GES: GÜVENLİ ETMEN SİSTEMİ

Suat UĞURLU[1] Nadia ERDOĞAN[2]

Bilgisayar Mühendisliği Bölümü
Elektrik-Elektronik Fakültesi
İstanbul Teknik Üniversitesi, 34469, Ayazağa, İstanbul
[1]e-posta: suat@suatugurlu.com [2] e-posta: erdoğan@cs.itu.edu.tr

Anahtar sözcükler:Hareketli Etmenler,GüvenliHareketli Etmen Sistemleri

ÖZET

Bilgisayar sistemlerinin gelişim sürecine baktığımızda, yazılım metodolojilerinin gün geçtikçe merkezi çalışma modellerinden, dağıtık çalışma modellerine doğru evrim geçirmekte olduğunu görmekteyiz. Mesajlaşma ile başlayan dağıtık çalışma yöntemleri, ağ ortamında farklı bilgisayarlardaki nesnelerin paylaşımına izin veren CORBA,DCOM, RMI gibi yapılar ile süregelmiştir. Bu gelişimin son aşamasında, nesnelerin ağ ortamında hareketliliğini sağlayan etmenleri görüyoruz. Hareketli etmen sistemleri, programcıya, birbirleriyle haberleşebilen ve ağ üzerindeki düğümler arasında hareket edebilen etmenler yazması için gerekli olan alt yapıyı sunarlar.

Kodun hareketliliğine dayanan hareketli etmen sistemlerinde, güvenlik düşünülmesi gereken önemli bir unsurdur. Hareketli olan kod, yaşam süresince bir çok risk altındadır. Bu bildiride, güvenli bir hareketli etmen mimarisinin tasarımı ve uygulama ayrıntıları incelenmektedir. Öncelikle hareketli etmen sistemlerindeki güvenlik açıklarından ve mevcut sistemlerin bu açıklara karşı geliştirdikleri koruma yöntemlerinin eksikliklerinden söz edilecektir. Daha sonra tasarlanan ve geliştirilen mimarinin ayrıntılarına değinilecek, mevcut sistemlerden farklılıkları ve artıları belirtilecektir. Bu bağlamda, etmen hareketliliği ve haberleşmesi için sağladığı alt yapı incelenecek, etmen veri ve kodunun korunması için uygulanan yöntemlerden söz edilecektir.

1. GİRİŞ

Tarihsel olarak dağıtık ortamlardaki birimlerin veri işlemek için kullandıkları haberleşme yöntemlerini incelediğimizde, ilk aşamada, "mesajlaşma"yı görmekteyiz. Eş seviyede olan birimler birbirlerine iskele, mesaj kutusu, ya da boru hattı gibi yapılar üzerinden mesajlar yollarlar. Haberleşme yöntemlerindeki gelişimin bir sonraki aşamasında, istemci-sunucu modeli yer alır. Birimler ana-uydu ilişkisi ile birbirlerine bağlıdırlar. İstemci isteklerini mesaj aracılığı ile sunucu tarafa iletir, sunucu taraf kendi üzerindeki veriyi işler ve sonucu istemciye yollar. Uzaktan yordam çağırma (RPC) bu modelde en çok kullanılan yöntemdir. Nesneye yönelik programlama yöntemlerinin gelişmesi ile birlikte uzaktan yordam çağırma yerine DCOM, CORBA, RMI[1] gibi yapılar ile ağda farklı düğümler üzerinde yer alan nesnelerin kullanılabilmesi sağlanmıştır.

Bir sonraki aşamanın, bu yazılım nesnelerinin hareketliliğinin sağlanması olduğu görülmektedir. Hareketli etmen mimarisinde yazılım parçaları (dolayısı ile nesneler) ağda hareket halindedirler. İstemci bir hareketli etmen yaratır ve bu etmen ağ üzerinde kaynakların olduğu sunucular üzerine kendini taşıyarak veriyi yerinde işler ve isterse istemci tarafa geri gelebilir. Etmen hareket sırasında çalışan koduyla birlikte o ana kadar topladığı verileri, yani durumunu da taşıma yeteneğine sahiptir.

