HLA Tabanlı Bileşenler ile Otomatik Uygulama Geliştirme

Cengiz TOĞAY

Bilgisayar Mühendisliği Bölümü Orta Doğu Teknik Üniversitesi

e-posta: ctogay@ceng.metu.edu.tr

Özet

Bu çalışma, belirli bir alanda birbirlerinin ihtiyacı olan fonksiyonaliteyi sağlayacak bileşenlerin bulunması ve beraber çalışmalarındaki engellerin giderilmesi için bir altyapı sunmaktadır. Bileşen teknolojileri için geliştirilen yaklaşımlar bu çalışmada uygulama alanı olarak seçilen High Level Architecture (HLA) ortamına uygulanmıştır. Benzetimlerin oluşturulması sırasında HLA federeleri arasında ortaya çıkacak uyumsuzlukların giderilmesine yönelik mekanizmalar tasarlanmıştır.

Abstract

This research proposes an infrastructure to inhibit obstacles for locating the components that require functionalities from each other and their collaboration, within a certain domain. The approaches are developed for general component technologies and applied to the High Level Architecture (HLA) medium. Mechanisms are designed to resolve the incompatibilities among the HLA federates.

1. Giriş

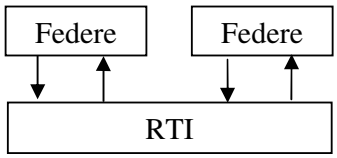
Geliştirme maliyeti ve süresine olan olumlu yansımasından dolayı Bileşenlere Dayalı Yaklaşımlar (Component Oriented Approaches) zamanla önem kazanmaktadır. Ancak uygun bileşenin bulunması ve sisteme dahil edilmesinde problemler yaşanmaktadır. Şu anda, geliştiricinin bileşen satmakta olan sitelerden isim ve bir kaç anahtar kelime ile bileşenleri bulması ve bileşenin üreticisi ile bağlantı kurup işine yarayıp yaramayacağını test etmesi gerekmektedir. Bileşen sayısının artması, geliştiricinin sistemi ile uyumlu bileşen bulma olasılığını arttırırken test etmesi gereken bileşen sayısını da arttırmaktadır. Bileşenlerin sadece isim, işletim sistemi gibi bilgiler ile tanımlanması otomatik bileşen bulma ve bağlama açısından yeterli değildir. Bileşenlerin arayüzleri ile sunduklarının yanısıra ihtiyaçlarının da bilinmesi gerekmektedir. Biz bileşen olarak sadece hizmet veren birimler değil, aynı zamanda bir yada daha fazla bileşenden hizmet alan birimleri de algılamaktayız. Çünkü amacımız seçilen ve uygun çalışması sağlanan bileşenler ile geliştiricinin ya hiç kod yazmadan ya da minimum kod ile uygulama kurabilmesini sağlıyabilmektir. Bu doğrultuda uygulama alanı olarak Yüksek Sevyeli mimari (High Level Architecture(HLA)) tabanlı benzetimler seçilmiştir. HLA tabanlı benzetimde, bileşenler yani "federe"ler birbirlerinin ihtiyaçlarını karşılayacak şekilde tasarlanmaktadırlar. HLA standartlarının IEEE tarafından belirlenmesi, üretici ve tüketici arasında ortak bir dilin oluşmasına katkıda bulunmaktadır. Böylece bileşenin sadece adı ve arayüz tanımlamaları gibi belirli bir kaç bilginin yanısıra kullanılan ve paylaşılan etkileşim ve özelliklerin sağlanması, yarı otomatik sistemlerin kurulmasına imkan vermektedir.

Bu çalışmada önerilen mekanizmalar ile son kullanıcının, bir alan modeli üzerinde yapacağı seçimler ışığında uygun bileşenlerin bulunması ve onlardan çalışan bir uygulamanın kurulması için gerekli altyapı sağlanmaktadır. Çalışmada, son kullanıcının mümkün olduğunca soyut düzeyde

kalması sağlanmaya çalışılmıştır. Bileşenler ile ilgili ayrıntılı bilgiler sadece mekanizma ve 'bileşen geliştirici' arasında yer almaktadır.
Bu bildiride kisaca calismanin temelini oluşturan sirasi ile HLA, Nitelik Modeli ve Alan Tanımlama Modelin(ATM)'den bahsedilmektedir. Calisma bölümünde ise mevcut ATM'nin otomatik birleşimindeki engellerin ortadan kaldırılması için yapılan eklentiler yer almakta ve bildiri sonuc ve kaynaklar ile son bulmaktadir.

