

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

**HLA TEMELLİ KOŞUM KAYIT VE TEKRAR OYNATIM FEDERESİ
GERÇEKLEŞTİRİMİ**

YÜKSEK LİSANS TEZİ

Selçuk Giray ÖZDAMAR

Anabilim Dalı : Bilgisayar Mühendisliği

Programı : Bilgisayar Mühendisliği

EYLÜL 2010

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

**HLA TEMELLİ KOŞUM KAYIT VE TEKRAR OYNATIM FEDERESİ
GERÇEKLEŞTİRİMİ**

**YÜKSEK LİSANS TEZİ
Selçuk Giray ÖZDAMAR
(504061529)**

**Tezin Enstitüye Verildiği Tarih : 13 Eylül 2010
Tezin Savunulduğu Tarih : 30 Eylül 2010**

**Tez Danışmanı : Prof. Dr. Nadia ERDOĞAN (İTÜ)
Diğer Jüri Üyeleri : Yrd. Doç. Dr. Feza BUZLUCA (İTÜ)
Yrd. Doç. Dr. Yunus Emre SELÇUK
(YTÜ)**

EYLÜL 2010

Eşime ve aileme,

ÖNSÖZ

Çalışmamın ortaya çıkmasında emeği bulunan başta eşim olmak üzere, tez danışmanım sayın Prof. Dr. Nadia Erdoğan'a, gerekli bilgi birikimini edinmemde her türlü desteği veren ve tezimi sonuçlandırmam hususunda özverili davranışlarından ötürü arkadaşlarıma teşekkür ederim. Çalışmamın bu alanda yapılacak çalışmalara ışık tutmasını temenni ederim.

Eylül 2010

Selçuk Giray ÖZDAMAR

Bilgisayar Mühendisi

İÇİNDEKİLER

Sayfa

ÖNSÖZ.....	v
İÇİNDEKİLER	vii
KISALTMALAR	ix
ÇİZELGE LİSTESİ.....	xi
ŞEKİL LİSTESİ.....	xiii
ÖZET.....	xv
SUMMARY	xvii
1. GİRİŞ	1
2. YÜKSEK SEVİYELİ MİMARİ	3
2.1 Yüksek Seviyeli Mimarının Tarihsel Gelişimi.....	3
2.2 HLA Kavramları	5
2.3 HLA Bileşenleri	6
2.3.1 HLA kuralları.....	7
2.3.2 Arayüz tanımlamaları.....	7
2.3.3 Nesne model şablonu (OMT).....	9
2.3.3.1 Federasyon nesne modeli (FOM)	9
2.3.3.2 Benzetim nesne modeli (SOM)	10
2.3.3.3 Yönetim nesne modeli (MOM)	10
2.3.3.4 Nesne model şablonu bileşenleri	10
3. KOŞUM KAYIT ve TEKRAR OYNATIM.....	13
3.1 Dağıtık Etkileşimli Benzetim	14
3.2 Yüksek Seviyeli Mimari.....	15
3.3 Pasif Koşum Kayıt İşlemi	15
3.4 Koşum Kayıt Federesi.....	15
3.5 Mimari Yaklaşımlar	16
3.5.1 Merkezi mimari yaklaşım	16
3.5.2 Dağıtık mimari yaklaşım.....	18
3.5.3 Tam dağıtık mimari yaklaşım	20
4. KOŞUM KAYIT VE TEKRAR OYNATIM FEDERE UYGULAMASI.....	23
4.1 Genel Bakış	23
4.2 Yazılım Teknolojileri Ve Yardımcı Hla Araçları	24
4.2.1 RPR FOM	24
4.2.2 MAK RTI.....	24
4.2.3 VR-Forces	25
4.2.4 QT uygulama geliştirme kütüphanesi	26
4.3 Benzetim Yönetim Federesi (Simulation Manager Federate).....	27
4.3.1 Federe yaşam döngüsü etkileşim sınıfları.....	27
4.3.2 Benzetim yönetim federesi çalışma esası.....	30
4.3.3 Federe yaşam döngüsü	32
4.4 Benzetim Nesne Modeli Yapısı.....	32

4.4.1 Etkileşim sınıfı tanımlama	33
4.4.2 Nesne sınıfı tanımlama	34
4.5 Federe Mimarisi ve Tasarımı	36
4.5.1 RTI elçisi (RTIAmbassador) ve Federe elçisi (FederateAmbassador)	36
4.5.2 Uygulamanın federasyona katılımı	39
4.5.3 Federe SOM yapısı	41
4.5.4 Abone ve yayınlama istek işlemlerinin yapılması	42
4.5.5 Geri çağrım fonksiyonları	45
4.5.6 Uygulamanın federasyondan çıkışı	45
4.5.7 Özel etkileşim mesajları	47
4.6 Koşum Kayıt Birimi (Logger Module)	49
4.6.1 Koşum verisi saklama	49
4.6.2 Serileştirme (Serialization)	52
4.6.3 Etkileşim mesajlarının kaydedilmesi	56
4.6.4 Nesne keşif mesajlarının kaydedilmesi	57
4.6.5 Nesne güncelleme mesajlarının kaydedilmesi	57
4.6.6 Nesne silinme mesajlarının kaydedilmesi	58
4.6.7 Koşum kaydı sonlandırma	59
4.6.8 Kayıt dosyası yapısı	59
4.6.9 Yeni kayıt sınıfı eklenmesi	60
4.6.10 Koşum kayıt birimi mimari tasarım detayları	61
4.6.10.1 Gözlemci tasarım deseni (observer design pattern)	61
4.6.10.2 Nesne güncelleme mesajı kayıt eşiği (update threshold)	62
4.7 Tekrar Oynatma Birimi (Replay Module)	63
4.7.1 Koşum kayıt dosyasının okunması	65
4.7.2 Kayıtların federasyonda yayınlanması	66
4.7.3 Koşum tekrar oynatımın başlatılması / bekletilmesi	68
4.7.4 Koşum tekrar oynatımın istenilen zamana ilerletilmesi	68
4.8 Veri Tabanı Aktarım Birimi (Export Module)	68
4.8.1 SQL komut dosyaları ile veritabanı yapısının oluşturulması	70
4.8.1.1 Veri tabanının fiziki olarak oluşturulması	71
4.8.1.2 Veri tabanı tablolarının oluşturulması	72
4.8.1.3 Saklı yordamların kaydedilmesi	74
4.8.2 Kod üretim teknikleri	76
4.8.2.1 Kod üretiminde programlama dilleri karşılaştırımı	78
4.8.2.2 Katman kod üretim yöntemi	79
4.8.3 Kod üretiminin aktarım katmanı sınıflarına uygulanması	80
4.8.3.1 Ruby programlama diliyle kod üretimi gerçekleştirilmesi	84
4.8.3.2 Veri tabanı yığın aktarım işlemi (bulk insert)	84
4.9 Koşum Analizi	87
4.9.1 Çıktıların analiz edilmesi	89
4.9.2 Koşum analiz uygulaması	90
4.9.3 Analiz işlemlerinde saklı yordam kullanım avantajları	92
4.9.4 Uygulama kaynak kodu seviyesinde saklı yordam çağırımı	93
4.9.5 İsteğe uyarlanmış koşum analiz işlemi	94
5. SONUÇ VE ÖNERİLER	99
KAYNAKLAR	105

KISALTMALAR

API	: Application Programmers Interface
CSV	: Comma Separated Values
DARPA	: Defense Advanced Research Projects Agency
DDM	: Data Distribution Management
DM	: Declaration Management
DoD	: United States Department of Defense
FDD	: FOM Document Data
FOM	: Federation Object Model
HLA	: High Level Architecture
IEEE	: Institute of Electrical and Electronics Engineers
ISO	: International Standards Organization
M&S	: Modeling and Simulation
MOM	: Management Object Model
OMT	: Object Model Template
PDU	: Protocol Data Unit
RDBMS	: Relational Database Management System
RID	: RTI Initialization Data
RTI	: Runtime Infrastructure
SOM	: Simulation Object Model
XML	: Extensible Markup Language

ÇİZELGE LİSTESİ

Sayfa

Çizelge 4.1 : Benzetim senaryosu etkileşim mesajı.....	28
Çizelge 4.2 : Koşum başlangıç etkileşim sınıfı.	28
Çizelge 4.3 : Koşum beklet/durdur etkileşim mesajı.....	29
Çizelge 4.4 : Senaryo sonlandır etkileşim sınıfı.	30
Çizelge 4.5 : Örnek senaryo dosyası içeriği.	31
Çizelge 4.6 : Benzetim nesne modelinde etkileşim sınıfı tanımlama.	33
Çizelge 4.7 : Benzetim nesne modelinde nesne sınıfı tanımlama.	34
Çizelge 4.8 : Benzetim yönetim federesi SOM içeriği.	35
Çizelge 4.9 : Federasyon ve federe yönetim fonksiyonları.	38
Çizelge 4.10 : Etkileşim ve nesne mesajı ileten fonksiyonlar.	38
Çizelge 4.11 : Abone/Yayın isteği bildiren fonksiyonlar.	39
Çizelge 4.12 : Uygulama SOM dosyasında etkileşim sınıfı tanımlama.	41
Çizelge 4.13 : Uygulama SOM dosyasında nesne sınıfı tanımlama.....	42
Çizelge 4.14 : Etkileşim sınıfı abone istek fonksiyonu.	43
Çizelge 4.15 : Etkileşim sınıfı yayınlama istek fonksiyonu.	43
Çizelge 4.16 : Nesne sınıfı abone istek fonksiyonu.....	43
Çizelge 4.17 : Nesne sınıfı yayınlama istek fonksiyonu.	43
Çizelge 4.18 : Boost nesne serileştirme kod örneği.....	54
Çizelge 4.19 : Koşum kayıt veri yapıları.	55
Çizelge 4.20 : InteractionType serileştirilebilir veri yapısı.	56
Çizelge 4.21 : DiscoverType serileştirilebilir veri yapısı.	57
Çizelge 4.22 : UpdateType serileştirilebilir veri yapısı.	57
Çizelge 4.23 : Nesne güncelleme mesajı kayıt fonksiyonu.	58
Çizelge 4.24 : RemoveType serileştirilebilir veri yapısı.	58
Çizelge 4.25 : Yeni kayıt sınıfı eklenmesi.....	61
Çizelge 4.26 : Nesne oluşturma.	62
Çizelge 4.27 : Koşum kayıt dosyası okuma fonksiyonu.	66
Çizelge 4.28 : Veri tabanı oluşturma sql komut dosyası içeriği.....	72
Çizelge 4.29 : Veri tabanı tablosu oluşturma komutu.	73
Çizelge 4.30 : Sabit uzunluklu FOM veri yapıları.....	74
Çizelge 4.31 : Saklı yordam kaydetme.	75
Çizelge 4.32 : Nesne yazdırma sınıf başlık üretim şablonu.....	82
Çizelge 4.33 : Kod üretici tarafından üretilen başlık dosyası.....	82
Çizelge 4.34 : Bilgi sınıf üretici xml girdisi.	83
Çizelge 4.35 : Bilgi sınıfı başlık dosyası üretim şablonu.	83
Çizelge 4.36 : SurfaceVessel.csv virgül ayrımlı dosya içeriği.	85
Çizelge 4.37 : Yığın aktarım T-SQL komutu.	87
Çizelge 4.38 : Saklı yordam kaydetme.	91
Çizelge 4.39 : Saklı yordam çağırısı.....	91
Çizelge 4.40 : Analiz işlemlerinde sql ifadesi kullanımı.	92
Çizelge 4.41 : Kaynak kodu seviyesinde saklı yordam çağırımı.....	94
Çizelge 5.1 : Hizmet karşılaştırma.	104

ŞEKİL LİSTESİ

Sayfa

Şekil 3.1 : Merkezi koşum kayıt ve tekrar oynatım mimarisi.....	17
Şekil 3.2 : Merkezi koşum kayıt ve tekrar oynatım mimarisi kavramsal gerçekleştirimi	17
Şekil 3.3 : Dağıtık koşum kayıt ve tekrar oynatım mimarisi	18
Şekil 3.4 : Dağıtık koşum kayıt ve tekrar oynatım mimarisi kavramsal gerçekleştirimi	19
Şekil 3.5 : Tam dağıtık koşum kayıt ve tekrar oynatım mimarisi	21
Şekil 4.1 : MAK RTI sunucu yazılımı	25
Şekil 4.2 : VR-Forces uygulaması	26
Şekil 4.3 : Benzetim yönetim federesi kullanıcı arayüzü	30
Şekil 4.4 : Federe yaşam döngüsü	32
Şekil 4.5 : Elçi sınıfları ile federe-federasyon iletişim arayüzü.....	37
Şekil 4.6 : Uygulamanın federasyona katılım akış diyagramı	40
Şekil 4.7 : Federe abone ve yayınlama istek akış diyagramı	44
Şekil 4.8 : Uygulamanın federasyondan çıkış akış diyagramı.....	46
Şekil 4.9 : Koşum başlangıç mesajı ile kayıt işleminin başlaması	48
Şekil 4.10 : Koşum kayıt federesi etkileşim mesajları durum diyagramı.....	49
Şekil 4.11 : Koşum verisi saklama yöntemleri	50
Şekil 4.12 : RTI– federe – koşum kayıt birimi kavramsal modeli	52
Şekil 4.13 : RTI Mesajlarının Koşum Kayıt İşlemi Akış Diyagramı	55
Şekil 4.14 : Logger sınıf diyagramı	62
Şekil 4.15 : Koşum tekrar oynatım için kayıt dosyası seçimi	64
Şekil 4.16 : RTI – federe – tekrar oynatma birimi kavramsal modeli	65
Şekil 4.17 : Koşum tekrar oynatım akış diyagramı	67
Şekil 4.18 : Tekrar oynatımda koşumun istenilen zamana alınması	68
Şekil 4.19 : Veri tabanı aktarım işlemi blok diyagramı.....	69
Şekil 4.20 : Veri tabanı aktarım birimi akış diyagramı	70
Şekil 4.21 : Surfacevessel nesne sınıfı veri tabanı tablo yapısı	73
Şekil 4.22 : Saklı yordam çalıştırma ve sonuçların alınması.....	76
Şekil 4.23 : Katman kod üretici işleyişi	79
Şekil 4.24 : Yazdırma sınıfları sınıf diyagramı	80
Şekil 4.25 : Yığın aktarım işlemi blok diyagramı	85
Şekil 4.26 : Kayıtların veri tabanı tablosuna tek tek aktarım yaklaşımı.....	86
Şekil 4.27 : Kayıtların yığın aktarım ile tek adımda aktarım yaklaşımı.....	86
Şekil 4.28 : Koşum analizi uygulamasında sorgulama yapma	92
Şekil 4.29 : Koşum analiz uygulamasında sql ifadesi işletilmesi.....	93
Şekil 4.30 : Uyarlanmış koşum analiz işlemi için sınıf tablosu seçimi	95
Şekil 4.31 : İsteğe uyarlanmış koşum analizi kıstas giriş ekranı	95
Şekil 4.32 : Kullanıcı tanımlı sorgu sonuç ekranı	96

HLA TEMELLİ KOŞUM KAYIT VE TEKRAR OYNATIM FEDERESİ GERÇEKLENMESİ

ÖZET

Modelleme ve benzetim sistemlerinin 1980'li yılların sonlarından itibaren askeri alanlarda, özellikle Amerikan Savunma Bakanlığı (United States Department of Defense - DoD) tarafından kullanımı giderek yaygınlaşmış, Amerikan ordusu Simulator Network (SIMNET) programını uygulamaya almıştır. Belirli standartların sağlanması amacıyla, birden fazla bilgisayar arasında platform düzeyinde gerçek zamanlı savaş oyunu yürütebilmek için Dağıtık Etkileşimli Benzetim (Distributed Interactive Simulations - DIS) protokolü açık standart olarak kabul edilmiştir. Geliştirilen benzetim sistemlerinin birlikte çalışabilirliğini sağlamak için Bütünleşik Seviye Benzetim Protokolü (Aggregate Level Simulation Protocol - ALSP) ortaya çıkmıştır. Mevcut prokollerin yeniden kullanılabilirlik ve birlikte çalışabilirlik açısından eksik olması Yüksek Seviyeli Mimarinin (High Level Architecture - HLA) ortaya çıkmasında etkili olmuştur. 2000 yılında bu çalışmalar IEEE tarafından 1516 standardı (HLA1516) olarak tanımlanmıştır. HLA dağıtık benzetim sistemleri için genel amaçlı mimari olarak kabul edilmiştir.

Modelleme ve benzetim sistemlerinde koşum sırasında üretilen verinin kaydedilerek saklanması, kaydedilen veriler ile daha sonra koşumun tekrar oynatılması “Koşum Kayıt ve Tekrar Oynatım” olarak nitelendirilmektedir. Benzetim sistemlerinin geliştirilme sürecinde hata ayıklama, eğitim amaçlı sistemlerde belirli bir senaryonun/akışın tekrar edilmesi, koşum sonu inceleme ve değerlendirme, analiz ve ölçüm, veri görselleştirmesi ve raporlama gibi ihtiyaçlar koşum kayıt ve tekrar oynatım sistemlerinin varlığını zorunlu kılmıştır.

Bu çalışma merkezi mimari yaklaşıma sahip koşum kayıt ve tekrar oynatım federesinin kod üretme teknikleriyle gerçekleştirilmesini ele almıştır. Geliştirilen federenin standartlara uygunluğu ve farklı FOM'lar ile çalışabilir olması hedeflenmiştir. Ayrıca kaydedilen koşum bilgisinin analiz ve raporlama amaçlı olarak ilişkisel veri tabanı sistemlerinde (Relational Database Management System - RDBMS) saklanabilmesi amaçlanmış, farklı veri tabanı sistemleri ile çalışabilir olması hedeflenmiştir.

HLA BASED EXECUTION LOGGING AND REPLAY FEDERATE IMPLEMENTATION

SUMMARY

In the late 1980's there was a proliferation of modeling and simulation use within the Department of Defense (DoD). The Defense Advanced Research Projects Agency (DARPA) Simulation Networking (SIMNET) program had been adopted by the U.S. Army. Efforts were underway in the real-time platform level community to broaden and standardize the protocol used in that environment. This ultimately became the Distributed Interactive Simulation (DIS) protocol. At the same time, aggregated discrete event simulations, used to support higher level war games, were being brought together using the Aggregate Level Simulation Protocol (ALSP), which enabled synchronization of simulation times as well as features such as save and restore, which were not generally seen as requirements by the real-time simulation community. Re-usability and interoperability of the existing simulation protocols is missing has been effective in the initiation of the work on the High Level Architecture (HLA). The IEEE version of the HLA specification (HLA1516) was approved in September of 2000, and accepted as a general-purpose architecture for distributed simulation systems.

In modeling and simulation systems "Data Collection and Replay" can be described as the collection of the data generated during simulation execution and replay of collected data after simulation execution. In distributed simulation exercises collection of data exchanged between simulation participants during an execution has been accomplished using data loggers. Data collection and replay (DC&R) of simulation systems are useful for a number of purposes including debugging, evaluation, after action review, replay of particular scenario demonstrations and analysis.

The purpose of this study is to design and implement a "Data Collection and Replay" federate application. The federate application is intended to comply with IEEE 1516 HLA standards and compatible with reference FOMs such as SISO RPR FOM. For the post exercise analysis the application developed in this study can export the simulation data to Relational Database Management Systems (RDBMS) and also compatible with different RDBMS.

1. GİRİŞ

Bu çalışmada HLA standardına uygun koşum kayıt ve tekrar oynatım federe uygulaması geliştirilmesi konusu incelenmiştir. Hazır ticari çözümlerin fiyat dezavantajı ve benzetim sistemlerinin tüm ihtiyaçlarını karşılamada yetersiz kalabileceği düşüncesinden yola çıkılarak benzetim sisteminin ihtiyaçları doğrultusunda böyle bir uygulamanın geliştirilmesi ele alınmıştır.

Uygulamanın farklı nesne modelleri ile koşum kaydı ve tekrar oynatım işlemi yapabilmesi hedeflenmiştir. Geliştirilen uygulamayı nesne model değişikliklerinden mümkün olduğunca yalıtacak bir yazılım altyapısı tasarımı ve geliştirilmesi ele alınmıştır. Kaydedilen koşumun analizini yapabilmek için ilişkisel veri tabanına aktarım katmanının kod üretim teknikleri ile tasarımı ve geliştirilmesi ele alınmıştır.

Çalışmanın bir sonraki bölümünde “IEEE 1516” standardı olarak kabul edilen “Yüksek Seviyeli Mimari” nin gelişim süreci anlatılarak mimarinin genel kavramları, tanımlamaları ve temel bileşenleri hakkında bilgiler verilmiştir.

Üçüncü bölümde “Koşum Kayıt ve Tekrar Oynatım” uygulamaları hakkında genel bilgilere değinilmiş, farklı mimari yaklaşımlar avantaj ve dezavantajları ile birlikte değerlendirilmiştir.

Dördüncü bölümde yapılan çalışma mimari detayları ile birlikte detaylı olarak anlatılmıştır. Uygulamanın geliştirilmesinde kullanılan yazılım teknolojileri ve araçlara değinilmiştir. Koşum kayıt ve tekrar oynatım işleminin nasıl gerçekleştirildiği, ilişkisel veri tabanına aktarım katmanının kod üretim teknikleri ile oluşturulması ve detayları anlatılmıştır. Benzetim sisteminin analiz gereksinimleri doğrultusunda veri tabanında sorgulama ve analiz yapılmasına imkan tanıyan koşum analiz uygulaması detayları ile birlikte ele alınmıştır.

Sonuç bölümünde geliştirilen uygulama artı ve eksileri ile birlikte değerlendirilerek, diğer koşum kayıt / oynatım araçları ile özellik bakımından karşılaştırması yapılmıştır. Uygulamanın geliştirilmesi gereken yönleri de bu bölümde ele alınmıştır.

2. YÜKSEK SEVİYELİ MİMARİ

Modelleme ve benzetim sistemlerinin 1980'li yılların sonlarından itibaren askeri alanlarda, özellikle Amerikan Savunma Bakanlığı (United States Department of Defense - DoD) tarafından kullanımı giderek yaygınlaşmış, Amerikan ordusu Simulator Network (SIMNET) programını uygulamaya almıştır. Belirli standartların sağlanması amacıyla, birden fazla bilgisayar arasında platform düzeyinde gerçek zamanlı savaş oyunu yürütebilmek için Dağıtık Etkileşimli Benzetim (Distributed Interactive Simulations - DIS) protokolü açık standart olarak kabul edilmiştir. DIS protokolü uzay araştırmaları ve tıp gibi sivil alanlarda da kullanılmıştır. Geliştirilen benzetim sistemlerinin birlikte çalışabilirliğini sağlamak için Bütünleşik Seviye Benzetim Protokolü (Aggregate Level Simulation Protocol - ALSP) ortaya çıkmıştır. Mevcut prokollerin yeniden kullanılabilirlik ve birlikte çalışabilirlik açısından eksik olması Yüksek Seviyeli Mimarinin (High Level Architecture - HLA) ortaya çıkmasında etkili olmuştur. 2000 yılında bu çalışmalar IEEE tarafından 1516 standardı (HLA1516) olarak tanımlanmıştır. HLA dağıtık benzetim sistemleri için genel amaçlı mimari olarak kabul edilmiştir.

2.1 Yüksek Seviyeli Mimarinin Tarihsel Gelişimi

1980'lerin sonlarına doğru (1990'lı yılların başında) modelleme ve benzetim sistemlerinin Amerikan Savunma Bakanlığı (United States Department of Defense - DoD) tarafından kullanımı giderek yaygınlaşmış, Amerikan ordusu SIMNET programını uygulamaya almıştır [1].

SIMNET : Geniş alan ağı üzerine kurulu tank, helikopter, uçak vb. platformların benzetimleri ve görsel ekranlardan oluşan gerçek zamanlı sanal savaş benzetimidir.

Modelleme ve benzetim alanındaki protokollerin belirli bir standarda oturtulması amaçlanmış ve bu doğrultuda Dağıtık Etkileşimli Benzetim (Distributed Interactive Simulations - DIS) protokolü ortaya çıkmıştır. Bütünleşik Seviye Benzetim Protokolü eşzamanlama (synchronization) kavramını gerçekleştirmiş, kaydet ve yenile (save & restore) gibi yeni özellikler sunmuştur. Mevcut protokollerin yeniden

kullanılabilirlik ve birlikte çalışabilirlik açısından eksik olması DoD'un yeni bir mimari arayışına girmesinde etkili olmuştur.

Yeniden Kullanılabilirlik : Bir benzetimdeki bileşenin farklı benzetim senaryolarında ve uygulamalarında da kullanılabilir olması anlamına gelmektedir. Yeniden kullanılabilirlik bileşenlerin tekrar kodlanmasına gerek kalmadan diğer benzetim bileşenleri ile etkileşim içerisinde çalışmasına imkan tanır.

Birlikte Çalışabilirlik : Farklı benzetim bileşenlerinin dağıtık çalışma ortamında birleştirilebilmesi anlamına gelmektedir. Bu yaklaşım mevcut benzetim bileşenlerinin birbirleri arasındaki etkileşim mekanizmalarının tekrar gözden geçirilmesini zorunlu kılmıştır.

DoD, bileşen temelli dağıtık benzetim sistemleri geliştirimi için, yeniden kullanılabilirliği ve birlikte çalışabilirliği destekleyen esnek bir mimariye ihtiyaç duymuştur. İhtiyaç duyulan ve “Yüksek Seviyeli Mimari” olarak adlandırılan mimarinin detayları DoD tarafından hazırlanan “Modeling and Simulation Master Plan” dökümanında ayrıntılı şekilde anlatılmıştır.

DoD, talep ve maliyetlerin artması ile birlikte DARPA'nın öncülüğünde ihtiyaç duyulan mimarinin çalışmalarını başlatmış, Mart 1995'de “Yüksek Seviyeli Mimari”nin başlangıç tanımı yapılmıştır. Prototip uygulamalardan yararlanılarak, mimarinin temel tanımlamaları (kurallar, arayüz tanımlamaları ve nesne model şablonu) belirlenmiştir. Prototip federeler temel tanımlamaların ortaya çıkmasında önemli rol oynamıştır.

İlk HLA tanımlamasının ortaya çıkmasından sonra 6 aylık dönemlerden oluşan kontrol noktaları belirlenmiş, her bir kontrol noktasında HLA tanımlamalarının yer aldığı yeni bir versiyon yayınlanmış, tanımlamalara uygun geliştirilen RTI yazılımı da güncellenmiştir. Bu çalışmanın sonunda DoD HLA v1.3 (Amerikan Savunma Bakanlığı 1998a, 1998b, 1998c) tanımlamaları ortaya çıkmıştır.

“Yüksek Seviyeli Mimari”nin DoD versiyonu ortaya çıktıktan sonra, mimarinin açık versiyonunun geliştirilmesi için endüstri standardı bir grubun altında çalışmalara devam edilmesi amaçlanmış, bu doğrultuda OMG (Object Management Group) çalışmalara dahil edilmiş, Kasım 1998'de OMG standardına uygun HLA arayüz tanımlamaları yayınlanmıştır. IEEE'nin de çalışmalara katılmasıyla, ticari standartlara uygun yeni HLA tanımlamaları 21 Eylül 2000 tarihinde IEEE 1516

standardı olarak yayınlanmıştır. Bundan sonraki çalışmalar IEEE versiyonu üzerinden devam etmiştir.

“Yüksek Seviyeli Mimari”nin Uluslararası Standartlar Organizasyonu (International Standards Organization - ISO) altında ticarileştirilmesi de amaçlanmıştır. Mimarinin ticari standarda kavuşturulması ile kullanımı yaygınlaşmış, HLA’in sadece savunma alanında değil, farklı alanlarda geliştirilen dağıtık uygulamalar için de genel bir mimari olduğu görülmüştür. Avrupa ülkeleri ve NATO çalışmaları destekleyerek katkılarda bulunmuşlardır. NATO tarafından hazırlanan “Modeling and Simulation Master Plan” dökümanında benzetim sistemlerinin birlikte çalışabilirliği için HLA kullanımı açıkça belirtilmiştir.

DoD, “Yüksek Seviyeli Mimari” nin olabildiğince yaygınlaşması için Amerika içindeki ve dışındaki diğer kullanıcıların da kolayca erişip kullanabilmesini amaç edinmiştir. Günümüzde trafik benzetimi ve üretim/imalat hattı benzetimi gibi sivil uygulamalarda HLA kullanımı görülmektedir.

2.2 HLA Kavramları

Çalışma boyunca HLA temelli benzetim sistemleri için sık kullanılan kavramlara aşağıda değinilmiştir.

Federe : HLA temelli benzetici (simulator) uygulamalara verilen isimdir. Federe RTI’a açılan arayüzü ile bir federasyona dahil olabilir, federasyondan çıkabilir. Tek başına veya diğer simülatör uygulamalarla birlikte çalışabilirler. Belirli türdeki bir federenin federasyonda birden fazla örneği bulunabilir. Örneğin Boeing 747 veya F-16 simülatörü bir federasyonda birden fazla sayıda federe olarak bulunabilir.

Federasyon : Ortak bir amaç için bir araya gelmiş federelerin oluşturduğu sistemdir. Her federasyonun kendisine ait bir Federasyon Nesne Modeli (FOM) bulunur.

Koşum Zamanı Altyapısı (RTI) : HLA arayüz (I/F) tanımlamalarında belirtilen genel servisleri sağlayan ve federasyonun çalışması sırasında senkronizasyon ve veri dağıtımı gibi servisleri yerine getiren yazılımdır.

Federasyon Koşumu : Koşum zamanı altyapısı ile birbirine bağlı federelerin belirli bir zamanda gerçekleştirdikleri etkileşimdir.

Katılımcı Federe : Federasyon Koşumuna katılan her bir üye federeye verilen addır.

Federasyon Nesne Modeli (FOM) : Federasyonda belirli bir amacı yerine getirmek için kořum zamanında karřılıklı alınıp verilen verinin belirli bir formatta tanımlanmasıdır. Nesne sınıfları ve öznitelikleri ile etkileřim sınıfları ve bunların parametrelerinden oluşur.

Simulasyon Nesne Modeli (SOM) : Bir federenin HLA temelli bir federasyona gönderebildiđi ve federasyondaki diđer federelerden alabildiđi verinin belirli bir formatta tanımlanmasıdır.

Etkileřim : Bir federe tarafından federasyon kořumu sırasında gerçekleştirilen ve federasyondaki bir veya birden fazla federe üzerinde etki yaratan eylemdir.

Etkileřim Sınıfı : Belirli özelliklere sahip etkileřim řablonudur. Federeler etkileřim sınıflarının sahip olduđu özellikler dođrultusunda etkileřimler ile ilişkilenebilir.

Etkileřim Parametresi : Etkileřim sınıflarının sahip olduđu özelliklere verilen addır.

Nesne Sınıfı : Ortak özelliklere sahip nesne örneklerinin řablonudur. Gerçek dünyanın farklı seviyelerde soyutlanmış kavramsal gösterimidir. Federelerin birlikte çalışabilirliğini sağlar.

Nesne Sınıfı Özelliđi : Nesne sınıfının belirli bir niteliđidir. Kořum sırasında federeler tarafından gönderilip (publish) alınabilir (subscribe) .

Nesne Sınıfı Örneđi : Belirli bir nesne sınıfının tekil örneđine verilen addır. Federasyon kořumu sırasında, bir nesne örneđinin durumu özelliklerinin toplamıdır.

Pasif Üyelik : Katılımcı bir federenin belirli türdeki verileri (nesne ve etkileřim sınıfları) alabilmek için RTI'a abone isteđinde bulunarak, kořum esnasında herhangi bir veri yayınlamaması pasif üyelik olarak nitelendirilmektedir. Pasif üyelik örneđi genellikle kořum kayıt federelerinde görölmektedir.

2.3 HLA Bileřenleri

Yüksek seviyeli mimari üç temel bileřenden oluşur:

- HLA Kuralları
- Arayüz Tanımlamaları
- Nesne Model řablonu (OMT)

2.3.1 HLA kuralları

HLA kuralları federasyondaki federelerin birbirleri ile uygun şekilde etkileşim kurmasını sağlar. Federe veya federasyonun HLA uyumlu olması için tanımlanmış kurallara uyması zorunludur [2].

Federasyon kuralları:

1. Federasyonun HLA OMT formatına uygun bir FOM'u bulunmalıdır.
2. Federasyondaki tüm nesne örnekleri RTI'da değil, federelerde bulunur.
3. Federasyon koşumu sırasında, federeler arasında FOM verisinin dağıtımı RTI tarafından gerçekleştirilir.
4. Federasyon koşumu sırasında, federelerin RTI ile etkileşimi HLA arayüz tanımlamalarına uygun olarak gerçekleşir.
5. Federasyon koşumu sırasında, bir nesne sınıfı örneğinin özelliği belli bir anda sadece bir federe tarafından sahip olunabilir.

Federe kuralları:

1. Her federenin HLA OMT formatına uygun bir SOM'u bulunmalıdır.
2. Federeler SOM'larında belirtilen nesne sınıflarına ait güncellemeleri ve etkileşim sınıfı mesajlarını gönderip alabilirler.
3. Federeler federasyonun koşum sırasında nesne özelliklerinin sahipliğini SOM'larında belirtildiği şekilde dinamik olarak alıp devredebilir.
4. Nesne güncellemelerinin hangi koşullarda yapılacağı (eşik değer vb.) federelerin SOM'unda belirtilmiştir.
5. Federeler federasyondaki diğer federeler ile veri paylaşımını koordine etmek için kendi yerel zamanlarını yönetebilir.

2.3.2 Arayüz tanımlamaları

HLA arayüz tanımlamaları, dağıtık benzetim ortamında federeler arasındaki işlevsel arayüzleri, iletişimi sağlayan yazılımsal servisleri ve koşum zamanı altyapısını (RTI) tanımlar. Tanımlamalar herhangi bir yazılım dilinden bağımsızdır. Arayüzlere ve servislere uygun geliştirilen federeler yeniden kullanılabilirliği de sağlar [3].

RTI, arayüz tanımlamalarının bir parçası değil, arayüz tanımlamalarına uygun geliştirilmiş yazılıma verilen addır. RTI, HLA uyumlu bir benzetim sistemi için gerekli yazılımsal servisleri içerir. RTI yazılımının farklı versiyonlarına rastlamak mümkündür. Bunlardan birisi DMSO tarafından geliştirilen, HLA v1.3 uyumlu RTI-NG 1.3 yazılımıdır. Ticari amaçlı RTI yazılımlarına, “MÄK Technologies” tarafından geliştirilen “MÄK Real-time RTI” ve “Pitch Technologies”in geliştirdiği “Pitch pRTI” örnek olarak gösterilebilir.

Servisler, DIS ve ALSP gibi eski standartları iyileştirerek, benzetim sistemini ve veri iletişimini birbirinden ayırmayı amaçlar. Federasyonun başlatılması ve sonlanmasını basitleştirir. Federeler arası veri iletişimi için gerekli nesne tanımlamaları ve bunların yöntemini sağlar. Federasyonun zaman yönetimini kolaylaştırır. Federeler arasında verimli iletişim sağlar.

Arayüz tanımlamaları altı temel gruptan oluşur :

Federasyon Yönetimi: Federasyon koşumunun başlatılması, yönetimi ve sonlandırılması ile ilgili servisleri tanımlar.

Deklarasyon Yönetimi: Federeler, federasyona göndereceği ve federasyondan kendisine iletilmesini istediği bilgiyi (nesne ve etkileşimler) veri yönetim servisleri ile bildirmek zorundadır.

Nesne Yönetimi: Bu gruptaki HLA servisleri, nesne sınıfı örneği kayıt ettirme, güncelleme ve silme ile etkileşim gönderme ve alma tanımlamalarını içerir.

