Model Tabanlı Sistem Mühendisliği Tekniklerini Kullanarak Uygulama Yazılım Mimarisi Tasarlanması: Dış Işıklandırma Grubu ve Kısa Far Özelliği Üzerine Vaka Çalışması

Application Software Architecture Design via Model Based System Engineering Techniques: A Case Study on the Exterior Light Group and Low Beam Feature

Tolgahan Karayazı[1], Yiğit Burak Varol[1], Sena Koçak[1], Rafet Temel[2]

[1]FEV Türkiye
Kontrol Sistemleri, İstanbul
{karayazi,varol,kocak}@fev.com

[2]FEV Türkiye
E/E Sistemleri ve Mimari, Bursa
temel@fev.com

Özetçe

Bu çalışmada model tabanlı sistem mühendisliği teknikleri kullanılarak bir uygulama yazılım mimarisi geliştirilmiştir. Gövde kontrol modülüne ait dış ışıklandırma ve kısa far sistemi vaka olarak seçilmiştir. Mimari geliştirilirken SysML (*Systems Modelling Language*) dili kullanılmış ve gösterimler iç blok diyagramı kullanılarak gerçekleştirilmiştir. Mimari oluşturulurken ve sinyal isimlendirmelerinde AUTOSAR standardı göz önünde bulundurulmuştur. Mimari Enterprise Architect ortamında gösterilmiştir. Bu çalışma ile otomotiv alanında bir uygulama yazılımı geliştirirken izlenilmesi gereken süreçler model tabanlı sistem mühendisliği teknikleri ile entegre edilerek gösterilmiştir. Böylece, bir uygulama yazılımı geliştirmek için bir kılavuz oluşturulmuştur.

Abstract

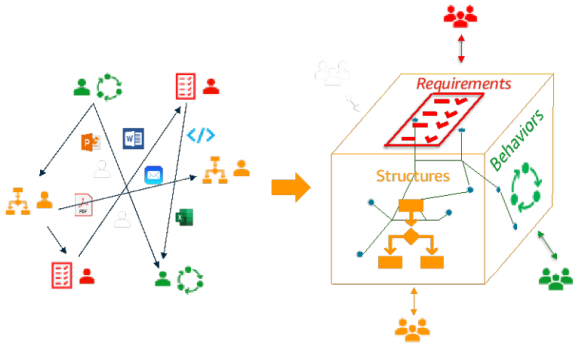
In this study, an application software architecture was developed using model based systems engineering techniques. The body control module's exterior lighting and low beam system have been chosen as the case. SysML language was used, and the representations were made using the internal block diagram while designing the architecture. The AUTOSAR standard was considered when creating the architecture and signal naming. The architecture is depicted in the Enterprise Architect environment. This study shows the procedures to be followed while developing application software in the automotive field by integrating with model based systems engineering techniques. Thus, a guideline was created for developing application software.

1. Giriş

Günümüzde mühendislik sistemlerinin daha büyük ölçekli ve karmaşık bir yapıya sahip olduğu görülmektedir. Bu sebeple; modelleme konsepti, sistem tanımlama, tasarım, analiz ve sentez gibi konular önem kazanmıştır.

Yeni bir ürün tasarlamak uzun ve yinelemeli bir süreçtir [1]. Sistem tasarım süreci, farklı ayrıntı ve kesinlik düzeylerinde bir dizi analiz ve sentez aşamasına dayanmaktadır [2]. V döngüsünün her adımı, farklı sorunları ele alan ve giderek daha ayrıntılı tasarım seçeneklerine izin veren farklı modeller kullanılmaktadır. Ön tasarım, yani V döngüsünün ilk aşamaları genellikle analitik modeller kullanırken, ayrıntılı tasarım çoğunlukla sayısal simülasyon yöntemlerini kullanılmaktadır. Bu aşamada sistem mühendisliği tekniklerinin entegrasyonu oldukça önemlidir.

Karmaşık sistemlerin varlığı Model Tabanlı Sistem Mühendisliği (MBSE) kavramının doğmasına sebep olmuştur [3]. MBSE; yalnızca bir dizi araç ve dil değil, sistem modelinin sistematik bir açıklamasıdır. MBSE, sistem mimarisi, validasyon ve verifikasyon (V&V), kantitatif ve simülasyon analizi ve sistemlerin tüm yaşam döngüsü boyunca iyileştirme gibi birçok sistem bilimi alanında uygulanmıştır [3,4].



Şekil 1. Geleneksel yaklaşım ve MBSE [5].

