att utvärdera användarvänligheten av HLAMiddleLayer, samt för att se att syftet med mellanlagret verkligen uppfylldes.

4.2.1 NätSim-projektet och en redan befintlig applikation

NätSim, förkortningen för Nätverksbaserad Modellering och Simulering, är ett projekt som bedrivs på Institutionen för Systemmodellering på Avdelningen för Systemteknik. Projektet initierades 2003 och studerar området nätverksbaserad M&S, samt utvecklar tekniker och metoder, med målet att utvärdera vad området kan ha för potential för FM. En huvudaktivitet i projektet omfattar design av en arkitektur med syftet att kunna utgöra en gemensam plattform för försvarsrelaterad M&S. En prototyp är under

utveckling och integrerar flertalet olika tjänster, samt verktyg för M&S. All simulering följer för närvarande HLA. För att på ett naturligt sätt utvärdera funktionaliteten i HLAMiddleLayer, initierades samarbete med NätSim-projektet. Mellanlagret kunde på så vis användas för flera olika syften och utvärderas samt vidareutvecklas utefter erhållna resultat.

Praktiskt så utnyttjades en redan befintlig applikation i NätSim, en applikation av spelet Tetris som hade byggts om för att kunna spelas i multiplayer-mod, det vill säga av flera spelare distribuerat och samtidigt. Spelet beskrivs i detalj i bifogade artikeln i Appendix C. Kommunikationen mellan de distribuerade spelkomponenterna är baserad på HLA och RTI. För syftet som applikationen var ombyggd för och för att förbättra prestanda behövde applikationen tillhandahålla olika synkroniseringsmekanismer, något som inte tidigare fanns. Mekanismerna skulle gå att ändras i körtid. Att använda HLAMiddleLayer skulle därför tillföra precis det man sökte och applikationen var därför mycket lämplig som ett första praktiskt exempel på att integrera HLAMiddleLayer i.

4.2.2 Integrering och utvärdering

Att integrera lagret visade sig inte medföra några svårigheter. Det som var krångligt var att implementera stöd för att TimeWarp skulle kunna tillämpas, såsom metoder för lagring av status och för att totalt starta om federaten vid eventuell rollback. Dessa metoder måste implementeras i själva federaten, och det visade sig att mellanlagret gav mycket stöd vad gäller själva HLA-implementationen av TimeWarp. Det visade sig också att det var mycket enkelt att ändra mod för federatens tidshantering, något som inte är en självklar fråga om HLAMiddleLayer eller liknande stöd inte används.

Under arbetets gång uppstod ett antal frågor angående implementeringen och metoder som kunde förbättras. Ett exempel var att federaten skulle kunna ändra sin ”mod” i körtid, en funktion som lagret till en början inte tillhandahöll. Dessa och även andra i integreringsarbetet identifierade brister åtgärdades därför under projektets gång. De som inte hanterades presenteras delvis i Kapitel 5.2. En annan fråga gällde de metoder som skulle implementeras i samband med TimeWarp, varför en generell testfederat som utnyttjar HLAMiddleLayer implementerades som exempelfederat och för att underlätta utveckling och användning av lagret och TimeWarp.

4.3 Summering av projektaktiviteter

Förutom forskningen kring HLA som utfördes under hela projektets gång, kan aktiviteterna i huvudsak presenteras som följer:

- Design: En komponentbaserad, federat-anknuten design till ett mellanlager för HLA utvecklades.
- Implementation: HLAMiddleLayer implementerades med framgång enligt den design som skapades och de implementeringsval som gjordes.
- TimeWarp: TimeWarp-algoritmen färdigställdes och formulerades på ett lyckat sätt.
- Examensarbete: Implementeringen av TimeWarp skedde delvis i samarbete med ett examensarbete som utformades och utfördes inom ramarna för detta projekt.
- Dokumentation: Mjukvaran dokumenterades enligt Suns Javadoc-princip, för att få en så lättanvänd mjukvara som möjligt.
- Experiment: För att verifiera funktionaliteten av HLAMiddleLayer och för att underlätta för användning av olika synkroniseringsfunktioner och lagret, utfördes 3 olika experiment.
- Samarbete: I ett samarbete med NätSim-projektet integrerades HLAMiddleLayer praktiskt i en befintlig applikation, för att utvärdera funktionaliteten praktiskt och för att få feedback.
- Artikel: Resultatet presenterades även i en artikel som antogs inför en konferens och kommer att presenteras under hösten 2005, se Appendix C.

5 Summering samt framtida arbete

5.1 Summering och slutsatser

Ett mellanlager för tidshantering i HLA, *HLAMiddleLayer*, har med framgång designats och utvecklats i Java. Lagret följer bl.a. kraven att inte tillföra HLA-utvecklingen någon overhead, samt att implementeringen ska följa HLA-specifikationerna och således inte påverka RTI-implementationen något. Olika algoritmer för synkronisering har implementerats i lagret och en algoritm för implementering av TimeWarp i HLA har formulerats och implementerats. De olika funktionerna kan användas genom att ange olika moder då lagret initieras, och till dags dato har tre huvudmoder implementerats: *relaxed*, *strict* och *optimistic*.

För att underlätta för effektiv användning av mellanlagret och för att verifiera implementeringen utfördes tre olika experiment för HLAMiddleLayer. De första två var prestandaexperiment och det tredje var en praktisk integrering i en applikation i ett projekt på samma institution, NätSim-projektet. Integreringen verifierade mellanlagrets funktionalitet samt användes för utvärderingssyfte vad gäller användning och gav värdefull input till vidareutveckling som delvis redan har utförts. Prestandaexperimenten verifierade att HLA och RTI fungerar dugligt som kommunikationsarkitektur för distribuerade system, samt gav underlag för användning av de olika synkoniseringsprotokoll som tillhandahålls, genom att utvärdera den overhead som respektive tidshanteringsmod i lagret producerar.

5.2 Förslag till framtida arbete

HLAMiddleLayer implementerades lyckosamt men det finns frågor som skulle kunna behandlas vidare. Som exempel finns det en uppsjö av varianter på synkroniseringsalgoritmer, som uppvisar olika prestanda och funktionalitet. Lagret skulle efter hand kunna kompletteras med även fler varianter. Om däremot för många typer implementeras, kommer slutligen inte användaren att veta vilken variant som ska användas var, och syftet med lagret blir inte uppfyllt.

Det vore givande om en funktion för ”adaptiva protokoll” implementerades. En sådan funktion anpassar tidshantering och synkronisering efter beteendet hos simuleringen. Detta är en ganska komplex fråga att hantera, men vore mycket intressant att implementera och fördelaktigt för många simuleringar där simuleringens beteende varierar kraftigt över tiden.

Ytterligare prestandaexperiment föreslås och som fokuserar på skalbarheten hos HLA och RTI. De experiment som utfördes i denna rapport avsedde endast två respektive delvis fyra noder. Det kan förväntas att prestanda för olika synkroniseringsalgoritmer varierar utifrån antalet noder i simuleringen. Få experiment avseende RTI-prestanda då ett stort antal noder används har genomförts. Det vore intressant att utföra sådana experiment, inte bara ur tidshanteringssynpunkt, utan även generellt för användning av HLA vid simuleringar med ett stort antal simuleringskomponenter.

