Aygıt Ağacı Yazılımlarının Model Güdümlü Geliştirilmesinin Tersine Mühendislik ile Desteklenmesi

Sadık Arslan[1,2], 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) yapıları bir gömülü sistem donanımının içerisindeki fiziksel cihaz bileşenlerinin düğümler ile tanımlanmasını sağlamaktadır. DT kaynak dosyalarının metin-tabanlı ve güncel programlama dillerinden farklı bir yapıdaki sözdizimlerini kullanmadaki zorluğu aşmak ve çeşitli işlemci mimarileri için DT kodlarını otomatik üretmek için model güdümlü geliştirme kullanılabilir. Ancak bazı durumlarda üretilen DT'ler üzerinde yeni donanım ekleme, donanım özelliği değiştirme, vb. amaçlarla kod ekleme veya değiştirme gerekmektedir. Bu bildiride DSML4DT adı verilen bir alana-özgü modelleme dili kullanılarak otomatik oluşturulan DT kodlarında sonradan yapılan değişikliklerin modellere yansıtılmasını ve daha önce başka yöntemlerle oluşturulmuş DT kodlarından yazılım modellerini elde etmeyi sağlayan bir tersine mühendislik yöntemi tanıtılmaktadır. Önerilen tersine mühendislik yaklaşımının toplu taşıma bilgi sistemleri üreten bir firmanın donanımlarında ihtiyaç duyulan DT tabanlı yazılımların geliştirilmesinde uygulanması örnekleri verilmiş ve uygulama sonuçları değerlendirilmiştir.

Anahtar Kelimeler: Model-güdümlü Geliştirme, Alana-özgü modelleme dili, Aygıt Ağacı, Gömülü Yazılım, Tersine Mühendislik.

Reverse Engineering Support for the Model-driven Development of Device Tree Software

Sadik Arslan[1,2], Geylani Kardas[2]

[1] Kent Kart R&D Center, Kent Kart Ege Elektronik A.S., Izmir, Turkey
[2] Ege University, International Computer Institute, Izmir, Turkey
sadik.arslan@kentkart.com.tr, geylani.kardas@ege.edu.tr

Abstract. Device Trees (DTs) provide the description of physical devices inside an embedded system hardware with node specifications. Model-driven development may be used for dealing with the difficulties encountered in DT de-

velopment due to the text-based syntax of DT source files which has a different structure from the well-known general purpose programming languages; and hence also enables automatic DT code generation for various processor architectures. However, in some situations, there needs later modifications on the generated DT codes for adding new devices, changing device specifications, etc. In this paper, we introduce a reverse engineering method which supports both 1) reflecting the changes made in DT codes auto-generated by using a domain-specific modeling language, called DSML4DT, into models and 2) generating models from the existing DT codes produced with any other methods. The application of the proposed approach is exemplified and evaluated with the implementation of DT based software for the various hardware manufactured by a company whose main expertise is on the public transportation systems.

Keywords: Model-driven Development, Domain-specific modeling language, Device Tree, Embedded Software, Reverse Engineering.

1 Giriş

Donanım bilgisi ve yapılandırmasını içeren Aygıt Ağacı (DT) dosyaları [1] çeşitli gömülü platformlar için işletim sistemlerinin derlenmesi sırasında kullanılırlar. DT'ler bir gömülü sistem donanımının içerisindeki fiziksel cihaz bileşenlerinin düğümler ile tanımlanmasını sağlamaktadır [2]. Ancak gömülü sistem yazılımı geliştiricileri DT kaynak dosyalarının metin-tabanlı ve güncel programlama dillerinden farklı bir yapıdaki sözdizimlerini kullanmada çoğu zaman zorlanmaktadır. Yeni her gömülü platform için ihtiyaç duyulan DT dosyaları baştan sona ve ayrı ayrı hazırlanmalıdır. Bununla birlikte geliştiricilerin DT dosyalarını hazırlaması için mikro-işlemciye özel donanımlara da hakim olması gerekmektedir. İlgili donanımlar hakkında alan bilgisine sahip ancak yazılım geliştirme bilgi ve deneyimi az olan mühendisler için bu dosyaların hazırlanması aşaması olduça zahmetlidir. Üstelik farklı mikro-işlemci mimarileri ve çeşitleri için DT yazılımlarına ait bileşenlerin kodlanması ve konfigüre edilmesi yine birçok geliştirici için zor ve zaman alıcı olmaktadır. Literatürde DT tabanlı gömülü sistem yazılımı geliştirmeye dair çeşitli çalışmalar [3-9] bulunmakla birlikte bu çalışmalar daha çok DT konfigürasyonu oluşturmanın örneklerini sunmakta ve yukarıda sözü edilen DT yazılımı geliştirme zorluklarının giderilmesini göz önüne almamaktadır.

Model güdümlü geliştirme teknikleri DT yazılımlarına ait kaynak dosyalarının farklı işlemci mimarileri ve çeşitleri için otomatik üretilmesini sağlayabilir ve yine yukarıda listelenen yazılım geliştirme süreci zorluklarını azaltabilir. Bu amaçla daha önceki çalışmamızda [10] DT yazılımlarının modellenmesi için kullanılabilecek bir üstmodel ve somut bir sözdizim geliştirilmiştir. Bu sözdizimi kullanan ve bir dönüşümsel semantik ile DT modellerinden gömülü sistemlerde ihtiyaç duyulan DT yazılımlarının otomatik elde edilebildiği DSML4DT adı verilen bir alana-özgü modelleme dili (DSML) oluşturulmuştur.