Sahiplik Yönetimi: Federeler ve RTI, nesne özelliklerinin sahipliğinin federasyondaki diğer federeler arasında aktarılması için sahiplik yönetimi ile ilgili servisleri kullanır. Nesne örneğinin belirli bir özelliği herhangi bir zamanda sadece bir federe tarafından sahip olunabilir. Koşum sırasında nesne örneğinin belirli bir özelliği sadece sahibi olan federe tarafından güncellenebilir.

Zaman Yönetimi: Federasyon koşumu süresince mesajların tutarlı sırada iletilmesinden zaman yönetimi servisleri sorumludur. Zaman yönetimi servisleri, koşum sırasında, farklı federeler tarafından gönderilen mesajların, federasyondaki diğer bir federeye tutarlı sırada iletilmesine olanak sağlar.

Veri Dağıtım Yönetimi: Federeler yersiz bilgi gönderimi ve alımını azaltmak için DDM servislerinden yararlanır. Gönderici ve alıcı federenin ikisi de ilgili servislerden yararlanarak, RTI'n gereksiz veri iletmesinin önüne geçmiş olur. RTI, kullanıcı tanımlı boyutlar (user-defined dimensions) ile etkileşim veya nesne özelliğinin iletilebilir olup olmadığına karar verir. Gönderici ve alıcıların kendileri için tanımladığı alt-üst sınır değerlerin kesişimi verinin RTI tarafından iletileceği uygun alanı temsil eder.

2.3.3 Nesne model şablonu (OMT)

Yüksek seviyeli mimari, federenin yeniden kullanım ve birlikte çalışabilirliği için iletişimde kullanılan verinin (nesne ve etkileşimler) ortak bir formatta tanımlanmasını zorunlu kılmaktadır. Nesne Model Şablonu (OMT), federasyondaki federeler arasında veri iletişimi için ortak standardı tanımlar, federelerin sahip olduğu yetenekler hakkında bilgi verir. HLA nesne model tasarımı için yazılımsal araçların geliştirilmesinde kolaylık sağlar. Federasyon Nesne Modeli (FOM), Benzetim Nesne Modeli (SOM) ve Yönetim Nesne Modeli (MOM) tanımlamalarını kapsar [4].

HLA nesne modeli:

- Federenin benzetim nesne modelini (Simulation object model - SOM) tanımlamak
- Federasyonun nesne modelini (Federation Object Model - FOM) tanımlamak için kullanılır.

Her iki durumda da, HLA nesne model şablonunun temel amacı federelerin yeniden kullanım ve birlikte çalışabilirliğini sağlamaktır.

2.3.3.1 Federasyon nesne modeli (FOM)

HLA federasyonunda, federeler arası veri iletişiminin gerçekleşebilmesi için federasyon içerisinde ortak bir anlayışın mevcut olması gerekmektedir. FOM'un temel amacı, federeler arası veri iletişimi için gerekli veri tanımlamalarının ortak ve standart bir yapıda sağlanmasıdır.

HLA uyumlu her federasyon bir FOM'a sahiptir. FOM federasyonda veri iletişimi için kullanılan ortak veriyi (nesne ve etkileşim sınıfları ile bunların özellikleri) tanımlar. FOM nesne sınıfları ve bunların özellikleri ile etkileşim sınıfları ve bunların parametrelerinden oluşur. FOM aynı zamanda federelerin birlikte çalışabilirliği için gerekli bilgi modelidir.

2.3.3.2 Benzetim nesne modeli (SOM)

HLA uyumlu her federenin kendisine ait bir SOM'u vardır. SOM, federenin federasyona gönderdiği ve federasyondaki diğer federelerden aldığı bilginin tanımlamalarını içerir. Standart biçimde tanımlanan SOM, bir federenin federasyona katılımının uygun olup olmadığına karar vermeyi de sağlar. SOM'un içeriği de nesne ve etkileşim sınıfları ve bu sınıfların özellik ve parametrelerinin tanımlamalarından oluşur.

FOM ve SOM genel olarak HLA OMT formatına uygun olarak tanımlanır. Ortak nesne model şablonunun olması, SOM'lardan FOM üretilmesine imkan sağlar.

2.3.3.3 Yönetim nesne modeli (MOM)

Federasyon koşumu ile ilgili önceden tanımlı HLA yapılarını (nesne ve etkileşim sınıfları) içerir. FOM içerisinde tanımlanır ve her FOM'da bulunması gerekir. Federe, federasyon koşumuna etki edebilmek için yönetim nesne modeline başvurmalıdır. MOM, federasyonda aşağıdaki işlevleri sağlar :

- Federasyon koşum bilgisine erişimi sağlar.
- Koşum zamanı altyapısının, federasyon koşumunun ve federelerin işleyişini kontrol eder.

2.3.3.4 Nesne model şablonu bileşenleri

Nesne model şablonu bileşenleri aşağıda listelenmiştir. Bu bileşenlerden tez çalışması için önemli görülen bileşenlerin detayı aşağıda anlatılmıştır.

- Nesne Modeli Tanımlama Tablosu
- Nesne Sınıfı Yapı Tablosu
- Etkileşim Sınıfı Yapı Tablosu
- Öznitelik Tablosu
- Parametre Tablosu
- Boyut Tablosu
- Zaman Temsil Tablosu
- Kullanıcı Tanımlı Etiket Tablosu
- Senkronizasyon Tablosu
- İletim Tipi Tablosu
- Anahtar Tablosu (Switches table)
- Veri Tipi Tablosu (Datatype tables)
- Not Tablosu (Notes table)
- FOM/SOM Sözlüğü (FOM/SOM lexicon)

Nesne modeli tanımlama tablosu : Bu tablonun amacı HLA nesne modelinin kimlik bilgisini tanımlamaktır. Bu tablo:

- Yeniden kullanılabilir nesne modeli araştıran geliştiriciler için tanımlama
- Nesne modelinin neden ve nasıl oluşturulduğu
- Nesne modeli hakkında bilgisi olan kişi (Point of Contact - POC) ile ilgili bilgileri içerir.

Nesne sınıfı yapı tablosu: Benzetim veya federasyon etki alanındaki sınıflar ve bunlar arasındaki hiyerarşik ilişkiler (ata/alt-sınıf ilişkileri) nesne sınıf yapısı içerisinde tanımlanır. Aynı ortak özelliklere sahip nesneler kümesi bir HLA sınıfını temsil eder. Belirli bir sınıftan oluşturulmuş her bir nesne bu sınıfın bir örneğidir.

Sınıf tanımlamaları abone/yayın bilgisini de içerir. SOM için abone/yayın bilgisinin alabildiği değerleri aşağıdaki gibidir:

- P: Federe bu sınıfın en az bir özelliğini yayınlatabilir.
- S: Federe bu sınıfın en az bir özelliğine abone olabilir.
- PS: Federe bu sınıfın en az bir özelliğine abone olup, en az bir özelliğini yayınlatabilir.
- N: Federe bu sınıfa ait özelliklerden hiçbirisine abone olamaz, yayınlamaz.

FOM içerisinde tanımlı nesne sınıflarının abone/yayın bilgisinin olması, en az bir federenin bu sınıfa olan abone/yayın isteğini ifade eder.

Etkileşim sınıfı yapı tablosu : Federasyon koşumu sırasında, bir federe tarafından gerçekleştirilen ve diğer federeler üzerinde etki/reaksiyon oluşturan eyleme etkileşim (interaction) adı verilir. Nesne sınıfı yapı tablosu tanımlamalarında olduğu gibi, etkileşim sınıfları ve bunlar arasındaki hiyerarşik ilişkiler (ata/alt-sınıf ilişkileri) bu tabloda tanımlanır. Etkileşimlerin hiyerarşik yapısı genelleştirme veya özelleştirmeler şeklindedir. Örneğin, angajman etkileşimi, havadan-karaya angajman veya gemiden-havaya angajman şeklinde özelleştirilebilir. "HLAinteractionRoot" etkileşim sınıfı FOM veya SOM'da üst-sınıf (superclass) olarak tanımlanabilir. SOM ve FOM için etkileşim sınıfı abone/yayın bilgisi nesne sınıflarında anlatıldığı şekildedir.

Öznitelik tablosu : Benzetim etki alanındaki nesne sınıfları belirli özniteliklere sahiptir. Benzetim koşumu süresince değerleri değişebilen bu öznitelikler (platform konumu, hızı vb.) nesnenin durumunu oluşturur. Nesne örneğinin öznitelikleri RTI aracılığıyla güncellenir, federasyondaki ilgili diğer federelere iletilir. Federelerin ve federasyonların SOM ve FOM'larında nesne sınıflarının özniteliklerini bu tabloda tanımlamaları gerekir. "HLAobjectRoot" nesne sınıfı FOM veya SOM için üst-sınıf (superclass) olarak tanımlanabilir.

Parametre tablosu : Etkileşim sınıflarını sahip olduğu parametreler karakterize eder. Parametreler nümerik, karakter dizisi gibi değerler alabileceği gibi, nesne sınıfı örneğinin öznitelik değerini de alabilir. Etkileşim sınıfı yapı tablosunda tanımlı her bir etkileşim sınıfının tüm parametreleri parametre tablosunda tanımlanmalıdır.

3. KOŞUM KAYIT ve TEKRAR OYNATIM

Benzetim sistemlerinde koşum sırasında nesne durumlarının ve etkileşimlerin kayıt edilerek, kaydedilen koşum verilerinin daha sonra tekrar oynatılması “Koşum Kayıt ve Tekrar Oynatım” olarak nitelendirilmektedir. “Koşum Kayıt ve Tekrar Oynatım”ın günümüz benzetim sistemlerinde önemi ve yararları açıkça görülmüş, farklı kullanım amaçları doğrultusunda kendisine yer bulmuştur. Koşum Kayıt ve Tekrar Oynatım sisteminin farklı kullanım amaçları aşağıda listelenmiştir:

- *Koşum sonu inceleme*: Eğitim amaçlı olarak yapılan benzetim senaryolarında, koşum sonunda inceleme ve değerlendirme.
- *Gösterim*: Belirli bir senaryonun / akışın tekrar edilmesi
- *Analiz*: Benzetimde üretilen verinin görselleştirilmesi ve ölçüm (etkinlik, performans vb.) amaçlı
- *Hata Ayıklama*: Yazılım geliştirme ve entegrasyon süreçlerinde geliştiriciler için hata ayıklama (debugging) amacıyla kullanılmaktadır.

DIS temelli benzetim sistemlerinde koşum kayıt ve tekrar oynatım sistemleri yukarıda bahsedilen amaçlar doğrultusunda yaygın bir şekilde uygulanmış, bu yaklaşımın diğer benzetim alanları için de uygulanabilir olduğu açıkça görülmüştür [5].

Dağıtık benzetim sistemlerinde, özellikle büyük ölçekli sistemlerde, çoğunlukla veri kayıt mekanizması kullanılmaktadır. Kayıtçı bir yandan benzetim ortamındaki iletişim ağı trafiğini dinlerken bir yandan da topladığı veriyi saklamaktadır. Kayıt işlemi genellikle bir dosyaya yapılmaktadır. Kayıt dosyası daha sonra sıralı şekilde okunarak analiz, inceleme veya tekrar oynatma işlemleri için kullanılmaktadır.

SIMNET, DIS ve ALSP temelli benzetim sistemleri için geliştirilen koşum kayıt uygulamaları koşum sırasında yayınlanan tüm veriyi dinleyip kaydetme kabiliyetine sahiptir. Geliştirilen uygulamalarda genellikle tek bir koşum kayıtçısı bu işlemi üstlenmektedir [6].

HLA temelli benzetim sistemlerinin yaygınlaşması geliştiricilerin veri toplama ve analiz konularında stratejilerini tekrar gözden geçirmelerine neden olmuştur. Kullanıcılar koşum sırasında ve koşum bittiğinde kritik cevaplara hızlı şekilde ulaşmak istemekte, federasyonun kapsamı büyüdükçe istenilen cevaplara ulaşmak için gerekli çözümün karmaşıklık düzeyi de artmaktadır. Analiz kavramının temelinde, koşum esnasında toplanan bilginin mümkün olduğu kadar hızlı şekilde hazır edilmesi yatmaktadır. HLA temelli koşum kayıt sisteminin farklı FOM alternatifleri ile uyumlu çalışabilen, ölçeklenebilir bir sistem şeklinde tasarlanması gerekir. Özel amaçlı sorgulara cevap verebilmesi ve kayıt edilen veriyi görselleştirebilmesi kullanıcılar tarafından olması istenilen özellikler arasındadır [7].

DIS ve HLA protokolleri arasında verinin formatı ve yayınlanma yöntemiyle ilgili bazı farklılıklar bulunmaktadır. Koşum kayıt ve tekrar oynatım açısından bakıldığında, HLA ve DIS protokolleri arasındaki bu farklılıklara değinmekte fayda vardır.

3.1 Dağıtık Etkileşimli Benzetim

DIS protokolüne göre, koşum sırasında önceden belirlenmiş sabit bir aralıkla veya nesnenin herhangi bir özelliği değiştiğinde nesnenin tüm durumu benzetim ortamına yayınlanır. Benzetim ortamına yayınlanan veri (Protocol Data Unit - PDU) adı verilen sabit formatlı yapıdadır. Koşum kaydı açısından bakıldığında, DIS protokolünün avantajı olduğu gibi dezavantajı da söz konusudur. Bir nesnenin herhangi bir andaki durumu hesaplama veya birleştirmeye gereksinim duymadan bilinmektedir. Bu durum tekrar oynatım açısından değerlendirildiğinde büyük bir avantajdır. Diğer yandan, bir PDU abone yöntemi ile ilgili diğer federe tarafından alındığında nesnenin hangi özelliklerinin değiştiğini saptamak güçtür. Bu amaçla nesnenin özelliklerinin bir önceki değerleriyle karşılaştırılması gerekmektedir. Ancak bu işlem ek bir yük getirmesi nedeniyle karşımıza bir dezavantaj olarak çıkmaktadır. Veri yönetimi açısından, analiz ve disk boyutu gibi faktörler de göz önünde bulundurulduğunda, delta yöntemi ile nesnenin sadece değişen özelliklerinin kayıt edilmesi daha etkili bir yöntem olacaktır [7].

3.2 Yüksek Seviyeli Mimari

HLA standardının birlikte çalışabilirlik ve ölçeklenebilirlik bakımından daha güçlü olması bu mimariyi öne çıkaran nedenlerdendir. HLA temelli bir benzetim sisteminde, bu iki özellikten dolayı koşum kayıt ve tekrar oynatma işlemleri daha karmaşıktır. RTI, HLA standardında tanımlanmış olan servisleri yerine getirir. Veri kaydı açısından iki önemli nokta söz konusudur. Birincisi RTI sadece iletişim mekanizması sağlar. Federeler ilgilendikleri nesnelerin idame (maintain) edilmesinden kendileri sorumludur. RTI veriyi abone yöntemi ile ilgili federelere ilettikten sonra, federelerin bu veriyi nasıl kullandıkları ile ilgilenmez. HLA temelli bir benzetim sisteminde koşum kayıt sistemi bir federe olmalı, bu federe nesnelerin durumlarını ve etkileşimleri kayıt edebilir olmalı, tekrar oynatma için nesne güncellemeleri yapabilir (object update) ve etkileşim gönderebilir olmalıdır. RTI'nin nesnelerin sadece değişen özelliklerini gönderebilmesi, koşum kayıt sisteminde nesnelerin sadece değişen özelliklerini kayıt etme imkanı sunması açısından yararlı bir özelliktir. İkinci nokta ise, RTI federasyondaki her bir nesne ve etkileşime tekil tanımlayıcı (unique id) atamaktadır. Bu tanımlayıcı bilgi koşum sonrası analiz ve çıkarımlar yapabilmek için önemlidir [7].

3.3 Pasif Koşum Kayıt İşlemi

Benzetimde, koşum sırasında üretilen verinin kayıt işleminde ana hedef, federasyonun genel çalışma performansını etkilemeden olabildiğince fazla bilgiyi toplamak olmalıdır. Bunu en mükemmel şekilde başarmanın yolu da koşum kayıt federesi oluşturmaktan geçmektedir. Koşum kayıt federesi, gerekli nesne ve etkileşimlere abone olurken, koşum sırasında RTI üzerinden herhangi bir veri göndermemelidir. Koşum kayıt federesi her bir nesnenin belirli bir andaki durum bilgisini saklamalı, RTI tarafından nesne güncelleme mesajı alındığında gerekli kayıt işlemlerini yapmalıdır [7].

3.4 Koşum Kayıt Federesi

Koşum kayıt federesine federasyonda aktif rolü olmayan bir federe olarak bakılabilir. FOM'da tanımlı tüm veriye abone olan koşum kayıt federesi, nesnelerin güncellemelerini ve etkileşimlerini kaydeder. Nesne güncellemeleri ve etkileşimlerin

hangi federeden yapıldığı ile ilgili koşum kayıt federesinin bilgisi olmasına gerek yoktur. Koşum kayıt federesi her nesne güncellemesi sonrası “queryAttributeOwnership” fonksiyon çağrımını yaparak, güncellemenin hangi federe tarafından yapıldığını öğrenebilir, ancak koşum esnasında nesne özelliği sahiplik bilgisi (attribute ownership) kayıt federesinin bilgisi dışında değişebileceği için bu türden bir fonksiyon çağrımı yapmak gereksizdir. Koşum kayıt sistemi merkezi tek bir federe şeklinde olabileceği gibi, federasyonda dağıtık durumda birden fazla federeden de oluşabilir [8].

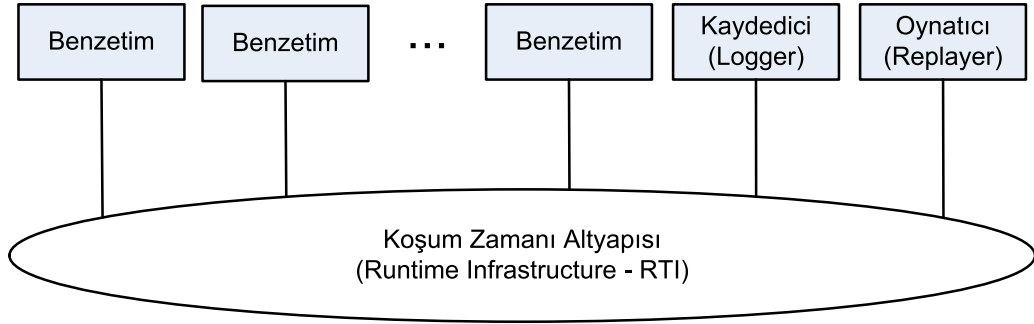
3.5 Mimari Yaklaşımlar

Bu kısımda “Koşum Kayıt ve Tekrar Oynatım” sistemi için farklı mimari yaklaşımlar hakkında bilgi verilecek, her mimari yaklaşım avantaj ve dezavantajları ile birlikte değerlendirilecektir.

3.5.1 Merkezi mimari yaklaşım

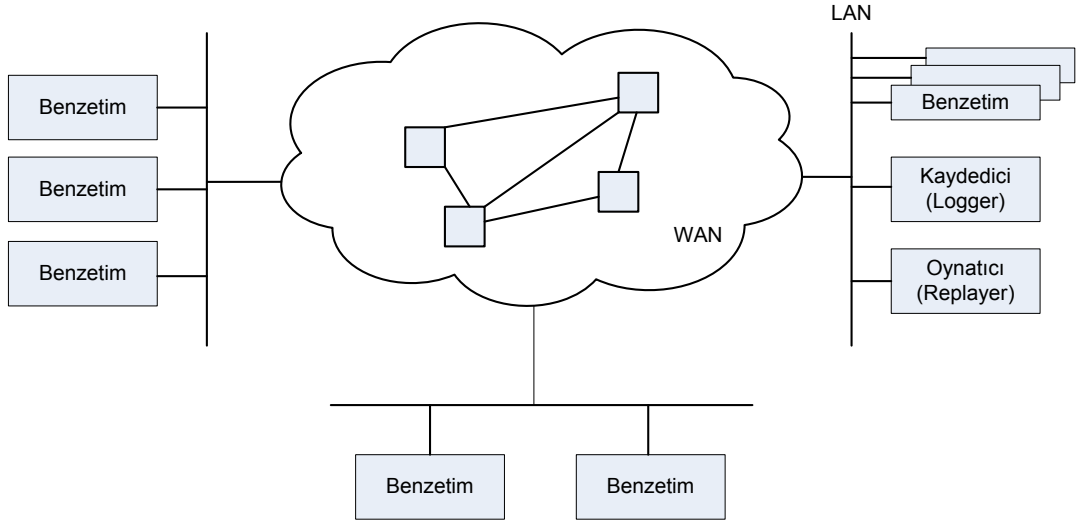
Merkezi Mimari Yaklaşım (Central Approach) Şekil 3.1 ve Şekil 3.2’de gösterilmiştir. Bu senaryoda tek bir merkezi kayıt federesi federasyonun koşum verisini kaydetmektedir [5].

Merkezi yaklaşımın kavramsal olarak somutlaştırılarak resmedildiği Şekil 3.2’de, merkezi kayıt federesinin tek bir yerel ağ (LAN) bölümünde yer aldığı görülmektedir. Merkezi kayıt federesi federasyondaki nesne ve güncellemelere abone olduğu için, federasyon verisi RTI tarafından bu yerel ağ bölümüne doğru yönlendirilecektir. Kayıt federesi bu ağ segmentinde gereksiz ağ trafiği oluşmasına neden olacaktır. Bu segmentte yer alan diğer federeler, kayıt federesinin ilgilendiği verinin büyük bir kısmı ile ilgilenmiyor olabilir. Bu durumda bu segmentte kayıt federesinden dolayı gereksiz ağ trafiği oluşmaktadır.



Şekil 3.1 : Merkezi koşum kayıt ve tekrar oynatım mimarisi

Merkezi mimarinin en büyük avantajı basit olmasıdır. Bütün nesne ve etkileşimlere abone olan tek bir kayıt federesi ile koşum kayıt sistemi gerçekleştirilebilir. Merkezi yaklaşımın bir diğer avantajı ise koşum sonunda federasyonun tüm koşum verisine erişim mümkündür. Dağıtık yapıdaki kayıt federelerde olduğu gibi kayıt verilerinin transfer edilip bütünleştirilmesine gerek yoktur.



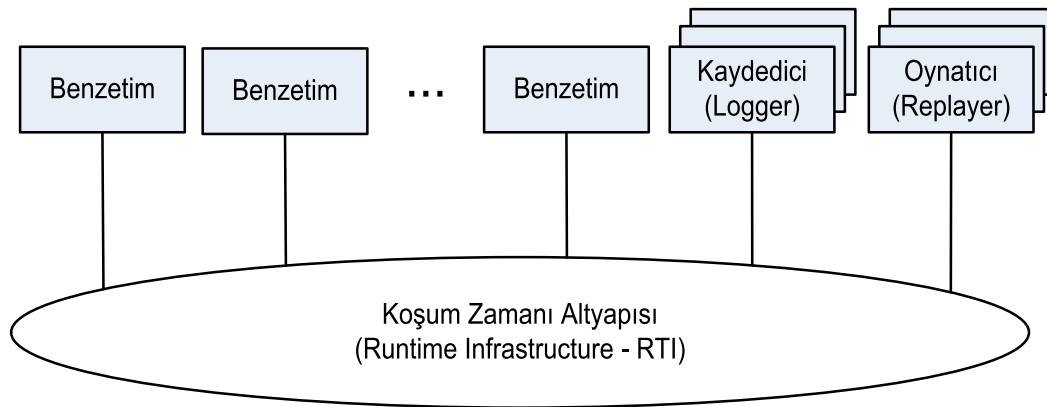
Şekil 3.2 : Merkezi koşum kayıt ve tekrar oynatım mimarisi kavramsal gerçekleştirimi

Diğer taraftan bu mimari yaklaşımın iki büyük dezavantajı bulunmaktadır. Birincisi ve en önemlisi federasyon tarafından üretilen koşum verisinin tek bir yöne doğru iletilmesi sorunudur. Asimetrik veri akışı ağ yönetimini zorlaştırarak ağ bileşenlerinin aşırı yüklenmesine ve özellikle koşum kayıt federesinde işlem yükünün artmasına neden olmaktadır. Diğer dezavantajı ise, federasyonun farklı noktalarında kayıt edilen veriye istenildiği anda ulaşmak mümkün olmadığı için ilgili noktada (federede) bağımsız analiz ve inceleme imkanı sunmamasıdır.

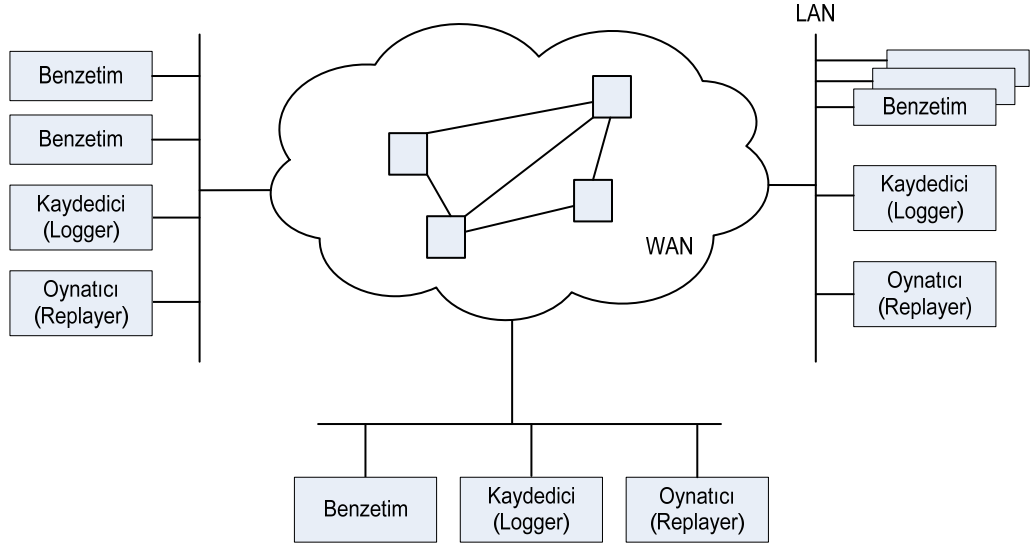
3.5.2 Dağıtık mimari yaklaşım

Dağıtık Mimari Yaklaşımında (Distributed Approach) birden fazla sayıdaki kayıt federesi federasyonun koşum verisini kaydetmektedir. Dağıtık mimarinin kavramsal olarak gösterildiği Şekil 3.3’de, her bir yerel ağ (LAN) segmentinde bir kayıt federesinin yer aldığı görülmektedir. Bu mimari yaklaşımın özünde yatan ana fikir, her bir koşum kayıt federesinin federasyondaki belirli bir federe kümesinin kayıt işlevinden sorumlu olmasıdır. Her bir kayıt federesi federasyonun tüm koşum verisinin belirli bir alt kümesinin kaydının tutulmasından sorumludur. Kayıt federelerinin topladığı koşum verileri daha sonra tek bir veritabanı üzerinde birleştirilmelidir [5].

Şekil 3.3 ve Şekil 3.4’de dağıtık koşum kayıt yaklaşımı gösterilmiştir.



Şekil 3.3 : Dağıtık koşum kayıt ve tekrar oynatım mimarisi



Şekil 3.4 : Dağıtık koşum kayıt ve tekrar oynatım mimarisi kavramsal gerçekleştirimi

Kayıt federelerinin hangi verilerin kaydından sorumlu olacağının belirlenmesinde birkaç yöntem vardır:

- Sorumlu olduğu federelerin yayın / abone (publish / subscribe) mekanizması ile RTI üzerinden aldığı ve gönderdiği tüm nesne güncellemeleri ve etkileşimlerin kayıt altına alınması.
- Sorumlu olduğu federelerin yayın / abone (publish / subscribe) mekanizması ile RTI üzerinden gönderdiği (publish) tüm nesne güncellemeleri ve etkileşimlerin kayıt altına alınması.

Bahsedilen her iki yöntemde de federasyonun çalıştığı ağ üzerindeki yerel ağ segmentlerine gereksiz veri akışı oluşmayacaktır.

Dağıtık mimarinin sağladığı avantajlardan birisi yük paylaşımı sağlamasıdır. Bir kayıt federesi üzerindeki veri yükü sorumlu olduğu federelerin üzerindeki toplam veri yükü kadardır. Kayıt işleminden dolayı federasyonun üzerinde çalıştığı ağdaki veri akışı olumsuz etkilenmez, federasyon koşumu gereksiz meşgul edilmemiş olur. Yerel kayıt dosyasının bulunması yerel birimde koşum gözden geçirmesi (AAR) yapabilme olanağı sunmaktadır.

Diğer taraftan dağıtık mimari yaklaşımın dezavantajları da bulunmaktadır. Her bir kayıt federesinin kaydını tutmakla yükümlü olduğu federelerin abone olduğu nesne ve etkileşimlerden dinamik olarak haberdar olmaları gerekmektedir. Bunun için RTI servislerine ihtiyaç vardır. Diğer bir dezavantaj ise, federasyonun tüm koşum

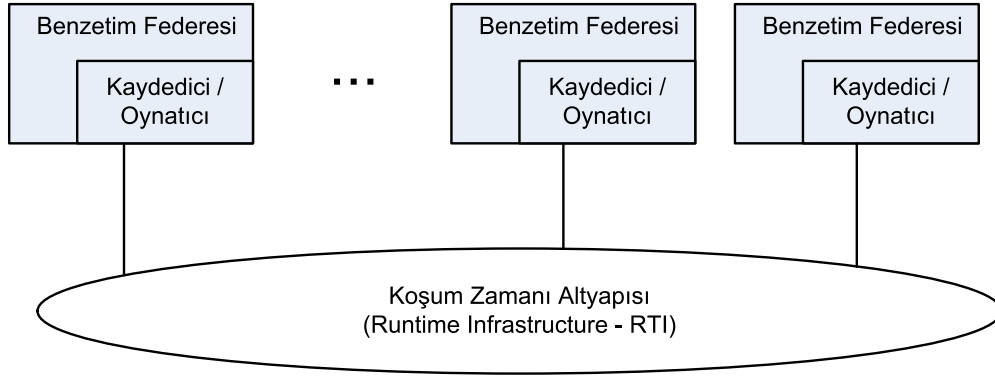
verisinin birleştirilmiş olarak bulunmuyor olmasıdır. Ancak tüm koşum verisinin sonradan birleştirilerek elde edilmesi mümkündür. Bu da beraberinde bazı sorunları getirmektedir. Koşum sırasında yayınlanan bir bilginin birden fazla kayıt federesinde kayıt edilmesi sonucu tekrarlanan bilgiler ortaya çıkabilmektedir. Tekrarlanan bilgilerin tespit edilerek temizlenmesi gerekmektedir. Bunun yanı sıra, tek bir koşum veri tabanı elde etmek için yerel kayıt dosyalarının ağ üzerinde transfer edilmesi gerekmektedir. Federasyonun çalışma performansını etkilememesi ve ağ üzerinde gecikmelere neden olabileceği için bu işlemin yapılması ancak koşum sonu mümkün görünmektedir. [5]

Federasyonun üzerinde çalıştığı ağ Şekil 3.4'deki gibi parçalı yapıdaysa (segmented) dağıtık yaklaşım uygun bir tekniktir. Ancak ağ parçalı yapı arz etmiyorsa bu mimari ağ trafiği açısından herhangi bir avantaj sağlamayacak, merkezi yaklaşımda olduğu gibi çalışacaktır.

3.5.3 Tam dağıtık mimari yaklaşım

Tam Dağıtık Mimari Yaklaşım (Fully Distributed Approach) Şekil 3.5'de gösterilmiştir. Bu mimaride federasyondaki her federenin kendisine ait veri kaydetme/tekrar oynatım modülü bulunur. Federenin RTI sunucusu ile arasındaki gerçekleşen her türlü veri etkileşimi (nesne güncellemeleri, etkileşim mesajları vb.) bu modül tarafından kayıt edilir [9].

Diğer mimari yaklaşımların aksine, kayıt işleminin federelerde yerel olarak yapılmasından dolayı ağ üzerinde gereksiz veri akışı ve yük oluşmaz.



Şekil 3.5 : Tam dağıtık koşum kayıt ve tekrar oynatım mimarisi

Anlatılan diğer mimarilerde olduğu gibi "tam dağıtık mimari" yaklaşımın da dezavantajları bulunmaktadır. Dağıtık mimaride olduğu gibi bu mimaride de federasyonun tüm koşum verisine hemen ulaşmak mümkün değildir [9]. Federelerde yerel olarak bulunan kayıt dosyalarının ağ üzerinde merkezi bir noktaya transfer edilerek tek bir koşum veri tabanı elde etmek mümkündür. Dağıtık mimaride anlatıldığı gibi, tekrarlanan bilgilerin de tespit edilerek temizlenmesi gerekmektedir. Federasyonun çalışma performansını etkilememesi ve ağ üzerinde gecikmelere neden olabileceği için bu işlemin yapılması ancak koşum sonu mümkün görünmektedir.

4. KOŞUM KAYIT VE TEKRAR OYNATIM FEDERE UYGULAMASI

Tez çalışmasında, önceki bölümde anlatılan koşum kayıt ve tekrar oynatım mimari yaklaşımlarından merkezi yaklaşım esas alınmıştır. Merkezi koşum kayıt federesi dahil olduğu federasyonda tek bir yerel ağ (LAN) bölümünde yer alarak benzetim sırasında koşum bilgisini kaydetme işlevini yerine getirir. Kaydedilen koşumun tekrar oynatılması için tekrar oynatım (replay) modunda da çalışabilmesini sağlayacak şekilde tasarlanmıştır. Aynı zamanda kaydedilen koşum verisi üzerinde analiz ve hesaplamalar yapabilmek için kaydedilen verinin ilişkisel veri tabanında saklanabilmesi de sağlanmıştır.

Çalışma kapsamında bir takım hazır ticari benzetim araçları ve açık kaynak kodlu yazılım araçlarından da yararlanılmıştır. Koşum kayıt ve tekrar oynatım federesinin yanı sıra, federasyonun koşum yönetimi için “Benzetim Yönetim Federesi” (Simulation Manager Federate) tasarlanmış ve geliştirilmiştir.

4.1 Genel Bakış

Geliştirilen uygulama iki farklı çalışma modunu (koşum kayıt/tekrar oynatım) da destekleyecek şekilde tasarlanmıştır.

Koşum kayıt (Execution Logging) modu, koşum sırasında dahil olunan federasyonda yayınlanan etkileşim mesajları ile nesne güncellemelerini kayıt edebilecek şekilde tasarlanmıştır. Koşum kayıt modunda, federenin SOM dosyasında abone isteği ile tanımlanmış etkileşim ve nesne sınıflarına ait mesajlar koşum zamanında kaydedilir. FOM’da tanımlı bir etkileşim sınıfına ait tüm mesajların benzetim koşumu sırasında koşum kayıt birimi tarafından kayıt edilmesi isteniyorsa, bu etkileşim sınıfı koşum kayıt federesi SOM dosyasında da abone (subscribe) isteğiyle tanımlanır. Bu sayede, istenilen her türlü etkileşim ve nesne güncellemesinin kayıt edilebilmesi amaçlanmış, esnek bir yapı tasarlanmıştır.

4.2 Yazılım Teknolojileri Ve Yardımcı Hla Araçları

Yapılan çalışmada uygulamanın geliştirilmesi ve tasarımında HLA standartlarına uyumluluk göz önünde bulundurulmuş, bu doğrultuda kullanılan yazılım araçları ve teknolojileri aşağıda anlatılmıştır.

