Gizlilik Eklentili Wright Betimlemeleri İçin Bir XML Şeması

Burak Yolaçan[1], Cemil Ulu[2], Halit Oğuztüzün[3]

[1,2,3] Orta Doğu Teknik Üniversitesi, Bilgisayar Mühendisliği Bölümü, 06531, Ankara
[1] burak.yolacan@ceng.metu.edu.tr, [2] cemil@ceng.metu.edu.tr
[3] oguztuzn@ceng.metu.edu.tr

Özet. Dünya Çapında Web Konsorsiyumu (W3C - Worldwide Web Consortium) tarafından geliştirilmiş bir standart olan Genişletilebilir İşaretleme Dili'nin (XML - Extensible Markup Language), bir Mimari Betimleme Dili (ADL - Architecture Description Language) olan Wright'ın gizlilik eklentili şekli ile ifade edilen tasarımların kodlanması için uygun bir gösterim olduğuna karar verildi ve dilin grameri bir XML Şeması'na (XML Schema) çevrildi. Böylece mimari betimlemeleri girdi kabul eden uygulamaların XML araç desteğiyle kullanabileceği ortak bir form oluşturuldu. Örnek bir uygulama olarak, XML'de yazılmış gizlilik eklentili Wright tasarımlarını, güvenlik denetleyici bir aracın kabul ettiği gösterime çeviren bir çözümleyici, bir W3C standardı olan Belge Nesne Modeli (DOM - Document Object Model) arayüzüne sahip Java tabanlı Xerces XML Parser temel alınarak geliştirildi.

1 Giriş

Yazılım kaynak kodları ve formel belirtimleri genellikle okunması ve yazılması insanlar için özel bir zorluk içermeyen düz metinlerdir. Ne var ki bu tür bir gösterim şekli, kodları ayrıştıracak ve işleyecek olan araçlar için pek elverişli olmamaktadır. Bu bağlamda, işlenmesi kolay olan esnek bir gösterimin kullanılması, daha az çabayla ve daha az zamanda uygulamalar geliştirilmesine öncülük edebilir. Söz konusu gösterimin, geniş bir araç desteğine de sahip olması daha büyük kolaylıklar sağlayabilecektir.

Bu çalışmada, sözü edilen yaklaşım biçimi, bir mimari betimleme dili olan Wright'ın [1] gizlilik eklentili şekli (Wright/C – Wright with confidentiality annotations) [2] ile yazılmış tasarımların kodlanması için XML'in bir gösterim ortamı olarak kullanılmasıyla hayata geçirilmiştir. Bu amaçla, Wright/C ile bir yazılım sistemini betimlemek için kullanılan gösterim şekli, XML ayrıştırıcıları (parser) tarafından doğrudan kullanılabilecek bir biçime dönüştürülmüştür.

Bu tebliğin bundan sonraki kısmı şu şekilde düzenlenmiştir. İkinci bölüm, çalışmanın bağlı olduğu Wright ve Wright/C hakkında bilgi vermektedir. Üçüncü bölüm XML tabanlı gösterim şekli ile ilgili diğer çalışmalara değinmektedir. Dördüncü bölümde, bu çalışmada izlenen yaklaşım anlatılmıştır. Beşinci bölüm geliştirilen bir araçla izlenen yaklaşımın bir örnek üzerinde gösterilmesi üzerinedir. Son olarak, altıncı bölüm ulaşılan durumu ortaya koymaktadır.

2 Artalan

2.1 Wright

Yazılım sistemlerinin mimari tasarımları sırasında, gerek sözdizimi gerek semantiği sağlam bir temele oturmayan notasyonların (geleneksel kutu-çizgi diyagramları gibi) kullanılmasından dolayı belirsiz ve eksik tasarımların ortaya çıkması muhtemeldir. Mimari tasarım dilleri bu anlamda, söz konusu probleme, formel dillere dayalı bir yaklaşımla çözüm üretmeye çalışmaktadır.

