AYGIT AĞACI ÜRETİMİNİN AZ-KODLU GELİŞTİRME YAKLAŞIMIYLA KOLAYLAŞTIRILMASI

Mehmet Samet Hakut
Ege Üniversitesi, Uluslararası Bilgisayar Enstitüsü
msamet.hakut@proton.me, ORCID: 0000-0001-5700-4692

Prof. Dr. Geylani Kardaş
Ege Üniversitesi, Uluslararası Bilgisayar Enstitüsü
geylani.kardas@ege.edu.tr, ORCID: 0000-0001-6975-305X

Özet
Gömülü sistem yazılımlarının geliştirilmesinde sıklıkla kullanılan aygıt ağaçlarının (DT) mevcut programlama dillerinden farklı bir söz dizime sahip olması nedeniyle gömülü yazılım geliştiricileri güçlüklerle karşılaşabilmektedir. Birçok alanda kullanılan model-güdümlü mühendislik ve bunu destekleyen az-kodlu geliştirme yöntemleri DT geliştirmede karşılaşılan zorlukları azaltmaya yardımcı olabilir. Bu noktadan hareketle bu çalışmada gömülü sistemlerde DT yazılımlarının az-kodlu geliştirilmesini sağlayan LCDP4DT adlı bir yazılım geliştirme platformu tanıtılmaktadır. Kullanıcıların görsel olarak DT konfigürasyonlarını oluşturabilecekleri bir ortam için öncelikle bir üst model ve bunu kullanan sözdizim tanımları yapılmıştır. Farklı bakış açılarından DT modellerinin tanıtılan bu görsel sözdizim ile oluşturulması ve bu modeller üzerinden gerekli DT yazılımlarının otomatik üretilmesi süreci de yine bildiride bir durum çalışması üzerinden örneklenmiştir. Kullanıcılara çevrim içi ve işbirlikçi olarak DT geliştirme imkânı sunan LCDP4DT'nin, mevcut diğer yaklaşımlar ile farkları ve geliştirme süresini kısaltma, hata oranını düşürme ve bakım kolaylığı sağlama potansiyeli bildiride tartışılmıştır.

Anahtar Kelimeler: LCDP4DT, Aygıt ağacı, Az-kodlu geliştirme, Model-güdümlü mühendislik

FACILITATING DEVICE TREE GENERATION WITH A LOW-CODE DEVELOPMENT APPROACH

Abstract
Embedded software developers may encounter difficulties during the preparation of the device trees (DTs) frequently used in embedded system software development since DTs have a syntax that is different from existing programming languages. Model-driven engineering and low-code development methods that support it, used in many areas, can help reduce the difficulties encountered in DT development. Hence, this study introduces a software development platform called LCDP4DT that enables low-code development of DT software in embedded systems. First, a metamodel and the syntax definitions that use it were made for an environment where users can visually create DT configurations. The process of creating DT models from different viewpoints with this introduced graphical syntax and automatically generating the necessary DT software from these models is also exemplified in the paper through a case study. LCDP4DT, which offers users the opportunity to develop DTs online and collaboratively, is discussed in the paper, along with its differences from other existing approaches and its potential to shorten development time, reduce error rates, and provide ease of maintenance.

Keywords: LCDP4DT, Device tree, Low-code development, Model-driven engineering

1. GİRİŞ

Gömülü sistemler, kişisel bilgisayarlardan farklı olarak, belirli ve önceden tanımlanmış görevleri yerine getirmek üzere tasarlanmış, genellikle sınırlı işlemci gücü, bellek ve diğer çevre birimlerine sahip sistemlerdir. Bu sistemler, cep telefonlarından araç takip cihazlarına, ağ ekipmanlarından motor kontrol ünitelerine, fren sistemlerinden ev otomasyon ürünlerine, hava savunma sistemlerinden tıbbi cihazlara ve ölçüm sistemlerine kadar geniş bir yelpazede uygulama alanı bulmaktadır (Saritas & Kardas, 2014).

Günümüzde, teknolojik ilerlemelerle birlikte bu sistemlerde kullanılan donanım çevre birimlerinin çeşitliliği sürekli artmakta, aynı zamanda işletim sistemleri ve çekirdekleri de sık sık güncellenmektedir. Bu dinamik ortam, donanım ve yazılım arasındaki etkileşimi karmaşıklaştırmaktadır. Geleneksel olarak, işletim sistemi çekirdekleri, tüm çevre birimlerinin sürücü dosyalarını doğrudan kendi kaynak kodlarında barındırmaktadır (Likely & Boyer, 2008). Bu durum, saat sinyali frekansı, pin adları veya genel olarak kullanılan çipler gibi donanım bilgilerinde yapılacak herhangi bir değişiklik için çekirdek kodunun manuel olarak değiştirilmesini ve tekrardan derlenmesini gerektirmektedir. Bu süreç hem yüksek bakım maliyetleri yaratmakta hem de uzun geliştirme sürelerine yol açmaktadır (Likely & Boyer, 2008). Bu zorlukları aşmak amacıyla Aygıt Ağacı (ing. Device Tree) (DT) yaklaşımı geliştirilmiştir. DT, donanım bilgilerini işletim sistemi çekirdeğinden ayırarak, çekirdeğin daha genel amaçlı bir yapıya sahip olmasını sağlamıştır (Devicetree Community, 2025). Bu sayede, çevresel işlemler çekirdek kaynak koduna müdahale etmeden gerçekleştirilebilmektedir. Gömülü sistemlerin sürekli evrimi, donanım çeşitliliğinin ve işletim sistemi karmaşıklığının artmasıyla belirginleşmektedir. Bu durum, Aygıt Ağaçları gibi soyutlama mekanizmalarına olan ihtiyacı daha da tetiklemektedir. Bu, sadece teknik bir tercih olmanın ötesinde, artan sistem karmaşıklığını yönetmek ve geliştirme yükünü azaltmak için stratejik bir yanıt olarak değerlendirilmelidir. Artan donanım çeşitliliği ve hızlı yazılım evrimi, geliştirme ve bakım süreçlerinde darboğazlara yol açmaktadır. DT, donanım özelliklerini çekirdekten ayırarak bu karmaşıklığı yönetmeye yardımcı olan temel bir soyutlama katmanı sunmaktadır (Madieu, 2017). Bu yaklaşım, gömülü yazılım geliştirme alanındaki ölçeklenebilirlik sorunlarına yönelik temel bir mimari çözüm getirmektedir (Arslan & Kardaş, 2018).

