Aygıt Ağacı Yazılımlarının Modellenmesi

Sadık Arslan[1,2] ve Geylani Kardaş[2]

[1] Kent Kart Ar-Ge Merkezi, Kent Kart Ege Elektronik A.Ş., İzmir, Türkiye
[2] Ege Üniversitesi, Uluslararası Bilgisayar Enstitüsü, İzmir, Türkiye
sadik.arslan@kentkart.com.tr, geylani.kardas@ege.edu.tr

Özet. Aygıt Ağacı (DT) dosyaları çeşitli gömülü sistem donanımları içindeki fiziksel aygıtların ve çevre birimlerinin tanımlanmasını ve yapılandırmasını sağlar. Ancak, DT kaynak dosyalarının bilinen genel amaçlı programlama dillerinden farklı bir yapıya ve karmaşık bir sözdizimine sahip olması ve geliştiricilerin bu dosyaların hazırlanması için farklı mikroişlemcilere özgü donanımların detaylarını bilmesi gerekliliği DT uygulamalarının geliştirilmesini zorlaştırmaktadır. DT yazılım geliştirme süreçlerinin bu zorluklarını ortadan kaldırmak amacıyla, bu bildiride farklı gömülü sistem platformları için DT yazılımlarının model güdümlü geliştirilmesine ve bu yazılımların otomatik yapılandırılmasına imkân verecek bir üstmodel tanıtılmaktadır. Bu üstmodelin içerdiği DT yazılımı bakış açıları bildiride anlatılmış ve bu bakış açılarına göre DT yazılımlarının görsel olarak bir araç desteği ile nasıl modellenebileceği bir durum çalışması üzerinden örneklendirilmiştir.

Anahtar Kelimeler: Model-Güdümlü Yazılım Geliştirme, Üstmodel, Aygıt Ağacı, Gömülü Yazılım.

Modeling Device Tree Software

Abstract. Device Tree (DT) files support the description and the configuration of physical devices and peripherals inside the hardware of embedded systems. However, software developers encounter difficulties while developing such DT applications due to the structure and the syntax of DT source files which are complex and different from the well-known general purpose programming languages. Furthermore, the developers also need to be familiar with the hardware specifications of each different microprocessor to prepare such DT files. In order to eliminate these difficulties of current DT software development processes, a metamodel, which may provide model-driven generation and automatic configuration of DT software for different embedded system platforms, is introduced in this paper. Modeling viewpoints of this metamodel are described and modeling DT software visually by using these viewpoints is exemplified with a case study.

Keywords: Model-Driven Software Development, Metamodel, Device Tree, Embedded Software.

1 Giriş

Bir Aygıt Ağacı (DT), gömülü sistem donanımının fiziksel aygıt bileşenlerinin düğümlerle tanımlanmasını sağlayan bir veri yapısıdır [1]. "Open Firmware" ve "Power Architecture Platform Requirements" gibi standartlar dâhilinde kullanılan DT, gömülü sistemdeki donanım bileşenlerinin eksiksiz bir teknik tanımını sağlamaktadır [2]. Bir işletim sistemi çekirdeği derlendikten sonra, farklı donanım yapılandırmaları için düzenlenmiş, farklı ağaç yapılarına sahip, büyük işlemci aileleri için destekleri içermektedir. Birçok şirket çok sayıda mikro işlemci ürününe sahiptir. Yonga Üstü Sistem (SoC) platformlarında farklı sistemler için işletim sistemi dağıtımlarının sürekli yayınlandığı da göz önüne alındığında bu mikro işlemci ürünlerinde DT kullanımı giderek önem kazanmaktadır. DT dosya desteğinin bulunmadığı durumlarda saat sinyali frekansı, bacak adı ve kesmeler gibi donanım bilgilerini değiştirmek için işletim sistemi çekirdek kodunun tekrar tekrar değiştirilmesi gerekir. Bu sürecin hem bakım maliyeti yüksektir hem de uzun zaman almaktadır. DT kullanımı bu süreci kolaylaştırmaktadır.

İşletim sistemi çekirdeğinde bulunan donanım özellikleri, derlenmiş bir aygıt ağacı yapısına dönüştürülür ve çekirdekten dışa aktarılır. Çekirdek derlendiğinde, donanım sisteminin özellikleri hariç tutulur ve genel amaçlı bir işletim sistemi çekirdeği elde edilir. Ancak, DT'yi özellikle çok sayıda farklı mikroişlemci mimarisinde çalışacak sistemlerin geliştirilmesinde uygulamak zordur. DT kaynak dosyalarının bilinen genel amaçlı programlama dillerinden farklı bir yapıya ve karmaşık bir sözdizimine sahip olması ve geliştiricilerin hem bu farklı DT sözdizimini öğrenmesi hem de bu dosyaların hazırlanması için farklı mikroişlemcilere özgü donanımların detaylarını bilmesi gerekliliği DT uygulamalarının geliştirilmesini zorlaştırmaktadır. Üstelik DT çalışmaları için kullanılacak her yeni platforma özel metin dosyasının ayrı olarak ve en baştan hazırlanması gerekmektedir. Farklı mikroişlemci mimarileri için DT bileşenlerinin kodlanması ve yapılandırılması, birçok geliştirici için zahmetli ve zaman alıcıdır.

