Gizlilik Eklentili Wright Betimlemeleri için 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 kolay olan düz metinlerdir. Ne var ki bu tür bir gösterim şekli, kodları analiz edip 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, esnek yapısının yanında geniş bir araç desteğine de sahip olması daha büyük kolaylıklar sağlayabilecektir.

Bu çalışmada, sözü edilen yaklaşım biçimi, bir mimari betimleme dili olan Wright'ın gizlilik eklentili şekli ile yazılmış tasarımların kodlanması için XML'in bir gösterim ortamı olarak kullanılmasıyla hayata geçirilmiştir. Bu amaçla, Wright'ın genişletilmiş hali ile bir yazılım sistemini betimlemek için kullanılan gösterim şekli, belgelerin analizinde 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'ın güvenlik eklentileri ile genişletilmiş uzantısı 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 koymakta ve ek bölümü kullanılan örnek ile ilgili bilgi sunmaktadır.

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 yaklaşım getirerek 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 etkileşimi ve haberleşmeyi, kapıların üstleneceği işlevleri (role) tanımlayarak mümkün kılan mimari bir ögedir. Bağlayıcılar aynı zamanda iştirak eden bileşenlerin hesaplamalarını birleştiren bir tutturucu (glue) da tarif ederler. Arabirimler, tutturucular ve hesaplamalar için süreçler tanımlarlar. 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 sistemin tüm ilingesini (topology) ve davranışını belirler.

2.2 Genişletilmiş Wright

Güvenlik saldırıları sadece çok büyük ekonomik kayıplara neden olmakla kalmamakta aynı zamanda hayati zararlara da sebebiyet veren ciddi bir problem olarak ortaya çıkmaktadır. Yazılım sistemlerinin daha mimari tasarım aşamasındayken güvenlik kaygılarının dikkate alınması, bu sorunun çözümü için etkili bir yol olabilir. Wright'ın bu çeşit güvenli sistemler için gizlilik eklentileri ile desteklenmiş uzantısı [2], sözü edilen yaklaşımın bir gerçeklenmesi olarak kendini göstermektedir. Bu eklentiler, bileşenlerin kapılarına ve bağlayıcıların işlevlerine yetki (clearance) atamak için kullanılmaktadır. Bu şekilde sadece yeterli yetkiye sahip kapılar arasında belirli gizlilik derecesindeki bilgi akışına izin verilerek sistem kaynaklarına olan girişlerin düzenlenmesini sağlayan bir kontrol mekanizması kurulmuş olmaktadır.

3 İlgili Çalışmalar

XML tabanlı mimari yazılım dillerinde genellikle, Belge Türü Tanımlaması (DTD – Document Type Definition) ve şema yönelimli olmak üzere iki tür yaklaşım kullanılmaktadır. Mimari Tasarım İşaretleme Dili (ADML - Architecture Description Markup Language) [3] bir mimari tasarım değişim (interchange) dili olan Acme'nin [4] 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 [5], hem yazılım mimarilerinin gösteriminde doğrudan bir betimleme dili olarak hem de şemaların kalıtım (inheritance) özelliğinin sağladığı genişletilebilirliği ile yeni

betimleme dillerinin tanımlanmasında kullanılabilir. Bu şekilde, çekirdek dili olan Acme için tanımlanan şemaların xArch'ın temel şemasına eklenmesiyle xAcme [6] dili oluşturulmuştur.

Betimleme dillerinin yanısıra programlama dillerindeki kaynak kodların gösterimi için yapılmış olan XML tabanlı çalışmalar da bulunmaktadır. C++ İşaretleme Dili (CppML - C++ Markup Language) ve Java İşaretleme Dili (JavaML – Java Markup Language) kapsamında ilgili dilin grameri için birer DTD tanımlanmıştır. [7].

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 [8].

İkinci olarak, kaynak kodlarının özgün gösterimlerindeki olası değişiklikler 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 yakın zamanlı 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 XML Şeması'na 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 ve doğrudan işaretleme meydana getirmeyen bir simge olması durumunda 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. Kendisine tekabül eden BNF'nin dönüştürülmesiyle elde edilen genişletilmiş Wright XML Şeması ve BNF'nin kendisine [9] numaralı kaynaktan ulaşılabilir.