Stödjande mjukvara för HLA-utveckling underlättar betydligt simuleringsutvecklingen. HLAMiddleLayer stödjer endast tidshantering. Ytterligare funktionalitet, såsom utökningar till HLA Object Management, skulle kunna införas i samma lager.

För att öka användning av HLA-stödjande mjukvara borde det finnas en officiell, internationell webbplats för HLA-support och –användning. En sådan knutpunkt skulle likt ett forum för produkter eller programmeringsspråk kunna fungera som bas för HLA-utveckling och forum för diskussioner. Med hjälp av detta skulle utvecklare kunna stödja varandra och utbyta erfarenheter om HLA: Där skulle HLAMiddleLayer och andra utvecklade produkter inom HLA-domänen kunna publiceras och finnas tillgängliga, vilket skulle stödja HLA-utvecklingen och även föra vidareutvecklingen av HLA framåt. Trots att behovet finns, existerar inte någon sådan webbplats idag. Ett förslag är att om det inte sätts upp en internationell sådan webbplats det i alla fall inom Försvarsmakten skapas ett sådant forum. Detta för att främja HLA-användningen inom försvaret, samt öka möjligheten till återanvändbarhet och kvalitet av modeller inom försvarsmakten.

6 Referenser

[Blümel et al.] Blümel, E., Schenk, M., Schumann, M. *Distributed virtual worlds with HLA?* Proceedings of the Simulation Interoperability Workshop 2002 (FallSIW´02), Orlando, USA, 2002.

[Fujimoto 2000] Fujimoto, R. *Parallel and distributed Simulation Systems*. John Wiley & Sons. ISBN: 0-471-18383-0. Januari 2000.

[Fujimoto 1998] Fujimoto, R. *Time Management in the High Level Architecture*. Publikation Simulation, Vol. 71, No. 6, pp. 388-400, december 1998.

[Greenhalgh & Vaghi] Greenhalgh, C., Vaghi, I. *Demanding the impossible: Data Consistency in Collaborative Environments*. Opublicerat document från Department of Computer Science, University of Nottingham, UK. Kan nås via: http://www.crg.cs.nott.ac.uk/research/projects/nait/private/Consistency-1-2.doc. Senast besökt augusti 2005.

[Huang et al.] Huang, J., Tung, M., Wang, K., Hui, L., Lee, M., Wu, J., Wai, S. *Smart Time Management – The Unified Time Management Mechanism (03E-SIW-038).* Proceedings of the European Simulation Interoperability Workshop 2003 (ESIW´03), Stockholm, Sverige, juni 2003.

[Kuhl et al.] F. Kuhl, R. Weatherly, J. Dahmann, *Creating Computer Simulation Systems*. Prentice Hall, ISBN: 0-13-022511-8. 1999.

[Radeski et al.] Radeski, Al, Parr, S., Keith-Magee, R., Wharington, J. *Component-Based Development Extensions to HLA*. Proceedings of the Spring Simulation Interoperability Workshop, Orlando, USA, mars 2002.

[Shen et al.] Shen et al: Shen. X., Hage, R., Georganas, N.D.. *Agent-aided Collaborative Virtual Environments over HLA/RTI*. Proceedings of the IEEE/ACM Third International Workshop on Distributed Interaction and Real Time Applications (DIS-RT '99), University of Maryland College Park MD, USA, oktober 1999.

[Turner et al.] Turner, S., Cai, W., Chen, J. *A Middleware Approach to Causal Order Delivery in Distributed Simulations*. Proceedings of the European Simulation Interoperability Workshop 2003 (ESIW´03), Stockholm, Sverige, juni 2003.

[Ulriksson et al.] Ulriksson, J., Moradi, F., Svensson, O. *A Webbased Environment for building Distributed Simulations*. Proceedings of the European Simulation Interoperability Workshop 2002 (ESIW´02), London, Storbritanien, juni 2002.

[Vardanega & Maziero] Vardanega, F., Maziero, C. *A Generic Rollback Manager for Optimistic HLA Simulations*. Proceedings of the Conference on Distributed Simulation and Real Time Applications 2000 (DS-RT'00), San Francisco, USA, augusti 2000.

[Vuong et al.] Vuong, S., Scratchley, C., Le, C., Cai, X., Leong, I., Li, L., Zeng, J., and Sigharian, S. *Towards a Scalable Collaborative Environment (SCE) for Internet Distributed Application: A P2P Chess Game System as an Example*. Opublicerat document 2003, University of British Columbia [http://www.magnetargames.com/Technology/DAIS-Vuong Chess-230603R.doc]. Senast besökt augusti 2005.

[Yan et al.] Yan, H., Zhang, Y., Sun, G., Zhong, L. *Research on time warp mechanism in HLA*. Proceedings of the Second International Conference on Machine Learning and Cybernetics, Xi'an, Kina, november 2003.

Appendix A – Förslag till examensarbete

Förslag till examensarbete för två personer á 20 poäng som användes för att anställa två examensarbetare våren 2004. Arbetet avslutades vintern 2004/2005, men har tyvärr inte ännu resulterat i en rapport.

Time Warp för synkronisering av distribuerad simulering i HLA

Kort om institutionen

Totalförsvarets forskningsinstitut (FOI) bedriver forskning för totalförsvaret (www.foi.se). Organisationen är indelad i åtta forskningsavdelningar. En av dem är avdelningen för Systemteknik, belägen i Ursvik i Sundbybergs kommun. Denna avdelning är i sin tur indelad i sex institutioner, varav en är institutionen för Systemmodellering. Institutionen bedriver forskning kring grundläggande metoder och strukturer för Modellering och Simulering (MoS) och deltar aktivt i utveckling av datormodeller av sammansatta och ofta komplexa system och processer.

Bakgrund

På Systemmodellering bedrivs ett forskningsprojekt med namnet "*Miljö för nätverksbaserade simuleringsmodeller*", som förkortas NätSim. Ett av huvudmålen med projektet är att utveckla en nätverksbaserad miljö för MoS. Miljön är helt implementerad i Java och är baserad på Peer-to-Peer-teknologi (P2P) och komponentbaserad MoS (HLA, se nedan). Ett problem som behandlas särskilt är synkronisering av de distribuerade simuleringskomponenterna i HLA. För att hantera detta enklare än som det fungerar idag, har ett mellanlager som tillhandahåller denna funktionalitet ovanpå HLA börjat desginas och implementeras. Ett särskilt intressant protokoll som avses är optimistisk synkronisering med Time Warp. Denna uppgift är av sådan karaktär att den är lämplig för att utföras ramarna för två examensarbeten.

Distribuerad simulering och HLA

Distribuerad simulering (DS) är ett stort forskningsområde världen över, som har behandlats inom försvarsmakten under ett antal år. DS möjliggör för processer och simuleringar att exekveras över flera datorer i ett nätverk [1]. Detta reducerar krav på datorkapacitet och tillåter mycket tunga processer att exekveras. Hastigheten hos exekveringen kan härmed ökas betydligt (s.k. *speedup*), vilket möjliggör exekvering även av mycket tidskrävande processer. DS medför många fördelar, men att distribuera processer ger problem som måste hanteras, såsom upprätthållande av konsistent information, *overhead* och tidsfördröjningar i nätverket, samt synkronisering av parallella processer.