Uluslararası Sistem Mühendisliği Konseyi'ne (INCOSE) göre ise MBSE, "Kavramsal tasarım aşamasında başlayan ve geliştirme boyunca devam eden ve daha sonraki yaşam döngüsü aşaması boyunca süregelen sistem gereksinimleri, tasarım,

analiz, doğrulama ve doğrulama faaliyetlerini desteklemek için modellemenin resmileştirilmiş uygulamasıdır" [6].

MBSE yöntemine aşağıda sıralanan sebepler yüzünden ihtiyaç duyulmaktadır:

- Karmaşık sistemlerin yönetilebilmesi
- Analiz ve tahmin yeteneğinin arttırılması
- Tasarım kararlarının desteklenmesi
- Verifikasyonlar ve validasyon süreçlerinin desteklenmesi
- İletişim ve iş birliğinin arttırılması

Bu çalışmada, kısa far sistemi için uygulama yazılım mimarisi geliştirme süreci açıklanmıştır. Öncelikle, kullanılacak çeşitli tanımlamalar verilmiş ve arayüzler tanımlanmıştır. Daha sonra, mimari ait katmanlar belirlenmiş ve arayüzler/altsistemler/sinyaller için kısaltmalar sunulmuştur. Son olarak bu mimari için Enterprise Architect ortamında bir gösterim yapılmıştır.

2. Sistem Modelleme ve V Döngüsü

Bu çalışmada, kısa far vakası için bi uygulama yazılımı mimarisi geliştirilmesi süreci ele alınmıştır. Bu bölümde, kullanılan modelleme dili SysML ve uygulama yazılımı geliştirilirken takip edilen V döngüsü ile alakalı detay verilmiştir.

2.1. SysML Modelleme Dili

SysML, sistem mühendisliği uygulamaları için genel amaçlı bir sistem mimarisi modelleme dili ve standarttır. Karmaşık sistemlerin modellenmesi ve analizine destek olmaktadır. UML (*Unified Modelling Language*) dilinden türetilmiştir.

SysML, çok çeşitli sistemlerin ve sistem sistemlerin spesifikasyonunu, analizini, tasarımını, doğrulamasını ve onaylanmasını desteklemektedir. Bu sistemler donanım, yazılım, bilgi, süreçler, personel ve tesisleri içerebilmektedir.

Farklı sistem mühendisliği aşamalarına entegre edilebilmekte ve sistemin tüm yaşam döngüsü boyunca kullanılabilmektedir.

SysML, UML 2'nin bir diyalektidir ve UML 2 Profili olarak tanımlanır.

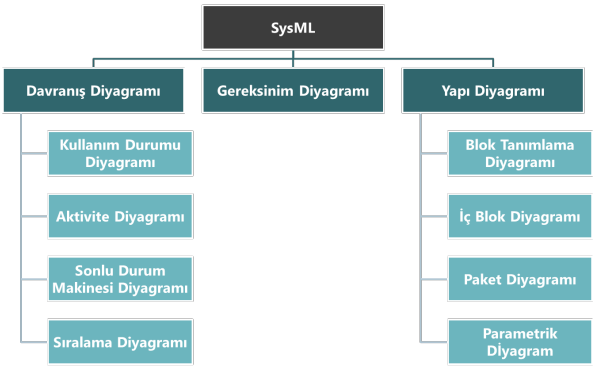
SysML, model tabanlı sistem mühendisliği (MBSE) için olanak sağlayan bir teknolojidir. SysML ilk olarak 2003 yılında SysML Ortaklarının SysML Açık Kaynak Özelliği Projesi tarafından oluşturulmuştur. SysML, 2006 yılında Object Management Group (OMG) tarafından OMG SysML olarak uyarlanmış ve benimsenmiştir [7].

SysML dışında; UML, AADL, VHDL-AMS, Verilog-A ve Modelica dilleri de sistem mühendisleri tarafından tercih edilebilmektedir.

2.1.1. SysML Diyagram Tipleri

SysML'de kullanılan diyagramlar sistemlerin farklı yönlerini modellemek ve sistemi daha iyi anlamak için güçlü bir yapı sunmaktadır. Her diyagram tipi farklı bir perspektifi temsil etmekte ve sistem mühendislerine tasarım/analiz süreçlerinde destek olmaktadır.

SysML gereksinim ve ihtiyaçlara göre faydalanılabilecek çeşitli diyagram tipleri sunmaktadır.



Şekil 2. SysML diyagram tipleri.

Bu diyagram tiplerinden tek biri tercih edilerek modelleme yapılabileceği gibi hibrid bir yaklaşım kullanılarak da bir çalışma gerçekleştirilebilir.

Diyagram tipleri ile alakalı açıklamalar aşağıdaki tabloda verilmiştir [8].

