T. C.
İSTANBUL TEKNİK ÜNİVERSİTESİ
Bilişim Enstitüsü

AĞ GÜVENLİĞİ
Prof. Dr. Bülent Örencik

"Model Güdümlü Güvenlik" Dönem Projesi Raporu

Çağdaş CİRİT
704061005

Nisan, 2007

İÇİNDEKİLER

1. **Giriş…… 3**

2. **Altyapı…… 5**
   2.1. Bir Tasarım Problemi…….. 5
   2.2. Unified Modelling Language (UML) ……...5
   2.3. Model Güdümlü Mimari (MGM) ……6
   2.4. Rol Tabanlı Erişim Denetimi (RTED) ……..6

3. **Model Güdümlü Güvenlik : Tanıtım…… 7**
   3.1. Güvenlik Modelleme Dilleri……. 7
   3.2. Sistem Tasarım Dilleri ve Diyalektler……... 8
   3.3. Model Dönüşümü…….. 8

4. **GüvenliUML…… 9**
   4.1. Soyut Sentaks……. 9
   4.2. Somut Sentaks……10
   4.3. Semantik……11

5. **Örnek Bir Modelleme Dili: BileşenUML……...12**
   5.1. Soyut Sentaksı Genişletme……12
   5.2. Somut Sentaksı Genişletme……14
   5.3. Yetki Politikasını Modelleme……15

6. **Bir Güvenlik Altyapısı Sistemini Üretme: EJB……16**
   6.1. EJB için temel üretim kuralları…….. 16
   6.2. EJB erişim denetim altyapısını üretme…….. 16

7. **Değerlendirme ve Sonuç…… 17**

8. **Kaynaklar…… 18**

ŞEKİLLER

1. Şekil 1: Model Güdümlü Mimari…… 3
2. Şekil 2: Model Güdümlü Güvenlik…… 4
3. Şekil 3: Toplantı Düzenleyici uygulamasının Sınıf Diyagramı…… 5
4. Şekil 4: Güvenlik Tasarım Dili Şeması…… 7
5. Şekil 5: GüvenliUML metamodeli…… 9
6. Şekil 6: Güvenli UML'in somut sentaksına bir örnek…… 10
7. Şekil 7: BileşenUML metamodeli…… 12
8. Şekil 8: Toplantı Çizelgesi Uygulaması…… 12
9. Şekil 9: BileşenUML metamodeli için GüvenliUML diyalekti…… 13
10. Şekil 10: Yetki Politikaları eklenmiş Toplantı Çizelgesi Uygulaması…… 15

TABLOLAR

1. Tablo 1: GüvenliUML soyut ve somut sentaks eşlemesi……...10
2. Tablo 2: GüvenliUML Diyalekt İşlem Hiyerarşisi……14
3. Tablo 3: BileşenUML için işlem referans tipleri……14
4. Tablo 4: Atomik işlem EJB metot eşleşmesi……...…..17

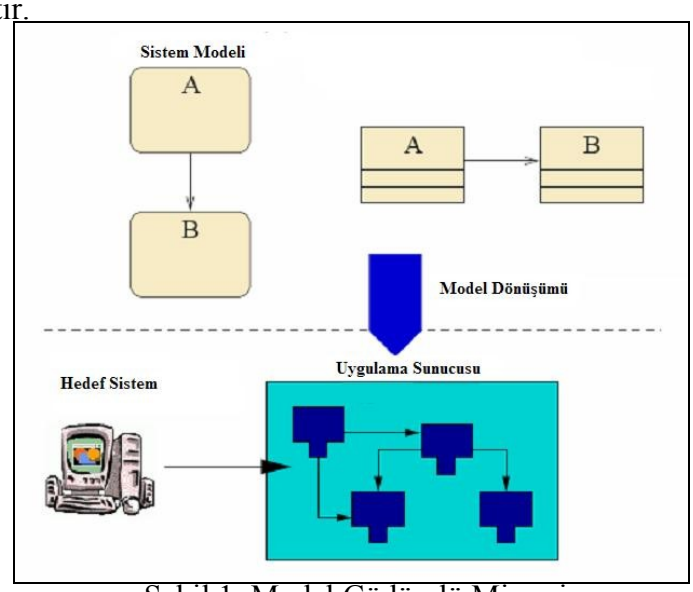
KISALTMALAR

MGG: Model Güdümlü Güvenlik
MGM: Model Güdümlü Mimari
UML: Unified Modelling Language
RTED: Rol Tabanlı Erişim Denetimi
EJB: Enterprise JavaBeans
OCL: Object Constraint Language
MOF: Meta-Object Facility
GMD: Güvenlik Modelleme Dili

1. GİRİŞ

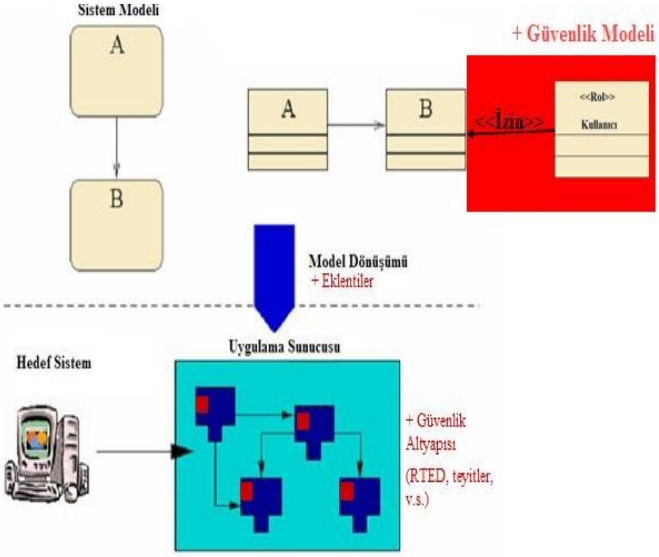
Model inşası yazılım mühendisliğinde standart bir pratiktir. Sistem gereksinimlerinin analizi ve sistem tasarımı aşamalarında inşa edilen model, erken analiz ve olası hataları belirleyebilme olanağı sağlayarak ortaya çıkan sistemin kalitesini artırabilir. Ortaya çıkan sistemin geliştirilme aşamasında modeller sistem tanımlayıcı olarak da kullanılabilirler. Modeller yeterince biçimsel şekilde inşa edilebilirlerse modelden otomatik olarak kod üretme imkânı sağlanmış olur.

Model inşası, güvenliğin modellenmesinde ve politika tanımlamalarında da kullanılır fakat sistemin tüm gelişim sürecine entegrasyonu sorunlu olmaktadır. Bu sorunun başlıca iki sebebi vardır. Birincisi; sistem tasarım modellerinin ve güvenlik modellerinin ayrık olmaları ve farklı yöntemlerle ifade edilmeleridir. İkincisi; güvenlik tanımlamaları ve tehditler sistem gereksinimlerinin analizi gibi erken tasarım aşamalarında göz önünde bulundurulsa da güvenlik mekanizmaları tasarımın son aşamasında sisteme dâhil edilir. Sonuç olarak sisteme sonradan dâhil edilmeye çalışılan güvenlik, ortaya çıkan sistemin bakımını zorlaştırır ve güvenilirliğini azaltır.



Şekil 1: Model Güdümlü Mimari

Model Güdümlü Güvenlik (MGG) sistem tasarım modeli ile güvenlik modeli arasında köprü vazifesi görmeyi hedefler. Model Güdümlü Mimari (MGM) [Frankel 2003], MGG kavramının geliştirilmesi çalışmalarının ilk adımını oluşturur. MGM, model-merkezcil ve üretken bir yaklaşım sunarak yazılım geliştirme sürecine yardımcı olur. Şekil 1'de de görüldüğü gibi MGM 3 parçadan oluşur: geliştiriciler Unified Modelling Language (UML) gibi yüksek seviye modelleme dillerini kullanarak (1) sistem modelleri yaratırlar; araçlar vasıtasıyla modelden koda otomatik dönüşüm (2) sağlanır; sonuç olarak ortaya hedef sistem (3) çıkar.



