EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

GÖMÜLÜ SİSTEMLER İÇİN TASARIM DESENLERİ KULLANARAK NESNEYE YÖNELİK, GERÇEK ZAMANLI BİR MİKROÇEKİRDEK TASARIMI

Kasım Sinan YILDIRIM

Bilgisayar Mühendisliği Anabilim Dalı

Bilim Dalı Kodu: 619.01.00

Sunuş Tarihi: 06.01.2006

Tez Danışmanı: Yrd. Doç. Dr. Aylin KANTARCI

Bornova-İZMİR

III

Kasım Sinan YILDIRIM tarafından yüksek lisans tezi olarak sunulan "**Gömülü Sistemler İçin Tasarım Desenleri Kullanarak Nesneye Yönelik, Gerçek Zamanlı Bir Mikroçekirdek Tasarımı**" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 06/01/2006 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri: İmza

Jüri Başkanı : Yrd. Doç. Dr. Aylin Kantarcı .....

Raportör Üye : Prof. Dr. Yasemin Topaloğlu .....

Üye : Prof. Dr. Turhan Tunalı .....

ÖZET

GÖMÜLÜ SİSTEMLER İÇİN TASARIM DESENLERİ KULLANARAK NESNEYE YÖNELİK, GERÇEK ZAMANLI BİR MİKROÇEKİRDEK TASARIMI

YILDIRIM, Kasım Sinan

Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Yrd. Doç. Dr. Aylin KANTARCI

Ocak 2006, 103 sayfa

Bu çalışmada gerçek zamanlı ve gömülü sistemler üzerinde çalışan işletim sistemleri ve bu işletim sistemlerini kullanan gömülü yazılımların yazılımsal özellikleri ve gereksinimleri incelenmiştir. Bu doğrultuda uygulama ihtiyaçlarına yönelik olarak şekillenebilen, taşınabilir, gerçek zamanlı, gömülü ve nesneye yönelik bir işletim sistemi olan eGe Gömülü İşletim Sistemi (eGİS); C++ dili, açık kaynak koda sahip ve ücretsiz yazılım araçları olan Gcc derleyicisi, Gdb hata ayıklayıcısı, Ld bağlayıcısı ve Bochs emülatörü kullanılarak gerçekleştirilmiştir.

eGİS mimarisindeki esnekliğin ve değişebilirliğin sağlanmasında kullanılan tasarım desenlerinin, sisteme kattığı faydalar ve yan etkileri tartışılmıştır. eGİS işletim sisteminin var olan diğer gerçek zamanlı ve gömülü işletim sistemleri ile kıyaslaması yapılmış, eGİS'in bu sistemlere olan benzerlikleri ve bu sistemlerden farkları ortaya koyulmuştur.

Anahtar Sözcükler: Gerçek zamanlı sistemler, Gömülü sistemler, Gerçek zamanlı işletim sistemleri, Nesneye yönelik programla, Tasarım

Desenleri, Mikroçekirdek, Monolitik Sistemler, C++, Gcc, Ld, Gdb, Bochs.

ABSTRACT

DESIGN OF AN OBJECT ORIENTED AND REAL-TIME MICROKERNEL FOR EMBEDDED SYSTEMS USING DESIGN PATTERNS

YILDIRIM, Kasım Sinan

MSc in Computer Engineering

Supervisor: Yrd. Doç. Dr. Aylin KANTARCI

January 2006, 103 pages

In this study, the software properties and requirements of applications and operating systems that work on the real-time and embedded systems are studied. Within this context, eGe Gömülü İşletim Sistemi (eGİS), which is a portable, real-time, embedded, object oriented and configurable operating system, is implemented by using C++ language, Gcc compiler, Gdb debugger, Ld linker and Bochs emulator; which are open source and free software development tools.

The advantages and disadvantages of the design patterns, which were used to improve the scalibility and configurability of the eGİS architecture, are discussed. The differences and similarities of eGİS with the other real-time and embedded operating systems are stated.

Keywords: Real-Time Systems, Embedded Systems, Real-Time Operating Systems, Object-Oriented Programming, Design Patterns, Microkernel, Monolithic Systems, C++, Gcc, Ld, Gdb, Bochs.

TEŞEKKÜR

Bu çalışmam süresince yardımları ve desteği ile her zaman arkamda olan çok değerli danışmanım Yrd. Doç. Dr. Aylin Kantarcı'ya teşekkür ederim.

Gerek lisans eğitimim gerek de yüksek lisans eğitimim süresince üzerimde ölçülemez emekleri olan bölümümüz öğretim üyelerinin hepsine teşekkürü bir borç bilirim.

Çalışma şevki ve gayreti ile bana örnek olan değerli dostum Arda Göknil'e, beni C dili ile tanıştıran Zekai Demirezen'e, yapıcı eleştirileri için Tayfun Elmas'a, değerli arkadaşlarım Mehmet Kış ve Emre Kalaycı'ya ve burda adını yazamadığım tüm arkadaşlarıma teşekkür ederim.

Tez süresince bana en büyük desteği sağlayan annem Sülbiye Yıldırım'a, babam Yasin Yıldırım'a, kardeşim Onur Yıldırım'a ve üzerimde çok büyük emeği olan anneannem Şengül Yılmaz'a sonsuz teşekkürlerimi sunuyorum.

Beni işletim sistemi kavramı ve alt seviyeli sistemler ile tanıştıran ve öğrencisi olmaktan onur duyduğum, aramızdan kısa bir süre önce

ayrılan Prof. Dr. Sinan Yılmaz'a en içten teşekkürlerimi sunuyor ve çalışmamı kendisinin anısına adıyorum.

XI

İÇİNDEKİLER

SAYFA

XIII

ŞEKİLLER DİZİNİ

Şekil **Sayfa**

1 GİRİŞ

Gömülü sistemler günümüzde giderek önem kazanan sistemlerdir. Palm cihazları ve mobil telefonlar gibi taşınabilir cihazlar, televizyon gibi büyük ve karmaşık sistemler gömülü yazılım içeren bazı sistemlerdir. Bu sistemler kısıtlı sistem kaynakları, gerçek zaman ve çok değişken uygulama gereksinimlerine sahip olmaları ile geleneksel yazılım sistemlerinden ayrılmaktadır.

Gömülü sistemler için gerçekleştirilen işletim sistemleri de, genel amaçlı işletim sistemlerinden farklı tasarım amaçlarına ve servislere sahiptirler. Verimli bir kaynak yönetimini ve gerçek zamanlı sistem algoritmaları içeren gömülü işletim sistemleri, giderek modern yazılım kavramlarının sunmuş olduğu avantajlara gereksinim duymaktadır.

Bu tez çalışması kapsamında, gerçek zamanlı gömülü uygulamaların gereksinimlerini karşılayacak olan **eGe Gömülü İşletim Sistemi** (*eGİS*), C++ dili ve açık kaynak koda sahip ücretsiz yazılım geliştirme araçları olan Gcc derleyicisi, Gdb hata ayıklayıcısı, Ld bağlayıcısı ve Bochs emülatörü kullanılarak gerçekleştirilmiştir. eGİS modern yazılım kavramları ve özellikle tasarım desenlerini taban alan mimarisi ile, var olan gömülü işletim sistemlerinden ayrılmaktadır. Ayrıca eGİS bir uygulamaya yönelik işletim sistemidir; uygulama ihtiyaçlarına göre değiştirilebilir ve biçimlendirilebilirdir.

Tamamiyle Türkçe gerçekleştirime sahip olan eGİS, gerçek anlamda ticari ürünlerde ve gömülü gerçek zamanlı sistemlerde kullanılabilir bir şekilde tasarlanmış ve geliştirilmiştir.

1.1 İlgili Çalışmalar

eGİS işletim sistemi, *Linux* ve *Windows* gibi genel amaçlı işletim sistemlerinden hem mimarisel hem de amaç olarak ayrılmaktadır. *Linux* ve *Windows* gibi masaüstü uygulamalarına yönelik olan işletim

sistemleri, gerçek zaman kısıtları düşünülerek gerçekleştirilmiş işletim sistemleri değildir. İçerdikleri süreç yönetim algoritmaları gerçek zamanlı sistemler için uygun değildir (Tanenbaum, 2001).

Ayrıca hem *Windows* hem de *Linux* monolitik bir modüler yapıya sahiptirler. İşletim sisteminin genişlemesi ve eklentileri *Linux*'te dinamik olarak eklenebilen modüllerle, *Windows*'da ise DLL'ler ile yapılmaktadır. Windows'da bütün kullanıcı arayüzü işletim sistemine gömülmüştür. Her iki işletim sisteminde ise bellek problemi olmayan çok geniş sistemler için tasarlanmıştır. eGİS ise gömülü bir mikroçekirdektir. Gerçek zaman kısıtlarını göz önüne alan yönetim algoritmalarını içermektedir.

uCOS-II işletim sistemi, gömülü gerçek zamanlı sistemler için gerçekleştirilmiş ticari bir işletim sistemidir. C dili ile gerçeklenen *uCOS-II*, bir monolitik yapıya sahiptir. Tahmin edilebilir algoritmalara sahip olan *uCOS-II*, modern yazılım kavramları göz önüne alınarak gerçekleştirilmiş bir işletim sistemi değildir. Öncelik tabanlı kesilebilir bir süreç yönetim algoritması kullanan *uCOS-II*, bu algoritmanın değişimi için bir mekanizma sağlamamaktadır. Bu algoritmanın değişimi, işletim sisteminin önemli bir bölümünün yeniden yazılmasını gerektirir. Küçük bir kod büyüklüğüne sahip olan *uCOS-II*, oldukça iyi sistem başarımına sahiptir. Ancak uygulamanın ihtiyaçlarına göre şekillenebilen bir sistem olmadığı için, değişimlere duyarlı değildir (Labrosse, 2003).

eGİS *uCOS-II* ile kıyaslandığında, daha düşük fakat kabul edilebilir bir başarıma sahip olsa da, daha yüksek bir değişebilirliğe sahiptir. Mikroçekirdek mimarisi sayesinde eGİS, *uCOS-II*'nin sahip olmadığı genişleyebilirlik ve donanım bağımsızlık özelliklerini içermektedir.

Nucleus gömülü gerçek zamanlı işletim sistemi de, uCOS-II gibi ticari bir işletim sistemidir. *Nucleus* işletim sistemi de C dili ile gerçekleştirilmiş, moduler yapıya sahip bir mimariye sahiptir. Ancak *Nucleus* da *uCOS-II* gibi modern yazılım kavramları göz önüne alınarak gerçekleştirilmiş bir işletim sistemi değildir (Accelerated Technology, 2001).

Nesneye yönelik tasarlanmamış olan *Nucleus* işletim sistemi, değişimlere eGİS gibi duyarlı değildir. *Nucleus* işletim sistemi de, belirli bir süreç yönetim algoritmasına bağlıdır. Bu algoritmanın değişebilirliği sistem içerisinde sağlanmamıştır. İşletim sisteminin donanım bağımsızlığı da bir ek katman ile sağlanmıştır. Bu ek katman eGİS sistemindeki köprü deseni ile kıyaslandığında çok esnek değildir.

eCOS işletim sistemi, C++ dili kullanılarak nesneye yönelik olarak geliştirilmiş açık kaynak koda sahip bir işletim sistemidir. *eCOS*'un ana amacı uygulama ihtiyaçlarına göre şekillenebilen bir işletim sistemi mimarisidir. Ancak *eCOS* düzenli bir nesneye yönelik ayrıştırım içermemekte ve genellikle sınıf kalıtımına dayanmaktadır. *eCOS*'un sınıf hiyerarşisi ve mimarisi, tasarım deseni tabanlı değildir. Bir çok mimariye taşınmış olan ve birçok bileşen içeren *eCOS*'un nesneye yönelik mimarisi katmanlı, bir modüler yapıya benzemektedir. eGİS ise mikroçekirdek mimarisini tasarım desenlerinin getirdiği esneklik ile değişimlere açık hale getirmiştir (Red Hat, Inc, 2003).

PURE işletim sistemi gömülü sistemler için geliştirilmiş ve uygulamaya yönelik bir işletim sistemi olmayı amaçlayan bir işletim sistemidir. *PURE* bir sınıf hiyerarşisi ve sınıf kalıtımına dayalı sistem mimarisine sahiptir. eGİS ile kıyaslandığında, çok geniş ve karmaşık bir sınıf hiyerarşisine dayanır. eGİS sistemi ise arayüz kalıtımına dayanan bir işletim sistemidir (Haack et al.; Hartig et al., 1997; Beuche et al., 1999).

CHORUS bir mikroçekirdek mimarisine sahip işletim sistemidir. C++ dili ile gerçekleştirilmiştir ve bir sınıf kalıtımı ve hiyerarşisine dayanmaktadır. Ayrıca *CHORUS* bileşenleri çok büyük ve çok da değiştirilebilir ve uygulama amaçlarına göre değiştirilebilir değildir. eGİS ise küçük ve bağımsız bileşenlere sahiptir ve arayüz kalıtımına dayanmaktadır. *CHORUS* tasarım deseni tabanlı bir sistem değildir (Levchenko; Fröhlich, 2001; Bricker et al., 1991).

TINYOS işletim sistemi bileşen tabanlı bir işletim sistemidir. Tasarım desenleri bu işletim sistemi içerisinde de yer almaktadır. Ancak TINYOS işletim sisteminde kullanılan desenler bileşen tabanlı desenlerdir ve bu bileşenlerin derleme anında sistem ile birleştirilmesi işlemlerini yerine getirmek için kullanılırlar. İşletim sistemini kullanacak

olan uygulamanın gereksinimlerine göre seçilen TINYOS işletim sistemi bileşenleri, tasarım desenlerinin derleme anında sunmuş olduğu değişebilirlik özelliği ile sistemle birleştirilmektedir. TINYOS, sisteme özgü nesC dili ile, var olan bileşenlerin uygulama gereksinimlerine göre birbirlerine bağlanması için bir tasarım deseni mimarisine sahiptir (Gay et al., 2004).

EPOS işletim sisteminin ana amacı işletim sistemini kolaylıla uygulama programının istediği yapıya kavuşturmaktır. Bu işletim sisteminde de arayüz ve gerçekleştirim ayırımından ve yeniden kullanılabilirlikten söz edilmektedir. Ancak *EPOS* tasarım desenleri tabanlı bir sistem değildir. *EPOS* ilgiye yönelik kavramları da göz önüne almıştır ve bu noktada eGİS'ten ayrılmaktadır (Fröhlich and Schröder-Preikschat; Fröhlich, 2001).

1.2 İçerik

Tez 6 bölümden oluşmaktadır. Tezin 2. bölümünde gerçek zamanlı sistemlerin temel özellikleri belirtilmiş ve gerçek zamanlı işletim sistemi kavramları açıklanmıştır.

Tezin 3. bölümünde tasarım deseni kavramları açıklanmış ve gerçek zamanlı gömülü sistemlerin yazılımsal özellikleri ve gereksinimleri anlatılmıştır. Tasarım desenlerinin gerçek zamanlı gömülü sistemlerde nerelere uygulanabileceği belirtilmiştir.

Tezin 4. bölümünde eGİS işletim sisteminin tasarım amaçları ve modern gömülü işletim sistemlerinin sahip olması gereken önemli özellikler belirtilmiştir.

Tezin 5. bölümünde eGİS işletim sisteminin mimarisi açıklanmış; tasarım desenlerinin gömülü bir işletim sistemi olan eGİS içerisinde kullanımları gösterilmiş ve desenlerin eGİS sistemine kazandırdığı avantajlar ve getirmiş olduğu ek yükler tartışılmıştır.

Tezin 6. bölümünde eGİS işletim sisteminin nasıl gerçekleştirildiği anlatılmış ve gerçekleştirimde kullanılan yazılım araçları tanıtılmıştır. Ayrıca tez çalışmasının sonuçları tartışılmış ve gelecek çalışmalarda gerçekleştirilebilecek noktalar ortaya konulmuştur.

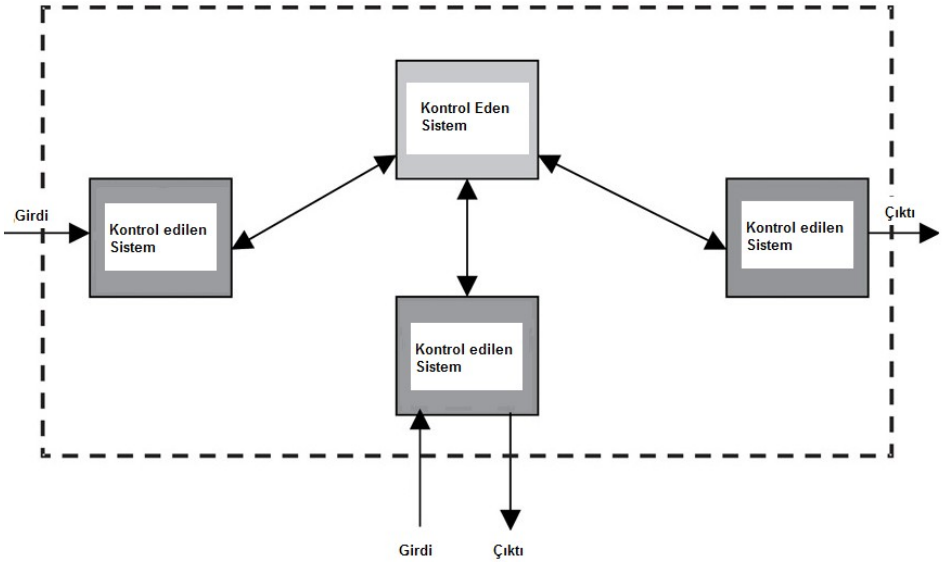
2 GERÇEK ZAMANLI SİSTEMLER

Bu bölümde gerçek zamanlı sistem kavramları anlatılmış ve gerçek zamanlı sistemlerin yazılımsal özellikleri incelenmiştir. Bölüm 2.1'de temel gerçek zamanlı sistemlerin özellikleri ortaya koyulmuş ve geleneksel sistemlerden ayrıldığı noktalar anlatılmıştır. Gömülü sistemlerin gerçek zamanlı sistemler içerisinde nasıl sınıflandırılabileceği ve temel özellikleri incelenmiştir.

2.1 Gerçek Zamanlı Sistemler

Gerçek zamanlı sistemler mantıksal ve işlevsel doğruluğun yanı sıra zamansal doğruluğa da sahip olması gereken sistemlerdir. Bilindiği gibi gerçek zamanlı sistemler çok keskin zaman kesin kısıtlarına sahip olan ve daha esnek zaman kısıtlarına sahip sistemler olmak üzere iki kısımda incelenebilirler. Çoğu gerçek zamanlı sistem, bu iki tip sistemin bir birleşimi olarak düşünülebilir. Günümüzde gerçek zamanlı sistemlerin çoğu gömülü sistemlerdir. Gömülü gerçek zamanlı sistemlere örnek olarak çamaşır makineleri, televizyonlar, mikrodalga fırınlar ve yazıcılar gibi cihazlar verilebilir.

Şekil 2.1, gerçek zamanlı sistemlerin genel yapısını göstermektedir. Bir gerçek zamanlı sistem bir *kontrol eden sistem* ve en az bir adet *kontrol edilen sistem* içermektedir. Kontrol eden sistem, kontrol edilen sistemlerle periyodik ya da periyodik olmayan bir şekilde iki yönlü iletişim kurabilir. Genel sistem, dış dünyadan gelen girdilere, içsel sistemler arasındaki iletişimler sonucunda çıktılar üretmektedir (Li ve Yao, 2003).



Şekil 2.1 Gerçek Zamanlı Sistemlerin Genel Yapısı

2.2 Gömülü Sistemler

Adanmış sistemlerin bir sınıfı olan gömülü sistemler, özel amaçlar için tasarlanmış sistemlerdir. Çoğu gömülü sistem, işlemlerini güvenilir ve tahmin edilebilir bir şekilde yapmalıdır. İşlemlerini gerçek zaman kısıtları içerisinde yapmak zorunda olan gömülü sistemlere, gerçek zamanlı gömülü sistemler denmektedir. Dolayısıyla her gömülü sistem gerçek zamanlı olmayabilir.

Gömülü sistemler birbirlerine çok bağımlı donanımsal ve yazılımsal sistemlerin birleştirilmesi ile özel bir işlevi yerine getiren işleme sistemleridir.

Masaüstü bilgisayarları için geliştirilmiş olan işlemcilerin çoğu genel amaçlıdır. Genellikle karmaşık bir tasarıma sahip olan bu işlemciler, geniş bir yelpazeyi kapsayan fonksiyonlar içermektedir. Örneğin çoğu masaüstü işlemci, bir bellek yönetim birimi (*MMU)*

içermektedir. Bu da işlemcilerin yapısını ve tasarımını karmaşıklaştırmıştır ve işlemcilerin maliyetini arttırmıştır. Gömülü sistemler ise gömülü işlemcilere gereksinim duyarlar. Genel amaçlı işlemcilerin aksine, gömülü sistemlere özgü işlemciler, özel bir işlevi yerine getirmek üzere tasarlanmışlardır. Bu işlemciler üzerinde çalışacak uygulamalar, özel bir işlemi yerine getirecek ve belirli bir sınıfa ait olan uygulamalardır. Örneğin bir PDA sisteminde kullanılacak işlemcinin işleme gücü, büyüklüğü ve güç tüketimi özelleşmiş gereksinimlerdir.

Gömülü sistemlerde, donanım ve yazılım birbirine paralel olarak gerçekleştirilmektedir. Donanım ve yazılım tasarımcıları, birbirleri ile sürekli iletişim halindedirler ve bu iletişimin sonucunda sistemin hangi özelliklerinin yazılım, hangi özelliklerinin donanım yoluyla gerçekleştirilebileceği belirlenir. Eğer donanımı karmaşıklaştıracak ve daha pahalı hale getirilebilecek bir işlev, gerçek zaman kısıtlarına zarar vermeden yazılım tarafından yapılabiliniyorsa, bu kısım yazılım kapsamına alınır. Bu durumun tersi de geçerlidir. Yazılımın başarımı çok etkileyecek noktaları donanım tarafından gerçekleştirilebilmektedir. Yazılım ve donanımın bu işbirliğine **yazılım - donanım işbirlikçi tasarımı** denmektedir (Li ve Yao, 2003).

Gömülü sistemlerde yazılım **çapraz geliştirme ortamı** yöntemi ile geliştirilmektedir. Yazılım *ana ortamda* geliştirilir. Ana ortam yazılım geliştirildiği donanım, işletim sistemi ve yazılım geliştirme araçlarını içermektedir. Bu şekilde geliştirilen yazılım, ana ortamdan *amaç ortama* taşınır. Ana ortamdan amaç ortama taşınma işleminde esas önem **çapraz derleyici** sistem aracındadır. Çapraz derleyici, bir işlemci mimarisi üzerinde çalışan ve değişik işlemci mimarileri için kod üretebilen sistem yazılımıdır.

2.3 Arkaplan / Önplan Sistemler

Genellikle küçük ve az karmaşıklığa sahip gerçek zamanlı sistemler arkaplan ve önplan mekanizmasını kullanarak gerçekleştirilirler.

Bu tip sistemlerde sadece bir uygulama vardır ve uygulama bir sonsuz döngü içerisinde dönen ve sırası ile ilgili fonksiyonları çağıran bir kod bloğu içermektedir. Bu *arkaplan* sistemi oluşturur. Kesmeler ise asenkron olarak işlenmektedir ve kesme olayları ise sistemin *önplanını* oluşturmaktadır.

Arkaplan/önplan sistemleri iyi tepki sürelerine sahiptirler çünkü bu sistemlerin çalışması ve işlemci yönetimi üzerinde çalışılan donanıma bağlıdır. Sistem donanımın sunmuş olduğu kesme mekanizması ve bu kesmeleri gerçek zamanlı olarak işleyen önplan sistemi işbirliği ile gerçek zaman kısıtlarını yerine getirir. Sistem büyük bir döngü içerisinde çalıştığı için, bu tip sistemlerde döngünün ne kadar süreceği ve hangi işlem adımlarından geçileceği çok iyi belirlenmelidir (Labrosse, 2003).

Arkaplan/önplan sistemlerinin önemli bir dezavantajı, bu tip sistemlerin kod değişimlerinden çok etkileniyor olmalarıdır. Eklenen veya çıkarılan her kod parçası sistemdeki zamanlamaların değişmesine neden olmaktadır. Ayrıca sistem donanımsal hatalara aşırı duyarlıdır. Arkaplan/önplan sistemler, sistem ihtiyaçları arttıkça ve büyüdükçe geliştirilmesi ve hatalardan arındırılması zor sistemlerdir (Laplante, 1997).

2.4 Gerçek Zamanlı İşletim Sistemleri

Gerçek zamanlı işletim sistemi, arkaplan/önplan gibi basit bir şekilde gerçekleştirilemeyecek gerçek zamanlı uygulamaların geliştirilebilmesi için uygun bir yazılım ortamı sağlamaktadır. Tüm gömülü sistemler bir gerçek zamanlı işletim sistemi kullanılarak hazırlanmazlar. Ancak karmaşık ve geniş bir gömülü yazılımın daha

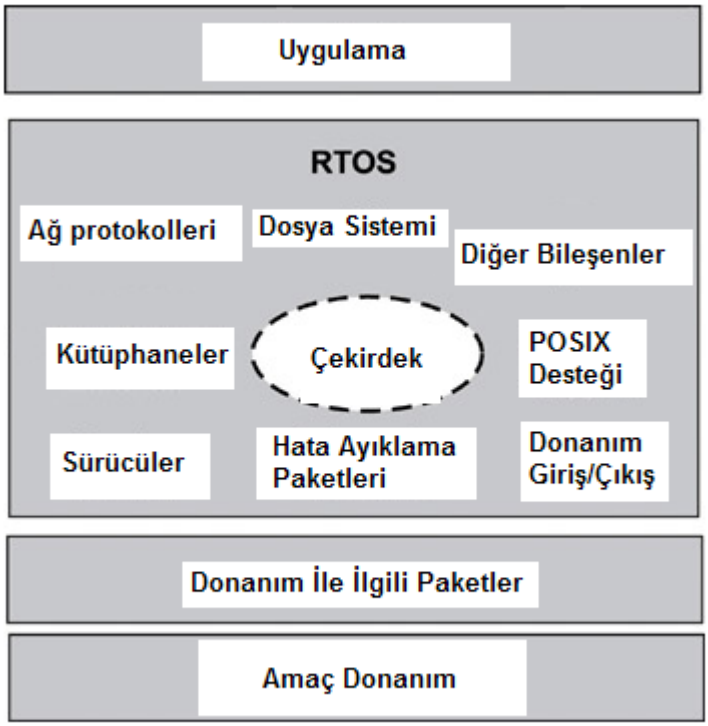
kolay ve zahmetsiz geliştirilebilmesi için, gerçek zamanlı işletim sistemlerine ihtiyaç vardır.

