T.C.
EGE ÜNİVERSİTESİ
Fen Bilimleri Enstitüsü

KESİKLİ GÜÇLE ÇALIŞAN CİHAZLAR İÇİN HIZLI VE HATASIZ PROGRAM GELİŞTİRME

Yüksek Lisans Tezi

Murat MÜLAYİM

Bilgisayar Mühendisliği Anabilim Dalı

İzmir
2020

T.C.
EGE ÜNİVERSİTESİ
Fen Bilimleri Enstitüsü

KESİKLİ GÜÇLE ÇALIŞAN CİHAZLAR İÇİN HIZLI VE HATASIZ PROGRAM GELİŞTİRME

Murat MÜLAYİM

Danışman : Doç. Dr. K. Sinan YILDIRIM

Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Mühendisliği Yüksek Lisans Programı

İzmir
2020

Murat MÜLAYİM tarafından yüksek lisans tezi olarak sunulan “Kesikli güçle çalışan cihazlar için hızlı ve hatasız program geliştirme” başlıklı bu çalışma EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliği ile EÜ Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi’nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 09.06.2020 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:		**İmza**
Jüri Başkanı	**:** Doç. Dr. Kasım Sinan YILDIRIM	**.....**
Raportör Üye	**:** Dr. Ögr. Üyesi Birol ÇİLOĞLUGİL	**.....**
Üye	**:** Doç. Dr. Tolga AYAV	**.....**

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Yüksek Lisans Tezi olarak sunduğum "Kesikli güçle çalışan cihazlar için hızlı ve hatasız program geliştirme" başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

09 / 06 / 2020

Murat MÜLAYİM

ÖZET

KESİKLİ GÜÇLE ÇALIŞAN CİHAZLAR İÇİN HIZLI VE HATASIZ PROGRAM GELİŞTİRME

MÜLAYİM, Murat

Yüksek Lisans Tezi, Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Doç. Dr. K. Sinan YILDIRIM

Haziran 2020, 53 sayfa

Pilsiz cihazlar, çevresel ortamda bulunan enerji kaynaklarından enerji hasadı yaparak nesnelerin interneti (IoT) için bağımsız ve sürdürülebilir uygulamalar sağlamaktadır. Bu tür cihazlar depolanan enerji miktarı yeterli seviyeye ulaştığında hesaplama, algılama ve iletişim yaparken; enerji tamamen bittiğinde aniden ölürler. İşleme ve ölme döngüleri ile devam eden bu işleyiş biçimi kesikli işleyiş olarak adlandırılır. Kesikli işleyiş ile çalışan uygulamaların geliştirilmesi ve gerçekleştirimi için görev tabanlı bir programlama modeli kullanmak gerekmektedir. Fakat var olan görev tabanlı kesikli işleyiş ile çalışan uygulamalar, bu uygulamaları işletecek çalışma zamanı ortamlarına bağımlıdır. Bu bağımlılık nedeniyle uygulamaların hedef platforma yüklenmeden hatalarının ayıklanması ve test edilebilmesi zor olmaktadır.

Tez kapsamında, görev tabanlı ve kesikli çalışan uygulamaların geliştirilmesini sağlayan bir alana özel dil olan TaskDSL tasarlanmıştır. Mühendislerin TaskDSL dilini kullanarak uygulama geliştirmelerine ve bu uygulamaların hatalarını genel amaçlı bir bilgisayar üzerinde ayıklamalarına olanak sağlayan Taskify aracı geliştirilmiştir. Taskify otomatik olarak hedef çalıştırma ortamına bağlanabilecek ve hedef platforma yüklenebilecek C programı da üretmektedir. Taskify bir Eclipse eklentisi olarak gerçeklenmiştir ve kesikli işleyiş ile çalışan üç uygulama ile değerlendirilmiştir.

Anahtar sözcükler: Enerji hasadı, kesikli işleyiş ile çalışan yazılım, pilsiz, araç, hata ayıklayıcı

ABSTRACT

FAST AND BUG-FREE APPLICATION DEVELOPMENT FOR INTERMITTENTLY-POWERED DEVICES

MÜLAYİM, Murat

MSc in Computer Engineering

Supervisor: Assoc. Prof. Dr. K. Sinan YILDIRIM

Jun 2020, 53 pages

Battery-less embedded devices rely only on ambient energy harvesting that enables stand-alone and sustainable applications for the Internet of Things. These devices perform computation, sensing and communication when the harvested ambient energy in their energy reservoir is sufficient; they die abruptly when the energy is completely drained out. This kind of operation, the so-called intermittent execution, dictates a task-based programming model for the development and implementation of intermittent applications. However, today's task-based intermittent programs are tightly-coupled to the underlying run-time environments. This makes their debugging and testing difficult before deploying them into the target platform.

In order to overcome this difficulty, this thesis introduces Taskify, a tool that enables engineers to develop and debug task-based intermittent programs in TaskDSL, i.e., a domain specific language that has been designed for the development of intermittent programs on any general-purpose computer. Taskify automatically transforms these programs into C programs that can be linked to the underlying run-time environment and deployed into the target platform. Taskify is implemented as an Eclipse plugin. It has been evaluated on three intermittent applications.

Keywords: Energy Harvesting, Intermittent Software, Battery-less, Tool, Debugger

ÖNSÖZ

Teknolojinin hızlı bir şekilde geliştiği 21. yüzyılda nesnelerin interneti alanına duyulan ihtiyaç ve internete bağlanan cihazlarda çeşitlilik artmaktadır. Uygulamaların bağımsız ve sürdürülebilir kılınması amacıyla kesikli güçle çalışan pilsiz cihazlar yerini almaktadır.

Kesikli işleyiş ile çalışan uygulamaların geliştirilmesi ve hatalarından arındırılması sürecindeki zorluklar nedeniyle ortak bir arayüz ve beraberinde uygulama geliştirme ortamı ihtiyacı görülmüştür. Yapılan literatür araştırması neticesinde örnek uygulamalar baz alınarak bir dil tanımlanmış, uygulama geliştirme ortamının özellikleri belirlenmiş ve gerçeklenmiştir.

Literatürde çalışma zamanı ortamlarının kıyaslanması amacıyla kullanılan 3 adet uygulamalar ile değerlendirilen sistem temel fonksiyonelliklere sahiptir. Araç, kesikli güçle çalışan cihazlar için farklı çalışma zamanı ortamlarına destek verilmesi amacıyla genişletilebilmektedir.

İZMİR
09/06/2020

Murat MÜLAYİM

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (Devam)

Sayfa

İÇİNDEKİLER (Devam)

Sayfa

İÇİNDEKİLER (Devam)

Sayfa

ŞEKİLLER DİZİNİ

ŞEKİLLER DİZİNİ (Devam)

TABLOLAR DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

Simgeler	Açıklama

Kısaltmalar

ADC	Analog to Digital Convertor
ANTLR	Another Tool for Language Recognition
AST	Abstract Syntax Tree
CEM	Cold-Chain Equipment Monitoring
DSL	Domain Specific Language
EDB	Energy-interference-free Debugger
EBNF	Extended Backus–Naur Form
FRAM	Ferroelectric Random Access Memory
InK	Intermittently-Powered Kernel
IoT	Internet of Things
RF	Radio Frequency
RFID	Radio Frequency Identification
UML	Unified Modeling Language
WISP	Wireless Identification and Sensing Platform

1 GİRİŞ

Literatürde güncel olarak, ihtiyaç duyulan enerjinin ortamda bulunan radyo frekanslarının (Radio Frequency - RF) hasadı ile elde edildiği ve ultra düşük güç gereksinimlerine ihtiyaç duyan donanımlar yer almaktadır (Gollakota et al., 2013; Smith et al., 2006). Bu bilgisayarlar pilsiz olarak çalışmakta ve nesnelerin interneti (Internet of Things - IoT) için bağımsız ve sürdürülebilir uygulamalar sağlamaktadır (Gollakota et al., 2013; Smith et al., 2006; Prasad et al., 2013). Özellikle IoT cihazlarının insan vücudu, gün içerisinde kullanılan kıyafetler ve elverişsiz ortamlarda kullanımı pilsiz çalışma ile mümkün hale gelmektedir (Buettner et al., 2008). Pilsiz çalışma sağlayan bu bilgisayarlar pilsiz cihazlar (Batteryless devices) olarak adlandırılmaktadır.

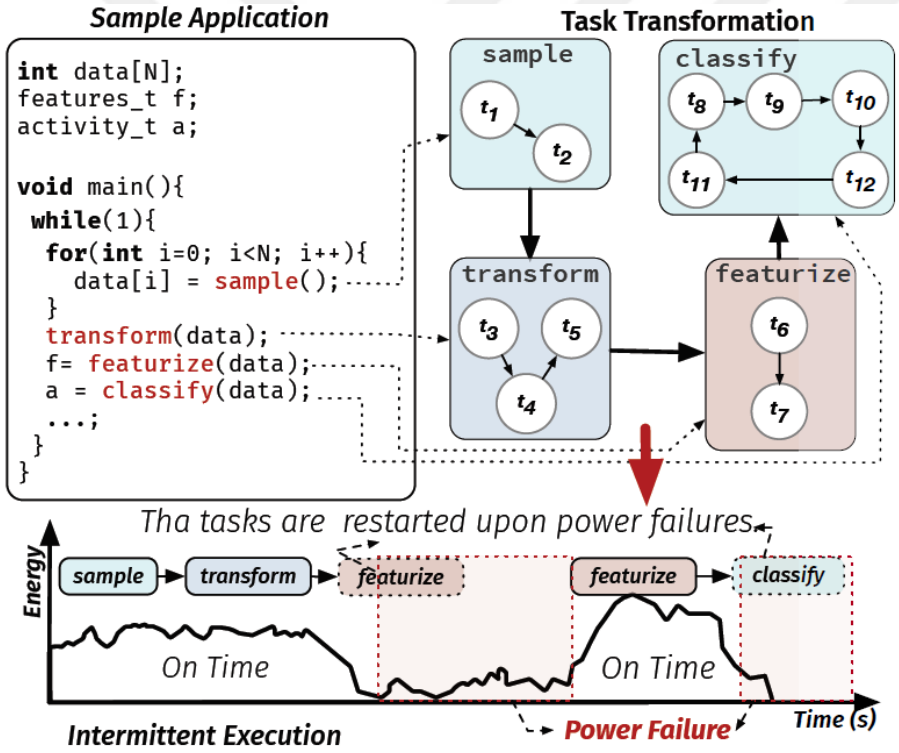
Pilsiz cihazlar tipik olarak ultra düşük güç tüketimine sahip bir mikro-denetleyici, çeşitli sensörler, geçici olmayan ikincil bellek ve enerji depolamak amacıyla da bir enerji depolama birimi ile donatılmıştır (Smith et al., 2006). Pilsiz cihazlar, kapasitörde depolanan enerji miktarı yeterli seviye ulaştığında ortamı dinler, hesaplama yapar ve iletişime geçer. Depolanan enerjinin bitmesi durumda ise cihaz aniden ölür. Ortamdaki enerji kaynağının tahmin edilemez olması nedeniyle, güç kesintileri saniyede 10 kez olabilecek sıklığa ulaşabilmektedir (Pinuela et al., 2013; Ransford et al., 2011; Ma et al., 2015). Bir güç kesintisinde sistemin geçici durumu yani genel ve özel amaçlı saklayıcılar ile birlikte geçici bellekte yer alan bilgiler kaybolmaktadır. Sık sık gerçekleşen güç kesintileri, bu bilgisayarlar üzerinde çalışan yazılımlar için kesikli işleyiş modeli gerektirmektedir. Fakat programın kesikli işleyişi esnasında aşağıdaki sorunlar da ortaya çıkmaktadır:

- Sık gerçekleşen güç kesintileri sonucu sistemin geçici durumunun yok olması nedeniyle hesaplamanın ilerleyişi garanti altına alınamayabilir (Buettner et al., 2011).
- Güç kesintisinden sonra uygulamanın tekrar işlenmesi ile semantik olarak yanlış sonuçlar alınmasına ve/veya sistemin bozulmasına neden olabilir (Ransford and Lucia, 2014; Colin et al., 2016).

Yukarıda listelenen problemleri adresleyen çok sayıda çalışma bulunmaktadır. Bu çalışmalarda iki çalışma modeli benimsenmektedir: Kontrol noktası tabanlı sistemler ve görev tabanlı sistemler.

Kontrol noktası tabanlı sistemlerde genel olarak sistemin geçici durumunun periyodik olarak kalıcı belleğe kaydedilmesi amacıyla kontrol noktası olarak adlandırılan bir mekanizma kullanılmaktadır (Ransford et al., 2011; Balsamo et al., 2016; Jayakumar et al., 2014; Lucia and Ransford, 2015; Van Der Woude and Hicks, 2016). Güç kesintisi durumunda hesaplama, son kontrol noktasından devam etmektedir.

Görev tabanlı sistemler statik olarak tanımlanan görevler modeli (Şekil 1.1) ile ciddi anlamda yükten kurtarmaktadır (Colin et al., 2016b; Hester and Sorber, 2017; Yıldırım et al., 2018; Maeng et al., 2017). Bu sistemler uygulama geliştiricilere, programların bir dizi görevlere bölünmesini gerektirmektedir. Geliştirici ayrıca işletilecek sonraki görevleri belirlemeli ve görevler arasında veri paylaşımı için kullanılacak verileri de belirlemelidir.



Şekil 1.1 Görev tabanlı modellere genel bakış

1.1 Problemin Tanımı

Görev tabanlı uygulamaların fonksiyonel açıdan doğruluğunun sağlanması ve kesikli çalışan bir sistemde hata ayıklanması uygulama geliştiriciler için büyük bir yüktür. Bu yük aşağıdaki nedenlerden kaynaklanmaktadır:

- Hesaplamanın çeşitli görevler şekline bölünmesi, kontrol akışının belirlenmesi ve görevler arasında iletişim ve veri paylaşımı önemli ölçüde geliştirici çabası gerektirmektedir (Kortbeek et al., 2020). Bu işlem ayrıca hata yapma eğilimini de artırmaktadır. Diğer taraftan uygulamanın görevler şekline bölünmesi işleminin doğrulanması ve görev tabanlı uygulamanın fonksiyonellik açısından doğruluğu zaman gerektiren bir işlemdir.
- Görev tabanlı kesikli işleyiş ile çalışan uygulamalar, donanıma özgü çalışma zamanı ortamlarınca sağlanan programlama yapıları kullanılarak yazılmaktadır (Colin et al., 2016b; Hester and Sorber, 2017; Yıldırım et al., 2018; Maeng et al., 2017). Bu nedenle, esas alınan çalışma zamanı ortamı ile donanım arasında sıkı bir bağımlılık bulunmaktadır. Hata ayıklama işlemi uygulamanın yalnızca hedef donanım platformuna yüklenmesi ile yapılabilmektedir.

1.2 Katkılar

Tezde, görev tabanlı kesikli işleyiş ile çalışan uygulamaların geliştirilmesine, genel amaçlı bilgisayar üzerinde hatalarının ayıklanmasına ve test edilebilmesine destek veren Taskify aracı anlatılmaktadır. Taskify aracının sahip olduğu özellikler sayesinde:

- Alana özel bir dil (Domain Specific Language - DSL) kullanılarak hesaplama tabanlı kesikli işleyiş ile çalışan uygulama geliştirilebilmektedir.
- Uygulamaların hedef platforma yüklenmeden hataları ayıklanabilmektedir.
- Alana özel dil ile yazılan uygulamalardan görev tabanlı çalışma zamanı ortamına bağlanmak üzere otomatik olarak C uygulaması üretilmektedir.

Taskify aracının bir parçası olarak, var olan görev tabanlı çalışma zamanı ortamlarının gerektirdiği detaylardan arındırılmış TaskDSL dili bulunmaktadır. TaskDSL diline ilişkin hata ayıklayıcı ile, TaskDSL ile geliştirilmiş uygulamalarda hataların ve yanlış kontrol akışı bildirimlerinin yakalanıp giderilmesi sağlanmaktadır. TaskDSL uygulamalarından InK çalışma zamanı ortamına (Yıldırım et al., 2018) bağlanmak üzere otomatik olarak C uygulaması üretilmektedir. Otomatik olarak üretilen C uygulamaları sık sık gerçekleşen güç kesintilerine rağmen işlenmek üzere hedef platforma yüklenebilmektedir. Taskify aracı bir Eclipse eklentisi olarak gerçeklenmiş ve literatürde değerlendirme amaçlı kullanılan kesikli işleyiş ile çalışan üç adet uygulama ile değerlendirilmiştir.