Yukarıda belirtilen DT yazılımı geliştirme süreci zorlukları model-güdümlü geliştirme teknikleri uygulanarak azaltılabilir. Bu amaçla DT yazılımlarının modellenmesinde kullanılacak bir üstmodel ve içerdiği bakış açıları bu bildiride tanıtılmaktadır. Bir DT organizasyonunun tüm yönlerini kapsayan bu üstmodele dayalı bir görsel sözdizim oluşturularak gömülü sistem DT yazılımlarının uluslararası "Devicetree" tanımı ve standardına [2] uygun olarak geliştirilmesi sağlanmaktadır. DT'lerin Model-Güdümlü Mühendislik'e (ing. Model-Driven Engineering) (MDE) [3] dayalı geliştirilmesi sırasında grafiksel araçların nasıl kullanılabileceği de bu çalışmada gösterilmiştir.

Bildirinin 2. bölümünde, DT yapısı hakkında kısaca bilgi verilmiştir. DT üstmodeli 3. bölümde anlatılmıştır. Bölüm 4'te, önerilen DT üstmodeline dayalı bir görsel somut sözdizim tanıtılmaktadır. Bölüm 5'te, geliştirilen bu sözdizimin kullanılmasını örnekleyen bir durum çalışması verilmiştir. Bölüm 6'da, ilgili literatürdeki önceki çalışmalar anlatılmaktadır. Son bölümde sonuçlar ve ileriye yönelik çalışma önerileri yer almaktadır.

2 Aygıt Ağacı Yapısı

Genellikle Linux işletim sistemi çekirdeği ile kullanılan DT, Acorn RISC Machine (ARM), x86, MicroBlaze, PowerPC ve SPARC işlemci mimarileri ile çalışabilmektedir. DT, donanım konfigürasyonunu açıklayan bir veri yapısıdır. Bu yapı, çalışılan gömülü sistemin işlemcisi, belleği, veri yolları ve çevre birimleri gibi birçok bölüm hakkında bilgi içerir. İşletim sistemi, önyükleme sırasında DT yapısını ayrıştırır ve mikroişlemciyi nasıl yapılandıracağını burada belirler. Ayrıca yüklenecek aygıt sürücüleri hakkında kararlar almak için DT yapısı kullanılır.

DT yapısı, "/" karakteri ile gösterilen, kök olarak adlandırılan bir düğümle başlamaktadır. DT'de her düğümden çok sayıda bulunan ve ismi ve numarası olabilen çocuk düğümler oluşturulabilmektedir. Düğümler isteğe göre, ek veri içeren nitelik değerlerini saklamaktadırlar. DT yapısı, IEEE 1275-1994 standardı tarafından daha önceden belirlenmiş kurallara uygun olarak tasarlanmaktadır. DT yapısında, genel bir ağaç yapısına bezer bir ebeveyn ve çocuk düğümleri vardır. Örnek olarak, basit bir kök düğümü ve alt düğümleri içeren DT dosyası Şekil 1'de görülebilir. Gerçek bir gömülü sistem uygulamasına ait DT dosyaları Şekil 1'deki metinsel yapıya sahip olmakla birlikte Evrensel Dizisel Veriyolu (USB), Evrensel Eşzamansız Alıcı-Verici (UART), Seri Çevresel Arayüz (SPI) gibi birçok farklı arayüzün ve özelliğin ayarlanması için manuel olarak hazırlanması gereken yüzlerce satır koddan oluşmaktadır.

```
01  / {                                  // kök düğümü
02   bir-nitelik;                        //bir donanım özelliği
03   bir-cocuk-dugum {                   //ilk çocuk düğüm
04      dizi-niteligi = <0x10 53>;
05      yazi-niteligi = "merhaba, dunya";
06      };
07   diger-cocuk-dugum {                 //diğer çocuk düğüm
08      binary-nitelik = [0221ALI];
09      yazi-listesi = "evet","hayir","belki";
10      };
11 };
```