Basit olarak bir etmen; otonom, birlikte çalışabilen, öğrenebilen, akıllı yazılım parçaları olarak adlandırılabilir[2]. Bunların yanında hareketlilik, uzun yaşam süresi ve çevresindeki değişikliklere tepki gösterebilme özellikleri de, bir etmeni herhangi bir yazılımdan ayıran özellikler olarak sayılabilir. Hareketlilik, bir etmen için taşınması zorunlu olan bir özellik değildir, ancak hareketlilik özelliğinin bir etmeni çok daha yetenekli hale getirdiği söylenebilir. Hareketli bir etmen, çalışmaya başladığı ortam ile sınırlı değildir. Kendini ağ üzerindeki bir konaktan herhangi bir konağa taşıma yeteneğine sahiptir. Bu niteliği, etmene, etkileşimli olarak çalışmak istediği nesne ile aynı ortamda olabilme özelliğini kazandırır. Hareketli etmen mimarisi dağıtık ortamlarda çalışmak için yeni bir felsefe sunmaktadır. Ağ yükünü, gecikmelerini ve protokol bağımlılıklarını azaltmaları, asenkron ve otonom çalışabilmeleri, heterojen ortamlarda rahatça çalışabilmeleri ve hata hoşgörülü olmaları hareketli etmenlerin en önemli yararlarıdır.

Bugüne kadar kod hareketliliğini sağlamak için çeşitli sistemler geliştirilmiştir. Telescript[3], Tacoma[4], AgentTcl[5], Aglets[6], Voyager[7], Concordia[8] ve Ajanta[9] bunların başlıcalarıdır. Sağladığı faydalardan dolayı günümüzde hareketli etmen sistemleri çoğunlukla Java dili ile yazılmaktadır.

Hareketli etmen sistemlerinde güvenlik düşünülmesi gereken önemli bir unsurdur çünkü hareketli olan kod ve veri yaşam süresi boyunca çeşitli riskler altındadır[10][11]. Güvenlik açıkları genel olarak etmenlerin etmenlere verebileceği zararlar ve etmenlerin konaklara ya da konakların etmenlere verebileceği zararlar olarak iki ana başlık altında incelenebilir. Etmen verisinin veya kodunun çalınması, etmenin çalışmasının durdurulması yada geciktirilmesi veya etmenin yanlış bilgilendirilerek görevinden saptırılması etmenin karşılaşabileceği önemli ataklardır. Konağın aşırı kaynak tüketimi sonucu hizmet kıtlığına uğraması, gizli bilgilerinin çalınması türünden ataklar ise konakların maruz kalabileceği temel ataklar olarak sıralanabilir.

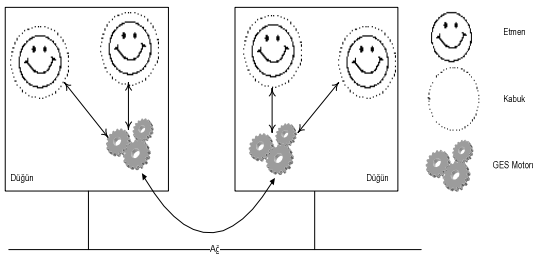
Güvenli bir hareketli etmen sistemi; hareket edebilme, haberleşebilme gibi temel etmen fonksiyonlarını yerine getirdiği gibi, etmen ve konağı korumaya dayalı fonksiyonları da programcıya ek yük getirmeden yerine getirmelidir. Yukarıda sayılan hareketli etmen sistemlerinin etmen ve konakların karşı karşıya olduğu güvenlik açıklarına karşı sundukları çözümlerin yetersiz olduğu gözlemlenmiştir. Ayrıca bir çok yazar tarafından çeşitli güvenlik çözümleri önerilmiş [12] [13] [14] [15] [16] [17] [18], ancak bunlar pratikte hareketli bir etmen sistemi içinde etkin bir şekilde uygulanamamıştır. **Güvenli Etmen Sistemi (GES)**, programcının hareketli etmenler oluşturmasına olanak sağlayan esnek bir ortam sağladığı gibi, güvenli mimarisi ile programcıya ek yük getirmeden hem konakları hem de etmenleri korur.

