

T.C.
GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
MÜHENDİSLİK VE FEN BİLİMLERİ
ENSTİTÜSÜ

HLA TABANLI DAĞITIK BENZETİMLER
İÇİN TAM DAĞITIK KAYIT VE YENİDEN
OYNATMA MİMARİSİ

Emrah KAYA

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİMDALI

GEBZE

2009

T.C.
GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
MÜHENDİSLİK VE FEN BİLİMLERİ
ENSTİTÜSÜ

HLA TABANLI DAĞITIK BENZETİMLER
İÇİN TAM DAĞITIK KAYIT VE YENİDEN
OYNATMA MİMARİSİ

Emrah KAYA

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİMDALI

TEZ DANIŞMANI
YRD. DOÇ. DR. FATİH ERDOĞAN SEVİLGİN

GEBZE

2009



**GEBZE YÜKSEK
TEKNOLOJİ
ENSTİTÜSÜ**

MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ

JÜRİ ONAY FORMU

JÜRİ

ÜYE (BAŞKAN) : Yrd. Doç. Dr. Fatih Erdoğan SEVİLGİN

ÜYE : Yrd. Doç. Dr. Abdullah BAL

ÜYE : Yrd. Doç. Dr. Hacı Ali MANTAR

Gebze Yüksek Teknoloji Enstitüsü Mühendislik ve Fen Bilimleri Enstitüsü Yönetim Kurulu'nun/..... tarih ve/..... sayılı kararı ile yukarıdaki öğretim elemanlarından oluşmuş jüri tarafından düzenlenen/..... tarihli Tez Savunma Tutanağı neticesinde Yüksek Lisans öğrencisi Emrah Kaya'nın çalışması GYTE Mühendislik ve Fen Bilimleri Yönetim Kurulu/..... tarih ve/..... sayılı kararıyla Bilgisayar Mühendisliği Anabilim Dalında Yüksek Lisans tezi olarak onaylanmıştır.

İMZA/MÜHÜR

ÖZET

TEZİN BAŞLIĞI: HLA Tabanlı Dağıtık Benzetimler için Tam

Dağıtık Kayıt ve Yeniden Oynatma Mimarisi

YAZAR ADI : Emrah KAYA

Benzetim sistemlerinde benzetim verilerinin kaydı önemli bir yer tutmaktadır. Eğitim amaçlı benzetimlerde kullanıcıların performansının ölçülebilmesi, analiz benzetimlerinde koşum esnasında gerçekleşen olayların incelenmesi, her iki türde de benzetim geçmişine dönerek olayların tekrar gözden geçirilmesi gibi ölçüme yönelik ihtiyaçlar için benzetim verileri gerekmektedir. Hata ayıklama, birim ve modül test yapma gibi geliştirmeye yönelik ihtiyaçlar da benzetim verilerinin toplanabilmesi ile karşılanabilmektedir. Ayrıca birden çok öğeden oluşan benzetimlerin daha etkin ve hızlı geliştirilebilmesi için gerekli olan bağımsız hata ayıklama, test ve analiz gibi imkânlar ancak benzetim öğelerinin diğer öğelerin verilerinin kaydedilip kullanılabilmesi sonucu bağımsız olarak geliştirilebilmesi ile mümkündür.

Dağıtık benzetimler, sonuçların daha hızlı bir şekilde elde edilebilmesi, farklı coğrafi bölgelerdeki kullanıcıların aynı benzetimde yer alabilmesi gibi özelliklerinden ötürü yaygın olarak kullanılmaktadırlar. Bu benzetimlerde kayıt işlemleri eşzamanlılık sorunları ve sistemde birbirinden farklı birden çok öğenin bulunması sebebiyle daha karmaşıktır. Dağıtık benzetim mimarilerinden olan HLA standardı getirdiği eniyileme, tekrar kullanılabilirlik ve benzetim öğelerinin birlikte işlerliğini arttırmak gibi faydalar getirmiştir. Fakat bu faydaları sağlayan kısmi veri güncellemesi, tüme gönderim olmaması, özel veri tipleri gibi özellikleri daha karmaşık ve maliyetli bir veri toplama çözümü sonucu doğurmuştur. Ayrıca benzetim öğesinin yayınlamadığı özel verilerinin de kaydedilmesi ve eşzamanlılık sorunlarının çözülmesi gerekmektedir.

Bu çalışmada öncelikle veri toplama ihtiyaçları ve önceki çalışmalarda sunulan çözümler incelenmiştir. Tam dağıtık bir mimari önerisi üzerinde durularak bu mimaride benzetim verilerinin yanında özel verilerin kaydı sağlanmıştır. Yeniden oynatma sırasındaki eşzamanlılık sorunu çözülerek önerilen sistem gerçekleşmiş, performans ve doğruluk deneyleriyle etkinliği ve uygulanabilirliği ortaya konulmuştur.

SUMMARY

**TITLE : A Fully Distributed Data Collection and Replay
Architecture for HLA Based Distributed Simulations**

AUTHOR : Emrah KAYA

The data collection is an important part of simulation systems. Reviewing requirements like evaluating the trainees' performance in training simulations, examining the events in analysis simulations and rolling back to a previous simulation state can be met with the collection of the simulation data. Development requirements like debugging and unit-module testing also require this data. More effective and faster development of simulations that consist of more than one simulator is possible with the independent debugging, test and analysis. This can be realized if only the simulators are developed independent from the other simulators by collecting and supplying the other simulators' data.

Distributed simulations are widely used for a number of reasons. Getting faster results and allowing users in different geographical locations to participate on the same simulation are some of those. Synchronization problems and existence of numerous distinct simulators make the data collection and replay more difficult in distributed simulations. The widely used distributed simulation architecture, HLA, came with not only features like better optimization, reusability and interoperability but also a more complicated and costly data collection availability because of its partial data update, simulation specific data-types and multicasting features. Collection of the private data that the simulators do not publish into the simulation network and synchronization of the simulators are the other problems to be solved.

In this study, firstly the data collection needs and the solutions in the previous studies are examined. A fully distributed data collection architecture capable of not only collecting the simulation data communicated between simulators but also simulators' private data is proposed. A methodology to solve the synchronization problem during replay of simulation is proposed. The proposed solutions are implemented as data collection and replay system. The efficiency and the feasibility of the system are demonstrated with performance and validity tests.

TEŞEKKÜR

Tez çalışmalarımın başladığı günden son güne kadar bilgisini ve emeğini hiç eksik etmeyen, tezin bu aşamaya gelmesinde en büyük katkıyı sağlan çok değerli hocam saygıdeğer Yrd. Doç. Dr. Fatih Erdoğan SEVİLGİN'e içtenlikle teşekkür ederim.

Gerek iş gerekse yüksek lisans çalışmalarımda en yakın ve büyük desteği veren fedakâr eşim Gülşah KAYA'ya en derin sevgilerimle teşekkür ederim.

İÇİNDEKİLER DİZİNİ

ÖZET	iv
SUMMARY	v
TEŞEKKÜR	vi
İÇİNDEKİLER DİZİNİ	vii
SİMGELER VE KISALTMALAR DİZİNİ	ix
ŞEKİLLER DİZİNİ	x
TABLOLAR DİZİNİ	xi
1. GİRİŞ	1
1.1. Veri Toplama İhtiyaçları	2
1.1.1. Koşum Sonrası Gözden Geçirme: (After Action Review-AAR)	3
1.1.2. Benzetim Geçmişine Dönme:	4
1.1.3. Çevrimdışı Analiz:	5
1.1.4. Hata Ayıklama - Birim ve Modül Test:	5
1.2. Problemler	6
1.3. Öneri	7
2. DAĞITIK BENZETİMLER	9
2.1. Benzetim Mimarileri/Standartları	10
2.1.1. HLA	11
3. DAĞITIK BENZETİM SİSTEMLERİNDE KAYIT VE YENİDEN OYNATMA	13
3.1. Kayıt Bakış Açısından HLA	13
3.2. Yapılan Çalışmalar, Mimariler	14
3.2.1. Merkezi Kayıt	14
3.2.2. Kısmen Dağıtık Kayıt	16
3.2.3. Tam Dağıtık Kayıt	18

3.3. Yerel/Özel Veriler	21
4. ÖNERİLEN ÇÖZÜM	23
4.1. Genel Mimari	24
4.2. Eşzamanlılık	26
4.3. Özel Veriler	31
5. GERÇEKLEME	32
6. DENEYSEL ÇALIŞMALAR	36
6.1. Performans Deneyleri	38
6.2. Doğruluk Deneyleri	43
7. SONUÇ VE ÖNERİLER	47
KAYNAKLAR	50
ÖZGEÇMİŞ	52

SİMGELER VE KISALTMALAR DİZİNİ

<u>Simge/Kısaltma</u>	<u>Açıklama</u>
HLA	High Level Architecture (Yüksek Seviyeli Mimari)
DIS	Distributed Interactive Simulation (Etkileşimli Dağıtık Benzetim)
PDU	Protocol Data Unit (İletişim Kuralı Veri Birimi)
RTI	Run-time Infrastructure (Koşum Zamanı Altyapısı)
AAR	After Action Review (Koşum Sonrası İnceleme)
SBT	Simulation Based Training (Benzetim Tabanlı Eğitim)
IEEE	Institute of Electrical and Electronics Engineers (Elektrik ve Elektronik Mühendisleri Enstitüsü)
LAN	Local Area Network (Yerel Ağ)
WAN	Wide Area Network (Geniş Alan Ağı)
FOM	Federation Object Model (Federasyon Nesne Modeli)
SOM	Simulation Object Model

ŞEKİLLER DİZİNİ

<u>Sekil</u>	<u>Sayfa</u>
3.1 Merkezi Kayıt	15
3.2 Kısmen Dağıtık Kayıt	17
3.3 Tam Dağıtık Mimari	19
4.1 Kayıt Sistemi Genel Mimari	25
4.2 Kayıt/Yeniden Oynatma Sistemi Mimarisi Detayı	25
4.3 Federasyona girişte eşzamanlılığın etkileşimlerle sağlanması	28
4.4 Koşum sırasında eşzamanlılığın sağlanması	30
5.1 Kayıt/Yeniden Oynatma Sistemi sınıf yapısı	33
6.1 "Kovalayan" federe kendi hareket denklemine göre ilerliyor	37
6.2 "Kovalayan" federe diğer federenin pozisyon bilgilerine göre ilerliyor	38
6.3 Veri Boyutu - Dosya Boyutu Değişimi	40
6.4 Veri Boyutu - Kayıt Süresi Değişimi	40
6.5 Nesne/Etkileşim Sayısı -Dosya Boyutu Değişimi	41
6.6 Nesne/Etkileşim Sayısı-Kayıt Süresi Değişimi	42
6.7 Karşılaştırma durumunun ekran görüntüsü	45

TABLÖLAR DİZİNİ

Tablo**Sayfa**

6.1 Normal ve Kayıt Verileri ile Koşum Karşılaştırması

44

1. GİRİŞ

Gerçek bir sistemin modellenerek yazılımsal ve/veya donanımsal bileşenler üzerinde bir benzerinin yapılması benzetim(simülasyon) sistemi olarak adlandırılmaktadır. Bu benzetim sistemleri, gerçek sistemin belli durumlar ve girdilere göre davranışının incelenmesi, gerçek sistemde uzun süre alan işlemlerin sonucunun çabuk bir şekilde alınması gibi gerçek sistemin analizini amaçlayan sistemler olabilmektedirler. Savaş benzetim sistemleri gibi uygulamalarda ise gerçek sistemi kullanacak kişilerin eğitimi amacıyla geliştirilmektedirler. Ayrıca geliştirilmesi düşünülen bir gerçek sistem modellenerek üretiminden önce belirli deneylerle fizibilitesi ölçülebilmektedir.

Böylece gerçek sistemde zaman ve maliyet ihtiyacı doğurabilen ya da hataya sebep olabilen durumlar benzetim sistemi üzerinde kolaylıkla incelenerek istenilen amaca daha kolaylıkla ulaşılabilir. Örneğin bir hava tahmin benzetimi ile bir bölgede bir hafta içerisinde oluşabilecek hava durumları kısa bir sürede gerçekçi bir biçimde tahmin edilebilir. Bir fabrikadaki üretim bandının benzetimi ile hangi noktada ne kadar iş yükü oluşabileceği önceden öğrenilerek buna göre planlama yapılabilir. Bir savaş benzetimi ile can veya mal kaybı olmadan askerler eğitilebilir.

Benzetimler, amaç, çalışma şekli, kullanıcı etkileşimi gibi konular dikkate alındığında analiz ve sanal ortam/eğitim benzetimleri olarak ikiye ayrılabilir [Fujimoto, 1999][1]. Analiz benzetimlerinde istenilen sonuçlar olabildiğince çabuk alınmaya çalışılırken sanal ortam/eğitim benzetimlerinde gerçek zamanlı bir koşum sağlanmaya çalışılmaktadır. Kullanıcıların genelde gözlemci olarak kaldıkları analiz benzetimlerinde amaç gerçek sistemin ölçülebilir özellikleri kullanılarak performans, neden-sonuç gibi açılardan incelenmesidir. Sanal ortam/eğitim benzetimlerinde ise kullanıcılar sistemin içinde bir aktör olarak öğelerin davranışlarına etki ederler. Bu benzetimlerde amaç, kullanıcılara gerçek sistemin en uygun gerçeklikte sunulmasıdır.

Benzetimlerin sayılan geniş kullanım alanları zamanla daha karmaşık gerçek sistem sistemlerin benzetimlerinin gerçekleştirilmesini ve benzetimlerden beklenen yeteneklerin artmasını sonuç vermiştir. Gerçek sistemin modeli karmaşıktıkça

benzetim için ihtiyaç duyulan işlem miktarı artmış ve tek bir bilgisayarın işlem kapasitesini aşmıştır. Ayrıca benzetim sisteminin birden çok model ve/veya yazılım/donanım bileşeninden oluşması her bir ögenin farklı geliştiriciler/firmalar tarafından geliştirilmesini gerektirmiştir. Bunların yanında kullanıcıların Dünya'nın farklı yerlerinde geniş alanlı bir ağ (Wide Area Network-WAN) üzerinden aynı benzetim üzerinde buluşması da özellikle eğitim amaçlı benzetimlerde ortaya çıkmış bir ihtiyaçtır. Böylece, kişiler tek bir yere toplanmaya ihtiyaç duymadan benzetim üzerinden eğitim alabilmesi amaçlanmıştır.

Sayılan bu özelliklerin sağlanması, birden çok benzetim ögesinden oluşan, bu ögelerin birbirleri ile haberleştiği benzetim sistemlerini doğurmuştur. Bu sistemler gerçekleşmesi ise paralel ve dağıtık olarak iki şekilde olabilmektedir. Paralel benzetimlerde benzetim ögeleri genel olarak birbirinin aynı özelliklerde (homojen) olup bir işlemin parçalara bölünerek daha çabuk şekilde sonuç alınmasını hedeflemektedir. Dağıtık benzetimlerde ise farklı özellik ve işlevlerdeki (heterojen) benzetim ögeleri farklı modellerin gerçekleşmesini sağlayarak benzetim sistemini oluşturmaktadır. Dağıtık benzetimlerde herhangi bir benzetim ögesi paralel çalışan alt ögelerden oluşabilir. Böylece dağıtık ve paralel benzetimler iç içe kullanılabilir.

Paralel benzetimlerde benzetim ögeleri, genellikle bir ana ögenin kontrolünde kendilerine verilen işleri bitirmek amacıyla çalışırken, dağıtık benzetimlerde amaç ve çalışma şekli olarak birbirinden daha bağımsız benzetim ögeleri birbirlerinden ve kullanıcılardan aldıkları veriler doğrultusunda işlemleri gerçekleştirmektedirler. Dağıtık bir benzetimin doğru çalışabilmesi için her bir benzetim ögesinin ihtiyaç duyduğu verileri diğer ögelerden doğru zamanda alması gerekir. Dolayısıyla benzetimin bir ögesi diğer ögelerin verilerine bağlı olmaktadır.

1.1. Veri Toplama İhtiyaçları

Benzetim sonucunda üretilen sonuçların analizi veya benzetimin performansı gibi kıstasların ölçülmesi amacıyla benzetimin incelenmesi gerekmektedir. Bu da, benzetim ögelerinin ürettiği sonuçların kaydedilerek analizi ile mümkün olabilmektedir. Benzetim ögelerinin geliştirilmesi ve test edilmesi aşamasında ise yine diğer ögelerden gelecek verilere ihtiyaç duyulmaktadır. Benzetimin bütün olarak çalıştığında geliştirilen ögenin nasıl davranacağını gözlemleyebilmek ancak

diğer öğelerin ihtiyaç duyulan verilerinin doğru zamanda bu öğeye aktarılması ile mümkündür. Bunu sağlamanın başka bir yolu benzetimin bütün olarak çalıştırılması olabilir. Fakat bu şekilde bir yaklaşım, hem geliştirmeyi yavaşlatacak hem de test sürecinde diğer öğelere bağımlılık oluşturacağından sağlıklı bir sonuç elde edilemeyecektir.

Gerek analiz, gerekse eğitim benzetimleri, gerçekleşen olayların, işlemlerin ve girdi-çıktı ilişkilerinin incelenmesi, oluşan bir hatanın nedeninin anlaşılması gibi ihtiyaçlarla benzetimin tekrar oynatılmasına ihtiyaç duyabilmektedir. Tekrar oynatma, benzetimdeki her olayın aynı benzetim zamanında tekrarlanması olarak özetlenebilir. Bunu sağlamak için benzetim öğelerinin aynı parametrelerle ayarlanıp benzetimin baştan koşturulması uygun görülebilir. Fakat özellikle birbirleri ile yerel veya geniş bir ağ üzerinden haberleşen bu benzetim öğelerinin birbirlerine gönderdikleri mesajlar/veriler, tekrar oynatma sırasında aynı benzetim zamanında ulaşmayabilir. Ayrıca sisteme kullanıcı girdileri varsa aynı benzetim zamanında aynı girdileri sağlamak mümkün olmayabilir. Bu durumda ise gelişen olayların zamanı ve/veya içeriği değişebilir ve tekrar oynatma sağlanamamış olur. Yukarıda bahsedildiği gibi benzetimdeki tekrarlanabilirliği sağlamak için olayların aynı benzetim zamanında gerçekleşmesi gerekir. Bunu elde etmek ya benzetimdeki olayların, mesajların zaman bilgileri ile beraber kaydedilip doğru zamanda yayınlanması, sergilenmesi ile mümkün olabilir.