Söz konusu amacı taşıyan ve bu çalışmanın ilgi alanına giren bir dil de Wright'tır [1]. Wright ile tasarlanmış tipik bir yazılım sistemi, bileşenler (component), bağlayıcılar (connector), arabirimler (interface), biçemler (style) ve yapılanışlardan (configuration) oluşmaktadır. Bir bileşen, kapı (port) arabirimi vasıtasıyla hesaplama hizmeti sunan mimari bir yapı taşıdır. Bir bağlayıcı, bileşenler arasındaki etkileşim ortamını, kapıların üstleneceği rolleri tanımlayarak sağlayan mimari bir ögedir. Bağlayıcılar aynı zamanda iştirak eden bileşenlerin hesaplamalarını birleştiren bir tutturucu (glue) da tarif ederler. Gerek bileşen ile bağlayıcı etkileşim örüntüleri, gerekse tutturucular ve hesaplamalar için süreçler, Hoare'ın CSP (Communicating Sequential Processes) diliyle [3] ifade edilir. Biçemler, içinden sistem tasarımcılarının geliştirmek istedikleri mimariye göre seçebileceği bir takım kurallarla (constraints) bir arada bulunan bileşen ve bağlayıcılardan oluşan gruplardır. Son olarak da bir yapılanış, bileşenleri ve bağlayıcıları biraraya toplayan belirli bir sistemin tüm ilingesini (topology) ve davranışını belirler.

2.2 Gizlilik Eklentili Wright

Güvenlik, yazılım sistemlerinin en önemli işlev-dışı (non-functional) özelliklerinden biridir. Güvenlik kaygılarının, yazılım sistemleri henüz mimari tasarım aşamasındayken dikkate alınması, işletim esnasında oluşabilecek zararların azaltılmasına katkı sağlayabilir. Wright'ın bu çeşit güvenli sistemler için gizlilik (confidentiality) eklentileri ile desteklenmiş bir uzantısı olan Wright/C, bu yaklaşımın bir gerçeklenmesi olarak ortaya konulmuştur. Söz konusu eklentiler esas olarak üç işlevi desteklemektedir: (i) gizlilik sınıfları örgüsünün (gizlilik politikasına bağlı olarak) tanımlanması, (ii) bir mimari yapılanışta, bileşenlerin kapılarına yetki (clearance) atanması ve (iii) çıktı verilerini tutan

değişkenlere (istenirse parametrik olarak) gizlilik sınıfı atanması. Bu şekilde sadece yeterli yetkiye sahip kapılar arasında belirli gizlilik derecesindeki bilgi akışına izin verildiğinin güvencesini mimari tasarım düzeyinde sağlamak mümkün olmaktadır. Wright/C eklentilerinin sözdizim ve semantik tanımlamaları ile sistem geliştirme döngüsü içindeki yeri hakkındaki açıklamalara [2] numaralı kaynaktan ulaşılabilir.

3 İlgili Çalışmalar

XML tabanlı mimari yazılım dillerinde genellikle, Belge Türü Tanımlaması (DTD – Document Type Definition) ve şema yönelimli olmak üzere iki tür yaklaşım izlenmektedir. Mimari Tasarım İşaretleme Dili (ADML - Architecture Description Markup Language) [4] bir mimari tasarım değişim (interchange) dili olan Acme'nin [5] gösterimi için DTD teknolojisinden yararlanmaktadır. XML tabanlı betimleme alanındaki bir diğer çalışma da mimari bir tasarımda kullanılabilecek bileşenler, bağlayıcılar ve arabirimler gibi tipik ögelerin betimlenmesinde kullanılmak üzere çekirdek bir XML Şeması sağlayan xArch olarak kendini göstermektedir. xArch [6], hem yazılım mimarilerinin gösteriminde doğrudan bir betimleme dili olarak hem de şemaların kalıtım (inheritance) özelliğinin sağladığı genişletilebilirliği ile yeni betimleme dillerinin tanımlanmasında bir üst-dil olarak kullanılabilir. Bu şekilde, çekirdek dil olan Acme için tanımlanan şemaların xArch'ın temel şemasına eklenmesiyle xAcme [7] dili oluşturulmuştur. xArch şemasının genişletilmesiyle elde edilen bir başka betimleme dili de xADL 2.0'dır [8]. Diğer taraftan xADL'in DTD yönelimli 1.0 versiyonu ile oluşturulan xC2, xDarwin ve xSADL genişletilebilir betimleme dilleri de ilgili diğer çalışmalardır [9, 10].

