Açık Kaynak Kodlu Portico RTİ için Yeni İletişim Modelleri

Serkan ÖZEN* **, Selim TEMİZER**
Orta Doğu Teknik Üniversitesi, Bilgisayar Mühendisliği Bölümü, Çankaya, 06800, Ankara, Türkiye
(Geliş / Received : 26.08.2014 ; Kabul / Accepted : 09.02.2015)

ÖZ

Koşum-zaman altyapılarında (Run-time infrastructure, RTI) federe ile federasyon arasındaki bağlantı modeli, altyapının dizaynında dikkat edilmesi gereken önemli bir unsurdur. Mevcut kapalı ve açık kaynak kodlu sistemlerde bağlantı modelleri olarak paylaşımlı hafıza, UDP ve TCP/IP gibi yöntem ve protokoller kullanılmaktadır. Portico RTI, anlaşılır bir şekilde yazılmış kodu, esnek mimarisi ve geliştiriciler tarafından sağlanan destek bakımından açık kaynak kodlu RTI yazılımları arasında önemli bir yere sahiptir. Portico RTI'nın halihazırda iletişim modeli olarak sunduğu 2 farklı iletişim modeli bulunmaktadır. Bunlardan ilki Jgroups tabanlı iletişim modelidir. Jgroups, IP tabanlı güvenilir birden çoğa yayın (reliable IP multicast) tabanlı bir kütüphanedir. Bu iletişim modeli ile birlikte, ağ üzerinde dağıtık olarak çalışan simülasyon programları birbirleri ile haberleşebilmekte ve HLA (High Level Architecture) standardının kuralları çerçevesinde birlikte çalışabilmektedirler. Jgroups ağ tabanlı bir kütüphane olduğundan zaman açısından birtakım gecikmeler yaşatabilmektedir. Simülasyon programlarının farklı bilgisayarlar üzerinde değil de aynı bilgisayarlar üzerinde çalıştığı durumlarda Jgroups tabanlı iletişim modeli yerine kullanılabilecek farklı iletişim modeli arayışları gelişmiştir. Bunun sonucu olarak JVM (Java Virtual Machine) tabanlı iletişim modeli geliştirilmiştir. JVM modelinde aynı bilgisayar üzerinde koşan simülasyonlar farklı işlem parçaları (thread) olarak çalışmaktadır. Bu iletişim modelinin de birtakım problemleri bulunmaktadır. JVM tabanlı iletişim modelinde herhangi bir işlem parçasının hata vermesi sonucu tüm JVM işlemi (process) sonlanabilir ve böylece birlikte çalışabilirlik ilkesine aykırı bir durum oluşmuş olur. Bir diğer problem ise, simülasyonların farklı birer işlem parçası olarak çalışmasını sağlayacak ara bir program hazırlanmak zorundadır. Bu durum simülasyonların doğrudan birlikte çalışmasını engellemektedir. Tüm bu durumlar göz önüne alındığında, kapalı kaynak kodlu sistemlerde de sıkça kullanılan paylaşımlı hafıza (shared memory) modelinin Portico RTI için geliştirilmesi ihtiyacı ortaya çıkmıştır. Paylaşımlı hafıza modelinde, simülasyonlar farklı birer işlem olarak çalışacak olup, hafıza alanı üzerinden haberleşeceklerdir. Bu da JVM modelinde bahsedilen dezavantajları giderecek ve simülasyonlara Jgroups modeline göre daha hızlı bir iletişim altyapısı sunacaktır. Java tabanlı bir yazılım olan Portico'da hafıza operasyonları JNI (Java Native Interface) teknolojisi kullanılarak gerçekleştirilecektir. JNI Java programlarına doğrudan hafıza operasyonları yapma olanağı tanıyan bir kütüphanedir. Bu çalışmada, her bir bilgisayarda birden fazla olmak üzere ağ üzerinde dağıtık olarak çalışan simülasyonların haberleşmelerini sağlayacak hibrit iletişim modeli de anlatılacaktır. Bu modele göre aynı bilgisayar üzerinde çalışan simülasyonlar paylaşımlı hafıza modeli ile, farklı bilgisayardakiler ise Jgroups modeli altyapısı ile haberleşeceklerdir. Bu iki yeni model var olan Jgroups altyapısına göre daha hızlı bir haberleşme imkanı sunmaktadır. Daha önce Portico için paylaşımlı hafıza ve hibrit iletişim modelleri geliştirilmediği için bu çalışma iki yeni iletişim modeli sunarak Portico ile haberleşen simülasyonlar için daha hızlı bir iletişim altyapısı sağlamaktadır.

Anahtar Kelimeler: Paylaşımlı-hafıza iletişim modeli, hibrit iletişim modeli, dağıtık simülasyon, koşum-zaman altyapısı, yüksek seviye mimari

New Communication Models for Open Source Portico RTİ

ABSTRACT

The communication model between federates and the federation is an important point that should be considered carefully when designing run-time infrastructure (RTI) software. Some of the communication methods and protocols used in existing closed-source systems include shared memory, UDP and TCP/IP. Portico has a very special place among other open source RTIs since it has a well-written code, flexible architecture and it is actively supported by its developers. Portico RTI currently has 2 different communication models. First of these models is Jgroups based communication model. Jgroups is an IP multicast based library. Federates, distributed over network, can communicate with each other and run together by following the HLA (High Level Architecture) standard rules. Since Jgroups is a network based library, it may cause some latency. The need for new communication models has risen for the scenarios where simulations run on the same computer. As a result of this need, JVM (Java Virtual Machine) communication model has been developed for Portico RTI. Simulations, running on the same computer, are invoked as separate threads in JVM model which has some deficiencies. Failing one of simulation threads may cause JVM process to fail which in turn makes other simulations fail which breaks the interoperability rule. Another problem of JVM model is that a wrapper class is needed to invoke simulations as separate threads. This prevents simulations from running directly without any need for a wrapper class. A need for new communication models have raised because of the deficiencies current communication models have. In shared memory model, simulations run as separate processes and communicate through a shared memory region. Shared memory model overcomes the problems JVM model has and provides a faster communication model to simulations. JNI (Java Native Interface) technology will be used in order to realize memory operations in Portico which is a Java based software. Not only shared memory model but also a hybrid communication model will be presented in this paper. In hybrid communication model, simulations running on the same computer will communicate through shared memory model and simulation running on different computers will communicate through Jgroups communication model. Both shared memory and hybrid communication model perform better than Jgroups communication model. Since both hybrid and shared memory communications model have not been developed for Portico until now, this study proposes new communication models for Portico and improves the total communication times of simulations communicating through Portico.

Keywords: Shared-memory communication model, hybrid communication model, distributed simulation, run-time infrastructure, high level architecture

* *Sorumlu Yazar (Corresponding Author)*
e-posta: serkan.ozen@metu.edu.tr
Digital Object Identifier (DOI) : 10.2339/2015.18.3 165-174

1. GİRİŞ (INTRODUCTION)

1.1. Yüksek Seviye Mimari (High Level Architecture)

Yüksek Seviye Mimari (High Level Architecture, HLA) standardı Amerika Birleşik Devletleri Savunma Bakanlığı (USA Department of Defence, DoD) tarafından, modelleme ve simülasyon sistemlerinin rahatlıkla haberleşebilmesi ve bir arada çalışabilmesine imkan verecek ortak bir altyapı oluşturmak için hazırlanmıştır [1]. Yüksek seviye mimari standardı üç ana başlık altında özetlenebilir:

1. Çerçeve ve Kurallar (Framework and Rules): Federelerin (simulasyonların veya gerçek sistem arayüzlerinin) federasyon (birlikte çalışan federeler grubu) ile güvenilir şekilde haberleşmesini sağlayan kuralları ortaya koyar [1].
2. Yüksek Seviye Mimari Nesne Model Şablonu (HLA Object Model Template, OMT): Bu doküman federasyon içinde üretilen ve kullanılan nesnelerin (mesajlar, gerçek dünya varlıklarını temsil eden yazılım nesneleri, vs.) özelliklerini tanımlar [2].
3. Servisler (Services): Federelerin birbirileriyle iletişim kurabilmeleri için gereken servisleri tanımlar [3]. Bu servisler aşağıdaki kategorilere ayrılırlar:

a) Federasyon Yönetimi (Federation Management): Bu servisler federasyonun yaratılması, çalıştırılması ve sonlandırılması ile ilgili servislerdir.

b) Beyan Yönetimi (Declaration Management): Federelerin, federasyon içinde hangi verileri tüketip hangilerini üreteceklerini beyan ettikleri servislerdir.

c) Nesne Yönetimi (Object Management): Federasyon içindeki nesnelerin yaratılması, kayıt altına alınması, modifiye edilmesi ve silinmesini sağlayan servislerdir.

d) Sahiplik Yönetimi (Ownership Management): Federelerin, federasyon içindeki nesnelerin (veya nesnelere ait alt nesne parçalarının) sahipliklerini devralmalarını veya devretmelerini sağlayan servislerdir.