Her ne kadar DSML4DT ile DT yapılarının tüm bakışaçılarına hitap eden modeller oluşturulabiliyor ve bu modellerden kodlar üretilebiliyor olsa da DSML4DT'nin endüstriyel boyutta kullanımları sırasında bazı gömülü sistem donanımları için

oluşturulan bu otomatik kodlara yeni eklemelerin ya da değişikliklerin yapılmasının gerektiği durumların da oluştuğu gözlenmiştir. Bazı durumlarda da DT kodunun önceden bir yazılım modeli yoktur ancak bu kodların DSML4DT ile yazılım geliştirme sürecine entegre edilmesi için modellere ihtiyaç duyulabilmektedir.

Yazılımların tersine mühendisliği yazılımların yönetilmesine ve zaman içerisindeki değişimlere göre evrimleşmesine yardımcı olmaktadır [11]. Kaynak kodlar incelenerek yazılım sistemlerinin ana elemanları belirlenebilir ve yazılımın önceden tanımlanmış bir düzeydeki soyutlaması elde edilebilir [12]. Bu bildiride, hem DSML4DT'nin kod üretimi mekanizması sonucunda oluşturulmuş DT kodlarındaki değişikliklerin DSML4DT modelerine yansıtılmasını hem de daha önce başka yöntemlerle oluşturulmuş DT kodlarından yazılım modellerini elde etmeyi sağlayan bir tersine mühendislik yöntemi tanıtılmaktadır.

Bildirinin 2. bölümünde DT yapıları ve DSML4DT dili hakkında kısaca bilgi verilmiştir. DT yazılımlarının geliştirilmesi için önerilen tersine mühendislik yöntemi 3. bölümde anlatılmıştır. Tersine mühendislik yaklaşımının ve geliştirilen aracın kullanılmasını örnekleyen bir durum çalışması 4. bölümde yer almaktadır. Bölüm 5'te, ilgili literatürdeki önceki çalışmalar anlatılmış ve önerilen yöntemin farkları belirtilmiştir. Son bölümde çalışmanın bir değerlendirmesi, elde edilen sonuçlar ve ileriye yönelik çalışma hedefleri bulunmaktadır.

2 DT Yapıları ve DSML4DT

Bir DT, donanım tanımı sağlamaktadır ve her aygıtın düğümler şeklinde ve özellikleri ile birlikte temsil edildiği bir ağaç yapısına sahiptir. DT, çalışılan gömülü sistemin işlemcisi, belleği, veri yolları ve çevre birimleri gibi birçok bölüm hakkında bilgi içerir. İşletim sistemi, önyükleme (ing. bootloading) sırasında DT yapısını ayrıştırır ve mikroişlemciyi nasıl yapılandıracağını burada belirler. DT yapısı, "/" karakteri ile gösterilen ve kök olarak adlandırılan bir düğümle başlamaktadır. Ağaç yapısına benzer bir şekilde, DT'de ebeveyn konumundaki her düğümden çok sayıda buluan, ismi ve numarası olan birden fazla çocuk düğüm oluşturulabilmektedir. Düğümler isteğe göre, ek veri içeren nitelik değerlerine sahip olabilmektedir. Metinsel hazırlanan DT tanımları, ses kartı, ekran kartı gibi birçok farklı arayüz ve özelliğin ayarlanması için elle hazırlanması gereken çok sayıda koddan oluşmaktadır [6].

DSML4DT, uluslararası DT standardına [1] uygun olarak gömülü sistem DT yazılımlarının model-güdümlü geliştirilmesini sağlamaktadır. DSML4DT üstmodeli toplamda 70'ten fazla DT bileşeni ve bunların ilişkilerini içermektedir. Üstmodel farklı DT modelleme bakış açılarına göre modellemeyi sağlayan 5 alt üstmodelin birleşiminden oluşmaktadır. Bunlar: Temel işlemci ve bellek birimlerinin tanımlandığı *Core*, yonga üzeri sistem entegre devrelerinin özelliklerinin modellenebildiği *SoC*, düşük bant genişliğine sahip SoC çevresel bileşenlerinin hazırlanabildiği *Aips_Bus*, harici birimlerlerle iletişimi paylaşımlı çevresel hat arayüzü üzerinden kurgulayan *Spba_Bus* ve yonga üzerinde bulunmayıp yongaya dışardan bağlı çevresel birimlerin modellenebilidği *Peripheral*.