2. Yüksek Sevyeli Mimari (HLA)

HLA, benzetimlerin birlikte çalışabilirliğini sağlamak amacıyla 1997 yılında Amerikan Savunma Bakanlığı tarafından geliştirilmiş bir standarttır. HLA'yi kurallar[1], arayüz tanımlamaları[2] ve Nesne Model Kalıbı (Object Model Template (OMT))[3,4] kavramları oluşturmaktadır. Kurallar, dağıtık benzetim (federasyon) ve onu oluşturan dağıtık bileşenlerin(fedele) birbirleri ile olan davranışlarını belirlemektedir. HLA'da yer alan federeleri biz bileşen olarak düşünmekteyiz. Arayüz tanımlamaları federeler ile Koşum Zamanı Altyapısı (Runtime Infrastructure (RTI)) arasındaki standartları tanımlamaktadır. Şekil 1'den de görüleceği üzere RTI, benzetimdeki tüm iletişimi, bir diğer ifade ile federe ve federasyonlar arası eşgüdümü sağlamakla görevlidir. Tüm iletişim RTI kutuphanesı üzerinden yapıldığı için federelerin hangi programlama dili ıle hazırlandıkları(Java, Microsoft VC++) önemini yitirmektedir.



Şekil 1. RTI üzerinden yapılan HLA bileşenlerinin haberleşmesi

Federelerin birbirleri ile doğrudan iletişim kurmaması gerekmektedir. OMT, federelerin ve federasyonların tanımlanması için bir çatı sağlamaktadır.
HLA'nın OMT'si federelerinin tekrar kullanılabilirliğini sağlayan en önemli kavramıdır. Federelerin tanımlanması için Benzetim Nesne Modeli (Simulation Object Model (SOM)) kullanılır. Bu modelde sadece ilgili federenin dış ortam ile paylaştığı sınıflar, özellikler, etkileşimler, veri yapilari, veri tipleri gibi bilgiler yer almaktadır. Federasyonların tanımlanması için ise Federasyon Nesne Modeli (Fedaration Object Model (FOM)) kullanılır. SOM'dan farklı olarak federasyonu oluşturan tüm federelerin kullandığı ortak bilgiler yer alır. Federasyona dahil olan her federe bu yapıları kullanmak zorundadır. SOM ve FOM'da yer alan tablolardan bazıları tablo 1'de yer almaktadır. SOM'da yer alan 13 tablodan federenin paylaştığı ya da ihtiyaç duyduğu tüm etkileşim ve özellikleri, veri tipleri ile birlikte elde edilebilmektedir.

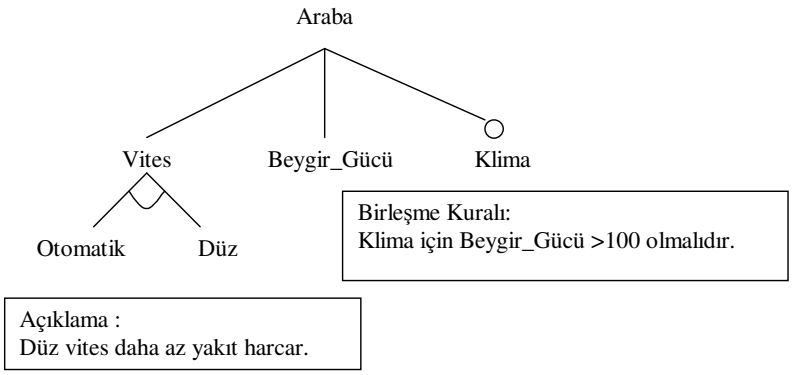
Tablo 1. OMT tabloları

Nesne Model Tanımlama Tablosu
Nesne Sınıf Yapı Tablosu
Etkileşim Sınıf Yapı Tablosu
Özellik Tablosu
Parametre Tablosu
Veri Tipleri Tablosu

Federeler arasında FOM'da belirtilen etkileşim ve özelliklerin kullanımı "yayın/kayıt"(publish/subscribe) yöntemi ile gerçekleşmektedir. Diğer federelere kullandırılmak istenen etkileşim yada özellik "yayın"lanır. Hizmeti almak isteyen federe ise ilgili etkileşim yada özelliğe "kayıt" olur. Federeler uygulamaya dahil olduklarında "yayın/kayıt" işlemleri ile sisteme ihtiyaç ve hizmetlerini sunarlar. Ancak bir özellik yada etkileşime "kayıt" olmak için onun daha önceden "yayın"lanmış olması gerekmektedir.

