İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

HİERARŞİK, ARAMA YAPILABİLEN, GÜVENLİ VE KALICI YAYINLA/ABONE YAZILIM ÇATISI

YÜKSEK LİSANS TEZİ

Pınar OSANMAZ ÇELİK

Anabilim Dalı : Bilgisayar Bilimleri

Programı : Bilgisayar Bilimleri

Tez Danışmanı: Prof. Dr. Nadia ERDOĞAN

HAZİRAN 2011

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

HİYERARŞİK, ARAMA YAPILABİLEN GÜVENLİ VE KALICI YAYINLA/ABONE YAZILIM ÇATISI

YÜKSEK LİSANS TEZİ
Pınar OSANMAZ ÇELİK
(704081010)

Tezin Enstitüye Verildiği Tarih : 10 Mayıs 2011
Tezin Savunulduğu Tarih : 09 Haziran 2011

Tez Danışmanı : Prof. Dr. Nadia ERDOĞAN (İTÜ)
Diğer Jüri Üyeleri : Yrd. Doç. Dr. Feza BUZLUCA (İTÜ)
Prof. Dr. Coşkun SÖNMEZ (YTÜ)

HAZİRAN 2011

Sevgili kızıma,

ÖNSÖZ

Çalışmalarımın her aşamasın da bana desteklerini esirgemeyen danışmanım Prof. Dr. Nadia ERDOĞAN'a teşekkürlerimi sunarım. Ayrıca desteklerini her zaman yanımda hissetiğim değerli aileme de teşekkürlerimi sunarım.

Haziran 2011

Pınar Çelik Osanmaz

(Bilgisayar Mühendisi)

İÇİNDEKİLER

Sayfa

KISALTMALAR

JMS **:** Java Message Service
JPA **:** Java Persistence API
WSE&N **:** Web Service Eventing and Notification
API **:** Application Programming Interface

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

HİYERARŞİK, ARAMA YAPILABİLEN, GÜVENLİ VE KALICI YAYINLA / ABONE YAZILIM ÇATISI

ÖZET

Java Message Service(JMS)[1] , Web Service Eventing and Notification (WSE&N) [2]gibi dağıtık yayınlama/abonelik (distributed publish/subscribe) sistemleri kuyruklama (queueing), yayınlama (publishing) ve abonelik (subscribing) gibi temel servisler geliştiricilere sağlamaktadır. Ancak doğrulama (authentication), yetkilendirme (authorization), kalıcılık (persistence), hiyerarşi (hierarcy) ve keşif (discovery) konularında ne JMS ne de WSE&N herhangi bir destek sağlamamaktadır. Bu servisleri tamamen geliştirmek ya da geliştirilmiş kütüphaneleri kullanarak bu işlevleri gerçekleştirmek geliştiricilerin görevi olarak addedilmektedir.

Söz konusu servislerde dağıtık yayınlama/abonelik sistemlerinin karmaşık yapısı ve çok sayıda işlevin bir arada uyumlu bir biçimde çalışma zorunluluğu nedeniyle makul bir sürede hatasız yazılım geliştirme son derece zordur. Bu çalışmada JMS 'in sağladığı olanaklara ek olarak doğrulama, yetkilendirme, kalıcılık, hiyerarşi ve keşif hizmetlerini içerisinde barındıran, tak çalıştır yöntemiyle geliştiricilerin kullanabileceği ek bir ara yazılım çerçevesi tasarlanmış ve gerçeklenmiştir. Yazılım çerçevesinin kullanımın örneklendirmek ve işlevlerini sınamak için örnek bir uygulamayı da ayrıca geliştirilmiştir.

HIRARCHICAL, SEARCHABLE, SECURE AND PERSISTENT PUBLISH / SUBSCRIBE FRAMEWORK

SUMMARY

Distributed publish/subscribe systems such as Java Message Service, Web Service Eventing and Notifications supply basic services such as queueing, publishing, subscribing. All the other services related to applications such as authentication, authorization, persistence, hierarchy and discovery is left to the application development. Even if all these services are already implemented in many distributed enterprise server or service-oriented middleware, it is development team's responsibility to wire them to make any application to fulfil any useful functioanlity.

Since this is a quite complicated software development process, there is a need a common framework for distributed publish and subscribe systems which includes standard service such as authentication, authorization, persistence, hierarchy and discovery. We develop such as software infrastructure. As a sample, we developed an e-learning application using the framework.

1. GİRİŞ

Son yıllarda, insanlar iletişimlerinin birçoğunu Web üzerinden yapma eğilimindedir. Bu yaklaşım hem zaman kazancı hemde daha çok bilgi paylaşımına olanak sağlamaktadır. Tabi bu bilgilerin kolay yayınlama ve yayınlanan bilgilere ulaşmak için çeşitli araçlar geliştirilmiştir. Birçok insan ilgilendiği, merak ettiği, ya da sorumlu olduğu konulardan haberdar olmak ister. Ayrıca sadece o bilgilerin kendine gelmesini isteyebilir. Böylelikle, hem daha hızlı bilgiye ulaşmış hem de ağ üzerindeki dar bant durumunu optimum şekilde kullanmış olur. Bu durumlar için web üzerinde Yayınla/Abone(Publish/Subscribe) sistemler mevcuttur.[10] Bu tür sistemler kişiye ilgilendiği konuya abone olma olanağı sağlar. Sadece o konuyla ilgili mesajlar kendisine iletilir. Ayrıca kişi istediği mesajı da kolayca yayınlayabilir.[7] Bu tez çalışmasında geliştirilen yazılım, yayınla/abone ol hizmetlerini sağlayan Java Message Service üzerinde geliştirilmiştir.

Yayınla/Abone sistemleri, örneğin Java Message Service (JMS), abonelik, asenkron mesajlaşma gibi temel servisleri geliştiricilere sunmaktadır.[11] Ancak Yayınla/Abone sistemiyle çalışan yazılımlar, temel servislerin ötesinde güvenlik (security), kalıcılık (persistence), hiyerarşik yayınlama ve nitelik tabanlı erişim gibi hizmetlere ihtiyaç duymaktadır. Bu hizmetler için JMS herhangi bir destek sağlamamaktadır. Yayınla/Abone yöntemiyle yazılım geliştirenler için aşağıdaki sorunların çözümü için herhangi bir destek bulunmamaktadır:

- Kurumsal Doğrulama ve Yetkilendirme Sistemlerinin Karmaşıklığı
- Mesaj Sistemlerinde Kalıcılık Desteği Eksikliği
- Hiyerarşik ve Yol(Path) Tabanlı Teslimat ve Erişim Desteği
- Çalışma Zamanında Konu Oluşturma Desteği Eksikliği
- Geleneksel Mesajlaşma Sistemlerindeki Performans Sorunu

Bu tür standart sorunlara geliştiriciler standart olmayan çözümler geliştirmek durumunda kalmakta, JMS gibi servislerdeki eksiklikleri kendileri giderme yoluna gitmektedirler. Biz burada yaptığımız çalışma ile yukarıda bahsedilen eksiklikleri gidermek için bir altyapı önerisi geliştirdik. Geliştirdiğimiz yazılım çatısı ile doğrulama ve yetkilendirme (kullanıcılar, roller, yetkiler), kalıcılık (persistence, mesajların yayınlandıktan çok sonra da okunabilmesi), konular arasında hiyerarşik yol ve nitelik desteği (mesajların ve konuların ağaç yapısında kategorik işlenmesi) ve mesaj trafiğinde performans kazancı (alıcıların sadece ilgilendikleri mesajları almaları) gibi konularda her uygulamanın tak-çalıştır biçiminde kolay bir biçimde kullanabileceği bir ortam sağlanmaktadır.

II. bölümde Mesajlaşmanın tanımı ve mesajlaşma tipleri incelenmiştir III. Bölümde Yayınla/Abone sistemleri açıklanmıştır. IV. bölümde Java Message Service tanımı ve temel kavramları açıklanmıştır. V. bölümde Abone/Yayınla Uygulama geliştirmenin temel sorunlarına yer verilmiştir. VI. bölümde sorunlara ilişkin çözüm önerileri sunulmuştur. VII. Bölümde örnek bir uygulama anlatılmıştır. Son olarak VIII. bölümde sonuçlar tartışılmaktadır.

2. MESAJLAŞMA

Mesajlaşma, yazılım veya uygulamalar arasında bir iletişim metodudur. İki biçimde gerçekleştirilebilir: senkron veya asenkron. Senkron haberleşmede mesajı alan kişi mesajın gönderildiği sırada karşı tarafta olmalı, gönderim sırasında mesajları almaya devam etmelidir. Bunu telefonla konuşmaya benzetebiliriz. Bir telefon görüşmesinin gerçekleşmesi için konuşan kişi kadar dinleyen kişinin de hatta etkin konumda olması gerekir. Asenkron iletişim ise alıcının gönderim sırasında etkin olmadığı mesajlaşma biçimidir[12]. Gönderici mesajı belli bir konuma yollar, alıcı mesajı uygun olduğu sırada, genellikle mesaj gönderildikten bir süre sonra alır ve değerlendirir. Bu tür iletişime örnek olarak mektuplaşma verilebilir. Mektup yazıldığı veya gönderildiği sırada okuyacak kişinin hazır durumda olması gerekmez. Web Servisleri gibi client/server tarzında çalışan sistemler senkron haberleşme sağlarken Java Message Service (JMS) asenkron iletişim sağlamaktadır. Abone/Yayınla sistemlerinde asenkron haberleşmeyi destekler.

3. YAYINLA/ABONE SİSTEMLERİ

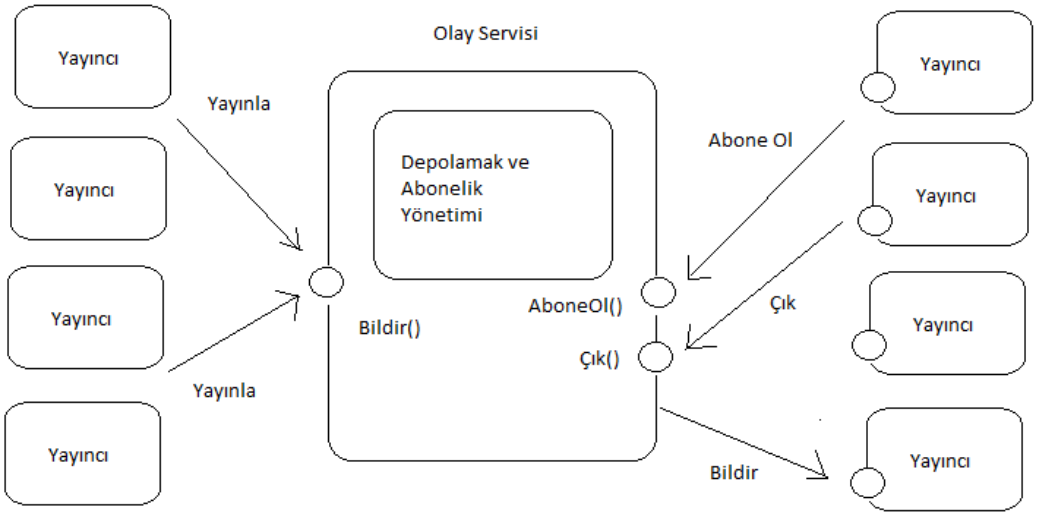
Yayınla/Abone (Publish/Subscribe) adı verilen sistem, göndericilerin alıcılara doğrudan mesaj göndermediği, mesajların belli konulara gönderilmesi ve alıcıların mesajları bu konulara abone olmak suretiyle alması biçiminde çalışır. Bu yaklaşımda mesajlar, bir alıcı tarafından alınıp alınmayacağı veya hangi alıcıların mesajları alacağı dikkate alınmadan yayınlanır. Aynı biçimde alıcılar, mesajları hangi göndericilerin ürettiği ile ilgilenmez, sadece mesajın hangi konuda olduğuyla ilgilenir. Bu biçimde göndericiler (yayıncılar) ve alıcılar (abone) birbirlerini tanımadan iletişim kurmuş olurlar. Bu şekilde hem gevşek bir network sistemi hem de daha değişime açık bir yapı kurulmuş olur. Sonradan sisteme bir yayıncı veya abone eklenmesi yapıda herhangi bir değişikliği gerektirmez. Abone/Yayınla sistemleri, Mesaj Kuyruğu sistemlerine alternatif oluştururlar. Mesaj Kuyruğunda göndericiler belli bir alıcıya mesaj gönderirler. Başka bir deyişle gönderici kime gönderdiğini alıcı da kimden aldığını bilir. Genellikle Java Message Service (JMS) sistemleri hem Publish/Subscribe hem de Mesaj Kuyruğu biçimlerini desteklemektedir.

Yayınla/Abone sistemlerinde olağan durumlarda alıcılar mesajların belli bir kısmını alırlar. Başka bir deyişle mesajlar alıcı tarafından süzülerek alınır. Mesajların süzülmesi üç yöntemle gerçekleştirilebilir.

- Konu Tabanlı Süzme : Bu durumda mesajlar belli konulara gönderilir ve alıcı o konuya abone olur. Belli bir konuya gönderilen mesajlar sadece o konuya abone olan alıcılar tarafından alınabilir.
- İçerik Tabanlı Süzme : Bu durumda mesajların içeriğine ilişkin nitelikler belirtilir ve alıcı o niteliğe sahip mesajları alır. Başka bir deyişle belli bir niteliğe sahip her mesaj hangi kategoride olursa olsun alıcı tarafından içeriğine göre alınır veya alınmaz[13].

- Hem Konu Hem İçerik Tabanlı Süzme : Bazı sistemler her iki süzme biçiminin bir karışımını destekler. Göndericiler bir konuya mesaj gönderir. Ancak alıcılar o konuya gelen mesajlardan sadece belli bir niteliğe sahip olanları alır.

Yayınla /Abone sistemler [3] üç bileşen içerir: Yayınlayıcılar, sistemde bilgi üretir ve besler, aboneler, özel bir bilgi ile ilgilenir ve alt yapı sağlayıcısı, yayınlanan içerikleri ilgili aboneleri eşleştirir ve ilgili abonelere bilgiyi dağıtır.



Şekil 3.1 : Yayınla/Abone sistemi tabanlı basit bir nesne.

Yayınla/abone sistemlerinde, abone ile yayınlayıcılar zamansal, yer ve senkronizasyon açısından birbirinden bağımsızlardır.[4] Böylelikle, daha geniş ölçeklenebilinir sistemlerin oluşmasına olanak tanır.

Şekil 3.1 görüldüğü gibi, abone bir olaya abone olmak istediğinde olay servisi üzerinde bulunan AboneOl() operasyonunu çağırır. Abone olma bilgisi olay servisi üzerinde kalır ve yayıncılara iletilmez. Aboneler abonelikten çıkmak istediklerinde de Çık() operasyonunu çağrılır. Bir olay yaratmak içinde, yayıncılar Bildir() operasyonunu çağrılır. Olay servisi yayınlanan olayı ilgili abonelere iletir.

Olay servisi, yayıncılar ve aboneler arasında zaman,yer ve sekronizasyon bağımsızlıkları sağlar:

-Zaman: Yayıncı ve abone aynı anda haberleşmek zorunda değildir. Yayıncı olayı yayınladığında abone bağlı olmaya bilir. Ya da, abone olaydan haber olduğunda yayıncı sisteme bağlı olmayabilir.