Bu tez on bölümden oluşmaktadır. İkinci bölümde Taskify aracına olan ihtiyacın altındaki gerekçeler anlatılmaktadır. Sonrasında, üçüncü bölüm içerisinde gerçeklenen araç tanıtılacak olup; alana özel TaskDSL dilinin tanımları ile ilgili detaylara dördüncü bölümde değinilecektir. Beşinci bölümde TaskDSL ile geliştirilen uygulamalarda hata ayıklama özelliği tanıtılarak; TaskDSL uygulamalarından otomatik olarak C kodu üretimi altıncı bölümde anlatılacaktır. Yedinci bölümde Taskify aracının çekirdeğinde yatan teknik bölümleri ile ilgili detaylardan bahsedilecektir. Değerlendirme uygulamaları ile Taskify aracına ait değerlendirme sonuçları sekizinci bölümde paylaşılacaktır. Dokuzuncu bölümde Taskify aracının eksikleri ele alınarak; son bölümde çalışmadan elde edilen sonuçlarla birlikte Taskify aracına eklenmesi planlanan özellikler listelenmektedir.

Bu tez kapsamında aşağıdaki çalıştay bildirisi yayınlanmıştır:

Mulayim, Murat and Goknil, Arda and Yildirim, Kasım Sinan, *"Taskify: An Integrated Development Environment to Develop and Debug Intermittent Software for the Batteryless Internet of Things"*, 2nd International Workshop on Wirelessly Powered Systems and Networks WPSN'20 (co-located with DCOSS). Marina Del Rey, LA, California, 2020.

2 GENEL BİLGİLER VE GEÇMİŞ ÇALIŞMALAR

Yeni ortaya çıkan gömülü sensör sistemleri yalnızca belirli bir ortam üzerinde (Gollakota et al., 2013; Smith, 2013; Soyata et al., 2016) ve/veya yalnızca sisteme atanmış kablosuz enerji ile (Huang and Zhou, 2015; Bi et al., 2015) çalışabilmektedir. Kablosuz enerji ile çalışan bu sensör sistemler pilsiz cihazlar olarak adlandırılmaktadır (Hester and Sorber, 2017).

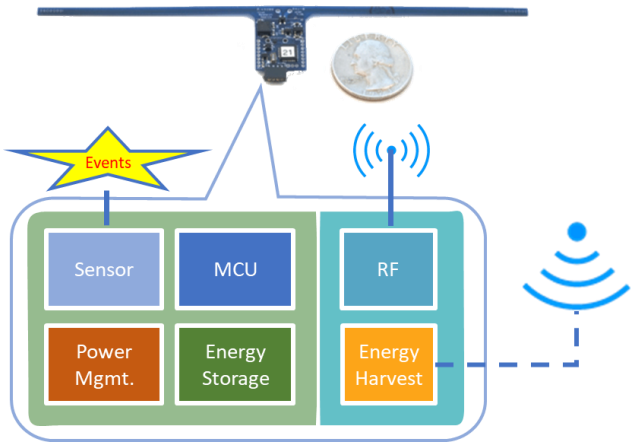
2.1 Pilsiz Cihazlar

Herhangi bir pil, batarya gibi güç kaynağı bulunmayan cihazlar ihtiyaç duyduğu enerjiyi ortamda bulunan güneş, radyo dalgası gibi enerji kaynaklarından kablosuz şekilde karşılamaktadır (Yıldırım et al., 2018). İhtiyaç duyulan enerji, düzenli olmayan enerji kaynağının hasadı ile sağlanmaktadır. Enerji kaynağının düzensiz olması, kaynaktan devamlı sabit miktarda enerji gelmemesi nedeniyle elde edilen enerji kapasitör olarak adlandırılan küçük depolama birimlerinde depolanmaktadır. Pilsiz cihazların temel olarak bileşenleri şu şekildedir:

- Cihazın çalışması için ihtiyaç duyulan enerjinin ortamdaki kaynakların kullanılarak elektrik akımına çevrilmesinden sorumlu enerji hasadı birimi
- Geçici olmayan dahili bir saklama alanı ile birlikte düşük güç tüketimine sahip işlemci birimi
- Düşük güç tüketimine sahip bir iletişim mekanizması sağlayan geri saçılım birimi

Pilsiz cihazlara örnek olarak literatürde Wireless Identification and Sensing Platform (WISP) (Philipose et al., 2005) ve Flicker (Hester and Sorber, 2017) platformları tipik olarak yer almaktadır. Şekil 2.1'de yer alan WISP cihazı ihtiyaç duyduğu enerjiyi kablosuz olarak uzaktaki bir RFID okuyucudan (RFID reader) sağlamaktadır ve radyo dalgalarını hasat etmektedir.

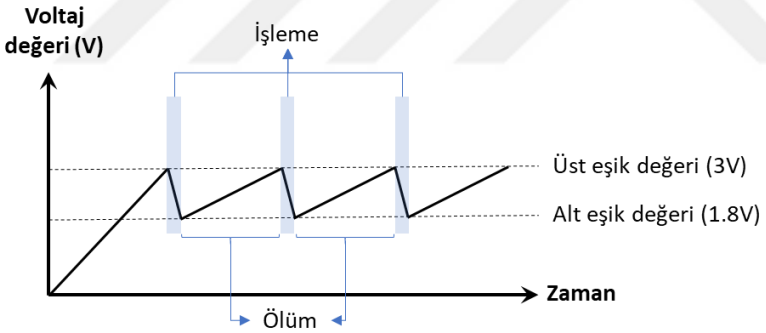
Bu platformlar çok düşük güç tüketimine sahip bir mikro-denetleyici, örnek olarak MSP430FR5969 (Texas Instruments, 2015), içermektedir. Bu mikro-denetleyici geçici (SRAM gibi) ve geçici olmayan (FRAM gibi) bellek kombinasyonları kullanmaktadır. FRAM geçici olmayan bellek (Texas Instruments, 2020) güç kesintilerinde bilgi saklamak için kullanılabilmektedir.



Şekil 2.1 Wireless Identification and Sensing Platform (WISP)

2.2 Kesikli İşleyiş

Pilsiz cihazların düzenli bir enerji kaynağı olmaması nedeniyle elde edilen enerji küçük depolama birimlerinde depolanmaktadır. Depolanan enerji ise hızlıca tükenmekte ve cihaz aniden ölmektedir. Cihazda depolanan enerjinin voltaj değerinin zaman içerisindeki değişim grafiği Şekil 2.2'de verilmiştir.

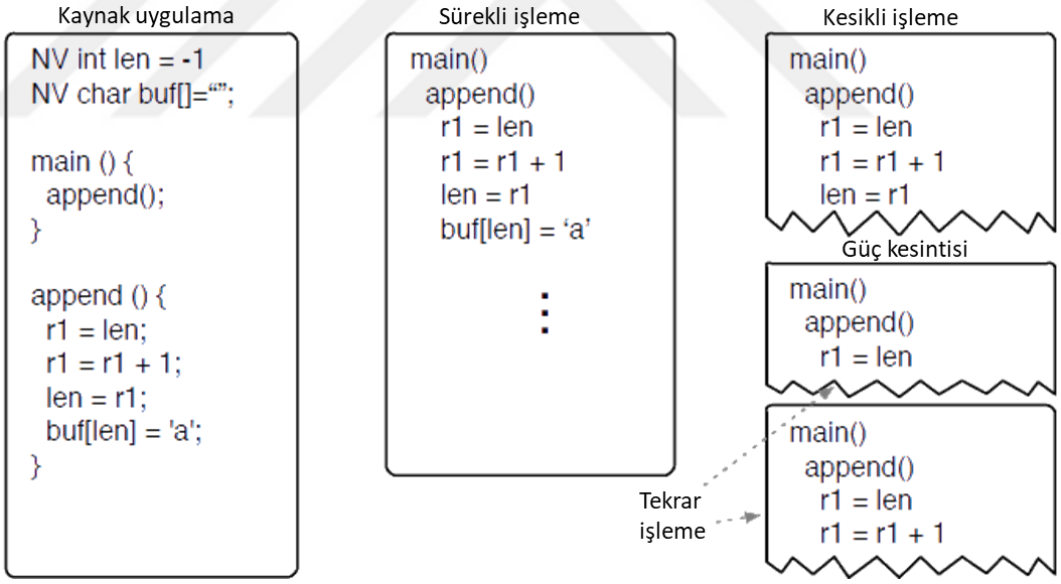


Şekil 2.2 Kesikli işleyişte depolanan enerjinin voltaj değerinin zamana göre değişim grafiği. Voltaj değeri üst eşik değere ulaştığında cihaz çalışır, alt eşik değere ulaştığında cihaz ölür.

Tipik olarak cihaz enerjisi olduğunda çalışırken, bittiğinde ölmektedir. Tekrar işleme esnasında ise işlemenin en baştan başlamamalı, akışın ilerlemesi sağlanmalıdır. Bir enerji kesintisinde sistemin durumunu ifade eden veriler kaybolmaktadır. Bu nedenle bu verilerin geçici olmayan belleğe kaydedilmesi gerekmektedir. Enerji kesintisi durumunda işlemenin kesilmesine karşın verilerin tutarlılığı da sağlanmalıdır.

2.3 Akışın İlerlemesi ve Bellek Tutarlılığı

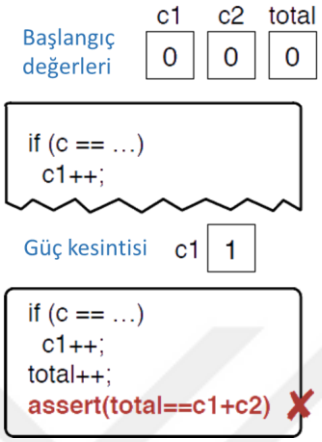
Ortamdaki kaynağın hasadı ile elde edilen enerji çok düşüktür ve enerjiye erişim her zaman mümkün olmayıp tahmini de yapılamamaktadır (Lucia et al., 2017). RFID okuyucu ve WISP platformu kullanılarak (2 metre mesafe ile) elde edilen enerjinin erişilebilirliği Mementos çalışma zamanı ortamı kullanılarak örneklenmiş olup 250 örnekte işleme süresi yaklaşık olarak ortalama 100 ms olarak ölçülmüştür (Ransford et al., 2011). Bu sonuçlar, işlemenin sık sık güç kesintisine uğradığını göstermektedir. WISP de düzenli bir kaynak ile enerji hasadı operasyonu için benzer grafiği çıkarmıştır (Tan et al., 2016). Bu nedenle güç kesintisinden sonra işlemenin ilerlemesini sağlayacak sistem tasarımlarına ihtiyaç duyulmaktadır. Çünkü sürekli güçle çalışan cihazlar için geliştirilmiş uygulamalar kesikli işleyiş ile çalışmamaktadır. Şekil 2.3 sürekli güçle çalışan cihazlar için yazılmış bir uygulamanın kesikli işleyiş ile çalıştırıldığında tekrar işlemede akışın en başa döndüğünü göstermektedir. İşlemenin ilerlemesini sağlayacak tasarımlar ile tekrar işlemenin, işlemenin güç kesintisine uğradığı noktadan devam edilmesi sağlanmalıdır.



Şekil 2.3 İşleyiş modellerinin karşılaştırılması a) Sürekli güçle çalışan cihazlar için geliştirilmiş bir uygulama b) Sürekli işleme uygulama bitinceye kadar çalışmaktadır. c) Kesikli işleyiş ile uygulamanın akışı en baştan başlamaktadır. DINO (Lucia and Ransford, 2015) çalışmasındaki Şekil 1'den alınmıştır.

Bir güç kesintisinden sonra işlemenin tekrar başlatılması yanlış sonuçlar verebilmekle birlikte geçici olmayan bellekte veri tutarsızlığı olarak adlandırılan

yan etkileri de olabilmektedir (Ransford and Lucia, 2014). Şekil 2.4'te veri tutarsızlığı problemi yer almaktadır ve işleme esnasında "c1++" satırında "c1" değeri 1 olarak hesaplanmıştır. Tekrar işleme esnasında ise "c1" değişkeni için 0 değeri ile hesaplama yapılması gerekirken; 1 değeri ile hesaplamada yanlış sonuç alınmıştır.



Şekil 2.4 Veri tutarsızlığı problemini gösteren örnek. Tekrar işleme esnasında "c1" değeri 0 olarak okunmalıdır. DINO (Lucia and Ransford, 2015) çalışmasındaki Şekil 8'den alınmıştır.

2.4 Kesikli İşleyiş ile Hesaplama Yaklaşımları

Kesikli işleyiş ile çalışan programlarda akışın ilerlemesini ve veri tutarlılığını sağlamak amacıyla literatürde 2 ana hesaplama yaklaşımı benimsenmiştir: Kontrol noktası tabanlı, görev tabanlı.

Kontrol noktası tabanlı (Checkpointing-based) yaklaşımlarda uygulama geliştirici veya derleyici tarafından sistemin durumunun kalıcı belleğe kaydedilmesi (ya da kontrol noktası oluşturulması) için uygulamaya ek komutlar eklenir (Ransford et al., 2011; Lucia and Ransford, 2015; Van Der Woude and Hicks, 2016; Kortbeek et al., 2020). Genellikle sistemin durumu içerisinde mikro-denetleyicinin genel ve özel amaçlı saklayıcıları ve geçici bellekteki içerik (program yığın gibi) yer almaktadır. Bu nedenle, kontrol noktası yüksek bir seviyede ek yük getirmekle birlikte ve programın işleme süresini de uzatmaktadır. Çalışma zamanında besleme gerilimi izlenerek ölçülen değerin önceden tanımlı bir eşik değerinin altına düşmesi durumunda sistemin durumu dinamik olarak kaydedilebilir. Fakat voltaj izleme işlemi için ekstra enerji harcanması, sistemin harcadığı enerjiye oranla önemsiz bir miktarda, söz konusudur. Eğer geleneksel ADC (Analog-to-digital converter) ölçüm devreleri dışında başka bir donanım gerekiyorsa, gerekli donanım maliyeti de göz önünde bulundurulmalıdır.

Görev tabanlı programlama modellerde her biri bağımsız işlevselliğe sahip görevler bulunmaktadır. Görev tabanlı modeli uygulayan Colin ve arkadaşları görevler arasında girdi ve çıktı için kullanılan kanal soyutlamalarını geçici olmayan belleğe erişim ile sağlamaktadır (Colin et al., 2016b). Görev tabanlı modeli gerçekleyen Alpaca, Mayfly gibi diğer çalışma zamanı ortamları farklı bellek modelleri uygulayarak geçici olmayan belleğe erişimler için iyileştirmelerde bulunmuşlardır (Hester et al., 2017; Yıldırım et al., 2018; Maeng et al., 2017; Durmaz et al., 2019). Bu çalışmalar enerji açısından görev tabanlı modellerin kontrol noktası tabanlı modellere göre planlı ve daha etkin oldukları görülmüştür. Ortaya atılan çalışmalar arasında en güncel görev tabanlı çalışma zamanı InK'tir (Yıldırım et al., 2018).

2.5 InK: Görev Tabanlı Çalışma Zamanı Ortamı

InK, hesaplamanın görevler şeklinde bölünmesini, görevler arası paylaşımlı değişkenler ve kontrol akışı tanımlamasını gerektirmektedir. Bir InK uygulaması, InK kütüphanesi tarafından tanımlanan C makroları kullanılarak geliştirilmiş bir C dosyasıdır. Şekil 2.5 örnek bir InK uygulamasını göstermektedir. InK'te __shared anahtar kelimesi paylaşımlı geçici olmayan değişkenlerin bildiriminde kullanılmaktadır (2. satır). Hesaplamadaki giriş görevi ENTRY anahtar kelimesi ile tanımlanır (4-11 satırları). Ardışık diğer görevler TASK blokları kullanılarak gerçeklenir (13-16 satırları). Kontrolün bir görevden sonrakine aktarılması NEXT anahtar kelimesi ile sağlanır (10 ve 15 satırları). Görevler arası değişkenlere erişim __GET ve __SET çağrımları ile yapılmalıdır (8. satır). InK uygulaması hedef donanıma yüklenmek üzere statik InK kütüphanesi ile bağlanır ve C derleyicisi ile derlenir.

```
1  // task-shared variables.
2  __shared(int data[10]; int i;);
3  // the entry task
4  ENTRY(Start){
5   // sample sensor
6   int read = sample();
7   // data[i] = read
8   __SET(data[__GET(i)],read);
9   ...
10  NEXT(Last); // next task is Last
11 }
12
13 TASK(Last){
14  ...
15  NEXT(null); //task finishes
16 }
```

Şekil 2.5 Örnek InK uygulaması

InK çift tamponlama (double buffering) ile statik versiyonlama gerçeklemektedir (Yıldırım et al., 2018). Görevler arası paylaşımlı değişkenler geçici olmayan bellekte her değişken için 2 versiyon oluşturularak saklanır. Bir görevin işlenmesi esnasında, InK çalışma zamanı görev tarafından değişikliğe uğrayan her paylaşımlı değişken için ayrıca 3. bir versiyon (scratch copy) oluşturmaktadır. Görevin işlenmesi esnasında yalnızca 3. versiyonlar üzerinde değişiklik yapılır. Bir güç kesintisi durumunda, paylaşımlı değişkenlerin orijinal değerleri üzerinde değişiklik olmadığından görev güvenli bir şekilde yeniden başlatılır.

