Avrupa Bilim ve Teknoloji Dergisi
Sayı 25, S. 6-11, Ağustos 2021

Derleme Makalesi

www.ejosat.com ISSN:2148-2683

European Journal of Science and Technology
No. 25, pp. 6-11, August 2021

Review Article

Gerçek Zamanlı Gömülü Sistemlerin Durum Diyagramları ile Modellenmesi

Sonay Duman[1*], Abdullah Elewi[2], Fırat Duman[3]

[1*] Mersin Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Mersin, Türkiye, (ORCID: 0000-0001-9182-8177), sonayduman13@gmail.com
[2] Mersin Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Mersin, Türkiye (ORCID: 0000-0001-9774-5292), elewi@mersin.edu.tr
[3] İskenderun Teknik Üniversitesi, Meslek Yüksekokulu, Mekatronik Bölümü, İskenderun, Türkiye (ORCID: 0000-0002-2412-9867), firat.duman@iste.edu.tr

(İlk Geliş Tarihi 21 Şubat 2021 ve Kabul Tarihi 06 Mayıs 2021)

(**DOI:** 10.31590/ejosat.884420)

ATIF/REFERENCE: Duman, S., Elewi, A. & Duman, F. (2021). Gerçek Zamanlı Gömülü Sistemlerin Durum Diyagramları ile Modellenmesi. *Avrupa Bilim ve Teknoloji Dergisi*, (25), 6-11.

Öz

Gömülü gerçek zamanlı sistemler, Birleşik Modelleme Dili'nin en başarılı uygulama alanlarından biridir. Durum diyagramları davranış modellemesi için kullanılan Birleşik Modelleme Dili tekniklerinden biridir. Durum diyagramları bir sistemdeki kontrolü ve sıralamayı modellemek için kullanılır. Bu özellikle, büyük ölçüde duruma bağlı olan gerçek zamanlı gömülü sistemler için önemlidir. Duruma bağlı olarak çalışan sistemlerin anlık eylemleri yalnızca sistem girdisine değil aynı zamanda sistemde daha önce ne olduğuna da bağlıdır. Bu sebeple, her durumun önem taşıdığı gerçek zamanlı gömülü sistemleri durum diyagramları ile modellemek daha doğru sonuçlara ulaşmak açısından önem taşımaktadır. Bu çalışmada, gerçek zamanlı gömülü sistemlerin modellenmesi için durum diyagramlarının kullanımı ve çeşitleri kod-diyagram dönüşümlü örnekler üzerinde açıklanarak incelenmiştir.

Anahtar Kelimeler: Gerçek zamanlı, Gömülü sistemler, Durum diyagramları, UML, SCXML.

Modeling Real-Time Embedded Systems with State Charts

Abstract

Embedded real-time systems are one of the most successful application areas of Unified Modeling Language. State diagrams are one of the Unified Modeling Language techniques used for behavior modeling. State machines are used to model the controlling and sequencing of a system. This is especially important for real-time embedded systems that are highly state-dependent. The instantaneous actions of state-based systems depend not only on system input but also on what happened before in the system. For this reason, it is important to model real-time embedded systems with state machines in order to achieve more accurate results. In this study, the use and types of state machines for modeling real-time embedded systems are examined by explaining code-diagram alternating examples.

Keywords: Real-time, Embedded systems, State chart, UML, SCXML.