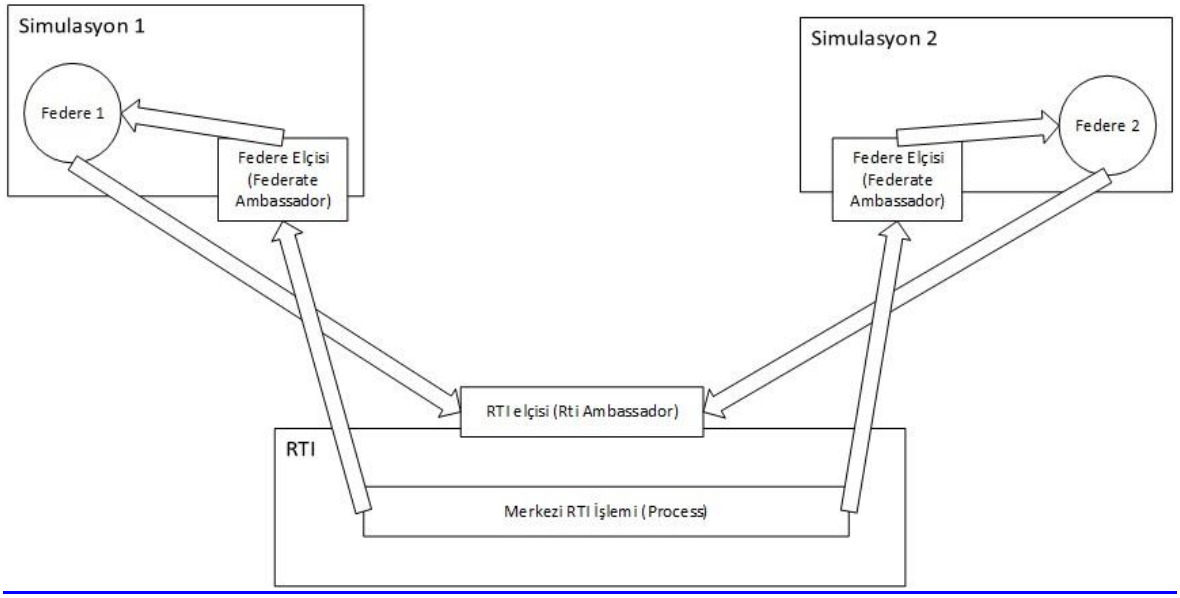
e) Zaman Yönetimi (Time Management): Federasyon içinde dolaşan verinin zaman yönetimini sağlayan servislerdir.

f) Veri Dağıtım Yönetimi (Data Distribution Management): Federasyon içinde dolaşan verinin dağıtımını yöneten ve gereksiz veri transferlerini engelleyen servislerdir.

g) Destek Servisleri (Support Services): Federelerin federasyon koşumu sırasında ihtiyaç duyduğu destek servisleridir.

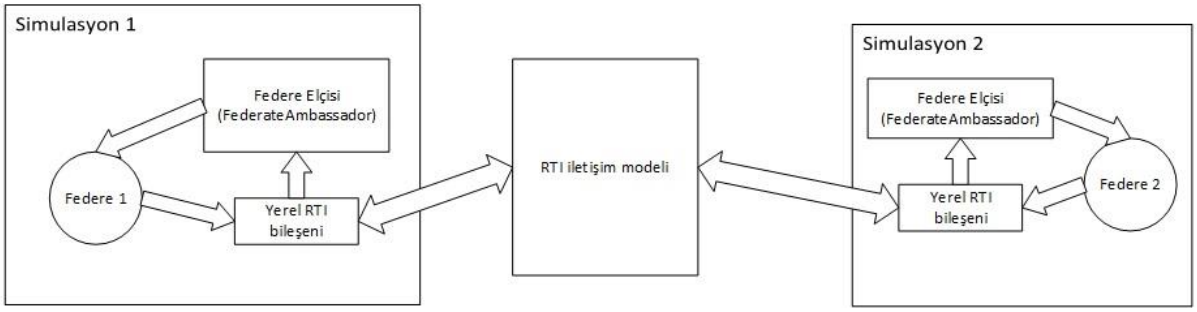
1.2. Koşum-Zaman Altyapısı (Run-time Infrastructure)

Koşum-zaman altyapı yazılımı (Run-time infrastructure, RTI), HLA standart kurallarını uygulayarak yazılan bir arakatman yazılımıdır [4]. RTI, simülasyon yazılımlarına güvenilir bir şekilde birbirileriyle haberleşebilecekleri bir arakatman altyapısı sunmaktadır. RTI yazılımları yapılarına göre 2 gruba ayrılabilir: merkezi ve dağıtık RTI [5]. Merkezi arakatman yazılımının yapısı Şekil 1.1'de gösterilmiştir. Bu yapıda merkezi bir RTI işlemi (process), federelere RTI servislerini sunar. Federeler bu RTI servislerine RtiAmbassador elçi yazılım birimi üzerinden ulaşırlar. RTI'nın federelere mesaj göndermesini sağlayan yapı ise FederateAmbassador elçi yazılım birimidir. RTI bu modüldeki geri arama (callback) servislerini kullanarak diğer federelerden gelen mesajları ilgili federelere iletir. Şekil 1.2'de dağıtık RTI



Şekil 1.1 Merkezi RTI (Central RTI)

yapısı sunulmuştur. Şekilde görüldüğü üzere, dağıtık RTI yapılarında her bir federe, yerel bir RTI kütüphanesi ile çalışır. Burada RTI merkezi bir işlem değil, her bir federeyle birlikte çalışmak üzere dağıtılmış RTI kütüphanelerinden oluşmaktadır. Dağıtık RTI'da yerel RTI bileşeni, federeden aldığı mesajları RTI iletişim modeli üzerinden (ağ üzerinden, ortak hafıza alanı kullanarak, vs.) diğer federelerin yerel RTI bileşenlerine iletir. Yerel RTI bileşenleri gelen mesajları FederateAmbassador modülünün sağladığı servisleri kullanarak federelere iletir. Her iki modelde de federe sadece RTI servislerini kullanarak diğer federelere mesaj gönderir ve Federe elçisi üzerinden mesaj kabul eder. Hiçbir şekilde diğer federelerle doğrudan iletişim söz konusu değildir.