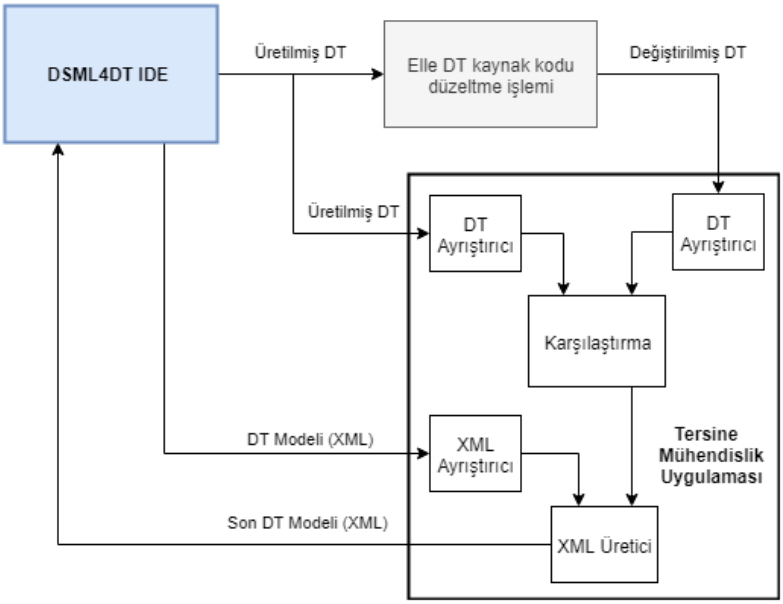
Yukarıda isimleri verilen DT bakışaçılarının her birindeki üst-varlıklar için DSML4DT'de görsel somıt sözdizim notasyonları vardır ve her bir bakışaçısı için ayrı ayrı sistem modelleri görsel bir şekilde oluşturulabilmektedir. DSML4DT'nin entegre geliştirme ortamı (IDE) çeşitli model-güdümlü araçların geliştirilmesinde yaygın olarak kullanılan Eclipse tabanlı Sirius platformu üzerine kurgulanmıştır. DSML4DT IDE'sinde yer alan bir palette her bakış açısı için görsel notasyonlar listelenmektedir. Yazılım geliştiriciler ilgili paletten sürükle-bırak yöntemi ile kendi DT modelleri için gerekli elemanları IDE'deki modelleme ortamına getirirler. Modelleme sırasında DSML4DT'nin statik semantik kuralları ve diğer model kısıtları doğrudan işletilir ve yanlış bir model oluşturma durumunda geliştirici uyarılır. Model doğrulama tamamlanmadan modeller XMI ile serileştirilerek saklanamaz ve kod üretimi yapılmaz. Oluşturulan görsel DT modellerinden otomatik DT kodlarının üretimi Sirius'a entegre Acceleo'da yazılan modelden metne kuralların işletilmesi ile sağlanmaktadır. Her bir DSML4DT model örneği üzerinde Accelo dili kullanılarak yazılan dönüşüm kuralları işletilir ve sisteme ait DT kodları .dts dosya formatında oluşturulur. Yer kısıtları nedeniyle bu bildiride gösterilemeyen DT yapısı örneğine, DSML4DT'nin sözdizimine ve IDE'nin görünümüne [10]'dan erişilebilinir.

3 DSML4DT için Tersine Mühendislik Yöntemi

Mevcut DT yazılımlarından DSML4DT'ye uyan DT modellerinin otomatik oluşturulması için bir yöntem bu çalışmada önerilmektedir. Geliştirilen tersine mühendislik yaklaşımının hayata geçirilebilmesi için de Java dilinde yazılmış bir uygulama geliştirilmiştir. Bu uygulama temelde bir ayrıştırma (ing. parse) işlemi yapmaktadır. DSML4DT IDE'si kullanılarak hazırlanan her bir görsel DT modeli DSML4DT semantik kurallarına göre geçerliliği kontrol edildikten sonra Genişletilebilir İşaret Dili (XML) formatındaki bir dosya içerisinde saklanmaktadır. DT yazılımları için tersine mühendisliği sağlayan bu uygulama DT kod dosyalarını ve XML formatında saklanan DT model dosyalarını girdi olarak almakta ve çıktı olarak yine DT model dosyaları üretmektedir. Ayrıştırma uygulaması hem DSML4DT ile önceden oluşturulmuş kodlar (senaryo 1) hem de herhangi bir DT modeli bulunmayan ve DSML4DT ile geliştirilmemiş kodlar üzerinde (senaryo 2) çalışarak DT yazılım modellerini otomatik oluşturabilmektedir. Uygulama bu bildiri hazırlandığı tarihte DSML4DT IDE'sine entegre değildir ve IDE'den bağımsız olarak çalışmaktadır.

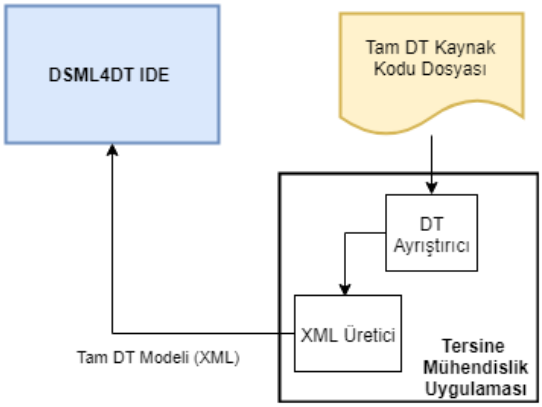
Uygulama, DSML4DT tarafından üretilmiş olan yazılım kodlarını içeren dosyaları aldığında öncelikle bunlar üzerinde bir ayrıştırma yapmaktadır. Ağaç yapısındaki DT dosyalarında düğümlerin bulunması gerekmektedir. Ebeveyn ve çocuk düğümlerin ilişkisi DT yapısının temelini oluşturmaktadır. Bu düğümler DT'nin sözdiziminde "{" ve "}" karakterleri arasında bulunmaktadır. Çocuk düğümler de üst ebeveyn düğümün içinde yine bu karakterler arasına yerleşmektedir. Uygulama temelde dosyalardaki karakterleri tek tek taramakta ve bu karakterlere göre düğümleri oluşturmaktadır. Ağaç yapısı, bulunan düğüm sınırlarına ve ayrıştırılan düğüm isimlerine göre oluşturulmaktadır. DT kaynak kodu üzerinde yapılan bu ayrıştırmadan sonra DSML4DT modeline ait XML formatındaki dosya üretilmektedir.