[*] Sorumlu Yazar: sonayduman13@gmail.com

1. Giriş

Yazılım tasarımı, tüm yazılım geliştirme sürecinin önemli bir parçasıdır ve ürün geliştirme süreçlerinin ayrı bir alt süreci olarak adlandırılmaktadır. Elektrik ve Elektronik Mühendisleri Enstitüsü (IEEE) Standart Elektrik ve Elektronik Terimler Sözlüğü (IEEE Std 100 - 2000), "tasarım" terimini, bir sistemin mimarisini, bileşenlerini, ara yüzlerini ve diğer özelliklerini tanımlama süreci şeklinde açıklamaktadır. Bu nedenle, yazılım tasarımı aşamasında, sorumluluk atama ve yerine getirme, sistem mimarisi, görevlerin ayrılması, katmanlama ve modülerleştirme ile ilgili çok sayıda karar alınır (Bernstein ve Yuhas, 2005). Modellerin bu tür karmaşık ilişkileri anlamanın etkili bir yolu olarak kullanılması mühendislik kadar eski bir yöntemdir. Bir model, ilgisiz detayları kaldırdığı veya gizlediği için tasarımcıların temel unsurlara daha kolay odaklanmasına olanak tanımaktadır. İyi modeller, hem sorunların hem de çözümlerin anlaşılmasını kolaylaştırmakla kalmaz, aynı zamanda tasarım amacını etkili bir şekilde iletmeye de hizmet eder. Asıl sistemi inşa etme masrafına ve zahmetine girmeden önce, farklı tasarım alternatiflerinin ilginç özelliklerini tahmin etmek ve böylece riski en aza indirmek için sıklıkla kullanılırlar. Gömülü uygulamalarda, ürün geliştirme süreci, gerçek zamanlı donanım ve alt sistem geliştirme gibi süreçleri de içerebilmektedir. Yazılım tasarımcıları, soruna göre gereksinimleri, doğrudan uygulama veya programlama için yeterli olan çözümün fiziksel modellerine dönüştürür. Gerçek zamanlı yazılım sistemler daha karmaşık hale geldikçe, bu tür sistemleri kesin olarak belirtmek, bu tür sistemlerdeki görevler arasındaki ve çevreleriyle olan etkileşimleri tanımlamak, kodun doğruluğunu tasarlamak ve doğrulamak daha da zorlaşmaktadır (Huang ve Sarjoughian, 2004).

Bu çalışmada, öncelikle yazılım modelleme kavramı açıklanarak, gerçek zamanlı sistemler ve bu tarz sistemlerin modellenmesi, modellemede durum diyagramlarının kullanımı detaylarıyla incelenmiştir. Son bölümde iste durum diyagramı “Kontrol Soyutlaması için Durum Diyagramı Gösterimi” olarak bilinen, “Durum Diyagramı genişletilebilir İşaretleme Dili” kısaltması SCXML(State Chart XML) olan dil kullanılarak durum diyagramı örnekleri verilecektir.

2. Yazılım Modelleme

Modeller, genellikle yazılım geliştirmede benimsenen açıklama biçimleridir. Gereksiz ayrıntılardan yoksun, önemli olanı temsil etmek ve iletmek için, geliştiricilerin araştırılan sorunun karmaşıklığı veya geliştirilmekte olan çözümle başa çıkmalarına yardımcı olmak amacıyla kullanılan soyutlamalardır. Modelleme, diğer tasarım ve mühendislik biçimlerinde kullanılır. İşin temel öğeleri veya süreçlerin nasıl çalıştığı gibi bazı modeller sorun alanının özelliklerini yakalamak için kullanılırken, diğer modeller yazılımın nasıl olduğu gibi farklı yönlerini dikkate almak için kullanılır. Her model, sistemin bazı görüşlerinin soyut bir temsilidir ve bu tür görüşler, geliştirme süreci ilerledikçe değişebilir.

Yazılım geliştirmede farklı bakış açılarında modeller oluşturmak mümkündür. Alan modelleme, belirli bir problem için bağlam bilgisini anlamak ve modellemekle ilgilenir. Bir alan modeli, gerçek dünya problem bağlamındaki ana kavramların bir temsilidir. Spesifikasyon modeli, bir soruna yazılım çözümünde kullanılan yazılım öğelerini temsil eder ve temelde yazılım tarafından sağlanan hizmetlerin yüksek düzeyde soyutlamayla tanımlanmasıyla ilgilidir. Tasarım modellemesi, yazılım sisteminin kendisini, çeşitli bölümlerine sorumlulukların tahsisi, davranışı ve kontrol akışı ile tanımlar. Modeller, genellikle teknikler olarak adlandırılan belirli dil kuralları takip edilerek oluşturulur ve bu kuralları takip ederler; örneğin, modeller anlatıya, diyagramlara ve hatta matematiğe dayalı olabilir (Bernstein ve Yuhas, 2005; Chan, 2003).

