T.C.
MALTEPE ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

SERVİS ODAKLI MİMARİLERDE WEB SERVİSLERİN VERSİYONLANMASI İÇİN BİR TASARIM YAKLAŞIMI

FATİH BALIKCI

Yüksek Lisans Tezi

Tez Danışmanı
Yrd. Doç. Dr. Fatih YÜCALAR

İSTANBUL – 2014

T.C.
MALTEPE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

SERVİS ODAKLI MİMARİLERDE WEB SERVİSLERİN VERSİYONLANMASI İÇİN BİR TASARIM YAKLAŞIMI

YÜKSEK LİSANS TEZİ

FATİH BALIKCI

Tez Danışmanı
Yrd. Doç. Dr. Fatih YÜCALAR

İSTANBUL – 2014

Bu tez çalışması, Maltepe Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 12/02/2014 tarih ve 2014/03 sayılı kararıyla oluşturulan jüri tarafından ***Bilgisayar Mühendisliği Yüksek Lisans Tezi*** olarak kabul edilmiştir.

JÜRİ

Yrd.Doç.Dr. Fatih YÜCALAR
Danışman

Yrd.Doç.Dr. Turgay BİLGİN
Üye

Yrd.Doç.Dr. Erdal GÜVENOĞLU
Üye

ÖZET

Yüksek Lisans Tezi, Servis Odaklı Mimarilerde Web Servislerin Versiyonlanması İçin Bir Tasarım Yaklaşımı, Maltepe Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı.

Versiyon, geliştirilen veya kullanılan yazılımlarda değişikliğin takip edilmesi ve yönetilebilmesi adına yazılımların sahip olduğu tekil sürüm numaralarını modelleyen mimaridir. Bu tez çalışması web servislerde değişikliğin takip edilmesi ve yönetilebilmesi adına yazılımların gerçek hayatta nasıl versiyonlandığına yönelik bir tasarım yaklaşımı sunmaktadır.

Versiyonlama, kendine özgü kuralları olan, hayatımızın farklı alanlarında kullanılan bir yaklaşımdır. Web servislerin iyi versiyonlanabilmesi için detaylı bir analize ve modellemeye ihtiyaç duyulur.

Bu tezin amacı, versiyonlamayı oluşturan teknolojilerin, servis sözleşmelerine ilişkin versiyon revizyon kontrolü ile ilgili bazı düşüncelerin, terminolojilerin ve stratejilerin versiyonlama içindeki önemini araştırmaktır. Tez ile birlikte sunulan örnek senaryo ile servis odaklı mimarilerde servis versiyonlanarak, bu mimarideki teknoloji ve yaklaşımın nasıl hayata geçirildiği gösterilmeye çalışılmıştır.

Bu tez 2014 yılında tamamlanmıştır ve 93 sayfadan oluşmaktadır.

Anahtar Kelimeler: Web Servisler, Web Servis Tanımlama Dili, Versiyonlama, Servis Odaklı Mimari

ABSTRACT

Master Thesis, Versioning of Web Services in Service Oriented Architecture: A Design Approach, T.C. Maltepe University, Institute of Natural Sciences, Department of Computer Engineering.

Versioning is an architecture modeling singular version number of software to track and manage software developed or used. In this study a design approach is intended for how software are versioned in real world. Management of changes in web service is also presented.

Versioning is an approach having its own rules, being used in different areas of our lives. A detailed analyze and modeling is needed for good versioning of web services.

The purpose of this thesis is to investigate the importance of technologies that create versioning, some thoughts about version revision control related to service contracts, and related terminologies and strategies. In this thesis a sample scenario presented to demonstrate how such a technology and approach is implemented by versioning the service in service-oriented architectures.

This thesis is completed in 2014 and consists of 93 pages.

Keywords: Web Service, Web Service Description Language, Versioning, Service Oriented Architecture

TEŞEKKÜR

Tez konusunu seçmemde beni yönlendiren, tez süreci boyunca destek ve yardımlarını esirgemeyen, değerli bilgilerinden istifade ettiğim danışman hocam Yrd. Doç. Dr. Fatih YÜCALAR'a, tez sürecinde bana gösterdiği olağanüstü anlayış ve yardımları için eşim Elif BALIKCI'ya, maddi ve manevi desteğini benden hiçbir zaman esirgemeyen çok değerli aileme ve çalışmalarım sırasında emeği geçen herkese teşekkürlerimi sunarım.

İÇİNDEKİLER

KISALTMALAR

Kısaltma	İngilizcesi	Türkçesi
HTTP	Hyper Text Transfer Protocol	Hiper Metin Aktarım İletişim Kuralı
MEP	Message Exchange Patterns	Mesaj Değişim Deseni
QR	Quick Response	Hızlı Yanıt
SOA	Service Oriented Architecture	Servis Odaklı Mimari
SOAP	Simple Object Access Protocol	Basit Nesne Erişimi Protokolü
UPA	Unique Particle Attribution	Benzersiz Partikül Öznitelik
WS	Web Service	Web Servis
WSDL	Web Service Description Language	Web Servis Tanımlama Dili
XML	eXtensible Markup Language	Genisletilebilir Bağlantılı

ŞEKİLLER

TABLOLAR

1. GİRİŞ

Servis Odaklı Mimari'nin temel fikri servistir. Servis, servis anlaşmasıyla yapılabilen iş işlevselliğinin ayrık bir birimi olarak tanımlanır. Servis anlaşması, servis sunucusu ve servis kullanıcısı arasındaki etkileşimi göstermektedir. Servis arayüzü servis operasyonlarını belirtir. Bunlar servisin ne yaptığı, servisin operasyona giren ve çıkan parametreleri, servisin özellikleri ve sağladığı protokolleridir.

Versiyon, geliştirilen veya kullanılan yazılımlarda değişikliğin takip edilmesi ve yönetilebilmesi adına yazılımların tekil sürüm numaralarına sahip olmasıdır. Geliştirilen yazılımların herbiri tekil sürüm numaralarına sahip olmalıdır. Bir uygulamanın yeni sürümü her zaman için bir önceki sürümden daha üst sürüm numarasına sahip olmalıdır [1].

Bu tez çalışmasında geliştirilen veya kullanılan web servislerde değişikliğin takip edilmesi ve yönetilebilmesi adına yazılımların gerçek hayatta nasıl versiyonlandığına yönelik bir tasarım yaklaşımının anlatılması hedeflenmektedir.

Tezin ikinci bölümünde versiyon revizyon kontrol temellerinden, temel kavramlardan, majör ve minör versiyon ilişkisinden, üçüncü bölümde WSDL tanımlama versiyonundan, sürüm tanımlamaları ve sürüm oluşturma adımlarından bahsedilmektedir. Dördüncü bölüm de şemaların versiyonlanması, beşinci bölümünde ise gelişmiş versiyonlama stratejileri ele alınmıştır. Tezin altıncı bölümünde web servis versiyonlaması için tasarlanan uygulamaya ve son bölümde ise elde edilen sonuçlara ve önerilere yer verilmiştir.

2. VERSİYON REVİZYON KONTROL TEMELLERİ

Yeni bir web servis sözleşmesi yayınlandıktan sonra, kullanıcı programları bu servise bağımlı olarak oluşturmaya başlar. İlerleyen zamanlarda web servis sözleşmesinde değişiklik ortaya çıkması durumunda aşağıdaki soruların dikkate alınması gerekmektedir.

- Değişiklikler mevcut servis sözleşmesini olumsuz yönde etkileyecek mi?
- Yapılan değişiklikler uygulayıcıları ve ilgisi olan kullanıcıları nasıl etkileyecek?

Bu konular, versiyon revizyon kontrolüne gerek olduğu sonucunu ortaya çıkarır. Bir Servis Odaklı Mimari (Service Oriented Architecture – SOA) projesine versiyon revizyon kontrol kavramı getirmek istenildiği zaman, bir çok soru gündeme gelecektir. Bu sorular;

- Yeni bir servis sözleşmesine ait versiyon nelerden oluşacaktır?
- Majör ve minör versiyon arasındaki farklılık ne olacaktır?
- Versiyon numaralandırma sistemi neleri belirtir?
- Yeni versiyon sözleşmesi, eski versiyon sözleşmesini kullanan mevcut kullanıcılar tarafından kullanılabilecek mi?
- Kullanıcıların versiyon değişikliklerinden minimum etkilenmesi için mevcut sözleşme değişikliklerinde ilavelere ilişkin ideal yöntem ne olmalıdır?
- Eski ve yeni sözleşmeleri aynı anda bulundurma ihtiyacımız olacak mı?

Bu tür soruların cevapları ve genel versiyon revizyon kontrol problemlerinin çözüm alternatifleri, ilerleyen kısımlarda daha detaylı anlatılacaktır. Bu kısımda, servis sözleşmelerine ilişkin versiyon revizyon kontrolü ile ilgili bazı temel düşünceler, terminolojiler ve stratejiler anlatılacaktır.

2.1 Temel Kavram ve Terminolojiler

Bu başlık altında web servislerin versiyonlanması ile ilgili temel kavramlar ve terminolojiler alt başlıklar halinde incelenmiştir.

2.1.1 Web Servisler ve WSDL

Web servis internet üzerinden erişilebilen, her türlü platform ile bağlantı kurabileceğimiz, geriye sonuç döndüren işlevselliklere ve hizmetlere sahip olan bir uygulama parçasıdır. Web servisler, Genişletilebilir İşaretleme Dili (Extensible Markup Language – XML) temeline dayanır, yani XML tabanlı bir teknolojidir [14].

Web servislerin kullanım alanlarına bir örnek vermek gerekirse; Merkez Bankası her gün döviz kurlarını kendi sistemi üzerinde belirler ve bu "kur bilgisi" üzerinde işlem yapan kurumlar merkez bankasının Web Servisleri aracılığıyla gün içerisinde değişen kur bilgilerini kendi sistemlerine çekerler ve bu şekilde merkezi ve güncel bir yapının oluşması sağlanmış olur.

Web Servis Tanımlama Dili (Web Service Description Language – WSDL) ise web servisleri ve onlara nasıl erişilebildiğinin tanımlandığı XML tabanlı bir dildir. WSDL, web servisi tanımlayan bir XML belgedir. WSDL ile web servisin metotlarının yeri ve işlevleri belirtilir [18].

WSDL kullanmanın avantajları sıralayacak olursak [18]:

- Haberleşecek sistemlerin platformlarının uyumlu olması gerekmez.
- Java ile geliştirilmiş ve UNIX sistem üzerinde çalışan bir uygulama ile .NET ile geliştirilmiş ve Windows işletim sistemi üzerinde çalışan bir uygulama, birbirlerinin çalışma ortamlarından bağımsız olarak, XML iletişim standartları aracılığıyla iletişim kurabilir.

- Bir uygulamanın fonksiyonlarını, internet üzerinden çok geniş bir kullanıcı kitlesine açmak mümkündür.
- HTTP (Hyper Text Transfer Protocol) üzerinden çalıştırıldığı için, sistemleri dışarıya güvenlikten pek fazla ödün vermeden açmak mümkün olur. Firewall'lar, uygulamalar arasında veri paylaşmak istediğimizde kullanıcılara engel olmaz.

2.1.2 Versiyon ve Kapsamı

Geliştirilen veya kullanılan yazılımlarda değişikliğin takip edilmesi ve yönetilebilmesi adına yazılımların tekil sürüm numaralarına sahip olmasına versiyon denir. Geliştirilen yazılımların herbiri tekil sürüm numaralarına sahip olmalıdır. Bir uygulamanın yeni sürümü her zaman için bir önceki sürümden daha üst sürüm numarasına sahip olmalıdır [1].

Majör versiyonlamada sürüm numarası projedeki önemli değişiklikler sonrasında artmaktadır. Majör versiyonlama uyumsuz bir değişiklik olup büyük(majör) versiyon numarasını artırır ve yeni bir WSDL tanım hedef adının oluşmasına sebep olur. Projelerde yaşanabilecek önemli mimari değişiklikler, programlama dili değişikliği, veritabanı değişikliği ve teknolojik değişiklikler majör versiyonlamaya örnek verilebilir [3].

Minör versiyon yazılımın modüllerindeki değişlikler, iş biriminden gelen taleplerin gruplanması ve benzeri sebeplerle ortaya çıkan uyumlu bir değişikliktir. Küçük(minör) versiyon numarasını artırır ve WSDL tanım hedef adını değiştirmez [3].

Web servis sözleşmesi örnek olarak aşağıdakilerden oluşabilir [12]:

- WSDL tanımlamaları
- XML taslak (şema) tanımlamaları
- WS-İlke (Policy) tanımlamaları

Bunlara ek olarak, tanımlanan bu dokümanların her biri diğer web servis sözleşmeleri ile ortak kullanılabilir.

- Bir merkezi XML taslak tanımlaması, genellikle çoklu WSDL tanımlamaları tarafından kullanılacaktır [12].
- Bir merkezi WS-İlke (Policy) tanımlaması, genellikle çoklu WSDL tanımlamaları tarafından uygulanacaktır [12].

Bir web servisin sözleşmesini oluşturan en önemli teknik arayüz, WSDL tanımlamaların özet açıklamalarıdır. Bu, bir web servis sözleşmesinin çekirdeğini temsil eder. Daha sonra genişletilir ve taslak tanımlamalarla, ilke tanımlamalarıyla ve bir/birçok somut WSDL açıklamalarıyla detaylandırılır [12].

Yeni bir web servis sözleşmesi oluşturmaya gerek duyduğumuzda, somut WSDL tanımlamalarında veya somut WSDL tanımlamaları ile ilgili bir sözleşme belgesinde değişiklikler olacağını kabul ederiz. Bir WSDL'ın değişik yapılarının nasıl uyumlu hale getirilebileceği ilerleyen kısımda detaylandırılacaktır.

Web servis sözleşmesinde değişikliğe uğrayabilecek kısım, genellikle özet tanımlamaların mesaj açıklamalarını kapsayan XML taslaklarıdır. İlerleyen kısımlarda, mesajlar için temel taslak uygulamaların nasıl değiştirilebileceği ve geliştirilebileceği açıklanmaktadır.

2.1.3 Versiyonlamada Hassas ve Kaba Sınırlar

Versiyon değişiklikleri çoğunlukla sınır değerlerinin ve detaylarının artış veya azalımı ile ilgilidir.

Şekil 2.1'de verilen örnekte kırmızı ve kalın olarak belirtilen kısımlara dikkat edelim:

```
<xsd:element name="LineItem" type="LineItemType"/>
<xsd:complexType name="LineItemType">
  <xsd:sequence>
    <xsd:element name="productID" type="xsd:string"/>
    <xsd:element name="productName" type="xsd:string"/>
    <xsd:any minOccurs="0" maxOccurs="unbounded"
      namespace="##any" processContents="lax"/>
  </xsd:sequence>
  <xsd:anyAttribute namespace="##any"/>
</xsd:complexType>
```

Şekil 2.1 Versiyonlamada hassas ve kaba sınırlar [5]

Kalın metin ile gösterildiği gibi, özel isimler ve data türleri vardır. Bunlar, mesaj tanımının bir bölümünü hassas sınırlama derecesi olarak temsil eder. Programa ait bütün mesajların (bu yapıda oluşturulacak gerçek XML dokümanlar) geçerli olabilmesi için bu sınırlamalarla uyumlu olmaları zorunludur (bu sebepten bu tanımlamalar, mutlak olarak kabul edilen "minimum" sınırlamalardır).

Kırmızı metin kompleks tiplerin içerdiği unsurları ve özel sembol karakterlerini gösterir. Bunlar mesaj tanımının parçalarını son derece kaba kısıtlama detayları ile temsil eder. Mesaj'ın komple yapısına uyumlu olmalarına gerek yoktur.

"Hassas Sınırlama" ve "Kaba Sınırlama" terimlerinin kullanımı sübjektiftir, kişilerin bakış açısına göre değişebilir. Bir sözleşmede yer alan hassas kısıtlama derecesi diğer bir sözleşmede aynı özellikte olmayabilir. Önemli olan bir mesaj tanımlamasının parçalarını veya farklı mesaj tanımlamalarını birbiri ile karşılaştırırken bu terimlerin nasıl uygulanacağını anlamaktır.

2.2 Versiyon Revizyonu ve Uyumluluk

Yeni bir servis sözleşme versiyonu geliştirildiğinde ve yayımlandığında mevcut kurulmuş veya kurulacak sistemlere etkisinin ne olacağını bilmek önemlidir. Bu etkinin ölçümü, yeni sözleşme versiyonunun, eski versiyon ve ilgili diğer programlar ile direkt uyumlu olup olmadığıdır.

2.2.1 Geriye Dönük Uyumluluk

Bir web servis sözleşmesinin yeni bir versiyonu, eski versiyon'daki kullanıcı programlarını destekleyip, çalışmalarını problemsiz devam ettirirse; "eski ile uyumlu (backwards-compatible)" olarak adlandırılır. Tasarım bakış açısı olarak, yeni sözleşmede yapılan değişiklikler mevcut kullanılan programları olumsuz yönde etkilemeyecektir [15].

Mevcut bir WSDL tanımlamasında yeni bir işlemin ilavesi geriye doğru uyumlu değişikliğe örnek olacaktır:

```
<definitions name="Purchase Order" targetNamespace=
  "http://actioncon.com/contract/po"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po"
  xmlns:po="http://actioncon.com/schema/po">
  ...
  <portType name="ptPurchaseOrder">
    <operation name="opSubmitOrder">
      <input message="tns:msgSubmitOrderRequest"/>
      <output message="tns:msgSubmitOrderResponse"/>
    </operation>
    <operation name="opCheckOrderStatus">
      <input message="tns:msgCheckOrderRequest"/>
      <output message="tns:msgCheckOrderResponse"/>
    </operation>
    <operation name="opChangeOrder">
      <input message="tns:msgChangeOrderRequest"/>
      <output message="tns:msgChangeOrderResponse"/>
    </operation>
```

```
    <operation name="opCancelOrder">
      <input message="tns:msgCancelOrderRequest"/>
      <output message="tns:msgCancelOrderResponse"/>
    </operation>
    <operation name="opGetOrder">
      <input message="tns:msgGetOrderRequest"/>
      <output message="tns:msgGetOrderResponse"/>
    </operation>
  </portType>
</definitions>
```

Şekil 2.2 Yeni işlemin ilavesi ile geriye dönük uyumluluk [8]

Şekil 2.2'de yeni bir işlem ekleyerek, sözleşmenin yeni bir versiyonunu oluşturuyoruz. Ancak, bu değişiklik geriye dönük uyumludur ve mevcut kullanıcıları olumsuz etkilemeyecektir. Geriye dönük uyumlu bir mesaj tanımlama taslağında yapılan örnek bir değişiklik opsiyonel bir öğe ilavesidir:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="available" type="xsd:boolean"
        minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Şekil 2.3 Opsiyonel öğe ilavesi ile geriye dönük uyumluluk [8]

Şekil 2.3'te Sipariş Mektup servisinin XML taslak tanımına ilişkin basitleştirilmiş versiyon kullanılmaktadır. Opsiyonel olarak geçerli olan öğe, kompleks tip olan SatırKayıtTip (LineItemType)'ine eklenir. Bunun mevcut kullanıcılar aleyhine bir etkisi yoktur. Çünkü, mesajlarda bu öğe'nin olması istenmez. Opsiyonel olarak öğe sağlayabilecek bu taslak ile çalışacak yeni kullanıcılar veya kullanıcı programları yeniden tasarlanır.