Senaryo 1'de DSML4DT ile modellenen ve bu modelden üretilen DT yazılım kodlarında daha sonra yazılım geliştiricinin manuel yaptığı değişikliklerin (yeni DT elemanları için kod ekleme, mevcut DT yapısına ait kodu günceleme, vb.) yazılım modeline yansıtılması göz önüne alınmaktadır (Şekil 1). Mevcut DSML4DT örnek modelinden otomatik elde edilen (değiştirilmemiş) DT kodları ve manuel değiştirilmiş DT kodları yukarıda tarif edilen uygulama ile ayrıştırılır. Ayrıca uygulama mevcut görsel modelin XML formatında saklanan versiyonu üzerinde de benzer bir ayrıştırma uygular. Ayrıştırılan kaynak kodu dosyaları (otomatik ve manuel değiştirilen) yine bu uygulama ile karşılaştırılır ve ağaç yapıları belirlenir. İki dosya arasındaki farka göre de mevcut DSML4DT örnek modeli XML dosyasındaki değişiklikleri yaparak güncellenmiş DT modelini oluşturmaktadır. Bu yeni modele ait XML dosyası DSML4DT IDE'sinde açıldığında yazılımcının manuel yaptığı değişikliklere göre güncellenmiş DT modelinin grafiksel (görsel) hali görüntülenebilmektedir.



Şekil 1. DT yazılımları için geliştirilen tersine mühendislik uygulaması – Senaryo 1

Bu çalışmada geliştirilen tersine mühendislik yaklaşımının uygulanabileceği 2. senaryo ise DT yazılım modeli halihazırda olmayan ve/veya DSML4DT dili kullanılmadan oluşturulan DT kaynak dosyalarına karşılık gelen yazılım modellerinin otomatik oluşturulmasıdır. Herhangi bir DT kaynak kodu yine uygulamanın ayrıştırıcı kısmından geçirilerek bu dosyada tanımlı tüm DT yapılarının ve özelliklerinin görsel gösterimini sağlayacak model dosyasının XML ile "encode" edilmiş hali otomatik oluşturulmaktadır. Bu dosya DSML4DT IDE'sinde açıldığında model yine dilin görsel notasyonu ile yazılım geliştiricilere gösterilir. Böylece başka araçlarla ya da yöntemlerle oluşturulmuş DT dosyaları da DSML4DT ortamına dahil edilebilir.



Şekil 2. DT yazılımları için geliştirilen tersine mühendislik uygulaması – Senaryo 2

Uygulamanın çalışmasındaki temel Java programı *DTS_RevEn* olarak adlandırılmıştır. Bu programda öncelikle dışarıdan girilen otomatik üretilmiş ve elle değiştirilmiş DT kaynak dosyaları alınmaktadır. Bu programda kullanılan *dtsParse* objeleri ayrıştırma işlemlerini yapmaktadır. Programa alınan DT kaynak kodu dosyaları içerisinde bir döngü aracılığı ile satır satır gezilerek düğümler belirlenir ve nesne koleksiyonlarına dahil edilir. Bu koleksiyon nesnelerinde, girdi konumundaki otomatik üretilmiş DT kodları ve üzerinde değişiklik yapılan (manuel) kodlar arasında karşılaştırmalar ve diğer işlemler kolaylıkla yapılabilmektedir. *listGen* ve *listMod* isimli ve Java *Collection* tipinde oluşturulmuş dizilerde DT dosyalarının düğümlerinin isimleri tutularak işlemler yapılmaktadır. Programın *dtsParse* sınıfından türetilen *dtsGen* ve *dtsMod* nesneleri ile *phraseGen* ve *phraseMod* karakter değişkenleri kullanılarak ayrıştırma yapılmaktadır. *findParanthesis* metodu ile DT kaynak dosyasındaki düğüm parantezleri bulunmaktadır. Daha sonra kullanılan *findNodes* metodu ile DT kaynak kodlarındaki düğümler bulunmaktadır. *generatedList* ve *modifiedList* dizi listeleri üzerinden de elle eklenen DT kodlarının farkı *removeAll* metodu ile belirlenmektedir. Bu işlemlerden sonra *generateXML* metotu ile DSML4DT IDE'sinde açılabilecek DT modeli XML dosyası üretilmiş olmaktadır. Şekil 3'te uygulamanın yukarıda anlatılan temel adımlarını tesmil eden basit sözde kod (ing. pseudocode) verilmiştir. Senaryo 1 için Şekil 3'te listelenen tüm adımlar uygulanmaktadır. Senaryo 2 için sadece satır 2, satır 8-10 ve satır 13'ün işletilmesi yeterlidir. *generateXML* metodu ayrıştırılmış olan DT kaynak kodlarından farka göre yeni DT modelinin XML'ini oluşturmaktadır. Bu amaçla DOM XML ayrıştırıcı ve üreticiyi kullanır. Burada yapılan işlemlerde mevcut DT modeli ayrıştırılmakta ve ilgili yerlere yeni XML satırları eklenmektedir. Her XML satırı bir DT düğümüne denk gelmektedir. Tüm DT dosyasının oluşturulması benzer mantıkla yapılmaktadır. DT'nin ağaç yapısı üzerinde gezilirken yeni XML dosyası da oluşturulmaktadır. Oluşturulan XML dosyası DSML4DT içerisinde açılabilir ve işletilebilir bir haldedir.

```
01  Adding generated DTS file
02  Adding modified DTS file
03
04  Parsing generated DTS()
05      Finding Paranthesis(generated DTS file)
06      Finding Nodes(generated DTS file)
07
08  Parsing modified DTS(modified DTS file)
09      Finding Paranthesis(modified DTS file)
10      Finding Nodes(modified DTS file)
11
12  Printing different nodes(delta nodes)
13  Generating XML file
```