Şekil 1.2 Dağıtık RTI (Distributed RTI)

RTI yazılımlarının açık ve kapalı kaynak kodlu olmak üzere birçok versiyonu bulunmaktadır. Popüler kapalı kaynak kodlu RTI yazılımlarının bazıları MÄK RTI [1], Pitch RTI [2] ve RTI NG Pro [3] 'dur. Bunlardan MÄK RTI, TCP, UDP ve paylaşımlı-hafıza (shared-memory) modellerini federasyon iletişim modeli olarak sunmaktadır [6]. Bir başka değişle federe ve RTI arasındaki iletişim opsiyonel olarak bu 3 iletişim modelinden biri ile gerçekleştirilebilmektedir. Pitch RTI da bu 3 iletişim model seçeneğini sunmaktadır [7]. RTI NG ise TCP ve UDP modellerini iletişim modeli olarak sunmaktadır. Görülebileceği gibi paylaşımlı-hafıza modeli popüler RTI yazılımları tarafından kullanılmakta olan bir alternatiftir. Bu çalışmada, açık kaynak kodlu bir RTI yazılımı olan Portico [4] için geliştirilen paylaşımlı-hafıza ve hibrit iletişim modellerininin yapısı ve halihazırdaki Portico iletişim altyapısı ile karşılaştırma sonuçları sunulacaktır.

2. RTI İLE İLGİLİ OLARAK GERÇEKLEŞTİRİLMİŞ OLAN ÇALIŞMALAR (PREVIOUS STUDIES ON RTI)

2.1.RTI Performans Çalışmaları (RTI Performance Studies)

RTI'ları performans açısından karşılaştırmak çoğu araştırmacının ilgisini çekmektedir. Bu karşılaştırmalar sonucu RTI'lar performanslarına göre değerlendirilebilmektedir. Aşağıdaki karşılaştırma sonuçlarında Portico RTI'nın yanısıra yer alan bir diğer popüler açık kaynak kodlu RTI ise Certi'dir [5].

Malinga ve Leroux, araştırmaları sırasında inceledikleri RTI'ların güvenilir (reliable) modda çalıştıklarında benzer performans sergilediklerini ancak, en-iyi-çaba (best-effort) modunda çalıştırıldıklarında ise MÄK RTI'nın daha iyi sonuç sergilediklerini göstermişlerdir [8]. Şekil 2.1'de görüldüğü üzere toplam mesajlaşma süresi, veri boyutu yükseldikçe artmaktadır. 1024 KB paket boyutuna kadar MÄK RTI'nın diğer RTI'lardan daha iyi çalıştığı gözlenmektedir. 1024 KB paket boyutunda ise, Portico'nun Certi'den daha iyi çalıştığı gözlenmektedir. En-iyi-çaba modundaki sonuçlar ise Şekil 2.2'de yer almaktadır. MÄK RTI'nın en-iyi-çaba modunda 64 KB paket boyutu sınırı vardır. Bu sonuçlara göre, 64 KB'a kadar MÄK RTI diğerlerinden daha iyi performans sergilemektedir. 1024 KB'a kadar Certi, Portico'dan daha iyi performans sergilemekte ancak, 1024 KB paket boyutunda Portico ve Certi'nin aşağı yukarı aynı performansı sergilediği görülmektedir.

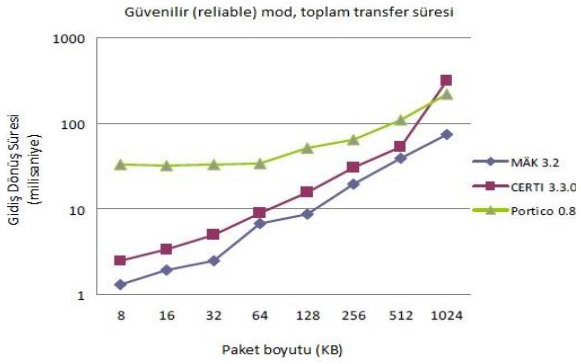
Performans konusunda yapılan bir başka çalışmada RTI NG, Pitch RTI, MÄK RTI ve DRTI (RTI'nın alt kümesini uygulamasından ötürü deneysel bir RTI'dır) karşılaştırılmıştır [9]. Şekil 2.3'te görüldüğü gibi, 64 KB paket boyutuna kadar MÄK RTI en iyi performansı göstermiş ve RTI NG ise 2. sırada kalmıştır. DRTI 16 KB paket boyutunun dışında kalan tüm paket boyutlarında Pitch RTI'dan daha iyi bir performans sergilemiştir. 64 KB ve daha büyük paket boyutlarında ise performans sonuçları en iyiden en kötüye doğru sırasıyla MÄK RTI, RTI NG, DRTI and Pitch RTI şeklindedir. Şekil 2.4'te görüldüğü gibi, paket boyutu 1024 KB'a yaklaştıkça DRTI'in geometrik oranda yavaşladığı gözlemlenmiştir.

2.2.RTI Mimari Çalışmaları (RTI Architecture Studies)

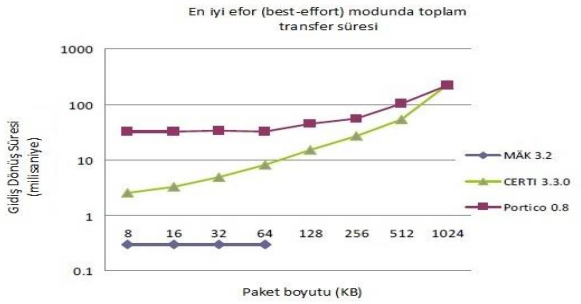
RTI mimarisi ile ilgili birçok çalışma bulunmaktadır. Bu çalışmalar genel olarak RTI'yı web ile bütünleştirmek ve RTI servislerini web servisi olarak sunmak üzerine yoğunlaşmıştır. Morse çalışmasında, federelerin geniş alan ağları üzerinden simülasyona katılmalarını sağlayacak yeni bir RTI mimarisi geliştirmiştir [10]. Drago-

[1] http://www.mak.com/products/link/mak-rti.html (26 Temmuz 2014)
[2] http://www.pitch.se (26 Temmuz 2014)
[3] http://www.raytheon.com/capabilities/products/rti (26 Temmuz 2014)
[4] http://www.porticoproject.org (16 Ağustos 2014)
[5] http://savannah.nongnu.org/projects/certi (24 Temmuz 2014)

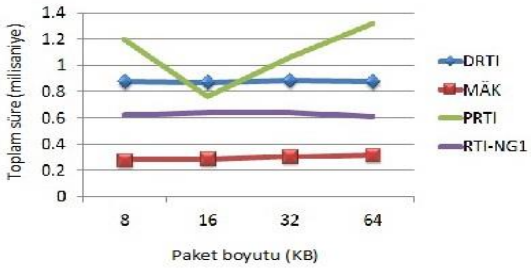
ciea yaptığı çalışmada RTI servislerinin servis tabanlı mimari üzerinden kullanılabileceği yeni bir mimari geliştirmiştir [11]. Benzer şekilde Zhang, HLA ve servis tabanlı mimariyi entegre edecek programlama dili ve platform bağımsız yeni bir çerçeve (framework) geliştirmiştir [12]. Zhiying web servislerini Portico RTI'ya entegre etmiştir. Yerel ağ bağlantısı içindeki federeler Portico iletişim altyapısıyla, yerel ağ bağlantısı dışındaki federeler ise Portico ile web servisleri aracılığıyla haberleşmektedir [13]. Kapolka ve Andrzej RTI'nın eksiklerine değinerek, yeni ve geliştirilebilir bir RTI'nın (Extensible RTI, XRTI) gerekliliğine vurgu yapmışlardır. XRTI'nın sahip olması gereken özelliklerinin; kolay kullanılması, standart haline dönüşebilecek bir iletişim altyapısına sahip olması, nesne modellerinin (object models) dinamik olarak yaratılıp geliştirilebilir olması ve XRTI'nın açık kaynak kodlu olması gerektiği olduğunu belirtmişlerdir [14]. PLA Bilim ve Teknoloji Üniversitesi'nde yapılan bir çalışmada RTI servisleri, model tabanlı mimari (model driven architecture) uygulanarak geliştirilmektedir. Bu da RTI servislerinin tekrar kullanılabilirliğini arttırmaktadır [15]. Ting-xue çalışmasında HLA tabanlı yeni bir veri iletişim modeli geliştirmiştir [16]. Zhou ise dağıtık nesne sahiplik algoritmaları geliştirmiş ve bu dağıtık algoritmaların merkezcil çalışan algoritmalara göre daha iyi performans sağladıklarını göstermiştir [17]. Quan ve Zhong DVE RTI adında yeni bir dağıtık RTI geliştirmişlerdir. Bu yeni RTI'nın özelliği ise RTI'yı oluşturan modüllerin birbirlerine bağımlılıklarının düşük olmasıdır (loosely coupled). Bu özellik RTI'ya yeni servislerin daha kolay entegre edilmesini, var olanların daha kolay değiştirilebilmesini sağlamaktadır [18]. Web ve HLA entegrasyonunun dezavantajlarını gösteren çalışmalar da bulunmaktadır. Byrne yaptığı çalışmada, servis tabanlı mimarinin ve RTI'nın entegre edilmesinin simulasyon hızlarında kayıplara neden olacağına, kullanıcı arayüzünde kısıtlamalara gidilmesi gerekebileceğine ve simülasyonların kararlılığının sarsılabileceğine değinmiştir [19]. Whang ve Zhang çalışmalarında servis tabanlı mimari ve RTI entegrasyonunun dezavantajlarını gidermek için yeni bir servis-dağıtık RTI (service-distributed RTI) SDRTI önermişlerdir [20]



