8.1 Uzunluk kodlamasının üç biçimi

PER, BER kısa ve uzun belirli uzunluk ve belirsiz uzunluk biçimlerinin bir eşdeğerine sahiptir, ancak bir takım önemli farklılıklar vardır ve kısa belirli biçim dışında kodlamalar BER ile aynı değildir .

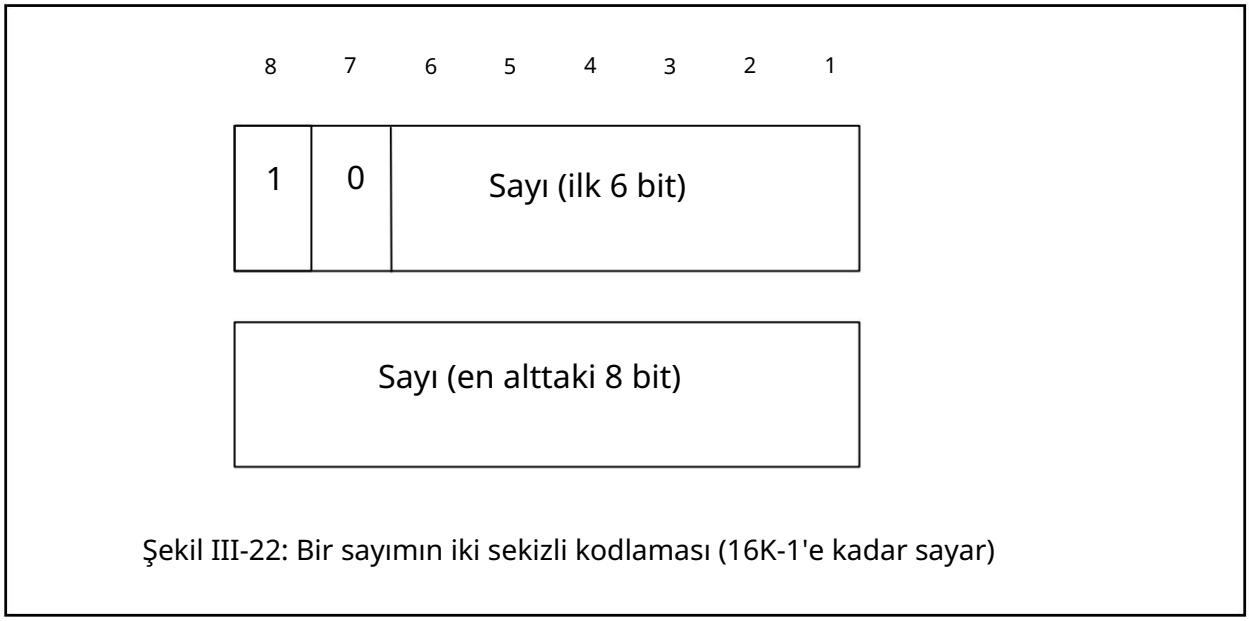
BER yaklaşımına oldukça benzer, ancak farklı ve biraz daha az ayrıntılı.

Bu madde, potansiyel olarak sınırsız olan bir sayımın gerekli olduğu durumlarda uzunluk belirleyicileri için kullanılan formu açıklar. Bu genellikle yalnızca dizelerin uzunluğu, SEQUENCE OF ve SET OF yineleme sayıları veya tamsayıların boyutu üzerinde PER-visible kısıtlamaları olmadığında geçerlidir.

Bu tür kısıtlamaların olduğu yerlerde, PER çok daha optimize edilmiş bir uzunluk alanına (daha sonra açıklanacaktır) sahip olacaktır veya hiç uzunluk alanına sahip olmayacaktır.

BER'den ilk önemli fark, PER'nin saydığı şeydir. (BER her zaman içerikteki sekizli sayısını sayar). PER, bir BIT DİZGİSİ değerindeki bit sayısını, bilinen çarpan karakter dizisi değerlerindeki soyut karakterleri, bir SEQUENCE OF veya SET OF içindeki yineleme sayısını ve diğer tüm durumlarda sekizlileri sayar. Uzunluk determinantındaki sayılardan bahsediyoruz .

Şekil III-21: Bir sayımın tek sekizli kodlaması (127'ye kadar sayar)

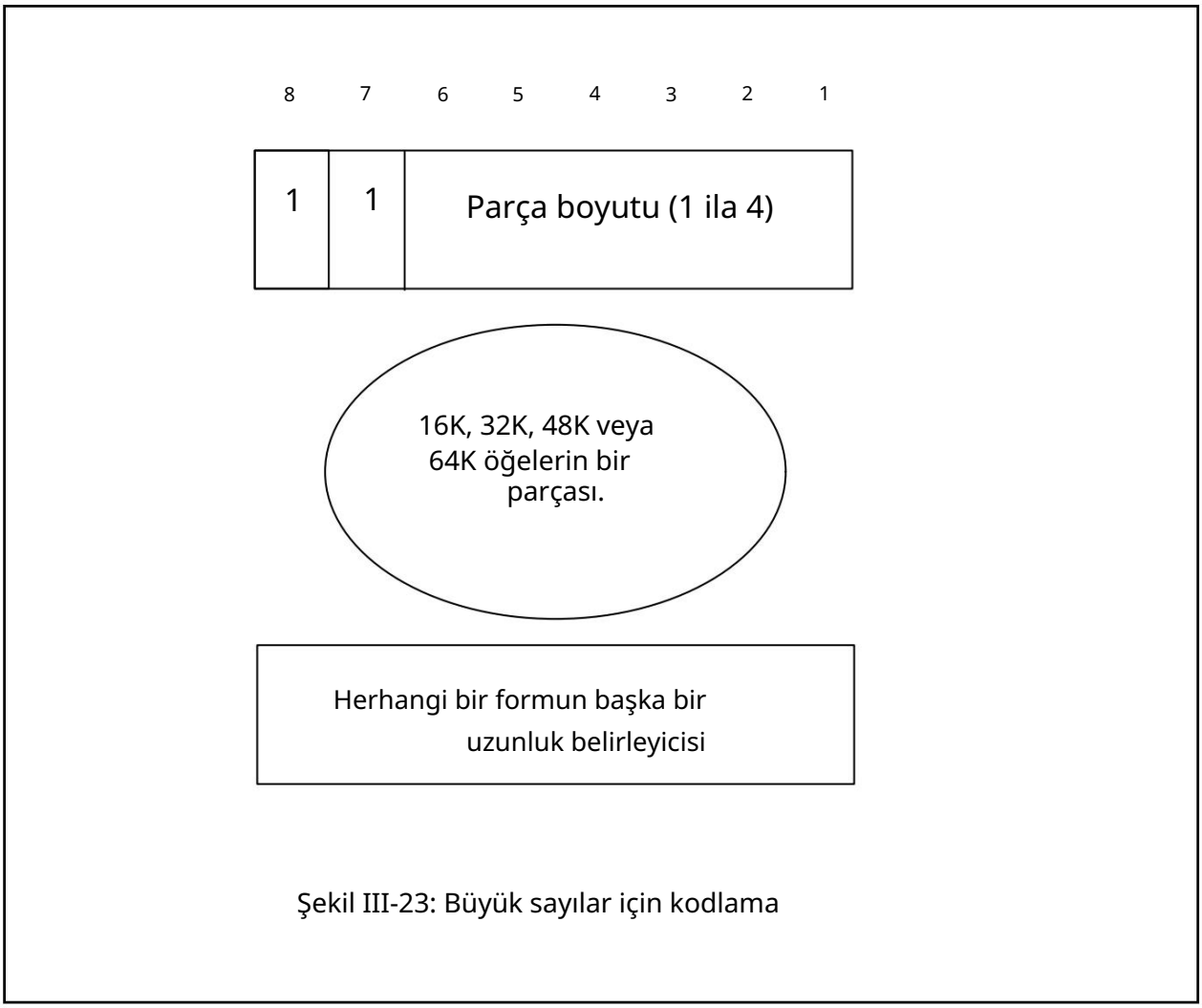


Şekil III-22: Bir sayımın iki sekizli kodlaması (16K-1'e kadar sayar)

Şekil III-21 ila III-23, uzunluk belirleyici için üç kodlama biçimini göstermektedir.

İlk formda (BER kısa formuna karşılık gelir, ancak PER bu terimi kullanmaz), BER ile aynı kodlamaya sahibiz, kodlama sekizli hizalanmış bir bit alanına yerleştirildi (başka bir deyişle, olacak ALIGNED türevlerinde dolgu bitleri). Sekizlinin üst biti sıfıra ayarlanır ve sekizlinin geri kalanı sıfırdan 127'ye kadar sayma değerlerini kodlar.

İkinci formda (kabaca BER uzun tanımlı forma karşılık gelir), her zaman tam olarak iki sekizli uzunluk determinantı vardır. Birinci sekizli, birinci biti 1'e ve ikinci biti sıfıra ayarlar ve bu iki sekizlinin geri kalan 14 biti, 128'den 16K-1'e kadar sayım değerlerini kodlar.



Şekil III-23: Büyük sayılar için kodlama

Üçüncü biçim (kabaca BER belirsiz biçimine karşılık gelir, ancak çok farklı bir mekanizma ile), ilk iki bitin 1'e ayarlandığı bir ilk sekizliye sahiptir. Kalan altı bit, 1'den 4'e kadar olan değerleri kodlar (sağa dayalı) değer "m". Bu sekizli iki şey söylüyor:

> İçeriğin "m" çarpı 16K bitlerinin, yinelemelerinin, soyut karakterlerinin veya sekizlilerinin ardından geldiğini söylüyor.
>
> İçeriğin bu parçasından sonra, içeriğin geri kalanı veya başka bir parça için başka bir uzunluk alanı (üç formdan herhangi biri) olacağını söylüyor.

PER, her parçanın olabildiğince büyük olmasını gerektirir, bu nedenle "m" seçiminde kodlayıcının seçenekleri yoktur. Prensipte izin verilen en büyük "m"nin çok daha büyük hale getirilebileceğine dikkat edin (kodlamak için altı bit vardır), ancak PER tasarımcıları parçalamayı en fazla 64K (4 kez 16K) parçaya uzun süre zorlamayı seçtiler. sekizli dizeler vb.

Şekil III-24, 5, 130, 16000, 32768 ve 99000 sayım değerleri (örneğin bir DİZİ için) için kodlamayı (ikili olarak) gösterir. Bir veya daha fazla dolgu bitinin eklenmesi bir "P" ile gösterilir, uzunluk belirleyicisinin önüne "L:" ve içerik parçalarının önüne "C:" eklenir (bu bölüm boyunca kullanılan bir kural).

```
    5:  L:P00000101 C:(5 içerik)
  130:  L:P10000000 10000010 C:(130 içerik)
16000:  L:P10111110 10000000 C:(16000 öğe içerik)
32768:  L:P11000010
        C:(32 bin içerik)
        L:P00000000
99000:  L:P11000100
        C:(64 bin içerik)
        L:P11000010
        C:(32 bin içerik)
        L:P10000010 10111000
        C:(696 öğe içeriği)
```

Şekil III-24: Çeşitli yineleme sayılarını kodlama

Şekil III-24'te parçalanma elde ettiğimizde, parçalar aynı türden 16K değerlerin katlarını kodlayacak olsa da, yinelenen türün uzantıları varsa her bir değer için kodlamaların aynı uzunlukta olması gerekmediğine dikkat edin, bu nedenle dolgu bitleri tüm bu uzunluk belirleyicileri sekizli hizalanmış olarak belirtildiğinden, bir parçadan sonra uzunluk belirleyicisinden önce tekrar gerekli olacaktır.

8.2 "Normalde küçük" değerlerin kodlanması

Sınırsız, evet, ancak genellikle küçük - PER optimizasyonu.

PER, potansiyel olarak sınırsız olan sayımlar için başka bir kodlamaya sahiptir. Bu kodlama, kodlanması gerekebilecek değerler üzerinde herhangi bir üst sınır olmamasına rağmen, değerlerin "normalde küçük" (ve tümü sıfır veya pozitif) olmasının beklendiği durumlarda kullanılır, bu nedenle bu, " olarak tanımlanır. normalde küçük, negatif olmayan bir tam sayıyı kodlama".

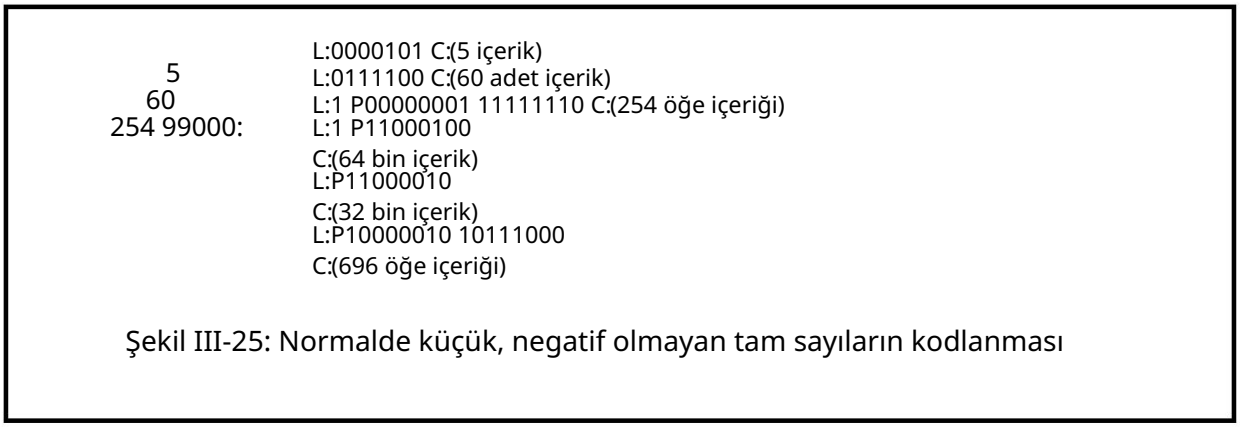
Bu durum, kökte olmayan bir seçim alternatifi için bir seçim indeksini kodlamak için uygulanır - Sürüm 2'de milyonlarca ek seçenek olabilir ve Sürüm 1 sisteminin kaç tane olduğu hakkında hiçbir fikri yoktur, ancak aslında olması pek olası değildir. birkaç taneden fazla.

İkinci bir uygulama, kökün dışında kalan ve yine olası değerlerin sınırsız olduğu ancak genellikle küçük olacakları sıralanmış bir türdeki değerleri kodlamaktır.

Her iki durumda da, değeri sınırsız bir tamsayı değeri olarak kodlamak (yukarıdaki gibi - genellikle 1'e ayarlanmış - bir sekizli hizalanmış uzunluk alanı ve bir sekizlinin bir tamsayı kodlaması gerektirir) en uygun değildir. Bu durumda belirtilen kodlama, bunun yerine, üst bit sıfıra ve diğer altı kodlama değeri 63'e ayarlı olarak yalnızca yedi bit (sekizli hizalı değil) kullanmaktır. Böylece, sekizli hizalamadan kaçınırız ve yalnızca yedi bit kullanırız, on altı değil Neden sekiz bit yerine yedi bit kullanıyorsunuz? Bu kodlamanın genellikle bir uzantı bitinin ardından görüneceğini unutmayın, bu nedenle ikisi birlikte bize tam olarak sekiz bit verir ve başlangıçta hizalamamız varsa, hala ona sahibiz.

Küçük değerler için optimize etmenin elbette bir cezası var! Normalde küçük, negatif olmayan tam sayının aslında 63'ten büyük olduğu ortaya çıkarsa, o zaman bire bir bitlik bir bit alanı kümesi ekleriz, ardından yukarıda açıklandığı gibi bir genel uzunluk alanı tarafından minimum sekizlilere kodlayan bir pozitif tamsayı gelir. .

Şekil III-25, bir sayımın 5, 60, 254 ve 99000 değerleri için normalde küçük, negatif olmayan bir tam sayı olarak kodlanmasını göstermektedir. bu durum için kod sağlamak - basitçe "desteklenmiyor" demek - yakalanma olasılığı çok düşüktür! Bununla birlikte, belirtim eksiksizdir ve ne kadar büyük olursa olsun herhangi bir değeri kodlayacaktır.) ilk iki vaka



Şekil III-25: Normalde küçük, negatif olmayan tam sayıların kodlanması

8.3 Sınırsız sayıların kodlamaları hakkında yorumlar

Her şey önemli mi? BER daha basit mi (veya sadece daha tanıdık mı)? Yargılamalısın!

PER'deki parçalanma mekanizması, iç içe TLV yapılarına bağlı değildir ve herhangi bir içerik kodlamasına ve özellikle sınırsız tamsayıların kodlamalarına uygulanabilir. 64K parça sayısı sınırsız olduğu için, PER gerçekten sonsuz büyük tamsayıları kodlayabilir, ancak BER'in dayattığı gerçek sınırın tüm pratik amaçlar için alakasız olduğunu zaten gördük. Bununla birlikte, PER'nin parçalanma mekanizması, özellikle kodlayıcı seçeneklerinin olmaması, muhtemelen BER'inkinden daha basittir.

PER kodlamasının BER'e göre ana avantajı, uzunluk alanlarının genellikle iki sekizli olması ve kısa form kullanılarak 128'den küçük sayımların yapılması gerekmesidir . BER ile, üç sekizli uzunluk alanlarına (uzun tanımlı form) izin verilir (ve bazı uygulamalar bunları her zaman kullanır), hatta beş sekizli içerik uzunluğu için bile. Bu, bu tür uygulamalar için büyük bir ayrıntı yüküdür.

Normalde küçük, negatif olmayan tam sayıların kodlanmasının ana avantajı, bunların (genellikle) dolgu bitleri olmadan bir bit alanına kodlamalarıdır. Değer çok büyük olursa (uygulamada gerçekleşmesi pek olası değildir), genel uzunluk kodlaması üzerinden yalnızca bir bitlik ek bir ceza vardır.

9 İSTEĞE BAĞLI bit haritasının ve SEÇİM dizininin kodlanması.

Bunlar "jenerik" kodlamalarımızın sonuncusu. Bu maddeden sonra, belirli türlerin tam kodlamasını ele alıyoruz.

9.1 İSTEĞE BAĞLI bit haritası

Bir diziyi veya ayarlanan değeri kodlarken, PER'nin bir bit alanına bir önsöz kodladığını zaten biliyoruz,

her İSTEĞE BAĞLI veya VARSAYILAN öğe için bir bit ile (İSTEĞE BAĞLI veya VARSAYILAN öğe yoksa sıfır bit). Kodlamada öğenin bir değeri varsa bit bire ayarlanır, aksi takdirde sıfıra ayarlanır. Bunu her bir öğenin kodlaması takip eder.

Bu, kökteki öğeler için geçerlidir. Uzatma ekleri olan öğeler için ekleme noktasında benzer bir bit haritası kullanılır, ancak bu daha sonra açıklanacaktır.

Normal şartlar altında, bu bit haritası için bir uzunluk belirleyicisi yoktur (çünkü hem gönderen hem de alıcı, uzunluğunu tür tanımından bilir), ancak eğer (ve asla gerçekleşmeyecekse, bu nedenle bir araçtan "desteklenmeyen" bir yanıt alınırsa) Tamam!) bit haritasının uzunluğu (isteğe bağlı veya varsayılan öğelerin sayısı) 64K'yı aşıyor, ardından bir uzunluk belirleyicisi dahil ediliyor ve bit haritası parçaları 64K parçalara ayrılıyor.

9.2 SEÇİM dizini

Bir SEÇİM değeri için yine bir giriş vardır. Tür genişletilebilir değilse veya değer kökteyse, bu seçim dizininde bir üst sınıra sahibiz (ve sıfırın alt sınırı - seçim dizini, açıklandığı gibi en düşük etiket değerine sahip alternatifle sıfırdan başlar) daha erken). Bu değer, hem üst hem de alt sınırı olan kısıtlanmış bir tamsayı değeri olarak kodlanmıştır. Aşağıda, örneğin 0 ila 15 (CHOICE türünde en fazla 16 alternatif) aralığıyla sınırlandırılmış tamsayı değerlerinin dört bitlik bir bit alanına kodlandığını göreceğiz.

Seçilen alternatif kökün dışındaysa, o zaman "uzantı bitimizi" bir bit alanında (daha önce açıklandığı gibi) bire, ardından bir bit alanında normalde küçük negatif olmayanları kodlayan (genellikle) yedi bit alırız. Uzatma eklemeleri içindeki alternatifin indeksi olan tam sayı (ilk toplama alternatifi sıfır değeri olarak alınır). Bir SEÇİM'de sürüm parantezlerine izin verilirken, bunların varlığının kodlamada hiçbir fark yaratmadığına, kodlamanın yalnızca SEQUENCE ve SET için etkilendiğine dikkat edin.

Bir sekizli sınırında başlarsak, tam olarak sekiz bit eklediğimize ve bir sekizli sınırında kalacağımıza ve bu kodlamalarda herhangi bir sekizli hizalamayı zorlamadığımıza dikkat edin. Bu kodlamaların çizimleri, seçim değerlerinin eksiksiz kodlamasını açıklayan Madde 16'da verilmiştir.

10 NULL ve BOOLEAN değerlerinin kodlanması.

Sonunda! Aslında bir şeyi nasıl kodlarsınız? Biz

Bunlar kolay. Görünür PER kısıtlamaları uygulanamaz ve isteğe bağlılık, bit haritasına göre sıralanır.

NULL için sıfır bit. Tüm ihtiyacın olan bu. BOOLEAN için bir bit - DOĞRU için 1'e ve YANLIŞ için sıfıra ayarlayın. Ve tabii ki ALIGNED versiyonunda herhangi bir dolgu biti yoktur.

11 INTEGER değerlerinin kodlanması.

Unutmayın - aşağıda kısıtlamalardan bahsettiğimizde, daha önce tartışıldığı gibi yalnızca PER-görünür kısıtlamalarla ilgileniyoruz.

Bu tartışmanın tek ilginç kısmı, "minimum bitler" kullanılma eğiliminde olduğunda, kısıtlı tamsayıların kodlanmasıyla ilgilidir. Kısıtlanmamış tamsayılar için, standart uzunluk determinantını ve minimum sekizlilerde bir kodlamayı elde ederiz. Bununla birlikte, ALIGNED ve UNALIGNED varyantları arasında farklılıklar vardır (doldurma bitlerinin eklenmesi veya eklenmemesi ışında).

11.1 Kısıtlanmamış tam sayı türleri

INTEGER türlerinin kodlanmasında en önemli olan değer üzerinde bir alt sınırın olup olmadığıdır. Olmazsa, sekizli sayısını içeren bir genel uzunluk belirleyicisiyle (daha önce açıklandığı gibi) minimum sekizlileri işaretli bir sayı olarak kodlarız. Yani:

Alt sınır yoksa, genel bir uzunluk belirleyicisiyle (tüm varyantlar) minimum sekizlilere 2'ye tümleyen bir kodlama elde ederiz.

integer1 INTEGER ::= 4096 integer2
INTEGER (MIN .. 65535) ::= 127 integer3 INTEGER (MIN ..
65535) ::= -128 tamsayı4 INTEGER (MIN .. 65535) ::= 128

tümü "kısıtlanmamış" olarak tanımlanır ve şu şekilde kodlanır (varsa - uzunluk belirleyicisinden önce "L:" ve varsa içerik kodlamasından önce "C:" ile):

tamsayı1:	L:P00000010 C:00010000 00000000
tamsayı2:	L:P00000001 C:01111111
tamsayı3:	L:P00000001 C:10000000
tamsayı4:	L:P00000010 C:00000000 10000000

Bu, BER ile aynıdır (127 sekizliye kadar olan değerler için), ancak tanımlayıcı sekizli içermez.
UNALIGNED varyantında P bitlerinin asla eklenmediğini unutmayın.

11.2 Yarı kısıtlı tamsayı türleri

Bir alt sınıra sahip olduğumuzda (ki bu genellikle sıfır veya bir olur, ancak herhangi bir şey olabilir), kodlanacak değer olarak tabandan ofseti kullanarak yalnızca pozitif bir değer kodlamamız gerekir.

Alt sınırdan (pozitif) ofseti kodlayın.

Kısıtlanmamış tamsayı türlerine gelince, kodlama sekiz bitin minimum gerekli katına ve sekizli sayısını sayan bir uzunluk belirleyicisine yapılır. Yani:

integer5 INTEGER (-1.. MAX) ::= 4096 integer6 INTEGER
(1 .. MAX) ::= 127 integer7 INTEGER (0 .. MAX) ::= 128

olarak kodlayın:

tamsayı5:	L:P00000010 C:00010000 00000001
tamsayı6:	L:P00000001 C:01111110
tamsayı7:	L:P00000001 C:10000000

(Tamsayı7'nin kodlamasını tamsayı4'ün kodlamasıyla karşılaştırın.)

11.3 Kısıtlı tamsayı türleri

PER'nin "mantıklı olanı yapmak" için en çok çabaladığı, hem alt hem de üst sınıra sahip tamsayıların kodlanmasıdır. Ancak UNALIGNED varyantını savunanların belirlediği "mantıklı şey", ALIGNED varyantını savunanların belirlediği "mantıklı şey"den farklı çıktı, dolayısıyla yaklaşımlar pek de aynı değil. Hangisi en mantıklı, yargılamalısın !

UNALIGNED varyantları bit (aralık için minimum bitler) ve ALIGNED versiyonu arasındaki farklar - aralığa bağlı olarak birkaç farklı kodlama.

Standart, "aralığı" üst sınır eksi alt sınır artı 1 olarak tanımlayarak değerlerin "aralığından" bahseder. Dolayısıyla (0..3) kısıtlamasının dört "aralığı" vardır. Bu nedenle, "aralık" esasen üst ve alt sınırlar arasındaki (ve dahil) değerlerin toplam sayısı olarak tanımlanır.

"Aralık" bir ise, yalnızca bir değer mümkündür. Bunun pratikte meydana gelmesi muhtemel değildir, ancak kodlama doğal olarak daha geniş aralıkların işlenmesinden kaynaklanır ve NULL'un işlenmesine benzer: kodlamada hiç bit yoktur!

Önce ortaya çıkabilecek tüm durumları açıklıyoruz, ardından örnekler veriyoruz.

Daha geniş aralıklar için, HİZALANMAMIŞ durumu tanımlaması en kolay olanıdır. Aradaki tüm değerleri desteklemek için alt sınırdan ofseti gereken minimum bit sayısına kodlar. Yani (1..3) - veya (6..8) veya (11..13) veya (-2..0) - kısıtlamasının üç aralığı vardır ve değerler şu bit alanına kodlanır: 2 bit (4'lük bir aralıkta olduğu gibi). (0..65535) kısıtlaması, tüm değerlerin kodlamalarını tam olarak 16 bit olarak üretecektir , vb. UNALIGNED varyantlarında hiçbir zaman dolgu biti olmadığını unutmayın, bu nedenle bu son durumda SEQUENCE OF INTEGER (0..65535) kodlamasındaki ardışık tamsayıların tümü 16 bit uzunluğunda olacaktır, ancak tümü 3. bitten () başlayabilir. diyelim) bir sekizli.

ALIGNED durumu biraz daha çeşitlidir!

Aralık 255'ten küçük veya eşitse (not: 255, 256 değil), bu durumda kodlama, aralığı kodlamak için gereken minimum olan bir bit alanına yapılır ve dolgu bitleri olmaz. Bununla birlikte, aralık 256 ise - örneğin, kısıtlama (0..255) veya (1..256) olabilir - o zaman değer sekiz bit olarak kodlanır, ancak sekizli hizalanmış bir alana giderler - şunu elde ederiz: gerekirse dolgu bitleri.

Aralık 256'dan büyük ancak 64K'dan büyük değilse, iki sekizli (sekizli hizalanmış) elde ederiz.

