İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

AGENT DESTEKLİ BİLGİ TOPLAMA SİTEMİ

YÜKSEK LİSANS TEZİ
Müh. Suat UĞURLU
(504971612)

Tezin Enstitüye Verildiği Tarih : 9 Şubat 2001
Tezin Savunulduğu Tarih : 6 Şubat 2001

Tez Danışmanı: **Doç.Dr. Nadia ERDOĞAN**
Diğer Jüri Üyeleri: **Doç.Dr. Coşkun SÖNMEZ(İ.T.Ü.)**
Doç.Dr. Oya KALIPSIZ(Y.T.Ü.)

Şubat 2001

ÖNSÖZ

Bu projede ağ ortamında haberleşmeye dayalı çoklu-agent yapısı ile destekli bir bilgi toplama sistemi gerçekleştirilmiştir. Çalışmamda benden yardımlarını esirgemeyen ve beni agent yapılı sistemler üzerinde bilgi edinmeye sevkeden sayın tez hocam Doç Dr. Nadia ERDOĞAN 'a teşekkürü borç bilirim.

ŞUBAT 2001 **Suat UĞURLU**

İÇİNDEKİLER

KISALTMALAR

CGI	: Common Gateway Interface
ASP	: Active Server Pages
JATLite	: Java Agent Template,Lite
TCP/IP	: Transfer Control Protocol,Internet Protocol
UDP	: User Datagram Protocol
KQML	: Knowledge Query and Manipulation Language
SMTP	: Simple Mail Transfer Protocol
POP3	: Post Office Protocol
HTTP	: Hyper Text Transfer Protocol
FTP	: File Transfer Protocol
AKF	: Agents Kernel Language
KIF	: Knowledge Interchange Format
JDK	: Java Development Kit
ASCII	: American Standard Code for Information Interchange
CORBA	: Common Object Request Broker Architecture
SQL	: Structured Query Language
DNS	: Domain Name Service
RMI	: Remote Methode Invocation
AMR	: Agent Message Router

TABLO LİSTESİ

ŞEKİL LİSTESİ

AGENT DESTEKLİ BİLGİ TOPLAMA SİSTEMİ

ÖZET

Bu proje bir dağıtık bilgi toplama sisteminin mimarisini ve uygulama ayrıntılarını incelemektedir. Benzer geleneksel bilgi toplama sistemlerine göre daha fazla esneklik sağlayabilmek için çoklu-agent mimarisi kullanılmıştır. JATLite agent oluşturulması ve agent haberleşmesi için gerekli arabirimi sağlamaktadır.

Öncelikle yazılım agentlarına kısaca bir giriş yapılmış ve özellikleri tanıtılmıştır.Ne türden yazılımları agent olarak adlandırdığımız ve uygulamaları çoklu agent olarak gerçekleyip mimarimizi buna göre kurmanın faydaları anlatılmıştır.

Daha sonra çoklu agent mimarisi kurabilmek için kullanılabilecek ve agent yazma ve bu agentları haberleştirme için arabirim sunan JATLite paketi incelenmiştir.Neden bu paketin kullanıldığı, faydaları, avantajları ve özellikleri ayrıntılı olarak anlatılmştır.

Son olarakta çoklu agent mimarisi üzerine kurulu bilgi toplama sistemi anlatılmıştır. Farklı görevdeki agentlar ayrıntıları ile incelenmiş ve sistemin işleyişi açıklanmıştır.

Agent Destekli Bilgi Toplama Sistemi; kullanıcı adına bilgi toplamak amacıyla çalışmaktadır ve çeşitli görevler için farklı agentlar görevlendirilmiştir. Sistemde Kullanıcı Agent,Yönlendirici Agent,Yedek Agent ve Kaynak Agent olarak dört farklı agent bulunmaktadır. Kullanıcı Agent,kullanıcıdan grafik arayüz ile aranacak bilgileri alır ve bu bilgileri Yönlendirici Agent'a yollar ve Kaynak Agentlardan gelen cevaplarıda yine aynı grafik arayüzü ile kullanıcıya gösterir.Yönlendirici Agent kullanıcı isteklerini hangi Kaynak Agentlara yönlendirmesi gerektiğini bilir ve sistemdeki agentlar hakkında en son güncel bilgilere her zaman sahiptir. Bu agent otonom çalışır ve akılldır. Sistemde tek olan Yönlendiri Agent'ın güvilirliğini Yedek Agent sağlar. Bir problem durumunda yeni bir Yönlendiri Agent yaratabilir. Veri tabanından kullanıcı isteklerini iste Kaynak Agentlar araştırır ve sonuçları doğrudan kullanıcıya yada verilen bir elektornik posta adresine yollarlar.

Sistem geleneksel bilgi toplama sistemlerine göre avantajlıdır. Dağıtık veri tabanı kullanma olanağı, düşük ağ band genişliğinde daha performanslı çalışma,güvensiz hatların telafisi,kullanıcı esneklikleri ve kullanıcı bilgi eksikliğinin giderilmesi Bilgi Toplama Sistemini geleneksel sistemlere göre olan başlıca üstünlükleridir.

AN AGENT BASED INFORMATION RETRIEVAL SYSTEM

SUMMARY

This project presents the design and implementation details of an information retrieval system. A multi-agent architecture has been adopted to allow for extended flexibilites over similar traditional systems. JATLite provides the basic infrastructure for creation of agents and their communication.

First,software agents are shortly described and their features are analized. It is also explained that what kind of softwares are called as agents and what are the utilization of a system that are consructed with multi agents.

Then,Jatlite, a software packet that is used for agent creation and communication ,is explained in detail.Its advantages,features and the reason for its usage are written.

Finally, the information retrieval system which has a multi-agent architecture is explained in detail. Agents with different tasks are described.

The Agent Based Information Retrieval System works to retrieve user requests from a computer network and are constructed with different types of agents for different tasks.The sistem consists of four type of agent named User Agent,Broker Agent,Backup Agent and Resource Agent.The User Agent takes user requests by a grafical interface from a user and passes them to the Broker Agent.It also shows the result by the same grafical interface. The Borker Agent knows to which resource agent it should redirect the user requests.It also knows the most recent and detailed information about other agents in the system. The Broker Agent is atonomous and intelligent. The Backup Agent provides the relability of the Broker Agent which is unique in the system and creates a new Broker Agent in case of a failure.Resource Agents search their database for the user requests and send the results to User Agents or to a suplied email address.

The system has advantages over traditional information retrieval systems.Usage of distributed database architecture,user flexibilities,compensation of lack of user knowledge and unreliable network connections ,high performance,low bandwith usage are some of the advantages that The Agent Based Information Retrieavl System has.

1. GİRİŞ

Her geçen gün internet üzerinde çalışan uygulamalar daha dikkat çekmeye başlıyor. Internetin bütün avantajlarından yararlanmak için yeni bir takım mimariler ve programlama teknikleri üretmek gerekiyor. Geleneksel dağıtık uygulamalar, etrafındaki ağ hakkında bilgisi olmayan ve belirli bir işi yapmak üzere tasarlanmış görevleri bir takım yerel kaynaklara atamaktan öteye gitmemektedir. Agent tabanlı programlama metodu çalıştıkları ağdan haberdar , birinin bozulmasına karşın diğerinin görevini üstlenebilecek agent adı verilen program parçaları ile uygulama geliştirmeyi tanımlar.

Şu an internette çeşitli bilgi toplama sistemleri kullanılmaktadır. Çoğu merkezi bir veri tabanı kullanır. Bir tarayıcı bir istek yolladığında sunucu tarafta bir CGI ya da ASP programı çalışır. Bu program veritabanından gerekli veriyi çeker ve istemciye yollar. Bu tip bir sistemin bir çok eksikliği agent tabanlı bir mimari kullanılarak kapatılabilir. Dağıtık bir veri tabanı kullanılarak elde edilen performans artışı ve kullanım olanağı, gelişmiş kullanıcı olanakları, düşük seviyede ağ trafiği bunlardan sadece bir kaçıdır. Bu projede , agent oluşturulması ve haberleşmesi için JATLite kullanılarak geliştirilmiş agent tabanlı bir bilgi toplama sisteminin mimari yapısı ve uygulama ayrıntıları açıklanacaktır.

2. YAZILIM AGENTLARI

2.1 Agent Nedir

Bütün özelliklerini içerebilecek ve çoğu araştırmacının söylediklerinin hepsini kapsayacak ve ne olup olmadıklarını açıkça anlatabilecek bir agent tanımı yapmak oldukça zor. Agentların birer program oldukları biliniyor,ancak ne tip özelliklere sahip programların agent olarak adlandırıldığı ayrıntılı olarak anlatılacaktır.Stan Franklin ve Art Graesser 'in " Is it an agent ,or just a program?"[2] adlı makaleleri bu konuyu oldukça açık bir şekilde anlatmaktadır.

Genel olarak agentlar aşağıdaki karakteristiklere sahiplerdir.[1]

- Otonom
- Uyum sağlayan /Öğrenen
- Hareketli
- Amaca yönelik
- Haberleşen/Birlikte çalışan
- Aktif/Durağan
- Esnek
- Sürekli

Agentlar ayrıca küçük yapıdadırlar. Tek başlarına bir uygulamanın tamamını oluşturmazlar. Bunun yerine diğer agentlarla birlikte çalışır. Kısaca, küçük ve sınırlı fonksiyonları olan ve yukarıdaki özelliklerin birçoğunu sağlayan uygulamalardır.

Bir agenttan beklenen şey, zamanımız olduğun da kendimizin yapabileceği şeylerin,nasıl yapılacağını bilmesidir.

Bir agent, görevini başarıyla yerine getirebilmesi için çevresinden edindiği bilgileri iyi bir şekilde kullanmasını bilmelidir.Çevresine uyum sağlayabilmelidir ki şartlar değiştiğin de hala istediğimiz sonuç elde edilebilsin. Yani agent uyum sağlayan ve öğrenen bir nitelikte olmalıdır.

Agentlar herhangi bir yerden müdahele olmadan kendi başlarına çalışabilen,hareketleri ve o anki durumları üzerinde kontrol yetenekleri olan uygulamalardır. Yani otonom olma özelliğine sahiptirler.

Haberleşme insan hayatında olduğu gibi yazılım dünyasında da oldukça önemlidir. Sözkonusu olan ister altyordamlar, ister threadler ister agentlar olsun bütün uygulamalar ellerindeki veriyi paylaşma ihtiyacı (haberleşme) duyarlar.

Agentlar ya arka planda yada aktif olarak ön planda sürekli çalışan uygulamalardır.Tek bir girişi alıp tek bir çıkış üreten sonrada sonlanan uygulamalar değillerdir.

Agentlar aynı zamanda bir ağ ortamında istediği bilgiyi diğer yazılımlara sormak yerine kendini ortamdan ortama taşıyarak bilgiye kendi ulaşabilirler.

Bir agent ,karmaşık bir görevi nasıl yerine getireceğini bilmelidir.Görevi en iyi şekilde nasıl küçük parçalara ayıracağına, bunları hangi sırada nasıl yapacağına,kendisi karar verir.

2.2 Agentlar Üzerinde Çalışma Nedenleri

Agentlar bilgisayar biliminin bir çok parçasını taşımaları nedeniyle ilgi çekici bazı özelliklere sahiplerdir. Nesneler ve dağıtık nesne mimarisi, adaptif öğrenme sistemleri,yapay zeka, uzman sistemler, genetik algoritmalar, dağıtık işleme,dağıtık algoritmalar, birlikte çalışabilme agentların iş görebileceği alanlar arasındadır.

Şu anki durumları ile henüz çok fazla popüler olmamakla beraber ileride kullanım alanlarının oldukça artacağı söylenebilir. Agentlar üzerindeki ticari ilgi de gitgide artmaktadır.

Agent mimarisi aynı zamanda günümüzde kullanıcıları oldukça yoran bazı problemlerin çözümünde de oldukça faydalıdırlar. Sadece onlara ne yapmaları gerektiğini söylüyoruz ve onlar bizim için işi sonlandırıp bizi bilgilendiriyorlar. Internette belli bir konuda bilgi arama buna verilebilecek güzel bir örnektir.

2.3 Agentların Çözdüğü Problemler

Farklı açılardan bakınca agentların çeşitli problemleri çözmemize yardımcı olduklarını görmekteyiz.

Taşınabilir agentlar , klasik istemci/sunucu ağ bant genişliği problemini çözerler. Ağ bant genişliği dağıtık bir ortamda oldukça önemli bir kaynaktır. Bir istemci ve bir sunucu arasındaki sorgu , ağ üzerinde gidip gelmesi gereken yüklü bir trafiğe neden olabilir. Bu da performansı düşük bir uygulamaya neden olacaktır. Bir agent oluşturup bunu sunucuya yollayarak yapılacak işi orada yerel olarak gerekleştirmek bant genişliği problemini çözecektir. Burada ağ üzerinden gönderilmesi gereken sadece agent'ın kendisidir.

Örnek bir durum şu şekilde verilebilir. Bir İstemci/Sunucu mimarisi tasarlanırken mimariyi tasarlayan kişi istemci ve sunucunun görevlerini belirler ve her parçayı değerlendirir. Bunların hepsini de geliştirme aşamasında yapar. Tasarlayan kişi , ağ trafiği, işlem hacmi, istemci ve sunucu sayısı ve diğer bir çok faktörü göz önünde bulundurarak mimarideki her parçanın görevlerini belirler ve ona göre geliştirir. Eğer bu tahminler yanlışsa yada tasarlayan kişi yanlış kararlar verirse, uygulamanın performansı oldukça düşecektir. Ne yazık ki uygulama geliştirildikten sonra bu tip problemleri çözmek yada uygulamayı değiştirmek oldukça zordur. Hareketli agentlar üzerine geliştirilen bir yapı bu türden problemlerin giderilmesi için daha uygundur. Geliştirme aşamasında mümkün olduğunca az karar verilir ve sistemi değiştirmek çok daha kolaydır. Adaptif ağlardaki yük dengelemeyi destekleyen agent mimarileri, gerekli değişikleri otomatik olarak gerçekleştirecektir.

Agentların çözdüğü diğer bir problem de sürekli bağlantıda olmayı gerektiren ağ alt yapısındaki bozulmalardan etkilenmedir. Ağ üzerinden çalışan çoğu program ağın sürekli ayakta olmasını gerektirir. Yarıda kalan bir sorgulamayı istemci yeniden başlatmak zorundadır. Ancak agent mimarisinde istemci bu işi yapmak üzere ağ çalışır durumda iken bir agent görevlendirir ve kendi bağlantısını keser. İşi yapan agent, istemci tekrar bağlantı kurduğunda onu bilgilendirir.

Agentların bizim yerimize düşünebilmesi için de uzun süredir çalışmalar devam etmektedir. Bu da yapay zeka problemini agentlarla çözmek anlamına gelmektedir.