Dağıtık benzetimlerin bahsedilen özellikleri göz önüne alındığında benzetim verilerinin toplanmasının gerekli olduğunu ortaya çıkmaktadır. Literatürdeki diğer ihtiyaçlar da dikkate alınarak dağıtık benzetimlerdeki veri toplama ihtiyaçları şu şekilde sınıflandırılabilir [Magee & Shanks, 1999], [Van Hook & Calvin, 1997], [Bachinsky et al., 1999], [Li et al., 2006]:

1.1.1. Koşum Sonrası Gözden Geçirme: (After Action Review-AAR)

AAR, benzetim koşumu sonrasında benzetimdeki olayların, girdi ve çıktıların, koşumun ilerleyişinin tekrar gözden geçirilmesidir. Eğitim amaçlı benzetim sistemlerinde, benzetim sonrasında olayların tekrar gözlenip yapılanlar üzerinde yorum yapabilmek, yapılan eğitimin kalitesini ölçmek, eğitimin başarılı olma durumunu izlemek için kullanılarak öğrencilerin performansını en üst düzeye

çıkarmak amaçlanmaktadır. Ayrıca gelecekteki çalışmalar için önemli bir geri dönüş sağlamaktadır. Örneğin benzetim tabanlı eğitim sistemlerinde (Simulation Based Training/SBT) benzetimin esas amacı olan iyi eğitilmiş öğrenciler oluşturmak ancak öğrencilerin performanslarını inceleyerek gerekli olduğu durumlarda aynı benzetimin farklı durumlarda tekrarlanması ile elde edilebilir. Hatta eğitim sırasında öğretmen benzetimi geçmiş bir zamana döndürerek öğrencilere aslında ne yapmış olmaları gerektiğini göstererek benzetimi bu durum ile tekrarlayabilir. Analiz amaçlı benzetimlerde ise analiz sonuçlarının beklendiği gibi çıkmadığı durumlarda, benzetim sırasında nelerin yanlış gittiğini bulmak için AAR önemlidir.

AAR sistemleri genel olarak üç kısımdan oluşur: Veri toplanması, Veri analizi/veri derlemesi ve Veri sunumu [Vasend, 1995]. Bunlar arasında işleyiş olarak da ilk sırada yer alan “veri toplama”, başarılı bir AAR sisteminin temelini oluşturur. Çünkü toplanan verinin niteliği daha sonraki adımları doğrudan etkilemektedir. Ayrıca veri toplama için ne kadar farklı yapı ve özellikteki benzetimlere hitap eden bir çözüm bulunabilirse o kadar geniş bir AAR çözümü elde edilmiş olacaktır. Bu yüzden yapılacak veri toplama ve yeniden oynatma sisteminin olabildiğince geniş alandaki dağıtık sistemlere uygulanabilmesi ve onları en az düzeyde etkileyerek bu işi gerçekleştirmesi gerekir.

1.1.2. Benzetim Geçmişine Dönme:

Benzetim geçmişine dönme, herhangi bir anda (benzetim sisteminin koşumu sırasında da olabilir) benzetimin geçmişte bir zamana döndürülüp bu zamandan itibaren benzetimin tekrar çalıştırılmasıdır. Benzetim geçmiş bir duruma döndürülüp farklı girdilerle tekrar koşturulabileceği gibi, beklenmedik bir sonuç alındığında geçmişe dönerek bu duruma neyin sebep olduğu da araştırılabilir. Eğitim amaçlı bir benzetimde eğitmen benzetimi geçmiş bir durumuna döndürüp sistemin girdilerini ve durumunu değiştirerek öğrencilerin farklı durumlardaki performanslarını inceleyebilir, onlara bu durumlarda hangi girdinin nasıl sonuç vereceğini pratik olarak gösterebilir. Analiz amaçlı benzetimlerde ise analiz sürecinde veya sonucunda beklenmedik bir durum gözlenirse, benzetim geçmişteki kararlı bir duruma döndürülerek koşuma bu noktadan devam edilebilir ve hataya neden olan durum gözlemlenebilir.

1.1.3. Çevrimdışı Analiz:

Çevrimdışı analiz, toplanan benzetim verilerinin benzetim sonrasında analiz edilerek gerek benzetim sisteminin performansının ve gerekse de ağ yoğunlukları gibi donanımsal sorunların incelenmesidir Çevrimdışı analiz, ARR'dan farklı olarak benzetimin doğrudan kendisinin analizinin yapılmasını ifade etmektedir. Bu sayede veri alışverişinde sorun yaşayan ya da işlem yükü nedeniyle diğer öğelere gerekli verileri zamanında yollayamayan benzetim öğeleri tespit edilebilir ve buna karşı önlem alınabilir. Benzetim öğelerinin yolladığı ve aldığı veriler boyut ve sayı yönünden incelenerek bu analiz gerçekleştirilebilir.

1.1.4. Hata Ayıklama - Birim ve Modül Test:

Dağıtık benzetim öğelerinin geliştirilmesi aşamasında, diğer benzetim öğelerinden gelmesi beklenen veriler önem kazanmaktadır. Geliştirilecek benzetim öğesi bu verileri girdi olarak kullanacak ve koşumu buna göre şekillenecektir. Dolayısıyla geliştirilen öğenin öncelikle bu veriler kullanılarak test edilmesi gerekir. Diğer öğelerden bu verileri sağlamak geliştirilen öğenin benzetime dâhil edilmesi ile olabilir. Fakat bu yöntem diğer öğelerin doğru şekilde çalıştıklarını varsaymakta ve bütün öğelerin beraber sorunsuz çalışabilmesini gerektirmektedir. Ayrıca daha test aşamasından geçmemiş bir öğenin benzetime dâhil edilmesi ile benzetimin yanlış çalışmasına sebep olunabilir. Bir başka yöntem, diğer öğelerle benzer davranışta bulunacak daha basit modelleri içeren öğeler geliştirilmesi yeni öğenin bu öğelerle birlikte çalıştırılması olabilir. Bu yaklaşımda ise hem gerekli olan diğer bütün öğelerin modellerinin anlaşılıp basitleştirilmesi hem de bir öğe için birden çok öğenin geliştirilmesi gerekmektedir.

Diğer bir yöntem ise benzetim öğelerinin yayınladığı/aldığı verilerin kaydedilmesi ve geliştirilmesi düşünülen öğeye bu verilerden ihtiyaç duyduklarının girdi olarak verilmesi olabilir. Böylece diğer benzetim öğelerine ihtiyaç duyulmadan geliştirilen öğe bu verilerle tekrar tekrar test edilebilir. Geliştirme sırasında yapılan değişikliklerin etkileri gözlemlenebilir, bir hata yakalanması durumunda hangi zamanda, hangi veri geldikten sonra sorunun gerçekleştiği görülerek hata ayıklama kolaylaştırılabilir.

1.2. Problemler

Yukarıda sayılan ihtiyaçlar, daha güvenilir ve daha etkin benzetim sistemlerinin geliştirilmesi için benzetim verilerinin toplanmasının oldukça gerekli olduğunu ortaya koymaktadır. Fakat dağıtık benzetim sistemlerinin doğası gereği, bunu kolay ve verimli bir şekilde sağlamak mümkün olmayabilmektedir. Farklı donanımlar üzerinde birden çok benzetim ögesinin yayınladığı/aldığı verileri kaydetmek gerek verilerin tutarlılığı gerekse performans açısından sorunlar içermektedir. Ayrıca, her ne kadar çeşitli mimariler ile benzetim sistemlerinin aynı yapıya getirilmesine çalışılsa da her sistemin kendine özgü davranışları sonucu, tüm sistemlerde uygulanabilen, ölçeklenebilir bir veri toplama sistemi oluşturmak güçleşmektedir. Her benzetim sisteminde farklı tipte, boyutta ve sıklıkta veriler iletebilmesi bu değişken veri miktarına ve farklı veri tiplerine uygun bir sistem geliştirmeyi karmaşıktırmaktadır. Kullanılan ağ yapısı ve büyüklüğü de ayrı bir parametre olarak veri toplama sisteminin ölçeklenebilirliğini etkilemektedir. Örneğin yerel ağ (Local Area Network-LAN) üzerinde yeterli performansta ve doğrulukta çalışabilen bir sistem, geniş alanlı bir ağ (Wide Area Network-WAN) üzerinde oluşan ağ gecikmeleri sebebiyle hem performans hem de verilerin zamanlamaları konularında sorun yaşayabilir.

Veri kayıt sistemleri, benzetim ögesinin diğer ögelere yollamadığı, kendine özel ama benzetim ögesinin işleyişi açısından ya da analiz için önemli olan verilerin kaydına da imkân sağlaması gerekmektedir. Ancak bu şekilde yukarıda saydığımız analiz, geçmişe dönme ve hata ayıklama gibi ihtiyaçlar tam olarak sağlanabilir. Çünkü benzetim ögesinin doğru koşumu sağlaması kendi içerisinde bulunan değişken ve parametrelere doğrudan bağlıdır.

Literatürde bu benzetim verilerinin kaydı konusunda pek çok çalışma yapılmış fakat yapılan çalışmalar daha çok benzetim ögelerinin birbirine yolladığı mesajların kaydı üzerine yoğunlaşmış ve veri kaydının en verimli şekilde sağlanması için çeşitli metotlar/mimariler önermişlerdir. 2.1 bölümünde daha detaylı incelenen bu mimarilerin bir kısmı özellikle ölçeklenebilirlik açısından istenen performansı sağlamakta güçlük çekebilmektedir. Sunulan önerilerden çok azı [Magee & Shanks, 1999] tam dağıtık bir yapıyı öngörmüştür. Fakat bu çalışmalarda da özellikle tekrar oynatma sırasında benzetim ögelerinin birbirleri ile eşzamanlı çalışmasının

sağlanmasından bahsedilmemektedir. Hâlbuki özellikle eğitim amaçlı benzetim sistemlerinde tekrar oynatma ancak bütün benzetim ögelerinin birbiri ile eşzamanlı olarak ilerlemesi ile anlam kazanacaktır. Örnek olarak bir savaş benzetim sisteminde hedef ve mermi olarak iki benzetim ögesinin var olduğunu kabul edelim. Normal çalışma sırasında merminin hedefe çarpması ile hedefin patladığı bir senaryo olsun. Eğer tekrar oynatma sırasında hedef mermi çarpmadan önce patlarsa tekrar oynatma amacına ulaşamaz ve olayların nasıl geliştiği gibi analizler tam olarak sağlanamaz.

Yukarıda belirtildiği gibi yapılan çalışmalarda daha çok benzetim ortamında ögelerin birbirine yolladığı mesajlar kaydedilmekte, benzetim ögesinin özel verilerinin toplanması için çözüm sunulmamaktadır. Hâlbuki özellikle benzetim geçmişine dönmek ve detaylı bir analiz yapabilmek için bu veriler oldukça önemlidir. Benzetim geçmişine dönmek aslında her bir benzetim ögesinin kendi geçmiş durumuna dönmesi ile mümkün olabilir. Dolayısıyla, her bir ögenin kendisinin geçmişteki durumunu tam olarak bilmesi gerekir. Bu da özel verilerinin de kaydı ile mümkün olabilir.

1.3. Öneri

Bu çalışmada, dağıtık benzetim sistemlerinde yukarıda sayılan ihtiyaçları karşılamak için önceki çalışmalardan farklı olarak her bir benzetim ögesi ile bütünleşik bir kayıt mimarisi önerilmektedir. Bu mimari ile işlemci ve ağ iletişimi yükü en aza indirilmiş, tam dağıtık bir yapı ile ölçeklenebilirlik ve bağımsız gerçekleştirme, test ve analiz sağlanmış ve tekrar oynatma sırasında sağlanan eşzamanlılık ile analiz işlemleri kolaylaştırılmıştır. Tam dağıtık kayıt ve yeniden oynatmanın temel problemlerinden olan eşzamanlık problemi için bir çözüm önerilmiştir. Ayrıca geçmişe dönme sırasında ve detaylı bir analiz için gerekli olan benzetim ögesinin özel verilerinin nasıl kaydedileceği üzerinde durulmuş ve gerçekleştirilen sistem anlatılmıştır. Önerilen mimarinin nasıl gerçekleştiği anlatılmış, farklı yazılım kütüphanelerine uygun ve benzetim ögesine kolaylıkla bütünleştirilebilen bir gerçeklemenin nasıl sağlandığı üzerinde durulmuştur. Daha sonra, geliştirilen sistem performans ve doğruluk açısından deneylere tabi tutulmuş ve kabul edilebilir bir performans sunduğu görülerek tutarlı ve doğru sonuçlar verdiği izlenmiştir.

Bu tez çalışmasında öncelikle 2. Bölümde dağıtık benzetim sistemleri ve mimarilerinden bahsedilmiş; 3. Bölümle beraber bu sistemlerde kayıt ve yeniden oynatmanın nasıl olması gerektiği ve bunu gerçeklemedeki zorluklar incelenmiştir. 3.1 bölümünde daha önce yapılan çalışmalar incelenerek ihtiyaç duyulan sistem açısından bu çalışmaların artı ve eksileri üzerinde tartışılmıştır. 3.3 bölümünde öneminden bahsedilen benzetim ögesine özel verilerin kaydedilmesi için önerilmiş çözümler üzerinde durulmuştur. 4. bölüm, önerilen mimarinin anlatımını, eşzamanlılığın nasıl sağlanacağı ve özel verilerin kaydının hangi yöntemle gerçekleştirileceğini içermektedir. 5. bölümde ise gerçekleştirme detayları üzerinde durulmuştur. Deneysel çalışmalar performans ve doğruluk deneylerini içerecek şekilde 6. bölümde anlatılmıştır. 7. bölümde deneylerden alınan sonuçlar ışığında, gerçekleştirilen sistemin olabilirliği ve başarımı incelenmiş, gelecekte hangi kısımlarda ne gibi çalışmalar yapılabileceği anlatılmış ve çalışmanın genel incelemesi ile tez çalışması sonlandırılmıştır.

2. DAĞITIK BENZETİMLER

Benzetimlerin sayılan geniş kullanım alanları zamanla daha karmaşık gerçek sistem sistemlerin benzetimlerinin gerçekleşmesini ve benzetimlerden beklenen yeteneklerin artmasını sonuç vermiştir. Gerçek sistemin modeli karmaşıklıkla benzetim için ihtiyaç duyulan işlem miktarı artmış ve tek bir bilgisayarın işlem kapasitesini aşmıştır. Ayrıca benzetim sisteminin birden çok model ve/veya yazılım/donanım bileşeninden oluşması her bir ögenin farklı geliştiriciler/firmalar tarafından geliştirilmesini gerektirmiştir. Bunların yanında kullanıcıların Dünya'nın farklı yerlerinde geniş alanlı bir ağ (Wide Area Network-WAN) üzerinden aynı benzetim üzerinde buluşması da özellikle eğitim amaçlı benzetimlerde ortaya çıkmış bir ihtiyaçtır. Böylece, kişiler tek bir yere toplanmaya ihtiyaç duymadan benzetim üzerinden eğitim alabilmesi amaçlanmıştır.

Benzetim sistemlerinin, geniş kabul görmeye başladıkça daha karmaşık gerçek sistemlerin benzetimi için tercih edilmeye başlandığından ve farklı yerlerdeki kullanıcıların aynı benzetim ortamında birleştirilmek istenmesinden giriş bölümünde bahsedilmişti. Benzetim sisteminin birden çok model ve/veya yazılım/donanım bileşenini içermesi ve her bir ögenin farklı geliştiriciler/firmalar tarafından geliştirilmesini gerekebildiği de vurgulanmıştı. Bunlar dağıtık benzetimlerin ortaya çıkmasındaki en temel ihtiyaçlar olmuştur. Böylece çeşitli şekillerde birbirleri ile haberleşen benzetimler/benzetim öğeleri geliştirilmiş ve bunların birbirleri ile zamanlama ve veri iletimi açısından uyumlu şekilde koşturulmaları sağlanmaya çalışılmıştır.

Her bir benzetim ögesinin giderek gerçek sistemi daha iyi modellemesi sağlanmaya çalışılmış ve bunun neticesinde her bir model daha özel bir alan bilgisi gerektirmiştir. Farklı üreticiler, kuruluşlar belirli alanlara yoğunlaşarak daha büyük bir benzetimin alt parçaları sayılabilecek modellerin benzetimini yapmaya çalışmışlardır. Örneğin bir savaş benzetiminde yer alan tank, uçak, hatta insan modelleri farklı firmaların ayrı ayrı üzerine yoğunlaştıkları modeller olmuştur. Bunun yanında zaman ve çalışan sayısı gibi kısıtlar da her bir modelin farklı firmalar tarafından yapılmasını gerektirmiştir.

Ayrı ayrı geliştirilen benzetim öğelerinin birbirleri ile uyumlu bir şekilde iletişim kurlmaları için ortak bir haberleşme mimarisi gerekmiştir. Ayrıca geliştirilen modellerin tekrar tekrar kullanılabilmesi için de ortak bir mimari ile gerçeklenmeleri gerektirmiştir. Bunun neticesinde dünya çapında üzerinde anlaşılmış benzetim standartları/mimarileri geliştirilmiştir.

2.1. Benzetim Mimarileri/Standartları

Benzetim öğelerinin birbiri ile haberleşmelerinin daha uyumlu olması, tekrar kullanılabilirliklerinin artması amacıyla çeşitli mimariler ve standartlar literatüre sunulmuş ve kabul görmüştür. SIMNET, ALSP, DIS ve HLA bunların bazılarıdır. İlk başlarda benzetim öğelerinin birbirleri ile aynı yapıda olabilmesi ve uyum içerisinde çalışması amacıyla geliştirilen bu mimariler/ standartlar, zamanla mimari kullanımının getirdiği ağ ve işlemci yükleri gibi fazladan maliyetleri azaltmak yönünde geliştirilmiştir.

Bunların arasında günümüzde yaygın olarak kullanılan HLA (High Level Architecture/Yüksek Seviyeli Mimari) benzetim öğeleri arasındaki etkin ve az maliyetli (ağ ve işlemci kullanımı yönünden) iletişim olanağı, tekrar kullanılabilirliğe daha açık olması gibi özelliklerle öne çıkmıştır. Zaman yönetimi hizmeti ile benzetim öğelerinin birbirleri ile olan eşzamanlılıklarının tam olarak sağlanabilmesini sağlarken veri dağıtım hizmeti kullanarak ve yayın/abone sistemi ile benzetim ortamındaki nesnelerin sahipliklerinin yönetilerek bir nesnenin verilerinin farklı benzetim öğeleri tarafından yayınlanabilmesi gibi kabiliyetlerle de önemi artmıştır. DIS (Distributed Interactive Simulation/Etkileşimli Dağıtık Benzetim) ise hâlâ yaygın olarak kullanılan bir diğer benzetim mimarisidir. Bu mimaride benzetim öğeleri, PDU (Protocol Data Unit/İletişim Kuralı Veri Birimi) adı verilen mesajlarla birbirleri ile haberleşmekte ve benzetim öğelerinin eşzamanlılığı sabit aralıklarla gönderilen mesajlarla sağlanmaktadır.