4 Yaklaşım

4.1 Neden XML?

Herşeyden önce, XML'in halihazırda sunduğu çok sayıdaki ücretsiz ve ticari yazılım aracından oluşan geniş yelpazesinin sağlayacağı potansiyel destek sayesinde XML ile kodlanmış gösterimlerin işlenmesinde büyük kolaylıklar sağlanabilecektir. Bu şekilde, sözü edilen araçların yeni baştan geliştirilmesi için harcanacak çaba, belgelerin analiz ve işlenmesinde kullanılabilecektir [11].

İkinci olarak, kaynak kodlarının gösteriminde değişiklik yapılması durumunda, XML'in genişleyebilirlik özelliğinden faydalanarak duruma uyum sağlamak mümkün olabilecektir.

Üçüncü olarak, XML'in metne bağlı kodlamasının bir getirisi olan platform bağımsızlığı sayesinde XML'de gösterilmiş herhangi bir kaynak kod farklı bilgisayar sistemleri arasında paylaşılabilecektir.

Dördüncü olarak, XML uygulamadan bağımsız (application independent) bir yapıya sahiptir. DTD ve şema tanımlayabilme yetisi sayesinde, iki ya da daha fazla XML belgesinin ve aynı zamanda bunları paylaşan uygulamaların karşılıklı işlerliği (interoperability) mümkün olabilecektir.

Son olarak, XML, üzerinde herhangi bir üreticinin tek taraflı değişiklikler yaparak sistemler arasındaki karşılıklı işlerliği engelleyemeyeceği bir açık standarttır.

4.2 Neden XML Şeması?

DTD'nin geniş çaplı kullanımına karşın, daha yeni bir yaklaşım olan XML Şeması şu yarar kazanımlarına sahip bulunmaktadır:

- Şemalar, XML ile aynı sözdizimini kullanmaktadır. Yeni bir öğrenme çabasına ve ilgili grameri çözümlemede başka bir araca gerek kalmamaktadır.
- Şemalar, DTD'nin sağladığı sınırlı desteğe karşılık zengin bir veri türü kümesi ortaya koymaktadır.
- Yeni veri türlerinin tanımlanması mümkündür.
- Var olan bir veri türü üzerinde genişletme veya sınırlama yapılabilmektedir.
- Farklı bağlamlarda aynı isimle öge tanımlanması mümkün olabilmektedir.

4.3 BNF-Şema Dönüştürümü

BNF notasyonundaki bir bağlam-duyarsız grameri bir XML şemasına dönüştürme işleminde takip edilebilecek bazı yöntemler bulunmaktadır. Uç olmayan (non-terminal) bir simge (symbol), bir işaretleme meydana getiriyorsa uygun bir biçim imi (tag) kullanılarak şema ögesi olarak eşlenebilir. Uç olmayan bir simge doğrudan işaretleme meydana getirmemesi durumunda ise biçim imi oluşturulmayacağından 'group' göstergesi kullanılarak örtük bir şekilde eşlenebilir. Uç simgeler, tasarımcı tarafından tanımlanmış basit veri türleriyle ya da katar ve tamsayı gibi temel veri türleriyle eşleşebilir. Bir dizi ögenin, uç olmayan bir simge tarafından tanımlanması durumunda, 'sequence' göstergesini kullanmak elverişli olacaktır. Uç olmayan bir simge tarafından '|' işareti kullanılarak ortaya konan birden fazla yeniden-yazma kuralı (rewrite rule) varsa 'choice' göstergesi kullanılabilir. Bir yeniden-yazma kuralının sağ tarafında kendini tekrarlayan yapıların olması durumunda 'minOccurs' ve 'maxOccurs' göstergelerinden faydalanılabilir. Seçimli bir varlık, 'minOccurs' özniteliği (attribute) için sıfır değeri belirlenerek tanımlanabilir. Hem bir öge hem de öznitelik olarak belirtilebilecek bir oluşum durumunda ikincisini tercih etmek daha kısa ve anlaşılır olabilir. Dönüşüm

yöntemlerinin bir uygulaması olarak Şekil 1'de 'Component' simgesinin BNF'teki tanımı ile Şekil 2'de buna karşılık gelen şema gösterimi örnek olarak verilebilir.

```
Component := "Component" SimpleName ['(' FormalCCParams')']
 [PortList]
 "Computation" BehaviorDescription;
```

Şekil 1. 'Component' simgesi için BNF gösterimi

```
<x:complexType name="Component">
 <x:sequence>
  <x:element name="param" type="FormalCCParam" minOccurs="0"
maxOccurs="unbounded"/>
  <x:group ref="PortList" minOccurs="0"/>
  <x:element name="Computation" type="BehaviorDescription"/>
 </x:sequence>
 <x:attribute name="name" type="SimpleName" use="required"/>
</x:complexType>
```