3. Nitelik Modeli

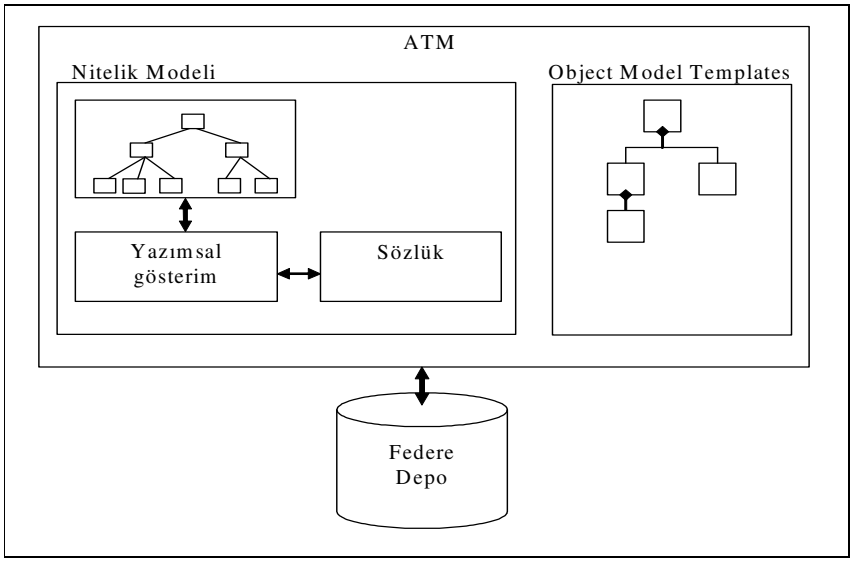
Nitelik Modeli, Feature Oriented Design Analysis(FODA)[5] ve onun gelişmiş versiyonu olan Feature Oriented Reuse Modeling (FORM)[6]'in alan hakkındaki bilgilerin saklanması amacıyla kullandıkları grafik modeldir. Şekil 2'de örnek olarak araba alanı için tanımlanmış bilgileri ifade eden bir nitelik diyagramı yer almaktadır. Diyagramda klima opsiyonel bir niteliktir ve seçilebilmesi için 100 beygirlik motor gücüne ihtiyaç duyulmaktadır. Vites ise Otomatik yada Düz olabilir.



Şekil 2. Örnek Feature Diyagram [5]