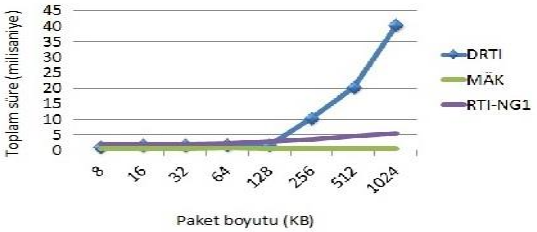
Şekil 2.1 Güvenilir modda Portico, Certi ve MÄK RTI'ların karşılaştırılması (Portico, Certi, MAK RTI comparison in reliable mode)



Şekil 2.2 En-iyi-çaba modunda Portico, Certi ve MÄK RTI'ların karşılaştırılması (Portico, Certi and MAK RTI comparison in vest effort mode)



Şekil 2.3 DRTI, MÄK RTI, Pitch RTI ve RTI NG'nin karşılaştırılması (Comparison of DRTI, MÄK RTI, Pitch RTI and RTI NG



Şekil 2.4 1024 KB paket boyutunda DRTI, MÄK RTI ve RTI NG'nin karşılaştırılması (Comparison of DRTI, MÄK RTI and RTI NG for 1024 KB packet size

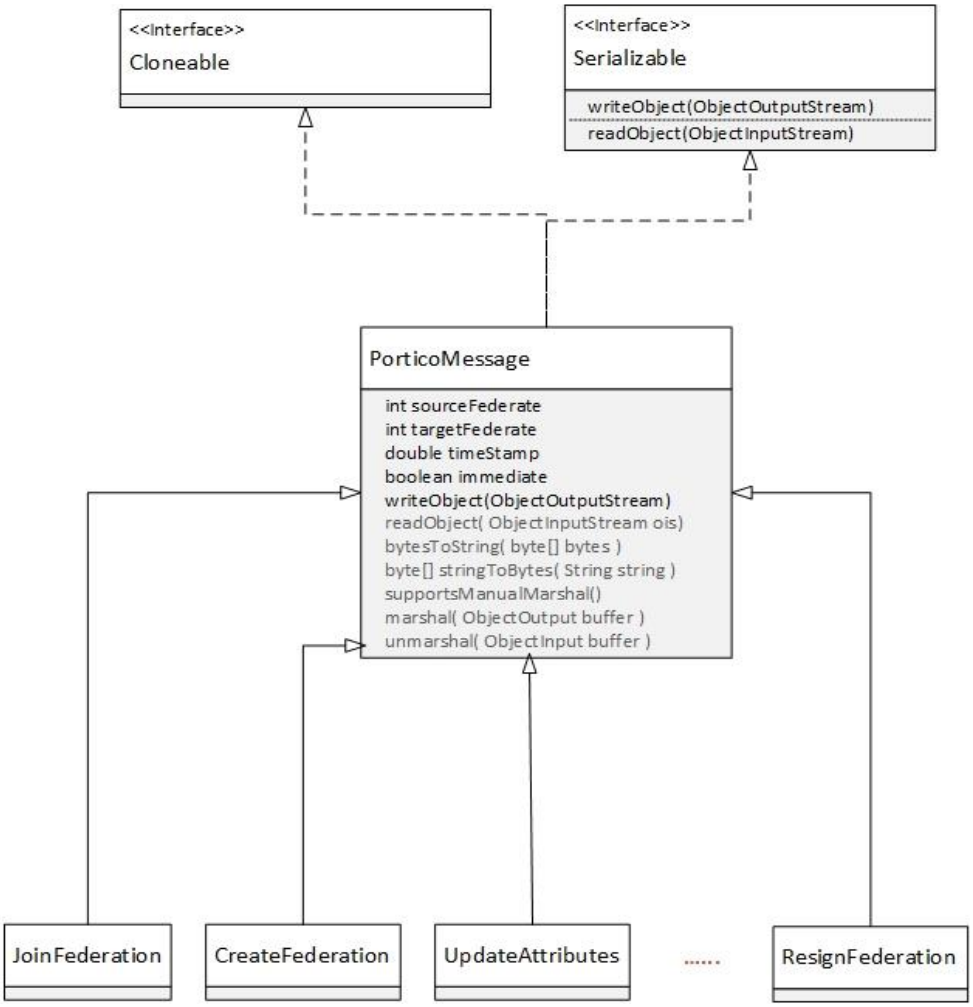
3. PORTICO RTI

Portico RTI, Avustralya merkezli Calytrix Technologies firması tarafından, Avustralya Savunma Simülasyon Ofisi'nin desteğiyle aktif olarak geliştirilen açık kaynak kodlu bir RTI yazılımıdır [21].

3.1.Portico RTI Mesajlaşma Altyapısı (Portico RTI Messaging Infrastructure)

Portico, dağıtık bir RTI olması sebebiyle, her bir yerel Portico RTI bileşeninin diğer yerel Portico RTI bileşenleriyle haberleşmesi gerekmektedir. Bu haberleşme, byte'lardan oluşan bir seri haline getirilebilen (serializable) mesaj sınıfları sayesinde gerçekleştirilebilmektedir. Şekil 3.1'de görüldüğü gibi federeler arası iletişimde ihtiyaç duyulacak her veri parçası bir Java sınıfıyla temsil edilmektedir. Federeler bu mesaj sınıflarını byte serileri haline getirip Portico iletişim altyapısı üzerinden diğer federelere göndermektedirler. Şekil 3.1'de görül-

düğü gibi federeler arası mesajlaşmada kullanılacak tüm mesaj sınıfları



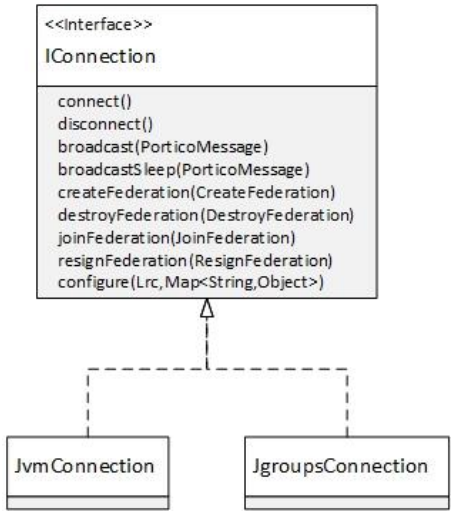
Şekil 3.1 Portico mesajları sınıf diyagramı (Portico messages class hierarchy)

delleri oluşturulmak istenirse, yine Portico'nun sağladığı bu arayüz ile uyumlu olacak şekilde tasarlanmalıdırlar.

