BAŞKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ TEZLİ YÜKSEK LİSANS PROGRAMI

FEDERELERİN YÜKSEK SEVİYE MİMARİ STANDART UYUMLULUĞUNUN DENETLENMESİ İÇİN ÇÖZÜM ÖNERİSİ

HAZIRLAYAN

ARZUM BERRAK ARIBAL

YÜKSEK LİSANS TEZİ

ANKARA - 2021

BAŞKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ TEZLİ YÜKSEK LİSANS PROGRAMI

FEDERELERİN YÜKSEK SEVİYE MİMARİ STANDART UYUMLULUĞUNUN DENETLENMESİ İÇİN ÇÖZÜM ÖNERİSİ

HAZIRLAYAN

ARZUM BERRAK ARIBAL

YÜKSEK LİSANS TEZİ

TEZ DANIŞMANI

DR. ÖĞR. ÜYESİ TÜLİN ERÇELEBİ AYYILDIZ

ANKARA – 2021

BAŞKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

Bilgisayar Mühendisliği Anabilim Dalı Bilgisayar Mühendisliği Tezli Yüksek Lisans Programı çerçevesinde Arzum Berrak ARIBAL tarafından hazırlanan bu çalışma, aşağıdaki jüri tarafından Yüksek Lisans Tezi olarak kabul edilmiştir.

Tez Savunma Tarihi: 04/01/2021

Tez Adı: Federelerin Yüksek Seviye Mimari Standart Uyumluluğunun Denetlenmesi İçin Çözüm Önerisi

Tez Jüri Üyeleri (Unvanı, Adı - Soyadı, Kurumu) **İmza**

Prof. Dr. Ali Hikmet DOĞRU – Orta Doğu Teknik Üniversitesi ..……...

Dr. Öğr. Üyesi Tülin ERÇELEBİ AYYILDIZ – Başkent Üniversitesi ……

Dr. Öğr. Üyesi Emre SÜMER – Başkent Üniversitesi ……

ONAY

Prof. Dr. Ömer Faruk ELALDI

Fen Bilimleri Enstitüsü Müdürü

Tarih: … / … / ……..

BAŞKENT ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZ ÇALIŞMASI ORİJİNALLİK RAPORU

Tarih: … / … / ……..

Öğrencinin Adı, Soyadı: Arzum Berrak ARIBAL

Öğrencinin Numarası: 21810275

Anabilim Dalı: Bilgisayar Mühendisliği Anabilim Dalı

Programı: Bilgisayar Mühendisliği Tezli Yüksek Lisans Programı

Danışmanın Unvanı/Adı, Soyadı: Dr. Öğr. Üyesi Tülin ERÇELEBİ AYYILDIZ

Tez Başlığı: Federelerin Yüksek Seviye Mimari Standart Uyumluluğunun Denetlenmesi İçin Çözüm Önerisi

Yukarıda başlığı belirtilen Yüksek Lisans tez çalışmamın; Giriş, Ana Bölümler ve Sonuç Bölümünden oluşan, toplam 34 sayfalık kısmına ilişkin, 11/01/2020 tarihinde tez danışmanım tarafından Turnitin adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı %5'tir. Uygulanan filtrelemeler:

1. Kaynakça hariç

2. Alıntılar hariç

3. Beş (5) kelimeden daha az örtüşme içeren metin kısımları hariç

"Başkent Üniversitesi Enstitüleri Tez Çalışması Orijinallik Raporu Alınması ve Kullanılması Usul ve Esaslarını" inceledim ve bu uygulama esaslarında belirtilen azami benzerlik oranlarına tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Öğrenci İmzası: …….

ONAY

Tarih: … / … / ……..

Dr. Öğr. Üyesi Tülin ERÇELEBİ AYYILDIZ

Bu tezi bana her zaman destek veren aileme ithaf ediyorum.

Arzum Berrak ARIBAL

Ankara – 2021

TEŞEKKÜR

Yüksek lisans araştırmalarımda öncelikle danışmanım Dr. Öğr. Üyesi Tülin ERÇELEBİ AYYILDIZ'a sabrı, motivasyonu, coşkusu ve bilgisi ile sürekli sağladığı desteğinden dolayı içten şükranlarımı sunarım.

Danışmanımın yanı sıra, CEY Savunma şirketinde birlikte çalışma fırsatı bulduğum yöneticilerim: Dr. Hürkan Orkun ZORBA'ya ve Serkan ÖZKAYMAK'a anlayışlı davranışları, değerli fikirleri ve önerileri için teşekkür ederim. Onların rehberliği ve desteği, bu araştırmanın her aşamasında bana yardımcı oldu.

Son olarak, hayatımın her döneminde hiçbir fedakarlıktan kaçınmayan, beni cesaretlendiren, maddi ve manevi olarak destekleyen başta anneannem olmak üzere sevgili aileme teşekkür etmek isterim.

ÖZET

Arzum Berrak ARIBAL

FEDERELERİN YÜKSEK SEVİYE MİMARİ STANDART UYUMLULUĞUNUN DENETLENMESİ İÇİN ÇÖZÜM ÖNERİSİ

Başkent Üniversitesi Fen Bilimleri Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalı

2021

Yüksek Seviye Mimari (High Level Architecture, HLA), günümüzde simülasyon sistemlerinin birlikte çalışabilirliğini ve yeniden kullanılabilirliğini desteklemek amacıyla simülasyon mimarisini tanımlayan, Elektrik ve Elektronik Mühendisleri Enstitüsü (Institute of Electrical and Electronics, IEEE) tarafından geliştirilmiş bir standarttır. Özellikle askeri amaçlı olmak üzere farklı kuruluşlar tarafından farklı teknolojiler kullanılarak geliştirilmiş HLA uyumlu simülasyonlar mevcuttur. Müşterek harekât konseptinin gelişmesine paralel olarak farklı simülasyonların entegre bir şekilde çalışması ihtiyacı doğmuştur. Bununla birlikte HLA uyumlu simülasyonların entegrasyonu ve birlikte çalışabilirliği için standart uyumluluğu gözetmek kaçınılmaz hale gelmiştir. Gerçekleştirilen birçok projede HLA kurallarının sadece belirli bir kısmı uygulanmakta olup bu yaklaşım simülasyon sistemlerinin birlikte çalışabilirliğini sağlamak açısından çoğu zaman yetersiz kalmaktadır. Bu nedenle federelerin HLA uyumlu tasarlanması büyük önem taşımaktadır. Bu çalışma bir federasyona dahil olan federelerin HLA uyumluluk sınamasının gerçekleştirilmesi için bir temel teşkil etmektedir. Federelerin HLA uyumunun denetlenmesi için mevcut yöntemler analiz edilmiş ve bu analiz sonrası HLA uyumluluğu ile beraber birlikte çalışabilirlik problemlerinin çözümü için bir yöntem önerilmiştir. Bu yöntem geliştirilen araç seti ile desteklenerek geliştirilmesi devam eden bir Çalışma Zamanı Altyapısı (Run-Time Infrastructure, RTI) ürünü ile birlikte denenmiş: birlikte çalışabilirlik problemleri en aza indirilmiş ve simülasyonların birlikte çalışabilirliği için gerekli olan HLA uyumluluğu test edilmiştir. Araştırma sonucunda birlikte çalışabilirlik faaliyetleri kapsamında karşılaşılan uyumluluk problemlerine bir çözüm üretilmiştir.

ANAHTAR KELİMELER: Yüksek Seviye Mimari, Modelleme ve Simülasyon, Birlikte Çalışabilirlik.

ABSTRACT

High Level Architecture (HLA) is a standard developed by IEEE (Institute of Electrical and Electronics), which defines simulation architecture in order to support the interoperability and reusability of simulation systems at the present time. There are HLA compatible simulations developed by different organizations using various technologies, especially for military purposes. In parallel with the development of the joint operation concept, the need for different simulations to work integratedly. However, it has become inevitable to observe standard compatibility for the integration and interoperability of HLA compatible simulations. Only a certain part of the HLA rules are implemented in many developed projects, and this approach is often insufficient to ensure the interoperability of simulation systems. For this reason, it is very important to design the federates in compliance with HLA. This study is the basis for performing the High Level Architecture (HLA) compatibility test of federates included in a federation. Existing methods were analyzed to check the HLA compliance of federates, and after this analysis, a method is proposed for the solution of interoperability problems with HLA compatibility. This method has been tested with a Run-Time Infrastructure (RTI) product that is still being developed by supporting it with the developed toolkit: interoperability problems are minimized and the HLA compatibility required for simulations interoperability has been tested. As a result of the research, a solution has been produced for compatibility problems encountered within the scope of interoperability activities.

KEYWORDS: High Level Architecture, Modelling and Simulation, Interoperability.

ÖNSÖZ

Simülasyon sistemleri için birlikte çalışabilirlik ve yeniden kullanılabilirlik HLA standart uyumsuzlukları nedeniyle oldukça zor bir hedeftir. Simülasyon birlikte çalışabilirliğini sağlamak; teknik, sözdizimsel, anlamsal ve pragmatik düzeyde çaba ve standardizasyon gerektirir. HLA standardında yer alan kurallar; federasyon, federe ve RTI birimlerinin üzerindeki sorumlulukları tanımaktadır. Özellikle HLA federe kuralları simülasyonların birlikte çalışılabilirliğini sağlamak açısından önemlidir. Nesne model şablonu ise federasyon ve federeler için tanımlanmış HLA nesne modellerinin formatını belirlemektedir. Bahsedilen bu nesne modelleri; nesne ve etkileşim tanımlamalarını içermektedir. Federasyon ve federeler için nesne modelleri sırasıyla FOM ve SOM olarak adlandırılmıştır. FOM ve SOM, simülasyonların birlikte çalışabilmesi için gereken veri değişimi ve veri tanımlaması açısından önemlidir. Fakat temel seviyede birlikte çalışabilirliği sağlamak pratikte yeterli görülmekle birlikte, teorik olarak uygulamada bu konuda sorunlar yaşandığı bilinmektedir. Gelişen teknolojiyle birlikte günümüzde simülasyon sistemleri için birlikle çalışabilirlik, yeniden kullanılabilirlik çalışmalarının artması ve HLA uyumluluğun sağlanması ihtiyacı doğmaktadır. Bu nedenle simülasyon sistemlerinin HLA kurallarına uyumluluğu denetlenerek birlikte çalışabilirlik sorunları ortadan kaldırılabilir. Bu amaçla bu çalışmada federenin HLA uyumluluğunun sınanması için HLA standardında yer alan bu kurallar gözden geçirilerek bir test aracı geliştirilmiştir. Bu araç simülasyon sistemleri birlikte çalışabilirliği için CEY Savunma (2016 yılında kurulmuş, Hacettepe Teknokent'te faaliyetlerini yürüten savunma sanayii alanında yazılım, modelleme ve simülasyon konularında çalışan bir şirket) bünyesinde geliştirilmesi devam eden HLA uyumlu, yüksek performanslı, yerli ve milli bir RTI yazılım çözümü testlerinde de kullanılmaktadır. Bu kapsamda federelerin uyumluluğu IEEE HLA 1516-2000 standardına uygun olarak sınanmıştır.

İÇİNDEKİLER

TABLOLAR LİSTESİ

ŞEKİLLER LİSTESİ

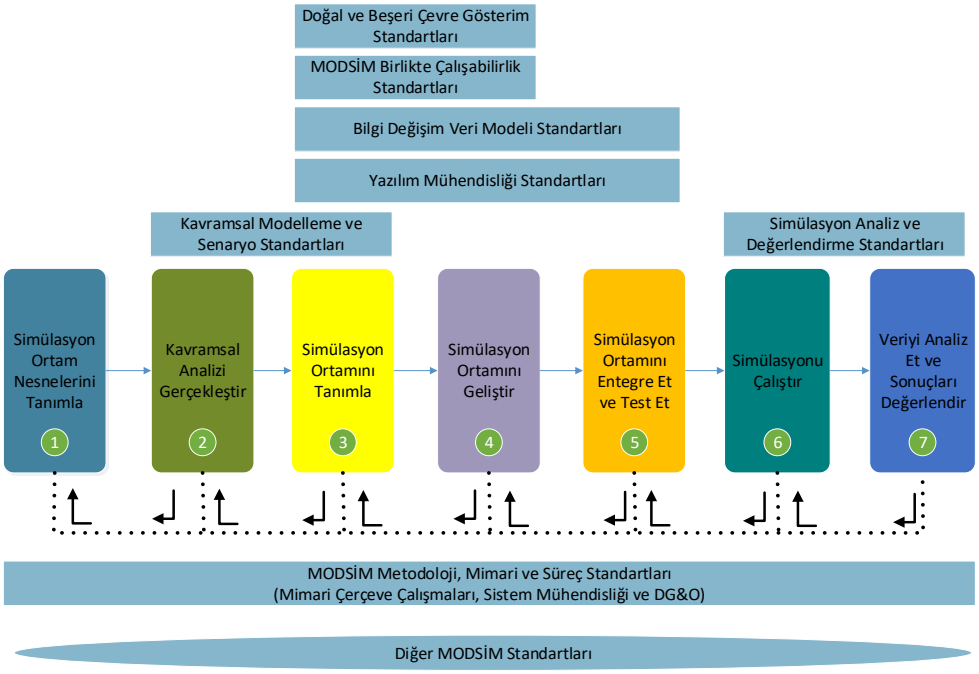
SİMGELER VE KISITLAMALAR LİSTESİ

ALSP	Aggregate Level Simulation Protocol – (Biriktirilmiş Seviye Simülasyon Protokolü)
DIS	Distributed Interactive Simulation – (Dağıtılmış Etkileşimli Simülasyon)
DMSO	Defense Modelling and Simulation Office – (Savunma Modelleme ve Simülasyon Ofisi)
DOD	Department of Defense (Amerikan Savunma Bakanlığı)
DSEEP	Distributed Simulation Engineering and Execution Process – (Dağıtık Simülasyon Mühendislik ve İşletim Süreci)
FED	Federation Execution Data – (Federasyon Yönetimi Verisi)
FEDEP	Federation Development and Execution Process – (Federasyon Geliştirme ve Çalıştırma Süreci)
FOM	Federation Object Model – (Federasyon Nesne Modeli)
FUT	Federate Under Test – (Test Edilen Federe)
GALT	Greatest Available Logical Time – (Mevcut En Büyük Mantıksal Zaman)
HLA	High Level Architecture – (Yüksek Seviye Mimari)
IEEE	Institute of Electrical and Electronics – (Elektrik ve Elektronik Mühendisleri Enstitüsü)
MODSIM	Modelling & Simulation – (Modelleme ve Simülasyon)
OMT	Object Model Template – (Nesne Modeli Şablonu)
RPR-FOM	Real-time Platform Reference FOM – (Gerçek Zamanlı Platform Referans FOM)
RTI	Run-Time Infrastructure – (Çalışma Zamanı Altyapısı)
SOM	Simulation Object Model – (Simülasyon Nesne Modeli)

1. GİRİŞ

Modelleme ve Simülasyon (Modelling & Simulation, MODSIM) sistemleri için birlikte çalışabilirlik, birden fazla benzer sistemin fiziksel olarak birbirine bağlı olması, birbirlerine mesaj, veri veya hizmet sunması ve sunulan bu mesaj, veri ve hizmetlerin bir arada çalışmaları için etkin bir biçimde kullanılması olarak tanımlanabilir. Böylece farklı simülasyon çözümlerinin güçlü yanları birleştirilerek, bu alanda kullanıcı ihtiyaçlarının daha iyi karşılanması hedeflenmektedir. Birlikte çalışabilirlik ise; "bir sistemin ya da sürecin, ortak standartlar çerçevesinde diğer bir sistemin ya da sürecin bilgisini ve/veya işlevlerini kullanabilme yeteneği" olarak tanımlanabilir [1][2]. Birlikte çalışabilirlik için sadece veri değişimi değil, simülasyonlar arası senkronizasyonda önem taşmaktadır.