Yazılım geliştirme paradigmaları, geleneksel kodlama yaklaşımlarından daha yüksek soyutlama seviyelerine doğru önemli bir değişim geçirmektedir. Bu değişimin merkezinde, bilgisayar programlarını daha hızlı, daha verimli ve minimum maliyetle geliştirmeyi hedefleyen Model-Güdümlü Mühendislik (ing. Model-driven Engineering) (MDE) yer almaktadır (Brambilla et al., 2022). MDE, yazılımın temel yapı taşlarını modeller aracılığıyla ifade etmeyi ve bu modellerden otomatik olarak kod üretmeyi amaçlar.

Bu bağlamda, az-kod geliştirme (ing. low-code development) (LCD) modern yazılım mühendisliğinde giderek artan bir önem kazanmaktadır (Cabot, 2020). Az-kod, manuel kod yazma miktarını azaltarak yazılım geliştirmeyi basitleştirmeyi amaçlayan ve genellikle görsel bileşenlerle çevrim içi olarak yazılım geliştirmeye imkân veren bir yaklaşımdır. Bu yaklaşım, geliştirici olmayan ancak alan uzmanı olan kişilerin bile yazılım geliştirme süreçlerine katkıda bulunmasını sağlayarak, geliştirme ekiplerinin verimliliğini artırmaktadır. MDE ve az-kodun benimsenmesi sadece verimlilikle ilgili bir tercih değildir; aynı zamanda modern yazılım sistemlerinin doğal karmaşıklığını soyutlama seviyesini yükselterek yönetmeye yönelik stratejik bir değişimi ifade etmektedir (Di Ruscio et al., 2022). Temel fayda sadece geliştirme hızı değil, karmaşıklığın ele alınış biçiminde daha köklü bir dönüşümdür (Arslan et al., 2023). Kod detaylarını görsel modellere soyutlayarak, MDE ve az-kod, birçok yazılım uygulama alanı için artan karmaşıklığı daha etkili bir şekilde yönetme yeteneği sağlamaktadır (Bock & Frank, 2021).

Gömülü sistem yazılım geliştiricileri genellikle metin tabanlı ve mevcut programlama dillerinin sözdiziminden farklı olan DT kaynak dosyalarını kullanmakta güçlük çekmektedirler (Arslan & Kardas, 2020). Aygıt ağacı var olmadan söz konusu cihazların yalnızca Linux çekirdeğine bağlı olarak çalışmadığı ve aygıt ağacı geliştirme için bir modelleme dilinin var olmayışı göz önüne alınırsa geliştiriciler bu söz dizimini de öğrenmek zorunda kalmakta ve bu durum yazılım geliştirmede yavaşlamaya sebep olmaktadır. LCD'nin yukarıda değinilen yazılım geliştirmeye getirdiği kolaylıktan DT konfigürasyonlarının geliştirilmesinde de yararlanılabilir. DT kodlamanın getirdiği bilişsel yük ve hata yapma eğilimi LCD ile azaltılabilir. Ayrıca düşük seviyeli uygulama detayları yerine, alanın temel mantığına odaklanmayı mümkün kılacağından yazılım geliştirme bilgi ve becerisi kısıtlı gömülü sistem uzmanlarının da daha kolay bir şekilde DT uygulamalarını geliştirmesine LCD imkân verebilir. Bu noktadan hareketle bu bildiride DT'lerin MDE'sini kolaylaştıracak ve kısaca LCDP4DT adı verilen yeni bir az-kodlu geliştirme platformu (ing. low-code development platform) (LCDP) tanıtılmaktadır.

Bu bildiride tanıtılan LCDP4DT daha önce DT'lerin MDE teknikleri ile geliştirilmesine imkân veren ve (Arslan & Kardas, 2020)'de anlatılan DSML4DT isimli bir alana-özgü modelleme dilini temel

almaktadır. (Arslan & Kardas, 2020)'de DT'ler için tanımlanan model elemanları ve ilişkiler bu bildiride anlatılan üst modelde güncellenmiş ve DSML4DT'nin bakış açılarındaki düğüm temsilleri kullanımlarını kolaylaştırmak amacıyla birleştirilmiştir. DSML4DT kullanırken DT uygulamalarını sadece masaüstü ve çevrim dışı geliştirme zorunluluğu LCDP4DT'de ortadan kaldırılmıştır ve geliştiricilerin herhangi bir yerden ve herhangi bir cihazdan DT modellerini çevrim içi oluşturmasına ve yönetmesine olanak tanınmıştır. Böylece geliştiricilerinin bilgisayarları üzerinde bir entegre geliştirme ortamını (IDE) ayrıca kurmasına gerek kalmaz ve birden fazla geliştirici ağ üzerinden aynı anda DT geliştirme yapabilir. Bu özellikler geliştiriciler arasında iş birliği ve kaynak tasarrufu sağlamaktadır.

Bildirinin ikinci bölümünde ilgili çalışmalardan bahsedilmiştir. Üçüncü ve dördüncü bölümlerde LCDP4DT'nin dil yapısında bulunan soyut ve somut sözdizimler anlatılmaktadır ve DT modellerinin doğruluğuna yardım eden model kısıt kontrolleri gösterilmektedir. Altıncı bölümde önerilen platformun kullanımı bir durum çalışması üzerinden örneklenmiştir. Yedinci ve son bölümde sonuçlar ve ileriye yönelik çalışmalar verilmiştir.

2. İLGİLİ ÇALIŞMALAR

Gömülü sistemlerde donanım-yazılım ayrımını sağlamak amacıyla geliştirilen DT yapısı, Linux çekirdeğinde gömülü donanım tanımını soyutlamak için kullanılmaktadır. Bu konuda (Likely & Boyer, 2008) tarafından sunulan çalışma, DT yapısının tarihsel gelişimini ve gömülü donanımı tanımlamadaki rolünü kapsamlı bir şekilde açıklamaktadır. (Simmonds, 2015) ise, gömülü Linux sistemlerde DT'nin pratik kullanımını detaylı örneklerle ele almaktadır. Literatürde DT'ler ve DT-tabanlı konfigürasyonların örneğin Linux dosya sistemlerinde çoklu cihaz desteği (Rodeh et al., 2013), ana kartlardaki mikroişlemcilerin paket optimizasyonları (Gioia et al., 2016), toplu taşımada sürücü terminallerinin hazırlanması (Arslan et al., 2017), etkileşimli sanal hidroelektrik üretme ekipmanlarını oluşturma (Li et al., 2018), trafik işaretlerini tanıma (Farhat et al., 2019) ve gerçek zamanlı sağlık izleme cihazı üretme (Swaroop et al., 2019) gibi amaçlar için kullanıldığı çalışmalar bulunmaktadır. Ancak bu çalışmalarda DT'lerin geliştirilmesi için MDE yaklaşımları takip edilmemiştir.