Önceki örnekte mevcut öğelerin gereklilikten opsiyonele (minoccurs="0" ekleyerek) dönüştürülmesi geriye dönük uyumluluk olarak düşünülebilir. Gelecek versiyon tasarımını web servis sözleşmesinde nasıl değiştirebileceğimizi kontrol edebiliyorsak, geriye dönük uyumluluğa genelde ulaşılabiliriz. Ancak, mecburi

(kanun veya yönetmelikler gereği yapılan değişiklikler gibi) yaptığımız değişiklikler geriye dönük uyumluluğun sık sık bozulmasına sebep olabilir.

2.2.2 İleriye Dönük Uyumluluk

Bir web servis sözleşmesi gelecekte kullanılacak programları destekleyebilecek şekilde düzenlenirse, bu sözleşme ileriye dönük uyumluğu içerir. Bu web servis sözleşmesi zaman içinde kullanıcı programlarının gelişimini kapsar [15].

Şekil 2.4'de en yaygın ileriye dönük uyumluluk yöntemi ifade edilmektedir:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:any namespace="##any" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##any"/>
  </xsd:complexType>
</xsd:schema>
```

Şekil 2.4 İleriye dönük uyumluluk [8]

Bir mesaj tanımlaması içinde ileriye dönük uyumluluğu desteklemek için genelde XML taslak sembollerine gerek vardır.

Şekil 2.4'de web servis sözleşmesi tarafından kabul edilecek değerler bilinmeyen "xsd:any" ve "xsd:anyAttribute" tipleri ile ifade edilir. Diğer bir deyişle, taslak gelecekte öngörülmeyen değişiklikleri de kapsayacak şekilde tasarlanır.

İleriye dönük uyumluluğu destekleyen seçenekler WSDL tanımlamalarında sınırlıdır.

İleriye dönük uyumluluk, gelecekteki ihtiyaçları tam karşılayacak demek değildir. İleriye dönük uyumlu bir servis sözleşmesinde olan bir mesaj içeriği bütün süreci

kapsamayabilir. Sözleşme, tasarlandığı zamana göre daha geniş kapsamlı olarak yeniden tasarlanabilir.

2.2.3 Uyumlu Değişiklikler

Bir servis sözleşmesinde bir değişiklik yaptığımızda mevcut kullanıcıları olumsuz yönde etkilemiyorsa, bu değişiklik uyumlu değişiklik olarak kabul edilir.

Uyumlu değişikliğe bir örnek olarak Şekil 2.5'te gösterildiği gibi, "minOccurs" unsuru "1" den "0" a değiştirildiğinde gerekli bir eleman opsiyonel konuma gelir.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"
        minOccurs="0"/>
      <xsd:element name="available" type="xsd:boolean"
        minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Şekil 2.5 Uyumlu değişiklikler [8]

"minOccurs" unsurunun varsayılan değeri "1" dir. Bu unsur önceden "productName" de olmadığı için, gerekli bir eleman olarak kabul edildi. "minOccurs=0" yapıldığında, bu unsur opsiyonel bir elemana dönüşür ve sonuç olarak uyumlu değişiklik ortaya çıkmış olur.

Web servise gönderilen elemanın değerinin değişikliği mevcut kullanıcı programlarını olumsuz etkilemeyeceği gibi gelecekteki kullanıcı programlarını da olumsuz etkilemeyecektir.

Opsiyonel öğe ilavesi eklediğimiz uyumlu değişikliğin bir diğeri Şekil 2.3'te verilmişti. Tipi komple yeni eleman olarak tanımlasakta, opsiyonel olduğu için uyumlu değişiklik olarak kabul edilir.

2.2.4 Uyumsuz Değişiklikler

Yapılan değişiklikten sonra sözleşme eğer kullanıcı ile uyumlu değilse, uyumsuz bir değişiklik yapılmış olduğu kabul edilir. Bu tip değişiklikler mevcut sözleşmeyi bitirebilir ve dolayısıyla versiyon revizyonu yapılmak istendiğinde zorunlu yükümlülükler ortaya çıkarabilir.

Uyumsuz değişiklikler geriye dönük uyumluluğun olmadığını gösterir. İleriye dönük uyumluluğu etkileyecek uyumsuz değişiklik ise ileriye dönük uyumsuz değişiklik olarak nitelendirilir.

Şekil 2.6'da görüldüğü gibi, eğer "minOccurs" teriminin değerini "0" dan yüksek bir değere atanırsa uyumsuz bir değişiklik yapmış oluruz.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"
        minOccurs="3"/>
      <xsd:element name="available" type="xsd:boolean"
        minOccurs="3"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Şekil 2.6 Uyumsuz değişiklikler [8]

"minOccurs" terimine, varsayılan değerden daha yüksek bir değere atanırsa, bu durum otomatik olarak uygumsuz bir değişikliktir.

Önceden opsiyonel olan bir terim şimdi gerekli duruma gelir. Bu değişiklikler yeni duruma göre uyumlu tasarlanmamışsa kullanıcı programlarını kesinlikle etkileyecektir. Çünkü, yeni bir unsurun eklenmesi sözleşmede zorunlu sınırlamalar getirir.

Uyumsuz değişiklikler, web servis sözleşme versiyon revizyonunda karşılaşılan en büyük sorunlardır.

2.3 Versiyon Tanımlayıcılar

Web servis sözleşme tasarımı ile ilgili en temel tasarım tiplerinden biri "Versiyon Tanımlama" modelidir. Versiyon numaraları sadece sözleşmede değil, mesaj tanımlamalarının temelini oluşturan taslak versiyonların sağ alt köşesinde de açıkça belirtilmelidir.

Etkili bir versiyon (sürüm) oluşturma stratejisinin ilk adımı yaygın kullanımlar için karar vermektir. Bu versiyonlar web servis sözleşmesi içinde belirlenirler ve gösterilirler [6].

Versiyonlar neredeyse her zaman sürüm numaraları ile bilinir. En yaygın biçim aşağıda gösterildiği gibi, bir onlu rakam (decimal) ve noktadan sonra gelen diğer bir onlu rakamdır [6]:

version="2.0"

Bazen aşağıda olduğu gibi, versiyon numarasını daha detaylı göstermek için ekstra onlu rakamlarda eklenebilir:

version="2.0.1.1"

Bu numaraların genel anlamı yapılan değişikliklerin boyutu ve önemidir. İlk onlu rakamın artışı yazılım da önemli bir versiyon değişikliğini (veya upgrade - versiyon

yükseltme) gösterir. Buna karşı, noktadan sonraki onlu sayının artışı genellikle önemsiz ve küçük versiyon değişikliklerini temsil eder.

Uyumluluk bakış açısına göre bu numaralara ek anlamlar verilebilir. Özellikle, aşağıda belirtilen durumlar endüstride yaşanmıştır [10]:

- Yapılan versiyon değişikliği önemsiz ve küçük ise diğer versiyonlar ile geriye dönük uyumlu bir değişikliktir. Örnek verilecek olursak; 5.2 versiyon numarasına sahip olan bir program, 5.0 ve 5.1 programları ile geriye dönük tam uyumludur.
- Yapılan versiyon değişikliği önemli ve büyük ise diğer versiyonlar ile geriye dönük uyumluluk bozulmuş olur. Örnek verilecek olursak; 5.0 versiyon numaralı bir program, 4.0 versiyon numaralı bir program ile geriye dönük uyumlu değildir.

Büyük ve küçük versiyon değişikliklerini gösteren numaralar uyumluluk garantisini gösterir. Diğer bir yaklaşımla, versiyon numara değişikliği yapılan "yazılım çalışmaları" nın boyutunu gösterir. Küçük versiyon numara değişikliği önemsiz yazılım çalışmalarını işaret ederken, büyük versiyon numara değişikliği önemli ve kapsamlı yazılım çalışmalarını gösterir.

Bu iki gösterim genelde birleşir. Sonuçta, versiyon numaraları uyumluluk yönünden birbirleri ile irtibatlıdır. Fakat bazen, yapılan çalışmaların miktarına göre versiyon numarasındaki ondalık sayıların çeşidi artabilir.

Versiyon numaralarını göstermek için çeşitli kod seçenekleri vardır. XML ile başlayan bir açıklama komutu, XML özelliğinin versiyon numarasını gösterir [12].

Örneğin;

<?xml version="1.0"?>

Aşağıda gösterildiği gibi, aynı versiyon özelliği dizin “xsd:schema” unsuru içinde kullanılabilir:

<xsd:schema version="2.0" ...>

Tanımlanan herhangi bir unsura atanabilen değişik formlarda özellik oluşturabiliriz (bu durumda “versiyon” özelliğine bir isim vermemiz gerekmez).

<LineItem version="2.0">

Aşağıda gösterildiği gibi diğer bir alternatif yaklaşım, versiyon numarasını bir namespace (adboşluğu) içine katmaktır:

<LineItem xmlns="http://actioncon.com/schema/po/v2">

Aşağıda gösterildiği gibi versiyon revizyon çalışmalarında tarih bilgilesi “namespace” içinde kullanılabilir:

<LineItem xmlns="http://actioncon.com/schema/po/2010/09">

Bu durumda, versiyon tanımlayıcısı olarak tarih değişikliği rol oynar. XML taslak tanımlama versiyon deyimlerinin WSDL tanımlama versiyonları ile uyumlu olmalarını korumak için tarih bilgileri yerine versiyon numaralarının kullanılması gerekir. Ancak, XML şema tanımlamalarının ayrı olduğu serbest bir ortamda çalışıldığında, versiyon tanımlayıcılarının WSDL tanımlamaları tarafından kullanılanlardan farklı olması olağandışı değildir [15].

Hangi seçenek seçilirse seçilsin, “Kurallara Uygun Versiyon (Canonical Versioning)” oluşturma modelinin dikkate alınması önemlidir. Bu modelde, bir

servis envanter sınırları içinde olan bütün servis sözleşmelerindeki versiyon bilgi tanımlamalarının standart olmasına dikkat edilir. Geniş ortamlarda; doğrusallığın, uyumluluğun ve versiyon bilgi kalitesinin tanımının garantisi için merkezi bir otoriteye ihtiyaç duyulacaktır. Bu tür kurallar, hizmet sonlandırma bilgisinin nasıl aktarılabileceğini ifade eder.

2.4 Versiyon Oluşturma Stratejileri

Herkese doğru gelecek bir versiyon oluşturma yöntemi yoktur. Çünkü versiyon oluşturma, bir servisin kullanımı süresince kontrolü temsil eder.

Web servis sözleşmelerini kapsayan WSDL, XML taslak ve WS-İlke'lerini içeren fiili bir versiyon oluşturma tekniği olmasa bile, bir çok genel versiyon oluşturma yöntemi ortaya çıkmıştır [4].

Aşağıda ayrı ayrı bilinen üç strateji açıklanacaktır. Bu stratejiler;

- Otoriter Strateji (Yeni Değişiklik, Yeni Sözleşme)
- Esnek Strateji (Geriye Dönük Uyumluluk)
- Serbest Strateji (Geriye ve İleriye Uyumlu)

Otoriter (Sıkı): Herhangi bir uyumlu veya uyumsuz değişiklik, bir servisin sözleşmesinin yeni versiyonunun gerekliliğini ortaya çıkaracaktır. Bu yöntem geriye veya ileriye dönük uyumluluğu desteklememektedir.

Esnek: Herhangi bir uyumsuz değişiklik, bir servis sözleşmesinin yeni versiyonunun gerekliliğini ortaya çıkaracaktır. Bu yeni sözleşme geriye dönük uyumluluğu sağlayacak şekilde tasarlanırken ileriye dönük uyumluluğu desteklemez.

Serbest: Herhangi bir uyumsuz değişiklik, bir servis sözleşmesinin yeni versiyonunun gerekliliğini ortaya çıkaracaktır. Bu yeni sözleşme geriye ve ileriye dönük uyumluluğu sağlayacak şekilde tasarlanır.

2.4.1 Otoriter Strateji

Web servis sözleşme versiyonu oluşturmada en basit yöntem, herhangi bir bölümde herhangi bir değişiklik yapılırsa yeni bir versiyon oluşturmadır.

Bu genellikle, sözleşme ile ilgili olarak WSDL, XML taslak'ta veya WS-İlke içeriğine uyumlu veya uyumsuz değişiklikler yapıldığında bir WSDL tanım değerindeki hedef "namespace" değiştirilerek uygulanır. "namespace" ler bir versiyonun özniteliği yerine versiyon tanıtımı olarak kullanılır. "namespace" değerinin değiştirilmesi durumunda otomatik olarak kullanıcı programını komple değiştirmek zorunda kalır [5].

Bu "süper-kuralcı" yaklaşım pratik değildir. Fakat web hizmet sözleşmesi değişikliklerinin yasal sonuçları olduğunda en güvenli ve bazen en garantili bir durumdur. Çünkü, uyumlu ve uyumsuz değişikliklerin her ikisi de yeni bir hizmet sözleşme versiyonu oluşturma sonucuna sebep olacaktır. Bu yaklaşım ne geriye ne de ileriye dönük uyumluluğu desteklemeyecektir.

Avantaj ve Dezavantajlar

Bu stratejinin yararı, hizmet sözleşmesinin evrimi üzerinde tam denetime sahip olmaktır. Çünkü, ileriye ve geriye doğru uyumluluk kasten göz ardı edildiği için özellikle herhangi bir değişikliğin etkisi ile endişeye kapılmanıza gerek yoktur (çünkü tüm değişiklikler etkin olarak sözleşmeyi bozar).

Olumsuz yanı ise her değişiklik ile sözleşme üzerine yeni bir "namespace" zorlayarak, mevcut tüm servis kullanıcı programlarının herhangi bir yeni sürüm ile uyumlu olmayacağını garanti edersiniz. Eski sözleşme yeni versiyonun yanında dururken kullanıcılar sadece web servisi ile veya kullanıcılar kendileri yeni sözleşmeyi uygun şekilde güncelleştirinceye kadar haberleşmeye devam edebileceklerdir.

Bu nedenle, bu yaklaşım bireysel hizmetler yönetim yükünü artıracak ve dikkatli geçiş stratejileri gerektirecektir. Aynı zamanda iki veya daha fazla versiyonun var olması destekleyici sözleşme envanter altyapısına ihtiyaç olan ortak bir gereklilik haline gelebilir.

2.4.2 Esnek Strateji

Geriye dönük uyumlu değişiklikler ile mevcut kullanıcı programlar genişleyebilir veya büyüyebilir ancak olumsuz etkilenmezler. En genel örnek WSDL tanımına yeni bir operasyon ilavesi veya bir mesajın şema tanımına isteğe bağlı bir öğenin eklenmesidir.

Avantaj ve Dezavantajlar

Bu yaklaşımın temel avantajı kullanıcı programlarını sürekli geriye dönük uyumlu tutarken çeşitli değişiklikleri de barındırabilir. Ancak, uyumlu değişiklikler yapıldığında bu değişiklikler kalıcı olur ve uyumsuz bir değişiklik yapılmadan geri alınamazlar. Bu nedenle, sözleşmelerin karmaşık olmaması için her değişikliğin değerlendirildiği bir yönetimi gereklidir.

2.4.3 Serbest Strateji

Önceki iki yaklaşım gibi; bu strateji, uyumsuz değişiklikler sonucu yeni bir hizmet sözleşmesi versiyonu oluşturulmasını gerektirir. Buradaki fark, hizmet sözleşmelerinin başlangıçta nasıl tasarlandığıdır.

Bilinen veri değişim gereksinimlerinin yerine, kendiliğinden genişleyebilen sözleşmelerin parçalarını yapmak için WSDL, XML şema ve WS-İlke dillerinden

diğer özellikler kullanılır. Böylece bilinmeyen veri değişim gereksinimleri geleceği desteklemeye devam etmektedir.

- WSDL 2.0 dil tarafından sağlanan "anyType" öznitelik değeri herhangi bir geçerli XML belgesinin oluşması için bir mesaj verir.
- XML taslak sembolleri bilinmeyen bir dizi verilerin mesaj tanımları içinde aktarılmasını sağlamakta kullanılabilirler.
- Yok sayılabilir ilke ifadeleri servis özellikleri ile iletişim kurmak için tanımlanabilirler ve gelecekteki kullanıcılar tarafından isteğe bağlı kabul edilebilirler.

Avantaj ve Dezavantajlar

Sembollerin tanımsız içeriklerinin web servis sözleşmeleri aracılığı ile geçmelerine aracı oldukları bir gerçektir. Bu durum kabul edilebilir mesaj unsuru ve veri içeriği aralıklarının daha da genişletilmesi için daimi bir fırsat verir. Diğer taraftan, sembollerin kullanımı temel hizmet mantığı doğrulama yükünün olduğu belirsiz ve aşırı kaba servis sözleşmeleri sonucunu getirecektir.

3. WSDL TANIMLAMA VERSİYONU

Mesaj tanım türleri ve politikalarında yapılan değişikliklerin daha az belirgin ve genellikle tamamlayıcı olmasına karşı, bir WSDL belgesinde yapılan değişikliklerin etkisi daha görünür olacaktır [18].

Bu başlık altında yer alan alt başlıklarda "Sıkı" ve "Esnek" versiyon oluşturma yaklaşımlarının ifade edilmesi hedeflenmektedir. "Gevşek" yaklaşıma ilişkin teknikler bu başlığın sonunda "WSDL Tanımlamaları ve İleriye Dönük Uyumluluk" kısmında özetlenmiştir.

3.1 Sürüm ve WSDL Tanımlamaları

Bir sözleşmeye ait doküman içerisindeki WSDL tanımı uyumlu veya uyumsuz değişikliklere uğrayabilir. Versiyon strateji ve kurallarımıza bağlı olarak en iyi tanımlayıcıları kullanılarak bu değişikliklerin nasıl uygulanacağına karar verilir.

XML şeması "xsd: schema" öğesi aksine, WSDL tanımlama öğeleri kullanmamız için yerleşik bir versiyon sağlamaz. Bu nedenle, sürüm numarasını saklamak için WSDL belgeleme öğeleri kullanılır [18].

Versiyon şeması oluşturma kurallarımız aşağıdaki gibidir:

- Küçük ve büyük sözleşme sürüm numaraları, öğe tanımının açıldığı bir dokümantasyon elemanı kullanılarak ifade edilecektir. Sürüm numaraları aşağıda olduğu gibi "Sürüm" sözcüğünden sonra gösterilir [9]:
  <documentation>Version 1.0</documentation>

- Aşağıda gösterildiği gibi, WSDL tanım hedef adı'na ana sürüm numaraları eklenecek ve önüne "v" harfi konulacaktır:
  http://fatihbalikci.com/contract/po/v2
- Önceki kuralın istisnası, WSDL tanımının ilk sürümünün yayımlandığı zamandır. Bu durumda, herhangi bir versiyon numarası ad'a (namespace) eklenmez.
- Uyumlu bir değişiklik, küçük (minör) versiyon numarasını artırır ve WSDL tanım hedef adını değiştirmez [3].
- Uyumsuz bir değişiklik, büyük (majör) versiyon numarasını artırır ve yeni bir WSDL tanım hedef adının oluşmasına sebep olur [3].

Şekil 3.1'de sürüm tanıtım stratejimize bağlı olarak bir WSDL tanımının 2.1 sürümünün nasıl ifade edilebileceğini gösterir:

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po/v2"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po/v2"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation>Version 2.1</documentation>
  ...
</definitions>
```

Şekil 3.1 Sürüm tanımlayıcıları [5]

Şekil 3.1'de hedef ad ile tanım elemanı web hizmeti sözleşmesinin ikinci sürüm olduğunu ifade eder.

