İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

GÖRSEL PROGRAMLAMA TEKNİKLERİ VE BİR KALIP TASARIM UYGULAMASI

YÜKSEK LİSANS TEZİ

Müh. Somer ÇALBAŞ

Tezin Enstitüye Verildiği Tarih	:	27 MAYIS 1996
Tezin Savunulduğu Tarih	:	13 HAZİRAN 1996
Tez Danışmanı	:	Prof. Dr. Nadir YÜCEL
Diğer Jüri Üyeleri	:	Doç. Dr. Füsun TUNALI
	:	Yrd. Doç. Dr. Tevfik AKGÜN

HAZİRAN 1996

ÖNSÖZ

Nesne yönelimli grafik bazlı program geliştirme ve görsel ortamlar konularını işlediğim tezimin görsel programlama ortamlarında çalışacak kişiler için bir temel teşkil edeceği inancındayım. Yüksek lisans sürem boyunca olaylara bakış açısı ile bana daima örnek olan Danışman Hocam Prof. Dr. Nadir Yücel'e, tez konumun hazırlanışı sırasında desteklerini ve yönlendirmelerini eksik etmeyen hocalarım Doç. Dr. Füsun Tunalı ve Y. Doç. Dr. Tevfik Akgün'e ve teknik bilgi desteğini sağlayan Sayın Belgin Bozkurt'a teşekkürü bir borç bilirim.

Sümer ÇALBAŞ
İstanbul 1996

İÇİNDEKİLER

ÖZET

GÖRSEL PROGRAMLAMA TEKNİKLERİ VE BİR KALIP TASARIM UYGULAMASI

Anahtar Kelimeler: Görsel Programlama Ortamları, MFC, Kalıp, Visual C++, Nesne Yönelimli Programlama

Bu çalışmada Microsoft Windows görsel kulanıcı arayüzü özelliklerinin ve bu işletim sistemi altında kullanılan görsel programlama ortamlarının tanıtımı yapılmış, bu ortamlarda uygulama geliştirme metodolojisi verilirken görsel bir uygulamanın temel oluşturma adımları ve bu aşamalarda geliştiriciye getirilen kolaylıklar incelenmiştir. Grafik ortamlarına tipik bir örnek olan tekstil kalıp tasarım sisteminde bu metodolojinin uygulaması yapılmıştır.

SUMMARY

VISUAL DEVELOPMENT TOOLS AND A PATTERN DEVELOPMENT SYSTEM

Keywords: Visual Development Environments, MFC, Pattern, Visual C++, Object Oriented Programming

Starting in early 90's, MS Windows, has so far successfully conquered the PC software market, ending an era of DOS predominance. The user friendly interface of Windows has encouraged and helped many hesitant end users to become more acquainted with computers and applications. When more users started using software applications, the need for programs with better and more intuitive user interfaces became far more evident. This necessity led toward the birth of new programming techniques and environments. Many companies, but Microsoft in particular, invested heavily for development of visual user interfaces. In this new visual market Visual Basic and Visual C++ have become the most popular programming environments. Introduction of these visual programming tools based on object oriented programming concepts has substantially changed programming styles and habits of many software developers.

MS Windows

Windows has become extremely popular among users and consequently grew to be the market leader through the ease of use it provided with its user interface. The most important feature of the Windows interface is that most visual components are based on metaphores which the users are able to relate to objects they use in their daily lives. (Eg: A command button with a raised effect to start a process.)

One of the many other attractive features of Windows is its multitasking capability. This allows a user to switch among many applications and exchange data without having to close any of them. In classical DOS based applications users had to close an application and then start a new one. This feature improved the vision of many users and drastically boosted the utilization of the software.

In the previous DOS based systems the use of peripheral devices like Printers, Plotters was highly constrained since these devices required their drivers to be incorporated within applications. The use of a new device meant a modification in source code and recompilation. Windows on the other hand, provides a standard interface for all applications allowing a program to prepare a device independent output, leaving Windows to take care of device specific details to format the output and send to the device.

Visual Basic

Visual Basic is the most widely used programming system to develop software to run under Windows. Being a product with features for both the programmers and end users it improved its market shares sharply. In 1995 only, Visual Basic was sold more than 2.000.000 copies. Visual Basic is preferred especially for database applications.

Visual Basic is an event-driven and document-centered application. With these orientations, Visual Basic can use some features of Object Oriented Programming Philosophy. But Visual Basic is not a real Object Oriented Programming environment. You cannot create new objects and cannot modify existing ones. Development environment of new objects is Visual C++. The Visual C++ environment is fully object-oriented.

Visual C++

In programming market, Visual C++ is the most powerful and efficient programming environment. Most application programs (for Windows) have been developed with Visual C++. Some advantages of Visual C++ are as follows:

- Handling of Huge Projects. With increases in project sizes, handling application code becomes too difficult with classical programming environments such as C or FORTRAN. In Visual C++, programmers can easily handle applications with more than 25.000 program lines.
- Visual C++ codes are expandable and reusable.
- Full Object Oriented Programming Environment.
- Access to low level functions.
- Speed

With Visual C++, programmers are able to construct the below software pieces:

- Windows EXE
- Windows DLL
- Windows Static Library
- DOS EXE, COM
- Visual Basic VBX (Custom Controls)

In addition to these types, Visual C++ supports both 16 bit and 32 bit operating environments.

Due to newer requirements from computer users, new design and programming philosophies were developed by computer scientists. One of the leading examples of these philosophies is Object Oriented Programming.

Object Oriented Programming

In ancient times users loaded their programs using switches on the control panels. This was an appropriate way for programs of small size. With the increases in program sizes other and more efficient methods were needed. High level languages such as Assembler and FORTRAN satisfied these requirements. In the meantime the advent of structured methodologies followed by the first structured languages, provided the developers with better and tighter control over program

code; but when program sizes began to exceed 20.000-25.000 lines maintenance and modification of existing code became an unbearable task. Upon facing such challenge software scientists came up with a different approach which was built on logical grouping of concepts and flow of data among these logical objects through inter-object communication, namely Object Oriented Methodology. Object Oriented Programming Languages have three major features:

- Encapsulation
- Polymorphism
- Inheritance

In the design phase of the project, the first task is to determine which objects are needed in the application. This phase is the longest period of the project development phase. In general these projects have an accelerating nature. When the correct objects are in place, it is a rather simplified task to proceed with the rest of the project.

Although most software is built for rather varying purposes, the essence of all requires or comprises similar components, for this reason it is an awkward solution to rewrite these components. With such component reusability concerns programmers make use of object libraries which include base classes.

The most popular class library is Microsoft Foundation Classes supplied by Microsoft Corporation. This library includes most of the needed base classes for developing a Windows application.

Computers In Textile Industry

In textile industry, people use computers for 3 different purposes :

- Accounting
- Inventory Control and Manufacturing
- Pattern Design

The last of these purposes falls within the scope of this thesis. Pattern design is a matter of computer graphics. Many of computer graphical algorithms are used for implementing pattern design and modification. Some important features in pattern design are stated below:

- Lines and curves
- Adding and removing points from pattern
- Symmetry of a pattern
- Cutting a pattern
- Joining two distinct patterns
- Rotating a pattern
- Zooming in and out
- Adding seams to a pattern
- Grading
- Sizing and deforming patterns
- Adding grain lines, notes and special signs to a pattern

This thesis consists of four sections. First a brief introduction to graphical interfaces has been provided followed by comprehensive information on Visual

C++ and Visual Basic. Succeeding section covers the design methodology and development steps of a an application with a visual interface. Finally, a graphical pattern design system has been provided to serve as an example of this methodology and the tools.

ŞEKİL LİSTESİ

TABLO LİSTESİ

BÖLÜM 1. GİRİŞ

Gelişen bilgisayar teknolojisinin getirdiği olanaklar hem kullanıcı sayısının artmasına hem de bilgisayar kullanım alanlarının genişlemesine önayak olmaktadır. Bu da bilgisayar kullanımının uzmanlar tekelinden alınıp standart özellikli insanlara yaygınlaşmasına sebep olmaktadır. Böylece uzun yıllar çekinilerek yaklaşılan bilgisayarlar normal kullanıcılar için artık daha fazla araştırılacak ve özelliklerinden maksimum yararlanılacak cihazlar olarak görülmektedir. İşte bilgisayar kullanıcılarındaki bu yaygınlaşma normal özellikteki insanlar için kullanımı kolay görsel kullanıcı arabirimlerinin yaygınlaşmasına neden olmuştur.

Kişisel bilgisayarlarda DOS bazlı işletim sistemlerinin saltanatının sona ermeye başladığı yıllarda bilgisayarcılar bilgisayar ekranının pencereler halinde kullanımının işleri kolaylaştırdığını ve ekrandaki çeşitli görsel etkilerin kullanıcı üzerinde hem hatırlama hem de kullanımı kolay olma etkisini gördüler. Bu nedenle kullanıcının bilgisayarla etkileşiminin birinci elemanı ekranı kullanıcının gerçek hayatında kullandığı nesnelere benzer görsel elemanlarla doldurdular. Örnek olarak bir komutun çalıştırılması gerçek hayattakine benzer bir şekilde dışarı doğru çıkıntılı bir tuşla yapıldı. Bu çalışmalar görsel kullanıcı arayüzlerinin ilk örnekleri oldular. Kullanıcı arayüzlerinin kullanıcılar üzerindeki etkisi; psikologların da yardımı ile test edildi ve görülen cisimlerin insan psikolojisi ve görme fizyolojisi ile bağlantıları, yeni arayüzler için temel bilgileri oluşturdu.

Zaman içindeki gelişimi ile şu andaki halini alan görsel arayüzler arasında Windows işletim sistemi kullanım kolaylığı ile rakipleri arasından sıyrılarak önemli bir pazar kazandı. Windows işletim sistemi aşağıda tanıtılan özellikleri basit bir kişisel bilgisayar düzeyine indirerek bilgisayar sektörünün bu kadar yaygınlaşmasında gerçekten önemli rol oynadı.

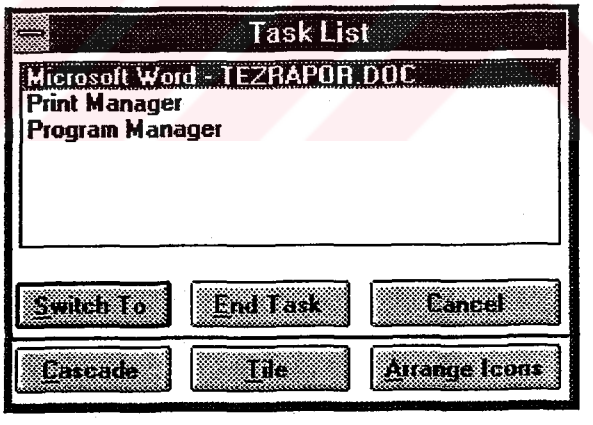
Tez kapsamı içinde; Windows kullanıcı arayüzünün görsel özellikleri, Windows işletim sisteminin dayandığı nesneye yönelik programlama temelleri ve nesneye yönelik grafik bazlı bir Windows uygulamasının geliştirme yöntemi verilmiştir.

BÖLÜM 2. GÖRSEL PROGRAMLAMA ORTAMLARI

Görsel ortamlarda çalışan sistemler kullanıcılara aşağıdaki avantajları sunarlar:

2.1 Çoklu Çalışma (Multitasking)

DOS bazlı sistemlerde kullanıcı bir anda bir uygulama programı ile çalışabiliyordu. Dolayısıyla örneğin bir metin işleme programından işlem programına geçebilmek için önce metin işleme programını kapaması ve sonra da işlem programını açması gerekiyordu. İki uygulama arasında bilgi taşınması gerekirse de muhtemelen kağıt üzerine alınan notlar diğer uygulamada tekrar girilmek zorunda kalınıyordu. Windows programlarında ise uygulamalar zaman paylaşımlı olarak paralel olarak çalışırlar. Kullanıcı bir diğer uygulamaya geçebilmek için sadece bir görev değişim komutunu çalıştırabilir.



Şekil 2-1 Görev Değiştirme Ekranı

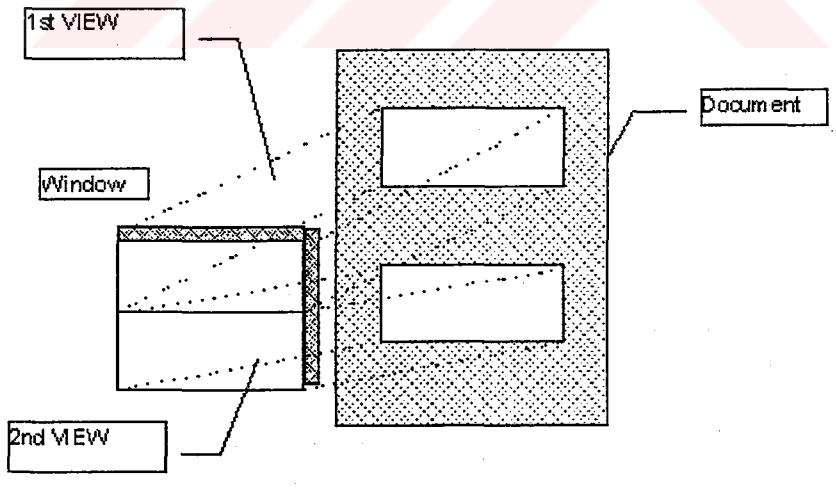
Aynı zamanda ön planda kullanıcı bir uygulama ile çalışırken arka planda bir diğer görev işletim sistemi tarafından yerine getirilebilir.

2.2 Görsel Kullanıcı Standartları (Visual Standarts)

Kullanıcıların bilgisayar kullanım alanları arttıkça birçok değişik uygulama ile çalışır oldular. Bu da her bir uygulama için eğitim süreçlerinin gerekliliğini ortaya çıkardı. Farklı uygulamalar arasında geçişlerin kolaylaşması amacıyla görsel ortamlarda program geliştirme ile ilgili standartlar tanımlandı. Bu standartlar bir komut butonu veya araç çubuğunun büyüklüğünden kullanılacak renklere ve hatta bu renklerin birbirleri arasında geçişlere kadar uzandı.

2.3 Çoklu Ekranla Çalışabilme

Ayrı ayrı uygulamalar veya aynı uygulama üzerinde birden fazla ekranla çalışabilme standart Windows desteğidir. Windows tarafından sağlanan MDI Multiple Document Interface tekniği birçok pencere ile çalışabilme ve bu pencereler arasında bir tuş veya fare hareketi ile geçiş yapabilme olanağı sağlamaktadır. Ayrıca standart olarak desteklenen kesme, kopyalama, yapıştırma işlemleri ile de pencereler arası veri aktarımı çok kolay bir şekilde sağlanmaktadır.



Şekil 2-2 Çok Görüntülü (View) Bir Document

2.4 Nesne Bağlama ve Gömme (OLE Object Linking And Embedding)

Windows tarafından sağlanan nesne gömme ve bağlama teknikleri ile uygulamalar arasında veri taşınması ve gerektiğinde bu veriler üzerinde tekrar işlem yapılabilmesi desteklenmektedir. Bu sayede kullanıcı metin işleme programı içine bir tablolama programının verisini koyabilmekte ve gerektiğinde metin üzerindeki bu tabloyu tekrar değiştirebilmektedir.

2.5 Genel Sürücü Desteği

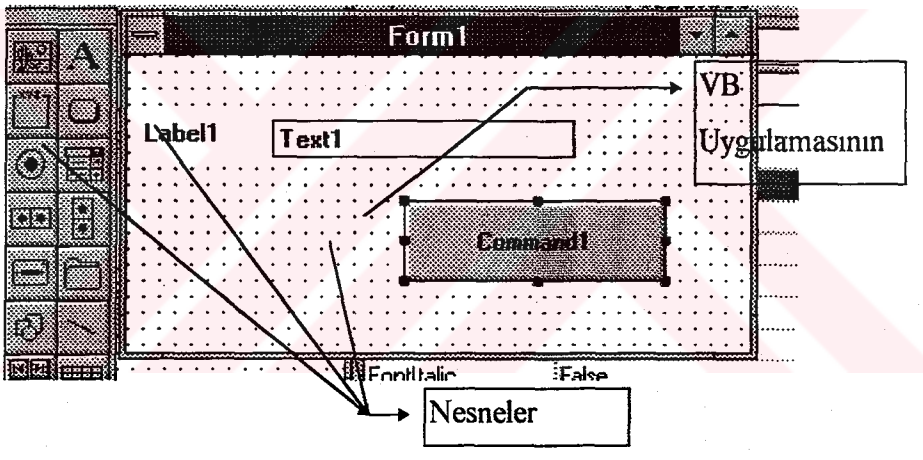
Tüm dünyada büyük oranda kullanılan Windows platformu bakımı,kurulumu ve kullanımı kolay ve ucuz olan bir ortam sağlamaktadır. Windows tarafından sürücüsü bulunan tüm yardımcı aygıtlar (printer, plotter, mouse, scanner, digitizer vb.) ek bir çaba gerektirmeden windows altında çalışan uygulamalar tarafından kullanılabilmektedir. Uygulama çıktı veya girdi işlemlerini standart olarak Windows'a bildirdikten sonra karşı cihazın özellikleri sürücüler tarafından Windows'a aktarılmakta ve Windows bunları standart bir yapıda uygulamaya geçirmektedir. Bu da herbir uygulama için bu işlemlerin tekrarlanması gerekliliğini ortadan kaldırarak hem kullanıcıları rahatlatmış hem de kullanıcıların daha fazla teknik bilgi gerekliliğini ortadan kaldırmıştır.

Ayrıca uygulama temel olarak çokdilli bir yapıda geliştirildiğinden ülkeye özgü para birimi ve formatı, tarih formatı gibi birçok özellik Windows'tan otomatik olarak alınabilmektedir. Ayrıca uygulamada statü çubuğu, araç çubuğu ve diğer görsel araçlar standart olarak kullanılmakta dolayısıyla diğer Windows uygulamalarından alışık olduğu şekilde görüntü ve kullanım standartlarını (Look & Feel) sağlamaktadır.

Görsel programlama ortamlarının en popüler iki örneği Visual Basic ve Visual C++ dilleridir. Tez amaçlarından biri olan grafik ortamının gerçeklenmesi bu iki ortamda da mümkündür. Grafik amaçlı kullanım için bu iki ortamın da genel özellikleri aşağıda verilmiştir.

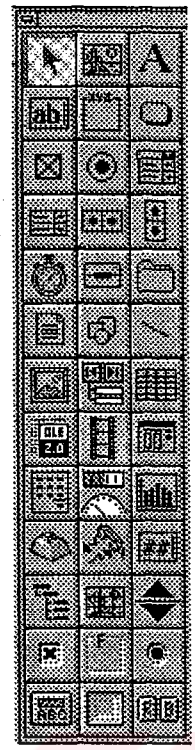
2.6 Visual Basic

VB Olay-Sürümlü (Event-Driven), Döküman-Bazlı (Document-Centered) bir geliştirme ortamıdır. Program akışı klasik prosedür bazlı programlama dillerinden farklı olarak durmadan çalışan bir döngü içerisinde değildir. Uygulama programı içerisinde bulunan nesneler (object) kendileri ile ilgili bir olay (event) gerçekleştiğinde canlanırlar ve kendilerine atanmış fonksiyonları çalıştırırlar. Dolayısıyla sistemde herhangi bir olay oluşana kadar uygulama bekleme durumunda kalır. Ayrıcı Döküman-Bazlı bir yapıda olduğu için yazılacak herhangi bir uygulamanın bir form üzerine kurulması gerekmektedir.



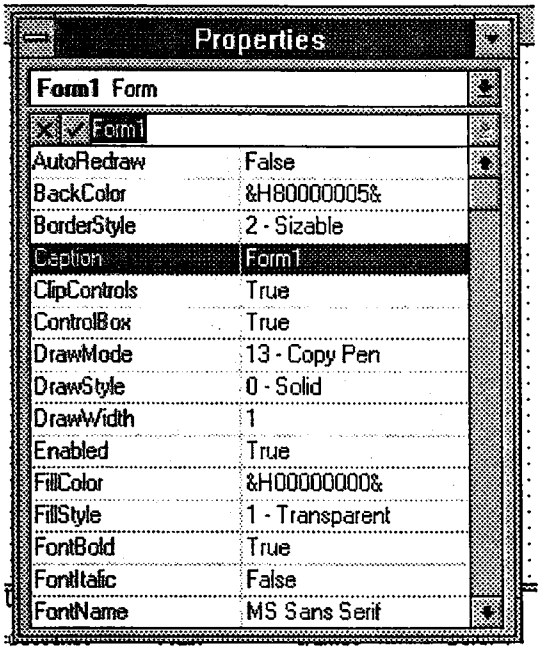
Şekil 2-3 Visual Basic Formu

VB gerçek anlamda bir Object Oriented geliştirme ortamı değildir. VB ile uygulama geliştirilirken VISUAL C++ ile yaratılmış programlama nesnelerine ihtiyaç vardır. VB ile yeni bir nesne tanımlanamaz, üzerinde modifikasyon yapılamaz. Varolan nesneler üzerinde ise oldukça esnek bir şekilde programcı hakimiyeti bulunmaktadır. Sistemde tanımlı nesneler uygulama içinde dinamik olarak yaratılabilir. Form üzerinde kullanılacak nesneler ancak nesne kutusundan (Toolbox) seçilebilir.



Şekil 2-4 ToolBox

Sisteme yeni nesneler eklenebilir. Bunlar VISUAL C++ ortamında yazılmış VBX (Visual Basic Custom Control) diye adlandırılan ek rutinlerdir. Eklenen yeni VBX'ler sayesinde programcı uygulamasına yeni boyutlar katabilir. Ancak eklenen nesnelerin özellikleri (properties) VBX ile sağlanan standart özelliklerdir. Bu özelliklere ekleme yapılamaz. Sistemde varolan bir VBX'in özellikleri Özellikler penceresinden (Property Window) görülebilir.



Şekil 2-5 Property Window

VB aynı zamanda yine VISUAL C++'de yazılmış DLL(Dynamic Link Library) rutinlerini kullanarak programlama ortamında bulunmayan fonksiyonaliteyi ve Windows'un API(Application Programming Interface) fonksiyonlarını da kullanabilmektedir. Ama yine açıktır ki VB ile DLL yazılamaz ve varolan DLL'ler üzerinde geliştirme yapılamaz.

VB şu anda 16 Bit Windows platformunda çalışmaktadır. Ve yapılan uygulamalar yine 16 Bit olarak çalışmaktadır. Yeni teknoloji ürünü Windows NT ailesi ve Windows 95 işletim sistemleri 32 Bit mimariyi desteklemesine rağmen bilgisayar dünyasında önemli bir pazar payı olan MS Windows ailesi şu anda büyük oranda 16 Bitlik Windows işletim sistemlerini kullanmaktadır. Dolayısıyla bir süre daha 16 Bitlik işletim sistemleri ve 16 Bitlik uygulama ortamları varlığını sürdürecektir.

VB uygulamaları tam anlamıyla EXE (executable) tipli program olmamaktadır. Hazırlanan programlar her zaman bir run-time modüle ihtiyaç duymaktadır. Bu yorumlayıcı desteğinden dolayı da performans olarak hiçbir zaman gerçek derlenmiş bir program düzeyine erişememektedir.

VB çabuk ve kolay geliştirme sağladığından özellikle veritabanı uygulama geliştiricilerin tercih ettiği bir platform olmaktadır. Ancak donanımın daha fazla niteliğinin kullanılması gereken uygulamalarda (sistem programları, grafik uygulamaları vb.) VB yeterli olmamakta ve VISUAL C++ ile yazılmış destek rutinlere ve daha fazla özelliği bulunan nesnelere ihtiyaç duyulmaktadır.

2.7 VISUAL C++

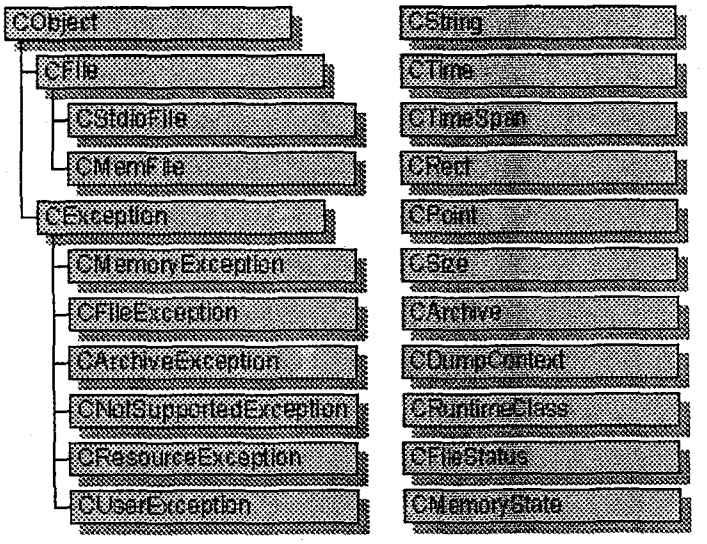
Atası olan C programlama dilinin tüm özelliklerini barındıran VISUAL C++ Windows platformunda programcıya sonsuz esneklik sağlayarak gerçek bir geliştirme ortamı sağlamaktadır. Hem 16 Bit hem de 32 Bit platformlarda çalışabilen programlar yazılabilmektedir. Ayrıca hazırlanacak programlar aşağıdaki program tiplerinde olabilmektedir:

- Windows EXE
- DOS EXE
- Windows DLL
- Windows VBX
- Win32 Static Library

VISUAL C++ tam anlamıyla Object-Oriented bir programlama dilidir. Bir OOP dilinin taşıması gereken tüm özellikleri taşır. Windows Message sistemi üzerinde programcının komple kontrolü olduğundan windows işletim sisteminin gerekli tüm özelliklerini kullanılabilir.