Gerçek zamanlı işletim sisteminin merkezinde *çekirdek* vardır. Birden fazla sürecin bulunduğu sistemlerde, süreçlerin yönetimini ve süreçler arası iletişimi sağlayan temel birim çekirdektir. Çekirdek işlemcinin uygulamanın gereksinimlerine göre daha verimli kullanılmasını sağlamaktadır. Çekirdek sunmuş olduğu *semafor*, *kilit*, *mesaj kutuları*, *kuyruklar*, *zamanlayıcı yönetimi* gibi servislerle, sistemin daha kolay ve verimli işleyişini sağlamaktadır. Gerçek zamanlı bir çekirdek, gerçek zamanlı uygulamaların daha kolay geliştirilmesini ve genişleyebilmesini sağlamaktadır.

Şekil 2.2 bir gerçek zamanlı işletim sisteminin (*Real Time Operating System – RTOS*) yapısı ve sistemdeki yerini göstermektedir. Bazı gerçek zamanlı işletim sistemleri sadece çekirdek birimini içermektedir. Çekirdek, sadece süreç ve kaynak yönetimi gibi işlevlere sahip olabilir. Bazı sistemlerde ise işletim sistemi daha karmaşık ve genişlemiş olabilir. Bir gerçek zamanlı işletim sistemi:

- çekirdek
- dosya sistemi (kalıcı bellek dosya sistemi gibi)
- ağ protokolleri ve TCP/IP bileşenleri
- sürücüler
- genel programlama kütüphaneleri
- donanımsal paketler
- POSIX gibi arayüz destek paketleri
- hata ayıklama gibi işlevleri sağlayan paketler

ve benzeri sistem bileşenlerini içerebilmektedir (Li ve Yao, 2003).

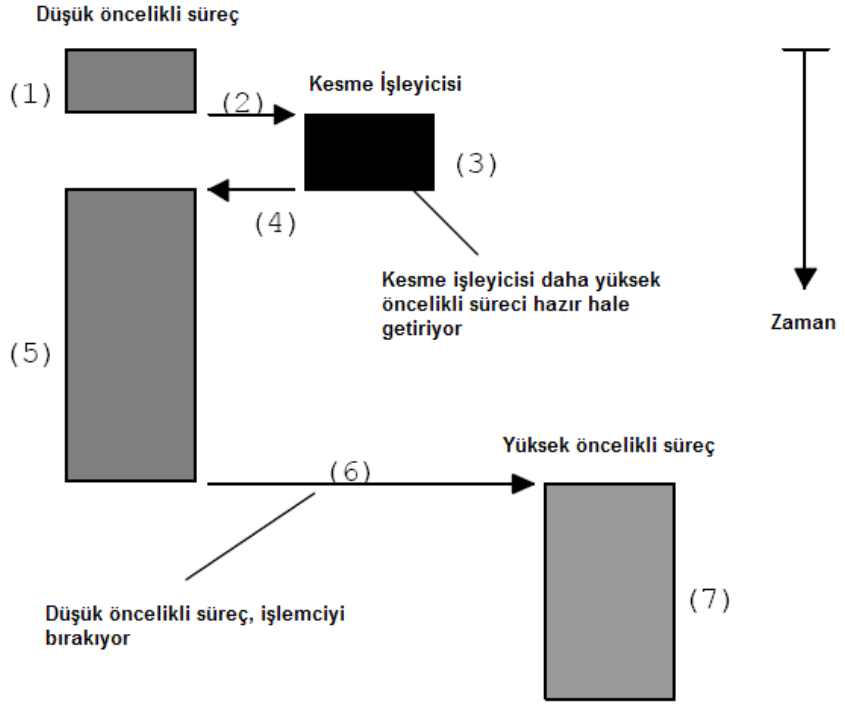


Şekil 2.2 Gerçek Zamanlı Bir İşletim Sistemi

Gerçek zamanlı bir işletim sistemi, **kesilebilir** ya da **kesilemeyen** işleyişe sahip bir çekirdek içerebilir.

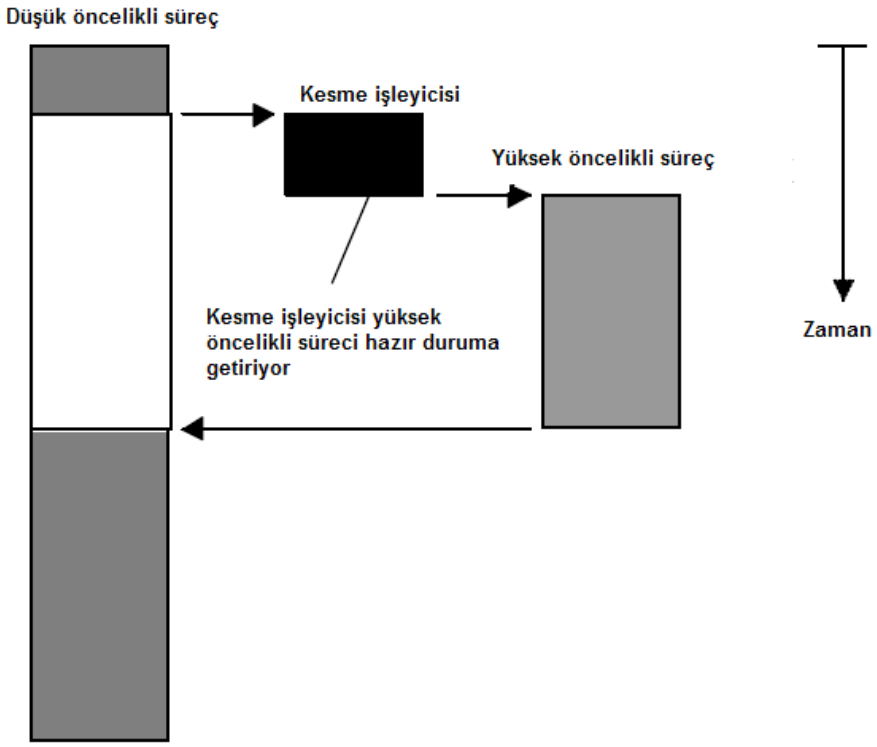
Kesilebilir işleyişe sahip çekirdeklerde, çalışan süreçlerin çalışması kesilebilir. Daha yüksek önceliğe sahip bir süreç, süreç yöneticisi tarafından hazır hale getirildiğinde, daha düşük önceliğe sahip ve o an çalışan sürecin çalışması kesilerek; yüksek öncelikli sürece çalışma hakkı verilir.

Kesilemeyen işleyişe sahip çekirdek, kesilebilen bir çekirdekle kıyaslandığında daha kötü bir sistem tepkisine sahiptir. Şekil 1.3, kesilemeyen işleyişe sahip bir çekirdeğin iki süreci nasıl yönettiğini göstermektedir. Düşük öncelikli süreç, bir kesme olayı nedeniyle çalışmasına ara vermektedir (1). Kesme işleyicisi işlemciyi ele geçirir ve daha yüksek önceliğe sahip olan diğer süreci hazır durumuna sokar (2-3). Kesme işleyicisi, işlemciyi bırakır ve daha düşük önceliğe sahip süreç çalışmaya kaldığı yerden devam eder (4-5). Düşük öncelikli süreç işlemciyi bırakır ve hazır durumda bekleyen daha yüksek öncelikli süreç çalışmaya başlar (6-7). Görüldüğü gibi yüksek önceliğe sahip süreç, hazır durumda olduğu halde, kendisinden daha düşük öncelikli süreci beklemek zorundadır (Labrosse, 2003).



Şekil 2.3 Kesilemeyen İşleyişe Sahip Bir Çekirdek

Şekil 2.4, kesilebilen işleyişe sahip bir çekirdeğin iki süreci nasıl yönettiğini göstermektedir. Yüksek öncelikli süreç, hazır hale gelir gelmez çalışmaya başlamaktadır. Düşük öncelikli süreç ancak yüksek öncelikli süreç işlemciyi bıraktıktan sonra, çalışmaya kaldığı yerden devam edebilmektedir (Labrosse, 2003).



Şekil 2.4 Kesilebilen İşleyişe Sahip Bir Çekirdek

Li ve Yao'ya (2003) göre, gerçek zamanlı işletim sistemleri, genel amaçlı işletim sistemleri ile şu benzerliklere sahiptirler:

- Çoklu programlamaya olanak sağlarlar
- Yazılımsal ve donanımsal kaynakları yönetirler
- Donanımsal ortamı uygulamalardan soyutlarlar

Gerçek zamanlı işletim sistemlerinin genel amaçlı işletim sistemlerinden ayrıldığı noktalar is şunlardır:

- Gömülü uygulamalar için daha güvenilirlerdir.
- Uygulama gereksinimlerine göre genişleyebilir ya da daralabilirler.
- Daha iyi başarıma sahiptirler.
- Daha az bellek gereksinimlerine sahiptirler.
- İşlemci yönetim algoritmaları gerçek zamanlı sistemlere yöneliktir.
- Değişik donanım ortamlarına daha kolay taşınabilme özelliklerine sahiptirler.

2.5 Gerçek Zamanlı İşletim Sistemlerinin Kıyaslanması

Gerçek zamanlı işletim sistemleri şu özelliklerine göre birbirleri ile kıyaslanabilirler (Li ve Yao, 2003; Laplante, 1997):

- *Nasıl bir çekirdeğe sahip olunduğu*: Arkaplan/Önplan, kesilemeyen ya da kesilebilir bir çekirdek

- *Kesme gecikme süreleri*: Kesmelerin kapalı olduğu ve kesmeye tepki verilmeye başlanıncaya kadar geçen süre

- *Kesme dönüş süresi*: Kesmeden dönüp kalınan yerdeki ilk kodu işlemeye başlanıncaya kadar geçen süre

- *Zamanlayıcı yönetimi*: Sistem içerisinde kurulan zamansal işlemlerin ne kadar doğrulukta işlendiği, zamansal tepkilerin ne doğrulukta verildiği

- *Bellek gereksinimleri*: Sistemin minimum bellek gereksinimi ve belleği ne kadar verimli kullandığı

- *Güvenilirlik*: Hatalı durumlarda sistemin davranışı, güvenli bir duruma gelip gelemediği

- *Tahmin edilebilirlik*: İşlemlerin sistemde işlenmeleri için gereken maksimum çalışma sürelerinin belirlenebiliyor olması

- *Genişleyebilirlik*: Sistemin uygulama ihtiyaçlarına göre şekillenebilmesi, yeni bileşenlerin kolaylıkla eklenip çıkarabilinmesi

Uygulamanın kapsamına ve ihtiyaçlarına göre, işletim sistemlerinin yukarıda belirtilen parametrelere göre kıyaslaması yapılabilinir. Seçilen işletim sistemi, uygulama gereksinimlerine göre şekillendirilebilir. Gerçek zamanlı bir uygulamanın gereksinimlerini basit bir arkaplan/önplan sistemi de, birçok bileşeni içeren geniş bir işletim sistemi mimarisi de karşılayabilir.

3 TASARIM DESENLERİ

Desenler mimarisel bir kavram olarak ilk defa Christopher Alexander tarafından ortaya atılmıştır. Bir mimar olan Alexander, binaların yapısal özelliklerinin yeniden kullanılabilir desenler olduğunu ileri sürmüştür ve kaliteli binaların daha önceden kullanılmış desen parçalarının seçilip, bir desen diliyle bir araya getirilerek oluşturulabildiğini göstermiştir. Alexander'ın bu çalışması, yazılım tasarımcılarını da etkilemiştir. 1990'ların başında, yazılım geliştiricileri Alexander'ın mimarisel desen kavramlarının yazılıma da uygulanıp uygulanamayacağını tartışmaya başlamışlardır.

OOPSLA '91, yazılım mimarisi üzerine yapılmış ortak çalışmaları içeren ve yazılım mimarları için temel bir el kitabı oluşturmayı amaçlayan bir çalıştaydı. Bu çalıştay ile yazılım tecrübesi ve uzmanlığının dökümante edilmesi gerekliliği tartışılmaya başlanmıştır.

1990'lı yıllarda çoğu kişi tasarım desenleri üzerine çalışmakta olsa da, tasarım desenleri 1994 yılında Gamma, Helm, Johnson ve Vlissides tarafından yayımlanan *Design Patterns: Elements of Reusable Object-Oriented Software* isimli kitap ile popüler hale gelmiştir (Gamma et al.,1994).

3.1 Tasarım Deseni Kavramı

Bir yazılım mühendisliği kavramı olarak tasarım deseni, yazılım tasarımında ortaya çıkan benzer problemlerin genel çözümleridirler. Tasarım desenleri genel bir tasarım şablonudur. Tamamiyle bitmiş ve doğrudan kodlanabilen tasarımlar değillerdir. Ortaya koymuş oldukları genel çözüm yolları uygulamaya göre şekillendirilerek ve değiştirilerek kullanılırlar.

Tasarım desenleri nesneler ya da sınıflar arasındaki ilişkileri ve sorumlulukları belirtirler. Uygulamaya yönelik gösterimleri ve ilişkileri barındırmazlar, problemin genel çözüm yoluna ilişkin tasarımsal

gösterimleri içerirler. Algoritmik ifadeler de tasarım desenleri içerisinde yer almazlar. Çünkü algoritmalar işlemsel problemleri çözerler, tasarım problemlerinin bir parçası değillerdir; gerçekleştirim tabanlıdırlar (Gamma et al., 1994).

Tasarım desenleri genel tasarım çözümlerinin probleme özel olmayacak şekilde dökümante edilmiş halleridir. Bu dökümanlar geliştiricilerin yazılım ilişkilerini iyi tanımlanmış ve anlaşılmış yöntemlerle kurmalarını sağlar. Ayrıca desenler kodun okunabilirliğini ve düzenliliğini sağlarlar, tasarımcılar için ortak bir dil oluşturmaktadırlar.

Tasarım desenleri kullanılmış ve başarıya ulaşmış tasarım ve mimarilerin yeniden kullanılabilmesini sağlarlar. Bu sayede üretilen sistemin de yeniden kullanılabilirliği artmaktadır. Zaten kullanılmış ve başarıya ulaşmış tasarımları kullanmak, uygulamaya özel problemlerin çözümü için iyi bir başlangıç yapmayı sağlar ve düşülebilecek potansiyel hataları engeller. Geliştiriciler ve tasarımcılar zaten çözülmüş problemlerle tekrar tekrar uğraşmazlar. Bunun yerine başarıya ulaşmış iyi tasarımları yeniden kullanarak kendi sistemleri için özelleştirirler (Douglass, 2002).

Tasarım desenleri yeni gereksinimleri önceden sezerek sistemi genişleyebilir olarak tasarlamak için kullanılırlar. Sistem değişimlere ve genişlemeye uygun tasarlanmadıysa, ilerde ortaya çıkan gereksinimler sistemin belki de baştan yazılmasına ve test edilmesine neden olacaktır. Tasarım desenleri, sistemin daha rahat, hızlı ve sağlam bir değişebilirlik özelliğine sahip olmasını sağlarlar (Gamma et al., 1994).

Tasarım desenleri, probleme ve probleme özgü tasarıma yüksek seviye bir bakış açısı ile yaklaşılmasını sağlarlar. Bu sayede probleme özgü detaylarla, zamanı geldiğinde ilgilenilir, tasarımın başlangıcında detaylarla zaman kaybedilmesi engellenmiş olur. Bu sayede sistem daha sağlıklı bir biçimde gelişir, detayların sistemde yaratabileceği bağımlılıklar mümkün olduğunca azaltılır (Gamma et al., 1994).

Tasarım desenlerinin bir yazılım sistemine katabileceği özellikler aşağıdaki gibi özetlenebilir (Shalloway ve Trott, 2000):

- Benzer problemlerde var olan ve kalitesi kanıtlanmış çözümleri ve tasarımları kullanmak
- Yazılım geliştirme takımları arasında ortak bir terminoloji ve dil kurmak
- Problem çözümleri için daha yüksek seviyeden bir bakış açısı ile yaklaşmayı sağlamak
- Sadece doğru sonuçlar üreten bir sisteme değilde, aynı zamanda doğru ve sağlam bir tasarıma da sahip olmak
- Kodun değişebilirliğini ve yeniden kullanılabilirliğini sağlamak

3.2 Tasarım Deseni Dökümanı

Tasarım deseni dökümanı desen hakkında yeterli derecede bilgi veren, desenin hangi kapsamda kullanıldığını ortaya koyan ve önerilen çözüm yolunun sunulduğu dökümandır. Desenlerin dökümante edilmesinde kullanılan genel şablon şu bölümleri içermektedir (Gamma et al., 1994):

- *Desen İsmi ve Sınıflandırması*: Her desenin kendini iyi bir şekilde ifade eden ve kendisi işaret edilirken kullanılan bir ismi vardır. Ayrıca desenin, hangi sınıfa girdiği burada belirtilmektedir.
- *Amaç*: Bu bölüm desenin amacını ve kullanım nedenini belirtir. Desenin işaret ettiği problemi ortaya koyar.

- *Desenin Diğer İsimleri*: Bir desen birden fazla isme sahip olabilir. Desenin diğer isimleri bu bölümde belirtilir.

- *Motivasyon*: Bu bölüm problemi içeren bir senaryo ve bu senaryo dahilinde problemin nasıl çözüldüğünü gösterir.

- *Uygulanabilirlik*: Bu bölüm desenin hangi durumlarda faydalı olabileceğini belirtir. Desenin net kullanım amacı burada belirtilir.

- *Yapı*: Desenin grafiksel gösterimi burada belirtilir. Sınıf diyagramları ve ilişkisel gösterimlere burada yer verilir.

- *Katılımcılar*: Desende kullanılan sınıf ve nesnelere ve bunların tasarımdaki rollerine burada yer verilir.

- *İşbirlikleri*: Desendeki sınıflar ve nesnelerin birbirleri ile olan ilişkilerine burada yer verilir.

- *Sonuçlar*: Bu bölüm desen kullanımının sonuçlarını, yan etkilerini ve dezavantajlarını belirtir.

- *Gerçekleştirim*: Bu bölüm desenin gerçekleştiriminde kullanılacak teknikleri ve yolları ortaya koymaktadır.

- *Örnek kod*: Örnek bir programlama dilinde desenin uygulanışı gösterilir.

- *Bilinen Kullanımlar*: Bu bölüm desenin var olan gerçek sistemlerde kullanımına örnekler verir.

- *İlgili Desenler*: Bu desenle bağlantısı ve benzerliği olan diğer desenlere burada yer verilmektedir.

3.3 Desenlerinin Sınıflandırılması

Desenler hangi tasarımsal sorunu çözdüklerine göre sınıflandırılabilirler. Bazı desenler bir yazılım sistemini alt sistemlere ayırmada kullanılırken, diğer desenler alt sistemler ve bileşenlerin tasarımı ve bunlar arasındaki ilişkileri ortaya koymaktadırlar. Ayrıca bazı desenler belirli bir programlama diline yönelik çözümleri içermektedirler. Desenleri guruplandırabileceğimiz sınıflardan bazıları şunlardır (Buschmann et al., 1996):

- *Mimarisel Desenler* : Sistemin alt sistemlere nasıl ayrıştırılması gerektiğini ve alt sistemler arasındaki ilişkilerin nasıl olması gerektiğini belirtirler.

- *Tasarım Desenleri* : Alt sistemler ve bileşenlerin yapısı ve ilişkilerini ortaya koyarlar. Belirli bir kapsamda, genel bir tasarım probleminin çözümü için birbirleri ile iletişimde bulunan bileşenlerin genel yapısını tanımlarlar.

- *Deyimler* : Belirli bir programlama diline özgü alt seviye desenlerdir. Programlama dilinin sunmuş olduğu özellikler ile, bir yazılım bileşeni ve diğer bileşenlerle olan ilişkisinin nasıl gerçekleştirileceğini tanımlarlar.

Tasarım desenleri ise çözmüş oldukları tasarım problemine göre üç sınıfta incelenebilirler: (Gamma et al., 1994).

- *Yaratımsal Tasarım Desenleri* : Sistemdeki nesnelerin somut sınıfları belirtilerek nasıl yaratılması gerektiğini belirtirler.

- *Yapısal Tasarım Desenleri*: Nesneler ve sınıflar arasındaki kalıtım ve içerme ilişkilerini belirtirler

- *Davranışsal Tasarım Desenleri* : Nesnelerin sorumluluklarını aralarındaki etkileşimleri belirtirler.

3.4 Tasarım Desenlerinin Çözdüğü Temel Problemler

Tasarım desenlerinin çözdüğü bazı temel problemler aşağıda sıralanmıştır. Bu noktalar dikkate alınmadan geliştirilen yazılımın, çoğu zaman yeniden tasarlanıp yazılmasına gerek duyulur (Gamma et al., 1994).

- *Nesneleri sınıfları açıkça belirterek yaratmak*: Nesneleri yaratırken sınıf isimlerini açıkça belirtmek, nesneleri belirli bir gerçekleştirime bağlamaktadır. Bu da ilerideki değişimlere karşı sistemi güçsüzleştirir.

- *Belirli operasyonlara bağımlılık*: Belirli bir işlem için istekleri açık isimlerle belirtmek, istekleri gerçekleştirimlere bağlar. Bu isteklerin daha genel belirtilmesi, ilerideki değişimlere karşı sistemi daha açık yapar.

- *Yazılım ve donanım ortamlarına bağımlılık*: Kullanılan işletim sistemi ya da uygulama programlama arayüzüne bağımlı yazılmış yazılımlar, taşınması zor yazılımlardır. Bu bağımlılıklar ortadan kaldırılmalıdır.

- *Nesnelerin gösterimine ve gerçekleştirimlerine bağımlıkık*: İstemciler kullandıkları nesnelerin nasıl gösterildiğinden ve gerçekleştiriminden haberdar olmamalıdır. Çünkü nesne değişimleri sistemi olumsuz yönde etkiler.

- *Algoritmik bağımlılıklar*: Algoritmalar genellikle iyileştirilebilen, genişleyebilen ve değişebilen kısımlardır. Algoritmaya bağımlı olan nesne, algoritma değiştiğinde değişmek zorunda kalmaktadır. Sistemde bu durum engellenmelidir.

- *Nesnelerin birbirlerine bağımlılığı*: Birbirine bağımlı olan nesneleri yeniden kullanmak ve değiştirmek zordur. Sistem

mümkün olduğunca bağımsız nesneler üzerinden tasarlanmalıdır.

- *Kalıtım mekanizmasının getirdiği yan etkiler*: Nesnelerin kalıtım yolu ile genişletilmesi zordur ve bu durum sistemin yeniden kullanılabilirliğini azaltır. Bunun yerine nesne içerme mekanizması tercih edilmelidir.

- *Nesnelerin değişiminin zor olması*: Sistemde bulunan merkezi ve karmaşık bir sınıfı değiştirmek gerekebilir. Bu işlemlerin daha kolay ve sistemin genişleyebilirliğini düşünerek yapmak gerekir.

3.5 Bir Tasarım Deseni Nasıl Seçilmelidir

Tasarım desenleri bir sistem içerisinde kullanılırken, tasarım deseni dökümanı ve sistemin genel gereksinimleri göz önüne alınır ve analiz yapılır. Şu noktalar desenin seçiminde izlenebilir (Douglass, 2002):

- *Amaç bölümü taranır*: Her bir desenin amacı incelenerek desenin karşılaşılan probleme yönelik çözümü olup olmadığına bakılır.

- *Uygulanabilirlik bölümü incelenir*: Desenin uygulanmasının sistem için uygun olup olmadığı belirlenir. Karşılaşılan problemin mimarisel ya da daha detaylı bir problem olup olmadığına bakılır.

- *Desenlerin ilişkilerine bakılır*: Desenlerin birbirleri ile kurmuş oldukları ilişkiler ve bu ilişkiler sonucunda çözdükleri problemler incelenmelidir. Bunların, uygulanacak sistemle olan ilişkilerine bakılır.

- *Tasarımda değişebilen noktalar belirlenmelidir*: Sistemde değişebilecek noktalar belirlenmelidir. Değişebilen bu

noktalara hangi desenin uygun olabileceği incelenmelidir. Bu sayede sistem yeniden tasarlanmaya gerek kalmadan değişebilecektir.

- *Sistemdeki zamansal açıdan kritik olan noktalar belirlenmelidir*: Desenin getirdiği soyutlamaları ve başarım kaybını taşımayan sistem noktaları belirlenmelidir. Yeniden kullanılabilirlik, güvenilirlik, taşınabilirlik ve bellek kullanımı gibi servis parametreleri arasında bir karar verilmelidir.

3.6 Gerçek Zamanlı Sistemlerin Yazılımsal Gereksinimleri ve Tasarım Desenleri

Bilgisayar arastırmacıları uzun bir süre gömülü yazılıma olan ilgilerini düşük tutmuştur. Yazılımın küçük olması ve genellikle birleştirici dili ile yazılması; ayrıca sistemin donanım maliyetinin oldukça fazla ve önemli oluşu, yazılımın gömülü sistemlerdeki temel sorunu olmuştur. Yazılım mühendisliğinin sunduğu çözümler gömülü sistemler için çok pahalı görünmüştür. Yazılım mühendisliği araştırmalarının buraya uygulanabilir olmaması, buradaki yazılımsal araştırmaları ikinci plana atmıştır (Lee, 2000).