Şekil 1. DT yapısı örneği

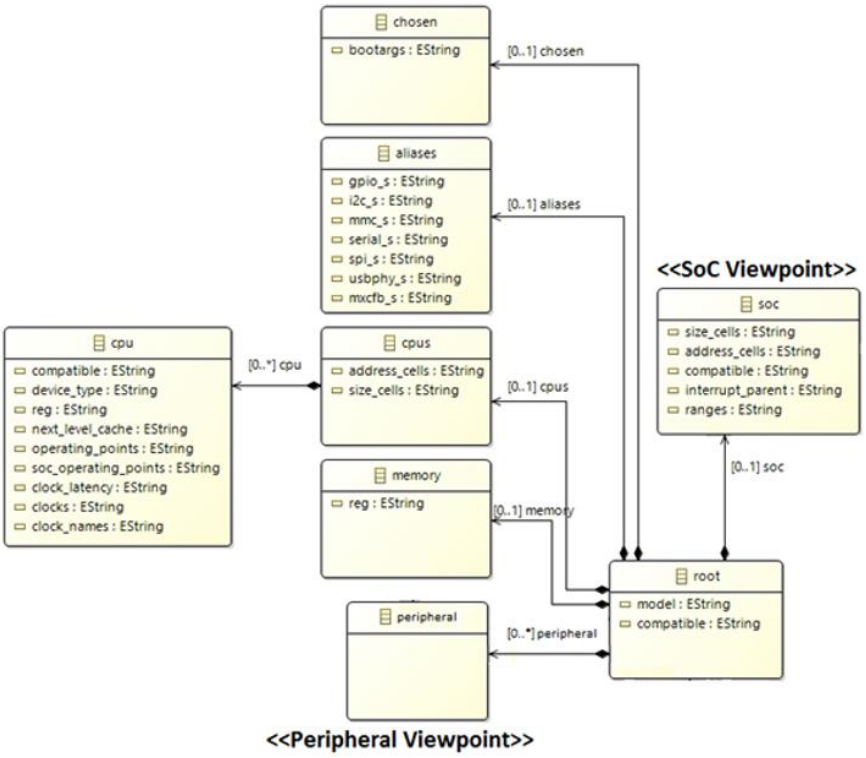
3 DT Üstmodeli

DT yazılımlarının modellenmesini sağlayacak üstmodel [2]'de yer alan DT tanımlamaları göz önünde bulundurularak oluşturulmuştur. Üstmodel hem DT standartlarındaki elemanların ilişkilerini yansıtabilmek hem de kolay anlaşılmayı ve verimli kullanımı sağlamak amacıyla beş farklı bakış açısına (ing. viewpoint) bölünmüştür. Bunlar Core, SoC, Aips_Bus, Spba_Bus ve Peripheral olarak isimlendirilmişlerdir. Toplamda 60'tan fazla DT bileşeni ve bunların ilişkilerini içeren üstmodelin bakış açıları aşağıdaki altbölümlerde anlatılmaktadır. Ancak bildirideki yer kısıtları nedeniyle sadece Core bakış açısının Eclipse Ecore ile encode edilmiş haline ait diyagram Şekil 2'de verilebilmiştir. Diğer üstmodel bakış açıları metinsel

olarak anlatılmaktadır. Anlatım sırasında üstmodele ait varlıklar orijinal tanımlarındaki İngilizce isimleri ile metinde geçmektedir. Normal metinden ayrılmaları için eğik olarak yazılmışlardır.

3.1 Core Bakış Açısı

Core bakış açısının tüm üst-varlıkları (ing. meta-entity) DT standartlarında tanımlanan düğümlerdir. *root* elemanı, tüm sistemin türetildiği temel düğümdür. *root* kendinden türetilen diğer üst-varlıklarla *has-a* ilişkisi içindedir. Gömülü sistemin işlemci ve belleği bu bakış açısında tanımlanmıştır. Çok çekirdekli işlemcilerde, her çekirdek birimi için gerekli türevler *cpus* elemanından yapılır. Ek olarak, herhangi bir donanım ilişkisine sahip olmayan *aliases* ve *chosen* üst-varlıkları burada bulunmaktadır. Bu üst-varlıklar, tüm DT yapısında kullanılacak kısaltmalar, tanımlamalar ve önyükleme parametre geçişlerini sağlarlar. Şekil 2, geliştirilen üstmodelin Core bakış açısını göstermektedir. Üstmodel varlıkları sarı dikdörtgenlerle gösterilmiştir. "<<" ve ">>" diğer bakış açıları ile olan ilişkileri göstermektedir. Bu unsurlar bakış açıları arasında ortak unsurlardır ve bakış açıları arasındaki geçişleri sağlarlar. Örneğin, Core bakış açısında bulunan SoC üst-varlığı <<SoC Viewpoint>>'ten gelmektedir. Bu gösterim Şekil 2'de sağ tarafta görülebilmektedir.



Şekil 2. DT Core bakış açısı

3.2 SoC Bakış Açısı

Bu bakış açısı, gömülü sistemlerde SoC entegre devrelerinin özelliklerine yönelik desteği içermektedir. İlgili bakış açısındaki tanımlamalar kullanılarak ses, görüntü ve zamanlayıcı gibi birçok SoC özelliğinin parametre ayarları modellenebilir. Burada, *soc* ve *interrupt_controller* adlı iki temel üst-varlık vardır. Bu üst-varlıklar, *root* elemanı ile sahiplik ilişkisi içerisindedir. *aips_bus*, *ipu*, *gpmi_nand*, *timer*, *l2_cache*, *pcie*, *hdmi_core*, *hdmi_video*, *hdmi_audio*, *hdmi_cec* ve *gpu* üst-varlıkları, *soc* üst-varlığından türetilmiştir. *interrupt_controller* üst-varlığı, gömülü sistemdeki tüm kesmelerin üretilmesini ve ayarlanmasını yönetir. Ayrıca, *soc* üst-varlığında *interrupt_controller*'dan bir parametre kullanılmaktadır. Bu iki üst-varlık arasında *interrupt_parent_for_soc* ilişkisi mevcuttur. *soc* üst-varlığının kesme ebeveyni *interrupt_controller*'dır.

3.3 Aips_Bus Bakış Açısı

Düşük bant genişliğine sahip SoC çevresel bileşenleri, gömülü sistemlerde Aips Bus arayüzü üzerinden SoC birimleri ile iletişim kurmaktadırlar. Aips_Bus bakış açısı ve üst-varlığı, *soc*'dan türetilmiştir. Bu üstmodel elemanından birçok başka eleman üretilebilir. Gömülü bir sistemde *caam, iomuxc, ldb, usb, fec, i2c, uart, pwm, flexcan, gpio, wdog, clks, usbphy* ve *spba_bus* gibi elemanlar, SoC uyumluluğu ile *aips_bus* üst-varlığından üretilebilirler. Bu özellikler sistemde mevcutsa, bu üstelemanların örnekleri hazırlanan sistem modelinde kullanılırlar. Kriptografik Hızlanma ve Güvence Modülü (*caam*) ve *caam*'dan üretilen diğer üst-varlıklar *interrupt_controller* ile *interrupt_parent* ilişkilerinde bulunur.