InK paylaşımlı değişkenler üzerinde çift tamponlama ile versiyonlama yaparken; kopya tamponlardaki değerlerin kalıcı belleğe kopyalanması için çift aşamalı işleme (two phase committing) mekanizmasına sahiptir (Yıldırım et al., 2018). Görevin başarılı bir şekilde tamamlanması durumunda; geçici bellekte yer alan 3. versiyon geçici olmayan bellekte yer alan geçici tampona kopyalanır (1. aşama). Sonrasında geçici tamponda bulunan değerler orijinal tampona kopyalanarak görevler arası paylaşımlı değişkenler güncellenir (2. aşama). Kopyalama işleminde okuma sonrası yazma bağımlılığı olmadığından 2. aşamada iken sistemin kesintiye uğraması sonucu çalışma zamanının yeniden başlaması bellek tutarsızlığa neden olmamaktadır. Kontrol akışında yer alan sonraki görev 2. aşamanın tamamlanması sonrasında işletilir. Bahsi geçen işlemler, görevlerin atomik olarak tamamlanmasını ve görevler arası paylaşımlı değişkenlerin tutarlılığını sağlamaktadır.

2.6 Kesikli İşleyiş Sistemleri İçin Araçlar, Hata Ayıklayıcılar ve Test Platformları

Literatürde pilsiz ve kesikli güçle çalışan sistemler için araçlar, hata ayıklayıcılar ve test ortamları bulunmaktadır. Bu çalışmalar çoğunlukla kesikli işleyişin taklit edilmesi veya pilsiz cihazlarda enerji yönetimi üzerine odaklanmaktadırlar. Çalışmaların sahip oldukları özellikleri, katkıları Tablo 2.1'de yer almaktadır.

Ekho enerji hasadı yapan ortamların enerji desenlerini kaydeden bir donanımdır (Hester et al., 2014). Donanım ayrıca pilsiz uygulamaların hedef donanımdaki davranışlarını gözlemlemek amacıyla, kaydedilen enerji desenini uygulama üzerinde tekrar uygulamaktadır. Hata ayıklama desteği yoktur.

EDB (Energy-interference-free Debugger), uygulamaların hedef platformda kesikli işleyiş ile işlenmesi esnasında enerji seviyelerine müdahalede bulunmaksızın uygulamalarda hata ayıklamaya izin veren yazılım ve donanım sunmaktadır (Colin et al., 2016a).

Tablo 2.1 Kesikli işleyiş sistemleri için araçlar, hata ayıklayıcılar ve test platformları. Sütunlarda yer alan çalışmalar satırlarda bulunan özellikleri desteklemektedir.

	Ekho	**EDB**	**IBIS**	**Shepherd**	**Taskify**
Uygulama geliştirme	×	×	×	×	✓
Detayları gizleme	×	×	×	×	✓
İşlemede duraklama	×	✓	×	×	✓
Hata ayıklama	×	✓	×	×	✓
Test	✓	✓	×	✓	✓
Enerji duyarlılığı	✓	×	✓	✓	×
Donanım bağımsız	×	×	×	×	✓

IBIS derleme ve çalışma zamanı esnasında girdi-çıktı kaynaklı hataları tespit eden program analiz aracıdır (Surbatovich et al., 2019). Hata ayıklama desteği yoktur.

Shepherd ise birkaç cihaz için ortamların enerji karakteristiklerini kaydeden ve tekrar uygulayan test platformu sunmaktadır (Geissdoerfer et al., 2019).

Literatür araştırmalarına göre; çalışma zamanı ortamlarının detaylarından arındırılmış şekilde uygulama geliştirmeyi, uygulamaların donanım üzerine yüklenmeden hatalarının ayıklanmasını ve test edilmesini sağlayan bir çalışma bulunmamaktadır.

3 TASKIFY: ARACA GENEL BAKIŞ

Görev tabanlı modeller kesikli güçle çalışan sistemler için çalışma ortamı sunmaktadırlar. Fakat her örnek model kendine özgü kodlama tekniği sunmaktadır ve kendine özgü anahtar kelimelere sahiptir. Kullanılan dilin dışında pilsiz cihazlar için geliştirilen uygulamalar yalnızca hedef cihaz üstünde çalışabilmektedir. Genel amaçlı bilgisayarlar üzerinde çalışmamaktadır. Bu nedenle uygulama geliştirme süreci uzamakta ve uygulama geliştiricisi için hata ayıklama zor olmaktadır.

Yukarıda belirtilen nedenlerden dolayı;

1 Geliştiricilere üst seviye modern bir dil sunmak,

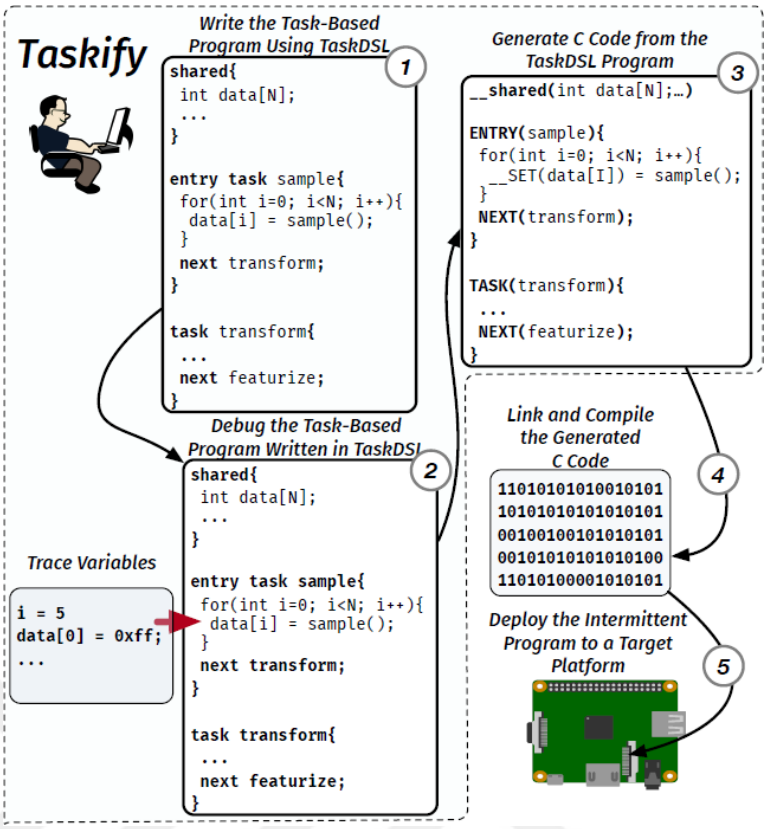
2 Geliştirilmekte olan uygulamanın bilgisayar üzerinde çalışabilir olmasını kılmak,

3 Ve en önemlisi uygulamaların genel amaçlı bilgisayarlar üzerinde çalıştırılarak hata ayıklamayı (debugging) mümkün kılmak

Amaçlarıyla aşağıdakiler sağlanmıştır:

- Daha üst seviye bir dil ile uygulama geliştirmek amacıyla alana özel bir dil tanımlanmıştır.
- Alana özel dil ile uygulama geliştirilmesi ve uygulamanın genel amaçlı bilgisayarlar üzerinde çalıştırılmasına olanak sağlanmaktadır.
- Hata ayıklama desteği ile herhangi bir zamanda değişkenlerin izlenebilmesi ve uygulamanın daha hedef ortamda çalıştırılmadan hatalarından ayıklanması sağlanmıştır.
- Tanımlanan alana özel dil ile yazılan uygulamanın derlenmek üzere hedef kaynak koduna dönüşümü sağlanmıştır.

Şekil 3.1'de yer alan süreç Taskify aracının genel bakışını temsil etmektedir. Taskify düzenlenmiş bir Eclipse uygulama geliştirme ortamıdır. Sürecin 1. aşamasında uygulama geliştirici TaskDSL dili ile görev tabanlı kesikli işleyiş ile çalışan uygulama geliştirmektedir. TaskDSL dili Taskify aracı tarafından sunulmaktadır. TaskDSL dili var olan görev tabanlı çalışma ortamlarının detaylarını gizlemektedir. 1. aşamanın çıktısı TaskDSL dilinin tanımıdır.



Şekil 3.1 Taskify aracına genel bakış

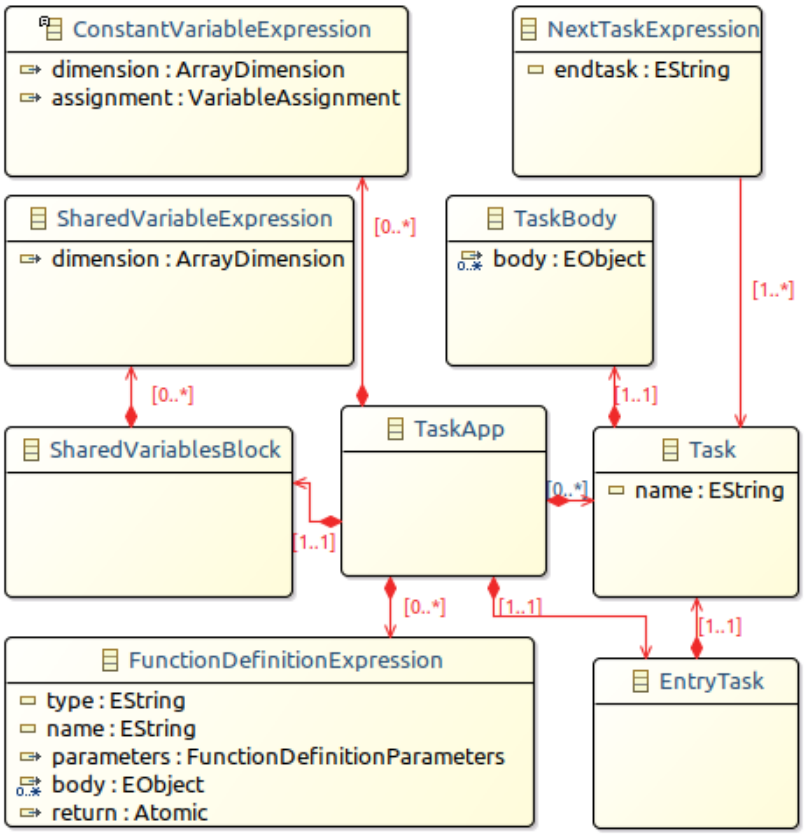
Sürecin 2. aşamasında geliştirilen TaskDSL uygulamasında yer alan yanlış kontrol akışları ve hatalar yakalanmaktadır. Hataların giderilmesi amacıyla Taskify aracı ile hata ayıklama işlemi genel amaçlı bilgisayarlar üzerinde yapılmaktadır. 2. aşamanın çıktısı hataları bulmaya yönelik TaskDSL dilinin tanımıdır.

3. aşamada TaskDSL uygulamasından otomatik olarak C uygulaması üretilmektedir. 1, 2 ve 3. aşamalar direkt olarak Taskify tarafından desteklenmektedir.

4. aşamada üretilen C uygulaması hedef çalışma zamanı ortamı InK ile bağlanmakta (Yıldırım et al., 2018) ve derlenmektedir. 5. ve son aşamada ise derleme sonrası elde edilen program hedef cihaza yüklenmektedir. İlerleyen bölümlerde Şekil 3.1'de yer alan adımlar TaskDSL dili ile yazılmış kesikli çalışan örnek uygulama üzerinden anlatılmaktadır.

4 GÖREV TABANLI KESİKLİ ÇALIŞAN PROGRAMLARIN TANIMLANMASI

Uygulama geliştirme sürecinin ilk aşaması olarak Taskify aracı, özel Eclipse editörü, tarafından sağlanan TaskDSL dili kullanılarak görev tabanlı kesikli işleyiş ile çalışan uygulama yazılmaktadır. InK çalışma ortamı (Yıldırım et al., 2018) için uygulama geliştirilmesine olanak sağlayan TaskDSL dili, görev tabanlı programlama modelini temsil etmek amacıyla temel programlama yapılarına sahiptir. TaskDSL dili kullanılarak program yazılması ile atomik olarak işletilecek kod blokları, atomik kod blokları tarafından paylaşılan değişkenler ve işleyişin ilerleyişini belirleyen kontrol akışlarının bildirilmesine olanak sağlanmıştır.



Şekil 4.1 Temel TaskDSL meta-model gösterimi

Şekil 4.1 TaskDSL dilinin meta-modelinin bir parçasını göstermektedir. Meta-modelde yer alan TaskApp kesikli işleyiş ile çalışan birden fazla görevin işlendiği bir programı, Task ise anlamsal olarak bir işlevi olan veya olmayan, hesaplama içeren, kod bloğunu temsil etmektedir. Bu kod bloğu anlık enerji kesintisine uğrayabilmekte olup enerji kesintisi durumunda; görev tarafından bellekte yapılan bütün değişiklikler geri alınarak görev yeniden başlatılır. Kesikli

işleyiş ile çalışan bir programda ilk olarak işletilen görev EntryTask iken NextTaskExpression, işletilmekte olan görevden sonraki görevi belirtmektedir. Task, bütün hesaplamaları içeren yalnız bir tane TaskBody içermektedir. Görev içerisindeki hesaplamalar SharedVariableExpression ile ifade edilen paylaşımlı değişkenler üzerinde işlemler yapmaktadır. Paylaşımlı değişkenler görevler arasında görevler arasında veri paylaşımı ve iletişim amaçlarıyla kullanılmaktadır. Her paylaşımlı değişken SharedVariablesBlock içerisinde bildirilmekte olup; çalışma ortamında geçici olmayan bellekte (non-volatile memory) saklanmaktadır. Bu paylaşımlı değişkenler çift tamponlama kullanılarak versiyonlanmaktadır (Yıldırım et al., 2018; Colin et al., 2016b). Bu sayede görevler, paylaşımlı değişkenlerin orijinal versiyonu üzerinde işlem yapmamakta olup; kopya versiyon ile işlem yapmaktadır. Görevin başarılı bir şekilde, kesintiye uğramadan, tamamlanması sonucunda değişkenlerin kopya versiyon içerisinde saklanan değerleri orijinal versiyonlarına atomik olarak kaydedilmektedir. Görevin işlenmesi sırasında güç kesintisi nedeniyle kesintiye uğraması durumunda orijinal değişkenler hiç değişmemiş olarak kalmakta olup; görevin tekrar işlenmesi esnasında paylaşımlı değişkenlerin tutarlılığı korunmaktadır. ConstantVariableExpression programdaki sabit, programın işlenmesi süresince değişmeyen, bir değişkeni temsil etmektedir. Görev tanımlamaları dışında kesikli işleyiş ile çalışan program görevler tarafından çağrılan fonksiyonlar (FunctionDefinitionExpression) da içerebilmektedir. Bu fonksiyonlar paylaşımlı değişkenler üzerinde okuma ve yazma işlemi yapamazken; yalnızca çağrım esnasında gönderilen parametreler ve fonksiyonun işlenmesinin sonucunda döndürülecek değerler üzerinde işlem yapabilmektedir.