2.1. Birleşik Modelleme Dili (UML)

Yazılım geliştirmede iyi bilinen modelleme teknikleri, Birleşik Modelleme Dili (Unified Modeling Language - UML) altında tanımlanmıştır. UML, şu anda yazılım endüstrisi tarafından kullanılan en popüler ve başarılı standartlardan biridir. UML, 1980'lerde ve 1990'ların başında ortaya çıkan, yeni teknikler ve hatta gösterimi daha da genişletmek için bir mekanizma ile güçlendirilen birkaç notasyonun birleştirilmesinin sonucudur. UML'nin bir geliştirme süreci değil, bir modelleme dili olduğuna dikkat etmek önemlidir. UML bir dizi teknik sunar, ancak bu tekniklerin geliştirme sırasında kullanılıp kullanılmayacağını veya nasıl kullanılması gerektiğini belirtmez. Aslında, UML birleştirme alıştırmasının sonucu olduğu için birçok farklı süreç ve uygulamada esnek bir şekilde kullanılabilir ve aynı teknik farklı amaçlara hizmet edebilir. Ayrıca, UML modelleri farklı hassasiyet seviyelerinde oluşturulabilir. UML, hem bir taslak notasyonu olarak, örneğin fikirleri not almak veya paydaşlar arasında iletmek için hem de yazılım sistemlerinin yönlerinin kesin bir açıklaması için, örneğin sistem işlevlerinin yarı-biçimsel belirtimi olarak eşit derecede etkili bir şekilde kullanılabilmektedir.

2.2. Gerçek Zamanlı Sistemleri Modellemede UML Kullanımı

Gerçek zamanlı sistemler son zamanlarda, çoğunlukla kontrol amacıyla kullanılan küçük, gömülü cihazların gittikçe daha güçlü hale gelmesinden dolayı giderek daha fazla ilgi görmeye başlamıştır. Gerçek zamanlı sistemler, gittikçe daha karmaşık görevleri gerçekleştirmeye olanak sağlamaktadır. Bu durum, bu tür sistemlerin karmaşıklığının ve geliştirme maliyetinin artmasıyla sonuçlanabilmektedir. Tasarlanan sistemin kalite beklentilerini karşılayıp karşılamadığını önceden kontrol etmek büyük önem taşımaktadır. Nihai sistemin kalitesini tahmin etmek için mevcut modelleme yöntemleri bulunmaktadır (Tomaszewski, n.d.). UML, nesne yönelimli yazılım modellerinin oluşturulması için standart bir yöntem olarak OMG tarafından benimsenmiştir (OMG, n.d.). 1997'deki tanıtımından bu yana, UML oldukça hızlı bir şekilde yaygınlaşmış ve şu anda hem endüstri hem de akademi tarafından yaygın olarak kullanılmaktadır. Bununla birlikte, birçok yazılım sisteminin niteliksel yönlerini modellemek için uygun olduğu kanıtlanmış olsa da, orijinal tanım bu sistemlerin nicel yönlerini ifade etmek için standart bir araç sağlamadı. Örneğin, gerçek zamanlı sistemlerde, belirli eylemlerin maksimum kabul edilebilir süresi, gerekli ve mevcut iletişim üretim hızları ve zaman aşımı değerleri gibi bir modelin belirli öğeleriyle ilişkili zamansal kısıtlamaların belirlenmesi genellikle gereklidir. Sonuç olarak, bu bilgiyi bir UML modeline dâhil etmek için birçok farklı ve karşılıklı olarak yöntem tanımlanmıştır (Douglass, 1999; Gomaa, 2006; Kabous ve Nebel, 1999; Kähkipuro, 1999; Lanusse, Gérard ve Terrier, 1998; Selic, 1999; Selic, 2000). Bu çeşitlilik, bir standart kullanmanın bazı temel faydalarını açıkça azaltmıştır. Bunu

düzeltmek için OMG, bir UML modelinde geçici özellikleri belirtmek için standartlaştırılmış yolları tanımlayacak bir profil isteyerek orijinal UML spesifikasyonunu tamamlamaya çalışmıştır (OMG, n.d.). Bu profil, "Planlanabilirlik, Performans ve Zaman için UML Profili" olarak adlandırıldı. Profilin, zaman ve zamanlama mekanizmalarının modellemesini standartlaştırmaya ek olarak, programlanabilirlik ve çeşitli performans ölçüleri gibi belirli zamanla ilgili özellikler için UML modellerinin resmi analizini desteklemeyi amaçladığı söylenebilmektedir (OMG, n.d.; Selic ve Motus, 2003).