-Yer: Yayıncılar ve aboneler genelde birbirlerinin referanslarını tutmazlar. Yani ne yayıcının abone hakkında bilgisi nede abonenin yayıncı hakkında bilgisi vardır.

-Senkronizasyon: Yayıncılar mesaj gönderirken bloke edilmezler ve aboneler bazı eş zamanlı işler yaparken bilgiyi alabilirler. Üretim ve tüketim olaylarında aboneler ve yayıncılar arasında ana akış kontrolü olmaz. Bu yüzden eş zamanlı bir yönetim değildir.

4. JAVA MESAJLAŞMA SERVİSİ

4.1 Giriş

JMS (Java Message Service) iki veya daha fazla istemci arasında mesaj alış verişini sağlayamaya yönelik bir yazılım standardı ve kütüphanesidir [1]. JMS, Java Platformu Kurumsal Sürümü'nün bir parçasıdır. Bir ağda ayrı makinelerde çalışan Java uygulamalarının aralarında mesaj oluşturma, gönderme, alma ve okuma hizmetleri sağlar. Dağıtık uygulamaların farklı bileşenleri arasında gevşek bağlanmış (loosy-coupled), güvenilir ve asenkron (asynchronous) iletişim kurulmasını sağlar.

4.2 JMS' in Temel Öğeleri

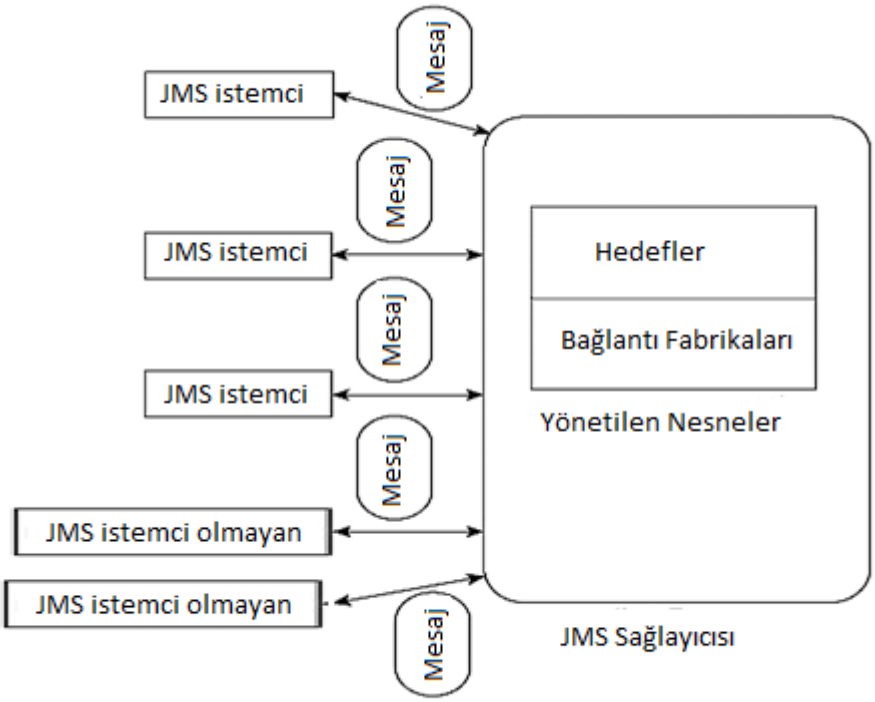
Sağlayıcı (Provider) : JMS arayüzünü gerçekleştiren uygulamadır. Kendisi de Java 'da yazılmış olabilir veya başka dilde yazılmış gerçekleştirmeleri uyarlayabilir.

JMS İstemci (JMS Client) : Mesaj üreten ve alan uygulama veya süreç. JMS istemcileri Java diliyle yazılmalıdır. [9]

Non JMS İstemci(Non-JMS Client) : Non JMS istemcileri JMS API yerine farklı API kullanarak mesaj alan ve üreten uygulamalardır.

Yönetilen Nesneler (Administered Objects) : İstemcilerin bu nesleri kullanarak JMS sağlayıcı ile bağlantı kurarlar.

Mesaj (Message) : Mesajlar mesajlaşma servisinin temelini oluşturur. Bir nesnedir ve istemcilere arasında bilgi alış verişini sağlar.



Şekil 4.1 : JMS mimarisinin beş ana unsuru.

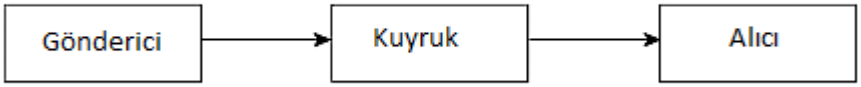
4.3 JMS Modelleri

JMS istemcilerin ve gönderimin yapısısna gore iki modeli destekler. Bu modeller Noktadan-Noktaya (Point-To-Point) ve Yayıncı/Abone (Publish/Subscribe) modelleridir.

4.3.1 Noktanadan Noktaya Modeli : Kuyruk Yapısı

Nokatadan-Noktaya iletişimde gönderici mesajı bir kuyruğa gönderir ve alıcı mesajı alır. Gönderici mesajın nereye gideceğini bilir ve ona göre hedefini belirler. Noktadan-Noktaya iletişim özelikleri şunlardır[14]:

- Mesaj sadece tek bir tüketiciye gider.
- Mesaj gönderilirken tüketicinin, alınırken de üreticinin çalışır durumda olmasına gerek yoktur.
- Başarılı olarak gönderilmiş her mesaj için tüketiciden üreticiye kabul edildiğine dair bilgi gider.

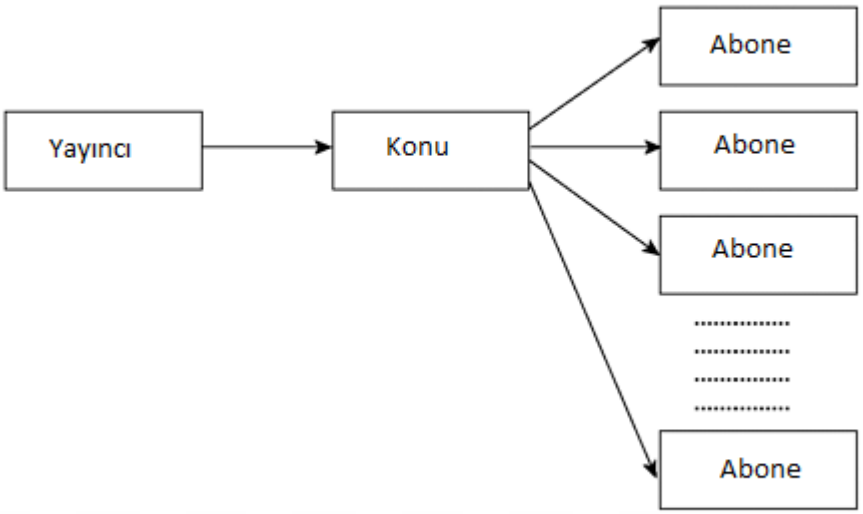


Şekil 4.2 : Noktandan noktaya mesaj modeli.

4.3.2 Yayınla/Abone Modeli : Konu Yapısı

Yayınla /Abone modelinde belli bir konu için mesaj gönderilmesine olanak verilir. Bir istemci belirli bir konu için mesaj yayınlar. Diğer bir istemci belirli bir konuya abone olur. Ne yayıncı ne de abone birbirini tanımaz. Yayıncı/ Abone modeli aşağıdaki özelliklere sahiptir[14]:

- Mesaj birden çok kullanıcıya gidebilir veya hiç kimseye gitmeyebilir.
- Aboneler abone olduktan sonra yayınlanan mesajı alabilirler. Noktandan Noktaya iletişimde yeni bir tüketici kuyruktaki tüm mesajları alabilir. Fakat Yayınla/Abone modelinde abone sadece abone olduktan sonraki mesajları alabilir, daha önce yayınlanmış mesajları alamaz.



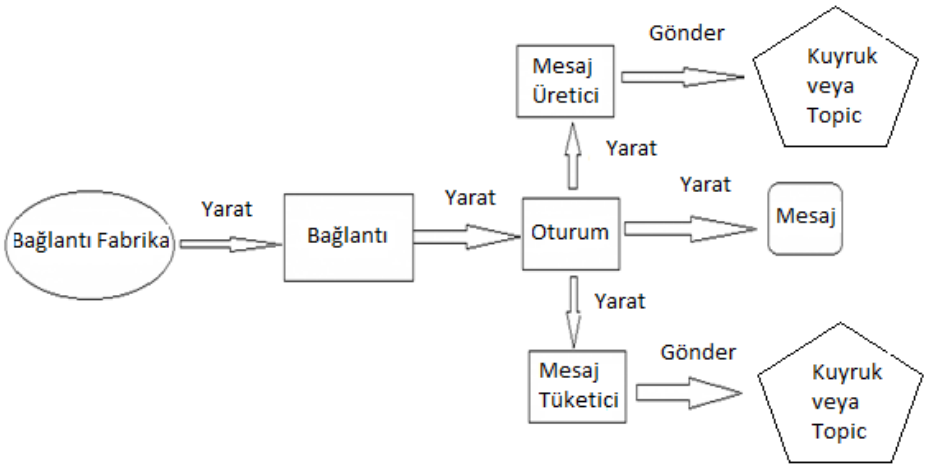
Şekil 4.3 : Yayınla/Abone mesaj modeli.

4.4 JMS API Programlama Modeli

Bir JMS uygulaması aşağıdaki temal yapı taşlarından oluşmaktadır:

- Yönetilen Nesneler
- Bağlantılar(Connections)
- Oturumlar(Sessions)
- Mesaj Üretici
- Mesaj Alıcı
- Mesajlar

Şekil 4.4 de görülen tüm nesneler bir JMS istemci uygulamasında yer alır.



Şekil 4.4 : JMS API programlama modeli.

4.4.1 Yönetilen Nesneler

Bu nesneler JMS programlarının dışında tutulur. . İki farklı yönetilen nesne vardır: Bağlantı Fabrikası (Connection Factory) ve Hedefler (Destinations) Bir yönetici tarafından bu nesneler yaratılr veya yok edilir.

4.4.1.1 Bağlantı Fabrikası (Connection Factory)

İstemcinin JMS sağlayıcısı ile bağlantı kurması için bir Connection Factory nesnesini yaratması gerekir. Kuyruk veya konu olabilir. Bir Bağlantı Fabrika bağlantı yapılandırma parametreleri kümesini içerir. Aşağıdaki komutlar ile yaratılabilir:

j2eeadmin -addJmsFactory jndi_name queue

j2eeadmin -addJmsFactory jndi_name topic

JMS istemci uygulaması JNDI API kullanarak Connection Factory[8] nesnesine ulaşır.

```
Context ctx = new InitialContext();
QueueConnectionFactory queueConnectionFactory =
  (QueueConnectionFactory) ctx.lookup("QueueConnectionFactory");
TopicConnectionFactory topicConnectionFactory =
  (TopicConnectionFactory) ctx.lookup("TopicConnectionFactory");
```

4.4.1.2 Hedefler (Destinations)

Hedef mesajın gideceği varış noktasıdır. JMS de iki tür hedef vardır. Kuyruk(queue) ve konu(topic) hedefleridir. Noktdan Noktaya iletişim modelinde kuyruk, Yayınla Abone modelinde konu(topic) hedefi kullanılır ve aşağıdaki şekilde yaratılır:

j2eeadmin -addJmsDestination <jndi_name_for_queue> queue

j2eeadmin -addJmsDestination <jndi_name_for_topic> topic

Ek olarak hedef nesnesi yaratmak için :

```
Topic myTopic = (Topic) ctx.lookup("MyTopic");
Queue myQueue = (Queue) ctx.lookup("MyQueue");
```

4.4.2 Bağlantılar (Connections)

Bağlantı istemci ve JMS sağlayacı arasında bir bağlantı kurar. Gerçekte bu bir soket bağlantısıdır. İstemci uygulama geliştiricisi düşük seviye socket programlama ile uğraşmaz. Bunu yerine ConnectionFactory sınıfının createConnection methodunu kullanarak bir bağlantı oluşturur. Connection nesnesini kullanarak bir veya daha fazla oturum yaratılabilinir. QueueConnectionFactory veya TopicConnectionFactory nesneleri bağlantı yaratmak için kullanılır.

QueueConnection queueConnection = queueConnectionFactory.createQueueConnection();

TopicConnection topicConnection = topicConnectionFactory.createTopicConnection();

Uygulama sonlandığı zaman , yaratılan herhangi bir bağlantının kapatılması gerekir.

queueConnection.close();

topicConnection.close();

Uygulama mesajı tüketmeden önce bağlantının start metodu kullanılmalıdır.

4.4.3 Oturumlar (Sessions)

JMS oturumları mesaj gönderip alırken kullanılan yardımcı bir sınıftır. Oturum nesnesi mesajı üreticilerden tüketicilere iletmekten sorumludur. İki tür oturum vardır: TopicSsesion ve QueueSession. Bir oturum oluşturmak için Connection sınıfı kullanılır.

TopicSession topicSession = topicConnection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);

QueueSession queueSession = queueConnection.createQueueSession(true, 0);

4.4.4 Mesaj Üreticileri (Message Producers)

Bir mesaj yaratmak için Mesaj Üretici nesnesini yaratmak gerekir. Session sınıfını kullanarak Mesaj Üretici nesnesi yaratılır.

QueueSender queueSender = queueSession.createSender(myQueue);

TopicPublisher topicPublisher = topicSession.createPublisher(myTopic);

Oluşturulan bir mesajı göndermek için aşağıdaki komut kullanılır:

queueSender.send(message);

topicPublisher.publish(message);

4.4.5 Mesaj Tüketicileri (Message Consumers)

Mesajı hedeften almak için Message Consumer nesnesi kullanılır. Session nesnesi kullanarak Consumer oluşturulur.

QueueReceiver queueReceiver = queueSession.createReceiver(myQueue);

TopicSubscriber topicSubscriber = topicSession.createSubscriber(myTopic);

Bir mesaj tüketici yaratıldığında, bağlı olduğu hedefe(kuruk veya konu) aktif olduğunu söylemeli. Mesaj tüketicisi Close metodunu kullandıktan sonra mesaj alamaz. Receive metodunu kullanarak sadece senkron olarak mesaj alınır.

Connection.start();

Message m= consumer.receive();

Diğer yandan Message Listener asenkron şekilde mesaj almak için kullanılan arayüzdür. Bu arayüzü OnMessage () adında tek bir metoda sahiptir. Bir mesaj geldiği zaman bu metot otomatik olarak yürütülür. Message Listener yaratıldıktan sonra, message listener nesnesini mesaj tüketicisi ile ilişkilendirmek için setMessageListener metodu kullanılır.

Listener myListener = new Listener();

Sonsumer.setMessageListener(myListener);

4.4.6 Mesajlar (Messages)

Mesajlar bilgi içeren kısımlardır ve üç parçadan oluşurlar.

- Mesaj Başlığı(Message Header)
- Mesaj Özellikleri(Message Properties)
- Mesaj Gövdesi(Mesaj Body)

Bu parçalardan sadece mesaj başlığı zorunludur, diğerleri opsiyoneldir.