Donanımın daha ucuz hale gelmesi ve yazılımın daha büyük ve karmaşık bir hal alması nedeniyle gömülü sistemlerdeki yazılım araştırmaları yeniden popüler hale gelmiştir.

Günümüz gerçek zamanlı sistemleri büyük ve karmaşık yazılımlardır. Birbirinden farklı kısıtlara ve isteklere sahiptirler. Yeni mimariler ve değişen piyasa koşulları, gerçek zamanlı yazılım sistemlerinin tasarım mekanizmalarında da modern yazılım kavramlarının kullanılmasını gerektirmektedir. Artık gömülü sistemler birleştirici dili ile kolayca yazılabilen ve tasarlanabilen sistemler değildir.

Modern gömülü işletim sistemlerinin sahip olması gereken temel özellikler şu sekilde sıralanabilir (Gien, 1991) :

- İyi bir yazılım mimarisi
- Sistem bileşenlerinin yeniden kullanılabilir olması
- Dağıtıklık (Alt sistemlerin dağıtılabilir olarak tasarlanması)
- Ortamdan bağımsız olma ve değişik sistem mimarilerine ve donanımsal ortamlara taşınabilme
- Genişleyebilir ve isteğe göre değiştirilebilir alt sistemlere sahip olma
- Kaynakların verimli kullanılması
- Uygulama ihtiyaçlarına göre degişebilme

Gömülü işletim sistemlerinde esas tasarım yükü, sistemi alt sistemlere en düzgün biçimde ayırabilmek ve alt sistemlerin arayüzlerini en iyi ve genel şekilde belirlemektir. Alt sistem arayüzlerinin belirlenmesinde, çok iyi bir işletim sistemi problem uzayı analizi yapılmalıdır. Gömülü işletim sisteminin çözüm sunacağı alan analizi yapıldıktan sonra alt sistem ayrışımı ve arayüz belirlenmesi gerçekleştirilir. Arayüzler belirlendikten sonra alt sistemler, uygulama ihtiyaçlarına göre gerçekleştirilirler. Aynı arayüzlere sahip olan farklı alt sistem gerçekleştirimleri, uygulama ihtiyaçlarına göre birbirleri ile değiştirilerek işletim sisteminin şekillendirilmesi yapılır.

Gömülü işletim sistemlerinde tasarım desenlerinin kullanılması, alt sistemlerin farklı gerçekleştirimlerinin birbirleri ile değiştirilebilir olmasını sağlar ve uygulama gereksinimlerine göre ilgili işletim sistemi alt sisteminin seçilmesini kolaylaştırır. Uygulama gereksinimleri gömülü sistemlerde derleme anında belirli olduğu için, burada tasarım desenlerinin çalışma zamanında sunduğu değişebilirlik değil; *derleme*

zamanında sunduğu değişebilirlik daha çok ön plandadır. İşletim sistemi derleme zamanında, uygulama gereksinimlerini karşılayan alt sistemleri ile derlenir. Alt sistem değişebilirliğinin sağlanmasında tasarım desenleri önemli yer tutar.

Tasarım desenleri, tasarımın yeniden kullanılabilirliğine dayanmaktadır. Ancak tasarım desenlerinin ortaya koyduğu çözüm yolları, sistemin belirli özelliklerini geliştirir ve iyileştirirken, bazı özelliklerini ise ters yönde etkilemektedir. Yani bir sistem tasarlanırken, o sistemin gereksinimlerini tam ve eksiksiz bir şekilde yerine getirmesi için, tasarım desenlerinin getirdiği iyileştirmeler ve ek yükler dikkatli bir şekilde göz önüne alınmalıdır.

Tasarım desenlerinin bir sistemde etkileyebileceği noktalar şu şekilde sıralanabilir (Douglass, 2002):

- Sistem Başarımı
- Tahmin edilebilirlik
- Planlanabilirlik
- Verim
- Güvenilirlik
- Yeniden kullanılabilirlik
- Dağıtıklık
- Taşınabilirlik
- Bakım
- Genişleyebilirlik

- Karmaşıklık
- Kaynak kullanımı
- Enerji tüketimi
- Donanım maliyeti
- Sistem geliştirme emeği ve maliyeti

Görüldüğü gibi, gerçek zamanlı ve gömülü sistemler için belirtilen noktalar çok önemlidir. Sistemin tasarımında tasarım desenleri kullanılırken, bu noktaların nasıl etkilendiği dikkatlice değerlendirilmelidir. Sistem, tasarım desenlerinin getirebileceği yüklerden olumsuz yönde etkilenebilir. Tasarım desenlerinin kullanılması ile ortaya çıkan başarım kaybı, sistem içerisindeki görevlerin belirlenmiş zaman kısıtı altında bitirilmesini engelleyebilir. Gerçek zamanlı sistemlerin tahmin edilebilirlik özelliği, tasarım deseni kullanmanın getirmiş olduğu ek soyutlama katmanlarından dolaylı olarak etkilenebilir. Bu sorunlardan dolayı gerçek zamanlı sistemlerde desenler kullanılırken, genel amaçlı sistemlerden daha farklı yöntemlerin izlenmesi gerekir.

4 MODERN İŞLETİM SİSTEMİ KAVRAMLARI AÇISINDAN eGİS TASARIM AMAÇLARI

İşletim sistemleri, tasarımı zor ve zahmetli olan çok büyük yazılımlardır. Kapsadığı problem uzayı ve gereksinimlerinin kolay belirlenememesi, tasarım hedeflerinin gerçeklenmesinin çoğu zaman donanımsal kısıtlar nedeniyle zorlaşması, diğer büyük yazılımlardan ayrıldığı noktalardır. İşletim sistemleri ölmeyen ve uzun süre yaşayan yazılımlar olmaları nedeniyle de diğer yazılımlardan ayrılırlar. Örneğin Unix işletim sistemi 1970'lerden bu yana yaşayan ve günümüzdeki birçok işletim sistemine bir tasarım tabanı olan önemli bir yazılımdır (Bach, 2001).

Windows 2000, 29 milyon satır koda sahip bir işletim sistemidir. Bu kadar büyük bir kodun tek bir insan tarafından anlaşılıp düzenlenebilmesi imkansız görünmektedir. Bu büyüklükte bir kodun yazımı ve bakımının kolay olması, ilerleyen zamanlardaki yeniliklere ve gereksinimlere açık olarak tasarlanabilmesi, ayrıca verimli ve düzgün bir performansa sahip olması, işletim sistemlerinin gerçeklenmesinde birbirleri ile çoğu noktada zıt düşen birçok noktanın göz önünde bulundurulmasını gerektirir. İşletim sistemleri sürekli karmaşıklaşan ve büyüyen yazılımlardır. Dolayısıyla, bakımlarının kolay yapılabilmesi ve gelişen isteklere cevap verebilmeleri için, genişleyebilir ve yeniden kullanılabilir parçalara sahip olmaları gerekmektedir. Bu ise ancak modern yazılım kavramları göz önüne alınarak yapılmış iyi bir tasarım gerektirir.

Gerçekte, işletim sistemleri dünyadaki en karmaşık sistemler olarak görülmemelidirler. Diğer karmaşık sistemler çok büyük olmalarına karşılık birbirleriyle etkileşmeyen alt sistemlere daha çabuk ayrılabilirler. Ancak işletim sistemlerindeki alt birimler birbirleri ile sürekli etkileşim halindedirler. Çoğu noktada verimli etkileşimin sağlanması ve sistem içerisindeki alt birimlerin birbirlerinden soyutlanması, birbirleri ile çelişen noktalardır. İç içe girmiş karmaşık sistemler ölmeye mahkumlardır. Ancak estetik olarak güzel görünen soyutlamalar da belirli noktalardaki kritik işlerde kullanılamazlar. Nerelerde soyutlamalar yapılması gerektiği, nerelerde daha verimli kod yazılması gerektiği iyi

analiz edilmelidir. Örneğin eğer donanım bir işi çok verimli ve hızlı bir şekilde yapma kapasitesine sahipse, soyutlamalar kullanmak yerine donanımın sağladığı bu özellik doğrudan olarak kullanılabilir. Fakat bu seçim yapılırken donanıma bağımlılık artmış olur. Soyutlamalar donanımın sunmuş olduğu çoğu şeyden faydalanmayı önleyebilir.

Monolitik işletim sistemleri modüler olarak tasarlanmışlardır. İşletim sistemi çekirdeği sistemdeki gerekli tüm önemli servisleri sunmaktadır. Bellek yönetimi, süreç yönetimi, giriş/çıkış yönetimi gibi işlevlerin tamamı çekirdeğin içerisinde yönetilmektedir. Monolitik sistemler modüler yapıya sahip olmalarına rağmen, genişleyebilirlik ve dağıtıklık özelliklerinin bu sistem mimarisi içerisinde gerçeklenmesi zordur.

Mikroçekirdek mimarisi, modern ve yazılım mühendisliğine daha yatkın bir işletim sistemi mimarisidir. Burada amaç minimal bir çekirdek sağlamaktır. Bu sayede dağıtıklık, güvenlik ve hataya duyarlılık sağlanacaktır. Hafıza yönetimi gibi önemli bir işlev eğer kullanıcı seviyesinde bir alt sistem olarak gerçeklenirse alttaki donanımdan ve işletim sisteminden bağımsız bir hafıza yönetim mekanizmasına sahip olunur. Bu sayede servis sağlayan alt sistemler çökse bile sistem çökmeyecek ayrıca bu alt sistemler yenileri ile değiştirilebileceklerdir. Bu sayede yeni gereksinimler, genişleyebilirlik ve tutarlılık daha iyi sağlanmış olacaktır. Böylece donanım bağımsız bir işletim sistemi de gerçeklenebilecektir (Liedke, 1996).

Yeniden kullanılabilirlik, işletim sistemleri ve gömülü yazılımlar için de çok önemli bir noktadır. Hata bulma ve ayıklama işlemlerinin diğer yazılım sistemlerine göre daha zahmetli olduğu düşünülürse, aynı kod tekrar tekrar yazılıp hata ayıklama işlemlerinin yapılması yeniden kullanılabilirlik ile engellenecektir. Ayrıca işletim sistemi kod büyüklüğünün düşmesi gömülü sistemler için önemlidir. İyi test edilmiş ve verimlilik göz önüne alınarak gerçekleştirilmiş alt sistemlerin yeniden kullanılması ile, kod büyüklüğü azalacaktır.

İşletim sistemi kullanıcıları ve geliştiricileri, daha genel ve değişik istekleri daha rahat karşılayacak bir mimariye sahip sistemlere gün geçtikce daha fazla ihtiyaç duymaktadırlar. Özellikle gerçek zamanlı

sistemlerin farklı ortam ve çok değişik istek ve kısıtlara sahip olduğu düşünülürse, genel ve bu isteklere göre degiştirilip uyarlanabilecek bir işletim sistemi mimarisi büyük bir ihtiyaç olarak ön plana çıkmaktadır.

4.1 eGİS Tasarım Hedefleri

Nesneye yönelik programlamanın sağladığı olanaklar ve sunmuş olduğu esneklik ile, daha kolay genişleyebilen ve ilerideki gereksinimlere çabuk adapte olabilecek gömülü bir işletim sistemi tasarımı, eGİS tasarımının esas hedefidir.

Gömülü sistemler son yıllarda oldukça önem kazanmaktadır. Sınırlı bir donanımsal ortamda belirli bir işlevi yazılım/donanım işbirliği ile çözen gömülü sistemler, milyonlarca cihazda kullanılacağı için işletim sistemleri için yeni bir alan ve gelişim ortamı sunmaktadır. eGİS, gerçek zamanlı ve gömülü sistemlerin kısıtlamaları göz önüne alınarak bir gömülü bir mikroçekirdek işletim sistemi olarak tasarlanmıştır. Özellikle gömülü işletim sistemlerine duyulan ihtiyaç ve yazılım teknolojilerindeki yenilikler, gömülü işletim sistemi çekirdeklerinin temel yazılım mühendisliği ve modern yazılım kavramları göz önüne alınarak gerçekleştirilmelerini gerektirmektedir.

Özetleyecek olursak, gömülü sistemler için geliştirilen eGİS işletim sisteminin temel tasarım amaçları şu şekilde sıralanabilir:

- Gerçek zamanlı ve mikroçekirdek mimarisine sahip bir çekirdek ortaya koymak
- Tasarım desenlerinin kullanıldığı düzenli bir mimari sunmak
- Uygulama ihtiyaçlarına göre genişleyebilir sistem ortaya koymak

- Taşınabilir sistem ortaya koymak
- Yeniden kullanılabilir bileşenler ile sağlam bir sistem mimarisi ortaya koymak
- Modern yazılım kavramlarının kullanıldığı nesneye yönelik bir tasarım ortaya koymak
- Uygulama isteklerine göre biçimlenebilen bir sistem ortaya koymak

4.2 eGİS İşletim Sisteminin Gerçekleştiriminde Göz Önüne Alınan Nesneye Yönelik Kavramlar

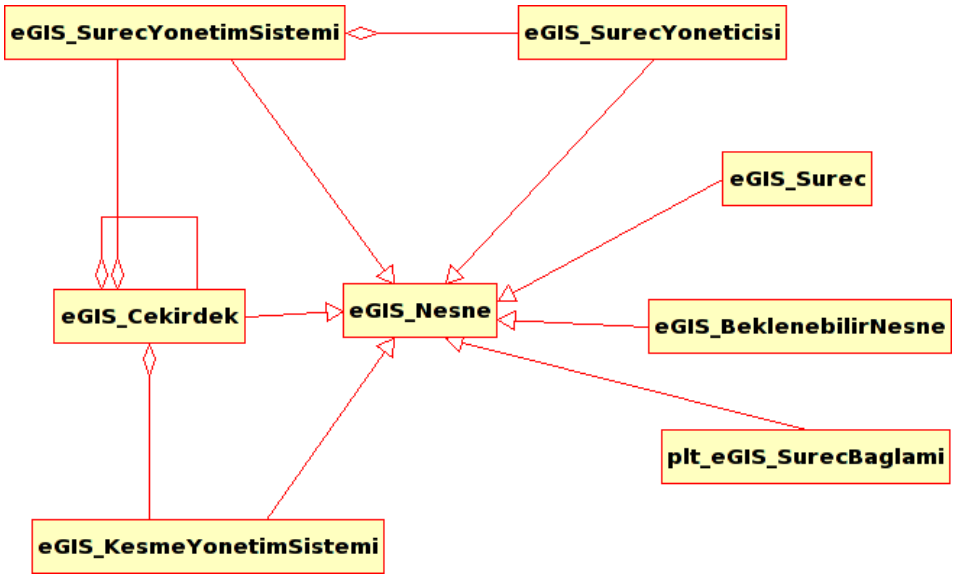
eGİS işletim sistemi, nesneye dayalı programlama paradigması ve C++ programlama dili kullanılarak gerçekleştirilmiştir. Gerçekleştirimde, temel nesneye yönelik tasarım yöntemleri ve nesneye yönelik programlamanın sunmuş olduğu kavramlar kullanılmıştır. Temel nesneye yönelik programlama kavramları olan *çokyapılılık* ve *dinamik bağlama* sistemde çoğu noktada uygulanmıştır.

Dinamik baglama ile, bir isteğin nesne tarafından karşılanması sırasında yapılan işlem; o isteğe ve o isteği karşılayan nesneye bağlıdır. Aynı istek arayüzünü sağlayan farklı nesneler, isteği farklı biçimlerde yerine getirebilirler. Yani isteğin farklı nesnelerdeki gerçekleştirimi farklı olabilir. Çalışma zamanında isteğin ilgili nesne ile ilişkilendirilmesine **dinamik bağlama** denilmektedir. eGİS sisteminde, dinamik bağlama ile aynı arayüzlere sahip alt sistemlerin ve nesnelerin birbirleri ile değişebilirliği, sistemin ortam bağımsız hale gelmesi ve test edilebilir olması kolaylaşmıştır.

Dinamik bağlama ile aynı arayüzlere sahip nesneler aynı isteği karşılayabilirler ve nesneler birbirleri ile değiştirilebilirler. Nesnelerin

birbirleri yerine geçebilme özelliğine ise **çok yapılılık** denmektedir. Çokyapılılığın kullanılması ile, eGİS sisteminde bulunan nesnelerin birbirleri ile olan ilişkileri azalmıştır. Nesneler birbirlerinin gerçekleştirimlerinden haberdar değillerdir.

eGİS sisteminde her şey bir sınıftır ve sistemdeki tüm sınıflar *eGIS_Nesne* sınıfından türemektedir. *eGIS_Nesne* sınıfı tüm nesneler için bir ortak arayüz ve temel oluşturmaktadir. Şekil 4.1'de eGİS sistemindeki nesne hiyerarşisine bir örnek verilmektedir. Sistemdeki arayuzler C++ dilinde **saf sanal fonksiyonlar** ile sağlanmıştır. Çok yapılılık ise C++ diline özgü **sanal foksiyonlar** ile gerçekleştirilmektedir .



Şekil 4.1 eGIS Sistemindeki En Temel Nesne Hiyerarşisine Bir Örnek

eGİS sisteminde sınıf kalıtımı yerine arayüz kalıtımı kullanılmıştır. Bir nesnenin sınıfı, o nesneye ait gerçekleştirimleri içermektedir.

Nesnenin tipi olan arayüzü ise nesnenin sunduğu işlemleri göstermektedir. Sınıf kalıtımında, türetilen sınıf türemiş olduğu sınıfın gerçekleştirimlerini içermektedir. Bir anlamda türemiş olduğu sınıfın kodunu yeniden kullanmaktadır ve paylaşmaktadır. Arayüz kalıtımında ise amaç nesnelerin birbirlerinin yerine geçebilmelerini sağlamaktır. Sınıf kalıtımında amaç türetilen sınıfı genişletmek ve ek işlemler eklemektir. Eski sınıfın üzerinden yeni nesneler türetilir. Ancak sınıf kalıtımında neyin alt sınıfta gerçekleştirileceği, neyin üst sınıfta gerçekleştirilecegi bir problemdir. Ayrıca sınıf hiyerarşisi çok fazla sayıda sınıf olduğunda oldukça karışık ve hantal bir yapı oluşturmaktadır.

Örneğin *CHORUS* işletim sistemi, nesneye yönelik bir gerçekleştirime sahiptir ve sınıf kalıtıma dayalı işletim sistemi mimarisine sahiptir. Bu da işletim sisteminin bakımını ve değiştirilebilir olmasını zorlaştırmaktadır (Campbell et al., 1991).

Arayüz kalıtımı dikkatli bir şekilde kullanıldığında ise türetilen tüm sınıflar aynı arayüze sahip olmaktadırlar. Arayüzde sahip olunan tüm işlemlere türetilen sınıflar cevap vermektedirler. Ayrıca sınıf kalıtımında olduğu gibi üst sınıfın arayüzdeki işlemleri nasıl gerçekleştirdiğini bilmeye gerek yoktur. Bu da gerçekleştirimde bağımsızlığın artmasını sağlamaktadır.

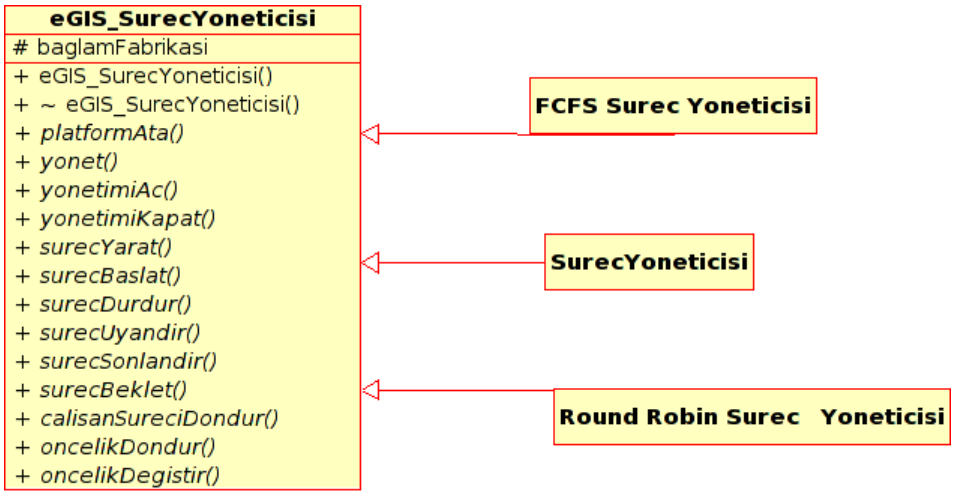
Soyut arayüzler kullanılarak yapılan kalıtım avantajları şöyle özetlenebilir (Gamma et al., 1994):

- Metodları çağırılan nesnenin hangi tipte olduğunun bilinmesine gerek yoktur.

- Metodları çağıran istemcilerin nesnelerin bu metodları nasıl gerçekleştirdiğini bilmesine gerek yoktur.

Görüldüğü gibi arayüz kalıtımı ile nesnelerin birbirleri ile olan bağımlılıkları azalmıştır. Dolayısıyla sistem daha sağlam bir tasarım tabanına oturmaktadır. eGİS sisteminde arayüz kalıtımı kullanılmıştır ve sistemin gerçekleştirimlerden bağımsızlığı sağlanmıştır. Sistem soyut

arayüzler üzerine oturmaktadır. Soyut arayüzlerin somut gerçekleştirimler ile bağlanması çalışma zamanında olmaktadır.



Şekil 4.2 eGİS Sistemindeki Arayüzler ve Bunların Gerçekleştirimlere Bağlanmalarına Bir Örnek

Şekil 4.2' de eGİS sistemindeki bir arayüz kalıtımının sisteme getirdiği esneklik örneklenmektedir. eGİS sistemi içerisinde kullanılacak tüm süreç yöneticileri *eGİS_SurecYoneticisi* arayüzünden türemektedir. Süreç yönetim servislerini kullanacak olan istemciler, bu arayüzün sağladığı metodları çağırarak ve bu metodlardan dönen değerleri kullanarak işleyişlerini sürdürmektedirler. İlk gelen ilk çalışır algoritmasını gerçekleştiren, Round-Robin algoritmasını gerçekleştiren ve öncelik tabanlı kesilebilir süreç yönetim algoritmasını gerçekleştiren süreç yöneticileri *eGİS_SurecYoneticisi* arayüzünden türedikleri için, sistemde birbirleri ile değiştirilebilirler. Bu da eGİS işletim sisteminin süreç yönetim algoritmasına olan bağımlılığını ortadan kaldırmış ve uygulamanın ihtiyaçlarına uygun bir süreç yönetim algoritmasının sisteme birleştirilmesine olanak sağlamıştır.

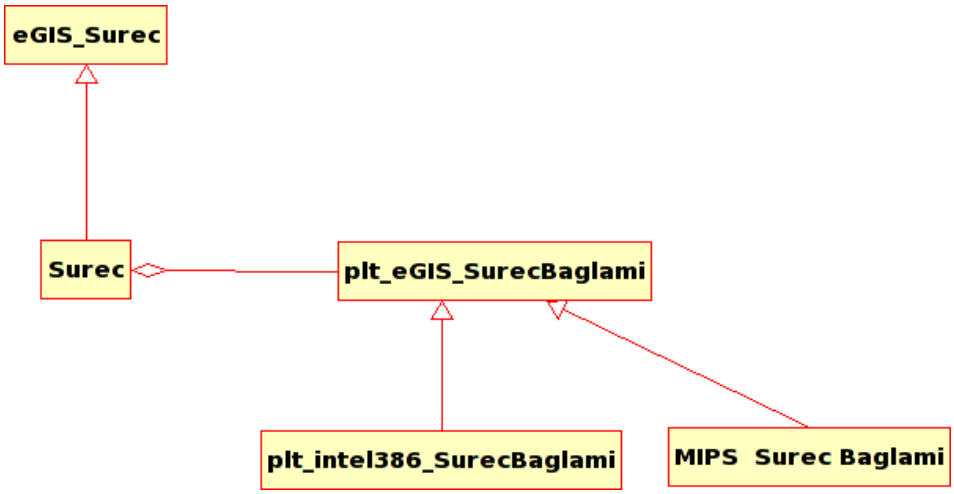
Sistem gerçeklenmesi sırasında sınıf kalıtımı ile ve nesne içerme arasındaki farkların da göz önüne alınması gerekmektedir. Yukarıda belirtildiği gibi sınıf kalıtımında üst sınıfın kodu alt sınıfta kullanılmaktadır ve alt sınıf üst sınıfın gerçekleştirimlerinden haberdardır. Bu tip kullanım bir **beyaz kutu** yeniden kullanımı olarak da algılanabilir. Çünkü üst sınıfın içeriği türetilen sınıf tarafından bilinmektedir (Gamma et al., 1994).

Nesne içerme ise bir **kapalı kutu** kullanımına örnektir ve sınıf kalıtımına bir alternatiftir. Burada bir sınıf, ek işlevselliği diğer bir nesneyi içererek kazanmaktadır. İçerilen nesnenin belirli bir arayüzü yani tipi vardir. Bu sayede nesneyi içeren sınıf, nesnenin içsel yapısı hakkında bilgi sahibi değildir, sadece nesnenin arayüzünü bilmektedir.

Sınıf kalıtımı derleme esnasında belirlenen ve işlenen bir mekanizmadır. Kalıtım derleme esnasında tanımlanır ve çalışma zamanında üst sınıftan alınan gerçekleştirimler değiştirilemez. Her şey derleme esnasında belirlenmiştir. Ayrıca üst sınıf, alt sınıf için bir gerçekleştirim bağımlılığı oluşturmuştur. Türetilen sınıf türediği sınıfa , daha da kötüsü gerçekleştirimine bağımlı hale gelmiştir. Türetilen sınıfa ait işlevler, türenen sınıfa ait gerçekleştirimlerin çoğu zaman bilinmesini gerektirir. Bu da gerçekleştirim işini zorlaştırır. Yeniden kullanılabilirlik daha da zor bir hal almaktadır.

Nesne içerme ise dinamik olarak çalışma anında gerçekleşen bir işlemdir ve sadece nesnelerin arayüzlerine bağlıdır. Her sınıf böylelikle birbirinden bağımsız gerçekleştirilebilir ve birbirlerinin gerçeklestirimlerinden haberdar olmak zorunda değildir. Ayrıca büyük sınıf kalıtım hiyerarşilerinin ortaya çıkmasının önüne geçilmiş olunur. eGİS sisteminde mümkün olduğunca nesne içerme yöntemi uygulanmıştır.