Modellerin ve simülasyonların birlikte çalışabilirliğini sağlamak karmaşık bir konudur. Bu nedenle modelleme ve simülasyon dünyasında birlikte çalışabilirlik için birçok standart tanımlanmıştır. Bu standartlar farklı soyutlama seviyelerini hedeflemektedir. Örneğin Dağıtık Simülasyon Mühendislik ve İşletim Süreci (Distributed Simulation Engineering and Execution Process, DSEEP) [3] simülasyon birlikte çalışabilirliğini sağlamak için ön koşul olan simülasyon geliştirme sürecini tanımlar. Bu doğrultuda standart grupları altında verilen standartların bir çoğu DSEEP süreç adımlarında uygulanabilecek şekilde tanımlanmıştır. Bu standart kategorilerinin DSEEP süreci ile ilişkisi Şekil 1'de verilmiştir [4].



Şekil 1 – DSEEP Simülasyon Mühendislik Süreci ve Standart Kategorileri [4]

Daha düşük soyutlama seviyelerinde birlikte çalışabilirliği ve tekrar kullanılabilirliğini sağlamak için geliştirilen standartlardan en önemlisi Yüksek Seviye Mimari (High Level Architecture, HLA)'dir [5]. HLA, 1995 yılında Amerikan Savunma Bakanlığı (Department of Defense, DoD) Savunma Modelleme ve Simülasyon Ofisi (Defense Modeling and Simulation Office, DMSO) tarafından askeri projelerin ihtiyaçlarına uygun olarak önerilmiştir. Amacı, Dağıtılmış Etkileşimli Simülasyon (Distributed Interactive Simulation, DIS) [6], Biriktirilmiş Seviye ve Simülasyon Protokolü (Aggregate Level Simulation Protocol, ALSP) [7] gibi önceki standartlardan öğrenilmiş dersleri uygulayarak, çoklu donanım, yazılım ve ağ mimarilerini destekleyecek fonksiyonel bir mimariyi tanımlayarak mevcut simülasyon sistemlerinin birlikte çalışabilirliği ve yeniden kullanılabilirliğini sağlamak için genel bir simülasyon çerçevesi oluşturmaktır. HLA üç versiyona sahiptir: 1998'de DMSO tarafından yayınlanan HLA 1.3, 2000 yılında Elektrik ve Elektronik Mühendisleri Enstitüsü (Institute of Electrical and Electronics, IEEE) tarafından yayınlanan HLA1516-2000 [8] ile 2010 yılında IEEE tarafından yayınlanan ve HLA Evolved da olarak bilinen HLA1516-2010 [9]. HLA başlangıçta tamamen askeri uygulamaların ihtiyaçlarına uygun olarak geliştirilmiştir. Fakat günümüzde simülasyon sistemlerinin kullanıldığı bir çok farklı alanda uygulanmaktadır [5][10].

Bir simülasyon sisteminin HLA standardı ile uyumluluğu, "Çalışma Zamanı Altyapısı (Run-Time Infrastructure, RTI) Uyumluluğu" ve "Federe Uyumluluğu" olmak üzere iki farklı adım ile test edilmektedir. RTI Uyumluluğu, DMSO tarafından HLA standardı ile uyumluluk için 2000 civarında test ile gerçekleştirilir. Federe Uyumluluğu ise HLA servislerinin testi yapılarak sağlanabilir. Uyumluluk (Sertifikasyon) için temel adımlar şu şekildedir;

- Web üzerinden test uygulamasının çalıştırılması
- Federe geliştiricisinin web üzerinden Test Edilen Federe (Federate Under Test, FUT) için bir uygunluk çizelgesi göndermesi
- Arayüz çevre verilerinin sunulması
- Arayüz dokümantasyonu ve raporlamanın sunulması

Öte yandan DoD Ocak 2015'te bütçe kısıtları nedeniyle RTI sertifikasyon faaliyetlerini durdurduğunu açıklamıştır. Bu tarihten sonra simülasyon alanında gerçekleştirilen ürünler standartta tanımlanan servisleri gerçekleştirmesine rağmen "sertifikalanmış" (*certified*) veya "doğrulanmış" (*verified*) olarak tanımlanamamakta,

ancak bu standartla “uyumlu” (*compliant*) olduğundan bahsedilebilmektedir. Bu nedenle de günümüzde HLA Evolved (IEEE 1516-2010) standardı ile sertifikalanmış veya doğrulanmış herhangi bir ürün bulunmamakta, geliştiriciler ürünlerini bu standartla uyumlu olarak geliştirmektedir.

Simülasyonun çok uzun bir geçmişi olmamasına rağmen, son yıllarda dünyada ve ülkemizde yaygın bir şekilde kullanılmaya başlanan modelleme ve simülasyon teknolojilerinin artan önemine bağlı olarak, ülkemizde özellikle savunma sanayiinde bu konuda ciddi çalışmalar yapılmaktadır. Bu kapsamda dünyada bu işle uğraşan diğer kurum ve kuruluşlarla rekabet edecek HLA uyumlu birçok taktik, operatif ve stratejik seviye simülasyon sistemi geliştirilmiştir ve geliştirilmeye devam etmektedir. Geliştirilen bu modelleme ve simülasyon projelerinde de standartlara uyumluluk konusuna önem verilmekte, bu amaçla geliştirilen sistem ve projelerin uluslararası standartlara uyumluluk kapsamında bazı hazır çözüm yazılım ve altyapılar kullandıkları görülmektedir. Bununla birlikte literatürde yapılan araştırmalar doğrultusunda bilindiği kadarıyla, bir simülasyonun HLA standardına uyumluluğunu test eden bir çalışmaya rastlanamamıştır.

Bu çalışmada simülasyon projelerinde yer alan federelerin HLA uyumluluğunun sınanması için mevcut yöntemler analiz edilmiştir. Yöntemler, HLA Standardı’nda yer alan federe kurallarına dayanarak analiz edilmiştir. Bu analiz sonucunda bir araç seti geliştirilmiş ve federenin HLA uyumluluğu HLA 1516-2000 standardına uygun olarak sınanmıştır. Giriş bölümünde HLA’yı odakta tutan Modelleme ve Simülasyon temel kavramlarına genel bakış verilmiştir. Bu tezin geri kalan kısmı ise şu şekilde düzenlenmiştir. Bölüm 2’de Yüksek Seviye Mimari ile ilgili konulara genel bir bakış sağlanmış, Bölüm 3’te federenin uyumluluğunun denetlenmesi ile ilgili yöntemler listelenmiş, Bölüm 4’te, standarda uygun federe geliştirimi ile ilgili mevcut çalışmalar incelenerek elde edilen sonuçlar tartışılmıştır.

2. YÜKSEK SEVİYE MİMARİ

Bu bölümde HLA standardında genel bir bakış verilerek standart uyumluluğunu sınamak için gerçekleştirilecek adımlardan bahsedilmiştir.

2.1. Yüksek Seviye Mimari (High Level Architeture, HLA) Bileşenleri

HLA'yı (i) HLA Çerçeve ve Kurallar (IEEE Std 1516-2000); (ii) Federe Arayüz Spesifikasyonu (IEEE Std 1516.1-2000); (iii) Nesne Modeli Şablonu (Object Model Template, OMT) (IEEE Std 1516.2-2000); (iv) Federasyon Geliştirme ve Çalıştırma Süreci (Federation Development and Execution Process, FEDEP) (IEEE Std 1516.3-2003) kavramları oluşturmaktadır [8]. HLA'yı oluşturan kavramlar Tablo 1 üzerinde yer almaktadır.

HLA Çerçeve ve Kurallar, federasyon iletimi sırasında, federelerin doğru şekilde etkileşimi için gerekli prensiplerdir. Kurallar, federelerin ve federasyonu tasarlayan kişilerin sorumluluklarını tanımlamaktadır.

Federe Arayüz Spesifikasyonu, federeler ve RTI arasındaki ara yüzü tanımlamaktadır. Simülasyonların birbirleriyle nasıl etkileştiğini açıklar.

Nesne Modeli Şablonu (Object Model Template, OMT), simülasyonda nesneleri, nitelikleri ve aralarındaki iletişim biçimini tanımaktadır [11]. FOM'nin yapısını belirler. Federasyon Nesne Modeli (Federation Object Model, FOM) ve Simülasyon Nesne Modeli (Simulation Object Model, SOM) olmak üzere iki farklı modeldir.

Federasyon Geliştirme ve Çalıştırma Süreci (Federation Development and Execution Process, FEDEP), federasyonun geliştirilmesinde ve işletilmesinde uygulanması önerilen süreç adımlarını tanımlamaktadır. FEDEP altı ana süreç adımına ayrıştırılmıştır: federasyon hedef tanımı, kavramsal model geliştirme, federasyon tasarımı, federasyon geliştirme, federasyon entegrasyonu ve testi ve sonuçların yürütülmesi ve hazırlanması [12].

Tablo 1. Yüksek Seviye Mimari (High Level Architecture, HLA) Standardı

Standart Tanımı	**Açıklama**
IEEE 1516–2000	IEEE Standard for Modeling and Simulation (M&S) HLA Framework and Rules, approved by IEEE on September 21st, 2000.
IEEE 1516.1–2000	IEEE Standard for M&S HLA Federate Interface Specification, approved by IEEE on September 21st, 2000.
IEEE 1516.2–2000	IEEE Standard for M&S HLA Object Model Template (OMT) Specification, approved by IEEE on September 21st, 2000.
IEEE 1516.3–2003	IEEE Recommended Practice for HLA Federation Development and Execution Process, approved by IEEE on April 23rd, 2003.

2.2. Modelleme ve Simülasyon Terimleri

Bu bölümde HLA standardında yer alan terimler hakkında genel bilgi verilmiştir.

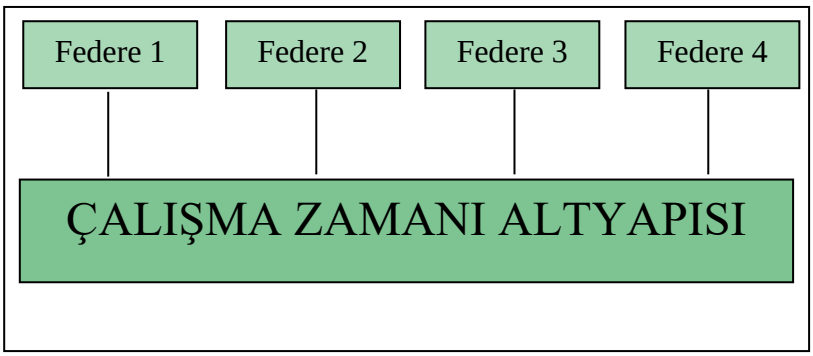
2.2.1. Federasyon

HLA standardında birlikte çalışan simülasyonların bütünü federasyon olarak ifade edilmektedir. Federasyon üç bileşenden oluşmaktadır [13]:

Federe, federasyonun bir üyesidir ve tek noktadan RTI'ya bağlıdır.

FOM, federasyon geliştirici tarafından tanımlanan, federelerin kendi aralarında değiş tokuş yaptığı veriler arasındaki ilişkileri tanımlamaktadır. Bir kısmı RTI tarafından okunan federeler için tanımlanmış bir anlaşmadır.

Çalışma Zamanı Altyapısı (Run-Time Infrastructure, RTI), federasyonun işletilmesini sağlamaktadır. İleri ki bölümlerde detaylandırılacaktır. Şekil 2'de RTI ile ilgili genel bir gösterim verilmektedir.



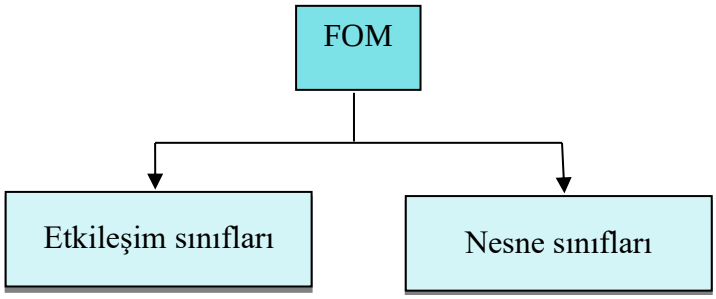
Şekil 2 – Çalışma Zamanı Altyapısı (Run-Time Infrastructure, RTI)

2.2.2. Federe

HLA standardında her bir simülasyon ise federe olarak isimlendirilmektedir. Federeler federasyon içerisinde işletilir (*Federation Execution*). Federe tek bir gemiyi, bir filoyu, bir ülkenin ordusunu temsil eden bir simülasyon vb. şeklinde olabilir.

2.2.3. Nesne Modeli Şablonu (Object Model Template, OMT)

Önceki bölümde de belirtildiği gibi, OMT tüm FOM'ler için bir yapı tanımlamaktadır Her bir federasyon, o federasyon için tanımlanmış FOM'yi kullanmaktadır. FOM federeye özel bilgiler bulundurmaz; sadece, bir federasyon işletimi sırasında RTI üzerinden federelerin alıp vereceği veriyi tanımlamaktadır. Federasyon işletiminin başında FOM, RTI tarafından okunmaktadır. Şekil 3'de gösterildiği gibi FOM, nesne sınıfları ve etkileşim sınıfları olarak iki temel bileşenden oluşmaktadır.



