İSTANBUL TEKNİK ÜNİVERSİTESİ « FEN BİLİMLERİ ENSTİTÜSÜ

GÜVENLİ BİR HAREKETLİ ETMEN SİSTEMİNİN TASARIMI VE GERÇEKLENMESİ

DOKTORA TEZİ

Y. Müh. Suat UĞURLU
(504002304)

Tezin Enstitüye Verildiği Tarih : 28 Eylül 2006
Tezin Savunulduğu Tarih : 12 Şubat 2007

Tez Danışmanı :	**Prof.Dr. Nadia ERDOĞAN**
Diğer Jüri Üyeleri	**Prof.Dr. Bülent ÖRENCİK (İ.T.Ü.)**
	Prof.Dr. Levent AKIN (B.Ü.)
	Yrd.Doç.Dr. Ali Gökhan YAVUZ (Y.T.Ü.)
	Yrd.Doç.Dr. Zehra ÇATALTEPE (İ.T.Ü.)

ŞUBAT 2007

ÖNSÖZ

Tez çalışmam süresince hiç bir koşulda desteklerini benden esirgemeyen ve yol gösteren sayın hocam Prof. Dr. Nadia Erdoğan'a, büyük bir sabırla tez çalışmamı başarıyla bitirmemi bekleyen nişanlım Funda Aylin Dikmen'e ve uzun süredir yeteri kadar zaman ayıramadığım ancak manevi desteklerini her zaman arkamda hissettiğim aileme en içten teşekkürlerimi sunarım.

Şubat 2007 Suat UĞURLU

İÇİNDEKİLER

KISALTMALAR

JVM	: Java Virtual Machine
RMI	: Remote Method Invocation
GES	: Güvenli Etmen Sunucusu
GY-GES	: Güvenlik Yönetici Güvenli Etmen Sunucusu
G-GES	: Gözleyici Güvenli Etmen Sunucusu
S-GES	: Standart Güvenli Etmen Sunucusu
SSL	: Secure Socket Layer
DES	: Data Encryption Standart
TCP	: Transport Control Protocol
IP	: Internet Protocol
XML	: Extended Markup Language
SECMAP	: Secure Mobile Agent Platform
HTTP	: Hyper Text Transfer Protocol
WWW	: World Wide Web
ASP	: Active Server Pages
PHP	: Personel Home Page
KQML	: Knowledge Query and Manipulation Language

TABLO LİSTESİ

ŞEKİL LİSTESİ

HAREKETLİ BİR GÜVENLİ ETMEN SİSTEMİNİN TASARIMI VE GERÇEKLENMESİ

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, uzaktan yordam çağırma ile devam etmiş ve en sonunda ağ ortamında farklı bilgisayarlardaki nesnelerin kullanılmasına izin veren yöntemler ile süregelmiştir. Bu gelişimin son aşamasında, nesnelerin ağ ortamında hareketliliğini sağlayan ve heterojen ortamlarda çalışabilen hareketli etmenleri görüyoruz. Hareketli etmen mimarisi istemci-sunucu çalışma modeline karşın dağıtık işlemeye farklı bir yaklaşım sunmaktadır.

Kabul görmüş genel bir yaklaşıma göre, bir etmen belirli özelliklere sahip küçük yazılım parçasıdır. Otonom olma, çevresine uyum sağlama, diğer etmenlerle birlikte haberleşip belirli bir amaç için işbirliği yapabilme, akıllı olma, çoğalma ve karar verme gibi özellikler bir etmeni herhangi bir yazılım parçasından ayıran özelliklerdir.

Hareketli olma, yani etmenin çalıştığı ortam ile sınırlı kalmayıp uzak noktalara kendini taşıyarak göç edebilme yeteneği, bir etmen için olmazsa olmaz bir özellik olmamasına rağmen, bu özelliğin bir etmeni özellikle dağıtık işleme açısından ne kadar yetenekli hale getirdiği kolayca görülebilir. Bu çalışma modeli, heterojen ortamlarda veri işleme için de yeni bir anlayış sunmaktadır.

Hareketli bir etmen bulunduğu ortam ile sınırlı değildir ve kendi kararı ile belirli bir anda ağ üzerindeki başka bir düğüm üzerine göç edebilir. Göç etme sırasında sadece çalıştırılabilir ve dinamik olan kod değil, etmenin o ana kadar edinmiş olduğu veri kümesi de etmen ile birlikte taşınır. Bu özellik, etmene içinde bulunduğu durumu uzak yerlerde de koruma olanağı sağlar.

Hareketli etmen sisteminin amacı ise, hareketli bir etmen için gerekli olan çalışma ortamını sağlamaktır. Hareketli bir etmenin yaratılması, çalıştırılması, göç ettirilmesi, çoğaltılması (cloning), mesajlaşması ve yok edilmesi gibi gerekli olan önemli işler

bu çalışma ortamı tarafından sağlanır. Hareketli bir etmenin ne kadar yetenekli olduğu, bu fonksiyonların ne kadar esnek bir şekilde sunulduğu ile yakından ilgilidir.

Kodun hareketliliğine dayanan hareketli etmen sistemlerinde, güvenlik düşünülmesi gereken önemli bir unsurdur; çünkü artık durağan bir yazılım parçası değil, kodunu ve verisini uzak düğümlere taşıyabilen yazılımlar söz konusudur. Hem kod hem de veri izlenme ve değiştirilme gibi tehlikeler ile karşı karşıyadır. Birbirleri ile haberleşebilen etmenlerin mesajlaşmaları sırasında da aynı tehlike söz konusudur. Tabii ki güvenlik riskleri bunlarla sınırlı değildir. Daha da önemlisi güvenlik riskleri ile karşı karşıya olan sadece etmenler de değildir. Etmenleri üzerlerinde çalıştıran düğümler de aynı ölçüde tehlikeler ile karşı karşıyadırlar. Bu tehlikeler ayrıntılı olarak ileriki bölümlerde işlenecektir.

Bugüne kadar hareketli etmen modelini destekleyen bazı sistemler geliştirilmiştir Sözkonusu sistemlerin genel olarak temel etmen fonksiyonlarını (Göç etme, mesajlaşma, çoğalma, vb.) gerçekledikleri görülmektedir ancak neredeyse tamamına yakınının yeterli ölçüde güvenlik fonksiyonları sunmadığı gözlenmektedir. Etmenler için gerekli güvenlik fonksiyonları sunan hareketli etmen sistemlerinde ise düğüm güvenliği için sunulan etkili çözümler bulunmamaktadır. Bu çözümler çoğu zaman programcıya yada geleneksel ağ ve düğüm güvenliği yöntemlerine bırakılmıştır.

Hareketli etmenler sürekli göç halinde bulundukları ve heterojen ortamlarda çalışabildikleri için çevresel koşullar da sık sık değişebilir. Bir hareketli etmen sistemi bu değişikliklere etmenlerin uyum sağlayabilmesi için gerekli fonksiyonları da sunmalıdır. Çevresel değişikliklere örnek olarak farklı donanım yada işletim sistemi platformlarında çalışan düğümlerin sisteme eklenmesi yada çıkarılması, bu düğümlerdeki kaynakların değişmesi ya da güvenlik gereksinimlerinin değişmesi sayılabilir. Bu gibi durumlarda etmen programcısı etmeni yeniden programlamak zorunda kalmamalı, bunun yerine hareketli etmen sistemi etmeni bu gibi durumlardan soyutlayacak gerekli alt yapıyı sunmalıdır ve daha da önemlisi alt yapı bu değişimlere çalışma anında uyum sağlayacak gerekli değişiklikleri yapabilme olanağı sağlamalıdır.

Mevcut hareketli etmen sistemlerindeki güvenlik çözümlerinin yetersizliğini göz önüne alan tez çalışması, güvenliği ilk planda değerlendiren, esnek, güvenli, sürekliliği ve performansı yüksek bir hareketli etmen sistemi olan Güvenli Etmen Sisteminin (GES) tasarlanması ve geliştirilmesini içermektedir. Sadece etmenlerin güvenliği değil, aynı zaman da düğümlerin de güvenliğine yönelik tüm gereksinimler mimari içinde gerçeklenmiştir.

GES, yüksek güvenlik sağlamasının yanında, kullanıcıya son derece esnek bir etmen programlama arayüzü de sunmaktadır. Şüphesiz bu özellikler hareketli etmen sisteminin kullanılabilirliğini arttıran en önemli etkenlerdir.

GES mimarisinin diğer mimarilerden farklı kılan en önemli özelliklerinden biri de güvenlik politikalarının kullanımıdır. Değişken çevresel şartlara uyum sağlama ve bir çok güvenlik gereksiniminin karşılanması, politikaların oluşturulması ve oluşturulan politikaların etmen ya da düğümlere atanması ile mümkündür. Çalışma anında değiştirilebilen politikalar hareketli etmen sistemine büyük ölçüde esneklik kazandırmaktadır. Temel etmen aktivitelerinin sisteme kaydedilmesi ve izlenebilmesi mimari tarafından sunulan diğer bir önemli özelliktir.

Süreklilik, iş paylaşımı yapmış etmenler için büyük önem taşır. Dolayısı ile hareketli bir etmen sistemi, sadece tek bir düğümün çalışıyor olduğu bir durumda bile sunmakta olduğu fonksiyonları yitirmemeli, düğüm çökmelerine karşı gerekiyorsa görev atamaları farklı düğümlere kaydırılabilmelidir. GES yüksek derecede süreklilik sağlayan mimarisi ile bu ihtiyaçlara karşılık verebilen yapıları barındırmaktadır.

Çalışmanın sağladığı bir diğer önemli özellik ise sistemin uzaktan bir tarayıcı yardımıyla yönetilebiliyor ve izlenebiliyor olmasıdır. Böylece düğümler üzerinde çalışan hareketli etmen sisteminin herhangi bir modülü, uzak düğümlerden yönetilebilmekte, sistemdeki tüm etmenler tek bir pencereden ve herhangi bir düğümden izlenebilmektedir.

GES, sağladığı etkin güvenlik özellikleri yanında, güçlü ve esnek yapısı nedeniyle de kullanılmaya değer yeni bir hareketli etmen sistemidir.

DESIGN AND IMPLEMENTATION OF A SECURE MOBILE AGENT SYSTEM

SUMMARY

When we look at the development of software over the past years, we observe that new methodologies built are based on distributed processing methods rather than central processing ones. Distributed processing, starting with message exchange, has continued its development with Remote Method Invocation (RPC) and then has utilized new techniques that provide access to remote objects on the network. At the last step of this developement process, we see mobile agents that can work in heterogeneous environments and support the mobility of objects over the network. Mobile agent architectures present a new and different approach than the cilent-server model to distributed processing.

According to an accepted definition, an agent is a small application with some special features. Being autonomous, capable of adapting itself to its environment, communicating with other agents for coordination or cooperation, intelligence, ability to clone itself and ability to make decisions are the features that can distinguish an agent from an ordinary software.

Even though mobility, ability to migrate from one host to another host, is not a required feature, agents with this ability have advantages especially in terms of distributed data processing. This model also presents a new aspect in heterogeneous environments for distributed processing.

A mobile agent is not restricted to the node where it is running and can migrate to anywhere on the network of its own accord. While moving from one host to another, not only the agent's executable code is transferred, but also data that the agent has collected or constructed are transferred as well. Thus, the agent can preserve its state even when it is mobile.

The execution framework necessary for a mobile agent to be active is provided by a mobile agent system. This framework simply provides the basic agent related tasks and functions such as agent creation, activation, migration, communication, cloning

and destruction. The competence and power of a mobile agent system depends on how flexibly these functions are implemented.

Even though using mobile agent technologies provides potential benefits to applications, an agent's ability to move introduces significant security risks. Mobile agents are under security threats during their life times. Since the code is mobile, it can be stolen or altered by a third party. The same danger is present for the messages agents send to each other and for the data that determines the agent state. Furthermore, not only the agents but also hosts are also under many security risks in mobile agent systems. The security threats will be analyzed in detailed later in the book.

Several mobile agent systems have been proposed and developed up to now. They all have their software agent specific features. Although most of them have enough features for mobile agents to communicate with each other and migrate to remote hosts, agent security related tasks are not available in most of them. Some provide limited security for agents, but do not provide any features to protect hosts. Most of these mobile agent systems leave the security to agent programmer or to traditional network security solutions.

Because mobile agents require a dynamic environment where conditions and requirements may change rapidly, it should be possible to carry out the necessary changes without reprogramming agents. An environmental change may be the addition of a new host to the system, running a different hardware or operating system than the others or a change in the security configuration of a host. In such cases, the mobile agent system should provide a transparent execution environment for the mobile agent and the agent reprogramming should not be necessary at all.

The scope of the thesis is the design and implementation of a new, secure, flexible, highly available and fast mobile agent system (SECMAP). The architecture of the system is especially designed for security purposes and, requirements not only for agent security but also for host security are also provided.

Besides ensuring security of both agents and hosts, SECMAP also presents a very flexible agent programming interface. Naturally, these features play an important role on the usability and popularity of the system.

SECMAP also presents a policy based management framework to protect system-level resources and agents against unauthorized access, as well. The policy architecture allows for dynamic manipulation of policy content, which results in an

adaptive and flexible framework that eliminates the reprogramming of the agents on changing conditions. Logging and monitoring of the basic agent activities are also possible.

Availability is very important for the collaborating agents. For this reason, a mobile agent system should be up and running even only one host in the system is active. When necessary the system should be able to transfer the duties of a dead host to another one in the system. SECMAP includes very powerful algorithms to ensure the availability of the overall system.

Another important feature is that the system and agents can be managed and monitored from a browser in the network. All agents present in the system can be monitored from a single window. Any module of the system can also be managed by a browser from remote hosts.

SECMAP is worth being used not only for the security features it presents for agents and hosts, but also for its very flexible and powerful agent programming interface.

1. GİRİŞ

Bu bölümde öncelikle hareketli etmen sistemlerinde güvenlik ihtiyaçlarının neler olduğu ve önemi açıklanacak, daha sonra önerilen güvenli hareketli etmen mimarisinin bu ihtiyaçlar için sunduğu çözümler üzerinde kısaca durulacaktır.

1.1 Problemin İncelenmesi

Bilgisayar sistemlerinin gelişim sürecine baktığımızda, yazılım metodolojilerinin gün geçtikçe merkezi çalışma modellerinden, dağıtık çalışma modellerine doğru evrim geçirmekte olduğunu görmekteyiz. Merkezi bir bilgisayardan istekte bulunan terminaller bu evrim sürecinin başlangıcında yer alırken, farklı bir bilgisayardan kodun yüklenmesiyle yerel olarak çalışmaya başlayan java appletlerini sürecin son aşamalarında görmekteyiz. Yakın zamanda yeni bir yazılım felsefesi olarak bir adım daha ileriye gidilmiş, daha geniş ölçekli platformlar oluşturabilmek ve bütünüyle dağıtık işlemeye elverişli bir çalışma modeli meydana getirmek için hareketli etmen sistemleri oluşturulmuş ve yazılım nesnelerinin hareketliliği sağlanmıştır.

Basitçe tanımlamak gerekirse, bir etmen, otonom, kendi başına karar verebilen, akıllı, çevresine uyum sağlayabilen, haberleşebilen ve işbirliği içinde çalışabilen yazılım parçaları olarak adlandırılmaktadır. Hareketli bir etmen ise bu özelliklerin yanısıra, bulunduğu ortamla sınırlı olmayıp, istediği an ağ üzerindeki başka bir düğüm üzerine kendini taşıyabilme yeteneğine sahiptir. Etmen bu taşınma sırasında sadece çalıştırılabilir kodunu değil, o ana kadar edindiği verileri de taşıyarak durumunu korur.

Hareketli bir etmen sistemi, hareketli etmenler için gerekli olan çalışma ortamını sunar. Öyle ki, etmenin, yaratılma, aktif olma, göç etme, çoğalma, mesajlaşma gibi ihtiyaçları bu alt yapı aracılığı ile gerçeklenir.

Kodun hareketliliğine dayanan hareketli etmen sistemlerinde, güvenlik düşünülmesi gereken önemli bir unsurdur. Hareketli olan kod, yaşam süresince bir çok risk altındadır[1] [2]. Hareketli olan etmenin kodu ve verisi taşınma sırasında çalınabilir veya değiştirilebilir. Etmenin çalıştırılması geciktirilebilir, yanlış bilgilendirme ile

amaçlanan sonuca erişmesi engellenebilir. Taşındığı ortamda yer alan diğer etmen ya da üçüncü parti yazılımların müdahalesi ile karşı karşıya kalabilir.

Benzer şekilde, düğümler de kötü niyetli etmenler tarafından kötüye kullanılabilirler. Düğüm kaynakları aşırı kullanım sonucu tüketilebilir, gizli verisi çalınabilir ya da düğüm kullanıcısı yanlış yönlendirilerek kandırılabilir.

Çok kısaca açıklanan ve hem etmenlerin hem de düğümlerin karşı karşıya oldukları güvenlik tehlikeleri bunlarla da sınırlı değildir. Bu nedenle, hareketli etmenlerin kullanımı ancak güvenlik ihtiyaçlarına çözümler üretebilen hareketli bir etmen sisteminin oluşturulması ile mümkün olabilir. Ayrıca değişen çevresel koşullara kısa zamanda uyum sağlayabilmek için de gerekli mekanizmalar hareketli etmen sistemleri içinde barındırılmalıdır. Diğer önemli bir ihtiyaç ise, etmen programcısına mümkün olduğunca esnek bir programlama ve yönetim arayüzünün sunuluyor olmasıdır.

1.2 Çözüm Önerisi: Yeni bir Güvenli Hareketli Etmen Sistemi

Hareketli etmen sistemlerinin güvenlik gereksinimleri, sadece, güvenlik problemlerinin tasarım aşamasında dikkate alınarak, çözümlerin oluşturulduğu bir sistem tarafından karşılanabilir. Mevcut hareketli etmen sistemleri incelendiğinde, genellikle güvenlik gereksinimlerini göz önüne alan tasarımlar yapılmadığı, bu gereksinimlerin, sonradan, yama yöntemleri ile karşılanmaya çalışıldığı görülür. Ayrıca sınırlı olan etmen güvenlik çözümleri düğümler için ise hiç düşünülmemiştir.

Çözüm; güvenlik problemine çözümlerin tasarım anında düşünülerek gerçeklendiği, hem etmenleri hem de düğümleri tehditlere karşı koruyacak gerekli fonksiyonları içeren, gerektiğinde dinamik yapısı nedeniyle çevresel değişimlere kolay uyum sağlayabilecek, kolay yönetilebilen, izlenebilen, hızlı yeni bir hareketli etmen sisteminin gerçeklenmesidir.

1.3 Tezin Amacı ve Özgün Katkıları

Bu tez çalışmasının amacı, hareketli etmen sistemlerindeki güvenlik gereksinimlerine cevap verebilen, esnek ve güçlü bir programlama ve yönetim arayüzü sunan yeni bir hareketli etmen sisteminin tasarlanması ve gerçeklenmesidir. Ana hedef, mevcut sistemlerin cevap veremediği güvenlik tehditleri için yeni çözümler üretmek, yan

hedefler ise oluşturulan sistemin mümkün olduğu kadar kullanılabilirliği arttıracak özelliklere sahip olmasıdır.

Tez kapsamında gerçeklenen hareketli etmen sistemi; Güvenli bir Etmen Sistemi (GES) olarak adlandırılmıştır ve JAVA dili ile gerçeklenmiştir. Platform bağımsız olan sistem hareketli bir etmen sisteminden beklenen temel güvenlik ihtiyaçlarının hepsine karşılık vermektedir. Aynı zamanda etmen programcısı için çok esnek bir arayüz sunmaktadır.

GES'i diğer sistemlerden ayıran temel özellikler aşağıda kısaca açıklanmıştır. Bu özellikler ve bunların dışındaki diğer özellikler ileriki sayfalarda ayrıntıları ile anlatılmıştır.

1.3.1 Sarmalanmış Etmen Modeli

Hareketli bir etmen için en önemli güvenlik tehditlerinden biri, çalıştığı ortamdaki kötü niyetli diğer etmen ya da yazılımlar tarafından gelebilecek saldırılardır. GES, bu tür tehlikelere karşı etmeni "Etmen Kabuğu" adı verilen bir yapı ile korur. Etmen kabuğu, etmene yönelebilecek doğrudan erişim girişimlerini, etmeni bulunduğu ortamda çevresinden soyutlayarak engeller. Kabuk nedeniyle dış dünya ile tüm ilişkileri kesilen etmen, ancak kabuk tarafından sunulan bir arayüz üzerinden çevresiyle etkileşim kurabilir. Bu arayüz aracılığı ile iletilen her türlü istek tüm güvenlik kontrollerinden geçirilerek yerine getirilir ve yetkisiz hiç bir aktiviteye izin verilmez.

1.3.2 Etmen Kod ve Verisinin Gizlenmesi

GES, bir etmene ait kod, veri ve tüm denetim bilgilerini etmenin yaşam döngüsü boyunca şifreli olarak tutar. Dolayısı ile dışarıdan bakıldığında bir etmen kara kutudan farksızdır. Bu şifreli dosyaları açmak ve etmeni sistem içinde çalışır duruma getirmek ise, sadece gerekli kontrollerden geçerek güvenilirliğini kanıtlamış GES sunucuları tarafından mümkündür. Etmen kod ve verisinin açık olduğu tek yer düğüm belleğidir. Göç etme sırasında da tüm haberleşmeler şifreli olarak gerçeklenir. GES ayrıca bir etmenin kod, veri ya da politikasındaki en ufak değişikliği farkedecek mekanizmalara da sahiptir.

1.3.3 Sertifika Tabanlı Güvenilir Sunucu Modeli

Sistemde yer alan her GES sunucu haberleşmek için birbirlerine güvenmek zorundadırlar. Bu güven, birbirlerinin sertifikalarını güvenilir olarak tanımlamaları

ile başlar. Ayrıca sertifikalar için yaratılan açık-gizli anahtar ikilileri sadece GES sunucuları arasındaki haberleşmelerin değil, etmenler arasındaki tüm haberleşmelerin de SSL (Secure Soket Layer) protokolü ile gerçeklenmesini sağlar. Bu güvenli iletişim etmenlerden saydam olarak gerçeklenir. Oysa mevcut sistemlere baktığımızda genelde etmen-etmen haberleşme güvenliği için etmen programcısına ek yük getirildiğini görüyoruz.

1.3.4 Güvenlik Politikaları

Sistemi diğer etmen sistemlerinden ayıran en önemli özelliklerden biri de güvenlik politikalarının kullanımıdır. Güvenlik politikaları hem etmenlere, hem de düğümlere atanabilmektedir. Bir etmenin, mesaj gönderebilme ve alabilme, göç edebilme, çoğalabilme gibi aktiviteleri politikalar ile belirlenmekte, hatta bu politikalar çalışma anında değiştirilebilmektedir. Benzer şekilde, bir düğümün kaynaklarının etmen tarafından kullanımı, etmenin sahibi veya kaynağı göz önüne alarak tanımlanan politikalar aracılığı ile sınırlandırılabilmektedir. Bu olanaklar sayesinde, GES, etmenlerin yeniden programlanmasına gerek kalmadan, çevresel koşullardaki değişimlere kolayca uyum sağlamalarını mümkün kılar.

1.3.5 Güvenilir ve Ölçeklenebilir Konumdan Saydam Mesajlaşma Altyapısı

GES mimarisinin sahip olduğu mesajlaşma altyapısı, sistemi bloke etmeyen ve farklı türden haberleşme gereksinimlerini destekleyen fonksiyonlar ile esnek bir haberleşme ortamı sunar. Etmenler ve GES sunucular arasındaki bütün mesajlaşmalar SSL protokolü ile gerçeklenir ve etmen programcısı bunun için ek kod geliştirmez. Etmenlere ait mesaj giriş-çıkış kuyruklarının oluşturulması, bir mesaj geldiğinde etmenin otomatik olarak bu durumdan haberdar edilmesi, iletilen her mesaj için güvenlik kontrollerinin yapılması, senkron-asenkron haberleşme olanakları mesajlaşma mimarisine oldukça ölçeklenebilir ve güvenilir bir yapı kazandırmaktadır. Etmenler arasındaki mesajlaşma bulundukları yerden bağımsız olarak gerçeklenir; diğer bir deyişle, bir etmene mesaj göndermek için onun ağ üzerinde adresinin bilinmesi gerekmez.

1.3.6 Gelişmiş Etmen Programlama Arayüzü

Doğrudan güvenlik ile ilgili olmamasına rağmen GES, etmen uygulamalarının geliştirilmesi için programcıya çok esnek fonsksiyonlar sunmaktadır. Önerilen etmen modeli bir şablon halinde sunulur ve programcıdan beklenen, etmenin yaşam döngüsü boyunca içinde yer alabileceği farklı durumlarda yürütmesi istenen adımları

bu şablon aracılığı ile programlamasıdır. Böylece programcı, sürekli durum değiştiren bir etmenin davranışlarını denetlemek için gerek duyulacak karmaşık algoritmalara gerek duymadan uygulamayı programlayabilir.

1.3.7 Kolay Yönetim

Yine güvenlikle doğrudan ilgisi olmasa da, kolay yönetim hareketli bir etmen sistemi için çok önemli bir özelliktir.Gerçeklenen etmen mimarisinde, tüm GES sistemini veya sistemde yer alan herhangi bir etmeni uzaktan tarayıcı aracılığı ile izlemek ve yönetmek mümkündür.

1.3.8 Yüksek Süreklilik

GES mimarisi sistemde tek bir düğüm kaldığı zaman bile düzgün şekilde çalışabilir durumda kalmayı sağlayacak mekanizmalar içermektedir. Sunucu görev dağılımları, çalışması aksayan bir düğümün belirlenmesiyle birlikte hemen yeniden yapılandırılır. Bu çalışmalar programcıya saydam olarak gerçeklenerek, etmenlere etkin olabilecekleri sürekli bir yürütme ortamı sağlanır.

1.3.9 Sonuç

Bu çalışmada güvenli bir hareketli etmen mimarisinin tasarımı ve uygulama ayrıntıları anlatılmış, mevcut sistemler ile olan benzer ve farklılıklar incelenmiştir. Mimarinin sağladığı güçlü güvenlik desteğinin ana dayanağı, güvenlik fonksiyonlarının mimariye tasarım aşamasında eklenmesidir. Ayrıca, mimari şifreleme tekniklerinden de çokça faydalanmaktadır.

GES, öncelikle sahip olduğu güvenlik özellikleri nedeniyle, mevcut hareketli etmen sistemlerinin cevap veremediği bir çok gereksinimi karşılayabilecek yetenektedir. Geliştirilmiş olan sistem, etmenlere esnek bir çalışma ortamı sağlayan fonksiyonlar sunmaktadır. Aynı zamanda, çalışma ortamının sürekliliğini sağlayan güçlü mekanizmalara da mevcuttur. Tümüyle dağıtık yapıya sahip olan mimari, sistemde tek bir birimin çalışır durumda kalması halinde bile, bütünüyle çökmeyen bir etmen sisteminin oluşturulmasına olanak sağlamaktadır. Esnek programlama arayüzü ve etmenlerin JAVA dili ile geliştirilebiliyor olması GES'in kullanılabilirliğini arttıracak diğer özelliklerdir.

Hareketli etmen mimarisi, dağıtık işlemeye ve hareketliliğe verdiği destekten dolayı JAVA dili ile gerçeklenmiştir. Etmenler de, JAVA dilinin bütün olanakları kullanılarak geliştirilebilmektedir.

Kitabın ikinci bölümünde, etmen, hareketli etmen ve hareketli etmen sistemleri tanıtılmış, geliştirilen yeni hareketli etmen sisteminin mevcut sistemler ile olan benzer ve farklı yanları incelenmiştir.

Üçünü bölümde hareketli etmen sistemlerindeki güvenlik problemleri incelenmiş ve bu problemler için geliştirilmiş çözümlerden söz edilmiştir.

Dördüncü bölümde Java dilinin hareketli etmen sisteminin geliştirilmesinde sık kullanılan önemli özellikleri örnekler kullanılarak tanıtılmıştır.

Beşinci bölümde "Güvenli Etmen Sistemi" (GES) kısaca tanıtılarak genel özellikleri verilmiştir.

Altıncı bölümde GES yazılım geliştirme arayüzü tanıtılarak, mimarinin programcıya sunduğu olanaklar ayrıntılarıyla incelenmiştir.

Yedinci bölümde GES mimarisi ayrıntılarıyla anlatılmıştır.

Sekizinci bölümde GES haberleşme alt yapısının detaylarına değinilmiştir.

Dokuzuncu bölümde GES mimarisinin güvenlik politikaları için verdiği destekten söz edilmiş, güvenlik politikalarının kullanımı incelenmiştir.

Onuncu bölümde sistem izlerinin tutulması için kullanılan alt yapı incelenmiştir.

Onbirinci bölümde GES sistem yönetimine değinilmiş, izleme ve yönetim için sunulan yapının mimarisinden söz edilmiştir.

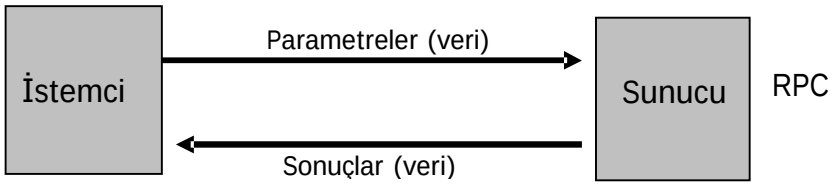
Onikinci bölümde GES performans analizi yapılarak çeşitki ölçüm sonuçları verilmiştir.

Onüçüncü bölümde örnek uygulamalar geliştirilerek etmen ve sistem davranışları incelenmiştir.

2. ETMEN VE HAREKETLİ ETMENLER

Dağıtık ortamlardaki birimlerin veri işlemek için kullandıkları haberleşme yöntemlerini incelediğimizde, ilk aşamada, "mesajlaşma" yı görmekteyiz. Bu yöntemde birimler eşit seviyelerde bulunurlar ve birbirlerine iskele (port), mesaj kutusu (mailbox), ya da boru hattı (pipe-line) gibi yapılar üzerinden mesajlar yollarlar. İki birim birbirlerinin adres alanlarına erişemedikleri için, mesaj içerisine verinin kendisi (işaretçisi değil) yerleştirilerek karşı tarafa iletilir. Mesajlaşma ayrıntıları ile uğraşmak, gelen mesaj paketleri içinden istenilen veriyi ayrıştırmak gibi işler çoğu zaman programcının yerine getirmesi gereken görevlerdir.

Haberleşme yöntemlerindeki gelişimin bir sonraki aşamasında, istemci-sunucu (client-server) modelini görmekteyiz. Birimler ana-uydu ilişkisi ile birbirlerine bağlıdırlar. Şekil 2.1 de görüldüğü gibi istemci isteklerini mesaj aracılığı ile sunucu tarafa iletir, sunucu taraf kendi üzerindeki veriyi işler ve sonucu istemciye yollar. Uzaktan yordam çağırma bu modelde en çok kullanılan yöntemdir. İstemci, kullanıcıya saydam olarak, yerel bir alt program çağrısına benzer şekilde bir uzak çağrı üretir, uzaktan yordam çağırma (remote procedure call:RPC) arayüzü bu isteği bir mesaj haline dönüştürerek sunucu tarafa iletir, sunucu tarafta bu isteğin hedeflediği yordam yerel olarak çalıştırılır ve sonuçları istemci tarafa yeniden yollanır.



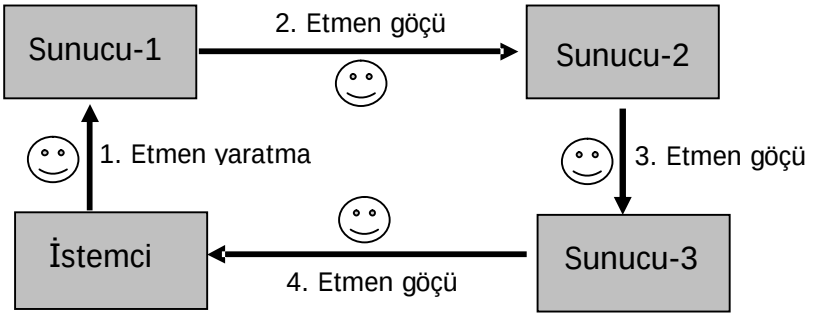
Şekil 2.1 : İstemci-Sunucu modeli (uzaktan yordam çağırma)

Bu yöntemde programcı mesajlaşma ayrıntıları ile ilgilenmez. RPC arayüzü bütün bu ayrıntıları programcıdan soyutlar. Uygulamalar arası haberleşme yöntemlerinin en popüler olanları arasında olan RPC, günümüzde hala tercih edilen bir yöntemdir.

Nesneye yönelik programlama yöntemlerinin gelişmesi ile birlikte uzaktan yordam çağırma yerine, ağdaki herhangi bir birim üzerinde yaratılmış olan nesneyi kullanma

fikri ortaya çıkmış ve bu amaca yönelik olarak CORBA[3] (common object request broker architecture) , DCOM[3] (Distributed component object model) ve RMI[1] (Remote method Invocation) gibi çözümler yaratılmıştır[3]. Bu yöntemlerde, ağda herhangi bir konakta yer alan nesneler, diğer konaklar tarafından kullanılabilmekte, hatta uzaktan nesne yaratılabilmektedir. (Bu özelliğe, Remote Method Activation adı verilmektedir ve RMI dan farklıdır). Temelde uzaktan yordam çağırmadan farklı olmayan bu yöntem, yerel makine dışındaki nesnelere erişiminin sağladığı programlama esnekliği nedeniyle daha fazla tercih edilir bir yöntem haline gelmiştir. CORBA gibi programlama modellerinde bu işlemlerin programlama dilinden ve platformdan bağımsız bir şekilde gerçekleştirilmesi hetorojen ortamlar için bu yöntemleri daha da kullanılır kılmıştır.

Dağıtık veri işleme yöntemlerinin son halkasında hareket eden yazılım nesnelerini görmekteyiz. Nesneye dayalı programlama dillerindeki gelişmelerle birlikte, tek başına bağımsız olarak çalışan bir program parçası bile nesne olarak görülmektedir. Dolayısı ile bu nesnenin ağ üzerindeki diğer birimler tarafından kullanılma olanağı olmakla birlikte nesne kendisinin karar verdiği durumlarda ağdaki herhangi bir düğüm üzerine de şekil 2.2 de görüldüğü gibi göç edebilir. Yazılım nesnelerine, dolayısı ile programlara hareketlilik özelliğini kazandıran bu sistemlere **Hareketli Etmen Sistemleri** adı verilmektedir. Etmen (şu an için nesne) hareket sırasında, çalışan koduyla birlikte, o ana kadar topladığı verileri, yani durumunu da (state) taşıma yeteneğine sahiptir.



Şekil 2.2 : Hareketli etmen modeli

Henüz üzerinde anlaşılmış bulunulan basit bir etmen tanımı olmamasına rağmen [4], etmeni belirli özelliklere sahip bir yazılım nesnesi olarak tanımlamak mümkündür. Sonuçta, her etmen aslında bir programdır, ancak her programı etmen olarak kabul etmek mümkün değildir.

Bir etmen, aşağıdaki özelliklere sahip olan bir yazılım nesnesi olarak düşünülebilir.

- **Otonom (autonomous):** Kendi başına hareket edebilme. Etmen adına etkin olduğu kullanıcıyı temsil etmeli ve onun adına kararlar alabilmelidir [28].

- **Birlikte çalışabilen, haberleşebilen (collaborative,communicative):** Amaca ulaşmak için diğer etmenlerle ortak çalışabilme ve haberleşebilme. Belirli bir görevi yerine getirmek için birden çok etmen kullanılabilir ve bu sırada etmenler işbirliği amacıyla birbirleri ile haberleşebilir.

- **Öğrenebilen (learning):** Önceki deneyimlerini kullanabilme. Etmen, tıpku bir kullanıcı gibi geçmiş deneyimlerini karar alma anında kullanabilmelidir.

- **Akıllı (intelligent):** Karar verme yeteneğine sahip olabilme.

Bunların yanında **hareketlilik** (mobility), **uzun yaşam süresi** ve **çevresindeki değişikliklere reaksiyon verebilme** (reactive) özellikleri de, bir etmeni herhangi bir yazılımdan ayıran özellikler olarak sayılabilir.

Hareketlilik, bir etmenin taşıması zorunlu olan bir özellik değildir. Ancak hareketlilik özelliğinin bir etmeni çok daha yetenekli hale getirdiği rahatça söylenebilir. Hareketli bir etmen, çalışmaya başladığı ortam ile sınırlı değildir. Kendini, ağ üzerindeki bir düğümden herhangi bir düğüme taşıma yeteneğine sahiptir. Bu hareketlilik etmene etkileşimli olarak çalışmak istediği nesne ile aynı ortamda olabilme özelliğini kazandırır. Hareketli etmen mimarisi dağıtık ortamlarda çalışmak için yeni bir felsefe sunmaktadır.

2.1 Hareketli Etmenlerin Sağladığı Yararlar

Hareketli etmenlerin sağladığı yararlar kısaca şu başlıklar altında incelenebilir.

- **Ağ yükünü azaltırlar**. Ağ ortamında çalışan etkileşimli prosesler genellikle ağ üzerinde yük oluşturacak şekilde veri alışverişinde bulunurlar. Hareketli bir etmen, çalışmak istediği ortama göç edip, gerek duyduğu verileri olduğu yerde, yerel olarak işleyerek sonucu hesaplayıp, geri dönebilir. Bu çalışma şekli veri akışının neden olacağı ağ yükünü oldukça azaltır.

- **Ağ gecikmelerini azaltırlar**. Üretim sektöründeki bir robot gibi gerçek zamanda çalışan bazı kritik sistemler çevrelerindeki değişikliklere mümkün olduğunca çabuk yanıt vermek zorundadırlar. Bu sistemlerin ağ üzerinden kontrolü önemli gecikmelere neden olabilir. Oysa merkezi bir denetçiden yollanan hareketli bir etmen, yerel olarak sistemi kontrol altında tutabilir.

- **Protokol bağımlılığını azaltırlar**. Dağıtık bir ortamda veri alışverişinde bulunurken, her birim gönderdiği ve aldığı verileri işleyebilecek bir protokolü de barındırmak zorundadır. Hareketli etmenler heterojen ortamlarda da farklı birimlere taşınarak kendi aralarındaki veri alışverişi için haberleşme kanalları oluşturabilirler.

- **Asenkron ve otonom çalışabilirler**. Geleneksel dağıtık sistem haberleşmesinde, haberleşecek birimler arasındaki ağ bağlantısı sürekli olmak zorundadır. Bağlantının kopması, bağlantının yeniden oluşturularak bütün işlemlerin yeniden yapılmasını gerektirir. Hareketli bir etmen, oluşturulduktan sonra, kendini oluşturan prosesten bağımsız olarak çalışabilir. Bu nedenle, bağlantının yokluğundan etkilenmeden çalışmasını tamamlar ve bağlantının gelmesiyle kendini oluşturan prosese geri dönebilir.

- **Dinamik olarak adaptasyon sağlayabilirler**. Hareketli etmenler çevrelerindeki değişiklikleri farkedip en iyi çözümü sağlayacak adaptasyonu gerçekleyebilirler.

- **Heterojen ortamlarda çalışırlar**. Dağıtık ortamlar hem yazılım hem de donanım açısından heterojen ortamlardır. Hareketli etmenler bilgisayar ve ulaşım katmanından bağımsız oldukları için heterojen ortamlarda çalışabilme özelliğine sahiptirler.

- **Hata hoşgörülüdürler**. Bir etmen üzerinde çalıştığı konakta herhangi bir problemle karşılaştığında kendini başka bir konağa taşıyabilir ve çalışmasına orada devam edebilir. Bu yetenek hata hoşgörülü ve daha dayanıklı sistemlerin oluşturulabilmesine olanak sağlar.

Hareketli etmenler sağladıkları yararlar nedeniyle, dağıtık veri işleme, yük paylaşımı, izleme (monitoring), arama (searching) ve yönetim (management) gibi bir çok alanda kullanılabilir.

2.2 Mevcut Hareketli Etmen Sistemleri ve GES ile Karşılaştırma

Bugüne kadar kod hareketliliğini sağlamak için çeşitli sistemler geliştirilmiştir. Bir çoğu, hareketlilik, mesajlaşma gibi temel etmen fonksiyonlarını desteklemektedir. İlk hareketli etmen sistemlerine baktığımızda programlama dili olarak TCL gibi metin (script) dilleri kullanıldığını görmekteyiz, ancak JAVA dilinin yaygınlaşması ile birlikte yeni etmen sistemlerinin hemen hemen hepsi bu dil kullanılarak geliştirilmiştir. Aşağıda en yaygın hareketli etmen sistemleri ve özellikleri kısaca anlatılmıştır.

- **Telescript:** General Magic firması tarafından geliştirilmiş nesne tabanlı ve güvenlik desteğine sahip olan yeni bir dil içeren sistemdir[5]. Telescript sunucular, hareket eden etmenlere servislerini sunar, etmenler adlandırma yöneticisini (DNS-Domain name system) kullanarak konaktan konağa hareket ederler. Telescript'in güvenlik açısından önemli desteği vardır. Her ortam ve etmenin kendine ait bir yetkisi mevcuttur ve bir etmen ortamı, kendisine taşınan bir etmenin yetkisini sorgulayabilir ve haklarını kısıtlayabilir. Etmenlere kota kontrolü uygulanabilir ve kotasını aşanlar sonlandırılabilir. Telescript ticari açıdan başarılı olamamıştır, çünkü öğrenilmesi gereken yepyeni bir dil sunmuştur. General Magic daha sonra aynı çerçeve üzerine kurulu java tabanlı Odyssey adı verilen sistem üzerinde çalışmıştır.

- **Tacoma:** Tromso (Norveç) ve Cornell üniversitesinin ortak bir çalışmasıdır[6]. Etmenler Tcl ile yazılmaktadır ve teknik olarak diğer dillerde yazılmış scriptleri de taşıyabilirler. Bir etmenin durum bilgisi daha sonra bir çantada toplanan kataloglarda tutulur. Bir etmen, programın, CODE ismi verilen özel bir kataloğa konmasıyla yaratılır. Daha sonra etmenin üzerinde bulunan konağın ismi de HOST adı verilen diğer bir özel kataloğa yazılır. Bu konağa göç, "meet" ilkeli ile gerçeklenir.Meet ilkelinin parametrelerinden biri, gelen kodu çalıştırma yeteneğine sahip olan etmen ismidir. CODE, HOST ve diğer uygulama bilgilerini içeren kataloglar bir çanta içinde karşı etmene yollanır. Senkron ve asenkton haberleşmeye destek verir. Hiç bir güvenlik mekanizması uygulanmamıştır. Hata hoşgörüsü için etmenleri takip eden "rear-guard" isimli özel etmenler kullanır.

- **Agent Tcl:** Dartmouth kolejinde geliştirilmiştir[7]. Sunucular arasında Tcl scriptlerin hareketine olanak sağlar. Değiştirilmiş Tcl yorumlayıcısı kullanılarak, görevcik seviyesinde çalışma anı durum bilgisinin taşınması sağlanır. Etmen taşındığı zaman, kodu, verisi ve durum bilgisi de birlikte taşınır. Göçler, yer bağımlı isimler kullanılarak gerçeklenir. Etmenler yer bağımlılığı olan DNS isimleri ile tanımlanırlar. Etmenlerin tehlikeli kod parçalarını çalıştırmasını engelleyen güvenli Tcl çalıştırma ortamı sağlanır. Sistem erişim kontrol listeleri tutarak, göç eden her etmenin bazı erişim kontrollerinden geçmesini sağlar. Gerektiğinde PGP (Pretty Good Privacy) kullanarak kimlik sınaması ve iletilen verinin güvenliği için şifreleme de kullanılabilmesine rağmen, şifreleme ilkelleri etmen programcısına açık değildir.

- **Aglets:** IBM tarafından geliştirilmiş java tabanlı bir sistemdir[8]. Etmenler agglet olarak adlandırılır. Etmen sunucuları arasındaki göç, ağdaki farklı konaklar arasında gerçeklenir. Sistemin önemli bir avantajı olay tabanlı kod yazmaya

olanak sağlamasıdır. Örneğin etmen göç ettiği anda "onArrival" metodu, etmeni programlamak için kullanılabilir. Hereketlilik Javanın "object serilization" özelliği kullanılarak gerçeklenir ve iplik seviyesinde çalışma durum bilgisi taşınamaz. Etmenler mesajlaşma ile haberleşebilirler. Etmenler, yer bağımlılığını kaldırmak ve dil seviyesinde güvenlik için proxy adı verilen nesneler kullanırlar. Etmenler birbirlerinin metodlarını kullanamazlar. Sistem henüz sınırlı güvenlik olanakları sunmaktadır.

- **Voyager:** Object Space tarafından geliştirilmiş java tabanlı bir sistemdir[9]. Uzaktan erişilebilir sanal sınıflar yaratılarak,nesnelere sanal referanslar kullanılarak yerden bağımsız olarak erişilmesi sağlanır. Etmenlere global tekil belirteçler verildiği gibi sembolik isimler de verilebilir. Sistemde etmenlere sembolik isimlerini yada tekil belirteçlerini kullanarak erişmeyi sağlayacak adlandırma yönetisici kullanılır. Alıcı, DNS ismi ve port numarası verilerek belirtilir. Çalışma durum bilgisi görevcik seviyesinde taşınmaz. Etmen haberleşmeleri sanal referanslara dayanan metod çağırma ile yapılır. Etmenler, senkron, tek yönlü çağrılar yapabilirler. Etmenler hiyerarşik olarak gruplara ayrıldıkları için çoklu gönderimler yapılabilir.

- **Concordia:** Mitsubishi Electric tarafından Java ile geliştirilmiştir[10]. Java object serilization mekanizması ile etmen göçü sağlanır, görevcik seviyesi çalışma durum bilgisi taşınmaz. Her etmen nesnesi etmenin izleyeceği yolları ve her konakta çalıştırılacak metodları belirten bir güzergah nesnesi ile ilişkilendirilmiştir. Concordia etmen haberleşmesi için oldukça gelişmiş haberleşme alt yapısı sunar. Etmen transferi için güvenlik mekanizmaları içerir. Sunucular erişim listeleri ile korunurlar. Her etmen bir kullanıcı ile ilişkilendirilmiştir ve kullanıcının şifresi tek yönlü hash fonksiyonundan geçirilerek etmen ile birlikte taşınır.

- **Ajanta:** Java tabanlı ve güvenlik mekanizmaları diğerlerine göre daha fazla gelişmiş olan bir sistemdir[11]. Etmen transferi şifreli olarak gerçeklenir. Yetkilendirme mekanizmaları kullanılır. Etmen durum ve kodunun değiştiğini farkedecek mekanizmalar geliştirilmiştir. Etmenler arasındaki haberleşme proxy nesnesi aracılığı ile yordam çağırma ile gerçeklenir.

- **Wishnu:** Hareketli etmen uygulamaları için önerilen yeni bir mimaridir. Önerilen mimarinin bir prototipinin de gerçeklendiği belirtilmektedir. Wishnu[12] her konakta yer alan ve konağa uğrayan her etmen için güvenlik kontrolü yapan SMC (Security Management Component) , güvenlik politikalarının üretildiği ve dağıtıldığı bir SMA (Security Management Authority), ve kendisiyle birlikte

güvenlik sınanmasında kullanılan pasaportları taşıyan SeA (Security Enhanced Agent) lardan oluşur. Mimari bir çok atağa karşı etkili olabilmesine rağmen hizmet kıtlığına neden olabilecek atakları sezmeye dayalı hiç bir fonksiyon sunmamaktadır. Mimari kimlik denetimi, yetkilendirme gibi hizmetleri geniş anlamda sunsa da etmen göçünün bir izini tutmamaktadır. Etmen göçünü izleme sistemde daha sonra toplanan bilgilerin analizi ile atak sezmede kullanılan etkili bir yöntemdir. Yine ağı sürekli dinleyip etmenin iletim sırasındaki deseninden kimliğini belirleyip, etmenin davranışını anlayıp buna göre bilgi toplamaya çalışan atak yöntemlerine karşı da bir koruma mekanizması yoktur.

- **S-agent**: Güvenli bir hareketli etmen sistemi için önerilen diğer bir mimaridir[13]. Kaynaklara ve etmenlere erişmek için erişim politikaları kullanılır. Ayrıca hizmet kıtlığına neden olacak atakları sezmeye yönelik fonksiyonları mevcuttur. Mimarinin en önemli eksikliği, mimari içine gömülü bir şifreleme yönetiminin olmamasıdır. Mimari gizli anahtarların oluşturulması, kullanılması gibi ayrıntıları programcıya bırakmaktadır. Mimarideki diğer bir eksiklik, etmenin ağda göç etmesi sırasında bazı bilgileri de kendi dinamik verisine ekleyerek taşımasıdır. Çok fazla göçün olduğu ortamlarda bu da probleme neden olabilir, etmenin büyüklüğünü arttırıp taşınmasını zorlaştırabilir.

- **JavaSeal**: Güvenlik amacıyla geliştirilmiş bir hareketli etmen sistemidir[14]. Etmen sistemi, hiyerarşik olarak birbirine bağlı, aralarında komşuluk ilişkileri ve haberleşme kanalları olan ve mühür (seal) adı verilen nesnelerden oluşur. Etmenlerin haberleşmesi ve hareketliliği için gerekli fonksiyorı mevcuttur. Etmenin kendi aktiviteleri sırasında güvenlik kontrolü yapılması yerine, etmenlerin etkileşimi sırasında güvenlik kontrolleri yapılarak performanstan kazanma hedeflenmiştir. Nesneler bölge (domain) ismi verilen yapılarla birbirlerinden ayrılmışlardır. Referans izleyicisi isimli birim tarafından kullanılan güvenlik politikalarıyla domainler arası nesne erişimleri denetim altına alınmıştır. Güvenlik politikaları ile, etmen yaratılması, yok edilmesi, etmenin içinde iplik (Thread) yaratılması, etmenler arası haberleşmeler denetim altına alınmıştır. Etmenlerin sonlanma metodlarında döngü türünden kodların konması engellenerek çok sınırlı servis kıtlığı ataklarına engel olunması hedeflenmiştir. Etmenler ve servisler birbirlerinden izole edilmişlerdir. Mesajlaşma ve hareketlilik için sunulan fonksiyonlar yeteri kadar esnek değildir. Etmenlerin izlenmesi ve dinamik olarak yönetilmesi içinde gerekli fonksiyonlar sunulmamıştır. Genellikle güvenlik, düğüm üzerinde etmeni izole ederek ve bölgeler arası güvenlik kontrolleri yaparak sağlanılmaya çalışılmıştır. Etmen kod ve verisinin gizliliği uygulama seviyesine bırakılmıştır.

- **JATLite**: JATLite[15] Stanford Üniversitesinde 1999-2000 yıllarında ilk sürümü tamamlanmış, daha çok etmen haberleşmesi için dizayn edilen bir etmen sistemidir. Aslında JATLite, hareketli etmen oluşturmak için herhangi bir destek vermez ancak hareketli etmenlerin JATLite mimarisini kullanımına izin verir. Etmenler önce "Etmen Mesaj Yönlendiricisi"ne kendilerini bir isimle kaydederler ve sonrada kendilerine bu isimleri kullanarak KQML standardında mesaj alıp verirler. Sistemin en önemli avantajı, mesaj gönderilen etmenin o an çalışıyor olmasının gerekmemesidir, sisteme bağlanan etmen kendisine daha önce gönderilen mesajları bağlandığında alabilir. Etmen Mesaj Yönlendiricisi değişen etmen adreslerinide güncellediği için yerden bağımsız bir mesajlaşma alt yapısı sunulmaktadır. Güvenlik için önerilen bir fonksiyonlar ise çok yetersizdir. Örneğin iletilen mesajların güvenliği programcıya bırakılmıştır.