Şekil 3. DT tersine mühendislik uygulaması sözde kodu

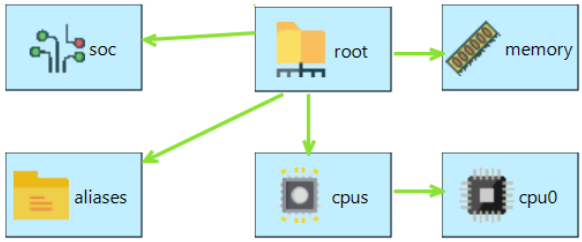
4 Durum Çalışması

Bu bildiride önerilen tersine mühendislik yaklaşımı toplu taşımada kullanılan bir araç sürücü terminaline ait gömülü sistemin DT tabanlı geliştirilmesi için uygulanmıştır. Uygulama T.C. Sanayi ve Teknoloji Bakanlığı'nca 2013 yılından beri akredite olan Kent Kart Ege Elektronik A. Ş. (kısaca Kentkart) firmasının Ar-Ge merkezinde gerçekleştirilmiştir. Kentkart otomatik ücret toplama, araç takibi ve yönetimi ve gerçek zamanlı yolcu bilgilendirme gibi çeşitli alanlarda toplu taşıma bilgi sistemleri üreten bir firmadır. Kentkart aynı zamanda toplu taşımada kullanılan ücret toplama makineleri, turnike validatörleri ve araç sürücüsü terminalleri gibi donanımları da üretmektedir. Tüm bu donanımlar üzerinde çalıştırılan yazılımların önemli bir kısmı DT tabanlıdır ya da DT yapılarına göre konfigüre edilmektedir.

Çalışmada kullanılan araç sürücüsü terminali Kentkart firması tarafından üretilen ve çift çekirdekli iMX6 işlemcisine sahip bir gömülü sistemdir. Bu tip bir cihazın DT yapısı tasarlanırken toplamda 100 civarında farklı özelliği içeren DT düğümünün hazırlanmasına ihtiyaç vardır.

Bu örnek çalışmada söz konusu sürücü terminalinin DSML4DT kullanılarak oluşturulan Core bakış açısı modelinden elde edilen otomatik kodlarda Kentkart mühendislerinin manuel gerçekleştirdiği bir değişikliğin bu bildirinin 3. bölümünde anlatılan yöntem kullanılarak sistemin DT modeline nasıl yansıtıldığı göz önüne alınacaktır. DSML4DT Core bakış açısında bir gömülü sistemdeki işlemci ve bellek gibi temel blokları tanımlayan düğümler ve bunların birbirleri ile ilişkileri modellenir. Burada bulunan *root* düğümü tüm DT yapısının türetildiği düğümdür ve bu düğümden işlemciler için *cpus*, ana bellek için de *memory* düğümleri türetilmektedir. Sistemin geri kalan tüm özellikleri de bu bakış açısındaki *soc* düğümünde bulunmaktadır. Ancak bu bakış açısında *soc* düğümüne bağlı olan çocuk düğümler görülememektedir. Bu düğümler farklı bir bakış açısı olan Soc bakış açısında bulunmaktadır. DSML4DT'nin bu bakışaçıları için detaylı bilgi [10]'da bulunmaktadır.

DSML4DT dili kullanılarak araç sürücüsü terminali için oluşturulan örnek Core modelinin ilk hali Şekil 4'te verilmiştir. Sistemde işlemci çekirdekleri *cpus* ebeveyn düğümünden türetilmektedir. Bu DT modelinde şu an sadece *cpu0* adında bir çekirdek olduğu görülebilmektedir.



Şekil 4. Araç sürücü terminali DT yapısı Core modelinin ilk hali

Şekil 4'teki DT modelinden otomatik kod üretimi yapıldığında DSML4DT IDE aracılığıyla Şekil 5'te satır 1-26 arasında görülen DT kaynak kodları elde edilmektedir. Buradaki kaynak kodunda "/" karakteri ile başlayan yerde *root* düğümü bulunmaktadır. *root* düğümünün *model* ve *compatible* isminde öznitelikleri vardır. Ardından gelen *cpus* düğümü *root* düğümünün çocuğudur. *cpu0* düğümü de *cpus* düğümünün çocuğu durumundadır. Otomatik üretilen bu koda yazılım geliştirici daha sonra ikinci bir cpu tanımını manuel eklemiştir. Şekil 5'te koyu olarak verilen (satır 27-32 arasındaki) kodlar *cpu1* adlı bu düğüm için manuel eklenen kodlardır.

```
01  /{
02      model = Kentkart i.MX6 DualLite Smart Device Board;
03      compatible = "fsl,imx6dl-sabresd", "fsl,imx6dl";
04
05      cpus{
06          address-cells = <1>;
07          size-cells = <0>;
08
09          cpu0{
10              compatible = "arm,cortex-a9";
11              device_type = "cpu";
12              reg = <0>;
13              next-level-cache = <&L2>;
14              operating-points = <
15                  996000 1275000
16                  792000 1175000
17                  396000 1150000
18                  >;
19              fsl,soc-operating-points = <
20                  996000 1175000
21                  792000 1175000
22                  396000 1175000
23                  >;
24              clock-latency = <61036>;
25              };
26
27          cpu1{
28              compatible = "arm,cortex-a9";
29              device_type = "cpu";
30              reg = <1>;
31              next-level-cache = <&L2>;
32              };
33      };
```