TaskDSL dili ile yazılmış örnek bir uygulama Şekil 4.2 'de gösterilmektedir. Şekil 4.1 'de yer alan meta-model task, entry, SHARED, CONSTANT, next ve end anahtar kelimeleri ile Şekil 4.2 'de ifade edilmektedir. Görevler task kod blokları (Şekil 4.2 20-28 ve 30-35 satırları) içerisinde gerçeklenmektedir. Görevler arasında iletişim SHARED bloğu (4-8 satırları) içerisinde bildirilen görevler arası paylaşımlı değişkenler ile sağlanmaktadır. TaskDSL string, integer, float ve boolean veri tipleri ve dizi tanımlamalarına destek vermektedir. 1. satırda yer alan CONSTANT anahtar kelimesi bütün program tarafından erişilebilen sabit, sadece okuma işlemi yapılabilen, değişken tanımlaması için kullanılmaktadır. Örnek uygulama 2 adet görev içermektedir (t_init ve use_shareds). İşletilecek ilk görev örnek uygulama satır 20'de yer alan entry anahtar kelimesi ile belirtilir. Bu görev "operand1" ve "factor" görevler arası paylaşımlı değişkenlere matematiksel çarpma işlemi yapmak üzere ilk değerlerini

atamakta ve işlemin sonucunu paylaşımlı olan “result” değişkeninde saklamaktadır (24. satır). İkinci görev ise önceki görev tarafından sağlanan “operand2” ve “result” paylaşımlı değişkenlerini toplamakta olup; sonucu “result” paylaşımlı değişkene yazmaktadır (30-35 satırları). TaskDSL fonksiyon tanımlanmalarına da destek vermektedir. Örnek uygulamada görevler matematiksel çarpma ve toplama işlemlerini add ve mult ile yapmaktadır.

demo.mydsl

```
1  CONSTANT integer factor = 7
2
3  // Define task shared variables
4  SHARED {
5      integer operand1
6      integer operand2
7      integer result
8  }
9
10 integer add (integer val1, integer val2){
11     integer res = val1 + val2
12     return res
13 }
14
15 integer mult (integer val1, integer val2){
16     integer res = val1 * val2
17     return res
18 }
19
20 entry task t_init {
21     // Write directly to shared variables
22     operand1 = 3
23     operand2 = 12
24     result = mult(operand1, factor)
25
26     // Jump to next task
27     next use_shareds
28 }
29
30 task use_shareds {
31     result = add(result, operand2)
32     print("Result is: ", result)
33
34     end
35 }
36
```

Şekil 4.2 Örnek TaskDSL uygulaması

Uygulamanın kontrol akışı next ve end anahtar kelimeri (27 ve 34 satırları) ile belirtilmelidir. Bir görevin sonlandırılması ve sonraki göreve geçişi belirtme, uygulamanın sonlandırılarak kontrolün çalışma ortamına devredilmesi bu anahtar kelimeler ile gerçekleştirilmektedir.

4.1 TaskDSL EBNF Tanımı

Uygulama geliştiricilere görev tabanlı InK modeli (Yıldırım et al., 2018) için üst seviye ve alana özel dil tanımlanmıştır. Dilin tanımında Eclipse tarafından sağlanan Xtext iş çerçevesi kullanılmaktadır (Xtext, 2019). Dilin tanımı Genişletilmiş "Backus-Naur" formu (Extended Backus–Naur Form - EBNF) benzeri ifadeler kullanılarak oluşturulmuş ve temel olarak aşağıdaki kurallara yer verilmiştir:

- (ifade1 | ifade2): ifade1 veya ifade2 arasında bir seçim olacağı
- (ifade)*: ifadenin sıfır veya daha fazla
- (ifade)?: ifadenin ya hiç ya da bir kez
- (ifade)+: ifadenin en az bir kez
- ifade veya (ifade): varsayılan, ifade tam olarak 1 adet

Dilde satır sonu için özel bir karakter tanımı bulunmamaktadır. Sözcüksel çözümleyici (lexer) Xtext tarafından ANTLR (Another Tool for Language Recognition) kullanılarak üretilmektedir (Schill, 2017). Sözcüksel çözümleyici tarafından çözümlenen sonraki jeton (token) baz alınarak yeni bir ifade veya şu anki ifadenin devamı olup olmadığını belirlenmektedir. Yani bütün kodun bir satırda yazılması mümkündür.

4.1.1 Değişken bildirimi ve tanımlanması

Değişkenler bildirim ve paylaşımı, fonksiyonlar gibi birçok yerde kullanılmaktadır. Değişken bildirimi, değişkenin veri tipi ve isminden oluşurken, bildirime ilk değer atanması ile tanımlanması yapılır. Hata ayıklayıcı bu bildirimler ve tanımlamalar ile değişkenlerin kapsamını (scope) ve varsa başlangıç değerlerini sembol tablosuna eklemektedir. Uygulamada yer alan her değişkenin ismi tekil olmalıdır, aksi durumda hata alınmaktadır.

- *DeğişkenBildirimi* ::= <veri tipi> <değişken ismi>
- *DeğişkenTanımı* ::= <DeğişkenBildirimi> '=' <değişken değeri>

4.1.2 Sabit değişken tanımlama

Sabit değişkenler yalnızca bir kez tanımlanabilir ve uygulama kapsamındaki her yerden erişilebilir. Bütün görevler ve fonksiyonlar sabit değişkenlerden okuma yapabilir fakat değişkene yazma izinleri yoktur. Aksi durumda geliştirme ortamı hata vermezken; hata ayıklayıcı hata vermektedir. Tanımlanan sabit değişkenler dizi olabilir. İfade:

“CONSTANT” <değişken bildirimi>

4.1.3 Paylaşımlı değişken bloğu ve bildirimi

Görevler arası paylaşımlı değişkenleri bildirim amacıyla SHARED anahtar kelimesi ile kod bloğu tanımlanmıştır. Eğer en az iki görev içerisinde bir değişken üzerinde işlem yapılacaksa veya bir görevden diğerine veri aktarımı olacak ise; değişken paylaşımlı olarak bildirilmelidir. Paylaşımlı değişken bloğu bir kez açılır ve gerekli değişkenler bildirilir. Değer ataması yapılmamalıdır, aksi durumda geliştirici hata ile karşılaşır. Birden fazla paylaşımlı değişken bildirimi için değişkenler tek bir SHARED kod bloğu içerisinde ardışık olarak yazılmalıdır. Bildirilen paylaşımlı değişkenler dizi olabilir. İfade:

“SHARED” ‘{‘
    (<DeğişkenBildirimi>)*
‘}’

4.1.4 Fonksiyon tanımlama bloğu

Belirli bir amaca hizmet etmek ve bir değer döndürmek amacıyla kullanılan fonksiyonlar en az bir tane parametre almalı ve işlemler sonucunda bir değer döndürmelidir. Paylaşımlı değişkenler üzerinde yazma işlemi yapılamaz; aksi durumda veri tutarlılığının bozulmasına neden olabilmektedir. Bunun için uygulama geliştirme ortamı hata vermemektedir.

<veri tipi> <fonksiyon ismi> ‘(‘ <DeğişkenBildirimi> (‘,’ <DeğişkenBildirimi>)* ‘)’‘{‘
    (<DeğişkenBildirimi> | <DeğişkenTanımı>)*
    <diğer ifadeler>
    “return” <değişken veya değer>

‘}’

Fonksiyon bloğuna örnek olarak Şekil 4.3 parametre olarak “deger1” ve “deger2”yi toplayıp “sonuç” değişkenini döndürmektedir.

```
integer fonksiyon_topla(integer deger1, integer deger2) {
    integer sonuc = deger1 + deger2
    return sonuc
}
```

Şekil 4.3 TaskDSL ile fonksiyon tanımlama örneği

4.1.5 Görev bloğu

InK, hesaplamanın küçük işlere bölündüğü görevler gerektirmektedir (Yıldırım et al., 2018). Görevler parametre almaz, paylaşımlı değişkenler üzerinden veri aktarımı yapılır. Dilde, görevler task anahtar kelimesi ve onu takip eden görev ismi ile tanımlanır. Her görevde öncelikle değişken bildirim ve/veya tanımlamaları yapılmalı ardından işlemler yapılmalıdır. Görev bitiminde ise sonraki görev belirtilmelidir. Sonraki görev bildirimi sonrasında işlem yapılmamalı; aksi halde işlenmeyecek ve geliştirme ortamı hata verecektir.

Görev ::= “task” <görev ismi> ‘{‘
(<DeğişkenBildirimi> | <DeğişkenTanımı>)*
<diğer ifadeler>
(“next” <sonraki görev ismi> | “end”)
‘}’

4.1.6 Giriş görevi bloğu

Görev bloğundan türetilmiş ve her uygulamada yalnızca bir tane bulunması esas kılınan giriş görevi bloğu, uygulamanın giriş yapacağı noktayı belirtmektedir. C, Java gibi genel amaçlı dillerdeki “main” fonksiyonu ile benzer işleve sahiptir. Giriş görevi bloğunda da görev bloğu gibi sıra ile değişken bildirim ve tanımlamaları, operasyonlar ve sonraki görev bildirimi yapılmalıdır. İfade:

“entry” <Görev>

4.1.7 Sonraki görev çağrımı

TaskDSL dili görev tabanlı modeller için yazılmış alana özel dildir ve işlemler görevler içerisinde yapılmaktadır. Bu nedenle uygulamanın akışını sağlamak, işlemenin bir görevden sonraki göreve geçmesi, gerekmektedir. Sonraki göreve geçiş ile bir sonraki görev bloğu işlenmeye başlar. Bir görevden sonra ya bir görev olabilir ya da işlenen görev son görev olabilir. Görev sonunda ne yapılacağını belirtmek amacıyla next anahtar kelimesi ile işlemenin sonraki görev ile devam edeceği, end anahtar kelimesi ile de görevin son görev olduğu, uygulamanın sonlandırılacağı belirtilir.

(“next” <görev ismi> | “end”)

Her görev sonunda veya içerisinde en az bir tane kullanılmalıdır. Aksi durumda, geliştirme ortamı hata vermemekte olup, hata ayıklayıcı için sorun teşkil edecektir. İleriki çalışmalarda hata ayıklayıcı probleme karşın duyarlı olacaktır.

4.1.8 Veri tipleri

Sabit değişken tanımı, paylaşımlı değişken bildirimi, lokal değişken bildirim ve tanımlamalarında belirtilen değişkenin veri tiplerini ifade etmek amacıyla; TaskDSL aşağıdaki 4 adet veri tipi içermektedir:

- Karakter dizisi için “string”
- Tam, ondalık sayılar için “integer”
- Kayan noktalı sayılar için “float”
- Mantıksal değerler için “boolean”

Her değişken bildirim ve tanımlamalarında yalnızca bir adet veri tipi belirtilmesi gerekmektedir. Veri tipinin birden fazla belirtilmesi veya hiç belirtilmemesi durumunda geliştirme ortamı hata vermektedir.

(“string” | “integer” | “boolean” | “float”)

“string” veri tipi için 100 elemanlık karakter dizisi tanımlanmıştır ve deneysel test yapılmıştır. İlerleyen çalışmalarda karakter olarak değiştirilmesi planlanmaktadır. “integer” tipi InK çalışma ortamının anlayacağı “uint32_t” yani

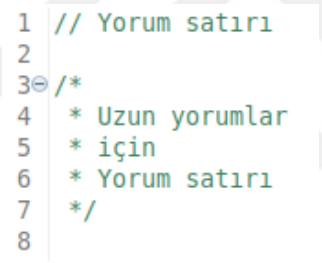
32 bitlik işaretsiz tamsayı tipine eştir. “integer” veri tipinden türeyen “float”, kayan noktalı sayılar kümesini kapsar. Mantıksal veri tipi olan “boolean” ise true ve false, doğru ve yanlış, değerlerinden oluşmaktadır.

Hata ayıklayıcı tutarlı olmayan veri tiplerinde işlemlerde hata vermektedir. Sonraki çalışmalarda geliştirme ortamı için veri tipi kontrolü eklenecektir.

4.1.9 Yorumlar

Uygulamanın kaynak kodu içerisine uygulama geliştiricisi kendisi veya başkası için yorumlar eklemek isteyebilir. Bu yorumlar hata ayıklayıcı tarafından işlenmez. TaskDSL dili iki farklı yorum ekleme özelliğine sahiptir: Yorum satırı ve yorum bloğu.

Yorum satırı, “//” ifadesi ile başlayıp, tüm satır yorum olarak kabul edilir. Diğer ifadelerden sonra da kullanılabilir.



Şekil 4.4 Yorum satırı ve bloğu

Başlangıcı “/*” ve bitişi “*/” olan ifadeler yorum bloğu için kullanılır. Yorum bloğu ile birden fazla satırı içeren yorumlar yazılabilir (Şekil 4.4). Bu sayede birden fazla satır kapsayan yorumlarda ayrı ayrı yorum satırı kullanılmasına ihtiyaç ortadan kalkar.

4.1.10 Diziler

Aynı veri tipinde değerler ile oluşan dizi yapısı TaskDSL tarafından da desteklenmektedir. Dizi bildirimi C programlama diline benzerken; tanımlanmasında farklıdır. Diziler, 4.2.8 alt başlığı altında bahsi geçen 4 veri tipi ile kullanılabilir. TaskDSL dizilerin tek boyutlu olmasına izin vermektedir. Birden fazla boyut için geliştirme ortamı hata vermektedir. Dizi tanımlama ifadesi:

<veri tipi> <dizi ismi> ’[‘ <boyut> ’]’ (‘=’ ‘[‘ <değer> (‘,’ <değer>)* ‘]’)?

Dizinin bir elemanına erişim köşeli parantez([]) ile mümkündür. Yazma işleminde indeksi belirtmek için de köşeli parantez kullanılmaktadır. Dizinin bir elemanına değer atama:

<dizi ismi> ’[‘ <indeks> ’]’ = <değer veya İşlemİfadesi>

Dizi tanımlamalarında, atanan değerler için veri tipi kontrolü yoktur, ilerleyen çalışmalarda genel amaçlı veri tipi kontrolü eklenecektir. Boyutun atanan değerler sayısından küçük olması durumunda hata ayıklayıcı tarafından hata verilmektedir.

4.1.11 Operatörler

Matematiksel, mantıksal, karşılaştırma veya bit düzeyinde işlemler yapmak amacıyla operatörler 3 ayrı grup altına toplanmıştır. Operatörlerin işlem içinde kullanım ifadesi:

- *Operatör* ::= (<AritmetikOperatör> | <MantıksalOperatör> | <BitDüzeyiOperatör>)
- *İşlemİfadesi* ::= <sol değer> <Operatör> <sağ değer>

4.1.11.1 Atama operatörü

Bir değişkene, bir değer veya işlem sonucunu atamak amacıyla atama operatörü kullanılır.

- *AtamaOperatör* ::= ‘=’
- *Atamaİfadesi* ::= <AtamaOperatör> <İşlemİfadesi>

4.1.11.2 Aritmetik operatörler

Aritmetik işlemler için TaskDSL dili 5 operatöre sahiptir. Toplama, çıkarma, çarpma bölme ve mod operatörleri tanımlıdır.

AritmetikOperatör ::= (‘+’ | ‘-’ | ‘*’ | ‘/’ | ‘%’)

4.1.11.3 Mantıksal operatörler

TaskDSL mantıksal olarak “ve” ve “veya” operatörlerini içermektedir. Mantıksal operatörler mantıksal değerler (true ve false) ile kullanılmalıdır. Aksi durumda hata ayıklayıcı hata vermektedir. Değerin tersini alma operatörünü desteklememektedir, sonraki çalışmalarda eklenecektir.

MantıksalOperatör ::= (“AND” | “OR”)

4.1.11.4 Bit düzeyinde operatörler

InK çalışma ortamı (Yıldırım et al., 2018) üzerinde çalışmak üzere yazılmış örnek uygulamalarda (TU Delft Sustainable Systems Laboratory, 2020) sıklıkla bit düzeyinde işlemlere yer verilmektedir. Bu nedenle TaskDSL dili ile uygulama geliştiricilerine bit düzeyinde işlemler yapılmasına olanak sağlamak amacıyla 5 adet operatör tanımlanmıştır.

BitDüzeyiOperatör ::= (“AND_BITWISE” | “OR_BITWISE” | “XOR_BITWISE” | “SHIFT_LEFT” | “SHIFT_RIGHT”)

4.1.11.5 Karşılaştırma operatörleri

TaskDSL karşılaştırma operatörlerinin kullanılmasına yalnızca döngüler ve koşullu ifadelerde izin vermektedir. Karşılaştırma operatörü olarak büyüklük-küçüklük ve eşitlik ifadeleri olmak üzere 6 adet operatör tanımı bulunmaktadır.