DIS ve HLA günümüzde en yaygın olarak kullanılan iki mimari olduğundan ve ileriye dönük yapılan çalışmaların da bu iki mimarinin üzerinden gitmesi sebebiyle üzerlerinde durmak gerekir. Tarihsel olarak bakıldığında HLA'nın DIS'den daha sonra ortaya çıktığı ve DIS'de karşılaşılabilen performans sorunlarına çözüm

bulmayı ve benzetim öğeleri arasındaki birlikte işlerlik özelliklerinin arttırılmasını ve tekrar kullanılabilirliği sağlamayı amaçladığı düşünülebilir. DIS'te, her bir benzetim öğesi yukarıda bahsettiğimiz PDU'ları belli zaman aralıklarında bütün olarak tekrar tekrar yollamak durumundadırlar. Bu yollanan veriler doğrudan bir bilgisayara değil, “çoklu gönderim (multicast)” ile ağdaki bütün benzetim öğelerine gönderilmektedir. Bunun sonucunda bütün benzetim öğelerinin yolladığı mesajlar ağda dolaşmakta ve ciddi bir ağ yükü oluşturabilmektedir. HLA, bu sorunları çözmek için çeşitli çözümler ortaya atmıştır. Bunları bir sonraki bölümde detaylandıracağız.

2.1.1. HLA

HLA'da dağıtık benzetim öğelerinin her biri “Federe” olarak adlandırılır. Bu federelerin bir araya gelerek oluşturdukları dağıtık benzetim sistemine ise “Federasyon” adı verilmektedir. Federeler, federasyon ortamında (belli bir süre için de olsa) kalıcı olacak öğelerini “Nesne (Object)” olarak oluşturur ve bu nesnelere ait özellikleri (attributes) istedikleri zamanlarda güncelleyerek diğer federelere bu bilgiyi iletirler. Diğer kalıcı olmayan anlık bilgi iletimini hedefleyen haberleşmeler ise “Etkileşim (Interaction)” adı verilen yapılarla gerçekleştirilir. Nesnelerin ve etkileşimlerin isimleri, özellikleri ve veri tipleri FOM (Federation Object Model / Federasyon Nesne Modeli) adı verilen, yazılım tarafından okunabilen bir doküman içerisinde tanımlanacaktır. Böylece federasyondaki her bir federe, yayınlanacak verilerin adlarının ve içeriklerinin ne olacağını bilebileceklerdir.

HLA, yayınlanması düşünülen bir nesnenin özelliklerinin bütününün veya bazılarının yayınlanabilmesine imkân sağlamakta ve böylece sadece değişen veya önemli özelliklerin yayınlanabilmesi ile ağda gereksiz yük oluşmasını engellemektedir. Ayrıca yayın/abone (publish/subscribe) sisteminin kullanılmasını zorunlu kılarak bir benzetim nesnesinin güncellemelerini almak isteyen bir federenin, ancak bu nesneye (ya da nesnesinin belli özelliklerine) abone olarak bu verileri alabilmesini sağlamıştır. Böylece, veriler sadece abone olan federelere gönderilir, çoklu gönderim yapılmaz ve bu sayede ağ trafiği ciddi oranda azalabilir.

Her bir federenin yayınlayacağı ve abone olduğu nesne ve etkileşimler FOM'a benzeyen bir dosya olan SOM (Simulation Object Model/Benzetim Nesne Modeli) dokümanında belirtilir. Böylece bir federenin benzetim sistemi ile olan arayüzü tam

olarak tanımlanmış olur. HLA, bu sayede federelerin taşınabilirliğini ve tekrar kullanılabilirliğini arttırmayı sağlamıştır.

Federasyondaki iletişimi, federasyonun oluşturulmasını/sonlandırılmasını, federelerin federasyona katılmalarını ve ayrılmalarını RTI (Run-time Infrastructure/Koşum zamanı Altyapısı) adlı yazılım sağlamaktadır. RTI, bütün federelerin bağlı oldukları benzetimi yöneten bir öge olarak görülebilir. Zaman yönetimi ile federeler arasındaki eşzamanlılığın sağlanması da RTI tarafından sağlanmaktadır. Federeler RTI'dan benzetim zamanına ilerlemek için istekte bulunmakta, RTI da diğer federelerin istek ve durumlarını da göz önüne alan bir yöntemle istenilen zamana ilerleme izni vermektedir. Zaman yönetimi istenilirse devre dışı da bırakılabilir. Bu durumda federeler ilk gelen veriyi işleme esasına göre verileri almakta ve yollamaktadır. Bu durumda federeler arası eşzamanlılık sağlanmamaktadır. Yukarıda bahsi geçen yayın/abone mekanizması da RTI üzerinden sağlanır. Bir federe hangi nesneleri/etkileşimleri göndereceğini RTI'ya bildirir. Daha sonra başka bir federe bu nesne/etkileşime abone olduğunu RTI'ya bildirdiğinde, RTI gerekli çoklu gönderim(multicast) bağlantılarını ayarlayarak ilgili verinin yayınlayan federeden abone olan federeye iletilmesini sağlar. Federasyona ait diğer bütün işlemler yine RTI üzerinden sağlanmaktadır.

Federenin RTI ile iletişimi esas olarak iki sınıf üzerinden gerçekleşmektedir. HLA standartlarına göre bu iki sınıf RTIAmbassador ve FederateAmbassador'dur [IEEE Std 1516.1-2000]. RTIAmbassador federenin RTI katmanına yapacağı istekleri, RTI hizmetlerini sağlarken FederateAmbassador RTI'nın federeye geribildirim yapmasını sağlayan bir sınıftır. Örneğin, federe bir etkileşim göndermek istediği zaman RTIAmbassador'un *sendInteraction* fonksiyonunu kullanırken abone olduğu bir etkileşimin kendisine ulaşması FederateAmbassador'un *receiveInteraction* fonksiyonunun RTI tarafından çağırılması ile sağlanır.

3. DAĞITIK BENZETİM SİSTEMLERİNDE KAYIT VE YENİDEN OYNATMA

3.1. Kayıt Bakış Açısından HLA

HLA'nın önceki bölümde sayılan özellikleri her ne kadar verimliliği ve iletişim yeteneklerini arttırsa da veri toplamayı zorlaştırmaktadır. DIS mimarisinde bütün veriler belli aralıklarla benzetim ortamına aktarıldığından ağ trafiğini dinleyerek benzetime ait verilerin önemli ve yeterli bir kısmını elde etmek mümkün olabilmektedir. Veri zaten ağda bulunduğundan fazladan bir ağ trafiği oluşturmadan bu veriyi almak mümkündür. Benzetim öğelerine belli aralıklarla gönderilen sinyaller ile de eşzamanlılık sağlanabilmektedir.

HLA'de ise bu mimarinin getirdiği eniyileme ve standartlar sebebiyle kayıt işlemleri daha zor ve/veya maliyetli gerçekleştirilebilmektedir. Öncelikle HLA'da çoklu gönderme yerine doğrudan veriyi isteyen federeye verinin gönderilmesi, kayıt yapacak öğenin de (eğer bir federe olacaksa) RTI'dan bu veriyi özellikle istemesini gerektirmektedir. Bu ise verinin fazladan bu öğeye/federeye de gönderilmesini gerektireceğinden ağda bir yük oluşturmaktadır. Ayrıca nesnelerin kısmi olarak güncellenebilmesi kayıt verisinde nesnelerin bütünlüğünü sağlamayı zorlaştırmakta ve DIS'deki gibi belli aralıklarla gelen bir sinyalin olmaması tutarlı kayıtlar oluşturmayı engelleyebilmektedir. Herhangi bir anda bütün federelerin aynı benzetim zamanında olmayabilirler ve bunun neticesinde kayıt edilen verilerin zaman sıralamaları yanlış olabilir. Ayrıca FOM'da her federasyon kendine özgü veri tipleri tanımlayabileceğinden kayıt sisteminin bunu da göz önüne alarak genel bir çözüm sunması gerekmektedir.

HLA'nın bu ve diğer özellikleri göz önünde bulundurulduğunda yapılacak kayıt ve yeniden oynatma sisteminin şu özellikleri taşıması gerektiği söylenebilir:

1. Ağ trafiğini etkilememeli, fazladan trafik oluşturmamalı;
2. Ölçeklenebilir olmalı; federe sayısı, federelerin bulunduğu ağ büyüklüğü (LAN/WAN), gönderilen veri miktarı, ortamdaki nesne/etkileşim sayısı gibi etkenleri karşılayabilmeli;

3. Zaman yönetimine bağlı olmadan eşzamanlılığı sağlayabilmeli;
4. Farklı tipte (veri tipleri, çalışma yöntemleri...) federasyonlarda çalışabilecek; birlikte çalıştığı federasyonun özniteliklerinden bağımsız olmalı;
5. Çalıştığı federasyonun yazılım kodunda ciddi değişiklikler gerektirmeyecek ve kolaylıkla aktif/pasif hale getirilebilmeli;
6. Kullanıcı girişleri, oluşturulan rastgele sayılar gibi federeye özgü, federenin diğer federelere göndermediği özel verileri de kaydedebilmelidir.

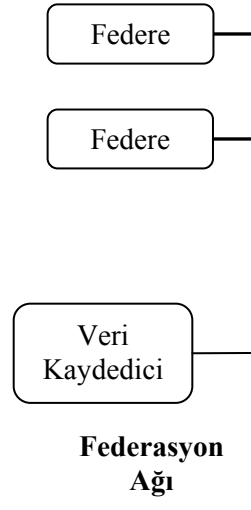
3.2. Yapılan Çalışmalar, Mimariler

Genel olarak tüm dağıtık benzetim sistemleri için düşünülebilse de özellikle HLA tabanlı benzetim sistemlerinde kayıt mimarisi için kolay gerçekleştirilebilirlik, ölçeklenebilirlik, kayıt verilerinin yönetimi, eşzamanlılık gibi özellikler göz önünde bulundurularak çeşitli mimariler önerilmiştir. Bu mimariler kaydedici ögenin bulunduğu konum, bu ögelerin sayısı ve yetenekleri göz önüne alınarak gruplandırılmıştır [Magee & Shanks, 1999], [Wilcox et al., 2000], [Van Hook & Calvin, 1997], [Li et al., 2006]. Bu çalışmada da esas alınan bu gruplandırma şu şekildedir:

- Merkezi Kayıt
- Kısmen Dağıtık Kayıt
 - Standart Yapı
 - Ajan Tabanlı Yapı
- Tam Dağıtık Kayıt

3.2.1. Merkezi Kayıt

Bu yapıda, benzetim ortamında tek bir kaydedici bulunur. Yayınlanan bütün verileri bu, tek kaydedici kaydederek tekrar yayınlamak, analiz etmek gibi amaçlara uygun olarak hazırlar (Şekil 3.1).



Şekil 3.1 Merkezi Kayıt

Bu yapının esas avantajı, kayıt verilerinin kayıt sonrası tek bir yerde hazır bulunabilir olması ve tekrar oynatırken tek bir noktadan veriler gönderileceği için eşzamanlılık gibi problemlere sebep vermemesidir. Ayrıca kayıt verilerinin tek bir yerde hazır olması ile bu veri analiz için kolaylıkla hazırlanabilir, bütün veriler tek bir seferde işlenerek istenilen özellikler çabuk bir şekilde incelenebilir.

Diğer taraftan bu yapı ile ağda tek noktada yoğun bir trafik ve tek bilgisayarda yoğun bir işlem yükü oluşmaktadır. Bu ise hem kaydederken hem de tekrar oynatırken sorun olabilmektedir. Dolayısıyla, küçük çaplı benzetimler için etkin ve hızlı bir çözüm olmasına karşın, büyük ve özellikle WAN üzerinden uzak mesafelerden haberleşen benzetim öğelerinin olduğu bir benzetimde verimsiz olan bir mimari yapıdır. Literatür araştırması niteliğinde olan çalışmalarda da merkezi kayıt ve yeniden oynatmanın bu özellikleri üzerinde durulmuş ve daha çok küçük çaplı dağıtık benzetim sistemleri için önerilmiştir [Wilcox et al., 2000], [Bachinsky et al., 1999], [Van Hook & Calvin, 1997].

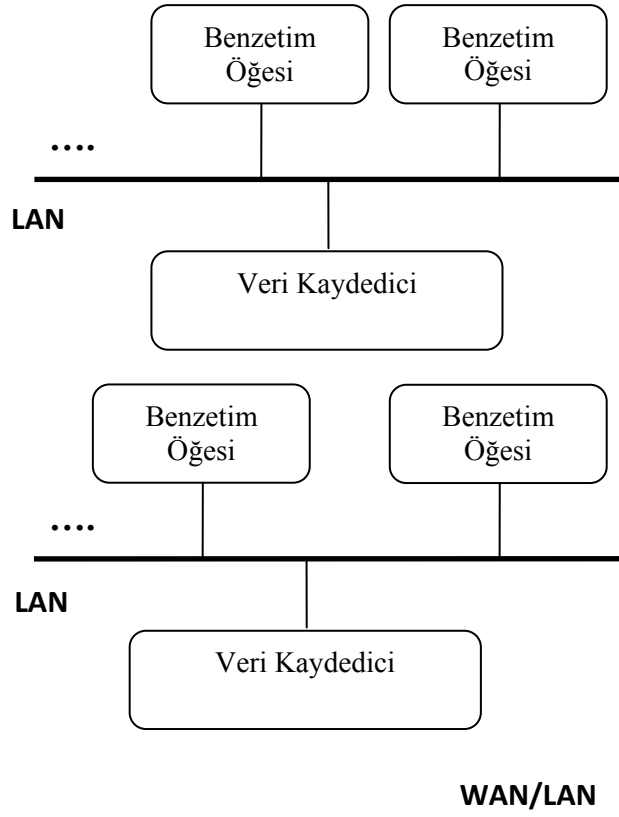
Bu mimaride özel verilerin federeden alınarak kaydı doğrudan mümkün değildir. Benzetim öğesinden bu verilerin alınmasının tek yolu benzetim öğesinin bu verileri diğer benzetim verileri gibi belli aralıklarla yayınlamasının sağlanmasıdır. Bu ise gerek ağ yükünü arttırması, gerekse işlemci kapasitesinin bu boyutlarda büyük verileri işlemesinin sorun olabileceği için pratik bir uygulama değildir. Sadece özel

verilerin merkezi kaydını yapan ayrı bir kaydedici olsa bile özel verilerin çok büyük boyutlarda olabileceğinden bu şekilde de etkin bir kullanımı olamayacağı açıktır.

Bu yapı ile veriler tek bir merkezde kayıt edilmeye çalışıldığından ağ gecikmeleri, aşırı yüklenme gibi sorunlarla kaydedilecek verilerin ağdaki paket kayıpları ile kaydediciye hiç ulaşamaması ya da doğru zaman bilgilerine sahip olamaması gibi sorunlar taşıdığından bahsedilmişti. Tekrar oynatmada bütün veriler tek bir noktadan geri yayınlandığından bu sorun belirgin olarak gözlemlenmeyebilir. Fakat bu verileri kullanarak herhangi bir benzetim ögesini test etme, hata ayıklama gibi işlemleri gerçeklemek mümkün olmayabilir. Veriler doğru zamanda doğru sırayla test edilecek federeye sağlanamazsa tekrarlanabilirlik göz ardı edilmiş olur ve elde edilen sonuçlar normal koşumdakinden farklı olabilir. Dolayısıyla merkezi kayıt, test ve hata ayıklama işlemleri için güvenilir veriyi sağlamakta güçlük çekmektedir.

3.2.2. Kısmen Dağıtık Kayıt

Bu yapıda, benzetim ortamında birden çok kaydedici bulunur (Şekil 3.2). Her bir kaydedici yayınlanan verinin bir bölümünü kaydetmekten sorumludur. Bu yapıda esas amaç kaydedici başına düşen yükün azaltılmasını sağlamaktır. Ayrıca ya insan yardımı ile ya da kaydedicinin kendi karar mekanizması ile her bir kaydediciye düşen yük eşitlenmeye çalışılır. Kısmen dağıtık kayıt ve yeniden oynatma, kaydedicilerin içsel özelliklerine göre standart yapı ve ajan tabanlı yapı olarak ikiye ayrılabilir.



Şekil 3.2 Kısım Dağıtık Kayıt

Standart Yapı:

Bu yapıda kullanıcı tarafından benzetim verilerinin hangi kısmını kaydedeceği belirlenmiş olan kaydediciler bulunur. Özellikle bölütlenmiş ağlarda her bir bölümün bir kaydedici tarafından kaydedilmesi sağlanarak kaydetmeden dolayı oluşan ağ trafiği en aza indirilmiş ve her bir bölümdeki benzetim öğelerinin o bölümdeki kayıt verisi ile bağımsız test ve analiz imkânı sağlanmış olur.

Bu yapının sorunları ise, benzetimin tam olarak tekrar oynatılması için ya farklı yerlerde olan verinin bir yere toplanması ve tekrar verilerinin de çıkarılmasını içeren bir birleştirme işleminden geçmesi ya da her bir kaydedicinin kendi kaydettiği verilerle benzetimi oynatması gerekir. İlk yöntemde verilerin birleştirilmesinin zaman sorunu yaratabileceği açıktır. İkinci yöntemde ise bu kaydedici/oynatıcıların birbirleri ile senkronize olabilmesi ve doğru zamanlama ile verileri yollaması gerekir.

Ajan Tabanlı Yapı:

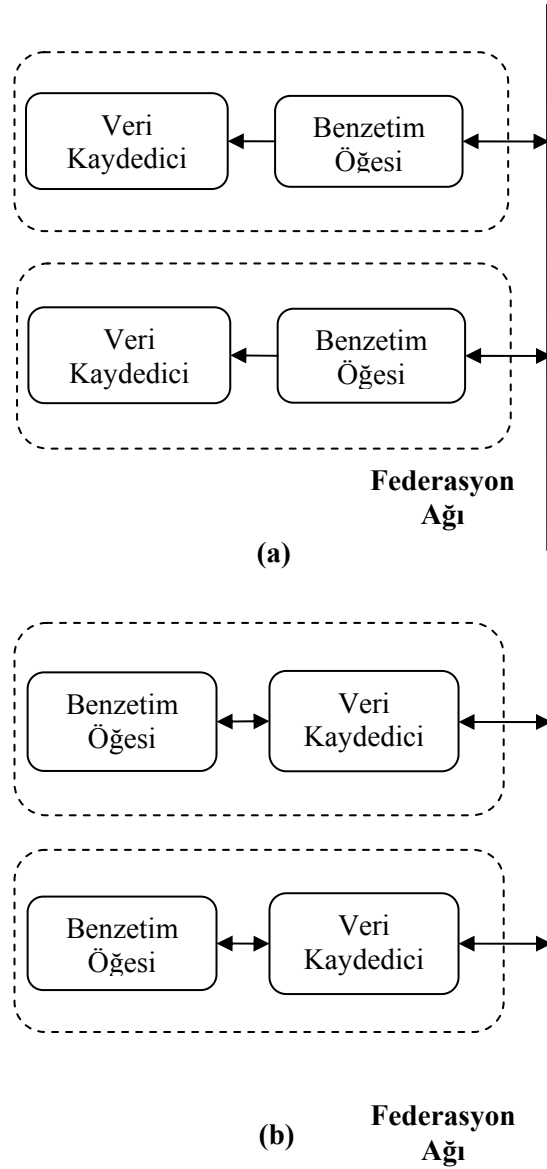
Bu yapıda kaydediciler kayıt esnasında birbirleri ile haberleşerek hangi verinin kimin tarafından kaydedileceğine karar verecek ve eşit bir yük dağılımı sağlayacaklardır.

