

46851

T.C.
EGE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI
10.2700.0000.011

**GENİŞ ÖLÇEKLİ TEKRAR KULLANIM ORTAMLARINDA
BİLEŞEN YÖNELİKLİ YAZILIM GELİŞTİRME**

YÜKSEK LİSANS TEZİ

HAZIRLAYAN
Rıza Cenk ERDUR

DANIŞMAN
Doç.Dr. Bahar Alakent

Bu tez
Prof. Dr. Halil Şengonca
Doç. Dr. Bahar Alakent
Yrd. Doç. Dr. Alp Kut
'dan oluşan jüri tarafından oybirliği ile kabul edilmiştir.

BORNOVA-İZMİR
1995

ÖZET

GENİŞ ÖLÇEKLİ TEKRAR KULLANIM ORTAMLARINDA BİLEŞEN YÖNELİKLİ YAZILIM GELİŞTİRME

Erdur, Rıza Cenk

Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Bahar Alakent

Temmuz 1995

Birleştirmeye dayanan bir tekrar kullanım yaklaşımı olan 'Bileşen Yönelikli Yazılım Geliştirme' (BYYG), nitelikli ve belli standartlara uyumlu yazılım bileşenlerinin birbiriyle veya geliştirilmekte olan yazılımlar ile birleştirilmesi yolu ile yazılım geliştirme yöntemidir. BYYG ortamlarında bileşenlerin saklanması ve bileşenlere erişim için etkin araçlar tasarlanması üzerinde en çok çalışılan konulardandır. Bu çalışmada diğer yöntemlerden farklı olarak, BYYG ortamındaki bileşenlerin çoklu-nitelikli kütük yapılarında saklanması ve bu yapılar üzerinde çalışan bir bileşen geri elde etme aracı kullanılması önerilmektedir. Bu amaçla, çok kullanılan ve bilinen bir çoklu-nitelikli kütük yapısı olan grid kütük yapısı kullanılmaktadır. Yapılan çalışmalar sonunda, çoklu-nitelikli kütük yapılarına dayanan bileşen geri elde etme araçlarının tek-nitelikli kütük yapılarına dayananlardan daha yüksek verime sahip olduğu görülmüştür.

Bilgisayar ağları teknolojisindeki gelişmelerin sonucu olarak bilgisayar ağı üzerinde farklı yazılım ve donanım alt yapılarında olan kullanıcıların, ağ üzerindeki farklı bileşen kitaplıklarına erişerek bileşen yönelikli yazılım geliştirmeleri olası olmaktadır. Bu tür yazılım geliştirme biçimini destekleyen ortamlar geniş ölçekli tekrar kullanım (GTK) ortamları olarak adlandırılmaktadır. Bu çalışmanın diğer bir amacı GTK ortamları için bir yazılım mimarisi önererek, bu konuda görünen bir eksikliği kapatmaktır. GTK ortamları, çok sayıda kullanıcı ve bileşenden oluştuğu için, yüksek verimli bileşen geri elde etme araçları kullanımı özellikle önem kazanmaktadır. Bu nedenle önerilen mimaride grid kütük yapısı kullanan bir bileşen geri elde etme aracı bulunmaktadır. Ortam bir dağıtık nesne yönetim standardı olan CORBA ile modellenmiştir. Mimaride bulunan bileşen geri elde etme aracı ise DEC-ACA yazılımı ile gerçekleştirilmiştir.

Anahtar Kelimeler: Tekrar Kullanım, Bileşen Yönelikli Yazılım Geliştirme, Çoklu-Nitelikli Kütükler, Grid Kütük Yapısı, Geniş Ölçekli Tekrar Kullanım, CORBA.

ABSTRACT
**COMPONENT ORIENTED SOFTWARE DEVELOPMENT IN LARGE SCALE
REUSE ENVIRONMENTS**

Erdur, Rıza Cenk
M.S. in Computer Engineering
Supervisor: Assoc. Prof. Dr. Bahar Alakent
July, 1995

'Component Oriented Software Development' (COSD) which is a compositional reuse approach, is a style of software development. COSD means developing software by integrating high quality software components designed for reuse. Storing components and designing efficient retrieval mechanisms are among the most important issues in a COSD environment. In this study as an alternative to other methods, it is proposed that the components in a COSD environment should be stored in multi-attribute file structures. It is also proposed that the component retrieval mechanisms should be based on multi-attribute file structures. For this purpose grid file -a well known multi-attribute file structure- is used. It has been shown that multi-attribute file structures have better performance than single-attribute file structures when used in component retrieval mechanisms.

As a result of improvements in computer network technology, it is possible for users on different hardware and software platforms on the network to develop software by integrating the components chosen from different libraries. The environment which support this kind of software development is called large scale reuse (LSR) environment. Second aim of this study is to propose a software architecture for LSR environments. For this purpose, a high performance software architecture which uses the grid file structure in component retrieval tool and which depend on CORBA is proposed. The retrieval tool in the proposed architecture is implemented by DEC-ACA which is a product of CORBA standart.

Keywords: Software Reuse, Component Oriented Software Development, Multi-Attribute File Structures, Grid File Structure Large Scale Reuse, CORBA.

TEŞEKKÜR

Tez çalışması süresince değerli yardımlarını esirgemeyerek her konuda destek veren danışmanım Sayın Doç.Dr. Bahar Alakent'e teşekkür ederim.

Tez çalışması süresince gösterdikleri yakın ilgi ve sağladıkları olanaklar için, Bilgisayar Mühendisliği bölüm başkanı Sayın Prof.Dr. Şaban Eren ve bölüm başkan yardımcısı Sayın Doç.Dr. Mustafa Türksever'e teşekkür ederim.

Tezin düzeltme aşamasındaki önerileri için Sayın Prof.Dr. Halil Şengonca'ya teşekkür ederim.

Değerli tartışmaları ve faydalı önerileri için Sayın Oğuz Dikenelli'ye, gerçekleştirim aşamasındaki katkıları için lisans öğrencileri Damla Sezgin, Görkem İplikçioğlu, İsmail Erhan Kurt ve Rıza Dindir'e teşekkür ederim.

En son fakat en az olmayarak, bu çalışma süresince sağladıkları sürekli destek için aileme teşekkür ederim.

İÇİNDEKİLER

	<u>Sayfa</u>
ÖZET	iii
ABSTRACT	iv
TEŞEKKÜR	v
KISALTMALAR DİZİNİ	x
1.GİRİŞ	1
2.YAZILIM MÜHENDİSLİĞİNDE TEKRAR KULLANIM	3
2.1.Tekrar Kullanım Kavramı	3
2.2.Yazılım Mühendisliğinde Tekrar Kullanım Tarihçesi	4
2.3.Yazılım Mühendisliğinde Tekrar Kullanım Bakış Açıları	5
2.3.1.Tekrar kullanılabilir birimlerin cinsine göre tekrar kullanım	6
2.3.2.Uygulama alanının özelliklerine göre tekrar kullanım	7
2.3.3.Gerçekleştirme şekline göre tekrar kullanım	7
2.3.4.Tekrar kullanım teknikleri	8
2.3.5.Birimlerin tekrar kullanım şekilleri	8
2.4.Tekrar Kullanımda Krueger Sınıflaması	9
2.4.1.Yüksek düzeyli dillerde tekrar kullanım	9

İÇİNDEKİLER(devam)**Sayfa**

2.4.2.Tasarım ve kaynak kod parçaları alıntısı	10
2.4.3.Kaynak kod kitaplığı	10
2.4.4.Yazılım şemaları	11
2.4.5.Uygulama üreticileri	11
2.4.6.Çok yüksek düzeyli diller	11
2.4.7.Dönüşümsel sistemler	12
2.4.8.Yazılım mimarileri	12
3.BİLEŞEN YÖNELİKLİ YAZILIM GELİŞTİRME	14
3.1.Tekrar Kullanılabilir Bileşenlerin Üretimi	18
3.1.1.Tekrar kullanılabilir bileşen üretimi için bir yöntem	19
3.1.2.Nesneye dayalı yaklaşımın tekrar kullanımı destekleyici özellikleri	21
3.2.Sınıflandırma Tasarımları ve Bileşen Geri Elde Etme Araçları	22
3.2.1.Sınıflandırma tasarımları	22
3.2.2.Bileşen geri elde etme araçları	28
4.BİLEŞEN GERİ ELDE ETME ARAÇLARINDA ÇOKLU-NİTELİKLİ KÜTÜK YAPILARININ KULLANIMI	32
4.1.Çoklu-nitelikli Kütükler ve Grid Kütük Yapısı	32
4.1.1.Grid kütük yapısında kayıtlara erişim	35
4.1.2.Grid kütük yapısı oluşturmak için bir algoritma	37
4.2.Grid Kütük Yapısı ve Boyutlara Ayırmaya Dayanan Sınıflandırma İlişkisi	43

İÇİNDEKİLER(devam)	<u>Sayfa</u>
4.3.Grid Kütük Yapısının Verimi	44
4.4.Bileşen Tanımlayıcılarının Grid Kütük Yapısında Saklandığı Bileşen Geri Elde Etme Aracı Gerçekleştirimi	46
5.GENİŞ ÖLÇEKLİ TEKRAR KULLANIM	52
5.1.GTK Ortamlarının CORBA ile Modellenmesi	54
5.2.Birlikler Topluluğu Olarak GTK Ortamı	61
5.3.DEC-ACA ile GTK Ortamında Bileşen Geri Elde Etme Aracı Gerçekleştirimi	64
5.3.1. DEC-ACA	64
5.3.2.Bileşen geri elde etme aracının gerçekleştirim adımları	66
6.SONUÇLAR	71
7.TARTIŞMA	77
KAYNAKLAR	78
EKLER	82
Ek-1 İki Boyutlu Grid Kütük Yapısı Oluşturmak İçin C Dilinde Yazılan Program	83
Ek-2 Üç Boyutlu Grid Kütük Yapısı Oluşturmak İçin C Dilinde Yazılan Program	97

İÇİNDEKİLER(devam)**Sayfa**

Ek-3 Beş Boyutlu Grid Kütük Yapısı Kullanan Bileşen Geri Elde Etme Aracı Gerçekleştirimi İçin C Dilinde Yazılan Program	112
Ek-4 Bileşen Geri Elde Etme Aracının DEC-ACA Nesne Sunucu(Object-broker) ile İstemci / Sunucu Biçiminde Modellenmesi İçin Yazılan Programlar	137



KISALTMALAR DİZİNİ

Kısaltmalar

Açıklama

BYYG	Bileşen Yönelikli Yazılım Geliştirme
CORBA	Common Object Request Broker Architecture
DEC-ACA	Digital Equipment Corporation-Application Control Architecture
DNY	Dağıtık Nesne Yöneticisi
DNYS	Dağıtık Nesne Yönetim Sistemi
ESPRIT	European Strategic Programme for Research in Information Technology (Bilgi Teknolojileri Araştırmaları için Avrupa Stratejik Programı)
GTK	Geniş Ölçekli Tekrar Kullanım
OMG	Object Management Group (Nesne Yönetim Grubu)
ORB	Object Request Broker (Nesne İstek Sunucu)
RPC	Remote Procedure Call (Uzaktan Yordam Çağırma)

BÖLÜM-1 GİRİŞ

Tekrar kullanım, gerek günlük hayatta gerekse bilimsel alanlarda sık kullanılan genel bir kavramdır. Aynı tür tuğlaların değişik binaların yapılışında kullanımı, mimarların aynı tasarım yöntemlerini değişik projelerde uygulaması, fizikçilerin aynı fizik kurallarını farklı problemlerin çözümünde kullanması, bilgisayar mühendislerinin tümleşik devrelerle mikro-işleyicili sistemler tasarlaması tekrar kullanım örnekleridir. Genel anlamda tekrar kullanım, aynı bilgilerin veya nesnelerin değişik uygulamalarda kullanımı olarak tanımlanmaktadır (Li, 1992).

Yazılım mühendisliğinde tekrar kullanım, tekrar kullanılabilir yapılar ve bilgilerin değişik yazılım geliştirme süreçlerinde kullanımı olarak tanımlanmaktadır. Tekrar kullanılabilir yapı ve bilgiler, analizleri, tasarımları, kaynak kodları, belgelemeleri, sınama senaryolarını, geçmiş deneyimleri kapsamaktadır.

Yazılım geliştirmenin maliyeti ve süresi üzerinde yapılan istatistikler, yeni baştan yapılan tasarımları ve gerçekleştirmeleri (kodlamalar) maliyet ve süreyi artıran en önemli unsurlar olarak göstermektedir (Griss, 1993). Bu durumda getirilebilecek en iyi çözümlerden birisi tekrar kullanıma dayanan yazılım geliştirme olmaktadır. Tekrar kullanım, yazılım geliştirmenin maliyetini düşürmekte, süreci hızlandırmakta ve yazılımların niteliğini artırmaktadır (Griss, 1993; Li, 1992). Maliyeti düşürüp, süreci hızlandırmasının nedeni, yazılan kod miktarının azalmasıdır. Niteliğin artmasının nedeni, tekrar kullanılabilir yapı veya bilgilerin daha önceden sınanmış ve belli bir standarda uyumlu olmalarıdır. Bunlara ek olarak, tekrar kullanım ile modüler yazılımlar geliştirilmesi olası olduğu için, yazılım geliştirme sürecinin bakım aşaması kolaylaşmaktadır.

Birleştirmeye dayanan bir tekrar kullanım yaklaşımı olan 'Bileşen Yönelikli Yazılım Geliştirme' (BYYG) yaklaşımının uygulanabilmesi için bir BYYG ortamı tasarlanmalıdır. BYYG ortamlarında, bileşenlerin saklanması, taranması, bileşenlere erişilmesi, bileşenlerin birleştirilmesi amaçlarına yönelik araçlar bulunmaktadır. Bileşenlere erişimi sağlayan geri elde etme araçları, BYYG ortamındaki en önemli unsurlardan olmakla beraber, yüksek verimli bileşen geri elde etme araçları tasarımı konusunda eksiklikler görülmektedir. Bu çalışmanın amaçlarından birisi, yüksek verimli bileşen geri elde etme aracı tasarlanmasıdır. Bu tür bir araç, aşağıda sözü edilen geniş ölçekli tekrar kullanım ortamlarında özellikle önem kazanmaktadır. Çoklu-nitelikli kütük

yapılarının kullanılmasının bileşen geri elde etme araçlarının verimini artıracakı düşünöldüğü için bu çalışma kapsamında, çoklu-nitelikli kütük yapılarına dayanan bir bileşen geri elde etme aracı tasarlanmaktadır.

Bilgisayar ağlarındaki gelişmeler, BYYG yaklaşımına değışik boyutlar getirerek 'Geniş Ölçekli Tekrar Kullanım' (GTK) kavramının ortaya çıkmasına neden olmuştur. BYYG yaklaşımı, önceleri sadece belli bir kuruluş içinde geliştirilen bileşenlerin yine o kuruluş içinde tekrar kullanımına dayanan bir yazılım geliştirme biçimi iken günümüzde bilgisayar ağı bağlantılarının artmasıyla ulusal ve uluslararası düzeyde (geniş ölçekte) yazılım paylaşımını destekleyen bir yazılım geliştirme yöntemine temel olmaktadır. GTK ortamlarının tasarımına yönelik bir yazılım mimarisi önermek ve bu mimariyi kullanarak GTK ortamlarında yazılım geliştirme yöntemlerinin belirlenmesi bu çalışmanın diğeri bir amacıdır. GTK ortamları, çok sayıda tekrar kullanılabilir bileşen kitaplıkları ve binlerce tekrar kullanıcıdan oluşmaktadır. Bu nedenlerle, bu ortamlarda bileşen geri elde etme işlemini hızlandırmak gerekmektedir. Çoklu-nitelikli kütük yapıları kullanan geri elde etme araçlarının GTK ortamlarında da kullanılmasının GTK ortamının verimine katkıda bulunacakı düşünölmektedir.

Yukarıda sözü edilen konular, ileride gelen 4 bölümde yer almaktadır. Tekrar kullanım kavramı, tekrar kullanım alanında bugüne dek yapılan çalışmalar ve değışik tekrar kullanım yaklaşımları bölüm 2'de ayrıntılı olarak incelenmektedir. Bölüm 3, BYYG yaklaşımını tanıtmakta ve yerel bir BYYG ortamı üzerinde durmaktadır. Çoklu-nitelikli bir kütük yapısı olan 'Grid' (Izgara) kütük yapısı kullanan bir bileşen geri elde etme aracının tasarımı bölüm 4'de verilmektedir. Bölüm 5'de, çoklu-nitelikli kütük yapısı kullanan bileşen geri elde etme araçlarını içeren ve bir dağıtık nesne yönetim sistemi standardı olan CORBA (Common Object Request Broker Architecture) standardını temel alan bir yazılım mimarisi ortaya konmaktadır. Bu bölümde, ortaya konulan mimaride bulunan bileşen geri elde etme aracının CORBA standardına uygun DEC-ACA yazılım aracı ile gerçekleştirimi de anlatılmaktadır. Bölüm 6, çalışmadan elde edilen sonuçlar üzerinde durmakta, bölüm 7 ise bu çalışmanın nasıl geliştirilebileceğini anlatmaktadır.

BÖLÜM-2 YAZILIM MÜHENDİSLİĞİNDE TEKRAR KULLANIM

2.1 Tekrar Kullanım Kavramı

Tekrar kullanım kavramı, insanların problem çözmeye başlamasından beri varolan bir kavramdır. Bir problemin çözümünde, daha önceden çözülen benzer problemlerde kullanılan yöntemlerin uygulanması etkin problem çözme teknikleri arasındadır. Aynı tür problemlerin çözümünde defalarca kullanılan yöntemler tutulmakta ve standartlaşmaktadır. Genel matematiksel modeller tekrar kullanım için çok uygundur çünkü bu modeller birçok değişik mühendislik alanına ilişkin belirli problemlerin çözümünde kullanılabilirler. Bu modeller tekrar kullanılabilir soyutlamalar olarak nitelendirilmiştir (Prieto-Diaz, 1993). Genel matematiksel modellere dayanan tekrar kullanım yanında, standart yapılara ve elemanlara dayalı tekrar kullanım da çeşitli mühendislik alanlarında görülmektedir. Standart ölçülerdeki vidalar makina mühendisliği, yapı elemanları inşaat mühendisliği, tümleşik devreler ise elektrik mühendisliğinde tekrar kullanılan birimlere örnek olarak verilebilmektedir.

Yazılım mühendisliği alanında tekrar kullanım, geçmiş yazılım geliştirme süreçlerinin ürünlerinin yeni bir yazılım geliştirme sürecinde kullanılması olarak tanımlanmaktadır. Tekrar kullanılabilir yazılım ürünleri arasında analizler, tasarımlar, genel algoritmalar, kaynak kodlar, sinama senaryoları, belgeler, geçmiş deneyimler sayılabilmektedir. Genel algoritmalar, analizler, tasarımlar, sinama senaryoları, belgeler ve geçmiş deneyimler tekrar kullanılabilir soyutlamalara karşılık gelmektedir. Kaynak kodlar (fonksiyonlar, alt-yordamlar, makrolar, nesneler) ise tekrar kullanılabilir yapılardır. Tekrar kullanılabilir yapılar arasında bulunan nesneler geleceğin en önemli tekrar kullanım birimleri olarak görülmektedir. Bunun nedenleri arasında, nesneye dayalı yöntemlerin yazılım tekrar kullanımını destekleyici özellikleri olması (Coleman et al., 1994; Jacobson et al., 1993) ve nesneye dayalı yöntemlerin giderek yaygınlaşması gösterilebilmektedir. Nesneye dayalı yöntemlerin tekrar kullanımı destekleyici özellikleri ilerideki bölümlerde anlatılmaktadır.

Gerek tekrar kullanılabilir soyutlamaların, gerekse tekrar kullanılabilir yapıların tekrar kullanım amacıyla tasarlanmış, nitelikli, güvenilir, sınanmış, kolay anlaşılabilir nitelikte olmaları gerekmektedir. Bu tür soyutlama ve yapıların tekrar kullanımına dayanan bir yazılım geliştirme

süreci daha az işgücü ve zamana gereksinim duymakta ve daha nitelikli yazılımlar üretilmesini sağlamaktadır.

2.2 Yazılım Mühendisliğinde Tekrar Kullanım Tarihçesi

Tekrar kullanımın yazılım mühendisliği alanında uygulanması ile ilgili ilk düşünceler, 1968 yılında McIlroy tarafından ortaya atılmıştır (Prieto-Diaz and Freeman, 1987; Krueger, 1992). McIlroy, sayısal hesaplamalar, giriş/çıkış işlemleri, metinsel işlemler ve dinamik bellek yönetimi konularında tekrar kullanılabilir fonksiyonlar hazırlanmasını, bu fonksiyonların kaynak kodlarının bir kitaplıkta, tanımlarının ise kataloglarda saklanmasını önermiştir. Böylece, kataloglardan seçilen fonksiyonların kitaplıktaki kaynak kodlarının yeni bir yazılım geliştirme sürecinde kullanılabilmesi olanağı doğmuştur.

McIlroy'dan sonra, 1974 yılında, 'yazılım fabrikası' fikri ortaya atılmıştır (Griss, 1993). Sadece tekrar kullanılabilir fonksiyonlar (kaynak kod) içeren bir kitaplıktaki fonksiyonların yeni yazılım geliştirme süreçlerinde kullanımı için tekrar kullanıcının analiz ve tasarım çalışmalarını önceden tamamlaması gerekmektedir. Bunun nedeni tasarımın gereksinim duyduğu fonksiyonların - eğer kitaplıkta varsa - seçilecek olmasıdır. Analiz ve tasarımların , varolan tekrar kullanılabilir yapılar gözönüne alınarak ve tekrar kullanılabilir soyutlamalardan (analiz, tasarım, belgelemeler) da yararlanarak gerçekleştirilmesinin daha iyi bir yöntem olduğu görülmektedir. Yazılım fabrikası fikri, bu yöntemi desteklemek üzere ortaya atılmıştır. Bu yöntem ile, yazılım geliştirme süreci ve biçimi değişmektedir. Bu tür bir biçimi destekleyici kurumlara yazılım fabrikası (Griss, 1993) denmektedir. İlk olarak, 'Systems Development Corporation' isimli kuruluş tarafından kullanılan bu fikir daha sonraları özellikle Japonya'da birçok kuruluş tarafından temel alınarak belli bir sistematik çerçevesinde geliştirilmiştir. Yazılım fabrikası fikri, yazılım geliştiricilerin sadece kaynak kod değil, aynı zamanda her türlü eski ürünü (analiz, tasarım, belgeleme vb.) belli kurallar çerçevesinde kullanılmasını desteklediği için, McIlroy'un kaynak kod kitaplığı fikrinin geliştirilmiş bir şekli olarak nitelendirilebilmektedir.

Yazılım mühendisliğinde tekrar kullanım ile ilgili bilimsel çalışmalar 1970'li yılların sonlarında başlamıştır. Araştırma projeleri ile ilgili sonuç raporları 1983 yılında toplanan

'Programlamada Tekrar Kullanım' isimli ilk uluslararası tekrar kullanım kongresinde tartışılmıştır (Biggerstaff and Perlis, 1984; Prieto-Diaz, 1993).

. Yazılım mühendisliğinde tekrar kullanım, günümüzde, ülke politikaları çapında da destek görmektedir. Örneğin, ' Bilgi Teknolojileri Araştırmaları için Avrupa Stratejik Programı (European Strategic Programme for Research in Information Technology-ESPRIT) ' isimli proje bu çerçevede yürütülmektedir. ESPRIT projesinin alt kolları olarak yürütülen Ithaca (Fugini et al., 1992) ve Reboot (Morel and Faget, 1993) projeleri, tekrar kullanıma dayanan yazılım geliştirme ortamlarının tasarımı üzerinde çalışmaktadır. Yine bu projede ' Avrupa Yazılım Fabrikası (Euroka Software Factory) ' fikri üzerinde de durulmaktadır (Prieto-Diaz, 1993).

Yazılım mühendisliğinde tekrar kullanım alanındaki çalışmalar 1980'li yılların sonlarına doğru yoğunlaşmıştır. Tekrar kullanılabilir yapılar ve soyutlamaların saklanabileceği kitaplıkların tasarımı, bu kitaplıklardaki yapıların ve soyutlamaların sınıflandırılabilmesi için yeni teknikler geliştirilmesi (Prieto-Diaz, 1991; Sorumgard et al., 1993), kitaplıklara etkin erişim araçları tasarlanması (Fugini and Faustle, 1993; Chen et al., 1993; Dikenelli et al., 1995b), kitaplıktan seçilen tekrar kullanılabilir yapıların birleştirilerek yazılım haline getirilmesini sağlayan araçların tasarımı (Fugini et al., 1992; Hall and Weedon, 1993), tekrar kullanıma dayanan yazılım geliştirme sürecinin, bilgisayar ağları teknolojisi gözönüne alınarak geliştirilmesi (Dikenelli et al., 1995a) günümüz araştırma konuları arasındadır.

Yazılım mühendisliğinde tekrar kullanım alanında, teknik olmayan konular da önemli bir yer tutmaktadır (Prieto-Diaz, 1993). Bunlar; tekrar kullanımın yazılım kuruluşları politikalarına katılması, kültürel ve ekonomik konulardır.

2.3 Yazılım Mühendisliğinde Tekrar Kullanım Bakış Açıları

Prieto-Diaz (1993), tekrar kullanımı değişik bakış açılarına göre sınıflandırmıştır. Bu sınıflandırmanın amacı, yazılım alanında tekrar kullanım ile ilgili konuları, araştırmaları, problemleri değerlendirmek ve yorumlamak için bir temel oluşturmaktır.

Bundan sonraki 5 alt bölümün her birisinde, yazılım alanında tekrar kullanım değişik bir bakış açısından ele alınmaktadır.

2.3.1 Tekrar kullanılabilir birimlerin cinsine göre tekrar kullanım

2.3.1.1 Fikirlerin tekrar kullanımı

Belli problemlere ait genel çözümlerin, benzer problemlerde de tekrar kullanımıdır. Genel amaçlı algoritmalar bu gruba girmektedirler. Tekrar kullanımı kolaylaştırmak amacı ile bu algoritmaların nasıl tekrar kullanılabileceğine ilişkin ayrıntılı bilgileri de sağlamak gerekmektedir.

2.3.1.2 Parçaların tekrar kullanımı

Kalitesi ölçülmüş ve güvenilir parçaların, birçok yeni yazılım geliştirme sürecinde yapı taşları olarak kullanımıdır. Bazı parçaların kullanıcı isteklerine göre uyarlanması zorunda kalılabilmektedir. Nesneye dayalı yöntemler parçaların tekrar kullanımını desteklemektedir.

2.3.1.3 Yordam tekrar kullanımı

Geçmişte tasarlanmış uzman sistemlere ait bazı uzman yordam bilgilerinin, yeni uzman sistemlerin gerçekleştirilmesinde de kullanılabilmesi amacını gütmektedir ve bu konudaki araştırmalar başlangıç aşamasındadır.

2.3.2 Uygulama alanının özelliklerine göre tekrar kullanım

2.3.2.1 Dikey tekrar kullanım

Belli bir uygulama alanı içinde gerçekleşen tekrar kullanımdır. Bu yaklaşımda belli bir uygulama alanı analiz edilerek, bu alan içinde tekrar kullanılabilir yapılar ve soyutlamalar ortaya çıkarılmaktadır. Büyük ve uzun süreli projelerde bu yaklaşım kullanılmaktadır.

2.3.2.2 Yatay tekrar kullanım

Yatay tekrar kullanımın amacı, değişik uygulamalarda genel tekrar kullanılabilir yapıların veya soyutlamaların kullanılabilmesidir. Bu yaklaşımda, geniş erişim olanakları olan kitaplıklar kurulması ve bu kitaplıkların birbiriyle bağlantısının sağlanması önem kazanmaktadır.

2.3.3 Gerçekleşme şekline göre tekrar kullanım

2.3.3.1 Sistematiğe ve planlı tekrar kullanım

Bu tekrar kullanım yaklaşımında, tekrar kullanım bir takım kurallar içinde gerçekleşmekte ve tekrar kullanım verimliliğini ölçmek için birtakım ölçütler getirilmektedir. Planlanmış tekrar kullanım, belli bir yatırım ve belli bir kuruluş politikası saptanması gerektirmektedir. Ayrıca alışlagelen yazılım geliştirme yöntemlerinden daha farklı bir yaklaşım olduğu için, yazılım geliştiriciler kendilerini planlanmış tekrar kullanıma uyarlamak zorundadırlar.

2.3.3.2 Belli bir sistematiğe dayanmayan tekrar kullanım

Tekrar kullanım amacıyla hazırlanmamış yapıların ve soyutlamaların belirli kısımlarının bir takım kurallara bağlı kalmadan yeni yazılım geliştirme süreçlerinde kullanılmasıdır.

2.3.4 Tekrar kullanım teknikleri

2.3.4.1 Birleştirmeye dayanan tekrar kullanım

Daha önceden varolan tekrar kullanılabilir parçaların yeni sistemlerin oluşturulmasında yapı taşları olarak kullanılmasını sağlamaktadır. Bu yaklaşımda etkin kitaplık sistemlerine gereksinim duyulmaktadır. Tekrar kullanılabilir parçaların seçimi, gerektiğinde uyarlanabilmesi ve en önemlisi birleştirilerek yeni bir yazılıma döndürülmesi için etkin araçlar tasarlanmalıdır.

2.3.4.2 Üretken tekrar kullanım

Uygulama üreticilerindeki (Bilgisayar Destekli Yazılım Mühendisliği araçları) tekrar kullanım yaklaşımıdır. Kullanıcıdan analizleri alarak, kendi içinde tanımlı bir takım genel algoritmalar yardımıyla kod üretilmektedir.

Draco (Freeman, 1987) yaklaşımında, bir uygulama alanına ilişkin analiz kullanılarak, o uygulama alanına ilişkin biçimsel bir dilbilgisi yapısı ortaya çıkarılmakta, daha sonra bu dilbilgisi yapısına bağlı kalarak kod üretilmektedir.

Bu konudaki araştırmalar, belli bir uygulama alanına ait uygulama üreticilerini üretebilecek genel üreticilerin tasarımı üzerinde yoğunlaşmıştır.

2.3.5 Birimlerin Tekrar Kullanım Şekilleri

2.3.5.1 Siyah kutu yöntemi

Tekrar kullanılabilir yapı veya soyutlamalar üzerinde hiçbir değişiklik yapılmadan gerçekleşen tekrar kullanımdır. Bu tür yapı veya soyutlamalar standart arayüzlere sahip olacak biçimde tasarlanmalıdır. Bu yaklaşım güvenilir ve nitelikli yazılımlar ortaya çıkmasını sağlamaktadır, fakat bu tür yapı ve soyutlamaları tasarlamak zor olmaktadır.

Siyah kutu tekrar kullanımı için gerçekleştirilen yapı ve soyutlamaların sınanması, güvenilirliklerinin değerlendirilmesi için belli ölçütler getirilmesi üzerinde çalışmalar yapılmaktadır.

2.3.5.2 Beyaz kutu yöntemi

Bu yöntemde, tekrar kullanılabilir yapı veya soyutlamalar kullanıcı isteklerine göre değiştirilip, kullanıcı ortamına uyarlanabilmektedir. Bu yaklaşım analiz ve tasarımların tekrar kullanımını daha fazla desteklemektedir.

2.4 Tekrar Kullanımda Krueger Sınıflaması

Yazılım sürecinde tekrar kullanım yaklaşımları (Krueger, 1992) tarafından değişik sınıflar altında toplanmıştır. Bu sınıflar arasında, yüksek düzeyli diller, dönüşüm sistemleri, yazılım şemaları, kaynak kod kitaplıkları, uygulama üreticileri (Bilgisayar Destekli Yazılım Mühendisliği araçları), yazılım mimarileri (kullanıcı arayüzleri, veri tabanları, vb.) sayılabilmektedir. Bundan sonraki alt bölümlerde, değişik yaklaşımlar kısaca incelenmektedir.

2.4.1 Yüksek düzeyli dillerde tekrar kullanım

Birleştirici dili (Assembler) ile yazılan programlarda çok sık kullanılan ve belli birleştirici dili komutları dizisinden oluşan birtakım kalıplar, yüksek düzeyli dillerdeki temel yapılara karşılık gelmektedir. Bu nedenle, yüksek düzeyli bir dildeki tekrar kullanılabilir yapılar birleştirici dili kalıplarıdır. Yüksek düzeyli bir dildeki tekrar kullanılabilir yapıların (dilindeyimleri) seçimi dile ilişkin kullanma kılavuzundan yapılabilmektedir. Programcılar, dilindeyimlerindeki parametrelendirilmeye uygun boşlukları belli bir şekilde doldurarak tekrar kullanılabilir yapıları isteklerine göre uyarlamaktadır. Deyimler, dilinde genel kuralları çerçevesinde birleştirilerek programlar ortaya çıkmaktadır.

Yüksek düzeyli dillerde tekrar kullanım, birleştirici dilinin yazılım geliştirme sürecinin kodlama aşamasını karmaşık bir hale getirmesini önlemektedir. Bu yaklaşım, programcılar tarafından bir tekrar kullanım bilinci gerektirmeden uygulanmaktadır.

2.4.2 Tasarım ve kaynak kod parçaları alıntısı

Birçok programcı, varolan yazılımlardan belli parçaları alarak kendi geliştirdiği yazılımın içine katma yoluna gitmektedir. Alınan parça, bir tasarımın gerçekleştirimi kadar büyük ise tasarım alıntısı, aksi halde kaynak kod alıntısı sözkonusudur. Tekrar kullanıcı, alıntı yaptığı parçayı kendi ortamına uyarlamak için en alt ayrıntılarına kadar anlamak zorundadır. Bu durum zaman kaybına neden olacağı için zararlı olmaktadır. Bu yaklaşımdan en fazla yararı sağlamak için, alıntı yapılan parçaların en az değişiklikle kullanıcının yazılımına katılabilir nitelikte olması gerekmektedir.

2.4.3 Kaynak kod kitaplığı

Bu yaklaşım, daha önceden tekrar kullanım amacıyla hazırlanmış ve kaynak kod içeren yapıları birleştirerek daha büyük yazılımlar geliştirmeye dayanmaktadır. Bir önceki yaklaşımdan farkı, tekrar kullanılabilir yapıların tekrar kullanım amacıyla tasarlandıktan sonra sistematik bir şekilde kitaplıkta saklanmasıdır. Bu kitaplığa erişen tekrar kullanıcıların, kitaplığı arayabilmeleri ve istedikleri yapıyı seçebilmelerini sağlayan araçlar bulunmalıdır. Tekrar kullanıcılar seçtikleri bileşenleri birbirleriyle veya geliştirmekte oldukları yazılımla birleştirebilmelidir.

Bu yaklaşımın en önemli zararı, tekrar kullanılabilir yapıların sadece kaynak kod içermesidir. Tekrar kullanım, yazılım geliştirme sürecinin tüm aşamalarına katılmalıdır.

2.4.4 Yazılım şemaları

Yazılım şemaları, kaynak kod yerine algoritmalar ve veri yapılarının tekrar kullanımını desteklemektedir. Bir yazılım şemasının işlevinin soyut bir gösterimle tanımlanması, hangi koşullar altında ve nasıl tekrar kullanılabileceğinin belirlenmesi önemlidir. Tekrar kullanılabilir yazılım şeması, belirli parametrelerle özelleştirilmekte ve özelleştirilmiş şemanın işletimi ile kod üretimi sağlanmaktadır. Şemaların işlevlerini tanımlayabilmek için geliştirilen soyut gösterimler bir şemalar kitaplığında saklanabilmektedir. Kitaplıktan istenilen özellikte bir şema seçebilmek ve seçilen şemayı kullanıcı isteklerine göre özelleştirebilmek için araçlar tasarlanmalıdır. Bu yaklaşımın kötü yanı, şemaların işlevlerini tanımlayan gösterimlerin karmaşık olduğu durumlarda belli bir şemanın seçimi ve anlaşılabilmesinde ortaya çıkabilecek zorluklardır.

2.4.5 Uygulama üreticileri

Yazılım şemaları, algoritma ve veri yapılarının tekrar kullanımını desteklerken, uygulama üreticileri bir adım daha ileri giderek tüm bir yazılım sistemi tasarımının tekrar kullanımına dayanmaktadır. Uygulama üreticisi sistem analizini girdi olarak alır ve kaynak kodu çıktı olarak üretir. Bu durumda tekrar kullanım otomatik olarak kullanıcının dışında gerçekleşmektedir. Birçok uygulama için istenen özellikte ve verimlilikte uygulama üreticisi bulmak zordur. Her uygulama üreticisi ancak sınırlı bir uygulama alanında çalışmaktadır.

2.4.6 Çok yüksek düzeyli diller

Bu diller 'işletilebilir özellik-belirleme dilleri' olarak da adlandırılmakta ve analizlerin işletilebilir koda dönüştürülmesi açısından uygulama üreticisine benzemektedir. Bu dillerin uygulama üreticilerinden farkı genel amaçlı, uygulama alanı bağımsız ve matematiksel soyutlamalar kullanmalarındadır. Kullanıcının bir yazılım sisteminin analizini dilin genel ve matematiksel soyutlamalarını kullanarak hazırlaması gerekmektedir. Kaynak kod, analizden kendiliğinden olarak üretilmektedir. Bu yaklaşımın uygulama üreticilerinden üstünlüğü, belli bir uygulama alanına bağımlı olmamasıdır. Matematiksel soyutlamalar ile analizleri yapmanın her

kullanıcı için kolay olmayışı ve soyutlamalardan elde edilen kodun işletim zamanı veriminin kötü olması (Krueger, 1992) bu yaklaşımın kötü yanları olarak görülmektedir.

2.4.7 Dönüşümsel sistemler

Dönüşümsel sistemler, çok yüksek düzeyli bir dil ile ifade edilmiş analizlerden üretilen kaynak kod parçalarının anlamsal olarak eşdeğer fakat daha etkin kod parçalarına dönüştürülmesi amacını gütmektedir. Dönüşümler ile çok yüksek düzeyli dillerdeki işletim zamanı verimliliği sorunu ortadan kaldırılmaya çalışılmıştır.

Dönüşümsel sistemlerde tekrar kullanılabilir yapılar, tek bir dönüşüm ya da dönüşümler grubu olabilmektedir. Bir P programına T_1 dönüşümü uygulanması, $P \times T_1 \rightarrow P'$ (P' : P programına anlamsal olarak eşdeğer fakat daha etkin çalışan bir program) olarak gösterilmektedir. Dönüşümlerin, kaynak kodlara benzer şekilde bir kitaplıkta saklanarak ileride tekrar kullanımları sağlanabilmektedir.

2.4.8 Yazılım mimarileri

Yazılım mimarileri en geniş tekrar kullanım birimleridir. Uygulama alanları analiz edilerek ilgili genel tasarımlar ortaya çıkarılmaktadır. Daha sonra bu tasarımlar gerek tekrar kullanılabilir parçaların birleştirilmesinde yol gösterici olarak, gerekse kod üretiminde kullanılabilmektedirler. Yazılım mimarilerinin kod üretiminde kullanılabilmesi, uygulama üreticilerini de kapsadığının göstergesidir. Uygulama üreticilerinden farklı olarak, bir yazılım mimarisi daha üst düzeyli mimarilerin tasarımında yapı taşı olarak kullanılabilmektedir.

Değişik uygulamalarda kullanılan veri tabanı sistemleri, kullanıcı arayüzü mimarileri (x-windows, vb.) yazılım mimarilerine ait örneklerdir.

Yazılım mimarileri, diğer yaklaşımların bazılarında olduğu gibi erişimi ve aramayı etkinleştirecek şekilde düzenlenerek bir kitaplıkta saklanabilmektedir. Genel amaçlı, sınanmış,

nitelikli yazılım mimarilerinden oluşan bir kitaplık kurmanın güçlüğü (Krueger, 1992) bu yaklaşımın kötü yanıdır.

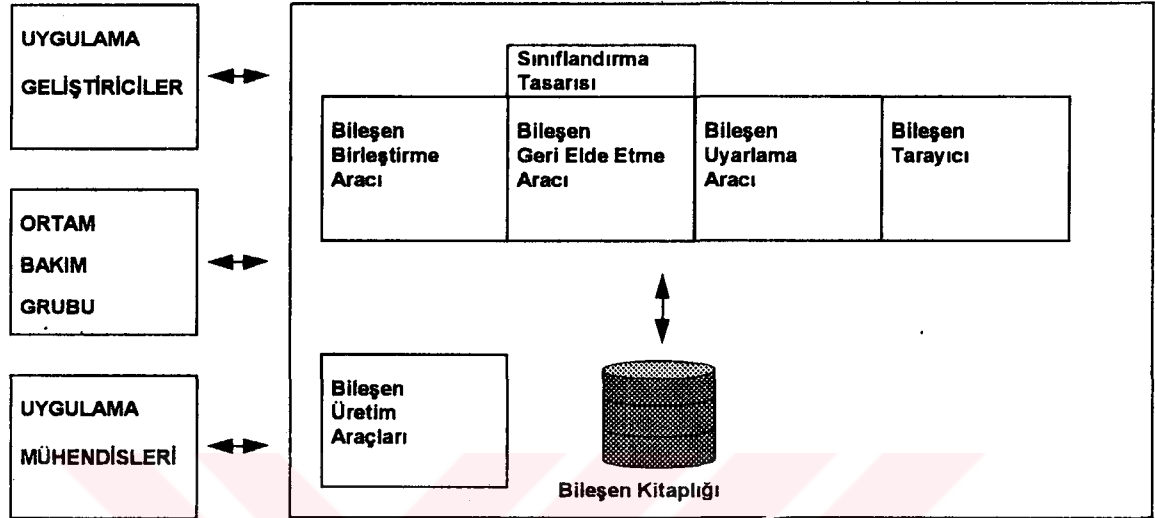


BÖLÜM-3 BİLEŞEN YÖNELİKLİ YAZILIM GELİŞTİRME

Nesneye dayalı analiz ve tasarım yöntemleri ile nesneye dayalı programlamanın yaygınlaşması sonucunda yazılım mühendisliğinde yeni bir yazılım geliştirme yaklaşımı ortaya çıkmıştır. Bu yaklaşım "Bileşen Yönelikli Yazılım Geliştirme" (BYYG) olarak adlandırılmaktadır. Bu yaklaşımın en önemli amaçlarından birisi yazılım yaşam döngüsünün tüm ürünlerini tekrar kullanım sürecine katmaktır. Bu nedenle, kaynak kod yanında, analizler, tasarımlar, sınamaya senaryoları, belgeler kısaca bir yazılım yaşam döngüsünün her türlü ürünü tekrar kullanılabilir. Yazılım yaşam döngüsünün tekrar kullanılabilen her türlü ürünü yazılım bileşeni (Fugini et al., 1992; Nierstrasz et al., 1992; Sorumgard et al., 1993; Weber, 1993) olarak nitelendirilmektedir. BYYG yaklaşımında, tekrar kullanılabilir analiz ve tasarımlardan faydalanılarak oluşturulan tasarımların gerektirdiği kaynak kodları içeren bileşenler, belgelerden de yararlanarak seçilmekte ve birleştirilmektedir. Birleştirilecek olan kaynak kodlar, fonksiyon, sınıf veya sınıflar topluluğu olabilmektedir. Sonuçta oluşan yazılımlar da tekrar kullanılabilir sınamaya senaryolarından yararlanarak sınanabilmektedir. BYYG, tekrar kullanım tekniği açısından birleştirmeye dayanan bir tekrar kullanım şeklidir. Krueger sınıflamasında sözedilen kaynak kod kitaplıkları fikrini temel alan fakat bu fikri yazılım yaşam döngüsünün tüm ürünlerinin tekrar kullanımı ile geliştirmiş olan bir yöntemdir.

