T.C.
İSTANBUL SABAHATTİN ZAİM ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR BİLİMİ VE MÜHENDİSLİĞİ PROGRAMI

OCC-OPENCV KÜTÜPHANESİ İÇİN BLOK TABANLI PROGRAMLAMA ARACI

YÜKSEK LİSANS TEZİ

Harun ELKIRAN

İstanbul
Ocak, 2020

T.C.
İSTANBUL SABAHATTİN ZAİM ÜNİVERSİTESİ
BİLGİSAYAR BİLİMİ VE MÜHENDİSLİĞİ
YÜKSEK LİSANS PROGRAMI

OCC-OPENCV KÜTÜPHANESİ İÇİN BLOK TABANLI PROGRAMLAMA ARACI

YÜKSEK LİSANS TEZİ

Harun ELKIRAN

Tez Danışmanı
Dr. Öğr. Üyesi Aydın Tarık ZENGİN

İstanbul
Ocak, 2020

TEZ ONAYI

Fen Bilimleri Enstitüsü Müdürlüğüne,

Bu çalışma, jürimiz tarafından Bilgisayar Mühendisliği Anabilim Dalı, Bilgisayar Bilimi ve Mühendisliği Bilim Dalında YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

Danışman — Dr. Öğr. Üyesi Aydın Tarık ZENGİN

Üye — Doç. Dr. Tahir Çetin AKINCI

Üye — Dr. Öğr. Üyesi Gökhan ERDEMİR

Onay

Yukarıdaki imzaların, adı geçen öğretim üyelerine ait olduğunu onaylarım.

.....

Prof. Dr. Ahmet Korhan BİNARK

Enstitü Müdürü

BİLİMSEL ETİK BİLDİRİMİ

Yüksek lisans tezi olarak hazırladığım "**OCC - Opencv Kütüphanesi İçin Blok Tabanlı Programlama Aracı**" adlı çalışmanın öneri aşamasından sonuçlandığı aşamaya kadar geçen süreçte bilimsel etiğe ve akademik kurallara özenle uyduğumu, tez içindeki tüm bilgileri bilimsel ahlak ve gelenek çerçevesinde elde ettiğimi, tez yazım kurallarına uygun olarak hazırladığımı, bu çalışmamda doğrudan veya dolaylı olarak yaptığım her alıntıya kaynak gösterdiğimi ve yararlandığım eserlerin kaynakçada gösterilenlerden oluştuğunu beyan ederim.

Harun ELKİRAN

ÖNSÖZ

Araştırmamdaki her aşamada bana yardımlarını ve desteklerini esirgemeyen değerli tez danışmanım Dr. Öğr. Üyesi A. Tarık ZENGİN'e, eğitim alanında dersleriyle bize vizyon katan çok değerli hocalarıma, yüksek lisans eğitimim boyunca her aşamada beni destekleyen ve yüksek fedakârlık gösteren sevgili eşim Elif'e, küçük meleğim Zeynep'e ve aileme teşekkürlerimi sunarım.

Harun ELKIRAN

İstanbul, 2019

ÖZET

OCC-OPENCV KÜTÜPHANESİ İÇİN BLOK TABANLI PROGRAMLAMA ARACI

Harun ELKIRAN

Yüksek Lisans, Bilgisayar Bilimleri ve Mühendisliği

Tez danışmanı: Dr. Öğr. Üyesi Aydın Tarık ZENGİN

Ocak-2020, 82 Sayfa + XII

Bu tez, web tabanlı ve genel amaçlı OpenCV kütüphanesi için görsel ve blok akış tabanlı programlama platformunun tasarımını ve uygulamasını sunmaktadır. Platform bir görsel programlama ara yüzü, işleme sistemi, çalışma alanı ve Python için kod çıktısı üreten arabirime sahip uygulama yapısından oluşur. Bu arayüz sayesinde web tarayıcı üzerinden kullanıcılar platforma erişebilir, OpenCV fonksiyonlarını kontrol edebilir ve sonuçlarını görüntüleyebilir. Uygulama ara yüzü akış tabanlı programlama sisteminden esinlenerek tasarlanmıştır ve her işlem giriş ve çıkışları olan düğümler üzerinden yürütülür. Platform ile çıktı soketlerinin diğer düğümlerin giriş soketlerine bağlanması, verilerin düğümler arasında akması ve işlem hatlarının oluşturulması sağlanmaktadır. Ayrıca sistem programlama uzmanlığı olmayan kullanıcıların kolayca karmaşık veri akışı işlemleri oluşturmasına ve çalıştırmasına olanak tanır.

Anahtar Kelimeler: OpenCV Kütüphanesi, Akış Tabanlı Programlama

ABSTRACT

OCC: A BLOCK-BASED PROGRAMMING TOOL FOR OPENCV LIBRARY

Harun ELKIRAN

Master of Science, Computer Sciences and Engineering

Supervisor: Dr. Aydin Tarik Zengin

January-2020, 82 Pages + XII

This dissertation presents visual and block cast based programming design and application platform for flow-based and general purpose opencv library. The platform is composed of a visual programming interface, a processing system, workspace, and an application structure which has an interface and produces code output for python. Through this interface, the users can reach the platform by web, can also control the opencv functions, and observe the results. The application interface was designed from inspiration of flow-based programming system. Every process is carried out through nodes which have inputs and outputs. Connections between platform and exit sockets with nodes of entrance of other sockets; flow of datas throughout the nodes, and lines of process are available. It also provides to users, who do not have expertise on system programming, easily constitution and operation of complex data flow processes on opecv library.

Keywords: OpenCV Library, Flow-Based Programming

İÇİNDEKİLER

ŞEKİLLER LİSTESİ

KISALTMALAR LİSTESİ

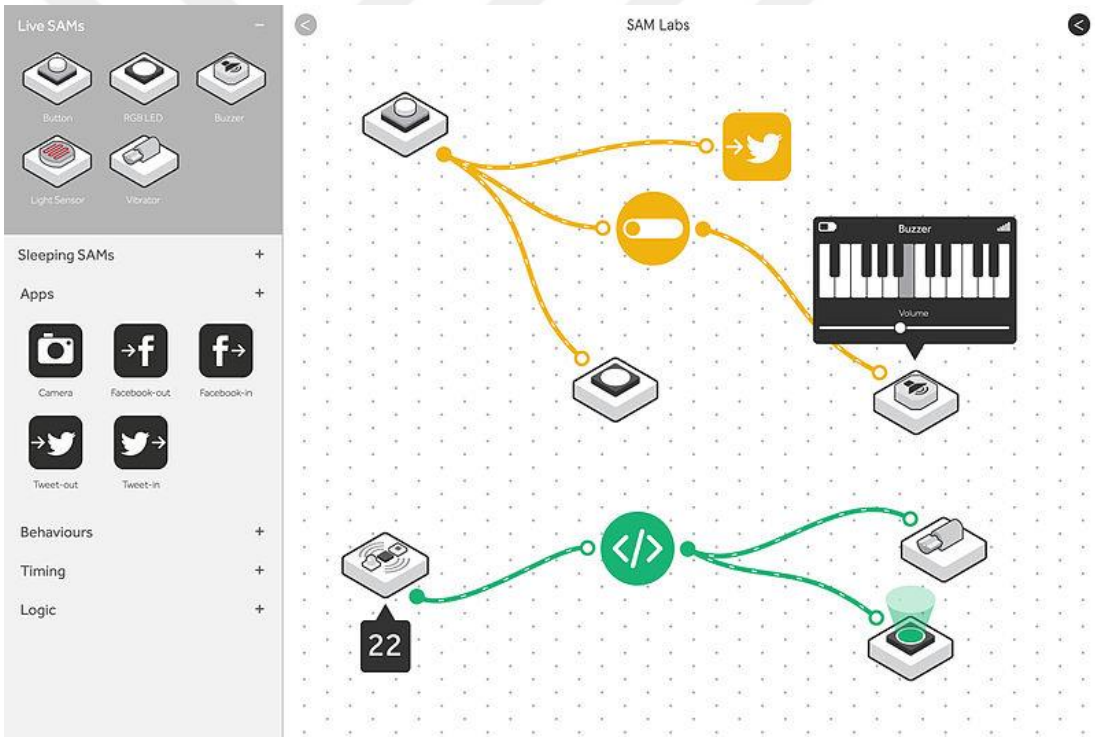
3D	:	3 Dimension : 3 Boyutlu
API	:	Application Programming Interface : Uygulama Programlama Arayüzü
BDL	:	Backtracking Data Flow Programming Language
BSD	:	Berkeley Software Distribution
BT	:	Bilgi Teknolojileri : Information Technology
ConMan	:	Connection Manager
CSS	:	Cascading Style Sheets
DFP	:	Data-Flow Programming : Veri-Akışlı Programlama
FBP	:	Flow-Based Programming
FTP	:	File Transfer Protocol : Dosya Aktarım Protokolü
GRC	:	GNU Radio Companion
HTML	:	HyperText Markup Language
IaaS	:	Infrastructure as a Service : Hizmet olarak Altyapı
IP	:	Information Package : Bilgi Paketi
IVI	:	intuitive visual interface : Sezgisel Görsel Arabirim
JSON	:	JavaScript Object Notation
ML	:	Machine Learning : Makine Öğrenimi
MLA	:	Multi-Layered Architecture : Çok-Katmanlı Mimari
MVC	:	Model, View, Controller
NIST	:	National Institute of Standards and Technology : Ulusal Standartlar ve Teknoloji Enstitüsü
NoSQL	:	Not only SQL :
OCC	:	OpenCV Companion
OOP	:	Object Oriented Programming : Nesne Yönelimli Programlama
ORM	:	Object-Relational Mapping

PaaS	:	Platform as a Service : Hizmet olarak Platform
PD	:	Pure Data
PL	:	Process Line : İşlem Hattı
POSIX	:	Portable Operating System Interface : Taşınabilir İşletim Sistemi Arayüzü
REST	:	Representational State Transfer
RF	:	Radio Frequency : Radio Frekansı
SaaS	:	Software as a Service : Hizmet olarak Yazılım
SDLC	:	Systems Development Life Cycle : Sistem Geliştirme Yaşam Döngüsü
SMTP	:	Simple Mail Transfer Protokol : Elektronik Posta Gönderme Protokolü
SOA	:	Service Oriented Architecture : Servis Odaklı Mimari
SOAP	:	Simple Object Access Protocol : Basit Nesne Erişim Protokolü
SQL	:	Structured Query Language : Yapılandırılmış Sorgu Dili
SSE	:	Streaming SIMD Extensions
STL	:	Standard Template Library : Standart Şablon Kütüphanesi
UI	:	User Interface : Kullanıcı Arayüzü
UX	:	User Experience : Kullanıcı Deneyimi
VFX	:	Visual Effects : Görsel Efektler
VPL	:	Visual Programming Language : Görsel Programlama Dili

BİRİNCİ BÖLÜM

1. GİRİŞ

OCC (Opencv Companion) uygulamasının temel fikri programlama bilgisi çok az olan veya hiç olmayan kullanıcıların OpenCV kütüphanesine erişimini sağlamak, uygulama yapmak ve ideal olarak Simulink [35], GNU Radio [36] gibi platformlara benzer sezgisel bir görsel arabirim (intuitive visual interface) geliştirmek olarak ifade edilebilir. Fikrin arka planındaki ilkeler görsel programlama ile ilgilidir [1, 5]. Veri akışı programlama, tüm süreci girdi ve çıktılarla soyut bir kara kutu (black box) olarak temsil edilen programlama paradigmasıdır. Bir kara kutunun çıktıları bir sonrakini girdi olarak besler ve bu şekilde ham verileri sonuçlara dönüştürene kadar işlem hattı (process line) devam eder (Şekil 1.1).



Şekil 1.1: Akış tabanlı programlamaya sahip SAM Yazılım Arayüzü[1]

Kara kutular temsil ettikleri süreçler göz önünde tutulduğunda *verileri çıktılara dönüştüren algoritmalar* olarak tanımlanabilir. Yeterli programlama becerisine sahip kullanıcılar tarafından geliştirilmektedir ve bir kez geliştirildikten sonra, kara kutu

[1] https://int.samlabs.com/

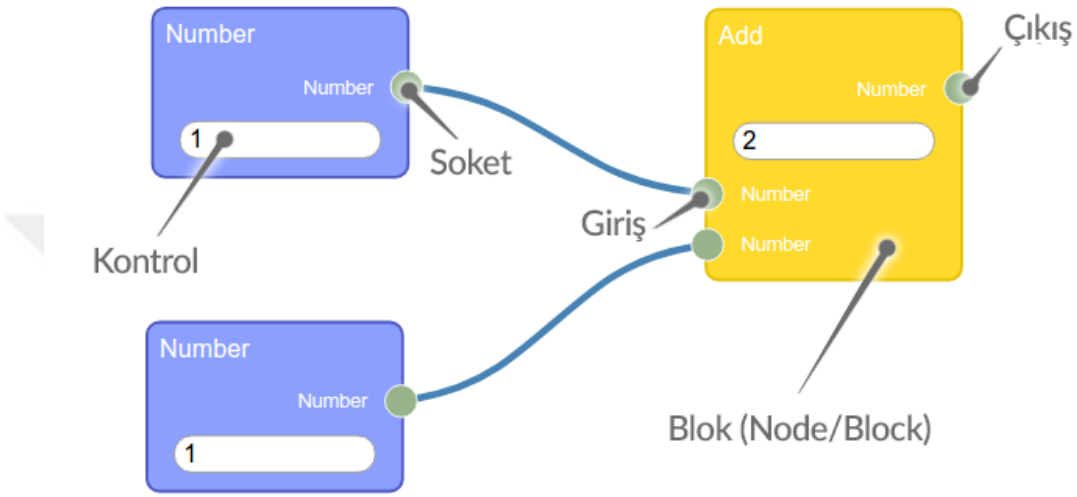
işlemleri, onları anlamlı bir şekilde birbirine bağlayabilen herkes tarafından kullanılabilir.

Kullanıcı için ardışık düzenin (pipelines) kurgulanması ve oluşturulması aşamasında sürecin doğruluğu ve tüm ilgili verilerin arka planda işlendiğinden emin olunması çıktıların anlık belirsizliği doğrultusunda can sıkıcı olabilir. Bu nedenle, tüm sürecin otomasyonu, bir sürecin başlatılabileceği koşulu oluşturan tetikleyiciler kullanılarak öngörülmüştür. Kullanıcılar, ilgili girdi verilerinde işlemleri otomatik olarak çalıştıran tetikleyiciler ayarlayabilir. Bu tür tetikleyicilerin zincirlenmesi ardışık düzen işlemleri oluşturabilir. Bu şekilde, ardışık düzen işlemleri oluşturmak için kullanıcının işlemlere karşılık gelen tetikleyicileri bir kez ilgili işleme ataması gerekir. Daha sonra, platforma her yeni veri eklendiğinde, sadece tetikleyiciler ayarlanarak otomatik olarak sürecin işlemesi sağlanır. Veri akışı programlama için otomasyon önemlidir çünkü bazı işlem ve görevlerinin tamamlanması çok uzun sürebilir. Tetikleyiciler, işlemin denetim olmadan devam edebilmesini sağlar. Tetikleyicilerin bir diğer önemli yönü ise girdilerdeki değişiklikleri işleme yeteneğidir. Ardışık düzen işlem hattı içindeki bir işlemin bazı parametrelerinin değiştirilmesi daha önce bu işlem için oluşturulan ayarların geçerliliğini yetirmesine sebep olacaktır. Tetikleyiciler, bu tür değişiklikleri algılarsa ardışık düzenin geri kalanını otomatik olarak güncelleyebilir veya kullanıcıyı bu tür tutarsızlıklar konusunda bilgilendirecek şekilde yapılandırılabilir.

Önerilen platform (OCC), kapsamı bakımından opencv kütüphanesine bağlıdır ve prensip olarak kütüphanede mevcut olan her türlü veriyi işlemelidir. Ancak, başarılı olmak için daha net bir strateji ve odaklanma gereklidir. Platformun ana odağı opencv kütüphanesinin sunduğu fonksiyonların görsel programlama ile görselleştirilmesi ve python[2] çıktısı üretmesidir. OCC için yürütülen uygulama fikri, çok katmanlı ve gevşek bir şekilde bağlanmış (loose coupling) mimaridir. Sistemin her parçası bağımsız ve değiştirilebilir olmalı, parçalar arasındaki tüm iletişim uygulama programlama arabirimleri (API) aracılığıyla yapılmalıdır. Uygulamanın çekirdek katmanları web uygulaması, uygulama sunucusu ve işleme katmanıdır. Bazı katmanlar için uygulamaya yönelik teknolojiler modern kütüphane ve çerçevelerden seçilmiştir. Diğer katmanlar için ise (özellikle web uygulamasına bakan istemci) ek araştırmalar

[2] https://www.python.org/

gerektirmektedir. Öncelikle web uygulaması oluşturmak için bir JavaScript çerçevesi araştırması yapıldı (ayrıntılar için bkz. bölüm 2.3). Teknoloji çözümleri belirlendikten sonra mimari planlamaya geçildi. Son olarak, uygulama yazılım geliştirme döngüsü için önce temel bir uygulama oluşturmak ve yavaş yavaş ek özelliklerle genişletilen ve yükseltilen tekrarlanmalı işlem (iterative process) [55] ve şelale modeli (waterfall model) [56] seçildi.



Şekil 1.2: OCC Editör Ekranı.

Geliştirme sırasında dayanılan teorik kavramlar ve mimari uygulamalara bölüm 1.1.'de değinilmiştir. Bölümün ilk kısmı çok katmanlı yazılım mimarisi ile ilgili kavramlar, veri akışı programlaması, tezin ana konseptini oluşturan OpenCV kütüphanesine genel bir bakış ve hem ilgili masaüstü hem de web uygulamalarının incelendiği benzer çalışmaları kapsamaktadır. Bölüm 2, platformu uygularken teorik kavramların ve fikirlerin nasıl uygulandığını açıklamaktadır. Ayrıca, bölüm 2 geliştirme hedefleri ve mimarisi ile başlar, uygulamanın her bir bölümü için daha spesifik çözümlerle devam eder ve son olarak kilit parçaların uygulama detaylarını açıklar. Bölüm 3, yapılan çalışmalara odaklanarak ve gelecekte de uygulanabilecek diğer özellikleri göz önünde bulundurarak Tez'in sonuç ve önerilerini kapsar. Bu tez projesi ile görülen lüzum 18 milyondan fazla kullanıcısı[3] olan opencv kütüphanesinin kullanımını IDE bağımsız sağlamak ve daha önce yapılmamış uygulama eksikliğini kapatmak olacaktır. Bu yüzden uygulama kendi alanında ilk olacaktır.

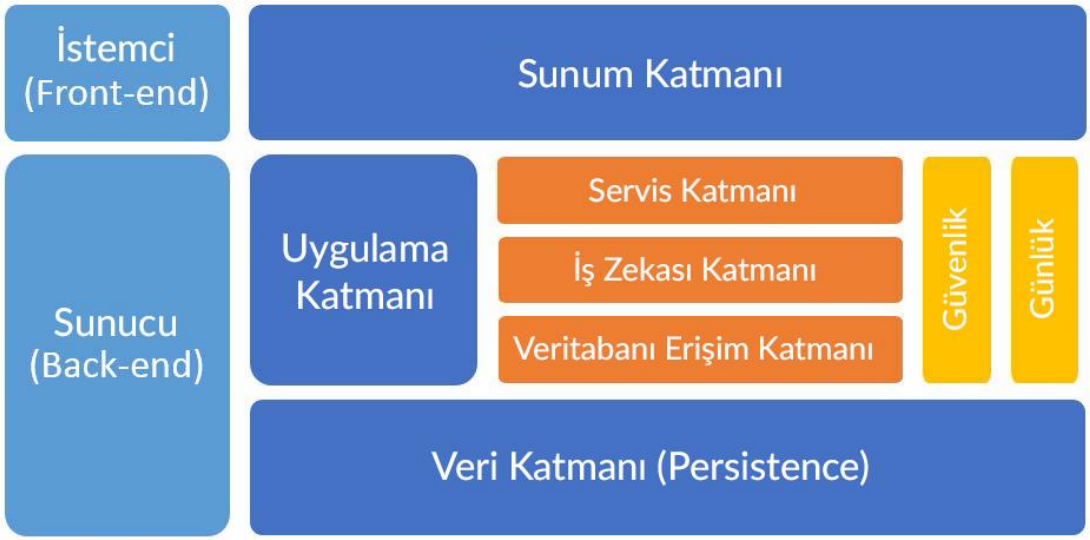
[3] https://sourceforge.net/projects/opencvlibrary/files/stats/timeline?dates=2001-09-20+to+2019-01-30

1.1. Mimari

Günümüzde genel olarak istemci-sunucu (client-server) mimarisi internet tabanlı herhangi bir uygulamanın iskeletini oluşturmaktadır. Bu durum aynı şekilde web tarayıcıları için de merkezi işlem (centralized computing) zamanlarından, istemci-sunucu iletişimine dönüştü. Bulut bilişimin gelişmesiyle birlikte, uygulamaların mimarisi çok daha karmaşık hale geldi, ancak en genel ifadeyle sistem hala bir servis sağlayıcısının (service provider) ve bir servis alıcısının (service consumer) istemci-sunucu ayrımına bağlı. Modern web uygulamaları için en iyi tasarım uygulamaları ve standartları çok katmanlı mimari (multi-layered architecture) temelli geliştirilmektedir. Çok katmanlı mimaride en yaygın olarak üç katman vardır: sunum katmanı, uygulama (iş, iş mantığı, mantık veya sadece orta katman olarak da adlandırılır) katmanı ve persistans (veya veri) katmanı. Bir mimariyi tarif ederken katman (layer / tier) teriminin kullanımı ile ilgili bazı karışıklıklar vardır, bu iki kelime kullanım alanına göre genellikle karıştırılmaktadır. Terimler arasındaki fark kullanım alanlarına göre değişmektedir [12]. Örneğin: “tier” fiziksel bir ayrımı (farklı makine gibi.) ifade eder. Dolayısıyla çok katmanlı bir uygulama mutlaka çok katmanlı fiziksel makineye ihtiyaç duymaz, uygulama ve veri katmanı aynı fiziksel makinede çalışabilir.

Bir web uygulamasının sunum katmanı (presentation layer), kullanıcının uygulama ile etkileşime girdiği ara yüzdür. Bu katmanda bilgiler ve sonuçlar görüntülenir, kullanıcıların eylemleri işlenir ve gerektiğinde bu eylemler uygulama katmanı için taleplere dönüştürülür. Sunum katmanı, istemci-sunucu mimarisinin istemci kısmıdır ve bu nedenle genellikle ön uç (front-end) olarak da adlandırılır; aynı benzetme diğer katman için de geçerlidir ve arka uç (back-end) terimi uygulamanın sunucu kısmına atıfta bulunmak için kullanılır. Veri katmanı (persistence layer), veri depolama ve depolanan verilere erişim ile ilgilidir. Sunum katmanı ve veri katmanı arasında iletişimi ve web uygulamasının tüm işlevlerini işleyen uygulama katmanı (application layer) bulunur. Uygulama katmanı genellikle katmanların en karmaşık olanıdır ve genellikle kendi içinde de çok katmanlı bir yapıya dönüşür. Bu nedenle, üç katmanlı mimari kendi içinde ek katmanlara ayrılması dikkate alınarak çok- veya n-katmanlı mimari olarak adlandırılır. Uygulama katmanının içindeki ek katmanlara örnek olarak hizmet, iş mantığı / etki alanı, veri erişimi ve işleme katmanlarını

verebiliriz. Bahsi geçen katman ayrımları mantıksal ya da fiziksel olabilir. Uygulama katmanının bazı bileşenleri mantıksal olarak ayrı olsa bile örneğin günlük (logging) veya güvenlik (security) gibi uygulama parçalarının performans işlevselliğini görüntülemek için katmanlar arasında dikey olarak birleştirilebilir. Örnek mimari Şekil 1.3'de gösterilmektedir ve güvenlik (security) ve günlük (logging) blokları, uygulama katmanının iç katmanları boyunca erişilebilir modüllere örnektir.



Şekil 1.3: İstemci ve sunucu (Client/Server) ayrımını katmanlara göre gösteren çok katmanlı bir mimari.

Hizmet katmanı, sunum katmanının bağlandığı ara katmanı (middle-ware) temsil etmektedir. Hizmet katmanı, kullanıcı ara yüzü ve uygulama katmanı arasında tipik talep yanıt iletişimini sağlar. Böyle bir soyutlamanın en önemli faydası çeşitli istemcilerin uygulamaya erişebilmesini sağlayan API servisidir. API ile hizmete sadece kullanıcı arabirimi yerine diğer uygulamalar tarafından da erişilebilir. Bu kavram bölüm 1.1.4'de daha ayrıntılı olarak açıklanmaktadır. İş mantığı veya etki alanı katmanı (business logic / domain layer) problem alanının kavramları, kuralları ve işlevselliğinin bir soyutlamasıdır ve verilerle neler yapılabileceğinin olasılıklarını kodlar. Bu katmana “yazılımın kalbi” diyebiliriz [20]. Veri erişim katmanı, veri katmanıyla iletişimden sorumlu olan alt orta-katmandır. Veri erişim katmanı genellikle nesne-ilişkisel eşleştirme (object-relational mapping) sistemini içerir.

Son olarak, işlem katmanı (processing layer) vardır, ancak işlem "parçası" ifadesi muhtemelen daha doğru olacaktır. Geleneksel mimari tanımlar ana tartışma

çerçeveyi oluşturmamaktadır, ancak bu mimari disiplinlerin yazılım mimarilerinin önemli bir konusu olduğu unutulmamalıdır. İşlem katmanı, işlem birimlerinin (processing units) planlanmasını yöneten katman olarak anlaşılabilir. Bir kullanıcı tarafından arabirim aracılığıyla talep edilen işler uzun sürüyor ve çok sayıdaysa, yürütülmesi için özel bir sistem gerekir. Uygulamanın ölçeklenebilirliği (scalability) göz önünde tutulduğunda bu konu önemli hale gelir ve bölüm 1.1.2'de daha ayrıntılı incelenmektedir.

Çok katmanlı mimarinin en büyük faydası *kaygıların ayrılması* ilkesinden (separation of concerns) [3] kaynaklanmaktadır. Kaygıların ayrılması disiplini temel yazılım mimarisi paradigmalarından biridir ve bütünü farklı kaygıları ele alan parçalara (katmanlara) ayırmayı amaçlamaktadır. Bu teknik, yazılım gerçekleştirim aşamasında kullanıcının kaygılarını bütün yerine katmanlara bağımsız bir yönde odaklamasının disiplini olarak açıklanabilir [4]. Bu prensip ayrıca modüler tasarım fikrinin hayata geçirilebilmesi için temeldir.

1.1.1 Modüler Tasarım

Modüler tasarım, ürünün esnekliğinin ve anlışabilirliğinin yanı sıra geliştirme süresini de kısaltan tasarım ilkesi olarak D. Parnas [9] tarafından dünyaya tanıtılmıştır. Bir sistem için iyi bir modüler tasarım kullanımı *bilgi gizlemenin anahtarı* olarak adlandırılabilir. İlgili model ile uygulamanın kısmi veya genel düzenlemesi, hata ayıklaması çok kolay olacaktır. Modüler tasarım sayesinde bütünsel işlevsellik değişikliğinde dahi her zaman arabirime erişim sağlanacaktır.

