*eGİS: Gömülü Sistemler İçin Tasarım Desenleri Tabanlı, Nesneye Yönelik ve Gerçek Zamanlı Bir Mikroçekirdek

K. Sinan YILDIRIM Aylin Kantarcı
Bilgisayar Mühendisliği Bölümü, Ege Üniversitesi, 35100, Bornova, İzmir, Türkiye
sinan.yildirim@mail.ege.edu.tr
aylin.kantarci@ege.edu.tr

Özet

*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) geliştirilmiştir. Bunun yanı sıra, eGİS mimarisindeki esnekliğin ve değişebilirliğin sağlanmasında kullanılan tasarım desenlerinin sisteme kattığı faydalar ve getirdiği yan etkiler 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 konulmuştur.*

1. Giriş

Bilgisayar araştırmacıları uzun bir süre gömülü yazılıma olan ilgilerini düşük tutmuştur. Yazılımın küçük olması ve genellikle birleştirici dili ile yazılması; ayrıca sistemin donanım maliyetinin yazılım maliyetine göre fazla oluşu, gömülü sistemler için yazılım geliştirmenin temel sorunları olmuştur. Yazılım mühendisliğinin sunduğu çözümlerin gömülü sistemler için çok pahalı görünmesi ve bu alana uygulanabilir olmaması, gömülü sistemlerdeki yazılımsal çalışmaları ikinci plana atmıştır. Ancak, günümüzde donanımın daha ucuz hale gelmesi, buna ek olarak yazılımın daha büyük ve karmaşık bir hal alması nedeniyle gömülü sistemlerdeki yazılım araştırmaları tekrar popüler hale gelmiştir [1].

Günümüz gerçek zamanlı sistemleri giderek daha büyük ve karmaşık yazılımlar içermektedirler. Artık gömülü sistemler birleştirici dili ile kolayca yazılabilen ve tasarlanabilen yazılım sistemleri değildirler. Yeni mimarilere taşınabilme, değişen piyasa koşullarına göre alınan değişim isteklerine duyarlı olabilme ve çok değişken gereksinimlere sahip sistemlerde de yeniden kullanılabilme gibi özellikler gömülü yazılımlarda da modern yazılım mühendisliğinin ortaya koymuş olduğu kavramların göz önünde bulundurulmasını gerektirmektedir.

Modern gömülü yazılım sistemlerinin sahip olması gereken temel özellikler şu sekilde sıralanabilir [2] :

- Güçlü 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 ve uygulama ihtiyaçlarına göre yapılandırılabilme
- Kaynakların verimli kullanılması

Gömülü sistem yazılımlarının yukarıda belirtilen özelliklere sahip olması, kendini ispatlamış ve başarıya ulaşmış kaliteli tasarımların yeniden kullanılmasına dayanan tasarım desenlerinin [3] bu sistemlere de uygulanması ile gerçekleştirilebilir. Mevcut başarıya ulaşmış tasarımları kullanmak, uygulamaya, alanına özel problemlerin çözümü için iyi bir başlangıç yapmayı sağlar ve ortaya çıkabilecek hataları engeller.

Tasarım desenlerinin bir sistemde etkileyebileceği noktalar şu şekilde sıralanabilir [4]:

- Sistem başarımı
- Tahmin edilebilirlik
- Planlanabilirlik
- Verim
- Güvenilirlik

* Bu çalışma TÜBİTAK-BAYG (Türkiye Bilimsel ve Teknolojik Araştırma Kurumu) tarafından desteklenmiştir.

- Yeniden kullanılabilirlik
- Dağıtıklık
- Taşınabilirlik
- Bakım
- Genişleyebilirlik
- Karmaşıklık
- Kaynak kullanımı
- Enerji tüketimi
- Donanım maliyeti
- Sistem geliştirme emeği ve maliyeti

Görüldüğü gibi tasarım desenlerinin sunduğu çözüm yolları, getirdikleri avantajların yanında, özellikle gömülü sistemler için oldukça önemli noktalarda sistemi olumsuz yönde etkileyebilmektedir. Örneğin, 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. Dolayısıyla gömülü 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.

Gömülü sistemler için gerçekleştirilen işletim sistemleri, gömülü sistemler üzerinde çalışacak diğer yazılımlar gibi, 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. Gömülü işletim 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 ortamlarda çalışma gerekliliği ve çok değişik istek ve kısıtlara sahip olduğu düşünülürse, genel ve bu isteklere göre değiştirilip uyarlanabilecek bir işletim sistemi mimarisi büyük bir ihtiyaç olarak ön plana çıkmaktadır.

Gömülü işletim sistemlerinde tasarım desenlerinin kullanılması, işletim sisteminin uygulama gereksinimlerine göre değiştirilip yeniden yapılandırılmasını 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 ve alt sistemlerin kendi aralarında değişebilirliğinin sağlanmasında tasarım desenleri önemli yer tutar.

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 kodun 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 azaltılabilir.

Bir işletim sisteminin tasarımı sırasında, diğer bir önemli nokta ise monolitik ve mikroçekirdek mimariler arasında doğru bir seçim yapabilmektir.

Monolitik [5] 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 daha zordur.