UML'nin bir diğer uzantısı olan Gerçek Zamanlı UML(UML-RT), Gerçek Zamanlı Nesneye Yönelik Modellemeye dayalı ek modelleme yapıları sunar. Bununla birlikte karmaşık, olay güdümlü ve dağıtılmış gerçek zamanlı sistemleri modellemeyi hedeflemektedir. Gerçek zamanlı sistemlerin hem mantıksal hem de fiziksel yönlerinin belirlenmesini desteklemektedir. UML-RT, sistemlerin bireysel yapısal ve davranışsal yönlerini ve bunların ilişkilerini modellemek için kullanılabilir (Selic, 1999).

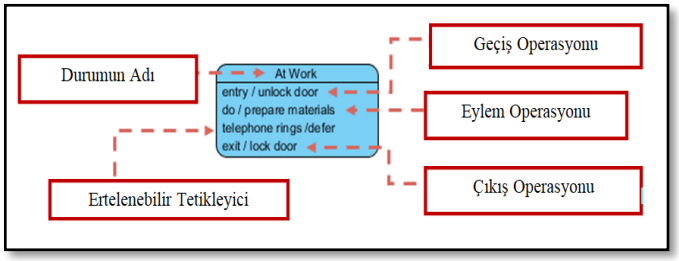
2.1.1. Durum Diyagramları

Gerçek zamanlı sistem geliştirmenin karmaşıklığı, ağırlıklı olarak ana özelliği olan “zamanlama” ile bağlantılıdır. Bir sistemin doğruluğu, yalnızca sistemin ne yaptığını açıklayan işlevsel gereksinimlerle değil, aynı zamanda hizmet sunumunun kalite yönlerine atıfta bulunan işlevsel olmayan gereksinimlerle de değerlendirilir. “Geç cevap yanlış cevaptır” kuralı, hangi kalite unsurlarının en önemli kabul edildiğini açıkça belirtir. Bu durumda, gerçek zamanlı sistemler modellenirken sistemin “zamanlama” konusunda ince detaylarıyla tasarlanması gerekmektedir. UML Durum diyagramı diyagramları gerçek zamanlı sitemlerin tasarlanması için oldukça kullanışlı diyagramlardır. Durum geçişi, bir giriş olayının neden olduğu bir durum değişikliğidir. Bir varlığın davranışı yalnızca girdilerinin doğrudan bir sonucu değildir, aynı zamanda önceki durumuna da bağlıdır. Bir varlığın geçmişi en iyi durum diyagramları ile modellenebilir. UML Durum Diyagramları (veya bazen durum makinesi veya durum şeması olarak da anılır) bir varlığın farklı durumlarını gösterir. Durum diyagramları, bir varlığın bir durumdan diğerine geçerek çeşitli olaylara nasıl tepki verdiğini de gösterebilir (Gomaa, 2011). Durum diyagramı, bir sistemin dinamik doğasını modellemek için kullanılan bir UML diyagramıdır. UML'de David Harel'in durum diyagramları kavramına dayanan sistemlerin davranışını açıklamak için kullanılan bir diyagram türüdür. Durum diyagramları, izin verilen durumları ve geçişleri ve bu geçişleri etkileyen olayları gösterir. Nesnelerin tüm yaşam döngüsünü görselleştirmeye ve böylece duruma dayalı sistemlerin daha iyi anlaşılmasına yardımcı olur (Harel, 1387). Durum diyagramı gösterimleri özellikleri, türlerine bakılmaksızın genel olarak aşağıdaki gibi sıralanabilir.

- Bir durum bir zaman aralığını kaplar.
- Bir durum genellikle bazı koşulu / koşulları karşılayan bir varlığın öznitelik değerlerinin bir soyutlamasıyla ilişkilendirilir.
- Bir varlık(entity), durumunu yalnızca mevcut girdinin doğrudan bir sonucu olarak değiştirmez, aynı zamanda girdilerinin geçmiş geçmişine de bağlıdır.