Şekil 2: Model Güdümlü Güvenlik

MGG, MGM'in özelleştirilmiş halidir. Bu özelleştirme işlemi için genel amaçlı modelleme dilleri ve dönüşüm formülleri, basit modelleme parçaları kullanarak ve sistem geliştirme aşamasına güvenliği de katarak, genişletilir. Şekil 2'de görüldüğü gibi MGM'nin üç parçası model erişim denetimi gereksinimleri ve erişim denetimi altyapısı oluşturma çerçevesinde özelleştirilmiştir. Birinci parça, erişim denetim gereksinimlerinin sistem tasarım modellerine entegre olacak şekilde özelleştirilmiştir. *Diyalekt*, tasarım dilinin hangi elemanlarının güvenlik dilinin korunan kaynakları olacağını tanımlayarak güvenlik modelleme dili ile sistem tasarım dilini birleştirir.

MGG kavramının geliştirilmesi aşamasındaki bir sonraki adım; Rol Tabanlı Erişim Denetimi'ni (RTED) [Ferrailo et al, 2001] genelleştiren erişim denetim gereksinimlerini modelleyen güvenlik modelleme dilini tanımlamaktır. Görsel modelleme desteği için bu dil UML'e bir genişletme olarak gömülür, ortaya çıkan dile GüvenliUML denir.

Enterprise JavaBeans (EJB) ve .NET gibi standartlar için dönüşüm fonksiyonları uygulanarak MGG çatısı altında güvenlik altyapıları için otomatik kod üretmek mümkündür. Güvenlik modeliyle sistem geliştirme modeli arasındaki boşluğu dolduran bu yeni yapıya güvenlik tasarım modeli denir.

MGG'nin başlıca avantajları;

1. Teknoloji bağımsız, tekrar kullanılabilir ve evirilebilir sistemler ortaya çıkarır. Kullanılan teknolojiler değişse bile dönüşüm fonksiyonlarındaki küçük değişikler yeni gelen teknolojiye ayak uydurmaya yetecektir.
2. Biçimsel güvenlik politikalarıyla tutarlı güvenli uygulamalar geliştirmeye yardımcı olur.
3. UML'e birçok geliştiricinin aşinadır ve geniş bir kesim tarafından kabul görmüştür. UML modellerini işleyebilecek birçok araç mevcuttur.

2. ALTYAPI

2.1. Bir Tasarım Problemi

Tasarlanmak istenen sistemde bir grup kullanıcı ve toplantılar olacaktır. Bir toplantının sahibi, katılımcıları, yeri ve zamanı vardır. Kullanıcılar toplantılar üzerinde yaratma, silme, okuma ve düzenleme gibi işlemler yapabileceklerdir. Bir kullanıcı bir toplantıyı iptal edebilecektir. Bu işlemi takiben katılımcılar uyarılacak ve toplantı silinecektir.

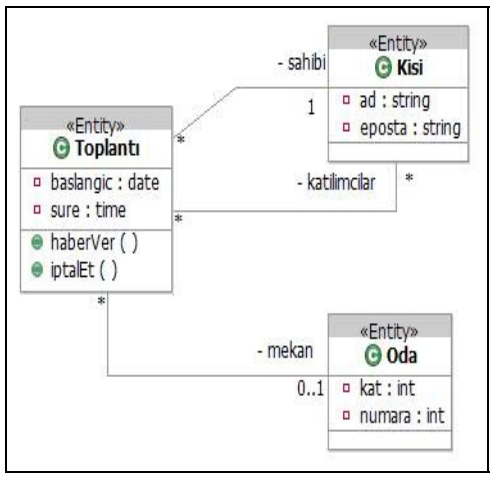
Sistemin güvenlik politikaları ise şu şekildedir.

1. Kullanıcıların tümü toplantı oluşturabilecektir ve var olan tüm toplantıların bilgilerine erişebilecektir.
2. Sadece toplantının sahibi toplantıyı iptal edebilecek, silebilecek ve bilgilerini güncelleyebilecektir.
3. Bir yönetici herhangi bir toplantıyı iptal edebilecektir.

2.2. Unified Modelling Language (UML)

UML [Rumbaugh et al., 1998], geniş kesimler tarafından kullanılan, nesneye dayalı sistemleri modellemek için kullanılan grafiksel bir dildir. Sistemin tanımı soyut ve somut sentaks (notasyon) olmak üzere ikiye ayrılır. Soyut sentaks modellerin inşa edilmesinde kullanılan temel yapıları tanımlar. Somut sentaks ise bu temel yapıların grafiksel gösterimini tanımlar. UML, farklı model element tiplerini ve karşılık düşen diyagram tiplerini kullanarak sistemin yapısının ve davranışının tanımlanmasına destek verir.

Sistemin yapısal çehresi sınıflar kullanılarak tanımlanır. Her bir sınıf ortak servisler, özellikler ve davranışlar barındıran nesneleri biçimselleştirir. Servisler metotlarla, özellikler attribute'larla ve ilişkilerle ifade edilir. Bağlantılı olduğu diğer sınıflarla ilişkileri ilişki uçları ve ilişki kat sayılarıyla ifade edilir.



Şekil 3: Toplantı Düzenleyici uygulamasının Sınıf Diyagramı

Şekil 3 Toplantı Düzenleyici uygulamasının yapısını gösterir. Model üç tane sınıftan; Toplantı, Kişi ve Oda; oluşur. Toplantı başlangıç saatini ve toplantı süresini barındıracak attribute'ler içerir. Toplantı sahibi, katılacak kullanıcılar ve toplantının yeri ilişki uçları; sahibi, katılımcılar ve mekân; olarak ifade edilmiştir. HaberVer metodu katılımcıları değişikliklerden haberdar eder. İptalEt metodu ise katılımcıları toplantı iptalinden haberdar ederek oda rezervasyonunun kaldırır.

UML ayrıca birinci derece mantık tabanlı Object Constraint Language (OCL) tanım dilini sağlar. OCL tanımlamaları sınıfların sabit niceliklerini, metotların önkoşullarını ve sonkoşullarını biçimselleştirmek için kullanılır. Örnek olarak Şekil 3'deki Toplantı sınıfına şu OCL tanımını ekleyebiliriz:

context Toplantı **inv:**
self.katılımcılar->includes(self.sahibi)

Bu tanımla anlatılmak istenen; Toplantının sahibi katılımcılar arasında olmalıdır. Her OCL tanımı ismi belirtilen bir sınıfın bağlamında değerlendirilir. Self bağlam içinde bulunduğu sınıfı ifade eder. Sınıfın attribute'larına, metotlarına ve ilişki uçlarına "." veya "->" gösterimleriyle ulaşılır. Örnekteki tanımda toplantının katılımcı listesi attribute'ının includes metoduna toplantının sahibi parametre olarak geçirildiğinde doğru cevabı dönmelidir.

Stereotipler, çekirdek UML tiplerini alt tiplere ayırarak yeni dil yapıtaşları tanımlar. Şekil 3'de sınıflara eklenmiş olan <<Entity>> stereotipi bu sınıfların iş domeni modelinin temel sınıfları olduklarını ifade eder.

2.3. Model Güdümlü Mimari (MGM)

MGM, uygulamaları tanımlamak ve geliştirmek için sistemlerin modeller olarak gösterildiği bir ortam sunar. Dönüşüm fonksiyonları modellerden çalıştırılabilir kod üretmek için kullanılır [Frankel, 2003]. Bu bağlamda güvenlikte biçimselleştirilerek erişim denetimi çatısı altında güvenlik altyapıları üretmek için kullanılabilir.

Domene-özgü diller kullanımı MGM'nin en önemli özelliğidir. Örneğin; modelleme dilleri farklı iş domenlerini (bankacılık, finans, sağlık) , sistem çehrelerini (güvenlik gibi) veya somut teknolojileri (EJB, .NET v.s.) biçimselleştirme kabiliyetine sahiptir. UML modelleme dâhilinde böylesi dilleri tanımlamak için üç tane farklı yöntem vardır. Birincisi; domene özgü diller direkt olarak UML'de Stereotipler yardımıyla tanımlanır. İkincisi; UML metamodel Meta-Object Facility (MOF) [Object Management Group, 2002] vasıtasıyla genişletilir. Üçüncüsü; MOF yardımıyla yeni bir modelleme dili tanımlanır. Bu yeni dilin somut sentaksı UML profilleri kullanılarak tanımlanır. MGG kavramının geliştirilmesinde üçüncü yol takip edilmiştir.