Mikroçekirdek [5] mimarisi ise, modern ve yazılım mühendisliğine daha yatkın bir işletim sistemi mimarisidir. Burada amaç minimal bir çekirdek sağlamaktır. Küçük bir mikroçekirdek ile daha temiz bir arayüz sunulur ve bu sayede daha modüler bir sistem elde edilir. Ayrıca küçük olan çekirdek ile bakım işlemi kolaylaşır ve hatalara karşı duyarlılık artar. Ayrıca mikroçekirdeğin servislerinden faydalanan sunucular ile dağıtıklık ve sistemin taşınabilirliği artacaktır.

Bu çalışma kapsamında, gerçek zamanlı gömülü uygulamaların gereksinimlerini karşılayacak olan eGe Gömülü İşletim Sistemi (eGİS) [6], C++ dili ve açık kaynak koda sahip ücretsiz yazılım geliştirme araçları olan Gcc derleyicisi, Gdb hata ayıklayıcısı, Ld bağlayıcısı ve Bochs emülatörü kullanılarak geliştirilmiştir. eGİS'in tasarımında modern yazılım kavramları ve özellikle tasarım desenlerini taban alan mikroçekirdek mimarisi göz önüne alınmıştır. eGİS bir uygulamaya yönelik işletim sistemi olarak uygulama ihtiyaçlarına göre değiştirilebilir ve biçimlendirilebilir bir yapıya sahiptir. eGİS'in bir başka özelliği ise tamamiyle Türkçe gerçekleştirime sahip olmasıdır.

Bildirinin geri kalan kısmı şu şekilde düzenlenmiştir: İkinci bölümde eGİS yaygın olarak

kullanılan ticari ya da açık kaynak kodlu diğer gömülü işletim sistemleri ile karşılaştırılmış, benzerlikler ve farklılıklar belirtilmiştir. Üçüncü bölümde eGİS'in mimarisi ve tasarım hedefleri açıklanmıştır. Dördüncü bölümde tasarım desenlerinin eGİS işletim sisteminde uygulandığı noktalar belirtilmiş; bunların sisteme kattığı faydalar açıklanmıştır. Beşinci bölümde ise eGİS'in Bochs emülatörü üzerinde çalıştırılması sonucunda elde edilen deneysel veriler kullanılarak tasarım desenlerinin sisteme getirdiği ek yükler incelenmiştir. Altıncı bölümde, çalışma özetlenerek gelecekte yapılabilecek araştırma ve geliştirmeler tartışılmıştır.

2. 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 [7]. 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 sistemi de bellek problemi olmayan çok geniş sistemler için tasarlanmıştır. eGİS ise gömülü bir mikroçekirdektir. Gerçek zaman kısıtlarını göz önüne alan yönetim algoritmalarını içermektedir.

uCOS-II [8] işletim sistemi, gömülü gerçek zamanlı sistemler için gerçekleştirilmiş ticari bir işletim sistemidir. C dili ile gerçeklenen uCOS-II, monolitik bir 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. 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 donanıma daha rahat taşınabilirlik özelliklerini içermektedir.

Nucleus [9] gömülü gerçek zamanlı işletim sistemi de, uCOS-II gibi ticari bir işletim sistemidir. Nucleus işletim sistemi de C dili ile gerçekleştirilmiş olup moduler yapıya sahip bir mimariye sahiptir. Ancak Nucleus da uCOS-II gibi modern yazılım kavramları göz önüne alınarak gerçekleştirilmiş bir işletim sistemi değildir. 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 [10] 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.

PURE [11] işletim sistemi gömülü sistemler için geliştirilmiş ve uygulamaya yönelik bir işletim sistemidir. PURE sınıf hiyerarşisi ve sınıf kalıtımına dayalı bir sistem mimarisine sahiptir. eGİS ile kıyaslandığında, çok geniş ve karmaşık bir sınıf hiyerarşisine dayanır. eGİS sistemi ise tasarım desenlerinin de üzerine oturduğu arayüz kalıtımına dayanan bir işletim sistemidir.

CHORUS [12] mikroçekirdek mimarisine sahip bir işletim sistemidir. C++ dili ile gerçekleştirilmiştir ve bir sınıf kalıtımı ve hiyerarşisine dayanmaktadır. Ayrıca CHORUS bileşenleri çok büyük ve uygulama amaçlarına göre değiştirilebilir değildir. eGİS ise küçük ve bağımsız bileşenlere sahiptir ve arayüz kalıtımına dayanmaktadır. Ayrıca, eGİS'ten farklı olarak, CHORUS tasarım deseni tabanlı bir sistem değildir.

TINYOS [13] 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 bileşenler derleme anında sistem ile birleştirilmesi işlemlerini yerine getirmek için kullanılırlar. TINYOS, kendi sistemine özgü nesC dili

ile, var olan bileşenlerin uygulama gereksinimlerine göre birbirlerine bağlanmasını tasarım desenleri ile sağlar. İşletim sistemini kullanacak olan uygulamanın gereksinimlerine göre seçilen TINYOS işletim sistemi bileşenleri, tasarım desenlerinin derleme anında sunmuş olduğu değişebilirlik özelliği ile sistemle birleştirilmektedir. eGİS'te ise şu anda işletim sistemi yapılandırılması herhangi bir üst seviye dille değil elle yapılmaktadır. Dolayısıyla eGİS'teki tasarım desenleri bileşenlerin birbirleri ile bağlanmasından daha çok sistem içerisinde yer alan alt sistemlerin tasarımları ile ilgilidir. Ayrıca eGİS'te de tıpkı TINYOS'taki gibi desenlerin derleme anında sunduğu avantajlardan faydalanılmıştır.