4.4.6.1 Mesaj Başlığı

Mesaj başlığı bazı zorunlu alanlar içerir. Bu alanlar istemci ve sağlayıcı mesajları tanımlamak ve yönlendirmek için kullanırlar. Örneğin JMSMessageID alanı mesajın benzersiz (unique) tanımlayıcısıdır. JMSDestination alanı mesajın gönderileceği kuyruk veya hedefi gösterir.

Her bir başlık alanı mesaj arayüzünde tanımlanan getter ve setter metotlarına sahiptir. Aşağıdaki tabloda JMS mesaj başlığı alanlar gösterilmiştir:

Çizelge 4.1 : JMS başlık alanları.

Başlık Alanı	Belirleyen Taraf
JMSDestination	Gönder veya yayınla metodu
JMSDeliveryMode	Gönder veya yayınla metodu
JMSExpiration	Gönder veya yayınla metodu
JMSPriority	Gönder veya yayınla metodu
JMSMessageID	Gönder veya yayınla metodu
JMSTimeStamp	İstemci
JMSCorrelationID	İstemci
JMSType	İstemci
JMSRedelivered	İstemci

4.4.6.2 Mesaj Özellikleri

Mesaj içinde mesaj özellikleri yaratılabilir. Örneğin, sipariş fiyatı özelliğini eklenebilir.

MyMessage.setIntProperty(“siparisFiyati”,3000);

Bu özellikler, mesajı tüketici tarafından seçici (selector) olarak kullanabilir.

Mesaj seçicileri, belirli bir özeliğe sahip olan mesajlar alınmak istendiğinde kullanılırlar. Belirlenen özelliğe göre inceleme yapılır. Örneğin,

String selector = “siparisFiyati>2500” ;

TopicSubriber mySubcriber = session.createSubcriber(topic,selector,false);

4.4.6.3 Mesaj Gövdesi

JMS API Tablo 4.2 ‘ de gösterilen altı farklı mesaj formatına sahiptir. Mesaj gövdesi iletilmek istenen bilgiyi içerir.

Çizelge 4.2 : JMS mesaj gövde tipleri.

Mesaj Tipi	Gövde İçeriği
TextMessage	Java lang.String nesnesidir.(XML tabanlı iletiler için uygundur.)
MapMessage	Kimlik-değer ikililerini taşır.
ByteMessage	Bu tür mesajlar bir sekizi dizisidir. Var olan meajtürlerini sarmak için kullanılabilirnir.
StreamMessage	Basit türlerin bir akımı, dosya, ağ gibi yavaşyavaş okunabilen belli bir kaynaktan zamanla gelecek verilerle çalışmaya izin verirler.
ObjectMessage	Serileştirilebilir(Serializable) java nesnelerini içerir.
Message	Boş mesaj. Sadece başlık alanları ve özelliklerden oluşur. Eğer mesaj bir gövde gerektirmiyorsa bu mesaj tipi kullanıma uygundur.

Hangi tür ileti kullanılacağı uygulama gereksinimleri tarafından belirlenir.

5. YAYINLA/ABONE UYGULAMASI GELİŞTİRMENİN SORUNLARI

Yazılım dünyasında birçok yayınla/abone veya mesajlaşma sistemi bulunmaktadır: Glassfish JMS[6], Redhat JBoss MQ, Oracle JMS, Microsoft Message Queueing (MSMQ). Birçoğu abonelik, asenkron mesajlaşma gibi temel servisleri geliştiricilere sağlamaktadır. Ancak gerçek dünyada kullanılabilir uygulamalar geliştirilebilmesi için herhangi bir mesajlaşma sisteminin güvenlik(security), kalıcılık(persistence), hiyerarşik yayınlama ve nitelik tabanlı erişim gibi doğrudan mesajlaşmanın kendisiyle ilgisi olmayan ancak herhangi bir mesajlaşma uygulamasının içermesi gereken servisler sağlanması gerekmektedir.

5.1 Kurumsal Doğrulama Ve Yetkilendirme Sistemlerinin Karmaşıklığı

Java Kurumsal Sürüm (Enterprise Edition) gibi sistemlerdeki güvenlik yapıları son derece karmaşıktır ve basit bir doğrulama (authentication) ve yetkilendirme (authorization) için kullanılmaları uygun değildir. Mesajlaşma sistemleri için gömülü bir güvenlik mekanizması olmadığı için geliştiriciler güvenlik yazılımlarını ya karmaşık altyapılar üzerine kurmaya veya sıfırdan kendileri geliştirmeye çalışmaktadırlar. Yayınla/abone sistemlerinin güvenlik ihtiyaçları çok da karmaşık değildir ama her uygulama için yeniden geliştirilmesini gerektirecek bir durum arzetmemektedirler. Kullanıcı tanımı içeren basit bir kullanıcı adı-şifre tabanlı doğrulama, yetkilendirme için Yönetici, Üretici ve Tüketici rolleri yeterlidir.

5.2 Mesajlaşma Sistemlerinde Kalıcılık Desteği Eksikliği

JMS gibi mesajlaşma sistemlerinde kalıcılık desteği bulunmamaktadır. Dolayısıyla tüm mesajlar belli bir süre sonra kaybolmak durumundadır. Bu geçicilik durumundan kurtulmak için veritabanı üzerinde çalışan bir mesajlaşma sistemi kurmak olanaklıdır ancak o durumda da mesajlaşmanın sağladığı asenkron olma özelliği gerçekleştirmek çok zordur. Birçok geliştirici kalıcı bir mesajlaşma sistemi kurmak için mesajlaşma tarafında JMS, kalıcılık tarafında JPA olan bir yapı geliştirmek durumunda kalmaktadır [5].

5.3 Hiyerarşik Ve Yol Tabanlı Teslimat ve Erişim Desteği

Standart yayınla/abone sistemleri hiyerarşik değildir. Genel bir konuya abone olup onun altındaki özel konulara gelen mesajları almanın yolu yoktur. Ya tüm mesajlar alınır ya da hiç alınmaz. Bazı mesajlaşma sistemleri alıcıların mesajları süzmesine olanak sağlayacak şekilde nitelik desteğine sahiptir ancak bu destek düz, yani hiyerarşik olmayan bir biçimdedir. Mesajlar tam nitelik değerleri ile veya joker karakterleri ile gönderilmelidir. Sadece belirli bir kategorideki mesajları almaya olanak sağlamazlar çünkü nitelikler arasında alt üst ilişkisi bulunmaz.

5.4 Çalışma Zamanında Konu Oluşturma Desteği Eksikliği

JMS uygulamalar arasında mesaj alış verişi temel işlevleri sağlamaktadır fakat JMS yapısında Konular arasında hiyerarşik bir yapı sağlanmamaktadır. Yani yaratılan konular arasında hiçbir ilişki kurulmaz. Ayrıca istemcilerin bazıları birçok konu içinden özel bir başlık ile ilgilene bilir. Bu durumda kullanıcı bütün konulara üye olmak zorundadır. Mesaj içerisinde ilgili başlığı seçilebilir fakat istemciye ulaşan bilgi tümüdür. Bu hem ağ trafiğini artırmakta hem de aboneye ek yük getirmektedir.

5.5 Geleneksel Mesajlaşma Sistemlerindeki Performans Sorunu

Mesajlaşma sistemlerinde bir alıcının mesajları alabilmesi için bir konuya abone olması gerekmektedir. Abonelik yoksa mesajlar alıcıya ulaşmaz. Ancak konuların sabit olması nedeniyle bir alıcının bir konuya abone olması durumunda o konudaki tüm mesajlar ilgisiz de olsa alıcılara ulaşır. Başka bir deyişle her kullanıcı önceden belirlenmiş konulara ya abone olur tüm mesajları alır ya da abone olmaz ve hiçbir mesajı almaz. Bu durum bir konudaki bazı mesajlarla ilgilenen alıcıların çok sayıda konuya abone olmalarına, istemeseler de tüm mesajları almalarına yol açmaktadır. Bu da mesaj trafiğini son derece artırdığı için ağ açısından sorun oluşturmaktadır.

6. ÇÖZÜM İÇİN YAPILAN ÇALIŞMA

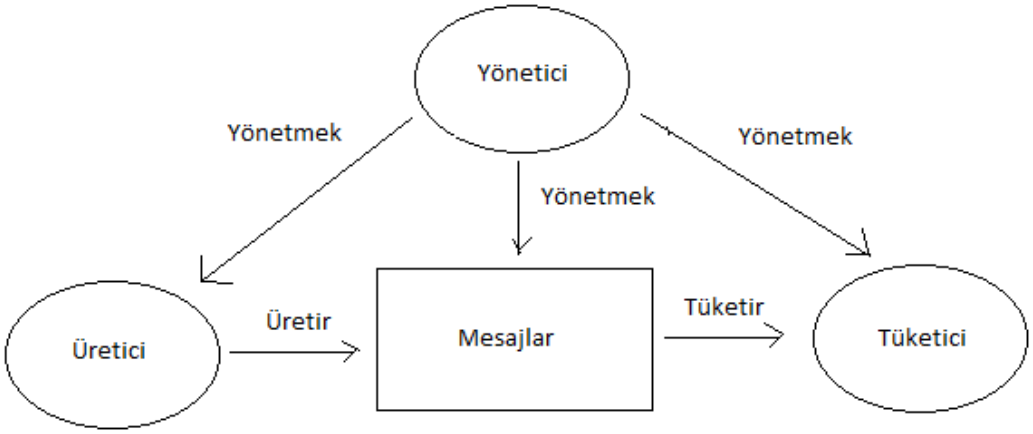
Burada sunduğumuz yapı ile istemci tüm bilginin gitmesi değil sadece istediği bilginin yollanılması sağlanılmıştır. Ayrıca yine topicler arasında hiyerarşi yapısı sağlanarak abone en üst konuya abone olunca onun çocuklarından gelen mesajlarında otomatik olarak aboneye gitmesi sağlanmaktadır. Böylece abone tek tek her konuya abone olma zorunluluğu ortadan kalmaktadır.

6.1 Doğrulama & Yetkilendirme

Bu uygulamada temel üç rol bulunmaktadır: Yönetici, üretici, ve tüketici bulunmaktadır. Tüm kullanıcılar yönetici rolüne sahip kullanıcılar tarafından oluşturulur ve kullanıcıların rolleri yine yöneticiler tarafından belirlenir. Bir yönetici başka bir kullanıcıya yönetici rolü atayabilir. Dolayısıyla sistemde ilk olarak tek bir yönetici kullanıcı bulunur. Diğer yöneticileri bu kullanıcı atar.

Yönetici rolü atanmış kullanıcılar başka kullanıcıları oluşturur ve onların Üretici ve Tüketici rölü verebilir. Yöneticilerin temel rolü konu açmak veya mesaj göndermek değildir. Ancak gerektiği durumlarda, örneğin bir kullanıcı bir mesajın yanlışlıkla gönderildiğini belirttiğinde ya da yöneticiye mesajın güvenlik v.s. nedenlerle sakıncalı bilgi içerdiği bildirilirse mesajları veya konuları silebilir.

Yönetici rolüne sahip kullanıcı, başka kullanıcıları oluşturabilir, silebilir ve her türlü düzenlemeyi yapabilir. Üretici rolüne sahip kullanıcılar konu açabilir ve konulara mesaj gönderebilirler. Ancak başka kullanıcıları oluşturma yetkisine sahip değildirler. Tüketici rolüne sahip kullanıcılar sadece mesaj gönderebilirler. Konu açamazlar ve kullanıcı oluşturamazlar.



Şekil 6.1:Yönetim.

Tüm kullanıcılar için sisteme giriş kullanıcı adı (username) ve parola (password) ile doğrulaması ile yapılır. Her kullanıcının sisteme girebilmesi için kullanıcı adı ve parolası olması gerekir. Giriş esnasında kullanıcının rolü bulunur ve o role ait sayfalar ve modüller kullanıcıya gösterilir. Dolayısıyla sisteme giren kullanıcı sadece kendi rolüne ait yetkileri kullanabilir.

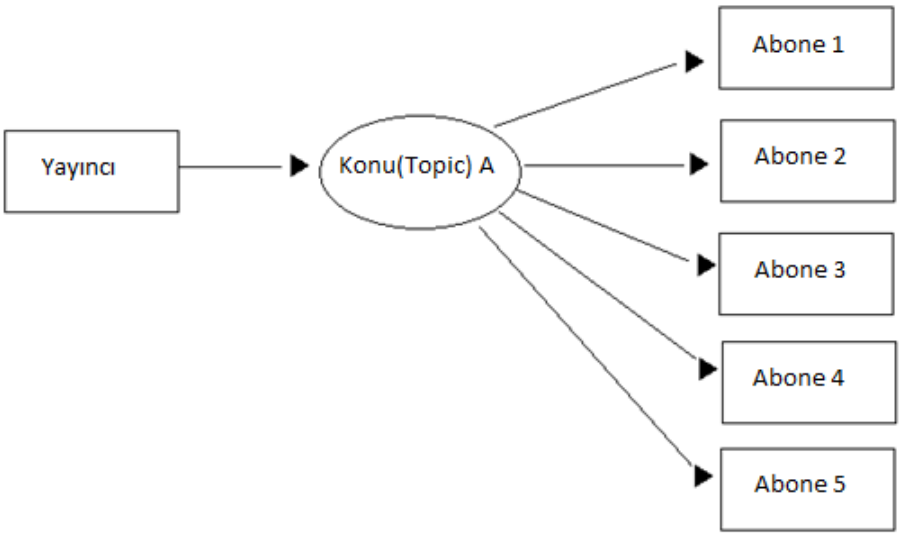
6.2 Roller

Sistemdeki roller ve yetkileri şöyledir:

- Tüketiciler : Sistemimizde tüketici rolüne sahip kullanıcılar mesaj gönderebilir.
- Üreticiler : Üretici rolüne sahip kullanıcılar da mesaj gönderebildikleri gibi konu açabilirler.
- Yöneticiler : Yöneticiler mesaj gönderme, konu açma dışında kullanıcı açma işlevleri bulunmaktadır.

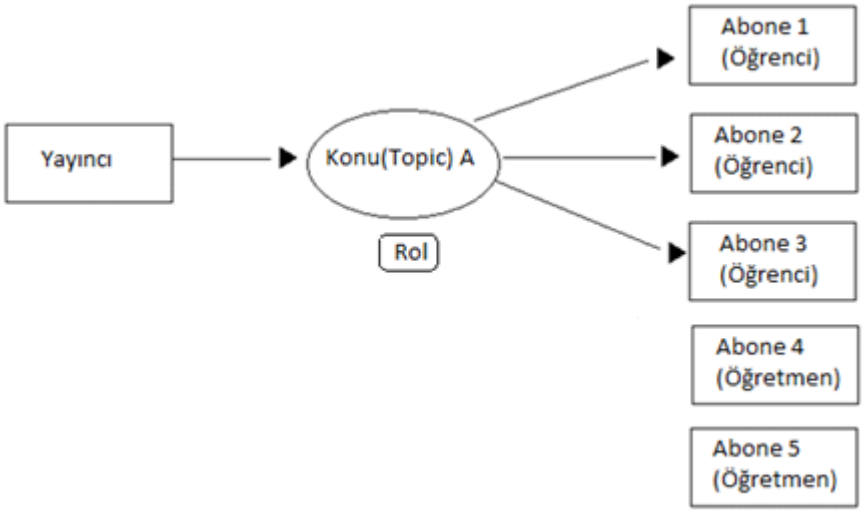
Roller, yetkilendirme dışında mesaj gönderiminde de kullanılmaktadır. Mesaj gönderilirken istenirse sadece belli bir role mesaj gönderilebilir. Bu şekilde tüketiciler veya üreticiler, yönetici ve üreticilere mesajların içeriği ile ilgili konularda, örneğin silinmesi veya değiştirilmesi talebinin mesajlar gönderebilirler. Bu şekilde mesaj gönderilirken belli bir rol hedeflenir ve gereksiz yere diğer rollere mesaj gitmesi engellenir.