İki sekizliyi aşmamız gerekirse (aralık 64K'dan fazladır), her değeri (alt sınırdan pozitif bir tamsayı ofseti olarak) gerekli minimum sekizli sayısına kodlarız (sıfır her zaman bir sekizli olarak kodlanır hariç) sıfırlar, sıfır bitlerine değil, bu nedenle her zaman minimum bir sekizliye sahibiz) ve kullanılan sekizli sayısını veren bir uzunluk determinantının önüne ekleyin. Ancak bu durumda, daha önce açıklanan genel uzunluk belirleyicisi kullanılmaz ! Bunun yerine, bu sekizli sayımının alabileceği değer aralığına bakarız (alt sınır birdir, unutmayın, çünkü sıfır bir sekizliyi kodlar) ve uzunluk değerini pozitif bir sayıyı kodlamak için gereken minimum bit sayısıyla kodlarız. bu aralık, birden dengelenir.

Bazı örnekler verelim. Aşağıdakiler doğru değer gösterimi değildir - örneklerin kompaktlığı için bir değer, sonra bir virgül, sonra başka bir değer vb. veririz ve aynı şekilde kodlamaları ayırmak için virgül kullanırız.

integer8 INTEGER (3..6) ::= 3, 4, 5, 6 integer9 INTEGER (4000..4254) ::=
4002, 4006 integer10 INTEGER (4000..4255) ::= 4002, 4006 integer11 INTEGER
(0 ..32000) ::= 0, 31000 integer12 INTEGER (1..65538) ::= 1, 257, 65538

aşağıdaki gibi kodlayacaktır:

tamsayı8		C:00, C:01, C:10, C:11
tamsayı9		C:00000010, C:00000110
tamsayı10		C:P00000010, C:P00000110
tamsayı11		C:P00000000 00000000, C:P01111001 00011000
tamsayı12 (HİZALANMAMIŞ)		C:0 00000000 00000000, C:0 00000001 00000000, C:1 00000000 00000001
	(HİZALI)	L:00 C:P00000000, L:01 C:P00000001 00000000, L:10 C:P00000001 00000000 00000001

Uzunluk belirteci olmayan yerlerde, alanın türün tüm değerleri için aynı boyutta olduğunu ve tür notasyonundan çıkarılabileceğini göreceksiniz. (Bu doğru olmasaydı, PER bir engelleme belirtimi olurdu!) Alan boyutunun değiştiği yerde, kod çözücünün alanın boyutunu bilmesi için bir uzunluk belirleyicisi kodlanır ve uzunluk belirleyicisinin uzunluğu tüm değerler için aynıdır. ve yine tip tanımından türetilebilir. Daha önce belirtildiği gibi, bunlar bir kodlayıcı ve kod çözücünün birlikte çalışabilmesi için gerekli koşullardır. Bu örnekleri inceleyin!

Kısıtlanmış bir tamsayının ALIGNED varyantını kodlamak için bir (ve son) durum daha vardır: Tamsayı değer aralığını kodlamak için gereken sekizli sayısı 64K'yı aşarsa ..... Devam etmeli mi? Bu pratikte asla ortaya çıkmayacak ! Ancak öyleyse, genel bir uzunluk kodlaması kullanılır ve daha önce tartışılan parçalama prosedürleri devreye girer.

11.4 Ve eğer tamsayı üzerindeki kısıtlama genişletilebilir ise?

Burada yeni veya beklenmedik bir şey yok. Genişletilebilir türleri kodlamanın ilkeleri zaten tartışılmıştır.

> Ön tarafta olağan bir bit, kökteyse kısıtlı bir kodlama ve aksi takdirde kısıtlanmamış bir kodlamadır.

Ama bazı örnekler verelim:

integer13 INTEGER (MIN .. 65535, ..., integer14 INTEGER (-1..MAX, ..., 65536 .. 4294967296) ::= 127, 65536
integer15 INTEGER (3..6, ..., 7, 8) ::= 3, 4, 5, 6, 7, 8 integer16 INTEGER (0 ..2^32) ::= 4096, -8
(1..65538, ..., 65539) ::= 1, 257, 65538, 65539

şu şekilde kodlayacaktır ("uzantı biti" netlik için önüne "E:" konur):

```
tamsayı13:                      E:0 L:P00000001 C:01111111,
                                E:1 L:P00000011 C:00000001 00000000 00000000
tamsayı14:                      E:0 L:P00000010 C:00010000 00000001,
                                E:1 L:P00000001 C:11111000
tamsayı15:                      E:0 C:00, E:0 C:01, E:0 C:10, E:0 C:11,
                                E:1 L:P00000001 C:00000101,
                                E:1 L:P00000001 C:00001000
tamsayı16: (HİZALANMAMIŞ) E:0 0 00000000 00000000,
                                E:0 0 00000001 00000001,
                                E:0 1 00000000 00000001,
                                E:1 L:00000011 C:00000001 00000000 00000010
                  (HİZALI)      E:0 L:00 C:P00000000,
                                E:0 L:01 C:P00000001 00000000,
                                E:0 L:10 C:P00000001 00000000 00000001,
                                E:1 L:00000011 C:00000001 00000000 00000011
```

Tamam - Artık her şeyi biliyorsunuz! Zor değil, ama hatırlanması gereken birçok durum var. BER'e geri dön! Diğer tüm türler çok daha basittir! Hiç şüphesiz bu kısım üzerine notlar yazmak isteyeceksiniz ve sınavınızın Açık Kitap sınavı olmasını umarsınız! Ancak şimdiye kadar (eğer bu kadar ileri gittiyseniz!) PER kodlamalarında yer alan ilkeleri kesinlikle çok iyi anlamış olmalısınız.

12 Kodlama SAYILANDIRILMIŞ değerler.

Öncelikle, genişletilebilir olarak işaretlenmemiş numaralandırılmış bir türün kodlamasını ele alıyoruz (ve hatırlayın, kökte bulunan bir değer için genişletilebilir bir türün kodlaması, sıfıra ayarlanmış bir uzantı bitinin önünde olması dışında tamamen aynıdır). Kök dışındaki numaralandırmaların kodlanması daha sonra ele alınacaktır.

Bir numaralandırmayla ilişkili sayısal değer her zaman üstte ve altta sınırlanmıştır. Ayrıca, numaralandırmaları artan düzende sıralamak (bazılarının negatif ilişkili değerleri olsa bile) ve ardından her numaralandırmayı sıfırdan yukarıya doğru yeniden numaralandırmak mümkündür.

BER'de numaralandırılmış değerler, tür tanımında verilen ilişkili sayısal değeri kullanarak benzer tamsayıları kodlar, ancak PER'de kısıtlı oldukları bilinir ve normalde küçük ilişkili sayısal değerlere sahiptir, bu nedenle daha basittir!

Bu bize bir alt ve bir üst sınırı olan kompakt bir tamsayı değerleri kümesi (sayım dizini adı verilir) verir. Numaralandırılmış türün herhangi bir değeri artık karşılık gelen kısıtlanmış tamsayı gibi kodlanır.

Prensip olarak, tüm olası kısıtlı tamsayı kodlamaları mümkündür, ancak pratikte, numaralandırılmış türlerin tanımları hiçbir zaman birkaç on numaradan fazlasını içermez - genellikle çok daha azdır, bu nedenle, esasen numaralandırma indeksini şuna eşit boyutta bir bit alanına kodluyoruz: indeks aralığı ile başa çıkmak için gereken minimum.

Numaralandırma genişletilebilir ise, kökün dışındaki numaralandırmalar yine ilişkili sayısal değerlerine göre sıralanır ve yeniden sıfırdan başlayarak kendi numaralandırma dizinleri verilir.
(Unutmayın, uzantılar biti, kodlanmış bir değerin kök olup olmadığını tanımlar, bu nedenle belirsizlik olmaz ve sıfırdan yeniden başlamak, dizin değerlerini mümkün olduğunca küçük tutar). Kökün dışındaki bir değer için kodlama, numaralandırma dizininin daha önce açıklanan "normalde küçük, negatif olmayan bir tam sayı" olarak kodlanmasıdır.

Hiç şüphe yok ki bazı örnekler istiyorsunuz! İşte başlıyor (önce bir çıkış örneği ile!) - ve yine kısa olması için değer listelerini ve kodlamaları ayırmak için virgül kullanıyoruz:

```
enum1 SAYILANDIRILMIŞ {red(-6), mavi(20), yeşil(-8)} ::= kırmızı, mavi, yeşil
        enum2 SAYILANDIRILMIŞ {kırmızı, mavi, yeşil, ..., sarı, mor} ::=
kırmızı, sarı, mor
```

Bunlar şu şekilde kodlanır:

```
enum1: C:01, C:10, C:00 enum2: E:0 C:00,
E:1 C:0000000, (Bunlar "normalde küçük"

        E:1 C:0000001 sıfır ve bir kodlamaları.
                "P" yokluğuna dikkat edin)
```

63'ten fazla uzantı eklememiz olsaydı .... Hayır! Bunun için bir örnek vermeyeceğim. olmayacak! Kendi örneğini üret! (Bunu yapabilmeniz için size yeterince söylendi).

13 Dizilerin kodlama uzunluğu belirleyicileri vb.

Bu maddenin başlığındaki "vb", SEQUENCE OF ve SET OF'deki yineleme sayılarını ifade eder.

Yineleme sayıları için uzunluk determinantının yineleme sayısını, bit dizilerinin uzunluğu için bit sayısını, bilinen çarpan karakter dizilerinin uzunluğu için soyut karakterlerin sayısını ve diğer her şey için sekizli sayısı.

Etkili bir boyut kısıtlaması tarafından kısıtlanan bir uzunluk belirleyici, tam olarak eşdeğer bir kısıtlamaya sahip bir tamsayının kodlayacağı şekilde kodlar (yani, neredeyse - isterseniz aşağıdaki ayrıntıları okuyun!).

Bununla birlikte, bir uzunluk determinantı, etkili bir boyut kısıtlaması tarafından kısıtlanan değerlere sahip olabilir ve birçok yönden bunu, bir tamsayı değerinin (bir sayım) tamsayı üzerindeki doğrudan bir kısıtlama tarafından kısıtlandığı duruma benzer olarak görebiliriz.

Burada sadece dizilerin uzunluklarından veya yineleme sayılarından bahsettiğimize dikkat edin - tamsayı değerleri için uzunluk belirleyicisinin biçimi daha önce tamamen ele alındı (ve gösterildi). Ayrıca, PER-görünür boyut kısıtlamalarının olmadığı bir uzunluk belirleyicisinin genel durumunu daha önce tartışmıştık. Dolayısıyla, bu yan tümcede yalnızca etkili bir boyut kısıtlamasının olduğu durumdan bahsediyoruz ve önceki yan tümcelerde olduğu gibi, önce bir uzantı işaretçisi olmayan bir kısıtlama durumunu ele alıyoruz (eğer varsa kök içindeki kodlama sayıları için de geçerlidir) . bir uzantı işaretçisi).

için uzunluk kodlamalarının tartışılması, tamsayı kodlamalarının açıklaması verilinceye kadar kasıtlı olarak ertelenmiştir ve okuyucu okumaya devam etmeden önce bu açıklamayı gözden geçirmek isteyebilir.

Bir uzunluk veya yineleme sayısı temel olarak bir tamsayı değeridir, ancak her zaman aşağıda sınırlandırılır (başka bir alt sınır belirtilmezse sıfır ile), bu nedenle dizelerin uzunluklarını kodlamamız gerekirse, kavramlardan (ve text!), tamsayı türündeki değerlerin kodlamasını açıklamak için kullanılır.
Yarı kısıtlı bir sayım için (üst sınır yok), yarı kısıtlı bir sayım yapmak anlamsız olacaktır.

tamsayı değeri ("uzunluk uzunluğu" kodlamasıyla) ve bunun yerine Madde 8'de açıklandığı gibi genel bir uzunluk belirleyicisi kodlanır.

Sınırlı bir sayım için, sayımın tek bir değerle sınırlandırıldığı (örneğin, sabit uzunluklu bir dize veya bir dizideki sabit sayıda yineleme), o zaman uzunluk belirleyicisi yoktur - biz sadece içeriği kodlarız. Aksi takdirde, bir uzunluk belirleyicisine ihtiyacımız var.

Kısıtlanmış bir sayım için, izin verilen maksimum sayının 64K'yı aştığı durumlar dışında , sayı tam olarak karşılık gelen bir kısıtlanmış tamsayı kodlaması gibi kodlanır (hem HİZALALI hem HİZALANMAMIŞ sürümlerde) . Bu ikinci durumda, kısıtlama, kodlama amacıyla göz ardı edilir ve gerçek değer 64K'dan fazla bit, oktet, yineleme veya soyut karaktere sahipse, 64K parçaya (Madde 8'de açıklandığı gibi) parçalanma ile birlikte genel bir uzunluk belirleyicisi kullanılır. .

Son olarak, genişletilebilir bir kısıtlama düşünmemiz gerekiyor. Etkin boyut kısıtlaması türü genişletilebilir yapıyorsa, daha önce tartışılan genişletilebilir türleri kodlamaya yönelik genel hükümler türün tamamı için geçerlidir - uzunluk belirleyicisi için genişletilebilir bir tam sayı kodlamayız. Bu nedenle, sayımın (ve kullanılan alfabe gibi değerin diğer herhangi bir yönünün) kökte olup olmadığını söyleyerek uzantıları önceden alırız ve öyleyse sayımı kökteki boyut kısıtlamasına göre kodlarız. Değilse, uzantı biti bire ayarlanır ve genel bir uzunluk belirleyicisi kullanılır.

Özetlemek gerekirse:

PER-görünür boyut kısıtlaması olmadan veya 64K'yı aşan sayımlara izin veren bir kısıtlama olmadan, genel bir uzunluk belirleyicisi kodlarız.

Kök dışındaki soyut değerler için yine genel bir uzunluk determinantı kullanılır.

Sayım için sabit bir değer veren bir boyut kısıtlamasıyla uzunluk belirleyici kodlama yoktur.

Aksi takdirde, sayımı tam olarak eşdeğer kısıtlamaya sahip bir tamsayı gibi kodlarız.

Bunu bazı IA5String örnekleriyle gösteriyoruz, ancak aynı uzunluk determinant kodlamalarının yineleme sayıları vb. için de geçerli olduğunu unutmayın. Örneklerde, içeriklerde dolgu bitleri için "P" göreceksiniz. Bunlar, ikiden fazla karakter içeren ana türün IA5String olmasının bir sonucudur ve örnekler için BIT STRING kullanmış olsaydık (veya uzunluğu en fazla iki karakterle sınırlandırılmış bir IA5String'imiz olsaydı - daha sonra bakın) mevcut olmazdı. . Dolgu bitlerinin uzunluk belirleyicisinde gösterildiği yerde, bunlar tüm tipler için mevcut olacaktır. Kısa olması için E: ve L: alanlarını ikili olarak, C: alanlarını ise onaltılık olarak veriyoruz.

Okuyucu biraz alıştırma isterse, sonraki cevapları okumadan önce her bir değerin kodlamasını yazmayı deneyin! (Çok uzun dizeler için, içeriği parantez içindeki karakter sayısıyla belirtiriz ve kodlamayı verirken de aynısını yaparız).

Aşağıdaki değer tanımlarıyla:

```
string1 IA5String (BOYUT (6)) ::= "012345" string2 IA5String
(BOYUT (5..20)) ::= "0123456" string3 IA5String (BOYUT (MIN..7)) ::= "abc"
string4 IA5String ::= "ABCDEFGH" string5 IA5String (SIZE (0..7, ..., string6
IA5String (SIZE (65534..65535)) ::= "(65534 karakter)" string7 IA5String
(SIZE (65537)) :: = "(65537 karakter)"                    8)) ::= "abc", "abcdefgh"
```

aşağıdaki kodlamaları elde ederiz (uygunsa onaltılı veya ikili kullanarak): ©

string1: C:P303132333435 string2: L:0001
C:P30313233343536 string3: L:011 C:P616263 string4:
L:P00001000 C:4142434445464748 string5: L:011
C:P616263,

L:P00001000 C:6162636465666768
string6: L:0 C:(65534 sekizli) string7: L:P11000100
C:(65536 sekizli) L:P00000001 C:(1 sekizli)

14 Kodlama karakter dizisi değerleri.

14.1 Karakter başına bit sayısı

Dizilerin uzunluklarının kodlanmasını yukarıda tartıştık. Özetlemek gerekirse, uzunluk belirleyicisi, "bilinen çarpan" karakter dizisi türleri için soyut karakter sayısını ve diğer karakter dizisi türleri için sekizli sayısını verir.

Bilinen çarpan karakter dizilerinin kodlanması, her karakter için minimum bit sayısını kullanır, ancak ALIGNED varyantlarında hizalamayı kaybetmemek için bu sayı ikinin katına yuvarlanır.

Bilinen çarpan karakter dizisi türleri durumunda, PER'nin UNALIGNED değişkenlerinin kodlanmasında kullanılan bit sayısı, her bir karakteri açık bir şekilde temsil etmek için gereken minimum sayıdır. ALIGNED sürümler için, karakterler arasında sekizli hizalamanın kaybolmamasını sağlamak için her karakterin bit sayısı ikinin katına (bir, iki, dört, sekiz, on altı, vb.) yuvarlanır.

Kısıtlanmamış türün içerecek şekilde tanımlandığı karakter sayısıyla (ve HİZALANMAMIŞ değişkenlerdeki kodlamayı iyileştirmek için hariç tutmanız gereken sayıyla) bilinen çarpan türleri şunlardır:

Tip adı	karakter sayısı	Azaltma sayısı
		daha iyi kodlama için gerekli 64 10 31 3
IA5Dizesi	128 karakter	2**31 2**15
YazdırılabilirDize	74 karakter	
VisibleString	95 karakter	
NumericString	11 karakter	
UniversalString	2**32 karakter	
BMPString	2**16 karakter	

Diğer tüm karakter dizisi türleri için, uzunluk belirleyicisi sayımı sekizli olarak verir, çünkü her karakteri temsil etmek için kullanılan sekizli sayısı farklı karakterler için değişebilir. Bu ikinci durumda, kısıtlamalar PER-visible değildir ve her karakterin kodlaması, temel spesifikasyon tarafından belirtilendir, bu bölümün kapsamı dışındadır ve BER ile aynıdır.

Geriye kalan tek şey, bilinen çarpan karakter dizisi tiplerindeki her karakterin kodlamasını tartışmak, çünkü bu karakterlerin kodlaması etkin alfabe kısıtlamasından etkilenir (bkz. Madde 6) ve sekizli hizalanmış alanların ne zaman hizalı olup olmadığını görmek karakter dizisi kodlamaları için kullanılır.
Yine ALIGNED ve UNALIGNED varyantları arasında farklılıklar görüyoruz, ancak kodlamalar muhtemelen beklediğiniz veya kendinizin icat ettiği şeylerdir!

Bilinen çarpan karakter dize türlerinin her biri, BER kodlamasındaki sayısal değere (IA5String için ASCII değeri,

PrintableString, VisibleString ve NumericString, BMPString için UNICODE değeri ve UniversalString için Temel Çok Dilli Düzlem dışındaki karakterler için ISO 10646 32-bit değeri).
Bu değerler, kanonik bir karakter sırası sağlamak için kullanılır. Her karakteri kodlamak için kullanılan değerler, etkin alfabe kısıtlamasının izin verdiği ilk soyut karaktere sıfır, ikinciye birden vb. atanarak belirlenir. türü (bu belirlemede yalnızca PER-visible kısıtlamaları kullanılarak). n-1 değerini pozitif bir tamsayı olarak kodlamak için gereken minimum bit sayısı vardır ve HİZALANMAMIŞ varyantlarda, bu tam olarak her karakteri kodlamak için kullanılan bit sayısıdır. Örneğin:

Tip tanımı	Karakter başına bit sayısı
My-chars1 ::= IA5String (FROM ("T"))	Sıfır
My-chars2 ::= IA5String (FROM ("TF"))	Bir
My-chars2 ::= UniversalString (FROM ("01"))	Bir
My-chars2 ::= NumericString (FROM ("01234567") Üç	

Yukarıda, sınırlandırılan asıl temel türün bilinen çarpan karakter dizisi türlerinden herhangi biri olabileceğini ve sonucun aslında aynı kodlama olacağını unutmayın! Kendi karakter setinizi etkili bir şekilde tasarlarsınız ve ardından PER, her karakter için verimli bir kodlama atar.

ALIGNED varyantları için, kullanılan bit sayısı her zaman ikinin bir kuvvetine yuvarlanır - sıfır, bir, iki, dört, sekiz, on altı, otuz iki, dizi içinde sekizli hizalamanın kaybolmamasını sağlamak için.

Kodlama için bu değerlerin yeni değerlerle eşleştirilmesinin küçük bir istisnası vardır. Orijinal karakter kümesi, ortadaki (genel olarak) bazı "deliklerle" ilişkili değerlere sahiptir. Orijinal değerleri sıfırdan n-1'e kadar kompakt bir aralığa yeniden eşlemek, PER kodlamasında (hangi varyant kullanımda olursa olsun) karakter başına bit sayısında bir azalma sağlamıyorsa , yeniden eşleme yapılmaz ve orijinal ilişkili değer kodlamada kullanılır. Pratikte bu, "boşluğun" varlığının şu anlama geldiği NumericString durumu dışında, yeniden eşlemenin ALIGNED PER için ALIGNED PER'den daha olası olduğu anlamına gelir (burada karakter başına bit sayısı her zaman ikinin katıdır). her iki varyantta da (kısıtlama olmasa bile), yeniden eşleme gerçekleşir ve kodlama karakter başına maksimum dört bit'e düşürülür.

Böylece:

```
My-Boolean ::= IA5STRING (FROM ("TF"))(SIZE(1))
```

Kodlama, bir bit alanında tek bir bit olacaktır (uzunluk kodlaması olmadan) - başka bir deyişle, bir BOOLEAN kodlamasıyla aynı olacaktır!

14.2 Dolgu bitleri

ALIGNED durumunda dolgu bitlerini ne zaman alırız?
Burada, etkin boyut kısıtlaması (her değerdeki soyut karakterlerin sayısını kısıtlayan) ve etkili alfabe kısıtlamasının (her karakteri kodlamak için kullanılan bit sayısını belirleyen) kombinasyonuna bakmamız gerekir. Bunların kombinasyonu, bu sınırlandırılmış tipteki bir değer için toplam kodlama boyutunun asla on altı biti geçemeyeceği şekildeyse, o zaman dolgu biti yoktur . Karakter dizisi değeri bir bit alanına kodlanır. Bununla birlikte, 16 bitten fazlasını gerektirebilecek bazı değerler varsa, o zaman kodlama bir sekizli hizalanmış bit alanına yapılır ve hiçbir karakter bir sekizli sınırını geçemez (HİZALI durumda).

> Boyut, kodlanmış bir dize değeri asla 16 biti geçmeyecek şekilde kısıtlanırsa dolgu yapılmaz.

Kodlamaları dolgu bitleri üretmeyen bazı karakter dizileri örnekleri:

```
String1 ::= NumericString (SIZE (0..4))
String2 ::= IA5String (FROM ("TF")) (SIZE (0..16))
String3 ::= IA5String (BOYUT (0..2))
String4 ::= BMPString (SIZE (0..1))
```

Yine, maksimum "16 bit" kuralı, PER'nin pragmatik olmasının başka bir örneğidir. Sınır, 32 veya 64 bit olarak da ayarlanmış olabilirdi. Felsefe şu ki, kısa diziler için hizalamayı zorlamak istemiyoruz, ancak uzun diziler için hizalamayı dizinin başında yapmak (ve sonra onu korumak) dengede en iyi karardır.

14.3 Genişletilebilir karakter dizisi türleri

Genişletilebilir (PER-görünür kısıtlamalarla) bilinen çarpan karakter dizisi türünün kodlaması normal modeli izler - kökteyse sıfıra ayarlanan bir uzantı biti, aksi takdirde bir, ardından kök değerler için yukarıda açıklanan optimize edilmiş kodlama ve bir kodlama kökte değilsek (genel bir uzunluk determinantı ile) kısıtlanmamış türden. (Bununla birlikte, sınırlandırılmamış bir NumericString için 4 bitlik bir kodlama üretmek üzere ilişkili değerlerin eşlenmesinin hala gerçekleştiğine dikkat edin).

Burada sürpriz yok: bir uzantı biti, kökteyse optimize edilmiş kodlama, aksi takdirde genel kodlama - ancak yalnızca bilinen çarpan türleri için

Yukarıdakilerin tümü yalnızca bilinen çarpan türleri için geçerlidir. Diğer karakter dizisi türleri için hiçbir zaman bir uzantı biti yoktur, genel kodlama her zaman tüm değerler için geçerlidir.

Son olarak, bilinen bir çarpan türünün alfabe veya boyut kısıtlamaları için genişletilebilir olup olmadığına ilişkin kodlamaların belirlenmesinde herhangi bir endişe olmadığına dikkat edin. Önemli olan tek şey, PER-visible kısıtlamalarının onu genişletilebilir yapıp yapmadığı ve kök için etkili alfabe ve etkili boyut kısıtlamalarının ne olduğudur. Kodlama tamamen buna göre belirlenir.

15 Kodlama SEQUENCE ve SET değerleri.

Uzatma işaretleyicisi olmayan bir DİZİ için, önceki metin (Madde 9) kodlamayı açıklamıştır.
İSTEĞE BAĞLI veya VARSAYILAN olan her öğe için bir bit içeren, o öğenin bir değeri için mevcut bir kodlama varsa, bire ayarlanan bir ön ek (bir bit alanı olarak kodlanmış, sekizli hizalı değil) vardır. aksi takdirde sıfır. Ardından, her öğe için basitçe kodlama vardır.

Burada söylenecek çok az şey var - zaten hepsini biliyorsunuz - hay aksi! Tam olarak doğru değil - uzantı eklemelerinin kodlanmasının nasıl ele alındığını tartışmamız gerekiyor. Tek karmaşık kısım bu.

Daha önce, bir SEQUENCE ile tam olarak aynı şekilde kodlayan bir SET'in öğeleri için kanonik bir düzen sağlamak için etiketlerin kullanımını tartıştık.

Bir VARSAYILAN değere eşit değerlerin ne zaman/ne zaman mevcut olması veya olmaması gerektiğini veya bir kodlayıcı seçeneğine sahip olup olmadığımızı tartışmak için bu maddede kaldık. Ayrıca uzantı eklemelerinin kodlanma şeklini de tartışmamız gerekiyor.

Ama önce, basit bir dizi tipinin bir değerini kodlamaya bir örnek verelim. Örnek Şekil III-26'da ve kodlama Şekil III-27'de gösterilmektedir. İSTEĞE BAĞLI/VARSAYILAN bit haritasından önce "B:", içerik "C:", uzunluk belirleyicisi "L:" ve bir veya daha fazla dolgu biti daha önceki örneklerde olduğu gibi "P" ile gelir.

```
dizilim-val
        DİZİ {öğe-kodu
            öğe-adı aciliyeti        INTEGER (0..254), IA5String
                                     (SIZE (3..10))OPSİYONEL, SAYILANDIRILMIŞ {normal, yüksek}
                                     VARSAYILAN normal } ::= {öğe-kodu 29, öğe-adı "SHERRY"}
```

Şekil III-26: Kodlanacak bir dizi değeri

```
B:10                    (öğe-adı mevcut, aciliyet eksik)
C:00011011 (ürün kodunun değeri)
L:011 C:P534845525259 (öğe adının uzunluğu ve değeri)
```

Şekil III-27 Sekans değerinin kodlanması

Bu PER kodlamasının toplam uzunluğunun yedi sekizli olduğunu belirtmekte fayda var. BER'de (kodlayıcının varsayılan değerleri kodlama seçeneğini kullandığını ve her ikisi de basitlik gerekçesiyle her zaman 3 sekizli belirli uzunluk alanı kullandığını varsayarsak), toplam 24 sekizli elde ederiz. Kodlayıcı daha fazla bant genişliği bilincine sahipse ve varsayılan değerin kodlamasını atlarsa ve kısa belirli uzunluklar kullanırsa (ki bu durumda yeterlidir), BER 13 sekizli üretecektir.

15.1 VARSAYILAN değerlerin kodlanması

Burada CANONICAL-PER (tamamen standarttır) ve BASIC-PER (nadiren ortaya çıkan karmaşık durumlarda kodlayıcı seçeneklerine sahiptir) arasında bazı farklılıklar buluyoruz.

