Yazılımdaki "Aspect"lerin Bileşene Dayalı Yazılım Mühendisliği Yardımıyla Ele Alınması

Barış UZ

ODTÜ, Bilgisayar Mühendisliği, Ankara GeoTech Group

e-posta: baris_uz@yahoo.com, buz@ags-group.com

Özet

Bu çalışma, "aspect"lerin neden bileşenlerden farklı olarak ele alınması gerektiğini açıklamak, "aspect"leri yeni bir bileşen haline getirmeden yapılması gerekenleri vurgulamak ve tüm yapıtaşları ile birlikte bir çerçeve çizmektir. Bu çerçeve dahilinde, "aspect"e dayalı yazılım mühendisliğinin gerekleri tartışılacaktır.

Abstract

This study is to explain why "aspect"s should be considered different than components, to emphasize necessary steps to avoid "yet-another-component result" in the implementation of "aspect"s and to define a framework for "aspect"-oriented software engineering together with all of the building blocks. Within this context, requirements of "aspect"-oriented software engineering will be discussed.

1. Giriş

Oldukça yeni ve ilgi çekici bir yazılım mühendisliği dalı olan "aspect"e[1] dayalı yazılım mühendisliğinin daha verimli ve etkili uygulamaya geçirilmesi için çalışmalara devam etmektedir. Bileşene Dayalı Yazılım Mühendisliği'nin (BDYM'nin) getirdiği yaklaşımlar, "aspect"lerin uygulanmasında gerçekten "aspect"e dayalı yazılım geliştirilmesi yerine, "aspect"lerin bir bileşen gibi ele alınması sonucunu doğurmuştur.

Son yıllarda giderek daha da fazla önem kazanmaya başlayan "aspect"e dayalı yazılım mühendisliği konusu, tüm yazılım geliştiricilerin ciddi anlamda çaba sarfettiği bir alan olmuş ve özellikle son dönemlerde Internet'in popüler olmasıyla, yazılım mimarilerinin her seviyesinde sağlanması gereken gizlilik ve güvenlik gibi konular dolayısıyla da yaygın bir kullanım alanı bulmaya başlamıştır.

"Aspect"e dayalı yazılım mühendisliği, tasarım aşamasından başlayarak alışılagelmişin dışında bir yaklaşım gerektirmektedir [1]. Ne yapısal tasarımdan alışkın olduğumuz aşağıdan yukarıya tasarımla, ne de nesneye dayalı tasarımdan alışkın olduğumuz yukarıdan aşağıya tasarımla

[1] Bu makalede, İngilizce "aspect" kelimesinin Türkçe karşılığı olan "bakış açısı" kullanılmamış, terminoloji yerine oturuncaya kadar "aspect" kelimesinin kullanılması yeğlenmiştir.

halledilebilmektedir. “Aspect”ler, tüm yazılım mimarisindeki seviyelerde yer alan, ve tüm seviyelerle ilişkisi olan bir kavram olduğundan, “aspect”e dayalı yazılım tasarımı sırasında çok dikkatle ele alınmalı ve her seviyedeki işlevi dikkatle çözümlenmelidir.

Bu bağlamda, tasarıma başlamadan önce, “aspect”lerin ayrıştırılması ve nasıl ele alınacağı konusu oldukça önem kazanmaktadır. Detaylı çözümleme, “aspect”’e dayalı yazılım mühendisliğinde en önemli noktalardan biridir ve dikkatle yerine getirilmelidir [1]. “Aspect”ler, yazılımdaki tüm mantıksal seviyeleri etkilediğine göre, gerektiği yerde işletim sisteminin de gerekli müdahaleleri yapabilmesi, üretilen kodun bu tür kesintilere izin vermesi ve tüm yapının tam ve uyumlu bir biçimde çalışabilmesi gerekmektedir. Eğer “aspect”leri, belli bir görevi yerine getiren bir yapı olarak görüp, gerektiği yerde, gereken yöntem ve işlevlerini çağırırsak, “aspect”ler sıradan bir bileşen olacak ve “aspect”ten beklenen işlevi yerine getiren yeni bir bileşen halini alacaktır.