PorticoMessage sınıfından türetilmektedir. PorticoMessage sınıfı ise hem byte serisi haline getirilebilen hem de klonlanabilen (clonable) özelliklere sahiptir. Bu sayede federeler her türlü mesajlaşma sınıfını birbirilerine gönderebilmektedirler.

3.2.Portico İletişim Modelleri (Portico Communication Models)

Portico iletişim modeli, Portico mesaj sınıflarının, alıcılarına hangi teknoloji ve protokol kullanılarak iletileceğini belirler. Hali hazırda Portico'da iki adet iletişim modeli bulunmaktadır. Bunlardan birincisi Java Sanal Makinesi (Java Virtual Machine, JVM) modeli, diğeri ise Jgroups modelidir. Şekil 3.2'de görüldüğü gibi bu modeller Portico'da bulunan IConnection isimli arayüze (interface) uygun olarak Portico'nun iletişim modellerini oluşturmaktadırlar. Bunların dışında yeni iletişim mo-

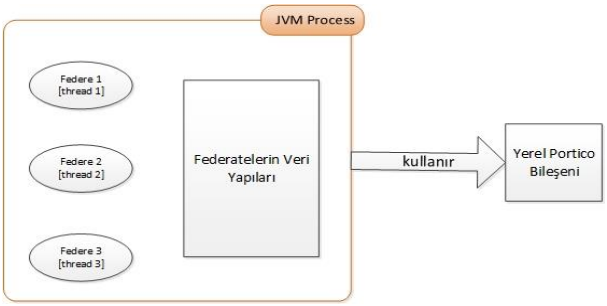


Şekil 3.2 IConnection arayüzü ve arayüzü uygulayan sınıflar (Iconnection interface and classes implement it)

3.2.1 JVM İletişim Modeli (JVM Connection Model)

Bu modelde federelerin her biri ayrı birer işlem parçası (thread) olarak tek bir JVM işlemi (process) bünyesinde koşarlar. Mesajları doğrudan birbirlerinin veri yapılarının içine yazarlar.

Şekil 3.3'te görüldüğü üzere federeler aynı işlem üzerinde koştukları için birbirlerinin veri yapılarına ulaşabilmektedirler.

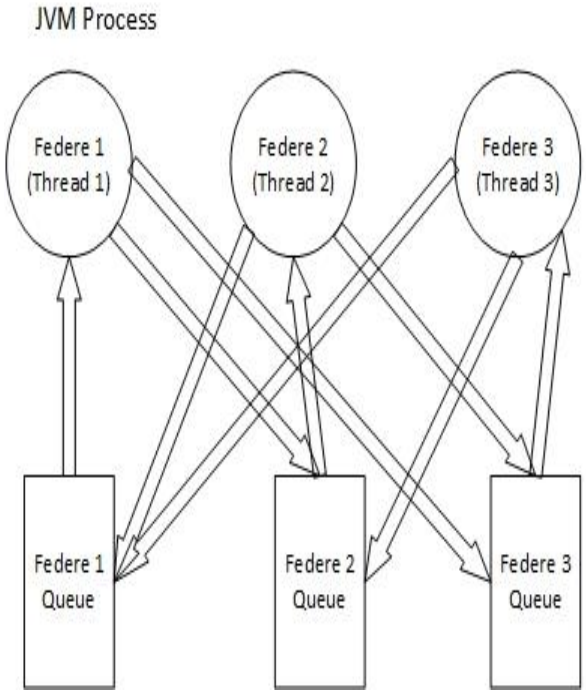


Şekil 3.3 JVM işlemi ve federe işlem parçaları (JVM process and federate threads)

Şekil 3.4'te federelerin birbirileriyle nasıl iletişim kurdukları görülmektedir. Her federe diğer federelerin mesaj dizilerine ulaşabilmektedir ve böylece iletmek istediği mesajı doğrudan yazabilmektedir. Federeler, kendilerine gelen mesajları ise kendi mesaj dizilerinden okuyabilmektedirler.

3.2.2.Jgroups İletişim Modeli (Jgroups Communication Model)

Jgroups [6], IP tabanlı bir çoğa-gönderim (IP multicast) kütüphanesidir. Jgroups modelinde her bir federasyon bir Jgroups kanalı (channel) tarafından temsil edilmektedir. Federasyona katılmak isteyen bir federe, bu Jgroups kanalına üye olmaktadır. Dolayısıyla bir kanala üye olan tüm federeler bu kanal vasıtasıyla birbirilerine mesaj gönderebilmektedirler. Şekil 3.5'te görüldüğü gibi bir kanala gönderilen mesaj, o kanala üye olan tüm federeler tarafından kabul edimektedir.

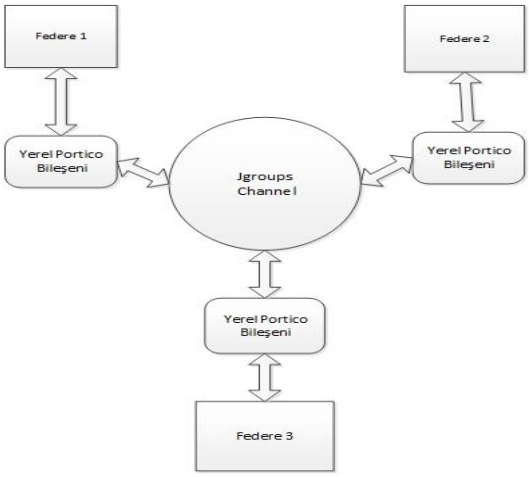


Şekil 3.4 Federe işlem parçalarının mesajlaşması (Communication of federate threads)

Tablo 3.1'de Jgroups'un diğer yaygın protokoller arasındaki yeri gösterilmiştir.

4. MEVCUT PROBLEMLER VE ÖNERİLEN ÇÖZÜMLER (PROBLEMS AND PROPOSED SOLUTIONS)

Portico RTI'nın halihazırdaki iletişim modellerinin bazı devantajları bulunmaktadır. JVM iletişim modeli federelere paylaşımlı-hafıza benzeri bir yapı sunmaktadır. Ancak görülebileceği üzere bu model tam olarak paylaşımlı-hafıza modelinin avantajlarını sağlayamamaktadır. Gerçek bir dağıtık sistemde, federelerin birbirilerinden bağımsız işlemler olmaları



Şekil 3.5 Jgroups iletişim modeli (Jgroups Communication Model)

[6] http://www.jgroups.org/ (18 Ağustos 2014)

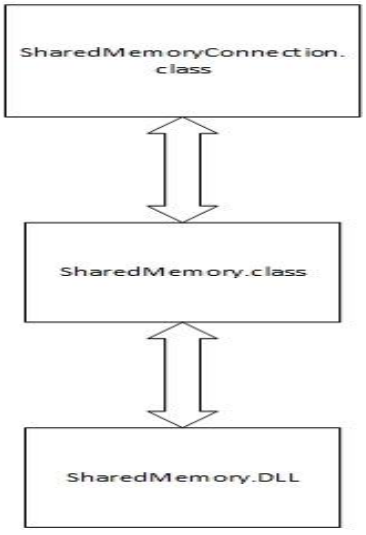
Tablo 3.1 Jgroups'un yaygın protokoller arasındaki yeri (Place of Jgroups among other protocols)

	Garantisiz (Unreliable)	**Güvenilir (Reliable)**
Teke-gönderim (Unicast)	UDP	TCP
Çoğa-gönderim (Multicast)	IP multicast	JGroups

beklenir. Ancak JVM modelinde her bir federe ayrı bir işlem olarak değil, ayrı bir işlem parçası olarak çalışmaktadır. Bu durum federelerin dağıtık olarak çalışmasını engellemektedir. Jgroups modelinde ise federeler Jgroups kütüphanesini kullanarak ağ üzerinden birbirileriyle iletişim kurabilmektedirler. Ancak federeler aynı bilgisayar üzerinde koşsalar bile Jgroups kütüphanesinin metodlarını (ağ tabanlı metodlar) kullanarak haberleşmek zorundadırlar. Bu durum ise gereksiz performans kayıplarına neden olmaktadır. Bu çalışma kapsamında, haberleşme için sarfedilen zaman performansının artırılması için, tüm federelerin aynı bilgisayar üzerinde koşması durumlarında *paylaşımlı-hafıza iletişim modelinin* kullanılması, birden fazla bilgisayardan oluşan bir senaryoda ve her bilgisayarda birden fazla federenin koştuğu durumlarda ise *hibrit iletişim modelinin* kullanılması önerilmektedir.