HLA, *the High Level Architecture*, är ett ramverk för distribuerad simulering framtaget av det amerikanska försvaret, och har inneburit ett genombrott för området [2]. Det blev nyligen en IEEE standard (IEEE 1516), vilket har medfört att användningsområden för HLA expanderar, inte bara inom försvarsdomänen. Arkitekturen är ett ramverk för modellering och simulering av återanvändbara interoperabla simuleringskomponenter (federater). I HLA sker all kommunikation och hantering av objekt mellan federaterna via ett HLA-specifikt operativsystem (*Runtime Infrastructure* RTI).

En viktig parameter vid distribuerad simulering är synkronisering av simulerings-komponenterna. Om denna implementeras ineffektivt kan overhead och tidsfördröjningar bidra till att simuleringen blir ineffektiv eller ogenomförbar. Synkronisering delas traditionellt in i två typer; konservativ och optimistisk [1]. Konservativ innebär att hela simuleringen strikt kontrolleras kontinuerligt, ingen delsimulering kan här fortsätta om inte alla delsimuleringar godkänner detta. Optimistisk synkronisering, varav den vanligaste

metoden är Time Warp, är mer komplex. Här tillåts de olika delsimuleringarna att exekvera fritt oberoende av varandra. Endast vid vissa tidpunkter eller händelser kontrolleras att hela simuleringen är i fas och om något fel har uppstått tas felet om hand först då. Konservativa metoder är enklare att implementera, men innebär ofta tidsfördröjningar och overhead i nätverket. Optimistiska är i många fall mer effektiva, men komplicerade. HLA ger idag endast delvis stöd för optimistisk synkronisering.

Mellanlager för synkronisering

Erfarenhet av HLA har visat att trots den stora funktionalitet som ramverket uppbringar, finns det luckor där viss funktionalitet saknas. Forskning inom området har visat att detta bl.a. gäller hantering av synkronisering i HLA och protokoll för detta. Behovet av valfrihet av synkroniseringsprotokoll har uppdagats och möjligheten att utvärdera olika protokoll avseende lämplighet och effektivitet, för respektive simulering som de ska användas i. Detta kan tillhandahållas av ett HLA-lager som separerar HLA-logik från simuleringen och tillför extra funktionalitet. Design och implementering av ett sådant lager har därför initierats och är under arbete.

Uppgift – Time Warp för HLA

Ett särskilt intressant protokoll för mellanlagret är ett protokoll för optimistisk synkronisering som går under namnet Time Warp [1]. För detta finns det till viss del redan stöd i HLA, men problemet kring vad som finns och vad som behöver läggas till kvarstår. En grundlig förstudie och kartläggning av behov och existerande lösningar för problemet behöver därför göras. Därefter implementeras en Java-baserad version av Time Warp i mellanlagret. Slutligen testas och utvärderas funktionaliteten hos protokollet och dokumenteras. Detta är tänkt att utföras inom ramarna för 2 st. 20p-examensarbeten som utförs parallellt. De enskilda uppgifterna och arbetsbelastningen fördelas efter gemensam överenskommelse, efter att förstudien är genomförd.

Handledare

FOI:
Jenny Ulriksson, FOI, Institution för Systemmodellering, 08-5550 3718, jenulr@foi.se
KTH:
Prof. Rassul Ayani, IMIT, rassul@imit.kth.se

Redovisning

Redovisningsformerna bestäms av KTH och FOI krav. För FOIs del innebär detta att examensarbetet demonstreras i lab.miljö, med fördel i det nätverkslab. som finns på FOI. Resultatet sammanfattas skriftligt i en FOI-rapport som kommer att klassas som en vetenskaplig rapport. Den bör skrivas engelska, samt redovisas i ett slutseminarium.

Referenser

[1] r. Fujimoto, *Parallel and Distributed Simulation Systems*. John Wiley & Sons Inc, January 2000, ISBN: 0471183830.

[2] F. Kuhl, R. Weatherly, J. Dahmann, *Creating Computer Simulation Systems – An introduction to the High Level Architecture*. Prentice Hall, 1999, ISBN 0-13-022511-8.

Appendix B – Implementerade algoritmer i HLAMiddleLayer

Detta Appendix presenterar de tre moder och algoritmer som har implementerats i HLAMiddleLayer. I de tre exemplen visas endast *pseudokod*, det vill säga icke-komplett kod. Första delen i varje exempel demonstrerar den kod som måste representeras i varje federat. Efter den streckade linjen visas därefter de algoritmer som har implementerats för respektive mod i HLAMiddleLayer.

Såsom visas är mängden kod och antalet metoder som måste implementeras för respektive synkroniseringsalgoritm mycket olik. Den algoritm som kräver mest stöd och metoder är TimeWarp, vilken därför är den algoritm som en utvecklare vinner mest på att använda mellanlagret för. Dessutom kan moderna enkelt varieras och såsom synes utan mycket variation i själva federatens kod.

I de fall argument till metoder har ersatts av tre punkter är argumenten för samtliga moder likadana. Endast de argument som är ytterligare jämfört med de andra moderna är utskrivna.

RELAXED

```
// the federate´s constructor
public HLAFederate {
      _rtiAmbassador = new HLAMiddleAmbassador (federationName, myName,
                  rtiAddress, rtiPort, handleToThisFederateInstance);
      _rtiAmbassador.createAndJoinFederation (…);
      …
} // end constructor

// all simulation logic and behavior
run () {
      …
      _rtiAmbassador.sendMessage (info);
} // end run

-------------------- HLAMiddleLayer provided logic --------------------

// send a message
sendMessage (info) {
      _rtiAmbassador.sendInteraction (_message, parameters, bytes);
} // end sendMessage

// method for creating and joining an HLA federation
createAndJoinHLAFederation {
      _rtiAmbassador.joinFederationExecution (federateName,
                  federationName, thisFederate, null);
} // end createAndJoinHLAFederation

receiveInteraction (…) {
      …
} // end receiveInteraction
```

STRICT

```
// the federate´s constructor
public HLAFederate {
      _rtiAmbassador = new HLAMiddleAmbassador (federationName, myName,
                  rtiAddress, rtiPort, handleToThisFederateInstance);
      _rtiAmbassador.createAndJoinFederation (…);
      …
} // end constructor

// all simulation logic and behavior
run () {
      …
      _rtiAmbassador.sendMessage (info);
      _rtiAmbassador.advanceTime (timeToAdvanceTo);
} // end run

-------------------- HLAMiddleLayer provided logic --------------------

// send a message
sendMessage (info) {
      _rtiAmbassador.sendInteraction (_message, parameters, bytes,
                  timeStamp);
} // end sendMessage

// advances federate's local time
advanceTime (timeToAdvanceTo) {
      _rtiAmbassador.timeAdvanceRequest (timeToAdvanceTo);
      while (!timeAdvanceGranted) {
            // sleep
      }
} // end advanceTime


// method for creating and joining an HLA federation
createAndJoinHLAFederation {
      _rtiAmbassador.joinFederationExecution (federateName,
                  federationName, thisFederate, null);
      _rtiAmbassador.enableTimeConstrained ();
      _rtiAmbassador.enableTimeRegulation (lookaheadValue);
} // end createAndJoinHLAFederation

// method for receiving messages
receiveInteraction (…, receivedTime, receivedOrdering,
                  messageRetractionHandle) {
      …
} // end receiveInteraction

// called from RTI when granted advance:
timeAdvanceGrant (theGrantedTime) {
      thisFederatesLocalTime = theGrantedTime;
} // end timeAdvanceGrant
```