BYYG yönteminin uygulanabilmesi için bir BYYG ortamı tasarlanmalıdır. Bu ortam; bileşen kitaplığı, tekrar kullanımı destekleyici yazılım araçları, bir yazılım geliştirme yöntemi, uygulama geliştiriciler (tekrar kullanıcılar), uygulama mühendisleri ve ortamın bakımını sağlayan bir gruptan oluşmaktadır. Uygulama mühendisliği kavramı (Fugini et al., 1992), belirli uygulama alanlarında tekrar kullanılabilir bileşenler üretmeyi açıklamaktadır. Uygulama geliştiriciler ise uygulama mühendisleri tarafından üretildikten sonra sistematik bir şekilde bileşen kitaplıklarında saklanan bileşenleri kullanarak yazılım geliştiren kişilerdir. Ortam bakımını sağlayan grup değişen koşullara göre tekrar kullanımı destekleyici araçlar üzerinde gerekli değişiklikleri yapmak, kitaplıkta bulunan bileşenlerin kullanım sıklıklarını izleyerek hiç kullanılmayan bileşenlerin kitaplıktan çıkarılmasına karar vermek, yeni bileşenler eklenmesi durumunda kullanılan sınıflandırma tasarısına göre kitaplığı yeniden düzenlemek gibi görevlere sahiptir. Yukarıda sözü edilen birimleri bulunan bir BYYG ortamı Şekil 3.1'de görülmektedir.

BYYG ortamında bulunan tekrar kullanımı destekleyici araçların görevleri aşağıda kısaca açıklanmıştır:



Şekil 3.1 : Örnek BYYG Ortamı

a) Bileşen tarayıcı araç, bileşen kitaplığının herhangi bir bileşenden başlanarak ve o bileşenin ilişkili olduğu diğer bileşenlere ulaşmak yolu ile taranmasını sağlayan ve kullanıcıyı görsel bir arayüzle destekleyen bir araçtır.

b) Bileşen geri elde etme aracı, belli bir sınıflandırma tasarısına göre düzenlenmiş bileşen kitaplığına etkin bir erişim sağlamak amacı gütmektedir. Kullanıcı, isteğini bir sorgu şeklinde bileşen geri elde etme aracına vermektedir. Bu araç, sorguyu işleterek, kullanıcının istediği özelliklere sahip bileşenleri kullanıcıya sunmaktadır. Bir geri elde etme aracı belli bir sınıflandırma tasarısına dayanmaktadır. Bu sebeple kullanıcıların sorgularını sınıflandırma tasarısını dikkate alarak oluşturmaları gerekmektedir.

c) Bileşen uyarılama aracı, kitaptan seçilen bileşen, kullanıcı ortamına taşınmadan önce bileşen üzerinde yapılması gereken değişikliklerin gerçekleştirilmesini sağlamaktadır. Bu aracın

kullanılması zorunlu değildir. Siyah kutu yöntemine göre üretilen bileşenlerin bulunduğu ortamlarda bu tür bir araca gerek duyulmamaktadır.

d) Bileşen birleştirme aracı, kitaplıktan seçilen ve gerektiğinde uyarlanan bileşenlerin birleştirilerek bir yazılıma döndürülmesi amacını gütmektedir. Bu tür araçların, kullanıcıya bileşenlerin birleştirilmesi sırasında görsel arayüzler sunması, aynı zamanda değişik dillerde veya işletim sistemleri altında üretilmiş olan bileşenlerin bir araya getirilmesini desteklemesi gerekmektedir. Bu konuda bazı modeller geliştirilmiştir. Vista bu tür bir araca örnektir (Fugini et al., 1992).

e) Bileşen üretim aracı, sadece uygulama mühendisleri tarafından tekrar kullanılabilir bileşen üretiminde kullanılmaktadır. Bu tür bir aracın uygulama alanı analizi (domain analysis) gibi bileşen üretiminde kullanılan bazı teknikleri desteklemesi gerekmektedir.

BYYG ortamı içinde ve yukarıda kısaca açıklanan araçların kullanımına dayanan bir yazılım geliştirme yöntemi aşağıdaki adımlardan oluşmaktadır:

1) Kullanıcılar, kullanılan sınıflandırma tasarısı hakkında bilgi sahibi olmalı ve bir ön analiz çalışması yapmalıdırlar. Sınıflandırma tasarısına ve ön analize uygun olarak isteklerini bir sorgu şeklinde bileşen geri elde etme aracına vermelidirler. Kullanıcılar, bileşen tarayıcının sunduğu görsel arayüzü kullanarak da bir bileşen seçebilmektedirler.

2) Geri elde etme aracı tarafından istenen özellikte veya yakın özelliklere sahip bileşen(ler) kullanıcıya sunulmaktadır. Bu bileşenler ayrıntılı analiz ve tasarım içermektedir.

3) Kullanıcı, analiz ve tasarımın gerektirdiği kaynak kod içeren bileşenlere erişmek için de aynı şekilde bileşen geri elde etme aracını kullanmaktadır. Bir bileşeni elde ettikten sonra, eğer gerekiyorsa, bileşen uyarlama aracı ile bileşen üzerinde değişiklik yapabilmektedir. Bileşen üzerinde değişiklik yapma kararı, bileşene ait belgeleri içeren diğer bileşenler incelendikten sonra verilmektedir.

4) Son aşama, birleştirici araç kullanılarak bileşenlerin birleştirilmesi ve bir yazılıma dönüştürülmesidir.

Bileşen kitaplığı ve etrafında bulunan tekrar kullanımı destekleyici araçlar BYYG süreci için temel unsurlar olmakla beraber, etkin bir tekrar kullanımı sağlamak için kuruluş düzeyinde bir işleyiş belirlenmelidir. Bu amaçla aşağıda bir işleyiş önerilmektedir. Şekil 3.2'de görülen işleyiş içindeki birimler ve işleyiş hakkında bilgiler aşağıda verilmiştir.



Şekil 3.2 : Kuruluş Düzeyinde BYYG

Yönetim Grubu:

Tüm birimlere mali desteği sağlamak ve tekrar kullanımı artırmak için kuruluş politikaları belirlemek yönetim grubunun görevidir. Örneğin, kuruluştaki programcıları ödüllendirmek için yazılan kodun uzunluğuna değil, tekrar kullanılan bileşen sayısına bakılması bu tür bir politikaya örnek olarak verilebilmektedir.

Eğitim Grubu:

Kullanıcılara ortamdaki araçların kullanımı, yazılım geliştirme yöntemi ve ortamda bulunan bileşenler hakkında ayrıntılı bilgi sağlamalıdır. Bu grup ayrıca, yönetim tarafından belirlenen tekrar kullanım politikalarının kuruluşta uygulanmasından, uygulama geliştiricilerin kendilerini BYYG ortamına uyarlamalarından sorumludur. Kullanıcılar her türlü sorunlarını eğitim grubuna iletmektedir. Uygulama mühendisleri, kuruluşta yürütülen projeler çerçevesinde gereksinim duyulacak tekrar kullanılabilir bileşenlerin belirlenmesi, tasarımı, gerçekleştirimi ve sınanmasından sorumlu olan gruptur.

Bir BYYG ortamında bileşen üretimi üzerinde önemle durulmalıdır çünkü üretilen bileşenler geliştirilen yazılımların yapı taşlarını oluşturmaktadır. Bu çalışmada bileşen üretimi ayrıntılı olarak incelenmemekte fakat bölüm 3.1'de, bileşen üretiminde kullanılan bazı yöntemler hakkında bilgiler verilmektedir.

Bu tez çalışmasının amaçlarından biri BYYG ortamında bulunan yüksek verimli bileşen geri elde etme aracı tasarımı ve gerçekleştirmidir. Bu gerçekleştirim ile ilgili bölüme temel oluşturması amacıyla, BYYG ortamındaki sınıflandırma tasarımları ve bileşen geri elde etme araçları konusunda daha önce yapılan çalışmalar bölüm 3.2'de anlatılmaktadır.

3.1 Tekrar Kullanılabilir Bileşenlerin Üretimi

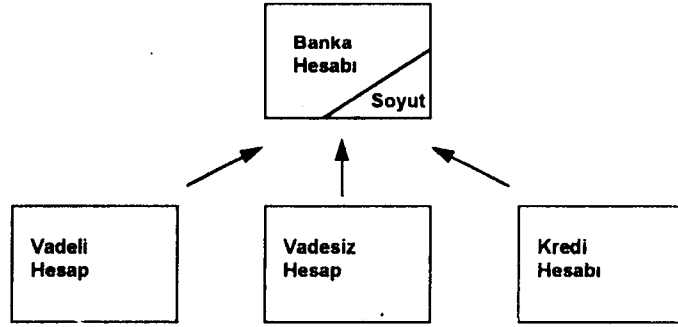
Tekrar kullanılabilir bileşenlerin belli standartlarda ve nitelikte üretilmesi zorunluluğu tekrar kullanılabilir bileşen üretimini, klasik yazılım geliştirme sürecinden farklı kılmaktadır. Tekrar kullanılabilir bileşen üretimi zaman ve emek isteyen tekrarlı bir süreçtir. Bu nedenle klasik yazılım geliştirme süreçlerinde kullanılan yöntemler tekrar kullanılabilir bileşen üretimini destekleyecek şekilde geliştirilmelidir (Gossain and Anderson, 1990). Bu yöntemler, tekrarlı tasarım ve sınıf sıradüzenlerini destekleyici özellikler katılarak geliştirilmektedir. Nesneye dayalı yaklaşım sahip olduğu birtakım özelliklerden dolayı tekrar kullanımı desteklemektedir. Nesneye dayalı yaklaşımın tekrar kullanıma katkı veren özellikleri bölüm 3.1.2'de incelenmektedir. Aşağıda, tekrar kullanılabilir bileşen üretimi için bir yöntem verilmektedir. Bileşen kelimesi analiz, tasarım, kod ve

belgelemeyi kapsadığı için, bileşen üretimi tekrar kullanılabilir yazılım üretimi ile aynı kavramlar olmaktadır. Verilecek olan yöntem, Gossain ve Anderson (1990) tarafından ortaya konan tekrarlı tasarım modeli ve bir nesneye dayalı analiz ve tasarım modeli olan birleştirme yöntemi (Coleman et al., 1994) temel alınarak oluşturulmuştur.

3.1.1 Tekrar kullanılabilir yazılım üretimi için bir yöntem

Tekrar kullanılabilir bileşenler, sadece belli bir anda geliştirilmekte olan yazılımların değil, aynı zamanda gelecekte geliştirilecek olan birçok yazılımın yapı taşları olarak düşünülmelidir. Bu yüzden tekrar kullanılabilir bileşen üretme sürecinin ilk aşaması, bileşenlerin geliştirilmekte oldukları uygulama alanına ilişkin genel özelliklerin belirlendiği aşama olmalıdır. Bu aşama, uygulama alanı analizi (domain analysis) olarak nitelendirilmektedir. Uygulama alanı analizi, sözkonusu uygulama alanındaki tüm sistemler için ortak işlemleri tanımlayarak, o uygulama alanına ilişkin nesnelerin tüm karakteristiklerini ortaya koyan bir uygulama alanı modeli çıkartmak amacıyla yapılmaktadır (Gossain and Anderson, 1990). Uygulama alanı analizi yapılarak, geliştirilen tekrar kullanılabilir bileşenlerin sadece belirli bir yazılım geliştirme sürecine ait olması engellenmiş olmaktadır. Sadece temel veri yapıları ve algoritmalar içeren bileşenler, belli bir uygulama alanı dışında kaldıklarından bu tür bileşenler için uygulama alanı analizi yapmaya gerek yoktur. Nesneye dayalı yaklaşımlarda analiz aşamasının amacı problem alanına ilişkin modeller ortaya çıkartmaktır. Bu nedenle, uygulama alanı analizinin gerçekleştirilmesi ile tekrar kullanılabilir bileşen üretiminin analiz aşaması sona ermiş olmaktadır.

Tasarım aşaması, birkaç alt adımdan oluşmaktadır. Alt adımlardan ilki soyutlama adımdır. Bu adımın amacı, uygulama alanına ait soyut sınıfların elde edilmesidir. Soyut sınıflar, bir sınıfın arayüzü ile o sınıftan türetilebilecek alt sınıfları belirlemektedir. Soyut sınıflar sadece belirli davranışlara sahiptir, gerçekleştirmeleri türetilen alt sınıflarda yapılmaktadır. Soyut sınıf belirleme işlemi belirli özellikleri taşıyan sınıfların bulunması işlemidir. Bu özellikler arasında ortak temel nitelikler, davranışlar ve amaçlara sahip olma sayılabilmektedir. Ortak temel nitelikler, bir sınıflar topluluğunu diğerlerinden ayırtetmede kullanılabilecek niteliklerdir. Örneğin bir banka sistemindeki vadeli hesap, vadesiz hesap, kredi hesabı gibi değişik sınıflar topluluğunun ortak niteliği 'hesap' olma durumudur. Bu nedenle, bu sınıflar topluluğu 'banka hesabı' isimli bir soyut sınıf ile modellenenilmektedir. Bu sıradüzen Şekil 3.3'de görülmektedir.



Şekil 3.3 : Soyut Sınıf İçeren Bir Sıradüzen

İkinci alt adım ise özelleştirme (specialization) adıdır. Bu adımda, önceki adımda belirlenen soyut sınıflardan yararlanarak uygulama alanına ait somut sınıflar ortaya çıkarılmaktadır. Bu işlem, soyut sınıflardan kalıtım yoluyla yeni sınıflar türetilmesi şeklinde yapılmaktadır ve türetilen sınıflar arasındaki ilişkiler ve bağımlılıklar da belirlenmektedir. Örneğin, banka hesabı isimli soyut bir sınıftan türetilen vadeli, vadesiz ve kredi hesabı isimli üç alt sınıf bu adımın ürünleridir (Bkz. Şekil 3.3).

Tasarım aşamasının son alt adımı ise değerlendirme ve gerekli değişikliklerin yapılması adıdır. Bir önceki adımda belirlenen tüm somut sınıfların arayüzleri, sıradüzendeki yerleri, diğer sınıflarla olan ilişkileri, temsil ettikleri kavramlar incelenerek iki tür değişiklik yapılmaktadır. Bu değişiklikler sınıflar-arası ve sınıf-içi değişikliklerdir. Sınıflar-arası değişiklikler; birbirine çok benzer işlevlere sahip sınıfların birleştirilmesi, bazı sınıflar özelleştirilirken bazılarının biraz daha genelleştirilmesi, birden fazla ve farklı kavramı temsil eden bir sınıfın alt sınıflara ayrılması gibi işlemlerden oluşmaktadır. Sınıf-içi değişiklikler ise sınıflar-arası değişikliklerin sonucunda zorunlu hale gelmektedir. Sınıf sıradüzendinde değişiklikler olması, bazı sınıflara değişik fonksiyonlar eklenmesi veya bazılarının çıkarılması, arayüzlerin değişmesi sebebiyle bazı sınıflara özel üyeler (private members) eklenmesi, bazılarının üye fonksiyonlarının parametrelerinin değiştirilmesi gerekmektedir.

Tasarım aşamasından sonra kodlama ve sınamaya adımlarından oluşan gerçekleştirme aşaması gelmektedir. Dikkatli ve mantıksal çerçevede yapılmış bir tasarım bu aşamayı kolaylaştırmaktadır.

Üretilen tekrar kullanılabilir bileşenler arasında belgeleri içeren bileşenlerin de bulunması gerekmektedir. İyi belgeleme, bileşenin kullanıcılar tarafından daha kolay anlaşılmasını sağlamaktadır. Bileşenler için üç tip belge hazırlanmalıdır. Bunlar bileşen veri belgesi, bileşen topluluk belgesi ve bileşen tanımlama belgesidir (Jacobson et al., 1993). Bileşen veri belgesi, bileşen hakkında açıklayıcı bilgiler vererek hangi şartlar altında tekrar kullanılabilirliğini belirlemektedir. Bileşen tanımlama belgesi, bileşenin alt düzey gerçekleştirim ayrıntılarını kapsamaktadır ve bileşenin kullanıcılar tarafından uyarlanabilmesini kolaylaştırmaktadır. Bileşen topluluk belgeleri, bileşenlerin bir kitaplıkta toplandığı durumlarda hazırlanmaktadır. Bu tür bir belge, tekrar kullanılabilir bileşen kitaplığı hakkında ayrıntılı bilgileri içermektedir.

3.1.2 Nesneye dayalı yaklaşımın tekrar kullanımı destekleyici özellikleri

Nesneye dayalı yaklaşım, yazılım geliştirme sürecinin karmaşıklığını azaltan ve tekrar kullanımı destekleyen özelliklere sahiptir. Aşağıda bu özelliklerin neler olduğu ve tekrara nasıl katkı sağladığı anlatılmaktadır.

Soyutlama özelliği ile gerçek hayat olayları ve nesneleri daha kolay modellenen ve nesneleri birbirinden ayıran tüm karakteristiklerin belirlenmesi sağlanmaktadır. Belirli bir soyutlamaya sahip bileşenlerin kullanıcılar tarafından anlaşılması daha kolay olmaktadır.

Bileşenlerin dış dünyaya bir takım arayüzler yolu ile belli işlemleri sağlaması beklenmektedir ve koruma (encapsulation) özelliği bunu desteklemektedir. Ayrıca bu özellik, arayüzler sayesinde gerçekleştirim ayrıntılarını kullanıcılardan saklama amacı güttüğü için, kullanıcılar gerçekleştirim ayrıntılarını incelemekten kurtulacaktır. Bu durumun tam tersi de düşünülebilmektedir; gerçekleştirim kısmında kullanıcının yapacağı değişiklikler arayüzü değiştirmeyeceği için geliştirilen uygulamanın diğer kısımları etkilenmemektedir.

Genel kalıplar (class templates), genel gösterim özelliği sağlayarak tekrara katkıda bulunmaktadır. Örneğin, bir yığıt sınıfına, yığıtın içine konacak veri türü parametre ile geçirilerek ilgili fonksiyonda kullanılması sağlanmaktadır. Böylece yığıtın içine konacak her veri türü için değişik bir sınıf tanımlanması önlenmektedir.

Çoklu-gerçekleştirim (Polymorphism) özelliği, belli bir isimdeki fonksiyonun, kalıtım ile birbiriyle ilişkili bir sınıf kümesinde değişik gerçekleştirmeleri olabileceğini göstermektedir. Böylece, seçilen tekrar kullanılabilir bir fonksiyon değişik sınıflar içinde hiçbir değişikliğe gerek kalmadan kullanılabilir.

Kalıtım özelliği kullanılarak, bir bileşenden istenen özellikte yeni bileşenler türetilmektedir. Böylece bileşenin yeni ortamlara uyum sağlaması için değiştirilmesi kolaylaştırılmış olmaktadır. Fakat kalıtım özelliğinin dikkatli kullanılması gerekmektedir. Çok karmaşık ve çok fazla sayıda sınıftan oluşan bir sıradüzen tekrar kullanımı güçleştirmektedir.

3.2 Sınıflandırma Tasarıları ve Bileşen Geri Elde Etme Araçları

BYYG sürecindeki en önemli problemlerden birisi, binlerce bileşenden oluşan bir kitaplıktan istenilen özellikte bileşenlerin seçilmesidir. Bu soruna bir çözüm getirebilmek için, kitaplığın belli bir sınıflandırma tasarısına göre düzenlenmesi ve bu sınıflandırma tasarısını kullanan bileşen geri elde etme araçlarının tasarlanması gerekmektedir. Tekrar kullanılabilir bileşenler için değişik sınıflandırma tasarıları bölüm 3.2.1'de, boyutlara dayanan (faceted) sınıflandırma tasarısı kullanan bir bileşen geri elde etme aracı ise bölüm 3.2.2'de incelenmektedir.

3.2.1 Sınıflandırma tasarıları

Sınıflandırma, benzer nesne veya kavramları gruplandırmaktır. Bu gruplandırma sonucunda oluşan bir sınıfın tüm üyeleri, diğer sınıftaki üyelerin sahip olmadığı en az bir ortak özelliğe sahip olmalıdır. Sınıflandırma, çok sayıda elemandan oluşan topluluklar için önem kazanmaktadır.

Bir sınıflandırma tasarısının sahip olması gereken birtakım özellikler vardır (Prieto-Diaz, 1991; Sorumgard et al., 1993). Bu özelliklerden bazıları aşağıda verilmektedir:

a) Bir yazılım geliştirme sürecinin tüm aşamalarına ilişkin bileşenlerin (analiz, tasarım, kod, belgelenmeler) sınıflandırılabilmesini sağlamalıdır.

b)Gerek kitaplık sorumluları gerekse kullanıcılar tarafından kolay anlaşılabilir nitelikte olmalıdır. Karmaşık bir sınıflandırma tasarısı tekrar kullanımı zorlaştırmaktadır.

c)Otomatikleştirilmeye uygun olmalıdır.

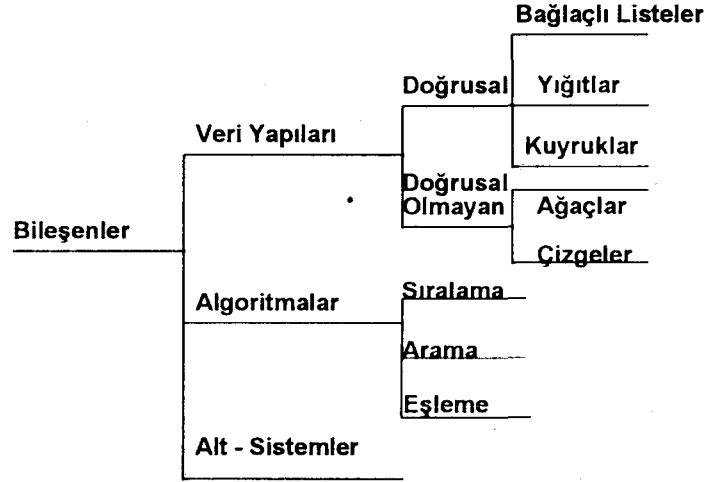
d)Sınıflandırılacak bilginin değişik niteliklerini belirlemeyi ve tanımlamayı destekleyici özellikte olmalıdır.

e)Sürekli gelişen ve büyüyen bileşen topluluklarına uygulanabilmelidir. Ayrıca, bazı bileşenlerin sınıflandırma tasarısından çıkarılması, bazılarının yeniden sınıflandırılabilmesi gibi işlemler kolaylıkla yapılmalıdır.

f)İstenen özellikte bileşenler bulunamadığı durumda, benzer veya yakın işlevdeki bileşenlerin bulunabilmesini desteklemelidir.

Sınıflandırma tasarıları, sıradüzensel tasarılar ve anahtar kelime yöntemine dayanan tasarılar olmak üzere iki grupta toplanmıştır (Jacobson et al., 1993). Sıradüzensel tasarılar, bileşenlerin bir ağaç yapısında sınıflandırılmasını öngörmektedir. Bir bileşen aranırken, ağacın kökünden başlanıp bileşende olması istenen özelliklere göre çeşitli dallara sapılmaktadır. En genel bileşen ağaç yapısının kökünde bulunmaktadır. Ağacın yapraklarından birine ulaşılması, bileşenin bulunduğu anlamına gelmektedir. Genel amaçlı birtakım algoritmalar ve temel veri yapıları ile alt-sistemlerden oluşan bir bileşen kitaplığının sınıflandırılması Şekil 3.4'de görülmektedir. Alt-sistemler, çeşitli veri yapıları ve algoritmaların birleşmesiyle oluşan birimlerdir.

Anahtar kelime yöntemine dayanan sınıflandırma tasarısı anahtar kelimelere dayalı aramayı temel almaktadır. Bu yöntemde, herbir bileşen için bir tanımlayıcı (anahtar) oluşturulmaktadır. Bu tanımlayıcılar standart ve önceden belirlenmiş bir sözlük içinden seçilen anahtar kelimelerden oluşmaktadır. Herbir bileşen tanımlayıcısı, bileşenler gibi saklanmakta ve daha sonra bileşenlere erişimde kullanılmaktadır.



Şekil 3.4 : Sıradüzensel Sınıflandırma Tasarısı

Anahtar kelime yöntemine dayanan sınıflandırma tasarıları iki grupta incelenmektedir (Sorumgard et al., 1993). Bunlar sayı verilmiş (enumerative) sınıflandırma ile boyutlara ayırmaya dayanan (faceted) sınıflandırmadır. Sayı verilmiş sınıflandırma tasarısında, bileşenlerin içinde yer alabileceği tüm alt sınıflar önceden belirlenerek, bir sınıflar listesi oluşturulmaktadır. Yeni bir bileşenin sınıflandırılması sözkonusu olduğunda, kitaplık görevlisi, mevcut sınıflar listesi içinden bileşenin sahip olduğu özellikleri kapsayan bir alt sınıf bulmaktadır. Bu alt sınıfın ismi bileşen tanımlayıcısı (anahtarı) olmakta ve daha sonra bileşene erişimde kullanılmaktadır. Örneğin, değişik bilgisayar sistemlerine ait alt birimlerin sayı verilmiş yaklaşımla sınıflandırılması sözkonusu olduğunda, gerekli tüm alt sınıflar belirlenmelidir. Mikrobilgisayarların birincil bellekleri sınıfı, mikrobilgisayarların merkezi işlem birimleri sınıfı, mikrobilgisayarların giriş/çıkış birimleri sınıfı, ana çerçeve bilgisayarların birincil bellekleri sınıfı, ana çerçeve bilgisayarların giriş/çıkış birimleri sınıfı sözkonusu alt sınıflardan bazılarıdır. Bir kişisel bilgisayar klavyesi veya faresi, mikrobilgisayarların giriş/çıkış aygıtları sınıfına girmektedir. Herhangi bir birimin sınıflandırılması için uygun bir alt sınıf bulunamadı ise, o birimin özelliklerini kapsayan yeni bir alt sınıf türetilmelidir. Kişisel bilgisayar sabit disk gibi bir birimi sınıflandırmak için kullanılabilecek bir alt sınıf yukarıda sayılan sınıflar arasında bulunmamaktadır. Bu nedenle, mikrobilgisayarların ikincil bellekleri isminde yeni bir alt sınıf türetilmelidir. Bu durum sayı

verilmiş sınıflandırma tasarısının esnekliğini ve sürekli gelişen topluluklara uygulanabilmesini kısıtlamaktadır.

Boyutlara ayırarak sınıflandırma yaklaşımı, bilginin sınıflandırma işleminde kullanılabilecek farklı niteliklerini gözönüne almaktadır. Her farklı nitelik bir boyut ile gösterilmektedir. Her boyut, temsil ettiği nitelik ile ilgili birtakım terimlere (anahtar kelimelere) sahiptir. Boyutların içindeki terimler, o boyuta ilişkin terimler uzayını oluşturmaktadır. Bu yaklaşımda, bir bileşen farklı boyutlardan seçilen terimlerin birleştirilmesinden oluşan tanımlayıcı (anahtar) ile sınıflandırılmaktadır.

Boyutlara ayırmaya dayanan yaklaşımın daha iyi anlaşılabilmesi için aşağıda bazı tanımlamalar yapılarak bir örnek verilmektedir.

B_i : i numaralı niteliğe ilişkin boyutu göstermektedir. Diğer bir deyişle , B_i , i numaralı nitelik üzerinde tanımlı terimlerin tümünün kümesidir.

T_{ij} : i numaralı niteliğe ilişkin boyuttaki j numaralı terime karşılık gelmektedir.

Buna göre, $B_i = \{ T_{ij} \mid k \geq j \geq 1 \}$ (k : i numaralı niteliğe ilişkin boyuttaki terim sayısı) şeklinde tanımlanmaktadır.

Bu yaklaşım bilginin sınıflamada kullanılabilecek farklı niteliklerini gözönüne aldığı ve sınıflanacak bilgi de tekrar kullanılabilir yazılım bileşenleri olduğu için, yazılım bileşenlerinin farklı nitelikleri ortaya konmalıdır. Aşağıdaki örnekte, bir yazılım bileşenin 3 nitelik ile tanımlandığı ve bu niteliklerin bileşenin işlevi, üzerinde işlem yaptığı nesneler ve kullandığı sistem türü olduğu düşünülmektedir.

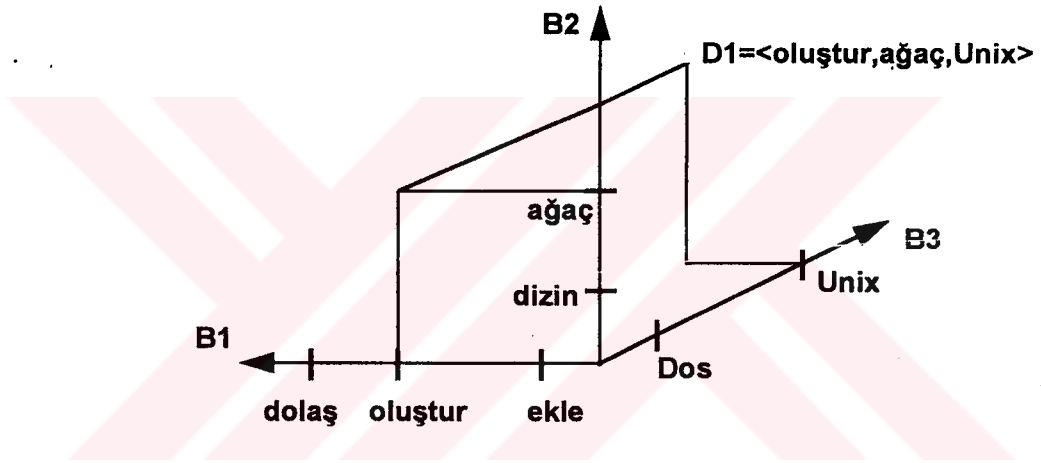
Örnek:

$B_1 = \{ \text{ekle, oluştur, dolaş, ...} \}$ (bileşen işlevi niteliği)

$B_2 = \{ \text{ağaç, dizin, ...} \}$ (üzerinde işlem yapılan nesneler)

$B_3 = \{ \text{Dos, Unix, ...} \}$ (sistem türü)

Niteliklere ilişkin boyutların yukarıdaki gibi olduğu bir durumda, Unix ortamında bir ağaç oluşturan bileşeni sınıflandırmak için $\langle \text{oluştur, ağaç, Unix} \rangle$ şeklinde bir tanımlayıcı (anahtar) oluşturmak gerekmektedir. Bu tanımlayıcı üç boyutlu bir sistemde Şekil 3.5'deki gibi görülmektedir.



Şekil 3.5 : Boyutlara Ayırmaya Dayanan Sınıflamada Tanımlayıcılar

Pricto-Diaz (1987), tekrar kullanılabilir yazılım bileşenlerini sınıflandırmada boyutlara ayırmaya dayanan sınıflandırma yöntemini kullanmış ve yazılım bileşenlerini tanımlamak için altı nitelik belirlemiştir. Bu nitelikler; bileşenin işlevi, bileşenin üzerinde işlem yaptığı nesneler (tamsayılar, karakter bilgi, satırlar, vb.), bileşenin işlevini yerine getirmede kullandığı daha üst düzeyde yapılar (ağaç, yığıt, tablo, kütük, vb.), bileşenin içinde bulunduğu ve birçok uygulamada ortak kullanılabilen modüller (matris ters alıcısı, vb.), bileşenin işlev alanı (veri tabanı yönetim sistemi, derleyici tasarımı uygulaması, vb.) ve uygulamanın gerçekleştirildiği yerlerdir (üniversite, yazılım evi, fabrika bilgi işlem bölümü, vb.).

Boyutlara ayırmaya dayanan sınıflandırma tasarısı, bir sınıflandırma tasarısında bulunması gereken özelliklerin çoğuna sahiptir. Yazılım geliştirme sürecinin tüm aşamalarına ilişkin ürünlerin sınıflandırılması, ürün isimli bir boyut ekleyerek mümkün olmaktadır. Bu boyut, analiz, tasarım, kod, belgeleme gibi bilgiler içermektedir. Bu yaklaşım, ayrıca gelişen bileşen topluluklarına çok kolay uygulanabilmekte ve sınıflandırma tasarısının bakımını kolaylaştırmaktadır. Yeni bir bileşeni sınıflandırmak için, mevcut boyutlardan bileşeni tanımlayacak birer terim seçerek bileşen tanımlayıcısı (anahtar) oluşturulmaktadır. Bazı durumlarda, bazı boyutlara yeni terimler eklenebilmektedir. Bir bileşenin, sınıflandırma tasarısından çıkartılması, tanımlayıcısının silinmesi ve gerekirse belirli terimlerin ilgili boyutlardan çıkartılmasını gerektirmektedir. Bu tür işlemler, sayı verilmiş yaklaşımda yeni alt sınıflar türetme veya bazı alt sınıfların kaldırılması gibi daha fazla zaman isteyen işlemlerdir. Bir boyutun terimler uzayının, terimler arasındaki ilişkileri ve kavramsal yakınlıkları ortaya koyabilecek bir yapıda tutulması benzer bileşenlerin elde edilmesini kolaylaştırmaktadır. Bu nedenlerle boyutlara ayırmaya dayanan yaklaşım, tekrar kullanılabilir bileşenleri sınıflandırmada etkin olarak kullanılmaktadır.

Boyutlara ayırmaya dayanan sınıflandırma yaklaşımının nesneye yönelik yöntemlerle oluşturulmuş bileşenlerin sınıflandırılmasında kullanımına yönelik çalışmalar da yapılmıştır (Sorumgard et al., 1993). Bu amaçla, nesneye yönelik yöntemlerle oluşturulan bileşenleri tanımlayacak nitelikler belirlenmiştir. Bu nitelikler; soyutlama, işlemler, işlenenler ve bağımlılıklar olarak adlandırılmıştır. Soyutlama ve işlemler nitelikleri iki alt boyuta sahiptir. Alt boyutlardan birincisi o niteliğe ilişkin ana terimleri içeren boyut, diğeri ise o niteliğe ilişkin yardımcı terimleri içeren boyuttur. Bileşenler için tanımlayıcılar oluşturulurken, gerekirse iki alt boyutu olan niteliklerin her alt boyutundan birer terim seçilebilmektedir. Böylece bir 'ana terim, yardımcı terimlerin eklenmesi ile değişik kavramları gösterebilmektedir. Örneğin, ana terimin 'arama', yardımcı terimlerin 'sıradan' ve 'ikili' olduğu bir durumda, ikili-arama ve sıradan-arama gibi iki farklı kavram anlatılabilmektedir.

Bu yaklaşımın daha iyi anlaşılabilmesi için yukarıda sözü edilen dört nitelik kısaca açıklanmaktadır. Soyutlama niteliğine ilişkin ilk boyutta bulunan ana terimler, sınıflanan bileşenin gerçek yaşamdaki işlevini belirlemek için kullanılmaktadır. Bu niteliğe ilişkin ikinci alt boyutta bulunan yardımcı terimler ise bileşenin yazılım yaşam döngüsünün hangi aşamasında olduğunu belirlemektedir. İşlemler niteliğine ilişkin ilk alt boyutta bulunan ana terimler, bileşenin temel

işlevini tanımlamaktadır. İkinci alt boyutta bulunan yardımcı terimler ise, bileşenin değişik amaçlara yönelik farklı gerçekleştirmelerini tanımlamaya yardımcı olmaktadır (kabarcık-sıralama, hızlı-sıralama, vb.). İşlenenler niteliği, bileşenin üzerinde işlem yaptığı nesneler ile ilgilidir. Bir bileşen, genelde birtakım veri türleri ve veri yapıları üzerinde işlem yapmaktadır. Tamsayı veya karakter veri türleri, bağlaçlı listeler, yığıtlar, ağaçlar, kütükler, tablolar bu veri türleri ve yapılarıdır. Bir bileşen ayrıca, diğer bileşenler üzerinde de işlem yapabilmektedir. Bu nedenle, işlenenler niteliğini oluşturan terimler ya veri türleri veya veri yapıları ile ilgili terimler ya da soyutlamalar niteliğindeki terimlerden oluşmaktadır. Bağımlılıklar niteliği, bileşenin geliştirilme ortamını tanımlayan terimlerden oluşmaktadır. Bu terimler, bileşenin gerçekleştirildiği programlama dili veya işletim sistemi isimlerinden oluşmaktadır. Microsoft-C++, AT&T-C++, Dos, Unix, Vms bu tür terimlere ilişkin örneklerdir.

3.2.2 Bileşen geri elde etme araçları

Bu bölümde, boyutlara ayırmaya dayanan sınıflandırma yaklaşımı kullanan bir geri elde etme aracı tanıtılmaktadır.

Bileşenler boyutlara ayırmaya dayanan sınıflandırma yaklaşımı ile sınıflandırıldıktan sonra, her bileşen için değişik niteliklere ilişkin boyutlardan seçilen terimlerden (anahtar kelimeler) oluşan bir bileşen tanımlayıcısı (bileşen anahtarı) oluşturulmaktadır. Oluşan bileşen tanımlayıcısı sadece sözkonusu bileşenindir (tektir). Bileşen tanımlayıcıları değişik niteliklere ilişkin boyutların terimlerinden oluştuğu için bir ilişkisel veri tabanı çoklusu (relational tuple) biçimindedirler. Bu nedenle, ilişkisel veri tabanları bileşen tanımlayıcılarının saklanması için uygun ortamlardan biri olarak görülmektedir. Prieto-Diaz (1987), bileşen tanımlayıcılarını ilişkisel veri tabanında saklamıştır. Prieto-Diaz'ın (1987) önerdiği bileşen tanımlayıcıları, sınıflandırma tasarıları alt bölümünde bahsedilen ve yazılım bileşenlerini tanımlamada kullanılan altı niteliğe ilişkin terimler ile bileşenin kendisinin saklandığı yeri gösteren bir işaretçiden oluşmaktadır. Bileşenler kütüklerde veya değişik veri tabanlarında (ilişkisel veya nesneye dayalı) saklanabilmektedir.

Tanımlayıcılarının ilişkisel veri tabanlarında saklandığı bileşenlere ulaşabilmek için kullanıcılar bir "istek tanımlayıcısı" oluşturmaktadır. İstek tanımlayıcısı, bileşen tanımlayıcısına

benzer bir yapıda olup, herbir boyuttan seçilen terimlerden oluşmaktadır. Oluşturulan istek tanımlayıcısı daha sonra Şekil 3.6'da görülen geri elde etme aracındaki arayüz tarafından, bileşen tanımlayıcılarının saklandığı veri tabanına ilişkin bir sorguya çevrilmektedir.

Örneğin bileşenlerin, nesneye dayalı yaklaşımla oluşturulduğu ve soyutlama, işlemler, işlenenler ve bağımlılıklar gibi dört nitelik ile tanımlandığı bir ortamda niteliklere ilişkin boyutların içerikleri aşağıdaki gibi olsun:

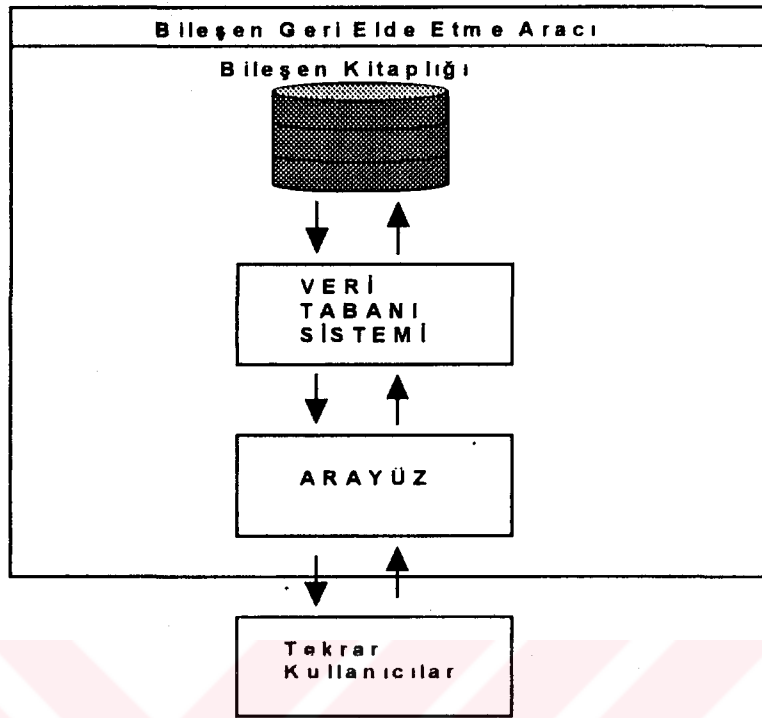
$$B_{\text{soyutlama}} = \{ \text{yığıt, kuyruk, ağaç, ...} \}$$

$$B_{\text{işlemler}} = \{ \text{ekle, çıkar, karşılaştır, ...} \}$$

$$B_{\text{işlenenler}} = \{ \text{tamsayı, karakter, ...} \}$$

$$B_{\text{bağımlılıklar}} = \{ \text{Unix, Dos, ...} \}$$

Bu durumda tekrar kullanıcı, Unix ortamında bir yığıtta tamsayı bir eleman ekleyen bir bileşeni aramak ve geri elde edebilmek için $\langle \text{yığıt, ekle, karakter, Unix} \rangle$ şeklinde bir istek tanımlayıcısı oluşturulacaktır. Bileşen tanımlayıcılarının SQL sorgu dili kullanan bir ilişkisel veri tabanında ve bileşen_tablosu adında ilişkisel bir tabloda saklandığı düşünülürse, yukarıdaki istek tanımlayıcısı " SELECT işaretçi_sahası FROM bileşen_tablosu WHERE soyutlama=yığıt, işlemler=ekle, işlenenler=karakter, bağımlılıklar=Unix " şeklinde bir SQL sorgusuna dönüştürülmektedir. Bu sorgunun işletimi sonucunda elde edilen işaretçi sahası veya sahalarının gösterdiği bileşenler tekrar kullanıcılara Şekil 3.6' da görülen arayüz yoluyla listelenmektedir.