Örneğin, sistemimizde A konusun abone olan 5 tane abone var. Bu abonelerden 3 öğrenci diğer 2 side öğretmen olsun. Bu durumda her hangi bir anda yayınlayıcı (Publisher) tarafında A konusuna bir mesaj geldiğinde bu konuya abone olan 5 abone yede mesaj gitmektedir. Yani hem öğrencilere hemde öğretmenlere mesaj iletilmektedir.



Şekil 6.2: Standart konu.

Fakat yapımızı aşağıdaki şekilde gibi düzenlersek yani A konusuna Roller alma özelliğini eklediğimiz düşünelim. Bu durumda Topiğe gelen mesaj abonelerden sadece 3 abone gitmektedir.



Şekil 6.3: Rol özelliğine sahip konu.

Bu durumda ağ trafiğinde 2/5 oranında azalma olacaktır.

6.3 Kalıcılık (Persistence)

JMS mesajların kalıcılığını sağlamaz. Yani bir mesaj gelir ve silinir. Burada mesajın kalıcını sağlamak için JPA (Java Persistence Architectue) teknolojisi kullanıldı. JPA ile Java nesnelerinin veritabanına SQL yazmadan saklanabilinmesi ve geri alınabilinmesi sağlanıldı. JMS sistemleri mesajlara kalıcılık sağlamamakta, bu şekilde gidip gelen mesajların alıcı tarafından okunduktan sonra kaybolmasına yol açmaktadır. Mesajların kalıcılığının sağlanması, gönderici ve alıcılar dışında sistemdeki mesajları inceleyen, bu mesajlaşmanın özetini çıkarmak isteyen veya kullanıcıların hangi konularda ne tür mesajlar gönderdiğini analiz etmek isteyen yöneticiler için faydalı olmaktadır. Bunun dışında en çok hangi konuda mesaj var, hangi kullanıcı daha çok mesaj göndermiş gibi istatistikî bilgilere de erişilebilir.

Sistem var olan her varlık için denetleyiciler (controllers) yaratıldı. Denetleyicileri kullanarak veritabanına yazma, okuma, silme ve güncelleme işlemleri yapıldı.

Şekil 6.4: Mesajların saklanması.

6.4 Yol (Path)

Her konu'nun içinde yer aldığı bir yapı vardır. (Dosya path veya XML içindeki xpath gibi) Örneğin

Lessons/Mathematic/Geometry/Triangles path nin anlamı Triangles nodu Geometry konusunu altındır, Geometry konusuda Mathematic konusun altındır ve son olarak bu grupun en üste de Lessons konusunu bulunmaktadır. Herhangi bir konuya mesaj yollandığında bu konu'nun altında yer alan diğer konu abonelerine otomatik olarak mesaj gitmektedir. Mathematic konusuna gönderilen mesaj hem Geometry konusuna hemde onun çocuklarına iletilmektedir.

Konu hiyerarşisine konu eklemek veya silmek üretici rolüne sahip kullanıcıların görevi ve hakkıdır. (Hata durumlarında yönetici müdahale edebilir). Hiyerarşinin nasıl kurulacağı, konuların nasıl belirleneceği belli bir üretici kullanıcı tarafından değil hepsinin ortak çabası sonucu oluşur. Herhangi bir üretici bir konu açtıktan sonra başka bir üretici o konunun altında başka bir konu oluşturabilir. JMS sistemlerinde olmayan, konuların dinamik olarak belirlenebilmesi özelliği sağlanmaktadır.

Mesaj alındığında, ilk olarak mesajın path'i keşif edilir.

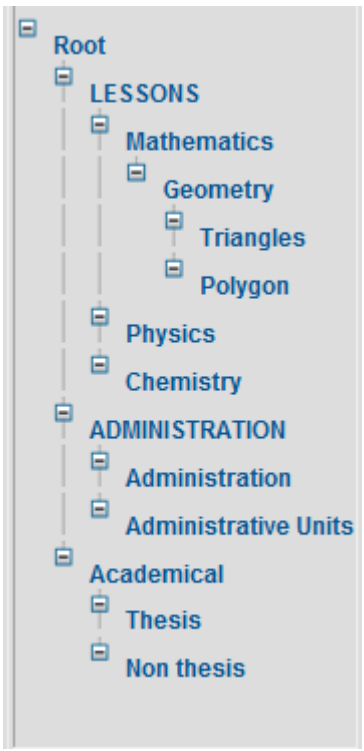
```
pathStart=pathStart+"%";
String string = "select s from Subject as s where s.subjectPath like :pathStart";
EntityManager em = getEntityManager();
Query q = em.createQuery(string);
q.setParameter("pathStart",pathStart);
List<Subject> list = q.getResultList();
```

Yukardaki kod ile konu ’nun tüm çocukları bulunur. Daha sonra aşağıdaki kod ile mesajı alacak olan kullanıcılar belirlenir.

```
String string = "select s from Subscription as s where s.subject.subjectId=:subjectId";

    EntityManager em = getEntityManager();

    Query q = em.createQuery(string);

    q.setParameter("subjectId", subjectId);

    List<Subscription> list = q.getResultList();
```

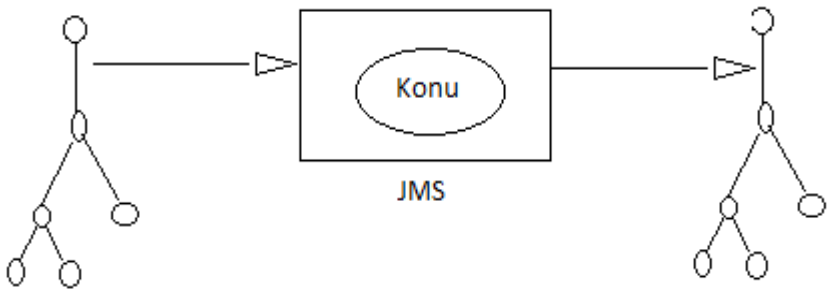
Son olarakta mesaj abonelere gönderilir.

Bazı durumlarda konu’lar arasındaki bu hiyerarşik yapı bizi birçok istemeyen durumdan kurtarır. Konu’lar arasında hiyerarşi olmaması durumunda: Örneğin bir öğrencinin hem Dersler hemde Matematik dersine abone olması durumunda sadece Dersler ve Matematik dersine gelen mesajları alabilir. Daha sonra açılan, Fizik dersinin açıldığından haberi olmadığından Fizik dersine abone olmamış. Olması durumunda, Fizik dersi ile alakalı hiçbir mesajı almaz. Oysa aşağıdaki şekildeki gibi hiyerarşinin oluşturulması sonucunda sadece Dersler konusuna abone olarak, Dersler altında yer alan tüm başlıklara ait mesajlara abone olmadan da alır.



Şekil 6.5: Hiyerarşik yapı.

Bizim yaptığımız yapı ile tek bir konuyu kullanarak sanal konular yaratıyoruz.



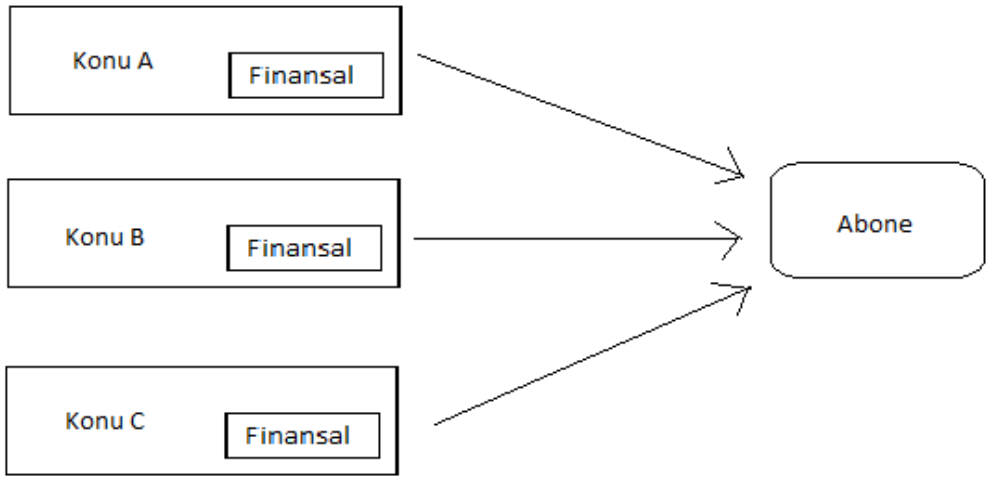
Şekil 6.6: Sanal konular.

6.5 Nitelikler (Attributes)

Gerçeklenen çalışma sonucu, istenirse sadece belirli bir niteliğe(attribute) abone olunabilinir. Şekil 6.2 gösterildiği gibi abone konu'lara değilde sadece Finansal niteliğe abone olduğunu gösterir. Bu durumda abone sadece finansal niteliğe sahip konu'lardan mesaj alabilir.

Kullanıcılar bir konuya abone olabildikleri gibi bir niteliğe abone olabilmektedirler. Bunun anlamı mesajın konusu ne olursa olsun içeriğini abone olunan niteliğe gelen mesajların alınmak istenmesidir.

Mesaj oluşturulurken kullanıcı mesajın niteliğini seçer. Mesaj seçilen konuya olduğu gibi seçilen niteliğe de gitmiş olur. Alıcılar arasında o konuya abone olanlar dışında o niteliğe abone olanlar da mesajı alır.



Şekil 6.7: Niteliğe abone olma.

Abone Konu A, Konu B ve Konu C abone olmadan bile Finansal özelliğe sahip olan bu konulara ait mesajları alabilir.

6.6 Perfomans Etkisi

Önerdiğimiz sistemde hiyerarşi ve ve nitelik yapısı ile birlikte istendiği kadar konu açma özelliği bulunduğundan mesaj trafiği son derece azalmaktadır. Bu durumu açıklamak gerekirse aşağıdaki gibi bir örnek durum düşünülebilir:

Konu (topic) sayısı: 1

Mesaj sayısı: 1000

Abone sayısı: 1000

olsun.

Abone ilgilensede ilgilenmesede ağ trafiği içinde 1000*1000 tane mesaj alış verişi yapılacaktır.

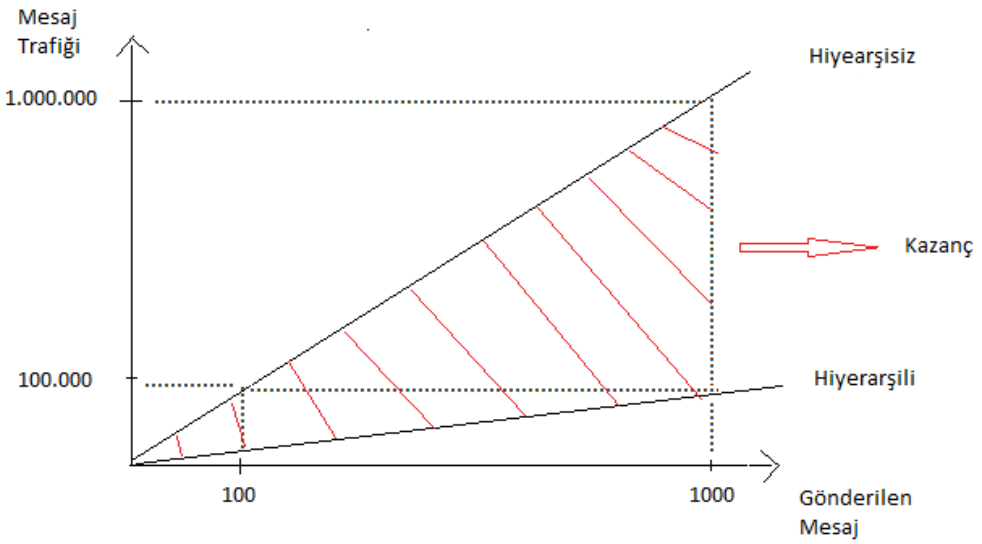
Konu sayısının: 10 olduğunu varsayalım:

1000 mesaj atıldığında 10 konu olduğundan her konuya 100 mesaj düşmektedir. 1000 kullanıcı 10 konuya abone olursa 1 konuya 100 abonelik düşmektedir. Her konuya 100 mesaj geldiğinden ve 100 aboneye gitmesi gerektiğinden dolayı her konu için toplam 100x100= 10.000 mesaj kopyası alıcılara gitmektedir. 1 konu için 10.00 mesaj gitmesi durumunda 10 konuya 100,00 mesaj gider. Başka bir deyişle

10 Konu x 100 Mesaj x 100 Abone= 100.000 Mesaj

etmektedir. Görüldüğü gibi giden mesaj sayısı 1.000.000 yerine 100.000 olmaktadır.

Sonuç olarak Şekil 6.3 de gösterildiği gibi %10 ağ trafiğin düşürülmesini sağlamış oluruz.



Şekil 6.8:Ağ trafik kazancı.

Ayrıca her konuya nitelik(attribute) eklersek:

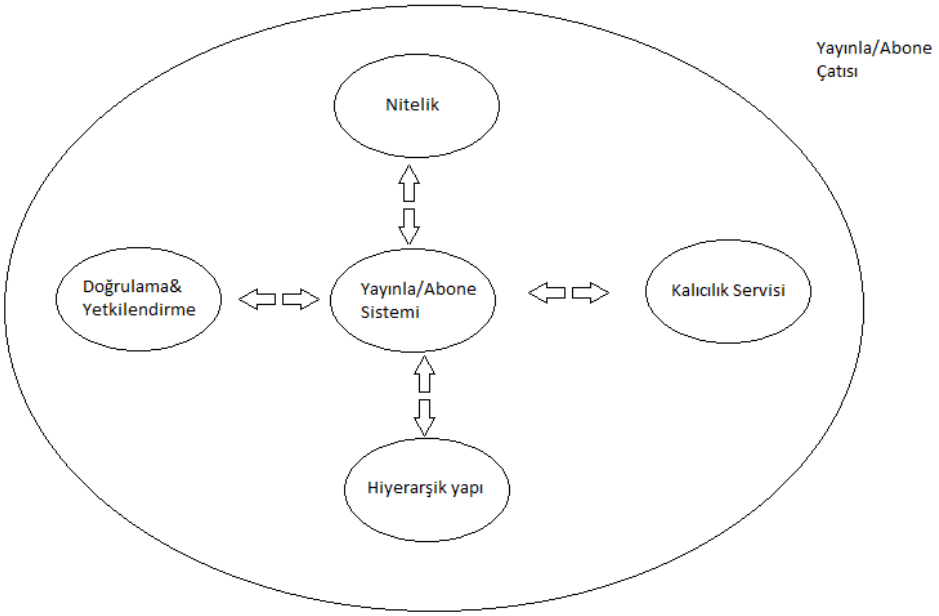
Nitelik(attribute) sayısı: 10

	Attr1	Att2	Attr3	Att4	Att5	Att6	Att7	Att8	Att9	Att10
Konu1	10									
Konu2	10									
Konu3	10									
Konu4	10									
Konu5	10	10	10	10	10	10	10	10	10	10
Konu6	10									
Konu7	10									
Konu8	10									
Konu9	10									
Konu10	10									

Şekil 6.9: Konu nitelik dağılımı.