Varolan sınıflar kullanılarak sınıfın sahip olduğu özelliklere ek bazı özellikler getirilebilir. Dolayısıyla VISUAL C++ ile yazılan programların en önemli noktası veri yapılarının bulunduğu sınıfların doğru kurulması sınıflar arası ilişkilerin doğru tanımlanması olmaktadır. Bu yapı kurulduktan sonra program sınıfların member variable ve fonksiyonlarının yazılması ile oluşturulacaktır. Her uygulama için bu sınıfların yeniden kurulması bir yük getireceği için hazır sınıf library'leri bulunması uygulama geliştirmek için büyük avantaj sağlayacaktır. MS bu amaçla genel amaçlı bir Class Library hazırlamıştır. MFC (Microsoft Foundation Classes) adı verilen bu

library içinde bir uygulama için gerekli olan ana sınıflarla birlikte bir grafik programı gerçeklemek için bir çok hazır sınıf bulundurmaktadır.



Şekil 2-6 Genel amaçlı sınıflar

2.8 VB , VISUAL C++ Karşılaştırması

1.Bir grafik programı VB ile de yazılabilir. Fakat şu anda VB'de bulunan nesneler (VBX) istenen grafik özelliklere sahip değildir. Dolayısıyla istenen özelliklere sahip bir VBX geliştirilmesi gerekmektedir. Bu VBX'in geliştirme ortamı da VISUAL C++ platformudur. Dolayısıyla VISUAL C++'de geliştirme gerekmektedir. VISUAL C++'de bu geliştirme prosesine girildiğinden uygulamanın kalan kısmının da VISUAL C++'de geliştirilmesi VISUAL C++'nin OOP özelliklerinin kullanılması ve alt seviye program yapısından gelen göreceli performans fazlalığı grafik bazlı profesyonel projelerin VISUAL C++ üzerinde devam edilmesi fikrinin gerçekleşmesine sebep olmuştur.

BÖLÜM 3. GÖRSEL ORTAMLARDA UYGULAMA GELİŞTİRME METODOLOJİSİ

3.1 Giriş

Microsoft Visual C++; Microsoft Windows işletim sistemi altında yüksek performanslı uygulama geliştirme ortamlarına yeni bir bakış açısı getirmiştir. Öncelikle tam anlamıyla windows bazlı bir geliştirme ortamı sağlamasının yanında klasik C/C++ geliştirme sürecine popüler görsel kullanıcı arayüzünü adapte ederek güçlü bir geliştirme ortamı sağlamıştır.

Bir Visual C++ uygulaması MFC(Microsoft Foundation Class Library),Visual Workbench ve AppStudio yardımcı araçlarını kullanılarak Windows için geliştirilmiş bir uygulamadır. Tümüyle entegre çalışan bir sistemi sağlamak amacıyla program geliştirme süreci kullanıcının programı kullanacağı sırada görsel arabirim elemanları (visual interface elements) ile etkileşimi izlenerek ve aynı metod kullanılarak yapılır. Görsel programlama terminolojisinde bu elemanlar "kullanıcı-arayüzü nesneleri" diye adlandırılır. Uygulama geliştirici öncelikle kullanıcı etkileşimini sağlayan bu arayüzü tasarladıktan sonra uygulama için gerekli program kodlarını bu kullanıcı arayüzü arkasına ekler.

3.1.1 Visual Workbench

Visual Workbench asıl program yazma ve debug etme ortamıdır. İçinde bir metin yazıcı, proje yöneticisi, browser ve bir debugger içerir. Visual C++ program geliştirme ortamında aşağıda tanımlanan yardımcı geliştirme ortamları bulunmaktadır.

3.1.2 App Studio

App Studio kullanıcı görsel arayüzünde kullanılacak dialog box, menü, ikon, bitmap, cursor vb. gibi diğer elemanların yaratıldığı bir ortamdır. Bu araç kendi başına çalışan bir uygulama olarak varolan resource'lar üzerinde düzeltme yapabileceği gibi proje geliştirme ortamı içinde de yardımcı bir araç olarak kullanılabilir.

3.1.3 AppWizard

Visual C++ kullanılarak gerçeklenen bir uygulamanın iskelet yapısını kurmak için kullanılır. AppWizard sayesinde sadece belli seçenekleri işaret ederek Windows temelli bir uygulamanın birçok fonksiyonuna sahip bir baz uygulama hiç kod yazmaksızın oluşturulabilir. Tüm seçenekleri işaretlenmiş bir AppWizard uygulaması aşağıdaki özellikleri içerir:

- MDI - Multiple Document Interface (Çoklu Döküman Arayüzü),
- Aynı anda birden fazla ekranla çalışma desteği.
- Dosya saklama, okuma, yazma(printing) ve baskı önizleme(print preview) fonksiyonlarının gerçeklenebilmesi için dialog box ve menüler,
- OLE - Object Linking and Embedding (Nesne Gömme ve Bağlama) desteği sağlamak amacıyla client ve server nesneler,
- Veritabanı erişimi için kayıt erişimi ve ekran hazırlama desteği,
- Visual Basic custom kontrolleri (VBX) kullanabilme,
- Help (Yardım) desteği,
- Toolbar (Araç çubuğu) ve Status Bar (Durum çubuğu) desteği,

3.1.4 ClassWizard

ClassWizard yardımcı programı kullanılarak uygulamaya

- Yeni class'lar eklenebilir
- Class-Member fonksiyonlarına Windows mesajları map edilebilir.
- Class-Member fonksiyonlarına kontroller map edilebilir.
- ClassWizard ile yeni bir class yaratma işlemi varolan base-class'lardan biri seçilip yeni class'ın adının yazılması ile gerçekleşir. Bu aşamadan sonra ClassWizard gerekli member fonksiyon ve deklarasyonları otomatik olarak koda ekler.
- ClassWizard çoğunlukla App Studio ile yaratılmış görsel elemanları koda bağlamak amacıyla kullanılır. Örnek olarak Ana menüye "Kontrol" adında bir yeni seçenek App Studio ile eklendikten sonra bu seçeneğin gerektirdiği kod ClassWizard ile bu nesneye bağlanır.

Visual C++ ile görsel bir uygulama geliştirme esas olarak iki safhalıdır: Uygulama yaratma ve uygulama geliştirme. Birinci safha adından da anlaşılacağı gibi doğrudan bir işlemdir. İkincisi ise iteratif özellikli ve birçok bileşene sahip bir yapı gösterir.

3.1.5 Uygulama yaratma safhası

Bir Visual C++ uygulamasının ilk safhası olan uygulama yaratma; AppWizard kullanılarak başlangıç için gerekli dosyaların oluşturulması ile başlar. Bu işlem için AppWizard çalıştırılır ve uygulamanın gerektirdiği opsiyonlar AppWizard'a girilir.

3.1.6 Uygulama geliştirme Safhası

Yaratım safhasında elde edilen baz uygulama üzerine App Studio, ClassWizard, Visual Workbench yardımcı araçları kullanılarak gerekli kod ve nesneler eklenir. Bu safha sırasında uygulama test ve debug edilerek gerekli hata düzeltme işlemleri

yapılır. Bu işlemler genelde iteratif ve duruma bağlı oldukları için belirli bir prosedürü yoktur ve uygulama geliştiren kişiye bağlı olarak kendine has bir özellik gösterir.

MFC uygulama geliştiricilere Windows grafik ortamında uygulama yaratabilmek için gerekli bir grup class tanımı içerir. Bu class grubu aynı zamanda application framework olarak da tanımlanır.MFC ile desteklenen bu application framework; Object Oriented bir class kütüphanesi olduğu için hem çok güçlü hem de yeniden kullanılabilir bir yapı oluşturur. Framework'ün kaynak kodlarının manuel olarak yazılması yerine MFC içinde varolan bir class'tan bu class özelliklerini içeren ve istenen yeni özellikler ile zenginleştirilmiş yeni bir class yaratılabilir.

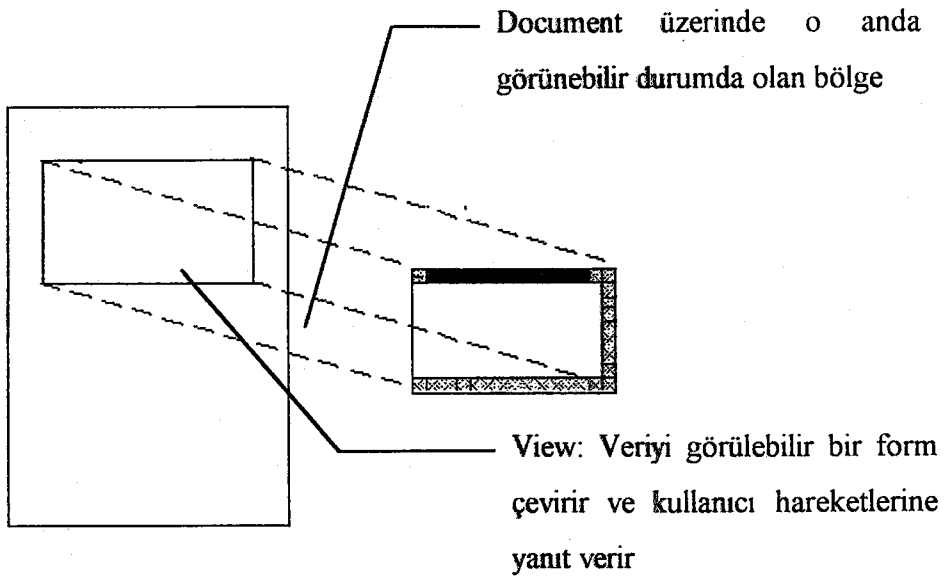
Application framework altında geçen tanımlar şu şekilde listelenebilir:

Windows için geliştirilmiş uygulamanın kalbi olan "application object"

Kullanıcının üzerinde veri ile çalıştığı "document". Döküman class'ı veriyi yükler, kayıt eder ve manipüle eder.

Kullanıcı document ile etkileşimini bu document üzerinde tanımlanmış "view" ile yapar. View klavye ve fare hareketlerini alarak document'a bildirir ve seçme/yazma gibi fonksiyonların kotarılmasını sağlar.

Kullanıcı arayüzündeki nesneler (menüler , tuşlar gibi) document, view veya diğer nesnelere komutlar göndererek uygulama mantığını oluştururlar.



Şekil 3-1 Document - View İlişkisi

3.2 Genel Dizayn Felsefesi

Microsoft Windows; C++ dili bu denli popüler olmadan önce tasarlanmıştı. Binlerce uygulama C dilinde yazılmış C-language Windows application programming interface (API) arayüzünü kullanırlar. Herhangi bir C++ Windows arayüzü procedural C-dilindeki API ler üzerine kurulur. Dolayısıyla bu gerçek C++ uygulamalarının her zaman C uygulamaları ile birlikte varolacağını garanti etmektedir.

3.2.1 The Application Framework

Microsoft Foundation Class Library nin çekirdeği Windows API sisteminin büyük bir bölümünü C++ formunda içerir. Bunların içinde pencereler, diyalog kutuları, device contextleri, fırça ve kalem gibi genel amaçlı GDI nesneleri ve diğer standart Windows elemanları bulunur. Fakat Microsoft Foundation Class Library bu temel üzerine ek bir uygulama fonksiyonalitesi katmanı daha ekler. Bu katman çalışan bir Windows programının görsel kullanıcı arayüzünün büyük bir bölümünün gerçeklenebilmesini sağlar.

3.2.2 C-Language API İle İlişkisi

Microsoft Foundation Library içindeki sınıfların diğer sınıf kütüphanelerinden en önemli farkı C dili ile yazılmış olan Windows API sistemine çok yakın bir yapı göstermesidir. Bu sayede uygulama geliştirilirken MFC çağrıları ile Windows API fonksiyonlarına direk çağrılar serbestçe karıştırılarak kullanılabilir. Fakat bu API fonksiyonlarının tamamen ortadan kalkması demek değildir. Bazı durumlarda geliştiriciler Windows fonksiyonlarına direk çağrı yapmak durumunda kalabilirler (Örn. GetSystemMetric fonksiyonu). Bu anlamda bir Windows fonksiyonu ancak belirgin bir yarar sağlanıyorsa sınıfın bir member fonksiyonu ile değiştirilmelidir.

Tüm bunlar gözönüne alınırsa Microsoft Foundation Class Library içindeki sınıflar Windows üzerinde geliştirilecek uygulama için bir "application framework" oluştururlar. En genel anlamda, framework; bir uygulamanın iskeletini oluşturur ve bu iskelet üzerinde kullanıcı arayüzünün temel uygulanışını standart olarak sağlar. Programcının bu aşamadan sonra görevi uygulamaya bağlı olarak gerekli özellikleri bu iskelet üzerine kurmaktır.

3.2.3 Framework

Aşağıda bir Windows uygulamasının geliştirilmesi için gerekli ana sınıflar ve bu sınıfların oluşturduğu framework'ün basitleştirilmesi için kullanılan üç yardımcı araç tanıtılmıştır. Sınıflardan çoğu Microsoft Windows application programming interface (API) nin büyük bir kısmını içerir.Diğer sınıflar döküman, view ve application nesneleri olarak uygulamanın kavramlarını içerir.

3.2.4 SDI ve MDI

Microsoft Foundation Class Library hem single document interface (SDI) hem de multiple document interface (MDI) uygulamalarla kolayca çalışabilme olanağı sağlamaktadır.

SDI uygulamaları bir anda ancak bir dökümanla çalışabilmektedir. MDI uygulamaları ise uygulamanın aynı instance'ı içinde birden fazla döküman penceresi açabilmektedir. MDI uygulamalarındaki MDI child pencereler herbiri

ayrı ayrı dökümanları içeren pencerelerdir. Bazı uygulamalarda child pencereler ayrı ayrı tiplerde olabilir. Bu durumda menüler, araç ve durum çubukları da bu döküman tipine göre değişirler.

3.2.5 Documents, Views, and the Framework

Framework'ün merkezinde döküman ve view kavramları bulunur.Döküman kullanıcının bir oturum boyunca etkileşimde bulunacağı veri nesnesidir. Tipik olarak File Open/New komutları ile yaratılır ve bir disk dosyasına kayıt edilir.View ise kullanıcının dökümanla etkileşimde bulunacağı bir arayüz sağlar.

Çalışan bir uygulamanın anahtar nesneleri:

- The document(s)

CDocument sınıfından türetilir.

- The view(s)

CView sınıfından türetilir. Kullanıcının veri üzerindeki penceresidir. View sınıfı kullanıcının veriyi nasıl göreceğini ve nasıl etkileşimde bulunacağını tanımlar. Bazı durumlarda verinin birden fazla bölgesinin görünmesi istenir. Kaydırma (Scrolling) yeteneği de sağlanmak isteniyorsa CScrollView sınıfından türetilmelidir. Veri sadece tekst üzerinde çalışıyorsa CEditView, diyalog bazlı bir formla çalışıyorsa da CFormView sınıfından türetilir. [1]

3.2.6 ·The frame windows

Viewlar "document frame windows" içindeki alanda görünürler. SDI uygulama içinde document frame window aynı zamanda main frame window'dur. MDI uygulamanın içindeki tüm client pencereler ise main frame window içinde kalırlar.SDI uygulamalar için document frame window CFrameWnd sınıfından, MDI uygulamalar için main frame window CFrameWnd sınıfından türetilirler. CMDIChildWnd MDI uygulamalarda tüm değişik tipli child pencereler için kullanılır.

- The document template(s)

Document template döküman, view ve frame windowların yaratılmasını yönetir. Bir tip Document template sınıfı o tipteki tüm dökümanların yaratılması ve yönetimini üstlenir. Birden fazla tipte child pencere içeren uygulamalarda tip sayısı kadar değişik Document template vardır.SDI uygulamalar için CSingleDocTemplate, MDI uygulamalar için ise CMultiDocTemplate sınıfı kullanılır.

3.2.7 The application object

CWinApp'dan türetilen application sınıfı yukarıda tanımlanan tüm nesneleri kontrol eder ve uygulamanın başlaması ve bitişi gibi durumları kotarır. Bir uygulamada bir ve yalnız bir application sınıfı bulunur.

Çalışan bir uygulama içinde tüm bu nesneler birlikte kullanıcı hareketlerine yanıt verirler. Bir application nesnesi bir veya daha fazla document template sınıfını yönetir. Her document template bir veya daha fazla sayıda dökümanı(MDI veya SDI olmasına göre) yaratır ve yönetir. Kullanıcı frame window içinde viewlar sayesinde dökümanlarla etkileşir.

3.2.8 Framework Üzerinde Uygulama Kurulumu

Uygulama geliştirilirken programcının rolü uygulamaya yönelik kodun girilmesi ve uygulama içindeki bileşenleri belirleyip bunların birbirleri ile haberleşmesi ve komutlara yanıt vermelerinin düzenlenmesidir. C++ dili ve klasik C++ teknikleri kullanılarak base sınıfın standart davranışı alınır ve eğer bunun üzerine değişik bir yapı kurulmak isteniyorsa bu özellik override edilir. Tablo da uygulama geliştirirken framework ile programcının sorumlulukları verilmiştir [1].

Tablo 3-1 Framework ve kullanıcının karşılıklı sorumlulukları

You Do	The Framework Does
Create a skeleton application. Run	AppWizard creates the files for a skeleton

AppWizard. Specify the options you want in the Options dialog box.	application, including source files for your application, document, view, and frame windows; a resource file; a project file (.MAK); and others ¾ all tailored to your specifications.
See what it offers without adding a line of your own code. Build the skeleton application and run it in Visual Workbench.	The running skeleton application derives many standard File, Edit, View, and Help menu commands from the framework. For MDI applications, you also get a fully functional Window menu, and the framework manages creation, arrangement, and destruction of MDI child windows.
Construct your application's user interface.	
Use App Studio to visually edit the application's user interface: · Create menus. · Define accelerators. · Create dialog boxes. · Create and edit bitmaps, icons, and cursors. · Edit the toolbar bitmap created for you by AppWizard. · Create and edit other resources. You can also test the dialog boxes in App Studio.	The default resource file created by AppWizard supplies many of the resources you need. App Studio lets you edit existing resources and add new resources, easily and visually.
Map menus to handler functions. Use ClassWizard to connect menus and accelerators to handler functions in your code.	ClassWizard inserts message-map entries and empty function templates in the source files you specify and manages many manual coding tasks.
Write your handler code.	Use ClassWizard to jump directly to the code in the Visual Workbench editor. Fill in the code for your handler functions.

	ClassWizard brings up the editor, scrolls to the empty function template, and positions the cursor for you.
Map toolbar buttons to commands.	
Map each button on your toolbar to a menu or accelerator command by assigning the button the appropriate command ID.	The framework controls the drawing, enabling, disabling, checking, and other visual aspects of the toolbar buttons.
Test your handler functions.	
Rebuild the program and use Visual Workbench's built-in debugging tools to test that your handlers work correctly. You can step or trace through the code to see how your handlers are called. If you've filled out the handler code, the handlers carry out commands. The framework will automatically disable menu items and toolbar buttons that are not handled.	
Create additional classes. Use ClassWizard to create additional document, view, and frame-window classes beyond those created automatically by AppWizard.	ClassWizard adds these classes to your source files and helps you define their connections to any commands they handle.
Implement your document class. Implement your application-specific document class(es). Add member variables to hold data structures. Add member functions to provide an interface to the data. The framework already knows how to interact with document data files. It can	

open and close document files, read and write the document's data, and handle other user interfaces. You can focus on how the document's data is manipulated.	
Implement Open, Save, and Save As commands. Write code for the document's Serialize member function.	The framework displays dialog boxes for the Open, Save, and Save As commands on the File menu. It writes and reads back a document using the data format specified in your Serialize member function.
Implement your view class. Implement one or more view classes corresponding to your documents. Implement the view's member functions that you mapped to the user interface with ClassWizard.	The framework manages most of the relationship between a document and its view. The view's member functions access the view's document to render its image on the screen or printed page and to update the document's data structures in response to user editing commands.
Enhance default printing.	
If you need to support multipage printing, override view member functions.	The framework supports the Print, Print Setup, and Print Preview commands on the File menu. You must tell it how to break your document into multiple pages.
Add scrolling. If you need to support scrolling, derive your view class(es) from CScrollView.	The view automatically adds scroll bars when the view window becomes too small.
Create form views. If you want to base your views on dialog-template resources, derive your view class(es) from CFormView.	The view uses the dialog-template resource to display controls. The user can tab from control to control in the view.

Create a simple text editor. If you want your view to be a simple text editor, derive your view class(es) from CEditView.	The view provides editing functions, Clipboard support, and file input/output.
Add splitter windows. If you want to support window splitting, add a CSplitterWnd object to your SDI frame window or MDI child window and hook it up in the window's OnCreateClient member function.	The framework supplies splitter-box controls next to the scroll bars and manages splitting your view into multiple panes. If the user splits a window, the framework creates and attaches additional view objects to the document.
Add dialog boxes. Design dialog-template resources with App Studio. Then use ClassWizard to create a dialog class and the code that handles the dialog box.	The framework manages the dialog box and facilitates retrieving information entered by the user. Initialize, validate, and retrieve dialog-box data.
You can also define how the dialog box's controls are to be initialized and validated. Use ClassWizard to add member variables to the dialog class and map them to dialog controls. Specify validation rules to be applied to each control as the user enters data. Provide your own custom validations if you wish.	The framework manages dialog-box initialization and validation. If the user enters invalid information, the framework puts up a message box and lets the user reenter the data.
Build, test, and debug your application. Use the facilities of Visual Workbench to build, test, and debug your	Visual Workbench is closely coupled with AppWizard, App Studio, and ClassWizard. It lets you adjust compile, link, and other

application.	options. And it lets you browse your source code and class structure.

Görüldüğü gibi AppWizard, App Studio, and ClassWizard yardımcı araçları uygulama kodlarının büyük bölümünü kolayca yönetebilmek için oldukça kolaylık sağlamaktadırlar. Uygulamaya yönelik kodların büyük kısmı döküman ve view sınıfları içindedir [2].

Tüm bu işlemleri manuel olarak kodla yazmak mümkündür. Fakat açıkça görülmektedir ki bu araçlar sayesinde geliştirme zamanı ve harcanan enerji çok azalmakta ve hata yapma ihtimali de oldukça düşmektedir.

3.2.8.1 Application Framework Kodu Çağırışı

Uygulama geliştirme sırasında programcı tarafından yazılan kodlar ile application framework kodları arasındaki ilişkinin anlaşılması önemlidir. Uygulama çalıştırıldığında program akışının büyük bir kısmı framework tarafından sağlanan kodlar üzerinde olacaktır. Framework kullanıcı hareketleri sonucu oluşan Windows mesajlarını mesaj sistemi içinde çevirir. Framework'ün cevap verdiği event'lerin bir kısmının ise uygulama kodu ile ilgisi yoktur. Örneğin, framework kullanıcı seçeneği sonucu pencerelerin nasıl kapanacağı ve bir uygulamadan nasıl çıkılacağını bilir. Bu görevleri yerine getirmek için mesaj handlerlarını ve C++ görsel fonksiyonlarını kullanır. Bu arada yönetim MFC sisteminin elindedir. Programcı tarafından yazılan kod framework tarafından uygulamaya yönelik eventlerin handle edilmesi içi kullanılır. Örnek olarak kullanıcı menüden seçim yapmışsa framework bu komutu bir dizi C++ nesnesi içinde döndürür. Bu nesneler aktif view ve frame window, viewın bağlı olduğu document, dökümanın document template'i ve application nesnesi olarak verilebilir. Eğer bu nesnelerden herhangi biri komutu handle ederse ilgili mesaj kotarma fonksiyonunu çağırır. Verilen herhangi bir komut için çağırılan kod programcının veya framework'ün yazdığı kod olabilir. Bu bağlam klasik Windows programlama teknikleri ve event driven(olay sürümlü)

programlama tekniklerinden çok farklı bir yapı göstermemektedir. Dolayısıyla bu alanlarda tecrübesi olan programcılara çok farklı görünmeyecektir.

3.2.9 Documents, Views, Frame Windows, Templates ve Application Nesneleri Arasındaki İlişkiler

Document/View yaratım işlemini anlayabilmek için çalışan bir program gözönüne alınabilir. Uygulamada bir application nesnesi, bir document, bir frame window ve bir view bulunduğu varsayımı ile:

- Document kendisine bağlı bir view listesi ile kendisini yaratan document template'ine bir işaretçi tutar.
- View bağlı olduğu document'a ait bir işaretçi tutar ve parent frame pencerenin bir child'ıdır.
- Document frame window o anda aktif olan view'a ait bir işaretçi tutar.
- Document template açık tüm dökümanların bir listesini tutar.
- Application tüm document templatelerinin bir listesini tutar.
- Windows tüm açık pencerelerin bilgisini tutar ve onlara mesaj gönderir.

Tüm bu ilişkiler document/view yaratımı sırasında kurulur [3].

Tablo 3-2 How to Access Other Objects

From Object	How to Access Other Objects
Document	Use GetFirstViewPosition and GetNextView to access the document's view list.
	Call GetDocTemplate to get the document template.

View	Call GetDocument to get the document. Call GetParentFrame to get the frame window.
Document frame window	Call GetActiveView to get the current view.
MDI frame window	Call MDIGetActive to get the currently active CMDIChildWnd.

Tipik olarak bir frame window'un bir view'ı vardır. Fakat bazen bölünmüş ekranlarda aynı frame window'un birden fazla view'ı olabilir.Frame window aktif view'ına bir işaretçi tutar ve bir başka view aktif olduğunda değiştirir [2].

3.3 Nesneye Yönelik Programlama -NYP (Object Oriented Programming - OOP)

"Nesneye Yönelik Programlama" (Object Oriented Programming), veya kısaca NYP, programcılıkta yeni bir yaklaşımdır. Programlama dillerinin ortaya çıktığı ilk yıllardan bu yana "programlama", hep değişik teknikler aracılığı ile gerçeklenmiştir. Programcılığın gelişim sürecindeki her kritik noktada, gittikçe kompleks hale gelen problemleri çözmek için programcının önüne yeni metodlar sürülmüştür. Bunun en önemli sebeplerinden biri gittikçe karmaşıklaşan program isteklerine cevap verebilmektir [4].