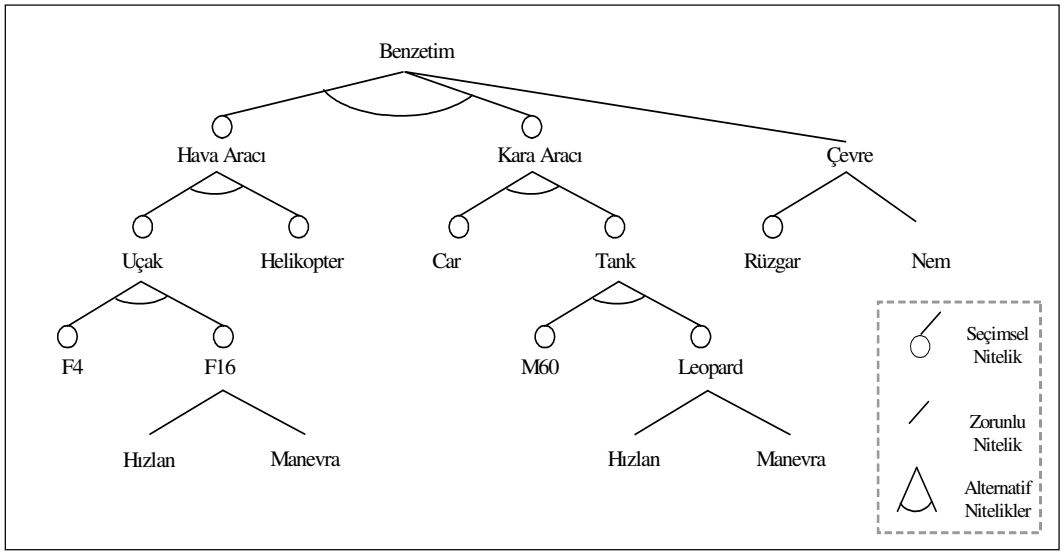
4. Alan Tanımlama Modeli

Bu çalışmada Alan Tanımlama Modeli (ATM) geliştirilerek, olgunlaşmış bir alandaki bilgilerin ve bu bilgilerin birbirleriyle olan ilişkilerinin tek bir çatı altında temsil edilmesi mümkün olmaktadır. ATM, Şekil 3'de görülüceği üzere Nitelik Modeli ve OMT'yi içermektedir. Nitelik Modeli[5,6] ile alan bilgileri grafiksel olarak ifade edilmektedir. İçerilen bilgiler hem grafiksel olarak hem de metin tabanlı yöntemlerle saklanmaktadır. Böylece anlaşılabilirlik ve otomasyon kabiliyetleri desteklenmektedir[7,8].



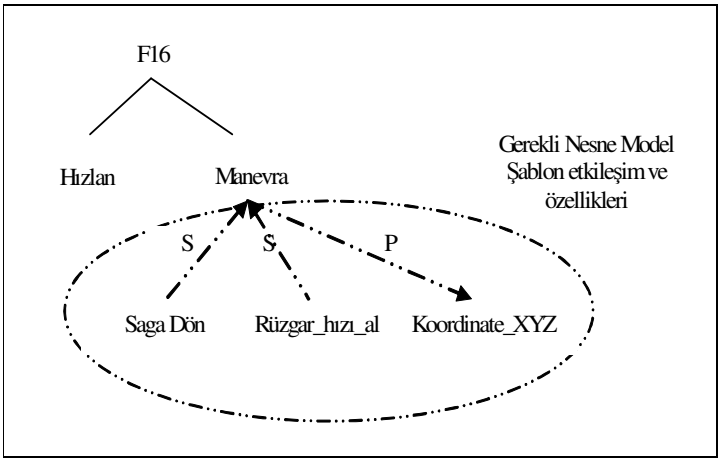
Şekil 3. Alan Tanımlama Modeli ve Federe Deposu

Bileşenlerin tanımlanmasında nitelik modelindeki alan bilgilerinin yanı sıra bileşenlerin bir anlamda arayüzlerini ifade eden OMT, tamamlayıcı bilgi olarak kullanılmaktadır. ATM'nin tasarlanmasındaki amaç son kullanıcının soyut düzeyde yapacağı seçimler sonucunda tanımlamasını yaptığı uygulamayı gerçekleştirecek bileşenlere ulaşmasıdır.



Şekil 4. Askeri araçlar için bir Nitelik Modeli örneği

Şekil 4'de ki örnek alan tanımında Nitelik Modeli ile son kullanıcının bir benzetim uygulaması yapması için gerekli alan bilgisi yer almaktadır. Son kullanıcı bu model üzerinden yapacağı seçimler ile uygulamayı tasarlamış olmaktadır. Seçimler sırasında grafikte görülmeyen ancak Nitelik Modelin yazımsal gösteriminde yer alan sınırlamalar, ilişkiler vb. kullanıcıyı doğru seçimler yapması konusunda yönlendirmektedir.