7. UYGULAMA TANIMI

Standard Yayınla/Abone sistemlerine yetkilendirme, kalıcılık, hiyerarşi ve nitelik gibi özellikler eklenmesiyle oluşan bir yazılım çatısı (framework) geliştirdik. Bu çatıyı kullanarak bir E-Learning uygulaması yazıldı. Böylelikle herhangi bir uygulamanın geliştirilmesinde geliştirdiğimiz yazılım çatısının yazılım geliştiricilere ne gibi katkılar sağladığını uygulamalı olarak göstermiş olduk.



Şekil 7.1: Yayınla/Abone çatısı.

7.1 Amaç

Geliştirdiğimiz çatı ve E-Learning uygulaması temel olarak kullanıcılara mesajlaşma olanağı sağlamaktadır. Ancak bu mesajlaşma kalıcıdır, konulara göre mesaj gönderimine izin verir, nitelik tanımları yapılarak içeriğe göre filtreleme yapılabilir. Sistem kendi içerisine kullanıcı ve rolleri içermektedir. Kullanıcılar kullanıcı adı ve şifresine dayalı bir sistemle girişte doğrulanmaktadır. Her kullanıcı kendi rolüne uygun işlemleri yapabilmektedir.

Uygulamada bir doğrulama ve yetkilendirme modülü bulunmaktadır. Giriş yaptıktan sonra Yönetici, Üretici ve Tüketici olmasına göre ayı bir ekran kullanıcının önüne gelmektedir. Önüne açılan sayfalarda yetkilisine göre mesaj ekleme, konu açma ve kullanıcı oluşturma işlevlerini gerçekleştirmektedir.

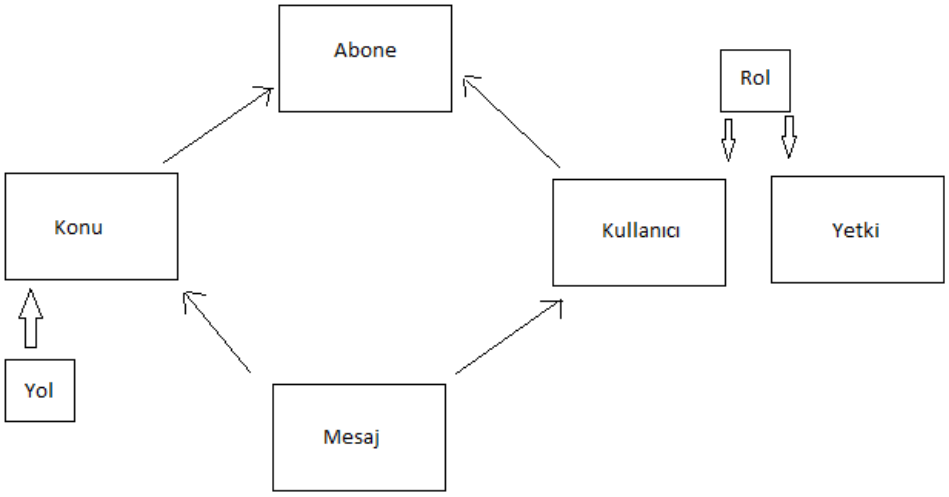
7.2 Geliştirme Ortamları

Programlama dili olarak Java, geliştirme ortamı olarak NetBeans kullanılmıştır. Uygulama Java Enterprise Edition (JEE) 6 standardına uygun olarak geliştirilmiştir. Uygulama Glassfish Application Server üzerinde çalıştırarak denendi. Ancak JEE 6 standardını destekleyen herhangi bir kurumsal sunucu, örneğin Redhat JBoss, Oracle WebLogic ve IBM WebSphere üzerinde çalıştırılabilir. Uygulama N-Tier (çok katmanlı) olarak tasarlandı. Grafik kullanıcı arayüzü olarak JSF 2.0 kullanıldı ve tüm web sayfaları MVC (Model View Controller) mimarisine uygun olarak geliştirildi. Tüm sayfalar XHTML standardına tam uygun olarak Facelet teknolojisiyle geliştirildi. Buna göre HTML içerisinde hiç Java kodu yazılmazken Java kodu içerisinde hiç HTML üretilmedi. Veritabanı tarafında JPA (Java Persistence Architecture) kullanıldı. Buna göre ORM (Object-Relational Mapping) yapısı kullanıldı ve Java nesneleri hiç SQL yazılmadan ilişkisel veritabanına yazıldı ve veritabanından okundu. Sorgular için SQL benzeri ancak nesne-yönelimli olan JPA-QL dili kullanıldı. Enterprise Java Bean (EJB) sistemine uygun olarak Message Driven Bean nesneleri yazıldı ve JMS mesajları bu nesne aracılığıyla okundu.

7.3 Uygulamada Kullanılan Varlıklar (Entities)

- User: Sistemin içindeki her kulllanıcıya bir rol atanır. (Yönetici, Öğretmen, Öğrenci, Çalışan.)
- Authorization: Kullanıcıların sahip olduğu haklar.
- Message: Yayıncı ve abone arasındaki iletişim sağlayan nesnedir.
- Subject: Mesajların gönderildiği hedeftir.
- Subscription: Kullanıcıların bir konudan mesaj almak için abone olması gerekmektedir.

•Attribute: Mesajların sahip olduğu nitelik.



Şekil 7.2: Varlık ilişkisi.

7.4 Sınıflar (Classes)

•ElearningMessageBean: Bu sınıfın bir Message Driven Bean dir. Java Message Service mesajları alır ve mesajı işler. Bu bean mesajı otomatik olarak alır.

•MessageBussiness: Bu sınıf mesajı alır ve hedef konuya gönderir. Mesajı sadece hedef konuya abone olmuş kullanıcılara değil bu konunun çocuklarına da iletilir. Yani hiyerarşide ağaçın alt çocuklarında gönderilir. Ayrıca mesajın içindeki niteliğe abone olan hedef kitleye da mesajı ulaştırma sorumluluğunda olan sınıftır.

•ElearningMessageProcuder: Bu sınıfın ana görevi mesajı konuya göndermektir.

•MessageObject: : Konulardan gönderilen ve alınan mesaj nesnesidir.

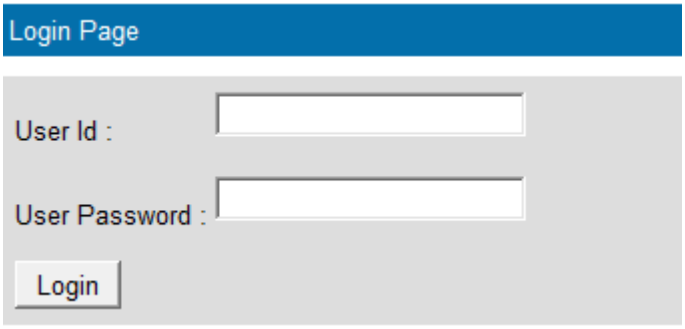
7.5 Sınıflar (Classes)

Uygulama dört adet kullanıcı arayüzüne destek vermektedir:

- Yönetici: Yöneticiler kullanıcı ekleyebilir veya silebilir. Bazı hatalarda olduğunda konuları veya mesajları güncelleyebilir.
- Öğretmen: Hiyerarşi üzerindekinde herhangi bir konuyu yaratıp veya silebilir. İlgili abonelere mesaj gönderip alabilir.
- Öğrenci: Öğrenciler mesaj alıp gönderebilirler. Fakat konu yaratıp ve silemezler.
- Çalışan: Mesajları gönderip alabilirler.

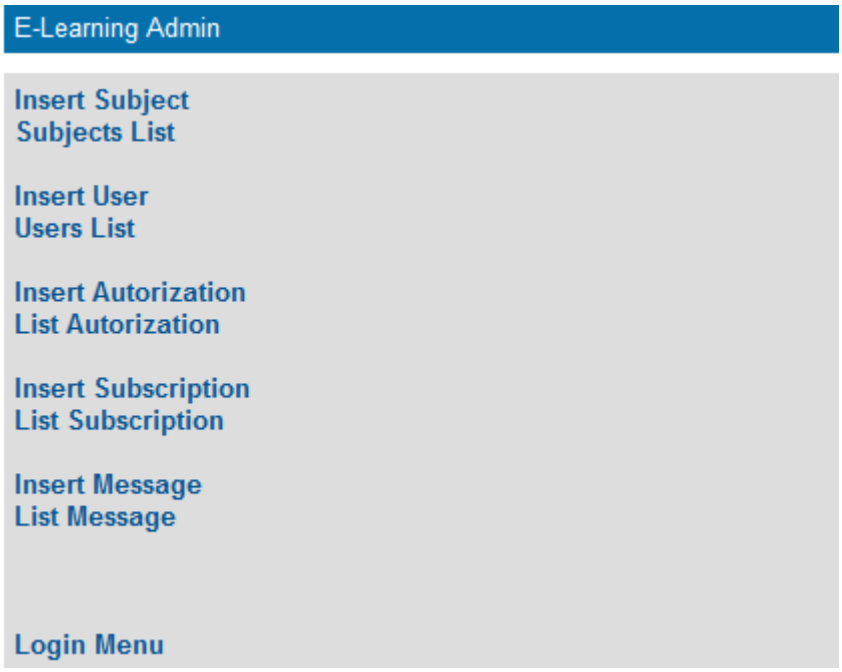
7.6 E-Öğretim (E-learning) Uygulama

Her kullanıcı, kullanıcı numarasını ve poralasını girerek sisteme girer. Burada kullanıcının rolune (yönetici, öğretmen, öğrenci, çalışan) göre ilgili sayfalar gelir.



Şekil 7.3: Giriş(Login) sayfası.

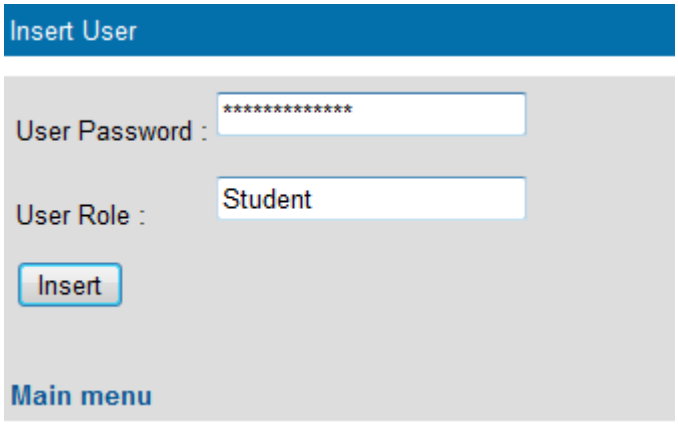
İlk olarak yönetici rolü ile sisteme bağlandığımızda, Şekil 7.2 görüldüğü gibi yöneticinin sayfası gelir.



Şekil 7.4:Yönetici sayfası.

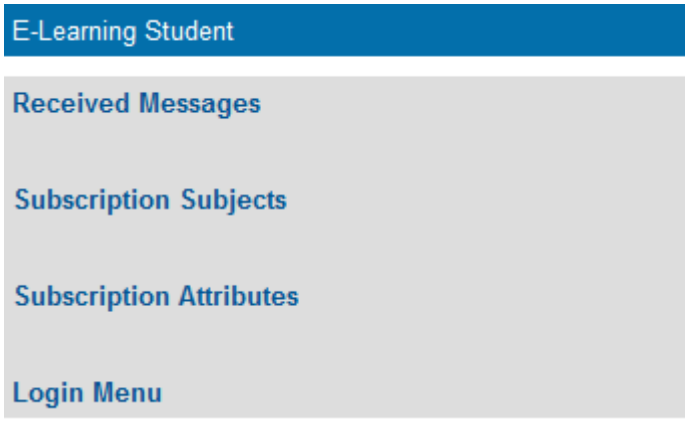
Yönetici rolünün temel amacı kullanıcı tanımlamak ve kullanıcılara rol vermektir. Bunların yanında konu yaratmak, ilgilendiği konulara abone olmak, mesaj göndermek ve silmek gibi işlemlerinde yapabilir.

Yönetici kullanıcı yaratmak için:



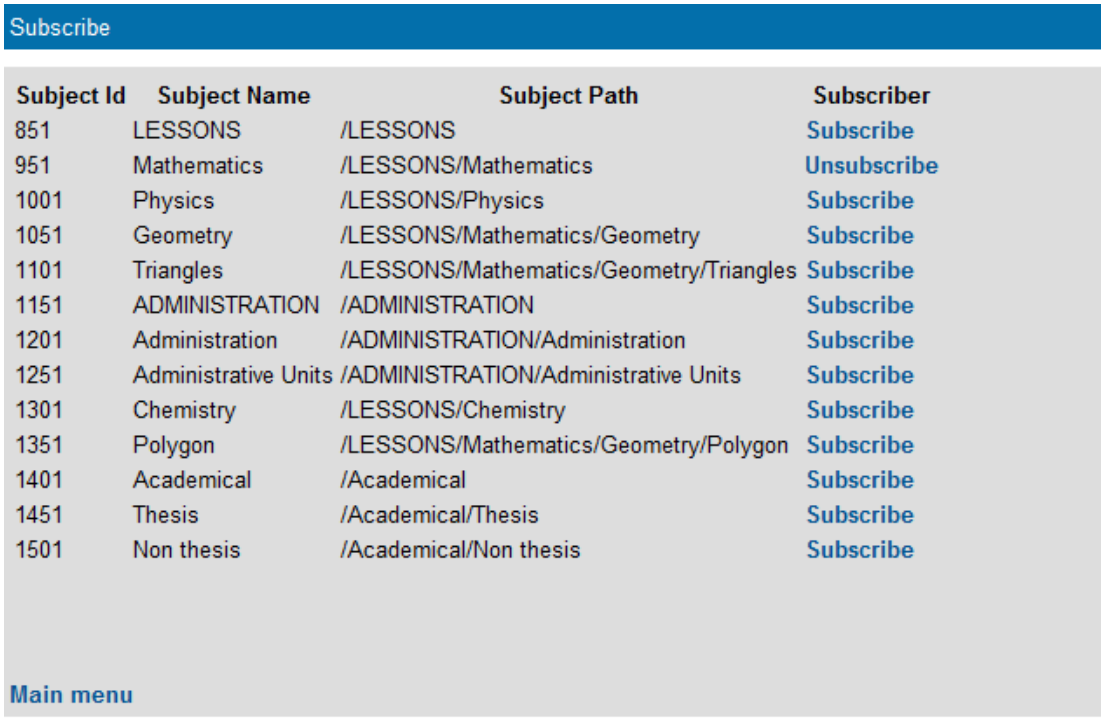
Şekil 7.5: Kullanıcı yaratma.