W. Stevens [11] bir sistemin modülerliğini tanımlamak için iki ek terim kullanmaktadır- bağlılık ve bağlaştırma (*cohesion and coupling*). Bağlılık (cohesion), bir modülün elemanlarının birbirine bağlanma derecesini tanımlar; her modül hem uygulaması hem de gerçekleştirebileceği işlevler açısından son derece uyumlu olmalıdır. Örneğin, birden fazla işlevi olan bir uygulamanın bir kısmı, her biri kendi içinde farklı işlevi olan modüllere ayrılarak uygulamanın bütünlüğü sağlanabilir. Bağlaştırma (coupling) modüller arasındaki ilişkiyi açıklar ve birbirlerine ne kadar bağlı olduklarını inceler. Modüler tasarım bağımsız modülleri ilke edinir ve bir modülün diğer modüllerin uygulamasında çok az veya hiç bağı olmamalıdır. İdeal

olarak, aynı ara yüze ve tamamen farklı uygulamalara sahip iki modül, sistemi bozmadan değiştirilebilir olmalıdır.

Nesne yönelimli programlama (OOP) paradigmalarından yazılım kütüphanelerinin geliştirilmesine kadar modüler tasarım örnekleri tüm yazılım geliştirme seviyelerinde bulunabilir. Modüler tasarımı daha yüksek seviyelerde bir sistemin bölümlerini mantıksal birimlere ayırmada görebiliriz. Modüler tasarım ilkesi uygulama programlama ara yüzleri aracılığıyla iletişim kuran modüller ile sonuçlandırılır ve modüller arasında çok az veya hiç bağ olmamalıdır [12]. API, bir modülün işlevselliğine erişildiği sabit arabirimdir. API işlevselliğin bir spesifikasyonu olarak da anlaşılabilir. Bir yazılım parçasının yeteneklerini ortaya çıkarmak için arabirimini tanımlamak ve aynı zamanda işlevselliğini uygulamak API için geliştirme sürecinin her iki yönde de olabileceğini ifade eder. Bazen API'ler standartlaştırılabilir, örneğin Unix işletim sistemlerinin hizmet spesifikasyonlarını tanımlayan POSIX standart ailesi gibi. Konu Bölüm 1.1.4'de daha ayrıntılı olarak açıklanmaktadır. Yukarıda belirtildiği üzere, eşdeğer API'ye sahip modüller, uygulamalarına bakılmaksızın değiştirilebilir olmalıdır.

Modüler tasarımın faydalarının incelenmesine Parnas'ın hedeflerine ve bu hedeflere nasıl ulaştığına bakılarak başlanabilir. Parnas'a göre modüllerin geliştirilmesinde ortak ara yüzler üzerinde çalışma alanı oluşturulduktan sonra aynı anda farklı bağımsız modüller geliştirilebilir ve bu durumda *geliştirme süresi* kısalır. Bir modülde yapılan değişiklik sadece o modülü etkilediği için uygulamanın ek bölümlerinde değişiklik olmayacaktır, yine aynı şekilde *bakım süresi* de kısalacaktır. İyi kurgulanmış modüler tasarım sayesinde uygulama üzerinde yapılan değişiklikler, uygulamanın diğer bölümlerini etkileyen *dalgalanma etkisini* önleyecektir [13]. Tek parça geliştirilen ve tek bir sistem üzerinde sıkıca bağlanmış uygulamalara kıyasla modüler tasarım önemli ölçüde dinamiktir, çünkü her modül uygulanan role bağlı olarak uygulamanın birden çok parçası yerine tek bir noktasında işlevsellik gösterecektir. Bu durumda *esneklik* de artar. Yukarıda belirtildiği gibi, bir modül ideal olarak eşdeğer bir API ile farklı bir modülle kolayca değiştirilebilir olmalıdır. Bu durum yazılımın modernizasyonu veya yenilikçi alternatif çözümlerin uygulanması aşamasında önemli hale gelecektir (bu durumda sadece ara yüzün uyumlu olması dahi güncellenen modülü sisteme entegre etmek için kolaylık sağlayacaktır).

Modülün yeniden kullanılabilirliği uygulama için bir diğer önemli faydadır (bir uygulamada aynı işlevsellik ihtiyacı oluştuğunda daha önce oluşturulan modülün yeniden kullanılabilir olması modülasyonun önemli bir avantajıdır). Yazılan kodun beklendiği gibi çalıştığından emin olmak için testler yapılır. Özellikle, birim (bileşen olarak da bilinir) testi genellikle her parça fonksiyonun düzgün çalıştığını test etmek için kullanılır. Modüler tasarım sayesinde her seviyedeki bileşenlerin kolayca testi sağlanabilir.

Elbette modüler tasarımın bazı dezavantajları da vardır. En büyük dezavantajı iletişim yüküdür. Genel kaynak tüketim miktarı bir sistemin boyutuna ve heterojenliğine bağlıdır. Aynı etki alanı (domain) içindeki modüler uygulama ara yüzleri minimal ek yükler oluştururken, makineler ve teknolojilere yayılmış çok katmanlı (n-tier) uygulamalarda iletişim her zaman performans üzerinde büyük ve önemli bir etkiye sahiptir. Farklı teknolojilere dayalı bileşenler arasındaki iletişim genellikle standart mesajlar aracılığıyla gerçekleşir. Bu tür mesajların kullanımından önce taşınması, çevrilmesi ve yorumlanması gerekir ve bu süreç protokoller aracılığıyla gerçekleşir. Her protokol kendi içinde bir zaman anlaşmasını içerir. Bir diğer potansiyel dezavantaj *bağımlılık cehennemi* (dependency hell)'dir. Kendi bağımlılıklarıyla birlikte gelen çeşitli modüller ve paketler kullanılırken bağımlılık cehennemi oluşabilir. Bu paketler ve modüller ne kadar çok kullanılırsa bir projenin sürdürülebilirliği üzerinde zararlı etkileri olan çatışan bağımlılık potansiyeli o kadar fazla oluşabilir.

Modüler tasarımın tüm faydaları göz önüne alındığında iletişim yükü makul bir dezavantajdır. Farklı teknolojilerin birbirleriyle iletişim kurduğu durumlarda, her zaman bir çeşit API kullanılması gerekecektir. Bağımlılıklarla ilgili potansiyel problemler dikkatli bir planlama veya bağımlılıkların yönetimi için alternatif sistemler kullanılarak önlenebilir. Genel olarak modüler tasarım herhangi bir ölçekte kullanılması planlanan yazılım geliştirmesi için faydalar içerir, ancak modüler tasarımın en büyük avantajı çok katmanlı (multi layer) uygulamalarda görünmektedir.

1.1.2 Ölçeklenebilirlik

Kapsamlı bir web uygulaması/platformu geliştirirken, ölçeklenebilirliği her zaman göz önünde bulundurmak çok önemlidir (sistem karşılaşabileceği tüm

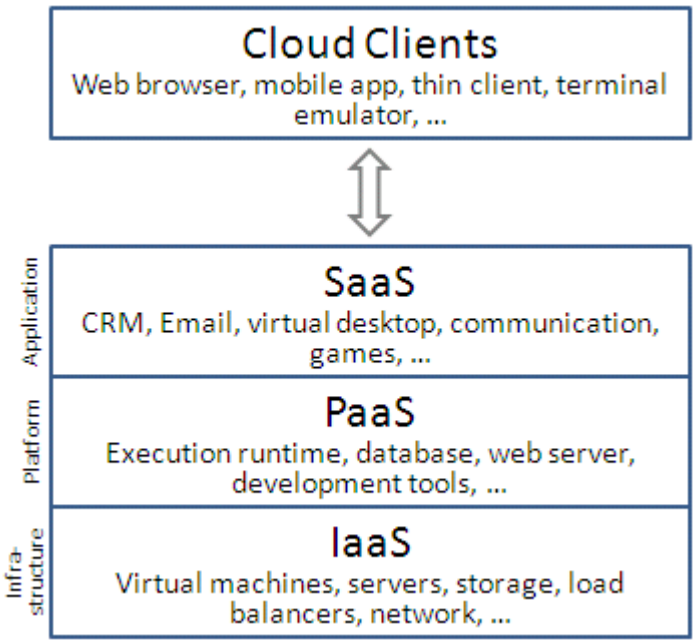
problemlere karşı fault-tolerant şekilde çalışmalıdır). Ölçeklenebilirliği kavramak için iki önemli kavramı örnekleyebiliriz: büyük veri ve bulut bilişim. Her iki kavramında yıllar boyunca ortaya çıkan eğilimlere uyacak şekilde değişen ve gelişen net bir tanımı yoktur. Büyük verinin orijinal tanımı 2001 yılında D. Laney tarafından yapılan bir araştırma raporuna kadar gider [14]. Laney Büyük veriyi 3V (hacim, hız ve çeşitlilik (volume, velocity, variety)) düzeyinde artış gösteren veriler olarak tanımlamaktadır. Boyutu ve kapsamı her geçen gün artan çok çeşitli veriler eşi görülmemiş bir hızda büyüyor (hız sadece veri oluşturma hızını değil aynı zamanda bu verilerin işlenmesi gereken hızı da ifade etmektedir). Bu tür (büyük) veriler, kendi zamanının geleneksel veri işleme prensipleri ile işlenemedi; verilerin işlenebilmesi için yeni yaklaşımlar ve fikirler gerekiyordu. Büyük verinin kabul edilen temel kavramı günümüzde hala yapılmak istenilen işlevlerin (örn: veriyi toplama, saklama, temizle, analiz etme vb.) *işlenemeyecek* kadar büyük ve karmaşıklığı oluşturmaktadır [15].

Bulut bilişim de aynı şekilde yeni ve içinde bazı belirsizlikler bulunan bir terimdir. "Demystifying Cloud Computing" [16] adlı makalede yazarlar 22 farklı bulut bilişim tanımını ortaya atıyor ve içinden bir tanesini bulut bilişimini tanımlamak için seçiyorlar. Bu tanım; "Bulut bilişim, büyük ölçüde ölçeklendirilebilen BT teknolojilerine ihtiyaç duyan kullanıcılar için hizmet tabanlı (as a service) bilgi işlem stilidir"[4]. Bu yaklaşım bulut bilişimin en doğru tanımıdır, ancak çoğunlukla altyapı, platformlar ve yazılımların bir hizmet olarak sunulduğu bulut teknolojisi olarak adlandırılır. Ayrıca bu tanımın içinde olmayan diğer bir seçenek ise kullanıcıların kendi bulut bilişim altyapılarını oluşturma opsiyonudur.

Bulut bilişim sağlayıcılarının ürünlerini sunduğu modeli *"Hizmet olarak X"* (XaaS) veya *hizmet olarak herhangi bir şey* olarak tanımlayabiliriz. İlgili hizmetler bulut sağlayıcıları tarafından hizmet/ücret karşılığında (veya bazen ücretsiz olarak) sunulur. Şekil 1.4, bulut bilişim hizmetlerin hiyerarşisini göstermektedir. Genellikle web tarayıcıları veya mobil uygulamalar ve aynı zamanda çeşitli diğer istemciler hizmeti alan aygıtlar veya yazılımlardır. Kullanıcıya hizmet aracılığıyla verilen işlevsellik veya denetim düzeyi, kullanıcının kullandığı hizmet düzeyine karşılık gelir. Aşağıdaki açıklamalar NIST tanımlarına dayanmaktadır [17].

[4] https://www.slideshare.net/innoforum09/gens

Hizmet Olarak Altyapı (IaaS) genellikle sanallaştırılmış temel bilgi işlem kaynakları (işlem gücü, depolama, ağlar, vb.) sunar. Kullanıcıya bileşenler üzerinde tam veya yüksek düzeyde denetim verilir, kullanıcı kendi sanal makinelerini oluşturabilir ve yönetebilir. İlgili denetim hiçbir zaman temel bulut altyapısı üzerinde olmayacaktır. Bir sonraki seviye, kullanıcıya uygulamalarını çalıştırabilecekleri kaynakları içeren bir platformun sağlandığı Hizmet Olarak Platform (PaaS) 'dur (Eğer uygulamaların yürütülmesi servis tarafından destekleniyorsa bu hizmet geçerli olacaktır). Böylece kullanıcı uygulama üzerinde denetime sahip olacaktır ve sistemin çalışma şeklini yönetebilir ancak aynı şekilde altyapının temel kurulum ve çalıştırılma ortamına erişimi yoktur. Kullanıcı genellikle uygulamanın dağıtımını ve yapılandırmasını yönetebilir. IaaS ile karşılaştırıldığında PaaS'in en büyük yararı, kullanıcıların altyapı planlaması, bakım, yazılım kurulum gibi hizmetleri yönetmek yerine efor, zaman ve odaklarını uygulamaya yoğunlaştırmasıdır. Bu sayede altyapı yönetim karmaşıklığın azalması hedeflenmektedir. Yazılım geliştiricileri, uygulamaları üzerinde ihtiyaç duydukları kontrol düzeyine bağlı olarak IaaS ve PaaS arasında seçim yaparlar. Durumun ilginç yönü PaaS sağlayıcıları genellikle kendi bulutlarını oluşturmazlar, hizmetlerini sunmak için IaaS kullanırlar.



Şekil 1.4: Örnek bir bulut bilişim modelinin hiyerarşisi [52].

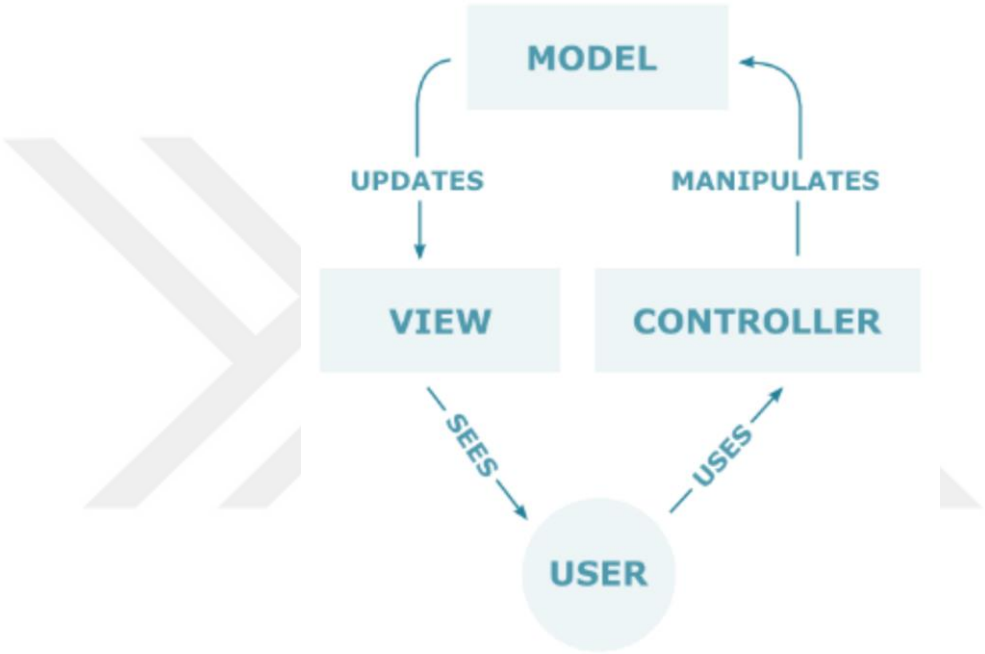
Hizmet Olarak Yazılım (SaaS), son kullanıcılara bulut altyapısında dağıtılan uygulamaların kullanımını sağlar. SaaS'e daha detaylı bakarsak yazılımın merkezi olarak barındırıldığı ve kullanıcılara istemciler aracılığıyla erişildiği bir yazılım dağıtım ve lisanslama modeli olarak tanımlanabilir. Geliştiriciler, binary dosyaları aracılığıyla düzenli yazılım dağıtımına kıyasla (FTP) SaaS'in sunduğu çok sayıda avantaj nedeniyle bu modeli kullanırlar. Bu avantajlara yazılım güncellemeleri üzerinde kontrol, kullanıcı verilerine ve davranışlarına erişim, platformlar arası cihaz uyumluluğunu denetlemek örnek olarak verilebilir. Son kullanıcılar da listelenen avantajların bazılarından yararlanır ama modelin dezavantajlarını incelediğimizde son kullanıcıların kendi verilerinin kontrolünü bırakmaları, özellikle potansiyel gizlilik ve güvenlik sorunlarıyla karşı karşıya kalmaları ve yazılımın tam bir kopyasının hiçbir zaman kendilerine ait olmaması olarak değerlendirebilir. Sunulan platformun güvenlik ve gizlilik yönlerini dikkate almak SaaS ürünlerini başarılı bir şekilde geliştirmenin temel anahtarlarından biridir.

Son olarak, ölçeklenebilirliğe değinmek gerekirse A. Bondi [18] ölçeklendirmeyi "Bir sistemin artan sayıda eleman veya nesneyi barındırma, büyüyen iş hacimlerini işleme ve/veya genişlemeye duyarlı olma" olarak tanımlamaktadır. Genel olarak, sisteme kaynak eklemenin iki yolu vardır: yatay veya dikey ölçeklendirme [19]. Bir sistemin çalışan fiziksel donanımın gücünü artırarak örneğin düğüme ek bellek veya CPU gücü ekleyerek dikey ölçeklendirilebilir. Yatay ölçeklendirme ek hesaplama düğümleri eklemek anlamına gelir. Sistemin kaynakları doğru şekilde kullanabilmesi için yatay ölçeklenebilirlik düşünülerek tasarlanması gerekir. Bölüm 1.1.1'de belirtilen modüler tasarıma bağlı kalınarak yatay ölçeklenebilir uygulamalar kodlanabilir.

Yukarıdaki tanımları incelediğimizde bulut bilişim, bir web platformunun uygun fiyatlı, esnek ölçeklenebilirliğini sağlamanın temel yoludur. IaaS ve PaaS, bir uygulama için basitçe sistemin ihtiyaç duyduğu ek fiziksel donanımları kolayca yönetmek ve yatay ölçeklendirmeye olanak verecek şekilde kullanılabilir. Ölçeklendirmenin en büyük faydası, çok katmanlı uygulamaları modellerken ve planlarken gelecekte karşılaşabilecek varsayımsal senaryolar için uygulanabilirlik dinamizmi olacaktır. Aynı ilke uygulamanın tüm bölümleri için geçerlidir.

1.1.3 View-Model-Controller (MVC)

Model-View-Controller (MVC) kullanıcı ara yüzlerini uygulamak için kullanılan mimari bir modeldir. Verinin uygulama içinde nasıl sunulduğunu (model), kullanıcının verileri nasıl gördüğünü (view) ve kullanıcının verilerle nasıl etkileştiğini (controller) ayırmanın bir yolu olarak G. Krasner ve S. Papa tarafından tanımlanmıştır [20]. Bu ayrıştırma, arayüz bileşenlerinin modülerliğini ve taşınabilirlik özelliğini artırır.



Şekil 1.5: Temel MVC diagramı [53].

Uygulamanın model kısmı, uygulama üzerinde hangi verilerin belirlenen bir durumu nasıl temsil edeceği ve değiştireceğini temsil eder. Görünüm (view), modelin kullanıcı ara yüzünde bilgi olarak çıktısıdır. Kontrolör (controller), kullanıcının girebileceği olası etkileşimleri tanımlar ve kullanıcı tarafından gönderilen girişleri komutlara dönüştürür. Kontrolör, modelde bazı değişiklikleri sağlamak için ön tanımlı komutlar gönderir. Modeldeki değişiklikler görünümün güncellemesine neden olur.

Aynı model, birden çok görünüme (view) ve kontrolör'e (controller) tanımlanmış olabilir. Örnek olarak varsayımsal bir çizim uygulamasını (hypothetical

drawing application) verebiliriz. Bu örnekte çizimin durumu ve olası manipülasyonları modelde tanımlanır. Çizimin görünümü tuvali temsil eder (tuvalin bir kısmının yakınlaştırılmış görünümü dahil). Kontrolör fare girişlerini (örn. daire çizme) model komutlarına dönüştürür ve çizim durumundaki değişiklikler görünüme yansıtılır. Mimarinin modüler yapısı nedeniyle girişler için kalem ve tablet cihaz desteğini sağlamak nispeten basit olacaktır (bu durumda sadece kontrolörün güncellenmesi gerekir).

MVC'nin esnek spesifikasyonu sayesinde çeşitli uygulamalar için farklı desenler üzerinde fikir bağımsız (Model-View-Adapter, Model-View-ViewModel, Model-View-Presenter, vb.) çalışılabilir. Bu tür kalıp desenlerin tüm sınıfına genel olarak MV * denir, çünkü çoğunlukla orijinal desenin kontrolör olarak neyi ifade ettiği ve üç parça arasındaki bağlantıları nasıl tanımladıkları farklılık gösterir.

1.1.4 Temsili Durum Transferi (RST)

Hizmet odaklı mimari (SOA), bileşenler arasındaki iletişimin, genellikle bir ağ üzerinden servis bazlı gerçekleştiği mimari modeldir (Sistemin bir bileşeni servisi sağlar ve diğeri sağlanan servisi kullanır). Temeldeki servis platformları çok çeşitli olabileceğinden, servislerin standart hale getirilmesi sağlayıcılar ile kullanıcılar arasındaki iletişim protokolüne bağlıdır. Konuyla ilgili birçok alternatif iletişim protokolü önerilmiştir, standart için en iyi çözüm Basit Nesne Erişim Protokolü (SOAP) olmuştur[5]. SOAP, mesajları saklar ve HTTP veya SMTP gibi yaygın aktarım protokolleriyle aktarır. SOAP temelde uygulamalar arasında genişleme için tasarlandı ve bunun yanında orijinal protokolün sorunlarına ve eksikliklerine çözüm olarak birçok ek standart ve protokol getirdi. Bu tür çözümlere örnek olarak uçtan uca güvenlik sağlamak için geliştirilen WS-Security ve işlemlerin atomikliğini sağlamak amacıyla geliştirilen WS-Atomic Transaction verilebilir. Web servislerle ilgili spesifikasyonlara sıklıkla WS- * terimi kullanılarak atıfta bulunulur.

SOAP, tüm WS-* kümesi ile hedeflenen basit bir servisi ortaya koyarken bile nispeten karmaşık ve kompleks olarak algılanmaktadır [21]. Alternatif bir çözüm olarak Temsili Durum Transferi (REST veya ReST) gösterebilir ve ilk olarak R. Fielding ve R. Taylor [2] tarafından tanımlanmıştır. Bu stile göre geliştirilen API'lere

[5] www.w3.org/TR/soap12

RESTful API'ler denir. Çoğu uygulama bir dizi kurala uysada, REST standart olmayan mimari bir stildir. REST, XML tipinde veri paketleyen SOAP ile karşılaştırıldığında, http protokolünün yeteneklerini ve kaynaklarını (GET, PUT, POST, DELETE gibi) kullanır. RESTful kaynak verileri hemen kullanılabilen bir forma dönüştürür ve çoğunlukla JSON tipiyle eşleşir.

Uygun bir çözüm seçmek çoğunlukla kullanım amacına bağlıdır. Geliştiriciler genellikle HTTP üzerinden çalışan ve kullanımı basit RESTful API'leri kullanmaktadır fakat çoklu platform ve iletişim gerektiren uygulamalarda ise gelişmiş özellikler gerektiren ve daha karmaşık çözümler sunan WS- * / SOAP tercih edilmektedir [21].

1.1.5 Nesne-İlişkisel Eşleme (ORM)

Sorgu dilleri genellikle uygulama ile veri tabanı arasında iletişim kurmak için kullanılır ve günümüzde en sık kullanılan sorgu dili yapılandırılmış sorgu (SQL) dilidir. SQL sorgusu verilerin getirilmesi gibi amaçlar için oluşturulur ve yürütülür. Döndürülen veri uygulama içi kullanılacak nesnelere dönüştürülür. Nesne ilişkisel eşleme (ORM) yazılımcıların nesnelerle çalışması için veriler üzerinde kaydetme, sorgu oluşturma, getirme ve nesnelere dönüştürmeyi otomatikleştiren işlemlerin soyutlamasıdır (oop abstraction). [23]

ORM soyutlama için en çok tercih edilen ve kullanılan framework'tür. Her seferinde yazılması gereken SQL kod miktarını önemli ölçüde azaltır ve programcıların nesne yönelimli programlama (OOP) çerçevesinde kalmasını sağlar. ORM sistemi temel olarak veri tabanıyla ilgili her şeyi nesnelerle eşleşen tabloların oluşturulmasından başlayarak işler. Ek olarak, bir veri tabanı çözümünün farklı bir veri tabanı çözümü ile değişimi her zaman göz önünde tutulmalıdır (örneğin MsSQL'den PostgreSQL'e) ve ilgili işlem uygulamanın kod tarafında çok büyük değişiklikler yapılmadan, sadece iletişimi işleyen adaptörün parametrelerinin değiştirilmesiyle sağlanıyor olmalıdır.

ORM çerçeveleri (frameworks) sık sık verimsizlik, bellek şişmesi ve kullanıcıyı şaşırtma (obfuskasyon)'dan eleştirilir ayrıca nesne yönelimli ve ilişkisel yaklaşımlar arasındaki boşluğu ORM çerçeveleri ile kapatmak da karşılaşılan zorluklar arasındadır (bu duruma *empedans uyumsuzluğu* denilmektedir). [24] Nesneler-veriler arasındaki dönüşümler her zaman ek yüke sebep olur ve bu yüzden oluşturulan sorgular verimsiz

olabilir. Nesne yönelimli dillerde programlama yapılırken veri tabanındaki veriler nihai olarak nesnelere dönüştürülür, bu durum programcının nesneler ve veri tabanı içindeki ilişkiler arasında her zaman eşleme yazması anlamına gelmektedir. Bu durum neden çözüm olarak sadece ORM çerçevelerinin kullanılmadığına cevap olacaktır. Çoğu ORM çerçevesi kritik performans kaygıları ve farklı fayda ihtiyaçları için elle yazılmış sorguları destekler. Nesne-ilişkisel eşleme (ORM) çeşitli modül / paketler / çerçevelerle yaygın olarak kullanılan tüm nesne yönelimli diller (OOPL) tarafından desteklenir ve sıkça SQL'den bahsedilse de NoSQL veri tabanları için ORM çözümleri de mevcuttur. [25]

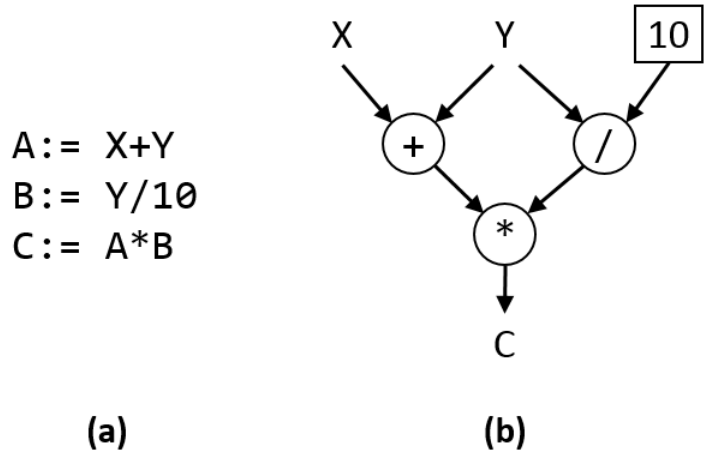
1.2. Veri Akışı Programlama

Veri akışı programlama, işlemleri temsil eden düğümlerle veri akışı arasındaki programı grafik yönlü modelleyen bir paradigmadır [5, 26]. Veri akışı, programlar hakkındaki geleneksel düşünceden uzaklaşarak von Neumann işlemcisi [6] tarafından tek tek yürütülen talimatlar dizisini temel fikir olarak alır. Yönlendirilmiş veri akışı grafiğinde bir işlem girdileri hazır olur olmaz yürütülebilir. Bu durum birbirinden bağımsız talimatların paralel yürütülmesini sağlar. Bu ayrı işleme modeli nedeniyle bilgisayar mimarisi gibi paradigmasını uygulamayı amaçlayan *veri akışı mimarisinden* ve yine yazılımla paradigma hedeflerine ulaşmaya çalışan *veri akışı programlamadan* (DFP) söz edilebilir.