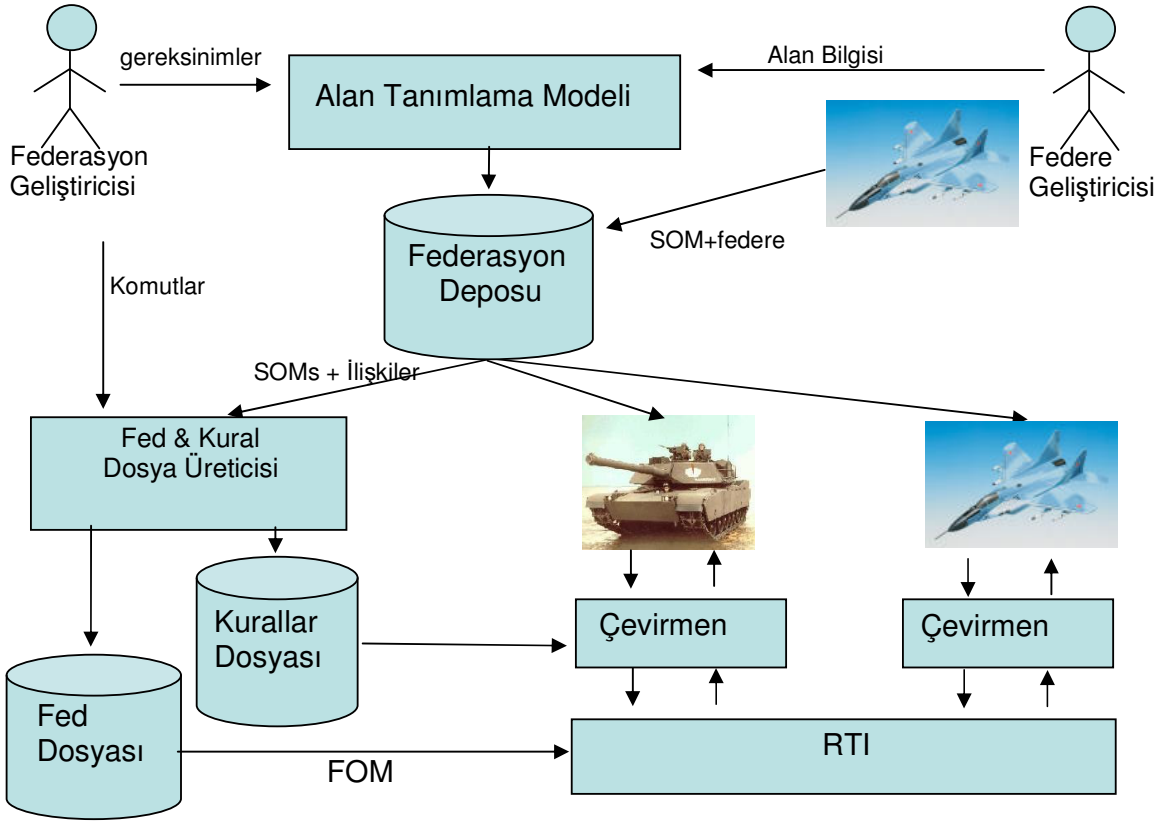
Şekil 5. OMT ve Nitelik Model ilişkisi

Şekil 5'de mevcut bileşenlerden birinin manevra yeteneğini gerçekleştirmek için ihtiyaç duyduğu etkileşim ve özelliklerin bazıları görülmektedir. Dolayısı ile F16'yı ve onun yetenek niteliği olan manevrayı seçmiş olan kullanıcı bileşenin ihtiyacı olan "Koordinat_XYZ" özelliğini sağlıyan bir başka bileşeni daha seçmesi için yönlendirilmektedir.

5. Çalışma

Daha önce yapmış olduğumuz çalışmalarda[7,8] ATM'nin içeriği oluşturulmuştu. ATM ile federasyon geliştiricisine belirli bir alanda nitelik modeli üzerinde yapacağı seçimler doğrultusunda ihtiyacı olacak federelerin belirlenmesi sağlanmıştır. Ancak, ATM federelerin bulunmasının yanı sıra bazı uyum problemlerinin aşılmasına yönelik bazı hizmetler ve imkanlar vermektedir. Eğer seçilen federelerin kaynak kodları var ise mevcut yaklaşımlar ile kodlarda yapılacak değişiklikler, saran(wrapper) kodlar ile uyum problemleri giderilebilir. Ancak gerek daha ucuz olacağı, gerek daha önceden alınmış olması gibi gerekçeler yüzünden kaynak kodları olmayan federelerin çalıştırılması ihtiyacı, çözülmesi gereken problemler doğurmaktadır. Bahsedilen uyum problemlerden bazıları şunlardır:

1- Kullanılacak olan federelerin SOM'larındaki farklı sınıf yapıları.
2- Sınıf, özellik, parametre ve etkileşimlerdeki isim farklılıkları (Örneğin, aynı etkileşim birinde getwind, diğerinde getw şeklinde tanımlanmış olabilir.)
3- Özellik ve parametrelerdeki tip farklılıkları (Örneğin, bir özellik birinde integer, diğerinde float tanımlanmış olabilir)
4- Etkileşimlerdeki parametre sayılarının farklılıkları (Örneğin, aynı işi yapan etkileşim diğerine göre farklı parametre sayısına sahip olabilir)
5- Bir federenin ihtiyaç duyduğu etkileşim, bir diğer federede birden fazla etkileşimin çağrılması şeklinde olabileceği gibi sonrasında da yapılması gereken işlemlere ihtiyaç olabilir. (Örneğin, get_kmh etkileşimine ihtiyaç duyan bir federenin, mil ile çalışan bir başka federe ile çalışması gerekiyor ise get_milh etkileşiminin ardından mili km'ye çevirilmesi ve sonucun get_kmh olarak döndürülmesi gerekir)
6- Bazı federelerin çalıştırılması için bazı özelliklerin başlangıçta kurulması gerekebilir.

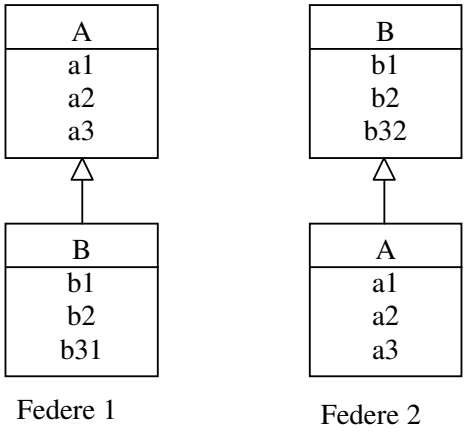


Şekil 6. Yapiya Genel Bakiş

Yapıya eklenen yeni moduller ile yukarıda bahsedilen uyum problemlerinin çözümüne yönelik ATM'den sağlanan bilgilere ve otomatik ürünlere ek olarak federasyon geliştiricisinin de katılımı sağlanmıştır. Şekil 6'da görüleceği üzere RTI ve HLA standartları korunmuştur. Daha önceki yapıya 3 yeni modül eklenmiştir. Bunlar Fed ve kural dosya üreticisi, Çevirmen, ve Kurallar dosyasıdır.

5.1 Fed ve Kural Dosya Üreticisi

Şekil 6'dan de görüleceği üzere bu modülün girdileri federasyon geliştiricisinden gelen komutlar, Federe deposundan gelen SOM bilgileri ile seçilen federeler arasındaki sınırlama ve ilişkileri gösteren bilgilerdir. Uyuma yönelik tüm çalışmalar bu modülde gerçekleşmektedir. Yukarıda bahsedilen uyum problemlerine yönelik çözüm önerileri şunlardır:



Şekil 7. Örnek OMT

1- OMT uyumsuzlukları

Şekil 7'de en basit uyumsuzluklardan biri gösterilmektedir. Federe1'de b1'e ulaşmak için A.B.b1 şeklinde tanımlanmışken Federe 2 için B.b1 şeklinde tanımlanmıştır. Dolayısı ile isim, tip gibi diğer uyumların sağlansa bile bu iki federenin beraber çalışmasına imkan yoktur. Bu nedenle Federe 1 ile Federe 2 arasında çevirme işlemine ihtiyaç duyulmaktadır. Bunun için Kural dosyasına çevirmen modülnün bu işi yapması için "A.B.b1 = B.b1"tir kuralı yazılır. Fed dosyasına ise ihtiyaca göre A.B.b1, B.b1 yada c gibi bir başka tanımlama yapılabilir(bu durumda kurala "=c" eklenmelidir). Önemli olan federelerin özellik yada etkileşim ihtiyaçlarında uygun işleyicinin(handler) sağlanabilmesidir.