2. GÜVENLİ ETMEN SİSTEMİ (GES)

GES, programcıya hareketli etmenler oluşturması için gerekli fonksiyonları sunan, etmenlerin birbirleriyle haberleşmesini ve hareket etmeleri için gerekli alt yapıyı oluşturan sistemin adıdır. GES, hareketli bir etmen sisteminden beklenen aşağıdaki isterleri yerine getirmektedir:

- Mimari herhangi bir düğüm üzerinde birden çok etmenin çalışmasına olanak sağlar.
- Mimari etmenlerin bulunduğu düğümden saydam olarak birbirleriyle haberleşmesine olanak sağlar.
- Mimari etmenlerin dağıtık ortamdaki düğümler arasında hareket etmesine olanak sağlar.
- Mimari gerekli gördüğü anda bir etmenin çalışmasını sona erdirebilir.
- Mimari etmenlerin birbirlerine doğrudan erişimini engeller.
- Mimari etmenlerin yaratılma, aktive edilme, mesajlaşma, hareket etme, yok edilme gibi etkinliklerini kaydeder.
- Mimari her etmene tekil bir kimlik numarası verir.
- Bütün haberleşmeler şifreli olarak gerçeklenir. Şifreleme standardı olarak SSL kullanılır.
- Etmen kodu kara kutu güvenliği ismi verilen yöntemle korunur. Sistem içinde yerini alacak olan etmen, güvenlik yöneticisi tarafından şifrelenir ve sisteme bu şekilde alınır. Etmen, düğümler üzerinde veya hareket halinde iken şifrelidir ve etmen kodu bu aşamada çalıştırılabilir bir kod değildir. Bir düğüm üzerinde aktive edilecek olan etmen sadece belleğe yüklenme aşamasında kara kutu açılır ve etmen çalışmasına başlar. Etmen kodu sadece bellekte çalıştırılabilir kod parçası olarak görülmektedir.

GES, ortamdan bağımsız olabilmesi için Java dilinde gerçeklenmiştir. Bu nedenle, etmenler de Java dilinde geliştirilmelidir. Java dilinin seçilme nedenleri arasında, kod hareketliliğine olanak veren fonksiyonların gerçeklenmesine sağladığı destek, ortamdan bağımsız olması ve nesneye yönelik programlamaya büyük ölçüde destek veriyor olması sayılabilir.



Şekil 2.1-GES Etmen Mimarisi

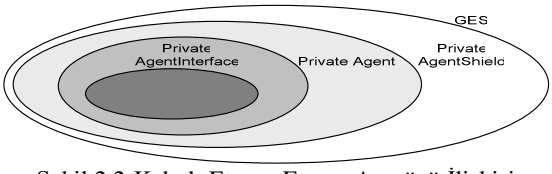
Şekil 2.1 de GES sistem mimarisi gösterilmiştir. Teorik olarak GES, üzerinde yer aldığı düğüm kaynakları yeterli olduğu sürece yeni bir etmenin oluşmasına izin verebilir. Tüm etmen aktiviteleri (yaratılma, sonlanma, aktif-pasif kılınma, mesajlaşma, transfer edilme gibi) GES denetiminde gerçekleşir. Etmenler birbirlerine doğrudan erişemezler. GES, yeni bir etmen yaratıldığında, etmeni, kendisini koruyan bir kabukla sarmalar. GES, etmenin çevresiyle olan etkileşimi için etmene bir arayüz sunar. Etmen bu arayüz aracılığı ile diğer etmenlere mesaj yollar, görünür duruma geçme veya ağdaki bir başka düğüme göç etme isteklerini kabuk aracılığı ile GES'e iletir. Etmen, diğer etmenlerin kendisine bir isim ile erişmesini istiyorsa bu ismi görünür (visible) duruma gelerek üzerinde çalıştığı GES'e duyurur. Böylece diğer etmenler bu ismi kullanarak etmen kimliğini öğrenebilir.

GES, şekil 2.2 de görüldüğü gibi her etmen için "`AgentShield`" sınıfından bir nesne yaratır. İlgili etmen de bu yeni yaratılan AgentShield sınıfındaki nesnenin özel (private) değişkenidir.

Etmenin çevresi ile etkileşimini sağlamak üzere etmene ait özel değişken olan "`AgentInterface`" sınıfından bir nesne mevcuttur. "`AgentShield`" nesnesi etmen arayüzü (AgentInterface) ile GES arasındaki iletişimi sağlar. Her etmenin ayrı bir AgentShield nesnesinin özel değişkeni yapılması ve etmenin çevresi ile etkileşiminin sadece AgentInterface nesnesi aracılığı ile mümkün olması, etmenlerin doğrudan birbirlerinin açık (public) değişkenlerine yada metodlarına erişmeleri engelleyen bir yapı oluşturur.