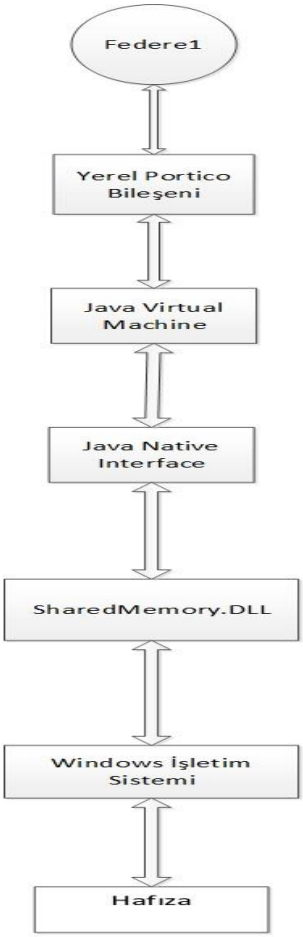
4.1.Paylaşımlı-Hafıza İletişim Modeli (Shared-Memory Connection Model)

Tüm federelerin aynı bilgisayar üzerinde çalıştığı durumlarda, federelerin Jgroups'un ağ tabanlı servislerini kullanmaları sonucu performans kaybına uğramaları



Şekil 4.1 Paylaşımlı hafıza modeli yapısı (Shared-memory model structure)

gerkesiz bir durumdur. Aynı bilgisayar üzerinde koşan federeler ortak bir hafıza alanı üzerinden iletişim kurabilirler. Şekil 4.1'de görüldüğü üzere, paylaşımlı hafıza iletişim modeli (SharedMemoryConnection), C++ hafıza metodlarının Java'daki karşılığının bulunduğu SharedMemory sınıfını kullanarak, SharedMemory.dll kütüphanesindeki hafıza metotlarına erişim sağlamaktadır. Şekil 4.2'de federelerin hafıza operasyonlarına erişebilmeleri için kullandıkları platformlar arası hiyerarşi gösterilmektedir. Buna göre, her federe sadece yerel Portico RTI bileşeninin sağladığı servisleri kullanabilmektedir. Hiçbir şekilde doğrudan hafıza operasyonları kütüphanesine (SharedMemory.dll) erişmesi söz konusu değildir. Yerel Portico bileşeni ise sahip olduğu JNI metodlarını kullanmaktadır.



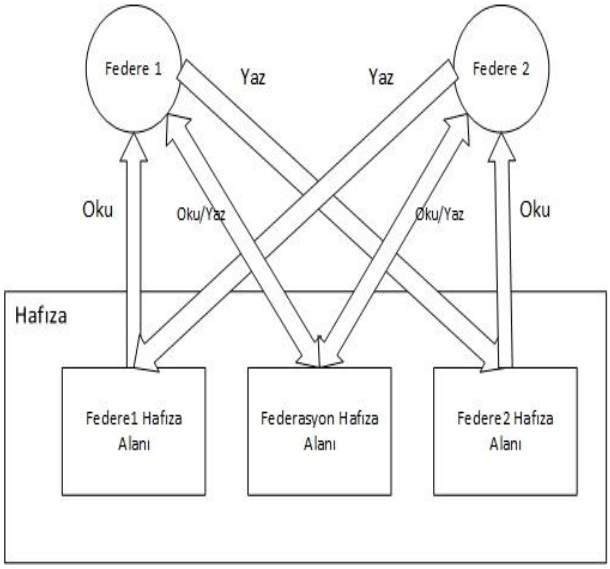
Şekil 4.2 Paylaşımlı Hafıza Modeli platformlar hiyerarşisi (Shared-memory model platforms hierarchy)

Bu metodlar JVM ve JNI üzerinden SharedMemory.dll içerisinde yer alan C++ hafıza metodlarına erişim sağlarlar. Bu metodlar ise Windows işletim sistemi aracılığıyla hafıza üzerinde gerekli yazma ve okuma işlemlerini yerine getirirler.

4.2.Paylaşımlı-Hafıza Modelinde Federe ve Federasyon Hafıza Alanı (Federate and Federation Memory Regions in Shared-memory Model)

Paylaşımlı hafıza modelinde her bir federasyon bir hafıza alanı ile temsil edilir. Bu federasyon alanında belirli veri yapıları ve federasyona katılan federelerin hafıza alanlarının isimleri bulunur.

Böylece diğer federelere mesaj yollamak isteyen bir federe, federasyon hafıza alanından federe isimlerini okuyarak o isimlere karşılık gelen hafıza alanlarına mesajlarını yazar. Şekil 4.3'te görüldüğü gibi, federe 1 ve federe 2 kendi hafıza alanlarına gelen mesajları okurlar. Diğer federelere mesaj göndermek istediklerinde, diğer federelerin hafıza alanlarının isimlerini okurlar ve o hafıza alanlarını açarak mesajı o alanlara yazarlar.



Şekil 4.3. Federe ve hafıza etkileşimi (Federate and memory interaction)

Şekil 4.4'te federe hafıza alanı daha ayrıntılı olarak görülebilmektedir. Bir federenin hafıza alanında yer alan mesaj dizini (message array), toplam hafıza alanı sayısı kadar bölmelere ayrılır. Her federe kendi ismine karşılık gelen bölmeye mesaj yazabilmektedir. Böylece her bölmede bir yazan ve bir okuyan olmak üzere yalnızca 2 işlem çalışmakta ve işlemler arası etkileşimden doğabilecek olası komplikasyonlar giderilmektedir. Peterson algoritması [7] kullanılarak, 2 işlem için yeterli olacak işlem senkronizasyonu (mutual exclusion) gerçekleştirilmektedir.

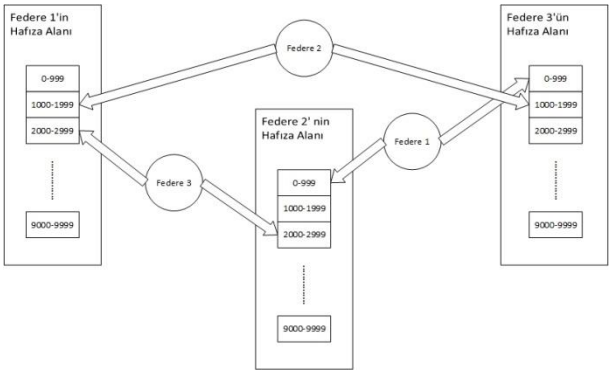
4.3. Paylaşımlı-Hafıza Modelinde Federenin Hafıza Alanını Okuması (Reading the Federate Memory Region in Shared-memory Model)

Federeler kendilerine ait hafıza alanlarını okuyabilmek için yerel Portico bileşenlerinin kendilerine sağladığı hafıza okuma metodlarını belirli zaman aralıklarıyla çağırırlar ve gelen mesajları okuyup işlerler. Şekil 4.5'te görüldüğü gibi, federeler daha önce bahsedilen her bir mesaj dizini bölmesi için ayrı bir işlem parçası çalıştırırlar. Böylece her bir bölmedeki mesajları paralel olarak okuyup işleyebilirler.