Şekil 3 – Federasyon Nesne Modeli (Federation Object Model, FOM)

2.2.4. Etkileşim Sınıfları (Interaction Class)

Etkileşim sınıfları, belirli bir anda bir federeden diğer federelere RTI üzerinden gönderilen veri bütününe verilen isimlerdir. Etkileşim sınıfları, simülasyonda oluşan ve diğer simülasyonların (*federe*lerin) ilgilendiği bir oluşum ya da olaydır. Örneğin patlama olayı etkileşim kullanılarak diğer federelere iletilebilir. Bir federe etkileşimi gönderir (*Send Interaction*) ve ilgilenen diğer federeler etkileşimi alır (*Receive Interaction*). Etkileşim sınıflarında olay başlar ve biter, süreklilik içermez. Etkileşim sınıfları parametrelerden oluşmaktadır.

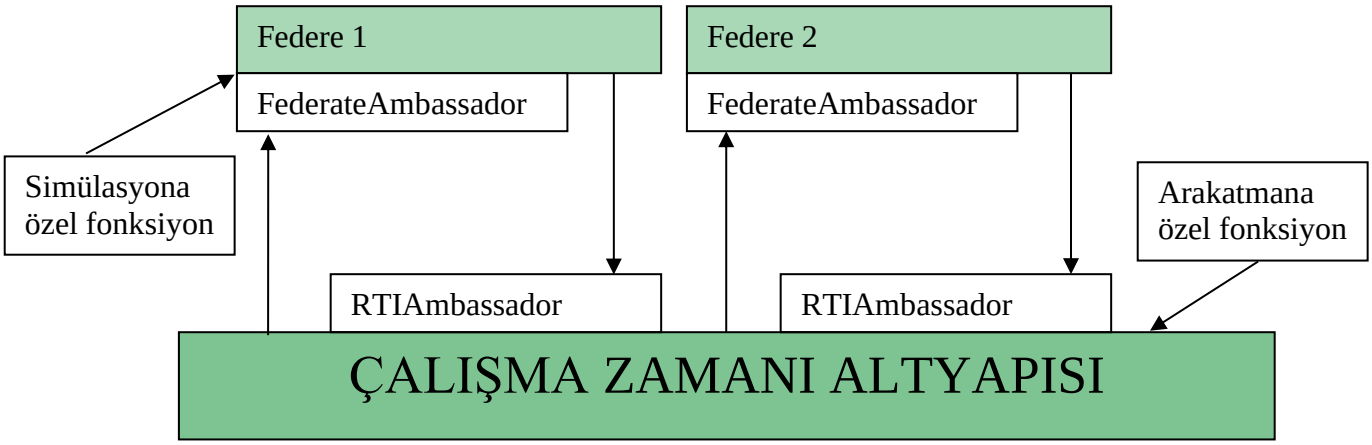
2.2.5. Nesne Sınıfları (Object Class)

Nesne sınıfları, simüle edilmiş varlıklardır. Birden fazla federenin ilgisi dahilinde tanımlanırlar. Etkileşim sınıflarından farklı olarak anlık değil, belirli bir süre boyunca kalıcılıklarını sürdürürler. OMT, nesne sınıflarını tanımlamaktadır. Etkileşim sınıflarının

parametreleri içermesi gibi, nesne sınıfları da öznitelikleri (*attribute*) içerir. Federeler, nesnelerin örneklerini (*instance of objects*) yaratabilirler. Her bir örnek farklı bir kimliğe (*identity*) sahiptir. Özniteliklerde o örnek için farklı kimliğe sahiptir. Federeler, bir nesne örneğinin durumunu, öznitelikleri güncelleyerek değiştirirler. Federeler birbirleriyle, RTI üzerinden etkileşim ve nesneler aracılığıyla haberleşirler. Her bir federe kendi içinde tuttuğu simüle varlıkları, FOM içinde tanımlanmış nesnelere dönüştürmek zorundadır.

2.2.6. Çalışma Zamanı Altyapısı (Run-Time Infrastructure, RTI)

RTI, bir federasyonda tüm federelere iletişim hizmetleri sağlayan bir arakatman yazılımıdır. Şekil 4 üzerinde görüldüğü gibi federeler birbirleri ile doğrudan konuşmazlar. RTI her bir federe için RTIAmbassador arayüzünü sunmaktadır ve federe bu arayüzü kullanarak RTI servislerini çağırmaktadır. Bu servisler federe tarafından başlatılan (*federate-initiated*) servisler olarak ifade edilmektedir. Her bir federe FederateAmbassador arayüzünü sunar. RTI bu arayüzü kullanarak federe ile olan etkileşimini yürütür. Bu servisler ise, RTI tarafından başlatılan (*RTI-initiated*) servisler olarak ifade edilir. FederateAmbassador ve RTIAmbassador kullanılarak çağrılan tüm servisler standartta tanımlanmıştır. FederateAmbassador soyut sınıftır ve federe bu soyut sınıftan türetilmek zorundadır.



Şekil 4 – FederateAmbassador & RTIambassador Arayüzü

2.3. Federasyon ve Federe Kuralları

HLA standardı federeler için 5 kural ve federasyonlar için de yine 5 kural olmak üzere, toplam 10 kural tanımlamaktadır. Bir simülasyon modelinin HLA uyumlu olduğunun söylenebilmesi için HLA standardında yer alan 10 kuralın hepsine uyması

yeterli bir koşul olacaktır. Başka bir deyişle eğer simülasyon modeli bu 10 kuralın hepsini sağlıyorsa HLA uyumlu olduğu kabul edilmektedir.

Federasyon kuralları aşağıdaki gibidir;

Kural 1: Federasyonlar, OMT'ye göre tanımlanmış FOM'ye sahip olmalıdır. OMT, RTI'nın ihtiyaç duyduğu bilgilerden daha fazlasını içerir. Bu ekstra bilgi, federasyon için tutarlılığı sağlamak için mühendisler tarafından kullanılır.

Kural 2: Bir federasyonda nesne örnekleri RTI yerine federelerde tutulmalıdır. Bu RTI'nın jenerik kalmasını sağlar. Bu beraberinde bir kısıtı da getirir. RTI hiçbir nesnenin özniteliğinin değerini saklamamalıdır.

Kural 3: Federasyon işletimi sırasında, federeler arasında alıp verilen ve FOM'de tanımlanan veri RTI aracılığı ile federelere ulaştırılmalıdır. Başka bir deyişle federeler birbirleri ile doğrudan haberleşemez. Bu kuralın amacı tekrar kullanılabilir simülasyon bileşenleri oluşturmaktır.

Kural 4: Federasyon işletimi sırasında federeler RTI ile Federe Arayüz Spesifikasyonu dokümanında belirtildiği üzere haberleşmelidir. Bu kuralın amacı farklı RTI sürümlerini kullanmanın yolunu açmaktır.

Kural 5: Federasyon işletimi sırasında bir nesnenin özniteliğine sadece tek bir federe sahip olabilir.

Federe Kuralları aşağıdaki gibidir.

Kural 6: Federeler, HLA Nesne Modeli Şablonu (OMT) dokümanına uygun olarak oluşturulmuş bir SOM'ye sahip olmalıdır.

Kural 7: Federeler, Simülasyon Nesne Modeli (SOM)'da belirtilen nesne özniteliklerini almalı ya da vermelidir. Benzer şekilde SOM'de belirtilen etkileşimi almalı ya da göndermelidir. Bu kurala uyulduğu takdirde, federe veri açısından HLA standardı ile uyumlu olur.

Kural 8: Federeler dinamik olarak bir özniteliğin sahipliğini alıp verebilirler. Bu bilgi SOM'de bulunmak zorundadır.

Kural 9: Federeler öznitelikleri yayınlarken koşulları (örneğin threshold) değiştirebilirler. Bu bilgi SOM'de belirtilmek zorundadır.

Kural 10: Federeler kendi lokal zamanlarını yönetebilirler.

2.4. Yüksek Seviye Mimari Servisleri

HLA servisleri 6 grup altında toplanmıştır. RTI, federeleri ve federeler arasındaki ilişkileri yönetirken bu 6 servis grubunu kullanmaktadır. Bu servis gruplarının isimleri Şekil 5'te verilmiştir.

Şekil 5 – HLA Servis Grupları

2.4.1. Federasyon Yönetimi (Federation Management)

Federasyon Yönetimi; federasyonun yaratılması, federasyona katılma, federenin federasyondan ayrılması, federasyonunun yok edilmesi, federasyon kayıt, yükleme ve senkronizasyon servislerinden oluşmaktadır. Toplam 24 servis bulunmaktadır.

2.4.2. Deklarasyon Yönetimi (Declaration Management)

HLA standardına göre federeler birbirlerine doğrudan veri gönderemezler ancak veriyi federasyon için erişilebilir hale getirebilirler. RTI bu veriyi ilgilenen federelere ulaştırmakla görevlidir. Bu işlem Deklerasyon Yönetimi servisleri ile gerçekleştirilir. Deklarasyon Yönetimi 12 servisten oluşmaktadır. Bu servisleri kullanarak federeler ilgilerini deklare ederler. Veriyi üretebilirler (*publish*) ya da veriyi tüketebilirler (*subscribe to*). RTI bu deklarasyonları kullanarak veriyi yönlendirir, değiştirir ve ilgi yönetimi yapar.

Yönlendirme, RTI'nın abonelik (*subscription*) bilgilerini kullanarak hangi federelerin veriyi alacağına karar vermesi ile gerçekleştirilir.

Değiştirme, verinin abonelik durumuna göre tekrar adlandırılması ile gerçekleştirilir. İlgi yönetimi, veriyi üreten (*publish*) federeler için yapılır. RTI, veriyi üreten federeye o veriye herhangi bir federenin abone olup olmadığını söyler. Aboneliğin olmadığı durumda federe veriyi üretmeyebilir.

2.4.3. Nesne Yönetimi (Object Management)

Nesne Yönetimi servisleri, verinin federeler arası alınıp verilmesinden sorumludur. Bu servisler ile etkileşimleri alıp göndermek mümkündür. Nesne örnekleri yaratılabilir ve bu örneklerin öznitelikleri güncellenebilir. Diğer federeler yine bu servisleri kullanarak nesne örneklerini keşfedebilir (*discover*) ve öznitelikleri alabilir. Nesne Yönetimi 19 servisten oluşmaktadır. Bu servisler verinin nasıl iletileceğini, veri güncelleme yöntemlerini belirlemek için kullanılabilir.

2.4.4. Sahiplik Yönetimi (Ownership Management)

HLA servislerini kullanarak bir nesnenin özniteliğinin sahipliğini başka bir federeye devretmek mümkündür fakat HLA'nın 5. ve 8. Kuralına göre, bir federe bir nesnenin özniteliğini değiştirebilmek için önce o nesneye sahip olmalıdır. Sahiplik Yönetimi grubunda 17 servis yer almaktadır.

2.4.5. Zaman Yönetimi (Time Management)

Federelerin zaman uyumunu sağlayan servislerdir [14]. Her bir federenin kendine ait iş parçacığı (*thread*) ile çalışması, federeler arasında taşınan mesajların sıralanması problemine neden olur. HLA Standardında mesajların sıralanması, mantıksal zaman (*logical time*) ile ifade edilir. Mantıksal zaman soyut bir kavramdır. HLA Zaman Yönetiminde yer alan 23 servis temelde 2 görevi yerine getirir.

- Her federenin kendi mantıksal zamanını diğer federelerle koordinasyon işinde ilerletmesine imkân tanır.
- Zaman etiketli mesajların dağıtımını kontrol eder. Bu dağıtım sayesinde herhangi bir federe diğer federelerden geçmiş oluşan bir mesajı talep etmek zorunda kalmaz. Başka deyişle kendi mantıksal zamanının öncesinde gerçekleşen bir mesajı almaz.

2.4.6. Veri Dağıtım Yönetimi (Data Distribution Management)

Veri Dağıtım Yönetimi servisleri federelerin üretici-tüketici ilişkilerini düzenler. Bu düzenlemeyi bölgeler üzerinden gerçekleştirir. Veriler için tanımlanan bölgelerde paylaşılması ile gerçekleşmektedir. Veri Dağıtım Yönetimi 12 servisten oluşmaktadır.

2.5. Yönetim Nesne Modeli (Management Object Model, MOM)

HLA yeni tanımlandığı dönemlerde, federelerin RTI hakkında yönetim verilerine erişmesi ve bazı durumlarda federasyonu denetlemesi gerekliliği ortaya çıkmıştır. RTI, federasyon ve federeler hakkında hangi bilgilerin gerekli olabileceği konusunda yapılan çalışmalar sonrasında, MOM ortaya çıkmıştır. MOM, standart tarafından belirlenen FOM'nin parçasıdır. Federasyon durumunu tanımlamak ve yönetmek için kullanılır. RTI, MOM'da tanımlanmış nesne ve öznitelikleri yaratır. Örneğin bir federe federasyona katıldığında, RTI Manager.Federate sınıfından bir örnek yaratır. Eğer bir federe federasyona kaç federenin katıldığını bilmek isterse Manager.Federate sınıfında abone olur ve gerekli bilgiyi alır. Sistem yönetimi, MOM sınıflarına abone olan bir yönetici federe ile bu şekilde sağlanabilir. MOM tüm federasyon için aynı olduğundan yönetici federe yeniden kullanılabilir.

2.6. Standarda Uygun Federe Geliştirimi

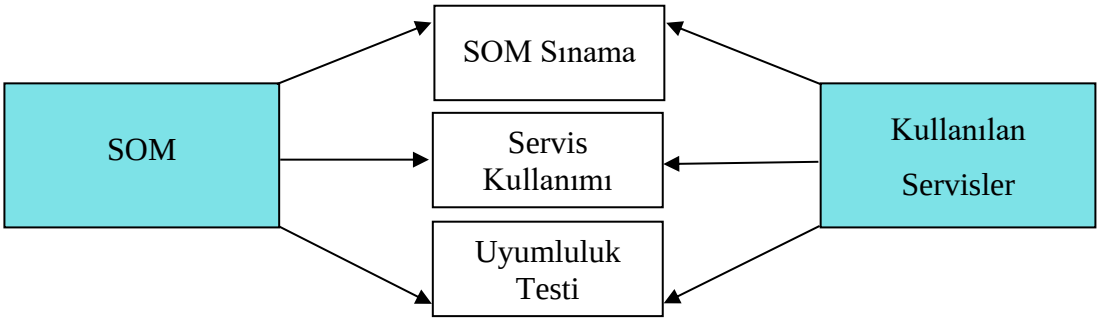
Federenin HLA uyumlu geliştirilmesi için, HLA standardında federeler tarafından uyulması gereken kurallar belirtilmiştir. Federelerin SOM ile bu kurallara uyması beklenir. Bu kurallara uymadığı durumda federe amaçları bilinmediği için birlikte çalışabilirlik faaliyetlerini yürütmek zorlaşmaktadır [15]. Bu nedenle HLA uyumluluğunun sınanması gerekmektedir.