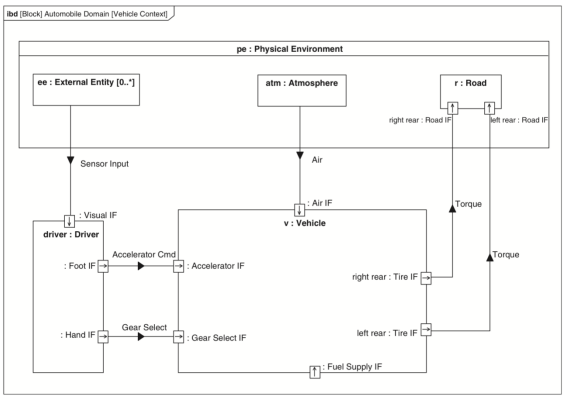
Tablo 1. SysML diyagram tiplerine ait açıklamalar.

Diyagram	Kısaltma	Kullanımı
Davranış Diyagramı		
Kullanım Durumu Diyagramı	uc	Sistem modellemenin odağını her seferinde tek bir senaryoya daraltmaktadır.
Aktivite Diyagramı	act	Sistemdeki aktivite akışını tanımlamaktadır.
Sonlu Durum Makinesi Diyagramı	smd	Sistemin veya sistemin parçalarının geçişlerini ayrık durumlar aracılığıyla tanımlamaktadır.
Sıralama Diyagramı	sd	Öğeler arasındaki mesaj akışını tanımlamaktadır.
Gereksinim Diyagramı		
Gereksinim Diyagramı	req	Sistemin ne yapması gerektiğini tanımlamaktadır. Gereksinimlerin ayrışması, izlenebilirliği ile yapının ve testlerin gereksinimlerle nasıl eşleştiğini göstermektedir.
Yapı Diyagramı		
Blok Tanımlama Diyagramı	bdd	Çeşitli türlerdeki öğelerin yapısını hiyerarşik olarak tanımlamaktadır.
İç Blok Diyagramı	ibd	Öğeler arasındaki arayüzleri ve öğe akışlarını tanımlamaktadır.
Paket Diyagramı	pkg	SysML diyagramlarını gruplamak için esnek bir yöntem sağlamaktadır.
Parametrik Diyagram	par	Sistem performansını sınırlayan denklemleri tanımlamakta ve iç parametreleri bu denklemlere bağlamaktadır.

2.1.1. İç Blok Diyagramı

Bu çalışmada, iç blok diyagramı yardımıyla modelleme yapılacaktır.

Yapısal diyagramlar sistemlerin hiyerarşilerini ve ilişkilerini açıklamak için kullanılmaktadırlar. İç Blok Diyagramı yöntemi de SysML'in yapısal Diyagramlar ana başlığı altında bulunmaktadır. Bir sistemi modellerken kullanılan blokların birbirleri ile olan ilişkilerini tanımlamak için kullanılan bir modelleme türüdür [9].



Şekil 3. İç blok diyagramı örneği [8].

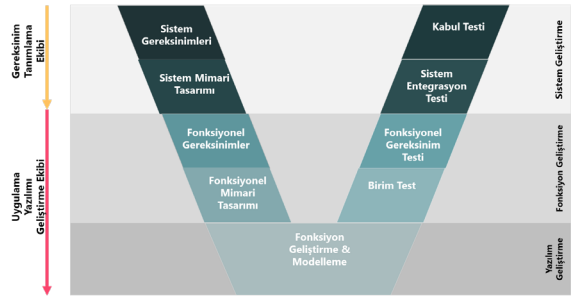
Bununla birlikte, iç blok diyagramı yaklaşımından faydalanılmıştır. Bunun sebepleri aşağıda sıralanmıştır:

- Sistemin iç yapısını görev ve parçlarını değerlerndirerek gösterilmesini sağlamaktadır.
- Alt modüller arasındaki bağlantı ve arayüzleri kolaylıkla göstermektedir.
- Hem iç hem de dış arayüzleri göstermektedir.

2.2. V Döngüsü

Yazılım geliştirmede, V modeli [10] yazılımın yaşam döngüsünü gösteren ve yöneten bir süreçtir. İşlem adımları, doğrusal bir şekilde aşağı doğru hareket etmemektedir. Bu yaklaşım, geliştirme yaşam döngüsünün her aşaması ile bununla ilişkili test aşaması arasındaki ilişkileri göstermektedir.

V döngüsü proje özelinde şekillenebilse de genel olarak benzer bir yapıya sahiptir. Bu çalışmada aşağıda verilen döngü takip edilmiştir.



Şekil 4. V döngüsü.