Şekil 2. 'Component' simgesi için XML şeması gösterimi

Bu örnekte, uç olmayan 'Component' simgesi tarafından tanımlanan bir dizi öge için 'sequence' göstergesi, seçimli olan ve kendini tekrarlayabilen parametreler için 'param' biçim imli 'element', sıfır değerli 'minOccurs' ve limitsiz 'max occurs'göstergeleri, uç olmayan ve doğrudan işaretleme meydana getirmeyen 'PortList' seçimli simgesi için 'group' ve sıfır değerli 'minOccurs' göstergeleri, bir işaretleme meydana getiren 'BehaviorDescription' simgesi için 'Computation' biçim imli 'element' göstergesi, hem öge hem öznitelik olarak belirtilebilecek 'SimpleName' simgesi için daha kısa bir gösterim oluşturan 'attribute' göstergesi kullanılmıştır.

Wright/C BNF ve dönüştürülmesiyle elde edilen Wright/C XML Şeması'na [12] numaralı kaynaktan ulaşılabilir. Wright dilinin özgün sözdizimini tanımlayan gramer kaynak [13]'ten alınmıştır.

5 Örnek Olay İncelemesi

Kaynak kodunun XML'de kodlanmasının Wright/C tasarımlarının gösterimi için uygun bir yöntem olup olmadığının sınanması amacıyla bir çözümleyici gerçekleştirildi. Söz konusu çözümleyici, veri akışına dayalı gizlilik denetleyici bir aracın kullanabileceği soyut sözdizimin (abstract syntax) oluşturulmasından sorumludur [2]. Bu amaçla çözümleyici, Wright/C tasarımlarında görülebilecek mimari oluşumlara ilişkin veri türü tanımları oluşturur. Bunun yanında, kapılar arasında bağımlılık analizi yapmak, kapılarda alınan ve gönderilen verilerin güvenliği ile ilgili veri yapılarını oluşturmak ve örgü (lattice) işlevleri [2] ile ilgili algoritmaları gerçekleştirmek durumundadır.

5.1 Neden Xerces ve DOM?

Xerces-J XML ayrıştırıcısının [14] Wright/C tasarımlarının işlenmesi için seçilmesinde, öncelikle W3C XML Şema Tavsiyesine [15, 16] uyan tek güncel araç olması, DOM [17] arabirimini desteklemesi ve son olarak da platform bağımsız bir programlama dili olan Java [18] tabanlı olması etkili olmuştur.

XML dokümanlarının işlenmesi ve analizinde genellikle kullanılan iki çeşit arabirim bulunmaktadır: XML için Basit Programlama Arabirimi (SAX – A Simple API for XML) [19] ve DOM. Sonraki, öncekine kıyasla bazı yarar kazanımları sağlamaktadır. İlk olarak SAX, kullanıcının belge boyunca belirli yerlerde yerine getirilecek belirli davranışları tanımladığı olaya dayalı bir arabirimdir. Bu, birbirinden farklı sorguların olması durumunda belgenin ağaç gösterimi üzerinden birden fazla geçişin sebep olacağı gecikmeden dolayı pek verimli olmayan bir yöntemdir. Diğer taraftan DOM ile belgeyi bir seferde belleğe almak ve sorguları daha kısa sürede cevaplamak mümkün olabilmektedir. İkinci olarak, SAX kullanıcısı tekil olaylardan elde edilen parça bilgileri yeni bir yorumlama için biraraya getirmek zorundadır. Buna karşılık DOM kullanıcısının, genellikle dökümanın ağaç yapısınının önceden elde bulunmasından dolayı bilgiye daha doğrudan erişmesi mümkün olabilmektedir. Son olarak, DOM'un bir W3C standardı olma özelliği, aynı standarttaki diğer araçlarla veri değişiminde bulunmasına imkan vermektedir. Öte yandan, SAX doküman büyüklüğünde ana belleğe bağlı bir sınırlamaya tabi olmadan performansını sürdürebilmektedir. DOM'da SAX'ın bu özelliğinin olmayışı özellikle büyük hacimli betimlemelerde bellek kaynaklı sorunlara yol açabilecektir.