3. FEDERE UYUMLULUĞUNUN SINANMASI

Bu tez çalışmasında federenin HLA uyumluluğunun sınanması için "Test Simülasyonu" olarak adlandırılan bir test aracı, Visual Studio 2015 tümleşik geliştirme ortamı kullanılarak, C++ dilinde geliştirilmiştir. Geliştirilen test aracı yardımı ile test senaryoları hazırlanmaktadır. Bu kapsamda federelerin HLA uyumluluğu HLA 1516 2000 standardına uygun ve standart kapsamında verilen HLA 1516 2000.1, 2000.2 ve 2000.3 dokümanlarına uygun olarak gerçekleştirilmiştir.

Federe kurallarında görüldüğü gibi Kural 10 dışındaki bütün kurallarda SOM'den bahsedilmektedir. Ancak bir simülasyon altyapısı için SOM kullanımı zorunlu değildir. Zorunlu olmaması nedeniyle kullanılmaması durumunda federeler HLA uyumlu olamamaktadırlar. Bir federenin HLA uyumlu olmaması ise HLA standardı tarafından vurgulanan birlikte çalışabilirlik ve tekrar kullanılabilirlik hedeflerinin gerçekleştirilmediği anlamına gelmektedir. Bu nedenle federenin HLA uyumluluğunu test etmek için SOM'nin kullanılması gerekmektedir.

Uyumluluğun sınanması için gerekli adımlar Şekil 6 üzerinde görülmektedir. Bu şekle göre herhangi bir federenin HLA uyumluğunun sınanması için birtakım girdilere ihtiyaç vardır. Bunlardan ilki SOM girdisidir. SOM'ye sahip federenin Gerçek Zamanlı Platform Referans FOM (Real-Time Platform Reference FOM, RPR-FOM) [16][17] uyumlu olması gerekmektedir. SOM verisi simülasyonda mevcut ise bu sınama için ilk girdi sağlanmış olmaktadır. Verilen bu girdi ile SOM Sınama adımı gerçekleştirilerek SOM verisinin standarda uyumluluğu denetlenir. İkinci test girdisi olan Kullanılan Servisler, federenin kullandığı HLA servislerinin listesidir. Bu girdi ile Servis Kullanımı Sınama adımında federenin kullanacağını beyan ettiği servislerin ve federenin kullanmasının zorunlu olduğu servislerin kullanılıp kullanılmadığı sorgulanır. Uyumluluk Testi sınama adımları ise HLA standardında bulunan 6 farklı servis grubunda yer alan servislerin ön koşullarını sağlaması ile kontrol edilir. Kontrol edilen servisler için istisna (exception) durumları değerlendirilir. Verilen iki veri girdisi sağlanırsa sınama adımları da başarılı şekilde işletilir.