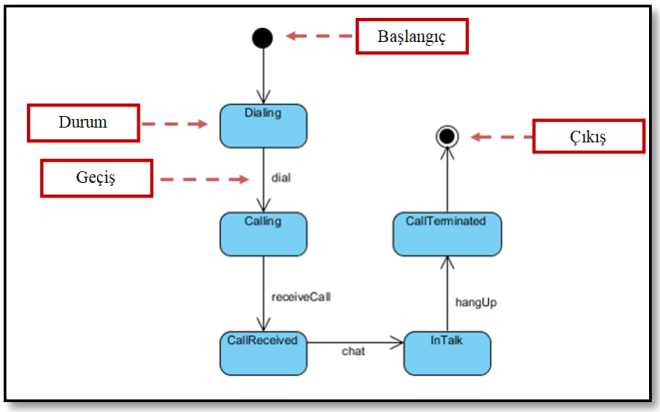
Durum, bir nesnenin yaşam döngüsündeki bir kısıtlama veya bir durumdur, burada bir kısıtın tuttuğu, nesne bir etkinliği yürütür veya bir olayı beklemektedir (Şekil 1). Durum diyagramı, aşağıdakilerden oluşan bir grafiktir:

- Durumlar (basit durumlar veya bileşik durumlar)
- Durumları birbirine bağlayan durum geçişleri



Şekil 1. Durum Gösterimi

Bir durum, bir nesnenin yaşam döngüsündeki bir kısıtlama veya bir durumdur, burada bir kısıtın tuttuğu, nesne bir etkinliği yürütür veya bir olayı beklemektedir(Şekil 1). Durum diyagramı, durumlar (basit durumlar veya bileşik durumlar) ve durumları birbirine bağlayan durum geçişlerinden oluşan bir grafiktir (Şekil 2).



Şekil 2. Durum Diyagramı Örneği

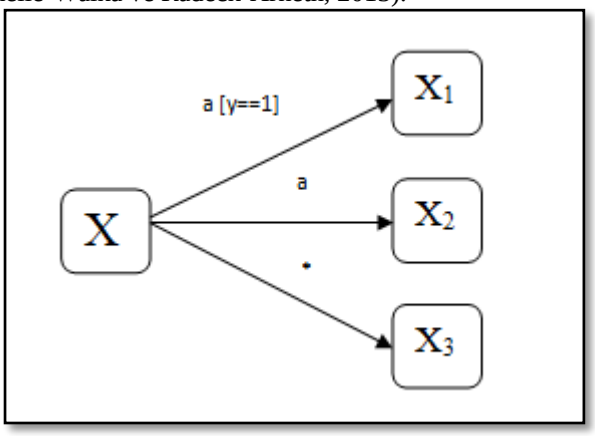
3. Durum Diyagramı XML (SCXML)

SCXML, açılımı Durum Diyagramı XML olan ve “Kontrol Soyutlaması için Durum Diaygramı Gösterimi” anlamına gelen durum diyagramlarının XML kodlaması ve Harel durum diyagramlarına dayalı genel durum diyagramı tabanlı bir yürütme ortamı sağlayan bir biçimlendirme dilidir (W3C, n.d.). SCXML, karmaşık sonlu durum diyagramlarını tanımlayabilir. Örneğin, SCXML'de alt durumlar, paralel durumlar, senkronizasyon veya eşzamanlılık gibi gösterimleri açıklamak mümkündür. Kısaca SCXML, anlambilimin biçimsel bir tanımını da içeren, durum diyagramlarının XML kodlamasıdır. Bu bölümde SCXML dilinin gerçek zamanlı sistemleri modellemek için kullanımı anlatılacaktır.

3.1. SCXML ile Gerçek Zamanlı Sistemlerin Modellenmesi

SCXML semantiği çoğunlukla Harel durum diyagramlarını kullanır. Klasik durum diyagramlarında, durumu tanımlayan her geçerli parametre kombinasyonu için farklı düğümlerin oluşturulmasını gerektirir. Bu, en basit sistemler (durum ve geçiş çarpışması) dışında tümü için çok fazla sayıda düğüme ve düğümler arasında geçişlere yol açabilir. Bu karmaşıklık, durum diyagramının okunabilirliğini azaltır. Harel durum diyagramları ile durum diyagramları içinde çoklu fonksiyonlar arası geçişi

modellemek mümkündür. Bu çapraz işlevli durum diyagramlarının her biri, durum tablosundaki diğer durum diyagramlarını etkilemeden geçiş yapabilir (Harel, 1987). Örneğin, durum diyagramları, durumlar arasında geçişleri gerçekleştirirken ve durumlara girip çıkarken eylemler gerçekleştirebilir. Ayrıca, geçişler koşullar tarafından korunabilir. Bu nedenle, geçişler yalnızca adı "event" özniteliğiyle eşleşen bir olay ortaya çıkarsa ve geçişteki "cond" koşulu yerine getirilirse seçilecektir. Şekil 3-4, örnek SCXML kodunu ve ilgili durum diyagramını göstermektedir (Radomski, Schnelle-Walka ve Radeck-Arneth, 2013).