2.4 Agent Mimarisi

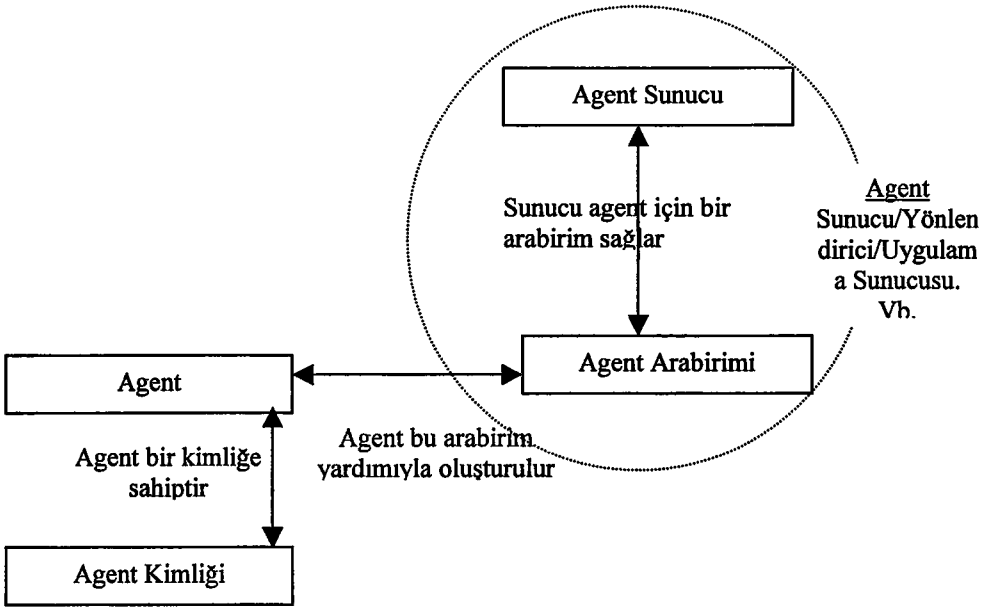
Burada bir agent mimarisi geliştirirken nelere dikkat edilmesinin gerektiği gösterilecektir. Bundan önce bahsedilen agent karakteristiklerinin desteklenmesi için bir takım gereksinimlerin sağlanması gerekmektedir.

Bu gereksinimleri şu şekilde sıralayabiliriz;

- Bir agent kendi tekil tanımlayıcına sahip olmalıdır.
- Bir agent bir çok agent'ın var olmasına ve aynı anda çalışmasına izin vermelidir.
- Agentlar diğer agentların agent sunucuda ne çalıştırdığını saptayabilmelidirler.
- Agentlar diğer agentların hangi mesajları alıp hangilerini gönderebileceklerini saptayabilmelidirler.
- Bir agent sunucu,agentların birbirleriyle ve agent sunucuyla haberleşmesine olanak sağlamalıdır.
- Bir agent sunucu bir agentın çalışmasını durdurup başka bir sunucuya transferini gerçekleştirebilmelidir.
- Bir agent sunucu , agentların doğrudan birbirlerinin işlerine karışmalarını engellemelidir.

Bu gereksinimler bahsedilen agent karakteristiklerinin bir çoğunun desteklenmesini sağlar.

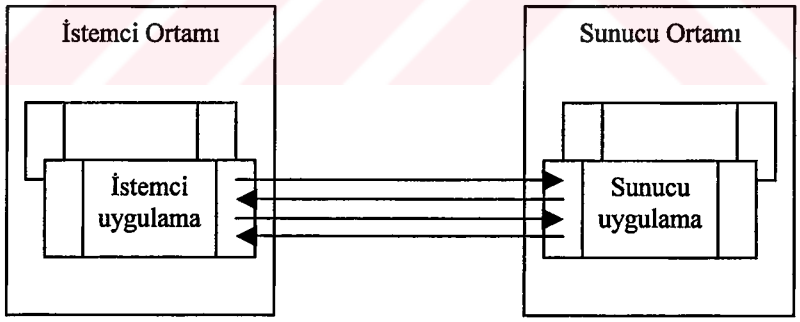
Agent mimarisin de , sistemdeki agentları denetleyen , birbirleriyle haberleşmelerini sağlayan , agentların taşınmasına yardım eden , agentların oluşturulması için arabirim olanağı sağlayan ve yukarıdaki özelliklere sahip bir **Agent Sunucu** kullanılır. Bu sunucuyu geliştirilen mimariye göre farklı şekilde adlandırmak mümkündür.



Şekil 2.1 Bir agent mimarisi.

2.5 Hareketli Agentlar

Hareketli agentların davranışlarını daha iyi anlayabilmek için öncelikle geleneksel ağ mimarisini incelemek gerekir. Aşağıdaki şekil tipik bir sunucu/istemci mimarisini göstermektedir.



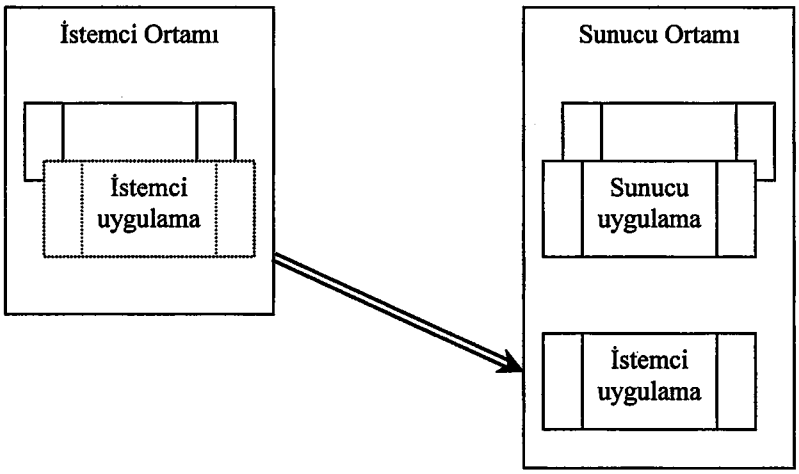
Şekil 2.2 Tipik istemci sunucu haberleşmesi.

Tipik istemci sunucu uygulama haberleşmesi istek ve cevaplarla olur, bu da ağ üzerinde yük oluşturur.

Tipik bir istemci sunucu uygulaması iki parçadan oluşur. İstemci ve Sunucu. Genellikle istemci ve sunucu ayrı makinelerde olup ağ üzerinden haberleşirler. İstemcinin bilgiye ihtiyacı olduğu zaman ağ üzerinden sunucuya bir istek bildirir.

Sonra sunucu da bu isteğe yanıt verir. Bu el sıkışma geleneksel istemci/sunucu mimarisinde çok sık olur. Her istek ve cevap ağ üzerinde yüke neden olmaktadır.

Yukarıdaki istemci/sunucu modeli ile hareketli agent mimarisini de şu şekilde karşılaştırmak mümkündür.



Şekil 2.3 Hareketli agent haberleşmesi.

Hareketli agent mimarisinde , istemci gerçekte sunucu tarafına gider ve isteği ağ üzerinden değil de doğrudan sorar.

Aynı istemci/sunucu mimarisinde olduğu gibi yine bir istemci ve bir de sunucu vardır. Farklılık haberleşme şekillerindedir. Hareketli agent mimarisinde istemci sunucunun sunduğu bir veriye ulaşmak istediği zaman sunucu ile ağ üzerinden konuşmaz. Bunun yerine , istemci sunucu tarafına kendini taşır. Bir kere sunucu makinesine ulaştığı anda da isteklerini doğrudan sunucuya bildirir. Bütün hareketler bittiğinde hareketli agent sonuçlarla birlikte geri döner.[3]

2.5.1 Hareketli Agentların Özellikleri

Hareketli agentların göze batan en önemli karakteristikleri program kodunun hareket edebilmesidir. Bununla beraber bu özellik gerekli olmasına rağmen yalnız başına yeterli değildir. Çünkü program kodunun hareketliliği ve verinin olduğu yerde hesaplamayı bizzat yapabilmek hareketli agentlara özgü bir karakteristik değil. Gerçekte bu tip bir hareketlilik bir çok modern sisteminde bir özelliğidir. Örneğin veritabanı dünyasında bunu, sunucuda çalışan altyordamlar (stroed procedure) olarak adlandırıyoruz. Sunucuda çalışan altyordamlar, istemci program kodunun sunucu tarafında çalışan küçük bir parçasıdır. Bazı uygulamalarda uygulamanın istemci

tarafı, dinamik olarak bu çeşit altyordamları sunucuya yükleyebilir. Sonra orada çalışıp sonuçları istemci tarafa iletirler.

Peki sunucuda çalışan altyordamlar ile hareketli agentlar arasındaki fark nedir?. Sunucuda çalışan altyordamlar esneklik ve otonomluk gibi bir takım özellikleri barındırmazlar. Önemli bir fark aslında hareketliliği sağlama metodudur. Sunucuda çalışan altyordamlar yüklendikleri sunucuya aittirler ve sürekli orada kalırlar. Eğer bir çok sunucudan bilgi çekme ihtiyacı doğarsa altyordamlar biraz önce anlattığımız geleneksel istemci/sunucu modelindeki çalışma şekli gibi,ağ üzerinde dolaşmaya zorlanır. Kısacası sunucuda çalışan altyordam bir sunucudan diğerine kendini taşıyamaz.

2.5.2 İstemci/Sunucu Modeline Karşı Hareketli Agentlar

Hareketli agentların klasik istemci/sunucu modeline tercih edilmesinin üç ana sebebi vardır.

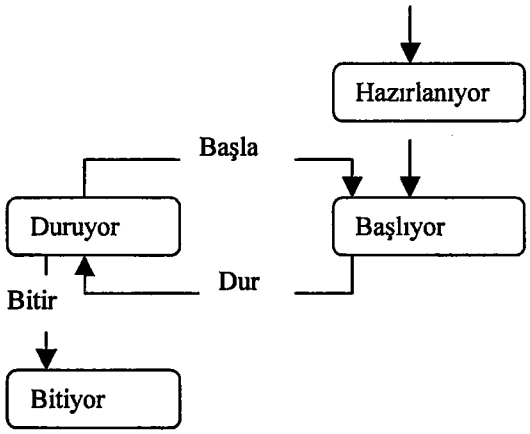
İlk olarak, hareketli agent mimarisi istemci/sunucu ağ bant genişliği problemini çözer. Bunu, sorguyu ağ üzerinden sunucuya göndermeyip sürekli el sıkışmayı gerektiren bir yapıdan kurtararak yapar.

İkinci olarak, agentlar tasarım riskini azaltır. Agentlar kodun konumu hakkında geliştirme aşamasında kabuller yapmaya destek verir. Mimari sistem tamamen bitirilip çalışır halde iken bile değişiklik yapmaya izin verir.

Üçüncü olarak , hareketli agent mimarisi güvenilir olmayan ve yavaş hat hız problemini de çözer. Agentlar çok kolaylıkla bağlantısız halde çalışmak için tasarlanabilir ve bağlantılı olduklarında da sonuçları iletmek için programlanabilirler.

2.5.3 Hareketli Bir Agentın Yaşam Süresi

Hareketli agentlar iki parçadan oluşurlar. İlk parça agentın ne yapacağını gösteren program kodunun kendisi ,ikinci parçada agentın şu anki durumudur. Bir agent yeni bir makineye (host) göç edeceği zaman bu iki parça da iletilmelidir. Aşağıdaki şekil hareketli bir agentın yaşam süresini göstermektedir.



Şekil 2.4 Hareketli bir agent'ın yaşam süresi.

2.6 Akıllı Agentlar

Bir agent'ı neyin akıllı yaptığı sorusunu cevaplamak zordur. Bu konu uzun zamandır yapay zeka alanının bir tartışma konusudur ve basit bir tanım henüz yapılamamıştır. Ancak bir agent'ı neyin akıllı yaptığına ilişkin aşağıdaki yazı akıllı agent deyince neyin anlaşılması gerektiğini ayrıntılı bir şekilde anlatmaktadır.

"Akıl, öğrenme ve karar verme yeteneğinin derecesidir. Agent için bu, kullanıcının amaçlarına yönelik emirleri almak ve üzerine düşen görevi yerine getirmek olarak açıklanabilir.

En azından , istekler ve bu istekleri yerine getirmeye yönelik tercih mekanizmaları , ki bu kurallar kümesi de olabilir , olmalıdır. Akıllı olmanın bir üst seviyesi , kullanıcı isteklerinin yerine getirilebilmesine yönelik neden gösterme ve anlama yeteneğinin olması ve bunların amaca yönelik olarak kullanılmasını içerir.

Bir sonraki aşama ise çevresine uyum sağlayabilen ve öğrenme kabiliyeti olan sistemleri içerir. Böyle bir sistem,bir asistan gibi, yeni ilişkileri , bağlantıları ya da düşünceleri keşfedebilme özelliğine ve bunları kullanıcı isteklerini yerine getirmek için uygulama yeteneğine sahiptir."

Yapay zeka bilimine getirilen eleştiriler aşağıdaki nedenlerden dolayı azaltılabiliyor;

- Yapay zeka biliminin , uzun yıllardır süren çalışmalara rağmen beklenilen kadar iyi sonuçlar veremediğini biliyoruz. Yazılım agentları bunun için bir kaçış noktası olarak kullanılabilir. Yapay zeka problemlerinin bazılarını çoklu bir agent

mimarisi (bağımsız agentlar bütün görevleri yerine getirmek için birlikte çalışıyorlar) kurmadıkça çözmek oldukça zordur.

- Agentlar ağlar arasındaki sınırların da kalkmasını sağlar.

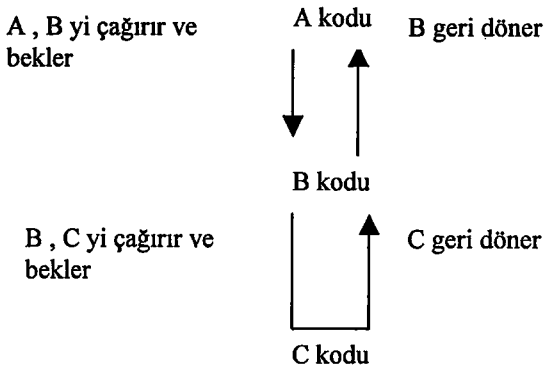
Akıllı agentlar çoğu zaman kavram olarak otonom agentlarla birlikte kullanılırlar. Otonom agent zaten akıllı bir agent olmak zorundadır. Otonomdan kasıt ise agentın bağımsız olabilmesi , kendi kararlarını verebilmesidir. Bu özelliklere öğrenme yeteneğinin de eklenmesi bir agent'ı hem otonom hem de akıllı yapar. Siz agent'a bir görev vereceksiniz ve o da bir şekilde sizin amacınızı yerine getirmeye çalışacaktır. Bunun için ağdaki bilgisayarları dolaşması, dolaştığı kaynaklardan bilgi çekmesi, başka agentlarla haberleşmesi ve öğrenmesi gerekecektir, sonra da sizin adınıza kararlar verip isteğinize ulaşmanızı sağlayacaktır.

2.7 Birbirleriyle Konuşan Agentlar

Tıpkı canlı yaşamında olduğu gibi söz konusu olan programlar,alt yordamlar,threadler yada agentlar da olsalar sahip oldukları veriyi diğerleriyle paylaşmak (haberleşmek) isterler. Bilgisayar bilimi , haberleşmeyi sağlamak için bir kaç mekanizma geliştirmiştir. Bunlardan en çok kullanılan üç tanesi aşağıda ayrıntılı olarak incelenmiştir.

2.7.1 Alt yordam çağırma yöntemi

Modern bilgisayar dilleri ile kullanılmaya başlayan ilk yöntemdir.



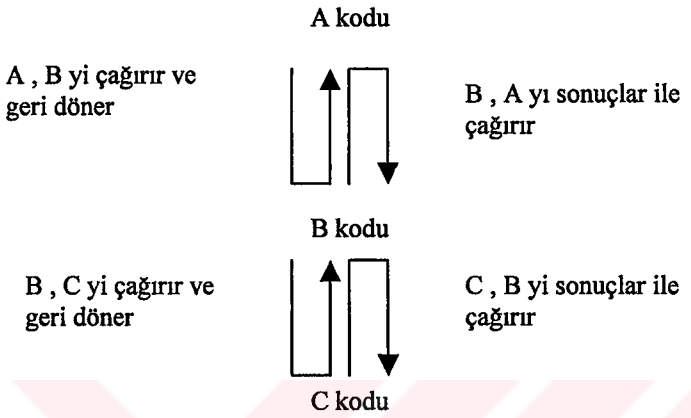
Şekil 2.5 Alt yordam çağırma yöntemi

A kodunun işini yapabilmesi için B nin servislerine ihtiyaç duyduğunu farz edelim. Ve B de A ya olan sorumluluğu yerine getirebilmek için C nin sunduğu servislere

ihtiyaç duysun. B bir süre çalışır ve C ye ihtiyacı olduğunu anladığında C den istekte bulunur. Programcılar olarak biz bu işleyişi her alt yordam çağırdığımızda görürüz.