Bu çalışma "Fonksiyonel Mimari Tasarımı" aşamasının gerçekleştirilmesi üzerine kuruludur.

3. Uygulama Yazılımı Mimarisi

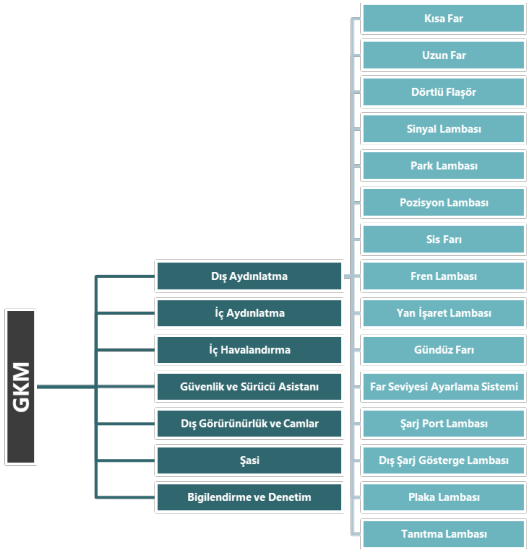
Uygulama yazılım mimarisinin oluşturulması temel olarak üç aşamada gerçekleştirilmiştir:

- Uygulama yazılım mimarisinin tanımlanması
- Arayüzlerin tanımlanması
- Bir yazılım aracı üzerinde gösterim yapılması

Bu bölümde, mimari fonksiyonel bir şekilde açıklanmıştır. Mimari tasarımında AUTOSAR tarafından GKM arayüzleri üzerine hazırlanan dökümandan faydalanılmıştır [11]. Ayrıca; mimari *composition*, *component* ve *runnable* olarak ayrıştırılmıştır. Ardından, bahsedilen birimlere ait arayüzler tanımlanmıştır. Bu esnasında AUTOSAR tarafından sunulan isimlendirme standardı kullanılmıştır [12]. Son olarak, fonksiyonel mimarinin Enterprise Architect üzerinde gösterimi yapılmıştır.

3.1. Uygulama Yazılım Mimarisinin Tanımlanması

Mimariye ait tanımlamayı yapmadan önce öncelikle GKM'e ait altsistemler gösterilmiştir.



Şekil 5. GKM'ye ait alt sistemler.

Sistem tasarımcısı öncelikle temel bir mimari belirlemelidir. Bu mimarinin yazılan gereksinimler ile uyumlu olması ve bu gereksinimleri kapsaması gerekmektedir. Bununla birlikte, bir standart veya kılavuzu takip etmek yönlendirici olabilmektedir. Sunulan vaka analizinin otomotiv uygulaması olması sebebiyle AUTOSAR standartları takip edilecektir. AUTOSAR, aşağıdaki verilen özellikleri sunan belirli bir mimarinin takip edilmesini tavsiye etmektedir:

- Sistemde yer alan karmaşıkları yönetebilen
- Yazılım değişikliği, yükseltme ve güncelleme esnekliğini sağlayabilen
- Yazılım hatları içinde ve genelinde çözümlerin ölçeklenebilirliğini sağlayabilen
- Sistemin kalitesini ve güvenilirliğini garanti edebilen
- İşlevselliğe göze alarak yazılım izolasyonu sunabilen

Mimari ile alakalı detay vermeden önce bazı tanımlar yapılması gerekmektedir.

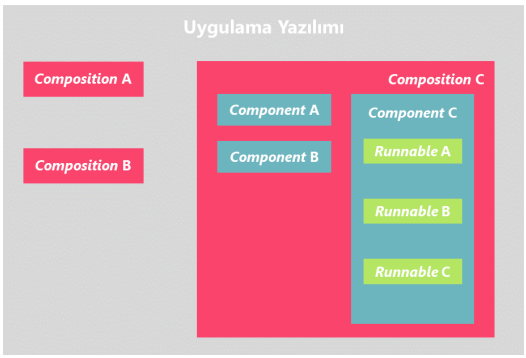
3.1.1. Tanımlar

Aşağıda mimari oluşturulurken kullanılan çeşili tanımlamalar verilmiştir:

Composition: İlgili yazılım *component* gruplarını bir araya getiren bir mimari birimidir. Hiyerarşik olarak en kapsayıcı ve büyük modüldür. Örnek; *Composition* Dış Aydınlatma, *Composition* Kısa Far.

Component: Bir veya birden fazla algoritmayı içeren modüldür. Doğru şekilde tanımlanmış bağlantı noktaları aracılığıyla ortamıyla iletişim kurmaktadır. Örnek; *Component* Kısa Far *Adapter*.