Donanım konfigürasyonlarının mantıksal yapısı (Schüpbach et al., 2012), sanal makine (Nikkel 2016), çevresel komponent arabağlantı arayüzleri (Devigne et al., 2017) ve alan programlanabilir kapı dizilerinin oluşturulması (Neuendorffer, 2018) gibi ihtiyaçlar için DT'nin kullanımını açıklayan çalışmalar bulunmaktadır. Ancak bu çalışmaların hiçbiri bu bildiride tanıtılan LCDP4DT gibi otomatik DT yazılımı üretimini desteklememektedir. (Jassi et al., 2016)'daki çalışmada, donanım sürücü kodu oluşturmak için DT üretiminin yapıldığı belirtilmiş, ancak yöntem, kapsam ve deneysel sonuç bölümlerinde DT üretimi ile ilgili bir bilgi verilmemiştir. Ayrıca (Jassi et al., 2016) çalışmasında Evrensel Seri Veriyolu, Seri Çevresel Arayüz vb. birçok farklı arayüz için bileşenlerin MDE'si yoktur. DT'lerin MDE'si için (Arslan & Kardaş, 2018) ve (Arslan & Kardas, 2020)'de model-güdümlü DT geliştirme ihtiyacı vurgulanmış ve DSML4DT isimli bir alana-özgü modelleme dili tanıtılmıştır. Bu çalışmalarda grafiksel ortamda aygıt ağaçlarının modellenmesi ve otomatik kod üretimi mümkün hale getirilmiştir. Ancak bildirinin ilk bölümde de açıklandığı üzere DSML4DT'de Core, SoC, Aips Bus, Spba Bus gibi DT bakış açıları için yazılım modelleme ayrı ayrı yapılmaktadır ve modeller arası geçişler için kullanıcının ek işlemler yapması gerekmektedir. Ayrıca DSML4DT'nin sunduğu IDE sadece masaüstündedir; çevrim içi ve işbirlikçi geliştirmeyi desteklemez. Bu bildiride tanıtılan LCDP4DT sunduğu geliştirilmiş üst model ve web tabanlı ortam ile DSML4DT'nin bu eksikliklerini gidermektedir.

3. SOYUT SÖZDİZİM

Modelleme dillerinin soyut sözdizimi bir alandaki kavramları ve bunların ilişkilerini tanımlar. Bu sözdizim de genellikle bir üst modele bağlı olarak tanımlanır (Kardas et al., 2023). LCDP4DT'nin soyut sözdizimi de bir üst model (metamodel) ile oluşturulmuştur ve bu üst model kolay ve verimli kullanım amacıyla beş farklı mantıksal bakış açısına (viewpoint) bölünmüştür: Core, SoC, Aips_Bus, Spba_Bus ve Peripheral. Bu bakış açıları, DT standartlarındaki elemanların ilişkilerine göre ayrılmıştır. Bu gruplandırma, (Arslan & Kardas, 2020)'de tanıtılan ve LCDP4DT'nin de temel aldığı sözdiziminin geliştirilmesini kolaylaştırmakta ve DT geliştiricileri tarafından daha rahat yorumlanmasını ve daha etkin kullanılmasını sağlamaktadır. Örneğin, belirli LCDP4DT bakış açılarının güncellenmesi, tek bir

bakış açısı kullanımına ve büyük bir model üzerinde çalışmaya kıyasla çok daha kolaydır. Ayrıca, bu çoklu görünüm yapısı, mevcut sözdiziminin gelecekteki özel ihtiyaçlar durumunda genişletilmesine olanak tanır. LCDP4DT'deki modelleme bakış açıları aşağıda kısaca anlatılmıştır ve örnek olarak Şekil 1'de Core bakış açısında yer alan üst varlıklar ve ilişkileri gösterilmiştir.

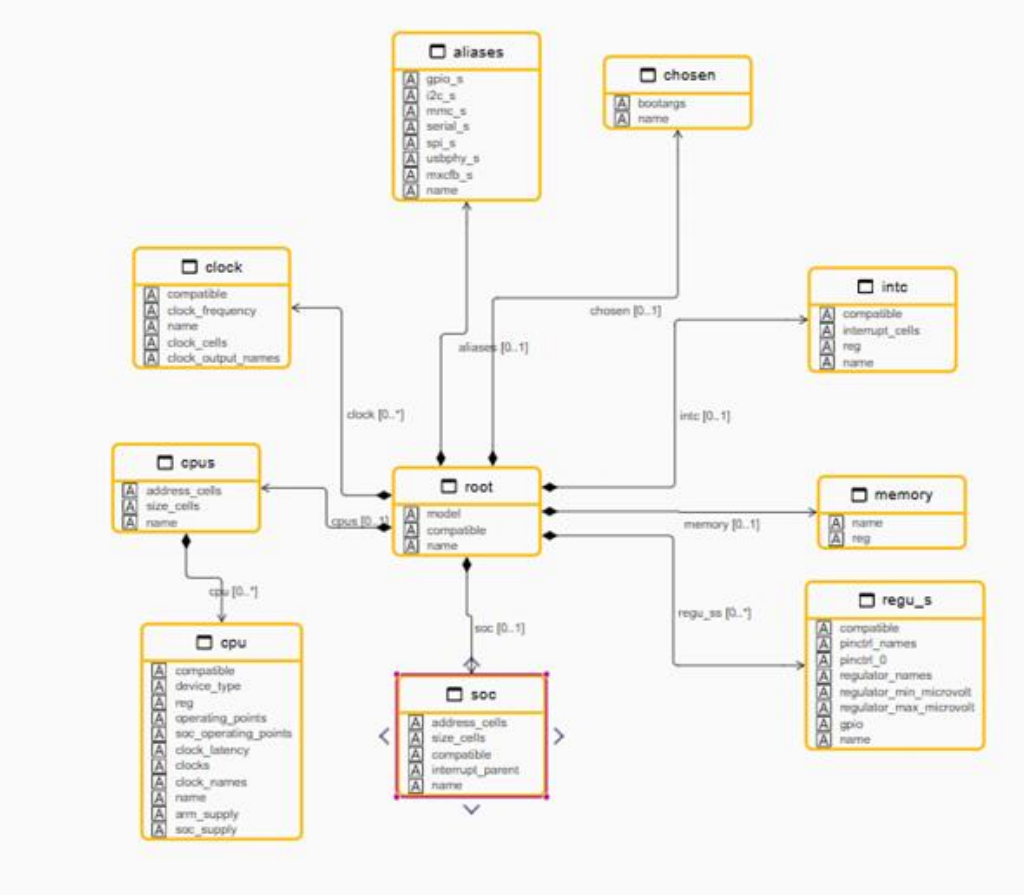
Core Bakış Açısı: Tüm sistemin türetildiği temel düğüm olan "root" elemanını içerir. Gömülü sistemin işlemci ve belleği bu bakış açısında tanımlanır. Bu bakış açsında yer alan varlıklar kullanılarak bir gömülü sistemin en temel özellikleri kurgulanabilir ve işletim sisteminin hangi tür çekirdek ve bellekle çalıştığını ve nasıl başlayacağını bilmesini sağlar. Çok çekirdekli işlemcilerde, her çekirdek birimi için gerekli türevler "cpus" öğesinden yapılır. Ayrıca, "aliases" ve "chosen" gibi tüm DT yapısında kullanılacak kısaltmalar ve önyükleme parametre geçişlerini sağlayan öğeler de burada bulunur.