1.2.1. Veri Akışı Yürütme Modeli

Veri akışı paradigması ile modellenen bir program aritmetik, karşılaştırma vb. işlemleri temsil eden düğümlerin olduğu yönlendirilmiş grafik (directed graph) ile temsil edilir. Aşağıdaki paragraf şekil 1.6'da sunulan en temel veri akışı programının açıklamasıdır. Grafiğin yönlendirilmiş okları düğümler arasındaki veri bağımlılıklarını ifade eder. *Girişler* düğüme (node) doğru akar ve *çıkışlar* da düğümden akar. Veri, düğümler arasında veri belirteçleri (tokens) biçiminde akar ve oklar belirteçler için ilk giren ilk çıkar kuyruğu işlevini görür. Programın başlangıcında, *aktivasyon düğümleri* (activation nodes) veri belirteçlerini anlık oklara yerleştirir ve ağ işlemeye başlar. Bir düğüm oklarına doğru belirtilen girdi belirteçleri setine sahip olduğunda *ateşlenebilir* olduğu anlamına gelir. Bu tür bir düğüm ateşlenebilir hale geldikten sonra tanımlanmamış bir zamanda yürütülür, girdi belirteçleri kuyruktan kalktıktan sonra

işlenir ve çıkış oklarının bir kısmına veya tümüne yeni veri belirteçleri yerleştirilir. Bundan süreçten sonra düğüm tekrar ateşlenene kadar bekler. Bu durum von Neumann yürütme modelinden en temel farkıdır (komut program sayacına ulaştığı durumda yürütülür, daha önce yürütülmüş olsa bile). Veri akışı işleme, sadece komut seviyesinde paralel yürütmeyi destekler. [27]



Şekil 1.6: Örnek program (a) veri akışı eşdeğeri (b) [5].

Büyük harfler değişkenleri ve kutudaki sayı sabit kodlanmış bir değeri temsil eder. Y değişkeninden çıkan iki ok, değişkenin değerinin bir kopyasını yani ortak kullanımını temsil eder, kopyalar ilgili düğümlerinde sona erer. (a) programının tamamlanması için üç zaman birimi gerekir, ancak veri akışı iki zaman birimde tamamlanabilir, çünkü ilk işlem seviyesi paralel olarak tamamlanabilir. Veri akışı düğümleri işlevseldir, yani giriş verilerini değiştirmezler ve diğer düğümler üzerinde etkisi yoktur.

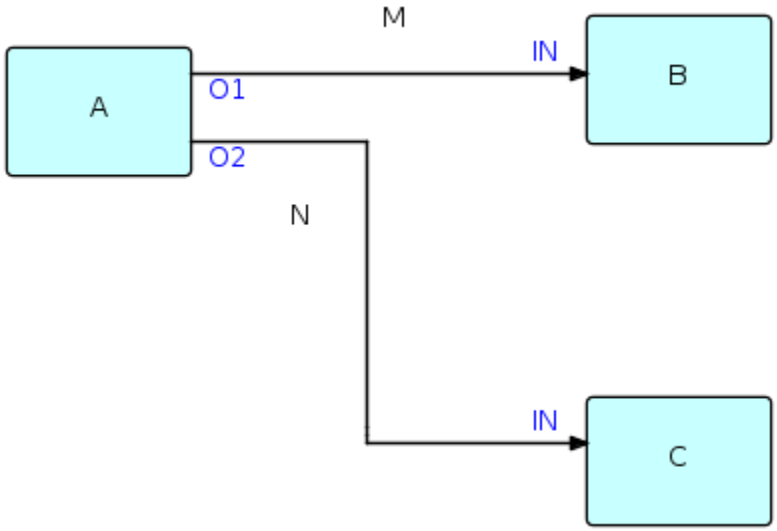
Şekil 1.6'da sunulan program veri akışı yürütme modelinin komut seviyesindeki en temel açıklamasıdır. 1990'larda araştırmacılar von Neumann veri akışı mimarilerinin birbirini dışlamadığını savunsalar da mimariler arasında iki uç nokta bulunmaktadır [28]. İnce taneli (fine-grained) veri akışı, her makine seviyesi komutun kendi iş parçacığında yürütüldüğü *çok iş parçacıklı* (multithreaded) bir mimari olarak görülüyordu. Algıdaki bu değişim, araştırmayı *hibrid veri akışına* doğru kaydırdı. Bu durum veri akışı yürütme modelini daha makro bir perspektife taşımıştır.

1.2.2. Akış-Tabanlı Programlama

Akışa dayalı programlama (FBP) 1970'lerin başında Morrison [1] tarafından icat edildi. Eskiden veri akışı programlama (DFP – Dataflow Programming) olarak betimlenen sistem veri akışı alanında yapılan araştırma ve veri akışı kapsamının daha geniş olması göz önünde tutularak FBP olarak yeniden adlandırıldı.

FBP bir uygulamayı eşzamansız (asenkron) süreçler ağı olarak görür ve bilgi paketleri (IP – Information Packets) adı verilen yapılandırılmış veri akışlarını alıp göndererek uygulama ile iletişim kurar. Bu durumun odak konusunu veri akışında olduğu gibi veri alışverişi ve ona uygulanan dönüşüm oluşturur. Ağ süreçlerden ve işlemlerden bağımsız ve birbiri ile bağlanan liste olarak tanımlanır. Bu bağlantılar Morrison'un *programlayıcı* (scheduler) olarak adlandırdığı bir yazılım parçası tarafından yorumlanır ve yürütülür. FBP'yi derinlemesine incelemeden önce hem FBP paradigmalarını hem de uygulama şekillerini Morrison'un tanımladığı biçimde anlamak önemlidir. Bu bağlamda tezin ana konusunu oluşturan OCC'nin temelini oluşturacak uygulama gerçekleştirimi için temel kavramların anlaşılması gerekmektedir.

Esasen, FBP bir uygulamanın geliştirilmesini iki bölüme ayırır- *ağ diyagramını tasarlama ve bileşenleri uygulama* (düğümler, süreçler vb.). Bileşenler, giriş ve çıkışlara sahip fonksiyonel kara kutular (black boxes) olarak kabul edilir ve ağ aralarındaki bağlantıları açıklar. Şekil 1.7, çok basit bir ağ diyagramına örnektir. A, B ve C kara kutu süreçleridir, M ve N bileşenler arasındaki bağlantıyı, O1, O2 ve 2 IN bağlantı noktalarını temsil eder. Morrison, B ve C kutularının aynı kodu yürütmesinin mümkün olduğunu ve bu nedenle her bileşenin kendi yerel depolama ve kontrol bloklarıyla diğerlerinden bağımsız olması gerektiğini belirtir.



Şekil 1.7: FBP diyagram örneği [6]

Bu nedenle paylaşılan bağlantı noktası adları yalnızca bileşen bağlamında birbiri ile ilgili olduğundan sorun oluşturmaz. Bağlantılar, sabit bilgi paketi (IP) kapasitesi olan *sınırlı tamponlar* (bounded buffers) olarak ifade edilir; bu durumda arabellek doluysa, işlem beslemesi durur ve arabellek boş olduğunda işlem askıya alınır. Bu bakış uygulama üzerinde bileşenlerin nasıl tanımlanacağına ve FBP'in bölüm 1.2.1'de açıklaması yapılan klasik veri akışı yürütme modeli (DFP) yaklaşımına benzerliğini vurgulamak için referans olacaktır.

FBP çeşitli programlama dillerinde uygulanabilir ve Morrison kişisel web sitesinde bazı örnek uygulamalar sunmuştur[7]. Uygulama programcıların üst düzey bileşenleri yazmalarına ve geleneksel programlama dilleriyle konvansiyonel olarak çalışmalarına olanak tanır. Morrison, *programlayıcının* (scheduler) düşük seviyeli bir dille yazılmış olmasına rağmen farklı programlama dillerinde yazılmış bileşenlerin tek bir ağda birlikte kullanılabileceğini belirtir.

FBP, modüler tasarımın özelliklerini nispeten düşük bir seviyede sergiler ve uygulamaların bileşenlerini oluşturmak için fonksiyonel kod parçalarını birbirine bağlar. Bileşenlerle uygulama geliştirmek doğal olarak zayıf bağlaşıma (loose coupling) yol açar, çünkü bileşenler yalnızca kendi aralarında veri alışverişi yapar.

[6] Şekil Morrison tarafından paylaşılmıştır. (https://jpaulm.github.io/fbp/index.html)
[7] https://jpaulm.github.io/fbp/examples.html

Ayrıca FBP bileşenin yeniden kullanımı, daha kolay bakım, güncelleme ve yatay ölçeklendirme için olanak sağlar. Morrison ayrıca gerçek süreç mantığı içeren bileşenlerle değiştirilebilen, FBP ile hızlı prototiplenmiş ve simüle edilmiş bileşenleri kullanmanın yararlı olacağını belirtmektedir. Bu simüle yöntemi günün sonunda varılmak istenilen uygulamanın her bir bileşenin / modülün işlevlerine makro bir genel bakış sunar. Bir araç olarak ağ şeması, tasarımcılardan geliştiricilere, kullanıcılardan yönetime kadar uygulamaya dahil olan herkesin arasındaki iletişimi kolaylaştırır.

1.2.3. Görsel Programlama Dilleri

Geçmişte, veri akışı programlama (DFP) diyagramları (şema/grafikte denilmektedir) sadece sistem veya konsepti anlamayı kolaylaştırmak amaçlı kullanılıyordu. V*eri akış diyagramı* konseptinin bahsi bölüm 1.1.1'de tartışılan yapılandırılmış tasarımın (structured design) yazarlarına [11] kadar uzanmaktadır. Diyagramlar sistemleri anlamak, muhakeme etmek, inşa etmek ve yürütmek için çok faydalıdır. Bu fayda farklı *görsel programlama dillerinin* (VPLs) icadı ile sonuçlandı ve süreç içinde kullanıcılar tarafından güçlü bir araç olarak benimsendi ve teşvik edildi. VPL'ler çok basit programlama yapılarından, tüm modüllerin bir sistem içinde birbirine bağlı olduğu değişik soyutlama seviyelerinde kullanılabilir.

Johnston vd. [5] görsel veri akışı programlama dillerinin referansı en kuvvetli tarihsel incelemesini sunmaktadır. İnceleme yazısı tekniğin son durumu hakkında bir değerlendirme ve hala geliştirilmesi gereken alanları içerse de genel yoğunlukta veri akışı yürütme paradigmalarına ve uygulamasına odaklanır. Makale farklı başlıkları içerse de tezin konusunda yer alan veri görsel programlama ortamları hakkındaki sonuçları birincil olarak incelenecektir.

Geliştiriciler tasarım aşamasından kodlama aşamasına geçiş yaparken bir paradigma değişimiyle karşı karşıya kalmaktadır [54] ve tasarım aşamasında veri akışı yaklaşımını kullanmaya eğilimlidirler ancak kodlamada bu yaklaşım genellikle zorunludur. Görsel programlama dilleri, tasarım ve kodlama aşamalarını tek bir süreçte harmanlar. Yukarıda belirtildiği gibi grafiksel gösterim birçok soyutlama düzeyinde geliştiricilere yardımcı olur. Baroth ve Hartsough [7], görsel programlama araçlarının kullanımı nedeniyle müşteri, geliştirici ve bilgisayar arasındaki gelişmiş iletişimin bir sonucu olarak verimlilikteki çarpıcı kazanımlardan bahseder. Başarı

düzeyi sadece VPL'e bağlı değil aynı zamanda kullanıldıkları geliştirme ortamına da bağlıdır. VPL'lerin grafiksel doğası nedeniyle çevre ve dil arasında ayrım yapılması zordur. Programların yürütmeden önce animasyonlu simülasyon sağlaması ve çok sayıda soyutlama kabiliyeti ile sorunsuz bir şekilde çalışan geliştirme ortamları, özellikle tasarım ve kodlama aşamasına büyük ölçüde fayda sağlayabilir. Görsel programlama dillerinin son olarak ve belki de en önemli faydası ilgili geliştirme ortamının programlama deneyimi çok az olan veya hiç olmayan kişiler için sezgisel olmasıdır.

Görsel veri akışı programlama ara yüzlerini, geliştirme ortamının bir parçası olarak kullanan birçok başarılı araç ve uygulama vardır. Örneğin MathWorks Simulink[8] dinamik sistemleri modellemek ve simüle etmek için grafiksel bir programlama ortamıdır, MIT Scratch[9] ücretsiz bir eğitim VPL'i ve LEGO Mindstorms[10] robotları kodlama amacıyla üretilen VPL'dir. Ayrıca, çocuklara programlama ve robotik kodlama öğreten birçok eğitim aracı da görsel programlama ara yüzlerini kullanmaktadır. Veri akışı arabirimlerinin en ideal kullanım senaryosuna son kullanıcının karmaşık işlemlere ihtiyaç duyduğu ancak uygulamak için programlama bilgisine sahip olmadığı durum örneklenebilir.

1.2.4. Tetikleyiciler

Veri akışı yürütme modeli kavramına değinildiği için tezde tetikleyicilere yer verilmiştir. Bir düğüm, tüm girişleri mevcut olduğunda ateşlenebilir hale gelir ve daha sonra belirsiz bir zamanda işlenir. Son girdinin iletilmesinden hemen sonra yürütmenin hemen gerçekleşmeyeceğine dikkat etmek önemlidir ve bu durum *etkinleştirilen* ve yürütülen bir görev arasındaki (tüm ön koşullar yerine getirildiğinde) temel farkı oluşturur [8]. Yürütmeye yol açan en önemli dış koşul *tetikleyicidir*. Düğüm tarafından iletilen talimatlar temel donanım-yazılım uygulaması ile işlenir ve veri akışı yürütme modeli bu talimatların iletilmesi ile oluşur (örneğin Morrison'ın programlayıcısı).

Tetikleyiciler iki şekilde anlaşılabilir [29];

[8] https://www.mathworks.com/products/simulink.html
[9] https://scratch.mit.edu/
[10] mindstorms.lego.com

a. *Fiil olarak,* bir olay bir eylemi tetikler (etkinliğin gerçekleşmesi eylemin gerçekleştirilmesine neden oluyorsa)

b. *İsim olarak*, tetikleyicinin çalışmasına neden olan bir nesne veya olay (object or event)'dır.

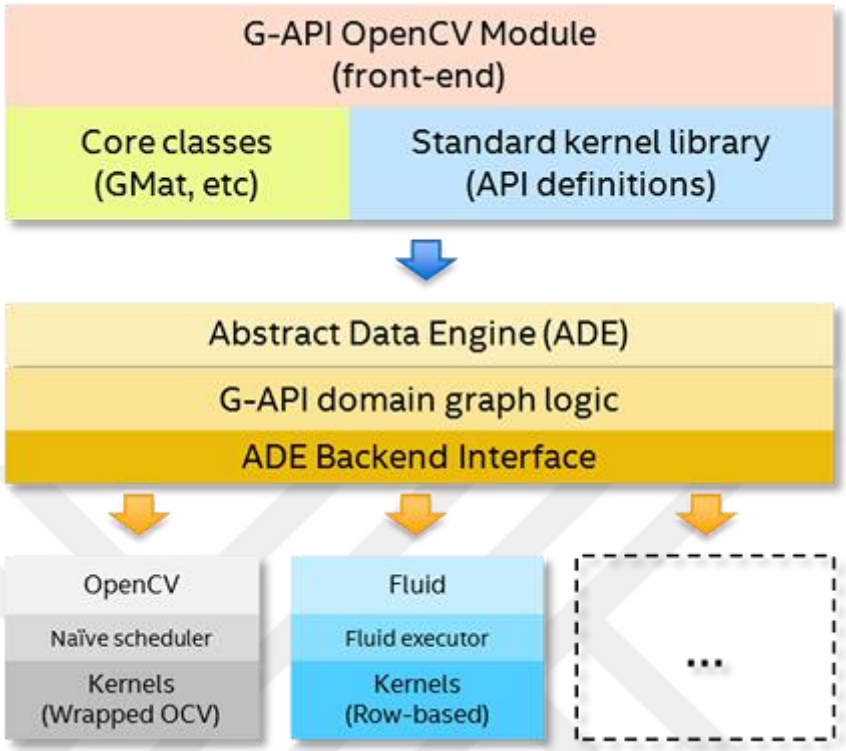
Tetikleyiciler çoğunlukla iş süreçlerinin ve faaliyetlerinin yönetimi ve lojistiğine bağlı iş akışı yönetimi kapsamında tartışılmaktadır. [29, 8]. Van der Aalst dört tip tetikleyici arasında ayrım yapar: otomatik, kullanıcı, mesaj ve zaman tetikleyicileri. Her biri, bir iş akışında bir görevin yürütülmesine yol açan harici bir nedeni temsil etmektedir. Tezin ana konusunu kapsamayan iş akışı yönetimi, görevlerin yerine getirilebileceği varsayımsal bir web platformu olarak ele alınacaktır. K*ullanıcı* olaylarını incelediğimizde etkin bir görevin yürütülmesi genellikle bir arabirim aracılığıyla bir insan tarafından olası bir eylem seçilerek tetiklenir. *Zaman ve mesaj* olayları bir tür istek (request) gelmesi üzerine periyodik olarak eylemsel tetiklenir. Son olarak, görev etkinleştirilir etkinleştirilmez bir görevin yürütülmesini tetikleyen *otomatik* tetikleyiciler vardır. Tezde açıklanan veri akışı yürütme modeli *otomatik tetiklemeli* bir sistem olarak düşünülebilir.

Platformu tasarlarken, otomatik işlemenin yapılabileceği bir sistem öngörülmüştür. İdeal olarak kullanıcı verileri yükler ve bağlantılı görevler otomatik olarak yürütülür. Ancak otomatik sistemin kullanıcının istediği akışta hizalanması/işlenmesi gerekliliği göz önünde tutulmalıdır (tetikleyiciler kullanıcının görevler için belirleyebileceği kurallar / filtreler kümesi olarak öngörülmelidir). Potansiyel giriş verileri önceden ayarlanmış kurallarda ve belirtilen koşulları karşılıyorsa *otomatik* olarak işlenir. Tüm iş akışı modeli bu şekilde uygun verilerin gelmesi üzerine otomatik olarak çalışacak şekilde ayarlanabilir.

1.3. OpenCV Görüntü İşleme Kütüphanesi

OpenCV [49] (Open Source Computer Vision Library), açık kaynaklı bir bilgisayar görme ve makine öğrenimi yazılım kütüphanesidir. OpenCV, bilgisayar görme uygulamaları için ortak bir altyapı sağlamak ve ticari ürünlerde makine algısının kullanımını hızlandırmak için inşa edilmiştir. BSD lisansı altında ücretsizdir. Kullanım alanı interaktif sanattan, mayın denetimine, haritaları birleştirmeden,

gelişmiş robotiğe kadar değişmektedir. 2.500'den fazla görüntü işleme, makine öğrenimi ve bilgisayar görme algoritması içermektedir.



Şekil 1.8: Opencv High-Level Tasarıma Genel Bakış[11].

OpenCV, 47 binden fazla kullanıcı topluluğu ve tahmini 18 milyon'u aşan indirme sayısına sahiptir. Kütüphane birçok köklü şirket tarafından kullanılmaktadır, bazı örnekler Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda ve Toyota'dır. OpenCV C ++ ile yazılmıştır ve STL[12] konteynırları ile sorunsuz çalışan bir ara yüze sahiptir. OpenCV, modüler bir yapıya sahiptir. Bu modülerlik kütüphanenin birkaç paylaşılan veya statik kitaplık içerdiği anlamına gelir. Aşağıdaki temel modüller mevcuttur [50]:

- **Çekirdek işlevsellik (core)** - yoğun çok boyutlu dizi Mat ve diğer tüm modüller tarafından kullanılan temel işlevler dahil olmak üzere temel veri yapılarını tanımlayan kompakt bir modüldür.

[11] https://docs.opencv.org/master/de/d4d/gapi_hld.html
[12] https://www.geeksforgeeks.org/the-c-standard-template-library-stl/

- **Görüntü İşleme (imgproc)** - doğrusal ve doğrusal olmayan görüntü filtreleme, geometrik görüntü dönüşümleri (yeniden boyutlandırma, perspektif çarpıtma, genel tablo tabanlı yeniden eşleme), renk alanı dönüştürme, histogramlar vb. İçeren bir görüntü işleme modülüdür.

- **Video Analizi (video)** - hareket tahmini, arka plan çıkarma ve nesne izleme algoritmaları içeren bir video analiz modülüdür.

- **Kamera Kalibrasyonu ve 3D Rekonstrüksiyon (calib3d)** - temel çoklu görüntü geometri algoritmaları, kamera kalibrasyonu, nesne poz tahmini, stereo mesajlaşma algoritmaları ve 3D rekonstrüksiyonun elemanları için kullanılan modüldür.

- **Nesne Algılama (objdetect)** - önceden tanımlanmış sınıfların nesnelerinin ve örneklerinin algılanması (örneğin yüzler, gözler, kupalar, insanlar, arabalar vb.) için kullanılan modüldür.

- **High-Level GUI (highgui)** - basit UI yeteneklerine göre kullanımı kolay bir arayüzdür.

- **Video I / O (videoio)** - video yakalama ve video codec bileşenleri için kullanımı kolay bir arayüzdür.

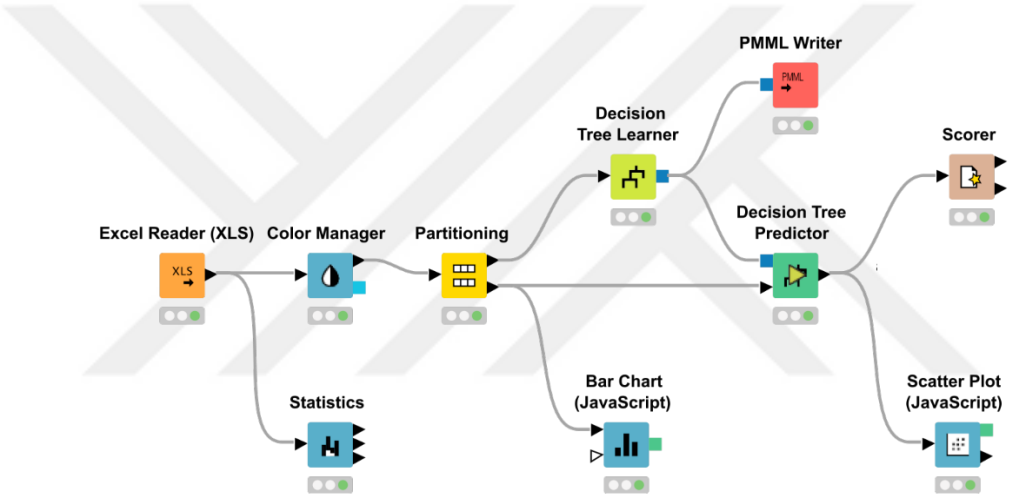
OpenCV Kütüphanesi çoklu işletim sistemini desteklemektedir (Windows, Linux, Mac OS). OpenCV çoğunlukla gerçek zamanlı görüntü uygulamalarına odaklanır ve MMX (Intel tarafından tasarlanan komut seti) ve SSE'den (Streaming SIMD Extensions) yararlanır. OpenCV, yürütülen işletim sistemi üzerinde Intel'in Entegre Performans İlkellerini (Integrated Performance Primitives) algılarsa hız ve performans için optimize edilmiş bahsi geçen rutinleri kullanır.

1.4. Benzer Çalışmalar

Bu bölümde tezin konusunu oluşturan platforma benzer veya ilgili mevcut yazılım çözümleri incelenecektir. Bu çözümlerin bazıları tezin konusunu oluşturan uygulama (OCC) için ilham kaynağı olarak sunulmuştur. Diğer örnekler veri akışı programlama (DFP) ve görsel programlama (VPL) olduğu için farklı çözümlerinin sunulması hedeflenmiştir.

1.4.1. Knime

Konstanz Information Miner (KNIME) [31] veri analizi için açık kaynaklı güçlü bir görsel programlama ara yüzüdür. Uygulama Java ile yazılmış ve çevrimiçi bir bileşeni yoktur. Bununla birlikte ücretsiz açık kaynak platformu için satın alınabilir eklentiler sunmaktadır. Bu ücretli eklentilerle platform merkezi yürütülen ve yönetilebilen kurumsal düzeyde bir çözüm haline gelebilir. Kurumsal kullanıcılar bir masaüstü istemciyle çalışabilir, oluşturdukları iş akışlarını uzaktan yürütebilir ve uygulama sonuçlarına web tarayıcılarından da erişilebilir. KNIME tipik anlamda bir web uygulaması değildir ve web istemcisi üzerinden tam işlevselliğe erişim yoktur (kurumsal sürümde SOAP tabanlı API bulunmasına rağmen).



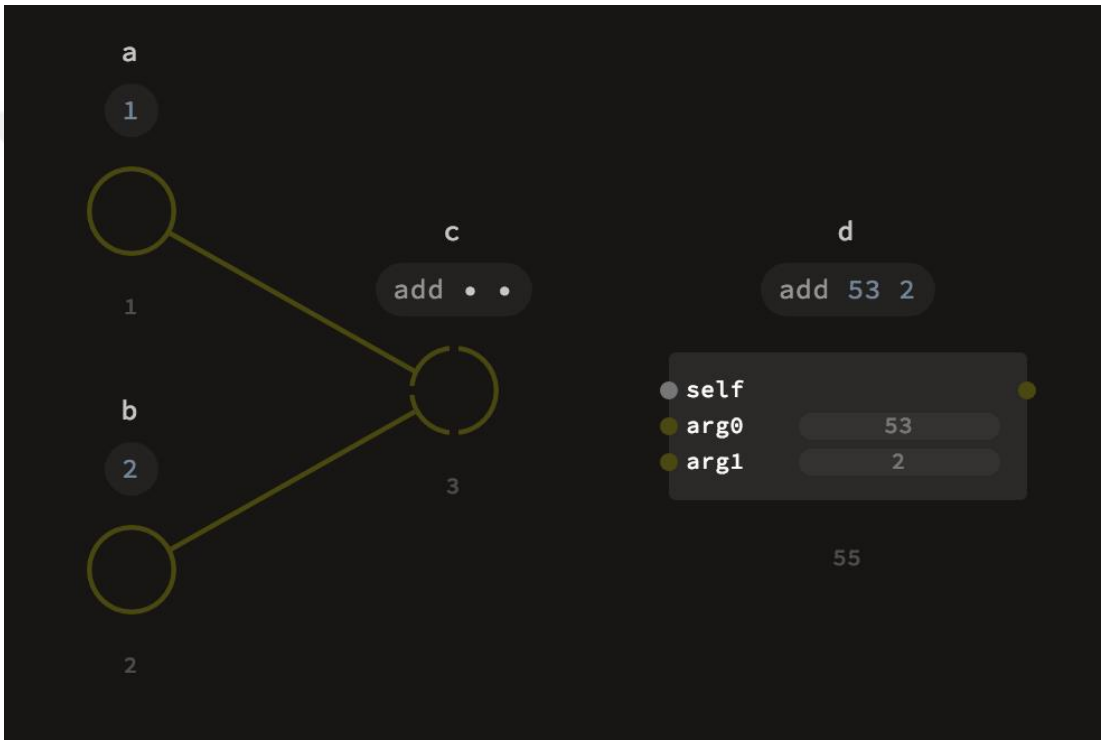
Şekil 1.9: KNIME platformundan bir akış örneği[13]

Orange'da (bakınız bölüm 1.4.7) olduğu gibi, ilgilenilen bileşen görsel ara yüzdür. Aynı zamanda veri akışı paradigmalarının iyi bir uygulamasıdır. Platformun dikkat çekilen özel bir ilgi alanı ise kullanıcılarını yürütme durumu hakkında bilgilendirmenin sezgisel bir yolu olarak bildirim semaforları kullanmasıdır (kırmızı hatayı, turuncu işleme hazır olduğunu ve yeşil işlemin bittiğini gösterir).