Her iki kodlama kuralı için de, kodlanacak gerçek değer "basit türler" için varsayılan değere eşitse (SET, SEQUENCE, SET OF, Sequenence of, SEQUENCE OF, CHOICE, EMBEDDED PDV, EXTERNAL veya sınırsız karakter dizesi türü olmayan herhangi bir şey olarak tanımlanır) , kodlayıcının hem CANONICAL-PER hem de BASIC-PER'de kodlamayı atlaması gerekir (her ikisi de standarttır).

Bununla birlikte, yukarıda listelenen türler için, değer varsayılan değere eşitse CANONICAL-PER yine ihmal edilmesini gerektirir, ancak BASIC-PER bunu bir kodlayıcının seçeneği olarak bırakır ve bir değerin eşitliği için muhtemelen karmaşık bir çalışma zamanı kontrolü yapmayı gereksiz kılar. VARSAYILAN değer ile.

15.2 Kodlama uzantısı eklemeleri

Genişletilebilir türleri kodlamanın genel ilkeleri geçerlidir: önde bir uzantı bitimiz var (İSTEĞE BAĞLI veya VARSAYILAN öğelerin bit haritasından önce), soyut değer kökteyse sıfıra, aksi takdirde bire ayarlanır. © OS, 31 Mayıs 1999

Uzantı eklemeleri pratikte İSTEĞE BAĞLI (veya VARSAYILAN) olarak işaretlenme eğilimindedir, ancak bu bir gereklilik değildir. Sürüm 2'de bir ekleme bu şekilde işaretlenmemişse, Sürüm 2 sistemleri her zaman eklemeleri kodlamak zorunda kalır ve uzantı biti her zaman bir olarak ayarlanır. Yalnızca sürüm 1 sistemleri bunu sıfıra ayarlardı.

Uzantı eklemeleri için değerler her zaman ekleme noktasının konumunda kodlanır ve bir kod çözücü, uzantı biti 1'e ayarlanırsa bu tür kodlamaları bekler, aksi halde değil.

İlk olarak, bir SEQUENCE'daki uzantı eklemelerini özetlemeliyiz. Bunlar, tek bir öğe (uzantı ekleme türü olarak adlandırılır) veya sürüm parantezlerinde bulunan bir öğeler grubu (uzantı ekleme grubu olarak adlandırılır) olabilir.

Bir uzantı ekleme grubunun işlenmesini (ve belirtimde açıklanma biçimini) açıklamanın en kolay yolu, okuyucunun tüm öğe grubunu ve sürüm parantezlerini zihinsel olarak tek bir İSTEĞE BAĞLI DİZİ ile değiştirmesidir. Toplama grubunun elemanları. Yalnızca bir sürücü vardır: grubun tüm öğeleri kodlamada çıkarılacaksa (hepsi İSTEĞE BAĞLI veya VARSAYILAN olarak işaretlenmiştir), o zaman tüm DİZİ için kodlama yoktur ve en dıştaki İSTEĞE BAĞLI bit haritası kaydeder onun yokluğu (Bunun bir örneği ileride verilecektir).

Şimdi sorunu, bazıları veya tümü İSTEĞE BAĞLI olarak işaretlenebilen ve dolayısıyla bir kodlamada eksik olabilen uzantı ekleme türlerinin basit bir listesine indirdik. Kökteki öğelerde olduğu gibi, bir kod çözücünün kodlamada hangi öğelerin var olduğunu ve hangilerinin olmadığını bilmesi gerekir ve bir kez daha bir bit haritası kullanılır. Bununla birlikte, bu durumdaki sorun, Versiyon 1 sistemlerinin, belirtimde kaç tane uzantı ekleme türü olduğunu bilmemesi ve dolayısıyla bit haritasının uzunluğunu bilmemesidir. Ayrıca, bu tür sistemler bir uzantı ekleme türünün isteğe bağlı olarak işaretlenip işaretlenmediğini bilemeyeceklerdir. Bu, kök öğeler için kullanılan bit haritasından iki fark üretir:

- Bit haritası, isteğe bağlı olarak işaretlenmiş olsun veya olmasın, kodlamadaki varlığını veya yokluğunu kaydeden her uzantı ekleme türü için bir bit içerir.

- Bit haritasından önce, bit haritasındaki bit sayısını veren bir sayı gelir.

Bit haritası uzunluğu için sayı, normalde küçük bir tam sayı olarak kodlanır.

Sayının normalde küçük bir tam sayı olarak kodlanmasının etkisi, uzantı eklemelerinin sayısı 64K'yı aşarsa, uzantı eklemeleri bit haritasının 64K'lık parçalara bölünmesi için yine bir hükmün olmasıdır. Eklemelerin yaklaşık bir yıldan daha kısa aralıklarla gerçekleşmesinin muhtemel olmadığı sürüm gruplarının varlığıyla, bir araçtan "desteklenmiyor" yanıtı tamamen uygun olacaktır!

Bit eşlemesini takiben, uzantı ekleme türlerinin değerini kodlarız, ancak bu durumda bir Versiyon 1 sistemi ilgili gerçek türleri bilmez ve bir uzantı eklemenin kodlamasının sonunu bulamayacaktır, bu nedenle uzantı ekleme türlerinin her biri, önceki bir uzunluk belirleyicisiyle "sarılır". Ancak durum bundan biraz daha kötü. Kod çözücünün sarılmış türü bilmediği göz önüne alındığında, uzunluk belirleyicisi ne olmalıdır?

Açıkça tek olasılık bitler veya sekizlilerdir ve sekizliler seçilmiştir.

Bu nedenle, her uzantı ekleme türü, kodlanan bir dış düzey türüymüş gibi ele alınır. Varsa, ancak sıfır biti varsa (ortaya çıkma olasılığı yoktur - örneğin bir NULL), o zaman bir bit olarak kodlar.
Daha sonra, tam bir sekizli sayısını oluşturmak için sonuna sıfır dolgu biti eklenir ve daha sonra kodlamaya bir genel uzunluk determinantı (hatırlayın, sekizli hizalıdır) tarafından eklenir.

Bu "sarmalama", bant genişliği açısından oldukça pahalı olabilir ve bu nedenle (esas olarak) "sürüm parantezleri" tanıtıldı. Bir sürüm parantezindeki tüm öğeler (optimal olarak), tek bir uzantı eklemesi olarak ele alınan bir İSTEĞE BAĞLI DİZİ öğeleri olarak kodlandığından, her öğe için bir yerine yalnızca bir "sarmalayıcı" elde ederiz.

```
my-sequence-val SEQUENCE
    {item-code INTEGER
            (0..254), item-name IA5String (SIZE (3..10))OPSİYONEL, !
              1 -- istisna işleme için bkz. paragraf 14.6 --,
               ...
            (0..254), [[ alternatif öğe kodu INTEGER (0..254), !
                                                    kod 29, öğe adı "SHERRY",


                                aciliyet yüksek, alternatif öğe kodu 45, alternatif öğe adı "PORT" }
```

Şekil III:28: Kodlama için genişletilmiş sıra değeri

"Sarmalama" aynı zamanda önemli bir uygulama maliyetine sahiptir, çünkü uzunluk sargı sayısı bilinmeden önce üretilecek uzantı ilavesinin tam kodlamasını (veya en azından bunun ilk 64K sekizlisini) ve gerekli herhangi bir dolgu bitinin yerleştirilmesini gerektirir . ve kodlanabilir.
(Bu, bir SEQUENCE'ın uzunluğunu kodlamak için belirsiz biçimden ziyade BER'de uzun belirli biçimin kullanılması sorununa benzer). Ancak, Versiyon 2 ve Versiyon 1 sistemleri arasında birlikte çalışmak istiyorsak (her şey için bir TLV yaklaşımına geri dönmedikçe) bu toparlamanın alternatifi yoktur.

```
D:1                                     (uzantı biti SET) (öğe-adı mevcut)
B:1                                     (öğe-kod değeri)
C:00011011
L:011 C:P534845525259 (öğe adının uzunluğu ve değeri)
L:000010 B:11                           (uzunluk - normalde küçük tam sayı
                                      ve uzantı bit haritasının değeri)
L:P0000001 C:10000000 (genel uzunluk ve
                                      aciliyetin dolgulu değeri) (versiyon parantez
L:P00000011                             eklemesinin genel uzunluğu) (alternatif-öğe-kodu) (alternatif-öğe-isminin
C:00101101                              uzunluğu ve değeri)
L:001 C:P504F5254
```

Şekil III-29: Genişletilmiş dizi değerinin kodlanması

Şimdi, bir uzantı ekleme türü ve bir uzantı ekleme grubu eklenmiş genişletilebilir bir DİZİ kodlama örneği için. (Bunu daha önceki dizi tipi örneğine dayandırıyoruz.) Şekil III-28, kodlanacak değeri gösterir ve Şekil III-29, kodlamayı gösterir (kullanılan notasyon, önceki kodlama örnekleriyle aynıdır).

Bu, toplam 18 sekizli verir. Yine, daha önce açıklandığı gibi en kötü durum BER kodlamasını alırsak, bu 37 sekizli verir ve en iyi durum 25 verir.

16 Kodlama SEÇİM değerleri.

Hem kök alternatifler hem de kök dışındakiler için seçim indekslerinin kodlanması daha önce tam olarak açıklanmıştır. Burada dikkat edilmesi gereken tek nokta, dizi için olduğu gibi, seçilen alternatif kökün dışındaysa, Versiyon 1 sistemi bunun sonunu bulamayacak, bu nedenle yine tam olarak kodlanmış bir "sarmalayıcımız" var. SEQUENCE veya SET'teki uzantı eklemeleriyle aynı şekilde.

Burada bu durumların her birine birer örnek veriyoruz.

Seçim türü uzantılarında sürüm parantezlerine izin verildiğini, ancak bunların kodlamayı etkilemediğini ve yalnızca insanlar için bir belgeleme yardımcısı olarak hizmet ettiğini unutmayın. Önemli olan, farklı sürüm parantezlerinde olsalar bile, her birinin farklı dış düzey etiketlere sahip olması gereken eklenen alternatiflerin listesidir.

Kodlanacak değerler Şekil III-30'da gösterilmiştir (bir otomatik etiketler ortamı varsayın) ve kodlamalar, seçim indeksi kodlamasını tanıtmak için "I:"nin kullanıldığı Şekil III-31'de gösterilmiştir.

```
Seçim örneği ::= CHOICE {normal NULL, yüksek
                         NULL, !2 -- istisna işleme için
                           bkz. para 14.6 -- orta NULL }
                           ...

birinci seçim Örnek Seçim ::= normal:NULL ikinci seçim Örnek Seçim ::= orta:BOŞ
```

Şekil III-30: Kodlama için iki seçim değeri

```
İlk seçenek:          E:0 I:0 C:              (toplam iki bit)

ikinci seçenek: E:1   C:000000                (uzantı biti seti) (normalde küçük
                                               bir tam sayı olarak dizin)

                      L:P00000001 (genel uzunluk "sarmalayıcı")
                      C:00000000 (NULL'ın dolgulu kodlaması)
```

Şekil III-31: Seçim değerlerinin kodlamaları

Bu örnekte, en kötü durum BER her iki durumda da dört sekizli ile kodlanır ve en iyi durum BER iki sekizli ile kodlanır. PER, saniyede üç sekizli aldı. Bu, PER'nin aslında BER'den daha kötü kodlamalar üretebileceği az sayıdaki durumdan yalnızca biridir, ancak bu genellikle böyle değildir!

17 SEQUENCE OF ve SET OF değerlerinin kodlanması.

Buraya eklenecek başka bir şey yok. Yineleme sayısını veren bir uzunluk belirleyicisi vardır. Bunun biçimi (SEQUENCE OF veya SET OF üzerindeki herhangi bir SIZE kısıtlamasına bağlı olarak) daha önce tam olarak tartışılmıştır.

Söyleyecek başka bir şey yok - her şeyi biliyorsunuz!

Bu türlerin, içinde bir uzantı işaretçisi bulunan bir SIZE kısıtlaması olabileceğini unutmayın. Her zaman olduğu gibi, kök dışındaki değerler boyut kısıtlaması yokmuş gibi kodlanır.

Şekiller III-32 ve III-33'te iki örnek gösterilmektedir. Sayılar, açıklama kolaylığı için kasıtlı olarak küçük tutulmuştur. Örnekte hem yineleme sayısı hem de yinelenen türün genişletilebilir olduğuna dikkat edin. SEQUENCE OF değerinin kendi kökünde olması için, yalnızca yineleme sayısının kök içinde olması gerekir. 4 tamsayı değerinin üçüncü yinelemede INTEGER kökünün dışında olması, INTEGER kodlamasında işaretlenir ve SEQUENCE OF için uzantı bitini etkilemez.

```
My-dizi-dizisi (SIZE(1..4), ...,                    4) TAM SAYI (0..3, ...,                    4)

        Değerim-1 Dizilimim ::= {1, 3, 4}

        My-value-2 My-dizisi ::= {1, 2, 3, 4}
```

Şekil III-32: Kodlama için iki SEQUENCE OF değeri

```
Değerim-1:
        E:0                                 (uzantı biti) (yineleme
        L:10                                sayısı 3)
                E:0 C:01 (değer 1)
                E:0 C:11 (değer 3)
                E:1 L:P0000001 C:00000100 (değer 4)
Değerim-2:
        D:1                                 (uzantı biti) (yineleme
        L:P00000011                         sayısı 4)
                E:0 C:01 (değer 1)
                E:0 C:10 (değer 2)
                E:0 C:11 (değer 3)
                E:1 L:P00000001 C:00000100 (değer 4)
```

Şekil III-33: İki SEQUENCE OF değerinin kodlamaları

18 GERÇEK ve OBJECT TANIMLAYICI değerlerinin kodlanması.

Kutu her şeyi söylüyor! Sekizli olarak bir sayı veren genel bir uzunluk belirleyicimiz var, ardından REAL için (hem BASIC-PER hem de CANONICAL-PER için) REAL'in CER/DER kodlamasının içerik sekizlileri (bunlar aynıdır). NESNE TANIMLAYICI kodlamaları için, spesifikasyon aslında BER kodlamasına atıfta bulunur, ancak CER/DER kodlamaları tamamen aynıdır.

Sekizli cinsinden uzunluk sayısı, içeriğin CER kodlaması

19 Açık Tip Kodlama

Dizi ve set ve seçim tiplerindeki uzantılar için bir "sarmalayıcı" sağlamak üzere bir dış düzey kodlamanın ve genel bir uzunluk belirleyicisinin biçimini tartıştık. Bir Açık Türü (herhangi bir ASN.1 türünü içerebilen bir "delik") sarmak için tam olarak aynı mekanizma kullanılır. Genel olarak, bir kod çözücüye "deliğe" - Açık Tür alanına hangi türün kodlandığını söyleyen protokol alanı, kodlamada bu alandan daha sonra görünebilir, ancak PER ile bir kod çözücü bulamayacaktır. türü bilmeden "delik" kodlamanın sonu. (Tüm tiplerin dış seviyesinde standart bir TLV sargısının olduğu ve ek sargının gerekmediği veya kullanılmadığı kontrast BER). Dolayısıyla PER'de sarmalayıcı genel durumda önemlidir ve her zaman kodlanır.

Genel uzunluk ve ardından dolgulu kodlama

PER Open Types'a bir sarmalayıcının dahil edilmesi, kesinlikle gerekli olmasa bile, bazı uygulamalar tarafından bir kodlamanın parçalarını "sarmalamak" için istismar edilmiştir.

Kolay modüler uygulamalar için bir ASN.1 türünü "sarmalamak" istiyorsanız, tür kısıtlaması olan bir Açık Tür kullanın.

Aşağıdakilerden oluşan büyük bir DİZİ öğesini ele alalım:

```
güvenlik-veri GÜVENLİK-TİPLERİ.&Türü (Tip1)
```

Bu, Açık Tür üzerindeki bir "tür kısıtlaması" örneğidir ve okuyucu, kullanışlılığının bir açıklaması için bu maddeye yönlendirilmiştir.

Soyut değerler açısından, bu tam olarak şuna eşittir:

```
güvenlik verileri Type1
```

Bununla birlikte, PER kodlaması, ikinci durumda değil, birinci durumda bir Tip1 sarmalayıcı turuna sahip olacaktır (tür kısıtlamaları PER-görünür değildir).

Bu, bir uygulamada yararlı olabilir, çünkü protokolün ana gövdesinin uygulamaya özel bir şekilde ele alınmasını sağlar, güvenlik verilerini paketlenmemiş ve işlenmemiş halde bırakır, tam bir paket olarak bazı yaygın "güvenlik çekirdeğine" iletir. hayata geçirme.

Belirleyicilerin bu tür yapıları kullandığı yerler genellikle yalnızca güvenlik alanındadır.

20 Kalan türlerin kodlanması

GeneralizedTime, UTCTime, ObjectDescriptor, hepsi bir sekizli sayımı veren bir genel uzunluk determinantı ile kodlanır ve içerikler BER veya CER ile aynıdır (sırasıyla BASIC-PER ve CANONICAL-PER için). Bunun, BASIC-PER'in basitlik açısından kanonik olmadığı dördüncü oluşum olduğuna dikkat edin - diğer üçü:

Sonunda! PER kodlamalarını açıklayan son madde. Keşke bu kitap bir internet sitesi olsaydı da kaç kişinin ta buraya kadar okuduğunu görebilseydim! Bunu yapanların aferin!

Bir tür kümesinin kodlama değerleri.

GeneralString ve ilgili karakter dizisi türlerinin kodlanması.

Bir dizi veya set tipinde (basit bir tip olmayan) bir DEFAULT öğesinin kodlanması.

Kanonik PER, elbette her zaman kanoniktir.

Bu, yalnızca "ValueSetTypeAssignment" notasyonu, yani aşağıdaki gibi notasyon kullanılarak tanımlanan türleri bırakır:

```
MyInt1 TAM SAYI ::= { 3 | 4 | 7}
MyReal1 GERÇEK ::= {0 | ARTI-SONSUZ | EKSİ-SONSUZ}
```

Bunlar şuna eşdeğerdir:

```
MyInt2 ::= TAM SAYI ( 3 | 4 | 7)
MyReal2 ::= GERÇEK (0 | ARTI-SONSUZ | EKSİ-SONSUZ)
```

Başlangıçta PER standardı bu türlerin özelliklerini göz ardı etti, ancak bu dönüşümü kullanarak kodladıklarını söyleyen bir Düzeltme yayınlandı.

21 Sonuç

Böyle bir bölümde, ne yazarın ne de bu materyalin yayınlanmasında yer alan herhangi birinin metindeki hatalardan hiçbir şekilde sorumlu tutulamayacağını vurgulamak önemli görünüyor.

Uyarı Emptor!

PER kodlamalarının tek yetkili tanımı, Standartların/Önerilerin kendisinde belirtilendir ve uygulamaları üstlenen herkes, çalışmalarını bu eğitim metnine değil, bu birincil belgelere dayandırmalıdır.

Bununla birlikte, bu metnin faydalı olacağı ve uygulayıcıların gerçek spesifikasyonları daha kolay okumalarına ve anlamalarına yardımcı olacağı umulmaktadır.

Okuyucu artık optimum kodlamalar sağlamak için PER'de kullanılan ilkeleri iyi bir şekilde kavramalıdır, ancak gereksiz uygulama karmaşıklığından kaçınmak için pragmatik kararlarla yumuşatılmalıdır.

64K'dan büyüklerse bit haritalarını parçalamak veya bir INTEGER tek bir değerle sınırlandırılmışsa sıfır bitleri kodlamak gibi bazı şeyler gereksiz yere karmaşık görünebilir , çünkü bu tür şeyler gerçek dünyada asla gerçekleşmez. Ancak bu spesifikasyonlar, genel bir uygulamanın sonucudur.

ilkesini (ve bir uygulamadaki genel kodu) daha geniş bir koşul yelpazesine taşır ve ekstra uygulama karmaşıklığı değildir.

Örneklerde, PER kodlamalarının ayrıntılarda BER'e göre nasıl önemli kazanımlar elde ettiğini ve temel spesifikasyonda kısıtlamaların mantıklı kullanımı yapıldığında daha da büyük kazanımlar elde ettiğini de gördük.

Bu bölüme gelecek sadece bir bölüm daha var (bundan çok daha kısa!). Bu, Uluslararası standartlar haline gelmeyi hiçbir zaman tam olarak başaramayan (veya henüz başaramayan!) bazı diğer kodlama kurallarını ve "kendi kodlama kurallarınızı yuvarlamanın" avantajlarını ve (esas olarak) dezavantajlarını tartışır.

Bölüm 4
ASN.1 ile ilgili diğer kodlama kuralları

(Veya: Yani özel gereksinimleriniz mi var?)

Özet:

Bu bölüm, zaman zaman yapılmış olan ASN.1 kodlama kuralları için diğer önerileri kısaca açıklamaktadır. Bunların hiçbiri şu anda ASN.1 spesifikasyonunun bir parçası olarak Uluslararası Standardizasyon yolunda değildir ve bu bölüm, entelektüel olarak meraklı olanlar dışında herkes tarafından güvenle atlanabilir. "ASN.1 nedir, onu nasıl yazarım ve onu kullanarak tanımlanmış protokolleri nasıl uygularım" ile ilgilenen çoğu okuyucunun ilgisini çekmez. Ancak, ASN.1 gösterimini farklı kodlama kurallarıyla geliştirmeye yönelik diğer girişimlerin (eksik) bir resmini verir.

Kapsama sırası zaman sıralaması değildir (bazen kapalı bir toplulukta bir fikrin tohumunun ilk ne zaman ortaya çıktığını söylemek kolay değildir), temelde rastgeledir! Kısaca şunlara değinilmiştir:

LWER - Hafif Ağırlık Kodlama Kuralları

MBER - Minimum Bit Kodlama Kuralları

OER - Sekizli Kodlama Kuralları

XER - XML (Genişletilmiş Biçimlendirme Dili) Kodlama Kuralları

BACnetER - BAC (Bina Otomasyon Komitesi) net Kodlama Kuralları

Kodlama Kontrol Spesifikasyonları (ECS)

Hiç şüphe yok ki orada gizlenen başkaları da var!

1 İnsanlar neden yeni kodlama kuralları önerir?

Temel bir iş atı olarak, BER'in iyileştirilip iyileştirilemeyeceği şüphelidir. Basit, anlaşılır ve sağlamdır. Temel "TLV" yaklaşımını sürdürürseniz, yapılabilecek çok az iyileştirme vardır.

Başlangıçta kaos vardı. Ve daha büyük Tanrılar indi ve her biri yeni bir Standart yarattı ve insanlar Standartlara taptı ve "Bize daha fazlasını verin, bize daha fazlasını verin!" dedi. Böylece daha büyük Tanrılar daha fazla Standart doğurdu ve daha fazla ve işte, bir kez daha kaos çıktı!

Ancak 1984'te daha verimli bir şekilde kodlamanın mümkün olması gerektiği açıktı.

BER'den daha fazla ve esasen PER benzeri kodlamalar üretmek için PER'nin piyasaya sürülmesinden önce veya bu süre zarfında birkaç girişimde bulunuldu. Kodlama kurallarının çoğalmasını önlemek için, PER 1990'ların başında değil, 1980'lerin sonunda geliştirilip standartlaştırılmalıydı, ama olmadı! Bu nedenle, boşluğu doldurmak için birkaç "endüstriye özgü" kodlama kuralı ortaya çıktı.

Şu anda, büyük araç satıcıları yalnızca BER ve PER'yi desteklemektedir. Belirli sektöre özgü protokoller için diğer kodlama kuralları desteği (tüm ASN.1 türleri yerine yalnızca bu protokollerde kullanılan türleri destekler), kodlamanın belirli bölümlerini gerçekleştirmek için bir yordam kitaplığı tarafından (bir ASN.1 derleyicisi tarafından değil, ancak Bölüm I Bölüm 6'da tanımlandığı ve açıklandığı gibi) mevcuttur.

Yeni kodlama kurallarının üreticileri genellikle satırda BER'den daha az ayrıntı veya PER'den daha fazla basitlik (veya her ikisi!)

Ancak bugüne kadar, ASN.1'in standartlaştırıcıları, sunulan alternatif kodlama kuralı taslaklarının hiçbirini ASN.1 paketi içinde standartlar olarak ilerletmek için yeterli değere sahip olarak değerlendirmediler.

Bu, (örneğin) PER'den daha düşük bir dengede oldukları anlamına gelmez - herkes şu anda bildiklerinizle yeniden başlarsanız PER'nin geliştirilebileceğini kabul eder - ancak kodlama kuralları için çok benzer başka bir standart sağlar. PER ve üzerinde sadece marjinal bir gelişme herhangi bir anlam ifade etmeyecektir. Araç satıcıları bunu desteklemek istemez ve elbette mevcut protokol uygulamalarının dikkate alınması gerekir. ASN.1 kodlama kuralları, dünyanın her yerinde her gün her dakika dolaşan "hat üzerindeki bitler" nedeniyle yüksek derecede atalete sahiptir (gösterim çok daha kolay değiştirilebilir).

Bununla birlikte, belirli bir endüstri için belirli bir protokolü desteklemek için biraz farklı kodlama kuralları sağlamaya yönelik girişimler devam etmektedir, genellikle bu endüstri ile ilişkili bazı danışmanlık veya yazılım şirketleri tarafından, bu kodlama kurallarının fiili standart haline geleceği umuduyla önerilmektedir. o endüstri. Bununla birlikte, bu tür kodlama kuralları, ana ASN.1 derleyici araçlarına dahil edilmelerine veya tüm endüstrilerde kullanım için ASN.1 kodlama kuralları için uluslararası standartlar olarak onaylanmalarına yol açan pazar talebini nadiren karşılar.

Protokol tasarımında yeni olan pek çok endüstrinin mesajlarını tanımlamak için ASN.1'i kullanmayı seçmesi, belki de ASN.1 notasyonunun başarısının bir işaretidir, ancak belki de NIH (burada icat edilmedi) faktörüdür. notasyonu azaltmak ya da onun için farklı kodlamalar üretmek arzusuna yol açar. Kim bilir?

2 LWER - Hafif Ağırlık Kodlama Kuralları

Standart çalışma onaylandı, ancak sonunda terk edildi - çok fazla sorun var!

Hafif Ağırlık Kodlama Kuralları ilk olarak 1980'lerin sonlarında ASN.1 derleyicileri ortaya çıkmaya başladığında önerildi ve en başından beri Deutsches Institut für Normung (DIN)'in bunların uluslararası standartlar olarak geliştirilmesine şiddetle karşı çıkmasıyla birlikte birçok tartışma konusu oldu.

LWER için öneriler, PER üzerinde önceden tarihli bir çalışmaydı ve endişe, BER'in ayrıntılarıyla değil, bir BER kodlaması yapmak için gereken CPU döngü sayısıyla ilgiliydi. ISO içinde bir Çalışma Öğesi olarak onaylandılar ve terk edildikleri 1990'ların ortalarına kadar devam ettiler (nedenlerle aşağıya bakın).

2.1 LWER yaklaşımı

Temel fikir basitti ve şu gözleme dayanıyordu:

> ASN.1 değerinizi tutan belleğin içeriğini makinenizden başka bir makineye göndermeniz yeterlidir (çekirdek içi bellek temsilini standartlaştırır). Çok basit bir yaklaşım.

Bir ASN.1 derleyicisi, bir ASN.1 türünün değerlerini tutmak için bir çekirdek içi veri yapısı için model oluşturur (bu genellikle bir dizi bağlantılı liste ve benzer yapılara işaretçilerdir), bu çekirdek içi veri yapısını bir ASN.1 kullanarak tanımlar. üst düzey programlama dili.

Çalışma zamanı desteği, daha sonra hat boyunca iletilen kodlamalar (CPU döngülerinde bir miktar maliyetle) oluşturmak için bu yapıyı ağaçta gezdirir.

Bir kod çözücü, hattın diğer ucunda (çok benzer) bir çekirdek içi yapı üretir.