2.4. Rol Tabanlı Erişim Denetimi (RTED)

Matematiksel olarak, erişim denetimi ihtiyaçları, Kullanıcılar kümesi ve İzinler kümesi arasındaki ED ilişkisi olarak ifade edilebilir.

$$ED \subseteq \text{Kullanıcılar} \times \text{İzinler}$$

Kullanıcı *k*, *i* izne sahiptir sadece ve sadece $(k,i) \in ED$ iken.

RTED'in ana fikri roller kümesi tanımlamak ve ED ilişkisini şu iki ilişkiye ayrıştırmaktır; kullanıcı tayini KT ve izin tayini İT:

$$KT \subseteq \text{Kullanıcılar} \times \text{Roller}, \qquad İT \subseteq \text{Roller İzinler}$$

Erişim denetim ilişkisi basitçe bu ilişkilerin bileşkesidir:

$$ED = İT \circ KT$$

"o" ilişki bileşkesini sembolize eder. ED'nin bir başka tanımı:

$$ED = \{(k,i) \in \text{Kullanıcılar} \times \text{İzinler} \mid \exists\, rol \in \text{Roller}.(k,rol) \in KT \wedge (rol, i) \in İT\}$$

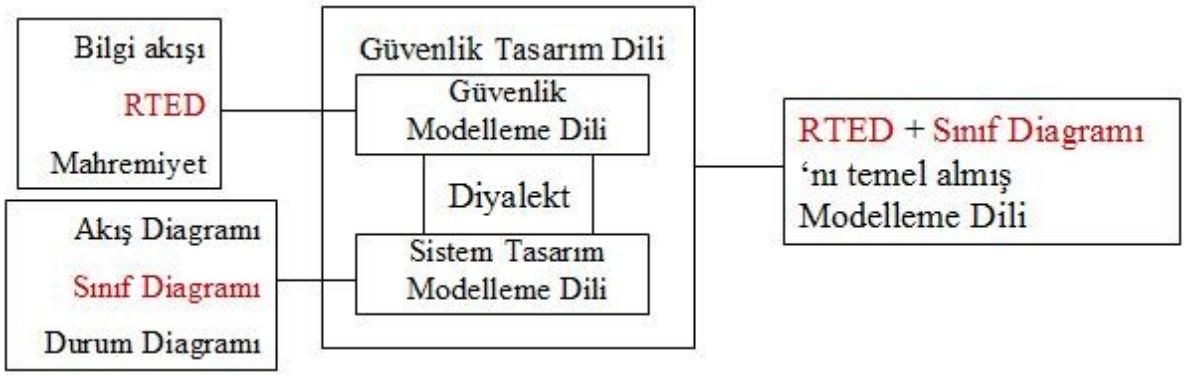
RTED ayrıca hiyerarşi için de notasyona sahiptir. Bu hiyerarşi matematiksel olarak roller kümesindeki kısmi sıra ≥ ;geniş roller küçük rollerin izinlerini miras alır; olarak gösterilebilir. Bu bilgiye dayanarak erişim denetimi şu eşitlikle ifade edilebilir:

$$ED = İT\ o\ \geq o\ KT$$

Sistemin dinamik değerleri, zaman gibi, üzerine hazırlanan güvenlik politikaları RTED kullanılarak biçimselleştirilemez. Bu kısıt sorunu için RTED yetki kısıtlamalarıyla genişletilmiştir ve böylece Güvenlik Modelleme Dili'nin (GMD) temelini oluşturmuştur.

3. MODEL GÜDÜMLÜ GÜVENLİK : TANITIM

MGG girişte de bahsedildiği gibi güvenlik modeli ile sistem tasarım modeli, tasarım ve uygulama arasındaki boşluğu doldurmayı hedefler. Bu hedefe ulaşmak için güvenliği modelleme diline entegre ederek bir model güdümlü geliştirme süreci ortaya atar.



Şekil 4: Güvenlik Tasarım Dili Şeması

Güvenlik modeli için sadece bir dil tanımlamak yerine MGG'de böylesi bir dilin modüler olarak nasıl inşa edileceğini tarif eden bir şema tanımlanmıştır. Bu şemanın genel yapısı Şekil 4'de gösterilmektedir. Şema şu üç dille parametrik hale getirilmiştir:

1. Güvenlik politikalarını tanımlayabilmek için bir *güvenlik modelleme dili*
2. Tasarım modellerini yapılandırmak için bir *sistem tasarım modelleme dili*
3. (1) ve (2) arasında temas noktaları tanımlayarak bir köprü vazifesi gören *diyalekt*

Bu üç parametreyi farklı şekillerde tanımlayarak farklı çeşit tasarımlar ve güvenlik politikaları barındıran farklı diller üretmek mümkünüdür.

3.1. Güvenlik Modelleme Dilleri

Bir güvenlik modelleme dili iyi tanımlanmış somut ve soyut sentaksa sahip biçimsel bir dildir. Soyut sentaks gramer gibi resmi olarak tanımlanırken, somut sentaks gayri resmi olarak tanımlanır. Soyut sentaks MOF kullanılarak oluşturulurken, somut sentaks UML profilleri kullanılarak oluşturulur. Daha sonra ayrıntılandırılacak olan GüvenliUML MGG'nin güvenlik modelleme dilidir. GüvenliUML'nin soyut sentaksı ve semantiği UML'den bağımsız erişim denetim politikaları tanımlar.

3.2. Sistem Tasarım Dilleri ve Diyalektler

Bir tasarım modelleme dilini güvenlikli hale getirebilmek için bir güvenlik modelleme dili ile birleştirmek gerekmektedir. Bu birleştirme aşamasında her iki dilin soyut ve somut sentaksı düzeyinde sözlükleri bütünleştirilir. Fakat daha fazlası gerekmektedir: farklı dillerden alt tanımları birleştiren birleştirilmiş dilde, tanımlar inşa edilebilmesi gerekmektedir. Güvenlik modelleme dilinde ifade edilmiş güvenlik politikaları, tasarım modelleme dilinde tanımlı attribute'lar veya sistem kaynakları hakkında ifadeler tanımlayabilmelidir. İşte bu bağlantıyı sağlamak diyalektin görevidir.

GüvenliUML, korunan kaynaklar üstündeki işlemler için erişim denetim politikalarını tanımlayan bir dil sağlar. Fakat nelerin korunan kaynak olduğunu ve hangi işlemlerin istemcilere sunulduğu konusunu açık bırakır. Örneğin bileşene-dayalı modelleme dilinde kaynaklar erişilmek istenen metotlar olabilir. Alternatif olarak, işleme-dayalı modellemede ise kaynaklar başlatılmak veya durdurulmak istenen işlemler olabilir. Veya bir dosya sistemi modelleniyorsa kaynaklar bu sefer dosyalar olacaktır. Diyalekt, sistem tasarım dilinin hangi elemanlarının, erişim denetim bilgisi dâhilinde korunacak kaynak olduğunu tanımlayarak GüvenliUML'i sistem tasarımına dâhil eder. Bu kaynaklarla ilgili güvenlik politikalarını şunları tanımlayarak biçimselleştirir;