Bir doktora tezinin ön çalışmasını ve yöntemler bütününü özetleyen bu makaledeki amacımız, “aspect”lerin bileşenlerden neden ve nasıl ayrılması gerektiğini tartışmak ve gerekli çerçeveyi çizmektir. Bu çalışma dahilinde, “aspect”lerin, bileşene dayalı yazılım mühendisliğinde nasıl ele alınabileceği, bileşene dayalı yazılım mimarisinin “aspect”e dayalı yazılım mimarisini ele almadaki eksikliklerinin nasıl giderilebileceği, önerilecek çerçevedeki modüllerin çözümlenerek ortaya çıkarılması ve gerçekleştirime yönelik kestirimler özetlenmiştir.

2. Öncalışma

Bu makalede tartışılacak olan yönteme esas olan “aspect”e dayalı yazılım mühendisliği (ADYM) kavramı, XEROX Palo Alto Research Center’da ortaya çıkarılmıştır. [2]’de anlatıldığı gibi, “aspect”e dayalı programlama, bir yazılımın tümünde aynı ya da benzer şekilde gerçekleştirilen işlemlerin (*cross-cutting concerns*) soyutlanmasına (abstraction) yöneliktir. Burada, bileşenleri ve “aspect”leri birbirinden açık bir şekilde ayırmak gereklidir. ADYM’nde amaç, bileşenleri diğer bileşenlerden, “aspect”leri diğer “aspect”lerden ve bileşenleri de “aspect”lerden ayırmaktır.

Yine aynı makalede özetlendiği gibi, bir matris uygulamasındaki “aspect”lere örnek olarak, matrisin gösterimi, permutasyonları ve matematiksel hatalar verilebilir. Bizim çalışmamıza örnek teşkil eden coğrafi bilgi sistemlerinde ise, yine benzer şekilde, coğrafi gösterimler, coğrafi işleçler, koordinat dönüşümü fonksiyonları birer “aspect” olarak ele alınabilir. Bunlar, uygulamanın bütününde, ya da bir çok yerinde tekrarlanmakta ve bu bağlamda da, uygulamayı kesen bir yapı sergilemektedir.

Burada yapılan çalışmada, “aspect”lerin gerçekleştirimi oldukça önemli gibi görünse de, esas önemli olan altyapıdır. Alt yapıda her bir “aspect” birer bileşen olarak gerçekleştirilebilecektir. Burada çalışma zamanı ortamına da müdahale etmek gereklidir. Bu konuda bir çalışma [3]’de gösterilmiştir. Çalışma zamanı ortamına müdahale edilmemiş, sadece kodun hangi bölümünde hangi “aspect”in çalışacağı belirtilmiştir. Bundan farklı olarak, bu makalede önerilen, hangi bölümde hangi “aspect”in çalışacağının çalışma zamanında belirlenmesi ve yazılımla ilgili gerekli tüm korumanın (bellek (memory), yığın (heap) ve yığıt (stack) korumaları gibi) çalışma zamanı ortamında ele alınmasıdır.

3. Hazırlık – “Aspect” mi, bileşen mi?

Yazılım Mühendisliği kapsamında yer alan bileşenler, yazılım mühendisliğinin, elektronik mühendisliği gibi, birbirine entegre olmuş devreler gibi geliştirilmesine olanak sağlayacak birer

yapıtaşı olarak düşünülmüşlerdir. [5]'de anlatıldığı gibi, sistemlerin bileşenlerden yaratılması da kurulu bir yöntemler bütünü ve süreci gerektirir. Her yazılımın kendine has gereksinimleri ve farklı koşulları nedeniyle, bu düşüncenin gerçekleştirilmesi oldukça zaman almış ve bugünkü anlamıyla bileşene dayalı yazılım geliştirme yöntemi ortaya atılmıştır. Temelde, yeniden kullanılabilirlik ilkesine dayanan bu yöntemde, bileşenlerin tek başlarına nasıl geliştirildiği önemli değildir. Yazılımın tamamı, bir çeşit entegrasyon projesi haline gelmekte ve edinilen bileşenlerin nasıl entegre edileceği, bu entegrasyon sırasında ortaya çıkabilecek sorunların çözümlenmesi temel sorunları teşkil etmektedir. "Aspect"lerin bu konuda nasıl yardımcı olabileceğine ait bir çalışma [4]'te verilmiştir. Örnek problemler yardımıyla, nesneye dayalı dillerdeki yeniden kullanılabilirlik, kalıtım (inheritance) gibi problemlerin nasıl çözülebileceği anlatılmıştır.