Benzer bir sistem daha önce literatüre sunulmuştur [Song et al., 2007]. Bu çalışmada kaydediciler, yük durumlarına göre gelen verinin kaydedilip kaydedilmeyeceğini belirleyen modüller içermektedirler. Eğer kaydedici gelen veriyi kabul edemeyecekse bu veriyi kaydedilmesi için diğer kaydedicilere yollar.

Bu yapıda, esasında ağ bölütlenmesi dikkate alınmamaktadır. WAN üzerinden haberleşmenin olduğu benzetimlerde bu durum sorun oluşturabilir. Bu yapının diğer bir dezavantajı ise yük dengelemesi için kayıt esnasında kaydedilecek verinin paylaşımının değişmesi sonucu oluşan parçalı kayıt verisidir. Örneğin A bilgisini 1 numaralı kaydedici kaydediyor olsun. Belli bir zaman sonra yük durumu değişince 1 numaralı kaydedici A bilgisini 2 numaralı kaydediciye iletir. Benzetim bittiğinde A bilgisinin bir kısmı 1 numaralı kaydedicide, bir kısmı da 2 numaralı kaydedicide olacaktır. Tekrar oynatmanın doğru işleyebilmesi için A bilgisinin 1’den 2’ye geçişi de kaydedilmeli ve bu dikkate alınarak veriler benzetim ortamına yayınlanmalıdır. Dolayısıyla bu tip veri sahipliği bilgilerinin kaydı ve yönetimi gereklidir. Ayrıca, bir benzetimin kayıt edilmiş nesnelerinin bir kısmını bir kaydedici, bir kısmını da başka bir kaydedici kaydetmişse tekrar oynatırken bu nesnenin nasıl oluşturulacağı da dikkate alınması gereken bir problemdir.

3.2.3. Tam Dağıtık Kayıt

Bu mimaride, her bir benzetim ögesi kendi kaydedicisine sahiptir (Şekil 3.3). Bu mimarinin esas amacı hem ölçeklenebilirliği korumak hem de fazladan ağ yükü oluşturmayı engellemektir. Her bir benzetim ögesi kendi veri kaydedicisindeki verileri kullanarak diğer ögelerden bağımsız olarak hata giderme, test ve analiz işlemlerini gerçekleştirebilir.



Şekil 3.3 Tam Dağıtık Mimari

Bu yapıda, her bir benzetim ögesi diğer benzetim ögelerine ihtiyaç duymadan tekrar oynatılabilir. Fakat bütün benzetimin tekrar oynatılması isteniyorsa ya bu bağımsız veri kaydedici/oynatıcılar arasındaki eşzamanlılık korunarak her biri kendi kaydettiği verileri benzetime gönderirler ya da bütün veriler tek bir yerde toplanarak tek noktadan yayın yapılır. Bu durumda ya birbirinden bağımsız kaydedicilerin haberleşmesi sağlanarak eşzamanlılık mekanizması gerçekleştirilmeli ya da verileri tek noktada birleştirme sırasında oluşacak tekrar edilen verilerin temizlenmesi, verilerin sıralanması gibi sorunlar çözülmelidir. İkinci durumda ayrıca verilerin tek noktaya iletilmesi, büyük boyutlardaki verilerin çabuk bir şekilde işlenmesi gibi problemler

vardır. Analiz gibi ihtiyalar iin btn benzetim ğelerinin verisine ihtiya duyulabilmekte ve yukarıda sayılan işlemler gerekleřtirilerek ayrı ayrı olarak toplanmış verilerin birleřtirilmesi gerekmektedir. Fakat bu durumda analiz işlemini gerekleřtirmek iin benzetimin kořumundan sonra belli bir zaman beklemek gerekecektir ve dolayısıyla hızlı bir analiz imkânı zorlařacaktır.

Bu mimari iki farklı řekilde gereklenebilir:

- Benzetim gesi aldıđı verileri veri kaydedicinin bazı fonksiyonlarını ađırarak kaydedebilir (řekil 3.3(a)). Bu durumda diđer benzetim ğeleriyle haberleřme benzetim gesinin dođrudan zerinden gerekleřmekte ve bunun neticesinde mesajları almak, işlemek ve dođru zamanda veri kaydedicinin dođru fonksiyonlarını ađırarak gelen verileri kaydetmek benzetim gesinin sorumluluđunda bulunmaktadır. Bu durumda benzetim gesinin geliřtiricisi kayıt işlemlerini her adımda dikkate almak ve dođru zamanlarda kayıt fonksiyonlarını ađırdıđından emin olmak durumundadır.

Veri kaydedici benzetim gesi ile benzetim ađı arasında yer alabilir (řekil 3.3(b)). Bu yntemde benzetim ortamı ile olan btn mesajlařma veri kaydedicinin zerinden geer. Bylece benzetim gesinde deđiřiklik yapılmadan veri kaydetme/yeniden oynatma sađlanabilir. Genelde benzetim ğeleri diđer ğelerle haberleřmek iin bir yazılım ktphanesi kullandıđından bu ktphanenin arayzlerini koruyan ama aynı zamanda kayıt ve yeniden oynatma işlemlerini gerekleřtiren yeni bir yazılım ktphanesi oluřturulabilir. Bu yntemle benzetim gesi kayıt işlemlerinden soyutlanmış olacak ve bu işlemleri dikkate almak durumunda kalmadan geliřtirilebilecektir.

Tam dađıtık kayıt zel verilerin kaydına en uygun yapıdır. Kaydedici, benzetim gesinin bir parası konumunda olduđundan rahatlıkla benzetim gesinin zel verilerine ulařabilir. Bu verilerin ađ ortamına yollanmasına gerek kalmadan hızlı ve daha az maliyetli bir řekilde kaydedilmesi sađlanabilir. Ayrıca veriler ađ zerinden gnderilmediđi iin daha byk veriler kaydedilebilecektir.

3.3. Yerel/Özel Veriler

HLA’da bir benzetim ögesine ait verinin sadece gerekli kısımları sadece ilgili benzetim öğelerine gönderilmektedir. Dolayısıyla bir veri kaydedici, istenilen verilere ayrıca “abone olup” ilgili benzetim ögesinden istemek durumundadır. Fakat benzetim ögesi, kendi verisinin sadece diğer öğeleri ilgilendiren kısımlarını yolladığından bu verilerin toplanması benzetimin geçmişe dönmesi, analiz edilmesi ve hata ayıklanması için yeterli olmaz. Tabii ki bu özel verilerin her bir benzetim ögesi tarafından benzetim ağına yollanması sağlanarak gerekli bütün veri toplanabilir. Fakat bunu yapmak ağda çok ciddi bir yük oluşmasına ve yaşanacak ağ gecikmeleri neticesinde benzetimin tutarsız çalışmasına sebep olacaktır. Zaten böyle bir yöntem, HLA’nın daha az ağ yükü gibi kullanım amaçlarının tersine bir kullanım olacaktır.

Veri kaydedicinin bir şekilde benzetim ögesine bütünleşmiş olduğu ve benzetim ögesinin özel verilerine erişebildiği öngörülürse farklı alternatifler ile HLA mimarisi kullanılırken bu özel verilerin kaydedilmesi mümkün olabilir:

1. Federe, veri kaydedicinin belli bir fonksiyonunu herhangi bir benzetim zamanında çağırarak istenilen verinin kayıtlara eklenmesini sağlayabilir. Genel uygulama programlarına yönelik olarak yapılan bir çalışmada [Kasbeka et al., 1999], benzer bir yöntem kullanılmıştır. Fakat bu yöntem çok kullanışlı değildir. Çünkü federenin doğru ve kendisinin kayıt için uygun kararlılıkta olduğu zamanlarda bu fonksiyonu uygun parametrelerle çağırması gerekmektedir. Bu durum, federenin geliştiricisinin yazdığı her kod parçasında kayıt sistemini dikkate almasını gerektirecektir.
2. Özel verilerin kaydı, kaydedilecek verilerin hafıza adreslerinin ve tiplerinin kaydedici sisteme verilmesi ve bu sistem tarafından belli aralıklarla bu değerlerin kaydedilmesi ile gerçekleştirilebilir. Böylece federe geliştirici sadece kaydı gerekli olan veriyi bu sisteme tanıtır ve daha sonra kayıt işlemlerini dikkate alması gerekmez. Bu yöntem özel verilerin kaydını yarı otomatik hale gelir ve federe geliştiriciyi kayıt işlemlerinden ciddi oranda soyutlar.
3. Yazılım kodunu okuyarak gerekli yerlere kaydetme ile ilgili kod parçalarını otomatik olarak ekleyen bir araç geliştirilebilir [West & Panesar, 1996], [Feng & Lee, 2006]. Bu yöntemde ilgilenilen verinin değiştirildiği kod parçaları, hem

kayıt hem de deęiřtirme iřleminin yapıldıęı kod paraları ile deęiřtirilir. Örneęin “a” deęiřkenine atama yapılan “a=b” řeklindeki bir kod parası “KAYDET_VE_ATA_a(b)” gibi otomatik üretilen bir fonksiyon ile deęiřtirilerek gerekli kayıt ve atama iřlemi bu fonksiyon ierisinde gereklenir.

Bu yöntemin en göze arpan dezavantajları kodun her deęiřiklikten sonra bu araç ile okutularak yeni kodun oluřturulmasının saęlanması gerektirmesi ve koddaki basit iřlemlerin daha karıřık iřlemlerle deęiřtirilmesi neticesinde performansın etkilenmesidir.

4. Kayıt iin özel veri tipleri kullanılabilir [Ronngren et al., 1996]. Bu yöntemde federe geliřtiricisi kaydedilecek veriler iin standart veri tipleri yerine kaydedici kütüphane tarafından saęlanan aynı özelliklerdeki “kaydedilebilir” veri tiplerini kullanır. Genelde bu veri tipleri řablon (template) tipler olup herhangi bir veri tipi ile oluřturulabilmektedir. Örneęin geliřtirici, kaydetmeyi öngördüęü bir deęiřkeni “int” tipinde tanımlamıřsa artık bunun yerine “Kaydedilebilir<int>” tipini kullanır. Bu yeni veri tipi önceki kullanılan veri tipinin bütün özelliklerini geirgen (ilgili iřlevleri parametre olarak aldıęı veri tipine doğrudan ileterek) bir řekilde saęlar. Kayıt iřlemlerini ise isel bir řekilde gerekleyerek federeyi bu iřlemlerden yalıtır.

Her ne kadar daha geirgen ve az maliyetli bir yöntem olarak gözükse de bu yöntem ile federenin kodu veri kaydetme kütüphanesine doğrudan baęlanmış olur. Ayrıca, kayıt iřlemleri aktif edilsin ya da edilmesin bu tiplerin kullanımı veri dönüşümleri ve çift iřaretleme nedeniyle federenin alıřma performansını olumsuz etkileyebilir. Çift iřaretlemeyi řöyle tanımlayabiliriz: Yukarıdaki durumdaki gibi “int” olan bir veri yapısı yerine “Kaydedilebilir<int>” tipi kullanılmış olsun. Bu durumda bu deęiřkene atama yapılırken eskiden olduęu gibi doğrudan deęer ataması yerine “Kaydedilebilir” sınıfının eřitleme fonksiyonu alıřtırılır. Bu fonksiyonda kayıt iřlemleri iptal edilmiş olsa bile doğrudan atama yapmak yerine bu fonksiyonun aęırılması ile performans düşüřü olur.

4. ÖNERİLEN ÇÖZÜM

Dağıtık benzetim sistemlerinde kayıt ve yeniden oynatma, benzetim öğelerinin yayınladığı verilerin kaydedilmesi ve benzetim ortamına bu verilerin tekrar gönderilerek benzetimin tekrarlanması şeklinde olabileceği gibi, her bir benzetim öğesinin ilgili koşum esnasında aldığı verilerin kaydedilerek daha sonra benzetim öğelerine bu verilerin tekrar sağlanıp benzetim öğelerinin tekrar koşturulması ile de olabilir. İlk durumda benzetim her seferinde aynı şekilde çalışabilmekte ve önceki koşum tekrar gözlenebilmektedir. İkinci durumda ise buna ilave olarak her bir benzetim öğesi, kendine ait parametrelerin değiştirilmesi ile aynı verilerle farklı şekillerde çalıştırılabilir. Böylece, benzetim öğesi önceki aldığı verilerle bağımsız olarak test edilebilir, oluşmuş bir hatanın hangi durumda olduğu bulunarak bu hata ayıklanabilir, benzetim öğesi tekrar tekrar koşturularak analizler yapılabilir. Dolayısıyla ikinci durum, daha geniş kullanım alanı olan bir çözüm sunacaktır.

Verilerin her bir öğeye tekrar sağlanması her bir öğenin kendi kaydedicisi/yeniden oynatıcısı olması ile en etkili ve verimli şekilde olabilir. Başka türlü bir yapı, örneğin merkezi kayıttaki gibi tek bir yerde bu verilerin kaydedilip öğelere tekrar yollanması, öncelikle ölçeklenebilirlik yönünden problemler taşıyacaktır. Ayrıca ağ trafiği oluşturmamak da ancak benzetim öğesine ağ üzerinden erişmeden yerinde kayıt ve yeniden oynatma işlemlerinin gerçekleşmesi ile sağlanabilir. HLA mimarisi göz önüne alındığında merkezi kayıt veya kısmen dağıtık kayıt sistemleri fazladan ağ trafiği oluşturacak ve yoğun çalışma durumunda ağda gecikmelere sebep olarak benzetim sisteminin kayıt sisteminin devrede olması ve olmaması durumlarında farklı çalışmasına sebep olacaktır. Dolayısıyla bu da yanlış bir koşum sonucu ve hatalı bir kayıt verisini sonuç verecektir. Bu sorun çözülse bile ölçeklenebilirlik ciddi bir sorun olarak hâlâ durmaktadır. Benzetim sistemindeki öğeler arttıkça merkezi kaydın performansı düşecek, ağdaki bölümler arttıkça da kısmi dağıtık sistemin bakımı ve ayarlanması daha karmaşık hale gelecektir. Oluşan yük durumu göz önüne alınarak daha fazla kaydedici benzetime eklenerek hangi kaydedicinin hangi verileri kaydedeceğinin belirlenmesi gerekecektir.

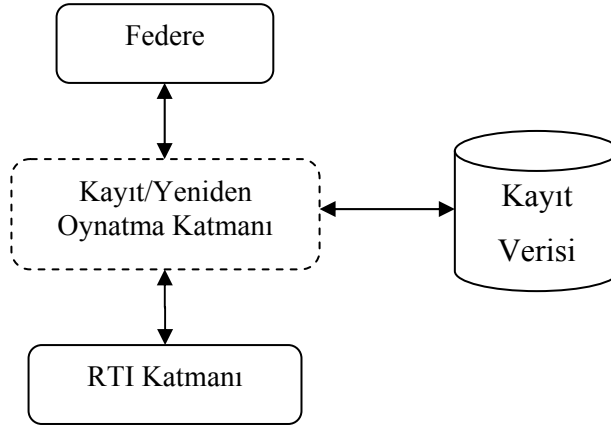
Kayıt sistemi diğer öğelerden fazladan bilgi istemediği, sadece birleşik olduğu benzetim öğesine gelen ve bu öğeden giden verileri benzetim öğesi ile beraber aldığı

için ağda fazladan trafik oluşturmayacaktır. Her bir benzetim ögesinin kendine ait kaydedici/oyuncusının olması kayıt sisteminin performansının sadece benzetim ögesinin ilgilendiği veri miktarıyla orantılı olmasını ve benzetimdeki öge sayısından bağımsız olmasını sağlamaktadır. Dolayısıyla ölçeklenebilir bir çözüm olabilmektedir.

3.1 bölümünde HLA'nın sahip olduğu özelliklerin kayıt sistemini nasıl etkileyeceği ve HLA üzerinde geliştirilecek bir kayıt sisteminin hangi kıstasları taşıması gerektiğinden bahsedilmiştir. Bu kıstaslar göz önüne alındığında tam dağıtık bir kayıt sisteminin en etkili çözüm olacağı ortaya çıkmaktadır. Ayrıca benzetim ögesine özel verilerin kaydı da en verimli bu şekilde yapılabilecektir.