Şekil 4.3 eGİS sistemindeki içerme mekanizması sayesinde nesnelerin birbirleri ile değişebilir olduğunu göstermektedir. Aynı arayüzü gerçekleştiren nesneler, farklı gerçekleştirim mantıkları içerse de, birbirlerinin yerine kullanılabilirler.



Şekil 4.3 eGİS Sistemindeki İçerme ve İçerme Sayesinde Değişebilirliğin Sağlanması

eGİS sistemindeki her süreç bir süreç bağlamı ile ilişkilendirilmiştir. Süreç bağlamı sürece ait saklanacak olan donanımsal bilgileri tutan ve bu bilgiler üzerindeki işlemlerin tanımlandığı sınıftır. eGİS sisteminde her bağlam, *plt_eGIS_SurecBaglami* arayüzünden türemektedir. *plt_intel386_SurecBaglami*, intel 386 işlemci ailesi için gerçekleştirilmiş olan süreç bağlamı sınıfıdır. Görüldüğü gibi arayüz kalıtımı sayesinde, intel 386 platformu için gerçekleştirilmiş bu sınıf, MIPS mimarisine ait bir işlemci mimarisi için bağlam arayüzünü gerçekleştiren başka bir sınıf ile değiştirilebilir. Sistem sadece *plt_eGIS_SurecBaglami* arayüzünü görmektedir. Bu sayede eGİS sisteminin ortam bağımsızlığı sağlanmıştır.

5 EGİS İŞLETİM SİSTEMİ

Bu bölümde eGİS işletim sistemi mimarisi açıklanmış ve tasarım desenlerinin eGİS sistemi içerisinde nerelerde ve nasıl kullanıldığı gösterilmiştir. Tasarım desenlerinin eGİS işletim sistemine katmış olduğu avantajlar ve getirmiş olduğu ek yükler belirtilmiştir.

5.1 eGİS İşletim Sistemi Mimarisi

eGİS gerçek zamanlı ve gömülü bir mikroçekirdek olarak tasarlanmıştır. eGİS çekirdeğinin temel olarak sunduğu servisler şöyle özetlenebilir:

- Süreç yönetimi
- Eş zamanlama yönetimi
- Kesme yönetimi

eGİS, yukarıda belirtilen işlevleri yerine getiren bir alt sistemler mimarisine sahiptir. Bu alt sistemler birbirinden bağımsız olarak tasarlanan temel işletim sistemi bileşenleridir.

Süreç yönetimi kapsamında eGİS işletim sistemi tarafından sunulan temel servisler şunlardır:

- Süreç yaratımı
- Süreç başlatma

- Süreç uyandırma
- Süreç bekletme
- Süreç sonlandırma
- Süreç öncelik değiştirme

Eş zamanlama yönetimi kapsamında eGİS işletim sistemi tarafından sunulan servisler şunlardır:

- Semafor yönetimi
- Kilit yönetimi

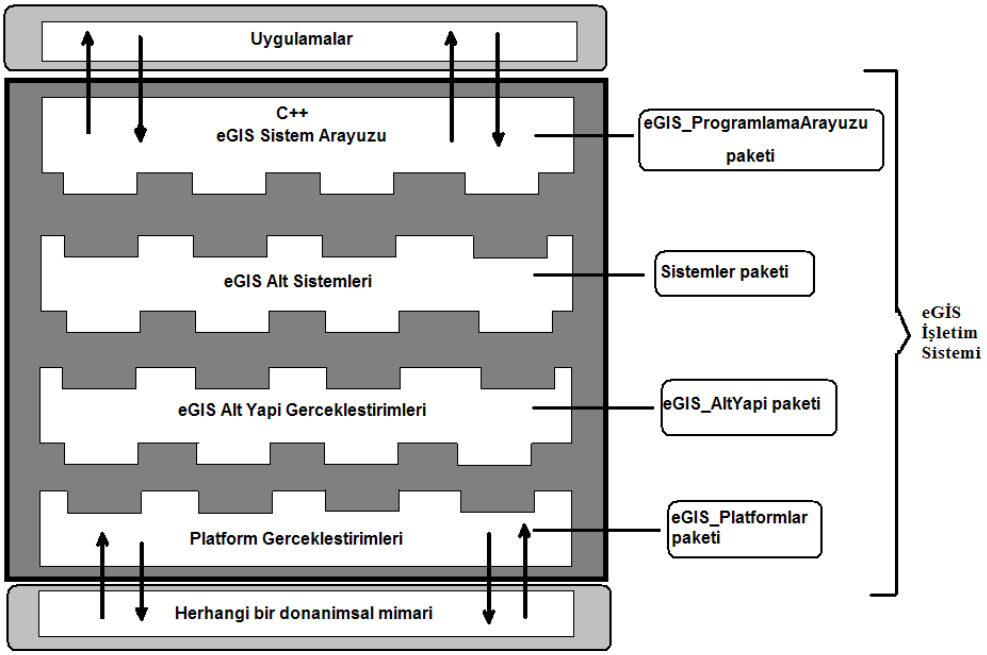
Kesme yönetimi kapsamında eGİS işletim sistemi tarafından sunulan servisler şunlardır:

- Kesme servisçisi atama
- Temel kesme işleme servisi

eGİS işletim sistemi, yukarıda belirtilen servislere ek servislerin de eklenebileceği ve var olan servis gerçekleştirimlerinin de yenileri ile değiştirilebileceği bir mimarisel yapıya sahiptir.

5.1.1 eGİS'in Mimarisel Gösterimi

Belirtilen servisleri sunan, temel eGİS mimarisi şekil 5.1'deki gibi gösterilebilir:



Şekil 5.1 eGİS İşletim Sisteminin Katmanlı Mimarisi

eGİS;

- C++ arayüzü
- alt sistemler

- alt yapı
- platform

gerçekleştirimlerini içerir ve katmanlı bir mikroçekirdek yapısına sahiptir. Sistemde temel soyutlamalar, arayüzler ve varsayılan gerçekleştirimler *eGIS_AltYapi* paketi içerisinde yer almaktadır. *Sistemler* paketi *eGIS_AltYapi* paketi içerisinde yer alan soyutlamalara ait gerçekleştirimleri içermektedir.

Örneğin bir süreç yönetim sınıfına ait temel arayüz tanımlamaları *eGIS_AltYapi* paketi içerisinde yer almakta, bu sınıfa ait uygulamaya yönelik algoritmanın gerçekleştirimi ise *Sistemler* paketi içerisinde gerçekleştirilmektedir. Bu sayede arayüzler ve gerçekleştirimler birbirinden ayrılmıştır.

eGIS_Platformlar paketi, *eGIS_AltYapi* paketi içerisinde donanımsal bağımlılıkları ortadan kaldırmak için tanımlanmış arayüzlerin gerçekleştirimlerini içermektedir. Bu sayede eGİS işletim sisteminin ortam bağımsızlığı sağlanmıştır. Sistem içerisinde yer alan hiçbir paket ortama bağımlılık içermez. Seçilen ortama ait gerçekleştirim, işletim sistemine bağlandığı anda işletim sistemi eksiksiz çalışacaktır.

eGİS içerisinde yer alan paketlerin birbirinden bağımsız olmasını sağlayan nedenlerden biri de temel arayüzlerin iyi bir gömülü sistem alan analizinden sonra belirlenmiş olmasıdır. Çok genel arayüzler sayesinde, tüm bileşenler arayüzler üzerinden işlem yapmakta ve birbirlerinin gerçekleştirimlerinden haberdar olmamaktadırlar.

eGIS_ProgramlamaArayuzu paketi ise uygulama sınıflarına bir C++ arayüzü sunmaktadır. Sistemde yer alan bileşen gerçekleştirimlerinden bağımsız olan ve yine arayüzler üzerinden işlem yapan bu paket sayesinde, uygulamalar da standart bir arayüzle işletim sistemi servislerini kullanmaktadır.

Çizelge 5.1, eGİS işletim sistemi içerisindeki paketleri ve bu paketlerin görevlerini göstermektedir. *eGİS_AltYapi* katmanı, işletim

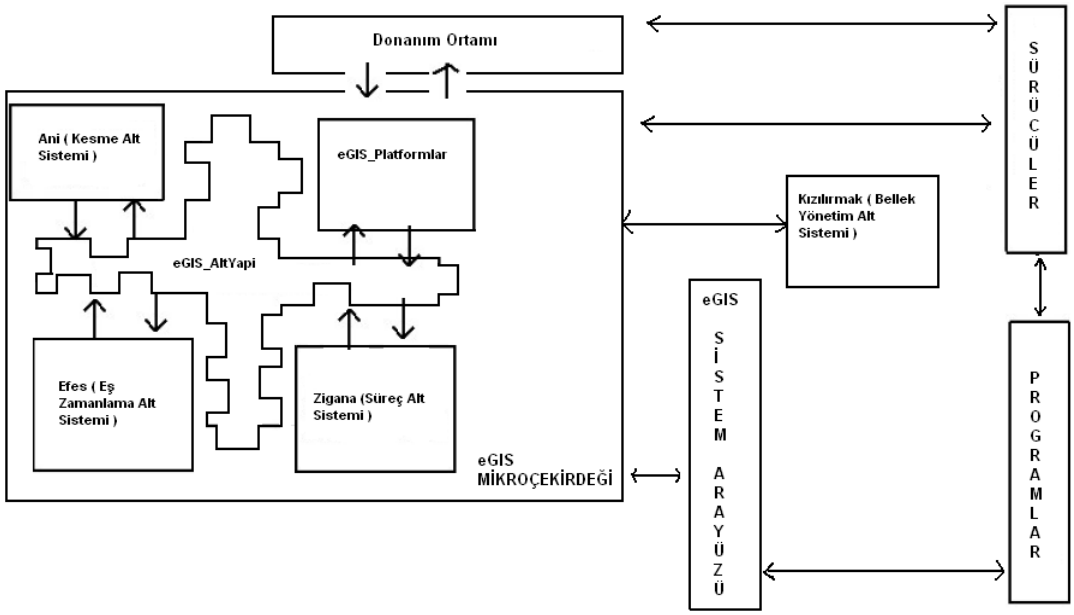
sisteminin temel iskeletini oluşturmakta ve diğer paketler ise bu altyapı üzerine kurulan alt sistem gerçekleştirimlerini içermektedir.

Paket İsmi	**İçeriği ve Görevi**
eGİS_ProgramlamaArayuzu	Uygulama programlarına genel bir arayüz sunmak.
Sistemler	Mikroçekirdeği oluşturan değişebilir alt sistemleri içerir. İçerdiği paketler *Zigana* süreç alt sistemi, *Ani* kesme yönetim alt sistemi ve *Efes* eş zamanlama alt sistemidir.
eGİS_AltYapi	Mikroçekirdeğin en temel gerçekleştirimlerini ve alt sistemler için temel çatıyı oluşturmaktadır. İçerdiği paketler *eGIS_SurecEsZamanlamaSistemi*, *eGIS_KesmeSistemi*, *eGIS_SurecSistemi*, *eGIS_Temel* ve *eGIS_Cekirdek*'tir.
eGİS_Platformlar	Mikroçekirdeğin donanım bağımlı gerçekleştirimlerini içerir. *plt_intel386* paketi intel 386 işlemci ailesine ait gerçekleştirimleri içerir.

Çizelge 5.1 eGİS İşletim Sisteminin Paketleri

5.1.2 eGİS Mikroçekirdeğinin İçsel Yapısı

eGİS işletim sistemi, bibirbirleri iyi tanımlanmış arayüzler ile iletişim halinde olan alt sistemlerden oluşmaktadır. Mikroçekirdek süreç, kesme ve eş zamanlama servislerini içermektedir. Alt sistemler, birbirleri ile olan ilişkilerini çekideğin sunmuş olduğu alt yapı servisleri ile kurmaktadırlar. Bu alt sistemler, çekirdeğin içerdiği arayüz gerçekleştirimlerini içermektedirler. Şekil 5.2'de alt sistemlerin çekirdek içerisindeki yerleşimi gösterilmiştir.



Şekil 5.2 eGİS İşletim Sisteminin İçsel Yapısı

Kesme servisleri için eGİS_AltYapı paketi içerisinde tanımlanmış soyutlamaları gerçekleştiren *Ani* kesme alt sistemi, süreç servisleri için eGİS_AltYapı paketi içerisinde tanımlanmış soyutlamaları içeren *Zigana* süreç alt sistemi ve eş zamanlama servisleri için eGİS_AltYapı paketi içerisinde tanımlanmış soyutlamaları gerçekleştiren *Efes* alt sistemi; eGİS işletim sisteminin algoritmik bağımlılıklarının soyutlandığı alt sistemlerdir. Uygulama ihtiyaçlarına göre yeni alt sistem gerçekleştirimleri sisteme eklenebilir, var olan alt sistemler yenileri ile değiştirilebilirler.

Zigana süreç yönetim alt sistemi, öncelik tabanlı kesilebilir süreç yönetim algoritmasını gerçekleştirmektedir. Bu algoritmaya göre sistemde her sürecin bir önceliği vardır ve o an çalışmakta olan sürecin çalışması kendisinden daha öncelikli bir süreç aktif hale geldiğinde durdurulmakta ve işlemci daha öncelikli olan sürece verilmektedir.

Ani kesme alt sistemi, basit bir kesme yönetim algoritmasını gerçekleştirmekte ve temel kesme ekleme ve çıkartma servislerini sunmaktadır.

Efes eş zamanlama alt sistemi semafor ve kilit gerçekleştirimlerini içermektedir. Semafor ve kilit yönetiminde yine öncelik tabanlı kesilebilir süreç yönetim algoritmasına uygun süreç değerlendirilmesi yapılmaktadır. Bir semafor sinyallendiğinde ya da bir kilit açıldığında, o nesne üzerinde bekleyen en yüksek önceliğe sahip süreç hazır hale getirilmektedir.

Detaylı mimaride görüldüğü gibi alt sistemler tamamiyle alttaki donanım ortamının ne olduğundan ve birbirlerinden habersizdirler. Bu sayede alt sistemlerin gerçekleştiriminin bağımsız yapılmasını ve diğer alt sistemlerden soyutlanarak test edilebilmesini sağlanmaktadır.

Uygulama programlarının etkileşimi sistem arayüzü üzerinden gerçekleşmektedir. Uygulama programı içsel alt sistemlerden haberdar olmamakta, sadece kendisine sunulan uygulama geliştirme arayüzlerini görmektedir.

Sürücüler mikroçekirdeğin dışında gerçekleştirilmiş olan, gerekirse mikroçekirdeğin sunmuş olduğu servislerden faydalanabilecek ayrı bir yazılım katmanıdır.

Kızılırmak alt sistemi, bir bellek yönetim alt sistemi olup, eGİS içsel alt sistemlerinden farklı olarak çekirdeğin dışında gerçekleştirilmiş bir alt sistemdir. Çekirdek içerisindeki dinamik bellek tahsis işlemleri, bu alt sistem üzerinden gerçekleştirilmektedir. Her uygulamanın kendine ait bellek tahsis algoritması olacaktır ve bellek tahsis işleminin uygulamanın gerçekleştirimi kapsamında olacağı düşünülmüştür. Çekirdeğin kendi içsel bellek tahsis işlemleri, uygulama programlarından bağımsızdır. Kızılırmak dışsal alt sistemi, ardışıl bellek atama algoritmasını gerçekleştirmektedir (Blunden, 2003).

Çekirdeğin bir uygulamaya yönelik işletim sistemi olan eGİS'te bir alt sistemler yığını olarak tasarlanması, uygulama ihtyaçlarına göre mikroçekirdeğin şekillenebilmesini kolaylaştırmıştır.

Görüldüğü gibi katmanlı ve alt sistemlere ayrıştırılmış bir mimari ile, eGİS sisteminin

- Taşınabilirliği
- Değişebilirliği
- Bakımı
- Alt sistemlerin bağımsızlığı ve düzenliliği

sağlanmıştır. Alt sistemlerin ve paket yapılarının tasarımında, tasarım desenlerinin sağladığı olanaklar büyük ölçüde kullanılmıştır.

5.2 Tasarım Desenlerinin Gerçek Zamanlı eGİS İşletim Sisteminde Uygulanışları

eGİS işletim sistemi, bir uygulamaya yönelik işletim sistemidir. Bu da sistemin sunmuş olduğu servislerin ve işleyiş mantığının, uygulama gereksinimlerine göre degişebilmesini gerektirir. Tasarım desenleri, eGİS işletim sisteminin uygulama gereksinimlerine göre değişebilecek noktalarında kullanılmıştır.

eGİS sisteminde tasarım desenlerinin temel olarak çözdüğü problem alanları aşağıda özetlenmiştir:

- Sistemin genişleyememesi ve yeni gereksinimlere cevap verememesi, dağıtıklaştırılamaması: Uygulama gereksinimlerine uygun sistem şekillendirilebilmesi için, alt sistemlerin genişleyebilir ve değişebilir tasarlanmaları gerekmektedir. **Mikroçekirdek** mimarisel tasarım deseni, eGİS alt sistem ayrıştırılmasını sağlar ve sistemi daha da genişleyebilir hale getirir.

- Nesneleri açıkça, somut sınıfının adını kullanarak yaratmak: Bu, sistemi belirli bir sınıf gerçekleştirimine bağlamaya neden olmaktadır. **Soyut fabrika** ve **tekil** tasarım desenleri bu problemi eGİS sisteminde çözmüştür.

- Donanım ve yazılım ortamlarına bağımlılık: Belirli bir ortama bağımlılık, sistemin genişleyebilirliğini ve taşınabilirliğini azaltmaktadır. **Köprü** ve **soyut fabrika** tasarım desenleri, eGİS sisteminde bu problemi çözmek için kullanılmıştır.

- Var olan arayüzler ve sistemlerle uyumsuzluk: Zaten belirli bir arayüz kümesi düşünülerek yazılmış olan uygulamaların eGİS sistemine taşınabilmesi için, arayüz dönüştürme işi

yapılmalıdır. **Adaptör** tasarım deseni bu problemi çözmektedir.

- Algoritmik bağımlılıklar: eGİS sisteminde birbirleri ile değişebilecek alt sistemlerin tasarımında algoritmik bağımlılıkların ortadan kaldırılmasına gerek vardır. Bu problem **strateji** deseni ile çözümlenmiştir.

- Sistemdeki nesnelerin birbirine bağımlılığı: eGİS sisteminde, nesnelerin birbirinden izole edilmesi ve nesnelerin birbirlerine olan bağımlılıklarının ortadan kaldırılması gerekmektedir. Bu durum, sistemin genişleyebilir ve değişebilir olması için gereklidir. Bu problem eGİS sisteminde **köprü**, **inceleyici**, **ön yüz** ve **soyut fabrika** desenleri ile çözülmüştür.

- Gerçek zamanlı sistemlerin gereksinimlerini uygulamaya özel çözümlerle gerçeklenmesi: Çoğu gerçek zamanlı sistem genişleyebilirlik ve bakım konuları göz önüne alınarak tasarlanmazlar. Gerçek zamanlı sistemlerde yer alan mesaj kuyruğu, kesme ve bellek yönetimi gibi temel işlevlerin daha genel ve sağlam bir şekilde yapılması **mesaj kuyruğu**, **havuz tabanlı tahsis** ve **kesme** tasarım desenleri ile sağlanmıştır.

eGİS işletim sisteminde kullanılan ve yukarıda belirtilen tasarım desenleri beş grup içinde incelenebilir. Bunlar sırasıyla:

- Mimarisel tasarım desenleri

- Yaratımsal tasarım desenleri

- Yapısal tasarım desenleri

- Davranışsal tasarım desenleri
- Gerçek zamanlı sistemlere özgü tasarım desenleri

5.2.1 eGİS İşletim Sisteminde Kullanılan Mimarisel Tasarım Desenleri

Mimarisel tasarım desenleri, bir yazılım sisteminin yapılandırılması ile ilgili desenlerdir. Bu desenler sistemin alt sistemlere ayrıştırılmasını, alt sistemlerin sorumluluklarının ve alt sistemlerin birbirleri ile olan ilişkilerinin belirlenmesini sağlar. Sistemin kontröllü bir şekilde dağıtık, uyumlu, genişleyebilir ve değişebilir alt sistemlere ayrıştırılması bu desenler yardımı ile sağlam bir şekilde yapılabilmektedir (Buschmann et al., 1996).

Sistemin iyi bir tasarım ile alt sistemlere ayrıştırılması, sistemin devamlılığı için önemlidir. Kötü bir tasarım sonucunda birbiri içerisine geçmiş ve karışık sorumluluklara sahip alt sistemlere ayrıştırılmış bir yazılım sistemi, yeni gereksinimlere cevap veremeyecek ve kısa zamanda ölecektir.

eGİS sisteminde kullanılan mimarisel tasarım deseni *mikroçekirdek* desenidir.

5.2.1.1 Mikroçekirdek Mimarisel Tasarım Deseni

Desenin Kullanım Amacı :

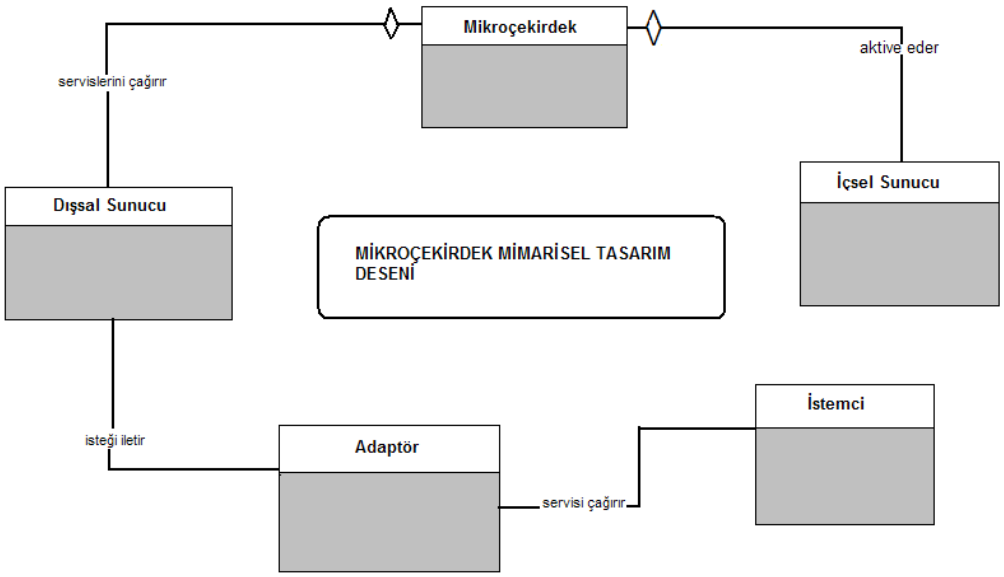
Mikroçekirdek mimarisel tasarım deseni, eGİS sisteminin temel mimarisini oluşturmaktadır. Bu desen, değişen sistem gereksinimlerine

sahip yazılım sistemleri için uygun ve uygulanması oldukça fayda sağlayacak bir tasarım şablonu ortaya koyar. Mikroçekirdek deseni sayesinde, temel bir mekanizma ve bu mekanizmanın farklı gereksinimler için değiştirilmesi ve genişletilmesi işlemleri birbirinden ayrıştırılmış olur.

Mikroçekirdek mimarisinin genel yapısında

- İçsel sunucular
- Dışsal sunucular
- Adaptörler
- İstemciler
- Mikroçekirdek

bileşenleri bulunmaktadır. Şekil 5.3'te mikroçekirdek mimarisinin genel gösterimi sunulmaktadır (Buschmann et al., 1996).



Şekil 5.3 Mikrokernel Mimarisinin Temel Yapıtaşları

Mikroçekirdek bileşeni desenin temel noktasıdır ve sistemin çekirdek servislerini sunar, alt sistemlerin birbirleri ile haberleşmesini yönetir ve alt sistemlerin kendi servislerini sunması için bir alt yapı oluşturur. Sisteme özel kısıtlar, donanım bağımlılıkları mikroçekirdek katmanı tarafından soyutlanmıştır. Mikroçekirdek katmanının sunduğu servisleri kullanan alt sistemler, donanımsal bağımlılıkları mikroçekirdek tarafından yok edilmiş sistemlere dönüşürler. Mikroçekirdek, bir alt sistemin kullanacağı en temel mekanizmaları içerir.

İçsel sunucular, mikroçekirdek mekanizmalarının genişletildiği ve mikroçekirdeğin uzantısı olan servisleri gerçekleştiren alt sistemlerdir. Mikroçekirdek olabildiğince küçük olmalıdır ve dış dünyaya sadece belirli sayıda servis sunmaktadır. İçsel alt sistemler sadece mikroçekirdek tarafından erişilebilen alt sistemler olduğu için, mikroçekirdeğin bir parçası gibi de düşünülebilirler ve mikroçekirdek kapsamında gerçekleştirilmemiş servisleri mikroçekirdeğe sunarlar. Mikroçekirdek, içsel sunuculara ihtiyaç duyulduğu anda bu sistemleri aktif hale getirerek ilgili servislerinden faydalanmaktadır.