Uyumlu veya uyumsuz değişikliklerin temeli nerede olursa olsun, Şekil 3.1'deki yaklaşım ile sürüm tanımlamanın yapıldığı ifade edilmektedir. Sürüm tanımlama ile sadece WSDL alanındaki versiyon numaralarının değişikliği ile işlemler sınırlanmamaktadır. Genellikle değişimin tetiklendiği web servis sözleşmesi kısmına sürüm kimliğinin eklenmesi hedeflenmektedir.

Örnek, bir şema içindeki mesaj tanımı ile ilişkili XML şema türünde yapılan uyumsuz bir değişiklik versiyon numarasının artırımını gerektirir. İlave olarak

WSDL tanım hedef adında ve dokümantasyon öğelerinde majör (büyük) versiyon numarasının artımını da gerektirir.

Temel Örnek: Önümüzdeki başlıklar altında, Sipariş Satın Alma servisi için mevcut bir web hizmet sözleşmesine çeşitli değişikliklerin ilavesi yapılmaktadır. Bu değişikliklerin çoğu soyut tanımı etkiler.

Burada, orijinal soyut tanım içeren "Satın Alma Sipariş" WSDL tanımı bir temel örnek olarak verilmektedir. Önceki başlıklardaki tasarıma daha önce açıklanan yeni sürüm tanımlayıcıları ilave edilmiştir. Dolayısıyla, temel bir örneğimiz olan "Sipariş Hizmet Sözleşmesi" 2.1 sürümünü temsil eder. Bu örnekte yapılacak değişiklikler kırmızı veya kalın metin ile ifade edilecektir [5].

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po/v2"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns=" http://actioncon.com/contract/po/v2"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation>Version 2.1</documentation>
  <types>
    <xsd:schema>
      <xsd:import namespace="http://actioncon.com/schema/po"
        schemaLocation=
        "http://actioncon.com/schema/purchaseOrder.xsd"/>
    </xsd:schema>
  </types>
  <message name="msgSubmitOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgSubmitOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgCheckOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgCheckOrderResponse">
   <part name="Status" element="po:status"/>
  </message>
  <message name="msgChangeOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgChangeOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
```

```
  </message>
  <message name="msgCancelOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgCancelOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <portType name="ptPurchaseOrder">
    <operation name="opSubmitOrder">
      <input message="tns:msgSubmitOrderRequest"/>
      <output message="tns:msgSubmitOrderResponse"/>
    </operation>
    <operation name="opCheckOrderStatus">
      <input message="tns:msgCheckOrderRequest"/>
      <output message="tns:msgCheckOrderResponse"/>
    </operation>
    <operation name="opChangeOrder">
      <input message="tns:msgChangeOrderRequest"/>
      <output message="tns:msgChangeOrderResponse"/>
    </operation>
    <operation name="opCancelOrder">
      <input message="tns:msgCancelOrderRequest"/>
      <output message="tns:msgCancelOrderResponse"/>
    </operation>
  </portType>
  ...
</definitions>
```

Şekil 3.2 Örnek WSDL [5]

3.2 Sürüm Oluşturma Adımları

WSDL tanımında yapılan değişikliklerin büyük çoğunluğu genellikle işlemler üzerinde gerçekleşmektedir. Sürüm oluşturma adımlarının büyük kısmı, ortak operasyonlar ile ilgili değişiklikleri kapsar:

- Yeni bir işlem ekleme
- Mevcut bir işlemin yeniden adlandırılması
- Mevcut bir işlemin kaldırılması
- Mevcut bir işleminin MEP'ini değiştirme
- Mevcut bir işleme hata mesajı eklenme

3.2.1 Yeni Bir İşlemin Eklenmesi

WSDL tanıma bir işlemin esnek (uyumlu) ve otoriter (sıkı) strateji kullanılarak eklenmesi alt başlıklar halinde incelenmiştir.

3.2.1.1 WSDL'a Bir İşlemin Uyumlu Bir Değişiklik Olarak Eklenmesi

Bir WSDL tanımı uygulanıyor ve kullanılıyorsa, kullanıcı programları mevcut işlem tanımlamalarına bağlı olarak oluşturacaktır. Sözleşmeye yeni işlem tanımlamalarının eklenmesi bu bağımlılıkları etkilemeyecektir. Bu değişikliğin varsayılan olarak geriye doğru uyumlu bir değişiklik olduğu düşünülür [9].

Şekil 3.3'deki vurgulanan metinde görüldüğü üzere, yeni eklenen işlem yapısı ve ilgili mesaj yapısı versiyon numarası hariç WSDL tanımının herhangi bir parçasını etkilemez.

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po/v2"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po/v2"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation>Version 2.2</documentation>
  <types>
    <xsd:schema>
      <xsd:import namespace="http://actioncon.com/schema/po"
        schemaLocation=
        "http://actioncon.com/schema/purchaseOrder.xsd"/>
    </xsd:schema>
  </types>
  <message name="msgSubmitOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgSubmitOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgCheckOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgCheckOrderResponse">
   <part name="Status" element="po:status"/>
  </message>
  <message name="msgChangeOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgChangeOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgCancelOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgCancelOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
```

```
<message name="msgGetOrderRequest">
  <part name="PONumber" element="po:poNumber"/>
</message>
<message name="msgGetOrderResponse">
 <part name="PurchaseOrder" element="po:purchaseOrder"/>
</message>
<portType name="ptPurchaseOrder">
  <operation name="opSubmitOrder">
    <input message="tns:msgSubmitOrderRequest"/>
    <output message="tns:msgSubmitOrderResponse"/>
  </operation>
  <operation name="opCheckOrderStatus">
    <input message="tns:msgCheckOrderRequest"/>
    <output message="tns:msgCheckOrderResponse"/>
  </operation>
  <operation name="opChangeOrder">
    <input message="tns:msgChangeOrderRequest"/>
    <output message="tns:msgChangeOrderResponse"/>
  </operation>
  <operation name="opCancelOrder">
    <input message="tns:msgCancelOrderRequest"/>
    <output message="tns:msgCancelOrderResponse"/>
  </operation>
  <operation name="opGetOrder">
    <input message="tns:msgGetOrderRequest"/>
    <output message="tns:msgGetOrderResponse"/>
  </operation>
 </portType>
</definitions>
```

Şekil 3.3 Esnek yaklaşım ile WSDL'a yeni işlemin eklenmesi [5]

Şekil 3.3'de yeni mesaj yapıları ile birlikte oluşturulan yeni bir işlem, mevcut bir WSDL tanımına basitçe eklenir. Yeni işlemin ilave edilme şekli esnek versiyon yaklaşımını destekler ve yeni bir sözleşme versiyon değişimini zorlamadan uyumlu bir değişiklik yapılmasına imkan tanır.

3.2.1.2 WSDL'a Bir İşlemin Uyumsuz Bir Değişiklik Olarak Eklenmesi

Sıkı bir yaklaşım izlenildiği zaman, yeni işlem ve mesaj tanımlarının ilaveleri sözleşme versiyon numarasını yükseltecektir. Şekil 3.4'de görüldüğü gibi yeni sürümü yansıtacak biçimde WSDL hedef ad değerini değiştirecektir.

```
<definitions name="PurchaseOrder"
  targetNamespace= "http://actioncon.com/contract/po/v3"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po/v3"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation>Version 3.0</documentation>
  ...
</definitions>
```

Şekil 3.4 Sıkı yaklaşım ile WSDL'a yeni işlemin eklenmesi [5]

3.2.2 Mevcut Bir İşlemin Yeniden Adlandırılması

Mevcut bir işlem öğesinin değeri WSDL belgesi dağıtıldıktan sonra değiştirilmesi gerekiyor ise, herhangi bir kullanıcı bundan etkilenecektir [8].

Bu tür bir değişikliği ele almak için iki genel yol vardır:

- Sözleşmenin yeni büyük bir versiyonunun oluşturulması
- Mevcut bir sözleşmeye değiştirilmiş işlem adının eklenmesi

Bir işlemin adını değiştirmek uyumsuz bir değişikliği temsil eder. İlk metot hem Sıkı hem de Esnek versiyon oluşturma stratejisi ile uyumludur. İkinci metot uyumlu bir değişiklik olarak önerilir.

3.2.2.1 Yeni Bir Ana Sözleşme Versiyonunu Zorlamak

Mevcut işlem adının değiştirilmesi, sözleşmede uyumsuz bir değişikliğe neden olur. Şekil 3.5'de gösterildiği gibi yeni bir ana sözleşme versiyonunun oluşturulması gerekecektir.

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po/v3"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po/v3"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation>Version 3.0</documentation>
  <types>
    <xsd:schema>
      <xsd:import namespace=
        "http://actioncon.com/schema/po"
        schemaLocation=
        "http://actioncon.com/schema/purchaseOrder.xsd"/>
    </xsd:schema>
  </types>
  <message name="msgSubmitOrdersRequest">
    <part name="PurchaseOrder" element="po:purchaseOrders"/>
  </message>
  <message name="msgSubmitOrdersResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgCheckOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgCheckOrderResponse">
   <part name="Status" element="po:status"/>
</message>
<message name="msgChangeOrderRequest">
  <part name="PurchaseOrder" element="po:purchaseOrder"/>
</message>
<message name="msgChangeOrderResponse">
 <part name="Acknowledgement" element="po:acknowledgement"/>
</message>
<message name="msgCancelOrderRequest">
  <part name="PONumber" element="po:poNumber"/>
</message>
<message name="msgCancelOrderResponse">
 <part name="Acknowledgement" element="po:acknowledgement"/>
</message>
<message name="msgGetOrderRequest">
  <part name="PONumber" element="po:poNumber"/>
</message>
<message name="msgGetOrderResponse">
 <part name="PurchaseOrder" element="po:purchaseOrder"/>
</message>
```

```
  <portType name="ptPurchaseOrder">
    <operation name="opSubmitOrders">
      <input message="tns:msgSubmitOrdersRequest"/>
      <output message="tns:msgSubmitOrdersResponse"/>
    </operation>
    <operation name="opCheckOrderStatus">
      <input message="tns:msgCheckOrderRequest"/>
      <output message="tns:msgCheckOrderResponse"/>
    </operation>
    <operation name="opChangeOrder">
      <input message="tns:msgChangeOrderRequest"/>
      <output message="tns:msgChangeOrderResponse"/>
    </operation>
    <operation name="opCancelOrder">
      <input message="tns:msgCancelOrderRequest"/>
      <output message="tns:msgCancelOrderResponse"/>
    </operation>
    <operation name="opGetOrder">
      <input message="tns:msgGetOrderRequest"/>
      <output message="tns:msgGetOrderResponse"/>
    </operation>
  </portType>
</definitions>
```

Şekil 3.5 WSDL'da mevcut bir işlemin adının güncellenmesi [5]

Şekil 3.5'de "opSubmitOrder" işlemi, "opSubmitOrders" olarak değiştirilmiştir ve aynı anda birden fazla siparişi eşzamanlı olarak desteklemesi için yeniden geliştirilmiştir. Bu değişiklik mesaj tanımlarını ve temel şema türlerini etkileyebilir.

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po/v3"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po/v3"
  xmlns:po="http://actioncon.com/schema/po/v3"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation>Version 3.0</documentation>
  <types>
    <xsd:schema>
      <xsd:import namespace=
        "http://actioncon.com/schema/po/v3"
        schemaLocation=
        "http://actioncon.com/schema/purchaseOrder.xsd"/>
    </xsd:schema>
  </types>
  ...
</definitions>
```

Şekil 3.6 XML şema'da "namespace" güncellemesi [5]

Şekil 3.6'da XML şema tanım hedef isim alanı (namespace), temel şema içeriğinde yapılan uyumsuz değişiklik dikkate alınarak değiştirilir.

3.2.2.2 Adı Değiştirilen İşlem ile Sözleşmenin Uzatılması

Değişikliğinin nedenlerine göre, sadece mevcut sözleşmeyi korumak ve yeni bir adla yeni bir operasyon tanımı eklemek faydalı olabilir.

Şekil 3.5'deki işlemin aşağıdaki türevinde, orijinal "opSubmitOrder" işlemi ile birlikte "opSubmitOrders" işlemi ilave edilir.

```
<definitions name="PurchaseOrder" targetNamespace=
 "http://actioncon.com/contract/po/v2"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:tns="http://actioncon.com/contract/po/v2"
 xmlns:po="http://actioncon.com/schema/po"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <documentation>Version 2.2</documentation>
 <types>
   <xsd:schema>
     <xsd:import namespace=
       "http://actioncon.com/schema/po"
       schemaLocation=
       "http://actioncon.com/schema/purchaseOrder.xsd"/>
   </xsd:schema>
 </types>
 <message name="msgSubmitOrderRequest">
   <part name="PurchaseOrder" element="po:purchaseOrder"/>
 </message>
 <message name="msgSubmitOrderResponse">
  <part name="Acknowledgement" element="po:acknowledgement"/>
 </message>
 <message name="msgSubmitOrdersRequest">
   <part name="PurchaseOrder" element="po:purchaseOrders"/>
 </message>
 <message name="msgSubmitOrdersResponse">
  <part name="Acknowledgement" element="po:acknowledgement"/>
 </message>
 <message name="msgCheckOrderRequest">
   <part name="PONumber" element="po:poNumber"/>
 </message>
 <message name="msgCheckOrderResponse">
  <part name="Status" element="po:status"/>
 </message>
 <message name="msgChangeOrderRequest">
   <part name="PurchaseOrder" element="po:purchaseOrder"/>
 </message>
 <message name="msgChangeOrderResponse">
  <part name="Acknowledgement" element="po:acknowledgement"/>
 </message>
 <message name="msgCancelOrderRequest">
   <part name="PONumber" element="po:poNumber"/>
 </message>
```

```
<message name="msgCancelOrderResponse">
 <part name="Acknowledgement" element="po:acknowledgement"/>
</message>
<message name="msgGetOrderRequest">
  <part name="PONumber" element="po:poNumber"/>
</message>
<message name="msgGetOrderResponse">
 <part name="PurchaseOrder" element="po:purchaseOrder"/>
</message>
<portType name="ptPurchaseOrder">
  <operation name="opSubmitOrder">
    <input message="tns:msgSubmitOrderRequest"/>
    <output message="tns:msgSubmitOrderResponse"/>
  </operation>
  <operation name="opSubmitOrders">
    <input message="tns:msgSubmitOrdersRequest"/>
    <output message="tns:msgSubmitOrdersResponse"/>
  </operation>
  <operation name="opCheckOrderStatus">
    <input message="tns:msgCheckOrderRequest"/>
    <output message="tns:msgCheckOrderResponse"/>
  </operation>
  <operation name="opChangeOrder">
    <input message="tns:msgChangeOrderRequest"/>
    <output message="tns:msgChangeOrderResponse"/>
  </operation>
  <operation name="opCancelOrder">
    <input message="tns:msgCancelOrderRequest"/>
    <output message="tns:msgCancelOrderResponse"/>
  </operation>
  <operation name="opGetOrder">
    <input message="tns:msgGetOrderRequest"/>
    <output message="tns:msgGetOrderResponse"/>
  </operation>
 </portType>
</definitions>
```

Şekil 3.7 Uyumlu bir değişikliğe örnek olarak bir işlemin ilave edilmesi [5]

Şekil 3.7'de yeni bir "opSubmitOrders" işlemi uyumlu bir değişiklik olarak ilave edilir. Bu senaryoda; yeni işlem ilavesi, sözleşme'de normal olmayan tasarım modelinin uygulamasını gösterir ve mevcut hizmet sözleşmesinde örtüşen işlevsellik ile operasyonlara izin verir.

3.2.2.3 İşlemin Sonlandırılması

Bu durumda adı değiştirilmiş işlem, orijinal işlem gibi tam olarak aynı temel işlevselliği temsil eder. Bu yaklaşım geçiş olarak kabul edilebilir. Yeni bir sürümü zorlayan bir sözleşme gibidir ve aynı anda hem eski ve hem de yeni sözleşmeleri içerir. Bağlantı noktası türü, aynı işlemin eski ve yeni sürümlerini içerecektir.

Şekil 3.8'de gösterildiği gibi, hedef sona erme tarihi ile iletişim kurmak için bir açıklama eklenir.

```
<portType name="ptPurchaseOrder">
  <documentation>
    opSubmitOrder is Scheduled for Terminatation on 01/12
  </documentation>
  <operation name="opSubmitOrder">
    <input message="tns:msgSubmitOrderRequest"/>
    <output message="tns:msgSubmitOrderResponse"/>
  </operation>
  ...
</portType>
```

Şekil 3.8 Bir işleme hedef sona erme tarihinin eklenmesi [5]

"opSubmitOrder" işleminin sona erme tarihini göstermek için bir açıklama eklenir.

3.2.3 Mevcut Bir İşlemin Kaldırılması

Bir sözleşmeden bir işlem tanımının gerçekten kaldırılması tahmin edilebileceği gibi bağımlı olan tüketiciyi etkileyecektir. Çünkü, işlem kaldırıldıktan sonra ilgili web servis bu işlemi kullanmak isterse tahmin edilebileceği gibi başarısız olacaktır [16].

Bir işlemin kaldırılması ve bu değişikliğin kapsanması için iki genel yaklaşım vardır:

- İşlem tanımı silinir, sözleşmenin yeni ana versiyonu zorunlu hale gelir.
- İşlem tanımı fonksiyonel bir rezerv olarak korunur.

3.2.3.1 Yeni Ana Sözleşme Versiyonunun Zorlanması

İşlem yapısı ile ilgili mesaj yapıları kaldırılır ve yeni WSDL tanımı yayınlanır.