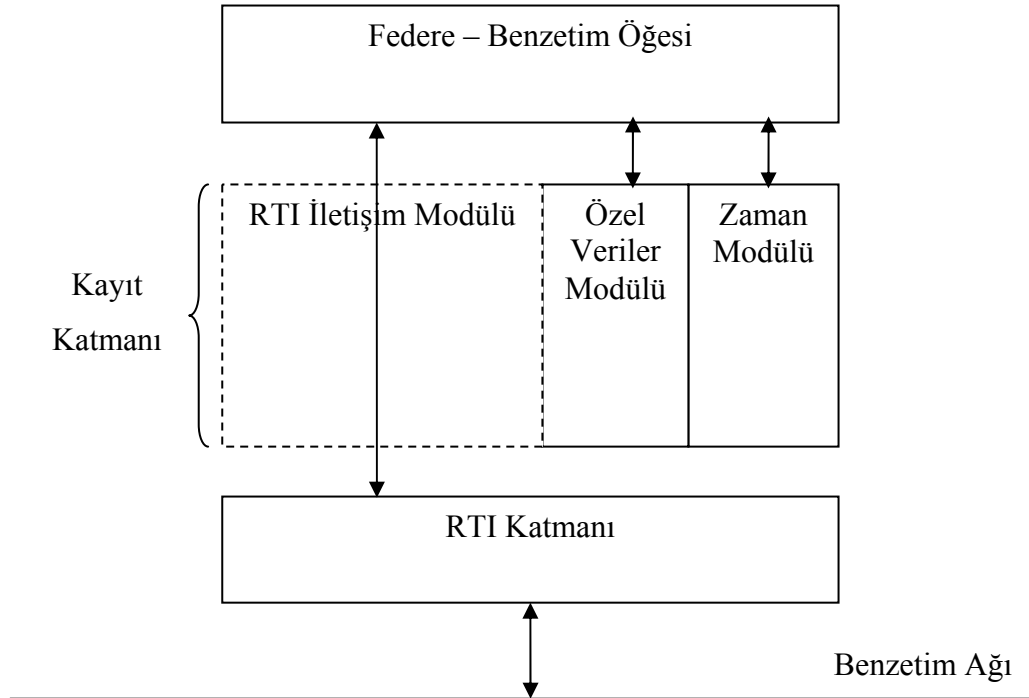
4.1. Genel Mimari

3.2.3 bölümünde tam dağıtık yapı anlatılırken iki şekilde gerçekleştirilebileceğinden bahsedilmişti. Olabildiğince genel ve federeyi en az etkileyecek bir çözüm olması için federe ile federasyon ağı arasında bir katmanda, federenin yazılım arayüzünü değiştirmeden yapılacak bir kayıt sistemi kurulmasının gerektiği ortaya çıkmaktadır. Böylece, federe diğer ögelerle iletişimini eskisi gibi sürdürürken bu ara katman gerekli kayıt ve yeniden oynatma işlemlerini yapabilecektir. Bunun için federenin iletişimini sağlayan katman (RTI katmanı) ile federenin arasında yer alacak ve bu katmanla aynı yazılım arayüzüne sahip olan bir katman geliştirilmelidir (Şekil 4.1). Böylece, federe normal çalışmasında nasıl davranıyorsa aynı şekilde davranacak, kayıt işlemlerinden habersiz olarak çalışacak ama aynı zamanda kayıt ve yeniden oynatma sağlanabilecektir. RTI katmanından gelen ve federe tarafından RTI katmanına yollanacak veriler aradaki bu katman aracılığıyla kaydedilecektir. Yeniden oynatmada ise bu arakatman önceki verileri kayıtlardan okuyarak sanki RTI katmanından geliyormuş gibi uygun zamanlarda federe katmanına iletecektir. Şekil 4.1'te görülen genel mimaride federe, iletişimini kayıt katmanı üzerinden yapmaktadır. Fakat bu katman RTI katmanı ile aynı yazılım arayüzüne sahip olduğu için federe benzetim ağı ile eskiden olduğu gibi haberleşmektedir. Dolayısıyla kayıt katmanı federe için görünmez bir durumdadır.



Şekil 4.1 Kayıt Sistemi Genel Mimari

Şekil 4.2’de kayıt ve yeniden oynatma sisteminin mimarisi daha detaylı bir şekilde görülebilir. Bu kısma kadar anlatılan benzetim ortamına giden ve buradan gelen verileri kaydeden kısım şekilde RTI İletişim Modülü olarak gösterilmiştir. Bu modül, federenin RTI katmanı ile olan arayüzünü aynen taşıyan ve federenin diğer federelerle iletişimini, RTI hizmetlerine ulaşmasını sağlayan fonksiyonları içeren hizmetleri sağlamaktadır. Önceki bölümde bahsedilen eşzamanlılık ve özel verilerin kaydı ihtiyaçlarını karşılamak için bu kayıt katmanına yeni modüllerin de eklenmesi kaçınılmazdır. Özel verilerin kaydını ve eşzamanlılığı sağlayan bu modüller ileriki kısımlarda anlatılmıştır.



Şekil 4.2 Kayıt/Yeniden Oynatma Sistemi Mimarisi Detayı

4.2. Eşzamanlılık

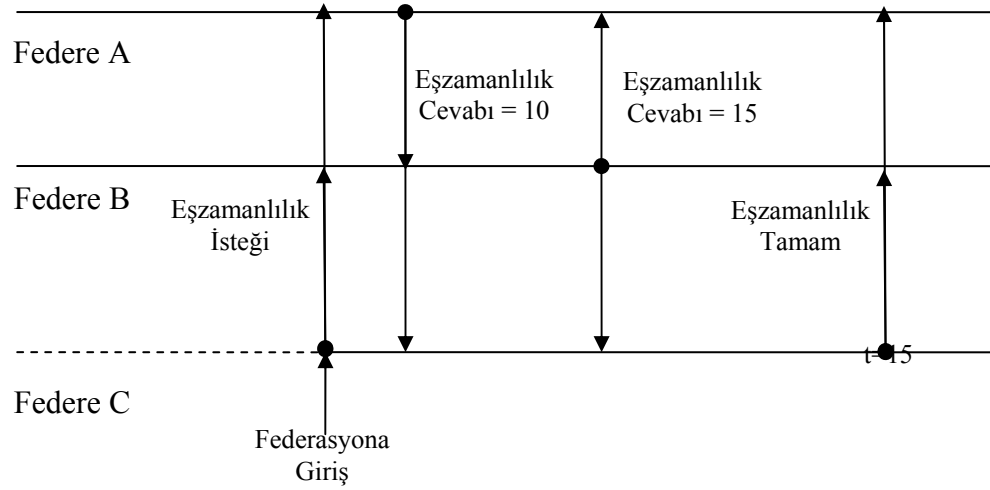
Tekrar oynatma sırasında verilerin doğru zamanda federeye iletilmesi önemlidir. Bunu sağlamak ya RTI'nın sağladığı zaman yönetimi kullanarak ya da kayıt katmanının federenin benzetim zamanını bilerek bu eşzamanlılığı kendi içinde halletmesi ile mümkün olabilir. RTI'nın zaman yönetimi kullanıldığında, zaman bilgisi nesne ve etkileşimlerle beraber bulunduğundan bu bilgiler kullanılarak uygun zamanda veriler federeye sağlanabilir. Zaman yönetimi olmadığında ise federe, kendi zamanını kayıt katmanına ortak kullanılan bir zaman bilgisi modülü ile iletecektir. Böylece kayıt katmanı, verilerle federe zamanını eşleştirerek bir kayıt oluşturabilecektir.

Federenin zaman bilgisini kayıt katmanına iletmesi sadece verilerin doğru zaman bilgileriyle kaydedilmesi için değil aynı zamanda birden çok federenin eşzamanlı olarak tekrar oynatılmak istenmesi, bunların çıktılarının aynı anda görülmek istenmesi gibi durumlarda da gereklidir. Koşum Sonrası İnceleme'nin de bir parçası olarak birden çok federe ya da bütün federasyon normal koşumdan sonra tekrar oynatılmak istenebilir. Bu tekrar oynatma sırasında benzetimdeki olayların aynı zamanlarda gerçekleşmesi federelerin eşzamanlı çalışması ile mümkündür. Eğer bunun için önlem alınmazsa geç başlatılma, ağ gecikmesi, federenin üzerinde çalıştığı makinenin öncekine göre daha yavaş veya hızlı çalışması gibi nedenlerle bu eşzamanlılık bozulabilir. Bunu önlemek için federelerin birbiri ile haberleşmesi ve zaman bilgilerini birbirleri ile paylaşarak ortak bir zamanda buluşmaları gerekir. Zaman yönetimi olduğu durumlarda bu, federelerin RTI'dan belirli bir zamana ilerleme istekleri sırasında gerçekleşir. RTI federelerin zaman bilgilerine ve ilerlemek istedikleri zamana göre onlara hangi zamana ilerleyebileceklerini bir geribildirim fonksiyonu ile iletir. Kayıt sisteminin zaman yönetimine bağlı olmaması gerektiğinden, RTI zaman yönetiminin kullanılmaması durumunda alternatif bir yol gerekmektedir.

Önerilen kayıt sisteminde zaman yönetimi olmadığı durumlarda eşzamanlılık kayıt katmanının RTI üzerinden sadece diğer kayıt katmanlarına ulaşacak etkileşimler göndererek birbiri ile haberleşmesi ile sağlanmaktadır. Bu yapıda federasyona daha sonra katılan federe diğer federelerden “eşzamanlılık noktası/zamanı” ister. Diğer federeler de kendi zaman bilgilerine ve zaman

adımlarına (time step) göre diğer federe ile eşzamanlı olabilecekleri en yakın zamanı yine bir etkileşimle gönderirler. Bu etkileşimleri bütün federeler hem gönderebilir hem de alabilir olduğundan bir federeden yayınlanan “eşzamanlılık noktası” bilgisi bütün federelere gidecektir. Her federe kendi yolladığı “eşzamanlılık noktası”nı ve diğer federelerden gelecek “eşzamanlılık noktası” değerini dikkate alarak en ilerideki zamanı “eşzamanlılık noktası” olarak kabul eder; bu noktaya kadar çalışmalarını devam ettirip bu noktada federasyona sonradan giren federeden eşzamanlılığın sağlandığına dair etkileşimi beklerler. Federasyona sonradan giren federe ise bu “eşzamanlılık noktası”na ulaşınca diğer federelere eşzamanlılığın sağlandığına dair etkileşimi gönderir ve normal çalışmasına devam eder. Bu etkileşimi alan diğer federeler de beklemeyi sona erdirip normal çalışmalarına dönerler.

Örnek bir durum Şekil 4.3’de görülmektedir. Burada federasyonda önce A ve B federeleri bulunmaktadır. Daha sonra C federesi federasyona katılır ve diğer federelere eşzamanlılık noktası isteğini bir etkileşimle gönderir. A federesi kendi zaman durumunu hesaplayarak $t=10$ anını eşzamanlılık olarak hem C’ye hem de B’ye gönderir. Aynı anda B de kendi zamanını dikkate alarak $t=15$ zamanını eşzamanlılık noktası olarak C ve A’ya gönderir. Bu durumda A federesi $t=10$ ’u hesaplamış olmasına rağmen B’den gelen etkileşimi dikkate alarak bu zamanı $t=15$ olarak günceller. B federesi ise A’dan gelen etkileşimdeki zaman kendi hesapladığından küçük olduğu için bunu dikkate almaz. A ve B federeleri bu etkileşimleri gönderdikten sonra $t=15$ anına kadar çalışmalarına devam ederler. $T=15$ ’e geldiklerinde ise C’den gelecek etkileşimi beklemeye başlarlar. C federesi ise A ve B’ye benzer şekilde gelen etkileşimlerdeki verilerin en büyüğünü dikkate alarak $t=15$ anını etkileşim noktası olarak belirler ve bu noktaya kadar çalışır. $t=15$ olduğunda ise A ve B federelerine eşzamanlılığın tamamlandığını belirten bir etkileşimi gönderir ve normal çalışmasına devam eder. A ve B federeleri de bu etkileşimi aldıktan sonra normal çalışmalarına dönerler.

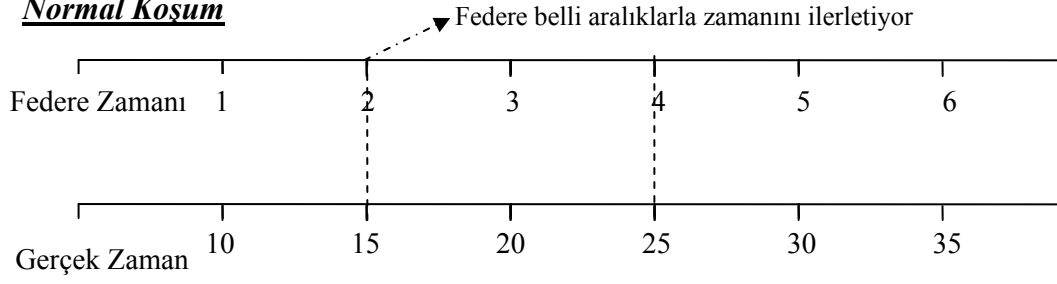
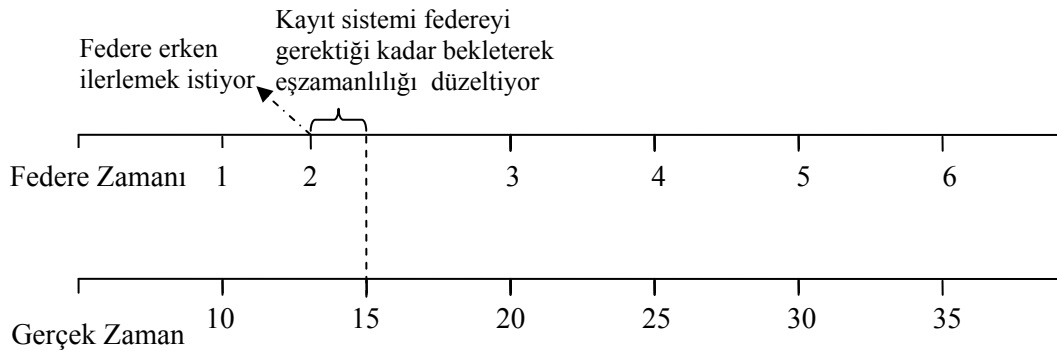
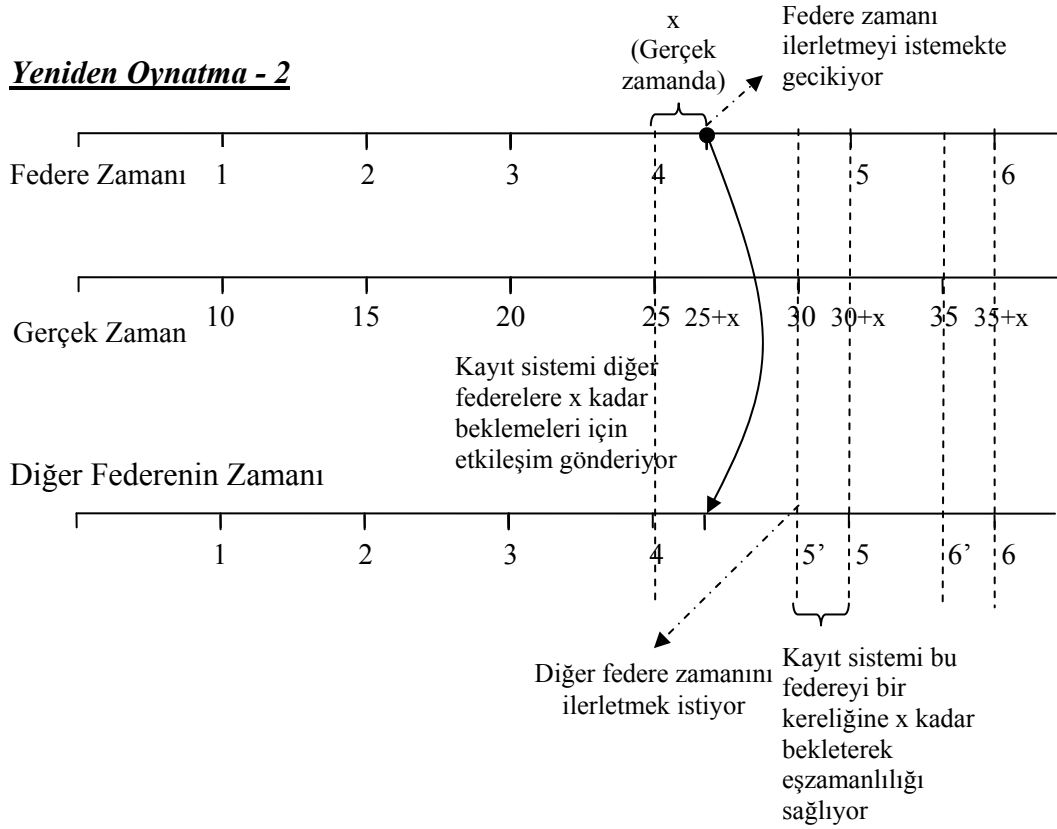


Şekil 4.3 Federasyona girişte eşzamanlılığın etkileşimlerle sağlanması

Eşzamanlılık sadece federasyona giriş sırasında değil, koşum sırasında da bozulabilir. Bunun için özellikle yeniden oynatmada eski koşumdaki zaman durumlarına uyulması gerekir. Bunu sağlamak için zaman modülü, federenin normal koşumu sırasında Federe Zamanı-Gerçek Zaman (WCT/Wall Clock Time) ilişkilerini de kaydeder. Tekrar oynatma sırasında, normal koşum sırasında kaydedilmiş olan Benzetim Zamanı-Gerçek Zaman ilişkileri göz önüne alınarak, federenin belli bir benzetim zamanına ilerleme isteğinin erken olması durumunda federeyi bekletmek ve federenin bir benzetim zamanına ilerleme isteğinin geç gelmesi durumunda ise diğer federelere beklemelerini belirten etkileşimi göndermek suretiyle koşum sırasındaki eşzamanlılık sağlanır. Şekil 4.4’de örnek iki durum görülmektedir Federe normal koşumunda benzetim zamanını birer birim arttırarak ilerlemektedir. Bu benzetim zamanları belirli gerçek zaman değerlerine denk düşmekte ve kayıt katmanı bu benzetim zamanı-gerçek zaman eşleşmelerini kaydetmektedir. Yeniden oynatma sırasında ilk durumda görüldüğü gibi federe benzetim zamanını erken ilerletmek isterse, yani benzetimi normal koşumdan hızlı koşturmak isterse kayıt katmanı önceki kayıt bilgilerini kullanarak federenin bu isteğinin gerçek zamanda ne kadar önce olduğunu hesaplayarak federeyi bu fark kadar bekletmektedir. Böylece benzetimin normal koşumdaki hızı ile aynı sürede ilerlemesini sağlamaktadır.

İkinci durumda ise federenin normal koşumdakinden geç ilerlemesi durumu incelenmiştir. Bu durumda kayıt katmanı federeyi hızlandıramayacağı için diğer federelerin beklemesini sağlayarak eşzamanlılığı sağlamaktadır. Federe normal koşumdaki hızıyla ilerlerken belli bir anda gerçek zamanda x birim kadar gecikmiş

ve kayıt katmanına zaman ilerleme isteğini x kadar geç bildirmiştir. Bu durumda kayıt katmanı RTI üzerinden diğer federelerin kayıt katmanlarına x kadar beklemeleri gerektiğini bildiren birer etkileşim göndermektedir. Diğer federelerin kayıt katmanları ise bu bilgiyi alarak kendi federelerini bir sonraki zaman ilerletme isteklerinde gerçek zamanda x birim kadar bekleterek bütün federelerin aynı zamana sahip olmalarını sağlamaktadır.

Normal Koşum**Yeniden Oynatma - 1****Yeniden Oynatma - 2**

Şekil 4.4 Koşum sırasında eşzamanlılığın sağlanması

4.3. Özel Veriler

Federeye özel verilerin kaydedilmesi için 3.3 bölümünde bahsedilen çözümlerden ikincisi bu yapıya ve amaçlarına daha uygundur. Şöyle ki, önerilen mimaride federenin kayıt işlemlerinin aktif olmasından olabildiğince az etkilenmesi ve federe kodunun en az değişikliğe uğraması hedeflenmiştir. Genel mimaride olduğu gibi özel verilerin kaydında da federe geliştiricisine mümkün olan en az sorumluluk verilmelidir. Bunun için federenin “özel veri modülü” ile kaydedilmesini istediği değişkenleri kayıt sistemine bildirmesi ve kayıt işlemlerinin kayıt sistemi tarafından otomatik olarak yapılması düşünülmüştür. Normalde RTI arayüzünde özel verilerin kaydı için bir öge bulunmadığından bir modül aracılığıyla federenin bu işlemleri yapması için arayüz sağlanmıştır.