Tez kapsamında geliştirilen GES hareketli etmen siteminin mevcut hareketli etmen sistemlerine göre farklı ve üstün özellikleri mevcuttur. Bu özellikler ileriki bölümlerde ayrıntıları ile anlatılacaktır. Diğer hareketli etmen sistemlerine göre dikkati çeken önemli özellikler şu şekilde sıralanabilir. Bu özellikler mevcut hareketli etmen sistemlerinde ya hiç bulunmamakta yada çok yetersiz kalmaktadır.

- **Yeni ve etkin bir etmen koruma modeli**: GES mimarisi etmeni, çevresindeki diğer etmenlerden koruyacak gerekli mekanizmalara sahiptir. Bu amaçla GES sunucusu, bir etmeni aktif etmeden önce "Etmen Kabuğu" (AgentShield) isimli bir nesne yaratır. "Sarmalanmış Etmen Modeli" adı verilen bu yöntemde her etmen bir etmen kabuğu tarafından korunur. Etmen, etmen kabuğu nesnesinin özel bir değişkenidir ve doğrudan etmene gelebilecek her türlü erişim engellenmiştir. Etmen dış ortam ile etmen kabuğunun etmene tahsis ettiği etmen arayüzünü kullanarak etkileşime geçebilir ve etmen arayüzündeki her aktivite güvenlik kontrolünden geçirilir. Diğer bazı hareketli etmen sistemlerinde "proxy" adı verilen etmene erişim için ara bir katman oluşturan farklı bir nesne kullanılmaktadır, ancak bu modelin, etmen kabuğunun diğer özellikleri ve sağladıkları ile karşılaştırıldığında yetersiz kaldığı açıkça görülmektedir.

- **Etmen gizliliği ve bütünlüğünün sağlanması**: GES, bir etmene ait kod ve veriyi etmen yaşam döngüsü boyunca şifreli olarak tutar. Ayrıca etmen kodu ve verisinin iletim sırasında veya sonradan değiştirilmesi durumlarını sezebilme yeteneğine sahiptir. Bunun için etmen programcısı ek programlama yapmaz.

- **Sertifika tabanlı hazır iletişim güvenliği**: GES mimarisi içinde, GES-GES ve etmen-etmen iletişimlerinin hepsi SSL protokolü ile şifreli olarak gerçeklenir. Bu, etmen programcısının iletişim güvenliği için ek programlama yapma

zorluğunu ortadan kaldırır. Sertifika tabanlı çalışma modeli vardır dolayısı ile sertifikasına güvenilmeyen sunucular ile iletişim mümkün değildir. Diğer hareketli etmen sistemlerin iletişim güvenliğini programcıya bıraktığını görmekteyiz.

- **Güçlü mesajlaşma altyapısı**: GES, çok esnek ve güvenli bir mesajlaşma alt yapısına sahiptir. Gönderici ve alıcılar için mesajlaşma kuyrukları kullanılır, sistemi bloke etmeyen olay tabanlı bir iletişim süreci mevcuttur. İletilen mesajlar SSL ile taşınır ve etmen programlama arayüzü diğer etmen sistemlerine göre çok esnektir.

- **Yüksek süreklilik**: Diğer hareketli etmen sistemlerinin hiç birinde bulunmayan diğer bir özellik, GES mimarisinde sistemin bütünüyle çökmesinin neredeyse olanaksız olmasıdır. Çok esnek ve sürekli bir sistem konfigürasyonu sistem yöneticileri tarafından gerçeklenebilir ve bu farklı mimariler programcıdan ve etmenlerin bütün aktivitelerinden saydam olarak gerçeklenir. Yani, sistem yöneticisi değişen çevresel şartlara göre yeni bir konfigürasyon yaptığında etmenler bunu sezemezler. "Gözleyici" ve "Güvenlik Yöneticisi" düzeninde çalışan GES sunucular arasındaki ortaklık ilişkileri sisteme yüksek süreklilik kazandırır.

- **Politika desteği**: Yine bir çok hareketli etmen sisteminde olmayan bir özellik ise hem etmenleri hem de düğümleri ilgilendiren güvenlik politikalarının destekleniyor olmasıdır. Bu konu ayrıntıları ile ele alınacaktır ancak güvenlik politikalarının hem etmen ve düğüm güvenliği hem de değişen çevresel şartlara uyum için çok gerekli olduğu unutulmamalıdır.

- **Düğümleri korumaya dayalı teknikler**: GES dışındaki hiç bir etmen sistemi (sınırlı olarak s-agent hariç) düğümleri korumaya dayalı güvenlik mekanizmaları içermemektedir. Düğüm kaynaklarının etmenler tarafından kullanımının sınırlandırılması güvenlik için çok gerekli olduğu durumlar olabilir.

- **Yönetilebilirlik-İzlenilebilirlik**: GES mimarisi tüm sistem yönetiminin ağ üzerinde herhangi bir tarayıcı ile yapılabilmesine olanak sağlamaktadır. Ayrıca etmen ve sistem aktivitelerinin izlenebilme olanağı da mevcuttur.

Aşağıdaki tablo, GES ve diğer bazı etmen sistemlerinin karşılaştırılmalı özelliklerini göstermektedir. Mevcut hareketli etmen sistemlerinin yeni versiyonları çıktıkça bu tabloda değişiklikler olması muhtemeldir.

Tablo 2.2.1: Hareketli etmen sistemlerinin karşılaştırılmalı özellikleri

Kategori	Özellik	GES	Voyager	Agglets	JATLite	JavaSeal
Genel						
	Hareketlilik	+	+	+	-	+
	Mesajlaşma	+	+	+	+	+
	Çoğalma (cloning)	+	+	+	-	-
	Web tabanlı yönetim	+	Konsol	-	-	-
	Güvenlik Politikaları	+	+	-	-	+
	Yüksek Süreklilik	+	-	-	-	-
	Ölçeklenebilirlik	+	Sınırlı	-	-	-
	Etmen kalıcılığı	+	+	+	-	+
Güvenlik						
	Etmen izolasyonu	+	+	-	-	+
	Kimlik denetimi	+	+	-	+	+
	Etmen gizliliği	+	-	-	-	+
	Kaynak kontrolü Ve koruması	+	Kısıtlı	Kısıtlı	-	+
	Erişim yetkilerinin atanabilmesi	+	-	+	-	+
	Güvenli etmen transferi	+	+	-	-	+
	Etmen koduna müdahelenin sezilmesi	+	-	-	-	-
Mesajlaşma						
	Konumdan Saydamlık	+	+	+	-	-
	Metod/Mesaj	Mesaj	Metod	Mesaj	Mesaj	Mesaj
	Etmen tarafından/ Sistem tarafından	Sistem	Sistem	Sistem	Sistem	Sistem
	ORB/RMI/Diğer	RMI	ORB	RMI	Diğer	Diğer
	Dinamik mesajlaşma	+	+	+	-	-
	Merkezi/Dağıtık	Dağıtık	Hiyerarşik, Dağıtık, Homojen	-	Merkezi	-
	Yayın desteği	+	+	-	-	-
	Senkron/Asenkron/ Randevu	S/A	S/A	S/A	A	Randevu
	Mesaj tamponlama (Etmene erişilemesse)	-	-	-	+	-
	Şifreli Mesajlaşma	+	-	-	-	-
	Mesaj Kuyruğu Yönetimi	+	+	+	+	-
	Ölçeklenebilir Altyapı	+	-	-	-	-

Kategori	Özellik	GES	S-agents	Wishnu	Concordia	Ajanta
Genel						
	Hareketlilik	+	+	+	+	+
	Mesajlaşma	+	+	+	+	+
	Çoğalma (cloning)	+	-	-	+	+
	Web tabanlı yönetim	+	-	Konsol	Konsol	-

	Güvenlik Politikaları	+	-	+	Erişim Listesi	+
	Yüksek Süreklilik	+	-	-	-	-
	Ölçeklenebilirlik	+	-	-	-	-
	Etmen kalıcılığı	+	Açık değil	Açık değil	+	+
Güvenlik						
	Etmen İzolasyonu	+	+	+	-	+
	Kimlik denetimi	+	+	+	+	+
	Etmen gizliliği	+	-	+	+	-
	Kaynak kontrolü Ve koruması	+	+	-	+	+
	Erişim yetkilerinin atanabilmesi	+	+	+	+	+
	Güvenli etmen transferi	+	-	+	+	+
	Etmen koduna mü-dahelenin sezilmesi	+	-	+	+	-
Mesajlaşma						
	Konumdan Saydamlık	+	-	-	+	+
	Metod/Mesaj	Mesaj	Mesaj	Mesaj	Mesaj	Mesaj
	Etmen tarafından/ Sistem tarafından	Sistem	Sistem	Sistem	Sistem	Sistem
	ORB/RMI/Diğer	RMI	Açık değil	Açık değil	RMI	RMI
	Dinamik mesajlaşma	+	-	-	+	+
	Merkezi/Dağıtık	Dağıtık	-	-	Hiyerarşik, Dağıtık, Heterojen	Dağıtık
	Yayın desteği	+	-	-	-	-
	Senkron/Asenkron/ Randevu	S/A	Açık değil	Açık değil	S/A	S
	Mesaj tamponlama (Etmene erişilemez ise)	-	-	-	-	-
	Şifreli Mesajlaşma	+	-	-	+	-
	Mesaj Kuyruğu Yönetimi	+	+	+	+	+
	Ölçeklenebilir altyapı	+	-	-	-	-

3. HAREKETLİ ETMEN SİSTEMLERİNDE GÜVENLİK

Kod hareketliliğinin, sağladığı yararların yanısıra, güvenlik açısından da bir o kadar dezavantajı vardır. Bir bilgisayar üzerinde çalışan herhangi bir kod parçası, sistem ve üzerinde bulunduğu kullanıcılar için güvenlik, gizlilik ve bütünlük açısından bir tehdittir[32]. Ağ ortamında hareket eden etmenler de sürekli tehdit altındadırlar. Ayrıca üzerine taşındıkları konaklar da etmenler kadar tehlikeli ataklara maruz kalabilirler[17]. Hareketli etmen sistemlerinde karşılaşılabilecek tehditleri, kaynağını göz önüne alarak, iki kategoride inceleyebiliriz.

- Konakların etmenler tarafından kötüye kullanılması.
- Etmenlerin konaklar tarafından veya diğer etmenler tarafından kötüye kullanılması.

Atak çeşitlerini ise genel olarak beş kategoride incelemek mümkündür.

- Zarar vermeye yönelik ataklar
- Hizmet kıtlığına neden olan ataklar (denial of service)
- Gizliliğin açığa çıkarılmasına neden olan ataklar
- Rahatsız edici ataklar
- Dezenformasyona dayalı ataklar

Zarar vermeye yönelik ataklar – Konak konfigürasyonunun bozulması veya donanım/ yazılımına zarar verilmesi, ya da bir etmenin görevine zarar verilmesi.

- **Konağa yönelik ataklar**: Konak üzerindeki etmen, konağı yeniden konfigüre ederek kaynaklarına zarar verebilir, diskte yer alan dosyalarını silebilir veya güvenlik politikasını değiştirebilir.
- **Etmene yönelik ataklar**: Konak, üzerindeki etmeni silerek ona zarar verebilir. Etmen o ana kadar elde ettiği bütün verileri kaybeder. Konak, etmene kaynak ayırmayarak görevini yapmasını engelleyebilir. Aynı konakta aynı çalıştırma katmanını kullanan etmenler birbirlerini etkileyebilir.

Hizmet kıtlığına neden olan ataklar – Konağın kaynaklarının ya da etmenin kaynak yada servislere olan erişiminin tüketilmesi.

- **Konağa yönelik ataklar**: Etmen üzerinde çalıştığı konağın kaynaklarını tüketebilir. Örneğin, sürekli bir ağ bağlantısı oluşturup bu bağlantı üzerinden büyük miktarda veriler yollayarak konağın ağ kaynaklarını tüketebilir.

- **Etmene yönelik ataklar**: Konak etmene kaynaklarını kullanma olanağı sağlamayabilir. Bu durumda, etmen çalışma olanağı bulamayıp, askıya alınabilir.

Gizliliğin açığa çıkarılmasına neden olan ataklar – Gizli bilgilere erişilmesi veya konaktan ya da etmenden veri çalınması.

- **Konağa yönelik ataklar**: Etmen üzerinde çalıştığı konaktan gizli verileri, örneğin da kullanıcı şifreleri gibi, elde edip bunları ağ üzerinden gönderebilir.

- **Etmene yönelik ataklar**: Etmen konak üzerinde çalışmak zorunda olduğundan her zaman için ikili kodu, varsa sahip olduğu anahtarlar, sertifikalar ya da şifreler gibi gizli verileri risk altındadır.

Rahatsız edici ataklar – Sürekli ataklarla rahatsız etme.

- **Konağa yönelik ataklar**: Kötü programlanmış bir etmen üzerinde çalıştığı konağı rahatsız edebilir, örneğin istenmeyen resimleri ekranda görüntüleyebilir, klavyenin kullanılmasını engelleyebilir, veya sürekli ses çıkartarak konakta çalışan kullanıcıyı rahatsız edebilir.

- **Etmene yönelik ataklar**: Hareketli bir etmen, çalışması geciktirilerek ya da güzergahı izlenip[33] etmen ve göndericisi hakkında bilgi toplamak suretiyle rahatsız edici ataklara maruz kalabilir.

Dezenformasyona dayalı ataklar – Kullanıcıların, konakların ya da etmenlerin yanlış bilgilendirme ile kötüye kullanılması.

- **Konağa yönelik ataklar**: Kötü amaçlı bir etmen ağ üzerinde gezinip kullanıcılardan sistem yöneticisini taklit ederek şifrelerini yada önemli bilgilerini isteyebilir.

- **Etmene yönelik ataklar**: Konaklar etmenleri yanlış bilgilendirerek ya da yanlış yönlendirerek etmenlerin istenmeyen konaklara göç etmesine neden olabilirler. Örneğin en iyi fiyatı arayan bir etmen, en iyi fiyatın belirli bir

konak tarafından sunulduğu konusunda yanlış bilgilendirilerek belirli bir konağa yönlendirilip yanlış sonuçlara ulaşmasına neden olunabilir.

Kısaca açıklanan bu atakların yanısıra, daha da karmaşık ve bir kaç partinin katılımı ile gerçekleşebilecek ataklardan da söz etmek mümkündür.

3.1 Hareketli Etmen Sistemlerinde Güvenliği Sağlama Teknikleri

Hareketli etmen sistemlerinde risk altında olanlar konaklar ve etmenler olduğu için, geliştirilen güvenlik teknikleri de konakları korumaya dayalı teknikler ve etmenleri korumaya dayalı teknikler olarak iki ana grupta incelenmektedir.

3.1.1 Konakları Korumaya Yönelik Teknikler

- **Kimlik kontrolü (Authentication)** : Etmenler sayısal olarak imzalanır ve bu imza etmenin ikili koduna eklenir. Konak, sayısal imzadan konağın ve göndericisinin kimliğini sınayabilir, sonuca göre çalışmasına izin verebilir veya vermeyebilir. Ancak, kimlik kontrolü[16] etmenin zararsız olduğunu kesin olarak garanti etmez.

- **Erişim seviyesi izleme ve kontrolü (Access Level Monitoring)**: Sistemde etmenlerin sahip olduğu erişim yetkilerini sınayan bir Güvenlik Yöneticisi yer alır. Yönetici güvenlik politikalarına başvurarak bir etmenin sahip olduğu, diski veya ağ bağlantısını kullanma gibi haklarını kontrol eder. Erişim yetkisine sahip olmadığı halde bir dosyayı okumaya çalışan etmen Güvenlik Yöneticisi tarafından engellenir. Kısacası, etmenlerin kaynaklar kullanımı erişim hakları aracılığı ile denetim altında tutulur[17]. Bir etmene kaynaklara erişim hakkı vermemenin anlamsız oluşundan ve kaynaklara erişim izni verilen etmenin de zararsız olduğundan emin olunamayacağından, bu yöntem de tek başına güvenliği sağlamakta yetersiz kalmaktadır.

- **Kod onaylanması (Code Verification)**: Etmenin sahip olduğu kod incelenerek geçerli bir kod parçası olup olmadığı sınanabilir[18]. Kod içerisinde zararlı komutlar sezen bir onaylama programı etmeni çalışmaya başlamadan etkisiz hale getirebilir. Örneğin, Javada bu birim “byte-code verifier” adı verilen onaylama programıdır. “Verifier”, konağa ulaşan Java etmenini inceler ve örneğin belleğin istenmeyen adreslerine yazma yönünde erişim sağlayan kod parçaları bulunursa, etmeni hiç çalıştırmaz. Yavaş olması nedeni ile kod onaylanması pek fazla kullanılmaz.

- **Sınırlama (Limitation)**: Konak, etmenin ancak belirli bir süre üzerinde çalışmasına izin verir veya etmenin sadece belirli sayıda konağa göç etmesine izin verilir. Sınırlama teknikleri etmenin çok fazla sayıda çoğalarak konaklara zarar vermesini engelleyebilen etkili yöntemlerdendir[17].

- **Aktivite izleme (Tracing)**: Etmenin bütün aktivitesi izlenir ve bunlar kaydedilir[17]. Daha sonra bunlar incelenerek bir atak sezilmesi halinde ilgili parti uyarılır.

Konak korumaya yönelik teknikler, hareketli etmen sistemi tarafından etmenlerin, konağa ulaşması, çalıştırılması ya da konaktan ayrılması anlarının bir ya da bir kaçında uygulanır.

3.1.2 Etmenleri Korumaya Yönelik Teknikler

Hareketli etmenleri korumaya yönelik teknikleri hata hoşgörüsü ve şifrelemeye dayalı olmak üzere iki ana grup altında inceleyebiliriz. Hata hoşgörüsüne dayalı koruma tekniklerinin amacı, etmeni öngörülemeyen ortamlarda daha dayanıklı hale getirmek, ağ bağlantılarının kopmasından etkilenmemesini sağlamak ve etmeni kötü niyetli konaklardan korumaktır. Şifrelemeye dayalı tekniklerin amacı ise etmenin sahip olduğu kodun veya bilginin gizlenerek zarar görmesini engellemek ya da etmenin elde ettiği bilgilerin çalınmasını zorlaştırmaktır.

3.1.2.1 Hata Hoşgörüsüne Dayalı Teknikler

- **Kopyalama ve oylama (Replication and Voting)**: Bu yöntem etmenin değişmeden alıcısına gittiğinden emin olmak için kullanılan bir tekniktir[19]. Ana fikir, bir işi yapmak için bir tek etmen değil de, etmenin birden çok kopyasının kullanılmasıdır. Eğer ortamda kötü niyetli konaklar var ise ve bunlar etmenin bazı kopyalarına zarar vermiş olsalar bile, sağlam olan diğer kopyalar aracılığı ile görev tamamlanır. Yapılacak iş bir kaç evreye bölünür ve bir sonraki evreye geçmenin koşulu etmenin kopyalarından bir alt kümenin değişmemiş olduğundan emin olunmasıdır. Etmenlerin değişmediğini belirlemenin bir yolu, etmenler tarafından bilinen gizli bir şifre olabilir. Kopyalama ve oylama yöntemi, etmenlerin problemsiz olarak çoğalabileceği ortamlarda uygun bir yol olmasına rağmen, çoğaltılan etmenlerin fazladan kaynak tüketmesi bakımından olumsuz bir etkiye de sahiptir.

- **Kalıcılık (Persistence)**: Bu teknik, konak çökmelerine karşı, çalışan bir etmenin kopyasının (kod ve durum bilgisinin) geçici olarak konakta saklanmasına

dayanır[20]. Her hangi bir problemden dolayı konak çalışamaz duruma düştüğün de etmen, durumunun saklandığı son hali ile disk üzerinde saklanır. Konak tekrar çalışır duruma geçtiği zaman, etmenin kopyası yeniden canlanır ve çalışmasına kaldığı yerden devam eder. Bu yöntem ilk bakışta geçerli bir yöntem gibi görünmesine karşın eğer konak uzun süre çalışamaz durumda kalır ve daha sonra canlanırsa, etrafta istenmeyen bir çok kopya görme ihtimali vardır, çünkü bu arada etmen sahibi etmenin kaybolduğunu düşünüp yeni bir kopya yaratmış olabilir yada etmenin kopya sayısındaki azalmadan dolayı diğer etmenler yeniden çoğalma isteğinde bulunmuş olabilirler. Zaman sınırlamaları uygulandığı taktirde kalıcılık metodu daha anlamlı bir koruma yöntemi olabilir.

- **Yeniden yönlendirme (Redirection)**: Bu yöntemde, hareketli bir etmen göç etmeden önce varmak istediği konağa doğru bir haberleşme kanalı açmaya çalışır (örneğin ethernet üzerinden) ve ulaşamazsa alternatif yollar dener (örneğin telsiz haberleşme yöntemleri). Problemli ağlarda etkin bir yöntem olarak kullanılabilir.

3.1.2.2 Şifrelemeye Dayalı Teknikler

- **Kayan Şifreleme (Sliding Encryption)**: Hareketli etmen herkese açık bir anahtar (public key) ile gizlemek istediği bilgiyi şifreler[21]. Anahtar herkese açık olduğu için bilinmesinde bir sakınca yoktur. Şifre sadece uygun özel anahtar (private key) ile çözülebilir. Bu şekilde kötü niyetli konaklar etmenin taşıdığı önemli bilgiyi de çalamazlar. Görüldüğü gibi, bu yöntem etmenlerin maruz kalabileceği ataklardan sadece gizliliğin açığa çıkarılmasına dayalı ataklara karşı etkili olabilir.

- **Güzergah karmaşıklaştırma (Trail Obscuring)**: Bu yöntemde, hareketli etmen ikili kodunu sürekli değiştirip izlediği güzergahı karmaşıklaştırmaya çalışır; böylece izini sürmek isteyen ve ne yapmak istediğini anlamaya çalışan kötü niyetli etmen ve konakların işi zorlaşır[21]. Kötü niyetli etmenler veya konaklar ikili kodu değişen etmenin aynı etmen olduğu farkedemezler.

- **Kod Saklama (Code Obfuscation)**: Bu yöndemde bütünüyle şifrelenmiş hareketli etmen bir çalıştırma katmanı (execution layer) ile birlikte karşı tarafa iletilir[22]. Bu çalıştırma katmanı şifreli olan etmeni şifreyi çözmeden çalıştırabilme yeteneğine sahiptir. Konak doğrudan etmenin kodu ve verisine erişemez. Bu yöntem gizliliğin açığa çıkarılması ve etmenin değiştirilerek amacından saptırılması gibi önemli ataklara karşı etkilidir, ancak konak açısından da bir o kadar risklidir. Kod kara bir kutuya dönüştürülür ve sadece karşı tarafa birlikte iletildiği çalıştırma katmanı bu kara kutuyu okuyup çalıştırma yeteneğine

sahiptir. Böylece konak ile etmenin bağlantısı kesilerek etmen korunur. Kodun kara kutuya dönüştürülme yöntemlerinde çok fazla fikir birliğine varılmamasına rağmen, şifreleme bu yöntemlerden biri olabilir. Çeşitli kod saklama yöntemleri üzerine de çalışmalar yapılmıştır. Saklama algoritmaları üzerinde henüz bir konsensüs oluşmaması ve bu algoritmaların, kodun çözülmesini sağlayacak alt sınırlarının belirlenememiş olması, bu yaklaşımın pratikte uygulanmasını zorlaştırmaktadır.

- **Şifreli Veri İşleme (Encrypted Data Manipulation)** : Bu yöntemde hareketli etmenin verisi şifreli olarak taşınır ve işlrmlrt bu veriler üzerinde şifreli durumdayken de yapılabilir[23]. Böylece etmenin kodu konak tarafından incelenebilmesine rağmen verisi incelenemez. Etmen tekrar göndericisine döndüğü zaman şifrelenmiş veriler açılabilir. Yöntem sadece gizliliğin açığa çıkarılması ataklarına karşı etkili olabilmektedir.

- **Durum Değerlendirme Fonksiyonları (State Appraisal Functions)** : Bu yöntem hareketli etmenin şifrelenmemiş dinamik verisinin değiştirilmediğini anlamak için kullanılır[24]. Normalde etmeni şifrelemek tek alıcılı sistemlerde kolay olabilir ancak etmen bir çok yere göç ediyor ve dinamik bağlamı sürekli değişiyorsa, şifreleme zorlaşır. Bu nedenle durum değerlendirme fonksiyonları kullanmak daha kolay olur. Gönderici hareketli etmenin ikili koduna ekleyip birlikte gönderdiği durum değerlendirme fonksiyonları çalışma anında kabul edilebilir parametrelerin varlığını sınar. Örneğin, bir malın en iyi fiyatını bulmakla görevli bir hareketli etmenin çalışma katmanı kötü niyetli bir konak ya da etmen tarafından değiştirilebilir ve etmen, bu malın bir tanesinin değil de yüz tanesinin fiyatını sorguluyor duruma getirilebilir. Bu şekilde sorgu yapan etmen yanlış cevaplar alıp doğru sonuca ulaşamayacaktır. Durum değerlendirme fonksiyonları bu gibi ataklara karşı etkili olabilmektedirler. Durum değerlendirme fonksiyonları etmenin içine şifreli yada sayısal imza kullanılarak konur ki kötü niyetli etmen ve konaklar tarafından bozulamasınlar. Kısaca etmen yada konağın bu fonksiyonlara verdikleri giriş çıkış parametreleri kabul edilebilir değerler ise etmenin saldırıya uğramamış olduğu varsayılır.

3.2 Güvenli Bir Hareketli Etmen Sisteminden Beklenenler

Görüldüğü gibi etmen ve konaklara karşı bir çok tür atak olasılığı, bu atakları engellemek için de bir o kadar koruma yöntemi mevcuttur. Güvenli bir hareketli etmen sistemi hem konağı hem de etmeni koruyacak fonksiyonları içermelidir. Akla gelebilecek bütün atak çeşitlerini engelleyebilecek bir sistem mevcut değildir;

dolayısı ile etmen sistemi güvenlikten fazlaca ödün vermemeli ancak hala kullanılabilir özellikte olmalıdır.

Hareketli etmen sistemlerinde bugüne kadar geliştirilen güvenlik çözümlerine baktığımızda, sunulan yöntemlerin oldukça kısıtlı alanda yeterli olduğu, yukarıda belirtilen bir çok ataktan sadece bir veya bir kaçına çözüm getirebildikleri görülmektedir. Bu yöntemlerin bazıları kod hareketliliğini sağlayan etmen sistemler ile birlikte sunulmuş ve pratik olarak gerçekleştirilmiş, bazıları bu sistemlerdeki eksiklikleri gidermek için sonradan eklenmiş, bazıları ise tamamen teori aşamasında kalıp, gerçeklenmemişlerdir. Bütün güvenlik tehlikelerine karşı bir mimari geliştirmek aslında oldukça da zor bir konudur çünkü bir çok atak sadece etmenlerin özelliklerinden dolayı ortaya çıkmış değildir. Ağ ortamının artılarından yararlanmaya başladığımız anda bir çok güvenlik açığıyla da karşı karşıya kaldığımız bir gerçektir. Ayrıca güvenlik çoğu zaman tek bir noktada çözülmesi gereken bir problem değildir. Hem ağ seviyesinde, hem de uygulama seviyesinde hatta fiziksel seviyede bile bir takım tedbirler almayı gerektirmektedir. Ancak hareketli bir etmen mimarisinin heterojen ve tehlikeleri öngörülemeyen ortamlarda hizmet vermesi gerekebileceğinden, mimarinin güvenlik için gerekli fonksiyonları sunması beklenmelidir.

Kodun hareketliliğini sağlayan sistemlerde amaç hareketli etmenler oluşturup bunların ağ ortamında dolaşmasını sağlamak olduğundan, güvenlik ilk adımda ayrıntılarıyla düşünülmemiştir. Bu yüzden de daha sonra yukarıda açıklanan yöntemler sunulmuştur. Güvenlik fonksiyonlarının, gerçeklenmiş bir sisteme sonradan yamalanması mümkün olmasına rağmen, mimarinin tasarım aşamasındayken güvenlik işlevlerinin düşünülmesi daha etkin ve sağlam bir yöntemdir.

Tasarlanan bir etmen mimarisinin güvenli sayılabilmesi için aşağıdaki fonksiyonları destekliyor olması gerekir.

- **Gizlilik (Confidentiality)**: Etmenin sahip olduğu ve başkaları tarafından görülmesinin sakıncalı olacağı bütün veriler gizli kalmalıdır. Gizliliği sağlamanın en etkin yolu ise şifreleme olarak karşımıza çıkmaktadır. Mimari, şifreleme için esnek bir yönetilebilirlik sunmalı (anahter yaratılması, dağıtılması, vb.), ayrıca programcıdan mümkün olduğunca saydam olmalıdır.

- **Bütünlük (Integrity)**: Etmen kodu, yetkisiz erişimlerden korunmalıdır. Güvenli bir etmen mimarisi yetkisiz erişimleri sezecek mekanizmalar içermelidir.

- **İzlenilebilirlik (Accountability)**: Etmenler ve konakların bütün aktivitelerinin izlenip, incelenmesine olanak sağlayan fonksiyonlar sunulmalıdır. Mimarideki her hareket, tek olarak tanımlanabilmeli, kimlik denetiminden geçmeli ve bir iz bırakmalıdır.

- **Süreklilik (Availability)**: Kaynak yönetimi, ölümcül kilitlenmeleri sezme, yazılım veya donanım hatalarının sezilmesi ve önlem alınması gibi fonksiyonların mimari tarafından sunuluyor olması gerekir.

Bir hareketli etmen sisteminin desteklemesi gereken, güvenlik dışında diğer özellikler de vardır. Örneğin, sistem etmen programcısına esnek bir programlama arayüzü sunmalıdır. Kolay yönetilebilir olmak da diğer bir gerekli özelliktir. Sonuçta güvenlik istenilen yüksek bir düzeye getirilebilir, ancak esneklik ve kolay kullanılabilirlik nitelikleri güvenlik ile ters orantılıdır. Çok fazla güvenlik kullanılırlığı azaltır ya da çok fazla esneklik güvenlikten ödün vermeyi gerektirebilir. Sistemde olması gereken tatmin edici bir düzeyde güvenlik sağlarken yine tatmin edici düzeyde esnek ve kullanılabilir olmaktır.

4. JAVA DİLİ VE ETMEN SİSTEMLERİNDE KULLANIMI

Günümüz etmen sistemlerinin tümü JAVA dili ile yazılmıştır çünkü dil, dağıtık işlemeye ve hareketliliğe verdiği desteğin yanında platformdan bağımsız oluşu ile de tercih edilmektedir. Doktora çalışması sonucu gerçeklenen hareketli etmen sistemi GES (Güvenli Etmen Sistemi) de JAVA dilinde yazılmıştır. Mimari ayrıntılarını incelemeden önce, JAVA'nın önemli destek sağlayan özelliklerini incelemek faydalı olacaktır. Sırasıyla aşağıdaki başlıklar ele alınacaktır.

- **Nesne Serileştirme** (Object Serialization)
- **Uzaktan Metod Çağırma** (Remote Method Invocation)
- **Yansıma** (Reflection)

4.1 Nesne Serileştirme

JAVA dili, çalışma zamanında yaratılan nesnelerin sürekliliğini sağlamak için nesnelerin disk ortamında saklanmasına olanak vermektedir. Bunun için nesnenin bütün yapısı bir sekizli akışına (byte stream) dönüştürülür ve diske yazılır. Nesne durağan ortamdan bu sekizli akışı okunarak tekrar elde edilebilmektedir. Nesnenin sekizli dizisi haline dönüştürülmesine nesne serileştirme adı verilmektedir. Burada kilit nokta nesnenin çalışma anında oluşan durum bilgisinin de serileştirilmesi, böylece nesneye kalıcılık sağlanmasıdır[25].

Nesnelerin serileştirilmesi JAVA'nın **ObjectOutputStream** ve akışın okunması ile nesnenin geri elde edilmesi de **ObjectInputStream** nesneleri ile yapılmaktadır. Bu iki akış nesnesi nesnelerin yazılacağı fiziksel yapıları (dosya, soket gibi) temsil eden akış nesneleri üzerinde yaratılır. Örneğin, nesne bir dosyaya yazılacaksa en alt seviyedeki akış nesnesi **FileOutputStream** olmalıdır.

Aşağıdaki örnek, günün tarih bilgisinin nasıl serileştirilerek diske yazıldığına basit bir örnektir.

```
1-    FileOutputStream out = new FileOutputStream("tarih");
2-    ObjectOutputStream s = new ObjectOutputStream(out);
```

```
3-    s.writeObject("Bugun");
4-    s.writeObject(new Date());
5-    s.flush();
```

Birinci satırda serileştirelecek nesnenin yazılacağı dosya ismi belirlenmiştir (tarih) ve "out" ismi ile dosyayı temsil eden bir dosya yazma akış nesnesi yaratılmıştır. İkinci satırda "s" isimli nesne yazma akışı tanımlanmış ve out nesnesi ile ilişkilendirilmiştir. 3. satırda nesne yazma akışının "writeObject" metodu ile "Bugun" değerli katar tipinden bir nesne, 4. satırda da "Date" tipinden olan ve değeri şu anki tarih bilgisini tutan yeni bir nesne akışa yazılmıştır. 5. ve son satırda ise akışa yazılan nesnelerin fiziksel olarak "tarih" isimli dosyaya yazılmaları sağlanmıştır.

Nesne serileştirme sırasında, eğer nesne diğer nesnelere referans tutuyorsa o nesneler de serileştirme işlemine tabii tutulur. "Integer", "String" gibi temel ilkel tiplerinden olan nesnelerin serileştirilmesi JAVA tarafından otomatik olarak desteklenmektedir, daha sonra göreceğimiz gibi yaratılan daha üst seviyedeki bir nesnenin serileştirilebilmesi için özel derleyici direktifleri kullanılmalıdır. Eğer nesne serileştirilebilir bir nesne değilse, serileştirme çağrıları "NotSerializableException" hatası üretirler.

Nesneler serileştirilip diske yazıldıktan sonra benzer yöntemle dosya okuma akışı ve nesne okuma akışı nesneleri ile tekrar oluşturulabilirler. Aşağıdaki kod parçası ile yukarıda serileştirilen nesne tekrar elde edilmiştir.

```
1- FileInputStream in = new FileInputStream("tarih");
2- ObjectInputStream s = new ObjectInputStream(in);
3- String bugun = (String)s.readObject();
4- Date date = (Date)s.readObject();
```

Benzer şekilde nesne okuma akışı (ObjectInputStream) fiziksel seviyeyi temsil eden başka bir nesne üzerinde yaratılmalıdır. Burada "s" isimli nesne okuma akışı "in" isimli dosya okuma akışı üzerinde tanımlanmıştır. 3. satırda nesne okuma akışının "readObject" metodu ile serileştirilmiş nesne "bugun" isimli Katar (String) türünden bir nesneye dönüştürülmüş, 4. satırda ise "date" isimli ve "Date" türünden olan nesne serileştirilmiş nesne üzerinden tekrar yaratılmıştır. "readObject" metodu serileştirilmiş nesnenin referans nesnelerini de otomatik olarak okuyarak nesnenin durum ve ilişki bilgisini son haliyle elde eder.

Yaratılan özel tanımlı bir tür sınıfından herhangi bir nesnenin serileştirilebilmesi için sınıf tanımlanırken “Serializable” direktifi kullanılmalıdır. Aşağıdaki kod parçası ile serileştirilebilir yeni bir sınıf oluşturulmuştur.

```
public class MySerializableClass implements Serializable {
    ...
}
```

Bu sınıftan bir nesnenin serileştirilmesi ile aşağıda bilgiler otomatik olarak serileştirilmiş nesne ile birlikte saklanır.

- **Nesnenin sınıfı**
- **Sınıfın imzası**
- **non-transient ve non-static olmayan bütün üyeler**

Eğer nesne içinde serileştirilmemesi istenen üyeler varsa bunlar “transient” olarak tanımlanmalıdır. Aynı şekilde “static” değişkenler de serileştirilemezler.

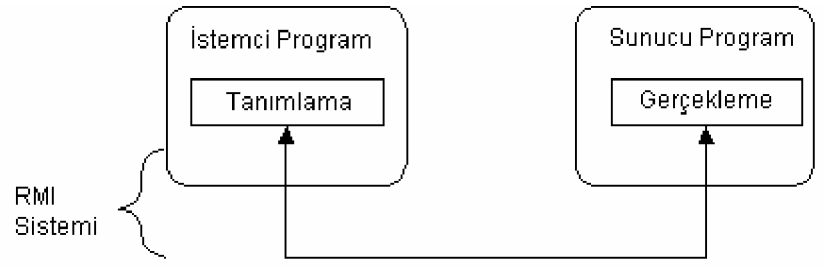
Hareketli etmen sisteminde nesne serileştirilme özelliği sıkça kullanılmıştır. Etmenin bütün durum bilgisi bu sayede saklanmaktadır. Etmen pasif duruma geçerken serileştirilip diske kaydedilmekte, aktif duruma geçerken de akış diskten okunarak yeniden bellekte yaratılmaktadır. Tabii ki mimarideki yapı bu kadar basit değildir; daha sonra da görüleceği gibi güvenlik nedeniyle serileştirilme işlemi mimaride özelleştirilmiş, sekizli dizisine çevrilen her veri diske şifreli olarak yazılmıştır.

4.2 Uzaktan Metod Çağırma (Remote Method Invocation)

RMI[26], programcıya uzak nesne metodlarına çağrı gerçekleştirme olanağı sunar. Ağ üzerindeki nesnelerin metodlarını kullanabilmek dağıtık veri işleme içinde büyük yararlar sağlamaktadır. Uzak nesnelerin çağrılması programcıdan saydam olarak gerçeklenmektedir. Programcı haberleşme ayrıntıları ile ilgilenmek zorunda değildir. RMI alt yapısı programcıyı her türlü alt seviye ayrıntılarından soyutlamaktadır.

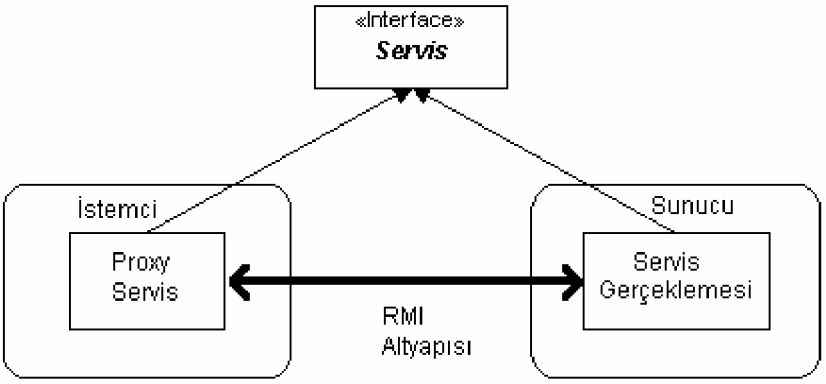
Temel olarak RMI, metodların tanımları ve metodların gerçeklemeleri olarak iki ana kısımdan oluşmaktadır. Metodlar JAVA dilinin “Interface” yapısı kullanılarak tanımlanır, gerçeklemeler ise sınıf (Class) tanımlayarak oluşturulur. Bilindiği gibi, “Interface” gerçeklemelerin yer almadığı tanımlama direktiflerini içerir. “Class” tanımı ise hem tanımlamaları hem de gerçeklemeleri içermektedir. İstemci program

tanımlamaları içerirken, uzak sunucu programı sınıf yapısını barındırır. Şekil 4.1 RMI alt yapısını yalın haliyle göstermektedir.



Şekil 4.1 : RMI mimarisi

RMI, istemci tarafında, uzak nesneye işaretçi olarak "proxy" adı verilen ve sadece tanımlamaları barındıran bir nesne oluşturur. Aslında, hem sunucu taraf, hem de istemci taraf aynı arayüzü (interface) gerçekleyen iki ayrı sınıftır.



Şekil 4.2 : RMI da istemci ve sunucu aynı arayüzü gerçekler

İstemci program proxy nesne üzerinde metod çağrıları yapar, RMI bu çağrıları uzak JVM'e yollayarak servis gerçeklemesine yönlendirir. Geri dönüş değerleri de RMI altyapısı tarafından proxy nesnesine geri yollanır.

Nesne serileştirilmesi RMI alt yapısında kullanılan en önemli özelliktir. Uzaktaki metoda gönderilecek olan parametrelerin sınıfları serileştirilebilir özellikte olmalıdır. Basit olarak çağrı sırasında aşağıdaki adımlar işletilir.

- İstemci program proxy nesnesinin metoduna bir çağrı yapar.

- İstemci RMI, çağrıyı sunucu RMI ya yönlendirir. Bu yönlendirme yapılırken bütün giriş parametreleri serileştirilerek karşı tarafa iletilir. Uzak çağrı söz konusu olduğu için parametrelerin adresleri değil gerçek değerleri yollanmalıdır.

- Çağrı isteğini ve serileştirilmiş parametreleri alan sunucu RMI, parametre nesnelerini serileştirilmiş veriden yeniden oluşturur ve

- Çağrı sonucu olarak üretilen geri dönüş parametresi serileştirilerek tekrar istemci RMI ya iletilir ve istemci RMI alt yapısı seriden geri dönüş nesnesini yeniden oluşturarak istemci programa iletir.

RMI incelenirken en sık karşılaşılan sorulardan biri de istemci programın uzak RMI servisini nasıl bulduğudur. Bunun için RMI alt yapısında "Adlandırma Servisi" kullanılır. "Adlandırma Servis Sunucusu" ise hem istemci tarafından hem de sunucu tarafından bilinen bir adreste yer almalıdır. Sunucu program gerçeklemeleri içeren bir nesne yarattığında, bu nesneyi RMI ya ihraç eder. İhraç işlemi sonucu sunucu RMI, belirli bir porttan bu nesneyi kullanmak isteyen çağrıları dinleyen bir servis açarak dinleme düzenine geçer. Bundan sonra da sunucu bir isim altında bu servisi Adlandırma Sunucusuna kaydettirir.

İstemci tarafındaki RMI, Adlandırma Sunucusundan istediği zaman sorgu yaparak kullanmak istediği uzak servise bir işaretçi elde edebilir.

Bir örnek ile RMI'ın nasıl çalıştığı daha iyi anlaşılabilir. Öncelikle hem istemci hem de sunucu tarafın kullandığı arayüz tanımlanmalıdır. Bu arayüz erişilecek nesne metodlarına tanımlamaları içerir.

```
public interface Calculator extends java.rmi.Remote {
    public long add(long a, long b) throws java.rmi.RemoteException;
    public long sub(long a, long b) throws java.rmi.RemoteException;
    public long mul(long a, long b) throws java.rmi.RemoteException;
    public long div(long a, long b) throws java.rmi.RemoteException;
}
```

Bu arayüz (interface) ile uzak metodların isimleri, giriş ve çıkış parametreleri tanımlanmıştır. Bu arayüzü hem istemci hem de sunucu program kullanacaktır.Bir sonraki adım, sadece sunucu tarafta bulunacak ve bu metodların gerçeklemelerini içerecek olan kod parçalarının yazılmasıdır.

```
Public class CalculatorImpl extends
java.rmi.server.UnicastRemoteObject  implements Calculator {
```

```
    public CalculatorImpl() throws java.rmi.RemoteException {
        super();
    }

    public long add(long a,long b) throws java.rmi.RemoteException {
       return a + b;
    }

    public long sub(long a,long b) throws java.rmi.RemoteException {
       return a - b;
    }

    public long mul(long a,long b) throws java.rmi.RemoteException {
        return a * b;
    }

    public long div(long a,long b) throws java.rmi.RemoteException {
        return a / b;
    }
}
```

İstemci taraf için gerekli olan metod tanımlamaları ve sunucu taraf için gerekli olan metod gerçeklemeleri yazılmıştır, bundan sonra yapılması gereken istemci ve sunucu programın yazılmasıdır. Ancak programlar yazılmadan önce istemci ve sunucu taraf için istekleri karşılayacak olan RMI servisleri için gerekli olan nesnelerin oluşturulması gerekmektedir. Bunun için programcı ek bir kod yazmaz sadece RMI alt yapısı ile birlikte gelen RMI derleyicisini (rmic) çalıştırır. RMI derleyicisi istemci ve sunucu taraf için gerekli olan bu nesneleri oluşturur.

İlk adım olarak sunucu programı yazarsak, yapılacak tek şey, metod gerçeklemelerini barındıran nesneyi yaratmak, sonra da bunu Adlandırma Sunucusuna kaydettirmektir.

```
import java.rmi.Naming;
public class CalculatorServer {
   public CalculatorServer() {
```

```
      try {
        Calculator c = new CalculatorImpl();
        Naming.rebind("rmi://localhost:1099/CalculatorService", c);
      } catch (Exception e) {
        System.out.println("Hata: " + e);
      }
    }

    public static void main(String args[]) {
      new CalculatorServer();
    }
}
```

İstemci program ise öncelikle Adlandırma Sunucusundan kullanmak istediği servise bir işaretçi elde edecek sonra yerel metod çağrısı gibi çağrılarını yapacaktır.

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.net.MalformedURLException;
import java.rmi.NotBoundException;

public class CalculatorClient {
    public static void main(String[] args) {
        try {
            Calculator c = (Calculator) Naming.lookup(
                 "rmi://localhost:1099/CalculatorService");
            System.out.println( c.sub(4, 3) );
            System.out.println( c.add(4, 5) );
            System.out.println( c.mul(3, 6) );
            System.out.println( c.div(9, 3) );
        }
        catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}
```

4.3 JAVA Yansıma (Java Reflection)

Temel olarak JAVA yansıma[27], çalışma anında sınıflar, nesneler ve arayüzler hakkında belirli bir takım bilgilerin edinilmesini ve hatta üzerlerinde işlemler yapılabilmesini sağlar. Yansıma kullanılarak çalışma anında şu sorulara yanıt bulunabilir:

- Belirtilen nesne hangi sınıfa aittir?
- Belirtilen sınıftaki alanlar (field names) nelerdir?
- Belirtilen alanın tipi nedir?
- Belirtilen sınıfta tanımlı olan metodlar nelerdir?
- Belirtilen metodun parametreleri nelerdir?
- Belirtilen sınıfın kurucu (constructor) metodu nedir?

Yansıma ile nesneler üzerinde aşağıdaki işlemler yapılabilir

- Nesnenin metodu çağrılabilir
- Nesnenin bir değişkenine değer atanabilir
- Dinamik diziler yaratılabilir.

JAVA, "Class" sınıfından özel bir nesne ile diğer sınıf ve nesneler hakkında bilgi edinilmesini sağlar.

```
String myObject = "my string";
Class theClass = myObject.getClass();
System.out.print("The object type is :");
System.out.println(theClass.getName());
```

"theClass" değişkeni ile nesnenin sınıf bilgisine erişilmiştir. Bu özellik ile nesnenin sınıf bilgisi çalışma anında edinildiği için programcıya oldukça esneklik sağlanmaktadır.

Aşağıdaki örnekle bir sınıfın içinde tanımlı olan alan isimleri öğrenilmiştir.

```
import java.lang.reflect.*;
import java.awt.*;
class SampleField {
```

```
        public static void main(String[] args) {
           GridBagConstraints g = new GridBagConstraints();
           printFieldNames(g);
        }

        static void printFieldNames(Object o) {
           Class c = o.getClass();
           Field[] publicFields = c.getFields();
           for(int i = 0; i < publicFields.length; i++) {
              String fieldName = publicFields[i].getName();
              Class typeClass = publicFields[i].getType();
              String fieldType = typeClass.getName();
              System.out.println("Ad: " + fieldName +
               ", Tip: " + fieldType);
            }
           }
     }
```

Aşağıdaki örnekte ise bir sınıfın metod bilgileri öğrenilmiştir.

```
class SampleMethod {

   public static void main(String[] args) {
      Polygon p = new Polygon();
      showMethods(p);
   }

   static void showMethods(Object o) {
      Class c = o.getClass();
      Method[] theMethods = c.getMethods();
      for (int i = 0; i < theMethods.length; i++) {
         String methodString = theMethods[i].getName();
         System.out.println("Ad: " + methodString);
         String returnString =
                        theMethods[i].getReturnType().getName();
         System.out.println("   Dönüş Tipi: " + returnString);
         Class[] parameterTypes = theMethods[i].getParameterTypes();
         System.out.print("   Parametre Tipi:");
         for (int k = 0; k < parameterTypes.length; k ++) {
```

```
            String parameterString = parameterTypes[k].getName();
            System.out.print(" " + parameterString);
         }
         System.out.println();
      }
   }
}
```

Yansıma özelliğinin sıklıkla kullanılan fonksiyonu nesne metodlarının çağrılmasıdır (invoke). Aşağıdaki örnek yansıma ile metod çağırmayı göstermektedir.