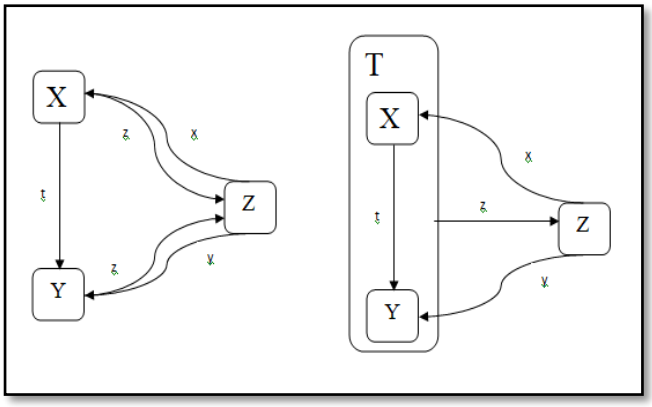
Şekil 3. Durum Diyagramı Örneği

```
<state id="x">
        <transition event="a" cond="y==1" target="x1"/>
        <transition event="a" target="x2"/>
        <transition event="*" target="x3"/>
    </state>
```

Şekil 4. SCXML ile Gösterimi

Burada, hem a olayı meydana geldiğinde hem de y 1'e eşit olduğunda sistem X_1'e geçecektir, ancak e olayı meydana gelirse ve y 1'e eşit değilse, X_2'ye geçecektir. Son olarak, başka bir olay meydana gelirse, X_3'e geçecektir (W3C, n.d.).

Harel durum diyagramlarındaki en güçlü kavram, bileşik durumların iç içe geçmiş durumlar içerebilen hiyerarşi kavramıdır. Bir bileşik durum, paylaşılan davranışı temsil eden, böylece fazlalıktan kaçınan ve geçişlerin geçersiz kılınmasını sağlayan bir atomik veya diğer bileşik durumları grubudur. Aşağıdaki iki eşdeğer durum diyagramını düşünün: İkincisi, X ve Y olmak üzere iki atomik durumdan oluşan bir bileşik T durumuna sahiptir. İlk şekilde z olayı üzerindeki geçişler, ikinci şekilde T bileşik durumundan tek bir geçişe soyutlanmıştır. Yeni bir ana durumda iki veya daha fazla durumun bu şekilde kümelenmesi bu örnekte önemsiz görünse de, büyük ve karmaşık sistemlerde faydalı olmaktadır. Bileşik durumlar, geçişlerin seçilme şeklini etkiler (Harel ve Politi, 1998). Bu bileşik durum kavramını açıklamak için, Şekil 5'te örnek 1 ve 2'de verilen anlamsal olarak eşdeğer iki durum diyagramı verilmiştir. İkinci durum diyagramı, iki atomik durum X ve Y'den oluşan bir bileşik durum T'ye sahiptir.



Şekil 5. Örnek 1 ve Örnek 2 Durum Diyagramları

Örnek 1 ve örnek 2 için SCXML kodu gösterimleri Şekil 6 ve 7'de verilmiştir.

```
<state id="A">
        <transition event="c" target="C"/>
        <transition event="d" target="B"/>
    </state>

    <state id="B">
        <transition event="c" target="C"/>
    </state>

    <state id="C">
        <transition event="a" target="A"/>
        <transition event="b" target="B"/>
    </state>
```

Şekil 6. Örnek 1'in SCXML ile Gösterimi

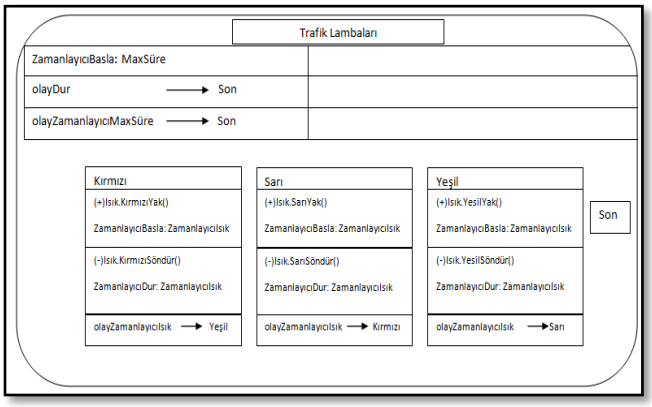
```
<state id="D">
        <state id="A">
            <transition event="d" target="B"/>
        </state>
        <state id="B"></state>
        <transition event="c" target="C"/>
    </state>

    <state id="C">
        <transition event="a" target="A"/>
        <transition event="b" target="B"/>
    </state>
```