Alt yordam çağırma yöntemi hızlıdır. Ve program akışının gözlenmesini kolaylaştırır. Ancak senkron bir çalışma sağlar ve paralel çalışmaya yatkın değildir.

2.7.2 Geri Çağırma (Call Back) Yöntemi



Şekil 2.6 Geri çağırma yöntemi

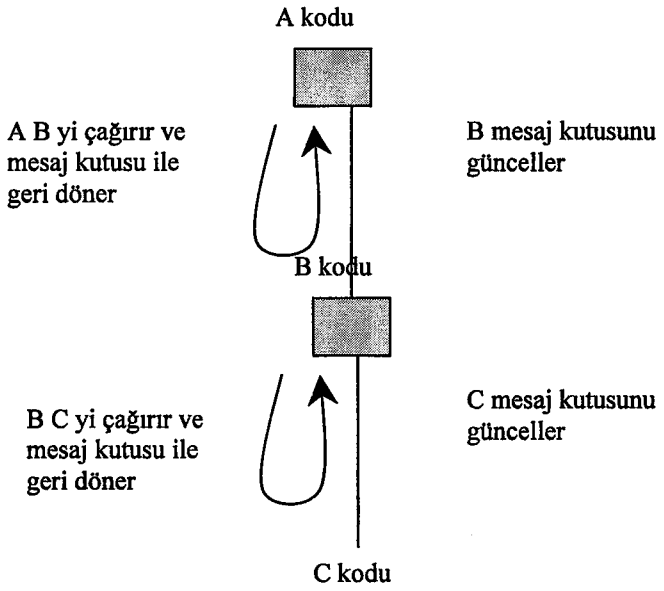
Bu yöntem altyordam çağırma yöntemine benzer. Bununla beraber bir istekte bulunup sonucu beklemek yerine A ve B istekte bulunup işlemlerine devam ederler. B ve C yapılması istenilen işlemleri bitirdikleri zaman kendilerini çağıran kod parçalarına bir çağrı ile sonuçları bildirir. İlk çağıran kod parçası,yaptığı iş varsa durdurup bu sonuçları almak zorundadır.

Bu yöntemin faydası asenkron çalışmaya olanak sağlamasıdır. Bununla beraber karmaşık olup program akışının gözlenmesini zorlaştırır.

2.7.3 Mesaj Kutusu Yöntemi

Yukarıda anlatılan iki yöntemin arasında olan bir yöntemdir.

Burada A ,B yi bir iş yapması için çağırır ve sonuçları mesaj kutusuna bırakmasını söyler, işini yapmaya devam eder. Sonrada periyodik olarak mesaj kutusunu sonuçların gelip gelmediğini anlamak için kontrol eder. Yada A çalışmasını durdurup B nin işini bitirmesini bekler. B ve C de aynı şekilde çalışır.



Şekil 2.7 Mesaj Kutusu Yöntemi

Mesaj Kutusu yöntemi , uygulama açısından daha zordur. Ancak asenkron bir çalışma sağlar ve program akışının gözlenme zorluğunu da giderir. Bu iki faktör dağıtık sistemlerde daha da önem kazanır.

2.7.4 Agentlar İçin En İyi Yöntem

Dağıtık sistemlerde mesaj kutusu yöntemi en uygun çözümdür. Çünkü proseslerin yaptıkları işlerin ağ üzerinde birden fazla makineyi ilgilendirdiği durumlarda program akışının gözlenebilmesi daha zordur ve geri çağırma gibi yöntemler bunu daha da zorlaştırır. İkinci olarak agentlar, doğaları gereği otonom, bağımsız dağıtık yazılım parçalarıdır. Gecikmeli bir ağ ortamında agentlar ,gelecekte ihtiyaç duyacakları şeyler için beklemeyi tercih etmezler.

Agentlar mesaj kutusu yöntemini şu şekilde kullanırlar:

Önce gönderen agent parametreleri içeren bir mesaj yaratır ve bunu alıcı agent'a yollar. Sonra alıcı agent gönderici agent'ın isteğinin karşılanıp karşılanmadığını anlaması için kullanabileceği ve mesaj kutusuna koyduğu "mesaj yanıtı" olarak adlandırılan bir "token" verir. Gönderici agent bu token'a sahip olduktan sonra bilgiyi mesaj kutusundan çekmek için başka bir şeye ihtiyaç duymaz.

2.8 Agent Haberleşme Dilleri

Agent tabanlı uygulamalarda agentlar agent bağımsız bir semantiğe sahip bir dil kullanırlar.[5]

- Agent Process Interaction Languages ,agent uygulamaları geliştirmek için oluşturulmuş, çoklu yürütme (multi tasking) ,ağdan bağımsız mesaj iletimi ve sembolik işleme sağlayan Imperial College de ESPRIT projesinin bir parçası olarak geliştirilen bir dildir.

- Shoham'ın AGENT0 prototipi agent tabanlı programlamanın prensiplerini sunar.

- Coen'nin SodaBot adı verilen dili , agent davranışlarının insan seviyesinde tanımlamalarla dizayn edildiği genel amaçlı bir dildir.

- The AGENTS Kernel Language (AKL), İsviçre bilgisayar bilimleri enstitüsünde geliştirilen bir agent haberleşme dilidir.

- Darpa Knowledge Sharing Effort haberleşme dillerini kullanan agent tabanlı araştırmalar için destek vermiştir. Bu çalışmalar sonucunda da standartlaşmış agent haberleşme dilleri oluşmuştur. KQML (Knowledge Query and Manipulation Language) ve KIF (Knowledge Interchange Format) yüksek seviyeli agent haberleşme dilleridir.

2.8.1 Knowledge Query and Manipulation Language (KQML)

KQML yada "Knowledge Query and Manipulation Language"[6] , veri ve bilgi alışverişi için kullanılan bir dildir. Dil , Darpa Knowledge Sharing Effort 'un tekrar kullanılabilir ve paylaşılabilir büyük tabanlı bilgi tabanları oluşturmak için yaptığı çalışmaların bir sonucudur. KQML hem mesaj formatı hem de agentlar arasında çalışma zamanı (run time) bilgi paylaşımını sağlayan mesaj protokolüdür. KQML bir uygulama içinde akıllı diğer bir uygulama ile konuşmak için bir dil olarak yada bir veya birden fazla sistemle beraber problem çözme amacıyla bilgi paylaşımı için kullanılabilir.

KQML agent tabanlı programlar için, programların asenkron ve otonom olmaları açısından kullanılabilir en esnek ve faydalı dildir. Otonomluğun anlamı agentların farklı hatta çakışan gündemlerinin olabilmesidir ki KQML mesajının anlamı mesaj alıcısı yerine göndericisi üzerindeki sınırlamalar baz alınarak açıklanır. Bu, mesaj alıcısına fonksiyonlarına uygun düşen davranışları seçme olanağı verir. Tabiki

agentların mümkün olduğunca beraber çalışabiliyor olmaları istenir ancak tıpkı insan yaşamında olduğu gibi tam anlamıyla bir birliktelik çoğu zaman mümkün olmaz.

KQML mesajı "performative" olarak adlandırılır. Bu meaj gönderenin bir girişimde bulunup iş yapma isteğini anlatan bir kelimedir. Agentlar KQML dilinin rezerve edilmiş performative leri dışındaki bütün performative leri kullanabilirler. Ve hatta agentlar kendi performativelerini oluşturabilme imkanına da sahiplerdir.

2.8.2 KQML Mesaj Formatı

Aşağıda bir KQML mesaj formatı gösterilmiştir.

```
(Performative :sender sender_name :receiver
receiver_name :language KQML :content (content))
```

Örnek;

```
(tell :sender useragent :receiver brokeragent :language
KQML :content(do first job))
```

Performative; Mesaj gönderen agent'ın mesajı gönderdiği agent'a yapmasını söylediği yada istediği davranışı anlatan sözcüktür. Bu tamamiyle programcının agent'ı yazarken karar verdiği ve ayrılmış (rezerved) sözcüklerin dışındaki herhangi bir sözcük olabilir. Örneğin tell,ask,read, sendme,get,put gibi. Kısaca burada davranış biçimini belirliyoruz. Mesajı alan agent mesajdaki performative alanını çözdükten sonra davranışlarını belirleyebilir. Tabi ki önceden belirli bir program mantığı oluşturulmuş olmalıdır. Yani bir agent "ask" olarak performative alanını belirleyip mesaj göndermiş ise mesajı alan agent kendisine "ask" performative alanlı bir mesaj geldiğinde ne yapması gerektiğini bilmelidir. Ya buna uygun bir davranış sergileyecek, ya mesajı gönderen agent'a farklı ya da aynı performative değerine sahip yeni bir mesaj gönderecek yada mesajla ilgilenmeyecektir.

Mesajı alan agent genelde aşağıdaki program kodunu işletir.

```
Perform=parseferpormative (message)
Sender=parsesender(message)
Content=parsecontent(message)
If (perform = = perform1) { Do something }
Else
```

```
If   {perform= =perform2){ Do something }
else {Do nothing}
```

content; Mesajı alan agent performative alanını okuduktan sonra eğer bir iş yapması gerekiyorsa işin ayrıntısını yada bu mesaj kendisine iletilen bir veriyse bunun ayrıntısını "content" alanından öğrenir. Bu alan oldukça uzun bir karakter kümesi olabilir.

Sender; Mesajı gönderen agent'ın ismidir.

Receiver; Mesajın gönderildiği agent'ın ismidir.

Language; Mesaj formatının tipidir .(KQML) KIF formatında mesaj da yollanabilir.

Örneğin toplama işlemini başka bir agenta yaptıran bir agent yazalım. İlk agent toplama işlemi yapmak istediği zaman toplanacak sayıları başka bir agenta göndersin ve toplama işlemini yapan agent ise geriye sonucu göndersin.

AgentA:

```
X = readln("X=?")
Y = readln("Y=?")
KQMLmessage = "plus :sender AgentA :receiver AgentB
:Language KQML :content(X,Y))"
Sendmessage(AgentB,KQMLMessage)
Do
If there_is_message_for_me
{
perform = = parseperformative(message)
 if( perform = = "plusresult")
     {
     Printf(parsecontent(message))
     Stop=true
     }
else
```

```
printf("I don't understand this message"}

While (stop= =true)
```

AgentB:

```
Stop = false

Do

If there_is_message_for_me

{

perform = = parseperformative(message)

 if( perform = = "plus")

      {

      params = parsecontent(message)

      firstparam = parse(params)

      secondparam = parse(params)

      Total = firstparam+secondparam

      KQMLmessage = "plusresult :sender AgentB :receiver
      AgentA :Language KQML :content(Total))"

      Sendmessage(AgentA,KQMLMessage)

      }

else

printf("I don't understand this message"}

While (stop= =true)
```

Görüldüğü gibi KQML oldukça yüksek seviyeli bir agent haberleşme dilidir. Dilin güzel yanı kolay anlaşılır ve esnek olmasıdır. Programcı kendi performative lerini kolayca belirleyebilir. Agentlar insanlar gibi birbirleriyle böylece yüksek seviyeli bir dille konuşabilir hale gelebilmektedirler. KQML sadece bir mesaj formatı değil aynı zamanda bu mesajların belirli kurallara göre iletilmesini sağlayan bir mesaj iletim protokolüdür. Tıpkı SMTP veya FTP gibi. Nasıl SMTP de mesaj gönderen kişi mesaja kendi adresini , alıcının adresini, konuyu ve içeriği yazıyorsa KQML de de gönderen agent'ın ismi, alıcı agent'ın ismi, performative alanı ve içerik bilgisi mesajı oluşturur ve mesaj iletilir.

Dikkat edilmesi gereken nokta burada agentların asenkron bir haberleşme alt yapısı ile karşı karşıya olduklarıdır. Mesajı gönderen agent cevabı beklemek zorunda değildir. Daha önce de açıkladığımız gibi agentlar haberleşirken mesaj kutusu yöntemini kullanırlar dolayısı ile mesajı gönderen agent kendisine mesaj gelip gelmediği mesaj kutusunu kontrol ederek anlamaktadır.

Daha sonra da ayrıntılı açıklanacağı gibi, agentlar internet ortamından mesajları doğrudan birbirlerine yollamazlar. Aynı birbirlerine elektronik posta gönderen iki kişi gibi gönderici mesajı posta sunucusuna gönderir ve sunucu bu mesajı alıcısına yollar. Mesaj istemci kendisine gelen mesajları mesaj kutusunu kontrol ederek görür. Agent haberleşmesinde neden böyle bir mekanizmanın kurulduğu daha ayrıntılı olarak incelenecektir.

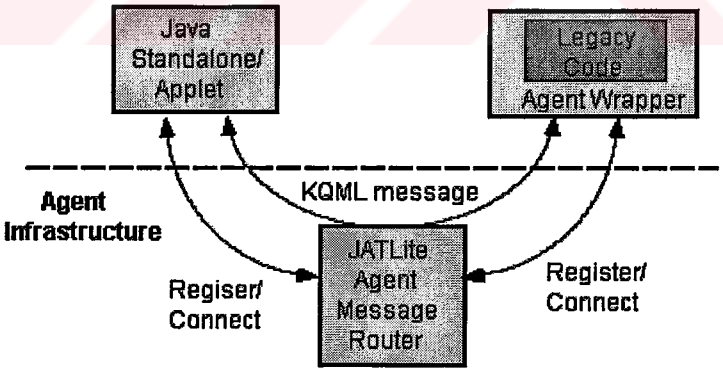
2.9 Neden Java

Agent tabanlı uygulamaların dil sınırlaması yoktur. Dolayısıyla agent'ı yazarken bilinen herhangi bir dil kullanılabilir. Ancak bir uygulamanın agent tabanlı olmasının ne demek olduğunu , ve agentların hangi genel kriterlere sahip olmaları gerektiği görülmüştü. Özelliklede internet ortamında agent tabanlı bir yapı kurulması isteniyorsa ortamdan ve platformdan bağımsız bir mimarinin kurulması gerektiği açıktır. Ayrıca seçilecek dilin, hareketliliğe (mobility) ve paralel çalışmaya da destek veriyor olması gerekmektedir. Javanın platformdan bağımsız olması , internet ortamında istemci uygulaması için tarayıcı(browser) tarafından yorumlanabiliyor olması , uzak makinelerdeki kaynakların kullanımı için hazır fonksiyonlar sunması (RMI) , ağ haberleşmesi için socket programlamayı desteklemesi agentları yazarken en çok kullanılan dil olmasını sağlamıştır.

3. JAVA AGENT TEMPLATE,LITE (JATLite)

3.1 JATLite Nedir

JATLite (Java Agent Template,Lite) [4] internet üzerinde güvenli bir şekilde haberleşebilen yazılım agentları yazabilmek için java dilinde geliştirilmiş program paketidir. JATLite aynı zamanda aşağıdaki şekilde görüldüğü gibi agentların isim ve şifre vererek kayıt yaptırabilecekleri, bağlantı kurup koparabilecekleri, mesaj gönderip alabilecekleri,FTP gibi protokoller ile dosya alışverişinde bulunabilecekleri, birbirleriyle haberleşebilecekleri bir yapı sağlar. JATlite ilk kez 1996 yılında Stanford üniversitesinde Prof. Mark Cutkosky önderliğin de H. Robert Frost adlı bir master öğrencisi tarafından yazılmıştır. Amaç bütün agentların haberleşebileceği standart bir haberleşme ortamı oluşturabilmekti. Bu aşamadan sonra JAT, ücretsiz olarak bu ürünü kullanmak isteyen herkes için Stanford Üniversitesi web sitesine konulmuştur. Frost Stanford Üniversitesinden ayrılmadan önce üç farklı ve daha karmaşık versiyonunu yazmıştır. Mart 1996 yılında Mr. Heecheol Jeon JAT'a Agent Message Router (AMR) alt yazılımını ekleyerek appletlerin de çalışabilmesine olanak sağlamıştır ve adı JATLite olarak anılmaya başlamıştır.Yine bu çalışmalar Prof. Cutkosky ve Dr. Petrie önderliğin de yapılmıştır.