SoC Bakış Açısı: SoC entegre devrelerinin özelliklerine yönelik desteği içerir. Ses, görüntü ve zamanlayıcı gibi birçok SoC özelliğinin parametre ayarları modellenebilir. Bir SoC, genellikle CPU, bellek ve diğer birçok çevre birimini içeren tek bir çip üzerindeki bir mini bilgisayar gibidir. "Aips_bus", "gpmi", "ocram" bu bakış açısının temel öğeleridir.

Aips_Bus Bakış Açısı: Düşük bant genişliğine sahip SoC çevresel bileşenlerinin SoC birimleri ile iletişim kurduğu Aips Bus arayüzünü modellemeyi sağlar. Bu veriyolu, bir SoC içinde yaygın bir iletişim yoludur ve öncelikle daha düşük bant genişliğine sahip çevresel bileşenleri ana SoC birimlerine bağlamak için kullanılır. "caam", "iomuxc", "ldb", "usb", "fec", "i2c", "uart", "pwm", "flexcan", "gpio", "wdog", "clks", "usbphy" ve "spba_bus" gibi elemanlar bu bakış açısından üretilebilir.

Spba_Bus Bakış Açısı: Paylaşımlı Çevresel Hat Arayüzü (SPBA) veriyolunu kullanarak harici birimlerle iletişimi modellemeyi sağlar. Genellikle doğrudan SoC ana veriyoluna entegre edilmeyecek özel harici çevre birimlerini bağlamak için kullanılan başka bir veriyolu türüdür. SoC biriminin gelişmiş ses işleme gibi belirli görevleri yerine getiren bazı harici çipler veya modüllerle nasıl iletişim kuracağını ayarlamak için kritik öneme sahiptir. "spdif", "esai", "ssi" ve "ecspi" gibi birimler spba_bus elemanından üretilebilir.

Peripheral Bakış Açısı: SoC entegre devresinde olmayan ve SoC'a dışarıdan bağlı üniteleri içerir. Farklı ses ve video dönüştürücüleri gibi entegre devre çevre birimleri burada tanımlanır. Bunlar, çipin içine yerleşik olmak yerine SoC'ye fiziksel olarak dışarıdan bağlanan birimlerdir. Bu bakış açısı, DT'nin kapsamını SoC bileşeninin ötesine genişleterek, eksiksiz bir gömülü sistemi oluşturan tüm ek donanımı tanımlamaya ve yapılandırmaya olanak tanır. "clocks", "sound", "lcd" ve "backlight" gibi öğeler bu bakış açısındadır.

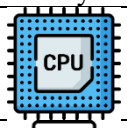
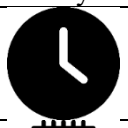
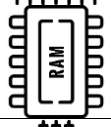
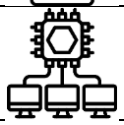
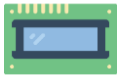
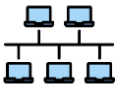


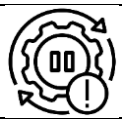
Şekil 1: Somut Sözdizim Core Bakış Açısının Soyut Sözdizimdeki Karşılığı

4. SOMUT SÖZDİZİM

Bölüm 3'te verilen DT üst modelindeki varlıklar ve ilişkilerinin örnek modeller üzerindeki temsilleri arasında bir eşleme sağlamak amacıyla görsel bir somut sözdizim tanımlanmıştır. Dilin sunumunu ve inşasını kolaylaştıran bir dizi notasyon belirlenmiştir ve bunların IDE'de içerisindeki grafiksel temsilleri ayarlanmıştır. Tablo 1'de, LCDP4DT'nin DT modelleme için içerdiği kavramlara karşılık gelen görsel notasyonların bir kısmı örnek olarak listelenmiştir. Yer kısıtları nedeniyle sadece LCDP4DT'deki temel öğelerin grafiksel gösterimleri sunulmuştur.

Tablo 1: LCDP4DT somut sözdizim kavramları ve gösterimleri

Konsept	Notasyon	Konsept	Notasyon
CPU		CLOCK	
MEMORY		SOC	
AIPS_BUS		LCD	
SPBA_BUS		SOUND	

INTERRUPT_CONTROLLER		BACKLIGHT	

LCDP4DT sözdizimi araçları, modelleme işlemi sırasında kullanıcıya tasarımda yardımcı olan bazı kısıtlamalar ve denetimler gerçekleştirir. Bu kontroller metamodelden ve görsel arayüz aracından gelmektedir. Aşağıdak alt bölümlerde aynı zamanda LCDP4DT modelleri için statik semantik sağlayan bu kısıtlama ve denetimler anlatılmaktadır.

4.1. Model Kısıtlamaları ve Doğrulama Kuralları

LCDP4DT IDE'si, Eclipse Sirius Web (Sirius-web, 2025) altyapısına dayanmaktadır ve Sirius'taki LCDP4DT Ecore modellerinden türetilen kapsamlı kısıtlamalar, tüm bakış açılarındaki örnek modeller için tutarlılık ve doğruluk sağlar. Bu kısıtlamalar, hataları önleyerek ve doğru bir DT hiyerarşisi oluşturarak modelleme sürecini büyük ölçüde kolaylaştırır.

Bu kısıtlamalar ve doğrulama kuralları şunları içerir:

- İlişki Kontrolü: Model elemanları arasındaki bileşimsel ilişkiler otomatik olarak denetlenir. Örneğin, root elemanından memory ve battery elemanlarının türetilmesi gibi hiyerarşik bağlantılar kontrol edilir. Yanlış bir bağlantı kurulması kapsama özelliğinden dolayı mümkün değildir. Kullanıcı alt düğümleri sadece ilgili üst düğüme bağlayabilir çünkü diğer düğümler kullanıcıya seçenek olarak sunulmaz.
- Eleman Sayısı Kısıtlamaları: Bir örnek modeldeki elemanlar arasındaki ilişki sayısı (bire bir, bire çok, çoktan çoğa) ve ilişkinin yönünü tanımlayan kaynak ve hedef kısıtlamaları uygulanır. Bu, belirli bir modülün yalnızca bir ana veriyoluna bağlanmasına izin verilirken, birden fazla çevre biriminin bir veriyoluna bağlanabilmesi gibi senaryoları destekler.
- Benzersizlik Kontrolü: Belirli model elemanlarının sistemde benzersiz olması gerekiyorsa, platform bunu denetler ve aynı elemandan birden fazla bulunması durumunda uyarı verir. "Tüm öğeler için sistemde birleştirme" özelliği, benzersiz öğelerin bir listeye kaydedilerek farklı bakış açılarında kullanılmasını sağlayarak bu kontrolü destekler.
- İlişki-Eleman Bütünlüğü: Bir öğe kaldırıldığında, ilgili tüm ilişkiler modelden otomatik olarak kaldırılır. Bu, modelin tutarlılığını korur ve kopuk bağlantıları engeller.
- Üst Düğüm Kısıtlaması: Her bir düğüm, içerme özelliği sayesinde bir üst düğüm olmadan oluşturulamaz. Bu kural, Aygıt Ağacının doğru ve hiyerarşik yapısının kurulmasını garanti eder.