```
class SampleInvoke {
   public static void main(String[] args) {
      String firstWord = "Hello ";
      String secondWord = "everybody.";
      String bothWords = append(firstWord, secondWord);
      System.out.println(bothWords);
   }

   public static String append(String firstWord, String secondWord)
{
      String result = null;
      Class c = String.class;
      Class[] parameterTypes = new Class[] {String.class};
      Method concatMethod;
      Object[] arguments = new Object[] {secondWord};
      try {
        concatMethod = c.getMethod("concat", parameterTypes);
        result = (String) concatMethod.invoke(firstWord, arguments);
      } catch (NoSuchMethodException e) {
          System.out.println(e);
      } catch (IllegalAccessException e) {
          System.out.println(e);
      } catch (InvocationTargetException e) {
          System.out.println(e);
      }
      return result;
   }
```

Hareketli etmen sisteminin geliştirilmesinde JAVA yansıma özelliği sıkça kullanılmıştır. Örneğin, etmen programcısı mesaj geldiğinde yapılmasını istediklerini etmenin "OnMessageArrive" isimli bir metodu içine yazmaktadır. Bu metod, etmene mesaj geldiğinde sistem tarafından yansıma kullanılarak çağrılır. Diğer bir örnek etmen programcısının mesaj gönderildiğinde gelen yanıtı işleyecek olan metodu belirleyebilmesidir. Bu durumda etmen yanıt için beklemez, çünkü yanıtı karşılayacak olan metodu gerçeklemiş ve bildirmiştir. Aynı şekilde sistem bu metodu yansıma ile kullanır. Etmen sisteminin bir tarayıcı ile yönetilmesi sırasında da gelen istekleri karşılayan kod parçaları yansıma özelliği kullanılarak çalıştırılır.

Yansıma ile bir nesnenin erişilmesine izin vermediği alanlarına erişmek mümkün değildir. Örneğin nesnenin "private" olan metodları yada alanları buna örnektir. Yansıma nesneye yönelik programlama felsefesinin güvenlik özelliklerini aşarak bu bilgilere erişemez.

5. GÜVENLİ ETMEN SİSTEMİ

JAVA dilinde geliştirilmiş yeni bir hareketli etmen sistemli olan Güvenli Etmen Sistemi (GES), mevcut etmen sistemlerinden, özellikle güvenlik mekanizmaları göz önüne alınarak tasarlanmış güvenli mimarisiyle ayrılır. GES, tüm temel etmen fonksiyonlarını desteklemesinin yanında, etmen programcısına esnek geliştirme fonksiyonları ve çalışma ortamı da sunar.

GES mimarisi ayrıntılı olarak incelenmeden önce sık söz edilecek olan bazı terimleri tanımlamak faydalı olacaktır.

Hareketli Etmen (Mobile Agent): Etmen programcısı tarafından mimarinin sunduğu kurallara bağlı kalarak JAVA dili ile yazılmış, durum bilgisi barındıran ve bu bilgiyi güncelleyen, diğer etmenler ile haberleşebilen ve ağ üzerinde farklı bilgisayarlar üzerine göç edip çalışmasını orada sürdürebilen kod parçaları olarak tanımlanır.

Etmen Sahibi (Agent Owner): Etmenin ilk kez sisteme dahil edildiği düğümün adresi olarak tanımlanır. Bu adres bilgisi etmen sunucusunun üzerinde çalıştığı düğümün IP adresini ve TCP port numarasını içerir.

Hareketlilik (Mobility) : Hareketli olan etmenler belirli durumlarda ağda yer alan diğer makineler üzerine göç edebilir. Göç sonunda sahip olduğu son durum bilgisi ile çalışmasına devam eden etmen, göç etmeden önce yürütmüş olduğu program satırını izleyen satırı işleyerek çalışmasına devam eder, ya da etmen aktivasyon kodunun ilk satırından çalışmaya başlar. Bu çalışma düzenlerinden ilki "güçlü hareketlilik" (strong mobility), ikincisi ise "zayıf hareketlilik" (weak mobility) olarak adlandırılır. GES, zayıf hareketlilik çalışma düzenini desteklemektedir. Bu tezde, GES bağlamında hareketlilikten kasıt, güçsüz hareketlilik olacaktır. JAVA dili ile geliştirilmiş ve güçlü hareketliliği destekleyen bir etmen sistemi henüz mevcut değildir.

Etmen Göçü: Belirli koşulların oluşması halinde, etmenin çalışmasını durdurup, ağ üzerindeki bir başka düğüm üzerinde çalışmasını sürdürmek üzere, uzak düğümlere kendini taşıması olarak adlandırılır. Etmen, göç sırasında son durum bilgisini de taşır.

GES, etmen programcısına olay tabanlı kod geliştirmesi için gerekli ortamı sunar. Programcıya etmenin yaşam döngüsü içinde yer alabileceği, ayrıntılı olarak sonraki bölümlerde incelenecek olan, her durum için farklı bir metod yazma olanağı sunar. Bu model programcıya esnek bir programlama arayüzü sağlar.

Sistem, etmenlerin, mesajlar aracılığıyla, birbirleriyle senkron ve asenkron olarak haberleşmelerine olanak sağlar. Mesajlaşma alt yapısı ölçeklenebilir olma özelliğine sahiptir. Ayrıca sistem, etmenlere birbirlerine o an üzerlerinde çalıştıkları düğüm adresinden bağımsız olarak mesaj gönderme olanağı sunmaktadır. Konuma göre saydam olan mesajlaşma altyapısı sayesinde, bir etmen diğer bir etmene mesaj gönderebilmek için onun ağ üzerinde nerede olduğu bilgisine ihtiyaç duymaz.

Mimarinin en önemli özelliklerinden biri de güvenlik politikalarını etkin bir şekilde kullanarak, hem etmeni hem de düğümleri koruyan fonksiyonlar içermesidir. Güvenlik politikaları dinamik olarak tanımlanabilir, değiştirilebilir ve uygulanabilir olma özellikleri esnek bir çalışma ortamı sunar. Bu yaklaşım, etmenin değişen çevresel koşullara uyum sağlamasını kolaylaştırdığı gibi, etmen programcısına da, etmen kodunda değişiklik yapmadan etmen davranışlarını farklılaştırma olanağı sağlar.

GES, etmen durum ve kod bilgisinin korunması içinde gerekli araçlara sahiptir. Etmen kod ve durumu bir etmenin yaşam döngüsü boyunca, üzerinde çalıştığı düğümün ana belleği hariç, hiç bir şekilde açık biçimde tutulmaz. Şifreleme metodları ile etmen kodu, durumu ve politikası, yetkisiz değişimleri de sezme özelliğine sahip fonksiyonlar ile sürekli koruma altındadır.

Sistemdeki tüm uzak haberleşmeler de ulaşım katmanı seviyesinde SSL protokolü ile gerçeklenir. Bu özellik, ağı dinleyen yetkisiz etmen veya kişilerin etmenlere ait özel bilgileri çalmasını yada sistemdeki etmenleri izlemesini engeller. Tüm mimari bileşenleri, ağ üzerindeki herhangi bir düğüm üzerinde çalışan bir tarayıcı (web browser) ile izlenebilir ve yönetilebilir. Bu özellik yönetilebilirlik açısından sisteme büyük esneklik sağlamaktadır.

Etmen yaratılması, etmenin aktif-pasif duruma geçirilmesi yada göç etmesi veya mesajlaşması gibi aktivitelerin saklanması ve izlenmesi, etmenin düğüm üzerinde çalışırken diğer etmenlere karşı korunması gibi daha bir çok özellik mimaride mevcuttur. Konu başlıkları altında ilgili konular ayrıntıları ile incelenecektir.

6. ETMEN YAZILIMI GELİŞTİRME ARAYÜZÜ

GES mimarisi, programcıya karmaşık etmen davranışını programlayabileceği basit bir şablon ve oldukça esnek fonksiyonlar sunmaktadır. Şablon, etmenin yaşam döngüsü boyunca içinde bulunabileceği her farklı durum için, programcının bu durumlarda etmenin davranışlarını yönlendirecek kodlarını yazabileceği hazır metodlar içermektedir. Etmen şablonu aşağıda verilmiştir.

```
import agent.*;
public class Main extends Agent {
  public void OnMessageArrive() {
  }
  public void OnCreate() {
  }
  public void OnActivate() {
  }
  public void OnInactivate() {
  }
  public void OnEnd() {
  }
  public void OnTransfer() {
  }
}
```

Yazılacak yeni etmen, “Agent” sınıfından türemeli ve “Main” isimli yeni bir sınıfa ait olmalıdır. “Main” sınıfı içinde verilen metodların bazıları etmenin içinde bulunabileceği durumları, diğerleri ise önemli fonksiyonlarını temsil eder. Metodlar ve anlamları şu şekildedir.

OnMessageArrive: Etmen kodu içinde mesaj gelmesi için bekleyen komutlara gerek duyulmaz. Etmene yeni bir mesaj geldiği zaman bu metod çalıştırılır. Etmen

bu metod içinde mesaja uygun işlemler yürütüp, gelen mesajın yanıtını veya yeni bir mesaj yollayabilir, ya da göç isteğinde bulunabilir. Çalıştırılan metod kodu ayrı bir iplik(thread) içinde işletilir. Bu metodun çalıştırılabilmesi için etmenin aktif durumda olması gerekmektedir. Bu metod, etmene her yeni mesaj geldiğinde çalıştırılır, ancak eğer bir önceki mesajın işlenmesi devam ediyorsa çalışma bekletilir. Etmene ait aktif durum metodu (OnActivate) da ayrı bir iplik içinde çalıştığından, bazı durumlarda etmen iki ayrı iplik içinde (OnActivate ve OnMessageArrive) çalışıyor olabilir. Programcı bu durumun varlığını bilerek etmeni programlamalıdır.

OnCreate: Etmenin sisteme alınıp, yeni bir kimlik bilgisi verilerek yaratıldığı anda çalıştırılan metoddur. Bu metod etmen yaşam döngüsü boyunca sadece bir kez çalıştırılır. Etmene ait bu metod çalıştırıldıktan sonra aktif durumu belirleyen "OnActivate" metodu çalıştırılır. Bu durumda etmen, kabuk tarafından kendisine atanan arayüz fonksiyonlarını henüz kullanamaz.

OnActivate : Etmen aktif duruma geçirildiği zaman çalıştırılan metoddur. Kontrol bu metodun ilk satırına geçtiği andan itibaren etmen aktif durumdadır. Bu metod da ayrı bir iplik içinde çalıştırılır. Metod kodunun son satırı da çalıştırılıp, bitirilmiş olsa dahi etmen aktif olma durumunu sürdürür, çünkü programcı "OnMessageArrive" metodunu programlayarak gelen mesajları işleyen bir yapı kurmuş olabilir. Kısaca etmenin aktif durumdan çıkması, bu metodun sonlanması ile değil, göç etme, uyuma gibi aktivitelerin gerçekleşmesiyle olur. Programcı bu metod içinde kod yazmamış ancak mesaj bekleyerek gerekli işleri yapacak kodu "OnMessageArrive" metodu içinde gerçeklemiş olabilir. Bu durumda etmen yine de aktif durumdadır. Etmen aktif duruma geçerken son durum bilgisi diskten okunur ve belleğe alınır. Bu durumda etmen, kabuk tarafından kendisine atanan arayüz fonksiyonlarının hepsini kullanabilir.

OnInactivate : Etmenin pasif duruma geçerken çalıştırılan metoddur. Etmen pasif duruma geçerken bellekteki son durumu diske yazılarak saklanır. Pasif durumda olan bir etmen iletişim kuramaz, mesaj gönderip alamaz. Ayrıca, bu metod içinde mesaj göç etme istekleri de karşılanmaz.

OnEnd : Etmenin kendisini yok etmeye karar verdiği anda çalıştıran metoddur. "OnInactivate" metodundan sonra çalıştırılır. Bu metodun sonlanmasıyla etmen kodu, durum bilgisi ve politikası diskten silinmiş olur.

OnTransfer: Etmenin göç etmeden önce çalıştırdığı metoddur. Bu metod içinde mesaj alma ve gönderme mümkündür. Göç etme isteğinde bulunan etmenin bu

metodu çalıştırıldıktan sonra “OnInactivate” metoduda sistem tarafından otomatik olarak çağrılır.

Yukarıda tanıtılan metodların tümü etmen kodu içinde tanımlanmalıdır, ancak istenirse içlerine kod yazılmayıp, boş bırakılabilir. Programcı, etmenin davranışlarını belirleyen başka metodlar da yazabilir. Tanıtılan metodların içinde yeni yazılan diğer metodları çağırmak da mümkündür. Etmen kodu JAVA ile yazıldığı için JAVA’nın bütün olanaklarından faydalanılabilir. Ancak bir etmene ait kod birden fazla JAVA paketine (Package) bölünemez.

6.1 Etmen Arayüzü (AgentInterface)

GES sunucusu bir etmeni aktif duruma getireceği zaman ilk önce bir etmen kabuğu yaratır ve denetimi kabuğa bırakır. Kabuk etmeni aktif duruma geçirmek üzere gerekli adımları yürütür ve etmene, üzerinden çevresi ile iletişim kurabileceği bir arayüz (AgentInterface) sunar. Etmen, bu arayüzü kullanarak, JAVA API’leri dışındaki bütün isteklerini yerel GES sunucusuna iletir.

6.1.1 Etmen Çağrıları

Aşağıda etmen arayüzünü kullanarak etmenin çevresi ile iletişime geçmek için kullanabiceği çağrılar ve kullanım örnekleri verilmiştir. Çağrılar önce tablo 6.1.1.1 de listelenmiş daha sonra örnekler ile açıklanmıştır.

Tablo 6.1.1.1 Etmen çağrı listesi

Metod	**Kullanım Amacı**
public String getAgentHostName()	Etmenin o an üzerinde çalıştığı düğümün adresini öğrenmesi sağlanır.
public void setVisibleOn(String strIdentifier)	Etmen bir isimle kendini ortama duyurur.
public void setVisibleOff()	Etmen kendini ortamdan gizler.
public Enumeration getAgentPointer(String strIdentifier)	Etmen, ismini verdiği bir diğer etmene bir işaretçi elde eder.

public Enumeration getAgentPointer()	Etmen sistemdeki tüm etmenleri gösteren bir liste elde eder.
public MessageID sendMessage(AgentPointer agentpointer, Message message)	Etmen diğer bir etmene mesaj yollar.
public MessagePacket receive()	Etmen giriş kuyruğundaki mesajı alır.
public boolean isMessageFailed(MessageID id)	Etmenin mesajının başarılı bir şekilde gönderilip gönderilmediğini öğrenir.
public void sendReply(MessagePacket packet,Object reply)	Etmen, bir mesajın yanıtını yollar.
public boolean isReplyReady(MessageID id)	Etmen, daha önce gönderdiği mesajın yanıtının hazır olup olmadığını öğrenir.
public boolean waitForReply(MessageID id,long ms)	Etmen gönderdiği mesajın yanıtı için belirli bir süre bekler.
public Object getReply(MessageID id)	Etmen giriş kuyruğundan daha önce gönderdiği bir mesajın yanıtını alır.
public boolean isAgentAlive()	Etmenin hala canlı olup olmadığı sorgulanır.
public boolean waitForMessage(long ms)	Etmen mesaj almak için belirli bir süre bekler.
public TransferResponse move(String strAgentHostName)	Etmen uzak düğümlere göç eder.
public void suspend(long suspendTime)	Etmen belirli bir süre uykuya geçer.
public AgentPointer createClone()	Etmen bir kopyasını yaratır.
public void setRepliedMessageHandler(String method)	Etmen, gönderdiği mesajnın yanıtı geldiğinde çağrılmasını istediği metodu belirler.

6.1.1.1 Genel Çağrılar

getAgentHostName : Etmen bu çağrı ile üzerinde çalıştığı etmen sunucusunun adresini öğrenir. Özellikle sıkça göç eden bir etmenin, daha önce ziyaret edip etmediği bir düğümde olup olmadığını anlaması açısından yararlı bir fonksiyon olabilir. Etmen sunucu adresi "protokol://Ip_Adresi:Port/Etmen_sunucu_adı" formatındadır. Protokol olarak "RMI" kullanılır. "Ip_Adresi", sunucunun üzerinde çalıştığı düğümün IP adresidir, port ise etmen sunucusunun istekleri dinlediği TCP port numarasıdır. Bir düğüm üzerinde birden çok etmen sunucusu çalışabilir. Aynı düğüm üzerindeki etmen sunucularını ayırt etmek için "Etmen_sunucu_adı" kullanılır.

Aşağıda örnek kod parçası verilen etmen, düğüm üzerinde aktif olduğu zaman o düğümü daha önce ziyaret edip etmediğinin kontrolünü yapmaktadır. Daha önce ziyaret ettiği düğüm adreslerini "visitedHosts" isimli bir değişken üzerinde tutmaktadır. Düğüm ilk kez ziyaret ettiği bir düğüm ise, ziyaret edilme zamanı ile birlikte, etmen sunucu adresini listesine eklemektedir. "visitedHosts" isimli değişkenin "Hashtable" sınıfından olup serileştirilebilir özelliktedir. Etmen bu değişkenin değerini göç sırasında güncel olarak saklayacaktır.

```
import java.util.*;

import agent.*;

public class Main extends Agent {

  private Hashtable visitedHosts;

  public void OnActivate() {

    AgentInterface myInterface = getAgentInterface();
    String currentHost = myInterface.getAgentHostName();
    if (visitedHosts.contains(currentHost)) {
      System.out.println("Bu düğümü daha önce ziyaret ettim");
    }
    else {
      visitedHosts.put(currentHost,new Date());
      System.out.println("Bu düğümü ilk kez ziyaret ediyorum");
    }

  }

......
}
```

setVisibleOn, SetVisibleOff: Etmenler birbirlerinin varlıklarından haberdar olabilmek için kendilerine isim verirler ve "setVisibleOn" çağrısıyla bu ismi ortama duyururlar. Aşağıda örnek kod parçası verilen etmen, aktif olduğu anda kendini

“counter” ismiyle etrafına duyurmuştur. Bir etmen tekrar bu çağrıyı yaparak ismini değiştirebilir. Etmenler isimleri ile değil, kimlikleri ile ayırt edilirler; tıpkı aynı isimli ancak farklı kimlik kartlarına sahip kişilerin var olması gibi. Etmen sorgulayarak belirli isimdeki etmenleri sorgulayabilir, ama gerçekten haberleşmek istediği etmeni diğerlerinden ayırt etmek için başka işlemler de yürütmelidir.

```
import agent.*;

public class Main extends Agent {

  public void OnActivate() {

    System.out.println("I am the counter agent and activating");
    getAgentInterface().setVisibleOn("counter");

  }
.........
}
```

Yukarıda etmenin tanımlanması gereken diğer metodları yazılmamıştır.

Bir etmen, bir diğer etmeni bulmak için ilk önce ismini sorgulamayı dener. Eğer sistemde aradığı isimde etmenler var ise, kendisine, bir liste halinde, bu etmenlere işaretçiler geri döndürülür. Etmen, bu işaretçiler (AgentPointer) üzerinden hedef etmenlere mesaj gönderebilir.

getAgentPointer : Bu çağrı ile etmen, ismini verdiği etmenlere bir işaretçi listesi edinir. Giriş parametresi olarak katar türünden etmen ismini alan fonksiyon dönüş olarak “AgentPointer” türünden nesneleri içeren bir liste (Enumeration) elde eder.

Aşağıdaki örnek kodda etmen, belirli bir anda sistemde yer alan “counter” isimli etmenlerin listesini edinip, eğer bu isimle kenidini duyurmuş etmenler varsa, sayılarını çıkışa aktarmaktadır.

```
import agent.*;

public class Main extends Agent {

  public void OnActivate() {

    .......

    System.out.println("counter isimli etmenler aranıyor..");
    AgentInterface agentinterface = getAgentInterface();
    Enumeration agentlist =
                  agentinterface.getAgentPointer("counter");
    if (agentlist == null) {
      System.out.println("Sistemde counter isimli etmen yok.");
    }
   else{
```

```
   int i=0;
   while (agentlist.hasMoreElements()) {
      AgentPointer agentpointer = (AgentPointer)
                                    agentlist.nextElement();
      inc(i);
    }
    System.out.println("counter isimli etmen sayısı :"+ i);
   }


  }
..........
}
```

"AgentPointer" isimli sınıf, etmen kimliğini tanımlamak için kullanılan "AgentIdentity" sınıfının sadece okunabilir değişken ve metodlarını içeren bir sınıftır. Sorgu yapan etmene bir bakıma sadece okunabilir etmen kimlikleri döndürülmüştür.

Eğer etmen bu fonksiyonu parametresiz kullanırsa sistemdeki tüm etmenlerin (kendine isim vermiş olsun yada olmasın) bir listesini elde eder.

Etmen bu çağrıyı yaptığında yerel etmen sunucusu, iletişimde olduğu "Gözleyici" düzende çalışan etmen sunucu ile haberleşerek (GGES) güncel etmen listesini edinir. Sonra kendini etmenin istediği isimle duyurmuş olanları liste halinde etmene iletir. Çağrı sırasında etmen, yerel etmen sunucusundan yanıt gelene kadar bloke olur.

isAgentAlive: GES etmenlerinin her biri ayrı bir (bazen birden fazla) iplik içinde çalışmaktadır. Herhangi bir nedenle etmen sunucusu etmeni pasif duruma geçirdiği zaman, etmene ait aktif kodun sonlanmasını bekler. Ancak JAVA'da herhangi bir ipliğin çalışmasını sonlandırmak her zaman mümkün olmaz. İpliğe kesme işareti yollanarak komut verilebilir ancak bu sırada ipliğin kesme işaretini bekliyor olması gerekmektedir. Etmen sunucusu etmeni pasif duruma geçirirken, etmenin o anki durumunu saklar ve etmene ait, etmen kabuğu, giriş çıkış kuyrukları, etmen politikası gibi bütün yapıları bellekten siler. Ancak etmen kodu içinde o an çalışan aktif kod parçası (bu bir iplikte olabilir) varsa bellekte çalışmaya devam eder. Örneğin "while" döngüleri bu tür koda örnek olarak verilebilir. Aşağıdaki kod parçasını yürütmekte olan bir ipliği ana programı sonlandırmadan kesmek mümkün değildir.

```
while (true)
{
....
}
```

Bu nedenle, etmenin "OnActivate" metodu içinde bu tür kod parçası varsa, etmen gerçekte pasif duruma geçmiş olmasına rağmen bu kod çalışmaya devam edebilir.

Böyle durumlarda etmen arayüzünü kullanmaya çalışmak aykırı durum oluşmasına neden olacaktır, çünkü etmen artık aktif durumda değildir.

Bu gibi durumları sezmek için programcı, “isAgentAlive” çağrısı yaparak etmenin hala aktif ve çalışıyor durumda olduğundan emin olur. Çağrının geri dönüş değeri etmen aktif ise “true”, değil ise “false” tur.

```
import agent.*;

public class Main extends Agent {

  public void OnActivate() {
    getAgentInterface().setVisibleOn("calculator");

      while (getAgentInterface().isAgentAlive())
      {
        ..........
      }


}
..........
}
```

suspend : Aktif durumda olan bir etmenin belirli bir süre pasif durumda kalmak için kullanabileceği fonksiyondur. Giriş parametresi saniye cinsinden pasif durumda kalma süresidir. Etmenin o anki durumu saklanır, pasif duruma geçirilir ve süre sonunda son durum bilgisi ile birlikte tekrar aktif duruma getirilir. GES sunucuların belirli bir süre için pasif duruma geçmiş olan etmenleri zamanı geldiğinde aktif duruma geçirecek prosesleri mevcuttur. Sunucu içinde çalışan bu proses, sürekli pasif etmen var mı, varsa ne kadar süre ile pasif durumda kalacak bilgisini kontrol eder ve pasif olma süresi dolanları tekrar aktif duruma geçirir.

```
import agent.*;

public class Main extends Agent {


  public void OnMessageArrive() {

    MessagePacket packet = getAgentInterface().receive();
    Message message = packet.getMessage();
      if (message.getMessageName().equalsIgnoreCase("Suspend")) {
      //10 saniye pasif durumda bekle..
      getAgentInterface().suspend(10)
      }
  }

.....
}
```

“Suspend” isimli bir mesaj aldığında 10 sn süre ile pasif duruma geçen bir etmen kodu yukarıdaki gibi olacaktır. Bu çağrıdan sonra etmen herhangi bir kod işletmemelidir çünkü artık aktif durumda değildir. Pasif duruma geçen her etmende olduğu gibi bu çağrı sonunda da etmenin pasif duruma geçtiği “Gözleyici” düzende çalışan GES sunucuya iletilir ki, güncel etmen listesinden silinebilsin. Etmen bu çağrı sonunda belirtilen süre kadar pasif durumda kaldığı için diğer etmenlerden mesaj alamaz. Aslında bu etmen diğer etmenler tarafından sistem de görülmez.

6.1.1.2 Haberleşmeye Yönelik Çağrılar

sendMessage: Etmenin mesaj göndermek için kullandığı fonksiyondur. Giriş parametreleri mesaj nesnesi ve alıcı etmen işaretçisidir. Dönüş parametresi olarak mesajın numarası döner, böylece etmen gönderdiği mesajın akibetini bu numaradan sorgulayarak öğrenebilir. Örnek olarak iki sayının toplamını başka bir etmene hesaplaması için yollayan bir etmen kodu aşağıdaki gibi olacaktır.

```
import agent.*;

public class Main extends Agent {

  public void OnActivate() {

    .......

    AgentInterface agentinterface = getAgentInterface();
    Enumeration agentlist =
                  agentinterface.getAgentPointer("calculator");
    if (agentlist == null) {
      System.out.println("Sistemde hesaplayici etmen yok.");
    }
   else{
      AgentPointer agent = (AgentPointer)
                                    agentlist.nextElement();
      Message message = new Message("Topla", "123","456");
      MessageID id = agentinterface.sendMessage(agent, message);
      //mesaj gönderildi, ancak akibet sorgusu yapılmalı..

    }
   }


  }
..........
}
```

Yukarı kod parçasında etmen önce sistemde kendini “calculator” ismiyle duyurmuş olan etmenlerin listesini edinmeye çalışır, eğer bu isimde etmen var ise kendine liste halinde gelen etmenlerden ilkine topla isminde ve parametreleri “123” ve “456” olan bir mesaj oluşturur ve yollar.

“sendMessage” çağrısı etmeni bloke etmez. Çağrı mesajın etmenin çıkış kuyruğına eklenmesiyle sonlanır ve etmen çalışmasına devam eder. Arka planda kabuk işlevleri etmenin çıkış kuyruğundan mesajı alır ve yerel etmen sunucusuna alıcısına iletmesi için istekte bulunur. Gönderim işlemleri yürütülürken etmen çalışmasına devam edecektir, ancak, gerekli görürse etmeni bloke eden yeni bir çağrı ile cevabı bekleyebilir. (waitForReply çağrısı)

receive : Etmen kendisine ulaşan bir mesajı alma isteği için bu fonksiyonu kullanır. Daha önce de açıklandığı gibi etmene bir mesaj ulaşıp giriş kuyruğuna yerleştirildiği anda, kabuk “OnMessageArrive” metodunu çağırarak etmeni uyarır. Etmen isterse gelen mesajı receive çağrısı ile giriş kuyruğundan çeker. “receive” çağrısının giriş parametresi yoktur, dönüş parametresi ise “MessagePacket” sınıfından bir nesnedir. Etmen mesaj paketini aldıktan sonra paket içinden mesajı, mesajın adını, parametrelerini öğrenebilir. Örneğin yukarıda iki sayının toplanması için mesaj gönderen etmenin alıcısı aşağıda verilen koda benzer bir uygulamaya sahip olacaktır.

```
import agent.*;

public class Main extends Agent {

public void OnMessageArrive() {
    MessagePacket packet = getAgentInterface().receive();
    Message message = packet.getMessage();
    if (!message.getMessageName().equalsIgnoreCase("Topla")) {
      System.out.println("Anlamadığım bir mesaj geldi");
      return;
    }
    System.out.println("Mesajın Adı :" +
                                  message.getMessageName());
    System.out.println("Mesajın geldiği sunucu :" +
                                  packet.getSourceAgentHost());
    Object[] parameters = message.getParameters();
    int paramCount = message.getParameters().length;
    System.out.println("Masajın parametre sayısı :" +
                                                    paramCount);
    for (int i = 0; i <= paramCount - 1; i++) {
      //mesaj parametre tipinin String olduğu kabul ediliyor..
      System.out.println("Parametre[" + i + "] :" + (String)
                                                  parameters[i]);
    }

    ..........
  }

  public void OnActivate() {

    System.out.println("I am the calculator agent and activating");
    getAgentInterface().setVisibleOn("calculator");
```

```
  }
..........
}
```

“OnMessageArrive” metodu içinde etmen önce mesaj paketini almış, sonra mesaj paketinden mesajı çözmüştür. Mesaj adının “Topla” olması halinde işlemlerine devam etmiştir.

“receive” çağrısı sonunda mesaj, etmen giriş kuyruğundan silinir.

isMessageFailed : Mesaj gönderen etmen, bu çağrılar ile giriş parametresi olarak mesajın numarasını vererek mesaj gönderiminin başarılı olup olmadığını kontrol eder. Daha önce mesaj gönderim için örnek verilen etmen kodu, mesajın akibetini sorgulayacak fonksiyonları da içerecek şekilde aşağıdaki duruma gelecektir.

```
import agent.*;

public class Main extends Agent {

  public void OnActivate() {

    .......

    AgentInterface agentinterface = getAgentInterface();
    Enumeration agentlist =
                  agentinterface.getAgentPointer("calculator");
    if (agentlist == null) {
      System.out.println("Sistemde hesaplayici etmen yok.");
    }
   else{
      AgentPointer agent = (AgentPointer)
                                    agentlist.nextElement();
      Message message = new Message("Topla", "123","456");
      MessageID id = agentinterface.sendMessage(agent, message);
      .....

      if (agentinterface.isMessageFailed(id))
      System.out.println(“Mesaj gönderilemedi);

    }
   }


  }
..........
}
```

Mesajın başarılı gönderilmesinin anlamı, mesajın alıcı etmen giriş kuyruğuna konması anlamına gelmektedir. Alıcı etmenin aktif olmaması, etmenin üzerinde çalıştığı sunucu ile iletişim kurulamaması gibi durumlarda ise mesaj gönderimi başarısız kabul edilir. Mesajın başarılı şekilde gönderilmesi, alıcı etmenin mesajı alıp işlediği anlamına gelmemektedir.

sendReply: Etmen aldığı bir mesajın yanıtını bu metodu çağırarak yollar. Giriş parametresi olarak yanıtı gönderilen mesaj paketi ve yanıtı içeren nesneyi kullanır. Dönüş parametresi yoktur. Yanıt nesnesi JAVA'nın Object sınıfından türeyen herhangi bir nesne olabilir. GES sunucuları yanıt mesajlarını da, normal mesaj paketleri gibi yollarlar. Ancak mesaj paketinin sonundaki ACK alanı doldurularak (true), iletilen mesajın bir yanıt paketi olduğu belirlenir. Yanıt paketleri gönderici etmenin yanıt kuyruğuna konur. Daha önce incelenen etmen kodu örneği, yanıt göndermeyi de kapsayacak şekilde aşağıda genişletilmiştir.

```
import agent.*;

public class Main extends Agent {

public void OnMessageArrive() {
    MessagePacket packet = getAgentInterface().receive();
    Message message = packet.getMessage();
    if (!message.getMessageName().equalsIgnoreCase("Topla")) {
      System.out.println("Anlamadığım bir mesaj geldi");
      return;
    }
    System.out.println("Masajın Adı :" +
                                  message.getMessageName());
    System.out.println("Masajın geldiği sunucu :" +
                                  packet.getSourceAgentHost());
    Object[] parameters = message.getParameters();
    int paramCount = message.getParameters().length;
    System.out.println("Masajın parametre sayısı :" +
                                                    paramCount);
    long Toplam=0;
    for (int i = 0; i <= paramCount - 1; i++) {
      //mesaj parametre tipinin String olduğu kabul ediliyor..
      System.out.println("Parametre[" + i + "] :" + (String)
                                                  parameters[i]);
      Toplam = Toplam + Long.parseLong((String)parameters[i]);
    }
    //yanıt gönderiliyor..
    getAgentInterface().sendReply(packet, Toplam);
  }

  public void OnActivate() {

    System.out.println("I am the calculator agent and activating");
    getAgentInterface().setVisibleOn("calculator");

  }
..........
}
```

Gönderilen yanıt nesnesi serileştirilebilir özellikte herhangi bir nesne olabilir. Alıcı ve gönderici etmen yanıt nesnelerinin hangi türden olduklarını biliyorlarsa tip dönüşümleri için ek kod yazmaya gerek kalmayabilir; bilmemeleri halinde bile nesnelerin tiplerini "nesne.getClass()" JAVA çağrısı ile öğrenebilirler.

“sendReply” çağrısı yanıt paketinin çıkış kuyruğuna konması ile sonlanır. Yanıtın mesajı gönderen etmene ulaşıp ulaşmadığı kontrolü ayrıca yapılmaz. Bir mesajın takibinden gönderici etmen sorumludur ve eğer belirli bir süre sonra yanıt alamadıysa tekrar mesaj göndermelidir.

isReplyReady : Etmen herhangi bir anda göndermiş olduğu mesajın yanıtının gelip gelmediğini bu çağrı ile öğrenebilir. Giriş parametresi olarak gönderilen mesajın numarası verilir, dönüş parametresi yanıt gelmiş ise “true” gelmemişse “false” değerini içeren “boolean” bir değişkendir.

```
import agent.*;

public class Main extends Agent {

  public void OnActivate() {

    .......

    AgentInterface agentinterface = getAgentInterface();
    Enumeration agentlist =
                  agentinterface.getAgentPointer("calculator");
    if (agentlist == null) {
      System.out.println("Sistemde hesaplayici etmen yok.");
    }
   else{
      AgentPointer agent = (AgentPointer)
                                    agentlist.nextElement();
      Message message = new Message("Topla", "123","456");
      MessageID id = agentinterface.sendMessage(agent, message);
      //mesaj gönderildi, ancak akibet sorgusu yapılmalı..
      ......
      //Etmen çalışmasına devam ediyor..
      ......
      //yanıtın gelip gelmediği kontrol ediliyor.
      if(agentinterface.isReplyReady(id))
      {
      //yanıtı alıp işleyecek kod...
      }
    }
   }


  }
..........
}
```

waitForReply, getReply: Mesaj gönderiminin asenkron bir çağrı olduğu, isteğini ileten etmenin yanıt gelene kadar diğer işlerine devam edebileceğini söylemiştik. Ancak, etmen isterse kendi belirlediği bir süre için yanıtın gelmesini bekleyebilir. Bu çağrıyı kullanan etmen verdiği süre boyunca bloke olur. Eğer yanıt süre dolmadan önce gelirse, çağrı sonlanır. Giriş parametresi olarak yanıtı beklenen mesajın numarasını ve bekleme süresi verilir. Metod, dönüş değeri olarak “boolean”

türünden bir bilgi getirir; eğer yanıt belirlenen süre içinde ulaşırsa “true”, aksi durumda “false” değeri taşınır.

Yanıt geldiğinde etmen, “getReply” çağrısı ile cevap nesnesini alıp işleyebilir. Yanıt nesnesi de “Object” sınfından türeyen herhangi bir nesne olabilir. Gerekli tip dönüşümlerini etmen yapmalıdır. “getReply” çağrısının giriş parametresi gönderilen mesajın numarası, dönüş parametresi ise yanıt nesnesidir.

Daha önce mesaj gönderim için örnek verilen etmen kodu cevabı bekleyecek şekilde yeniden yazılırsa şu şekilde olacaktır.

```
import agent.*;

public class Main extends Agent {

  public void OnActivate() {

    .......

    AgentInterface agentinterface = getAgentInterface();
    Enumeration agentlist =
                  agentinterface.getAgentPointer("calculator");
    if (agentlist == null) {
      System.out.println("Sistemde hesaplayici etmen yok.");
    }
   else{
      AgentPointer agent = (AgentPointer)
                                    agentlist.nextElement();
      Message message = new Message("Topla", "123","456");
      MessageID id = agentinterface.sendMessage(agent, message);
      //yanıt için 5 saniye bekleniyor..
      if (agentinterface.waitForReply(id,5000))
      {
      System.out.println("Mesaj yanıtı geldi");
      String result = (String)agentinterface.getReply(id);
      System.out.println("123+456 = "+result);
      }
      else
      System.out.println("5 sn içinde yanıt gelmedi.. ");
    }
   }


  }
..........
}
```

waitForMessage: Bir mesaj geldiğinde, etmenin kabuk tarafından “OnMessageArrive” metodunun çağrılmasıyla uyarıldığını söylemiştik. Bu işleyişin hatasız çalışması için etmen programcısı “OnMessageArrive” metodunu gerekli gördüğü işlemleri yürütecek şekilde yazmalıdır. Ancak, etmen isterse aktif duruma geçtikten sonra, kendi belirlediği bir süre boyunca mesaj için bekleyebilir. Bu durumda, uyarılarak yeni bir mesajın varlığından haberdar olmak yerine kendisi

mesaj için bekler duruma geçer. Çağrı, etmeni verilen süre boyunca bloke eder. Belirtilen süre sonunda mesaj gelmişse çağrı “true”, gelmemişse “false” döndürür.

Daha önce “OnMessageArrive” metodu içinde gelen mesajı alan etmen kodu örneği bu durumda aşağıdaki gibi olacaktır.

```
import agent.*;

public class Main extends Agent {

public void OnMessageArrive() {
//boş bırakılır
  ..
  }

  public void OnActivate() {
    getAgentInterface().setVisibleOn("calculator");

    //10 saniye için yeni bir mesaj bekleniyor..
      if (getAgentInterface().waitForMessage(10000)) {
    // 10 saniye içinde mesaj geldiyse..

    MessagePacket packet = getAgentInterface().receive();
    Message message = packet.getMessage();
    if (!message.getMessageName().equalsIgnoreCase("Topla")) {
      System.out.println("Anlamadığım bir mesaj geldi");
      return;
    }
    System.out.println("Masajın Adı :" +
                                    message.getMessageName());
    System.out.println("Masajın geldiği sunucu :" +
                                    packet.getSourceAgentHost());
    Object[] parameters = message.getParameters();
    int paramCount = message.getParameters().length;
    System.out.println("Masajın parametre sayısı :" +
                                                    paramCount);
    for (int i = 0; i <= paramCount - 1; i++) {
      //mesaj parametre tipinin String olduğu kabul ediliyor..
      System.out.println("Parametre[" + i + "] :" + (String)
                                                parameters[i]);
    }

    ..........
}


}
..........
}
```

setRepliedMessageHandler: Daha önce de açıklandığı gibi, bir etmen gönderdiği mesajın gelecek olan yanıtını beklemeden yürütülmesi gereken işlemler varsa çalışmasını sürdürebilir; ancak belirli zaman aralıklarında yanıtın gelip gelmediğini kontrol etmelidir (isReplyReady çağrısı ile). Bu çalışma şekli için geliştirilmesi gereken kod oldukça karmaşık olacaktır. “setRepliedMessageHandler” çağrısı ile

daha sade ve etkin bir kod yaratmak mümkün olur. Etmen, herhangi bir mesajına yanıt geldiği zaman yürütülmesini istediği işlemleri yazar ve "setRepliedMessageHandler" metodunu kullanarak bu kod parçasını ait olduğu mesaj ile ilişkilendirir. Mesajın yanıtı ulaştığı zaman sistem doğrudan belirlenen kodu etkin kılacaktır; etmenin ayrıca yanıtın gelişini kontrol etmesine gerek kalmaz.

Bu çağrıyı kullanabilmek için etmen içinde herhangi bir isimde, gelen yanıtları karşılayan yeni bir metod tanımlanır. Metod aşağıdaki formatta olmalıdır.

public void MetodAdı(MessageID id)

"id" parametresi yanıtı gelecek olan mesajın numarasını göstermektedir. "setRepliedMessageHandler" fonksiyonuna "null" parametresi verildiği anda yanıtları karşılayacak kod parçası çalıştırılmaz. Yanıtı gelen her mesajı karşılayacak olan kod ayrı bir iplik içinde çalıştırılır. Bu nedenle, etmen programcısı bu çalışma düzeninin bilincinde olarak karşılıklı dışlamanın gerekebileceği her durum için önlemini almalıdır.

Daha önce "waitForMessage" çağrı örneğinde verilen kod parçası bu işleyiş modeline göre aşağıdaki gibi olacaktır.

```
import agent.*;

public class Main extends Agent {

  public void OnActivate() {

    .......

    getAgentInterface().setRepliedMessageHandler("replyHandler");
    AgentInterface agentinterface = getAgentInterface();
    Enumeration agentlist =
                 agentinterface.getAgentPointer("calculator");
    if (agentlist == null) {
      System.out.println("Sistemde hesaplayici etmen yok.");
    }
   else{
      AgentPointer agent = (AgentPointer)
                                  agentlist.nextElement();
      Message message = new Message("Topla", "123","456");
      MessageID id = agentinterface.sendMessage(agent, message);
   }
  }


public void replyHandler(MessageID id) {
String result = (String)agentinterface.getReply(id);
System.out.println("Yanıt geldi : "+result);
  }

..........
```

```
}
```

GES sunucuları bu işleyişi desteklemek için “JAVA Reflection” özelliğinden faydalanırlar. Eğer etmen, yanıt mesajları karşılamak üzere bir metod ismi tanımlamış ise, etmenin gönderdiği bir mesajın yanıtı geldiği zaman, kabuk işlevleri bu metodu “Reflection” ile ayrı bir iplik içinde çalıştırırlar. Beklenen yanıt geldiği anda sunucu tarafından çalıştırılan metod kodu aşağıda verilmiştir.

```
public void executeRepliedMessageHandler(final MessageID messageid)
   {
     Thread handlerThread = new
                        Thread("Replied Message Handler Thread"){
         public void run()
          {
           Object[] rgobj = new Object[1];
           rgobj[0] = messageid;
           Method method = null;
           Class[] paramClasses = new Class[1];
           paramClasses[0] = messageid.getClass();
           try {
              method = (Method)getAgent().getTarget().
            getClass().getMethod(repliedMessageHandler,
                                                    paramClasses);
            method.invoke(getAgent(), rgobj);
           } catch (Exception ex) {
           Utils.writeErrorLog("Error while invoking
                                    agent replied message handler");
           Utils.logFullTrace(ex);
           }
           }
           };
           handlerThread.start();
   }
```

Eğer belirtilen formatta bir metod ismi, yanıtları karşılayacak olan metod ismi olarak verilmemiş ise hata alınacak, sonuç olarak yanıtları karşılayan kod çalıştırılmayacaktır.

6.1.1.3 Göç Etmeye Yönelik Çağrılar

move : Etmen göç etme isteğini bu metodu çağırarak iletir. Giriş parametresi göç edilecek olan etmen sunucusunun adresidir. Dönüş parametresi ise “TransferResponse” sınıfından bir nesnedir ve göç isteğinin başarılı olup olmadığını sorgulamak için kullanılır. Bu çağrıdan etmene ait bütün yapıların hedef sunucuya iletilene kadar geri dönülmez, yani etmen bloke olur. Çağrı başarısız olursa etmen çalışmasına devam edebilir. Eğer başarılı şekilde etmen gönderilirse yerel etmen pasif duruma geçirilir ve yerel sunucu üzerinden silinir. Programcı bu çalışma modeline göre etmeni programlamalıdır. Örneğin başarılı biten “move” çağrısı

sonunda etmenin yerel GES sunucu ile etkileşimi olabilecek herhangi bir kod yazılmamalıdır. Aksi durumda bu koda bellekte çalışmaya devam eden ancak bu durum GES için herhangi bir anlam ifade etmeyecektir ve GES sunucusunun yakalayabildiği bütün aktiviteler hata (exception) alacaktır.

Bir etmen bir düğümden diğerine transfer edilirken şu aşamalardan geçer:

- Etmenin çalışması durdurulur (pasif duruma geçirilir), "OnInactivate" metodu çalıştırılır.
- Etmenin o ana kadar edindiği durum bilgisi (değişkenler) saklanır.
- EEtmen kodu, verisi ve politikası uzak GES sunucuya iletilir. Bu iletim şifreli olaran gerçeklenir.Dolayısı ile ağın izlenerek etmen kodu yada verisinin elde edilmesi mümkün değildir.
- Etmen yerel GES sunucu üzerinden silinir. Etmen kabuğu, etmen giriş çıkış kuyukları, bu kuyrukları dinleyen iplikler, etmen politikası, disk üzerindeki etmen kodu ve son durumu silinir.
- Etmen uzak GES sunucuda aktif duruma geçirilir.

Uzak GES sunucudan transfer isteği alan yerel GES sunucu, uzak GES sunucudan transfere başlatmak üzere uzak GES sunucunun "beginTransfer" metodunu çağırır. Transfere başla mesajına yanıt olumlu ise sırasıyla etmene ait kodu, problem yoksa durumunu ve yine problem yok ise politika dosyasını transfer etmesini sağlayan uzak GES sunucunun "transferResource", "transferData" ve "transferPolicy" metodlarını çağırır. Transfer bittiğinde uzak GES sunucu, etmene ait kod ve veriyi siler, yerel GES sunucu ise etmeni yaratarak aktif duruma getirir ve etmen verileri güncel olacak şekilde çalışmasına yeniden başlar.

Etmenlerin göç etme aşamasında aktif-pasif ve pasif-aktif durum geçişlerinde sunucuların bu bilgiyi "Gözleyici" düzende çalışan etmen sunucusuna (GGES) ilettiğini unutmamak gerekir. Böylece konumu değişen etmenin güncel yer bilgisi korunmuş olur. Göç etme komutunu bir mesajla alan etmenin move çağrısına örnek kod aşağıda verilmiştir.

```
import agent.*;

public class Main extends Agent {
```

```
  public void OnMessageArrive() {

    MessagePacket packet = getAgentInterface().receive();
    Message message = packet.getMessage();
    if (message.getMessageName().equalsIgnoreCase("Start")) {
      ....
    }
    else
    if (message.getMessageName().equalsIgnoreCase("Stop")) {
      ....

    }
    else
    if (message.getMessageName().equalsIgnoreCase("Move")) {
      Object[] parameters = message.getParameters();
      String hosttoMove = (String) parameters[0];
      TransferResponse moveResult =
                          getAgentInterface().move(hosttoMove);
     if (moveResult.wasSuccessful()) {
               break;
             }
      else {System.out.println("Göç etme başarısız... "}
    }

    else {
      System.out.println("Message couldn't be resolved : " +
                                      message.getMessageName());
    }

  }

  public void OnTransfer() {
    System.out.println("I am the counter agent and transferring");
  } //move çağrısı yapıldığı anda bu metod çalışır.

.....
}
```

Yukarıdaki etmen kodunda, etmen “Move” isimli bir mesaj aldığında mesajın ilk parametresinin de taşınacağı sunucunun adresi olduğunu varsayarak göç etme isteğinde bulunmaktadır. “TransferResponse” sınıfından olan geri dönüş nesnesi göçün başarılı olduğunu bildiriyor ise, artık etmen bu aşamadan sonra herhangi bir kod işletmemelidir çünkü yerel sunucu üzerinde artık aktif durumda değildir. Bu kontrolü “isAgentAlive” çağrısı yaparakta anlayabilir.

6.1.1.4 Çoğalmaya Yönelik Çağrılar

createClone : Etmenler, herhangi bir anda kendi kopyalarını yaratabilirler (Cloning). Kopya yaratma istediğinde bulunan bir etmenin kodu, son durumu ve politikası ile aynı, ancak yeni bir kimlik bilgisine sahip olan yeni bir etmen aktif duruma getirilir. Çağrı herhangi giriş parametresi almaz, çıktı olarak yeni oluşturulan kopya etmene bir işaretçi döner. Eğer geri dönüş parametresi “null” ise kopyalama işlemi başarısız olmuş demektir.