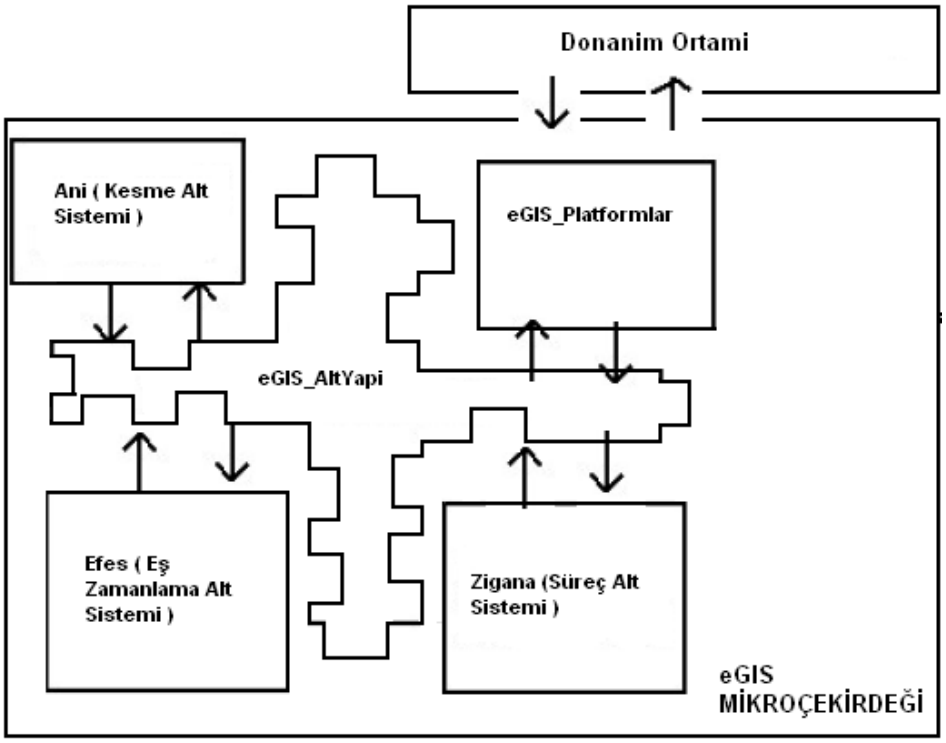
Dışsal sunucular, mikroçekirdek mekanizmalarını ve mikroçekirdeğin dış dünyaya sunduğu arayüzleri kullanan sistemlerdir. İstemciler ile mikroçekirdek arasında bir arayüz oluşturmaktadırlar. İstemcilerden mikroçekirdeğin sunmuş olduğu haberleşme mekanizmaları ile gelen istemler dışsal sunucular tarafından alınmakta, bu istekler dışsal sunucular tarafından mikroçekirdeğe iletilmekte ve mikroçekirdeğin ürettiği sonuçlar ise dışsal sunucular tarafından istemcilere iletilmektedir.

Adaptörler, istemciler ile dışsal sunucular arasındaki arayüzlerdir ve istemcilerin dışsal sunuculara olan bağımlılıklarını ve uyumsuzluklarını yok etmek için kullanılırlar.

Desenin eGİS İçerisinde Kullanılışı :

eGİS'te her bir özelleşmiş isteğin yerine getirilmesi, ayrı alt sistemler tarafından sağlanmaktadır. Bu alt sistemler benzer çekirdek servislerini kullanan fakat her biri ayrı bir işlevselliğe ve gerçekleştirime sahip sistemlerdir. Alt sistemlerin bir arada ve uyum içerisinde çalışması ve düzenlenmesi; ayrıca sistemin kapsamı ve sorumluluğu belirlenmiş sağlam alt sistemlere ayrıştırılması işlemlerinde mikroçekirdek mimarisi kullanılmıştır. eGİS çekirdeği, temel servisleri sunmakta ve çekirdeğe kayıt edilen alt sistemler ise bu temel servisleri uygulamanın özelleşmiş isteklerini yerine getirmek için kullanmaktadır.

eGIS_AltYapi paketi, mikroçekirdek mimarisinin temelini oluşturmaktadır. İşletim sisteminin tüm temel işlevileri ve arayüzleri bu paket içerisinde tanımlanmıştır. Diğer alt sistemlerin yapacağı şey , bu temel servisleri ve arayüzleri kullanıp bunları kendi özelleşmiş gerçekleştirimleri ile birleştirmektir.

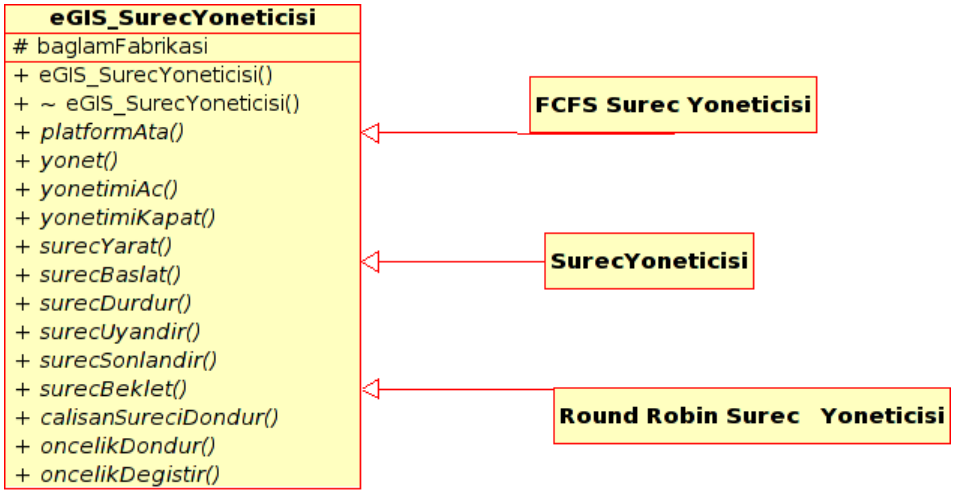


Şekil 5.4 eGİS Mikroçekirdeğinin Yapısı

eGIS_AltYapi paketi içerisinde yer alan *eGIS_SurecSistemi*, *eGIS_KesmeSistemi*, *eGIS_EsZamanlamaSistemi* paketleri, yönetim algoritmalarının arayüzlerini, soyutlamalarını ve temel bir alt sistem çatısını sunmaktadırlar. Bir süreç içsel sunucusu olan *Zigana* paketi,

belirtilen *eGIS_SurecSistemi* paketinin temel servislerini ve arayüz çatısını kullanmaktadır. *Zigana* paketinin içerisindeki gerçekleştirimler tamamiyle *eGIS_AltYapi* paketi içerisinde sunulan arayüzler ve mekanizmalar kullanılarak yapılmaktadır. Bu durum *eGIS_KesmeSistemi* alt sisteminin içerisindeki arayüzleri gerçekleştiren ve mekanizmaları kullanan *Ani* kesme alt sistemi; ve ayrıca *eGIS_EsZamanlamaSistemi* mekanizmalarını kullanan ve arayüzlerini gerçekleştiren *Efes* eş zamanlama alt sistemi için de geçerlidir. Şekil 5.4'te mikroçekirdeğin yapısı üzerinden bu alt paketlerin nasıl sistemle birleştirildiği ve bu paketlerin nasıl yenileri ile değişebildiği gösterilmektedir. Tüm bu alt sistemler, alt yapı paketinin sunmuş olduğu arayüzler ile sistemler birleşmekte ve ortam bağımsız olmaktadırlar.

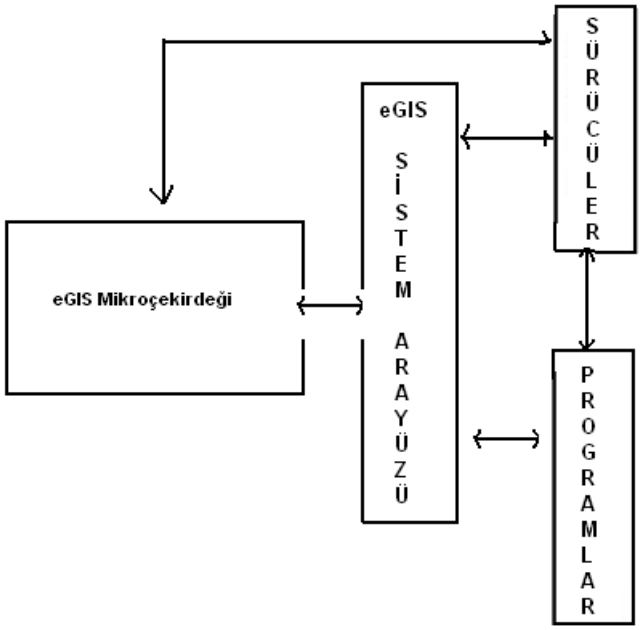
Farklı bir uygulama ihtiyacı için, örneğin farklı bir süreç yönetim algoritması ihtiyacında, *Zigana* süreç alt sistemi yerine aynı arayüzleri yeni algoritmaya göre gerçekleştiren farklı bir alt sistem gerçekleştirilerek sisteme kaydedilebilir. Bu sayede sistem daha esnek hale gelmiş olur. Şekil 5.5'te bu durum gösterilmiştir. Farklı süreç yönetim algoritmaları, aynı arayüzü gerçekleştirdikleri için birbirleri ile değişebilirler. eGİS sisteminde bir Round-Robin süreç algoritmasını gerçekleştiren süreç yöneticisi, ilk gelen ilk çalışır algoritmasını gerçekleştiren farklı bir süreç yöneticisi ile değiştirilebilir.



Şekil 5.5 Süreç Yöneticisinin Arayüzü ve Farklı Gerçekleştirimleri

eGİS mikroçekirdeği ile uygulama programları, *eGIS_SistemArayuzu* paketi yardımı ile haberleşirler. Bu paket mikroçekirdek mimarisi içerisinde bir adaptör ve dışsal sunucu katmanlarının birleşimi olarak görülebilir. Uygulama paketinin istekleri, bu adaptör katmanı yardımıyla eGİS servis isteklerine dönüştürülür. Mikroçekirdeğin işlemlerinin sonuçları, yine bu katman ile uygulama programlarına döndürülür.

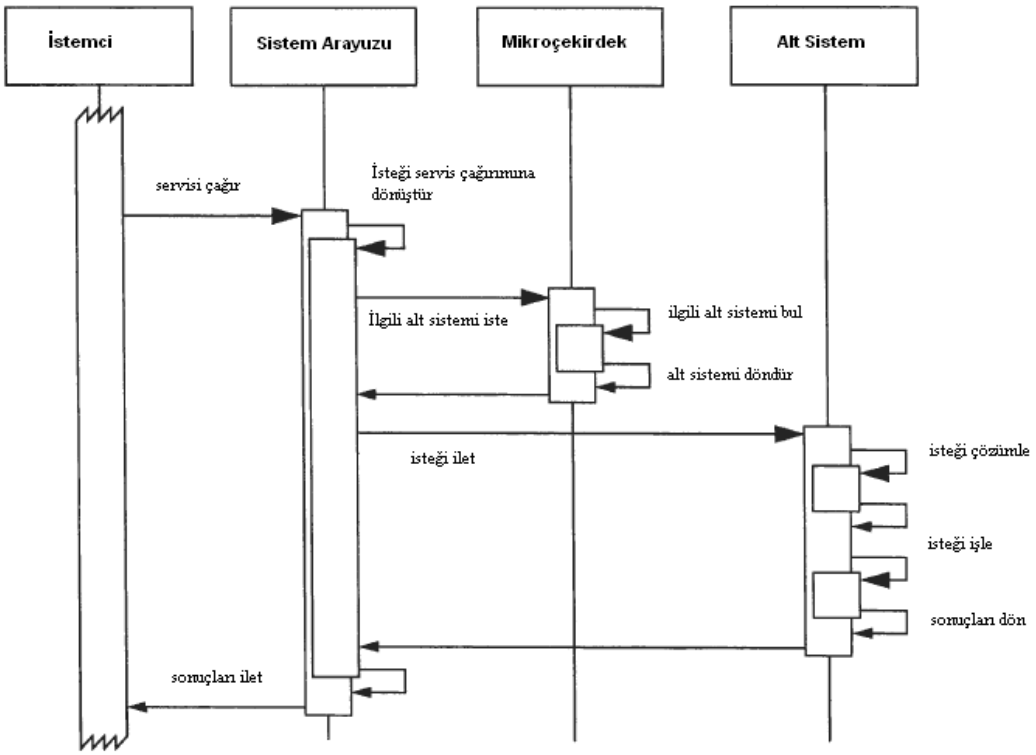
Şekil 5.6'da bir uygulama programı ile eGİS mikroçekirdeğinin iletişimi gösterilmektedir. Uygulama programı sürücüler ve sistem arayüzü ile etkileşmektedir. Mikroçekirdeğin içsel yapısı ve alt sistemlerine ilişkin herhangi bir bilginin uygulama programı tarafından bilinmesine gerek yoktur. Eğer mikroçekirdeğin içerisinde yer alan bir alt sistem başka bir alt sistem ile değiştirilirse, uygulama programları bu değişimden etkilenmeyecektir. Uygulama kodunda herhangi bir değişiklik yapılmasına gerek kalmayacaktır.



Şekil 5.6 Uygulama Programı ve eGİS Çekirdeği

eGİS sistemi içerisinde yer alan alt sistemler birer pasif sunucu olarak ele alınabilirler. Çünkü bu alt sistemler ayrı ayrı birer iş parçacığı olarak gerçekleştirilmemişlerdir. Her bir alt sistem, kendisini kullanan sistem iş parçacığının kapsamı içinde çalışan birer pasif sunucudur. Bu bağlamda, alt sistemler ve eGİS mikroçekirdeği uygulama programları için bir paylaşımlı kütüphane olarak görülebilir.

eGİS sistemi içerisinde bir uygulama programı ve mikroçekirdek etkileşimi, şekil 5.7'deki senaryo ile gösterilebilir (Buschmann et al., 1996):



Şekil 5.7 Uygulama Programı ve eGİS Çekirdeği İle Etkileşimi Senaryosu

- Uygulama programı sistem arayüzü yolu ile bir servis isteğinde bulunur.

- Sistem arayüzü katmanı bu isteği ilgili sistem çağırımı biçimine dönüştürür ve bu servisi gerçekleştirmiş alt sistemi mikroçekirdek katmanından ister.

- Mikroçekirdek katmanı ilgili alt sistemi arayüz katmanına döndürür.

- Arayüz katmanı, alt sistem üzerinden ilgili servis isteğini yapar.

- Alt sistem, isteği çözümler, işler ve sonuçları arayüz katmanına döndürür.

- Arayüz katmanı ise bu istekleri istemciye döndürmektedir.

Desenin Kullanımı Sonucunda Ortaya Çıkan Avantajlar:

Mikroçekirdek mimarisinin eGİS sistemine kazandırdığı başlıca özellikler ve faydalar şu şekilde özetlenebilir (Liedke, 1995; Crowley, 1997; Gien and Grob, 1992):

- *Taşınabilirlik*: eGİS mikroçekirdeğinin değişik bir donanımsal ortama taşınmasında, mikroçekirdeğin mekanizmalarını kullanan alt sistemler bundan etkilenmezler. Mikroçekirdeği yeni bir donanım ortamına taşımak, sadece donanımsal bağımlılık içeren kısımlarının değişmesini gerektirir. *eGIS_Platformlar* paketi, donanımsal bağımlılıkların soyutlamalarının gerçekleştirimlerini içermektedir. *eGIS_AltYapi* içerisinde yer alan donanımsal soyutlamalar, bu paket içerisinde gerçekleştirilmiştir. Dolayısıyla taşınan ortama ilişkin gerçekleştirimler bu pakete eklenirse, eGİS mikroçekirdeği yeni ortama taşınmış olur. Hiçbir alt sistem bundan etkilenmez.

- *Genişleyebilirlik ve Esneklik*: Mirkoçekirdek deseni sayesinde, uygulama ihtiyaçlarını karşılayan uygun alt sistemler çekirdeğe kaydedilebilir. Bu sayede, mikroçekirdek tarafından arayüzleri belirlenmiş ama gerçekleştirimleri farklı çok sayıda alt sistem eGİS sistemine uygulama ihtiyaçlarina göre eklenebilir. Yeni bir

özelliğin eklenmesi, yeni bir alt sistemin gerçeklestirilmesi ve çekirdeğe kayıt edilmesi ile sağlanır.

- *Arayüzler ve Gerçekleştirim Bağımsızlığı*: Arayüzler ve bu arayüzlerin gerçekleştirimleri birbirinden ayrılmıştır. Bu da sistemin birbirinden bağımsız ve değişebilir parçalar olarak tasarlanmasını ve gerçekleştirilmesini sağlamıştır.

Desenin Kullanımı Sonucunda Ortaya Çıkan Dezavantajlar :

Mikroçekirdek mimarisinin eGİS sistemine yüklediği başlıca dezavantaj başarım kaybı ve kod büyüklüğünün artmasıdır. Aynı özelliklere sahip modüler ve yapısal bir programlama dili ile hazırlanmış sistemle karşılaştırılınca, başarım açısından kayıplar vardır. Ayrıca tasarım ve gerçekleştirim, mikroçekirdek mimarisine uygun ayrıştırımlar yapılırken zorlaşmıştır. Ancak eGİS işletim sisteminin uygulama ihtiyaçlarına göre şekillenebilmesi ve donanım bağımsızlığı mikroçekirdek mimarisinin eGİS tasarım amaçları doğrultusunda kattığı önemli avantajlardır (Liedke, 1995; Buschmann et al., 1996) .

5.2.2 eGİS İşletim Sisteminde Kullanılan Yaratımsal Tasarım Desenleri

Yaratımsal tasarım desenleri nesnelerin sistem içerisindeki yaratım işlemlerini soyutlayarak, sistemin nesnelerin yaratımından ve içsel yapısından bağımsız olmasını sağlamaktadır. Nesne yaratımlarının açık bir şekilde ve nesnelerin tipleri verilerek yapılması, sistem genişledikçe

ve değişik ortamlara ve gereksinimlere taşındıkça bakımı ve gerçekleştirimleri zorlaştırmaktadır.

Nesnelerin yaratımı sistem içinde bilinen bir noktada yapılırsa ve bu işlem bilinen bir sınıfa yüklenirse; yaratılan nesneler değiştikçe ya da yeni nesneler yaratılmak istendikçe, sistem bundan en az şekilde etkilenecektir. Burada temel nokta nesnelerin paylaştığı arayüzün iyi bir şekilde belirlenmiş olması gerekliliğidir (Gamma et al., 1994).

eGİS işletim sisteminde kullanılan yaratımsal tasarım desenleri *tekil* ve *soyut fabrika* desenleridir.

5.2.2.1 Tekil Tasarım Deseni

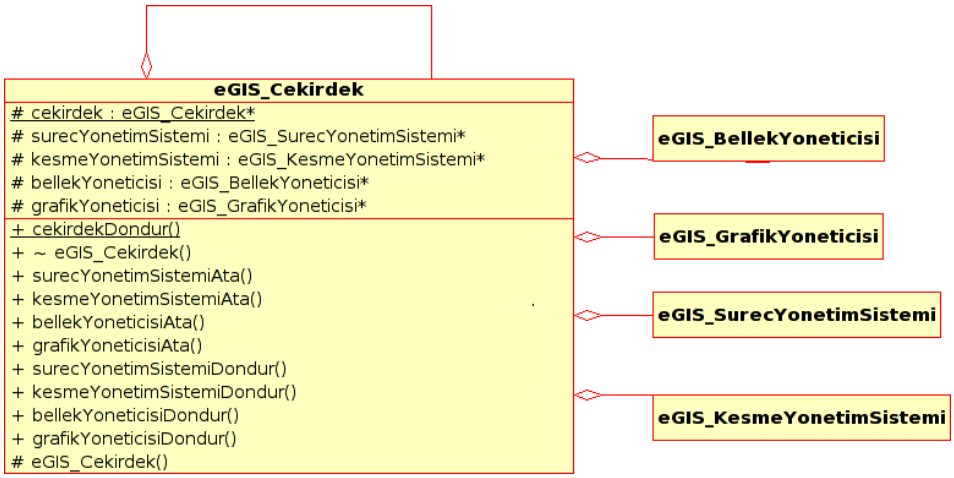
Desenin Kullanım Amacı:

eGİS gerçek zamanlı işletim sisteminde belirli sınıflar sistem içerisinde sadece bir örneğe sahiptir. Tekil tasarım deseni ile tek bir örneğe sahip olacak sınıfların yaratımı ve istemcilerin bu örnekle ilişkilendirilmesi sağlanmıştır. Tekil tasarım deseni sayesinde bu tekil nesneye erişim kolaylaştırılmıştiir. Ayrıca bu sınıftan türetilecek yeni sınıflar, üst sınıfı kullanan istemcilerin kodunda hiçbir değişikliğe neden olmayacaktır.

Desenin eGİS İçerisinde Kullanılışı :

eGİS sisteminde yer alan *eGIS_Cekirdek* sınıfı, tekil tasarım deseni kullanılarak gerçekleştirilmiş, mikroçekirdek sistemin alt sistemlerinin kaydedildiği ve bu alt sistemlere ulaşmak için kullanılacak olan sınıftır. Görüldüğü gibi bir tane örneği bulunacak olan bu sınıf, ileride eklenebilecek alt sistemlerle daha da genişleyebilir bir yapıya kavuşturulabilir. Bu sınıfın genişlemesinde, eski sınıf gerçekleştirimini

kullanan sınıfların etkilenmemesi gerekmektedir. Ayrıca sınıfın yaratımı işlemi, istemcilerden soyutlanmalıdır.



Şekil 5.8 Tekil Tasarım Deseninin Uygulandığı eGIS_Cekirdek Sınıfı

Şekil 5.8'de, tekil tasarım deseninin eGİS sisteminde kullanılışına ait bir örnek gösterilmektedir. eGIS mikroçekirdeğinin alt sistemlerine ulaşılması ve alt sistemlerinin kaydedilmesi, *eGIS_Cekirdek* tekil nesnesi üzerinden olmaktadir. İstemciler bu nesnenin yaratımı ile ilgilenmemektedirler. Bu nesneye erişim temiz ve sağlam bir yöntemle yapılmaktadır. Ayrıca bu nesneyi kullanacak olan istemciler ve alt sistemler, kod açısından kirlenmeden kolayca bu nesne ile iletişime geçerler.

Desenin Kullanımı Sonucunda Ortaya Çıkan Avantajlar :

Tekil tasarım deseni ile :

- *eGIS_Cekirdek* nesnesine kontrollü bir erişim sağlanmıştır.
- Tek bir genel nesneye geleneksel yolla erişim yerine, daha genel ve genişleyebilir bir erişim mekanizması kurulmuştur.
- *eGIS_Cekirdek* sınıfı genişletilebilir ve yeni alt sınıflar türetilebilir. Sistemin bu işlemlerden en az şekilde etkilenmesi sağlanmıştır. İstemciler bu değişimlerden etkilenmemektedirler.
- Sistemde *eGIS_Cekirdek* nesnesinden sadece bir tane örneğin olması garanti altına alınmıştır.

Desenin Kullanımı Sonucunda Ortaya Çıkan Dezavantajlar :

Geleneksel genel nesne kullanımında, sistemdeki nesneye doğrudan erişim vardır. Bu genel nesneye erişim için ek bir metod çağırımı yapılmamaktadır. Tekil tasarım deseni kullanımında ise, sınıf örneğine bir metod ile erişim sağlamaktadır. Gerçek zamanlı bir sistemde çok kritik noktalarda her işlenen komutun ve süresinin önemi vardır. Dolayısıyla tekil tasarım deseni sistem başarımını bu ek metod çağırımı yüzünden azaltmaktadır. Ancak sunmuş olduğu genişleyebilme ve yaratımdan soyutlama, olumlu etkileridir. Bu başarım düşüklüğünü kaldırabilecek gerçek zamanlı sistemlerde kolaylıkla uygulanabilir.

5.2.2.2 Soyut Fabrika Tasarım Deseni

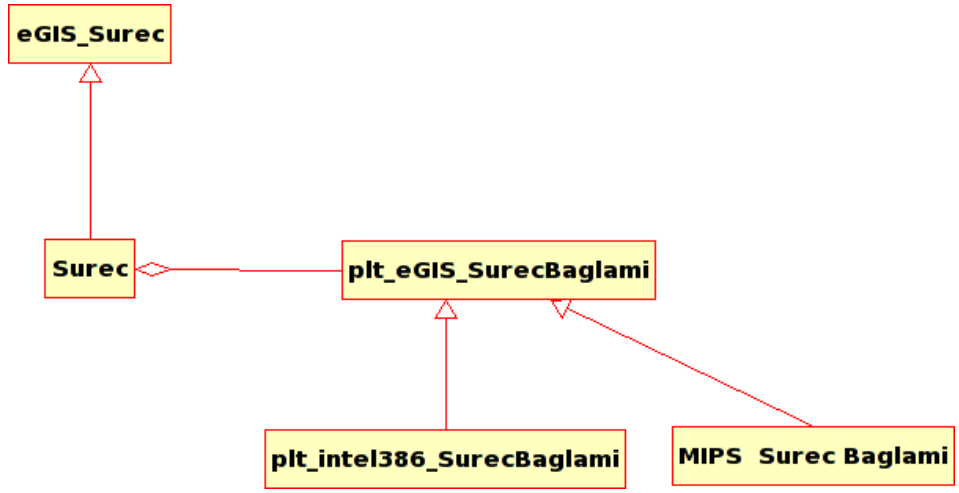
Desenin Kullanım Amacı :

eGİS sistemi içerisinde, nesneler ortak arayüzleri paylaşmaktadırlar. Sistem içerisinde nesneler ortak arayüzlerinden ötürü birbirleri ile değiştirilebilirler. Dolayısıyla, istemciler istekleri ilgili nesnelere gönderirken nesnelerin gerçekleştirimlerinden haberdar değillerdir. İstemciler için nesnelerin tipi değil arayüzleri göz önüne alınmaktadır.

Geleneksel yaklaşımda, sistem içerisinde nesneler somut sınıfları belirlitelerek yaratılmaktadırlar. Nesnelerin somut sınıflarının belirtilmesi, sistemin nesnelerin gerçekleştirimlerine bağımlı hale getirmektedir. Bu bağımlılığın ortadan kaldırılması için, nesne yaratım işleminin somut sınıflar belirtilmeden yapılması gerekmektedir.