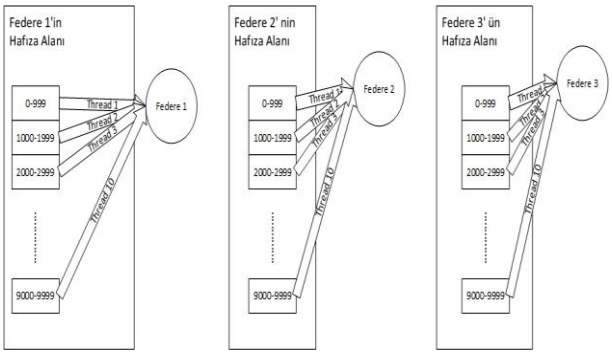
[7] http://en.wikipedia.org/wiki/Peterson's_algorithm (17 Ağustos 2014)

4.4. Hibrit İletişim Modeli (Hybrid Communication Model)

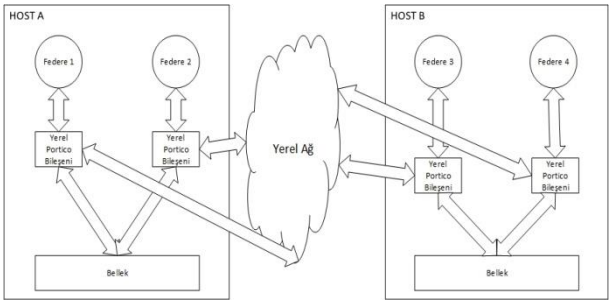
Paylaşımlı-hafıza modeli, ancak tüm federelerin aynı bilgisayar üzerinde çalıştığı durumlarda kullanılabilecek yüksek performanslı bir modeldir. Birden fazla bilgisayara dağılmış olan ve her bilgisayarda birden fazla federesi bulunan federasyonlar için hibrit bir yapının gerekliliği çok açıktır. Böyle bir durumda, Şekil 4.6'daki gibi, aynı bilgisayardaki federeler birbirileriyle paylaşımlı hafıza alanı üzerinden, diğer bilgisayardaki federelerle ise ağ üzerinden iletişim kurabileceklerdir.



Şekil 4.4. Federelerin birbirlerinin hafıza alanlarına erişmeleri



Şekil 4.5. Federelerin hafıza alanlarındaki mesajları okuması (Federates' process of reading from memory)



Şekil 4.6. Hibrit ortamda federe haberleşmesi (Federates' communication in hybrid environment

Görüldüğü gibi hibrit bir ortamda federasyon da hibrit bir yapıya sahiptir. Hibrit iletişim modelinde her bir federasyon, bir federasyon hafıza alanı ve bir Jgroups ka-

nalından oluşmaktadır. Böylece aynı bilgisayardaki federeler ortak hafıza alanı üzerinden, ağ üzerine dağılmış federeler ise Jgroups üzerinden haberleşebileceklerdir.

5.TEST SONUÇLARI (TEST RESULTS)

Yukarıda anlatılan yeni iletişim modelleri, Jgroups'un hali hazırdaki iletişim altyapısını oluşturan Jgroups iletişim modeli ile karşılaştırılmıştır.

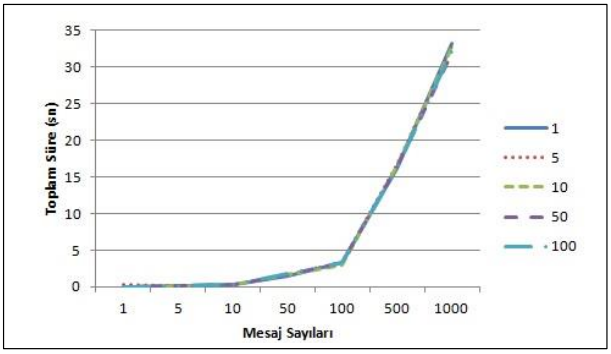
5.1. Tek Bilgisayarlı Testler (Federates on Single Computer)

İlk test kurgusunda 2 federe aynı bilgisayar üzerinde koşmaktadır. Testin ilk aşamasında federeler Jgroups üzerinden, ikinci aşamasında ise paylaşımlı-hafıza modeli üzerinden haberleşmektedirler. Test bilgisayarı 2.67 GHz Intel Core i5 M480 işlemci, 8GB RAM ve Windows 8.1 64 bit işletim sistemi barındırmaktadır. Test senaryosu olarak, federeler birbirilerine değişken miktarda mesaj göndermektedirler. Her bir deney 100 defa tekrar edilip bu tekrarlar sonucunda standart sapma değeri de hesaplanmaktadır. Bu deney sonucunda Şekil 5.1'deki grafik elde edilmiştir. Şekil 5.2'de ise aynı test ortamında, aynı federeler bu sefer paylaşımlı-hafıza modelini kullanmaktadırlar. Sonuçlardan da görüldüğü üzere 1000 mesajlık bir haberleşme simülasyonunu, Jgroups modelini kullandıklarında 32.27 saniyede tamamlayan federeler, paylaşımlı-hafıza modelini kullanarak bu simulasyonu 7.37 saniyede tamamlamaktadır. Buradan görüleceği gibi paylaşımlı-hafıza modeli iki federeyi daha hızlı bir şekilde haberleştirebilmektedir. Paylaşımlı-hafıza modelinde, Jgroups'ta olan ağ tabanlı kontrollerden kaynaklanan süre kaybı yaşanmamaktadır. Her iki modelin karşılaştırması Şekil 5.3' te görülmektedir. Buna göre standart sapma değerleri dahil edildiğinde bile paylaşımlı-hafıza modeli Jgroups'tan daha iyi bir performans sergilemektedir.

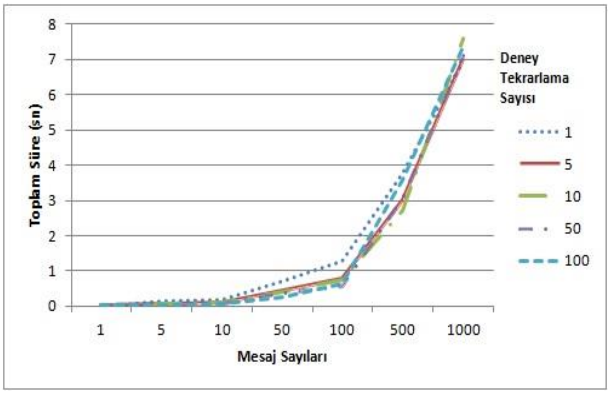
5.2. Çift Bilgisayarlı Testler (Federates on Two Computers)

Bu test kurgusunda her bir bilgisayarda 2 tane olmak üzere, toplam 2 bilgisayarda 4 federeden oluşan bir federasyon yapısı vardır. Bilgisayarlardan birisi 2.67 GHz Intel Core i5 M480 işlemci, 8GB RAM ve Windows 8.1 64 bit işletim sistemine sahipken diğeri, 2.40 Ghz Intel Core i7 3630QM işlemci, 32GB RAM ve Windows 8.1 64 bit işletim sistemine sahiptir. İlk test senaryosunda bu federeler Jgroups iletişim modelini kullanarak haberleşirken, ikinci test senaryosunda hibrit iletişim modelini kullanarak haberleşmektedirler.

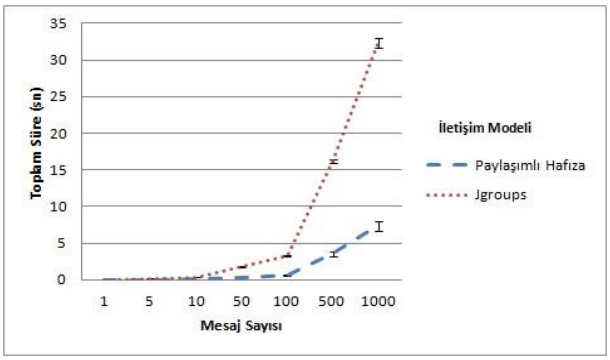
Şekil 5.4' te görüldüğü gibi 4 federe Jgroups modelini kullanarak 1000 mesajlık simulasyonlarını 33.23 saniyede tamamlarken, hibrit iletişim modelinde bu simulasyonu 16.09 saniyede tamamlamaktadır. Standart sapma değerleri dahil edilerek elde edilen sonuçlarda hibrit iletişim modeli, Jgroups'tan daha iyi performans sergilemektedir.