```
import agent.*;

public class Main extends Agent {


  public void OnMessageArrive() {

    MessagePacket packet = getAgentInterface().receive();
    Message message = packet.getMessage();
      if (message.getMessageName().equalsIgnoreCase("Clone")) {

      System.out.println("Counter Agent: clone message arrived");
      AgentPointer clone = getAgentInterface().createClone();
      if (clone != null) {
        System.out.println("cloning successful");
      }
      else {
        System.out.println("cloning not successful");
      }

      }
  }
.....
}
```

7. GÜVENLİ ETMEN SİSTEMİ MİMARİSİ

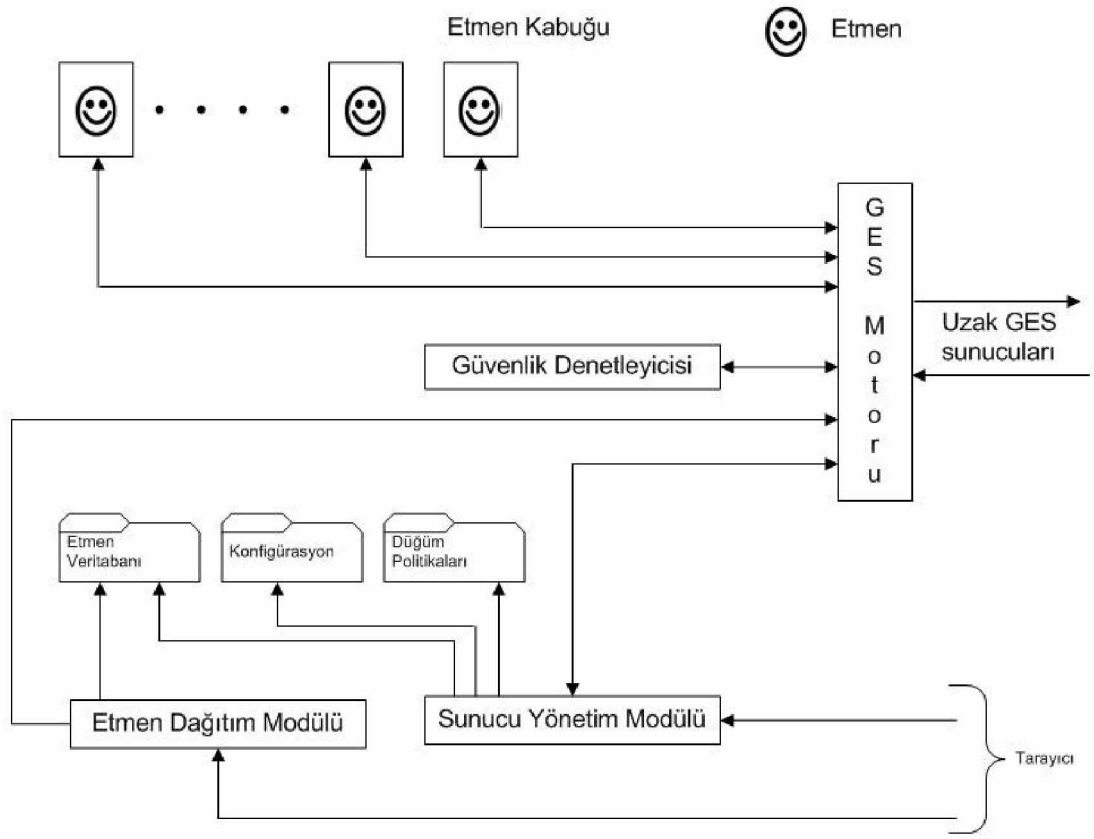
Bu bölümde GES mimarisinin ana bileşenleri ayrıntıları ile açıklanacaktır.

7.1 GES Sunucusu

GES mimarisinin ana bileşeni, etmen kabul edecek her düğüm üzerinde kurulu ve çalışır durumda olması gereken GES sunucusudur. JAVA dilinde geliştirilmiş olan GES sunucusu, etmen yaratma, çalıştırma, etmen haberleşmesi ve etmen göçü gibi temel fonksiyonları sağlar. Oldukça karmaşık sayılabilecek fonksiyonları olmasına rağmen modüler bir yapıya sahip olması nedeniyle yeni fonksiyonlar eklenmeye olanak veren bir özelliğe sahiptir.

GES sunucusu, ayrıntıları daha sonra incelenecek olan üç farklı düzende çalışabilir. Bunlardan standart düzende çalışanın görevi, etmen yaratma, çalıştırma, yok etme, etmen haberleşmesini sağlama ve etmen göçü gibi temel etmen fonksiyonlarını yerine getirmektir. Diğer iki düzen, Güvenlik Yönetici düzen (GYGES) ve Gözleyici düzendir (GGES). Şekil 7.1.1 Standart düzende çalışan bir GES sunucunun ana bileşenlerini göstermektedir.

Sunucunun ana bileşeni GES motorudur. GES motoru, sunucunun diğer sunucular ile olan haberleşmesini sağlayan düşük seviyedeki temel fonksiyonları içerir. Bu fonksiyonlar gerektiğinde etmen kabukları tarafından kullanılabilir ya da sunucunun, örneğin güncel etmen listesinin GGES'e iletilmesi gibi (ayrıntıları ilerleyen bölümde verilecektir) amaçları için kendi isteği doğrultusunda kullanılabilir.



Şekil 7.1.1 : GES sunucusu ana bileşenleri

Güvenlik Denetleyicisi: Yürütülen etmen aktivitelerinin (mesajlaşma, göç etme, diske yazma, soket bağlantısı kullanma, vb.) güvenlik politikalarına uygun olup olmadığını kontrol eder. GES motoru fonksiyonları bu türden aktiviteleri yerine getirmeden önce isteği Güvenlik Denetleyicisine iletir ve aldığı yanıta göre sadece izin verilen aktivitelerin yürütülmesini sağlar.

Etmen Kabuğu: Etmeni bir kabuk gibi sarmalayıp, diğer etmenlerden ya da yazılımlardan kaynaklanabilecek doğrudan erişimleri engelleyerek etmeni korur. Etmen Kabuğu etmenin dış dünya ile iletişimi için bir arayüz sunar. Ayrıca etmen giriş çıkış mesaj kuyruklarının yönetiminden de sorumludur.

Etmen Veritabanı: Etmenlerin saklanan son durumları, kodu ve politikasının tutulduğu disk alanıdır. Bu bilgilerin hepsi şifreli olarak disk üzerinde tutulur.

Konfigürasyon: Etmen sunucusuna ait konfigürasyon bilgisinin tutulduğu alandır. Örneğin sunucunun hangi düzende çalıştığı, ortaklık ilişkisi içinde olduğu diğer sunucuların adresleri, dinleyen tcp port numaraları gibi ayarlar burada bulunur.

Düğüm Politikaları: Düğüm yöneticisinin düğümler için oluşturduğu politikaların tutulduğu disk alanıdır.

Sunucu Yönetim Modülü: Sunucunun uzaktan web tarayıcı ile yönetilmesi için gerekli olan fonksiyonları barındırır. Etmen programcısı veya düğüm yöneticisi, etmenleri ve sunucuyu uzaktan bu bileşen aracılığı ile izleyebilmekte ve yönetebilmektedir.

Etmen Dağıtım Modülü: Uzaktan etmen programcısının yazdığı etmeni tarayıcı aracılığı ile sunucu üzerine dağıtma görevini yerine getirir.

GES sunucu mimarisi katmanlı bir yapıya sahiptir. Bu özellik, katmanların ayrı ayrı programlanarak yeteneklerinin geliştirilebileceği anlamına gelir. Katmanlar birbirlerinin gerçekleme ayrıntıları ile ilgilenmezler ancak birbirlerine sundukları ara yüzleri ve formatlarını tanırlar. Şekil 7.1.2 GES'in katmanlı mimarisini göstermektedir.

Şekil 7.1.2 : GES'in katmanlı mimarisi

7.1.1 GES Motoru

GES sunucusunun ana bileşeni GES motorudur. Uzak sunucular ile olan tüm iletişimler GES motoru aracılığı ile sağlanmaktadır. GES motorunun görevlerinin başlıcaları şunlardır:

- Uzak GES sunucular ile olan iletişimleri gerçeklemek
- Etmen için etmen kabuğu yaratarak kontrolü kabuğa aktarmak

- Etmenler (dolayısı ile kabuklar) arası iletişimi gerçeklemek
- Etmen göçlerini sağlamak

GES motoru GES sunucusunun içinde çalıştığı düzene bağlı olarak oldukça karmaşık sayılabilecek görevleri yerine getirir. Standart düzende çalışan bir GES sunucusunda GES motoru yukarıda sayılan görevleri yerine getiriyor olmasının yanında, kimlik denetim bilgilerinin tutulması, standart GES sunucuları kimlik denetiminden geçirme, etmen şifreleme anahtarlarını tutma ve dağıtma, güncel etmen listesini tutma ve dağıtma gibi daha bir çok görevi yerine getirmektedir.

Tablo 7.1.1.1, GES motoru fonksiyonlarını ve ne amaçla kullanıldıklarını içermektedir.

Tablo 7.1.1.1 GES motoru fonksiyonları

Metod	**Kullanım Amacı**
public Hashtable getAllAgents(Ticket tk)	GGES ten güncel etmen listesi istenir.
public void registerAgenttoMasterBrowser(Ticket tk, AgentIdentity agentidentity)	GGES'e aktif olan bir etmenin kimliği bildirilir.
public void unregisterAgentfromMasterBrowser(Ticket tk, AgentIdentity agentidentity)	GGES'e pasif olan bir etmenin kimliği bildirilir.
public boolean serverUP(Ticket tk,String strAgentHostName)	Herhangi bir GES sunucunun çalışıp çalışmadığı kontrol edilir.
public boolean pingSM(String destinationSM)	Herhangi bir GYGES'in çalışıp çalışmadığı kontrolü yapılır.
public boolean _acceptMessage(Ticket tk, MessagePacket packet)	Uzak GES sunucuda çalışan etmenin yolladığı mesaj alınır.
public void _acceptReply(Ticket tk,MessagePacket packet)	Uzak GES sunuya daha önce gönderilen mesajın yanıtı alınır.
public void requestToTransfer(Ticket tk, AgentHost agenthost, Voucher voucher, AgentIdentity agentidentity)	Etmenin uzak GES sunucuya transfer isteği bildirilir.

public void beginTransfer(Ticket tk, Voucher voucher, AgentIdentity agentidentity)	Etmenin uzak GES sunucuya transferi başlatılır.
public void endTransfer(Ticket tk,Voucher voucher, AgentIdentity agentidentity)	Etmenin uzak GES sunucuya transferi bitirilir.
public byte[] transferResourceFile(Ticket tk, Voucher voucher, AgentIdentity agentidentity)	Etmenin kodu uzak GES sunucuya iletilir.
public byte[] transferDataFile(Ticket tk,Voucher voucher, AgentIdentity agentidentity)	Etmenin durumu (verisi) uzak GES sunucuya iletilir.
public byte[] transferPolicyFile(Ticket tk,Voucher voucher, AgentIdentity agentidentity)	Etmenin politikası uzak GES sunucuya iletilir.
public Ticket authenticateServer(String host,String key)	GES sunucu GYGES ten kimlik denetimi yapar.
public boolean validateTicket(Ticket tk)	GES sunucu herhangi bir bileti GYGES ten onaylar.
public boolean validateTicketforSMPartner(String partner, String authKey, Ticket tk)	GYGES sunucu kendisinde olmayan bir bileti ortağı olan GYGES ten onaylamasını ister.
public Policy getRemoteAgentPolicy(Ticket tk, AgentIdentity agentidentity)	Uzak bir GES sunucuda çalışan etmenin politikası alınır.
public boolean setRemoteAgentPolicy(Ticket tk, Policy policy,AgentIdentity agentidentity)	Uzak bir GES sunucuda çalışan etmenin politikası değiştirilir.
public String getAgentEncryptionKeyFromSM(Ticket tk)	GES sunucu Şifreli etmenlerin kod, veri ve politikasını çözecek anahtarı GYGES ten ister.

7.1.1.1 GES Sunucu Güvenli İletişim Protokolü

Çalıştığı düzen ne olursa olsun GES sunucular arasındaki tüm iletişimler “SSL over RMI”[26] adı verilen protokol ile gerçekleştirilir. SSL protokolünün çalışması için de sertifika gerektiğinden, her GES sunucusunun standart X.509[39] sertifika

yaratmaları için gerekli fonksiyonlar mevcuttur. GES sunucuyu yöneten programcı yada yönetici yönetim modülü ile sunucu sertifikasını yaratabilir. Diğer bir GES sunucu ile sertifikalar kullanılarak güvenli iletişimi sağlamak için ise haberleşeceği GES sunucunun sertifikasının yerel sunucu üzerinde "güvenilir" nitelikte tanımlanmış olması gerekmektedir. Bu işleyiş aşağıdaki adımlarla yürümektedir. Haberleşecek sunucuları A ve B sunucu olarak adlandırırsak;

- A sunucusu herkese açık (public) ve özel (private) anahtar ikilisi oluşturarak bunu konfigürasyon dosyasına kaydeder. Oluşturulan anahtarlar 1024 bitlik RSA[40] anahtarlarıdır.

- A sunucusu yarattığı herkese açık anahtardan sunucu bilgilerini de içeren imzalanmış sertifikası olan x .509 sertifikasını oluşturarak bunu ihraç eder ve B sunucusuna iletir.

- B sunucusu A dan almış olduğu setifikayı "güvenilir sertifikalar" listesine ekleyerek saklar.

- Aynı şekilde A sunucusu da B nin yaratıp, imzalamış ve ihraç etmiş olduğu sertifikasını kendi "güvenilir sertifikalar" listesine ekler.

- Bu andan itibaren A ve B sunucuları SSL üzerinden haberleşebilir durumdadırlar. Bu adımlardan herhangi birinin eksik olması sunucular arasındaki iletişimim başlayamamasına neden olur.

RMI, ulaşım katmanı protokolü için "tcp" kullanır. JAVA dili ulaşım katmanı için yaratılan soket nesnesinin değiştirilmesine izin vermektedir. Yapılan değişikliklerle ulaşım katmanı için SSL üzerinden haberleşmeyi sağlayacak özel bir socket yaratılmış ve RMI'ın haberleşmeyi bu yeni soket üzerinden yapması için gerekli kodlar yazılmıştır. İstemci ve sunucu için yazılan yeni soket sınıfları ve kodları aşağıda verilmiştir. Yaratılan yeni soketler bu sınıflar kullanılarak oluşturulur.

RMISSLClientSocketFactory.java

```
package server;

import java.io.IOException;
import java.io.Serializable;
import java.net.Socket;
import java.rmi.server.RMIClientSocketFactory;

import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
```

```
public class RMISSLClientSocketFactory implements
RMIClientSocketFactory, Serializable {

    public Socket createSocket(String host, int port) throws
IOException {
        SSLSocketFactory factory =
                (SSLSocketFactory) SSLSocketFactory.getDefault();
        SSLSocket socket = (SSLSocket) factory.createSocket(host,
port);
        return socket;
    }
}
```

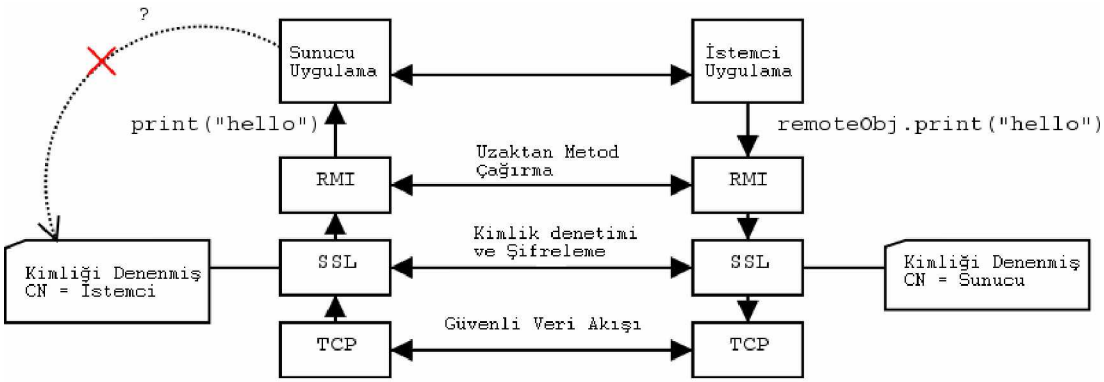
İstemci tarafında oluşturulacak olan yeni soket, JAVA nın hazır sağlamış olduğu "SSLSocketFactory" sınıfından olan yeni bir soket nesnesidir; bunun dışında ek bir iş gerekmez. Ancak sunucu için oluşturulacak soket nesnesi için daha karmaşık kod gerekmektedir, çünkü sunucu öncelikle istekte bulunan istemcinin güvenilen bir sertifikaya sahip olup olmadığını kontrol etmelidir.

RMISSLServerSocketFactory.java

```
package server;

import java.io.*;
import java.net.ServerSocket;
import java.rmi.server.RMIServerSocketFactory;
import java.security.KeyStore;

import javax.net.ssl.*;
import utility.*;

public class RMISSLServerSocketFactory implements
RMIServerSocketFactory, Serializable {

    public ServerSocket createServerSocket(int port) throws
IOException {
        SSLServerSocketFactory ssf = null;
        try {

            SSLContext ctx;
            KeyManagerFactory kmf;
            KeyStore ks;

            char[] passphrase =
Globals.currentServerParameters.getParameterValue("SecurityDbPasswor
d").toCharArray();
            ctx = SSLContext.getInstance("TLS");
            kmf = KeyManagerFactory.getInstance("SunX509");
            ks = KeyStore.getInstance("JKS");
            ks.load(new FileInputStream(Globals.securityDBFileName),
passphrase);
            kmf.init(ks, passphrase);
            ctx.init(kmf.getKeyManagers(), null, null);
            ssf = ctx.getServerSocketFactory();
        } catch (Exception e) {
            Utils.logFullTrace(e);
```

```
        }
        return ssf.createServerSocket(port);
    }
}
```

RMI'ın SSL protokolü üzerinden çalışma şekli şekil 7.1.1.1.1 de gösterilmiştir.



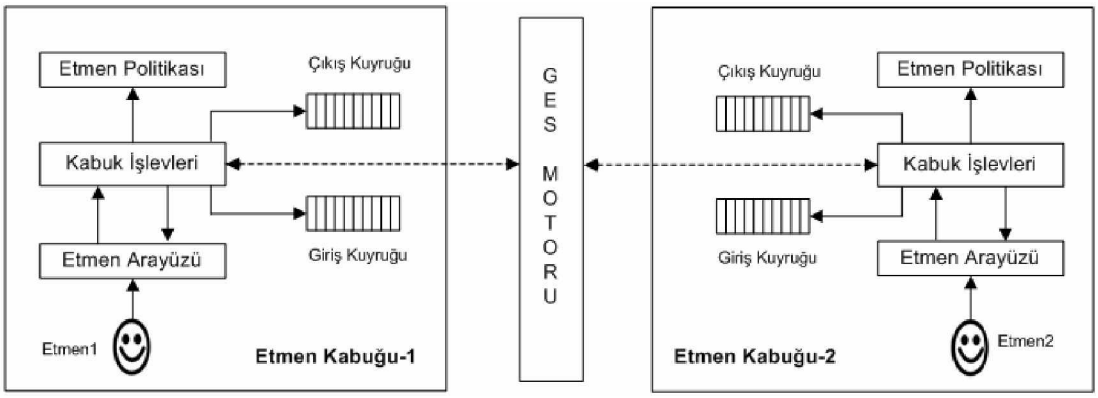
Şekil 7.1.1.1.1 : SSL üstünde RMI çalışması

7.1.2 Etmen Kabuğu

Hareketli etmen sistemlerinde güvenlikten bahsederken, risklerden birinin de, etmenin çalıştığı ortamdaki diğer etmenler tarafından kötüye kullanılması olduğu belirtilmişti. GES mimarisi etmeni, çevresindeki diğer etmenlerden koruyacak gerekli mekanizmalara sahiptir. Bu amaçla GES sunucusu, bir etmeni aktif duruma geçirmeden önce "Etmen Kabuğu" (AgentShield) isimli bir nesne yaratır. "Sarmalanmış Etmen Modeli" adını verdiğimiz bu modelde her etmen bir etmen kabuğu tarafından korunur. Etmen, etmen kabuğu nesnesinin özel bir değişkeni olarak tanımlanıp, dışarıdan gelecek erişimlere karşı korunur. Etmen kabuğunun başlıca görevleri şunlardır.

- Etmeni sarmalayarak etmene doğrudan erişimi engellemek.
- Etmen arayüzü ile etmenin ilettiği istekleri gerçekleştirmek.
- Etmen için mesaj giriş ve çıkış kuyrukları oluşturmak.
- Etmen politikası ile etmeni ilişkilendirmek.
- Etmen kodu, durumu ve politikasını şifrelemek ve çözmek.

Sarmalanmış Etmen Modeli şekil 7.1.2.1 de gösterilmiştir.



Şekil 7.1.2.1 : Sarmalanmış etmen modeli

Etmen kabukları arasında doğrudan iletişim bulunmamaktadır, tüm istekler GES motorundan geçmelidir.

Etmen kabuğunun temel görevi olan etmeni çevresinden gelebilecek doğrudan erişimlere karşı koruma dışında mesajlaşma ile ilgili de önemli görevleri vardır. Bu ayrıntılar "Mesajlaşma Altyapısı" başlığı altında incelenecektir.

Etmen kabuğu, etmen kodu ve durumunun şifrelenerek diske yazılmasından da sorumludur. Etmen kod ve durum bilgisi DES[40] protokolü kullanılarak şifrelenir. Gerekli olan anahtar ise "Güvenlik Yöneticisi" düzende çalışan GES sunucudan alınır. Şifreleme fonksiyonları için javanın **javax.crypto** ve **javax.crypto.spec** paketlerinden yararlanılmıştır. Aşağıda etmen kodunu şifrelemek için kullanılan metod bulunmaktadır; bu metodun geri dönüş değeri şifrelenmiş etmen kodudur. Aynı metod mimari içinde şifelenmeye ihtiyaç duyan her türlü ikili veri için kullanılmaktadır. Giriş parametreleri DES algoritması için kullanılacak anahtar ve şifrelenecek ikili dizisidir.

```
public static ByteArrayOutputStream
            encryptBinaryData(String enckeystr, byte[] plainin) {

        ByteArrayInputStream is = null;
        ByteArrayOutputStream os = new ByteArrayOutputStream();
        byte enckey[] = enckeystr.getBytes();

        try {
            CipherInputStream cis;
            SecretKeySpec secretKey = new SecretKeySpec(enckey,
                                                        "DES");
            Cipher encrypt =
                        Cipher.getInstance("DES/ECB/PKCS5Padding");
            encrypt.init(Cipher.ENCRYPT_MODE, secretKey);

            try {
                is = new ByteArrayInputStream(plainin);
```

```
            } catch (Exception err) {
                System.out.println(err.toString());
            }
            cis = new CipherInputStream(is, encrypt);
            os = new ByteArrayOutputStream();
            byte[] b = new byte[8];
            int i = cis.read(b);
            while (i != -1) {
                os.write(b, 0, i);
                i = cis.read(b);
            }
            cis.close();
            is.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return os;

    }
```

Etmenin disk ortamından okunup aktif duruma geçirilmesi gerektiğinde de şifreli verinin çözülmesi gerekmektedir. Şifre çözme amacıyla kullanılan metodun kodu ise aşağıdaki gibidir. Giriş parametresi olarak şifreli veri ve DES algoritmasına girecek olan anahtar kullanılmaktadır. Dönüş parametresi ise şifresi çözülmüş olan ikili veridir.

```
public static ByteArrayOutputStream
            decryptBinaryData(String strkey, byte[] encryptedin) {

        byte key[] = strkey.getBytes();
        ByteArrayInputStream is = new
                              ByteArrayInputStream(encryptedin);
        ByteArrayOutputStream os = new ByteArrayOutputStream();
        try {
            SecretKeySpec secretKey = new SecretKeySpec(key, "DES");
            Cipher decrypt =
                        Cipher.getInstance("DES/ECB/PKCS5Padding");
            decrypt.init(Cipher.DECRYPT_MODE, secretKey);
            CipherInputStream cis =
                              new CipherInputStream(is, decrypt);
            byte[] b = new byte[8];
            int i = cis.read(b);
            while (i != -1) {
                os.write(b, 0, i);
                i = cis.read(b);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return os;
    }
```

Varsayılan olarak Java Virual Machine (JVM), herhangi bir sınıf dosyasını yüklemek için ".class" uzantılı dosya arar. Bu dosya, JVM'in çalıştırabileceği "ikili kod" (byte code) adı verilen formata sahiptir. Sınıf dosyaları disk üzerinde dosya_adi.class

şeklinde tutulur ve JVM, "dosya_adi" isimli sınıfı yükleyeceği zaman, "dosya_adi.class" fiziksel dosyasından ikili kodu okuyarak belleğe yükler. Eğer bu klasik sınıf yükleme metodu değiştirilecekse, JAVA programcıya kendi yazdığı sınıf yükleyicisini kullanabilme olanağı sunar. Kabuk, yaratılıp etmenle ilgili görevleri üstlendiği zaman öncelikle gerekli yapıları oluşturup, etmeni aktif duruma geçirmekle görevlidir. Ancak bunun için şifreli ve sıkıştırılmış durumdaki etmen kodunu belleğe yükleyebilecek bir sınıf yükleyicisine ihtiyaç vardır, çünkü JAVA'nın sağladığı standart sınıf yükleyicisi bu özelliği desteklememektedir. GES, şifreli ve sıkıştırılmış etmen kodunun belleğe yüklenmesi için özel olarak geliştirilmiş yeni bir sınıf yükleyicisi kullanır.

Yeni bir sınıf yükleyicisi oluştururken JAVA'nın "ClassLoader" sınıfından türeyen yeni bir sınıf oluşturulur. Bu yeni sınıfta yeniden yazılması gereken tek metod ise "Loadclass" metodudur. "LoadClass" metodunda sırasıyla şu adımlar işletilir.

- Sınıf ismi doğrulanır
- Belirtilen sınıfın hali hazırda yüklü olup olmadığı kontrolü yapılır
- Sınıfın bir sistem sınıfı olup olmadığı kontrol edilir
- JVM için sınıf tanımlanır
- Sınıf içinde referans edilen başka sınıflar var ise belirtilir
- Sınıf çağırana geri döndürülür

Güvenli Etmen Sistemi içinde geliştirilen şifreli ve sıkıştırılmış etmen kodu yükleyen "AgentClassLoader" bu adımların tümünü gerçekler.

```
Final class AgentClassLoader extends ClassLoader {

.....

}
```

Bu yeni sınıf içinde etmeni yükleyen metod ise şu şekilde yazılmıştır.

```
    private final Class
            loadAgentClass(String strClassName) throws
                                    ClassNotFoundException {
        Class c = null;

      //class zaten yüklenmişse hiç bir şey yapma
        if ((c =
```

```
            (Class) hashtableClasses.get(strClassName)) != null) {
            return c;
        }

      //Sıkıştırılmış dosya içinden etmene ait şifreli kod ve durum
bilgisini oku.

        String strFileName = transform(strClassName);
        ZipEntry zipentry = zipfile.getEntry(strFileName);

        if (zipentry == null) {
            throw new ClassNotFoundException(strClassName);
        }

        byte[] rgb = null;

        try {
            int n = (int) zipentry.getSize();
            rgb = new byte[n];
            InputStream inputstream =
                            zipfile.getInputStream(zipentry);

            int m = 0;
            while (m < n) {
                m += inputstream.read(rgb, m, n - m);
            }
        } catch (IOException ex) {
            throw new ClassNotFoundException(strClassName);
        }


      //şifreli etmen class dosyasını çöz

        byte[] rgb1 = null;
        try {
            String strkey =
Globals.currentServerParameters.getParameterValue
                                          ("AgentEncryptionKey");
rgb1 = Utils.decryptBinaryData(strkey, rgb).toByteArray();
        } catch (Exception ex) {
            System.out.print(ex.toString());
            return null;
        }

        c = defineClass(strClassName, rgb1, 0, rgb1.length);
        hashtableClasses.put(strClassName, c);

      //Çağırana class'ı geri dön.
        return c;
    }
```

7.2 Güvenlik Yöneticisi GES

GES sunucu üç farklı düzende çalışmayı destekler. Her GES sunucusu standart düzende çalışan bir GES sunucunun (SGES) desteklediği fonksiyonları sağlar. Bununla birlikte istenirse bir GES sunucu, "Gözleyici" düzen veya "Güvenlik

Yöneticisi" düzen de ya da her iki düzende birlikte çalışacak şekilde de ayarlanabilir. "Güvenlik Yöneticisi" düzen (GYGES) ve "Gözleyici" düzen (GGES) sistemde özel görevleri olan GES sunucular için kullanılır.

Herhangi bir GES sunucu, başarılı bir şekilde başlayıp etmenlere ev sahipliği yapmadan önce kimlik denetiminden geçmek zorundadır. Bu kimlik denetimini kendisi için düğüm yönetici tarafından tanımlanmış olan GYGES ten karşılar. Her GES sunucu için birden fazla GYGES tanımı yapılabilmesine rağmen aynı anda aktif olan sadece bir GYGES olabilir. Aktif olan GYGES'e ulaşılamaz ise GES sunucu güvenlik ihtiyaçları için tanımlanmış olan diğer GYGES sunuculara bağlanmaya çalışır. Kimlik denetiminden geçebilen GES sunucuları artık uzak sunucular ile iletişime geçebilir ve etmenlere ev sahipliği yapabilir. Güvenlik Yöneticisi düzen de çalışan bir GES sunucunun görevleri şunlardır.

- GES sunucuları kimlik denetiminden geçirmek
- İletişim içinde olan iki GES sunucu için güvenilir otorite görevini üstlenerek ortak kimlik denetimi işlevini yerine getirmek

Bu görevleri yerine getirebilmek için GYGES, her GES için bir kimlik denetim anahtarı tanımlar. Bir GES sunucu kimlik denetiminden geçebilmek için GYGES'e kendisi için belirlenmiş olan bu anahtarı sunmak zorundadır. Anahtar en az 20 karakter uzunluğunda katar tipinde bir veridir. GYGES'in, bir GES için kimlik denetim anahtarı tanımlarken kullandığı anahtar yaratma algoritması aşağıdaki gibidir.

```
public String generateKey() {
        String strkey = null;
        String strAlphabet =
"AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz0123456789><()/
.;[]#$!%-+*";
        char[] rgc = new char[20];
        for (int i = 0; i < rgc.length; i++) {
            int n = (int) (Math.random() * (double)
                                          strAlphabet.length());
            rgc[i] = strAlphabet.charAt(n);
        }
        strkey = new String(rgc);
        return strkey;
    }
```

GYGES kimlik denetiminden geçirdiği her GES'e belirli bir yaşam süresine sahip olan bir bilet (ticket) verir. Bileti alan GES uzak GES sunucular ile olan haberleşmesinde bu bileti de sunmak zorundadır. Uzak GES sunucusu kendisine gelen bir isteğin sistemde kimlik denetiminden geçmiş bir GES sunucuya ait olup

olmadığını anlamak için bileti GYGES'e yollar ve gerçekten GYGES'in bu bileti verip vermediğini kontrol eder. GYGES, bileti verdiğini onaylar ise isteği karşılar aksi durumda karşılamaz. Bilet yapısı şekil 7.2.1 de görüldüğü gibi tanımlanmıştır.

Bilet ID	Yaratılma Tarihi	Sahibi	Veren GYGES Sunucu	Yaşam Süresi

Şekil 7.2.1 : Bilet

Bilet ID: Her bilet için tekil bir belirteçtir. Rastgele üretilir

Yaratılma Tarihi: Biletin yaratılıp GES sunucuya verildiği tarihtir

Sahibi: Bileti alan GES sunucunun adresi

Veren GYGES Sunucu: Bileti veren Güvenlik Yöneticisi GES'in adresi

Yaşam Süresi: Biletin ne kadar süre ile geçerli olduğudur

Bilet sınıfı aşağıdaki gibi tanımlanmıştır.

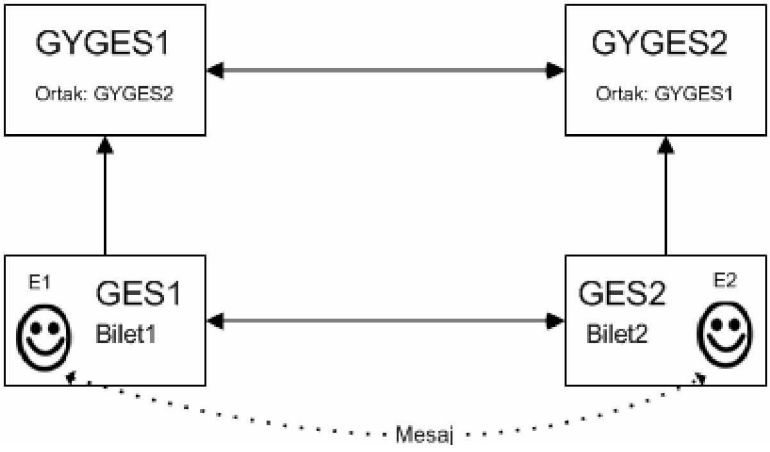
```
public class Ticket implements Serializable {

    private MessageID ID = null;
    private Date beginTime = null;
    private String Owner = null;
    private String givenBy = null;
    private long refreshTime = 300;

.......
}
```

Her GES sunucu bileti aldıktan sonra, yaşam süresi dolmadan yeni bir bilet almak zorundadır aksi durumda uzak GES sunucular ile haberleşemez. GES sunucu, bilet yaşam süresinin 4 te 3 ü geçtiği andan itibaren GYGES ten yeni bir bilet isteğinde bulunur. Eğer başarısız olunursa (Örneğin GYGES'e ulaşamıyordur) varsayılan olarak 10 sn süreyle yeniden dener.

7.2.1 Güvenlik Yöneticisi GES Sunucular Arası Ortaklık İlişkisi

GES mimarisi sistemin hataya dayanıklı olması ve sürekliliği sağlamak üzere, sistemde birden çok GYGES'in etkin olmasına olanak sağlamaktadır. İki standart GES sunucu, GYGES olarak farklı sunucuları kullanıyor olabilir. Bu durumda kimlik denetiminden geçtikten sonra iki standart GES'in aldıkları biletler farklı sunucular tarafından verilmiş olacaktır. Bu iki sunucu üzerinde çalışan iki etmen birbirlerine

mesaj yolladıkları zaman, mesajların gerçekten kimlik denetiminden geçmiş bir GES sunucudan gelip gelmediği kontrolü yapılacaktır ki bu durumda bu kontrolün hangi GYGES üzerinden yapılacağı problemi ortaya çıkmaktadır. Mimari bu problemin çözümü için GYGES ler arasında "ortak" çalışma anlayışını kullanır. Bu çalışma modeline göre ortak bir anahtarı paylaşan iki veya daha çok GYGES birbirlerinin ortağı olabilir. Hangi GYGES sunucuların birbirleri ile ortaklık ilişkisi kuracağını düğüm yöneticisi belirler. Bir GYGES bir onaylama isteği yerine getiremiyor ise, bu isteği ortaklarına yönlendirebilir. GYGES, herhangi bir ortağından yönlendirilmiş bir istek söz konusu ise bu isteği karşılamaya çalışır, ancak olumsuzluk durumunda ölümcül kilitlenmelere yol açmamak için isteği tekrar kendi ortaklarına yönlendirmez. Bu çalışma modeline örnek olarak farklı GYGES leri kullanacak iki GES sunucusu üzerindeki etmenlerin birbirlerine mesaj gönderdiklerini farzedelim. (şekil 7.2.1.1)



Şekil 7.2.1.1 : Farklı GYGES ler kullanan GES sunucular

- Etmen1 (E1), etmen kabuğu tarafından verilen arayüz aracılığı ile etmen2 ye (E2) mesaj iletme isteğini bildirir.

- Mesaj GES1 tarafından etmen 2 nin çalıştığı GES2 ye iletilir. Mesaj paketi içinde kimlik denetimi sonucu GYGES1 den aldığı bilet de iletilir.

- GES2 kendisine gelen bu isteği karşılamadan önce mesajı gönderen GES1'in biletini okur ve bileti onaylaması için GYGES2 ye iletir.

- GYGES2 biletin kimin tarafından verildiği bilgisini sınar (bilet içinde bu bilgi açık olarak yer alır). Eğer kendisinin verdiği bir bilet ise, daha önce dağıttığı ve henüz yaşam süresi dolmamış biletler arasında

bu biletin yer alıp almadığına kontrol eder; var ise isteği onaylar, yok ise onaylamaz.

- Eğer GYGES2 bileti başka bir GYGES'in verdiğini belirlerse (örnek durumda bu GYGES1 dir), bileti veren GYGES ile ortak olup olmadığını kontrol eder. Tanımlı ortaklar arasında bileti veren GYGES yer alıyor ise isteği ortağına, onunla paylaştığı anahtar ile birlikte yönlendirir.

- İsteği alan ortak GYGES (GYGES1) öncelikle isteği yönlendiren GYGES (GYGES2) ile ortak olup olmadığını ve ortak iseler, isteğin doğru anahtar ile birlikte iletilip iletilmediğini kontrol eder.

- Eğer istek tanımlı bir ortaktan geliyorsa, biletin kendisi tarafından verilip verilmediğini aynı şekilde kontol ederek yanıtını ortağına döner.

- Yanıtı alan GYGES2 onay yada red mesajını GES2 ye iletir. GES2, biletin güvenilir olduğu bilgisini GYGES2 den alırsa mesajı etmen2 nin giriş mesaj kuyruğuna koyar ve etmen2 uyarılır.

Gönderilen her mesaj için, güvenlik amacıyla bu kadar fazla işlemin yürütülmesi performans düşüşüne neden olacaktır. Etmen1'in etmen2'ye birinci mesajın hemen ardından ikinci bir mesaj göndermesi durumunda tüm bu adımların tekrar tekrar işletilmesi gerekmektedir. Bu nedenle mimari içinde performans düşüşlerini önleyecek mekanizmalar da geliştirilmiştir.

Herhangi bir GES sunucu, bir bileti güvenlik yöneticisinden ilk kez onayladığı anda bu bileti "güvenilir bilet" olarak saklar. Bu saklama süresi biletin yaşam süresi kadardır. Bu süre içinde, aynı bilet ile gelen yeni bir istek için tekrar güvenlik yöneticisine onay için başvurulmaz, çünkü bilet kısa bir süre önce zaten onaylanmıştır. En kötü durumda, bir GES sunucu bir biletin yaşam süresi dolmak üzereyken onu "güvenilir bilet" olarak işaretleyebilir ve yaşam süresi kadar da güvenilen bir bilet olarak saklarsa, biletin yaşam süresi olması gerekenden iki kat fazla olacak şekilde işlem görür. Bilet yaşam süresi GYGES yöneticisi tarafından ayarlanabilir bir parametredir, bu durum göz önüne alınarak parametre belirlenmelidir.

7.3 Gözleyici GES

Hareketli bir etmen sisteminde etmenler arasında, konumdan (yer) bağımsız bir mesajlaşma alt yapısının etkin olabilmesi için, her etmenin güncel yer bilgisinin tutuluyor olması gerekmektedir. Gözleyici düzende çalışan GES sunucunun görevi güncel "etmen-yer" bilgilerini tutmak ve istendiğinde bunu GES sunuculara iletmektir.

GES etmenleri aktif duruma geçtikleri zaman kendilerini bir "isim" ile ortama duyururlar. Etmenler birbirlerine mesaj göndermeden önce, etmen isimlerini sorgular ve geriye sorguladıkları etmene bir referans (AgentPointer) elde ederler. Bu referansı kullanarak birbirlerine mesaj gönderebilirler. Mesajlaşma alt yapısı Bölüm 8 de ayrıntıları ile anlatılacaktır.

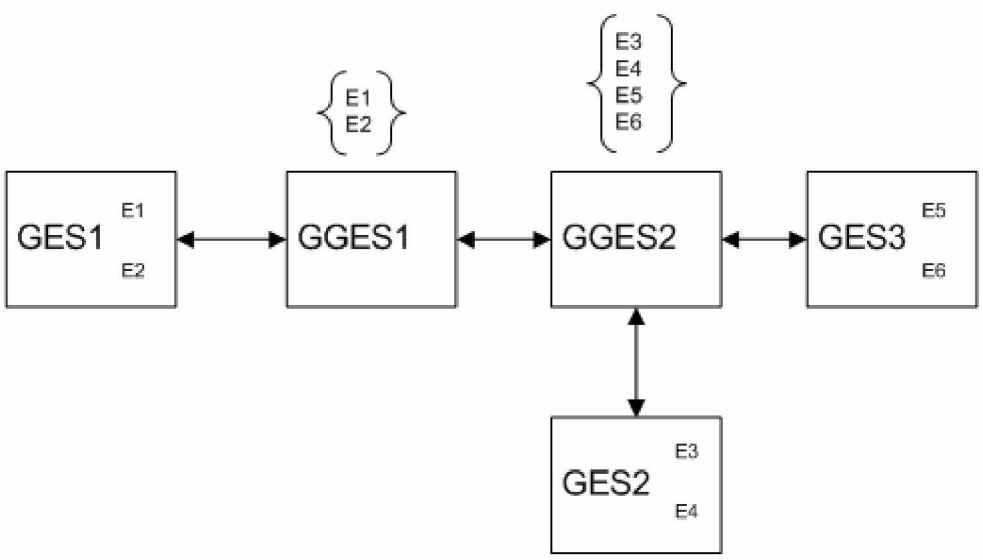
Herhangi bir GES sunucu, gözleyici olarak en az bir GES sunucuyu tanımlıyor olmalıdır. Birden fazla gözleyici GES tanımı olması durumunda, sunucu başarılı bir şekilde iletişim kuracağı gözleyici GES'i bulana kadar sırayla bağlantı kurmayı dener. Ancak, belirli bir anda, sadece bir gözleyici GES'i kullanıyor olabilir (Aynen bir GES sunucu için aynı andan aktif sadece bir GYGES'in olması gibi).

Herhangi bir GES sunucu, üzerindeki bir etmen aktif veya pasif duruma geçtiği anda bu bilgiyi "Gözleyici" düzende çalışan GES sunucuya (GGES) iletir. Bunun yanında, GES sunucu, ayarlanabilir süre aralıklarında, üzerindeki tüm etmenlerin bir listesini GGES'e iletir. Böylece GGES, etmenlerin üzerinde çalıştıkları sunucu bilgilerine güncel olarak her zaman sahip olur. Ancak üzerindeki etmen listesini GGES'e ilettikten sonra herhangi bir problemden dolayı çalışamaz duruma geçen bir GES sunucu olması halinde, GGES hala bazı etmenlerin bu GES sunucu üzerinde yer aldığı bilgisini tutmaya devam edecektir. Bu gibi durumları engellemek için GGES, belirli aralıklarla kendisine daha önce etmen listesi göndermiş GES sunucuların problemsiz çalışıp çalışmadıklarını kontrol eder. İletişime geçemediği bir GES sunucu olursa, bu GES sunucuya ait bütün etmenleri listesinden siler.

Güvenlik yöneticisi düzende çalışan GES sunucular gibi Gözleyici düzende çalışan GES sunucular arasında da "ortaklık" ilişkisi vardır. Farklı olarak GGES'ler GYGES'ler gibi ortaklık ilişkisi için paylaşılan bir anahtar kullanmazlar, çünkü GGES'ler zaten GYGES'lerden kimlik denetimi almış olan sunuculardır; birbirlerini ortak olarak tanımlamaları yeterlidir.

Herhangi bir GES sunucu, bir etmenin sistemdeki tüm etmenleri öğrenme isteğini karşılamaya çalıştığı durumlarda GGES'ten güncel etmen listesini istediği zaman

GGES, ortak olarak tanımladığı diğer GGES'lerden güncel listelerini elde eder ve kendi listesi ile birleştirerek istekte bulunan GES'e iletir.

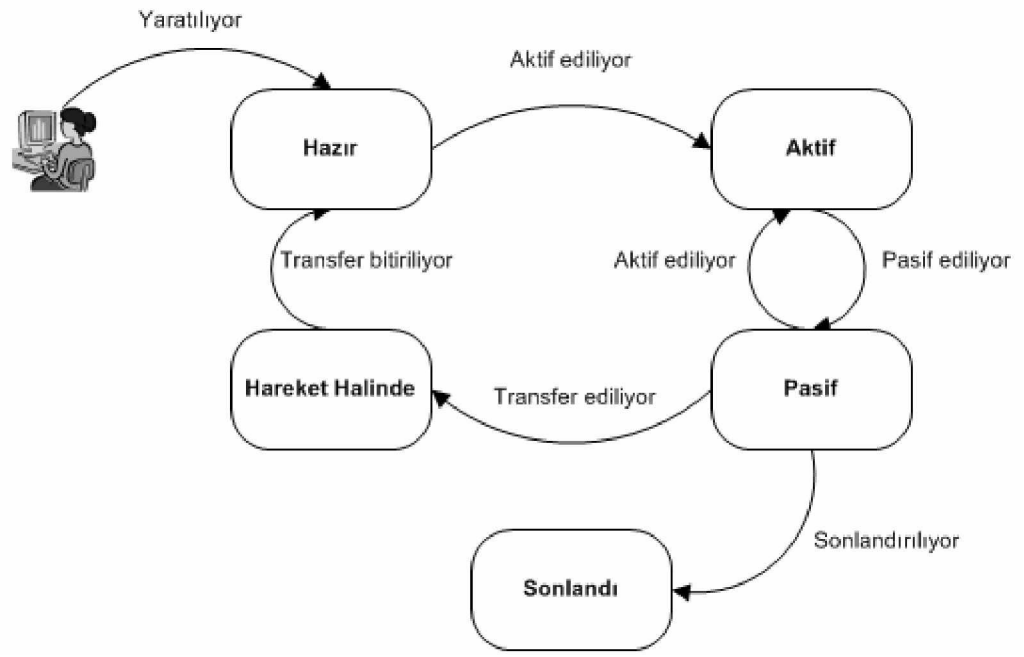


Şekil 7.3.1 : GGES ler arasındaki ortaklık ilişkisi

Örnek olarak Şekil 7.3.1 de görülen yapıyı ele alalım. GES1 sunucusu gözleyici GES sunucu olarak GGES1'i ve GES2 ve GES3 sunucuları ise Gözleyici GES olarak GGES2 yi tanımlamış olsunlar. Her bir sunucu üzerinde çalışan etmenler Ei olarak gösterilmiştir. Dolayısı ile, belirli bir anda GGES1, GES1 sunucusu üzerindeki etmenleri tanır ve GGES2 ise GES2 ve GES3 sunucuları üzerindeki etmenleri tanır. GES1 sunucusu güncel etmen listesini istediği zaman GGES1 önce ortağı olan GGES2 den güncel etmen listesini alır (E3,E4,E5,E6) ve kendi listesi (E1,E2) ile birleştirerek tüm etmenleri içeren listeyi GES1 sunucusuna iletir. GGES1 ve GGES2 arasındaki iletişim daha önce kendi GYGES sunucularından aldıkları biletler üzerinden olur. Bu nedenle, iki GGES'in de güvenlik yöneticileri aynı olmalı, veya farklı iseler, sözkonusu GYGES sunucular ortaklık ilişkisi içinde olmalıdırlar.

7.4 GES Etmenleri

GES mimarisi içinde etmenler JAVA dili ile geliştirilmelidirler. Her etmen yaşam döngüsü boyunca belirli bir anda yaratılma, aktif, pasif, hareket halinde ve sonlanma durumlarından birinde olabilir. Bu durumlar etmenin davranışını belirleyen durumlardır ve etmen programcısı, bu durumların her biri için, etmenin yürüteceği adımları etmen kodu içinde belirtmelidir. Etmen programcısı kendisine verilen etmen şablonu içinde bu durumlara karşılık gelen metodları etmenin işlevine uygun olarak gerçekler. Bit etmenin yaşam döngüsü şekil 7.4.1 de gösterildiği gibidir.



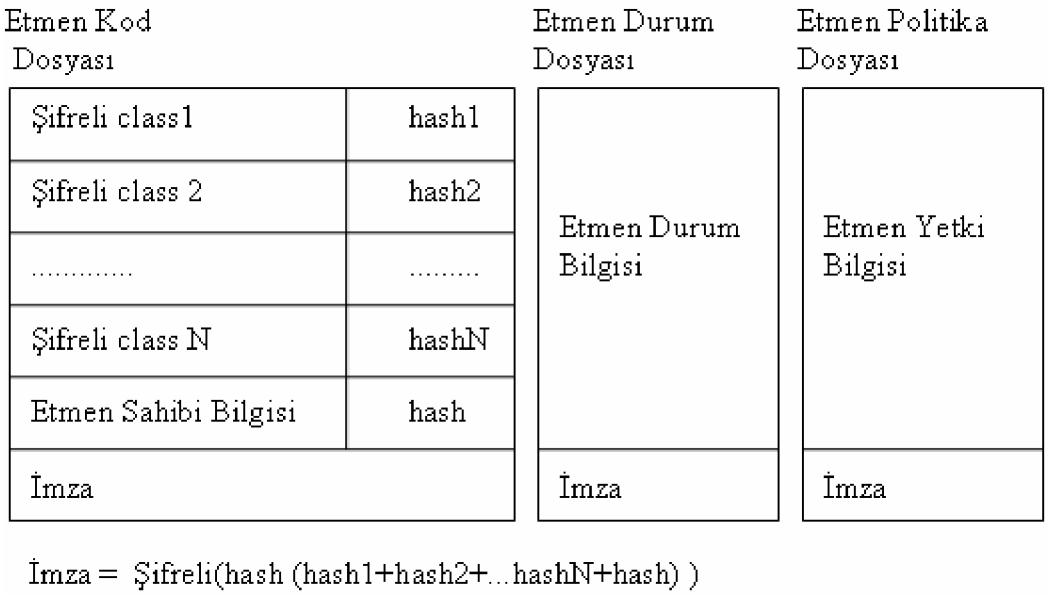
Şekil 7.4.1 : Bir etmenin yaşam döngüsü

Etmen durumları ve her durum için GES tarafından yerine getirilen işlemler şu şekildedir.

Yaratılma: Etmen yaşam döngüsü boyunca bu durumda sadece bir kez bulunur. Etmenlere yaratıldıkları anda bir kimlik bilgisi atanır. Bu kimlik bilgisi tekil olmayı sağlar ve GES sunucular arasındaki etmenlere ait bütün mesajlaşmalar bu kimlik bilgilerini içerir. Tekillik, çakışma ihtimali neredeyse bulunmayan gelişigüzel 128 sekizli uzunluğunda bir veri yaratılarak sağlanır. Etmen kimliği etmen ile birlikte taşınır. Etmen kimliği, etmeni belirleyen ve rastgele yaratılan bir katarı, etmenin sahibini, etmenin o an üzerinde çalıştığı GES sunucunun adresini ve etmen koduna ait imza bilgisini içerir. Bu bilgilerden sadece etmenin o an üzerinde çalıştığı GES sunucu adresini belirleyen alan değişkendir, diğer alanlar etmen yaşam süresince değişmez.

Çalışmaya henuz başlamamış olan etmen henüz durum bilgisine sahip değildir, ancak değişkenlerine etmen programcısının belirlediği ilk değerler bu aşamada atanır. Bir sonraki adım etmen kod ve politikasının şifrelenerek GES sunucu üzerinde saklanmasıdır. Etmen kodu bir veya daha fazla JAVA sınıf dosyasından oluşmaktadır. GES sunucusu etmene ait bütün sınıf dosyalarını bir sıkıştırılmış dosya içinde tutar. Etmene ait sınıf dosyalarının hash değerleri de hesaplanır ve bu bilgi şifrelenerek etmen kod imzası olarak dosyanın içine yazılır. Şifreleme için gerekli anahtar “Güvenlik Yöneticisi” düzende çalışan GES sunucudan elde edilir. GES sunucu bir etmeni aktif hale getirmeden önce etmene ait sınıf dosyalarının hash

değerlerini tekrar hesaplar ve daha önce hesaplanmış değer ile karşılaştırır. Eğer aynı sonuç çıkmaz ise, etmen koduna yetkisiz müdahele olduğuna karar verir ve etmeni aktif hale getirmez. Etmen kodu ve içindeki imza değeri, hem etmen durum bilgisini tutan dosyaya hem de etmen politikasını tutan dosyaya da yazılır; böylece etmene ait kod, durum ve politika dosyalarının tutarlılığı sağlanır (Şekil 7.4.2).



Şekil 7.4.2 : Etmen kodunun imzalanması ("+" katar birleştirme işaretidir)

Aktif Duruma Getirme: Daha önce yaratılmış olan bir etmenin çalıştırılabilir kod parçası belleğe alınır ve artık etmen bu aşamadan sonra çevresi ile etkileşime geçebilir. GES sunucusu etmenleri, bir ya da daha çok iplik (Thread) olarak çalıştırır. Etmen aktivasyon kodu yeni bir ipliktir ancak programcı yeni bir mesaj geldiğinde yürütülmesi gereken adımları "OnMessageArrive" metodu içinde tanımlamışsa, mesaj kabulü için yürütülmesi gereken adımlar ayrı bir iplik (Thread) içinde çalıştırılır.

Aktif olan etmen mesaj gönderip alabilir, göç isteğinde bulunabilir, bir kopyasını yaratabilir, kendini çevresine bir isim ile duyurabilir veya pasif duruma geçme isteğinde bulunabilir.

Pasif Duruma Getirme: Pasif hale geçirilen bir etmen artık etkin değildir ve son durum bilgisi GES sunucu diskinde saklanmıştır. Etmene ait çalıştırılabilir kod parçası bellekte yer almaz ve diğer etmenler pasif bir etmenin varlığından haberdar olmazlar. Etmen istediği anda bir süre için pasif duruma geçirilip, sonra tekrar aktif kılınabilir. GES sunucusu da belirli zamanlarda etmeni pasif duruma geçmeye zorlayabilir; örneğin, göç edecek bir etmen önce pasif duruma geçirilir. Pasif

durumdaki bir etmen hiç bir şekilde çevresi ile etkileşime geçemez, mesaj gonderip alamaz.

Hareket Halinde Olma: Aktif olan bir etmen istediği anda başka bir düğüm üzerine taşınmak için göç isteğinde bulunabilir. Bu durumda etmen önce pasif duruma geçirilir, daha sonra kodu, durumu ve politikası uzak düğüme iletilir ve kaynak düğümdeki etmen silinerek karşı düğümde tekrar aktif hale getirilir. Bu arada geçen zaman etmenin hareket halinde olma durumudur. Bu anda etmen mesaj gönderip alamaz.

Yok edilme: Etmenin herhangi bir anda kendini yok etme isteğinde bulunarak sistemden silinmesi durumudur. Etmen önce pasif duruma geçirilir ve kodu, durumu ve politikası disk üzerinden silinir.

7.4.1 Etmen Kimliği

Etmenler ilk yaratıldıklarında her birine tekil olan bir kimlik bilgisi atandığı belirtilmişti. Etmenlere sistemde bu kimlik bilgileri aracılığı ile erişilir. Etmen kimliğini belirleyen sınıf aşağıdaki şekilde tanımlanmıştır.