OPTIMISTIC

```
// the federate´s constructor
public HLAFederate {
      _rtiAmbassador = new HLAMiddleAmbassador (federationName, myName,
                  rtiAddress, rtiPort, handleToThisFederateInstance);
      …
      _rtiAmbassador.createAndJoinFederation (…);
} // end constructor

// all simulation logic and behavior
run () {
      …
      _rtiAmbassador.sendMessage (info);
      _rtiAmbassador.advanceTime (timeToAdvanceTo);
} // end run

// method for restoring state
restoreState (state) {
…
}

-------------------- HLAMiddleLayer provided logic --------------------

Vector IN, OUT, STATES;
boolean timeAdvanceGranted;

// send a message
sendMessage (info) {
      MessageRetractionReturn mrr = _rtiAmbassador.sendInteraction
                  (_message, parameters, bytes, timeStamp);
      OUT.add (mrr); // saved to be used in case of rollbacks
} // end sendMessage

// tries to advance federate’s local time
advanceTime (timeToAdvanceTo) {
      stateSave (); // adds states frequently to vector STATES
      _rtiAmbassador.flushQueueRequest (timeToAdvanceTo);
} // end advanceTime

// method for creating and joining an HLA federation
createAndJoinHLAFederation {
      _rtiAmbassador.joinFederationExecution (federateName,
                  federationName, thisFederate, null);
      _rtiAmbassador.enableTimeConstrained ();
      _rtiAmbassador.enableTimeRegulation (lookaheadValue);
} // end createAndJoinHLAFederation

// method for receiving messages
receiveInteraction (…, receivedTime, receivedOrdering,
                  messageRetractionHandle) {
      …
      IN.add (receivedMessage);
      if (receivedTimeStamp < LocalTime) {
            rollback ();
      }
} // end receiveInteraction

// called from RTI when granted advance:
timeAdvanceGrant (theGrantedTime) {
      fossileCollectStates (theGrantedTime);
```

```
      // since lower messages don't exist!
} // end timeAdvanceGrant

// called from RTI when a message needs to be retracted (antimessage)
requestRetraction (messageRetractionHandle) {
      rollback (toTimeThatEqualsRetractionHandle);
} // end messageRetraction

// method that handles rollback
rollback (timeToRollbackTo) {
      retractMessages (timeToRollbackTo);
      cropStates (timeToRollbackTo); // since now they are not valid
      cropOUT (timToRollbackTo); // since now they do not exist
      this.restoreState (rollbackState); // restore federate's state
} // end rollback

// method that sends antimessages
retractMessages (rollbackTime) {
      MessageRetractionHandle mrh = OUT.get (rollbackTime);
      _rtiAmbassador.retract (mrh);
} // end retractMessages
```

Appendix C – Artikel

Artikel som har antagits och kommer att presenteras på konferensen: Distributed Simulation and Real-time Applications, Montreal, Kanada, 10-12 Oktober 2005.

Consistency Overhead using HLA for Collaborative Work

Jenny Ulriksson
Swedish Defense Research Agency
Dept. of Systems Modeling
SE-164 90 Stockholm, Sweden
E-mail: jenny.ulriksson@foi.se

Rassul Ayani
Royal Institute of Technology (KTH)
Dept. of Microelectronics and Information Technology
SE-16440 Stockholm, Sweden
E-mail: rassul@imit.kth.se

Abstract

CSCW (Computer Supported Cooperative Work) has been around for many years. However, despite growing use of CSCW, general infrastructures that support it are few, and seldom provide data consistency management. A question at issue is how to easily provide different consistency policies for CSCW applications. In a modeling and simulation project at the Swedish Defense Research Agency we have evaluated technologies for developing a CSCW infrastructure. A frequently used distributed simulation architecture, the HLA, appeared as a candidate for beneficially providing CSCW services. Hence we have investigated the use of HLA for the purpose, with successful result. This paper presents some of the outcome, and the experiences from adapting an application to CSCW utilizing HLA. It presents performance experiments for evaluation of three consistency policies for CSCW using HLA, conclusions, future work and some recommendations.

1. Introduction and motivation

The influence of computers and networks grows continuously. New technologies within the field offer new possibilities for many areas, one of them being human collaboration. Allowing geographically dispersed people to communicate and collaborate through computer environments is often referred to as *Computer Supported Cooperative Work* (CSCW). Several environments and applications have been developed for the purpose, but none have yet been truly successful. A problem developers of CSCW software face, is the complexity of implementing services such as consistency and group management. As a result, most professional software products are single-user applications, and the respective collaborative software often lag behind in functionality [1]. A challenging issue here is consistency, i.e. the ordering and update control of user interactions, and there is a lack of supportive frameworks and software assisting developers in this work, which is especially complex within distributed systems. Many systems have been developed, but few have left the laboratory, and few support more than specific aspects of collaborative work.

Due to the lack of general CSCW infrastructures, and with motive in the issues mentioned above, we have initiated development of a CSCW infrastructure at the Swedish Defense Research Agency (FOI). The infrastructure is application independent and will provide necessary services and support for plugging applications into the framework. For the development we have chosen a combination of XML and architecture originally not meant for CSCW but instead for Modeling and Simulation (M&S), the *High Level Architecture* (HLA). Utilizing HLA has the potential of providing among others consistency management and group management in a smooth way. But when doing so, a few questions need to be answered: First, is HLA suitable for the purpose? And if so, what is there to gain?

This paper describes utilization of HLA for distributed CSCW. We also evaluate different synchronization methods to be used for consistency management. Experiences from adapting a simple application to HLA, and the obtained results are also discussed.

2. Background

CSCW is a collective name for human collaboration in computer environments. Software that provides it aim at extending the possibilities of human-human cooperation, despite being seated in geographically dispersed locations [2]. In domains such as the defense, where military is often spread over large distances, this kind of support may not substitute face-to-face activities, but can act as a valuable and easily accessed alternative. This applies to the area of M&S as well, as M&S activities require access to expertise and knowledge, and a high level of cooperation between developer and customer, in order to guarantee the final result and simulation model quality. Within the defense, M&S is an important tool for areas such as decision support and command and control, as well as in training and education. If provided, CSCW can assist in making required expertise and knowledge accessible, and making the M&S related activities more easily accomplished.

2.1 The NetSim project

NetSim is a project at the FOI Department of Systems Modeling, in which the area of network based M&S is studied from the perspective of defense purposes [7]. An important research issue within this research is CSCW, with specific focus on collaborative defense M&S. A simple prototype for a collaborative distributed M&S environment was developed within the project in 2003 [8]. It was based on Peer-to-Peer technology, and provided valuable experiences for the problem at stake, but its capabilities were quite sparse. One of its limitations was that implementation of specific services, such as consistency and concurrency control, was not supported by the chosen technology. This forced us to integrate all functionalities within the collaborative application, which was neither a reusable nor efficient solution. It could benefit from being separated from the application, a favorable separation that has been pointed out also by others [9]. Another reason for dismissing the prototype was that the technology used was not mature enough for the purpose.