İkinci olarak öğrenci rolüne sahip bir kullanıcı ile sisteme bağlandığımızı düşünelim:



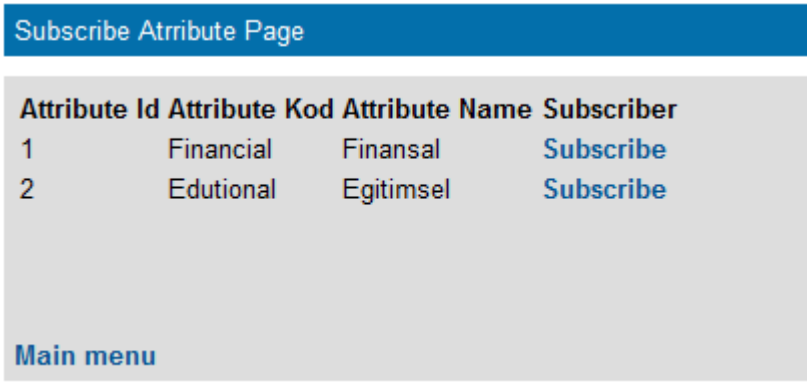
Şekil 7.6:Öğrenci sayfası.

Şekil 7.4 de görüldüğü gibi bir öğrenci mesajları alabilir, konulara veya niteliklere abone olabilir. Ancak konu veya nitelik yaratamaz. Aşağıdaki sayfada öğrenci abone olduğu veya olmadığı konuları görebilir ve ilgilendiği konuya abone olabilir. Öğrencinin sadece Matematik konusuna abone olduğu görülür. Yani bu durumda öğrenci Matematik ve onun altında bulunan konulardan mesaj alabilir.



Şekil 7.7:Öğrenci abonelik sayfası.

Öğrenci konulara değilde sadece niteliğede abone olabilir. Böylece istediği nitelikteki mesajları alması sağlanır.

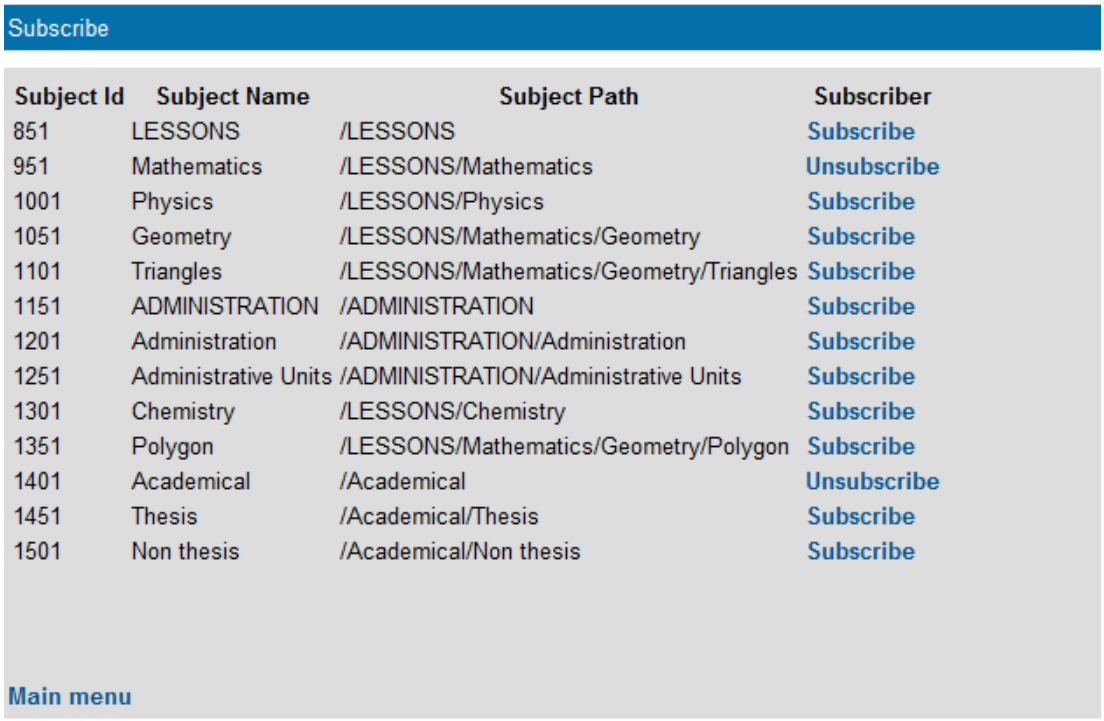


Şekil 7.8: Niteliklere abone olma.

Üçüncü olarak çalışan rolü ile sisteme bağlandığında öğrenci rolüne benzer şeklinde işlemleri yapabilecektir.

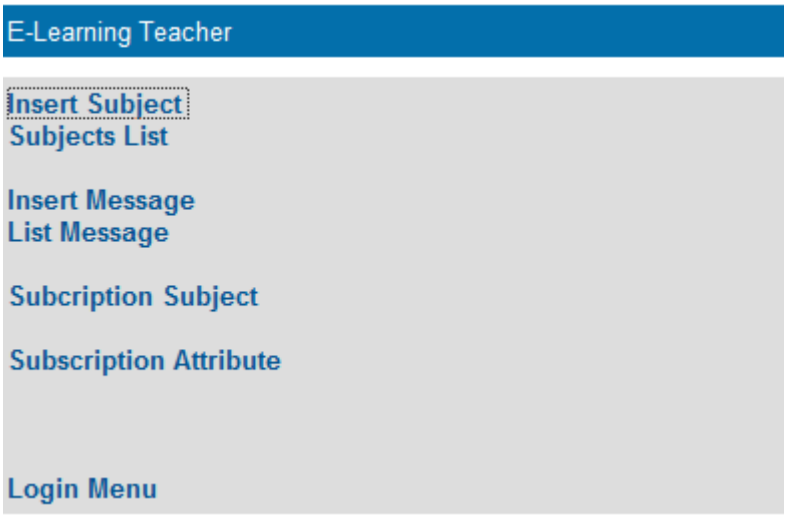
Şekil 7.9:Çalışan sayfası.

Şekil 7.8 de çalışan rolüne sahip kullanıcı Matematik ve Akademik konularına abone olduğu görülür.



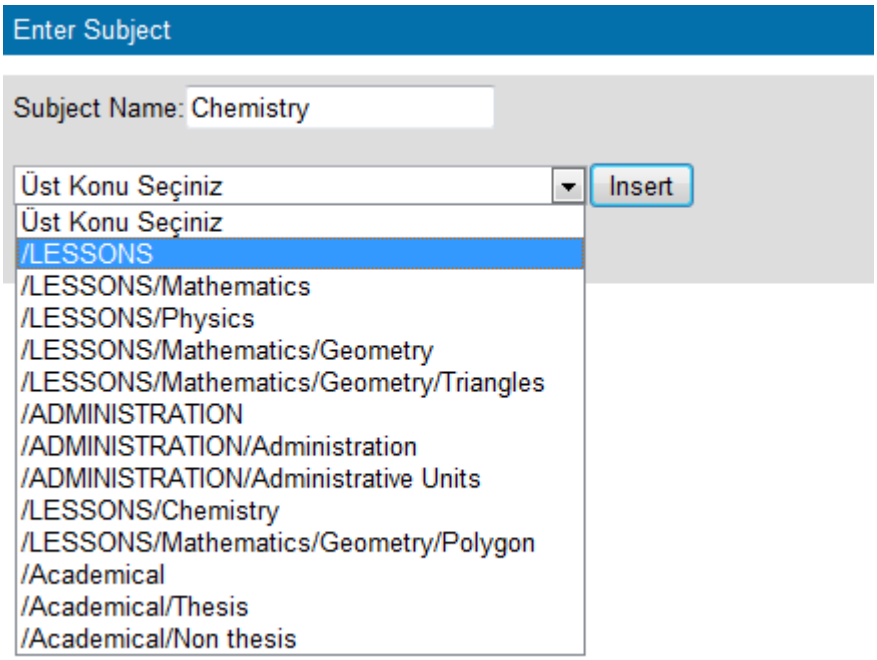
Şekil 7.10:Çalışan abonelik sayfası.

Son olarakta, öğretmen rolü ile bağlandığımızda aşağıdaki sayfa karşımıza gelecektir.



Şekil 7.11:Ögretmen sayfası.

Öğretmen kullanıcısı yeni bir konu açmak istediğinde Konu Ekle (Insert Subject) menüsü seçer.

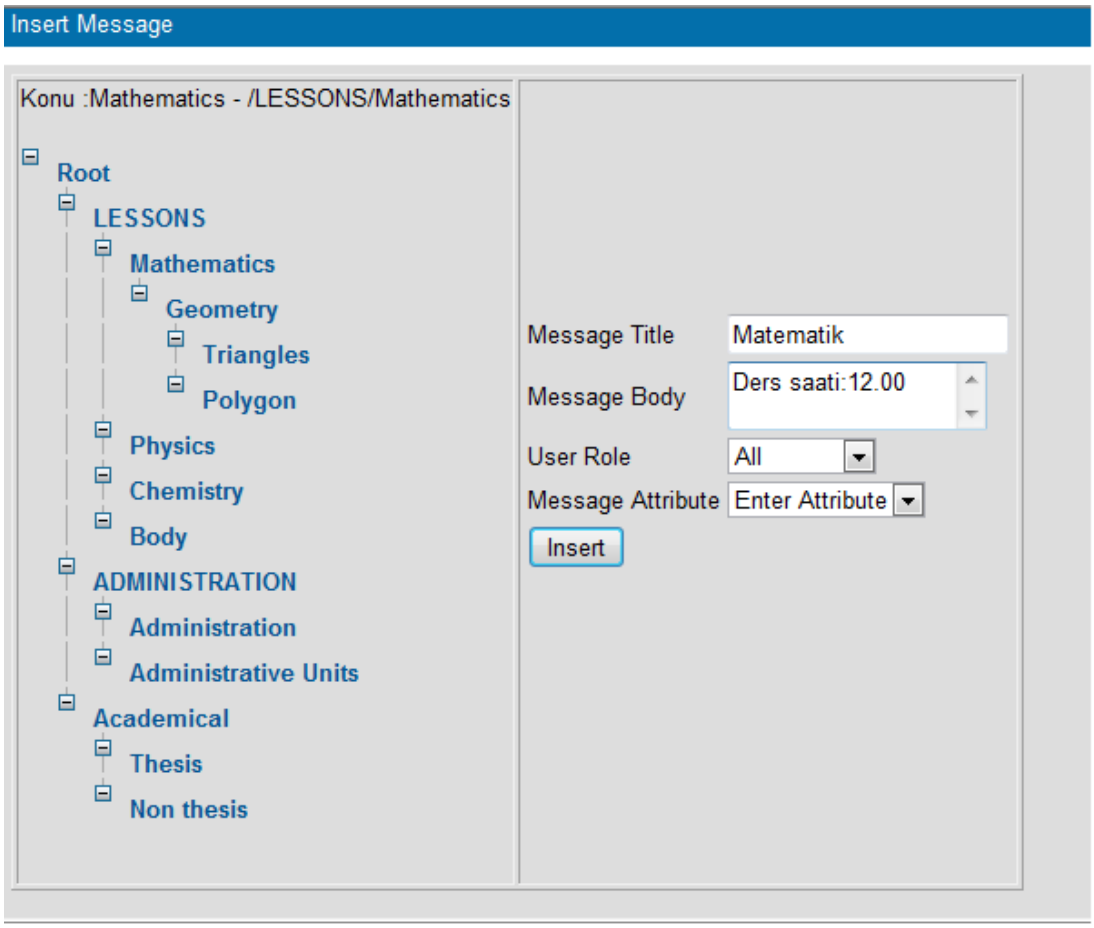


Şekil 7.12: Konuların girilmesi.

Yukarıdaki sayfada konunun adı ve hangi konunun üzerinde veya altında olduğu seçilip konu girişi yapılır. Sayfadaki örnekte, Kimya(Chemistry) konusu Dersler (Lessons) konusunun altında tanımlanmıştır.

Konulara arasında hiyerarşi ağaç tabanlı sisteme dayanmaktadır. Aşağıdaki şekilde de görüldüğü gibi ağacın en üstünde Kök dizini vardır. Her konu altındaki çocuk noduna bağlıdır.

Mesaj Ekle(Insert Message) menüsüne tıklandığında, mesajı girmek için aşağıdaki arayüz karşımıza çıkar.



Şekil 7.13: Mesajın girilmesi.

Bu ekrandan, öğretmen mesajın hangi konuya gideceğini seçtikten sonra, mesaj başlığını, mesaj içeriğini, mesajın gideceği rolü ve mesajın niteliğini belirleyebilir. Ekranda yazılan mesajı abonelerin alması için sadece konuya abone olmaları yetmez. Ayrıca öğrenci rolüne sahip olmaları gerekir. Yani mesaj, matematik veya dersler konularından herhangi birine abone olan ve öğrenci rolüne sahip olan aboneler iletilir.

Aşağıdaki dört durumun söz konusudur:

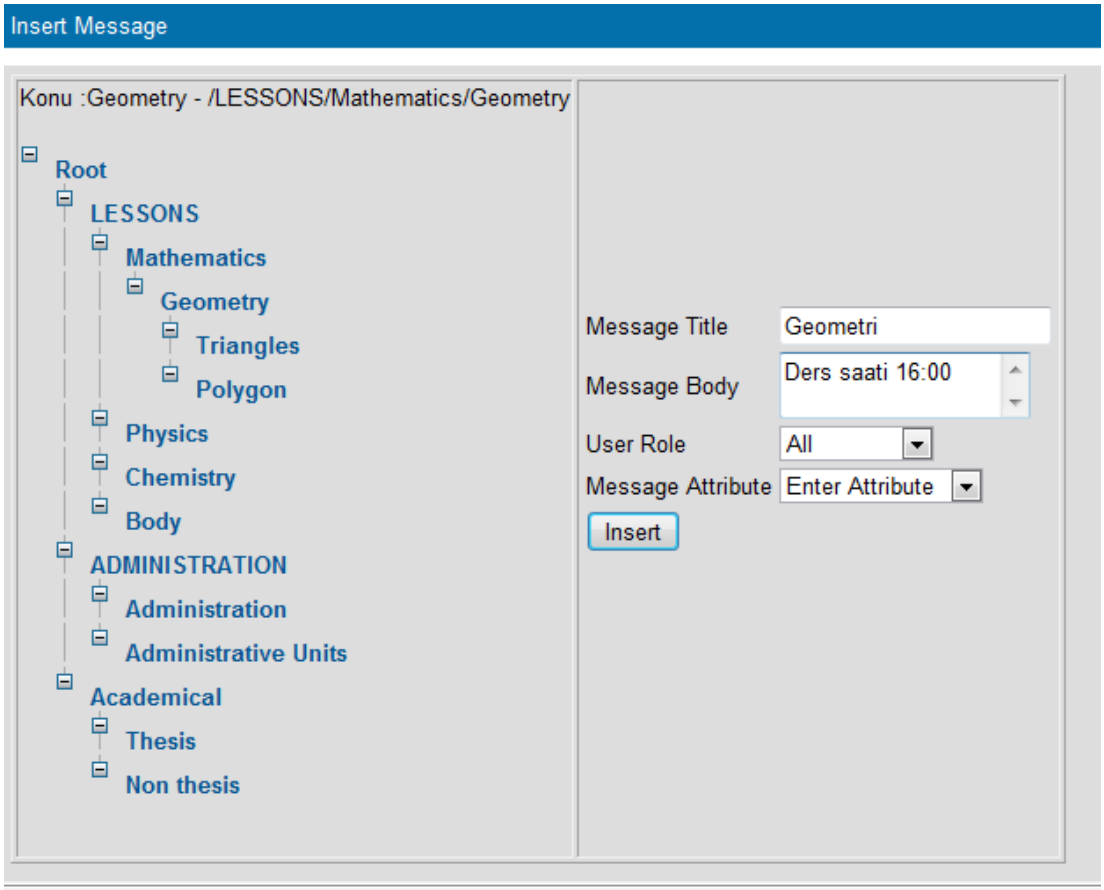
- Kullanıcı **alt** konuya abone olur, mesaj **alt** konuya gelir.
- Kullanıcı **üst** konuya abone olur, mesaj **alt** konuya gelir.
- Kullanıcı **alt** konuya abone olur, mesaj **üst** konuya gelir.
- Kullanıcı **üst** konua abone olur, mesaj **üst** konuya gelir.