3.4 Spba_Bus Bakış Açısı

Gömülü sistemlerde, paylaşılan bazı harici birimlerle iletişim kurmak için Paylaşımlı Çevresel Hat Arayüzü (SPBA) veriyolu kullanılır. Bu arabirim Akıllı Doğrudan Bellek Erişimi (SDMA) çekirdeği ve çevre birimleri arasında iletişim kurar. DT yapısında bu iletişim için birçok farklı alt birim kullanılmaktadır. Üstmodelde bulunan *spba_bus, aips_bus* üst-varlığından üretilir. *spdif, esai, ssi* ve *ecspi* gibi dijital ses çıkışı ve ses birimleri *spba_bus* üst-varlığından üretilebilir. Bu elemanlar sistemin yapısına bağlı olarak birden fazla olabilirler.

3.5 Peripheral Bakış Açısı

SoC entegre devresinde olmayan ve SoC'a dışarıdan bağlı üniteler üstmodelin Peripheral bakış açısında bulunur. Bu bakış açısında, farklı ses ve video dönüştürücüleri gibi entegre devre çevre birimleri tanımlanmıştır. Bu elemanlar doğrudan Core bakış açısındaki *root* üst-varlığından üretilir. Bu bakış açısındaki birimler, benzer isimlere sahip olsalar bile, SoC'ta bulunmayan çevresel birimleri temsil eder. *clocks* üst-varlığı elemanı, SoC entegre devresine bağlı saat sinyallerinin bilgilerinin girildiği bölümdür. Gömülü sistem herhangi bir bataryadan besleniyorsa,

battery üst-varlığı kullanılır. Sistemde açma/kapama düğmeleri gibi düğmeler varsa *gpio_keys* kullanılır. Her yeni özel düğme *gpio_keys*'den oluşturulur. Sisteme harici olarak eklenebilen SPDIF ve HDMI dönüştürücüleri için sırasıyla *sound_spdif* ve *sound_hdmi* üst-varlıları üstmodele dahil edilmiştir. Sistemde bulunan Kırmızı Yeşil Mavi (RGB) LCD'ler için bir *LCD* üst-varlığı oluşturulmuştur. LCD'lerin arka ışığını ayarlamak için, *backlight* üst-varlığı DT yapısında bulunmaktadır.

4 DT Modelleme için bir Somut Sözdizim

Bir önceki bölümde tanıtılan DT üstmodeli DT yazılımlarının MDE'sinde kullanılabilecek bir görsel modelleme dilinin soyut sözdizimi olarak kabul edilebilir. Bu soyut sözdizime karşılık gelen ve bu sözdizimdeki kavramlar ve bunların örnek modeller üzerindeki temsilleri arasında bir haritalama sağlayan görsel bir somut sözdizim bu bölümde tanıtılmaktadır.

DT görsel modelleme somut sözdizimini oluşturmak ve bu sözdizime dayalı olarak DT yazılımlarının grafiksel modellenmesine imkan verecek MDE aracını oluşturmak için bu çalışmada Sirius platformu [4] kullanılmıştır. Sirius, Eclipse modelleme teknolojilerini kullanarak grafik modelleme ortamını kolayca oluşturmaya olanak tanımaktadır. Sirius ile oluşturulan bir modelleme ortamı, kullanıcıların Eclipse Ecore modelleri oluşturmasına, düzenlemesine ve görüntülemesine izin veren Eclipse editörlerinden oluşur. Editörler, modelleme ortamının bütün yapısını, davranışlarını, tüm baskı ve navigasyon araçlarını tanımlayan bir model ile tanımlanır. Bir Sirius modelleme ortamının tanımı, Eclipse IDE içinde dinamik olarak yorumlanır. Sağladığı bu özellikler nedeniyle çalışmada altyapı olarak Sirius'un kullanılması tercih edilmiştir.

Somut sözdizimi geliştirmek için ilk olarak, DT üstmodelindeki üst-varlıklar (kavramlar) için grafiksel gösterimler belirlenmiştir. Düğümleri Ecore dosyasındaki alan kavramlarına bağlamak için yine Sirius aracı kullanılmıştır. Tablo 1'de DT üstmodelinin Core bakış açısındaki üst-varlıkların grafiksel gösterimleri bulunmaktadır. Yine yer kısıtları nedeniyle diğer bakış açılarına ait gösterimler burada verilememiştir.

Tablo 1. Core bakış açısı için somut sözdizim kavramları ve gösterimleri

Kavram	Gösterim	Kavram	Gösterim
Root		Cpu	
Chosen		Memory	
Aliases		Soc	
Cpus			

Görsel diyagramlarda kolayca eleman ve ilişkileri göstermek amacıyla Ecore modelleri geliştirme sürecinde kullanılmaktadır. Bu işlem sırasında, semboller hem paletler hem de şekiller için ayarlanır. Bunu sağlamak için simge geometrisi Sirius içerisinde tanımlanmış ve bazı kısıtlama kontrolleri dikkate alınmıştır. Ortaya çıkan yapı, geliştiricilerin DT somut sözdizimiyle eşleşen gerekli bakış açılarının her biri için modeller tasarlayabilecekleri bir görsel modelleme aracıdır. Araç modelleme işlemi sırasında bazı kısıt kontrollerini otomatik olarak sağlamaktadır. Böylece DT üstmodelinin tanımlamalarına ve semantiğine tam uyumlu yazılım modelleri oluşturulabilmektedir. İlgili kontroller ve model doğrulama kuralları aşağıdadır:

Model Kısıtlamaları: Ecore ile kodlanan DT üstmodelinden gelen kısıtlamalar, tüm bakış açılarındaki örnek modeller için sağlanmıştır. Bu kısıtlamalar aşağıdadır:

Bölme kısıtlamaları: DT üstmodelinde üst-varlıklar arasındaki bileşimsel ilişkiler kontrol edilmektedir. Örneğin, Core bakış açısındaki *root* üst-varlığından *memory* ve *battery* üretilebilir. Ancak, bu ilişkinin bulunamayacağı varlıklarda bu gerçekleşmez.

İlişki sayısı kısıtlamaları: Örnek modeldeki elemanlar arasındaki ilişkilerin sayısı bire bir, bire çok, çoktan çoğa ilişkilere bağlı olarak kontrol edilir. Örneğin, *root* üst-varlığından sadece bir *cpus* elemanı türetilebilir. Ancak *cpu* öğelerinin sayısı konfigürasyona bağlı olarak daha yüksek olabilir.

İlişki kaynağı ve hedef kısıtı: İlişkinin yönü ilişkinin kaynağını ve hedefini tanımlar. İlgili kısıtlama Ecore seviyesinde belirlenmiştir. Örneğin, *root* ve *battery* arasındaki ilişkinin yönünün değiştirilmesine modelleme sırasında izin verilmez.

Grafiksel Araç Kısıtlamaları: Üstmodel kısıtlamalarına ek olarak, DT yazılım modeli oluştururken kullanıcıya tasarımda yardımcı olan editöre bağlı bazı kısıtlamalar vardır. Bunlar:

Bakış açıları arasında geçiş: Bu kısıtlama, farklı bakış açılarının editörleri arasında geçiş sağlayarak sistemin birliğini tesis etmektedir. Bu yapı sistemin adım adım oluşturulmasını sağlar. Araç, diyagram dosyaları oluştururken oldukça esnektir. Bir kullanıcı talebi durumunda, bakış açısı şemalarını ayrı ayrı oluşturmak mümkündür. Örneğin, kullanıcı Core diyagramını tasarlamadan SoC diyagramı tasarlayabilir.

Birleşim: Bu özellik tüm üst-varlıklar için sistemde birleştirme sağlar. Editör diyagramları bakış açılarına dayanmaktadır. Öte yandan, sistem üstmodeli bir bütün model olarak düşünülmelidir. Örnek olarak, Core bakış açısı düzenleyicisinde oluşturulan bir *root* verilebilir. Bu *root* Peripheral bakış açısında da kullanılmalıdır. DT modelleme aracımız, eğer bakış açısı yapılarında tanımlanmışsa, bu elemanı Peripheral'a da ekler. Öğeler, diyagramlarda gerektiğinde paletlerden sürükleyip bırakılarak kullanılabilir.

İlişki-varlık bütünlüğü: Aracın bu kısıtlamasına göre, herhangi bir örnek modelde oluşturulan bir DT varlığı kaldırıldığında, bu varlığın dahil olduğu tüm ilişkiler de modelden kaldırılacaktır. Bu, tüm modelin bütünlüğünün korunmasını ve modelin bu değişikliklerden sonra tutarlı olmasını sağlamaktadır.

Doğrulama Kuralları: Yukarıda açıklanan kısıtlamaların ve özelliklerin yanı sıra Sirius tabanlı görsel DT modelleme aracı üzerinde doğrulama kuralları tanımlanmıştır. Örneğin, bazı üst-varlık örneklerinin (ing. instance) mutlaka diyagramlarda bulunması gerekiyorsa bunlar için doğrulama kuralları araç içerisinde

mevcuttur ve bunlar modelleme sırasında otomatik olarak kontrol edilir. Sunulan "Validate Diagram" işlemi, DT görünüm noktalarına göre hazırlanmış olan DT şemalarında gerçekleştirildiğinde, hata mesajları bu kurallara göre görüntülenir. Tüm DT üstmodel bakış açıları için toplam 23 adet kural tanımlanmıştır. Bu doğrulama kurallarının işletimi bir sonraki bölümde örneklendirilmiştir.