2.2. CSCW consistency

As mentioned, a vital issue in CSCW applications is *consistency*. Consistency in CSCW deals with the problem of how to maintain data up-to-date with all users [5]. Two main approaches to consistency are *centralization* and *replication* [4]. When replicated, all clients (nodes) execute equal application replicas, and only states are distributed and coordinated among participants. Centralized, instead the application executes server-wise. As a user interaction occurs, the server executes the application and propagates total updated application copies of states to all participants. The centralized solution is more easily implemented but as described in [3] not efficient, and since replication is more fault-tolerant [3, 4] this is the solution chosen here.

The various forms of consistency are often referred to as *consistency policies*, and the implementation of policies are the *consistency protocols*. Different policies support diverse aspects of consistency [5]. *Strict consistency* is for example defined by Greenhalgh & Vaghi [5] as a policy that lets all participants experience the same view and interactions at the exact same time. Whereas *totally relaxed consistency* is totally non-synchronized, without even event-ordering of interactions. Policies can also be more or less relaxed, in several ways, of which one is discussed in Section 3.2.

2.3. Flexible consistency

In most CSCW applications only one consistency policy is supported, which may be sufficient. But as easily seen, different consistency policies require different levels of synchronization, and hence produce varying computing and network (communication) overhead. Thus in some cases, where the intensity of user interactions vary highly, there could arise a need for, and be more efficient with, *flexible consistency*. The term is here referred to as the conception of when consistency policies are variable in runtime. Properties affecting the efficiency of a consistency protocol, and hence the proposed need for flexibility, are for example communication overload and intensity of user interactivity. To the best of our knowledge, there exists no CSCW software supporting flexible consistency. This is an interesting research topic that has been pointed out in the literature, for example Chung and Dewan [6].

As an example, consider two persons cooperatively reading and commenting a scientific paper. They are seated at two locations, each experiencing they are reading page by page at the same time. They may read at different pace, but when anyone has a comment it must be guaranteed they are at the same page. It would probably cause unnecessary overhead if providing *strict consistency*, i.e. if synchronizing the two views continuously, since it is of no use to synchronize the clients unless anyone has a comment. If the other person has already read the

commented page, he merely has to go back to the page referred to. This can be compared to a totally different situation, interactive flight simulation training, where interactions most likely occur frequently. Here, strict synchronization must be applied in order to provide real-time updates and consistent real-time interaction. A relaxed policy would not provide the same real-time experience as a strict. But the suitability of a policy is a factor that can change during runtime, depending on the current collaborative situation. A trade-off situation occurs, where the overhead of a policy must be balanced towards the required strictness of the synchronization, mentioned further in Section 4.

3. The High Level Architecture

3.1. Distributed simulation and the HLA

Whereas parallel simulation copes with executing a simulation using several processes in parallel to enhance execution time, distributed simulation (DS) deals with management of executing distributed individual simulation components [10]. Different to parallel simulation, the aim of DS is to allow for physically spread simulation components to be jointly executed, sometimes for the goal of carrying out a simulation too demanding to execute on a single computing resource. An important mechanism for DS is Time Management (TM), which should provide a distributed shared notion of time and services for synchronizing and managing the distributed simulation.

The *High Level Architecture* (HLA) is a result of several years of DS activities within the *Defense Modeling and Simulation Office* (DMSO). HLA is an IEEE standardized distributed simulation architecture. It is a framework of rules and software that allow for individual simulation components (in HLA *federates*) to be developed in an interoperable way, and executed in a distributed simulation (in an HLA *federation*) [11]. The central software, actually a kind of distributed operating system, is called the *HLA Runtime Infrastructure* (RTI), and is provided in different fashions by several contributors. RTI provides convenient services, for instance *Federation Management* (creating and controlling federation execution), *Time Management* (advanced federation time management), and *Data Distribution Management* (mechanisms for information routing amongst federates).

3.2. HLA Time Management

The HLA RTI offers flexible and advanced means of simulation time management, and support for different synchronization protocols [12]. Following the HLA rules, a feature is that RTI allows for federates using different TM modes to exist and communicate in the same federation. Furthermore, the TM federate mode can be changed during runtime.

Synchronization is in DS context divided into two main categories, the *conservative* and the *optimistic*. When conservative, logical processes (LPs) can only process events when no temporal discrepancy can be guaranteed. As an opposite, a relaxation of the conservative policy is the optimistic, which allows an LP to process events *optimistically*, with no regard to other LPs. The most often used synchronization protocol for this is *TimeWarp*. Executing optimistically, the federate needs support for recovering from discovered discrepancies. If a message is received with a $t < LT$ (a *straggler* message), the federate needs to *rollback* to the time of the straggler event. This requires frequent federate state save. During rollback, the federate performs *cancellation* of messages sent to other federates. Cancellation messages (*antimessages*) may produce new rollbacks in other federates. Specific TimeWarp mechanisms are not provided by the RTI, but rather possibilities of implementing them [13].

3.3. HLA for CSCW – Collaborative Core

As described in the Introduction, environments and applications developed primarily for collaborative work often lag behind in functionality compared to single-user applications [1]. Moreover, some services, such as consistency management, are not easily implemented and could beneficially be separated from the application [8, 9]. And last, what is missing in the CSCW domain is general infrastructures for CSCW, providing advanced and required services, but *without* being application dependent. This lead us to focus on design and development of a more general infrastructure, which we call *Collaborative Core* (CC), with the objective to efficiently provide CSCW required services for development of collaborative software. Experiences from our first prototype, and result from our research activities, have lead us to the idea that HLA, despite being developed for distributed simulation and not real-time applications, can act as a foundation for our implementation.

Since HLA was accepted as an IEEE standard in 2000 (IEEE 1516), the use of HLA has broadened to

include both non-military and non-simulation fields of application. Examples are the development of an online multi-player framework [14] and a collaborative virtual shopping mall [15]. Utilizing HLA for the purpose of CSCW can offer the potential of mature distributed simulation technology. In this context, the HLA TM can support in providing a shared view, and the HLA TM services can be used for coordinating the propagation of participants' shared states, i.e. for guaranteeing the consistency of participants' views. In order to avoid too much technology dependence, the infrastructure designed was based on HLA and RTI as communication architecture, and XML information models for platform independent representation of messages and other information, described in more detail in [16].

4. Hypothesis

Different CSCW situations may require different levels of consistency strictness, but the policies can cause unacceptable overhead. There is a trade-off between the overhead of a consistency policy, and the preferred and chosen strictness. *Flexible consistency* can make CSCW consistency management more efficient. It can allow for strictness relaxation, which can be beneficial in for example situations with highly intense user interactivity.

Utilizing HLA and the RTI does not only have the potential to provide already built-in services for group and time management, but can also be used to implement *flexible consistency*. Evaluation of the overhead of implemented policies will support appropriate utilization of policies for varying CSCW situations.