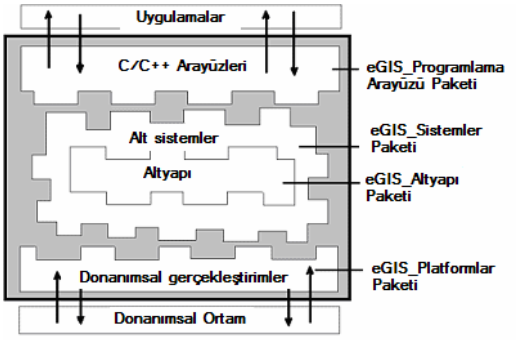
EPOS [14] işletim sisteminin ana amacı işletim sistemini kolaylıkla uygulama programının istediği yapıya kavuşturmaktır. Bu işletim sisteminde de arayüz ve gerçekleştirim ayırımı ile yapılır. EPOS'ta sistem derleme anında kullanıcıdan elde edilen bilgiler doğrultusunda ilgili alt sistemlerle derlenir. Burada kullanıcı bilgisi sayesinde sisteme ek yük getiren soyutlamalardan derleme anında kurtulunur. eGİS'te EPOS'taki gibi bir çalışma henüz yapılmamıştır. Sistem mümkün olduğunca sanal fonksiyon ve dinamik bellek tahsisinden kaçınsa da, henüz desenlerin kullanımı ile oluşan ek yük EPOS'taki gibi giderilememiştir.

3. eGİS Sistem Mimarisi

eGİS gerçek zaman kısıtlarına sahip gömülü sistemler için, genişleyebilir, değiştirilebilir ve taşınabilir bir nesneye yönelik mimariye sahip olacak şekilde tasarlanmış ve tasarım desenleri kullanılarak gerçekleştirilmiş bir mikroçekirdektir. eGİS süreç yönetimi, kesme yönetimi ve süreçler arası haberleşme işlevlerini yerine getiren üç alt sistemden oluşmaktadır.

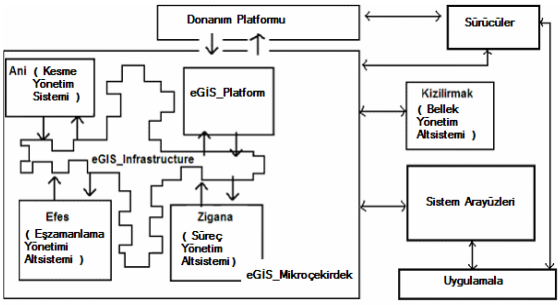
eGİS'in katmanlı mimarisi Şekil 1'de gösterilmiştir. eGIS *C++ sistem arayüzleri*, *alt sistemler*, *altyapı* ve *platform* katmanlarını içermektedir. Sistemdeki temel arayüzler ve soyutlamalar altyapı katmanında tanımlanmakta ve altsistemler katmanında bu soyutlamaların gerçekleştirimleri yer almaktadır. Örneğin temel süreç soyutlamaları altyapı katmanında yer alırken, uygulama gereksinimlerine göre süreç gerçekleştirimi altsistemler katmanında yer almaktadır. Platform katmanı donanıma özgü gerçekleştirimleri içermektedir. Altsistemler ve altyapı katmanları bu sayede hiçbir donanımsal bağımlılık içermezler.

C++ arayüzleri katmanı uygulamalar için bir uygulama geliştirme arayüzü sunmaktadır. Bu sayede uygulamaların katmanlı mimariden ve sistemin iç yapısından haberdar olmaları engellenmiştir. Ayrıca alt katmanlarda meydana gelen herhangi bir değişiklik, uygulama katmanında bir değişikliğe neden olmayacaktır.



Şekil 1. eGİS'in katmanlı mimarisi

eGİS mikroçekirdeğinin içsel yapısı Şekil 2'de gösterilmiştir. Kesme, süreç ve haberleşme sistemleri sırası ile ***Ani***, ***Zigana*** ve ***Efes*** alt sistemlerinde gerçekleştirilmişlerdir. *Zigana* öncelik tabanlı kesilebilir süreç yönetim algoritmasının gerçekleştirildiği alt sistemdir. *Zigana* süreç yaratma, öldürme, durdurma gibi servisleri sunmaktadır. *Ani* basit ve verimli bir kesme sistemi olup, kesme servislerine kayıt olmak ve bu servisleri işleyebilmek için basit bir mekanizma sağlamaktadır. *Efes* ise süreçler arası haberleşme ve eşzamanlamayı sağlayan semafor/kilit sinyalleme ve serbest bırakma mekanizmalarının gerçekleştirimlerini barındırır.



Şekil 2. eGİS mikroçekirdeğinin iç yapısı ve dış bileşenlerle olan etkileşimi

Şekil 2'de görüldüğü gibi, eGİS içerisinde yer alan tüm alt sistemler donanımdan soyutlanmışlardır. 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, *eGIS_AltYapı* içerisindeki donanımsal soyutlamalarının gerçekleştirimlerini içermektedir. Dolayısıyla taşınan ortama ilişkin gerçekleştirimler bu pakete eklenirse, eGİS mikroçekirdeği yeni ortama taşınmış olur. Hiçbir alt sistem bundan etkilenmez. Bu sayede sistem gerçekleştirimi sırasında, tüm alt sistemler kişisel bilgisayar üzerinde birbirlerinden bağımsız olarak test edilebilmişlerdir.