5.2 Güvenli Yazdırma Sunucusu

Çözümleyiciyi sınamak için kaynak [2]'den alınan Güvenli Yazdırma Sunucusu (SPS - Secure Print Server) örneği kullanılmıştır. SPS'in görevi, dokümanların gizliliklerine (secret/public) göre basılmasını sağlamak ve bu bağlamda gizli bir dokümanın herkese açık bir yazıcıda basılmasını engellemektir. Söz konusu sistem farklı yerlerde bulunan biri herkese açık diğeri erişimi kısıtlı olan iki yazıcı, yazdırma isteklerini kontrol eden bir yazdırma sunucusu ve biri sıradan (everyone) diğeri yetkili (authorized) olan iki sınıf kullanıcıdan oluşmaktadır. Ayrıca kullanıcı-sunucu ve sunucu-yazıcı iletişimini sağlayan bağlayıcılar bulunmaktadır. Şekil 3'te SPS için Wright/C tasarımı görülebilir.

```
Lattice CSL
  Security Labels
    PUBLIC, SECRET
  Ordering
    PUBLIC, SECRET
  ClearanceList
    EVERYONE : PUBLIC
    AUTHORIZED : SECRET
End Lattice (* örgü dosyası "ACLattice.txt" *)

Style ClientServerPrinting
  Import Lattice CSL "ACLattice.txt"
  Component Client(t : SecurityLabel) =
    Port PrintP = request! x^t -> PrintP
    Port PrintS = request! x^SECRET -> PrintS
    Computation = PrintP.request!x^t ->
      Computation []
      PrintS.request! x^SECRET->Computation
  Component Printer =
    Port Receive = request? x -> Receive
    Computation = Receive.Request?x ->
      DoPrint -> Computation
  Component PrintServer =
    Port RequestP = request?x -> RequestP
    Port RequestS = request?x -> RequestS
    Port OutputP = Print!x  -> OutputP
    Port OutputS = Print!x -> OutputS
    Computation = RequestP.Request?x –>
      OutputP.Print!x ->  Computation []
      RequestS.Request?x ->
      OutputS.Print! x ->  Computation
    Connector PrintConnector =
      Role ClientP = request?x -> ClientP
      Role ServerP = request!x -> ServerP
      Glue = ClientP.request?x ->
```

```
Configuration  PrintExample
  Style ClientServerPrinting
  Instances
    UA: Client(CSL.min())
    UB: Client(CSL.max())
    PS: PrintServer
    SECUREPRINTER: Printer
    PUBLICPRINTER: Printer
    CONN1: PrintConnector
    CONN2: PrintConnector
    CONN3: PrintConnector
    CPRINTS: PrintConnector
    CPRINTP: PrintConnector
  Clearance
    UA: EVERYONE
    UB: AUTHORIZED
    UB.PrintP: EVERYONE
    PS.RequestP: EVERYONE
    PS.RequestS: AUTHORIZED
    PS.OutputP: EVERYONE
    PS.OutputS: AUTHORIZED
    SECUREPRINTER: AUTHORIZED
    PUBLICPRINTER: EVERYONE
  Attachments
    UA.PrintP as CONN1.ClientP
    PS.RequestP as CONN1.ServerP
    UB.PrintS as CONN2.ClientP
    PS.RequestS as CONN2.ServerP
    UB.PrintP  as CONN3.ClientP
    PS.RequestP as CONN3.ServerP
    PS.OutputP as CPRINTP.ClientP
    PUBLICPRINTER.Receive as
                          CPRINTP.ServerP
    PS.OutputS as CPRINTS.ClientP
    SECUREPRINTER.Receive as
                          CPRINTS.Server
P
```

```
                ServerP.request!x -> Glue [] §          End Configuration
End Style
```

Şekil 3. Gizlilik sınıfları örgüsü ile SPS mimari biçem ve yapılanışı

SPS'in XML betimlemesi, çözümleyici tarafından oluşturulan soyut sözdizimi ve çözümleyicinin kodlarına [12] numaralı kaynaktan erişilebilir. Şekil 4 örnek olarak 'Printer' bileşeninin XML'de kodlanmış halini göstermektedir.

```
<Component name="Printer">
 <Port name="Receive">
  <CSPExp> request? x -> Receive </CSPExp>
 </Port>
 <Computation>
  <CSPExp>Receive.Request?x->DoPrint->Computation</CSPExp>
 </Computation>
</Component>
```

Şekil 4. 'Printer' bileşeninin XML gösterimi

Çözümleyici, literatürden [1] alınarak bir gizlilik politikasına göre uyarlanmış olan AEGIS sistem mimarisi ile de sınanmıştır. SPS'e kıyasla daha geniş çaplı olan bu örnek ile ilgili betimlemeler ve soyut sözdizimi [12] numaralı kaynakta görülebilir.