```
public final class AgentIdentity implements Serializable {
    private String strAgentHostName = null;
    private String strVisibleAgentName = null;
    private String strOwner = null;
    private String toStringValue = null;
    private String signature=null;
    private final static SecureRandom securerandom = new
                                          SecureRandom();
    private final static int nL = 128;
    private byte[] rgbID = new byte[nL];


    //constructor method..
    public  AgentIdentity(String owner,String signature) {
        this.toStringValue = this.generateRandomString(32);
        this.strOwner = owner;
        this.signature = signature;
        int n = 0;

        while (n == 0) { securerandom.nextBytes(rgbID);
            n = rgbID[nL - 1] | rgbID[nL - 2] | rgbID[nL - 3] |
            rgbID[nL - 4];
        }
     }
```

```
      ......
      ......
    public final int hashCode() {
      return (rgbID[0] & 0xFF) +((rgbID[1] & 0xFF) << 8) +
             ((rgbID[2] & 0xFF) << 16) +((rgbID[3] & 0xFF) << 24);
    }
      .....
      .....
}
```

Etmen kimlik sınıfından bilgiler (AgentIdentity) GES sunucular arasında RMI aracılığı ile parametre olarak ağ üzerinden taşınacağı, etmen kimliğinin disk üzerinde etmen kodu ve politikası ile birlikte saklanması gerektiği için "Serializable" (serileştirilebilir) olarak tanımlanmıştır. "strAgentHostName" etmenin o an üzerinde çalıştığı GES sunucu adresini, "strVisibleAgentName" etmenin kendini çevresine duyurduğu ismi, "strOwner" etmenin ilk yaratıldığı GES sunucu adresini, "signature" ise daha önce açıklandığı gibi etmen kodundan hesaplanan hash değerinin şifreli değerini tutmak için kullanılmıştır. "toStringValue, "AgentPointer" sınıfına dönüşüm için kullanılmaktadır. Bir etmen diğer bir etmenin kimliğini öğrenmek istediği zaman etmenin gerçek kimlik nesnesini değil, kimliğinin sadece okunabilir değişkenlerini içeren yeni bir "AgentPointer" nesnesini elde eder.

Etmen tekilliğinin sağlanması için JAVA nın "SecureRandom" sınıfı kullanılmaktadır. 128 byte uzunluğunda güvenli rastgele sayı üreteci kullanılarak yaratılan değer, her etmen için tekil bir kimlik bilgisi atama görevini yerine getirir. Bu sınıfın önemli metodlarından biri de, JAVA'nın yaratılan her yeni sınıf için otomatik olarak oluşturduğu ve "Object" sınıfından miras kalan "hashCode" metodudur. Nesnelerin eşitliği için JAVA nın kendi içinde kullandığı bu metod "AgentIdentity" sınıfı için yeniden yazılmıştır. Bu metodun yeniden yazılma sebeplerinden biri de yine JAVA'nın her nesne için var olan "Equals" isimli metodunun yeniden yazılmış olmasıdır. Bu metodun yeniden yazılmış olmasının nedeni iki etmen nesnesinin karşılaştırılırken eşit olarak algılanması için gerekli olan şartların belirtilmesi gerekliliğidir. Eğer "Equals" metodu yeniden yazılacaksa (override) "hashCode" metodunun da yeniden yazılma zorunluluğu vardır (Java dilinin gereksinimi). Etmen sınıfından türeyen iki nesnenin eşitlik karşılaştırmaları "Equals" ve "hashCode" metodlarının sonucuna bağlıdır. İki metod içinde de tekilliği sağlayan 128 bytelık bilginin kullanıldığına dikkat edilmelidir.

7.4.2 Etmen Arayüzü

GES sunucusu bir etmeni yarattığı zaman "Etmen Kabuğu" (AgentShield) isimli yeni bir nesne yaratır ve etmen nesnesi de "Etmen Kabuğu" nesnesinin özel (private) bir değişkeni olur. "Etmen Kabuğu" etmenin çevresi ile etkileşimi için etmene bir arayüz (AgentInterface) sunar. Etmen bu arayüz aracılığı ile GES sunucuya, mesajlaşma, göç etme, kopya yaratma (clone) ya da pasif hale gelme gibi isteklerini iletir. "Etmen Arayüz" (AgentInterface) sınıfı aşağıdaki şekilde tanımlanmıştır.

```
abstract public class AgentInterface {

 abstract public String  getAgentHostName();
 abstract public MessageID
 sendMessage(AgentPointer agentpointer,Message message);
 abstract public void sendReply(MessagePacket packet,Object reply);
 abstract public boolean isFailed(MessageID id);
 abstract public boolean isSent(MessageID id);
 abstract public boolean isReplyReady(MessageID id);
 abstract public boolean waitForReply(MessageID id,long ms);
 abstract public Object getReply(MessageID id);
 abstract public MessagePacket receive();
 abstract public boolean waitForMessage(long ms);
 abstract public boolean isAgentAlive();
 abstract public AgentPointer createClone();
 abstract public TransferResponse  move(String strAgentHostName);
 abstract public void setVisibleOn(String strIdentifier);
 abstract public void setVisibleOff();
 abstract public Enumeration getAgentPointer(String strIdentifier);
 abstract public Enumeration getAgentPointer();
 abstract public void setRepliedMessageHandler(String method);
 abstract public void suspend(long suspendTime);
}
```

"AgentInterface" sınıfı içindeki her metod etmenin yürütebileceği işlemlere karşı düşer. Bu metodlar ve nasıl kullanılacakları "Etmen Yazılım Geliştirme Arayüzü" konu başlığı altında ayrıntılı incelenmiştir. Metodların "abstract" olarak tanımlandığına dikkat edilmelidir. Çünkü bu metodlar henüz gerçeklemeleri yapılmamış olan metodlardır. Etmen, bu metodlardan birini çağırır, gerçekleme işini bazen sadece Etmen Kabuğu, bazen de hem Etmen Kabuğu hem GES motoru yerine

getirir. Her etmen sınıfı ayrıca “AgentRoot” isimli bir sınıftan türer. AgentRoot sınıfının görevi etmene verilecek olan AgentInterface nesnesini tanımlamaktan ibarettir.

Aşağıdaki şekilde tanımlanmıştır.

```
public class AgentRoot {

    private transient AgentInterface agentinterface = null;

    final protected void
            setAgentInterface(AgentInterface agentinterface) {
        this.agentinterface = agentinterface;
    }

    protected final AgentInterface
            getAgentInterface() {
        return agentinterface;
    }
}
```

AgentInterface nesnesinin “transient” direktifi ile serileştirilemez olduğu belirtilmiştir. Bu işlemin nedeni, arayüz etmenin gittiği her yeni düğümde yeniden yaratılan “Etmen Kabuğu” tarafından etmene yeniden verilecek olmasıdır. Yani, etmen arayüzü taşınabilir nitelikte değildir, olması da gerekmemektedir.

Artık etmen sınıfının tanımlanması için gerekli olan bütün sınıflar hazırdır. Daha önce de belirtildiği gibi, etmen programcısı etmeni, etmenin yaşam döngüsü içinde bulunabileceği durumlara karşılık gelen metodları yazarak programlar. Dolayısı ile, etmen sınıfının tanımında bu durumların her birini görmek mümkündür. Ayrıca, etmene bir mesaj geldiğinde yürütülmesi istenen adımları belirleyen bir metod da yer alır. Tabii ki etmen kodu içinde sadece bu metodların bulunması gibi bir sınırlama yoktur. Ancak etmen davranışlarını bu özel metodlar belirler. Etmen sınıfı da aşağıdaki gibi tanımlanmıştır.

```
abstract public class Agent extends AgentRoot
                          implements Serializable{


private Object objTarget = null;

public final Object getTarget() {
    return objTarget;
  }

protected  Agent() {
    objTarget = this;
  }

Protected Agent(Object objTarget) {
    this.objTarget = objTarget;
```

```
  }

//Etmene mesaj geldiğinde yapılacaklar..
abstract public void OnMessageArrive();

//Etmen yaratıldığında yapılacaklar..
abstract public void OnCreate();

//Etme aktif durumdayken yapılacaklar..
abstract public void OnActivate();

//Etmen pasif duruma geçerken yapılacaklar..
abstract public void OnInactivate();

//Etmen yok edilirken yapılacaklar..
abstract public void OnEnd();

//Etmen göç etmeden hemen önce yapılacaklar..
abstract public void OnTransfer();

}
```

Etmen sınıfı “Serializable” olarak tanımlanmıştır. Durumlara karşılık gelen her metodun “abstract” olarak tanımlandığı görülmektedir. Etmen programcısı bu soyut metod tanımlamalarının içini dolduracaktır. Etmen programcısına temel olarak “Agent” sınıfından türeyen yeni bir sınıfı programlaması için hazır bir şablon verilir. Böyle bir şablon programcıya etmen davranışlarını kolayca programlayabilmesi için esnek bir programlama modeli sunar.

8. GES HABERLEŞME ALTYAPISI

GES mimarisi etmen haberleşmesi için oldukça esnek bir yapıya sahiptir. Etmeni bloke etmeyen ve güvenli iletişimi sağlayan mekanizmalar mevcuttur. Bir mesajlaşma sistemin güvenli sayılabilmesi için şu özelliklere sahip olması gerekmektedir [2].

Gizlilik: Haberleşme sırasında gönderilen verilerin, haberleşen birimler dışındaki partiler tarafından ele geçirilememesi.

Veri Bütünlüğü: İletilen verilerin değişmeden alıcısına ulaştığından emin olunması.

Kaynak Kimlik Denetimi: Mesajı gönderen kaynağın iddia ettiği kişi olduğunun belirlenebilmesi.

Yalanlayamama: Mesajı gönderen birimin, göndermediğini iddia ettiği durumlarda, gönderdiğinin ispatlanması ya da alan birimin, almadığını iddia ettiği durumlarda, aldığının ispatlanması.

Güvenli Etmen Mimarisi, ilk üç gereksinimi iletişim için SSL protokolü ve şifreleme teknikleri kullanarak sağlar. Mimari, sistemdeki bütün iletişim aktivitelerininin izlerini tutabildiği için, gönderici yada alıcının aktiviteyi yalanlaması da mümkün değildir.

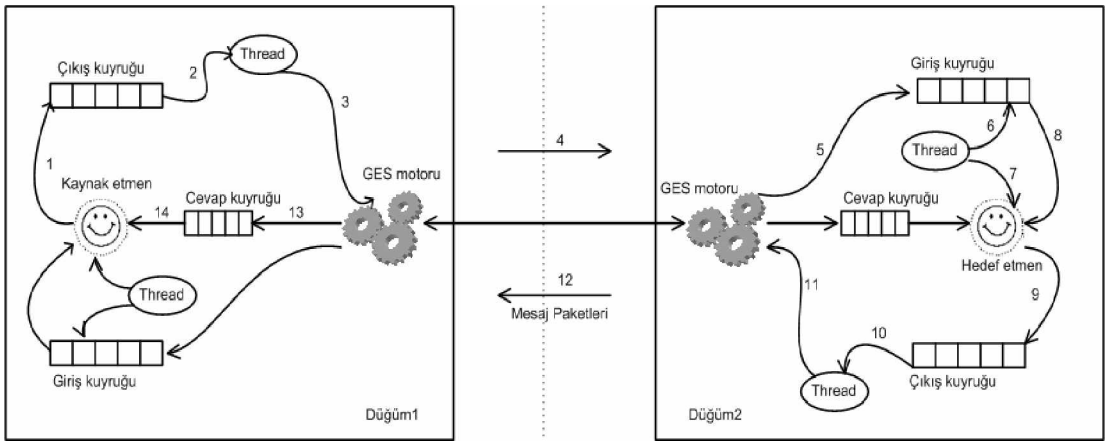
GES mesajlaşma mimarisi aşağıdaki özelliklere sahiptir.

- Etmenin çalıştığı yerden bağımsız bir mesajlaşma modeli sunar.
- İletişim için mesaj paketleri kullanır.
- Merkezi değil, dağıtık mesajlaşma yapısına sahiptir.
- İletilen mesajlar şifreli olarak gönderilir. Açık herhangi bir veri iletilmez.
- Mesajlaşma alt yapısı etmeni bloke etmeyen bir yapıya sahiptir; asenkron iletişim sözkonusudur.

- Her etmen için mesaj gönderim kuyruğu, mesaj alım kuyruğu ve yanıt alım kuyruğu oluşturulur. Bu kuyruklar Etmen Kabuğu (AgentShield) tarafından oluşturulurlar.

- Mesaj giriş ve mesaj çıkış kuyruklarını dinleyen ve mesaj varlığından etmeni haberdar eden ayrı iplikler oluşturulur. Bu nedenle, etmenin yeni bir mesajın varlığını belirlemek için kontrol kodu yürütmesine gerek yoktur; asli görevlerine yoğunlaşabilir. Ancak etmen isterse, mesaj alım kuyruğunu kendisi de dinleyebilir.

- Etmene mesajların akibeti konusunda bilgi edinebileceği ve yanıtları edinebileceği esnek fonksiyonlar sunulmuştur. Bu metodlar "Etmen Yazılım Programlama Arayüzü" başlıklı bölümde ayrıntılı incelenmiştir.



Şekil 8.1 : GES mesajlaşma mimarisi

Şekilde 8.1 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(thread) canlandırılır. Giriş kuyruğunu izleyen iplik, yeni bir mesaj ulaştığında etmeni, etmene ait "OnMessageArrive" metodunu ayrı bir iplik içinde çalıştırarak 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, önce mesaj göndereceği etmenin kimlik bilgisine bir işaretçi elde eder. Daha sonra bu işaretçi ve mesajı parametre olarak kullanarak mesaj gönderme isteğinde bulunur. Çağrı

sonunda Etmen Kabuğu" etmenin mesajını mesaj kuyruğuna yerleştirir. İletişim asenkron olarak gerçekleştiği için, bu noktadan sonra, etmen iletimin gerçeklenme ayrıntılarıyla ilgilenmez. Eğer bir yanıt bekliyorsa, mesaj yanıtı gelene kadar diğer işleriyle ilgilenebilir. Dilerse, uygun gördüğü bir zamanda, 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.

Mesaj nesnesi, mesaj adı ve parametre listesinden oluşmaktadır ve aşağıdaki şekilde tanımlanmıştır.

```
public final class Message implements Serializable {

private String strMessageName = null;
public final String  getMessageName() {
    return strMessageName;
  }

private Object[] rgobjParameters = null;

public final Object[] getParameters() {
return rgobjParameters;
  }

Public Message(String strMessageName) {
    Object[] rgobjParameters = new Object[0];
    constructMessage(strMessageName, rgobjParameters);
  }

Public Message(String strMessageName, Object obj) {
    Object[] rgobjParameters = new Object[1];
    rgobjParameters[0] = obj;
    constructMessage(strMessageName, rgobjParameters);
  }

Public Message(String strMessageName, Object obj0, Object obj1) {
    Object[] rgobjParameters = new Object[2];
    rgobjParameters[0] = obj0;
    rgobjParameters[1] = obj1;
    constructMessage(strMessageName, rgobjParameters);
  }

Public Message(String strMessageName, Object[] rgobjParameters) {
    constructMessage(strMessageName, rgobjParameters);
  }
```

```
 public final int hashCode() {
    return strMessageName.hashCode();
  }

........

}
```

Mesajlar ağ üzerinden RMI aracılığı ile iletildiği için "Serializable" olarak tanımlanmışlardır. Parametre listesi Object sınıfından türeyen herhangi bir nesne olabilir (JAVA' da ilkel tipler hariç bütün veri tipleri Object sınıfından türemiştir).

Etmen herhangi bir mesaj aldığında mesajın adını, parametre sayısını, tiplerini ve değerlerini sorgulayıp ona göre farklı davranış şekilleri gösterebilir.

Mesaj nesneleri etmenler arasında mesaj paketleri olarak iletilir. Bir mesaj paketinin yapısı şekil 8.2 de verilmiştir.

ID	Kaynak Düğüm	Kaynak Etmen	Hedef Düğüm	Hedef Etmen	ACK

Şekil 8.2 : Mesaj paketi

Bir mesaj paket nesnesi ise şu şekilde tanımlanmıştır.

```
public final class MessagePacket implements Serializable {

private MessageID ID=null;
private String SourceAgentHost = null;
private String DestinationAgentHost=null;
private AgentPointer SourceAgentPointer=null;
private AgentPointer DestinationAgentPointer=null;
private Object Object;
private boolean Ack=false; //0 = message 1=Reply

.........

}
```

MessageID: Paket numarası, rastgele üretilen büyük bir sayı. Etmen gönderdiği mesaj paketinin ID değerini öğrenip mesajın akıbetini bu değer ile sorgulayabilir.

Kaynak Düğüm: Mesaj gönderen etmeni çalıştıran GES sunucu adresi (IP adresi ve isimden oluşur)

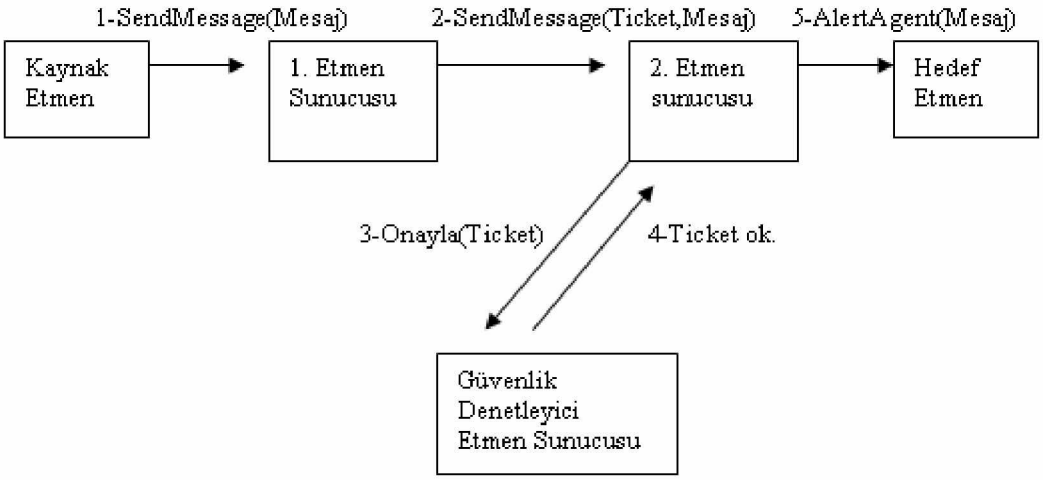
Kaynak Etmen: Mesajı gönderen etmenin kimliğini (AgentIdentity).

Hedef Düğüm: Mesaj alacak olan etmeni çalıştıran GES sunucu adresi.

Hedef Etmen : Mesajı alacak etmenin kimliğini.

ACK: Paketin mesaj veya yanıt paketi olduğunu gösteren bilgi.

Mesaj iletimi sırasında, etmen sunucuları arasındaki haberleşmelerin şifreli olmasının yanısıra, etmen sunucuları birbirlerini aynı zamanda kimlik denetiminden de geçirirler. Bunun için daha önce bahsedilen "Bilet" mekanizması kullanılır. Bu işleyiş şekli Şekil 8.3 te gösterilmiştir. Şekildeki kaynak etmen 1. Etmen sunucusu üzerinde bir makine üzerinde, hedef etmen ise 2. Etmen sunucusu üzerinde farklı bir makine üzerinde çalışmaktadır.



Şekil 8.3 : Mesaj iletimi sırasında bilet kullanımı

Kaynak etmen hedef etmene mesajı sendMessage çağrısı ile yollar. Mesaj gönderme fonksiyonunun parametreleri hedef etmene bir işaretçi ve mesajın kendisidir. Mesaj kaynak etmen sunucusuna geldiğinde hedef etmenin yeri, mesaj paketi içinde yer alan bilgilerden saptanır, sonra hedef etmenin üzerinde çalıştığı etmen sunucusuna mesaj gönderim isteği bildirilir. Ancak, bu adımda, gönderim isteği içinde yerel etmen sunucusunun kimlik denetimi sonucu edindiği bilet de yer alır. Hedef etmen sunucusu gelen istek içindeki bileti "Güvenlik Yöneticisi" düzende çalışan etmen sunucusuna onaylatır; eğer bilet geçerli bir bilet ise hedef etmen sunucu mesajı alıcı etmenin giriş kuyruğuna bırakır ve etmeni uyarır. Bilet onaylanamazsa istek gözardı edilir ve kaynak sunucuya hata dönülür. Kaynak sunucu ilgili mesajın gönderilemediği bilgisini saklar ve kaynak etmen mesajın akibetini sorguladığında ona bu bilgiyi döner.

Etmen sunucusu mesaj paketini alıcı etmen sunucusuna ilettiği zaman, daha önce hangi numaralı pakatleri ilettiği bilgisini de kaydeder. Herhangi bir yanıt mesaj paketi geldiğinde, bu yanıtın daha önce gönderilmiş bir mesaja ait olup olmadığı

kontrolünü de yapar. Böylece sistemde sahte yanıt paketleri oluşturarak etmenlerin yanlış davranişlara yönlendirebilecek durumlardan da sakınılmış olunur.

Gözleyici düzende çalışan GES hakkında bilgi verilirken, yerden bağımsız mesajlaşma olanağı sunmak için güncel etmen-yer bilgisinin sağlanması için bütün GES sunucular üzerlerinde aktif yada pasif olan etmen kimlik bilgilerini GGES'e iletildiği açıklanmıştı. GGES, en güncel etmen-yer bilgisini barındırır ve istendiğinde diğer etmen sunucularına bu bilgiyi iletir. GGES ler mesajlaşma yapısının temelini oluştururlar. Sürekliliği sağlamak için bir den çok GGES sunucu ortaklık ilişkisi içince çalışabilir. Ortaklıklar düğüm yöneticisi tarafından belirlenir. Ayrıca bir GES sunucu için birden çok GGES tanımı yapılabilir ancak aynı anda sadece bir GGES kullanılabilir. GES sunucu aktif GGES'e erişemez ise hemen tanımlı olan diğer bir GGES'e erişmeyi dener. Bu, çalışan bir GGES bulununcaya kadar devam eder.

9. GES GÜVENLİK POLİTİKALARI VE YÖNETİMİ

Hareketli etmenlerin uygulama alanlarının genişlemesi, beklenilen güvenlik ihtiyaçlarına cevap verecek yapıların geliştirilmesi ile mümkündür. Diğer önemli bir konu ise bu güvenlik ihtiyaçlarının değişken olmasıdır. Bugün için kullanılmasında sakınca olmayan bir kaynağın kullanımı, değişen çevresel faktörler nedeniyle ileride sakıncalı olabilir. O an geldiğinde bu değişikliğe uyum sağlamak mümkün olduğu kadar esnek ve kolay olmalıdır.

Politika tanımının temelinde yetkilendirme söz konusudur. Hareketli etmen sistemlerini düşündüğümüzde yetki verilecek birimler aktiviteler olarak değerlendirilebilir. Örneğin bir etmenin göç etmesi, mesajlaşması, düğüm diskine yazması birer aktivitedir. Yetkilendirme söz konusu olduğunda yetki verilecek birimleri de tanımlamak gerekmektedir. Hareketli etmen sistemlerinde yetki verilecek birimler, etmenler ve etmen sunucuları olarak tanımlanabilir.

Buna göre politika, etmen yada etmen sunucularına verilecek olan yetkilendirme kurallarının bütünüdür. Temel amacı, değişen güvenlik ihtiyaçlarına kolay cevap verebilmek, yönetimi kolaylaştırmak ve çevresel değişikliklere uyum sağlamayı hızlandırmaktır.

Hareketli bir etmen sistemi, sağladığı esnek fonksiyonların yanında, çevresel değişikliklere de uyum sağlamayı kolaylaştıracak mekanizmalara sahip olmalıdır. Bunu yaparken, mümkün olduğu kadar etmenin yeniden programlanması gerekmemeli, ve çalışan sistemde dinamik değişikliklere izin vermelidir.

Sisteme girmiş ve güvenilir olduğu kabul edilmiş herhangi bir etmenin taşındığı ortam üzerinde zarar verici etkilerinin olması hala muhtemeldir. Etmen, diskten okuma, yazma ya da silme işlemleri ile düğüm kaynaklarına zarar verebilir, ağ bağlantıları oluşturularak güvenlik için riskli olabilecek arka kapılar oluşturabilir veya düğüm üzerinde erişilmesi istenmeyen sistem değişkenlerine erişebilir.

Etmen sistemlerinde politika kullanımı ile ilgili olarak bazı çalışmaların yapıldığı görülmektedir. Örneğin Gallery[29] etmenlere yetkilerin verildiği ve hangi etmenin hangi platformda çalışabileceğinin tanımlandığı bir çatı (framework) oluşturmuştur. [30] referanslı çalışmada yazarlar sadece etmen hareketliliği için politika tabanlı bir

sistem oluşturmaya çalışmışlardır. [31] referanslı çalışmada ise yazarlar, düğümlerin etmenlere, geçmişteki davranışlarına bakarak yetki verebildiği bir yetkilendirme sistemi üzerinde çalışmışlardır. "Proof Carriying Code" [34] yönteminde ise etmen programcısının etmenin belirli bir güvenlik politikasına uyduğunu göstermesi istenir.

GES mimarisi, etmen ve düğümlere politikalar atanarak, etmen aktivitelerinin sınırlandırılabilmesine olanak sağlar. Düğümler de, düğüm politikaları oluşturularak korunur. Sistem bu desteği dinamik olarak vermektedir, dolayısı ile sistem aktif iken politikaları değiştirmek mümkündür. Bu çalışma modeli ile değişebilen çevresel şartlara göre etmenleri yeniden programlama gereksinimi de ortadan kalkmış olur.

9.1 Java Dilinin Güvenlik Yönetimi

JAVA dili, her uygulama için belirli kontrollerin yapılarak bazı JAVA çağrılarının kısıtlandırılabileceği bir mekanizma sunmaktadır. Her uygulama için "Java.lang.SecurityManager" sınıfından türeyen yeni bir sınıf yazılmasına olanak verir. Tehlikeli olabilecek herhangi bir aktivite gerçekleşmeden önce oluşturulan "Security Manager" 'a yönlendirilir ve eğer buradan onay alınırsa, işlem yerine getirilir. JAVA nın varsayılan Security Manager sınıfı hangi sınıfların hangi tür yetkilere izinli olduğunu bir dosyadan okuyarak anlar. Kontrol edilecek her işlem için yazılan bu sınıf içine "Check" ismi ile başlayan bir metod yazılmalıdır. Örneğin bu sınıf içinde tanımlanan "CheckRead" metodu çağrıyı yapan sınıfın dosya okuma izni olup olmadığı, ya da "CheckWrite" metodu dosyaya yazma izni olup olmadığını kontrol etmek için kullanılır. JAVA, aşağıdaki çağrılar için Security Manager sınıfı içinde "Check" ile başlayan metod yazımına izin verir. Bu, aynı zamanda kontrol edilebilen çağrı listesini de göstermektedir.

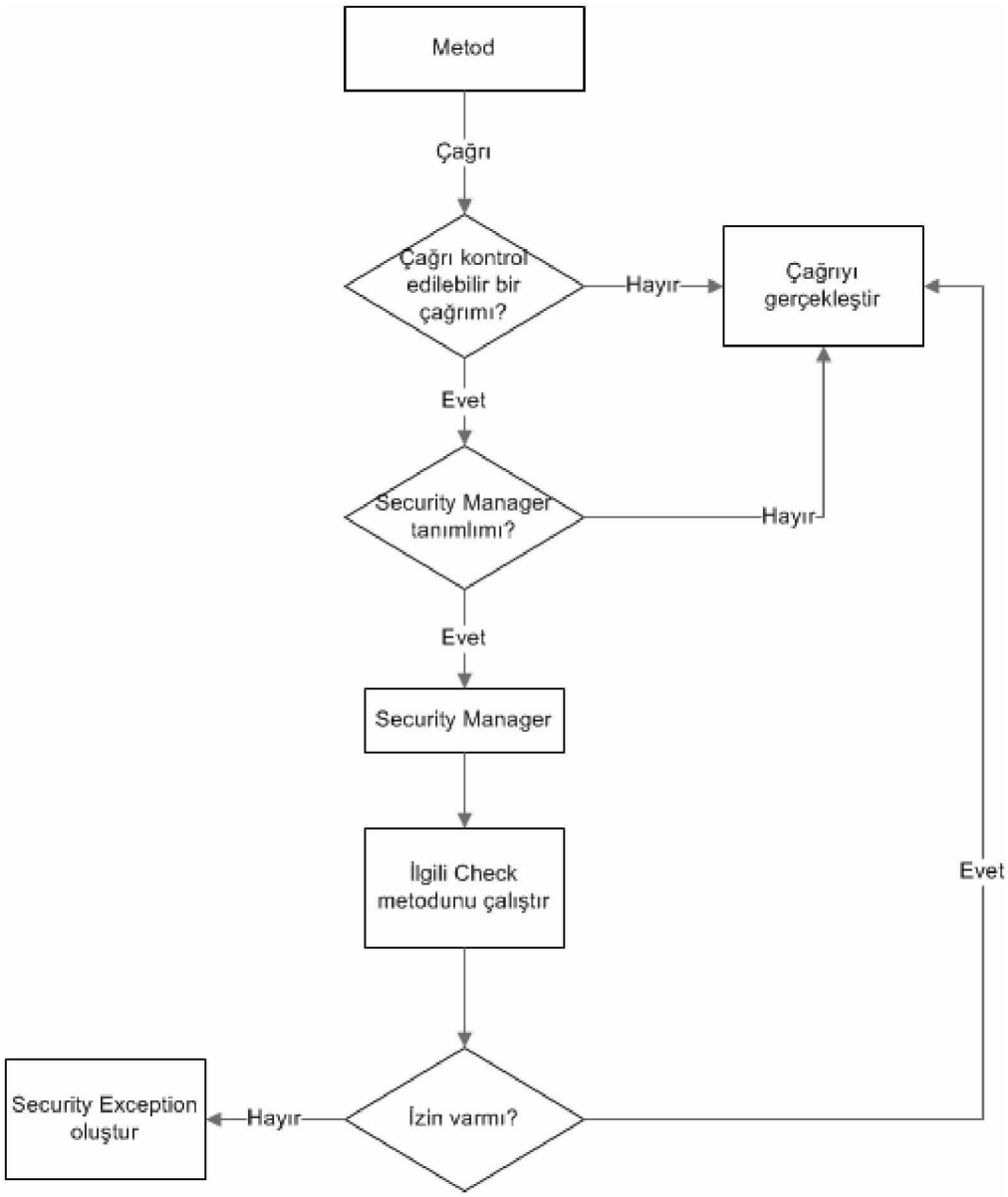
- Belirli bir düğümden, bir TCP portu üzerinden soket bağlantısı dinleme yada kabul etme
- Bir ipliğe müdahale (önceliğini değiştirme gibi)
- Bir düğüme, bir TCP portu üzerinden soket bağlantı isteği gönderme
- Yeni bir class yükleyicisi oluşturma
- Bir dosyayı silme
- Bir dosyadan okuma

- Bir dosyaya yazma
- Sistem değişkenlerini okuma ya da silme
- Uygulamayı sona erdirme

JAVA dili tehlikeli olabilecek iki tür kontrolün yapılmasına olanak sağlamamaktadır.

- Belleği aşırı kullanarak düğüme zarar verme
- Çok sayıda iplik oluşturarak düğüme zarar verme

Her uygulama için sadece bir "JAVA Security Manager" tanımlanabilir. Dolayısı ile yazılacak Security Manager sınıfı esnek politika yönetimini destekleyecek şekilde tasarlanmalıdır. Şekil 9.1.1 JAVA nın "Security Manager" çalışma modelini göstermektedir.



Şekil 9.1.1: JAVA Security Manager çalışma modeli

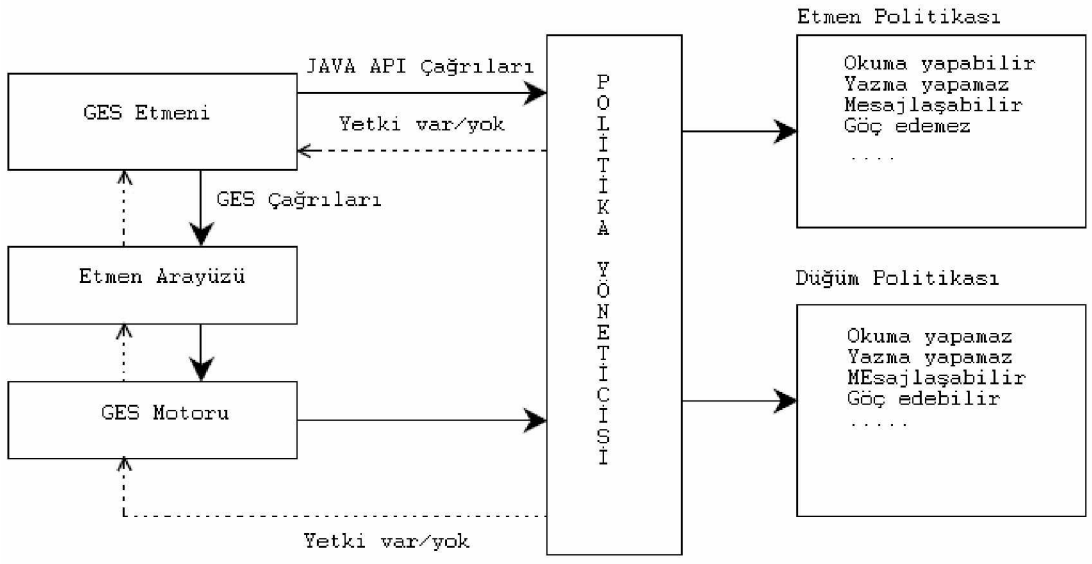
9.2 GES Güvenlik Denetleyicisi

Güvenli Etmen Sisteminde, her düğüm üzerinde kurulu olan etmen sunucuları, Javanın varsayılan "Security Manager"ından türeyen gelişmiş bir güvenlik denetleyicisine sahiptirler. Güvenlik politikaları dinamik olarak çalışma anında gözlenip, değiştirilebilmektedir.

GES, güvenlik politikalarını etmen ve düğüm politikaları olarak ikiye ayırır. Etmen politikaları etmen ile birlikte ağ üzerinde taşınırken, düğüm politikaları sabittir. Her iki politika da çalışma anında değiştirilip etkin duruma getirilebilir.

Bir GES etmeni şekil 9.2.1 de görüldüğü gibi genel olarak iki farklı türden çağrılar yapabilir. GES çağrıları ve Java API çağrıları. GES çağrıları ile etmen, haberleşme, göç etme, kopya yaratma ve diğer etmenlerin kimliklerini öğrenme gibi isteklerini etmen arayüzü aracılığı ile GES sunucusuna iletir. Her iki türden çağrı da gerçeklenmeden önce güvenlik denetleyicisi tarafından kontrol edilir ve etmen ve düğüm politika kurallarına göre işleme izin verilir ya da verilmez. Bütün JAVA API çağrıları kontrol edilemez. Buna rağmen aşağıda sıralanan ve güvenlik riski oluşturacak bütün çağrılar bu denetimden geçerler.

- Dosya sistemi fonksiyonları (yazma, okuma, silme)
- Ağ fonksiyonları (dinleme ve bağlantı amaçlı soket yaratma)
- Class yükleme fonksiyonları
- Sistem kaynaklarına erişim fonksiyonları (yazma kuyruğu, clipboard, olay kuyruğu, sistem özellikleri, vb)
- Uygulamayı sonlandırma



Şekil 9.2.1: GES etmeninin farklı türden çağrıları

GES sunucuları, aşağıdaki türden GES çağrılarını güvenlik politikaları ile denetleyebilirler.

- Mesaj gönderme
- Mesaj alma

- Göç etme

- Kopya yaratma

Politikaların kontrol edilerek yukarıda söz edilen çağrılara izin verilip verilmeyeceğini GES sunucuların "Güvenlik Denetleyici" isimli modülleri gerçekler. Bu modül, Javanın varsayılan "Security Manager" sınıfından türeyen oldukça yetenekli yeni bir güvenlik denetleyicidir

"Güvenlik Denetleyicisi", bir işlemi kontol ederken aşağıdaki algoritmayı işletir.

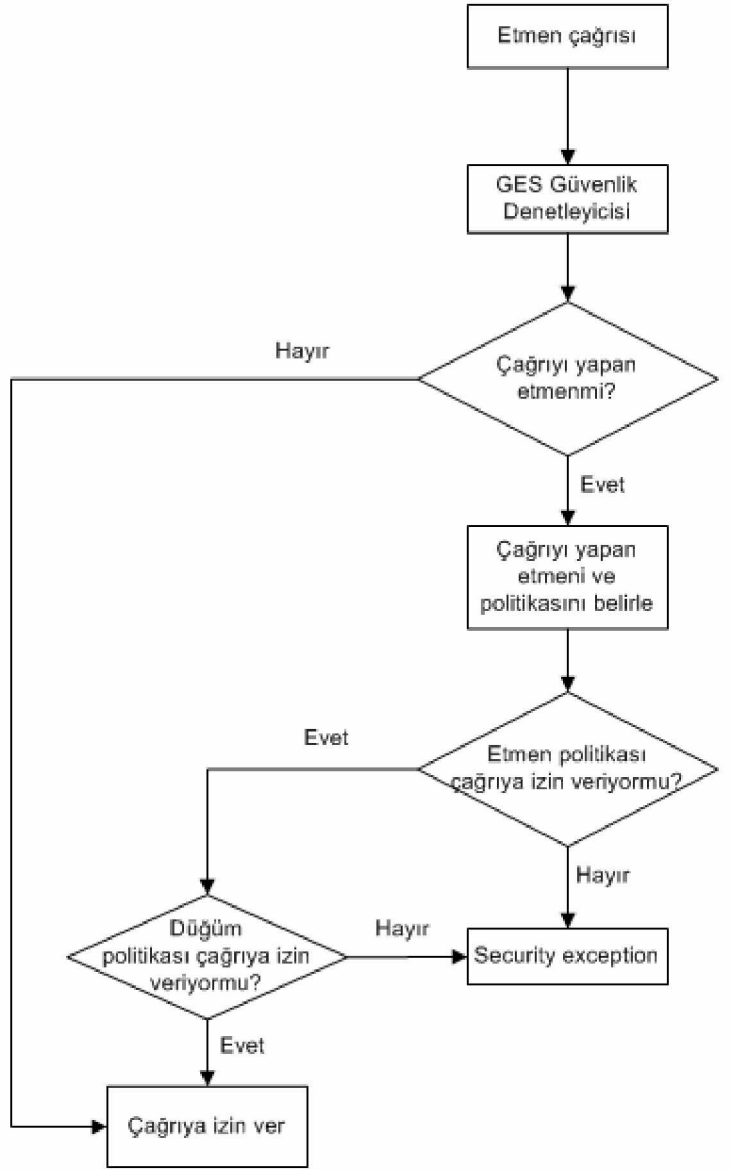
- Bir etmen ya da GES sunucusu tehlikeli olabilecek bir Java çağrısı yapar ya da bir etmen, göç etme, mesaj gönderme-alma gibi GES çağrısı yapar.

- Çağrı kontrol edilmek üzere güvenlik denetleyicisi tarafından yakalanır.

- Güvenlik denetleyicisi çağrıyı yapanı belirler ve eğer çağrıyı yapan GES sunucunun kendisi ise işleme izin verilir.

- Çağrı bir etmen tarafından yapılmış ise, etmen kimliği belirlenir. Etmen kimliğinden etmen politikası bulunur ve isteğe etmen politikasında izin verilip verilmediği kontrol edilir. Eğer etmen politikası çağrıyı onaylıyor ise, güvenlik denetleyicisi düğüm politikalarını okuyarak etmen çağrısının onaylanıp onaylanmadığına bakar, çağrı onaylanıyor ise işlem devam eder, onaylanmıyor ise çağrıyı yapan etmene "security exception" dönülerek çağrı bloke edilir.

GES güvenlik denetleyicisinin çalışma akışı şekil 9.2.2 de gösterilmiştir.



Şekil 9.2.2: GES Güvenlik Denetleyici akış diyagramı

9.3 Etmen Politikaları

Etmen politikaları, etmen programcısının ya da düğüm yöneticisinin etmenlerin aktivitelerini sınırlandırması için kullanılır. Örneğin programcı etmenin mesaj almasının bir güvenlik problemi yaratacağını düşünüyorsa, etmen politikasında mesaj almayı engelleyebilir ya da düğüm yöneticisi etmenin disk üzerinden okuma yapmasının ya da diske veri yazmasının bir güvenlik problemi olacağını düşünüyorsa etmene atadığı politika ile bunu engelleyebilir. Bu özellik etmen programcısının bu türden güvenlik gereksinimleri için program yazmasını engellediği gibi, düğüm yöneticisininde, düğüm kaynaklarının güvende olduğundan emin olmasını sağlar.

Etmen programcısı etmeni sisteme eklerken, aynı zamanda etmene, mesaj alıp-gönderme, göç etme, kendini kopyalama ve diske yazma-okuma gibi izinlerini belirleyen bir politika tanımlar ve atar. Etmen politikaları şifreli XML dosyaları olarak saklanır ve etmen ağ üzerinde hareket ettikçe onunla birlikte taşınır. Güvenli etmen sunucuları etmen kodunu, verisini ve politika dosyasını düğüm üzerinde güvenli bir alanda şifreli olarak tutarlar. Güvenli etmen sunucusu bir etmen aktif duruma geçeceği zaman etmene ait sınıf dosyalarını ve en son güncel durumunu içeren verisini yükledikten sonra etmen kabuğunu (AgentShield) yaratır. Etmen kabuğu etmene ait politika dosyasını okur ve içeriğini etmen kimliği ile ilişkilendirir. Etmen sahibi etmene ait politikayı çalışma anında görüntüleyebilir ve gerek duyarsa yürütme anında değiştirebilir. Bir değişiklik yapılması durumunda GES sunucuları etmen politikasını hem bellekte, hem de şifreli tutulan etmen politika dosyasında güncellerler.

Etmen politikasını tanımlayan sınıf aşağıdaki verilmiştir:

```
public class Policy implements Serializable {

    public String name;
    public String description;
    public boolean allowSendMessage;
    public boolean allowReceiveMessage;
    public boolean allowMove;
    public boolean allowClone;
    public boolean allowWrite;
    public boolean allowRead;
    public boolean allowOpenConnection;
    public boolean allowReceiveConnection;
    public boolean allowAccessSystemSettings;
    public boolean allowLoadClass;
    public String signature;

....
}
```

Etmen politikaları taşınabilir olduğu için “Serializable” olarak tanımlanmışlardır. Alan açıklamaları şu şekildedir.

name (String): Politikaya verilecek isimdir. GES sunucu bazında tekildir. Başka GES sunucular üzerinde aynı isimli politika olabilir, bu isim sadece GES sunucu yönetimini kolaylaştırmak için kullanılır. Sistem yöneticisi daha önce yaratılan etmen politikalarını görüntüleyebilir ve bunlardan her hangi birini herhangi bir etmene atayablir.

description (String): Politika içinde tanımlı olan hakları açıklayan ve yönetimi kolaylaştırma amacıyla tutulan açıklama alanıdır.

allowSendMessage (boolean): Etmenin mesaj gönderme iznine sahip olup olmadığını gösteren mantıksal alan.

allowReceiveMessage (boolean): Etmenin mesaj alma iznine sahip olup olmadığını gösteren mantıksal alan.

AllowMove(boolean): Etmenin göç etme iznine sahip olup olmadığını gösteren mantıksal alan.

AllowClone (boolean): Etmenin kendi kopyasını yaratma iznine sahip olup olmadığını gösteren mantıksal alan.

allowWrite (boolean): Etmenin düğüm üzerinde çalışırken dosyalar üzerinde yazma (silme, değiştirme) izni olup olmadığını gösteren mantıksal alan.

allowRead (boolean) : Etmenin düğüm üzerinde çalışırken dosyalar üzerinde okuma izni olup olmadığını gösteren mantıksal alan.

allowOpenConnection (boolean) : Etmenin düğüm üzerinde soket yaratarak bağlantı kurma iznine sahip olup olmadığını gösteren mantıksal alan.

allowReceiveConnection (boolean) : Etmenin düğüm üzerinde soket yaratarak gelen bağlantı isteğini kabul etme iznine sahip olup olmadığını gösteren mantıksal alan.

allowAccessSystemSettings (boolean) : Etmenin, düğüm üzerindeki sistem değişkenlerine (print queue, clipboard, vb) erişim iznine sahip olup olmadığını gösteren mantıksal alan.

allowLoadClass (boolean) : Etmenin, yeni bir JAVA class yükleyicisi oluşturup oluşturamama iznine sahip olup olmadığını belirleyen alandır. Etmen programcısının değiştirilmiş özel bir class yükleyicisi kullanması bir güvenlik riski oluşturabilir. Örneğin, değiştirilmiş bir class yükleyicisi, etmene ait sınıf tanımlarını disk üzerinden değil de, ağdaki herhangi bir adresten yükleyebilir yetenekte olması tercih edilen bir özellik olmayabilir.

signature (String) : Etmen yaratıldığı zaman, kodundan elde edilen hash değeri (imza) etmen politikasının da sonuna eklenir. Etmen aktif duruma geçirilirken kod dosyasından bu hash değeri yeniden hesaplanarak durum dosyası ve politika dosyasındaki hash değerlerleri ile karşılaştırılır. Bu işlem etmen kod, durum yada politika dosyasında yetkisiz erişimler sonucu bir değişiklik yapılıp yapılmadığını anlamaya yardımcı olur. Değişiklik sezildiği anda etmen aktif duruma geçirilmez.

Varsayılan “Güvenlik Denetleyici” java nesnesinin herhangi bir JAVA API çağrısı üzerinde denetim yapabilmesi için çağrıyı yapan sınıfın bildirilmesi gerekmektedir. Diğer bir deyişle, yetkilendirme sınıf ismine göre yapılmaktadır. Oysa GES sunucusunun üzerinde çalışan bütün etmenler aynı “Agent” sınıfından türedikleri için sınıf isimleri hepsinde aynıdır. Bu nedenle etmene göre yetkilendirme verebilmek için GES sunucuları etmene ait “hash” değerlerini kullanır. JAVA, isimleri aynı olsa da her nesne için farklı bir hash üretir veya üretilmesine izin verir (sınıfın hash üreten metodu yeniden yazılarak). GES güvenlik denetleyicisi, bir JAVA API çağrısı yapıldığında, önce çağrıyı yapan nesnenin bir etmen olup olmadığını belirler. Eğer bir etmen ise hash değerini elde ederek bu hash değerine sahip etmenin kimliğine erişir. Çağrı yapılıp, çağrıyı yapan etmen de belirlendikten sonra etmen politikası belirlenir ve çağrı türüne göre politikanın bu çağrıya izin verip vermediği belirlenir. Daha önce bahsedildiği gibi kontrol edilebilir her çağrı için geliştirilen yeni “Security Manager” sınıfı içinde bu çağrıyı işleyen ve “check” ile başlayan bir metod ismi olmalıdır. Örneğin, dosya yazma ve okuma çağrıları için GES güvenlik denetleleyici modüllerindeki metodlar aşağıdaki gibidir.