Neden çekirdek içi veri yapısının içeriğini doğrudan göndermiyorsunuz? Bu özünde LWER önerisiydi.

2.2 İlerlemenin yolu kararlaştırıldı

> ASN.1 değerlerinin standart bir çekirdek temsili üzerinde anlaşın ve bunun başka bir makineye nasıl gönderileceği konusunda anlaşın. Kolay.

Erken çalışma birkaç kilit nokta üzerinde anlaşmıştır:

İlk adım, çekirdek içi veri yapılarının tanımının dayandırılacağı bir bilgisayar belleği modeli üzerinde anlaşmaktı.

İkinci adım, herhangi bir ASN.1 türünün değerlerini tutmak için bellek tabanlı çekirdek içi bir yapıyı standart hale getirmekti.

Üçüncü adım, böyle bir yapının uzak bir sisteme nasıl iletileceğini standartlaştırmaktı.

2.3 Sorunlar, sorunlar, sorunlar

Tüm bu alanlarla ilgili ciddi sorunlarla karşılaşılmıştır.

> Bilgisayarlar, bilgileri ve diğer konumlara işaretçileri tutma biçimleri bakımından aslında çok farklıdır - basit "güvercin deliği seti" modeli tam olarak tutmaz!

Bir bilgisayar belleği modeli söz konusu olduğunda, çevirici dil düzeyinde (bugün zaten kimsenin kullanmadığı), bellek, tamsayılar veya diğer adreslenebilir birimlere veya karakter dizilerine işaretçiler içerebilen adreslenebilir birimlerden oluşur (basitleştirme, ancak yapacak). Ancak bu adreslenebilir birimlerin boyutu - baytlar, 16 bit sözcükler, 32 bit sözcükler - donanım çok değişkendir.

Ve eğer bir yapı böyle bir model kullanılarak tanımlanırsa, Java gibi belirli yüksek seviyeli dillerde bulunan özellikleri kullanarak bu yapıyı kopyalamak ne kadar kolay olacaktır?

Little-endian/big-endian sorunu daha önemliydi. (Yumurtaların "küçük ucundan" mı yoksa "büyük ucundan" mı kırılması gerektiği konusunda savaşan Jonathon Swift'in Gulliver'in seyahatlerindeki karakterlerin adını almıştır). Ancak bilgisayar tabiriyle, temel donanım mimarisine bakarsınız ve aşağıdaki gibi ilerlersiniz:

Bayt adreslemeyi varsayalım ve içinde iki baytlık tamsayılar bulunan belleğinizin bir resmini çizin.

Resminize düşük adreslerden yüksek adreslere doğru bir ok koyun. (Bazıları resmi ok soldan sağa, bazıları tersi olacak şekilde çizmiş olacak. Bu önemli değil, bu sadece kağıt üzerindeki tasviri etkiliyor.)

Şimdi, her bir tamsayı için, ok yönünde karşılaştığınız ilk baytın tamsayının en önemsiz sekizlisi (bir little-endian makinesi) veya tamsayının en önemli sekizlisi (bir big-endian makinesi) olup olmadığını yazın.).

Little-endians muhtemelen soldan sağa giden oku çizmiş olacak ve büyük endians muhtemelen sağdan sola doğru çizmiş olacak, ancak yukarıda belirtildiği gibi, bu önemli değil (her ikisi de bir ayna görüntüsü çizebilirdi) onların resmi). Önemli olan, bir tamsayının yüksek dereceli sekizlisinin, düşük dereceli sekizliden daha yüksek veya daha düşük bir adres konumunda olup olmadığıdır. Tamsayılar için geçerli olanın (her zaman) adresleri (işaretçiler) tutan alanlar için de geçerli olduğunu unutmayın.

Ne yazık ki, dünyada hem big-endian hem de little-endian makineleri var!

Ve bir little-endian makinesinde bir ASN.1 değerini temsil eden bir çekirdek içi veri yapınız varsa ve bunu bir big-endian makinesine kopyalarsanız, onu kullanılabilir bir kaynaktan deşifre etmek kesinlikle hafif olmayacaktır !

Bu yüzden, LWER'in bir big-endian ve bir little-endian varyantına ihtiyacımız var ve LWER'yi yalnızca benzer (endian-bilge) makineler arasında geçiş yapıyorsanız kullanabileceksiniz, aksi takdirde BER veya PER'ye geri dönersiniz.

Ancak bunların tümü, bayt adreslemeli ve 16 bitlik tamsayılar ve işaretçilere sahip makineler varsayıyordu. Şimdi 32 bitlik tamsayıların veya yalnızca (kolayca) 16 bitlik veya 32 bitlik sözcükleri adresleyebilen (işaret edebilen) makinelerin olası permütasyonlarını düşünün .....

Aniden, LWER'in oldukça fazla varyantına ihtiyacımız var gibi görünüyor!

DIN'in işe karşı çıkmasının temel nedeni buydu - standartlar üretilse bile, bunlar yalnızca çok sınırlı makine mimarisi aileleri arasındaki transferler için yararlı olacaktı.
Ve bu düşük seviyeli bellek tabanlı mimarileri yüksek seviyeli dillerde yansıtma problemlerini ekleyin. Takım satıcılarının, isterlerse, kendi takımları aynı tezgah aralığında kendisiyle iletişim kurarken kullanılmak üzere bir LWER (destekledikleri her tezgah aralığı için ayrı olanlar) tanımlayabilirler ve ne elde edersiniz ? ? Muhtemelen LWER ile elde edeceğiniz kadar karşılıklı çalışma!

LWER'in gösterdiği şey, herhangi bir makine mimarisinden bağımsız olan kodlama kurallarını (karakter tabanlı veya ikili tabanlı) tanımlamanın önemiydi - BER veya PER gibi bir şeye sahip olma fikri doğrulandı. (Ve tabii ki karakter tabanlı kodlamalar da mimariden bağımsızdır.)

2.4 LWER'in ölümü

Bir LWER standardı üretmenin başka sorunları da vardı ama temelde PER işi yine de yaptı!

Yukarıdaki sorunlar sıralanmış olsa bile, hattan ne gönderileceği konusunda hala sorunlar vardı. Bağlantılı liste yapılarının kapladığı toplam bellek gönderildiyse, güvenlik sızıntılarını önlemek için bu toplam yığın içindeki boş belleğin sıfırlanması gerekir. Boş bellek gönderilmezse, o zaman iletim için bir tür çöp toplama veya ağaçta yürüme gerekir ki bunların hiçbiri çok hafif görünmüyordu.

Ancak sonunda LWER çalışmasını öldüren şey, kimsenin beklemediği bir şeydi.
PER uygulamaları ortaya çıkmaya başladı. PER'nin bir kodlamanın uzunluğunda yaklaşık iki kat azalma sağlaması beklenirken (yaptı), iki kat daha hızlı kodlayıp kod çözmesi tamamen beklenmedik bir durumdu! LWER'in yapmaya çalıştığı işi yaptı!

Bildikten sonra, bariz görünüyor. PER'deki tüm karmaşıklık ve CPU döngüleri, tür tanımını analiz etmek ve kodlamanın ne olması gerektiğine karar vermekle ilgilidir. Bu, ya bir el uygulayıcısının beyin döngüleridir ya da bir aracın derleyici aşamasıdır. Çalışma zamanı CPU döngülerini etkilemez.

Çalışma zamanında, bilinen bir konumdan bir tamsayı değeri almak ve bu tamsayı değerinin alttaki üç bitini (diyelim ki) bir arabellekteki bir bit konumuna eklemek çok daha hızlıdır (kodun üretildiğini varsayarsak). BER için T ve L ve V'yi (muhtemelen alt program çağrılarını kullanarak) oluşturmaktan daha fazladır.

Ayrıca kazanımlar da vardı çünkü kodlamanın boyutunu küçültürseniz, protokol yığınının alt katmanlarının kodunda harcanan CPU döngülerini de azaltırsınız.

Ve son olarak, LWER 1980'lerin ortalarından sonlarına kadar tasarlandı, ancak makineler yıldan yıla hızlandı. Kademeli olarak, kodlama/kod çözme için harcanan CPU döngüleri önemsiz ve ilgisiz hale geldi (gerçek protokoller için uygulama işleme, kıyaslandığında daha karmaşık ve zaman alıcı hale geldi).

LWER öldü. Geliştirmekle ilgili çok fazla sorun vardı ve elde etmeye çalıştığı şey artık gerekli görünmüyordu. Sonunda 1997'de terk edildi.

3 MBER - Minimum Bit Kodlama Kuralları

Hey - ASN.1 harika! Peki ya eski protokollerimiz?

MBER, yaklaşık 1980'lerin ortalarında önerildi, ancak Standartlar yolu için hiçbir zaman onaylanmadı. Bununla birlikte, ilkelerinin çoğu, PER üretildiğinde benimsenmiştir.

MBER'in arkasındaki fikir, sınır bilgisinden tam olarak yararlanmak ve "beklediğiniz gibi" kodlamalar üretmekti.

Böylece bir BOOLEAN bir bit olarak kodlanır ve INTEGER (0..7) türü üç bit olarak kodlanır.

MBER hiçbir zaman olası tüm ASN.1 türlerinin kodlamasını ele almadı (ve özellikle PER'de bir seçim dizini ve İSTEĞE BAĞLI öğeler için bir bit haritası ile çözülen sorunları ele almadı).

MBER çalışmasının ana itici gücü, bir türün ASN.1 tanımını üretmeyi mümkün kılmaktı; bu, eğer MBER o türdeki değerlere uygulanırsa, hat üzerinde mevcut bazı el-işlemcilerle tam ve tam olarak aynı bitleri üretecekti. hazırlanmış protokol üretiyordu.

Tipik olarak amaç, Bölüm I Bölüm 1 Madde 5.1'de (sekizlilerin resimleri) açıklanan teknikleri kullanan protokol tanımlarından , hattaki bitlerde herhangi bir değişiklik olmaksızın ASN.1 spesifikasyonlarına geçmekti.

(Okuyucu pekala "Neden?" diye sorabilir, ancak bu, ASN.1 gösteriminin kullanımının bir protokol mesajındaki alanları tanımlamanın oldukça iyi (açık) bir yolu olduğunun oldukça gurur verici bir kabulüydü.)

MBER hiçbir zaman uluslararası düzeyde ilerletilmedi, ancak (yukarıda belirtildiği gibi) "minimum bit kodlamaları" fikri uzun vadeli bir etkiye sahipti ve PER'ye dahil edildi.

4 OER - Sekizli Kodlama Kuralları

Bu metni yazarken, OER'nin geleceği belirsiz olduğu gibi son şekli de tam olarak belirlenmemiştir. Bu metin yalnızca, bu belirtimin yazara 1990'ların (çok) sonlarında nasıl göründüğünün bir taslağını verir.

PER (BER bitleriyle) - bir farkla!

ABD'de belirli bir endüstri sektörü için kodlama kuralları olarak ve belki de o sektördeki protokollerle kullanım için uluslararası standardizasyon için önerilmiştir. Sanayi sektörü “akıllı otoyollar” ile ilgileniyor. Sektör, yol kenarındaki cihazlar arasında ve bunlarla kontrol merkezleri arasındaki iletişim protokollerini tanımlamak için ASN.1'i kullanıyor. Bazı durumlarda, aygıtlar büyük genel amaçlı bilgisayarlardır (BER veya PER'nin kesinlikle kolaylıkla işlenebileceği yerlerde). Bununla birlikte, bazı cihazlar daha sınırlı olacak ve PER'nin (iddia edilen) karmaşıklığını kaldıramayabilir, ancak PER'nin verimliliğinin büyük bir kısmının gerekli olduğu yerlerde.

("İddia edilen" ile ilgili olarak, kodlamanın ne olması gerektiğini analiz etmek için PER'deki tüm karmaşıklığın derleme aşamasında olduğunu unutmayın. Bu yapıldıktan sonra, PER'deki gerçek kodlama BER'den daha az kod ve daha basittir. çapraz derleyici sistemi, en basit cihazlar bile PER'i işleyebilmelidir.)

OER başlangıçta PER ile yaklaşık aynı zamanlarda geliştirildi, ancak PER çalışmasından habersiz (daha sonra ona katlandı). Yazma sırasında, BER (BER uzunluk kodlamaları kullanılarak) ve PER'nin bir karışımıdır.

Sekizli hizalanmış Kodlama Kuralları adı , bir OER kodlamasının tüm öğelerinin, onları sekiz bitlik bir integral yapan dolgu bitlerine sahip olmasından kaynaklanır. Böylece INTEGER (0..7) sekiz bit (etiket yok, uzunluk alanı yok) olarak kodlanacak ve BOOLEAN sekiz bit (etiket yok, uzunluk alanı yok) olarak kodlanacak.

BER tarzı uzunluk kodlamalarının kullanımının dışında OER, PER'ye çok benzer, ancak PER'nin bazı optimizasyonlarını atlayarak (tartışmasız) daha basit bir özellik üretir.

Bu kodlama kuralları, 1999'da ISO/IEC ve ITU-T ASN.1 gruplarının ortak toplantısında değerlendirildi ve PER'nin "TAM HİZALI" bir sürümünü sağlama fikri biraz destek gördü.
Bu, mevcut UNALIGNED (doldurma biti yok) ve ALIGNED (makul olan yerlerde dolgu bitleri) değişkenlerinin yanına giderek PER ailesini bir şekilde tamamlar.

Tartışmada, PER'nin "TAM HİZALANMIŞ" bir sürümünü haklı çıkarmak için henüz yetersiz müşteri talebi olduğu ve her halükarda böyle bir PER sürümünün aslında çok sayıda farklılık nedeniyle OER uyumlu olmayacağı hissedildi (OER ve PER arasında daha az optimizasyon ve BER özelliklerinin kullanımı).

Bu yazı yazıldığı sırada, OER'nin uluslararası standardizasyonu ASN.1 standardizasyonu kapsamında ilerlememektedir.

5 XER - XML (Genişletilmiş Biçimlendirme Dili) Kodlama Kuralları

XER, ASN.1 standardizasyonuna nispeten yeni (1999'da) gelmiştir. Üzerindeki çalışmalar elektronik posta grupları aracılığıyla büyük bir hızla devam ediyor ve bu kitabın metni yayınlandıktan yaklaşık bir ay sonra ISO/IEC ve ITU-T içinde ciddi bir şekilde ele alınacak! Bu tartışmanın sonucu kesin olarak tahmin edilemez, ancak bu kitabın herhangi bir ikinci baskısında XER hakkında önemli bir bölüm bulunabileceğine dair kötü bir his var içimde!

XML'e inanan biri misiniz? Bu sizin içindir!

Pek çok okuyucu, XML'in güçlü bir buhar kafasına ve birçok destekleyici araca sahip olduğunun farkında olacaktır. XML'in ASN.1 ile birleşmesi şüphesiz her ikisi için de iyi bir şey olacaktır. Ancak XER ÇOK ayrıntılı!

XER karakter tabanlıdır ve ASN.1 öğelerinin etrafında XML başlangıç ve bitiş işaretlemesi (genellikle ASN.1 SEQUENCES veya SETS veya CHOICES öğelerinin adları olan, genellikle çok uzun olan etiketler) taşır.

XER, yalnızca bir ASN.1 SEQUENCE'ın alanlarına karşılık gelen bir şema ile yapılandırılmış bir veri tabanı sistemine bir XER kodlaması gönderebilme ve gerçek diziden bağımsız kod kullanabilme vaadini yerine getiriyor gibi görünüyor. ASN.1 SEQUENCE tanımı (ve veritabanı satıcısının yazılımının bir parçası olan), alınan değerleri veritabanına otomatik olarak eklemek için. Bu, XER'in ayrıntılarının bedeline değebilir (belki!).

6 BACnetER - BAC (Bina Otomasyon Komitesi) net Kodlama Kuralları

Bu kodlama kuralları oldukça eskidir ve PER var olmadan önce PER üretmek için çok dürüst bir girişimdi! Uluslararası standardizasyon için hiçbir zaman ASN.1 grubuna sunulmadılar ve büyük ölçüde PER tarafından üstlenildi (ancak hala kullanılıyor).

Belki de ASN.1'i kullanmaya karar veren, ancak aynı zamanda "kendi" kodlama kurallarını uygulamaya karar veren ilk endüstri sektörlerinden biri.

Bunlar yine ABD'de "akıllı binalarda" kullanılan mesajlar için bir endüstri sektörü fiili standardıdır (yukarıdaki "akıllı otoyollar" tartışmasını karşılaştırın).

BACnet kodlamaları asansörleri, ışıkları, merkezi ısıtma sistemlerini vb. kontrol etmek için kullanılır.

Teknik açıdan bakıldığında, BACnetER'in kesin kodlamalar sağlamadığı bazı ASN.1 yapıları vardır ve bunların artık standart olan PER'ye göre gerçek bir avantajları yoktur, bu nedenle (bu yazarın görüşüne göre) olası değildir. uluslararası alanda daha fazla etkiye sahip

faliyet alani, sahne.

7 Kodlama Kontrol Spesifikasyonları

ASN.1 üzerindeki çalışmadaki çok yakın tarihli (1999) bir gelişme, büyük ölçüde OER gibi kodlama kurallarının varyasyonları için gerekliliklerin dikkate alınmasından kaynaklanan, "Kodlama Kontrol Spesifikasyonları" adı verilen ASN.1 gösteriminin uzantıları için metin üretimiydi.

> Herkes BER ve PER'i değiştiriyorsa, istedikleri değişiklikleri resmi olarak belirtmek için bir meta-dilimiz olsun. İyi bir fikir?

Buradaki fikir, bir Kodlama Kontrol Spesifikasyonunun tanımının (ASN.1'den çok farklı bir notasyon kullanılarak), bir stil sayfasının bir HTML sayfasıyla veya bir HTML sayfasıyla ilişkilendirilebilmesi gibi, bir ASN.1 modülüyle ilişkilendirilebilmesidir. XML. Kodlama Kontrol Spesifikasyonu, belirli türlerin kodlanma biçimini değiştirebilir, (belirtilen türler veya tüm türler için) PER veya BER uzunluk stillerini seçerek, etiketler ve/veya dolgu bitleri vb. dahil veya hariç tutabilir.

Bu çalışma (1999) daha çok başlangıç aşamasındadır. Sonuç, bir ASN.1 modülüne uygun bir Kodlama Kontrol Spesifikasyonunun o modüldeki tiplerin BACnetER ile kodlanması etkisi ile uygulanabileceği kadar güçlü bir meta-dil (kullanmak için bir araç oluşturulabilir) olabilir mi? OER (veya belki de XER) kodlamaları?

Çalışmanın genel amacı budur. Ancak bundan beş yıl sonra, onu hiç duymamış olabilirsiniz ve LWER kadar ölü olabilir veya birçok araç tarafından desteklenebilir ve ASN.1'e önemli ek esneklik sağlayabilir. bilmiyorum! Bu kitabın ikinci baskısını (eğer varsa!) edinin! (Ama henüz ISO'da resmi olarak onaylanmış bir Çalışma Öğesi bile değil, bu yüzden bu şeyler şu anda sadece gözlerde parlıyor.)

BÖLÜM IV

Tarihçe ve Uygulamalar

Bölüm 1
ASN.1'in geliştirilmesi

(Veya: Yaşlı bir adamın saçmalıkları!)

Özet:

Bu bölümün tarzı kitabın geri kalanından biraz farklı. (Bu özet, başlangıç için bir mermi listesi değildir!) Bazı gerçekleri içermekle birlikte, ASN.1'in geliştirilmesindeki aşamaların ve tarihlerin resmi bir kaydı değildir (bunun için Olivier Dubuisson'un kitabı daha iyidir) – Ek 5'teki bağlantıya bakın) yol boyunca meydana gelen çeşitli olaylara dair kendi kişisel hatıralarım.

Akademik bir metin için alışılmadık bir şekilde, bu bölümde çeşitli bölümlerde pervasızca "ben" şahıs zamirini kullanıyorum. Uygun görünüyordu.

ASN.1'e neredeyse ilk günlerinden beri dahil oldum (Sanırım yalnızca Jim White - bu bölümün ilk maddesinde Jim'den bahsediyorum - başından beri onu sonuna kadar gördüğünü iddia edebilir, ancak o Standartlardan "emekli oldu". 1980'lerin sonlarında çalışmak) günümüze kadar. ISO içinde bir dizi Standardizasyon alanında aktif olarak bulundum, ancak ASN.1, yaklaşık 20 yıllık zaman aralığı nedeniyle (bu metni yazarken) muhtemelen zamanımın en büyük bölümünü aldı.

ASN.1'in geliştirilmesi için zamanlarının büyük bir kısmını harcayan başka birçok insan vardı ve bunlardan bazılarını listelerseniz, ASN.1'i yeni bırakanlara karşı büyük bir haksızlık yapma (ve rencide etme) tehlikesiyle karşı karşıya kalırsınız. listenin sonunda, ancak yine de çalışmaya önemli katkılarda bulunanlar. Kimden bahsedeceğime dair kolay bir kriter yok ve artık isimlerini doğru yazamadığım ve devam kayıtlarını kaybettiğim eski çalışma arkadaşlarımdan bazıları var!

Ve tabii ki, ASN.1 çalışmasına, sadece mevcut sürücüler oldukları için hayattan daha büyük görünen mevcut katılımcılar var . Ama çoğunu görmezden geliyorum! Umarım kimse dışarıda bırakıldığı için gücenmez.

Bu bölümün yapısı basit bir zaman çizelgesi değildir. Bunun yerine, ana alt başlıklar için belirli temalar seçilmiştir, ancak bu alt başlıklar içinde malzeme büyük ölçüde bir zaman çizelgesi temelinde sunulmuştur. Bunun metinde daha fazla süreklilik sağlayacağını ve saf bir zaman çizelgesi çalışmasına göre daha kolay okunacağını umuyorum, ancak okuyucuya ana alt başlıkların büyük ölçüde bağımsız olduğu ve okunabileceği (veya atlanabileceği veya atlanabileceği) tavsiye edilir.) ilgi alanlarınıza bağlı olarak az ya da çok rastgele bir sırada.

Bu bölümün önemli bir kısmı, Kısım II Bölüm 2'de vaat edilen karakter kodlamalarının gelişim tarihini içermektedir.

1 kişi

Jim White, sonunda ASN.1'in dayandığı Xerox Courier spesifikasyonunun geliştirilmesinde aktif bir rol oynadı (belki de lider bir rol - emin değilim).

Önce bunu yoldan çıkaralım!

Courier, "XNS" protokol yığınının bir parçasıydı. Bence, kullanıcılara (yalnızca bilgisayar satıcılarına değil) olanak sağlamak için üst düzey dil sistemleri içindeki iyi tanımlanmış kodlama kuralları ve araçları tarafından desteklenen protokol mesajlarının tanımı için bir notasyon sağlama değerinin protokol mimarisinde ilk tanınmasını temsil ediyordu.) kendi protokollerini tanımlamak ve bu protokoller için kolay bir uygulama yoluna sahip olmak.

Jim (CCITT'de X.400 çalışması için notasyonel destek geliştirmekten sorumlu Raportör olarak), Courier ilkelerini uluslararası standardizasyona getirmekten ve zamanı geldiğinde X.409'un üretiminden büyük ölçüde sorumluydu.

Doug Steedman ayrıca bu ilk günlerde hem CCITT hem de ISO içinde çok aktifti ve (sanırım) ASN.1 hakkında tam uzunlukta bir eğitim metni yazan ilk kişiydi. Bu, bugün hala okunmaktadır, ancak ne yazık ki, Doug da 1980'lerin sonlarında Standartlar çalışmasından "emekli olduğu" için, ne yazık ki 1990'dan sonraki çalışmaları kapsayacak şekilde güncellenmemiştir.

İlk ISO metinleri için ISO Editörüydüm (ve X.409'dan sonra, CCITT metinleri ISO metinlerinin kopyalarıydı). Bancroft Scott, 1980'lerin sonunda (diğer "emeklilikler" nedeniyle), ISO'da ASN.1 çalışması için Raportör olduğumda ve ASN.1'in bir bölümünde Editör olmak için gönüllü olan Bancroft'un Kendisi, tüm farklı bölümlerin Editörüdür (artık ISO'da altı bölüm ve buna karşılık gelen altı ITU-T Önerisi), bu metnin yayınlandığı tarihte (1999) işgal etmeye devam ettiği bir roldür.

Daha yakın yıllarda, Olivier Dubuisson ASN.1'in geliştirilmesinde çok aktif bir rol oynamıştır ve ASN.1 ile ilgili ikinci/üçüncü/dördüncü büyük kitabın yazarıdır. (Kitabının Fransızca bir versiyonuyla bu metnin daha önce yayınlandığını iddia edebilir - ikinci metni yapıyor, ancak bunu yazdığı sırada umarım İngilizce versiyonu bu yayından sonra gelir ve onu dördüncü yapar - ama o üçüncü de olabilir! Dostluk rekabeti!)

Listeleyebileceğim ve belki de listelemem gereken pek çok başka kişi var, özellikle de yıllar içinde ASN.1'e büyük destek sağlayan BSI'daki meslektaşlarım, ancak daha sonra AFNOR bünyesinde faaliyet gösteren ve İsveç'ten meslektaşlarımdan ve BSI'daki meslektaşlarımdan da bahsetmeliyim. Bugün hala tüm dünyada kullanılan ASN.1 için ders materyali üreten ABD ve ...

Durmak! Bu madde yeter!

2 Daireler çizerek mi dönüyorsunuz?

Anlayışın ancak yavaş yavaş ortaya çıktığı bilgisayar iletişimleri için notasyon ve kodlama desteğinin pek çok alanı vardır. (Daha önce açıklanan "delikler" desteği ve protokol belirtimlerinin "sürüm 1" ve "sürüm 2" uygulamaları arasında birlikte çalışmayı sağlayan mekanizmalar da bunlardan biridir). Bazen gelişmeler ileriye dönük net adımlardır (ASN.1'in 1980'lerin başında piyasaya sürüldüğü durumda olduğu gibi), bazen bazı alanlarda ilerleme kaydetmek için bazı alanlarda geri adımlar atıyoruz.

Karanlık bir camdan görüyoruz. Bilgisayar mesajları iletişimini tanımlamaya çalışan insanlar için nedir? ASN.1'in sunabileceği en iyi "doğru" notasyonel çözüm var ve sorunların çoğunu fark etti (ve bazı iyi çözümler sağladı), ancak dünyanın daha kat etmesi gereken bir yol var.

ASN.1 1980'lerin başında doğduğunda, Açık Sistem Ara Bağlantı (OSI) Standartları "dilimlenmiş ekmekten bu yana en iyi şeydi" ve bu Standartları ISO ve CCITT içinde geliştirmek için yapılan toplantılara genellikle birkaç yüz kişi katılırdı. Ancak, uygulamalar için OSI Standartlarını tanımlayan tüm ISO gruplarında, o zamanlar, mesajı desteklemek için kullanılacak mesajları (semantikleri ve bit modelleri dahil) açıkça belirtmek için hangi notasyonun kullanılacağına dair bir şüphe, bir tartışma vardı. başvuru. Her grup, farklı yaklaşımlar ve farklı notasyonlarla kendi işini yapıyordu.

Bir BNF (Bacchus-Naur Formu) tarzı belirtim kullanımı, çoğu erken OSI taslağında yaygındı, genellikle karakter dizilerine dayalı bir kodlamayla (bugün birçok İnternet protokolünde olduğu gibi).

İlk ASN.1 metni (ve o günlerde ASN.1 olarak adlandırılmıyordu - bu başka bir hikaye - aşağıya bakın) CCITT'den ISO'ya bir bağlantı olarak gönderildiğinde, hemen hemen her bir uygulama katmanı standardizasyon grubu tarafından memnuniyetle karşılandı. ISO'da şu şekilde:

> Protokolleri belirlerken herkesin kullanması için ortak ve standart bir notasyona sahip olmak harika.
>
> Ayrıntılı metin tabanlı değiş tokuşlardan uzaklaşmak için harika.