4.2.1 RPR FOM

HLA1516 standardı, federelerin veri alış verişinde FOM (Federation Object Model) kullanımını dikte etmesine karşın, hangi FOM'un kullanılacağı ile ilgili bir kısıt getirmemiştir. Federelerin birbiriyle haberleşmesi ve birlikte çalışabilirliği sağlamak için birtakım referans FOM'lar geliştirilmiştir. Geliştirilmiş referans FOM'lar birçok benzetim aracı tarafından kullanılmaktadır. Referans FOM'lar, üzerinde değişiklik yapmadan kullanılabildiği gibi, farklı benzetim konseptleri için genişletilebilir formatta tasarlanmışlardır.

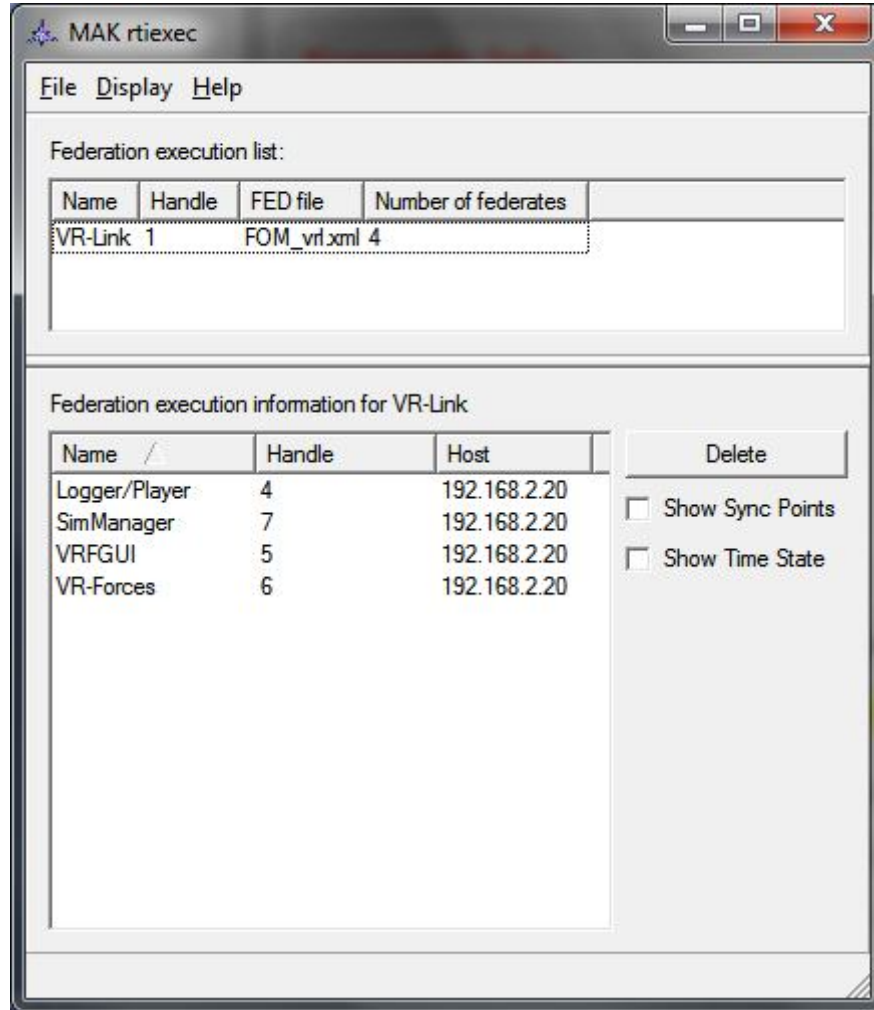
Çalışmada federasyon nesne modeli (FOM) için RPR FOM v1.0 referans alınarak geliştirme yapılmıştır. RPR FOM (SISO Real-time Platform-level Reference FOM), gerçek zamanlı ve platform seviyesi benzetimleri için uygun HLA sınıfları, parametre ve özniteliklerinin tanımlandığı, SISO tarafından geliştirilmiş referans nesne modelidir. RPR FOM v1.0, 1999 yılında "SISO-STD-001.1-1999" adıyla SISO standardı olarak kabul edilmiştir. Çalışmada, RPR-FOM v1.0 federasyon nesne modelinde etkileşim ve nesne sınıfı tablolarına gerekli görüldüğü noktalarda etkileşim ve nesne sınıfları eklenerek genişletilmiştir [10].

4.2.2 MAK RTI

HLA 1.3 ve IEEE 1516 standartları ile uyumlu yazılım geliştirme kütüphanesi ve federasyon koşumundaki federelerin birbirleri ile veri iletişimi için gerekli RTI sunucu yazılımından oluşur. Yazılım geliştirme kütüphanesi C++ programlama dili ile HLA standartlarına uyumlu federe yazılımı geliştirmek için gerekli yapılardan oluşur. MAK RTI 1516 sunucu yazılımı IEEE 1516 arayüz tanımlamalarında belirtilen servislerin tümünü gerçeklemiştir. MAK RTI, "MAK Technologies" firması tarafından geliştirilen DMSO onaylı ticari bir üründür. Windows, Linux ve Sun Solaris platformlarını desteklemesi, geliştirilen federe yazılımlarının farklı işletim platformlarında çalışmasına imkan tanır.

Yapılan çalışmanın HLA standartlarına uyumlu olması için, federe yazılımı MAK RTI yazılım geliştirme kütüphanesi kullanılarak Visual Studio 2005 entegre yazılım geliştirme ortamında C++ programlama dili ile gerçekleştirilmiştir. Koşum Zamanı Altyapısı için MAK RTI sunucu yazılımı kullanılmıştır [11].

Şekil 4.1’de MAK RTI sunucu yazılımı ve “VR-Link” isimli federasyon koşumu ve federasyona katılan dört federe görülmektedir.



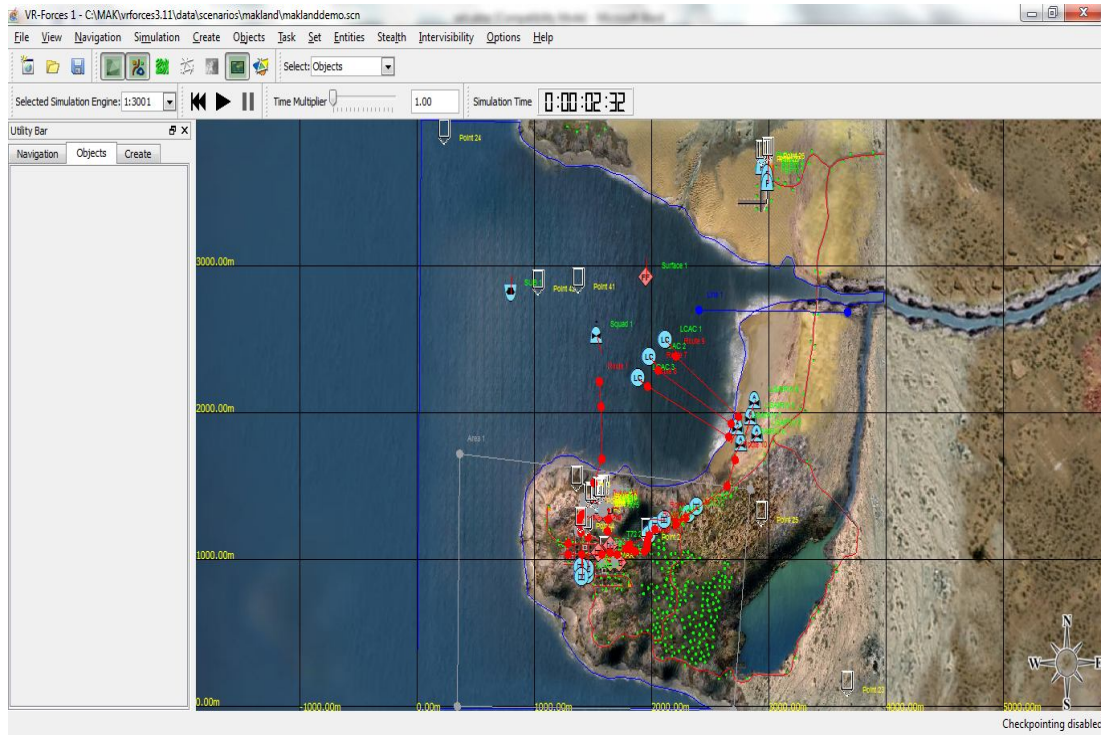
Şekil 4.1 : MAK RTI sunucu yazılımı

Geliştirilen uygulamanın koşum kayıt ve tekrar oynatım testleri için VR-Forces yazılımı yardımcı araç olarak kullanılmıştır.

4.2.3 VR-Forces

Savaş senaryoları tasarlamak ve çalıştırmak için, “MAK Technologies” firması tarafından geliştirilmiş güçlü ve esnek bir simülasyon aracıdır. Taktik eğitim, tehdit oluşturma, davranış modeli sınama ortamı, bilgisayar destekli oluşturulan kuvvetler

(Computer Generated Forces - CGF) vb. gerekli simülasyon özelliklerine sahiptir. HLA 1516 standardı ile uyumludur, RPR FOM 1.0 federasyon nesne modelini kullanır. Verilerin harita üzerinde görselleştirilmesi, senaryo tabanlı kullanım arayüzü imkanı sunması, koşum kayıt testleri için benzetim ortamına test amaçlı verinin yayınlanması ve tekrar oynatım sırasında benzetim ortamına yayınlanan verinin görselleştirilmesi için kullanılmıştır [12].



Şekil 4.2 : VR-Forces uygulaması

Yapılan çalışmada, grafiksel kullanıcı ara yüz (graphical user interface - GUI) tasarımları için Qt C++ yazılım geliştirme kütüphanesi ve beraberindeki Qt Designer yazılımı kullanılmıştır. Ayrıca geliştirilen uygulamanın veritabanı aktarım modülünün tasarımı ve geliştirilmesinde Qt kütüphanesinde yer alan Sql modülü kullanılmıştır.

4.2.4 QT uygulama geliştirme kütüphanesi

Trolltech firması tarafından geliştirilmiş C++ temelli uygulama geliştirme kütüphanesi ve beraberindeki diğer yazılım araçlarını kapsar. 2008 yılında Nokia

tarafından satın alınmasıyla, LGPL ve GPL lisans sözleşmesi ile ücretsiz olarak kullanıma açılmıştır. Farklı uygulama platformlarını (Embedded Linux, Mac OS X, Windows, Linux/X11, Windows CE/Mobile, Symbian, Maemo) desteklemesi ve C++ temelli olması çalışmada kullanılmasında tercih nedenidir.

Çalışmada, benzetim ortamında kaydı yapılan bir senaryonun verilerinin ilişkisel veri tabanında depolanması için Microsoft SQL Server 2005 veri tabanı yönetim sistemi kullanılmıştır. Geliştirilen uygulamanın, MySql Server gibi farklı ilişkisel veri tabanı sunucu yazılımları ile de uyumlu çalışabilmesi amaçlanmış, yazılım mimarisi buna uygun olarak tasarlanmıştır.

4.3 Benzetim Yönetim Federesi (Simulation Manager Federate)

Çalışmada, federasyona katılan federelerin yaşam döngüsü için gerekli koşum başlangıç, koşum durdurma, koşum bitirme, senaryo bilgisi gibi yönetsel etkileşim mesajlarını merkezi bir noktadan federasyona yayınlayan benzetim yönetim federesi tasarlanmış ve gerçekleştirilmiştir.

Benzetim yönetim federesi, federasyona katılan federelerin yaşam döngüsünü sağlamak için tasarlanmış özel bir federe'dir. Benzetim yönetim federesi, yayınladığı etkileşim mesajları ile federelerin benzer yaşam döngülerinde senkron çalışmalarını sağlar. Federasyon kontrolü ve senkronizasyonu, federelerin koşuma başlaması ve durdurulması, koşum hakkında bilgilendirilmesi gibi ortak bir anlayışın sağlanmasını temel alır.

4.3.1 Federe yaşam döngüsü etkileşim sınıfları

Benzetim yönetim federesi tarafından federe yaşam döngüsü için yayınlanan etkileşim sınıfları aşağıda anlatılmıştır:

Benzetim Senaryosu Etkileşim Mesajı (LoadScenario interaction) : Federasyonun koşumu ile ilgili senaryo bilgisi içeren etkileşim mesajıdır. Federasyona katılan federeler, koşum başlamadan önce koşumu yapılacak senaryo hakkında bu etkileşim mesajı ile bilgilendirilir. RPR-FOM v1.0 federasyon nesne modeli etkileşim sınıfı tablosunda "LoadScenario" adıyla tanımlanmıştır. Etkileşim sınıfı tablosundaki tanımlama bilgisi Çizelge 4.1'de verilmiştir.

Çizelge 4.1 : Benzetim senaryosu etkileşim mesajı.

```
<interactionClass name="LoadScenario" sharing="PublishSubscribe"
dimensions="NA" transportation="HLA Reliable" order="Receive"
semantics="senaryo başlangıç bilgileri">
  <parameter name="ScenarioFileName" dataType="HLAASCIIString"/>
  <parameter name="IPAddress" dataType="HLAASCIIString"/>
</interactionClass>
```

Koşum Başlangıç Etkileşim Mesajı (StartResume interaction) : Federasyon koşumunun başladığını bildiren etkileşim mesajıdır. Etkileşim sınıfı tablosunda “StartResume” adıyla tanımlıdır. Federelere, RTI tarafından bu etkileşim mesajının iletilmesi ile birlikte federeler koşum döngüsüne başlar. Bu etkileşim mesajı ile federasyonun ortak bir senaryo üzerinden koşumu gerçekleştirilerek ortak çalışma ve senkronizasyon sağlanır.

“StartResume” etkileşim sınıfının etkileşim sınıf tablosundaki tanımlama bilgisi Çizelge 4.2’de verilmiştir.

Çizelge 4.2 : Koşum başlangıç etkileşim sınıfı.

```
<interactionClass name="StartResume" nameNotes="13"
sharing="PublishSubscribe" dimensions="NA"
transportation="HLA BestEffort" order="Receive" semantics="A
Simulation Management (SIMAN) interaction, sent from a Simulation
Manager federate to either a) start simulating one or more
entities or b) resume simulation of one or more entities after a
pause.">
  <parameter name="SimulationTime" dataType="ClockTimeStruct"
semantics="The simulation time that the entity or entities should
use when they start/resume."/>
  <parameter name="RequestIdentifier" nameNotes="19"
dataType="UnsignedLong1" semantics="The Request ID is a
monotonically increasing integer identifier inserted by the
Simulation Manager into all Simulation management interactions.
It is used as a unique identifier to identify the latest in a
series of competing requests and identifying acknowledgements."/>
  <parameter name="ReceivingEntity"
dataType="EntityIdentifierStruct" semantics="The DIS entity ID of
the entity or application which is the intended recipient of the
interaction."/>
  <parameter name="RealWorldTime" dataType="ClockTimeStruct"
semantics="The real world time that the entity or entities should
start/resumed."/>
  <parameter name="OriginatingEntity"
dataType="EntityIdentifierStruct" semantics="The DIS entity ID of
the entity or application sending the interaction."/>
</interactionClass>
```

Koşum Beklet/Durdur Etkileşim Mesajı (StopFreeze interaction) : Federasyon koşumunun beklemeye alınacağını veya durdurulacağını bildiren etkileşim mesajıdır.

Etkileşim sınıfı tablosunda “StopFreeze” adıyla tanımlıdır. Federeler, StartResume etkileşim mesajı ile koşuma başladıktan sonra StopFreeze etkileşim mesajına kadar kendi koşumlarına devam eder. Yönetim federesi koşumun durdurulacağı/bekletileceği bilgisini bu etkileşim mesajı ile tüm federasyona bildirir. Her bir federe bu etkileşim mesajı ile birlikte kendi koşumunu bekletir/durdurur.

“StopFreeze” etkileşim sınıfının FOM etkileşim sınıfı tablosundaki tanımlama bilgisi bilgisi Çizelge 4.3’de verilmiştir.

Çizelge 4.3 : Koşum beklet/durdur etkileşim mesajı.

```
<interactionClass name="StopFreeze" nameNotes="13"
sharing="PublishSubscribe" dimensions="NA"
transportation="HLAbestEffort" order="Receive" semantics="A
Simulation Management (SIMAN) interaction, sent from a Simulation
Manager federate to request that one or more entities either a)
pause their simulation or b) stop their simulation.">
  <parameter name="UpdateAttributes" dataType="OMT13boolean"
semantics="Whether the entity or entities being stopped/frozen
should continue to update attributes when stopped/frozen."/>
  <parameter name="RequestIdentifier" nameNotes="19"
dataType="UnsignedLong1" semantics="The Request ID is a
monotonically increasing integer identifier inserted by the
Simulation Manager into all Simulation management interactions.
It is used as a unique identifier to identify the latest in a
series of competing requests and identifying acknowledgements."/>
  <parameter name="ReceivingEntity"
dataType="EntityIdentifierStruct" semantics="The DIS entity ID of
the entity or application which is the intended recipient of the
interaction."/>
  <parameter name="RunInternalSimulationClock"
dataType="OMT13boolean" semantics="Whether the entity or entities
being stopped/frozen should continue to run their internal
simulation clock when stopped/frozen."/>
  <parameter name="RealWorldTime" dataType="ClockTimeStruct"
semantics="The real world time that the entity or entities should
stop/freeze."/>
  <parameter name="OriginatingEntity"
dataType="EntityIdentifierStruct" semantics="The DIS entity ID of
the entity or application sending the interaction."/>
  <parameter name="ReflectValues" dataType="OMT13boolean"
semantics="Whether the entity or entities being stopped/frozen
should continue to reflect values when stopped/frozen."/>
  <parameter name="Reason" dataType="StopFreezeReasonEnum8"
semantics="The reason for the stop or freeze."/>
</interactionClass>
```

Benzetim Senaryosu Sonlandırma Etkileşim Mesajı (CloseScenario interaction) :
Benzetim senaryosu etkileşim mesajı (LoadScenario) ile başlatılan senaryonun sonlandırılacağı anlamına gelen etkileşim mesajıdır. Etkileşim sınıfı tablosunda “CloseScenario” adıyla tanımlanmıştır. Benzetim yönetim federesi tarafından

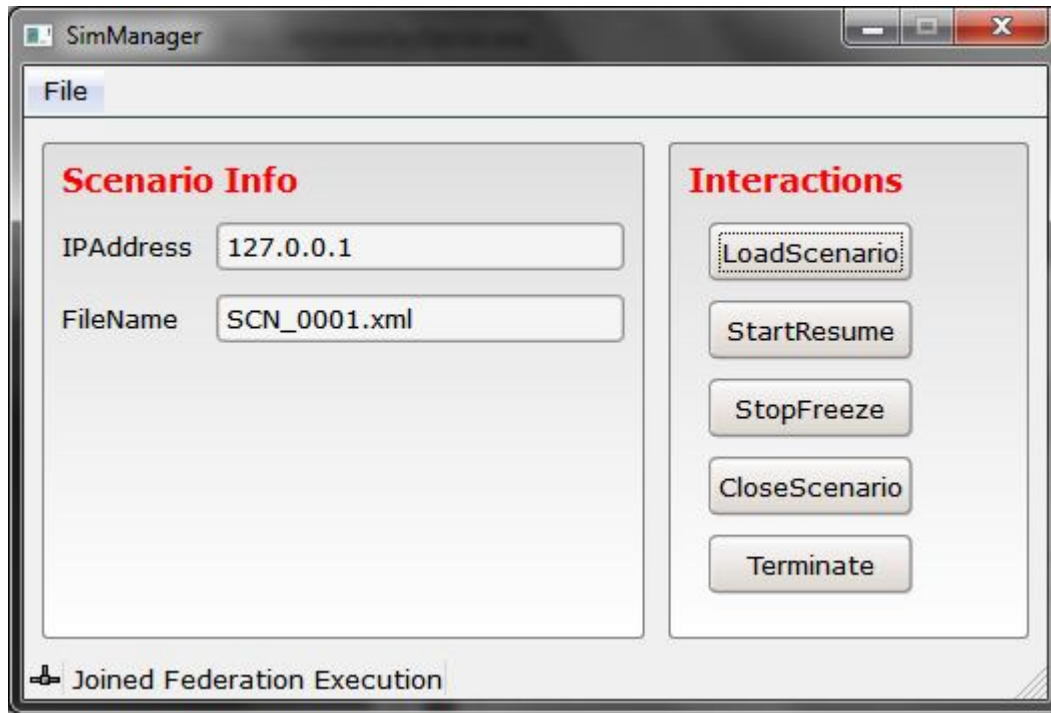
gönderilen “CloseScenario” etkileşim mesajının federeler tarafından alınması ile, her federe federasyona kayıt ettirdiği (register object instance) nesne sınıfı örneklerini silerek senaryo sonlandırma işlemini yapar. “CloseScenario” etkileşim sınıfının FOM etkileşim sınıfı tablosundaki tanımlama bilgisi Çizelge 4.4’de verilmiştir.

Çizelge 4.4 : Senaryo sonlandır etkileşim sınıfı.

```
<interactionClass name="CloseScenario" sharing="PublishSubscribe"
dimensions="NA" transportation="HLAreliable" order="Receive"/>
```

4.3.2 Benzetim yönetim federesi çalışma esası

Geliştirilen benzetim yönetim federesi uygulama kullanıcı arayüzü Şekil 4.3’de gösterilmiştir. Federasyona dahil olduktan sonra koşum yönetimi ile ilgili etkileşim mesajları arayüzden bunun yerine ilgili düğmeler yardımıyla gönderilebilmektedir.



Şekil 4.3 : Benzetim yönetim federesi kullanıcı arayüzü

Federasyona ilk önce LoadScenario etkileşim mesajı gönderilerek federeler koşumu yapılacak senaryo hakkında bilgilendirilir. LoadScenario etkileşimi FileName ve IPAddress parametrelerinden oluşur. FileName senaryo dosyasının adını, IPAddress ise senaryo dosyasının bulunduğu dosya sunucusunun adresini temsil eder.

Federasyondaki diğer federeler LoadScenario etkileşim mesajını aldığı anda ilgili adresteki dosya sunucusundan senaryo dosyasına ulaşarak, XML formatındaki senaryo dosyasından koşumu yapılacak senaryo hakkında bilgi edinir. Örnek senaryo dosyası içeriği Çizelge 4.5’de gösterilmiştir. Senaryo bilgisinin dosya sunucusunda XML formatlı dosyada tutulması esnekliği ve genişletilebilirliği sağlamıştır.

Çizelge 4.5 : Örnek senaryo dosyası içeriği.

```
<Scenario>
  <ScenarioInfo name="scenario information">
    <ParameterList>
      <Parameter name="date" value="2010-07-24"/>
      <Parameter name="time" value="23:02:00"/>
    </ParameterList>
  </ScenarioInfo>
  <ScenarioEntities>
    <Entity name="MapOrigin">
      <ParameterList>
        <Parameter name="latitude" value="41 30 0.00N"/>
        <Parameter name="longitude" value="26 16 12.00E"/>
      </ParameterList>
    </Entity>
    <Entity name="EnvironmentConditions">
      <ParameterList>
        <Parameter name="seaState" value="0"/>
        <Parameter name="temperature" value="5.048"/>
        <Parameter name="temperatureModel" value="1"/>
        <Parameter name="windDirection" value="0"/>
        <Parameter name="windMagnitude" value="0.10"/>
      </ParameterList>
    </Entity>
  </ScenarioEntities>
</Scenario>
```

Senaryo bilgisi etkileşim mesajı gönderildikten sonra StartResume etkileşim mesajı gönderilerek koşum başlatılır. Her federe kendi koşum döngüsüne başlayarak, içsel hesaplamalarını yapar, federasyona etkileşim mesajları ve nesne güncellemeleri (register,update, delete) gönderirler.

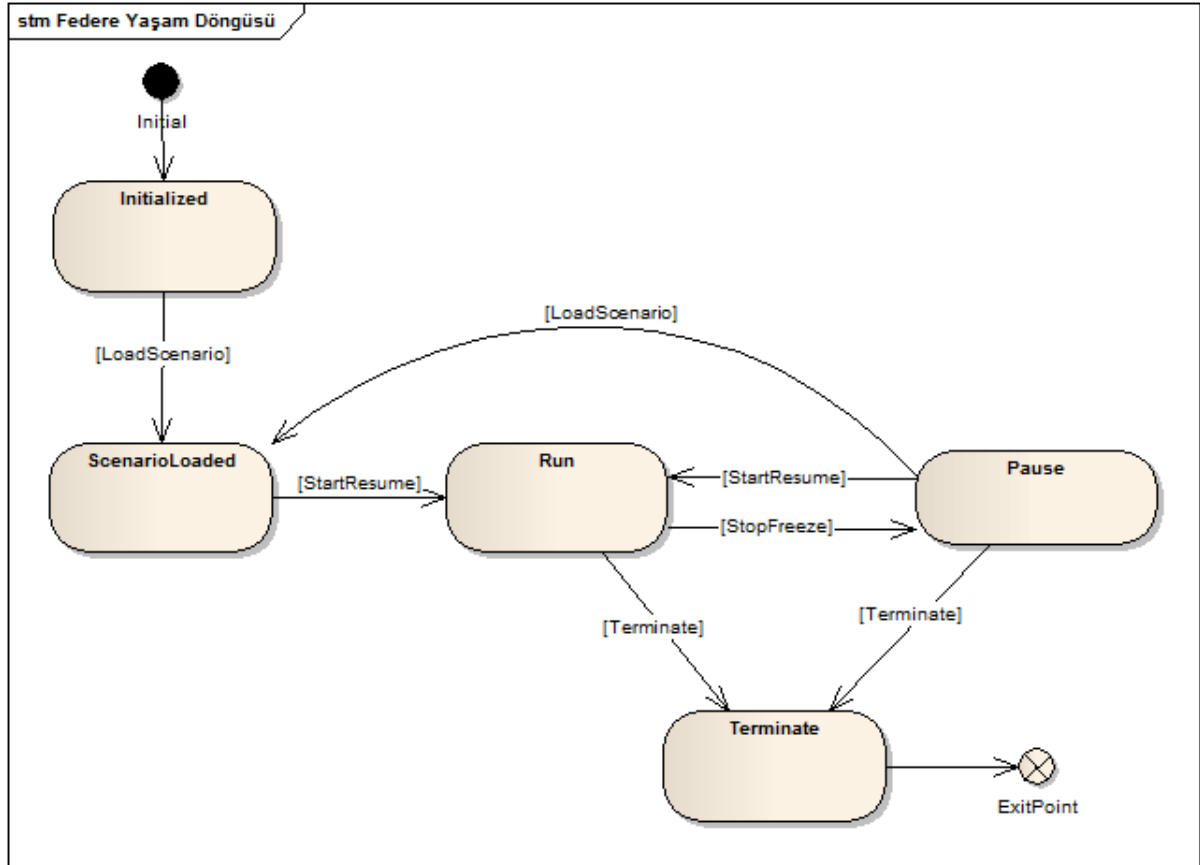
Koşuma ara vermek için yönetim federesi tarafından StopFreeze mesajı gönderilir. Koşumun kaldığı yerden devam etmesi için tekrar StartResume etkileşimi gönderilebilir.

Koşum senaryosunun sonlandırılması için CloseScenario etkileşimi gönderilir. Federeler bu etkileşimi aldıklarında, koşum sırasında RTI’ya kaydettirdikleri nesneleri silerek bekleme konumuna geçerler.

Federelerin federasyondan ayrılmaları (resign from federation) için Terminate etkileşimi gönderilir.

4.3.3 Federe yaşam döngüsü

Benzetim yönetim federesi tarafından gönderilen koşum yönetimi ile ilgili etkileşim mesajları federelerin yaşam döngüsünü sağlar. Federasyondaki diğer federelerin yaşam döngüsü durum diyagramı Şekil 4.4’de gösterilmiştir.



Şekil 4.4 : Federe yaşam döngüsü

4.4 Benzetim Nesne Modeli Yapısı

Federeler için gerekli benzetim nesne modeli (SOM) genişletilebilir işaretleme dili (XML - eXtensible Markup Language) kullanılarak tasarlanmıştır. Tasarlanan benzetim nesne modeli, hem yönetim federesi hem de koşum kayıt/tekrar oynatım federesi için uygulanmıştır. İçerisinde etkileşim ve nesne sınıfı tabloları yer alır. Her federe abone/yayın isteğine göre etkileşim ve nesne sınıflarını tanımlar.

4.4.1 Etkileşim sınıfı tanımlama

Geliştirilen benzetim yönetim federesinin SOM’unda tanımlı LoadScenario etkileşim sınıfı Çizelge 4.6’da gösterilmiştir.

Çizelge 4.6 : Benzetim nesne modelinde etkileşim sınıfı tanımlama.

```
<Interactions>
  <Interaction FedName="SimManager"
    FullName="HLAinteractionRoot.LoadScenario"
    ShortName="LoadScenario" PS="3">
    <Parameter ParamName="ScenarioFileName"
      ParamType="composite"/>
    <Parameter ParamName="IPAddress" ParamType="composite"/>
  </Interaction>
</Interactions>
```

Interactions : Etkileşim sınıfları bu düğümün (node) altında tanımlanır.

Interaction: Etkileşim sınıfı bu düğüm ile tanımlanır. FedName, FullName, ShortName ve PS özniteliklerine sahiptir.

FedName : Federe adını temsil eder. “SimManager” yönetim federesinin federasyondaki adını temsil etmektedir.

FullName : Etkileşim sınıfının federasyon nesne modelindeki (FOM) kalıtım ağacındaki tam yolunu temsil eder. LoadScenario etkileşim sınıfının HLAinteractionRoot sınıfından türediği görülmektedir. RPR FOM federasyon nesne modeli etkileşim sınıf tablosundaki en üst seviye ata sınıf HLAinteractionRoot sınıfıdır [10].

ShortName : Etkileşim sınıfının kısa adını temsil eder.

PS : Publish/Subscribe anlamına gelen abone/yayın isteğini temsil eder. 1,2 veya 3 değerlerinden birisini alabilir.

- PS=“1” : Federe ilgili etkileşim sınıfı mesajlarını yayınlatabilir (publish) .
- PS=“2” : Federe ilgili etkileşim sınıfına abone (subscribe) olarak bu etkileşim sınıfı mesajlarını alabilir.
- PS=“3” : Federe abone ve yayın işlevlerinin ikisini birden yerine getirebilir.

Parameter : Etkileşim sınıfı parametrelerini temsil eder. Parametre adı ve veri tipi özniteliklerinden oluşur. Etkileşim sınıfları sıfır veya daha fazla parametreye sahip olabilir.

4.4.2 Nesne sınıfı tanımlama

Federenin abone/yayın isteğiyle ilgili her bir nesne sınıfı nesne sınıfı tanımlama tablosu içerisinde yer alır. SurfaceVessel nesne sınıfının benzetim nesne modelinde içerisindeki tanımlaması Çizelge 4.7’de gösterilmiştir. SurfaceVessel, RPR FOM v1.0’da su üstü platformları temsil eden nesne sınıfı olarak tanımlıdır [10].

Çizelge 4.7 : Benzetim nesne modelinde nesne sınıfı tanımlama.

```
<Objects>
  <Object FedName="SimManager"
  FullName="HLAobjectRoot.BaseEntity.PhysicalEntity.Platform.Surface
  Vessel" ShortName="SurfaceVessel" PS="3">
    <Attribute AttribName="EntityType" AttribType="composite"/>
    <Attribute AttribName="EntityIdentifier"
      AttribType="composite"/>
    <Attribute AttribName="AccelerationVector"
      AttribType="composite"/>
    <Attribute AttribName="VelocityVector"
      AttribType="composite"/>
    <Attribute AttribName="WorldLocation" AttribType="composite"/>
  </Object>
</Objects>
```

Objects : Nesne sınıfları bu düğüm altında tanımlanır.

Object : Nesne sınıfı bu düğüm ile tanımlanır. FedName, FullName, ShortName ve PS özniteliklerine sahiptir.

FedName : Federe adını temsil eder.

FullName : Nesne sınıfının federasyon nesne modelindeki (FOM) kalıtım ağacındaki tam yolunu temsil eder. SurfaceVessel nesne sınıfının Platform sınıfından türediği görülmektedir. HLAobjectRoot sınıfı nesne sınıf tablosundaki en üst seviye ata sınıfı temsil eder [10].

ShortName : Nesne sınıfının kısa adını temsil eder.

PS : Publish/Subscribe anlamına gelen abone/yayın isteğini temsil eder. PS’nin aldığı farklı değerlere göre federenin abone/yayın isteği etkileşim sınıfı tanımlamasında anlatıldığı şekliyledir.

Attribute : Nesne sınıfı özniteliklerini temsil eder. Öznitelik tanımlama ad ve veri tipi öznitelikleri ile yapılır. Nesne sınıfları sıfır veya daha fazla özniteliğe sahip olabilir.

Geliştirilen benzetim yönetim federesi SOM'unun tüm içeriği çizelge 4.8'de gösterilmiştir.

Çizelge 4.8 : Benzetim yönetim federesi SOM içeriği.

```
<fedRootElem >
  <Interactions>
    <Interaction FedName="SimManager"
      FullName="HLAinteractionRoot.LoadScenario"
      ShortName="LoadScenario" PS="3">
      <Parameter ParamName="ScenarioFileName" ParamType="composite"/>
      <Parameter ParamName="IPAddress" ParamType="composite"/>
    </Interaction>
    <Interaction FedName="SimManager"
      FullName="HLAinteractionRoot.CloseScenario"
      ShortName="CloseScenario" PS="3" />
    <Interaction FedName="SimManager"
      FullName="HLAinteractionRoot.StartResume"
      ShortName="StartResume" PS="3">
      <Parameter ParamName="SimulationTime" ParamType="composite"/>
      <Parameter ParamName="RequestIdentifier" ParamType="composite"/>
      <Parameter ParamName="ReceivingEntity" ParamType="composite"/>
      <Parameter ParamName="RealWorldTime" ParamType="composite"/>
      <Parameter ParamName="OriginatingEntity" ParamType="composite"/>
    </Interaction>
    <Interaction FedName="SimManager"
      FullName="HLAinteractionRoot.StopFreeze"
      ShortName="StopFreeze" PS="3">
      <Parameter ParamName="UpdateAttributes" ParamType="composite"/>
      <Parameter ParamName="RequestIdentifier" ParamType="composite"/>
      <Parameter ParamName="ReceivingEntity" ParamType="composite"/>
      <Parameter ParamName="RunInternalSimulationClock"
        ParamType="composite"/>
      <Parameter ParamName="RealWorldTime" ParamType="composite"/>
      <Parameter ParamName="OriginatingEntity" ParamType="composite"/>
      <Parameter ParamName="ReflectValues" ParamType="composite"/>
      <Parameter ParamName="Reason" ParamType="composite"/>
    </Interaction>
    <Interaction FedName="SimManager"
      FullName="HLAinteractionRoot.Terminate"
      ShortName="Terminate" PS="3" />
  </Interactions>
  <Objects>
  </Objects>
</fedRootElem>
```

4.5 Federe Mimarisi ve Tasarımı

Geliştirilen federe uygulaması RTI ile olan haberleşmesini RTI Elçisi (RTIambassador) ve Federe Elçisi (FederateAmbassador) sınıfları ile sağlar.