[13] https://www.knime.com/knime-analytics-platform

1.4.2. Luna

Luna [39] görsel ve metinsel, işlevsel bir programlama dilidir. Kullanıcıya hem kategori hem de objektif ve fonksiyonel odaklı programlamanın imkanını verir. Kullanıcı için hem metinsel hem de görsel bir geri bildirim sağlar. Kod görsel modda gerçek zamanlı olarak düzenlenebilir. Gerektiğinde küçük düzeltmelerin metin olarak yapılmasına izin verir. Sistem geliştirme iş birliğine odaklanır. Kodun bir beyaz tahta gibi temsil edilmesiyle programcılar dışındaki ekip üyeleri sürecin daha kolay bir parçası olabilir.



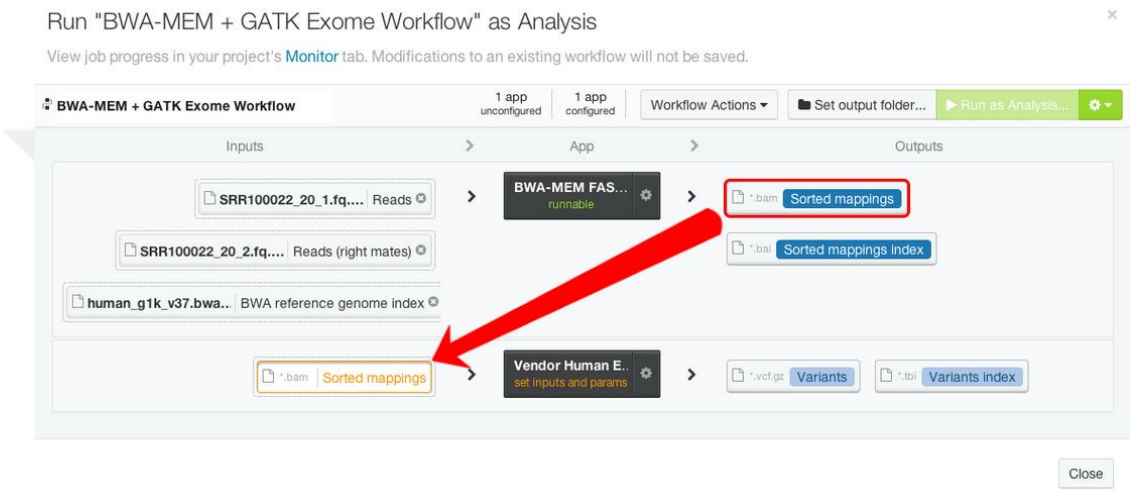
Şekil 1.10: Luna-Lang ile hazırlanmış örnek fonksiyonları çağırma ekranı[14].

1.4.3. DNANexus

DNANexus[15], biyoenformatik verilerin işlenmesi için bulut tabanlı platform sunan kendi alanına özgü bir çözümdür. Stanford merkezli bir start-up şirketinden türedi ve büyük girişim sermaye fonları aracılığıyla büyüdü. DNANexus'un rekabet avantajı katı güvenlik standartlarına, düzenlemelerine ve gizlilik yasalarına uymasından kaynaklanmaktadır. Amazon Web Services üzerine kurulu bulut tabanlı

[14] https://docs.luna-lang.org/luna-user-guide/calling_functions
[15] https://www.dnanexus.com/

bir çözüm olan DNANexus, mükemmel ölçeklenebilirlik, kullanıcı yönetimi, tekrarlanabilir sürüm kontrollü ardışık düzen ve ardışık düzenin (pipeline) tüm bölümlerine, hatta işlem kümelerindeki belirli düğümlere erişim kolaylığı sunar. DNANexus'un görsel veri akışı programlama ara yüzünde eksik olduğu şeylerden biri ardışık düzenin bir sırayla adımlar eklenerek örtük olmasıdır. Şekil 1.11'de daha yüksek basamağın çıktılarının alt kısımda girdi olarak kullanıldığı, iki aşamalı bir ardışık düzeni göstermektedir. Çıktıları girdilere bağlayarak ve uygulama kutucukları (siyah renkli) oluşturarak ek adımlar eklenebilir.

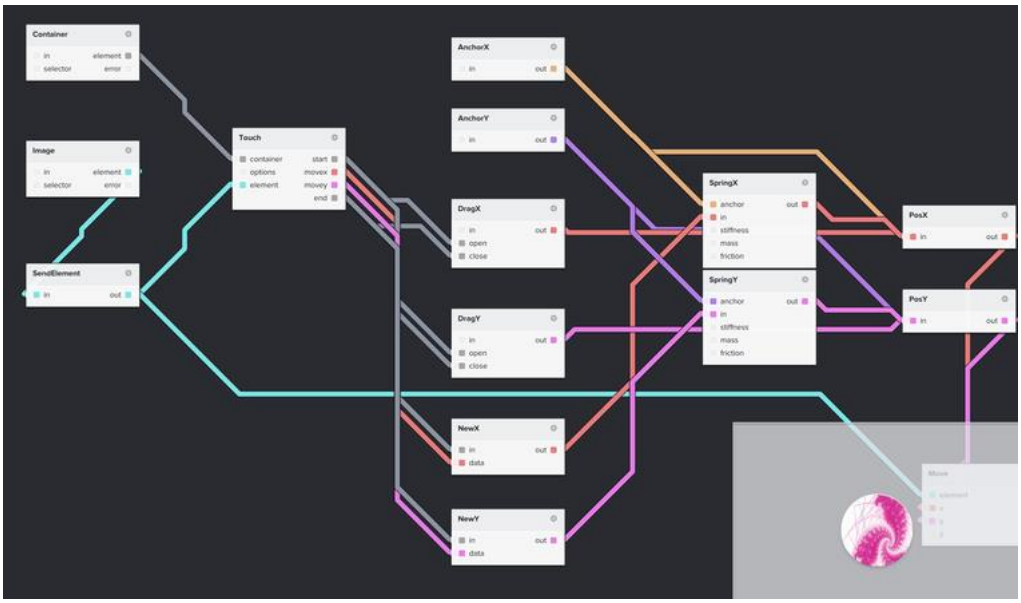


Şekil 1.11: DNANexus'ta bir pipeline işleme örneği[16].

1.4.4. NoFlo.JS

Noflo.js[17], Morrison'ın akış-tabanlı programlama (FBP)'sı üzerine geliştirilen JavaScript uygulamasıdır (bakınız bölüm 1.2.2). Proje, bir Kickstarter[18] kitle fonlaması kampanyasının ürünüdür ve FBP'ye dikkat çekmek için büyük çaba göstermiştir. Bu rağmen Morrison web sitesinde[19] uygulamanın Node.js üzerinde çalışmasına atıfta bulunarak "senkronize von Neumann paradigmasına bağlı" olduğunu belirtiyor ve gerçek FBP tabanlı olmadığını savunuyor.

[16] https://documentation.dnanexus.com/developer/workflows/building-and-running-workflows
[17] https://noflojs.org/
[18] https://www.kickstarter.com/projects/noflo/noflo-development-environment
[19] http://www.jpaulmorrison.com/fbp/noflo.html



Şekil 1.12: Noflo.js ile uygulanmış yay fiziği ile sürüklenebilir resim örneği[20].

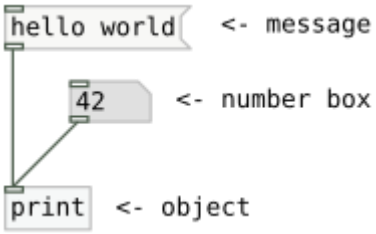
Morrison, FBP uygulaması oluşturmak için bileşen tabanlı bir yaklaşım ve bazı görsel temsillerle yapılandırılabilir modülerlik kullanımının yeterli olmadığını, yürütmenin de tamamen paralel olması gerektiğini belirtmektedir. Noflo'nun hedeflediği kullanıcıların geliştirici olduğunu belirtmek önemlidir (geliştiriciler ya kendi bileşenlerini yazarlar ya da uygulama oluşturmak için önceden yapılmış bileşenleri kullanırlar). Şekil 1.12'de noflo.js ile uygulanmış yay fiziği ile sürüklenebilir resim örneğinde her bir bileşen JavaScript'te (veya JavaScript'e çevrilen bir dilde) yazılır ve noflo ortamı içinde birbirine bağlanır. Baştaki düğümden (secondTick olarak adlandırılır) her saniye ağın geri kalanında işlemi tetikleyen bir paket gönderilir.

1.4.5. Pure Data

Pure Data (PD) [44] öncelikle animasyon ve müzik efektleri oluşturmak için kullanılan açık kaynaklı bir görsel programlama dilidir. Patcher programlama dillerinin önemli bir dalıdır ve gelen verileri değiştirmek için grafik benzeri bir görsel programlama stili kullanır. Programlama dili, nesneler, veri akışı, matematiksel işlemler ve üst düzey ses ve video manipülasyonu gibi birçok işlev ve özelliğe sahiptir. C veya C ++ tarafından derlenen PD-nesnelerini içe aktarma seçeneği de vardır. Pure

[20] https://noflojs.org/dataflow-noflo/demo/draggabilly.html

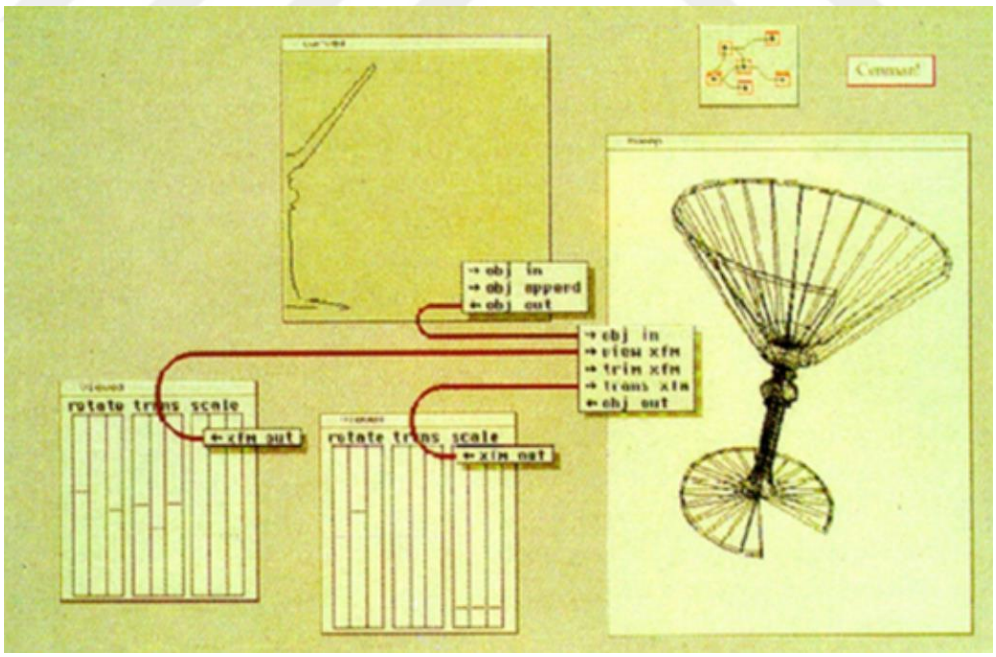
Data birçok işletim sistemi için destek vermektedir ve akıllı telefonlarda gömülü kütüphane olarak kullanılabilir[21].



Şekil 1.13: PD ile hazırlanmış basit bir uygulama[22].

1.4.6. ConMan

Bağlantı Yöneticisi, ConMan [43] interaktif programların oluşturulması, düzenlenmesi ve genişletilmesi için Haeberli tarafından öne sürülen bir uygulamadır. Küçük C parçacıklarını bağlamak için veri akışı diyagramlarını kullanılan üst düzey bir görsel programlama dilidir. ConMan ile oluşturulan uygulamalar görsel olarak temsil edilir ve kolayca düzenlenebilir ve değiştirilebilir. Daha sonraki yıllarda benzer amaçlarla çalışan ve kullanılan birçok farklı sisteme ilham kaynağı olmuştur.

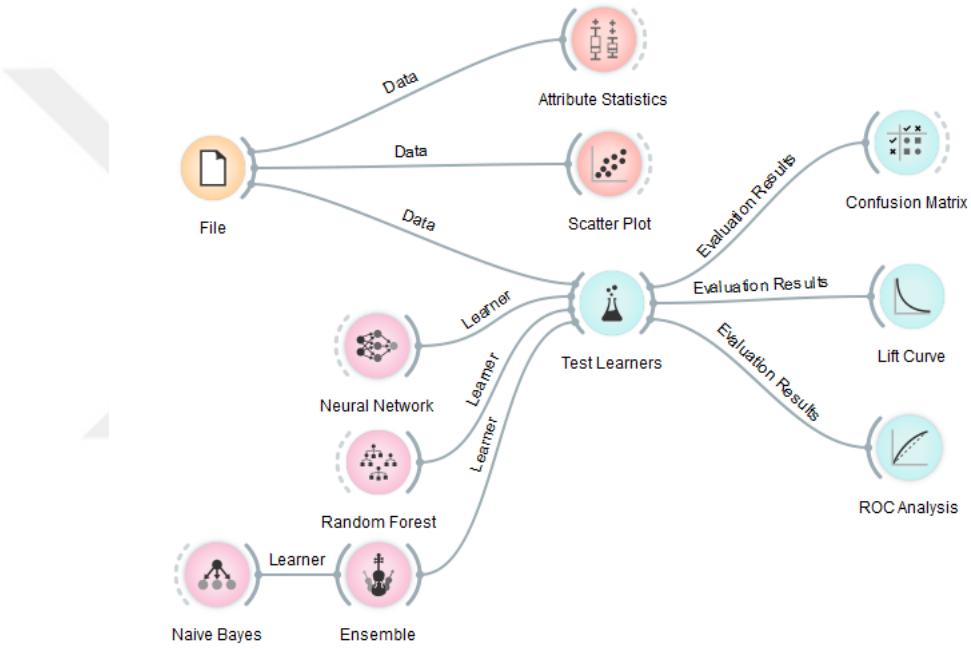


Şekil 1.14: Basit bir ConMan uygulamasına örnek [43].

[21] Libpd community site: https://puredata.info/downloads/libpd
[22] https://www.wikiwand.com/en/Pure_Data

1.4.7. Orange

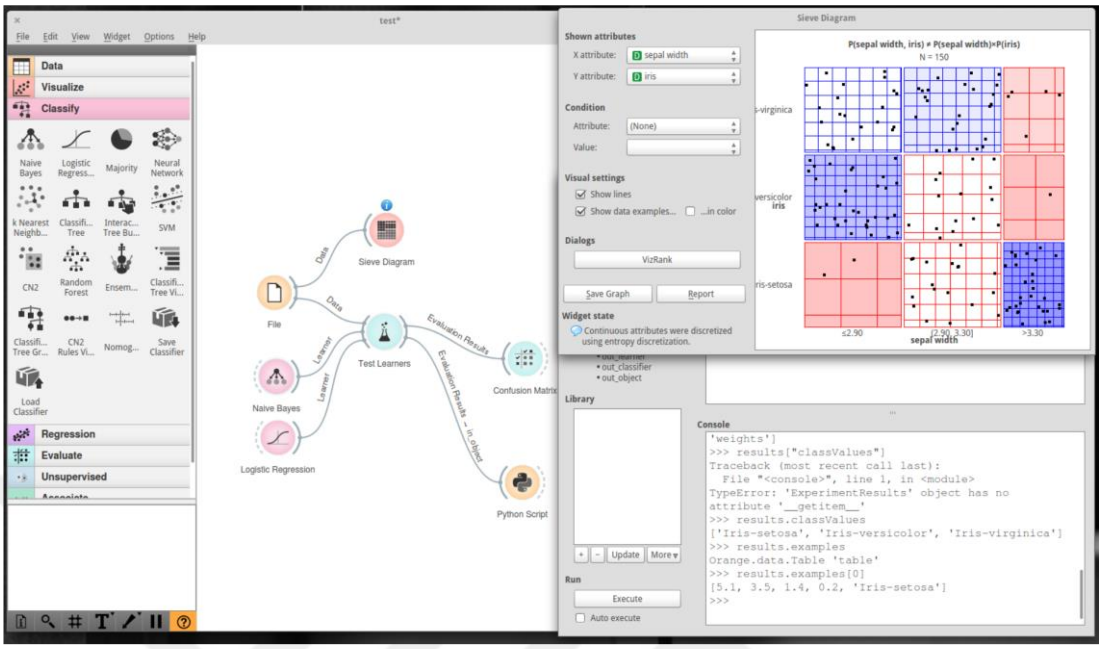
Orange [30] makine öğrenimi, veri madenciliği ve veri görselleştirme için kullanıma sunulan genel açık kaynaklı bir veri analiz yazılım paketidir. Orange, Python komut dosyası kitaplığının (python scripting library) yanı sıra kitaplıkla etkileşim kurmak için bir grafik arabirimi sunar. Görsel programlama ara yüzü oldukça ilgi çekicidir çünkü üst düzey görsel veri akışı programlamasının iyi bir uygulamasıdır. Orange Canvas ile oluşturulan bir veri akışı şeması örneği şekil 1.15'de gösterilmektedir.



Şekil 1.15: Orange Canvas ile oluşturulan bir veri akışı şeması örneği.

Basit sürükle ve bırak işlemiyle ara yüze işleme bileşenleri (widget'lar) eklenebilir. Mevcut widget'lar veya parametreler değiştiğinde veriler görselleştirme ve işleme widget'larına akar ve bileşenler otomatik olarak güncellenir. Orange'ın kullanıcı dostu ara yüzü ile tek bir kod satırı yazmaya gerek kalmadan nispeten karmaşık veri analizi iş akışları yürütülebilir (programcılar Python yazım dilini de kullanabilir). Şekil 1.16'da Orange ara yüzüne farklı bir örnek olarak tahmin sonuçlarının programsal olarak analiz edilebileceği açık bir python yorumlayıcı sunulmuştur. Orange, web uygulaması değildir ancak veri akışı paradigmalarını tam

olarak uygulamak ve mümkün olan en geniş kullanıcı tabanına ulaşmak için web platformu tarafından desteklemesi gereken görsel programlama arabirimidir.



Şekil 1.16: Orange'da sieve diyagramı ve tahmin sonuçlarını python yorumlayıcısında gösteren veri analizi iş akışı örneği.

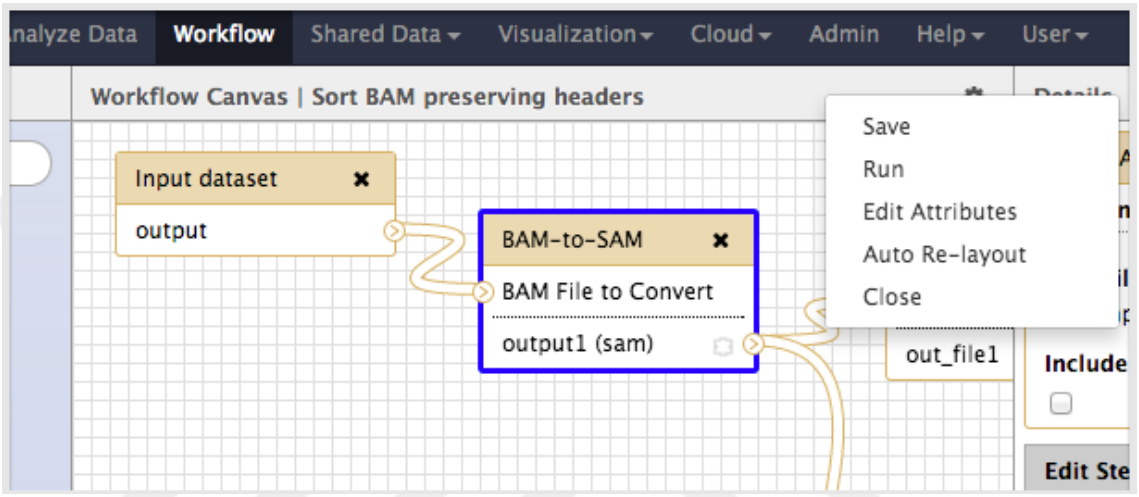
1.4.8. Galaxy

Galaxy [32, 33, 34] biyoinformatik alanında çok sayıda algoritmayı destekleyen açık kaynaklı ve web tabanlı bir veri işleme aracıdır. Görsel iş akışı oluşturucuları ile programlama deneyimi olmayan bilim insanlarına çeşitli hesaplama araçlarına erişim sağlar. Galaxy[23] web tabanlı olmasından dolayı genelde sorunlu olan kurulum sürecini gerektirmez ve tüm kullanıcıların aynı eşit ortamı kullanmasını sağlar. Ayrıca platform bilimsel yayıncılığın çok önemli bir parçası olan şeffaf ve tekrarlanabilir deneyler ve analizler sağlamayı amaçlamaktadır. Bu durum, tüm verilerin depolanması ve sunucu tarafındaki verilen işlenmesi ile gerçekleştirilir.

Analiz edilecek verilerin gizli ve kullanıcıya özel olması durumunda platformun açık kaynak lisansı kullanıcıların platformu kendi özel fiziksel donanımında kurmalarına imkân verir.

[23] https://usegalaxy.org/

Galaxy, alternatif olarak kullanıcıların kendi verilerini, kamusal verileri ve iş akışlarını kolayca işlemesi ve paylaşması için ücretsiz sunucular sağlar. Galaxy alanında çok güçlü bir oyuncu, ancak Genialis tarafından toplanan bilgilere göre Galaxy kullanıcıları platformun daha kullanıcı dostu olmasını istiyor. Bir diğer majör kullanım sorunu ise, Galaxy'nin büyük miktarda veri oluşturmasıdır. Çünkü Galaxy iş akışı ara yüzünde önceki adımların çıktılarını sonraki adımların girdilerine doğrudan aktaramaz; bunun yerine her adımda dosya oluşturur.



Şekil 1.17: Galaxy'nin iş akışı editörü örneği[24].

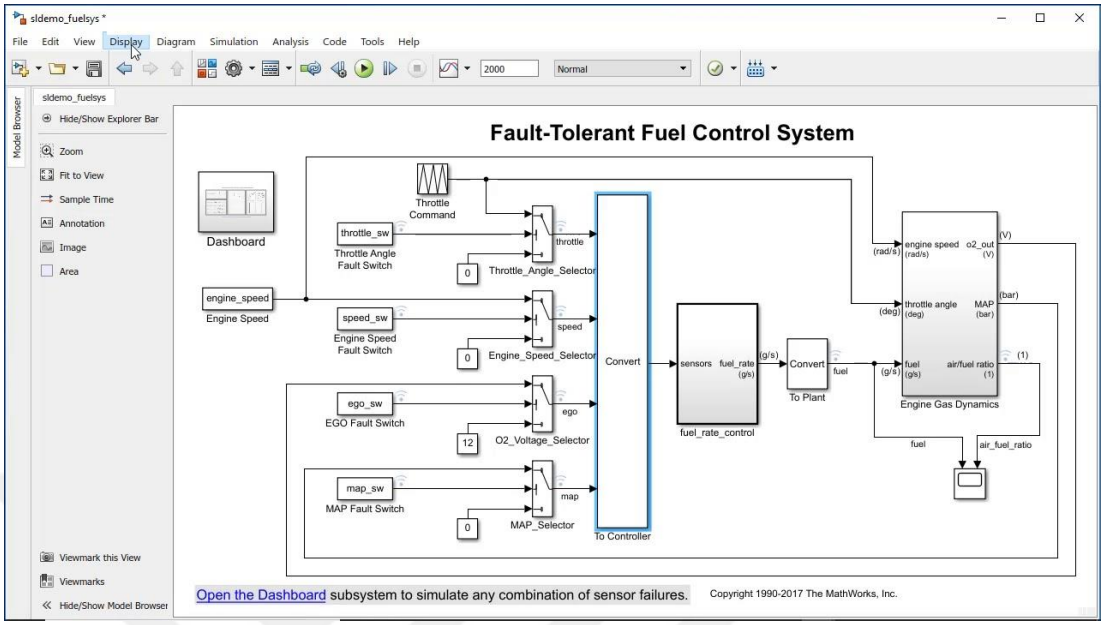
1.4.9. Simulink

MathWorks Simulink [35] temel olarak dinamik sistemleri modellemek, simüle etmek ve analiz etmek için kullanılan çok alanlı simülasyon ve model tabanlı sistem mühendisliğine odaklanan bir blok diyagram ortamıdır. Simulink, siber-fiziksel sistemlerin doğrulanması veya doğrulanması için sürekli testler ve simülasyonlar yapılmasını sağlayan bir çerçeve sağlar. Buna ek olarak, kod üretimi için de bazı opsiyonlar sağlamaktadır.

Simulink, çoğunlukla doğrusal veya doğrusal olmayan zaman odaklı sistemlerle çalışmak ve farklı davranışları modellemek için çeşitli blok kütüphaneleri, modelleme ve simülasyon sürecini destekleyici grafiksel bir editör sunar. Sunduğu işlevselliklere ek olarak, MATLAB ile entegre olduğundan, daha önce modellerde tanımlanmış olan

[24] https://galaxyproject.org/learn/advanced-workflow/

herhangi bir MATLAB algoritmasını veya fonksiyonunu birleştirme ve daha kapsamlı bir analiz için simülasyon sonuçlarını MATLAB'a aktarmayı da sağlar.