Şekil 5. Otomatik üretilmiş ve elle eklenmiş DT kod örneği

```
01  <devicetree_viewpoint:devicetree>
02    <root compatible="&quot;fsl,imx6dl-sabresd&quot;, &quot;fsl,imx6dl&quot;"
03        model="Kentkart i.MX6 DualLite Smart Device Board" name="root">
04      <cpus address_cells="1" name="cpus" size_cells="0">
05          <cpu0 clock_latency="61036" clock_names="&quot;arm&quot;,
06              &quot;pll2_pfd2_396m&quot;, &quot;step&quot;,&quot;pll1_sw&quot;,
07              &quot;pll1_sys&quot;, &quot;pll1_bypass&quot;, &quot;pll1&quot;,
08              &quot;pll1_bypass_src&quot;" compatible="&quot;arm,cortex-a9&quot;"
09              device_type="&quot;cpu&quot;" name="cpu0"
10              next_level_cache="&amp;L2"
11              operating_points="996000 1275000 792000 1175000 396000 1150000"
12              reg="0" soc_operating_points="996000 1175000 792000 1175000 396000
13                  1175000"/>
14      </cpus>
```

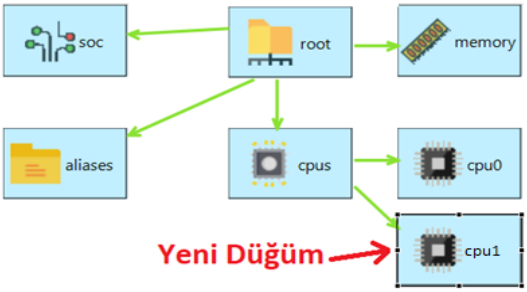
Şekil 6. DSML4DT ile hazırlanan görsel DT modeline ait XML dosyasından bir kısım

Şekil 4'teki DT modelinin DSML4DT IDE'si içerisindeki dosyasında görsel modeldeki bu düğümlerin ve özniteliklerin XML ile serileştirilmiş metinsel halleri tutulmaktadır. Şekil 6'da bu dosyadan *root*, *cpus* ve *cpu0* düğümlerinin yer aldığı bir kısım verilmiştir.

Buradaki uygulama bildirinin 3. bölümünde anlatılan tersine mühendislik yaklaşımının kullanım senaryolarından birincisinin bir örneğidir. Şekil 5'te görülen ve manuel eklemelerin yapıldığı koddaki değişikliğin Şekil 6'daki DT Core modeli XML dosyasına yansıtılabilmesi için geliştirdiğimiz tersine mühendislik uygulaması eklemelerin yapıldığı kod üzerinde çalışarak delta kodları belirlemiştir ve yeni kodlara karşılık gelen tanımları Şekil 6'daki DT modeli XML dosyasına eklemiştir. Buradaki değişiklik DT konfigürasyonuna eklenen *cpu1* düğümü içindir. Uygulama burada, *cpu1* düğümünü fark olarak bulmakta ve XML kodunda da *cpus* düğümü altına *cpu0* düğümünün kardeşi olarak uygun yere *cpu1* düğümünü yerleştirmektedir. Bu işlem sonucunda DT modeli XML dosyası Şekil 7'deki gibi olmaktadır. Dosyada koyu gösterilen kısımlar uygulama tarafından eklenmiştir.

```
01  <devicetree_viewpoint:devicetree>
02    <root compatible="&quot;fsl,imx6dl-sabresd&quot;, &quot;fsl,imx6dl&quot;"
03        model="Kentkart i.MX6 DualLite Smart Device Board" name="root">
04      <cpus address_cells="1" name="cpus" size_cells="0">
05          <cpu0 clock_latency="61036" clock_names="&quot;arm&quot;,
06              &quot;pll2_pfd2_396m&quot;, &quot;step&quot;,&quot;pll1_sw&quot;,
07              &quot;pll1_sys&quot;, &quot;pll1_bypass&quot;, &quot;pll1&quot;,
08              &quot;pll1_bypass_src&quot;" compatible="&quot;arm,cortex-a9&quot;"
09              device_type="&quot;cpu&quot;" name="cpu0"
10              next_level_cache="&amp;L2"
11              operating_points="996000 1275000 792000 1175000 396000 1150000"
12              reg="0" soc_operating_points="996000 1175000 792000 1175000 396000
13                  1175000"/>
14          <cpu1 clock_latency="" clock_names="" clocks="" compatible="&quot;arm,
15              cortex-a9&quot;" device_type="&quot;cpu&quot;" name="cpu1"
16              next_level_cache="&amp;L2" operating_points="" reg="1"
17              soc_operating_points=""/>
18      </cpus>
```

Şekil 7. Uygulama tarafından değiştirilmiş DT modeli XML dosyasından bir kısım



Şekil 8. Araç sürücü terminali DT yapısı Core modelinin değişiklik sonrası görünümü

Şekil 7'de bir kısmı verilen DT modeli artık manuel yapılan değişiklikleri de içerecek şekilde güncellenmiştir. İlgili XML dosyası DSML4DT IDE'si içerisinde açıldığında güncelleme görsel modelde de görülebilmektedir. Şekil 8'de verilen bu yeni modelde artık *cpu1* DT düğümü de yer almaktadır. Böylece DT kodunda sonradan yapılan değişikliklerin DT modeline başarılı bir şekilde yantısıldığı ve sistem kodları ile modelin senkronize edilebildiği görülmüştür.