Mimari içerisinde seçilen herhangi bir GES, gözleyici (master browser) rolünü üstlenir. Diğer görevlerinin yanında gözleyici olarak da çalışan GES, sistemdeki tüm etmenlere ait yer bilgisini tutar. Böylece ismi ile bir etmene mesaj yollamak isteyen etmenin bu isteği, gözleyici yardımıyla, saydam olarak gerçeklenir.

Etmenler GES üzerinde etmen kimlikleri ile tanımlıdırlar. Etmenler diğer etmenler ile haberleşmek istediklerinde karşı etmenin kimlik bilgilerini kullanmak zorundadırlar. GES, yeni bir etmen yaratıldığında etmene ait kimlik bilgisini de oluşturur. Etmen kimlik bilgisi, etmeni tanımlayan 128 sekizli uzunluğunda rastgele oluşturulmuş bir katar, etmen görünür duruma gelmiş ise kendini görünür durumda tanımlayan bir isim ve etmenin o anda üzerinde bulunduğu GES sunucu adresinden oluşur.



Şekil 2.2-Kabuk-Etmen-Etmen Arayüzü İlişkisi

2.1. GES ETMENLERİ

GES mimarisi, etmeni geliştirecek programcıya esnek bir ortam sunacak şekilde tasarlanmıştır. Programcı hazır olan etmen şablonuna bağlı kalarak etmenin davranışlarını programlar. Etmen haberleşmesi ve hareketliliği için sistem tarafından sunulan fonksiyonları kullanması yeterli olur. Bir etmen, yaşam döngüsü içinde, *yaratılma*, *aktif olma*, *iletilme*, *pasif olma* ve *yok edilme* temel durumlarınının birinde olur. Aşağıda da görüldüğü gibi, etmen şablonu etmenin her farklı durumdaki çalışma düzenini yansıtacak şekilde metotlardan oluşur.

```
public class Main extends Agent{
  public void OnMessageArrive(){... }
  public void OnCreate(){ ... }
  public void OnActivate(){... }
  public void OnInactivate(){... }
  public void OnTransfer(){... }}
```

Programcının, her duruma ilişkin metodu, geliştirdiği etmene özel davranışları yansıtacak şekilde tasarlaması beklenir. Örneğin, etmen yaratıldığında yapılmasını istediklerini `OnCreate` metodunda, etmenin aktif haldeyken yürütmesi beklenen kodu `OnActivate` metodunda veya bir mesaj geldiğinde ne şekilde yanıtlayacağını `OnMessageArrive` metodunda tanımlaması gerekir.

3. ETMENLER ARASI HABERLEŞME

GES mimarisi, etmenler arası asenkron haberleşmeye olanak sağlayan yetenekli bir alt yapı içerir. Bir etmene gönderilen mesaja yanıtın belirlenen bir süre kadar beklenmesi, mesaj iletim isteklerinin sonuçlarının sorgulanabilmesi ve bir yanıtın, hazır olduğunda, kabul edilmesine izin veren hazır fonksiyonlar ile programcıya zengin bir çalışma ortamı sunulur.

Şekilde 3.1 iki etmen arasındaki mesaj aktarımının gerçekleşme aşamalarını göstermektedir. Her yeni etmenin yaratılma aşamasında, etmeni sarmalayan kabuk nesne içinde etmene ait bir mesaj giriş kuyruğu, bir mesaj çıkış kuyruğu ve bir de yanıt kuyruğu oluşturulur. Etmen aktif duruma geçtiğinde, giriş ve çıkış kuyruklarını sürekli gözleyen birer iplik canlandırılır. Giriş kuyruğunu izleyen iplik, yeni bir mesaj ulaştığında etmeni durumdan haberdar eder. Çıkış kuyruğunu izleyen iplik ise, yeni bir mesajın kuyruğa eklenmesi üzerine, GES motorunu uyarır.