eGİS işletim sisteminin genişleyebilen ve uygulama gereksinimlerine göre değişebilen bir sistem olmasında, arayüz kalıtımı ve nesne içermeye dayalı bir mikroçekirdek mimarisine sahip olmasının rolü büyüktür. İşletim sistemine yeni bir özellik eklemek, mikroçekirdeğe yeni bir alt sistem eklenmesi ile sağlanmış olur. Aynı arayüzleri ancak farklı gerçekleştirimleri içeren alt sistemler, birbirleri ile değiştirilebilirler. Bu sayede, mikroçekirdek tarafından arayüzleri belirlenmiş ama gerçekleştirimleri farklı çok sayıda alt sistem eGİS sistemine uygulama ihtiyaçlarına göre eklenebilir. Yeni bir özelliğin eklenmesi, yeni bir alt sistemin gerçeklestirilmesi ve çekirdeğe kayıt edilmesi ile sağlanır. Örneğin öncelik tabanlı kesilebilir süreç algoritmasının gerçekleştirilmiş olduğu Zigana süreç sistemi, Round-Robin süreç yönetim algoritmasını gerçekleştiren başka bir sistemle değiştirilebilir. Arayüzler ve gerçekleştirimlerin bir işletim sistemi içerisinde ayrıştırılmış olması, eGİS sisteminin esnekliğini güçlendirmiştir.

eGİS işletim sisteminin katmanlı mimarisi özel olarak taşınabilirlik, genişleyebilirlik, bakım ve test edilebilirlik bakımından genel sisteme katkıda bulunmuştur. Ayrıca eGİS'in uygulama ihtiyaçlarına göre şekillenebilmesi sistem gerçekleştiriminde tasarım desenlerinin uygulanması ile kolaylaşmıştır. Desenler, eGİS'in uygulama ihtiyaçlarına göre değişebilecek noktalara uygulanmıştır ve bu noktalardaki değişimler; desenlerin uygulanması sayesinde sistem içerisinde sadece belirli kısımların değiştirilmesini gerektirir.

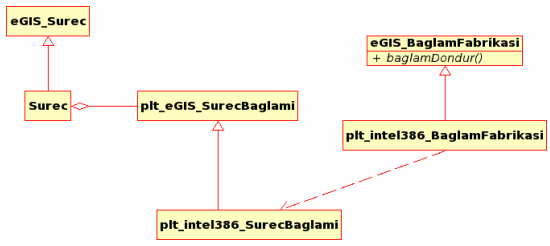
4. Tasarım Desenlerinin eGİS İşletim Sisteminde Uygulanışı

eGİS işletim sisteminde kullanılan tasarım desenleri *yaratımsal*, *yapısal* ve *davranışsal* tasarım desenleri olarak sınıflandırılabilir.

Tekil [3] ve ***Soyut Fabrika*** [3] desenleri, eGİS sistemi içerisinde kullanılan ***yaratımsal*** desenlerdir. Bu desenler sınıfları nesne yaratma işlemlerini soyutlarlar ve sistemin nesnelerin içsel yapıları ve gerçekleştirimlerinden bağımsız hale gelmesini sağlarlar. Sınıfların somut isimlerini açık bir şekilde kullanarak nesneleri yaratmak, sistemin değişebilirliğini ve uygulama gereksinimlerine göre şekillendirilebilir olmasını engellemektedir. Bunun önüne geçmek için, sistem içerisindeki nesne yaratımından dolayı doğacak olan bağımlılıkların ortadan kaldırılması gerekmektedir.

Tekil tasarım deseni ile sınıfların tek bir örneğinin olması garanti altına alınmıştır ve istemciler bu tek örneğin yaratılmasından soyutlanmışlarıdır. eGİS sistemi içerisinde yer alan mikroçekirdek sınıf tekil tasarım deseninin kullanılmasına bir örnektir. Alt sistemler kendilerini mikroçekirdek nesnesine kayıt ederler. Bundan ötürü mikroçekirdek nesnesi kontrollü bir erişimin olması gereken ve istemcilerin bu nesnenin yaratımından bağımsız hale getirilmesini gereken bir sistem elemanıdır.

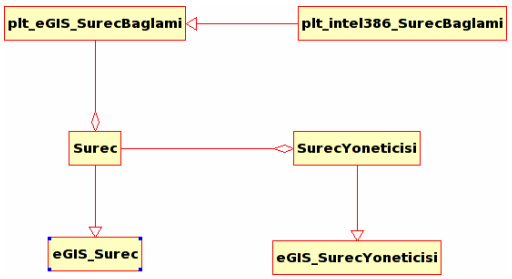
Soyut fabrika deseni, eGİS sistemi içerisinde yer alan nesneleri somut sınıflarını belirtmeden yaratmak için kullanılmıştır. Bu desenin eGİS içerisindeki örnek kullanımı Şekil 3'te gösterilmiştir. Süreç bağlamı ve süreçler arası geçişi sağlayan mekanizmalar, *plt_eGİS_SüreçBağlamı* arayüzünden türeyen sınıflar tarafından gerçekleştirilirler. *eGİS_Süreç* arayüzünden türeyen sınıflar sadece *plt_eGIS_SüreçBağlamı* arayüzünden haberdarlardır. Donanıma özgü alt seviye kodlar platforma özgü sınıflara gömülmüştür. Ancak, donanıma özgü kodlar içeren bu sınıfların sistem içerisinde bir şekilde yaratılmaları ve süreçlere bağlanmaları gerekmektedir. Eğer donanıma bağımlı bu sınıfların yaratımı *eGİS_Süreç* arayüzünden türeyen sınıflar tarafından yapılırsa, sistemde donanımsal kodlar içeren bu sınıflara bir bağımlılık ortaya çıkmaktadır. Sınıflar arasındaki bu bağımlılığı ortadan kaldırmak için platform bağımlı sınıflar *eGIS_BağlamFabrikası* arayüzünü gerçekleştiren sınıflar tarafından yaratılırlar. Sonuç olarak eGİS sistemi donanıma bağımlı olan somut sınıflardan bağımsız hale gelir. eGİS sistemi içerisinde yer alan istemci nesneler somut platform nesnelerinden haberdar olmadan ve sadece onların arayüzlerini kullanarak ilgili isteklerini yerine getirirler.