- *KarşılaştırmaOperatör* ::= (“>=” | “<=” | “==” | “!=” | ‘<’ | ‘>’)
- *Karşılaştırmaİfadesi* ::= <İşlemİfadesi> <KarşılaştırmaOperatör> <İşlemİfadesi>

4.1.11.6 Öncelik tablosu

Birden fazla operatörün tek bir işlem içerisinde kullanılması durumunda, operatörler arasında öncelik (çarpma işleminin toplama işlemine göre daha öncelikli olması gibi) sırasının olması gerekmektedir. Öncelik tablosunun oluşturulmasında C dili referans alınmış olup operatörler Tablo 4.1’ deki öncelik değerlerine sahiptirler (cppreference.com, 2020).

Tablo 4.1 Operatörler için öncelik tablosu

Öncelik Değeri	Operatör(ler)	Açıklama
3	* /	Aritmetik çarpma Aritmetik bölme
4	+ -	Aritmetik toplama Aritmetik çıkarma
5	SHIFT_LEFT SHIFT_RIGHT	Bit düzeyi sola kaydırma Bit düzeyi saga kaydırma
8	AND_BITWISE	Bit düzeyi and (ve)
9	XOR_BITWISE	Bit düzeyi xor
10	OR_BITWISE	Bit düzeyi or (veya)
11	AND	Mantıksal and (ve)
12	OR	Mantıksal or (veya)
100	(	Parantez açma
200	)	Parantez kapatma

4.1.12 Koşullu ifadeler

Gündelik hayatta herhangi bir işi yapılırken bir koşul doğrultusunda karar verilir. Yemek yemek örneği gibi eğer kişi kendini aç hissediyorsa açlık gereksinimini giderir. Kişi, durduk yere yemek yemez. Programlama dillerinde uygulama yazılırken de bir işin yapılıp yapılmayacağı bir veya birden fazla koşula bağlıdır. Eğer belirtilen koşul doğru ise; iş yapılır, değilse varsa diğer koşullar test edilir. TaskDSL dili de koşul olduğunu belirtmek için eğer yani “if” anahtar kelimesini kullanır. Anahtar kelimenin ardından parantez içerisinde koşul belirtilir. TaskDSL için dikkat edilmesi gereken husus; koşul olarak yalnızca tek bir karşılaştırma işlemine izin verilmesidir. İfade:

“if” ‘(‘ <KarşılaştırmaİfadesiAdı> ‘)’ ‘{‘
    <diğer ifadeler>
‘}’ (“else” “if” ‘(‘ <KarşılaştırmaİfadesiAdı> ‘)’ ‘{‘
    <diğer ifadeler>
‘}’)* (“else” ‘{‘

<diğer ifadeler>
‘}’)?

Koşullu ifade if anahtar kelimesi ile başlar. Ardından var ise ilk koşula uymayan diğer koşul veya koşullar “else if” anahtar kelimesi ifade edilir. Son olarak ise; yine varsa hiçbir koşula uymayan ifadeler else anahtar kelimesi ile belirtilen blokta belirtilir. İfade C ve Java gibi genel amaçlı programlama dillerine benzemekle beraber; blok için küme parantezi kullanımını zorunludur. Aksi durumda geliştirme ortamı hata vermektedir.

4.1.13 Döngüler

Bir veya birden fazla işlemin verilen koşul sağlandığı sürece işlenmesi gerekebilir. Bu durumda döngü kullanılması yazılan kod sayısını azaltmaktadır. TaskDSL döngüler için for ve while döngülerini desteklemektedir. Belirtilen bir başlangıç değeri için her döngüde koşulun test edildiği, koşul doğru ise döngüye devam edildiği for ve belirtilen koşul sağlandığı sürece döngünün devam ettiği while döngülerinde koşul doğru olmadığında döngü bitirilir ve sonraki ifade işlenir. Döngüler için ifadeler:

- *ForDöngüsü* ::= “for” ‘(’ <ilkleme> ‘;’ <koşul> ‘;’ <güncelleme> ‘)’ ‘{‘
  <diğer ifadeler>
  ‘}‘
- *WhileDöngüsü* ::= “while” ‘(’ <koşul> ‘)’ ‘{‘
  <diğer ifadeler>
  ‘}‘

Dizi olarak tanımlı bir değişkenin her elemanına erişerek konsola yazdırma fonksiyonu çağrımı yapan örnek “for” döngüsü Şekil 4.5 döngü örneğinde görülmektedir.

```
for (i = 0; i < BLOCK_SIZE; i = i + 1) {
    index = v_compressed_data_letter[i]
    print("v_compressed_data_letter[", i, "]: ", index)
}

while(j < BLOCK_SIZE) {
    index = v_compressed_data_child[j]
    print("v_compressed_data_child[", j, "]: ", index)
}
```

Şekil 4.5 TaskDSL’de döngüler a) “for” döngüsü b) “while” döngüsü

TaskDSL for döngüsü için ilkleme, koşul ve güncelleme ifadelerini zorunlu kılmakla birlikte; en az birinin boş bırakılması veya yanlış yazılması durumunda geliştirme ortamı hata vermektedir.

4.1.14 Gömülü fonksiyonlar

TaskDSL konsola veri yazma işlemi ve rastgele sayı üretmek amacıyla "print" ve "random" fonksiyonlarına sahiptir.

- *KonsolaYazdırÇağrımı* ::= "print" '(' <yazı veya değer> (',' <yazı veya değer>)* ')'
- *RastgeleSayıÜretÇağrımı* ::= "random" '(' ')'

Gömülü fonksiyon isimleri görev, değişken ve fonksiyon ismi olarak tanımlanmamalıdır. Aksi kullanım durumu Şekil 4.6'da görülmektedir.

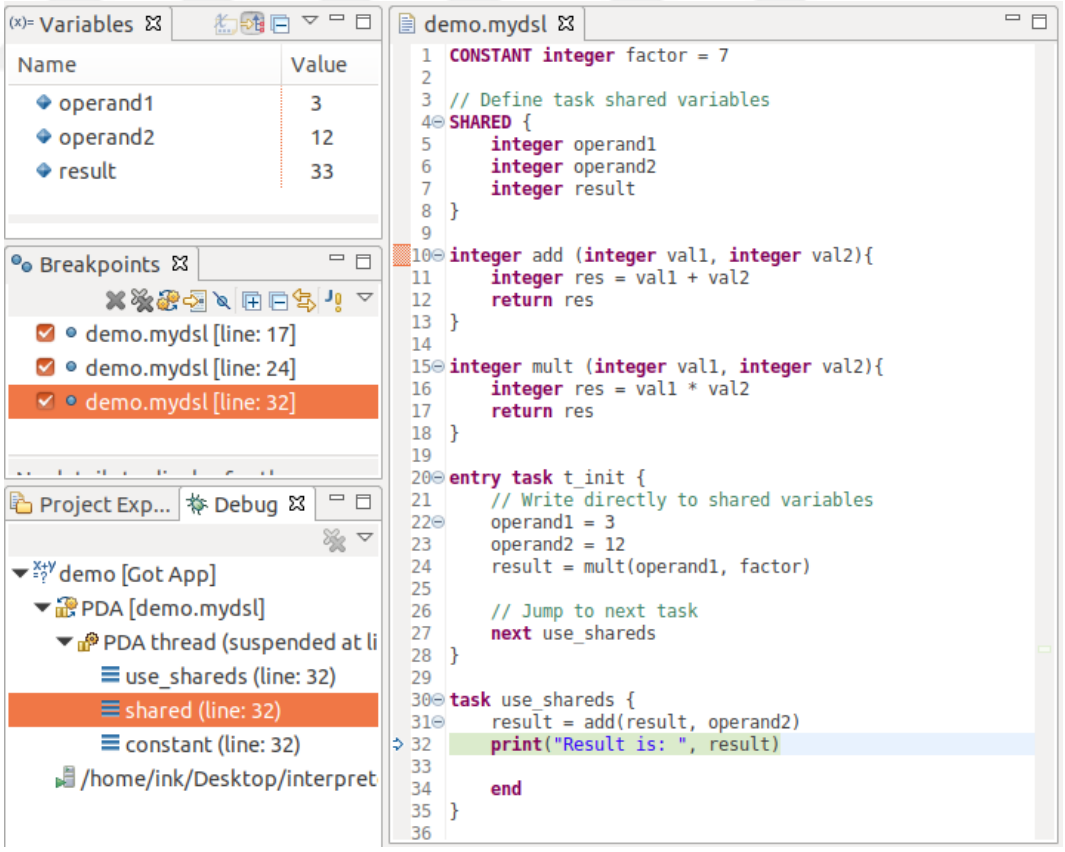
```
integer print(integer deger1, integer deger2) {
    print("sonuc: ", sonuc)
    return sonuc
}

entry task random {
    paylasimli1 = 0

    next task1
}
```

Şekil 4.6 Gömülü fonksiyon isimlerinin aksi durumda kullanımı

5 TaskDSL UYGULAMALARI İÇİN HATA AYIKLAMA

TaskDSL ile yazılmış kesikli işleyiş ile çalışan bir uygulamada bulunan hataları, kontrol akışındaki yanlışlıklar gibi, yakalamak ve çözmek amacıyla 2. aşamada (Bkz. Şekil 3.1) hatalar ayıklanmaktadır. Taskify, uygulamaların hedef donanıma yüklenmeden, genel amaçlı bir bilgisayar üzerinde, TaskDSL uygulamalarının çalıştırılmasına ve değişkenlerin takip edilebilmesine olanak vermektedir. Bu amaçla Taskify, bir kısmı TaskDSL meta-modelde (Bkz. Şekil 4.1) yer alan her bir ifadenin çalışma prosedürünü gerçeklemektedir. Taskify hata ayıklayıcı TaskDSL kullanılarak yazılmış olan bir uygulamanın soyut sözdizim ağacını (Abstract Syntax Tree - AST) oluşturmaktadır. Oluşturulan bu soyut sözdizim ağacı üzerinde dolaşılarak; ağaç içerisinde yer alan her bir düğüme ilişkin çalışma prosedürü çağrılmaktadır.

Şekil 5.1 Taskify aracının hata ayıklama ortamını göstermektedir. Sol üst pencerede TaskDSL uygulamasındaki değişkenler ve bu değişkenlerin hata ayıklama esnasındaki güncel değerleri listelenirken sol pencerede duraklama noktaları vurgulanmaktadır.



Şekil 5.1 Taskify uygulama geliştirme ortamı

6 TaskDSL TANIMLARINDAN C KODU ÜRETİMİ

Her bir TaskDSL tanımı 3. aşamada (Bkz. Şekil 3.1) otomatik olarak C koduna dönüştürülmektedir. TaskDSL meta-modelde (Bkz. Şekil 4.1) yer alan her bir ifade için, InK çalışma zamanı kütüphanesi (Yıldırım et al., 2018) esas alınarak C kodunu üreten gerekli rutinler gerçeklenmiştir. Taskify aracı TaskDSL ile yazılmış kesikli işleyiş ile çalışan örnek uygulamanın (Bkz. Şekil 4.2) soyut sözdizim ağacını -ağaçtaki düğümler TaskDSL meta-modelde (Bkz. Şekil 4.1) yer alan ifadelerdir- dolaşmaktadır. Ağacın dolaşımı esnasında, soyut sözdizim ağacında yer alan her bir eleman için kod üretiminden sorumlu rutinler çağrılmaktadır.

Şekil 6.1 örnek TaskDSL uygulamasından (Bkz. Şekil 5.1) üretilmiş C kodunu göstermektedir.

```c
#include "ink.h"
#define factor 7
// Define task-shared persistent variables.
__shared(
 uint32_t operand1;
 uint32_t operand2;
 uint32_t result;
)
// Declare tasks that will be implemented
ENTRY_TASK(t_init);
TASK(use_shareds);
// Called at the very first boot
void thread1_init(){
 // create a thread with entry task
 __CREATE(15, t_init);
 __SIGNAL(15);
}
// Define helper functions
int add (uint32_t val1, uint32_t val2) {
 uint32_t res = val1 + val2;
 return res;
}

int mult (uint32_t val1, uint32_t val2) {
 uint32_t res = val1 * val2;
 return res;
}
// Implementation of all tasks
ENTRY_TASK(t_init) {
 __SET(operand1, 3);
 __SET(operand2, 12);
 __SET(result, mult(__GET(operand1), factor));
 return use_shareds;
}

TASK(use_shareds) {
__SET(result,add(__GET(result),__GET(operand2)));
 return NULL;
}
```

Şekil 6.1 Örnek TaskDSL uygulamasından (Bkz. Şekil 5.1) üretilmiş C kodu

Üretilmiş C kodu 1. satırda, InK kütüphanesini (Yıldırım et al., 2018) kullanmaktadır. TaskDSL kodundan üretilmiş C kodu meta-modelde (Bkz. Şekil 4.1) yer alan bazı ifadelerle (SharedVariablesBlock ve SharedVariableExpression gibi) oldukça benzerdir. TaskDSL tanımında yer alan her bir görev için InK kütüphanesi kullanılarak görev bildirimleri (10-11 satırları) üretilmektedir (Yıldırım et al., 2018). InK çalışma ortamının uygulamaya ilişkin iş parçacıklarının ve önceliklerinin belirtilmesini gerektirmesi nedeniyle, Taskify otomatik olarak thread1_init (13-17 satırları) ilkleme rutinini oluşturmaktadır (Yıldırım et al., 2018). Görevlerden önce, Taskify TaskDSL dilinde tanımlanan fonksiyonlar için 19-27 satırları arasında görüldüğü üzere C fonksiyonlarını üretmektedir. Son olarak ise giriş görevinden başlanmak üzere geriye kalan görevler için C kodu üretilmektedir (29-34 ve 36-39 satırları). InK'te paylaşımlı değişkenler üzerinde okuma ve yazma işlemleri __GET ve __SET arayüzleri kullanılarak yapılmaktadır (Yıldırım et al., 2018). Bu nedenle Taskify, paylaşımlı değişkenler üzerinde yapılan işlemler için __GET ve __SET çağrımları eklemektedir (30-32 ve 37 satırları).

7 TASKIFY: GERÇEKLEŞTİRİM VE KULLANILABİLİRLİK

Taskify aracı bir Eclipse eklentisi olarak gerçeklenmiştir. Bu eklenti ile Taskify için kullanıcı ara yüzleri etkinleştirilmektedir. Ayrıca TaskDSL dili kullanılarak kesikli işleyiş ile çalışan uygulama yazma, görev tabanlı uygulamaların genel amaçlı bilgisayarlar üzerinde hatalarının ayıklanması ve yazılan uygulamalardan C kodu üretimi özellikleri de sunmaktadır.

Taskify yorum satırları ve üçüncü parti kütüphaneler dışında 4500 satır kod içermektedir. Taskify ile ilgili daha fazla detaylara, çalıştırılabilir dosyalar ve motivasyonları kapsayan ekran görüntüleri, aracın web sitesinden erişebilmektedir: https://github.com/tinysystems/Taskify.

TaskDSL gerçekleştiriminde, programlama dili ve alana özel dil geliştirilmesinde kullanılan, açık kaynak olarak sunulan Xtext iş çerçevesi kullanılmıştır (Eclipse, 2020a). Taskify, TaskDSL dili kullanılarak görev tabanlı kesikli işleyiş ile çalışan programlar yazmak ve hatalarını ayıklamak için özelleştirilmiş bir Eclipse kod editörü ortamıdır.

Eclipse, programlama dilinden bağımsız bir hata ayıklama modeli için arayüzler içeren hata ayıklama çerçevesi sunmaktadır (Eclipse, 2019). Bu hata ayıklama çerçevesi birçok dil için ortak hata ayıklama özelliklerini soyutlamaktadır. Bu sayede Eclipse ortamına yeni bir dil entegre edilebilmektedir.