Şekil 3.6 : Bileşen Geri Elde Etme Aracı

Her istek tanımlayıcısına karşılık gelen sorgunun işletimi sonucunda istenilen özellikte bileşen bulunamayabilmektedir. Bu durumda sorgu genişletme (query expansion) işleminin gerçekleştirilmesi gerekmektedir (Prieto-Diaz and Freeman, 1987). Sorgu genişletme işlemi, kullanıcıların belirlediği istek tanımlayıcısı içindeki bazı terimleri kavramsal olarak yakın terimlerle değiştirmektir. Sorgu genişletme işlemi ile benzer özelliklere sahip bileşenlerin bulunması sağlanmış olmaktadır. Kavramsal olarak yakın terimler, her niteliğe ilişkin boyutun terim uzayı için ayrı olarak tutulan kavramsal yakınlık çizgelerinden elde edilmektedir. Kavramsal yakınlık çizgeleri sınıflandırma işlemleri sırasında hazırlanmalıdır. Bir "i" niteliğine ilişkin boyut için hazırlanmış kavramsal yakınlık çizgesi Şekil 3.7'de görülmektedir. Çizgedeki düğümler T_{ij} terimlerine karşılık gelmektedir. Çizgede görüldüğü gibi her terim arasında belirli ağırlıklar vardır ve bu ağırlıklar kavramsal yakınlığı ifade etmektedir. Çizge üzerinde bir terime en az uzaklıktaki terim, o terime kavramsal olarak en yakın olan terimdir. $\langle T_{i1}, T_{j5}, T_{k3} \rangle$ gibi bir istek tanımlayıcısına karşılık gelen bir bileşen bulunmadığı durumda, üç boyuttan herhangi birine ilişkin kavramsal yakınlık çizgesi kullanılarak o boyuta ilişkin terim değiştirilebilmektedir.

BÖLÜM-4 BİLEŞEN GERİ ELDE ETME ARAÇLARINDA ÇOKLU-NİTELİKLİ KÜTÜK YAPILARININ KULLANIMI

Tekrar kullanılabilir bileşenlerin saklanması ve daha sonra geri elde edilmesi konusunda daha önce yapılmış çalışmaların birçoğunda bileşenler ve tanımlayıcıları ilişkisel veya nesneye dayalı veri tabanı sistemlerinde saklanmaktadır. Fakat veri tabanı sistemlerinin alt düzeyde dayandığı kütük sistemlerinden bahsedilmemektedir. Anahtar kelime yöntemine dayalı sınıflandırma tasarıları bileşenleri tanımlamak için birden fazla nitelik kullandığı ve her niteliğe eşit önem verdiği için çoklu-nitelikli (multi-attribute) kütük yapılarının anahtar kelime yöntemine dayanan bileşen geri elde etme araçlarında kullanılması uygun görünmektedir. Bilinen ve çok sık kullanılan bir çoklu-nitelikli kütük yapısı olan grid (ızgara) kütük yapısı bu amaçla kullanılmaktadır.

Çoklu-nitelikli kütükler ve grid kütük yapısı hakkında ayrıntılı bilgi bölüm 4.1'de verilmektedir. Grid kütük yapısının bir anahtar kelime yöntemi olan boyutlara ayırmaya dayanan sınıflandırma tasarıları ile sınıflandırılmış bileşenlere erişimde kullanımı bölüm 4.2'de, grid kütük yapısının bileşen geri elde etme araçlarında kullanıldığı zamanki performansını bulmak için yapılan çalışmalar ise bölüm 4.3'de anlatılmaktadır. Bölüm 4.4'de, grid kütük yapısı kullanımına dayanan bir bileşen geri elde etme aracının gerçekleştirim ayrıntıları verilmektedir.

4.1 Çoklu-nitelikli Kütükler ve Grid Kütük Yapısı

Tek-nitelikli bir kütük yapısı, bir doğrusal eksen üzerinde sıralı bir şekilde bulunan anahtarlardan oluşan bir-elemanlı çokluların (one-tuples) kümesi olarak tanımlanmaktadır. Tek-nitelikli bir kütük yapısı şekil 4.1'de görülmektedir.



Şekil 4.1 : Tek-Nitelikli Kütük Yapısı

Kütük yapılarında dizinler (directories), hızlı erişimi sağlamak ve çokluları disk sayfalarına etkin bir şekilde yerleştirmek görevine sahiptir. İndisleme (Indexing), dönüştürme ile ulaşım (hashing), ikili ağaçlar ile ulaşma gibi yöntemler tek-nitelikli kütük yapıları için dizin oluşturma işlemlerinde kullanılmaktadır.

Çoklu-nitelikli kütük yapısı, her boyutun bir niteliğe karşılık geldiği k boyutlu bir uzay olarak tanımlanmaktadır. Çoklu-nitelikli kütük yapılarında saklanan kayıtlara herhangi bir nitelikten veya belirli niteliklerin birleşiminden oluşan anahtarlar ile erişilebilmektedir. Daha biçimsel bir şekilde söylenecek olursa, n_i nitelikleri temsil etmek üzere $R=\{n_1, n_2, \dots, n_k\}$ şeklinde tanımlanan R kayıtlarından oluşan bir F kütüğüne, n_i değerlerinin çeşitli biçimlerde birleştirilmesi ile erişilebilmektedir. Örneğin isim, soyad, doğum tarihi ve numara gibi dört nitelikten oluşan kayıtların saklandığı bir kütüğe aşağıdaki erişimler mümkündür:

a)Soyad niteliği "Güneş" olan tüm kayıtların listelenmesi tek niteliği (birinci nitelik) kullanarak erişimdir.

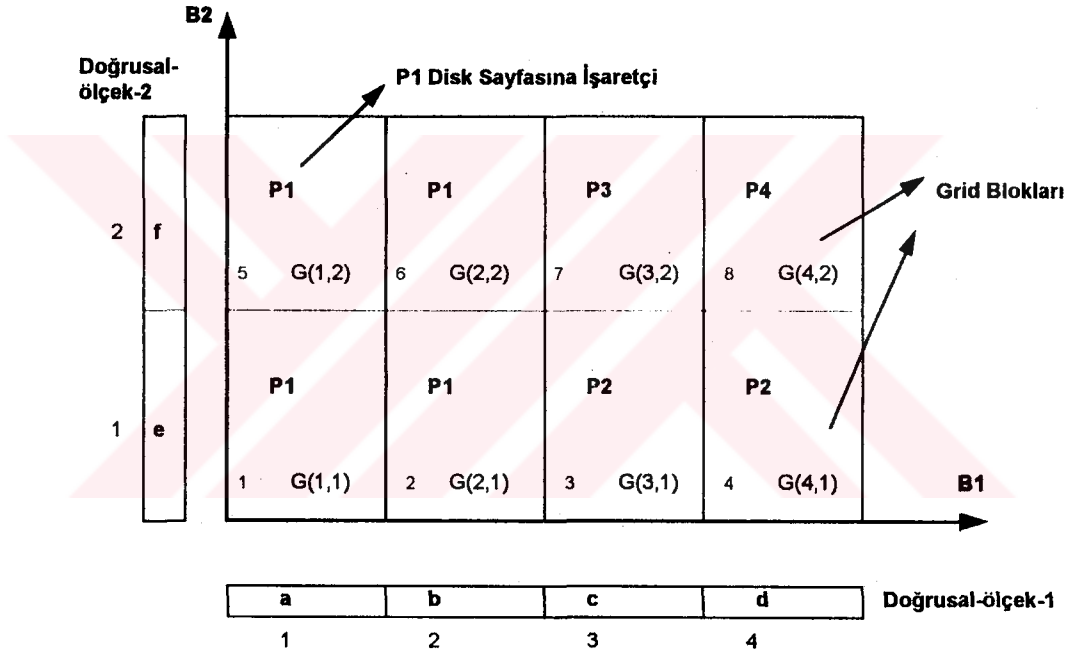
b)Doğum tarihi "1979" olan tüm kayıtların listelenmesi tek niteliği (üçüncü nitelik) kullanarak erişimdir.

c)İsim niteliği "Ali", soyad niteliği "Güneş" ve doğum tarihi niteliği "1975" olan tüm kayıtların listelenmesi 3 nitelik(birinci, ikinci ve üçüncü) kullanarak erişimdir.

d)İsim niteliği "Ali", soyad niteliği "Güneş", doğum tarihi niteliği "1975" ve numarası niteliği "3232" olan kayıtların listelenmesi tüm nitelikleri kullanarak erişimdir.

Grid kütük yapısı da, kayıtların k boyutlu bir uzayda noktalar olarak gösterildiği çok nitelikli kütük yapısıdır (Nievergelt and Hinterberger, 1984). Söz konusu k boyutlu uzay, grid blokları denilen birimlere ayrılmıştır. Grid blokları, $k=2$ boyut olma durumunda kare veya dikdörtgen, $k=3$ boyut olma durumunda küp veya dikdörtgenler prizması olmaktadır. Grid bloklarının herbirisi veya belli sayıda grid bloğundan oluşan bir grup fiziksel disk sayfalarına karşılık gelmektedir. Bir grid kütük yapısı, grid dizini (grid directory) ve belirli bir bölünme politikasına göre bölünmüş k -

boyutlu bir uzaydan meydana gelmektedir. Grid dizininin görevi, grid bloklarının fiziksel disk sayfalarına atanmasını sağlamaktır. Grid dizini iki kısımdan oluşmaktadır. Bu kısımlara grid dizisi (grid array) ve doğrusal ölçekler (linear scales) adı verilmektedir. Grid dizisi k boyutlu bir dizi olup dinamik bir yapıdır. Grid dizisinin her elemanı, bir grid bloğuna karşılık gelmektedir ve fiziksel disk sayfalarına olan işaretçileri tutmaktadır. Doğrusal ölçekler, tek boyutlu dizilerdir. Her ölçek, bir nitelik kümesine karşılık gelmektedir. Grid dizisi çok miktarda veri içerebileceği için diskte, doğrusal ölçekler ise ana bellekte tutulmaktadır. Şekil 4.2'de iki boyutlu grid kütük yapısı görülmektedir.



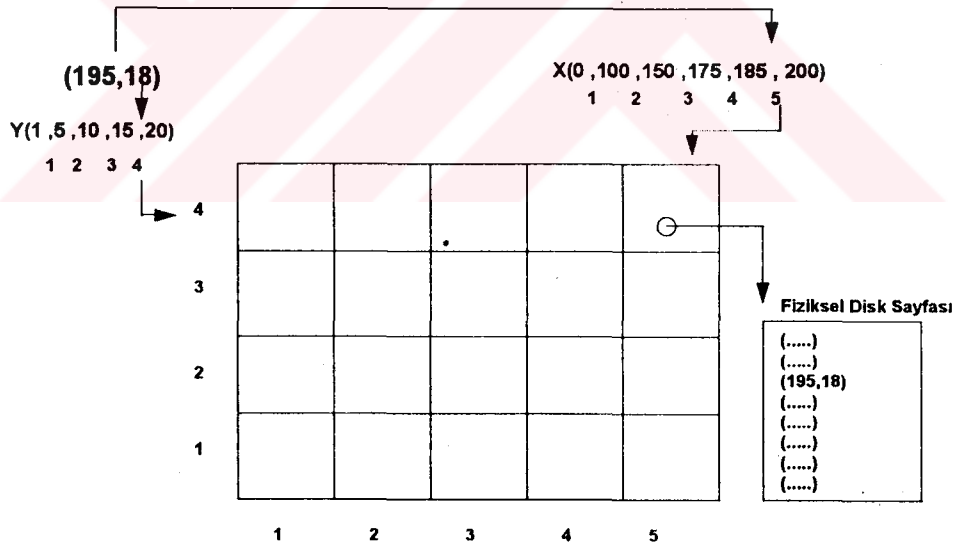
Şekil 4.2 : İki Boyutlu Grid Kütük Yapısı

Şekil 4.2'de görüldüğü gibi grid dizini iki boyutlu olup, 8 elemana sahiptir. Grid blokları da 8 tane olup, 4 tanesi 1 nolu disk sayfasına, 2 tanesi 2 nolu disk sayfasına, 1 tanesi 3 nolu disk sayfasına ve 1 tanesi de 4 nolu disk sayfasına karşılık gelmektedir. Grid dizisi $G(1..4, 1..2)$ ve doğrusal ölçekler $X(1..4)$ ve $Y(1..2)$ olarak tanımlanmaktadır.

4.1.1 Grid kütük yapısında kayıtlara erişim

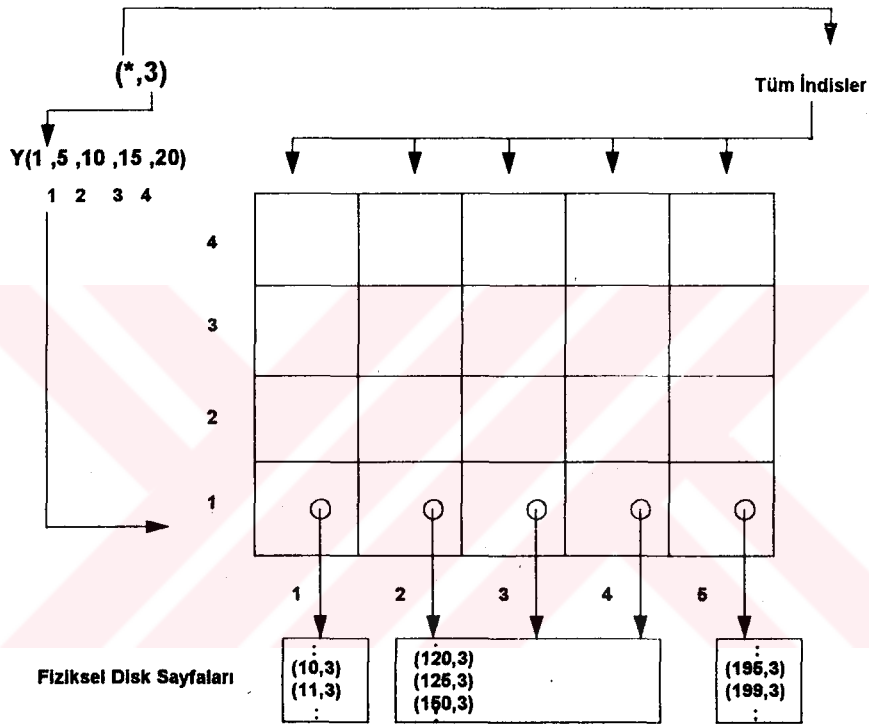
Grid kütük yapısında kayıtlara erişim iki disk erişimi gerektirmektedir. Bunlardan birincisi, Grid dizisine erişim, ikincisi de grid dizininin gösterdiği fiziksel disk sayfasına erişimdir. Doğrusal ölçekler grid dizisine erişimde kullanılmaktadır. Aşağıda iki boyutlu grid kütük yapısına erişimi gösteren bir örnek verilmektedir:

X(0,100,150,175,185,200) ve Y(1,5,10,15,20) doğrusal ölçekleri gösterdiği bir durumda "X niteliği 195 ve Y niteliği 18 olan kayıtları listele" şeklinde bir istek sonucunda ilk önce 195 değerinin X doğrusal karşılık geldiği indis bulunmaktadır. Bu indis 5 değerine karşılık gelmektedir. 18 değerinin Y doğrusal ölçeği üzerindeki indis değeri ise 4' dür. Bu indis değerleri ile grid dizinine erişilmektedir. G(5,4)'ün gösterdiği disk sayfasına ulaşarak, herhangi bir arama yöntemi ile disk sayfası arandığında (195,18) kaydı bulunmaktadır. Şekil 4.3, yukarıda anlatılan erişimi göstermektedir.



Şekil 4.3 : Grid Kütük Yapısında Kayıtlara Erişim

Grid kütük yapısı tüm nitelikleri belirlenmemiş sorgulara karşılık gelen kayıtlara da aynı şekilde erişmektedir. Bu durumda daha fazla disk sayfası aranmak zorundadır çünkü belirlenmeyen nitelikler için tek bir indis bulunamayacağı için, o niteliğe ait doğrusal ölçekteki tüm indisler erişimde kullanılmaktadır. Bu tip bir arama Şekil 4.4'de görülmektedir. Belirlenmeyen nitelik "*" ile gösterilmektedir.



Şekil 4.4 : Kısmi Belirli Sorgular İçin Grid Kütük Yapısında Kayıtlara Erişim

Bu tip bir erişim için n ve m tamsayılar olmak üzere, n adet fiziksel disk sayfası aranmakta ve m adet kayıt bulunmaktadır. Şekil 4.4'deki erişim için $n=3$ ve $m=7$ değerlerine sahiptir.

4.1.2 Grid kütük yapısı oluşturmak için bir algoritma

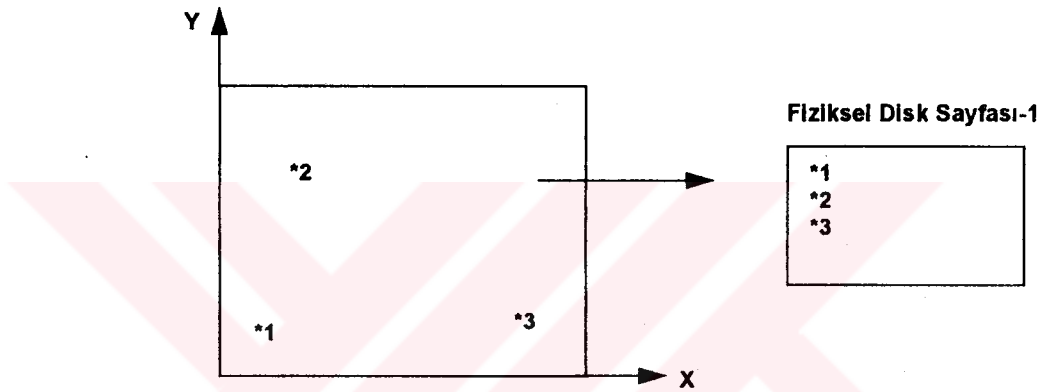
Aşağıda n boyutlu grid kütük yapısı oluşturmak için bir algoritma verilmektedir. Bu algoritmanın 2, 3 ve 5 boyut için gerçekleştirmeleri yapılmış olup, ekler bölümünde bulunmaktadır. 2 ve 3 boyutlu gerçekleştirmeler, grid kütük yapısının verimini ölçmede, 5 boyutlu gerçekleştirim ise örnek bir bileşen geri elde etme aracı gerçekleştiriminde kullanılmıştır.

Grid kütük yapısı oluşturma algoritmalarının temelinde grid bloklarının bölünmesi işlemi vardır. Bu nedenle grid kütük yapısı oluşturma algoritması verilmeden önce grid bloklarının bölünmesi için kullanılan yöntem açıklanmaktadır.

4.1.2.1 Grid bloklarının bölünmesi için bir yöntem

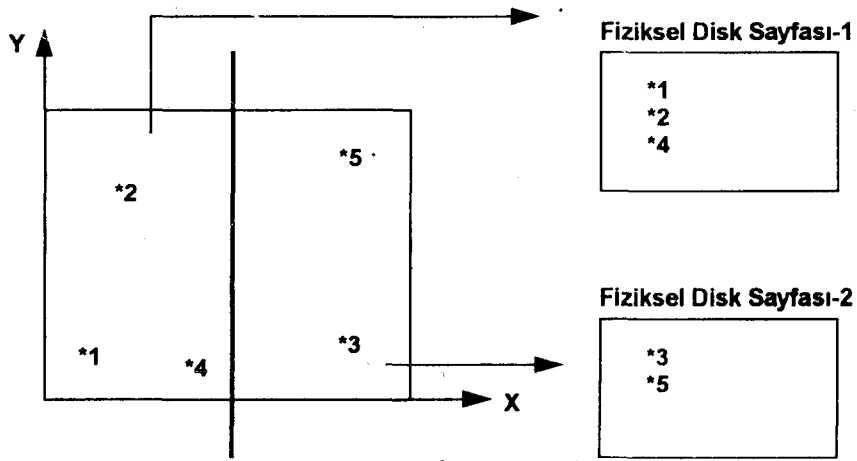
Grid bloklarının bölünmesi için çeşitli yöntemler geliştirilmiştir. Kullanılacak olan yöntem grid kütük yapısını gerçekleştirecek kişilerin seçimine bağlı olmaktadır. Grid bloklarının her defasında ayrı bir eksenenden bölünmesi ve bölünme ekseninin dairesel bir sıraya göre belirlenmesi en çok kullanılan yöntemlerden birisidir. Bu yöntemde n boyutlu bir grid kütük yapısı için ilk bölünme gerektiğinde 1 nolu eksenenden, ikinci bölünme gerektiğinde 2 nolu eksenenden, n nolu bölünme gerektiğinde n nolu eksenenden, $n+1$ nolu bölünme gerektiğinde yine 1 nolu eksenenden bölünmektedir. Aşağıda iki boyutlu bir grid kütük yapısının bu yöntemle bölünmesi adım adım gösterilmektedir. Örnekte fiziksel disk sayfasının kapasitesinin 3 olduğu kabul edilmektedir. Şekil 4.5, grid kütük yapısında üç kayıt bulunduğu (1, 2 ve 3 nolu kayıtlar) ve bu kayıtların 1 nolu fiziksel disk sayfasında saklandığı durumu göstermektedir. 1 nolu fiziksel disk sayfası alabileceği en fazla kayıt sayısı olan 3 kayıt içermektedir. Grid kütük yapısına bu andan başlayarak eklenmek isteyen kayıtlar, 1 nolu fiziksel disk sayfasında taşmaya neden olmakta ve grid kütük yapısının bölünmesini gerektirmektedir. Şekil 4.6, grid kütük yapısına iki kayıt (4 ve 5 nolu kayıtlar) daha geldikten sonraki durumu göstermektedir. 4 ve 5 nolu kayıtlar 1 nolu fiziksel disk sayfasını taşıracak için 2 nolu fiziksel disk sayfasının kullanıma sokulması ve grid bloğunun bölünmesi gerekmektedir. Bölünme işlemi ilk defa gerçekleşeceği için, X eksenı bölünmelidir. Bölünme işlemi sonucunda daha önce bir olan grid blokları sayısı ikiye çıkmaktadır. Bir başka deyişle grid dizisi $G(x,y)$ günlenmektedir. Grid bloğu bölünmesi sonucunda yapılması gereken diğer bir işlem ise yeni kullanıma sokulan ve önceden kullanımda olan fiziksel disk sayfaları arasında bazı

kayıtların aktarılmasıdır. Bunun nedeni, grid dizisi günlendiği için, grid bloklarının işaret ettiği fiziksel disk sayfalarının da değişmesidir. Şekil 4.6'da görüldüğü gibi bölünme işlemi sonucunda oluşan iki grid bloğundan soldaki 1 nolu fiziksel disk sayfasını, sağdaki de 2 nolu fiziksel disk sayfasını işaret etmektedir. Bunun sonucunda, 1 nolu fiziksel disk sayfasının 1,2 ve 4 nolu kayıtları, 2 nolu fiziksel disk sayfasının da 3 ve 5 nolu kayıtları tutması gerekmektedir. Bir önceki adım olan Şekil 4.5'e bakılacak olursa 3 nolu kaydın 1 nolu fiziksel disk sayfasında olduğu görülmektedir. Bu nedenle 3 nolu kayıt, 2 nolu fiziksel disk sayfasına aktarılmaktadır.

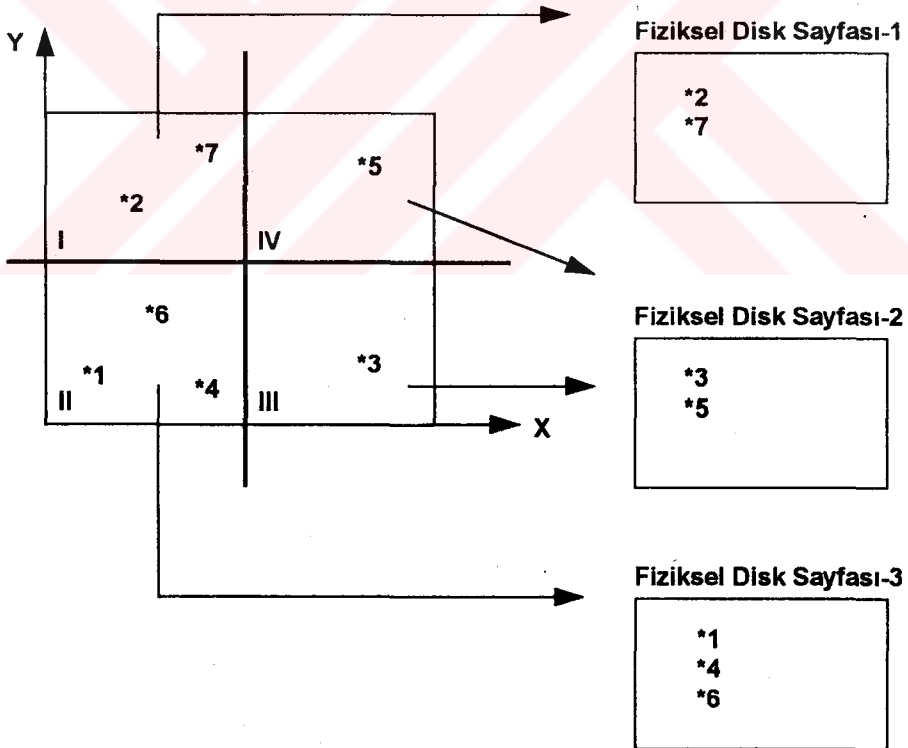


Şekil 4.5 : Grid Bloğu Bölünmesi - Adım1

Grid kütük yapısına 6 ve 7 nolu kayıtların geldiği düşünülürse, 1 nolu disk sayfasında tekrar taşma meydana gelmektedir. Böylece 3 numaralı fiziksel disk sayfası kullanıma sokulmaktadır. Bu durum Şekil 4.7'de görülmektedir.



Şekil 4.6 : Grid Bloğu Bölünmesi - Adım2



Şekil 4.7 : Grid Bloğu Bölünmesi - Adım3

İkinci defa bölünme işlemi gerçekleşeceği için y eksenı bölünmekte ve dört grid bloęu oluşmaktadır. Şekil 4.6'da sağ tarafta yer alan grid bloęu Şekil 4.7'de III ve IV nolu bloklar haline gelmektedir. Taşma söz konusu olmadığı için III ve IV nolu bloklar yine 2 nolu fiziksel disk sayfası ile eşlenmektedir. I ve II nolu grid blokları ise bölünmeye neden olan kayıtları içermektedir. I nolu grid bloęu I nolu disk sayfasına, II nolu grid bloęu ise 3 nolu fiziksel disk sayfasına eşlenmektedir. Bunun sonucunda, I nolu fiziksel disk sayfasında bulunan I ve 4 nolu kayıtlar 3 nolu fiziksel disk sayfasına aktarılmaktadır. Diğer kayıtlar için de bölünme aynı mantıkla devam etmektedir.

4.1.2.2 Algoritma

Yordam Grid-Oluştur

Adım-1:

Grid-bloklarının, fiziksel disk sayfaları benzetim yapılarının, doğrusal ölçeklerin ilklenmesi.

Adım-2:

Tüm doğrusal ölçekler üzerinde birincisi 0 ikincisi 2^n ($n \in \mathbb{Z}$) olmak üzere iki eleman yerleştirilmesi. Böylece tüm grid kütük yapısı başlangıçta üzerinde 0 ve 2^n elemanları olan doğrusal ölçeklere sahip tek bir blok olarak görölmektedir.

Adım-3:

Kayıt okunur

Adım-4:

Kayıtta bulunan her nitelik için doğrusal eksen üzerinde karşılık geldięi indislerin (indis-1, indis-2,.....indis-n) bulunması ve bu indisler ile önce grid dizini G'ye daha sonra onun işaret ettięi fiziksel disk sayfasına erişilmesi.

Adım-5:

Eğer erişilen disk sayfasında yer varsa kayıt bu disk sayfasına yerleştirilir.

Eğer erişilen disk sayfasında yer yoksa Adım-5.1 izlenir.

Adım5-1:

Bölünecek grid bloğunun grid dizisindeki yeri bulunur.

Bölünme yöntemi gözönüne alınarak bölünme eksenini bulunur.

Grid-Böl fonksiyonunu çağırılır.

Grid-Böl fonksiyonunun çağırılmasından sonra bazı disk sayfalarında o anda bile taşma varsa hiçbir disk sayfasında taşma olmayana dek Grid-Böl fonksiyonu çağırılır.

Fonksiyon Grid-Böl**Adım-1:**

Bölünecek olan grid bloğunun bölünme eksenini üzerindeki başlangıç ve bitim noktaları bulunur. Bu değerlerin farkı ikiye bölünür, sonuç başlangıç noktası değerine eklenerek bölünme noktası bulunur.

Adım-2:

Bölünen boyut üzerindeki doğrusal ölçekler üzerindeki elemanlar günlenir. Bu işlem bölünecek noktadan sonraki elemanlar bir ileri ötelenerek gerçekleştirilir.

Adım-3:

Grid dizisi G'nin tüm elemanlarının günlenmesi bir döngü içinde yapılmaktadır. Bu adımda döngü değişkenlerine boyut sayısı gözönüne alınarak değerler atanır. Bundan sonraki adımlar boyut sayısı kadar döngü içinde gerçekleşmektedir.

Adım-4:

Döngü içinde işlenen grid bloğunun bölünme eksenini üzerindeki bitim noktası, bölünen grid bloğunun sınırlarından küçükse işlenen grid bloğu üzerinde işlem yapılmaz. Grid dizisi G'nin günlenmesine gerek yoktur.

Döngü içinde işlenen grid bloğunun bölünme eksenini üzerindeki başlangıç noktası, bölünen grid bloğunun sınırlarından büyükse işlenen grid bloğu bir ötelenmekte ve grid dizisi güncellenmektedir.

Döngü içinde işlenen grid bloğu ile bölünen grid bloğu aynı sınırlara sahip ise adım4-1 izlenmektedir.

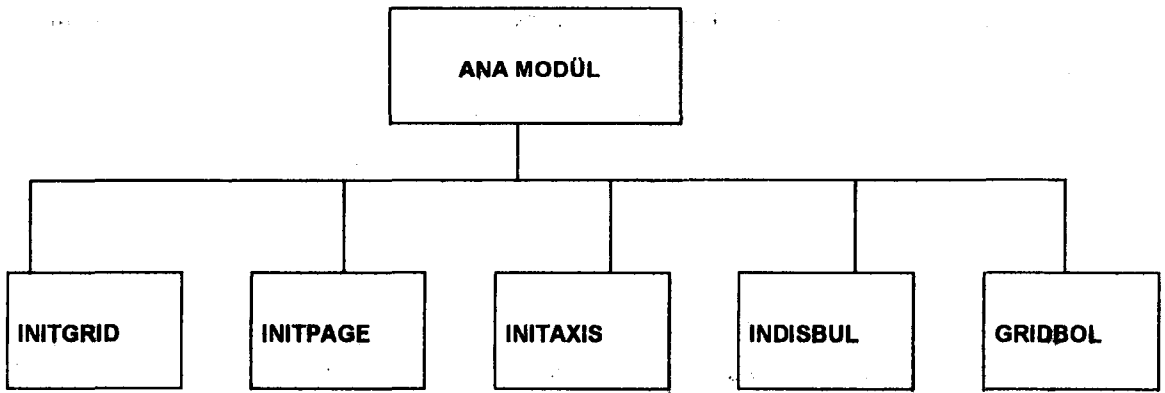
Adım4-1:

Döngü içinde işlenen grid bloğu taşımaya neden olan fiziksel disk sayfasına karşılık geliyorsa grid bloğu bölünmekte, bölünme sonucunda oluşan yeni grid bloğuna yeni bir fiziksel disk sayfası atanmakta ve kayıtların taşınan ve yeni kullanıma sokulan fiziksel disk sayfaları arasında paylaşılması işlemi yapılmaktadır.

Döngü içinde işlenen grid bloğu taşımaya neden olan fiziksel disk sayfasına karşılık gelmiyorsa, grid bloğu bölünmekte ve yeni oluşan grid bloğuna da aynı fiziksel disk sayfası atanmaktadır.

4.1.2.3 İki ve üç boyutlu grid kütük yapısı oluşturmak için yazılan programlar

Bu tür grid kütük yapıları oluşturmak için yazılan programlar, bir ana modül ve bu modül içinden çağrılan beş fonksiyondan oluşmaktadır. Bu fonksiyonlar; initgrid, initpage, initaxis, indisbul ve gridbol isimlerine sahiptir. Initgrid, grid bloklarını ilkleyen fonksiyondur. Initpage, fiziksel disk benzetim sayfalarını, initaxis ise doğrusal ölçekleri ilklemektedir. Indisbul fonksiyonu doğrusal ölçekler üzerinde aralık numarası bulmak için kullanılmaktadır. Gridbol, x-y eksenlerinin sırayla bölündüğü politikaya göre grid kütük yapısını bölmektedir. Ana modül bölüm 4.1.2.2'de verilen grid kütük yapısı oluşturma algoritmasına sahiptir ve bu algoritma çerçevesinde yukarıdaki fonksiyonları kullanmaktadır. Ana modül ve fonksiyonların ilişkileri Şekil 4.8'de görülmektedir. Bu programlara ilişkin kaynak kodlar ek-1 ve ek-2 bölümlerinde bulunmaktadır.

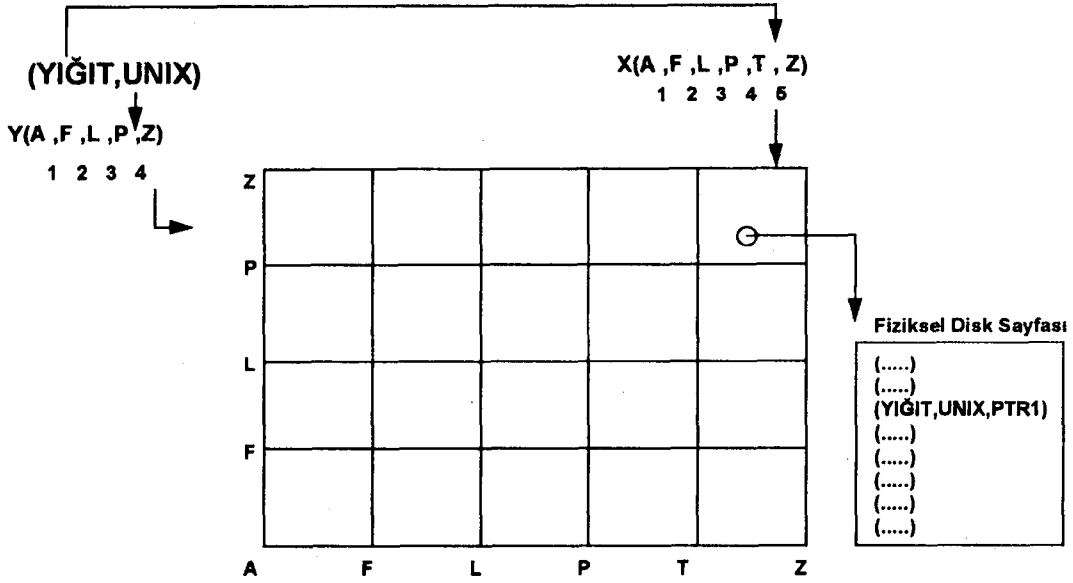


Şekil 4.8 : 2 ve 3 Boyutlu Grid Kütük Yapıları Oluşturan Programlarda Modüller

4.2 Grid Kütük Yapısı ve Boyutlara Ayırmaya Dayanan Sınıflandırma İlişkisi

Boyutlara ayırmaya dayanan sınıflandırma yaklaşımı ile sınıflanan yazılım bileşenlerini tanımlamada kullanılan her bir nitelik grid kütük yapısındaki bir boyuta karşılık gelmektedir. Belli harfler grid dizinine ilişkin doğrusal ölçekler üzerinde noktalar olarak tanımlanabilmektedir. Bileşen tanımlayıcılarındaki her bir terimin baş harfi doğrusal ölçek üzerindeki noktalarla karşılaştırılarak doğrusal ölçekler üzerindeki indis değerleri bulunmaktadır. Bu indis değerleri ile grid dizisine ve oradan da bileşen tanımlayıcılarının olduğu fiziksel disk sayfalarına erişilebilmektedir.

İki boyutlu grid kütük yapısında saklanan ve boyutlara ayırmaya dayanan sınıflandırma tasarısına göre oluşturulmuş bileşen tanımlayıcılarına erişim Şekil 4.9'de görülmektedir.



Şekil 4.9 : Bileşen Tanımlayıcılarına Grid Kütük Yapısı ile Erişim

Yığıt kelimesinin baş harfi olan "Y" doğrusal ölçekler üzerinde 5 nolu indise, Unix kelimesinin baş harfi olan "U" ise doğrusal ölçek üzerinde 4 nolu indise karşılık gelmektedir. Bu indisler ile G(5,4) ve onun işaret ettiği disk sayfasına erişilmektedir. Bu sayfa içinde bulunan bileşen tanımlayıcılarının ilk iki niteliği gözönüne alınarak istek tanımlayıcısı < Yığıt, Unix > ile eşlenmektedir. Eşleme işlemi başarılı ise, bileşen tanımlayıcısı içinde yer alan ve Şekil 4.9'de ptr1 ile gösterilen işaretçi kullanılarak bileşenin içeriğine ulaşılabilmektedir. Tüm niteliklerin belirlenmediği istek tanımlayıcılar ile bileşenlere erişim de aynı mantıkla yapılmakta ve sonuçta genellikle birden fazla disk sayfası aranarak birden fazla bileşen geri elde edilmektedir. Bu tip erişimler bölüm 4.1'de ayrıntılı olarak anlatılmaktadır (bkz. Şekil 4.4).

4.3 Grid Kütük Yapısının Verimi

Grid kütük yapısının verimini ölçebilmek amacıyla bölüm 4.1'de verilen algoritmalar kullanılarak 2 ve 3 boyutlu grid kütük yapıları oluşturulmuştur. Oluşturulan grid kütük yapılarında çeşitli sayılarda bileşen tanımlayıcıları saklanarak, daha sonra rastgele üretilen istek

tanımlayıcıları ile bileşenleri geri elde etmek istendiğinde, erişilen fiziksel disk sayfalarının sayısı bulunmuştur. Elde edilen değerler, aynı istek tanımlayıcılarını kullanarak tek-nitelikli kütük yapıları ile bileşen geri elde edilmek istendiğinde erişilen fiziksel disk sayfaları sayısı ile karşılaştırılmıştır.

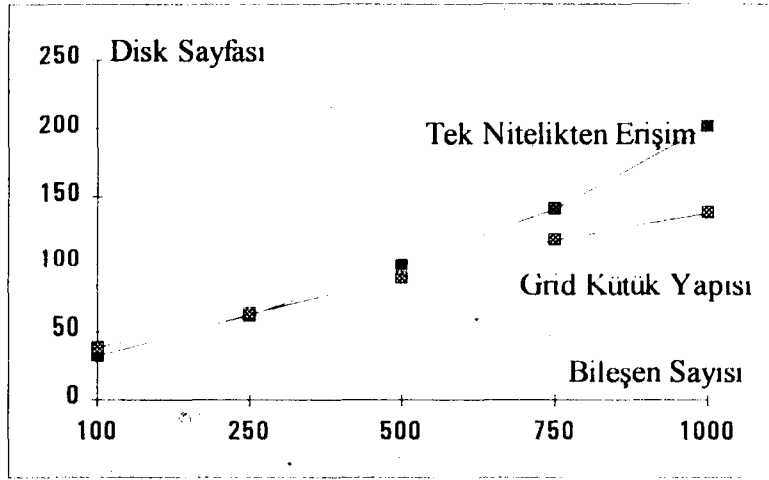
Bileşen tanımlayıcıları düzgün rastgele sayı üreticisi ile üretilmiştir. Bileşen tanımlayıcılarının ilk niteliği tek-nitelikten erişimde anahtar saha olarak alınmıştır. Grid kütük yapısı için belli bir niteliğin anahtar olarak tanımlanmasına gerek yoktur çünkü çoklu-nitelikli kütük yapılarında her nitelik veya belirli nitelikler anahtar olarak düşünülebilmektedir.

Her istek tanımlayıcısında kaç niteliğin belirleneceği, belirlenecek olan niteliklerin yerleri ve bu niteliklerin alacağı değerler düzgün rastgele sayı üreticisi ile üretilmiştir. Örneğin 3 boyutlu grid kütük yapısı için üretilen bir istek tanımlayıcısında 1. ve 3. nitelikler belirlenmiş olup, 2. nitelik genel bir karakter ile geçilebilmektedir. Hiçbir niteliğin belirlenmediği bir istek tanımlayıcısı türüne izin verilmemektedir.

Tek-nitelikli kütük yapısı kullanıldığında erişilen fiziksel sayfa sayısı aşağıdaki gibi hesaplanmaktadır:

Üretilen istek tanımlayıcısında ilk nitelik belirlenmiş ise indis yapısı kullanılarak tek bir disk sayfasına erişilmektedir. Eğer istek tanımlayıcısında ilk nitelik belirlenmemiş ise indisleme olamayacağı için tüm disk sayfaları sıradan aranmakta ve kullanımda olan tüm disk sayfalarına erişilmektedir. Örneğin, toplam 1000 bileşen tanımlayıcısı olduğu ve bir fiziksel disk sayfasının en fazla 50 bileşen tanımlayıcısı tutma kapasitesine sahip olduğu durumda ise bileşen tanımlayıcıları 20 fiziksel disk sayfasına yerleşmiş olmaktadır. Bu durumda ilk niteliğin belirlenmediği ve tek-nitelikli kütük yapısı ile yapılan geri elde etme işleminde, 20 fiziksel disk sayfasına da erişilmiş olmaktadır.