5 Örnek Olay İncelemesi

Kaynak kodunun XML'de kodlanmasının eklentili Wright tasarımlarının gösterimi için uygun bir yöntem olup olmadığının sınanması amacıyla bir çözümleyici gerçekleştirildi. Söz konusu çözümleyici güvenlik denetleyici bir aracın kullanabileceği soyut sözdizimin (abstract syntax) oluşturulmasından sorumludur [10]. Bu amaçla çözümleyici, genişletilmiş Wright 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 [11] genişletilmiş Wright tasarımlarının işlenmesi için seçilmesinde, öncelikle W3C XML Şema Tavsiyesi [12, 13] için destek veren tek güncel işlemci olması, DOM [14] arabirimini desteklemesi ve son olarak da platform bağımsız bir programlama dili olan Java 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) [15] 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 gereken 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 karşın, 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, kendisiyle geliştirilen araçların 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 [10] numaralı kaynaktan alınan Güvenli Yazdırma Sunucusu (SPS - Secure Print Server) örneği kullanılmıştır. SPS'in görevi, dökümanların gizliliklerine göre basılmasını ve bu bağlamda gizli bir belgenin herkese açık bir yazıcıda basılmamasını sağlayacak bir düzenek oluşturmaktır. Söz konusu örnek sistem farklı yerlerde bulunan biri herkese açık diğeri gizli olan iki yazıcı, yazdırma isteklerini kontrol eden bir yazdırma sunucusu ve biri sıradan diğeri yetkili olan iki çeşit kullanıcıdan oluşmaktadır. Ayrıca kullanıcılar ile sunucu arasında ve sunucu ile yazıcılar arasında iletişim sağlamak için bağlayıcılar bulunmaktadır.

SPS'in düz metindeki eklentili Wright tasarımları ek kısmından takip edilebilir. SPS'in XML betimlemesi, çözümleyici tarafından oluşturulan soyut sözdizimi ve çözümleyicinin kodlarına [9] numaralı kaynaktan erişilebilir. Aşağıdaki örnekte ek A.2'deki 'Printer' bileşeninin XML'de kodlanmış şekli görülmektedir.

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

Çözümleyici literatürden [1] uyarlanmış ve SPS'e kıyasla daha geniş çaplı bir örnek olan AEGIS ile de sınanmıştır. Örnek ile ilgili betimlemeler ve ilgili soyut sözdizimi [9] numaralı kaynakta görülebilir.

6 Sonuç

Genişletilmiş Wright tasarımlarının XML'de gösteriminin yapılması alışılagelmiş düz metin kodlaması ile elde edilmesi güç olanaklar sağlamaktadır. Bunlardan biri, varolan XML araçlarının sağladığı geniş çaplı desteğin kullanılması suretiyle, belgeleri işleyen uygulamaların geliştirilmesinde daha az çabanın gerekmesidir. Bir diğer önemli olanak, aynı XML arabirimi aracılığıyla iletişim sağlamaları beklenilen araçların karşılıklı işlerliği için oluşturulmuş ortak platformdur. Ayrıca XML standardı, ileride olabilecek marka bağımlılıkları için de bir engel vazifesi görmektedir.