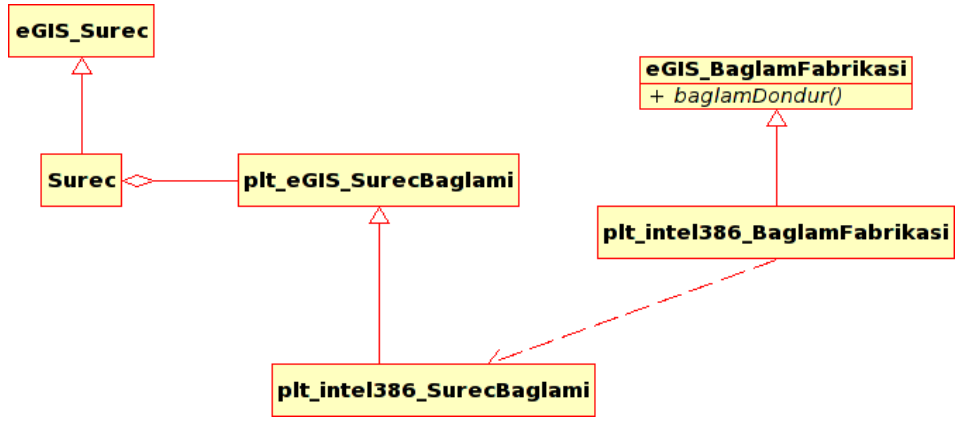
Desenin eGİS İçerisinde Kullanılışı :

eGİS sisteminde ortama bağımlılık, genel arayüzlerin tanımlanması ve bu arayüzü farklı ortamlar için farklı sınıflar ile gerçekleştirilmesi ile ortadan kaldırılmıştır. eGİS sistemi, hangi donanım ortamında oldugunu açıkça bilmeden, tanımlanmış bu arayüzler üzerinden işlemlerini gerçekleştirmektedir. Örneğin şekil 5.9'da gösterilen *Surec* sınıfı, sadece *plt_eGIS_SurecBaglami* arayüzünü görmekte ve gerçekte hangi gerçekleştirimini kullandığını bilmemektedir. Bu gerçekleştirim intel ya da MIPS mimarisi için yapılmış olabilir.



Şekil 5.9 Arayuzler ile platformdan bağımsızlığın sağlanması

Bu arayüzleri gerçekleştiren somut sınıflar, sistemde bir şekilde yaratılmalıdır. Soyut fabrika tasarım deseni, somut sınıflar belirtilerek nesne yaratım işlemlerini soyutlamakta ve sistemin somut sınıflara bağımlılığını ortadan kaldırmaktadır. eGİS sisteminde soyut fabrika deseninin kullanımı şekil 5.10'da gösterilmiştir.



Şekil 5.10 Soyut Fabrika Deseni ve Bağlam Üretimi

Soyut fabrika deseni için arayüz, *eGIS_BaglamFabrikasi* sınıfı ile ortaya konulmuştur. Değişik donanım ortamları için değişik süreç bağlamlarının üretilmesi, *eGIS_BaglamFabrikasi* arayüzünü gerçekleştiren bağlam fabrikası sınıfları ile yapılmaktadır. Hangi donanım ortamında çalışıldığı, *Surec* sınıfı tarafından bilinmemektedir. *Surec* sınıfı sadece *eGIS_BaglamFabrikasi*'nın arayüzündeki **baglamDondur** metodunu görmekte ve kendisine ait bağlamı bu metod ile almaktadır. Sürecin ilgili donanım ortamı ile ilişkilendirilmesi, bağlam fabrikası ile yapılmaktadır.

Desenin Kullanımı Sonucunda Ortaya Çıkan Avantajlar :

Soyut fabrika deseni eGİS sistemi içerisinde şu faydaları sağlamıştır:

- Sistemin somut sınıflardan bağımsız olması sağlanmıştır. Soyut fabrika deseni ile somut sınıflar belirtilerek nesne yaratım işlemi fabrika desenine gömülmüştür. İstemciler sadece ortak nesne arayüzünü görmekte ve gerçekte nesnenin hangi somut sınıf belirtilerek yaratıldığını bilmemektedir. İstemci kodunda somut sınıflara ait bir ifade bulunmamakta, tüm bağımlılık fabrika desenini gerçekleştiren sınıf içine gömülmektedir.

- eGİS sistemi içerisinde farklı bir arayüz gerçekleştirimine sahip nesne, istendiği zaman diğer gerçekleştirimlere sahip nesneler ile yer değistirebilir. Sonuçta yaratım işi desen sınıfına gömüldüğü icin, desen sınıfında istenilen somut sınıfa ait nesne yaratıldığı anda, işleyiş o somut sınıfa ait gerçekleştirim üzerinden yürür. Böylelikle, fabrika deseni kullanımı ile farklı bir somut sınıf tipine sahip nesne yaratıldığı anda, sistem otomatik olarak biçimlenmiş olur.

- Nesneler birbirlerinin gerçekleştirimleri yerine arayüzlerini bilirler. Birbirleri ile olan bağımlılıkları ortadan kalkar.

Desenin Kullanımı Sonucunda Ortaya Çıkan Dezavantajlar :

Soyut fabrika tasarım deseni, nesne yaratım işlemlerinin soyutlanabilmesi için ek bir katman olarak sistemde yer alır. Bu ek katman her nesne yaratımında fazladan bir nesne metodu çağırımını gerektirmektedir. Dolayısıyla bu desen, nesne yaratım işlemlerinin yoğun olduğu noktalarda sistem başarımını belirgin ölçüde düşürebilir. Ancak sistemin genişleyebilirliğini ve taşınabilirliğini oldukça kolaylaştırması nedeniyle gerçek zamanlı sistemlerde kullanılabilecek çok faydalı bir desendir. eGIS sistemi, bu deseni sistem başarımını çok etkilemeyeceği sonucuna varılan ve taşınabilirliği belirgin ölçüde kolaylaştıran süreç ve çalışma ortamı ilişkisini kurmakta kullanmıştır.

Bu desenin nesneleri yaratırken sisteme kaybettirmiş olduğu süre, nesne yaratım isteklerinin ihtiyaç oldukça yapılması ile sistem çalışma zamanına yayılmaktadır. Bunun yerine, sistem açılışında bir kerede ihtiyaç duyulabilecek belirli sayıda nesne yaratılarak, bu işlemlerin genele yayılması engellenebilir. Bu sayede kaybedilecek süre, sadece sistem açılışında kaybedilmiş olur.

5.2.3 eGİS İşletim Sisteminde Kullanılan Yapısal Tasarım Desenleri

Yapısal tasarım desenleri, nesnelerin ve sınıfların, daha büyük ve işlevsel bir yapıyı nasıl oluşturması ve şekillendirmesi gerektiği

problemine çözüm sunmaktadırlar. Yapısal tasarım desenleri arayüzlerin birbirleri ile uyumlu hale getirilmesi, ek özelliklerin sisteme eklenebilmesi ve sistemin değişebilir ve genişleyebilir olmasını, sistemin bağımsızca gerçekleştirilebilir olmasını ve sistemin taşınabilirliğini sağlamaktadır (Gamma et al., 1994).

eGİS işletim sisteminde yer alan yapısal tasarım desenleri *adaptör*, *köprü* ve *ön yüz* tasarım desenleridir.

5.2.3.1 Adaptör Tasarım Deseni

Desenin Kullanım Amacı :

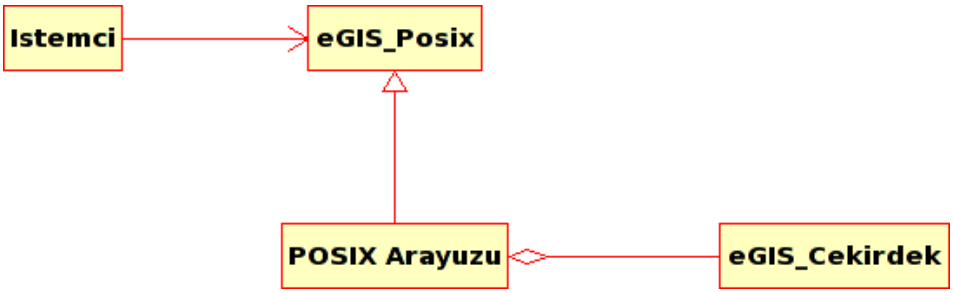
eGİS işletim sisteminin uygulama programları için sunduğu bir C++ arayüzü vardir. eGİS işletim sisteminin sunmuş olduğu servisleri bir uygulama programının kullanabilmesi için bu arayüzleri gerçekleştirmiş olan sınıfları kullanması gerekir.

Ancak başka bir işletim sistemi kullanılarak gerçekleştirilmiş olan gerçek zamanlı sistemler, eGİS sistem arayüzü ile uyumsuz olan servis çağırımlarına sahip olabilirler. Örneğin bazı gerçek zamanlı uygulamalar, POSIX genel gerçek zamanlı işletim sistemi arayüzüne göre yazılmış olabilir ve eGİS birebir POSIX uyumlu arayüzlere sahip değildir. Bu tip uygulamalar eGİS sistemi üzerinde çalıştırılmak istendiğinde, uygulama programlarının kullandığı POSIX arayüzü ile uyumlu servis çağırımlarının, eGİS işletim sistemi çağırımlarına dönüştürülmesi gerekmektedir.

İşte adaptör tasarım deseni, uyumsuz arayüzlerin eGİS işletim sistemi arayüzleri ile uyumlu hale getirilmesini sağlamaktadır.

Desenin eGIS İçerisinde Kullanılışı :

eGIS_ProgramlamaArayuzu paketi içerisinde yer alan *eGIS_Posix* arayüzü, istemciler için genel bir POSIX servis çağırım arayüzü oluşturmaktadır. Bu paket, POSIX çağırımlarını bire bir eGİS işletim sistemi çağırımlarına dönüştürür. Böylelikle, POSIX arayüzü gözetilerek yazılmış olan uygulama programları, eGİS sistemine de taşınabilir.



Şekil 5.11 Adaptör Deseni ve Arayüzlerin Uyumlu Hale Getirilmesi

Şekil 5.11'te gösterilen *POSIX_Arayuzu* sınıfı, POSIX arayüz servis çağırımlarını içeren *eGIS_Posix* arayüzünden türemektedir. Bu sınıf, gelen POSIX tabanlı çağırımları *eGİS_Cekirdek* sınıfı yardımı ile ulaşılan alt sistemlerin çağırımlarına dönüştürmektedir. Bu sayede uygulama tarafında hiçbir kod değişikliği yapılmamakta, sadece ek bir adaptör katmanı ile isteklerin dönüşümü gerçekleştirilmektedir.

Desenin Kullanımı Sonucunda Ortaya Çıkan Avantajlar :

Adaptör tasarım deseni, eGİS işletim sisteminde aşağıda belirtilen noktalarda faydalı olmuştur:

- eGİS işletim sisteminin POSIX ve benzeri genel işletim sistemi arayüzlerini kullanan uygulama programları tarafından da kullanılabilmesini sağlamıştır.

- eGİS işletim sistemi sadece C++ arayüzünü kullanan uygulama programları tarafından değil C dili kullanan sistemler tarafından da kullanılabilmektedir.

- eGİS işletim sisteminin yeniden kullanılabilirliği artmıştır.

- Uygulama programlarının eGİS sistemine taşınırken en az şekilde değişmesi sağlanmıştır.

Desenin Kullanımı Sonucunda Ortaya Çıkan Dezavantajlar :

Adaptör tasarım deseni eGİS'in arayüzlerinin uyumsuz oldugu sistemler için ek bir katmandır. Her işletim sistemi servis çağırımı ek bir katmandan geçerek eGİS arayüzlerine dönüştürülür. Bu da ek bir metod çağırımı ve dönüşüm işlemlerine neden olmaktadır. Dolayısıyla sistem başarımında düşüş yaşanmaktadır. Saf C ile yazılmış bir işletim sistemi ile kıyaslandığında, her ek uyumlandırma işlemi belirgin bir işlem yüküne neden olur. Ancak bu durum yine de çoğu durum için kabul edilebilir düzeydedir.

5.2.3.2 Köprü Tasarım Deseni

Desenin Kullanım Amacı :

eGİS işletim sisteminin donanım bağımsız bir işletim sistemi olması tasarım ana hedeflerinden bir tanesidir. Donanımdan bağımsız bir

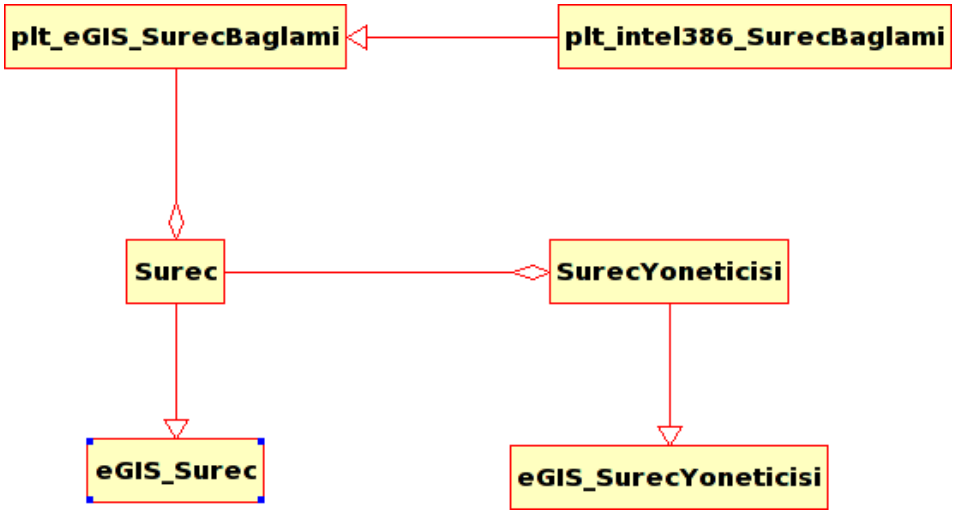
işletim sistemde donanıma ait soyutlamalar ve gerçekleştirimler birbirlerinden ayrılmalıdır.

Örneğin süreç yönetim sistemi, sistemin üzerinde çalışacağı ortama bağlıdır. Bir sürece ait saklanacak bilgi, süreçler arası geçiş ve sistemin kesmeler ile olan ilişkileri hep alt seviye kodlar içerecektir ve bu kodlar her ortam değişiminde yeniden yazılmalıdır. Eğer sistem bu ortama özel gerçekleştirimlere bağımlı hale gelirse, sistemin degişik bir ortama taşınması oldukça zorlaşacaktır. Sistemin bu ayrıştırılma yapılmadan tasarlanmış olması, ortam bağımsız olarak tasarlanabilecek olan kısımların her ortam değişiminde yeniden yazılmasını ve test edilmesini gerektirir.

Desenin eGİS İçerisinde Kullanılışı :

eGİS sisteminde, süreç bağlamları donanımsal ortama bağımlılık içermektedir. Süreçler arası geçişte, yazmaç değerleri gibi donanım ortamı değiştiği zaman değişebilecek bir veri bloğu saklanmalıdır. Ortam değişikliği gerçekleştiği zaman, saklanacak veri bloğundaki değişim sistemi en az şekilde etkilemelidir. Bundan dolayı sistemdeki bu donanımsal bağımlılık bir soyut arayüz olan *plt_eGIS_SurecBaglami* tanımlanarak ortadan kaldırılmıştır. Süreçler sadece bu arayüzü kullanmakta ve hangi donanım ortamında olunduğu bilgisi bilinmeden donanımsal servislerden faydalanmaktadır. Bu durum şekil 5.12'de gösterilmiştir.

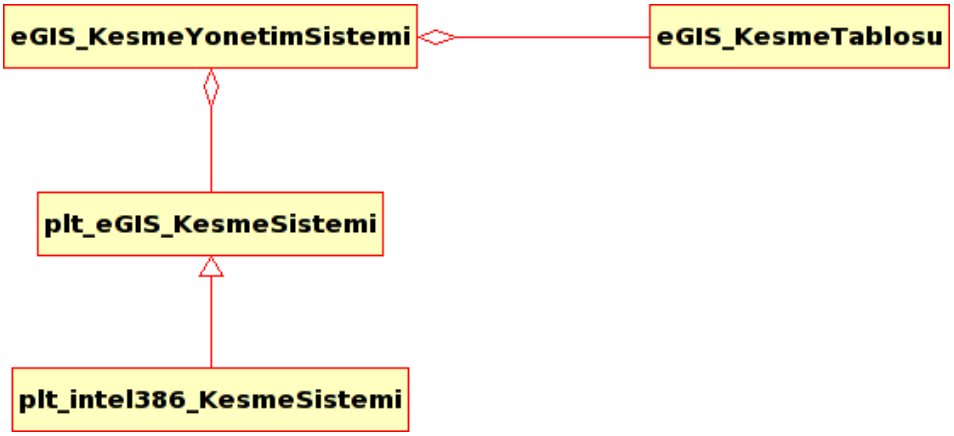
plt_intel386_SurecBaglami sınıfı, intel 386 işlemci ailesi için bağlam arayüzünü gerçekleştiren sınıftır. Eğer sistem yeni bir donanım ortamına taşınırsa, o donanım ortamı için *plt_eGIS_SurecBaglami* arayüzünü gerçekleştiren yeni bir sınıfı yazmak ve bunu sistemle birleştirmek yeterli olacaktır.



Şekil 5.12 Süreç Bağlamları ve Donanımdan Soyutlama

eGİS sisteminde, kesme sistemi de donanımsal bağımlılık içermektedir. Sistemde kesmelerin açılması ve kapanması, kesme vektörü tablosunun değiştirilmesi gibi işlemler hep donanımsal kodlar içermektedir. *plt_eGIS_KesmeSistemi* sınıfı bu donanımsal bağımlılığın ortadan kaldırılması için tanımlanmış bir arayüzdür. Bu durum şekil 5.13'te gösterilmiştir.

plt_intel386_KesmeSistemi ile de intel 386 işlemci ailesi için kesme platformuna özgü donanımsal işlemler gerçekleştirilmişdir. Eğer sistem yeni bir donanım ortamına taşınırsa, o donanım ortamı için *plt_eGIS_KesmeSistemi* arayüzünü gerçekleştiren yeni bir sınıfı yazmak ve bunu sistemle birleştirmek yeterli olacaktır.



Şekil 5.13 Köprü Deseni ve Donanımdan Soyutlanma

Desenin Kullanımı Sonucunda Ortaya Çıkan Avantajlar :

Köprü tasarım deseni, eGİS işletim sisteminde asağıda belirtilen noktalarda faydalı olmuştur :

- eGİS sistemi içerisindeki ortam bağımlı kısımların soyutlanması ve bu kısımların gerçekleştirimi birbirinden ayırılmıştır. eGİS sisteminin, derleme anında hangi ortamda çalışacağı belirlenir ve sistem o ortama özgü sınıf gerçekleştirimleri ile derlenirse, sistem eksiksiz çalışacaktır.

- Soyutlamaların gerçekleştirimindeki değişimler, istemciler tarafında bir değişikliğe neden olmaz. Çünkü istemciler sadece soyut arayüzler üzerinden işlem yapmaktadırlar.

- İstemciler hangi gerçekleştirimin kullanıldığından habersizdirler.

- eGİS sistemi daha da genişleyebilir bir hale gelmiştir. Yeni soyutlamalar ve yeni gerçekleştirimler sisteme kolayca eklenebilir.

- eGİS sisteminin taşınabilirliği kolaylaşmıştır. Başka bir ortam üzerinde çalışmak için, donanım soyutlamalarının taşınacak ortam için gerçekleştirimlerinin yapılması ve bunların derleme anında sisteme kaydedilmesi yeterlidir.

Desenin Kullanımı Sonucunda Ortaya Çıkan Dezavantajlar :

Köprü tasarım deseni, eGİS sisteminde taşınabilirliği ve değişebilirliği sağlamak için bir ek katman olarak düşünülebilir. Sistemin işleyişi, isteklerin donanım bağımlı gerçekleştirimlere iletilmesi ile olmaktadır. Her ortam bağımlı istek, köprü katmanından ilgili ortama dağıtılır. Bu ek katman, sistemin başarımını azaltmaktadır. Köprü deseni kullanılmadan, ortam bağımlı sistem istekleri direkt üzerinde çalışacağı ortama ileteceği için başarım kaybı söz konusu değildir. Köprü deseni ile sistem taşınabilirliği artmış ancak başarımı azaltmıştır. Uygulama tabanlı bir sistem olarak tasarlanan eGİS sisteminde, köprü tasarım deseninin kullanılması kaçınılmazdır. Özellikle, gömülü uygulamaların taşınacağı hedef ortam sürekli değiştiği için, bu desenin başarımdaki yan etkileri çoğu zaman kabul edilebilir olmaktadır. Sistemin hızlı ve kolay taşınabilir olması, gömülü uygulamalar için çok önemli bir avantajdır.

5.2.3.3 Ön Yüz Tasarım Deseni

Desenin Kullanım Amacı :

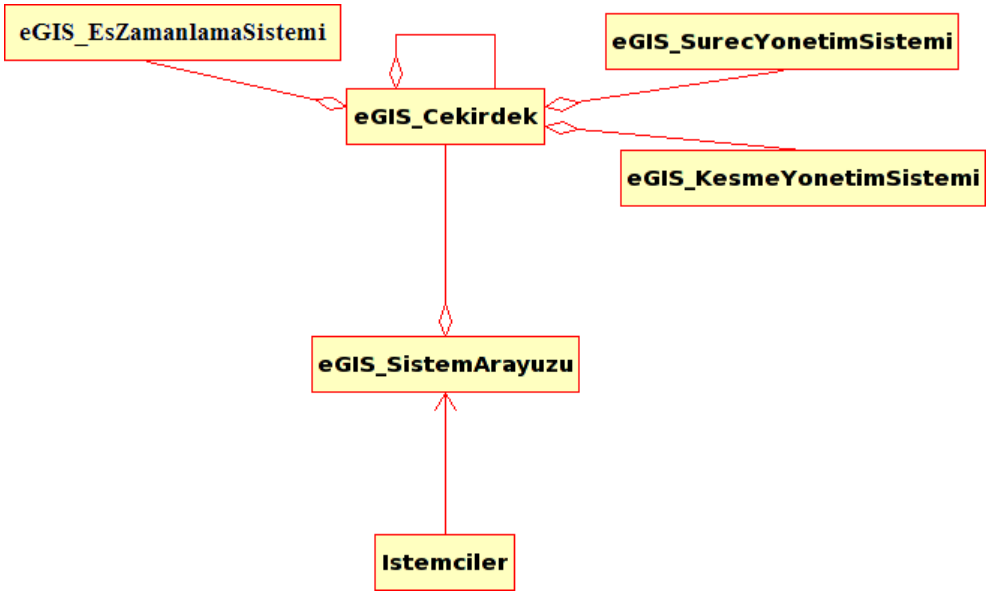
eGİS sistemi çok sayıda sınıf ve soyut arayüzlerden oluşmaktadır. Bu sınıflardan örneklenen nesneler sistemde belirli işlemleri birbirleri ile kurmuş oldukları ilişkilerle yerine getirmektedirler. Ancak, istemcilerin bu karmaşık sistem yapısından haberdar olmamaları gerekir. Sistemin genel ve basit bir arayüzünün olması, kullanım ve bakım kolaylığı sağlamaktadır.

Ön yüz tasarım deseni, sistem için genel ve üst seviye bir programlama arüyüzü sunmaktadır. İstemcilerin içsel sınıflara direkt erişimini engeller, güvenli ve sade bir şekilde sistem servislerinin kullanılmasını sağlar.

Desenin eGİS İçerisinde Kullanılışı :

eGİS sisteminin kullanıcılara sunmuş olduğu üst uygulama geliştirme arayüzü, istemcilerin alt sistemlerden ve sistemin içsel işleyiş mekanizmalarından haberdar olmalarını engeller. İstemciler sadece sistemin arayüzleri görürler, alt sistemlerle direkt ilişki kurmazlar.

eGIS_SistemArayuzu sınıfı, ön yüz tasarım deseninin eGİS sistemindeki uygulanma şeklidir. Şekil 5.14'te görüldüğü gibi, istemciler, sadece bu sınıf ile etkileşim halindedirler. Böylelikle istemcilerin alt sistemlere olan bağımlılıkları ortadan kalkmış, sistem basit ve temiz bir arayüze kavuşmuştur. *eGIS_SistemArayuzu* sınıfı, istemcilerin isteklerini eGIS_Cekirdek sınıfına kaydedilmiş olan alt sistemlere iletir ve bu isteklerin sonuçlarını istemcilere döndürür.



Şekil 5.14 Önyüz Deseninin eGİS Sisteminde Kullanılışı

Desenin Kullanımı Sonucunda Ortaya Çıkan Avantajlar :

Ön yüz tasarım deseni, eGİS işletim sisteminde aşağıda belirtilen noktalarda faydalı olmuştur:

- İstemciler alt sistemlerden ve içsel sınıflardan haberdar değildir. İstemcilerin servislerden faydalanabilmesi için iletişimde bulunması gereken nesne sayısı azalmıştır. Sade ve az sayıda nesne ile iletişime gerek duyulan bir arayüz oluşmuş olmaktadır.

- Alt sistemler ve istemcilerin birbirlerine olan bağımlılıkları azalmıştır. İstemciler alt sistemlere ait gerçekleştirimlerden haberdar değillerdir.

Desenin Kullanımı Sonucunda Ortaya Çıkan Dezavantajlar :

Ön yüz tasarım deseni, diğer desenlerde oldugu gibi ek bir katmandır. İstemcilerin alt sistemlerden bağımsız hale gelmeleri, ek bir katman olan ön yüz tasarım deseni ile sağlanmıştır. Bu ek katman, isteklerin ilgili alt sistemlere iletilmesini sağlamaktadır. Dolayısıyla ek metodların çağırımları başarımı azaltacaktır.

Ayrıca eGİS sistemi içerisindeki sınıf sayısı ve kod büyüklüğü bu ek katman yüzünden artmaktadır. Ancak bu katman sayesinde istemcilerin alt sistemlerle olan ilişkileri ve bağımlılıkları azalmıştır. Uygulama programları, daha temiz ve basit bir arayüz ile eGİS işletim sistemini kullanmaktadırlar.

5.2.4 eGIS İşletim Sisteminde Kullanılan Davranışsal Tasarım Desenleri

Davranışsal tasarım desenleri, sistem içerisindeki algoritmik bağımlılıkların ortadan kaldırılması ve nesnelerin sorumluluklarının ve ilişkilerinin değişebilir olmasını sağlamaktadır. Sistemdeki sınıfların birbirinden bağımsız gerçekleştirilebilmeleri ve nesnelerin birbirleri ile yer değiştirilebilmeleri davranışsal desenlerin sunduğu çözümlerin avantajlarıdır (Gamma et al., 1994).

eGİS sistemi içerisinde yer alan davranışsal tasarım desenleri *inceleyici* ve *strateji* desenleridir.