4.2. Grafiksel Araçlar

Metamodel kısıtlamalarına ek olarak, LCDP4DT modellemesi, kullanıcıya tasarımda yardımcı olan editöre bağlı kısıtlamalar ve özellikler içerir. Bu araçlar modellemenin esnekliğini ve verimliliğini artırır:

- Farklı Bakış Açıları Arasında Geçiş: Editör, farklı bakış açılarının editörleri arasında geçiş yapma imkânı sunarak sistemin adım adım ve modüler bir şekilde oluşturulmasını sağlar.
- Kod Üretim Komut Dosyası: Geliştirilen kod üretim komut dosyası sayesinde meta-model ve model eşlemeleri yapılarak modeldeki düğümlerin doğru DT düğümlerine çevrilmesi sağlanır. Bu, modelden kod üretme sürecini otomatize eder.

Bu tür statik model kontrolleri, geliştirme sürecindeki hataları erken aşamada tespit ederek hem zaman kazandırır hem de üretilen kodun kalitesini artırır. Geliştirilen kod üretim komut dosyası sayesinde meta-model ve model eşlemeleri yapılarak modeldeki düğümlerin doğru DT düğümlerine çevrilmesi sağlanmaktadır.

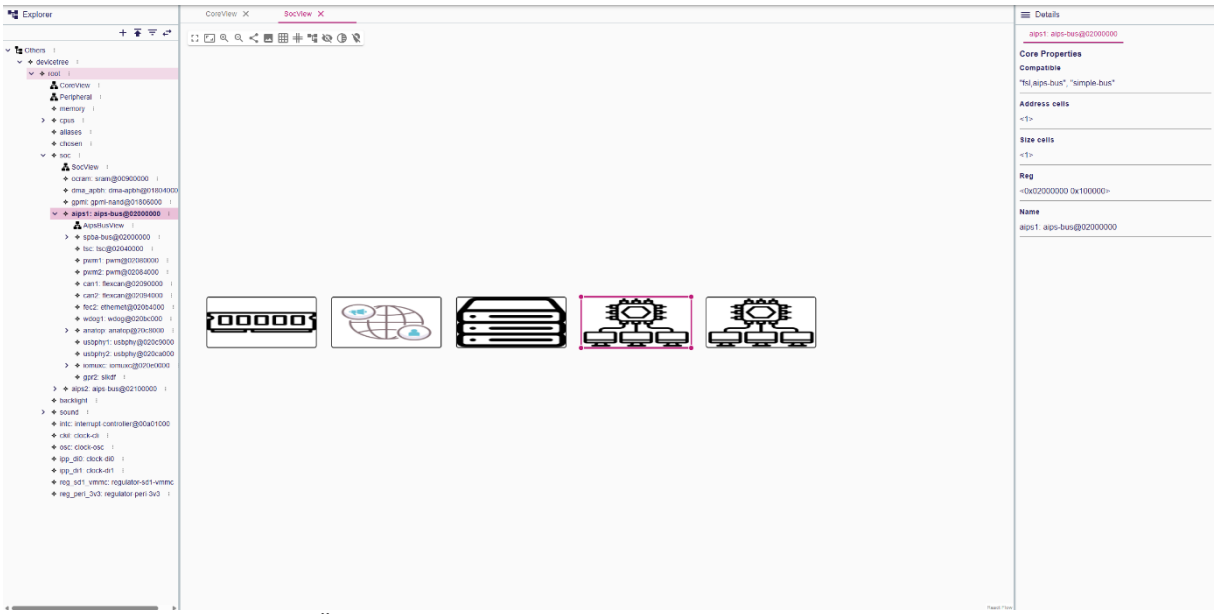
Şekil 2'de bir örneği verilen LCDP4DT'nin çevrimiçi DT modeli geliştirme ortamına bir kullanıcı girdiğinde yeni bir proje oluşturacak ve model tipi domain seçilerek bir isim verilecektir. Daha sonrasında bu projeye yeni bir nesne eklemeyi seçecek ve nesne tipi olarak önceden gelişirilmiş olan meta-model LCDP4DT'yi seçecektir. Başlangıçta sadece bir tane seçim hakkı vardır. Kapsama

prensibinden dolayı kardinalite'ye göre her bir nesnenin içine yeni ve onunla ilintili olan nesneleri seçecek ekleyecektir. Root altına girdiğinde kullanıcı artık core bakış açısında bulunmaktadır. Burada core altında olacak tüm nesneleri ekleyebilir. Burada ayrıca kademelendirmeden kaynaklı peripheral bakış açısındaki birtakım nesneleri de ekleyebilir. Bu bakış açıları görsel çerçevede ayrılmıştır. Sol taraftaki bilgileri ilgili şekilde tamamen doldurduktan ve tüm modeli oluşturduktan sonra kullanıcı oluşturduğu modeli indirmeli ve indirilen bu xml dosyasını LCDP4DT'nin işletimsel semantiğini oluşturan bir Python yazılımına iletmelidir. Söz konusu Phyton yazılımı DT modellerine karşılık gelen DT konfigürasyonlarını oluşturmak için LCDP4DT modellerini parse eder. Burada Python yazılımını çalıştırırken --xml-in model.xml --dts-out output.dts şeklinde kullanılmalıdır. Buna ek olarak ileride DT konfigürasyonlarından DT modellerini geri elde etme amacıyla --mapping mappings.json parametresi de eklenebilir. Bu sayede orijinal modeldeki düğüm isimleri ayrıca saklanmakta ve DT dönüşümünde yok olan xml düğümleri geri döndürülebilmektedir. Çünkü DT dönüşümünde xml düğümlerindeki name nitelikleri DT düğümlerine isim vermektedir. Kullanılan teknolojinin bize yeni bir xml yükleme özelliği de sunması ile ilerideki çalışmalarla oluşturulan yeni modeller sisteme geri yüklenip tekrardan grafiksel anlamda geliştirme sağlanabilecektir.