Şekil 3.1 JATLite Agent Mesaj Yönlendiricisi

3.2 JATLite'ın Faydaları

Agent sistemleri oluşturma ve bu sistemlerde hata ayıklama zor bir işlemdir. Agent tabanlı yeni sistemler farklı platformlarda çalışabilmesi ve appletlerin de geçici agentlar olarak çalıştırılabilmesi için Java dilinde yazılmaktadır. JATLite, bu tip sistemleri kolayca oluşturabilmek için java agent alt yapısı ve temel agent şablonları sunar.

Jatlite,yüksek seviyeli dil ve protokol kullanımını sağlayan agentları oluşturabilmek için şablonlar verir. Bu şablonlar programcıya, önceden tanımlı ve agent oluşturmayı sağlayan bir çok java sınıflarını hazır sunar. Daha da önemlisi bu sınıflar katmanlı bir yapıda sunulur, böylece programcılar oluşturacakları sistemler için hangi sınıfların gerektiğine kolayca karar verebilirler. Örneğin eğer programcı KQML kullanmak istemez ise KQML katmanındaki sınıflar da göz ardı edilir. Bununla birlikte eğer bu katmanlar kullanılırsa buradaki sınıflar da otomatik olarak sisteme dahil edilmiş olur.

3.3 JATLite'ı Farklı Yapan Özellikler

JATLite'ı tek yapan en önemli özellik,birlikte gelen agent altyapısıdır. Geleneksel agent sistemleri agentlar arasındaki bağlantılar için agent isim sunucusu kullanırlar. Bir agent basitçe bağlantı kuracağı agentın ismini ve IP adresini sunucudan öğrenir ve doğrudan agent ile bilgi alışverişi için bağlantı kurur.

Böyle bir sistemde bir agent'ın IP adresi değiştiğinde ilk agent ona bir mesaj gönderdiğin de ve bu başarısız olduğun da , durumdan haberdar olacaktır. Eğer ikinci agent herhangi bir nedenden dolayı çalışamaz duruma gelirse, başarısız mesajların tekrar alıcısına ulaştırılması ve güvenli bir haberleşme ortamının sağlanmasıyla sistemdeki her agentin kendisi meşgul olacaktır. Bu durumlarda dağıtık hesaplamaların ne kadar kolay bir şekilde başarısız kalabileceği anlaşılmaktadır.

Jatlite alt yapısı ile, Agent Mesaj Yönlendiricisine(AMR) her agent tek bir bağlantı kurar.Agent mesaj yönlendiricisi gelen mesajları isimleri ile en son bilinen IP adreslerine yönlendirir. Aynı zamanda aynı bir elektronik posta sistemi gibi ,mesaj yönlendiricisi mesajların hepsini alıcı agent mesajı alıp "silme" isteği yollayana kadar tutar.

Böylece agentlar mesaj yönlendiricisine kayıt olma isteklerini ve sonra da bağlantı kurma ve koparma isteklerini de mesajla bildirebilirler. Dağıtık hesaplama ortamı, her zaman mesaj yönlendiricisi tarafından korunur.

3.4 Agent Mesaj Yönlendiricisi(AMR)

JATLite agent mesaj yönlendiricisi, kayıtlı olan bir agenttan gelen mesajı doğru alıcısına yönlendiren özel bir uygulamadır. Alınan mesaj dosya sisteminde de kuyruk yapısı ile saklanır. Böyle bir uygulamanın geleneksel agent isim sunucu kullanmaya göre üç önemli avantajı vardır.

- Applet olarak çalışan bir agent, popüler tarayıcılardan kaynaklanan güvenlik sınırlamalarına rağmen diğer agentlarla haberleşebilir.
- Alıcı agentın mesajı alamaması durumunda asenkron haberleşme mümkündür.
- Bir agent diğer agentların adresleri veya gönderilemeyen mesajların durumu ile ilgilenmek zorunda değildir.

3.5 JATLite Agentları ve Akıllı Agentlar

JATLite,agentların haberleşmesi ve etkileşimli çalışmalarını sağlamak dışında özel yetenekler sunmaz. Yani JATLite ,insan davranışlarını taklit eden, bilgi arayan agent şablonları sağlamaz. Programcı sistemi için neyin doğru olduğuna kendisi karar verebilsin diye serbest bırakılmıştır. Jatlite bu amaçla özel bir araç sunmaz ama kullanımını destekler.

Jatlite'ın paket alt yapısı agentların bir makineden diğer makineye gitmelerine yani hareketli olmalarına , internet üzerinden otomatik mesaj biriktirme ve kuyruklama yöntemleri ile sisteme girmeye sistemden çıkmaya olanak sağlar. Bu özellikler, agentların bozulabileceği, konumlarını değiştirebileceği sistemlerde gereklidir ve Mesaj Yönlendiricisi alt yapısı ile sağlanmaktadır.

Mesaj Yönlendiricisi aynı zamanda hareketli agentlar tarafından da kullanılabilir. Jatlite otonom agent taşınması için bir metod sunmasa da yer değiştiren agentlar arasında mesaj geçirimi için Mesaj Yönlendiricisi çok uygundur. Kullanıcının agent'ı bir yerden bir yere taşınması için programlamış olup olmamasıyla ilgilenmez ancak agentın gelecekteki bütün mesajlarını toplamak için uğraşır.

Güzel bir özellik applet olarak çalışan agentlar üzerindeki bazı tarayıcıların sınırlamalarının mesaj yönlendiricisi ile ortadan kalkmasıdır, böylece applet agentlar her tür tarayıcı ile çalışır duruma gelmektedir.

3.6 JATLite'ın Kullandığı Standartlar

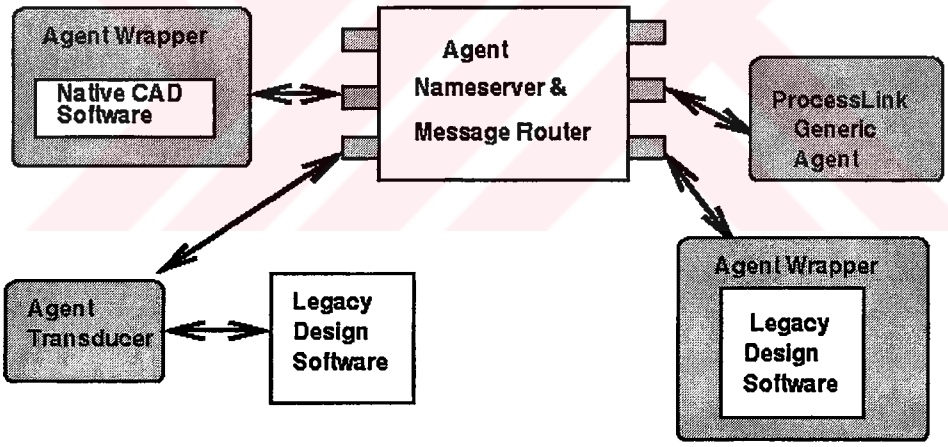
JATLite agent haberleşmesi için KQML dilini kullanır. KQML daha önce de söylendiği gibi hem mesaj formatını ham de mesaj iletim protokolünü kapsayan bir kavramdır. Haberleşme daha alt seviyede TCP/IP,SMTP,FTP gibi internet standartları üzerine oturtulmuştur. Ancak programcı diğer agent dillerini kullanan agentlar oluşturabilirler bunun için JATLite'ın katmanlı mimarisinden faydalanır.

JATLite , hiçbir otonom agent teorisi üzerine yönelmez. Çünkü agentların iç mimarileri hakkında henüz bir ortak anlayış yoktur.

JATLite , Sun'ın JDK 1.1 'i ile kullanılmak zorundadır. Jatlite agentları çalıştıran tarayıcılar JDK 1.1 plug'inini kullanmalıdır.

3.7 Agent Mesajları

JATLite'ın kullanılan en önemli özelliği,yaşayan uygulamalara,diğer programlarla haberleşme, mesaj gönderip alma için gerekli arayüzü sunmasıdır. Aşağıdaki şekil bu işleyişi göstermektedir.



Şekil 3.2 Jatlite'ın varolan uygulamalar için sağladığı arayüz

Jatlite, agent haberleşmesi için standart yazılım sağlar. KQML ve diğer agent haberleşme protokolleri mesaj alışverişi için standartlar sağlarlar.

Anahtar fikir; en olağan şekilde yazılım parçalarını haberleştirebilmektir. Nesne tabanlı bir dil oluşturma zorunluluğu da yoktur. ASCII karakter katarları ile bu sağlanabilmektedir. CORBA gibi nesne platformları gerekli değildir.

Diğer yazılımları JATLite'a entegre etmek çok kolaydır. JATLite agentları C,C++,LISP ' e de kolayca entegre edilebilir.

3.8 JATLite'ın Yetenekleri

- Paketin diğer parçalarını etkilemeden diğer teknolojilerle yer değiştirebilen katmanlı bir yapıya sahiptir.
- TCP/IP üzerine kurulu düşük düzeyli haberleşme Unix,Windows,MAC OS gibi bilinen işletim sistemleri tarafından desteklenir,diğer protokollerde kolayca eklenebilir.
- Agent mesajları KQML dili ve protokolü üzerine kurulmuştur. Mesajların dış katmanları için doğrudan sentaks kontrolü vardır. Mesajların iç yapıları diğer herhangi bir dilde olabilir (SQL,Express,KIF vb.)
- Çoklu thread işlemleri ve çoklu sunucu ve mesaj alma soketleri. Soket bağlantıları kalıcıdır ve zaman aşımına sahiptir.
- Agentların kaydolması, bağlanması,bağlantıyı kesmeleri isim ve şifre servisleri için Agent Mesaj Yönlendiricisi sunar.
- Mesaj Yönlendiricisi hareketli agentlar için mesaj biriktirme ve kuyruklama yeteneğine sahiptir.
- Java ve C++ ile yazılmış agentları ve tarayıcılar tarafından desteklenen applet agentlarını destekler.
- FTP ile dosya transferi olanağı

3.9 JATLite Varsayımları

JATLite Sun'ın JDK1.1'ini destekleyen herhangi bir patformda çalışabilir.Windows95, WindowsNT,Solaris, MAC OS gibi işletim sistemleri buna dahildir. Diğer java ortamları için değişiklik yapmak gerekebilir. Applet agentlar, Netscape,Internet Explorer gibi tarayıcılar yada Sun'ın applet görüntüleyicisi ile çalışabilir. JATLite agentları standart TCP/IP haberleşmesi ve prosesler arası haberleşme için de soketler üzerine kurulmuştur. Bütün mesajların , internete bağlı bilgisayarlarda java uygulaması olarak çalışan Agent Mesaj Yönlendiricileri aracılığı ile gönderildiği kabul edilir .

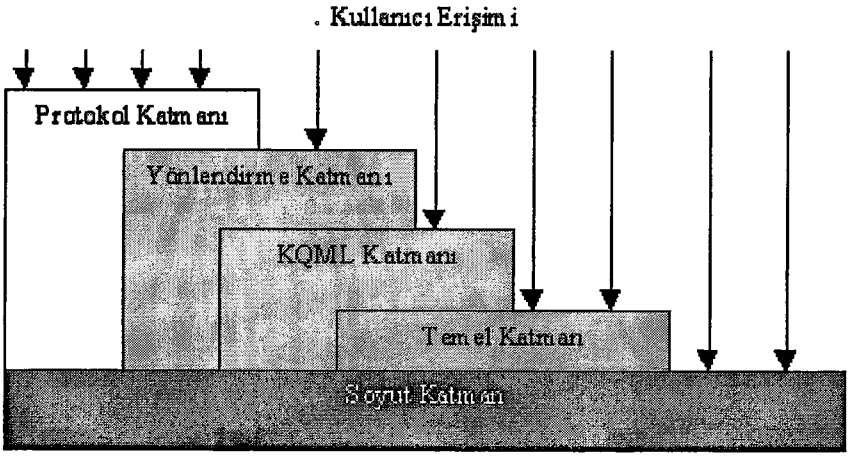
Bir uygulamanın JATLite uyumlu olabilmesi için KQML standardına göre ASCII mesajları gönderebiliyor olması ve mesaj yönlendiricisi bağlantı protokolü ile yönlendirici ile bağlantı kurabiliyor olması gerekmektedir.

Genel çalışma varsayımları;

- TCP/IP tabanlı haberleşme
- Mesaj alışverişi ile agent haberleşmesi
- Bağlantılı her agent için bir bağlantı
- Her agent'a tek bir adres(isim,makine,port,mesaj-metodu,açıklama). Eğer agent appletlerdeki gibi sunucu soketine sahip değilse host 'a bir değer atanmaz ve port numarası da ,"-1" e eşitlenir.
- Mesaj sonu karakteri "\004" 'e eşitlenir.

3.10 JATLite Katmanları

JATLite mimarisi aşağıdaki şekildeki gibi her biri özel olarak tasarlanmış katmanlı bir yapıya sahiptir. Böylece programcı, kuracağı yapının ihtiyaçlarına göre istediği katmandan başlayarak programını geliştirebilecektir. Mesela TCP/IP üzerine haberleşme isteyen ancak KQML yi kullanmak istemeyen bir programcı aşağıda açıklandığı gibi sadece Soyut ve Temel katmanları kullanacaktır.



Şekil 3.3 JATLite her biri büyük ölçüde özelleştirilmiş katmanlı bir yapıdan oluşur.

Soyut Katman,JATLite'ın kullanımı için gerekli olan soyut sınıfları sağlar. JATLite bütün haberleşmelerin TCP/IP olduğunu farz etmesine rağmen ,programcı isterse bu katmanı değiştirerek başka protokol mesela UDP kullanımını da sağlayabilir.

Temel Katman,soyut katman ve TCP/IP ya dayanan temel haberleşme rutinlerini sağlar. Temel katman mesela soketten okumayı sağlayacak ve çıkışları bir dosyaya yazacak şekilde geliştirilebilir. Çoklu mesaj portlu agentları destekleyecek şekilde de geliştirilebilir.

KQML Katmanı,KQML mesajlarının ayrıştırılması ve saklanmasını sağlar. Kayıt olma , bağlanma, bağlantıyı kesme gibi işlemlerde "Center for Design Research"'[6] un önerdiği KQML standardı biraz daha geliştirilerek standartlaştırılmıştır.

Yönlendirme Katmanı,agentlar için isim kaydetme,mesaj yönlendirmesi ve kuyruk mekanizması olanağı sağlar. Bütün agentlar diğer herhangi bir agenta agent mesaj yönlendiricisi aracılığı ile mesajlarını yollarlar. Bir agentın çalışması sona erer ya da bozulursa Yönlendirici mesajları agent geri gelene kadar saklar. Java ve World Wide Web güvenlik kısıtlamaları düşünülünce özellikle appletler için yönlendirici oldukça önem kazanmaktadır.

Protokol katmanı,SMTP,FTP,POP3,HTTP gibi internet standartlarını, hem java uygulamaları hemde appletleri için sağlar. Projede kullanılan versiyon SMTP ve FTP yi destekliyor ancak kısa zamanda diğer protokoller de ele alınıp geliştirilebilir. Eğer bir agent farklı bir koddaki veriyi veya uzun bir veriyi yada KQML mesajlarını email ile göndermeyi istiyorsa protokol katmanı oldukça yardımcı olacaktır.