Şekil 3. Soyut Fabrika desenin eGİS içerisinde kullanımı

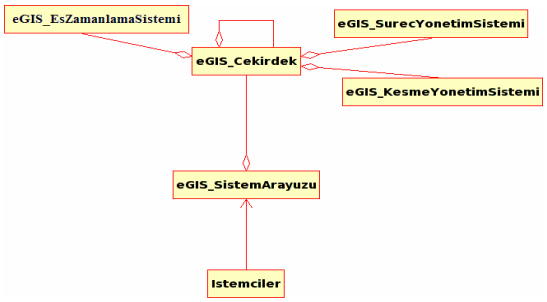
eGİS sistemi içerisinde kullanılan yapısal desenler sınıfların birleştirilerek daha büyük işlevsel sınıfların oluşturulmasını ve nesnelerin içerilmesi ile ek işlevsellik kazanılmasını sağlar. Bu sayede eGİS yeni donanımsal ortamlara daha rahat taşınır ve uygulama gereksinimlerine göre daha kolay genişleyebilir. Ayrıca nesneler arasındaki bağımlılıklar da ortadan kalkmış olur. eGİS işletim sistemi içerisinde kullanılan temel yapısal desenler ***Köprü*** [3] ve ***Önyüz*** [3] desenleridir.

Köprü deseni donanım bağımlı işlemleri soyutlar ve bu işlemlere ait arayüzler ve gerçekleştirimlerin birbirlerinden ayrılmasını sağlar. Şekil 4'te nesne içerme mekanizmasına dayanan bu desenin eGİS sistemi içerisindeki kullanımı örneklenmiştir. *plt_eGİS_SüreçBağlamı* arayüzünü gerçekleştiren sınıflar donanım ortamına bağlı yazmaç bilgilerini tutan ve süreçler arası geçişi sağlayan gerçekleştirimleri içerirler. *eGİS_Süreç* arayüzünü gerçekleştiren sınıflar *plt_eGİS_SüreçBağlamı* arayüzünü gerçekleştiren bu nesneleri içererek isteklerini bu nesnelerin sadece arayüzlerini kullanarak gerçekleştirirler. Bu sayede donanım ortamına olan bağımlılıkları ortadan kalkar. Süreç nesnelerinin içerdiği bağlam nesneleri Intel386 donanımsal ortamına ilişkin bir gerçekleştirime sahip olabileceği gibi Mips mimarisine sahip bir donanımsal ortama ait gerçekleştirimleri de barındırabilir. Sonuç olarak, süreç nesneleri donanımsal bağımlılıklar soyutlanmış ve bu da eGİS'in taşınabilirliğini arttırmıştır. eGİS'i başka bir donanım ortamına taşımak için, donanım ortamına uygun bir şekilde *plt_eGİS_SüreçBağlamı* arayüzünü gerçekleştiren sınıfların yazılması ve bunların süreç nesneleri ile ilişkilendirilmeleri gerekmektedir. Bunun dışında işletim sisteminin diğer taraflarında herhangi bir değişime gerek yoktur.



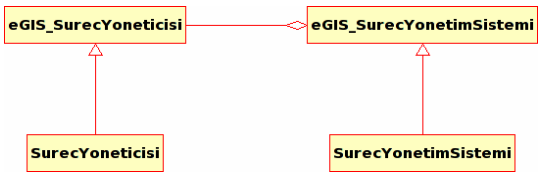
Şekil 4. Köprü desenin eGİS içerisinde kullanımı

eGİS işletim sistemi birçok sınıf ve arayüzleri içerisinde barındırmaktadır. Ancak eGİS'in sunduğu servislerden faydalanmak isteyen istemciler, içsel nesnelerden ve karmaşık sistem yapısından soyutlanmalıdırlar. Sistemin genel, basit ve temiz bir arayüzünün olması gerekmektedir. *Önyüz* tasarım deseni istemcilere üst seviye ve basit bir arayüz sunmaktadır. Şekil 5'te Önyüz deseninin gömülü sistem uygulamalarına bir sistem arayüzünü nasıl sunduğu gösterilmektedir. Üst seviye bir arayüz sunan *eGİS_SistemArayüzü* sınıfı, işletim sisteminin içsel yapısını ve sınıflarını uygulamalardan saklar. Sistem arayüzünü kullanan istemciler sadece *eGIS_SistemArayüzü* 'nden haberdardırlar. Bunlara ek olarak istemciler isteklerini daha az nesne ile haberleşerek gerçekleştirirler.