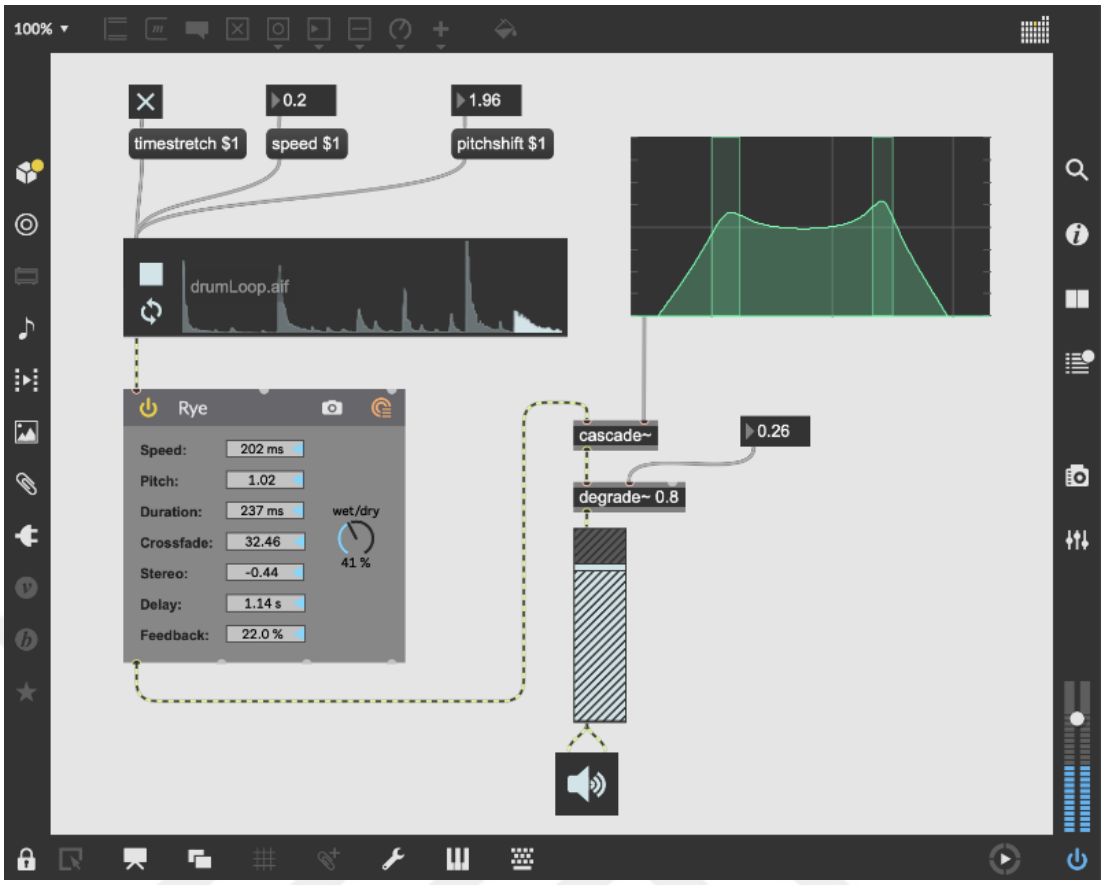
Şekil 1.18: Simulink ile oluşturulan "Sample Time" kullanımını gösteren örnek bir model ekranı [25]

1.4.10. Max 7

Max 7 [48], etkileşimli sesler, grafikler ve özel efektler oluşturmak amacıyla sanal ortam sunan bir sistemdir. Tıpkı Pure Data [44] gibi, yama tabanlı (patch) programlama ailesinden gelen görsel bir programlama dilidir, benzer şekilde birçok ilginç özelliğe de sahiptir. Max 7'deki farklı yamalar ses, grafik ve farklı bir kaynak kullanımı olabilir.

Max 7'yi farklı kılan en belirgin özelliği ise görsel yama ortamını, gerçek zamanlı kod oluşturma ile birleştiren *Gen* aracıdır. Ayrıca Max 7 kullanıcısı için programın hem görsel hem de metinsel versiyonunu aynı ara yüzde sunmaktadır.

[25] https://www.mathworks.com/videos/sample-time-legend-1519933428190.html

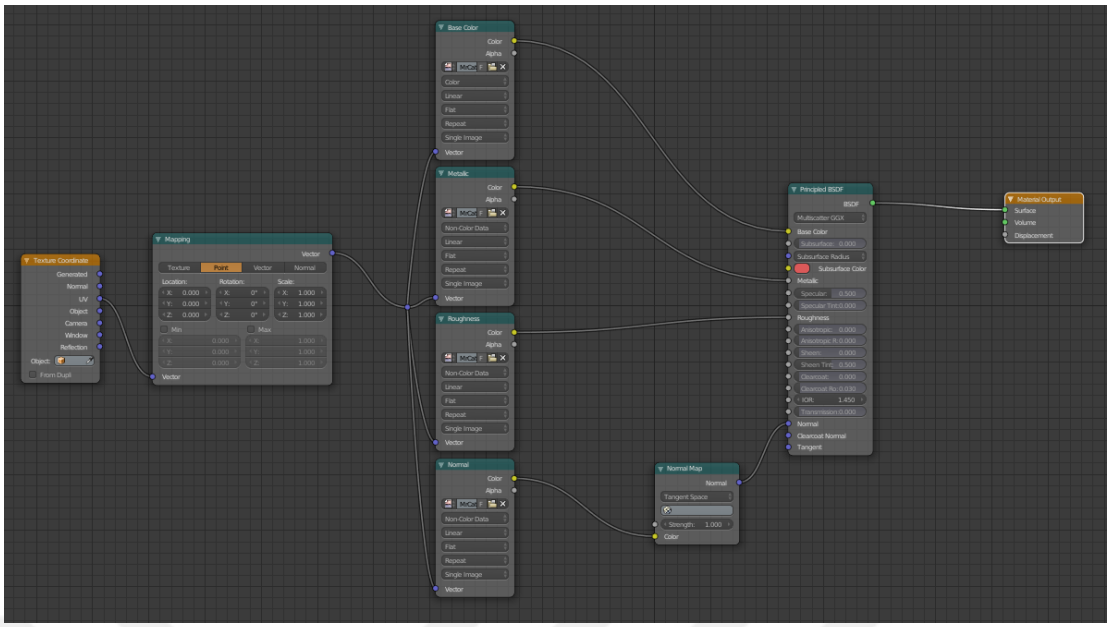


Şekil 1.19: Max 7 kullanıcı ara yüzü[26]

1.4.11. Blender

Blender [37], foto gerçekçi görüntüler, 3D animasyonlar, görsel efektler (VFX) ve video zenginleştirme amaçlı 3D görselleştirmeler oluşturmak için kullanılan ücretsiz ve açık kaynaklı yazılım paketidir. Blender, hemen hemen her türlü medya üretimini uygun hale getiren çok çeşitli araçlara sahiptir. Dünyanın dört bir yanındaki bireysel kullanıcılar ve stüdyolar Blender'ı hobi projeleri, reklamlar ve uzun metrajlı filmler için kullanıyor. Blender uçtan uca farklı işlevsellikleri kurgulamak, modellemek ve yönetmek için çekirdek özellik olarak akış diyagramı sunar.

[26] https://cycling74.com/products/max/

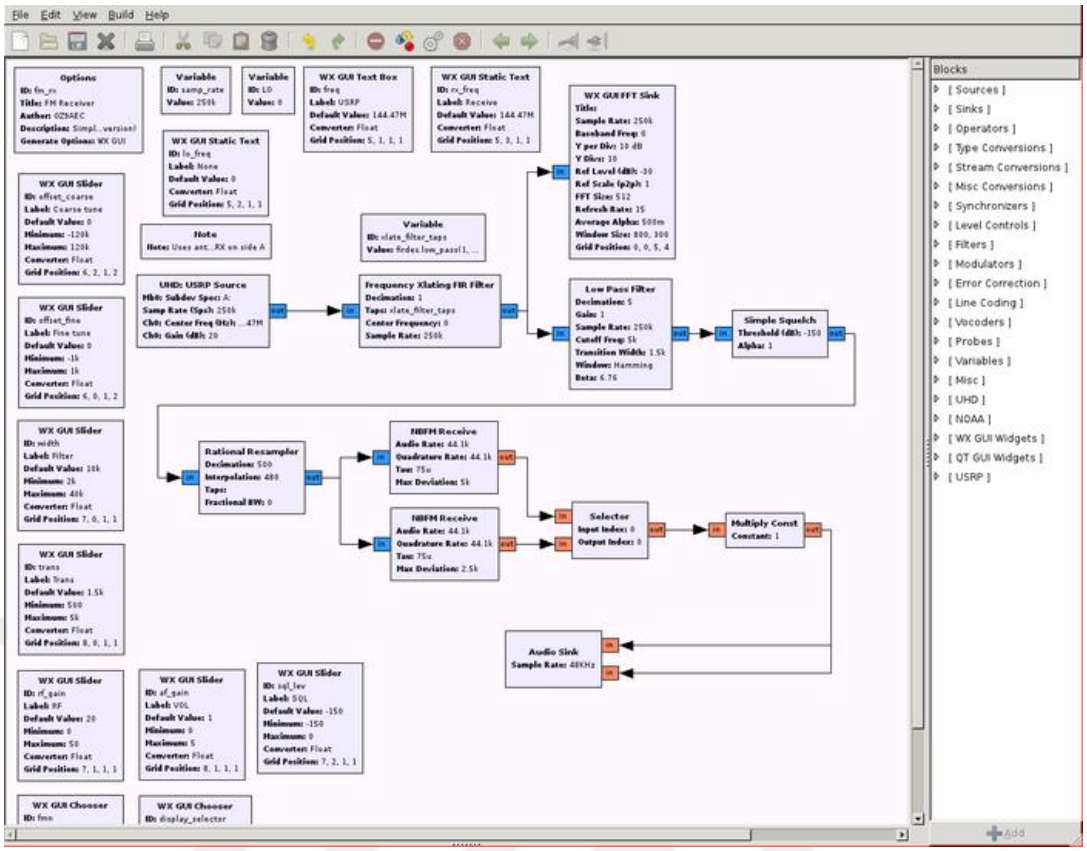


Şekil 1.20: Blender ile kurgulanmış görsel çıktı (render) örneği[27]

1.4.12. GNU Radio

GNU Radio [36], yazılım tabanlı radyoları uygulamak ve simüle etmek için kullanılan sinyal işleme blokları sağlayan ücretsiz ve açık kaynaklı bir yazılım geliştirme aracıdır. Yazılım tabanlı radyolar oluşturmak için kolayca temin edilebilen düşük maliyetli harici RF (Radio Frequency) donanımı ile veya simülasyon benzeri bir ortamda donanım olmadan kullanılabilir. Hem kablosuz iletişim hem de gerçek dünyadaki radyo sistemleri araştırmalarını desteklemek için hobi, akademik ve ticari amaçlı yaygın olarak kullanılmaktadır. GNU Radio Companion (GRC), sinyal akış grafikleri ve akış grafiği kaynak kodu oluşturmak için grafiksel bir araçtır[28].

[27] http://www.aendom.com/tuts/junkshop-workflow-en
[28] https://wiki.gnuradio.org/index.php/GNURadioCompanion



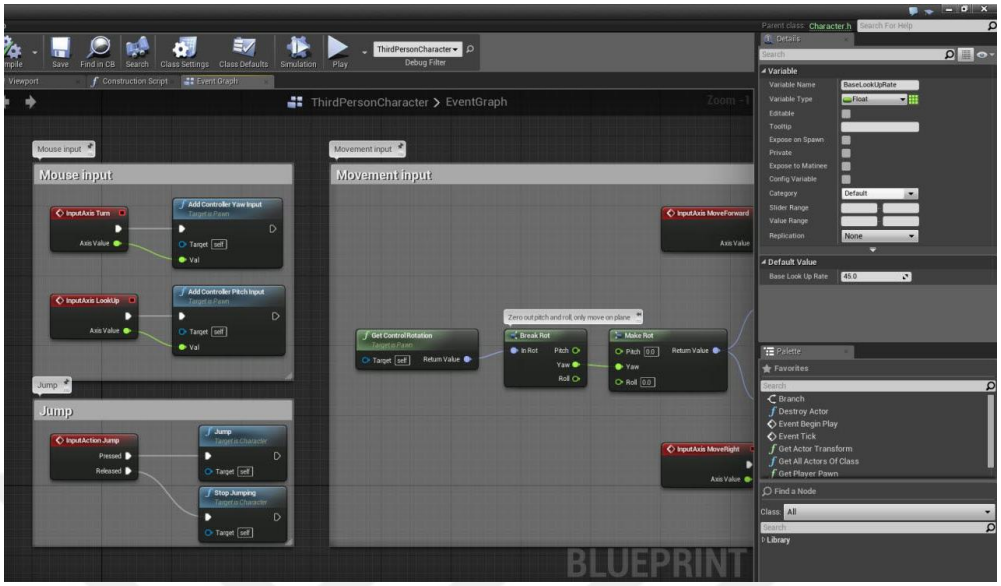
Şekil 1.21: GNU Radio Companion (GRC) ile kurgulanmış bir FM alıcısı örneği[29].

1.4.13. Unreal Engine: Blueprints

Blueprints [40], Unreal Engine'de kullanılan görsel bir betik (script) dilidir. Birçok seçenek ve işlevselliğe sahip karmaşık bir programlama ortamıdır. Realistik programlama yapmak için çok güçlü ve esnek araçları içerir. Genellikle oyun programlama için kullanılır, ancak farklı programlama ihtiyaçları için de kullanılmaktadır. Düğümler, events, diziler vb. öğeleri destekler ve düğümler diğer düğümlere bağlanabilir ve diziler herhangi bir tipte olabilir. Unreal Engine ile gelen güncelleme ile Blueprint'i C ++ 'a dönüştürme imkânı sağlar. Blueprint ile benzer amaçlar için kullanılan aynı segmentteki muhtemel rakiplerine FlowCanvas[30], Unity

[29] http://oz9aec.net/radios/gnu-radio/grc-examples
[30] "Flowcanvas, a modern visual scripting for unity." http://fowcanvas.paradoxnotion.com/

için Playmaker[31], Scirra tarafından üretilen Construct 2[32], Autodesk Stingray[33] örnek verebiliriz.

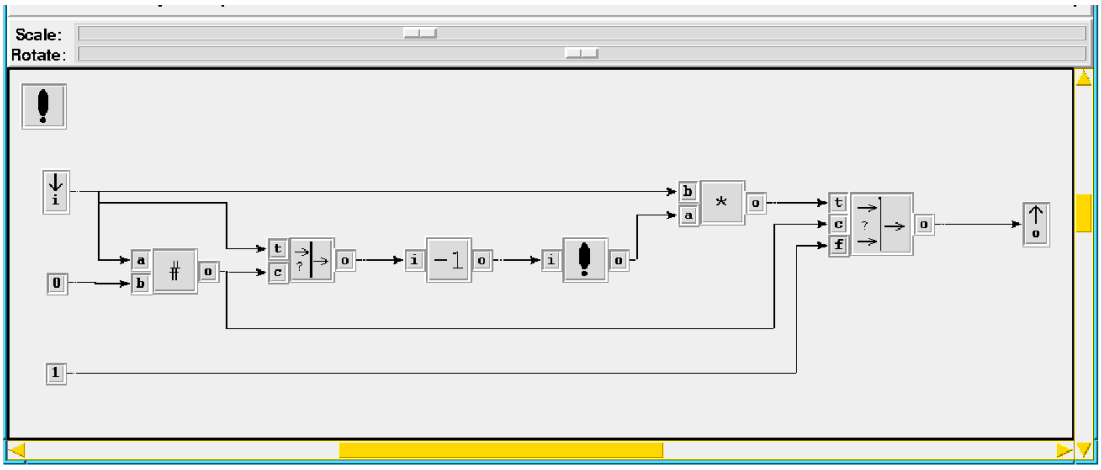


Şekil 1.22: Örnek Blueprint editör ekranı[34].

1.4.14. Backtracking DL

Geriye Dönük Veri Akışı Programlama Dili (Backtracking Data Flow Programming Language, BDL) [41], görsel programlamayı mantık tabanlı programlama ve geri takip ile birleştirerek ilginç bir yaklaşım benimsemektedir. Hem görsel dillerden hem de Prolog'dan özellikler kullanarak, iç çatışmaların (internal conflicts) çözülmesini destekleyen programlar üretmenin bir yolunu sağlar. Matematiksel operatörler, fonksiyon çağrıları, veri akışı, giriş / çıkış parametreleri ve koleksiyonları BDL'in kullandığı işlevselliklere örnek verilebilir. Oluşturulan programlar her zaman 1 veya 0 ile biter. Bu nedenle genel amaçlı programlama için kullanılamaz, mantıksal kavramlar için idealdir.

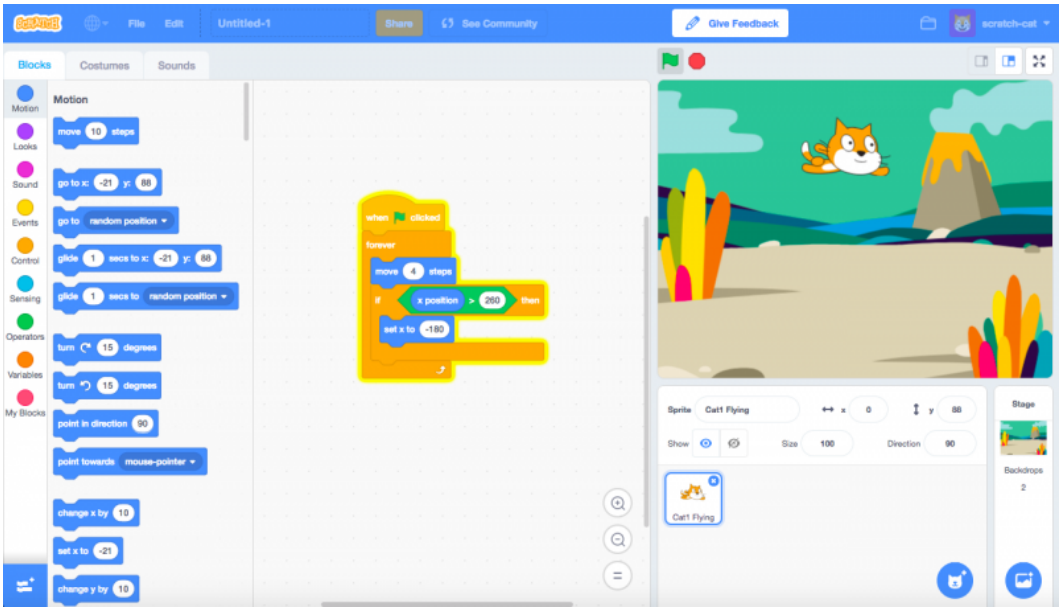
[31] Playmaker sayfası: http://www.hutonggames.com/
[32] https://www.scirra.com/
[33] https://knowledge.autodesk.com/support/stingray
[34] https://docs.unrealengine.com/en-US/Engine/Blueprints/Editor/index.html



Şekil 1.23: BDL ile tasarlanmış faktöriyel fonksiyon tanımı örneği.

1.4.15. Scratch

MIT Scratch [45] birçok yönden Blockly[35]'e benzeyen görsel bir programlama dilidir. Öncelikli hedef kitlesi çocuklar ve programlamaya yeni başlayanlardır. Uygulama, kod benzeri bir yapı oluşturan blokları birbirine kilitleyerek oluşturulur. Scratch yaygın olarak kullanılmakta ve 10 yılı aşkın süredir geliştirilmektedir. Scratch öncelikle animasyonları ve etkileşimli oyunları programlamak için kullanılır.



Şekil 1.24: Scratch editör ekranı[36]

[35] https://developers.google.com/blockly
[36] https://scratch.mit.edu/projects/editor/?tutorial=getStarted

1.4.16. Kodu

Kodu[37] uygulamayı modellemek için blokların birbirine kilitlendiği görsel bir programlama dilidir. Genel olarak Xbox konsolu için oyun programlamaya yeni başlayanlar ve çocuklar için hedeflenmiştir. Program basit olmayı amaçlamaktadır ve tamamen ikon tabanlıdır.

Fiziksel çarpışma efektleri, renk ve vizyon, arazi oluşturucu gibi gerçek dünya ilkellerini içeren güçlü bir üst düzey dildir.

Şekil 1.25: Kodu uygulama ekranı.

1.4.17. Viola

Viola [42] ekolojik sorunların modellenmesini amaçlayan görsel bir programlama dilidir. Sistem, karmaşık ekolojik simülasyonları kolayca geliştirebilmek için çok fazla programlama geçmişine sahip olmayan araştırmacılar için tasarlanmıştır. Dilin temeli heterojen peyzajlarda davranışsal olarak karmaşık hayvanların simülasyonlarını oluşturmak için önceden geliştirilmiş Biola adlı bir metin dili oluşturmaktadır.

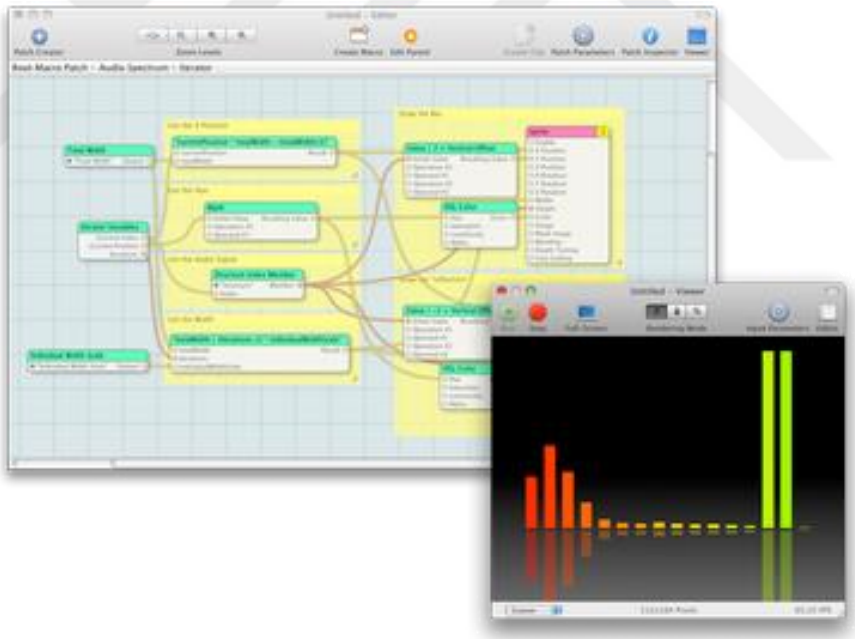
[37] https://www.microsoft.com/en-us/research/project/kodu/

1.4.18. Quartz Composer

Quartz Composer [47] kompozisyonlar adı verilen grafik işleme modülleri oluşturmak için kullanılan çok güçlü bir görsel programlama aracıdır. Temel amacı prosedürel hareket grafik programları olan kuvars kompozisyonları oluşturmaktır.

Quartz Composer yaygın olarak grafik efektler ve benzeri uygulamaları üretmek için kullanılır ancak birkaç eklenti ile platform son derece güçlü hale getirilebilir (örneğin yapay sinir ağı oluşturmak gibi) [46].

Bu aracın tasarım özelliklerinden biri, kodun katmanlar halinde oluşturulmuş olmasıdır. Bir blok kendi içinde çok sayıda blok tutabilir. Bu sayede Quartz Composer ara yüzü üzerinde farklı yamaları (patches) yönetirken daha net ve kullanıcı dostu bir bakış sağlar.

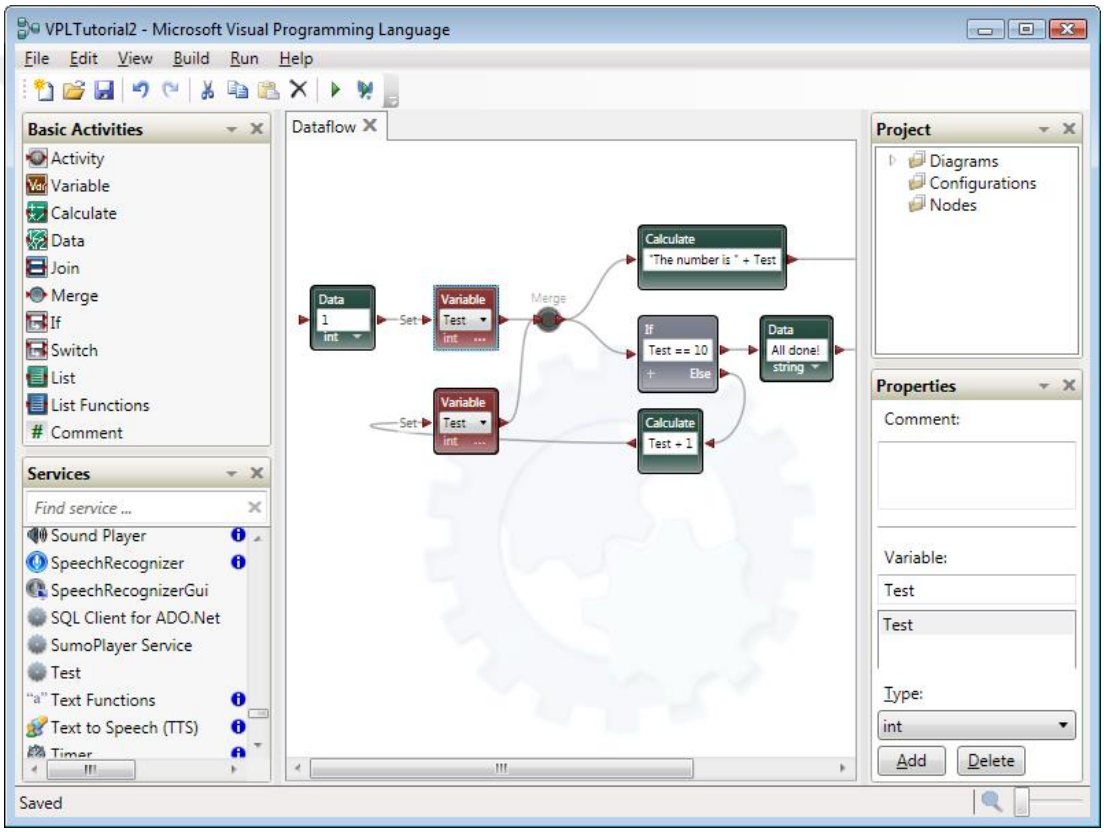


Şekil 1.26: Quartz Composer 3 Arayüzü[38].

[38] https://www.wikiwand.com/en/Quartz_Composer

1.4.19. Microsoft VPL

Microsoft görsel programlama dili (VPL) [38] uygulama geliştirme için kullanılan bir sistemdir. Grafiksel veri akış modeline dayanır ve zorunlu komutlar yerine, işlevsel bir dil gibi mesajlar ileterek etkileşen bir dizi aktivite yönetimli çalışır. Sistemin üretilen çıktısı, bağlı süreçler arasında bir veri akışı düzenlemesi veya bilginin koordinasyonu olarak adlandırılabilir. Bu dil gerçekten kullanışlı uygulamalar oluşturmak için kullanılabilir. Değişkenler, fonksiyonlar, listeler, şart ifadeleri vb. gibi geleneksel bir programlama dilinde beklenilen her türlü işlevselliği ve konsepti destekler.



Şekil 1.27: Örnek Microsoft VPL diyagramı[39]

[39] https://docs.microsoft.com/en-us/previous-versions/microsoft-robotics/bb483088(v=msdn.10)

İKİNCİ BÖLÜM

2. METOT

Bu Bölüm, OCC (OpenCV Companion) geliştirme amaç ve hedeflerine genel bir bakışla başlayıp, platform mimarisi ve uygulanmasını açıklamaktadır. Uygulamada kullanılan teknoloji çözümleri ve seçimlerinin mantıksal sebepleri de incelenmektedir. Ayrıca, elde edilen uygulamanın bazı seçilmiş bölümleri için uygulama ayrıntıları hakkında detay verilmiştir.