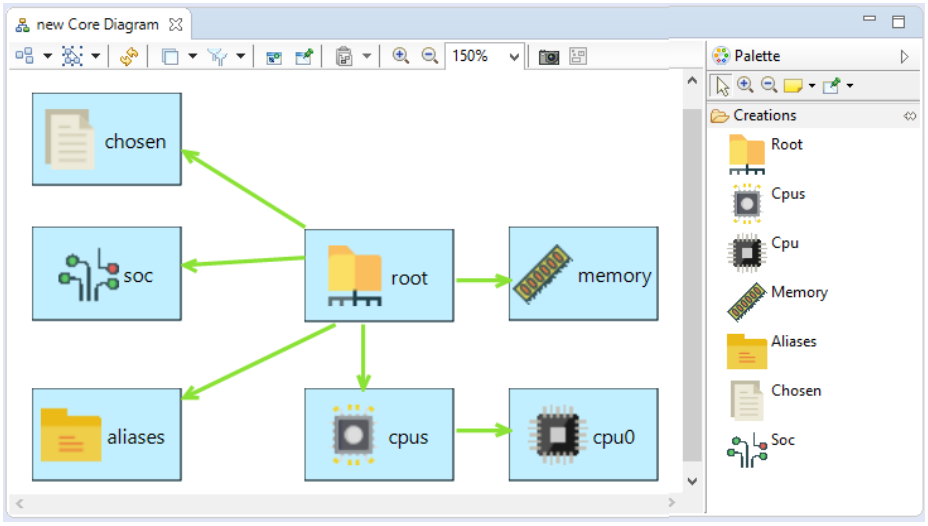
5 Durum Çalışması

Hazırlanan DT üstmodelinin ve bu üstmodele göre modellemeye imkan veren görsel modelleme aracının kullanımını örneklemek amacıyla bu bölümde toplu taşıma araçlarında kullanılan bir ağ video kayıt cihazı için DT yapısının geliştirilmesi göz önüne alınacaktır. Cihazda Linux işletim sistemi bulunmaktadır. Bu cihaz, ARM i.MX53 serisi bir mikro işlemciye sahiptir. Cihaz ayrıca 1GB ana bellek ve 512MB depolama ünitesine sahiptir. Örneği basit tutmak amacıyla cihazda ekran veya dokunmatik ekran ihtiyacının bulunmadığı varsayılmaktadır. Ayrıca ses ve video çalıştırmaya ait çevresellerin kullanılması da göz önüne alınmamıştır. İletişim için RS232, RS485, USB ve Ethernet arayüzleri kullanılmaktadır. Cihazda ayrıca ivme ölçer, sadece okunur bellek ve sıcaklık sensörü mevcuttur. Bu birimler Entegre-Arası Devre (I2C) arayüzü ile kontrol edilir. Cihaz, GPS modülüne sahiptir. Böylece küresel konumlandırma bir Evrensel Asenkron Alıcı/Verici (UART) arayüzü ile elde edilebilir. USB üzerinden dış dünya ile iletişim sağlamak için bir GSM modülü kullanılır. Wifi modülü, cihazda İnternet bağlantısı sağlamaktadır.

5.1 Cihaz İçin Sistem Modellemesi

Cihaza ait DT'nin Bölüm 3'te tanıtılan üstmodelin farklı tüm bakış açılarına göre modelleri görsel modelleme aracı kullanılarak oluşturulmuştur. Örneğin Şekil 3'te cihazın DT üstmodeli Core bakış açısına göre kısmi modeli görülmektedir. Hazırlanan görsel modelleme aracında hangi bakış açısına göre modelleme yapılacaksa o bakış açısının içerdiği üst-varlıkların görsel notasyonunu sunan ve sürükle-bırak ile model üzerine eleman taşınabilecek bir modelleme paleti aracın sağ tarafında kullanıcıya gösterilmektedir. Örneğin yine Şekil 3'te sağ tarafta şu an Core bakış açısına göre modelleme yapıldığından sadece bu bakış açısına ait elemanların yer aldığı palet bulunmaktadır. Ayrıca farklı bakış açıları için ortak elemanlara ait örneklerin bir bakış açısı modelinde oluşturulması durumunda bu örneğin diğer bakış açılarına da eklenmesi otomatik olarak gerçekleştirilmektedir. Örneğin, *soc* sembolü hem Core hem de Soc bakış açılarında bulunur. Bu nedenle, Core bakış açısı diyagramında oluşturulan *soc* düğümü, otomatik olarak Soc diyagramına da eklenir. Böylece modelin bakış açıları arasındaki tutarlılık sağlanır.

Şekil 3'te görüldüğü üzere video kayıt cihazının DT Core bakış açısı için *root* düğümü oluşturulmuş ve DT yapısı için zorunlu olan *memory, chosen* ve *aliases* düğümleri eklenmiştir. Cihazda tek çekirdekli işlemci olduğu için *cpus* düğümünde 1 adet *cpu* düğüm oluşturulmuştur. Ayrıca, entegre özellikleri üzerine sistemin oluşturulacağı *soc* düğümü modele dahil edilmiştir.

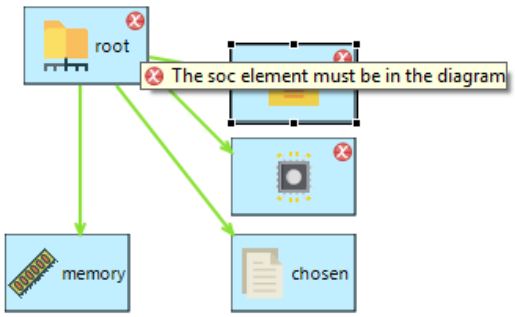


Şekil 3. Core bakış açısı için görsel modelleme

Cihazın entegre özellikleri ile ilgili olarak Soc bakış açısında birçok model elemanı tanımlanmıştır. Cihazın kesme ve haberleşme birimleri için *tzic* ve *aips* düğümleri oluşturulmuştur. Aips_Bus bakış açısında cihaza ait 2 adet Aips_Bus hattı (*aips1* ve *aips2)* tüm özellikleri ile birlikte modellenmiştir. Spba_Bus bakış açısı için 3 *ssi ve* 3 *ecspi* düğümleri oluşturulmuş; SPI iletişim hatları için parametre girişlerinin yapıldığı *ecspi1~2* örnekleri modele dâhil edilmiştir. Peripheral bakış açısı kapsamında pil desteği için *battery*, harici düğme için *gpio-keys* düğümleri modele eklenmiştir. Core bakış açısı haricinde yukarıda belirtilen bakış açılarına ait alt modeller ve bunların anlatımları yer kısıtları nedeniyle bu bildiride yer almamaktadır.