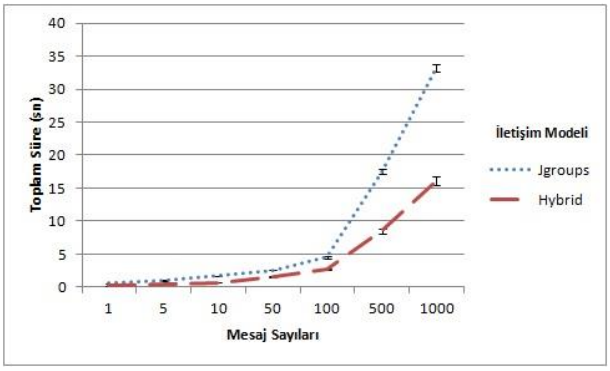
Şekil 5.1 Jgroups test sonucu (Jgroups test results)



Şekil 5.2 Paylaşımlı Hafıza test sonucu (Shared-memory test results)



Şekil 5.3 Paylaşımlı Hafıza ve Jgroups modellerinin karşılaştırılması (Comparison of Jgroups and shared-memory models)



Şekil 5.4 Hibrit Modelin ve Jgroups Modelinin karşılaştırılması (Comparison of Hybrid model and Jgroups model)

6. SONUÇ (CONCLUSION)

Bu çalışmada Portico RTI için 2 yeni iletişim modelinin yapıları anlatılmış ve bu yapının Portico'nun hali hazırdaki iletişim modeli olan Jgroups modeliyle karşılaştırması sonucunda elde edilen veriler paylaşılmıştır. Elde edilen sonuçlardan görüldüğü gibi paylaşımlı hafıza modeli Jgroups modelinden daha hızlı mesaj iletimi sağlamıştır. Aynı şekilde hibrit model de Jgroups modelinden daha etkin ve hızlı bir iletişim modelidir. Tüm bu veriler göz önüne alındığında paylaşımlı hafıza ve hibrit modeller simülasyon sistemlerinin haberleşmesinde daha hızlı bir çözüm olarak kullanılabilecektir. Paylaşımlı hafıza modelinin daha hızlı çalışmasının nedeni şüphesiz işlemler (process) arası mesajların hafıza alanına yazılıp oradan okumasıdır. Jgroups modelinde ise mesajlar her seferinde Jgroups'un ağ üzerinden haberleşme sağlayan metotlar kullanılarak gönderilmekte ve aynı şekilde kabul edilmektedir. Bu nedenle Jgroups doğası gereği yavaş bir iletişim modeli seçeneği olarak kalmaktadır. Hibrit iletişim modelinde ise aynı bilgisayarlar üzerinde paylaşımlı hafıza modeli kullanıldığından, toplam mesaj gönderim zamanı saf Jgroups (pure Jgroups) modeline göre daha iyi performans göstermektedir. Bu çalışmada hafıza operasyonlarını içeren SharedMemory.DLL kütüphanesi sadece Windows işletim sistemi üzerinde çalışmaktadır. İlerleyen süreçlerde bunun Linux üzerinde çalışan versiyonu hazırlanacaktır. Geliştirilen yeni modeller diğer açık kaynak kodlu RTI'lar ile entegre edilmeye çalışılacaktır.

7. KAYNAKLAR (REFERENCES)

[1] IEEE standard for modeling and simulation (M&S) High Level Architecture (HLA) framework and rules. IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000), pages 1-38, Aug (2010)

[2] IEEE standard for modeling and simulation (M&S) High Level Architecture (HLA)- Object Model Template (OMT) specification. IEEE Std 1516.2-2000, pages I - 130, (2001)

[3] IEEE standard for modeling and simulation (M&S) High Level Architecture (HLA) federate interface specification, (Revision of 1516.1-2000) – Redline: IEEE Standard for Modeling & Simulation (M & S) High Level Architecture (HLA) Federate Interface Specfication - Redline, page 1, (2011)

[4] K. L. Morse and M. D. Petty. High Level Architecture Data Distribution Management migration from DoD 1.3 to IEEE 1516. ***Concurrency and Computation: Practice and Experience***, 16(15):1527-1543, (2004)

[5] M. B. Martin Adelantado. HP-CERTI: Towards a high performance, high availability open source RTI for composable simulations. ***04F-SIW-014***, 1(1):2, (2004)

[6] M. RTI. MAK RTI connection technology. MAK RTI Web Site,(Erişim tarihi: 13 Temmuz 2014)

[7] P. RTI. Pitch RTI connect and incorporate. Pitch RTI Web Site, (Erişim tarihi: 12 Temmuz 2014)

[8] L. Malinga and W. H. le Roux. Hla rti performance evaluation. ***In Proceedings of the 2009 SISO European Simulation Interoperability Workshop***, ***SIW '09***, pages 39-44, Vista, CA, (2009)

[9] R. L. Pamella Knight, Aaron Corder. Evaluation of run time infrastructure (RTI) implementations. ***In 2001 Fall Simulation Interoperability Workshop***, 9-14 September, (2001)

[10] K. L. Morse, D. L. Drake, and R. P. Brunton. Web enabling HLA compliant simulations to support network centric applications. ***In Proceedings of the 2004 Symposium on Command and Control Research and Technology***, San Diego, CA, (2004)

[11] M. Dragoicea, L. Bucur, W. Tsai, and H. Sarjoughian. Integrating HLA and service oriented architecture in a simulation framework. ***(Proceedings - 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing***, CCGrid 2012):861-866, (2012)

[12] W. Zhang, T. Zhang, and Y.-B. Zha. Web service enabling of HLA based distributed simulation. ***Guofang Keji Daxue Xuebao/Journal of National University of Defense Technology***, 30(5):120-124, (2008)

[13] T. Zhiying, G. Zacharewicz, and D. Chen. Developing a web-enabled HLA federate based on Portico RTI. ***Proceedings of the 2011 Winter Simulation Conference (WSC)***, page 22-89, (2011)

[14] A. Kapolka. The extensible run-time infrastructure (XRTI): An emerging middleware standard for interoperable networked virtual environments. In Level Architecture, ***Master's thesis***, Naval Postgraduate School, (2003)

[15] W. Zhiteng, Q. Zhao, Z. Hongjun, Z. rui, X. Ying, and Y. Dejun. The application of MDA in distributed services of run-time infrastructure. ***Advances in Computer Science and its Applications ACSA***, 41(2), (2012)

[16] Z. J.-z. XU Ting-xue. Design and implementation of missile equipment martial and local integration maintenance support simulation system based on HLA. ***Systems Engineering - Theory & Practice***, 33(3):802, (2013)

[17] Z. Zhou, C. Xu, W. Hou, and W. Wu. Distributed ownership management algorithm for HLA/RTI implementation. ***Journal of Computational Information Systems***, 7(5):1628- 1637, (2011)

[18] L. Lu, Z. Zhou, W. Wu, and Q. Zhao. DVE-RTI: Distributed interactive simulation runtime infrastructure. ***Jisuanji Yanjiu yu Fazhan/Computer Research and Development***, 41(5):828-834, (2004)

[19] J. Byrne, C. Heavey, and P. Byrne. A review of web-based simulation and supporting tools. ***Simulation Modeling Practice and Theory***, 18(3):253 - 276, (2010)

[20] Z. Wang, H. Zhang, A Run-time infrastructure based on service-distributed architecture, ***Applied Mathematics & information sciences***, 1 June, (2014)

[21] P. P. W. Page. Portico history. Portico Project Web Page, 2(2):1, (2008)