"Aspect"ler ise, yazılım mühendisliği disiplini için yeni bir konudur. Önceki bölümde de özetlendiği gibi, bir araştırma geliştirme projesi olarak ortaya çıkmış ve yazılım mühendisliğinin özelleşmiş bir bölümüdür. "Aspect"ler, programlar tarafından paylaşılan, gerçekleştirilmesi göreli olarak zorluk teşkil eden yazılım kesimlerini oluşturur, ancak yazılım mimarisindeki katmanların (tier) tamamında ele alınması gereken yazılım kesimlerini içerir. Sıklıkla verilen, güvenlik "aspect"i örneği, yazılım mimarisindeki katmanların hepsinde geçerli olan ve yapılan işlemlere izin/red cevabı veren bir birimi teşkil eder. Güvenlik açısından yazılımın tüm parçalarının bu "aspect"le ilgili yapacakları olacaktır. Örneğin, tüm okuma ve yazma işlevleri, bu "aspect"in kontrolünde olacaktır.

Bu bağlamda, "aspect"ler gerektiğinde çağırılması gereken modüller olarak gözükebilir. Esas amacı bileşenleri ve "aspect"leri birbirinden ayırmaya yarayan mekanizmaları tanımlamak olan "Aspect"e Dayalı Programlama [2], yaygın olarak kullanılagelen nesneye dayalı ya da bileşene dayalı programlama yaklaşımlarını da desteklemektedir. Gerçekte "aspect"ler bir takım kod parçalarıdır. Bu kod parçaları, gerekli modülün başında veya yazılım modülünün başında tanımlanmakta ve ilgili oldukları alan (scope) belirlenmektedir. Yani, bir modülün içinde yer alan bir yordamın güvenlikle ilgili bir ihtiyacı olduğunu, kodlayıcı bilmek zorundadır ve bunu bilerek kodunu yazacaktır. İyi niyetli kullanıcılar açısından bu konuda herhangi bir sıkıntı olmamakla birlikte, kötü niyetli kullanıcılar "aspect"lerin gerektirdiği bu referanslama ya da ilişkilendirme mekanizmalarını kodun içine koymayarak, hem bilgiyi saklayabilir, hem de kötü amaçlı programlar yaygınlaşabilir.

Bu yaklaşımdaki en temel sıkıntı, aslında "aspect"lerin de birer bileşen gibi düşünülmesidir. Burada, aslında "aspect"ler, "aspect" olarak düşünülmektedir, ancak gerçekleştirimde, özel bir yeri olan, özellik arzeden bileşenler olarak gerçekleştirilmektedir. Yazılım Mühendisliği'nin geldiği bugünkü noktada, "aspect"lerin de birer bileşen olması adeta kaçınılmazdır. Zaten BDYM'nin temel amacı da yapı taşlarını kullanmaktır.

Uygulamadaki temel tıkanıklık, "aspect"lerin çalışma zamanı ortamı (run-time environment) gibi temel bir ögeden hariç düşünülmesidir. "Aspect"e dayalı yazılım geliştirme için en önemli konulardan biri olan çalışma zamanı ortamında "aspect"lerin nasıl müdahale edeceği hakkında farklı yorumlamalar mevcuttur. [3]'te anlatılan modelde, "aspect"lerin çalışan koda ne zaman ve nasıl müdahale edilmesi gerektiği açıkça belirtilmiştir. Ancak bu modelde çalışma zamanı ortamı değiştirilmemiştir. Mevcut bileşenler yardımıyla, kodun neresinde "aspect"lerin müdahalesi olacağı ifade edilmiştir. "Aspect"lerin birleşme ve ayrılma mantıkları, başka bir çalışmada yine kod seviyesinde ele alınmıştır [1].

4. Çerçeve Gereksinimleri

Bu bölümde, "aspect"e dayalı yazılım geliştirmek için gerekli olan çerçevenin (framework) tanımlanması ve ona göre ortam geliştirilmesi anlatılacaktır.

[3]'de anlatıldığı gibi, bu kapsamda oluşturulacak bir çerçevede en azından aşağıdaki özelliklerin bulunması gereklidir:

- "Aspect"lerin bağlanabileceği ve işin içine müdahale edebileceği bir çalışma ortamı.
- COM bileşenleri olarak gerçekleştirilmiş, kullanıcının tanımladığı COM bileşenleri.
- Hangi "aspect"in hangi COM bileşeni yardımıyla gerçekleştirileceğinin belirtildiği bir katalog.
- Yazılım içinden bileşenlerin "aspect"leri aktif hale getirmesine yarayacak bir yöntem.