(Sonuncu noktaya dikkat edin. Daha sonra BER'in ayrıntılarına yönelik güçlü eleştirilere ve sonunda PER'nin ortaya çıkmasına rağmen, her ikisi de metin tabanlı kodlamalardan çok daha az ayrıntılıdır.)

ASN.1, tüm uygulama katmanı OSI Standartları (ve ayrıca Sunum Katmanı için) için tercih edilen notasyon (ve BER kodlaması) haline geldi.

Ancak ASN.1'in OSI yığınının dışında yaygın olarak kullanılmaya başlaması 1980'lerin ortalarındaydı.
Hatta İnternet topluluğu içinde bir miktar alım oldu (genellikle küçültülmüş - bazıları piç diyebilir! - şeklinde), ancak ASN.1'in gerçek genişlemesi telekomünikasyon standartları belirleyicileri arasındaydı.

Bugün pek çok telekomünikasyon standardının (cep telefonları, akıllı ağlar, sinyalizasyon sistemleri, elektrik güç dağıtımının kontrolü, hava trafik kontrolü) ASN.1 kullandığı bir durumdur. (Bir sonraki bölüme bakın.)

Ancak bugün hala metin tabanlı protokolleri tercih edenler ile ASN.1 destekçileri arasında bir savaş görüyoruz. ASN.1 için XER'in (Genişletilmiş Biçimlendirme Dili - XML - Kodlama Kuralları) ortaya çıkışı, bazı yönlerden iki kampı evlendirmiştir. XER, türleri tanımlamak için ASN.1 notasyonunu temel alır, ancak bu türlerin değerlerinin aktarımı için tamamen karakter tabanlıdır (ve ayrıntılı!).
Ancak, bugün (bazı gerekçelerle) insanların şöyle dediğini duyacaksınız:

HTML (Netscape ve Microsoft ile), bir kez yazılan, her yerde okunan Web sayfaları için hüküm sağladı.

JAVA, bir kez yaz, her yerde çalıştır programları için hazırlık yaptı.

XML, bir kez yazılan, her yerde işlenen veriler için hazırlık sağlar.

Ve elbette, bir iletişim-özellik-dil yarışmacısı olarak hala CORBA (IDL gösterimi ve bir kodlama olarak IOP protokolü ile) var!

Ve mesajlar için tercih edilen tanımlama mekanizması olarak BNF ve karakter tabanlı değiş tokuşları kullanmayı seçen birçok İnternet Mühendisliği Görev Gücü (IETF) belirtimimiz var.

Dünyanın bilgisayar iletişimlerini tanımlamanın ve kodlamanın "doğru" yolunu anlaması, anlaması veya tanıması (ve bu, bugün bildiğimiz biçimde ASN.1 olabilir veya olmayabilir) henüz biraz zaman alabilir. . 1980'lerin başından bu yana (çözülecek sorunları ve sorunları anlama açısından) çok ilerleme kaydettik, ancak siyasi anlaşmalarda (küçük "p" harfiyle) daha az ilerleme kaydettik ve hala (endişe verici derecede yüksek) bir sayı protokolleri tanımlamada kullanılacak notasyon yarışmacılarının sayısı. Ve hala insanlar daha fazlasını önermeye devam ediyor! (Programlama dili sahnesinden daha kötü değil sanırım.)

kullanaÖngörü? Özel...ğız? ... 2020'de ... protokol standartları ...mek için hangi gösterimi

3 Standartları kim üretir?

Yıllar içinde ve bugün bilgisayar iletişimi ile ilgili Standartların üretilmesinde ve bu Standartları desteklemek için çeşitli notasyon biçimlerinin benimsenmesinde beş ana aktör grubu olmuştur.

Dünya çapında bilgisayar iletişimi için yasal ve fiili standartlar konusunda her zaman bir zorluk olmuştur. Ulusal Standartlar Enstitüleri genellikle gücü ellerinde tuttuklarını düşünür/umarlar. Ancak dünyadaki bilgisayarların nasıl iletişim kurduğuna karar verme konusundaki gerçek güç büyük ölçüde onların elinde değil, zamanla birçok aktör arasında değişti.

Beş kim?

Öneririm:

1970'lerde ana çerçeve bilgisayar satıcıları, ancak artık büyük ölçüde önemsiz.

1980'lerde ve 1990'larda CCITT (1990'ların başında ITU-T olarak yeniden adlandırıldı) ve bugün hala telekomünikasyon standartlarının belirlenmesinde baskın güç.

ISO, büyük ölçüde CCITT/ITU-T ile işbirliği içinde çalışıyor, ancak büyük etkisi 1980'lerin OSI gelişmeleriyle sınırlı ve belki de bugün yalıtılmış alanlar dışında baskın bir güç değil.

IETF, görev güçleri ve çalışma grupları artık (birçok uygulama için) telekomünikasyon kullanıcıları arasındaki bilgisayar iletişimi için fiili standartlar haline gelen İnternet standartlarının geliştirilmesinden sorumludur (buna karşın ITU-T, iletişim telekomünikasyon mümkündür).

SET konsorsiyumu, World-Wide Web Konsorsiyumu (W3C) ve CORBA grubu da dahil olmak üzere çeşitli üretici ve diğer grup konsorsiyumları, günümüzde giderek artan etkiye sahiptir.

Bilgisayar satıcılarının protokol tanımındaki önemi, (daha önce belirtildiği gibi) orijinal ASN.1 kavramlarını doğuran dikkate değer XEROX istisnası dışında, ASN.1 sahneye çıkmadan önce büyük ölçüde azalmıştı.

Uluslararası bir spesifikasyon olan ASN.1, CCITT içinde X.409 olarak "Sunum Aktarımı Sözdizimi ve Gösterimi" adıyla hayata geçti. ("Aktarım sözdizimi"nin - İngilizce - başlıkta "gösterimde" değil, ilk sırada yer aldığına dikkat edin! Bugün, gösterimi muhtemelen ASN.1'in daha önemli bir parçası olarak görürüz). ASN.1'e yol açan çalışma, başlangıçta yalnızca X.400 serisi e-posta protokollerinin tanımı için notasyonel destek sağlamayı amaçlıyordu. Bununla birlikte, çok hızlı bir şekilde ISO'ya geçti ve 1980'lerin başlarında, çalışma işbirliğine dayalı olmasına rağmen, girdilerin çoğunun sağlandığı büyük ölçüde ISO Ulusal Kuruluşlarıydı (bunlara daha sonra "Üye Kuruluşlar" deniyordu).

1990'ların sonlarında sarkaç geri sallandı (kısmen OSI'nin düşüşünden ve kısmen de ISO içindeki yeniden organizasyonlardan dolayı), o zamana kadar ITU-T haline gelen şey, ASN.1 üzerinde yeni çalışma ilerletmede koşunun çoğunu yaptı.

IETF içinde, ASN.1'in alınması her zaman çok düzensizdi. Bu muhtemelen en azından kısmen, IETF'deki taşıyıcıların çoğunun halka açık (ücretsiz) araçlardan destek alan bir belirtim dili istemesinden kaynaklanıyordu. BNF tabanlı metin kodlamaları bu gereksinimi karşıladı. ASN.1 yapmadı ve bugüne kadar da yapmıyor (1999). Dolayısıyla, ASN.1'in IETF dünyasındaki çoğu kullanımı, ASN.1'in herhangi bir araç kullanılmadan kolayca kodlanabilen/kodlanabilen bir küçültülmüş sürümünü kullanıyordu (ve kullanıyor).

Buna karşılık, ITU-T telekomünikasyon spesifikasyonları, ASN.1'in tüm gücünü kullanır ve bu spesifikasyonları uygulayan telekomünikasyon ve anahtar satıcıları, mevcut araç ürünlerinden tam olarak yararlanarak protokollerin kolay, hızlı ve (büyük ölçüde) hatasız uygulanmasını sağlar. bant genişliği gereksinimleri açısından oldukça verimli.

4 sayı oyunu

ASN.1 belirtimleri çeşitli tanımlamalardan geçmiştir.

Son yirmi yılda ASN.1 spesifikasyonlarının tanımlarının sıkıcı bir listesi - atlayın!

İlk yayınlanan spesifikasyon X.409 (1984) idi. X.409, "Soyut Sözdizimi Gösterimi Bir (ASN.1)" teriminin kullanımından öncesine sahipti ve X.400 serisinin bir parçasıydı. Oldukça basit bir şekilde, X.400 (OSI e-posta) takımındaki protokollerin belirlenmesine yardımcı olacak bir notasyon (ve kodlama kuralları) olarak görülüyordu.

Daha sonra tamamen yeniden yazıldı (teknik değişiklik yapılmadan - sonraya bakın!) ve ISO tarafından 1986'da ISO 8824 ve ISO 8825 olarak yayınlandı (bazı eklemelerle) ve aynı metin (yine bazı eklemelerle) daha sonra CCITT tarafından yayınlandı. 1988'de X.208 ve X.209 olarak. Bu metnin daha sonraki bir versiyonu (küçük düzeltmelerle birlikte) ISO ve IEC tarafından 1990'da ISO/IEC 8824 ve ISO/IEC 8825 olarak ortaklaşa yayınlandı. ASN.1'in 1990 sürümü".

"ASN.1'in 1994 versiyonu" (1990 versiyonuna çok büyük uzantılarla birlikte), ISO/IEC ve CCITT tarafından bir dizi yeni belge olarak ortaklaşa yayınlandı ve aynı metin aşağıdaki paralel sütunlarda gösterildi:

İTÜ-T X.680 İTÜ-T	ISO/IEC 8824-1
X.681 İTÜ-T X.682	ISO/IEC 8824-2
İTÜ-T X.683 İTÜ-T	ISO/IEC 8824-3
X.690 İTÜ-T X.691	ISO/IEC 8824-4
	ISO/IEC 8825-1
	ISO/IEC 8825-2

Yine daha sonra, ortak bir ISO/IEC ve ITU-T "1997 versiyonu" vardı (1994 versiyonunda sadece nispeten küçük değişiklikler ve eklemeler ile). Ancak, "nihai" metin 1997'de onaylanırken, ne ITU-T ne de ISO henüz insanların satın alabileceği yayınlanmış bir kopya üretmedi (bugünkü tarih 1999'un başları)! Ama bu alanı izleyin, çok yakın! (Daha sonra düzeltin - artık ITU-T'den satın alabilirsiniz!)

Okuyucular, 1994'te (ve 1997'de) X.680'in, özellikle karakter seti alanında bazı uzantılarla kabaca eski X.208 olduğunu not etmelidir. X.681, Bilgi Nesnesi konseptiyle ilgili uzantılardı. X.682, tablo ve ilişkisel ve kullanıcı tanımlı kısıtlamalardı ve X.683, parametrelendirmeydi. X.690, CER ve DER eklenmiş eski X.209'du ve X.691, PER spesifikasyonuydu.

Vay! Sayılardan nefret ediyorum ! Nuff dedi.

5 İlk yıllar - X.409 ve tüm bunlar

5.1 Taslaklar değiştirilir ve ASN.1 adı atanır

X.409'un ilk taslakları CCITT'de üretildi. O günlerde hem ISO hem de CCITT'nin OSI için bir "7 katmanlı modeli" vardı ve tamamen farklı metinlerdi (teknik olarak çok benzer, ancak büyük ölçüde bağımsız olarak geliştirildi). İki grup arasında güçlü bir işbirliği dönemi henüz gelmemişti ve iletişimin çoğu, genellikle bazı spesifikasyonların bir taslağının eşlik ettiği yazılı "irtibat beyanları" ile sağlanıyordu.

İsim nedir? Bir "nokta" içinde ne var? Aslında çok fazla.
Bir nokta olmadan kafa karışıklığı hüküm sürüyor, noktayı ekleyin ve her şey yolunda!

Bu şekilde (1982'de) X.409 ISO TC97 SC16'ya (Teknik Komite 97 - bilgisayarla ilgili tüm standartlardan sorumlu, Alt Komite 16 - OSI modelinden ve Ağ üzerindeki OSI standartları üzerindeki tüm çalışmalardan sorumlu) ulaştı. Katman). İlk başta, bu X.409 kavramlarının OSI modeline nasıl uyduğu net değildi ve taslağı değerlendirmek için geçici bir grup (sanırım Lloyd Hollis'in başkanlığında) kuruldu. Bu çalışmanın OSI'nin Sunum Katmanına yerleştirilmesi gerektiği hızla anlaşıldı ve çalışmayı karşılayan bir irtibat bildirisi gönderildi.

Bu X.409 taslağı bir ISO boşluğuna geldi - ya da belki de ilkel bir plazmayı kastediyorum! Tüm farklı uygulama katmanı standartlarının, protokollerini tanımlamak için hangi gösterim mekanizmalarını kullanacaklarını merak ettiği ve hepsinin farklı yaklaşımlara sahip olduğu bir anarşi vardı. Yeni gösterim, protokollerini tanımlama aracı olarak her bir Uygulama Katmanı standart grubu tarafından son derece hızlı bir şekilde kabul edildi.

Bu sırada gösterim için bir isim düşünüldü ve ISO grubu Soyut Sözdizimi Gösterimi Bir veya "ASN1" önerdi. CCITT grubu, "Tamam, ama asla bizimle konuşma" yanıtını verdi.

ASN.1'in (1994) ASN.2 olarak adlandırılması gerektiğini savunanlar olmasına rağmen, ASN2 hiçbir zaman önerilmedi (sonraki metne bakın).

Son paragrafta "ASN"den sonra nokta olmadığına dikkat edin. Bu bir yazım hatası değildi! Önerilen orijinal ad gerçekten de "ASN1" idi. Bununla birlikte, altı ay içinde, insanların onu sık sık "ANS1" olarak yanlış yazdıkları ve/veya onu "ANSI" - Amerikan Ulusal Standartlar Enstitüsü - olarak yanlış okudukları ortaya çıktı. Önemli bir kafa karışıklığı yaratılıyordu! ABD delegasyonu başkanının (aynı zamanda SC16 Başkanı!) ASN.1 grubuna gelip "Bakın, "ANSI" olmadığını biliyorum ama o kadar yakın ki sorun çıkarıyor dediği günü hatırlıyorum. , adını değiştiremez misin?".
kargaşa! Patlama! Ama ortalık yatışınca "nokta" girilmişti ve elimizde "ASN.1" vardı.
Bundan sonra hiç kimse onu yanlış yazmadı veya ANSI ile karıştırmadı!

"Nokta" emsalsiz değildir - tüm CCITT Tavsiyeleri bir nokta ile yazılmıştır - X.400, X.25, V.24, bu nedenle ASN.1 kolayca kabul edilmiştir.

Bu sırada "BER" (Temel Kodlama Kuralları) terimi türetildi, ancak bu durumda hem ISO hem de CCITT'de başka ve belki de daha iyi kodlama kurallarının üretilebileceği kabul edildi, ancak PER'den önce on yıl geçti (Paketlenmiş Kodlama Kuralları) sonunda ortaya çıktı.

5.2 BER'i notasyondan ayırma

Bu ilk yıllarda bazı zor anlar yaşandı. Soyut belirtimi (Uygulama Katmanı) kodlama sorunlarından ayırmanın önemi konusunda çok güçlü bir görüşe sahip olan CCITT değil, ISO idi (yayınlanan ilk X.400 belirtimi, Sunum Katmanı olmaksızın doğrudan Oturum Katmanı üzerinde yekpare bir protokoldü).

X.409 taslağı (ve sonunda yayınlanan X.409 (1984)) paragraf paragraf serpiştirilmiş, ASN.1 gösteriminin bir parçasının açıklamasını ve karşılık gelen BER kodlamasının belirtimini içeriyordu.

ISO, Sunum Katmanı konusunda ciddiydi. Kodlama ayrıntıları, uygulama semantiğinden açıkça ayrı (ayrı belgelerde) tutulmalıdır. Harika bir fikir, ancak CCITT bu konuda pek evanjelik değildi. Ancak ASN.1 olmadan konsept muhtemelen asla gerçeğe ulaşamazdı.

ISO'nun yapmaya karar verdiği ilk şey, bu parçaları parçalara ayırmak ve (teoride teknik değişiklik olmaksızın) biri notasyonu açıklayan (bu sonunda ISO 8824 oldu) ve diğeri BER'i açıklayan iki ayrı belge olarak tamamen yeniden yazmaktı (bu sonunda ISO 8825 oldu).

İlerleyen yıllarda (ve özellikle ASN.1 çalışmasında) ISO ve CCITT arasında daha yakın ve daha yakın işbirliği meydana geldikçe, elbette şu soru ortaya çıktı: CCITT, ASN.1 için ISO metnini kabul edip X.409'u bırakacak mıydı? Biraz ıstırap çektikten sonra oldu ve 1988'de X.409 geri çekildi ve X.200 serisinde X.208 ve X.209 olmak üzere iki yeni CCITT tavsiyesi vardı.
Tavsiye X.200'ün kendisi, OSI Referans Modeli'nin CCITT/ITU-T yayınıydı (ve öyledir) - sonunda ISO'nunkiyle uyumlu hale getirildi, ancak teknik olarak OSI'den çok orijinal CCITT taslağına doğru eğildi - ancak bu ayrı bir Öykü! (Web'de bulunan "OSI'yi Anlamak" kitabıma bakın.) ASN.1 spesifikasyonlarını X.200 serisine koymak, ASN.1'in X.400'ü geride bırakarak OSI'nin tamamı için genel bir araç haline geldiğinin kabul edilmesiydi. . 1994'te X.680 ve X.690 serisine geçişinin OSI'yi aşmasını temsil ettiğini düşünmeyi seviyorum, ancak bunun daha çok şu anda altı Tavsiyeye ihtiyaç duyması ve uygun yer kalmamasından kaynaklandığını düşünüyorum. X.200 aralığında! (ISO'da benzer sorunlar yoktur - ISO 8824 gibi tek parçalı bir Standart, numarasını değiştirmeden ISO 8824 Bölüm 1 (ISO 8824-1), Bölüm 2, vb.'ye dönüşebilir.)

X.409 oldukça resmi olmayan bir tarzda yazılmıştır, ancak ISO topluluğu içinde yeniden yazıldığında, ISO Standartları için gerekli olan oldukça yapmacık "standart" dil kullanılmıştır. Örneğin, "must" asla kullanılmamalıdır - bunun yerine "shall" kullanın (bunun nedeni, Fransızca'ya çeviri iddia edilen zorluklardır), örnekler veya nedenler vermeyin, yalnızca gereksinimlerin ne olduğunu açıkça ve tam olarak belirtin - bir makale yazıyorsunuz Açıklayıcı bir metin parçası değil, insanların Standarda uymak için ne yapması gerektiğinin belirtilmesi .

ASN.1'e nazik bir giriş yapmak isteyenlere genellikle X.409'un (1984) eski bir kopyasını bulmalarını ve onu okumalarını tavsiye ederim - daha gayri resmi bir dilde yazılmıştır ve kodlamalar notasyonun yanında belirtilmiştir , Yeni başlayanların kavramasının daha kolay olduğuna inanıyorum. Ama Olivier'in kitabında 8824/8825'in X.409'dan daha okunabilir ve daha iyi özellikler olduğunu iddia ettiğini görmek ilgimi çekti! Sanırım hepimizin neyin iyi bir özellik olduğuna dair kendi görüşlerimiz var!

5.3 Değişiklikler ne zaman teknik değişikliklerdir?

> Bir imla düzeltin, bir örneği kaldırın, önemsiz şeyler. Sorun değil. İnanma!

Gerçekten de ISO, X.409'u teknik değişiklikler yapmadan yeniden yazmaya çalıştı, ancak ikisi araya girdi. İlki, "GeneralizedTime" türüyle ilgiliydi. Bunlar, insanların yazı yazmak için kelime işlemciler yerine insan sekreterlere sahip olduğu günlerdeydi. X.409 ABD'de yazılmıştır. 8824/8825 için ISO metninin Birleşik Krallık Editörü (mea culpa) vardı ve Editörün bilmediği sekreter (başka bir isim - Barbara Cheadle!), yazımı "GeneralisedTime" olarak düzeltti. Bu, tüm resmi oylamalarda fark edilmedi, ancak 8824 gerçekten yayınlanmadan önce sonunda düzeltildi!

Neyin "doğru" İngilizce olduğu konusundaki tartışmalara bakılmaksızın, "GeneralizedTime" terimi geçerli olmalıydı, çünkü bu notasyonun resmi bir parçasıydı ve yazımındaki herhangi bir değişiklik teknik bir değişikliği temsil ediyordu!

İkinci değişiklik ancak 1990'ların başında fark edildi! Bu konuda bir şey yapmak için çok geç!
Bir örnekte yalnızca X.409'da belirtilen TeletexString karakter dizisi türüyle ilgili bir ayrıntı noktası vardı. Örnek 8824'te kayboldu ve ayrıntı noktası onunla birlikte kayboldu - Korkarım ayrıntı noktasının kesin ayrıntılarını unuttum!

5.4 ASN.1'in neredeyse sona ermesi - İŞLEM ve HATA

> Kolay savaşlar, yanlış anlamaya veya anlayış eksikliğine dayanır (zor olanlar, gerçek kişisel çıkar çatışmalarına dayanır). Bu kolay bir savaştı, ancak barışı sağlamak için gereken kısa süreler çatışmayı büyüttü.

İlk günlerle ilgili bu maddede anlatmak istediğim son olay, ASN.1'i neredeyse tamamen raydan çıkaran olaydır.

O zamanlar CCITT, Çalışma Dönemi olarak adlandırılan dört yıllık bir zaman çerçevesine kilitlenmişti ve dört yılın başında "Sorular" (büyük Q!) formüle edilmişti. (Her Soru genellikle yeni bir Öneriye veya mevcut bir Önerinin güncellenmesine yol açtı.) Çalışma Döneminin sonunda, tamamen yeni bir CCITT Önerileri seti yayınlandı (her dönemde farklı bir renkle kaplandı). 1980'de renk Sarı, 1984'te Kırmızı ve 1988'de Mavi idi.

(1988, bu tam yeniden basımın gerçekleştiği son yıldı, bu nedenle, yeni durumda bir Mavi kitap setiniz varsa, onları saklayın - bundan elli yıl sonra değerli olacaklar!)

İdarenin bu yeni metinleri yayına hazırlaması zaman aldı ve o günlerde CCITT, Çalışma Döneminin bitiminden yaklaşık on iki ay önce, yeni veya tadil edilmiş Tavsiyelerin sonuçlandırılmasıyla ve yalnızca "kauçuk" ile "büyük bir uykuya" girdi. Ertesi yıl "damgalama" toplantıları. CCITT ASN.1 grubu en son X.409 taslağını ISO grubuna gönderdiğinde, 1993 yılının ortalarında, "büyük uyku" başlamak üzereydi - gece yarısına beş dakika kalmıştık -.

Çoğunlukla sadece ufak tefek şeylerdi, ancak ASN.1 sözdizimine aşağıdaki gibi yapılar yazma becerisini "donanımlı" hale getiren yepyeni bir bölüm eklenmişti:

```
arama İŞLEMİ
        ARGUMENTS isim Bazı tür
        SONUÇ adı Sonuç türü
        HATALAR {geçersizİsim, isimBulunamadı} ::= 1
```

ve

```
adBulunamadı HATA ::= 1

geçersizAdı HATA
        PARAMETRE nedeni BITSTRING
                {nameTooLong(1),
                  illegalCharacter(2), belirtilmemiş(3) }

        ::= 2
```

Pekala ... okuyucu bu kitabın önceki bölümlerini ve özellikle Kısım II Bölüm 6 ve 7'yi okuduysa, bu sözdizimi oldukça tanıdık gelecek ve anlamı belki oldukça açık olacaktır. Ancak revize edilmiş ASN.1'i tanımlayan basit bir irtibat beyanıyla karşılaşan (ve ROSE çalışmasının varlığı hakkında bile kesinlikle hiçbir anlayışa veya bilgiye sahip olmayan) ISO grubundakiler için tam bir anlaşılmazlık vardı.

Bunun soyut bir sözdizimi (ve karşılık gelen kodlama kuralları) için veri türlerini tanımlamakla ne ilgisi vardı? HATA ve İŞLEM nasıl kodlandı (taslakta herhangi bir kodlama belirtilmedi)? Bir "operasyon" veya "hata" neydi? Hepsini sökün! Daha fazla zaman olsaydı .... Ancak ISO grubu, bu şeylerin planlanan ISO Standartlarına girmesinin mümkün olmadığına karar verdi. CCITT içindeki ıstıraplar. Saklayın ve ASN.1 için farklı Öneriler ve Standartlar riske mi atın?

Bir sonraki X.409 taslağı ISO'ya ulaştığında gece yarısına bir dakika kalmıştı. Rahatsız edici OPERATION ve ERROR sözdizimi kaldırıldı - derin bir rahatlama - ancak bir "makro notasyonu" tanımlayan yeni bir Ek eklendi. Bu Ek çok, çok belirsizdi! Ancak birçok programlama dilinde, dili desteklemek için bir "makro notasyon" vardı. (Bunlar genellikle gerçek parametrelerle çeşitli yerlerde somutlaştırılabilen sahte parametreler içeren bazı şablon metinleri biçimini alırdı - sonunda ASN.1'in parametrelendirme özellikleriyle tanıtılan şey). Ve gece yarısına bir dakika vardı . Ve CCITT grubu , OPERATION ve ERROR sözdizimini geri çekmeyi kabul etti ve karşılığında bir iyiliği hak etti. ISO grubu, Ek makro gösterimini kabul etmeyi kabul etti. Barış sağlandı ve ASN.1 kurtarıldı!

Geçmişe bakıldığında, 1990'ların sonlarına kadar yansımaları olsa da, tüm bu olay muhtemelen iyi bir şeydi. İŞLEM ve HATA birbirine bağlı kalsaydı ve makro notasyon olmasaydı, ASN.1'in Bilgi Nesneleri ile ilgili kavramları geliştirmesi çok daha zor olurdu (ve zaten oldukça zordu!). Aşağıda bu konu hakkında daha fazla bilgi.

6 Organizasyon ve yeniden organizasyon!

Açık Sistemler Ara Bağlantısı fikri ilk olarak ISO'da düşünüldüğünde, TC97 SC6'da HDLC (Yüksek Seviye Veri Bağlantı Kontrolü) üzerine yapılan çalışmadan, "HDLC'yi neyin doldurduğunun formatlarını kim - ve nasıl - tanımlayacak" sorusundan geldi. çerçeveler?"

> Organizasyon yapıları biraz önemlidir, ancak yukarıdaki yeniden organizasyona rağmen teknik çalışmalar genellikle devam edebilir. Ancak bazen çok fazla çalkantı, çalışmanın resmi olarak ilerlemesini (ve dolayısıyla yayın durumuna ulaşmasını) zorlaştırabilir. Neyse ki, ITU-T/CCITT ve ISO/IEC ortak projesiyle bir forumda ilerleyemiyorsanız, muhtemelen diğerinde ilerleyebilirsiniz!

TC97'nin Sidney'deki bir toplantısında, OSI için bir model geliştirme göreviyle görevlendirilecek yeni bir alt komite olan SC16'nın oluşturulmasına karar verildi ve ilk toplantısında, ana komitelerin her birinden yaklaşık altı farklı önerilen model sunuldu. ancak OSI'nin nihai şekline en çok benzeyen sunum, Avrupa Bilgisayar Üreticileri Birliği'nden (ECMA) gelen sunumdu. ABD, yeni bir alt komitenin kurulmasına karşı oy kullandı, ancak oldukça ilginç bazı siyasi manevralarla (yine bu metnin kapsamı dışında!) Sekreterya oldu ve SC16'nın Başkanlığını sağladı.

SC16, tüm ISO'daki en büyük alt komitelerden biri haline geldi ve altın çağında ancak tam bir büyük Üniversite kampüsünü devralarak toplanabildi. ASN.1, SC16'nın Sunum Katmanı Raportör Grubu içinde nispeten bağımsız bir grup haline geldi.