5.2 Modellenen Sistemin Doğrulanması

DT modelleri oluşturulurken hazırlanan araç içerisinde 4. bölümde anlatılan kısıtların kontrolleri yapılmakta ve bir dizi doğrulama işlemleri yine araç tarafından otomatik olarak yerine getirilmektedir. Araçta doğrulama işlemi için daha önce de belirtildiği gibi "Validate Diagram" seçimi vardır. Bu seçim yapıldıktan sonra DT üstmodel tanımlamalarına ve statik semantik kurallarına aykırı durumlar varsa yazılım geliştiriciye hata mesajları ile bildirilir. Örneğin Şekil 4'te Core bakış açısına göre ağ video kayıt cihazı modellemesi sırasında alınan bir hata görülebilmektedir. DT üstmodeli Core_5 kuralına göre, Core diyagramda mutlaka *soc* düğümünün olması gerekmektedir. Araç bu model elemanının eksikliğini belirlemiştir ve bu durum yazılım geliştiriciye bildirilmektedir. Model tasarımları ancak tüm bu hatalar düzeltildikten sonra tamamlanabilmektedir.



Şekil 4. Araç içerisinde bir DT model doğrulama örneği

6 İlgili Çalışmalar

Literatürdeki çalışmalar incelendiğinde, donanım sürücülerinin ve/veya kod üretiminin MDE kapsamında gerçekleştirilmesine yönelik bazı öneriler olduğu, ancak bunların DT içeren yazılımlarının oluşturulmasını dikkate almadıkları görülmektedir. [5]'teki çalışmada, Linux çekirdeği, sürücü ve "Makefile" dosyalarının üretildiği ve tasarımda model-güdümlü tekniklerin kullanıldığı belirtilmiştir. Bununla birlikte, önerilen sistem DT ve DT yazılımlarını içermemektedir. Aynı ekibinin başka bir çalışmasında modele dayalı sürücü üretimi yapılmış, ancak DT yazılımları kullanılmamıştır [6]. Başka bir çalışmada [7], sürücü kodları "Bluespec Codesign" adlı dil kullanılarak üretilmiştir; ancak önerilen yöntem yine DT yapısını içermemekte ve tek bir özel platform için MDE desteğini sağlamaktadır. Bizim çalışmamızda önerilen yöntem ile farklı mikroişlemci mimarileri için DT yazılımının geliştirilmesi mümkündür. [8]'de Unix benzeri sistemler için DT içermeyen sürücü kodları üretilmektedir. MOPCOM [9] adlı yöntem gömülü sistemler için UML modellerini oluşturma kurallarını tanımlamaktadır. Bu yöntemde soyut model, çalışma modeli ve ayrıntılı modelleme düzeyleri tanımlanmıştır. Çalışmanın MDE yaklaşımı, DT yapısı ve ilişkili yazılım geliştirme sürecini desteklememekle birlikte, soyutlama düzeyleri ve uygulaması bizim çalışmamıza benzemektedir.

DT yazılım geliştirme çalışmaları dikkate alındığında, [10]'da, DT yazılımında düğüm tipleri, hiyerarşisi, sözdizimi konuları anlatılmış ve bir platformda DT'nin kullanımına ilişkin deneysel bir çalışma yapılmıştır. Buna ek olarak, bir sanal makine ve Çevresel Birim Arabağlantı (PCI) arayüzlerinin oluşturulmasında DT'nin kullanımını açıklayan çalışmalar [11, 12] vardır. Ancak bu çalışmaların hiçbiri DT yazılımı modellemeyi içermemektedir. [13]'te DT, Alan Programlanabilir Kapı Dizileri (FPGA) tasarımında belirtilmiş olmasına rağmen, modelleme ve/veya otomatik üretim için bir yaklaşım yoktur. Bir DT derleyicisinin, "Altera SoC EDS" [14] adlı bir üründe kullanıldığı görülmüştür ancak üretim yalnızca Linux'un tek bir çekirdek sürümü için mümkündür ve bizim çalışmamızda hedeflenen farklı platformları destekleyen genel bir yapıya sahip değildir. [15]'teki çalışmada, donanım sürücü kodu oluşturmak için IEEE 1685-2014 IPXACT standardını [16] kullanan

gömülü sistemler için bir yöntem önerilmiştir. DT üretiminin de yapıldığına değinilmiştir ancak yöntem, kapsam ve deneysel sonuç bölümlerinde DT üretimi ile ilgili bilgi elde edilememiştir. Üstelik bu çalışmadaki DT desteği, bizim önerdiğimiz modelleme süreci ile karşılaştırıldığında oldukça sınırlıdır. Bizim çalışmamızda, [15]'te desteklenmeyen USB, SPI, I2C gibi birçok farklı arayüz için bileşenlerin modele dayalı geliştirilmesi dikkate alınmıştır. Son olarak, [17]'de, gömülü bir sistem platformu için DT yazılımlarının nasıl geliştirilebileceği anlatılmış; model-güdümlü DT geliştirme ihtiyacı vurgulanmıştır.