İlk programlar, bilgisayarın ön panelindeki anahtarların değiştirilmesi ile yapılıyordu. Dolayısıyla bu metod, sadece küçük programlar için uygundu ve bu tip programların yazılabilmesine olanak tanıyordu. Daha sonraları "Assembly" dili bulundu ki, bu dil daha uzun programların yazılabilmesini mümkün kılıyordu. Bundan sonraki gelişme, 1950'li yıllarda ilk yüksek seviyeli dil (High-Level Language) olan FORTRAN dilinin icat edilmesi ile ortaya çıktı.

Yüksek seviyeli bir dili kullanarak, programcı binlerce satır uzunluğunda kaynak koda sahip olan programlar yazabilir. Nispeten kısa programlar için mükemmel bir teknik olan yüksek seviyeli dil ile programlama, çok uzun programlarda okunması zor (aynı zamanda kontrol altında tutulabilmesi de güç) "spagetti kod" oluşmasına sebep oluyordu. Bu "spagetti kod" un yok edilmesi, 1960'lı yıllarda geliştirilen "yapısal programlama dilleri" ve " yapısal programlama tekniği" ile mümkün oldu. "Algol" ve "Pascal" dili, bu teknik ile geliştirilen ilk dillerdi. "C" dili de bu tip programlama dilleri arasındadır. Yapısal programlamanın en belirgin özelliklerini, hatasız tanımlanmış kontrol yapıları, blok kodlar, GOTO teriminin kullanılmaması (veya olabildiğince az kullanılması) ve yerel değişkenler ile rekürsif yapıyı destekleyen altprogramlar olarak belirtmek mümkündür. Yapısal programlama tekniğini kullanarak, sıradan bir programcı 50.000 satıra ulaşan programlar yazabilir.

Kompleks ve uzun olan programlar için uygulanan yapısal programlama tekniği, her ne kadar mükemmel sonuç veriyorsa da yine belli bir limiti aşan programlar için yetersiz kalıyordu. Çok daha kompleks olan programların yazılabilmesi için -ki teknolojik gelişim bu tip programları yazılmasını gerekli kılıyordu - programlama işine yaklaşımda yeni bir metoda yönelmek gerekiyordu. Son aşama olarak "Nesneye Yönelik Programlama" kavramı ortaya çıktı. NYP tekniği, yapısal programlamanın kurallarını güçlü ve yeni olan başka kavramlarla birleştirerek, programların çok daha değişik bir biçimde oluşturulmasına olanak tanır. NYP, programcıyı var olan problemi birbiri ile ilişkisi olan alt gruplara ayırmasına teşvik eder. Her alt grup, bir "nesne" yi teşkil eder ve her nesnenin kendine ait değişkenleri, verileri ve fonksiyonları vardır. Bu şekilde komplekslik azaltılmış olur ve programcı daha uzun programları rahatlıkla kontrol altında tutabilir.

"C++" dilini de içine alan NYP dilleri, üç ortak özelliği bünyesinde barındırırlar [4]:

- Verinin Kapsanması (Encapsulation)
- Çokşekillilik (Polymorphism)

- Devralma (Inheritance)

3.3.1 Nesne ve Verinin Kapsanması

Yazılım dünyasındaki nesneler (objects), gerçek dünyadaki varlıklar gibi davranış gösteren program parçaları ve verileridir. Bir nesnede onu tanımlayan bilgiler, veriler ve bunlar üzerinde işlem yapmaya yarayan metodlar bulunur. Nesnelerin en büyük avantajı, birçok kereler kullanılabilir olmasıdır. Böylelikle önceden hazırlanmış, test edilmiş ve çalışır hale gelmiş programlar daha sonra çok kısa bir sürede yeni bir programa uyarlanabilir. Nesnelere mesaj göndermek ve bu mesajlara yapılan işlemlerin sonuçlarını görmek mümkündür.

Nesneler kullanıldıkları yerlere göre dört kategori halinde incelenebilirler:

Denetim : Nesneler, bir programın normal akışı sırasında bilgi işleme denetim elemanı olarak görev yapabilirler. Program içinde çalışacak olan sistem, bir nesne ile gösterilir ve denetim bu nesnenin fonksiyonları ile sağlanır.

Uygulama : Bir sistemde belirli amaçların gerçekleştirilmesi için nesneler kullanılabilir. Çeşitli varlıkların bilgisayar ortamında gösterimi nesnelerle yapılabilir. Bu daha kolay bir soyutlama ve kapsama sağlar.

Veri Yönetimi : Üzerinde işlem yapılacak veriler nesne haline getirilir ve bu nesneler üzerinde yapılan işlemler metod olarak tanımlanır. Nesneye yönelik veri tabanı yönetim sistemi buna bir örnektir.

Arabirim : Kullanıcı ile uygulama arasındaki arabirim nesnelerle sağlanabilir. Bu da her iki düzey arasında bağımsızlık yarattığından herhangi birinde yapılacak değişiklik veya gelişim diğerini de etkileyecek büyük sorunlar çıkarmaz.

Nesne içinde bir kısım veri veya program kodu özel (private) kısımda yer alır. Bu kısım sadece o nesne tarafından kullanılabilir, nesne dışından erişilemez . Özel kısımdaki bu elemanlara erişmek, ancak genel (public) kısımda bulunan üye fonksiyonlar (member functions) ile mümkündür. Bu şekilde kullanıcının nesnenin özel kısmını yanlış bir şekilde kullanması veya değiştirmesine karşı önlem alınmış

olur. Nesnenin korunmuş (protected) kısmında yer alan elemanlar, bu nesneden devralma yolu ile türetilen alt nesneler tarafından erişilebilirler. Program kodu ve verinin bir nesne içinde bu şekilde birbirine bağlanmasına **kapsama** (encapsulation) denir. Özel ve korunmuş kısımlarla sağlanan bu kapsama özelliği istendiğinde friend kullanımı ile kaldırılabilir. Kapsamanın desteklenmesi ile güvenilirlik artar ve tutarlılık korunur.

3.3.2 Çokşekillilik

Çokşekillilik (polymorphism), bir ismin birbiriyle bağlantılı fakat aslında değişik olan birden fazla amaç için kullanılmasıdır. Çokşekilliğin amacı, bir tek ismi genel bir işlevler kümesini tanımlamada kullanmaktır. Hangi işlevin gerçekleşeceği, üzerinde işlem yapılacak veri tipi tarafından belirlenir. Örneğin, değişik veri tiplerinin taşıdığı bilgileri, yaygın programlama dillerinden birini kullanarak ekrana yazdırmak için şu şekilde bir yordam tanımlanabilir:

```
procedure Çiz( x: değişen _ kayıt)
begin
   case x.tip_adı is
            daireT    :  daire_çiz(x.daire)
            çokgenT :   çokgen_çiz(x.çokgen)
            eğriT      :  eğri_çiz(x.eğri)
   end case;
end;
```

Bu örnekteki altprogram, ilgili nesnelere çizdirme mesajını gönderdiği için bu yöntem gönderici tarafından yönetilen bir denetim biçimidir. Çokşekilliliğin kullanılması halinde bu yordam

Çiz(x)

şeklinde yaratılabilecektir. Bu durumda her bir tip ayrı sınıflara gönderilecek ve her sınıfın kendine ait Çiz( ) fonksiyonu olacaktır. Bu yöntem alıcı tarafından yönetilen bir denetime sahiptir. Dikkate alınması gereken nokta, nesne sayısının çok fazla olduğu durumlarda yukarıda case yapısının daha da büyümesidir. Her yeni tip ilave edilmesi gerektiğinde, bu fonksiyonun tekrar değişmesi gerekecektir [4]. Oysa çokşekillilik bize var olan kod üzerinde değişiklik yapılmamasını sağlar; yalnızca yeni eklenen nesneye yeni bir fonksiyon ilave etmek yeterli olur.

Çokşekillilik, yorumcularda (interpreter) çalıştırma anında (run-time) sağlanır. C++, derlenerek kullanılan bir dil olduğu için çokşekillilik, hem derleme hem de çalıştırma anında kullanılır.

C++ dilinde çokşekillilik türetilmiş sınıflar ve sanal (virtual) fonksiyonlarla elde edilir. Bir sanal fonksiyon temel sınıf (sınıf, asal olarak "struct" veri yapısına benzeyen ve nesnelerin özelliklerini belirleyen yapılardır) içinde **virtual** olarak bildirilmiş ve bir ya da daha fazla sınıf içinde yeniden tanımlanmış bir fonksiyondur. Bir sanal fonksiyon temel sınıf işaretçisi kullanarak erişildiğinde, C++ hangi fonksiyonu kullanacağını, nesnenin tipine bağlı olarak çalışma anında belirler.

3.3.3 Devralma

Devralma (inheritance), bir nesnenin bir başka nesneye ait özellikleri kullanabilmesi demektir. Bu özellik, sınıflandırma kavramını sağladığı için önemlidir. Gerçek hayatta da birçok sınıflandırma örnekleri görmek mümkündür. Sarı Uçak, uçak sınıfından belirli bir nesnedir. Uçak sınıfı hava taşıtları sınıfına ait, o da daha büyük bir sınıf olan taşıt sınıfına aittir. Sonuç olarak Sarı Uçak da bir

taşıt olduğundan en üst temel sınıfı olan taşıt sınıfının bazı özelliklerini (örneğin tekerlek sayısı özelliği) almıştır. Bu sınıflandırma olmasaydı, her nesne tüm özelliklerini açıkça belirtmek zorunda kalırdı. Fakat sınıflandırma kullanarak bir nesnenin yalnızca kendisini o sınıf içinde ayrıcalıklı yapan özellikleri belirtmek yeterlidir. Örnek olarak bir sınıf bildirimini basit bir şekilde ele alalım :

```
class Araç
{
    int tekerlek;
    int ağırlık;
public:
    void  TekerlekAdedi (int a);
    int    KaçTekerlek( );
    void  Ağırlık (float a);
    float  NeKadarAğır ( );
};
```

Bu sınıf bir araç tipini genel olarak tanımlar. Bu sınıftan devralma yoluyla başka sınıflar türetmek mümkündür.

```
class O tobüs  : public Araç
{
```

```
    int yolcu;
public:
    void YolcuAdedi (int a);
    int    KaçYolcu( );
};
```

Otobüs yolcu taşıyan bir araçtır. Genel Araç sınıfına yalnızca yolcu isimli bir üye eleman ekleyerek, daha genel olan Araç sınıfını kullanmak mümkündür.

Türetim işlemini gerçekleştiren devralmanın genel şekli şöyledir:

```
class yeni_sınıf_adı : erişim_şekli ana_sınıf_adı {.....};
```

Burada kullanılabilen üç tip erişim şekli vardır: genel **(public)**, özel **(private)** ve korunmuş **(protected)**.

Devralma hiyerarşisinin açıkça görüldüğü bir örnek de iki boyutlu şekiller olabilir. Aşağıdaki şekilde sınıflar arası hiyerarşi görülmektedir.

Sınıflar

Bir nesneye yönelik programlama dilinin en önemli özelliklerinden biri sınıftır (class). Bir nesne (object) yaratmat için önce onun bir genel formunu **class** sözcüğü ile tanımlamak gereklidir.

Bir sınıf asıl olarak bir veri yapısına (structure) benzer. Aşağıdaki örnekte bir kuyruk (queue) sınıfı yaratılmaktadır:

```
class Kuyruk  {
```

```
    int k[100];
    int eleman;
public:
    void Boşalt(void);
    void Koy(int i);
    int Çıkar(void);
    int Eleman_sayısı( );
};
```

Temel olarak bir sınıf, tanımında 3 kısıma sahip olabilir : **private** (özel), **public** (genel) ve **protected** (korunmuş). Sınıf içinde tanımlanan her eleman otomatik olarak (default) "private" durumundadır. Yukarıdaki örnekte k ve eleman isimli değişkenler "private" sözcüğü kullanılmadığı halde özel durumundadırlar. Bu kısımlar, o sınıfa ait olmayan hiçbir fonksiyon tarafından erişilemez. Böylelikle NYP'nin temel ilkelerinden biri olan Kapsama Kuralı (encapsulation) uygulanmış olmaktadır. Bazı veri elemanları ve fonksiyonlar özel yapılarak erişimleri kısıtlanır. Aynı şekilde fonksiyonlar da özel olarak tanımlanabilir. Bu tip fonksiyonlar sadece o sınıfa ait diğer üye fonksiyonlar tarafından kullanılabilir.

Bir sınıfın bazı kısımlarının programın diğer kısımlarından da erişilebilmelerini sağlamak için **public** sözcüğü kullanılır. Sınıf içinde "public" sözcüğünden sonra tanımlanan her şey sınıf dışından erişilebilir. NYP'nın ilkelere gereği, bütün veriler **private** yapılmalı ve bu verilere erişim **public** fonksiyonlar ile sağlanmalıdır.

Sınıfların **protected** (korunmuş) kısımlarındaki veri ve fonksiyonlara ise yalnızca o sınıftan türetilen alt sınıflar doğrudan erişebilir, başka sınıflar erişemez.

Örnekte yer alan Eleman_sayısı( ), Koy ( ) ve Çıkar( ) fonksiyonlarına **üye fonksiyonlar** (member functions) denir. Sınıf içinde tanımlanan fonksiyonlar o sınıfın private kısımlarına erişebilirler.

Bir sınıf **class** sözcüğü kullanarak tanımlandıktan sonra o sınıf tipinde bir nesne yaratılabilir:

```
Kuyruk int_kuyruğu;
```

Sınıf bildiriminin genel şekli şöyledir:

```
class sınıf_adı {
   // özel veriler ve fonksiyonlar
public:
   // genel veriler ve fonksiyonlar
protected:
   // korunmuş veriler ve fonksiyonlar
}nesne_listesi;
```

Fonksiyonların asıl kodu yazılacağı zaman derleyiciye hangi sınıfa ait olduğu belirtilmelidir. Bunun için görünürlük operatörü (::) kullanılır.

```
void Kuyruk::Koy(int I )
{
```

```
...
}
```

Programın herhangi bir yerinden belirli bir üye fonksiyonun çağırılması ise nokta operatörü (.) ve nesne adı ile yapılır.

```
Kuyruk a.b;
int x,s;
a. Koy(x);
s = b. Eleman_sayısı( );
```

Gerçek C++ uygulamalarında programa ait sınıf tanımlamaları bir başlık dosyası içine, bu sınıfların üye fonksiyonlarının kodu da ayrı bir kaynak kod dosyası içine konur.

3.3.4 Nesne Yapıcı Ve Yokediciler

NYP'nin en önemli özelliği nesnelerle işlem yapılmasıdır. Bu nesnelerin yaratılması veya yokedilmesi belirli kurallarla gerçekleştirilir. Nesneye yönelik programlama dillerinde nesneler yapıcı fonksiyonlarla (constructor) yaratılır ve yokedici fonksiyonlarla (destructor) bellekten silinirler. Programın çalışması sırasında bu fonksiyonlar kendiliğinden çağrılarak işlerini yaparlar. Programcının yapması gereken birçok işlem bu şekilde otomatik olarak yapılır.

3.3.5 Yapıcılar

Yapıcılar genellikle yeni bir nesne yaratmak ve bu nesneye ait özel değişkenlere ilk değerlerini atamak için kullanılırlar. Bir nesnenin elemanlarına ilk değer atanması iki şekilde olur. Birincisi yapıcının gövdesi içinde normal atama operatörü kullanmaktır. İkincisi de ilk değer listesi kullanmaktır. İlk değer listesi yapıcının

argümün listesinin hemen ardından ve gövdesinin hemen önünde yer alan ilk değer verilecek üyelerin bir listesidir. Bu ifade ':' operatörü ile belirtilir. İlk değer listesi kullanan bir yapıcı fonksiyonun genel şekli şöyledir :

```
sınıf_adı(argümanlar) : eleman_adı(ilk_değer), eleman_adı(ilk_değer),...
{
    // fonksiyon gövdesi)
}
```

Yapıcılar bir nesne bildirimi yapıldığı anda otomatik olarak çağırılırlar. Çağrılma şekli için bir örnek şu şekilde olabilir :

```
class RasyonelSayı {
private:
    int pay, payda;
public:
    RasyonelSayı(int a, int b);  // yapıcı
    ...
    // diğer üye fonksiyonlar
};
RasyonelSayı::RasyonelSayı(int a, int b)
{
    pay = a;
```

```
    payda = b;

}
```

Böyle bir sınıfa ait nesnenin bildirimi de şu şekilde olabilir:

```
RasyonelSayı rs = RasyonelSayı(3,5);
```

Daha kısa olarak şu bildirim de yapılabilir:

```
RasyonelSayı rs(3,5);
```

Bu gösterilen yöntem, yapıcının kurulması ve kullanımı için en temel ve kullanılan yöntemdir. Bundan başka yapıcıların kullanımı için pek çok yöntem daha vardır.

Yapıcıların kullanımı konusunda iki ana kısıtlama vardır. Bunlardan birincisi yapıcıların **void** dahi olsa geriye bir değer döndürmemeleri, ikincisi ise sanal (virtual) olmamalarıdır.

3.3.6 Yokediciler

Yokediciler, yapıcıların yaptığı işin tersini yaparlar. Program akışı sırasında varlığına artık ihtiyaç duyulmayan nesnelerin bellekten temizlenmesi işleminde yokediciler kullanılırlar. Yokedici, yapıcı ile aynı adı taşıyan ve önünde _ işareti bulunan bir üye fonksiyondur.

Bir yokedici fonksiyonun genel bildirim şekli şöyledir:

```
sınıf_adı(void)
```

Özel olarak bir yokedici tanımlanmazsa bazı durumlar için derleyici otomatik olarak bir yokedici kodu üretir. Genel bir kural olarak, **new** anahtar sözcüğü kullanılarak yaratılan bellek parçasının silinmesi için **delete** sözcüğü içeren bir yokedici fonksiyon tanımlanır. Yokediciler programcı tarafından açık olarak çağrilmazlar. Yokedici kullanılması gereken yerlerde (örneğin delete sözcüğü

kullanılırsa), derleyici, kullanılan nesnelere ait yokedici olup olmadığına bakar, varsa bunları otomatik olarak çağırır, yoksa kendisi bir kod üretir.

Yokedicilerin bildirminde dikkat edilmesi gereken üç önemli nokta vardır :

- Bir sınıfın birden fazla yokedicisi olamaz.
- Yokediciler herhangi bir argüman almazlar.
- Yokediciler **void** dahil hiçbir tipte değer döndürmezler.

3.3.7 Ekkullanım

NYP çok önemli bir özelliğe daha sahiptir. Bu da ekkullanımla yüklemedir (overloading). Ekkullanım, iki şekilde olabilir. Birincisi, operatörlerin yüklenmesi; ikincisi de fonksiyonların yüklenmesidir.

3.3.7.1 Operatör Ekkullanımı

Nesneye yönelik programlama dillerinde aritmetik işlem ve diğer bazı operatörler, asıl işlevlerinden başka bir görevi yerine getirmek için de kullanılabilirler. Buna operatör ekkullanımı (operator overloading), operatörün fazladan yüklenmesi veya kısaca operatörün yüklenmesi denir. Örneğin, normal olarak + operatörü, solunda ve sağında bulunan iki sayıyı toplamak için kullanılır. Aynı operatör örnek olarak iki kompleks sayının toplanması için veya iki kümenin birleşimi için de kullanılabilir. Başka bir örnek olarak özel değişkenleri lira ve kuruş olan bir **para** sınıfının tanımlanıdığı kabul edelim. ++ operatörünü de kuruşun 1 artımı olarak yükleyelim:

```
class Para{

..

..

public
```

```
Para operatör++(); // operatör ekkullanımı
..
}
Para Para::operatör++()
{
    kuruş++;
    if (kuruş>99)
{
        lira++;
        kuruş = 0;
}
        return *this;
}
```

Böylelikle artık ++ operatörünü program içinde lira hesabına girmeden kuruşu bir arttırmak üzere kullanabiliriz. Operatör ekkullanımının mümkün olduğu ve mümkün olmadığı operatörler aşağıdaki tablolarda verilmiştir.

Tablo 3.3 Ekkullanımı mümkün olan operatörler

+	-	*	/	%	^	&

|	-	!	,	=	<	>
<=	>=	++	--	<<	>>	= =
!=	+=	- =	/=	*=	%=	^=
&=	|=	<<=	>>=	&&	||	( )
-	->	->*	new	delete		

Aşağıdaki tabloda da ekkullanımı mümkün olmayan operatörler verilmiştir. * operatörü yukarıda çarpma anlamındadır, aşağıda ise işaretçi anlamındadır.

Tablo 3.4 Ekkullanımı mümkün olmayan operatörler

	::	*	?
?:	sizeof	#	##

3.3.7.2 Fonksiyon Ekkullanımı

Nesneye yönelik programlama dillerinde bir fonksiyonu diğerlerinden ayırt etmek için adından başka parametre tipi ve sayısı da kullanılır. Bu şekilde aynı ada sahip, fakat farklı argümanlar alan fonksiyonlar yaratılabilir. Bunlar derleyici tarafından değişik olarak nitelendirilir. Bu işleme fonksiyon ekkullanımı (function overloading) denir. Bu özellik yazılım geliştirme işinde programcıya yardımcı olur. Kullanılan fonksiyon isimlerinin aynı olması programcının takip etmesi gereken isim sayısını azaltır. Yazılan programın anlaşılabilirliği artar, takip edilmesi kolaylaşır.

3.3.8 NYP İle Yazılım Geliştirme

Son yıllarda programcılıkta yeni bir devrim olarak nitelendirilen NYP ile yazılım geliştirmek sanıldığı kadar zor değildir. Bir proje için tasarım yapılırken bir tek sınıfı tek başına düşünerek tasarlamaya çalışmak uygun bir yöntem değildir, çünkü bir sınıf tek başına var olamaz. Mantıksal bağlantı ancak diğer sınıflarla birlikte ele alındığında tanımlı hale gelir. Bazen bir yazılım parçası içindeki tüm sınıflar bir tek hiyerarşi oluştururlar. Bu sınıflar kümesine birim (component) veya kütüphane (library) denir. Bir birim tasarlanırken kısaca şu aşamalardan geçilir:

Kavram ve sınıflarla bunlar arasındaki temel ilişkilerin belirlenmesi.

- Sınıflar üzerindeki işlemlerin belirlenmesi ve geliştirilmesi.
- Sınıfların diğer sınıflara bağlılık durumlarının belirlenmesi.
- Sınıflar arasındaki arabirimin belirlenmesi.

Böylelikle ortaya çıkan birimler projenin kilometre taşlarını meydana getirirler. Yazılımı geliştirirken NYP'nin kesin kurallarına uymak, ortaya çıkacak sonucun etkinliği ve güvenilirliği açısından son derece önemlidir. NYP, bu yolda sadece bir araçtır.

3.4 The Microsoft Foundation Class Library MFC

Görsel programlama dillerinin nesne yönelimli programlama ilkelerine göre çalıştığı gözönüne alınırsa doğru yerlerde doğru nesne sınıflarına erişebiliyor olması uygulama geliştirme performansı açısından önem kazanmaktadır. Bu da normal olarak zaman içinde kullanılan sınıfların saklanarak yeni uygulamalarda tekrar kullanılabilmesi gereğini oluşturur. Düzenli bir şekilde kataloglanmayan sınıflara erişim oldukça zor olacak ve yeni uygulamalarla birlikte bu kütüphaneyi güncel tutabilmek zorunlu ve gerçekten zorlu bir uğraş haline gelecektir. İşte bu amaçla çeşitli firmalar gerekli sınıfları sağlayan class paketleri geliştirmişlerdir. Bunlar arasında en popüler ve Windows altında görsel kullanıcı arabirimli uygulama geliştirilebilmesi için en güçlü kütüphanelerden biri The Microsoft Foundation

Class Library MFC' dir. Visual C++ tarafından standart olarak sağlanan bu paketin içinde programcıya gerekli çok çeşitli sınıflar kullanılmaya hazır olarak sunulmuştur.