Her ne kadar bu olanaklar araç geliştiricilerin çalışmasını kolaylaştırsa da, kullanıcı tarafı, okuması, yazması ve takip edilmesi pek kolay olmayan XML tabanlı gösterimden memnun kalmayabilir. Bu yüzden, hem mimari tasarımcıların hem de uygulama geliştiricilerin 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", Progress Report I, Computer Engineering Department, METU, Aralık 2002.
3. ADML Ana Sayfası, http://www.opengroup.org/architecture/adml/adml_home.htm
4. 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
5. xArch Ana Sayfası, http://www.isr.uci.edu/architecture/xarch/
6. xAcme Ana Sayfası, http://www-2.cs.cmu.edu/~acme/pub/xAcme/
7. E. Mamas ve K. Kontogiannis, "Towards portable source code representation using XML", WCRE'2000, Kasım 2000, s. 172-182. http://www.swen.uwaterloo.ca/~evan/Papers/wcre2000.pdf
8. 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
9. Proje Ana Sayfası, http://www.ceng.metu.edu.tr/~e112075/project
10. C. Ulu, "Access control in software architectures", Progress Report II, Computer Engineering Department, METU, Mart 2003.
11. The Apache Group, Xerces Java Parser, http://xml.apache.org/xerces-j/
12. World Wide Web Consortium Web Site, http://www.w3c.org
13. World Wide Web Consortium. XML Schema W3C Recommendation, Mayıs 2001, http://www.w3.org/TR/xmlschema-0/
14. World Wide Web Consortium. Document Object Model (DOM) Level 1 Specification, W3C Recommendation, Ekim 1998, http://www.w3.org/TR/REC-DOM-Level-1
15. SAX, http://www.saxproject.org

Ek: SPS'in Eklentili Wright Tasarımı

A.1 SPS için Örgü

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

A.2 SPS Tasarımı

```
Style ClientServerPrinting
   Import Lattice CSL "ACLattice.txt"

   Component Client(t : SecurityLabel) =
       Port PrintServiceP = request! x^t   -> PrintServiceP
       Port PrintServiceS = request! x^SECRET  -> PrintServiceS
       Computation  =  PrintServiceP. request! x^t -> Computation
              [] PrintServiceS. request! x^SECRET-> Computation

   Component Printer =
       Port Receive = request? x   -> Receive
       Computation = Receive. Request? x -> DoPrint-> Computation

   Component PrintServer =
       Port PrintPublic = request?x -> PrintPublic
       Port PrintSecret = request?x -> PrintSecret
       Port OutputPublic = Print!x  -> OutputPublic
       Port OutputSecret = Print!x -> OutputSecret
          Computation = PrintPublic.Request?x –>
                           OutputPublic.Print!x ->
                        Computation
                           []  PrintSecret.Request?x ->
                                 OutputSecret.Print!x ->
                               Computation

   Connector  PrintConnector    =
       Role ClientP = request? x  -> ClientP
       Role ServerP = request!x   -> ServerP
       Glue = ClientP.request?x -> ServerP.request!x -> Glue
           [] §
End Style
```

```
Configuration PrintExample
       Style ClientServerPrinting
       Instances
              U1           : Client(CSL.min())
              U2           : Client(CSL.min())
              PS           : PrintServer()
              SECUREPRINTER: Printer()
              PUBLICPRINTER: Printer()

              CONN1        : PrintConnector()
              CONN2        : PrintConnector()
              CONN3        : PrintConnector()
              CPRINTS      : PrintConnector()
              CPRINTP      : PrintConnector()

       Clearances
              U1                         : EVERYONE
              U2                         : PRIVATE
              U2.PrintServiceP           : EVERYONE
              PS.PrintPublic             : EVERYONE
              PS.PrintSecret             : PRIVATE
              PS.OutputPublic            : EVERYONE
              PS.OutputSecret            : PRIVATE
              CONN1, CONN3, CPRINTP      : EVERYONE
              CONN2, CPRINTS             : PRIVATE
              SECUREPRINTER              : PRIVATE
              PUBLICPRINTER              : EVERYONE

       Attachments
              U1.PrintServiceP as CONN1.ClientP
              PS.PrintPublic  as CONN1.ServerP
              U2.PrintServiceS  as CONN2.ClientP
              PS.PrintSecret as CONN2.ServerP
              U2.PrintServiceP  as CONN3.ClientP
              PS.PrintPublic as CONN3.ServerP
              PS.OutputPublic  as CPRINTP.ClientP
              PUBLICPRINTER.Receive as CPRINTP.ServerP
              PS.OutputSecret  as CPRINTS.ClientP
              SECUREPRINTER.Receive as CPRINTS.ServerP

End Configuration
```