Sistemdeki bir alıcı etmene mesaj iletmek isteyen gönderici etmen, bir çağrı ile bu isteğini bildirir. Çağrı istek mesajını, mesaj çıkış kuyruğuna yerleştirir. Bu noktadan sonra, etmen iletimin gerçeklenme ayrıntılarıyla ilgilenmez. Eğer bir yanıt bekliyorsa, mesaj yanıtı gelene kadar diğer işleriyle ilgilenebilir. Dilerse, haberleşme isteğinin sonucunu da sorgulayabilir. Bu arada, çıkış kuyruğunu gözleyen iplik, mesaj iletim isteğini GES motoruna aktarır. GES motoru alıcı etmenin üzerinde yer aldığı düğümde etkin olan GES motoruyla yaptığı işbirliği sonucu mesajın alıcı etmenin giriş kuyruğuna yerleştirilmesini sağlar. Giriş kuyruğunu gözleyen iplik, derhal alıcı etmeni uyarır. Etmen bu uyarıdan sonra, uygun gördüğü anda mesajı alma isteğinde bulunabilir. Yanıtlar da mesaj şeklinde iletilir, ancak gönderici kabuk, yanıt mesajlarını paketin sonuna özel bir işaret (ACK) koyarak, GES motorunun yanıt mesajını karşı tarafın yanıt kuyruğuna yerleştirmesini sağlar. Gönderici etmen yanıt kuyruğundan istediği zaman yanıtları çekebilir.

"`Message`" sınıfından bir nesne ile tanımlanan her mesajın bir ismi ve parametre listesi mevcuttur ve "`sendMessage`" çağrısı ile mesajı iletme isteğinde bulunulur. Bu çağrının geri dönüş değeri ile mesaj iletiminin durumu hakkında bilgi edinmek mümkündür. Örneğin ismi "Topla" olan ve iki sayının toplamını öğrenmek için gönderilecek bir mesaj aşağıdaki şekilde oluşturulur.

```
Message message = new
                Message("Topla","123","456")
```

Mesaj iletiminden önce gönderici etmen çevresi ile etkileşim için bir arayüz oluşturur.

```
AgentInterface agentinterface =
                      getAgentInterface()
```

Daha sonra hedef etmenin kimliğini öğrenir. Hedef etmen örneğin "Hesaplayici" ismiyle görünür duruma gelmiş olsun.

```
AgentIdentity hedefetmen  =
Agentinterface.getVisibleAgentIdentity("Hesap
layici")
```

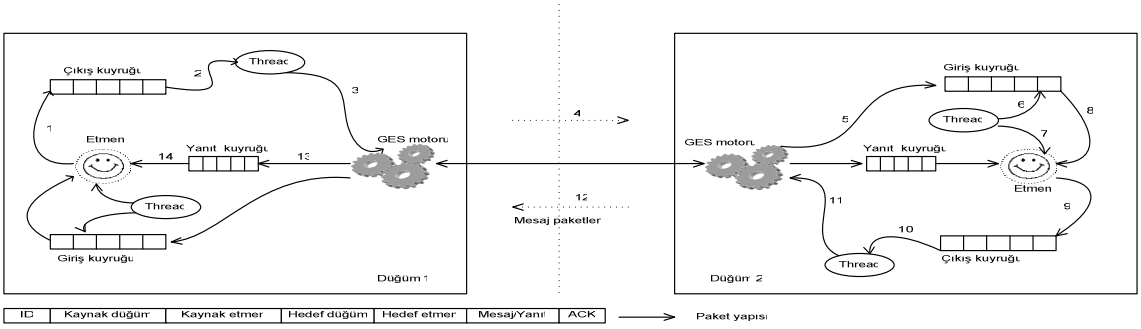
Son aşamada gönderici etmen mesajı iletir, mesaj iletim çağrısı "`Pidentifier`" sınıfından bir nesne geriye döndürür. Etmen, bu değer ile isteğinin sonucunu sorgular ve isterse yanıtını alır.

```
Pidentifier id= agentinterface.sendMessage
       (hedefetmen,message)
İf (!agentinterface.waitForReply(id,5000))
 //5 sn içinde yanıt gelmez ise bekleme.
System.out.println("Timeout.."+"\n");else
{
String result = (String)
agentinterface.getReply(id);
//yanıtı ekrana yaz.
System.out.println("123+456"+" = "+result+);}
```

Alıcı etmenin yapması gereken mesajı gelen paket içinden almak, parametreleri çözmek ve işlemi yapıp ilgili paketin yanıtını göndermektir. Alıcı etmen bu işlemleri ister "`OnMessageArrive`" metodu içinde isterse "`OnActivate`" metodu içerisinde gerçekleyebilir. Ancak "`OnActivate`" metodunda gerçekliyor ise, yeni bir mesajın varlığını "`waitForMessage`" çağrısı ile kendi kontrol etmelidir.

```
Packet packet=getAgentInterface().receive();
Message message = (Message)
                      packet.getObject();
Object[] parameters =
                message.getParameters();
String par1 = (String) parameters[0];
String par2 = (String) parameters[1];
int returnValue =
Integer.valueOf(par1).intValue() +
Integer.valueOf(par2).intValue();
getAgentInterface().sendReply(packet,
                String.valueOf(result));
```