Runnable: Mimari içerisindeki en küçük yapı birimidir. Bağımsız olarak programlanabilmekte ve çalıştırılabilmektedir. Örnek; *Runnable* Aktivasyon Talep Et. Aşağıda *composition*, *component* ve *runnable* yapılarının hiyerarşisi verilmiştir.



Şekil 6. Composition, *component* ve *runnable* yapıları.

Manager: İlgili *component* biriminin işlevsel olarak çekirdeğini oluşturmaktadır. Tekrar yapılandırılabilir olması tavsiye edilmektedir. *Manager* yapısı standartlandırabilmeye müsaade etmelidir. Ayrıca, orijinal ürün üreticilerine özgü herhangi bir senaryoya uyarlanması da mümkün olmalıdır. Örnek; Kısa Far *Manager*

Adapter: Bu yapı yazılım *component*leri arasındaki veri akışının birleştirilmesi, önceliklendirileri, saklanması ve geri çağrılmasından sorumludur. Ayrıca veri yönetimi ve kontrol stratejisini temsil etmektedir. Örnek; Kısa Far *Adapter*

Actuator: *Manager* biriminden gereken talepleri yerine getirmekte ve talep durumuna bağlı olarak ilgili *compositiona* ait statü bilgisini hesaplamaktadır. Gerekli durumlarda, statü bilgili tekrar *manager* yapısına iletmektedir. Bu statü bilgisinin yanı sıra mod bilgisi de üretebilmektedir. İhtiyaç durumuna göre ayrıştırılabilir olması beklenmektedir. Örnek; Kısa Far *Actuator*

HMI: Kullanıcı isteklerini toplamaktan sorumlu tüm sensör *component*lerini içermektedir. Filtreleme, koşullandırma ve arıza tespiti gibi işlevsellikleri kapsamaktadır. Örnek: Kısa Far *HMI*

Arayüz: Arayüzler bilgi alışverişini sağlamaktadır. Örnek: LoBmReqd

***Interdomain Interfaces* (IDI):** *Composition* ve/veya *component* arası bilgileri kapsamaktadır. Vites pozisyonu, araç durumu veya enerji yönetimi gibi verilerinin akışı sağlamaktadır. Çift yönlü veri akışına imkan vermektedir.

***Application Level Error Management* (ALEM):** Uygulama yazılımı kapsamında hata yönetimi yapan birimdir.

3.1.2. Uygulama Yazılım Mimarisinin GKM ve Kısa Far için Tanımlanması

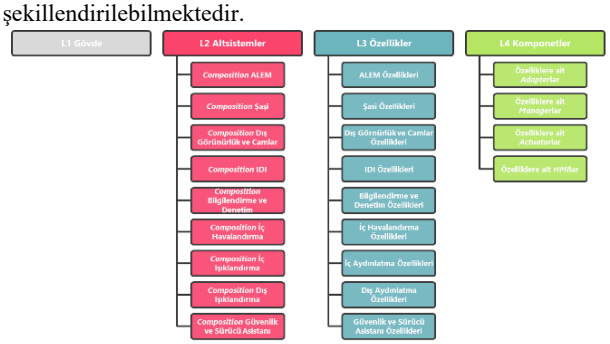
Bu vaka çalışmasında aşağıdaki hiyerarşi gözetilmiştir:

- GKM bir *composition*
- Dış Aydınlatma, İç Aydınlatma, Şasi, Güvenlik ve Sürücü Asistanı, Dış Görünürlük ve Camlar, Bilgilendirme ve Denetim bir *composition*
- Kısa Far, Uzun Far, Dörtlü Flaşör, Sağ/Sol Sinyal Lambası, Park Lambası, Yan İşaret Lambası, Pozisyon Lambası, Gündüz Farı, Arka/Ön Tanıtma Lambası, Sis Lambası, Fren (ve Adaptif) Lambası, Far Seviyesi Ayarlama Sistemi, Şarj Port Lambası, Dış Şarj Gösterge Lambası bir *composition*
- Kısa Far *Adapter*, Kısa Far *Manager*, Kısa Far *HMI*, Kısa Far *Actuator* bir *component*
- Giriş Sinyallerini İşle, Aktivasyon Talep Et, Aktivasyonu İşle, Hata Gönder, Talep Gönder bir *runnable*

Buna göre hiyerarşi aşağıdaki gibi görselleştirilmiştir.

Şekil 7. GKM uygulama yazılımı hiyerarşik yapısı.