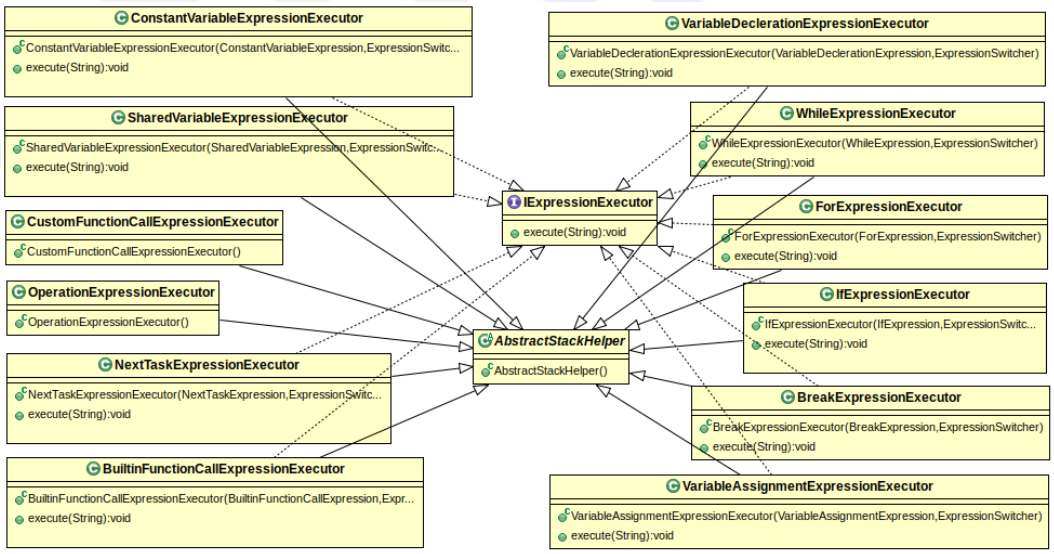
TaskDSL ile yazılmış kesikli işleyiş ile çalışan bir uygulamadan otomatik C kodunun üretilmesi amacıyla Xtend dili (Eclipse, 2020c) kullanılmıştır. Xtend genel amaçlı, Java sanal makinaları için yüksek seviye programlama dilidir (Bettini, 2016).

7.1 Hata Ayıklayıcı

TaskDSL dili ile geliştirilmiş uygulamaların Eclipse ortamında çalıştırılabilmesi amacıyla hata ayıklayıcı (debugger) gerçeklenmiştir. Hata ayıklayıcı Java dili ile gerçeklenmiş olup çalıştırılabilir Java paket dosyası (runnable jar) olarak dışa aktarılmıştır (export). Bu sayede hata ayıklayıcı ile Eclipse ortamı arasında bağımlılık bulunmamaktadır.

Hata ayıklayıcı projesi olarak “org.xtext.example.mydsl.interpreter” oluşturulmuş olup; “smallD” örnek hata ayıklayıcı projesi (Kim, 2019) baz alınmıştır.

TaskDSL diline ait temel bileşenler meta-modelde (Bkz. Şekil 4.1) yer almaktadır. Hata ayıklayıcı, meta-modelde yer alan her temel ifade için ortak bir arayüzü gerçekleyen birer işleyici (executor) sınıf içermektedir. İşeyicilere ilişkin UML (Unified Modeling Language) diyagramı Şekil 7.1’de yer almaktadır. Hatalarından ayıklanmak istenen bir TaskDSL uygulamasının öncelikle hata ayıklayıcı tarafından TaskDSL meta-modeline uygun soyut sözdizimi ağacı oluşturulur. Soyut sözdizimi ağacı elde edilen TaskDSL uygulaması, “ConstantVariableExpression” ifadesinden başlanmak üzere takibinde “SharedVariablesBlock” ifadesi ile istenen “run” veya “debug” modlarında, meta-modeli işleyen işleyiciler metot çağrımları ile uygulama çalıştırılır. Eğer uygulamada hata(lar) var ise hata ayıklayıcı anlaşılabilir şekilde hata vererek; işlemeyi sonlandırmaktadır.



Şekil 7.1 Hata ayıklayıcıda işleyiciler için UML diyagramı

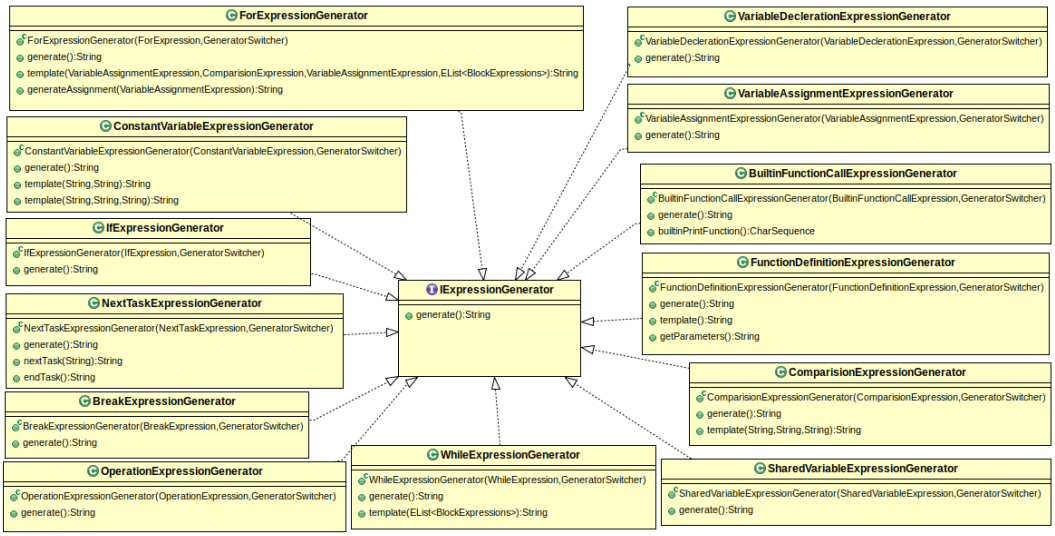
Hata ayıklayıcıda yer alan işleyicilere ait UML diyagramına göre her işleyici ortak bir “IExpressionExecutor” arayüzünü (interface) gerçeklemektedir. Bu ortak arayüz ile “ExpressionSwitcher” sınıfı tarafından, ifadenin tipine göre işleyicinin ilgili fonksiyon çağrımı yapılmaktadır. İşlenen her ifadenin işlenmesi öncesinde de hata ayıklama arayüzü ile senkron olunması amacıyla, kontrol noktaları takip edilerek; işlemenin devam edip etmeyeceği kararı verilmektedir.

Değişkenler hata ayıklama süreci boyunca sembol tablolarında saklanmaktadır. Bu sembol tabloları “shared”, “constant”, fonksiyon ve görevlerin isimlerinden oluşan kapsam alanları içermektedir. Belirlenen kapsam alanları ile, değişkene erişilmeye çalışılan noktaları karşılaştıracak erişilebilirlik karar mekanizması tanımlanmıştır.

7.2 Kod Üretici

TaskDSL uygulama geliştiricilere daha üst seviye ve alana özel bir dil sunmak üzere geliştirilmiştir. TaskDSL dilinin gerçeklenmesinde kullanılan Xtext projesi boş bir kod üretici taslağı içermektedir. Bu boş taslak TaskDSL meta-modelde (Bkz. Şekil 4.1) yer alan her bir ifade için C kodu üretilmesi amacıyla genişletilmiştir. Xtext, bir TaskDSL uygulamasının soyut sözdizim ağacını sunmaktadır. Boş şekilde bulunan kod üretici şablonu InK kütüphanesi (Yıldırım et al., 2018) ile bağlanacak C kodu üretecek şekilde Xtend dili (Eclipse, 2020c) kullanılarak genişletilmiştir.

Kod üretici için, hata ayıklayıcıya benzer bir mimari uygulanmaktadır. Her temel ifade karşılık gelen birer kod üretici tanımlanmış olup; kod üreticiler ortak bir arayüz gerçeklemektedir. Kod üreticilere ait UML diyagramı Şekil 7.2’de yer almaktadır. Kod üreticiler, gerçeklenen “ExpressionSwitcher” sınıfı sayesinde ifadeye ilişkin ilgili üretici fonksiyonların çağrımları ile uygulamanın tamamı için C kodu oluşturmaktadır.



Şekil 7.2 Kod üreticide yer alan üreticiler için UML diyagramı

TaskDSL dilinin tanımında kullanılan Xtext, kod üretimi aşamasında ilgili uygulamanın TaskDSL meta-modele karşılık gelen soyut sözdizim ağacını otomatik olarak sunmaktadır (Eclipse, 2020b). Sağlanan soyut sözdizim ağacı kullanılarak; öncelikle TaskDSL meta-modelde (Bkz. Şekil 4.1) yer alan "ConstantVariableExpression" ve onu takip eden "SharedVariablesBlock" ifadeleri için kod üretimi başlar. Görevler için kod üretimi giriş görevi ile başlar. Sonraki görev çağrımlarında döngü olabilmesi nedeniyle; kod üretimi yapılan her görev için işaret tutularak görevlerin tekrar tekrar üretilmesinden kaçınılmaktadır. Üretilen kodlara ilişkin dosyalara ilgili TaskDSL uygulaması projesi dizini içerisinde "src-gen" klasörü altında erişilebilmektedir.

7.3 Görünüm Bileşenleri

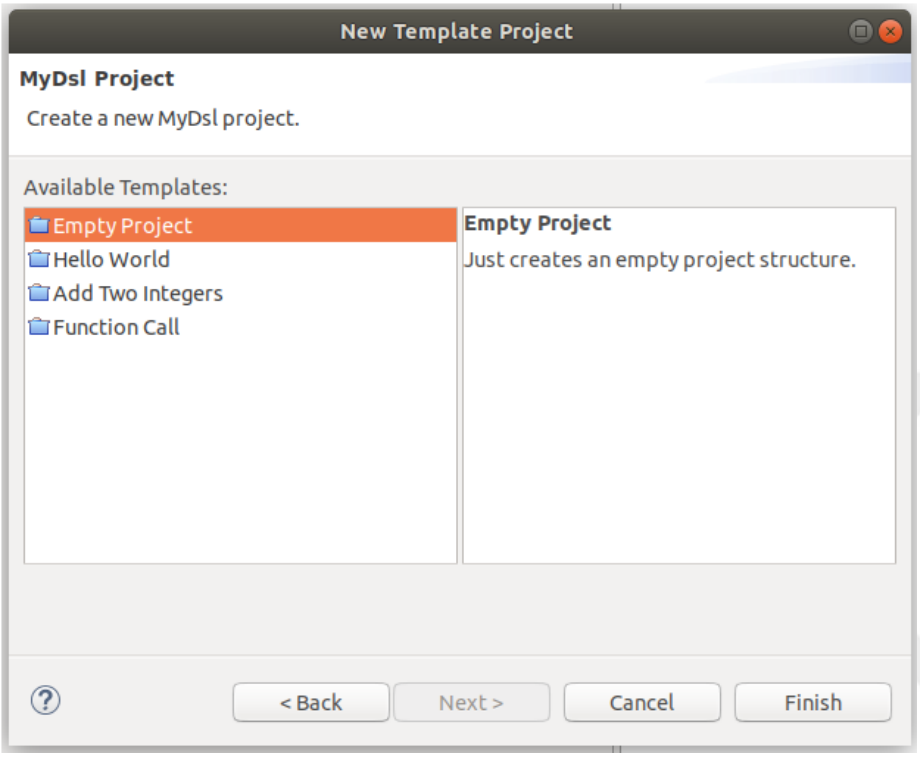
Geliştirme ortamları uygulama geliştiriciler için hataları daha hızlı tespit etmelerini kolaylaştıracak ve erişilecek bilgilere daha hızlı erişmeyi sağlayacak olanaklar sunmaktadırlar. Taskify aracı ile uygulama geliştirme sürecini kısaltmak ve hataların daha kolay bulunmasını sağlamak amacıyla Eclipse görünüm bileşenleri ile ilgili çalışmalar yapılmıştır.

Eclipse görünüm bileşenleri ile etkileşime geçmek, sunulan arayüzleri gerçeklemek amacıyla "org.xtext.example.mydsl.debug.ui" projesi oluşturulmuştur. Projenin gerçeklenmesinde Eclipse tarafından paylaşılan "Platform Debug" projesi (Eclipse Foundation, 2019) baz alınmıştır.

TaskDSL uygulamasının hata ayıklama modunda işlenmesi esnasında değişkenlerin değişiminin takibi amacıyla "Variables View", sol üst pencere, ve kullanıcı tarafından duraklama noktaları eklenebilmesi amacıyla "Breakpoints View" (sol orta pencere) bileşenlerine katkılar sağlanmıştır. Örnek uygulama için hata ayıklama esnasında işleme 32. satırda (Bkz. Şekil 5.1) duraklama durumundadır. "Variables View" içerisinde görüntülenen değişkenler paylaşımlı değişkenlerdir. "Debug View" içerisinde seçim yapılarak yerel ve sabit değişkenler de görüntülenebilmektedir. "Breakpoints View" içerisinde ise duraklama noktası olarak seçilen 17, 24 ve 32 satırları listelenmektedir.

TaskDSL dili ile yeni bir proje oluşturmak veya hazır şablonları kullanmak için "New Project Wizard" ve hata ayıklama amacıyla konfigürasyonlar eklenmesi amacıyla da "Run/debug configuration" pencereleri üzerinde geliştirmelerde bulunulmuştur. "New Project Wizard" penceresine Şekil 7.3'te görüldüğü üzere "Empty Project", "Hello World", "Add Two Integers" ve "Function Call"

projeleri eklenmiştir. Kullanıcı yeni bir TaskDSL projesi oluştururken, hazır durumda bulunan 4 örnek projeden birini seçebilir veya örnekleri göz ardı ederek boş bir proje oluşturabilmektedir. “Run/debug configuration” penceresinde ise işletilmek istenen projenin seçimi sağlanmıştır.



Şekil 7.3 Yeni bir TaskDSL projesi için şablon projeler

Taskify aracının Eclipse pencerelerine erişebilmesi amacıyla Eclipse tarafından sunulan eklenti noktaları gerçeklenmiştir. Görünüm bileşenlerine ilişkin proje içerisinde yer alan “plugin.xml” dosyasında gerçeklenen eklenti noktaları temel olarak aşağıdaki gibidir:

- *org.eclipse.debug.ui.launchConfigurationTabGroups:* “Run/debug configuration” penceresine alt menüler ekleyecek sınıf gerçeklenmiştir.
- *org.eclipse.debug.ui.launchConfigurationTypeImages:* Eklenen run/debug konfigürasyona ilişkin ikon resmi belirtilir.
- *org.eclipse.debug.ui.debugModelPresentations:* “Debug View” içerisinde iş parçacıkları ve elemanlar için veri sağlayacak sınıf gerçeklenmiştir.
- *org.eclipse.ui.editorActions:* Kod editörü üzerinde farenin sol butonuna çift tıklanması ile alınacak aksiyonu belirleyen sınıf belirtilir.

- *org.eclipse.ui.popupMenus:* Farenin sağ butona tıklanması ile açılacak menüye eklenecek seçenek ve ilgili sınıf belirlenir.

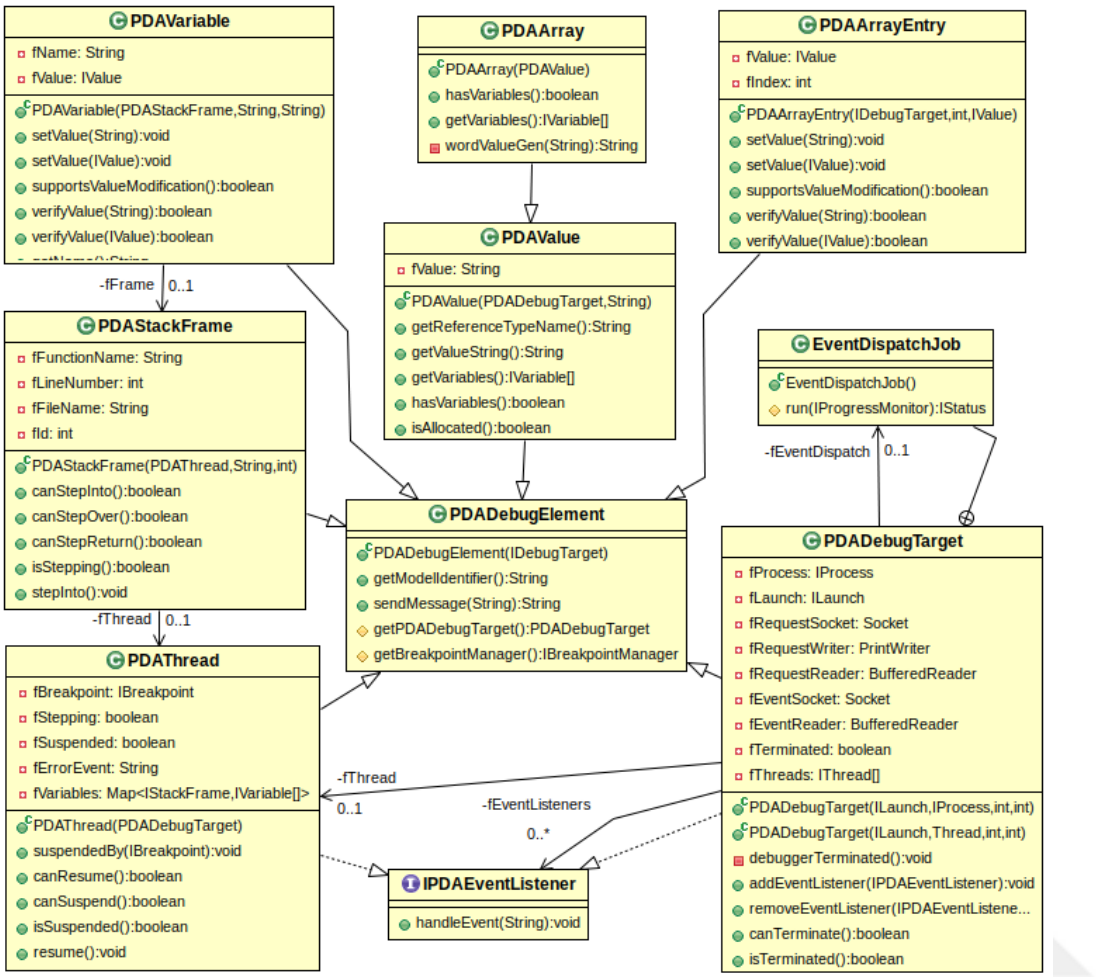
7.4 Hata Ayıklama Arayüzü

Hata ayıklayıcı, Eclipse ortamından bağımsız Java arşiv dosyası olarak gerçeklenmiştir. Hata ayıklayıcı ve görünüm bileşenlerini bütünleştirmek amacıyla hata ayıklama arayüzüne ihtiyaç duyulmuştur. Hata ayıklama arayüzü temel olarak 2 görevi yerine getirmektedir. Bunlar:

1. Hata ayıklayıcı ile iletişime geçmek,
2. Hata ayıklayıcıdan alınan mesajların ayrıştırılarak ve görünüm bileşenlerine anlamlandırılmış veriler sağlamak.

