Yazılım Mimarileri Tanımlama Amaçlı Alana Özgü Programlama Dili

Sezer Akar[1], M. Ufuk Çağlayan

Özet:

Bu çalışmada yazılım mimarileri tanımlama amaçlı alana özgü programlama dili oluşturulmaya çalışılmıştır. Amaç yazılım mimarilerini tanımlamayı kolaylaştırarak, yazılım mimarilerinin tüm faydalarından yararlanmaktır. Buradaki programlama dili, DSLSA, ile tanımlanan yazılım mimarisi bir başvuru dokümanı olmaktan ziyade yazılım geliştirme sürecinin temel yapıtaşlarından biri olacaktır. Bütün sistemi ilgilendiren önemli özelliklerin, yapısal özellikler ya da güvenlik gibi, yazılım geliştirme sürecinin başında, mimari üzerinde tanımlanması ve kontrol edilmesi hedeflenmiştir. DSLSA ile tanımlanan mimari bu önemli özelliklere göre kontrol edildikten sonra eğer geçerli bir mimari ise uygulama kodu mimari üzerinden direkt üretilebilir. Yazılım mimarileri, tasarımdan farklı olarak fonksiyonel olmayan özellikleri içerdiğinden üretilen kaynak kod, uygulama geliştiriciler için bir çerçeve sağlayacaktır. Uygulama alanına özgü geliştirmenin geliştirici tarafından yapılması gerekmektedir. Ancak bu çerçeve yapı yazılımın bir bütün olarak kalmasını, yazılım geliştirme sürecin en başında konan mimari özellikleri kaybetmemesini sağlar.

Alana özgü dil oluşturma sürecinin ilk aşaması üst düzey bir modelde yaratılacak dilin soyut söz dizim ağacını oluşturmakla başlar. Bu çalışmada üst düzey model olarak EMF Core (Ecore) [5] seçilmiştir. DSLSA Ecore'da tanımlandıktan sonra, dilin söz dizimi grafiksel olarak tanımlanmıştır. Bunun için Eclipse GMF kütüphanesinden yararlanılmıştır. Dil tanımlandıktan sonra oluşturulan ara yüz üzerinde dilin kurallarına uyan herhangi bir yazılım mimarisi tanımlanabilir. Yazılım mimarı, mimariyi tanımladıktan sonra, DSLSA sayesinde dilde tanımladığı bir takım özelliklere göre kontrol edebilir. Doğruluğu ispatlanan mimari üzerinden JAVA kodları üretilir ve uygulama geliştiricilere sunulur. Üretilen kod üzerinde, uygulama alanına ait ve mimari özellik taşıyan kaynak kodlar birbirinden ayrılmıştır, dolayısıyla mimari değiştiği ve tekrar kod üretildiği zaman geliştiricilerin önceden yazdıkları kaynak kodlar kaybolmaz.

Bu çalışmanın amacı, değiştirmesi kolay ve çalıştırılabilir mimarileri tanımlamayı kolaylaştırmaktır. Bu sayede yazılım ilerledikçe mimarinin yazılımın kendisinden kopup sadece bir doküman haline gelmesi engellenmek istenmiştir. Art arda döngüler halinde tanımlanan bir yazılım mimarisinin bütün sistemi bir arada tutan kaynak kodları üretebilecek ve gereksinimleri karşılayacak bir yol gösterici olacağı düşünülmüştür.

Anahtar Kelimeler: Alana özgü diller, kaynak kod üretimi, model-güdümlü yazılım geliştirme, yazılım mimarileri

[1] Boğaziçi Üniversitesi, Bilgisayar Mühendisliği Bölümü, İstanbul
sezer.akar@boun.edu.tr, caglayan@boun.edu.tr

1. GİRİŞ

Alana özgü diller, belirli bir uygulama alanı için geliştirilmiş programlama dilleri olarak tanımlanabilir. Bu kavram yeni değildir ancak son zamanlardaki alana özgü modelleme ve model-güdümlü yazılım geliştirme yöntemleri sayesinde tekrar gündeme gelmiştir. Alana özgü dillere örnek olarak, donanım tasarım alanı için VHDL, yazılım derleme alanı için Make, söz dizimi yazım alanı için BNF gösterilebilir. Alana özgü diller genelleştirmeye karşın belirli bir alanda özelleşmeyi ve anlaşılabilirliği tercih eder. Bu yaklaşım belirli alanda daha yüksek düzeyde bir dili, insanın daha kolay anlayabileceği bir programlama dilini teşvik eder. Bunun sonucunda yazılım geliştirme takımı içerisinde ve iş uzmanları ile geliştirme takımı arasında iletişim artar. Alana özgü diller, programlama yapabilmek için gereken bilgiyi azaltır ve uygulama alanını daha geniş bir kitleye açar. Bunlara ek olarak, şu sebeplerden ötürü insanlar alana özgü dilleri genel amaçlı programlama dillerine tercih ederler : [1]

- Alana özgü diller genel amaçlı dillerde sınırlı olan işlem kümesiyle kısıtlı kalmazlar ve kullanıcının kendi işlemlerini tanımlamalarına olanak sağlarlar.