5. Using HLA to adapt an application to CSCW

The test bed was composed of a PC network environment, and an application that was attached with an HLA interface, suited for experiments.

5.1. Requirements

The most important boundaries and requirements for the CSCW intended here are summarized here:

- Distributed CSCW
- Synchronous work: The CSCW intended here is immediate and synchronous, i.e. real-time
- Collaborative M&S: M&S activities are assumed, which require a highly varying user interaction intensity depending on activity
- Small groups: Groups are assumed small (2-4 persons). Could be larger, but would then require social support, an issue not covered here

5.2. Architecture

The environment chosen is totally distributed apart from the HLA RTI[1], which is a hybrid between a centralized and distributed architecture. Furthermore, our collaborative application is replicated, i.e. each client executes a replica of it. This means every entity is responsible for its own updates and is independent of other federates.

5.3. Software test bed – CollabTetris

We wanted to be able to not only measure, but also to visualize the overhead of various consistency mechanisms, i.e. visualize eventual differences between application replicas as players play. We also wanted the user interactivity to be easily varied. A simple but demonstrative application was developed, *CollabTetris*, a collaborative version of the game Tetris. In Tetris, objects with different shapes occur from the upper part of the game area (see Figure 1 below). One object at a time appears, and falls downwards at a given constant rate (and can not be moved upwards). The user turns objects into positions and in orientation, such that they fit in with already landed objects. After an object has landed, a new random object appears from above. When the user has succeeded in matching the landed objects, so that at least one line is filled, that line disappears and the user scores. The goal is to fit as many of the landed objects as possible, so that the game area is not filled up, and obtain a high score. The rate of falling objects increases as the game level rises. Eventually, the rate gets too high and the objects are filling up too fast, for the user to be able to erase them in time.

CollabTetris provides multi-gaming possibilities, i.e. clients may together play the game in a distributed fashion. This way, with applied communication facilities, two or more users can discuss and collaborate about managing the objects the best way, together reaching a high score. Although Tetris is a simple game, it can in our context be compared to an easily implemented interactive simulation (where the simulation is a game), and end-users (players)

---

[1] The specific implementation of HLA RTI chosen here is the one from Pitch, see `<http://www.pitch.se>`.

collaboratively use and affect the simulation. Several useful parameters in the game could be varied for tests. All participants had full rights to affect the game area and move the objects. Social aspects such as contention were not of interest or covered here, since the suitability of HLA and consistency policies were of primary interest.

Figure 1. CollabTetris: Game application adapted to HLA for collaborative use.

5.4. Application federate – Adapting the application to HLA

An HLA-compliant interface was created and attached to the application, developed as a general HLA federate, the *ClientFederate*, that acts as the connection to the *collaboration federation.* When an application is launched, and started in multi-player mode, a ClientFederate is initiated and connects to the federation. Using the interface, services such as group management and support for guaranteeing interactions from and to other clients, can be utilized. All communication was handled through the RTI.

5.5. Consistency policies

Three different consistency policies were implemented in the ClientFederate and with supporting functionality in CollabTetris. The first was used as a reference to the other two, since it used minimum required restrictions and handshaking procedures etc. when communication through RTI.

- RELAXED: totally relaxed consistency, no synchronization at all and no time management, as defined in [5],
- STRICT: strict consistency as defined in [5], i.e. continuously synchronized participants, and
- OPTIMISTIC: optimistic consistency management, meaning applications process events optimistically, but have the ability to discover temporal discrepancies and recover from them.

An In-House developed middle layer for HLA TM, called the *HLAMiddleLayer*, was used for implementation. This provided required TM services in a federate transparent way, and significantly facilitated development. Specific policy implementation was performed as follows:

- RELAXED: implemented as non-time-regulating and non-time-constrained (i.e. utilizing no TM mode at all), and no synchronization mechanisms whatsoever
- STRICT: using event-ordered transmission, time-constrained and time-regulating TM mode. Strict conservative synchronization, i.e. no event could be processed if the RTI could not guarantee no old events would appear
- OPTIMISTIC: implemented according to [12] and [17] as time-regulating, time-constrained, and synchronized with the TimeWarp (TW) protocol. Additional methods had to be developed in the CollabTetris for rollback, restoring state and state saving among others

The method that required most modification was the OPTIMISTIC and implementing the TW protocol. Additional methods such as `retrieveState()` (called from federate to retrieve latest status for eventual future rollbacks) and `restoreState(State restore)` (called from federate to rollback to specific state) implied great modification to the original application. Despite using the very supportive HLAMiddleLayer, modifications were challenging.

6. Experiments and results

6.1. Physical test bed

Experiments focused on evaluating general use of HLA for CSCW and evaluation of some policies for this purpose. Moreover, since only small groups of people were considered in tests, we made the assumption that two computers were sufficient for these specific tests. Two equally configured computers[2] were connected with a switch[3] in a closed LAN. One client resided on each computer. No real

[2] Performance: 256Mb RAM and 1.0 Gb Intel Pentium III Processor.

[3] Ethernet LAN with maximum transmission speed of 10 mbps.

persons were used as clients, only fictive scenarios of realistic human interactions. This to ensure that the exact same scenarios were used for all experiments (when varying parameters etc.). Some tests were made with four equally configured computers to verify the functionality of the software and test bed, but since collaboration groups are assumed small, as an assumption only experiments using two computers were conducted.

Since Java's built-in timer has a granularity of tenths of milliseconds, which did not correspond to our test requirements, it was dismissed. Instead we used Roubtsov's timer package[4], which has a granularity close to microseconds.

6.3. Result I: HLA evaluation – HLA compared to sockets

Initial tests were performed to validate the overall use of HLA as communication architecture for CSCW. Extensive experiments with varying message sizes and corresponding message transmission times using the RTI were performed and are presented in [16]. Though packet sizes in the specific CollabTetris were assumed relatively low according to the nature of updates, of about 1-5 kilobytes, different collaborative applications uses differing packet sizes, which is why this parameter was varied highly in tests.

Furthermore, experiments evaluating the transmission times compared to pure sockets were performed. In these experiments two equally set up test scenarios were used, the first one using communication based on HLA and the pRTI, and the second based purely on sockets. The same number of same-sized packets were sent using the two communication architectures, and sizes were varied. Presented below is a collective image of how well pRTI performed compared to sockets. In Table 1 measured mean values of transmission times are shown, and in Figure 2 the diagram for those values. It is shown that the RTI was in general slower than pure socket communication, but if considering that transmission times are still acceptable low for the situation to be used in, RTI performed relatively well. The tests and conclusion background are described in more detail in [16] and support the idea that HLA can actually be used for the purpose of CSCW, but only if regarding small message packet sizes.

[4] Read more on: <http://www.javaworld.com/javaworld/javaqa/2003-01/01.qa-0110-timing.html>. [accessed February 18, 2005].

Table 1. Measured mean values of transmission times for various packet sizes, using sockets compared to HLA and the pRTI.