2.1. Uygulamanın Amacı ve Hedefleri

Daha önce belirtildiği gibi, bu tez projesinin amacı opencv kütüphanesinin fonksiyonlarını akış-tabanlı programlama arayüzü ile simüle etmek için genel amaçlı bir web platformu oluşturmaktır. OpenCV kütüphanesinin en çok Python programlama dili[40] üzerinde kullanıldığı varsayılırsa geliştirilecek uygulamanın python kodu çıktısı sağlaması sayesinde kodlama zamanından ciddi oranda tasarruf sağlanması hedeflenmektedir. Python kodlaması ile uygulanmak istenilen çalışmanın ilerleyiş durumu anlık olarak öngörülemeyecektir. Kullanıcının, geliştirilecek OCC uygulamasının akış tabanlı programlama ara yüzünde çalışmasının anlık ilerleyişi ve sonuç çıktısını simüle edebilir olması uygulamanın amacına yüksek oranda katma değer sağlayacaktır. Uygulamanın odak noktası ölçeklenebilirlik, genişletilebilirlikve kullanım kolaylığı olarak ifade edilebilir. Kullanıcılar OpenCV [49] kütüphanesinin sunduğu her türlü işlevi akış tabanlı editör üzerinden kurgulayabilmeli ve platformda bulunan ön tanımlı araçlarla işleyebilmelidir. Bu araçlar veya uygulama parçacıkları yeterli programlama becerisine sahip herkes tarafından geliştirebilir biçimde ve MVC standartlarına uygun olmalıdır. Bu uygulamalar, yalnızca arka planda komut satırı işlevlerini çağırabilen basit işlemlerden, karmaşık görselleştirmelere kadar değişebilir. Kullanıcıların diğer kullanıcıların süreçlerini ve verilerini etkilemeden uygulamalarını özel olarak çalıştırabilecekleri bir sistemine ihtiyaçları vardır. Bu yüzden kullanıcılara çok fazla geliştirme özgürlüğü sağlamak ve tüm uygulamanın anlaşılır ve saf kod mantığı ile geliştirilmesi öncelikli hedef olmuştur. Ek olarak, daha az programlama yetkinliğine sahip ve kendi uygulamalarını geliştiremeyen kullanıcılar, OpenCV

[40] https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html

fonksiyonlarını görsel programlama dillerine dayanan bir grafik ara yüzü ile işlemeyi kolay bulmalıdır [50].

Devam etmeden önce, aşağıdaki bölümlerde sıklıkla görünecek ve platformu anlamanın temelini oluşturan birkaç terimi tanımlamak faydalı olacaktır.

Veri (ayrıca veri girişi) bir bilgi parçasıdır. Platform her türlü dosyayı (resim, video, metin vb.) kabul edebilir olmalı. Veriler, yeni veriler oluşturmak için bloklara girdi olarak kullanılabilir. Veri girişi, belirli veri türünün nasıl tanımlandığına bağlı olarak tek bir dosya veya birden fazla dosya formatını içerebilir.

İşlemci, veri akışı programlama ile çok yakından ilgili bir kavramdır. Verileri girdi olarak kabul edebilen, verileri işleyebilen ve çıktı verileri oluşturabilen tek bir iş akışı düğümüdür. Bir süreç her zaman yeni veriler üretir ve üretilen çıktı verileri işlemci ile yakından ilişkilidir. Tüm işlemciler verileri girdi olarak almaz; örneğin, yükleme / içe aktarma işlemcileri, diğer veri girişleri yerine ham dosyalardan veri girişleri oluşturur.

Uygulama, verileri akış tabanlı programlama arayüzü ile işlemenin temel yoludur ve tüm veri girişleri en az bir projeye aittir. Böyle bir temel organizasyon birimine ihtiyaç duyulmasının sebebi tek bir kullanıcı bile büyük olasılıkla benzersiz kurgu ve görevler üzerinde çalışacaktır.

2.2. Uygulama Süreci

Y. Bassil'e [51] göre yazılım geliştirme yaşam döngüsü veya kısaca SDLC, bilgi ve endüstriyel sistemlerin tasarımı, inşası ve bakımı için bir metodolojidir. Bir yazılım çözümü geliştirmek için sırayla tamamlanacak beş faz içeren Şelale modeli olan birçok SDLC modeli vardır.

Analiz: Uygulama varsayımları göz önüne alınarak kullanıcıya yönelik faydaları değerlendirilir. Temel ihtiyaçlar belirlenir, uygulama için fizibilite çalışmaları yapılır ve uygulama planlaması gerçekleştirilir.

Tasarım: Mimari tasarım, UI (kullanıcı arayüzü), UX (kullanıcı deneyimi) tasarım konsept dokümanları oluşturulur. Gerçekleştirim aşamasından önce ihtiyaç duyulan modüller görselleştirilir ve kullanışlılık olarak kolaylığı analiz edilir.

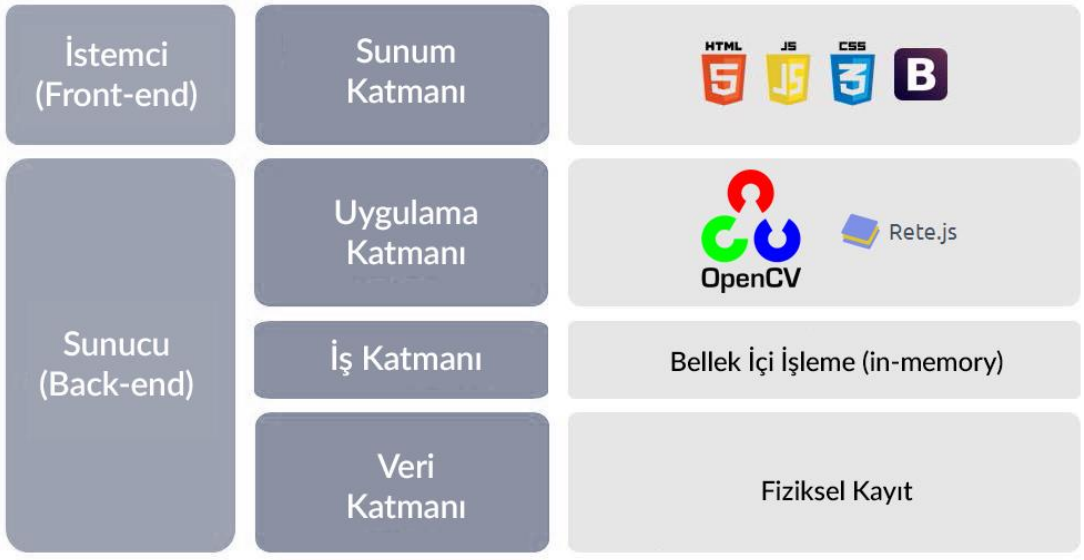
Gerçekleştirim (Kodlama): Tasarım aşamasının belirli bir olgunluğa ulaşmasıyla birlikte kodlama (yazılım) aşaması başlar.

Test: Birim testleri, stres testleri, yanlış değer testleri, kabul testleri, kullanım senaryo testleri, yük testleri, kullanıcı kabul testi, test otomasyonu gibi sürece ve duruma göre uygulanabilecek çok farklı kategoride ve derinlikte test türlerinin uygulanması sağlanır.

Bakım: Uygulama devreye alındıktan sonra yazılımın bir süre izlenmesi ve çıkabilecek yazılım hatalarına hızlı ve kalıcı müdahaleler bu süreçte çok önemlidir.

2.3. Platform Mimarisi

Bölüm 2 modüler çok katmanlı bir web uygulaması için seçilen mimariyi ele almıştır. Şekil 2.1, her katmanda bulunan mimariye ve başlıca teknolojik çözümlere genel bir bakış sunmaktadır. Sunum katmanı, platformu HTML5, JavaScript, CSS ve Bootstrap ile oluşturulan bir uygulama aracılığıyla sunar. Bu bölüm, her bir katmanın genel yapısını ve hedeflerini incelemektedir.



Şekil 2.1: Katmanlara Genel Bakış ve İlgili Teknolojik Çözümler.

Sunum katmanı, çok katmanlı bir uygulamanın arabirimidir. Web uygulamaları söz konusu olduğunda, genellikle web tarayıcıları aracılığıyla erişilebilir. Bu katmana genellikle uygulamanın ön ucu (front-end), bazen de istemci (client) tarafı denir.

Uygulama terimi hem ön uç hem de platforma bir bütün olarak atıfta bulunabilir; uygulama terimi (bölüm 2.1'de açıklanmıştır) buna bir örnektir.

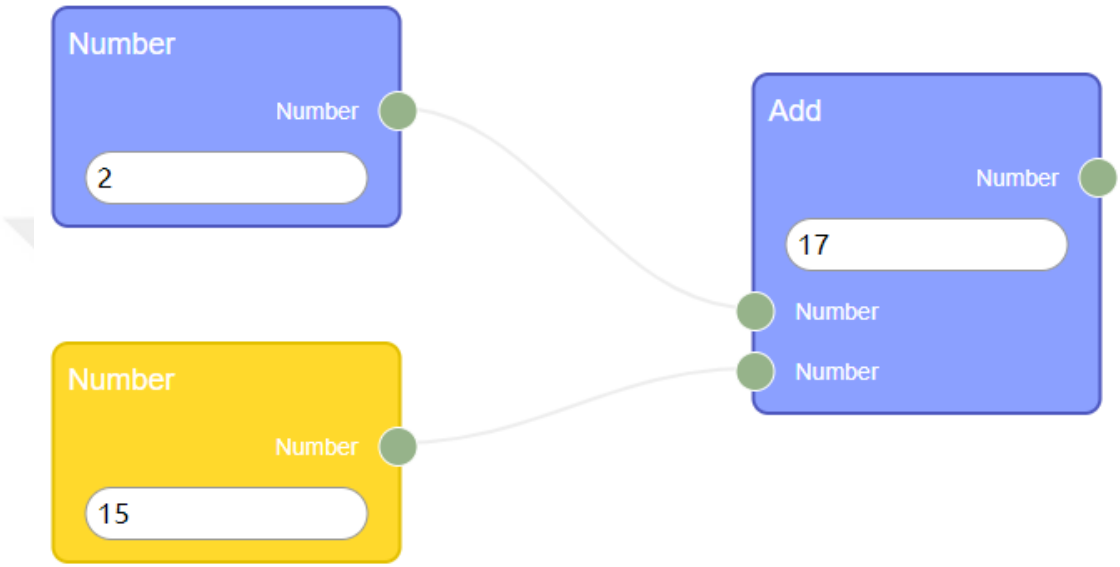
Web tarayıcıları başlangıçta çeşitli statik bilgileri almak ve oluşturmak için tasarlanmıştır ancak günümüzde tarayıcılar çok farklı işlevsellikler için kullanılır. Örneğin, yazılan kodun hata ayıklaması veya derlemesi gibi. Hangi arayüz (front-end) teknolojisinin kullanılacağına karar verilirken alternatifler üzerinden tüm tarayıcılarda eşit çalışan HTML5 ve JavaScript kombinasyonu seçildi. Kombinasyon, ek çalışma kitaplıklarının (additional run-time libraries) yüklenmesini gerektirmediği için web tarayıcıları tarafından yaygın olarak desteklenmektedir. 2013'ün bugüne JavaScript web uygulamalarına olan eğilim de katlanarak artmaktadır.

Teknoloji seçimini ilk adım olarak tanımlarsak mevcut çok sayıda JavaScript çözümü arasından en uygun olanı seçmek oldukça zor bir süreçtir. Karar verilecek seçim için hedef, MVC mimari modeline göre kullanıcı ara yüzünün geliştirilmesini kolaylaştıracak, iyi desteklenen ve kullanımı kolay bir çerçeve olacaktır (bölüm 1.1.3'de tanımlandığı gibi).

JavaScript genellikle iki kategoriye ayrılır: kütüphaneler ve çerçeveler (libraries and frameworks). Kütüphaneler, kodu basitleştiren ve yaygın sorunları çözen sınıflar ve işlevler koleksiyonudur; çerçeveler ise genellikle uygulamanın ihtiyaçlarına göre uyarlanmış daha kapsamlı mimari çözümlerdir (örneğin opencv.js gibi). Javascript seçimi için çok sayıda faktörü inceleyerek dört olası çözüm ele alındı ve çözüm için şu sorular soruldu: MVC ara yüzlerini ne şekilde destekliyor? Ne kadar olgunlaşmış ve iyi dokümante edilmiş? Üzerinde çalışmak için kolay ve sezgisel mi? RESTful API ile ne kadar kolay entegrasyon sağlanır? Uluslararası standartları ne kadar iyi destekliyor? Ek bir önemli nokta ise çözümün ne kadar tutarlı olduğu sorusudur (seçimin mevcut halini tamamı ile değiştirerek arayüz, fonksiyon ve kurgusal değişiklere gitmesi ileriki geliştirmeler için problem oluşturacağı varsayılarak bu soru sorulmuştur). Olasılıklar üzerine bir araştırmadan sonra OpenCV.js çerçevesi ile uyumlu çalışacak ve akış tabanlı programlamaya imkân sağlayacak rete.js[41] kütüphanesi seçildi. Seçilen kütüphanenin uyumluluğu uygulama yazılımına geçilmeden önce bir test web uygulaması üzerinde denendi (bakınız şekil 2.2).

[41] https://rete.js.org/

Github[42] topluluğu tarafından yeterince büyük bir desteğe (3.7K) ve 90'nın üzerinde uygulama projesinde kullanıma sahip[43] rete.js belirlenen ölçütleri karşıladı. İlgili kütüphanenin seçimi, uygulama yazılımı sürecinde hızlanmak ve alternatif kütüphaneler arasındaki yaklaşım farklılıklarını aşmak için süreç içinde biraz zaman aldı. Sonuçta rete.js kodu özel direktiflerle son derece modüller tuttuğu için tercih edildi.



Şekil 2.2: Basit bir test uygulaması.

Uygulamanın tasarımı için en popüler CSS çerçevelerinden biri olan Bootstrap[44] tercih edildi. Tasarım, uygulamanın geliştirilmesi esnasında birincil önceliği değildi, bu yüzden Bootstrap ideal bir seçimdi. Bootstrap, iyi görünümlü bir uygulama oluşturmak için hızlı ve kullanımı kolay araç ve hazır yapılar sunmaktadır[45].

2.4. Uygulamanın Gerçekleştirimi

OCC (OpenCV Companion) uygulaması ile belirlenen amaç direk web tarayıcısı üzerinden son kullanıcıların OpenCV kütüphanesinin fonksiyonlarını görsel programlama ve modüler bir çerçeve ile kullanması olacaktır. Bu kapsam dahilinde

[42] https://github.com/
[43] https://github.com/retejs/rete
[44] https://getbootstrap.com/
[45] https://getbootstrap.com/docs/4.4/examples/

uygulamanın mimarisiyle ilişkili gerçekleştirimi editör, yürütücü (engine), bileşenler, düğümler, soketler, kontroller ve olaylar başlığı altında incelenecektir.

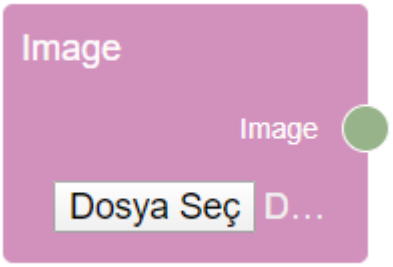
2.4.1. Kontroller (Controls)

Düğümlerin işlevselliğini genişletmek için kontroller (controls) gereklidir. Herhangi bir HTML şablonu veya üçüncü taraf nesnesi (giriş, metin, seçim, resim vb.) eklenmesi için gerekmektedir. Varsayılan olarak, tüm denetimler bir eklenti kullanılarak görüntülenir, ancak eklenti arabirimleri kullanılarak denetimler başka şekillerde oluşturabilir ve görüntülenebilir. Bu yüzden, kontroller yalnızca bazı bilgileri görüntülemekle kalmaz aynı zamanda daha fazla işlem için verileri bir düğümde kaydedebilir.

Aşağıdaki devralma (inherit) yöntemi her bir kontrol örneği için kullanılabilir:

```
getData('key')
setData('key', value)
```

Uygulama üzerinde her bir kontrol dinamik olarak düğümlerde kullanılmak üzere ayrı ayrı tanımlanmıştır. Bu kontrollere buton, checkbox, dosya yükleme, seçim, metin ve video örneğini verebiliriz. Ek 1'de dosya yükleme (fileupload) kontrolüne örnek verilmiştir. Şekil 2.3'te dosya yükleme amaçlı oluşturulmuş düğüme ve içinde yer alan kontrole örnek verilmiştir.



Şekil 2.3: Resim yükleme bloğu (düğümü) örneği.

2.4.2. Soketler (Sockets)

Soketler, hangi girişlerin ve çıkışların birbirine bağlanabileceğini belirlemek için kullanılır. Bir düğümden diğerine aktarılabilecek veri türünden sorumludurlar.

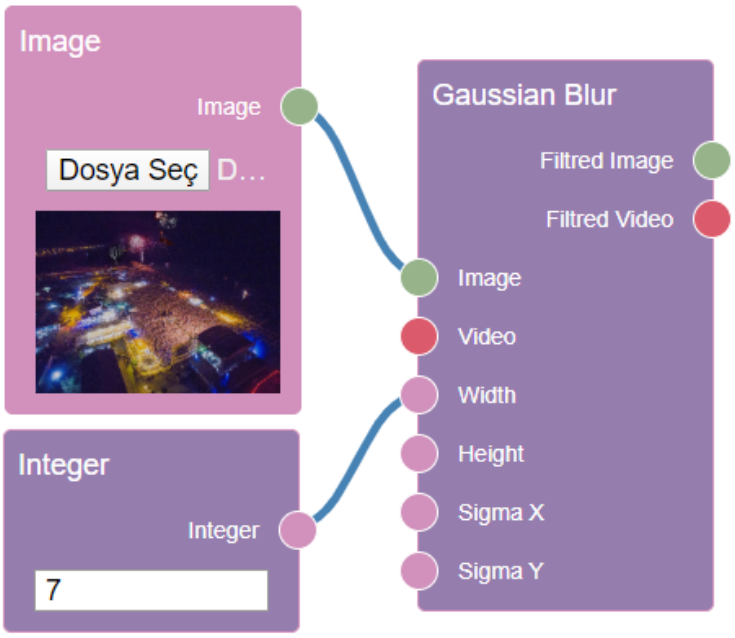
Editörde kullanılacak tüm soketler öncelikle tanımlanmalıdır. Tanımlanan bir soket örneği:

```
var pointSocket = new D3NE.Socket("Point", "Number Value", "hint");
var boolSocket = new D3NE.Socket("Bool", "Bool Value", "hint");
```

Tanımlanan parametre benzersiz olmalıdır. Bu durumda düğümler üzerinde eşlenecek girdilerin kullanıcı tarafından daha iyi anlaşılması için stil tanımlanacak bir biçiminde CSS sınıfı oluşturulabilir. Örneğin:

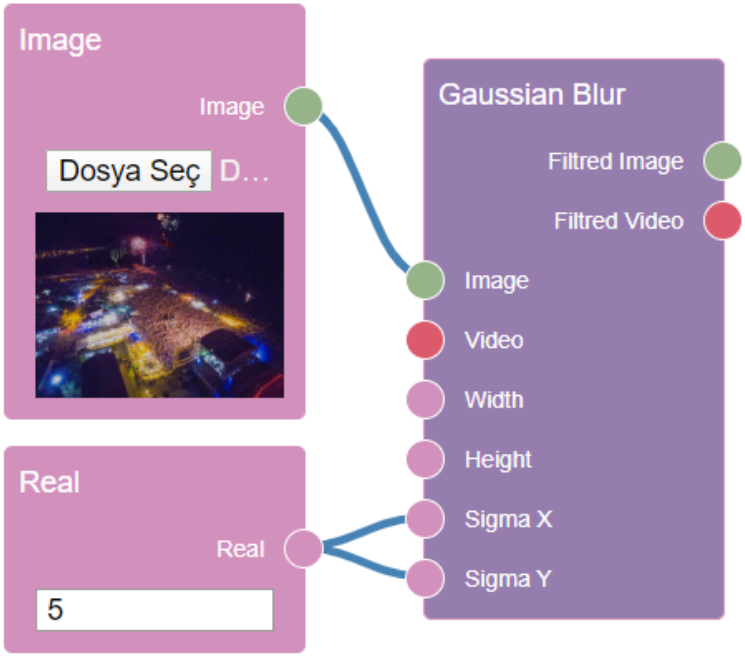
```
.socket.number {
  background: #96b38a
}
```

Sonuç olarak, girişler ve çıkışlar yalnızca aynı konektörlerle bağlanabilir (bakınız şekil 2.4).



Şekil 2.4: Aynı tip verilerin bağlantısına örnek.

Her ne kadar ilişkili veriler birbirine bağlantı sağlasa da farklı soketlerin birbirine bağlama gerektiği durumlar olabilir. Bunun için combineWith yöntemi kullanılmıştır (bakınız şekil 2.5).



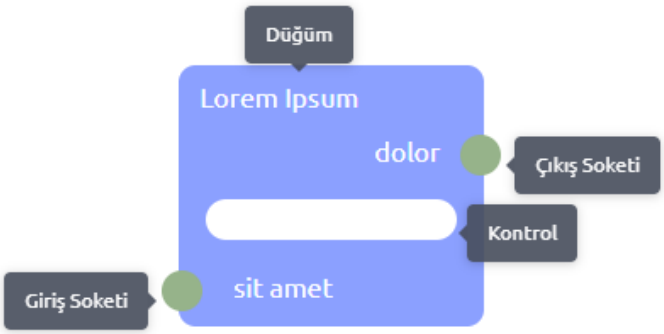
Şekil 2.5: combineWith yöntemine örnek.

'combineWidth' metodu örneği:

```
const anyTypeSocket = new D3NE.Socket('Any type');
numSocket.combineWith(anyTypeSocket);
```

2.4.3. Düğümler (Nodes)

Tüm düğümler ad, giriş(ler), çıkış(lar) ve kontroller içerebilir (bakınız şekil 2.6)



Şekil 2.6: Örnek düğüm (node).

Düğümün adı mutlaka tanımlamalı ve amacı açıkça belirtmelidir. Girişler ve çıkışlar sırasıyla düğümün solunda ve sağında olmalıdır (özel düğümler için istisnalar olabilir). Bir soketle temsil edilirler ve adları olabilir. Tüm çıkışlarda sınırsız sayıda

bağlantı olabilir. Varsayılan olarak, girişlerin yalnızca bir bağlantısı olabilir. Bu durum hem girişler hem de çıkışlar için değiştirilebilir.

Sınırsız sayıda bağlantıya sahip bir giriş, yalnızca bir bağlantıya sahip bir çıkışa izin verir. Örneğin:

```
const inp = new D3NE.Input("Image", imageSocket);
var out1 = new D3NE.Output("Image", imageSocket);
```

Kontroller doğrudan düğümün kendisinde bulunabilir veya belirli bir girişe eklenebilir. Aslında, ilk durumda yer oluşturma için eklentinin standart şablonunda belirlenir. İkinci durumda, girişte bağlantı olmadığında kontrol görüntülenir. Bu durum kontrolün başka bir düğümden iletilmediğinde gereklidir (Ek 2'de şekil 4.3'de yer alan resim düğümü için örnek kod çıktısı sunulmuştur).

2.4.4. Bileşenler (Components)

Bileşenler, yakın ilişkili fonksiyonları birleştirerek oluşturma ve işleme (building and processing) düğümlerini geliştirmeyi basitleştirmek için tasarlanmıştır.

Örnek Model:

```
class MyComponent extends D3NE.Component {
  constructor() {
    super("My Component"); // bileşen ismi
  }

  builder(node) {
    /// düğümü değiştir
  }

  worker(node, inputs, outputs) {
    /// veriyi işler
  }
}
```

Oluşturucu (builder) ve işçi (worker) birbirinden bağımsız olarak yürütülür, fakat aynı düğüm içinde birbirleriyle yakından ilişkilidirler. Bileşenleri kaydetmek için:

```
var comp = new MyComponent();

editor.register(comp);
engine.register(comp);
```

Düğüm Oluşturucuları (Node builders)

Bu yöntem (methods) düğüm örneklemesini düzenlemek ve tüm düğümleri JSON verilerinden geri yüklemek için editörce gereklidir. JSON verileri düğümlerin işleyiş mantığını ve akışını değil yalnızca statik bilgileri depolamalıdır.

Her özgün bileşende olması gereken oluşturucular:

```
class NumberComponent extends D3NE.Component {
  constructor() {
    super('Number');
  }

  builder(node){
    // düğümü düzenle
    node.data.num = 3;
    node.addInput(new Rete.Input('key1', 'Number', numSocket));
    node.addOutput(new Rete.Output('key2', 'Number',
numSocket));
  }
}
```

Düğüm İşçileri (Node Workers)

Düğüm verilerini işlemek için kullanılan bir fonksiyondur. İşlem sırasında aktarılan düğüm parametrelerini, girişleri, çıkışları ve bağımsız değişkenleri alırlar (isteğe bağlı).

```
class NumberComponent extends D3NE.Component {
  constructor(){
    super('Number');
  }

  async worker(node, inputs, outputs){
    // inputs['key1']
    outputs['key2'] = node.data.num;
  }
}
```

Yukarıdaki kod çıktısında görüldüğü gibi asenkron fonksiyonlar (asynchronous functions) kullanabilir. Asenkron kullanım ana iş parçacığını engellemeden, karmaşık hesaplamalar yapmak için gereklidir (örneğin, WebWorker kullanımı[46]).

[46] https://www.w3schools.com/html/html5_webworkers.asp

2.4.5. Eylemler (Events)

Çerçeve mimarisi, eylemler (events) modeli temelinde inşa edilmiştir. Bu mimari bileşenler arasında bağlantı kurmayı ve düzenleyici üzerinde ön tanımlı değişiklikler uygulamayı mümkün kılar. EK 3'de yer alan tabloda editör tarafından kullanılan eylem listesi yer almaktadır.

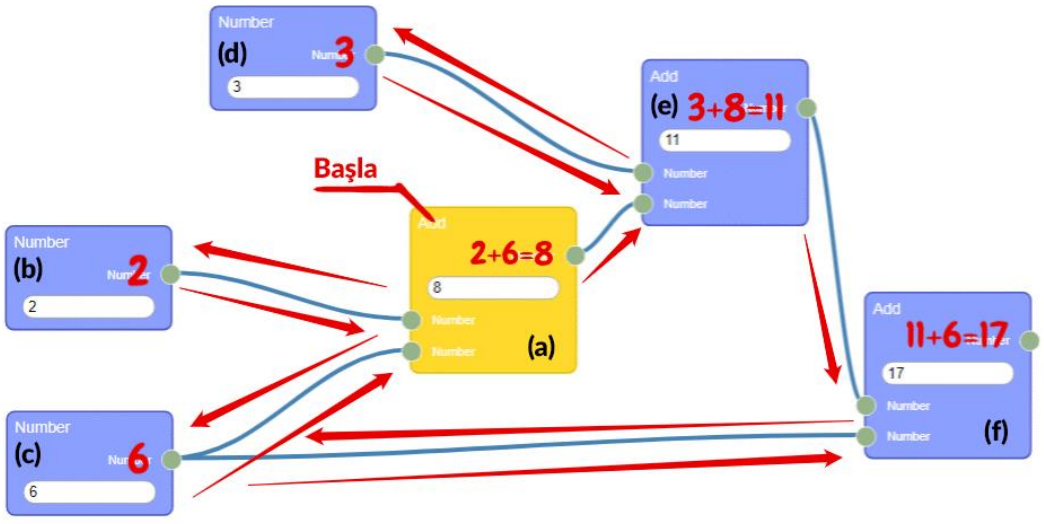
2.4.6. Yürütücü (Engine)

Bu sınıf, düğümlerdeki akışlara göre verileri işlemeye ve çıkış verilerinden girişe aktarmaya olanak tanır. Yürütücü (engine), editörün diğer bileşenlerine bağlı değildir ve editörden bağımsız olarak çalışabilir. Yürütücü, tanımlayıcı (identifier), bileşen işçileri (component worker) ve JSON verilerine ihtiyaç duyar. Örnek kod:

```
var engine = new D3NE.Engine(moduleDataId, components);
```

İşleme (Treatment)

Düğümler arası atlama (bypassing) şekil 2.7'de ki gibi gerçekleştirilir:



Şekil 2.7: Düğümler arası atlama (bypassing) örneği.