Burada çalışma ortamı tanımlanmıştır. Ancak koda müdahale etmek için gerekli olacak kurallar tam olarak belirlenmemiş ve gerçekleştirim için açık bir kapı bırakılmıştır. Bizim yaklaşımımız da, benzer bir modeli izleyecek, ancak kod çalıştırmadan önce, kod çalıştırılması sırasında ve sonrasında gerekli işlevlere müdahale edecek bir sanal makine (virtual machine) geliştirilmesi önerilecektir. Zira, "aspect"lerin ele alacağı işlevleri, bir COM bileşeni yardımıyla yapmayı önermek, bizim konumumuzda yeni bir çeşit COM bileşeni ortaya çıkarmak olur ki, bunun da BDYM'den farkı kalmaz. Bizim tasarlayacağımız "aspect"ler, akıllı olacak ve müdahale gerektirecek yerleri erken bağdaştırma (early binding) ya da dinamik bağdaştırma (dynamic binding) yoluyla tesbit ederek müdahale edecektir.

Geliştirilecek çerçevede, yukarıda sayılanlara ek olarak

- Oluşturulan kod parçasının platformdan bağımsız yürütülmesini sağlayacak bir sanal makine ve
- "Aspect"lerin hangi bileşen/yordam aracılığıyla gerçekleştirildiğinin belirtilmesine (diğer bir deyişle, "aspect" kataloğu oluşturulmasına) yarayan bir dil (Markup-language)

tasarlanması planlanmaktadır.

İdeal olarak, sanal makine kodda herhangi bir değişiklik yapmadan, kod seviyesinde açıkça belirtme gerektirmeden, ilgili "aspect"'in müdahalesini sağlayan bir mekanizma olarak düşünülmüştür.

İkinci maddede sözü edilen dil, aslında bir "aspect" kataloğu ortaya çıkarılmasını amaçlamaktadır. Bütün sektörleri kapsayan bir katalog üretilmesi, bu çalışmanın amaç ve sınırlarının çok ötesindedir. Ancak bu kataloğun gerçekleştirime ne tür katkıları olduğu ya da olmadığının gösterilmesi için, coğrafi bilgi sistemleri için örnek bir katalog oluşturulacaktır.

5. Çerçevenin Gerçekleştirilmesi

Ortaya konulan önerinin gerçekleştirilmesinde bir coğrafi bilgi sistemi uygulaması yapılacaktır. Bu gerçekleştirim üzerinde "aspect"lerin rolü, nasıl ele alınacağı, koda nasıl müdahale etmeleri gerektiği açıklanmaya çalışılacaktır.

"Aspect"e dayalı yazılımlara bir takım örnekler [2]'de verilmiştir. Buna bir örnek de, coğrafi bilgi sistemlerinde yer alan spatial (konumla ilgili) işleçler ve koordinat dönüşümlerine ilişkin yordamlardır. Bilindiği üzere, coğrafi bilgi sistemleri, bilgileri içeren katmanlardan oluşan ve coğrafi veri tabanları üzerinde işlem yapan yazılım sistemleridir. Her katmanın ayrı bir özelliği olup, bu özellikler nokta, çizgi ve çokgen olabilmektedir. Konumsal (spatial) işleçler, farklı

katmanlarda yer alan coğrafi nesnelerin birbirleriyle kesişim, birleşim, teğet olma ve benzeri konumsal işlemlerini yapmaya yarar. Coğrafi bilgi sistemleri nasıl geliştirilirse geliştirilsin, coğrafi bilgi sistemlerinde yazılımın tümünde aynı ya da benzer şekilde gerçekleştirilen işlemler (cross-cutting concerns), yani "aspect"ler aşağıdaki şekilde listelenebilir:

1. Konumsal işleçler – kesişim, birleşme, teğet olma, içinde olma, dışında olma, tamamen içinde olma, kapsama, vb.

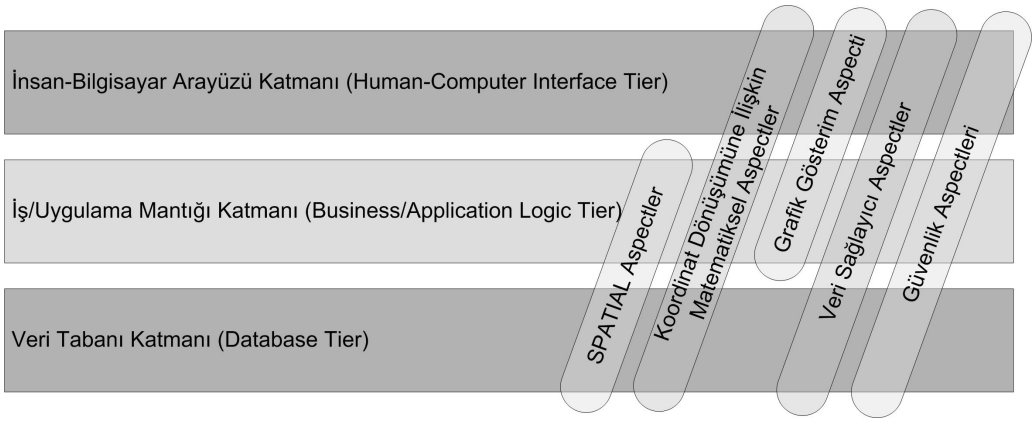
2. Koordinat dönüşümleri – çeşitli koordinat sistemleri arasında dönüşüm yapmaya yarayan matematiksel formüllere dayalı "aspect"ler.

3. Grafik yordamlar – gerekli verilerin (nokta, çizgi, çokgen) ekrana çizilmesini sağlayan "aspect"ler.

4. Veri sağlayıcı sistemler – coğrafi bilgi sistemi veri tabanından bilgi çekmeye yarayan "aspect"ler.

5. Güvenlik sistemleri – coğrafi bilgi sistemindeki işlevlerin kısıtlanması ya da yetki gerektiğinde devreye girecek "aspect"ler.

Çalışmamız, bu tür bir yazılımın geliştirilmesi üzerine kurulu olacaktır. Şekil 1'de, "aspect"lerin değişik yazılım modüllerini mantıksal olarak nasıl kestiği gösterilmiştir.



Şekil 1. Bir coğrafi bilgi sistemi uygulamasındaki "aspect"ler.

Burada programın genel yapısal mimarisi tanımlanmış ve katmanlar belirlenmiştir. Bu katmanlar arasında ortak kullanımda olan bazı önemli "aspect"ler vardır. Bu "aspect"ler tüm katmanları kesebileceği gibi, yalnızca birkaç katman için geçerli de olabilir. Örnek verecek olursak, güvenlikle ilgili "aspect", tüm katmanlar için geçerli bir "aspect" iken, konumsal "aspect" yalnızca verinin oluşturulmasına/işlenmesine ilişkin katmanlar olan Veri Tabanı ve İş/Uygulama Mantığı Katmanlarında geçerlidir. (Yukarıdaki şemada verilen katmanlar, kendi içlerinde birden fazla bileşen tarafından gerçekleştirimi yapılan katmanlardır. Her biri atomik bir bileşen değildir.) Bu bağlamda, gerçekleştirim sırasında "aspect"lerin içerisinde gerçekleştirilecek tüm işlevler sanal makine aracılığıyla anlaşılacak ve kod yürütülmesi işlemi bu sanal makine tarafından gerekli

“aspect”e yönlendirilecektir. Hangi “aspect”in ne zaman devreye gireceği konusu “aspect” kataloğunda belirlenmiş olacaktır.

Bu bağlamda, aslında modüler olarak ayrıştırılmış programımızın farklı katmanlarında (tier) “aspect”lerden nasıl faydalanacağı açıkça görülmektedir.

Her bir “aspect”in kendisinden beklenen işlevi yerine getirmesi için gerekli kod parçacıkları, çeşitli bileşenler aracılığıyla yazılmış ve hazır bulunmaktadır. Bu çalışmadaki temel varsayımımız budur.

Ana sorular, kod çalışması sırasında, “aspect” koduna geçişle ilgilidir:

1. Geçişi hangi birim yapacak?
2. Bu geçiş sürecinde gerekli bellek ve disk erişimi korumalarından hangi birim sorumlu olacak?
3. Hangi “aspect”e geçiş yapılacağını hangi aşamada (early-binding, dynamic-binding, late-binding) belirteceğiz, hangi aşamada ele alacağız?