- Çoğu zaman alana özgü kavramlar direkt olarak bir fonksiyona ya da bir nesneye birebir eşlenecek basitlikte olmayabilir.

- Alana özgü dillerin kullanımı, analiz, doğrulama, kaynak kod üretme gibi bir çok alanda genel amaçlı programlama dillerine oranla kolaylıklar sağlar.

Model-güdümlü yazılım geliştirme yöntemi, yazılım geliştirme sürecinin başında modellerin oluşturulması daha sonra mümkün olduğu kadar kaynak kodun bu modellerden oluşturulması ilkesine dayanır. Microsoft Software Factory, Eclipse Modeling Project model-güdümlü yazılım geliştirme girişimlerine örnektir. Son yıllarda alana özgü dillerin model-güdümlü yazılım geliştirmeyi destekleyecek şekilde kullanımı artmıştır.

Yazılım mimarileri ise tanımı üzerinde dahi tartışmaların sürdüğü bir konudur. Bu çalışmada kullanılan yazılım mimarisi tanımı, onu çok karıştırıldığı detaylı tasarımdan ayıracak şekilde ortaya konmuştur. Kısaca; yazılım mimarileri fonksiyonel olmayan, performans, güvenlik gibi gereksinimlerle ilgiliyken, detaylı tasarım daha çok fonksiyonel gereksinimlerle ilgilidir. Bu ayrımı daha belirgin hale getirmesi için, Intension/Locality Hypothesis' den yararlanılmıştır. [2] Bu hipotezdeki yerellik özelliğine göre, eğer bir kural yazılımın herhangi bir yerinde bozulabiliyorsa yerel değildir ve bu yüzden detaylı tasarıma değil yazılım mimarisine aittir.

Bu çalışmanın amacı, yazılım mimarisini tanımlama amaçlı bir programlama dili gerçekleştirmektir. Bu dil sayesinde yazılım mimarisini herkesin anlayacağı bir sadelikte oluşturmak ve kaynak kodların hatırı sayılır bir kısmını direkt olarak mimariden üretmek hedeflenmiştir. [4]

Çalışmanın ikinci bölümünde problem açıklanmıştır. Üçüncü bölümde çalışma süresinde geliştirilen uygulamadan, son bolümde ise sonuç ve önerilerden bahsedilmiştir.

2. PROBLEM TANIMI

Birçok yazılımda mimari yazılım geliştirme sürecinin bir parçası olmaktan ziyade, güncellenmeyen dokümanlar olarak kalmaktadır. Bunda yazılım mimarisinin tanımlanmasında kullanılan dilin çoğunlukla UML gibi genel bir dil olması yüzünden mimari gereksinimlerin kolaylıkla tanımlanamaması, gerçekleştirilen mimariden kaynak kod üretilememesi, kaynak kod üretilse dahi mimariden kod üretme işinin yazılım geliştikçe kolaylıkla tekrarlanamamasının payı büyüktür. Çoğunlukla gelinen son nokta mimarinin kimsenin başvurmadığı ve gerçeği yansıtmayan bir doküman halini almasıdır.

3. UYGULAMA

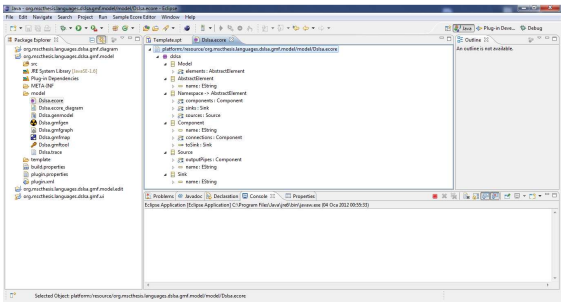
Alana özgü dil geliştirme aşağıdaki adımlardan oluşur. [3]

1. *Soyut söz dizimi ağacı oluşturma* : Bu aşamada yaratılacak dilin üst düzey modeli oluşturulur. Buradaki uygulamada ilgili adım, Eclipse Ecore üst düzey modeli kullanılarak yaratılacak dilin üst düzey modelini oluşturmadır.

2. *Somut söz dizimi oluşturma* : Alana özgü dilin kullanıcıları somut bir söz dizim kullanarak kendi model örneklerini oluşturabilmeliler. Bu söz dizimi metin olabileceği gibi, görsel de olabilir.

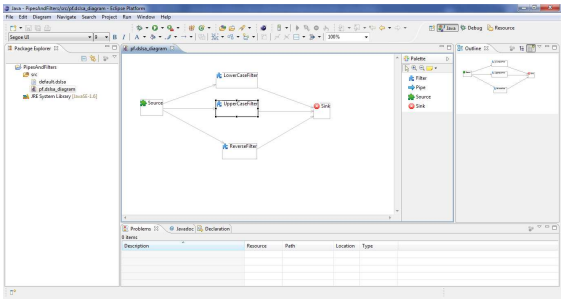
3. *Model-Metin ya da Model-Model dönüşümü :* Model-Metin dönüşümleri bir model i kaynak kod gibi metinlere dönüştürürken, model-model dönüşümleri bir modelden genelde daha basit bir modele dönüşüm sağlar.