Şekil 4.10, 2 boyutlu grid kütük yapısında 25 istek tanımlayıcısının işletimi sonucunda elde edilen verimlilik sonuçlarını vermektedir.



Şekil 4.10 : 2 Boyutlu Grid Kütük Yapısı için Verimlilik Sonuçları

Değişik sayılardaki istek tanımlayıcılarının işletimi sonucu 2 ve 3 boyutlu grid kütük yapısına ilişkin ayrıntılı verimlilik sonuçları bölüm 6'da bulunmaktadır.

Şekil 4.10'da görüldüğü gibi, bileşen sayısı arttıkça, tek nitelikli ve grid kütük yapıları arasındaki erişilen disk sayfası sayısı farkı grid kütük yapısını destekleyici biçimde artmaktadır. Bu durum, bileşen kitaplığında genelde çok sayıda bileşen olduğu için önemli bir sonuçtur. Grid kütük yapısının diğer bir üstün yanı istek tanımlayıcılarının belirlenmesi işlemi sırasında ortaya çıkmaktadır. Grid kütük yapısı, istenilen nitelik veya nitelik kombinasyonu ile bileşenlere erişimi sağlamaktadır. Tek-nitelikten erişimde ise anahtar sahasını belirlemek zorunluluğu kısıtlayıcı bir etken olmaktadır.

4.4 Bileşen Tanımlayıcıların Grid Kütük Yapısında Saklandığı Bileşen Geri Elde Etme Aracı Gerçekleştirimi

Bu amaçla ilk önce temel veri yapıları ve algoritmalar konusunda tekrar kullanılabilir bileşenler içeren bir bileşen kitaplığı oluşturulmuştur. Bu kitaplık, E.Ü. Bilgisayar Mühendisliği bölümünde bir lisans tezi (Dindir ve Kurt, 1995) kapsamında ve nesneye dayalı yaklaşımla geliştirilmiştir. Kitaplık, yığıl işlemleri, kuyruk işlemleri, listeler ile ilgili işlemler, ağaç veri yapısı

üzerinde işlemler, sıralama algoritmaları, arama algoritmaları, matris işlemleri ve karmaşık sayı işlemleri yapan bileşenlerden oluşmaktadır. Kitaplıktaki bileşenler bölüm 3'de anlatılan ve Sorumgard (1993) tarafından önerilen yaklaşım temel alınarak sınıflandırılmıştır. Sınıflandırma işlemlerinde 5 nitelik kullanılmıştır. Bu nitelikler bileşen-türü (analiz, tasarım, kod, belgelemeler, sinama senaryoları), soyutlama, işlemler, işlenenler ve bağımlılıklardır. Bu niteliklere ilişkin boyutların içerikleri aşağıdaki gibidir:

Bileşen-türü

- 1-Analiz
- 2-Tasarım
- 3-Kod
- 4-Belgelemeler
- 5-Sinama Senaryoları

Soyutlama

- 1-Doğrusal Veri Yapıları
- 2-Doğrusal Olmayan Veri Yapıları
- 3-Genel Algoritmalar
- 4-Karmaşık Sayılar
- 5-Matris İşlemleri

İşlemler

- 1-Çarpma
- 2-Toplama
- 3-Çıkarma
- 4-Bölme
- 5-İkili-Arama
- 6-Sıradan-Arama
- 7-Kabarcık-Sıralama
- 8-Seçmeli-Sıralama
- 9-Karşılaştırma
- 10-Sıradan dolaşma
- 11-Sondan dolaşma

- 12-Baştan dolaşma
- 13-Eleman ekle
- 14-Eleman Çıkar
- 15-Eleman Değiştir
- 16-Eleman Elde Et
- 17-Rastgele sayı üret

İşlenenler

- 1-Ağaç
- 2-Yığıt
- 3-Liste
- 4-Tamsayılar
- 5-Karakterler

Bağımlılıklar

- 1-Sadece Dos
- 2-Sadece Unix
- 3-Unix-Dos

Bu çalışmada oluşturulan kitaplık toplam 29 kaynak kod, 29 belgeleme, 1 analiz, 1 tasarım ve 1 sinama senaryosu içeren toplam 61 bileşenden meydana gelmiştir ve gelişmeye açıktır. Yeni bileşenler eklendiğinde, bileşenlere ilişkin bileşen tanımlayıcıları grid kütük yapısına eklenmektedir.

Bileşen tanımlayıcıları, her niteliğe ilişkin terimin boyutta bulunduğu yeri gösteren indislerden ve bileşenin içeriğinin bulunduğu yeri gösteren işaretçiden meydana gelmektedir. Örneğin, Unix veya Dos ortamında bağlaçlı listeye tamsayı ekleyen, kaynak kod içeren ve bileşenin içeriğini gösteren işaretçi değeri 1 olan bir bileşeni tanımlamak için kullanılan bileşen tanımlayıcısı $\langle 3, 1, 13, 4, 3, 1 \rangle$ şeklindedir. Tüm bileşenlere ilişkin tanımlayıcılar yukarıdaki gibi oluşturulmuş, 5 boyutlu ve doğrusal ölçekleri 0 ile 32 arasında elemanlar içeren grid kütük yapısında saklanmıştır. 5 boyutlu grid kütük yapısı bölüm 4.1'de verilen algoritmaya göre oluşturulmuştur.

5 boyutlu grid kütük yapısı kullanan bileşen geri elde etme aracının gerçekleştirimi için yazılan program, temelde 2 ve 3 boyutlu grid kütük yapısı oluşturmak için yazılan programlar ile benzerdir. Aralarındaki fark, bu programdaki ana modülün daha ayrıntılı olmasındadır. Bu programdaki ana modül, 5 boyutlu grid kütük yapısını oluşturma yanında, kullanıcıların istek tanımlayıcılarını oluşturmalarını ve bu istek tanımlayıcılarına karşılık gelen bileşenlerin bulunması ve listelenmesini de sağlamaktadır. Kullanıcıların, istek tanımlayıcıları belirleyebilmeleri için tüm boyutlar içerdikleri terimlerle birlikte kullanıcının önüne gelmekte, kullanıcı seçtiği terime karşılık gelen indisi belirlemektedir. Herhangi bir boyuttan terim seçmek istemeyen kullanıcılar indis olarak 9999 değerini girmelidir. Kullanıcıların tüm niteliklere 9999 değeri vermeleri engellenmiştir. Boyutlardan terim seçmeyi sağlayan ekran görünümü şekil 4.11'deki gibidir.

BEŞİNCİ BOYUT İÇERİĞİ Seçmek İstedığınız Terimin Numarasını Giriniz... Terim Seçmek İstemiyorsanız 9999 Giriniz... 1-SADECE DOS 2-SADECE UNIX 3-DOS-VE-UNIX Seçiminiz?3

Şekil 4.11 : Boyutlardan Terim Seçilen Ekran Görünümü

İstek tanımlayıcı oluştuktan sonra kullanıcıya gösterilerek kullanıcıdan onay alınmakta ve işleme sokulmaktadır. Kullanıcıdan onay alınan ekran görünümü şekil 4.12'deki gibidir.

Oluşturulan Sorgu <kod, matris-işlemleri, toplama, tamsayılar, Unix-Dos> Sorguyu Onaylıyor musunuz?E/H Onaylamak için e veya E tuşuna basınız... E

Şekil 4.12 : İstek Tanımlayıcısı Onay Ekranı

İstek tanımlayıcı ile grid kutük yapısı içinde saklanan bir veya birden fazla bileşen tanımlayıcısı eşlenmekte ve bulunan bileşenlerin içerikleri kullanıcıya listelenmektedir. Bu aşamadan sonraki ekran görünümü şekil 4.13'deki gibidir. Kullanıcı istiyor ise ekranda gördüğü bileşenleri ismini kendisinin belirlediği bir kutükte saklamaktadır

1 Adet Bileşen Bulundu

Bileşenin İçeriği

void matris-topla()

```

{
int A[5][5], int B[5][5], int C[5][5];
int i;
for (i=0;i<5;i++)
for (j=0;j<5;j+4)
{
C[i][j]=A[i][j] + B[i][j];
}
}

```

Bu bileşeni kütüğe yazmak istiyor musunuz? E/H

E

Bileşeni yazmak istediğiniz kutük adını giriniz...

Sakla 1.Dat

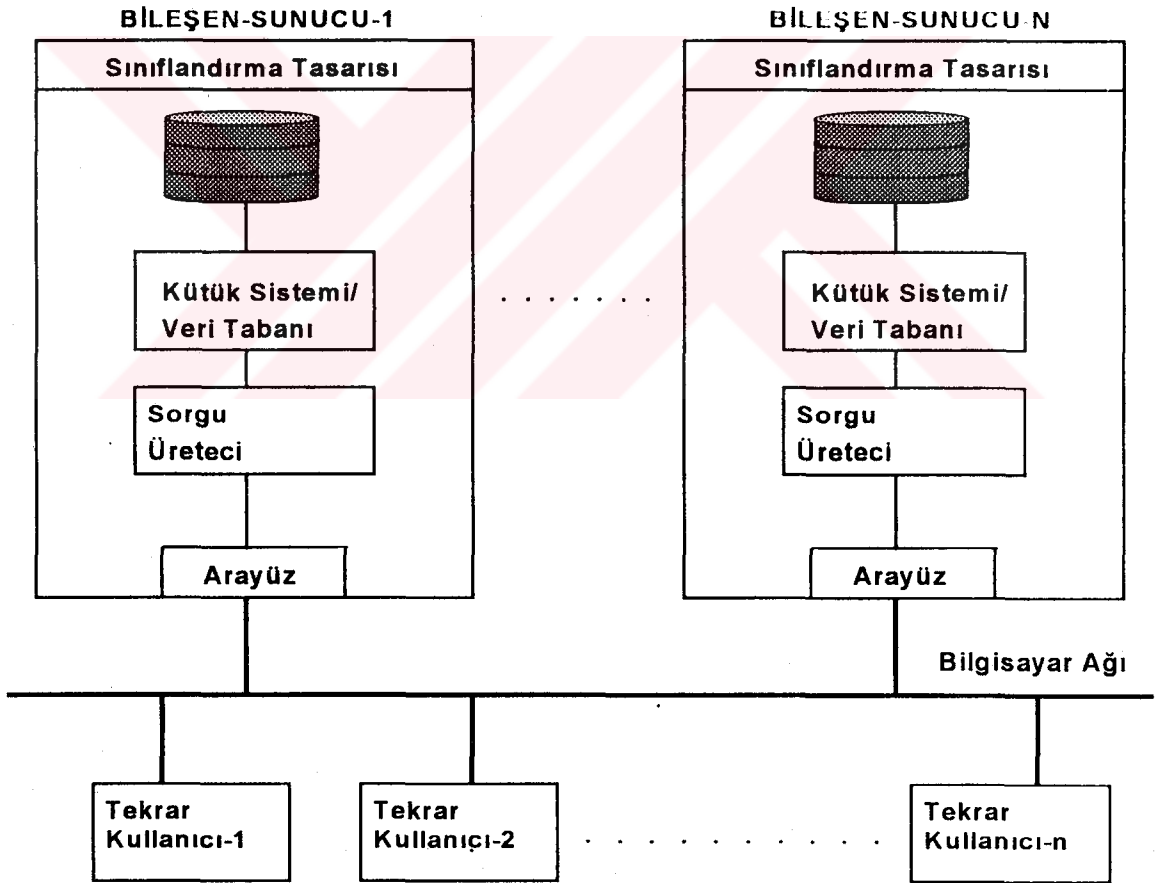
Şekil 4.13 : Bileşenlerin Listelenme Ekranı

5 boyutlu grid kütük yapısı kullanan bileşen geri elde etme aracının gerçekleştirimi için yazılan ve yukarıda işleyişi örneklenen programa ilişkin kaynak kodlar ek-3 bölümünde bulunmaktadır.



BÖLÜM-5 GENİŞ ÖLÇEKLİ TEKRAR KULLANIM

Bilgisayar ağları teknolojisindeki gelişmeler, BYYG yaklaşımına değişik boyutlar getirerek GTK(Geniş Ölçekli Tekrar Kullanım) kavramının (Tsichritzis, 1992) ortaya çıkmasına neden olmuştur. GTK, bilgisayar ağı üzerinde farklı yazılım ve donanım alt yapıları olan ortamlarda bulunan tekrar kullanıcıların, ağ üzerinde farklı yerlerde bulunan bileşen-sunuculardan seçtikleri tekrar kullanılabilir bileşenler ile yazılım geliştirmeleri olarak tanımlanmaktadır. Şekil 5.1'de görülen GTK ortamı, bilgisayar ağı, bileşen-sunucular ve kullanıcılardan meydana gelmektedir. Kullanıcılar tekrar kullanıma dayalı yazılım geliştiren kişilerdir ve yerel ortamlarında bileşen geri elde etme aracı dışındaki tekrar kullanımı destekleyici araçlar bulunmaktadır.



Şekil 5.1 : GTK Ortamı Genel Yapısı

Bir GTK ortamında bulunan bileşen-sunucular aşağıdaki bölümlerden oluşmaktadır:

a)Tekrar kullanılabilir bileşenlerin ve bu bileşenlerin tanımlayıcılarının belli bir sınıflandırma tasarısına dayanarak saklandığı veri tabanları veya kütük sistemleri.

b)Kullanıcılardan gelen istek tanımlayıcılarının sorgulara dönüştürülerek işletildiği bir sorgu üretici.

c)Ağ üzerindeki diğer birimlerle iletişimini ve ortak çalışmasını sağlayan arayüzler.

Bileşen-sunucuda temel olarak 3. bölümde anlatılan bileşen geri elde etme aracı ile temelde aynı bölümler bulunmaktadır. Tek farkı ağ üzerindeki farklı kullanıcılardan gelen isteklere yanıt vermesidir.

GTK ortamlarında çeşitli biçimlerde yazılım geliştirilebilmektedir. İlk yaklaşım, erişilen bir bileşen-sunucudan seçilen bileşenin içeriğinin ağ üzerinden kullanıcı ortamına gönderilmesidir. Kullanıcı elde ettiği bileşeni kendi ortamında istediği şekilde kullanabilmektedir. Örneğin, bileşen kaynak kod içeriyor ise, bileşeni derleyip işletilebilir kod elde etmek, bileşeni bileşen birleştirme aracı ile geliştirmekte olduğu yazılıma katmak veya bileşeni değiştirip yeni bileşen elde etmek olası olmaktadır.

İkinci yaklaşım, ağ üzerinde farklı bileşen-sunucularda bulunan değişik bileşenlerin birbiriyle birleştirilmesi yoluyla tekrar kullanıcı ortamlarında yazılımlar geliştirmektir. Birleştirilen bileşenler, matris toplaması yapan bir program kadar kısa veya bir veri tabanı yönetim sistemi kadar uzun olabilmektedirler. Bu yaklaşım aynı zamanda mega-programlama (Wiederhold, 1992) fikrini de desteklemektedir. Mega-programlama kavramından bölüm 5.1'de daha ayrıntılı olarak bahsedilmektedir.

Yukarıdaki iki yaklaşımın da gerçekleştirilebileceği GTK ortamları dağıtık bilgi işleme sistemleri şeklinde modellenmektedir. Bölüm 5.1'de, bir dağıtık bilgi işleme sistemi türü olan dağıtık nesne yönetim sistemleri incelenmekte ve bir dağıtık nesne yönetim sistemi standardı olan

CORBA (Common Object Request Broker Architecture- Ortak Nesne İstek Sunucu Mimarisi)'nın GTK ortamı modellemede kullanımı üzerinde durularak, GTK için bir yazılım mimarisi önerilmektedir.

GTK ortamları, veri tabanı birliklerine (federated database systems) benzer şekilde, tekrar kullanıcılar, bileşen-sunucular, sınıflandırma tasarımları, ortamdaki arayüzler gibi birimlerden oluşan birlikler topluluğu olarak düşünülebilmektedir. Her birlik, kendi içinde birtakım kurallar ve standartlar getirmektedir. Bölüm 5.2 bu tür bir birlikler topluluğu oluşturulması üzerinde durmaktadır.

Bölüm 5.3, GTK ortamlarındaki bir bileşen geri elde etme aracının gerçekleştirimine ayrılmıştır. Bu gerçekleştirim, CORBA standardına uygun bir ürün olan DEC-ACA ile yapılmıştır.

5.1 GTK Ortamlarının CORBA ile Modellenmesi

Dağıtık bilgi işleme, iki veya daha fazla sayıda yazılım parçasının birbiri ile iletişim kurarak, bilgi paylaşımı yoluyla içten-işletilebilirlik (interoperability) gerçekleştirmesidir. Söz konusu iki veya daha fazla yazılım parçası aynı bilgisayar üzerinde ya da bir bilgisayar ağı ile birbirine bağlı birden fazla bilgisayar üzerinde çalışmaktadır.

Dağıtık bilgi işleme sistemlerinin tasarımı ve oluşturulmasında geçilmesi gereken iki aşama içten-bağlılık (interconnectivity) ve içten-işletilebilirlik (interoperability) sağlanmasıdır (Manola, 1992). İçten-bağlılık, dağıtık sistem üzerindeki yazılım parçalarının birbiriyle ileti geçişleri yoluyla iletişimini sağlamaktır. İçten-bağlılık, genel RPC (Remote Procedure Call- Uzak Yordam Çağırma) yöntemleri ve istemci/sunucu (client/server) modeller ile sağlanabilmektedir. İstemci/sunucu modelde iki tür yazılım vardır. Belli bir bilgi veya servis isteyen yazılımlar istemci, istenen bilgi veya servisi sağlayan yazılımlar sunucu olarak adlandırılmaktadır.

İkinci aşama olan içten-işletilebilirlik ise iki veya daha fazla yazılım parçasının geliştirme dilindeki, işletim ortamındaki veya modellemedeki farklılıklara rağmen bir işin gerçekleştiriminde

işbirliği yapmasıdır. İçten-işletilebilirlik ile diğer ortamlardaki kaynaklar kullanılabilmekte ve karşılıklı bilgi alışverişi yapılabilir. (Konstantas, 1993). İçten-işletilebilirlik en alt düzeyde tür uyumluluğu sağlayan bir RPC yöntemi ile gerçekleştirilmektedir. Tür uyumluluğu, hem basit türler hem de kullanıcı tanımlı türler için gösterim farklılıklarını gizleme anlamına gelmektedir. Örneğin bir yığının gerçekleştirim ayrıntıları yığıtı paylaşan programlara yansımamalıdır.

Nesneye dayalı yöntemlerin dağıtık sistemlere uygulanması içten-işletilebilirliği sağlamak için daha iyi bir yaklaşımdır. Nesneler, bir dağıtık sistem için doğal bir model oluşturmaktadır. Bunun nedeni, dağıtık sistemlerin temel özelliklerinin nesneye dayalı yaklaşım ile standart olarak sağlanmasıdır. Bu temel özellikler aşağıda listelenmiştir.

a) Dağıtık sistemleri oluşturan birimler birbirleriyle belirli arayüzler ile daha kolay iletişim yapabilmektedir.

b) Dağıtık sistemdeki birimler belli işlevleri olan yerel yordamlara sahiptir.

c) Dağıtık sistemlerde birimler çeşitli soyutlamalar ile ayırd edilmektedir (Yazıcı servisi, dosya servisi, vb.).

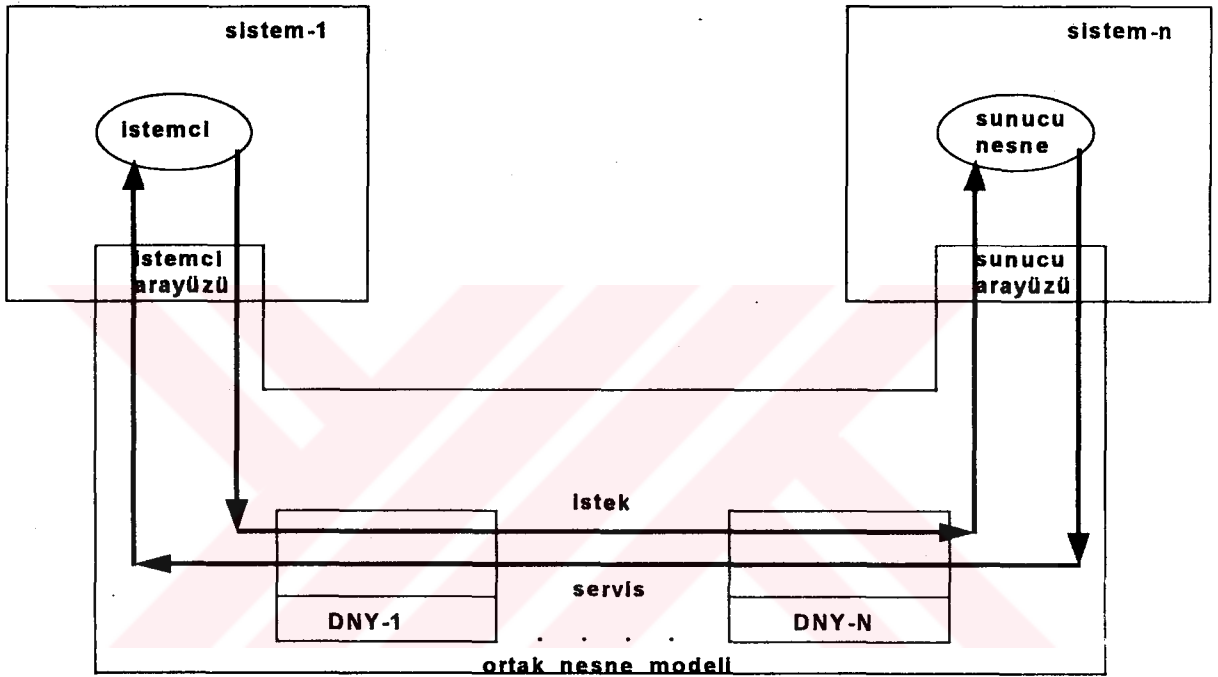
Yukarıda anlatılanlara ek olarak, dağıtık sistemlerde nesnelerin kullanılması çok farklılığı (heterojenlik) sağlar çünkü dağıtık nesnelere giden iletiler sadece o nesnenin arayüzüne bağlı olmaktadır, gerçekleştirim ayrıntıları ile ilgili değildir. Ayrıca herbir nesne, kendine ilişkin arayüzü değiştirmek yoluyla, sistemdeki başka hiçbir nesneyi etkilemeden değiştirilebilmektedir.

Nesneye dayalı yaklaşımla modellenmiş dağıtık sistemlere DNYS (Dağıtık Nesne Yönetim Sistemi) denmektedir. DNYS, birbiriyle etkileşen, ortak bir arayüz tanımlama standardına uygun arayüzleri olan nesneler topluluğu olarak tanımlanmaktadır (Manola, 1992).

DNYS belli sayıda dağıtık sunucu (node) ve istemcilerden oluşmaktadır. Her sunucu bir veya daha fazla uygulama programı veya nesne içermektedir. Bu sunucular sistemin bilgi işleme

kaynaklarını oluşturmaktadır. İstemciler bu kaynakları kullanarak bazı işlemler gerçekleştirmek isteyen birimlerdir.

Şekil 5.2'de genel bir DNYS mimarisi görülmektedir. Bu mimariyi oluşturan birimler aşağıda kısaca açıklanmaktadır.



Şekil 5.2 : DNYS (Dağıtık Nesne Yönetim Sistemi) Mimarisi

Sunucu Nesne: DNYS kaynakları olarak nitelendirilen nesnelerin gerçekleştirimleridir. Bu gerçekleştirimler, ortak bir arayüz standardına bağlı kalarak yapılmalıdır. Standartlara uymayan gerçekleştirimler yapılırsa, DNYS'nin, nesne uyarlayıcı (object adapter) adı verilen ve standartlara

uymayan gerçekleştirimleri standartlara uygun hale getiren araçları kullanması gerekmektedir. Şekil 5.2'deki mimaride nesne uyarlayıcılar gösterilmemiştir.

İstemci: Geleneksel yöntemlerle geliştirilmiş uygulamalar veya nesneye dayalı yaklaşımla geliştirilmiş nesneler olabilmektedir.

İstemci Arayüzü: İstemcilerin DNY (Dağıtık Nesne Yöneticisi)'den belli işlemlerin yapılması için istekte bulunmalarını ve sonuçları geri elde etmelerini sağlayan arayüzdür.

Sunucu Nesne Arayüzü: DNY'nin istenilen sunuculara ulaşmada kullandığı arayüzlerdir.

DNY (Dağıtık Nesne Yöneticisi): Bir DNYS içinde birden fazla sayıda olabilmektedir. DNY, sistemin tüm kaynaklarının-kaynaklar nesneye dayalı olsun veya olmasın- nesneler olarak görülmesini ve istemcilerin sistem kaynaklarından (sunucu nesneler) çeşitli isteklerde bulunabilmelerini sağlamaktadır. DNY, istemci isteklerinin karşılık geldiği nesne gerçekleştirimlerini bulup çalıştırır ve sonuçları tekrar istemciye iletir. İstemciler kaynakların yerlerini ve gerçekleştirim ayrıntılarını bilmek zorunda değildir.

Ortak Nesne Modeli: Tüm DNY'lerin desteklediği standart bir modeldir. DNYS oluşturan nesneler, ortak nesne modeline bağlı kalmak koşuluyla bağımsız olarak tasarlanabilmektedir.

DNYS için değişik standartlar geliştirilmiştir. Nesne yönetim grubu (Object Management Group-OMG) tarafından ortaya konulan CORBA (Common Object Request Broker Architecture-Ortak Nesne İstek Sunucu Mimarisi) günümüzde en çok kullanılan standartlardan birisidir. Değişik yazılım kuruluşları tarafından CORBA standardına uygun değişik ürünler çıkarılmıştır. DEC-Object Broker, DEC-ACA, Sunsoft-Distributed Objects Everywhere, IBM-System Object Model, HP-Distributed Smalltalk CORBA mimarisini temel alan değişik ürünlerdir (Kılıç vd., 1995).

CORBA, ORB (Object Request Broker-Nesne İstek Sunucu) birimini istemci ve sunucular arasında bir arabirim olarak tanımlamaktadır. İstemciler, isteklerini ORB'ye bildirirler ve ORB istekleri yerine getirebilecek bir sunucu veya sunucular arar. Sunucuların ağ üzerinde nelerde

olduğunu sadece ORB bilmek zorundadır. ORB, DNYS mimarisindeki DNY birimlerine karşılık gelmektedir. CORBA'da istemci ve sunucunun birbirinden ayrımı iyi tanımlanmıştır. İstemci sadece belli bir sunucuyu nasıl çağıracağını bilmek zorundadır, servisin nasıl gerçekleştirildiği istemci açısından önemli değildir. İstemci ve sunucu "istek" adı verilen iletiler ile iletişime geçmektedir. İstemciden gelen istekler, yapılması gereken işi ve işin hangi nesne üzerinde yapılacağını belirlemektedir. Bir istemcinin istekte bulunabileceği ve bir sunucunun karşılık verebileceği nesneler ve işlemler arayüzler ile tanımlanmaktadır. Bu arayüzler hem istemci hem de sunucular tarafından desteklenmektedir.

Arayüzler, CORBA'da arayüz tanımlama dili (Interface Definition Language-IDL) ile tanımlanmaktadır. Arayüzlerde tanımlanan her işlem için sunucuda bir gerçekleştirim bulunmaktadır. Aşağıda bir örnek arayüz tanımlaması verilmektedir.

```
Interface Aritmetik
```

```
{
```

```
integer topla (IN integer A, IN integer B, OUT integer C)
```

```
integer çarp (IN integer A, IN integer B, OUT integer C)
```

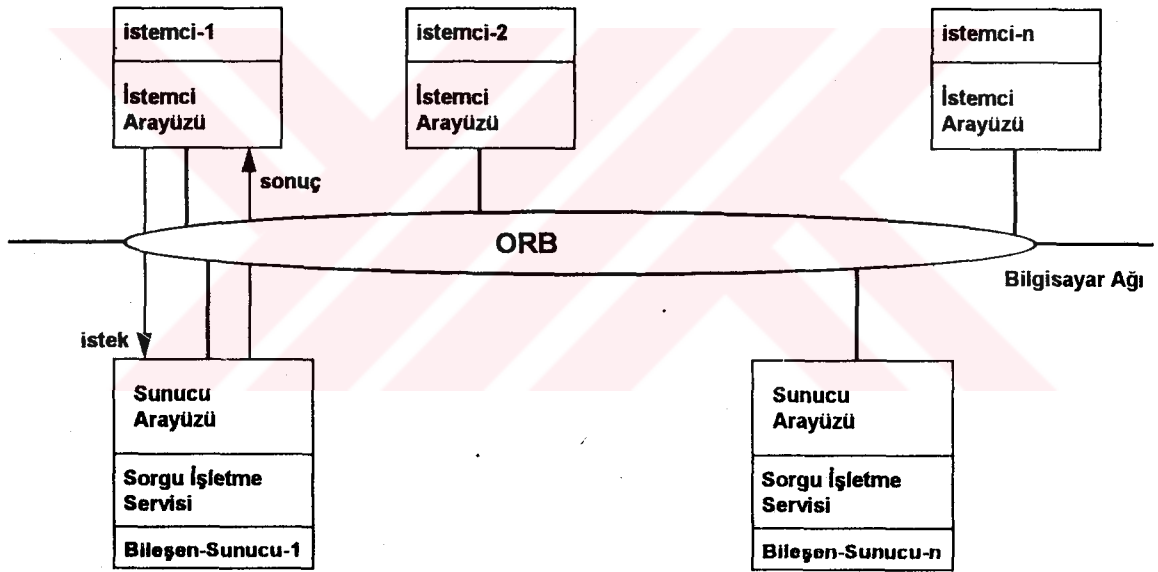
```
};
```

Yukarıda iki aritmetiksel işlem için bir arayüz tanımlanmıştır. Bu arayüz ile bir istemci, toplama veya çarpma işlemlerinin yapılmasını sunucudan isteyebilmektedir. İsteğin karşılanması için A ve B gibi iki tamsayının sunucuya iletilmesi gerekmektedir. Sonuçta bulunan değer C tamsayısı içinde istemciye geri döndürülmektedir. Toplama ve çarpma işlemlerini yapan gerçekleştirmeler sunucuda bulunmaktadır.

CORBA'da istemciler ve sunucular tek bir süreç veya birden fazla süreç olabilmektedir. İstemciler ve sunucular arasında birebir eşleme olması koşulu yoktur. Tek bir istemci birden fazla sunucudan istekte bulunabilmekte veya bir sunucu birden fazla istemciyi besleyebilmektedir.

CORBA, eş-zamanlı ve eş-zamansız iletişim türlerini desteklemektedir. Eş-zamanlı iletişim, bir yazılımın diğerine ileti gönderip yanıt gelene dek beklemesini söylerken, eş-zamansız iletişim ileti gönderen yazılımın yanıt gelmesini beklemeyerek işleme devam etmektedir. Hangi tür iletişimin seçileceği geliştirilen uygulamanın türüne bağlıdır.

CORBA standardının GTK ortamı modellemeye kullanımı Şekil 5.3'de görülmektedir.



Şekil 5.3 : GTK Ortamının CORBA ile Modellenmesi

İstek ve sonuçların iletimi şekli daha karmaşık bir hale getirmemek için sadece istemci-1 ve bileşen-sunucu-1 üzerinde gösterilmiştir.

Şekildeki mimari, GTK ortamları için daha önce sözü edilen ilk yazılım geliştirme yaklaşımını desteklemektedir. Bileşen-sunucu ortamında her bir bileşen bir sunucu olarak tanımlanıp sunucu arayüzleriyle birlikte Şekil 5.3'deki sorgu işletme servisine eklendiği zaman ikinci yaklaşım da desteklenmiş olmaktadır.

İlk yaklaşımın desteklenebilmesi için istemci yazılımlar istek tanımlayıcılarını belirlemek için kullanılmaktadır. Bu yaklaşımda sunucu, sorguyu işletip, bulduğu bileşenin içeriğini istemciye geri göndermektedir.

İkinci yaklaşımın desteklenmesinde istemci ve sunucu birimlerinin görevleri daha farklıdır. Bu yaklaşımda bileşen-sunuculardaki tüm bileşenler ayrı bir sunucu olarak düşünülmektedir. İstemci tarafında diğer yaklaşımda olduğu gibi istek tanımlayıcıları belirlenmemektedir. İstemci yazılım, değişik bileşen-sunuculardaki bileşenlerden (sunuculardan) birtakım isteklerde bulunmaktadır. Bu durumda istemci yazılım, değişik bileşenlerin birleştirildiği yazılım olmaktadır. Bu yaklaşımın diğer yaklaşımla ilişkisi, istemci tarafından birleştirilecek bileşenlerin daha önceden ilk yaklaşımdaki sorgu işletme servislerinin kullanımı ile belirlenebilmesindendir.

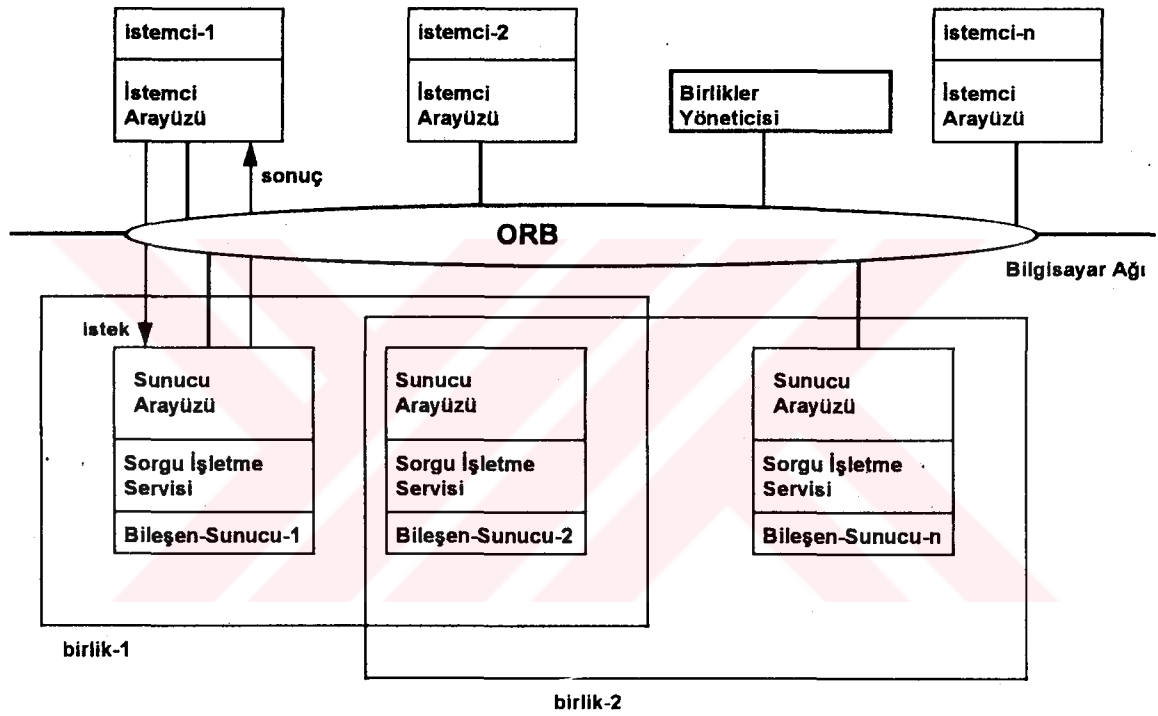
Bu yaklaşım aynı zamanda mega-programlama fikrini de desteklemektedir. Mega-programlama, mega-modüller denilen yazılım bileşenleri ile programlamadır (Wiederhold, 1992). Mega-modüller ağ üzerinde farklı yerlerde ve bankalar, havayolu rezervasyon sistemleri, taşımacılık şirketleri uygulamaları gibi geniş uygulama alanlarına ilişkin her türlü veri, yordam ve fonksiyonlara sahiptir. Programlar, mega-programlama dili ile yazılmaktadır. Burada mega-modüller bileşenlere (sunucular) -bileşenlerin kısa veya çok uzun olabileceğinden sözedilmiştir-, mega-programlama dili de istemci yazılıma karşılık gelmektedir. Mega-program, içinde bulunan mega-modüllere argümanlar geçirmekte, mega-modüller bu argümanları ilk önce işleyip daha sonra sonuçları mega-programa geçirmektedir. Mega-programdan mega-modüllere argüman geçişi yoluyla iletişim, istemciden sunucu arayüzüne uygun argümanlarla sunucuların çağırılmasına benzemektedir.

5.2 Birlikler Topluluğu Olarak GTK Ortamı

Bir veri tabanları birliğini tanımlayan 3 özellik GTK ortamında da vardır. Bu üç özellik dağıtıklık, çok türülük ve tasarım kendi-yönetimidir (otonomi). Dağıtıklık özelliği, verinin birliği oluşturan veri tabanları arasında dağıtılması anlamına gelmektedir. (Sheth, 1990). GTK ortamında da bileşenler değişik bileşen-sunuculara dağılmış biçimdedir. Çok türülük özelliği, birliği oluşturan veri tabanları yönetim sistemlerinin değişik donanım alt yapılarında, farklı işletim sistemleri üzerinde olmasından kaynaklanmaktadır. Ayrıca, VTYS(Veri Tabanı Yönetim Sistemi) 'nin değişik yapılarda (sıradüzensel veya ilişkisel) ve farklı sorgu dilleri içeriyor olması, verinin değişik gösterimlerde saklanması çok türülük özelliğinin sonucudur. GTK ortamındaki bileşen-sunucular da farklı donanım alt yapılarında, farklı işletim sistemleri üzerinde olmakta, değişik sınıflandırma tasarımları kullanılarak sınıflandırılmakta, geri elde etme araçları farklı kütük sistemleri veya VTYS dayanmaktadır. Tasarım kendi-yönetimi özelliği, birliği oluşturan veri tabanı sistemlerinin bağımsız olarak tasarlanmasıdır. GTK ortamındaki bileşen-sunucular da bulundukları ortamlarda bağımsız olarak tasarlanmakta, içerdikleri bileşenler üretilmekte ve daha sonra birliğe katılma başvurusu yapmaktadırlar. Bu noktada sıkı-eşlenmiş (tightly-coupled) birlik kavramının açıklanması gerekmektedir. Bu tür birliklerde, belli bir birlikler yönetim grubu olmalıdır. Bu grubun, birliklerin kurulması, tekrar kullanıcıların birliklere üye olması, bazı birliklere yeni bileşen-sunucular eklenmesi, bazılarının çıkarılması, birliklere katılmak isteyen tüm bileşen-sunucular için standartlar getirilmesi gibi görevleri bulunmaktadır.

Bir GTK ortamında birden fazla birlik bulunabilmektedir. Birlikler de belli sayıda bileşen-sunuculardan oluşmaktadır. Birbirine yakın uygulama alanlarına ilişkin bileşenleri içeren bileşen-sunucuların bir birlik oluşturmaları bir yaklaşımdır. Bir bileşen-sunucu birden fazla birlikte yer alabilmektedir. GTK ortamındaki her bileşen-sunucuda farklı sınıflandırma tasarımları olabilmektedir. Bu durumda, GTK ortamındaki herhangi bir birlikte bulunan bileşen-sunuculara ilişkin tüm sınıflandırma tasarımları o birliğe üye olmak isteyen kullanıcı ortamlarına gönderilmektedir. Böylece, istek tanımlayıcı belirleme işlemlerinin kullanıcı tarafında yapılması sağlanarak, ortamdaki erişim sayısı ve dolayısıyla ağ yükü azalmaktadır. GTK ortamı CORBA ile modellenirken istemci yazılımların istek tanımlayıcılarını belirleyecek biçimde tasarlanması bu nedendendir (bkz. Şekil 5.3).

Birliklerden oluşan GTK ortamlarındaki istemci yazılımların görevleri arasında kullanıcıların bileşenleri seçecekleri birlikleri ve bileşen-sunucuları belirlemede yardımcı olmak, istek tanımlayıcısının belirlendikten sonra ilgili bileşen-sunucuya iletilmesi, sonuçların geri alınması sayılabilmektedir. Her kullanıcı istediği birliğe üye olabilmektedir. Her birlik kendi içinde bulunan bileşen-sunuculardaki bileşenlerin dağıtımı ve pazarlanması için kendi politikalarını belirlemektedir. Kullanıcılar belli bir birliğe üye olmadan önce, o birliğe ilişkin dağıtım ve pazarlama politikalarını kesinlikle incelemek zorundadır.



Şekil 5.4 : Birliklerden Oluşan GTK Ortamı

Şekil 5.4 iki birlikten oluşan GTK ortamını göstermektedir. Birlikler yöneticisinin görevleri aşağıda sıralanmıştır.

a)Birliklere yeni bileşen-sunucular katılması veya bir bileşen-sunucunun çıkarılmasının sağlanması.

b)Yeni birlikler oluşmasına karar vermek.

c)Değişik bileşen-sunucuların sınıflandırma tasarılarındaki değişiklikleri kullanıcı ortamlarına bildirmek ve kullanıcı ortamlarındaki istemci yazılımların ve arayüzlerin güncellenmesini sağlamak.

d)Kullanıcıların istedikleri birliklere üye olmasını ve istediklerinden çıkarılmasını sağlamak.

e)Belli bir birliğe katılmak isteyen bileşen-sunucular için standartlar getirmek.

Şekil 5.4 'de görülen ortamda daha önce sözü edilen ilk yaklaşımla yazılım geliştirme yöntemi aşağıda verilmiştir:

1-Kullanıcılar birlikler listesinden bir birlik seçer.

2-Kullanıcılar seçtikleri birliğe ilişkin bileşen-sunucular listesinden bileşen-sunucu seçer.

3-İstemci yazılım seçilen bileşen-sunucunun sınıflandırma tasarısını tekrar kullanıcının önüne getirerek istek tanımlayıcıyı belirlemesini ister.

4-İstek tanımlayıcı belirlendikten sonra ORB ile sunucuya gönderilir. Burada istek tanımlayıcısı ilgili kütük sistemi veya veri tabanından kayıtlara erişebilecek sorgulara çevrilerek işletilir. Sonuçta bileşenin kendisi veya işletim sonuçları tekrar kullanıcı ortamına gönderilir.