CCITT cephesinde ASN.1, Çalışma Grubu VII'nin bir parçası haline geldi ve nispeten sakin (organizasyonel olarak) bir yaşam sürdü. CCITT, adını ITU-T olarak değiştirdiğinde, en alt seviyelerde organizasyonel etkisi çok az oldu, asıl değişiklik, SG VII'nin SG 7 olmasıydı! Bu, bugüne kadar ASN.1'in evidir (SG 7'nin Çalışma Grubu 5 içinde).

ISO cephesinde, ISO bilgisayar konularının standardizasyonunun Uluslararası Elektro-Teknik Komisyonu (IEC) ile ortak bir sorumluluk olduğu konusunda anlaştığında ve IEC ile birlikte yeni bir "Ortak Teknik Komite" oluşturduğunda, üst düzey bir yeniden yapılanma gerçekleşti. TC97'nin yerine 1". (Bir JTC2 hiçbir zaman olmadı ve muhtemelen olmayacak). Bunun ASN.1 çalışması üzerinde sıfır etkisi oldu, ancak Standartların kapak sayfasında artık ISO logosunun yanında IEC logosu yer alıyordu ve resmi numara ISO 8824 yerine ISO/IEC 8824 oldu. JTC1 tam olarak aynı SC'yi devraldı. yapı ve orijinal olarak TC97'de olduğu gibi aynı görevliler ve üyeler. O sırada ISO çalışmasına katkıda bulunanların adı "Üye Kuruluş"tan "Ulusal Kuruluş"a değişti, ancak bunlar hâlâ aynı kuruluşlardı - BSI, ANSI, AFNOR, DIN, JISC, bunlardan sadece birkaçı.

SC5 (programlama dilleri ve veritabanları) ve SC16'nın (OSI) yeni bir SC21 ve SC22 olarak yeniden şekillendirilmesi biraz daha yıkıcı bir yeniden düzenleme oldu, ancak geçiş sorunsuzdu ve ASN.1 çalışması gerçekten etkilenmedi.

Ancak 1990'ların sonunda, SC21 Sekreterliği alt komiteye artık kaynak sağlayamayacağına karar verdi ve komite SC32 ve SC33 olarak ikiye ayrıldı. ASN.1 tam teşekküllü bir Çalışma Grubu olarak SC33'e yerleştirildi (önceki tüm tarihi boyunca bir Çalışma Grubu içinde bir Raportör Grubunun alt statüsüne sahipti), ancak Ulusal Bir Organ olmadığı için bu grup altında hiç toplanmadı Bunun için Sekreterliği sağlamaya hazırlandı ve SC33 neredeyse hiç var olmadan dağıtıldı. ASN.1 (devam eden X.400 standardizasyonu da dahil olmak üzere orijinal OSI çalışmasının diğer kalıntılarıyla birlikte), SC6'ya (alt katman protokol standartlarından sorumlu ve çok uzun bir geçmişe sahip çok eski bir alt komite) atandı. CCITT/ITU T SG VII/SG ile yakın çalışma ilişkisi 7). Bu muhtemelen ISO içinde ASN.1 için iyi bir yuva olacaktır.

Bu son geçiş, daha önceki yeniden yapılanmalardan daha az pürüzsüzdü ve ASN.1 çalışmasının ISO içindeki resmi ilerlemesi kesintiye uğradı, ancak teknik düzeyde çalışma yine de devam etti ve belgelerin resmi ilerlemesi ITU içinde üstlenildi. -T yapıları.

7 Araç satıcıları

Elbette ASN.1, 1980 ila 1984 CCITT Çalışma Döneminde "icat edildiğinde", gösterimi destekleyecek hiçbir araç yoktu. Konseptlerinin çoğu için Xerox Courier'den yararlanırken, Xerox araçlarından hiçbiri ASN.1 için uzaktan kullanışlı olmayacak kadar farklıydı.

> Araç satıcıları. ASIMOV'un "Vakfı" Tüccarları. Kendi başlarına bir yasa, ancak girişimin başarısı için hayati ve orta yıllardaki gelişimine son derece katkıda bulunuyor.

Araçlar ortaya çıkmaya başlamadan önce 1980'lerin ortalarıydı ve bunlar genellikle sadece sözdizimi denetleyicileri ve güzel basılmış programlardı. 1980'lerin sonlarında, artık bildiğimiz araçlar ortaya çıkmaya başladı ve ASN.1 araç satıcısı endüstrisi doğdu. (ASN.1 araçları hakkında daha fazla bilgi için Bölüm I'deki Bölüm 6'ya bakın).

Tabii ki, ilk zamanlarda, ASN.1 üzerinde çalışan herkes esasen "kullanıcı" idi - bilgisayar üreticilerinin veya telekomünikasyon şirketlerinin (bazen Üniversitelerin) çalışanları ve genellikle ASN.1'i kendi protokolü olarak kullanan bazı protokollere büyük ilgi duyanlar. protokol tanımı için gösterim.
Ancak ASN.1 grubunun son toplantısında (1999), masanın etrafındakilerin çoğunluğunun bazı ASN.1 araçlarının satıcısıyla şu ya da bu şekilde güçlü bağları vardı - ASN.1 reşit olmuştu!

1980'lerin sonlarında, araç satıcılarının Standartlar toplantılarına katılmaya başladıkları ve ASN.1 sözdiziminin bilgisayarların okumasını zorlaştıran bazı özellikleri olduğundan şikayet ettikleri ilginç bir geçiş noktası vardı (asıl sorun, atama ifadeleri arasında bir ayırıcı olarak noktalı virgül - sonunda CHOICE ve ANY değerleri için değer notasyonuna iki nokta üst üste koyarak çözümlendi). O zamanlar, ASN.1'in bilgisayar tarafından işlenebilir bir dil olmadığına ve asla böyle olması amaçlanmadığına dair güçlü argümanlar vardı. Daha ziyade, bir grup insan (protokol standartlarını yazanlar) ile başka bir grup insan (bu protokollerin uygulamalarını üretenler) arasındaki iletişim için bir ortamdı. Bu görüş hızla yıkıldı ve bugün ASN.1 tam anlamıyla bir bilgisayar dili olarak görülüyor ve 1990'ların başında yapılan değişikliklerin çoğu, onu tamamen bilgisayar dostu hale getirme ihtiyacından kaynaklandı.

8 nesne tanımlayıcısı

8.1 Uzun ya da kısa, insan ya da bilgisayar dostu, işte bütün mesele bu

Nesne tanımlayıcıları (aşağıda resmi olmayan OID kısaltmasını kullanacağım), bugün bu kavramla yakından ilişkili olmalarına rağmen, "Bilgi Nesnesi" kavramından en az beş yıl öncesine aitti.

> Yine, bir ismin içinde ne var? Protokolünüzde taşıyorsanız uzunluk önemli olabilir!

1980'lerin ortalarında, OSI içindeki birçok farklı grubun, protokollerinin uğraştığı şeyleri tanımlamak için net isimlere ihtiyaç duyduğu ve dünya çapında birçok grup tarafından dağıtılmış bir şekilde atanabileceği anlaşıldı.

Benzer bir sorun birkaç yıl önce SC6'da ele alınmıştı, ancak "Ağ Hizmeti Erişim Noktası Adresleri" denen - İnternet'teki IP adreslerinin OSI eşdeğeri olan NSAP adresleri - için bir ad alanı sağlamaya yönelik daha dar bir odakla. Okuyucu NSAP adresleme şemasını incelerse, Nesne Tanımlayıcı sistemiyle bazı benzerlikler görülecektir, ancak çok önemli bir farkla, NSAP adreslerinin uzunluğu her zaman nispeten kısa tutulurken, uygulama katmanı protokolleri için uzun(ish) nesnedir. tanımlayıcılar uygun kabul edildi.

1986 civarında NESNE TANIMLAYICI türü üzerine çok fazla kan döküldü ve kolayca tamamen zıt bir yöne gidebilirdi (ama bence sonunda doğru karar verildi). Bu bir CCITT - ISO kavgası değildi - bu zamana kadar iki grup ortaklaşa buluşuyordu ve aralarındaki bölünmeler nadiren görülüyordu. (Bu durum, herhangi bir toplantıda, çeşitli katılımcıların genellikle her iki kampı da temsil ettiğini iddia edebildikleri, ancak bir kamptan veya diğerinden delege olduklarında, tartışmanın neredeyse hiçbir zaman iki kamp etrafında kutuplaşmadığı bugüne kadar devam ediyor.)

OID'lere dönmek için! Tartışma, bir OID'nin mümkün olduğu kadar kısa olması, yalnızca sayıların kullanılması mı yoksa çok daha insan dostu ve karakter tabanlı olması mı gerektiği ve içinde bileşenler olarak oldukça uzun adlar kullanmaya teşvik edilip edilmeyeceği üzerineydi.

Nihai uzlaşma, bugün sahip olduğumuz şeydi - her yayda benzersiz sayıları olan, ancak her yayda adları sağlamak için oldukça gevşek bir hüküm bulunan bir nesne tanımlayıcı ağacı. Nesne tanımlayıcıları için değer gösteriminde, sayılar her zaman görünür (adların esasen sayıların iyi bilinen eşanlamlıları olduğu üst düzey yaylar dışında), ancak insanlara yardımcı olmak için adlar da eklenebilir. Ancak kodlamalarda yalnızca sayılar aktarılır.

Uzlaşmanın bir diğer kısmı, uzun insan dostu metinleri taşımak için "ObjectDescriptor" türünün tanıtılmasıydı, ancak metin dünya çapında net olması garanti edilmedi ve bu nedenle bilgisayarlar için pek kullanışlı değildi. Daha önce belirtildiği gibi, "ObjectDescriptor" türü, ASN.1 cephaneliğinin tamamındaki en büyük nemli mermiydi!

Bir yıl kadar sonra X.500 grubu içinde çok benzer bir savaş - ama oldukça iyi bir şekilde tersi bir sonuçla - şiddetlendi. X.500 adları ("Ayırt Edici Adlar" olarak adlandırılır), temelde (biraz daha basitleştirilerek) şu şekilde olan bir ASN.1 veri türüdür:

```
DİZİ DİZİSİ {attribute-
    id feature-value
        TYPE-IDENTIFIER.&Type}          TİP-TANIMLAYICI.&id,
```

"TYPE-TANIMLAYICI.&id"nin aslında "OBJECT TANIMLAYICI" ile eşanlamlı olduğunu unutmayın, dolayısıyla X.500 adlarının ASN.1 adlarından çok daha uzun olduğu açıktır.

1980'lerin sonlarında (X.500 dışındaki gruplardan) X.500 için, Ayırt Edici Adlarının yanında basit bir tek NESNE TANIMLAYICISI ("kısa biçim" adı verilen ad) kullanımını desteklemesi için baskı vardı (yani- "uzun biçimli" isimler olarak adlandırılır) ve bunun olması gerektiğine SC21 içinde resmi olarak karar verildiğine inanıyorum, ancak bence asla olmadı!

8.2 Nesne tanımlayıcı ağacı nerede tanımlanmalıdır?

NESNE tanımıyla ilgili başka bir sorun
TANIMLAYICI türü, yalnızca bir veri türü tanımlaması değil, zımnen bütün bir kayıt yetkisini oluşturmasıdır.
yapı.

Sınır anlaşmazlıkları. Ah!

Bu, ASN.1 grubunun görev alanının ötesine geçti (OSI'de ayrı bir grup, kayıt otoritesi sorunlarını çözmekle görevlendirildi ve kendi standardını üretti). Bu, neredeyse on yıldır devam eden bir çekişmenin kaynağıydı. Başlangıçta (1980 ortası), insanlar "Nesne tanımlayıcı ağacın açıklaması ASN.1'den Kayıt Yetkilisi Standardına taşınmalıdır" diyorlardı, ancak CCITT çalışanları "Olmaz - ASN. 1 kullanıcı bu metni ASN.1 Standardının bir parçası olarak okuyabilmek istiyor ve bunun kontrolü ASN.1 grubunda kalmalıdır."

1990 yayınına kadar (ve dahil) ASN.1 Standardında kaldı. Ancak 1990'ların başlarında, roller tersine döndü ve metni X.680'den (ISO/IEC 8824-1) X.660'a (büyük ölçüde ASN.1 çalışmasının dışından) taşımak için ITU-T'den (büyük ölçüde ASN.1 çalışmasının dışından) baskı geldi (ISO/IEC 9834-1). ASN.1 grubunun kendi içinde bir miktar muhalefet vardı, ancak hareket gerçekleşti ve ilgili metin X.680/8824'ten silindi ve X.660/9834'e yapılan bir referansla değiştirildi. O zamandan beri, sürekli tutarlılığı sağlamaya çalışmak için ilgili standartların koruyucuları arasında çeşitli bağlantılar olmuştur! Neyse ki, nesne tanımlayıcı ağacı üzerindeki çalışma uzun zaman önce tamamlandı ve çok kararlı. (Ama bir sonraki maddeye bakın!)

8.3 En üst düzey yaylar için savaş ve BAĞIL OID'lerin tanıtılması

CCITT'den ITU-T'ye isim değişikliği basit bir üst düzey isim değişikliğiydi, değil mi?
Ancak, nesne tanımlayıcı ağacının en üstteki yaylarından ikisinin "ccitt" ve "joint-iso-ccitt" olduğunu unutmayın.

Herkes ağacın tepesinde olmak ister, ancak bu durumda iyi sebeplerden dolayı - protokollerinin ayrıntılarını azaltır.

ITU-T, "itu-t" ve "joint-iso-itu-t" için (yeni numaralarla) iki yeni yay önerdi. Şekil III-13 ile ilgili metni okuyanlar, bu talebi kabul etmenin tamamen imkansız olmasa da çok zor olacağını fark edeceklerdir! Sonunda, yeni isimler mevcut yayların eşanlamlıları olarak kabul edildi (aynı sayıları koruyarak).

Bundan kısa bir süre sonra, uluslararası kuruluşlar tarafından bir üst yay kullanan nesne tanımlayıcı ad alanı için artan bir talep oldu. Kuruluşlar, tahsis ettikleri (ve protokollerinde kullandıkları) nesne tanımlayıcı değerlerinin, ağacın tepesine "asılsalar" daha kısa olacağını fark ettiler. ITU-R, Uluslararası Posta Birliği ve IETF, ISO ve ITU-T'den bazı üst düzey eğrileri çekip alma arzusunu ifade eden (çeşitli derecelerde güçlerle) kuruluşlar arasındaydı (kesinlikle tahsis edilenlerin hepsini asla kullanmayacaklardı). onlara).

Bu sorun bugün (1999) RELATIVE OID adlı yeni bir türün eklenmesiyle etkisiz hale getiriliyor gibi görünüyor. (Evet, yazıldığı sırada NESNE TANIMLAYICISI değil , OID'dir.) BAĞIL OID değeri, bazı (statik olarak belirlenmiş) kök düğümlerin altında bulunan nesne tanımlayıcı ağacının bölümlerini tanımlar ve bu değerlerin kodlamaları yalnızca ortak öneki atlayarak o kök düğümün altındaki düğümler.

Oldukça basit olan bu öneri, ortak ön ekin bir iletişim örneğinde iletilmesine ve sonra

daha sonra o iletişim örneğinde iletilen belirli göreceli oid değerleri ile otomatik olarak ilişkilendirilir.

(Kitap yazarken hızla güncelliğini yitirmemek her zaman için çok zordur - ya BAĞIL OID gibi şeyler hakkında konuşmazsınız ya da yayımlandıktan birkaç hafta sonra kitabın geri çekildiğini görme tehlikesiyle karşı karşıya kalırsınız. veya önemli ölçüde değişti.Ancak bu durumda, ASN.1'e yukarıda açıklandığı gibi ekleneceğinden oldukça eminim.)

9 GERÇEK tip

GERÇEK tip yeterince zararsız görünebilir, ancak aynı zamanda 1986 civarında tartışmaların da kaynağıydı.

Muhtemelen sadece akademik bir alıştırma - gerçek protokollerde kimse GERÇEK'i kullanmaz. Ama kendi hararetli anlarını üretti.

Herkes buna sahip olmamız gerektiği konusunda hemfikirdi, ancak nasıl kodlanacak? (Nihayetinde üzerinde anlaşmaya varılan gerçek kodlama Kısım II Kısım 2 madde 3.5'te tam olarak açıklanmıştır ve ilgili okuyucu buna başvurmalıdır.)

İkili ve karakter kodlamalarının biri olduğu birkaç sorun vardı. Her zamanki gibi, kolay uzlaşma her ikisine de izin vermekti, ancak bu daha sonra kanonik kodlamalara ihtiyaç duyulduğunda sorunlara yol açtı ve matematiksel olarak eşit olan 2 tabanı ve 10 tabanı değerlerinin farklı soyut değerler olarak kabul edildiğini söylemek gibi oldukça kirli bir aldatmaca yapılmak zorunda kaldı. ve bu nedenle standart kodlama kurallarında bile farklı şekilde kodlayın.

Ancak asıl sorun, ikili kodlama biçimindeydi. O zamanlar bilgisayar sistemleri için kayan nokta biçimleri için (oldukça yeni) bir standart vardı ve genellikle yazılımda kayan nokta işleyen kişiler tarafından kullanılıyordu, ancak mevcut donanım tarafından kullanılmıyordu (daha sonra çiplerde uygulandı). Doğal olarak, ASN.1 kodlamaları için bu formatın kullanılmasını savunanlar vardı.

Bununla birlikte, karşı argüman sonunda galip geldi (ve yine bunun doğru karar olduğunu düşünüyorum). Karşı argüman, kayan nokta biçimleri için fiili bir standarttan biraz uzakta olduğumuz ve önemli olanın, donanımınızın sahip olduğu kayan nokta birimi ile kolayca kodlanıp kodu çözülebilen bir biçim bulmak olduğuydu.

Bu ilke, örneğin, bir "işaret ve büyüklük" mantisinin ("ikinin tümleyeni" veya "birin tümleyeni" yerine) kullanılmasını dikte etti, çünkü "işaret ve büyüklük" diğer iki formun donanımı tarafından kolaylıkla üretilebilir veya işlenebilir. , ancak sohbet doğru değil. 3.5.2'de açıklanan "F" ölçeklendirme faktörünü içeren oldukça ilginç formatın (herhangi bir gerçek kayan noktalı donanım veya pakette mevcut olmayan) ortaya çıkmasına neden olan da bu ilkeydi.

Son olarak, "3.14159..." ve "2.7183..." gibi, aksi takdirde sonlu temsili olmayan "ortak ve önemli" sayıları tanımlayan belirli kodlamaları ve ayrıca "taşma" ve "sayı değil" gibi değerler, ancak sonunda eklenen tek şey, GERÇEK türüyle ilgili diğer şeylerin tanımlanması için bol miktarda kodlama alanıyla birlikte ARTI-SONSUZ ve EKSİ-SONSUZ'u tanımlayan kodlamalardı. sonra. Bu ek kodlamaları sağlama baskısı ortadan kalktı ve herhangi bir uzantı yapılmadı ve şu anda da olası görünmüyor.

10 Karakter dizisi türü - kısa tutmaya çalışalım!

"Karakterler" (ve bir "karakter"in tam olarak ne olduğu üzerine tartışma) için kodlamaların gelişim tarihi, ASN.1'den çok daha geniştir.

Bölüm II'de size tarih sözü verildi, işte burada! Web'de muhtemelen daha iyi geçmişler vardır - gidip onları arayın!

ASN.1 bu çalışmaya gerçekten katkıda bulunmadı, bunun yerine ASN.1 kullanıcılarının protokollerinde bu çeşitli karakter kodlama standartlarına açık ve basit bir şekilde başvurmalarına izin verebilecek kullanılabilir notasyona sahip olmalarını sağlamak için elinden gelenin en iyisini yaptı.

Ancak sonuç, ASN.1'deki karakter türlerinin sayısında yıllar içinde istikrarlı bir artış oldu ve şu anda oldukça eskimiş pek çok bagaj taşınıyor.

Kısım II Kısım 2, burada karakter kodlama şemalarının gelişim tarihinin ve bunun ASN.1 üzerindeki yıllar içindeki etkisinin bir açıklamasını sağlayacağımıza söz verdi.
Aşağıda, ASN.1 üzerindeki etkisi ile birlikte, bu tarihin ana bölümleri yer almaktadır (ancak ayrıntılar bazen eksiktir ve tam bir tarih değildir - bu diğer metinlere bırakılmıştır).

10.1 Baştan ASCII'ye

En eski karakter kodlama standartları, telgraf sistemi için ve delikli kağıt bant ve kartlarda kullanıldı. En eski formatlar, büyük harflere, rakamlara ve birkaç ek karaktere izin vermek için "alfa kaydırma" ve "sayısal kaydırma" için bir kodlama ile her karakteri (32 olası kodlama) temsil etmek için 5 bit kullanıyordu.

Beş bitlik kodlar, yedi bitlik kodlar. Ve sonra gelmek üzere, 16 bitlik kodlar ve 32 bitlik kodlar! Hiç kimsenin 64 bit kod önereceğinden şüpheliyim ... ama ikinci kez düşündüğümde, Microsoft Word yazı tiplerini vb. belirtmek için kaç bit alır? (Tamam, bu genellikle paragraf başına, karakter başına değil, ama gelecekte ...?)

Daha sonra 7 bitin sekizinci eşlik biti ile kullanımı fiili standart haline geldi ve bu, sonunda mevcut bilgisayarların 8 bitlik baytlarında kutsandı. ASCII kod seti, esasen 32 sözde "kontrol karakteri" (çoğu işlevi erken protokol paketlerinin çerçevelenmesiyle ilgili) ve 94 sözde "grafik karakteri" (yazdırma) ile en iyi bilinen 7 bitlik kodlamadır. karakterler), artı BOŞLUK ve DEL (sil). (DEL, elbette, hepsi birler konumunda - 127 ondalık - çünkü delikli kağıt bantta bir hata yaptıysanız yapabileceğiniz tek şey, geri kalan tüm delikleri delmekti - bir delik!).

ASCII, yaklaşık kırk yıldır karakter kodlama şemalarımızın temelini oluşturdu ve ancak şimdi değiştiriliyor. ASCII aslında, belirli karakter konumlarında bir dizi "ulusal seçeneği" tanımlayan uluslararası standart ISO 646'nın Amerikan varyantıdır ve diğer birçok ülke benzer (ancak farklı) ulusal varyantları tanımlamıştır. İngiltere varyantı genellikle (yanlış!) "UK ASCII" olarak adlandırıldı.

10.2 Karakter setlerinin uluslararası kaydının ortaya çıkışı

İlk bilgisayar protokolleri 7 bitlik kodlamalar kullandı ve sekizinci bitin bir eşlik biti olarak kullanılmasını sürdürdü. Bu nedenle, bugün e-posta üzerinden isteğe bağlı bir ikili dosya göndermek isterseniz, bunun yedi bitlik bir biçime dönüştürüldüğünü ve daha fazla veya daha fazlasının yapıldığını görüyoruz.

Dünyadaki tüm karakterler için kodlamalar sağlamak - ilk deneme ve fena değil.

boyutu daha az ikiye katlar! Daha modern protokoller (Web sayfalarına erişmek için kullanılanlar gibi), "tam sekiz bitlik şeffaflık" olarak adlandırılan şeyi sağlar ve sekizinci bit, kullanıcı bilgilerini taşıyabilen tamamen sıradan bir bittir.

Protokoller geliştikçe, bir eşlik bitinin kullanımı, eksiksiz bir bilgi paketinde bir hata tespit kodu olarak bir Döngüsel Artıklık Kodu (CRC) lehine çok hızlı bir şekilde bırakıldı ve karakter kodlama şemaları, 8 bitlik bir kodlamaya geçmekte serbestti. 256 karakteri temsil edebilir.

Bununla ilgili iki gelişme oldu: Bunlardan ilki 1973 gibi erken bir tarihte geliştirildi.
Bu, dünyadaki tüm karakterlerin temsili için bir çerçeve (ISO 646'ya dayalı) oluşturan ISO 2022 idi. (Korkarım aşağıdaki açıklama zorunlu olarak biraz basitleştirildi - sözde çok baytlı biçimler ve 2022'nin dinamik olarak yeniden tanımlanabilir karakter kümelerinden aşağıda bahsedilmiyor.)

ISO 2022'nin çalışma şekli, ASCII yapısının ilk iki sütununu (kontrol karakterlerini tutan 32 hücre), herhangi bir sözde C karakter kümesini içerebilen (temsil eden, tanımlayan) hücreler olarak ve kalan 94 konumu (kontrol karakterlerini tutan) tanımlamaktı. SPACE ve DEL konumları SPACE ve DEL olarak sabitlenmiştir) herhangi bir sözde G-setini içerebilen (temsil eden, tanımlayan) hücreler olarak. Ayrıca, C-kümesi konumları içinde, ASCII ESC karakteri her zaman bu kesin konumda tutulacaktı, bu nedenle C-kümesi karakterlerinin aslında yalnızca 31 kontrol işlevi olmasına izin verildi.

Eski parite biti, C0 ve C1 seti olarak adlandırılan C-kümelerinin kodlamaları için iki anlamdan birini (iki karakter kümesinden biri) tanımlamak için kullanılabilir. Kullanımdaki C setlerinden biri "shift-outer" ve "shift-inner" için kontrol karakterlerini içeriyorsa (bu, G setinin yorumlanmasını etkiledi, ancak C set kodlarını etkilemedi), o zaman bunların eski ile birlikte kullanılması kombinasyonu G0, G1, G2 ve G3 olarak adlandırılan dört adede kadar G-setine (kodlamalarına) eşlik biti etkinleştirilmiş referans .

Son olarak, her kayıt girişi için ASCII yapısındaki her konuma karakter atayacak olan bir C-kümeleri ve G-kümeleri kaydı kavramı vardı. Herhangi bir zamanda, en fazla iki C-set ve en fazla dört G-set C0, C1, G0, G1, G2 ve G3 konumlarına "belirlenebilir ve çağrılabilir". ESC karakterine (tüm C-setlerinde aynı pozisyonda olması gerekli, unutmayın) özel bir anlam verildi. Her kayıt girişi, herhangi bir kayıt girişini bir C0 veya C1 pozisyonuna (C girişleri için) veya G0 ila G3 pozisyonlarından birine (G-girişleri için) "belirlemek ve çağırmak" için ESC karakterini takip edebilen ikili kodların özelliklerini içeriyordu.).

Geriye kalan tek şey kayıt girişlerini yapmaktı! Bu, genellikle "karakter kümelerinin uluslararası kaydı" olarak anılan "Escape Dizileriyle kullanılacak Uluslararası Kodlu Karakter Kümeleri Kaydı" oldu.

Kayıt, başlangıçta Avrupa Bilgisayar Üreticileri Derneği (ECMA) tarafından tutuldu ve neredeyse tüm dünyadaki karakter setlerini kapsayan 200'den fazla girişe ulaştı.
Bugün, BSI ve ANSI ile AFNOR ve DIN'in Japonca eşdeğeri olan Japon Endüstriyel Standartlar Komitesi (JISC) tarafından sürdürülmektedir. Hem ECMA hem de JISC, ilgili taraflara ücretsiz kopyalar ve ücretsiz güncellemeler sağlar, ancak JISC artık her kayıt girişinin yer aldığı bir web sitesine sahiptir. (Bu siteye erişmek istiyorsanız Ek 5'e bakın).

ASN.1, GraphicString ve GeneralString ile ISO 2022 için tam destek sağlar ve diğer birçok karakter dizisi türünün tanımı için International Register'a güvenir.

10.3 ISO 8859 ise geliştirme

ISO 8859 çok daha sonra (1987'de) geldi ve bir dizi "parça" halinde geldi.

2022 şemasıyla ilgili sorun, yeni adlandırmalar ve çağrılar yapmak için ESC dizilerinin dahil edilmesi nedeniyle, karakterlerin kodlamalarının sabit uzunlukta olmamasıydı.

Verimli bir kodlama ile Avrupa dillerine tam kapsam sağlamak - ASN.1 tarafından göz ardı edilen bir standart! Uluslararası Standardizasyonda Avrupa kimin umurunda? (Avrupa Komisyonu Başkanı, lütfen bunu okumayın!)