Şekil 7. Örnek 2'nin SCXML ile Gösterimi

3.1.1. Trafik Lambası Örneği

Trafik sinyalleri dönüşümlü olarak bir kavşaktaki farklı trafik hareketlerine geçiş hakkı atar ve araç trafiğinin en kontrollü şekilde akmasına izin verilir. Sinyal ekranlarını değiştirmek için bir kontrolör kullanılır. Kavşaklardaki sinyal dizisi kırmızı, sarı ve yeşildir. Sarı bir sinyalin görüntülendiği standart süre üç saniyede sabittir. Yeşil sinyalin süresi, kontrol yöntemine bağlı olacaktır. Sinyal dizisi döngüsünün 120 saniyeden fazla olması tavsiye edilmemektedir. Şekil 8'de trafik lambalarının çalışma yapısı durum diyagramı şeklinde

verilmiştir. Zamanlayıcının başlaması bir eylem operasyonudur ve ardından olayDur, olayZamanlayıcıMaxSüre olarak geçiş operasyonları tanımlanmıştır. Bu operasyonlar işlemlerin sonlanması için "son" basamağına geçiş yapıldığını göstermektedir. Daha sonra Kırmızı, Sarı ve Yeşil durum operasyonları tanımlanmıştır. Durum operasyonları içerisinde, ışıkların yanması ve durması için eylem operasyonları, ışıklar arası geçişin sağlanması için de geçiş operasyonları tanımlanmıştır. Trafik lambası örneğinin durum diyagramı SCXML yapısı ile kodlanmıştır. Öncelikle "event" öznitelikleri tanımlanarak ardından durumlar (state) oluşturulmuştur. Durumlar içerisinde geçiş ve eylem operasyonları yer almaktadır. Son olarak, sistemin durduğu nokta olan "End" durumu tanımlanmıştır.



Şekil 8. Trafik Lambası Durum Diyagramı

Trafik Lambası Durum Diyagramı için SCXML kodu aşağıdaki gibidir.

```
<events>
        <eventSource name="ManagementEvents">
            <event id="evStop"/>
        </eventSource>
        <eventSource name="TimerEvents">
            <timer id="evTimerMaxDuration" name="MaxDuration"/>
            <timer id="evTimerLight" name="TimerLight"/>
        </eventSource>
    </events>
    <!-- States -->
    <state name="TrafficLight">
        <onEntry>
            <timerStart timer="MaxDuration"
duration="light.getMaxDuration()"/>
        </onEntry>
        <transition event="evStop" nextState="End"/>
        <transition event="evTimerMaxDuration" nextState="End"/>
        <state name="Red">
            <onEntry>
                <action>light.turnOnRed()</action>
                <timerStart timer="TimerLight"
duration="light.getRedDuration()"/>
            </onEntry>
            <onExit>
                <action>light.turnOffRed()</action>
                <timerStop timer="TimerLight"/>
            </onExit>
            <transition event="evTimerLight" nextState="Green"/>
        </state>
        <state name="Yellow">
            <onEntry action="light.turnOnYellow()">
                <timerStart timer="TimerLight"
duration="light.getYellowDuration()"/>
            </onEntry>
            <onExit action="light.turnOffYellow()"/>
            <transition event="evTimerLight" nextState="Red"/>
        </state>
        <state name="Green">
            <onEntry action="light.turnOnGreen()">
                <timerStart timer="TimerLight"
duration="light.getGreenDuration()"/>
            </onEntry>
            <onExit action="light.turnOffGreen()"/>
            <transition event="evTimerLight" nextState="Yellow"/>
        </state>
        <state name="End" kind="final"/>
    </state>
```