Şekil 5. Önyüz desenin eGİS içerisinde kullanımı

Davranışsal desenler eGİS sisteminin algoritmik bağımlılıklarını ortadan kaldırmak için kullanılmıştır. ***Strateji*** [3], eGİS içerisinde kullanılan temel davranışsal desendir. Farklı sistem yönetim algoritmaları strateji deseni sayesinde sisteme kolayca eklenebilir. Bu sistem içerisindeki yönetim algoritmalarının uygulama ihtiyaçlarına uygun bir şekilde seçilebilmesini ve sistemin uygulama ihtiyaçlarına göre şekillendirilebilir olmasını sağlar. Algoritma arayüzünü gerçekleştiren uygun bir sınıf, işletim sistemi kodu ile sorunsuz bir şekilde bağlanabilir. Şekil 6 strateji deseninin kullanılması ile eGİS sisteminin kazanmış olduğu esnekliği göstermektedir. eGİS sistemi içerisinde yer alan her süreç yöneticisi, *eGİS_SüreçYöneticisi* arayüzünü gerçekleştirmek zorundadır. Süreç yöneyim servislerini kullanmak isteyen istemciler sadece *eGİS_SüreçYöneticisi* arayüzünü bilmektedirler. Bu sayede algoritmik gerçekleştirimlere olan bağımlılık ortadan kaldırılmıştır. Farklı süreç yönetim algoritmaları yeni gerçekleştirimleri barındıran sınıfların sisteme eklenmesi ile işlevsel hale gelmiş olur. Sistemin diğer kısımlarında herhangi bir değişiklik yapmaya gerek yoktur. Ayrıca algoritmik arayüzler ve gerçekleştirimlerin birbirinden ayrılması yeniden kullanılabilirliği arttırmaktadır.



Şekil 6. Strateji desenin eGİS içerisinde kullanımı

Özetleyecek olursak, eGİS sistemi içerisinde kullanılan desenler işletim sistemi mimarisini daha genişleyebilir ve değiştirilebilir kılmıştır. Bunun sonucunda eGİS, uygulama gereksinimlerine göre şekillendirilebilir ve değişik donanım ortamlarına kolay bir şekilde taşınabilir hale gelmiştir. Donanımdaki ilerlemeler ve gömülü sistemlerde kullanılan işlemcilerin daha da güçlendiği düşünülürse, desenlerin sisteme getirdiği ek yükler giderek daha önemsiz hale gelecektir.

5. Gerçekleştirim ve Deneysel Sonuçlar

eGIS C++ dili ile Gcc [15] C++ derleyicisi, Gdb [16] hata ayıklayıcısı ve Bochs [17] i386 PC emulatörü gibi açık kaynak koda sahip araçlar kullanılarak geliştirilmiştir. Şu anda eGİS sadece i386 ortamına taşınmıştır ve 3 alt sistem ile yaklaşık 60 adet sınıftan oluşmaktadır.

```
plt_intel386_BaglamFabrikasi baglamFabrikasi;
SurecYoneticisi surecYoneticisi;
SurecYonetimSistemi surecYonetimSistemi;

KesmeYonetimSistemi kesmeYonetimSistemi;
KesmeTablosu kesmeTablosu;
plt_intel386_KesmeSistemi plt_KesmeSistemi;

/* baslangic mesajlari */
cout<<"eGIS Isletim Sistemi ver 1.0...\n";
cout<<"Sistem ilkleniyor...\n";

/* surec yonetim alt sistemi olan Zigana kaydediliyor */
surecYoneticisi.platformAta(&baglamFabrikasi);
surecYonetimSistemi.surecYoneticisiAta(&surecYoneticisi);
eGIS_CEKIRDEK->surecYonetimSistemiAta(&surecYonetimSistemi);
cout<<"Surec yonetim sistemi ilklendi...\n";

/* kesme yonetim sistemini ata */
kesmeYonetimSistemi.kesmeTablosuAta(&kesmeTablosu);
kesmeYonetimSistemi.platformAta(&plt_KesmeSistemi);
eGIS_CEKIRDEK->kesmeYonetimSistemiAta(&kesmeYonetimSistemi);
cout<<"Kesme yonetim sistemi ilklendi...\n";

kesmeYonetimSistemi.sistemIlkle();
kesmeYonetimSistemi.kilitle(true);
cout<<"Kesmeler ilklendi...\n";

surecYonetimSistemi.sistemIlkle();
cout<<"Surec yonetim sistemi ilklendi ...\n";

/* bos durum surecini yarat */
bosDurumSureciYarat();
cout<<"Bos surec yaratildi...\n";

/* uygulama surecini yarat */
uygulamaSureciYarat();
cout<<"Uygulama sureci yaratildi ...\n";
```

Şekil 7. eGİS sistem giriş noktası

eGİS'in ana giriş noktası Şekil 7'de gösterilmiştir. Süreç, kesme ve haberleşme yönetimi servislerini sunan alt sistemler sırası ile mikroçekirdeğe kaydedilmekte ve ilgili donanımsal ortam bu sistemler ile ilişkilendirilmektedir. Görüldüğü gibi, desen tabanlı mikroçekirdek sayesinde eGİS'in uygulama isteklerine göre şekillendirilebilmesi kolaydır. Sistem başlangıcı esnasında ilgili alt sistemlerin seçimi ile eGİS yeniden yapılandırılır ve sistemin diğer taraflarında başka bir değişiklik yapılmasına kalmaz. eGİS'i yeni bir donanım ortamına taşımak, ortama özgü gerçekleştirimleri barındıran nesnelerin sisteme kaydedilmesi ile olmaktadır.

```
/*
 * Sureclerin yaratilmasi ve surec yonetim
 * isteklerinin ilgili yoneticilere aktarilmasi
 * islemleri icin temel arayuz sinifi
 */
class e_IsParcacigi : public e_Nesne
{
    public:

        e_IsParcacigi(SurecOnceligi_t oncelik);
        virtual ~e_IsParcacigi();

        /* tum is parcaciklari icin genel giris noktasi */
        static void *GenelGirisNoktasi(void *parametre);

        void basla();
        uint8_t onceligiDegistir(SurecOnceligi_t oncelik);

        /* tum is parcaciklari tarafindan gerceklesstirilecek temel metod */
        virtual void calismaNoktasi() = 0;

    protected:

        SurecKimligi_t kimlik;
        eGIS_SurecOzellikleri surecOzellikleri;
};
```