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po/v3"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po/v3"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation>Version 3.0</documentation>
  <types>
    <xsd:schema>
      <xsd:import namespace=
        "http://actioncon.com/schema/po"
        schemaLocation=
        "http://actioncon.com/schema/purchaseOrder.xsd"/>
    </xsd:schema>
  </types>
  <message name="msgSubmitOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgSubmitOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>

  <!-- msgCheckOrderRequest Removed 01/12 -->

  <!-- msgCheckOrderResponse Removed 01/12 -->

  <message name="msgChangeOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgChangeOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgCancelOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgCancelOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgGetOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgGetOrderResponse">
   <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
```

```
  <portType name="ptPurchaseOrder">
    <operation name="opSubmitOrder">
      <input message="tns:msgSubmitOrderRequest"/>
      <output message="tns:msgSubmitOrderResponse"/>
    </operation>

    <!-- opCheckOrderStatus Removed 01/12 -->

    <operation name="opChangeOrder">
      <input message="tns:msgChangeOrderRequest"/>
      <output message="tns:msgChangeOrderResponse"/>
    </operation>
    <operation name="opCancelOrder">
      <input message="tns:msgCancelOrderRequest"/>
      <output message="tns:msgCancelOrderResponse"/>
    </operation>
    <operation name="opGetOrder">
      <input message="tns:msgGetOrderRequest"/>
      <output message="tns:msgGetOrderResponse"/>
    </operation>
  </portType>
</definitions>
```

Şekil 3.9 Mevcut bir işlemin WSDL tanımından kaldırılması [5]

Şekil 3.9'de bir işlem ve mesaj yapısının WSDL tanımından silinmesi ifade edilmektedir.

Kırmızı ile vurgulanan açıklamalar operasyon ile ilgili içeriğin sözleşmeden çıkarıldığını belirtir. Bu kullanıcı tasarımcıları için tarihsel bir referans olarak yardımcı olabilir.

3.2.3.2 Bir İşlemi İşlevsel Kalıntıya Dönüştürmek

Bir işlemin işlevselliğini silmenin bir yolu, (ancak ortadan kaldırmadan) orijinal işlem tanımını bir kalıntıya (stub) dönüştürmektir:

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po/v3"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po/v3"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:er="http://actioncon.com/schema/po/errors"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation>Version 3.0</documentation>
  <types>
    <xsd:schema>
      <xsd:import namespace=
        "http://actioncon.com/schema/po"
        schemaLocation=
        "http://actioncon.com/schema/purchaseOrder.xsd"/>
    </xsd:schema>
    <xsd:schema targetNamespace=
      "http://actioncon.com/schema/po/errors">
      <xsd:element name="statusError" type="xsd:string"/>
    </xsd:schema>
  </types>
  <message name="msgSubmitOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgSubmitOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgCheckOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgCheckOrderResponse">
   <part name="StatusError" element="er:statusError"/>
  </message>
  <message name="msgChangeOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgChangeOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgCancelOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgCancelOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgGetOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgGetOrderResponse">
   <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <portType name="ptPurchaseOrder">
    <operation name="opSubmitOrder">
      <input message="tns:msgSubmitOrderRequest"/>
      <output message="tns:msgSubmitOrderResponse"/>
    </operation>
```

```
    <operation name="opCheckOrderStatus">
      <input message="tns:msgCheckOrderRequest"/>
      <output message="tns:msgCheckOrderResponse"/>
    </operation>
    <operation name="opChangeOrder">
      <input message="tns:msgChangeOrderRequest"/>
      <output message="tns:msgChangeOrderResponse"/>
    </operation>
    <operation name="opCancelOrder">
      <input message="tns:msgCancelOrderRequest"/>
      <output message="tns:msgCancelOrderResponse"/>
    </operation>
    <operation name="opGetOrder">
      <input message="tns:msgGetOrderRequest"/>
      <output message="tns:msgGetOrderResponse"/>
    </operation>
  </portType>
</definitions>
```

Şekil 3.10 Kaldırılan işlemin korunması ve davranışının değiştirilmesi [5]

"msgCheckOrderRequest" (sipariş istek kontrol) mesajı değişmeden kalırken, "opCheckOrderStatus" (sipariş durum kontrolü) işlemi aynı giriş mesajını kabul etmeye devam eder. Ancak sipariş durumunu dönmek yerine, bu işlemin artık kullanılamayacağını gösteren bir hata mesajı kullanıcıya dönmektedir.

Mesaj ayrı bir şema olan metin yapının içindedir ve "purchaseOrder.xsd" (sipariş mektubu) şema belgesi üzerinde versiyon oluşturmayı gerektirmez.

Kullanıcı tarafından da başarıyla onaylandıktan sonra hata mesajının şema tipi ile uyumlu olup olmamasına karar verilmesi gerekmektedir.

Bu uygun bir değişiklik gibi görünse de "opCheckOrderStatus" işlemi ile ilgili veri ve davranış önemli ölçüde artmıştır ki, bu hemen hemen her zaman uygun olmayan bir değişikliğe neden olacaktır. Dolayısıyla bu da yeni bir ana sözleşme versiyonunu gerektirir.

3.2.4 Mevcut Bir İşlemin MEP'ini Değiştirme

Müşterilerin mevcut durumda kullandıkları işlemlerin giriş ve çıkış mesaj tanımlamaları MEP (Message Exchange Patterns)'e bağımlıdır. Bu nedenle mevcut bir işleme giriş veya çıkış mesajı eklenmesi veya bu mesaj tanımlamalarının sıralamasının değiştirilmesi sözleşmede uyumsuz bir değişikliğe sebep olacaktır [16].

3.2.4.1 Yeni Bir Büyük Sözleşme Versiyonunun Zorlanması

Şekil 3.11'de "opCancelOrder" işlemideki istek (request) ve yanıt (response) mesajlarından yanıt mesajını kaldıran tek-yönlü MEP'e geçilmektedir.

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po/v3"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po/v3"
  xmlns:pc="http://actioncon.com/schema/po"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation>Version 3.0</documentation>
  <types>
    <xsd:schema>
      <xsd:import namespace=
        "http://actioncon.com/schema/po"
        schemaLocation=
        "http://actioncon.com/schema/purchaseOrder.xsd"/>
    </xsd:schema>
  </types>
  <message name="msgSubmitOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgSubmitOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgCheckOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgCheckOrderResponse">
   <part name="Status" element="po:status"/>
  </message>
  <message name="msgChangeOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgChangeOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgCancelOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
```

```
<!-- msgCancelOrderResponse Removed 01/12 -->

<message name="msgGetOrderRequest">
  <part name="PONumber" element="po:poNumber"/>
</message>
<message name="msgGetOrderResponse">
 <part name="PurchaseOrder" element="po:purchaseOrder"/>
</message>
<portType name="ptPurchaseOrder">
  <operation name="opSubmitOrder">
    <input message="tns:msgSubmitOrderRequest"/>
    <output message="tns:msgSubmitOrderResponse"/>
  </operation>
  <operation name="opCheckOrderStatus">
    <input message="tns:msgCheckOrderRequest"/>
    <output message="tns:msgCheckOrderResponse"/>
  </operation>
  <operation name="opChangeOrder">
    <input message="tns:msgChangeOrderRequest"/>
    <output message="tns:msgChangeOrderResponse"/>
  </operation>
  <operation name="opCancelOrder">
    <input message="tns:msgCancelOrderRequest"/>
  </operation>
  <operation name="opGetOrder">
    <input message="tns:msgGetOrderRequest"/>
    <output message="tns:msgGetOrderResponse"/>
  </operation>
 </portType>
</definitions>
```

Şekil 3.11 Yanıt(response) yapısının kaldırılması [5]

Şekil 3.11'de "msgCancelOrderResponse" mesaj tanımlamasına artık gerek olmayacağı için kaldırılmıştır. Bu değişikliğin sonucunda yeni bir sözleşme versiyonuna gerek duyulmaktadır. Yeni bir hedef isim uzayına ihtiyaç duyulduğundan dolayı Şekil 3.11'de uygun olmayan bir değişiklik ifade edilmektedir.

3.2.4.2 Değiştirilmiş MEP İçeren İşlemle Sözleşme'yi Genişletme

Alternatif bir teknik de yeni MEP içeren yeni bir işlem tanımını WSDL tanımına eklenmesidir. Şekil 3.12'de gösterildiği gibi orijinal işlem değiştirilen MEP ile birlikte işlem yapmasına izin vermektedir:

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po/v2"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po/v2"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation>Version 2.2</documentation>
  <types>
    <xsd:schema>
      <xsd:import namespace="http://actioncon.com/schema/po"
        schemaLocation=
        "http://actioncon.com/schema/purchaseOrder.xsd"/>
    </xsd:schema>
  </types>
  <message name="msgSubmitOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgSubmitOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgCheckOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgCheckOrderResponse">
   <part name="Status" element="po:status"/>
  </message>
  <message name="msgChangeOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgChangeOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgCancelOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgCancelOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgCancelOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgCancelOrderNotifyRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
```

```
<message name="msgGetOrderRequest">
  <part name="PONumber" element="po:poNumber"/>
</message>
<message name="msgGetOrderResponse">
 <part name="PurchaseOrder" element="po:purchaseOrder"/>
</message>
<portType name="ptPurchaseOrder">
  <operation name="opSubmitOrder">
    <input message="tns:msgSubmitOrderRequest"/>
    <output message="tns:msgSubmitOrderResponse"/>
  </operation>
  <operation name="opCheckOrderStatus">
    <input message="tns:msgCheckOrderRequest"/>
    <output message="tns:msgCheckOrderResponse"/>
  </operation>
  <operation name="opChangeOrder">
    <input message="tns:msgChangeOrderRequest"/>
    <output message="tns:msgChangeOrderResponse"/>
  </operation>
  <operation name="opCancelOrder">
    <input message="tns:msgCancelOrderRequest"/>
    <output message="tns:msgCancelOrderResponse"/>
  </operation>
  <operation name="opCancelOrderNotify">
    <input message="tns:msgCancelOrderNotifyRequest"/>
  </operation>
  <operation name="opGetOrder">
    <input message="tns:msgGetOrderRequest"/>
     <output message="tns:msgGetOrderResponse"/>
   </operation>
 </portType>
</definitions>
```

Şekil 3.12 Değiştirilmiş MEP içeren işlemle sözleşmeyi genişletme [5]

Şekil 3.12'de kırmızı ile gösterilen yerler yeni mesaj tanımı ile birlikte WSDL tanımlamasına eklenen yeni "opCancelOrderNotify" işlemini göstermektedir. Koyu olarak gösterilenler ise korunan orijinal "opCancelOrder" ile ilgili öğelerdir.

3.2.5 Mevcut Bir İşleme Hata Mesajı Ekleme

Bu değişikliğin gerekliliği web servis sözleşmesinde kullanılan bir işlemin daha önce bir hata mesajı gerektirmediği, ancak yeni bir müşterinin bu işlemi kullanmak istediğinde ortaya çıkar.

3.2.5.1 Hata Mesajının Eklenmesi

Bir hata mesajının eklenmesi gibi bir değişikliğin yapılması durumunda sözleşme'nin geriye dönük uyumluluğunu etkileyip etkilemeyeceği sorusu akıllara gelir. Şekil 3.13'de gösterildiği gibi hem sözleşme hem de mevcut bir işlemin davranışında önemli değişiklik yapılmaktadır.

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po/v3"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po/v3"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:ft="http://actioncon.com/schema/po/faults"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation>Version 3.0</documentation>
  <types>
    <xsd:schema>
      <xsd:import namespace="http://actioncon.com/schema/po"
        schemaLocation=
        "http://actioncon.com/schema/purchaseOrder.xsd"/>
    </xsd:schema>
    <xsd:schema targetNamespace=
      "http://actioncon.com/schema/po/faults">
      <xsd:element name="poFault" type="xsd:string"/>
    </xsd:schema>
  </types>
  <message name="msgSubmitOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgSubmitOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgSubmitOrderFault">
    <part name="POFault" element="ft:poFault"/>
  </message>
  <message name="msgCheckOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
  <message name="msgCheckOrderResponse">
   <part name="Status" element="po:status"/>
  </message>
  <message name="msgChangeOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgChangeOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgCancelOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
  </message>
```

```
<message name="msgCancelOrderResponse">
 <part name="Acknowledgement" element="po:acknowledgement"/>
</message>
<message name="msgGetOrderRequest">
  <part name="PONumber" element="po:poNumber"/>
</message>
<message name="msgGetOrderResponse">
 <part name="PurchaseOrder" element="po:purchaseOrder"/>
</message>
<portType name="ptPurchaseOrder">
  <operation name="opSubmitOrder">
    <input message="tns:msgSubmitOrderRequest"/>
    <output message="tns:msgSubmitOrderResponse"/>
    <fault message="tns:msgSubmitOrderFault"/>
  </operation>
  <operation name="opCheckOrderStatus">
    <input message="tns:msgCheckOrderRequest"/>
    <output message="tns:msgCheckOrderResponse"/>
  </operation>
  <operation name="opChangeOrder">
    <input message="tns:msgChangeOrderRequest"/>
    <output message="tns:msgChangeOrderResponse"/>
  </operation>
  <operation name="opCancelOrder">
    <input message="tns:msgCancelOrderRequest"/>
    <output message="tns:msgCancelOrderResponse"/>
  </operation>
  <operation name="opGetOrder">
    <input message="tns:msgGetOrderRequest"/>
    <output message="tns:msgGetOrderResponse"/>
  </operation>
 </portType>
</definitions>
```

Şekil 3.13 Mevcut bir işleme hata mesajı'nın eklenmesi [5]

Şekil 3.13'de sözleşmeye hata mesaj tanımlaması için gerekli tip eklemektedir.

Bazı durumlarda müşteriye beklemediği bir hata mesajının gönderilmesi ihmal edilebilir. Hata mesaj gönderildi, hata durumuna düştü ve bir hata kaydı yapıldı ancak müşteri bu olaydan hiç rahatsız olmadı ve bu değişiklikten hiç etkilenmedi. Böyle bir seçeneğin olup olmaması gerçek zamanlı servis sağlayıcı olan web servise ve müşteriye bağlıdır.

Bu durumlar uyumlu bir değişiklik gibi görülsede tavsiye edilen bir yaklaşım değildir. Gelecekte yapılacak platform veya teknolojik değişiklikler hata mesajı işlevinin davranışını değiştirebileceği için tahmin edilemeyen sonuçların ortaya çıkmasına neden olabilir.

3.2.5.2 Hata Mesajı İçeren İşlemle Sözleşmeyi Genişletme

MEP'de yapılan değişiklikler sonucunda aynı operasyona farklı bir operasyon adı ile hata verebilen bir versiyon eklenmesi alternatifi de mevcuttur:

```
<portType name="ptPurchaseOrder">
  ...
  <operation name="opSubmitOrder">
    <input message="tns:msgSubmitOrderRequest"/>
    <output message="tns:msgSubmitOrderResponse"/>
  </operation>
  <operation name="opSubmitOrderWithFault">
    <input message="tns:msgSubmitOrderRequest"/>
    <output message="tns:msgSubmitOrderResponse"/>
    <fault message="tns:msgSubmitOrderFault"/>
  </operation>
  ...
</portType>
```

Şekil 3.14 WSDL tanımına "opSubmitOrderWithFault" işleminin eklenmesi [5]

Şekil 3.14'de görüldüğü gibi "opSubmitOrder" ve "opSubmitOrderWithFault" işlemleri giriş ve çıkış mesajları için aynı mesaj tanımlamalarını içermektedir.

3.3 Port Tipi Tanımlamaların Versiyonlanması

Port tipini etkileyen genel WSDL tanımlamalarının nasıl versiyonlanacağını ve tek bir sözleşmenin nasıl çoklu "portType" yapıları ile (herbiri farklı bir versiyonu temsil eden) değiştirilebileceği incelenmektedir.

3.3.1 Port Tipine Bir Versiyon Belirleyici İlave Etme

Verilen bir "port tipi (portType)" sözleşmesi içerisindeki tüm işlemlerdeki versiyonlarla tam olarak iletişime geçmek için "portType" öğesinin "name" özelliğine ana versiyon numarasını ekleyebilir. Her WSDL tanımlamasının hedef "isim uzayı" değiştirildiğinde güncellenebilir [5].

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po/v3"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po/v3"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation>Version 3.0</documentation>
  ...
  <portType name="ptPurchaseOrder-v3">
    ...
  </portType>
  ...
</definitions>
```

Şekil 3.15 WSDL tanımlamasında "portType" isim değişikliği [5]

3.3.2 Çoklu Port Tipleriyle Versiyonlama

WSDL tanımlamalarına çoklu "portType" yapıları "host" edebildiği için çok-arayüzlü WSDL dokümanı oluşturulabilir.

Şekil 3.16'daki WSDL tanımlaması uygun olmayan değişiklikleri içeren üç "portType" yapısını ifade etmektedir.