3.4.1 MFC'nin Avantajları

Microsoft Foundation Class Library Windows'a gerçek anlamda object oriented arayüzü aşağıdaki avantajlarla sağlar [1]:

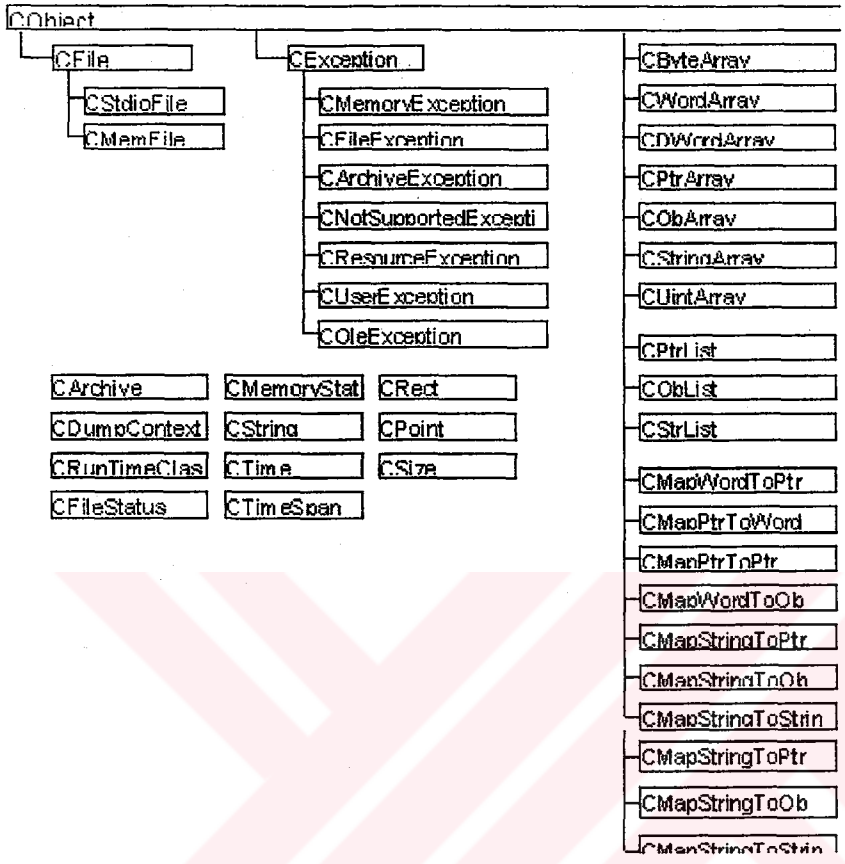
- Windows bazlı bir uygulama programlamanın getirdiği eforda belirgin bir düşüş
- C-bazlı API'lerle karşılaştırılabilir çalışma hızı
- Kod boyutunun minimize edilmesi
- Herhangi bir Windows C fonksiyonunu direk olarak çağırabilme
- Varolan C uygulamalarının C++'a kolayca aktarılabilmesi
- Daha önceki C bazlı Windows programlama desteğinden faydalanabilme
- Windows API fonksiyonlarının C++ ile C'den daha kolay çağırılabilmesi
- C++ dilinin özelliklerini etkin olarak kullanabilecek doğru Windows API

Görsel programlama dillerinin çalışma platformu Windows olduğundan MFC yardımcı kütüphanesindeki sınıfların büyük çoğunluğu Windows altında çalışan programlar için tasarlanmıştır. Kullanılan bu sınıflar ile MFC Windows altında bir "application framework" yaratır. Bundan sonra programlama için gerekli olan tüm sınıflar arasındaki ilişkilerin tanımlanması ve uygulamaya yönelik kodların yazılmasıdır. MFC aşağıdaki kategorilerde sınıf kütüphanesi sağlar:

- Root Class
- Application Architecture Classes

- Windows Application Class
- Command-Related Classes
- Document/View Classes
- Visual Object Classes
- Window Classes
- View Classes
- Dialog Classes
- Control Classes
- Menu Class
- Device-Context Classes
- Drawing Object Classes
- General-Purpose Classes
- File Classes
- Diagnostics
- Exceptions
- Collections
- Miscellaneous Support Classes
- Macros and Globals

Yukarıdaki sınıfların bazıları genel amaçlı sınıflar olup hem framework hem de MS-DOS programları için kullanılabilirler.



Şekil 3-2 Class Hiyerarşisi

3.4.1.1 Root Class

Microsoft Foundation Class Library içindeki sınıfların pek çoğu sınıf hiyerarşisi üzerindeki bir base sınıftan türetilirler. CObject sınıfı kendi türevi tüm sınıflara çok az bir performans kaybı ile bir çok ek özellikler getirir.

3.4.1.2 CObject

Hemen hemen tüm sınıfların en üst seviyesindeki sınıftır. Verinin diske saklanması ve okunması (serializing) ve sınıf hakkında run-time informasyon alınması işlemlerini destekler.

3.4.1.3 Application Architecture Classes

Bu kategorideki sınıflar Windows altında çalışan uygulamaların içerdiği fonksiyonları sağlarlar. Framework tarafından sağlanan bu ortak özelliklere ek olarak uygulamaya yönelik kodlar sisteme eklenir. Bu işlem tipik olarak bu uygulama mimarisi sınıflarından birinin base sınıf olarak alınıp uygulamada istenen özelliklere bağlı olarak yeni member değişkenlerin eklenmesi ve varolan member fonksiyonlarının override edilmesi şeklinde gerçekleşir. Windows altında çalışan bir uygulama içi anahtar sınıflar aşağıda verilmiştir;

- CWinApp sınıfından türetilen bir uygulama nesnesi
- Cdocument sınıfından türetilen bir veya daha fazla döküman nesnesi
- CView'dan türetilen bir veya daha fazla View nesnesi, bu viewların herbiri bir döküman nesnesine ve dolayısıyla bir window'a bağlıdır.

3.4.1.4 Windows Application Class

Her uygulama bir ve yalnız bir uygulama nesnesine sahiptir. Bu nesne CWinApp'dan türetilir ve çalışan program içindeki tüm diğer nesneleri çalışmasını koordine eder [2].

3.4.1.5 CWinApp

Uygulamanın çalışmaya başlaması, çalışması ve çalışmasının bitirilmesi için kodlar içerir.

3.4.1.6 Command-Related Classes

Kullanıcı uygulama ile; menülerden seçim yaparak veya fare ile tuşlara basarak etkileşimde bulunur. Bu etkileşim ile uygulama etkileşimde bulunulan nesneden ilgili bir command-target nesnesine mesajlar gönderir. Command-target sınıflar CCmdTarget sınıf'ından türetilmişlerdir ve CWinApp, CWnd, CDocTemplate, CDocument, Cview sınıfları bu sınıftadırlar. CCmdUI sınıfı nesnenin durumunu

güncelleştirmek için menü veya bir tuş şeklinde bir command user-interface nesnesi tanımlar.

CCmdTarget

Mesaj alan ve aldığı mesaja yanıt veren tüm nesne sınıfları için base sınıf oluşturur.

CCmdUI

Menü seçenekleri ve araç çubukları gibi kullanıcı arayüzü nesnelerinin seçilme, aktif/pasif duruma geçmeleri için gerekli programatik arayüz sağlar.

3.4.1.7 Document/View Classes

Document template nesneleri tarafından yaratılan document nesneleri uygulamanın üzerinde çalıştığı veriyi yönetir. View nesneleri ise document nesnesinin içerdiği veriyi kullanıcıya gösterir ve kullanıcının bu veri ile etkileşimini sağlar.

CDocTemplate

Document template'lerinin base sınıfıdır. Document template nesnesi document, view, and frame window nesnelerinin yaratılmalarını koordine eder.

CSingleDocTemplate

SDI- Single Document Interface - Tekil Döküman Arayüzü tipli döküman nesneleri için bir şablon yapı verir. SDI tipli uygulamalar bir anda ancak tek bir döküman açabilirler.

CMultiDocTemplate

MDI- Multiple Document Interface - Çoklu Döküman Arayüzü tipli döküman nesneleri için bir şablon yapı verir. MDI tipli uygulamalar bir anda birçok döküman açabilirler.

CDocument

Uygulamaya özel, döküman sınıfları için bir şablon sınıf verir. Programcı kendi döküman sınıfları için base sınıf olarak CDocument sınıfını kullanır.

CView

Uygulamaya özel view sınıfları için base sınıftır. Döküman verisini gösterir ve kullanıcıdan seçme ve yazma gibi işlemler için etkileşim alır.

CCreateContext

Document. view ve frame window sınıflarının yaratılmını koordine edebilmek için document template nesnesinden window-yaratım fonksiyonlarına parametre olarak geçen bir yapıdır.

3.4.1.8 Visual Object Classes

Bu kategorideki sınıflar görsel kullanıcı-arayüzü nesneleridir. Bu sınıf içine pencereler, diyalog kutuları, kontroller ve menüler girer. Ayrıca bu nesnelerin ekran üzerinde görüntülenmesi için yardımcı görev alan device context ve kalem ve fırça gibi çizim elemanları da bu kategoriye girerler.

3.4.1.9 Window Classes

CWnd ve bundan türetilmiş sınıflar Windows penceresine atanmış bir seri numarası (HWND) alırlar. CWnd sınıfı kendi başına kullanılabilir veya diğer bir sınıfa base sınıf olarak tanımlanabilir. MFC tarafından sağlanıp Cwnd sınıfından türetilen Window sınıfları çok değişik tipli pencereler yaratılabilir.

CWnd

Tüm pencereler için base sınıftır. Direk olarak bu sınıf kullanılabilir veya aşağıdaki türev sınıflar kullanılabilir.

CFrameWnd

Bir SDI tiple uygulamanın ana penceresinin base sınıfıdır.

CMDIFrameWnd

Bir MDI tiple uygulamanın ana penceresinin base sınıfıdır.

CMDIChildWnd

Bir MDI uygulamanın döküman frame pencereleri için base sınıftır.

3.4.1.10 View Classes

CView ve Cview'dan türemiş sınıflar frame window'un client bölgesini belirlerleyen çocuk pencerelerdir.

CView

Döküman verisinin uygulamaya yönelik görüntülerini oluşturan view'lar için base sınıftır. Programcı view sınıfları için CView sınıfını, otomatik kaydırma (scrolling) yeteneğini de istiyorsa CScrollView sınıfını base sınıf olarak kullanmalıdır.

CScrollView

Kaydırma yeteneği olan viewlar için base sınıftır.

CFormView

Form görüntüsü bir diyalog resource'unda tanımlı olan kaydırılabilir bir viewdır.

CEditView

Tekst-edit, arama, değiştirme ve kaydırma yeteneklerine sahip bir view sınıfıdır. Eğer döküman verisi tekst bazlı işlemler gerektiriyorsa tercih edilmelidir.

3.4.1.11 Dialog Classes

CDialog ve türetilmiş sınıfları diyalog-kutusu fonksiyonalitesini barındırırlar. Diyalog kutusu da özel bir çeşit pencere olduğundan CDialog sınıfı da CWnd sınıfından türer. Windows sistemi altında tanımlı bazı genel diyalog kutusu tipleri

de bulunmaktadır. Bunlar dosya açma, kayıt etme, yazma ve font veya renk belirleme gibi genel amaçlar için kullanılırlar. Dolayısıyla uygulama geliştirici; kendi amacına uygun olarak CDialog sınıfından bir diyalog kutusu yaratabileceği gibi bu genel amaçlı sınıflardan da yararlanabilir.

CDialog

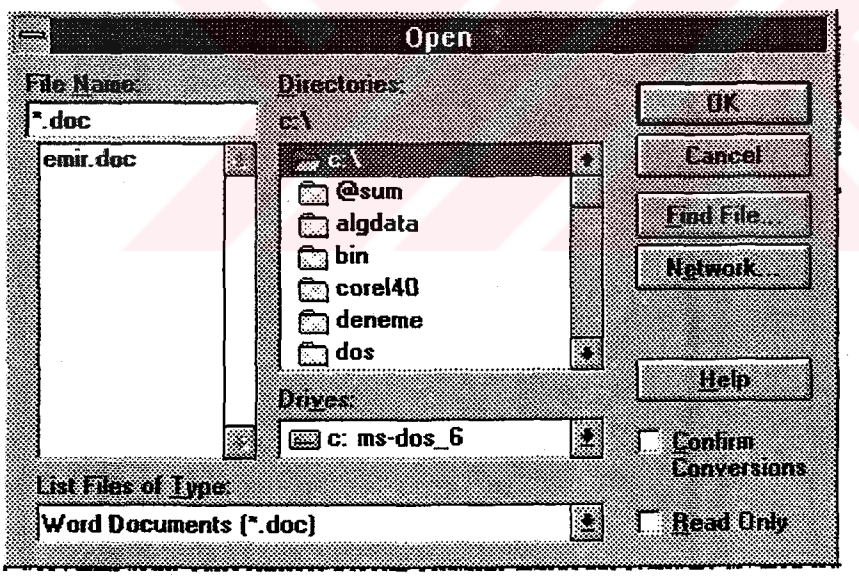
Tüm diyalog kutuları için base sınıf modal ve modeless'tir.

CDataExchange

Diyalog kutuları için başlama ve sona erme ile ilgili onformasyon sağlar.

CFileDialog

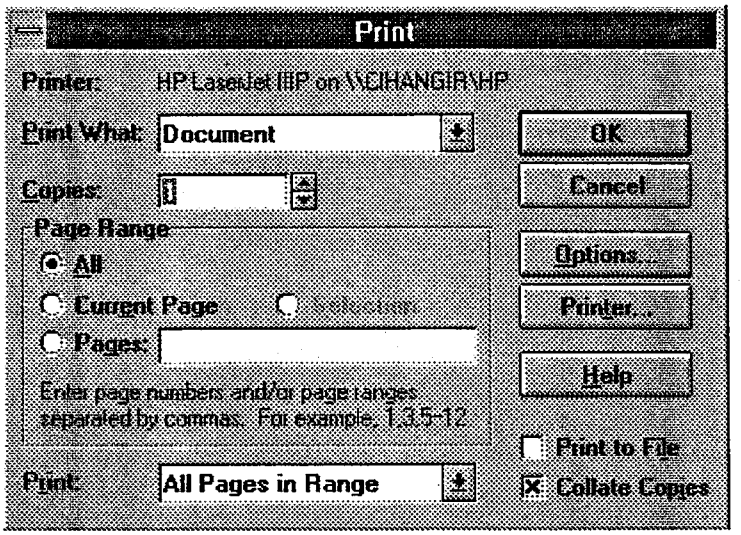
Dosya açma veya kayıt etme için Windows standart diyalog kutusunu açar.



Şekil 3-3 Windows Standart File Open Diyalog Kutusu

CPrintDialog

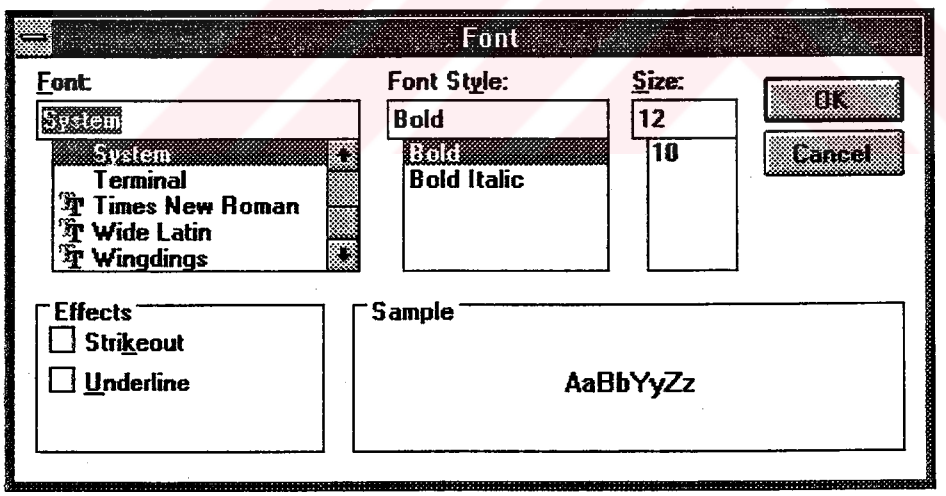
Bir dosyanın print edilebilmesi gerekli bilginin alınacağı standart diyalog kutusunu açar.



Şekil 3-4 Windows Standart Print Diyalog Kutusu

CFontDialog

Font seçimi için gerekli standart diyalog kutusunu açar.



Şekil 3-5 Windows Standart Font Diyalog Kutusu

CColorDialog

Renk seçimi için gerekli standart diyalog kutusunu açar.

CFindReplaceDialog

Tekst içinde arama ve değiştirme fonksiyonunu gerçekleştirmek için gerekli standart diyalog kutusunu açar.

3.4.1.12 Control Classes

Kontrol sınıfları butonlar, listeler, combo box'lar gibi standart Windows kontrolleri ile birlikte resimli butonlar, pen computing için gerekli kontroller ve VBX custom kontrolleri gibi yeni kontrolleri de içerir.

CStatic

Sabit metin içeren bir kontrol penceresidir. Genelde veri giriş veya gösterme ortamlarının etiketi olarak kullanılırlar.

CButton

Buton kontrol penceresidir. Bu sınıf pushbutton, check box veya radio button kontrollerini içerir.

CEdit

Metin-yazılabilir kontrol penceresidir. Kullanıcıdan metin girişi için kullanılır.

CScrollBar

Belli bir aralık içerisinde kaydırma etkisi yaratılabilmesi için kullanılır.

CListBox

Liste penceresidir. Kullanıcını görmesi ve listeden seçebilmesi için kullanılır.

CComboBox

Combo-box kontrol penceresidir. Combo box; list box fonksiyonlarına ek olarak üzerine metin girilebilmesini sağlar.

CHEdit

Pen Editing cihazlarını kullanarak metin girişi sağlar.

CBEdit

Pen Editing cihazlarını kullanarak metin girişi sağlar.

CControlBar

Standart araç çubuğu (toolbar) ve durum çubuğu (status bar) gibi ana pencere üzerinde altta veya yukarıda tanımlı kontrolleri içerir.

CStatusBar

Durum çubuğu (status-bar) kontrol penceresi için base sınıftır..

CToolBar

Standart araç çubuğu (toolbar) kontrol penceresi için base sınıftır..

CDialogBar

Bir kontrol çubuğunun modeless diyalog kutusudur.

CBitmapButton

Üzerinde metin yerine resim bulunan bir buton sınıfıdır.

CVBControl

Visual Basic Custom Control - VBX içeren bir penceredir.

CSplitterWnd

Kullanıcının çalışma ortamını birden fazla panoya bölebileceği bir penceredir.

3.4.1.13 Menu Classes

Kullanıcıya uygulama menüleri ile etkileşimini sağlayacak bir arayüz sağlar. Run-time sırasında menüler dinamik olarak eklenebilir, çıkarılabilir veya değiştirilebilir.

CMenu

Uygulama menüsü veya pop-up menüler için bir handle içerir.

3.4.1.14 Device-Context Classes

Aşağıdaki sınıfların birçoğu bir Windows device context(Windows Cihaz Bağlamı) ile ilgili bir handle içerir.Device context; ekran veya printer gibi bir çizim ortamı ile ilgili özellikleri içerir. Tüm çizim fonksiyonları bir device-context nesnesi üzerinden çağırılır.

CDC

Tüm Device contextleri için base sınıftır. Tüm ekrana erişim veya printer gibi ekran olmayan cihazlara erişimi sağlar.

CPaintDC

Pencerelerin OnPaint member fonksiyonlarında veya viewların OnDraw member fonksiyonlarında kullanılan bir ekran contextidir. Construction sırasında BeginPaint ve Destruction sırasında EndPaint fonksiyonlarını otomatik olarak çağırır.

CClientDC

Pencerelerin client bölgeleri için kullanılan bir ekran contextidir. Örnek oarak bir fare eventine yanıt vermek için kullanılır.

CWindowDC

Tüm client ve frame pencereleri içeren tüm pencereler için bir ekran contextidir.

CMetaFileDC

Windows metafile dosyaları için bir ekran contextidir

3.4.1.15 Drawing Object Classes

Aşağıdaki sınıflar Handle-bazlı GDI (Graphical Design Interface) nesneleri içerir. Bu sınıflar genel GDI çizim nesnelerini C++ sentaksında sağlarlar.

CGdiObject

GDI çizim araçları için base sınıftır.

CBitmap

GDI Bitmap BMP 'leri kullanabilmek için bir arayüz içerir.

CBrush

Ekran contexti içerisinde aktif olarak kullanılacak GDI fırçasını içerir.

CFont

Ekran contexti içerisinde aktif olarak kullanılacak GDI fontunu içerir.

CPalette

Uygulama ile renk çıkışlı cihaz (örn. Ekran) arasında arayüz olarak kullanılacak GDI renk paletini içerir.

CPen

Ekran contexti içerisinde aktif olarak kullanılacak GDI kalemi içerir.

CRgn

Pencere içerisinde eliptik veya polinomal olarak belirlenecek bir bölge oluşturabilecek bir GDI bölgesi içerir.CDC sınıfının kırpma member fonksiyonları ile birlikte kullanılır.

3.4.1.16 General-Purpose Classes

Bu kategorideki sınıflar geniş bir yelpazede (dosya I/O gibi)genel amaçlı hizmetleri sağlayacak nesneleri içerir.

3.4.1.17 File Classes

Özellikle geliştirici kendi özel giriş/çıkış işlemlerini gerçekleştirecekse kullanılır. Normalde application framework kullanılıyorsa bu tip işlemler application framework tarafından otomatik olarak sağlanır.

CFile

Binary disk dosyaları için programatik bir arayüz sağlar.

CMemFile

Bellek-içi dosyalar için programatik bir arayüz sağlar.

CStdioFile

Özellikle tekst modda buffered stream disk dosyaları için programatik bir arayüz sağlar.

CArchive

Nesnelerin serialization member fonksiyonları sırasında kalıcı saklama sağlanabilmesi için Cfile sınıfı ile birlikte kullanılır.

Diagnostics

CDumpContext ve CMemoryState sınıfları dabugging sırasında yardım amacıyla kullanılır.

CDumpContext

Diagnostic dump için bir hedef sağlar.

CMemoryState

Bellek kullanımının o andaki durumunu sağlar. Ayrıca daha önceki durumlarla karşılaştırır.

CRuntimeClass

Run-Time'da bir nesnenin asıl sınıfının belirlenmesini sağlar.

3.4.1.18 Exceptions

Exception-handling mekanizmasının sağlanmasını sağlar.

CException

Exceptionlar için base sınıf oluşturur.

CArchiveException

Arşivleme (veri saklama) exception'ı.

CFileException

Dosya bazlı exception.

CMemoryException

Bellek doldu exceptionı.

CNotSupportedException

Desteklenmeyen bir özellik sebebi ile oluşan exception.

CResourceException

Bir Windows resource'u yüklenmesi sırasında oluşan exception.

CUserException

Kullanıcı tarafından başlatılan bir operasyon sonucu oluşan exception.

3.4.1.19 Collections

Veri grupları oluşturabilinmesi amacıyla çeşitli veri grupları içerir. Bu sınıflar Windows uygulamaları dışında da kullanılabilir olmasına karşın esas olarak application framework içinde dökümanın veri yapısının tanımlanması amacıyla kullanılır.

CByteArray

BYTE tipli elemanları bir dizide saklar.

CDWordArray

Doubleword tipli elemanları bir dizide saklar.

CObArray

Cobject veya Cobject'ten türetilmiş nesnelerin işaretçilerini bir dizide saklar.

CPtrArray

Void (generic pointers) işaretçileri bir dizide saklar.

CStringArray

Cstring nesnelerini bir dizide saklar.

CWordArray

Word tipli elemanları bir dizide saklar.

CUIntArray

UINT tipli elemanları bir dizide saklar.

CObList

Cobject veya Cobject'ten türetilmiş nesnelerin işaretçilerini bir bağlantılı listede saklar.

CPtrList

Void (generic pointers) işaretçileri bir bağlantılı listede saklar.

CStringList

Cstring nesnelerini bir bağlantılı listede saklar.

CMapPtrToWord

Void pointerları WORD tipli verilere karşılık düşürür. Void pointerlar bu verilerin aranması için anahtar olarak kullanılır.

CMapPtrToPtr

Void pointerları Void pointerlara karşılık düşürür. Void pointerlar bu verilerin aranması için anahtar olarak kullanılır.

CMapStringToOb

CString nesnelerini CObject işaretçilere karşılık düşürür. CString nesneler bu verilerin aranması için anahtar olarak kullanılır.

CMapStringToPtr

CString nesnelerini Void pointerlara karşılık düşürür. CString nesneler bu verilerin aranması için anahtar olarak kullanılır.

CMapStringToString

CString nesnelerini CString nesnelerine karşılık düşürür. CString nesneler bu verilerin aranması için anahtar olarak kullanılır.

CMapWordToOb

WORD tipli verileri CObject işaretçilere karşılık düşürür. WORD tipli veriler bu verilerin aranması için anahtar olarak kullanılır.

CMapWordToPtr

WORD tipli verileri Void pointerlara karşılık düşürür. WORD tipli veriler bu verilerin aranması için anahtar olarak kullanılır.

3.4.1.20 Miscellaneous Support Classes

Aşağıdaki sınıflar çizim koordinatlarının. metinlerin ve tarih/saat bilgilerinin çizilebilmöesini sağlar.

CPoint

(x, y) koordinat çiftini tutar.

CSize

Uzaklık, relatif pozisyon veya çift değerleri tutar.

CRect

Dikdörtgen alanları tutar.

CString

Karakter katarlarını tutar.

CTime

Tarih ve saat bilgisini tutar.

CTimeSpan

Relatif tarih ve saat bilgisini tutar.

3.4.1.21 Macros and Globals

- Data types
- Run-time object model services
- Diagnostic services
- Exception processing
- CString formatting and message-box display
- Message maps
- Dialog data exchange and validation
- Application information and management
- Standard commands and window Ids

BÖLÜM 4. GÖRSEL ORTAMDA UYGULAMA GELİŞTİRME ÖRNEĞİ

Klasik programlama dillerinin tanıtımında bilgisayar ekranına bir mesaj çıkartma standart bir başlangıç yöntemi olarak görülmektedir. Bu bölümde görsel programlama ortamını tanıtmak amacıyla basit bir GUI programının oluşturma adımları tanıtılacaktır. "DENEME" programı bu amaçla basit bir çizim programı olarak verilecektir. Kullanıcı fare yardımı ile çalışma ortamında serbest çizim modunda çizim yapabilecek ve bu çizimleri bir dosyada saklayabilecektir.

- MFC kullanarak bir uygulama yaratılması sırasında genelde aşağıdaki adımlar izlenir:

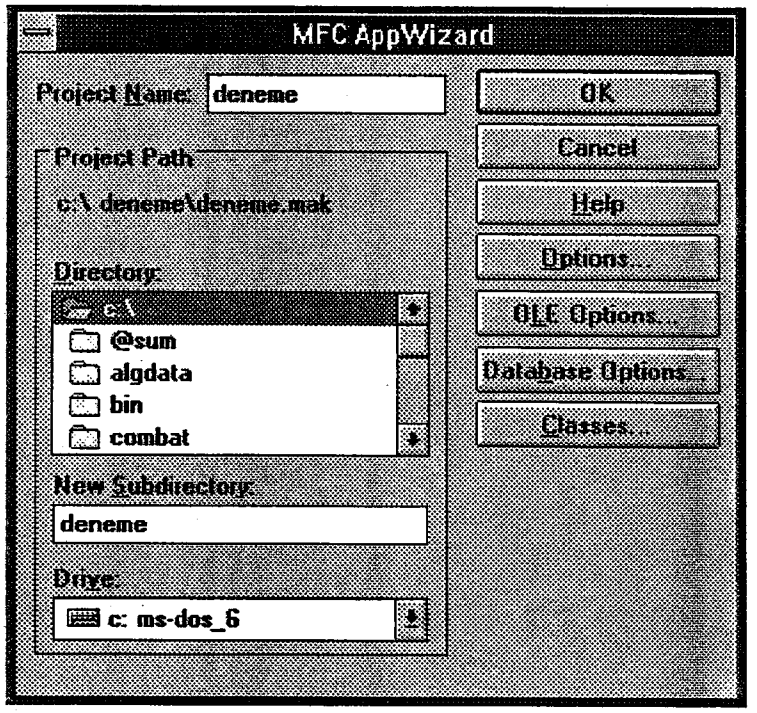
- AppWizard yardımcı aracı kullanılarak uygulama iskeleti yaratılır. Visual C++ resource editor yardımcı programı kullanılarak sistemin kullanıcı arayüzü hazırlanır.