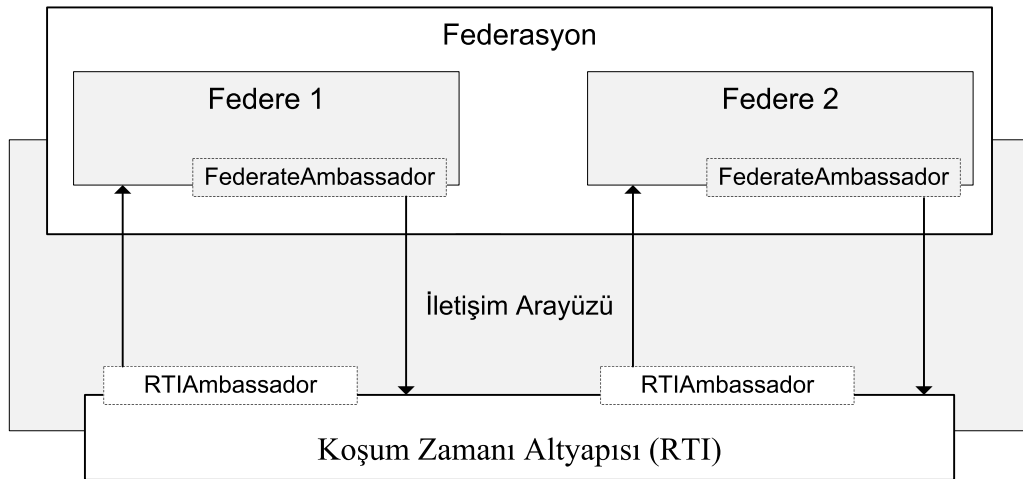
4.5.1 RTI elçisi (RTIambassador) ve Federe elçisi (FederateAmbassador)

RTI Elçisi sınıfı, federenin RTI üzerinde çağırabileceği temel servisleri gerçekleştirmek üzere sağlanan sınıftır. Sınıfın federe tarafından RTI üzerinde yapılacak çağrılar amacıyla sağladığı temel fonksiyonlar ve işlevleri aşağıdaki verilmiştir.

- Federe ve federasyon yönetimi fonksiyonları
 - createFederationExecution
 - joinFederationExecution
 - destroyFederationExecution
 - resignFederationExecution
- RTI'a abone/yayın isteği bildiren fonksiyonlar.
 - publishObjectClassAttributes
 - subscribeObjectClassAttributes
 - publishInteractionClass
 - subscribeInteractionClass
- RTI'a etkileşim mesajı ve nesne güncellemesi ileten fonksiyonlar
 - sendInteraction
 - registerObjectInstance
 - updateAttributeValues
 - deleteObjectInstance

Federe Elçisi sınıfı, RTI tarafından federe üzerinde çağrılacak olan ve RTI üzerinden sisteme dahil olacak her bileşenin gerçekleştirmesinin zorunlu olduğu fonksiyonların sağlandığı sınıftır. Bu metotlar, RTI tarafından sağlanmış soyut (abstract) bir arayüz sınıfı içerisinde soyut fonksiyonlar (pure virtual) olarak tanımlanmış ve simülasyona katılacak her federe tarafından gerçekleştirilmesi gereken fonksiyonlardır. Sınıfın, RTI tarafından federe üzerinde yapılacak çağrılar amacıyla sağladığı temel metotlar ve işlevleri aşağıdaki verilmiştir.

- discoverObjectInstance: nesne oluşturulma mesajı Geri çağrım (callback) fonksiyonu
- receiveInteraction: etkileşim mesajı callback fonksiyonu
- removeObjectInstance: nesne silinme mesajı callback fonksiyonu
- reflectAttributeValues: nesne güncelleme mesajı callback fonksiyonu



Şekil 4.5 : Elçi sınıfları ile federe-federasyon iletişim arayüzü

Çizelge 4.9 : Federasyon ve federe yönetim fonksiyonları.

Fonksiyon Adı	İşlevi
<pre>void createFederationExecution (std::wstring const& federationExecutionName, std::wstring const& fullPathNameToTheFDDfile);</pre>	Federasyon oluşturma fonksiyonudur.
<pre>FederateHandle joinFederationExecution (std::wstring const& federateType, std::wstring const& federationExecutionName, FederateAmbassador& federateAmbassador);</pre>	Federenin, federasyona katılım fonksiyonudur.
<pre>void destroyFederationExecution (std::wstring const & federationExecutionName);</pre>	Federasyonu yok etme fonksiyonudur.
<pre>void resignFederationExecution (ResignAction resignAction);</pre>	Federenin, federasyondan ayrılma fonksiyonudur.

Çizelge 4.10 : Etkileşim ve nesne mesajı ileten fonksiyonlar.

Fonksiyon Adı	İşlevi
<pre>void sendInteraction (InteractionClassHandle theInteraction, ParameterHandleValueMap const& VariableLengthData const& theUserSuppliedTag);</pre>	Etkileşim mesajı gönderme fonksiyonudur.
<pre>void updateAttributeValues (ObjectInstanceHandle theObject, AttributeHandleValueMap const& theAttributeValues, VariableLengthData const& theUserSuppliedTag);</pre>	Nesne güncellemesi gönderme fonksiyonudur.
<pre>ObjectInstanceHandle registerObjectInstance (ObjectClassHandle theClass, std::wstring const& theObjectInstanceName);</pre>	Nesne sınıfından oluşturulan bir örneği RTI'a kaydettirme fonksiyonudur.
<pre>void deleteObjectInstance (ObjectInstanceHandle theObject, VariableLengthData const& theUserSuppliedTag);</pre>	Nesne sınıfından örneği silme işlemini RTI'a bildirim fonksiyonudur.

Çizelge 4.11 : Abone/Yayın isteği bildiren fonksiyonlar.

Fonksiyon Adı	İşlevi
void publishObjectClassAttributes (ObjectClassHandle theClass, AttributeHandleSet const& attributeList);	Federenin belirtilen nesne sınıfına ve özniteliklerine yayınlama (publish) isteği yapılır.
void subscribeObjectClassAttributes (ObjectClassHandle theClass, AttributeHandleSet const& attributeList, bool active = true);	Federenin belirtilen nesne sınıfına ve özniteliklerine abone (subscribe) isteği yapılır.
void publishInteractionClass (InteractionClassHandle theInteraction);	Federenin belirtilen etkileşim sınıfına yayınlama (publish) isteği yapılır.
void subscribeInteractionClass (InteractionClassHandle theClass, bool active = true);	Federenin belirtilen etkileşim sınıfına abone (subscribe) isteği yapılır.

4.5.2 Uygulamanın federasyona katılımı

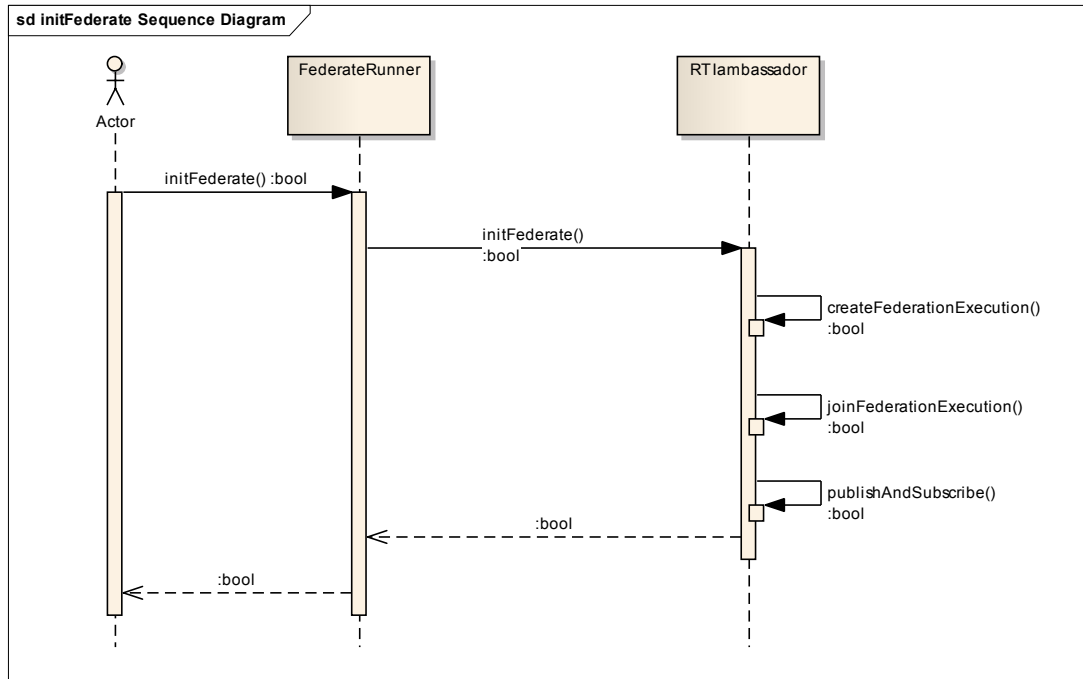
Federenin RTI ile olan etkileşimi RTI Elçisi sınıfı (RTI Ambassador) ve Federe Elçisi sınıfı (Federate Ambassador) örnekleri ile sağlanır. RTI kütüphanesi RTI Elçisi sınıf gerçeklemesini geliştiricilerin kullanımına açmıştır. Federe ilk olarak RTI Elçisi sınıfından bir örnek (instance) oluşturması gerekir. RTI üzerinde servis çağırımları yapmak için RTI Elçisi sınıf örneği kullanılır.

Federe uygulaması, Federe Elçisi soyut sınıfının gerçekleştirilmesinden sorumludur. MÄK RTI yazılım geliştirme kütüphanesinde Federe Elçisi sınıfı FederateAmbassador adıyla tanımlıdır. NullFederateAmbassador, herhangi bir işlevi olmayan boş gövdeye sahip somut sınıf (concrete class) tanımlamasıdır. Çalışmada geliştirilen federe uygulamasında FederateRunner sınıfı NullFederateAmbassador sınıfından türetilmiş, Federe Elçisi sınıfının gerekli üye fonksiyonları, özellikle discoverObjectInstance, removeObjectInstance, reflectAttributeValues ve

receiveInteraction gereklenmiřtir. Bunlar RTI tarafından federeye iletilecek olan geri aęrım metotlarıdır.

Geliřtirilen federe uygulamasının federasyona katılımı 4 temel adımda gerekleřtirilir. řekil 4.6’da akıř diyagramı ile gsterilmiřtir.

1. Federasyon kořumu (federation execution) oluřturmak iin fonksiyon aęrımı yapılır. (createFederationExecution). Federasyon nceden bařka bir federe tarafından oluřturulmuřsa tekrar oluřturulmaz. Federasyon ilk katılan federe tarafından oluřturulur.
2. Federasyon kořumuna katılım iin fonksiyon aęrımı yapılır. (joinFederationExecution)
3. Nesne sınıfı znitelikleri abone/yayın (ps) iin fonksiyon aęrımı yapılır. (publishObjectClassAttributes, subscribeObjectClassAttributes)
4. Etkileřim sınıfı abone/yayın (ps) iřlemi iin fonksiyon aęrımı yapılır. (publishInteractionClass, subscribeInteractionClass)



řekil 4.6 : Uygulamanın federasyona katılım akıř diyagramı

4.5.3 Federe SOM yapısı

Uygulamanın koşum kayıt modunda çalışması sırasında RTI'dan nesne ve etkileşim mesajları alabilmesi için kaydedilecek nesne ve etkileşim sınıflarına abone (subscribe) olması gerekir. Kaydedilmesi istenen her türlü etkileşim sınıfı etkileşim sınıf tablosunda, nesne sınıfları ise nesne sınıf tablosunda tanımlanmalıdır.

Benzetim yönetim federesi tarafından gönderilen etkileşim mesajlarının kaydedilmesi amaçlanmıyor olsa bile, federe uygulamasının koşum kayıt işlemi için gerekli, koşum başlatma / durdurma ve koşum senaryo bilgisi etkileşim sınıflarına abone olunması zorunludur.

Uygulama tekrar oynatma modunda, kayıt dosyasında bulunan nesne ve etkileşim kayıtlarının RTI'a gönderilebilmesi için ilgili nesne ve etkileşim sınıflarına yayın (publish) isteğinin RTI'a bildirilmesi gerekir.

Geliştirilen uygulama (federe) kayıt ve tekrar oynatım modlarının her ikisini de destekleyecek şekilde tasarlanmıştır. Etkileşim ve nesne sınıflarına abone/yayın (PS) değeri "3" olarak tanımlanmıştır. PS = 3 olarak tanımlanması RTI'dan o sınıfa ait mesajları alabilmesi ve gönderebilmesi anlamına gelir. (Bölüm xxx'de bahsedilmiştir)

"MunitionDetonation" etkileşim sınıfının SOM'da tanımlanması Çizelge 4.12'de gösterilmiştir. Böylece koşum kaydı sırasında federasyondaki diğer federeler tarafından yayınlanan bu etkileşim sınıfı mesajları 3 parametresi ile birlikte kaydedilebilecektir.

Çizelge 4.12 : Uygulama SOM dosyasında etkileşim sınıfı tanımlama.

```
<Interaction FedName="Logger_Player"
  FullName="HLAinteractionRoot.MunitionDetonation"
  ShortName="MunitionDetonation" PS="3">
  <Parameter ParamName="DetonationLocation"
    ParamType="composite"/>
  <Parameter ParamName="TargetObjectIdentifier"
    ParamType="composite"/>
  <Parameter ParamName="FiringObjectIdentifier"
    ParamType="composite"/>
</Interaction>
```

"SurfaceVessel" nesne sınıfının SOM'da tanımlanması Çizelge 4.13'de gösterilmiştir. Koşum kaydı sırasında federasyondaki diğer federeler tarafından yayınlanan bu nesne sınıfına ait mesajlar (discovery, update, delete)

kaydedilebilecektir. Update mesajları içerisinde nesne örneği öznitelik (attribute) değerleri bulunur ve kayıt modunda bu öznitelik değerleri ile birlikte kaydedilir.

“SurfaceVessel” nesne sınıfının FOM’da tanımlı birçok özneliği bulunmasına karşın, sadece kaydedilmesi istenen öznitelikler SOM’da yer almaktadır. Analiz ihtiyaçları doğrultusunda bir kısım öznitelikler eklenip çıkarılabilir.

Çizelge 4.13 : Uygulama SOM dosyasında nesne sınıfı tanımlama.

```
<Object FedName="Logger_Player"
FullName="HLAobjectRoot.BaseEntity.PhysicalEntity.Platform.Surface
Vessel"
ShortName="SurfaceVessel" PS="3">
  <Attribute AttribName="EntityType" AttribType="composite"/>
  <Attribute AttribName="EntityIdentifier"
AttribType="composite"/>
  <Attribute AttribName="AccelerationVector"
AttribType="composite"/>
  <Attribute AttribName="AngularVelocityVector"
AttribType="composite"/>
  <Attribute AttribName="VelocityVector" AttribType="composite"/>
  <Attribute AttribName="WorldLocation" AttribType="composite"/>
  <Attribute AttribName="DeadReckoningAlgorithm"
AttribType="composite"/>
  <Attribute AttribName="Orientation" AttribType="composite"/>
  <Attribute AttribName="DamageState" AttribType="composite"/>
</Object>
```

4.5.4 Abone ve yayınlama istek işlemlerinin yapılması

Abone ve yayınlama istek işlemleri uygulamanın federasyona katılımı yapıldıktan sonra RTI Elçisi (RTIambassador) aracılığıyla gerçekleştirilir.

Federenin XML formatındaki “benzetim nesne modeli” (SOM) dosyasının ayrıştırma işlemi yapılarak etkileşim ve nesne sınıf tablolarında bulunan tüm sınıflar için abone ve yayınlama istek işlemleri yinelemeli (iterative) olarak gerçekleştirilir.

Belirli bir etkileşim sınıfına abone ve yayınlama isteği için geliştirilen kod blokları Çizelge 4.14 ve 4.15’de gösterilmiştir.

Çizelge 4.14 : Etkileşim sınıfı abone istek fonksiyonu.

```
bool FederateRunner::subscribeInteraction(fedInteraction& intNode)
{
    InteractionClassHandle handle =
        mRtiAmb->getInteractionClassHandle(intNode.FullName());
    mRtiAmb->subscribeInteractionClass(handle);
    return true;
}
```

Çizelge 4.15 : Etkileşim sınıfı yayınlama istek fonksiyonu.

```
bool FederateRunner::publishInteraction(fedInteraction& intNode)
{
    InteractionClassHandle handle =
mRtiAmb->getInteractionClassHandle(intNode.FullName());
    mRtiAmb->publishInteractionClass(handle);
    return true;
}
```

Çizelge 4.16 : Nesne sınıfı abone istek fonksiyonu.

```
bool FederateRunner::subscribeObject(fedObject& objNode) {
    ObjectClassHandle handle =
        mRtiAmb->getObjectClassHandle(objNode.FullName());
    AttributeHandleSet hSet;
    fedObject::Attribute::container::const_iterator i;
    for (i=objNode.Attribute().begin();
        i!=objNode.Attribute().end(); i++) {
        AttributeHandle att;
        att = mRtiAmb->getAttributeHandle(handle,
            i->AttribName());
        hSet.insert(att);
    }
    mRtiAmb->subscribeObjectClassAttributes(handle,hSet);
    return true;
}
```

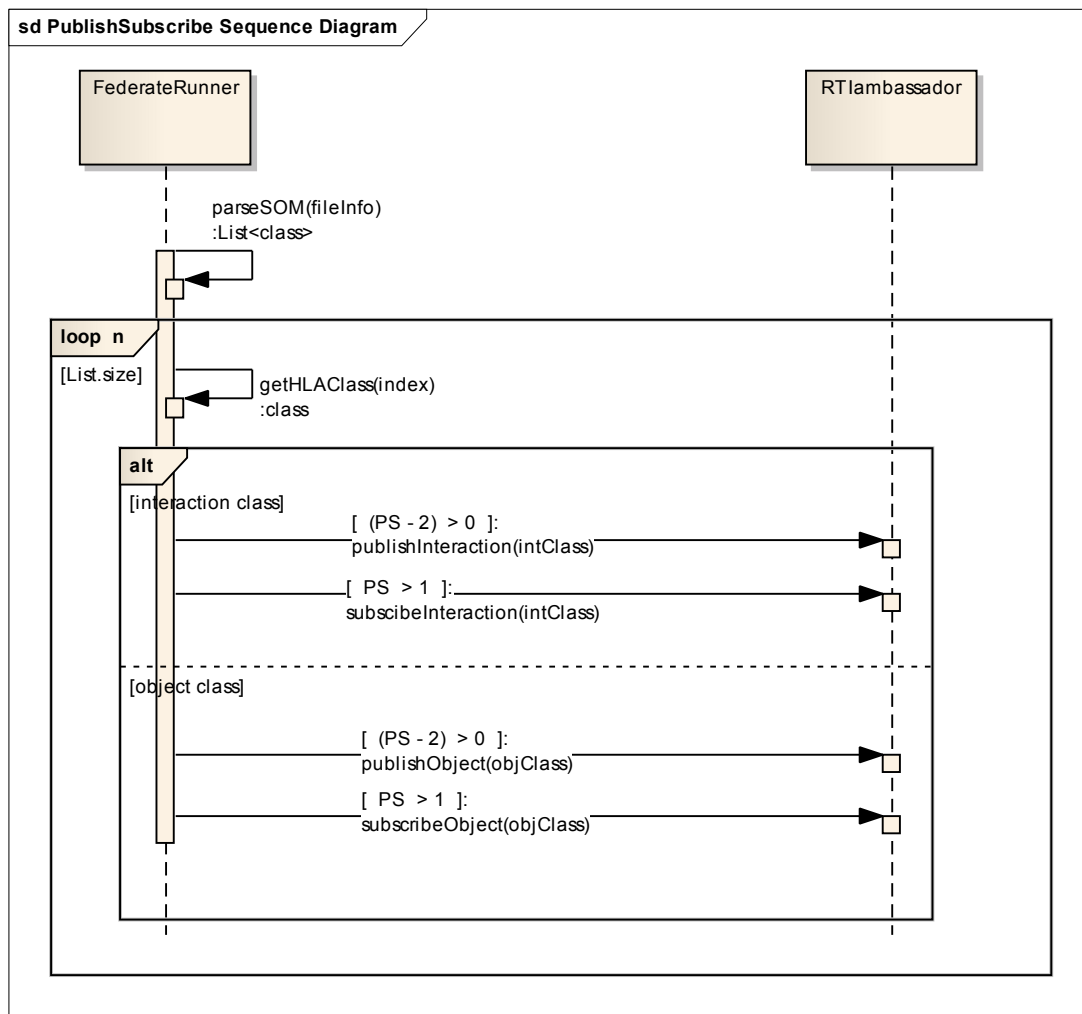
Çizelge 4.17 : Nesne sınıfı yayınlama istek fonksiyonu.

```
bool FederateRunner::publishObject(fedObject& objNode) {
    ObjectClassHandle handle =
        mRtiAmb->getObjectClassHandle(objNode.FullName());
    AttributeHandleSet hSet;
    fedObject::Attribute::container::const_iterator i;
    for (i=objNode.Attribute().begin();
        i!=objNode.Attribute().end(); i++) {
        AttributeHandle att;
        att = mRtiAmb->getAttributeHandle(handle,
            i->AttribName());
        hSet.insert(att);
    }
    mRtiAmb->publishObjectClassAttributes(handle,hSet);
    return true;
}
```

Benzetim nesne modelinde sınıf düğümünde yer alan PS özniteliği aldığı farklı değerlere göre:

- PS = “1” : sadece yayınlama işlemi gerçekleştirilir.
- PS = “2” : sadece abone işlemi gerçekleştirilir.
- PS = “3” : abone ve yayınlama işlemi gerçekleştirilir.

Federenin abone ve yayınlama istek işlemlerinin akış diyagramı Şekil 4.7’da gösterilmiştir.



Şekil 4.7 : Federe abone ve yayınlama istek akış diyagramı

4.5.5 Geri çağrım fonksiyonları

Federe uygulaması RTI üzerinde ilgili nesne ve etkileşim sınıfları abone/yayınlama istek işlemlerini yaptıktan sonra, RTI abone olunan etkileşim ve nesne sınıfı ile ilgili mesajları Federe Elçisi (FederateAmbassador) aracılığıyla geri çağrım fonksiyonlarına iletir.

RTI üzerinden abone ile alınan etkileşim sınıfı mesajları “receiveInteraction” callback fonksiyonuna iletilir. Abone isteği yapılmış etkileşim mesajlarının tümü koşum sırasında bu fonksiyona iletilir. Fonksiyona iletilen mesaj, mesajın ait olduğu sınıfa ve federenin çalışma amacına göre işleme konacaktır.

Bir federe RTI üzerinden üç farklı türde nesne mesajı alır:

- Nesne Kayıt (register) mesajı : “discoverObjectInstance” geri çağrım fonksiyonuna iletilir.
- Nesne Güncellenme (update) mesajı : “reflectAttributeValues” geri çağrım fonksiyonuna iletilir.
- Nesne Silinme (remove) mesajı : “removeObjectInstance” geri çağrım fonksiyonuna iletilir.

Abone isteği yapılmış nesne sınıfı mesajları, mesajın türüne göre ilgili callback fonksiyonuna iletilir. Geri çağrım fonksiyonuna iletilen mesaj, mesajın hangi sınıf örneği olduğuna ve federenin çalışma amacına göre işleme konacaktır.

Geliştirilen federe uygulaması koşum kayıt modunda, geri çağrım fonksiyonlarına iletilen ikili (binary) biçimdeki RTI mesajlarını herhangi bir mantıksal işlemde ve dönüşümden geçirilmeden doğrudan “Kayıt Modülü”ne aktarır. Kayıt modülünde ikili formattaki mesaj serileştirme işlemi ile ikili formatta kaydedilir.

4.5.6 Uygulamanın federasyondan çıkışı

Uygulama kullanıcı tarafından başlatıldığında ilk olarak federasyona katılma işlemi gerçekleştirilir. Daha sonra benzetim kontrol federesi tarafından gönderilen etkileşim mesajları doğrultusunda kayıt modunda çalışmaya başlayabilir, koşum kayıt işlemi sonlandırılabilir. Kullanıcı isterse daha önce kaydedilmiş koşumlardan

birisini seçerek tekrar oynatma işlemi yapabilir. Uygulama kullanıcı tarafından sonlandırılırken federasyondan düzgün bir şekilde ayrılması gerekmektedir.

Uygulama (federe) tekrar oynatma modunda çalışırken, RTI'a nesne sınıfı örneklerini kaydettirir ve kayıtların sırasına göre nesne güncelleme mesajları yayınlar. Uygulama sonlanırken federasyondan düzgün bir şekilde çıkması gerekir. Böylece federe tarafından tekrar oynatma modunda RTI'a kaydettirilen nesne örnekleri RTI tarafından otomatik olarak silinir.

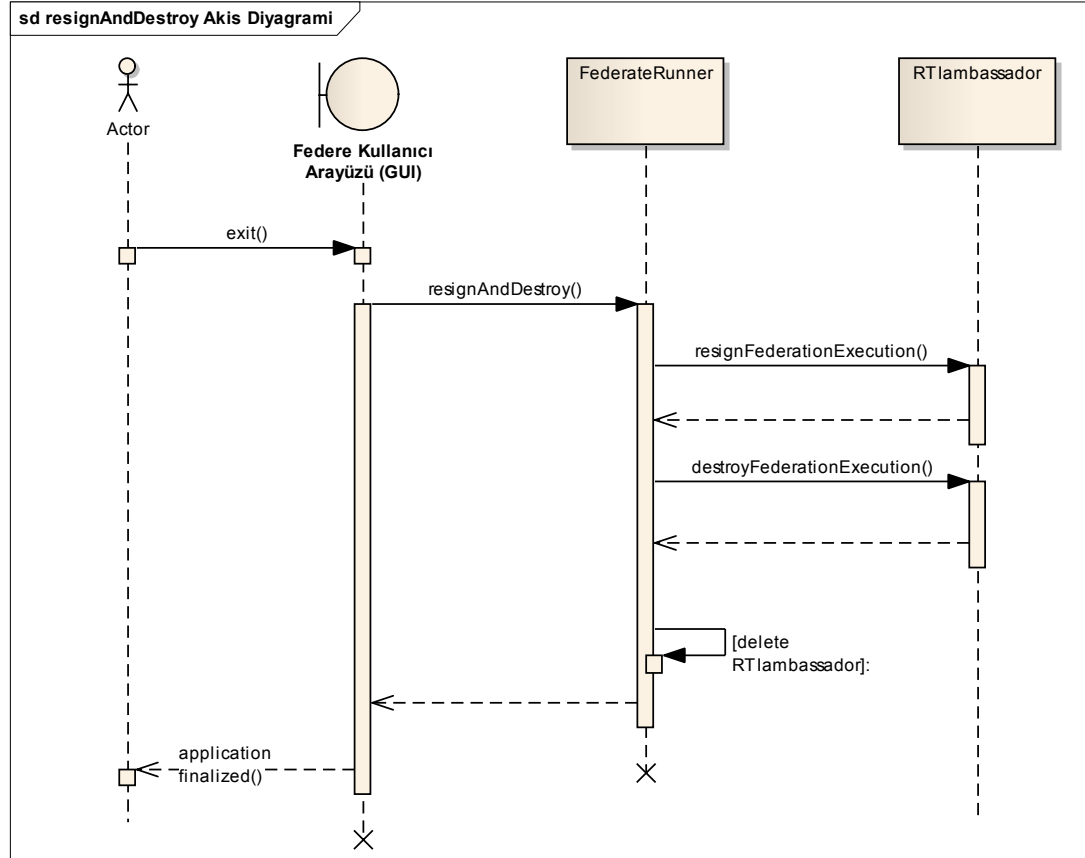
Federasyondan çıkış iki adımda gerçekleşir:

- ***resignFederationExecution*** : Federenin federasyondan ayrılma işlemini sağlayan servis fonksiyonudur.

```
mRtiAmb->resignFederationExecution(rti1516::DELETE_OBJECTS);
```

“DELETE_OBJECTS” argümanı ile federasyondan ayrılma işlemi yapılırken, varsa federenin sahip olduğu nesne örneklerinin silme işlemi de yapılır.

- ***destroyFederationExecution*** : Federasyonu sonlandıran servis fonksiyonudur.



Şekil 4.8 : Uygulamanın federasyondan çıkış akış diyagramı

4.5.7 Özel etkileşim mesajları

Geliştirilen federe uygulaması için bir takım etkileşim sınıfı mesajları özel anlam ifade etmektedir. Uygulamanın koşum kayıt modunda çalışabilmesi için gerekli olan bu etkileşim mesajları federe tarafından özel olarak ele alınırlar. Bu etkileşim mesajları aynı zaman diğer federeler için gerekli federe yaşam döngüsü mesajlarıdır ve benzetim yönetim federesi tarafından gönderilir.

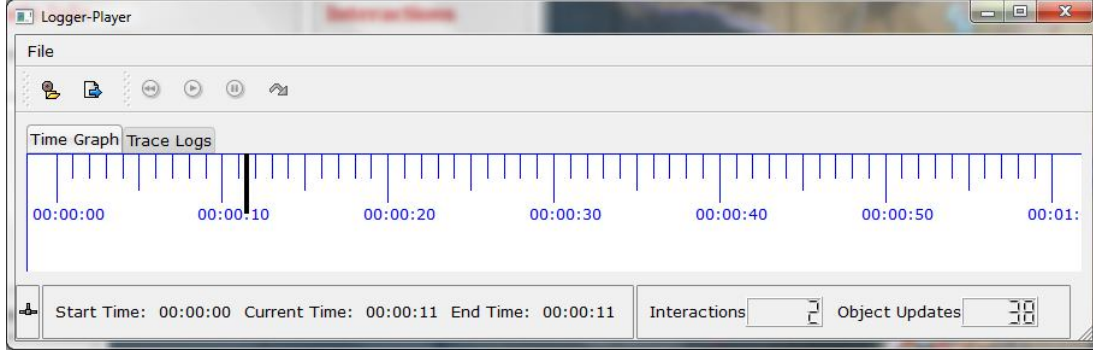
Benzetim Senaryosu Etkileşimi (LoadScenario interaction) : Uygulama koşum kayıt modu için gerekli senaryo bilgisini bu etkileşim mesajı ile alır. Senaryo etkileşim mesajında yer alan “ScenarioFileName” parametresi okunarak, kayıt dosyası bu bilgiyi içerecek şekilde isimlendirilerek kaydedilir.

Yönetim federesi tarafından etkileşim mesajının (ScenarioFileName: “SCN_001.xml”) parametre değeri ile gönderilmesi, koşum kayıt dosyası adı da senaryo ismi ile aynı olacak şekilde “SCN_0001_<ZamanPuluBilgisi>.drf” olarak belirlenir. Kayıt dosyası isminin sonundaki zaman etikeki etkileşimin alındığı zamanı ifade eder. Aynı senaryo dosyası ile birden fazla kez koşum yapıldığı durumlarda, kayıt dosyasının karışmaması için bu şekilde bir çözüm düşünülmüştür.

Koşum kayıt dosyası ismi bu etkileşimin alınması ile belirlendikten sonra, uygulama koşum kayıt başlangıcı için hazır durumda bekler.

Koşum Başlangıç Etkileşimi (StartResume interaction) : Koşum senaryosu etkileşim mesajı alındıktan sonra uygulama “Koşum Başlangıç” etkileşim mesajını beklemeye başlar. Koşum başlangıç etkileşim mesajının alınmasıyla birlikte, koşum kayıt işlemi başlar. Federeye iletilen her türlü etkileşim ve nesne mesajı kaydedilir.

Şekil 4.9’da koşum başlangıç etkileşim mesajından sonra koşum kayıt işleminin başladığı görülmektedir. Sağ altta yer alan interactions ve object updates sayaçlarında o ana kadar kayıt edilen etkileşim ve nesne mesajı sayısı görülmektedir.



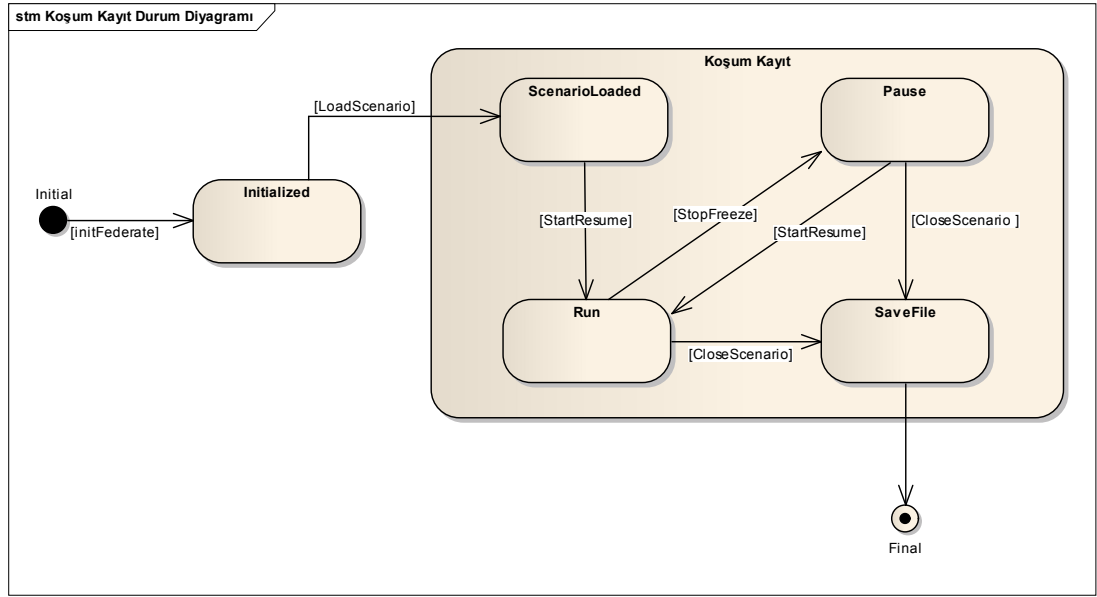
Şekil 4.9 : Koşum başlangıç mesajı ile kayıt işleminin başlaması

Koşum Beklet/Durdur Etkileşimi (StopFreeze interaction) : Koşum kayıt sırasında bu etkileşim mesajının alınması ile, koşum kayıt işlemine ara verilir, uygulama bekleme moduna geçer. Bu sırada alınan etkileşim ve nesne mesajları kaydedilmez. Tekrar “StartResume” etkileşim mesajı alınırsa koşum kayıt işlemine kaldığı yerden devam edilir.

Benzetim Senaryosu Sonlanma Etkileşimi (CloseScenario interaction) : Koşum kayıt işlemi sırasında veya koşum kayıt bekleme durumundayken bu etkileşim mesajının alınması ile, senaryonun sonlandırılacağı anlaşılır. Koşum kayıt sırasında alınan tüm kayıtlar dosyaya, “SCN_0001_<ZamanPuluBilgisi>.drf”, yazılır.

Uygulama tekrar oynatma modu işlemleri (başlatma,durduma, ileri sarma vb.) kullanıcının (operatörün) kontrolüne bırakılmıştır. Yönetim federesi tarafından gönderilen etkileşim mesajları uygulama tekrar oynatma modunda herhangi bir etkiye sahip değildir.

Uygulamanın koşum kayıt modunda yönetim federesi tarafından gönderilen etkileşim mesajlarına göre durum diyagramı Şekilde 4.10’da gösterilmiştir.



Şekil 4.10 : Koşum kayıt federesi etkileşim mesajları durum diyagramı

4.6 Koşum Kayıt Birimi (Logger Module)

Geliştirilen uygulamanın koşum kayıt birimi (module) federasyonda yayınlanan tüm etkileşim ve nesne sınıfı mesajlarını kaydedebilecek yapıda tasarlanmıştır. Analiz safhasında koşum kaydının yorumlanması ve raporlardan çıkarsamalar yapılabilmesi için bazı veriler önemsiz kalırken bazı verilerin daha önemli olduğu durumlar olabilir. Tasarımda bu durum göz önünde bulundurulmuş, bu doğrultuda sadece kaydedilmesi istenen sınıfların tanımlanması yoluyla kayıt işlemi gerçekleştirilir. Kaydedilmek istenen etkileşim ve nesne sınıfları uygulamanın SOM dosyasında tanımlanır.