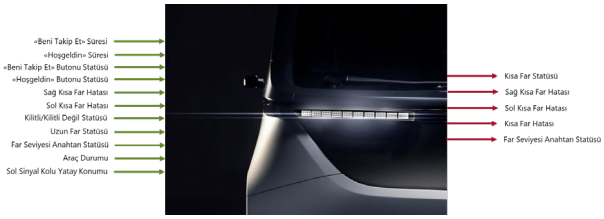
Ayrıca mimari yapısı aşağıdaki gibi şekillendirilebilmektedir.



Şekil 8. GKM mimarisinin sınıflandırılması/seviyelendirilmesi.

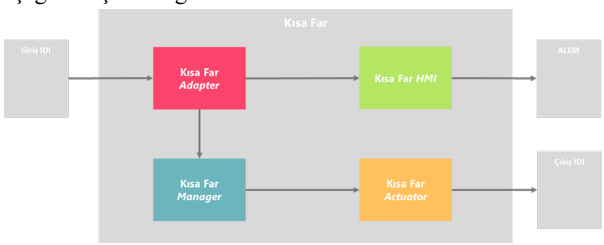
3.2. Arayüzlerin Tanımlanması

Sistem mühendisleri tarafından belirlenen isterleri ve kısa fara ait işlevsilliği kapsayacak arayüzler temel olarak aşağıdaki şekilde tanımlanmıştır.



Şekil 9. Kısa far arayüzleri

Buna göre kısa far için *component*ler arası veri akışı aşağıdaki şekilde gösterilebilmektedir.



Şekil 10. Kısa far veri akışı.

Yukarıdaki veri akışını detaylandırmak için aşağıdaki tablolar oluşturulmuş. İlk tabloda *component*e ait katmanlar ve *component*in içerdiği *runnable*lar verilmiştir.

Tablo 2. Kısa far katmanları.

Component	Kısaltma	*Katman*	*Composition* 1	*Composition* 2	*Composition* 3	*Runnable(s)*
Kısa Far *Adapter*	LoBmAdpr	APSW	BCM	ExtrLi	LoBm	LoBmAdpr_ProcInp
Kısa Far *Manager*	LoBmMgr	APSW	BCM	ExtrLi	LoBm	LoBmMgr_ReqActvn
Kısa Far *Actuator*	LoBmActr	APSW	BCM	ExtrLi	LoBm	LoBmActr_ExcActvn
Kısa Far *HMI*	LoBmHMI	APSW	BCM	ExtrLi	LoBm	LoBmHMI_SndErr LoBmHMI_SndReq

Ardından, bir componente giren ve çıkn sinyaller gösterilmiştir. Sinyal isimlendirmesinde AUTOSAR standardı kullanmış ve (standart gereği isimlendirmeler sinyallerin İngilizce karşılıklarına göre yapılmıştır.

Tablo 3. Kısa far arayüzleri ve isimlendirmeleri.

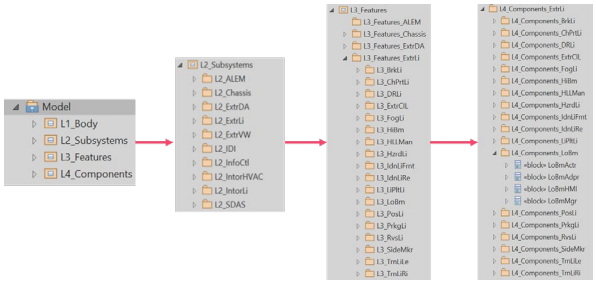
Component	Kısaltma	Giriş	Çıkış	Giriş - Kısaltma	Çıkış - Kısaltma
Kısa Far *Adapter*	LoBmAdpr	Uzun Far Statüsü Araç Durumu Sağ Kısa Far Hatası Sol Kısa Far Hatası Far Seviyesi Anahtarı Statüsü Sol Sinyal Kolu Yatay Konumu Kilitli/Kilitli Değil Statüsü «Beni Takip Et» Butonu Statüsü «Hoşgeldin» Butonu Statüsü «Beni Takip Et» Süresi «Hoşgeldin» Süresi	Hata Uzun Far İndikatörü Kısa Far Talep Edildi «Beni Takip Et» Talep Edildi «Hoşgeldin» Talep Edildi Reset Bayrağı Geçerli Araç Durumu Sağ Kısa Far Hatası Sol Kısa Far Hatası	HiBmActr_stHiBm InpIDI_stVeh InpIDI_bLoBmRiFailr InpIDI_bLoBmLeFailr InpIDI_stHdlampSwt InpIDI_stStlkLeHozlPos InpIDI_bLock InpIDI_bFolwMeBtn InpIDI_bWelcmBtn InpIDI_tiFolwMe InpIDI_tiWelcm	LoBmAdpr_bDfct LoBmAdpr_bHiBmIndcr LoBmAdpr_bLoBmReqd LoBmAdpr_bFolwMeReqd LoBmAdpr_bWelcmReqd LoBmAdpr_bRstFlg LoBmAdpr_bVehStVld LoBmAdpr_bLoBmRiFailr LoBmAdpr_bLoBmLeFailr
Kısa Far *Manager*	LoBmMgr	Uzun Far İndikatörü Kısa Far Talep Edildi «Beni Takip Et» Talep Edildi «Hoşgeldin» Talep Edildi Reset Bayrağı Geçerli Araç Durumu	Kısa Far Aktivasyon Statüsü	LoBmAdpr_bVehStVld LoBmAdpr_HiBmIndcr LoBmAdpr_bLoBmReqd LoBmAdpr_bFolwMeReqd LoBmAdpr_bWelcmReqd LoBmAdpr_bRstFlg	LoBmMgr_bLoBmActvn
Kısa Far *Actuator*	LoBmActr	Kısa Far Aktivasyon Statüsü Hata	Kısa Far Statüsü	LoBmMgr_bLoBmActvn LoBmAdpr_bDfct	LoBmActr_stLoBm
Kısa Far *HMI*	LoBmHMI	Hata Sağ Kısa Far Hatası Sol Kısa Far Hatası Far Seviyesi Anahtarı Statüsü	Hata Sağ Kısa Far Hatası Sol Kısa Far Hatası Far Seviyesi Anahtarı Statüsü	LoBmAdpr_bDfct LoBmAdpr_bLoBmRiFailr LoBmAdpr_bLoBmLeFailr InpIDI_stHdlampSwt	LoBmHMI_bDfct LoBmHMI_bLoBmRiFailr LoBmHMI_bLoBmLeFailr LoBmHMI_stHdlampSwt