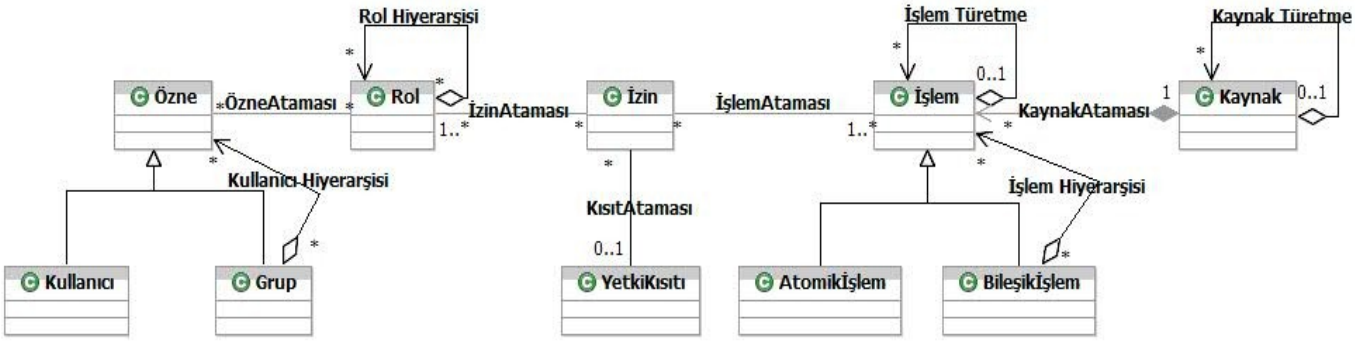
- sistem tasarım modelinde nelerin korunması gereken kaynaklar olacağı
- bu kaynaklar üstündeki işlemler ve bu işlemlerin hiyerarşik yapıları
- bir işlem için açıkça bir izin tanımı yapılmadıysa, varsayılan erişim denetimi politikasının ne olacağı (erişim her zaman kabul mü edilecek yoksa ret mi edilecek)

3.3. Model Dönüşümü

Sistem modelinden çalışabilir kod üretebilmek için dönüşüm fonksiyonları kullanılır. Güvenlik kavramını tasarım aşamasına taşımadan önce kullanılan dönüşüm fonksiyonları yeni kavrama uygun olarak değiştirilir veya genişletilir. Yeni fonksiyonlar tasarlanırken dikkat edilmesi gereken en önemli husus güvenlik dâhil edilmeye çalışırken tasarım tarafındaki dönüşümün bozulmamasıdır. Güvenlik modelleme dilinin semantiğine uygun bir dönüşüm fonksiyonuna ihtiyaç vardır öyle ki; bu fonksiyon semantik sakınan olmalıdır.

Dönüşüm fonksiyonlarının doğruluğunu yargılayabilmek için hedeflenen güvenlik mimarilerinin biçimsel bir semantiğine ihtiyaç vardır. MGG'de hedeflenen sistem yüksek seviye soyutlama ile modellenir. Sistemin çalışması, korunan işlemleri çalıştırabilmek için yapılan bir dizi denemeler olarak soyutlaştırır. Her bir denemede işlemin çalıştırılabilir olduğunu kontrol etmek için güvenlik monitörü (1) statik rol ve izin atamalarını kontrol eder ve kodu (dönüşüm fonksiyonları tarafından üretilen) işleyerek sonucun "Doğru" döndüğünü kontrol eder. Dönüşüm fonksiyonlarının doğruluğundan kasıt güvenlik monitörü bir işlemin çalıştırılabilir olduğuna sadece güvenlik tasarım dilinin semantiğinde izin verildiyse onay vermelidir.

4. GüvenliUML



Şekil 5: GüvenliUML metamodeli

4.1. Soyut Sentaks

Şekil 5'de GüvenliUML'in soyut sentaksını tanımlayan metamodel gösterilmektedir. Bu dil bazı yönlerden genişletilmiş RTED'i temel alır. Diyagramın sol tarafı genişletilmiş RTED'i göstermektedir. RTED Kullanıcıları Gruplar kullanılarak genişletilmiş ve ortak üst tip olan Özne kullanılarak kullanıcıların ve grupların rollere atanmaları sağlanmıştır. Diyagramın sağ tarafı kaynaklara işlemlerin atanması ve bu işlemlerin üzerindeki izinleri göstermektedir. İzinler, belli sistem durumlarında *yetki kısıtları* vasıtasıyla kısıtlanır. RTED'de standart olarak bulunan rol hiyerarşisine ek olarak işlemler üzerinde de hiyerarşi tanımlanmıştır.

Özne tüm kullanıcıların ve grupların taban sınıfıdır ve soyut tiptedir. Bir Kullanıcı sistem entity'sini, bir insan ya da bir işlem gibi, gösterirken bir Grup kullanıcı ve grup kümelerini temsil eder. Özneler rollere ÖzneAtaması ilişkisiyle atanmıştır.

Bir Rol bir işi temsil eder ve o işi gerçekleştirebilmek için gereken tüm imtiyazları bir araya getirir. Bir İzin bir ya da daha fazla işleme rol tarafından erişimi tahsis eder. İşlemler İşlemAtaması tarafından İzinlere atanırlar. İzinler ise rollere İzinAtaması tarafından atanırlar. Roller hiyerarşik olarak sıralanabilirler. Bu hiyerarşi RolHiyerarşisi aggregation'ı ile tanımlanır. Bu hiyerarşideki altta kalan (rolü genişleten) roller üstteki rollerin tüm imtiyazlarını miras alırlar.

Bir YetkiKısıtı mantıksal bir beyandır ve İzin'e KısıtAtaması ilişkisiyle bağlanır. Sistemin durumuna örneğin sistem zamanı ya da attribute değerlerine göre izinin geçerli olup olmadığına karar verir. YetkiKısıtı ifadelerinin tanımlayabilmek için OCL kullanılır. Sistem modelleri OCL'in sözlüğünü oluşturur (sınıflar ve metotlar), bu standart sözlük *çağıran* sözcüğüyle genişletilir. Çağıran, işlemi gerçekleştirmek isteyen kullanıcının adını temsil etmektedir.

Kaynak, korunan kaynakları temsil eden tüm model elemanlarının taban sınıfıdır. Kaynak üzerinde gerçekleştirilebilecek olası operasyonlar İşlem sınıfıyla ifade edilir. Her kaynak üzerinde gerçekleştirilebilecek bir ya da daha fazla işlem varken, bir işlem sadece bir kaynak üzerinde tanımlıdır ve KaynakAtaması ile ilişkilendirilir. İşlemin AtomikİşlemBileşikİşlem olmak üzere iki alt tipi vardır. AtomikİşlemBileşikİşlem direkt hedef platformla ilişkili düşük seviye işlemdir. Örneğin, bir sınıfın metodunu çağırmak bir AtomikİşlemBileşikİşlem'dir. BileşikİşlemBileşikİşlem direkt olarak bir hedef platformla ilişkili olmayan yüksek seviye işlemdir. Bileşik işlemler bir İşlemHiyararşi'si içinde sıralanmışlardır ve işlemleri gruplamak için kullanılırlar.

BileşikİşlemŒle ilişkilendirilmiş bir İzin, hedef kaynak üzerinde tanımlanmış ve bu bileşik işlemler arasında bulunan herhangi bir işlemi yapabilme yetkisini tanımlar. Örneğin; bir sınıf üzerinde tanımlı "Okuma" BileşikİşlemŒi, o sınıfın tüm attributeŒlarını okuma izni ve yan-etki barındırmayan tüm metotlarını çağırma iznini barındırır. Bu BileşikİşlemŒe bağlı İzne sahip olan roller grup dâhilindeki işlemleri yapma hakkına sahiptirler.

4.2. Somut Sentaks

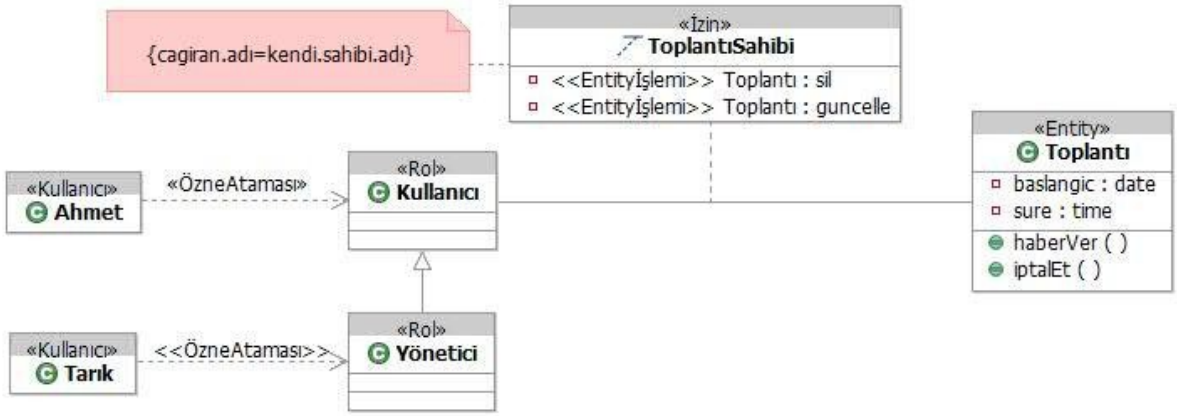
GüvenliUMLŒin somut sentaksı UML tabanlıdır. Somut sentaksın oluşturulması için GüvenliUMLŒin modelleme notasyonunu biçimselleştiren, stereotipler kullanılarak bir UML profili tanımlanmıştır.