3.11 JATLite'ın Geleceği

Gelecekte JATLite'a elektronik posta haberleşmesi için bir katman daha eklenmesi düşünülmektedir. Böylece JATLite agentları diğer agentlarla elektronik posta aracılığı ile haberleşebilecektir. Bu türden bir haberleşme ağ koruyucu (firewall) içeren yapılar için olumludur

JATLite Agent Mesaj Yönlendiricisine DNS 'in çalışma mantığı gibi bir yapı düşünülmektedir. Böylece JATLite agentları daha ölçeklenebilir hale gelmektedir.

Mesaj Yönlendiricisi test edilmesine rağmen şu anki yapıda sistem mesaj yönlendiricisinin bozulması ile çalışamaz hale gelmektedir. Mesaj yönlendiricisini

kontrol eden ve bozulduğu anda yenisini yaratabilen yapılar üzerinde çalışmalar sürmektedir.

KQML dili desteklenmeye devam edilecektir ancak bir çoğu KQML katmanını kullanmayarak kendi sistemleri için diğer dilleri kullanma yoluna gidebilecektir , veritabanı işlemleri için SQL olabileceği gibi.

3.12 CORBA veya Java RMI

CORBA ve RMI ,JATLite ile uyumludur ve bir çok açıdan beraberce kullanılabilir. Eğer programcının yazdığı agent dili java arayüzüne sahip değilse ,örneğin, CORBA çevirici olarak kullanılabilir.

JATLite'ın yerini tamamen alabilmek açısından iki dezavantaj vardır. Bütün yazılım parçaları CORBA nesneleri yollamalıdır ve diğeri, Mesaj Yönlendiricisi yeniden yazılmalıdır.

3.13 Katman Seçimi

Dil ve protokoller üzerindeki kısıtlamalar karşın JATLite beş farklı katman sunar.

- Eğer KQML kullanılacaksa,temel haberleşme metodları üzerinde düşünülmesine gerek kalmadan doğrudan KQML katmanından başlanabilir. Alınan mesaj KQML sentaksına göre kontrol edilecek ve KQMLmesaj objesi içine saklanacaktır.

- Eğer Agent İsim Sunucusu ve Mesaj Yönlendiricisi kullanılacaksa, O zaman Yönlendirme katmanından başlanabilir. Bu katman mesaj yönlendirme , alma, saklama için güzel metodlar sağlar.

- Eğer agentların veri alışverişinde bulunmaları isteniyorsa protokol katmanından başlanabilir. Bu katman FTP ile veri alışverişi için metodlar sağlar.

3.14 Şablonlar

JATLite her katman için şablonlar sunar.

BaseLayer: BagentAction , Abstract.AgentAction nın alt sınıfı.

KQMLLayer: KQMLAgentAction, BaseLayer.BagentAction'ının alt sınıfı.

RuterLayer:RouterClientAction,KQMLLayer.KQMLAgentAction'ının alt sınıfı.

ProtocolLayer:IPRouterClientAction,
RouterLayer.AgentClient.RouterClientAction'ının alt sınıfı

Act(Object) ve ProcessMessage(String,Object) metodlarınından faydalanmak için şablonları kullanmak ve yeni veri üyeleri ve metodları eklemek istenebilir.

3.15 Şablonların Kullanımı

Şablonları tanımlamak için Yönlendirme Katmanı ve RouterClientAction üzerinden açıklama yapmak daha kolay olacakır.

Aşağıdaki basit adımlarla, agentınızın applet yada uygulama olmasından ve diğer agentların çalışıyor yada çalışmıyor olmasından bağımsız olarak , agentınız JATLite'ın sağladığı çeşitli servisler (mesaj yollama,kuyruklama,isim sunuculuğu,KQMLmessage nesnesine otomatik mesaj dönüştürme, KQML parse işlemi, agent listesi,vb.) sayesinde diğer agentlar ile bağlantılar kurabilmektedir.

RouterClientAction şablonunu kullanmak için aşağıdaki adımlar yürütülmelidir.

A- AgentAction metodonuz için kurucu fonksiyonları hazırlanır (mesela MyRouterClientAction gibi);

Genelde super class(RouterClientAction) 'ını kullanmak yeterli olur. RouterClientAction() kurucusu my adresss, Router Adresse, ve registrar adresss dışındaki tüm veri üyelerine ilk değerleri atayacaktır. Bu yüzden eğer bu kurucu fonksiyon kullanılacaksa,RouterClientAction nesnesine adresleri vermek için setMyAddress(Address) , setRouterAddress(Address) , setRegistrarAddress(Address) metodlarını kullanmak gerekir.

RouterClientAction(Address routerAddress, Address registrarAddress,Address myAddress, int maxidleTime) kurucusunu adresleri set etmek için kullanılabilir. Eğer bağlantı maksimum zaman aşımı süresini kullanılacaksa, maxidleTime değerine -1 verilmelidir yada eğer değer verilmeyecekse bu değere dakika cinsinden atamalar yapılmalıdır.

B- Act(Object) ve processMessage(String,Object) metodlarını doldurulur;

Eğer diğer agentlardan bir mesaj alınırsa Act(Object) metodu otomatik olarak canlanır. Bu yüzen mesaj alındığı farzedilip buna göre agentların davranışları burada belirlenir.processMessage(String,Object) özel bir metod değildir ancak grafik arayüzler ve threadlerle etkileşimleri tanımlamak için kullanılabilir. ProcessMessage metodu kullanılmayacaksa boş bırakılmalıdır

Public void processMessage(String cmd, Object obj) {}

C- RouterClientAction nesnesi yaratılır ve gerekliyse kayıt (register) ettirilir sonra da yönlendiriciye bağlanılır;

Şimdi RouterClientAction nesnesi hazır durumdadır. Bir örnek yaratıp önce register() metodu ile yönlendiriciye kayıt olunmalı sonra da bağlanmak için connect() metodu çağırılmalıdır.

```
MyRouterClientAction action =
new MyRouterClientAction(routerAddress, registrarAddress ,myAddress,
100);
// Yönlendiriciye bir kez kayıt olmanız gerekiyor.

action.register();
action.connect();
```

Adres bilgileri program içinden ya da kullanıcı giriş ekranından alıp parametre olarak aktarılabilir.

3.16 AgentClientClass Sınıfının Kullanımı

AgentClientClass sınıfının kullanımı iki integer sayı üzerinde işlem (+,-,*,/) yapıp sonucu geri yollayan örnek bir agent üzerinde anlatılacaktır.

A- AgentAction class :

```
public class CalcServer extends RouterClientAction {
......
}
```

B- Router adresi, registrar adresi ve agent adresi için bir kurucu yazılmalıdır. Maksimum zaman aşımı için de parametre geçirilebilir.

```
public CalcServer(Address myaddress,
Address routeraddress,
Address registraraddress,
int durationtime) {
//  RouterLayer.AgentClient.RouterClientAction
kurucusunu kullan
super(myaddress,routeraddress,registraraddress,durat
iontime);
}
```

C- Act(Object obj) metodu doldurulur : Act() metodu yönlendiriciden her mesaj geldiğinde canlanacaktır.Sadece diğer agentlardan mesaj alındığı farzedilip agentın ne yapması gerekiyorsa o belirlenmelidir.

```
public boolean Act(Object o) {
String stampmsg = (String)o;
try {
//KQMLmail  KQMLmessage'ını kapsar
KQMLmail mail = new KQMLmail(stampmsg,0);

//  mail'i  _mailQueue ya ekle.
// Burası mesaj alındıkan sonra silinmesiyle
ilgilidir

_mailQueue.addElement(mail);

// KQMLmessage objesi için KQMLmail getKQMLmessage()
metodunu kullan.
KQMLmessage kqml = mail.getKQMLmessage();

// performative değerini bul.
String perf = kqml.getValue("performative");

// mesaj içeriğini al.

String content = kqml.getValue("content");

String receiver = kqml.getValue("sender");

//hata kontrolü
```

```
if(content == null) {
sendErrorMessage(kqml);
return false;

}

if(perf.equals("ask-one")) {
.....

}
else {
// hata mesajı
sendErrorMessage(kqml);
return false;
}

addToDeleteBuffer(0);
} catch (Exception e) {
return false;
}

return true;
}
```

D- processMessage(String,Object) metodu doldurulmalıdır : Bu metod genelde GUI yada diğer threadlerde kullanılır. Bu örnek için uygulanmamıştır.

AgentAction 'ı yaratılır , kayıt olunup bağlanılır.

```
public static void main(String args[]) {
// RCAddressHelper kullanılacaksa
if(args.length == 1) {
try {
....
// dosyadan adres bilgilerini oku.
...
CalcServer server = new
CalcServer(myAddress,routerAddress,
registrarAddress,idletime);
```

```
// Uygulama olduğu için ,sunucu thread'i yaratın
createServerThread(myaddress.getID(),Thread.NORM_PRI
ORITY);

if(registerRequest == true)
server.register(registraraddress,myaddress);
// agent adresi ile yönlendiriciye bağlanın

server.connect(myaddress);

// mesaj almak için beklemeye geç
server.start();

} catch (Exception e) {
System.out.println(e.toString());
System.exit(0);
}
}
}
```

E- Uygulama dosyası CalcServer.java olarak saklanılır ve derleyip çalıştırılır.
RCApplet (yada IPRcApplet)'i kullanarak CalcServer'a mesaj yollanır.

CalcAddressFile dosyasını değiştirilir;

(RouterLayer/Resource/CalcAddressFile):
CalcServer port numarası değiştirilmeli (Yönlendirici ve CalcServer aynı makinede çalışıyorlarsa port numaraları farklı olmalı)

CalcServer 'ı çalıştırılır;
java RouterLayer.AgentClient.Example.CalcServer.CalcServer
RouterLayer/Resource/CalcAddressFile
Yada java RouterLayer.AgentClient.Example.CalcServer.CalcServer

F- Mesaj yollanır

Performative'i 'ask-one' ve receiver'ı 'CalcServer' yapıp ve aşağıdaki mesajlar yollanılır.
+ 1 4
- 4 235

* 134 346
/ 345 5

3.17 Yüksek Seviyeli public RouterClientAction metodları

- **createServerThread**(String id, int priority) - Eğer agentınız bir java uygulaması olarak çalışacak ise diğer agentarın bağlanması için bu metodu kullanmalısınız
- **setRouterAddress**(Address) - Yönlendiricinin adresinin belirlenmesi.
- **setRegistrarAddress**(Address) - Registrar'ın adresinin belirlenmesi.
- **setMyAddress**(Address) -Agent adresinin belirlenmesi.
- **register**(Address registrarAddress,Address myAddress) throws ConnectionException - Yönlendiriciye kayıt olma.Adresler gönderilmelidir.
- **register**() throws ConnectionException - Yönlendiriciye kayıt olma. Agent adresi ve regirtrar adresi önceden belirlenmelidir.
- **connect**() throws ConnectionException - Yönlendiriciye bağlanma. Agent adresi ve yönlendirici adresi önceden belirlenmelidir.
- **connect**(Address myAddress) throws ConnectionException - Yönlendiriciye bağlanma.Yönlendirici adresi önceden belirlenmedilir.
- **connect**(String myName) throws ConnectionException - Yönlendiriciye kaydolma. Agent adresi ve yönlendirici adresi önceden belirlenmelidir. .
- **sendMessage**(String receiver,String msg) throws ConnectionException - Diğer agenlara mesaj yollama.
- **sendMessage**(String msg) throws ConnectionException - Yönlendiriciye mesaj yollama.
- **sendKQMLMessage**(String msg) throws ConnectionException, ParseException - Yölendiriciye KQML sentaks kontrolü yaparak mesaj yolla.
- **addToDeleteBuffer**(int) - Mesajı sil..'delete-message' yönlendiriciye son durumda yollanmalıdır.
- **disconnect**() - Yönlendirici ile bağlantıyı kes.

- **unregister**() - Yönlendiriciden kaydını sil.
- **endAction**() - Agent'ı temizle ve durdur.

3.18 Protokol Katmanının Kullanımı

Eğer SMTP/FTP gibi protokol katmanı servislerinden faydalanılmak isteniyorsa; ProtocolLayer.IPRouterAction sınıfı kullanılmalıdır.

IPRouterClientAction sınıfının kullanımı RouerClientAction sınıfının kullanımı ile neredeyse aynıdır. IPRouterClientAction processmessage metodunu SMTP/FTP kullanımını sağlamak için kullanır. Bunu aşağıdaki kodda belirtildiği gibi kullanmak gerekmektedir. Bununla beraber sendEmailMessage(), FTP() yi uygulamalar için CreateEmailConnection() ve CreateDataTransferConnection metodlarını da appletler için kullanılabilir.

```
public void processMessage(String cmd,Object obj) {
super.processMessage(cmd,obj);
///// Buraya uygulamanızı yazın
.....
}
```

'cmd' parametresi "SMTP" ve "FTP" ile başlamamalıdır.

Yalnız başına çalışan java uygulamaları ve appletler farklı güvenlik kısıtlamalarına sahip oldukları için SMTP ve FTP için durum aşağıdaki gibidir.

3.19 Java Uygulamaları

- Java uygulamaları sunucu soketi yaratmada ve diğer makinelere bağlanmada serbest olduğu için SMTP ve FTP kullanan uygulamalardaki gibi doğrudan kullanılır.
- SMTP: SMTP kullanabilmek için kurucu fonksiyona parametre olarak geçen IPRouterClientAction nesnesi içinde agentın email adresini belirtmek gerekir. KQML mesajlarını email olarak yollamak için sendEmailMessage(Address sender,Address receiver,String msg) yada sendEmailMessage(String kqmlstr) metodları kullanılabilir. Eğer agentınızın email adresi "Address sender" parametresiyle belirtiliyorsa ve agentınız alıcının email adresini biliyorsa, ilk metod,alıcının adresini bilmiyorsa ikinci metod kullanılabilir.İkinci metod

alıcının email adresini yönlendiriciden öğrenecektir. Eğer yönlendiriciye daha önceden email adresi bildirilmemişse ConnectionException hatası oluşacaktır.

- FTP: FTP yi kulanmanın en kolay yolu,FTPCon nesnesini kullanmak olacaktır. FTPCon nesnesi tanımlanır ve FTP(FTPCon) metodu çağırılır. Bu metod hatalar için ConnectionException hatası oluşturacaktır.

3.20 Appletler

- Appletlerde, soket bağlantılar için bir takım kısıtlamalar olduğu için, SMTP ve FTP sunucuları yönlendiricinin çalıştığı HTTP sunucudan farklı bir yerde ise bu sunuculara doğrudan bağlantı kurulamaz. Appletler için yönlendirici SMTP ve FTP sunuculara erişimde proxy sunucu görevi görür ancak KQML mesajlarında olduğu gibi yönlendirici hiçbir mesajı saklamaz.

- SMTP: SMTP yi kullanmak için kurucu fonksiyona parametre olarak geçirilen IPRouterClientAction nesnesi içinde agentın email adresinin belirtilmesi gerekmektedir. CreateEmailConnection(String kqmlmsg) yada createEmailConnection(KQMLmessage kqml) metodları KQML mesajlarını email olarak yollamak için kullanılabilir. Yönlendirici appletin bağlanacağı yerde sunucu soketi oluşturur ve applete 'port-info' mesajı yollar. Hata durumlarında 'port-info' mesajı yerine hata mesajı yollar. 'port-info' mesajı appletin bağlanacağı sunucu soket numarasını da içerir. Applet sunucu sokete bağlantı kurduktan sonra,IPRouterClientAction otomatik olarak SMTP veri akışını başlatacaktır (sendEmailToRouter(KQMLmessage) metodu kullanılarak) böylece doğrudan sendEmailToRouter(KQMLmessage) metodunu çağırmak zorunda kalmayız. Hata durumlarında bu metod connectionExceptin hatası oluşturacaktır.