5.2.4.1 İnceleyici Tasarım Deseni

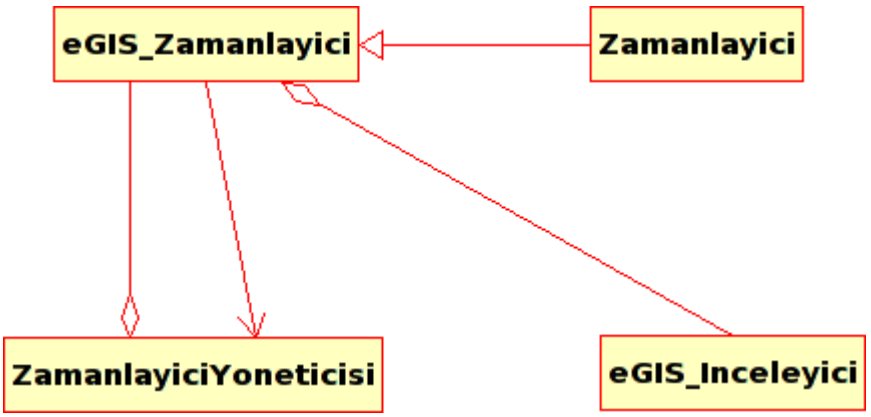
Desenin Kullanım Amacı :

eGİS sistemi içerisinde bazı nesnelerin birbirlerinin durum değişikliklerinden haberdar olması gerekmektedir. Nesneler arasında durum bilgisinin aktarımınının olabilmesi, nesnelerin bazı noktalarda birbirlerine bağımlı olmasına neden olur. İnceleyici tasarım deseni nesnelerin bu bağımlılıklarının ortadan kaldırılmasını sağlamaktadır.

Desenin eGİS İçerisinde Kullanılışı :

eGİS sisteminde, zamanlayıcı nesneleri kurulmaktadır. Kurulan zamanlayıcı nesneleri, süreçleri kurulan zaman sonunda uyarmalıdır. Zamanlayıcılar sadece süreçleri değil sistem içerisindeki eş zamanlama nesnelerini de uyarabilirler. Dolayısıyla zamanlayıcı nesnelerinde olan durum değişikliklerinden sistem içerisindeki diğer nesnelerin haberdar olması ve nesnelerin bu durum değisikliklerine uygun işlemler yapmaları gerekmektedir. Bu nesnelerin birbirine olan bağımlılıkları, inceleyici deseni tarafından ortadan kaldırılmaktadır.

Şekil 5.15'te gösterilen *eGIS_Zamanlayıcı* arayüzünü gerçekleştiren nesneler, *eGIS_Inceleyici* arayüzünü gerçekleştiren her nesneyi kendi durum değişikliklerinden haberdar edebilirler. Bu nesneler birbirlerinin somut gerçekleştirimlerinden haberdar olmazlar. Sadece uygun arayüzün gerçekleştirildiğini bilirler. Böylelikle, bu nesneler arasındaki bağımlılık en aza indirgenmiştir.



Şekil 5.15 Zamanlayıcılar ve İnceleyici Tasarım Deseni

Belirli bir süre sonra uyandırılmak istenen nesneler, *eGIS_Zamanlayici* arayüzünü gerçekleştiren zamanlayıcılar tarafından uyandırılırlar. Tüm uyarılmak istenen nesneler, *eGIS_Inceleyici* arayüzünü gerçekleştirmektedir. Bunun dışında nesneler arasındaki somut bağımlılıklar ortadan kalkmıştır.

eGİS işletim sisteminde zamanlayıcılar BSD zamanlayıcı tekerleği (*timer wheels*) algoritması kullanılarak gerçekleştirilmişlerdir (Costello and Varghese, 1995).

Desenin Kullanımı Sonucunda Ortaya Çıkan Avantajlar :

Ön yüz tasarım deseni, eGİS işletim sisteminde aşağıda belirtilen noktalarda faydalı olmuştur:

- Nesnelerin birbirlerine olan bağımlılıkları ortadan kaldırılmıştır. Uyarılan ve uyaran nesneler birbirlerinden bağımsız olarak gerçeklenebilirler ve genişleyebilirler.

- Bir nesnede meydana gelen durum değişikliği diğer nesnelerde de durum değişikliğine yol açıyorsa, bu işlem otomatik olarak yapılmaktadır.

- Bir nesne diğer nesneleri uyarırken, o nesnelerin somut yapıları hakkında bilgi sahibi olmak zorunda değildir.

Desenin Kullanımı Sonucunda Ortaya Çıkan Dezavantajlar :

İnceleyici tasarım deseni, geribildirim (*callback*) fonksiyonlarının nesneye yönelik programlama paradigması için kullanılan biçimidir (Gamma et al. 1994). Sistemde yapısal programlama ile geribildirim mekanizması kullanılarak yapılan işlemler, eGİS işletim sisteminde inceleyici deseni uygulanarak yapılmaktadır. İnceleyici deseni, sanal fonksiyon mekanizmasının nesneye yönelik programlara kattığı başarım kaybı dışında belirgin bir dezavantaj getirmez.

5.2.4.2 Strateji Tasarım Deseni

Desenin Kullanım Amacı :

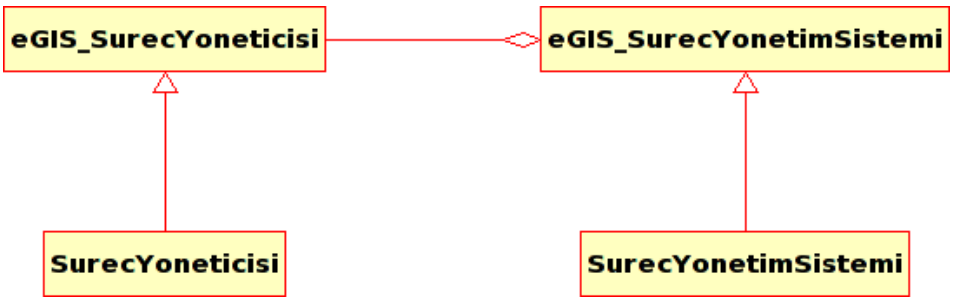
Strateji deseni, sistem içerisinde algoritmik bağımlılıkların ortadan kalkmasını ve algoritmaların kolayca farklı algoritmalar ile

değiştirilebilmesini sağlamaktadır. Bu desen, algoritmaların istemcilerden bağımsız hale gelmesini sağlar.

Desenin eGİS İçerisinde Kullanılışı :

eGİS sistemi içerisinde, sürec yönetimi uygulamanın ihtiyaçlarına ve ortama göre değişebilecek bir noktadır. Dolayısıyla sistemin bu işlem için belirli bir algoritmaya bağımlı hale gelmemesi gerekir.

Şekil 5.16, strateji deseninin eGİS işletim sisteminde kullanılışını göstermektedir. *eGIS_SurecYonetimSistemi*'ne kaydedilen *SurecYoneticisi*, *eGIS_SurecYoneticisi* arayüzünü gerçekleştirmektedir. *eGIS_SurecYonetimSistemi*, *SurecYoneticisi*'nin sadece *eGIS_SurecYoneticisi* arayüzünde belirtilmiş metodlarına erişim yaptığı için, farklı bir uygulama gereksinimine uygun yazılmış, değişik bir algoritma içeren ve *eGIS_SurecYoneticisi* arayüzünü gerçekleştiren bir süreç yöneticisi, eskisi ile yer değiştirilebilir. Bu işlem sistem içerisinde kod değişikliğine yol açmaz ve sistemi etkilememiş olur.



Şekil 5.16 Strateji Deseni ve eGİS Sisteminde Kullanılışı

Desenin Kullanımı Sonucunda Ortaya Çıkan Avantajlar :

Strateji tasarım deseni, eGİS işletim sisteminde aşağıda belirtilen noktalarda faydalı olmuştur:

- Değişik yönetim algoritmaları, eGİS sistemine kolay bir şekilde yerleştirilebilmekte ve var olan algoritmalar zahmetsizce daha farklı algoritmalarla yer değiştirilebilmektedir.

- İstemciler, sistemde hangi algoritmaların kullanıldığından haberdar değildirler. İstemcilerin alt sistemlerin algoritmik gerçekleştirimlerine olan bağımlılıkları ortadan kalkmıştır.

- Algoritmik gerçekleştirimleri içeren sınıfların yeniden kullanılabilirliği artmıştır.

Desenin Kullanımı Sonucunda Ortaya Çıkan Dezavantajlar :

eGİS sistemi, bir uygulamanın ihtiyaçlarına göre değişebilecek ve şekillendirilebilecek bir işletim sistemidir. Dolayısıyla sistem içerisindeki algoritmalar da uygulama ihtiyaçlarına göre değişebilecektir. Değistirilebilir algoritmalar, algoritmik isteklerin strateji katmanından geçerek ilgili algoritma gerçekleştirimine iletilmesi ile sağlanır. Bu da ek katmandaki işlemlerin ve ilgili gerçekleştirimin işletilmesi ile olur. Dolayısıyla bu ek katmandaki işleme ve ek metod çağırımları başarımın

düşmesine neden olmaktadır. Ayrıca sistemdeki nesne sayısında ve bellek kullanımında da artış olmaktadır.

5.2.5 eGİS İşletim Sisteminde Kullanılan Gerçek Zamanlı Sistemlere Özgü Tasarım Desenleri

eGİS işletim sistemi, masaüstü ve benzeri geleneksel bilgisayar sistemlerinden farklı olarak daha kısıtlı bir ortamda çalışmaktadır. Bu nedenle, eGİS sistemi işlemci ve bellek kısıtlarını göz önüne almalı ve bu kaynakları iyi kullanmalıdır. eGİS sistemi birçok gerçek zamanlı sistem gibi güvenilirlik gereksinimlerini de yerine getirmelidir.

Gerçek zamanlı sistemler üzerinde çalışan ve bu uygulama alanında yazılım geliştiren geliştiriciler de, bire bir aynı olmayan ancak çok benzer problemlerle karşılaşmışlar ve bu problemlere gerçek zaman kısıtları altında sunmuş oldukları çözüm yollarını soyutlamışlar ve genelleştirmişlerdir (Douglass, 2002).

eGİS işletim sistemi içerisinde kullanılan gerçek zamanlı tasarım desenleri *mesaj kuyruğu*, *kesme* ve *havuz tabanlı tahsis* desenleridir.

5.2.5.1 Mesaj Kuyruğu Tasarım Deseni

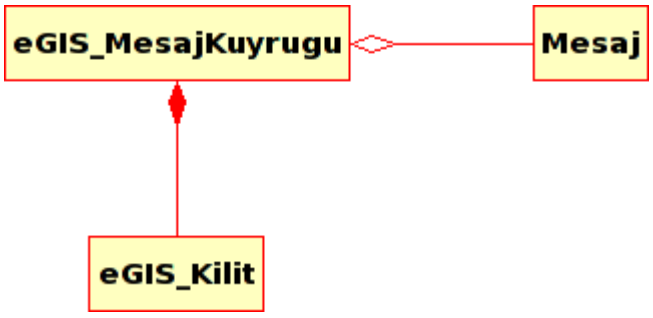
Desenin Kullanım Amacı :

Mesaj kuyruğu tasarım deseni, eGİS işletim sisteminde iş parçacıkları arasında bilgi aktarımını sağlamak için kullanılmıştır. Çoğu çoklu iş parçacığının olduğu sistemde, iş parçacıkları arasında bilgi aktarımı olması, eş zamanlama gibi problemlere yol açmaktadır. Mesaj kuyruğu deseni, iş parçacıkları arasındaki bilgi aktarımı sırasında

oluşacak bozulmaları ve iş parçacıklarının çalışma sırasının değişimi ile oluşacak hatalı durumları ortadan kaldırır (Douglass, 2002).

Desenin eGİS İçerisinde Kullanılışı :

Şekil 5.17'de mesaj kuyruğu deseninin eGİS işletim sistemi içerisindeki kullanımı gösterilmiştir. *eGIS_MesajKuyrugu* sınıfı, süreçler arasında olacak bilgi iletimini sağlar. *eGIS_Kilit* kilidi ile içerisindeki mesajlara güvenli erişim garanti altına alınmış olur. İş parçacıkları mesaj kuyruğunun arayüzünde bulunan mesaj alma ve mesaj postalama arayüzlerini kullanarak, diğer iş parçacıklarının yapısından ve gerçekleştirimlerinden haberdar olmadan onlarla haberleşirler. Haberleşmede kullanılan bilgi biçimi *Mesaj* sınıfında tanımlandığı gibi olmaktadır. Genellikle *Mesaj* sınıfı sistemdeki en temel nesneden türemektedir.



Şekil 5.17 Mesaj Kuyruğu Deseni

Desenin Kullanımı Sonucunda Ortaya Çıkan Avantajlar :

Mesaj kuyrugu tasarım deseni, eGİS işletim sisteminde aşağıda belirtilen noktalarda faydalı olmuştur:

- Bu tasarım deseni ile, hemen her gerçek zamanlı sistemde kullanılan bir mekanizma gerçekleştirilmiştir.

- Süreçler arası basit, güvenli ve en az bağımlılık yaratacak bir iletişim mekanizması sağlanmıştır.

- İş parçacıkları arasında çalışma sırası ve eş zamanlama problemleri ile oluşacak hatalı veri paylaşımı ortadan kaldırılmıştır.

Desenin Kullanımı Sonucunda Ortaya Çıkan Dezavantajlar :

Mesaj kuyrugu tasarım deseni, iş parçacıkları arasındaki veri paylaşımının verimsiz bir şeklidir. Verimli ve hızlı bir veri paylaşımı için uygun değildir. Ayrıca veri kopyalanarak mesaj kuyruğuna taşındığı için verimsiz bellek kullanımı da vardır. Ancak mesaj kuyruğu kullanımı ile süreçlerin çok sade ve genel bir veri paylaşım imkanı sunulmuş olunur. Mesaj kuyrukları, çok yoğun veri paylaşımı yapmayan süreçler için ideal bir mekanizmadır.

5.2.5.2 Kesme Tasarım Deseni

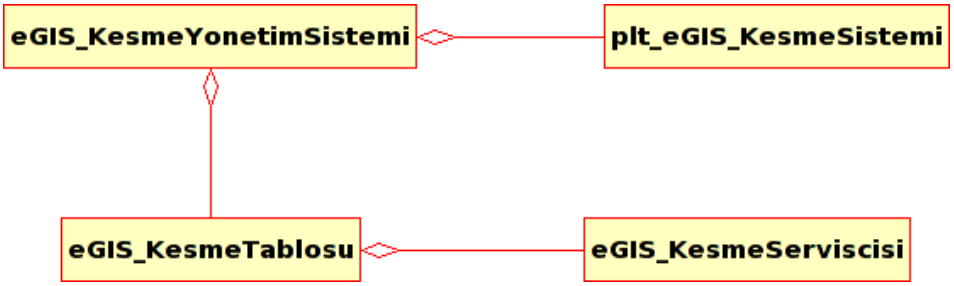
Desenin Kullanim Amacı :

Çoğu gerçek zamanlı sistemde, kesmeler hızlı ve verimli bir şekilde işlenmelidirler. Kesmeler çok hızlı bir şekilde çözümlenmeli ve o kesme olayına doğru bir şekilde cevap verilmelidir. eGİS işletim sisteminde yer alan kesme sistemi, kesme yönetimini bu desen ile yerine getirmektedir. Kesme deseni ile genişleyebilen ve kolayca yönetilebilen bir alt sistem ortaya koyulmuş olur.

Desenin eGİS İçerisinde Kullanılışı :

Şekil 5.18'de kesme deseninin eGİS işletim sistemi içerisinde kullanımı gösterilmiştir. *eGIS_KesmeYonetimSistemi*, kesme takip işlerini *eGIS_KesmeTablosu* sınıfı üzerinden yapmaktadır. Kesmeleri işleyecek olan kesme servisçileri, *eGIS_KesmeServiscisi* arayüzünü gerçekleştirirler ve bu servisçiler kesme tablosuna kaydedilirler. Donanım ortamına özgü kesme işlemleri ise *plt_eGIS_KesmeSistemi* arayüzünü gerçekleştiren sınıflar üzerinden yürütülmektedir.

Sisteme yeni bir kesme servisçisinin eklenmesi ve çıkarılması, *eGIS_KesmeTablosu'na* yeni bir girdi eklemek ve çıkarmak işlemlerini içerir. *plt_eGIS_KesmeSistemi* değiştirilerek diğer sınıflar değiştirilmeden donaım ortamı değiştirilebilir.



Şekil 5.18 Kesme Deseni

Desenin Kullanımı Sonucunda Ortaya Çıkan Avantajlar :

Kesme tasarım deseni, eGİS işletim sisteminde aşağıda belirtilen noktalarda faydalı olmuştur:

- Kesme yönetimi basit, hızlı ve verimli hale gelmiştir.
- Kesme yönetim sistemi donanım bağımsız hale gelmiştir.
- Kesme yönetim algoritması kolaylıkla değişebilir.

Desenin Kullanımı Sonucunda Ortaya Çıkan Dezavantajlar :

Kesme deseni, kesme ekleme ve yükleme işlemleri için ek bir katman olduğu için, bu işlemlerde sistem başarımını düşürmektedir. Ayrıca herhangi bir kesme olayı olduğunda, kesme işleme noktasına gelmek için ek metodların çağırılması gerekir. Bu da çok önemli kesmelerin işlenişinde kabul edilebilecek bir durum olmayabilir. Ancak

bu desen genel bir kesme sistemi tasarımı ortaya koyar ve kesme yönetiminin değişebilirliğini kolaylaştırmıştır.

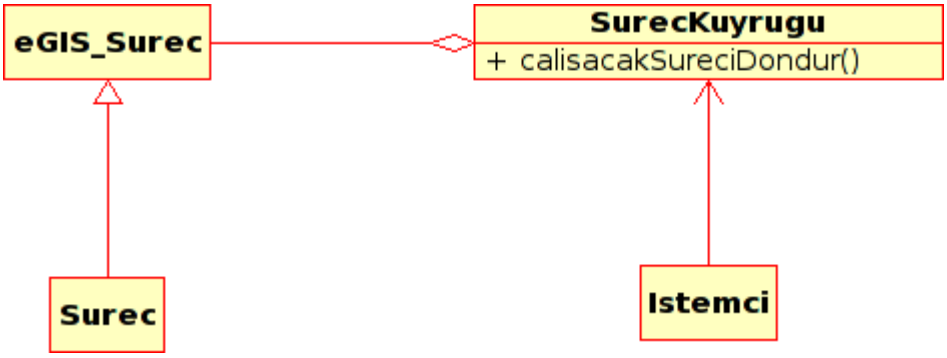
5.2.5.3 Havuz Tabanlı Tahsis Tasarım Deseni

Desenin Kullanım Amacı :

Havuz tabanlı tahsis, gerçek zamanlı sistemlerde çok kullanılan bir desendir. Bazı gerçek zamanlı sistemlerde, durağan bellek tahsisi gerçekleştirilemez çünkü sistemin çalışma zamanındaki bellek ihtiyacı belirlenemez. Dinamik bellek tahsisi ise sistemin kaldırabileceği bir yöntem çoğu zaman değildir. Havuz tabanlı bellek tahsisinde, belirli sayıda nesne sistem başlangıcında yaratılır ve ilklenir. Nesne ihtiyacı olduğunda, nesneler bu havuzdan istemcilere verilir. Nesneler kullanılıp ihtiyaçları karşıladıkları zaman, yine bu havuza geri gönderilirler (Noble and Weir, 2001).

Desenin eGIS İçerisinde Kullanılışı :

eGİS sistemi içerisinde yaratılabilecek en fazla süreç sayısı, uygulama yazılırken belirlenebilir. *SurecKuyrugu* sınıfı sistem açılışında, belirlenen sayıda süreci yaratarak içinde saklar. Sistem çalışırken ortaya çıkan süreç ihtiyacı, *SurecKuyrugu* sınıfının süreç havuzundan sağlanır. Uygulamanın süreçlerle işi bittiği anda, kullanılmayan süreçler yine bu havuza döndürülür. Böylelikle sabit bir bellek bölgesi üzerinden bellek alış verişi yapılmış olur. Bu durum şekil 5.19'da gösterilmiştir.



Şekil 5.19 Havuz Tabanlı Tahsis Deseni

Desenin Kullanımı Sonucunda Ortaya Çıkan Avantajlar :

Havuz tabanlı tahsis tasarım deseni, eGİS işletim sisteminde aşağıda belirtilen noktalarda faydalı olmuştur:

- Dinamik bellek tahsisinde nedeniyle ortaya çıkabilecek tahmin edilebilirlik özelliğinin azalması sorunu ortadan kalkar.
- Bellek parçalanması sorunu ortadan kalkar.
- Yoğun nesne kullanımında bellek daha verimli kullanılmış olur.

Desenin Kullanımı Sonucunda Ortaya Çıkan Dezavantajlar :

Havuz tabanlı tahsis tasarım deseni, eGİS işletim sisteminde sistem açılışının yavaşlamasına neden olmaktadır. Ayrıca eğer başlangıçta ihtiyaç duyulabilecek nesne sayısından fazla nesne yaratılırsa, belleğin atıl kalması sorunu ortaya çıkar. Gereğinden az bellek tahsisi yapılırsa, sistem dinamik olarak bellek tahsis etmeye başlayacak ve tahmin edilemez sistem davranışı ortaya çıkabilecektir (Noble and Weir, 2001; Douglass, 2002).

Ancak bu desen, dinamik bellek ihtiyacı olan sistemlerin bellek gereksinimleri karşılanırken, tahmin edilebilir bir davranışa sahip olmalarını sağlar ve çoğu gerçek zamanlı sistemde kullanılan bir bellek tahsis yöntemidir.

5.2.6 eGİS Sisteminde Uygulanan Tasarım Desenlerinin Getirdiği Sonuçlar

Tasarım desenlerinin gerçek zamanlı ve gömülü bir işletim sistemi olan eGİS'te uygulanması, diğer genel yazılım sistemlerine tasarım desenlerinin uygulanması ile kıyaslandığında az da olsa farklar göstermektedir.

Gerçek zamanlı sistemlerin tahmin edilebilir olmaları gerekliliği ve belirli zaman kısıtlarına sahip olmaları, uygulanan tasarım desenlerinin iyi ve verimli bir şekilde kodlanmasını gerektirir. Bazı noktalarda uygulanabilecek desenler varken, bu desenlerin kullanılması sistem başarımını çok olumsuz yönde etkileyebileceği için uygulanamamıştır. Örneğin *sanal makine* tasarım deseni, sistemin donanım bağımsızlığını sağlamakta kullanılabilecek başka bir desen iken, çoğu gerçek zamanlı sistemde uygulanamaz.

Gömülü sistemlerin bellek ve işleme kısıtları vardır. Bu kısıtlar da buraya özgü tasarım desenlerinin doğmasına olanak sağlar ve var olan desenlerin daha değişik bir şekilde gerçekleştirilmesini gerektirir. Örneğin bellek yönetimi için kullanılmış olan *havuz tabanlı tahsis* deseni, genel sistemlere göre eGİS sistemi için daha farklı bir şekilde kodlanmıştır. Verim ve başarım daha ön plandadır.

Tasarım desenleri, eGİS'in diğer sistemlere kıyasla daha değişebilir olmasını sağlarken başarım açısından da daha kötü olmasına neden olmuştur. Özellikle daha güçlü işlemciler üzerinde çalışan ve diğer ufak sistemlere göre daha az bellek sorunu olan sistemlerde, eGİS iyi bir başarım sağlayacak ve sistemin kısıtlarını yerine getirecektir. Bu sistemlerde eGİS'in farkı daha iyi ortaya çıkacaktır. Uygulama isteklerine göre kolayca şekillendirilebilir olacağı için, diğer sistemlerin taşıma, genişleme ve bakım gibi eGİS sisteminde yaşanmayacaktır.

6 SONUÇ

Günümüzde işletim sistemleri denince akla gelen şey genellikle genel bilgisayar sistemleri üzerinde çalışan ve uygulama programları ile donanım arasında bir katman olarak görev alan bir yazılım katmanı olmaktadır. Ancak gün geçtikçe gömülü sistemler gibi adanmış sistemler için üretilen işlemci sayısı artmakta, bu sistemler üzerine geliştirilen yazılımlar daha da önemli hale gelmektedir.

Adanmış sistemler özel uygulama amaçlarına göre gerçekleştirilirler ve genellikle sistem gereksinimleri sistem geliştirilirken belirlidir. Dolayısıyla masaüstü ve benzeri uygulamalar için geliştirilmiş genel işletim sistemleri, adanmış sistemler için uygun değildir. Genel olan çözümler genellikle en uygun çözümler değildir. Çünkü her uygulamanın kendine göre gereksinimleri vardır ve ancak uygulama özelleşmiş bir işletim sistemi genel işletim sisteminin çözemediği problemleri çözebilir.