Etmenler arası haberleşme oldukça basit ve esnek bir arayüz ile gerçekleşmektedir. Asenkron çalışma düzenine uygun olan bu yapı, etmeni bloke etmeyerek ölçeklenebilir olma özelliğine de sahiptir. Bütün mesajlaşmalar SSL ile şifreli olarak gerçeklenmektedir.



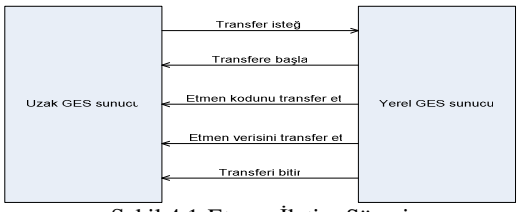
Şekil 3.1-Etmenler Arası Haberleşme

4. ETMEN GÖÇLERİ

GES sunucular birbirleriyle iş birliği yaparak gerektiğinde bir etmenin göç etme isteğini karşılarlar. Etmen hangi düğüm üzerindeki hangi GES sunucu üzerine taşınmak istediğini Şekil 4.1 deki gibi bir göç etme isteğiyle belirtir.

GES etmenleri kod ve veri olarak iki parçadan oluşurlar. Kod parçası etmenin davranışı belirleyen programlanmış kısmı, veri parçası ise etmenin durum bilgisini tutan değişkenlerini içeren kısmıdır. Bir etmen bir düğümden diğerine taşınırken şu aşamalardan geçer:

- Etmenin çalışması durdurulur.
- Etmenin o ana kadar edindiği durum bilgisi (değişkenler) ve kodu saklanır.
- Etmen kodu ve verisi uzak GES sunucuya iletilir.
- Etmen uzak GES sunucuda yaratılır ve aktive edilir.
- Etmen yerel GES sunucu üzerinden silinir.



Şekil 4.1-Etmen İletim Süreci

Göç etmek isteyen etmen "`Move`" çağrısı ile taşınma isteğinde bulunur. Daha sonra, "`Transfer Response`" sınıfından bir nesne yardımıyla isteğinin

gelişimi hakkında bilgi edinebilir. Etmen kodu ve verisi iletim sırasında şifrelenir. Böylece ağı izleyen üçüncü birimler bütün paketleri elde etseler bile bunları çözmek pratik olarak çok uzun zaman alacaktır.

```
  while (yapilacak_is_var) do_isler();
  transfersonuc=false;
  while((address!=null)||(!transfersonuc)){
  TransferResponse transferresult =
     getAgentInterface().Move(address);
  if(transferresult.FAILED)
address=newaddress();
else transfersonuc=true;}
```

Yukarıdaki kod parçası, yapılacak işi olduğu sürece düğüm üzerinde çalışan, sonra yeni düğümlere göç etmek isteyen bir etmene aittir. Etmen, bir çevrim içinde, herhangi bir düğüme göç etme isteği başarısız olduğu durumda yeni hedef adresleri denemektedir.