Aşağıdaki çizgelde belirtilen roller ve abonelik durumlarının olduğunu düşünelim:

Çizelge 7.1: Rol ve abonelik durumları.

Kullanıcı	Roller	Abonelik Durumu
13	Öğretmen	-
14	Öğrenci	Matematik
15	Öğrenci	Fizik
16	Çalışan	Matematik

Öğretmenin Geometri konsuna mesaj gönderdiğini ve rol ayrımı yapmadığını düşünelim:

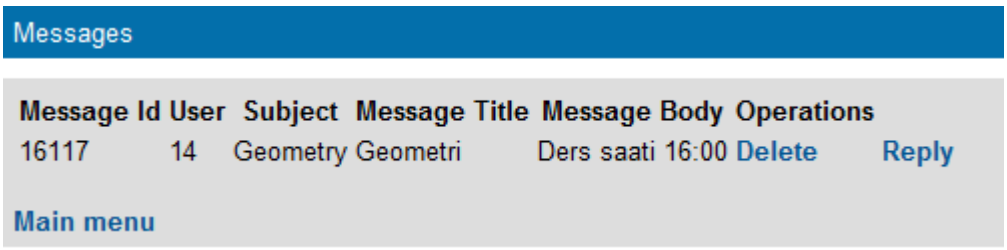


Şekil 7.14: Geometri ders mesajının gönderilmesi.

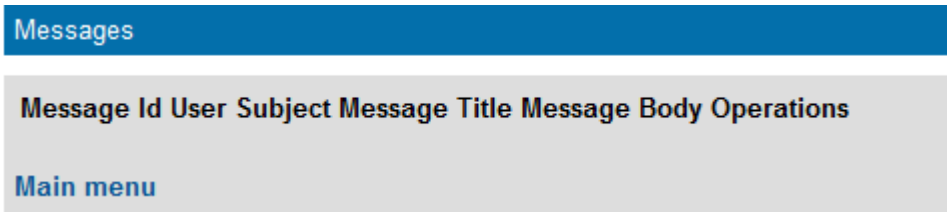
Bu durumda:

- Matematik dersine abone olan tüm rollere mesaj gitmeli.(Geometri dersi Matematik dersinin alt çocuğudur.)

- Fizik dersine abone olan öğrenciye gitmemelidir.

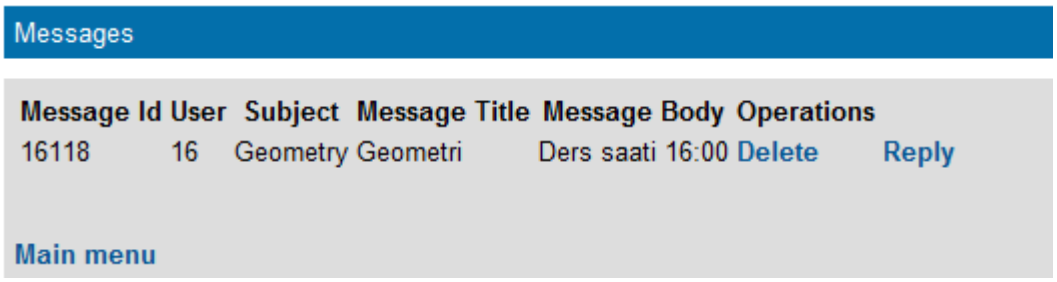


Şekil 7.15:14 numaralı öğrencinin aldığı mesaj.



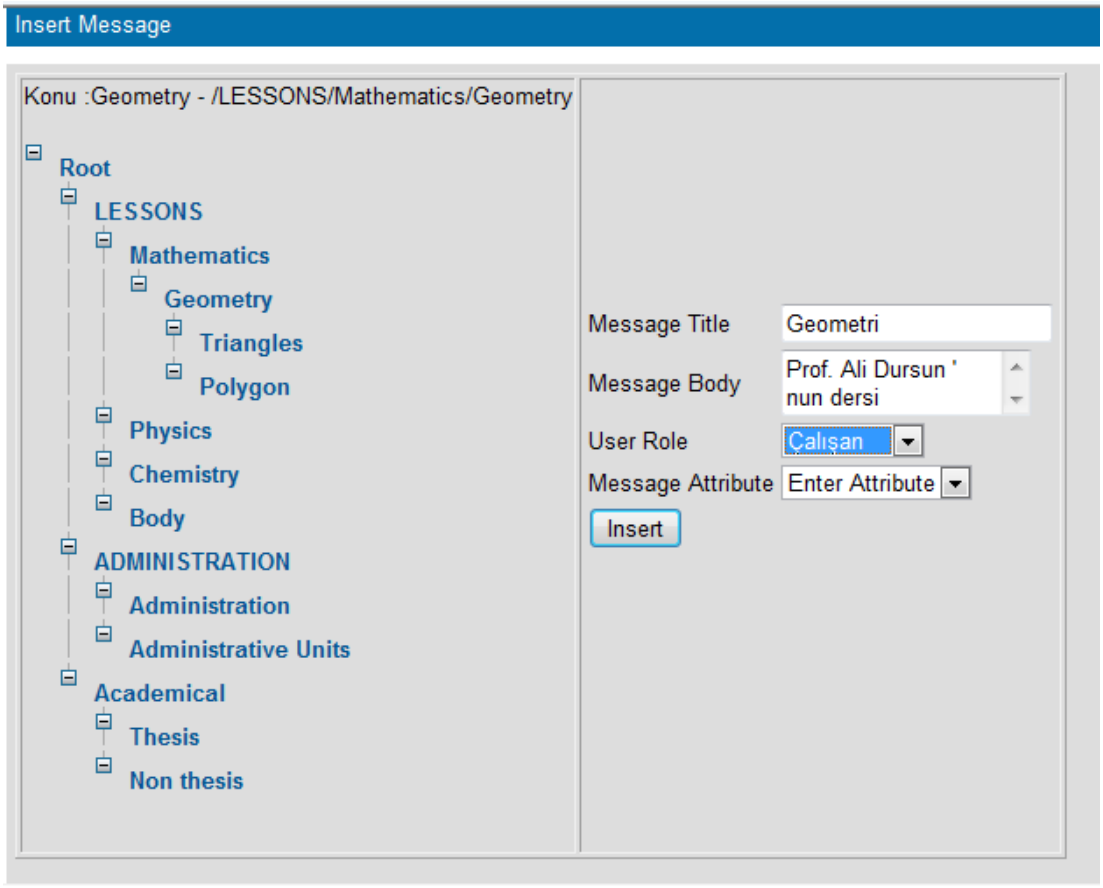
Şekil 7.16:15 numaralı öğrencinin aldığı mesaj.

Şekil 7.16 da görüldüğü gibi Fizik dersine abone olan öğrenci Geometri dersine gönderilen mesajı almamıştır.



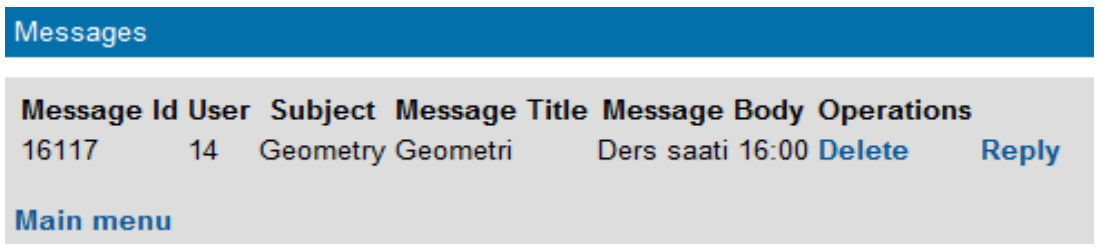
Şekil 7.17:16 numarılı çalışanın aldığı mesaj.

Öğretmenin rol seçimini yaptığını düşünelim:

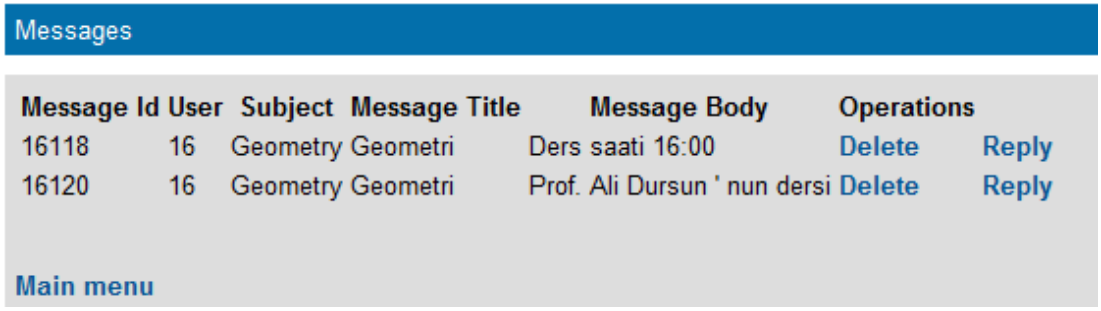


Şekil 7.18:Çalışan rolüne mesaj yollanması.

Bu durumda mesaj sadece Matemetik dersine abone olan çalışana gitmeli.

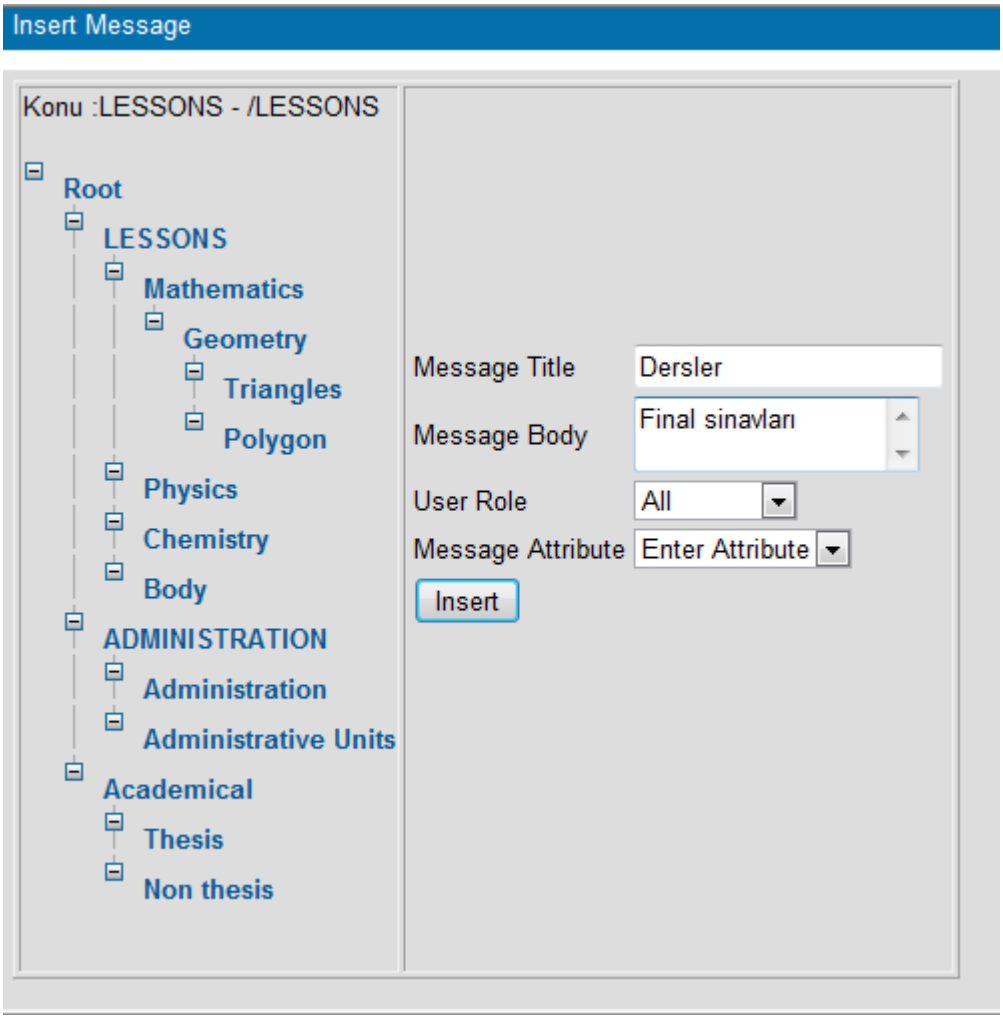


Şekil 7.19:14 numaralı öğrencinin mesajı almaması.



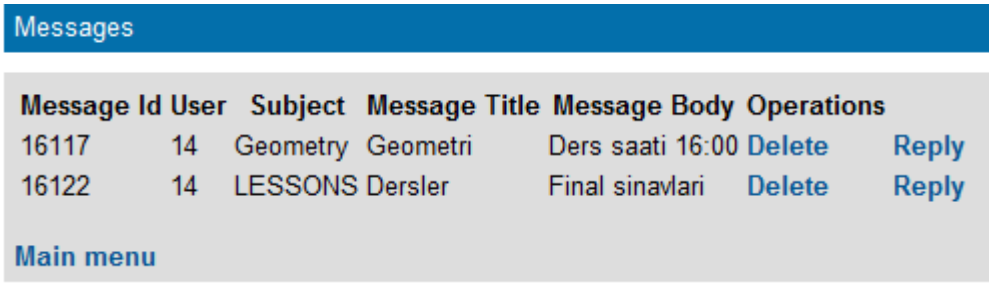
Şekil 7.20:16 numaralı çalışanın mesajı alması.

Üçüncü durum olarakta öğretmenin Dersler konusuna mesaj gönderdiğini düşünelim:

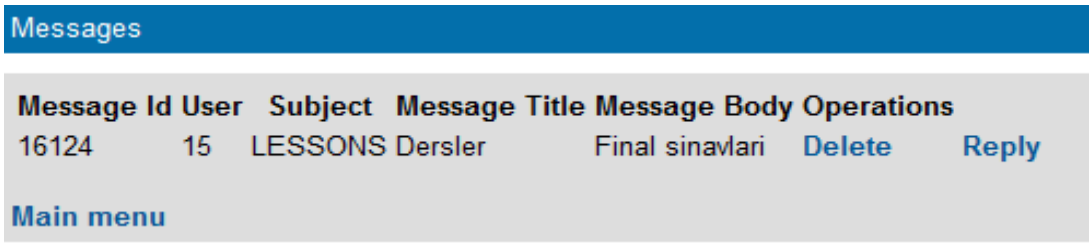


Şekil 7.21: Ders konusuna mesaj gönderilmesi.