5.3 DEC-ACA ile GTK Ortamında Bileşen Geri Elde Etme Aracı Gerçekleştirimi

Bu bölümde, CORBA standartlarına uygun bir yazılım aracı olan DEC-ACA kullanılarak GTK ortamındaki bir bileşen geri elde etme aracının gerçekleştirimi anlatılmaktadır. Sözkonusu GTK ortamı bölüm 5.1'de önerilen mimari yapısındadır (bkz. Şekil 5.3). Bileşen geri elde etme aracı, istemci yazılım, sunucu yazılım, DEC-ACA servisleri, metodları, metod-sunucuları ile bir bileşen kitaplığından (bkz. Bölüm 4.4) oluşmaktadır. Bu tür bir bileşen geri elde etme aracının gerçekleştirim adımları bölüm 5.3.2'de verilmeden önce, DEC-ACA yazılım aracı bölüm 5.3.1'de kısaca tanıtılmaktadır.

5.3.1 DEC-ACA

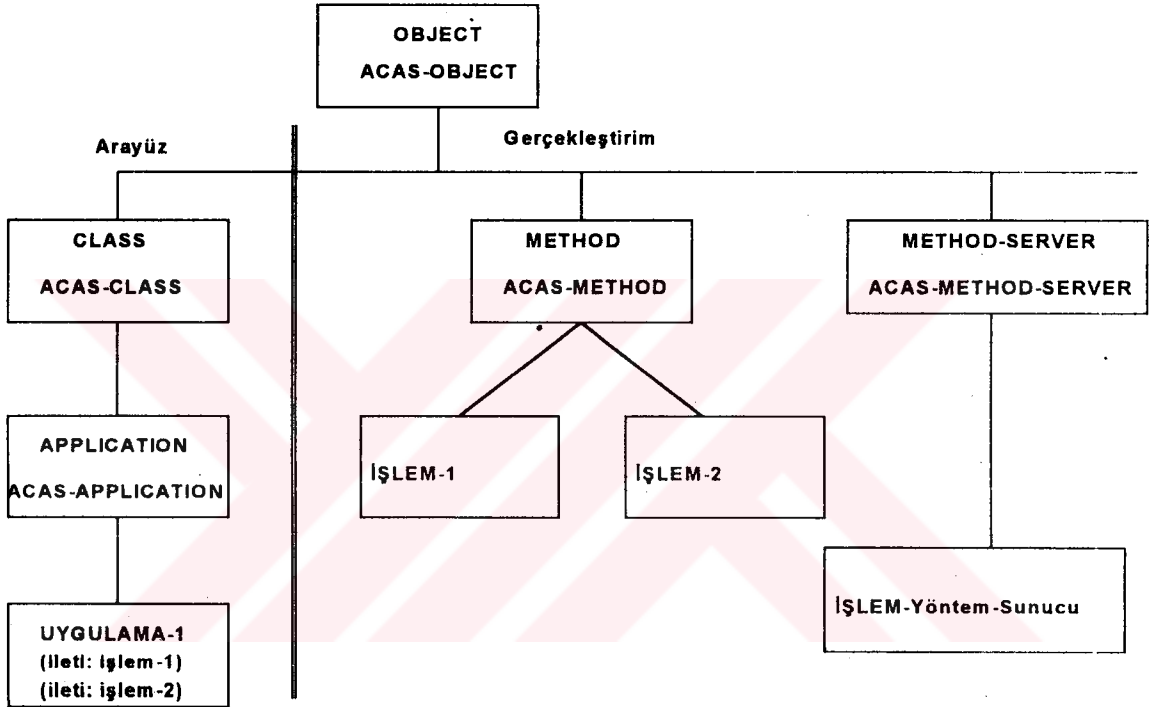
Bu bölümde verilen bilgiler, (DEC, 1992) kaynağından derlenmiştir.

DEC-ACA (Application Control Architecture-Uygulama Kontrol Mimarisi), farklı yazılım ve donanım ortamlarındaki uygulamaları birleştirebilme veya bu uygulamaların içten-işletilebilirliğini (interoperability) sağlama olanaklarını sunan alt düzeyli servisler topluluğudur.

DEC-ACA servislerinden faydalanmak isteyen uygulamaların nesneye dayalı yöntemlerle modellenmesi gerekmektedir. DEC-ACA, uygulamaların modellenmesi için bir takım temel sınıflar sunmaktadır. Uygulamaları modellemek için programcılar tarafından belirlenen sınıflar mutlaka temel sınıflardan birisine bağlı olmak zorundadır. Modelin en üstünde ACAS-OBJECT isimli bir sınıf bulunmaktadır.

Bir uygulama modellenirken, uygulamanın arayüzü (interface) ve gerçekleştirimi (implementation) ayrı ayrı modellenmektedir. Uygulamanın arayüzünü gösteren sınıflar, ACAS-CLASS ve ACAS-APPLICATION temel sınıflarına bağlı olarak tanımlanmaktadır. Bu sınıflar birtakım iletiler içermektedir. İletiler, o sınıflara ilişkin davranışları belirlemektedir. Bu iletilerin değişik gerçekleştirimleri ise metodlar olarak yapılmaktadır. Metod gerçekleştirimleri ACAS-

METHOD temel sınıfına bağlı olarak modellenmektedir. Bir metod-sunucu (method-server), bir grup metod içermektedir ve gerektiğinde ilgili metodun işletimini sağlamaktadır. Metod-sunucular, ACAS-METHOD-SERVER temel sınıfına bağlı olarak modellenmektedir. Metod ve metod-sunucu tanımları modelin gerçekleştirim kısmını oluşturmaktadır. Şekil 5.5, DEC-ACA ile modellenen bir uygulamayı göstermektedir. Uygulama-1, İşlem-1, İşlem-2, ve İşlem-Yöntem-Sunucu isimli sınıflar kullanıcı tanımlı sınıflar, diğerleri DEC-ACA temel sınıflarıdır.



Şekil 5.5 : DEC-ACA ile Uygulama Modellemesi

Sınıf bilgi tabanları (class repositories) ve çevresel nesneler (context objects) DEC-ACA'nın temel öğelerindendir. Sınıf bilgi tabanları, tüm kullanıcı tanımlı sınıfların tanımlarını ve bu sınıfların hangi temel sınıflara bağlı olduğuna ait bilgileri içermektedir. Çevresel nesneler, ACA servislerine kullanıcı seçimlerini ve uygulamaların çalışma ortamlarını (işletim sistemi, donanım

alt yapısı, bellek gereksinimi, vb.) tanıtan tablolardan oluşmaktadır. Tablolar ise kullanıcı tarafından belirli değerler atanmış niteliklerden (attributes) meydana gelmektedir.

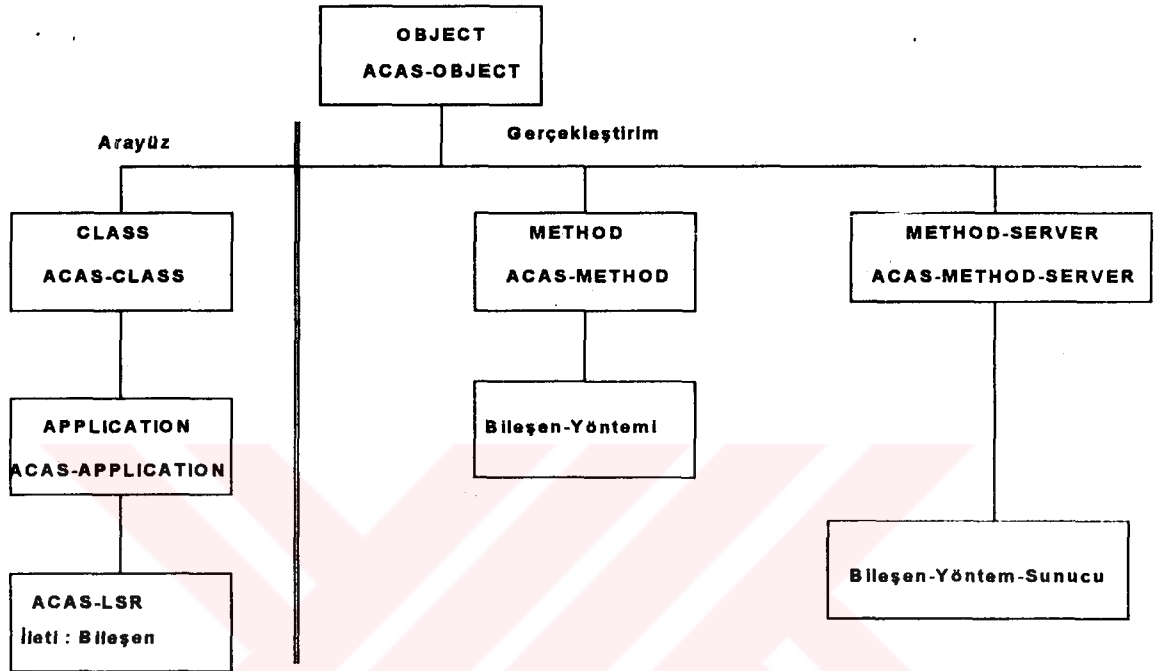
DEC-ACA'ya ilişkin diğer bir önemli öge ise denetim sunucusudur (control server). Denetim sunucusu, bir istemci veya sunucu yazılım çalıştırmadan önce kesinlikle çalıştırılmalıdır. Denetim sunucusunun görevi, çalıştığı düğümde (node) tüm metod-sunucularının çalışmasını başlatmak ve izlemektir. Denetim sunucusu, işletilebilir bir halde DEC-ACA paketi içinde yer almaktadır. Unix ortamında genelde "etc" dizini altında bulunmaktadır.

Bir istemcinin DEC-ACA servislerini kullanması, genellikle bir istemci program içinden "ACAS-Invoke-Method" komutunun belirli parametreler ile kullanılması yoluyla gerçekleşmektedir. Bir istemci yazılım, bu komut ile bir servis istediğinde ilk önce isteğe karşılık gelen metodun seçilmesi işlemi yapılmaktadır. Belli bir metod seçildikten sonra, o metodun içinde bulunduğu metod-sunucu seçilmektedir. Metod ve metod-sunucu seçilme işlemleri sınıf bilgi tabanları ve çevresel nesnelerdeki bilgiler gözönüne alınarak yapılmaktadır. Metod-sunucu isteğe karşılık gelen metodu çalıştırdıktan sonra sonuçları istemciye geri döndürmektedir. Geri döndürülen sonuçları içeren parametreler, metod tanımında "OUT" anahtar kelimesinden sonra tanımlanmaktadır.

5.3.2 Bileşen geri elde etme aracının gerçekleştirim adımları

Örnek bir bileşen geri elde etme aracını DEC-ACA ile gerçekleştirmek için ilk önce geliştirilecek olan uygulamanın, DEC-ACA'ya tanıtılmak amacıyla nesneye dayalı yaklaşımla modellenmesi gerekmektedir. Bu amaçla modeldeki arayüzler, metodlar ve metod-sunucuların belirlenmesi gerekmektedir. Örnek GTK ortamında, boyutlara ayırmaya dayanan sınıflandırma yaklaşımı ve grid kütük yapısı kullanıldığı, ayrıca, bölüm 5.1'de bahsedilen ilk yaklaşım temel alınarak yazılım geliştirildiği düşünülmektedir. Bu durumda, istemci yazılım, boyutlardan terimlerin seçilerek istek tanımlayıcılarının oluşturulduğu ve istek tanımlayıcılarının sunucu yazılımlara gönderildiği yazılım olmaktadır. Sunucu yazılımın içinde bulunan metod, istemciden aldığı istek tanımlayıcısını işletmekte ve sonuçları geri döndürmektedir. Tanımlanacak olan metod-

sunucu ise sözkonusu metodu içermektedir. Şekil 5.6, bileşen geri elde etme aracının GTK ortamının DEC-ACA ile modellenmesini göstermektedir.



Şekil 5.6 : GTK Ortamında Bulunan Bileşen Geri Elde Etme Aracının DEC-ACA ile Modellenmesi

Şekil 5.6'da görülen modelde ACAS-LSR, Component-Method ve LSR-Method-Server kullanıcı tanımlı sınıflardır. Component-Method istek tanımlayıcısını parametre olarak almakta ve bulunan bileşen(lerin) içeriklerini gösteren işaretçileri geri döndürmektedir.

Uygulama modellendikten sonra, sınıf bilgi tabanları ve çevresel nesneler tanımlanmakta ve ACASLD komutunun belli parametrelerle kullanımı ile derlenmekte ve yüklenmektedir. Sınıf bilgi tabanları ve çevresel nesneler yüklendikten sonra, istemci ve sunucu yazılımlar C dilinde yazılmalıdır. İstemci yazılımın ve sunucu yazılımın algoritmaları aşağıdaki gibidir:

İstemci Yazılım Algoritması

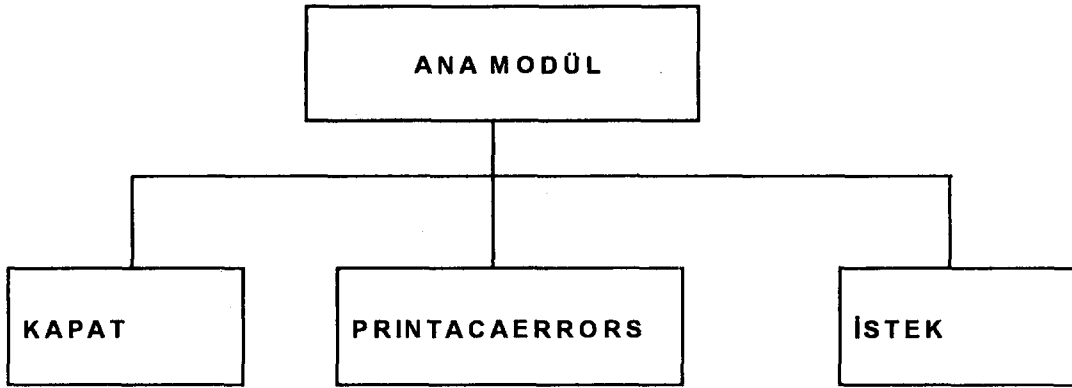
1. Boyutlardan terim seçerek istek tanımlayıcılarını oluştur.
2. ACAS-Invoke-Method ile istek tanımlayıcılarını işleten ve sonuçları döndüren metodu çağır.
3. Sonuçları bekle ve işleme koy.

Sunucu Yazılım Algoritması

1. Örnek kitaplığı kullanarak grid kütük yapısını oluştur.
2. İstek tanımlayıcısına karşılık gelen bileşen tanımlayıcısını grid kütük yapısında bul.
3. Bileşenin içeriğini gösteren işaretçiyi döndür. (Diğer bir seçenek bileşenin içeriğini döndürmektir. DEC-ACA ile karakter katarı şeklinde bir dizin döndürülemediği için tamsayı tanımlı işaretçiler döndürülmüştür.)

GTK ortamlarındaki bileşen geri elde etme aracının DEC-ACA ile gerçekleştirimi için yazılan istemci programın yapısı Şekil 5.7'de görülmektedir.

İstemci programdaki kapat isimli fonksiyon, metod sunucu ile bağlantıyı kesmektedir. PrintACAErrors isimli fonksiyon, ACA servisleri kullanımı sırasında oluşan hataları listelemektedir. İstek isimli fonksiyon, kullanıcıların istek tanımlayıcılarını oluşturmalarını sağlamak ve oluşturulan istek tanımlayıcısına karşılık gelen bileşenlerin bulunması için sunucu programdan istekte bulunmaktadır.



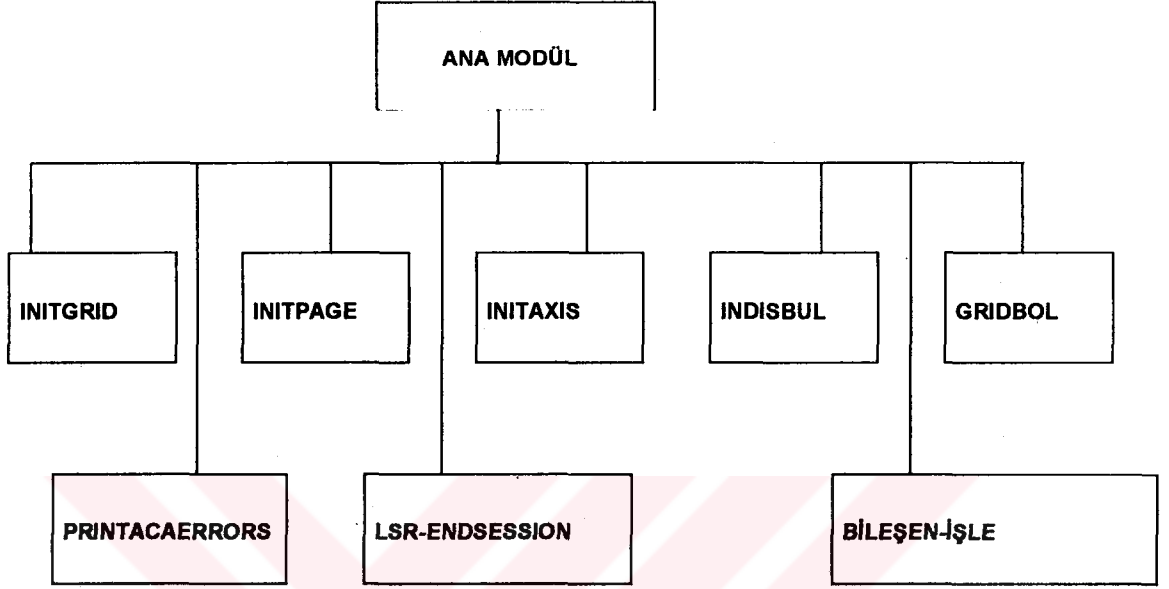
Şekil 5.7 : İstemci Program Yapısı

Sunucu programın yapısı Şekil 5.8'de görülmektedir. Initgird, initpage, initaxis, indisbul ve gridbol daha önce anlatılan ve grid kütük yapısı oluşturmak için kullanılan fonksiyonlardır. PrintACAErrors, ACA servisleri kullanımı sırasında oluşan hataları listelemektedir. LSR-EndSession, sunucu program işlevini tamamladığında ilgili metod sunucuyu kullanımdan kaldırmaktadır. Bileşen-İşle isimli fonksiyon, istemciden gelen istek tanımlayıcılarını grid kütük yapısında eşlemekte ve eşlenen bileşenlerin bulundukları yerleri gösteren işaretçileri istemciye geri döndürmektedir.

İstemci ve sunucu yazılımlar, C derleyicisi ile "-lacas" seçeneği kullanılarak derlenmelidir. Sunucu yazılımının bir kısmı (Metod-sunucunun metodu bulup çalıştırmasını sağlayan kısmı) ACASGEN komutu ile kendiliğinden üretilmektedir. DEC-ACA ile yazılan istemci ve sunucu programlara ilişkin kaynak kodlar ek-4 bölümünde bulunmaktadır.

Bu işlemlerden sonra ve denetim sunucusunun çalıştığından emin olduktan sonra, sunucu yazılım geri planda çalıştırılmakta ve istemci yazılımın istek tanımlayıcılarını iletmesini beklemektedir. GTK ortamının CORBA ile modellenmesinde görülen istemci yukarıda bahsedilen istemci yazılımdır. Sorgu işletme servisi olarak gösterilen birim ise sunucu yazılıma karşılık

gelmektedir. ORB ve diğer arayüzler ise DEC-ACA servisleri tarafından sağlanmaktadır (Bkz. Şekil 5.3).



Şekil 5.8 : Sunucu Program Yapısı

BÖLÜM-6 SONUÇLAR

Bu çalışmada ilk olarak, GTK ortamlarında da kullanılabilecek yüksek verimli bir bileşen geri elde etme aracı tasarlamak için çoklu-nitelikli kütük yapılarının kullanılması önerilmektedir. Bu önerinin doğruluğunu kanıtlamak amacıyla, çok bilinen ve kullanılan bir çoklu-nitelikli kütük yapısı olan grid kütük yapısı oluşturularak, tek-nitelikli kütük yapısı ile verimleri karşılaştırılmıştır. Bu amaçla 2 ve 3 boyutlu grid kütük yapıları oluşturularak, rastgele olarak üretilen belirli sayıdaki bileşen tanımlayıcısı bu yapılarda saklanmıştır. Yine rastgele olarak üretilen belirli sayılardaki istek tanımlayıcılarının oluşturulan grid kütük yapısı üzerinde bileşen tanımlayıcılar ile eşlenebilmesi için aranması gereken fiziksel disk sayfası sayısı bulunmuştur. Bu sayı, istek tanımlayıcılarının tek-nitelikli kütük yapıları üzerinde bileşen tanımlayıcıları ile eşlenebilmesi için gereken fiziksel disk sayfası sayısı ile karşılaştırılmıştır. Bileşen tanımlayıcıları ile istek tanımlayıcılarının hangi koşullara bağlı olarak rastgele üretildiği ve tek-nitelikli kütük yapıları üzerinde eşleme işlemi için aranması gereken fiziksel disk sayfası sayısının nasıl bulunduğu "Grid Kütük Yapısının Performansı" isimli bölümde (bölüm 4.3) ayrıntılı olarak anlatılmaktadır.

2 ve 3 boyutlu grid kütük yapıları oluşturmanın amacı, boyut sayısındaki değişimin verimlilik üzerine etkilerini araştırmaktır. Aşağıda, 2 ve 3 boyutlu grid kütük yapılarında değişik sayılarda bileşen saklandığı durumlarda, yine değişik sayılardaki istek tanımlayıcılarının işletimi sonucunda elde edilen erişilen fiziksel disk sayfası sayısına ilişkin sonuçlar hem grid kütük yapısı hem de tek-nitelikli kütük yapısı için grafikleri ile birlikte görülmektedir.

Boyut Sayısı:2
Disk Sayfası Kapasitesi:50
İstek Tanımlayıcı Sayısı:25

<u>Bileşen Tanımlayıcı Sayısı</u>	<u>Grid Disk Sayfa Sayısı</u>	<u>Tek-Nitelikte Disk Sayfa Sayısı</u>
100	35	33
250	47	47
500	74	97
750	79	116

1000

85

139

Boyut Sayısı:3
 Disk Sayfası Kapasitesi:50
 İstek Tanımlayıcı Sayısı:25

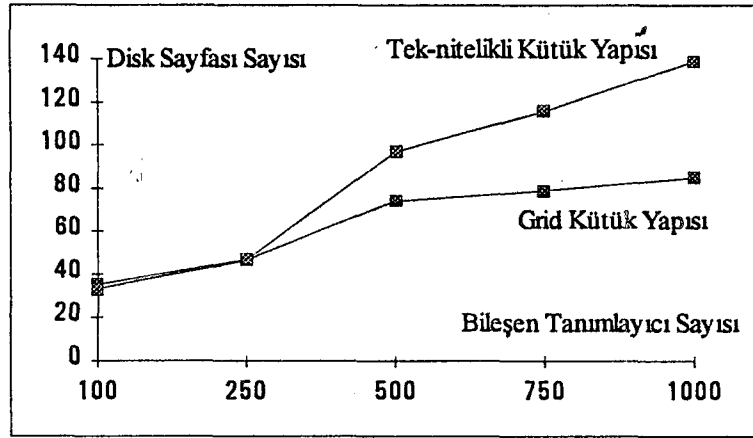
<u>Bileşen Tanımlayıcı Sayısı</u>	<u>Grid Disk Sayfa Sayısı</u>	<u>Tek-Nitelikte Disk Sayfa Sayısı</u>
100	39	33
250	65	61
500	92	99
750	101	134
1000	131	196

Boyut Sayısı:3
 Disk Sayfası Kapasitesi:50
 İstek Tanımlayıcı Sayısı:50

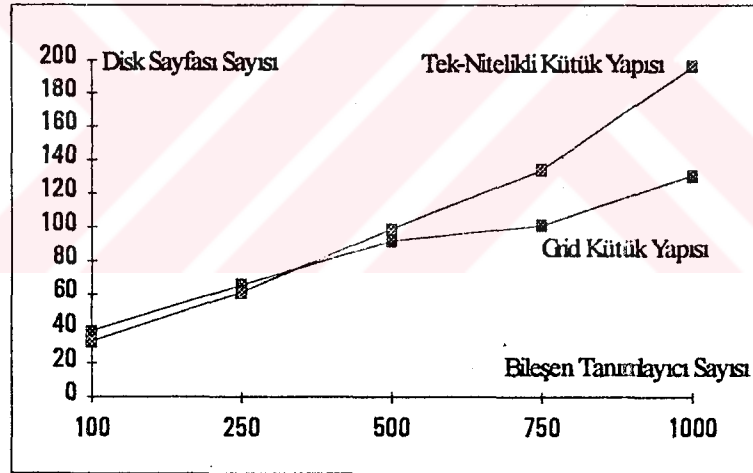
<u>Bileşen Tanımlayıcı Sayısı</u>	<u>Grid Disk Sayfa Sayısı</u>	<u>Tek-Nitelikte Disk Sayfa Sayısı</u>
100	76	64
250	110	112
500	181	185
750	214	302
1000	281	430

Boyut Sayısı:3
 Disk Sayfası Kapasitesi:50
 İstek Tanımlayıcı Sayısı:100

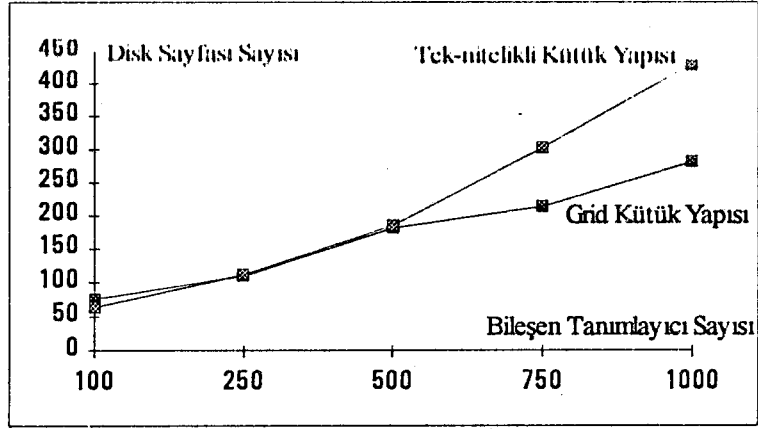
<u>Bileşen Tanımlayıcı Sayısı</u>	<u>Grid Disk Sayfa Sayısı</u>	<u>Tek-Nitelikte Disk Sayfa Sayısı</u>
100	159	133
250	227	220
500	360	379
750	406	527
1000	499	813



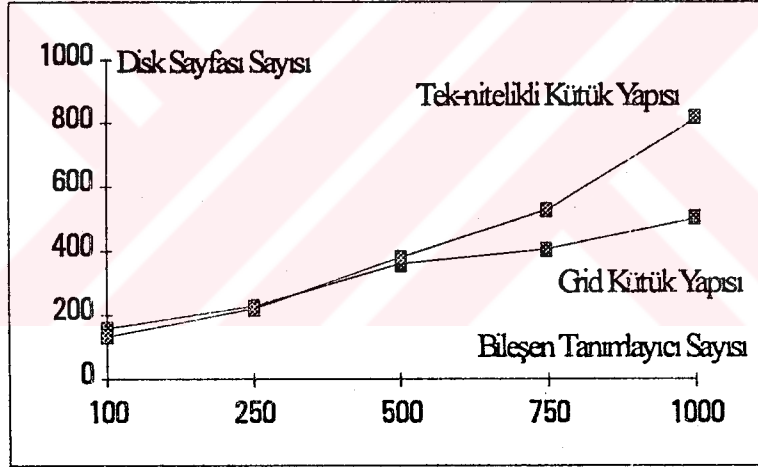
Şekil 6.1 : Boyut=2, İstek Tanımlayıcısı=25 iken Sonuçlar



Şekil 6.2 : Boyut=3, İstek Tanımlayıcısı=25 iken Sonuçlar



Şekil 6.3 : Boyut=3, İstek Tanımlayıcısı=50 iken Sonuçlar



Şekil 6.4 : Boyut=3, İstek Tanımlayıcısı=100 iken Sonuçlar

Sonuçlardan görüldüğü gibi, kitaplıkta az sayıda bileşen olduğunda grid kütük yapısının verimi, tek-nitelikli kütük yapılarından iyi olmamaktadır. Bir eşitlik noktasından sonra -bu nokta tüm çizgelerde yaklaşık 250 bileşen olarak görülmektedir- grid kütük yapısı ile erişim daha etkin

olmaktadır. Bu durum, çok sayıda bileşenin bulunduğu (250'den fazla) kitaplıklar için grid kütük yapısının veriminin daha iyi olduğunu göstermektedir.

Grafiklerden çıkarılan diğer bir sonuç ise boyut sayısı artmasına rağmen grid kütük yapısının aynı verimi devam ettirmesidir. Şekil 6.2 'de görüldüğü gibi boyut sayısının 2'den 3'e çıktığı durumunda da grid kütük yapısı 2 boyutlu durumdaki verimini sürdürmektedir.

İşletilen istek tanımlayıcı sayısının artması her iki kütük yapısına da doğrusal olarak etki etmektedir. Bunun anlamı, istek tanımlayıcı sayısının artması, az sayıda bileşen bulunduğu durumlarda tek-nitelikli kütük yapılarından yana, bileşen sayısı 250'den fazla iken grid kütük yapısından yana olmaktadır.

Grid kütük yapısının, kullanıcılar açısından bakıldığında verimlilik dışında diğer üstünlükleri de vardır. İlk üstün yan, istek tanımlayıcılarının belirlenmesi sırasında ortaya çıkmaktadır. Çoklu-nitelikli kütük yapılarında ve dolayısıyla grid kütük yapısında da her nitelik anahtar olabildiği için istek tanımlayıcılarının belirlenmesi işlemine esneklik kazandırılmış olmaktadır. Tek-nitelikli kütük yapısı kullanıldığında, indis olan niteliğin istek tanımlayıcılarında kesinlikle belirlenmiş olması gerekmektedir. Grid kütük yapısı ile istenen sayıda nitelik belirlenebilmekte ve bu niteliklerin yeri önemli olmamaktadır. İkinci üstün yan ise kitaplığa yeni bileşenler ve dolayısıyla yeni bileşen tanımlayıcılarının eklenmesinin grid kütük yapısının dinamik özelliğinden dolayı kolayca gerçekleşmesinden kaynaklanmaktadır.

Grid kütük yapısının yukarıda sözü edilen verimlilik üstünlüğü ve diğer üstünlükleri gözönüne alınarak , bu çalışma kapsamında ayrıca, örnek bir kitaplık üzerinde çalışan ve grid kütük yapısına dayanan bir bileşen geri elde etme aracı tasarlanmıştır. Bu araç ilk olarak yerel bir BYYG ortamı için tasarlanmıştır. Bileşen geri elde etme aracının sınıflandırma tasarısı olarak boyutlara ayırmaya dayanan sınıflandırma yaklaşımı kullanılmıştır. Bu yaklaşımın etkinliği birçok çalışmada kullanım bulduğu için bu konuda bir yenilik getirilmesi üzerinde durulmamıştır. Geliştirilen, geri elde etme aracı ile, temel veri yapıları ve algoritmalara ilişkin 61 bileşenin bulunduğu kitaplığa 5 boyutlu grid kütük yapısı üzerinden erişim sağlanmıştır. 5 boyutlu grid kütük yapısı kullanılmasının nedeni, bileşenleri sınıflandırmak için 5 niteliğin kullanılmış olmasından ileri gelmektedir. Örnek kitaplık nesneye dayalı yöntemlerle hazırlanmış temel veri

yapıları ve algoritmalarından oluşmaktadır. Bu kitaplık, Ege Üniversitesi Bilgisayar Mühendisliği bölümünde bir lisans tezi çalışması olarak hazırlanmıştır (Dindir ve Kurt, 1995).

Bu çalışmada son olarak, geniş ölçekli tekrar kullanım (GTK) ortamları için bir yazılım mimarisi ve BYYG yaklaşımının bu mimari üzerinde gerçekleştirilebilmesi için değişik yazılım geliştirme yöntemleri önerilmekte ve yerel BYYG ortamı için tasarlanan bileşen geri elde etme aracının GTK ortamlarına taşınması üzerinde durulmaktadır. Böylece bu konuda görülen eksikliklerin giderilmeye çalışılması amaçlanmaktadır. GTK ortamları, çok sayıda kullanıcı ve bileşen kitaplığı içeren ortamlardır. Bu nedenle yüksek verimli bileşen geri elde etme araçlarına özellikle GTK ortamlarında gereksinim duyulmaktadır. Önerilen GTK mimarisinde, etkinliği kanıtlanan çoklu-nitelikli kütük yapısına dayanan geri elde etme araçları kullanılmaktadır. GTK ortamları dağıtık sistem olarak modellenmeye uygundur. Bu nedenle önerilen mimari bir dağıtık nesne yönetim standardı olan CORBA'ya dayanmaktadır. Bu mimarinin, mega-programlama fikrini de destekleyecek şekilde oluşturulmasına çalışılmış ve mimaride bulunan bileşen geri elde etme aracı CORBA standardına uygun DEC-ACA yazılımı ile gerçekleştirilmiştir. Kullanılan DEC-ACA yazılım aracı 1992 yılına ilişkin bir ürün olup, birden fazla iş istasyonu üzerinde çalışması için DEC-NET isimli ağ protokolüne gereksinim duymaktadır. DEC-NET isimli protokolün kullanım alanı gün geçtikçe daralmakta olduğundan, bu protokolün yüklenmesinin bir anlam taşımayacağı düşünülmüştür. TCP/IP protokolünü destekleyen yeni DEC ürününün DEC-Object Broker olduğu öğrenilmiştir fakat iş istasyonlarına yükleme fırsatı bulunamamıştır. Bu nedenlerle, GTK ortamındaki bileşen geri elde etme aracı tek iş istasyonu üzerinde, fakat ileride istemci ve sunucu yazılımlarda hiçbir değişiklik yapılmadan birden fazla iş istasyonunda çalışacak şekilde gerçekleştirilmiştir. Bileşen geri elde etme aracının DEC-ACA ile gerçekleştiriminde, lisans tezi kapsamında DEC-ACA ile çalışan bir grup ile işbirliği yapılmıştır. DEC-ACA ile uygulama geliştirmenin ayrıntıları için (İplikçioğlu ve Sezgin, 1995) kaynağından yararlanmak mümkündür.

Bu çalışmada önerilen mimariye ek olarak, GTK ortamlarının birlikler topluluğu olarak modellenebilmesi fikri ilk defa ortaya konulmuştur. Bir GTK ortamının birlikler topluluğu olarak modellenmesi, GTK ortamının yönetimini kolaylaştırmakta ve belli kurallara dayandırmaktadır.

BÖLÜM-7 TARTIŞMA ve ÖNERİLER

Bir GTK ortamını tüm tekrar kullanımı destekleyici araçlarıyla birlikte gerçekleştirmek ve kullanıma açmak bu çalışmanın kapsamını aşmaktadır. Bu yüzden, sadece bileşen geri elde etme araçlarının tasarımı ve gerçekleştirimi üzerinde durulmuş, diğer araçların sadece önerilen mimari içindeki yerleri belirtilmiştir. Bu çalışmada, GTK ortamına ilişkin genel çerçeveyi ortaya koymak ve böyle bir ortamın gerçekleştirimi ve kullanıma sokulması için gerekli alt yapıyı araştırmak amacı güdülmüştür. Bu çalışma temel alınarak, bir GTK ortamını tümüyle ve etkin bir şekilde gerçekleştirmek için aşağıdakilerin yapılması önerilmektedir.

a)GTK ortamları için bileşen-tarayıcı aracının gerçekleştirimi

b)Kullanıcı ortamlarında bileşen-birleştirme ve bileşen-uyarlama araçları gerçekleştirimi

c)Bileşen geri elde etme aracına, kavramsal yakınlık çizgesi üzerindeki ağırlıkları bulmada kullanılacak "benzerlik fonksiyonları" tanımları eklenmelidir. Benzerlik fonksiyonlarının tanımlanması için yapay us tekniklerinden yararlanılması faydalı olmaktadır.

d)Grid kütük yapısının paralel bir hale getirilip istek tanımlayıcılarının paralel olarak işletiminin, GTK ortamındaki bileşen geri elde etme araçlarının verimini daha da artıracakı düşünülmektedir. Bu amaçla, GTK ortamının kullanımı uzun süre izlenmeli ve her bileşen için erişim sıklıkları bulunmalıdır. Bulunan erişim sıklıklarının yük olarak düşünülüp, paralel grid blokları arasında eşit olarak dağıtılabilmesi gerekmektedir.

e)GTK ortamlarında değişik uygulama alanlarına ilişkin çok sayıda bileşen kitaplığı bulunması gerekmektedir. Bu kitaplıklardaki bileşenlerin üretimi için bir yöntem geliştirilmeli ve bileşenler bu yöneme göre üretilmelidir.

KAYNAKLAR

Biggerstaff, T.J. and Perlis, A.J., 1984, Foreword, IEEE Transactions on Software Engineering, Se-10(5):474-477.

Chen, P.S., Hennicker, R. and Jarke, M., 1993, On The Retrieval of Reusable Components, in Advances in Software Reuse, IEEE Computer Society Press, Los Alamitos, California, 240p.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., 1994, Object Oriented Development: The Fusion Method, Prentice-Hall, New Jersey, 313p.

Digital Equipment Corporation, 1992, DEC-ACA Services System Integrator and Programmer Guide, Maynard, Massachusetts.

Dikenelli, O., Erdur, C. ve Alakent, B., 1995(a), Geniş Ölçekli Tekrar Kullanım İçin Bir Yazılım Mimarisi, TBD 12. Ulusal Bilişim Kurultayı, İstanbul.

Dikenelli, O., Erdur, C. and Alakent, B., 1995(b), A Software Architecture for Large Scale Reuse, submitted to ISCIS X, Ephesus, Turkey

Dindir, R. ve Kurt, İ., 1995, Nesneye Dayalı Yöntemlerle Yeniden Kullanılabilir Yazılım Modülleri Geliştirilmesi, Lisans Tezi, Ege Üniversitesi Bilgisayar Mühendisliği Bölümü, İzmir.

Freeman, P., 1987, A Conceptual Analysis of The Draco Approach to Constructing Software Systems, IEEE Transactions on Software Engineering, Se-13(7):830-844.

Fugini, M.G., Nierstrasz, O. and Pernici, B., 1992, Application Development Through Reuse: the Ithaca tools environment, in ACM SIGOIS Bulletin, 13(2):38-47.

Fugini, M.G. and Faustle, S., 1993, Retrieval of Reusable Components in a Development Information System, in Advances in Software Reuse, IEEE Computer Society Press, Los Alamitos, California, 240p.

Gossain, S. and Anderson, B., 1990, An Iterative Design Model for Reusable Object Oriented Software, ECOOP/OOPSLA'90 Proceedings, :12-25.

Griss, M.L., 1993, Software Reuse: from library to factory, IBM Systems Journal, 32(4):548-566.

Hall, P. and Weedon, R., 1993, Object Oriented Module Interconnection Languages, in Advances in Software Reuse, IEEE Computer Society Press, Los Alamitos, California, 204p.

İplikçioğlu, G., Sezgin, D., 1995, Application Development with DEC-ACA, Lisans Tezi, Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü, İzmir.

Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G., 1993, Object Oriented Software Engineering, Addison-Wesley Publishing Company, USA, 524p.

Kılıç, E., Ozhan, O., Dengi, C., Kesim, N., Köksal, P. and Doğan, A., 1995, Experiences in Using CORBA for a Multidatabase Implementation, METU.

Konstantas, D., 1993, Object Oriented Interoperability, Proceedings of ECOOP, Germany.

Krueger, C.W., 1992, Software reuse, ACM Computing Surveys, 24(2):131-183.

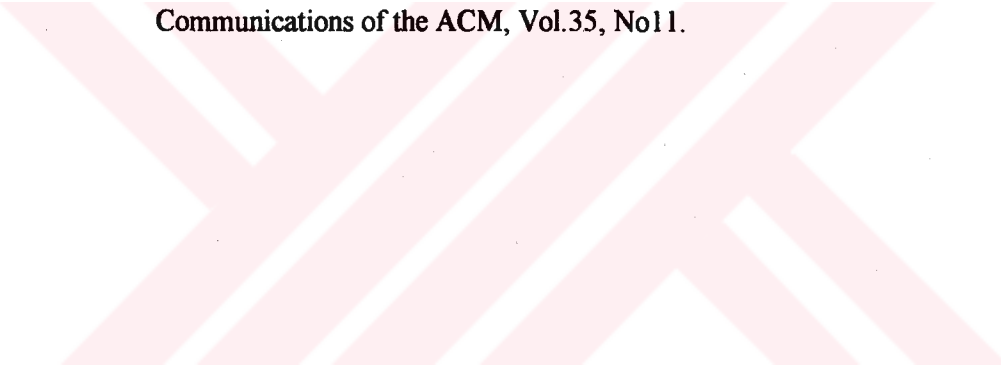
- Li, H. and Katwijk, J., 1992, Issues Concerning Software-in-the-Large, Delft University of Technology Faculty of Mathematics and Computer Science, Julianalaan 132, The Netherlands.
- Manola, F., Heiler, S., Georgakopoulos, D., Hornick, M. and Brodie, M., 1992, Distributed Object Management, *International Journal of Intelligent and Cooperative Information Systems*, Vol. 1, No. 1, pp.5-42.
- Morel, J. and Faget, J., 1993, The Reboot Environment, in *Advances in Software Reuse*, IEEE Computer Society Press, Los Alamitos, California, 204p.
- Nierstrasz, O., Gibbs, S. and Tsichritzis, D., 1992, Component Oriented Software Development, in *Communications of the ACM*, 35(9):160-165.
- Nievergelt, J. and Hinterberger, H., 1984, The Grid File: An Adaptable Symmetric Multikey File Structure, *ACM Transactions on Database Systems*, Vol.9, No.1.
- Prieto-Diaz, R. and Freeman, P., 1987, Classifying software for reusability, *IEEE Software*, January:6-16.
- Prieto-Diaz, R., 1991, Implementing Faceted Classification for Software Reuse, *Communications of the ACM*, 34(5):88-97.
- Prieto-Diaz, R., 1993, Status Report: Software Reusability, *IEEE Software*, May:61-66.
- Sheth, A. P. and Larsen, J. A., 1990, Federated Database Systems for Managing Distributed Heterogeneous and Autonomous Databases, *ACM Computing Surveys*, Vol.22, No.3.

Sorumgard, L.S., Sindre, G. and Stokke, F., 1993, Experiences from application of a faceted classification scheme, in *Advances in Software Reuse*, IEEE Computer Society Press, Los Alamitos, California, 240p.

Tsichritsiz, D., Nierstrasz, O. and Gibbs, S., 1992, Beyond Objects: Objects, *International Journal of Intelligent and Cooperative Information Systems*, Vol.1, No.1, pp.43-60.