- ClassWizard ve text editor kullanılarak uygulamaya özgü kodlar sisteme girilir.

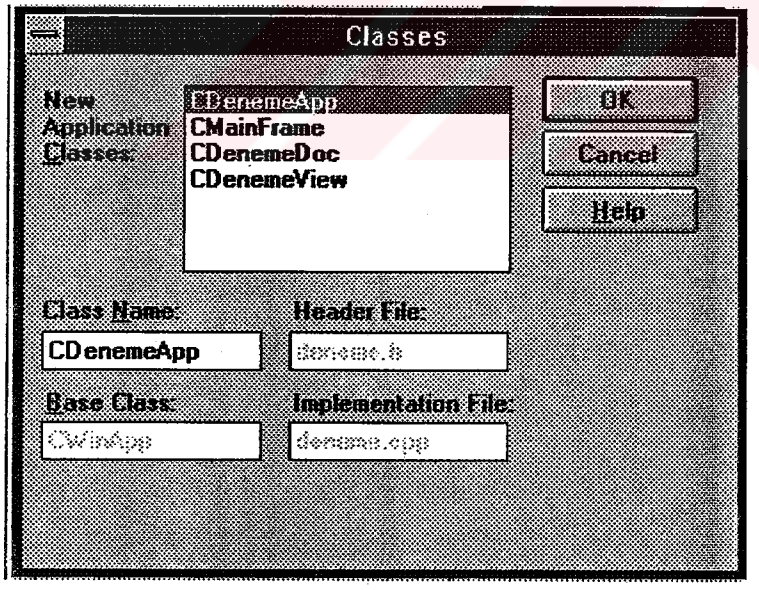
- Visual C++ kullanılarak uygulama test ve debug edilir. Ve gerekli düzeltme ve eklemeler yapılır.

AppWizard Kullanarak Program İskeletinin Yaratılması

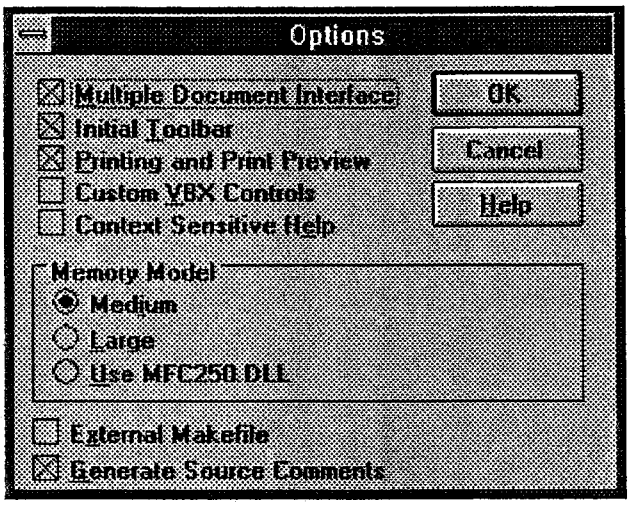
DENEME programının başlangıç kodlarının yaratılması için AppWizard kullanılır. Bu amaçla ile menüsünden New seçeneği seçilerek Yeni Proje ekranı açılır. Burada proje ismi olarak "deneme" girilir.



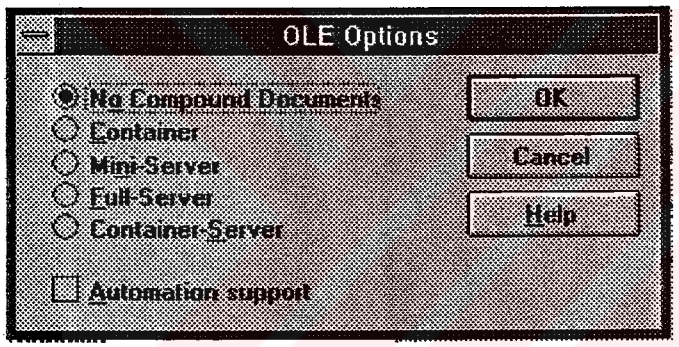
Şekil 4-1 Yeni Proje Diyalog Kutusu



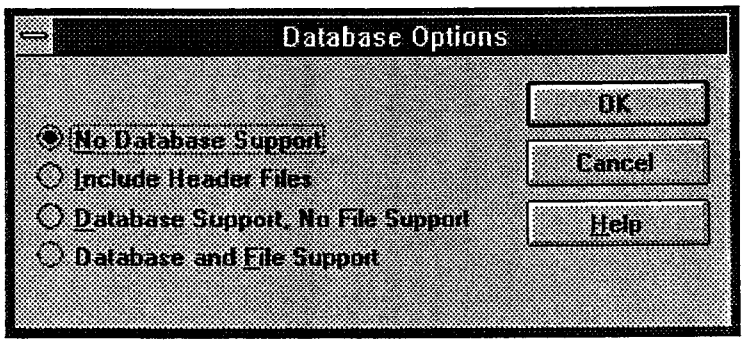
Şekil 4-2 AppWizard Tarafından Default Yaratılacak Classlar



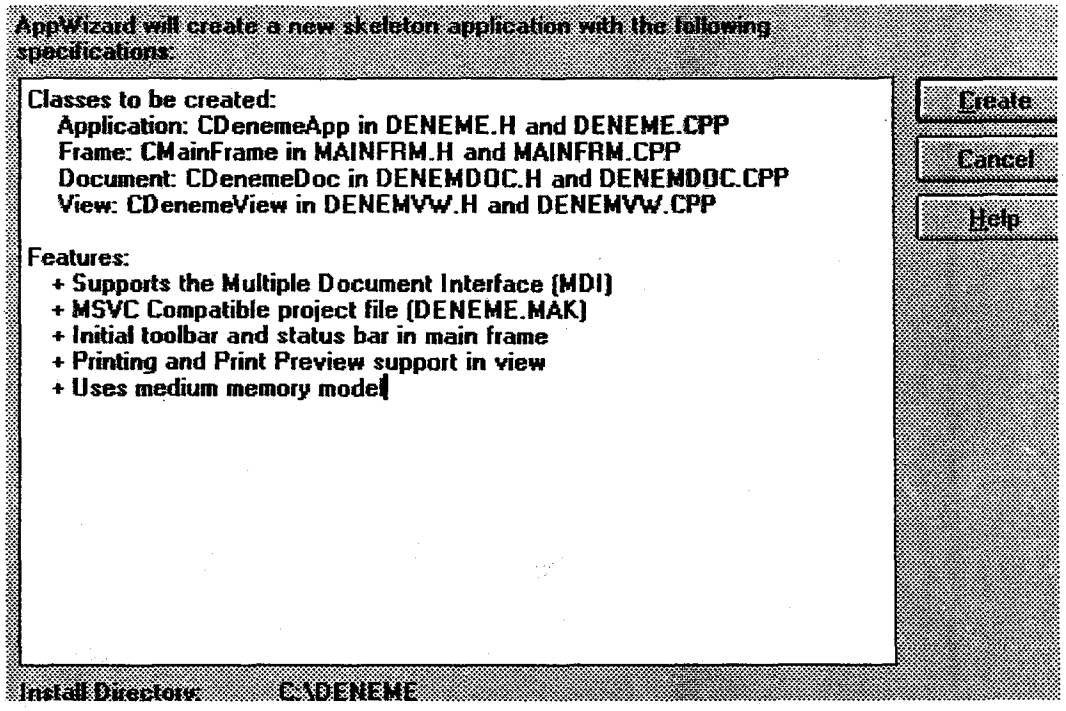
Şekil 4-3 Yaratılacak Uygulama İçin Seçenek Ekranı



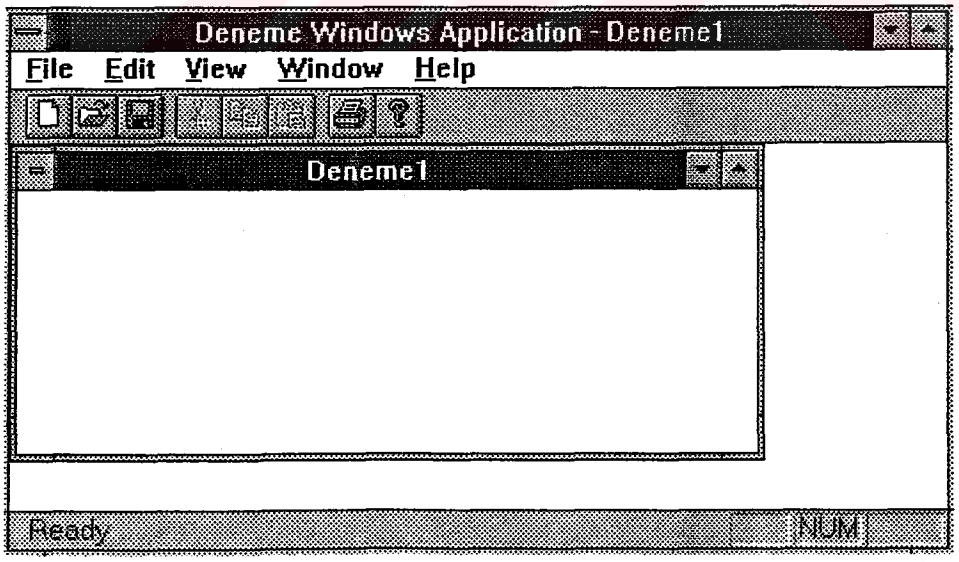
Şekil 4-4 OLE Seçenekleri



Şekil 4-5 Veri Tabanı Desteği



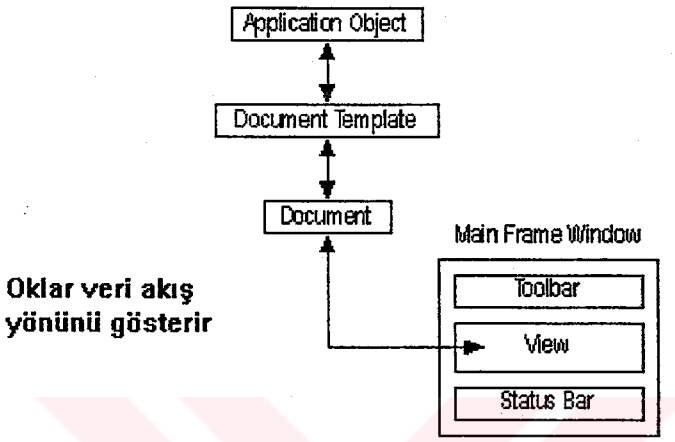
Şekil 4-6 Sonuç Raporu



Şekil 4-7 AppWizard Tarafından Yaratılan İskelet Uygulama

Görüldüğü gibi daha başlangıç aşamasında iken bile dosya saklama/okuma(File Open/Save), print ve print preview özelliklerini sağlayan ve standart bir araç ve durum çubuğuna sahip taban uygulama oluşmuştur.

4.1 Document Class Yaratımı



Şekil 4-8 Sınıflar Arası İlişkiler

Uygulamanın Run-Time çalışması sırasında uygulama içindeki biraraya gelmiş MFC nesneleri Windows mesajlarını birbirlerine göndererek ve bir diğerinin member fonksiyonlarını çağırarak haberleşirler. Dökümanlar

Tablo 4-1 Anahtar Nesneler

Object	Asıl Görevi	Diğer objectlerle ilişkisi
Application	Tüm diğer framework objectlerini kontrol eder.	Sistemde tanımlı tüm döküman tipleri için document templates in listesini tutar
Document template	Documentları yaratır ve	Belirli tipteki tüm açık

	kontrol eder.	dökümanları yönetir.
Document	Veriyi saklar.	Kendine bağlı veri üzerinde tanımlı çeşitli viewları kontrol eder
View	Dökümanlar ve kullanıcı arasındaki etkileşimi sağlar.	Dökümana bağlıdır. Frame windowun sahipliğindedir..
Frame window	Bir view'ı çerçeveler.	Bir dökümana bağlı view'ı sahiplenir..

Kullanıcı yeni bir document açtığında (varolan veya yeni) framework bir document object yaratır ve buna viewları bağlar. Eğer document bir dosyaya bağlı ise document veriyi dosyadan okur ve yazar. Kullanıcı herhangi bir view üzerindeki veriyi değiştirirse view document'ı uyarır. Document da kendine bağlı tüm viewlara bu değişikliği bildirir ve viewlar da kendi görüntülerinin tamamını veya güncellenmiş olan bölümünü bu yeni bilgiyle günceller.

MFC içinde documentlar **CDocument** class'ından yaratılır. Uygulama içinde geliştiricinin ve framework'ün sorumluluğu aşağıdaki tabloda verilmiştir:

Table 4.2 Döküman Uygulama Sorumluluğu

Programcının Görevi	Framework'ün Görevi
Cdocument classından yeni bir document türetmek.	Cdocument class'ına bağlı birçok document hizmetini vermek
Uygulamaya özgü data memberlarını eklemek.	
Uygulamaya yönelik initialization ve	Initialization ve cleanup kodlarını doğru

cleanup kodlarını yazmak.	zamanlarda çağırmak.
CDocument's class'ının Serialize member fonksiyonunu uygulamada istenen okuma yazma şekline göre değiştirmek	File Open/Save ve Save As komutlarının uygulanmesını sağlar.

Deneme uygulaması basit bir çizim programı olarak gerçeklenmiştir. Deneme içindeki dökümanlar herbir kullanıcı fare vuruşunda çizgileri saklar. Bir çizim en genel anlamda birçok vuruştan oluşacağı gözönüne alınırsa döküman kullanıcı tarafından yapılmış tüm vuruşların bir listesini saklar.

Şekil 4-9 Bir vuruş

AppWizard uygulamanın yaratılması için başlangıç kodunu yaratmasına rağmen amaca uygun çalışması için bazı ekler gerekmektedir. Deneme uygulaması içinde döküman class'ları CDenemeDoc adında yaratılmıştır. Bu nesnenin base class'ı Cdocument class'ıdır. Bu class tanımı içinde dökümanın içereceği vuruşları tutmak için m_VuruşListesi adında bir member değişken kullanılmıştır. Tüm bu tanımlar gözönüne alınarak CDenemeDoc class'ı için veri yapısı tanımlama kodu aşağıdaki şekilde verilmiştir:

```
// Veri yapısı tanımları

class CVuruş;

class CDenemeDoc : public CDocument
```

```
{
protected: // Create from serialization only
        CScribDoc( );
        DECLARE_DYNCREATE( CScribDoc )

// Attributes

        protected:
                CObList    m_VuruşListesi;    // Her bir eleman CVuruş Class'ından
                // Döküman tüm viewları için aynı kalem genişliği(Pen Width)
                // değerini tutmaktadır. Kalem genişliği değiştirildiği anda tüm
                // viewlar bu yeni değerle çizimlerini güncellerler.
                UINT    m_nPenWidth;              // Current user-selected width
                CPen    m_penCur;                 // Pen created according to
                                                  // user-selected pen style
(width)
public:
                CPen*   GetCurrentPen( ) { return &m_penCur; }
// Operations
```

```
public:
        void DeleteContents( );
        CVuruş* NewVuruş( );
        POSITION GetFirstVuruşPos( );
        CVuruş* GetNextVuruş( POSITION& pos );
// Implementation
public:
    virtual ~CScribDoc( );
    virtual void Serialize( CArchive& ar );    // Overridden for
                                               // document i/o
#ifdef _DEBUG
    virtual  void AssertValid( ) const;
    virtual  void Dump( CDumpContext& dc ) const;
#endif
protected:
        void InitDocument( );
    virtual BOOL OnNewDocument( );
        virtual BOOL OnOpenDocument( const char* pszPathName );
// Generated message-map functions
```

```
protected:
        //{{AFX_MSG(CScribDoc)
                // NOTE - the ClassWizard will add and remove member
                // functions here. DO NOT EDIT what you see in these
                // blocks of AppWizard code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP( )
};
```

Tablo 4-3 CDenemeDoc Data Members

Member	Description
m_VuruşListesi	Kullanıcı fare vuruşları listesi. Listedeki tüm elemanlar Cvuruş class'ındandır. Listenin kendisi MFC içindeki CObList (MFC tarafından sağlanan kolleksiyon class'larından biri)class'ındandır.
m_penCur	Çizimin yapılması için kullanılan Cpen nesnesi. Ana özelliği çizgi kalınlığıdır. Kalem, dökümanla birlikte yaratılır ve tüm vuruşların çizimi sırasında kullanılır.
m_nPenWidth	Kalemle çizilmiş tüm çizgilerin kalınlığı.

Not Programlama standartı olarak, class isimleri "C" ile, member değişkenlerin isimleri "m_" ile başlar.

Table 4.4 CDenemeDoc Member Functions

Member	Description
CScribDoc, ~CScribDoc	Döküman için Default constructor and a virtual destructor. Deneme uygulaması için bu başlangıç ve bitiş fonksiyonlarına kod eklemek gerekmemektedir.
DeleteContents	Döküman içinde çizimi oluşturan tüm vuruş listelerini siler.
GetCurrentPen	Herhangi bir anda çizim için gerekli kalemin işaretçisini alır.
InitDocument, OnNewDocument, OnOpenDocument	Yeni bir döküman yaratıldığında veya varolan açıldığında çağırılır. Cdocument classının varolan default fonksiyonlarını override eder.
NewVuruş	Yeni bir vuruş nesnesi yaratır ve m_VuruşListesi listesine ekler.
GetFirstVuruşPos	m_VuruşListesi içindeki ilk Vuruş nesnesine işaret eder..
GetNextVuruş	Liste içindeki bir sonraki vuruş nesnesine işaret eder.
Serialize	Overrides the Serialize member function of Cdocument classının Serialize member fonksiyonunu odverride eder. Vuruş listesinin diske nasıy yazılacağı ve diskten nasıl okunacağını belirler. AppWizard iskelet uygulama içinde bu kodu otomatik

	olarak oluşturur..
AssertValid	Nesnenin internal durumuna bakarak durumunu kontrol eder.
Dump	Debug sırasında nesnenin member fonksiyonlarını dökerek kontrol edilebilmesini sağlar.

```
// Declaration of class CScribDoc, then ...

    class CVuruş : public CObject

    {

    public:

        CVuruş( UINT nPenWidth );

    protected:

        CVuruş( );

        DECLARE_SERIAL( CVuruş )

// Attributes

        UINT   m_nPenWidth;  // Tüm vuruş için geçerli kalem genişliği

        CDWordArray   m_pointArray;  // Bağlı noktalar serisi


    // Operations
```

```
public:
        void AddPoint( CPoint pt );
        BOOL DrawVuruş( CDC* pDC );
// Helper functions
protected:
        CPoint GetPoint( int i ) const
                { return CPoint(m_pointArray[i]); }
public:
        virtual void Serialize( CArchive& ar );
};
```

Kod vuruş nesnesinin C++'taki class tanımını verir. Üyelik değişkenleri ve fonksiyonları vuruş nesnesinin veri yapısını tanımlar.

Table 4.5 CVuruş Data Members

Member	Description
m_nPenWidth	Vuruş çizilirken kullanılacak kalem genişliğini tutar.
m_pointArray	Vuruşu tanımlayan noktalar serisini saklar. Vuruşun yeniden çizilmesi gerektiği anda kullanılır.

Table 4.6 CVuruş Member Functions

Member	Description
CVuruş	Class biri protected biri de public olan iki constructor içerir.
AddPoint	Vuruş nesnesine yeni bir nokta ekler. Yeni nokta bir Cpoint nesnesi olarak verilir. MFC tarafından sağlanır.
DrawVuruş	View nesnesi döküman verisini yeniden çizme isterse her bir vuruş nesnesine yeniden kendini çizmesi için mesaj yollar(DrawVuruş member fonksiyonunu cağırır.)
GetPoint	Vuruşu belirleyen noktalar dizisinden verilen sıra numarasında bulunan oktayı döndürür.
Serialize	Döküman kendi verisini diske aktarırken Cvuruş classının Serialize fonksiyonunu çağırarak tek bir vuruşun ne şekilde saklanacağını ve hangi verilerin saklanacağını öğrenir.

Yeni Vuruşların Girilmesi ve Saklanması

// Last line of CScribDoc code, then ...

CStroke::CStroke()

```
{
// This empty constructor should be used by serialization only        }
CStroke::CStroke(UINT nPenWidth)
{
        m_nPenWidth = nPenWidth;      // Çizgi kalınlığı
}
```

İlk constructor; application framework tarafindan Cvuruş nesnelerinin initialization'ı anında kullanılır.Görüldüğü üzere parametre listesi ve kodu yoktur. İkinci constructor genel kullanım içindir. Yeni bir vuruş nesnesi yaratıldığı zaman bu public constructor kalem genişliğini alır. Cvuruş class'ına destructor fonksiyonu konmamıştır. Türediği class olan Cobject classının default destructor'unu kullanır.

```
// Constructor declarations, then ...
    void CStroke::AddPoint( CPoint pt )
    {
        m_pointArray.Add( MAKELONG( pt.x, pt.y ) );
    }
```

Vuruş nesnesi içindeki nokta listesi (m_pointArray) CDWordArray sınıfından bir member değişkendir. MFC tarafından sağlanan kolleksiyon sınıflarından biridir. CDWordArray içinde 32 bitlik doubleword tipli değişkenler tutulur. Cpoint sınıfı ise yatay ve dikey koordinatlarını 16 bit içinde sakladığından bir doubleword bir Cpoint nesnesini saklamak için kullanılmıştır.

Document Yönetimi

Tipik olarak;

(a) Dökümanın data memberlarını initialize etmek için,

(b) Sona erdirme işleminde (cleanup) veri için ayrıymış olan bellek alanının ve sistem kaynaklarının serbest bırakılması için,

kod yazılması durumunda kalınır. Yeni bir döküman açıldığında çizim için kullanılacak kalem nesnesinin sisteme tanıtılması ve döküman kapanırken de dökümanın vuruşları saklama için kullandığı bellek alanını bırakılması gerekir.

Initializing and Cleaning Up

Döküman menüden New seçeneği veya Open seçeneği ile açılabileceğinden CDenemeDoc dökümanı hem OnNewDocument hem de OnOpenDocument member fonksiyonlarını override ederek uygulama için gerekli kodları ekler. Fakat bu uygulamada hem yeni döküman açma hem de varolanı açma aynı fonksiyonları göstereceğinden iki fonksiyonda da InitDocument fonksiyonu çağırılır,

Framework yeni bir döküman açıldığında OnNewDocument fonksiyonunu ve varolan bir döküman açıldığında OnOpenDocument fonksiyonlarını otomatik olarak çağırır. İskelet uygulamada OnNewDocument fonksiyonunun iskelet versiyonunun bulunmasına karşın OnOpenDocument fonksiyonu programcı tarafından eklenmelidir.

CDenemeDoc Dökümanı initialization kodları

```
// Empty constructor and destructors, then ...

        void CScribDoc::InitDocument( )

        {

            m_nPenWidth = 2;  // Default 2-pixel pen width

            // Solid black pen
```

```
                m_penCur.CreatePen( PS_SOLID, m_nPenWidth,
RGB(0,0,0) );
        }
// InitDocument, then ...
BOOL CScribDoc::OnNewDocument( )
{
    if( !CDocument::OnNewDocument( ) )
        return FALSE;
        InitDocument( );
    return TRUE;
}
// OnNewDocument, then ...
    BOOL CScribDoc::OnOpenDocument( const char* pszPathName )
    {
        if( !CDocument::OnOpenDocument( pszPathName ) )
            return FALSE;
        InitDocument( );
        return TRUE;
    }
```

Her iki fonksiyon da öncelikle base sınıflarının member fonksiyonlarını çağırarak öncelikle default olarak yapılması gereken işlemleri yürütürler.

```
// Empty constructor and destructors, then ...

    void CScribDoc::DeleteContents( )

    {

        while( !m_strokeList.IsEmpty( ) )

        {

            delete m_strokeList.RemoveHead( );

        }

    }
```

1.DeleteContents fonksiyonu döküman verisinin serbest bırakılması için çok uygun bir yerdir.Framework bu fonksiyonu uygulamanın herhangi bir anında döküman verisinin boşaltılması için bu fonksiyonu otomatik olarak çağırır.

BÖLÜM 5. KALIP TASARIM SİSTEMİ

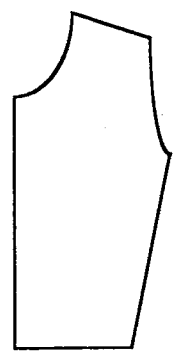
5.1 Kavramlar

5.1.1 Model

Bir firmanın ürettiği giysi tipidir. Firma aynı anda birden fazla modelle çalışabilir. Mevcut bir modelden birkaç değişiklikle yeni modeller türetebilir. Tekstil sektöründe özellikle fason üretim yapan firmalara gelen siparişler genelde bir model bilgisi ile birlikte gelir. Dolayısıyla zaman içinde çalışılan firmaların kullandığı modeller bir katalog oluşturur. Firma modelleri diğer bir firma modelinden ayrıktır.

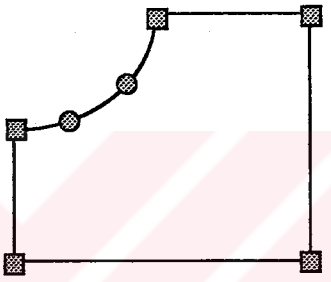
5.1.2 Kalıp

Bir giysiyi oluşturan birim parçadır. Bir model birçok kalıbın birleşmesi sonucu oluşur. Kalıplar giysinin karmaşılığına bağlı olarak bir modelde 5-50 arasında olabilir. Şekil 4-1'de bir kalıp örneği görülmektedir.