Şekil 9. Trafik Lambası Örneğinin SCXML ile Gösterimi

4. Sonuç

Bu çalışmada gerçek zamanlı gömülü sistemlerin durum diyagramları ile modellenmesi, SCXML yapısı ve kullanımı örnekler ile incelenmiştir. Özellikle karmaşık yapıda ve çalışırken oluşabilecek hataların büyük sonuçlara sebep olabileceği gerçek zamanlı sistemlerin tasarlanma aşamasında, durum diyagramları ile karmaşıklıkları en aza indirmek mümkün olabilmektedir. Bu bağlamda, yeni tasarlanan sistemleri veya var olan sistemlerde yapılacak değişikleri modellemek için durum diyagramları ile SCXML yapısını kullanmak geliştiricilere büyük kolaylık sağlayacaktır. Bununla birlikte, SCXML, kullanırken bazı zorluklarla birlikte gelen oldukça yeni bir teknolojidir. Örneğin, bu tür modellerin oluşturulmasına ve çalıştırılmasına izin veren yalnızca birkaç profesyonel geliştirme ortamı vardır. SCXML belgelerini düzenlemeyi destekleyen uzantılar ve araçlar arttıkça SCXML modellerinin kullanımını kolaylaştıracaktır.

Kaynakça

Bernstein, L., ve Yuhas, C. M. (2005). *Trustworthy systems through quantitative software engineering* (Vol. 1). John Wiley & Sons.

Chan, C. W. (2003). Knowledge and software modeling using UML. *Software and Systems Modeling, -1*(1), 1-1. doi:10.1007/s10270-004-0057-y

Douglass, B. P. (1999). *Doing hard time: developing real-time systems with UML, objects, frameworks, and patterns* (Vol. 1). Addison-Wesley Professional.

Gomaa, H. (2006, May). Designing concurrent, distributed, and real-time applications with UML. In *Proceedings of the 28th international conference on Software engineering* (pp. 1059-1060).

Gomaa, H. (2011). *Software modeling and design: UML, use cases, patterns, and software architectures*. Cambridge University Press.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of computer programming*, *8*(3), 231-274.

Harel, D., & Politi, M. (1998). *Modeling reactive systems with statecharts: The statemate approach*. New York: McGraw-Hill.

Huang, D., & Sarjoughian, H. (2004). Software and simulation modeling for real-time software-intensive systems. *Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications*. doi:10.1109/ds-rt.2004.37

Kabous, L., & Nebel, W. (1999, October). Modeling hard real time systems with uml the ooharts approach. In *International Conference on the Unified Modeling Language* (pp. 339-355). Springer, Berlin, Heidelberg.

Kähkipuro, P. (1999, October). UML based performance modeling framework for object-oriented distributed systems. In *International Conference on the Unified Modeling Language* (pp. 356-371). Springer, Berlin, Heidelberg.

Lanusse, A., Gérard, S., & Terrier, F. (1998, June). Real-time modeling with UML: The ACCORD approach. In *International Conference on the Unified Modeling Language* (pp. 319-335). Springer, Berlin, Heidelberg.

OMG (n.d.). RFP for scheduling, performance, and time. (n.d.). Retrieved February 15, 2021, from https://www.omg.org/spec/UML/About-UML/

OMG (n.d.). *UML profile for schedulability, performance, and time specification*.Retrieved February 15, 2021, from https://www.omg.org/spec/UML/About-UML/

OMG (n.d.). *Unified modeling language specification (version 2.5.1)*. Retrieved February 15, 2021, from https://www.omg.org/spec/UML/About-UML/

Radomski, S., Schnelle-Walka, D., & Radeck-Arneth, S. (2013, August). A prolog datamodel for state chart XML.

In *Proceedings of the SIGDIAL 2013 Conference* (pp. 127-131).

Selic, B. (1999). Turning clockwise: using UML in the real-time domain. *Communications of the ACM*, *42*(10), 46-54.

Selic, B. (2000). A generic framework for modeling resources with UML. *Computer*, *33*(6), 64-69.

Selic, B., & Motus, L. (2003). Using models in real-time software design. *IEEE Control Systems Magazine*, *23*(3), 31-42.

Tomaszewski, P. (n.d.). Real-time systems modelling using UML. Retrieved February 15, 2021, from https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.109.3394&rep=rep1&type=pdf

W3C (n.d.). *State chart Xml (scxml): State Machine notation for control abstraction*. Retrieved February 15, 2021, from http://www.w3.org/TR/scxml