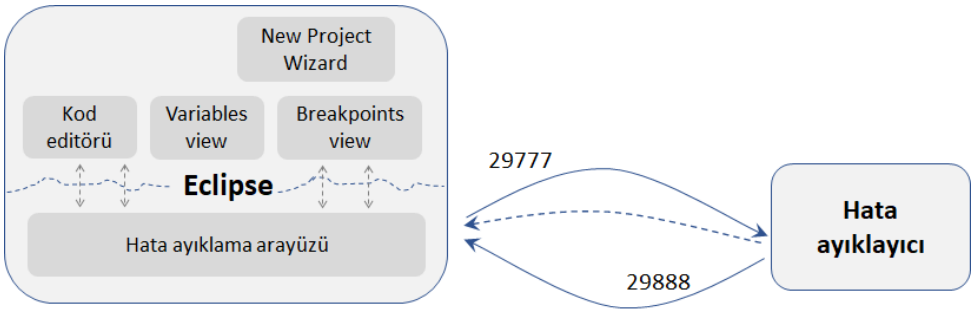
Yukarıdaki görevleri dolayısıyla hata ayıklama arayüzü Eclipse hata ayıklama modelinin çekirdeği konumundadır.

Eclipse hata ayıklama arayüzünün gerçekleştirimi için "org.xtext.example.mydsl.debug.core" projesi oluşturulmuş ve Eclipse tarafından paylaşılan "Platform Debug" projesi (Eclipse Foundation, 2019) esas alınmıştır. Hata ayıklama arayüzünün gerçekleştirimi için tanımlanan Java sınıflarına ilişkin UML diyagramı Şekil 7.4'te yer almaktadır (Wright and Freeman-Benson, 2004).



Şekil 7.4 Hata ayıklama arayüzü için sınıf diyagramı

Hata ayıklama arayüzü ile hata ayıklayıcı arasındaki iletişim soket (socket) haberleşme ile sağlanmaktadır. Hata ayıklama arayüzünden hata ayıklayıcıya mesaj gönderilmesi ve gönderilen mesaja karşılık hata ayıklayıcıdan gelen cevapların okunması amacıyla soket üzerinde 29777 portu kullanılmaktadır. Şekil 7.5 Eclipse ve hata ayıklayıcı arasındaki haberleşmeyi görsel olarak temsil etmektedir. Hata ayıklayıcıda meydana gelen olayların dinlenmesi ise 29888 portu üzerinden sağlanmaktadır. İki farklı port kullanılması ile iletişimin kesintiye uğramaması garanti altına alınmaktadır.



Şekil 7.5 Eclipse ve hata ayıklayıcı

Hata ayıklama arayüzünden hata ayıklayıcıya istek portu üzerinden mesaj gönderilmesi işlemi "PDADebugTarget" Java sınıfı tarafından üstlenilmektedir (Bkz. Şekil 7.4). Gönderilen her istek mesajının alındığına dair bir onay mesajı beklenmektedir. Gönderilen istek mesajları ve mesajlara ilişkin açıklamalar Tablo 7.1'de listelenmektedir.

Tablo 7.1 Hata ayıklama arayüzünden yapılan istek mesajları

İstek mesajı	Açıklaması
brk#<sayı>	Sayı ile belirtilen kod satırında duraklama noktası eklenmesi istenmektedir
resume	İşlemenin devam etmesi istenmektedir
stack	Değişkenlerin güncel durumları istenmektedir
unbrk#<sayı>	Sayı ile belirtilen kod satırında duraklama noktasının kaldırılması istenmektedir
step	İşlemenin duraklatıldığı noktadan sonraki satırın işlenmesi istenmektedir

Hata ayıklayıcıdan Eclipse ortamına mesaj gelmesi bir olayın gerçekleştiği anlamına gelmektedir. Çünkü, Eclipse ortamı tarafından bir istek olmayıp, hata ayıklayıcı bir durumu bildirmek istemektedir. Olay mesajlarının dinlenmesinden sorumlu Java sınıfı "PDADebugTarget" içerisinde tanımlı "EventDispatchJob" sınıfıdır (Bkz. Şekil 7.4). Olaylara ilişkin mesajların diğer ilgili sınıflara aktarılması "IPDAEventListener" arayüzü ile sağlanmaktadır. Olay mesajlarının diğer sınıflarca alınması için "IPDAEventListener" arayüzü gerçeklenmeli ve olaylara kayıtlanmalıdır. Hata ayıklayıcı tarafından gönderilen olay mesajları Tablo 7.2'de listelenmektedir.

Tablo 7.2 Hata ayıklayıcıdan gönderilen olay mesajları

İstek mesajı	Açıklaması
started	İşleme başlamıştır
set brk <sayı>	Sayı ile belirtilen kod satırına duraklama noktası eklenmiştir
resumed client	İşleme devam etmektedir
suspended breakpoint <sayı>	Sayı ile belirtilen kod satırında işleme duraklatılmıştır
resumed step	İşlemenin duraklatıldığı noktada bir satır devam ettirilmiştir
suspended step	İşlemenin duraklatıldığı noktadan bir satır sonrasına geçilmiştir
unset brk <sayı>	Sayı ile belirtilen kod satırında duraklama noktası kaldırılmıştır

Hata ayıklama arayüzüne ilişkin proje içerisinde bulunan "plugin.xml" dosyasında tanımlanan eklenti noktaları temel olarak aşağıdaki gibidir:

- *org.eclipse.debug.core.launchConfigurationTypes:* TaskDSL uygulamaları için yeni bir "run/debug" konfigürasyonu ve konfigürasyonun işletilmesinden sorunlu sınıf belirtilir.
- *org.eclipse.debug.core.sourceLocators:* Run/debug konfigürasyonu için kaynak dosya konumları ile ilgilenecek sınıf belirtilir.
- *org.eclipse.debug.core.breakpoints:* Duraklama noktalarının tipleri ve alınacak aksiyonları üstlenecek sınıf belirtilir.
- *org.eclipse.debug.core.logicalStructureTypes:* "Variables View" içerisinde gösterilecek karmaşık değişkenlerin gösterimini üstlenecek alternatif sınıflar tanımlanır.

8 DEĞERLENDİRME

Tezdeki amaç Taskify aracı kullanılarak kesikli işleyiş ile çalışan uygulama geliştirmenin faydalarını değerlendirmektir. Sistemin değerlendirilmesinde aşağıdaki araştırma sorularının cevaplandırılması amaçlanmıştır:

- *Araştırma sorusu 1:* Taskify aracının kesikli işleyiş ile çalışan uygulama geliştirme sürecine katkısı nedir ?
- *Araştırma sorusu 2:* Taskify aracı tarafından otomatik üretilen C dili uygulamaları ile var olan yapılar ile geliştirilen C dili uygulamaları karşılaştırıldığında; aracı kullanmak kod büyüklüğünü ve bellek gereksinimlerini düşürüyor mu ?
- *Araştırma sorusu 3:* Otomatik üretilen C dili uygulaması hedef platformda beklenen şekilde, doğru çalışıyor mu?

Yukarıdaki araştırma sorularını cevaplandırmak için diğer görev tabanlı modellerin kıyaslanmasında kullanılan uygulamalardan 3 tanesi Taskify aracı kullanılarak geliştirilmiştir (Colin et al., 2016b; Yıldırım et al., 2018). Bu uygulamalar aşağıda listelenmektedir.

- *Bit sayma (Bitcount):* Rastgele bir karakter dizisinde (string) yer alan bitleri sayan uygulamadır. Uygulama aynı işleve sahip 7 farklı yöntem ile bitleri saymaktadır. Her yöntem farklı karakter dizileri üçer deney yapmaktadır.
- *Guguk Filtresi (Cuckoo Filtering):* Rastgele sayıların saklanması amacıyla Cuckoo filtresi çalıştırılır ve aynı filtre kullanılarak dizi kurtarma işlemi gerçekleştirilir. Filtre için 4'te 1 doluluk oranıyla 128 elemanlık bir liste kullanılmaktadır.
- *Soğuk Zincir Ekipman İzleme (Cold-Chain Equipment Monitoring - CEM):* Sıcaklık sensörü verilerinin dinlenerek, verilerin periyodik olarak LZW algoritması ile sıkıştırıldığı uygulamadır (Welch, 1984). Değerlendirmede sıcaklık sensörü verileri bir örnekleme fonksiyonu aracılığı ile sağlanmaktadır. 512 elemanlık liste ve 64 baytlık sıkıştırılmış blok boyutu kullanılmıştır. Bu değerler ile 512 adet sıcaklık sensörü verisini temsil eden değerler LZW sıkıştırma algoritması sonucunda 64 elemanlık bir listeye sıkıştırılmaktadır.

Taskify aracının değerlendirilmesinde kullanılan uygulamalarda yer alan görevlerin, sabit ve paylaşımlı değişkenlerin sayıları Tablo 8.1'de listelenmektedir. Tabloya göre uygulamaların görev sayıları birbirine oldukça yakındır. Fakat paylaşımlı değişkenlere bit sayma uygulaması daha az sayıda erişim yaparken, soğuk zincir ekipman izleme uygulaması yüksek miktarlarda erişmektedir.

Tablo 8.1 Kıyaslama uygulamalarının özellikleri (Okuma, yazma sayısı sütunları paylaşımlı değişkenlere yaklaşık olarak erişim sayılarını belirtmektedir)

	Görev sayısı	**Sabit değişken sayısı**	**Paylaşımlı değişken sayısı**	**Okuma, yazma sayısı**	**Toplam görev sayısı**	**Toplam okuma, yazma sayısı**
Bitcount	10	4	11	23 / 41	32	80 / 90
Cuckoo Filt.	15	7	15	49 / 48	1600	7700 / 5100
Cold-Chain	12	7	18	41 / 43	10000	25000 / 13000

8.1 Uygulama Geliştirme Süreci Senaryosu

TaskDSL dili ile InK kütüphanesinin detayları gizlenmiş olup, uygulama geliştirici yalnızca hesaplamanın görevler biçiminde ayrılmasına odaklanmaktadır. Uygulama geliştirici kontrol akışını tanımlamakta ve InK kütüphanesinin gerektirdiği bildirim ve arayüzler (Yıldırım et al., 2018) ile uğraşmadan paylaşımlı değişkenler ile işlemler yapabilmektedir. Taskify aracı TaskDSL uygulamalarında sözdizimsel hataları belirtmektedir ve kod tamamlama, hata çözme gibi öneriler de sunmaktadır. Bu sayede uygulamaların geliştirilmesinde sözdizimsel hatalardan kaçınılmıştır. Ayrıca Taskify aracı kullanılarak, uygulamaların henüz hedef platforma yüklenmeden fonksiyonel hataları arındırılmıştır. Taskify aracının ve TaskDSL dilinin sağlamış olduğu özelliklerinden dolayı TaskDSL uygulamaları için geliştirme sürecinin kısalacağı düşünülmektedir.

8.2 Üretilen C Kodunun Analizi Senaryosu

Taskify aracı tarafından otomatik üretilen C kodu ve InK kod deposunda (TU Delft Sustainable Systems Laboratory, 2020) yer alan 3 uygulamaya ilişkin analiz sonuçları Tablo 8.2'de yer almaktadır. TaskDSL ile geliştirilen uygulamalar daha az sayıda kod satırı içermektedir (Tablo 8.2).

Tablo 8.2 Uygulamalardaki kod satır sayıları ve boyutları (.text: Program boyutu, .data: Tanımlanan değişkenler, .bss: Bildirilen değişkenler)

			Boyutlar (bayt cinsinden)		
Uygulama	**Tip**	**Satır Sayısı**	**.text**	**.data**	**.bss**
Bitcount	InK	380	2707	316	4592
	Taskify	326	3616	347	5412
Cuckoo Filt.	InK	500	3106	349	4864
	Taskify	370	3748	347	5424
Cold-Chain	InK	370	2628	316	11276
	Taskify	276	2830	316	18264

TaskDSL ile geliştirilen uygulamalardan üretilmiş C kodları derlendiğinde var olan uygulamalara göre daha yüksek boyutta program oluşturmaktadır (Tablo 8.2). Var olan uygulamalara göre daha yüksek bellek gereksinimlerine ihtiyaç duymaktadır.

TaskDSL dilinin güncel gerçekleştirimi C'de bulunan "char" veri tipini desteklememektedir. Ayrıca var olan uygulamalarda kullanılan “union”, “struct” gibi veri yapıları TaskDSL dilinde karşılığı yoktur ve TaskDSL uygulamaları farklı yöntemlerle geliştirilmiştir. Bu nedenle, TaskDSL dili ile tanımlanan ve bildirilen bütün değişkenler bellekte 4 bayt (byte) alan kaplamaktadır. Bütün değişkenlerin 4 baytlık alan kapsaması ve uygulamaların farklı yöntemler ile geliştirilmeleri nedeniyle kod boyutu ve bellek gereksinimleri artmaktadır. Bu sorunu çözmek için TaskDSL dilinde 1 baytlık veri tipi, sıklıkla kullanılan veri yapılarının desteği planlanmaktadır.

8.3 Hedef Platformda Doğruluğun İspatlanması Senaryosu

Taskify aracı kullanılarak geliştirilen TaskDSL uygulamalarından üretilen C kodunun doğru çalışıp çalışmadığının kontrol edilmesi amacıyla donanım kartına yüklenmek üzere derlenmiştir. Hedef donanım olarak üretilen kod TI MSP-EXPFR5969 kartı kullanılmıştır (Texas Instruments, 2015). Donanım üzerinde elde edilen sonuçlar doğrultusunda; Taskify ile geliştirilen 3 uygulama da istenen sonuçları vermiştir. Bu sayede TaskDSL hata ayıklayıcısının ve kod üreticisinin doğruluğu kanıtlanmıştır.

9 TASKIFY ÜZERİNE DÜŞÜNCELER

Bu bölümde Taskify aracının 3 ana başlık altında eksikleri ve bu eksiklerin giderilmesi için gerekli bilgiler yer almaktadır.

9.1 Girdi-Çıktı Tabanlı (I/O-Based) Uygulamalar

Taskify aracı şu anda yalnızca hesaplama tabanlı -çevresel aygıtlar ile etkileşim yapmayan- uygulamalar için geliştirme ve hata ayıklamaya destek vermektedir. Fakat nesnelerin interneti ortamı dinleyen ve ortamdan elde edilen veriler ile işlemler yapmak üzerine kurulmuş bir teknolojidir. Ortamı dinleyen uygulamalar girdi-çıktı tabanlı uygulamalardır. Taskify aracı girdi-çıktı tabanlı uygulamalar için destek vermemektedir. Girdi-çıktı tabanlı uygulamaların Taskify aracı ile geliştirilebilmesi ve hatalarının ayıklanabilmesi amacıyla aşağıdaki yol haritası izlenmelidir:

- Girdi-çıktı operasyonları için TaskDSL dilinde özel anahtar kelimeler tanımlanmalıdır.
- TaskDSL dilinde tanımlanan anahtar kelimeler için hata ayıklayıcıda ilgili işleyici veya işleyiciler tanımlanmalıdır.
- Donanıma bağımlı operasyonlar için uygulama geliştiricinin donanım kodunu taklit edebileceği kod bloklarının tanımlanması sağlanmalıdır.