Şekil 6 – Federe Uyumluluğunun Sınanması

3.1. SOM (Simulation Object Model) Sınama

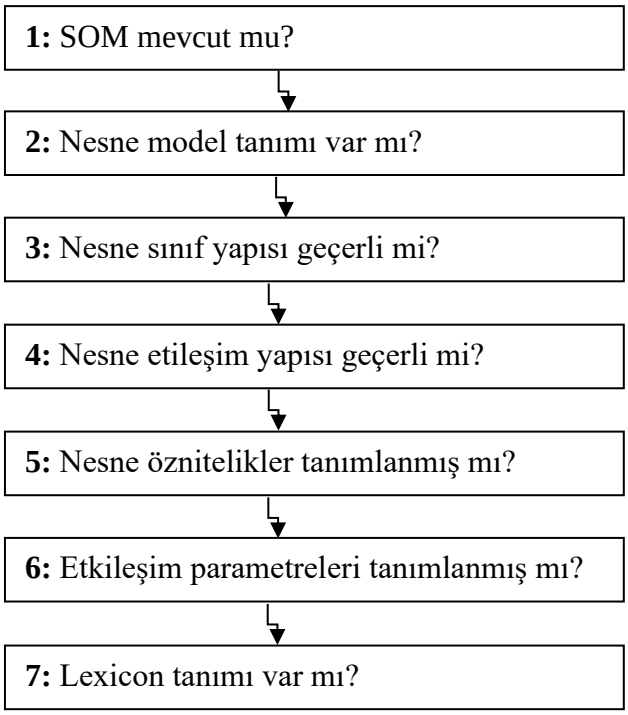
İlk sınama adımı federenin 2000.1 HLA kuralları ve 2000.3 OMT kurallarına uygunluğunun sınanması ile gerçekleştirilir. Bunun için SOM sınaması yapılır. Bu sınama adımı statiktir. Şekil 7'te verilen örnek SOM verisi okunarak SOM dosyasının içeriğinin OMT standardındaki kurallara uygunluğu test edilir.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE objectModel SYSTEM "hla.dtd"><objectModel DTDversion="1516.2" name="New FOM.xml" type="FOM" vers
<objects>
<objectClass name="HLAobjectRoot" sharing="Neither">
<attribute name="HLAprivilegeToDeleteObject" dataType="NA" updateType="NA" updateCondition="NA" ownership=
<objectClass name="GriddedData" sharing="PublishSubscribe" semantics="Depict global, spatially varying env
<attribute name="TotalValues" dataType="UnsignedLong3" updateType="Conditional" updateCondition="On change
<attribute name="RecordTotal" dataType="UnsignedShort4" updateType="Conditional" ownership="NoTransfer" sh
<attribute name="CoordinateSystem" dataType="EnvironmentDataCoordinateSystemEnum16" updateType="Conditiona
<attribute name="GridDataInfo" dataType="GridDataStructArray1" updateType="Conditional" updateCondition="C
<attribute name="SampleTime" dataType="UnsignedLongLong1" updateType="Conditional" updateCondition="On cha
<attribute name="RecordNumber" dataType="UnsignedShort4" updateType="Conditional" ownership="NoTransfer" s
<attribute name="GridAxisInfo" dataType="GridAxisStructArray1" updateType="Conditional" updateCondition="C
<attribute name="FieldNumber" dataType="UnsignedShort4" updateType="Conditional" ownership="NoTransfer" sh
<attribute name="ConstantGrid" dataType="EnvironmentGridTypeEnum8" updateType="Conditional" updateConditic
<attribute name="VectorDimension" dataType="HLAoctet" updateType="Conditional" updateCondition="On change'
<attribute name="GridID" dataType="EntityIdentifierStruct" updateType="Static" updateCondition="N/A" owner
<attribute name="Orientation" dataType="OrientationStruct" updateType="Conditional" updateCondition="On ch
<attribute name="EnvironmentType" dataType="EnvironmentTypeStruct" updateType="Static" updateCondition="N/
<attribute name="NumberOfGridAxes" dataType="HLAoctet" updateType="Conditional" updateCondition="On change
</objectClass>
<objectClass name="EmitterBeam" sharing="Subscribe" semantics="A sector of concentrated energy from a devi
<attribute name="BeamElevationCenter" dataType="Float1" updateType="Conditional" updateCondition="&gt; EE
<attribute name="PulseRepetitionFrequency" dataType="Float12" updateType="Conditional" updateCondition="&g
<attribute name="BeamIdentifier" dataType="HLAoctet" updateType="Static" updateCondition="N/A" ownership="
<attribute name="EventIdentifier" nameNotes="27" dataType="EventIdentifierStruct" updateType="Conditional"
<attribute name="EmissionFrequency" dataType="Float12" updateType="Conditional" updateCondition="&gt; EE F
<attribute name="PulseWidth" dataType="Float13" updateType="Conditional" updateCondition="&gt; EE PW THRSH
```

Şekil 7 – Örnek SOM Dosyası

SOM sınama işlemi için, Şekil 8 üzerinde yer alan adımlar uygulanır. 1 numaralı adımda yer alan SOM dosyası örneği Şekil 7 üzerinde yer almaktadır. SOM dosyası, federeler arasındaki etkileşimi/iletişimi açıklayan geçerli bir XML dosyası olmalıdır. Bölüm 2.2.4'te SOM dosyasında kullanılan etkileşimlerle ilgili, Bölüm 2.2.5'te ise nesne sınıfları ile ilgili bilgiler verilmiştir. Geçerli bir SOM dosyası mevcut ise 2 numaralı adım

kontrol edilmelidir. Şekil 7'de bulunan SOM dosyası birinci satırı üzerinde bir HLA nesne modeli (*objectModel*) yer almaktadır. HLA nesne modeli, model ismi, model tipi, model versiyonu, model referansı vb. gibi bilgiler barındırmalıdır. Eğer HLA nesne modeli içinde bulanan bir bilgi uygun değilse nesne modelinin ilgili bilgisine "NA" girilmelidir. 2 numaralı adımda yer alan nesne modeli kontrolü bu şekilde sağlanır. 3, 4, 5 ve 6 numaralı adımlarda, SOM dosyası üzerinde ikinci satırda ve daha sonraki satırlarda yer alan nesne sınıfları ve onların özniteliklerinin yapısı, etkileşim sınıfları ve onların parametrelerinin yapısı kontrol edilir. Nesne/Etkileşim/Öznitelik/Parametre isimlendirmeleri XML adlandırma kurallarına uygun olmalıdır. XML adlandırma kurallarında iki nokta üst üste ":" ve "hla" dizesiyle başlayan isimler veya (("H|h") ("L|l") ("A|a")) ile eşleşen dizeler dahil edilmemelidir. 7 numaralı adımda *lexicon* kontrolü gerçekleştirilir. Bu adımda SOM dosyasında bulunan nesne sınıfları, etkileşim sınıfları ve parametre sınıflarının her birinin içerisinde anlambilim (*semantic*) özelliği kontrol edilir. Eğer Şekil 8'da yer alan adımlar sonucunda HLA standardına uygun şekilde sonuçlar elde edilirse sınama işlemi başarılı kabul edilir.

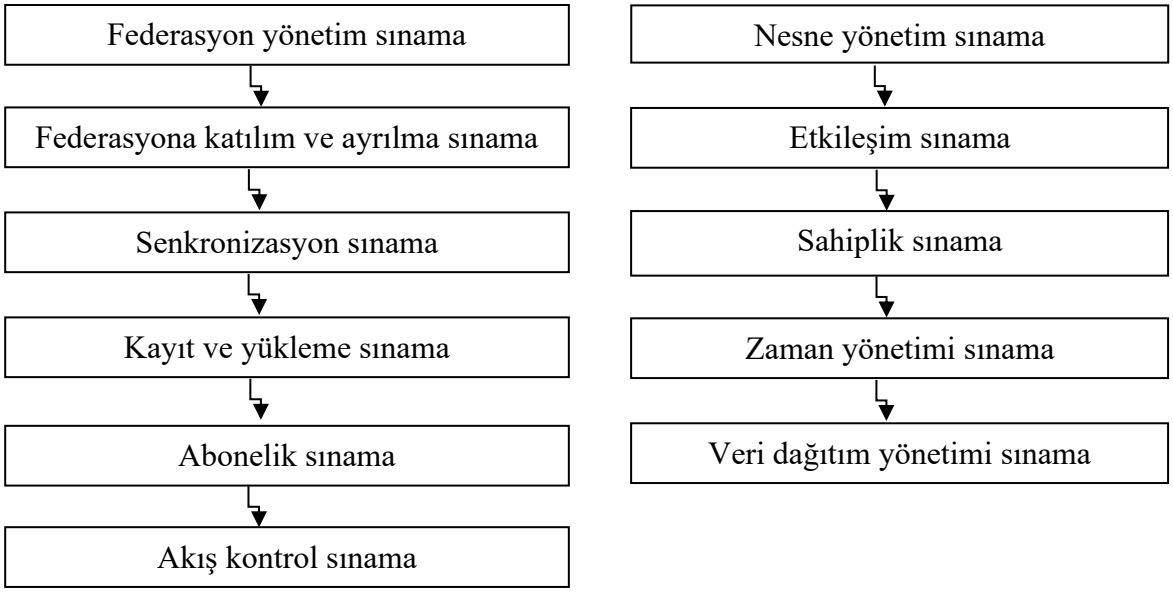


Şekil 8 – SOM Sınama Adımları

3.2. Servis Kullanımı

İkinci sınama adımı için geliştirilen test aracı kullanılarak bir test senaryosu hazırlanır. Bu test senaryolarının sınama adımları Şekil 9'de verilmiştir. Test senaryosu federenin kullandığını beyan ettiği servislerden oluşmaktadır. Sınaması yapılan federe ve

test federesi ile RPR-FOM verisi kullanılarak test senaryosu işletilir. Bu test sonrasında sınaması yapılan federenin beyan ettiği tüm servisler kullanılmış ise sınama adımı başarılı kabul edilir.



Şekil 9 – Servis Kullanım Aracı

Gerçekleştirilen bu sınama adımı aşağıdaki sorular ile yönetilir.

- Zorunlu olan servisler çağırılmış mı?
  - Federasyonu Yarat (Create Federation Execution)
  - Federasyonu Yoket (Destroy Federation Execution)
  - Federasyona Katıl (Join Federation Execution)
  - Federasyondan Ayrıl (Resign Federation Execution)
- Federenin kullandığını beyan ettiği diğer servisler çağırılmış mı?
  - Nesne Sınıfına Abone Ol (Subscribe Object Class)
  - Nesne Sınıfını Yayınla (Publish Object Class)

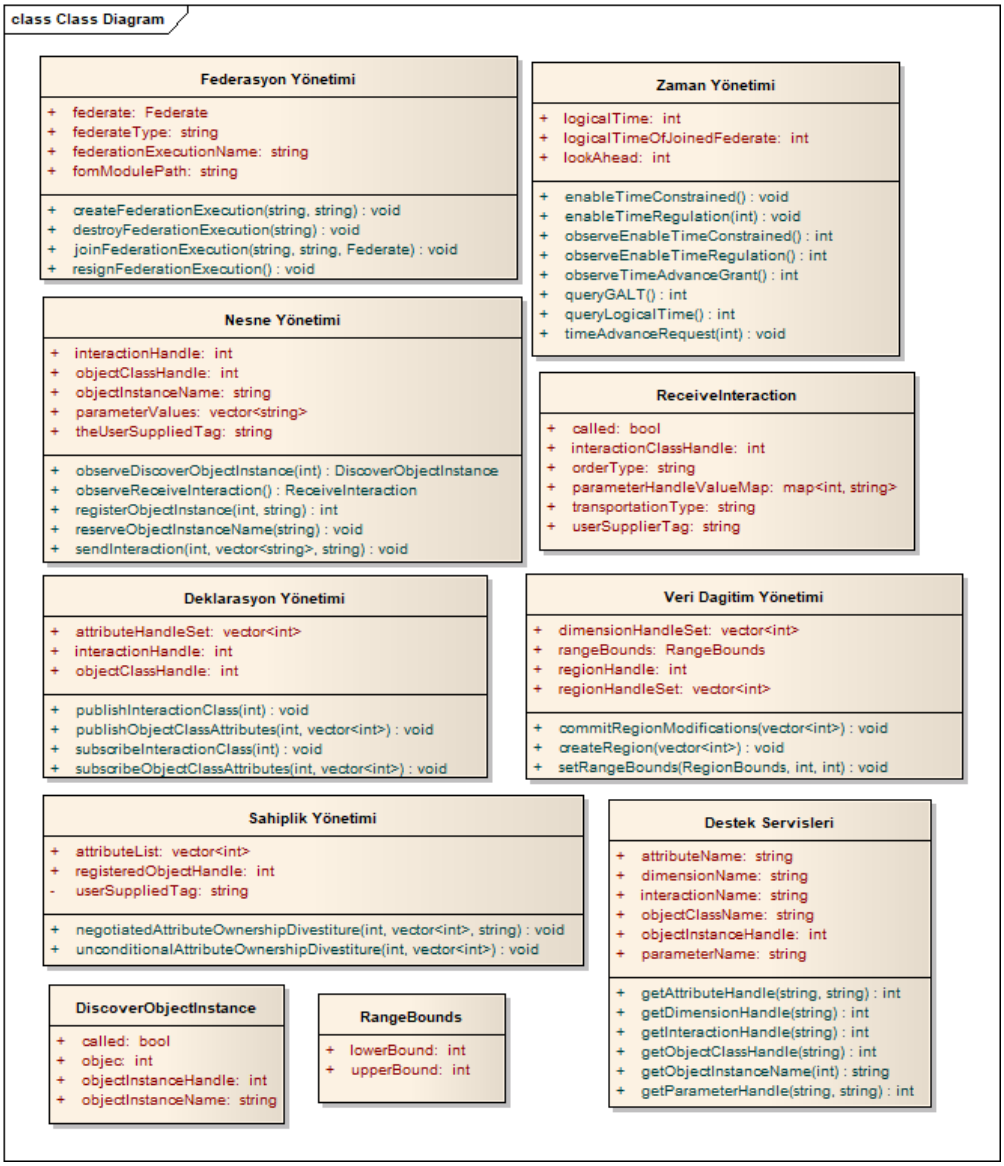
3.3. Uyumluluk Testi

Üçüncü sınama adımı ise SOM'de federe tarafından verilen bilgilerin sınanmasının denetlenmesidir. Bu sınama adımı 2. sınama adımı işletilirken RTI yazılımı vasıtasıyla toplanan veriler ile gerçekleştirilir. RTI yazılımı vasıtasıyla toplanan verilerin testleri Şekil 10'da yer almaktadır.

Şekil 10 – Uyumluluk Testi Adımları

- Yayınlamayı ve Abone Olmayı beyan ettiği sınıfları yayınlayıp abone olmuş mu?
- Sahipliği devredebileceğini beyan etmiş ise öznitelikleri devredebiliyor mu?
- Göndereceğini beyan ettiği etkileşimleri göndermiş mi?
- Güncelleyeceğini beyan ettiği öznitelikleri güncellemiş mi, nesneler yaratılmış mı?
- Abone olacağını beyan ettiği öznitelikleri almış mı, nesneleri keşfetmiş mi?
- Boyut (*dimension*) tanımı yaptığı öznitelikleri bölgelerle ilişkilendirmiş mi?

Uyumluluk testlerinde birkaç farklı adım gerçekleşmektedir. Gerçekleşen bu adımlarda servis gruplarında yer alan servisler ve ön koşulları anlatılmaktadır. Yapılan uyumluluk testlerinden bazı test senaryoları seçilerek örnek olarak verilmiştir. Verilen bu test senaryolarında kullanılan servisler için oluşturulan sınıf diyagramı Şekil 11 üzerinde yer almaktadır.



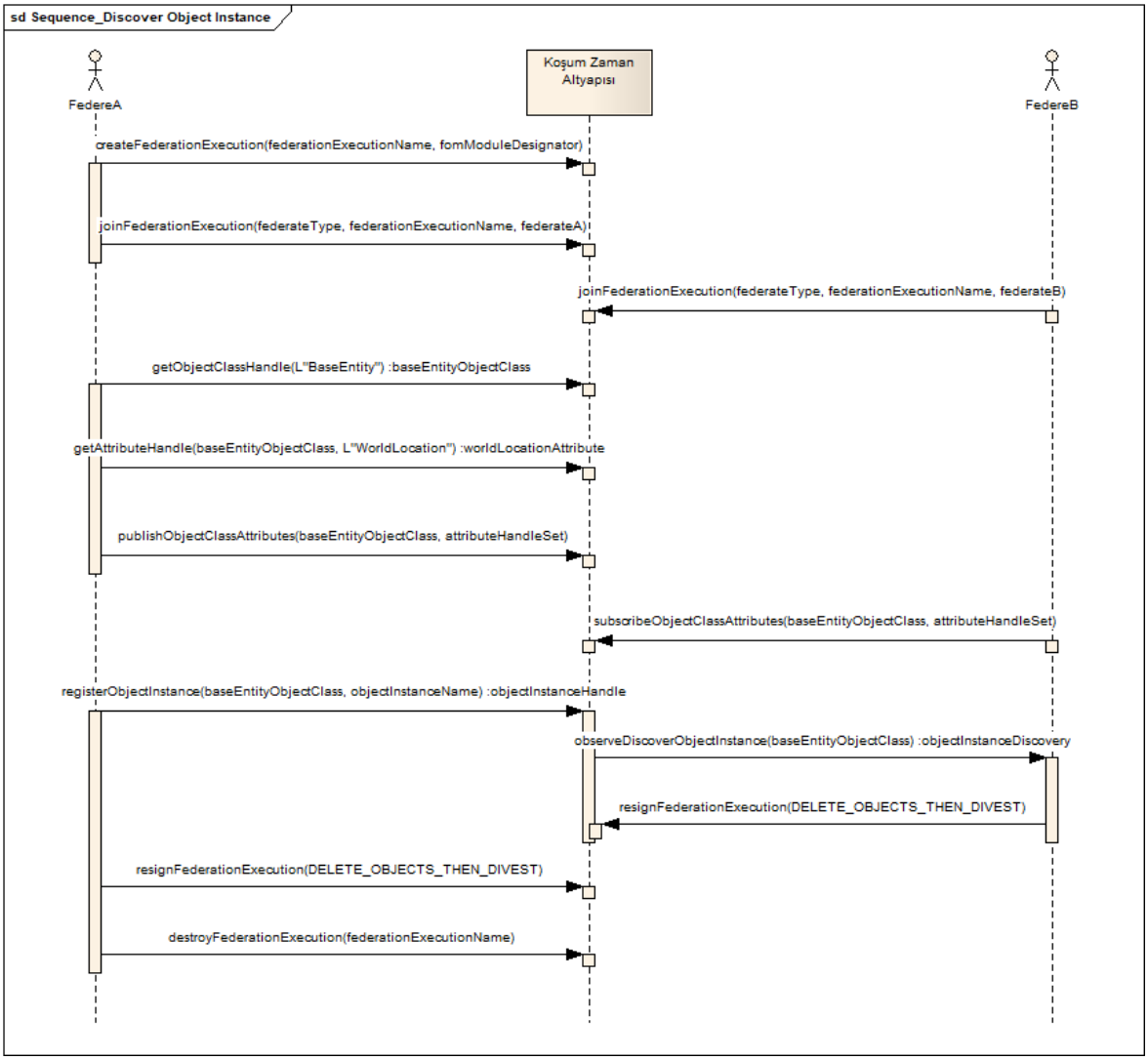
Şekil 11 – HLA Servisleri Sınıf Diyagramı

Standard, servis grupları için ön koşullar sunmaktadır. Bu ön koşulları sağlayarak uyumluluk testleri işletilmiştir. Örneğin bir federasyon yönetiminin işletilmesi için; federeler federasyona katılmadan önce federasyonun yaratılması gerekir. Federasyon yaratmak için (Create Federation Execution) servisi kullanılır. Bu servis federasyon adı ve Federasyon Yönetim Verisi (Federation Execution Data, FED (FOM)) dosyasının lokasyonunu parametre olarak alır. Bir federenin öncelikle federasyonu yaratması ve diğer federelerin federasyona katılması, federelerin çalışma sıralarının belirlenmesini gerektirir. Bunun yerine her federe federasyon yaratmaya çalışır. Önce çalışan federe federasyonu yaratır, diğer federeler istisna (*exception*) alır. Böylece sıralama sorunu çözülmüş olur. İstisna alan federeler istisnayı dikkate almazlar. Federasyon yaratıldıktan sonra, federeler (Join Federation Execution) servisi ile federasyona katılırlar. Federe bu servisi, federasyon adı ve tip parametreleri ile çağırmalıdır. Bu çağrı sonrası federe, federe belirteci (*federate*

designator) alır. Federasyon işletimi öncesi tüm federelerin federasyona katılma zorunluluğu yoktur. Bir federe herhangi bir anda federasyona katılabilir. Benzer şekilde bir federe herhangi bir anda federasyondan ayrılabilir (*resign*). Federasyonu sonlandırmak için tüm federelerin federasyondan ayrılması gerekir. Bu ayrılma işlemi (Resign Federation Execution) servisi ile gerçekleştirilir. Bu servis çağrısından sonra federe (Destroy Federation Execution) servisi dışında hiçbir servisi çağıramaz. (Destroy Federation Execution) servisi federasyonu sonlandırır. Bir federasyonun yaratılması ve sonlandırılmasındaki temel koşullar bunlardır.

3.3.1. Nesne Testleri

Nesne Testleri için bazı kod örnekleri EK 1 üzerinde verilmiştir. Verilen kod bloğu için oluşturulan sıralama diyagramı (*sequence diagram*) ise Şekil 12 üzerinde yer almaktadır. Kod karmaşıklığını ortadan kaldırmak adına daha sonraki örneklerin anlatılmasında sıralama diyagramları kullanılmıştır.



Şekil 12 – Nesne Testleri Örneği (Discover Object Instance)

Buradaki sıralama diyagramına göre işlemler şu şekilde gerçekleşmektedir:

- Görüldüğü gibi FedereA önce federasyon yaratmaktadır (Create Federation Execution), ardından federasyonu yaratan FedereA ve başka bir federe olan FedereB federasyona katılmaktadır (Join Federation Execution). Her iki federe aralarındaki iletişimi sağlamak için RTI kullanılmaktadır.
- FedereA, bir nesne sınıfının özelliklerini yayınlamak istediğini RTI'ya belirttikten sonra nesne yaratabilir (Register Object Instance).
- FedereA belirlediği nesne sınıfını yayınlamak için nesne sınıfının doğru belirtecine (*Object Class Handle*) ve özniteliklerinin doğru belirtecine (*Attribute Handle*) ihtiyaç duyar. Nesne sınıfı doğru belirteci (Get Object Class Handle) servisi ile alınmalıdır.

- Nesne sınıfının istenilen özniteliğinin doğru belirtecini almak için ise (Get Attribute Handle) servisi kullanılmalıdır.
- FedereA belirlediği nesne ve özniteliklerinin doğru belirtecleri ile nesne sınıfını yayınlayabilir (Publish Object Class Atributes).
- Belirtilen nesneyi keşfetmek isteyen FedereB nesneye ve özniteliklerinin doğru belirtecine abone olmalıdır (Subscribe Object Class).
- Nesne ile veri göndermenin ilk koşulu, bir federenin yeni bir nesne örneği yaratmasıdır (Register Object Instance).
- Nesne örneği yaratılması RTI'ya yeni bir unsurun federasyona girdiğini anlatmaktadır. RTI yaratılmış nesneyi nesne adı (*Object Name*) ve nesne belirteci (*Instance Handle*) ile verir. Federe bunun için bir isim seçebilir ya da isim RTI tarafından da verilebilir. RTI'nın verdiği isim HLA ile başlar. Federe'nin verdiği isim HLA ile başlayamaz. İsimler federasyon için tekil olmak zorundadır.