2- İsim farklılıkları

Farklı firmalar farklı kelimeler ile aynı özellik veya etkileşimi ifade edebilirler bu neden ile OMT uyumsuzluklarında olduğu gibi "b31 = b32" şeklindeki kurallar ile çevirmen modulu bilgilendirilir. Kelimeler arasındaki uyumsuzlukların giderilmesi amacı ile ATM içerisinde yer alan sözlükten faydalanılır. Federasyon geliştiricisinin eşlemeyi yapması beklenmektedir. Bu kısmın otomatik olarak yapılması gelecek çalışma olarak bırakılmıştır.

3- Tip farklılıkları

SOM dosyaları federelerin özellik, etkileşim ve parametreler için kullandıkları tipleri göstermektedir. Dolayısı ile Federe 1'in b1 özelliği integer iken federe 2'nin b1'i string ise bu durumda 1 nolu çözümde uygulanan yönteme göre çevirme kuralı hazırlanır. Örneğin, Fed dosyasına A.B.b1 yazılmış ise, sadece Federe 2'nin b1 isteklerinde integer'dan string'e çevirme işlemi gerekecektir. Bu durumda kural "Federe2 cast integer B.b1"olmalıdır.

4- Parametre sayı farklılığı

Federelerin aynı amaçla kullandıkları etkileşimlerin parametre sayıları farklı olabilir. Ancak parametre sayılarının eksik yada fazla olması durumlarında çevirmenin federasyon geliştiricisinin belirteceği kurallara bağlı olarak işlem yapar. Örneğin, gereksiz parametrenin boş bırakılması, bir başka etkileşimin sonucu yada bir özellik ile doldurulması sağlanabilir.

5- Birden fazla etkileşim yada özellik ile diğer bir federenin ihtiyacını karşılama

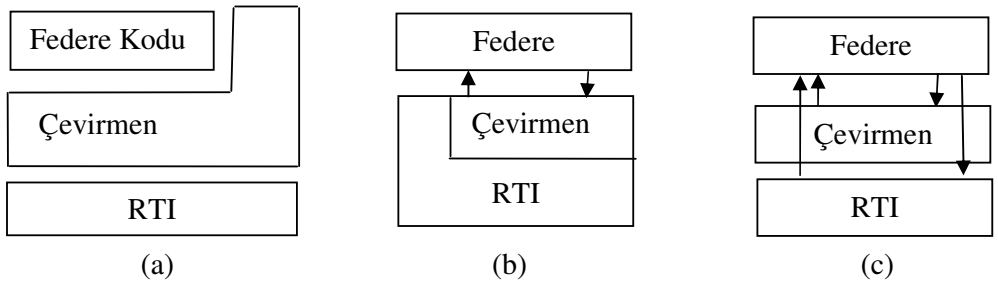
Bazen bir federenin ihtiyaç duyduğu bir özellik bir başka federe de birden fazla özellik ile tutuluyor olabilir. Örneğin, bir federe kompleks veri tipi olarak x, y ve z koordinatlarını

kullanırken diğeri ayrı ayrı kullanıyor olabilir. Dolayısı ile değişikliklerin x'de yapılan bir değişiklik diğerine bildirilirken kompleks veri tipine uygun şekilde bildirilir. Bununla ilgili kural yine kurallar dosyasına yazılır. Eğer federe1'in get_kmh etkileşimine ihtiyacı varsa ve federe2 get_milh etkileşimini yayınlıyor ise "federe1 get_kmh = (call get_kmh) * 1.609344" şeklinde kural yazilabilir.

6- İlk değer atama
Uyum için gerekli başlangıç değerlerinin atanması gerekebilir. Bu değerler de yine federasyon geliştiricisi tarafından belirtilir ve Kurallar dosyasına yazılır.

5.2 Çevirmen

Çevirmen modulu fedarasyon geliştiricisinin istek ve önerileri doğrultusunda hazırlanan kuralları gerçekleştiren moduldur. Bu modülü 3 şekilde gerçekleştirmek mümkündür.