Şekil 5-1 Bir kalıp örneği

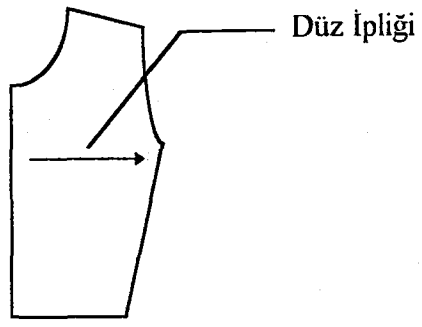
Bir kalıp düz ve eğri çizgilerden oluşur. Düz ve eğrilerin kullanıcı tarafından verilebilmesi için sistemde düz ve eğri noktalar tanımlanmıştır. Düz karakterli bir noktanın önündeki ve arkasındaki çizgiler düz olarak gelirken eğri tipli nokta kendinden önceki ve sonraki noktaları yumuşak bir kavisle bağlayacak bir eğri çizer. Uygulama ekranında kare noktalar düz tipli noktaları daire şeklindeki noktalar da eğri noktaları gösterir. Eğri çizimi sırasında kullanılan ara noktalar hep eğri çizgisi üzerinde kalırlar. Eğriyi oluşturan noktalardan herhangi biri yerdeğiştirirse tüm eğri yeniden hesaplanır ve çizilir. Şekil 5-2'de düz ve eğri noktaları bulunan bir kalıp görülmektedir.



Şekil 5-2 Düz ve eğri noktalar

5.1.3 Düz İpliği

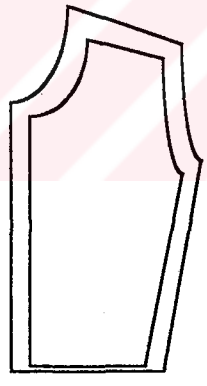
Kalıp çizimi sırasında kalıbın doğrultusu ve yönünün belirtilebilmesi için kalıp üzerine düz ipliği denilen bir işaret konur. Kalıbın kumaş üzerine yerleştirilmesi sırasında bu düz ipliğinden yararlanılarak kalıp doğrultusu ile kumaş doğrultusu çakıştırılır. Tarayıcı ile okuma sonucu sisteme alınan kalıp ekran üzerine düz ipliği doğrultusuna gelecek şekilde döndürülür. Şekil 5-3'de bir kalıp örneği üzerinde düz ipliği görülmektedir.



Şekil 5-3 Düz İpliği

5.1.4 Dikiş Payı

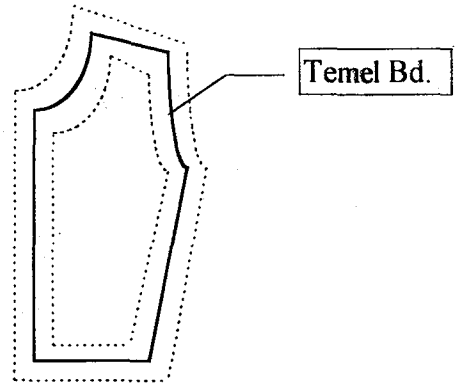
Kalıp hazırlandıktan sonra kumaş üzerinde kesilir ve dikilir. Dikim sırasında kalıpları birbirine tutturabilmek için kalıp profiline paylar bırakılır. Bu paylara dikiş payları denir. Şekil 5-5'de bir kalıp örneği üzerinde dikiş payları görülmektedir.



Şekil 5-4 Dikiş Payı

5.1.5 Serilendirme

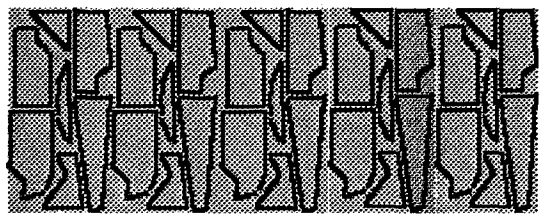
Bir model birden fazla beden içerebilir. Bu taktirde kalıp tasarımı belirli bir bedendeki kalıp üzerinde yapılır. Daha sonra kalıbın noktalarının bedenler arasındaki değişim şekilleri sisteme tanıtılarak modelin diğer bedenleri otomatik olarak uygulama tarafından oluşturulur. Şekil 5-5'de serilendirilmiş bir kalıp örneği görülmektedir.



Şekil 5-5 Temel Beden ve Seriler

5.1.6 Pastal

Modellerin çizimi hazırlandıktan sonra kumaş üzerinde kesilme safhası gelir. Bu amaçla öncelikle o üretim sırasında hazırlanması gereken kalıplar kumaş üzerine minimum kayıpla yerleştirilirler. Şekil 5-6'de bir pastal örneği görülmektedir.



Şekil 5-6 Pastal

5.2 Veri Tabanı

Tanımlanan sistemin veri yapısı aşağıda verilmiştir.

5.2.1 Firma Bilgileri (tbl Firma)

Üretici şirketin çalıştığı şirketlerin bilgisini tutar. Tablo 5-1'de tbl Firma Veri Yapısı görülmektedir.

Tablo 5-1 tbl Firma Veri Yapısı

Alan Adı	Tip	Uz.	Açıklama
Firma Kodu	Long	5	Firma kodu
F_Adı	Text	30	Firma adı
F_Özlük Bilgileri	Text	50	Ek firma bilgisi

5.2.2 Model (tbl Model)

Firma bazında modellerin katalog bilgilerini tutar. Tablo 5-2'de tbl Model Veri Yapısı görülmektedir.

Tablo 5-2 tbl Model Veri Yapısı

Alan Adı	Tip	Uz.	Açıklama
Model Kodu	Long	5	Model kodu
Firma Kodu	Long	5	Modelin firma kodu
M_Adı	Text	30	Model adı
M_Tanımı	Text	50	Model tanımı

Temel Beden	Text	10	Çizimde kullanılacak beden
M_O_Tarihi	Date	8	Model oluşturma tarihi
M_D_Tarihi	Date	8	Model değiştirme tarihi
Zoom Faktörü	Single	5	Model üzerinde en son kullanılan zoom faktörü

5.2.3 Beden (tbl Beden)

Modelde kullanılan bedenleri tutar. Tablo 5-3'de tbl Beden Veri Yapısı görülmektedir.

Tablo 5-3 tbl Beden Veri Yapısı

Alan Adı	Tip	Uz.	Açıklama
Model Kodu	Long		Bedenin model kodu
Beden Adı	Text		Beden adı

5.2.4 Kalıp Bedenleri (tbl Kalıp Bedenleri)

Modeldeki kalıpların hangi bedenlerde kullanılacağı bilgisini tutar. Tablo 5-5'de tbl Kalıp Bedenleri Veri Yapısı görülmektedir.

Tablo 5-4 tbl Kalıp Bedenleri Veri Yapısı

Alan Adı	Tip	Uz.	Açıklama

Model Kodu	Long	5	Model kodu
Kalıp Kodu	Long	5	Kalıp kodu
Beden Adı	Text	10	İlgili beden adı

5.2.5 Kalıp (tbl Kalıp)

Kalıp katalog bilgisini tutar. Tablo 5-5'de tbl Kalıp Veri Yapısı görülmektedir.

Tablo 5-5 tbl Kalıp Veri Yapısı

Alan Adı	Tip	Uz	Açıklama
Kalıp Kodu	Long	5	Kalıp kodu
Model Kodu	Long	5	Kalıbın model kodu
K_Adı	Text	10	Kalıp adı
K_Tanımı	Text	30	Ek kalıp tanımı
Kaç Tane	Integer	2	Modelde kalıp kaç tane kullanılacak
Max Dönme	Single	5	Pastal üzerinde ne kadar dönebilir?
Dikey Dönüş	Yes/No	1	Kalıp pastal üzerinde dikey dönebilir mi?
Yatay Dönüş	Yes/No	1	Kalıp pastal üzerinde yatay dönebilir mi?
Origin X	Single	5	Kalıp orijin noktası X koordinatı
Origin Y	Single	5	Kalıp orijin noktası Y koordinatı

Metin X	Single	5	Metin bilgisinin konacağı noktanın X koordinatı
Metin Y	Single	5	Metin bilgisinin konacağı noktanın Y koordinatı
Düz İplik X1	Single	5	Düz ipliğin başlangıç noktasının X koordinatı
Düz İplik Y1	Single	5	Düz ipliğin başlangıç noktasının Y koordinatı
Düz İplik X2	Single	5	Düz ipliğin bitiş noktasının X koordinatı
Düz İplik Y2	Single	5	Düz ipliğin bitiş noktasının Y koordinatı
Dikiş Payı Tipi	Text	5	Lst Dikiş Payı 'ndan gelen kalıp dikiş payı tipi
Dikiş Payı Baş Kalınlık	Single	5	mm olarak kalıp dikiş payı başlangıç kalınlığı
Dikiş Payı Son Kalınlık	Single	5	mm olarak kalıp dikiş payı bitiş kalınlığı
Çakışma Noktası X	Single	5	Çakışma noktası X koordinatı
Çakışma Noktası Y	Single	5	Çakışma noktası Y koordinatı
Çakışma Noktası Tipi	Byte	1	Çakışma noktası tipi (Yatay/dikey/herikisi)
K_O_Tarihi	Date	8	Kalıp oluşturma tarihi
K_D_Tarihi	Date	8	Kalıp düzeltme tarihi

5.2.6 Dikiş Payı Tipleri (lst Dikiş Payı Tipi)

Sistemde tanımlı dikiş payı tiplerini tutar. Tablo 5-6'da Lst Dikiş Payı Veri Yapısı görülmektedir.

Tablo 5-6 Lst Dikiş Payı Tipi Veri Yapısı

Alan Adı	Tip	Uz	Açıklama
Dikiş Payı Tipi	Text	5	Dikiş payı kodu
Tip Tanımı	Text	30	Dikiş payının detaylı açıklaması

5.2.7 Noktalar (tbl Nokta)

Kalıbı oluşturan noktaların koordinat ve tip bilgilerini tutar. Kalıbın ekranda oluşturulması bu tablodan okunan noktalarla yapılır. Tablo 5-7'de tbl Nokta Veri Yapısı görülmektedir.

Tablo 5-7 tbl Nokta Veri Yapısı

Alan Adı	Tip	Uz	Açıklama
Kalıp Kodu	Long	5	Noktanın bulunduğu kalıp kodu
Sıra No	Integer	2	Kalıp üzerinde noktanın kaçıncı sırada olduğu
X	Single	5	Noktanın X koordinatı
Y	Single	5	Noktanın Y koordinatı
Köşe/Kavis	Yes/No	1	Nokta köşe nokta mı? Kavis nokta mı?

Nokta Özelliği	Byte	1	Nokta ek özelliği (örneğin Pile mi?)
Kural Kodu	Long	5	Lst Serilendirme kurallarından gelen seri kuralı

5.2.8 Çıtlar (tbl Çıt)

Kalıp üzerinde tanımlanmış çıtların koordinat ve tip bilgilerini tutar. Tablo 5-8'de tbl Çıt Veri Yapısı görülmektedir.

Tablo 5-8 tbl Çıt Veri Yapısı

Alan Adı	Tip	Uz	Açıklama
Kalıp Kodu	Long	5	Kalıp kodu
Çıt Tipi	Text	5	Lst Çıt tablosundan gelen çıt adı
Çıt X1	Single	5	Çıtın bulunduğu noktanın X koordinatı
Çıt Y1	Single	5	Çıtın bulunduğu noktanın Y koordinatı
Ç_Extra	Single	5	Çıt tipine bağlı olarak gerekli olabilecek ek bilgi (Örn. Boy)

5.2.9 Pensler (tbl Pens)

Kalıp üzerinde tanımlanmış penslerin koordinat ve tip bilgilerini tutar. Tablo 5-9'da tbl Pens Veri Yapısı görülmektedir.

Tablo 5-9 tbl Pens Veri Yapısı

Alan Adı	Tip	Uz.	Açıklama
Kalıp Kodu	Long	5	Kalıp kodu
Pens Tipi	Yes/No	1	İç veya dış pens
Pens X1	Single	5	Pens 1. noktası X koordinatı
Pens Y1	Single	5	Pens 1. noktası Y koordinatı
Pens X2	Single	5	Pens 2. noktası X koordinatı
Pens Y2	Single	5	Pens 2. noktası Y koordinatı
Genişlik	Single	5	Pens genişliği
Uzunluk	Single	5	Pens uzunluğu

5.2.10 Çıt Tipleri (lst Çıt)

Sistemde tanımlı çıt tiplerinin kod ve tanımlarını tutar. Tablo 5-10'da Lst Çıt Veri Yapısı görülmektedir.

Tablo 5-10 Lst Çıt Veri Yapısı

Alan Adı	Tip	Uz.	Açıklama
Çıt Tipi	Text	5	Çıt isimleri

Ç_Tanımı	Text	30	Çıt ile ilgili detay açıklama

5.2.11 Serilendirme Kuralları (lst Serilendirme Kuralı)

Serilendirme sırasında noktalara uygulanacak kural listesini tutar. Tablo 5-11'de Lst Serilendirme Kuralları Veri Yapısı görülmektedir.

Tablo 5-11 lst Serilendirme Kuralı Veri Yapısı

Alan Adı	Tip	Uz.	Açıklama
Kural Kodu	Long	5	Kural kodu
SK_Adı	Text	10	Kural tanımı
Beden Adı	Text	50	Kuralın geçerli olduğu beden kodu
X Değişimi	Double	8	Kuralın X yönündeki değişimi
Y Değişimi	Double	8	Kuralın Y yönündeki değişimi

5.2.12 Kalıp Malzemeleri (tbl Kalıp Malzemeleri)

Kalıpta kullanılabilecek malzemeleri tutar. Tablo 5-12'de tbl Kalıp Malzemeleri Veri Yapısı görülmektedir.

Tablo 5-12 tbl Kalıp Malzemeleri Veri Yapısı

Alan Adı	Tip	Uz.	Açıklama
Kalıp Kodu	Long	5	Kalıp kodu

Malzeme Kodu	Long	5	Malzeme kodu

5.2.13 Malzemeler (lst Malzeme)

Tüm malzeme listesini tutar. Tablo 5-14'de tbl Malzeme Veri Yapısı görülmektedir.

Tablo 5-13 lst Malzeme Veri Yapısı

Alan Adı			
Malzeme Kodu	Long	5	Malzeme kodu
Malzeme Adı	Text	30	Malzeme açıklaması

5.2.14 Pastal (tbl Pastal)

Hazırlanmış pastalların karalog bilgisini tutar. Tablo 5-15'de tbl Pastal Veri Yapısı görülmektedir.

Tablo 5-14 tbl Pastal Veri Yapısı

Alan Adı			
Pastal Kodu	Long	5	Pastal kodu
Malzeme Kodu	Long	5	Lst Malzemeden gelen pastalın atıldığı malzeme kodu
P_Adı	Text	30	Pastal adı
P_Tanımı	Text	50	Pastal tanımı

Kumaş Eni	Single	5	Pastal kumaş eni

5.2.15 Pastal Bedenleri (tbl Pastal Bedenleri)

Pastalda hangi model ve kalıbın hangi bedenleri için yerleştirme yapıldığını tutar. Tablo 5-15'de tbl Pastal Bedenleri Veri Yapısı görülmektedir.

Tablo 5-15 tbl Pastal Bedenleri Veri Yapısı

Alan Adı			
Pastal Kodu	Long	5	Pastal kodu
Model Kodu	Long	5	Pastalda kullanılan model kodu
Beden Adı	Text	10	Model beden adı
Adet	Integer	2	Modelde bu bedenden kaç adet kullanılacak?

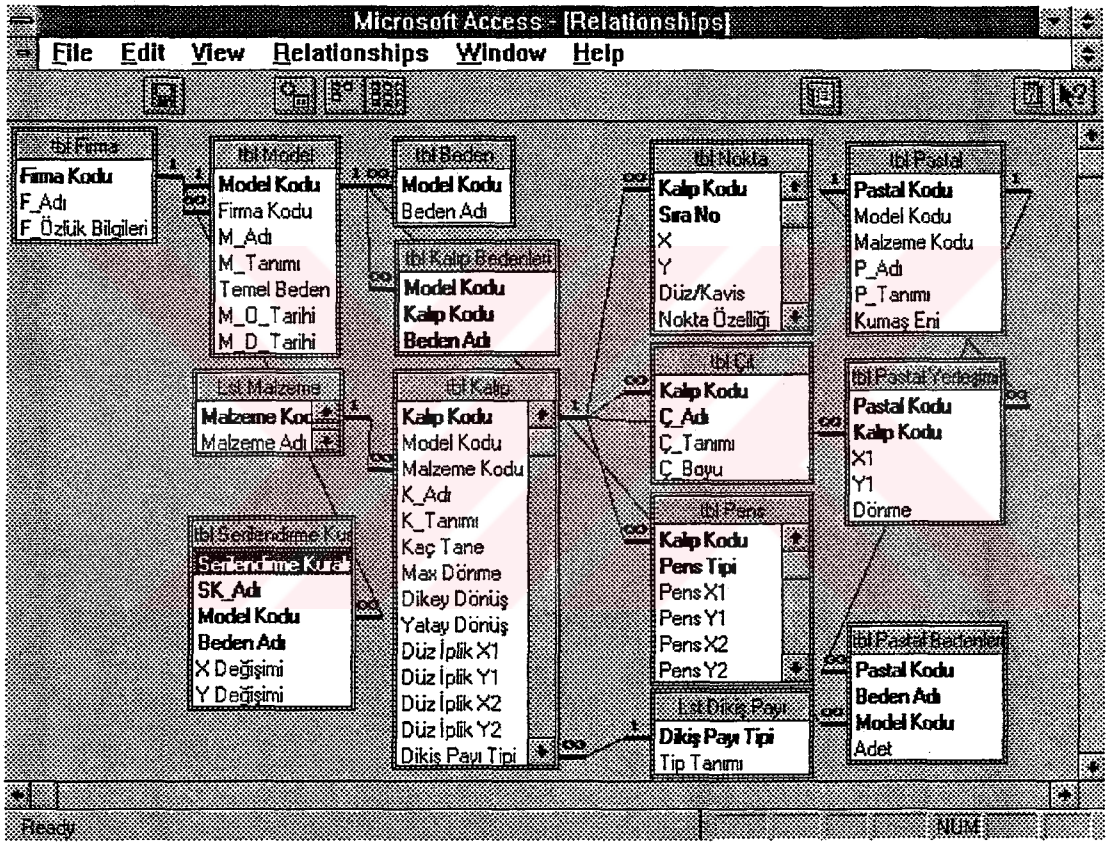
5.2.16 Pastal Yerleşimi (tbl Pastal Yerleşimi)

Pastal içinde bir kalıbı hangi koordinata ve düz ipliğine göre ne kadarlık bir dönme ile yerleştirildiğini tutar. Tablo 5-16'da tbl Pastal Yerleşimi Veri Yapısı görülmektedir.

Tablo 5-16 tbl Pastal Yerleşimi Veri Yapısı

Alan Adı			
Kalıp Kodu	Long	5	Kalıp kodu

Pastal Kodu	Long	5	Pastal kodu
X1	Single	5	Pastala yerleşecek kalıbın orijininin X koordinatı
Y1	Single	5	Pastala yerleşecek kalıbın orijininin Y koordinatı
Dönme	Single	5	Modelde bu bedenden kaç adet kullanılacak?



Şekil 5-7 Kalıp tasarım sistemi tablolar arası ilişkiler

5.3 Serilendirme (Grading)

5.3.1 Serilendirme Kuralları

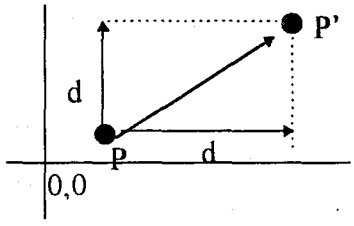
Tablo 5-17 Serilendirme Kuralları

Kısaltma	Tip
I	Artımlı Increment
IR	Referans Noktasına Bağlı Artımlı Increment Relative to the Reference Point
F	Aileden Sonra Yüzde Uygulamalı Percentages Following Family
FI	Aileden Sonra Yüzde ve Artırım Uygulamalı - Percentages Following Family with Increments
FIR	Referans Noktasına Bağlı Aileden Sonra Yüzde Uygulamalı Percentages Following Family with Increments Relative to the Reference Point
D	Bağımlı Dependent
DI	Bağımlı ve Artımlı Dependent with Increment
DIR	Referans Noktasına Bağlı Bağımlı ve Artımlı Dependent with Increment Relative to the Reference Point
DL	Dependent with Displacement through a Line of Dependence

DP	Dependent with Displacement through a Line of Union
PROP	Orantılı Proportional
CP	Paralel Eğri Parallel Curve
PF	Profil Profile
%	Yüzde Percentages
%I	Yüzdeli Artırımlı Percentages with Increments
%IR	Referans Noktasına Bağlı Yüzdeli Artırımlı Percentages with Increments Relative to the Reference Point
IRL	Referans Çizgisine Bağlı Artımlı Increments Relative to the Line of Reference
NIL	Yok NIL

5.3.2 I Increments (Artımlı)

I serilendirme kuralı her seri için X ve Y artımı olarak tanımlanır. Şekil 5-8'de I Serilendirme Kuralı görülmektedir.



Şekil 5-8 I Serilendirme Kuralı

Bu değerler orijin noktasına bağlı olarak hesaplanır.

Fonksiyon:

PerformGrade (Tip="I", Nokta, X , Y);

Parametreler:

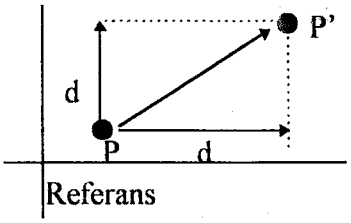
Tip: "I" For Increment

Nokta : Serilendirilecek Nokta

X,Y: Artım miktarı

5.3.3 IR Increment Relative to the Reference Point (Referans Noktasına Bağıl Artımlı)

IR serilendirme kuralı her seri için referans noktasına bağlı olarak X ve Y artımı olarak tanımlanır. Şekil 5-9'da IR Serilendirme Kuralı görülmektedir.



Şekil 5-9 IR Serilendirme Kuralı

Nokta aşağıdaki kurallara uygun şekilde hareket eder:

Artım pozitif ise referans noktası için girilen uzaklık arttırılır.

Artım negatif ise referans noktası için girilen uzaklık azaltılır.

Fonksiyon:

PerformGrade (Tip="IR", Nokta, X , Y);

Parametreler:

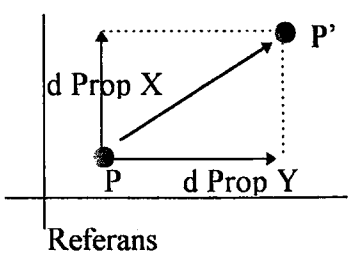
Tip: "IR"

Nokta : Serilendirilecek Nokta

X,Y: Artım miktarı

5.3.4 F Percentages Following Family (Aileden Sonra Yüzdeli)

Şekil 5-10'da F Serilendirme Kuralı görülmektedir.



Şekil 5-10 F Serilendirme Kuralı

Fonksiyon:

PerformGrade (Tip="F", Nokta, dPropX , dPropY);

Parametreler:

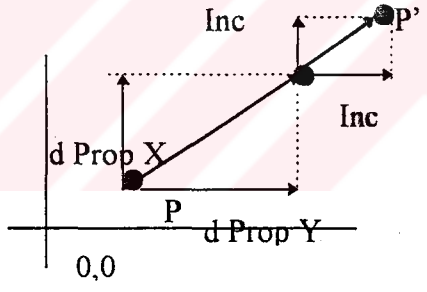
Tip: "F"

Nokta : Serilendirilecek Nokta

dPropX,dPropY:Oranlar

5.3.5 FI Percentages Following Family with Increments (Aileden Sonra Yüzdeli, Artımlı)

Serilendirme önce uzaklıkların boyla orantılı olarak hesaplanıp daha sonra X ve Y yönünde arttırımla yapılır. Şekil 5-11'de FI Serilendirme Kuralı görülmektedir.



Şekil 5-11 FI Serilendirme Kuralı

Fonksiyon:

PerformGrade (Tip="FI", Nokta, dPropX, dPropY, X , Y);

Parametreler:

Tip: "FI"

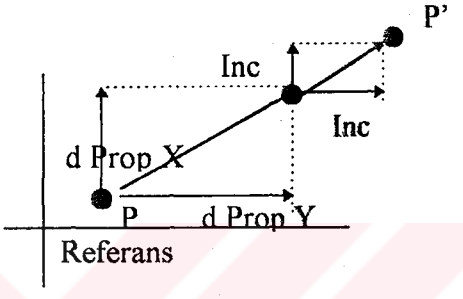
Nokta : Serilendirilecek Nokta

dPropX,dPropY:Oranlar

X,Y: Artım miktarı

5.3.6 FIR Percentages Following Family with Increments Relative to the Reference Point (Referans Noktasına Bağlı Aileden Sonra Yüzdeli, Artımlı)

Serilendirme önce uzaklıkların boyla orantılı olarak hesaplanıp daha sonra X ve Y yönünde arttırımla yapılır. Üstteki kuraldan farkı başlangıç noktası olarak referans noktasının alınmasıdır. Şekil 5-12'de FIR Serilendirme Kuralı görülmektedir.