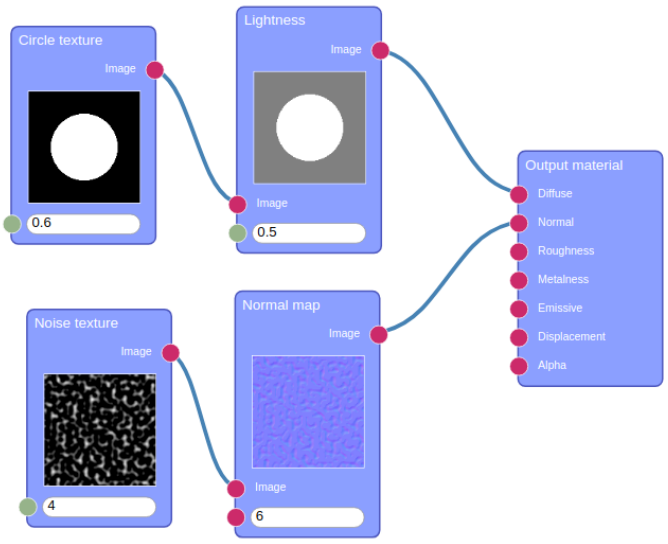
Şekil 2.7'de (a) düğümü öncelikle veriyi birinci girdi soketinden alır (b düğümünden gelen 2) daha sonra ikinci soketinden gelen veriyi alır (c düğümünden gelen 6) ve birincil işletimi gerçekleştirir (toplam 8). A düğümünün çıktı soketi veriyi aynı şekilde iletir. Her zaman birincil soketler işlem önceliğine sahiptir. Editörde çalışırken, değişikliklerin sonuçlarını hemen almak/görüntülemek gerektiğinde ilgili işlem 'process' eylemi (event) ile yapılabilir. Örnek kod:

```
editor.on('process', async () => {
    await engine.abort();
    await engine.process(editor.toJSON());
});
```

Genel olarak akış diyagramındaki her değişiklik için (düğümler, bağlantılar, düğüm verileri) işlem yapmak gerekir. İşçilerin (workers) asenkron olması nedeniyle, 'process' yöntemi de asenkrondur. İşlemeyi tetikleyen eylemler, önceki işlemenin tamamlanmasını beklemeden gerçekleşebileceğinden önceki işlemenin tamamlanmasını bekleyen ve veri bütünlüğünü garanti eden 'abort' yöntemine ihtiyaç vardır. Genellikle, işlemenin başlaması gereken veya tüm verilerin toplandığı bir tür birincil düğüm vardır. Bu düğümü belirmek için kullanılan kod:

```
engine.process(data, node.id);
```

Bu kullanım bir düğümün birden çok düğümden aynı anda veri istediği durumlarda önemli olabilir.



Şekil 2.8: Çoklu düğüm örneği.

Şekil 2.8'de belirtildiği gibi başlangıç düğümü "Output material" ise, üst ve alt düğüm zinciri paralel olarak işlenir. Eğer başlangıç düğümü "Lightness" ise alt düğümlerin işlenmesi mevcut düğümün sonucu alındıktan sonra başlatılır.

İşçilerin (workers) içine ek argümanlar da iletilebilir. Örneğin:

```
engine.process(data, null, arg1, arg2);
```

Argümanlar yukarıdaki örnekteki işlemle işlenen her bir çalışana aktarılacaktır. Örneğin:

```
worker(node, inputs, outputs, arg1, arg2) {
  outputs['num'] = node.data.num;
}
```

İşleme sırasında bir hata oluşursa (örneğin yineleme (recursion), yanlış startId, hatalı veri vb.) işlemin hata mesajı ve verileri alınabilir olmalıdır. Örneğin:

```
engine.on('error', ({ message, data }) => { });
```

2.4.7. OpenCv ile İletişim

Editör üzerinde opencv ile iletişim düğüm (node) oluşturulurken tanımlanmıştır. OpenCV fonksiyonlarının ihtiyaç duyduğu girdiler soketlerden gelen veriler ile beslenmektedir ve yine aynı şekilde çıktılar ise çıkış soketlerine parametre eşlemeleri ile iletilmektedir.

Daha iyi anlaşılması için "*Gaussian filtering*" fonksiyonu örnek alınırsa; bu fonksiyonda eşit filtre katsayılarından oluşan bir kutu filtresi yerine bir Gauss çekirdeği (kernel) kullanılır. OpenCv kütüphanesinde Cv2.GaussianBlur()[47] fonksiyonu ile uygulanır. Pozitif ve tek olması gereken kernel'in genişliği ve yüksekliği belirtmelidir. Ayrıca sırasıyla X ve Y yönlerindeki standart sapmayı, sigmaX ve sigmaY ile belirtilmelidir. Yalnızca sigmaX belirtilirse, sigmaY, sigmaX değerine eşit olarak alınır. Her ikisi de sıfır olarak girilirse, kernel boyutundan hesaplanır. Gauss filtresi, görüntüdeki Gauss gürültüsünü gidermede oldukça etkilidir. İlgili örneğin kullanım fonksiyonu:

[47] https://docs.opencv.org/master/dd/d6a/tutorial_js_filtering.html

```
cv.GaussianBlur (src, dst, ksize, sigmaX, sigmaY = 0, borderType =
cv.BORDER_DEFAULT)
```

Parametre tanımları;

src : Giriş görüntüsü; görüntü bağımsız olarak işlenen çok sayıda kanala sahip olabilir, ancak derinlik CV_8U, CV_16U, CV_16S, CV_32F veya CV_64F olmalıdır.

dst : src ile aynı boyutta ve türde çıktı görüntüsü.

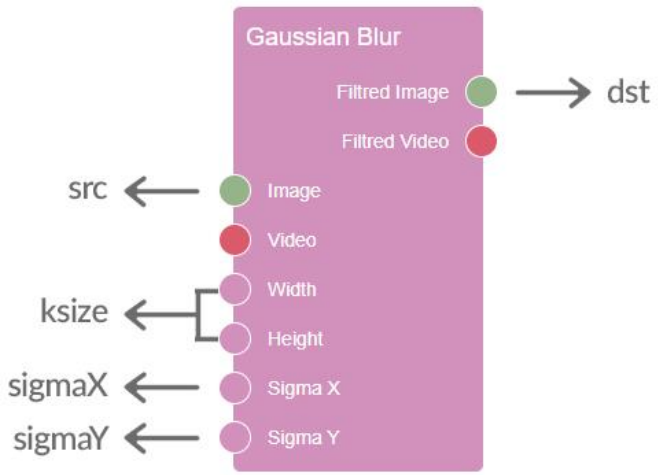
ksize : bulanıklığı belirleyen çekirdek (kernel) boyutu.

sigmaX : Gaussian çerkirdeğinin X yönünde standart sapması.

sigmaY : Gaussian çekirdeğinin Y yönünde standart sapması.

borderType : piksel dışdeğerleme (ekstrapolasyon) metodu.

Şekil 2.9'da OCC bileşeni olarak tanımlanan Gaussian Blur düğümünün OpenCV ile parametresel ilişkisi sunulmuştur.



Şekil 2.9: Gaussian Blur düğümünün OpenCV ile parametresel ilişkisini gösteren örnek.

Şekil 2.9'da örneği verilen *Gaussian Blur* bileşeni parametrelerinin opencv kütüphanesi tarafından sunulan kod bloğu örneği aşağıda sunulmuştur.

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let ksize = new cv.Size(3, 3);
cv.GaussianBlur(src, dst, ksize, 0, 0, cv.BORDER_DEFAULT);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete();
```

Yukarıda verilen kod bloğunun çıktısı Şekil 2.10'da incelenebilir.

Şekil 2.10: Gaussian Filtering uygulama örneği.

Gaussian Filtering fonksiyonu için sunulan tüm bu örnekler incelendiğinde sürecin tamamı ile parametreler ile yürütüldüğü anlaşılacaktır. Parametre eşlemeleri bileşenler oluşturulurken (bakınız bölüm 2.4.4) tanımlanmaktadır. Image Processing içinde yer alan Gaussian Blurring[48] fonksiyonu için örnek kod çıktısı EK 4'de sunulmuştur.

2.4.8. Python için Kod Çıktısı

Python[49], Guido van Rossum'un başlattığı, basitliği ve kod okunabilirliği nedeniyle kısa sürede çok popüler hale gelen genel amaçlı bir programlama dilidir. Programcının, okunabilirliğini azaltmadan fikirlerini daha az kod satırında ifade etmesini sağlar. C / C ++ gibi diğer dillerle karşılaştırıldığında, Python daha yavaştır. Ancak Python'un bir başka önemli özelliği de C / C ++ ile kolayca genişletilebilmesidir. Bu özellik, C / C ++ 'da hesaplama açısından yoğun kodlar

[48] https://docs.opencv.org/master/dd/d6a/tutorial_js_filtering.html
[49] https://www.python.org/about/

yazmaya sebep olsa bile aynı zamanda bir Python paketleyici (wrapper) oluşturmaya yardımcı olur. Bu paketleyici Python modülleri olarak kullanılabilir. Bu kullanım iki avantaj sağlar: ilk olarak, kod orijinal C / C ++ kodu kadar hızlı olacaktır (arka planda çalışan kod gerçek C ++ kodu olduğu için) ve ikinci avantaj ise Python'da kodlama çok kolaylaşacaktır. Bu tanım performans ve kullanım açısından OpenCV-Python'un nasıl çalıştığına örnektir. Numpy[50] sayısal işlemler için oldukça optimize edilmiş bir kütüphanedir. MATLAB tarzı bir sözdizimi kullanır. Tüm OpenCV dizi yapıları Numpy dizilerine dönüştürülebilir. OpenCV, görüntüleri saklamak için NumPy veri yapılarını kullanır. Bu yüzden OCC referans olarak numpy kütüphanesini de kod çıktısına eklemiştir. Bunun yanı sıra, Numpy'yi destekleyen SciPy, Matplotlib gibi diğer kütüphaneler de kullanılabilir. Tüm bu nitelikler incelendiğinde OpenCV ve Python bilgisayar görme problemlerinin hızlı prototiplenmesi için uygun bir araçtır.

Python kodu, tıpkı opencv ile iletişimde olduğu gibi (bakınız bölüm 2.4.7) girdi ve çıktı soketlerinden parametresel verilerin python kod betik biçimine dönüştürülmesi ile sağlanmıştır. OCC için python kod betik örneği Python-OpenCV[51] öğreticileri (tutorials) ilham alınarak oluşturulmuştur. İlgili öğreticide Gaussian Filtering için kullanılan kod örneği:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('opencv_logo.png')

blur = cv2.GaussianBlur(img,(15,15),0)

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```

Yukarıdaki kod örneği şekil 2.9'da sunulan tüm parametreleri kullanmaktadır ve kod çıktısı şekil 2.11'de sunulmuştur. OCC ara yüzünde python kodunu üretmek için öncelikle bileşen oluşturma aşamasında tüm parametrelerin tanımlanması gerekmektedir. Gaussian Blurring için python kodu parametre eşlemesini içeren ve bileşen oluşturucu içinde kullanılan kod EK 5'te sunulmuştur.

[50] https://numpy.org/
[51] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html

Original **Blurred**

Şekil 2.11: Python kodu ile üretilen Gaussian Filtering örneği.

Yukarıda yer alan açıklamalar doğrultusunda python kodu üretmek için ara yüzde yer alan python kodu üret butonuna tıkanıldığında aşağıdaki javascript dosyası çalışmakta ve kodu üretmektedir.

```
var readFunction = function (component) {
    return
$('script#'.concat(component)).html().trim().concat('\n\n');
}

var buildPythonCode = function () {
    let code = readFunction('pythonstart');

    components.data = editor.toJSON();
    const outputNodes = [];
    findOutputNodes(outputNodes);
    const inputNode = findInputNode(1);
    const firstNode = inputNode.outputs[0].connections[0].node;
    const activeNode = components.data.nodes[firstNode];
    const filterName = activeNode.title;

    const inputs = activeNode.inputs;

    code = code.concat(components.exportPython["Gaussian
Blur"](inputs));

    const lastLine = readFunction('pythonend');
    return code.concat(lastLine);
}
```

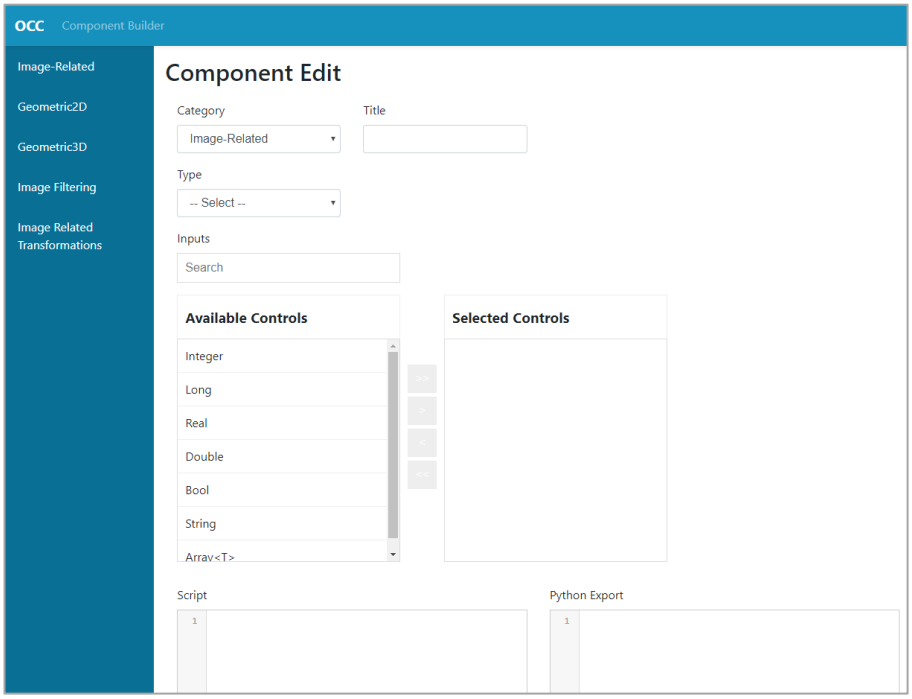
OCC kullanıcıları kendi python betik çıktısını tanımlamak istediğinde aşağıdaki script desenini kullanabilir.

```
<script type="text/html" id="pythonstart">
    import cv2
    import numpy as np
    from matplotlib import pyplot as plt

    img = cv2.imread('image.png',0)
</script>

<script type="text/html" id="pythonend">
    plt.show()
</script>
```

2.4.9. Bileşen Oluşturucu (Component Builder)

OCC uygulamasının editör ara yüzünde kullanılan düğümlerin oluşturulması için *bileşen oluşturucu modülü* geliştirilmiştir. Bu geliştirmenin en önemli nedeni; İlgili bileşenin kod tarafında oluşturulmasının bileşenlerin kontrolü ve opencv fonksiyonlarının yönetiminde zorluklar ve zaman maliyeti sebebi olacaktır.

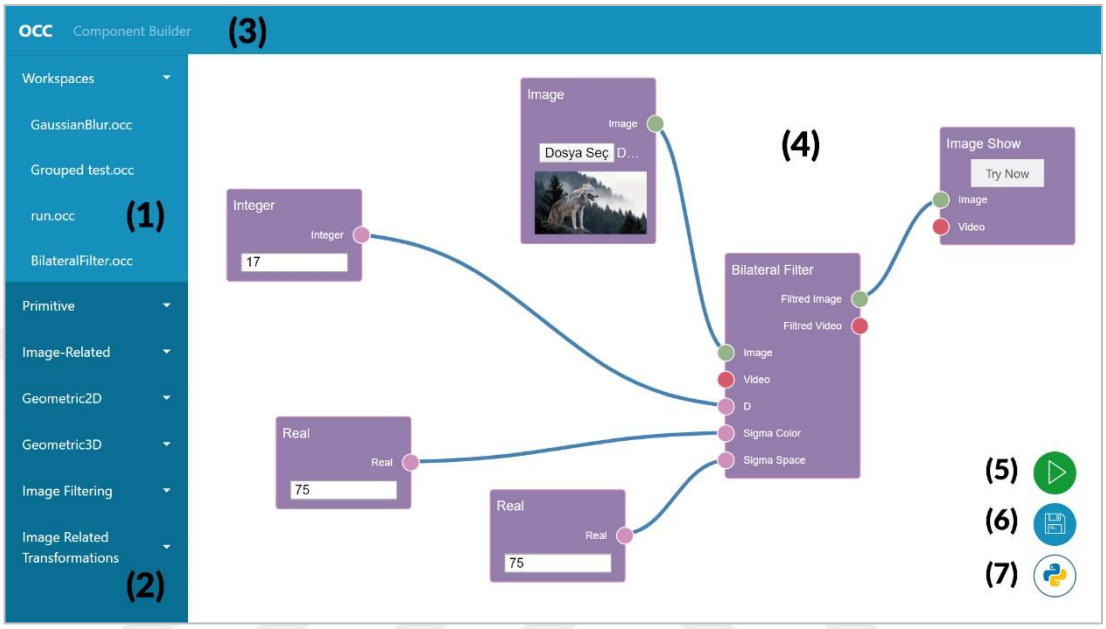
Bileşen oluşturucu modülü ile düğümün kategorisi, başlığı, tipi (kontrol veya bileşen), girdi ve çıktı soketleri, opencv kütüphanesi ile iletişimi sağlayacak scripti, python kod referans scripti ve menü sıralamasının belirlenmesi sağlanmıştır (şekil 2.12'de bileşen oluşturucu ara yüzü sunulmuştur).



Şekil 2.12: Bileşen Oluşturu (component builder) arayüzü.

2.4.10. Editör (Editor)

Editör, temel ifadeyle düğümler ve soketleri arasında bağlantıları olan ve son kullanıcıların yukarıda açıklaması yapılan tüm fonksiyonları görsel olarak kullanabileceği alanı ifade eder (Şekil 2.13'te editör ekranı örneği sunulmuştur).



Şekil 2.13: OCC Editör arayüzü.

Şekil 2.13'te yer alan şekil 'de (1) numaralı alan kaydedilen çalışma alanlarını, (2) numaralı alan menü çubuğunu, (3) numaralı alan üst menüyü, (4) numaralı alan çalışma alanını, (5) numaralı alan "sonuç göster" butonunu, (6) numaralı alan "çalışma alanını kaydet/güncelle" butonunu, (7) numaralı alan ise "python kodu oluştur" butonunu ifade etmektedir.

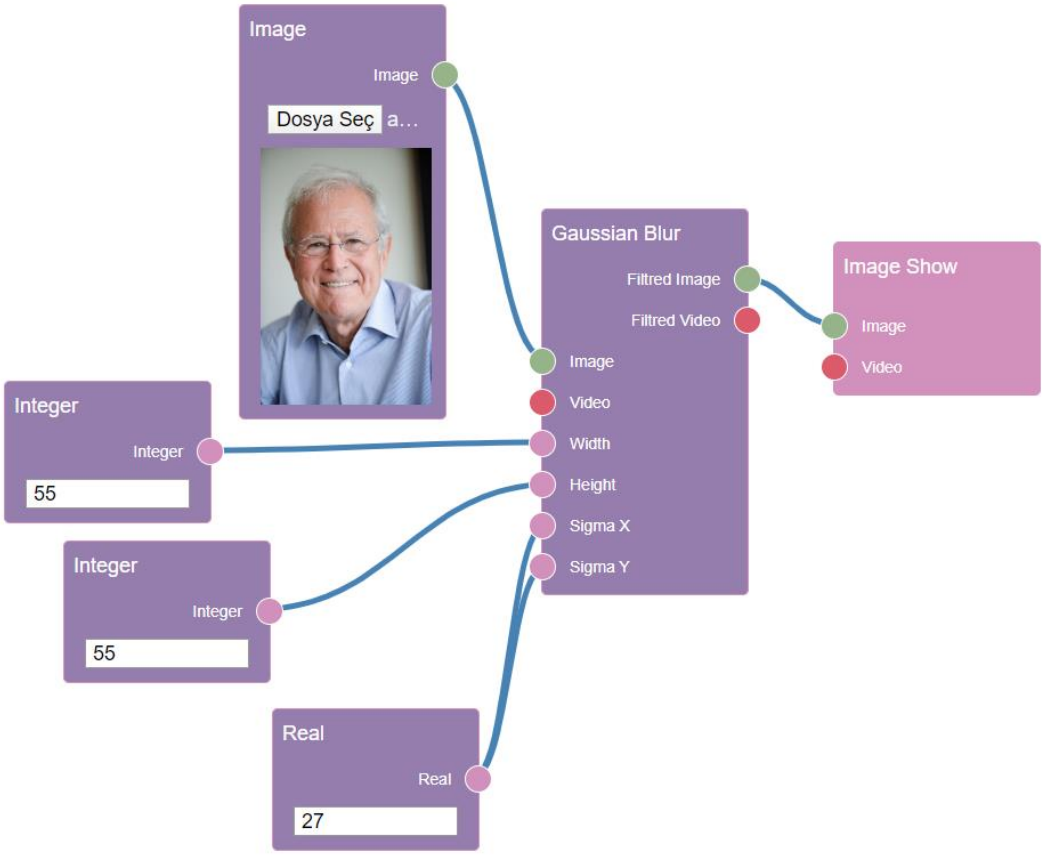
Aşağıda uygulanan editör özellikleri yer almaktadır:

- Çalışma alanı (taşıma, ölçeklendirme) ve kontrol düğümleri (taşıma, ekleme, silme) ile etkileşim.
- Bağlantıların, düğümlerin, girişlerinin / çıkışlarının ve kontrollerinin görüntülenmesi
- Olay İşleme (event handling) editörü

- Çalışma alanını (workspace) JSON biçiminde içe / dışa aktarma ve kaydetme.
- Bileşen editörü ile OpenCV fonksiyonlarının genişletilmesi
- Çalışma alanının (workspace), düğümlerin ve bağlantıların tasarımsal özelleştirilmesi (customization)

2.5. Uygulama Çıktısı

Bölüm 2.4'te detaylı bir şekilde aktarılan uygulama gerçekleştirimin sonuçları 2 kısımdan oluşmaktadır; veri ve kod. Veriye görsel, metin, sayısal grafik, video'yu örnek verilebilir. Kod kısmı ise çalışma alanında simüle edilen görsel programlamanın python kullanımı için çıktısını ifade etmektedir. Şekil 2.14'te OCC ile uygulanan örnek bir uygulama ve şekil 2.15'de veri çıktısı yer almaktadır.



Şekil 2.14: OCC görsel programlama ile uygulanan Gaussian Blurring fonksiyonu örneği.

Şekil 2.15 Gaussian Blurring için sunulan uygulamanın görsel çıktı örneği.

Yukarıda verilen görsel programlama örneğinin OCC tarafından üretilen Python kodu örneği aşağıda sunulmuştur.

```
import cv2
import numpy as np

    from matplotlib import pyplot as plt
    img = cv2.imread('image.png',0)