- Nesne yaratımı (Register Object Instance) servisi ile yapılır. Servis nesne sınıf belirteci (*Object Class Handle*) parametresini alır. Seçimli olarak nesne ismi (*Instance Name*) parametre olarak verilebilir. İsmi RTI verdi ise bu isime (Get Object Instance Name) servisi ile erişmek mümkün olur. Nesne yaratımı sonrası nesne örnek belirteci (*Object Instance Handle*) geri döndürülür. Bu belirteç RTI ve federe için tekildir. FedereA nesne ile ilgili tüm servis çağrılarını bu belirteç üzerinden gerçekleştirir. Bu belirteç başka bir federeye geçirilemez. Diğer federeler aynı isim kullanımlarına karşın aynı belirteci kullanmazlar.
- Belirlenen özniteliklere ya da özniteliklerin tamamına abone (*subscribe*) olan FedereB, yeni yaratılan nesneyi keşfetme işlemi gerçekleştirmektedir (Discover Object Instance).
- FedereA ve FedereB federasyondan ayrılır (Resign Federation Execution).
- Federasyon ismi vererek FedereA tarafından federasyon yok edilir (Destroy Federation Execution).

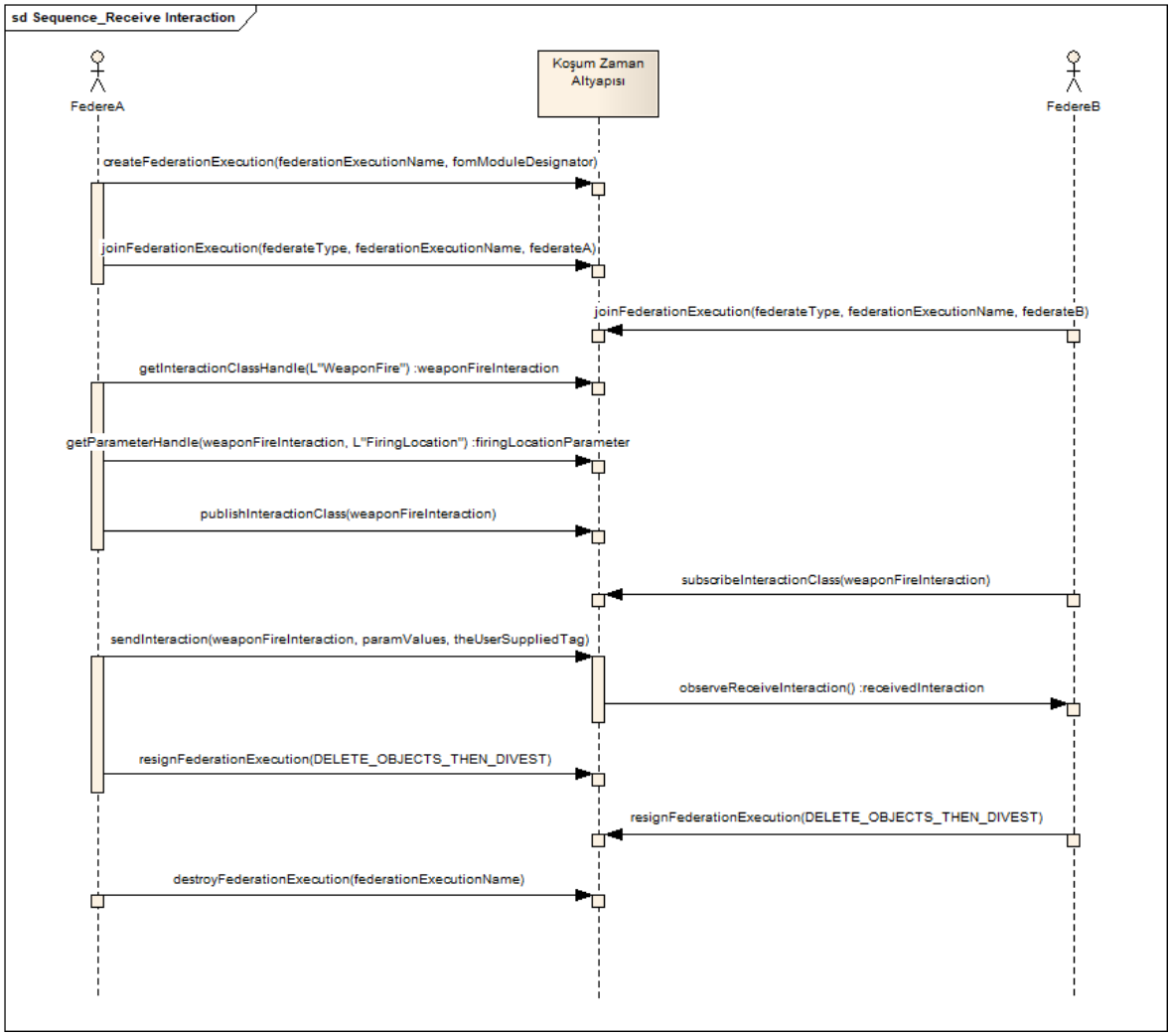
3.3.2. Etkileşim Testleri

Etkileşim Testleri de Deklarasyon Yönetimi servis grubunda bulunan nesne testleri kısmında detaylıca anlatılan yayınlama (*publication*) ve abonelik (*subscription*) servisleri tarafından kontrol edilmektedir. Bu yöntemle simülasyon bileşenlerinin birbirlerine gereksiz mesaj göndermeleri engellenebilir. SOM dosyasında yer alan etkileşimlerin abone

ol/yayınla mekanizması ile uygulanan testler olarak gerçekleştirilmiştir. Etkileşim (*interaction*) gönderme (Send Interaction) ve alma (Receive Interaction) Nesne Yönetimi servis grubunda yer alan servislerdir.

Etkileşim Testleri için oluşturulan diyagram Şekil 13 üzerinde yer almaktadır.

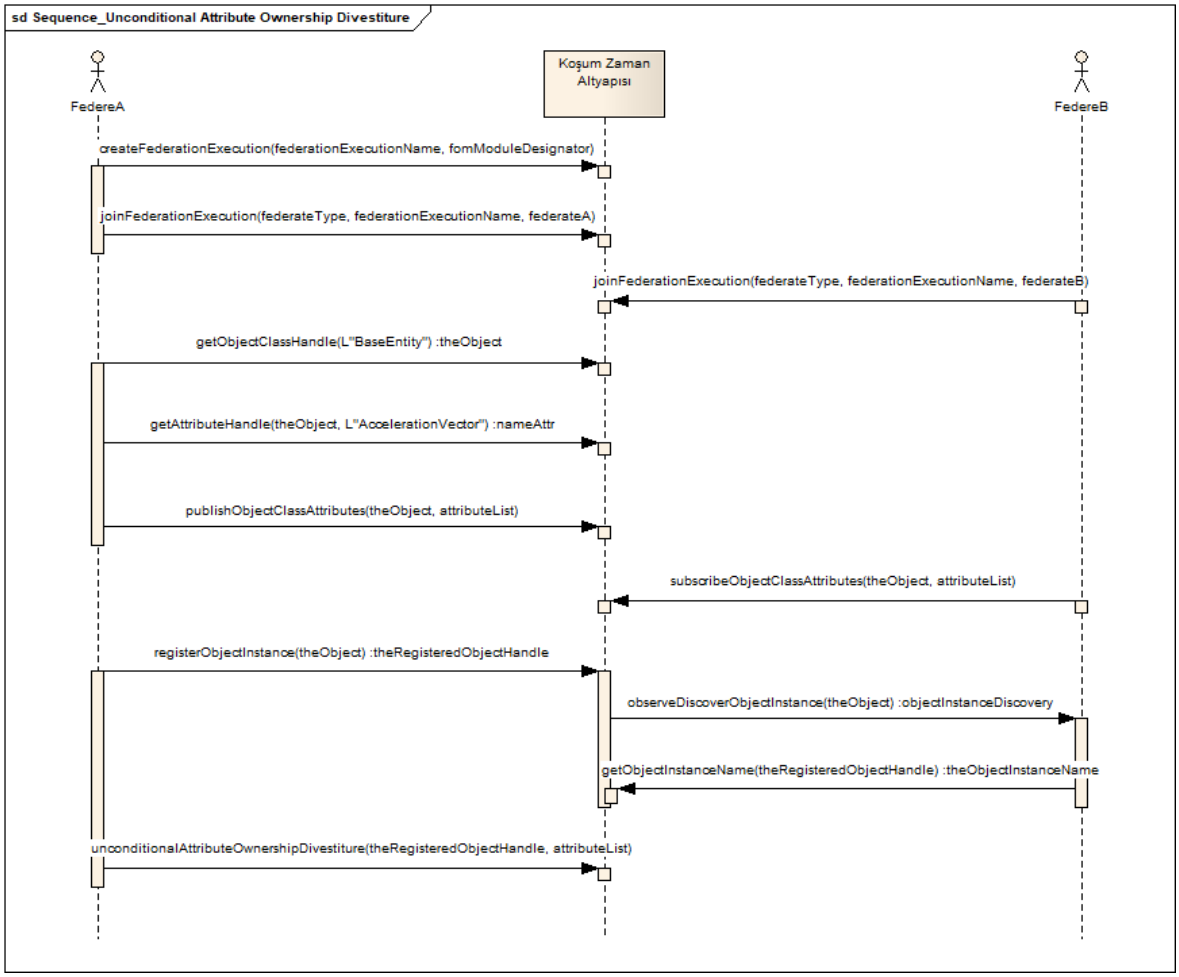
- FedereA önce federasyon yaratmaktadır (Create Federation Execution), ardından federasyonu yaratan FedereA ve başka bir federe olan FedereB federasyona katılmaktadır (Join Federation Execution).
- FedereA, bir etkileşim sınıfının özelliklerini yayınlamak istediğini RTI'ya bildirmelidir. FedereA belirlediği etkileşim sınıfını yayınlayabilir (Publish Interaction Class).
- Belirtilen etkileşimi almak isteyen FedereB etkileşime abone olmalıdır (Subscribe Interaction Class).
- FedereA etkileşimi (Send Interaction) servisi kullanarak gönderir. Federe etkileşimi göndereceğini önceden RTI'ya bildirmelidir. Bu bildirim sınıf bazında yapılır. Tek tek parametre bildirmek gerekmez.
- (Send Interaction) servisine tüm parametreleri vermek gerekmez. Bununla birlikte tek bir (Send Interaction) servisi ile gönderilen parametreler, tek bir (Receive Interaction) servisi ile alınır. (Send Interaction) servisi seçimli olarak mantıksal zaman alabilir. Bu durumda servis *retraction* belirteci üretir. (Send Interaction) ayrıca kullanıcı etiketi (*User Tag*) alabilir.
- FedereA ve FedereB federasyondan ayrılır (Resign Federation Execution).
- Federasyon ismi vererek FedereA tarafından federasyon yok edilir (Destroy Federation Execution).



Şekil 13 – Etkileşim Testleri Örneği (Receive Interaction)

3.3.3. Sahiplik Testleri

Sahiplik Testlerinde sahipliği devretmenin üç farklı yolu vardır. İlk olarak test edilecek federe FOM'de yer alan özniteliğinin sahipliğini koşulsuz olarak hemen elden çıkarabilir. Bunun için başka bir federe nesneyi alacak mı diye beklemesine gerek yoktur. Federe bu sahipliği elden çıkarır ve artık bu öznitelik sahipsiz olur (Divest Attribute Ownership Unconditionally). Bu servis için adımlar Şekil 14'de yer almaktadır.

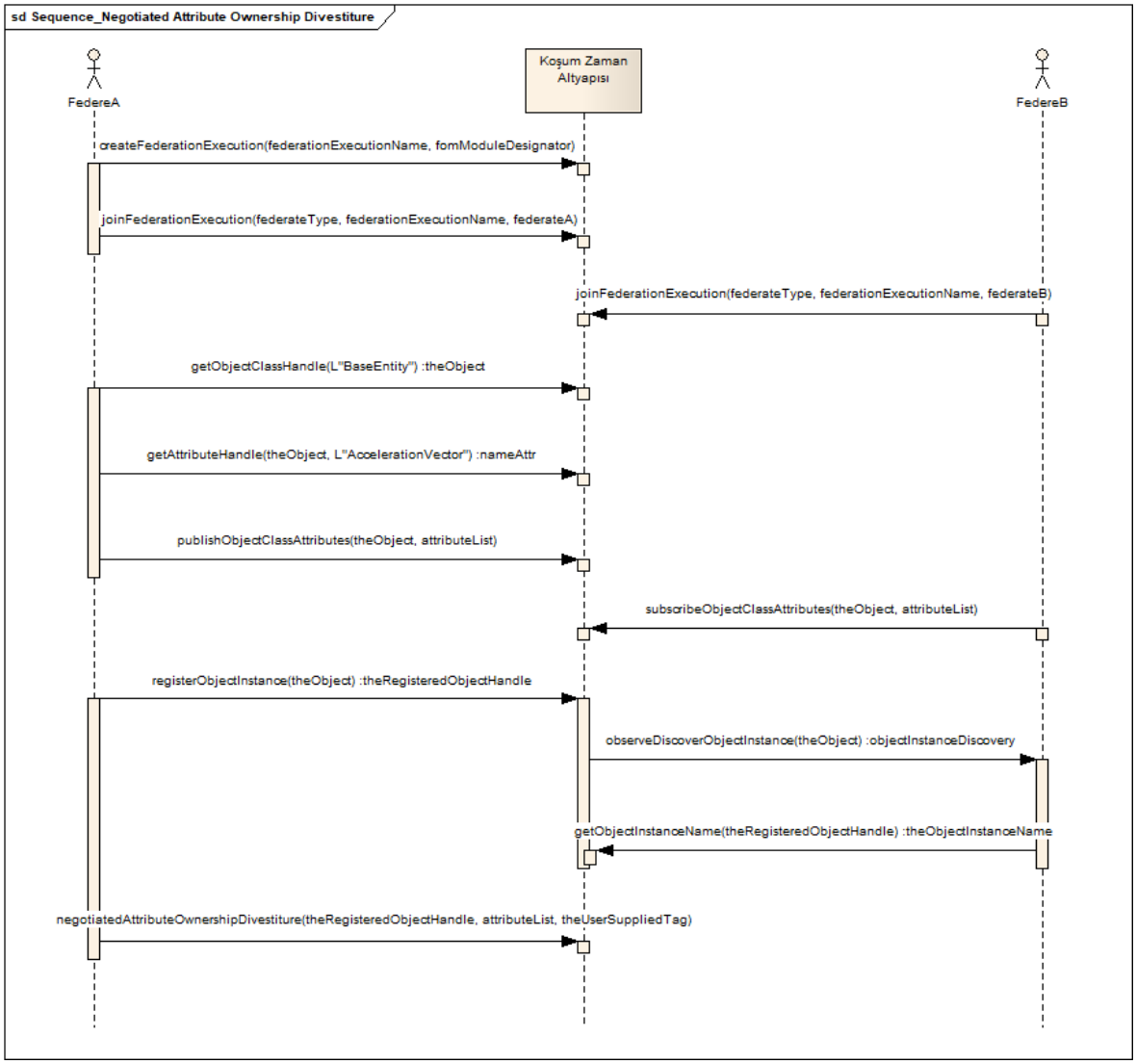


Şekil 14 – Sahiplik Testleri Örneği (Unconditional Attribute Ownership Divestiture)

- FedereA önce federasyon yaratmaktadır (Create Federation Execution), ardından federasyonu yaratan FedereA ve başka bir federe olan FedereB federasyona katılmaktadır. (Join Federation Execution).
- FedereA belirlediği nesne sınıfını ve özniteliklerini yayınlayabilir (Publish Object Class Atributes).
- Belirtilen nesneyi keşfetmek isteyen FedereB nesneye ve özniteliklerine abone olmalıdır (Subscribe Object Class).
- FedereA göndermek istediği nesne sınıfının örneğini yaratmalıdır (Register Object Instance). Yaratma işlemi tamamlandığında, FedereA, belirtilen nesne ve belirtilen özniteliğine sahip olmuş olur.
- FedereB ise keşfetme geri dönüş servisi (Discover Object Instance) ile bu değişikliklerden haberdar olur. Yaratma işleminden sonra sahipliği alan FedereA, sahipliği devredebilir hale gelir.
- FederateA, sahipliği koşulsuz devretmek istediğinde (Unconditional Attribute Ownership Divestiture), sahiplik anında RTI'ya geçer ve FederateA artık

belirtilen nesnenin belirtilen özniteliğini güncelleme sorumluluğundan feragat etmiş olur. RTI, FedereB'ye artık sahipsiz olan nesnenin özniteliğini önerir.

İkinci bir seçenek olarak test edilecek federe FOM'deki özniteliğinin sahipliğini elden çıkarmak istediğini RTI'ya bildirir. RTI bu isteği diğer federelere haber verir. Diğer federeler tarafından bu özniteliğe sahip olmak isteyen bir federe olursa RTI devir işlemini tamamlar ve tüm federelere haber verir (Divest Attribute Ownership By Negotiation). Bu servis için örnek servis testi Şekil 15'te yer almaktadır.



Şekil 15 – Sahiplik Testleri Örneği (Negotiated Ownership Divestiture)

- FedereA önce federasyon yaratmaktadır (Create Federation Execution), ardından federasyonu yaratan FedereA ve başka bir federe olan FedereB federasyona katılmaktadır. (Join Federation Execution).
- FedereA belirlediği nesne ve özniteliklerini yayınlayabilir (Publish Object Class Atributes).

- Belirtilen nesneyi keşfetmek isteyen FedereB nesneye ve özniteliklerine abone olmalıdır (Subscribe Object Class).
- FedereA göndermek istediği nesne sınıfının örneğini yaratmalıdır (Register Object Instance). Yaratma işlemini FedereA başlatır. Yaratma işlemi tamamlandığında, FedereA, belirtilen nesnenin belirtilen özniteliğine sahip olmuş olur.
- FedereB ise keşfetme geri dönüş servisi (Discover Object Instance) ile bu değişikliklerden haberdar olur.
- Yaratma işleminden sonra sahipliği alan FedereA, sahipliği pazarlıklı bir şekilde devredebilir hale gelir (Negotiated Attribute Ownership Divestiture). Sahipliği koşullu devretme servisi çağırıldığında, RTI, sahipliği devredilmek istenen nesnenin özniteliğini öncelikle FedereB'ye önerir. Eğer FedereB sahipliği almak isterse, FedereA, sahipliği devretme işlemini başlatabilir ve nesnenin özniteliğini güncelleme sorumluluğundan kurtulur.
- FedereB, RTI aracılığı ile sahipliği devralır ve o andan itibaren nesnenin özniteliğini güncelleme sorumluluğundadır.

Son olarak test edilecek federe özniteliğini elden çıkarmak istediğini diğer federelere bildirir almak isteyen olursa devri gerçekleştirir (Divest Attribute Ownership If Wanted).

Burada sahipliği devretmek için test edilen adımlar şunlardır:

Her test için;

- Öznitelik yayınlanmış mı?
- Nesne yaratma (*register*) işlemi yapılmış mı? (Bu adımda *register* eden federe özniteliğin sahipliğini almış demektir)
- Sahiplik, devretmek isteyen federede mi?
- Her notification servisi için geri dönüş var mı bakılır.

Sahiplik almak için test edilecek durumlar ise şunlardır:

Her test için;

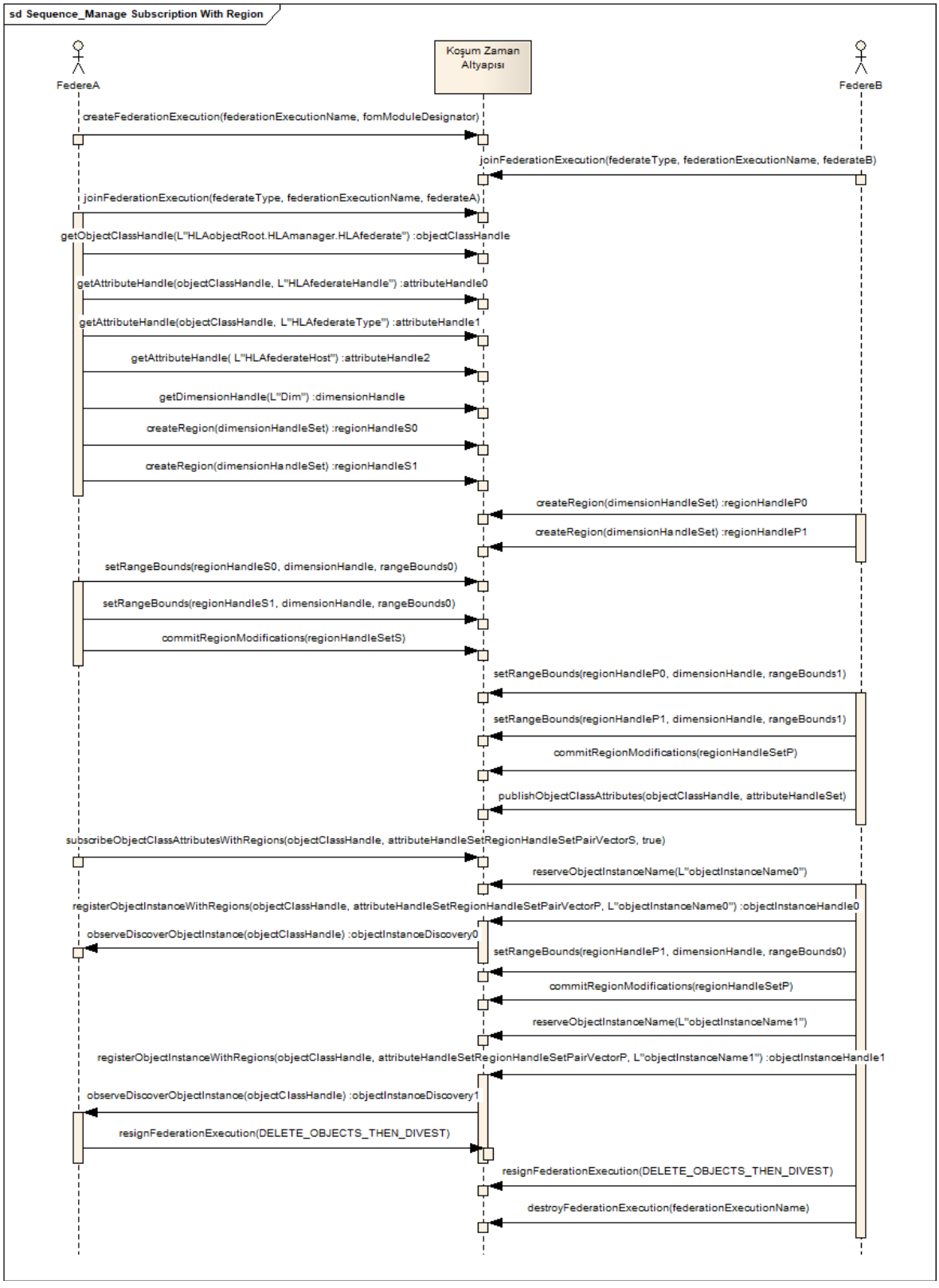
- Öznitelik yayınlanmış mı?
- Nesne yaratma (*register*) işlemi yapılmış mı?
- Sahipliği devralmak isteyen federe, almak istediği özniteliklere sahip olmamalı kontrolü (Olan bir öznitelik tekrar alınamaz)
- Her notification servisi için geri dönüş var mı bakılmalıdır.

3.3.4. Veri Dağıtım Testleri

Veri Dağıtım Testleri verilerin tanımlanan belirli bir alan için paylaşılması ile gerçekleşmektedir. Burada güncelleme bölgesi (*Registration Region*) ve abonelik bölgesi (*Subscription Region*) isimli iki ayrı bölge yaratılır. Bölgelerin alt ve üst limitlerinin ne olduğu RTI'ya bildirilir. Bu test üzerinde güncelleme bölgesinin alt limit 2, üst limit ise 4 olarak verilir. Abonelik bölgesinin ise alt limiti 5, üst limiti 7'dir. Güncelleme ve abonelik bölgeleri kesişmediği için test uygulaması federesinin yaptığı güncellemeler test edilecek federe tarafından alınmamaktadır. Daha sonra güncelleme bölgesinin alt limiti 2, üst limit 6 olarak güncellenerek RTI'ya bildirilir. Bu güncelleme sonucunda güncelleme bölgesi ve abonelik bölgesi 5 ve 6 aralığında kesiştiği için test uygulaması federesinin yaptığı bu bölge içinde bulunan güncellemelerin test edilecek federe tarafından alındığı görülür. Benzer bir örnek detaylandırılarak Şekil 16 üzerinde verilmiştir.

- FedereA önce federasyon yaratmaktadır (Create Federation Execution), ardından federasyonu yaratan FedereA ve başka bir federe olan FedereB federasyona katılmaktadır. (Join Federation Execution).
- İlk olarak FedereA (Get Dimension Handle) servisi ile FOM'de yer alan bir boyutunun en büyük ve en küçük değerlerini almaktadır.
- (Create Region) servisine alınan bu boyut doğru belirteçleri listesi verilerek 4 farklı bölge yaratılmıştır.
- Ardından (4,5) ve (6,7) aralıkları belirlenmelidir. (Set Range Bound) servisi kullanılarak belirlenen (4,5) aralığında 2 farklı abone olma ve (6,7) aralığında 2 farklı yayınlama alanı ayarlanmalıdır. FedereB belirlediği nesne ve özniteliklerini yayınlanmalıdır (Publish Object Class Atributes).
- Belirtilen nesneyi almak isteyen FedereA nesneye ve özniteliklerine abone olmalılıdır (Subscribe Object Class).
- FedereB göndermek istediği nesne sınıfının örneğini kendi belirlediği isimle (Reserve Object Instance Name) yaratmalıdır (Register Object Instance).
- Belirlenen özniteliklere ya da özniteliklerin tamamına abone (*subscribe*) olan FedereA, yeni yaratılan nesneyi keşfetme işlemini kesişen alan olmaması sebebiyle gerçekleştirememektedir (Discover Object Instance).
- Ardından (4,5) aralığında yayınlama alanı ayarlanmalıdır (Set Range Bound). Belirlenen nesne yaratılmalı ve sırasıyla keşfedilmelidir.

- Belirlenen nesne, nesnenin özniteliklerin bazıları ya da tamamına abone (*subscribe*) olan FedereA, yeni yaratılan nesneyi keşfetme işlemini kesişen bir alan olması sebebiyle keşfedebilmektedir.
- FedereA ve FedereB federasyondan ayrılır (Resign Federation Execution).
- Federasyon ismi vererek FedereA tarafından federasyon yok edilir (Destroy Federation Execution).



Şekil 16 – Veri Dağıtım Testleri Örneği (Manage Subscription With Region)

3.3.5. Zaman Yönetimi Testleri