```
public void checkRead(FileDescriptor f) {
    if (!this.checkAgentAccess(this.Access_checkRead)) {
        throw new SecurityException("Agent forbidden to read.");
    }
}

public void checkRead(String s) {
    if (!this.checkAgentAccess(this.Access_checkRead)) {
        throw new SecurityException("Agent forbidden to read.");
    }
}

public void checkRead(String s, Object o) {
    if (!this.checkAgentAccess(this.Access_checkRead)) {
        throw new SecurityException("Agent forbidden to read.");
    }
}

public void checkWrite(FileDescriptor f) {
    if (!this.checkAgentAccess(this.Access_checkWrite)) {
            throw new SecurityException(
                                "Agent forbidden to write.");
    }
}

public void checkWrite(String s) {
    if (!this.checkAgentAccess(this.Access_checkWrite)) {
            throw new SecurityException(
                                "Agent forbidden to write.");
    }
}
```

Yeni yazılan bu metodlarda çağrılar “checkAgentAccess” isimli yeni bir metoda yönlendirilmektedirler. “checkAgentAccess” isimli metod içinde, önce çağrıyı yapan

nesnenin etmen olup olmadığı saptanmakta, daha sonra çağrıyı yapan etmenin kimliği saptanmakta ve politikası okunarak ilgili çağrıya yürütme izni olup olmadığı kontrol edilip, çağrıya izin verilmekte veya "security exception" oluşturulmaktadır.

Etmen politikaları sadece etmenlere atanmamaktadır. Düğüm yöneticisi, isterse etmen politikalarını etmen sahiplerine göre düğümlere de atayabilir. Bu özellik sistemin esnekliğine büyük ölçüde katkı sağlamaktadır. Örneğin, bir düğüm yöneticisi güvendiği bir düğüm üzerinde yaratılıp, sisteme alınmış olan etmenlere kendi düğümü üzerinde yazma izni verebiliyorken, diğer düğümlerde bu izin verilmeyebilir. Ayrıca, izinler istendiği zaman da değiştirebilir.

9.4 Düğüm Politikaları

Düğüm politikaları, düğüm kaynaklarının yetkisiz etmenler tarafından kullanılmasını engellemek için kullanılır. Sistem, yetkilendirme desteklediğini etmen kaynağı ve etmen sahibine göre verebildiği için oldukça esnek bir çalışma ortamı sunarr. Etmen sahibinin tanımı daha önce yapılmıştı. Etmen sahibi bilgisi etmen ile birlikte taşınır ve etmen yaşam döngüsü boyunca değişmez. Etmen kaynağı ise şu şekilde tanımlanır.

Etmen kaynağı: Etmenin en son üzerinde çalıştığı GES sunucu adresidir. Örneğin bir etmen A sunucusu üzerinden B sunucusu üzerine göç ediyor ise , B sunucusu için etmen kaynağı A sunucu adresidir. Etmen kaynağı bilgisi etmen göç ettiği sürece değişmektedir ve saklanması gereken bir bilgi değildir.

Etmen sahibine göre politikalar etmen politikaları aracılığı ile verilir. Etmen programcısının etmeni sisteme sokarken ona bir politika atayabildiği gibi, düğüm yöneticisi de etmen sahiplerine bu politikaları atayabilir.

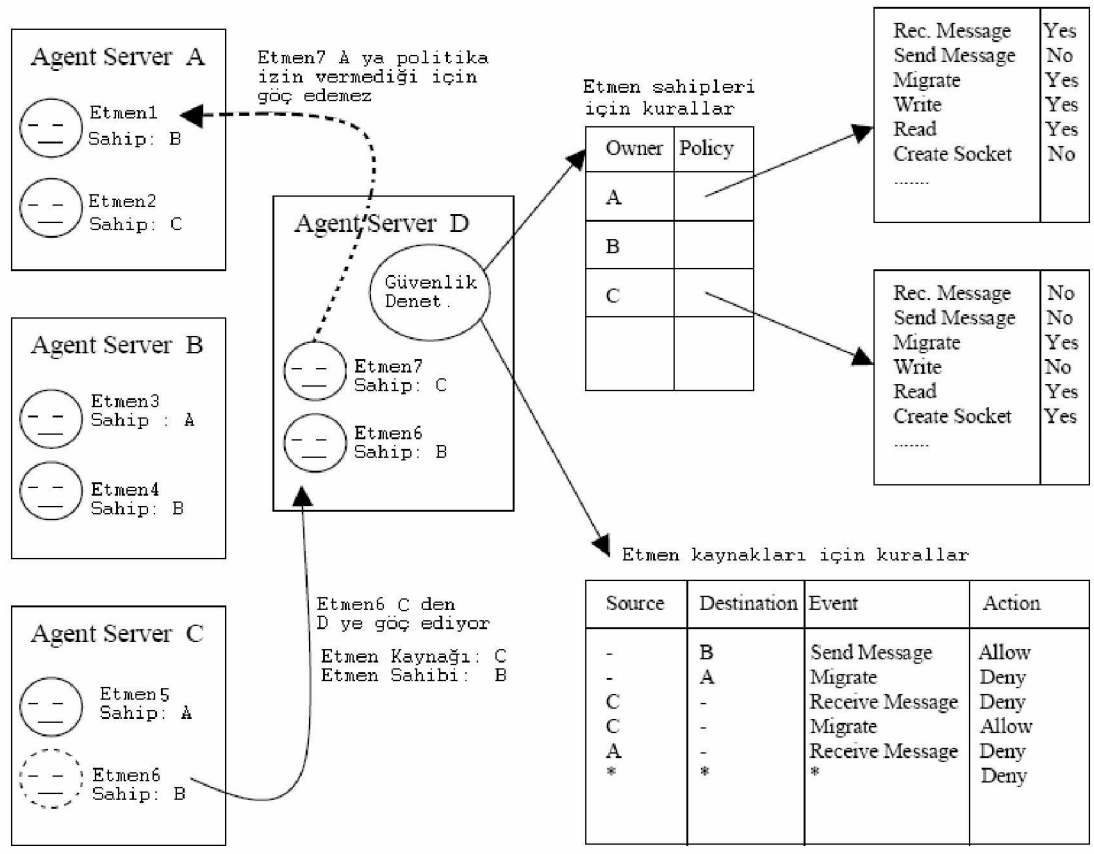
Etmen kaynağına göre ise, etmenin mesajlaşma ve göç etme aktiviteleri kısıtlanabilir. Bu sayede, örneğin, belirli bir düğüm üzerindeki etmenlerden mesaj alınmayabilir, yada belirli bir düğüme etmenlerin göç etmesine izin verilmeyebilir. Etmen kaynağına göre politikalar aktivite bazında verilir yani düğüm yöneticisi kendi düğümü üzerinde çalışan etmenlerin herhangi bir düğüm üzerindeki etmenlere mesaj göndermesine izin vermeyebilir ancak aynı düğümdeki etmenlerden mesaj almayı kabul edebilir. Yani kural kümesinde aynı etmen kaynağı için birden fazla kural bulunabilir.

Şekil 9.4.1 etmen kaynağına göre tanımlanan politika kurallara bir örnektir.

Direction	Server	Event	Action
Outbound	*	Migrate	Permit
Outbound	rmi://192.168.1.277:7123/AgentServer	Migrate	Deny
Inbound	*	Migrate	Permit
Inbound	rmi://192.168.1.77:7123/AgentServer	Receive Message	Deny
Outbound	*	Send Message	Permit
Inbound	*	Receive Message	Permit

Şekil 9.4.1 : Etmen kaynağına göre tanımlanan örnek politika kuralları

Verilen kurallara göre bu politikanın tanımlandığı düğüm üzerindeki etmenler 192.168.1.277 IP adresli 7123 nolu TCP portunda "AgentServer" ismiyle çalışan etmen sunucusuna göç edemezler. Yine 192.168.1.77 IP adresinde 7123 TCP portunda "AgentServer" ismiyle çalışan etmen sunucusunda çalışan hiç bir etmen bu bu politikaların tanımlı olduğu düğüme mesaj gönderemez. * kuralları çağrı ile ilişkili bir kural bulunamaz ise uygulanacak olan varsayılan kurallardır. Örneğin yukarıdaki kurallara göre herhangi bir düğüm için göç etme isteklerini içeren bir kural yok ise göç etme isteği kabul edilecektir (ilk kural), ya da bütün düğümlerdeki etmenlerden gelen mesajlar kabul edilecektir (son kural). Şekil 9.4.2 farklı etmen sahip ve kaynakları için farklı politikaların kullanımını göstermektedir.



Şekil 9.4.2 : Çalışma anında etmenler ve düğüm politikaları

Politikalar atanırken birbiri ile çakışma yaratan kuralların oluşturulmasına izin verilmez. Örneğin etmen kaynağına göre kural yazarken, aynı etmen kaynağına doğru göç etme isteğini yasaklayan bir kural varsa, buna izin veren başka bir kuralın tanımlanmasına izin verilmez. Aynı işleyiş etmen kaynağı için de geçerlidir. Aynı etmen sahibine birden fazla etmen politikası atanamaz.

Politikalar, etmen sunucusu başlarken konfigürasyon dosyasından okunarak belleğe alınır. Bellekte hash tablosu içinde tutulan politikaların işlenmesi de bu yüzden çok hızlı olur. Performans etkisi "politikaların performansa etkisi" başlığı altında ayrıntılı incelenecektir.

9.5 Politikaların Performansa Etkisi

Güvenlik denetiminden geçirilen her çağrının politikalara dayanarak kontrol edilmesinin performans açısından olumsuz bir yan etkisinin olması kaçınılmazdır. Bu performans düşüşü ortamda ne kadar etmen olduğu ve etmenlerin bu çağrı türlerini ne sıklıkla kullandığına bağlı olsa da, tek bir etmen için politika kullanımının performansa etkisini hesaplamak mümkündür.

Kontrol edilebilir tüm çağrılar etmen sunucuların daha önce açıklanan Güvenlik Denetleyicisi modülünde aynı algoritma ile işlenirler. Dolayısı ile, farklı çağrıların performansa etkisi farklı olmazlar. Bu nedenle ölçümler yapılırken örnek bir JAVA API çağrısı (dosyaya yazma) ele alınmıştır. Testler için aşağıdaki örnek java kodu kullanılmıştır. Bir döngü içinde güvenlik politikalarından geçen JAVA çağrıları yapılmış ve çağrıların tamamlanma süreleri kaydedilmiştir. Testler Celeron 2.4 Ghz işlemcili 512 MB ram li bir PC üzerinde yapılmıştır

```
1-    byte[] text = "Hello I am the helloworld agent".getBytes();
2-    FileOutputStream fo = null;
3-    for (int j = 1; j <= 10; j++) {
4-          long startTime = System.currentTimeMillis();
5-          for (int i = 1; i <= 50000; i++) {
6-                File a = new File("agentdata.txt");
7-                a.createNewFile(); //Politikalar kontrol ediliyor
8-                fo = new FileOutputStream(a);
9-                fo.write(text);
10-               fo.close();
11-         }
12-   long endTime = System.currentTimeMillis();
13-   System.out.println("Geçen zaman = ");
14-   System.out.println(endTime - startTime);
15-   }
```

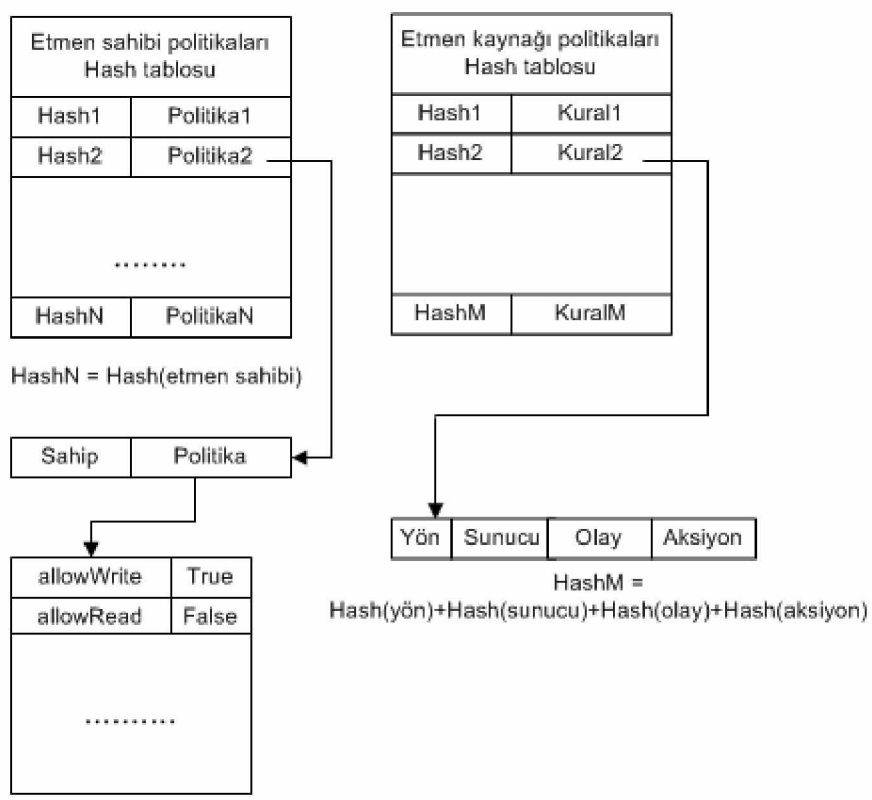
Testler öncelikle politikaların kullanım dışında olması durumunda gerçekleştirildi. Daha sonra farklı sayıda politika kuralları yazılarak testler tekrarlandı. Güvenlik kontrolü 7. satırdaki dosya yaratma fonksiyonu kullanıldığı an gerçeklenir. Sonuçlar yapılan testlerin ortalaması alınarak değerlendirilmiştir.

Tablo 9.5.1: Politikaların performans etkisi üzerine yapılan test sonuçları

Test	Politika pasif	Politikalar aktif Kural sayısı:1	Politikalar aktif Kural sayısı:10	Politikalar aktif Kural sayısı:30
1	31875 ms	35562 ms	35828 ms	35032 ms
2	30735 ms	35735 ms	33906 ms	35500 ms
3	29891 ms	35781 ms	36219 ms	37171 ms
4	34579 ms	35469 ms	37640 ms	35110 ms
5	29985 ms	35813 ms	34485 ms	35937 ms
6	28750 ms	36531 ms	36578 ms	36500 ms
7	36421 ms	35281 ms	35203 ms	33625 ms
8	32829 ms	35250 ms	37500 ms	37906 ms
9	31047 ms	36672 ms	35687 ms	35016 ms
10	30797 ms	36140 ms	35969 ms	36781 ms
ortalama	31691 ms	35823 ms	35902 ms	35858 ms

Sonuçlar incelendiğinde politika kullanımının performansa etkisi yaklaşık olarak %13 ((35823-31691)/31691) olduğu söylenebilir. Performanstaki bu düşüş, politika kullanımının sağladıkları düşünüldüğünde gözardı edilebilir seviyededir. Kural artışlarının performansı etkilemediği de görülmektedir. Bunun sebebi olarak da politika kurallarının bellekte bir hash tablosu şeklinde tutulması ve oldukça hızlı bir biçimde hash tablosunun taranmasıdır. Politikalar etmen sunucusunun çalışmaya başladığı anda konfigürasyon dosyasından okunur ve belleğe alınır. Dinamik olarak yapılan bütün değişiklikler önce bellekte yapılır, sonra konfigürasyon dosyasına yazılır.

Şekil 9.5.1 etmen sahibi ve etmen kaynağına göre oluşturulmuş politikaların bellekte tutuluş şeklini göstermektedir. Hash tablosundan herhangi bir satıra erişim çok hızlı gerçeklenir, çünkü her satır hash değeri tekildir. Hash tabloları bellekte indeksli bir yapıda tutulur, dolayısı ile anahtar değeri verilerek çok hızlı erişim mümkündür. Her kuraldan tekil bir hash değeri üretilir ve anahtar olarak bu hash değeri verilerek kural ile birlikte tabloda saklanır. Herhangi bir aktivite kontrol edilirken aktivite önce kural haline getirilir ve hash değeri tekrar oluşturularak tabloda bu hash değerinin olup olmadığı kontrol edilir. Eğer var ise ilgili kaydın aktiviteye izin verip vermediği kontrol edilir. Tabloda kayıt yoksa aktiviteye izin verilmez.



Şekil 9.5.1: Etmen sahibi ve kaynağı politikalarının bellek görünümü (+ katar birleştirme işareti olarak kullanılmıştır)

10. ETMEN AKTİVİTELERİNİN VE SİSTEM OLAYLARININ İZLENMESİ

GES mimarisi, belirli etmen aktiviteleri ve sistem olaylarının izlerinin saklanması ve gözlemlenmesine olanak sağlamaktadır. Etmen aktivitelerinin izlenebilmesi sadece gözlemleme açısından değil, güvenlik nedeniyle de önemlidir. Örneğin yazılan bir etmenin istenildiği gibi davranmadığı gözlenebilir yada beklenildiğinden çok daha fazla mesajlaşma aktiviteleri gerçeklediği farkedilebilir. Bunlar, yanlış bilgilendirme, diğer etmenler tarafından kötüye kullanılma yada atağa uğrama gibi güvenlik risklerinin habercisi olabilir. İzlenebilir etmen aktiviteleri şunlardır:

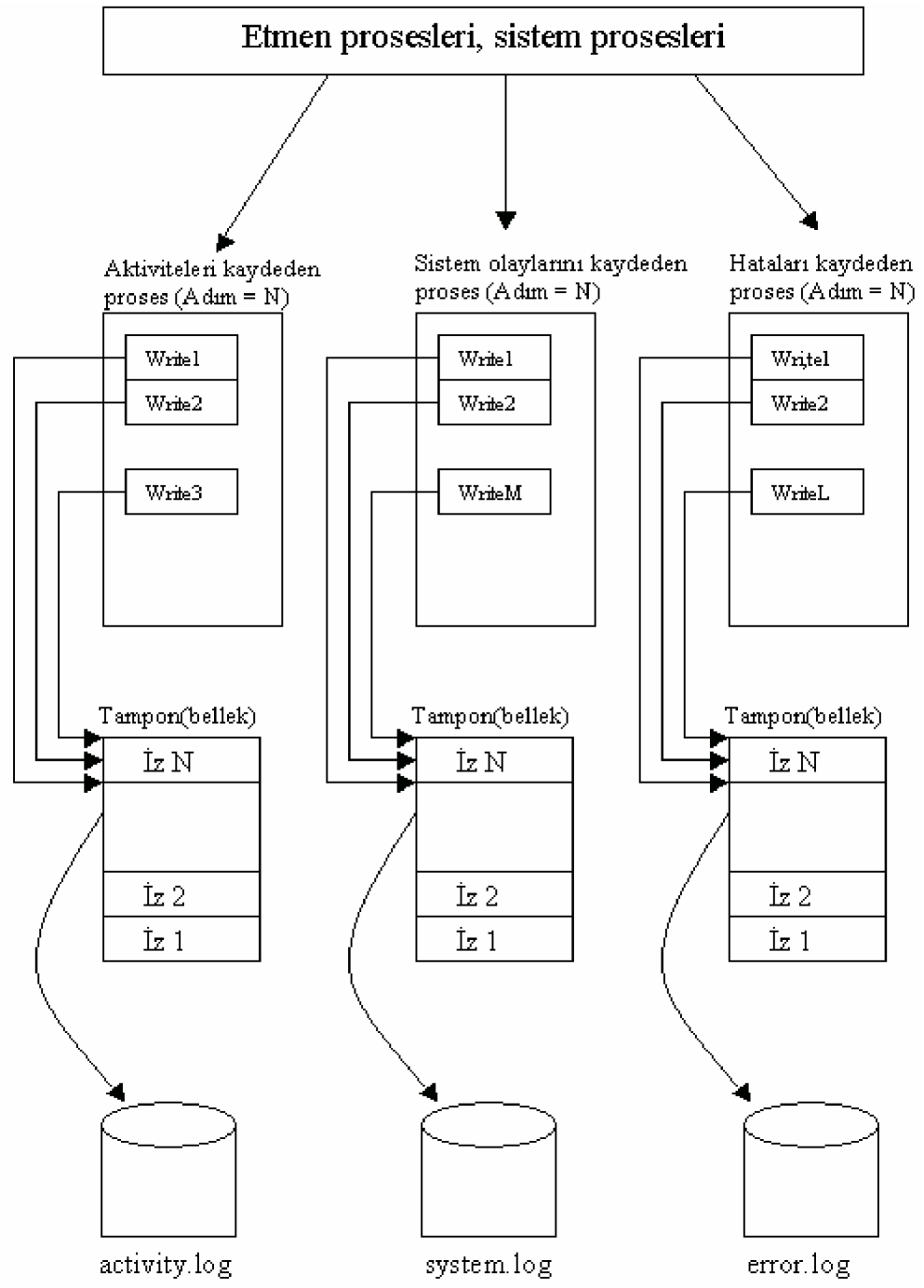
- Etmen yaratılması
- Etmenin aktif duruma geçirilmesi
- Etmenin pasif duruma geçirilmesi
- Etmenin mesaj yollaması
- Etmenin mesaj alması
- Etmenin göç etmesi

İzlenebilir sistem olayları ise şunlardır:

- Etmen aktiviteleri sırasında oluşan ait tüm hatalar (örneğin etmenin aktif olamaması)
- Güvenlik ile ilgili oluşan hatalar (örneğin etmen kodunun değiştiğinin sezilmesi)
- Etmen motoru proseslerinin izleri (örneğin geçerli bir gözleyici düzen sunucunun bulunamaması)
- Kimlik denetim hataları (örneğin GYGES ten kimlik denetimi yapılamaması)
- Bilet yaratılması ve sunuculara atanması

İzlerin kaydedilmesi için üç ayrı text türü dosya kullanılır. Bunlardan ilki etmen aktivitelerini, ikincisi sistem hatalarını, sonuncusu ise bilgi amaçlı sistem mesajlarını kaydetmek için oluşturulmuştur.

Çok sayıda etmenin bulunduğu bir etmen sunucusu üzerinde izlerin sürekli kaydediliyor olması performansa olumsuz yönde etki edebilir. Bu nedenle aktivite izleme prosesi ayarlanabilir sayıda izi önce belleğe alıp sonra diske yazabilme yeteneğine sahiptir. İzleme prosesleri ve yapısı şekil 10.1 de gösterilmiştir.



Şekil 10.1: İzleme prosesleri yapısı

GES motoru her bir iz türünü saklamak için üç ayrı iz kaydedici nesne (Logger) yaratır. Sistem çok iplikli çalıştığı için aynı anda bir çok iz oluşuyor olabilir. Dolayısı ile farklı proseslerden aynı anda bir çok iz kaydetme isteği gelebilir. İz kaydedici proses her iz kaydetme isteği için yeni bir metod çalıştırır (write metodu). Çağrılan metodun görevi kendisine verilen izi bellekteki kuyruğa yazmaktır. Bu metod aynı anda birden çok çağrı alabileceği için tampon bölgeye arişim karşılıklı dışlanma ilkesine göre çalışmaktadır.

İz kaydedici ana proseslerinin tampon bölge uzunlukları ayarlanabilir uzunluktadır ve "LogStep" isimli parametre ile belirlenir. Tampon bölge uzunluğu belirtilen değere ulaştığında en son tampon bölgeye yazan metod, tüm tampon içeriğini disk üzerindeki ilgili dosyaya yazan yeni bir iplik (thread) yaratır ve tampon temizlenir. Yine disk üzerindeki dosyaya yazma aynı anda birden çok ipliğin yazma ihtimaline karşı, karşılıklı dışlama ilkesine göre gerçeklenir.

İz kaydedici proses "Logger" isimli sınıf ile aşağıdaki şekilde tanımlanmıştır.

```
public class Logger {

  //Disk uzerindeki iz dosyası
  private  String FileName = null;
  //Tampon
  private  String[] List = null;
  private  int pos = 0;
  //Tampon bölge uzunlugu
  private  int BufferLen = 0;
  //Disk üzerindeki dosya için lock nesnesi
  private  Object FileLock = new Object();
  //Tampon bölge için lock nesnesi
  private  Object WriteLock = new Object();

  public  void Initialize(int bufferlen, String filename) {
    pos = 0;
    FileName = filename;
    BufferLen = bufferlen;
    List = new String[BufferLen];
  }
...
}
```

Tampondaki izleri disk üzerindeki ilgili dosyaya yazan prosesler ayrı iplikler halinde çalışır. Çünkü diske yazma işlemi zaman alan bir işlemdir bu nedenle izleri kaydeden ana proses beklememek için yeni bir iplik yaratır.

Aşağıda her bir iz dosyasının örnek içeriği verilmiştir.

activity.log

```
Sun Sep 10 12:49:58 EEST 2006 Agent received message :Test,
5J03MFRQB2U04UWUTD81NAJQI7KHM1GC(sample)
Sun Sep 10 12:50:12 EEST 2006 Agent received message :write,
5J03MFRQB2U04UWUTD81NAJQI7KHM1GC(sample)
Sun Sep 10 12:50:47 EEST 2006 Agent sends message:sendmessage to agent-
>TRPVSE1885KJ85I1CTGWSKLJ2X8AKZ7D(receiver), 5J03MFRQB2U04UWUTD81NAJQI7KHM1GC(sample)
Sun Sep 10 12:50:47 EEST 2006 Agent received message :sendmessage,
TRPVSE1885KJ85I1CTGWSKLJ2X8AKZ7D(receiver)
Sun Sep 10 12:51:07 EEST 2006 Agent Inactivating, TRPVSE1885KJ85I1CTGWSKLJ2X8AKZ7D
(receiver)
Sun Sep 10 12:51:33 EEST 2006 Agent Inactivating, 5J03MFRQB2U04UWUTD81NAJQI7KHM1GC
(sample)
Sun Sep 10 12:51:37 EEST 2006 Agent Activating,5J03MFRQB2U04UWUTD81NAJQI7KHM1GC(null)
Sun Sep 10 12:52:03 EEST 2006 Agent received message :TestMessage,
5J03MFRQB2U04UWUTD81NAJQI7KHM1GC(sample)
Sun Sep 10 12:53:32 EEST 2006 Agent sends message:SendMessage to agent-
>UGDX3TBA4FC3F8Z6DPKGJ1JE5L13EROV(receiver),5J03MFRQB2U04UWUTD81NAJQI7KHM1GC(sample)
Sun Sep 10 12:56:16 EEST 2006 Agent requested to move to-
>rmi://192.168.1.101:7123/AgentServer,5J03MFRQB2U04UWUTD81NAJQI7KHM1GC(sample)
Sun Sep 10 12:56:16 EEST 2006 Agent Inactivating, J03MFRQB2U04UWUTD81NAJQI7KHM1GC
(sample)
Sun Sep 10 12:57:20 EEST 2006 Agent Activating,5J03MFRQB2U04UWUTD81NAJQI7KHM1GC(null)
Sun Sep 10 13:02:30 EEST 2006 Agent received message :Write,
5J03MFRQB2U04UWUTD81NAJQI7KHM1GC(sample)
```

System.log

```
Sun Sep 10 20:15:20 EEST 2006 Updating agent host list
Sun Sep 10 20:15:20 EEST 2006 Sending agentlist to Master
Browser:rmi://192.168.1.40:7123/AgentServer
Sun Sep 10 20:15:40 EEST 2006 Updating agent host list
Sun Sep 10 20:15:40 EEST 2006 Sending agentlist to Master Browser:
rmi://192.168.1.40:7123/AgentServer
Sun Sep 10 20:15:54 EEST 2006 Using a new security manager :
rmi://192.168.1.40:7123/AgentServer
Sun Sep 10 20:15:54 EEST 2006 Generating a new ticket for host
:rmi://192.168.1.40:7123/AgentServer
Sun Sep 10 20:15:54 EEST 2006 The Secure Mobile Agent Server started succesfully
```

```
Sun Sep 10 20:15:54 EEST 2006 Starting ticket clean process
Sun Sep 10 20:15:54 EEST 2006 Starting ticket refresh process
Sun Sep 10 20:15:54 EEST 2006 Getting a new ticket from
:rmi://192.168.1.40:7123/AgentServer
Sun Sep 10 20:15:54 EEST 2006 Generating a new ticket for host
:rmi://192.168.1.40:7123/AgentServer
Sun Sep 10 20:15:56 EEST 2006 Using a new master browser :
rmi://192.168.1.40:7123/AgentServer
Sun Sep 10 20:16:04 EEST 2006 Updating agent host list
Sun Sep 10 20:16:24 EEST 2006 Sending agentlist to Master Browser:
rmi://192.168.1.40:7123/AgentServer
Sun Sep 10 20:16:26 EEST 2006 Successfully received agent source file: sender.zip
Sun Sep 10 20:16:36 EEST 2006 Successfully received agent source file: receiver.zip
Sun Sep 10 20:16:44 EEST 2006 Updating agent host list
Sun Sep 10 20:16:44 EEST 2006 Sending agentlist to Master
Browser:rmi://192.168.1.40:7123/AgentServer
Sun Sep 10 20:16:54 EEST 2006 Updating agent host list
```

Error.log

```
Sat Sep 09 22:13:30 EEST 2006
java.lang.ClassCastException
        at Server.AgentHostImplementation.lookupAgentHost
        (AgentHostImplementation.java:1553)
        at server.AgentHostImplementation.access$1100(AgentHostImplementation.java:20)
        at server.AgentHostImplementation$5.run(AgentHostImplementation.java:1983)
Sat Sep 09 22:13:30 EEST 2006 Couldn't connect to security manager :
rmi://192.168.1.101:7123/AgentServer
Sat Sep 09 22:13:30 EEST 2006 Error while notifying master browser
Sat Sep 09 22:13:30 EEST 2006 Couldn't find any alive master browser server
Sat Sep 09 22:13:30 EEST 2006 Couldn't connect to master browser :
rmi://192.168.1.101:7123/AgentServer
Sat Sep 09 22:13:30 EEST 2006 Couldn't find any alive master browser server
```

11. SİSTEM YÖNETİMİ

Mimari, sistem yönetimi için gelişmiş arayüzler sunmaktadır. Herhangi bir düğüm üzerinde çalışan etmen sunucusunu (GES Sunucu) uzak makinelerden tarayıcı aracılığı ile yönetmek mümkündür. Bunu sağlamak için GES, kendine özel ve standart HTTP protokolünü destekleyen bir web sunucu içermektedir. Web sunucunun özellikleri şu şekilde sıralanabilir.

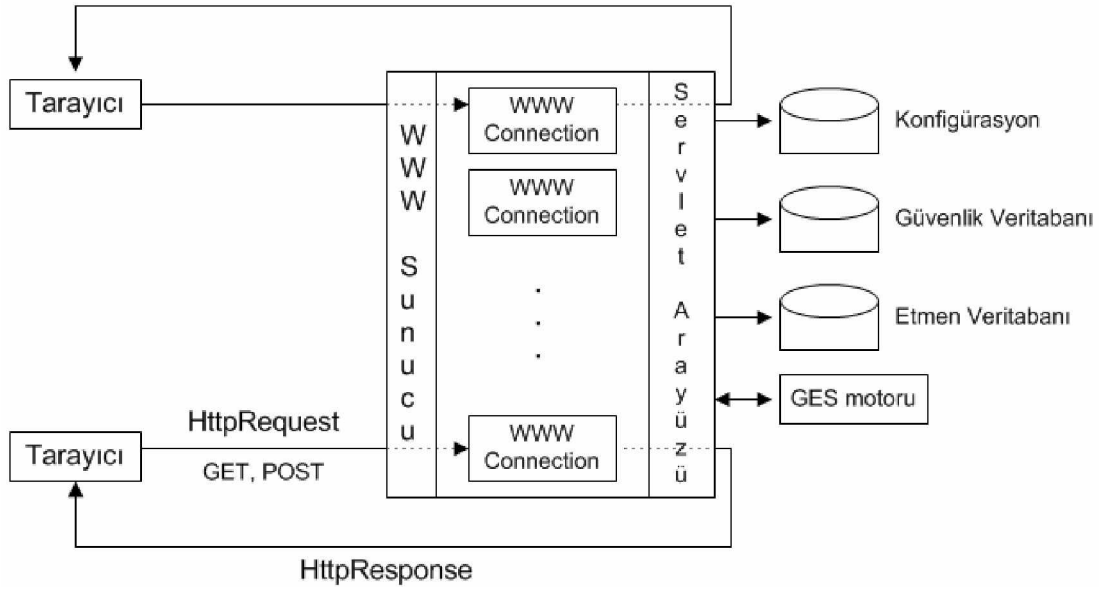
- Web sunucu java ile yazılmıştır ve çok iplikli çalışmayı destekler (multi threaded)
- Web sunucu standart HTTP 1.0 protokolünü destekler
- Web sunucu standart olmayan, kendine özgü bir servlet arayüzü sunar. Servlet arayüzü ayrıntılı olarak incelenecektir.
- Ayarlanabilir oturum zaman aşımını destekler.

Tüm sistemin yönetimi web sunucu tarafından karşılanmakta sonra ilgili servlet programı çalıştırılarak gelen isteğe yanıt verilmektedir. Yönetim arayüzleri ile sistem üzerinde aşağıda verilen işlemleri yapmak mümkündür.

- Sistem yönetimi için yetkili bir kullanıcı ile kullanıcı adı ve şifre verilerek sisteme bağlanılabilir. Yetkisiz erişimler engellenmiştir.
- Tüm sistem konfigürasyonu, örneğin sunucunun hangi düzende çalışacağı, GYGES adresi, oturum zaman aşımı, hangi izlerin kaydının saklanacağı, ayarlanabilir.
- Sunucu için sertifika üretilebilir ve diğer sunucuların sertifikaları güvenilir sertifika listesine eklenebilir.
- Sistemde o an çalışan aktif etmen listesi ve sunucu üzerinde pasif durumda olan etmen listesi görüntülenebilir.
- Etmenlere mesaj gönderilebilir, sunucu üzerindeki etmenler pasif duruma geçmeye zorlanabilir.

- Etmenlerin politikaları (sahibi olmak kaydıyla) değiştirilebilir.
- Düğüm politikası değiştirilebilir.
- Etmenler hakkında ayrıntılı bilgi edinilebilir (örneğin kaç mesaj gönderdiği, kaç mesaj aldığı, ne kadar zamandır sunucu üzerinde çalıştığı)
- Sistem izleri görüntülenebilir.
- Sisteme yeni bir etmen girişi yapılabilir.

Yönetim araçlarının genel bir görüntüsü şekil 11.1 de gösterilmiştir.



Şekil 11.1: Web sunucu ve sistem yönetim mimarisi

Web sunucu, tarayıcıdan gelen her isteği yeni bir iplik içinde karşılar. Böylece aynı anda farklı tarayıcılardan gelen farklı istekler de karşılanabilir. Web sunucu, HTTP protokolü standartlarına göre çalışır ve HTTP GET ve HTTP POST metodlarını destekler.

Web sunucu, "WebServer" sınıfı içinde tanımlanmıştır ve sürekli bir döngü içinde ayarlanabilen bir TCP port'u dinler. Bu porta bir HTTP isteği geldiği anda ise, yeni bir iplik yaratarak isteği yeni yaratılan bu iplik içinde karşılar ve dinlemeye devam eder. Her isteğin ele alındığı bu iplik ise "WebServerConnection" sınıfı içinde tanmlanmıştır.

11.1 Web Sunucu Sınıfı (WebServer)

Web sunucu, gelen her HTTP isteğini yeni bir iplik içinde karşılar. Ayrıca sisteme yetkili kullanıcı olarak bağlanan her kullanıcının oturumunu ayarlanabilir bir süre boyunca saklar. Bu süre boyunca sisteme bağlanmış kullanıcıdan herhangi bir istek gelmez ise oturum zaman aşımına uğrar ve kullanıcının yeniden kullanıcı adı ve şifresi istenir. "WebServer" sınıfı aşağıdaki şekilde tanımlanmıştır.

```
public class WebServer {
    static final String mServerName = "Mobile Agent Server/1.0";
    String mRoot;//Web sunucu varsayılan dizin ismi
    String mDefaultFile; //varsayılan ana sayfa
    char mPathSeparatorChar;
    Hashtable mMimeTypes; //istemciye gönderilecek dosya türleri
    boolean mStop = false;
    Hashtable sessionIDTable = new Hashtable();
    long sessionTimeout; //oturum zaman aşımı
    ......
}
```

WebServer sınıfından bir nesne yaratıldıktan sonra "run" isimli metodu çağrılarak sunucunun gelen istekleri dinlemesine başlanır. Ancak bundan önce istemciye gönderilecek dosya türleri belirlenir. Bir tarayıcı sunucunun gönderdiği verileri alırken ne türden bir veri almakta olduğunu bilmelidir. Örneğin gönderilen veri text türünden bir veri olabilir, ya da bir resim dosyası, ya da bunlardan farklı olarak bir ikili dosya (binary file) olabilir. Sunucu istemciye veri göndermeye başlamadan HTTP başlığında göndereceği verinin türünü ve uzunluğunu (Mime type) belirtmek zorundadır. Bu işleyiş şekli aslında tamamen HTTP protokolünün çalışma şekli ile ilgilidir.

Web sunucu sınıfının "run" metodu aşağıdaki şekilde tanımlanmıştır.

```
private void run(int port, String root, String defaultFile) {
        ServerSocket socket = null;

try {
mMimeTypes = new Hashtable();
mMimeTypes.put(new String(".zip"), new String("application/zip"));
mMimeTypes.put(new String(".gif"), new String("image/gif"));
mMimeTypes.put(new String(".jpg"), new String("image/jpeg"));
mMimeTypes.put(new String(".jpeg"), new String("image/jpeg"));
mMimeTypes.put(new String(".png"), new String("image/png"));
mMimeTypes.put(new String(".htm"), new String("text/html"));
mMimeTypes.put(new String(".html"), new String("text/html"));
mMimeTypes.put(new String(".text"), new String("text/plain"));
mMimeTypes.put(new String(".xml"), new String("text/plain"));
mMimeTypes.put(new String(".txt"), new String("text/plain"));
mMimeTypes.put(new String(".css"), new String("text/plain"));
mMimeTypes.put(new String(".java"), new String("text/plain"));
mMimeTypes.put(new String(".class"),new
```

```
                                String("application/octet-stream"));
mMimeTypes.put(new String(".jar"), new
                                String("application/octet-stream"));
mMimeTypes.put(new String(".js"), new String("text/html"));
        mRoot = root;
        mDefaultFile = defaultFile;
        socket = new ServerSocket(port);
        mPathSeparatorChar = File.separatorChar;
        System.out.println("Agent Server Web Daemon is running");
        } catch (Exception e) {
            System.out.println("WebServer.run/init: " + e);
            return;
        }

     try {
         while (!mStop) {
         Socket s = socket.accept();
         WebServerConnection c = new WebServerConnection(s, this);
         c.start();
         }
     } catch (IOException e) {
            System.out.println("WebServer.run/loop: " +
                                                  e.getMessage());
        }
    }
```

Görüldüğü gibi, dosya türleri belirlendikten sonra belirlenen sunucu portundan dinleme moduna geçilmekte, sonra istek geldiğinde yeni bir “WebServerConnection” nesnesi yaratılarak bundan sonraki işlemler ona aktarılıp, dinlemeye devam edilmektedir.

“WebServerConnection” nesnesi içinde gelen istek ayrıştırılır ve gerekli olan işlemler yapılır. Eğer bu bir kullanıcının sisteme yetkili olarak bağlanma isteği ise ve gerekli kullanıcı adı ve şifresi verilmiş ise kullanıcı için yeni bir oturum açılır ve oturum için verilen tekil kimlik bilgisi saklanır. Oturum kimlik bilgisi HTTP başlığı ile taşınan bir bilgidir; böylece kullanıcıdan daha sonra gelen istekler için kullanıcı adı ve şifresi istenmez. Oturum kimlik bilgilerini saklayan ve zamanı dolan oturumları silen gerekli metodlar da yine “WebServer” sınıfı içinde tanımlanmıştır.

Her oturumun ne zaman başladığı bilgisi ayrıca saklanır. Oturumların zaman aşımını kontrol eden metod, belirli sürelerle saklı olan oturumları tarar ve zaman aşımına uğramış olan oturumları siler. Yetkili bir kullanıcı ile bağlantı kurmuş bir oturuma ait yeni bir istek geldiğinde oturuma ait olan zaman sıfırlanır. Yani bir oturum için geçen zaman oturuma ait en son isteğin yapıldığı andan o ana kadar geçen süredir.

HTTP bağlantısına ait diğer bütün görevler “WebServerConnection” nesnesi tarafından yerine getirilir. Bu sınıf, gelen isteklerin gerekiyorsa servlet arayüzü ile gerçekleştirilmesini de sağlamak üzere gerekli metodları içermektedir.

11.2 Web Sunucu Bağlantı Sınıfı (WebServerConnection)

Web sunucu nesnesi, TCP soket üzerinden bir bağlantı isteği aldığı anda yeni bir "WebServerConnection" nesnesi yaratır. Bu yeni web sunucu bağlantı sınıfının görevi, gelen istek yeni bir HTTP bağlantı isteği ise, isteği ayrıştırmak ve yanıtı istemciye dönmektir. Gelen istek statik bir nesneyi (örneğin bir resim dosyası) alma isteği de olabilir, dinamik çıktı üretmesi gerken (örneğin bir servlet) bir istek de olabilir. Farklı istek türleri veya metodlarına göre (GET, POST) istekler farklı şekilde karşılanır.

Yeni bir web sunucu bağlantı nesnesinin ilk yapması gereken, tarayıcı ile kurulan soket nesnesi üzerinden gelen bütün veriyi okumaktır. Bu işlemi "run" isimli metodu içinde gerçekler.

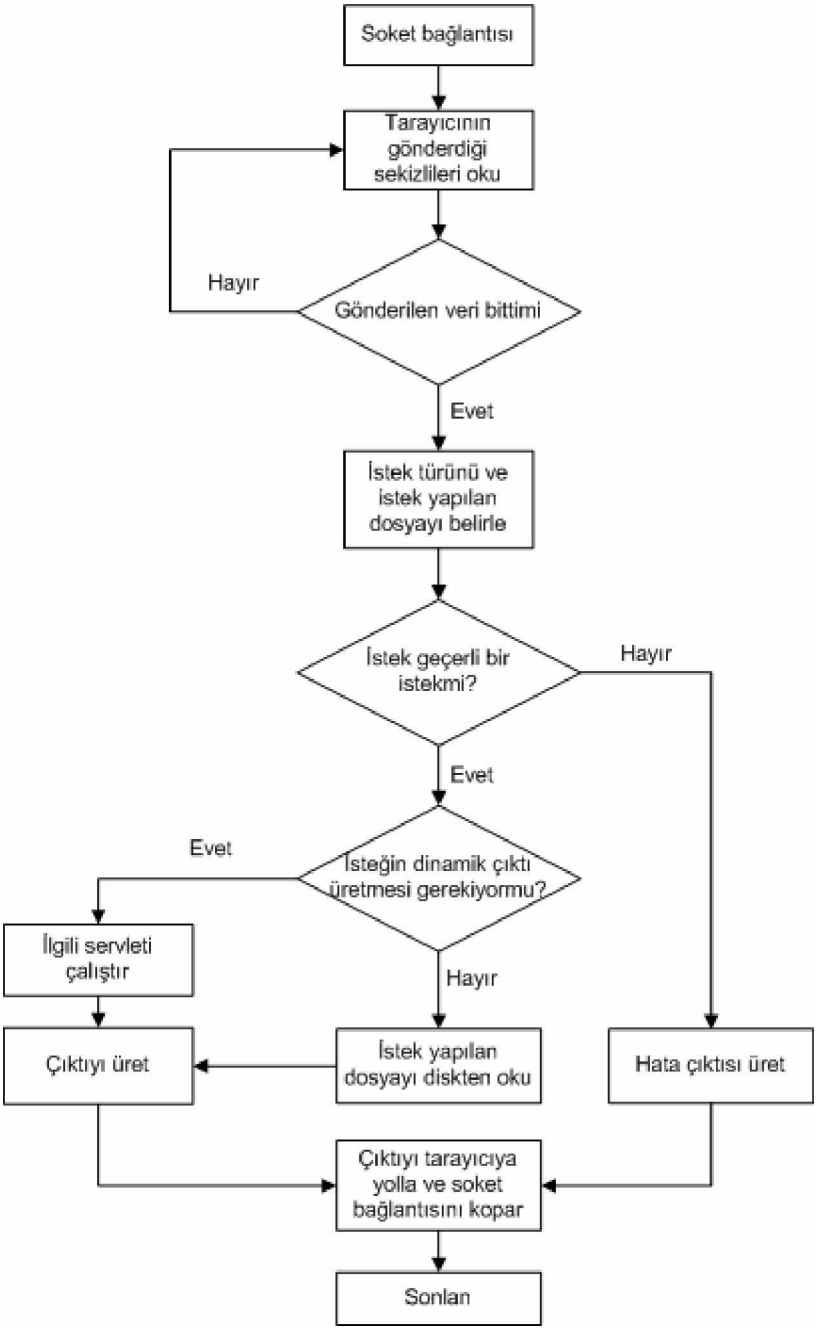
```
public void run() {
    try { this.setName("Web Server Connection Thread");
          BufferedReader input = new BufferedReader(new
                        InputStreamReader(mSocket.getInputStream(),
                                                          ENCODING));
          String request, name = null;
          int method = BAD_REQUEST;
          int len = -1;
          String serverVariableName = null;
          String serverVariableValue = null;

            while (true) {
                request = input.readLine();
                if (request == null) {
                    break;
                } ......
}
```

Görüldüğü gibi tarayıcının gönderdiği veri satır satır okunur. Bu istek HTTP standartlarına göre gelen bir istektir ve bir örneği şu şekildedir.

```
GET /test.html HTTP/1.0
Host: www.test.com
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
.NET CLR 1.1.4322)
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: tr
Connection: Keep-Alive
```

HTTP protokolü ayrıntılarına değinilmeyecektir ancak genel olarak yukarıdaki gibi bir istek yollayan bir tarayıcı ilk satırda istediği dosyayı (test.html), izleyen satırlarda da kendisiyle ilgili bazı bilgileri yollar. Burada önemli olan sunucunun kendisine ne türden bir istek geldiğini (GET ya da POST) ve neyin istendiğini ayrıştırılmasıdır.

Tarayıcının gönderdiği "Değişken_İsmi: Değişken Değeri" şeklindeki veriler de saklanır. Çünkü sunucu, bu değerlere uygun olarak, istemci tarayıcısının gerektirdiği farklı işlemler yürütmesi gerekebilir. Örneğin, istemci sıkıştırmayı destekleyen bir tarayıcıya sahipse, sunucu da göndereceği veriyi sıkıştırarak yollayabilir.



Şekil 11.2.1: Web isteklerinin karşılanması

Web sunucu bağlantı nesnesi, isteğe ait verinin tamamını alıp istek türünü ve istek yapılan dosya ismini de ayrıştırdıktan sonra, isteği yerine getirir ve istemciye yanıtı döner. Eğer istek statik bir dosya isteği ise, dosya diskten okunur ve istemci

tarayıcısına yollanır. İstek dinamik içerik üretecek bir istek ise ilgili servlet çalıştırılır ve çıktılar tarayıcıya yollanır.

Bir "WebServerConnection" nesnesi yaratılarak isteğin yeni bir iplik içinde karşılanması şekil 11.2.1 de akış diyagramı verilen algoritmaya göre yapılmaktadır.

Herhangi bir çıktı yine HTTP protokolü standartlarında tarayıcıya yollanırken önce gönderilecek verinin türü (resim, text yada ikili) ve verinin uzunluğu HTTP başlığında yollanır. Sonra da veri HTTP DATA paketleri şeklinde tarayıcıya yollanır. Verileri alan tarayıcı girdiyi ayrıştırır ve HTML standartlarında ekranda görüntüler.

Gönderilecek verinin dinamik bir çıktı olması durumunda önce ilgili servlet belirlenir ve çalıştırılarak çıktısı tarayıcıya yollanır. GES mimarisi kendine özel bir servlet kabı (Servlet Container) içerir.

11.3 Servlet Kabı ve Dinamik İşlemler

Web uygulamalarını ve JAVA'nın birlikteliği düşünüldüğünde akla ilk gelen applet türünden uygulamalardır. İstemci makinesindeki tarayıcı içinde açılıp çalışabilen JAVA appletleri, JAVA nın bütün olanakları kullanılarak istenildiği kadar gelişmiş özelliklere sahip olabilirler.

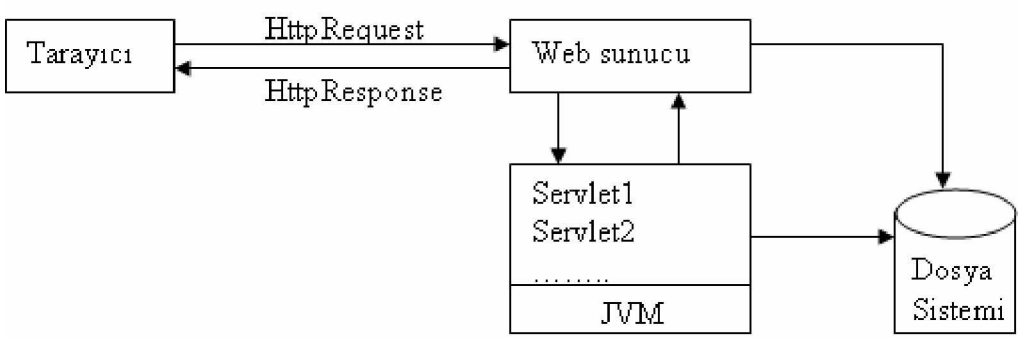
Sunucu tarafta web üzerinden gelen istekleri karşılayıp dinamik çıktı üretebilmek için bugüne kadar çok farklı yapılar önerilmiştir. Örneğin CGI[35] (Common Gateway Interface), herhangi bir dil ile yazılabilen programların bazı gereksinimlere bağlı kalarak web ortamında kullanımına izin veren ilk popüler yapılardan biriydi. Daha sonraları, web sunucusu ile entegre çalışan ve genellikle script dillerinin kullanıldığı yapılar geliştirildi. ASP[36], PHP[37] gibi diller bu alanda kullanılan en yaygın script dilleriydi. Script dilleri olmalarına rağmen, bu diller içinden bulunduğu platformun desteklediği bütün yapıları kullanmak da mümkündü. Bu özellik, esnek web uygulamaları geliştirebilmek için büyük bir katkı sağlamaktaydı.

Bu gelişmelerle birlikte bir web sunucu artık sadece statik sayfaları istemcilere sunan uygulamalar olmaktan çıkmış, dinamik çıktı üretebilecek uygulamalar geliştirilmesine olanak sağlayan yapılar (container) içermeye başlamışlardır.

Bu türden yapıların en önemli problemi, platform bağımsız olamadıkları için (örneğin asp uygulamaları windows ortamında çalışıyor, unix ortamında çalışmıyor) platform değişikliği gerektiği zaman yeniden yazılma gereksinimi duymalarıdır.

JAVA dilinin gelişmesiyle birlikte, platform bağımsız olma özelliği bu alanda da kendini göstermiş, esnek ve sunucu tarafında çalışan web uygulamaları geliştirirken artık JAVA tercih edilir duruma gelmiştir. Ancak JAVAnın bildiğimiz applet ve tek başına (standalone) çalışan uygulamaları bu amaca hizmet edecek özelliklere sahip değildir. Dolayısı ile, web sunucu olarak çalışan uygulamalar, beraberinde belirli JAVA kodlarını çalıştırıp çıktı üreten ve bu çıktının istemciye gönderilmesine olanak sağlayan fonksiyonları içermek için çeşitli mimariler içermeye başladılar. Sevlet[38] mimarisi bunun en iyi örneklerinden biridir.

Tıpkı bir applet standardı gibi, servlet standartları da oluşturulmuş, ve yazılacak servlet türü JAVA uygulamalarının yazım kuralları, gereksinimleri belirlenmiştir. Servlet kabı (servlet container) içeren her web sunucu üzerinde JAVA ile web uygulamaları yazmak mümkün hale gelmiştir. Şekil 11.3.1 servlet mimarisini göstermektedir.



Şekil 11.3.1: Servlet çalışma modeli

Sonuç olarak bir servlet, bir veya bir kaç JAVA sınıf dosyasından oluşan ve belirli kurallara uyularak yazılmış olan JAVA kodudur. Servlet, tarayıcıdan gelen istekleri alma ve tarayıcıya veri gönderme için gerekli olan fonksiyonları içerir. Servlet tarayıcı ile haberleşmek için gerekli olan protokol ayrıntıları ile ilgilenmez. Web sunucu servlet çıktısını tarayıcıya gönderecek yapıları zaten içermektedir.

GES mimarisi yönetim için sunucu tarafında çalışan web uygulamalarını çalıştırabilmek için kendine özgü bir servlet kabı içerir. JAVA'nın servlet standartlarına uygun olmayan bu kap, esnek web uygulamarı geliştirmek için gerekli bir çok fonksiyon içermektedir.

Tarayıcıdan gelebilecek HTTP isteği, HTTP GET veya HTTP POST isteği olabilir. Herhangi bir GES servlet'i dolayısı ile hem GET hem de POST için yerine getirilecek adımları içeren metodları uyguluyor olmalıdır. Bu nedenle bir GES servleti aşağıdaki gibi tanımlanmıştır.