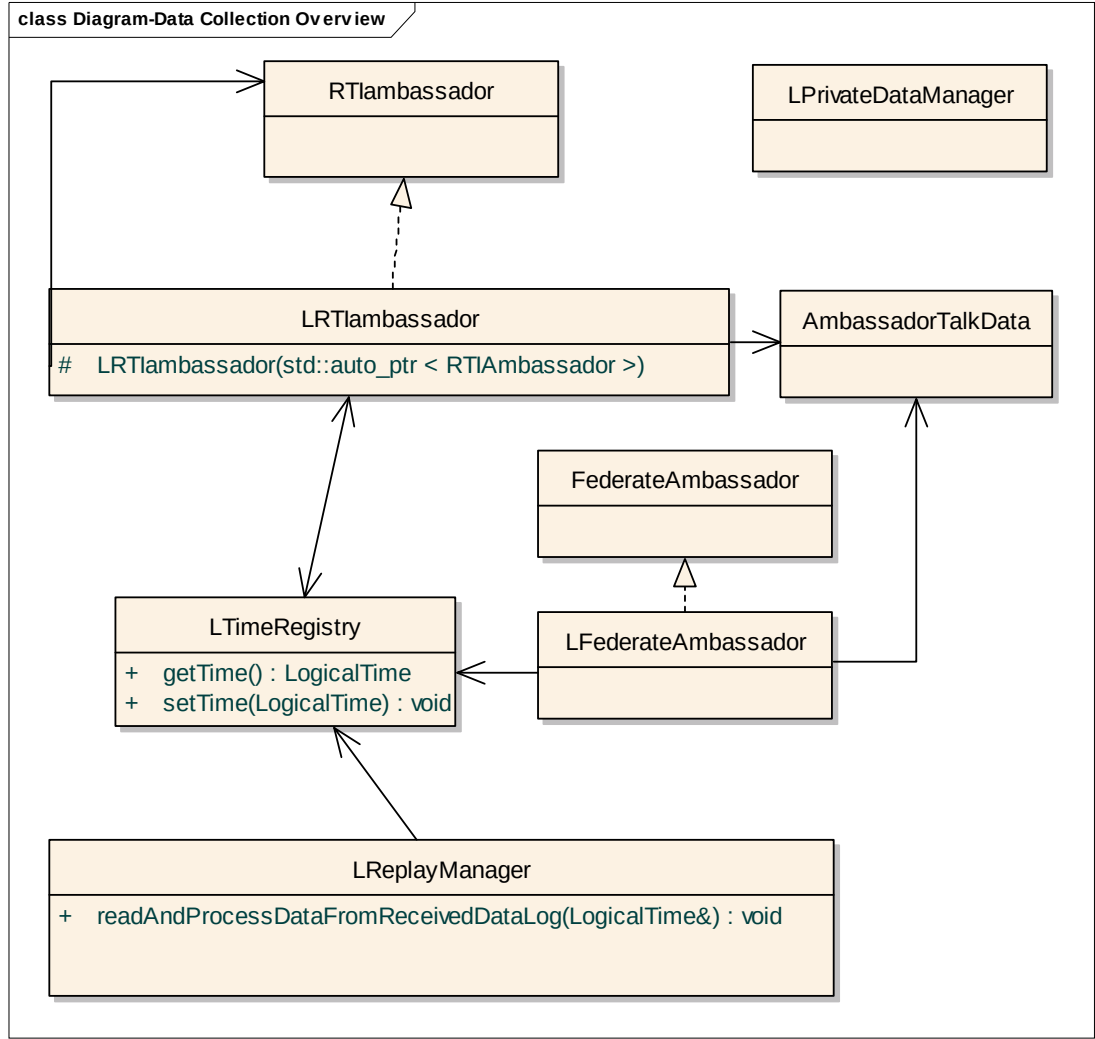
Önceki bölümlerde bahsedildiği gibi bu özel veriler yeniden oynatmadan çok federenin eski bir benzetim zamanına döndürülmesi sırasında gereklidir. Çalışması genel olarak kayıt için belirlenen değişkenler kayıt sistemi tarafından belli aralıklarla kaydedilmesiyle eski zamana geri dönüş sırasında ise o zamanki değerlerine döndürülmesi şeklindedir. Bunun gerçekleştirilmesi ile detayları gerçekleştirme bölümünde anlatılmıştır.

5. GERÇEKLEME

HLA'nın 2.1.1 bölümünde RTIAmbassador ve FederateAmbassador sınıflarının yazılım arayüzünün esasını oluşturduğu anlatılmıştı. Bu durumda, hem RTI katmanının arayüzünü hem de kayıt ve yeniden oynatma işlerini sağlayan bir katman yapılması için bu iki sınıfın türetilmesi, miras edinilmesi gerekir. FederateAmbassador zaten federenin kendisinden türeterek bir sınıf yazması ve RTI'dan gelen geribildirimleri alabilmesi için RTI kütüphaneleri tarafından türetilmeye açık yazılmıştır. Fakat RTIAmbassador, RTI katmanında genel olarak Fabrika Şablonu (Factory Pattern) ile örneklenebilen, kendisinden örnek (instance) ancak bir fabrika sınıfının fonksiyonu ile yapılabilen bir sınıf olarak tasarlanmıştır. Bu durumda RTIAmbassador'u türetmenin en sağlıklı yolu Bileşik Şablon (Composite Pattern) kullanmaktır. RTIAmbassador'dan türetilcek sınıfa daha önceden oluşturulmuş RTIAmbassador örneği verilir ve bu sınıf hem federeye RTI arayüzünü hem de kayıt katmanındaki diğer sınıflara kayıt ile ilgili diğer arayüzleri sağlar. RTI ile ilgili hizmetleri parametre olarak aldığı RTIAmbassador örneğine ileterek gerçekler. Dolayısıyla federe, RTI hizmetlerini yazılım arayüzünde herhangi bir farklılık olmadan bu iki sınıftan türemiş olan LRTIAmbassador ve LFederateAmbassador sınıfları aracılığı ile alabilir. Bu hizmetler sağlanırken de kayıt sistemi gerekli kayıt ve yeniden oynatma işlevlerini gerçekleştirir.

Kayıt sisteminin devreden çıkarılması, yeniden oynatma sırasında zaman yönetiminin kullanılıp kullanılmaması gibi kayıt sistemi ile ilgili ayarlar ise bir yapılandırma dosyasının değiştirilmesi ile sağlanır. Böylece federe kodunda değişiklik yapılmasına gerek kalmaz.

Kayıt ve yeniden oynatma aynı katmanda yapıldığından her iki sınıfın da (LRTIAmbassador ve LFederateAmbassador) yayınlanan/abone olunan nesne/etkileşimleri, federenin mevcut zamanını, alınan/yollanan verileri bilmesi gerekir. Bunun gerçekleşmesi için bu iki sınıfın ortak kullandığı bir sınıf, AmbassadorTalkData, gerçekleştirilmiştir. Böylece tekrar tekrar aynı verilerin RTI'dan istenmesi engellenmiş ve bir performans artışı sağlanmıştır. Kayıt sisteminin genel sınıf yapısı Şekil 5.1'de görülmektedir.



Şekil 5.1 Kayıt/Yeniden Oynatma Sistemi sınıf yapısı

Eşzamanlılığın “zaman modülü” kullanılarak sağlandığından bahsedilmiştir. Bu modülün dışa açık arayüzü olan sınıf LTimeRegistry olarak gerçekleştirilmiştir. Bu sınıfın amacı zaman yönetimi olmadığı durumlarda federenin zamanının hem diğer federelerle hem de kendinin önceki normal koşum sırasında kaydettiği zaman değerleri ile uyumlu olarak ilerlemesini sağlamaktır. Federe, belli bir zamana ilerleme isteğini bu sınıfın *setTime* fonksiyonunu kullanarak gerçekleştirir. LTimeRegistry, gerekli durumlarda federeyi bu fonksiyon içerisinde bekletir. Bu sınıf gerekli etkileşimleri göndermek için LRTIambassador’u kullanır. Dışarıdan gelen etkileşimler ise LFederateAmbassador tarafından bu sınıfa aktarılır. Tabii ki dışarıdan gelen ve dışarıya giden bu etkileşimler alt taraftaki federe katmanına iletilmez, böylece federe eşzamanlılık ile ilgili işlemlerden yalıtılmış olur. LTimeRegistry, hem federe hem de kayıt katmanı tarafından kullanılacağından tek

örnekli (singleton) olarak gerçekleştirilmiş ve setTime ve getTime fonksiyonları iki ayrı işlemciğin aynı anda ulaşmasına karşı korumaya alınmıştır.

Özel verilerin kaydı nispeten daha karmaşıktır. Çünkü özel verilerin tipleri genelde kullanıcının tanımladığı o federeye özel veri tipleri olabilir. Federasyondaki veri tiplerinin içeriği FOM'da belirtildiği için federasyonda kullanılan veri tipleri kayıt sistemi tarafından en alt tiplere kadar indirgenerek kaydedilebilmektedir. Federeye özel veriler için ise bu işlemin federe geliştiricisi tarafından yapılması gerekebilir. Aslında bu konu kullanılan programlama diline göre değişebilir. Java ve C# gibi üst düzey diller karma veri tipleri yerine sınıfları kullanmayı zorlamakta, geliştirilen her sınıfın ise otomatik olarak temel bir sınıftan türetmekte (Object sınıfı) ve bu temel sınıfın da otomatik olarak kayıt özelliğini sağlamaktadır. Böylece herhangi bir sınıf doğrudan kaydedilebilir durumda olur ve federe geliştiricisinin ayrıca kayıt için koduna eklemeler yapmasına gerek kalmaz. Bu tabii ki Java ve C#'in çalışma sırasında yorumlanan (interpreted) birer dil olması ile sağlanmaktadır. Fakat genelde benzetim sistemlerinde performans daha öncelikli olarak ele alındığından dil olarak C++ gibi doğrudan makine koduna çevrilen diller tercih edilebilmektedir. Bu dilde hazır olarak kayıt mekanizması yoktur; fakat bu işlemleri sağlamak için birçok kütüphane yazılmıştır. Boost ve sln bunlardan en önemlileridir. Bu kütüphaneler genel olarak temel veri tiplerini (int, float, vector) kaydetme yeteneğine sahip olup kullanıcının tanımladığı özel bir sınıfın veya karma veri yapısının (struct) kaydedilmesini bu sınıflar için kullanıcı tarafından yazılacak bir kayıt fonksiyonu ile sağlarlar. Bu fonksiyonlarda sınıflar içerdikleri daha basit veri yapılarını kütüphanenin sağladığı yazılım arayüzüyle kaydeder/geri yükler.

Bu çalışmada yapılan sistem de C++ üzerinde geliştirildiğinden, bahsi geçen kütüphaneler gibi kayıt katmanı da kullanıcının ya basit veri tiplerini kaydetmesini ya da kendi sınıflarına özel kayıt fonksiyonunu gerçekleştirmesini gerektirmektedir. LPrivateManager sınıfının sağladığı şablon (template) fonksiyon ile verilerin kaydedilmesi için bu sınıfa haber verir. LPrivateManager ise LTimeRegistry tarafından belli aralıklarla (bu aralıklar yapılandırma dosyasında değiştirilebilir) çağırılarak kaydedilmesi istenen bu veriler zaman bilgileri ile beraber kaydedilir. Federe geçmiş bir zamana dönmek istediğinde bunu LTimeRegistry'e normal

koşumda zamanını iletlediği gibi iletir. Bu durumda LTimeRegistry, LPrivateManager'a geçmiş bir zamana dönme isteğini iletir. LPrivateManager ise kayıt için belirlenmiş değişkenlerin o zamandaki değerlerini kayıt verilerinden okuyarak değişkenlere bu değerleri atar. Böylece federe bu değişkenlerin eski değerlerini otomatik olarak elde eder.

6. DENEYSEL ÇALIŞMALAR

Deneysel çalışmalar iki kısımdan oluşmaktadır:

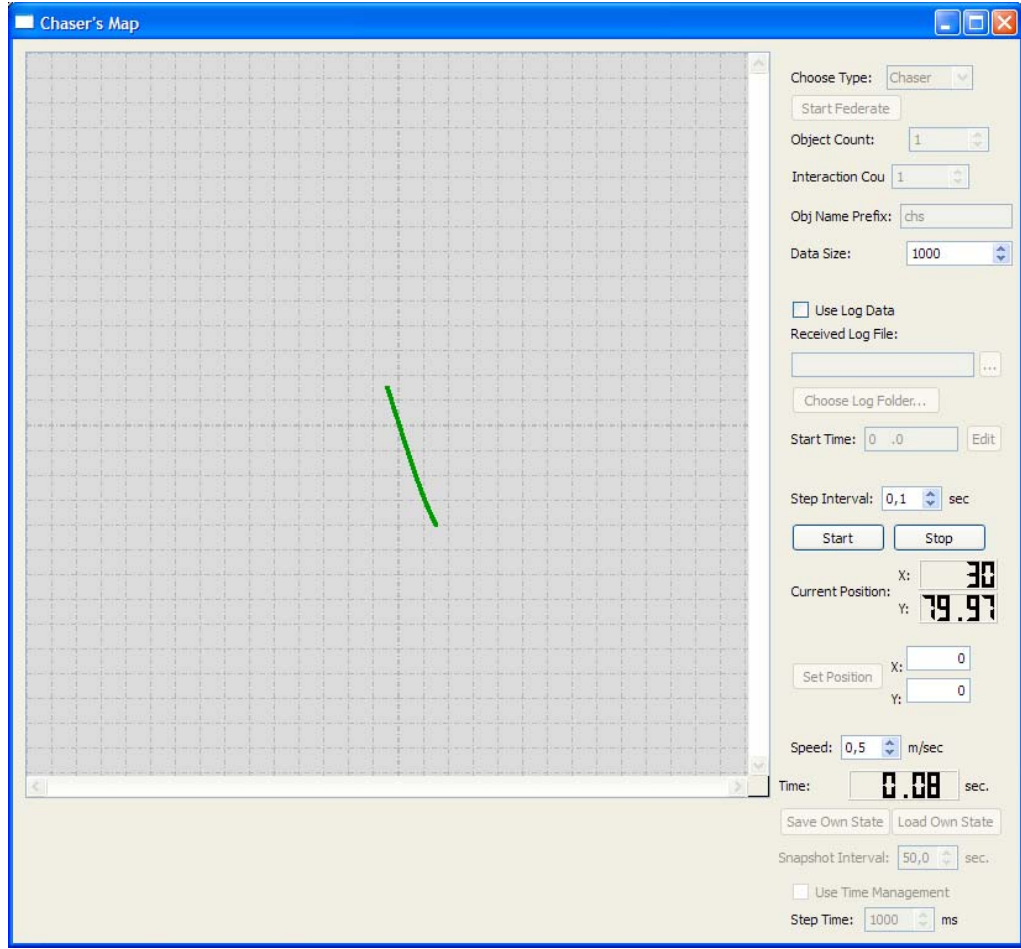
1. Performans Deneyleri
2. Doğruluk Deneyleri

Performans deneyleri veri toplama ve yeniden oynatmanın etkinliğini ve getirdiği yükü ölçmeyi hedeflerken, doğruluk deneyleri kaydedilen verilerle tekrar oynatmanın ve benzetim geçmişine dönmenin benzetimin çıktılarını etkilemediğinin gösterilmesini amaçlamaktadır.

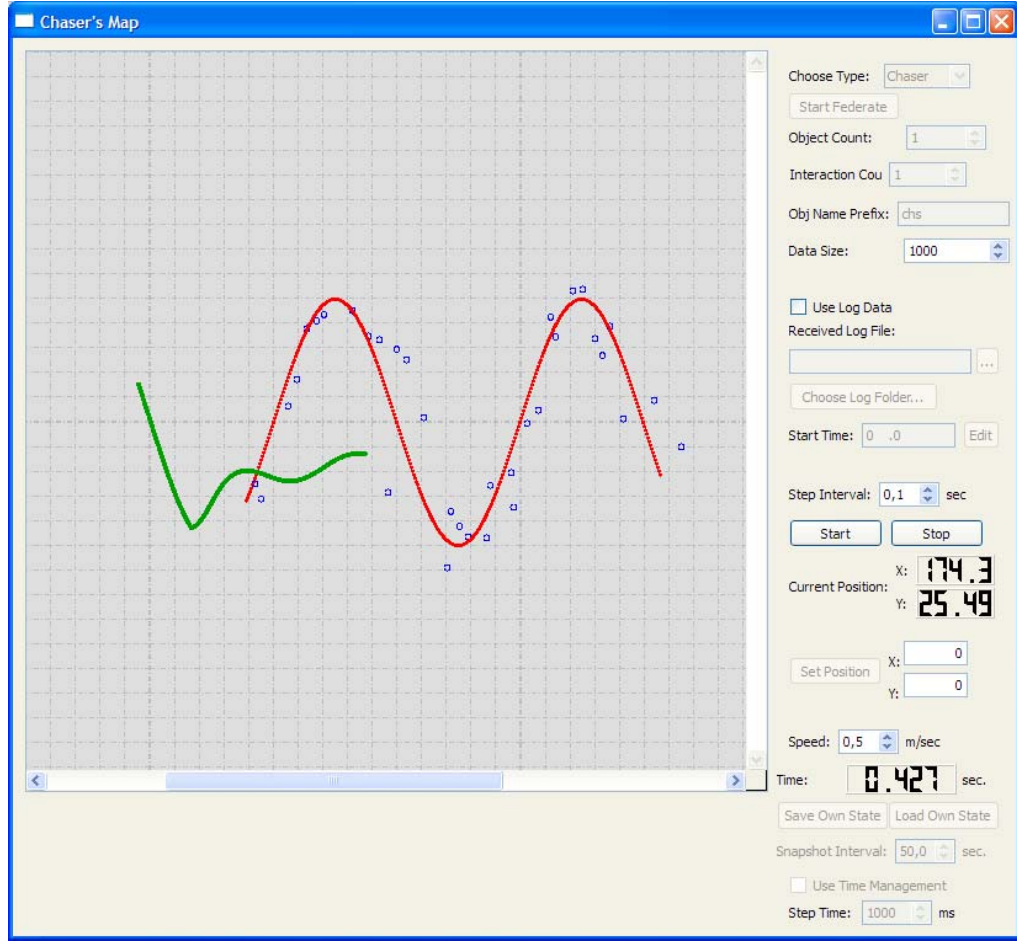
Deneysel çalışmalar bir yerel ağ üzerinde çalışan standart donanıma sahip (Intel(R) Core(TM)2 Duo 2.20 GHz, 2 GB DDR2 RAM) birkaç bilgisayar üzerinde yapılmıştır. Veri kaydediciler federelerle birleşik olduğundan oluşturdukları fazla yükün kullanılan ağdan bağımsız olduğu açıktır. Yani LAN veya WAN kullanılmasının deney sonuçları üzerinde etkisi olmamaktadır. Bu yüzden WAN üzerinde deney yapmaya gerek görülmemektedir. Deneylerde alınan verileri kaydedilmesi ve yeniden oynatma sırasında okunması ile federeye iletilmesi işlemlerinin oluşturduğu yük ve bu işlemlerin doğruluğu ölçülmeye çalışılmıştır.