7 Sonuç ve İleriye Yönelik Çalışmalar

Gömülü sistemlerde kullanılan DT yazılımlarının geliştirilmesi için bir üstmodel bu bildiride tanıtılmıştır. Üstmodel, DT yazılımlarında ihtiyaç duyulan tüm üst-varlıkları ve ilişkilerini çeşitli aygıt bakış açıları bünyesinde içermektedir. Üstmodelin kullanımı ile uluslararası "Devicetree" standartlarına göre DT yazılımlarının geliştirilmesi mümkündür. Üstmodel DT modellemede kullanılabilecek bir görsel modelleme ortamı için de somut sözdizimin oluşturulmasına imkân vermiştir. Eclipse tabanlı bir modelleme aracı içerisinde bu üstmodel ve sözdizim kullanılarak gömülü sistem yazılımları için DT modellemesi yapılabilmektedir. İlgili literatür göz önüne alındığında bu bildiride tanıtılan çalışmanın farklı mikroişlemci mimarisine sahip sistemler için DT yazılımlarının modellenmesine yönelik ilk çaba olduğu görülmektedir.

Bu bildiride tanıtılan üstmodel ve sözdizimler kullanılarak şu an DT yazılımlarının modellenmesini ve otomatik üretilmesini sağlayacak tam teşekküllü bir alana-özgü modelleme dilinin (DSML) geliştirilmesi çalışmaları devam etmektedir. Bu DSML için DT modellerinden Şekil 1'de örneklenen metinsel gösterime uygun bir şekilde gömülü sistem DT yazılımı ve konfigürasyonlarını otomatik üretecek bir işletimsel semantik yapısı üzerinde çalışılmaktadır. Eclipse üzerinde Acceleo açık-kaynaklı kod üreticisi kullanılarak bu semantiğin oluşturulmasına başlanmıştır. Ayrıca geliştirilecek DSML'in çeşitli endüstriyel uygulamalarda kullanımını içeren deneysel çalışmaların farklı gömülü sistem uygulaması geliştiren firmalarda yapılması ve dilin nitekliksel ve niceliksel değerlendirmelerin tamamlanması hedeflenmektedir.

Teşekkür

Bu çalışma TÜBİTAK ARDEB-EEEAG tarafından desteklenen 117E553 no'lu "Aygıt Ağacı Yazılımlarının Farklı Gömülü Sistem Platformları için Model Güdümlü Geliştirilmesi" başlıklı proje kapsamında gerçekleştirilmiştir.

Kaynaklar

1. Petazonni, T.: Device Tree, https://events.linuxfoundation.org/sites/events/files/slides/petazzoni-device-tree.pdf, last accessed 2018/06/12.

2. Devicetree Community, The Devicetree Specification, https://www.devicetree.org/, last accessed 2018/06/12.
3. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice: Second Edition. Morgan & Claypool Publishers (2017)
4. The Eclipse Sirius Modelling Project, 2013, http://www.eclipse.org/sirius/, last accessed 2018/06/12.
5. Chen, H., Godet-Bar, G., Rousseau, F., Petrot, F.: Me3D: A model-driven methodology expediting embedded device driver development. In proceedings of 22th IEEE International Symposium on Rapid System Prototyping, 171-177 (2011).
6. Chen, H., Godet-Bar, G., Rousseau, F., Petrot, F.: Device driver generation targeting multiple operating systems using a model-driven methodology. In proceedings of 25th IEEE International Symposium on Rapid System Prototyping, 30-36 (2014).
7. King, M., Dave, N., Arvind: Automatic generation of hardware/software interfaces. ACM SIGPLAN Notices 47(4): 325-336 (2012).
8. Katayama, T., Saisho, K., and Fukuda, A.: Prototype of the device driver generation system for UNIX-like operating systems, In proceedings of the International Symposium on Principles of Software Evolution (2000).
9. Lecomte, S., Guillouard, S., Moy, M., Leray, P. and Soulard, P.: A co-design methodology based on model driven architecture for real time embedded systems. Mathematical and Computer Modelling 53(3-4), 471-484 (2011).
10. Likely, G. and Boyer, J.: A Symphony of Flavours: Using the device tree to describe embedded hardware. In proceedings of 2008 Linux Symposium, Volume Two, pp. 27-37 (2008).
11. Nikkel, B.: NVM express drives and digital forensics. Digital Investigation 16, 38-45 (2016).
12. Devigne, C., Brejon, J.-B., Meunier, Q., L., Wajsbürt, F.: Executing secured virtual machines within a manycore architecture. Microprocessors and Microsystems 48, 21-35 (2017).
13. Nicolescu, G. and Mosterman, P. J.: Model-based Design for Embedded Systems. Taylor & Francis Group, USA (2010).
14. Rocketboards**:** Golden System Reference Design (GSRD), https://rocketboards.org/foswiki/view/Documentation/DeviceTreeGenerator, last accessed 2018/06/12.
15. Jassi, M., Sharif, U., Müller-Gritschneder, D., Schlichtmann, U.: Hardware-accelerated software library drivers generation for IP-centric SoC designs. In proceedings of the 26th Great Lakes Symposium on VLSI, 287-292 (2016).
16. IEEE Standard 1685-2014, IEEE Standard 1685-2014 for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows (2014).
17. Arslan, S., Türk E., ve Kardaş, G.: Gömülü Sistem Yazılımlarının Geliştirilmesinde Aygıt Ağacı Yapısının Kullanılmasına Yönelik bir Çalışma, 2. Uluslararası Bilgisayar Bilimleri ve Mühendisliği Konferansı (UBMK), 882-887, (2017).