6 Sonuç

Bu çalışmanın esas katkısı, mimari betimleme dili Wright'ın gizlilik eklentileriyle birlikte sözdizim tasarımlarının XML'de gösteriminin sağlanması olmuştur. Bu gösterim tarzı, alışılagelmiş düz metin kodlaması ile elde edilmesi güç olanaklar sağlamaktadır. Örnek çalışmada gözlenen husus, varolan XML araçlarından yararlanmak sayesinde, belgeleri işleyen uygulamaların geliştirilmesinde daha az çabanın gerektiğidir. Öte yandan, mimari betimlemeler üzerinde işlem yapan araçların karşılıklı işlerliği için XML temelinde ortak bir platform yaratılmış olmaktadır.

Bu olanaklar araç geliştiricilerin çalışmasını kolaylaştırsa da, kullanıcı tarafı okunması ve yazılması pek kolay olmayan XML tabanlı gösterimden memnun kalmayabilir. Bu yüzden, mimari tasarımcıların da hoşnut olması için düz metni XML'e dönüştüren araçların geliştirilmesine ihtiyaç duyulacaktır.

Kaynakça

1. R. Allen, "A formal approach to software architecture", Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, Mayıs 1997. http://reports-archive.adm.cs.cmu.edu/anon/1997/CMU-CS-97-144.pdf
2. C. Ulu, "Access control in software architectures", Gelişme Raporu, Bilgisayar Mühendisliği Bölümü, ODTÜ, Aralık 2002.
3. C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, 1985. Elektronik versiyon http://www.usingcsp.com.
4. ADML Ana Sayfası, http://www.opengroup.org/architecture/adml/adml_home.htm
5. D. Garlan, R. Monroe ve D. Wile, "Acme: An architectural description interchange language", Proceedings of CASCON'97, Kasım 1997, s. 169-183. http://www-2.cs.cmu.edu/afs/cs/project/able/ftp/acme-cascon97/acme-cascon97.pdf
6. xArch Ana Sayfası, http://www.isr.uci.edu/architecture/xarch/
7. xAcme Ana Sayfası, http://www-2.cs.cmu.edu/~acme/pub/xAcme/
8. xADL Ana Sayfası, http://www.isr.uci.edu/projects/xarchuci/index.html
9. R. Khare, M. Guntersdorfer, P. Oreizy, N. Medvidovic ve R. N. Taylor, "xADL: Enabling architecture-centric tool integration with XML", Proceedings of HICSS'34, Ocak 2001. http://www.ics.uci.edu/~taylor/ics228/HICSS 2001.pdf
10. E. Dincel, R. Roshandel ve N. Medvidovic, "ADL independent architectural representation in XML", University of Southern California, Mayıs 2000. http://sunset.usc.edu/publications/TECHRPTS/2000/usccse2000-519/usccse2000-519.pdf

11. S. Pruitt, D. Stuart, W. Sull ve T. W. Cook, “The merit of xml as an architecture description language meta-language”, Microelectronics and Computer Technology Corporation, Ekim 1998. http://citeseer.nj.nec.com/pruitt98merit.html

12. Proje Ana Sayfası, http://www.ceng.metu.edu.tr/~e112075/project

13. Wright Tools, http://www-2.cs.cmu.edu/~able/wright/wright_tools.html

14. The Apache Group, Xerces Java Parser, http://xml.apache.org/xerces-j/

15. World Wide Web Consortium Web Site, http://www.w3c.org

16. World Wide Web Consortium. XML Schema W3C Recommendation, Mayıs 2001, http://www.w3.org/TR/xmlschema-0/

17. World Wide Web Consortium. Document Object Model (DOM) Level 1 Specification, W3C Recommendation, Ekim 1998, http://www.w3.org/TR/REC-DOM-Level-1

18. J. Gosling, B. Joy ve G. Steele, The Java Language Specification, Addison Wesley, 1996.

19. SAX, http://www.saxproject.org