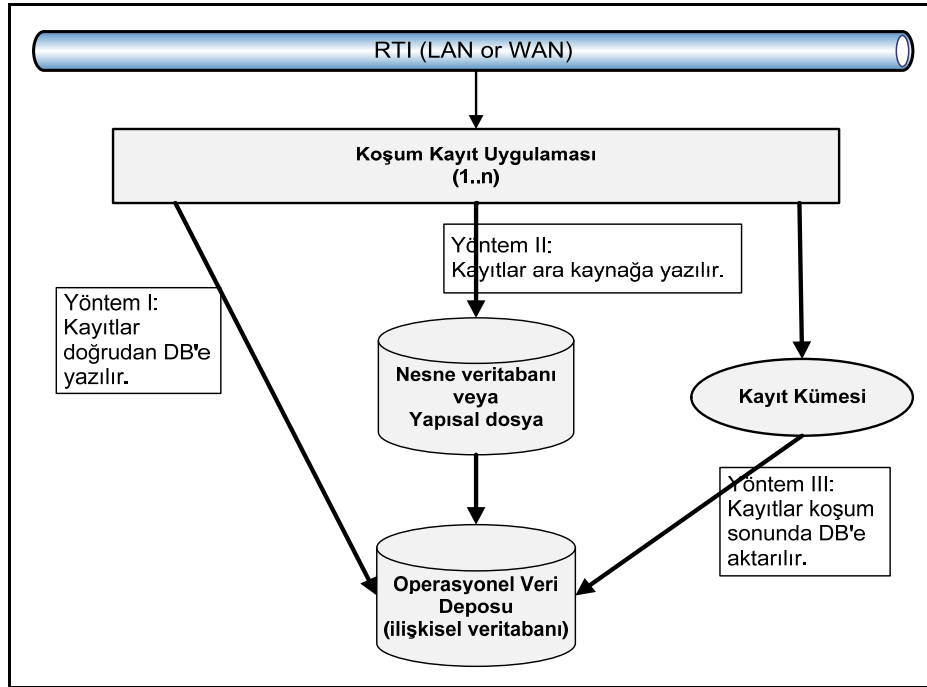
4.6.1 Koşum verisi saklama

Koşum verisini saklamanın en basit yöntemi kayıtları dosya (veya dosyalara) yazmaktır. Dosyaya kayıt yöntemi loglama kabiliyeti sunsa da, çok büyük miktarlarda verinin üretildiği benzetim sistemlerinde verinin analizi çok kolay olmayacaktır[13].

Koşum verilerinin ilişkisel veri tabanında saklanması bu sorunların üstesinden gelecek olan yöntemdir. Verinin merkezi mi yoksa dağıtık şekilde mi depolanacağı ile ilgili kararın verilmesi önemlidir. Merkezi saklama yöntemi analiz için birden fazla kişinin eşzamanlı olarak aynı veriye erişebilmesine imkan sunar. Bazen dış

faktörlerden dolayı (ağın bant genişliği, disk alanı vb. etkenler) merkezi yöntemin uygulanması zor olabilir. Bu gibi durumlarda dağıtık yöntemin kullanılması gerekebilir. Farklı fonksiyonel alanlar belirlenerek dağıtık yöntem uygulanabilir. Böylelikle ilgilenilen veriye de kolay şekilde ulaşılabilir. Ancak farklı fonksiyonel etki alanları arasında analiz yapılması gerektiğinde dezavantaj oluşturur. Verinin merkezi noktada saklanması:

- Veri analizi için esneklik sağlar.
- Gereksiz veriyi azaltır.
- Donanım , yazılım ve ağ kaynağı maliyetlerini azaltır.



Şekil 4.11 : Koşum verisi saklama yöntemleri

Yöntem I : Bu yöntemde koşum performansını olumsuz etkilemesi söz konusu. Sürekli veritabanına gidip yazmak geri dönmek DB performansını olumsuz etkileyecektir, tercih edilmez. Avantajı, bilgiler koşum sırasında anlık olarak DB'e aktarıldığı için anlık analiz imkanı sunar.

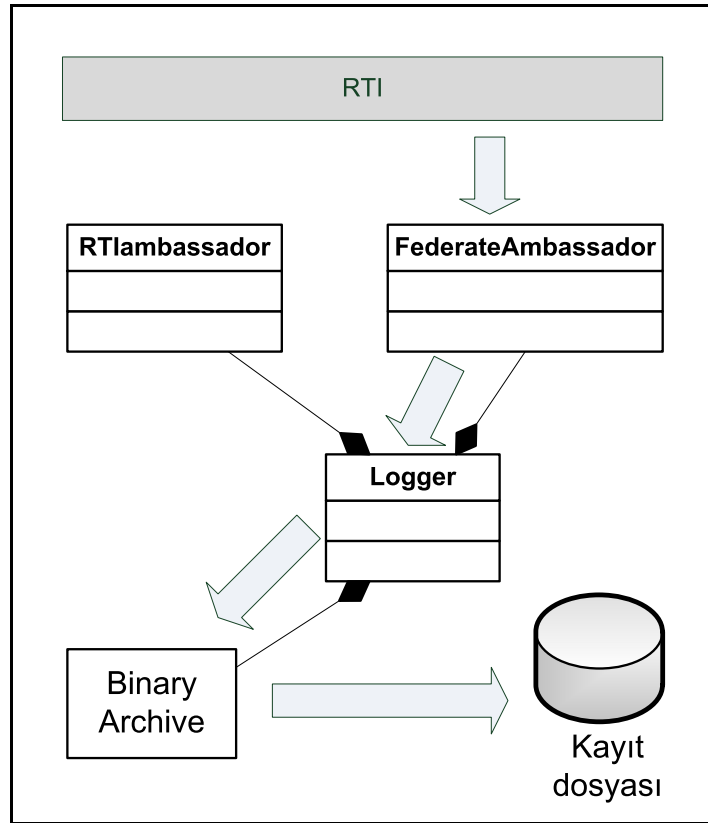
Yöntem II : En yaygın kullanılan yöntem budur. Bu yöntemde koşum performansını olumsuz etkilemesi söz konusu diğer yöntemlere göre daha azdır. Kayıt biriminin veri tabanından yalıtılmış olması önemli bir avantajıdır. DB'e aktarım öncesi kayıt dosyası işleminden geçirilip koşum özeti elde edilip daha sonra DB'e aktarım

yapılabilir. Koşum kayıt dosyasına hızlı şekilde erişebilen farklı amaçlı uygulamalar geliştirilmesi yönünden de avantajlı bir yöntemdir.

Kayıt sırasında RTI üzerinden federeye iletilen her mesajı doğrudan dosyaya yazmak yerine, arabelleğe (memory buffer) hızlı bir şekilde yazılabilir. Bu sayede diske yazma işleminden (I/O) dolayı çalışma kesintiye uğramamış olur. Koşum bitince kayıtlar dosyaya belirli bir formatta yazılır. Daha sonra bu kayıt dosyası analiz işlemleri için veri tabanına aktarılabilir. DB'e aktarım yapılmak istenmediği durumlarda büyük avantaj sağlar.

Yöntem III : Bu yöntemde koşum sırasında kayıtlar DB'e veya dosyaya aktarılmaz, bellekte depolanır. Federasyon koşumu bittiğinde kayıtlar DB'e aktarılmadan önce ön işleminden geçirilebilir. Örneğin threshold koyulup, belirli milisaniye aralıklarla kayıtlar elenerek DB'e aktarım gerçekleştirilir. Bu yöntemin dezavantajı, DB'e aktarım gerçekleşmeden koşumun geri bildirimini almak, analiz yapmak mümkün değildir.

Çalışmada geliştirilen koşum kayıt birimi, kayıt saklama için yukarıda anlatılan yöntemlerden “Yöntem II” esas alınarak geliştirme yapılmıştır. Uygulamanın kayıt modundaki kavramsal modeli Şekil 4.12’de gösterilmiştir. Okun yönü RTI nesne ve etkileşim mesajlarının ilerleyişini göstermektedir.



Şekil 4.12 : RTI– federe – kořum kayıt birimi kavramsal modeli

RTI üzerinden abone ile alınan etkileřim ve nesne mesajları kořum kayıt birimine aktarılır. Kayıt birimi alınan mesajın türüne göre (interaction, discovery, update, remove) alınan mesajı serileřtirme işleminden geçirerek arřive aktarma işlemi yapar. RTI’den alınan her etkileřim ve nesne mesajı için bu işlem tekrarlanır. Benzetim yönetim federesi tarafından kořum senaryosu sonlandır etkileřim mesajı (CloseScenario) alındığında o ana kadar arřivde saklanan kayıtlar kayıt dosyasına yazılır.

4.6.2 Serileřtirme (Serialization)

Serileřtirme, veri depolama (data storage) ve veri iletimi (transmission) bağlamında, bir veri yapısı veya nesnenin bayt dizisine dönüřtürülerek dosyada, arabellekte (memory buffer) depolanması veya network üzerindeki başka bir bağlantı noktasına transfer edilmesi işlemidir. Serileřtirme işleminde oluřan bayt dizisinden, aynı veri yapısı veya nesnenin bir kopyası oluřturulabilir. Karmařık veri tipleri, özellikle

referans içeren nesneler için serileştirme işlemi görüldüğü kadar basit bir işlem değildir.

Nesne serileştirme işlemi “marshalling” olarak da bilinmektedir. Oluşturulan bayt dizisinin tam tersi operasyonla geriye tekrar aynı nesnenin elde edilmesine deserialization (unmarshalling) adı verilir. Nesne sürekliliği (object persistence), uzak yordam çağrımında parametre geçirme vb. diğer farklı kullanım amaçları vardır.

1990’lı yılların sonlarına doğru standart serileştirme protokollerinin yanında alternatif olarak XML serileştirme protokolü ortaya çıkmıştır. XML serileştirme işleminde ortaya çıkan veri insanlar tarafından da okunabilen metin temelli kodlanmış veridir. İşletim platformu ve programlama dilinden bağımsız iletişim için uygundur. Dezavantajı, bayt-akım (byte-stream) temelli serileştirmede bulunan kompakt özellik kaybolur. XML serileştirme daha çok istemci ve sunucu temelli web uygulamalarında yapısal verinin asenkron iletilmesinde tercih edilen bir yöntemdir.

Tez çalışmasında serileştirme operasyonları için boost c++ kütüphanesinde bulunan serileştirme kütüphanesinden yararlanılmıştır [14].

Boost serileştirme kütüphanesinde, serileştirme işlemi ile oluşan bayt dizisine “arşiv” (archive) adı verilir. Arşiv, bir dosyada kayıtlı ikili (binary) veri, metin veya XML formatlı veri olabileceği gibi, yazılım geliştiricinin oluşturduğu farklı bir formatta veri de olabilir.

Boost serileştirme kütüphanesi ile nesne serileştirme ve arşiv kullanımı ile ilgili örnek kod Çizelge 4.18’de verilmiştir. İkinci adımda serileştirme işleminde oluşturulan byte dizisinden nesnenin tekrar geri elde edilmesi gösterilmiştir.

Çizelge 4.18 : Boost nesne serileştirme kod örneği.

```
class gps_position {
private:
    friend class boost::serialization::access;

    template<class Archive>
    void serialize(Archive & ar, const unsigned int version) {
        ar & degrees;
        ar & minutes;
        ar & seconds;
    }

    int degrees;
    int minutes;
    float seconds;
public:
    gps_position(){};
    gps_position(int d, int m, float s) :
        degrees(d), minutes(m), seconds(s)
    {}
};

int main() {
    std::ofstream ofs("filename"); // open a character archive for output

    const gps_position g(35, 59, 24.567f);

    // save data to archive
    {
        boost::archive::text_oarchive oa(ofs);
        oa << g; // write class instance to archive
    }

    // some time later restore the class instance to its original state
    gps_position newg;
    {
        std::ifstream ifs("filename"); // open an archive for input
        boost::archive::text_iarchive ia(ifs);
        ia >> newg; // read class state from archive
    }
    return 0;
}
```

RTI üzerinden abone ile alınan etkileşim ve nesne mesajları callback fonksiyonlarda işleme alınır. Uygulama kayıt modunda, RTI'dan alınan bu mesajlar koşum kayıt birimine aktarılır. Logger sınıfında uygun veri yapılarına dönüştürülerek ikili (binary) formatta serileştirme yapılır.

RTI üzerinden abone ile alınan etkileşim ve nesne mesajları dört farklı grupta toplanır:

- Etkileşim mesajı (interaction message)
- Nesne oluşum mesajı (object discovery)
- Nesne güncelleme mesajı (object update)
- Nesne silinme mesajı (object remove)

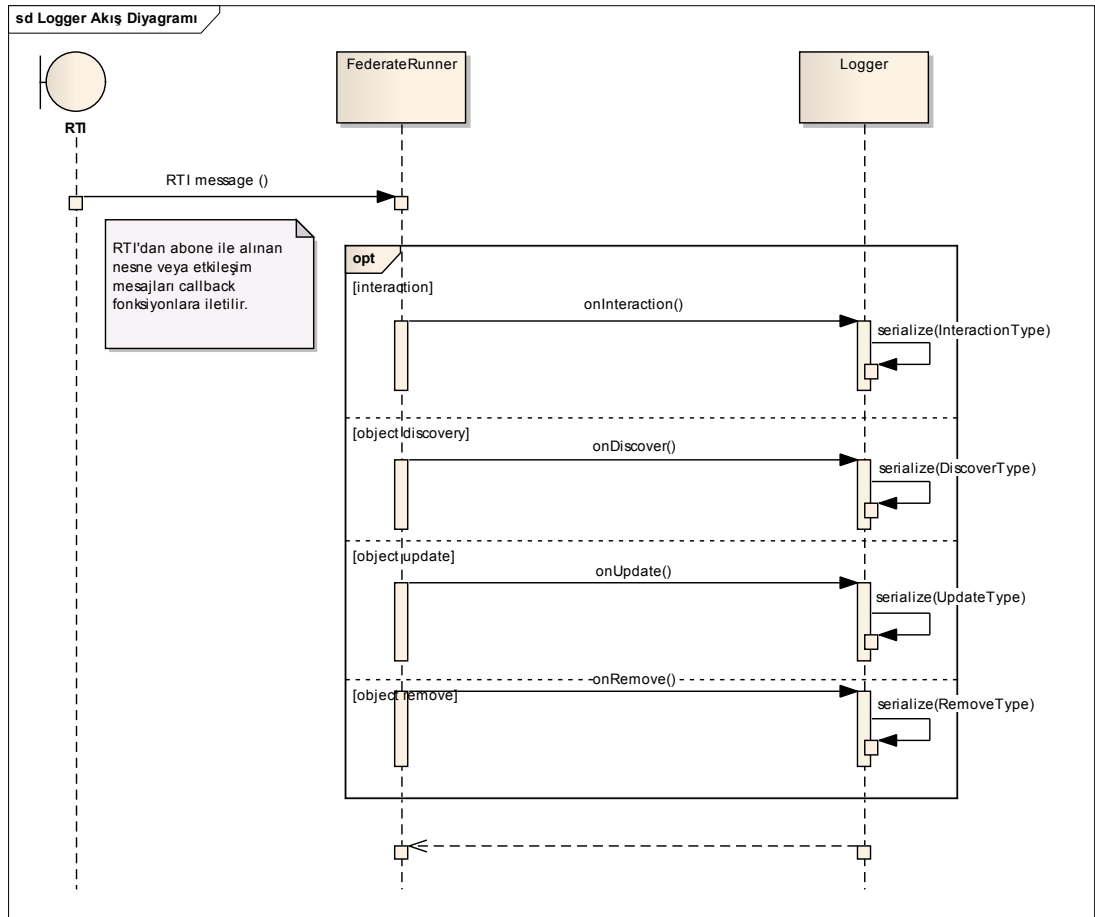
Her mesaj türü için birer numaralama (enumeration) türü ve mesajın içeriğine ilişkin veri yapısı tanımlanmıştır.

Çizelge 4.19 : Koşum kayıt veri yapıları.

```
enum RecordType {INTERACTION = 0, UPDATE = 1, DISCOVER = 2, REMOVE = 3};

struct InteractionType;
struct DiscoverType;
struct UpdateType;
struct RemoveType;
```

RTI'dan nesne ve etkileşim mesajlarının alınması ve koşum kayıt biriminde kaydedilmesine ilişkin akış diyagramı Şekil 4.13'de gösterilmiştir.



Şekil 4.13 : RTI Mesajlarının Koşum Kayıt İşlemi Akış Diyagramı

RTI üzerinden alınan nesne ve etkileşim mesajları, FederateRunner callback fonksiyonlarına “VariableLengthData” veri yapısı içerisinde iletilir. VariableLengthData, ikili formattaki bayt dizisidir. VariableLengthData veri yapısı için de serileştirme fonksiyonları ayrıca yazılmıştır. Böylece callback fonksiyondan kayıt birimine VariableLengthData türünde iletilen veri, doğrudan serileştirme fonksiyonlarına girdi olarak verilebilir.

Her veri yapısı kendi içerisinde “serialize” metodu ile serileştirme operasyonunu tanımlamıştır. Callback fonksiyonlarda alınan mesajlar, içeriği ve zaman bilgisi ile birlikte serileştirilerek arabellekte bulunan arşive eklenir. Böylece RTI üzerinden abone ile alınan mesajlar serileştirmesi yapılarak kaydedilmiş olur.

Uygulama kayıt modunda RTI üzerinden abone ile alınan etkileşim ve nesne mesajları Federe tarafından Logger sınıfında bulunan ilgili fonksiyona aktarılır. Logger sınıfı gelen mesajın arşive serileştirmesini yapar. Böylelikle federeye abone ile gelen mesajlar kaydedilmiş olur.

4.6.3 Etkileşim mesajlarının kaydedilmesi

RTI üzerinden abone ile alınan etkileşim mesajları, FederateRunner sınıfında bulunan receiveInteraction callback fonksiyonuna iletdikten sonra ikili formattaki veri, Logger sınıfında tanımlanan Logger::onInteraction fonksiyonuna aktarılır. Bu fonksiyonda alınan ikili veri serileştirilebilir veri yapısı olarak tasarlanan “InteractionType” veri yapısına dönüştürülerek zaman etiketiyle birlikte arşive serileştirme işlemi yapılır.

Çizelge 4.20 : InteractionType serileştirilebilir veri yapısı.

```
struct InteractionType {
    InteractionType(const std::string& name="", const ParamMap&
parameters=ParamMap())
        :mName(name), mParameters(parameters)
    {}
    template<class Archive>
    void serialize(Archive & ar, const unsigned int version) {
        ar & mName & mParameters;
    }
    std::string mName;
    ParamMap mParameters;
};
```

4.6.4 Nesne keşif mesajlarının kaydedilmesi

RTI üzerinden alınan nesne keşif (object discovery) mesajları, FederateRunner sınıfında bulunan discoverObjectInstance callback fonksiyonuna iletildikten sonra, oluşturulan nesne örneği tanıtım (handle), sınıf adı ve isim bilgileri ile birlikte, Logger sınıfında bulunan Logger::onDiscover fonksiyonuna aktarılır. Bu fonksiyonda alınan parametre değerlerine göre “DiscoverType” veri yapısına dönüştürülerek zaman etiketiyle birlikte arşive serileştirme işlemi yapılır.

Çizelge 4.21 : DiscoverType serileştirilebilir veri yapısı.

```
struct DiscoverType {
    DiscoverType(const std::string& name, const std::string& instanceHandle)
        : mName(name), mInstanceHandle(instanceHandle)
    {}
    template<class Archive>
    void serialize(Archive & ar, const unsigned int version) {
        ar & mName & mInstanceHandle ;
    }
    std::string mName;
    std::string mInstanceHandle;
};
```

4.6.5 Nesne güncelleme mesajlarının kaydedilmesi

RTI üzerinden alınan nesne güncelleme mesajları, FederateRunner sınıfında bulunan reflectAttributeValues callback fonksiyonuna iletildikten sonra, oluşturulan nesne örneği öznitelik değerleriyle birlikte Logger sınıfında tanımlanan Logger::onUpdate fonksiyonuna aktarılır. Bu fonksiyonda nesne öznitelik değerleriyle birlikte göre “UpdateType” veri yapısına dönüştürülerek zaman etiketiyle birlikte arşive serileştirme işlemi yapılır.

Çizelge 4.22 : UpdateType serileştirilebilir veri yapısı.

```
struct UpdateType {
    template<class Archive>
    void serialize(Archive& ar, const unsigned int version) {
        ar & mName & mHandle & mAttributes;
    }

    std::string mName; // nesne sınıfı adı
    std::string mHandle; // nesne tanıtıcı (handle)
    AttribMap mAttributes; // nesne sınıfı öznitelik isimleri ve değerleri
};
```

Nesne güncelleme mesajının kayıt işlemi için Logger sınıfındaki onUpdate fonksiyonu kod bloğu Çizelge 4.23’de gösterilmiştir. Diğer fonksiyonların kod içeriği onUpdate fonksiyonu ile benzer yapıda olduğu için her fonksiyon kaynak kodunun gösterimine gerek olmadığı düşünülmüştür.

Çizelge 4.23 : Nesne güncelleme mesajı kayıt fonksiyonu.

```
void Logger::onUpdate(ObjectInstanceHandle theObject, ObjectClassHandle
theObjectClass, AttributeHandleValueMap const& theAttributeValues) {
    RecordType type = UPDATE;
    std::string objectName = mFedRunner.getAmbassador()->
getObjectClassName(theObjectClass);
    std::string objectInstanceHandle = theObject.toString();

    std::vector<std::string> attribNames;
    AttribMap attributes;
    AttributeHandleValueMap::const_iterator iter =
theAttributeValues.begin();
    for (; iter != theAttributeValues.end(); ++iter) {
        std::string attribName = mFedRunner.getAmbassador()->
getAttributeName(theObjectClass, iter->first);
        attribNames.push_back(attribName);
        attributes[attribName] = iter->second;
    }

    UpdateType object(objectName, objectInstanceHandle, attributes);

    wcDiff = mWCTStrategy->currentTime(); // gerçek zaman
    stDiff = mSTStrategy->currentTime(); // benzetim zamanı
    wcMicrosecTime = wcDiff.total_microseconds();
    stMicrosecTime = stDiff.total_microseconds();

    *mArchive << wcMicrosecTime
               << stMicrosecTime
               << type
               << object;
}
```

4.6.6 Nesne silinme mesajlarının kaydedilmesi

RTI üzerinden alınan nesne silinme (object remove) mesajları, FederateRunner sınıfında bulunan removeObjectInstance callback fonksiyonuna iletdikten sonra, nesne örneği tanıtıcı parametresi (handle), Logger sınıfında bulunan Logger::onRemove fonksiyonuna aktarılır. Bu fonksiyonda veri “RemoveType” yapısına dönüştürülerek zaman etiketiyle birlikte arşive serileştirme işlemi yapılır.

Çizelge 4.24 : RemoveType serileştirilebilir veri yapısı.

```
struct RemoveType {
    RemoveType(const std::string& instanceHandle = "")
        : mInstanceHandle(instanceHandle)
    {}

    template<class Archive>
    void serialize(Archive & ar, const unsigned int version) {
        ar & mInstanceHandle ;
    }

    std::string mInstanceHandle;
};
```

4.6.7 Koşum kaydı sonlandırma

Federe, benzetim senaryosu sonlandırma etkileşimi mesajını (CloseScenario) aldıktan sonra, “ENDOFRECORD” türünde son bir kayıt oluşturarak zaman etiketiyle birlikte arşiv sonuna serileştirmeyi yaparak arşivi sonlandırır, ikili formattaki veriyi “.drf” uzantılı kayıt dosyasına yazar.

Benzetim senaryosu etkileşim mesajı senaryo dosya parametresi “SCN_01.xml” olan senaryonun koşum kaydı dosyası sonuna zaman etiketi eklenerek “SCN_01_18.07.2010_13.55.49.drf” adıyla .drf uzantılı olarak kaydedilir.

4.6.8 Kayıt dosyası yapısı

Federe tarafından RTI’den abone ile alınan etkileşim ve nesne mesajları ikili biçimdedir. İkili biçimdeki nesne öznitelik değerleri ve etkileşim parametre değerlerini ASCII biçimli elde edebilmek, veriyi bir dizi dönüşümden geçirmekle mümkün olabilir. İnsanlar tarafından da okunabilecek bir kayıt dosyasının ASCII biçimli oluşturulması gerekir. Bunun için RTI’den alınan mesajların da ASCII biçimine dönüşümünün yapılması ve dosyaya da aynı biçimde yazılması gerekir. Kayıt esnasında dönüşümden kaynaklanacak gecikmeleri en aza indirmek için, RTI’den alınan ikili veri dönüşüm yapılmadan doğrudan alındığı şekliyle ikili veri biçiminde kaydedilir.

Çalışmada uygulamanın koşum kaydı performansı göz önünde bulundurularak, kayıt dosyası ikili biçimde oluşturulmaktadır.

Koşum kayıt dosyasındaki son kayıt “ENDOFRECORD” kayıdır. Her kayıt, zaman etiketiyle birlikte kaydedilmiştir. Kayıtlar mesajın büyüklüğüne göre farklı uzunluklarda olabilmektedir. Etkileşim ve nesne sınıfı mesajlarının kayıt yapısı aşağıda listelenmiştir.

Etkileşim mesajı kayıt yapısı :

Wall Clock Time	Simulation Time	RecordType <INTERACTION>	Interaction class name	Parameters [0..n]
--------------------	--------------------	-----------------------------	---------------------------	----------------------

Nesne oluşum mesajı (discovery) kayıt yapısı :

Wall Clock Time	Simulation Time	RecordType <DISCOVER>	Object class name	Instance handle
--------------------	--------------------	--------------------------	----------------------	--------------------

Nesne güncelleme mesajı (object update) kayıt yapısı :

Wall Clock Time	Simulation Time	RecordType <UPDATE>	Object class name	Instance handle	Attributes [0..n]
--------------------	--------------------	------------------------	----------------------	--------------------	----------------------

Nesne silinme mesajı (object remove) kayıt yapısı :

Wall Clock Time	Simulation Time	RecordType <REMOVE>	Instance handle
--------------------	--------------------	------------------------	--------------------

Koşum kayıt dosyasında kayıtların dizilimi aşağıda gösterilmiştir.

Kayıt [1]	Kayıt [2]	Kayıt [3]	...	Kayıt [n-1]	ENDOFRECORD [n]
--------------	--------------	--------------	-----	----------------	--------------------

4.6.9 Yeni kayıt sınıfı eklenmesi

Geliştirilen uygulamanın benzetim nesnesinde modelinde (SOM) tanımlı her etkileşim ve nesne sınıfı parametre ve öznitelik değerleri ile birlikte kaydedilecek şekilde tasarlanmıştır.

Her benzetim sisteminin kendine has analiz ve yorumlama ihtiyaçları farklı olabilir, ve bu ihtiyaçlar zamanla bile değişkenlik gösterebilmektedir.

Bir nesne sınıfına ait tüm mesajların istenilen öznitelik değerleriyle birlikte kaydedilmesi için federe SOM dosyasına tanımlanması gerekir.

SOM dosyası güncelleme adımları aşağıda anlatılmıştır:

- Kaydedilecek etkileşim veya nesne sınıfının Federasyon Nesne Modelindeki (FOM) sınıf adı belirlenir.

Örneğin, FOM'da tanımlı "Aircraft" nesne sınıfına ait tüm mesajların kaydedilmesi için, "HLAobjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft" tam adı belirlenir.

- Etkileşim veya nesne sınıfı güncelleme mesajlarında hangi parametrelerin kaydedileceği belirlenir.
- "Aircraft" nesne sınıfının FOM'da tanımlı 44 adet özniteliği bulunmasına karşın, analiz ihtiyaçları doğrultusunda bunlardan bazıları kaydedilmek istenmeyebilir.

Bu örnekte, {EntityType, EntityIdentifier, VelocityVector, WorldLocation, Orientation} öznitelik bilgilerinin kaydedileceğini düşünebiliriz.

- Uygulama SOM dosyasında ilgili düzenleme yapılır. Eğer kaydedilecek sınıf etkileşim sınıfıysa etkileşim sınıf tablosuna, nesne sınıfıysa nesne sınıf tablosuna gerekli ekleme yapılır.

Çizelge 4.25 : Yeni kayıt sınıfı eklenmesi.

```
<Object FedName="Logger_Player"
FullName="HLAobjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft"
ShortName="Aircraft" PS="3">
  <Attribute AttribName="EntityType"/>
  <Attribute AttribName="EntityIdentifier"/>
  <Attribute AttribName="VelocityVector"/>
  <Attribute AttribName="WorldLocation"/>
  <Attribute AttribName="Orientation"/>
</Object>
```

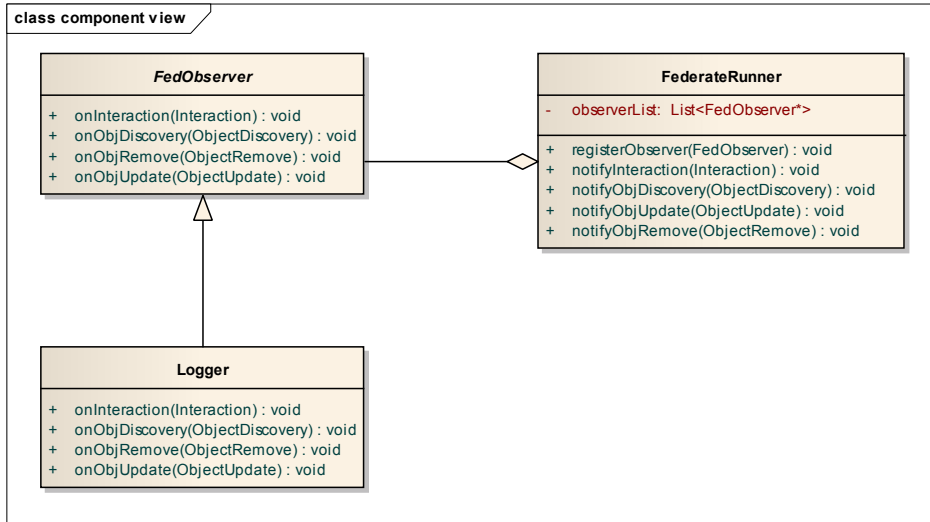
Geliştirilen bu mimari yapı, uygulama kaynak kodlarında değişiklik yapmaksızın yeni kayıt sınıfları eklenip çıkarma imkanı sunar. Bu tür değişiklikler karşısında uygulamanın kaynak kodlarının tekrar derlenmesi gereksiniminin önüne geçilmiştir. Bu yapı geliştirme maliyetlerini azalacak ve geliştiricilere olan bağımlılığı ortadan kaldıracak bir yöntem olarak düşünülmüştür.

4.6.10 Koşum kayıt birimi mimari tasarım detayları

Koşum kayıt biriminin (Logger) yazılım tasarımında iki tane kritik özellik göze çarpmaktadır. Bunlardan birisi gözlemci tasarım deseni temel alınarak geliştirilmiş olmasıdır. Diğer bir özellik ise, koşum kaydı performansını ilgilendiren, nesne güncellememesajları eşik (threshold) değeri ile ilgili yapılan tasarımdır.

4.6.10.1 Gözlemci tasarım deseni (observer design pattern)

“Koşum Kayıt Birimi”, nesneye yönelik tasarım desenlerinden "observer" tasarım deseni temel alınarak geliştirilmiştir. Bu tasarım deseninde, nesne kendisine bağımlı olan gözlemci nesnelerin listesini tutarak, kendisinde oluşan durum değişikliklerini gözlemcilerine bildirir. Özellikle dağıtık olay taşıma sistemlerinin gerçekleştiriminde kullanılır.



Şekil 4.14 : Logger sınıf diyagramı

“Koşum Kayıt Birimi” FederateRunner sınıfına gözlemci olarak bağlanır. Bu tasarım sayesinde, istenirse Logger sınıfı federasyondaki diğer federe kaynak kodlarına entegre edilerek, her bir federenin tam dağıtık mimari yaklaşımında anlatıldığı gibi kendi koşum kaydını yapabilmesi sağlanır. Ancak federelerin kendi çalışma döngüsü dışında ayrıca koşum kayıt işlemi yapması tercih edilen bir yöntem değildir.

4.6.10.2 Nesne güncelleme mesajı kayıt eşiği (update threshold)

Koşum sırasında federeler tarafından oluşturulan her nesne örneğine RTI tarafından tekil tanımlayıcı (handle) değer atanır. Çizelge 4.26’da yeni nesne oluşturma ve elde edilen “instanceHandle” tekil tanımlayıcısı gösterilmiştir.

Çizelge 4.26 : Nesne oluşturma.

```

std::string className =
    "HLAObjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft";
ObjectClassHandle classHandle =
    mRtiAmb->getObjectClassHandle(className);
ObjectInstanceHandle instanceHandle =
    mRtiAmb->registerObjectInstance(classHandle);
  
```

Koşum Kayıt uygulamasına RTI tarafından gönderilen nesne güncellemelerinin sıklığı her nesne örneği için değişkenlik gösterebilir. Federasyonda yer alan federeler sahip oldukları benzetim nesnelerini kendi çalışma prensipleri doğrultusunda RTI’a belirli aralıklarla güncelleme mesajları gönderir. Bir nesne 100ms aralıklarla güncellenirken, başka bir nesnenin güncellenme frekansı 10ms olabilir. Geliştirilen uygulamanın kayıt biriminde, abone olduğu her nesne güncelleme mesajını kaydetmesi performansı olumsuz yönde etkileyebilir ve koşum kayıt dosyasının

büyüklüğü de çok büyük değerlere çıkabilir. Bu durumu önlemek için Eşik değeri yaklaşımı geliştirilmiştir.

Koşum kayıt birimi, her nesnenin son kaydedilen güncelleme mesajı zamanı bilgisine göre, aynı nesnenin bir sonraki güncelleme mesajını kaydedip kaydetmeme kararını eşik değerine bakarak verir.

Eşik değeri = 250ms için, aynı nesneye ait güncelleme mesajları kayıt durumları aşağıdaki çizelgede gösterilmiştir.

<i>Güncelleme Zamanı (ms)</i>	t1 = 10	t2 = 300	t3 = 400	t4 = 580	t5 = 650
<i>Kayıt Durumu</i>	diff=0ms ✓	diff=290ms ✓	diff=100ms ✗	diff=280ms ✓	diff=70ms ✗

Eşik değeri, geliştirilen uygulamanın yapılandırma dosyasından kolayca değiştirilebilecek şekilde tasarlanmıştır. Uygulama kaynak kodlarının tekrar derlenmesine gerek kalmadan eşik değeri artırılıp azaltılabilir.

Eşik değeri ile koşum kaydı dosyasının gereksiz olarak yüksek boyutlara ulaşması engellenir. Koşum kaydının yüksek duyarlılıkla (high fidelity) yapılması isteniyorsa eşik değeri azaltılarak güncelleme mesajlarının daha sık aralıklarla kaydedilmesi sağlanır. Eşik değerinin çok yüksek tutulması, kayıt performansına olumlu yansıyacak, diğer taraftan koşum kaydının düşük sadakatli olmasına neden olacaktır.