```
public interface smasServlet {

    public void doGet(HttpRequest Request, HttpResponse Response);

    public void doPost(HttpRequest Request, HttpResponse Response);

}
```

Yazılan her yeni servlet, yukarıdaki JAVA arayüzünü uyguluyor olmalıdır. Bir servlet sınıfı içinde sadece bu iki metod değil, başka metodlar da bulunabilir ve bu yeni metodlar doGET ve doPost metodları içinden çağrılabilir.

Web sunucu, herhangi bir servlet'e HTTP GET isteği ulaştığı anda ilgili servlet sınıfının doGet metodunu, HTTP POST isteği ulaştığı anda ise doPost metodunu "Reflection API" kullanarak çağırır.

Herhangi bir isteğin bir servlet isteği olup olmadığı, istek yapılan dosyanın uzantısından anlaşılır. GES, servlet uzantıları için "action" uzantısını kullanır. Bu nedenle, örneğin http://www.server.com/test.action şeklindeki bir istek sunucuya geldiğinde, bunun "test" isimli bir servlet tarafından işleneceği anlaşılır.

"HttpRequest" ve "HttpResponse" sınıfları ise servletin yüksek düzeyde tarayıcı ile haberleşmesi için yazılmış iki sınıftır. HttpRequest nesnesi, servletin tarayıcının gönderdiği tüm veriyi veya tarayıcı hakkındaki bilgilerin alabilmesini sağlayan fonksiyonları içerirken, HttpResponse nesnesi, çıktıların tarayıcıya gönderilmesini sağlayan fonksiyonları içermektedir.

Bu sınıflara ait önemli metodlar tablo 11.3.1 de listelenmiştir.

Tablo 11.3.1 HttpRequest sınıfı metodları

Metod	**Kullanım Amacı**
public String getRawData()	Tarayıcıdan gelen verilerin hiç bir ayrıştırma işlemine sokulmadan olduğu gibi alınmasını sağlar.
public String getServerVariable(String name)	"name" isimli sunucu değişkeninin öğrenilmesini sağlar. Sunucu değişkenleri tarayıcının gönderdiği HTTP başlığındaki özel değişkenlerdir

	(cookie ismi, session ID, vb.)
public String getSessionID()	Mevcut oturumun kimliği (ID) öğrenilir.
public Enumeration getServerVariables()	Bütün sunucu değişlenleri liste halinde öğrenilir.
public void setServerVariable(String name, String value)	Herhangi bir sunucu değişkenine atama yapılır.
public String queryString(String name)	Tarayıcıdan gelen adres isteğinde "?" den sonra gelen ve değerleri "&" ile ayrılmış değişkenlerin içinden adı "name" olan değişkenin değeri öğrenilir.Örneğin; http://sunucu/a.action?no=5&sinif=10 gibi bir istekte "no" veya "sinif" değişkenlerinin değeri elde edilir.
public String getPostedVariable(String name)	Tarayıcının HTTP Post metodu ile gönderdiği bir değişkenin değeri elde edilir.

Örneğin "queryString" metodu tarayıcının gönderdiği HTTP isteğindeki "?variable_name=value" şeklindeki değerlerin öğrenilmesini sağlar. Mesela, tarayıcının adres bölümüne http://www.server.com/a.action?name=test&no=12 şeklinde yazılan bir istek web sunucuya geldiğinde queryString("name") metod çağrısının sonucu "test" ya da queryString("no") metod çağrısının sonucu "12" olacaktır. Bu metodun geri dönüş değeri her zaman String türündendir.

Önemli bir diğer metod ise "getPostedVariable" dır. Bir web sayfası üzerinden girilen bilgiler web sunucuya HTTP POST ile gönderildiğinde, sunucu tarafında servletin, tarayıcıdan hangi değişkenlerin geldiğini ve bu değişkenlerin değerlerinin ne olduğunu bilmesi gerekir.

Tarayıcı tarafından gönderilen tüm isteğin ayrıştırılması için (parsing) gerekli olan bütün fonksiyonlar "Utils" sınıfı içinde gerçeklenmiştir. Bu sınıf içindeki kod ayrıntılarına değinilmeyecektir.

"HttpResponse" sınıfı, HTTP protokol standartlarına uyacak haberleşme metodlarını içermektedir. "HttpResponse" nesnesi, servletin tarayıcıya veri göndermek istediği durumlarda ya da tarayıcı ile olan bağlantısını koparmak istediği durumlarda

kullanabileceği nesnedir. Bu sınıfın bazı önemli metodları tablo 11.3.2 de listelenmiştir.

Tablo 11.3.2 HttpResponse sınıfı metodları

Metod	**Kullanım Amacı**
public void saveSession(String sessionID)	Mevcut bağlantıya bir kimlik verilerek oturum açılması sağlanır.
public boolean isSessionAvailable(String sessionID)	Verilen kimlik bilgisinde bir oturumun mevcut olup olmadığı kontrol edilir.
private String getHead(String start, String end)	Http protokolü için başlık bilgisi oluşturulur.
public void writeByte(byte[] sendByte)	Belirtilen sekizli tarayıcıda yollanır.
public void writeText(String sendText)	Belirtlen metin tipi veri tarayıcıda yollanır.
public void writeHttpError(int code)	"code" değerli HTTP hata kodu tarayıcıya yollanır. Örneğin "401" HTTP kodu sunucuda dosya bulunamadı hatası anlamına gelir.
public void End()	Tarayıcı ile olan bağlantı sonlandırılır.
public void redirect(String site)	Tarayıcı "site" isimli adrese yönlendirilir.

Servlet örneğin tarayıcıya veri göndermek istediğinde writeText ("Hello Browser") çağrısı ile istediği veriyi yollayabilir. Gönderilecek veri HTML kodlarını içeren ifadelerde olabilir. Tarayıcı ile olan bağlantı "End()" metodu kullanılarak sonlandırılabilir.

Tarayıcıdan istek web sunucuya geldiğinde, web sunucu bunun bir servlet isteği olduğuna karar verirse (.action uzantısından) önce böyle bir servlet class dosyası olup olmadığına kontrol eder, eğer yok ise tarayıcıya 404 kodlu "HTTP NOT

FOUND" mesajı yollar. Servlet mevcut ise, isteğin GET yada POST olması durumuna göre ilgili servletin "doGet" yada "doPost" metodunu "Reflection API" ile çağırır. Örneğin bir servlet'e yapılan GET isteği web sunucuda aşağıdaki şekilde ele alınır.

Bütün servletler "WebServerConnection" nesnesi içinde çağrıldıkları için ayrı bir iplik içinde çalıştırılırlar.

GES sunucuları tarayıcılar ile yönetebilmek için bir çok servlet yazılmıştır. Bu servletlerin listesi tablo 11.3.3 te verilmiştir.

Tablo 11.3.3 Yönetim için yazılan servletler

Servlet	**Kullanım Amacı**
activateAgentServlet	Seçilmiş olan pasif etmen aktif duruma getirilir.
assignPolicyServlet	Etmene seçilen politika atanır.
createCertServlet	Sunucu sertifikası yaratmak için kullanılır.
delAgentPolicyServlet	Seçilmiş etmen politkasının silinmesini sağlar.
delAgentSourceServlet	Sunucuya ihraç edilmiş olan bir etmenin aktif duruma getirilmeden silinmesini sağlar.
deleteCertServlet	Güvenilen sertifikalardan seçilmiş olanın silinmesi sağlanır.
delOwnerPolicyServlet	Etmen sahibine göre belirlenen politikalardan seçilmiş olanı silinir.
delServerPolicyServlet	Etmen kaynağına göre belirlenen politikalardan seçilmiş olanlı silinir.
deployAgentServlet	Sunucuya etmen ihraç edilmesi sağlanır.
exportCertServlet	Sunucu sertifikası metin haline getirilerek yöneticiye sunulur.
getActiveAgentInfoServlet	Seçili aktif etmen hakkında ayrıntılı bilgi edinilmesini sağlar.
getAgentPoliciesServlet	Tanımlı etmen politikaları listelenir.
getAuthKeysServlet	GYGES üzerinde sunucular için tanımlı kimlik denetim anahtarları listelenir.

getHostPolicyServlet	Sunucu politikası listelenir.
getInactiveAgentInfoServlet	Pasif bir etmen hakkındaki bilgiler listelenir.
getParametersServlet	Sunucu konfigürasyon bilgileri listelenir.
getServerStatusServlet	Sunucu durumu (sunucu çalışıyormu gibi) öğrenilir.
getStatisticsServlet	Sunucu hakkında istatistiksel bilgiler elde edilir.
importCertServlet	Sunucunun güvenilir sertifika veritabanına diğer herhangi bir GES sunucunun sertifikası eklenir.
inactivateAgentServlet	Seçilen aktif etmen pasif duruma getirilir.
initializeSecDbServlet	Sunucu güvenlik veritabanı temizlenerek sıfırlanır.
loginServlet	Düğüm yöneticisi kimlik denetiminden geçerek yönetim ekranına girş yapar.
menuServlet	Düğüm yöneticisine menü ekranları gönderilir.
newAgentPolicyServlet	Yeni bir etmen politikası tanımlanır.
newAuthKeyServlet	GYGES üzerinde diğer herhangi bir GES sunucu için kimlik denetim bilgisi yaratılır.
saveHostPolicyServlet	Değiştirilen sunucu politikası saklanır.
sendMessageServlet	Seçilen aktif bir etmene mesaj gönderilir.
setParameterServlet	Sunucu konfigürasyon bilgileri değiştirilir.
startAgentServerServlet	Sunucu başlatılır.
stopAgentServerServlet	Sunucu durdurulur.
transferAgentServlet	Seçilen bir etmen göçe zorlanır.
viewAgentFileContentServlet	Seçili etmenin kodu (dosya isimleri) gösterilir.
viewAgentPolicyServlet	Aktif bir etmene atanmış politika gösterilir.
viewAgentSourcesServlet	Sunucuya ihraç edilmiş ancak henüz aktif edilmemiş etmen kodları

	(dosya isimleri) gösterilir.
viewAgentsServlet	Sistemdeki bütün aktif etmenler listelenir.
viewInactiveAgentsServlet	Sistemdeki bütün pasif etmenler listelenir.
viewLogServlet	Sistem ve etmen aktivitelerinin saklandığı dosyaların içeriği gösterilir.
viewOwnerPolicyServlet	Sunucu sahibine göre atanmış politikanın içeriği gösterilir.
viewSecDbServlet	Güvenlik veritabanının içeriği (sertifikalar) gösterilir.

Bunlardan biri örneğin kullanıcının yetkili bir kullanıcı olarak sisteme girmesini sağlayan "loginServlet" servletidir. GES sunucuyu yönetmek isteyen kişiye kullanıcı adı ve şifresinin sorulduğu bir ekran web sunucu tarafından yollanır. Şekil 11.3.2 bu ekranı göstermektedir.

Şekil 11.3.2: Yetkili kullanıcı giriş ekranı

Giriş ekranına ait HTML kodunun basitleştirilmiş hali ise aşağıdaki gibidir.

```
<HTML>
<BODY>
<FORM action="loginServlet.action" method="post" name="loginForm">
<TBODY>
<TR>
<TD>
<DIV align="left"><FONT size="2">Please enter your <I>Username</I>
and <I>Password</I> to
</DIV>
```

```
</TD></TR><TR>
<TD></TD>
<TD>
<INPUT type="text" size="22" name="txtusername">
</TD>
</TR>
<TR>
<TD>Password:</TD>
<TD>
<INPUT type="password" size="22" name="txtpassword">
</TD></TR><TR>
<TD></TD>
<TD>
<INPUT type="image" src="images/login.gif">
</FONT>
</TD>
</TR>
</FORM>
</BODY>
</HTML>
```

Görüldüğü gibi HTML FORM'un "action" alanına "loginServlet.action" yazılmıştır. Bunun anlamı; FORM'un giriş alanları doldurulup butona ya da enter tuşuna basıldığında, form üzerindeki değişkenler ve değerleriyle birlikte sunucudaki "loginServlet.action" dosyasına istek yapılacaktır. İsteği alan sunucu, istek yapılan dosyanın uzantısında "action" olduğu için bu isteğin bir servlet'i çalıştırma isteği olduğunu anlar ve "loginServlet.class" isimli JAVA kodunu arar. Sonrada "doPost" isimli metodunu çalıştırır.

Tarayıcıdan HTML FROM üzerinden girilen değişken ve değerleri (txtusername, txtpassword) "Request" nesnesi ile öğrenilir, gerekli yetkilendirme kontrolü yapıldıktan sonra kullanıcıya hata mesajı yada menu ekranı gönderilir.

GES'in servlet mimarisi, bir JAVA uygulamasının web üzerinde rahatça kullanılabilmesini sağlayan, çok iplikli, esnek fonksiyonlara sahiptir.

12. PERFORMANS ANALİZİ

Hareketli bir etmen sistemi, kendisinden beklenen gereksinimleri yerine getirirken mümkün olduğunca performanslı çalışmalıdır. Güvenlik ve performans birbirlerine karşı toleranslı olan kavramlar değillerdir. Genellikle güvenlik özellikleri arttırıldığında performans kaybı olur, ya da performans kazancı elde edilmek istendiğinde güvenlikten ödün verilmek zorunda kalınır. Nitekim "Politikaların Performans Etkisi" konu başlığı altında politika kullanımının ortalama %13 lük bir performans kaybına neden olduğu gösterilmişti. Bir hareketli etmen sisteminden beklenen; güvenlik ve performans arasında uyumlu bir denge sağlamasıdır. Bunun anlamı kesinlikle yerine getirilmesi gereken güvenlik önlemlerini almak, ancak bunları yaparken performansın da düşünülerek gerekli yapıların kurulmasıdır. Örneğin yine "Politikaların Peformans Etkisi" konusunda işlendiği gibi politika kuralları bellekte "hash" tabloları halinde tutulmuş bu nedenle kural sayısının performansa olumsuz bir etkisinin olmadığı gözlenmişti.

Bu bölümde GES'in çeşitli fonksiyonları yerine getirirken ölçülen performans değerleri incelenecektir. Performans üzerinde önemli olabilecek aşağıdaki aktivitelerin performans ölçümleri sırasıyla verilecektir.

- Etmen yaratılması
- Etmenin aktif duruma geçirilmesi
- Etmenin pasif duruma geçirilmesi
- Mesaj gönderme (Etmenin diğer bir etmene mesaj yollaması)
- Göç etme
- Diğer aktiviteler

Performans ölçümleri, 512 MB bellekli, celeron 2.4 Ghz işlemciye sahip makineler üzerinde yapılmıştır. Makineler arasındaki ağ hızı ise 10 Mbit tir ve pasif HUB (ethernet topolojisinde bilgisayarları birbirlerine bağlamak için kullanılan ağ cihazı) üzerinden birbirlerine bağlıdırlar.

12.1 Etmen Yaratılması

Bir etmenin yaratılma aşamasında etmen kod ve politikasının şifrelendiği, etmen sahibi bilgisinin oluşturulduğu ve etmene ait ilk değer atamalarının yapıldığı "GES Etmenleri" konu başlığı altında incelenmişti. Görüldüğü gibi etmenin yaratılma zamanını etkileyecek farklı nedenler vardır. Yaratılma zamanını etkileyebilecek faktörler şu şekilde sıralanabilir.

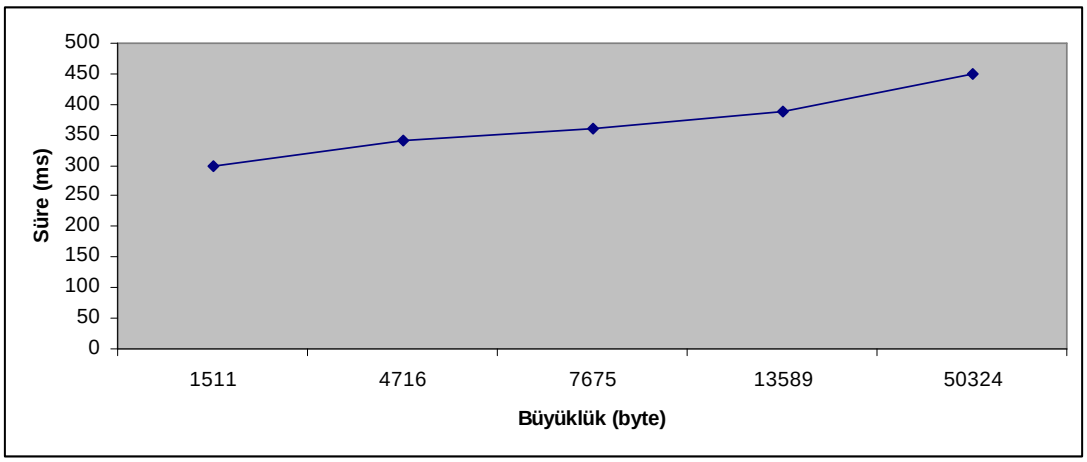
- Etmene ait "class" sayısı ve bu java class dosyalarının büyüklüğü.
- Etmene ilk değer atamalarının yapıldığı "OnCreate" metodunun içeriği.
- Disk okuma-yazma hızı.

"OnCreate" metodunun içeriği programcıya bağımlı olduğu için etmenin yaratılma aşaması GES açısından bu metodun ilk satırının işletilmesiyle sona ermiş kabul edilebilir. Farklı sayıda ve büyüklükte class dosyasına sahip bir etmenin ölçülmüş yaratılma süreleri tablo 12.1.1 de verilmiştir.

Tablo 12.1.1: Etmen yaratılma süreleri

Test	Etmen class sayısı	Toplam büyüklük (byte)	Yaratılma için geçen süre(ms)
1	1	1511	300
2	2	4716	340
3	3	7675	361
4	5	13589	389
5	20	50324	450

Görüldüğü gibi etmen dosya sayısı ve toplam etmen kod dosyalarının büyüklüğü arttıkça yaratılma zamanı da artmaktadır. Ancak yaratılma sürelerindeki artışın büyüklükteki artışa göre çok daha az ivmeli olduğu dikkat çekmektedir. Test sonuçları şekil 12.1.1 de grafik olarak gösterilmiştir.



Şekil 12.1.1: Etmen yaratılma süresi-etmen büyüklüğü ilişkisi

12.2 Etmenin Aktif Duruma Getirilmesi

Etmenin aktif duruma getirilme süresi etmen için kritik olabilecek bir performans değeridir. Yaratılma süresi etmenin tüm yaşam döngüsü boyunca sadece bir kez olurken, aktif olma için harcanan süre bir çok kez tekrarlanabilir. Etmen göç ettikten sonra da hedef düğüm üzerinde her defasında aktif duruma geçirilmelidir.

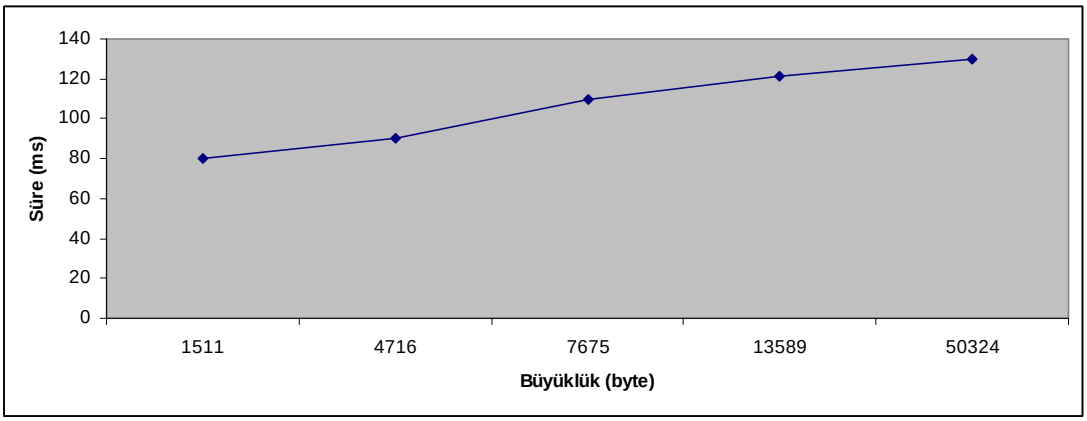
Etmen aktif duruma geçirilirken, etmen kod, durum ve politika dosyasının disk üzerindeki şifreli halleri çözülür, bu dosyalarda GES harici bir etkiden dolayı değişiklik olup olmadığı kontrolü yapılır, etmen için bir etmen kabuğu yaratılır ve bundan sonra denetim kabuğa aktarılır. Kabuk, etmene ait mesaj giriş-çıkış kuyruklarını yaratır, bu kuyrukları dinleyen iplikleri canlandırır ve etmen politikası ile etmeni ilişkilendirir. Daha sonra etmen ayrı bir iplik içinde çalıştırılmaya başlanırken, etmen bilgisi "Gözleyici" düzende çalışan GES sunucuya iletilir.

Etmenin aktif duruma geçtiği an, etmene ait "OnActivate" metodunun çalıştırılmaya başladığı andır. Aktif edilme süresi, disk üzerinde pasif duran bir etmene yönetim ekranından aktif edilme komutu verilmesinden, "OnActivate" metodunun çalıştırılmasına kadar geçen süre olarak alınmıştır. Ölçüm sonuçları tablo 12.2.1 de ve grafiği şekil 12.2.1 de gösterilmiştir.

Tablo 12.2.1: Etmen aktif edilme süreleri

Test	Etmen class sayısı	Toplam büyüklük (byte)	Yaratılma için geçen süre(ms)
1	1	1511	80
2	2	4716	90
3	3	7675	110
4	5	13589	121

5	20	50324	130



Şekil 12.2.1: Etme aktif edilme süresi-etmen büyüklüğü ilişkisi

12.3 Etmenin Pasif Duruma Getirilmesi

Etmenin pasif duruma geçirilmesi sırasında, etmenin pasif duruma geçirildiği "Gözleyici" düzende çalışan GES sunucuya iletilir,etmene ait "OnInactivate" metodu çalıştırılır, etmene ait mesaj giriş-çıkış kuyruklarının silinir, bunları dinleyen iplikler sonlandırılır, etmen son durumu saklanarak diske yazılır (şifrelenerek) ve kabuk yok edilir. Etmenin kod büyüklüğünden çok, son güncel durumunun büyüklüğü pasif duruma geçirilme süresini etkileyen faktördür. Çünkü pasif duruma geçirilme aşamasında sadece güncel durum şifrelenerek diske yeniden yazılır. Etmenin pasif duruma geçirilme süresi "OnInactivate" metodu içeriğine bağlı olduğu için bu metod test için kullanılan etmenlerde boş bırakılmıştır. Dolayısı ile sadece sistemin bu aktivite için harcadığı süre hesaplanmıştır. "Gözleyici" düzende çalışan GES sunucunun konumu etmeni pasif duruma geçirme zamanını etkilemez çünkü etmenin pasif olduğu bilgisi ayrı bir iplik içinde "Gözleyici" düzende çalışan GES sunucuya bildirilir. Örnek ölçümde etmenin üzerinde çalıştığı GES sunucu aynı zamanda "Gözleyici" olarak çalışan GES sunucu olarak ayarlanmıştır.

Tablo 12.3.1: Etmen pasif edilme süreleri

Test	Etmen class sayısı	Toplam büyüklük (byte)	Son durum büyüklüğü (byte)	Pasif duruma geçirilme süresi (ms)
1	1	1511	576	20
2	2	4716	576	20
3	3	7675	576	20
4	5	13589	576	20
5	20	50324	576	20
6	12	30925	656	80

Görüldüğü gibi etmenin kod büyüklüğü pasif duruma geçirilmesinde önemli değildir. Etmen son durum büyüklüğü büyüdükçe pasif edilme süresi de artmaktadır ki bu da beklenen bir durumdur.

12.4 Mesaj Gönderme

Etmen için önemli bir performans kriteri de gönderdiği mesajın hedef etmene ulaştırılma süresidir. Mesaj gönderme süresini etkileyen faktörler, mesajın büyüklüğü ve hedef etmenin konumu olarak sıralanabilir. Gönderici ve hedef etmen farklı düğümler üzerinde çalışıyorlarsa bu süre daha uzun olacaktır çünkü araya ağ haberleşmesi için geçen süre de girmektedir. Mesaj büyüklüğü iletim zamanını arttıracağı için mesaj iletme süresini doğrudan etkiler. Mesaj gönderme süresi, gönderici etmenin mesaj gönderme çağrısı ile alıcı etmenin yeni bir mesajın varlığından haberdar edilir edilmez, mesajı giriş kuyruğundan çekmesi arasında geçen zaman olarak tanımlanmıştır. Farklı mesaj boyları ve alıcı ve göndericinin aynı ve farklı düğümler üzerinde olduğu durumlar için ölçülen süreler aşağıdaki tablolarda verilmiştir.

Tablo 12.4.1: Mesaj gönderme süreleri

Test	Mesaj boyutu (byte)	Hedef etmen konumu	Mesaj iletim süresi (ms)
1	100	Yerel	≈ 20
2	1000	Yerel	≈ 20
3	10000	Yerel	≈ 20
4	100000	Yerel	≈ 20
5	1000000	Yerel	≈ 20
6	100	Uzak	≈ 300
7	1000	Uzak	≈ 300
8	10000	Uzak	≈ 300
9	100000	Uzak	≈ 500
10	1000000	Uzak	≈ 2200

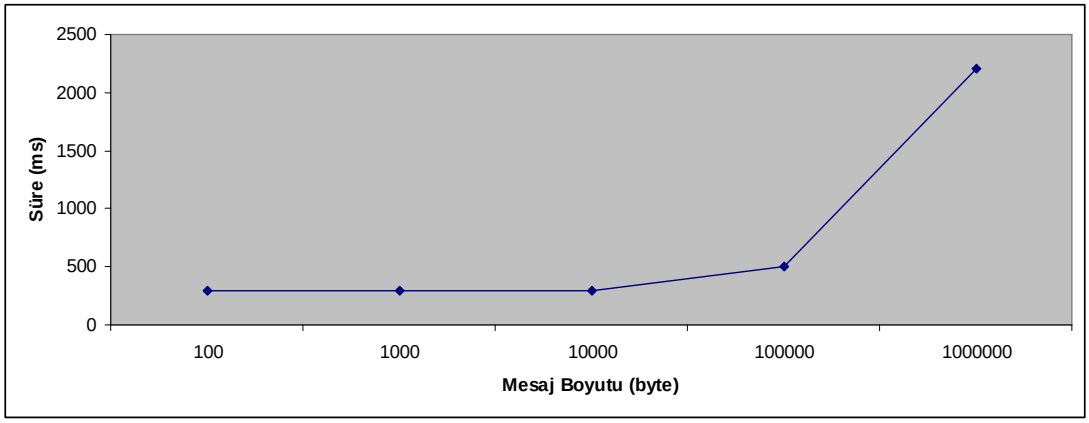
Görüldüğü gibi mesaj iletim süreleri gönderici ve alıcı etmenin aynı düğüm üzerinde olması durumunda mesajın boyutu ile değişmemektedir. Gerçekte mesaj boyutunun megabyte seviyelerinde arttırılması durumunda bu sürelerin artması beklenir ancak gerçek hayatta bir etmenin ilettiği mesajın megabyte seviyelerinde olması beklenmez. Mesaj gönderme süresinin etmenlerin aynı düğüm üzerinde olması durumunda mesaj boyutu artmasına rağmen değişmemesinin nedeni, bütün işlemlerin bellekte çok hızlı bir şekilde gerçekleniyor olmasıdır.

Etmenlerin farklı düğümler üzerinde bulunması durumunda mesaj boyutunun artması ile birlikte mesaj gönderme süresinin de artması gerektiği tahmin edilebilir. Nitekim

sonuçlar bunu doğrulamaktadır. Mesaj iletimi farklı düğümler arasında olduğu zaman mesajın tamamı ağ üzerinden önce serileştirilip sonra iletim sonunda tekrar mesaj nesnesine çevrilmektedir ki, bu da süreyi arttıran önemli bir etkendir. 10000 sekizli mesaj uzunluğuna kadar mesaj gönderme süresinin sabit kaldığı da dikkat çekicidir. Bu ise RMI protokolü ile ilgilidir. Yani RMI açısından 100 sekizli ile 10000 sekizli veriyi iletmek aynı süreye malolmaktadır.

Aynı düğüm üzerindeki etmenlerin haberleşmelerinde mesaj büyüklüğünün artmasına rağmen mesaj iletim süresinin artmaması, GES mesajlaşma alt yapısının oldukça etkin çalıştığını göstermektedir. Farklı düğümlerdeki etmenlerin haberleşmesi sırasında mesaj büyüklüğüne bağlı olarak artan iletim süresi tamamen ağ ortamında verinin iletilmesinden kaynaklanmaktadır.

Şekil 12.4.1 farklı düğümler üzerinde çalışan iki etmenin mesaj iletimi ile mesaj boyutu arasındaki ikişkiyi göstermektedir.



Şekil 12.4.1: Uzak etmenler arası mesaj iletim süresi-mesaj boyutu ilişkisi

12.5 Göç Etme

Etmenin göç etmesi, yerel düğüm üzerindeki etmenin kod, son durum ve politikasının uzak düğüme aktarılması, yerel düğüm üzerinden etmenin pasif duruma geçirilmesi ve uzak düğümde aktif edilmesi aşamalarını kapsamaktadır. Etmenin güncel yerinin "Gözleyici" GES sunucuya bildirilmesi de ayrıca iletim sonunda gerçeklenir.

Etmenin bulunduğu düğümden farklı bir düğüm üzerine göç etmesi etmen boyutu ve düğümler arası ağ ortamının hızı ile doğrudan orantılıdır. Ölçüm değerleri 10 Mbit ağ ortamında yapıldığı için etmen boyutu değiştirilerek etmen göç etme süreleri ölçülmüştür. Göç etme süresi, yerel etmenin "Move" çağrısı yaparak göç etme

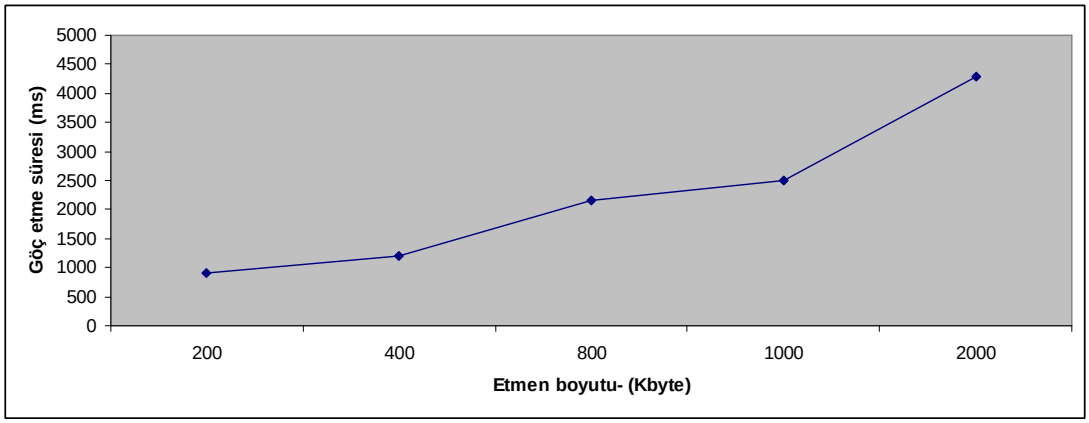
isteğini bildirmesi ile uzak düğümde aktif edilerek "OnActivate" metodunun çalıştırılmaya başlatılması arasında geçen süre olarak alınmıştır.

Tablo 12.5.1: Göç etme süreleri

Test	Toplam etmen boyutu Kod+durum+politika – (Kbyte)	Göç etme süresi (ms)
1	200	900
2	400	1200
3	800	2150
4	1000	2500
5	2000	4300

Görüldüğü gibi etmen boyutu artıkça doğru orantılı olarak etmen göç etme süresi de artmaktadır. Mesaj iletim süreleri incelenirken bin sekizli büyüklüğünde bir mesajın iletim süresinin 2200 ms olduğu ölçülmüştü, bu ölçümlerde ise aynı büyüklükteki bir etmenin göç etme süresi 2500 ms olarak bulunmuştur. Göç etme işleminde ek olarak daha başka işlemler de yapıldığı için (etmeni pasif duruma geçirme, aktif etme, etmen durumunu Gözleyici GES'e iletme, vb.) biraz daha uzun bir süre bulunmuş olması sonuçların tutarlı olduğunu göstermektedir.

Şekil 12.5.1 etmen boyutu ile göç etme süresi arasındaki ilişkiyi grafiksel olarak göstermektedir.



Şekil 12.5.1: Etmen boyutu-göç etme süresi ilişkisi

12.6 Diğer Aktiviteler

Son olarak GES sunucularının çeşitli aktiviteleri için ölçülmüş performans değerleri verilecektir. Bu aktiviteler ve ölçüm değerleri tablo 12.6.1 de verilmiştir.

Tablo 12.6.1: Çeşitli aktiviteler ve süreleri

Aktivite	Süre (ms)
Uzak GYGES ten kimlik denetiminden geçme	< 50
Etmenin uzak GGES kullanılarak sistemdeki tüm etmen listesini alması	< 90
Etmen durum değişimlerinin uzak GGES'e bildirilmesi	< 150
Bir izin diske yazılması	< 1
Politikaların diskten okunarak belleğe alınması	< 60

Performans ölçüm değerleri kullanılan ortama çok bağımlıdır. Örneğin bilgisayarlar birbirlerine 10Mbit değil de 100Mbit yada Gbit hızlı bağlantılarla bağlı iseler bir etmenin göç etmen süresi yukarıda bulunan sonuçtan çok daha kısa olacaktır. Ya da etmen internet ortamında göç ediyorsa bu süre daha da uzun olacaktır. Etmenlerin aktif ve pasif durumlara geçirilmesi, izlerin diske yazılması gibi işlemler de bilgisayarın disk okuma-yazma hızı da diğer önemli bir etkendir.

13. ÖRNEK ÇALIŞMALAR

Bu bölümde GES mimarisinin özellikleri ve fonksiyonları, geliştirilen örnek etmenlerin çalıştırılmasıyla incelenecektir. Etmenlerin çalışma düzenleri, etmen sunucu konfigürasyonları değiştirilerek gözlemlenecektir.

13.1 Örnek Uygulama-1

Birinci uygulama, dağıtık veri işlemeye örnek olabilecek bir senaryoyu gerçekleştirmektedir. Örnek problem olarak n adet mxm boyutundaki matrisin çarpımının determinant hesabı seçilmiştir. Problem, aşağıdaki matematiksel ifadenin sonucunun bulunması olarak gösterilebilir.

$$R = det(M_1 \times M_2 \times ...x\ M_n)$$

Bu matematiksel ifadenin hesabı, dağıtık işlemeyi kolaylaştıracak aşağıdaki eşitlik kullanılarak çözümlenebilir.

$$R = det(M_1 \times M_2 \times ...x\ M_n) = det(M_1) \times det(M_2) \times ...x\ det(M_n)$$

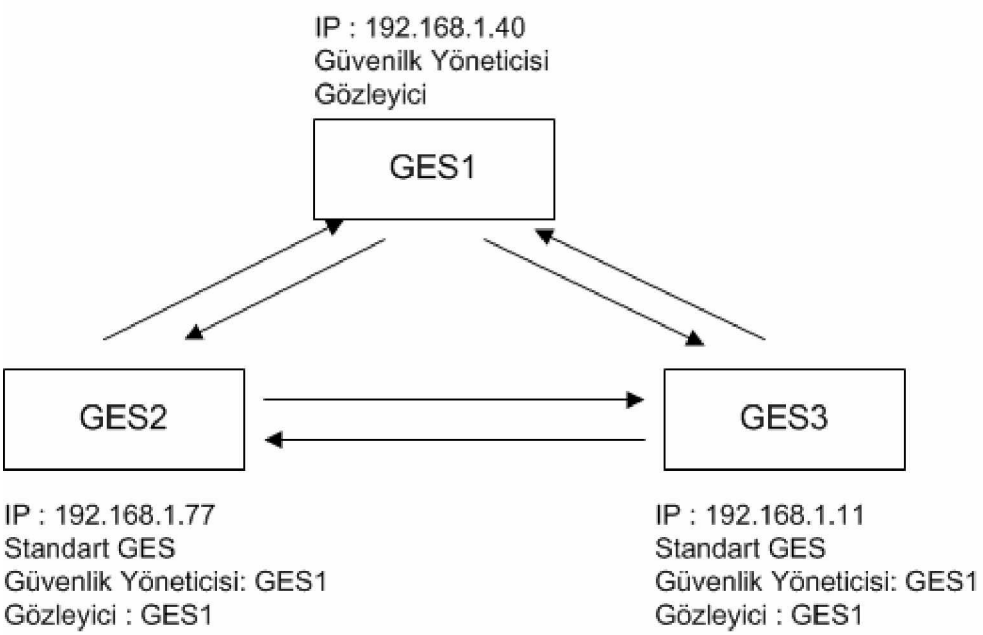
Buna ifade, n adet matrisin çarpımının determinantının, matrislerin aynı ayrı determinantlarının çarpımına eşit olduğunu göstermektedir. Bu eşitlikten faydalanarak iki farklı etmen türü yazılmıştır.

İlk etmen n adet mxm büyüklüğündeki matrisin hesabını k adet ikinci tür etmen arasında paylaştırır. “detManager” isimli bu etmenin görevi; grafik bir arayüz sağlayarak kullanıcıdan matrisleri almak, sonra bu matrislerin çarpımının determinantını hesaplamak için k adet ikinci tür etmenlere matrisleri eşit şekilde göndererek determinantlarını hesaplattırmak ve gelen sonuçları çarparak nihai determinant sonucunu elde etmektir. “detCalculator” isimli ikinci etmen türünün görevi; “detManager” etmeninin gönderdiği matrisin determinantını hesaplamak ve sonucu “detManager” etmenine yollamaktır. Birinci ve ikinci tür etmenler, ayrıca kendilerine gelen kopya yaratma (cloning), göç etme gibi mesajları da değerlendirirler.

13.1.1 Örnek Sistem Mimarileri ve Etmenlerin Davranışları

GES mimarisinin özelliklerini iyi inceleyebilmek için etmenler farklı sistem konfigürasyonlarında çalıştırılacaklardır. GES mimarisi, sistem sürekliliğini sağlamak için çok farklı şekillerde ayarlanabilir. Örnek iki tür sistem konfigürasyonu ele alınacaktır.

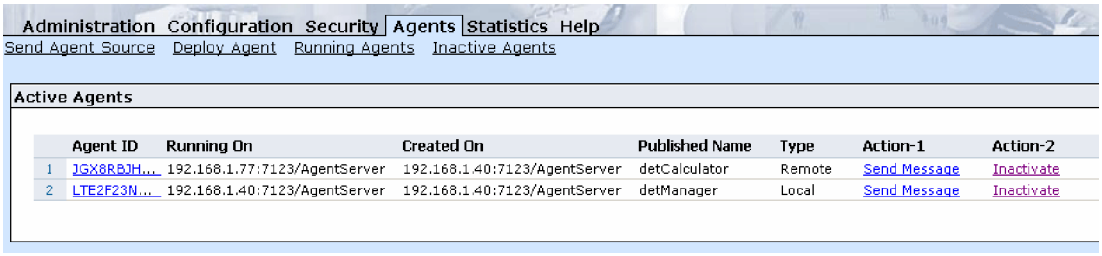
Birinci senaryoda sistemde üç adet farklı bilgisayarlarda çalışan etmen sunucuları bulunmaktadır. Etmen sunucuları şekil 13.1.1.1 de verilen örnek sistem konfigürasyonunu içerecek şekilde ayarlanacaktır.



Şekil 13.1.1.1: Birinci senaryo sistem konfigürasyonu

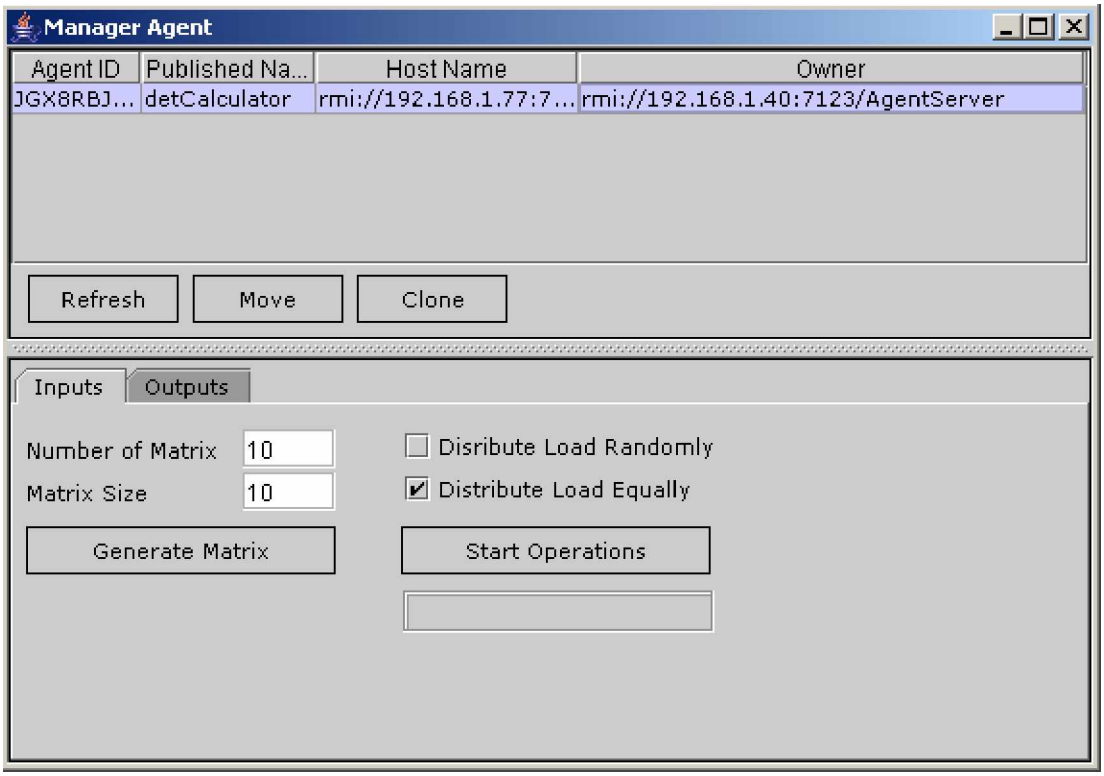
Sistemde tek bir "Güvenlik Yönetici" ve "Gözleyici" düzende çalışan GES sunucu bulunmaktadır (GES1). GES2 ve GES3 sunucuları standart düzen de çalışmakta ve "Güvenlik Yöneticisi" ve "Gözleyici" olarak GES1'i kullanmaktadır. GES2 ve GES3 için gerekli kimlik denetim bilgileri GES1 sunucusunda tanımlanmıştır.

İlk aşamada "detManager" isimli bir etmenin GES1 sunucusu üzerinde ve bir adet "detCalculator" isminde etmenin de GES2 sunucusu üzerinde olduğunu varsayıp etmenlerin çalışmaları gözlemlenmiştir. Herhangi bir GES sunucunun yönetim ekranından sistemdeki etmenleri listelediğimiz de şekil 13.1.1.2 elde edilmektedir.



Şekil 13.1.1.2: Aktif etmenlerin görüntülenmesi

"detManager" etmeni sistemde kendini "detCalculator" isminde duyurmuş olan etmenlerin listesini görüntüleyebilmekte ve onlara iş dağıtabilmektedir. Bu etmenin sağladığı arayüz şekil 13.1.1.3 teki gibidir.

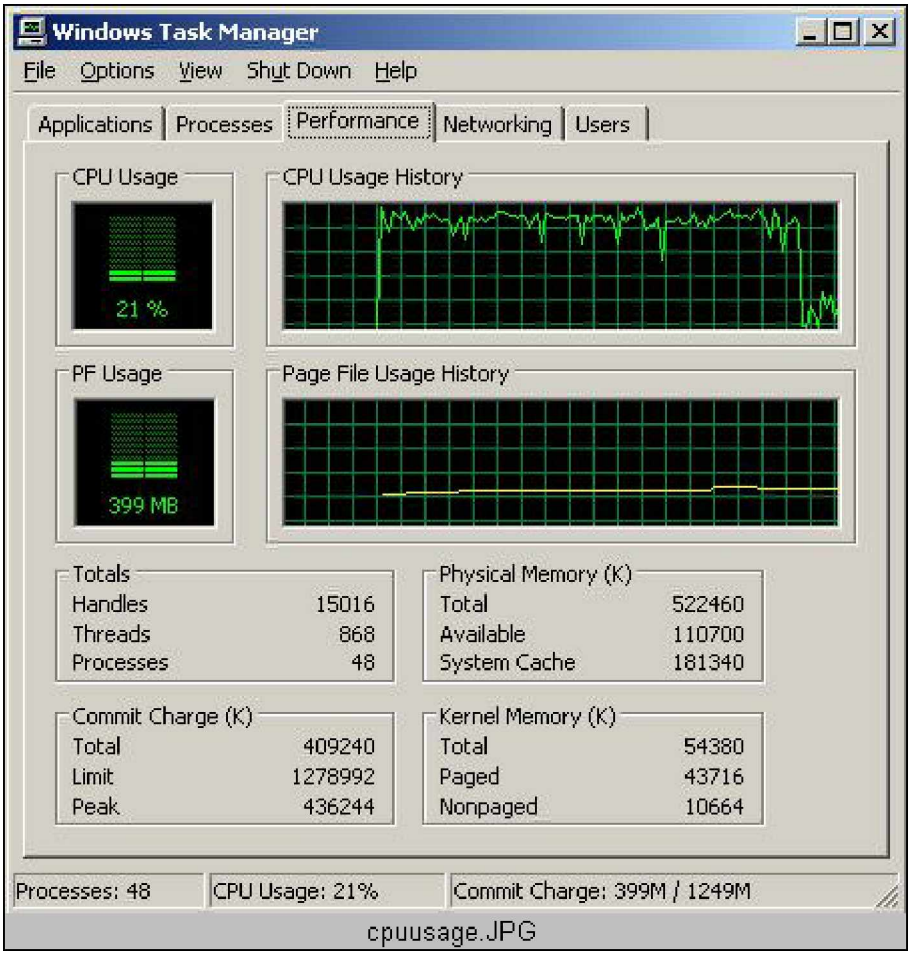


Şekil 13.1.1.3: "detManager" etmeninin arayüzü

"detManager" etmeni 192.168.1.77 IP adresli makine üzerindeki "detCalculator" etmenine "clone" mesajı yollayarak kendini kopyalamasını söyleyebilir. Bu durumda GES2 üzerinde iki adet "detCalculator" etmeni çalışıyor olacaktır.

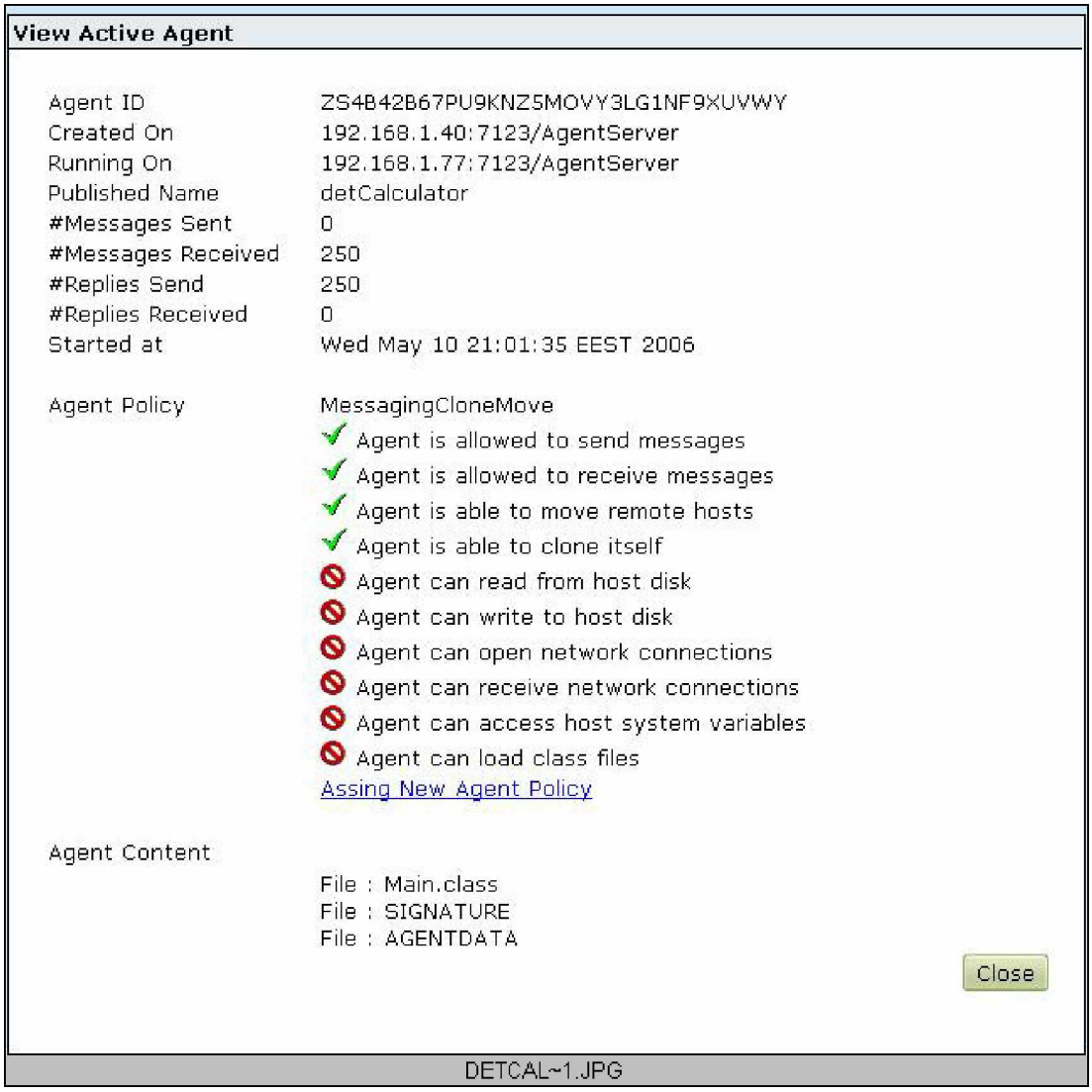
"detManager" etmeni 500 tane 8X8 boyutundaki matrisin çarpımının determinantını bulmak için matrisleri "detCalculator" isimli etmenlere eşit şekilde yollar ve matris determinantlarını hesaplamasını ister. Hesaplayıcı iki etmen de GES2 üzerinde olduğu için bu durumda tüm hesaplama yükünü GES2 sunucusu taşımaktadır.

Hesaplama anında GES2 sunucusunun işlemci kullanımı şekil 13.1.1.4 te gösterilmiştir.



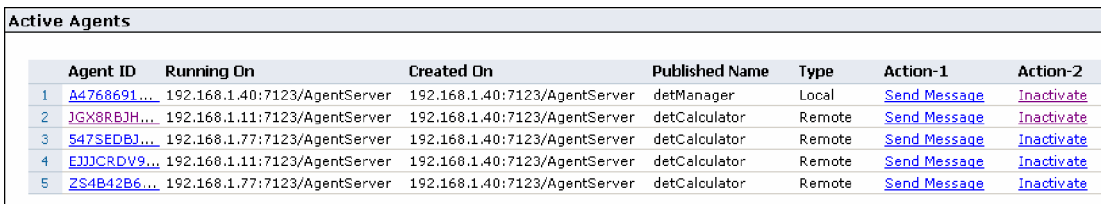
Şekil 13.1.1.4: Çalışma anında GES2 işlemci kullanımı

Tüm matrislerin çarpımının determinant hesabı 80.4 sn sürmüştür. detManager etmeni 500 matrisin 250 tanesini GES2 üzerindeki birinci etmene diğer 250 tanesini ise yine GES2 üzerindeki ikinci hesaplayıcı etmene yollamıştır. Bunun doğruluğunu yine yönetim ekranından herhangi bir etmenin ayrıntılı bilgilerine baktığımızda görebiliriz. Şekil 13.1.1.5 GES2 üzerindeki birinci etmen hakkında ayrıntılı bilginin görüldüğü izleme ekranını göstermektedir.



Şekil 13.1.1.5: "detCalculator" etmenlerinden birinin çalışma anında görüntülenmesi

"detManager" etmeni GES2 üzerindeki detCalculator etmenlerinden birinin GES3 etmeni üzerine göç etmesi için istekte bulunsun. Sonuçta GES2 üzerinde bir tane ve GES3 üzerinde de bir tane hesaplayıcı etmen bulunacaktır. Her iki sunucudaki etmenlere kendilerini kopyalama mesajı gönderdiğinde ise iki sunucuda da ikişer adet hesaplayıcı etmen bulunuyor olacaktır. Son durumda etmen izleme ekranının görüntüsü şekil 13.1.1.6 deki gibi olacaktır.



Şekil 13.1.1.6: Çalışma anında etmenlerin listesi

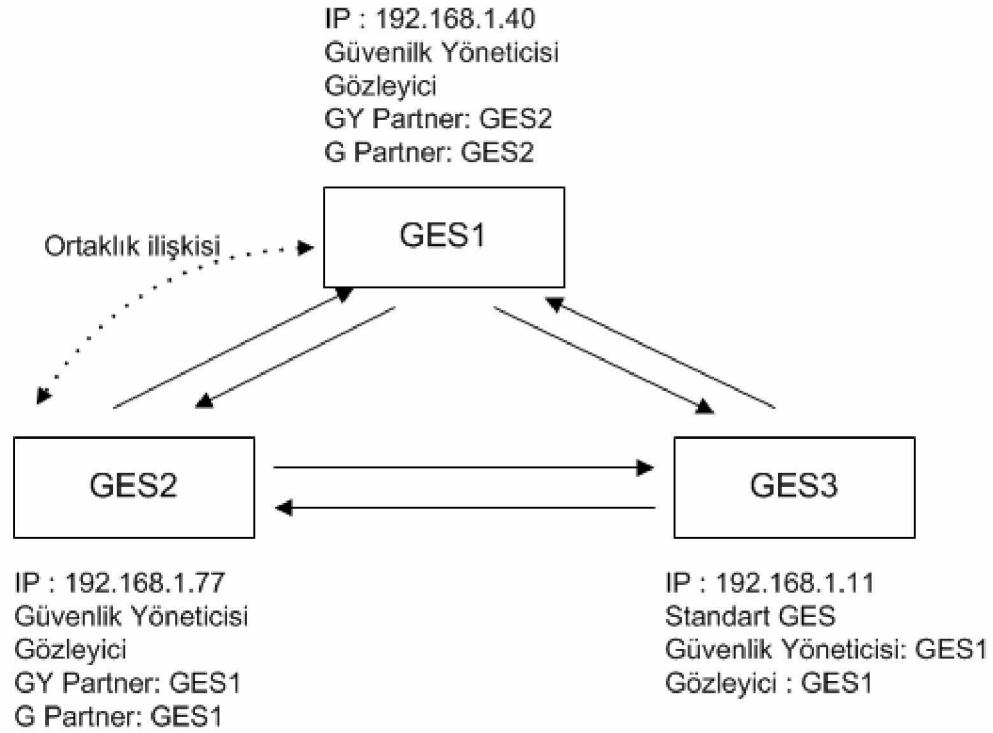
500 adet 8x8 büyüklüğündeki matrislerin çarpımının determinant hesabı bütün etmenler arasında eşit şekilde paylaşacak şekilde bir önceki işlem tekrarlandığın da hesaplama süresinin 107 sn olduğu görüldü. Hesaplayıcı etmenler farklı makinelere dağıldığı halde hesaplama süresinin artmış olmasının nedeni, GES3 sunucusunun üzerinde çalıştığı makinenin performans açısından GES2 sunucusunun çalıştığı makineye göre çok yetersiz olmasıdır. Yük, eşit şekilde dağıtıldığı için GES2 sunucusu isteklere çok hızlı cevap verebilmiş, GES3 ise çok daha yavaş cevap verebilmiştir. Dolayısı ile hesaplama zamanı, yükün büyük kısmını GES2 ye yönlendirmekle azaltılabilir. Bunun yolu da, GES2 üzerinde çalışan etmen sayısını arttırmaktır. GES2 üzerinde 5, GES3 üzerinde 2 etmen olacak şekilde test tekrarlandığında hesaplama süresi 54 sn olarak bulunmuştur. Farklı etmen sayılarına göre yapılan diğer test sonuçları tablo 13.1.1.1 de verilmiştir.

Tablo 13.1.1.1: Hesaplama sürelerinin farklı etmen dağılımına göre değişimi-1

Test	GES1 üzerindeki hesaplayıcı etmen sayısı	GES2 üzerindeki hesaplayıcı etmen sayısı	GES3 üzerindeki hesaplayıcı etmen sayısı	Hesaplama Süresi
1	0	2	0	84 sn
2	0	2	2	107 sn
3	0	5	2	54 sn
4	1	5	2	53 sn
5	3	5	2	62 sn
6	3	8	2	86 sn
7	1	3	0	71 sn
8	1	4	1	68 sn
9	1	1	0	49 sn
10	0	0	2	>120 sn

Hesaplama süresinin en kısa zamanda bitmesi için, yükü etmenler arasında dağıtırken etmenin üzerinde çalıştığı ortamın da dikkate alınması gerekir. Güçlü bir bilgisayar üzerinde çalışan etmene daha çok iş verilebilir, böylece performansından daha çok faydalanılabilir. Ayrıca hesaplama maliyetinin mesajlaşma maliyetini de geçmemesine dikkat edilmelidir. Uzun süren ve işlemci yüküne ihtiyaç duyan işleri uzak etmenler arasında bölüştürmek gerekir.

İkinci adım olarak sistem konfigürasyonunu değiştirip, iki adet “Güvenlik Yöneticisi” ve iki adet “Gözleyici” GES olacak şekilde konfigürasyon yapılıp testler tekrarlanmıştır. Bu konfigürasyonda sistem mimarisi şekil 13.1.1.7 deki gibidir.



Şekil 13.1.1.7: İkinci senaryo sistem konfigürasyonu

GES1 ve GES2 sunucuları hem "Güvenlik Yöneticisi" düzenin de hem de "Gözleyici" düzen de çalışmaktadırlar (Hangi düzende çalışırsa çalışsın, her GES sunucu Standart düzen de çalışmaktadır). GES3 sunucusu ile "Güvenlik Yöneticisi" ve "Gözleyici" GES olarak GES1 sunucusunu kullanmaktadır.

Tekrarlanan bazı testlerin sonuçları tablo 13.1.1.2 de gösterilmiştir..

Tablo 13.1.1.2: Hesaplama sürelerinin farklı etmen dağılımına göre değişimi-2

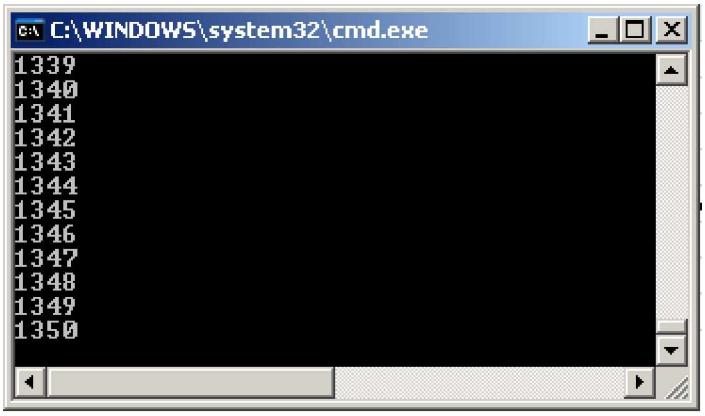
Test	GES1 üzerindeki hesaplayıcı etmen sayısı	GES2 üzerindeki hesaplayıcı etmen sayısı	GES3 üzerindeki hesaplayıcı etmen sayısı	Hesaplama Süresi
1	0	2	0	61 sn
2	0	2	2	94 sn
4	1	5	2	45 sn
5	3	5	2	43 sn

Test sonuçlarının bir önceki sonuçlara göre daha da iyi olduğu görülüyor. Aslında etmen sayıları değişmemesine rağmen daha iyi sonuçlar elde edilmesinin nedeni çok fazla mesajlaşma ve güvenlik kontrolü yapıldığı için "Güvenlik Yöneticisi" olarak çalışan sunucunun yükünün de dağıtılmış olmasıdır.

13.2 Örnek Uygulama-2

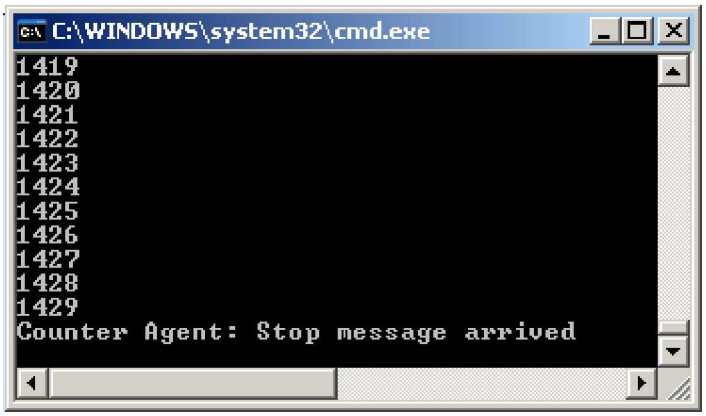
İkinci örnek uygulama, tek başına çalışan bağımsız bir etmenin durumunun göç sırasında nasıl saklandığını göstermek için yazılmıştır. “Counter” isimli bu etmen kendisine gelen “start” isimli mesaj ile her saniye saymaya başlar ve bunu ekrana yazar, “stop” isimli mesaj geldiğinde ise saymayı durdurur. Kendisine gelen “start” mesajı ile yine kaldığı yerden saymaya devam eder. Göç etme mesajlarına da yanıt veren etmen, göç ettiği düğüm üzerinde de saymaya kaldığı yerden devam eder.

Aktif durumda çalışan ve daha önce “start” komutu almış olan “Counter” etmeninin çıktısı şekil 13.2.1 deki gibidir.



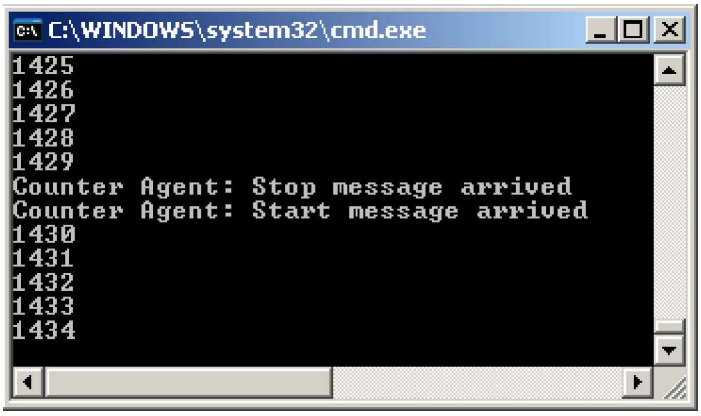
Şekil 13.2.1: “Counter“ etmeninin ekran çıktısı

Etmene “stop” isimli mesaj gelip saymayı durdurduğu anın ekran görüntüsü ise şekil 13.2.2 de verilmiştir.



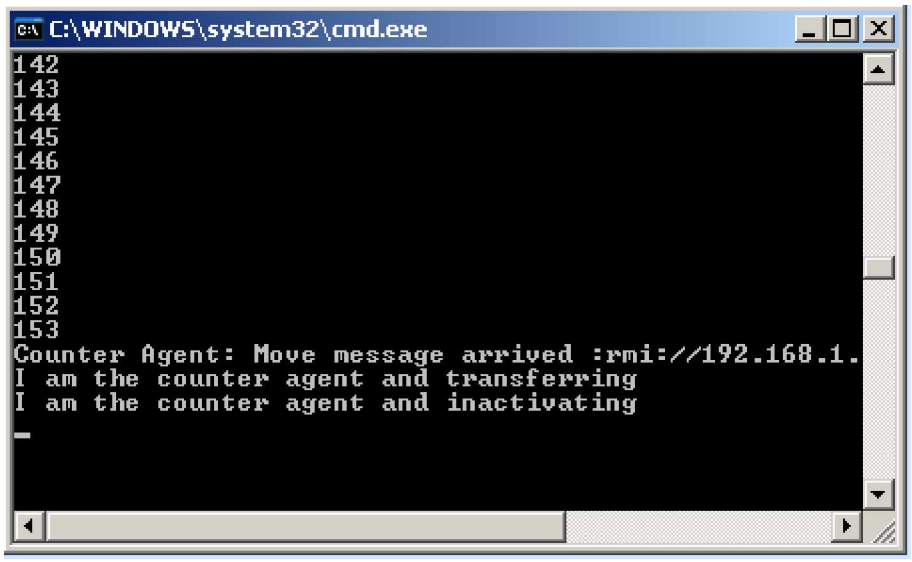
Şekil 13.2.2: “Counter” etmeninin çalışmasını durdurduğu an

Tekrar “start” isimli mesajı aldığında kaldığı yerden çalışmasına şekil 13.2.3 deki gibi devam etmektedir.

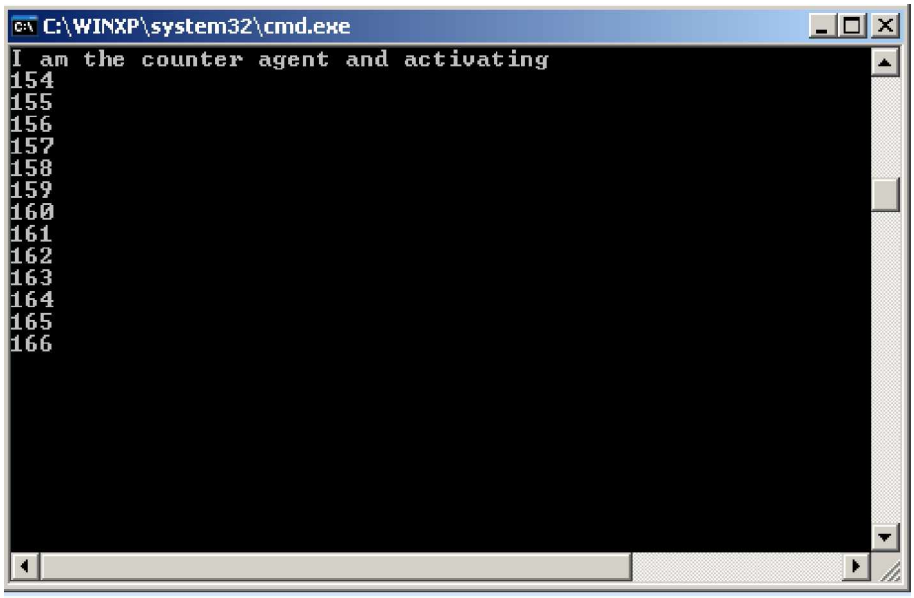


Şekil 13.2.3: "Counter" etmeninin çalışmasına devam etmesi

Etmene diğer bir sunucu üzerine göç etme mesajı geldiğinde iki sunucuda da çıktılar şekil 13.2.4 ve 13.2.5 teki gibi olmuştur. Görüldüğü gibi etmen, GES1 üzerinde çalışırken göç etme isteğinde bulunmuş, GES2 üzerine göç ettikten sonra da kaldığı yerden çalışmasına devam etmiştir.



Şekil 13.2.4: GES1 üzerindeki etmenin göç etme anındaki son durumu

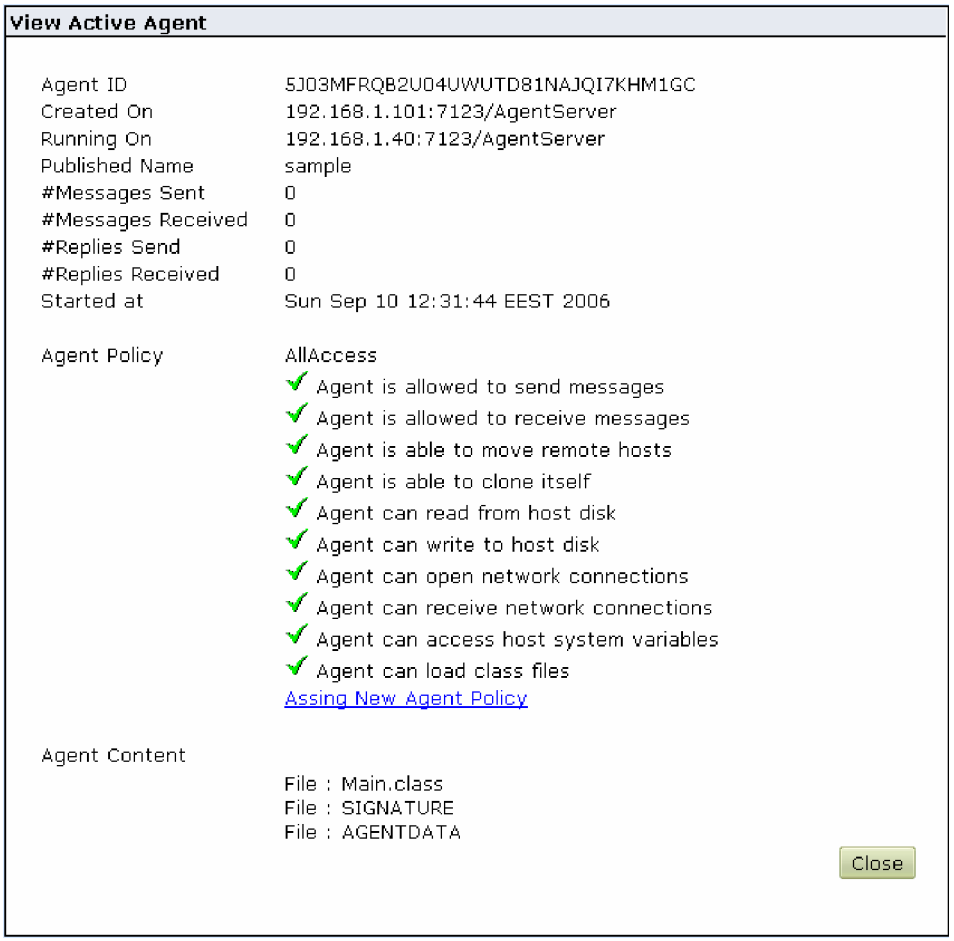


Şekil 13.2.5: GES2 üzerine göç eden etmenin göç sonrası durumu

13.3 Örnek Uygulama-3

Bu uygulamada güvenlik politikalarının kullanımı incelenmiştir. Farklı aktiviteler gerçekleyen bir etmene, farklı yetkilerdeki politikalar atanarak davranışları gözlemlenmiştir. "Sample" isimli örnek etmen göç etme, mesaj alma, mesaj gönderme, diske yazma aktivitelerini yerine getirecek çekilde programlanmıştır. Aktiviteleri gerçeklerken, ekrana bilgilendirici mesajlar vererek gerçeklemeye çalıştığı işlemlerin başarılı olup olmadığı izlenmiştir

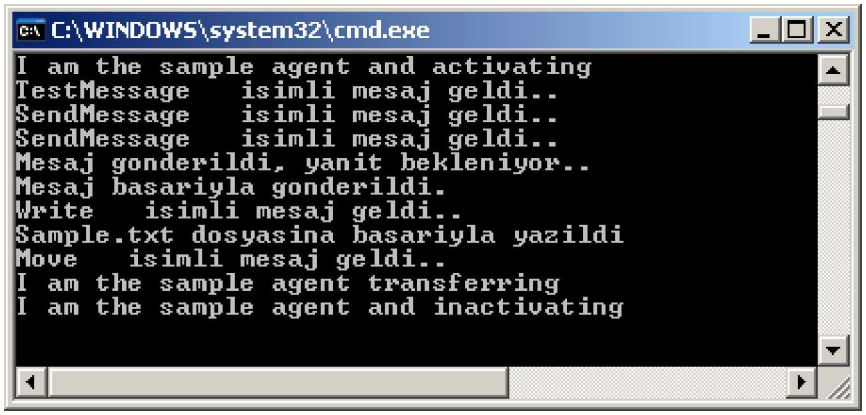
İlk aşamada "sample" isimli etmene (kendini "sample" ismiyle etrafına duyurmuştur) bütün yetkilerin olduğu bir etmen politikası atanmıştır. Çalışma anında etmene ait bilgilerin göründüğü ekran şekil 13.3.1 de gösterilmiştir.



Şekil 13.3.1: "Sample" etmeni çalışma anı bilgileri

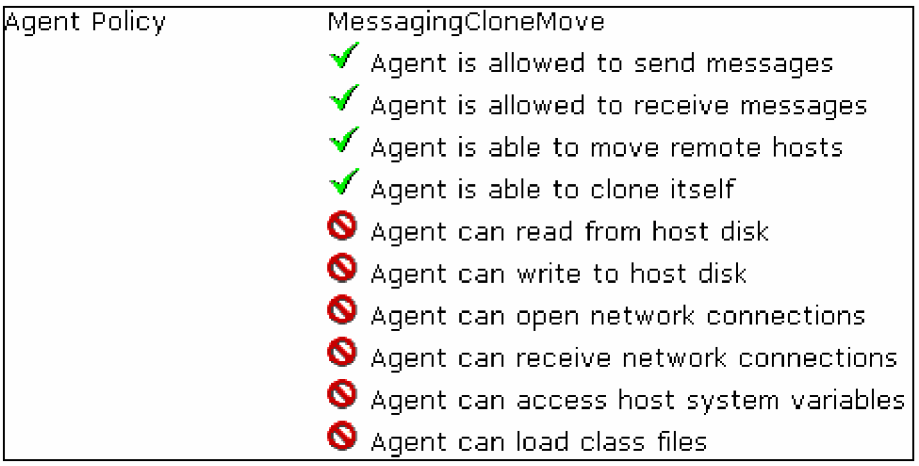
Görüldüğü gibi etmen politikası etmene bütün yetkileri vermektedir. Ancak düğüm üzerindeki politika da etmen üzerinde etkilidir. Bu aşamada düğüm politikası etmenin düğüm üzerinde her türlü aktivitesine izin verecek şekilde ayarlanmıştır.

Etmene mesaj gönderilerek, diske yazma, mesaj gönderme ve göç etme komutları verilmiştir. Gözlenen ekran çıktıları şekil 13.3.2 de gösterilmiştir.



Şekil 13.3.2: "Sample" etmeninin ekran çıktısı

Görüldüğü gibi etmen başarılı şekilde mesaj alabilmekte, mesaj gönderebilmekte, diske yazabilmekte ve göç edebilmektedir. Etmene çalışma anında yeni bir politika atayarak diske yazma hakkı geri alınmıştır. Etmene yeni atanan politika şekil 13.3.3 te gösterilmiştir.



Şekil 13.3.3: "Sample" etmenine atanan yeni etmen politikası

Tekrar diske yazma komutu yollandığında etmenin davranışı şekil 13.3.4 gibi olmuştur.

Şekil 13.3.4: "Sample" etmeninin diske yazım sırasında aldığı hata

Görüldüğü gibi bu sefer etmen diske yazmaya çalışmış ancak etmen politikası izin vermediği için bu işlemde başarılı olamamıştır.

İkinci aşamada etmenin 192.168.1.101 IP adresli düğüm üzerine göç etmesini engelleyecek şekilde düğüm politikası güncellenmiştir. Etmenin o an üzerinde çalıştığı düğüm olan 192.168.1.40 IP adresli düğüm üzerindeki etmen kaynağına göre olan politika şekil 13.3.5 teki gibi değiştirilmiştir. Buna göre düğüm üzerindeki

etmenler 192.168.1.101 IP adresli düğüm üzerindeki etmenlere mesaj gönderebilmektedirler ancak bu IP adresli düğüme göç edememektedirler.

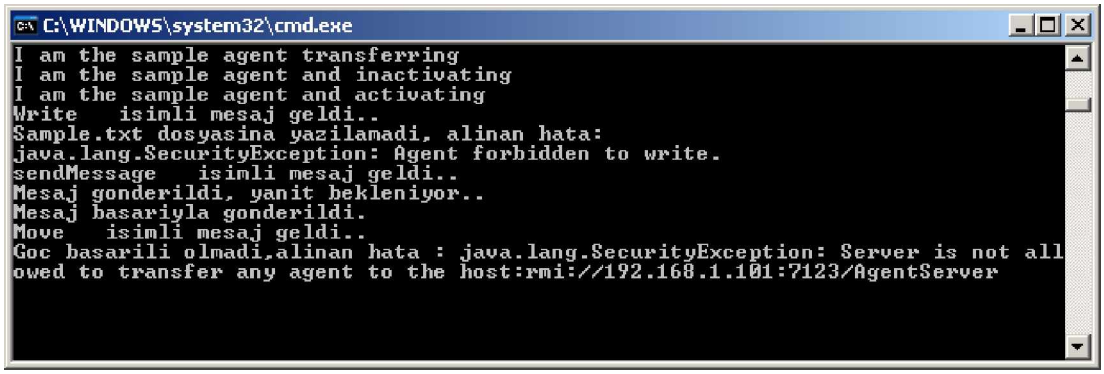
Policies For Agent Sources

Direction	Server	Event	Action	
Outbound	*	Migrate	Permit	Delete
Outbound	rmi://192.168.1.101:7123/AgentServer	Send Message	Permit	Delete
Inbound	*	Migrate	Permit	Delete
Outbound	rmi://192.168.1.101:7123/AgentServer	Migrate	Deny	Delete
Outbound	*	Send Message	Permit	Delete
Inbound	*	Receive Message	Permit	Delete

Direction	Server	Event	Action	
Inbound		Receive Message	Permit	Add Rule

Şekil 13.3.5: Düğüm politikasının etmen kaynağına göre olan kuralları

Etmene 192.168.1.101 IP adresli düğüm üzerindeki diğer bir etmene mesaj yollaması ve sonrada bu düğüm üzerine göç etmesi için gerekli komutlar mesaj gönderilerek tekrar verilmiştir. Ancak şekil 13.3.6 da da görüldüğü gibi politika nedeniyle mesaj gönderebilirken artık göç etme isteği yerine getirilmemektedir.



Şekil 13.3.6: "Sample" etmeni uzak düğüme mesaj gönderebilmekte, ancak göç edememektedir

Görüldüğü gibi çalışma anında pokitikalar değiştirilerek değişen yeni güvenlik ihtiyaçlarına yada çevresel şartlara göre, etmenleri yeniden programlamaya gerek kalmadan yeni düzenlemeler çok kısa ve kolay bir şekilde yapılabilmektedir.

13.4 Örnek Uygulama-4

Bu örnek uyulamada GES'in aktif bir atağa karşı nasıl etkili olduğu gösterilecektir. GES, etmenlerin kod, durum ya da politikalarında olabilecek yetkisiz değişiklikleri sezebilmekte ve böyle bir durum karşısında etmeni aktif duruma geçirmemektedir.

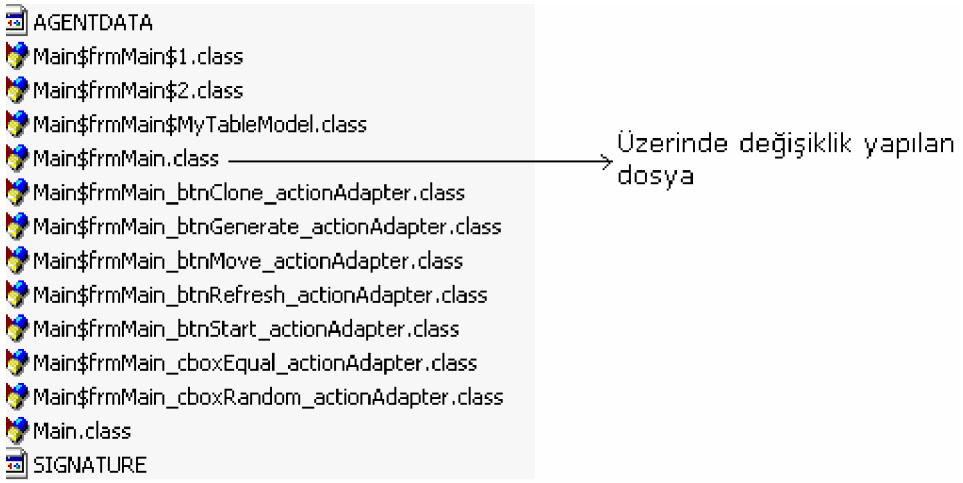
Örnek uygulama-1 de verilen “detManager” isimli etmen örnek etmen olarak ele alınmıştır. Bu etmene ait kod, durum ve politika dosyası disk üzerinde şu şekilde durmaktadır:

gjpjgylr.zip : Etmen kodunu tutan dosyadır. Etmene ait tüm class dosyaları bu sıkıştırılmış dosya içindedir.

gjpjgylr.dat: Etmen güncel durumunu tutan dosyadır.

gjpjgylr.pol: Etmen politikasını tutan dosyadır.

Kod dosyalarını tutan sıkıştırılmış dosya içindeki dosyalar geçici olarak diske açılmış sonra herhangi birinde çok ufak bir değişiklik yapılarak dosya yeniden oluşturulmuştur. Sıkıştırılmış kod dosyasının içeriği şekil 13.4.1 de gösterilmiştir.



Şekil 13.4.1: Class dosyalarından biri üzerinde oynama yapılan etmenin kod dosyası

Bu durumda iken etmen yeniden aktif duruma getirildiğinde aktif olamadığı görülmüş, log dosyasında da aşağıdaki izin oluştuğu gözlenmiştir.

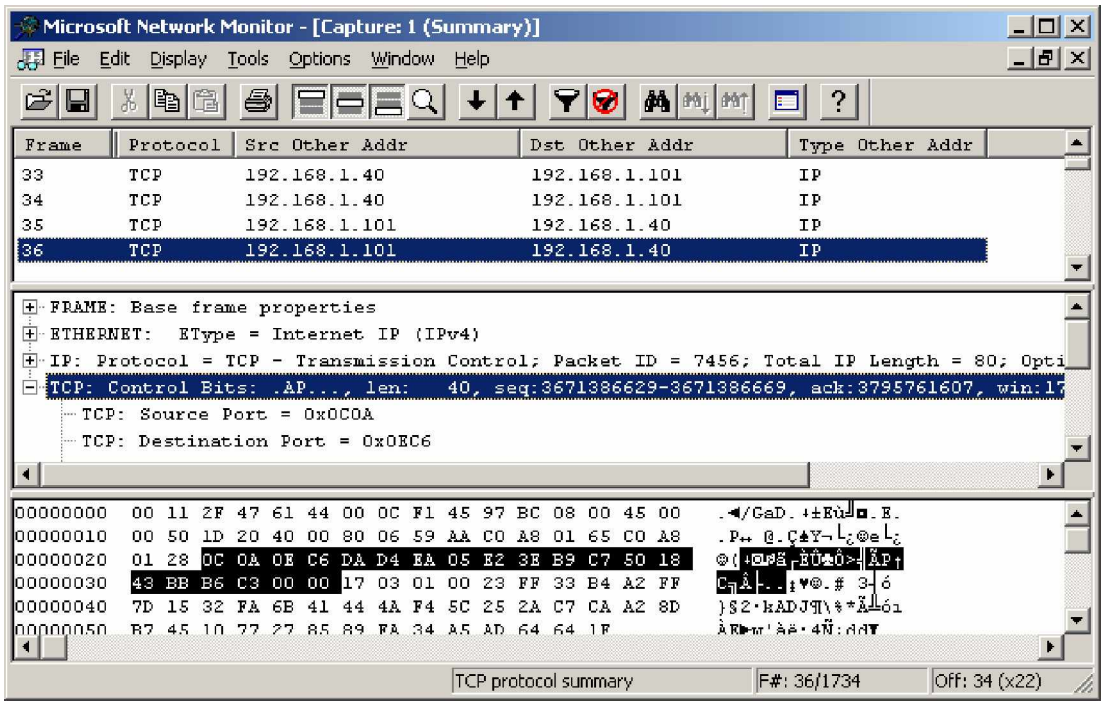
```
Sun Sep 10 15:01:39 EEST 2006 The agent source : gjpjgylr.zip is tampered.
```

GES, etmeni aktif etmeden önce etmene ait olan şifreli class dosyalarını açar, her birine ait “hash” değerlerinden tek bir imza değeri üreterek kod dosyası içindeki “SIGNATURE” dosyası ile karşılaştırır. Yukarıdaki müdahele sonunda bu işlemler aynı sonucu vermediği için etmen koduna yetkisiz müdahele edildiği anlaşılır. “SIGNATURE” dosyasının içeriği de değiştirilerek dosya kendi başına uyumlu duruma getirilebilir ancak imza değeri sadece kod dosyası içinde değil, durum ve politika dosyası içinde de tutulur böylece bu üç dosyanın birbirleri ile uyumlu olması sağlanır. Yani, durum yada politika dosyasındaki en ufak değişiklikte farkedilebilir.

13.5 Örnek Uygulama-5

Bu uygulamada ağ ortamını dinleyen aktif bir yazılımın GES ve etmenler arası iletişimi gözleyerek bilgi edinmesinin nasıl engellendiği gösterilecektir.

GES sunucular ve etmenler arası bütün iletişimler, sertifika tabanlıdır ve SSL protokolü ile gerçeklenir. Dolayısı ile ağı dinleyen herhangi bir birimin anlamlı veriler gözlemesi beklenmez. Bunu doğrulamak için ağı dinleyen bir "sniffer" yazılımı kullanılmış ve IP adresleri 192.168.1.101 ve 192.168.1.40 olan GES sunucular üzerindeki etmenlerin haberleşmesi sırasında iletilen paketler yakalanarak analiz edilmeye çalışılmıştır. Dinleme sonucu yakalanan bazı paketler şekil 13.5.1 deki gibidir.



Şekil 13.5.1: GES sunucular ve etmenler arası haberleşme trafiği

Yakalanan paketlerin içeriklerine bakıldığında, anlamlı hiç bir bilgi olmadığı görülmektedir. Bunun nedeni tüm iletimlerin SSL protokolü ile şifreli olarak gerçekleniyor olmasıdır. Bu nedenle ağı dinleyip etmenler hakkında bilgi toplamaya çalışan bütün atak çeşitleri başarısız olacaktır.

13.6 Örnek Uygulama-6

Bu uygulamada GES mesajlaşma alt yapısının güvenlik özellikleri test edilmiştir. Birbirleri ile haberleşen iki etmenden biri kötü niyetli bir etmen olarak programlanmıştır. Etmenler arası haberleşme şu şekildedir.

- Birinci etmen, ikinci etmene (kötü niyetli olan) herhangi bir mesaj yollar.
- İkinci etmen mesajı alır ve aldığını bildiren bir yanıt döner.
- Birinci etmen mesajın yanıtı alır.

Bu aşamaya kadar olan iletişim normaldir. Ancak ikinci etmen daha önce birinci etmenden aldığı mesajı saklar ve rastgele zamanlarda bu mesajı referans vererek yeni yanıtlar döner. Bu yanıtların birinci etmen için bir anlamı yoktur ancak kötü niyetli etmen birinci etmeni yanıltmak ya da çalışmasını engelleyecek bir yol aramaktadır.

GES mesajlaşma alt yapısı henüz tamamlanmamış bir mesaja ait geçmiş bilgisini saklar. Bu iletişime ait bütün paketler başarıyla iletildiğin de mesaja ait durum bilgilerini de siler. Dolayısı ile yukarıdaki gibi bir durumda birinci etmenin üzerinde çalıştığı GES kendisine yeni bir yanıt geldiğinde, bu yanıtın daha önceden gönderilmiş ve henüz iletişiminin bütün aşamaları tamamlanmamış bir mesajlaşmaya ait olup olmadığı kontrol eder ve kötü niyetli etmenin sonradan gönderdiği mesaj yanıtlarını dikkate almadan ve birinci etmene iletmeden doğrudan siler.

Birinci etmene ait kod parçası aşağıda verilmiştir.

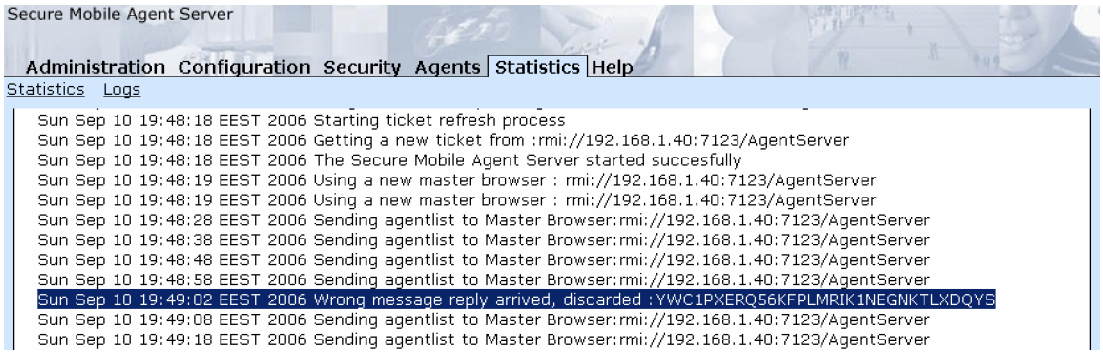
```
1-     public class Main extends Agent {
2-     public void OnMessageArrive() {
3-      MessagePacket packet = getAgentInterface().receive();
4-     Message msg = packet.getMessage();
5-     if (msg.getMessageName().equalsIgnoreCase("SendMessage")) {
6-     Enumeration agents = getAgentInterface().getAgentPointer("receiver");
7-     AgentPointer target = null;
8-     while (agents.hasMoreElements()) {
9-     target = (AgentPointer) agents.nextElement();
10-    }
11-    Message newmsg = new Message("take");
12-    MessageID msgid = getAgentInterface().sendMessage(target, newmsg);
13-    System.out.println("Mesaj gonderildi, yanit bekleniyor..");
14-    if (getAgentInterface().waitForReply(msgid, 2000)) {
15-    String reply = (String) getAgentInterface().getReply(msgid);
16-    System.out.println("Reply alindi :" + reply);
17-    } else  System.out.println("Reply alinamadi...");
18-    }
19-    }
20-    ........
```

Etmen kendisine gelen yeni mesajı üçüncü ve dördüncü satırlarda almakta, mesaj ismi “sendMessage” olması durumunda, altıncı satırda ismi “receiver” olan etmenleri bulmaktadır. Onbirinci satırda yeni bir mesaj oluşturmakta ve “receiver” isimli herhangi bir etmene bu mesajı onikinci satırda yollamaktadır. İki saniye süreyle yanıtı bekleyip gelen yanıtı onaltıncı satırda ekrana yazmaktadır.

İkinci etmene ait kod parçası da aşağıdaki gibidir.