ISO 8859, sabit (karakter başına sekiz bit) kodlama ile Avrupa dillerinin ihtiyaçlarını karşılamak üzere tasarlanmıştır. 8859'un her bir parçası, ASCII'yi sözde "sol yarısı" olarak tanımladı - eski parite biti sıfıra ayarlı olarak elde ettiğiniz kodlama ve "sağ yarısında" çeşitli Avrupa standartlarının ihtiyaçlarını karşılamak için tasarlanmış 94 baskı karakteri daha. Diller. Bu nedenle 8859-1, "Latin alfabesi No.1" olarak adlandırılır ve ASCII'ye ek olarak, bir dizi başka karakterle birlikte mezar, inceltme işareti, vurgulu vurgular, cedillas, tildas ve umlauts karakterleri sağlar. 8859-6 "Latince/Arapça" olarak adlandırılır ve sağ yarısında arapça karakterler bulunur.

ASN.1, 8859 için herhangi bir doğrudan destek sağlamadı, ancak 8859 kodlamaları Avrupa'daki bilgisayar sistemlerinde oldukça sık kullanılıyordu.

10.4 ISO 10646 ve Unicode'un ortaya çıkışı

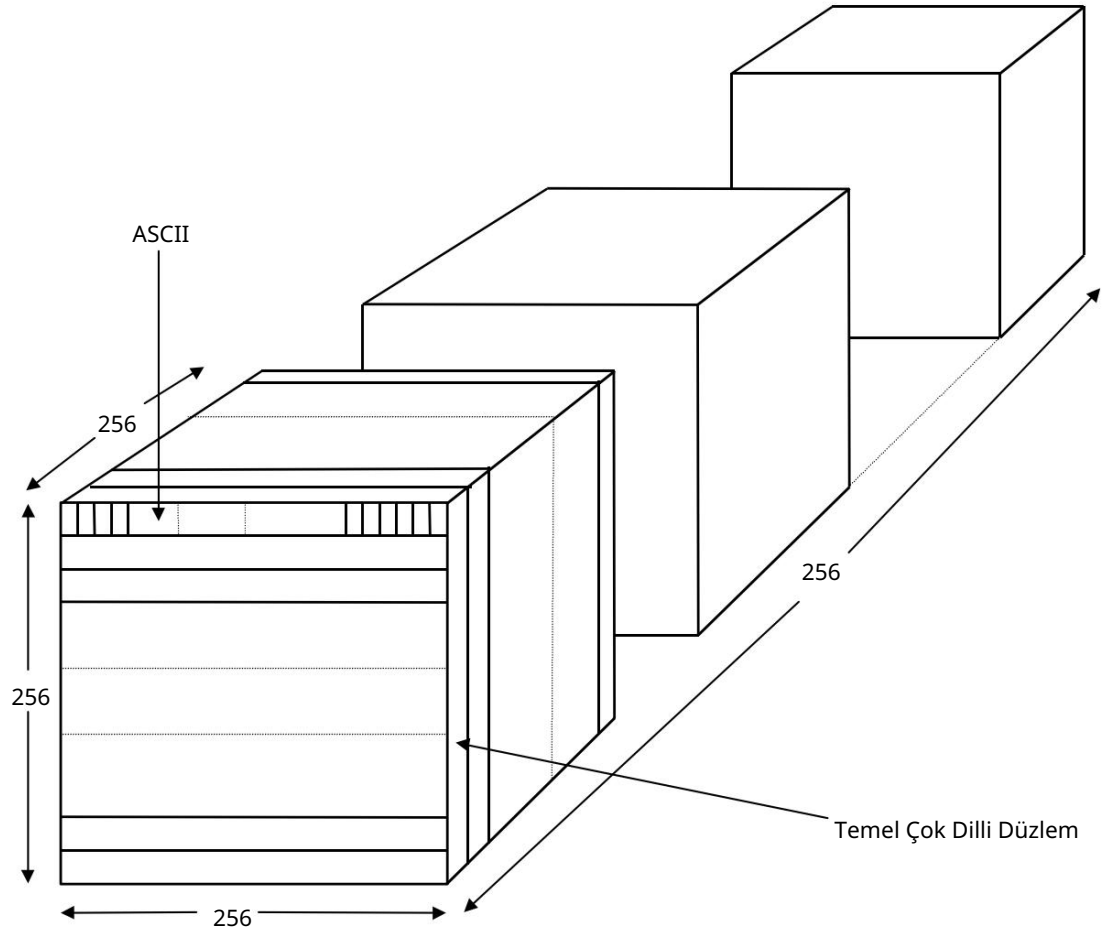
10.4.1 Dört boyutlu mimari

1990'ların başındaki çok büyük bir gelişme (neredeyse on yıl sonra, tamamen bilgisayar sistemlerine ve protokollerine girecek şekilde), ASCII yapısıyla tamamen ilgisiz, karakterleri kodlamak için tamamen yeni bir çerçevenin geliştirilmesiydi. (Ama elbette ASCII karakterlerini kodlayabilir!)

Muhtemelen şimdiye kadarki karakter seti kodlama çalışmalarındaki en önemli gelişme. Gelecekte herhangi bir zamanda bu mimariden olası bir değişiklik görmek zor. Vay! Gelecekte HERHANGİ bir zamanda? Aynen.

Burada şekil IV-1'e bakmalısınız (evet, bu bölümdeki ilk şekil - kendinizi mahrum hissediyor olmalısınız!). Bu, dört boyutlu bir yapı gösterir (ASCII 2 boyutlu kod tablosu ile karşılaştırıldığında).

Şekil IV-1 256 evden oluşan bir sokağı göstermektedir. Her evin içinde 256 "uçak" vardır (dikey olarak konumlandırılmış ve sokaktaki evin içinde soldan sağa doğru uzanır). Her düzlemde 256 sıra vardır (her evin her düzleminde yukarıdan aşağıya doğru). Ve her satırın içinde 256 hücre vardır (her satırda soldan sağa doğru). Her hücre farklı bir karakter içerebilir (tanımlayabilir, temsil edebilir).
(Aslında, bir ev için doğru teknik terim "grup"tur - "ev" kullanılmaz, ama ben onlara ev demeyi tercih ederim!)



Şekil IV-1: Her biri 256 hücreli, her biri 256 sıralı 256 düzlemli 256 ev

İlk evin (sıfır numarası) ilk düzlemine (sıfır numarası) Temel Çok Dilli Düzlem veya "BMP" denir. Bu düzlemin ilk satırı Latin Alfabesi No 1'i (8859-1) içerir ve dolayısıyla sol yarısında ASCII içerir.

(ISO 10646'nın ilk taslaklarında, 8859'un diğer bölümleri birbirini izleyen satırları işgal ediyordu ve bu nedenle ASCII birden çok kez göründü, ancak bu, Unicode ile "savaşta" kaldırıldı (aşağıya bakın) ve 8859'un diğer bölümleri yalnızca sağ el yarıları mevcut.)

Herhangi bir evin herhangi bir düzlemindeki herhangi bir satırın herhangi bir hücresinin, 0 ila 255 arasında dört değerle, yani 32 bit ile tanımlanabileceğine dikkat edin. Dolayısıyla, temel biçiminde ISO 10646, karakter başına 32 bitlik bir kodlama şemasıdır.

ASCII karakterleri için bu 32 bitin sayısal değerinin, 7 bitlik ASCII'deki karakterlerin yalnızca sayısal değeri olduğuna da dikkat edin - ilk 25 bitin tümü sıfırdır!

Şimdi, hayatın üzücü bir gerçeği, eğer

Dünyada var olan tüm karakterleri alırsınız ("a-grave" ve "inceltme işareti" gibi şeyleri ve hatta Tay dilinde kullanılan daha karmaşık karalama kombinasyonlarını sabit uzunlukta bir kodlama gerektiren ayrı ve farklı karakterler olarak tanımlarsınız); ve

Batılı bir göze son derece benzer görünen Çin, Japon ve Kore yazılarındaki gliflerin (karalamaların) aslında ayrı kodlamalara ihtiyaç duyan farklı karakterler olduğunu kabul ediyorsunuz; ve

Mısır mezar taşlarına ve en derin Afrika'da uzun süre korunmuş ağaç kabuğuna oyulmuş tüm karalamaları dahil ediyorsunuz; ve

8859'un her bir bölümünün tamamını BMP'nin ardışık satırlarına koyarak ASCII'yi birden çok kez dahil edersiniz; sonra

kodlamak isteyeceğiniz 2 üssü 32 "karakter"in yakınında hiçbir yerde olmadığını, ancak 2 üssü 16'dan çok daha fazla olduğunu bulursunuz.

ISO 10646 yapısı, bu tür tüm karakterlerin karakter başına sabit 32 bit ile temsil edilmesine izin verir, ancak bu aşırı öldürme midir? Mantıklı bir budama yaparsak, karakter başına sadece 16 bit ile idare edebilir miyiz?

10.4.2 Unicode'u Girin

(Web'deki Unicode materyaline bir işaretçi için Ek 5'e bakın).

> Üreticiler kaslarını esnetiyor. Karakter başına 32 bit, ticari açıdan önemli karakter kümeleri için gerekli veya mantıklı değildir! 16 bit çalışmak için yapılabilir.

ISO grubu JTC1 SC2, ISO 10646'yı geliştirmek için çaba sarf ederken, bilgisayar üreticileri bağımsız olarak bir araya gelerek ne ISO 2022 ne de ISO 8859 şemalarının dünyanın giderek artan küresel iletişim altyapısı ve metin işleme gereksinimleri için yeterli olmadığını kabul ediyorlardı. karakter başına tam 32 bit'e gitme konusunda dalga geçtiler. 16 biti yeterli yapamaz mıyız?

Pekala, yukarıda alınan bazı kararları tersine çevirebiliriz. Mısır hiyerogliflerini ve sadece kütüphanecileri ilgilendiren her şeyi görmezden gelelim. Gibi karalamalar yapabileceğimiz karakterleri birleştirme kavramını da tanıtalım (bu, Avrupa dilleri için fazla bir tasarruf sağlamaz, ancak Tayca gibi Doğu dilleri için çok şey kazandırır). Tabii ki, bir bakış açısından, karakterleri birleştirmenin kullanılması, artık her karakter için sabit uzunlukta bir kodlamaya sahip olmadığımız anlamına gelir, ancak bu, karakterin ne olduğuna ilişkin tanımınıza bağlıdır!

Son olarak, bir "birleşik kod" veya "Unicode" üretmek için "Han birleştirme" veya "CJK Birleştirme" gerçekleştirelim. CJK Birleştirme, Çince (C), Japonca (J) ve Korece (K) yazılarındaki karalamalara batılı bir gözle bakmamız ve benzer karalamaların üçünü de tek bir yazıya atayabilmemiz için yeterince benzer olduklarına karar vermemiz anlamına gelir. evlerimizin sokağında hücre.

Şimdi onu kırdık! Dünyada iki üzeri on altı (önemli) karakter vardır ve hepsini Temel Çok Dilli Düzlem'e sığdırabilir ve onları temsil etmek için karakter başına yalnızca 16 bit kullanabiliriz.

Tabii ki, ISO 10646 taslağının onaylanması için son oylama yapıldığında, "Unicode ile değiştirin" diyen büyük "HAYIR" oyları vardı!

10.4.3 Nihai uzlaşma

ISO 10646, 1993 yılında Uluslararası Standart olarak yayınlandı (yaklaşık 750 sayfa uzunluğunda!) ve Unicode belirtimi 1992'de Unicode Konsorsiyumu adına Addison Wesley tarafından yayınlandı ve Versiyon 2 1996'da çıktı.

Ve uluslararası standardizasyonla ilgili şaşırtıcı olan şey, genellikle tavizlere ulaşılması ve standartların üzerinde anlaşmaya varılmasıdır.

Unicode ve ISO 10646 hizalandı: CJK birleştirme ve karakterleri birleştirmenin dahil edilmesi üzerinde anlaşmaya varıldı ve ISO 10646'nın Temel Çok Dilli Düzlemi, Unicode spesifikasyonunda görünen karakterlerle tam olarak aynı karakterlerle dolduruldu ve yakın işbirliği o zamandan beri devam ediyor.

Ancak, iki metinde önemli farklılıklar kaldı. ISO metni, ISO 10646'nın üç "uygulama düzeyini" açıklar. 1. düzeyde, karakterlerin birleştirilmesi yasaktır. Her şey aynı sayıda bit ile kodlanmıştır, tüm sokağı istiyorsanız 32 (UCS-4) bit veya Temel Çok Dilli Düzlem'deki karakterleri istiyorsanız 16 (UCS 2) bit. 2. seviyede, karakterleri birleştirmeyi kullanabilirsiniz, ancak yalnızca istediğiniz karakter kalabalık bir hücrede yoksa (bu, "a" nın "a-grave" elde etmek için "grave" birleştirme karakteriyle kullanılmasını yasaklar). 3. seviyede her şey olur. Unicode bu seviyeleri tanımlamaz, ancak mümkün olan her yerde birleştirici karakterleri kullanmak Unicode'un ruhuna uygundur.

Metinler arasında, karakter kodlamasıyla ilgili olmayan (ve bu nedenle ASN.1 ile ilgisi olmayan) başka farklılıklar da vardır: Unicode belirtimi, hangi karakterlerin sayı, büyük/küçük harf eşlemeleri olarak kabul edilmesi gerektiğini söyleyen bazı mükemmel sınıflandırma materyalleri içerir. , ve benzeri; böyle bir metin ISO 10646'da eksik.

Unicode'un 1. Versiyonu ve ISO 10646'nın ilk yayınlanmasından sonra çalışmalar devam etti. Artık BMP'nin dışındaki hücrelerde karakterler var, ancak her iki grup da BMP'de ayrılmış karakterleri çıkış karakterleri olarak kullanarak 16 bitlik bir kodlama şemasında (UTF-16 - Evrensel Dönüşüm İşlevi 16 olarak adlandırılır) bunlara başvurmak için bir mekanizma üzerinde anlaştılar . diğer uçakları BMP pozisyonuna etkili bir şekilde belirleyin ve çağırın (kullanılan terminoloji bu olmasa da).

Bir başka son derece önemli gelişme, Bölüm II Bölüm 2'nin 12. maddesinde kısaca açıklanan UTF-8'in tanımıydı. Bu, karakter başına değişken sayıda sekizli sağlar, ancak tüm ASCII karakterleri, normal ASCII kodlamasıyla yalnızca bir sekizli ile temsil edilir (üst bit - eski eşlik biti - sıfıra ayarlanmış).

Programlama dillerindeki (ve işletim sistemi arayüzlerindeki) karakterlerin çekirdek içi işlenmesi için, bilgisayar satıcıları 16 bit (genellikle) veya 32 bit (bazıları) veya her iki karakter temsilini desteklemektedir. Ancak diskte depolama veya aktarım için UTF-8 çok popüler bir format olduğunu kanıtlıyor.

10.5 Ve tüm bunların ASN.1 üzerindeki etkisi?

Karakter seti cephesinde, ASN.1 zımbalarla yeni yuvarlandı. Ne repertuar tanımlarına ne de kodlamalara ciddi bir katkısı olmamıştır. Yapmaya çalıştığı şey, karakter seti standartlarına atıfta bulunmak için basit notasyonel destek sağlamaktır.

Karakter kümeleri için mevcut ASN.1 desteği Bölüm II'de açıklanmıştır ve okuyucunun bu metni karakter kümesi standartlarının geliştirilmesiyle ilişkilendirmesi artık mümkün olacaktır.
ASN.1'de karakter seti çalışmasının geçmişi

ancak, kullanıcılarının taleplerini karşılamaya çalışmak için uzun ve zorlu bir mücadele olmuştur. Her zaman herkesi mutlu etmeyi başaramadı!

X.409, ISO646String (kontrol karakterleri yok) ve IA5String (kontrol karakterleri dahil) tanımında kullandığı ISO 646 (CCITT International Alphabet #5'e eşittir) dışında hiçbir ISO karakter seti standardını kullanmadı. "ISO646String" hala izin verilen bir türdür, ancak "VisibleString" eşanlamlısı tercih edilir. NumericString ve PrintableString de X.409'da mevcuttu, ancak karakter repertuarları ve ASN.1'e fiziksel olarak bağlanmış kodlamalarla (bugün hala oldukları gibi).

X.409'daki diğer iki karakter dizisi türü, T61String (bugün TeletexString'in tercih edilen eşanlamlısı ile) ve VideotexString idi;

1980'lerin başında, ISO standartları yazarlarının ISO standardı olmayan herhangi bir spesifikasyona atıfta bulunmak için özel izin almaları gerekiyordu, bu nedenle TeletexString ve VideotexString bazı sorunlar çıkardı. Tanımların (hiçbir teknik değişiklik olmadan!) daha önce açıklanan uluslararası karakter kümeleri siciline referanslar açısından yeniden biçimlendirilmesine karar verildi (ISO 8824 ve ISO 8825'i oluşturan yeniden yazma işlemi yapıldığında) ve bu, başarıyla tamamlandı (bazı yeni kayıt girişleri ekleyerek!).

Aynı zamanda, International Register'a tam destek sağlamak için GraphicString ve GeneralString eklendi.

Bununla ilgili iki sorun vardı: Birincisi, kayıt defterine sürekli olarak yeni girişler yapılıyordu, bu nedenle GraphicString ve GeneralString uygulamalarının gerçekte ne anlama geldiği çok açık değildi - bunlar açık uçlu özelliklerdi. İkincisi ve belki de daha önemlisi, TeletexString'i belirli kayıt girişlerine bir referans olarak yeniden biçimlendirmek, onu 1984 T.61 tanımında etkili bir şekilde "dondurdu", ancak birçok ülke komut dosyalarını teleteks Tavsiyelerine eklemek için (başarılı) girişimlerde bulundu ve (belki de) oldu. ASN.1'deki TeletexString'in resmi tanımının hala bir parçası olmadıkları için sinirlendiler!

Sonunda, ASN.1'de TeletexString'i değiştirmeye yönelik siyasi baskı çok büyük hale geldi ve 1994'te, bir TeletexString kodlaması içinde atamak ve çağırmak için izin verilen girişler olarak bir dizi yeni kayıt girişi eklendi. Mevcut protokollerin mevcut uygulamaları ne olacak?
Siyasi baskı, bunun gibi küçük teknik meselelere saygı duymaz! TeletexString'in resmi tanımı değişti!

Bazı üzüntülere neden olan başka bir değişiklik daha oldu. Resmi olarak, VisibleString ve IA5String, ISO 646'nın sözde "Uluslararası Referans Sürümü" olan 2 numaralı kayıt girişine atıfta bulundu (ancak neredeyse herkes - yanlış bir şekilde - bunu "ASCII" olarak yorumladı). Ancak ISO 646, 1980'lerin sonlarında ASCII'de bulunan ancak ISO 646'nın Uluslararası Referans Versiyonunda bulunmayan "dolar" karakterini tanıtmak için değiştirildi. Bu nedenle ASN.1, referansı # 6 kayıt girişi (ASCII) olarak değiştirdi. Aynı zamanda tüm GraphicString ve GeneralString kodlamalarının başında ayarlanan varsayılan G0 ayarını #2'den #6'ya değiştirdi. Bu, X.400 grubundan büyük bir öfkeye neden oldu ve artık bu kodlamalarda G-kümelerinin özel olarak belirlenmesini ve kaçış dizileri tarafından çağrılmasını ve bir varsayılanın varsayılmaması gerektiğini tavsiye ediyor.

Sonra ISO 10646 geldi ve ASN.1 grubu, ISO çalışanları ile Unicode çalışanları arasındaki tartışmaları ilgiyle, ancak kenardan izledi. Bir uzlaşmaya varıldığında ve ISO 10646 yayınlandığında, kolay görünüyordu: ASN.1, çoklu ortamdaki karakterler için UniversalString (UCS-4 32-bit kodlama) ve BMPString (UCS-2 16-bit kodlama) olmak üzere iki yeni tür sağladı. -dil düzlemi. UCS-2 ve UCS-4, International Register'ı kullanarak kodlamalara kaçış sağladı - GeneralString kodlamalarını UniversalString'e etkili bir şekilde gömme yeteneği

veya BMPString. Basitlik açısından ASN.1, bu kaçış mekanizmalarını ASN.1 kodlamalarında kilitledi ve bugün yine deneyimli kullanıcılardan bazı şikayetler verdi!

Daha ciddi bir sorun, 1994 ASN.1 yayınında mürekkebin kurumasından hemen sonra, daha önce açıklanan UTF-8'in (ve UTF-16'nın) ISO 10646 ve Unicode'a değişiklikler olarak gelmesiydi.
UTF8String, 1997 sürümünde ASN.1'e eklendi, ancak bu yazı yazıldığı sırada UTF-16 için destek yok - ancak bunu sağlamak için biraz baskı var!

Bu karakter seti ve kodlama tartışmasında "alttan çıkmak" amacıyla ASN.1, 1994 yılında ISO 10646'da bir ek (ama yalnızca bilgilendirici!) karakter repertuarlarını (yukarıda açıklanan seviye 1 veya seviye 2 kısıtlamaları dahil) ve kodlama şemalarını (UCS-2 ve UCS-4) tanımlamak için kullanılacak belirtilen nesne tanımlayıcı değerleri.

"KARAKTER DİZİSİ" türünün başlangıçta çok verimli olması amaçlanmıştı, nesne tanımlayıcıları karakter özetini tanımlamak ve bir "KARAKTER DİZİSİ DİZİSİ" içindeki karakter dizilerinin sözdizimlerini aktarmak için yalnızca bir kez iletiliyor. Ne yazık ki, bunu sağlamak için kullanılan mekanizmada bazı ölümcül hatalar olduğu ortaya çıktı ve geri çekildi. Daha sonraki bir "dinamik kısıtlamalar" veya "çalışma zamanı parametreleri" mekanizması, eşdeğer destek sağlamaya çalıştı, ancak güç / karmaşıklık oranının çok düşük bulunması nedeniyle başarısız oldu.

(Bu, bu bölümün son maddesinde daha ayrıntılı olarak ele alınmıştır.)

ASN.1 ayrıca ISO 10646'daki karakter "koleksiyonlarının" adlarından ASN.1 (alt)tür adlarına eşlemeler sağladı ve ISO 10646'nın farklı "uygulama düzeylerine" karşılık gelen (alt)tür adları sağladı ve değer 10646'daki karakterlerin her biri için referanslar. (Bkz. Kısım II Bölüm 2.).

Bu, bugüne kadarki tarih, ancak bu alanı izleyin! Bence karakter kümeleri ve kodlamalar destanı muhtemelen henüz bitmedi!

11 HERHANGİ BİRİ, makrolar ve Bilgi Nesneleri - bu kadar kısa tutmak zor (başlık bile iki satıra çıkmış)!

Pekala, belki kısa tutabiliriz - bilgi nesnesi kavramı daha önce iyi ve tam olarak tartışılmıştır ve HERHANGİ BİR makro ve ASN.1'den 1994'te geri çekilmiştir, bu nedenle belki de söylenecek fazla bir şey yoktur!

> Bunların çoğunu (eğer baştan sona okuyorsanız!) zaten biliyorsunuz. Tarihsel ipleri birlikte çekelim.

Hikaye, yukarıda açıklandığı gibi İŞLEM ve HATA sözdiziminin 1982/83'te ASN.1'e dahil edilmeye çalışılmasıyla başlar.

Bu girişim başarısız oldu ve makrolar tanıtıldı. Makro notasyonun gerçekten sağladığı şeyin (sağlıyormuş gibi göründüğünü unutun!), ASN.1'e gelişigüzel sözdizimsel uzantıları (ancak bu uzantıları diğer ASN.1 yapılarıyla ilişkilendirecek anlambilim olmadan) tanımlama yeteneği olduğu ortaya çıktı.
1986 yılına kadar tanımlanmış sadece iki makro vardı. Bunlar ROSE'da tanımlandı ve (sürpriz, sürpriz!) OPERATION ve ERROR olarak adlandırıldı ve bu makroları içe aktaran herhangi bir ASN.1 modülünün daha önce açıklanan OPERATION ve ERROR sözdizimini tam olarak yazması sağlandı.

Tabii ki, gerçekte olan şey (ancak bu yalnızca yaklaşık beş yıl sonra fark edildi), sözdiziminin ROSE kullanıcılarına ROSE protokolünü - ASN.1 türlerini tamamlamak için gereken bilgileri sağlamak üzere oldukça kolay bir sözdizimi sağlamak için sağlanmasıydı. ve ROSE mesajlarında taşınacak işlem ve hataların tanımı ile ilgili değerler.

Başka bir deyişle bilgi nesneleri. Ancak makro gösterimi ROSE'a istediği sözdizimini tanımlama yeteneği verirken, altta yatan bilgi nesnesi kavramları eksikti ve bu sözdiziminin kullanımının (bir işlem veya hatayla ilişkili bilgileri tanımlamak için) ROSE mesajlarıyla resmi bir bağlantısı yoktu.

1986 civarında yeni makroların yazılmasında ani bir patlama oldu. ASN.1 kullanan hemen hemen her grup, ASN.1 gösterimine yeni sözdizimi ekleme ihtiyacı bulmuş gibi görünüyordu. Hepsi ne yapıyordu?

Pekala ... küresel bir resim açısından kimse gerçekten bilmiyordu. Bu yeni sözdiziminin kullanımları çok ve çeşitliydi ve işlemlerle veya hatalarla hiçbir ilgisi yoktu. Ayrıca, araç sağlayıcılar makro notasyondan şikayet etmeye başlıyordu.

Bir ASN.1 türünü tanımladığını iddia eden, ancak değer notasyonu (bir değer referans atamasındaki değer gösterimi, veya bir SET veya SEQUENCE öğesinde VARSAYILAN kullanımı).

İki önemli sorun daha vardı.

Birincisi, ASN.1 kullanıcılarına (makro notasyonu aracılığıyla), Bacchus-Naur Form (BNF) gösterimini kullanarak ASN.1'e keyfi olarak karmaşık sözdizimsel uzantıları tanımlama yetkisi verildi. BNF, programlama dillerinin sözdizimini tanımlamak için sıklıkla kullanılan (ve aslında ASN.1 notasyonunun kendisinin sözdizimini resmi olarak tanımlamak için kullanılan) son derece güçlü bir notasyondur. Bununla birlikte, programlama dillerinin tanımlayıcıları ve BNF'nin diğer kullanıcıları, ortaya çıkan sözdiziminin bilgisayar dostu olması (bilgisayarlar tarafından kolayca ayrıştırılması) isteniyorsa, BNF tanımında bazı orta derecede karmaşık ve karmaşık kısıtlamalara uyulması gerektiğini iyi bilirler. . ASN.1'de kullanımına böyle bir kısıtlama uygulanmadı.

İkinci sorun, bir makro tarafından tanıtılan yeni bir sözdiziminin sonunu, o makronun ayrıntılarını bilmeden bulmanın genellikle mümkün olmamasıydı. Ancak makronun tanımı, makro adının ve dolayısıyla yeni sözdiziminin ilk kullanımını pekala takip edebilir.

Hay aksi! Alet satıcıları bundan hoşlanmadı! Daha iyi araçlardan bazıları, bilinen uluslararası standartların çoğunda makrolar tarafından tanımlanan sözdizimine ilişkin araç bilgisine donanımsal olarak bağlandı ve ardından araca sağlanan gerçek sözdizimi tanımını (makro tanımı) basitçe göz ardı etti. İşe yaradı ama ....

1988 civarında ABD, SC21 içinde yeni makroların yazılmasına yönelik bir ambargo için güçlü bir kampanya yürüttü ve "makro gösterimi değiştirilene veya onunla ilgili sorunlar çözülene" kadar bu tür yeni makroları yasaklayan bir karar almayı başardı. Bu talebin, aslında ikame ile karşılanması yaklaşık beş yıl sürdü.