Zaman Yönetimi Testlerinde RTI, bir federenin zaman yönetimine katılacağı zamanı seçmesine izin verilir. Bir federe zaman kısıtlı olabilir (*time-constrained*), bu durumda kendi mantıksal zaman ilerlemesi diğer federeler tarafından kısıtlanır. Bir federe zaman kontrollü olabilir (*time regulating*), bu durumda mantıksal zaman ilerlemesi diğer federeleri etkiler. Bir federe hem zaman kısıtlamalı hem de zaman kontrollü olabilir ve ya hiçbiri olmayabilir. Bir çok federe zaman yönetimine tabi değildir. Federasyonun amaçlarına ve federenin ihtiyaçlarına bağlı olarak farklı seçimler yapılır. Hem düzenleyici hem de kısıtlı olan federelerin yerel saatlerini, federenin değeri belirli federelerin yerel saatinden daha az olan zaman damgalı bir olay alabilecek şekilde ilerletmesine izin verilir. Genellikle kullanımda federeler hem zaman kısıtlı hem de zaman kontrollü olurlar.

Federeler kendi istekleriyle mantıksal zamanlarını (*logical time*) ilerletebilirler. Örneğin FederaA 0. zaman adımında, FedereB 10. zaman adımındayken FedereA zaman adımını 2 olarak güncellemek isteyebilir. Bu isteği RTI'ya iletilir. RTI mevcut olan şartlara bakarak FedereA'ya izin verir ya da vermez. İzin vermesi durumunda FedereA'nın zaman adımını 2 olarak güncellenir. Benzer bir örnek detaylandırılarak Şekil 17 üzerinde verilmiştir.

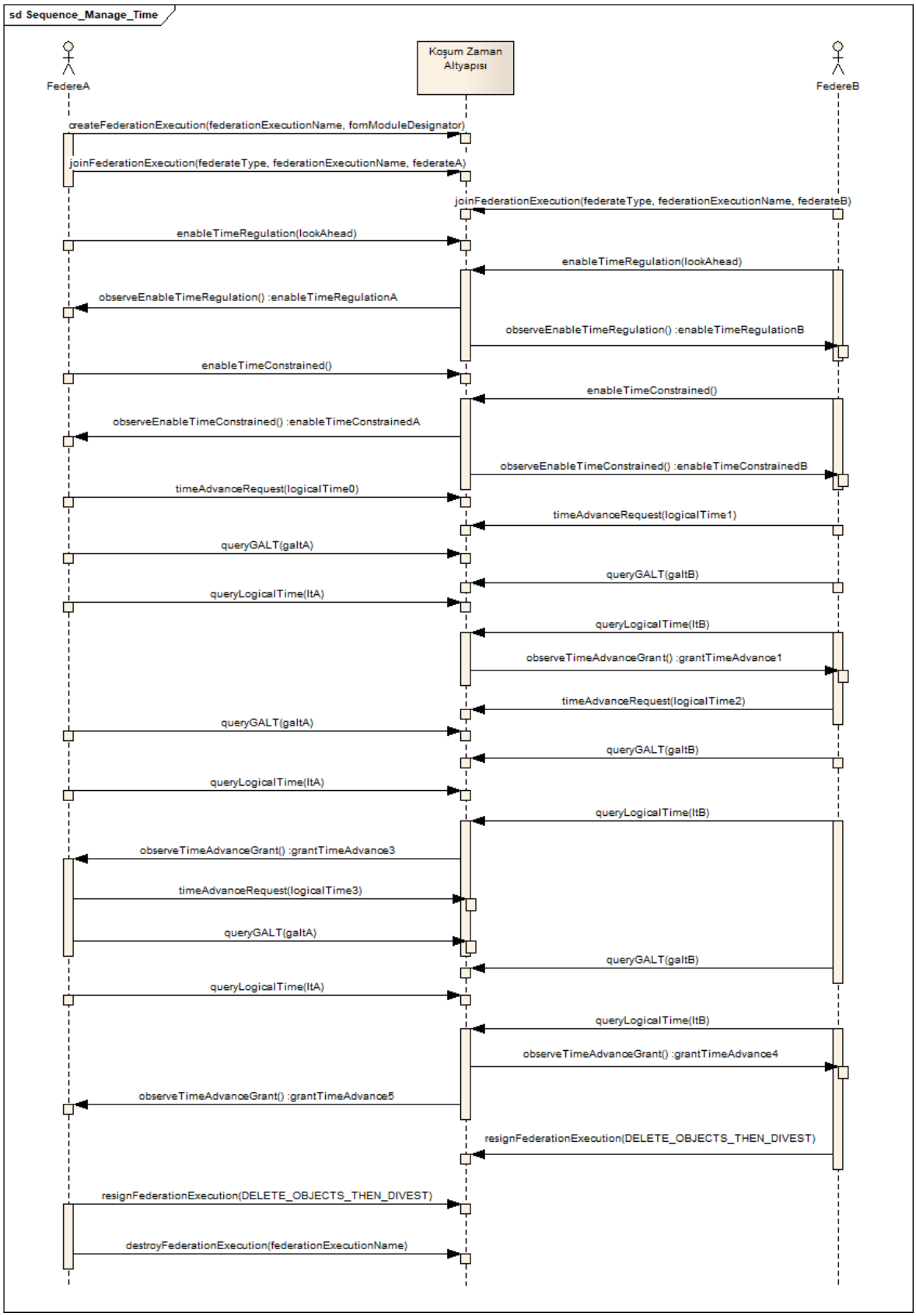
- FedereA önce federasyon yaratmaktadır (Create Federation Execution), ardından federasyonu yaratan FedereA ve başka bir federe olan FedereB federasyona katılmaktadır. (Join Federation Execution).
- İlk olarak sırasıyla FedereA ve FedereB zaman kontrollü bir federe olacağını RTI'ya ilerleme değeri (*lookahead)* vererek bildirir (Enable Time Regulation). FedereA zaman kontrollü olurken, verilen ilerleme değeri (lookahead) ile sahip olunan mantıksal zaman (logical time) değeri arasında mesaj yollanmayacağının garantisi verir.
- FedereA ve FedereB, RTI tarafından zaman kontrollü federe olarak kabul edilir (Time Regulation Enabled).
- Sırasıyla FedereA ve FedereB zaman kısıtlı federe olmak istediklerini RTI'ya bildirir (Enable Time Constrained).
- FedereA ve FedereB, RTI tarafından zaman kısıtlı federe olarak kabul edilir (Time Constrained Enabled).
- Başlangıçta FedereA ve FedereB 0. zaman adımındadır. FedereA (Time Advance Request) sevisini kullanarak zaman adımını 5'e ilerletmek isteğini RTI'a bildirir.

RTI, federasyonda bulunan bütün federelerin mevcut durumuna, federelerin isteklerine ve ilerleme değerlerine bakarak hesaplanan, mevcut en büyük mantıksal zaman (*greatest available logical time, GALT*) değeri ile ilerlenmek istenen zaman adımını karşılaştırır. Eğer ilerlenmek istenen zaman adımı GALT değerinden büyükse istek reddedilir ve federenin bu isteği tutulur. GALT değeri federe tarafından ilerlenebilecek en büyük sayıyı ifade etmektedir. GALT değeri federe tarafından gerçekleştirilen zaman adımı ilerleme isteklerinde sürekli olarak hesaplanmaktadır.

- FedereA tarafından gidilmek istenen 5. zaman adımı, GALT değerinden büyük olduğu için reddedilir. FedereA 0. zaman adımından ilerletilmez fakat RTI tarafından bu istek federe istek listesinde tutulur. Yeni GALT değeri hesaplanır.
- FedereB zaman adımını 1'e ilerletmek istediğini RTI'ya bildirir (Time Advance Reuqest).
- FedereA ve FedereB tarafından GALT değeri sorgusu gerçekleştirilerek, zaman adımı ilerlemesinin kontrolü sağlanır (Query GALT).
- FedereA ve FedereB tarafından mevcut mantıksal zaman değeri sorgusu gerçekleştirilerek, zaman adımı ilerlemesinin kontrolü sağlanır (Query Logical Time).
- RTI, 1. zaman adımı değeri hesaplanan GALT değerinden küçük olduğu için FedereB'nin zaman adımını 1. zaman adımına ilerletir (Time Advance Grant). GALT değeri hesaplanır.
- FedereB zaman adımını 8'e ilerletmek isteğini RTI'a bildirir (Time Advance Reuqest).
- FedereA ve FedereB tarafından GALT değeri sorgusu gerçekleştirilerek, zaman adımı ilerlemesinin kontrolü sağlanır (Query GALT).
- FedereA ve FedereB tarafından mevcut mantıksal zaman değeri sorgusu gerçekleştirilerek, zaman adımı ilerlemesinin kontrolü sağlanır (Query Logical Time).
- 8. zaman adımı değeri, federelerin ilerleme isteklerine göre yeni hesaplanmış olan GALT değerinden büyük olduğu için FedereB 8. zaman adımına da ilerletilemez. GALT değeri tekrar hesaplanır. Daha önce FedereA tarafından talep edilen zaman adımına ilerleme isteği (5) GALT değerinden küçük olduğu için federe istek

listesinden alınır ve FedereA'nın zaman adımı 5 olarak güncellenir (Time Advance Grand).

- FedereA zaman adımını 9'a ilerletmek istediğini RTI'ya bildirir (Time Advance Reuqest).
- FedereA ve FedereB tarafından GALT değeri sorgusu gerçekleştirilerek, zaman adımı ilerlemesinin kontrolü sağlanır (Query GALT).
- FedereA ve FedereB tarafından mevcut mantıksal zaman değeri sorgusu gerçekleştirilerek, zaman adımı ilerlemesinin kontrolü sağlanır (Query Logical Time).
- 9 değeri, ilerleme isteklerine göre yeni hesaplanmış olan GALT değerinden büyük olduğu için FedereA 9. zaman adımına ilerletilmez. GALT değeri tekrar hesaplanır. Önceki adımlarda bulunan FedereB tarafından RTI'ya bildirilen zaman adımına ilerleme isteği (8) GALT değerinden küçük olduğu için federe istek listesinden alınır ve FedereB'nin zaman adımı 8 olarak güncellenir (Time Advance Grand).
- Ardından GALT değeri güncellemelere bakılarak hesaplanır ve son olarak FedereA tarafından talep edilen zaman adımına ilerleme isteği (9) GALT değerinden küçük olduğu için federe istek listesinden alınarak FedereA'nın yeni zaman adımı 9 olarak güncellenir (Time Advance Grand).
- FedereA ve FedereB federasyondan ayrılır (Resign Federation Execution).
- Federasyon ismi vererek FedereA tarafından federasyon yok edilir (Destroy Federation Execution).



Şekil 17 – Zaman Yönetimi Testleri Örneği (Manage Time)

4. SONUÇ

SOM Sınama, Servis Kullanımı ve Uyumluluk Testi adımları işletilerek Federe Uyumluluğu sınanmıştır. Bu kapsamda gerçekleştirilen bu sınama işlemleri bir federenin HLA uyumluluğunun test edilmesi için yeterli adımlardır. Yine aynı bölümde verilen testler gözden geçirilerek bahsedilen ve gerçekleştirimi sağlanan testlerde olduğu gibi HLA federe kurallarına uyulduğu durumda federelerin ihtiyaç duyduğu ve dışarıya servis ettiği veriler bilinmektedir. Federelerin Uyumluluk Testi bölümünde bahsedildiği gibi sahiplik yönetimi ile ilgili bilgi almasını ya da Abone Ol/Yayınla mekanizmasında bahsedildiği gibi federelerin bilgi değişimini nasıl yaptıklarını anlamak mümkündür. Bahsedilen bölümde gerçekleştirilen sınama işlemleri gerçekleştirildikten sonra tanımlanan SOM ve Federe Uyumluluk verisi yeniden kullanılabilirlik açısından çok değerlidir. Burada oluşturulan SOM başka simülasyon sistemleri tarafından birlikte çalışabilirlik kapsamında şablon olarak kullanılabilecektir.

Bu tezde, federenin HLA uyumluluğunun denetlenmesi konusunda bir çalışma sunulmuştur. Federenin HLA uyumluluğunun sınanması ile ilgili adımlar ayrıntılı olarak açıklanmıştır. Federenin birlikte çalışabilirliğini sağlamak; teknik, sözdizimsel, anlamsal ve pragmatik düzeyde çaba ve standardizasyon gerektirmektedir. Modelleme ve Simülasyon sektöründe ülkemizde simülasyonların birlikte çalışabilirliği için gerekli olan HLA uyumluluğuna duyulan ihtiyaç, bu çalışmada geliştirilen araç setleri ile CEY Savunma şirketi bünyesinde geliştirilmekte olan RTI ürünü ile test edilmiştir. Bu testin gerçekleştirilmesi, birlikte çalışabilirlik sorunlarının bir kısmının giderilmesine yardımcı olmuştur. Bu sayede HLA uyumlu federeyi oluşturan simülasyon bileşeninin tekrar kullanılması durumunda oluşabilecek birlikte çalışabilirlik problemleri en aza indirilmiştir.

Ülkemizde savunma sektöründe pek çok firma tarafından HLA uyumlu olarak geliştirilen simülasyon sistemlerinin federe uyumluluğun sınanması için bilindiği kadarıyla herhangi bir test ve doğrulama adımı gerçekleştirilmemektedir. Yapılan bu çalışma ile HLA uyumlu olarak geliştirilecek simülasyonlarda oluşturulan bu araç setinin kullanılabileceği değerlendirilmektedir. Bu çalışmada sektörde en yaygın sürüm olması nedeniyle, HLA 1516 standardının 2000 yılında yayınlanan sürümüne odaklanmılmıştır. İlerki aşamalarda standardın Evolved (2010) sürümü ile uyumlu yapılacak çalışmalar genişleme potansiyeli olarak değerlendirilmektedir. Buna ilaveten federe uyumluluğunun sınanması için bir kural seti oluşturulup sektörle paylaşılması da hedeflenmektedir.

5. KAYNAKLAR

[1] A. Tolk, J. Muguira, 2003, IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries, New York, 1990.

[2] A. Tolk, J. Muguira, "The Levels of Conceptual Interoperability Model", in *Fall Simulation Interoperability Workshop*, Paper 03F-SIW-007, 2003.

[3] IEEE Standard 1730-2010 Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP), IEEE Std. 2000.

[4] W. Huiskamp et. al. Meeting the NATO M&S Interoperability Challenge, in M&S Journal, Summer 2014.

[5] F. Yılmaz, U. Durak., K. Taylan, H. Oğuztüzün., "Adapting functional mockup units for HLA-compliant distributed simulation", in *Proceedings of the 10th International Modelica Conference*, Lund, Sweden, Mar 10-12 2014, pp. 247-257.

[6] IEEE 1278.1–2012 Distributed Interactive Simulation (DIS) – Application Protocols, IEEE Standart, 2012.

[7] Aggregate Level Simulation Protocol (ALSP) – Management Plan, 26 May 93.

[8] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA): 1516–2000 (Framework and Rules); 1516.1-2000(Federate Interface Specification); 1516.2–2000 (Object Model Template (OMT) Specification) (2000), IEEE Std. 1516–2000.

[9] IEEE Standard for Modeling and Simulation HighLevel Architecture (HLA) – Framework and Rules, IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000), 2010, pp. 1-38.

[10] J.S. Dahmann., F. Kuhl., R. Weatherly, Standards for Simulation: As Simple As Possible But Not Simpler The High Level Architecture For Simulation, SIMULATION, sec 71. cilt, syf. 378-387,1998.

[11] A. Falcone, A. Garro, “Distributed Co-Simulation of Complex Engineered Systems by Combining the High Level Architecture and Functional Mock-up Interface”, in Simulation Modelling Practice and Theory, 2019, sec 97.

[12] H. Oğuztüzün, O. Topçu, “Towards A UML Extension For Federation Design”, *2nd Conference on Simulation Methods and Applications (CSMA-2000),* October, 2000.

[13] T. Gerlach, U. Durak, A. Knüppel, “Running High Level Architecture in Real Time for Flight Simulator Integration”, in *AIAA Modeling and Simulation Technologies Conference,* Washington, 2016.

[14] N. Sevim, Fotonik Ağlarla Bağlı Çok Çekirdekli İşlemciler İçin Yüksek Düzeyli Mimari (HLA) Standardında Veri İletişiminin Modellenmesi, syf. 13, 2014.

[15] S. Özkaymak, H.O. Zorba, A.B. Arıbal, M.B. Sayın, “Birlikte Çalışabilirlik Sorunları ve ARTI ile Çözüm Yaklaşımı”, in *Ulusal Savunma Uygulamaları Modelleme ve Simülasyon Konferansı (USMOS),* Ankara Turkey, Nov 19-20 2019.

[16] SISO-STD-001-2015 Standard for Guidance, Rationale, and Interoperability Modalities for the Real-time Platform Reference Federation Object Model, SISO, 2015.

[17] SISO-STD-001.1-2015 Real-time Platform Reference Federation Object Model, SISO, 2015.

EKLER

EK 1: Discover Object Instance Test