- FTP:Bir applet çalıştığı yerel kaynak üzerindeki dosya sistemine erişemez ancak JATLite'ın sağladığı FTP fonksiyonu sayesinde uzak sistemlere dosya transfer edebilir. CreateDataTransferConnection(FTPCon con) yada CreateDataTransferConnection(String conid , String remotePath,String host, int port,String userName,String password,String localPath,String[] files, int mode) metodlarını çağırabilirsiniz. SMTP de olduğu gibi IPRouerClientAction yönlendiriciden sunucu soketi isteğinde bulunacak ve yönlendirici de 'port-info' mesajı yollayacaktır. Sonra IPRouerClientAction sunucu soketine bağlandıktan sonra transfere başlayacaktır. 'port-info' mesajı , processRouterMessage(KQMLmessage) tarafından işlenecektir.

4. BİLGİ TOPLAMA SİTEMİ

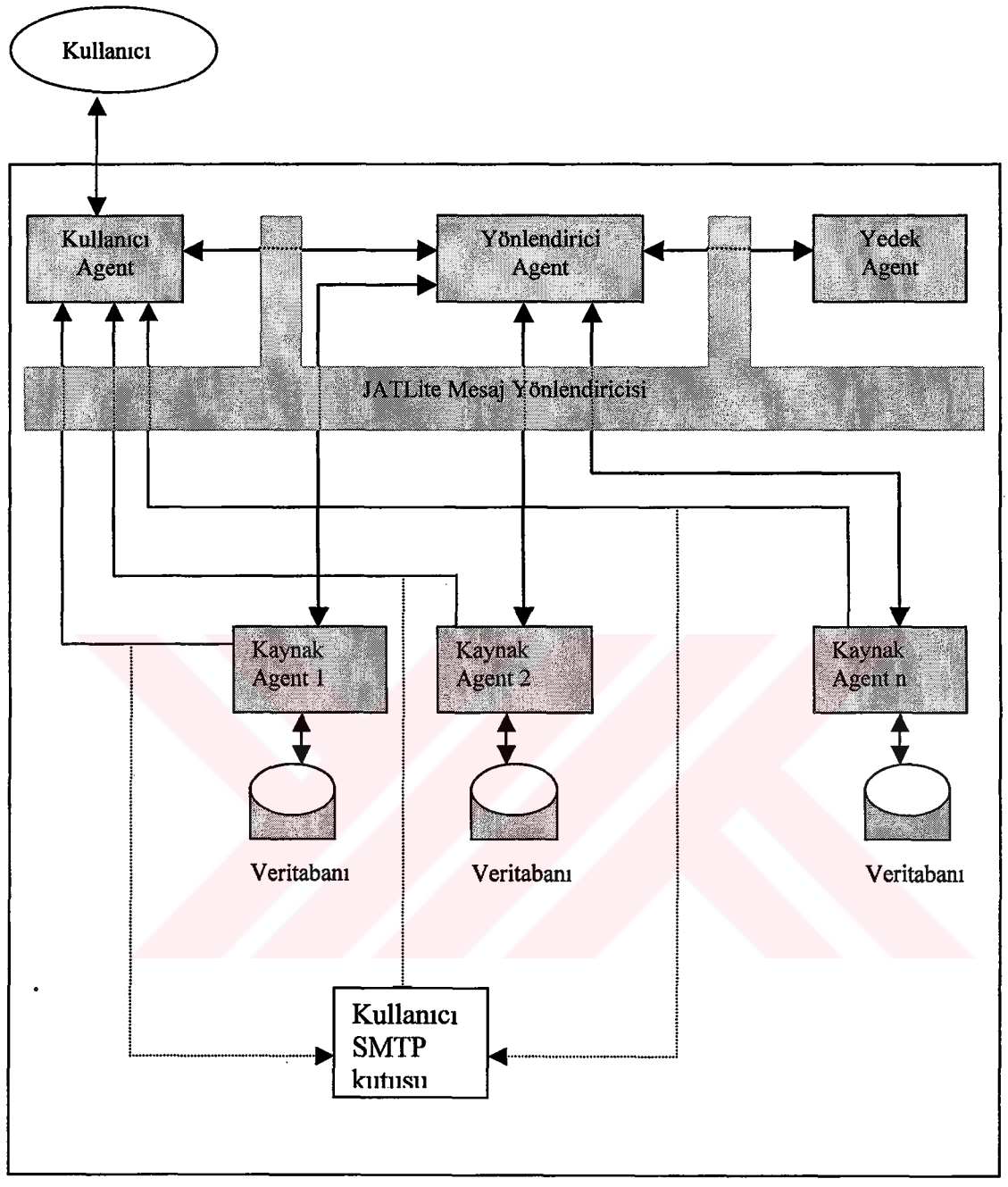
Bilgi Toplama Sistemi,kullanıcı adına bilgisayar ağında bilgi arayan agent tabanlı bir sistemdir. Kullanıcıya sağlanan bir arayüzden aranmak istenen bilginin türü ve aranacak bilgi girişi yapılır ve sistemde bu bilgiye ilişkin bir arama yapılarak sonuçlar kullanıcıya , yine sağlanan bu arayüzden gösterilir ve ya elektronik posta ile gönderilir. Altavista, Yahoo gibi arama motorları amaç olarak bu türden bir çalışmaya örnektir, ancak bilgi toplama sistemi agent tabanlıdır ve asenkron çalışabilir. Sistemin bu türden sistemlere göre avantajları ileriki bölümlerde açıklanmıştır.

Saydamlığı sağlamak ve farklı görevleri yapmak için çeşitli agentlara ihtiyaç vardır. Sistem 4 farklı agent üzerine oturtulmuştur, Kullanıcı agent, Yönlendirici agent, Kaynak agent ve Yedek agent. Bunlardan Yönlendirici agent ve Yedek agent sistemde bir tanedirler ancak çok sayıda Kaynak agent ve Kullanıcı agent örneği sistemde yer alabilir.

Her agent sistemde kendisini ismiyle tanımlayan bir tekil tanımlayıcıya sahiptir .Kullanıcı agent sisteöden aralanacak olan kullanıcıyı temsil ettiğinden, Kullanıcı agentlarını simgeleyen tekil bilgi , kullanıcı adı olarak belirlenmiştir. Sisteme girmek isteyen her agent bir isim ve şifre ile Agent Mesaj Yönlendiricisine (AMR) kayıt yaptırır. Bu, daha önceden aynı isimle başka bir agentın kayıt yaptırmamasıyla mümkündür. Kayıt yaptırdıktan sonra agent sisteme bağlanabilir. Kayıt yaptırma işlemi bir kez geçerli olan bir işlemdir. Bundan sonra agent, herhangi bir zamanda sisteme bağlanabilir veya bağlantısını kesebilir. Agentın kaydını sildirmesi de mümkün olmakla birlikte kullanıcı tarafında bu özellik aktif ettirilmemiştir.

Bilgi toplama sistemi JATLite kullanılarak java dilinde yazılmış bir uygulama paketidir. Bundan önceki bölümde anlatılan JATLite teknik özellikleri bilgi toplama sistemi için de geçerlidir. Ancak sistem mesajlaşma temeli üzerine kurulduğu için agentların KQML mesajları ile nasıl anlaştıkları ve birbirlerine nasıl veri aktardıkları daha ayrıntılı olarak incelenecektir. Her bir agent ayrıntıları olarak incelenecektir.

Şekil 4.1 bilgi toplama sisteminin genel mimarisidir. Oklar bilgi akışını göstermek için kullanılmıştır.



Şekil 4.1 Bilgi toplama sisteminin mimarisi

4.1 Yönlendirici Agent

Yönlendirici agent sistemde tektir ve merkezi karar verici elemandır. Kullanıcıdan gelen istekleri değerlendirir ve ilgili yerlere bu isteği yönlendirir. Onun kontrolünde istek bir veya daha fazla kaynak agentlar tarafından yönetilen veritabanlarına gönderilir.

Yönlendirici agent çevresinde olup biten çoğu şeyden haberdardır. Yeni agentların sisteme girişi ve çıkışından da bilgisi vardır. Özellikle kaynak agentlara periyodik olarak bir takım kontrol mesajları yollayarak çevresi hakkındaki en güncel bilgileri edinmeye çalışır bu da ona öğrenme ve karar verme özelliğini kazandırır. Kullanıcı isteklerini de sadece ilgili kaynak agentlara yollar çünkü bir süre sonra sistem hakkında yeterli bilgiye ve hangi istekleri hangi kaynak agentlara yönlendirmesi gerektiği bilgisine sahip olur.

Yönlendirici agent sisteme kayıt yaptıran agentları, sisteme bağlantı kurmuş agentları, kullanıcı istekleri hakkındaki ayrıntılı bilgiyi ,sonucun nasıl sunulması gerektiği gibi, izler. Yönlendirici agent java uygulaması olarak çalışır ve diğer agentlar onunla haberleşmek için ismini bilmek zorundadırlar. Ancak kullanıcı agenttan bu bilgi saydamdır.

4.1.1 Yönlendirici Agent Bilgi Tabanı

Yönlendirici agentın zamanla öğrenme yeteneğine sahip olduğunu söylemiştik. Zamanla oluşan bilgi tabanını ya karar vermek için yada kendisinden istenen isteklere cevap vermek için kullanır. Üç temel tablo yönlendirici agentın bilgi tabanını oluşturur ve bu tablolar dinamiktir.

a- Agents tablosu

Tablo 4.1 Agents tablosu.

Agent Adı	Agent Durumu
AgentA	Connected
AgentB	Disconnected
AgentC	Connected
AgentD	Connected
.....	.....

Agents tablosu yönlendirici agent 'ın "sistemde hangi agentlar var ve durumları nelerdir" sorusuna yanıt verebilmesi için tutulur. Sisteme her giren agent bu tabloya eklenir ve bağlantısını kesen her agentın durumu da güncellenir. Burada "connected" agentın sisteme bağlı olduğunu "disconnected" ise agentın kayıt yaptırdığını ancak şu anda sistemde bulunmadığını gösterir. Yönlendirici agent kullanıcıdan gelen istekleri kaynak agentlarına yönlendirirken bu tablodan faydalanır,sisteme bağlı olan agentlar,bu tablodan öğrenilir.

b- Categorytable tablosu

Tablo 4.2 Categorytable tablosu

Agent İsmi	Category İsmi
AgentA	Arabalar
AgentB	Oteller
AgentC	Ülkeler
AgentD	Yazarlar
.....	.....

Categorytable tablosu, Kaynak agentların hangi türde bilgi tuttuklarını anlamak için kullanılır. Bu bilgi kaynak agentlara periyodik olarak gönderilen kontrol mesajları ile öğrenilir. Yönlendirici agent belirli zaman aralıkları ile sistemdeki tüm kaynak agentlara ne türden bilgi tuttuklarını sorar ve bu mesajı alan her kaynak agent'ı bunu cevaplar. Cevaplarla güncellenen tablo, kullanıcı agentın belirli türden bilgiye ulaşmak istemesi durumunda başvurulan yerdir. Yönlendirici agent bu tablodan kullanıcının istediği kriterde bilgi tutan agentları öğrenir ve bu agentlara kullanıcı agent'ın mesajını yönlendirir.

c- Categorylist tablosu

Tablo 4.3 Categorylist tablosu

Kriter Adı
Kriter 1
Kriter 2
Kriter 3
.....

Categorylist tablosu sistemde hangi türden verilere erişilebileceği bilgisini tutar. Kullanıcı kriter listesini öğrenmek istediğinde yönlendirici agent bu tablodan bakıp yanıtı verir. Bu tabloda diğer iki tablo gibi dinamiktir ve sürekli güncellenir.

Yönlendirici agentın diğer bir bilgi tabanı da sistemde daha sonra yapılması gereken arama işlemlerine ait bilgilerdir. Kullanıcı aramak istediği bilginin ileriki bir tarihte yapılmasını isteyebilir, bu durumda istekler yönlendirici agent tarafından saklanır ve zamanı geldiğinde sisteme sunulur. İleri tarihe kayıt ettirilmiş arama istekleri yönlendirici agentın bir veri tabanında tutulur. Bu veritabanının yapısıda aşağıdaki gibidir.

Tarih	Saat	Kriter	Aranacak Bilgi	Aramayı yapan kullanıcı

Tarih: Kullanıcı sorgusunun hangi tarihte yapılmasını istiyor

Saat: Kullanıcı sorgusunun hangi saatte yapılmasını istiyor

Kriter: Hangi kriter alanında bilgi aranıyor

Aranacak bilgi: Aranması istenilen bilgi

Aramayı yapan kullanıcı: Sorguyu hangi kullanıcı (kullanıcı agent) yaptırıyor.

Tablo 4.4 Yönlendirici Agent Mesaj Tablosu

Performative	Alıcı	İçerik	Açıklama
Status	Kaynak Agent	Null	Kaynak agentlardan durumları öğrenilir.
ListUsers	JatLite Router	Null	Jatlite Routerdan sistemdeki agent listesi alınır.
Listreply	Kullanıcı Agent	Kriter Listesi	Kullanıcı agent'a arama yapabileceği krşter listesi yollanır.
Retreive	Kaynak Agent	Aranacak Bilgi+Kullanıcı Agent	Kullanıcı isteklerinin kaynak agentlara yönlendirilmesi, sonuçlar kullanıcı ekranında gözükecek.
Retreiveviaemail	Kaynak Agent	Aranacak Bilgi+Posta kutusu ismi	Kullanıcı isteklerinin kaynak agentlara yönlendirilmesi, sonuçlar kullanıcı posta kutusuna gönderilecek.
SearchRegistered	Kullanıcı Agent	Null	İleriki tarihte kullanıcının yapılmasını istediği sorguların tutulduğuna ilişkin onay mesajı.
Nodata	Kullanıcı Agent	Null	Kullanıcı Agent'a aradığı türden

			bir kayıt bulunamadığının bilgisi.
Sentviaemail	Kullanıcı Agent	Null	Sonuçların posta kutusuna gönderileceğinin onay mesajı.

4.1.2 Yönlendirici Agent 'ın Öğrenme Aşamaları

Yönlendirici Agent, periyodik olarak sistemdeki kaynak agentlara kontrol mesajları yollar. Bu mesaj tipi aşağıdaki gibidir.

Status	Kaynak_Agent_İsmi	Null

Performative: "Status"

Receiver: Kaynak Agent

Content:Null

Mesajı alan Kaynak agentlar yanıt olarak performative alanını yine "Status" yaparak mesaj içeriği kısmında tuttukları kriter bilgisini yollarlar. Böylece Yönlendirici agent "categorytable" ve "categorylist" tablosunu güncellemiş olur.

Yönlendirici agent sadece kaynak agentlara periyodik kontrol mesajları yollamaz. JATLite mesaj yönlendiricisinden de sistemdeki en son agent listesini öğrenir. Bu mesaj formatıda ağaşıdaki gibidir.

ListUsers	JATLite Router	Null

Performative: "ListUsers"

Receiver:JATLiteRouter

Content:Null

Mesajı alan JATLite Router yanıt olarak kendisine kayıt olan bütün agent isimlerini ve o anki durumlarını liste halinde geri yollar. Performative alanı "Registered-agent" ve mesaj içeriği olarakta liste halinde agent isimleri ve durumları yazılarak geri yollanır. Böylece yönlendirici agent "agents" tablosunu güncellemiş olur.

4.1.3 Yönlendirici Agentın Gönderdiği İstek ve Bilgi Mesajları

Yönlendirici agent çevresini tanıması için gereken kontrol mesajları dışında, kullanıcı isteklerini yanıtlamak ve bilgi vermek içinde bir takım mesajlar gönderir. İstek mesajlarının çoğu kullanıcı agent tarafından gönderilir.