	Mean values in [ms]	
	sockets	**HLA**
400 B	0,4	2,6
4 kB	1,5	3,6
40 kB	8,2	11,2
400 kB	42,1	143,5
4 MB	380,0	577,7

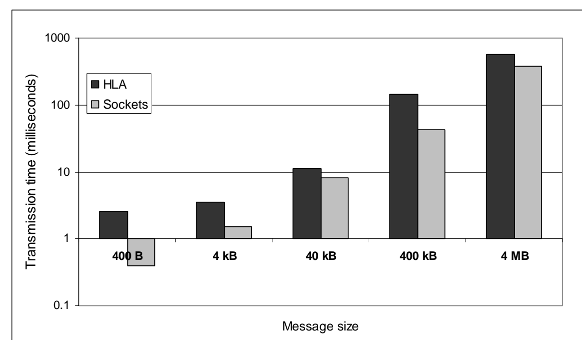


Figure 2. Equal experiments with various-sized packets sent between two computers using only sockets (grey staples) and only pRTI (black staples). Y-axis displays transmission times in milliseconds in log. scale. X-axis displays tested packet sizes.

6.4. How to measure consistency overhead

After validating HLA, we could focus on our primary target; to investigate the overhead that consistency policies produce. When collaborating, diverse forms of interaction are at stake, and many different levels of interactivity are expected (i.e. number of user interactions during a period of time). In real-time collaboration, participants' views are expected to be totally synchronized at all times. Delays in updates as result of a, for the situation, non-efficient consistency policy are not desired. In order to measure overhead we executed the same automated user scenarios for each of the three policies, and total execution times for each scenario were measured, for various I. Since general transmission times were already evaluated (see [16]) these values were of non interest. Instead the respective execution times the policies produced were compared with the reference policy, the totally relaxed. Execution times were then normalized regarding to the relaxed, and resulted in a parameter here called ε that was used in our evaluation.

A clarification of the parameters used in experiments:

- ε: Epsilon. Parameter for policy overhead. Defined as execution time divided by the execution time for the totally relaxed policy.
- I: User inter-interactivity times. The time-interval between two consecutive interactions (in queuing theory known as inter-arrival times).
- T_E : Total execution time. The total execution time of a user scenario for n number of interactions.

6.5. Experiment sessions

Experiments were conducted in the form of *collaborative sessions*. Each session consisted of n number of user interactions, predefined in a scenario. Clients were represented by different scenarios, and a client had the same scenario for all tests. I was defined by intervals, within which a random time period to the next interaction event produced was computed. This was done in order to simulate as natural user interactions as possible. Three consistency policies, MODEs, were tested for each I. Values of all parameters are represented in Table 2. Primary parameter measured was total execution times for a scenario and a client, T_E. Moreover, the number of rollbacks TimeWarp produced was also measured, and mean values for the number of rollbacks per event were estimated. Furthermore total execution times of performed rollbacks where measured, mainly to make sure no extraordinary rollbacks occurred, and they did not.

Table 2. Values of parameters in experiments.

Parameter	Values
n	1000
I	1. [5 < I < 10]; 2. [10 < I < 20]; 3. [25 < I < 50]; 4. [50 < I < 100]; 5. [100 < I < 200]; 6. [500 < I < 1000]; 7. [1000 < I < 2000]; 8. [5000 < I < 10000];
MODE	RELAXED; STRICT; TIMEWARP

6.6. Result II: Comparison of ε

Table 3 presents measured and computed values of ε for the three policies and all I, and Figure 3 presents the respective behavior of ε. As clearly seen the RELAXED policy performed best at all times, which could be expected. The OPTIMISTIC proved next best performance and better than the STRICT. The behavior of the RELAXED and OPTIMISTIC was very similar. And the number of rollbacks for TimeWarp did not vary much for various I, with an average of 0,16 rollbacks/event for all I.

Table 3. Measured ε for all policies and I.

	RELAXED	OPTIMISTIC	STRICT
I:1	1	1,219	2,81
I:2	1	1,160	2,292
I:3	1	1,147	1,513
I:4	1	1,145	1,276
I:5	1	1,152	1,166
I:6	1	1,141	1,085
I:7	1	1,144	1,085
I:8	1	1,153	1,084

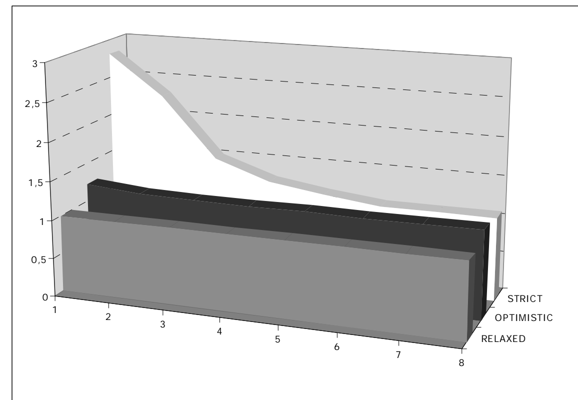


Figure 3. Depicting ε for all consistency policies measured. Y-axis displays ε, and X-axis the eight tested intervals for I.

The STRICT protocol displayed the most differing behavior. As the inter-interactivity was very small (to the left in Figure 3), it produced increasing overhead and more than compared to the other two policies. For small inter-interactivity values (I1 – I4 in Table 2), it was clearly not efficient to use the STRICT policy. This behavior can be explained by the different reasons for causing policy overhead. The OPTIMISTIC overhead was produced merely due to rollbacks, not due to strict synchronization methods. The timestamp to roll back to depends on the interval I. Since the number of rollbacks did not vary during different I, the overhead was constant in proportion to chosen I. On the contrary, the overhead for the STRICT policy was the result of time-consuming hand-shaking

synchronization methods. These methods required just as much time no matter using a large or a small `I`. In other words, as the user inter-interactivity decreased, the overhead did not increase but rather became more obvious. But on the other hand, when inter-interactivity assumed larger values, for example in `I` nr. 8 in Table 2, this overhead flattened out. Since total execution times to the right in Figure 3 are much larger, but the overhead is the same as for measures on the left, the overhead becomes invisible.

6.7. Result III: Turn mark for policy efficiency

Figure 4 illustrates another diagram for Figure 3. A turn mark is depicted with a dotted circle where the STRICT policy becomes more efficient than the OPTIMISTIC, due to the explanation of overhead reasons as presented in Section 6.6 above. The turn mark occurs when `I` is somewhere in `[100 < I < 200]`, which means if the inter-interactivity is greater than this (the right side of the figure), it is more efficient to use the STRICT than the OPTIMISTIC policy.

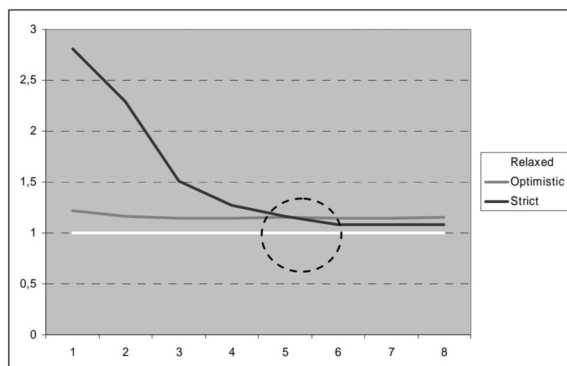


Figure 4. Another figure of ε for all consistency policies and all I. Y-axis displays ε, and X-axis the eight tested intervals for I. The ring marks the point where the STRICT mode is more efficient than the OPTIMISTIC.