Bu soruların cevabının çoğunun program geliştiren kişiye bırakılması, yeni bir bileşene dayalı yazılım mimarisi getirecektir.

Soruları cevaplamaya tersten başlarsak, genel bağlamı şöyle özetleyebiliriz:

Hangi “aspect”e geçiş yapılacağı, early-binding ile, kod yazım aşamasında belirlenecektir. Bu bağlamda, kullanıcı, hangi fonksiyonların hangi “aspect”lerde olduğunu bir dil yardımıyla açıkça belirtecektir. Tabi bu bağlamda, “aspect”lerin birer bileşen olarak ele alınma zorunluluğu gelmektedir.

Geçiş sürecinden sorumlu olarak, yeni bir sanal makine (virtual machine) tanımlanmalıdır. Bu sanal makine, “aspect”e dayalı yazılım geliştirme metodolojisi ile geliştirilmiş programların çalıştırılabilir kodunu sürecek ve gerekli bellek ve disk erişim korumalarını ve hangi “aspect”in aktif hale getirileceğinin kontrolünü yaparak, gerektiği yerlerde koda müdahale ederek kod akışını gerekli “aspect”lere yönlendirecektir.

Çalışmamızın temeli, böyle bir sanal makinenin geliştirilerek, kodlayıcıya oldukça az bir iş bırakılması, hatta “aspect”lerin yapacağı işin tamamen koddan çıkarılması üzerine kuruludur.

Önerilen çözüm üç aşamalıdır:
1. Sanal makinenin tanımlanması,
2. Meta-dilin gerekli etiketlerine (tag) karar verilmesi, örnek bir “aspect” kataloğu geliştirilmesi ve
3. Gerçekleştirim ve entegrasyon.

6. Sonuç

Bu çalışmada önerilen, yazılımdaki “aspect”lerin kod seviyesinde belirlenmesi yerine, aspectlerin çalışma zamanında uygulamaya müdahil olmasını sağlamaktır. Bir sanal makine yardımıyla çalışma zamanındaki koda müdahale edilecek ve hangi kod parçacığının çalıştırılması gerektiğine de “aspect” kataloğu yardımıyla karar verilecektir. Gereksiz kaynak israfını önleme, özelleşmiş

bileşenlerin sağlamlığı, kod parçalarının birbirinden bağımsızlığı, yeniden kullanılabilirliği artırma, bu çalışmanın amaçları arasında yer almaktadır.

Eğer yapısal (structured) ya da nesneye dayalı (object-oriented) mimariler yatay mimari ise, "aspect"e dayalı mimariyi dikey mimari olarak düşünebiliriz. Bu yüzden, bu yaklaşımlar, birbirlerinden ayrı düşünülmemeli, birbirine yardımcı, birbirini destekleyici ve tamamlayıcı yaklaşımlar olarak ele alınmalıdır [6, 7]. Unutulmamalıdır ki, nesneye dayalı programlama dilleri bile, halen yapısal kod parçalarını içermektedir. Bu yüzden, "aspect"e dayalı programların da, nesneye dayalı ve hatta yapısal kod parçalarını içermesi kaçınılmaz ve gereklidir.

Kaynakça

1. P. Tarr, M. D'Hondt, L. Bergmans ve C. V. Lopes, Workshop on Aspects and Dimensions of Concern: Requirements on, Challenge Problems For, Advanced Separation of Concerns, ECOOP 2000 Workshop Proceedings, Springer Verlag, 2000.
2. Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C. V., Loingtier J. ve Irwin J., "Aspect"-Oriented Programming", Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finlandiya, 1997.
3. AOP "Aspect"-Oriented Programming Enables Better Code Encapsulation and Reuse, MSDN Magazine, Mart 2002. http://msdn.microsoft.com/msdnmag/issues/02/03/AOP/default.aspx.
4. M. Aksit ve B. Tekinerdogan, Solving the Modeling Problems of Object-Oriented Languages by Composing Multiple Aspects Using Composition Filters, AOP'98 workshop position paper, 1998.
5. Crnkovic I., "Component-based Software Engineering: Building Systems from Software Components", Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC'02), IEEE Computer Society Publications, 2002.
6. Rege K., "Design Patterns for Component-Oriented Software Development", EuroMicro Conference, Italya, 8-10 Eylül, 1999.
7. Hess R. A., "A Component-Oriented Environment For Systems Development", 14th Digital Avionics Systems Conference, 1995.