DSLSA tanımlanırken birebir aynı adımlar izlenmiştir. Önce üst düzey modeli EMF core (Ecore) kullanılarak tanımlanmıştır. Şekil-1'de DSLSA üst düzey modeli görülebilir. Bu aşamada yapılan Eclipse Ecore kullanarak DSLSA 'nin soyut söz dizim ağacını oluşturmaktır.



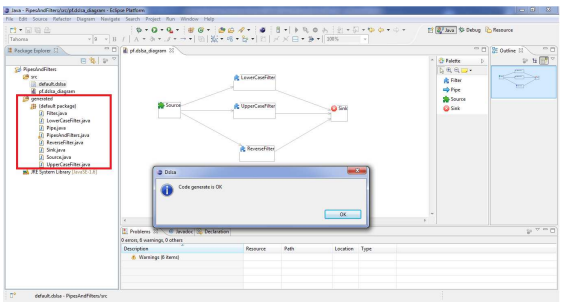
Şekil-1 DSLSA Üst Düzey Model

DSLSA görsel dil olarak tanımlanmıştır. Dolayısıyla somut söz dizimi DSLSA ara yüzünde kullanıcının yapabileceği işlemler şeklinde düşünülebilir. Bunun için Eclipse Graphical Modeling Framework (GMF) [5] kullanılmıştır. DSLSA soyut söz dizimi ağacındaki elementler için gösterimler, elementler arasındaki ilişkilerin ve arayüzde gösterilecek araç biriminin tanımları bu aşamada yapılmıştır. Şekil-2'de bu durum görülebilir. Buradaki örnekte DSLSA kullanılarak bir "Pipes And Filters" mimarisi tanımlanmıştır. "Pipes And Filters" mimarisinin özelliklerinden olan "Source" 'a hiçbir girdinin olmaması, "Sink" 'ten hiçbir çıktının olmaması gibi özellikler DSLSA görsel dili üzerinde tanımlanmış, böylece kullanıcının bu mimari stile uymayan bir mimari tanımlamasının önüne geçilmiştir. DSLSA ile tanımlanan mimari bu anlamda geçerli bir model örneğidir.



Şekil -2 DSLSA Arayüzü

DSLSA ile model örneği, yani mimari, tanımlandıktan sonra sıra kaynak kod (Model-Metin) dönüşümüne gelir. Bu aşamada kaynak kod üretme kütüphanesi olan XPAND' dan faydalanılmıştır. [5] XPAND' da DSLSA ile uyumlu şablonlar tanımlanarak her bir model elementinin karşılığı üretilecek JAVA kodu belirlenmiştir. Şekil-3'te DSLSA arayüzü üzerinde kullanıcının kaynak kod üretme işlemi ve üretilen kaynak kod görülebilir.



Şekil-3 DSLSA Kaynak Kod Üretimi

4. SONUÇ VE ÖNERİLER

Yazılım geliştirme sürecinin en önemli sorunlarından biri de geliştirme sürecinin en başında "istenen" ile geliştirme sürecinin sonunda "gerçeklenen" arasındaki farktır. Gerçeklenen yazılım en başta verilen gereksinimlerden çok farklı olabilir. Bu çalışmada yapılan alana özgü programlama dili, DSLSA 'nın bir amacı da gereksinmeler ile gerçeklenen yazılım arasındaki uçurumu kapamaya yardımcı olmaktır. Yazılım geliştikçe gelişecek, değiştirmesi kolay, çalıştırılabilir bir yazılım mimarisi tanımlamak DSLSA' nın hedefidir. Bu şekilde tanımlanan bir mimari, tüm sistemi bir arada tutan ve gereksinimleri karşılayan bir yapı olacaktır.

İleriki çalışmalarda, DSLSA' yı daha karmaşık mimari yapıları destekleyecek ve mimariden üretilen kaynak kodu ağ üzerindeki farklı makineler üzerine otomatik yerleştirecek şekilde geliştirme düşünülmektedir.

5. KAYNAKLAR

[1] M. Mernik et al., "When and how to develop domain-specific languages," ACM Computing Survies, vol. 37, no. 4, pp. 316–344, 2005

[2] Amnon H. Eden, Rick Kazman (2003). "Architecture, Design, Implementation", ICSE '03 Proceedings of the 25th International Conference on Software Engineering

[3] R. C. Gronback, "Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit", Addison-Wesley Professional, 2009.

[4] Markus Voelter, "Architecture As Language", IEEE Software Magazine, vol. 27, issue:2, pp. 56-65,2010

[5] Eclipse Modelling Project http://www.eclipse.org/projects/project.php?id=modeling