Deneylerde kullanılacak federelerin olabildiğince basit olması sağlanmaya çalışılmıştır. Bu sayede federelerin kendilerinden kaynaklı etkilerin deney sonuçlarını etkilememesi sağlanacaktır. Deneylerde iki tip federe kullanılmıştır. Bir tanesi “kovalayan” federe, diğeri ise “kovalanan” federe. Her iki federe de konum bilgisi içeren birer nesne oluşturmakta ve bunların bilgilerini zamanla güncellemektedir. Kovalayan federeler kovalanan nesnelerin pozisyon bilgilerini alarak kendi nesnelerini bu nesnelerden en yakınına yaklaştıracak yönde ilerleteceklerdir. Etkileşimleri de deneylere dâhil etmek için federelerin birbirine ateş etmesini benzeten etkileşimler göndermesi sağlanmıştır. Bu etkileşimler de hangi konuma ateş edildiği bilgisini içerecektir. Ayrıca iletilen ve kaydedilen verinin boyutunun değişmesi ile etkisinin gözlemlenebilmesi için bu nesne ve etkileşimlerin içerisine değişken boyutlu diziler konulmuş ve bunların boyutu değiştirilerek sonuçlar gözlemlenmiştir. Şekil 6.1 ve Şekil 6.2’te “kovalayan” federeye ait ekran görüntüleri görülmektedir. “Kovalayan” federe, diğer federeden bilgi alana kadar

kendi sinüs eğrisi üzerinde hareket etmekte, diğer federeden bilgiler gelmesiyle bu gelen bilgilerdeki nesne pozisyonuna göre konumunu belirlemektedir.



Şekil 6.1 "Kovalayan" federe kendi hareket denklemine göre ilerliyor



Şekil 6.2 "Kovalayan" federe diğer federenin pozisyon bilgilerine göre ilerliyor

6.1. Performans Deneyleri

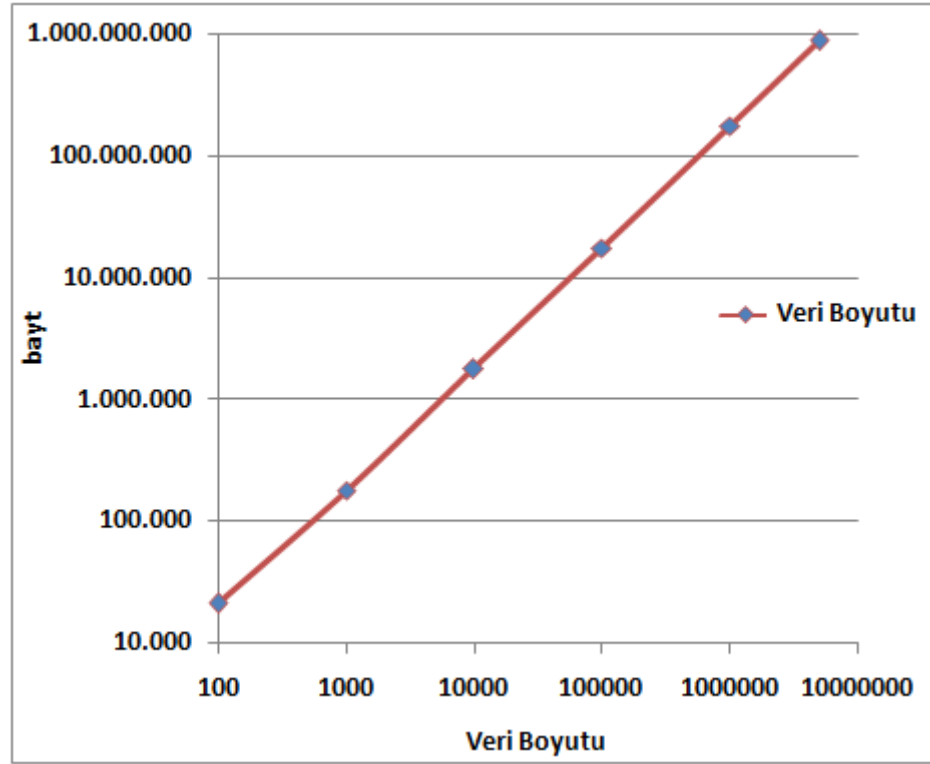
İki türlü deney gerçekleştirilmiştir:

- Nesne ve etkileşimlerin içerdikleri veri boyutu değiştirilerek bunun oluşan kayıt dosyası boyutuna ve kayıt süresine etkisi ölçülmüştür.
 - Nesne ve etkileşimlerin boyutu sabit tutularak nesne ve etkileşim sayısının artışının oluşan dosya boyutuna ve kayıt süresine etkisi ölçülmüştür.
- Kayıt sistemi tarafından toplanan veri bir yere kaydedilmek zorundadır. Bu çalışmada önerilen çözümde dosyaya kaydetmek tercih edilmiştir. Fakat etkin bir sistem için bu dosya boyutunun artan nesne/etkileşim sayısı ile sabit bir oranda arttığından emin olunması gerekir. Bu deneyde seçilen bir nesne ve etkileşime ait bir değişkenin (attribute/parameter) boyutu değiştirilerek nesne/etkileşimin

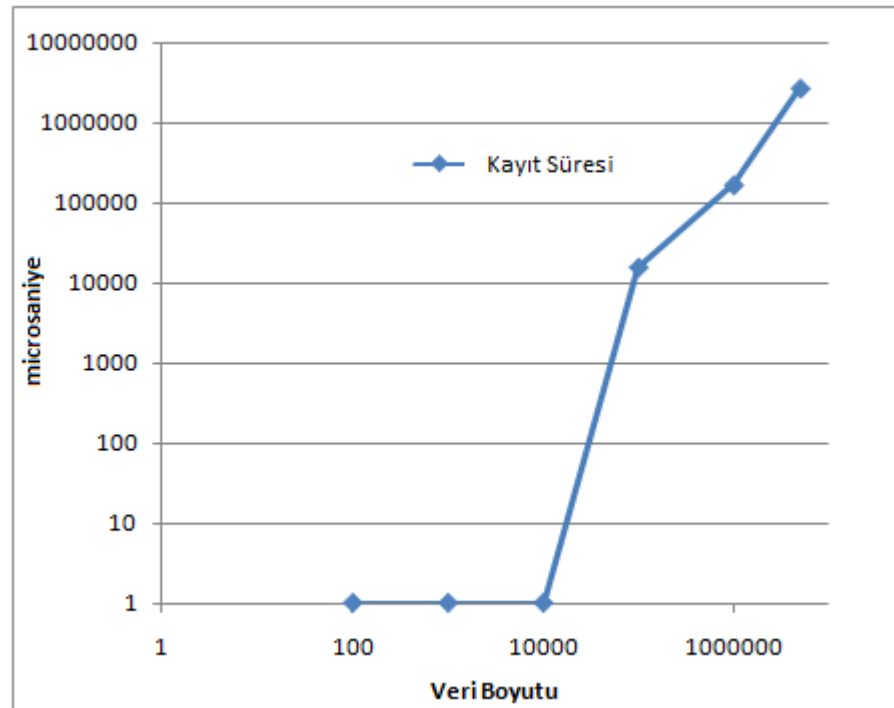
boyutunun deęiřmesi saęlanmıřtır. Bu deęiřken bir diziden oluřmaktadır. Deneyde bu dizinin boyutu arttırılarak ierisine rastgele sayılar atanmıřtır. Kayıt sırasında geen sreyi hesaplamak iin kayıt ncesinde ve sonrasında sistem zamanı bir yazılım ktphane ile alınarak aradaki fark llmřtr.

Deney sonuları bize kaydedilen verinin nesne/etkileřim boyutuna gre $O(n)$ olduęunu gstermektedir (řekil 6.3). lm sırasında federe belli bir zaman alıřtırılmıř ve gnderdięi tek bir nesne kaydedilerek oluřan dosya boyutu incelenmiřtir. Farklı kořumlarda farklı nesne boyutları ile deney tekrarlanarak řekil 6.3’deki sonu elde edilmiřtir.

Boyut deęiřiminin kayıt sresine olan etkisi ise kabul edilebilir seviyede grlmektedir řekil 6.4’de grldę gibi tek bir nesnenin kaydı sırasında nesnenin boyutu. 100 bin byte’aulařana kadar kayıt iřleminin kayda deęer bir etkisi grlmemiř. Kayıt sresini lmek iin kullanılan ktphane bu seviyeye kadar kayıt sırasında sistem zamanında bir deęiřim lmemiřtir. Dolayısıyla 100 bin bayt’a kadar olan kısımda kayıt sresi sıfır olarak llmřtr. 100 bin byte seviyesinde yaklařık 15 milisaniye tutan bir kayıt sresi oluřmuřtur. Aslında bu veri boyutu RTI zerinden ok sık gnderilebilecek bir byklkte deęildir. Dolayısıyla standart bir bilgisayar zerinde federeye ait dięer hesaplamaların da yapıldıęı bir ortamda bu sonu makul gzkmektedir.



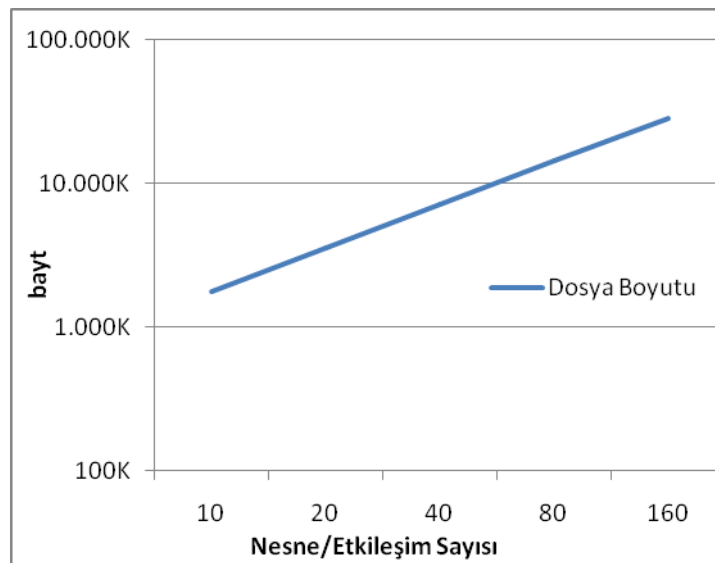
Şekil 6.3 Veri Boyutu - Dosya Boyutu Değişimi



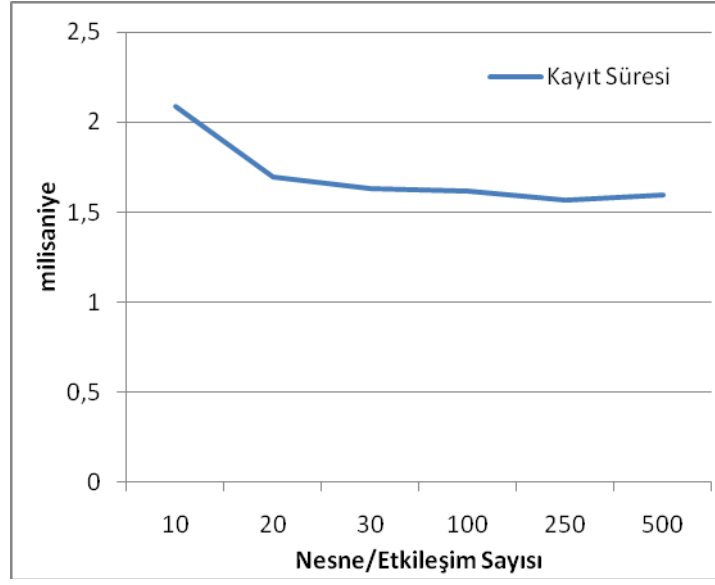
Şekil 6.4 Veri Boyutu - Kayıt Süresi Değişimi

- b. Benzetim ortamında iletilen veri miktarı bir önceki deneydeki duruma benzer şekilde tek bir nesnenin boyutunun değişmesi ile olabileceği gibi yayınlanan nesne/etkileşimlerin sayısının değişimi ile de olabilir. Bunun için bu deneyde nesne ve etkileşimlerin boyutları sabit tutularak bunların sayılarının artışının kayıt performansına ve dosya boyutuna etkisi incelenmiştir.

Deney sonuçlarına göre dosya boyutu nesne/etkileşim sayısına göre bir önceki deneyde olduğu gibi $O(n)$ olarak artmaktadır (Şekil 6.5). Nesne/etkileşim sayısının artmasının her bir nesne/etkileşimin ortalama kayıt süresine etkisi ise yaklaşık olarak sabittir (Şekil 6.6). Yani bir nesne/etkileşimi kaydetmek yaklaşık 1,5 milisaniye almakta, nesne/etkileşim sayısının artması ile toplam kayıt süresi sabit oranda artmaktadır. Şekilde nesne başına düşen kayıt süresinin önce azaldığı, sonra ise yaklaşık sabit kaldığı, en sonunda ise biraz arttığı görülmektedir. Nesne başına düşen kayıt süresi toplam kayıt süresinin nesne sayısına bölünmesi ile elde edilmiştir. Bu sebeple ilk durumda nesne sayısının çok az olması ile dosyayı açma kapama işlemlerinin toplam kayıt süresindeki etkisi daha çok olmuş ve nesne başına ortalama normalden çok ölçülmüştür. Son değer hafif artışı ise veri boyutunun çok artması ile bilgisayar donanımının dosya yazma hızına yetişmekte zorlanması sebebiyle olmuştur. Bu durumda nesne/etkileşim sayısının artışının performansa etkisi de $O(n)$ olarak ifade edilebilir. Bu durum, kayıt sisteminin nesne/etkileşim sayısına göre ölçeklenebilir olduğunu göstermektedir.



Şekil 6.5 Nesne/Etkileşim Sayısı -Dosya Boyutu Değişimi



Şekil 6.6 Nesne/Etkileşim Sayısı-Kayıt Süresi Değişimi

6.2. Doğruluk Deneyleri

Bu deneylerde temel amaç federenin normal koşum sırasında ürettiği sonuçların tekrar oynatma sırasında ürettiği sonuçlarla aynı olup olmadığının belirlenmesidir. Normal koşum esnasında diğer federelere ait veriler RTI üzerinden gelirken tekrar oynatma sırasında bu veriler kayıt verilerinden okunarak federeye sağlanmaktadır. Bu deneyi gerçeklemek için diğer federenin yayınladığı nesnenin pozisyon bilgisine göre hareket eden “kovalayan” federenin kendi nesnesine ait yayınladığı veriler zaman bilgileri ile beraber kaydedilmiş ve iki durumdaki sonuçlar karşılaştırılmıştır. Bu “kovalayan” federe kendi pozisyonunu diğer federeye göre ayarladığı için aynı pozisyon değerlerini üretmesi, aynı verileri aynı zamanlarda aldığını ve kayıt sisteminin doğru çalıştığını gösterecektir.

Deneyde 1 birim zaman aralıkları ile ilerleyen iki federe (kovalayan ve kovalanan) kullanılmıştır. İlk önce “kovalanan” federe çalıştırılmış; bu federe 26 birim zaman ilerleyince diğer “kovalayan” federe federasyona dâhil edilmiştir. Bundan sonra eşzamanlılık kısmında anlattığımız gibi “kovalanan” federe “kovalayan” federenin 26. zamana gelmesini beklemiş, “kovalayan” federe 26. zamana ulaşınca diğerine haber vererek normal koşumuna dönmesini sağlamış, kendisi de normal koşuma devam etmiştir. Kovalanan federe bundan sonra kendi nesnesine ait bilgiyi kovalayana göndermiş, o da 27. zamanda bu bilgiyi dikkate alarak ilerleyişini değiştirmiştir.

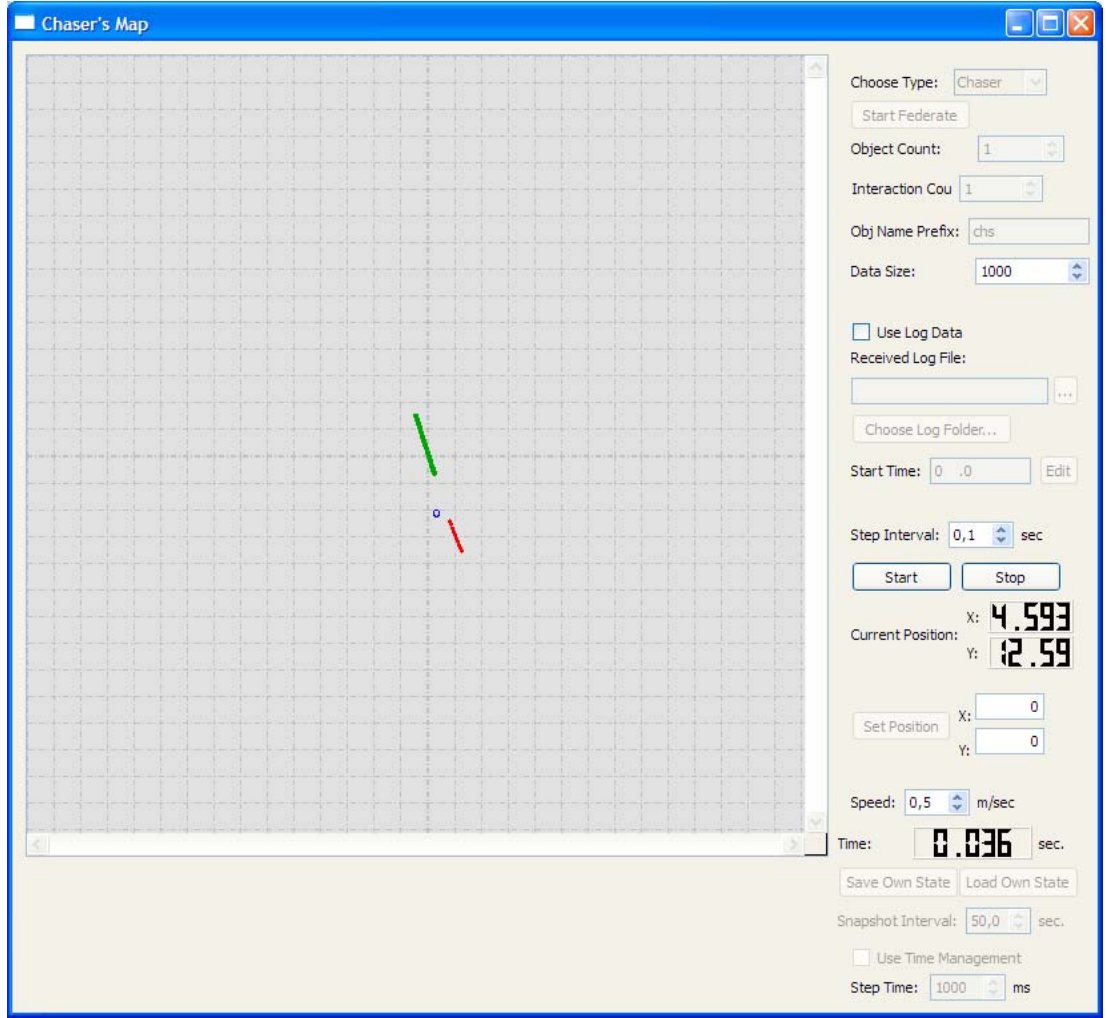
Bu koşum iki federenin önce normal olarak ve daha sonra da kovalayan federenin kayıt verilerini kullanarak çalıştırılması ile tekrarlanmıştır. Normal koşumda kovalayan federenin kayıt katmanı, diğer federeden gelen verileri ve zaman bilgilerini kaydederek kayıt verisini oluşturmuştur. Daha sonra kayıt katmanının yapılandırma dosyasında federenin RTI’den gelen verilerle değil kayıt verileri ile beslenmesini belirten ayarlama yapılmış ve federe tekrar çalıştırılmıştır. Bu tekrar oynatma sırasında kovalayan federe normal koşumdaki ile aynı şekilde çalışmış fakat bu sefer aldığı veriler kayıt katmanı tarafından sağlanmıştır. Federe verileri doğru zamanlarda aynı değerlerde aldığı için sonuç normal koşum ile aynı olmuştur. **Hata! Başvuru kaynağı bulunamadı.**’de koşum ve yeniden oynatma sırasında kovalayan federenin kendisine ait nesne pozisyonları ile kovalanan nesneye (hedef) ait pozisyonların zamana bağlı değişimleri gösterilmiştir. Verilerin çokluğu nedeniyle

tabloda sadece bir kısmı sergilenmiştir. 27. zamana kadar kovalanandan veri gelmediği için bu kısımlarda hedef pozisyonu yoktur. Şekil 6.7’de de bu verilerin toplandığı koşuma ait ekran görüntüsü görülmektedir.