UML metamodel tipi ve Stereotip	GüvenliUML metamodel tipi
Sınıf <<Kullanıcı>>	Kullanıcı
Sınıf <<Grup>>	Grup
Bağımlılık <<ÖzneGrubu>>	ÖzneGrubu
Bağımlılık <<ÖzneAtaması>>	ÖzneAtaması
Sınıf <<Rol>>	Rol
Sınıflar arasında stereotipler ile genelleştirme <<Rol>>	RolHiyerarşisi
İlişkiSınıfı <<İzin>>	İzin, İzin Ataması, İşlemAtaması, YetkiKısıtı ve KısıtAtaması

Tablo 1: GüvenliUML soyut ve somut sentaks eşlemesi

Tablo 1 GüvenliUML metamodeli elemanları ile UML tipleri arasındaki eşleşmeyi gösterir. Bir İzin, diğer elemanlarla ilişkileri ve üzerindeki olası yetki kısıdı sadece *UML ilişki sınıfı* ile gösterilir. Tüm GüvenliUML elemanları için bir eşleşme söz konusu değildir. Örneğin, Kaynakların nasıl tanımlanacağının notasyonu açık bırakılmıştır. Bu tanımı diyalektin yapması gerekmektedir. Arıca Özne içinde bir gösterim tanımlanmamıştır çünkü Özne tipi *soyut*tur.

GüvenliUML profilinde bir Rol, <<Rol>> stereotipli bir UML sınıfı ile gösterilmektedir. İki Rol arasındaki miras ilişkisi UML genelleme ilişkisi ile tanımlanmıştır. Bir Kullanıcı, <<Kullanıcı>> stereotipli bir UML sınıfı ile gösterilmektedir. Özne ile Rol arasındaki bağımlılık ilişkisi ise <<ÖzneAtaması>> stereotipi ile gösterilmiştir.



Şekil 6: Güvenli UMLŒin somut sentaksına bir örnek

Şekil 6'nın sağ kısmında korunan bir kaynağın üzerinde ki İzin gösterilmektedir. Her UML model eleman tipi bir GüvenliUML soyut sentaks korunan kaynağı olarak gösterilebilir. Bir GüvenliUML diyalekti sistem tasarım dilinin hangi elemanlarının korunacak kaynaklar olacağına karar vererek bu elemanların UML gösterimleriyle GüvenliUML soyut sentaksı kaynak tipleri arasında eşleşme sağlar. Örneğin; bir diyalekt, <<Entity>> stereotipli UML sınıflarını üzerinde "güncelleme" ve "silme" işlemleri tanımlı *korunan kaynak* olarak biçimselleştirebilir. Şekil 6'daki Toplantı bir korunan kaynaktır.

Bir İzin, Rollere İzinAtaması ile İşlemlere ise İşlemAtaması ile bağlanır. İzin, <<İzin>> stereotipli bir UML ilişki sınıfı şeklinde ifade edilir. İlişki sınıfı olarak tanımlanmasının avantajlarından biri aralarında ilişki kurduğu sınıflardan biri modelden kaldırılırsa ilişki sınıfı olan İzin'de otomatik olarak modelden kaldırılır. İzin, korunan kaynak olarak atanan bir UML sınıfı ile bir Rol'ü birleştirir. Şekil 6'da korunan kaynak *Toplantı* üzerinde güncelleme ve silme işlemlerini barındıran *ToplantıSahibi* iznine Kullanıcı tipindeki *Rol* sahiptir.

Her yetki kısıdı OCL ifadesiyle gösterilir ve somut sentaksta ilişki sınıfı olan İzin'e bir not şeklinde bağlanır. Bağlandığı bu ilişki sınıfı ve ilişkilendirilen sınıflar kısıdın bağlamını oluşturur. Şekil 6'da ToplantıSahibi izni bir yetki kısıdıyla kısıtlanmıştır;
çağıran.adı = kendi.sahibi.adı
bu ifade toplantının güncelleme ve silme işlemini sadece toplantının sahibinin gerçekleştirebileceğini anlatır.

4.3. Semantik

GüvenliUML erişim denetim kararlarını şu iki bilgiye dayanarak biçimselleştirir:

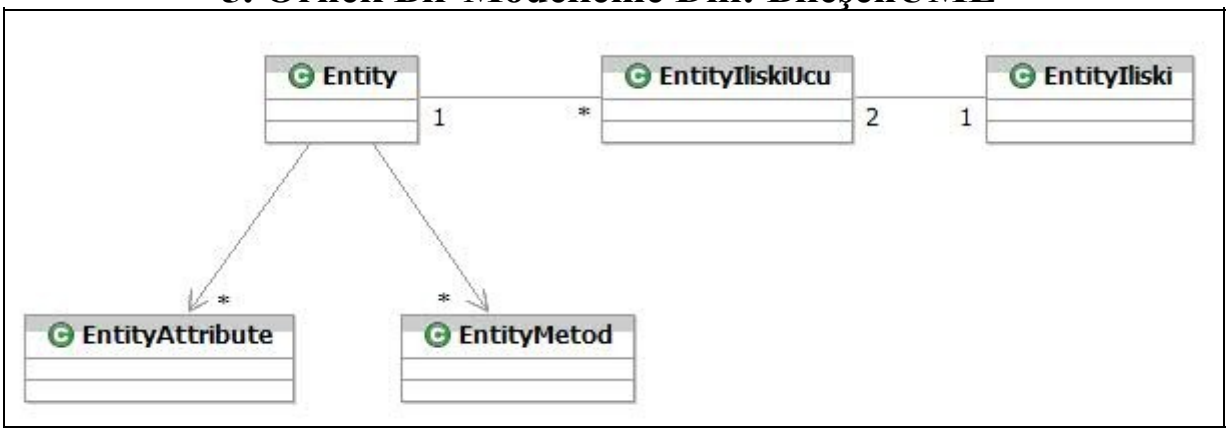
1. Statik bilgiye; RTED konfigürasyonunda tanımlandığı gibi kullanıcı ve izinlerin rollere atanması, dayanan *Deklaratif* erişim denetimi kararları.
2. Dinamik bilgiye; güncel sistem durumu içindeki yetki kısıtlarının karşılanması, dayanan Programatik erişim denetimi kararları.

Deklaratif erişim denetimi kararlarını (1) biçimselleştirebilmek için bir birinci derece yapı δ_{RTED} olarak RTED konfigürasyonu tanımlanır. Deklaratif erişim denetimi kararlarının semantiği $\delta_{RTED} \vDash \Phi_{RTED}(k,i)$ ile formülüze edilir. $\Phi_{RTED}(k,i)$: k Kullanıcısının i İşlemini gerçekleştirebilmesi içi gereken *doğru rol*'e sahip olması gerektiğini belirtir.

Programatik erişim denetimi kararlarını (2) biçimselleştirebilmek için birinci derece yapılar δ_{Sd} tarafından sistem durumları *sd* tanımlanır. İzin kısıtları birinci derece formüller $\Phi^i_{Sd}(k)$ (sd durumundan bağımsız) ile gösterilir. GüvenliUML metamodeline göre kısıtlar izinlerle ilişkilendirilmiştir (işlemler ile değil) ve bu formül hangi koşulda k kullanıcısının i iznine sahip olacağını biçimselleştirir. Bu koşul sd durumuna uysun ya da uymasın $\delta_{Sd} \vDash \Phi^i_{Sd}(k)$ mantıksal karar problemi olarak değerlendirilir.