3.2. Enterprise Architect Üzerinde Gösterim

Bu çalışmada Enterprise Architect üzerinde uygulama yazılım mimarisi mimarisini oluşturulmuştur. Sparx Systems Enterprise Architect, OMG UML'ye dayalı bir görsel modelleme ve tasarım aracıdır. Yazılım sistemlerinin tasarımı, iş süreçlerinin modellenmesi ve endüstri tabanlı sistemlerin modellenmesi gibi işlevlere sahiptir. İşletmeler ve kuruluşlar tarafından yalnızca sistemlerinin mimarisini modellemek için değil, aynı zamanda bu modellerin yaşam döngüsü boyunca uygulanma süreçlerini gözlemlemek amacıyla kullanmaktadır [13].

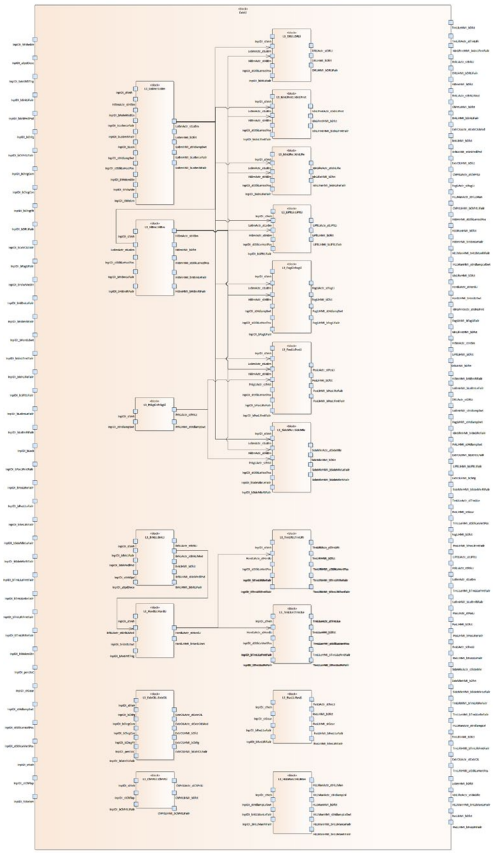
GKM ve kısa far mimarisini betimlerken modelleme dili olarak SysML tercih edilmiştir.

Öncelikle programda kullanılan dosyalama yapısı gösterilmiştir. Burada bağlantılı tüm dosyalar ve yapılar linklenmiştir.