```
1 SCENARIO("Discover Object Instance Test - Happy path",
"[ObjectManagementServiceTests][DiscoverObjectinstance][DOI]")
2 {
3   Federate federateA;
4   Federate federateB;
5
6   GIVEN("Discover object instance with valid parameters")
7   {
8       std::wstring federateType = L"type1";
9       std::wstring federationExecutionName =
federateA.getUniqueName();
10      std::wstring fomModuleDesignator =  L"...";
11
12      rti1516::AttributeHandleSet attributeHandleSet;
13      rti1516::ObjectClassHandle baseEntityObjectClass;
14      rti1516::AttributeHandle worldLocationAttribute;
15      rti1516::ObjectInstanceHandle objectInstanceHandle;
16      rti1516::ObjectInstanceDiscovery objectInstanceDiscovery;
17
18      WHEN("Register object instance")
19      {
20          THEN("Object instance discovered")
21          {
22              try
23              {
24                federateA.getRtiAmbassador()->
createFederationExecution(federationExecutionName,
fomModuleDesignator);
25                federateA.getRtiAmbassador()->
joinFederationExecution(federateType, federationExecutionName,
federateA);
26                federateB.getRtiAmbassador()->
joinFederationExecution(federateType, federationExecutionName,
federateB);
27
28                baseEntityObjectClass =
federateA.getRtiAmbassador()-> getObjectClassHandle(L"BaseEntity");
29                worldLocationAttribute =
federateA.getRtiAmbassador()->
getAttributeHandle(baseEntityObjectClass, L"WorldLocation");
30
31
attributeHandleSet.insert(worldLocationAttribute);
32                federateA.getRtiAmbassador()->
publishObjectClassAttributes(baseEntityObjectClass,
attributeHandleSet);
33                federateB.getRtiAmbassador()->
subscribeObjectClassAttributes(baseEntityObjectClass,
attributeHandleSet);
34
35                objectInstanceHandle =
federateA.getRtiAmbassador()->
registerObjectInstance(baseEntityObjectClass, objectInstanceName);
```

```
36                objectInstanceDiscovery =
federateB.observeDiscoverObjectInstance(baseEntityObjectClass);
37
38                REQUIRE(baseEntityObjectClass ==
objectInstanceDiscovery.theObjectClass);
39                REQUIRE(objectInstanceHandle ==
objectInstanceDiscovery.theObject);
40
41              federateB.getRtiAmbassador()->
resignFederationExecution(DELETE_OBJECTS_THEN_DIVEST);
42              federateA.getRtiAmbassador()->
resignFederationExecution(DELETE_OBJECTS_THEN_DIVEST);
43              federateB.getRtiAmbassador()->
destroyFederationExecution(federationExecutionName);
44             }
45            catch (rti1516::Exception& exception)
46            {
47               std::wcout << exception.what() << std::endl;
48               REQUIRE(false);
49            }
50        }
51    }
52 }
53}
```