cv2.GaussianBlur(img,(55,55),27)
plt.show()
```

2.6. Metot Uygulama Özeti

Uygulamanın gerçekleştirimi (bakınız bölüm 2.4) kısmında aktarılan kontrolden yürütücüye kadar tüm uygulama konsepti parça parça test edilerek birbiri ile ilişkilendirilmiştir. Oluşturulan ilişkilerin sonucunda opencv kütüphanesi için blok tabanlı görsel programlama web platformu oluşmuştur. Oluşturulan platform üzerinde bazı opencv fonksiyonları test edilmiş ve bir örneği Şekil 2.2'de aktarıldığı gibi başarılı olmuştur. Ayrıca platform tarafından oluşturulan python kod çıktıları farklı bütünleşmiş geliştirme ortamlarında (IDE) test edilmiş ve sonuç görsel programlama ara yüzünde alınan çıktı ile aynı olmuştur.

ÜÇÜNCÜ BÖLÜM

3. SONUÇLAR VE ÖNERİLER

Bu çalışma, görsel programlama ile opencv kütüphanesinin simüle edilmesini, python için kod üretilmesini ve bağımsız bir geliştirme ortamı sağlayan blok tabanlı programlama aracının geliştirilmesini sunmuştur. Platformunun fikir aşamasından, son kullanıcıya sunulan editör ekranına, tüm süreç bütünüyle sunulmaya çalışıldı. Öncelikli olarak OCC uygulamasının mimarisinin geliştirilmesine katkıda bulunulmak hedeflendi ve uygulamanın ilk sürümü geliştirildi. İlk bölümden itibaren modüler tasarım, görsel programlama ve iş mantığına odaklanılarak uygulamanın temel kavramları üzerinde durulmaya çalışıldı. Nihayetinde bu tez, açıklanan uygulama platformu fikri ve bu fikrin en iyi şekilde nasıl hayata geçirileceği konusunun sonucudur. OCC uygulaması hem sunucuda hem de fiziksel depolama düzeyinde yatay olarak ölçeklenebilir. Uygulama oldukça modüler ve platform bağımsız çalışmaktadır. Platform saf JavaScript kodu ile geliştirilmiştir. Bu yüzden herhangi bir programlama dili ile çok hızlı entegre edilebilir (.net, php, python, ruby, flask vb.) ve ilişkisel kısım kolayca değiştirilebilir. OCC uygulaması oldukça esnek ve genişletilebilir yapıdadır ve farklı uygulamalara dahili olarak kolayca eklenebilir. Nihai hedef opencv kullanıcılarının yoğun olarak OCC platformunu kullanması ve açık kaynak kod üzerinden ihtiyaçları doğrultusunda geliştirebilmesidir.

Kullanıcı ara yüzünün tüm parçaları kullanım kolaylığı göz önünde bulundurularak tasarlanmıştır ve modern ara yüz kütüphaneleri kullanılmıştır (örn. Bootstrap). Ancak başarı düzeyi geliştiriciler tarafından değil kullanıcılar tarafından değerlendirilecektir. Kullanıcı deneyim tasarımı ara yüz tasarımı kadar önemli bir konudur. Kullanıcı deneyimini, kullanıcıları merkeze alarak iş tarafının ihtiyaçları ve kısıtları ile teknolojinin imkanlarını birleştirerek optimuma ulaşmak olarak tanımlanabilir. Ara yüz öğrenmekle başlar ve sayısal verilerle optimize edilmesi bir uygulamanın kullanım alışkanlığı için çok önemlidir. [57] Proje içerisinde yer alan tüm blok, editör, bileşen ve menü tasarımları özelleştirilebilir yapıdadır. Ayrıca günümüzde en çok kullanılan cihaz ekran çözünürlüklerinin dikkate alınması (mobil, tablet, pc) uygulanacak tasarımların ekran bağımsız çalışması açısından faydalı

olacaktır [58]. Mevcut ara yüz için kullanılabilirlik testi uygulanmıştır lakin 5 sn testi, tıklama testi, a/b testi, direkt gözlem, uzman analizi, netnografi, ağaç testi, anket ve deneyim haritası[52] gibi kullanıcı deneyimine yönelik testlerin OCC geliştirme gönülleri tarafından uygulanması faydalı olacaktır.

Özellikle veri akışı ve tetikleyicilerin (otomatik işlemeyi kolaylaştırmak için) kullanımıyla OpenCV kütüphanesi için blok tabanlı programlama aracı oluşturmanın asıl amacı çoğunlukla yerine getirilmiştir. Yine de hiç uygulanamayan önemli bir kısım bulunmaktadır (opencv kütüphanesinin tüm fonksiyonlarının OCC'ye aktarılarak tanımlanması – tüm liste için EK 6'ya bakınız). OpenCV kütüphanesinin içinde yer alan tüm fonksiyonlar, metotlar vb. OpenCV geliştiricileri tarafından tek bir çıktı halinde sunulmamaktadır. Tüm fonksiyonları her yeni opencv versiyonu için güncellenecek biçimde farklı web teknikleri ile OCC bileşeni olarak otomatik aktarılmaya çalışıldı lakin zaman kısıtlamalarına bağlı olarak böyle bir uygulanmanın gerçekleştirimi minimal seviyede kalmıştır ve uygulama analizinde temel öncelik olmamıştır. Bununla birlikte, opencv kütüphanesinin tüm önemli fonksiyonlarının tek seferde OCC platformuna aktarımı gönüllü geliştiriciler tarafından geliştirme kararı verilmesi durumunda, entegrasyon için platformun bileşen oluşturucu modülünün bu işlem için hazır olduğunu belirtmek faydalı olacaktır.

OCC platformu aktif olarak geliştirilmekte ve temel uygulamaların birçok kısmı daha iyi işlevselliklere yükseltildi. Süreç içerisinde farklı geliştirme fikirleri de oluştu, bazıları belli oranda uygulanırken (örneğin bileşen oluşturucu / component builder) bazıları gönüllü geliştiriciler için açık kaynak olarak bırakıldı. Tezin en heyecan verici noktası opencv kütüphanesi için blok tabanlı programlama aracı ile birçok alanda (örn: görüntü anlamlandırma, haritalama, sağlık, insansız hava araçları, otonom araçlar vb) neler yapabileceğini referans veren özel uygulamaların geliştirilmesi olacaktır. Bu uygulamaların çoğunun çekirdeği, görüntü sınıflandırma, yerelleştirme ve algılama gibi görsel tanıma görevleridir. Derin öğrenme yaklaşımlarındaki son gelişmeler görsel tanıma sistemlerinin performansını ve bu alana olan ilgiyi büyük ölçüde geliştirmiştir. Bu yüzden OCC ile gelecekte ulaşılmak istenilen noktaya, kullanıcıların sıfır kod ile kendi yapay sinir ağlarını kurguladığı, uyguladığı, eğittiği ve hatalarını ayıkladığı görsel programlama platformu olarak örnek verilebilir.

[52] https://userpeek.com/blog/user-experience-testing/

Günümüzde bir uygulamanın başarı seviyesi entegre olduğu platform ve uygulama sayısı ile karşılaştırılmaktadır. Örneğin anlık analiz yapılması gereken yüzlerce video için OCC'nin bulut tabanlı bir yapıda sanallaştırılması gerekmektedir çünkü bu durumda temel kaygı hız ve performans olacaktır. Eğer binlerce saat uzunluktaki bir video üzerinde önemli bir ayrıntı analiz edilecekse bu noktada büyük veri modeli ile OCC kullanılacaktır. Aynı şekilde bir görüntü veya işlem üzerinde mantıksal operatör kullanılması gerektiğinde OCC'nin bu ortamı sağlayan bir platformla entegre olması kullanıcısına fonksiyonel üstünlük sağlayacaktır. Bu yüzden OCC açık kaynak olarak sunulmuştur ve farklı platformlarla kolay entegrasyonu için JavaScript tabanlı (platform bağımsız) geliştirilmiştir. Gelecekte geliştirilmesi planlanan uçtan uca API servisleri ile uygulamanın daha fazla platform tarafından benimsenmesi ve iç fonksiyonelliğine ilave edilmesi ayrıca önem taşımaktadır. Bu yüzden OCC gerek iç gerekse dış iletişimi için tümüyle JSON kullanmaktadır.

OpenCV araştırma ve akademi için günümüzde neredeyse temel görüntü işleme kütüphanesi olarak kullanılmaktadır. OpenCV v2.3'ten bu yana resmi olarak makine öğrenimini (ML) desteklemektedir. Ayrıca OpenCV mevcut ML modellerinde olmayan çok yönlülük sunmaktadır. Yazılım dünyası gelişirken, donanım da değişiyor. Kameralar artık daha ucuz, daha yetenekli ve cep telefonlarından trafik ışıklarına, fabrika ve ev izlemeye doğru kullanımı çoğaldı. Aynı şekilde nesnelerin interneti kullanım alanı da günden güne yaygınlaşmaktadır. Bu perspektifte OCC ile OpenCV kütüphanesinin tüm işlevselliklerinin donanımlar üzerinde kullanımın mümkün olduğunu belirtmekte fayda olacaktır.

Bilgisayar vizyonu (computer vision) ileride zengin bir geleceğe sahip anahtar sağlayıcı teknolojilerden biri olacaktır. Aynı şekilde bu teknolojinin en büyük taşıyıcılarından biri olarak OpenCV aday gösterilmektedir. OpenCV kütüphanesinin gelecekte konumlanacağı vizyon çerçevesinde farklı kullanım senaryoları ve alanları için (derin öğrenme, mobil, gözlükler, gömülü uygulamalar, 3d, ışık alanı kameraları, robotik, bulut bilişimi ve online eğitim gibi alanlarda) OCC doğru bir uygulama fikri olacaktır.

Alanında ilk uygulama olan OCC'nin, görüntü işleme topluluğuna katılmak isteyen herkesi gelecek çalışmalar için cesaretlendirmesi ve heyecanlandırması umut edilmektedir.

KAYNAKÇA

[1] J. P. Morrison, *Flow-Based Programming, 2nd Edition: A New Approach to Applictaion Development*. Paramount, CA: CreateSpace, 2010.

[2] Fielding, R. T., & Taylor, R. N. (2002). "*Principled Design of the Modern Web Architecture"*. *ACM Transactions on Internet Technology*, *2*(2), 115–150. https://doi.org/10.1145/514183.514185

[3] Piessens, F., Verhanneman, T., & Win, B. De. (2002). On the importance of the separation-of-concerns principle in secure software engineering. *Workshop on the Application of Engineering Principles to System Security Design*, 1–10. Retrieved from http://www.acsac.org/waepssd/papers/02-piessens.pdf

[4] Dijkstra, E. W., & Dijkstra, E. W. (1982). On the Role of Scientific Thought. *Selected Writings on Computing: A Personal Perspective*, 60–66. https://doi.org/10.1007/978-1-4612-5695-3_12

[5] Johnston, W. M., Hanna, J. R. P., & Millar, R. J. (2004). Advances in dataflow programming languages. *ACM Computing Surveys*, *36*(1), 1–34. https://doi.org/10.1145/1013208.1013209

[6] Arikpo, I. I., Ogban, F. U., & Eteng, I. E. (2008). Von neumann architecture and modern computers. *Global Journal of Mathematical Sciences*, *6*(2). https://doi.org/10.4314/gjmas.v6i2.21415

[7] E. Baroth and C. Hartsough, "*Visual programming in the real World"*. In M. Burnett, A. Goldberg, T. Lewis (eds.), Visual Object-Oriented Programming: Concepts and Environments, Prentice-Hall, Englewood Cliffs, NJ; Manning Publications, Greenwich, Connecticut; and IEEE, Los Alamitos, California, 1995

[8] Van Der Aalst, W. M. P. (1998). The application of Petri nets to workflow management. *Journal of Circuits, Systems and Computers*, *8*(1), 21–66. https://doi.org/10.1142/S0218126698000043

[9] D. L. Parnas, "On the Criteria to Be Used in Decomposing Systems into Modules," *Commun*. *ACM*, vol. 15, no. 12, pp. 1053–1058, Dec. 1972.

[10] Krasner, G. (1988). A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, *1*(3), 26–49.

[11] Stevens, W. P., Myers, G. J., & Constantine, L. L. (1999). Structured design. *IBM Systems Journal*, *38*(2), 231–256. https://doi.org/10.1147/sj.382.0231

[12] Beck, F., & Diehl, S. (2011). On the congruence of modularity and code coupling. *SIGSOFT/FSE 2011 - Proceedings of the 19th ACM SIGSOFT Symposium on Foundations of Software Engineering*, (September), 354–364. https://doi.org/10.1145/2025113.2025162

[13] Black, S. (2006). The Role of Ripple Effect in Software Evolution. In Software Evolution and Feedback (eds N.H. Madhavji, J.C. Fernández-Ramil and D.E. Perry). doi:10.1002/0470871822.ch12

[14] Laney, D. (2001). 3D Data Management: Controlling Data Volume, Velocity, and Variety (). META Group.

[15] Snijders, C., Matzat, U., & Reips, U. (2013). "Big Data": Big Gaps of Knowledge in the Field of Internet Science. *International Journal of Internet Science*, *7*(1), 1–5.

[16] Katz, R. N., Goldstein, P. J., & Yanosky, R. (2009). Demystifying Cloud Computing for Higher Education Highlights of Cloud Computing. *Educause Research Bulletin*, *2009*(19). Retrieved from http://www.educause.edu/Resources/DemystifyingCloudComputingforH/180117

[17] Mell, Peter & Grance, Timothy. (2011). The NIST Definition of Cloud Computing (Draft) Recommendations of the National Institute of Standards and Technology. Nist Special Publication. 145. 1-7.

[18] Bondi, A. B. (2000). Characteristics of scalability and their impact on performance. *Proceedings Second International Workshop on Software and Performance WOSP 2000*, 195–203. https://doi.org/10.1145/350391.350432

[19] Michael, M., Moreira, J. E., Shiloach, D., & Wisniewski, R. W. (2007). *Scale-up x Scale-out : A Case Study using Nutch / Lucene*.

[20] Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software: Amazon.de: Eric J. Evans: Fremdsprachige Bücher*. *7873*(415), 529. Retrieved from https://www.amazon.de/Domain-Driven-Design-Tackling-Complexity-Software/dp/0321125215

[21] Pautasso, C., Zimmermann, O., & Leymann, F. (2008). RESTful web services vs. "Big" web services: Making the right architectural decision. *Proceeding of the 17th International Conference on World Wide Web 2008, WWW'08*, 805–814. https://doi.org/10.1145/1367497.1367606

[22] Szydlo, T., Brzoza-Woch, R., Sendorek, J., Windak, M., & Gniady, C. (2017). Flow-based programming for IoT leveraging fog computing. *Proceedings - 2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2017*, 74–79. https://doi.org/10.1109/WETICE.2017.17

[23] Ege, R. K. (1992). *Concepts of object-oriented programming (abstract)*. (April 1993), 217. https://doi.org/10.1145/157709.157761

[24] Russell, C. (2008). Bridging the: Object-Relational. *Queue*, *6*(3), 18–28. https://doi.org/10.1145/1394127.1394139

[25] Dave, M. (2012). *International Journal of Advanced Research in SQL and NoSQL Databases*. (June).

[26] Sousa, T. B. (2012). Dataflow Programming: Concept, Languages and Applications. *Doctoral Symposium on Informatics Engineering*, *7*, 13. Retrieved from http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

[27] Kumar, R., & Singh, P. (2014). *Instructıon Level Parallelısm – The Role Of Archıtecture And Compiler*. (April 2013).

[28] Traub, K.R., & Papadopoulos G.M. (1991). *Multithreading: A Revisionist View of Dataflow Architectures G.M. Papadopoulos.* (May).

[29] Joosten, S. (1994). Trigger Modelling for Workflow Analysis. *Proceedings CON*, (JANUARY 1994), 236–247. Retrieved from http://www.researchgate.net/publication/228786079_Trigger_modelling_for_workflow_analysis/file/3deec517ab91a7ff5e.pdf

[30] Demšar, J., Curk, T., Erjavec, A., Gorup, Č., Hočevar, T., Milutinovič, M., … Zupan, B. (2013). Orange: Data mining toolbox in python. *Journal of Machine Learning Research*, *14*(October 2019), 2349–2353.

[31] Berthold, M. R., Cebron, N., Dill, F., Di Fatta, G., Gabriel, T. R., Georg, F., … Wiswedel, B. (2006). KNIME: The konstanz information miner. *4th International Industrial Simulation Conference 2006, ISC 2006*, *11*(1), 58–61. https://doi.org/10.1145/1656274.1656280

[32] Blankenberg, D., Kuster, G. Von, Coraor, N., Ananda, G., Lazarus, R., Mangan, M., … Taylor, J. (2010). Galaxy: A web-based genome analysis tool for experimentalists. *Current Protocols in Molecular Biology*, (SUPPL. 89), 1–21. https://doi.org/10.1002/0471142727.mb1910s89

[33] Goecks, J., Nekrutenko, A., Taylor, J., Afgan, E., Ananda, G., Baker, D., … Vincent, K. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, *11*(8). https://doi.org/10.1186/gb-2010-11-8-r86

[34] Giardine, B., Riemer, C., Hardison, R. C., Burhans, R., Elnitski, L., Shah, P., … Nekrutenko, A. (2005). Galaxy: A platform for interactive large-scale genome analysis. *Genome Research*, *15*(10), 1451–1455. https://doi.org/10.1101/gr.4086505

[35] The MathWorks. Simulink - Simulation and Model-Based Design, [Online] https://www.mathworks.com/help/simulink/.

[36] GNU Radio resmi dokümantasyon sayfası. [Online] https://wiki.gnuradio.org/

[37] Blender resmi dokümantasyon sayfası. [Online] https://docs.blender.org/

[38] "Vpl sayfası." [Online]. https://msdn.microsoft.com/en-us/library/bb483088.aspx

[39] "Luna. hybrid-visual textual functional programming language." [Online]. http://www.luna-lang.org/

[40] “Unreal engine blueprints, a visual programming language for games." [Online]. https://docs.unrealengine.com/latest/INT/Engine/Blueprints/

[41] Schurr, A. (1997). BDL - a nondeterministic data flow programming language with backtracking. *IEEE Symposium on Visual Languages, Proceedings*, 394–401. https://doi.org/10.1109/vl.1997.626610

[42] Topping, C. J., Rehder, M. J., & Mayoh, B. H. (1999). Viola: A new visual programming language designed for the rapid development of interacting agent systems. *Acta Biotheoretica*, Vol. 47, pp. 129–140. https://doi.org/10.1023/A:1002070223107

[43] Haeberli, P. E. (1988). ConMan: A visual programming language for interactive graphics. *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1988*, *22*(4), 103–111.

[44] "Pd community site." [Online]. https://puredata.info/

[45] "Scratch website." [Online]. https://scratch.mit.edu/

[46] M. Matas, “The brain: A neural network built entirely in quartz composer." [Online]. https://www.youtube.com/watch?v=eUEr4P_RWDA

[47] Tools, G. (2007). Quartz Composer User Guide. *Interface*.

[48] "Max connects objects with virtual patch cords to create interactive sounds, graphics, and custom effects." [Online]. https://cycling74.com/products/max

[49] OpenCV resmî web sayfası. https://opencv.org/

[50] Shubair, A. H. (2014). *The Usability Of A Visual, Flow-Based Programming Environment For Non-Programmers*.

[51] Bassil, Y. (2012). A Simulation Model for the Waterfall Software Development Life Cycle. *International Journal of Engineering & Technology (iJET)*, ISSN: 2049-3444, Vol. 2, No. 5.

[52] Sutradhar, Mukti Rani et al. *"A New Version of Kerberos Authentication Protocol Using ECC and Threshold Cryptography for Cloud Security."* 2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR) (2018): 239-244.

[53] WikiZeroo.org görsel arşivi. [Online]. https://www.wikizeroo.org/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvRmlsZTpNVkMtUHJvY2Vzcy5zdmc

[54] Burnett, M. M. (1993). Visual object-oriented programming. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA, Part F129675*(April 1994), 127–129. https://doi.org/10.1145/260303.261240

[55] Wautelet, Y., Kiv, S., & Kolp, M. (2011). An iterative process for component-based software development centered on agents. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6910*(January 2014), 41–65. https://doi.org/10.1007/978-3-642-24016-4_3

[56] Rahman, M. M. (2019). *Waterfall Model : The Scientific Method of Software Engineering Waterfall Model : The Scientific Method of Software Engineering*. (December).

[57] Lam, M. (2011). Methodologies , tools , and techniques in practice for Web application development. Journal of Technology Research, 3, 1–20. Retrieved from http://www.aabri.com/manuscripts/11985.pdf

[58] Pirchheim, C. (2007). Visual Programming for Hybrid User Interfaces Mixed Reality UIs. (January).

EKLER

Uygulama gerçekleştirim (bakınız bölüm 4.4.) bölümünde kullanılan bazı kod parçaları ve gerekli açıklamaları ile aşağıda sunulmuştur.

EK 1: Dosya Yükleme Kontrolü Kod Çıktısı

```
class FileUploadControl extends D3NE.Control {

    constructor(emitter, key, defaultValue) {
        super('<input type="file" id="fileInput"
accept="image/*"/><div class="component-temp"></div>', function
(elem, control) {
            //elem.value = defaultValue;
            control.putData(key, defaultValue);
            elem.addEventListener('change', () => {
                loadInputCanvas();
                const reader = new FileReader();
                reader.onload = function (ev) {
                    const url = ev.target.result;
                    control.putData(key, url);
                    const canvas =
document.getElementById("canvasInput");
                    var ctx = canvas.getContext("2d");

                    const img = new Image();
                    img.onload = function () {
                        ctx.drawImage(img, 0, 0);
                    };
                    img.src = url;

                    const element =
document.getElementsByClassName('component-temp')[0];
                    if (element.firstChild)
                        element.firstChild.remove();
                    element.appendChild(img);
                };

                reader.readAsDataURL(elem.files[0]);
                //control.putData(key, elem.value);
            });


            if (defaultValue !== undefined) {
                const canvas =
document.getElementById("canvasInput");
                const ctx = canvas.getContext("2d");

                const img = new Image();
                img.onload = function () {
                    ctx.drawImage(img, 0, 0);
                };
                img.src = defaultValue;

                setTimeout(function() {
                        const element =
document.getElementsByClassName('component-temp')[0];
                        if (element.firstChild)
                            element.firstChild.remove();
```

```
                        element.appendChild(img);
                    },
                    100);
            }
        });
        this.emitter = emitter;
        this.key = key;
        this.template = '<input type="file" id="fileInput"
:value="value" @input="change($event)" accept="image/*"/><div
class="component-temp"></div>';

        this.scope = {
            value: null,
            change: this.change.bind(this)
        };
    }

    onChange() { }

    change(e) {
        const input = e.target;
        if (input.files && input.files[0]) {
            loadInputCanvas();
            const reader = new FileReader();
            reader.onload = function (ev) {
                const url = ev.target.result;

                var canvas = document.getElementById("canvasInput");
                var ctx = canvas.getContext("2d");

                const img = new Image();
                img.onload = function () {
                    ctx.drawImage(img, 0, 0);
                };
                img.src = url;

                const element =
document.getElementsByClassName('component-temp')[0];
                if (element.firstChild)
                    element.firstChild.remove();
                element.appendChild(img);
            };

            reader.readAsDataURL(input.files[0]);

            // TODO: set base64 url
            this.scope.value = e.target.value;
            this.update();
            this.onChange();
        }
    }

    update() {
        if (this.key)
            this.putData(this.key, this.scope.value);
        this.emitter.trigger('process');
    }

    mounted() {
        this.scope.value = this.getData(this.key) || '...';
        this.update();
```

```
    }

    setValue(val) {
        this.scope.value = val;
    }

}
```

EK 2: Resim Bloğu Oluşturma Kod Çıktısı

```
var imageComponent = new D3NE.Component("Image", {

    builder(node) {
        var out1 = new D3NE.Output("Image", imageSocket);
        var ctrl = new FileUploadControl(this.editor, 'Image',
node.data.Image);

        return node
            .addControl(ctrl)
            .addOutput(out1);
    },
    change(node, item) {
        node.data.module = item;
        //console.log('InputComponent');
        this.editor.trigger('process');
    },
    worker(node, inputs) {
        //console.log('InputComponent', node.id, node.data);
    }
})
```

EK 3: Editör Tarafından Kullanılan Eylem Listesi

Olay (Event) Adı	Seçenekler	Önlenebilir	Yorum
nodecreate	node	✓	
nodecreated	node		
noderemove	node	✓	
noderemoved	node		
connectioncreate	{ output, input }	✓	
connectioncreated	connection		
connectionremove	connection	✓	
connectionremoved	connection		
translatenode	{ node, dx, dy }		
nodetranslate	{ node, x, y }	✓	
nodetranslated	{ node }		
nodedraged	node		
selectnode	node		
nodeselect	node	✓	
nodeselected	node		
rendernode	{ el, node, component, bindSocket, bindControl }		el = htmlelement
rendersocket	{ el, input, output, socket }		
rendercontrol	{ el, control }		
renderconnection	{ el, connection, x1, y1, x2, y2 }		
updateconnection	{ el, connection, points }		
componentregister	component		hem editörde hem de yürütücüde (engine) kullanılır

keydown	e		Kullanım parametresi = KeyboardEvent
keyup	e		Kullanım parametresi = KeyboardEvent
translate	{ transform, x, y }	✓	sahneyi hareket ettirerek tetiklenir
translated			
zoom	{ translate, zoom, source }	✓	zum yaparken tetiklenir
zoomed	{ source }		
click	{ e, container }		e = MouseEvent
mousemove	{ x, y }		fare konumu çalışma alanına göre güncellendiğinde tetiklenir
contextmenu	{ e, view, node }		menünün çağrıldığı nesneye bağlı olarak, görünüm veya düğüm olarak döner
import	data		
export	data		
process			harici kullanım için tasarlanmıştır
error	{ message, data }		
warn	message		
destroy			

EK 4: OpenCV - Gaussian Blurring Kod Çıktısı

```
components.processFilter['Gaussian Blur'] = function (inputs,
src, dst) {
       const width = components.getIntegerData(inputs, 2);
       if (width === null) {
           alert('Width node not found!');
           return false;
       };
       const height = components.getIntegerData(inputs, 3);
       if (height === null) {
           alert('Height node not found!');
           return false;
       };
       const sigmaX = components.getRealData(inputs, 4);
       if (sigmaX === null) {
           alert('SigmaX node not found!');
           return false;
       };
       const sigmaY = components.getRealData(inputs, 5);
       if (sigmaY === null) {
           alert('SigmaY node not found!');
           return false;
       };

       const ksize = new cv.Size(width, height);
       debugger;
       cv.GaussianBlur(src, dst, ksize, sigmaX, sigmaY,
   cv.BORDER_DEFAULT);
}
```

EK 5: Python Kod Oluşturucu Kod Örneği

```
components.exportPython['Gaussian Blur'] = function(inputs) {
    const width = components.getIntegerData(inputs, 2);
    if (width === null) {
        alert('Width node not found!');
        return false;
    };
    const height = components.getIntegerData(inputs, 3);
    if (height === null) {
        alert('Height node not found!');
        return false;
    };
    const sigma = components.getRealData(inputs, 4);
    if (sigma === null) {
        alert('Sigma node not found!');
        return false;
    };

    let code = readFunction('gaussianblur');
    return code
        .replace('{{width}}', width)
        .replace('{{height}}', height)
        .replace('{{sigma}}', sigma);
}
```

EK 6: OpenCV Fonksiyon Listesi

Camera Calibration and 3D Reconstruction
CreateCameraMatrix
CreateChessboardMatrix
CreateCirclesGridMatrix
cvCalibrateCamera
cvCalibrationMatrixValues
cvCorrespondEpilines
cvDrawChessboardCorners
cvEstimateAffine3D
cvFindChessboardCorners
cvFindCirclesGrid
cvGetOptimalNewCameraMatrix
cvInitCameraMatrix2D
cvRodrigues
cvStereoBM
cvStereoCalibrateAVL
cvStereoRectifyAVL
cvStereoSGBM

Drawing
cvArrowedLine
cvCircle
cvEllipse
cvFillConvexPoly
cvFillPoly
cvGetTextSize
cvLine
cvPolylines
cvPutText
cvRectangle

Feature Detection
cvCanny
cvCornerEigenValsAndVecs
cvCornerHarris
cvCornerMinEigenVal

cvCornerSubPix
cvGoodFeaturesToTrack
cvHoughCircles
cvHoughLines
cvHoughLinesP
cvLineSegmentDetector
cvPreCornerDetect

Geometric Image Transformations
cvGetAffineTransform
cvGetDefaultNewCameraMatrix
cvGetPerspectiveTransform
cvGetRectSubPix
cvGetRotationMatrix2D
cvInitUndistortRectifyMap
cvInvertAffineTransform
cvRemap
cvResize
cvUndistort
cvUndistortPoints
cvWarpAffine
cvWarpPerspective

Histogram Equalization
cvCLAHE

Image Filtering
cvBilateralFilter
cvBlur
cvBorderInterpolate
cvBoxFilter
cvBuildPyramid
cvCopyMakeBorder
cvDilate
cvErode
cvFilter2D
cvGaussianBlur
cvGetDerivKernels

cvGetGaborKernel
cvGetGaussianKernel
cvGetStructuringElement
cvLaplacian
cvMedianBlur
cvMorphologyEx
cvPyrDown
cvPyrUp
cvScharr
cvSepFilter2D
cvSobel
cvSqrBoxFilter

Miscellaneous Image Transformations
cvAdaptiveThreshold
cvApplyColorMap
cvCvtColor
cvDistanceTransform
cvFloodFill
cvGrabCut
cvInpaint
cvIntegral
cvPyrMeanShiftFiltering
cvThreshold
cvWatershed

Motion Analysis and Object Tracking
cvAccumulate
cvAccumulateProduct
cvAccumulateSquare
cvAccumulateWeighted
cvBackgroundSubtractorMOG
cvBackgroundSubtractorMOG2
cvEstimateRigidTransformAVL
cvKalmanFilter

Object Detection
cvMatchTemplate

Structural Analysis and Shape Descriptors
cvApproxPolyDP
cvArcLength
cvBoundingRect
cvConnectedComponents
cvContourArea
cvConvexHull
cvDrawContours
cvFindContours
cvFitEllipse
cvFitLine
cvHuMoments
cvIsContourConvex
cvMatchShapes_Images
cvMatchShapes_Shapes
cvMinAreaRect
cvMinEnclosingCircle
cvMinEnclosingTriangle
cvMoments
cvPointPolygonTest

Video Analysis
cvCalcGlobalOrientation
cvCalcMotionGradient
cvCalcOpticalFlowFarneback
cvCalcOpticalFlowPyrLK
cvSegmentMotion
cvUpdateMotionHistory

Data Types Reference

Primitive
Integer
Long
Real
Double
Bool
String
Array<T>

Image-related
Image
Pixel
Location
Box
Region
RegionOfInterest

Geometric2D
Point2D
Segment2D
Line2D
Circle2D
Vector2D
Path
Rectangle2D
Arc2D
CordinateSystem2D

Gemometric3D
Point3D
Segment3D
Line3D
Circle3D
Vector3D
Plane3D
Box3D
Point3DGrid
Surface

ÖZGEÇMİŞ

2010 yılında İşletme lisans eğitimini tamamlayan Harun ELKIRAN, lisans yandal olarak İstanbul Üniversitesi, Yönetim ve Bilişim Sistemleri programına devam etmektedir. 2000 yılından bugüne yüzlerce web ve mobil tabanlı yazılım geliştirme projesinde yer almıştır. Üniversiteler başta olmak üzere finans, e-ticaret, telekom, perakende ve güvenlik gibi sektörlerde ürün tasarım, yazılım ve uygulama danışmanlığı hizmeti vermektedir.

Bulut bilişimi, veri madenciliği, makine öğrenimi, derin öğrenme ve nesnelerin interneti, dinamik raporlama ve içerik yönetim sistemleri gibi alanlarda projeler üreten Harun ELKIRAN, Progress firmasının Türkiye ortaklığını yürütmektedir.

Evli ve 1 çocuk babasıdır.