Weber, H., 1993, Uniformity and invariance in support of reuse, in *Advances in Software Reuse*, IEEE Computer Society Press, Los Alamitos, California, 204p.

Wiederhold, G., Wegner, P. and Ceri, S., 1992, Toward Mega-programming, *Communications of the ACM*, Vol.35, No11.



EKLER

Ek.1

2 boyutlu Grid Kütük Yapısı Oluşturmak İçin C Dilinde Yazılan Program

Ek.2

3 boyutlu Grid Kütük Yapısı Oluşturmak İçin C Dilinde Yazılan Program

Ek.3

5 boyutlu Grid Kütük Yapısı Kullanan Bileşen Geri Elde Etme Aracı
Oluşturulması için C Dilinde Yazılan Program

Ek.4

Bileşen Geri Elde Etme Aracının DEC-ACA Nesne Sunucu (Object-Broker) ile İstemci /
Sunucu Biçiminde Modellenmesi için Yazılan Programlar

Ek1. İki Boyutlu Grid Kütük Yapısı Oluşturmak İçin C Dilinde Yazılan Program

Aşağıdaki program iş istasyonları ortamında Unix İşletim sistemi altında C dilinde yazılmıştır. Program, rastgele sayı üretme bölümündeki çok küçük değişiklikler ile kişisel bilgisayar üzerinde de çalıştırılmıştır.

Verilen programın kullandığı iki tip yapı vardır. Bunlardan birisi "gridstr" adında olup grid dizisine (grid array, $G(x,y)$) karşılık gelmektedir. Grid dizisinin her elemanın ise bir grid bloğuna karşılık geldiği bilinmektedir. Bu yapıda, grid blokların her doğrusal ölçek üzerindeki başlangıç ve bitim noktaları, grid blokların gösterdiği fiziksel disk sayfası numaraları ve 1 veya 0 değerleri alabilen işaret isimli bir değişken tutulmaktadır. İşaret değişkeni, dolmuş durumda olan ve bölünmesi gereken grid bloklarını belirlemektedir. İşaret değişkeninin "1" değerini alması grid bloğun bölüneceği anlamına gelmektedir. İkinci yapı "pagestr" adında olup, fiziksel disk sayfalarının ana bellek üzerinde benzetimi amacıyla kullanılmaktadır. Bu yapıda, fiziksel disk sayfasının numarası, bu disk sayfasının alabileceği en fazla kayıt sayısı ve kayıtların kendileri tutulmaktadır. Kayıtların bileşen tanımlayıcılar olduğu durumda, kayıtlar, bileşenin içeriğini elde etmek için bir işaretçi saha da içermektedir. Programda kullanılan eksen alt sıralı değişkeni doğrusal ölçekleri tutmak için, say alt sıralı değişkeni ise doğrusal ölçekler üzerindeki eleman sayılarını tutmak için kullanılmıştır. Grid oluşturma işlemine başlamadan önce çağrılan initgrid isimli fonksiyon grid bloklarını, initpage isimli fonksiyon benzetimi yapılan fiziksel disk sayfalarını, initaxis isimli fonksiyon ise doğrusal ölçekleri belli değerlerle ilkleme amacıyla kullanılmaktadır. İndisbul fonksiyonu ise, bir niteliğe ilişkin bir değer doğrusal ölçekler üzerinde karşılık geldiği indisi bulmaktadır. Gridbol isimli fonksiyon bölüm 4.1.2.1 'de anlatılan bölünme yöntemini uygulayarak grid bölünmesi işlemini gerçekleştirmektedir.

```
#include <stdio.h>
#include <stdlib.h>

#define KONST 10      /* Tek-nitelikli bir kütük yapısının sıradan arama durumunda
                        erişeceği toplam disk sayfası sayısı = COMPSAY / PAGESIZE */
#define BOYUT 2       /* Grid Yapısının Boyut Sayısı */
#define PAGESIZE 50   /* Bir fiziksel disk sayfasının alabileceği maksimum kayıt sayısı */
#define PAGESAYISI 70 /* Kullanımı mümkün olan fiziksel disk sayfası sayısı */
#define GRIDSAY 35    /* Grid-Dizisi (Grid-Array) uzunluğu tanımı */
#define COMPSAY 500   /* Bileşen tanımlayıcı sayısı */
```

```

struct gridstr
{
    int ilk[BOYUT+1];
    int son[BOYUT+1];
    int pageno;
    int isaret;
};

struct gridstr gptr[GRIDSAY+1][GRIDSAY+1];
struct gridstr gptryed[GRIDSAY+1][GRIDSAY+1];    /* Geçici Saha */

struct pagestr
{
    int pageid;
    int tupsay;
    int tuples[PAGESIZE+1][BOYUT+1];
};

struct pagestr pptr[PAGESAYISI+1];

int eksen[BOYUT+1][41];
int eksenyed[BOYUT+1][41];    /* Geçici Saha */
int bol,ensonpage,saklaind1,saklaind2;
int say[BOYUT+1];
int yedtuples[PAGESIZE+1][BOYUT+1];    /* Geçici Saha */
/* Grid bloklarını ilkleyen fonksiyon */
void initgrid()
{
    int i,j,k;
    for(i=1;i<=GRIDSAY;i++)
    {
        for(j=1;j<=GRIDSAY;j++)
        {
            gptr[i][j].pageno=0;
            gptr[i][j].isaret=0;
            gptryed[i][j].pageno=0;
            gptryed[i][j].isaret=0;
            for(k=1;k<=BOYUT;k++)
            {
                gptr[i][j].ilk[k]=0;
                gptr[i][j].son[k]=0;
                gptryed[i][j].ilk[k]=0;
                gptryed[i][j].son[k]=0;
            }
        }
    }
}

```

```
/* Disk sayfalarını ilkleyen fonksiyon */
```

```
void initpage()
```

```
{
    int i,j,k;
    for(i=1;i<=PAGESAYISI;i++)
    {
        pptr[i].pageid=i;
        pptr[i].tupsay=0;
        for(j=1;j<=PAGESIZE;j++)
            for(k=1;k<=BOYUT;k++)
                pptr[i].tuples[j][k]=0;
    }
}
```

```
/* Doğrusal ölçekleri ilkleyen fonksiyon */
```

```
void initaxis()
```

```
{
    int i,j;
    for(i=1;i<=BOYUT;i++)
    {
        for(j=1;j<=40;j++)
        {
            eksen[i][j]=0;
            eksenycd[i][j]=0;
        }
    }
}
```

```
/* Doğrusal ölçekler üzerinde indis bulan fonksiyon */
```

```
int indisbul(par1,eks)
```

```
{
    int par1,eks;
    {
        int il;
        int geri;
        int kon01;
        kon01=1;
        il=1;
        while(kon01==1)
        {
            il=il+1;
            if (il>say[eks]) kon01=0;
            if ( (kon01!=0) && (par1<=eksen[eks][il]) )
            {
                geri=il-1;
                kon01=0;
            }
        }
        return(geri);
    }
}
```

/* Grid kütük yapısını bölmek için kullanılan fonksiyon */

```
int gridbol(bol,ind1,ind2)
int bol,ind1,ind2;
{
    int deg[BOYUT+1];
    int deg1,deg2,deg3,orta,fark;
    int i,k,j,l,n,nn,m;
    int bul1;
    int yenibol,dongu1,tut;
    int cs;
    int yeni1,yeni2,temp;
    int pagno1,pagno2,sakladongu,kontroldeg1;
    int oo1,oo2;
    int a,b;
    int c1;
    int sonuc;
```

```
deg1=gptr[ind1][ind2].son[bol];
deg2=gptr[ind1][ind2].ilk[bol];
fark=deg1-deg2;
orta=fark/2;
deg3=deg2+orta;    /* Bölünme Noktası */
```

```
sonuc=0;
```

/* Bölünme eksenine ait doğrusal ölçüğün güncellenmesi */

```
for(i=1;i<=say[bol];i++)
{
    eksenyed[bol][i]=eksen[bol][i];
}
```

```
for(k=1;k<=say[bol];k++)
{
    if (eksen[bol][k]==deg2)
    {
        bul1=k;
        k=say[bol]+1;
    }
}
```

```
eksen[bol][bul1+1]=deg3;
say[bol]=say[bol]+1;
```

```
for(i=bul1+2;i<=say[bol];i++)
{
    eksen[bol][i]=eksenyed[bol][i-1];
}
```

```
/* Döngü değişkenlerine değerlerin atanması */
```

```
yenibol=bol;
if (yenibol==1)
{
    yenibol++;
    dongu1=yenibol;
}
else /* (if (yenibol==2) */
{
    yenibol=1;
    dongu1=yenibol;
}
```

```
ool=say[bol]-2;
oo2=say[dongu1]-1;
```

```
/* Boyut sayısı kadar döngü içinde grid bloklarının güncellenmesi */
```

```
for(a=1; a<=ool; a++)
for(b=1; b<= oo2; b++)
{
    if (bol==2)
    {
        i=a;
        j=b;
        temp=i;
        i=j;
        j=temp;
    }
    else
    {
        i=a;
        j=b;
    }
}
```

```
/* G(i,j)'ye karşılık gelen grid bloğun bölünme eksenini üzerindeki bitim noktası bölünen
grid bloğunun sınırlarından küçük ise cs=1 */
```

```
if ( (gptryed[i][j].ilk[bol] <= deg2) &&
(gptryed[i][j].son[bol] <= deg2) ) cs=1;
```

```
/* G(i,j)'ye karşılık gelen grid bloğu ile bölünen grid bloğunun sınırları aynı ise cs=2 */
```

```
if ( (gptryed[i][j].ilk[bol] == deg2) &&
(gptryed[i][j].son[bol] == deg1) ) cs=2;
```

/* G(i,j)'ye karşılık gelen grid bloğunun bölünme eksenini üzerindeki başlangıç noktası bölünen grid bloğunun sınırlarından büyük ise cs=3 */

```
if ( (gptrycd[i][j].ilk[bol] >= deg1) &&
      (gptrycd[i][j].son[bol] >= deg1) ) cs=3;
```

```
switch(cs)
```

```
{
```

```
case 1 : {
```

```
    /* G(i,j) üzerinde işlem yapılmaz*/
```

```
    for(l=1;l<=BOYUT;l++)
```

```
    {
```

```
        gptry[i][j].ilk[l]=gptrycd[i][j].ilk[l];
```

```
        gptry[i][j].son[l]=gptrycd[i][j].son[l];
```

```
    }
```

```
    gptry[i][j].pageno=gptrycd[i][j].pageno;
```

```
    gptry[i][j].isaret=gptrycd[i][j].isaret;
```

```
    } /* case 1 */
```

```
    break;
```

```
case 2 : {
```

```
    /* Bölünme işlemi gerçekleşir ve ilgili grid bloklarının sınırları yeniden belirlenir */
```

```
    gptry[i][j].ilk[bol]=gptrycd[i][j].ilk[bol];
```

```
    gptry[i][j].son[bol]=deg3;
```

```
    tut=1;
```

```
    for (n=1;n<=BOYUT;n++)
```

```
    {
```

```
        if (n != bol)
```

```
        {
```

```
            deg[tut]=n;
```

```
            tut=tut+1;
```

```
        }
```

```
    }
```

```
    for(nn=1;nn<=tut;nn++)
```

```
    {
```

```
        gptry[i][j].ilk[deg[nn]]=gptrycd[i][j].ilk[deg[nn]];
```

```
        gptry[i][j].son[deg[nn]]=gptrycd[i][j].son[deg[nn]];
```

```
    }
```

```
    gptry[i][j].pageno=gptrycd[i][j].pageno;
```

```
    gptry[i][j].isaret=0;
```

```

if (bol == 1) { yeni1=i+1; yeni2=j; }
if (bol == 2) { yeni1=i; yeni2=j+1; }

gptr[yeni1][yeni2].ilk[bol]=deg3;
gptr[yeni1][yeni2].son[bol]=gptryed[i][j].son[bol];

for(nn=1;nn<tut;nn++)
{
gptr[yeni1][yeni2].ilk[deg[nn]]-gptryed[i][j].ilk[deg[nn]];
gptr[yeni1][yeni2].son[deg[nn]]=gptryed[i][j].son[deg[nn]];
}

gptr[yeni1][yeni2].isaret=0;
if (gptryed[i][j].isaret == 0)
{
gptr[yeni1][yeni2].pageno=gptryed[i][j].pageno;
}
else
{
ensonpage=ensonpage+1;
gptr[yeni1][yeni2].pageno=ensonpage;
/* Bölünen ve yeni atanan sayfalar arasında kayıt aktarmaları yapılan bölüm */
pagno1=gptryed[i][j].pageno;
pagno2=gptr[yeni1][yeni2].pageno;
sakladongu=pptr[pagno1].tupsay;

if (bol == 1) c1=2;
if (bol==2) c1=1;

for(l=1;l<=sakladongu;l++)
{
if ( ( pptr[pagno1].tuples[l][bol] > deg3 ) &&
( pptr[pagno1].tuples[l][c1] <= gptryed[i][j].son[c1] ) )

{
pptr[pagno2].tupsay=pptr[pagno2].tupsay+1;
pptr[pagno1].tupsay=pptr[pagno1].tupsay-1;
pptr[pagno2].tuples[pptr[pagno2].tupsay][1]=
pptr[pagno1].tuples[l][1];
pptr[pagno2].tuples[pptr[pagno2].tupsay][2]=
pptr[pagno1].tuples[l][2];

pptr[pagno1].tuples[l][1]=0;
pptr[pagno1].tuples[l][2]=0;
}
}
}

```

```

kontroldeg1=0;
for(m=1;m<=sakladongu;m++)
{
    if ( (pptr[pagno1].tuples[m][1] != 0) &&
        (pptr[pagno1].tuples[m][2] != 0) )

    {
        kontroldeg1++;
        yedtuples[kontroldeg1][1]=pptr[pagno1].tuples[m][1];
        yedtuples[kontroldeg1][2]=pptr[pagno1].tuples[m][2];

        pptr[pagno1].tuples[m][1]=0;
        pptr[pagno1].tuples[m][2]=0;
    }
}
for(m=1;m<=kontroldeg1;m++)
{
    pptr[pagno1].tuples[m][1]=yedtuples[m][1];
    pptr[pagno1].tuples[m][2]=yedtuples[m][2];
}

/* Kayıt aktarma işlemi bitimi */

    if (pptr[pagno1].tupsay==PAGESIZE)
    {
        /* Dönüş Değerleri */
        sonuc=1;
        saklaind1=i;
        saklaind2=j;
    }
} /* else */
} /* case 2 */
break;

case 3 : {

/* G(i,j) ötelenir */
    tut=1;
    for(n=1;n<=BOYUT;n++)
    {
        if (n != bol)
        {
            deg[tut]=n;
            tut=tut+1;
        }
    }
}

```

```

        if (bol==1) { yeni1=i+1;yeni2=j; }
        if (bol==2) { yeni1=i;yeni2=j+1; }
        gptr[yeni1][yeni2].ilk[bol]=gptried[i][j].ilk[bol];
        gptr[yeni1][yeni2].son[bol]=gptried[i][j].son[bol];

        for(nn = 1;nn<tut;nn++)
        {
            gptr[yeni1][yeni2].ilk[deg[nn]]=
                gptried[i][j].ilk[deg[nn]];
            gptr[yeni1][yeni2].son[deg[nn]]=
                gptried[i][j].son[deg[nn]];
        }
        gptr[yeni1][yeni2].pagno=gptried[i][j].pagno;
        gptr[yeni1][yeni2].isaret=gptried[i][j].isaret;
    } /* case 3 */
    break;

} /* end switch */

} /* for j */
} /* for i */
return(sonuc);
} /* Grid bölünme fonksiyonu sonu */

```

```

main()
{
    int yuzde;
    int yuzdehes;
    int i,j,k;
    int ii;
    int pagno;
    int donus;
    int ind1,ind2,t1,t2;
    int terim[BOYUT+1][100];
    int a,b,tek-say,toplam;
    int x1,x2,ysay,pkont,sayac1,zzzz;
    int n[5];
    int m[5];
    int pagenu[400];
    int pagenums[300];
    /* Grid oluşturma işlemi başlıyor */
    initgrid();
    initpage();
    initaxis();
    srand(32); /*unix de rastgele sayı üretimi için başlangıç değeri belirlenmesi */

```

```
for(i=1;i<=BOYUT;i++)
say[i]=2;
```

```
bol=0;
ensonpage=1;
donus=0;
```

```
for(i=1;i<=BOYUT;i++)
{
eksen[i][1]=0;
eksen[i][2]=16384;
}
```

```
for(i=1;i<=BOYUT;i++)
{
gptr[1][1].ilk[i]=0;
gptr[1][1].son[i]=16384;
}
gptr[1][1].pageno=ensonpage;
```

/* Tüm bileşen tanımlayıcılarının grid kütük yapısına yerleştirildiği döngü */

```
for(ii=1;ii<COMPSAY;ii++)
```

```
{
```

```
t1=rand()%16384; /* 0 ile 16384 arası rastgele sayı üret */
```

```
t2=rand()%16384; /* t1 ve t2 bileşen tanımlayıcısını oluşturan terimlerdir. Bu terimlerin
tamsayı değerler olduğu düşünülmüştür. */
```

```
for(i=2;i<=say[1];i++)
```

```
{
```

```
if (t1 <= eksen[1][i])
```

```
{
```

```
ind1=i-1;
```

```
i=say[1]+1;
```

```
}
```

```
}
```

```
for(i=2;i<=say[2];i++)
```

```
{
```

```
if (t2 <= eksen[2][i])
```

```
{
```

```
ind2=i-1;
```

```
i=say[2]+1;
```

```
}
```

```
}
```

```
pagno=gptr[ind1][ind2].pageno;
```

```

/* Bileşen tanımlayıcısının yerleşeceği disk sayfası bulundu */
/* Bu aşamada disk sayfasının doluluk durumu ve bölünme gerekip gerekmediği araştırılacak */
if (pptr[pagno].tupsay < PAGESIZE)
{
    /* Yeterli yer var */
    pptr[pagno].tupsay=pptr[pagno].tupsay+1;
    pptr[pagno].tuples[pptr[pagno].tupsay][1]=t1;
    pptr[pagno].tuples[pptr[pagno].tupsay][2]=t2;
}
if (pptr[pagno].tupsay ==PAGESIZE)
{
    /* Sayfa dolmuş durumda ve bölünmeli */
    gptr[ind1][ind2].isaret=1;
    /* grid yedekleme */
    for(i=1;i<=GRIDSAY;i++)
    {
        for(j=1;j<=GRIDSAY;j++)
        {
            gptried[i][j].pageno=gptr[i][j].pageno;
            gptried[i][j].isaret=gptr[i][j].isaret;

            for(k=1;k<=BOYUT;k++)
            {
                gptried[i][j].ilk[k]=gptr[i][j].ilk[k];
                gptried[i][j].son[k]=gptr[i][j].son[k];
            }
        }
    }
    /* grid yedeklemesi bitimi */

    /* Bölünme yöntemine göre bölünme sırası gelen eksenin belirlenmesi */

    bol=bol+1;
    if (bol > BOYUT) { bol=1; }
    donus=gridbol(bol,ind1,ind2);

    /* Bölünen grid bloğunda halen daha taşma olduğu sürece bölünme işlemi
    tekrarlanacaktır */
    while (donus==1)
    {
        bol=bol+1;
        if (bol > BOYUT) { bol=1; }
        gptr[saklaind1][saklaind2].isaret=1;
    }

```

```

/* grid yedekleme */
for(i=1;i<=GRIDSAY;i++)
{
    for(j=1;j<=GRIDSAY;j++)
    {
        gptryed[i][j].pageno=gptr[i][j].pageno;
        gptryed[i][j].isaret=gptr[i][j].isaret;
        for(k=1;k<=BOYUT;k++)
        {
            gptryed[i][j].ilk[k]=gptr[i][j].ilk[k];
            gptryed[i][j].son[k]=gptr[i][j].son[k];
        }
    }
}

/* grid yedeklemesi bitimi */

donus=gridbol(bol,saklaind1,saklaind2);
}
} /* if */

} /* ana döngü sonu */

/* Grid kütük yapısı oluşturma işlemi bitti */
/* Bu aşamadan sonra grid kütük yapısının performansını ölçme bölümü gelmektedir */

/* 2 Terimden oluşan istek tanımlayıcıları üretilecek */

toplam=0; /* Grid ile erişimde toplam sayfa sayısını tutan değişken*/
tek-say=0; /* Tek-nitelikli kütük yapısı ile erişimde toplam sayfa sayısını tutan değişken
*/

/* 25 istek tanımlayıcısı üretimi ve grid kütük yapısında aranması */
for(i=0;i<25;i++)
{
    a=1+rand()%2;
    terim[1][i]=30000;
    terim[2][i]=30000;

    switch(a)
    {
        case 1: {
            b=1+rand()%2;
            terim[b][i]=rand()%16384;
            if (b==1) {tek-say=tek-say+1;}
            else {tek-say=tek-say+KONST;} /* Anahtar olan nitelik belirlenmezse tek-nitelikli
erişimde tüm sayfalar aranacak */
            break;
        }
    }
}

```

case 2:

```
{
    terim[1][i]=rand()%16384;
    terim[2][i]=rand()%16384;
    tek-say=tek-say+1;
    break;
}
```

/* İstek tanımlayıcısını, oluşturulan grid kütük yapısında arayan ve arama işlemi sırasında erişilen disk sayfası sayısını sayan bölüm */

```
for(j=1;j<3;j++)
{
    if (terim[j][i]==30000)
    {
        m[j]=1;
        n[j]=say[j];
    }
    else
    {
        m[j]=indisbul(terim[j][i],j);
        n[j]=m[j]+1;
    }
}
```

```
sayac1=0;
ysay=0;
```

```
for(x1=m[1];x1<n[1];x1++)
{
    for(x2=m[2];x2<n[2];x2++)
    {
        pagenu[ysay]=gptr[x1][x2].pageno;
        ysay=ysay+1;
        pkont=0;
```

```
if (ysay !=1)
{
    for(zzzz=0;zzzz<ysay-1;zzzz++)
    {
        if (pagenu[zzzz]==pagenu[ysay-1]) pkont=1;
    }
}
```

```
if (pkont!=1)
{
    pagenums[sayac1]=gptr[x1][x2].pageno;
    sayac1=sayac1+1;
```

```
    toplam=toplam+1;
  }
  }/*fors*/
}

}/*query i */

printf("allocate edilen page  %d\n",ensonpage);
printf("grid kütük yapısı ile erişimde toplam sayfa sayısı=%d\n",toplam);
printf("tek-nitelikli kütük yapısı ile erişimde toplam sayfa sayısı=%d\n",tek-say);

} /* end of main */
```



Ek2. Üç Boyutlu Grid Kütük Yapısı Oluşturmak için C Dilinde Yazılan Program

```

/* 3 boyutlu grid kütük yapısı oluşturan program */
/* Bu programda kullanılan tüm yapılar 2 boyutlu grid kütük yapısı
   oluşturan programdaki ile aynı görevdedir */
/* Bu programdaki açıklama satırları ek-1'de verilen 2 boyutlu
   grid kütük yapısı oluşturan programla farklılık olan yerlere
   konulmuştur */

#include <stdio.h>
#include <stdlib.h>

#define BOYUT 3
#define PAGESIZE 50
#define PAGESAYISI 150
#define GRIDSAY 40
#define COMPSAY 100
#define CONST 2

struct gridstr
{
    int ilk[BOYUT+1];
    int son[BOYUT+1];
    int pageno;
    int isaret;
};

/* 3 boyutlu grid olduğu için grid bloklarını gösteren yapı da 3 boyutludur*/
struct gridstr gptr[GRIDSAY+1][GRIDSAY+1][GRIDSAY+1];
struct gridstr gptried[GRIDSAY+1][GRIDSAY+1][GRIDSAY+1];

struct pagestr
{
    int pageid;
    int tupsay;
    int tuples[PAGESIZE+1][BOYUT+1];
};
struct pagestr pptr[PAGESAYISI+1];

int eksen[BOYUT+1][41];
int eksenyed[BOYUT+1][41];
int bol,ensonpage,saklaind1,saklaind2,saklaind3;
int say[BOYUT+1];
int yedtuples[PAGESIZE+1][BOYUT+1];

```

```

void initgrid()
{
int i,j,k,l;

for(i=1;i<=GRIDSAY;i++)
{
for(j=1;j<=GRIDSAY;j++)
{
for(l=1;l<=GRIDSAY;l++)
{
gp[tr[i][j][l]].pageno=0;
gp[tr[i][j][l]].isaret=0;
gp[tryed[i][j][l]].pageno=0;
gp[tryed[i][j][l]].isaret=0;

for(k=1;k<=BOYUT;k++)
{
gp[tr[i][j][l]].ilk[k]=0;
gp[tr[i][j][l]].son[k]=0;
gp[tryed[i][j][l]].ilk[k]=0;
gp[tryed[i][j][l]].son[k]=0;
}
}
}
} /* end initgrid() */

```

```

void initpage()
{
int i,j,k;

for(i=1;i<=PAGESAYISI;i++)
{
pp[tr[i].pageid=i;
pp[tr[i].tupsay=0;
for(j=1;j<=PAGESIZE;j++)
for(k=1;k<=BOYUT;k++)
pp[tr[i].tuples[j][k]=0;
}
} /* end initpage() */

```

```

void initaxis()
{
    int i,j;
    for(i=1;i<=BOYUT;i++)
    {
        for(j=1;j<=40;j++)
        {
            eksen[i][j]=0;
            eksenyed[i][j]=0;
        }
    }
}

/* end initaxis() */

```

```

int indisbul(par1,eks)
int par1,eks;
{
    int il;
    int geri;

    for(il=2;il<=say[eks];il++)
    {
        if (par1 <= eksen[eks][il])
        {
            geri=il-1;
            il=say[eks]+1;
        }
    }
    return(geri);
} /* end of function*/

```

/* Bu fonksiyon algoritma olarak 2 boyutlu grid bölen fonksiyon
ile temelde aynıdır */

```

int gridbol(bol,ind1,ind2,ind3)
int bol,ind1,ind2,ind3;
{

    int deg[BOYUT+1];
    int deg1,deg2,deg3,orta,fark;
    int i,k,j,l,n,m,u;
    int bull;
    int yenibol,dongu1,dongu2,tut;
    int cs;

```

```

int yeni1,yeni2,yeni3,temp;
int pagno1,pagno2,sakladongu,kontroldeg1;
int oo1,oo2,oo3;
int a,b,c;
int c1,c2;
int sonuc;

```

```

deg1=gptr[ind1][ind2][ind3].son[bol];
deg2=gptr[ind1][ind2][ind3].ilk[bol];
fark=deg1-deg2;
orta=fark/2;
deg3=deg2+orta;

```

```

sonuc=0;

```

```

for(i=1;i<=say[bol];i++)
{
    eksenyed[bol][i]=eksen[bol][i];
}

```

```

for(k=1;k<=say[bol];k++)
{
    if (eksen[bol][k]==deg2)
    {
        bul1=k;
        k=say[bol]+1;
    }
}

```

```

eksen[bol][bul1+1]=deg3;
say[bol]=say[bol]+1;

```

```

for(i=bul1+2;i<=say[bol];i++)
{
    eksen[bol][i]=eksenyed[bol][i-1];
}
yenibol=bol;
switch(yenibol)
{
case(1) : {
    dongu1=2;
    dongu2=3;
    }
    break;
}

```

```

case(2) : {
    dongu1=1;
    dongu2=3;
}
break;

case(3) : {
    dongu1=1;
    dongu2=2;
}
break;

} /* end switch */

/* 3 boyutlu grid için döngü değişkenlerinin değerlerinin atanması*/

oo1=say[bol]-2;
oo2=say[dongu1]-1;
oo3=say[dongu2]-1;

for(a=1; a<=oo1; a++)
for(b=1; b<= oo2; b++)
for(c=1; c<= oo3; c++)
{
    switch(bol)
    {
        case(1) : {
            i=a;
            j=b;
            u=c;
        }
        break;
        case(2) : {
            i=b;
            j=a;
            u=c;
        }
        break;
        case(3) : {
            i=b;
            j=c;
            u=a;
        }
        break;
    }
} /* end switch */

```

```

if ( (gptryed[i][j][u].ilk[bol] <= deg2) &&
      (gptryed[i][j][u].son[bol] <= deg2) ) cs=1;

if ( (gptryed[i][j][u].ilk[bol] == deg2) &&
      (gptryed[i][j][u].son[bol] == deg1) ) cs=2;

if ( (gptryed[i][j][u].ilk[bol] >= deg1) &&
      (gptryed[i][j][u].son[bol] >= deg1) ) cs=3;

switch(cs)
{
case 1 : {

    for(l=1;l<=BOYUT;l++)
    {
        gptr[i][j][u].ilk[l]=gptryed[i][j][u].ilk[l];
        gptr[i][j][u].son[l]=gptryed[i][j][u].son[l];
    }
    gptr[i][j][u].pageno=gptryed[i][j][u].pageno;
    gptr[i][j][u].isaret=gptryed[i][j][u].isaret;
    } /* case 1 */
    break;

case 2 : {

    gptr[i][j][u].ilk[bol]=gptryed[i][j][u].ilk[bol];
    gptr[i][j][u].son[bol]=deg3;
    tut=tut-1;
    for (n=1;n<=BOYUT;n++)
    {
        if (n != bol)
        {
            deg[tut]=n;
            tut=tut+1;
        }
    }
    for(nn=1;nn<tut;nn++)
    {
        gptr[i][j][u].ilk[deg[nn]]=gptryed[i][j][u].ilk[deg[nn]];
        gptr[i][j][u].son[deg[nn]]=gptryed[i][j][u].son[deg[nn]];
    }

    gptr[i][j][u].pageno=gptryed[i][j][u].pageno;
    gptr[i][j][u].isaret=0;

```

```

if (bol == 1) { yeni1=i+1; yeni2=j; yeni3=u; }
if (bol == 2) { yeni1=i; yeni2=j+1; yeni3=u; }
if (bol == 3) { yeni1=i; yeni2=j ; yeni3=u+1;}

gptr[yeni1][yeni2][yeni3].ilk[bol]=deg3;
gptr[yeni1][yeni2][yeni3].son[bol]=gptrycd[i][j][u].son[bol];

for(nn=1;nn<tut;nn++)
{
gptr[yeni1][yeni2][yeni3].ilk[deg[nn]]=gptrycd[i][j][u].ilk[deg[nn]];
gptr[yeni1][yeni2][yeni3].son[deg[nn]]=gptrycd[i][j][u].son[deg[nn]];
}

gptr[yeni1][yeni2][yeni3].isaret=0;

if (gptrycd[i][j][u].isaret == 0)
{
gptr[yeni1][yeni2][yeni3].pageno=gptrycd[i][j][u].pageno;
}
else
{
ensonpage=ensonpage+1;
gptr[yeni1][yeni2][yeni3].pageno=ensonpage;
/* aktarmalar yapılacak */
pagno1=gptrycd[i][j][u].pageno;
pagno2=gptr[yeni1][yeni2][yeni3].pageno;
sakladongu=pptr[pagno1].tupsay;

if (bol==1) { c1=2; c2=3; }
if (bol==2) { c1=1; c2=3; }
if (bol==3) { c1=1; c2=2; }
for(l=1;l<=sakladongu;l++)
{
if ( ( pptr[pagno1].tuples[l][bol] > deg3 ) &&
( pptr[pagno1].tuples[l][c1] <= gptrycd[i][j][u].son[c1] ) &&
( pptr[pagno1].tuples[l][c2] <= gptrycd[i][j][u].son[c2] ) )
{
pptr[pagno2].tupsay=pptr[pagno2].tupsay+1;
pptr[pagno1].tupsay=pptr[pagno1].tupsay-1;
pptr[pagno2].tuples[pptr[pagno2].tupsay][1]=
pptr[pagno1].tuples[l][1];
pptr[pagno2].tuples[pptr[pagno2].tupsay][2]=
pptr[pagno1].tuples[l][2];
pptr[pagno1].tuples[l][1]=0;
pptr[pagno1].tuples[l][2]=0;
}
}
}

```

```

kontroldeg1=0;
for(m=1;m<=sakladongu;m++)
{
    if ( (pptr[pagno1].tuples[m][1] != 0) &&
        (pptr[pagno1].tuples[m][2] != 0) &&
        (pptr[pagno1].tuples[m][3] != 0) )
    {
        kontroldeg1++;
        yedtuples[kontroldeg1][1]=pptr[pagno1].tuples[m][1];
        yedtuples[kontroldeg1][2]=pptr[pagno1].tuples[m][2];
        yedtuples[kontroldeg1][3]=pptr[pagno1].tuples[m][3];
        pptr[pagno1].tuples[m][1]=0;
        pptr[pagno1].tuples[m][2]=0;
        pptr[pagno1].tuples[m][3]=0;
    }
}

for(m=1;m<=kontroldeg1;m++)
{
    pptr[pagno1].tuples[m][1]=yedtuples[m][1];
    pptr[pagno1].tuples[m][2]=yedtuples[m][2];
    pptr[pagno1].tuples[m][3]=yedtuples[m][3];
}

/* aktarma islemi bitimi */

if (pptr[pagno1].tupsay==PAGESIZE)
{
    sonuc=1;
    saklaind1=i;
    saklaind2=j;
    saklaind3=u;
}
} /* else */
} /* case 2 */
break;

case 3 : {
    tut=1;
    for(n=1;n<=BOYUT;n++)
    {
        if (n != bol)
        {
            deg[tut]=n;
            tut=tut+1;
        }
    }
}

```

```

if (bol==1) { yeni1=i+1;yeni2=j;yeni3=u; }
if (bol==2) { yeni1=i;yeni2=j+1;yeni3=u; }
if (bol==3) { yeni1=i;yeni2=j;yeni3=u+1; }

gptr[yeni1][yeni2][yeni3].ilk[bol]=gptryed[i][j][u].ilk[bol];
gptr[yeni1][yeni2][yeni3].son[bol]=gptrycd[i][j][u].son[bol];

    for(nn=1;nn<tut;nn++)
    {
        gptr[yeni1][yeni2][yeni3].ilk[deg[nn]]=
            gptrycd[i][j][u].ilk[deg[nn]];
        gptr[yeni1][yeni2][yeni3].son[deg[nn]]=
            gptrycd[i][j][u].son[deg[nn]];
    }
    gptr[yeni1][yeni2][yeni3].pageno=gptrycd[i][j][u].pageno;
    gptr[yeni1][yeni2][yeni3].isaret= gptrycd[i][j][u].isaret;
    } /* case 3 */
    break;

} /* end switch */

} /* end for */

return(sonuc);

} /* end gridbol */

main()
{
    int i,j,k,l;
    int ii;
    int pagno;
    int donus;

    int toplam,randeg1,randeg2,randeg3;
    int ind1,ind2,ind3,t1,t2,t3,zzzz,sayac1,ysay,pkont,x1,x2,x3;
    int a,b,c;
    int tsay;
    int n[BOYUT+1]; int m[BOYUT+1];
    int terim[BOYUT+1][300];
    int pagenums[4400]; int pagenu[4500];

    srand(31);
    toplam=0;

```

```

for(j=0;j<4400;j++)
{
pagenums[j]=0;pagenu[j]=0;
}

initgrid();
initpage();
initaxis();

for(i=1;i<=BOYUT;i++)
say[i]=2;

bol=0;
ensonpage=1;
donus=0;

for(i=1;i<=BOYUT;i++)
{
eksen[i][1]=0;
eksen[i][2]=32768;
}

for(i=1;i<=BOYUT;i++)
{
gptr[1][1][1].ilk[i]=0;
gptr[1][1][1].son[i]=32768;
}
gptr[1][1][1].pageno=ensonpage;

for(ii=1;ii<COMPSAY;ii++) /* ana dongu */
{
t1=rand()%32768;
t2=rand()%32768;
t3=rand()%32768;

for(i=2;i<=say[1];i++)
{

if (t1 <= eksen[1][i])
{
indl=i-1;
i=say[1]+1;
}
}
}

```

```

for(i=2;i<=say[2];i++)
{
    if (t2 <= eksen[2][i])
    {
        ind2=i-1;
        i=say[2]+1;
    }
}

for(i=2;i<=say[3];i++)
{
    if (t3 <= eksen[3][i])
    {
        ind3=i-1;
        i=say[3]+1;
    }
}

pagno=gptr[ind1][ind2][ind3].pageno;

if (pptr[pagno].tupsay < PAGESIZE)
{
    /* disk sayfasında yeterli yer var */
    pptr[pagno].tupsay=pptr[pagno].tupsay+1;
    pptr[pagno].tuples[pptr[pagno].tupsay][1]=t1;
    pptr[pagno].tuples[pptr[pagno].tupsay][2]=t2;
    pptr[pagno].tuples[pptr[pagno].tupsay][3]=t3;
}

if (pptr[pagno].tupsay ==PAGESIZE)
{
    /* disk sayfası dolmuş durumda ve grid bölünecek */
    gptr[ind1][ind2][ind3].isaret=1;

    /* grid yedekleme */
    for(i=1;i<=GRIDSAY;i++)
    {
        for(j=1;j<=GRIDSAY;j++)
        {
            for(l=1;l<=GRIDSAY;l++)
            {
                gptryed[i][j][l].pageno=gptr[i][j][l].pageno;
                gptryed[i][j][l].isaret=gptr[i][j][l].isaret;
            }
        }
    }
}

```

```

    for(k=1;k<=BOYUT;k++)
    {
        gptryed[i][j][l].ilk[k]=gptr[i][j][l].ilk[k];
        gptryed[i][j][l].son[k]=gptr[i][j][l].son[k];
    }
}
}
}
/* grid yedeklemesi bitimi */

bol=bol+1;
if (bol > BOYUT) { bol=1; }
donus=gridbol(bol,ind1,ind2,ind3);

while (donus==1)
{
    bol=bol+1;
    if (bol > BOYUT) { bol=1; }
    gptr[saklind1][saklind2][saklind3].isaret=1;

    /* grid yedekleme */

    for(i=1;i<=GRIDSAY;i++)
    {
        for(j=1;j<=GRIDSAY;j++)
        {
            for(l=1;l<=GRIDSAY;l++)
            {
                gptryed[i][j][l].pageno=gptr[i][j][l].pageno;
                gptryed[i][j][l].isaret=gptr[i][j][l].isaret;
                for(k=1;k<=BOYUT;k++)
                {
                    gptryed[i][j][l].ilk[k]=gptr[i][j][l].ilk[k];
                    gptryed[i][j][l].son[k]=gptr[i][j][l].son[k];
                }
            }
        }
    }
    /* grid yedeklemesi bitimi */

    donus=gridbol(bol,saklind1,saklind2,saklind3);
}
} /* end of if */

} /* end ana dongu */
printf("GRID OLUSTURMA ISLEMI BITTI\n");

```

```
/* Burada rastgele istek tanımlayıcıları üretilip grid kutük
   yapısına erişim yapılacak */
```

```
tsay=0;
/* 100 istek tanımlayıcı için */
for(i=0;i<100;i++)
{
    /* Rastgele istek tanımlayıcılarının üretilmesi */
    a=1+rand()%3;

    terim[1][i]=99999;
    terim[2][i]=99999;
    terim[3][i]=99999;
```

```
switch(a)
{
case 1: {
    b=1+rand()%3;

    terim[b][i]=rand()%32768;
    if (b==1) { tsay=tsay+1; }
    else { tsay=tsay+CONST; }
    break;
}

case 2: {
    b=1+rand()%3;
    do
    {
        c=1+rand()%3;
    }
    while(b==c);

    terim[b][i]=rand()%32768;
    terim[c][i]=rand()%32768;
    if ( (b==1) || (c==1) )
    {
        tsay=tsay+1;
    }
    else
    {
        tsay=tsay+CONST;
    }
    break;
}
```

```

case 3: {
    terim[1][i]=rand()%32768;
    terim[2][i]=rand()%32768;
    terim[3][i]=rand()%32768;
    tsay=tsay+1;
    break;
}
}

/* Arama işlemi başlıyor */
for(j=1;j<4;j++)
{
    if (terim[j][i]==99999)
    {
        m[j]=1;
        n[j]=say[j];
    }
    else
    {
        m[j]=indisbul(terim[j][i],j);
        n[j]=m[j]+1;
    }
}

sayac1=0;

ysay=0;

for(x1=m[1];x1<n[1];x1++)
{
    for(x2=m[2];x2<n[2];x2++)
    {
        for(x3=m[3];x3<n[3];x3++)
        {
            pagenu[ysay]=gptr[x1][x2][x3].pageno;
            ysay=ysay+1;
            pkont=0;
            if (ysay !=1)
            {
                for (zzzz=0;zzzz<ysay-1;zzzz++)
                {
                    if (pagenu[zzzz]==pagenu[ysay-1]) pkont=1;
                }
            }
        }
    }
}

```

```
if (pkont !=1)
{
    pagenums[sayac1]=gptr[x1][x2][x3].pageno;
    sayac1=sayac1+1;
    toplam=toplam+1;
}
} /* for */
} /* for */
} /* for */
} /* end anadongu */
printf("toplam=%d\n",toplam);
printf("ensonpage=%d\n",ensonpage);
printf("tek-say=%d\n",tek-say);
} /* end of main */
```



Ek3. Beş Boyutlu Grid Kütük Yapısı Kullanan Bileşen Geri Elde Etme Aracı Gerçekleştirimi İçin C Dilinde Yazılan Program