RTED konfigürasyonu ile yetki kısıtlarını birleştirmek için birinci derece yapılar olan δ_{RTED} ve δ_{Sd}, ve birinci derece formüller olan $\Phi_{RTED}(k,i)$ ve $\Phi^i_{Sd}(k)$ birleştirilmelidir. Kabaca birleştirilen bu semantik şu şekilde ifade edilir: $< \delta_{RTED}, \delta_{Sd} > \vDash \Phi_{ED}(k,i)$. $\Phi_{ED}(k,i)$, $\Phi^i_{Sd}(k)$ ve $\Phi_{RTED}(k,i)$'nin birleştirilmesi ile oluşturulur. $< \delta_{RTED}, \delta_{Sd} >$ ise δ_{RTED} ve δ_{Sd} yapılarının bileşimi demektir.

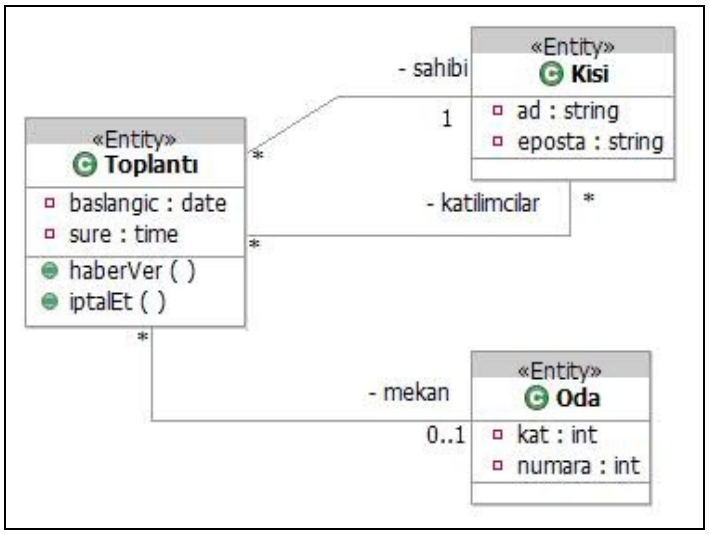
5. Örnek Bir Modelleme Dili: BileşenUML



Şekil 7: BileşenUML metamodeli

BileşenUML, bileşen-tabanlı sistemleri modellemek için kullanılan basit bir dildir. Şekil 7'de BileşenUML'in metamodeli gösterilmektedir. Entity tipinin elemanları dikkate değer bir domenin obje tiplerini temsil eder. Bir Entity birden çok metot ve attribute'a sahip olabilir, bunlar EntityMetod ve EntityAttribute ile temsil edilirler. İlişkiler diğer Entity'lerle olan bağı ifade ederler. Bir ilişki EntityIlişki model elemanıyla ifade edilir. Diğer Entity'lerle ilişkisi olan bir Entity bir ilişkiye EntityIlişkiUcu ile bağlanır.

BileşenUML, varlıkların <<Entity>> stereotipli UML sınıflarıyla temsil edildiği UML tabanlı bir notasyon kullanır. Her metot, attribute ya da sınıfa bağlı ilişki ucu, metot, attribute ya da entity'nin ilişki ucu olarak hesaba katıldığı için daha fazla stereotipin kullanılması gerekli değildir.



Şekil 8: Toplantı Çizelgesi Uygulaması

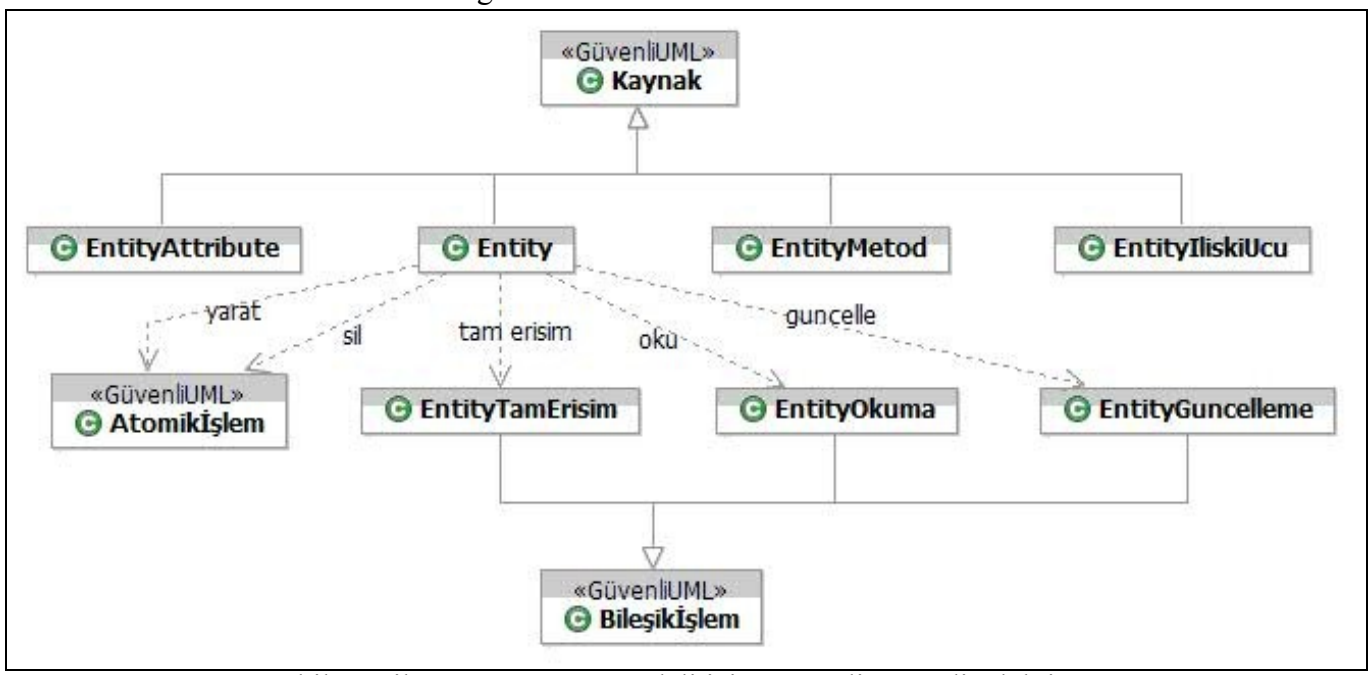
Şekil 8, toplantı çizelge uygulamasının yapısal modelini BileşenUML notasyonunda gösterir. Toplantı, Kişi ve Oda varlıkları, sınıflar yerine <<Entity>> stereotipli UML sınıflarıyla temsil edilmişlerdir.

5.1. Soyut Sentaksı Genişletme

BileşenUML güvenlikli hale getirilmek için birinci adım olarak soyut sentaksı, metamodeli GüvenliUML'in metamodeliyle birleştirilerek genişletilmiştir. Bu birleşme GüvenliUML'in

metamodelinin BileşenUML'in metamodeline dâhil edilmesi suretiyle gerçekleştirilmiştir. Bu BileşenUML'i GüvenliUML'in modelleme yapıları, örneğin İzin ve Rol ile genişletir.

İkinci adım olarak; BileşenUML'in model elemanlarından hangilerinin korunacak kaynaklar olacağına karar verilir ve bunlar bir GüvenliUML diyalekti olarak biçimselleştirilirler. Bunu yaparken hangi model elemanlarının üzerinde erişim denetimi istendiğine, hangi elemanların hedef sistemde korunmak istendiğine karar verilmelidir.



Şekil 9: BileşenUML metamodeli için GüvenliUML diyalekti

Örnekte BileşenUML'in korunan kaynak model elemanları; Entity, EntityMetod, EntityAttribute ve EntityIlişkiUcu'dur. Bu tanımlama Şekil 9'da görüldüğü gibi miras alma yöntemiyle gösterilmiştir. BileşenUML'in korunan kaynakları GüvenliUML'in Kaynak tipinin alt tipleridir. Böylelikle metatipler yetki politikaları tanımlamak için gereken tüm özellikleri miras alırlar. Ek olarak Şekil 9'da bazı işlem sınıflarının GüvenliUML'in BileşikİşlemEntityTamErisim tipinin alt tipi oldukları gösterilmiştir.