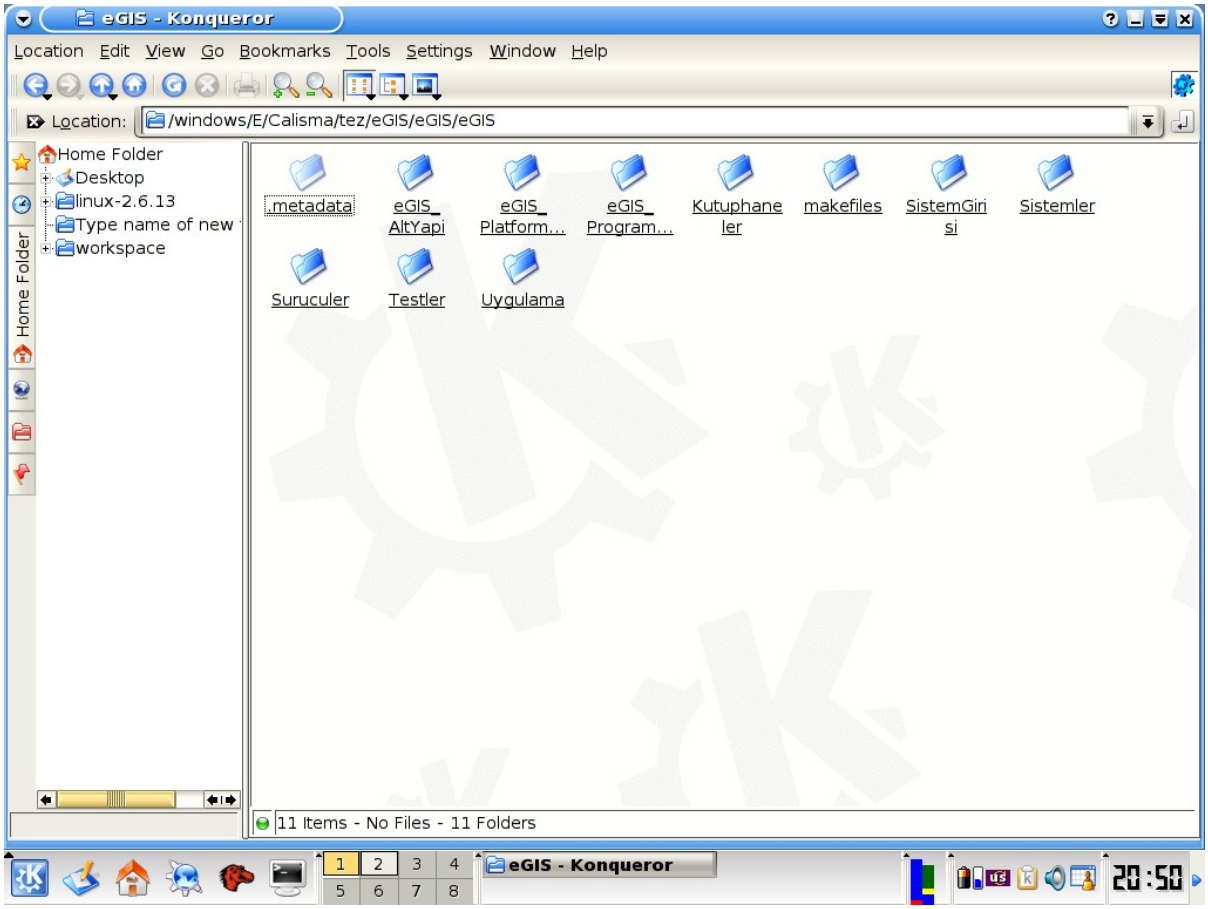
Adanmış sistemler için tasarlanmış işletim sistemlerinin çoğu değiştirilebilir, yeniden şekillendirilebilir ve taşınabilir değildir. Yeniden şekillendirilebilen ve uygulama ihtiyaçlarına göre değiştirilebilen bir işletim sistemi, adanmış sistemler için en uygun olanıdır. Uygulamaya yönelik olan bu işletim sistemi, uygulamanın özelleşmiş problemlerini çözebilir ve işlevselliği sağlayabilir.

Bu bakış açısı, gerçek zamanlı gömülü işletim sistemlerinin kendilerine özgü bir mimariye ve tasarım yaklaşımlarına sahip olmaları sonucunu doğurur. eGİS işletim sistemi, kendine özgü ve tasarım desenlerini temel alan tasarımı ile diğer işletim sistemlerinden ayrılmaktadır.

6.1 Gerçekleştirim

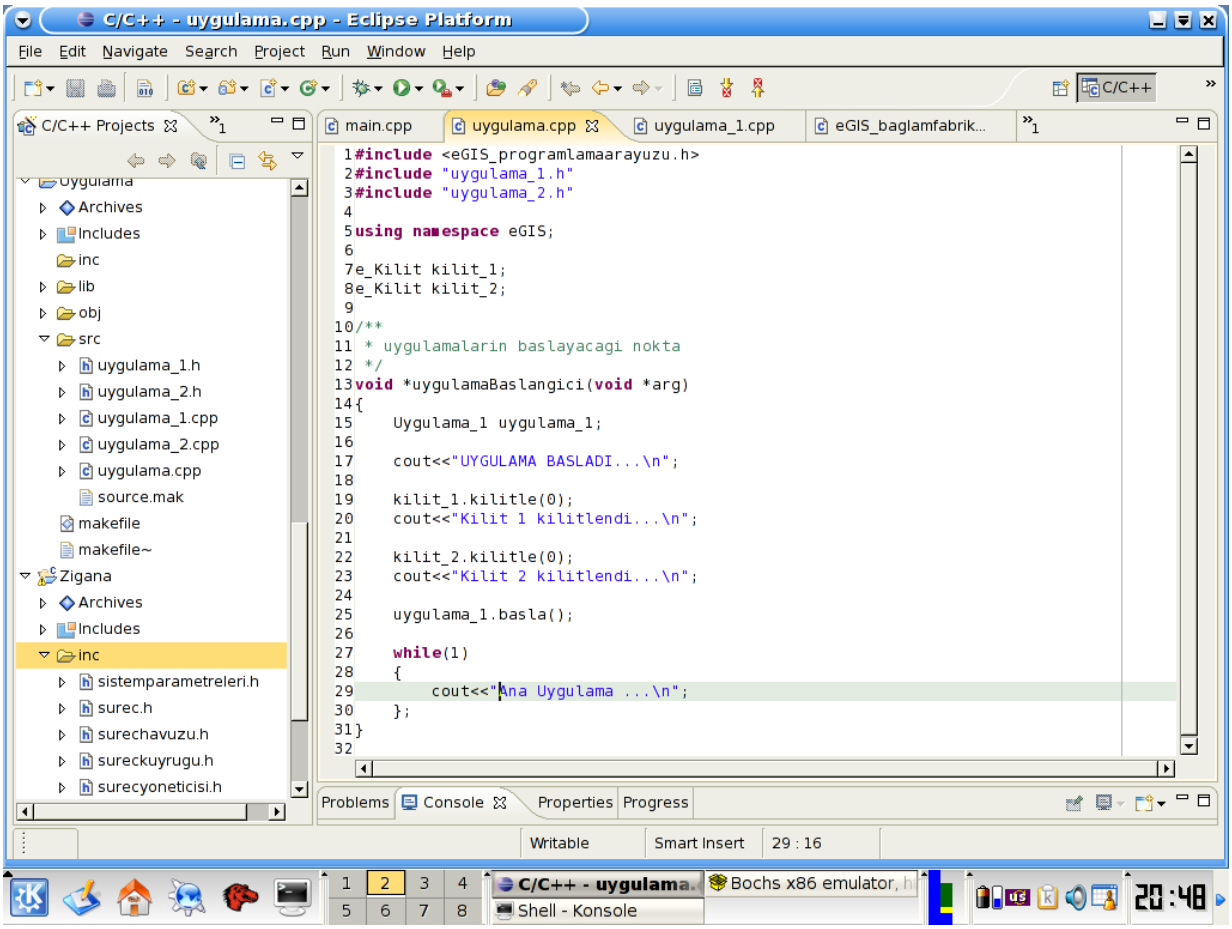
eGİS işletim sistemi Eclipse uygulama geliştirme ortamı, Gcc derleyicisi, Gdb hata ayıklayıcısı, Ld bağlayıcısı ve Bochs emülatörü kullanılarak gerçekleştirilmiştir.

eGİS'in modüler ve katmanlı mimarisi, dizin yapısına da yansımaktadır. Tüm katmanlar ve alt sistemler ayrı ayrı dizinler içerisinde tutulmakta ve alt sistemler derleme dosyası içerisine yolları verilerek uygulamaya bağlanmaktadır. Bu dizin yapısı içerisine yeni alt sistemler eklenebilir. eGİS'in dizin yapısı şekil 6.1'de gösterilmektedir



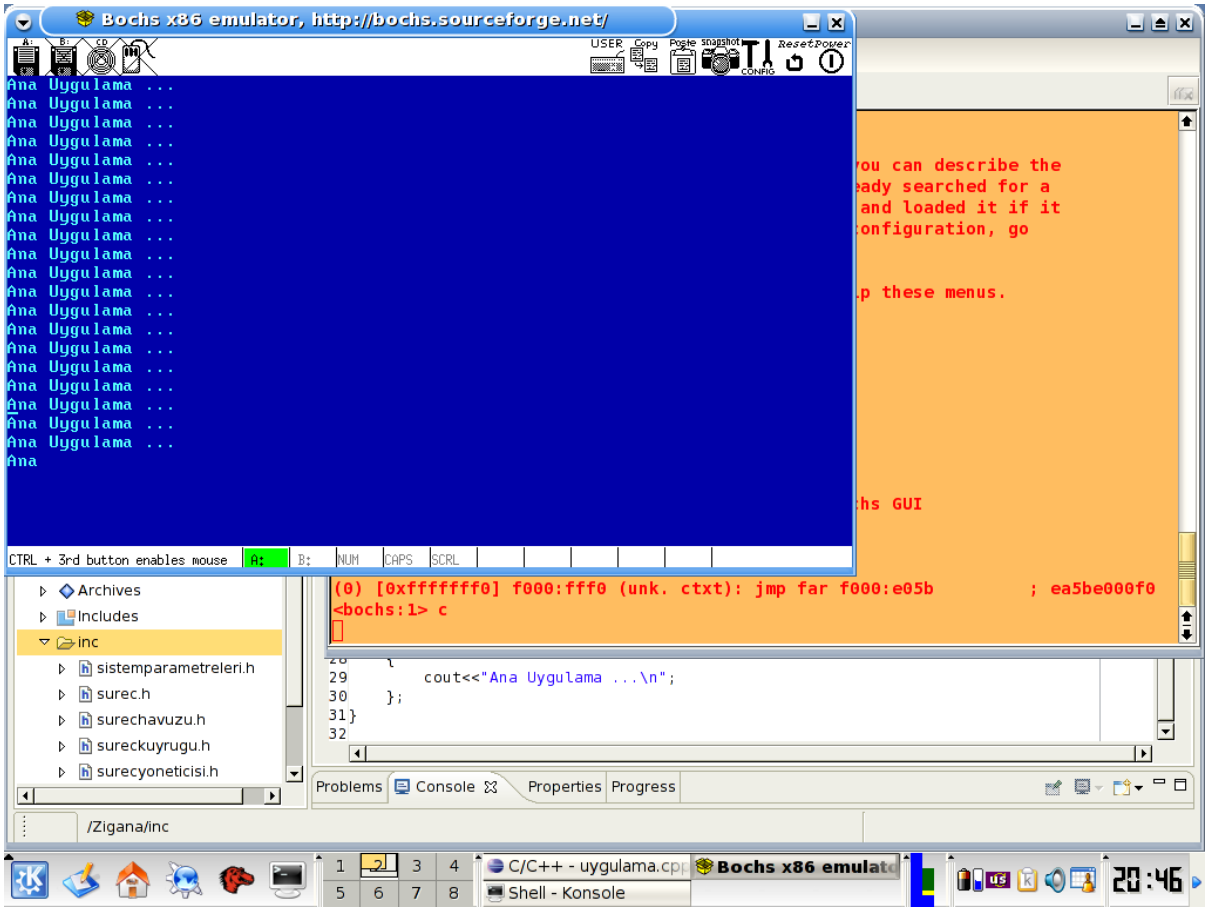
Şekil 6.1 eGİS'in Dizin Yapısı

Uygulama programları, eGİS işletim sisteminin sunduğu servislerden ilgili alt sistemlerin ne olduğunu bilmeden, sadece programlama arayüzü paketini görerek faydalanmaktadır. Bir uygulama programı, Eclipse ortamında şekil 6.2'deki gibi kodlanabilmektedir.



Şekil 6.2 eGİS'i Kullanan Bir Uygulama Programı

eGİS işletim sistemini kullanan uygulana programı, eGİS ile bağlandıktan sonra, donanım üzerinde doğrudan çalıştırılabilecek ikili kod oluşturulmuş olmaktadır. Bu ikili kod Bochs emülatörü üzerinde de çalıştırılabilir. Şekil 6.3'te ikili uygulama kodunun Bochs emülatörü üzerinde çalışması gösterilmektedir.



Şekil 6.3 eGİS'i Kullanan Bir Uygulama Programının Bochs Emülatörü Üzerinde Çalışması

6.2 eGİS İşletim Sisteminin Değerlendirilmesi

eGİS işletim sistemi modern yazılım kavramları göz önüne alınarak gerçekleştirilmiş bir mikroçekirdektir. Gerçekleştirilen bu çalışma ile, temel bir gerçek zamanlı işletim sistemi ortaya konulmuştur. Bu işletim sistemi, gerçek zamanlı yazılım sistemlerinde doğrudan kullanılabilir, uygulama ihtiyaçlarına göre genişleyebilir ve yeni bileşenler eklenebilir bir sistemdir.

Gömülü sistemler, çalışma zamanı gereksinimleri belirlenmiş ve özelleşmiş yazılımları içerirler. Dolayısıyla, gömülü sistemler için tasarlanmış olan işletim sistemleri uygulama istek ve kısıtlarının zorladığı değişimlerin kolay yapılabildiği sistemler olmalıdır. eGİS bu nokta göz önüne alınarak tasarlanmıştır.

eGİS işletim sistemi içerisinde mümkün olduğunca arayüzler iyi bir şekilde tasarlanmış ve bu arayüzlerin gerçekleştirimleri ayrıştırılmıştır. Arayüzlerin ve gerçekleştirimlerin birbirinden ayrılması, arayüze göre gerçekleştirimin yapılması, uygulamanın gereksinim ve kısıtlarına uygun olan ve aynı arayüzleri gerçekleyen alt sistemlerin birbiri ile değiştirilebilmesini sağlar. Bu da eGİS'in birbirinden bağımsız ve farklı gereksinimlere sahip sistemler tarafından kullanılabilmesini sağlar.

İşlemci ve bellek yönetim mekanizmaları, gerçek zamanlı sistemlerde değişebilen temel noktalardan bazılarıdır. Değişebilen benzeri noktaların tasarımlarında tasarım desenlerinin kullanılması, eGİS işletim sisteminin genişleyebilirliğini ve taşınabilirliğini arttırmıştır. eGİS karmaşıklaşıp büyüdükçe, yeniden kullanılabilen ve genişleyebilen, birbiri ile daha az etkileşimli ve iyi ayrıştırılmış alt sistemlerin faydası daha da görülecektir.

Gömülü işletim sistemi tasarımında mümkün olduğunca sınıf kalıtımından kaçınılması gerekmektedir. Sınıf kalıtımı sistemin genişleyebilirliğini ve alt sistemlerin birbirleri ile olan bağımsızlığını azaltmaktadır. Zaten tasarım desenlerinin sunmuş olduğu tasarım şablonları da sınıf kalıtımı yerine içerme mekanizmasının kullanımını teşvik etmektedir (Gamma et al., 1994). eGİS işletim sistemi de arayüz kalıtımına dayanan alt sistemler içermektedir.

Bilgisayar mühendisleri gerçek zamanlı sistemlerle çok nadir ilgilenmektedir. Bu alanda yazılım geliştiren genellikle elektronik mühendisleridir. Bu da modern yazılım kavramlarının bu sistemlere doğru bir şekilde uygulanmasını zorlaştırmaktadır. Ayrıca elektronik mühendisleri bilgisayar bilimleri, eş zamanlama, nesneye yönelik tasarım, işletim sistemleri konusunda yeterli bilgi ve tecrübeye sahip olmamaktadırlar. Bu tip kavramların bir gerçek zamanlı yazılım sisteminde uygun kullanılamaması ve modern yazılım tekniklerinden

uzak bir çözüm yolunun izlenmesi, yazılımın kalitesini düşürmektedir. Kaliteli bir gerçek zamanlı yazılım, verimli ve hızlı olmasının yanında bakım ve genişleme konularında iyi bir mimariye de sahip olmalıdır.

Tasarım desenleri gömülü işletim sistemlerin özel tasarım problemlerini çözebilir ve nesneye yönelik tasarımları genişleyebilir, yeniden kullanılabilir ve daha düzenli hale getirir. Günümüzdeki gerçek zamanlı ve gömülü yazılım sistemleri, mimarisel ve genel tasarım desenleri ve bu desenlerin işaret ettiği tasarım problemleri göz önüne alınarak tasarlanılmalıdırlar.

Gömülü sistemleri diğer yazılımlardan ayıran özellikleriden bir tanesi de sistem kaynaklarının sınırlı olmasıdır. Sistem kaynaklarının daha verimli kullanılması temel amaçlardan bir tanesi olmalıdır. Verimli bir sistemin daha çok yapısal programlamadan geçtiği ve C ile yazılması gerektiği gibi bir ön yargı vardır. Ancak C++ dili kullanılarak da çok verimli nesneye yönelik programlar yazılabilir (Meyers, 2002). Ayrıca çoğu noktada nesneye yönelik programlama ve tasarım desenlerinin kattığı ek yük kabul edilebilir olmaktadır. Dolayısıyla bu kavramların kullanılması sistemin gelişimini olumlu yönde etkileyecektir.

eGİS işletim sisteminin, Türkiye'de açık kaynak koda sahip ve GNU lisansı ile diğer mühendislerin de katkılarına açık bir ulusal gömülü işletim sistemi olması umulmaktadır. Tıpkı eCOS işletim sistemi gibi, eGİS de diğer mühendislerin katılımı ile, daha hızlı ve sağlam bir genişleme imkanına sahip olacaktır.

Sonuç olarak yapılan çalışma ile, bu konu hakkında çalışma yapmak isteyenler için bir kaynak ortaya konulmuştur. Bu kaynağın, işletim sistemleri ve gömülü sistemler ile ilgilenenlere faydalı olacağı düşünülmektedir.

6.3 Gelecek Çalışmalar

Gerçekleştirilen işletim sistemi çekirdeği, büyük bir işletim sistemi için başlangıç noktası teşkil edebilir. İleride geliştirilmek istenirse,

Türkiye'deki bilgisayar mühendisleri tarafından ortaya konacak temel bir gömülü işletim sistemi oluşturulabilir.

Ayrıca sistem programlama ve işletim sistemleri konusunda çalışma yapmak isteyenler de gerçekleştirilen koddan faydalanabilirler. eGİS İşletim sisteminin başarımın arttırılması, TCP/IP ve dosya sistem bileşenlerinin eklenmesi, yeni yönetim algoritmalarının eklenmesi, eGİS mimarisinin daha da iyileştirilmesi ve yeni çekirdek mekanizmalarının eklenmesi; ileriki çalışmalarda gerçekleştirilebilecek noktalar olabilir.

EK 1 TERİMLER SÖZLÜĞÜ

Abstract Factory : Soyut fabrika

Adaptor : Adaptör

Application Oriented : Uygulamaya yönelik

Assembly : Birleştirici dili

Background/Foreground Systems : Arkaplan/Önplan sistemler

Black Box : Kapalı kutu

Bridge : Köprü

Context : Bağlam

Creation : Yaratım

Cross Compiler : Çapraz derleyici

Cross Development Environment : Çapraz geliştirme ortamı

Component : Bileşen

Design Pattern : Tasarım deseni

Determinism : Tahmin edilebilirlik

Dynamic Binding : Dinamik bağlama

Embedded System : Gömülü sistem

Facade : Ön yüz

FCFS : İlk gelen ilk çalışır

Global Object : Genel Nesne

Hard Real Time System : Kesin zaman kısıtlarına sahip gerçek zamanlı sistem

Hardware / Software Codesign : Yazılım/ Donanım işbirlikçi tasarımı

Idiom : Deyim

Inheritance : Kalıtım

Interface : Arayüz

Interrupt : Kesme

TERİMLER SÖZLÜĞÜ (devam)

Kernel : Çekirdek

Microkernel : Mikroçekirdek

Memory Management Unit : Bellek yönetim birimi

Mutex : Kilit

Non-preemptive : Kesilemeyen

Observer : İnceleyici

PDA : Taşınabilir dijital asistan

Performance : Başarım

Polymorphism : Çok yapılılık

Pool Based Allocation : Havuz tabanlı tahsis

Preemptive : Kesilebilir

Process : Süreç

Pure Virtual Function : Saf sanal fonksiyon

Semaphore : Semafor

Sequential Fit : Ardışıl bellek atama

Singleton : Tekil

Soft Real Time System : Esnek zaman kısıtlarına sahip gerçek zamanlı sistem

Strategy : Strateji

Structural : Yapısal

Target Environment : Amaç ortam

Virtual Function : Sanal fonksiyon

White Box : Beyaz kutu

Workshop : Çalıştay

KAYNAKLAR DİZİNİ

Accelerated Technology, 2001, Nucleus Plus RealTime Operating System, *www.acceleratedtechnology.com/embedded/nuc_rtos.html*

Bach, M.J., 2001, The Design Of The Unix Operating System; *Prentice Hall*, 471p

Beuche, D., Guerrouat, A., Papajewski, H., Schröder-Preikschat, W., Spincysk, O., Spinczyk, U., 1999, The PURE Family of Object-Oriented Operating Systems for Deeply Embedded Systems; In *Proceedings of 2nd IEEE International Symposium on Object-Oriented Realt-Time Distributed Computing (ISORC'99)*

Blunden, B., 2003, Memory Management Algorithms and Implementation in C/C++; *Wordware Publishing*, 354p

Bricker, A., Gien, M., Guillemont, M., Lipkis, J., Orr, D., Rozier, M., 1991, A New Look at Microkernel-Based UNIX Operating Systems: Lessons in Performance and Compatibility; In *Proceedings of the EurOpen Spring'91 Conference*

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., 1996, Pattern – Oriented Software Architecture: A System Of Patterns; *John Wiley & Sons*, 467p

Campbell, R.H., Johnston, G.M., Madany, P.W., Russo, V.F., 1991, Principles of Object-Oriented Operating System Design

Costello, A.M., Varghese, G., 1995, Redesigning the BSD Callout and Timer Facilities; In *Proceedings of the EurOpen Spring'91 Conference*

Crowley, C., 1997, Operating Systems: A Design-Oriented Approach; *Irwin*, 844p

KAYNAKLAR DİZİNİ (devam)

Red Hat, Inc, 2003, eCOS: Embedded Cygnus Operating System, *eCOS Reference, http://ecos.sourceware.com*

Douglass, B.P., 2002, Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems; *Addison-Wesley*, 332p

Fröhlich, A.A., Schröder-Preikschat, W., EPOS: an Object-Oriented Operating System; In *Proceedings of the Workshop on Object-Oriented Technology, Lecture Notes In Computer Science Volume 1743*

Fröhlich, A.A.M., 2001, Application-Oriented Operating Systems; *Sankt Augustin: GMD-Forschungszentrum Informationstechnik*, 200p

Gamma, E., Helm, R., Johnson, R. and Vlissides, J., 1994, Design Patterns: Elements of Reusable Object-Oriented Software; *Addison-Wesley Professional Computing Series*, 395p

Gay, D., Levis, P., Culler, D., 2004, Software Design Patterns for TinyOS

Gien, M., 1991, Next Generation Operating Systems Architecture; In *Proceedings of the International Workshop on Operating Systems of the 90s and Beyond*

Gien, M., Grob, L., 1992, Micro-kernel Based Operating Systems: Moving UNIX onto Modern System Architectures; In *Proceedings of the UniForum'92 Conference*

KAYNAKLAR DİZİNİ (devam)

Haack, U., Schröder-Preikschat, W., Schön, F., Spinczyk, O., 1998, Design Rationale of the PURE Object-Oriented Embedded Operating System; In *Proceedings of The International IFIP WG 10.3/WG 10.5 Workshop on Distributed and Parallel Embedded Systemss*

Hartig, H., Hohmuth, M., Liedke, J., Schönberg, S., Wolter, J., 1997, The Performance Of µ-Kernel-Based Systems. In *16th ACM Symposium on Operating Systems Principles (SOSP '97)*

Kiczales, G., Lamping, J., 1993, Operating Systems: Why Object-Oriented?; In *Proceedings of The Third International Workshop on Object Orientation in Operating Systems*

Labrosse, J.J., 2003, MicroC/OS-II : The Real-Time Kernel Second Edition; *CMP Books*, 604p

Laplante, P.A., 1997, Real-Time Systems Design and Analysis: An Engineer's Handbook; *IEEE Press*,361p

Lee, E.A., 2000, What's Ahead for Embedded Software?; *Computer, v.33 n.9 p.18-26, IEEE Computer Society Press*

Levchenko, K., A Survey of Microkernel Operating Systems.; *http://www.cs.ucsd.edu/classes/fa01/cse221/projects/index.html*

Liedke, J., 1995, On µ-Kernel Construction; In *Proceedings of SIGOPS'95 Conference*

Liedke, J., 1996, Towards Real Microkernels; In *Communications of The ACM Vol. 39 No. 9 p.70-77*

KAYNAKLAR DİZİNİ (devam)

Li, Q., Yao, C., 2003, Real-Time Concepts for Embedded Systems; *CMP Books*, 294p

Meyers, **S.**,2002, Effective C++ : 50 Specific Ways to Improve Your Programs and Designs; *Addison-Wesley*, 256p

Noble, J., Weir, C., 2001, Small Memory Software: Patterns for systems with limited memory; *Addison-Wesley,* 333p

Shalloway, A., Trott, J.R., 2000, Design Patterns Explained: A New Perspective on Object-Oriented Design; *Addison-Wesley*, 334p

Tanenbaum, A.S., 2001, Modern Operating Systems; *Prentice Hall*, 952p

ÖZGEÇMİŞ

Kasım Sinan YILDIRIM 1981 yılında Mersin'de doğdu. 1998 yılında Bartın Anadolu Lisesi'ni, 2003 yılında ise Ege Üniversitesi Bilgisayar Mühendisliği Bölümü'nü bitirdi. 2003 yılından itibaren Ege Üniversitesi Bilgisayar Mühendisliği Bölümü'nde yüksek lisans eğitimine ve aynı yıl Beko Elektronik İzmir Ar-Ge'de yazılım mühendisi olarak çalışmaya başlamıştır.