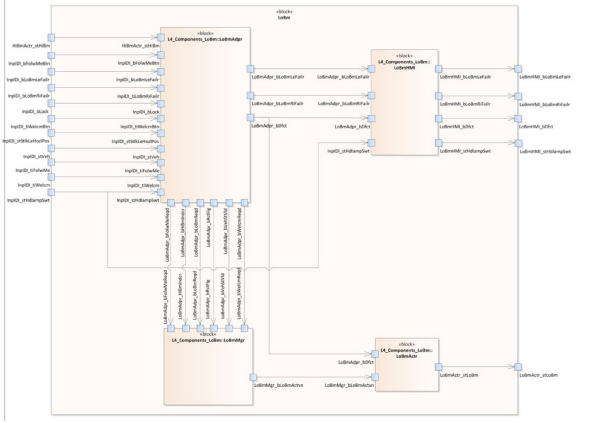
Şekil 11. Uygulama yazılım mimarisi dosyalama yapısı.

Ardından dış ışıklandırma grubuna ait iç blok diyagramı verilmiştir.



Şekil 12. Dış ışıklandırma grubuna ait iç blok diyagramı.

Daha sonra kısa far yapısına ait iç blok diyagramı sunulmuştur.



Şekil 13. Kısa fara ait iç blok diyagramı

4. Sonuçlar

Bu çalışmada MBSE metodolojisine dayalı bir uygulma yazılımı mimarisi geliştirilmiş ve MBSE konsepti tanıtılmıştır. GKM ve kısa far vaka olarak incelenmiştir. İlk olarak, MBSE konsepti ve SysML hakkında bilgi verilmiştir.

Uygulama yazılım mimarisi, AUTOSAR standardı dikkate alınarak oluşturulmuştur ve sinyal/modül isimlendirmelerinde de bu standart dikkate alınmıştır.

Bunun yanı sıra, uygulama yazılım mimarisi Enterprise Architect ortamında gösterilmiş ve gerekli linklemeler yapılmıştır. Bahsedilen gösterimin GKM kapsamındaki tüm özellikler için tekrarlanması gerekmektedir.

Böylece otomotiv alanında bir uygulama yazılımı geliştirirken izlenilmesi gereken süreçler gösterilmiştir. Ayrıca, bu süreçte MBSE teknikleri gözetilmiştir.

Gelecek çalışma olarak, test senaryoları ve gereksinimleri de Enterprise Architect ortamına entegrasyonu ile izlenebilirlik sağlanabilecektir. Ayrıca, *runnable*ların iç davranışları aktivite diyagramları ile modellenerek hibrid bir yaklaşım sunulacaktır.

Kaynakça

[1] N.F.M. Roozenburg, J. Eekels, Product Design: Fundamentals and Methods, Wiley, West Sussex, 1995.

[2] S. Diampovesa, A. Hubert, P.-A. Yvars, Designing physical systems through a model-based synthesis approach. example of a Li-ion battery for electrical vehicles, Comput. Ind. 129 (2021) 103440, http://dx.doi.org/10.1016/j.compind. 2021.103440.

[3] J.A. Estefan, Survey of model-based systems engineering (MBSE) methodologies, Incose MBSE Focus Group 25 (8) (2007) 1–12.

[4] T. Huldt, I. Stenius, State-of-practice survey of model-based systems engineering, Syst. Eng. 22 (2) (2019) 134–145.

[5] Model Based Systems Engineering (MBSE) on AWS: From Migration to Innovation (2023) https://docs.aws.amazon.com/ (Accessed: 26 July 2023).

[6] D.D. Walden, G.J. Roedler, K. Forsberg, R.D. Hamelin, T.M. Shortell, Systems engineering handbook: A guide for system life cycle processes and activities, John Wiley & Sons, 2015.

[7] SysML FAQ: What is SysML? (no date) SysML.org. Available at: https://sysml.org/sysml-faq/what-is-sysml.html (Accessed: 28 July 2023).

[8] L. Zdanis and R. Cloutier, "The Use of Behavioral Diagrams in SysML," 2007 IEEE Long Island Systems, Applications and Technology Conference, Farmingdale, NY, USA, 2007, pp. 1-1, doi: 10.1109/LISAT.2007.4312634.

[9] Friedenthal, S., Moore, A. and Steiner, R. (2011) in Practical guide to SysML. Morgan Kaufmann.

[10] Kevin Forsberg and Harold Mooz, "The Relationship of System Engineering to the Project Cycle", in Proceedings of the First Annual Symposium of National Council on System Engineering, October 1991: 57–65.

[11] Explanation of Application Interfaces of the Body and Comfort Domain (2022) https://www.autosar.org. Available at: Explanation of Application Interfaces of the Body and Comfort Domain (Accessed: 26 July 2023).

[12] Requirements on SW-C and System Modeling (2021) https://www.autosar.org. Available at: Requirements on SW-C and System Modeling (Accessed: 26 July 2023).x

[13] *Enterprise architect*. Sparx Systems. (n.d.). https://sparxsystems.com/products/ea/index.html