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation>Versions 1.0, 2.0, 3.0</documentation>
  <types>
    <xsd:schema>
      <xsd:import namespace="http://actioncon.com/schema/po"
        schemaLocation=
        "http://actioncon.com/schema/purchaseOrder.xsd"/>
    </xsd:schema>
  </types>
  <message name="msgSubmitOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
```

```
<message name="msgSubmitOrderResponse">
 <part name="Acknowledgement" element="po:acknowledgement"/>
</message>
<message name="msgCheckOrderRequest">
  <part name="PONumber" element="po:poNumber"/>
</message>
<message name="msgCheckOrderResponse">
 <part name="Status" element="po:status"/>
</message>
<message name="msgChangeOrderRequest">
  <part name="PurchaseOrder" element="po:purchaseOrder"/>
</message>
<message name="msgChangeOrderResponse">
 <part name="Acknowledgement" element="po:acknowledgement"/>
</message>
<message name="msgCancelOrderRequest">
  <part name="PONumber" element="po:poNumber"/>
</message>
<message name="msgCancelOrderResponse">
 <part name="Acknowledgement" element="po:acknowledgement"/>
</message>
<message name="msgGetOrderRequest">
  <part name="PONumber" element="po:poNumber"/>
</message>
<message name="msgGetOrderResponse">
 <part name="PurchaseOrder" element="po:purchaseOrder"/>
</message>

<!-- For ptPurchaseOrder-v2 -->
<message name="msgSubmitOrdersRequest">
  <part name="PurchaseOrder" element="po:purchaseOrders"/>
</message>
<message name="msgSubmitOrdersResponse">
 <part name="Acknowledgement" element="po:acknowledgement"/>
</message>

<portType name="ptPurchaseOrder-v1">
  <operation name="opSubmitOrder">
    <input message="tns:msgSubmitOrderRequest"/>
    <output message="tns:msgSubmitOrderResponse"/>
  </operation>
  <operation name="opCheckOrderStatus">
    <input message="tns:msgCheckOrderRequest"/>
    <output message="tns:msgCheckOrderResponse"/>
  </operation>
  <operation name="opChangeOrder">
    <input message="tns:msgChangeOrderRequest"/>
    <output message="tns:msgChangeOrderResponse"/>
  </operation>
  <operation name="opCancelOrder">
    <input message="tns:msgCancelOrderRequest"/>
    <output message="tns:msgCancelOrderResponse"/>
  </operation>
  <operation name="opGetOrder">
    <input message="tns:msgGetOrderRequest"/>
    <output message="tns:msgGetOrderResponse"/>
  </operation>
```

```
</portType>

<portType name="ptPurchaseOrder-v2">
  <operation name="opSubmitOrders">
    <input message="tns:msgSubmitOrdersRequest"/>
    <output message="tns:msgSubmitOrdersResponse"/>
  </operation>
  <operation name="opCheckOrderStatus">
    <input message="tns:msgCheckOrderRequest"/>
    <output message="tns:msgCheckOrderResponse"/>
  </operation>
  <operation name="opChangeOrder">
    <input message="tns:msgChangeOrderRequest"/>
    <output message="tns:msgChangeOrderResponse"/>
  </operation>
  <operation name="opCancelOrder">
    <input message="tns:msgCancelOrderRequest"/>
    <output message="tns:msgCancelOrderResponse"/>
  </operation>
  <operation name="opGetOrder">
    <input message="tns:msgGetOrderRequest"/>
    <output message="tns:msgGetOrderResponse"/>
  </operation>
</portType>

<portType name="ptPurchaseOrder-v3">
  <operation name="opSubmitOrder">
    <input message="tns:msgSubmitOrderRequest"/>
    <output message="tns:msgSubmitOrderResponse"/>
  </operation>
  <!-- opCheckOrderStatus Removed 01/12 -->
  <operation name="opChangeOrder">
    <input message="tns:msgChangeOrderRequest"/>
    <output message="tns:msgChangeOrderResponse"/>
  </operation>
  <operation name="opCancelOrder">
    <input message="tns:msgCancelOrderRequest"/>
    <output message="tns:msgCancelOrderResponse"/>
  </operation>
  <operation name="opGetOrder">
    <input message="tns:msgGetOrderRequest"/>
    <output message="tns:msgGetOrderResponse"/>
  </operation>
</portType>
...
</definitions>
```

Şekil 3.16 Üç "portType" yapısını barındıran WSDL tanımlaması [5]

Ana versiyonlama, yeni "port" tipleri ile gösterildiği için "definitions" elemanlarının hedef "isimuzayı" değişmemiş şekilde kalır ve ana versiyon numaraları "portType" elemanlarının "name" özellikleri içerisine birleştirilir. "documentation" elemanı versiyon numaralarının listesini tutarak desteklenen versiyonların takip edilmesini sağlar.

3.3.3 Versiyonlarla İlişkilendirmek İçin Örnek Kullanım

Değişik port tipi versiyonlarını desteklemek için yeni XML şema tipleri eklerken "isimuzayı" örnekleri versiyon numaralarını gösterecek şekilde verilebilir.

Şekil 3.17 "purchase-Order.xsd" şema içerisinde bulunmayan yeni bir tipe ihtiyaç duyan bir işlemi (operation) içeren farklı versiyon2 ve versiyon3 "portType" yapıları gösterilmektedir.

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:v2="http://actioncon.com/schema/po/errors"
  xmlns:v3="http://actioncon.com/schema/po/faults"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation>Versions 1.0, 2.0, 3.0</documentation>
  <types>
    <xsd:schema>
      <xsd:import namespace=
        "http://actioncon.com/schema/po"
        schemaLocation=
        "http://actioncon.com/schema/purchaseOrder.xsd"/>
    </xsd:schema>
    <xsd:schema targetNamespace=
      "http://actioncon.com/schema/po/errors">
      <xsd:element name="statusError" type="xsd:string"/>
    </xsd:schema>
    <xsd:schema targetNamespace=
      "http://actioncon.com/schema/po/faults">
      <xsd:element name="poFault" type="xsd:string"/>
    </xsd:schema>
  </types>
  <message name="msgSubmitOrderRequest">
    <part name="PurchaseOrder" element="po:purchaseOrder"/>
  </message>
  <message name="msgSubmitOrderResponse">
   <part name="Acknowledgement" element="po:acknowledgement"/>
  </message>
  <message name="msgSubmitOrderFault">
    <part name="POFault" element="v3:poFault"/>
  </message>
  <message name="msgCheckOrderRequest">
    <part name="PONumber" element="po:poNumber"/>
```

```
</message>
<message name="msgCheckOrderResponse">
 <part name="Status" element="po:status"/>
</message>
<message name="msgCheckOrderResponseErr">
 <part name="StatusError" element="v2:statusError"/>
</message>
<message name="msgChangeOrderRequest">
  <part name="PurchaseOrder" element="po:purchaseOrder"/>
</message>
<message name="msgChangeOrderResponse">
 <part name="Acknowledgement" element="po:acknowledgement"/>
</message>
<message name="msgCancelOrderRequest">
  <part name="PONumber" element="po:poNumber"/>
</message>
<message name="msgCancelOrderResponse">
 <part name="Acknowledgement" element="po:acknowledgement"/>
</message>
<message name="msgGetOrderRequest">
  <part name="PONumber" element="po:poNumber"/>
</message>
<message name="msgGetOrderResponse">
 <part name="PurchaseOrder" element="po:purchaseOrder"/>
</message>

<portType name="ptPurchaseOrder-v1">
  <operation name="opSubmitOrder">
    <input message="tns:msgSubmitOrderRequest"/>
    <output message="tns:msgSubmitOrderResponse"/>
  </operation>
  <operation name="opCheckOrderStatus">
    <input message="tns:msgCheckOrderRequest"/>
    <output message="tns:msgCheckOrderResponse"/>
  </operation>
  <operation name="opChangeOrder">
    <input message="tns:msgChangeOrderRequest"/>
    <output message="tns:msgChangeOrderResponse"/>
  </operation>
  <operation name="opCancelOrder">
    <input message="tns:msgCancelOrderRequest"/>
    <output message="tns:msgCancelOrderResponse"/>
  </operation>
  <operation name="opGetOrder">
    <input message="tns:msgGetOrderRequest"/>
    <output message="tns:msgGetOrderResponse"/>
 </operation>
</portType>

<portType name="ptPurchaseOrder-v2">
  <operation name="opSubmitOrder">
    <input message="tns:msgSubmitOrderRequest"/>
    <output message="tns:msgSubmitOrderResponse"/>
  </operation>
  <operation name="opCheckOrderStatus">
    <input message="tns:msgCheckOrderRequest"/>
    <output message="tns:msgCheckOrderResponseErr"/>
  </operation>
```

```
    <operation name="opChangeOrder">
      <input message="tns:msgChangeOrderRequest"/>
      <output message="tns:msgChangeOrderResponse"/>
    </operation>
    <operation name="opCancelOrder">
      <input message="tns:msgCancelOrderRequest"/>
      <output message="tns:msgCancelOrderResponse"/>
    </operation>
    <operation name="opGetOrder">
      <input message="tns:msgGetOrderRequest"/>
      <output message="tns:msgGetOrderResponse"/>
    </operation>
  </portType>

  <portType name="ptPurchaseOrder-v3">
    <operation name="opSubmitOrder">
      <input message="tns:msgSubmitOrderRequest"/>
      <output message="tns:msgSubmitOrderResponse"/>
      <fault message="tns:msgSubmitOrderFault"/>
    </operation>
    <operation name="opCheckOrderStatus">
      <input message="tns:msgCheckOrderRequest"/>
      <output message="tns:msgCheckOrderResponse"/>
    </operation>
    <operation name="opChangeOrder">
      <input message="tns:msgChangeOrderRequest"/>
      <output message="tns:msgChangeOrderResponse"/>
    </operation>
    <operation name="opCancelOrder">
      <input message="tns:msgCancelOrderRequest"/>
      <output message="tns:msgCancelOrderResponse"/>
    </operation>
    <operation name="opGetOrder">
      <input message="tns:msgGetOrderRequest"/>
      <output message="tns:msgGetOrderResponse"/>
    </operation>
  </portType>
  ...
</definitions>
```

Şekil 3.17 İsim uzayında port tipi'nin versiyonlarının gösterilmesi [5]

3.4 WSDL Tanımlamaları ve İleriye Dönük Uyumluluk

"Gevşek (Loose)" strateji desteği ile ilgili seçenekler sınırlı olduğu için bu başlık altında WSDL tanımlamalarındaki değişikliklerin Sıkı (Strict) ve Esnek (Flexible) versiyonlama yaklaşımları ile nasıl birbirine bağlandığı anlatılmaktadır. Şu ana kadar incelediğimiz değişikliklerin çoğu ileriye dönük uyumluluğu sağlamamaktadır. İleriye dönük uyumluluğu sağlayacak bazı tasarım seçenekleri vardır. Bu seçenekler alt başlıklarda örnekler ile açıklanacaktır [16].

3.4.1 İşlemlerin Parçalama Seviyelerini Azaltma

Potansiyel olarak gelecekte daha geniş çap'ta müşterilerle de uyumlu olabilecek gerçek bir WSDL tasarlamanın tek yolu, daha kaba ve daha az iyi olacak şekilde parçalama (granularity) seviyelerini azaltmaktır [16].

Şekil 3.18'de daha iyi-parçalanmış işlemleri tek bir kaba-parçalanmış işlem içerisinde toplanması gösterilmiştir.

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po"
  xmlns:po="http://actioncon.com/schema/po">
  ...
  <portType name="ptPurchaseOrder">
    <operation name="opDoSomething">
      <input message="tns:msgRequest"/>
      <output message="tns:msgResponse"/>
    </operation>
  </portType>
</definitions>
```

Şekil 3.18 İşlemlerin parçalama seviyelerini azaltma [5]

Şekil 3.18'de WSDL seviyesinde ileriye dönük uyumluluk tamamiyle kaba-parçalanmış işlemlerin ihtiyacıyla sonuçlanabilir.

Bu durumda bir işlem geniş çaptaki giriş mesaj verilerini kabul edebilecek şekilde tasarlanmıştır. Böyle bir tasarıma uyum sağlamak için belli başlı mesaj tiplerinin ve özel sembollerin (wildcards) kullanılabileceği kabullenilir.

Alternatif olarak herhangi bir nedenle mevcuttaki daha iyi parçalanmış işlemleri kullanamayan gelecekteki müşterilere uyum sağlamak için daha önce verilen örneklerde olduğu gibi "portType" yapısı üzerine "opDoSomething" işlemi de eklenebilir. Ancak bu tasarım tipi yaygın değildir.

3.4.2 WSDL 2.0 #any Özelliğinin Kullanımı

WSDL tanımlamalarının ileriye dönük uyumluluğu sağlayacak şekilde tasarlamanın diğer bir yoluda WSDL 2.0 ile sağlanan #any değerinin kullanılmasıdır. Bu değer bir işleme XML olarak oluşturulan her tür mesajı alacak (ve cevap verecek) şekilde tasarlanmasına izin verir.

```
<operation name="opSubmitOrder"
  pattern="http://www.w3.org/ns/wsdl/in-out">
  <input messageLabel="In" element="#any"/>
  <output messageLabel="Out" element="po:Acknowledgement"/>
</operation>
```

Şekil 3.19 WSDL 2.0 #any özelliği kullanımı [5]

Şekil 3.20'de WSDL 2.0 #any değerinin nasıl ileriye dönük uyumlu olduğu gösterilmektedir.

```
<description targetNamespace=
  "http://actioncon.com/contract/po/v2"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po/v2"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:soapbind="http://www.w3.org/ns/wsdl/soap">
  <documentation>Version 2.2</documentation>
  <types>
    <xsd:schema>
      <xsd:import namespace="http://actioncon.com/schema/po"
        schemaLocation=
        "http://actioncon.com/schema/purchaseOrder.xsd"/>
    </xsd:schema>
  </types>
  <interface name="ifPurchaseOrder">
    <operation name="opSubmitOrder"
      pattern="http://www.w3.org/2006/01/wsdl/in-out">
      <input messageLabel="In" element="#any"/>
      <output messageLabel="Out" element="po:acknowlegement"/>
    </operation>
```

```
    <operation name="opCheckOrderStatus"
      pattern="http://www.w3.org/2006/01/wsdl/in-out">
      <input messageLabel="In" element="po:poNumber"/>
      <output messageLabel="Out" element="po:status"/>
    </operation>
    <operation name="opChangeOrder"
      pattern="http://www.w3.org/2006/01/wsdl/in-out">
      <input messageLabel="In" element="#any"/>
      <output messageLabel="Out"
        element="po:acknowledgement"/>
    </operation>
    <operation name="opCancelOrder"
      pattern="http://www.w3.org/2006/01/wsdl/in-out">
      <input messageLabel="In" element="po:poNumber"/>
      <output messageLabel="Out"
        element="po:acknowledgement"/>
    </operation>
  </interface>
  ...
</description>
```

Şekil 3.20 İşlemlerin #any özelliğine geçişi [5]

Şekil 3.20'de her iki "opSubmitOrder" ve "opChangeOrder" işlemleri için "input" mesajının öğe özelliklerinin değerleri "po:PurchaseOrder" dan #any'e değiştirilmiştir.

Bu değişiklik mevcut müşterileri etkilememektedir. Gelecekteki müşterilerin iki işlem içinde daha geniş çapta giriş değerleri göndermesine izin vereceği için hem geriye hem de ileriye dönük uyumlu olarak düşünülebilir.

WSDL 2.0 tabanlı yazılımların diğer bir çözüm yöntemi de mevcut işlemlerin sınırlı yapısını azaltma yerine hem giriş hem de çıkış mesajları için #any değerini kullanan yeni, oldukça kaba-parçalanmış şekilde genişletmesidir.

```
<description targetNamespace=
  "http://actioncon.com/contract/po/v2"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://actioncon.com/contract/po/v2"
  xmlns:po="http://actioncon.com/schema/po"
  xmlns:soapbind="http://www.w3.org/ns/wsdl/soap">
  <documentation>Version 2.2</documentation>
  <types>
    <xsd:schema>
      <xsd:import namespace="http://actioncon.com/schema/po"
        schemaLocation=
        "http://actioncon.com/schema/purchaseOrder.xsd"/>
    </xsd:schema>
  </types>
```

```
<interface name="ifPurchaseOrder">
  <operation name="opSubmitOrder"
    pattern="http://www.w3.org/2006/01/wsdl/in-out">
    <input messageLabel="In" element="po:purchaseOrder"/>
    <output messageLabel="Out" element="po:acknowlegement"/>
  </operation>
  <operation name="opCheckOrderStatus"
    pattern="http://www.w3.org/2006/01/wsdl/in-out">
    <input messageLabel="In" element="po:poNumber"/>
    <output messageLabel="Out" element="po:status"/>
  </operation>
  <operation name="opChangeOrder"
    pattern="http://www.w3.org/2006/01/wsdl/in-out">
    <input messageLabel="In" element="po:purchaseOrder"/>
    <output messageLabel="Out"
      element="po:acknowledgement"/>
  </operation>
  <operation name="opCancelOrder"
    pattern="http://www.w3.org/2006/01/wsdl/in-out">
    <input messageLabel="In" element="po:poNumber"/>
    <output messageLabel="Out"
      element="po:acknowledgement"/>
  </operation>
  <operation name="opProcessOrder"
    pattern="http://www.w3.org/2006/01/wsdl/in-out">
    <input messageLabel="In" element="#any"/>
    <output messageLabel="Out" element="#any"/>
  </operation>
</interface>
...
</description>
```

Şekil 3.21 Kaba-parçalanmış bir işlemin #any özelliği ile eklenmesi [5]

Bu tip işlem pratikte her aralıkta giriş mesaj verisini almaya izin veren geniş bir yapı sağlamakta ve her yapıda çıkış mesajı ile cevap verebilmektedir.

4. ŞEMALARIN VERSİYONLANMASI

Web servis sözleşmelerinin hiçbir bölümü temel giriş, çıkış ve hata mesajlarına dayanan XML şema tanımlamaları kadar versiyonlama ile ilgili dikkat gerektirmez.

İlerleyen başlıklarda aşağıdaki dört genel değişiklik tipini inceleyeceğiz:

- Yeni bir şema öğesi ekleme
- Mevcuttaki bir şema öğesini silme
- Mevcuttaki bir şema öğesinin ismini değiştirme
- Mevcut bir şema öğesinin sınırlarını değiştirme

4.1 Temel Terimler ve Konular

Şemaların versiyonlanmasında karşılaşılan temel terimler ve konular alt başlıklar halinde incelenmiştir.

4.1.1 Bir Bileşenin Yeniden Tanımlanması

XML şema açısından bir bileşen (component) şema'nın temel parçasını gösterir. Bileşen öznitelik (attribute) veya öğe'yi (element) gösterecek şekilde sık olarak kullanılır.

4.1.2 İçerik Kümeleri

Aşağıdaki terimler bir mesajın kurallara uygun bileşen bildirimlerini sağlayıp sağlamadığı konusunda ayırım yapmamızı sağlar.

- *Tanımlanmış Küme (Defined Set):* Kendi altında bulunan şemaların bildirimlerinde daha önceden tanımlanan bir mesajın içeriğini gösterir.
- *İzin Verilmiş Küme (Allowed Set):* Mesaj tanımlamasının tipine uyumlu kalacak şekilde bir mesajın maksimum aralığını ifade etmektedir. İzin verilmiş küme, tanımlanmış kümeyi kapsar. Eğer şema özel sembol (wildcard)'leri kullanmaz ise izin verilmiş küme, tanımlanmış küme ile eşdeğerdir. Eğer özel sembol (wildcard)'ler kullanılır ise, izin verilmiş küme'de özel sembollere ilişkin ne gibi genişletmeler gerekiyor ise tanımlanmış kümede de genişletme yapılmalıdır.
- *Bilinmeyen Küme (Unknown Set):* Verilen bir mesaj tanımlamasında tanımlanmış küme ve izin verilmiş küme arasındaki aralığı (varsa) gösterir.
- *Bilinen Küme (Recognized Set):* Bilinen küme her zaman tanımlanmış kümeyi kapsar ve bilinmeyen küme'nin bir kısmını veya hepsini kapsayabilir.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="available" type="xsd:boolean"
        minOccurs="0"/>
      <xsd:any namespace="##any" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##any"/>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.1 İçerik kümelerine örnek [5]

Şekil 4.1'de özel semboller (wildcard) içeren karmaşık tip, tam izin verilmiş küme'nin içerisinde yer alan tanımlanmış küme ve bilinmeyen küme arasındaki farkı anlamamıza yardım eder.

Kırmızı ile gösterilen tüm kodlar tanımlanmış küme'ye aittir. Çünkü daha önceden tanımlanmış yapı ve bildirimler kurmaktadır. Koyu renk ile ifade edilenler ise bilinmeyen küme olarak sınıflandırılmış bir şema'nın parçasını gösterir. Çünkü geniş çap'ta tanımlanmayan verileri kabul etmesine izin vermektedir. Sonuç olarak hem

kırmızı hem de koyu kodlar izin verilmiş küme olarak düşünülür. Çünkü tanımlanmış küme ve bilinmeyen küme'nin birleştirilmesi izin verilebilir mesaj içeriğinin aralığını gösterir.

4.1.3 Versiyonlama ve UPA Kuralı

UPA (Unique Particle Attribution) kuralı XML şema özelliğinin bir parçasıdır. Bu kural her öğe'nin tam olarak şema içerisindeki bir özniteliğe dayandırılması gerekliliğini zorlamaktadır. UPA özellikle özel sembol (wildcard) öğeler ile birlikte seçenekli öğelerin kullanılması anlamına gelmektedir [7].

UPA kuralı özellikle ayrıştırıcı (parser)'ların etkinliğini geliştirmek için kullanılır. UPA kuralı olmadan bir ayrıştırıcı, verilen bir öğenin değerinin seçenekli bir öğe tanımı olup olmadığını veya değerin özel bir sembol ile ilgisinin olup olmadığını anlamak için bir tip tanımı içerisinde arama yapmaya ihtiyaç duyabilir. Mümkün olan her özel sembol değeri için arama yapılmasının gerekliliği birçok işlem zamanının kaybedilmesine sebebiyet vermektedir. UPA kuralı özellikle böyle bir arama kuralı ihtiyacını ortadan kaldırır [7].

Örnek: Şekil 4.2 UPA kuralına uymamaktadır. Çünkü "po2:available" öğesi hem "po2:available" öğesi seçeneği'nin hem de "##other" özel sembolü'nün altında geçerli durumdadır. İki veya daha fazla parçacığın (particles) öğe eşleşmesi tek parça UPA kuralına uymaz.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element ref="po2:available" minOccurs="0"/>
      <xsd:any namespace="##other" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other"/>
  </xsd:complexType>
</xsd:schema>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po2"
  xmlns:po2="http://actioncon.com/schema/po2">
  <xsd:element name="available" type="xsd:boolean"/>
</xsd:schema>
```

Şekil 4.2 UPA kuralını çiğneyen şema [5]

UPA kuralı ileriye ve geriye dönük uyumlulukta seçenekli öğe ve özel sembollerin kullanımına sınırlama getirdiği için şema versiyonlama ile de ilişkilidir.