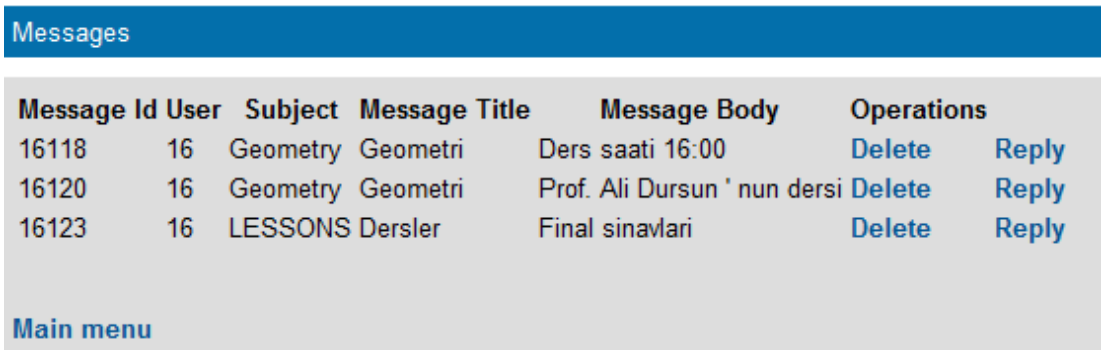
Bu durumda hiyerarşide desrlerin altında bulunan ve herhangi birine abone olmuş tüm abonelere mesaj gitmektedir.



Şekil 7.22:14 numaralı öğrencinin dersler konusundan mesaj alması.

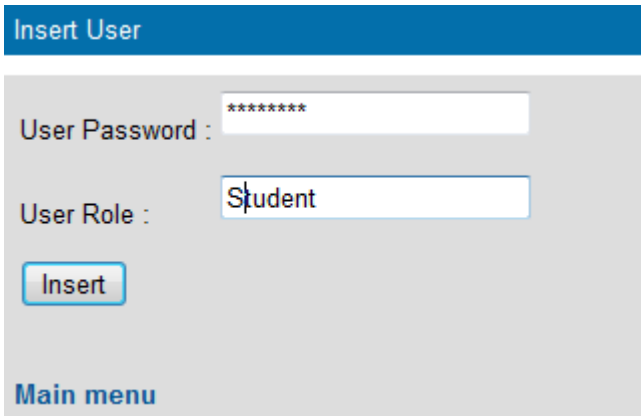


Şekil 7.23:15 numaralı öğrencinin dersler konusundan mesaj alması.

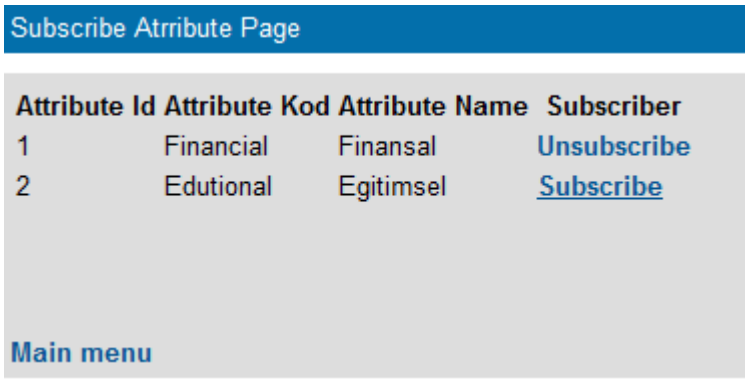


Şekil 7.24:16 numaralı çalışanın dersler konusundan mesaj alması.

Yeni bir kullanıcı yaratalım ve sadece Finansal niteliğe abone olduğunu düşünelim:

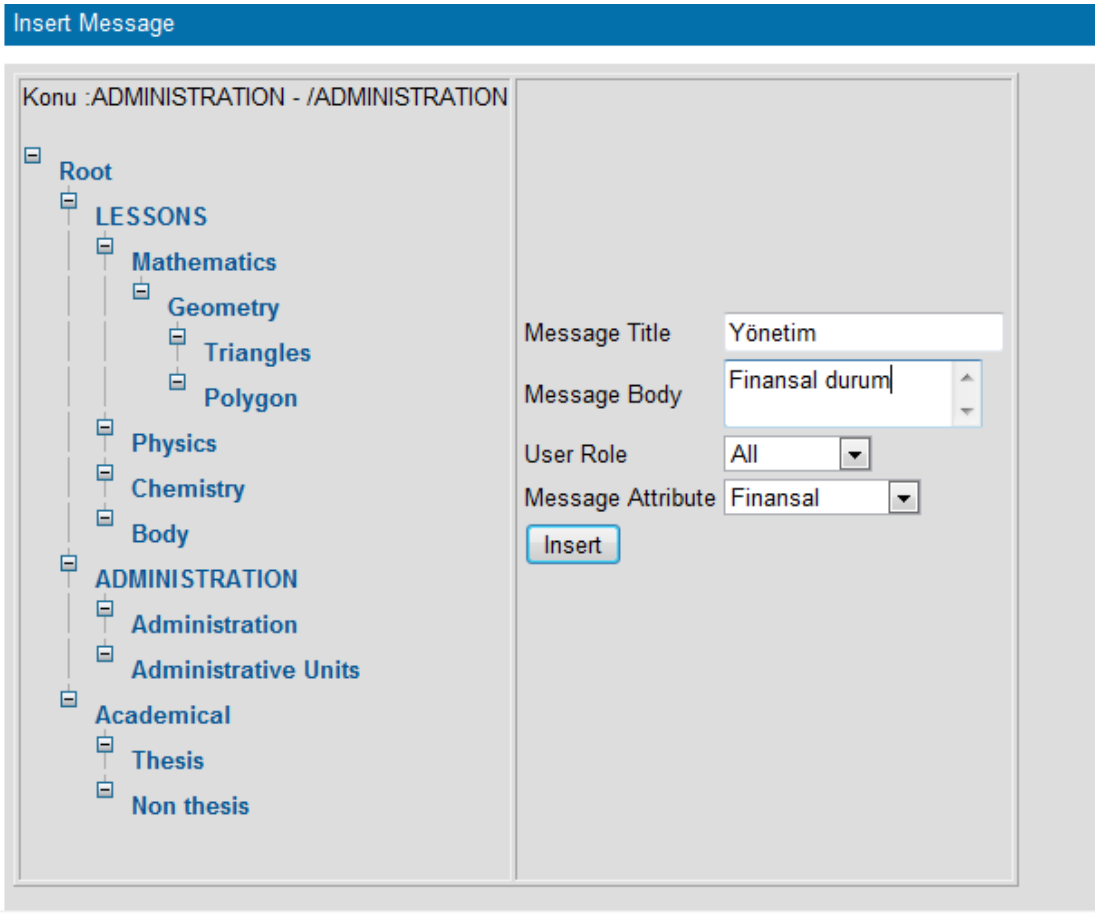


Şekil 7.25: Yeni bir öğrenci kullacısının yaratılması.

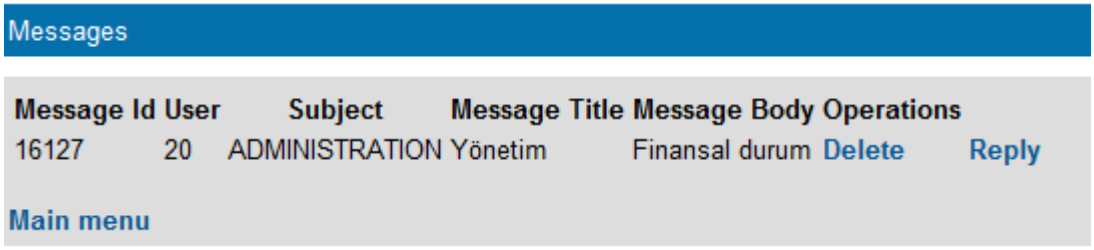


Şekil 7.26: Finansal niteliğe abone olmak.

Yeni yaratılan kullanıcı Finansal niteliğe sahip tüm mesajları alabilecektir.

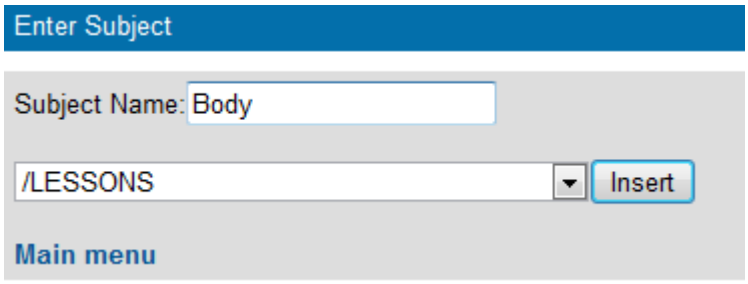


Şekil 7.27: Mesaj gönderiminde Finansal niteliğin seçilmesi.



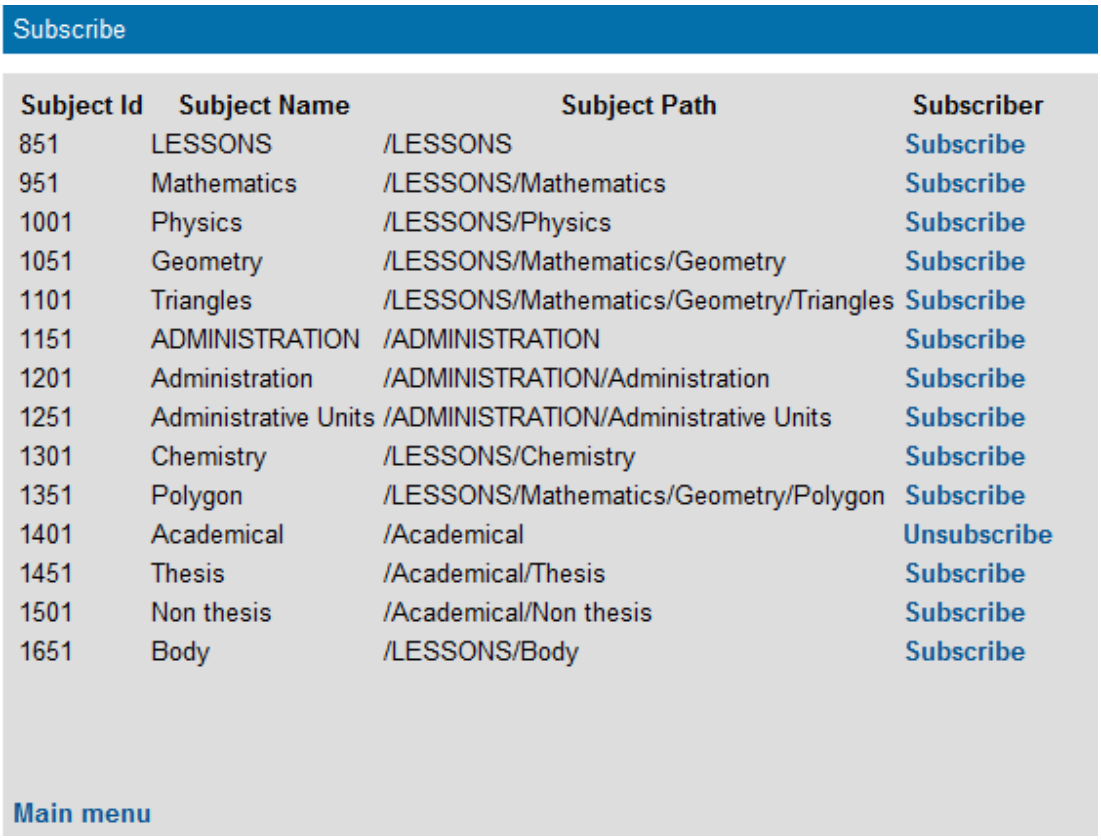
Şekil 7.28: Finansal niteliğe sahip mesaj alımı.

Öğretmen rolüne sahip kullanıcılar dinamik konu oluşturabilir.



Şekil 7.29: Dinamik konu oluşturma.

Çalışan rolündeki kullanıcıyı Matematik aboneliğinden çıkarmak :



Şekil 7.30: Matematik aboneliğinden çıkmak.

Çalışan rolüne sahip kullanıcının herhangi bir aboneliği kalmadığı için herhangi bir konudan mesaj almayacaktır.

8. SONUÇ VE ÖNERİLER

Tasarladığımız çatı (framework) ve bu çatı üzerinde geliştirdiğimiz örnek uygulama yayınla/abone sistemleri için hem çok faydalı hem de gerçekleştirilmesi çok az karmaşıktır. Geliştiricilerin üzerinden doğrulama ve yetkilendirme, kalıcılık (veritabanı işlemleri), hiyerarşik konu yapısı ve niteliklere göre içerikleri süzme gibi modülleri geliştirme yükünü almaktadır. Geleneksel Yayınla/Abone sistemlerin sağladıkları temel özellikler üzerinde yüksek düzeyli gelişmiş bir yazılım çatısı geliştirilmiş bulunmaktadır. Ayrıca standart mesajlaşma sistemlerinin karmaşık yapısı yerine son derece kullanımı basit bir yazılım altyapısı sağlanmış olmaktadır.

KAYNAKLAR

[1] **Richard Monson -Haefel, and David A. Chappel,** 2001: Java Message Service O'REILLY

[2] **Shrideep Pallickara, Geoffrey Fox and Harshawardhan Gadgil**, 2003. *On the Creation & Discovery of Topics in Distributed Publish/Subscribe Systems. Community Grid Labs, Indiana University.*

[3] **Venugopalan Ramasubramanian, Ryan Peterson, and Emin Gün Sier,** 2006. A High Performance Publish-Subscribe System for the World Wide Web. By Cornell University on April 20.

[4] **Patrick TH. Eugster, Pascal A. Felber, Rachid Guerraoiu and Anne-Marie Kermarrec,** 2003. The Many Faces of Publish/Subscribe. ACM Computing Surveys, Vol.35, No.2, June 2003.

[5] **Mike Ketin, and Merrick Schincariol,** 2006. Java Persistence API, PRO EJB 3

[6] **Michael Menth, Robert Henjes , Christian Zepfel, and Sebastian Gehrsitz**, 2006. Throughput Performance of Popular JMS Server. Debt. of Computer Science, Univertsity of Wuerzburg.

[7] **Gero Mühl, Andreas Ulbrich, Klaus Herrmann, Torben Weis,** 2004, Disseminating Information to mobile Clients Using Publish-Subscribe, IEEE

[8] **Url-1** <*http://support.sas.com*>, alındığı tarih 03.02.2011

[9] **JavaTM Message Service Tutorial, 2002 Sun Microsystems** alındığı tarih:03.02.2011,http://download.oracle.com/javaee/1.3/jms/tutorial/1_3_1-fcs/doc/basics.html#1023437

[10] **Anthony Tomastic, Charles Garrod, and Kris Popendorf**, 2006 On The Evalution of Symmetric Publish/Subscribe, Pittsburg, PA 15213, USA

[11] **Mark Happner, Rich Burridge and Rahul Sharma**. Sun Microsystems. Java Message Service Specifications. Alındığı tarih:03.01.201 http://java.sun.com/products/jms

ÖZGEÇMİŞ

Ad Soyad: Pınar Osanmaz Çelik

Doğum Yeri ve Tarihi: İstanbul 22.06.1981

Adres: Namık Kemal Mah. Elmas sok. No:6/3 Ümraniye/İstanbul

Lisans Üniversite: İstanbul Üniversitesi

Yayın Listesi:

- **Pınar Osanmaz Ç.**, **Nadia E.**, 2011: Hierarchical, Searchable, Secure and Persistent Publish/Subscribe Software Framework, EWG-DSS London-2011 WorkShop on Decision Systems.