5 İlgili Çalışmalar

Literatürde DT'ler ve DT'lere bağlı sistem konfigürasyonlarının kullanıldığı çalışmalar bulunmaktadır. Örneğin donanım konfigürasyonlarının cihaz kayıtçılarının program içeriğinden ayrıştırılması [3], BTFRS gibi Linux dosya sistemleri dahilinde çoklu cihaz desteğinin sağlanması [4], heterojen Nesnelerin İnterneti ağlarında geçit görevi üstlenecek bazı anakart destek paketlerinin mikroişleyiciler için optimizasyonu [5], bulut hesaplama sistemlerindeki sanallaştırılma katmanları dahilinde bilgisayar fiziksel belleklerinin misafir işletim sistemleri tarafından belirlenmesi [7], trafik işareti tanımlamalarında kullanılacak gömülü sistemlerin inşaası [8] ve gerçek-zamanlı sağlık takibi cihazlarının üretilmesi [9] amacıyla DT yazılımları geliştirilmiştir. Ancak bu çalışmalar birer DT uygulaması örneğidir ve DT yazılımı modellemeyi ve/veya otomatik DT kodu üretimini göz önünde bulundurmamaktadır.

"Altera SoC EDS" isimli üründe [13] bir DT derleyicisi yer almakla birlikte kod üretimi sadece tek bir Linux çekirdeği için mümkündür. Neuendorffer [14] model tabanlı yaklaşımların Alan Programlanabilir Kapı Dizileri (FPGA) içindeki sistemlerin tasarlanmasında kullanılmasından bahsetmiş ve sistem tasarımlarından DT'lerin oluşturulabileceğini belirtmiştir. İlgili tasarım akışı, sunulan bir durum çalışması üzerinden örneklenmekle birlikte DT yapılarının otomatik oluşturma süreci çalışmada belli değildir. Jassi vd. [15] yazılım uygulamalarının Linux üzerinde çalıştırılması durumunda geliştirdikleri GRIP isimli aracın DT kaynak dosyalarını da üretmesinin mümkün olduğunu belirtmişlerdir ancak bununla ilgili bir uygulama örneğine çalışmalarında rastlanmamaktadır. Bu bildiride tanıtılan tersine mühendislik yaklaşımı ile literatürdeki bu kısıtlı sayıdaki DT yazılımlarının otomatik üretilme ve model-güdümlü geliştirilmesi çalışmalarına üretilen kodlardaki değişikliklerin DT yazılım modellerine yansıtılması yönüyle bir katkı verildiğine inanılmaktadır.

Bildiğimiz kadarıyla mevcut çalışmalardan hiçbiri DT kodları ve modelleri arasındaki senkronizasyonu sağlamak için bir yöntem sunmamaktadır.

6 Değerlendirme ve Sonuç

Bu çalışmada, DT yazılımlarının model-güdümlü geliştirilmesi sırasında mevcut yazılım kodlarından DT modellerinin otomatik olarak geri elde edilmesini sağlayacak bir tersine mühendislik yöntemi tanıtılmıştır. Bu yöntemin DSML4DT adlı bir DSML ve bu dile ait IDE ile kullanılması amacıyla hazırlanan bir uygulamanın örneklenmesi de endüstriyel bir durum çalışması üzerinden gösterilmiştir. Bildiride tanıtılan uygulama kullanılarak DT yazılımı kaynak kodlarının ayrıştırılması yapılmakta ve DSML4DT ile uyumlu görsel DT modellerine ait tasarımları içeren XML dosyaları DT yazılımı kodlarında yapılan değişiklikleri içerecek şekilde otomatik olarak üretilmektedir. Böylece DT yazılımları ile bunların DSML4DT modelleri arasındaki senkronizasyon sağlanabilmektedir.

Bu çalışmada, detayları verilen araç sürücüsü terminali sistemi durum çalışmasına ek olarak Kentkart firmasında geliştirilmiş olan farklı gömülü sistem cihazları üzerinde çalışan çeşitli DT yazılımları için de DT kodlarındaki değişikliğin DSML4DT ile oluşturulan sistem modellerine yansıtılması sağlanmıştır. Değerlendirme amacıyla bu firma tarafından geliştirilen validatör (ücret düşüm) cihazı, araç içi kamera sistemi kontrol cihazı ve araç içi reklam görüntüleme cihazlarının hazır DT dosyalarından tersine mühendislik uygulaması ile DT görsel modellerinin sorunsuzca elde edilebildiği görülmüştür. Bununla birlikte bu çalışmada geliştirdiğimiz uygulama yine bu cihazlarda elle değiştirilmiş olan DT kodlarındaki değişiklikleri de bu cihazlara ait DSML4DT örnek DT modellerine başarılı bir şekilde yansıtabilmiştir. Bölüm 4'teki durum çalışmasında gösterilen işlemci düğümünden farklı olarak bellek, entegre arası devre (i2c), seri çevresel arayüz (spi), yüksek çözünürlüklü çoklu ortam arayüzü (hdmi) gibi birçok farklı DT düğümü için de geliştirdiğimiz uygulama kodlar ve görsel modeller arasındaki senkronizasyonu sağlayabilmektedir.