Temel Örnek: Aynı temel şema içerisinde gelecek olan versiyonlama tekniklerinin herbiri Şekil 4.3'de gösterilmiştir:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po"
  elementFormDefault="qualified"
  version="1.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.3 Temel karmaşık tip örneği [5]

Şekil 4.3'de basit içerikli iki öğeden oluşan temel karmaşık tip gösterilmekdir. Basit Nesne Erişim Protokolü (Simple Object Access Protocol - SOAP) mesajı'nın parçası olarak bu şema'nın bir örneği Şekil 4.4'de gösterilmektedir [2].

```
<LineItem xmlns="http://actioncon.com/schema/po">
  <productID>AY2345</productID>
  <productName>Service Blaster 2000</productName>
</LineItem>
```

Şekil 4.4 Temel şemanın XML doküman örneği [5]

Alt başlıklarda anlatılacak olan versiyonlama senaryolarında "LineItemType" karmaşık tipi oyun ürünü'nün şu an sağlanıp sağlanmayacağını gösteren bir "Boolean" özelliğini gösteren bir "available" öğesi ile genişletilmiştir.

4.2 XML Şema ve WSDL Hedef İsim Uzayları

Temel versiyon değişikliklerinde, XML şema hedef isim uzayı değerinin nasıl değişikliğe uğradığı incelenmektedir.

XML şema, yeni bir hedef isim uzayı gerektiren değişikliğe maruz kaldığı durumda bu değişiklik WSDL tanımlaması içinde yeni hedef isim uzayı ile sonuçlanacak şekilde WSDL seviyesine yayılmaktadır [11].

4.3 Sıkı Şema Versiyonlama

Sıkı (Strict) versiyonlamada herhangi bir değişiklik yeni bir sözleşme versiyonunu gerektirir. Bu nedenle aşağıdaki kısımdaki farklı kod örnekleri hemen hemen aynı sonucu vermektedir.

Yeni Bir Şema Bileşeni (Component) Ekleme:

Şekil 4.5'de temel şema'nın versiyon örneği gösterilmektedir. LineItemType" karmaşık tipine "available" öğe bildirimi eklenmiştir [11].

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po/v2"
  xmlns="http://actioncon.com/schema/po/v2"
  version="2.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="available" type="xsd:boolean"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.5 İsim uzayına "available" öğesinin eklenmesi [5]

Şekil 4.5'de "available" öğesinin eklenmesi uygun olmayan bir değişikliği getirmiştir. Bu versiyonlama yaklaşımının kurallarından biri uyumlu bir değişiklik bile Şekil 4.6'da gösterildiği gibi yeni bir isim uzayı değerini gerektirir.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po/v2"
  xmlns="http://actioncon.com/schema/po/v2"
  version="2.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="available" type="xsd:boolean"
        minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.6 İsim uzayına "available" öğesinin "minOccurs=0" değeri ile eklenmesi [5]

Şekil 4.6'da "available" öğesinin "minOccurs" değeri "0" a ayarlanan (öğenin seçenekli olamasını sağlar) eklenmiştir. Ancak bu ekleme isim uzayıda değişiklik gerekir.

Mevcut Şema Bileşenini (Component) Kaldırma:

Mevcut durumdaki şema'dan bir öğenin silinmesi uyumsuz bir değişikliğe sebebiyet vermektedir [15].

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po/v2"
  xmlns="http://actioncon.com/schema/po/v2"
  version="2.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <!-- productName Removed 09/12 -->
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.7 Mevcut şema öğesini kaldırma [5]

Şekil 4.7'de "ProductName" öğesinin silinmesi şema'nın yeni bir ana versiyonunun çıkılması ile sonuçlanmıştır. Öğe bildiriminin silinmesi gelecekte yorumlanmak amacıyla loglanmıştır.

Mevcut Şema Bileşeninin (Component) Yeniden İsimlendirilmesi:

Şekil 4.8'de gösterildiği gibi varolan bir öğenin yeniden isimlendirilmesi diğer değişiklikler gibi aynı etkiyi oluşturmaktadır.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po/v2"
  xmlns="http://actioncon.com/schema/po/v2"
  version="2.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName2" type="xsd:string"/>
      <xsd:element name="available" type="xsd:boolean"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.8 Mevcut şema öğesinin yeniden isimlendirilmesi [5]

Şekil 4.8'de mevcut şema öğesi olan "productName" elemanının "productName2" olarak yeniden isimlendirilmesi gösterilmektedir.

Mevcut Şema Bileşeninin (Component) Kısıtı (Constraint)'nın Değiştirilmesi:

Sıkı (strict) yaklaşımın kurallarından dolayı öğe kısıtının artırılması veya azaltılmasıyla sonuçlanacak değişiklikler yeni bir versiyon şemasının çıkılmasını zorlamaktadır.

Şekil 4.9'da daha önce gerekli olan öğe'ler seçimli bir öğe olarak güncellenmiştir.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po/v2"
  xmlns="http://actioncon.com/schema/po/v2"
  version="2.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"
        minOccurs="0"/>
      <xsd:element name="productName" type="xsd:string"
        minOccurs="0"/>
      <xsd:element name="available" type="xsd:boolean"
        minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.9 Mevcut şemanın kısıtının değiştirilmesi [5]

4.4 Esnek Şema Versiyonlama

Şema'ya esnek (flexible) versiyonlama uygulanırken geriye yönelik uyumluluk devam etmektedir. Aşağıdaki şekillerde uyumlu ve uyumsuz değişiklileri barındıran teknikler incelenmektedir [13].

Yeni Bir Şema Öğesi Ekleme:

Mevcut şema'ya yeni bir öğe eklenmesi durumunda eğer yeni eklenicek olan öğe zorunlu ise uyumsuz bir değişiklik ve opsiyonel ise de uyumlu bir değişiklik olarak düşünülür [13].

Sıkı (strict) yaklaşımda olduğu gibi yeni zorunlu öğe bildiriminin eklenmesi, yeni ana şema versiyonuna zorlar:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po/v2"
  xmlns="http://actioncon.com/schema/po/v2"
  version="2.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="available" type="xsd:boolean"
        minOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.10 Operasyona zorunlu bir öğenin eklenmesi[5]

Esnek (flexible) versiyonlamada geriye dönük uyumluluğun sağlanması için bizi yeni öğe eklemeye veya öğeleri seçenekli yapmaya teşvik eder. Şekil 4.11'de temel şema'ya opsiyonel "available" öğesi ekleniyor ve ara versiyon değeri artırılarak değişiklik gösteriliyor:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po"
  elementFormDefault="qualified"
  version="1.1">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="available" type="xsd:boolean"
        minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.11 Operasyona opsiyonel bir öğenin eklenmesi [5]

Şekil 4.11'deki mesaj örneğinde öğenin varlığını opsiyonel yapan "minOccurs" değeri "0" yapılarak "available" öğesinin eklenmesi gösterilmektedir.

Şekil 4.12'de ise yeni bir şema öğesinin eklenmesi bize mesajların opsiyonel olarak elimizdeki öğe'ye dahil edilebilmesine izin verir:

```
<LineItem xmlns="http://actioncon.com/schema/po">
  <productID>AY2345</productID>
  <productName>Service Blaster 2000</productName>
  <available>true</available>
</LineItem>
```

Şekil 4.12 Opsiyonel bir öğenin mesaj dokümanı örneği [5]

Yeni Öğe Bildirimlerinin Ayrı Şema'lara Konulması:

Yeni "available" öğesi Şekil 4.13'de gösterildiği gibi temel şema'da referans verilebilen ayrı bir şema olarak bildirilebilir.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po"
  version="1.1">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element ref="available" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po">
  <xsd:element name="available" type="xsd:boolean"/>
</xsd:schema>
```

Şekil 4.13 Ayrı şemadan yeni öğe bildiriminin referans edilmesi [5]

Şekil 4.13'de ilk şema'da geçen "LineItemType" karmaşık tipi ayrı bir şema'da referans edilen bir öğeyi içermektedir.

Mevcut Bir Şema Öğesinin Kaldırılması:

Mevcut bir XML şema tanımlamasından bir öğe bildiriminin kaldırılması yeni bir şema ve WSDL tanımlama versiyonu değişimini gerektirmektedir. Bu değişim uygun olmayan bir değişiklikle sonuçlanır [13].

Mevcut Bir Şema Öğesinin Yeniden Adlandırılması:

Mevcut öğe'nin isminin değiştirilmesi, yeni bir şema hedef isim uzayı ve buna karşı yeni bir şözleşme versiyonu gerektiren uygun olmayan bir değişiklikle sonuçlanır:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po/v2"
  xmlns="http://actioncon.com/schema/po/v2"
  elementFormDefault="qualified"
  version="2.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName2" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.14 Mevcut şema öğesinin yeniden adlandırılması [5]

Bu tip değişiklik Şekil 4.15 de belirtildiği gibi mesaj örneğini uyumsuz duruma sokmaktadır:

```
<LineItem xmlns="http://actioncon.com/schema/po">
  <productID>AY2345</productID>
  <productName2>Service Blaster 2000</productName2>
</LineItem>
```

Şekil 4.15 Öğe ismi değişmiş mesaj dokümanı örneği [5]

Esnek (Flexible) strateji ile şema öğelerinin yenisi ile eskisini korumak için "xsd: choice" grup öğesi kullanarak uygun bir değişiklik ile öğe yeniden isimlendirilebilir:

```
<xsd:complexType name="LineItemType">
  <xsd:sequence>
    <xsd:element name="productID" type="xsd:string"/>
    <xsd:choice>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="productName2" type="xsd:string"/>
    </xsd:choice>
    <xsd:element name="available" type="xsd:boolean"/>
  </xsd:sequence>
</xsd:complexType>
```

Şekil 4.16 "xsd:choice" seçim grubunun kullanımı [5]

Mevcut Bir Şema Öğesinin Kısıtı (Constraint)'ını Değiştirme:

Şekil 4.17'de "productID" öğesinin veri tipi "xsd:integer" olarak değiştirilmiştir. "productName" öğesinin "maxOccurs" değeri "unbounded" olarak ayarlanmıştır [13].

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po/v2"
  xmlns="http://actioncon.com/schema/po/v2"
  elementFormDefault="qualified"
  version="2.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:integer/>
      <xsd:element name="productName" type="xsd:string"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.17 Mevcut şema öğesinin kısıtının değiştirilmesi [5]

"productID" öğesine yapılan değişiklik uygun değildir. "maxOccurs=unbounded" mevcut müşterileri etkilemeyen uygun bir değişiklik olurken, "xsd:integer" orijinal veri tipi "xsd:string" tipinden daha sınırlayıcıdır.

Sonuç olarak Şekil 4.18'de yeni şema'ya gönderilen mesaj uyumlulukta hatalı olacaktır:

```
<LineItem xmlns="http://actioncon.com/schema/po">
  <productID>AY2345</productID>
  <productName>Service Blaster 2000</productName>
  <productName>Service Blaster 2010</productName>
</LineItem>
```

Şekil 4.18 Şema ile uyumsuz olan mesaj örneği [5]

Çoklu "productName" öğeleri kurallara uygundur. Ancak "productID" öğesinin değeri "xsd:integer" tipine uygun olmadığı için artık geçerli değildir.

Şekil 4.19'da tek zorunlu öğenin "productID" olması gerektiği ve "productName" öğesinin seçimli olduğu ifade edilmektedir:

```
<xsd:complexType name="LineItemType">
  <xsd:sequence>
    <xsd:element name="productID" type="xsd:string"/>
    <xsd:element name="productName" type="xsd:string"
      minOccurs="0"/>
    <xsd:element name="available" type="xsd:boolean"
      minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

Şekil 4.19 Öğenin opsiyonel hale getirilmesi [5]

Oyun ürünlerinin birçok defa yeniden isimlendirilmesi (productName) gerektiğini var sayıcak olursak eski isimleri bu durumda kaybedilecektir. Fakat oyunların yeni isimleriyle birlikte eski isimlerinin de tutulması gerekir ise "productName" öğesinin birden fazla olmasına(ki zaten öyledir) izin vermesine ihtiyaç duyulur. Bu nedenle Şekil 4.20'de gösterildiği gibi öğe'nin "maxOccurs" değerine "unbounded" değeri eklenir:

```
<xsd:complexType name="LineItemType">
  <xsd:sequence>
    <xsd:element name="productID" type="xsd:string"/>
    <xsd:element name="productName" type="xsd:string"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="available" type="xsd:boolean"
      minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

Şekil 4.20 "maxOccurs" değerine "unbounded" değerinin verilmesi [5]

4.5 Gevşek (Loose) Şema Versiyonlama

Bu başlık altında XML şema özel sembolleri kullanılarak nasıl farklı kapsamda ileride dönük uyumluluğun sağlanabildiği gösterilmektedir.

Yeni Bir Şema Öğesinin Eklenmesi:

Mesaj tipi değişikliklerini yönetmeye alternatif bir yaklaşım olarak özel sembollerin (wildcard) kullanılmasıdır. Özel semboller, değişik çaptaki öğeleri kabul edebildiği için bu yaklaşım ile web servis sözleşmeleri ileriye ve geriye dönük uyumlu olabilirler [19].

Şekil 4.21'de "xsd:any" ve "xsd:anyAttribue" özel sembol bildirimleri ile temel XML şema genişletilir.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po"
  elementFormDefault="qualified"
  version="1.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:any namespace="##any" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##any"/>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.21 Temel şemanın özel semboller ile genişletilmesi [5]

Bu durumda özel semboller “LineItemType” yapısının sonuna eklenmiştir. Bu tipe yapılan tüm değişiklikler ve genişletmeler özel semboller vasıtasıyla uygulanacaktır. Sonuç olarak başlangıçta tanımlanan kümenin (“productID” ve “productName” den oluşan) büyümesine izin verilmez iken şuan özel semboller aracılığı ile küme büyüyebilir.

Daha önce gösterilen şema için aşağıdaki mesaj tanımlanmış küme (defined set) içerisindeki tiplerle uyumlu olduğu için halen geçerli durumdadır.

```
<LineItem xmlns="http://actioncon.com/schema/po">
  <productID>AY2345</productID>
  <productName>Service Blaster 2000</productName>
</LineItem>
```

Şekil 4.22 Orijinal öğe bildirimlerine uyan XML doküman örneği [5]

Aynı zamanda daha önce bildirilemeyen elemanla genişletilen aynı mesajın geçerli duruma getirilmesi:

```
<LineItem xmlns="http://actioncon.com/schema/po">
  <productID>AY2345</productID>
  <productName>Service Blaster 2000</productName>
  <available>true</available>
</LineItem>
```

Şekil 4.23 Özel sembollerin kullanımı ile şema’ya uyan XML doküman örneği [5]

Mevcut Şema Öğesinin Kaldırılması:

Şekil 4.24'de "productName" öğesi için bildirimi kaldırmaya ihtiyacımız var:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po"
  version="1.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"
        minOccurs="0"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:any namespace="##any" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other"/>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.24 Bir öğenin kaldırılması [5]

"productName" öğe bildiriminin hemen arkasından özel sembol (wildcard) takip ettiği için Şekil 4.25'de kırmızı ile gösterilen yeri kaldırabiliriz:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po"
  version="1.1">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"
        minOccurs="0"/>
      <!-- productName Removed 09/12 -->
      <xsd:any namespace="##any" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other"/>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.25 Öğe bildirimi kaldırıldıktan sonra "LineItemType" yapısı [5]

"productName" öğe bildiriminin kaldırılması uyumlu bir değişikliktir. Çünkü özel sembol (wildcard) ile şu an yeni müşterilere önceden gerekli olan öğeyi sağlamazken mevcut müşterilerden gelen mesajlardaki "productName" öğesini kabul etmeye devam etmektedir.

Örnek (özel sembol öncesi olmayan bir öğe'nin kaldırılması): Direk olarak arkasından özel sembol bildirimi olmayan bir öğe bildiriminin kaldırılması daha zordur. Şekil 4.26'da "productName" öğe bildiriminin önündeki olan "productID" öğesini kaldırmak istiyoruz:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po"
  version="1.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:any namespace="##any" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other"/>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.26 "prodcutID" öğe bildirimi şemadan kaldırılacak şekilde tanımlanması [5]

Bu durumda öğenin bir özel sembol (wildcard) ile değiştirilmesi UPA kuralını çiğneme olarak sonuçlanacaktır.

Alternatif olarak "productID" öğe bildirimini muhafaza edebilir ve basitçe Şekil 4.27'de gösterildiği gibi tipini "xsd:anyType" olarak değiştirerek ve "minOccurs" değerine "0" ekleyerek değişiklik yapabiliriz.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po"
  version="1.1">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:anyType"
        minOccurs="0"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:any namespace="##any" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other"/>
  </xsd:complexType>
</xsd:schema>
```

Şekil 4.27 "productID" öğesinin tipi "xsd:anyType" olarak değiştirilmesi [5]

Bu teknik gerçekte fiziksel olarak “productID” öğe bildirimini kaldırmazken sınırlama değeri ile bir özel sembol (wildcard)’e benzeyecek şekile gelir. Ayrıca etkisini uyumlu bir değişik olacak şekilde sınırlar.