It also looks like the STRICT and OPTIMISTIC policies are almost as good as the RELAXED, but it is important to realize the reason for this. Moving right in Figure 4 means total execution times grow much larger, which means that even if ε for the STRICT policy is closer to the RELAXED, the total overhead time is larger than for the smaller inter-interactivity intervals, since: `overhead = ε * total time for the RELAXED`.

7. Conclusions

Experiments concluded that the HLA and RTI as communication infrastructure for CSCW worked well and performed reasonably well when comparing it to corresponding socket communication. Moreover, implementing consistency policies for CSCW, utilizing HLA, proved successful. Experiments showed that the HLA RTI worked well for utilization in CSCW and for propagating real-time user interactions within a CSCW application, and was hence successful for our purpose.

Not surprisingly the reference consistency policy used, the RELAXED, performed “best” concerning overhead, since it uses no mechanisms for consistency control. The OPTIMISTIC was more efficient than the STRICT protocol when smaller user inter-interactivity was considered, i.e. intervals lower than 100 ms between user interactions. When using larger inter-interaction times, instead a turn mark occurred, where the STRICT policy was more efficient than the OPTIMISTIC. Not much, but still enough to justify utilization of a STRICT protocol instead.

Adapting an existing application to support complex synchronization protocols (such as TimeWarp) that does not support functionalities such as `restoreState()` and `stateSave()`, is a rather complex task. HLA provided a good foundation for this, but not much support for TW. The HLAMiddleLayer was beneficial, supporting the lack of TW support, but can benefit from further development.

An important comment regarding our results from using HLA for CSCW is that it is sufficient for the specific kind of CSCW we intended here. It also proved beneficial for utilizing already built-in and advanced functionalities such as group and time management. But, if assuming really high communication intensity, such as streaming communication, or if packets are larger than assumed here, HLA has not yet been proved suitable (see [16]). Furthermore, the question of HLA scalability can become of interest to evaluate, if considering larger groups than assumed in this paper.

8. Future work

The developed infrastructure, CC, has been implemented successfully but is still a prototype. Future work includes further development and generalization, and also more advanced means of communication support. Moreover, consistency policies for and synchronization of distributed participants in CSCW will be further handled. An issue

to consider is relaxation of synchronization constraints towards CSCW utility. And additionally, extended work with the middle layer for HLA time management (HLAMiddleLayer) will be performed, in order to further simplify the use of HLA TM. Experiences from this work will be considered when extending it.

9. References

[1] Li, D. & Li, R. Bridging the Gap Between Single-User and Multi-user Editors: Challenges, Solutions, and Open Issues. *Session III of the Fourth International Workshop on Collaborative Editing Systems*, New Orleans, USA, 2002.

[2] Churchill, E., Snowdon, D. & Munro, A. Collaborative Virtual Environments – Digital Places and Spaces for Interaction. Springer-Verlag. ISBN 1-85233-244-1, 2001.

[3] Roberts, D. & Wolff, R. Controlling Consistency within Collaborative Virtual Environments. *Proceedings of the Eigth IEEE International Symposium on Distributed Simulation and Real-time Applications*, pp. 46-52, 2004.

[4] Prakash, A., Shim, H.S. & Lee, J.H. Data management issues and trade-offs in CSCW systems. *IEEE Transactions on Knowledge and Data Engineerin*g, pp. 213-227,1999.

[5] Greenhalgh, C. & Vaghi, I. Demanding the Impossible: Data Consistency in Collaborative Virtual Environments. Version 1.2, Department of Computer Science, University of Nottingham UK, 2002.

[6] Chung, G. & Dewan, P. Towards Dynamic Collaboration Architectures. *Proceedings of ACM conference on Computer Supported Cooperative Work*, Chicago, USA, November 2004.

[7] Eklöf, M., Ulriksson, J. & Moradi, F. A Network Based Environment for Modeling and Simulation. *Proceedings of the NATO Modelling and Simulation Group, Symposium on C3I and M&S Interoperability*, Antalya, Turkey, October 2003.

[8] Ulriksson, J., Moradi, F. & Ayani, R. Collaborative Modelling and Simulation in a Distributed Environment. *Proceedings of the European Simulation Interoperability Workshop*, Stockholm, Sweden, June 2003.

[9] Ahlers, K., Kramer, A., Breen, E., Chevalier, P.Y., Crampton, C., Rose, E., Tuceryan, M., Whitaker, R. & Greer, D. Distributed Augmented Reality for Collaborative Design Applications. *Computer Graphics Forum*, Vol. 14, Issue 3, pp. 3-14, 1995.

[10] Fujimoto, R. Parallel and Distributed Simulation Systems. Johnn Wiley & Sons, Inc. USA. ISBN: 0-471-18383-0, 2000.

[11] Kuhl, F., Weatherly, R. & Dahmann, J. Creating Computer Simulation Systems – An Introduction to the High Level Architecture. Upper Saddle River, NJ: Prentice Hall. USA. ISBN: 0-13-022511-8, 1999.

[12] Fujimoto, R. Time Management in the High Level Architecture. *Simulation,* Vol. 71, No. 6, pp. 388-400, December 1998.

13] Yan, H., Zhang, Y., Sun, G. & Zhong, L. Research on Time Warp Mechanism in HLA. *Proceedings of the Second International Conference on Machine Learning and Cybernetics*, Xi'an, China, November 2003.

[14] Vuong, S., Scratchley, C., Le, C., Cai, X.J., Leong, I., Li, L., Zeng, J., & Sigharian, S. Towards a Scalable Collaborative Environment (SCE) for Internet Distributed Application: A P2P Chess Game System as an Example [online]. Available via: <http://www.magnetargames.com/Technology/DAIS-Vuong-Chess-230603R.doc> [accessed February 14, 2005].

[15] Zhao, H. & Georganas, N.D. Collaborative Virtual Environments: Managing the Shared Spaces. *Networking and Information Systems Journal*, Vol. 3, nr. 2, pp. 1-23, 2001.

[16] Ulriksson, J., Moradi, F., Liljeström, M. & Montgomerie-Neilson, N. Building a CSCW Infrastructure utilizing an M&S architecture and XML. To appear in *Proceedings of the second international conference on Cooperative Design, Visualization and Engineering (CDVE2005).* Palma de Mallorca, Spain, September 2005.

[17] Huang, J., Tung, M., Wang, K., Hui, L., Lee, M., Wu, J., & Wai, S. Smart Time Management – The Unified Time Management Mechanism. *Proceedings of the European Simulation Interoperability Workshop*, Stockholm, Sweden, June 2003.

FOI är en huvudsakligen uppdragsfinansierad myndighet under Försvarsdepartementet. Kärnverksamheten är forskning, metod- och teknikutveckling till nytta för försvar och säkerhet. Organisationen har cirka 1350 anställda varav ungefär 950 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, bedömningen av olika typer av hot, system för ledning och hantering av kriser, skydd mot hantering av farliga ämnen, IT-säkerhet och nya sensorers möjligheter.

FOI
Totalförsvarets forskningsinstitut
Systemteknik
164 90 Stockholm

Tel: 08-555 030 00
Fax: 08-555 031 00

www.foi.se