4.7 Tekrar Oynatma Birimi (Replay Module)

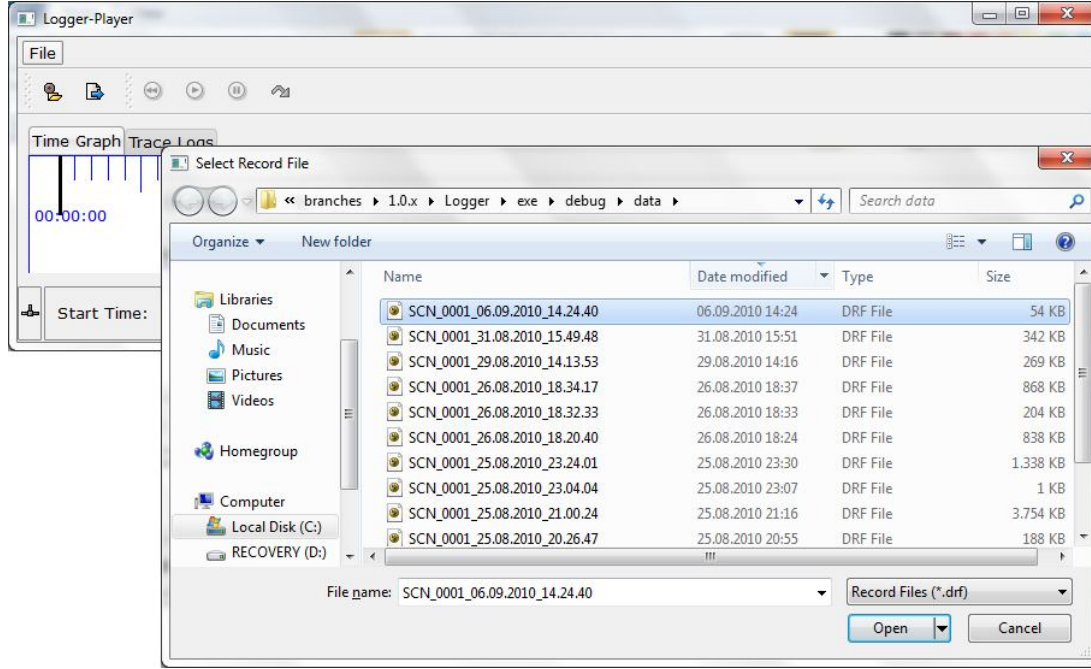
Geliştirilen uygulamanın asıl amacı federasyonun koşum kaydını yapmak ve kaydedilen koşumu tekrar oynatmayı sağlamaktır.

HLA zaman yönetiminde zaman sürekli olarak ileri doğru akar. Federeler zamanda geriye doğru gidemez. Bununla birlikte, eğitim amaçlı benzetim sistemlerinde, eğitmenin koşulan senaryoyu incelemesi sırasında tekrar oynatma, koşumu ileri / geri sarma vb. ihtiyaçlar olacaktır. Bu gibi isteklere cevap veren bir yazılımın olması operatör veya eğitmenler açısından önemli bir avantaj olacaktır.

Tekrar oynatma işlemini kolaylaştırmak için, koşum kayıt birimi aldığı veri paketini (etkileşim ve nesne mesajları) zaman bilgisi ile kaydeder. Tekrar oynatma işleminde

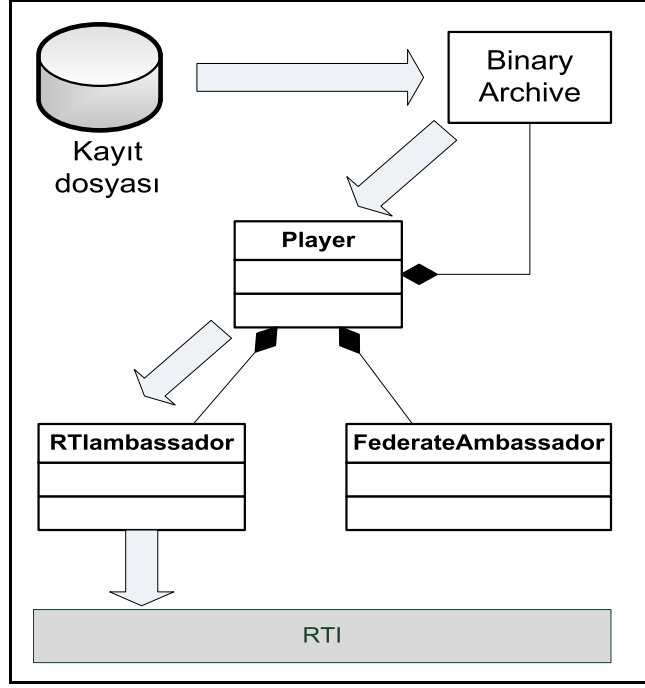
dosyadan okunan kayıtlar zaman bilgisiyle indekslenerek, kořumun istenilen zamana ilerletilmesi saęlanır.

Çalışmada geliştirilen uygulama bu gibi ihtiyaçlar düşünölerek tasarlanmıştır. Tekrar oynatma modunda kořumu istenilen zamana alma, oynatma hızını artırma / azaltma özellikleri geliştirilmiştir.



Şekil 4.15 : Kořum tekrar oynatım için kayıt dosyası seçimi

Uygulama tekrar oynatma modunda kavramsal modeli Şekil 4.16’da gösterilmiştir. Okun yönü etkileşim ve nesne mesajlarının ilerleyişini göstermektedir.



Şekil 4.16 : RTI – federe – tekrar oynatma birimi kavramsal modeli

4.7.1 Koşum kayıt dosyasının okunması

Koşum kayıt dosyasının yapısı daha önceki bölümlerde anlatılmıştır. Etkileşim ve nesne mesajları kayıt dosyasında belirli bir formatta zaman etiketli olarak depolanmıştır. Dosya sonunu ENDOFRECORD kaydı belirler.

Koşum kayıt dosyasından etkileşim ve nesne mesajlarının elde edilebilmesi için geriye serileştirme (deserialization) işlemi uygulanır. Serileştirme işlemiyle ikili biçime dönüştürülen bir veri yapısı geriye serileştirme işlemiyle geri elde edilebilir.

Tekrar oynatma için koşum kayıt dosyası okuma kipinde açılarak, geriye serileştirme işlemiyle etkileşim ve nesne mesajlarının tümü zaman bilgisiyle indekslenir. Bu işlemin sonunda uygulama tekrar oynatma işleminin başlamasına hazır durumdadır.

Kayıtların dosyadan okunarak zaman bilgisiyle indekslenmesi işlemi Çizelge 4.27’de gösterilmiştir.

Çizelge 4.27 : Koşum kayıt dosyası okuma fonksiyonu.

```
void Player::read( std::ifstream &ifs) {
    while (ifs.good()) {
        try {
            bp::time_duration::tick_type wcMicroseconds;
            bp::time_duration::tick_type stMicroseconds;
            *mArchive >> wcMicroseconds;
            *mArchive >> stMicroseconds;
            bp::time_duration time =
mPlayMode == WALLCLOCK_TIME ? bp::microseconds(wcMicroseconds) :
bp::microseconds(stMicroseconds);
            RecordType type;
            *mArchive >> type;
            if (type == ENDOFRECORD) {
                std::cout << "End of Record\n";
                mEndOfRecord = time;
                break;
            }
            switch (type) {
            case INTERACTION : {
                InteractionType inter;
                *mArchive >> inter;
                mRecords.insert(std::make_pair(time,
                    RecordVariant(inter)));
                break;
            }
            case UPDATE : {
                UpdateType object;
                *mArchive >> object;
                mRecords.insert(std::make_pair(time,
RecordVariant(object)));
                break;
            }
            case DISCOVER : {
                DiscoverType discover ;
                *mArchive >> discover;
                mRecords.insert(std::make_pair(time,
RecordVariant(discover)));
                break;
            }
            case REMOVE : {
                RemoveType remove ;
                *mArchive >> remove;
                mRecords.insert(std::make_pair(time,
RecordVariant(remove)));
                break;
            }
            } // end of switch
        } catch(boost::archive::archive_exception&) {
            if (ifs.eof()) {
                std::cout << "End of File\n";
            }
            break;
        }
    }
}
```

4.7.2 Kayıtların federasyonda yayınlanması

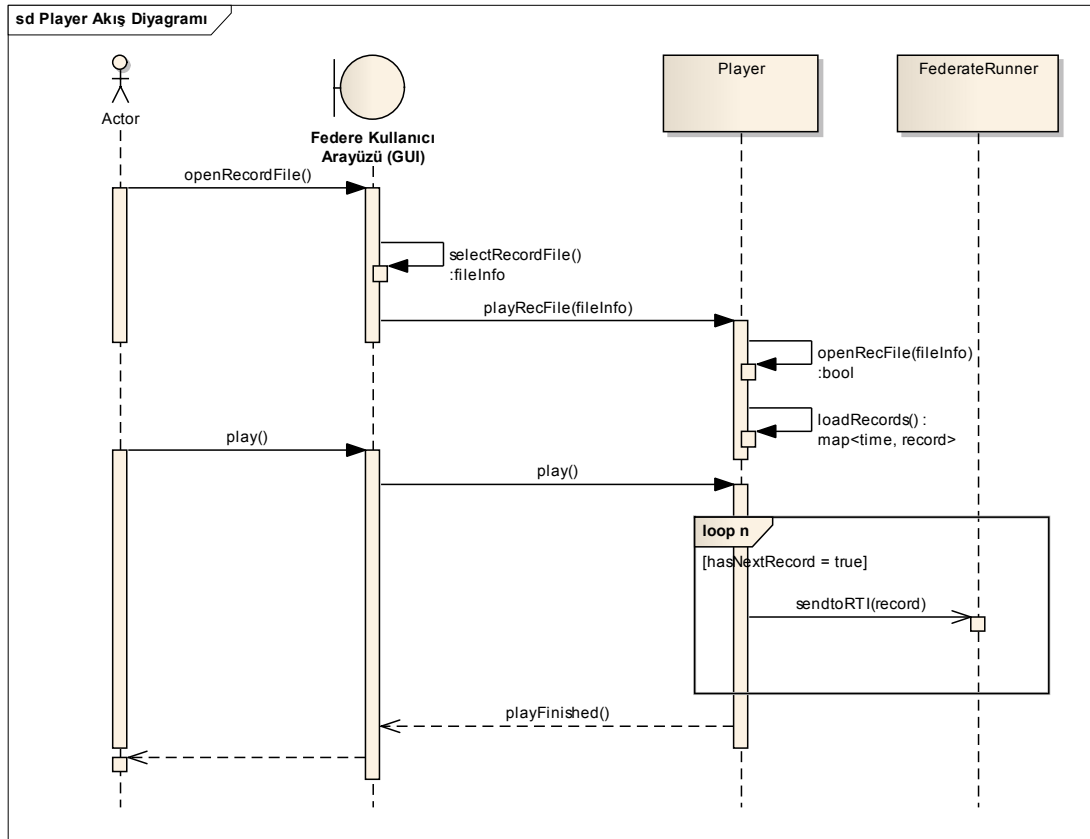
Koşum kayıt dosyasından okunan kayıtlar, tekrar oynatma işleminin başlamasıyla birlikte zaman sıralı olarak federasyonda yayınlanır.

Federe etkileşim ve nesne mesajı yayınlama işlemini RTI Elçisi (RTIambassador) sınıfının fonksiyonlar aracılığıyla gerçekleştirir.

- sendInteraction : Federasyona etkileşim mesajı yayınlama fonksiyonudur.
- registerObjectInstance : Federasyonda nesne oluşturma fonksiyonudur.
- updateAttributeValues: Federasyonda daha önce oluşturulmuş nesneyi güncelleme fonksiyonudur.
- deleteObjectInstance: Federasyonda daha önce oluşturulmuş nesneyi silme fonksiyonudur.

TickInterval parametresi ile tekrar oynatma işlemi 200 ms.'lik zaman dilimlerine ayrılmıştır. Oynatma sırasında, 200 ms.'lik zaman aralıklarında, o zaman dilimindeki etkileşim ve nesne mesajları RTI'a gönderilir.

TickInterval aralığı artırılıp azaltılarak kayıtların RTI'a gönderilme hassasiyeti de artırılıp azaltılabilir. Küçük TickInterval değeri kayıtların RTI'da yayınlanma zamanının sadakatini artırır. Koşum tekrar oynatım akış diyagramı Şekil 4.17'de gösterilmiştir.



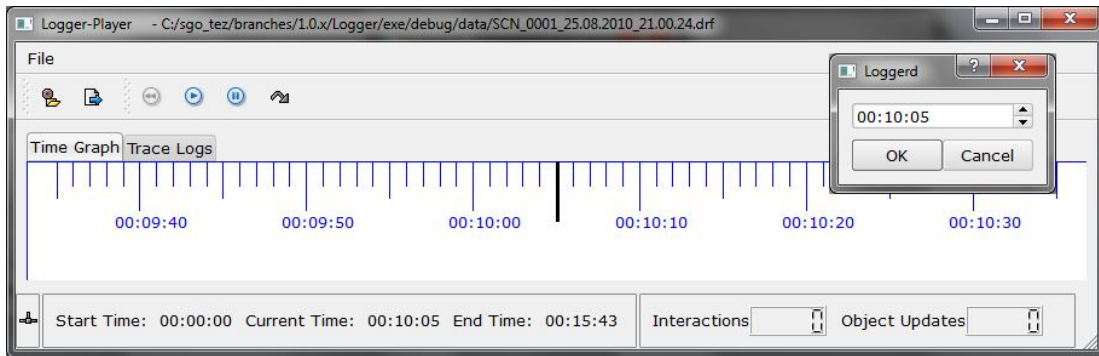
Şekil 4.17 : Koşum tekrar oynatım akış diyagramı

4.7.3 Koşum tekrar oynatımının başlatılması / bekletilmesi

Özellikle askeri eğitim amaçlı benzetim sistemlerinde eğitim amaçlı savaş senaryolarının koşum sonunda değerlendirme için tekrar oynatımı yapılır. Bu gibi eğitim amaçlı tekrar oynatım durumlarında, senaryonun belirli bir noktada bekletilmesi istenmektedir. Uygulama, oynatım sırasında bekletme / devam etme özelliklerini destekleyecek şekilde geliştirme yapılmıştır. Kullanıcı arabiriminden bu işlem kolaylıkla gerçekleştirilebilir.

4.7.4 Koşum tekrar oynatımının istenilen zamana ilerletilmesi

Özellikle askeri eğitim amaçlı benzetim sistemlerinde eğitim sırasında senaryonun tekrar oynatım yapılarak senaryonun değerlendirmesi yapılır. Bu gibi eğitim amaçlı tekrar oynatım durumlarında, senaryonun belirli bir noktada bekletilmesi istenmektedir. Bu amaçla tekrar oynatım birimi koşum kaydının tekrar oynatımı sırasında bekletme ve devam etme özellikleri geliştirilmiştir.



Şekil 4.18 : Tekrar oynatımda koşumun istenilen zamana alınması

4.8 Veri Tabanı Aktarım Birimi (Export Module)

Koşum kaydı sonucunda oluşturulan dosyanın içeriğinin ikili biçimde olması nedeniyle insanlar tarafından okunması ve anlaşılması zordur. Dosyanın ikili biçimde olması analiz yapmayı da oldukça zorlaştıran bir etkidir. Koşum kayıt dosyasında analiz ve sorgulama yapabilmek için en uygun yöntem, kayıt verilerinin ilişkisel veri tabanı sistemine aktarılmasıdır.

Koşum esnasında RTI'dan alınan etkileşim mesajları ve nesne güncellemelerinin çözümlenerek (decode) doğrudan veri tabanına aktarılması koşum kayıt performansını olumsuz etkileyecektir. Bunun için kayıtların veri tabanına aktarılması koşum kayıt işlemi bittikten sonra kullanıcının kendi tercihi bırakılmıştır. Bununla ilgili olarak “Koşum Verisi Saklama” bölümünde “Yöntem II” esas alınarak geliştirme yapıldığı belirtilmiştir.

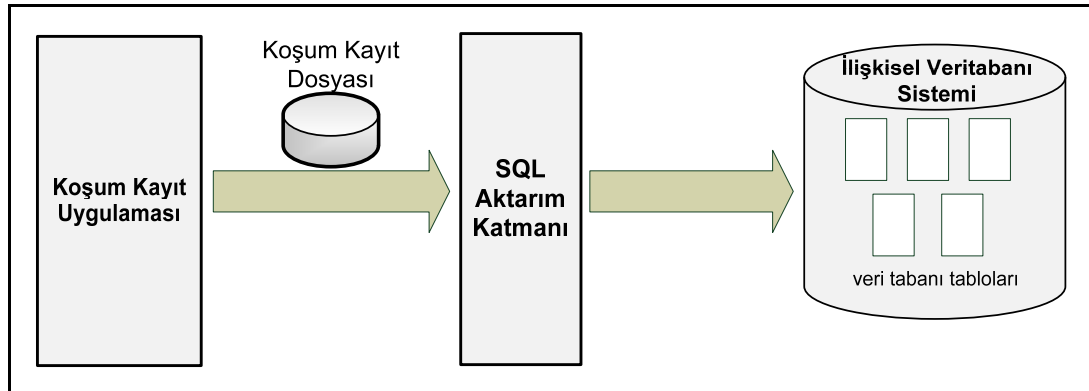
Kod üretme teknikleri kullanılarak kayıt dosyasında bulunan verinin çözümlenmesi ve ilişkisel veri tabanı sistemine aktarılması işlemleri bu bölümde anlatılacaktır.

İlişkisel veri tabanı sistemi için “Microsoft SQL Server 2005” temel alınarak geliştirme yapılmıştır.

Koşum kaydının veri tabanına aktarım işlemi temel olarak iki adımdan oluşur:

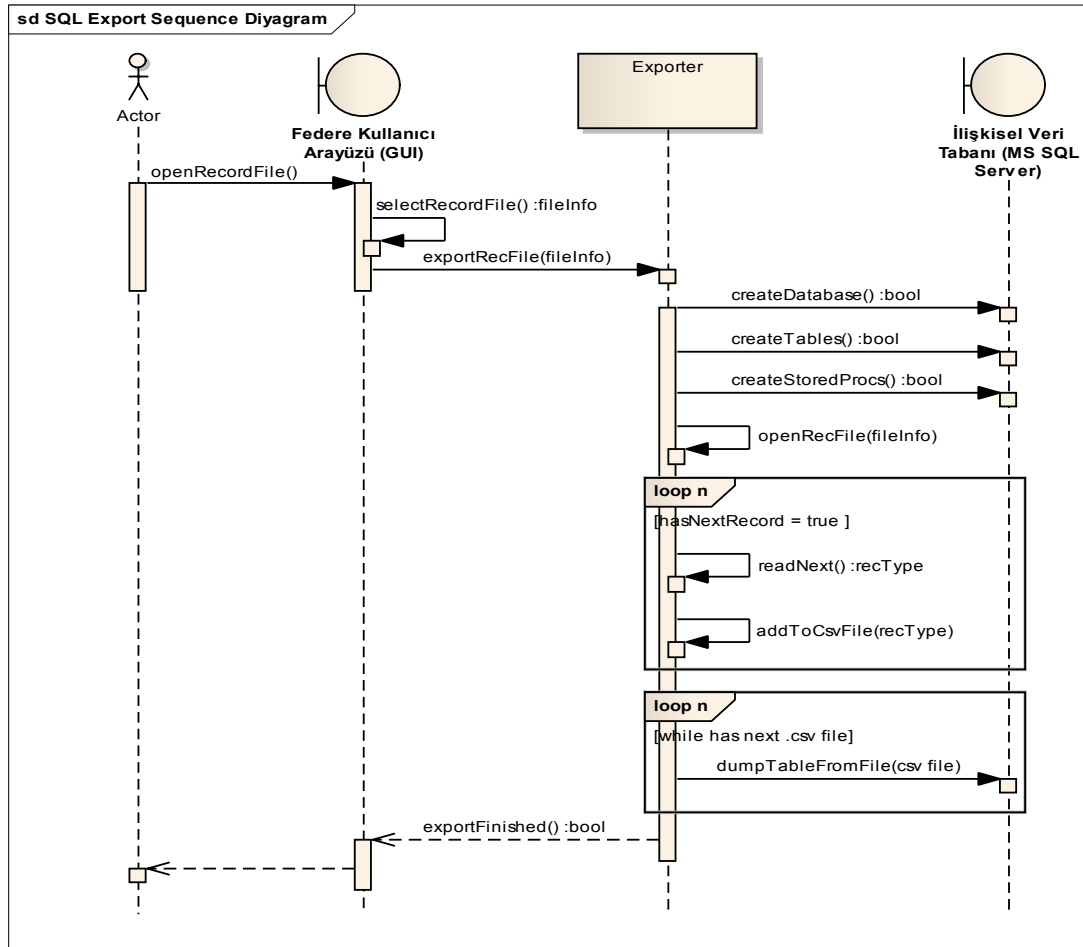
- Veri tabanı yapısının, tabloların ve saklı yordamların oluşturulması.
- Koşum kayıt dosyasındaki kayıtların çözümlenerek veri tabanındaki uygun tablolara aktarılması.

Koşum kaydının veri tabanına aktarım işlemi ile ilgili blok diyagram Şekil 4.19'da gösterilmiştir.



Şekil 4.19 : Veri tabanı aktarım işlemi blok diyagramı

Veri tabanı aktarım birimi akış diyagramı Şekil 4.20’de gösterilmiştir.



Şekil 4.20 : Veri tabanı aktarım birimi akış diyagramı

4.8.1 SQL komut dosyaları ile veritabanı yapısının oluşturulması

İlişkisel veri tabanı sistemlerinde, veri tabanında kayıt saklayabilmek için sözkonusu veri tabanının yapılandırılmış olması şarttır. Veri tabanı istemcisi uygulama, veri tabanına bağlantı kurarak okuma ve yazma işlemleri yapabilir. Veri tabanı örnek bağlantı cümlesi aşağıda gösterilmiştir:

```
DRIVER={SQL Server};SERVER={compeng-02};DATABASE={EXERCISE_01};
Trusted_Connection=yes;
```

Yukarıdaki bağlantı cümlesi, “EXERCISE_01” isimli veri tabanına bağlantı kurmayı sağlar. Kurulan bağlantı aracılığıyla veri tabanında bulunan tablolarda kayıt okuma, ekleme, silme, güncelleme işlemleri yapabilir.

Veri tabanı yapılandırma işlemi iki farklı yöntemle yapılabilir:

- Veri tabanı yönetim arayüzleri aracılığıyla elle yapılandırma.
- Yapılandırılmış Sorgu Dili (Structured Query Language - SQL) komut dosyaları ile otomatik yapılandırma.

Veri tabanı yönetim arayüzleri aracılığıyla veri tabanı oluşturma ve yapılandırma işlemi kullanıcı müdahalesi gerektiren bir işlemdir. Uygulamada kullanılacak veri tabanının başka bir sunucuya geçiş yapılması durumunda, diğer sunucuda tekrar elle yapılandırma gerekecektir. Bu gibi durumlarda her seferinde veri tabanı uzmanının ilgili sunucuda yapılandırma yapması gibi bir bağımlılık ortaya çıkar. Bu nedenle SQL komut dosyaları aracılığı ile yapılandırma yöntemi tercih edilmiştir. Bu yöntemin en büyük avantajı kullanıcı müdahalesi olmadan veri tabanının otomatik oluşturulmasıdır.

SQL komut dosyası içerisinde bir veya daha fazla SQL komutunu içerir. Komut dosyası içeriğinin tümü veri tabanı sunucusu üzerinde çalıştırılarak komutlar toplu olarak işletilmiş olur. Bu yöntemin avantajı, veri tabanının sunucusunun tek bir seferde birden fazla komutu toplu (batch) olarak işlemesidir.

Koşum kayıt dosyasında bulunan kayıtların ilişkisel veritabanı sistemine aktarılabilmesi için öncelikle veri tabanı yapısının oluşturulması gerekir. Veri tabanı sunucusunda koşum kaydı veri tabanının fiziki olarak oluşturulmasından sonra, veri tabanı yapılandırma işlemi yapılır.

Veri tabanının SQL komut dosyaları ile yapılandırılması üç adımdan oluşur:

1. Veri Tabanının fiziki olarak oluşturulması.
2. Veri tabanı tablolarının oluşturulması.
3. Saklı yordamların (Stored Procedure) kaydedilmesi.

4.8.1.1 Veri tabanının fiziki olarak oluşturulması

Veri tabanı istemcisi bir uygulamanın veri tabanı üzerinde işlem yapabilmesi için öncelikle ilgili veri tabanına bağlantı kurması gerekir. Bunun için ilgili veri tabanının sunucuda fiziki olarak oluşturulmuş olması ön şarttır. Veri tabanına bağlantı

sağlandıktan sonra bu veri tabanı üzerinden SQL komutları çalıştırılabilir. SQL komutlarına örnek olarak, yeni tablo oluşturulması, var olan tablonun silinmesi, veri tabanında bulunan tablolarda kayıt okuma ve yazma işlemi yapılması sayılabilir. Çizelge 4.28’de koşum kaydı veri tabanını oluşturan sql komut dosyası içeriği detaylar ile görülmektedir.

Çizelge 4.28 : Veri tabanı oluşturma sql komut dosyası içeriği.

```
USE [MASTER]
GO

IF EXISTS (SELECT name FROM sys.databases WHERE name = N'EXERCISE_DEV_00_01')
DROP DATABASE [EXERCISE_DEV_00_01]

CREATE DATABASE [EXERCISE_DEV_00_01] ON PRIMARY
( NAME = N'EXERCISE_DEV_00_01', FILENAME = N'C:\Program Files (x86)\Microsoft
SQL Server\MSSQL.1\MSSQL\data\EXERCISE_DEV_00_01.mdf' , SIZE = 3072KB ,
MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
LOG ON
( NAME = N'EXERCISE_DEV_00_01_log', FILENAME = N'C:\Program Files
(x86)\Microsoft SQL Server\MSSQL.1\MSSQL\data\EXERCISE_DEV_00_01_log.ldf' ,
SIZE = 1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO
EXEC dbo.sp_dbcmptlevel @dbname=N'EXERCISE_DEV_00_01', @new_cmptlevel=90
GO
IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [EXERCISE_DEV_00_01].[dbo].[sp_fulltext_database] @action = 'disable'
end
GO
```

4.8.1.2 Veri tabanı tablolarının oluşturulması

Geliştirilen uygulamanın federe SOM dosyasında tanımlı her etkileşim ve nesne sınıfının koşum sırasında kaydının yapılabildiği önceki bölümlerde anlatılmıştır. Veri tabanına aktarım işleminde, her etkileşim ve nesne sınıfına karşılık gelen bir veritabanı tablosu düşünülmüştür. Koşum kayıt dosyasındaki her etkileşim ve nesne güncellemesi, ait olduğu sınıfa karşılık gelen veri tabanı tablosuna aktarılır. Tablo isimleri “tbl_<SınıfAdı>” formatındadır. Örneğin SOM’da “SurfaceVessel” adıyla tanımlı nesne sınıfı için karşılık geldiği veri tabanı tablosu “tbl_SurfaceVessel” dır.

Etkileşim sınıfı parametreleri ve nesne sınıfı öznelikleri tabloda birer kolon olarak temsil edilir. Şekil 4.21’de “tbl_SurfaceVessel” veri tabanı tablosunun kolonları ile birlikte yapısı gösterilmiştir.

	Column Name	Data Type	Allow Nulls
	EntityIdentifier	varchar(50)	☒
	EntityType	varchar(50)	☒
	WorldLocationX	float	☒
	WorldLocationY	float	☒
	WorldLocationZ	float	☒
	OrientationPsi	float	☒
	OrientationTheta	float	☒
	OrientationPhi	float	☒
	RecTimeStamp	int	☒
	RecSimTimeStamp	int	☒
			☐

Şekil 4.21 : Surfacevessel nesne sınıfı veri tabanı tablo yapısı

Tüm etkileşim ve nesne sınıfları için karşılık gelen tabloları oluşturan komutlar tek bir sql komut dosyasında toplanmıştır. Böylelikle tek seferde bütün tablolar toplu olarak oluşturma imkanı elde edilmiştir. Çizelge 4.29’da SurfaceVessel nesne sınıfı için veri tabanı tablosu oluşturma komutu detayı gösterilmiştir.

Çizelge 4.29 : Veri tabanı tablosu oluşturma komutu.

```

IF EXISTS (SELECT * FROM dbo.SYSOBJECTS WHERE id =
object_id('tbl_SurfaceVessel') AND OBJECTPROPERTY(id, 'IsUserTable') = 1)
DROP TABLE tbl_SurfaceVessel;

CREATE TABLE tbl_SurfaceVessel (
    EntityIdentifier varchar(50) NULL,
    EntityType varchar(50) NULL,
    WorldLocationX float NULL,
    WorldLocationY float NULL,
    WorldLocationZ float NULL,
    OrientationPsi float NULL,
    OrientationTheta float NULL,
    OrientationPhi float NULL,
    RecTimeStamp int NULL,
    RecSimTimeStamp int NULL
);

```

FOM’da tanımlı sabit uzunluklu veri yapıları için veri tabanı tablosunda farklı kolonlara karşılık gelecek biçimde bir yapı düşünülmüştür. Aşağıda Çizelge 4.30’da WorldLocationStruct ve OrientationStruct sabit uzunluklu FOM veri yapıları gösterilmiştir.

Çizelge 4.30 : Sabit uzunluklu FOM veri yapıları.

```
<fixedRecordData name="WorldLocationStruct">
  <field name="X" dataType="Double1"/>
  <field name="Y" dataType="Double1"/>
  <field name="Z" dataType="Double1"/>
</fixedRecordData>

<fixedRecordData name="OrientationStruct">
  <field name="Psi" dataType="Float1"/>
  <field name="Theta" dataType="Float1"/>
  <field name="Phi" dataType="Float1"/>
</fixedRecordData>
```

WorldLocation veri yapısı üç farklı X, Y, Z değerlerinden oluştuğu yukarıda gösterilmiştir. Aktarım işleminde, katman sınıfları tarafından WorldLocationStruct özniteliğini çözümlenerek tabloda ayrı kolonlara aktarılır. Yukarıda SurfaceVessel nesne sınıfına karşılık gelen tabloda “WorldLocationX”, “WorldLocationY” ve “WorldLocationZ” adında üç farklı kolon görülmektedir. Aynı işlem “OrientationStruct” veri yapısı için de uygulanmıştır.

4.8.1.3 Saklı yordamların kaydedilmesi

Saklı Yordamlar, SQL Server üzerinde bulunan belli işlemleri gerçekleştirmek için oluşturulmuş ve derlenmiş T-SQL ifadelerdir. Standart SQL’de döngü ifadeleri, şart ifadeleri gibi programlanabilme özelliğine sahip terimler yoktur. Fakat T-SQL gibi özelleştirilmiş araçlar bu özelliklere sahiptir. Bu yapılar veritabanında ilk çalıştırlarında derlenirler ve sonraki kullanımlarında bir daha derlenmedikleri için hızlı çalışırlar ve performans artırır.

Oluşturulan veya var olan saklı yordamlar kullanıldığında birçok avantaj elde edilir. Bunlardan birkaçı dolayısıyla saklı yordam kullanım sebepleri aşağıdaki gibidir:

- Saklı yordamlar, SQL Server 'a esneklik ve hız kazandırır.
- Önceden derlenmiş olduğu için, normal kullanılan bir SQL sorgusunun tekrar tekrar çalıştırılmasına oranla daha fazla performans elde edilmesine sağlarlar.
- Bir kez yazılıp, tekrar tekrar kullanıldığı için modüler bir yapıda program geliştirilmiş olur.
- Aynı Transact-SQL cümleciğini birden fazla yerde kullanılacağı zaman, bunu bir saklı yordam haline getirerek, kullanımını sadece ismini çağırma ile gerçekleştirilebilir.

- Belirli girdi ve çıktı parametreleri olduğu için, saklı yordamların kullanımı ile güvenlik açısından yöneticiyi de sağlama almış olur.
- Ağ trafiğini azaltır. İstemci tarafından birçok satıra sahip SQL komutunun sunucuya gitmesindense, sadece saklı yordamın adının sunucuya gitmesi ağı daha az meşgul etmiş olur.

Saklı yordamların oluşturulması işlemi de koşum kayıt veri tabanının yapılandırması kapsamında değerlendirilmiştir. Çizelge 4.31’de örnek saklı yordam oluşturulması gösterilmiştir. Koşum sonu yapılacak analiz işlemleri için gerekli olabilecek saklı yordamlar komut dosyasında tanımlanarak tek seferde tüm saklı yordamların veri tabanında oluşturulması sağlanır.

Çizelge 4.31 : Saklı yordam kaydetme.

```
CREATE PROCEDURE [dbo].[getSubmersibleInfo]
@minTime int,
@maxTime int,
@eID varchar(50)
AS
SELECT (CONVERT(CHAR(8), DATEADD(millisecond, RecSimTimeStamp, ''), 108) + '.'
+ CAST((RecSimTimeStamp % 1000) AS CHAR(3)))
AS SimTime, WorldLocationX, WorldLocationY, WorldLocationZ
FROM tbl_SubmersibleVessel
WHERE (RecSimTimeStamp BETWEEN @minTime AND @maxTime)
AND (EntityIdentifier = @eID)
```

Oluşturulan saklı yordamların analiz işlemleri sırasında kullanımı oldukça basittir. Kullanıcının çok detaylı SQL kullanımı bilmesine gerek kalmadan saklı yordam çağırımı basit bir şekilde yapılabilir. Oluşturulan “getSubmersibleInfo” saklı yordamı çağırımı ve veri tabanından alınan sonuç aşağıda gösterilmiştir. Alınan sonuç ‘1:3001:105’ numaralı denizaltının 100.000 – 200.000 benzetim zamanı (milisaniye düzeyinde hassasiyet için) aralığında hangi konumlarda bulunduğunu göstermektedir.

The screenshot shows a SQL query execution window with two tabs: 'SQLQuery5.sql*' and 'SQLQuery1.sql*'. The active tab 'SQLQuery1.sql*' contains a single line of SQL code: `EXEC [dbo].[getSubmersibleInfo] 100000, 200000, '1:3001:105'`. Below the query editor, there are two tabs: 'Results' and 'Messages'. The 'Results' tab is active, displaying a table with 7 rows and 4 columns: 'SimTime', 'WorldLocationX', 'WorldLocationY', and 'WorldLocationZ'.

	SimTime	WorldLocationX	WorldLocationY	WorldLocationZ
1	00:01:47.303	3139611	5440381,5	1103007,25
2	00:01:56.964	3139553,25	5440414	1103011,25
3	00:02:00.835	3139533,25	5440425,5	1103012,625
4	00:02:02.505	3139526,25	5440429,5	1103013,125
5	00:02:06.414	3139513,25	5440437	1103014,125
6	00:02:12.749	3139500,5	5440444	1103014,875
7	00:02:25.344	3139494,5	5440447,5	1103015,25

Şekil 4.22 : Saklı yordam çalıştırma ve sonuçların alınması

Yukarıda anlatılan üç adımda veri tabanının yapısal olarak oluşturulması işlemi tamamlanır. Bundan sonra yapılan işlem koşum kayıt dosyasında kaydedilmiş olarak bulunan etkileşim mesajları ve nesne güncellemelerinin veri tabanında ilgili tablolara aktarılması işlemidir. Koşum kayıtlarının dosyadan okunarak veri tabanı tablolarına aktarımı kod üretim teknikleri ile üretilmiş sınıflar vasıtasıyla gerçekleştirilir.

4.8.2 Kod üretim teknikleri

Kod üretme tekniklerinin yazılım mühendisliği açısından birçok avantajı bulunmaktadır. Bu avantajlar bir kaç aşağıda listelenmiştir:

Kalite : Araştırmalara göre büyük ölçekli yazılım projelerinde geliştiriciler tarafından yazılan kodlarda hata olma olasılığı kodun büyüklüğüyle doğru orantılı şekilde artmaktadır. Şablonlar üzerinden üretilen kodlarda tutarlılığın artması ile birlikte hata olma olasılığı azalmaktadır. Hata düzeltme işleminde şablon üzerinde düzeltmeler yapılarak değişiklik tüm kodlara aynı şekilde yansıtılabilmektedir.