Şekil 9'da görüldüğü gibi her bir kaynak tipinin önerdiği işlemler, kaynak tipinden işlem sınıfına isimlendirilmiş bağımlılıklar tarafından tanımlanmışlardır. Her bir bağımlılık, kaynak tipinin bağlamı içinde referans edilen işlem tipindeki bir işlemi temsil eder. Bağımlılığın ismi işlemin ismini belirler. Örneğin şekilde görülüyor ki; bir Entity EntityTamErisim tipindeki tam erişim işlemine her zaman sahiptir ve AtomikİşlemEntityTamErisim tipindeki yaratma ve silme işlemlerini her zaman gerçekleştirebilir.

Kaynak Tipi	İşlem	Alt işlemler (**atomik**)
Entity	Tam erişim	**Yarat**, oku, güncelle ve **sil**
Entity	Okuma	Tüm attribute'ları ve ilişki uçlarını **oku** Yan-etki barındırmayan metotları **çalıştır**
Entity	Güncelleme	Tüm attribute'ları **güncelle** İlişki uçlarına **ekle** ve İlişki uçlarını **çıkart** Yan-etki barındıran metotları **çalıştır**
Attribute	Tam erişim	**Oku** ve **Güncelle**
İlişki Ucu	Tam erişim	**Oku**, **ekle** ve **çıkart**

Tablo 2: GüvenliUML Diyalekt İşlem Hiyerarşisi

Üçüncü ve son adımda roller üzerinde hiyerarşi tanımlanır. Bu işlem diyalektin her bir bileşik işlem tipi üzerinde GüvenliUML'in ilişki İşlemHiyerarşisi domenini OCL değişmezleriyle kısıtlamakla yapılır. BileşenUML için GüvenliUML diyalekti bileşik işlemleri Tablo 2'de anlatılmıştır. EntityTamErisim için OCL ifadesi;

context EntityTamErisim inv:
altİşlemler = kaynak.işlemler->seç(ad="yarat" ya da ad="oku" ya da ad="güncelle" ya da ad="sil")

Bu ifadenin anlamı BileşikİşlemTipindeki EntityTamErisim, entity üzerindeki işlemler olan; yaratma, okuma, güncelleme ve silmeden daha geniştir.

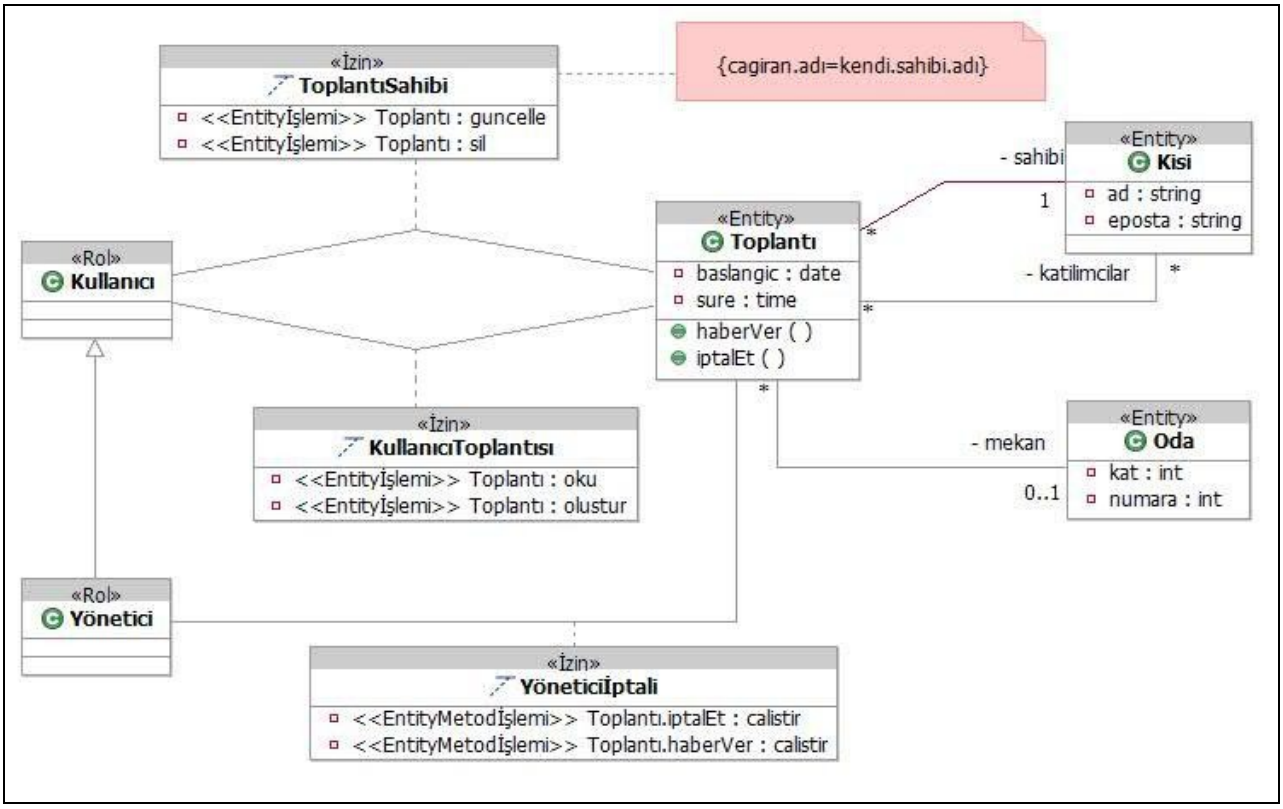
5.2. Somut Sentaksı Genişletme

Stereotip	Kaynak Tipi	İsimlendirme Kuralı
EntityİşlemiTipi	Entity	boş string
MetotİşlemiTipi	Metot	metot imzası
AttributeİşlemiTipi	Attribute	attribute ismi
İlişkiUcuİşlemiTipi	İlişkiUcu	ilişki ucu ismi

Tablo 3: BileşenUML için işlem referans tipleri

GüvenliUML notasyonunun BileşenUML notasyonu içine aktarılmasıyla somut sentaks genişletilir. Korunacak kaynakları temsil eden BileşenUML elemanlarında GüvenliUML primitif tipleri kullanılmasını kısıtlayan iyi biçimlendirilmiş kurallar tanımlanır. Tablo 3'de gösterildiği gibi, entity'ler, attribute'lar, metotlar ve ilişki uçları için işlem referans tipleri tanımlanır.

5.3. Yetki Politikasını Modelleme



Şekil 10: Yetki Politikaları eklenmiş Toplantı Çizelgesi Uygulaması

Şekil 10'da Bölüm 2.1'de tanımlanmış toplantı çizelgesi uygulamasının birleştirilmiş dilde nasıl biçimselleştirildiği gösterilmiştir. Gereksinimlerdeki üç güvenlik politikası maddesini biçimselleştirebilmek için modele İzin'ler eklenmiştir. Bölüm 4.2'de anlatıldığı gibi Kullanıcı ve yönetici rolleri de izinler vasıtasıyla kaynaklara bağlanmışlardır.

Birinci maddenin belirttiği; kullanıcıların tümü toplantı oluşturabilecektir ve var olan tüm toplantıların bilgilerine erişebilecektir, politikası KullanıcıToplantısı İzni ile biçimselleştirilmiştir. Bu izin Kullanıcı rolündeki kişilerin Toplantı üzerinde oku ve oluştur işlemlerini gerçekleştirebilmelerine imkân tanır.

İkinci maddenin belirttiği; sadece toplantının sahibi toplantıyı iptal edebilecek, silebilecek ve bilgilerini güncelleyebilecektir, politikası ToplantıSahibi İzni ile biçimselleştirilmiştir. Bu izin Kullanıcı rolündeki kişilerin Toplantı üzerinde güncelleme ve silme işlemlerini gerçekleştirebilmelerine imkân tanır. Bu İzin *çağıran.adı=kendi.sahibi.adı* yetki kısıdı ile kısıtlanmıştır yani işlemi çağıran Kullanıcı rolündeki kişinin adı Toplantının sahibinin adı ile aynı olmalıdır. Tablo 2'de de belirtilen *güncelle* işleminin tanımından ötürü bu izne sahip kişiler Toplantı sınıfının tüm attribute'larının ve ilişki uçlarının değerlerini değiştirme, haberVer metodunu ve yan-etki barındıran iptalEt metodunu çağırma yetkisine sahiptirler.