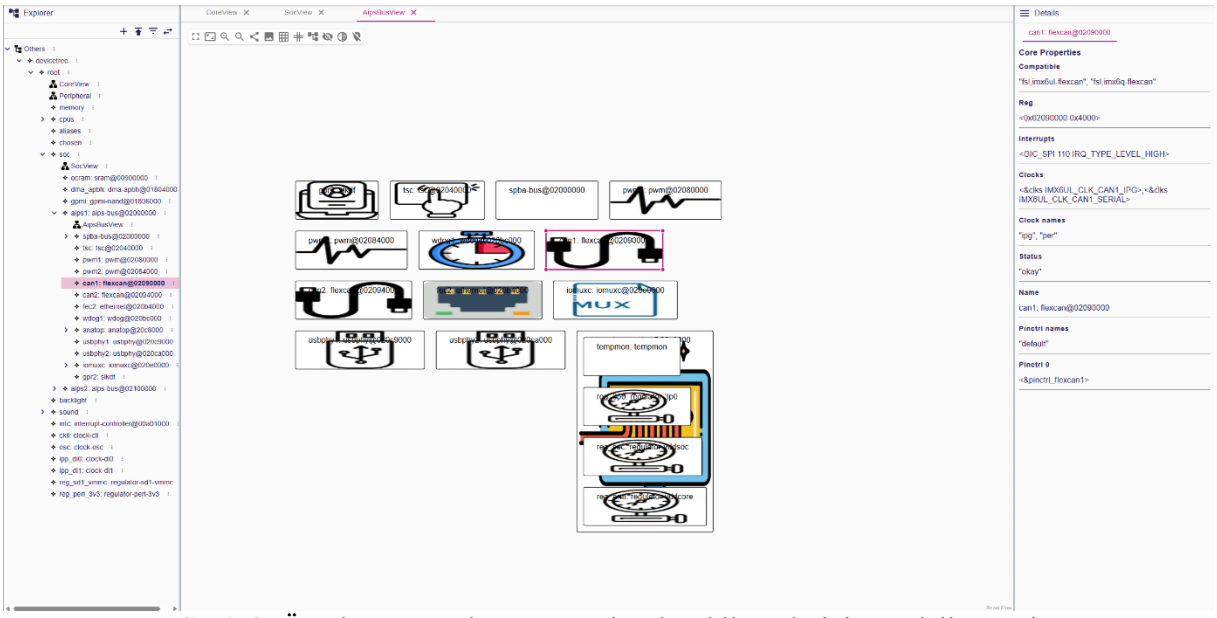
5. DURUM ÇALIŞMASI

Bu bölümde örnek çalışma olarak Linux çekirdeğinde imx6ul-14x14-evk.dtsi dosyasının LCDP4DT ortamında LCD yöntemi ile oluşturulması amaçlanmaktadır. imx6ul-14x14-evk.dtsi dosyası imx6ul işlemcisi ile oluşturulan genel amaç geliştirme kartının (NXP Semiconductor,2023) çalışması için üretici tarafından geliştirilmiş bir aygıt ağacıdır. i.MX6UltraLite işlemcisinin birçok en yaygın kullanılan özelliklerini gösteren ve özel bir geliştirme yapmadan önce kullanıcının işlemciye aşina olmasını sağlayan bir geliştirme kartının örnek aygıt ağacı konfigürasyonudur.. imx6ul-14x14-evk.dtsi dosyasında işlemci için geliştirilmiş imx6ul.dtsi dosyası kütüphane olarak dahil edilmiş ve bazı tanımlamaları üst dosyadan almıştır. Bizim örneğimizde ise genel anlamda bir aygıt ağacı üretimi amaçlanmaktadır. Bu sebeple örnek aygıt ağacında olanlara ek olarak çip temelli belirlenmiş registerlar ve saat frekansları gibi özelliklerle toplu bir aygıt ağacı oluşturulacaktır. Kullanıcı üst katmandan başlayarak alta doğru ilgili düğümleri oluşturarak bu düğümlerle ilgili bilgileri Şekil 2 ve Şekil 3'te sağda görünen kutucuklara girmektedir. Görsel bir bakış için önceden meta-modelde oluşturulmuş temsilleri kullanabilir. Böylelikle bakış açısına dayalı bir görsel elde edebilir. Tüm geliştirmeleri yaptıktan sonra kullanıcı ilgili modeli indirmeli ve Şekil 4'de gösterildiği gibi Python yazılımına iletmelidir. Sonuçta oluşturulan modele göre bir dts dosyası oluşturulmaktadır. Şekil 2'de gösterilen soc bileşeni ve içerisinde yer alan elemanların DT üretimi sonrası oluşan kodları Şekil 5'de gösterilmekte, aynı şekilde Şekil 3'te gösterilen aips_bus bileşeni ve içerisinde yer alan elemanların DT üretimi sonrası oluşan kodlar Şekil 6'da gösterilmektedir. Üretilen tüm DT yer kısıtı nedeniyle eklenememiştir.

LCDP4DT imx serisi için geliştirilmiştir. Örnek çalışma ise imx6ul serisinin geliştirme kartı için bir durum çalışmasıdır. Dts dosyalarında var olan kütüphanelerin dts dosyasına dahil edilmesi için yine Python yazılımına ek parametre geçerek bu kütüphaneler eklenebilir. Bunların dışında aygıt ağaçlarında her durumda bulunmayan ve birtakım çiplerin kullanımında eklenmesi gereken parametreler bu araç ile oluşturulamamaktadır. Daha genel ve bir örüntüsü olan adımlarda tamamen dönüşüm sağlamasına rağmen söz konusu DT dosyasının yaklaşık %90'ı sadece LCDP4DT'de modelleme ile otomatik olarak üretebilmektedir. Daha özelleşmiş yeni üst model elemanlarının LCDP4DT'ye eklenmesi ve bir arada kullanılmasıyla bu oran %100 oranına yaklaşabilir.