Şekil 5.12 FIR Serilendirme Kuralı

Fonksiyon:

PerformGrade (Tip="FIR", Nokta, dPropX, dPropY, X , Y);

Parametreler:

Tip: "FIR"

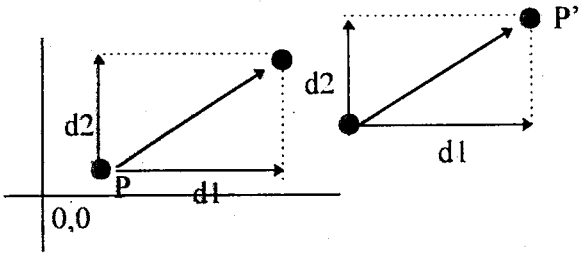
Nokta : Serilendirilecek Nokta

dPropX,dPropY:Oranlar

X,Y: Artım miktarı

5.3.7 D Dependent (Bağımlı)

Serilendirme noktası bağlı noktaya göre hesaplanır. Şekil 5-14'de D Serilendirme Kuralı görülmektedir.



Şekil 5-13 D Serilendirme Kuralı

Fonksiyon:

PerformGrade (Tip="D", Nokta, X , Y);

Parametreler:

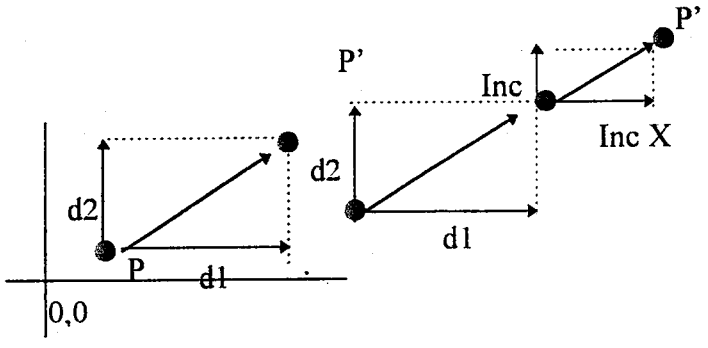
Tip: "D"

Nokta : Serilendirilecek Nokta

X,Y: Bağlı nokta artım miktarı

5.3.8 DI Dependent with Increments (Artımlı, Bağımlı)

Serilendirme noktası bağlı nokta kadar hareket ettirildikten sonra, X ve Y yönünde artım eklenerek bulunur. Şekil 5-15'de DI Serilendirme Kuralı görülmektedir.



Şekil 5-14 DI Serilendirme Kuralı

Fonksiyon:

PerformGrade (Tip="DI", Nokta,incX,incY, X , Y);

Parametreler:

Tip: "DI"

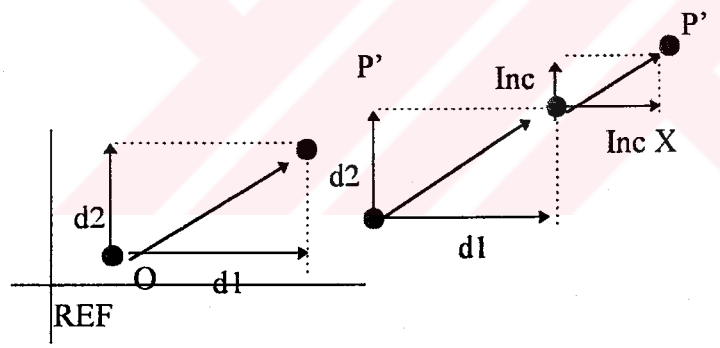
Nokta : Serilendirilecek Nokta

incX,incY:Artım miktarı

X,Y: Bağlı nokta artım miktarı

5.3.9 DIR Dependent with Increment Relative to the Reference Point (Referans noktasına Bağlı, Bağımlı, Artımlı)

Yukarıdaki kurala benzer olarak çalışır. Ancak orijin olarak referans noktası alınır. Şekil 5-15'de DIR Serilendirme Kuralı görülmektedir.



Şekil 5-15 DIR Serilendirme Kuralı

Fonksiyon:

PerformGrade (Tip="DIR", Nokta,incX,incY, X , Y);

Parametreler:

Tip: "DIR"

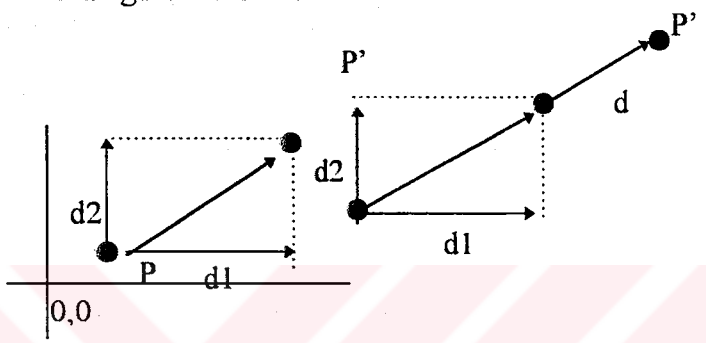
Nokta : Serilendirilecek Nokta

incX,incY:Artım miktarı

X,Y: Bağlı nokta artım miktarı

5.3.10 DL Dependent with Displacement through a Line of Dependence

Nokta bağlı nokta kadar ötelenir. Daha sonra bağlılık hattı uygulanır. Şekil 5-16'da DL Serilendirme Kuralı görülmektedir.



Şekil 5-16 DL Serilendirme Kuralı

Fonksiyon:

PerformGrade (Tip="DL", Nokta, X , Y, distance);

Parametreler:

Tip: "DL"

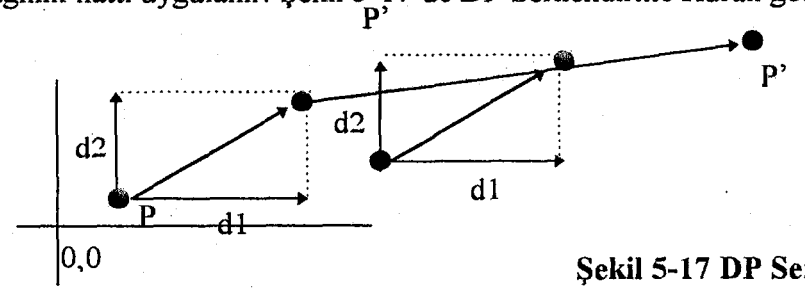
Nokta : Serilendirilecek nokta

X,Y: Bağlı nokta artım miktarı

distance: Ensonda uygulanacak doğru uzunluğu

5.3.11 DP Dependent with Displacement through a Line of Union (Birleşim Çizgisi Boyunca Yerdeğiştirmeli ve Bağımlı)

Nokta koordinatı bağlı nokta kadar arttırılır. Daha sonra iki noktadan geçen bağlılık hattı uygulanır. Şekil 5-17'de DP Serilendirme Kuralı görülmektedir.



Şekil 5-17 DP Serilendirme Kuralı

Fonksiyon:

PerformGrade (Tip="DP", Nokta, X , Y, distance);

Parametreler:

Tip: "DP"

Nokta : Serilendirilecek nokta

X,Y: Bağlı nokta artım miktarı

distance: Ensonda uygulanacak doğru uzunluğu

5.3.12 PROP Proportional (Orantılı)

Tüm bedenlerde eşit yüzdelerle değiştirilir.

Fonksiyon:

PerformGrade (Tip="PROP", Nokta, Yüzde);

Parametreler:

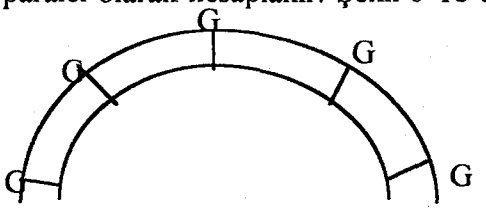
Tip: "PROP"

Nokta : Serilendirilecek nokta

Yüzde: Tüm bedenlerde uygulanacak yüzde

5.3.13 CP Parallel Curve (Paralel Eğri)

Noktalar kalıbın profiline paralel olarak hesaplanır. Şekil 5-18'de CP Serilendirme Kuralı görülmektedir.



Şekil 5-18 CP Serilendirme Kuralı

Fonksiyon:

PerformGrade (Tip="CP", Nokta, Kalınlık);

Parametreler:

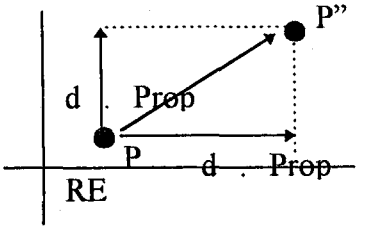
Tip: "CP"

Nokta : Serilendirilecek nokta

Kalınlık: Tüm noktalara uygulanacak dikiş payı kalınlığı

5.3.14 % Percentages (Yüzdeli)

Kural her nokta için X ve Y yönünde artımlar için yüzde olarak girilir. Şekil 5-19'da % Serilendirme Kuralı görülmektedir.



Şekil 5-19 % Serilendirme Kuralı

Fonksiyon:

PerformGrade (Tip="%", Nokta, YüzdeX, YüzdeY);

Parametreler:

Tip: "%"

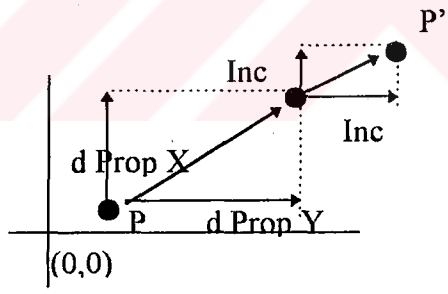
Nokta : Serilendirilecek nokta

YüzdeX: X Yönünde Yüzde

YüzdeY: Y Yönünde Yüzde

5.3.15 %I Percentages with Increments

Kural X ve Y yönünde yüzde olarak ilerletme yapıldıktan sonra X ve Y yönünde yerdeğiştirme yapılır. Şekil 5-20'de %I Serilendirme Kuralı görülmektedir.



Şekil 5-20 %I Serilendirme Kuralı

Fonksiyon:

PerformGrade (Tip="%I", Nokta, YüzdeX, YüzdeY,X,Y);

Parametreler:

Tip: "%I"

Parametreler:

Tip: "%I"

Nokta : Serilendirilecek nokta

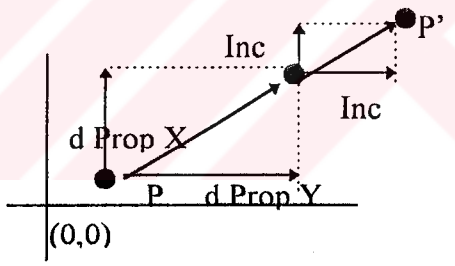
YüzdeX: X Yönünde Yüzde

YüzdeY: Y Yönünde Yüzde

X,Y: X ve Y yönünde artım

5.3.16 %IR Percentages with Increments Relative to the Reference Point (Referans Noktasına Bağlı Artımlı ve Yüzdeli)

Kural X ve Y yönünde yüzde olarak ilerletme yapıldıktan sonra X ve Y yönünde yerdeğiştirme yapılır. Yukarıdaki kuraldan farkı tüm orijinlerin referans noktasında alınmasıdır. Şekil 5-21'de %IR Serilendirme Kuralı görülmektedir.



Şekil 5-21 %IR Serilendirme Kuralı

Fonksiyon:

PerformGrade (Tip="%IR", Nokta, YüzdeX, YüzdeY,X,Y);

Parametreler:

Tip: "%IR"

Nokta : Serilendirilecek nokta

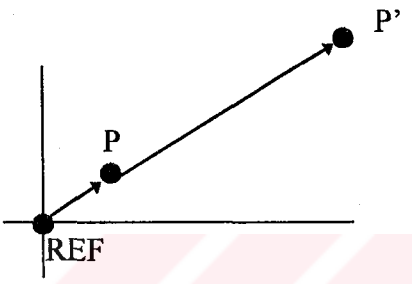
YüzdeX: X Yönünde Yüzde

YüzdeY: Y Yönünde Yüzde

X,Y: X ve Y yönünde artım

5.3.17 IRL **Increments Relative to the Line of Reference (Referans Çizgisine Bağlı Artımlı)**

Şekil 5-22'de IRL Serilendirme Kuralı görülmektedir.



Şekil 5-22 IRL Serilendirme Kuralı

5.3.18 NIL **NIL**

Bu kural serilendirmede değişiklik yapmaz.

5.4 Dikiş Payları (Seams)

Tablo 5-18 Dikiş Payı Tipleri

Kısaltma	Tip
A	Açı Angle
B	Kesişim Bisect

S	Simetri Symmetry
P	Prolongation Orantılı
CP	Orantılı Kes Cut Proportional
CD	Kalınlık Thickness
DB	Önünde Katlamalı Fold Before
DA	Arkasında Katlamalı Fold After
FB	Square Border Before Önünde Kare Köşeli
FA	Square Border After Arkasında Kare Köşeli
NC	No Seams Dikiş Payı Yok

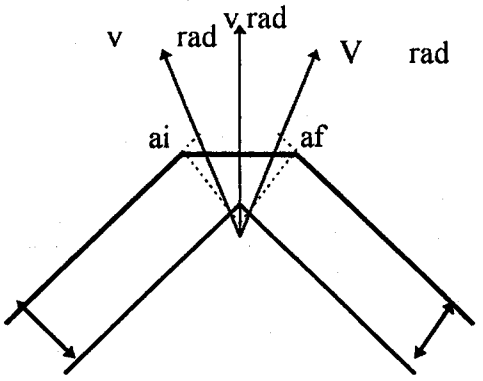
NIL	NIL
CONT	Sürekli Continue
NCB	No Seam Before Önünde Dikiş Payı Yok
NCA	No Seam After Arkasında Dikiş Payı Yok
TC	Intersect Curves Eğrileri Kesiştir

5.4.1 A Angle (Açı)

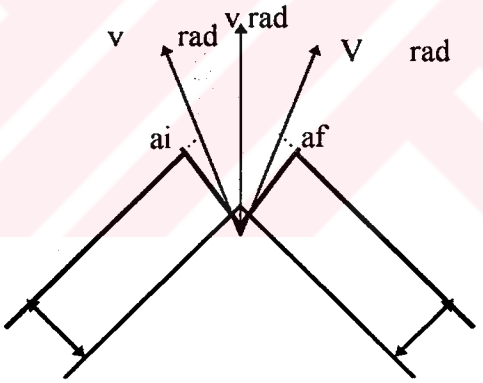
Dikiş payının uygulanacağı noktaya dikiş payı açısı tanımlar. Bu açı aşağıdaki değerlere bağlıdır:

- Dikiş payının uygulanacağı noktadan önceki açı ve kalınlık değerleri
- Dikiş payının uygulanacağı noktadan sonraki açı ve kalınlık değerleri

Kullanıcı bu noktada dikiş payını kesme veya kesmeme opsiyonuna sahiptir. Şekil 5-23'de A Dikiş Payı Tipi görülmektedir.



Şekil 5-23 A Dikiş Payı Tipi (Kesimsiz)



Şekil 5-24 A Dikiş Payı Tipi (Kesimli)

Fonksiyon:

CreateSeam(Tip , Nokta, Başlangıç Açısı, Bitiş Açısı,Başlangıç Kalınlığı,Bitiş Kalınlığı, KesimVarMı?);

Parametreler:

Tip: “A”

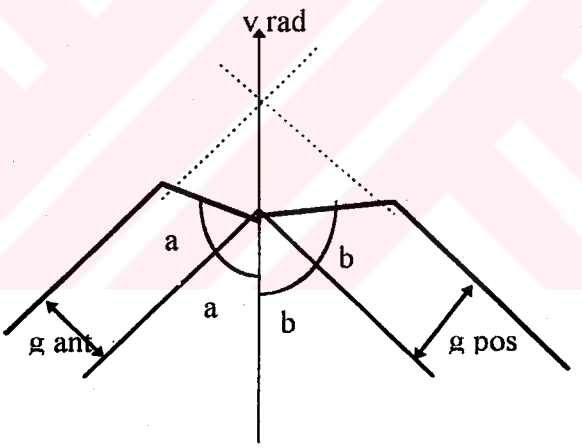
Kesim Var mı: Kesim uygulanacak mı?

5.4.2 B Bisect (Bölünmüş)

Dikiş payı noktasında belirli bir açıya bağlı olarak ikiye bölünmüş dikiş payı oluşturur. Bu bölme açısı aşağıdakilere bağlıdır: Şekil 5-25’de B Dikiş Payı Tipi görülmektedir.

Dikiş payı noktasından önceki kalınlık

Dikiş payı noktasından sonraki kalınlık



Şekil 5-25 B Dikiş Payı Tipi

Fonksiyon:

CreateSeam(Tip , Nokta, Başlangıç Kalınlığı,Bitiş Kalınlığı);

Parametreler:

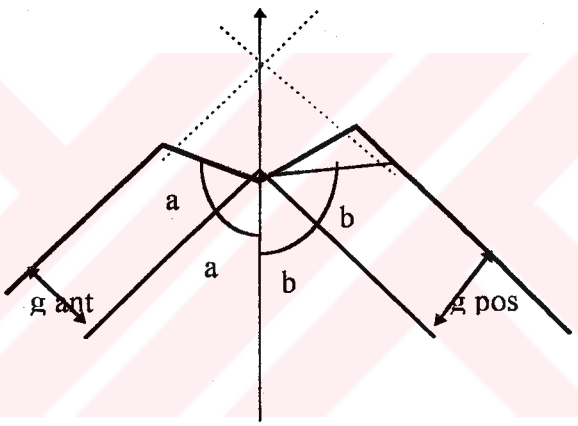
Tip: “B”

Başlangıç Kalınlığı

Bitiş Kalınlığı

5.4.3 S Symmetry (Simetri)

Simetri açılı dikiş payı yaratır. Noktadan önce ve sonraki kalınlık değerlerinden gelen doğrular kesiştirilerek bir nokta bulunur. Bu nokta ile profil noktası bir doğru ile birleştirilir ve simetri çizgisi oluşturulur. Açılar bu simetri çizgisi ile profil çizgilerinin oluşturduğu açıdan hesaplanır. Şekil 5-26'de S Dikiş Payı Tipi görülmektedir.



Şekil 5-26 S Dikiş Payı Tipi

Fonksiyon:

CreateSeam(Tip , Nokta, Başlangıç Kalınlığı,Bitiş Kalınlığı);

Parametreler:

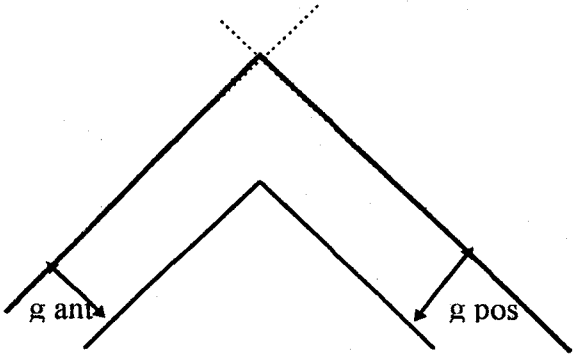
Tip: "S"

Başlangıç Kalınlığı

Bitiş Kalınlığı

5.4.4 P Prolongation (Orantılı)

Profil noktasının önündeki ve arkasındaki kalınlık değerlerinin uzatılması temeline dayalı dikiş payını oluşturur. Şekil 5-27'de P Dikiş Payı Tipi görülmektedir.



Şekil 5-27 P Dikiş Payı Tipi

Fonksiyon:

CreateSeam(Tip , Nokta, Başlangıç Kalınlığı,Bitiş Kalınlığı);

Parametreler:

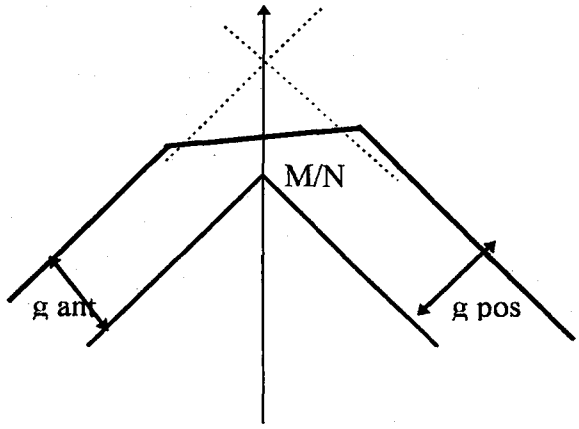
Tip: "P"

Başlangıç Kalınlığı

Bitiş Kalınlığı

5.4.5 CP Cut Proportional

Simetri çizgisi oluşturulduktan sonra profil noktasının önünden M ve arkasından N uzaklıklı noktalar bulunur. Bu noktaların oluşturduğu çizgi ters saat yönünde a açısı kadar döndürülerek kesim açısı bulunur. Şekil 5-28'de CP Dikiş Payı Tipi görülmektedir.



Şekil 5-28 CP Dikiş Payı Tipi

Fonksiyon:

CreateSeam(Tip , Nokta, Başlangıç Kalınlığı,Bitiş Kalınlığı, M, N, a);

Parametreler:

Tip: "CP"

N Uzaklığı

M Uzaklığı

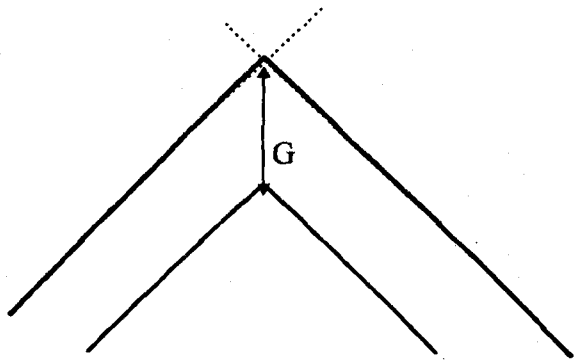
a Dönüş Açısı

Başlangıç Kalınlığı

Bitiş Kalınlığı

5.4.6 G Thickness (Kalınlık)

Profil noktasından istenilen kalınlıkta dikiş payı oluşturur. Şekil 5-29'da G Dikiş Payı Tipi görülmektedir.



Şekil 5-29 G Dikiş Payı Tipi

Fonksiyon:

CreateSeam(Tip , Nokta, Kalınlık);

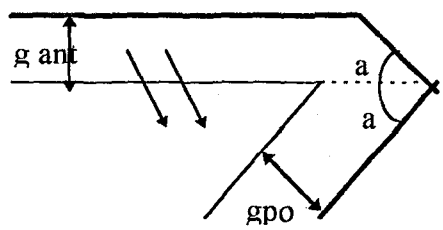
Parametreler:

Tip: "G"

Kalınlık

5.4.7 DB Fold Before

Nokta öncesi ve sonrası kalınlıklar belirlendikten sonra noktanın öncesindeki parça sonrası üzerine katlanacak şekilde dikiş payı yaratır. Şekil 5-30'da DB Dikiş Payı Tipi görülmektedir.



Şekil 5-30 DB Dikiş Payı Tipi

Fonksiyon:

CreateSeam(Tip , Nokta, Başlangıç Kalınlığı,Bitiş Kalınlığı);

Parametreler:

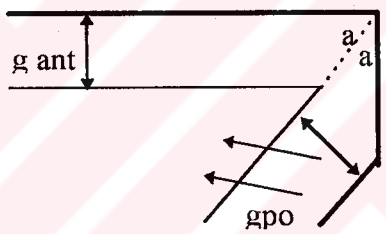
Tip: "DB"

Başlangıç Kalınlığı

Bitiş Kalınlığı

5.4.8 DA Fold After (Arkasında Katla)

Nokta öncesi ve sonrası kalınlıklar belirlendikten sonra noktadan sonraki parçanın parça önü üzerine katlanacak şekilde dikiş payı yaratır. Şekil 5-31'de dA Dikiş Payı Tipi görülmektedir.



Şekil 5-31 DA Dikiş Payı Tipi

Fonksiyon:

CreateSeam(Tip , Nokta, Başlangıç Kalınlığı,Bitiş Kalınlığı);

Parametreler:

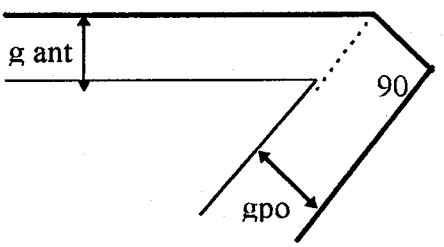
Tip: "DA"

Başlangıç Kalınlığı

Bitiş Kalınlığı

5.4.9 FB Square Border Before (Önünde Kare Köşeli)

Başlangıç ve son kalınlıkların oluşturduğu doğru belirlendikten sonra noktadan sonraki profil çizgisi uzatılarak noktadan önceki profil çizgisi ile kesiştiği noktada 90 derecelik çıkma ile oluşturur. Şekil 5-32'de A Dikiş Payı Tipi görülmektedir.



Şekil 5-32 FB Dikiş Payı Tipi

Fonksiyon:

CreateSeam(Tip , Nokta, Başlangıç Kalınlığı,Bitiş Kalınlığı);

Parametreler:

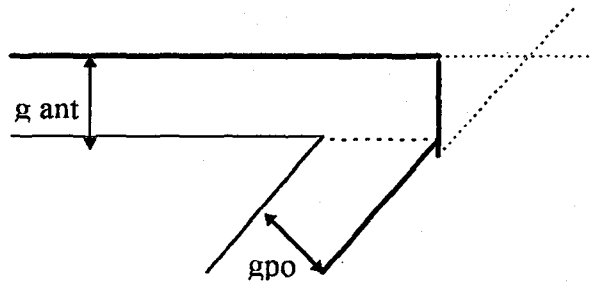
Tip: "FB"

Başlangıç Kalınlığı

Bitiş Kalınlığı

5.4.10 FA Square Border After (Arkasında Kare Köşeli)

Başlangıç ve son kalınlıkların oluşturduğu doğru belirlendikten sonra noktadan önceki profil çizgisi uzatılarak noktadan sonraki profil çizgisi ile kesiştiği noktada 90 derecelik çıkma ile oluşturur. Şekil 5-33'de fA Dikiş Payı Tipi görülmektedir.



Şekil 5-33 FA Dikiş Payı Tipi

Fonksiyon:

CreateSeam(Tip , Nokta, Başlangıç Kalınlığı,Bitiş Kalınlığı);

Parametreler:

Tip: "FA"

Başlangıç Kalınlığı

Bitiş Kalınlığı

5.4.11 NC No Seams (Dikiş Payı Yok)

İlgili nokta üzerinde dikiş payı tanımlarını kaldırır.