Kentkart firmasında yapılan örnek çalışmalarda yazılım geliştiricilerin DSML4DT kullanarak oluşturduğu DT yazılımlarında sonradan yaptıkları değişiklerin daha çok DT yapısına yeni düğüm eklemek ve/veya düğüm içlerindeki özniteliklerde güncellemelerde bulunmak olduğu tespit edilmiştir. Bu çalışmada tanıtılan uygulama bu değişiklikler için tersine mühendisliği tam olarak sağlamaktadır. Bununla birlikte bu bildirinin 3. Bölümünde tarif edilen 2. senaryodaki gibi daha önceden herhangi bir yazılım modeli bulunmayan DT dosyalarından da yazılım modellerinin elde edilebildiği görülmüştür. Ancak bu kodlardan daha önce DSML4DT kullanılmadan başka yöntemlerle üretilmiş olanlarının bir kısmında görsel modellere ait XML dosyalarının oluşturulmasında eksikliklerin olduğu gözlenmiştir. Bunun başlıca nedeni bu tip kodların DSML4DT bakış açılarına tam uyumlu olmayan tanımlamaları içermesi ve bunların ayrıştırıcı uygulamanın çalışması sırasında DSML4DT görsel somut sözdizimindeki karşılıklarının bulunamamasıdır. İleriye yönelik planlanan ilk çalışma bu yöndeki eksikliği tamamlayacak bazı dönüşüm mekanizmalarının

uygulamaya dahil edilmesidir. Ayrıca tersine mühendislik yaklaşımının başka endüstriyel DT yazılımı geliştirme çalışmalarında kullanılması ve daha geniş bir perspektife kullanımının değerlendirilmesi çalışmaları yürütülecektir.

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.

Kaynakça

1. The Devicetree Specification, https://www.devicetree.org/, last accessed 2019/08/19.
2. Madieu, J.: The Concept of a Device Tree, 139-166. Linux Device Drivers Development: Develop customized drivers for embedded Linux. Packt Publishing (2017).
3. Schüpbach, A., Baumann, A., Roscoe, T., Peter, S.: A Declarative Language Approach to Device Configuration. ACM Transactions on Computer Systems, 30(1), Article 5 (2012).
4. Rodeh, O., Bacik, J., Mason, C.: BTRFS: The Linux B-Tree Filesystem. ACM Transactions on Storage, 9(3), Article 9: 1-32 (2013).
5. Gioia, E., Passaro, P., Petracca, M.: AMBER: An advanced gateway solution to support heterogeneous IoT technologies. In proceedings of the 24th International Conference on Software, Telecommunications and Computer Networks, 1-5 (2016).
6. Arslan, S., Türk, E. Kardaş, G.: Gömülü Sistem Yazılımlarının Geliştirilmesinde Aygıt Ağacı Yapısının Kullanılmasına Yönelik bir Çalışma. 2. Uluslararası Bilgisayar Bilimleri ve Mühendisliği Konferansı, IEEE Conference Publications, 882-887 (2017).
7. Bielski, M., Rigo, A., Pacalet, R.: Dynamic Guest Memory Resizing - Paravirtualized Approach. In proceedings of the 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, 181-186 (2019).
8. Farhat, W, Sghaier, S, Faiedh, H, Souani, C: Design of efficient embedded system for road sign recognition. Journal of Ambient Intelligence and Humanized Computing, 10(2), 491-507 (2019)
9. Swaroop, K. N., Chandu, K., Gorrepotu, R., Deb, S.: A health monitoring system for vital signs using IoT. Internet of Things, 5, 116-129 (2019).
10. Arslan, S., Kardaş, G.: Aygıt Ağacı Yazılımlarının Modellenmesi, 12. Ulusal Yazılım Mühendisliği Sempozyumu, CEUR Workshop Proceedings, 2201: 1-12 (2018).
11. Canfora, G., Di Penta, M., Cerulo, L.: Achievements and challenges in software reverse engineering. Communications of the ACM 54(4): 142-151 (2011).
12. Washizaki, H., Gueheneuc, Y.-G., Khomh, F.: ProMeTA: a taxonomy for program metamodels in program reverse engineering. Empirical Software Engineering 23(4): 2323-2358 (2018).
13. Rocketboards: Golden System Reference Design, https://rocketboards.org/foswiki/view/Documentation/DeviceTreeGenerator, last accessed 2019/08/19.
14. Neuendorffer, S.: FPGA Platforms for Embedded Systems. 351-379. Model-Based Design for Embedded Systems. Nicolescu, G., Mosterman P. J. (Eds.), CRC Press (2018).
15. Jassi, M., Hu, Y., Mueller-Gritschneder, D., Schlichtmann, U.: Graph-Grammar-Based IP Integration (GRIP)—An EDA Tool for Software-Defined SoCs. ACM Transactions on Design Automation of Electronic Systems, 23(3), Article 40: 1-26 (2018).