Şekil 8. eGİS uygulama iş parçacığı arayüzü

Şekil 8 gömülü uygulamalar için genel bir iş parçacığı arayüzünü göstermektedir. İş parçacıkları eGİS sisteminde *eGİS_Süreç* ile temsil edilirler. Ancak uygulamalar için daha genel, detayların saklandığı bir arayüz gerekmektedir. *e_IsParcacigi* işletim sistemi içsel detaylarını uygulamadan saklar. Bu tip arayüzler sayesinde uygulamalar işletim sistemi içerisinde ortaya çıkacak değişimlerden etkilenmezler. Ayrıca, hangi alt sistemin kayıt edilmiş olunduğu ve hangi donanımsal ortamda çalışıldığı bilinmeden, uygulamalar isteklerini karşılayabilirler.

```
/**
 * Ornek uygulama
 */
class Uygulama_1 : public e_IsParcacigi
{
    public:

        Uygulama_1();
        virtual ~Uygulama_1();

        virtual void calismaNoktasi();
};
```

Şekil 9. eGİS uygulama arayüzünden türeyen bir uygulama sınıfı

Şekil 9, e_IsParcacigi arayüzünü gerçekleştiren bir uygulama sınıfını göstermektedir. Her işparçacığı, saf sanal fonksiyon olan ***calismaNoktasi*** metodunu gerçeklemelidir. Görüldüğü gibi uygulama sınıfında da, bu metod gerçekleştirilmiştir. Bu metod içerisinde uygulamanın işleyeceği kodlar tanımlanmıştır.

```
#include <eGIS_programlamaarayuzu.h>
#include "uygulama 1.h"


using namespace eGIS;

e_Kilit kilit_1;
e_Kilit kilit_2;

/**
 * uygulamalarin baslayacagi nokta
 */
void *uygulamaBaslangici(void *arg)
{
    Uygulama_1 uygulama_1;

    cout<<"UYGULAMA BASLADI...\n";

    kilit_1.kilitle(0);
    cout<<"Kilit 1 kilitlendi...\n";

    kilit_2.kilitle(0);
    cout<<"Kilit 2 kilitlendi...\n";

    uygulama_1.basla();

    while(1)
    {
        cout<<"Ana Uygulama ...\n";
    };
}
```

Şekil 10. Uygulamaların başlatılması

Tanımlanan uygulamalar, uygulamalar için bir başlangıç noktası olan ***uygulamaBaslangici*** fonksiyonu içerisinde başlatılırlar. İşletim sistemi zaten çalışmaya başlamıştır ve yönetim sistemleri bu noktada aktiftir. Şekil 10'da gösterildiği gibi uygulamaların başlatılması ile, işletim sistemi hangi iş parçacığı en öncelikli ise işlemciyi ona verecektir. Bu iş parçacığı işlemciyi kendisi bırakana dek ya da daha öncelikli bir iş parçacığı yaratılana dek işletim sistemi bu iş parçacığını çalıştıracaktır.

Desen	*İşlem*	*Desen katmanında harcanan saat darbesi*	*Geçen toplam saat darbesi*
Tekil	eGIS_Mikroçekirdek nesnesine erişim	12	12
Köprü	Bir sonraki adımda çalışacak iş parçacığının belirlenmesi ve çalıştırılması	29	168
Önyüz	İş parçacığı yaratımı	< 234	1146

Tablo 1. eGİS'in i386 mimarisinde çalıştırılması sonucunda elde edilen deneysel sonuçlar

Tasarım desenlerinin uygulanması sistemde oluşan ek yük Tablo 1'de gösterilmiştir. Bu sonuçlar Bochs emülatörü üzerinde alınmıştır. Örneğin *eGİS_Mikroçekirdek* örneğine erişmek 12 saat darbesi sürmektedir. Sistemin çok duyarlı zaman kısıtına sahip gerçek zaman gereksinimlerine ihtiyaç duyan noktaları dışında, bu desen güvenli bir şekilde kullanılabilir. Süreç yönetimi köprü deseninin kullanıldığı bir diğer sistem noktasıdır. Bir sonraki adımda hangi sürecin çalışacağına karar verilmesi yaklaşık 168 saat darbesi sürmektedir. Süreçler arası geçiş işleminde köprü katmanından geçiş 29 saat darbesi sürmektedir. Eğer gerçekleştirimlerin ve arayüzlerin ayrılması ile ortaya çıkan avantajları ve özellikle de donanımsal ortamlara taşınabilmenin kolaylaşması göz önüne alınırsa, köprü deseninin getirmiş olduğu 29 saat darbelik ek yük kabul edilebilir boyutlardadır. Önyüz katmanı üzerinden bir iş parçacığı yaratımı isteğinin ilgili alt sisteme iletilerek işlenmesi yaklaşık 1146 saat darbesi tutmaktadır. İstek önce önyüz katmanında işlenmekte ve buradan süreç yönetim sistemine iletilmektedir. Ön yüz katmanında geçen süre 234 saat darbesidir. Burada önyüz katmanında yapılan işlemlerin hepsinin bu desene özgü işlemler olmadığı belirtilmelidir. Örneğin iş parçacığı yaratım parametrelerinin eGİS mikroçekirdeğine geçirilmesi bu desen olmasa da yapılmak zorundadır. Dolayısıyla, 234 saat darbelik sure, önyüz katmanı için harcanan sürenin üst sınırını oluşturmaktadır.