Tutarlılık : Kod üretici tarafından üretilen kod programlama arayüzüne uygun olarak üretilerek, tüm kodların aynı kodlama standardında olması sağlanır. Üretilen kodların geliştiriciler tarafından anlaşılması ve kullanımı daha kolay olacaktır.

Bilginin Tek Noktadan Yönetimi : Üretilen kodda elle yapılması gereken bir değişiklik söz konusu olduğunda birçok noktada değişiklik yapma ihtiyacı ortaya

ıkabilir. Koda ilişkin dkmanların hazırlanmış olması durumunda dkmanların da aynı şekilde teker teker gncellenmesi gerekebilecektir. rneğın bir tablo isminin deėiřmesi birok kaynak kod dosyasında da tablo isminin deėiřmesini gerektirecektir. Kod reteci ile tek bir noktadan belirlenen tablo ismi ile, kodun tekrar retilmesi saėlanarak olası hataların ve zaman kaybının nne geilmiř olur.

Tasarıma Daha Fazla Zaman Ayırma : Elle kodlama yapılan projelerde mimari tasarımda oluřan hatalardan veya ngrlmeyen durumlardan dolayı tm kodlarda deėiřiklik mimari deėiřikliėe uėrama olabileceėi iin proje takviminde sapmalar yařanabilir. Kod retimi ile mimari tasarımda olabilecek deėiřiklikler neticesinde řablon kod zerinde ilgili deėiřiklikler yapılarak yeni kodlar retilir. Bu durumda retilen tm kodlar yeni mimariye uygun olacaktır. Kod retimi kazanılan zaman sayesinde tasarıma daha fazla zaman ayrılabilir.

4.8.2.1 Kod üretiminde programlama dilleri karşılaştırımı

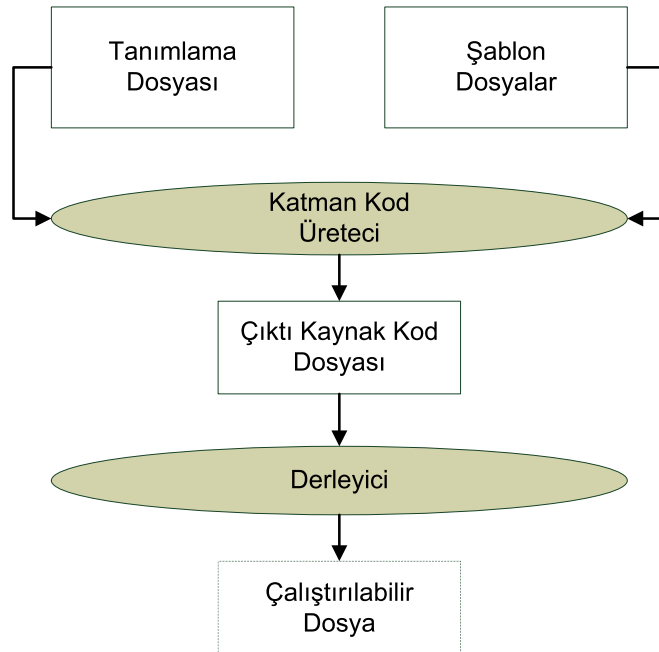
Programlama Dili	Avantaj	Dezavantaj
C ve C++	▪ Ticari Kod Üreteçleri için, müşterilerin kaynak kodlara erişimi ve müdahale etmesi mümkün değildir.	▪ Güçlü tip sistemi metin ayrıştırma uygulamaları için ideal değildir. ▪ Çalışma zamanı performansı kod üretimi için artı özellik katmaz. ▪ Dosya sistemi (I/O) yapısı farklı işletim platformları arasında taşınabilir değildir. ▪ XML ve düzenli deyim araçlarının kullanımı zordur.
Java	▪ Ticari Kod Üreteçleri için, müşterilerin kaynak kodlara erişimi ve müdahale etmesi mümkün değildir. ▪ Farklı işletim sistemlerinde çalışma imkanı sunar. ▪ XML araçları mevcuttur. ▪ Şablon araçları mevcuttur.	▪ Java metin ayrıştırma için uygun değildir. ▪ Güçlü tip (Strong typing) sistemi metin işleme için uygun değildir. ▪ Küçük ölçekli kod üretimi için üreteç gerçekleştirme daha maliyetlidir.
Perl Ruby Python	▪ Metin ayrıştırma (Parse) işlemleri basittir. XML araçları mevcuttur. ▪ Düzenli deyim (regular expression) kullanımı. ▪ XML kütüphanesi kullanımı çok basittir. ▪ Şablon araçları mevcuttur	▪ Yaygın dil olmadığı için kod üreticiye kod müdahalesi gerektiğinde diğer mühendislerin dili öğrenmesi gereksinimi doğacaktır.

Kod üretim teknikleri temel olarak “aktif kod üretimi” ve “pasif kod üretimi” olmak üzere iki kategoriye ayrılır. Pasif kod üretiminde, kod üretici tarafından oluşturulan koda uygulama geliştirici tarafından değişiklik yapılabilir. Entegre geliştirme ortamlarında (Integrated Development Environments – IDE) sihirbazlar aracılığıyla oluşturulan kaynak kodlar pasif kod üretimine örnektir.

Aktif kod üretiminde üretilen kod değiştirilmeden doğrudan derleyiciye girdi olarak verilir. Kodda değişiklik yapılması gerektiğinde kod üreticiye girdi verilerek yeni kod üretmesi sağlanarak yeni üretilcek kodlar kullanır.

4.8.2.2 Katman kod üretim yöntemi

Kod üretici katmanlı yapıda geliştirilen uygulamanın belirli bir katmanının kod üretimini gerçekleştirir. Bu yöntem genel olarak model temelli (model-driven) üretim için kullanılır. Üretece girdi olarak tanımlama dosyası (genellikle XML biçimli) verilerek kod üretimi gerçekleştirilir. Katman kod üretici tanımlama dosyasını okuyarak şablondan çıktı sınıfları üretir. İşleyişi Şekil 4.23’de gösterilmiştir.



Şekil 4.23 : Katman kod üretici işleyişi

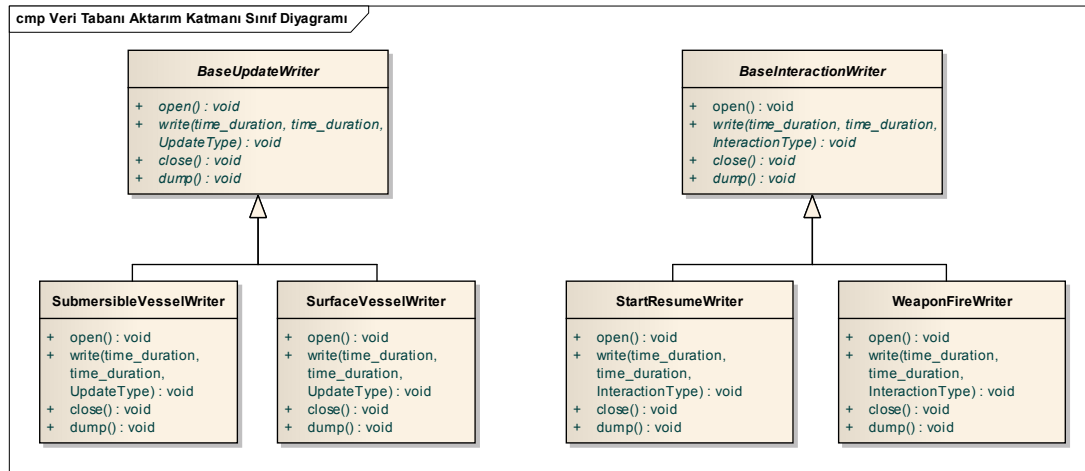
4.8.3 Kod üretiminin aktarım katmanı sınıflarına uygulanması

Tez çalışmasında geliştirilen uygulamanın aktarım katmanı sınıfları katman kod üretim yöntemiyle üretilmiştir.

Uygulamada kullanılan kod üretiminde temel amaç her etkileşim ve nesne sınıfı için birer “Yazdırma (Writer)” ve “Bilgi (Info)” sınıfı üretilmesini sağlamaktır. Kod üreticiye, nesne ve etkileşim sınıf bilgilerinin yer aldığı XML biçimli dosya girdi olarak verilir. Kod üretici çıktı olarak her etkileşim ve nesne sınıfı için “<SınıfAdı>Writer” ve “<SınıfAdı>Info” adında sınıflar üretilir. XML girdi dosyasında “SurfaceVessel” nesne sınıfı için üreteç tarafından “SurfaceVesselInfo” ve “SurfaceVesselWriter” sınıfları üretilir.

Yazdırma Sınıfları : Her etkileşim ve nesne sınıfı için karşılık gelen bir yazdırma sınıfı üretilir. Yazdırma sınıfının amacı, kendi FOM sınıfı ile ilgili koşum kayıt dosyasındaki kayıtların (etkileşim mesajı veya nesne güncellemesi) her birini zaman bilgisi ile birlikte karşılık geldiği veri tabanı tablosuna eklemektir. Tüm nesne sınıfları için ilgili yazdırma sınıfları “BaseUpdateWriter” soyut arayüz sınıfından türemişlerdir ve soyut metotların (pure virtual) gerçekleştirilmesi kod üreticinin oluşturduğu kaynak kod dosyalarında gerçekleştirilir.

Diğer taraftan tüm etkileşim sınıfları için karşılık gelen yazdırma sınıfları ise “BaseInteractionWriter” soyut arayüz sınıfından türemişlerdir ve ve soyut metotların gerçekleştirilmesi kod üreticinin oluşturduğu kaynak kod dosyalarında gerçekleştirilir.



Şekil 4.24 : Yazdırma sınıfları sınıf diyagramı

Yazdırma sınıfı üretimi için XML girdi dosyasında nesne sınıfı tanımlamalarının yapılması gerekmektedir. SurfaceVessel nesne sınıfı için örnek aşağıda gösterilmiştir.

```
<Object TableName="tbl_SurfaceVessel"
FullName="HLAobjectRoot.BaseEntity.PhysicalEntity.Platform.Surface
Vessel" ShortName="SurfaceVessel" >
  <Attribute Name="EntityIdentifier"
Type="EntityIdentifierStruct"/>
  <Attribute Name="EntityType" Type="EntityTypeStruct"/>
  <Attribute Name="WorldLocation" Type="WorldLocationStruct"/>
  <Attribute Name="Orientation" Type="OrientationStruct"/>
</Object>
```

TableName : Nesne sınıfının veri tabanında karşılık geldiği tablo adını temsil eder.

FullName : Nesne sınıfının FOM’da tanımlı tam adını temsil eder.

ShortName : Nesne sınıfı kısa adını temsil eder.

Attribute : Bu düğüm nesne sınıfını özneliklerini temsil eder. Name bilgisi öznelik adını, Type ise özneliğin FOM’da tanımlı veri tipini temsil eder.

WeaponFire etkileşim sınıfı için XML girdi dosyasında yapılan tanımlama aşağıda gösterilmiştir.

```
<Interaction TableName="tbl_WeaponFire"
FullName="HLAinteractionRoot.WeaponFire" ShortName="WeaponFire">
  <Parameter Name="FiringObjectIdentifier"
Type="RTIObjectIdStruct"/>
</Interaction>
```

TableName : Etkileşim sınıfının veri tabanında karşılık geldiği tablo adını temsil eder.

FullName : Etkileşim sınıfının FOM’da tanımlı tam adını temsil eder.

ShortName : Etkileşim sınıfı kısa adını temsil eder.

Parameter: Bu düğüm etkileşim sınıfı parametresini temsil eder. “Name” bilgisi parametre adını, “Type” ise parametrenin FOM’da tanımlı veri tipini temsil eder.

Kod üreticiye dört farklı kod şablonu girdi olarak verilir. Nesne Yazdırma sınıf başlık dosyası (header file) şablonu, Nesne Yazdırma sınıf kaynak kod (source code) şablonu, Etkileşim Yazdırma sınıf başlık kodu şablonu, Etkileşim Yazdırma sınıf

kaynak kod şablonu. Aşağıda Çizelge 4.32’de Nesne Yazdırma sınıf başlık şablonu gösterilmiştir.

Çizelge 4.32 : Nesne yazdırma sınıf başlık üretim şablonu.

```
class <%=short_name%>Writer : public BaseUpdateWriter {
    typedef void(<%=short_name%>Writer::*fn) (const
VariableLengthData&);
public:
    <%=short_name%>Writer();
    ~<%=short_name%>Writer();
    static bool initializeMapper();
    virtual void open();
    virtual void write(bp::time_duration recTime,bp::time_duration
        recSimTime,UpdateType update);
    virtual void close();
    virtual void dump();
    virtual std::vector<std::pair<std::string,std::string> >
        fileTableNames();
    <% attributes.each { |attr| %>
        void export<%=attr.name%>(const VariableLengthData& val);<% } %>
private:
    static std::map<std::string,fn> mAttributeMapper;
    <%=short_name%>Info mRow;
};
```

“SurfaceVessel” nesne sınıfı için kod üretici tarafından oluşturulan başlık dosyası içeriği aşağıda Çizelge 4.33’de gösterilmiştir.

Çizelge 4.33 : Kod üretici tarafından üretilen başlık dosyası.

```
class SurfaceVesselWriter : public BaseUpdateWriter {
    typedef void(SurfaceVesselWriter::*fn) (const
VariableLengthData&);
public:
    SurfaceVesselWriter();
    ~SurfaceVesselWriter();
    static bool initializeMapper();
    virtual void open();
    virtual void write(bp::time_duration recTime,
        bp::time_duration recSimTime,UpdateType update);
    virtual void close();
    virtual void dump();
    virtual std::vector<std::pair<std::string,std::string> >
        fileTableNames();
    void exportEntityIdentifier(const VariableLengthData& val);
    void exportEntityType(const VariableLengthData& val);
    void exportWorldLocation(const VariableLengthData& val);
    void exportOrientation(const VariableLengthData& val);
private:
    static std::map<std::string,fn> mAttributeMapper;
    SurfaceVesselInfo mRow;
};
```

Bilgi Sınıfları : Her etkileşim ve nesne sınıfı için karşılık gelen bir “Bilgi Sınıfı (Info class)” üretilir. Bilgi sınıfının amacı, kendi FOM sınıfına karşılık gelen veri tabanı tablosu yapısının yapısal temsidir. Yazdırma sınıfları “Bilgi” sınıfları aracılığıyla yazma işlemini gerçekleştirir. Bilgi sınıflarının üretimi için XML girdi dosyasında, nesne ve etkileşim sınıflarına karşılık gelen veri tabanı tablo yapıları ile eşleşmeyi sağlayacak tanımlamalar yapılır. Etkileşim ve nesne sınıfları için tanımlamada “Yazdırma” sınıflarında olduğu gibi bir ayrım söz konusu değildir. Bilgi sınıf üretici XML girdi dosyasında “SurfaceVessel” nesne sınıfı için gerekli tanımlama Çizelge 4.34’de gösterilmiştir.

Çizelge 4.34 : Bilgi sınıf üretici xml girdisi.

```
<Table Name="tbl_SurfaceVessel" ClassName="SurfaceVessel">
  <Column Name="EntityIdentifier" Order="0" Type="string"/>
  <Column Name="EntityType" Order="1" Type="string"/>
  <Column Name="WorldLocationX" Order="2" Type="float"/>
  <Column Name="WorldLocationY" Order="3" Type="float"/>
  <Column Name="WorldLocationZ" Order="4" Type="float"/>
  <Column Name="OrientationPsi" Order="5" Type="float"/>
  <Column Name="OrientationTheta" Order="6" Type="float"/>
  <Column Name="OrientationPhi" Order="7" Type="float"/>
  <Column Name="RecTimeStamp" Order="8" Type="int"/>
  <Column Name="RecSimTimeStamp" Order="9" Type="int"/>
</Table>
```

Kod üreticiye girdi olarak verilen sınıf başlık dosyası şablonu Çizelge 4.35’de gösterilmiştir.

Çizelge 4.35 : Bilgi sınıfı başlık dosyası üretim şablonu.

```
class <%=short_name%>Info {
public:
  <%=short_name%>Info();
  ~<%=short_name%>Info();

  <%=short_name%>Info(const <%=short_name%>Info&);
  <%=short_name%>Info& operator=(const <%=short_name%>Info&);

  void clear();
  void open();
  void write();
  void close();
  <% columns.each { |column| %>
  void write<%=column.name%>(<%=column.name%>);<%=column.name%> %>

  std::string tableName();
  std::string fileName();

  std::ofstream mFile;
  std::string mFilename;
  <% columns.each { |column| %>
  <%=column.type%> <%=column.name%>;
  bool m<%=column.name%>Updated;<%=column.name%> %>
};
```

4.8.3.1 Ruby programlama diliyle kod üretimi gerçekleştirilmesi

Ruby, sözdizimi Perl ve Smalltalk gibi dillerden ilham alınarak ortaya çıkmış dinamik, yansıtıcı, genel amaçlı bir programlama dilidir. Fonksiyonel ve nesne yönelimli de dahil olmak üzere birçok programlama paradigmasını destekleyen bir programlama dilidir.

Basitlik ve verimlilik odaklı dinamik ve açık kaynak programlama dilidir. Okumak ve yazmak kolaydır. Kod üretimi için gerekli şablon kullanımına uygun olması ve XML yapıları ile düzenli deyim kullanımının basitliği gibi nedenlerden dolayı kod üretimi için Ruby programlama dili seçilmiştir [15].

4.8.3.2 Veri tabanı yığın aktarım işlemi (bulk insert)

Koşum kayıt dosyasından okunan her kayıt kendi ait olduğu FOM sınıfına ilişkin Yazdırma sınıfı aracılığı ile virgül ayrımlı değerler (comma separated values – csv) dosyasına aktarılır. Her etkileşim ve nesne sınıfı için ayrı bir csv dosyası üretilir. Bir sınıfa ait tüm koşum kayıtları ilişkili csv dosyasına zaman sıralı olarak yazılır.

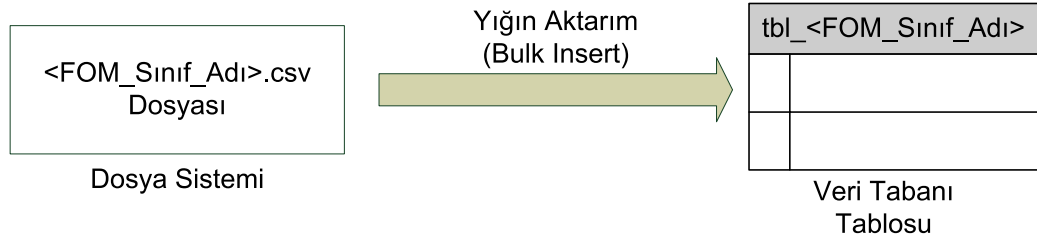
CSV Dosya Biçimi : Veritabanı tablo yapısı için basit bir metin biçimidir. Genellikle farklı bilgisayar programları arasında sekmeli veri taşımak için kullanılan basit bir dosya formatıdır. Örneğin, bir csv dosyası bir veritabanı programı ile hesap çizelgesi (spreadsheet) arasında bilgi aktarımı için kullanılabilir. Tablodaki her kayıt metin dosyasındaki bir satıra tekabül eder. Kayıt değerleri virgülle birbirinden ayrılmıştır.

SurfaceVesselWriter Yazdırma sınıfı tarafından oluşturulan “SurfaceVessel.csv” virgül ayrımlı dosyanın içeriğinden bir kesit Çizelge 4.36’da gösterilmiştir. Çizelgeden de görüldüğü üzere farklı nesne örneklerine (1:3001:19, 1:3001:20, 1:3001:21) ilişkin güncelleme mesajları aynı virgül ayrımlı dosyada zaman sıralı olarak yer almaktadır.

Çizelge 4.36 : SurfaceVessel.csv virgül ayrımlı dosya içeriği.

1:3001:19,1:3:225:11:1:0:0,3138906.5,5440928,1102726.5,2.465739965 44,0.710343003273, -1.80240595341,13581,13581
1:3001:20,1:3:225:11:1:0:0,3139033.75,5440879,1102607.125,2.461786 9854,0.722905993462,-1.80497705936,13584,13584
1:3001:21,1:3:225:11:1:0:0,3139128.5,5440853.5,1102464.375,2.46863 794327,0.700946986675,-1.80047702789,13589,13589
1:3001:102,1:3:222:6:0:0:0,3139014.25,5440780.5,1103143.875, -2.09409093857, -1.39579904079,-3.14159011841,14040,14040
1:3001:19,1:3:225:11:1:0:0,3138906.5,5440928,1102726.5,2.465739965 44,0.710343003273, -1.80240595341,14220,14220
1:3001:20,1:3:225:11:1:0:0,3139033.75,5440879,1102607.125,2.461786 9854,0.722905993462, -1.80497705936,14221,14221

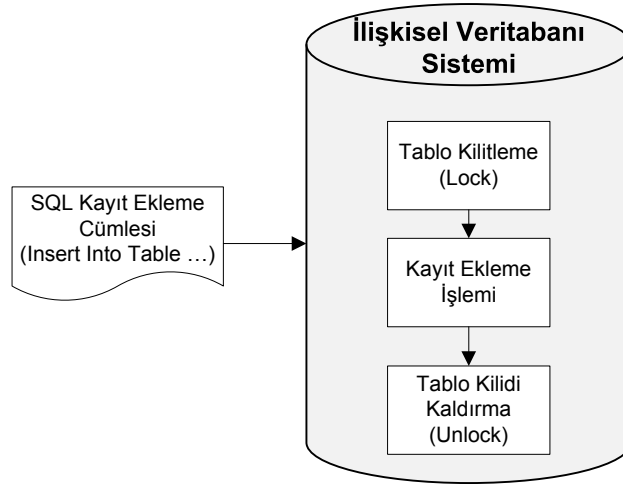
Koşum kayıtlarının csv dosyalarına aktarımı bittikten sonra her bir csv dosyası, karşılık geldiği veri tabanı tablosuna “yığın aktarım” komutu ile aktarılaraq tek adımda tüm kayıtların veri tabanı tablosuna aktarımı gerçekleştirilir.



Şekil 4.25 : Yığın aktarım işlemi blok diyagramı

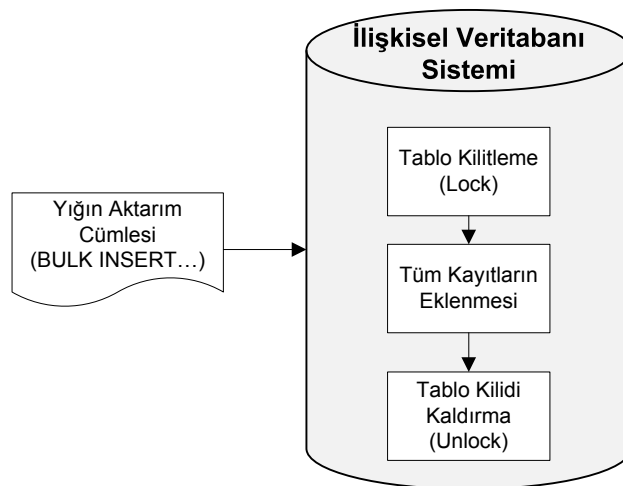
İlişkisel veri tabanı sistemlerinde veri tabanı tablosuna kayıt ekleme işleminde tablo yazma işlemi için kilitlenir, kayıt yazma bittikten sonra tablo kilidi kaldırılır. Kilitleme işlemi veri tabanı tutarlılığını sağlamaya yönelik olarak ilişkisel veri tabanı sistemi tarafından kullanıcı kontrolü dışında gerçekleşen bir işlemdir. Yığın aktarım işleminin avantajı, tüm kayıtlar tek seferde veri tabanı tablosuna aktarılaraq zamandan tasarruf sağlanır, tablo kilitleme ve kilit kaldırma bir kez yapılır.

Kayıtların veri tabanı tablosuna tek tek aktarımı ile ilgili blok diyagram Şekil 4.26’da gösterilmiştir. N tane kayıt için bu işlem N kez tekrarlanacaktır. N adet tablo kilit işlemine karşılık, N adet kilit kaldırma işlemi gerçekleşir.



Şekil 4.26 : Kayıtların veri tabanı tablosuna tek tek aktarım yaklaşımı

Kayıtların veri tabanı tablosuna yığın aktarımı ile ilgili blok diyagram Şekil 4.27’de gösterilmiştir. N tane kayıt için bu işlem tekrar bir adımda tamamlanır. Tek tablo kilit işlemi ve tek kilit kaldırma işlemi gerçekleşir. Bu yöntem büyük ölçekli koşum verilerinin veri tabanına aktarım zamanını oldukça azaltacak bir yaklaşımdır.



Şekil 4.27 : Kayıtların yığın aktarım ile tek adımda aktarım yaklaşımı

SurfaceVessel nesne sınıfına ilişkin kayıtların csv dosyasından veri tabanına aktarımı ile ilgili yığın aktarım komutu çizelge 4.37’de verilmiştir.

Çizelge 4.37 : Yığın aktarım T-SQL komutu.

```
BULK INSERT tbl_SurfaceVessel
FROM 'SurfaceVessel.csv'
WITH (FIELDTERMINATOR =',', ROWTERMINATOR ='\n')
```

Yazdırma sınıfları tarafından üretilen csv dosyalarının tümü yığın aktarım işlemi ile veri tabanı tablolarına aktarılır. Bu işlemle birlikte veri tabanı aktarım işlemleri son bulur.

Koşum analizi için gerekli tüm koşum bilgisi çözümlenmiş olarak veri tabanına aktarılmış olur. Analiz işlemleri için gerekli sorgular veri tabanında üzerinde çalıştırılabilir. İstenildiğinde veri tabanında bulunan saklı yordamlar ile cevaplar alınabilir. Saklı yordam çağırımı dışında, istenir SQL cümleleri ile doğrudan sorgulamalar da yapılabilecektir.

4.9 Koşum Analizi

Benzetim koşumu sırasında yapılacak analiz işlemlerinde, koşum henüz sonlanmadığından ve elde henüz yeterli derecede koşum bilgisi bulunamayabileceğinden analiz imkanları kısıtlı olacaktır. Eksik bilgi nedeniyle yanlış çıkarımlar yapılması söz konusu olabilmektedir. Koşum sırasında yapılacak analiz işlemi için kaydedilen bilgilerin anlık olarak ilişkisel veri tabanına aktarımı zorunlu kılmalıdır.

Koşum sırasında yapılacak analiz işlemi için koşum kayıt biriminin de buna uygun olarak tasarlanması gerekmektedir. Koşum kayıt birimi koşum sırasında aldığı her türlü etkileşim mesajı ve nesne güncellemesini anlık olarak ilişkisel veri tabanına kaydediyor olması gerekir. Bu yöntemin koşum kaydını yavaşlatması vb. dezavantajları önceki bölümlerde anlatılmıştır.

Koşum sonu yapılan analiz işlemleri literatürde “Koşum Sonu İnceleme (After Action Review – AAR)” olarak adlandırılır. Koşum sonu veri tabanında hazır olarak bulunan tüm benzetim verisi üzerinde sorgulama ve analiz işlemleri yapılarak koşum ile ilgili istenilen cevaplara ulaşılabilir.

Koşum kayıt birimi için yapılacak mimari tasarım, koşum sırasındaki veya sonundaki analiz ihtiyaçlarına doğrudan bağlıdır.

Analiz bakış açısından verinin bulunduğu 3 farklı nokta sözkonusudur :

- Koşum sırasında RTI üzerinde
- Koşum sonunda kayıt dosyasında (ara dosya vb.)
- Koşum sırasında doğrudan erişilebilen ilişkisel veri tabanında.

Analistler koşum esnasında ve sonunda koşum verisi üzerinde analiz yapmak için genellikle aynı arayüzleri (veri tabanı, dosya vb.) ve analiz araçlarını kullanır. Koşum sırasında RTI üzerindeki verinin analizi oldukça zor ve maliyetli bir işlemdir.

Koşum sırasında anlık veri analizinin gerekli olmadığı sistemler için, ilişkisel veritabanları ile entegre çalışan ticari koşum analiz araçları mevcuttur.

Koşum kayıt ve analiz yazılımı geliştiren mühendisler, projenin isterleri doğrultusunda (veritabanı, ara dosya , gerçek zamanlı) uygun tasarımı belirlemek durumundadır.

Analiz işlemi iki aşamadan oluşur: raporlama ve analiz. Raporlama aşaması veri üzerinde önceden tanımlı sorguların çalıştırılmasıdır. Sorguların çalıştırılması sonucu, tablo, grafik vb. çıktılarla en sık sorulan sorular hakkında cevaplar elde edilir. Bu safha “Ne oldu” sorusuna cevap aramaya yöneliktir.

Sorgulara örnek olarak:

- Kaç tane düşman tankı imha edildi?
- Koşumda kaç tane bileşen (entity) mevcut?

İyi bir analiz aracının anahtar unsurları şunlardır:

- Toplam farkındalık – yanıt süresini en aza indirmek için önceden hesaplanmış özet tablolardan yararlanmadır.
- Konu kesişimi oluşturabilme – Birden fazla tablo üzerinde sorgulama imkanı sunması anlamına gelir (mavi kuvvetler, kırmızı kuvvetler gibi). Bu özellik olmadığı takdirde, her tablodaki veri ayrı ayrı sorgulanarak sorgular sonucunda yerel işleminden geçirilmesi gerekir. Bu hem yazılım geliştirme süresini hem de sonuçların alınması için geçen zamanı artırır.

- SQL kısıtlamalarının üstesinden gelebilmeli – Analiz yapan kişinin programlama ve SQL bilgisinin olmadığı düşünülmemelidir.
- Çoklu Veri Kaynağı – analistler birden fazla veri kaynağından elde edilen bilgiyi tek bir raporda görmek isteyebilir. İyi bir analiz aracı farklı kaynaklardan veriyi tek bir raporda sunabilmelidir.
- Bütünleşme – Geliştiricilere başvurmaya gerek kalmadan sorgulama, raporlama ve analitik yeteneklere sahip olmalıdır. Veri üzerinde işlem yapma yeteneği olmalıdır. Ekranda oluşturulan grafikler farklı raporlama ve sunum araçları için değişik formatlara aktarılabilirdir.

4.9.1 Çıktıların analiz edilmesi

Bir benzetim projesinde tüm gereksinimler önceden bilinmiyor olabilir. Bazı gereksinimler vardır ki bunlar koşum sonunda ortaya çıkabilir. Esnek ve iyi tanımlanmış operasyonel veri deposu (operational data store - ODS) sayesinde her türlü benzetim verisine hızlı ve kolay erişim mümkün olacaktır.

Koşum verisi raporu elde edildiğinde, müşteri “Buna neden olan nedir?” türünden sorular sormaya başlayacaktır. Bu türden sorulara verilecek cevaplar eğitim amaçlı benzetim sistemleri için değerli bilgidir. Benzer örnekler :

- Federe amacına niçin ulaşamadı ?
- Gemideki hasar hangi coğrafi konumda gerçekleşti?
- En çok hasara hangi olay neden oldu?
- Karşı kuvvetler nehrin karşı tarafına ne zaman geçti ?

Bu aşama bilgi sistemleri analistlerin tarafından veri madenciliği (data mining) olarak tanımlanır. Veri tabanında depolanmış veri üzerinden çıkarsamalar yapılarak cevaplara ulaşılmaya çalışılır. Koşum sırasında analiz araçlarında tüm koşum verisi hazır olmadığı için çıkarsamalar yapmak mümkün olamayabilir.

Herhangi bir koşum kayıt sisteminin en önemli yönü büyük miktarlarda bilgiyi zamanında analiz etmektir. Benzetim verileri doğası gereği değişkenlik gösterir ve çok boyutludur. İki boyutlu tablo ve grafikler bu bilgiyi analiz etmek için tutarlı bir resim sunmayabilir. Veri tabanında bulunan bu türden bilgiyi üç boyutlu olarak

gösterebilen ticari yazılımlar mevcuttur. Bu ticari uygulamalar çok boyutlu verilerin verinin (özellikle coğrafi 3 boyutlu veriler) analizi ve sunumu için güçlüdür. Bu ticari uygulamalar, kaydedilen koşum verileri veri tabanında depolanmış olmasına karşın, belirli bir benzetim sistemi için tüm analiz gereksinimlerini karşılamayabilir. Bu nedenle farklı benzetim sistemleri kendi analiz ihtiyaçları doğrultusunda ihtiyaçlara cevap verebilen analiz uygulaması geliştirilmesi gerekebilmektedir.

4.9.2 Koşum analiz uygulaması

Tez çalışmasında koşum analiz işlemleri “koşum sonu analiz” çerçevesinde değerlendirilmiş ve “Koşum Analiz” uygulaması gerçekleştirilmiştir. Gerçeklenen uygulama, ilişkisel veri tabanına aktarımı gerçekleştirilen veriler üzerinde sorgulama yapma imkanı sunar.

Geliştirilen analiz uygulaması ile istenirse veri tabanında bulunan saklı yordam çağrımları yapılabileceği gibi, istenirse SQL cümleleri yazılarak uyarlanmış sorgular da çalıştırılabilir.

Saklı yordamlar büyük ve karmaşık sorgularda veri tabanından daha hızlı yanıt alınmasını sağlayacaktır. Bölüm 4.8’de anlatılan veri tabanı yapılandırılması kapsamında “Saklı Yordamların Oluşturulması” konusuna değinilmiştir. Saklı yordamlar veri tabanı yapılandırma safhasında “saklı yordam sql komut dosyası”nın çalıştırılması ile veri tabanına kaydedilirler. Saklı yordamlar kaydedildikten sonra veri tabanının bir parçası olurlar. İstemci uygulamaları veri tabanına bağlantı kurarak saklı yordamları çağırma işlemi yapabilir. Analiz gereksinimleri doğrultusunda, sql komut dosyasına yeni tasarlanan saklı yordamlar eklenebilir. Böylece koşum kaydı aktarım işleminde yeni saklı yordamların veri tabanında analiz için hazır olarak bulunması sağlanmış olur. Bu sayede, yukarıda bahsettiğimiz gibi, analiz için birtakım gereksinimler bilinmiyor olsa bile, süreç ilerledikçe yeni sorgular basitçe eklenebilecektir. Bu yöntemin en büyük avantajı kaynak kodlara müdahale etmeden, uygulamayı tekrar derlemeye gereksinim olmadan veri tabanının otomatik olarak yapılandırmayı sağlamasıdır.

Veri tabanının yapılandırma işlemleri sırasında kaydedilen örnek saklı yordam Çizelge 4.38’de gösterilmiştir.

Çizelge 4.38 : Saklı yordam kaydetme.

```
USE [EXERCISE_DEV_00_01]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[getSubmersibleInfo]
    @minTime int,
    @maxTime int,
    @eID varchar(50)
AS
SELECT
    (CONVERT(CHAR(8), DATEADD(millisecond, RecSimTimeStamp, ''),
    108)
    + '.' + CAST((RecSimTimeStamp % 1000) AS CHAR(3))) AS SimTime,
    STR (WorldLocationX,12,2) AS WorldLocationX,
    STR (WorldLocationY,12,2) AS WorldLocationY,
    STR (WorldLocationZ,12,2) AS WorldLocationZ,
    STR (VelocityX,12,5) AS VelocityX,
    STR (VelocityY,12,5) AS VelocityY,
    STR (VelocityZ,12,5) AS VelocityZ
FROM tbl_SubmersibleVessel
WHERE (RecSimTimeStamp BETWEEN @minTime AND @maxTime)
AND (EntityIdentifier = @eID)
```

Veri tabanında “getSubmersibleInfo” adıyla kaydedilmiş saklı yordam üç adet girdi parametresi ile çalıştırılabilir. Veri tabanında kaydedilmiş bu saklı yordamın amacı Kimlik bilgisi (@eID) verilen denizaltı örneğinin belirli iki zaman aralığındaki (@minTime - @maxTime) değerlerine erişim sağlamaktır. Koşum kayıt işlemi millisaniye hassasiyet düzeyinde yapıldığı için sorgulamalarda milisaniye düzeyinde yapılabilmektedir.