/* Yerel bir BYYG ortamında örnek bir kitaplık üzerinde çalışan
bileşen geri elde etme aracına ait program listesi */

```
#include <stdio.h>
#include <stdlib.h>
/* Kitaplıktaki bileşenleri sınıflandırmak için 5 nitelik
   kullanıldığı için 5 boyutlu grid kütük yapısı oluşturulmuştur */
#define BOYUT 5
#define PAGESIZE 50
#define PAGESAYISI 5
#define GRIDSAY 3

struct gridstr
{
    int ilk[BOYUT+1];
    int son[BOYUT+1];
    int pageno;
    int isaret;
};

struct gridstr gptr[GRIDSAY+1][GRIDSAY+1][GRIDSAY+1][GRIDSAY+1][GRIDSAY+1];
struct gridstr
gptryed[GRIDSAY+1][GRIDSAY+1][GRIDSAY+1][GRIDSAY+1][GRIDSAY+1];

struct pagestr
{
    int pageid;
    int tupsay;
    int tuples[PAGESIZE+1][BOYUT+2]; /* docptr tutmak için +2 */
};

struct pagestr pptr[PAGESAYISI+1];

int eksen[BOYUT+1][21];
int eksenyed[BOYUT+1][21];
int bol,ensonpage,saklained1,saklained2,saklained3,saklained4,saklained5;
int say[BOYUT+1];
int yedtuples[PAGESIZE+1][BOYUT+2];

/* Grid bloklarını, disk sayfalarını ve doğrusal ölçekleri ilkleyen
   fonksiyonlar aşağıda sıralanmaktadır */
void initgrid()
{
    int i,j,k,l,m,n;
```

```

for(i=1;i<=GRIDSAY;i++)
{
for(j=1;j<=GRIDSAY;j++)
{
for(l=1;l<=GRIDSAY;l++)
{
for(m=1;m<=GRIDSAY;m++)
{
for(n=1;n<=GRIDSAY;n++)
{
gp[tr[i][j][l][m][n].pageno=0;
gp[tr[i][j][l][m][n].isaret=0;
gp[tryed[i][j][l][m][n].pageno=0;
gp[tryed[i][j][l][m][n].isaret=0;

for(k=1;k<=BOYUT;k++)
{
gp[tr[i][j][l][m][n].ilk[k]=0;
gp[tr[i][j][l][m][n].son[k]=0;
gp[tryed[i][j][l][m][n].ilk[k]=0;
gp[tryed[i][j][l][m][n].son[k]=0;
}
}
}
}
}
}
}
} /* end initgrid() */

void initpage()
{
int i,j,k;

for(i=1;i<=PAGESAYISI;i++)
{
pp[tr[i].pageid=i;
pp[tr[i].tupsay=0;
for(j=1;j<=PAGESIZE;j++)
for(k=1;k<=BOYUT+1;k++)
pp[tr[i].tuples[j][k]=0;
}
} /* end initpage() */

```

```

void initaxis()
{
    int i,j;
    for(i=1;i<=BOYUT;i++)
    {
        for(j=1;j<=20;j++)
        {
            eksen[i][j]=0;
            eksenyed[i][j]=0;
        }
    }
}

/* end initaxis() */

/* Doğrusal ölçek üzerinde indis bulan fonksiyon */

int indisbul(par1,eks)
int par1,eks;
{
    int il;
    int geri;

    for(il=2;il<=say[eks];il++)
    {
        if (par1 <= eksen[eks][il])
        {
            geri=il-1;
            il=say[eks]+1;
        }
    }
    return(geri);
} /* end of function*/

/* 5 boyutlu Grid kütük yapısını bölen fonksiyon */
int gridbol(bol,ind1,ind2,ind3,ind4,ind5)
int bol,ind1,ind2,ind3,ind4,ind5;
{

    int deg[BOYUT+1];
    int deg1,deg2,deg3,orta,fark;
    int i,k,j,l,n,nn,m,u,t,v;
    int bul1;
    int yenibol,dongu1,dongu2,dongu3,dongu4,tut;
    int cs;
    int yeni1,yeni2,yeni3,yeni4,yeni5,temp;
    int pagno1,pagno2,sakladongu,kontroldeg1;
    int oo1,oo2,oo3,oo4,oo5;

```

```
int a,b,c,d,e;
int c1,c2,c3,c4;
int sonuc;
```

```
deg1=gptr[ind1][ind2][ind3][ind4][ind5].son[bol];
deg2=gptr[ind1][ind2][ind3][ind4][ind5].ilk[bol];
fark=deg1-deg2;
orta=fark/2;
deg3=deg2+orta;
```

```
sonuc=0;
```

```
for(i=1;i<=say[bol];i++)
{
    eksenyed[bol][i]=eksen[bol][i];
}
```

```
for(k=1;k<=say[bol];k++)
{
    if (eksen[bol][k]==deg2)
    {
        bul1=k;
        k=say[bol]+1;
    }
}
```

```
eksen[bol][bul1+1]=deg3;
say[bol]=say[bol]+1;
```

```
for(i=bul1+2;i<=say[bol];i++)
{
    eksen[bol][i]=eksenyed[bol][i-1];
}
```

```
yenibol=bol;
switch(yenibol)
```

```
{
    case(1) : {
        dongu1=2;
        dongu2=3;
        dongu3=4;
        dongu4=5;
    }
    break;
```

```

case(2) : {
    dongu1=1;
    dongu2=3;
    dongu3=4;
    dongu4=5;
}
break;
case(3) : {
    dongu1=1;
    dongu2=2;
    dongu3=4;
    dongu4=5;
}
break;
case(4) : {
    dongu1=1;
    dongu2=2;
    dongu3=3;
    dongu4=5;
}
break;
case(5) : {
    dongu1=1;
    dongu2=2;
    dongu3=3;
    dongu4=4;
}
break;

```

```

} /* end switch */

```

/* 5 boyutlu grid kütük yapısı oluşturulduğu için grid blokları
5 döngü içinde işlenmektedir. Döngü değişkenleri için değerler
aşağıda atanmıştır */

```

oo1=say[bol]-2;
oo2=say[dongu1]-1;
oo3=say[dongu2]-1;
oo4=say[dongu3]-1;
oo5=say[dongu4]-1;

for(a=1; a<=oo1; a++)
for(b=1; b<= oo2; b++)
for(c=1; c<= oo3; c++)
for(d=1; d<= oo4; d++)
for(e=1; e<= oo5; e++)
{

```

```
switch(bol)
{
case(1) : {
    i=a;
    j=b;
    u=c;
    t=d;
    v=e;
    }
    break;
case(2) : {
    i=b;
    j=a;
    u=c;
    t=d;
    v=e;
    }
    break;
case(3) : {
    i=b;
    j=c;
    u=a;
    t=d;
    v=e;
    }
    break;
case(4) : {
    i=b;
    j=c;
    u=d;
    t=a;
    v=e;
    }
    break;
case(5) : {
    i=b;
    j=c;
    u=d;
    t=e;
    v=a;
    }
    break;
} /* end switch */
```

```

/* İşlenmekte olan grid bloğunun sınırlarının
   bölünecek olan grid bloğu ile ilişkisinin
   belirlenmesi bölümü 2 ve 3 boyutlu grid oluşturma
   programlarındaki ile tamamen aynıdır */

if ( (gptried[i][j][u][t][v].ilk[bol] <= deg2) &&
      (gptried[i][j][u][t][v].son[bol] <= deg2) ) cs=1;

if ( (gptried[i][j][u][t][v].ilk[bol] == deg2) &&
      (gptried[i][j][u][t][v].son[bol] == deg1) ) cs=2;

if ( (gptried[i][j][u][t][v].ilk[bol] >= deg1) &&
      (gptried[i][j][u][t][v].son[bol] >= deg1) ) cs=3;

switch(cs)
{
case 1 : {

    for(l=1;l<=BOYUT;l++)
    {
        gptr[i][j][u][t][v].ilk[l]=gptried[i][j][u][t][v].ilk[l];
        gptr[i][j][u][t][v].son[l]=gptried[i][j][u][t][v].son[l];
    }
    gptr[i][j][u][t][v].pageno=gptried[i][j][u][t][v].pageno;
    gptr[i][j][u][t][v].isaret=gptried[i][j][u][t][v].isaret;
    } /* case 1 */
    break;

case 2 : {

    gptr[i][j][u][t][v].ilk[bol]=gptried[i][j][u][t][v].ilk[bol];
    gptr[i][j][u][t][v].son[bol]=deg3;
    tut=1;
    for (n=1;n<=BOYUT;n++)
    {
        if (n != bol)
        {
            deg[tut]=n;
            tut=tut+1;
        }
    }

    for(nn=1;nn<tut;nn++)
    {
        gptr[i][j][u][t][v].ilk[deg[nn]]=gptried[i][j][u][t][v].ilk[deg[nn]];
        gptr[i][j][u][t][v].son[deg[nn]]=gptried[i][j][u][t][v].son[deg[nn]];
    }

```

```
gptr[i][j][u][t][v].pageno=gptrycd[i][j][u][t][v].pageno;
gptr[i][j][u][t][v].isaret=0;
```

```
if (bol == 1) {yeni1=i+1;yeni2=j;yeni3=u;yeni4=t;yeni5=v;}
if (bol == 2) {yeni1=i;yeni2=j+1;yeni3=u;yeni4=t;yeni5=v;}
if (bol == 3) {yeni1=i;yeni2=j;yeni3=u+1;yeni4=t;yeni5=v;}
if (bol == 4) {yeni1=i;yeni2=j;yeni3=u;yeni4=t+1;yeni5=v;}
if (bol == 5) {yeni1=i;yeni2=j;yeni3=u;yeni4=t;yeni5=v+1;}

```

```
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].ilk[bol]=deg3;
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].son[bol]=gptrycd[i][j][u][t][v].son[bol];

```

```
for(nn=1;nn<tut;nn++)
{
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].ilk[deg[nn]]=gptrycd[i][j][u][t][v].ilk[deg[nn]];
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].son[deg[nn]]=gptrycd[i][j][u][t][v].son[deg[nn]];
}

```

```
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].isaret=0;
```

```
if (gptrycd[i][j][u][t][v].isaret == 0)
{
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].pageno=gptrycd[i][j][u][t][v].pageno;
}
else
{
ensonpage=ensonpage+1;
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].pageno=ensonpage;

```

```
/* aktarmalar yapılacak */

```

```
pagno1=gptrycd[i][j][u][t][v].pageno;
pagno2=gptr[yeni1][yeni2][yeni3][yeni4][yeni5].pageno;
sakladongu=pptr[pagno1].tupsay;
```

```
if (bol==1) { c1=2; c2=3; c3=4; c4=5; }
if (bol==2) { c1=1; c2=3; c3=4; c4=5; }
if (bol==3) { c1=1; c2=2; c3=4; c4=5; }
if (bol==4) { c1=1; c2=2; c3=3; c4=5; }
if (bol==5) { c1=1; c2=2; c3=3; c4=4; }

```

```

for(l=1;l<=sakladongu;l++)
{
if ( ( pptr[pagno1].tuples[l][bol] > deg3 ) &&
( pptr[pagno1].tuples[l][c1] <= gptryed[i][j][u][t][v].son[c1] ) &&
( pptr[pagno1].tuples[l][c2] <= gptryed[i][j][u][t][v].son[c2] ) &&
( pptr[pagno1].tuples[l][c3] <= gptryed[i][j][u][t][v].son[c3] ) &&
( pptr[pagno1].tuples[l][c4] <= gptryed[i][j][u][t][v].son[c4] ) )
{

pptr[pagno2].tupsay=pptr[pagno2].tupsay+1;
pptr[pagno1].tupsay=pptr[pagno1].tupsay-1;
pptr[pagno2].tuples[pptr[pagno2].tupsay][1]=
pptr[pagno1].tuples[l][1];
pptr[pagno2].tuples[pptr[pagno2].tupsay][2]=
pptr[pagno1].tuples[l][2];
pptr[pagno2].tuples[pptr[pagno2].tupsay][3]=
pptr[pagno1].tuples[l][3];
pptr[pagno2].tuples[pptr[pagno2].tupsay][4]=
pptr[pagno1].tuples[l][4];
pptr[pagno2].tuples[pptr[pagno2].tupsay][5]=
pptr[pagno1].tuples[l][5];
pptr[pagno2].tuples[pptr[pagno2].tupsay][6]=
pptr[pagno1].tuples[l][6];

pptr[pagno1].tuples[l][1]=0;
pptr[pagno1].tuples[l][2]=0;
pptr[pagno1].tuples[l][3]=0;
pptr[pagno1].tuples[l][4]=0;
pptr[pagno1].tuples[l][5]=0;
pptr[pagno1].tuples[l][6]=0;

}
}

kontroldeg1=0;
for(m=1;m<=sakladongu;m++)
{
if ( ( pptr[pagno1].tuples[m][1] != 0 ) &&
( pptr[pagno1].tuples[m][2] != 0 ) &&
( pptr[pagno1].tuples[m][3] != 0 ) &&
( pptr[pagno1].tuples[m][4] != 0 ) &&
( pptr[pagno1].tuples[m][5] != 0 ) )

{
kontroldeg1++;
yedtuples[kontroldeg1][1]=pptr[pagno1].tuples[m][1];

```

```

yedtuples[kontroldeg1][2]=pptr[pagno1].tuples[m][2];
yedtuples[kontroldeg1][3]=pptr[pagno1].tuples[m][3];
yedtuples[kontroldeg1][4]=pptr[pagno1].tuples[m][4];
yedtuples[kontroldeg1][5]=pptr[pagno1].tuples[m][5];
yedtuples[kontroldeg1][6]=pptr[pagno1].tuples[m][6];

```

```

pptr[pagno1].tuples[m][1]=0;
pptr[pagno1].tuples[m][2]=0;
pptr[pagno1].tuples[m][3]=0;
pptr[pagno1].tuples[m][4]=0;
pptr[pagno1].tuples[m][5]=0;
pptr[pagno1].tuples[m][6]=0;

```

```

}
}

```

```

for(m=1;m<=kontroldeg1;m++)

```

```

{
    pptr[pagno1].tuples[m][1]=yedtuples[m][1];
    pptr[pagno1].tuples[m][2]=yedtuples[m][2];
    pptr[pagno1].tuples[m][3]=yedtuples[m][3];
    pptr[pagno1].tuples[m][4]=yedtuples[m][4];
    pptr[pagno1].tuples[m][5]=yedtuples[m][5];
    pptr[pagno1].tuples[m][6]=yedtuples[m][6];

```

```

}

```

```

/* aktarma islemi bitimi */

```

```

if (pptr[pagno1].tupsay==PAGESIZE)

```

```

{
    sonuc=1;
    saklaind1=i;
    saklaind2=j;
    saklaind3=u;
    saklaind4=t;
    saklaind5=v;
}

```

```

} /* else */

```

```

} /* case 2 */

```

```

break;

```

```

case 3 : {

    tut=1;
    for(n=1;n<=BOYUT;n++)
    {
        if (n != bol)
        {
            deg[tut]=n;
            tut=tut+1;
        }
    }

    if (bol == 1) {yeni1=i+1;yeni2=j;yeni3=u;yeni4=t;yeni5=v;}
    if (bol == 2) {yeni1=i;yeni2=j+1;yeni3=u;yeni4=t;yeni5=v;}
    if (bol == 3) {yeni1=i;yeni2=j;yeni3=u+1;yeni4=t;yeni5=v;}
    if (bol == 4) {yeni1=i;yeni2=j;yeni3=u;yeni4=t+1;yeni5=v;}
    if (bol == 5) {yeni1=i;yeni2=j;yeni3=u;yeni4=t;yeni5=v+1;}

    gptr[yeni1][yeni2][yeni3][yeni4][yeni5].ilk[bol]=gptryed[i][j][u][t][v].ilk[bol];
    gptr[yeni1][yeni2][yeni3][yeni4][yeni5].son[bol]=gptryed[i][j][u][t][v].son[bol];

    for(nn=1;nn<tut;nn++)
    {
        gptr[yeni1][yeni2][yeni3][yeni4][yeni5].ilk[deg[nn]]=
            gptryed[i][j][u][t][v].ilk[deg[nn]];
        gptr[yeni1][yeni2][yeni3][yeni4][yeni5].son[deg[nn]]=
            gptryed[i][j][u][t][v].son[deg[nn]];
    }

    gptr[yeni1][yeni2][yeni3][yeni4][yeni5].pageno=gptryed[i][j][u][t][v].pageno;
    gptr[yeni1][yeni2][yeni3][yeni4][yeni5].isaret=gptryed[i][j][u][t][v].isaret;
    } /* case 3 */
    break;

} /* end switch */

} /* end for */

return(sonuc);

} /* Gridbol fonksiyonu sonu */

```

```

main()
{
FILE *file;
FILE *file1;

FILE *kut1;

long offset, reclen, recnum;
long yer;

int kutukbul, kutukyer, kontrolcik, kontrolcik1;
int i, j, k, l, mm, nn, qq, ll;
int ii;
int pagno;
int donus;
int sayiat, kk, kk1;
int toplam;
int t1, t2, t3, t4, t5, t6, zzzz, sayac1, ysay, pkont, x1, x2, x3, x4, x5;
int a, b, c, ana, findit, sayac2;

int ind1, ind2, ind3, ind4, ind5, sec1, sec2, sec3, sec4, sec5;

int sarray[10];
int farray[10];
int n[BOYUT+1]; int m[BOYUT+1];
int terim[BOYUT+1][50];
int pagenums[100]; int pagenu[100];
char kay[80];
char name[15];
char ch[3];
char *facet1[6];
char *facet2[10];
char *facet3[25];
char *facet4[10];
char *facet5[5];
char param[3000];
char *paramyed;
/* Boyutlara terimlerin atandığı bölüm */
/* İlk boyut bileşen türünü belirlemektedir */
facet1[1]="ANALIZ";
facet1[2]="TASARIM";
facet1[3]="KOD";
facet1[4]="BELGELEMELER";
facet1[5]="SINAMA-SENARYOLARI";

```

/* İkinci boyut bileşenlere ait soyutlamaları belirlemektedir */

```
facet2[1]="DOGRUSAL-VERI-YAPILARI";
facet2[2]="DOGRUSAL-OLMAYAN-VERI-YAPILARI";
facet2[3]="ALGORITMALAR";
facet2[4]="COMPLEX-SAYILAR";
facet2[5]="MATRISLER";
```

/* Üçüncü boyut bileşenlerin sunduğu çeşitli işlemleri belirlemektedir */

```
facet3[1]="CARPMA";
facet3[2]="TOPLAMA";
facet3[3]="CIKARMA";
facet3[4]="BOLME";
facet3[5]="IKILI-ARAMA";
facet3[6]="SIRADAN-ARAMA";
facet3[7]="KABARCIK-SIRALAMA";
facet3[8]="SECICI-SIRALAMA";
facet3[9]="KARSILASTIRMA";
facet3[10]="INORDER-DOLAS";
facet3[11]="POSTORDER-DOLAS";
facet3[12]="PREORDER-DOLAS";
facet3[13]="ELEMEN-EKLE";
facet3[14]="ELEMEN-CIKAR";
facet3[15]="ELEMEN-DEGISTIR";
facet3[16]="ELEMEN-ELDE-ET";
facet3[17]="RASTGELE-SAYI-URET";
```

/* Dördüncü boyut bileşenin üzerinde işlem yaptığı veri yapıları veya veri tiplerini belirlemektedir */

```
facet4[1]="AGAC";
facet4[2]="YIGIT";
facet4[3]="KUYRUK";
facet4[4]="LISTE";
facet4[5]="TAMSAYI";
facet4[6]="KARAKTER";
```

/* Beşinci boyut bileşenin çalışma ortamını belirlemektedir */

```
facet5[1]="SADECE-DOS";
facet5[2]="SADECE-UNIX";
facet5[3]="DOS-ve-UNIX";
```

```
for(i=0;i<10;i++)
{
sarray[i]=0;
farray[i]=0;
}
```

```

toplam=0;
for(j=0;j<100;j++)
{
pagenums[j]=0;pagenu[j]=0;
}

/* Grid oluşturma işlemi başlangıcı */
initgrid();
initpage();
initaxis();

for(i=1;i<=BOYUT;i++)
say[i]=2;

bol=0;
ensonpage=1;
donus=0;

for(i=1;i<=BOYUT;i++)
{
eksen[i][1]=0;
eksen[i][2]=32;
}

for(i=1;i<=BOYUT;i++)
{
gptr[1][1][1][1][1][1].ilk[i]=0;
gptr[1][1][1][1][1][1].son[i]=32;
}
gptr[1][1][1][1][1][1].pageno=ensonpage;

/* Bilgi.dat isimli kütük kitaplıktaki bileşenlere ait
tanımlayıcıları içermektedir */

kutl=fopen("bilgi.dat","r");
for(ii=1;ii<61;ii++) /* ana dongu */
{

fscanf(kutl,"%d\n",&t1);
fscanf(kutl,"%d\n",&t2);
fscanf(kutl,"%d\n",&t3);
fscanf(kutl,"%d\n",&t4);
fscanf(kutl,"%d\n",&t5);
fscanf(kutl,"%d\n",&t6);

```

```
for(i=2;i<=say[1];i++)
{
    if (t1 <= eksen[1][i])
    {
        ind1=i-1;
        i=say[1]+1;
    }
}
```

```
for(i=2;i<=say[2];i++)
{
    if (t2 <= eksen[2][i])
    {
        ind2=i-1;
        i=say[2]+1;
    }
}
```

```
for(i=2;i<=say[3];i++)
{
    if (t3 <= eksen[3][i])
    {
        ind3=i-1;
        i=say[3]+1;
    }
}
```

```
for(i=2;i<=say[4];i++)
{
    if (t4 <= eksen[4][i])
    {
        ind4=i-1;
        i=say[4]+1;
    }
}
```

```
for(i=2;i<=say[5];i++)
{
    if (t5 <= eksen[5][i])
    {
        ind5=i-1;
        i=say[5]+1;
    }
}
```

```
/* Bileşen tanımlayıcının içinde yer alması gereken grid
   bloğuna ulaşarak, bu bloğun gösterdiği disk sayfası
   elde ediliyor */
```

```
pagno=gptr[ind1][ind2][ind3][ind4][ind5].pageno;
```

```
if (pptr[pagno].tupsay < PAGESIZE)
```

```
{
    /* Disk sayfasında yeterli yer var */
    pptr[pagno].tupsay=pptr[pagno].tupsay+1;
    pptr[pagno].tuples[pptr[pagno].tupsay][1]=t1;
    pptr[pagno].tuples[pptr[pagno].tupsay][2]=t2;
    pptr[pagno].tuples[pptr[pagno].tupsay][3]=t3;
    pptr[pagno].tuples[pptr[pagno].tupsay][4]=t4;
    pptr[pagno].tuples[pptr[pagno].tupsay][5]=t5;
    pptr[pagno].tuples[pptr[pagno].tupsay][6]=t6;
}
```

```
if (pptr[pagno].tupsay ==PAGESIZE)
```

```
{
    /* Disk sayfası dolmuş durumda ve bölünecek */
```

```
    gptr[ind1][ind2][ind3][ind4][ind5].isaret=1;
```

```
    /* grid yedekleme */
```

```
    for(i=1;i<=GRIDSAY;i++)
```

```
    {
        for(j=1;j<=GRIDSAY;j++)
```

```
        {
            for(l=1;l<=GRIDSAY;l++)
```

```
            {
                for(mm=1; mm<=GRIDSAY;mm++)
```

```
                {
                    for(nn=1; nn<=GRIDSAY;nn++)
```

```
                    {
                        gptried[i][j][l][mm][nn].pageno=gptr[i][j][l][mm][nn].pageno;
```

```
                        gptried[i][j][l][mm][nn].isaret=gptr[i][j][l][mm][nn].isaret;
```

```
                        for(k=1;k<=BOYUT;k++)
```

```
                        {
                            gptried[i][j][l][mm][nn].ilk[k]=gptr[i][j][l][mm][nn].ilk[k];
                            gptried[i][j][l][mm][nn].son[k]=gptr[i][j][l][mm][nn].son[k];
                        }
                    }
                }
            }
        }
    }
}
```

```
/* grid yedeklemesi bitimi */
```

```

bol=bol+1;
if (bol > BOYUT) { bol=1; }
donus=gridbol(bol,ind1,ind2,ind3,ind4,ind5);

while (donus==1)
{
bol=bol+1;
if (bol > BOYUT) { bol=1; }
gptr[saklaind1][saklaind2][saklaind3][saklaind4][saklaind5].isaret=1;

/* grid yedekleme */
for(i=1;i<=GRIDSAY;i++)
{
for(j=1;j<=GRIDSAY;j++)
{
for(l=1;l<=GRIDSAY;l++)
{
for(mm=1; mm<=GRIDSAY;mm++)
{
for(nn=1; nn<=GRIDSAY;nn++)
{
gptryed[i][j][l][mm][nn].pageno=gptr[i][j][l][mm][nn].pageno;
gptryed[i][j][l][mm][nn].isaret=gptr[i][j][l][mm][nn].isaret;
for(k=1;k<=BOYUT;k++)
{
gptryed[i][j][l][mm][nn].ilk[k]=gptr[i][j][l][mm][nn].ilk[k];
gptryed[i][j][l][mm][nn].son[k]=gptr[i][j][l][mm][nn].son[k];
}
}
}
}
}
}
}

/* grid yedeklemesi bitimi */

donus=gridbol(bol,saklaind1,saklaind2,saklaind3,saklaind4,saklaind5);
}

} /* end if */

} /* end ana dongu */

fclose(kut1);

printf("GRID OLUSTURMA ISLEMI BITTI\n");
/* Grid oluřturma iřlemi bitimi */

```

/* Bu bölüm kullanıcıların istek tanımlayıcılarını oluşturduğu
ve sonuçları elde ettiği bölümdür */

```
kontrolcik1=1;
kontrolcik=1;
```

```
while (kontrolcik1==1)
{
    kontrolcik = 1;
    while (kontrolcik==1)
    {
        printf("-----SORGU OLUSTURMA BOLUMU-----\n");
        printf("Boyut Icerikleri Gormek ve Terim Secmek Icin Bir Tusa Basiniz\n");
```

```
        printf("BIRINCI BOYUT ICERIGI\n");
        printf("-----\n");
        printf("Secmek Istediginiz Terimin Numarasini Giriniz...\n");
        printf("Terim Secmek Istemiyorsaniz 9999 Giriniz...\n");
        for(j=1;j<=5;j++)
        {
            printf("%d-%s\n",j,facet1[j]);
        }
        printf("Seciminiz?");
        scanf("%d",&sec1);
        while (sec1 >5 && sec1 <9999)
        {
            printf("1-5 Arasi Sayi Giriniz\n");
            scanf("%d",&sec1);
        }
```

```
        printf("IKINCI BOYUT ICERIGI\n");
        printf("-----\n");
        printf("Secmek Istediginiz Terimin Numarasini Giriniz...\n");
        printf("Terim Secmek Istemiyorsaniz 9999 Giriniz...\n");
        for(j=1;j<=5;j++)
        {
            printf("%d-%s\n",j,facet2[j]);
        }
        printf("Seciminiz?");
        scanf("%d",&sec2);
        while (sec2>5 && sec2<9999)
        {
            printf("1-5 Arasi Sayi Giriniz\n");
            scanf("%d",&sec2);
        }
```

```

printf("UCUNCU BOYUT ICERIGI\n");
printf("-----\n");
printf("Secmek Istediginiz Terimin Numarasini Giriniz...\n");
printf("Terim Secmek Istemiyorsaniz 9999 Giriniz...\n");
for(j=1;j<=17;j++)
{
printf("%d-%s\n",j,facet3[j]);
}
printf("Seciminiz?");
scanf("%d",&sec3);
while (sec3>17 && sec3<9999)
{
printf("1-17 Arasi Sayi Giriniz\n");
scanf("%d",&sec3);
}

```

```

printf("DORDUNCU BOYUT ICERIGI\n");
printf("-----\n");
printf("Secmek Istediginiz Terimin Numarasini Giriniz...\n");
printf("Terim Secmek Istemiyorsaniz 9999 Giriniz...\n");
for(j=1;j<=6;j++)
{
printf("%d-%s\n",j,facet4[j]);
}
printf("Seciminiz?");
scanf("%d",&sec4);
while (sec4>6 && sec4<9999)
{
printf("1-6 Arasi Sayi Giriniz\n");
scanf("%d",&sec4);
}

```

```

printf("BESINCI BOYUT ICERIGI\n");
printf("-----\n");
printf("Secmek Istediginiz Terimin Numarasini Giriniz...\n");
printf("Terim Secmek Istemiyorsaniz 9999 Giriniz...\n");
for(j=1;j<=3;j++)
{
printf("%d-%s\n",j,facet5[j]);
}
printf("Seciminiz?");
scanf("%d",&sec5);

```

```

while (sec5>3 && sec5<9999)
{
    printf("1-3 Arasi Sayi Giriniz\n");
    scanf("%d",&sec5);
}

```

```

printf("-----OLUSTURULAN ISTEK TANIMLAYICI-----\n");
printf(" < ");

```

```

if (sec1==9999) { printf("* "); }
    else { printf("%s ",facet1[sec1]); }
if (sec2==9999) { printf("* "); }
    else { printf("%s ",facet2[sec2]); }

```

```

if (sec3==9999) { printf("* "); }
    else { printf("%s ",facet3[sec3]); }
if (sec4==9999) { printf("* "); }
    else { printf("%s ",facet4[sec4]); }

```

```

if (sec5==9999) { printf("*"); }
    else { printf("%s ",facet5[sec5]); }

```

```

printf(" >\n");

```

```

printf("\n");

```

```

if (sec1==9999 && sec2==9999 && sec3==9999 && sec4==9999 && sec5==9999)

```

```

{
    printf("En az bir terim belirleyiniz!\n");
}

```

```

else

```

```

{

```

```

    printf("Sorguyu Onayliyor musunuz? E/H\n");

```

```

    printf("Onaylamak icin e veya E giriniz\n");

```

```

    scanf("%s",ch);

```

```

    if (ch[0] == 'e' || ch[0] == 'E') kontrolcik=0;

```

```

}

```

```

}

```

```

/* Oluşturulan istek tanımlayıcısı terim dizisinde saklanarak
   eşleme işlemi başlıyor */

```

```

terim[1][0]=sec1;

```

```

terim[2][0]=sec2;

```

```

terim[3][0]=sec3;

```

```

terim[4][0]=sec4;

```

```

terim[5][0]=sec5;

```

```

for(j=1;j<6;j++)
{
    if (terim[j][0]==9999)
    {
        m[j]=1;
        n[j]=say[j];
    }
    else
    {
        m[j]=indisbul(terim[j][0],j);
        n[j]=m[j]+1;
    }
}

sayac1=0;
ysay=0;
for(x1=m[1];x1<n[1];x1++)
{
    for(x2=m[2];x2<n[2];x2++)
    {
        for(x3=m[3];x3<n[3];x3++)
        {
            for(x4=m[4];x4<n[4];x4++)
            {
                for(x5=m[5];x5<n[5];x5++)
                {
                    pagenu[ysay]=gptr[x1][x2][x3][x4][x5].pageno;
                    ysay=ysay+1;
                    pkont=0;
                    if (ysay !=1)
                    {
                        for (zzzz=0;zzzz<ysay-1;zzzz++)
                        {
                            if (pagenu[zzzz]==pagenu[ysay-1]) pkont=1;
                        }
                    }
                    if (pkont !=1)
                    {
                        pagenums[sayac1]=gptr[x1][x2][x3][x4][x5].pageno;
                        sayac1=sayac1+1;
                        toplam=toplam+1;
                    }
                } /* for */
            } /* for */
        } /* end for */
    }
}

```

```

sayac2=0;
for(ana=0;ana<sayac1;ana++)
{
for(i=1;i<=PAGESIZE;i++)
{
findit=1;
for(j=1;j<=BOYUT;j++)
{
if ( (terim[j][0] != 9999) &&
(pptr[pagenums[ana]].tuples[i][j]!=terim[j][0]) ) findit=0;
}
if (findit==1)
{
sayac2++;
sarray[sayac2]=pptr[pagenums[ana]].tuples[i][6];
farray[sayac2]=pptr[pagenums[ana]].tuples[i][1];
}
}
}
if (sayac2==0) { printf("Istenen Ozellikte Bilesen Bulunamadi\n"); }
else
{
printf("%d adet bilesen bulundu\n",sayac2);

printf("bilesenleri gormek icin bir tusa basiniz\n");
scanf("%s",ch);
}

nn=0;
for(i=1;i<=sayac2;i++)
{
printf("-----%d. BILESEN-----\n",i);
kutukbul=farray[i];
if (kutukbul==1) file=fopen("ANALIZ.DAT","r");
if (kutukbul==2) file=fopen("TASARIM.DAT","r");
if (kutukbul==3) file=fopen("KODLAR.DAT","r");
if (kutukbul==4) file=fopen("DOCLAR.DAT","r");
if (kutukbul==5) file=fopen("TESTSEN.DAT","r");
/* Bileşenin türüne göre ilgili kütüğe ulaşıldı */
/* Bileşen tanımlayıcısından elde edilen işaretçi
gözününe alınarak bileşen ilgili kütükten bulunur */
recnum=sarray[i];
yer=recnum;
etiket2:
/* Bileşenler kütükte '@' kontrol karakteri ile biten satırlar
halinde yer almaktadır. Kontrol karakterleri ayıklanarak
bileşenin içeriği elde edilir */

```

```

fscanf(file,"%s",kay);
if (kay[0]=='@' && kay[1]!='@')
{
    fscanf(file,"%d",&kutukyer);
    if (kutukyer==yer)
    {
        goto etiket1;
    }
    else
    {
        goto etiket2;
    }
}
else
{
    goto etiket2;
}
etiket1:
/* Bileşenin içeriği ekrana yazdırılır */
fscanf(file,"%s",kay);
while (kay[0]!='@' && kay[1]!='@')
{
    if (kay[0]!='%')
    {
        printf("%s ",kay);

        mm=0;
        while (kay[mm]!='\0')
        {
            mm=mm+1;
        }

        for(qq=0;qq<mm;qq++)
        {
            param[nn]=kay[qq];
            nn=nn+1;
        }
        param[nn]=' ';
        nn=nn+1;
    }
    else
    {
        printf("\n");
        param[nn]='@';
        nn=nn+1;
    }
}

```

```

    fscanf(file,"%s",kay);
}

/* Elde edilen bileşenlerin daha sonra kullanılmak üzere
   bir kütükte saklandığı bölüm */
if (i==sayac2)
{
    printf("Bileseni kutuge yazmak icin E Tusuna Basiniz\n");

    scanf("%s",ch);
    if (ch[0]=='e')
    {
        printf("Bilesenin saklanacagi kutuk adini giriniz\n");
        scanf("%s",name);

        file1=fopen(name,"w");

        mm=0;
        while (param[mm] != '\0')
        {
            mm=mm+1;
        }
        for(nn=0;nn<mm;nn++)
        {
            if ( param[nn] != '@' )
            {
                fprintf(file1,"%c",param[nn]);
            }
            else
            {
                fprintf(file1,"\n");
            }
        }

        fclose(file1);
    } /* if e */
} /* if i==sayac2 */

fclose(file);

}

printf("\n");
printf("\n");

printf("Sorgulamaya Son Icin H Tusuna Basiniz\n");
printf("Sorgulamaya Devam Etmek Icin E Tusuna Basiniz");

```

```
scanf("%s",ch);  
if (ch[0]=='h' || ch[0]=='H') kontrolcikl=0;  
  
} /* while */  
  
} /* end of main */
```



Ek4. Bileşen Geri Elde Etme Aracının DEC-ACA Nesne Sunucu (Object-Broker) ile İstemci / Sunucu Şeklinde Modellenmesi için Yazılan Programlar

/ Bu program önerilen geniş ölçekli tekrar kullanım mimarisi içindeki ACA istemci programdır. İstemci program kullanıcıdan aldığı istek tanımlayıcılarını işletilmek üzere sunucu programa iletmektedir */*

```
#include <stdio.h>
#include <stdlib.h>

/* ACA istemci Program için "Include" dosyaları */
#include <acasmgs.h>
#include <acasdef.h>
#include <acasapi.h>

#ifdef vms
#define EXIT_STATUS( s ) \
    exit(s);
#else
#define EXIT_STATUS( s ) \
    if (s == ACAS_SUCCESS) \
        exit(0); \
    else \
        exit(1);
#endif

/* Global Tanımlamalar */
ACASStatus status = ACAS_SUCCESS, free_status = ACAS_SUCCESS;
ACASContextHandle ctxhnd = NULL;
ACASInstanceHandle instance = NULL, out_handle = NULL;
ACASArgList arglist = NULL;
ACASInt arg_count;
ACASErrorList apperrorlist = NULL;
ACASErrorList inverrorlist = NULL;

/* Fonksiyon Prototipleri */
void PrintACASErrors();
void Kapat(void);
void Request(void);

void main(void)
{
    /* Sınıf adına ait "instance handle" yaratılması */
    status = ACAS_CreateInstanceHandle((ACASStringPtr)NULL,
        "ACAS_LSR",
```

```

        (ACASStringPtr)NULL,
        (ACASStringPtr)NULL,
        (ACASInt)0,
        &instance,
        (ACASErrorList *)NULL);

/* Standart Çevresel Nesneye (Context Object) ait "handle" elde edilir */
if (status == ACAS_SUCCESS)
    status = ACAS_GetCtxHandle (&ctxhnd, (ACASErrorList *)NULL);
Request();
} /* Main */

/* ACA'ya İlişkin Hataları Ekrana Yazdıran Fonksiyon */
void PrintACASErrors( errorlist )
ACASErrorList errorlist;
{
    if (errorlist != 0)
    {
        ACASStatus sts;
        ACASInt errorcount, buf_len;
        ACASStringPtr buf_ptr;

        errorcount = errorlist [ 0 ] .Count;
        buf_len = 256 * errorcount;
        buf_ptr = (ACASStringPtr) malloc( buf_len );
        sts = ACAS_FormatACASErrors(
            errorlist, errorcount,
            buf_len, buf_ptr, 0, "", NULL );
        if (sts != ACAS_SUCCESS)
            printf("error calling ACAS_FormatACASErrors: %u\n", sts );
        else
            printf( "%s", buf_ptr );
        free( buf_ptr );
    }
} /* End PrintACASErrors */

/* Metod-sunucu'yu (method-server) serbest bırakan fonksiyon */
void Kapat(void)
{
    free_status = ACAS_FreeInstanceHandle(out_handle,
        (ACASFlags)NULL,
        (ACASErrorList *)NULL);
    ACAS_FreeInstanceHandle(instance,
        (ACASFlags) 0,
        (ACASErrorList *)NULL);
    ACAS_FreeCtxHandle (ctxhnd, (ACASErrorList *)NULL);
}

```

```

if (status == ACAS_SUCCESS && free_status != ACAS_SUCCESS)
{
    EXIT_STATUS(free_status);
}
else
{
    EXIT_STATUS(status);
}
exit();
} /* End Kapat */

```

```

void Request(void)
{
    ACASInt kontrolcik1,kontrolcik;

    FILE *file1;

    ACASInt i,j,mm,nn;
    ACASInt sec1,sec2,sec3,sec4,sec5;

```

```

    ACASChar kay[80];
    ACASChar name[15];
    ACASChar ch[3];
    ACASChar *facet1[6];
    ACASChar *facet2[10];
    ACASChar *facet3[25];
    ACASChar *facet4[10];
    ACASChar *facet5[5];
    ACASChar param[500];
    ACASChar *param1;
    ACASChar *paramyed;

```

```

/* Boyutlara ait nitelikler için terimlerin belirlenmesi */
/* Boyut1 bileşenin yazılım geliştirme sürecinin hangi evresine
   ait bir ürün olduğunu göstermektedir */
facet1[1]="ANALIZ";
facet1[2]="TASARIM";
facet1[3]="KOD";
facet1[4]="BELGELEMELER";
facet1[5]="SINAMA-SENARYOLARI";

```

```

/* Boyut2 bileşenlere ait soyutlamaları içermektedir */
facet2[1]="DOGRUSAL-VERI-YAPILARI";
facet2[2]="DOGRUSAL-OLMAYAN-VERI-YAPILARI";
facet2[3]="ALGORITMALAR";
facet2[4]="COMPLEX-SAYILAR";
facet2[5]="MATRISLER";

```

/* Boyut3 bileşenlere ait çeşitli işlemleri göstermektedir */

```
facet3[1]="CARPMA";
facet3[2]="TOPLAMA";
facet3[3]="CIKARMA";
facet3[4]="BOLME";
facet3[5]="IKILI-ARAMA";
facet3[6]="SIRADAN-ARAMA";
facet3[7]="KABARCIK-SIRALAMA";
facet3[8]="SECMELI-SIRALAMA";
facet3[9]="KARSILASTIRMA";
facet3[10]="SIRALI-DOLAS";
facet3[11]="SONDAN-DOLAS";
facet3[12]="BASTAN-DOLAS";
facet3[13]="ELEMEN-EKLE";
facet3[14]="ELEMEN-CIKAR";
facet3[15]="ELEMEN-DEGISTIR";
facet3[16]="ELEMEN-ELDE-ET";
facet3[17]="RASTGELE-SAYI-URET";
```

/* Boyut4 bileşenlerin üzerinde işlem yaptığı veri tipleri ve veri yapılarını göstermektedir */

```
facet4[1]="AGAC";
facet4[2]="YIGIT";
facet4[3]="KUYRUK";
facet4[4]="LISTE";
facet4[5]="TAMSAYI";
facet4[6]="KARAKTER";
```

/* Boyut5 bileşenlerin çalışma ortamını göstermektedir */

```
facet5[1]="SADECE-DOS";
facet5[2]="SADECE-UNIX";
facet5[3]="DOS-ve-UNIX";
```

```
kontrolcik1=1;
```

```
kontrolcik=1;
```

```
while (kontrolcik1==1)
```

```
{
```

```
    kontrolcik=1;
```

/* Kullanıcılar boyutlardan terim seçerek istek tanımlayıcısını oluşturur */

/* Kullanıcılar her boyuttan terim seçmek zorunda değildir. Bir boyuttan terim seçilmediği zaman "9999" girilmelidir */

/* Kullanıcılar en azından bir boyuttan bir terim seçmek zorundadır */

```
while (kontrolcik==1)
```

```
{
```

```
    printf("-----İstek Tanımlayıcı Oluşturma Bölümü-----\n");
```

```

printf("Facet Icerikleri Gormek ve Terim Secmek Icin Bir Tusa Basiniz\n");
printf("BIRINCI BOYUT ICERIGI\n");
printf("-----\n");
printf("Secmek Istediginiz Terimin Numarasini Giriniz...\n");
printf("Terim Secmek Istemiyorsaniz 9999 Giriniz...\n");
for(j=1;j<=5;j++) printf("%d-%s\n",j,facet1[j]);
printf("Seciminiz?");
scanf("%d",&sec1);
while (sec1 >5 && sec1 <9999)
{
    printf("1-5 Arasi Sayi Giriniz\n");
    scanf("%d",&sec1);
}

printf("IKINCI BOYUT ICERIGI\n");
printf("-----\n");
printf("Secmek Istediginiz Terimin Numarasini Giriniz...\n");
printf("Terim Secmek Istemiyorsaniz 9999 Giriniz...\n");
for(j=1;j<=5;j++) printf("%d-%s\n",j,facet2[j]);
printf("Seciminiz?");
scanf("%d",&sec2);
while (sec2>5 && sec2<9999)
{
    printf("1-5 Arasi Sayi Giriniz\n");
    scanf("%d",&sec2);
}

printf("UCUNCU BOYUT ICERIGI\n");
printf("-----\n");
printf("Secmek Istediginiz Terimin Numarasini Giriniz...\n");
printf("Terim Secmek Istemiyorsaniz 9999 Giriniz...\n");
for(j=1;j<=17;j++) printf("%d-%s\n",j,facet3[j]);
printf("Seciminiz?");
scanf("%d",&sec3);
while (sec3>17 && sec3<9999)
{
    printf("1-17 Arasi Sayi Giriniz\n");
    scanf("%d",&sec3);
}

printf("DORDUNCU BOYUT ICERIGI\n");
printf("-----\n");
printf("Secmek Istediginiz Terimin Numarasini Giriniz...\n");
printf("Terim Secmek Istemiyorsaniz 9999 Giriniz...\n");
for(j=1;j<=6;j++) printf("%d-%s\n",j,facet4[j]);
printf("Seciminiz?");
scanf("%d",&sec4);