```
1-     public class Main extends Agent {
2-     MessagePacket attackedPacket = null;

3-     public void OnMessageArrive() {
4-     MessagePacket packet = getAgentInterface().receive();
5-     Message msg = packet.getMessage();
6-     if (msg.getMessageName().equalsIgnoreCase("take")) {
7-     attackedPacket = packet;
8-     getAgentInterface().sendReply(packet, "Received");
9-     return;
10-    }

11-    if (msg.getMessageName().equalsIgnoreCase("attack")) {
12-    getAgentInterface().sendReply(attackedPacket, "Received");
13-    }
14-    }
15-    .......
```

Birinci etmenin gönderdiği “take” isimli mesajı kötü niyetli olan bu etmen almakta ve “attackedPacket” değişkeninde yedinci satırda görüldüğü gibi saklamaktadır. Sekizinci satırda mesajın yanıtı göndermektedir. Daha sonra herhangi bir anda (bu örnekte kontrollü olması nedeniyle mesaj gönderilerek atağa başlaması sağlanmıştır) Onikinci satırda görüldüğü gibi sakladığı pakete ikinci bir yanıt dönmektedir. Birinci etmenin üzerinde çalıştığı GES sunucu bu yanıt paketinin geçersiz bir yanıt paketi olduğunu anlar ve doğrudan siler. Bu aktiviteye ilişkin izler, sistem izlerinin görüntülendiği yönetim ekranından da şekil 13.6.1 de olduğu gibi görülebilmektedir (ilgili iz koyu renkle işaretlenmiştir).



Şekil 13.6.1: Uygunsuz yanıt mesajının GES sunucu tarafından silinmesi

14. SONUÇLAR VE TARTIŞMA

Hareketli etmen sistemlerinin uygulamada yaygın kullanım alanı bulabilmesi için güvenlik tehditlerinin giderilmiş olması gerekmektedir. Güvenli Etmen Sistemi (GES), hem etmenlerin güvenlik gereksinimlerine yanıt vermek, hem de kolay kullanımlı ve esnek bir çalışma ortamı sunmak üzere tasarlanmış ve gerçeklenmiştir.

GES'in sağladığı güvenlik çözümleri tasarım aşamasında planlanıp, mimari altyapıya entegre edildikleri için sistemle bir bütün oluştururlar. GES, bu yönüyle mevcut hareketli etmen sistemlerine göre bir adım öndedir. Ayrıca düğümleri ve etmenleri korumaya yönelik, politika kullanımına dayalı dinamik mekanizmalar, etmen programcısı ve düğüm yöneticisine ek yük getirmeden, esnek bir çalışma ortamı sunmakta ve kendini kolaylıkla değişikliklere uyarlayabilen etmen uygulamalarının geliştirilmesini mümkün kılmaktadır.

GES, etmen programcısına çok esnek bir etmen geliştirme arayüzü sunar. Sistemin sunduğu etmen modeli, durum değişiklerinde etmen davranışlarını belirleyecek kodun geliştirme sürecini kolaylaştırarak, karmaşık algoritmalara olan gereksinimi ortadan kaldırır. Ayrıca, etmen programcısı ve düğüm yöneticisi, etmenleri izleme, yönetme, düğümleri ve GES modüllerini yönetme gibi önemli yönetim işlevlerini bir tarayıcı aracılığı ile rahatça uygulayabilirler.

GES mimarisini incelendiğinde, her biri ayrı önem derecesinde ve zorlukta bir kaç ana başlık karşımıza çıkmaktadır. Bunların ilki etmenlerin modellenmesidir ve etmenlerin sistem içinde nasıl tanımlandığı, etmen kod, durum ve politika bilgisinin nasıl saklanıp yönetildiği ve etmen aktivitelerinin nasıl gerçekleştirildiği gibi kritik konuları içerir. İkinci önemli başlık ise, etmen haberleşme alt yapısının tasarımıdır. Bu başlık, etmenlere sunulan konumdan bağımsız ve saydam haberleşme olanağının nasıl gerçekleneceği, etmenlerin farklı türden haberleşme isteklerini karşılayan fonksiyonların tasarımı, etmenlerin konum bilgilerinin sürekli nasıl izleneceği ve nerede saklanacağı gibi konuları kapsamaktadır. Üçüncü önemli başlık güvenlik politikalarının kullanımıdır. Etmen ve düğüm politikalarını nasıl tanımlanacağı, saklanacağı, taşınacağı, işleneceği ve güncelleneceği gibi konular bu başlık altında tasarlanmış ve gerçeklenmiştir. Dördüncü önemli başlık yönetim ve süreklilik konusudur. Bu başlık altında, etmen ve GES modüllerinin nasıl izleneceği ve

yönetileceği, sistemin sürekliliğini sağlayan mekanizmaların neler olduğu gibi konular yer verilmiştir. Bu ana başlıkların odak noktasında, tüm bu işlevlerin güvenli bir şekilde nasıl gerçekleneceği konusu bulunmaktadır. Tasarımın her aşamasında, güvenlik açığına neden olabilecek her türlü konu ele alınmaya çalışılmıştır. GES mimarisindeki her biri kendi içinde karmaşık algoritmalar içeren bu ana ve alt konular ulusal ve uluslar arası bilimsel toplantılarda bildiri olarak yayınlanmıştır. Etmen gizli verisinin saklanması[43], haberleşme alt yapısı[47], güvenlik politikalarının kullanımı[48] uluslararası toplantılarda, mimarinin geneli ise hem ulusal hem de uluslar arası toplantılarda incelenmiş ve sunulmuştur[41-42,44-46].

GES mimarisi geliştirmeye açık bir sistemdir. Katmanlı yapısı, her bir katmanda ekleme ve değişikliğin bağımsız bir şekilde yapılabilmesine olanak sağlamaktadır. Örneğin, etmenlerin kullanıcılar ile ilişkilendirilerek kullanıcı bazlı bir yetkilendirme mekanizmasının kurulması, hareket halindeki etmenlere gönderilen mesajların saklanarak etmen göçü sonunda etmenlere ulaştırılması, etmenlere rol desteğinin kazandırılması, etmen aktivite izlerinden anormal etmen davranışlarının sezilebilmesi, sahte JVM lerin sezilebilmesi, güvenlik politikalarının daha fazla kontrol içerecek şekilde genişletilmesi gibi, her biri bir tez konusu olabilecek önemli konuların yeni çalışmalar sonucu mimariye eklenmesi mümkündür.

GES sistemi ve etmenler JAVA dili ile yazıldıklarından JAVA dilindeki herhangi bir güvenlik açığı, mimaride de bir açığa neden olabilir. Ancak JAVA dili gün geçtikçe güvenlik adına önemli özellikler kazanmaktadır. Etmen geliştiren programcının uygulama seviyesindeki güvenlik ihtiyaçlarını dikkate alması beklenmektedir; GES'in yanlış tasarlanmış bir etmenin davranışlarını kontrol etmesi şu aşamada mümkün değildir. Örneğin, bir etmenin işbirliği yapmadan önce, ortama aynı isimle kendini duyurmuş olan hedef etmenler için kimlik kontrolü yapması, uygulama düzeyinde bir güvenlik önlemidir. Bu adım, ortak bir anahtarın paylaşılması veya sorulan bir soruya beklenen yanıtın alınması şeklinde gerçekleştirilebilir.

Sadece GES tarafından değil, diğer hareketli etmen sistemleri tarafından da engellenemeyecek olan bazı atak türleri bulunmaktadır. Örneğin, bir etmenin sonsuz döngüye girerek, düğüm üzerinde aşırı işlemci yükü oluşturmasını engellemek için belirgin bir yöntem bulunmamaktadır. Etmenlerin her biri ayrı bir iplik olarak çalıştıkları için, iplik önceliğini değiştirmek bu türden atakları engellemez ancak düğüme ait kaynakların daha iyi kullanımını sağlayabilir. Mimariye eklenebilecek diğer bir özellik güvenlik politikalarına etmen öncelik özelliğinin de eklenmesi olabilir.

GES, sahip olduğu önemli güvenlik özellikleri yanısıra, etmen programcısı ve sistem yöneticisine sağladığı esnek çalışma ortamı nedeniyle de kullanılmaya değer yeni ve açık bir hareketli etmen sistemidir.

KAYNAKLAR

[1] **Sander, T. and Tschudin, C.,** 1998. Protecting mobile agents against malicious hosts, *Mobile Agent Security*, *LNCS*, **1419**, 44-60.

[2] **Borselius, N.,** 2002. Mobile agent security, *Electronics & Communication Engineering Journal*, London, UK, Volume 14, **5**, 211.

[3] **Suresh Raj, G.,** 1998. A detailed comparison of CORBA, DCOM and Java/RMI, *http://my.execpc.com/~gopalan/misc/ compare.html*

[4] **Franklin, S. and Graesser, A.,** 1996. 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.

[5] **Peter Domel, B.L.,Oshima, M.,** 1996. Mobile Telescript agents and the web, *COMPCON Spring '96 - 41st IEEE International Computer Conference*, 52.

[6] **Johansen, D., Fred, B.S. and Van Renesse, R.,** 1998. What TACOMA taught us**,** *Mobility, Mobile Agents and Process Migration - An edited Collection*, Addison Wesley Publishing Company.

[7] **Robert S.G.**, 1996. Agent Tcl: A flexible and secure mobile-agent system, *Proceedings of the 1996 Tcl/Tk Workshop*, 9-23.

[8] **Danny B. L. and Oshima, M.,** 1998. Programming and deploying Java(TM) mobile agents with Aglets(TM), Addison-Wesley Professional.

[9] **Wheeler, T.,** 2002. Reducing development effort using the Voyager ORB, *http://www.recursionsw.com/Voyager/Ease_of_Development.pdf*, Recursion Software.

[10] **Mitsubishi Electric ITA**, 1997. Concordia: An infrastructure for collaborating mobile agents, Horizon Systems Laboratory, USA.

[11] **Anand, R. T., Neeran, M.K., Manish, K.V, Tanvir, A. and Ram D.S.**, 1999. Mobile agent programming in Ajanta, *19th IEEE International Conference on Distributed Computing Systems (ICDCS'99)*, 190.

[12] **Varadharan, V. and Foster, D.,** 2003. A security architecture for mobile agent based applications, *World Wide Web:Internet and Web Information System*, **6**, 93-122.

[13] **Makino, S., Okoshi, T., Nakazawa, J. and Tokuda, H.,** 2000. s-agent:The design of secure mobile agent system, Keio University, Japan.

[14] **Bryce, C. and Vitek, J.**, 2001. The JavaSeal mobile agent kernel, *Autonomous Agents and Multi-Agent Systems*, **4**, 359-384.

[15] **Jeon, H., Patrie C. And Cutkosky Mark, R.,** 2000. JATLite: A Java agent infrastructure with message routing, *IEEE Internet Computing*, Volume 4, **2**, 87-96

[16] **Jansen, W. and Karygiannis, T.,** 2000. Mobile agent security, *NIST Special Publication* 800-19- *National Institute of Standarts and Technology.*

[17] **Michael, S. G., Jennifer, C.B. and David, G.H,** July 1998, Mobile agents and security, *IEEE Communications Magazine*, **36**, 76-85

[18] **Bian, G., Nakayama, K., Kobayashi, Y. and Maekawa, M.,** 2004, Java mobile code security by bytecode behavior analysis, *Software Engineering and Applications,* **436**, 119.

[19] **Schneider, F.B.,** 1997. Towards fault-tolerant and security agentry, *Proc.11th Int. Wksp. Dist. Algorithims*, Germany, 1-14.

[20] **Mira da Silva, M. and Rodrigues da Silva, A.,** 1997. Insisting on persistent mobile agent systems, *First International Workshop on Mobile Agents, LNCS*, **1219**, 174-185.

[21] **Young, A. and Yung, M.**, 1997. Sliding encryption, a cryptographic tool for mobile agents, *Proc. 4th Int. Wksp. Fast Software Encryption*, Haifa, Israel, 240.

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

[23] **Sander T.**, 1997. On cryptographic protection of mobile agents, *Proc. 1997 Wksp. Mobile Agents and Security,* Univ. of Maryland

[24] **Farmer, W., Guttmann, J. and Swarup, V.,** 1996. Security for mobile agents: Authentication and state appraisal, *In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, Proc. of ESORICS 96, LNCS,* **1146**, 118-130.

[25] **Harold, E.R.,** 2006. Java I/O, 2 Edition, O'Reilly Media, CA.

[26] **Grosso, W.,** 2001. Java RMI, 1 Edition, O'Reilly Media, CA.

[27] **Forman, I.R. and Forman, N.,** 2004. Java reflection in action, Manning Publications, NY.

[28] **Karnik, N.M. and Tripathi, A.R**., 1998. Design issues in mobile-agent programming systems, *IEEE Concurrency*, **6 (3)**, 52–61.

[29] **Gallery, E.,** 2003. Towards a policy based framework for mobile agent authorisation in mobile systems, *Mobile VCE Research Group*, Royal Holloway, University of London.

[30] **Montanari, R., Lupu, E. and Stefanelli, C.**, 2004. Policy-based dynamic reconfiguration of mobile-code applications, *Computer Volume 37*, **7**, 73.

[31] **Dias, P., Riberio, C. and Ferreira P.**, 2003. Enforcing history-based security policies in mobile agent systems, *Fourth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'03),* 231.

[32] **Hauswirth, M., Kerer, C. and Kurmanowytsch, R.,** 2000. A flexible and extensible security framework for Java code, *Technical Report,* **TUV-1841-99-14**, Technical Univ. of Vienna.

[33] **Wilhelm, U.G. and Staaman, S.,** 1998. Protecting the itinerary of mobile agents, *Laboratoire de Systemes d'Exploitation*, Switzerland.

[34] **Necula, G.C. and Lee, P.,** 1998. Safe, untrusted agents using proofcarrying Code, *Mobile Agents and Security, LNCS*, **1419**, 61-91.

[35] **Weinman, W.E.,** 1996. The CGI Book, New Riders Pub

[36] **Weissinger, A.K.,** 2000, ASP in a Nutshell, 2nd Edition, O'Reilly Media, CA

[37] **Sklar, D.,** 2004, Learning PHP 5,1st Edition, O'Reilly Media, CA

[38] **Hunter, J.,** 2001, Java servlet programming, 2nd Edition, O'Reilly Media, CA

[39] **Williams, P. and Feghhi, J., 1998**, Digital certificates, Wesley Longman Publishing Co, Boston, USA

[40] **Stinson, D.R., 1995**, Cryptography theory and practice, CRC Press, Inc

[41] **Uğurlu, S. ve Erdoğan, N.,** 2004. Hareketli bir güvenli etmen sistemi, *Havacılıkta İleri Teknolojiler ve Uygulamaları Sempozyumu*, İstanbul, Türkiye, 633-637.

[42] **Uğurlu, S. ve Erdoğan, N.**, 2004. GES: Güvenli etmen sistemi, *ELECO 2004 – Elektrik, Elektronik ve Bilgisayar Mühendisliği Sempozyumu*, Bursa,Türkiye.

[43] **Uğurlu, S. ve Erdoğan, N.**, 2005. An approach to protect mobile agent privacy, *Proc. BİLTEK 2005 – International Informatics Congress*, Eskisehir-Turkey, 120-128.

[44] **Uğurlu, S. ve Erdoğan, N.**, 2005. Hareketli etmenler için güvenli bir çalışma ortamı, ABG 2005 – Ağ ve Bilgi Güvenliği Ulusal Sempozyumu, İstanbul, Turkiye, 9-16.

[45] **Uğurlu, S. ve Erdoğan, N.**, 2005. An overview of SECMAP secure mobile agent platform, *Proc. AAMAS 2005- the 4th International Conference on Autonomous Agents & Multi Agent Systems/SASEMAS 05-Workshop:Safety and Security in MAS*, Univ. Utrecht, The Netherlands, 122-126.

[46] **Uğurlu, S. ve Erdoğan, N.**, 2005. SECMAP: A secure mobile agent platform, *4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS) ,LNCS Vol.3690*, Budapest, Hungary, 102-111.

[47] **Uğurlu, S., Erdoğan, N.,** 2005. A Secure Communication Framework for Mobile Agents, *The 20th International Symposium on Computer and Information Sciences (ISCIS 2005), LNCS Vol. 3733*, Istanbul-Turkey, 412-421.

[48] **Uğurlu, S., Erdoğan, N.,** 2006. A flexible policy architecture for mobile agents, *32nd International Conference on Current Trends in Theory and Practice of Computer Science(SOFSEM 2006), LNCS Vol. 3831*, Merin, Czech Republic, 538-547.

ÖZGEÇMİŞ

Suat Uğurlu 1974 yılında Bayburt'da doğmuştur. 1997 yılında İstanbul Teknik Üniversitesi Kontrol ve Bilgisayar Mühendisliği Bölümü'nden lisans derecesini almıştır. 2000 yılında İstanbul Teknik Üniversitesi Kontrol ve Bilgisayar Mühendisliği Bölümü'nde yüksek lisans çalışmasını tamamlamıştır.