Bu deney sonucunda kayıt sisteminin verileri doğru olarak kaydedip doğru zamanlarda geri besleyebildiği görülmüştür.

Tablo 6.1 Normal ve Kayıt Verileri ile Koşum Karşılaştırması

<i>Federe Zamanı</i>	<i>Normal Koşum</i>				<i>Kayıt Verisi ile Koşum</i>			
	Kendi X	Kendi Y	Hedef X	Hedef Y	Kendi X	Kendi Y	Hedef X	Hedef Y
16	-2,0000	-7,8459			-2,0000	-7,8459		
17	-1,5000	-6,2791			-1,5000	-6,2791		
18	-1,0000	-4,7107			-1,0000	-4,7107		
19	-0,5000	-3,1411			-0,5000	-3,1411		
20	0,0000	-1,5707			0,0000	-1,5707		
21	0,5000	0,0000			0,5000	0,0000		
22	1,0000	1,5707			1,0000	1,5707		
23	1,5000	3,1411			1,5000	3,1411		
24	2,0000	4,7107			2,0000	4,7107		
25	2,5000	6,2791			2,5000	6,2791		
26	3,0000	7,8459			3,0000	7,8459		
27	3,1534	8,3218	16	48,1754	3,1534	8,3218	16	48,1754
28	3,3080	8,7973	17	50,9041	3,3080	8,7973	17	50,9041
29	3,4639	9,2724	18	53,5827	3,4639	9,2724	18	53,5827
30	3,6210	9,7471	19	56,2083	3,6210	9,7471	19	56,2083
31	3,7794	10,2213	20	58,7785	3,7794	10,2213	20	58,7785
32	3,9392	10,6951	21	61,2907	3,9392	10,6951	21	61,2907
33	4,1003	11,1684	22	63,7424	4,1003	11,1684	22	63,7424
34	4,2629	11,6412	23	66,1312	4,2629	11,6412	23	66,1312
35	4,4270	12,1135	24	68,4547	4,4270	12,1135	24	68,4547
36	4,5926	12,5853	25	70,7107	4,5926	12,5853	25	70,7107
37	4,7599	13,0565	26	72,8969	4,7599	13,0565	26	72,8969
38	4,9288	13,5271	27	75,0111	4,9288	13,5271	27	75,0111
39	5,0995	13,9971	28	77,0513	5,0995	13,9971	28	77,0513
40	5,2720	14,4664	29	79,0155	5,2720	14,4664	29	79,0155
41	5,4464	14,9350	30	80,9017	5,4464	14,9350	30	80,9017
42	5,6228	15,4028	31	82,7081	5,6228	15,4028	31	82,7081



Şekil 6.7 Karşılaştırma durumunun ekran görüntüsü

Özel verilerin kaydı da doğrulanması gereken bir özelliktir. Gerçeklenen kayıt sisteminde federe, çalışması için önemli verileri kayıt katmanı ile belli aralıklarla arşivlenmesi için kaydettirir. Kayıt katmanı da yapılandırma dosyasında belirtilen aralıklarla bu verileri dosyaya kaydeder. Herhangi bir anda federe kayıt katmanından eski bir benzetim zamanına gitmesini istediğinde kayıt katmanı kaydedilen bu verileri belirtilen zamandaki değerlerine döndürür. Böylece federe eski durumuna dönerek çalışmasına devam edebilir.

Bunu sağlamak için “kovalanan” federenin yerelinde bulunan koordinat bilgileri kayıt katmanına 10 benzetim zamanında bir kaydetmesi için bildirilmiştir. İlk kayıt 1. benzetim zamanında, sonraki 11. , sonraki 21. şeklinde devam etmektedir. Federe öncelikle 100. benzetim zamanına kadar çalıştırılmış, daha sonra 21. benzetim zamanına dönmesi kayıt katmanı aracılığıyla sağlanmış ve tekrar 100.

benzetim zamanına kadar çalışması sağlanmıştır. İlk durumdaki ve kayıt sonrası durumdaki 100. zamandaki koordinat bilgileri karşılaştırılmıştır. Sonuçta her iki durumda da aynı değerlere sahip olduğu görülmüş ve doğrulama sağlanmıştır.

7. SONUÇ VE ÖNERİLER

Önerilen mimari ile HLA mimarisi kullanılarak geliştirilen benzetimlerde kayıt ve yeniden oynatma için tam dağıtık bir yapı sunulmuştur. Bu mimari ile HLA tabanlı benzetimlerde ağ yükü oluşturmayan, her tür federenin kolaylıkla kullanılabileceği bir yapı oluşturmıştır. Benzetimde iletilen verilerin toplanarak federenin bu verilerle tekrarlanabilir şekilde yeniden oynatılması ile koşumun tekrarlanmasının yanı sıra hata ayıklama ve test gibi imkânlar elde edilmiştir. Kayıt sisteminin kullanılabilirliğinin ve etkinliğini incelemek için önerilen mimari gerçekleştirilerek performans ve doğruluk deneylerine tabi tutulmuş ve makul sonuçlar elde edilmiştir.

Deney sonuçları, önerilen mimarinin etkin bir şekilde çalışabildiğini göstermiştir. Benzetimden gelen verilerin kaydedilme hızı ve oluşan kayıt verisi büyüklüğü makul seviyelerde gözükmektedir. Önerilen yapı doğrudan federelere birleşik olduğundan sistemin ölçeklenebilirlik açısından sorun yaşamayacağı açıktır. Bu mimari sayesinde ağda oluşabilecek gecikmeler de verilerin tutarlılığını etkilemeyecektir. Kaydedilen veriler her federenin kendine özgü olduğu için kısmi dağıtık yapıdaki gibi toplanan verilerin tekrar birleştirilmelerine gerek kalmadan bütün benzetim, federelerin toplanan verileri kullanarak tekrar oynatılmaları ile tekrarlanabilmektedir.

Eşzamanlılık için sunulan çözümle zaman yönetimi olmadığı durumlarda da federeler eşzamanlı olarak tekrar oynatılabileceklerdir. Bu özellikle eğitim amaçlı benzetimler gibi hem performans ihtiyaçlarından dolayı zaman yönetiminin kullanılmadığı hem de benzetimin tekrarının tek bir yerde görüntülendiği benzetimlerde benzetim sonuçlarının tutarlı olarak sergilenmesini sağlamaktadır.

Benzetim verilerinin kaydı RTI katmanı içerisinde gerçekleştirildiğinden federe geliştiricisinin yazılım kodunda değişiklik yapmasına da gerek kalmamaktadır. Basit bir yapılandırma dosyası kullanılarak da kayıt sistemine ait özellikler değiştirilebilmektedir. Böylece herhangi bir federe kayıt ve tekrar oynatma sistemine rahatlıkla uyarlanabilmektedir.

Önerilen mimaride özel verilerin kaydı önemli bir yer tutmaktadır. Federenin özel verileri çok farklı çeşitlerde veriler içerebilir. Bu durumda eğer verinin büyüklüğü çok artarsa kayıt sistemi federenin çalışmasını etkileyebilir. Bunu önlemek için kaydedilecek veri miktarı azaltılmalıdır. RTI'nın ve ağ altyapısının özellikleri bir federenin birim zamanda yolladığı veri miktarını kısıtlamaktadır. Fakat yerel/özel verilerin teorik olarak bir sınırı yoktur(Hafıza, disk sınırları kadar). Dolayısıyla özellikle yerel/özel verilerin kaydı için kaydedilen verinin miktarının azaltılması gerekmektedir. Geliştirilen sistem, güncellenen verilerin bütün olarak kaydedilmesini öngörmektedir. Örneğin bir nesnenin özellikleri güncellendiğinde, güncellenen bu özelliklerin hepsi kaydedilmektedir. Hâlbuki sadece değişen veriler artımlı(incremental) olarak kaydedilse kayıt zamanı azalabilir. Aslında RTI, federelerin sadece değişen verileri yollamalarına olanak kılmaktadır. Federe bu özelliği kullanarak verilerini yolladığında kayıt sistemi zaten sadece bu değiştirilen özellikleri kaydedecektir. Fakat federenin bu özellikleri kullanmadığı durumlarda da kayıt sistemi sadece değişmiş olan verileri dikkate alarak kayıt yapabilir. Böylece sadece değişen veriler kayıt sistemi tarafından belirlenip sadece bu veriler kaydedilir ve hem zaman hem de boyut olarak ciddi kazanç sağlanmış olur. [Feng & Lee, 2006] gibi çalışmalarda bu artımlı(incremental) kayıt incelenmiştir.

Eşzamanlılığın zaman yönetimi olmadığı durumlarda etkileşimlerde sağlandığından bahsedilmiştir. Bu her ne kadar etkili bir çözüm olsa da RTI'nın çok yoğun olduğu durumlarda bu etkileşimlerin diğer federelere gitmesi gecikebilir. Daha iyi bir çözüm, basit bir zaman yönetimi sisteminin kayıt sistemi tarafından TCP/IP üzerinden kurulacak sabit bağlantılarla gerçekleştirilmesidir. RTI gibi bir bilgisayarda sunucu kurulabilir her federede kayıt sistemi içerisinde bulunan istemcilerle bun sunucuya bağlanarak eşzamanlılık sağlanabilir. Fakat bu sistemin de federeleri yavaşlatmadığından emin olunması gerekir.

Federe geçmişine dönme ile ilgili yapılan çalışmalarda federe tarafından yaratılan bir nesnenin, federenin bütün yaşam döngüsü boyunca yaşadığı, federe tarafından silinmediği varsayılmıştır. Bu durumda, federe geçmişe döndüğünde kayıttaki veriler okunarak varolan nesnelerin değerleri ayarlanmaktadır. Fakat yaratılan bir nesne silindikten sonra bu nesnenin varolduğu bir federe geçmişine dönmek istenirse bu durumda HLA'nın isim ayırtma(name reservation) özelliği

önceden kullanılan bir nesne isminin tekrar kullanılmasını önleyecek ve geçmişte varolan bu nesnenin oluşturulmasını engelleyecektir. Bunu çözmek için kayıt katmanının nesne isimleri için federeye bir filtreleme sistemi sağlaması öngörülmüştür. Bu sistemde federe örneğin “gemi” isminde bir nesne yaratmak istediği zaman kayıt katmanı bu ismi “gemi#1” şeklinde değiştirecek ve RTI’den bu isimde bir nesne yaratmasını isteyecektir. Keşfettiği nesnelerin isimlerini de bu açıdan inceleyerek bu şekilde özel karakterlerle değiştirilmiş isimleri anlayarak onları normal haline çevirerek(“gemi”) federeye iletacaktır. Böylece federe isim ayırtma sorunu yaşamadan federe geçmişine dönebilecek ve silinmiş nesnelerini tekrar elde edebilecektir. Tabii ki bu sistem sadece federe geçmişine dönmeye yönelik çalışmalı ve federe normal çalışması sırasında aynı isimde iki nesne yaratmak istediği zaman ayrı iki nesne yaratmamalı ve RTI’nın cevabını federeye iletmelidir.

Zaman yönetimi de federe geçmişine dönme ile çelişen bir durumdur. Bir federe benzetim zamanını her zaman ancak ilerletebilir, federe zamanını geçmiş bir zamana ayarlamak isterse RTI’nın zaman yönetimi buna izin vermeyecektir. Bu çalışmadaki geçmişe dönme deneylerinde zaman yönetimi kullanılmadığından bu sorun yaşanmamıştır. Bu problemi çözmek yine kayıt katmanı tarafında halledilebilir. Eşzamanlılığın sağlanmasına benzer bir çalışma şekli ile eğer bir federe geçmişe dönmek isterse bu durum diğer bütün federelere(bu federelerdeki kayıt katmanına) bildirilmeli ve bütün federeler federasyondan çıkmalı, federasyon silinmeli, federeler federasyona tekrar girerek federasyonu tekrar oluşturmalı ve istenilen geçmiş zamana ilerlemelidirler. Daha sonra eski nesnelerin kaydı gibi işlemler yapılabilir. Zaman yönetiminin kullanılmadığı durumda da tek bir federenin değil de bütün federasyonun geçmişe dönmesi isteniyorsa tutarlı bir koşum için yine kayıt katmanının bütün federelerin geçmiş duruma dönmelerini sağlaması gerekir.

Sonuç olarak önerilen ve geliştirilen tam dağıtık kayıt ve yeniden oynatma mimarisi ile HLA tabanlı benzetimler için bir çözüm sunulmuştur. Bu mimarideki eşzamanlılık çözümü ile tekrarlanabilir bir yeniden oynatma sistemi sağlanmıştır. Benzetim verilerinin yanı sıra özel verilerin kaydını da mümkün kılarak benzetim geçmişine dönebilme mümkün kılınmıştır. Yapılan deneyler ile de bu sistemin etkinliği, hızı ve uygulanabilirliği ortaya konulmuştur.

KAYNAKLAR

- [1] Bachinsky Stephen T., Tarbox Glenn H., and Powell Edward T., Data collection in an HLA environment. *Proceedings of the 1997 Spring Simulation Interoperability Workshop, Orlando, FL, Institute for Simulation and Training at the University of Central Florida, 1999.*
- [2] Feng Thomas Huining, Lee Edward A., Incremental Checkpointing With Application To Distributed Discrete Event Simulation, *Proceedings of the Winter Simulation Conference, Monterey, California, USA, December 3-6, 2006.*
- [3] Fujimoto Richard M., Parallel and Distribution Simulation Systems, *John Wiley & Sons, Inc., New York, NY, USA, 1999.*
- [4] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Federate Interface Specification, IEEE Std 1516.1-2000
- [5] Kasbeka Mangesh, Yajnik Shalini, Klemm Reinhard, Issues in the Design of a Reflective Library for Checkpointing C++ Objects, *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems, Lausanne, Switzerland, 1999.*
- [6] Li Wei, Yang Ming, Wang Zi-Cai, Flexible Simulation Data Collection and Replay Tool, *International Conference on Machine Learning and Cybernetics, pages 1210-1215, Dalian, China, 13-16 Aug. 2006.*
- [7] Magee Gerry, Shanks Graham, Lessons Learned from an Implementation of a Fully Distributed Data Collection Tool, *Proc of the 1999 Spring Simulation Interoperability Workshop, Orlando Florida, USA, March 1999.*
- [8] Ronngren Robert, Liljenstam Michael, Montagnat Johan, Transparent Incremental State Saving in Time Warp Parallel Discrete Event Simulation, *Proceedings of 10th Workshop on Parallel and Distributed Simulation, Philadelphia, USA, 1996.*
- [9] Song Heng-Jie, Shen Zhi-Qi, Miao Chun-Yan, Tan Ah-Hwee, Zhao Guo-Peng, The Multi-Agent Data Collection in HLA-based Simulation System, *21st International Workshop on Principles of Advanced and Distributed Simulation, San Diego, CA, USA, 12-15 June 2007.*
- [10] Van Hook Daniel J., Calvin James O., Execution Logging and Replay-issues and Approaches, *Proc of the 1997 Fall Simulation Interoperability Workshop, Orlando, FL, 1997.*
- [11] Vasend Gerald, After Action Review System Development Trends, *Proceedings of 1995 Winter Simulation Conference, Arlington, VA, USA, 3-6 Dec 1995.*
- [12] West Darrin, Panesar Kiran, Automatic Incremental State Saving, *10th Workshop on Parallel and Distributed Simulation, Philadelphia, USA, 1996.*
- [13] Wilcox Pauline A., Burger Albert G., Hoare Peter. Advanced distributed simulation: a review of developments and their implication for data collection and analysis. *Simulation Practice and Theory, vol. 8, No. 3-4, pp. 201-231, 2000.*

[14] www.boost.org/doc/libs/release/libs/serialization

ÖZGEÇMİŞ

Emrah KAYA, 1984 yılında Kayseri’de doğdu. İlkokulu Ankara’da Türk-İş Blokları İlkokulu’nda okudu. Ortaokulu ve liseyi Ankara Gazi Anadolu Lisesi’nde tamamladı. 2002 yılında Marmara Üniversitesi, Mühendislik Fakültesi Bilgisayar Mühendisliği bölümüne girerek 2006 yılında bu bölümden mezun oldu. Mezun olduğu yıl TÜBİTAK Marmara Araştırma Merkezi Bilişim Teknolojileri Enstitüsü’nde araştırmacı olarak çalışmaya başladı. Aynı yıl, Gebze İleri teknoloji Enstitüsü’nde Bilgisayar Mühendisliği Anabilimdalı’nda yüksek lisans eğitimine başladı. Benzetim teknolojileri ve yazılım mühendisliği konuları ilgi alanlarını oluşturmaktadır.