Yukarıdaki sorgunun veri tabanı uzmanı olmayan bir kişi tarafından hazırlanması oldukça zordur. Ancak veri tabanında kayıtlı bir saklı yordamın çalıştırılması oldukça basittir. Aşağıda Çizelge 4.39’da “getSubmersibleInfo” saklı yordamının çağrı ifadesi gösterilmiştir.

Çizelge 4.39 : Saklı yordam çağrısı.

```
EXEC [dbo].[getSubmersibleInfo] 20000, 40000, '1:3001:105'
```

Yukarıdaki saklı yordam çağrımında girdi değerleri sırasıyla :

- @minTime = 20000
- @maxTime = 40000
- @eID = '1:3001:105' olarak verilmiştir.

“GetSubmersibleInfo” saklı yordamının kořum analizi uygulamasında alıřtırılması ve alınan sonuç řekil 4.28’de gsterilmiřtir.



řekil 4.28 : Kořum analizi uygulamasında sorgulama yapma

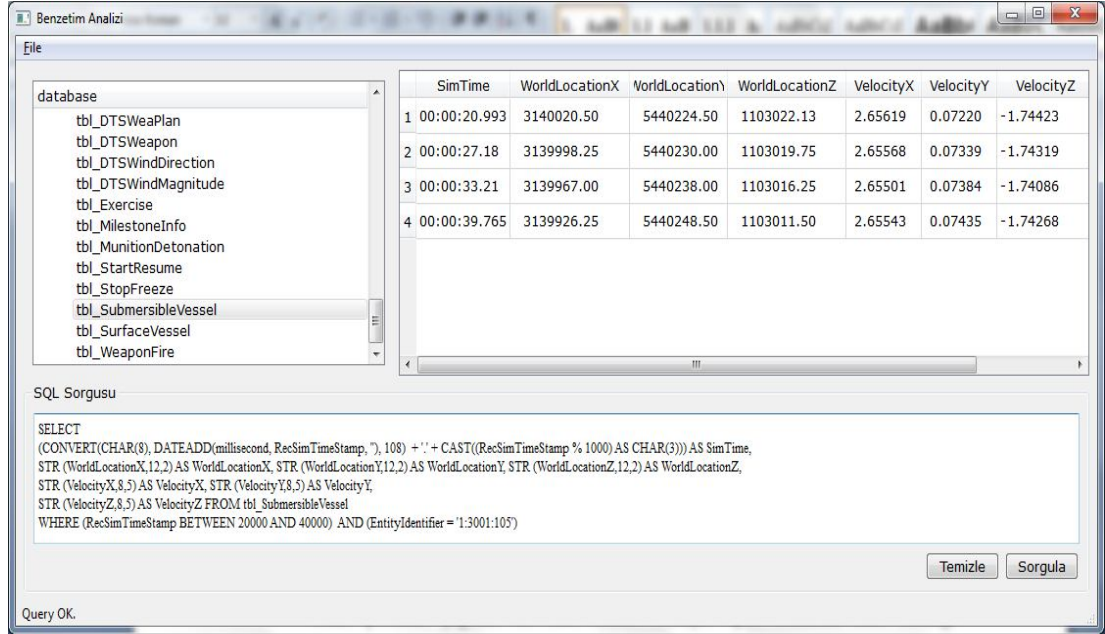
4.9.3 Analiz iřlemlerinde saklı yordam kullanım avantajları

Yukarıdaki bölümde saklı yordam aęrısı ile elde edilen sorgulama sonucu, istenirse tamamen Sql cmlelerinden kurulu komut cmlesi ile sorgulama yapılarak da elde edilebilir. Ařaęıda aynı sorgunun Sql cmleleri ile alıřtırılması iin gerekli Sql ifadesi verilmiřtir.

izelge 4.40 : Analiz iřlemlerinde sql ifadesi kullanımı.

```
SELECT
(CONVERT(CHAR(8), DATEADD(millisecond, RecSimTimeStamp, ''), 108)
+ '.' + CAST((RecSimTimeStamp % 1000) AS CHAR(3))) AS SimTime,
STR (WorldLocationX,12,2) AS WorldLocationX,
STR (WorldLocationY,12,2) AS WorldLocationY,
STR (WorldLocationZ,12,2) AS WorldLocationZ,
STR (VelocityX,8,5) AS VelocityX,
STR (VelocityY,8,5) AS VelocityY,
STR (VelocityZ,8,5) AS VelocityZ
FROM tbl_SubmersibleVessel
WHERE (RecSimTimeStamp BETWEEN 20000 AND 40000) AND
(EntityIdentifier = '1:3001:105')
```


Geliştirilen koşum analiz uygulaması yukarıdaki gibi Sql cümlelerini çalıştırabilecek şekilde tasarlanmıştır. Yukarıdaki Çizelge 4.40’da verilen Sql ifadesinin Koşum Analiz uygulamasında çalıştırılması ile elde edilen sonuç Şekil 4.29’da gösterilmiştir. Saklı yordam çağrısı ile elde edilen sonuçla aynı sonucun elde edildiği görülmektedir.



	SimTime	WorldLocationX	WorldLocationY	WorldLocationZ	VelocityX	VelocityY	VelocityZ
1	00:00:20.993	3140020.50	5440224.50	1103022.13	2.65619	0.07220	-1.74423
2	00:00:27.18	3139998.25	5440230.00	1103019.75	2.65568	0.07339	-1.74319
3	00:00:33.21	3139967.00	5440238.00	1103016.25	2.65501	0.07384	-1.74086
4	00:00:39.765	3139926.25	5440248.50	1103011.50	2.65543	0.07435	-1.74268

SQL Sorgusu

```
SELECT
(CONVERT(CHAR(8), DATEADD(millisecond, RecSimTimeStamp, 0), 108) + ':' + CAST((RecSimTimeStamp % 1000) AS CHAR(3))) AS SimTime,
STR(WorldLocationX, 12, 2) AS WorldLocationX, STR(WorldLocationY, 12, 2) AS WorldLocationY, STR(WorldLocationZ, 12, 2) AS WorldLocationZ,
STR(VelocityX, 8, 5) AS VelocityX, STR(VelocityY, 8, 5) AS VelocityY,
STR(VelocityZ, 8, 5) AS VelocityZ FROM tbl_SubmersibleVessel
WHERE (RecSimTimeStamp BETWEEN 20000 AND 40000) AND (EntityIdentifier = '1:3001:105')
```

Temizle Sorgula

Query OK.

Şekil 4.29 : Koşum analiz uygulamasında sql ifadesi işletilmesi

Görüldüğü üzere “saklı yordam” kullanımı ile karmaşık sorgular basit bir şekilde çalıştırılabilmektedir. Kullanıcının SQL ifadeleri üzerinde zaman harcamasına gerek kalmadan istenilen sorgular basitçe bu uygulama arayüzünden çalıştırılabilmektedir.

4.9.4 Uygulama kaynak kodu seviyesinde saklı yordam çağrımı

Analiz uygulaması aynı zamanda bir veri tabanı istemci uygulaması olduğu için veri tabanına bağlanarak, kurulan bağlantı üzerinden Sql sorguları ve saklı yordam çağrıları yapılabilir.

Uygulama kaynak kodu seviyesinde saklı yordam çağrısı ve Sql ifadelerini çalıştırmak için Qt Sql birimi kullanılabilir. Qt Sql birimi ile saklı yordam çağrısı için örnek kod Çizelge 4.41’de verilmiştir.

Çizelge 4.41 : Kaynak kodu seviyesinde saklı yordam çağrımı.

```
int minSimTime = 100000;
int maxSimTime = 200000;
QString strID = "1:3001:105";

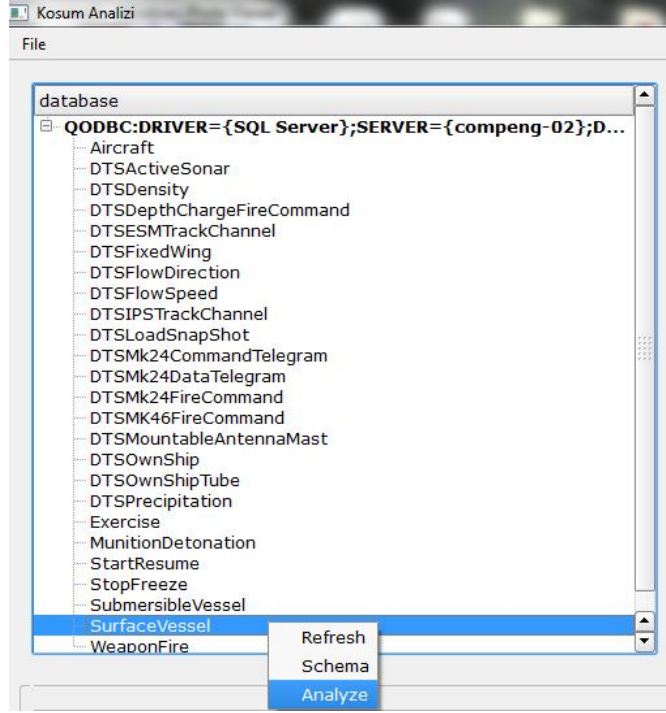
QSqlQuery query;
query.prepare("{CALL getSubmersibleInfo(?, ?, ?)}");
query.addBindValue(minSimTime);
query.addBindValue(maxSimTime);
query.addBindValue(strID);
query.exec();
```

4.9.5 İsteğe uyarlanmış koşum analiz işlemi

Yukarıdaki bölümde kullanıcının saklı yordamlar ve Sql ifadeleri ile koşum analiz uygulamasında analiz yapılabildiği anlatılmıştır. Bu iki yöntemde de müşterinin (kullanıcı) Sql sorgulama ve veri tabanı konularında bilgisi olduğu düşünüldüğünde sorgulama başarılı şekilde yapılabilir. Ancak kullanıcının bu konularda herhangi bir uzmanlığı ve bilgisinin olmadığı durumlar için ayrı bir analiz ekranı da tasarlanmıştır. Tasarlanan bu ekranın amacı, veri tabanında istenilen sorgulamaların ve analiz işlemlerinin yapılabildiğini göstermektir.

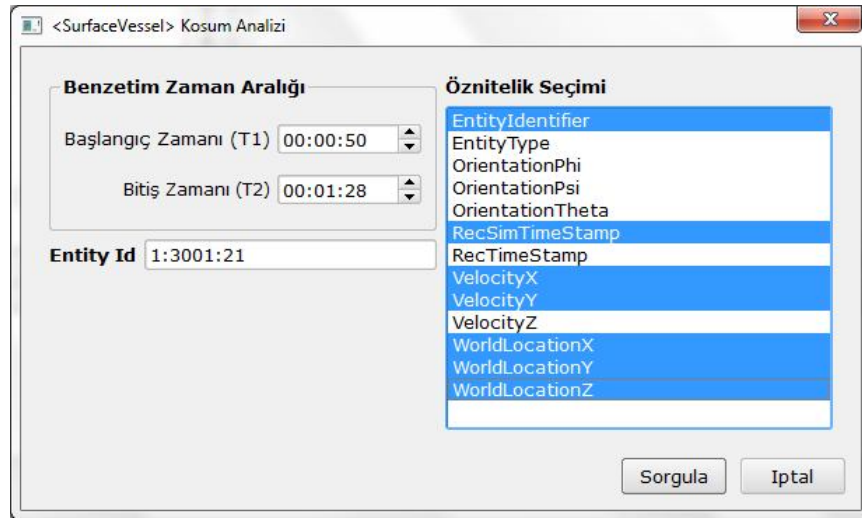
Tasarlanan ekran ile, kullanıcının Sql veya saklı yordam çağrısı yapmasına gerek olmadan, kullanıcının koşum kayıtları üzerinde basitçe sorgulama ve analiz yapabilmesi amaçlanmıştır. Kullanıcı ilgili nesne veya etkileşim sınıfa ilişkin tabloyu ekrandan seçerek sadece istediği alanların (field) değerlerini izleyebilir. Kullanıcının belirli bir zaman aralığına ilişkin sorgulama yapabilmesi de sağlanmıştır. Belirli bir nesne üzerinde sorgulama işlemi yapmak istediğinde nesnenin tekil kimlik bilgisini de analiz işlemine dahil edebilir.

Kullanıcı analiz yapmak istediği etkileşim veya nesne sınıfına ilişkin veri tabanı tablosunu seçerek bu tablo üzerinde bulunan “Analiz” seçeneği ile o sınıfa ilişkin analiz ekranına ulaşır. Şekil 4.30’da seçim ekranı gösterilmiştir.



Şekil 4.30 : Uyarlanmış koşum analiz işlemi için sınıf tablosu seçimi

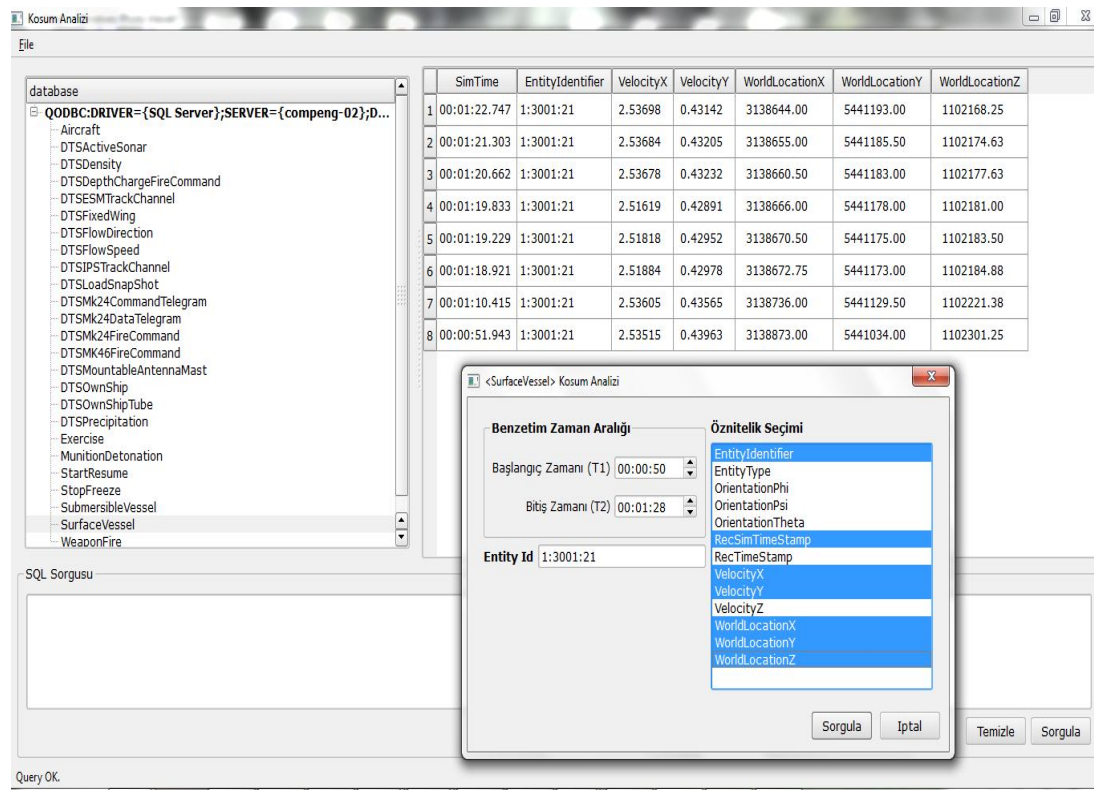
Analiz işleminin ekrandan seçilmesi ile açılan “<SınıfAdı> Koşum Analizi” ekranında kullanıcı zaman aralığı, bileşen tanımlayıcı değeri (seçimlik) ve öznitelik / parametre tercihlerini yaparak sorgulama işlemi yapabilir. Tasarlanan bu yapı ile kullanıcı sadece görmek istediği bilgileri görme imkanını elde eder. Koşumun belirli bir zaman aralığındaki nesne özniteliklerinin değişimini izleyebilir. Şekil 4.31’de parametre ve zaman kıstaslarının girişinin yapıldığı ekran tasarımı gösterilmiştir.



Şekil 4.31 : İsteğe uyarlanmış koşum analizi kıstas giriş ekranı

Kullanıcı sorgulama düğmesine bastıktan sonra seçilen kıstaslar doğrultusunda otomatik olarak bir Sql ifadesi oluşturulur. Bu yöntemin avantajı kullanıcının üst seviyeden kıstas seçimi alınarak kullanıcının Sql ifadeleri yazmasının önüne geçilmiş olmasıdır.

Sorgulama sonucu veri tabanından alınan sonuçlar uygulama içerisinde bir tabloda kullanıcıya zaman sıralı olarak gösterilir. Kullanıcının kıstas seçim ekranında herhangi bir parametre seçimi yapmaması halinde nesneye ilişkin tüm değerler gösterilir. Şekil 4.32’de sorgulama sonucu alınan raporun tabloda gösterimi yapılmıştır.



Şekil 4.32 : Kullanıcı tanımlı sorgu sonuç ekranı

Her benzetim sisteminin analiz gereksinimleri birbirinden farklıdır. Kullanıcıların ihtiyaçları zamanla değişebilir, yeni gereksinimler ortaya çıkabilir. Bu nedenle farklı benzetim sistemlerinin tüm analiz ihtiyaçlarına cevap verecek genel amaçlı bir analiz uygulaması geliştirilmesi oldukça zordur. Buradaki önemli nokta, ihtiyaçlar doğrultusunda analiz uygulamalarının geliştirilebileceği bir ortamın varlığıdır.

Farklı veri tabanı tablolarının birleştirilmesi (join) yöntemi ile hazırlanacak sorgular yardımıyla çok daha karmaşık sorulara cevap alınabilecektir. Tez çalışmasında, koşum kaydı yapılması ve koşum kaydının veri tabanına aktarımı ile koşum hakkında sorgulama yapmaya imkan tanıyacak yapılar geliştirilmiştir.

5. SONUÇ VE ÖNERİLER

Bu çalışmada HLA standardına uygun koşum kayıt ve tekrar oynatım federe uygulaması geliştirilmesi konusu incelenmiştir. Hazır ticari çözümlere alternatif olabilecek çözümün benzetim sisteminin ihtiyaçları doğrultusunda geliştirilmesi ele alınmıştır. Geliştirilen uygulamanın ihtiyaçlar doğrultusunda esnetilebilir olmasına özen gösterilmiştir. Herhangi bir ihtiyaca kolay cevap verebilecek mimari tasarımlar geliştirilmiştir. Kullanıcının analiz ihtiyaçlarına cevap verebilecek veri tabanı aktarım katmanının kod üretim teknikleri ile gerçekleştirilmesi ele alınmıştır. Dinamik Bağlı Kitaplık (Dynamic Link Library – DLL) olarak tasarlanan aktarım katmanı ile, verinin gösterimi ile ilgili kullanıcının değişiklik istekleri uygulamanın tekrar derlenmesine gerek kalmadan karşılanabilmektedir.

Gerçekleştirilen koşum kayıt ve tekrar oynatım federe uygulaması VR-Forces yazılımında bulunan hazır senaryolar çalıştırılarak sınanmıştır. “Benzetim Yönetim Federesi” tarafından gönderilen yaşam döngüsü etkileşim mesajları doğrultusunda kayıt işlemine başladığı ve kayıt dosyasını başarılı şekilde oluşturduğu test edilmiştir. Koşum kaydı tekrar oynatılarak VR-Forces harita ekranında tekrar oynatma izlenmiştir. Koşum kaydının Microsoft SQL Server 2005 veri tabanına başarılı şekilde aktarıldığı test edilmiştir. Geliştirilen “Koşum Analiz” uygulaması ile sorgulara başarılı şekilde yanıtlar alınmıştır.

Geliştirilen uygulamanın artı ve eksileri ile birlikte değerlendirmesi aşağıda 5 farklı başlık altında incelenmiştir :

1. Koşum zamanı analiz ve sorgulama : Yapılan çalışmada geliştirilen uygulamanın ticari uygulamalardan temel olarak en büyük eksiği koşum sırasında sorgulama ve analiz yapılamamasıdır. Uygulama koşum kayıt modunda, RTI’den aldığı nesne ve etkileşim mesajlarını çözümlemeyen ikili biçimde kayıt dosyasında saklar. Bu yöntemin avantajı, kayıt sırasında verinin çözümlemesinden doğacak zaman kayıplarını ortadan kaldırmasıdır. Verilerin kayıt dosyasında çözümlenmiş olarak saklanması, tekrar oynatma işleminde RTI’a gönderilecek verinin ikili biçime

dönüştürülmesini gerektirir. Tekrar oynatmanın kritik olduğu sistemler için bu özellik avantaj sağlayacaktır.

Geliştirilen uygulamanın koşum sırasında gerçek zamanlı sorgulara yanıt verebilmesi için koşum kayıt biriminin “Bellekte Veri Tabanı” veya “Yerel Dosya Veritabanı” sistemleri ile çalışacak şekilde tasarlanması alternatif çözüm olarak sunulabilir. Bu çözüm, uygulamanın koşum kayıt sırasında Sql ifadeleri yardımıyla sorgulamalara cevap verebilmesini sağlayacaktır. Okuma ve yazma hızı yüksek sabit disklerin tercih edilmesi yerel dosya veri tabanı kullanımında gecikmeleri en aza indirecektir. Yerel dosya veri tabanı kullanımı için “Sqlite” açık kaynak kodlu bir çözüm olarak tercih edilebilir. Bellek sorununun olmadığı sistemler için, bellekte veri tabanı çözümü sorgulamalara daha hızlı yanıt alınması açısından avantaj sağlayacaktır.

Koşum kayıt sırasında RTI’den alınan nesne ve etkileşim mesajları kuyruklanarak, ikili biçimdeki verinin çözülmesi işlemi ayrı bir “İş Parçacığı” tarafından yapılabilir. Böylece çözümleme işleminden dolayı federenin üzerindeki yük azalacaktır.

2. Dağıtık mimaride çalışma : Geliştirilen uygulamanın birden fazla kopyası aynı anda federasyona katılarak herbiri ayrı koşum kayıt işlemi yerine getirebilir. Böylelikle federasyonun koşum kaydı dağıtık mimaride gerçekleştirilebilir. Her bir kayıt federesi kaydetmek istediği sınıfları kendi SOM’larında belirtir. İki farklı kayıt federesinin aynı etkileşim veya nesne sınıfına abone isteği belirtmesi durumunda her iki kayıt federesi de aynı sınıfa ilişkin koşum verilerini kaydediyor olacaktır. Birden fazla kayıt federesinin aynı sınıfa ilişkin koşum verilerini kaydetmesi federasyonun çalışma performansını da olumsuz etkileyebilecektir. Federeler kaydettikleri koşum verisini tekrar oynatabileceği için tekrar oynatımın dağıtık mimaride yapılması sağlanabilir. Ancak burada dikkat edilmesi gereken unsur, aynı sınıfa iki farklı kayıt federesi tarafından abone olunmaması gerektiğidir. Aksi halde, tekrar oynatım sırasında aynı nesneye ilişkin güncellemelerin veya aynı etkileşim mesajının birden fazla kez federasyona yayınlanması sorunu ortaya çıkacaktır.

Dağıtık mimaride gerçekleştirilen koşum kayıt işlemi sonunda oluşan kayıt dosyalarının veri tabanına aktarım işleminin buna uygun olarak tasarlanması gerekmektedir. Yapılan tez çalışmasında geliştirilen uygulamada veri tabanı aktarım işlemi merkezi mimari yöntem temel alınarak tasarlanmıştır. Her bir uygulamanın

koşum kayıtlarını veri tabanına aktarımı öncesinde, veri tabanı yapılandırma işleminin merkezi bir noktadan yönetilerek bu sorun ortadan kaldırılabilir.

3. Veri tabanı sisteminden bağımsız aktarım katmanı : Geliştirilen uygulamanın farklı veri tabanı sistemleri ile uyumlu çalışabilmesi için gerekli mimari tasarım buna uygun olarak yapılmıştır. Veri tabanı yapılandırma işlemi Sql komut dosyaları ile gerçekleştirilmektedir. Farklı veri tabanı sistemlerinin veri tabanı yapılandırması ile ilgili Sql ifadeleri birbirinden farklı olabilmektedir. Farklı veri tabanı sistemleri için ayrı Sql komut dosyaları oluşturularak bu durum çözülebilir. Uygulamanın XML yapılandırma dosyasında veri tabanı bağlantı cümlesi değişken olarak tanımlanmıştır. Böylece uygulamanın kaynak koduna müdahale etmeden istenilen veri tabanı sistemine bağlantı kurularak aktarım gerçekleştirilebilir.

“Microsoft SQL Server 2005” veri tabanı sistemine aktarım için uygulamanın yapılandırma dosyasında veri tabanı bağlantı cümlesi aşağıdaki gibi tanımlanmıştır :

```
<AnalysisDatabase>
  <driver>QODBC</driver>
  <connection>DRIVER={SQL Server};
    SERVER={compeng-02};DATABASE={EXERCISE_DEV_00_01};
  </connection>
  <user>myUsername</user>
  <password>myPassword</password>
</AnalysisDatabase>
```

“MySql” veri tabanı sistemine aktarım için gerekli bağlantı cümlesi aşağıdaki gibi tanımlanır:

```
<AnalysisDatabase>
  <driver>QODBC</driver>
  <connection> DRIVER={MySQL ODBC 3.51 Driver};
    SERVER={compeng-02};DATABASE={EXERCISE_DEV_00_01};
  </connection>
  <user>myUsername</user>
  <password>myPassword</password>
</AnalysisDatabase>
```

4. Nesne güncelleme mesajı kayıt eşiği : Geliştirilen uygulama kayıt modunda nesne güncellemelerini belirli eşik değeri aralığında kaydeder. Belirli bir nesneye ilişkin çok sık güncelleme alındığı durumda eşik değerine bakılarak alınan güncellemenin kaydetme kararı verilir. Eşik değerinin yüksek tutulması nesne güncellemelerinin bir kısmının yok sayılmasına neden olacaktır. Eşik değerinin çok

küçük belirlenmesi kayıtların artmasına neden olacaktır. Yüksek hassasiyette kayıt yapılması isteniyorsa eşik değerinin küçük belirlenmesi gerekir.

Kayıt işleminde belirli nesne sınıfına ilişkin güncellemelerin eşik değerinin küçük (yüksek hassasiyette kayıt) , bazı nesne sınıflarının eşik değerinin daha büyük olması ihtiyacı doğabilir. Bu durumda geliştirilen uygulamanın SOM dosyasında yapılacak bir güncelleme ile mümkün olabilecektir. SOM’da aşağıda gösterilen değişiklik yapılarak bu özellik uygulamaya kazandırılabilir.

```
<Object FedName="LOGGER"
  FullName="HLAObjectRoot.BaseEntity.PhysicalEntity.Platform.SurfaceVessel"
  ShortName="SurfaceVessel" PS="3"
  updateThreshold="50" >
</Object>
<Object FedName="LOGGER"
  FullName="HLAObjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft"
  ShortName="Aircraft" PS="3"
  updateThreshold="500" >
</Object>
```

“SurfaceVessel” nesne sınıfının koşum kaydının “Aircraft” sınıfına göre daha yüksek duyarlılıkta yapıldığı XML düğümünde yer alan “updateThreshold” parametresine bakılarak anlaşılmaktadır.

5. “Gerçek Zaman” ve “Benzetim Zamanı” yaklaşımı : Geliştirilen uygulamada RTI’den alınan nesne ve etkileşim mesajları hem gerçek zaman bilgisi hem de benzetim zamanı bilgisiyle birlikte kaydedilir. Koşum sırasında federasyonun bekletilmesi (pause) sırasında gerçek zaman ilerlerken benzetim zamanı ilerletilmez. Benzetim Zamanı (Simulation Time) tekrar oynatma için kritiktir. Koşumun farklı hızlarda oynatılması için (1x, 2x, 4x) benzetim zamanı belirlenen hızda ilerletilerek nesne ve etkileşimler RTI’a gönderilir. Tekrar oynatım modunda farklı oynatım hızlarını destekleyecek yapılar geliştirilmiş olmasına karşın kullanıcı arayüzlerine bu özellik eklenmemiştir.

Tez çalışmasında ticari çözümlere alternatif olabilecek koşum kayıt ve tekrar oynatım federe uygulamasının geliştirilmesi anlatılmıştır. Böylece fiyattan da kazanç sağlanmıştır. Aşağıda ticari çözüm olarak “MAK Data Logger” ve “Pitch Recorder” incelenmiştir.

MAK Data Logger : “MAK Technologies” firması tarafından geliştirilmiş ticari bir uygulamadır. HLA ve DIS temelli benzetim sistemleri için koşum kayıt ve tekrar oynatım fonksiyonları sunar. RPR-FOM referans nesne modeli ile uyumlu çalışır. Grafikselsel kullanıcı ara yüzünden koşum kayıt işlemi başlatılarak HLA ve DIS mesajlarını .lgr uzantılı kayıt dosyasına kaydeder, kayıt dosyası ile tekrar oynatma işlemi yapabilir. Tekrar oynatım işlemi sırasında bekletme, hızlı ilerletme (fast forward), koşum hızlandırma / yavaşlatma ve istenilen koşum zamanına atlama (jump in time) özelliklerini destekler.

Koşum kayıt dosyasında bulunan kayıtların görsel bir arayüzde güncellenmesine imkan tanır. Koşum kayıt dosyasından istenilmeyen bölümler çıkarılabilir, kayıt dosyaları birleştirilerek yeni koşum kayıt dosyaları elde edilebilir. Büyük senaryoların kesilerek daha küçük bölümler halinde oynatımına imkan tanır.

Analiz işlemleri için veri tabanı aktarım fonksiyonu sunar. Koşum kaydı sırasında veya tekrar oynatma işlemi sırasında da veri tabanı aktarım işlemi yapılabilir. Veri tabanı aktarım modülü, aktarım için gerekli veri tabanı yapılandırma işlemini federasyon FOM’una uygun olarak gerçekleştirir. Veriler veri tabanına aktarıldıktan sonra MATLAB veya Excel gibi uygulamalar aracılığıyla benzetim sonuçları analiz edilebilir. Veri madenciliği uygulamaları aracılığı ile Sql sorguları yazılabilir. “MySQL” ve “Microsoft Access” veri tabanı sistemleri ile uyumlu çalışır.

“Logger Toolkit API” programlama arabirimi kullanılarak kayıt fonksiyonu federasyondaki federelere de entegre edilebilir. Bu sayede istenirse koşum kayıt işlemi tam dağıtık mimaride gerçekleştirilebilir. Programlama arabirimi ile “Logger” uygulamasına istenirse ek özellikler de kazandırılabilir. Toolkit API kullanımı için “VR-Link” geliştirici lisansı bulunması zorunludur. “Logger File API” programlama arabirimi ile kayıt dosyasına çevrimdışı erişim sağlanabilmektedir.

Kullanıcı tarafından ayarlanabilen filtre özelliği ile istenilen FOM sınıflarının kayıt edilip edilmemesi belirlenebilir.

Pitch Recorder : İsveç kökenli “Pitch Technologies” firması tarafından geliştirilmiş ticari bir uygulamadır. HLA temelli benzetim sistemleri için koşum kayıt ve tekrar oynatım fonksiyonları sunar. Koşum sırasında gerçek zamanlı olarak veri alış verişini izleme imkanı sunar. RPR-FOM referans nesne modeli ile uyumludur. Nesne güncellemeleri ile etkileşim mesajlarını kaydeder. Koşum kayıt sırasında veri hızını

grafiksel olarak gösterebilir, nesnelerin öznitelikleri gerçek zamanlı olarak izlenebilir. Anlık kayıt işlemleri için yerel veri tabanı yapısı kullanır. Koşum kaydının Excel formatına ve ilişkisel veri tabanı sisteminlerine aktarım özelliğine sahiptir. MS SQL Server 2008, MS SQL Server 2005, MySQL 5.0, Oracle10g veri tabanı sistemleri ile uyumlu çalışır. Bellekte veri tabanı ve yerel dosya veritabanını da destekler.

Tez çalışması kapsamında geliştirilen uygulamanın bu iki ticari çözüm ile hizmet karşılaştırması Çizelge 5.1’de yapılmıştır.

Çizelge 5.1 : Hizmet karşılaştırma.

Koşum Kayıt ve Tekrar Oynatım Fonksiyonları	Pitch Recorder	MAK Logger	Tez Çalışması
Koşum Zamanı Analiz ve Sorgulama	+	+	–
Veri Tabanı Aktarım Fonksiyonu	+	+	+
RPR FOM Uyumluluk	+	+	+
DIS Uyumluluk	–	+	+
Nesne ve Etkileşim Sınıfı Filtreleme	+	+	+
İstenilen Koşum Zamanına Atlama	+	+	+
Farklı Oynatım Hızlarını Destekleme	+	+	+

KAYNAKLAR

- [1] **Defense Modeling and Simulation Office**, 2005. Transition of the DoD High Level Architecture TO IEEE STANDARD 1516 Version 1.0., Virginia
- [2] **IEEE STD 1516**, 2000. Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules, *IEEE*, New York.
- [3] **IEEE STD 1516.1**, 2000. Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification, *IEEE*, New York.
- [4] **IEEE STD 1516.2**, 2000. Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification, *IEEE*, New York.
- [5] **Hook, V., Daniel, J. and Calvin, James O.**, 1997. Execution Logging and Replay : Issues and Approaches. *Simulation Interoperability Workshop*
- [6] **Bachinsky, Stephen T., Dr. Tarbox, Glenn H. and Dr. Powell, Edward T.**, 1997. A Data Collection In An HLA Environment. *Simulation Interoperability Workshop*.
- [7] **Bair, Kevin D.**, 1998. A Methodology for Federation Data Collection and Analysis. *Simulation Interoperability Workshop*
- [8] **Magee, G., Shanks, G.**, 1999. Lessons Learned from an Implementation of a Fully Distributed Data Collection Tool. *Simulation Interoperability Workshop*.
- [9] **Li, W., Yang, M. and Wang, Zi-Cai**, 2006. Flexible Simulation Data Collection And Replay Tool. *Proceedings of the Fifth International Conference on Machine Learning and Cybernetics*, Dalian, China.
- [10] **SISO Real-time Platform Reference Federation Object Model**, 1999, <http://www.sisostds.org/index.php?tg=fileman&idx=get&id=5&gr=Y&path=SISO+Products/SISO+Standards&file=RPR-FOMV1.0_Final.pdf>, *alındığı tarih 05.08.2010*.
- [11] **Url-1** < <http://www.mak.com/products/rti.php> >, *alındığı tarih 19.08.2010*
- [12] **Url-2** < <http://www.mak.com/products/vrforces.php>>, *alındığı tarih 19.08.2010*
- [13] **Bair, Kevin D.**, 1998. A Methodology for Federation Data Collection and Analysis. *Simulation Interoperability Workshop*.
- [14] **Url-3** < http://www.boost.org/doc/libs/1_44_0/libs/serialization/doc/index.html >, *alındığı tarih 19.08.2010*
- [15] **Url-4** < <http://www.ruby-lang.org> >, *alındığı tarih 19.08.2010*

ÖZGEÇMİŞ

Ad Soyad: Selçuk Giray ÖZDAMAR

Doğum Yeri ve Tarihi: Aydın, 1982

Lisans Üniversite: Ege Üniversitesi Bilgisayar Mühendisliği