```

```

while (sec4>6 && sec4<9999)
{
    printf("1-6 Arasi Sayi Giriniz\n");
    scanf("%d",&sec4);
}

printf("BESINCI BOYUT ICERIGI\n");
printf("-----\n");
printf("Secmek Istediginiz Terimin Numarasini Giriniz...\n");
printf("Terim Secmek Istemiyorsaniz 9999 Giriniz...\n");
for(j=1;j<=3;j++) printf("%d-%s\n",j,facet5[j]);
printf("Seciminiz?");
scanf("%d",&sec5);
while (sec5>3 && sec5<9999)
{
    printf("1-3 Arasi Sayi Giriniz\n");
    scanf("%d",&sec5);
}
printf("-----OLUSTURULAN ISTEK TANIMLAYICISI-----\n");
printf(" < ");

if (sec1==9999) { printf("* ,"); }
    else { printf("%s ,",facet1[sec1]); }
if (sec2==9999) { printf("* ,"); }
    else { printf("%s ,",facet2[sec2]); }

if (sec3==9999) { printf("* ,"); }
    else { printf("%s ,",facet3[sec3]); }
if (sec4==9999) { printf("* ,"); }
    else { printf("%s ,",facet4[sec4]); }
if (sec5==9999) { printf("**"); }
    else { printf("%s ",facet5[sec5]); }

printf(" >\n");
printf("\n");
if (sec1==9999 && sec2==9999 && sec3==9999 && sec4==9999 && sec5==9999)
{
    printf("En az bir terim belirleyiniz!\n");
}
else
{
    printf("Sorguyu Onayliyor musunuz? E/H\n");
    scanf("%s",ch);
    /* Oluşan istek tanımlayıcısı için kullanıcının son kez onayı alınır */
    if (ch[0] == 'e' || ch[0] == 'E') kontrolcik=0;
}
}

```

```

/* Metod için argüman listesi oluşturulur */
status = ACAS_BuildArgList ( &arglist,
                             &arg_count,

                             (ACASStringPtr)"Choice1",
                             (ACASStringPtr)ACAS_INTEGER,
                             (ACASStringPtr)&sec1,
                             (ACASULong)sizeof(ACASInt),
                             (ACASFlags)ACAS_IN,
                             (ACASFlags)ACAS_OVERWRITE,

                             (ACASStringPtr)"Choice2",
                             (ACASStringPtr)ACAS_INTEGER,
                             (ACASStringPtr)&sec2,
                             (ACASULong)sizeof(ACASInt),
                             (ACASFlags)ACAS_IN,
                             (ACASFlags)ACAS_OVERWRITE,

                             (ACASStringPtr)"Choice3",
                             (ACASStringPtr)ACAS_INTEGER,
                             (ACASStringPtr)&sec3,
                             (ACASULong)sizeof(ACASInt),
                             (ACASFlags)ACAS_IN,
                             (ACASFlags)ACAS_OVERWRITE,

                             (ACASStringPtr)"Choice4",
                             (ACASStringPtr)ACAS_INTEGER,
                             (ACASStringPtr)&sec4,
                             (ACASULong)sizeof(ACASInt),
                             (ACASFlags)ACAS_IN,
                             (ACASFlags)ACAS_OVERWRITE,

                             (ACASStringPtr)"Choice5",
                             (ACASStringPtr)ACAS_INTEGER,
                             (ACASStringPtr)&sec5,
                             (ACASULong)sizeof(ACASInt),
                             (ACASFlags)ACAS_IN,
                             (ACASFlags)ACAS_OVERWRITE,

                             (ACASStringPtr)"Param",
                             (ACASStringPtr)ACAS_INTEGER,
                             (ACASStringPtr)&param1,
                             (ACASULong)sizeof(ACASInt),
                             (ACASFlags)ACAS_OUT,
                             (ACASFlags)ACAS_OVERWRITE,

                             ACAS_LIST_END);

```

```

/* Metod çağrılır ve sunucu ile iletişim kurulur */
if (status == ACAS_SUCCESS )
{
    status = ACAS_InvokeMethod (&instance,
                                "Component",
                                arglist,
                                arg_count,
                                ctxhnd,
                                (ACASAttributeList)NULL,
                                (ACASInt)0,
                                (ACASFlags)
                                ACAS_CREATE_LONG_LIVED_APPL_HND |
                                ACAS_NO_NEW_SRV,
                                (ACASInstanceHandle *)&out_handle,
                                &inverrorlist,
                                &apperrorlist);
}

/* Metoddan dönen sonuç ekrana yazdırılır */
printf("%d", param1);

if (status == ACAS_INV_METHODFAIL && apperrorlist != NULL)
{
    status = apperrorlist[0].StatusValue.LoBitStatus;
    ACAS_DeallocateList((ACASList) apperrorlist);
}

/* ACA'ya ilişkin hatalar varsa yazdırılır */

if ((status != ACAS_SUCCESS) && (status != ACAS_INV_METHODFAIL))
{
    printf("Method invocation failed. Error list follows:\n");
    PrintACASErrors(inverrorlist);
}

printf("\n");

printf("Sorgulamaya Son İçin H Tusuna Basınız\n");
scanf("%s",ch);
if (ch[0]=='h' || ch[0]=='H') kontrolcik1=0;
} /* while */
} /* End Request Fonksiyonu */

```

```

/* Bu program önerilen geniş ölçekli tekrar kullanım mimarisi içindeki
   sunucu programdır */
/* Bu program istemciden gelen istek tanımlayıcılarını, 5 boyutlu
   grid kütük yapısında saklanan bileşen tanımlayıcıları ile eşlemektedir */
/* Bulunan bileşen(ler) için bileşen numaraları istemciye gönderilir */

```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

```

/* ACA "include" dosyalar */
#include <acasmgs.h>
#include <acasdef.h>
#include <acasapi.h>

```

```

#ifdef vms
#define EXIT_STATUS( s ) \
    exit(s);

```

```

#define EXTERN      globalref

```

```

#else
#define EXIT_STATUS( s ) \
    if (s == ACAS_SUCCESS) \
        exit(0); \
    else \
        exit(1);

```

```

#define EXTERN extern
#endif

```

```

#define BOYUT 5
#define PAGESIZE 50
#define PAGESAYISI 5
#define GRIDSAY 3

```

```

EXTERN ACASChar  LSR_METHOD_SERVER_ID [];
EXTERN ACASChar  LSR_METHOD_SERVER_NAM [];

```

```

/* Global Tanımlamalar */
struct gridstr

```

```

{
    ACASInt ilk[BOYUT+1];
    ACASInt son[BOYUT+1];
    ACASInt pageno;
    ACASInt isaret;
};

```

```

struct gridstr gptr[GRIDSAY+1][GRIDSAY+1][GRIDSAY+1][GRIDSAY+1][GRIDSAY+1];
struct gridstr
gptr[yed[GRIDSAY+1][GRIDSAY+1][GRIDSAY+1][GRIDSAY+1][GRIDSAY+1];

struct pagestr
{
    ACASInt pageid;
    ACASInt tupsay;
    ACASInt tuples[PAGESIZE+1][BOYUT+2]; /* docptr tutmak icin +2 */
};
struct pagestr pptr[PAGESAYISI+1];

ACASInt eksen[BOYUT+1][21];
ACASInt eksenyed[BOYUT+1][21];
ACASInt bol,ensonpage,saklaid1,saklaid2,saklaid3,saklaid4,saklaid5;
ACASInt say[BOYUT+1];
ACASInt yedtuples[PAGESIZE+1][BOYUT+2];

FILE *file;
FILE *kutl;

ACASULong offset,reclen,recnum;
ACASULong yer;
ACASInt kutukbul,kutukyer;
ACASInt i,j,k,l,mm,nn,qq,ll;
ACASInt ii;
ACASInt pagno;
ACASInt donus;
ACASInt sayiat,kk,kk1;
ACASInt toplam;
ACASInt t1,t2,t3,t4,t5,t6,zzzz,sayac1,ysay,pkont,x1,x2,x3,x4,x5;
ACASInt a,b,c,ana,findit,sayac2;
ACASInt ind1,ind2,ind3,ind4,ind5;
ACASInt sarray[10];
ACASInt farray[10];
ACASInt pagenums[100] , pagenu[100];

/* Fonksiyon Prototipleri */
void LSR_EndSession();
void PrintACASErrors();
void initgrid();
void initpage();
void initaxis();
ACASInt gridbol(bol,ind1,ind2,ind3,ind4,ind5);

```

```

main()
{
    ACASStatus status, LSR_METHOD_SERVER ();
    ACASAttributeList server_attrs;
    ACASInt server_attr_count;
    ACASErrorList error_list;
    void (*temp_void_routine)();

    temp_void_routine = LSR_EndSession;

    /* Otomatik olarak üretilen "dispatcher rutin" çağrılır */

    status = ACAS_BuildAttributeList ( &server_attrs,
                                      &server_attr_count,

                                      (ACASStringPtr)ACAS_METHOD_SERVER_NAME,
                                      (ACASStringPtr)ACAS_STRING,
                                      LSR_METHOD_SERVER_NAM,
                                      (ACASULong)strlen(LSR_METHOD_SERVER_NAM),
                                      (ACASFlags)ACAS_NO_OP,
                                      (ACASFlags)ACAS_OVERWRITE,

                                      (ACASStringPtr)ACAS_END_SESSION_RTN,
                                      (ACASStringPtr)ACAS_PROCEDURE,
                                      &temp_void_routine,
                                      (ACASULong)sizeof(ACASProcedure),
                                      (ACASFlags)ACAS_NO_OP,
                                      (ACASFlags)ACAS_OVERWRITE,

                                      (ACASStringPtr)ACAS_SERVER_REGISTRY,
                                      (ACASStringPtr)ACAS_STRING,
                                      (ACASStringPtr)"ACAS LSR Server",
                                      (ACASULong)15,
                                      (ACASFlags)ACAS_NO_OP,
                                      (ACASFlags)ACAS_OVERWRITE,

                                      ACAS_LIST_END );

    if (status != ACAS_SUCCESS)
    {
        printf("Could not build attribute list\n");
        EXIT_STATUS(status);
    }

    error_list = NULL;
    /* Metod-sunucuya yetki veren ACA komut kullanımı */
    status = ACAS_RegisterServer( LSR_METHOD_SERVER_ID,

```

```

        server_attr,
        server_attr_count,
        LSR_METHOD_SERVER,
        (ACASFlags)0,
        & error_list );

if (status != ACAS_SUCCESS)
{
    printf("Could not register as a server\n");
    PrintACASErrors( error_list );
    EXIT_STATUS(status);
}

for(i=0;i<10;i++)
{
    sarray[i]=0;
    farray[i]=0;
}

toplam=0;
for(j=0;j<100;j++)
    pagenums[j]=0;pagenu[j]=0;

/* Grid kütük yapısı oluşturma işlemi başlıyor */
initgrid();
initpage();
initaxis();

/* Grid dizisi ve doğrusal ölçeklerin ilklenmesi */
for(i=1;i<=BOYUT;i++)
    say[i]=2;

bol=0;
ensonpage=1;
donus=0;

for(i=1;i<=BOYUT;i++)
{
    eksen[i][1]=0;
    eksen[i][2]=32;
}
for(i=1;i<=BOYUT;i++)
{
    gptr[1][1][1][1][1][1][1].ilk[i]=0;
    gptr[1][1][1][1][1][1][1].son[i]=32;
}

```

```
gptr[1][1][1][1][1].pageno=enonpage;
```

```
/* Bilgi2.dat bileşen tanımlayıcılarını içeren kütüktür */
```

```
kut1=fopen("bilgi2.dat","r");
```

```
for(ii=1;ii<31;ii++)
```

```
{
    fscanf(kut1,"%d\n",&t1);
    fscanf(kut1,"%d\n",&t2);
    fscanf(kut1,"%d\n",&t3);
    fscanf(kut1,"%d\n",&t4);
    fscanf(kut1,"%d\n",&t5);
    fscanf(kut1,"%d\n",&t6);
```

```
for(i=2;i<=say[1];i++)
```

```
{
    if (t1 <= eksen[1][i])
    {
        ind1=i-1;
        i=say[1]+1;
    }
}
```

```
for(i=2;i<=say[2];i++)
```

```
{
    if (t2 <= eksen[2][i])
    {
        ind2=i-1;
        i=say[2]+1;
    }
}
```

```
for(i=2;i<=say[3];i++)
```

```
{
    if (t3 <= eksen[3][i])
    {
        ind3=i-1;
        i=say[3]+1;
    }
}
```

```
for(i=2;i<=say[4];i++)
```

```
{
    if (t4 <= eksen[4][i])
    {
        ind4=i-1;
```

```

    i=say[4]+1;
  }
}

for(i=2;i<=say[5];i++)
{
  if (t5 <= eksen[5][i])
  {
    ind5=i-1;
    i=say[5]+1;
  }
}

/* Pagno değışkeni, kullanımda olan grid bloğunun karşılık geldiği
fiziksel disk sayfasıdır */
pagno=gptr[ind1][ind2][ind3][ind4][ind5].pageno;

/* Fiziksel disk sayfasında yer var mı? */
if (pptr[pagno].tupsay < PAGESIZE)
{ /* Yer varsa bileşen tanımlayıcı bu sayfaya eklenir */
  pptr[pagno].tupsay=pptr[pagno].tupsay+1;
  pptr[pagno].tuples[pptr[pagno].tupsay][1]=t1;
  pptr[pagno].tuples[pptr[pagno].tupsay][2]=t2;
  pptr[pagno].tuples[pptr[pagno].tupsay][3]=t3;
  pptr[pagno].tuples[pptr[pagno].tupsay][4]=t4;
  pptr[pagno].tuples[pptr[pagno].tupsay][5]=t5;
  pptr[pagno].tuples[pptr[pagno].tupsay][6]=t6;
}

/* Fiziksel disk sayfasında taşma varsa grid kütük yapısı bölünür */
if (pptr[pagno].tupsay ==PAGESIZE)
{
  gptr[ind1][ind2][ind3][ind4][ind5].isaret=1;
  /* grid yedekleme */
  for(i=1;i<=GRIDSAY;i++)
  {
    for(j=1;j<=GRIDSAY;j++)
    {
      for(l=1;l<=GRIDSAY;l++)
      {
        for(mm=1; mm<=GRIDSAY;mm++)
        {
          for(nn=1; nn<=GRIDSAY;nn++)
          {
            gptryed[i][j][l][mm][nn].pageno=gptr[i][j][l][mm][nn].pageno;
            gptryed[i][j][l][mm][nn].isaret=gptr[i][j][l][mm][nn].isaret;
            for(k=1;k<=BOYUT;k++)

```

```

        {
            gptried[i][j][l][mm][nn].ilk[k]=gptr[i][j][l][mm][nn].ilk[k];
            gptried[i][j][l][mm][nn].son[k]=gptr[i][j][l][mm][nn].son[k];
        }
    }
}
}
}

/* Bölünme politikasına göre bölünme eksenini bulunması */
bol=bol+1;
if (bol > BOYUT) { bol=1; }
donus=gridbol(bol,ind1,ind2,ind3,ind4,ind5);
while (donus==1) /* Bölünmeye rağmen bazı disk sayfalarında halen
    {
        daha taşma var */
        bol=bol+1;
        if (bol > BOYUT) { bol=1; }
        gptr[saklaind1][saklaind2][saklaind3][saklaind4][saklaind5].isaret=1;
        /* grid yedekleme */
        for(i=1;i<=GRIDSAY;i++)
        {
            for(j=1;j<=GRIDSAY;j++)
            {
                for(l=1;l<=GRIDSAY;l++)
                {
                    for(mm=1; mm<=GRIDSAY;mm++)
                    {
                        for(nn=1; nn<=GRIDSAY;nn++)
                        {
                            gptried[i][j][l][mm][nn].pageno=gptr[i][j][l][mm][nn].pageno;
                            gptried[i][j][l][mm][nn].isaret=gptr[i][j][l][mm][nn].isaret;
                            for(k=1;k<=BOYUT;k++)
                            {
                                gptried[i][j][l][mm][nn].ilk[k]=gptr[i][j][l][mm][nn].ilk[k];
                                gptried[i][j][l][mm][nn].son[k]=gptr[i][j][l][mm][nn].son[k];
                            }
                        }
                    }
                }
            }
        }
        donus=gridbol(bol,saklaind1,saklaind2,saklaind3,saklaind4,saklaind5);
    }
} /* end if */
} /* end ana dongu */

```

```

fclose(kut1);
printf("grid olustu ");
/* Grid oluşturma işlemi bitimi */

```

```

/* İstemciden gelecek isteği bekleyen sonsuz döngü */

```

```

status = ACAS_MainLoop ( & error_list );
printf("in main loop\n");
if (status != ACAS_SUCCESS)
{
    printf("Error in main loop\n");
    PrintACASErrors(error_list);
    EXIT_STATUS(status);
}
}

```

```

/* İstek tanımlayıcısını bileşen tanımlayıcıları ile eşleyen metod */

```

```

ACASStatus Component_rtn (Choice1, Choice2,Choice3,Choice4,Choice5, Param,
                          InvocationContext, ErrorList)

```

```

ACASInt *Choice1, *Choice2, *Choice3,*Choice4,*Choice5;

```

```

ACASInt *Param;

```

```

ACASInvocationContext * InvocationContext;

```

```

ACASErrorList *ErrorList;

```

```

{
    ACASInt n[BOYUT+1];
    ACASInt m[BOYUT+1];
    ACASInt terim[BOYUT+1][50];
    ACASChar kay[80];
    ACASChar param[500];

```

```

    terim[1][0]=*Choice1;

```

```

    terim[2][0]=*Choice2;

```

```

    terim[3][0]=*Choice3;

```

```

    terim[4][0]=*Choice4;

```

```

    terim[5][0]=*Choice5;

```

```

    /* İstek Tanımlayıcısı terim dizini içine yerleştirildi */

```

```

    /* Arama işlemi başlıyor */

```

```

    for(j=1;j<6;j++)

```

```

    {

```

```

        if (terim[j][0]==9999)

```

```

        {

```

```

            m[j]=1;

```

```

            n[j]=say[j];

```

```

        }

```

```

    else

```

```

    {

```

```

m[j]=indisbul(terim[j][0],j);
n[j]=m[j]+1;
}
}

sayac1=0;
ysay=0;
for(x1=m[1];x1<n[1];x1++)
{
for(x2=m[2];x2<n[2];x2++)
{
for(x3=m[3];x3<n[3];x3++)
{
for(x4=m[4];x4<n[4];x4++)
{
for(x5=m[5];x5<n[5];x5++)
{
pagenu[ysay]=gptr[x1][x2][x3][x4][x5].pageno;
ysay=ysay+1;
pkont=0;
if (ysay !=1)
{
for (zzzz=0;zzzz<ysay-1;zzzz++)
if (pagenu[zzzz]==pagenu[ysay-1]) pkont=1;
}
if (pkont !=1)
{
pagenums[sayac1]=gptr[x1][x2][x3][x4][x5].pageno;
sayac1=sayac1+1;
toplam=toplam+1;
}
} /* for */
} /* for */
} /* end for */
}
}

sayac2=0;
for(ana=0;ana<sayac1;ana++)
{
for(i=1;i<=PAGESIZE;i++)
{
findit=1;
for(j=1;j<=BOYUT;j++)
{
if ( (terim[j][0] != 9999) &&
(pptr[pagenums[ana]].tuples[i][j]!=terim[j][0]) ) findit=0;
}
}
}

```

```

    if (findit==1)
    {
        sayac2++;
        sarray[sayac2]=pptr[pagenums[ana]].tuples[i][6];
        farray[sayac2]=pptr[pagenums[ana]].tuples[i][1];
    }
}

/* Arama işlemi sonu */

*Param = sarray[1];
if (sayac2==0) { printf("Istenen Ozellikte Bilesen Bulunamadi\n");
    *Param = 0;
}
return (ACAS_INV_METHODSUC);
} /* metod sonu */

/*Sırasıyla grid bloklarını, disk sayfalarını ve doğrusal ölçekleri
ilkleyen fonksiyonlar */

void initgrid()
{
    ACASInt i,j,k,l,m,n;

    for(i=1;i<=GRIDSAY;i++)
    {
        for(j=1;j<=GRIDSAY;j++)
        {
            for(l=1;l<=GRIDSAY;l++)
            {
                for(m=1;m<=GRIDSAY;m++)
                {
                    for(n=1;n<=GRIDSAY;n++)
                    {
                        gptr[i][j][l][m][n].pageno=0;
                        gptr[i][j][l][m][n].isaret=0;
                        gptryed[i][j][l][m][n].pageno=0;
                        gptryed[i][j][l][m][n].isaret=0;
                        for(k=1;k<=BOYUT;k++)
                        {
                            gptr[i][j][l][m][n].ilk[k]=0;
                            gptr[i][j][l][m][n].son[k]=0;
                            gptryed[i][j][l][m][n].ilk[k]=0;
                            gptryed[i][j][l][m][n].son[k]=0;
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
  }
}
} /* end initgrid() */

void initpage()
{
  ACASInt i,j,k;

  for(i=1;i<=PAGESAYISI;i++)
  {
    pptr[i].pageid=i;
    pptr[i].tupsay=0;
    for(j=1;j<=PAGESIZE;j++)
      for(k=1;k<=BOYUT+1;k++)
        pptr[i].tuples[j][k]=0;
  }
} /* end initpage() */

void initaxis()
{
  int i,j;
  for(i=1;i<=BOYUT;i++)
  {
    for(j=1;j<=20;j++)
    {
      eksen[i][j]=0;
      eksenyed[i][j]=0;
    }
  }
}

} /* end initaxis() */

```

/* Doğrusal ölçek üzerinde indis bulan fonksiyon */

```

ACASInt indisbul(par1,eks)
ACASInt par1,eks;
{
  ACASInt il;
  ACASInt geri;

  for(il=2;il<=say[eks];il++)
  {
    if (par1 <= eksen[eks][il])
    {

```

```

geri=i1-1;
i1=say[eks]+1;
}
}

return(geri);
} /* end function */

/* Grid kütük yapısını bölen fonksiyon */
/* 5 boyutlu grid kütük yapısını bölen bu fonksiyonun işlevi ve
gerçekleştirimi ek-3'de verilen ile tamamen aynıdır */

ACASInt gridbol(bol,ind1,ind2,ind3,ind4,ind5)
ACASInt bol,ind1,ind2,ind3,ind4,ind5;
{
ACASInt deg[BOYUT+1];
ACASInt deg1,deg2,deg3,orta,fark;
ACASInt i,k,j,l,n,nn,m,u,t,v;
ACASInt bull;
ACASInt yenibol,dongu1,dongu2,dongu3,dongu4,tut;
ACASInt cs;
ACASInt yeni1,yeni2,yeni3,yeni4,yeni5,temp;
ACASInt pagno1,pagno2,sakladongu,kontroldeg1;
ACASInt oo1,oo2,oo3,oo4,oo5;
ACASInt a,b,c,d,e;
ACASInt c1,c2,c3,c4;
ACASInt sonuc;

deg1=gptr[ind1][ind2][ind3][ind4][ind5].son[bol];
deg2=gptr[ind1][ind2][ind3][ind4][ind5].ilk[bol];
fark=deg1-deg2;
orta=fark/2;
deg3=deg2+orta;

sonuc=0;

for(i=1;i<=say[bol];i++) eksenyed[bol][i]=eksen[bol][i];

for(k=1;k<=say[bol];k++)
{
if (eksen[bol][k]==deg2)
{
bull=k;
k=say[bol]+1;
}
}
}

```

```

cksen[bol][bul+1]=deg3;
say[bol]=say[bol]+1;

for(i=bul+2;i<=say[bol];i++)
{
    cksen[bol][i]=ckseny[bol][i-1];
}

yenibol=-bol;
switch(yenibol)
{
case(1) : {
    dongu1=2;
    dongu2=3;
    dongu3=4;
    dongu4=5;
    }
    break;
case(2) : {
    dongu1=1;
    dongu2=3;
    dongu3=4;
    dongu4=5;
    }
    break;
case(3) : {
    dongu1=1;
    dongu2=2;
    dongu3=4;
    dongu4=5;
    }
    break;
case(4) : {
    dongu1=1;
    dongu2=2;
    dongu3=3;
    dongu4=5;
    }
    break;
case(5) : {
    dongu1=1;
    dongu2=2;
    dongu3=3;
    dongu4=4;
    }
    break;
} /* end switch */

```

```

oo1=say[bol]-2;
oo2=say[dongu1]-1;
oo3=say[dongu2]-1;
oo4=say[dongu3]-1;
oo5=say[dongu4]-1;

```

```

for(a=1; a<=oo1; a++)
for(b=1; b<= oo2; b++)
for(c=1; c<= oo3; c++)
for(d=1; d<= oo4; d++)
for(e=1; e<= oo5; e++)
{
switch(bol)
{
case(1) : {
i=a;
j=b;
u=c;
t=d;
v=e;
}
break;
case(2) : {
i=b;
j=a;
u=c;
t=d;
v=e;
}
break;
case(3) : {
i=b;
j=c;
u=a;
t=d;
v=e;
}
break;
case(4) : {
i=b;
j=c;
u=d;
t=a;
v=e;
}
break;

```

```

case(5) : {
    i=b;
    j=c;
    u=d;
    t=e;
    v=a;
}
break;

} /* end switch */

if ( (gptried[i][j][u][t][v].ilk[bol] <= deg2) &&
      (gptried[i][j][u][t][v].son[bol] <= deg2) ) cs=1;
if ( (gptried[i][j][u][t][v].ilk[bol] == deg2) &&
      (gptried[i][j][u][t][v].son[bol] == deg1) ) cs=2;
if ( (gptried[i][j][u][t][v].ilk[bol] >= deg1) &&
      (gptried[i][j][u][t][v].son[bol] >= deg1) ) cs=3;
switch(cs)
{
case 1 : {
    for(l=1;l<=BOYUT;l++)
    {
        gptr[i][j][u][t][v].ilk[l]=gptried[i][j][u][t][v].ilk[l];
        gptr[i][j][u][t][v].son[l]=gptried[i][j][u][t][v].son[l];
    }
    gptr[i][j][u][t][v].pageno=gptried[i][j][u][t][v].pageno;
    gptr[i][j][u][t][v].isaret=gptried[i][j][u][t][v].isaret;
} /* case 1 */
    break;

case 2 : {
    gptr[i][j][u][t][v].ilk[bol]=gptried[i][j][u][t][v].ilk[bol];
    gptr[i][j][u][t][v].son[bol]=deg3;
    tut=1;
    for (n=1;n<=BOYUT;n++)
    {
        if (n != bol)
        {
            deg[tut]=n;
            tut=tut+1;
        }
    }
    for(nn=1;nn<tut;nn++)
    {
        gptr[i][j][u][t][v].ilk[deg[nn]]=gptried[i][j][u][t][v].ilk[deg[nn]];
        gptr[i][j][u][t][v].son[deg[nn]]=gptried[i][j][u][t][v].son[deg[nn]];
    }
}
}

```

```

gptr[i][j][u][t][v].pageno=gptried[i][j][u][t][v].pageno;
gptr[i][j][u][t][v].isaret=0;

if (bol == 1) {yeni1=i+1;yeni2=j;yeni3=u;yeni4=t;yeni5=v;}
if (bol == 2) {yeni1=i;yeni2=j+1;yeni3=u;yeni4=t;yeni5=v;}
if (bol == 3) {yeni1=i;yeni2=j;yeni3=u+1;yeni4=t;yeni5=v;}
if (bol == 4) {yeni1=i;yeni2=j;yeni3=u;yeni4=t+1;yeni5=v;}
if (bol == 5) {yeni1=i;yeni2=j;yeni3=u;yeni4=t;yeni5=v+1;}

gptr[yeni1][yeni2][yeni3][yeni4][yeni5].ilk[bol]=deg3;
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].son[bol]=gptried[i][j][u][t][v].son[bol];

for(nn=1;nn<tut;nn++)
{
    gptr[yeni1][yeni2][yeni3][yeni4][yeni5].ilk[deg[nn]]=gptried[i][j][u][t][v].ilk[deg[nn]];

gptr[yeni1][yeni2][yeni3][yeni4][yeni5].son[deg[nn]]=gptried[i][j][u][t][v].son[deg[nn]];
}

gptr[yeni1][yeni2][yeni3][yeni4][yeni5].isaret=0;
if (gptried[i][j][u][t][v].isaret == 0)
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].pageno=gptried[i][j][u][t][v].pageno;
else
{
    ensonpage=ensonpage+1;
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].pageno=ensonpage;
pagno1=gptried[i][j][u][t][v].pageno;
pagno2=gptr[yeni1][yeni2][yeni3][yeni4][yeni5].pageno;
sakladongu=pptr[pagno1].tupsay;
if (bol==1) { c1=2; c2=3; c3=4; c4=5; }
if (bol==2) { c1=1; c2=3; c3=4; c4=5; }
if (bol==3) { c1=1; c2=2; c3=4; c4=5; }
if (bol==4) { c1=1; c2=2; c3=3; c4=5; }
if (bol==5) { c1=1; c2=2; c3=3; c4=4; }
for(l=1;l<=sakladongu;l++)
{
    if ( ( pptr[pagno1].tuples[l][bol] > deg3 ) &&
        ( pptr[pagno1].tuples[l][c1] <= gptried[i][j][u][t][v].son[c1] ) &&
        ( pptr[pagno1].tuples[l][c2] <= gptried[i][j][u][t][v].son[c2] ) &&
        ( pptr[pagno1].tuples[l][c3] <= gptried[i][j][u][t][v].son[c3] ) &&
        ( pptr[pagno1].tuples[l][c4] <= gptried[i][j][u][t][v].son[c4] ) )
    {
        pptr[pagno2].tupsay=pptr[pagno2].tupsay+1;
        pptr[pagno1].tupsay=pptr[pagno1].tupsay-1;
        pptr[pagno2].tuples[pptr[pagno2].tupsay][1]=pptr[pagno1].tuples[l][1];
        pptr[pagno2].tuples[pptr[pagno2].tupsay][2]=pptr[pagno1].tuples[l][2];
        pptr[pagno2].tuples[pptr[pagno2].tupsay][3]=pptr[pagno1].tuples[l][3];
        pptr[pagno2].tuples[pptr[pagno2].tupsay][4]=pptr[pagno1].tuples[l][4];
    }
}

```

```

pptr[pagno2].tuples[pptr[pagno2].tupsay][5]=pptr[pagno1].tuples[1][5];
pptr[pagno2].tuples[pptr[pagno2].tupsay][6]=pptr[pagno1].tuples[1][6];
    pptr[pagno1].tuples[1][1]=0;
pptr[pagno1].tuples[1][2]=0;
pptr[pagno1].tuples[1][3]=0;
pptr[pagno1].tuples[1][4]=0;
pptr[pagno1].tuples[1][5]=0;
pptr[pagno1].tuples[1][6]=0;
    }
}
kontroldeg1=0;
for(m=1;m<=sakladongu;m++)
{
    if ( (pptr[pagno1].tuples[m][1] != 0) &&
        (pptr[pagno1].tuples[m][2] != 0) &&
        (pptr[pagno1].tuples[m][3] != 0) &&
        (pptr[pagno1].tuples[m][4] != 0) &&
        (pptr[pagno1].tuples[m][5] != 0) )
    {
        kontroldeg1++;
        yedtuples[kontroldeg1][1]=pptr[pagno1].tuples[m][1];
        yedtuples[kontroldeg1][2]=pptr[pagno1].tuples[m][2];
        yedtuples[kontroldeg1][3]=pptr[pagno1].tuples[m][3];
        yedtuples[kontroldeg1][4]=pptr[pagno1].tuples[m][4];
        yedtuples[kontroldeg1][5]=pptr[pagno1].tuples[m][5];
        yedtuples[kontroldeg1][6]=pptr[pagno1].tuples[m][6];
        pptr[pagno1].tuples[m][1]=0;
        pptr[pagno1].tuples[m][2]=0;
        pptr[pagno1].tuples[m][3]=0;
        pptr[pagno1].tuples[m][4]=0;
        pptr[pagno1].tuples[m][5]=0;
        pptr[pagno1].tuples[m][6]=0;
    }
}
for(m=1;m<=kontroldeg1;m++)
{
    pptr[pagno1].tuples[m][1]=yedtuples[m][1];
    pptr[pagno1].tuples[m][2]=yedtuples[m][2];
    pptr[pagno1].tuples[m][3]=yedtuples[m][3];
    pptr[pagno1].tuples[m][4]=yedtuples[m][4];
    pptr[pagno1].tuples[m][5]=yedtuples[m][5];
    pptr[pagno1].tuples[m][6]=yedtuples[m][6];
}

```

```

if (pptr[pagno1].tupsay==PAGESIZE)
{
sonuc=1;
saklind1=i;
saklind2=j;
saklind3=u;
saklind4=t;
saklind5=v;
}
} /* else */
} /* case 2 */
break;
case 3 : {
tut=1;
for(n=1;n<=BOYUT;n++)
{
if (n != bol)
{
deg[tut]=n;
tut=tut+1;
}
}
if (bol == 1) {yeni1=i+1;yeni2=j;yeni3=u;yeni4=t;yeni5=v;}
if (bol == 2) {yeni1=i;yeni2=j+1;yeni3=u;yeni4=t;yeni5=v;}
if (bol == 3) {yeni1=i;yeni2=j;yeni3=u+1;yeni4=t;yeni5=v;}
if (bol == 4) {yeni1=i;yeni2=j;yeni3=u;yeni4=t+1;yeni5=v;}
if (bol == 5) {yeni1=i;yeni2=j;yeni3=u;yeni4=t;yeni5=v+1;}
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].ilk[bol]=gptried[i][j][u][t][v].ilk[bol];
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].son[bol]=gptried[i][j][u][t][v].son[bol];
for(nn=1,nn<tut,nn++)
{
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].ilk[deg[nn]]=
gptried[i][j][u][t][v].ilk[deg[nn]];
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].son[deg[nn]]=
gptried[i][j][u][t][v].son[deg[nn]];
}
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].pageno=gptried[i][j][u][t][v].pageno;
gptr[yeni1][yeni2][yeni3][yeni4][yeni5].isaret=gptried[i][j][u][t][v].isaret;
} /* case 3 */
break;
} /* end switch */
} /* end for j */
/* end for i */
return(sonuc);
} /* Gridbol fonksiyonu sonu */

```

```

/* Metod-sunucu serbest kalıyor */
void LSR_EndSession (MethodServerId, SessionId)
ACASStringPtr      MethodServerId;
ACASULong          SessionId;
{
    ACASStatus status;
    ACASErrorList error_list = NULL;
    status = ACAS_UnregisterServer (MethodServerId, & error_list);
    if (status != ACAS_SUCCESS)
    {
        printf("Error unregistering as a server\n");
        if (error_list != NULL)
        {
            PrintACASErrors(error_list);
        }
    }
}

/* ACA'ya ilişkin hataların yazdırılması*/
void PrintACASErrors( errorlist )

ACASErrorList errorlist;

{
    if (errorlist != 0)
    {
        ACASStatus sts;
        ACASInt errorcount, buf_len;
        ACASStringPtr buf_ptr;

        errorcount = errorlist [ 0 ] .Count;
        buf_len = 256 * errorcount;
        buf_ptr = (ACASStringPtr) malloc( buf_len );
        sts = ACAS_FormatACASErrors(
            errorlist, errorcount,
            buf_len, buf_ptr, 0, "", NULL );
        if (sts != ACAS_SUCCESS)
        {
            printf("error calling ACAS_FormatACASErrors: %u\n", sts );
        }
        else
        {
            printf( "%s", buf_ptr );
        }
        free( buf_ptr );
    }
} / Hata yazdırma fonksiyonu sonu */

```

/* Bu program bundan sonraki tüm açıklama satırları da dahil olmak üzere ACASGEN komutu tarafından üretilmiştir
 Bu programda üretilen "dispatcher rutinleri", sunucu yazılım tarafından kullanılmaktadır */

/*
 * Created Wed Jun 28 15:52:17 1995 by ACAS V2.1-40 (GENERATE)
 *
 */

/*
 * ACAS Dispatchers
 * -----
 *

* This module contains dispatcher routines for the following Method
 * Servers:

* LSR_METHOD_SERVER
 *

* How This Module Works
 * -----
 *

* A dispatch routine serves as the mechanism for executing a specific
 * method for a methodserver. The dispatch routine contains a call
 * to the actual method routine defined in the class repository for the
 * method.

* The dispatch routine executes the appropriate call for the method
 * specified by uuid. Additionally, it performs some argument validation
 * prior to executing the method. Specifically, the dispatch routine hashes
 * the method uuid to a unique id. This id is then used to 'switch' to the
 * appropriate method routine call defined for that method.

* This hashing operation is accomplished through two tables generated for
 * a specific dispatch routine:

* 1) BUCKET_TABLE - the bucket table contains the uuids of the methods
 * that hashed to that particular bucket. The method uuids are
 * represented as contiguous null terminated string. The entire buffer
 * is terminated with a NUL character. The bucket table is defined
 * as a structure in order to allow it to be SHARED READONLY. Buckets
 * are referenced from the HASH_TABLE by offset from the top of the
 * table. The methods in a bucket are sorted alphabetically.

* 2) HASH_TABLE - the hash table is the actual hash table that is indexed
 * by hashing the method uuid. Each hash table entry includes a start

```

*   id and an offset into the bucket table. The entry in the bucket table
*   will be searched for an exact match by method uuid. The start id
*   is the unique id for the first method in bucket. When a match is
*   found the unique id will be start_id + offset (by method uuid) into
*   the bucket. This id will then be used to 'switch' to the executable
*   code for that method.
*
*/

```

```

#include <stdio.h>
#include <acasdef.h>
#include <acasapi.h>
#include <acasmmsg.h>

```

```

/*
* The following macro definitions are provided in order to allow the
* tables and global variables to be included as SHARED READONLY in a
* shareable image. A PSECT name may be specified by defining RO_PSECT
*/

```

```

#ifndef GLOBAL_READONLY
#if defined(ACAS_ANSI) || !defined(vms)
#define GLOBAL_READONLY
#else
#ifndef RO_PSECT
#define RO_PSECT
#endif
#define GLOBAL_READONLY globaldef RO_PSECT readonly
#endif
#endif

```

```

#ifndef STATIC_READONLY
#if defined(ACAS_ANSI) || !defined(vms)
#define STATIC_READONLY static
#else
#define STATIC_READONLY static readonly
#endif
#endif

```

```

/*
* The following globals are the UUID and name for each of the method servers
* for which a dispatch routine is generated. It is recommended that this
* global be used to specify the UUID in the ACAS_RegisterServer routine call
* for registering the corresponding method server. The name global should be
* used for specifying the ACAS_METHOD_SERVER_NAME server attribute.
*/

```

```
GLOBAL_READONLY ACASChar LSR_METHOD_SERVER_ID [] =
"4e74efe600f7.02.10.1f.10.2f.00.00.00";
GLOBAL_READONLY ACASChar LSR_METHOD_SERVER_NAM [] =
"LSR_METHOD_SERVER";
```

```
/*
*
* ROUTINE NAME: LSR_METHOD_SERVER
*
* FUNCTIONAL DESCRIPTION:
*
* This is the method dispatch routine for the method
* server LSR_METHOD_SERVER.
*
* This routine calls the specified method. A hashing technique
* is used to determine which method routine to execute. This
* technique is described in the module header.
*
* FORMAL PARAMETERS:
*
* parameter name      [In|Out|In/Out]      [Reference|Value]
* -----
* invctx              In                    Reference
*                    The invocation context handle. This is
*                    simply passed to the method routine.
*
* method              In                    Reference
*                    The uuid of the method to execute.
*
* argcount             In                    Value
*                    The number of arguments for the method call.
*                    This is used to verify whether the number of
*                    passed arguments conforms with the method
*                    definition.
*
* args                In                    Reference
*                    The arguments for the method call.
*                    They will be verified and passed individually
*                    to the method routine.
*
* errorlist            Out                   ReferenceReference
*                    The error list created by the method
*                    execution.
```

* IMPLICIT INPUTS:

*

* NONE

*

* IMPLICIT OUTPUTS:

*

* NONE

*

* FUNCTION VALUE:

*

* ACAS_INV_MTHDNOTFND (the specified method was not found)

* ACAS_INV_ARGMISMATCH (the passed arguments do not conform in number
or type)

* ACAS_INV_BADARGLST (the argument list is NULL)

* Status returned by the invoked method

*

* SIDE EFFECTS:

*

* NONE

*

*/

ACASStatus EXPORT

LSR_METHOD_SERVER (invctx, method, argcount, args, errorlist)

ACASInvocationContext * invctx;

ACASStringPtr method;

ACASInt argcount;

ACASArgList args;

ACASErrorList * errorlist;

{

ACASInt hashvalue;

ACASArg * arg_ptr = args;

STATIC_READONLY struct {

ACASChar Bucket0 [38]; } BUCKET_TABLE = {

"6f2557a652ec.02.9b.df.01.0d.00.00.00\000"

};

STATIC_READONLY ACASHashEntry HASH_TABLE [] =

{

{ 0, 0 }

};

/*.....*/


```
    return(ACAS_INV_ARGMISMATCH);
    arg_ptr++;

    return( Component_rtn (
        args[0] . Value,
        args[1] . Value,
        args[2] . Value,
        args[3] . Value,
        args[4] . Value,
        args[5] . Value,
        invctx,
        errorlist ));
}

default :
    return(ACAS_INV_MTHDNOTFND);
}

}
```