9.2 Diğer Çalışma Ortamları

Taskify aracı otomatik kod üretimini yalnızca InK çalışma ortamı için yapmaktadır. Diğer görev tabanlı model Alpaca (Maeng et al., 2017) InK ile benzer programlama soyutlamaları içermektedir. Dolayısıyla Alpaca için de bir kod üretici planlanmaktadır. Chain (Colin et al., 2016b) ve Mayfly (Hester et al., 2017) gibi diğer çalışma ortamlarında kanal olarak adlandırılan iletişim mekanizması kullanılmaktadır. İki görev arasında tanımlanan bir kanal ile geçici olmayan bellekte saklanan değişkenlere erişim sağlanır. Kanal tabanlı bellek modellerinde görevler arasında kanal bağlantıları oluşturulması gerekmektedir. Ayrıca bir görev tarafından kanala yazılan girdilerin sonraki görev tarafından okunması gerekmektedir (Colin et al., 2016b). Bu nedenle Chain ve Mayfly gibi kanal tabanlı bellek modelini uygulayan çalışma zamanı ortamları için kod üretimi InK için kod üretiminden oldukça farklıdır.

9.3 Uygulamalarda Davranış Kontrolü

Taskify aracı güç kesintisi senaryolarının uygulama üzerinde test edilmesine destek vermemektedir. Bu nedenle önceden gözlemlenmiş güç kesintisi verileri ile aracın güç kesintilerini taklit etmesi sağlanmalıdır. Bu özellik ile uygulamaların güç kesintisi durumundaki davranışları hedef donanıma yüklenmeden Taskify aracı ile izlenebilir. Literatürde verilen bir veri kümesini meta-modele karşılık gelen ifadeler üzerinde uygulayan -test eden- çalışmalar bulunmaktadır. Bu işleme model muhakeme (model reasoning) denmektedir (Erata et al., 2018). Bu doğrultuda Taskify aracı, AlloyInEcore (Erata et al., 2018) gibi model muhakeme araçları ile bütünleştirilebilir.

10 SONUÇ VE ÖNERİLER

Bu tezde kesikli işleyiş ile çalışan uygulamalar geliştirilmesine ve geliştirilen uygulamaların hatalarının ayıklanmasını sağlayan Taskify aracı anlatılmıştır. Aracın en önemli özellikleri:

1. *TaskDSL dili ile uygulama geliştirme*. TaskDSL dili var olan çalışma zamanı ortamlarının detaylarını gizlemektedir. Dil ile görev tabanlı kesikli işleyiş ile çalışan uygulamalar geliştirilmektedir.
2. *TaskDSL uygulamalarında hata ayıklama*. TaskDSL uygulamaları genel amaçlı bilgisayarlar üzerinde çalıştırılabilmektedir. Bu sayede uygulamalardaki yanlış kontrol akışları ve hatalar hedef donanıma ihtiyaç duymadan tespit edilmektedir.
3. *TaskDSL uygulamalarından C kodu üretimi.* TaskDSL kodundan hedef çalışma zamanı ortamı için otomatik olarak kod üretimi yapmaktadır.

Taskify aracının değerlendirmesi 8. bölümde yer almaktadır. Değerlendirme sonuçları Taskify aracının, görev tabanlı kesikli işleyiş ile çalışan uygulama geliştirilmesinde ve uygulamaların hedef platforma yüklenmeden hatalarının ayıklanmasında daha pratik ve yararlı olduğunu göstermektedir. Bu doğrultuda ele alınması planlanan konular aşağıda listelenmektedir.

- Daha iyi değerlendirme sonuçları almak amacıyla aracın kullanılabilirliğini ve özelliklerini kapsayacak kapsamlı senaryolar ile deneyler yapılacaktır.
- Alpaca (Maeng et al., 2017) gibi diğer çalışma zamanı ortamları için de kod üretici tanımlanacaktır.
- Önceden elde edilen güç kesintisi verileri kullanılarak kesikli işleyiş ile çalışan uygulamalar için test ortamı oluşturulacaktır.
- TaskDSL dili ile girdi-çıktı tabanlı kesikli işleyiş ile çalışan uygulamalar geliştirilmesi sağlanacaktır.

KAYNAKLAR DİZİNİ

Balsamo, D., Weddell, A. S., Das, A., Arreola, A. R., Brunelli, D., Al-Hashimi, B. M., Merrett, G. V. and Benini, L., 2016, Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(12), 1968-1980pp.

Bettini, L., 2016, Implementing Domain-Specific Languages with Xtext and Xtend, *Packt Publishing Ltd., 426p.*

Bi, S., Ho, C. K. and Zhang, R., 2015, Wireless powered communication: opportunities and challenges, *IEEE Communications Magazine*, 53(4), 117-125pp.

Buettner, M., Greenstein, B., Sample, A., Smith, J. R. and Wetherall, D., 2008, Revisiting smart dust with rfid sensor networks, *in Proceedings of the 7th ACM Workshop on Hot Topics in Networks (HotNets-VII)*.

Buettner, M., Greenstein, B. and Wetherall, D., 2011, Dewdrop: an energy-aware runtime for computational rfid, *in Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation*, 197-210pp.

Colin, A., Harvey, G., Lucia, B. and Sample, A. P., 2016, An energy-interference-free hardware-software debugger for intermittent energy-harvesting systems, *ACM SIGARCH Computer Architecture News*, 44(2), 577-589pp.

Colin, A. and Lucia, B., 2016, Chain: tasks and channels for reliable intermittent programs, *in Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 514–530pp.

cppreference.com, C Operator Precedence: https://en.cppreference.com/w/c/language/operator_precedence (Erişim: 24 Ocak 2020)

Durmaz, C., Yildirim, K. S. and Kardas, G., 2019, Puremem: a structured programming model for transiently powered computers, *in Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 1544-1551pp.

KAYNAKLAR DİZİNİ (devam)

Eclipse, Eclipse debug project, https://wiki.eclipse.org/Debug (Erişim: Kasım 2019)

Eclipse, Eclipse xtext framework, https://www.eclipse.org/Xtext/ (Erişim: Nisan 2020)

Eclipse, Integration with EMF, https://www.eclipse.org/Xtext/documentation/308_emf_integration.html (Erişim: Şubat 2020)

Eclipse, Xtend, https://www.eclipse.org/xtend/ (Erişim: Şubat 2020)

Eclipse Foundation, eclipse.platform.debug, https://github.com/eclipse/eclipse.platform.debug (Erişim: Kasım 2019).

Erata, F., Goknil, A., Kurtev, I. and Tekinerdogan, B., 2018, AlloyInEcore: embedding of first-order relational logic into meta-object facility for automated model reasoning, *in Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 920-923pp.

Geissdoerfer, K., Chwalisz, M. and Zimmerling, M., 2019, Shepherd: a portable testbed for the batteryless IoT, *in Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, 83-95pp.

Gollakota, S., Reynolds, M. S., Smith, J. R. and Wetherall, D. J., 2013,. The emergence of rf-powered computing, *Computer*, 47(1), 32-39pp.

Hester, J., Scott, T. and Sorber, J., 2014, Ekho: realistic and repeatable experimentation for tiny energy-harvesting sensors, *in Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, 330-331pp.

Hester, J. and Sorber, J., 2017, Flicker: rapid prototyping for the batteryless internet-of-things, *in Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, 1-13pp.

Hester, J., Storer, K. and Sorber, J., 2017, Timely execution on intermittently powered batteryless sensors, *in Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, 1-13pp.

KAYNAKLAR DİZİNİ (devam)

Huang, K. and Zhou, X., 2015, Cutting the last wires for mobile communications by microwave power transfer, *IEEE Communications Magazine*, 53(6), 86-93pp.

Jayakumar, H., Raha, A. and Raghunathan, V., 2014, Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computer, *in 2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, 330-335pp.

Kim, SmallD, https://github.com/kimtth/smallD (Erişim: Kasım 2019)

Kortbeek, V., Yildirim, K. S., Bakar, A., Sorber, J., Hester, J. and Pawelczak, P., 2020, Time-sensitive intermittent computing meets legacy software, *in Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 85-99pp.

Lucia, B., Balaji, V., Colin, A., Maeng, K. and Ruppel, E., 2017, Intermittent computing: challenges and opportunities, *in 2nd Summit on Advances in Programming Languages (SNAPL 2017)*, 1-14pp.

Lucia, B. and Ransford, B., 2015, A simpler, safer programming and execution model for intermittent systems, *ACM SIGPLAN Notices*, 50(6), 575-585pp.

Ma, K., Zheng, Y., Li, S., Swaminathan, K., Li, X., Liu, Y., Sampson, J., Xie, Y. and Narayanan, V., 2015, Architecture exploration for ambient energy harvesting nonvolatile processors, *in 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 526-537pp.

Maeng, K., Colin, A. and Lucia, B., 2017, Alpaca: intermittent execution without checkpoints, *in Proceedings of the ACM on Programming Languages, 1(OOPSLA)*, 1-30pp.

Philipose, M., Smith, J. R., Jiang, B., Mamishev, A., Roy, S. and Sundara-Rajan, K., 2005, Battery-free wireless identification and sensing. *IEEE Pervasive computing*, 4(1), 37-45pp.

KAYNAKLAR DİZİNİ (devam)

Pinuela, M., Mitcheson, P. D. and Lucyszyn, S., 2013, Ambient rf energy harvesting in urban and semi-urban environments, *IEEE Transactions on microwave theory and techniques*, 61(7), 2715-2726pp.

Prasad, R. V., Devasenapathy, S., Rao, V. S. and Vazifehdan, J., 2013, Reincarnation in the ambiance: Devices and networks with energy harvesting, *IEEE Communications Surveys & Tutorials*, 16(1), 195-213pp.

Ransford, B. and Lucia, B., 2014, Nonvolatile memory is a broken time machine, *in Proceedings of the workshop on Memory Systems Performance and Correctness*, 1-3pp.

Ransford, B., Sorber, J. and Fu, K., 2011, Mementos: System support for long-running computation on rfid-scale devices, *in Proceedings of the Ssixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, 159-170pp.

Schill, H., 2017, Debugging Xtext grammars – what to do when your language is ambiguous, https://blogs.itemis.com/en/debugging-xtext-grammars-what-to-do-when-your-language-is-ambiguous (Erişim: Ekim 2019)

Smith, J. R., 2013, Wirelessly powered sensor networks and computational RFID. *Springer Science & Business Media, 271p.*

Smith, J. R., Sample, A. P., Powledge, P. S., Roy, S. and Mamishev, A., 2006, A wirelessly-powered platform for sensing and computation, *International Conference on Ubiquitous Computing*, 495-506pp.

Soyata, T., Copeland, L. and Heinzelman, W., 2016, RF energy harvesting for embedded systems: A survey of tradeoffs and methodology, *IEEE Circuits and Systems Magazine*, 16(1), 22-57pp.

KAYNAKLAR DİZİNİ (devam)

Surbatovich, M., Jia, L. and Lucia, B., 2019, I/o dependent idempotence bugs in intermittent systems, *in Proceedings of the ACM on Programming Languages*, 3(OOPSLA), 1-31pp.

Tan, J., Pawelczak, P., Parks, A. and Smith, J. R., 2016, Wisent: Robust downstream communication and storage for computational rfids. *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications* (pp. 1-9). IEEE.

Texas Instruments, 2015, MSP430FR5969 LaunchPad Development Kit, http://www.ti.com/tool/MSP-EXP430FR5969 (Erişim: 1 Mayıs 2020)

Texas Instruments, FRAM faqs. http://www.ti.com/lit/ml/slat151/slat151.pdf (Erişim: 1 Mayıs 2020)

TU Delft Sustainable Systems Laboratory, InK, https://github.com/TUDSSL/InK (Erişim: Ocak 2020)

Van Der Woude, J. and Hicks, M., 2016, Intermittent computation without hardware support or programmer intervention, *in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 17-32pp.

Welch, T. A., 1984, A technique for high-performance data compression, *Computer*, 8-19pp.

Wright, D. and Freeman-Benson, B., 2004, How to write an eclipse debugger, https://www.eclipse.org/articles/Article-Debugger/how-to.html (Erişim: Kasım 2019)

Xtext, The Grammar Language, https://www.eclipse.org/Xtext/documentation/301_grammarlanguage.html (Erişim: Aralık 2019)

Yıldırım, K. S., Majid, A. Y., Patoukas, D., Schaper, K., Pawelczak, P. and Hester, J., 2018, Ink: Reactive kernel for tiny batteryless sensors, *in Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, 41–53pp.

TEŞEKKÜR

Yüksek lisans eğitimim boyunca bana yol gösteren, ufkumu açan hocam ve danışmanım Doç. Dr. Kasım Sinan Yıldırım'a teşekkürlerimi sunarım. Ayrıca kariyerimin başlangıç noktasından bu zamana kadar destek olan Aytaç Zeren'e teşekkürü borç bilirim. Bu mesleği seçmemde öncülük eden sevgili annem Ümmü Mülayim başta olmak üzere tüm aileme ve sevgili nişanlım Züleyha Aylin Aktuna'ya varlıkları, destekleri ve sabırları için teşekkür ederim.

09 / 06 / 2020

Murat MÜLAYİM

ÖZGEÇMİŞ

Murat MÜLAYİM
Altayçeşme Mh., İlyasağa Çıkmazı Sk., No:4 MALTEPE / İSTANBUL
+90 546 547 38 83
murat.mulayim74@gmail.com

Kişisel Bilgiler
Uyruk: T.C.
Doğum Tarihi: 10.09.1993
Doğum Yeri: BARTIN
Sürücü Belgesi: B
Askerlik Durumu: Yapıldı

Eğitim

- 2017-2020 – Yüksek Lisans
  - Ege Üniversitesi – Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği
- 2012-2016 – Lisans
  - Ege Üniversitesi – Mühendislik Fakültesi Bilgisayar Mühendisliği
- 2007-2011 – Lise
  - Bartın Davut Fırıncıoğlu Anadolu Lisesi

Yayınlar

- Taskify: An Integrated Development Environment to Develop and Debug Intermittent Software for the Batteryless Internet of Things *WPSN'2020*

Staj

- 2015, Alcatel-Lucent – İSTANBUL

Deneyim

- 2018, Ar-Ge Yazılım Geliştirici, SIEMENS - İSTANBUL
- 2016, Ar-Ge Yazılım Geliştirici, Ericsson – İZMİR
- 2016, Ar-Ge Yazılım Geliştirici, Ericsson – İZMİR (Yarı Zamanlı)

TERİMLER SÖZLÜĞÜ

Türkçe	İngilize
Alana özel dil	Domain Specific Language
Arayüz	Interface
Bayt	Byte
Çalışma zamanı kütüphanesi	Runtime-library
Çalışma zamanı ortamı	Runtime environment
Çift tamponlama	Double buffering
Derleme zamanı	Compile-time
Dışa aktarmak	Export
Enerji hasadı birimi	Energy harvestor
Geçici bellek	Volatile memory
Geçici olmayan bellek	Non-volatile memory
Geri saçılım birimi	Backscatter
Gelişigüzel versiyon	Scratch copy
Girdi-çıktı tabanlı	I/O-based
Guguk Filtresi	Cuckoo Filtering
Hata ayıklama	Debugging
Hata ayıklayıcı	Debugger
İki aşamalı işleme	Two phase committing
İş çerçevesi	Framework
İşleyici	Executor

TERİMLER SÖZLÜĞÜ (Devam)

Türkçe	İngilize
Jeton	Token
Kapsam	Scope
Kontrol noktası tabanlı	Checkpointing-based
Model muhakeme araçları	Model reasoning tools
Nesnelerin İnterneti	Internet of Things
Pilsiz cihazlar	Batteryless devices
Radyo frekansı	Radio frequency
Radyo frekansı okuyucusu	RFID reader
Soğuk Zincir Ekipmanı İzleme	Cold-Chain Equipment Monitoring
Soyut sözdizim ağacı	Abstract Syntax Tree
Sözcüksel çözümleyici	Lexer
Ya hep ya hiç	All-or-nothing