5. GES VE DİĞER HAREKETLİ ETMEN SİSTEMLERİ

Literatürde yer alan hareketli etmen sistemlerinin etmen iletimi ve haberleşmesi için genellikle hazır fonksiyonlar sunduklarını görmekteyiz. Ancak, özellikle güvenlik konusunda, bu sistemlerin büyük bir bölümü yetersiz kalmaktadır. Ayrıca, bir çoğunda, gizlilik ve bütünlük türünden ataklara karşı koymak programcıya ek yük ve sorumluluk getirmektedir. GES, etmeni sarmalayan kabuk modeliyle, etmeni, bulunduğu ortamdaki diğer etmenlere karşı korur. Ayrıca, çevresi ile etkileşimi için kolay kullanımlı bir arayüz sunar. GES'in mesajlaşma alt yapısı esnek ve ölçeklenebilir niteliktedir. Her GES sunucu, güvenlik yöneticisi olarak görev yapan bir GES sunucudan aldıkları sertifikalar yardımıyla birbirleri ile SSL üzerinden şifreli olarak haberleşirler. Ayrıca etmenlere ait kod ve veri düğümler üzerinde şifreli olarak tutulur. Etmen, aktive edilmek üzere disk üzerinden okunma aşamasında, şifreli kod çözülerek işletilebilir koda dönüştürülür ve belleğe yüklenir. Kod ve verinin tekrar diske yazılması, bellekte şifrelendikten sonra gerçeklenir. GES, bütün etmen aktivitelerini kaydeder. Bu izlerin analiz edilmesi ile normal koşullarda sezilmesi çok güç olan yanlış bilgilendirme gibi atakların da belirlenmesi mümkün olabilecektir. Mimariye, çalışmanın ileriki aşamalarında güvenlik polikalarının entegre edilmesi hedeflenmiştir. Programcının etmen göçleri, haberleşmesi ve kaynak kullanımı ile ilgili politikalar oluşturarak, bunları GES sunucular üzerine yüklemesi sağlanacaktır. Bu yapı, mimariye, güçlendirilmiş güvenlik unsurları yanısıra, dinamizm ve esneklik de kazandıracaktır. Mimari altyapı, bu tür bir işleyiş düzeni öngörülerek tasarlanmıştır.

6. SONUÇ

GES, programcıya, güvenli bir çalışma ortamı içinde, etmenler yaratma, etmenler arası haberleşme ve düğümler arası etmen göçleri için olanaklar sunan bir hareketli etmen sitemidir. Sarmalanmış etmen yapısı sayesinde çevresinden yalıtılmış etmen modeli güvenli çalışmanın temel gereksinimlerine yanıt vermektedir. Çalışmaya ilerde eklenecek olan politika tabanlı yapı sistemin güvenlik yeteneklerini daha da geliştirilecektir.

GES, gecikmelerin söz konusu olduğu ağlarda, paralel veri işlemede, insansız hava veya kara araçlarının yeniden programlanmasında, gerçek zamanda veri işlemesi gereken sistemlerde, bilgi toplama sistemlerinde ve hareketli etmen yapısının klasik çalışma modellerine göre daha uygun olduğu belirlenen bir çok ortamda kullanılabilir.

KAYNAKLAR

[1] Gopalan Suresh Raj "A Detailed Comparison of CORBA,DCOM and Java/RMI" - http://my.execpc.com/~gopalan/misc/compare.html

[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] Telescript, http://www.science.gmu.edu/~mchacko/Telescript/docs/telescript.html

[4] Tacoma, http://www.cs.uit.no/forskning/DOS/Tacoma/

[5] AgentTcl, http://agent.cs.dartmouth.edu/general/agenttcl.html

[6] Aglets, http://researchweb.watson.ibm.com/trl/projects/aglets/index_e.htm

[7] Voyager, http://www.recursionsw.com/products/voyager/voyager.asp

[8] Concordia, http://www.merl.com/projects/concordia/

[9] Ajanta, http://www.cs.umn.edu/Ajanta

[10] Christian F. Tschudin "Mobile Agent Security" Deparment of Computer Systems , Uppsala Univercity, Sweeden

[11] V.Varadharan and D.Foster ,"A Security Architecture for Mobile Agent Based Applications" World Wide Web:Internet and Web Information System, 6,93-122,2003

[12] F. B. Schneider, "Towards Fault-Tolerant and Security Agentry" Proc.11th Int. Wksp. Dist. Algorithims , 1997

[13] Young and M.Yung," Sliding encryption, A Cryptographic Tool for Mobile Agents" Proc. 4th Int. Wksp. Fast Software Encryption , FSE 97

[14] F. Hohl, "Protecting mobile agents with blackbox security" Proc. 1997 Wksp. Mobile Agents and Security , Univ. of Maryland , Oct 1997

[15] T. Sander, "On cryptographic protection of mobile agents" Proc. 1997 Wksp. Mobile Agents and Security , Oct 1997

[16] Uwe G. Wilhelm and Sebastian Staaman "Protecting the itinerary of Mobile Agents" Laboratoire de Systemes d'Exploitation, Switzerland,1998

[17] Vipin Swarup "Trust Appraisal and Secure Routing of Mobile Agents" The Mitre Corporation 1997

[18] Wayne Jansen,Tom Karygiannis "Mobile Agent Security" NIST Special Publication 800-19