4.1.3.1 Kullanıcı agenta kriter listesinin gönderilmesi

Listreply	Kullanıcı Agent	Kriter Listesi

Performative: "Listreply"

Receiver: Kullanıcı Agent

Content: Kriter Listesi

Kullanıcı agent kriter listesini öğrenmek istediği zaman performative alanına "List" koyarak bunu yönlendirici agenta yollar , mesajı alan yönlendirici agent bildiği en son kriter listesini performative alanına "Listreply" ve content alanında da kriter listesini ekleyerek kullanıcı agenta yollar.

4.1.3.2 Arama istek mesajının yönlendirilmesi

Retreive/Retrieveviaemail	Kaynak Agent	Aranacak bilgi

Performative: "Retreive" / "Retrieveviaemail"

Receiver: Kullanıcı Agent

Content: Kriter Listesi

Yölendirici agent kullanıcı agenttan gelen ve performative alanı "Search" olan mesajı aldıktan sonra ilgili kaynak agentlara bu mesajı performative alanına "Retrieve" koyarak tek tek yollar.

Eğer kullanıcıdan sonuçların email ile yollanmasını belirten ve performative alanı "Sendviaemail" olan mesaj gelirse bu durumda yönlendirici agent performative alanına "Retrieveviaemail" koyarak kaynak agentlara yönlendirir.

4.1.3.3 Kullanıcı agentı bilgilendirme mesajları

Yönlendirici agent kullanıcı agenta yaptığı işlerin hangi aşamada olduğunu yada sonuçlarını mesajlarla bildirir. Bu amaçla üç mesaj çeşidi kullanır.

"Searchregistered"/"nodata"/"sentviaemail"	Kullanıcı agent	Null

Performative:

"Searchregistered", Kullanıcı agentın ilerki tarihte yapılmasını istediği sorgunun saklandığını bildirir.

"Nodata", Sistemde kullanıcının aradığı kriterde bilgi tutan bir agent olmadığını kullanıcı agenta bildirir.

"sentviaemail", Sonuçları email ile isteyen kullanıcı agenta isteğin yapıldığını bildirir.

Receiver: Kullanıcı agent

Content: Null.

4.2 Kullanıcı Agent

Kullanıcı agent gerçek kullanıcı ile bilgi toplama sistemi arasında köprü görevi görür. Applet olarak kullanıcı tarayıcısında çalışır. Grafik arabirimi vardır ve sonuçları bu arabirim üzerinden gösterebilir. Temel görevi isteği almak ve yönlendirici agenta yollamaktır. Diğer görevi sonuçları ekranda uygun formatta göstermektir. Kullanıcı agent bilgi kaynakları hakkında bir bilgiye sahip değildir. Her biri ayrı kullanıcıyı tanımlayan sayısız kullanıcı agent sistemde yer alabilir. Sistemdeki her bir kullanıcı aslında bir agent'ı temsil ettiğinden agent tekilliği kullanıcı tarafında "kullanıcı adı" ve "şifre" ile sağlanmaktadır. Sisteme girecek olan her kullanıcı önceden "kullanıcı adı" ve "şifre" alıp kendilerini kaydettirirler sonra istedikleri anda sadece sisteme bağlanarak bilgi arama isteğinde bulunabilirler.

4.2.1 Kulllanıcı Agent İstek Mesajları

Kullanıcı agent sistemde sadece yönlendirici agenttan haberdardır. Yönlendirici agenta gönderdiği istek mesajlarının yanıtlarını kaynak agentlardan doğrudan alır.

Tablo 4.5 Kullanıcı Agent Mesaj Tablosu

Performative	Alıcı	İçerik	Açıklama
List	Yönlen. Agent	Null	Kriter listesinin alınması
Search	Yönlen.Agent	Kriter+Aranacak Bilgi	Sorgu mesajı.
RegisterSearch	Yönlen.Agent	Zaman bilgisi+Aranacak bilgi	Sorgunun ileriki bir tarihte yapılmasını söyleyen mesaj.
SentviaEmail	Yönlen.Agent	Email Adresi+Aranacak bilgi	Sorgu sonuçlarının bir posta kutusuna yollanması.

4.2.2 Kriter Listesinin Alınması

Kullanıcı arama yapabileceği kriter listesini yönlendiriciden istemek için aşağıdaki mesajı yollar.

"List"	Yönlendirici Agent	Null

Performative: "List"

Receiver: Yönlendiri agent

Content: Null

Mesajı alan Yönlendirici agent performative alanına "ListReply" koyup content alanına da kriter bilgisini yerleştirerek kullanıcı agenta geri yollar. Kullanıcı agent bu listeyi ekranda uygun formatta gösterir. Content alanında "No_Data" olması mevcut herhangi bir kriter olmadığını gösterir.

4.2.3 Sorguların Yapılması

Kullanıcı agent bilgi aramak istediğinde aşağıdaki mesaj yönlendiriciye yollanır.

"Search"	Yönlendirici Agent	Kriter+Aranacak Bilgi

Performative: "Search"

Receiver: Yönlendirici agent

Content: Kriter+Aranacak bilgi

Mesajı alan Yönlendirici agent aranacak bilgiden kriter detayını çözer ve bu kriterde bilgi tutan agentlara aranacak bilgiyi ve bu bilgiyi isteyen kullanıcı agent ismini content alanını doldurarak yollar.

4.2.4 Sorguların Kaydedilmesi

Kullanıcı , isterse aramayı gelecekteki bir tarihe kaydettirebilir. Bu amaçla Yölendirici agenta gönderdiği mesaj aşağıdaki gibidir.

"RegisterSearch"	Yönlendirici Agent	Zaman bilgisi+Aranacak bilgi

Performative: "RegisterSearch"

Receiver: Yönlendirici Agent

Content: Zaman bilgisi+Aranacak bilgi

Mesajı Alan Yönlendirici Agent content alanından zaman bilgisini öğrenir ve sorguları sakladığı veri tabanına , tarih,saat,sorgu,sorguyu isteyen kullanıcı gibi bilgileri yazar. Periyodik olarak da bu veritabanını okuyarak zamanı gelmiş sorgular varsa onları işleme alır.

Yönlendirici agent bunları başarılı bir şekilde yaparsa performative alanı "Searchregistered" olan bir mesajı Kullanıcı agenta yollar.

4.2.5 Sonuçların Email Adresine Gönderilmesi

Kullanıcı isterse aradığı bilgileri ekranda görmek yerine bir SMTP mesaj kutusuna da gönderilmesini isteyebilir. Bu amaçla gönderdiği mesaj aşağıdaki gibidir.

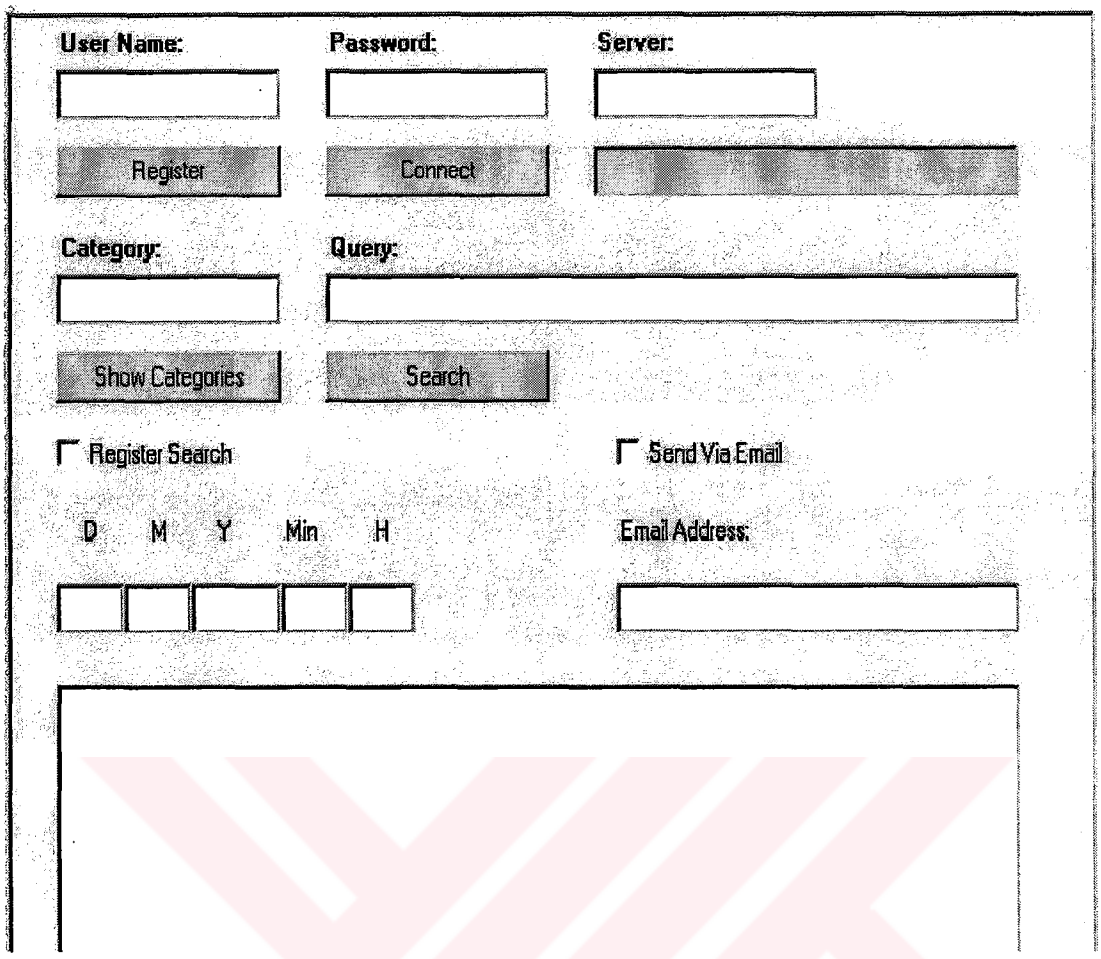
"Sentviaemail"	Yönlendirici Agent	Email Adresi+Aranacak bilgi

Performative: "Sentviaemail"

Receiver: Yönlendirici Agent

Content: Emai adresi+Aranacak Bilgi

Mesajı alan Yönlendirici Agent ilgili kaynak agentlara bu bilgiyi gönderir,kaynak agent ise sonuçları SMTP mesaj kutusuna yollarlar.



Şekil 4.2 User agentın sağladığı grafik arayüzü

4.3 Kaynak Agent

Kaynak agentlar bilgi veritabanlarına erişimi sağlarlar. Veritabanlarını gerektiği zaman sorgularlar ve kullanıcı isteğini doğrudan kullanıcıya yada verilen bir elektronik posta adresine yollarlar. Her kaynak agent kriter adı verilen belirli tipteki bilgiye erişimi sağlar. Bir kaynak agent sisteme girerken hangi kriterde bilgi tuttuğunu yönlendirici agenta söyler. Sisteme yeni bir kaynak agentın eklenmesi, sadece var olan bir kaynak agentin kopyalanması ve konfigürasyon dosyasının değiştirilmesiyle mümkündür.

Sistemde aynı yada farklı kriter de bilgiye ulaşımı sağlayan birden fazla kaynak agent bulunabilir.Kaynak agentlar birbirleriyle haberleşmezler. Her kaynak agent aynı yada farklı makinelerde bir java uygulaması olarak çalışırlar. Her bir kaynak agentın uygulama biçimi farklı olabilir. Veritabanına nasıl ulaştıkları, nasıl sorgulama teknikleri kullandıkları ve verinin veritabanında nasıl tutulduğu sadece

kaynak agenti ilgilendiren konulardır ve diğer agentlar tarafından bilinmezler. Agent sadece kendisine gelen ve kendisinin göndermesi gereken mesaj formatını bilmelidir.

Kaynak Agenta iki tür mesaj gelebilir. Bunlardan ilki yönlendirici agentın sistem hakkında bilgi toplamasına yönelik olan kontrol mesajlarıdır. Bu mesajla yönlendirici agent kaynak agent'ın hangi kriterde bilgi tuttuğunu öğrenir. Bu işlem periyodik olarak yapılır. İkinci türden mesaj Yönlendirici agentın Kullanıcı agenttan gelen sorgu isteklerini yönlendirdiği mesajlardır. Kaynak agen bu istekleri yerine getirir ve sonuçları doğrudan Kullanıcı agenta yollar.

Tablo 4.6 Kaynak Agent Mesaj Tablosu

Performative	Alıcı	İçerik	Açıklama
Status	Yönlen. Agent	Durum Bilgisi+Tutulan Kriter Bilgisi	Yönlendirici Agent'a kaynak agentın tuttuğu veri türünü bildirmesi.
RetrieveReply	Kullanıcı Agent	Sonuçlar	Sonuçların Kullanıcı Agent'a gönderilmesi

4.3.1 Yönlendirici Agenttan Gelen Kontrol Mesajları

"Status"	Kaynak Agentın Kendisi	Null

Performative: "Status"

Receiver: Kaynak agentın kendisi

Content: Null

Kaynak agent bu mesaja karşılık olarak "performative" alanı yine "Status" olan ve "content" alanında da kriter bilgisi olan cevap mesajını yönlendirici agenta geri yollar.

4.3.2 Kaynak Agent İstek Mesajları

Kaynak agentlara iki tür istek mesajı gelebilir.

a- Sonuçların doğrudan kullanıcıya gönderilmesi

"Retrieve"	Kaynak Agentın kendisi	User Agent+Aranacak bilgi

Performative: "Retrieve"

Receiver: Kaynak agentın kendisi

Content: Bilgiyi isteyen user agentı ve istenen bilgi.

Mesajı alan kaynak agent "content" alanından isteği yapan user agent'ı çözer sonra da istenen bilgiyi veritabanında sorgular. Sonucu da doğrudan kullanıcı agent'a yollar. Sonuçları dödüren mesaj ise aşağıdaki gibidir.

"Retrieverply"	User Agent	Sonuçlar

b- Sonuçların verilen email adresine yollanması

"Retrieveviaemail"	Kaynak Agent	Email Adresi+Aranacak bilgi

Performative: "Retrieveviaemail"

Receiver: Kaynak agentın kendisi

Content: Sonuçların gönderileceği email adresi ve istenen bilgi.

Bu mesajı alan kaynak agent "content" alanından email adresini çözer ve sonuçları bu email adresine yollar

SMTP için JATLite'ın var olan desteği kullanılmamıştır. SMTP alt programı java ile yeniden yazılmıştır.

4.3.3 Kaynak Agent Veritabanları

Her kaynak agentın veritabanına erişim yöntemi farklı olabilir. Sorgulama yöntemleri de aynı olmak zorunda değildir. Ancak, beklenen mesaj formatlarına uyarak sonuçların yollanmasıdır. Projede veritabanları text dosyalar olarak kullanılmaktadır. Dosyada hem anahtar sözcükler hemde verinin kendisi tutulmaktadır. Eğer sorgulama yöntemi ve veriyi tutma yöntemi farklı olmayacaksa sisteme yeni bir kaynak agentın eklenmesi çok kolaydır. Sadece Konfigürasyon dosyasının değiştirilmesi yeterlidir.Konfigürasyon dosyası içinde kaynak agentın

kendi adresi, yönlendirici adresi, JATLite mesaj yönlendiricisi adresi, veri dosyası, email sunucusunun adresi yer almaktadır.

<Anahtar sözcükler

>
<Anahter sözcükler

>
<Anahtar sözcükler

>

Örnek bir veri dosyası yukarıdaki gibidir. Kaynak agent veri dosyasından gelen isteği Anahtar sözcükler ile karşılaştırır ve eğer anahtar sözcükler içinde bulursa bir sonraki anahtar değerine kadar olan bütün veriyi kullanıcı agenta yollar.