Bize göre Tablo 1 'deki başarım sonuçları, desenlerin getirdiği ek yüklerin özellikle işlemcilerin daha güçlendiği gömülü sistemlerde kabul edilebilir boyutlarda olduğunu göstermektedir.

6. Sonuç

Bu çalışmada, yeni bir gerçek zamanlı ve gömülü işletim sistemi çekirdeği olan eGİS, tasarım desenleri göz önüne alınarak gerçekleştirilmiştir. eGİS'in

uygulama gereksinimlerine göre değişebilir ve şekillendirilebilir bir sistem olacak şekilde tasarlanmasına dikkat edilmiştir.

Tasarım desenlerinin eGİS sistemi içerisinde uygulanmasının çözmüş olduğu problemler aşağıdaki gibi özetlenebilir:

- Nesneleri somut sınıf isimlerini açık bir şekilde belirterek yaratma *Tekil* ve *Soyut* Fabrika desenleri ile ortadan kaldırılmıştır.
- *Köprü* yapısal deseni nesneler arasındaki bağımlılıkları ve ortam bağımlılıklarını ortadan kaldırmıştır.
- *Önyüz* deseni eGİS işletim sisteminin içsel yapısını istemcilerden soyutlar ve temiz bir uygulama arayüzü sunar.
- *Strateji* deseni yönetim algoritmalarının birbirleri ile değiştirilebilmesini ve eGİS sisteminin yönetim algoritmalarından bağımsız olması sağlanmıştır.

Gelecekte yapılacak çalışmalarda eGİS'e yeni alt sistemlerin eklenmesini planlamaktayız. Bellek ve disk üzerinden çalışabilen bir dosya sistemi, TCP/IP yığıtı, gömülü kullanıcı arayüzü sistemi gibi alt sistemlerin eklenmesi eGİS'i daha da işlevsel hale getirecektir. Yeni yönetim algoritmalarının sisteme eklenmesi, eGİS'i daha yüksek oranda uygulama gereksinimlerini karşılayabilecek hale getirecektir. Ayrıca eGİS mimarisinin daha da iyileştirilmesi ve eniyilenmesi diğer hedeflerden biridir. Ek olarak eGİS'in farklı donanım ortamlarına da taşınması gerekmektedir.

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. Bu amaçla eGİS açık kaynak kodlu projelerin yer aldığı www.sourceforge.net sitesinde paylaşıma açılmıştır [18].

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 umulmaktadır.

7. Referanslar

[1] E.A. Lee, "What's Ahead for Embedded Software? ", *IEEE Computer Society Press, Computer, v.33 n.9 pp.18-26,* 2000.

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

[3] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series, 1994.

[4] B. P. Douglass, *Realt-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*, Addison-Wesley, 2002.

[5] J. Liedtke, "On micro-kernel construction", *In Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, 1995.

[6] K. S. Yıldırım, "Gömülü Sistemler İçin Tasarım Desenleri Kullanarak Nesneye Yönelik, Gerçek Zamanlı Bir Mikroçekirdek Tasarımı", *Yüksek Lisans Tezi*, *Ege Üniversitesi Bilgisayar Mühendisliği Bölümü*, 2005.

[7] A. S. Tanenbaum, *Modern Operating Systems*, Prentice Hall, 2001.

[8] J. J. Labrosse, *MicroC/OS-II: The Real-Time Kernel, 2nd Ed.*, CMP Books, 2003.

[9] The Nucleus Plus RealTime Operating System anasayfası: www.acceleratedtechnology.com/embedded/nuc_rtos.html, 2001.

[10] The eCOS Operating System anasayfası: http://ecos.sourceware.org/

[11] U. Haack, W. Schröder-Preikschat, F. Schön, O. Spinczyk, "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 Systems,* 1998.

[12] K. Levchenko, "A Survey of Microkernel Operating Systems", http://www.cs.ucsd.edu/classes/fa01/cse221/projects/index.html

[13] D. Gay, P. Levis, D. Culler, "Software Design Patterns for TinyOS", *In Proceedings of the LCTES05*, 2005.

[14] A.A. Fröhlich, W. Schröder-Preikschat, "EPOS: an Object-Oriented Operating System", *In Proceedings of the Workshop on Object-Oriented Technology, Lecture Notes In Computer Science Volume 1743*, 1999.

[15] The GNU Compiler Collection anasayfası: http://gcc.gnu.org/

[16] The GNU Project Debugger anasayfası: http://www.gnu.org/software/gdb/

[17] Bochs IA32 PC Emulator homepage anasayfası: http://bochs.sourceforge.net/

[18] eGİS İşletim Sistemi kod ambarı adresi: http://sourceforge.net/projects/egis-eeos/