5. GELİŞMİŞ VERSİYONLAMA

Gelişmiş versiyonlama başlığı altında WSDL, XML şema ve WS-İlke (Policy) ile ilgili farklı versiyonlama problemlerinin çözümü ile ilgili birçok kılavuz, teknik ve düşüncelerle anlatılmaktadır.

5.1 Versiyonlama Politikaları

İlke (policy)'ler, şema'lar kadar servis sözleşmelerinin bir parçasıdır. Bu nedenle WS-İlke ifadeler ile çalışırken versiyonlama yapılmaya karar verildiğinde alternatifleri bilmek önemlidir. Çünkü bir versiyon ilkesinden diğerine uygun olmayan veya geçici yapılan geçişler önemli sorunlara sebep olabilir [19].

WS-İlke dili sade bir dil'in öğelerinden oluşsa bile ilke'lerin versiyonlanmasına gelindiğinde çok avantajlı bazı özellikler sağlamaktadır. Şekil 5.1'de versiyon 1.0 olarak temel bir örnek oluşturulmuştur.

```
<!-- Purchase Order Policy v1.0 -->
<wsp:Policy>
  <wsa:UsingAddressing/>
</wsp:Policy>
```

Şekil 5.1 İlke (policy) ifadesinin 1.0 versiyonu [5]

Şekil 5.2'deki gibi bu ilke'ye uygun olmayan bir değişiklik yapılması durumunda ana versiyon numarası arttırılacaktır.

```
<!-- Purchase Order Policy v2.0 -->
<wsp:Policy>
  <wsa:UsingAddressing/>
  <wsrmp:RMAssertion/>
</wsp:Policy>
```

Şekil 5.2 İlke ifadesine uygun olmayan bir değişikliğin yapılması [5]

İlke ifadesine uygun bir değişiklik yapılması durumunda sadece ara versiyon numarası arttırılır.

5.1.1 Alternatifler

Seçimli ilke alternatifleri ile geçmişe dönük uyumluluk sağlanabilir. Verilen bir web servisi sözleşmesi Şekil 5.1'deki temel ilke ifadesi ile başlayabilir. Mevcut ilke ifadesi değiştirmeye karar verildiğinde veya yeni bir ilke ifadesine gerek olduğunda "wsp:ExactlyOne" alternatif yapısı kullanılabilir.

```
<wsp:Policy>
  <wsp:ExactlyOne>

  <!-- Purchase Order Policy v1.0 -->
    <wsp:All>
      <wsa:UsingAddressing/>
    </wsp:All>

  <!-- Purchase Order Policy v1.1 -->
    <wsp:All>
      <wsa:UsingAddressing/>
      <wsrmp:RMAssertion/>
    </wsp:All>

  </wsp:ExactlyOne>
</wsp:Policy>
```

Şekil 5.3 Alternatif içeren ilke ifadesinin 1.1 versiyonu [5]

İlave alternatifler olarak yeni ilke versiyonları eklemeye devam edilebilir. Her yeni ilke ifadesi uyumlu bir değişiklik olarak düşünülür. Çünkü alternatif yapı geriye dönük uyumluluğu sağlamaya devam eder.

5.1.2 Seçimli İfadeler

"wsp:Optional" seçimli bir ilke ekleyerek bir web servis sözleşmesini genişletmede uygun bir yöntem sağlar. Bu yöntem uyumlu bir değişiklik sağlar.

Şekil 5.4'de "wsrmp:RMAssertion" ilke ifadesi mevcut ilke ifadesi içerisine eklenmiştir. Çünkü "wsp:Optional=true" ifadesi etiketlendiği için, eski müşteriler tarafından önemsenmeyecek olan seçimli fonksiyonellik sağlanır ve bu şekilde geriye dönük uyumluluk sağlanır.

```
<!-- Purchase Order Policy v1.1 -->
<wsp:Policy>
  <wsa:UsingAddressing/>
  <wsrmp:RMAssertion wsp:Optional="true" />
</wsp:Policy>
```

Şekil 5.4 Seçimli ifadeyle genişletilmiş 1.1 versiyonlu ilke ifadesi [5]

"wsp:Optional" ifadesi basitleştirilmiş ilke gösterimine izin veren bir yoldur. Seçimli bir ifade içeren ilke ifadesi ayıca "wsp:Optional" kullanımı ve ilgisiz olan alternatif bir sözleşme oluşturma arasında ayrım yapan bir ilke alternatifi olarak da açıklanabilir.

5.1.3 İhmal Edilebilir İfadeler

"wsp:Optional" ve "wsp:Ignorable" arasındaki temel farklar:

Birincisini müşteriler kullanmayı seçsin veya seçmesin mecburi olmayan bir işlem seçeneğini göstermektedir. İkincisi ise müşteri tarafından onaylamış olup olmadığına bakılmaksızın yürütülecek işlemi gösterir.

```
<!-- Purchase Order Policy v2.0 -->
<wsp:Policy>
  <wsa:UsingAddressing/>
  <custom:LogMessage wsp:Ignorable/>
</wsp:Policy>
```

Şekil 5.5 İhmal edilebilir ifadeler [5]

Şekil 5.5'de uygun olmayan bir değişiklik olarak düşünülen ihmal edilebilir ifadeyle genişletilen 2.0 versiyonlu bir ilke ifade edilmektedir.

Örnek: Servisinizi 10 farklı müşterinin kullanıdığını düşünecek olursak, bazı veri tiplerini log eden bir "logging" fonksiyonu eklediğinizi varsayalım. "Logging" olduğunu gösteren fakat müşterinin şu andaki servisle nasıl etkileştiğini değiştirmeyen mevcut ilke ifadesi içerisine ihmal edilebilir bir ifade yerleştirmiş olunur.

5.1.4 Ana Versiyon Değişikliklerinin WSDL Tanımlamalarına Yayılması

XML şema öğelerinde uygun olmayan değişiklikler yapılması durumunda uygun olmayan WS-İlke değişiklikleri de yeni bir sözleşme versiyonunun yapılmasını gerektirir. Bu, uygun olmayan değişikliği tetikleyen ilke ifadesinin nerede ve nasıl ifade edildiğine bağlı olarak değişik sonuçlar doğabilir [15].

Kod İçerisine Eklenmiş (Attached Inline): Uyumlu olmayan bir değişiklik yapılması yeni bir WSDL hedef isim uzayında yeni bir sözleşme versiyonu yapılmasını gerektirir [19].

Bağımsız Policy Tanımlaması (Independent Policy Definition): İlke'ler ayrı tanımlama dosyalarına ayrıldığında çoklu WSDL dokümanları boyunca paylaşılabilir. Bu yaklaşım genelde ilke merkezileştirme (Centralization) düzenine uygularken yapılır [19].

Bağımsız ilke ifadeleri kendi hedef isim uzayına ait olmayabilir. Düzenli ilke versiyon numaralarının takibi ihtiyacı gerebilir. Harici olarak bir veya daha fazla WSDL tanımlamasına eklenen ilke'lerde uyumlu olmayan bir değişiklik yapılması durumunda tüm sözleşme versiyonunu artırmak ve yeni bir WSDL hedef isim uzayına zorlamak için her WSDL dokümanına yayılma ihtiyacını doğurur.

Bir WSDL Tanımlaması İçerisindeki Gömülü İlke Tanımlaması: Bağımsız bir ilke tanımlaması oluşturmada diğer bir alternatif de ilke ifadelerini ayrı WSDL tanımlamaları içerisine yerleştirmektir. Bu yaklaşım aynı zamanda ilke ifadeleri

içeren WSDL tanımlamalarının her bir "targetNamespace" ismi olarak ilke'lerin kendi hedef isim uzaylarına atama faydasını sağlar.

İlke'lerde uyumlu olmayan değişiklik yapılması durumunda ilke tanımlamalarının hedef isim uzayında da değişiklik yapılmalıdır. Bu da ilke tanımlamasını kullanan her WSDL tanımlamasını etkiler. Aynı zamanda uyumlu olmayan değişikliğin WSDL tanımlamasının kendi hedef isim uzayına da yayılması ihtiyacını gerektirir.

5.2 "İhmal Edilmeyen" Öğelerin Tanımlanması

Özel sembol (wildcard)'lerin kullanımı, geniş çapta bilinmeyen öğelere izin veren ve geçerli bir veri olarak düşünülen bir tip tanımlaması içerisinde genişletilebilir noktalar kurulmasına izin verir.

Genişletilebilir noktalarla bir web servis sözleşmesi, içerisindeki servis mantığı bilinmeyen mesaj içeriğinin mümkün olabilecek tüm değişimleri işleyebilecek şekilde veya basitçe ihmal et seçeneği ile tasarlanabilir.

Bu seçenek web servis sözleşme versiyonlamasını esnek yapar. Girilen bir öğenin müşteriler tarafından anlaşılması için etikete izin veren SOAP dilinin bir özelliğini alma ihtiyacımız olacaktır.

Aşağıdaki alt başlıklarda "ihmal edilemez" etiketleme içeriği için iki teknik incelenmektedir.

5.2.1 "mustUnderstand" Kullanımı

Şekil 5.6'deki mesajın "mustUnderstand" ismini bilinmeyen öğelerin eklemesine izin verecek şekilde şemamıza eklenebilir.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po">
  <xsd:element name="lineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:any namespace="##any" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="mustUnderstand" type="xsd:boolean"/>
    <xsd:anyAttribute namespace="##any"/>
  </xsd:complexType>
</xsd:schema>
```

Şekil 5.6 “mustUnderstand" kullanımı [5]

Şekil 5.6’da “mustUnderstand” için tanımlaması kural olan “xsd:attribute” bildirimi yapılmaktadır. Diğer isimler isteğe bağlı olarak “xsd:anyAttribute” bildirimi olarak özel sembollere karşı geçerli kılınan herhangi öğeye eklenebilir.

Şekil 5.7’deki “mustUnderstand” isminin avantajını kullanan bir mesaj örneği ifade edilmektedir.

```
<LineItem xmlns="http://actioncon.com/schema/po">
  <productID>AY2345</productID>
  <productName>Service Blaster 2000</productName>
  <available mustUnderstand="true">
    true
  </available>
</LineItem>
```

Şekil 5.7 “mustUnderstand=true” ayarlanan “available” öğesi [5]

5.2.2 “soap:mustUnderstand” Kullanımı

Diğer bir yaklaşım da bilinmeyen öğeyi mesaj gövdesinden ayırmak ve SOAP header’ı içerisine yerleştirmektir.

```
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns="http://actioncon.com/schema/po">
  <soap:Header>
    <available soap:mustUnderstand="true">
      true
    </available>
  </soap:Header>
  <soap:Body>
    <lineItem>
      <productID>AY2345</productID>
      <productName>Service Blaster 2000</productName>
    </lineItem>
  </soap:Body>
</soap:Envelope>
```

Şekil 5.8 “soap:mustUnderstand” ismine izin veren “soap:header” yapısı [5]

Bu mesaj yapısını etkinleştirmek için “available” öğesinin WSDL yapısı içerisinde “SOAP:header” ına sınırlandırılması ihtiyacı vardır:

```
<binding name="bdPO-SOAP12HTTP" type="abs:ptPurchaseOrder">
<soap12:binding style="document"
  transport="http://schemas.xmlsoap.org/soap/http"/>
  operation name="opSubmitOrder">
  <soap12:operation soapAction=
    "http://actioncon.com/submitOrder/request"
    soapActionRequired="true" required="true"/>
    <soap:header message="po:available" use="literal"/>
    <input><soap12:body use="literal"/></input>
    <output><soap12:body use="literal"/></output>
  </operation>
  ...
</binding>
```

Şekil 5.9 “soap:header” içerisinde “available” öğesinin kullanımı [5]

”available” ve herhangi diğer genişletme öğeleri “soap:mustUnderstand” ismi ile birlikte gönderilebilir. Ancak bu yaklaşım daha karmaşık mesaj tasarımını etkiler.

5.3 Sonlandırma Bilgisinin Belirtilmesi

Bir servis versiyonundan diğerine pürüzsüz geçişi destekleyen bir metot daha eski servis versiyonu hakkında standartlaşmış bir sonlandırma bilgisini sağlamaktır [17].

Google ve Yahoo gibi organizasyonlar tarafından sağlanan herkesin kullandığı web servisler genelde 3 ay veya daha az sürede tedavülden çıkarılmaktadır. Tedavülden çıkarma için gerekli zaman miktarı aşağıdakiler dahil birçok faktöre bağlıdır:

- Servisi şuan kullanan toplam müşteri sayısı
- Müşteri sorumlusu veya sahibi geçişten ne kadar etkilenecek
- Müşterilerin yeni bir servis versiyonuna geçişinin tahmini maliyeti
- Geçişten kaynaklanacak tahmini risk ve bozulmalar

Bir web servisinin sonlandırılması ile ilgili ileride planlanması gereken diğer bir konu servisin belli bir zaman aralığında ne zaman etkinleştirileceğidir. Örneğin ana sistem geliştirme aşamasında iken ortama geçici bir erişim noktası sağlayan bir servis olabilir. Her iki durumda da sonlandırma tarihi belirlendiğinde bu bilgi web servis sözleşmesine eklenmelidir. En kolay ve en ilkel yol "dokumentation" öğesini kullanarak WSDL tanımlamasını isimlendirmektir:

```
<definitions name="PurchaseOrder" targetNamespace=
  "http://actioncon.com/contract/po" ...>
  <documentation>
    This service is scheduled for termination on Jan 31, 2010.
  </documentation>
  ...
</definitions>
```

Şekil 5.10 "Documentation" yapısı eklenen sonlandırma detayları [5]

Bu basit yaklaşım olmasına rağmen sonlandırma tarih'inin yorumlaması sadece insan ile sınırlanır. Bu bilginin servis müşterisi tarafından da erişilebilir olması için bunu özel bir ilke ifadesi kullanarak açıklayabiliriz.

5.3.1 İhmal Edilebilir Sonlandırma İfadeleri

Sonlandırma ihmal edilebilir bir ifade ile açıklanabilir. Şekil 5.11'de gösterildiği gibi müşteriye işleneceği veya işlenmeyeceği kararını veren bilgi sağlanmaktadır:

```
<wsp:Policy ...>
  <custom:termination wsp:Ignorable="true">
    Jan-31-2010
  </custom:termination>
</wsp:Policy>
```

Şekil 5.11 İhmal edilebilir ifade ile açıklanan sonlandırma bilgisi [5]

İhmal edilebilir sonlandırma ifadesi ayrıca Şekil 5.12'de gösterildiği gibi sadece eski versiyonu temsil edenlere uygulanan bir ilke alternatif ile de ilişkilendirilebilir:

```
<wsp:Policy ...>
  <wsp:ExactlyOne>
    <wsp:All>
      <wsa:UsingAddressing/>
      <custom:termination wsp:Ignorable="true">
        Jan-31-2010
      </custom:termination>
    </wsp:All>
    <wsp:All>
      <wsa:UsingAddressing/>
      <wsrmp:RMAssertion/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Şekil 5.12 "custom:termination" kullanımı [5]

İhmal edilebilir ifadelerin kullanılması ile ilgili riskler bulunmaktadır. İfadeyi işlemeye tasarlanmamış müşteriler onu ihmal edebilir ve bu nedenle sonlandırma tarihinden sonra da servisi kullanmayı deneyebilir.

5.3.2 Zorunlu Sonlandırma İfadeleri

Kontrollü bir ortamda ve özellikle sıfırdan başlayarak bir servis envanteri bina ederken tüm web servis sözleşmelerinin zorunlu sonlandırma ifadelerini kullanmayı gerektirecek güçlü bir durum oluşturulabilir.

Şekil 5.13'de ihmal edilemez sonlandırma ifadesi gösterilmektedir:

```
<wsp:Policy ...>
  <custom:termination wsp:Ignorable="false">
    Jan-31-2010
  </custom:termination>
</wsp:Policy>
```

Şekil 5.13 İhmal edilemeyen sonlandırma ifadesi [5]

Şekil 5.13'de "wsp:Ignorable" ismi varsayılan olarak "false" olduğu için ihmal edilebilir.

Tüm web servis sözleşmelerinde sonlandırma ifadelerinin kullanılması gerekliliği aşağıdakileri kazandırır [14]:

- Müşteri ile işlem yapan tüm servisler bilgilendirme ve süreç sonlandırma ifadelerini kullanacak şekilde tasarlanmalıdır.
- Sonlandırma ifadesinin kullanılması gerekliliği herhangi bir müşterinin programsal olarak bağımlı olduğu servisin geçerli olup olmadığını elde etme imkanını sağlar. Bu bilginin karşılığında müşteri kendi sorumlusuna bir uyarı gönderebilir veya bunu dikkate almaz.
- Sonlandırma ifadesi ile birlikte süresi dolan bir web servis sözleşmesi yayında kalabilir. Bu, müşteriye servisle veri alışverişi yapmadan önce ifadeyi kontrol etme imkanı sağlar. Eğer sonlandırma tarihi şu andaki tarihten sonra ise müşteri istisnai bir durum olmadan basitçe servisi çalıştırma sırasında bu veri alışverişini sonlandırabilir.

5.4 Bilinmeyen İçerikle Çalışma

Hem ileriye ve hem de geriye dönük uyumluluğu destekleyen özel sembollerin kullanımına dayanmaktadır.

Bu durumda bilinmeyen içerikle ilgili mesajı alanın nasıl davranacağı ile ilgili iki seçenek vardır:

- İçeriği ihmal edebilir ve işlemlerini tanımlanmış küme (defined set) içeriğindekilerle işleyerek sınırlar.
- Bilinmeyen içeriği özel bir alt kümeye yerleşecek şekilde tasarlanabilir. Bu durumda içeriği tanımlanmış küme (defined set) bu alt kümeye uygun olarak işlemini yapacaktır ve diğer her şeyi de ihmal edecektir.

Her iki durumda da tanımadığı veya tanımak istemediği içeriğe ne yapması gerektiğine karar vermesi gerekecektir. Varsayılan davranış, bilinmeyen içeriği ihmal ederken onunla ilgili bir şey yapmamaktır. Bu, mesaj alıcının verilen mesajdan ne ihtiyaç duyduğu ile ilgili katı sınırlamalarının olduğu durumlarda uygun olabilir.

Ancak bu içeriğin gerçekte anlamlı duruma dönüşebileceği durumlar da vardır. Bu durumlarda servisler ve müşteri programları "kabul et ve sakla" (accept and retain) tekniğini izleyecek şekilde tasarlanabilir.

5.4.1 Bilinmeyen İçeriğin Kabul Edilmesi ve Saklanması

Bilinmeyen küme (unknown set), özel sembollerin kullanılmasını kabul edilebilecek bilinmeyen içerikleri temsil ederken, tanımlanmış küme (defined det) daha önce tanımlı şema öğelerine karşılık gelen içeriği temsil eder.