4.4 Yedek Agent

Sistemde Yönlendirici agentın düzgün bir biçimde çalışamaması ihtimaline karşın yedek agent görev yapar. Yedek agent sürekli olarak Yönlendirici agentın çalışıp çalışmadığını kontrol eder ve çalışmadığını anlarsa aynı özelliklere sahip yönlendirici agentı tekrar yaratır. Yedek agent sadece Yönlendirici agent ile ilgilenir. Java uygulaması olarak ağdaki herhangi bir makinede çalışabilir.

Problem durumunda yaratılan yeni yönlendirici agent kontrol mesajları ile yeniden sistem hakkında bilgi toplar ve birkaç dakika içerisinde bir önceki yönlendirici agentın sahip olduğu bütün bilgileri tutmaya başlar. Yönlendirici agent ve yedek agentın farklı makinelerde çalışması durumunda ilk yönlendirici agentın ileriki tarihte yapacağı sorgulamaları tuttuğu veritabanı yeni yaratılan yönlendirici agent tarafından bilinmez. Ancak bu destek de yedek ve yönlendirici agent arasında bu bilgilerinde periyodik olarak aktarılması sağlanarak kolayca eklenebilir.

Tablo 4.7 Yedek Agent Mesaj Tablosu

Performative	Alıcı	İçerik	Açıklama
ListUsers	Jatlite Message Router	null	Yönlendirici Agentın çalışıp çalışmadığının bilgisi Jatlite routerdan elde edilir.

4.5 Konfigürasyon Dosyaları

Kullanıcı dışında tüm agentlar ilk çalıştıklarında konfigürasyon bilgisini bir dosyadan okurlar. Bu dosya içinde agentın ismi adresi, yönlendirici agentın ismi, JATLite mesaj yönlendiricisinin adresi,kaynak agentlar için veritabanı dosyaları ve email sunucu adresi, maksimum zaman aşımı süresi gibi bilgiler tutulmaktadır. Aşağıda konfigürasyon dosya yapıları gösterilmiştir.

Yönlendirici agent konfigürasyon dosyası;

```
MyAddress
Supervisoragent,null,5557,MessageRouter,(agent-info
:password Supervisor)
RouterAddress
Router,nt.egebank.com.tr,4444,MessageRouter,(MessageRoute
r)
RouterRegistrarAddress
RouterRegistrar,nt.egebank.com.tr,4445,MessageRouter,(Mes
sageRouterRegistrar)
RegisterRequest
yes
MaxIdleTime
100
```

Kaynak Agent konfigürasyon dosyası;

```
MyAddress
dbaseagent1,null,5556,MessageRouter,(agent-info
:password dbaseagent1)
RouterAddress
Router,nt.egebank.com.tr,4444,MessageRouter,(MessageRoute
r)
RouterRegistrarAddress
RouterRegistrar,nt.egebank.com.tr,4445,MessageRouter,(Mes
sageRouterRegistrar)
RegisterRequest
yes
MaxIdleTime
```

```
10000
Category
fish
Supervisoragentname
Supervisoragent
Mailserver
10.0.1.83
Dbasefile
c:\jatlite\Dbaseagents\Dbaseagent1\fish.dat
```

Yedek agent konfigürasyon dosyası:

```
MyAddress
Backupagent,null,5551,MessageRouter,(agent-info
:password Backup)
RouterAddress
Router,nt.egebank.com.tr,4444,MessageRouter,(MessageRoute
r)
RouterRegistrarAddress
RouterRegistrar,nt.egebank.com.tr,4445,MessageRouter,(Mes
sageRouterRegistrar)
RegisterRequest
yes
MaxIdleTime
10000
Supervisoragentname
Supervisoragent
```

Kullanıcı agent applet olarak çalıştığı için konfigürasyon bilgilerinin bir kısmı kodun içerisinde bir kısmı ise kullanıcı tarafından girilen bilgilere göre oluşturulmuştur.

4.6 Esneklikler

Sistem geleneksel bilgi toplama sistemlerine göre hem kullanıcılara hemde programcıya daha fazla esneklik sağlar. Kullanıcı sonuçların kendisine gönderilme zamanını belirleyebilir. Ya sonuçları hemen görmeyi isteyebilir yada belirlediği gün ve saatte sorgunun yapılmasını sisteme bildirir. Sonuçlar ya kullanıcı ekranına yada belirttiği bir elektronik posta adresine gönderilebilir. Sonuçlar ulaşmadan kullanıcının sistemi terketmesi kullanıcının sonuçları sisteme tekrar bağlandığında görmesine engel değildir. Sonuçlar kullanıcı için sistemde saklanırlar.

Bilgi kriter denen başlıklar altında dağınık şekilde tutulur. Kullanıcı hangi kriterde bilgi tutan kaynaklar olduğunu ekranından görüntüleyebilir. Sisteme yeni kaynak agentlar girdiğinde bu listenin azenginleşeceğini de söyleyebiliriz. Eğer kullanıcı

herhangi bir kriter girmezse yönlendirici bu isteği sisteme genel amaçlı ve belirli bir kriter de bilgi tutacağını söylemeyen kaynak agentlara yollar. Bu dağınık yapının bilgi arama zamanı ve ağ trafiği için ideal olduğu da söylenebilir.

4.7 Sistem Karakteristikleri

Bilgi Toplama Sisteminin temel karakteristikleri aşağıdaki gibidir.

Otonom ve Akıllı Olma : Otonom bir agent çevresinde olup bitenleri öğrendikten sonra kendi başına davranabilme özelliğine sahiptir. Yönlendirme agentının bu özelliğe sahip olduğunu görüyoruz. Yönlendirme agentının , karar verme yeteneğine sahip olduğu ve kullanıcı yerine düşünebildiği için akıllı olduğunu da söyleyebiliriz. Sistemdeki agentlar davranışlarını değiştirdikçe yönlendirme agentıda davranışlarını değiştirir.

Öğrenme: Yönlendirici agentın neler yapabileceğini gördük. Bütün bunların öğrenme sonucu oluştuğu açıktır. Yönlendirici agent sisteme ilk kez girdiğinde hiç bir şey bilmemektedir ancak periyodik olarak sistemdeki agentlara gönderdiği kontrol mesajları sayesinde sistem hakkındaki en son güncel bilgilere sahip olur.

Haberleşme/Birlikte Çalışma : Sistem JATLite agent mesaj yönlendiricisi üzerine oturtulmuştur. Bütün agentlar bu yönlendirici üzerinden haberleşerek bütün sistemin belirli görevlerini yapan parçalarını oluştururlar.

Esneklik : Kullanıcıya sunulan esnekliklerin yanında sisteme yeni kaynak agentların eklenmesi ile yapının kolaylıkla büyüyebileceğide görülmektedir.

Hata Toleranslı : Yedek Agent Yönlendirisi Agent'ı sürekli kontrol ederek gerektiğinde onun görevini yerine getirebilir.

4.8 Geleneksel Bilgi Toplama Sistemlerine Göre Üstünlükleri

Sistemin geleneksel yapılara göre oldukça avantajı olduğu görülmektedir. Geleneksel sistemler merkezi bir veritabanı üzerinde çalışırken bu yapı dağıtık bir veritabanı mimarisine sahiptir. Kriter bilgisinin hem arama zamanı hem de ağ trafiğini

düşürdüğünü de söyleyebiliriz. Kullanıcı bağlantısının gerek hattan dolayı gerekse diğer nedenlerden dolayı kopması aynı arama isteğini tekrar yapmasına neden olmaz. Tekrar sisteme girdiğinde sonuçlar kendisine ulaşacaktır. Bu, sonuçları kullanıcı adına sistemde tutmasından kaynaklanmaktadır. Geleneksel sistemlerde bu desteği göremiyoruz. Kullanıcı sorgu zamanını gelecekteki bir tarihe kaydettirebilir. Herhangi bir sorgu isteğinde bulunduğu zaman sonuçları beklemeden başka sorgularda yapabilir. Bunun geleneksel sistemlerde mümkün olmadığı bilinmektedir.

4.9 Diğer Agent Tabanlı Veri Toplama Sistemleri

Agent desteği kullanılarak geliştirilmiş bir çok bilgi toplama sistemi görmekteyiz.Timothy Finin,Charles Nicholas ve Jam es Mayfield'ın "Software Agent for Information Retrieval System" [8] adlı makalesinde bir agent tabanlı bilgi sisteminin mimarisi çizilmiş,Sistem;User Agents,Provider Agents (back-ends) ve Broker Agents adı verilen çoklu agent yapılı bir ortam olarak gösterilmiştir.Bu sistemde User Agentlar kullanıcıların neyi sevip neyi sevmediğini de akıllarında tutan akıllı agentlardır. Yine bilgi toplama sistemi gibi istekleri yollayıp sonuçları gösterme yeteneğine sahiptir. Broker Agentlar,User Agentlar ve veritabanı işlemlerini gerçekleştiren back-end ler arasındaki haberleşmeyi sağlar.Tıpkı bilgi toplama sistemi gibi ilgili sorguları ilgili back-end lere yollar.

Brendan M. McLear , "Agent-Based Information Retrieval and Filtering on the Web" [9] adlı makalesinde web tabanlı arama işlemi için hareketli agentların kullanıldığı görülmektedir. Bu iş için kullanıcı arayüzü sağlayan(user interface) ,sorgu analizi uygulayan(query analyzer), sorguyu gerçekleştiren(query agents) ve analizi yapan (analyzer agents) agentlar kullanmıştır. Kullanıcı arabirimi kullanıcıdan sorgu girişi yapılmasını sağlar.Sorgu analizi yapan agent kullanıcının girdiği sorguları alıp kullanıcıya arama anahtarları konusunda tavsiyede bulunur. Bu agent bir kez aranacak anahtar kelimeyi belirlediğinde arama işlemi başlar. Hareketli bir agent belirli arama motorlarına taşınarak buradaki veri kaynaklarının anlayacağı sorgu şeklinde kullanıcı isteklerini iletir. Daha sonra arama motorundan dönen sonuçları inceler ve birden fazla olan aynı yanıtları filtreler, sonra da gelen her hedef veri kaynağı için bir analyzer agent yaratır ve bu agentlar bu kaynaklara giderek orada kullanıcının işine yarayacak bilginin olup olmadığını ararlar. Kullanıcı bir kez isteğini yaptıktan sonra internete bağlantılı kalmak zorunda değildir.Sonuçlar analyzer agent tarafından kullanıcıya email ile yollanabilir.

Bilgi toplama sitemi ile diğer sistemlerin benzer ve farklı yanları olduğu görülmektedir.Çoklu agent yapısı kullanmak genelde bu türden sistemlerin ortak

yanıdır. Görüldüğü gibi hareketli agentlardan oluşan bir yapı da kurulabilir ya da bilgi toplama sisteminde olduğu gibi mesaj tabanlı çalışan bir yapı da oluşturulabilir. Göze çarpan diğer nokta; bu türden sistemlerde agentların akıllı olmasına dikkat edildiğidir. Akıllı agentlar hem ağ trafiğinin azalmasını hemde arama zamanının kısalmasını sağlayacaklardır.

SONUÇ

Bu çalışma çoklu–agent yapısı üzerine kurulu bir bilgi toplama sisteminin mimari yapısı ve gerçekleştirme ayrıntılarını içermektedir. Sistemin geleneksel bilgi toplama sitemlerine göre, kullanıcı esnekliklerini de içeren üstünlükleri vardır. Sistem kolayca büyüyebilme ve hatalara karşı toleranslı olma özelliğine de sahiptir.

Yönlendirici agent akıllı olarak yaptığı yönlendirme ile ağ trafiğini azaltmaktadır. Veritabanı agentları sadece aramakla yükümlü oldukları soguları yaparlar. Çünkü kullanıcı agent kullanıcı isteklerini yönlendirici agent'a gönderdikten sonra yönlendirici agent daha önce sistemin yapısı hakkında öğrendiği bilgilerden bu istekleri hangi veritabanı agenta incelenmesi için yollayacağını bilir.

Sisteme kolayca eklenen veritabanı agentları, sistemin büyüyebilmesini sağlamaktadır. Yönlendirici agent sadece gelen istekleri yönlendirme ve sistem hakkında bilgi toplamaktan sorumlu olduğu için özellikle kaynakları fazlasıyla harcayacak bir proses gerçekleştirmemektedir, yani bir darboğaza sebep olmaz. Asıl darboğazın veritabanı sorgulamalarını gerçekleştiren veri tabanı agentlarda olması beklenir ki bu problemde ortamda bir çok veritabanı agent'ın bulunabilmesi ve her agentın sadece belirli tipteki sorgulara yanıt vermesi ile çözülmüştür. Ancak sistem birden fazla yönlendirici agentın yük dengeleme algoritması ile çalıştığı bir şekle ek bir çalışma ile sokulabilir.

Yedek agent sistem güvenliğini bir seviyeye kadar sağlar. Sistemin tam anlamıyla hatalara toleranslı olması için ortamda bir den çok yönlendirici agentın çalışıyor olması gerekmektedir.

Sistemin java ile yazılmış olması ortamdan bağımsız bir yapı oluşturmaktadır. Bilgi aramak isteyen kullanıcıların ihtiyacı olan tek şey bir taryıcıdır (browser).

JATLite oldukça esnek bir arabirim sağlayarak, temel agent şablonları yaratılmasına ve standart agent haberleşme dili olan KQML dilinin kullanımına olanak veren programlama arayüzünü sunmuştur.

KAYNAKLAR

[1] M.R. Genesereth and S.P.Ketchpel."Software Agents", Comm. ACM, Vol. 37, No. 7, July 1994, pp.48-53.

[2] S. Franklin and A. Graesser "Is it an Agent, or just a program? A taxonomy for Autonous Agents" Proc. Third International Workshop on Agent Theories, Architecures,and Languages, Springer-Verlag,1996.

[3] D. Milojicic, "Mobile Agent Applications", IEEE Concurrency, Vol. 7, No. 3, July-Sept. 1999, pp. 80-90.

[4] JATLite (Java Agent Template, Lite) - http://cdr.stanford.edu/ProcessLink/Papers/Jatl.html

[5] Agent Communications Languages - http://www-uk.hpl.hp.com/projects/viceroy/language.html , March 1996

[6] Knowledge Querying and Manipulation Language- KQML standard http://www.cs.umbc.edu/kqml/

[7] Extensions to the KQML standard - http://cdr.stanford.edu/ProcessLink/kqml-proposed.html

[8] T.Finin , C.Nicholas, J. Es Mayfiel ,"Software Agents for Information Retrieval" - http://www.csee.umbc.edu/abir/ad198/sld001.htm, March 1998

[9] Brendan M. McLear ,"Agent Based Information Retrieval and Filtering on the Web" - http://www.cs.colorado.edu/~summer/cs3202/final-reports/brendan.html

ÖZGEÇMİŞ

Suat Uğurlu 1974 yılında Bayburt ta doğmuştur. İlk,orta ve liseyi Eyüp Alibeyköy Lisesi'nde tamamladıktan sonra İstanbul Teknik Üniversitesi Kontrol ve Bilgisayar Mühendisliği bölümünü kazanmış ve bölümü başarıyla bitirmiştir. Lisans tezi olarak OOP destekli BBS uygulaması yazmıştır. Halen İ.T.Ü. Kontrol Bilgisayar bölümünde master'ı sürmektedir. Master tezi olarak yazdığı Agent Tabanlı Bilgi Toplama Sistemi 1999 yılında Advis 2000 sempozyumunda sunulmuş ve LNCS series de yayınlanmıştır.Özel sektörde çalışmakta olan Suat Uğurlu bir taraftan da akademik eğitimini sürdürmektedir.