Fonksiyon:

CreateSeam(Tip , Nokta);

Parametreler:

Tip: "NC"

5.4.12 NIL NIL

İlgili nokta üzerinde boş bir dikiş payı tanımlar.

Fonksiyon:

CreateSeam(Tip , Nokta);

Parametreler:

Tip: "NIL"

5.4.13 CONTContinue (Sürekli)

Noktaya yumuşak geçiş yapacak şekilde dikiş payı tanımlar.

Fonksiyon:

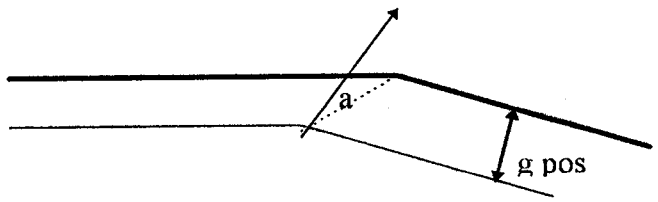
CreateSeam(Tip , Nokta);

Parametreler:

Tip: "CONT"

5.4.14 NCB No Seam Before (Önünde Dikiş Payı Yok)

Noktadan önceki dikiş payı devam ettirilerek noktadan sonraki dikiş payına birleştirilir. Şekil 5-34'de NCB Dikiş Payı Tipi görülmektedir.



Şekil 5-34 NCB Dikiş Payı Tipi

Fonksiyon:

CreateSeam(Tip , Nokta, Başlangıç Kalınlığı,Bitiş Kalınlığı);

Parametreler:

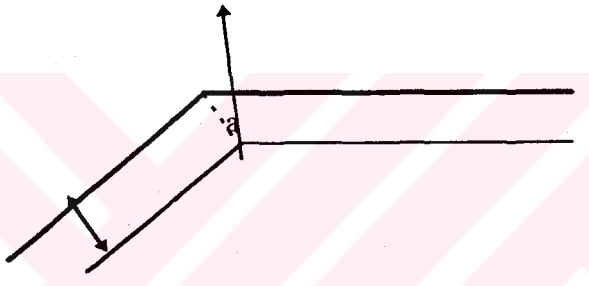
Tip: “NCB”

Başlangıç Kalınlığı

Bitiş Kalınlığı

5.4.15 NCA No Seam After (Arkasında Dikiş Payı Yok)

Noktadan sonraki dikiş payı devam ettirilerek noktadan önceki dikiş payına birleştirilir. Şekil 5-35’de NCA Dikiş Payı Tipi görülmektedir.



Şekil 5-35 NCA Dikiş Payı Tipi

Fonksiyon:

CreateSeam(Tip , Nokta, Başlangıç Kalınlığı,Bitiş Kalınlığı);

Parametreler:

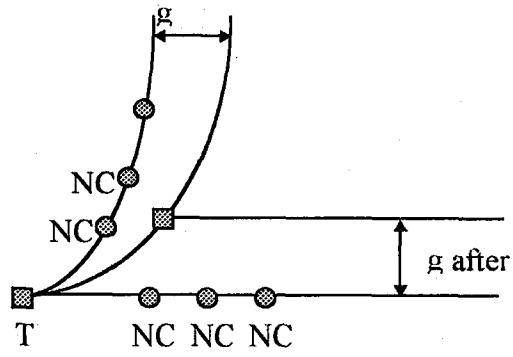
Tip: “NCA”

Başlangıç Kalınlığı

Bitiş Kalınlığı

5.4.16 TC Treatment Curve

Şekil 5-36'dA TC Dikiş Payı Tipi görülmektedir.



Şekil 5-36 TC Dikiş Payı Tipi

Fonksiyon:

CreateSeam(Tip , Nokta, Başlangıç Kalınlığı,Bitiş Kalınlığı);

Parametreler:

Tip: "TC"

Başlangıç Kalınlığı

Bitiş Kalınlığı

5.5 Dosya İşlemleri

5.5.1 Modeli Yükle

Sistemde varolan bir modelin çalışılmak üzere seçilmesi içi kullanılır. Verilen model kodu sistemde yoksa yeni bir model yaratılır.

Fonksiyon:

LoadModel(ModelKodu);

Parametreler:

ModelKodu:Çalışma ortamına yüklenecek modelin kodu

5.5.2 Modeli Sakla

Çalışılan model üzerinde yapılan değişiklikleri diske kaydeder.

Fonksiyon:

SaveModel;

Parametreler:

5.5.3 Kalıbı Yükle

Bir modelin belirli kalıpları ile çalışılmak isteniyorsa tüm kalıplar ekranda görülmek istenmiyorsa bu seçenekle modelin varolan kalıpları listelenir. Seçilen kalıp ekrana alınır.

Fonksiyon:

LoadPattern(KalıpKodu);

Parametreler:

KalıpKodu:Çalışma ortamına yüklenecek kalıbın kodu

5.5.4 Model Bilgisi

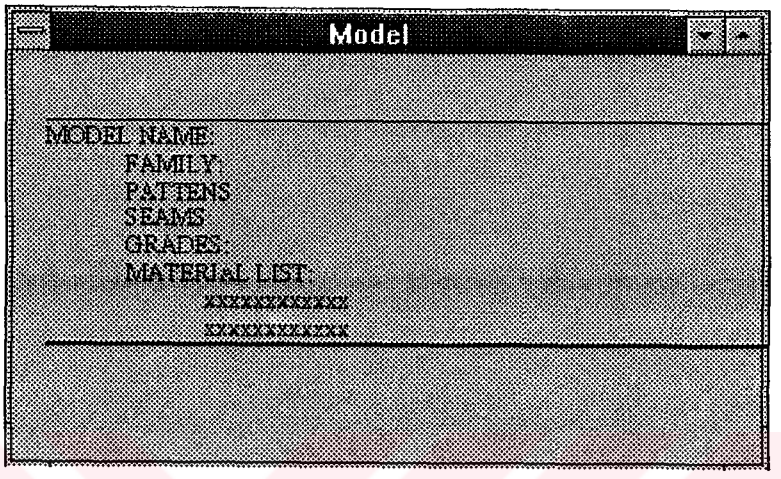
Model detay bilgilerinin sorgulandığı ekrandır.

Fonksiyon:

GetModelInfo(ModelKodu):

Parametreler:

ModelKodu: Katalog bilgileri görülmek istenen model kodu



Şekil 5-37 Model Katalog Bilgisi Ekranı

5.5.5 Kalıp Bilgisi

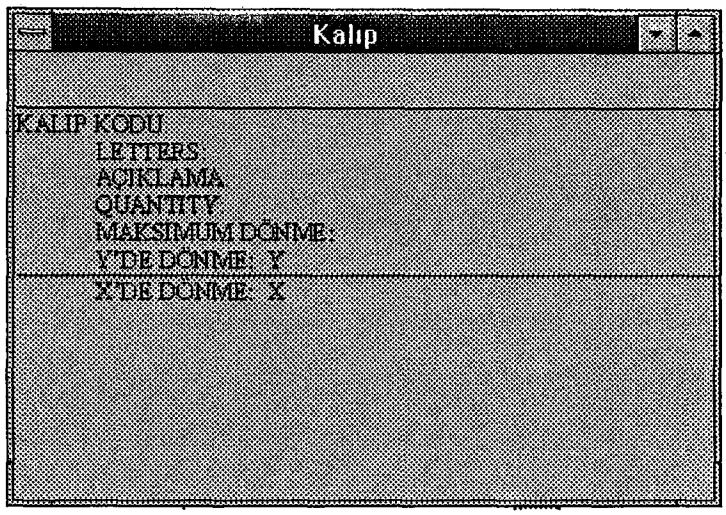
Seçilen kalıbın detay bilgilerini ekrana getirir.

Fonksiyon:

GetPatternInfo(KalıpKodu):

Parametreler:

KalıpKodu: Katalog bilgileri görülmek istenen kalıp kodu



Şekil 5-38 Kalıp Katalog Bilgisi Ekranı

5.6 Düz Nokta - Eğri Nokta Özelliğini Değiştirme

5.6.1 Düz Nokta - Eğri Nokta Özelliğini Değiştir

Seçilen noktanın düz noktadan eğri noktaya veya tersine değiştirilmesini sağlar.

Fonksiyon:

ChangePointType(Nokta,Tip);

Parametreler:

Nokta: Düz/Eğri özelliği değiştirilecek nokta

Tip Düz/Eğri

5.6.2 Seçili Bölge İçinde Düz Nokta - Eğri Nokta Özelliğini Değiştir

Seçilen nokta grubunun düz noktadan eğri noktaya veya tersine değiştirilmesini sağlar.

Fonksiyon:

ChangePointTypeInRegion(Region);

Parametreler:

Region:İçindeki noktaların özelliği değiştirilecek bölge. Bu bölge içindeki eğri noktalar düz, düz noktalar eğri nokta haline getirilir.

5.7 Eğri

5.7.1 Eğriyi Taşı

Tüm eğrinin mouse ile belirlenen bir notaya taşınmasını ağlar.

Fonksiyon:

MoveCurve (Curve, Point);

Parametreler:

Eğri: Taşınacak eğri

X,Y: Öteleme miktarları

5.7.2 Eğriyi Noktasından taşı

Eğri üzerinde seçilen noktanın hareket ettirilmesi ile tüm eğri yeniden oluşturulur.

Fonksiyon:

MoveCurve (Curve, Point);

Parametreler:

Eğri: Taşınacak eğri

X,Y: Öteleme miktarları

5.7.3 Eğriyi Kopyala

Bir kalıptan diğer bir kalıba eğri kopyalar. Birinci eğriden başlangıç ve bitiş noktaları arasındaki eğriye hedef kalıptan seçilen başlangıç ve bitiş noktaları arasına kopyalar.

Fonksiyon:

CloneObject;

5.8 Expand By Distance

Bir kalıbın veya kalıbın belirli bir bölgesinin belirli oranda küçültülmesi veya büyütülmesi için kullanılır. Önce işlem yapılacak ki nokta seçilir. Daha sonra istenilen uzunluk girilir.

Fonksiyon:

ExpandByDistance(Nokta1,Nokta2,Oran)

5.9 Kutu İçindeki Noktaları Taşı

Seçilen noktalar grubunu verilen arttırımlarla hareket ettirir.

Fonksiyon:

MovePointsInRegion(Region,X,Y);

Parametreler:

Region: Nokta seçim bölgesi

X,Y: Öteleme miktarları

5.10 Nokta Ekleme/Çıkarma

Kalıp profiline yeni noktalar eklenmesini sağlar.

5.10.1 Nokta Ekle

Seçilen bir başlangıç noktasından sonra fare ile seçilen koordinatlara yeni nokta ekler.

Fonksiyon:

AddPoint (X,Y);

Parametreler:

X,Y: Nokta eklenecek koordinat. Fare hareketlerinden alınır.

5.10.2 Belirli Bir Uzaklığa Nokta Ekle

Seçilen bir başlangıç noktasından sonra girilen uzaklıkta koordinata yeni nokta ekler.

Fonksiyon:

AddPoint (X,Y);

Parametreler:

X,Y: Nokta eklenecek koordinat. Uzaklığa bağlı olarak hesaplanır.

5.10.3 Belirli Koordinata Nokta Ekle

Seçilen bir başlangıç noktasından sonra girilen nokta koordinatına yeni nokta ekler.

Fonksiyon:

AddPoint (X,Y);

Parametreler:

X,Y: Nokta eklenecek koordinat,

5.11 Noktayı Taşı

Nokta hareketi sağlayarak kalıp profilinin değiştirilmesini sağlar.

Fonksiyon:

MovePoint (Nokta,X,Y);

Parametreler:

Nokta: Taşınacak nokta

X,Y: Nokta eklenecek koordinat

5.11.1 Noktayı Taşı

Fare ile işaretlenen bir noktayı görsel olarak hareket ettirir.

Fonksiyon:

MovePoint (Nokta,X,Y);

Parametreler:

Nokta: Taşınacak nokta

X,Y: Nokta eklenecek koordinat. Fare hareketlerinden sağlanır.

5.11.2 Noktayı Koordinata Taşı

Fare ile işaretlenen bir noktayı verilen X,Y koordinatlı noktaya hareket ettirir.

Fonksiyon:

MovePoint (Nokta,X,Y);

Parametreler:

Nokta: Taşınacak nokta

X,Y: Nokta eklenecek koordinat. Kullanıcı tarafından girilir.

5.11.3 X'e Getir

Fare ile işaretlenen bir noktayı seçilen noktanın X koordinatına kadar öteler.

Fonksiyon:

AlignX (Nokta,X);

Parametreler:

Nokta: Taşınacak nokta

X: Noktanın taşınacağı X değeri

5.11.4 Y'ye Getir

Fare ile işaretlenen bir noktayı seçilen noktanın Y koordinatına kadar öteler.

Fonksiyon:

AlignY (Nokta,Y);

Parametreler:

Nokta: Taşınacak nokta

Y: Noktanın taşınacağı Y değeri

5.11.5 Üç Noktayı Doğru Üzerine Taşı

Fare ile işaretlenen bir noktayı yine işaretlenen iki noktanın belirlediği doğru üzerine taşır.

Fonksiyon:

MakeLine(Nokta1, Nokta2, Nokta3);

Parametreler:

Nokta1: Nokta2 ve Nokta3 tarafından belirlenen doğru üzerine taşınacak nokta.

5.12 Nokta Silme

Kalıp profili üzerinden nokta siler.

Fonksiyon:

DeletePoint (Nokta);

Parametreler:

Nokta: Silinecek Nokta.

5.12.1 Noktayı Sil

Fare ile işaretlenen noktayı kalıp profilinden çıkarır.

Fonksiyon:

DeletePoint (Nokta);

Parametreler:

Nokta: Silinecek Nokta.

Fare ile Seçilmiş Ekran Bölgesi İçindeki Noktaları Sil

Fare ile işaretlenen noktalar grubunu kalıp profilinden çıkarır

Fonksiyon:

DeletePointsInRegion (Region);

Parametreler:

Region: İçindeki noktalar silinecek bölge.

5.13 Ekran İşlemleri

Ekranın veya kalıpların değişik bölümlerinin ekranda gösterilmesini sağlar.

Fonksiyon:

CenterScreen;

5.13.1 Ekranı Merkezle

Fare ile işaretlenen noktayı ekranın yeni merkezi olacak şekilde ekranı hareket ettirir.

Fonksiyon:

CenterScreenXY (X,Y);

Parametreler:

X,Y: Yeni ekran merkezi.

5.13.2 Zoom +

Ekran boyutunu arttırır.

Fonksiyon:

Zoom (Oran);

Parametreler:

Oran: Ekran büyütme oranı. >0

5.13.3 Zoom -

Ekran boyutunu azaltır.

Fonksiyon:

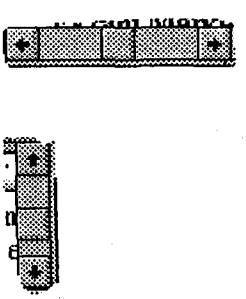
Zoom (Oran);

Parametreler:

Oran: Ekran küçültme oranı. <0

5.13.4 Kaydırma (Scrolling)

Kaydırma tuşlarını kullanarak ekranı aşağı, yukarı, sola, sağa doğru kaydırır. Fonksiyon Windows tarafından otomatik olarak desteklenir.



Şekil 5-39 Standart Windows Kaydırma Tuşları

5.14 Çakışma Noktaları

5.14.1 Dikey Çakışma Noktası

Modele dikey çakışma noktası ekler. Seçilen kalıp ile ekranda işaretlenen noktanın dikey çakışma noktasını bulur.

Fonksiyon:

VerticalMatchPoint (Kalıp,Nokta);

Parametreler:

Kalıp,Nokta: Çakışma noktası bulunacak kalıp, nokta ikilisi

5.14.2 Yatay Çakışma Noktası

Modele yatay çakışma noktası ekler. Seçilen kalıp ile ekranda işaretlenen noktanın yatay çakışma noktasını bulur.

Fonksiyon: HorizontalMatchPoint (Kalıp,Nokta);

Parametreler: Kalıp,Nokta: Çakışma noktası bulunacak kalıp, nokta ikilisi

5.14.3 Kare Çakışma Noktası

Modele kare çakışma noktası ekler. Seçilen kalıp ile ekranda işaretlenen noktanın kare çakışma noktasını bulur.

Fonksiyon:

SquareMatchPoint (Kalıp,Nokta);

Parametreler:

Kalıp,Nokta: Çakışma noktası bulunacak kalıp, nokta ikilisi

Çakışma Noktasını Sil

Çakışma noktasını siler.

Fonksiyon:

DeleteMatchPoint (Nokta);

Parametreler:

Nokta: silinecek çakışma noktası.

5.15 Özel İşaretler

5.15.1 Düz İpliği

Seçili kalıp üzerinde seçilen iki nokta arasına düz ipliği ekler veya değiştirir.

5.15.1.1 Dikey Düz İpliği

Seçili kalıp üzerinde seçilen iki nokta arasına dikey düz ipliği ekler. Düz ipliği dikey koordinat eksenine paralel olarak oluşturulur.

5.15.1.2 Yatay Düz İpliği

Seçili kalıp üzerinde seçilen iki nokta arasına yatay düz ipliği ekler. Düz ipliği yatay koordinat eksenine paralel olarak oluşturulur.

5.15.2 Referans Noktası

Seçili kalıp üzerinde seçilen bir noktaya referans noktası ekler. Kırmızı bir kare olarak görülür. Kalıp üzerindeki her noktanın koordinatı referans noktasına göre relatif olarak veri tabanına saklanır.

Fonksiyon:

ReferencePoint(Nokta);

Parametreler:

Nokta: Kalıbın yeni referans noktası.

5.15.3 Metin

Aktif kalıp plotter'a gönderilirken kalıp bilgisini içeren metinin nereye konulacağını belirler. Fare ile seçilir.

Fonksiyon:

TextCoordinate(Nokta);

Parametreler:

Nokta: Kalıbın metin noktası.

5.15.4 Başlangıç Noktası

Kalıp için yeni başlangıç noktası belirler.

Fonksiyon:

StartPoint (Nokta);

Parametreler:

Nokta: Kalıbın yeni başlangıç noktası.

5.16 Eğri Hassasiyeti

Kalıp üzerindeki eğrinin nokta sayısını arttırarak veya eksilterek eğrinin hassasiyetini değiştirir.

5.17 Geometri

Ekran üzerinde geometrik şekiller oluşturulmasını sağlar.

5.17.1 Daire

Merkez noktası fare ile seçilen nokta etrafında girilen çapa haiz daireyi oluşturur.

Fonksiyon:

Circle (Merkez, Çap);

Parametreler:

Merkez: Daire merkezi.

Çap: Daire çapı.

5.17.2 Kare

Ekranda kare şeklinde bir kalıp oluşturur.

Fonksiyon:

Square (Nokta1,Nokta2);

Parametreler:

Nokta1: Oluşturulacak karenin sol üst noktası.

Nokta2: Oluşturulacak karenin sağ alt noktası.

5.18 Ölçümler

Modeldeki kalıpları ölçeklendirir.

5.18.1 Area

Seçilen kalıbın alanını hesaplar.

Fonksiyon:

Double CalculateArea ();

5.18.2 Distance

Seçilen iki nokta arasındaki uzaklığı hesaplar.

Fonksiyon:

Double CalculateDistance(Nokta1,Nokta2);

Parametreler:

Nokta1, Nokta2: Aralarındaki uzaklık hesaplanacak noktalar.

5.18.3 Angle 3 Points

Seçilen üç nokta arasındaki açıyı hesaplar.

Fonksiyon:

Double CalculateAngle(Nokta1,Nokta2,Nokta3);

Parametreler:

Nokta1, Nokta2,Nokta3: Aralarındaki açı hesaplanacak noktalar.

5.18.4 Serilenmiş Uzaklık

Seçilen serilenmiş iki nokta arasındaki perimetreyi hesaplar.

5.18.5 Serilenmiş Perimetre

Seçilen serilenmiş iki nokta arasındaki uzaklığı hesaplar.

5.19 Kalıp Üzerindeki İşlemler

5.19.1 Deform In X

Seçili kalıbı X yönünde, girilen değer ile orantılı olarak deforme eder.

Fonksiyon:

DeformInX (Oran);

Parametreler:

Oran:

Deformasyon oranı.

5.19.2 Deform In Y

Seçili kalıbı Y yönünde, girilen değer ile orantılı olarak deforme eder.

Fonksiyon:

DeformInY (Oran);

Parametreler:

Oran:

Deformasyon oranı.

5.19.3 X Yönünde Esnet

Seçili kalıbı girilen esneme katsayısı oranında X yönünde esnetir.

Fonksiyon:

ElasticityInX (Oran);

Parametreler:

Oran:

Esneklik oranı.

5.19.4 Y Yönünde Esnet

Seçili kalıbı girilen esneme katsayısı oranında Y yönünde esnetir.

Fonksiyon:

ElasticityInY (Oran);

Parametreler:

Oran:

Esneklik oranı.

5.19.5 X Ekseninde Simetri

Seçili kalıbın X yönünde simetriğini alır.

Fonksiyon:

SymmetryInX ();

5.19.6 Y Ekseninde Simetri

Seçili kalıbın Y yönünde simetriğini alır.

Fonksiyon:

SymmetryInY ();

5.19.7 Kalıbı döndür

Eğer opsiyonlardan daha sonra değiştirilmemişse seçili kalıbı alfa açısı kadar döndürür.

Fonksiyon:

RotateObject (Açı);

Parametreler:

Açı: Radyan cinsinden döndürme açısı.

Kalıbı Sil

Seçili kalıbı siler.

Fonksiyon:

DeleteObject;

5.20 Zoom

5.20.1 Automatic Zoom

Ekranda tüm seçili kalıpların görülmesini sağlar.

5.20.2 Box Zoom

Pencere olarak seçili alanı büyütür.

5.20.3 Pattern Zoom

Kalıbı ekrana sığacak şekilde büyütür.

5.21 Kalıp Kesme

5.21.1 Two Points

Seçili kalıbı seçilen iki noktadan keser.

Fonksiyon:

CutPattern (Nokta1, Nokta2);

Parametreler:

Nokta1, Nokta2: Bu iki nokta tarafından belirlenen doğru ile kalıp kesilir.

5.21.2 Vertical

Seçili kalıbı seçilen noktanın dikey doğrultusundan keser.

Fonksiyon:

CutPatternVertical (Nokta1);

Parametreler:

Nokta1: Bu noktanın belirlediği dikey doğru ile kalıp kesilir.

5.21.3 Horizontal

Seçili kalıbı seçilen noktanın yatay doğrultusundan keser.

Fonksiyon:

CutPatternHorizontal (Nokta1);

Parametreler:

Nokta1: Bu noktanın belirlediği yatay doğru ile kalıp kesilir.

5.22 Uluslararası Dil Desteği

Bir uygulamanın birden fazla dilde çalışıyor olması diğer ülkelerde pazar payı olması demektir. Uygulamaya çokdilli çalışma özelliğinin eklenebilmesi için öncelikle veri yapısına bir karşılık tablosu eklenmelidir. Kullanıcı arayüzünde görünen her pencere öncelikle bu karşılık tablosuna bakarak üzerindeki etiket ve butonlarla, uygulamanın menü çubuğunda bulunan metinleri değiştirirler.

Tablo 5-19 Dil Çevirim Tablosu

Alan Adı	Alan Tipi	Alan Uzunluğu
Kod	Integer	5
Turkish	String	250
English	String	250

Bu amaçla her pencere kendi açılışında aşağıda sembolik tanımı verilen kodu çalıştırmalıdır.

TranslateWindow;

{

Pencere Üzerindeki Her Object İçin:

{

Object Tipi Label, Buton, Menü ise

```
                Object.Caption = GetText();   // Aktif dile bağlı olarak metni alır

        }

}
```

SONUÇLAR VE ÖNERİLER

Görüldüğü üzere bir grafik sisteminin Nesne Yönelimli bir programlama sistemi ile gerçeklenmesi hem tasarım anında tasarımcının sistem yapısını kurmasını kolaylaştırmakta hem de daha sonradan eklenebilecek ek özelliklerin sisteme çok kolay bir şekilde eklenebilmesini sağlamaktadır.

Tasarlanacak grafik sistemine ek olarak tasarlanmış kalıpların kumaş üzerinde kesilebilmesini sağlayacak kesici (cutter) sistemi de özellikle otomatik kontrolle ilgilenen kişiler için ilginç bir araştırma alanı oluşturacaktır.

İncelenecek bir başka konu da hazırlanmış kalıpların kumaş üzerinde optimal yerleştirilmesini içerecek bir yerleştirme (nesting) projesidir. Tekstil sektörünün en önemli konularından biri olan kumaş üzerine yerleşim üretim maliyetlerinin azaltılması açısından önem göstermektedir. Bu konuda yapılacak bir uygulama hem akademik araştırma olarak hem de ticari olarak ilgi toplayacaktır.

Tüm bu konularda çalışacak kişilere çalışma platformu olarak Windows ve Visual C++ ortamı hem yaygınlık hem de sağladığı araçlar olarak önerilir.

KAYNAKLAR

[1]Microsoft Press, Microsoft Visual C++ Programmers Guide, 1992

[2]Microsoft Press, Visual C++ Language Reference, 1993

[3]Microsoft Press, Visual C++ Run Time Library Reference, 1994

[4]M. E. Sarıdoğan, C++ ve Nesneye Yönelik Programlama, 1994

ÖZGEÇMİŞ

Sümer Çalbaş 1968 yılında İstanbul'da doğdu. 1985 yılında Kocaeli Gölcük Barbaros Hayrettin Lisesi'ni bitirdi. 1985 Dünya Gençlik Yılı'nda Kocaeli İli'ni temsilen Ankara'daki Gençlik Haftası Etkinliklerine katıldı. Lisans eğitimini İstanbul Teknik Üniversitesi Elektrik-Elektronik Fakültesi Kontrol ve Bilgisayar Mühendisliği Bölümü'nde 1992 yılında tamamladı.