Şekil 2: Örnek DT uygulamasının Soc bileşenlerinin modellenmesi



Şekil 3: Örnek DT uygulamasının Aips_bus bileşenlerinin modellenmesi

```
PS D:\Masters Degree> python.exe .\converter.py -h
usage: converter.py [-h] [--xml-in XML_IN] [--dts-out DTS_OUT] [--dts-in DTS_IN] [--xml-out XML_OUT] --map MAP
                    [--includes [INCLUDES ...]]

Convert Sirius-compatible XML ⇔ DTS

options:
  -h, --help            show this help message and exit
  --xml-in XML_IN       Input XML file
  --dts-out DTS_OUT     Output DTS file
  --dts-in DTS_IN       Input DTS file
  --xml-out XML_OUT     Output XML file
  --map MAP             Path to node_map.json
  --includes [INCLUDES ...]
                        List of DTS include files to add (e.g., --includes 'header.dtsi' '<system/header.h>')
PS D:\Masters Degree> python3.12.exe .\converter.py --xml-in '.\Others_final.xml' --dts-out test_dts.dts --map mapping.j
son --include "<dt-bindings/clock/imx6ul-clock.h>" "<dt-bindings/gpio/gpio.h>" "<dt-bindings/input/input.h>" "<dt-bindin
gs/interrupt-controller/arm-gic.h>" "imx6ul-pinfunc.h"
[✓] DTS written to test_dts.dts
[✓] Map written to mapping.json
PS D:\Masters Degree>
```

Şekil 4: Örnek DT uygulamasının LCDP4DT'nin Python tabanlı kod dönüştürücü çalıştırılarak üretilmesi

```
54      soc {
55          address_cells = <1>;
56          size_cells = <1>;
57          compatible = simple-bus;
58          interrupt_parent = <&gpc>;
59          ocram: sram@00900000 {
60              compatible = mmio-sram;
61              reg = <0x00900000 0x20000>;
62          };
63          dma_apbh: dma-apbh@01804000 {
64              compatible = "fsl, imx6q-dma-apbh", "fsl, imx28-dma-apbh";
65              reg = <0x01804000 0x2000>;
66              interrupts = <0 13 IRQ_TYPE_LEVEL_HIGH>,<0 13 IRQ_TYPE_LEVEL_HIGH>,<0 13 IRQ_TYPE_LEVEL_HIGH>,<0 13 IRQ_TYPE_LEVEL_HIGH>;
67              dma_cells = <1>;
68              dma_channels = <4>;
69              clocks = <&clks IMX6UL_CLK_APBHDMA>;
70          };
71          gpmi: gpmi-nand@01806000 {
72              compatible = "fsl, imx6q-gpmi-nand";
73              address_cells = <1>;
74              size_cells = <1>;
75              reg = <0x01806000 0x2000>, <0x01808000 0x2000>;
76              reg_names = "gpmi-nand", "bch";
77              interrupts = <0 15 IRQ_TYPE_LEVEL_HIGH>;
78              interrupt_names = "bch";
79              clocks = <&clks IMX6UL_CLK_GPMI_IO>,<&clks IMX6UL_CLK_GPMI_APB>,<&clks IMX6UL_CLK_GPMI_BCH>,<&clks IMX6UL_CLK_GPMI_BCH_APB>,<&clks IMX6UL_CLK_PER_BCH>;
80              clock_names = "gpmi_io", "gpmi_apb", "gpmi_bch", "gpmi_bch_apb", "per1_bch";
81              dmas = <&dma_apbh 0>;
82              dma_names = "rx-tx";
83              status = "okay";
84              pinctrl_names = "default";
85              pinctrl_0 = <&pinctrl_gpmi_nand_1>;
86          };
87 >        aips1: aips-bus@02000000 {···
22          };
23 >        aips2: aips-bus@02100000 {···
45          };
```

Şekil 5: LCDP4DT modelinden oluşturulan DT kodunun içerisinde soc düğümünün örneği

```
aips1: aips-bus@02000000 {
    compatible = "fsl, aips-bus", "simple-bus";
    address_cells = <1>;
    size_cells = <1>;
    reg = <0x02000000 0x100000>;
    spba-bus@02000000 {
        compatible = "fsl, spba-bus", "simple-bus";
        address_cells = <1>;
        size_cells = <1>;
        reg = <0x02000000 0x40000>;
        ecspi4: ecspi@02014000 {
            address_cells = <1>;
            size_cells = <0>;
            compatible = "fsl, imx6ul-ecspi", "fsl, imx51-ecspi", "spi-gpio";
            reg = <0x02014000 0x4000>;
            interrupts = <GIC_SPI 34 IRQ_TYPE_LEVEL_HIGH>;
            clocks = <&clks IMX6UL_CLK_ECSPI4>,<&clks IMX6UL_CLK_ECSPI4>;
            clock_names = "ipg", "per";
            status = "okay";
            pinctrl_names = "default";
            pinctrl_0 = <&pinctrl_spi4>;
            pinctrl_assert_gpios = <&gpio5 8 GPIO_ACTIVE_LOW>;
            gpio_sck = <&gpio5 11 0>;
            gpio_mosi = <&gpio5 10 0>;
            cs_gpios = <&gpio5 7 0>;
            num_chipselects = <1>;
        };
        sai2: sai@0202c000 {
            sound_dai_cells = <0>;
            compatible = "fsl, imx6ul-sai", "fsl, imx6sx-sai";
            reg = <0x0202c000 0x4000>;
            interrupts = <GIC_SPI 98 IRQ_TYPE_LEVEL_HIGH>;
            clocks = <&clks IMX6UL_CLK_SAI2_IPG>,<&clks IMX6UL_CLK_SAI2>,<&clks IMX6UL_CLK_DUMMY>, <&clks IMX6UL_CLK_DUMMY>;
            clock_names = "bus", "mclk1", "mclk2", "mclk3";
            dmas = <&sdma 37 24 0>,<&sdma 38 24 0>;
            dma_names = "rx", "tx";
            status = "disabled";
            pinctrl_names = "default";
            pinctrl_0 = <&pinctrl_sai2>;
            assigned_clocks = <&clks IMX6UL_CLK_SAI2_SEL>,<&clks IMX6UL_CLK_SAI2>;
            assigned_clock_parents = <&clks IMX6UL_CLK_PLL4_AUDIO_DIV>;
            assigned_clock_rates = <0>, <12288000>;
        };
    };
    tsc: tsc@02040000 {
        compatible = "fsl, imx6ul-tsc";
        reg = <0x02040000 0x4000>, <0x0219c000 0x4000>;
        interrrupts = <GIC_SPI 3 IRQ_TYPE_LEVEL_HIGH>,<GIC_SPI 101 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&clks IMX6UL_CLK_IPG>,<&clks IMX6UL_CLK_ADC2>;
        clock_names = "tsc", "adc";
        status = "okay";
        pinctrl_names = "default";
        pinctrl_0 = <&pinctrl_tsc>;
    };
    pwm1: pwm@02080000 {
        compatible = "fsl, imx6ul-pwm", "fsl, imx27-pwm";
        reg = <0x02080000 0x4000>;
        interrupts = <GIC_SPI 115 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&clks IMX6UL_CLK_PWM1>,<&clks IMX6UL_CLK_PWM1>;
        clock_names = "ipg", "per";
```

Şekil 6: LCDP4DT modelinden oluşturulan DT kodunun içerisinde aips_bus düğümünün örneği