Şekil 8. Yapının Gerçekleştirim Alternatifleri

Şekil 8.a'da çevirmen saran(wrapper) kod olarak tasarlanmıştır. Federe kodu orjinal RTI sınıfları yerine RTI'dan türemiş olan çevirmen sınıflarını kullanmaktadır. Bu yaklaşımda federe kodlarına ihtiyaç bulunmaktadır. Şekil 8.b, RTI üreticilerinin Çevirmen modülünü RTI'ya eklemesi şeklindeki gerçekleştirimidir. Şekil 7.c'de, çevirmen bazı özellik ve etkileşimler için hiç bir şey yapmadan RTI'ya erişime izin verirken, kurallar çerçevesinde bazılarında işlem yapmaktadır. Bunun için orjinal DLL dosyalarının hazırlanacak yeni DLL ile bazı RTI fonksiyonları direk orjinal DLL'den çağrılırken bazıları ise kurallar çerçevesindeki işlemlerin ardından orjinal RTI'dan çağrılması şeklinde gerçekleştirilir. Biz 3. yaklaşımın federe ve RTI kodunda değişiklik içermemesi nedeni ile daha esnek bulmaktayız.

5.3 Kurallar Dosyası

Bölüm 4.1 de bahsedilen çözüm önerilerine(kurallara) uygun olarak Kurallar Dosyası ATM den sağlanan bilgiler ışığında doldurulmaktadir. Problemin türüne bağlı olarak bazı kısımları otomatik olarak doldurulurken, bazı kısımları ise federasyon geliştiricisi tarafından doldurulmaktadir. Yapıda kullanılacak olan araçlar ile federasyon geliştirici yönlendirilecektir.

6. Sonuç

Önerilen yapı ile bileşenlerden otomatik uygulama geliştirmeye bir adım daha yaklaşılmıştır. Uygulama alanı olarak HLA`nın seçilmesinin nedeni sağlamış olduğu standartlar ile bileşenlerin kendilerini ifade etmelerinde arayüzlere oranla daha fazla bilgi ihtiva etmesidir. Bileşenlerin ortam ile paylaştıkları özellik ve metodların önceden bilinmesi otomatik birleştirmede ilişkili bileşenlerin bulunmasını sağlar. Tüm iletişimin RTI tarafından sağlanıyor olması farklı programlama dillerinde ve işletim sistemlerinde hazirlanmış olan federelerin beraber çalışması mümkün olmaktadır. ATM son kullanıcının, bileşenlerin uyum problemlerı haricinde, soyut düzeyde kalmasını sağlar. ATM aynı zamanda bileşen geliştiricisi için kaynak olmaktadır. Üretici ATM'den alacağı OMT'ler ile sistemdeki diğer bileşenler ile uyumlu bileşen üretmesi mümkün olmaktadır. Yapida yer alan Çevırmen Modulu federeler arsında birleşime yönelik uyum problemlerinin çözülmesinde kullanılmaktadır.

7. Kaynaklar

[1] Department of Defense (DoD),1996. High Level Architecture Rules, Verison 1.0, dated 15 August 1996.

[2] Department of Defense (DoD),1997. High Level Architecture Interface Specification, Verison 1.2

[3] Department of Defense (DoD),1996. High Level Architecture Object Model Template, Verison 1.1, dated 12 March 1997.

[4] IEEE Standart for Modeling and Simulation (M&S) High Level Architecture (HLA)- Object Model Template (OMT) Specification, IEEE std 1516.2-2000, 2000.

[5] Kang, K. C., Cohen, S. G., Hess, J. A., Nowak, W. E.,Peterson, A.S.,"Feauture Oriented Domain Analysis(FODA) Feasiblity Study", CMU/SEI-90-TR-21, ADA 235785, Software Engineering Institute, Carnegie Mellon Univesty, Pittsburg, PA, 1990.

[6] Kang, K. ,Kim, S., Lee, J., Kim, K., Shin, E., Huh, M., "FORM : A Feature Oriented Reuse Method with Domain-Specific Reference Architectures", Annals of Software Engineering, Volume 5, J. C. Baltzer AG Science Publishers, Red Bank, NJ, USA,1998, pp. 143-168.

[7] Togay, C., Dogru, A., "Federasyonların HLA Tabanlı Benzetimlere Tümleştirilme Otomasyonu için bir Mekanizma", 1. Ulusal Savunma Uygulamaları Modelleme Simülasyon Konferansı, 2005.

[8] Togay, C., Dogru, A., "Infrastructure Design for HLA Based Automated Federation Development", Integrated Design and Process Technology,2005