Üçüncü maddenin belirttiği; bir yönetici herhangi bir toplantıyı iptal edebilecektir, politikası YöneticiİptaliŞ İzni ile biçimselleştirilmiştir. Bu izin Yönetici rolündeki kişilerin Toplantı üzerinde iptalEt ve haberVer işlemlerini gerçekleştirebilmelerine imkân tanır.

Yönetici rolü Kullanıcı rolünü genişlettiği için Kullanıcı rolünün sahip olduğu tüm izinlere miras yöntemi sayesinde sahiptir.

6. Bir Güvenlik Altyapısı Sistemini Üretme: EJB

6.1. EJB için temel üretim kuralları

Üretim kuralları Entity'ler, attribute'ları, metotları ve ilişki uçları için tanımlanmıştır. Dönüşüm sonrasında oluşturulan kodlar EJB notasyonuna uygun Java kaynak kodları ya da XML kurulum tanımlayıcı kodlarıdır.

Bir Entity, gerekli arayüz ve gerçeklenecek sınıflar tanımlanmış bir EJB bileşeni olan "entity bean"'e dönüştürülür. Kurulum tanımlayıcı koda ise bir XML girişi olarak eklenir.

```
<entity>
    <ejb-name>Toplantı</ejb-name>
    <local-home>cizelgeci.ToplantıHome</local-home>
    <local>cizelgeci.Toplantı</local>
    <ejb-class> cizelgeci.ToplantıBean</ejb-class>
    ...
</entity>
```

Bir Metot, tanımlanmış EJB sınıfı içerisinde gövdesi boş bir metot tanımı şeklinde oluşturulur.

```
void iptalEt() { }
```

Her bir Attribute ve tekil İlişki Ucu, birer attribute olarak EJB sınıfı içinde tanımlanır ve bu attribute'ların getter ve setter metotları sınıfa eklenir.

```
private int sure;
public int getSure() { return sure; }
public void setSure(int sure) { this.sure = sure;}
```

Çoklu İlişki Uçları, birer Collection tipinde attribute olarak EJB sınıfı içinde tanımlanır ve bu listeye ekleme ve çıkarma metotları, attribute'un getter metodu sınıfa eklenir.

```
private Collection katilimcilar;
public Collection getKatilimcilar() { return katilimcilar; }
public void addKatilimci(Kisi katilici) { katilimcilar.add(katilimci); }
public void removeKatilimci(Kisi katilimci) { katilimcilar.remove(katilimci); }
```

6.2. EJB erişim denetim altyapısını üretme

Her bir İzin, kurulum tanıtıcı XML dosyasında *method-permission* girdisi olarak yaratılır. BileşenUML için GüvenliUML diyalektinde belirtildiği gibi güvenlik monitörünün varsayılan davranışı "erişime izin verilir" şeklinde olduğu için izinlerde belirtilmeyen işlemler için kod üretmek gerekli değildir. EJB' de rol hiyerarşisi tanımlı olmadığı için diğer rolleri genişleten roller izinli oldukları her işlem için genişlettikleri rolle birlikte koda eklenirler.

Kural#	Kaynak Tipi	İşlem	EJB Metotları
1	Entity	yarat	otomatik olarak üretilmiş fabrika metotları
2	Entity	sil	silme metotları
3	Metot	işlet	ilgili metot
4	Attribute	oku	attribute getter metodu
5	Attribute	güncelle	attribute setter metodu
6	İlişkiUcu	oku	ilişki ucu getter metodu
7	İlişkiUcu	güncelle	ilişki ucu add remove metotları

Tablo 4: Atomik işlem EJB metot eşleşmesi

Tablo 4'de kurulum tanıtıcısında üretilecek Atomik işlemlere denk düşen metot tanımları gösterilmektedir. Üretilen örnek bir İzin dönüşüm kodu şu şekilde olacaktır:

```
<method-permission>
    <role-name>Kullanici</role-name>
    <role-name>Yonetici</role-name>
    <method>
        <ejb-name>Toplanti</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>getSure</method-name>
        <method-params/>
    </method>
</method-permission>
```

Yukarıda deklaratif erişim denetimi altyapısının nasıl üretileceği anlatılmıştır. Erişim kısıtları gibi programatik erişim denetimi altyapısı elemanları direkt olarak Java koduna bir kısıtlayıcı olarak eklenir. Erişim kısıdı tanımlanmış bir İzin'in barındırdığı işlemlerden üretilen Java kodu metot gövdelerinin ilk satırlarına üretilen gerekli kod eklenir. Bu kod bir kapan vazifesi görerek yetkisiz erişimlerde istisna fırlatarak erişimi engeller. Örneğin bir kişinin bir toplantı attribute'ını güncelleyebilmesi için kişinin ya Kullanıcı ya da Yönetici rolünde olması ve toplantının sahibi olması gerekir. Bu kısıdı programatik olarak biçimselleştirebilmek için attribute'ın setter metodu gövdesinin ilk satırına şu kod bloğu eklenir:

```
if ( ! ( (ctxt.isCallerInRole("Yonetici") || ctxt.isCallerInRole("Kullanici"))
                && (ctxt.getCallerPrincipal.getName().equals(getOwner()) ) ){
        throw new AccessControlException("Erisim engellendi!");
}
```

7. Değerlendirme ve Sonuç

Raporda bahsi geçen örnek uygulama olan Toplantı Çizelgesi geliştirme çalışmaları sonucunda üretilen güvenlik altyapısı 179 satır XML kodu ve 268 satır JAVA kodundan oluşmuştur. Gerçek hayat projelerinden *Elektronik Evcil Hayvan Dükkânı* çalışması Model Güdümlü Güvenlik çerçevesinde geliştirilmiştir. Modellenmiş güvenlik politikası dâhilinde 6 rol (2 çeşit müşteri rolü, 4 çeşit çalışan rolü), 60 izin ve bunların 15 tanesine bağlanmış yetki kısıdı tanımlanmıştır. Bu uygulamanın EJB güvenlik altyapısı ilk örnek üreticilerde ürettirilmiş ve 5.000 satır XML kodu (uygulamada toplam 13.000 satır) ve 2.000 JAVA kodu (uygulamada toplam 20.000 satır) elde edilmiştir. Geliştiriciler bu uygulamanın Erişim Denetim Mimarisi için 2 gün, toplam geliştirme zamanı olarak 21 gün harcamışlardır.

Yapılan çalışmalar göstermiştir ki MGG;

- model yapılandırma çalışmaları için kavramsal destek sağlar.
- gereksinimlerin tasarlanması ve Model Güdümlü Geliştirme süreçlerine tam uyum gösterir.
- yarı biçimsel modellerden tam biçimsel modellere başarılı bir geçiş sağlar.
- daha hızlı, ucuz ve güvenilir uygulama geliştirme ortamı sağlar.
- daha yönetilebilir uygulamalar ortaya çıkartılmasına yardımcı olur.

8. Kaynaklar

- David Basin, Jürger Doser, Torsten Lodderstedt, 2006. Model Driven Security: From UML Models to Access Control Infrastructures. ACM Transactions on Software Engineering and Methodology, Vol. 15, No. 1, January 2006, Pages 39–91.

- David Basin, Jürger Doser, Torsten Lodderstedt, 2003. Model Driven Security for Process-Oriented Systems. SACMAT 2003.

- Torsten Lodderstedt, David Basin, Jürger Doser, 2002. SecureUML: A UML-Based Modeling Language for Model Driven Security.

- AHN, G.-J. AND SANDHU, R. S. 2000. Role-based authorization constraints specification. ACMTrans. Inform. Syst. Security 3, 4 (November), 207–226.

- CHEN, F. AND SANDHU, R. S. 1996. Constraints for role-based access control. In Proceedings of the 1st ACM Workshop on Role-based Access Control. ACM Press, 39–46. DAMIANOU, N. 2002. A policy framework for management of distributed.

- AKEHURST, D. AND KENT, S. 2002. A relational approach to defining transformations in a metamodel.InUML2002—The Unified Modeling Language. Model Engineering, Languages, Concepts, and Tools. 5th International Conference, Dresden, Germany, September/October 2002, Proceedings. LNCS, vol. 2460. Springer Verlag, 243–258.