6. SONUÇ

Bu bildiride, gömülü sistemlerde kullanılan DT yazılımlarının MDE'si ve az-kodlu geliştirilmesi için LCDP4DT isimli bir yazılım platformu tanıtılmıştır. LCDP4DT grafik modelleme ve otomatik kod oluşturma özelliklerini bir araya getirerek, DT yazılım geliştiricileri için yazılım geliştirme sürecinde önemli iyileştirmeler sağlamaktadır. Özellikle web tabanlı yapısıyla erişilebilirliği ve iş birliğini artıran LCDP4DT, geliştirme süresini kısaltma ve DT geliştirme sürecini basitleştirme potansiyelini yükseltmektedir. Sunulan durum çalışması, platformun kullanım kolaylığını ve DT konfigürasyonlarının otomatik elde edilmesindeki başarımını örneklemiştir. Gelecek çalışmalarda farklı DT geliştirme senaryoları üzerinden yine LCDP4DT'nin DT uygulamalarını geliştirmeye getirdiği zaman tasarrufu ve

yine kod üretim performansı değerlendirilecektir. Mevcutta başlanan DT konfigürasyonlarından DT modellerini geri elde etme çalışmalarına devam edilerek LCDP4DT IDE'sinde tersine mühendisliğin de desteklenmesi sağlanacaktır. Böylece DT modelleri ve kodları arasında senkronizasyon oluşturulabilecek ve herhangi birinde yapılan bir değişikliğin diğerine de otomatik yansıtılması mümkün hale getirilecektir.

KAYNAKÇA

Arslan, S. and Kardas, G. (2020) "DSML4DT: A domain-specific modeling language for device tree software", Computers in Industry, vol. 115, 103179, pp. 1-13, DOI: 10.1016/j.compind.2019.103179

Arslan, S. ve Kardaş, G. (2018) "Aygıt Ağacı Yazılımlarının Modellenmesi", 12. Ulusal Yazılım Mühendisliği Sempozyumu (UYMS 2018), 10-12 Eylül 2018, İstanbul, Türkiye, CEUR Workshop Proceedings, vol. 2201, pp. 1-12

Arslan, S., Ozkaya, M., Kardas, G. (2023) "Modeling languages for internet of things (IoT) applications: A comparative analysis study", Mathematics vol. 11 no. 5, 1263

Arslan, S., Türk, E. ve Kardaş, G. (2017) "Gömülü Sistem Yazılımlarının Geliştirilmesinde Aygıt Ağacı Yapısının Kullanılmasına Yönelik bir Çalışma", 2. Uluslararası Bilgisayar Bilimleri ve Mühendisliği Konferansı (UBMK 2017), 5-8 Ekim 2017, Antalya, Türkiye, IEEE Conference Publications, pp. 882-887, DOI: 10.1109/UBMK.2017.8093472

Bock, A. C., & Frank, U. (2021). Low-code platforms: An analysis of current state and future directions. Business & Information Systems Engineering, 63(5), 421–437. https://doi.org/10.1007/s12599-021-00712-y

Brambilla, M., Cabot, J., Wimmer, M. (2022) "Model-Driven Software Engineering in Practice, Second Edition", Springer

Cabot, J.. 2020. Positioning of the low-code movement within the field of model-driven engineering. Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. Association for Computing Machinery, New York, NY, USA, Article 76, pp. 1–3. DOI: 10.1145/3417990.3420210

Devicetree Community. (2025, Mayıs 19). The Devicetree Specification. https://www.devicetree.org/

Devigne, C., Brejon, J.-B., Meunier, Q.L., Wajsbürt, F., 2017. Executing secured virtual machines within a many core architecture. Microprocess. Microsyst. 48, 21–35

Di Ruscio, D., Kolovos, D., de Lara, J., Pierantonio, A., Tisi, M, Wimmer, M., 2022, "Low-code development and model-driven engineering: Two sides of the same coin?", Software and Systems Modeling, vol. 21, pp. 437-446

Farhat, W., Sghaier, S., Faiedh, H., Souani, C., 2019. Design of efficient embedded system for road sign recognition. J. Ambient Intell. Humaniz. Comput. 10 (2),491–507

Gioia, E., Passaro, P., Petracca, M., 2016. AMBER: an advanced gateway solution to support heterogeneous IoT technologies. Proc. 24th International Conference on Software, Telecommunications and Computer Networks, 1–5.

Kardas, G., Ciccozzi, F. and Iovino, L. (2023) "Introduction to the special issue on methods, tools and languages for model-driven engineering and low-code development", Journal of Computer Languages, vol. 74, 101190, pp. 1-2, DOI: 10.1016/j.cola.2022.101190

Li, B., Bi, Y., He, Q., Ren, J., Li, Z., 2018. A low-complexity method for authoring an interactive virtual maintenance training system of hydroelectric generating equipment. Comput. Ind. 100, 159–172

Likely, G., & Boyer, J. (2008). A symphony of flavours: Using the device tree to describe embedded hardware. Linux Symposium, 2, 27–37.

Jassi, M., Hu, Y., Mueller-Gritschneder, D., Schlichtmann, U., 2018. Graph-grammar-based IP integration (GRIP)—an EDA tool for software-defined SoCs. ACM Trans.Des. Autom. Electron. Syst. 23 (3), Article 40:1-26

Madieu, J., 2017. "The Concept of a Device Tree", 139-166. Linux Device Drivers Development: Develop Customized Drivers for Embedded Linux. Packt Publishing.

Neuendorffer, S., 2018. FPGA platforms for embedded systems. In: Nicolescu, G.,Mosterman, P.J. (Eds.), Model-Based Design for Embedded Systems. CRC Press,pp. 351–379

Nikkel, B., 2016. NVM express drives and digital forensics. Digit. Investig. 16, 38–45

NXP Semiconductor, (2023, Kasım 23). i.MX 6UltraLite EVK Board Hardware User Guide, Rev. 2

Rodeh, O., Bacik, J., Mason, C., 2013. BTRFS: the linux B-tree filesystem. ACM Trans.Storage 9 (3), Article 9:1-32.

Saritas, H. B. and Kardas, G. (2014) "A model driven architecture for the development of smart card software", Computer Languages, Systems & Structures, vol. 40, no. 2, pp. 53-72, DOI: 10.1016/j.cl.2014.02.001

Schüpbach, A., Baumann, A., Roscoe, T., Peter, S., 2012. A declarative language approach to device configuration. ACM Trans. Comput. Syst. 30 (1), Article 5:1-35.

Simmonds, C. (2015). Mastering Embedded Linux Programming (2nd ed.). Birmingham, UK: Packt Publishing.

Sirius-web. (2025, Mayıs 19). Documentation. Eclipse Foundation. https://github.com/eclipse-sirius/sirius-web/tree/master/doc

Swaroop, K.N., Chandu, K., Gorrepotu, R., Deb, S., 2019. A health monitoring system for vital signs using IoT. Internet Things 5, 116–129