"Kabul et ve sakla" yaklaşımını izlerken tanımlanmış (defined) ve bilinen (recognized) içeriği kabul ettikten sonra kalan bilinmeyen içeriği atmasın diye müşteri veya servisi genişletebiliriz. Genişletme yerine gelecekte referans olacak bir yerde saklamak daha iyi bir fikirdir.

Bu ekstra veriyi saklamanın neden iyi bir fikir olduğu ile ilgili birçok neden vardır:

- Veriler bilinen (recognized) içeriğe anlamlar katan ilave bilgiler sağlayabilir.
- Veriler gelecekte anlamlı hale gelebilir veya çözümün diğer parçaları için kullanışlı olabilir.

- Veriler değerlendirilebilir bir "logging referans" hizmeti görebilir.
- Verileri yakalamada tek imkanı bu olabilir.

Özellikle bilinen küme (Recognized Set) nin yanında tüm bilgileri bir mesajda toplayabilen yapı web servisin değerliliğini arttırabilir.

```
<lineItem>
  <productID>AY2345</productID>
  <productName>Service Blaster 2000</productName>
  <available>true</available>
</lineItem>
```

Şekil 5.14 "LineItem" yapısını genişleten "available" öğesi [5]

Şekil 5.14'de temel "LineItem" öğesinin "available" öğesi ile genişletildiği mesaj ifade edilmiştir.

Tablo 5.1 "Extensions (Uzantı)" sütunu olan "Line Item" tablosu [5]

Ürün Kimliği	**Ürün Adı**	**Uzantı**
AY2345	Service Blaster 2000	<available> true </available>

Tablo 5.1'de olduğu gibi ekstra verinin saklanmasıyla servis sonradan tablo içerisine bu öğeleri temsil eden (örnek: "available") ilave sütunlar ekleyerek "understand" genişletme öğeleri ile genişletilebilir.

Tablo 5.2 aslında bilinen küme (recognized set)'nin bir parçası olarak "available" öğesini dahil eden bir web servisinin 2. versiyon uygulamasını göstermektedir.

Tablo 5.2 "LineItem" tablosuna "Available (Uygun)" sütununun eklenmesi [5]

Ürün Kimliği	**Ürün Adı**	**Uygun**	**Uzantı**
AY2345	Service Blaster 2000	true	

Eğer bunun da saklanması gerekebileceğini düşünüyorsanız bunun yerine Tablo 5.3 tasarım düşünülebilir.

Tablo 5.3'de bazı veriler çoklu olsa bile orijinal XML hiyerarşisini ve bu yapının sahip olabileceği tüm bilgileri muhafaza etmektedir.

Tablo 5.3 Tüm XML dokümanı içeren yapıya örnek [5]

Ürün Kimliği	**Ürün Adı**	**İşlenmemiş**
AY2345	Service Blaster 2000	<lineItem> <productID> AY2345 </productID> <productName> Service Blaster 2000 </productName> <available> true </available> </lineItem>

5.5 Versiyonlama Mesaj Örnekleri

Aynı tablo içerisinde veri yenileyen birden fazla program varsa, bu veride bütünlük sağlama riski ortaya çıkmaktadır.

Aşağıdaki senaryoda aynı servis vasıtasıyla aynı veriye erişen iki müşteri bulunmaktadır.

1. Müşteri'nin aldığı ilk mesaj:

```
<lineItem xmlns="http://actioncon.com/schema/po/v1"
  version="1.0">
  <productID>AY2345</productID>
  <productName>Service Blaster 2000</productName>
</lineItem>
```

Şekil 5.15 Versiyon="1.0" örneği [5]

Şekil 5.15'de müşteri bu mesajdaki veriyi yenileyebilir fakat bu isimle değiştirmeye izin verilmez.

Aynı veriyi alan 2. Müşteri'nin aldığı mesaj:

```
<lineItem xmlns="http://actioncon.com/schema/po/v1"
  version="1.0">
  <productID>AY2345</productID>
  <productName>Service Blaster 2000</productName>
</lineItem>
```

Şekil 5.16 "lineItem" yapısı "versiyon=1.0" [5]

Şekil 5.17'de 1. Müşteri "available" öğesi ekleyerek veriyi değiştirmektedir:

```
<lineItem xmlns="http://actioncon.com/schema/po/v1"
  version="1.0">
  <productID>AY2345</productID>
  <productName>Service Blaster 2000</productName>
  <available>true</available>
</lineItem>
```

Şekil 5.17 "available" öğesi ile verinin yenilenmesi [5]

Bu değişiklik yapıldıktan sonra 2. Müşteri yenilenmiş bir "productName" değeri içeren kendi değişikliğini yapar.

```
<lineItem xmlns="http://actioncon.com/schema/po/v1"
  version="1.0">
  <productID>AY2345</productID>
  <productName>Service Blaster 2006</productName>
</lineItem>
```

Şekil 5.18 Yenilenmiş "productName" değerini içeren ifade [5]

Bu sırada servis 1. Müşteri'den gelen yenilemeyi işlemiştir ve bu verinin ilgili versiyon numarası "1.1" arttırılmıştır. Bu nedenle 2. Müşteri yenilenmiş veri alındığında servis versiyon numaranın "1.0" olduğunun farkına varır ve çarpışma olmaması için mesajı red eder.

Yenilemenin red edilmesine cevap olarak 2.Müşteri şu anda versiyon="1.1" eklenmiş verinin yeni versiyonunu alır:

```
<lineItem xmlns="http://actioncon.com/schema/po/v1"
  version="1.1">
  <productID>AY2345</productID>
  <productName>Service Blaster 2000</productName>
  <available>true</available>
</lineItem>
```

Şekil 5.19 "Versiyon=1.1" eklenmiş verinin yeni versiyonunun alınması [5]

En son veri ayarlanmasıyla 2. Müşteri "productName" öğesindeki değişikliği yapar ve yenileme için yeniden gönderir:

```
<lineItem xmlns="http://actioncon.com/schema/po/v1"
  version="1.1">
  <productID>AY2345</productID>
  <productName>Service Blaster 2006</productName>
  <available>true</available>
</lineItem>
```

Şekil 5.20 "productName" öğesindeki değişiklik[5]

Servis bu veri kümesi için kaydındaki "1.1" numarası ile alınan mesajdaki "1.1" versiyon numarası ile eşleştirir ve yenilemeyi kabul eder. Bunun sonucunda sonraki mesaj değişimi için versiyon numarası "1.2" olarak artırılır.

5.6 Versiyonlama Stratejisinin Uygulanması

Bu stratejiler başlangıç noktası olarak düşünülebilir ve kendi versiyonlama (ve yönetme) gerekliliklerinize uyacak şekilde uygulanabilir.

5.6.1 Şema Versiyonlama İçin Özel Stratejiler

Yeni İsim Uzaylarında Uyumlu ve Uyumsuz Değişiklikler:

Bu strateji aşağıdaki kurallara dayanır:

- Uyumlu bir değişiklik olduğunda mevcut XML şema öğeleri kendi mevcut isim uzaylarında kalırlar [17].
- Uygun bir XML şema öğesi eklendiğinde bu öğe yeni bir isim uzayı ile ilişkilendirilir.
- Herhangi tip'e uygun olmayan değişiklik yapıldığında mevcut ve yeni öğeler için yeni bir isim uzayı oluşturulur.

Mevcut İsim Uzaylarındaki Uyumlu Değişiklikler, Yeni İsim Uzaylarındaki Uyumsuz Değişiklikler:

Bu strateji aşağıdaki kurallara dayanır [11]:

- Uygun olan bir değişiklik olduğunda mevcut öğeler kendi isim uzaylarında kalırlar.
- Uygun olan bir şema öğesi eklendiğinde bu öğe mevcut isim uzayı ile ilişkilendirilir.
- Uygun olmayan herhangi bir değişiklik yapıldığında mevcut öğeler ve yeni öğeler için yeni bir isim uzayı oluşturulur.

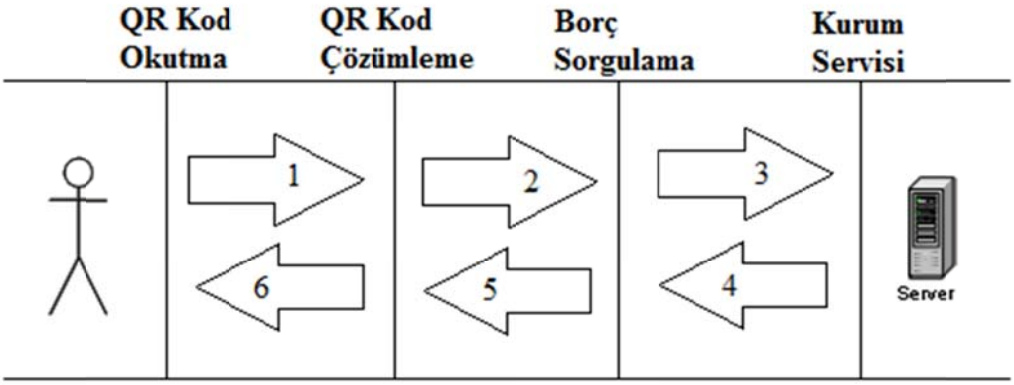
Bu kuralları güçlendirmek için aynı isim uzayına yerleştirilmiş herhangi bir uygun şema sağlamak için bir tasarım standart'ına ve uygun olmayan bir değişiklik olduğunda mesaj tanımının tüm şeması için oluşturulmuş yeni bir hedef isim uzayı oluşmasını garanti altına alan diğer bir tasarım standart'ına ihtiyaç vardır. Aynı zamanda özel sembollerin namespace="##any" ayarlamasını kullanması gerekir. Bu, mevcut isim uzayına yerleştirilecek yeni uygun şema öğelerine izin verir fakat onların gerekli olduğunda farklı isim uzaylarında olmasına da izin verir [17].

6. UYGULAMA

Uygulama QR (Quick Response) kod ve barkod ile fatura ödemeye olanak sağlamaktadır. Amacımız versiyon çıkmadan her kurumun QR kod ve barkod içeriklerine hızlı entegre olan bir servis tasarlamaktır.

Fatura üzerinde yer alan QR kod veya barkod içerisinde abone numarası, son ödeme tarihi, fatura numarası, fatura tutarı ve açık faturaları benzersiz (unique)'leştirmeye yarayan ekstra bilgiler bulunmaktadır (bu bilgiler kuruma göre değişiklik gösterebileceği gibi son ödeme tarihi, fatura no bilgiside olabilir).

Şekil 6.1'de görüldüğü üzere barkod veya QR kod içerisinde saklanan bilgiler çözümlendikten sonra kurumun servisinin borç sorgulama operasyonuna gitmektedir. Kurumda ilgili barkod içeriğine ait borç var ise ekrana yansıtılmaktadır.



Şekil 6.1 Uygulama detayları

Örnek: <BarkodIcerik>10050720137100224325 78</BarkodIcerik>

Fatura üzerindeki QR kod veya barkod okutulduktan sonra içerisinde saklanan bilgiler "BarkodIcerik" öğesine atanmaktadır.

<xsd:element name=" BarkodIcerik" type="xsd:string"/>

“BarkodIcerik” öğesinin data türü belirtilerek tanımlama yapılmıştır. “minOccurs” değeri atanmadığı için default olarak bu değer “1” olarak kabul edilmektedir. Yani bu bilgi opsiyonel değil zorunlu bir bilgidir. Bu tanımlama mesaj tanımını hassas sınırlama derecesi ile temsil eder. Programa ait mesajların geçerli olabilmesi için bu sınırlamalarla uyumlu olmaları zorunludur.

QR kod veya barkod içeriği çözümlendikten sonra kaba kısıtlama detayları ile tanımlanan “AboneBilgi” öğesinde bilgiler atanır. Mesaj’ın komple yapıya uyumlu olmasına gerek yoktur. Çünkü içerik son derece kaba olarak tanımlanmıştır.

```
<AboneBilgi>
   <AboneBilgiDetay>
      <AboneBilgiIcerik>AboneNumarası</AboneBilgiIcerik>
      <AboneBilgiDeger>7100224325</AboneBilgiDeger>
   </AboneBilgiDetay>
   <AboneBilgiDetay>
      <AboneBilgiIcerik>FaturaNumarası</AboneBilgiIcerik>
      <AboneBilgiDeger>63521</AboneBilgiDeger>
   </AboneBilgiDetay>
   <AboneBilgiDetay>
      <AboneBilgiIcerik>SonOdemeTarihi</AboneBilgiIcerik>
      <AboneBilgiDeger>2013-07-05</AboneBilgiDeger>
   </AboneBilgiDetay>
   <AboneBilgiDetay>
      <AboneBilgiIcerik>EkstraBarkodIcerik</AboneBilgiIcerik>
      <AboneBilgiDeger>349878</AboneBilgiDeger>
   </AboneBilgiDetay>
</AboneBilgi>
```

“AboneBilgi” öğesine gelen bilgiler aracılığı ile kurumun borç sorgulama servisine entegrasyon sağlanarak abone’ye ait borç ekrana yansıtılır.

Şekil 6.2'de parakod ile fatura ödeme akışı anlatılmaktadır.

Şekil 6.2 QR kod ile fatura ödeme akışı [20]

Uygulama ileriye ve geriye uyumluluğu tam desteklemektedir. "BarkodIçerik" tanımı son derece kaba kısıtlama detayları ile oluşturulur. İçeriğinin komple yapıya uyumlu olmasına gerek yoktur. Servis'in taslağı gelecekte öngörülmeyen değişiklikleri de kapsayacak şekilde tasarlanmıştır.

7. SONUÇ VE ÖNERİLER

Bu çalışmada web servislerin temel tanım ve bileşenlerinin genel açıklaması yapıldı. Özellikle servis odaklı mimarilerde, web servislerin versiyonlanması üzerinde durularak versiyonlamanın nasıl yapıldığı örnekler ile anlatıldı.

Versiyonlamada hassas ve kaba sınırların kullanımı, sürüm oluşturma adımları ve WSDL tanımlama versiyonu açıklandı. Versiyon revizyonu ve uyumluluğu maddeler halinde aktarıldı. Majör ve minör versiyonun kullanımı, versiyon oluşturma stratejileri, avantaj ve dezavantajları ile birlikte açıklandı.

Uygulamalı olarak versiyonlamayı gösterebilmek için, örnek bir senaryo tasarlanarak sürüm oluşturma adımlarındaki süreçler modellenerek aktarıldı. WSDL 2.0 #any değerinin kullanımı açıklandı. Örnek bir senaryo üzerinden şemaların versiyonlanması aktarıldı. Sıkı ve esnek versiyonlama detaylandırıldı.

Bu tez çalışmasında web servislerde değişikliğin takip edilmesi ve yönetilebilmesi adına yazılımların gerçek hayatta nasıl versiyonlandığına yönelik bir uygulama gerçek hayata aktarıldı.

Bu alanda uğraş verecek araştırmacılara servis sözleşme versiyonu geliştirildiğinde ve yayımlandığında mevcut kurulmuş veya kurulacak sistemlere etkisinin analizi konusunda çalışmaları önerilmektedir. Bu etkinin ölçümü, yeni sözleşme versiyonun, eski versiyon ve ilgili diğer programlar ile direkt uyumlu olup olmadığıdır. Ayrıca, araştırmacılara, servis'in taslağı gelecekte öngörülmeyen değişiklikleri de kapsayacak şekilde tasarlamaları önerilmektedir.

Herkese doğru gelecek bir versiyon oluşturma yöntemi yoktur. Çünkü versiyon oluşturma, bir servisin kullanımı süresince kontrolü temsil eder.

KAYNAKLAR

1. Schack Stephen R., "Object-Oriented & Classical Software Engineering", McGraw-Hill, ISBN: 13:978-007-125849-4, New York, 2007

2. http://en.wikipedia.org/wiki/SOAP (Erişim Tarihi: 10.09.2013)

3. http://www.oracle.com/technetwork/articles/web-services-versioning-094384.html (Erişim Tarihi: 04.05.2013)

4. http://stackoverflow.com/questions/2101566/what-are-your-webservice-versioning-best-practices (Erişim Tarihi: 17.03.2013)

5. Erl T., "Web Service Contract Design and Versioning for SOA", Prentice Hall PTR, ISBN: 013613517X, New Jersey, 2008

6. http://www.infoq.com/articles/contract-versioning-comp2 (Erişim Tarihi: 14.07.2013)

7. http://technet.microsoft.com/tr-tr/library/ms187822.aspx (Erişim Tarihi: 10.05.2013)

8. http://soa.dzone.com/articles/web-service-contract-versionin?page=0,0 (Erişim Tarihi: 12.04.2013)

9. http://www.ibm.com/developerworks/webservices/library/ws-version/ (Erişim Tarihi: 10.04.2013)

10. Erl T., "SOA Design Patterns", Prentice Hall PTR, ISBN: 0136135161, New Jersey, 2008

11. http://www.sei.cmu.edu/reports/11tn009.pdf (Erişim Tarihi: 01.02.2013)

12. http://www.w3schools.com/ (Erişim Tarihi: 09.03.2013)

13. http://soa.sys-con.com/node/44356 (Erişim Tarihi: 20.08.2013)

14. Erl T., "Service-Oriented Architecture: Concepts, Technology, and Design", Prentice Hall PTR, ISBN: 0-13-185858-0, New Jersey, 2005

15. http://msdn.microsoft.com/en-us/library/ms954726.aspx (Erişim Tarihi: 01.02.2013)

16. http://www.subbu.org/blog/2005/08/web-services-versioning-part-2 (Erişim Tarihi: 10.09.2013)

17. Osmar R.Zaiane, “Data and Behavioral Contracts forWeb services”, University of Bologna Department of Computer Sciences, 2007

18. Erl T., “Service-Oriented Architecture : A Field Guide to Integrating XML and Web Services”, Prentice Hall PTR, ISBN: 9780131428980, New Jersey, 2004

19. Salvendy G. and Karwowski W., “Service Engineering”, John Wiley&Sons, ISBN: 978-0-470-38241-7, New Jersey, 2010

20. http://www.isbank.com.tr/Userfiles/News/akis_parakod.jpg?src=Parakod (Erişim Tarihi: 01.11.2013)

ÖZGEÇMİŞ

Fatih BALIKCI, 1987 yılında Zonguldak / Ereğli'de doğdu. Öğrenimlerini sırasıyla Turgut Reis İlkokulu, Özel Utku Ortaokulu, Kadri Yılmaz Lisesi'nde tamamladı. 2005 yılında Maltepe Üniversitesi Bilgisayar Mühendisliği bölümünü kazandı ve 2010 yılında fakülte birinciliği ile mezun oldu. Mart 2011 – Ağustos 2012 arasında Maltepe Üniversitesi Bilgi İşlem Daire Başkanlığında Yazılım Uzmanı pozisyonunda çalıştı. Ağustos 2012'den bu yana Türkiye İş Bankası'na yazılım sağlayan SoftTech A.Ş.'de Yazılım Mühendisi olarak çalışmaktadır. Eylül 2011'de Maltepe Üniversitesi'nde başladığı Bilgisayar Mühendisliği Yüksek Lisans programını Şubat 2014'de tamamladı.