Bu zamanın çoğu, farklı grupların makroları tam olarak ne için kullandığını belirlemeye çalışmakla geçti ve sonunda gün aydınlandı ve neredeyse tüm durumlarda ASN.1 sözdizimindeki uzantıların tanımının (ROSE'da olduğu gibi) olduğu ortaya çıktı. Deliklerle dolu bir protokolün kullanıcılarına, bu deliklerin içeriğini belirtmek için insan dostu ancak resmi bir notasyon sağlamak amacıyla. Makro notasyonunun kullanımı (neredeyse) her zaman ASN.1 tanımlı mesajlarda "ANY" (ve daha sonra "ANY TANIMLANAN TARAFINDAN") kullanımıyla ilişkilendirilmiştir. (X.500'deki ŞİFRELİ makro gibi önemli istisnalar vardı, burada yeni sözdizimi gerçek bir

daha sonra bu metinde daha önce açıklanan kullanıcı tanımlı kısıtlama ve parametreleştirme kullanılarak karşılanan ASN.1 uzantısı.)

Bu sıralarda (1980'lerin sonu, 1900'lerin başı) "HERHANGİ BİRİ" ile ilgili sorunlar daha geniş çapta kabul görmeye başladı (bunlar 1985 gibi erken bir tarihte, "HERHANGİ BİRİ"ni "HER TANIMLAYAN" ile destekleme girişimleriyle işaretlenmiş olsalar da).

Hangi makroların kullanıldığını anlama ve makrolar ve HERHANGİ bir yerine uygun bir alternatif tanımlama girişimi, birkaç yıl boyunca birçok yinelemeden ve yanlış başlangıçlardan geçti.
"Kodlanamayan türler" ve "tablo türleri" icat edilen ve atılan terimlerdi.

Sonunda bir şey neredeyse hazırdı ama karmaşıktı ve terminoloji net değildi.
Kritik bir toplantı vardı (Sanırım Seul, Kore'de ve bunun Bancroft Scott'ın ilk uluslararası ASN.1 toplantısı olduğundan oldukça eminim), makroların yerine geçecek bir şey bulamamışız gibi göründü - önceki çalışma sadece çok karmaşık. Ancak uykusuz geçen bir gecenin ardından çözümler ortaya çıkmaya başladı. Ertesi gün Bilgi Nesnesi Sınıfı konseptini tartışmaya başladık ve işleri basit tutmak için sadece (örn.)'e izin vermeye karar verdik:

Operasyon türü

herhangi bir kısıtlama uygulanmadan. (Hala pişman olduğum bir şey!)

Ama Seul toplantısı iyi bir toplantıydı. (Başlangıçta) birkaç yıllık çalışmanın terk edilmesi gibi görünen şey, Bilgi Nesnesi Sınıfı terminolojisi ve bugün bildiğimiz şekliyle ilgili kavramlarla sona erdi.

Kısa bir süre sonra, başka bir önemli toplantı (muhtemelen alınan kararın büyüklüğünü kimsenin gerçekten anlamadığı) 1991 civarında gerçekleşti - Washington sanırım (odayı hatırlıyorum ama yerini hatırlayamıyorum!). Bu toplantı ASN.1'den çekilme kararı aldı:

Tüm makro gösterimi.

ANY ve ANY TANIMLI BY sözdizimi.

Bunların yerini, bilgi nesnesi sınıflarını, nesneleri ve kümeleri tanımlama gösterimi ve ilişkili "nesne sınıfından bilgi" gösterimi ve tablo ve ilişkisel kısıtlamaların uygulaması alacaktı.

Bu sıralarda, Birleşik Krallık Hükümeti hakkında bir memurun bir Kabine Bakanına sık sık "Bakan, bu sizin için çok cesurca" dediği popüler bir Birleşik Krallık televizyon dizisi vardı. Bakan yüzünü buruşturur ve neredeyse anında teklifini geri çekmeye çalışırdı.

Kimse ASN.1 grubuna makroyu ve ANY ve ANY DEFINED BY notasyonunu geri çekerken "çok cesur" olduklarını söylemedi, ama birisi söylemeliydi! Söylenseler bile geri adım atarlar mıydı bilmiyorum ama eminim ki olumsuz tepkinin boyutu tahmin edilmiyordu.

Bu, ASN.1'in yirmi yıllık (bugüne kadarki) geçmişinde geriye dönük olarak uyumlu olmayan ilk (ve tek) değişiklikti ve neredeyse bir yıl boyunca devam eden "ASN.1 1990 sorununa" - aşağıya bakın - yol açtı. on yıl.

12 ASN.1(1990) tartışması

> Asla, asla, asla daha önce yasal olan bir şeyi yasa dışı yapan bir şartname üretmeyin. Eğer yaparsan, pişman olacaksın! Ama belki bazen kötü bir özellikten kurtulmanın tek yolu budur?

ASN.1'in 1994 sürümü yayınlandığında, insanların ANY ve ANY TANIMLI BY ve makroların kullanımından bilgi nesnesi kavramlarının kullanımına yönelik özelliklerini değiştirmelerini sağlamak için eşlik eden bir kampanya vardı. Bence ASN.1 grubu, bunun "hattaki herhangi bir parçayı" değiştirmeyeceğinden, bunun büyük bir sorun olmadığını hissetti! Ancak, elbette , "kararlı" olan ve hemen yeni bir sürümde yeniden yayınlanmak üzere olmayan bir spesifikasyonda yapılan herhangi bir değişiklik (tek bir virgül eklemek için bile) aslında maliyetli bir uygulamadır. Kazançlar görünür olmalıdır.

ASN.1 grubunun şüphesi yoktu: makro gösterimde ve ANY'nin kullanımında çok fazla kusur vardı ve bilgi nesnesi kavramları ve ilgili gösterim çok daha iyiydi. Herkes geçiş yapmalıdır. Bir geçiş planı kabul edildi. Orijinal ROSE OPERATION ve ERROR makrolarında makro gösteriminin çoğu kullanıldı. Dolayısıyla, ROSE'un 1994'te değişeceği (eski makro tanımını bilgilendirici bir ek olarak koruyarak) ve ROSE kullanıcılarının en geç 1998'de değişeceği konusunda anlaşmaya varıldı.

Yeni spesifikasyonlar (SET - Secure Electronic Transactions gibi) elbette, bu kitabın okuyucuları(!) gibi, yeni kavramları benimsemede hiçbir sorun yaşamadılar - bunlar, boşlukları olan protokollerin spesifikasyonunda önemli netlik sağladılar.

Kendi makrolarını tanımlayan ve halen genişletilme sürecinde olan X.400 ve X.500 gibi spesifikasyonlar da kararlaştırılan zaman dilimini iyileştirdi. Yeni gösterimin daha net olduğunu fark ettiler ve 1990'ların başlarında ona geçtiler.

Ancak değişimi daha zor bulan ve daha uzun süre direnen bazı gruplar da oldu.
İlginç bir şekilde, ABD'nin yeni makrolar yazmaya koyduğu ambargo, protokolü neredeyse %50 "HER" (elbette abartıyorum!) onlar) protokollerini tamamlayacaktı.

Bu notasyona "Yönetilen Nesnelerin Genel Tanımı" (GDMO) adı verilir ve bugün o uygulamaya ve o notasyona özgü kendi araç seti tarafından desteklenir. Bu grup, ASN.1'in 1994 versiyonuna geçiş yapmak için en az teşvike sahip olan ve en uzun süren gruptur. ("ANY" kullanımlarının protokollerinden kaldırılması.)

Gözden geçirilmiş bir Standardın otomatik olarak önceki bir versiyonun yerini alması ISO'da normaldir. Eski sürümün artık satın alınamayacağı ve artık mevcut ISO Standartları kataloğunda kayıtlı olmadığı ve yeni Standartların eski sürüme atıfta bulunmasına izin verilmediği anlamında onun yerini alır.

ASN.1 (1994)'teki ASN.1 gösteriminin tanımı, ASN.1 (1990) tanımıyla geriye dönük olarak tam olarak uyumlu olmadığı için (ve ASN.1'e başvuran standartların kendi standartlarını güncellemesi için zamana ihtiyaç olduğunu herkes bildiği için) 1994 sürümlerine uyması için spesifikasyonlar), ASN.1'i (1990) "tutmak" için güçlü bir baskı vardı. ISO Merkez Sekreterliği, yıllık genel kurul toplantılarının her birinde SC21 tarafından bu yönde bir karar alınması koşuluyla, bunu kabul etti.

Elbette, bu kararlar, her yıl ASN.1 grubunun ASN.1 1990'ın geri çekilmesini giderek daha güçlü bir şekilde önermesiyle ve her yıl bazı grupların "henüz hazır değiliz" demesiyle, bir savaş alanının odak noktası haline geldi. ASN.1 (1990) nihayet rafa kaldırılmadan önce aslında 1999 yılıydı!

Bu yararlı bir ders oldu ve eğer bir ASN.1 toplantısında herhangi biri mevcut ifadeyle makul bir şekilde yasal olarak yorumlanabilecek herhangi bir şeyi yasa dışı kılacak bir değişiklik önermeye cesaret ederse, "1990, 1990" ulumaları duyulur ve teklif başarısız! Değişiklikler hattaki bitleri etkilemese bile, notasyon artık kutsaldır - çok fazla kişi kullanır ve mevcut spesifikasyonlar geriye dönük olarak yasadışı hale getirilemez.

13 PER'in ortaya çıkışı

13.1 İlk deneme - PER-2

Bunu "PER eksi 2" olarak telaffuz edin!

PER'i bugünkü konumuna getirmek için üç deneme gerekti - PER-2, PER-1 ve son olarak real-PER.

Daha iyi kodlama kuralları üretme çalışmaları, makroların nasıl kullanıldığını anlama ve makroları onarma veya değiştirme çalışmaları ile hemen hemen aynı zamanda başladı ve uzun bir süre bu çalışmanın gölgesinde kaldı ve yalnızca az sayıda insan gerçekten katkıda bulundu. yeni kodlama kuralları üzerinde çalışmak için.

Orijinal çalışma (bunu "PER-2" olarak adlandırmama izin verin, "PER eksi 2" olarak telaffuz edilsin!) BER'i kullanmaya ve onu "geliştirmeye" dayanıyordu. Tanıma, BER'in genellikle bir kod çözücünün (bir kodlayıcı tarafından kullanılanla aynı tip tanımına sahip olmaları koşuluyla) tamamen tahmin edebileceği sekizlileri iletmesiydi. O noktada gönderilmesi gereken buydu . Bu nedenle gönderilmesine gerek yoktu.

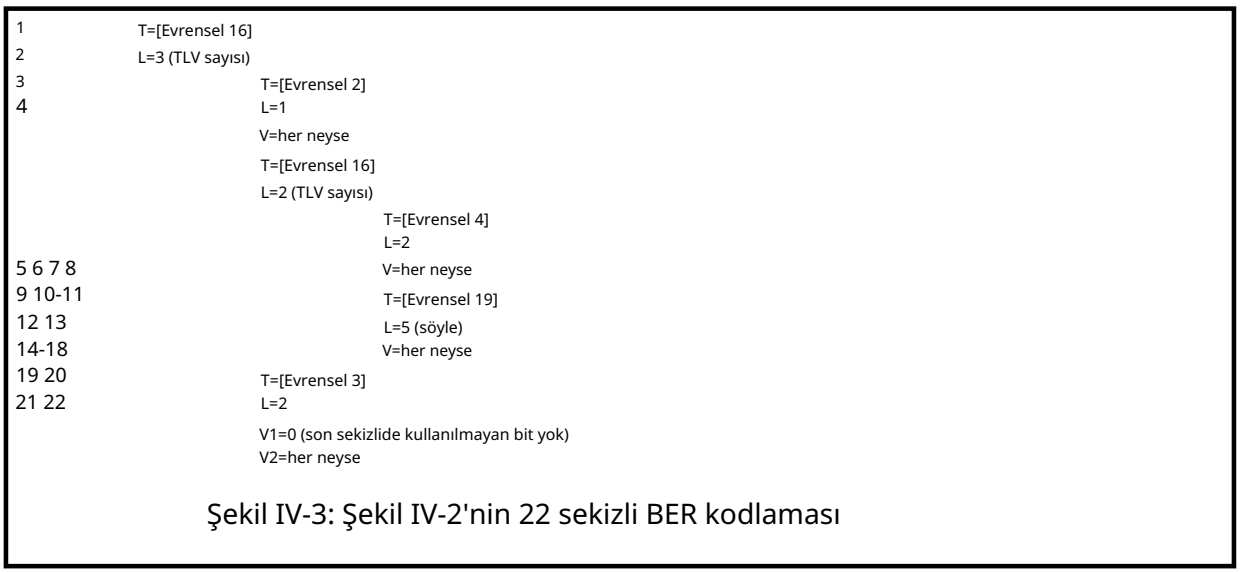
```
Kodlama için örnek ::= DİZİ {ilk öğe
                                                    INTEGER (0..127), ikinci öğe
                SEQUENCE {string OCTET STRING (SIZE (2)), name PrintableString (SIZE (1..8) ) }


                üçüncü eleman BİT DİZİSİ (BOYUT (8) ) }
```

Şekil IV-2: Kodlanacak örnek bir dizi

Ayrıca, yapılandırılmış bir kodlamanın uzunluk alanı, içeriklerdeki sekizli sayısı yerine içeriklerin kodlanmasındaki TLV'lerin sayısını sağlayacak şekilde değiştirilirse, daha fazla sekizlinin kaldırılabileceği kabul edildi. Ve son olarak, bir karakter dizisi alanının uzunluğu veya bir tamsayı boyutu üzerinde kısıtlamalar varsa, uzunluk alanlarının ihmal edilebileceği kabul edildi.

BER'deki bu değişiklikleri kabul edin ve kodlanacak türün (biraz uydurma) bir örneği olan şekil IV-2'yi ve bu türün BER kodlaması olan şekil IV-3'ü inceleyin.



Şekil IV-3: Şekil IV-2'nin 22 sekizli BER kodlaması

Şekil IV-3'e baktığımızda, BER kodlamasında 22 sekizlimiz var. Ancak 5, 10-11, 13-18 ve 22 sekizlileri (toplam 10 sekizli) hariç tümü bir kod çözücü tarafından tamamen bilinir ve asla iletilmesi gerekmez! PER-2 "sil onları!" dedi.

(İlginç bir şekilde, nihai real-PER spesifikasyonu bu erken yaklaşımdan tamamen farklı olsa da, mevcut real-PER'in ileteceği sadece bu 10 sekizli!)

PER-2 taslağı esas olarak şunları söyledi:

> Standart bir BER kodlaması yapın (yapılandırılmış kodlamalar için sekizli yerine TLV sayısını sağlamak için biraz değiştirildi).
>
> Kodlamadan sekizlileri silmek için aşağıdaki kuralları uygulayın.
>
> Alıcı tarafta, orijinal BER kodlamasını yeniden oluşturmak için kuralları tersine uygulayın.
>
> Standart bir BER kod çözme işlemi yapın (yine TLV sayımlarını kullanacak şekilde değiştirildi).

Sekizlileri ne zaman silebileceğinize ilişkin kurallardan bazıları açık ve anlaşılırdı, bazıları ise oldukça karmaşık hale geldi. Okuyucu, şekil IV-3'ün kodlamasındaki 22 sekizliden 12'sini silmemizi (aktarmamamızı) sağlayan kuralları tam olarak formüle etmeye çalışmak isteyebilir.

PER-2, kodlamaya gerçekten bir tür "uzman sistem" yaklaşımıydı. Sekizlileri ne zaman silip silemeyeceğinizi (makbuz üzerine yeniden ekleme ile) belirlemek için uygulanacak bir dizi kural vardı ve bunlar çok özeldi ve bir şekilde tamamlanmamış ve temel alınmamış gibi görünüyordu. herhangi bir iyi genel ilke. (Bunlar geçiciydi ve herhangi bir genel ilkeye dayanmıyorlardı!)

Ancak metin sonunda tamamlanmış kabul edildi ve oylamaya gönderildi. Oy pusulası yorumlarını değerlendirmek için düzenleme toplantısı New Jersey'de yapıldı ve bir hafta sürmesi planlandı (bu, değerlendirilen tek iş). Yönetimle ilgili bir şeyler ters gitti ve resmi Ulusal Organın oy pusulasına verdiği yanıtların kopyaları, toplantının ilk günü sabah 9:00'da yalnızca faks yoluyla alınabildi.

Yüzler düştü. Herkes kendi ülkesinin tepkisini biliyordu ama o zamana kadar başkalarının ne dediğini bilmiyorlardı. Her, evet her Ulusal Organ "ONAYLAMAYIN" oyu vermişti. Ve yorumların hiçbiri daha fazla ilerleme için hiçbir şekilde yardımcı olmadı. Aşağı yukarı hepsi "Bu çok karmaşık, çok geçici, asla işe yaramayacak" dedi. Hiçbiri PER-2 taslağını kabul edilebilir kılmak için değiştirilebilecek herhangi bir şey önermedi.

Toplantı o gün saat 11 civarında öğle yemeği için dağıldı ve birçok delege (beş veya altı ülkeyi temsil eden yaklaşık bir düzine vardı) havayollarını arayarak o gün geri uçmanın o gün yerine ne kadara mal olacağını öğrenmek için aradı. hafta sonunda planlanan uçuşları.
Diğer delegeler (ben dahil) acılarını bastırmak için bara çekildiler.

Yeterince bira tüketildikten sonra insanlar düşünülemez olanı düşünmeye başladılar. Neden TLV ilkesini bırakıp sıfırdan başlamıyoruz? Bir standardın farklı versiyonları arasında birlikte çalışmayı unutun (PER-2 zaten bunu gerçekten sağlamadı) - satırda minimum sekizli üretmek için maksimum insan zekasını kullanarak bir şeyleri nasıl kodlardık? "Sigara paketinin arkası" (aslında kağıt peçeteydi) tasarımı şekillenmeye başladı. (Keşke peçeteyi saklasaydım, ama sanırım WPB'ye emanet edilmişti. Önemli tarihi belgeler için bu kadar yeter!) Öğleden sonra saat 2'de başkan (Bancroft, Editör sanırım) "Bir araya gelip konuşalım mı?" dedi. Bu toplantıyı sonlandırmak mı?". "Hayır," o zamanlar hafifçe sarhoş olan bar grubunun yanıtıydı (sarhoş - asla!), "Bir yere varıyor olabiliriz." Sanırım toplantı sonunda o gün saat 16:00 civarında yeniden başladı. PER-1 (PER eksi 1), artık bildiğimiz şekliyle neredeyse PER (ama tam olarak değil) karşılanmıştı.

İlkeler yürürlükteydi:

> Etiketleri unutun - onları bırakın! (Bu açıklamayı yapmak için oldukça sarhoş olmanız gerekiyordu - TLV, kırılması zor olan bir tür zihniyetti.)
>
> Mümkün olduğunda uzunluk alanlarını kaldırmak için tamsayılar ve uzunluklar üzerindeki kısıtlamalar hakkındaki bilgilerden tam olarak yararlanın.
>
> SET elemanlarının rastgele sırada olması sorunu nasıl çözülür? Siparişi düzeltin! (Bunu da söylemek için biraz sarhoş olmalısın!)
>
> Bir SEÇİM'in seçilmiş bir unsuru nasıl belirlenir? Bir seçim dizini kodlayın.
>
> Bir DİZİ veya KÜMEDE eksik olan İSTEĞE BAĞLI öğeler nasıl belirlenir? SEQUENCE veya SET'in başında bir bit haritası kullanın.
>
> Bir BOOLEAN nasıl kodlanır - elbette, sadece bir bit kullanın!
>
> Ama .... sekizli hizalama? Zaman zaman dolgu bitlerine sahip olmanın iyi olduğunu kabul edin, böylece daha sonra sekizlilerin tam sayısı olan bir öğe dizisi olan malzeme bir sekizli sınırında yer alacaktır, ancak mantıklı göründüğü yerlerde sekizli hizalaması hakkında endişelenmeden minimum bit sayısını kullanın. .

Bu tasarıma hala "uzman sistem" yaklaşımının bazı unsurları vardı (mevcut PER'de olduğu gibi). Hangi alanların bit alanlarına (doldurma bitleri olmadan) ve hangilerinin sekizli hizalanmış bit alanlarına (doldurma bitleriyle) kodlanması gerektiğine ilişkin oldukça geçici bir karardır.

Çözülmesi gereken pek çok ayrıntı kaldı, ancak toplantı haftanın geri kalanında devam etti, taslaklar üretildi ve değerlendirildi ve sonraki birkaç ay boyunca iyi bir metin üretmek için daha sonraki editoryal çalışmaların yapılmasıyla birlikte PER-1 gerçeğe dönüştü.

Sonra yine dağıldı!

13.2 İkinci deneme - PER-1

PER-1 oylandığında, PER-2'den çok daha olumlu bir yanıt aldı, ancak yine de ABD'den "Maalesef, uzun tartışmalardan sonra, PER-1'i onaylamamak zorunda kaldık" diyen çok güçlü bir "reddet" oyu vardı. .

PER-1 ile sürüm 1 sisteminin sürüm 2 sistemiyle birlikte çalışmasının hiçbir yolu yoktur (ikiniz de aynı tür tanımıyla çalışmadığınız sürece bir kodlamanın sonunu bile bulamazsınız).

Bu şeyler Uluslararası Standartlar için işe yaramayacak. Öldür onu."

Hayır - TLV'ye geri dönmelisiniz. Yalnızca TLV, sürüm 1 ve sürüm 2 sistemleri arasında birlikte çalışma sağlayabilir. Denenmiş ve gerçek bir tekniktir. Son cümle doğru ama ikincisi doğru mu? Artık öyle olmadığını biliyoruz. 1992'de daha az emindik!

Bu toplantı öncekinden daha az travmatikti, ancak bu "birlikte çalışma" (veya bilindiği şekliyle "genişletilebilirlik" sorunu), son gerçek PER'nin üretimini on iki aydan biraz fazla geciktirdi.

13.3 Ve sonunda real-PER elde ederiz

İnsanlara bir protokolün 1. versiyonu ile 2. versiyonu arasında ne tür eklemeler veya değişiklikler yapmak isteyeceklerini açıklamaları için kağıt sağlamak üzere birçok ağaç kesildi. Ortaya çıkan fikir birliği, esasen "Sadece sonunda bir şeyler eklememiz gerekiyor."

Üç nokta gösterime (ve onunla birlikte istisna işaretçisine) girer ve uzantı biti PER'ye gider. Oraya vardık!

Üç nokta insanların bunu belirtmesi için sağlandı ve PER'deki uzantı biti kodlama desteği sağladı.

Real-PER yaklaşımı esas olarak şunları söylemektir:

> Spesifikasyonun bölümleri genişletilebilir olarak işaretlenmemişse, bunları verimli bir şekilde kodlayın. tavır.
>
> Parçalar genişletilebilir olarak işaretlenmişse, ancak değerler sürüm 1 spesifikasyonunun değerleriyse (kökte), bunu söylemek için bir bit sağlayın, ancak yine de bunları verimli bir şekilde kodlayın.
>
> Genişletilebilir parçalar kökün dışında değerlere sahipse (sürüm 2 eklemeleri), uzantılar bitini bire ayarlayın ve bir uzunluk sarmalayıcı sağlayın.

Birlikte çalışma endişesi olmadan verimli kodlamalar yapan bir tasarımdan (PER-1) başlamamış olsaydık, bu yaklaşımın geliştirilmesi pek olası değildir. PER yolundaki çeşitli travmalar, sürüm 1'den sürüm 2'ye birlikte çalışma sağlamanın tek yolu olan yerleşik TLV kodlama geleneğini kırmak için muhtemelen gerekliydi.

Bu hikayenin tam olarak sonu değil! Daha sonra dizilerin ve setlerin ortasına bir şeyler ekleyebilmek için güçlü bir baskı vardı ve versiyon parantezleri eklendi.

Ayrıca, hava trafik kontrol çalışanlarından dolgu parçalarından kurtulma ve PER'nin HİZALANMAMIŞ sürümünü üreten sekizli hizalamayı unutma baskısı vardı.

Ama bunlar küçük sorunlardı. PER-1'den son PER'e giden yol, bizi her zaman olması gerektiği kadar kesin olmayan bir metinle baş başa bıraktı ve özellikle genişletilebilirlik ve uzantılar bit kavramının PER-1 metnine entegrasyonu bugün hala bazı sorunlar yaratıyor (1999), genişletilebilirlik gösteriminin belirsiz kullanımlarıyla ilgili argümanlarla (ve muhtemelen sonunda düzeltmelerle) (neyse ki kimse henüz yazmadı ve belki de asla yazmayacak!). Bu sorunların birçoğu, kitaplarımızı yazmaya başladığımızda Olivier ve ben tarafından ortaya çıkarıldı! Neyse ki ikimiz de cevabın ne olması gerektiği konusunda hemfikirdik ve bence kitaplarımız da aynı hikayeyi anlatıyor!

14 DER ve CER

(Tanıdık geliyor mu? Evet, o kutuyu daha önce kullandım - üzgünüm!)

> Standartlaştırıcıların kalplerine kazınmış: İşiniz Standartlar üretmek. Kabul edemiyorsanız, isteğe bağlı yapın veya daha da iyisi başka bir Standart yapın. Sonuçta, bir Standart iyiyse, birçok standart daha iyi olmalıdır!

Bir BER kodlamasındaki ana "seçenek", yapılandırılmış kodlamalar için belirli veya belirsiz uzunlukların kullanılmasıdır. Hangisinin en iyi olduğu konusunda hiçbir zaman anlaşma olmadı ve BER spesifikasyonunda her ikisine de izin veriliyor. Yıllar boyunca, bazı profil oluşturma gruplarının bir formu veya diğerini zorunlu kılmaya çalıştığı her türlü tartışma yaşandı.

Kabaca söylemek gerekirse, kısa mesajlar için kesin uzunluklu biçim muhtemelen en mantıklı olanıdır, ancak uzun mesajlar için belirsiz biçim tercih edilmelidir. Seçeneği bir uygulayıcıya bırakmak iyi bir fikir gibi görünüyor, ancak elbette bu, kod çözücülerin her iki biçimi de ele alması gerektiği anlamına geliyor.

Bununla birlikte, kodlayıcı için hiçbir seçenek içermeyen kodlama kuralları istiyorsanız (Kısım III Bölüm 1, madde 10'da tartışıldığı gibi test sorununu en aza indirmek ve güvenlikle ilgili sorunlara yardımcı olmak için), o zaman mermiyi ısırmanız gerekir!

X.500 ilk olarak (yaklaşık yirmi satırlık bir belirtim olarak) BER'in kanonik bir kodlamasını üretme kurallarını üretti ve buna "seçkin" bir kodlama adını verdiler. Uygulamak istedikleri türleri kapsayacak kadar iş gördü, ancak tamamlanmadı. Ayrıca (tartışmalı olarak) bazı seçimleri optimal bir şekilde yapmadı.

ASN.1 grubu, adını X.500'den alarak "Ayırt Edici Kodlama Kuralları" olarak adlandırmaya karar verdiği BER'in kanonik versiyonu için bir standart üretmeye karar verdi.

ASN.1 spesifikasyonu ile X.500 spesifikasyonu arasındaki en büyük fark, X.500'ün zorunlu olarak belirli uzunluklu kodlamalar kullanması ve ASN.1 grubunun mümkün olan her yerde belirsiz uzunluk kullanmasıydı!

Bu arada, gönderilecek çok büyük mesajları olan ancak aynı zamanda kanonik kodlamalara ihtiyaç duyan başka bir standart olan ODA (Office Document Architecture) çalışanları, ASN.1 grup taslağını beğendi!

Böylece nihai sonuç etkili bir şekilde iki ayrı standart oldu, biri DER için (ilk X.500 metniyle tamamen uyumlu ve belirli uzunluktaki kodlamalar kullanılarak) ve biri CER için (© OS'de "geliştirme", 31 Mayıs 1999)

orijinal X.500 çalışması ve mümkün olduğunda belirsiz uzunluktaki kodlamaların kullanılması). Her iki "standart" da elbette BER ile birlikte X.690'da (ISO/IEC 8825-1) yayınlanmıştır.

DER'in X.500 kullanımı esas olarak sertifikalar içindir ve artık e-ticaretin geliştirilmesinde yoğun bir şekilde kullanılmaktadır. (Çoğu e-ticaret etkinliği, DER kodlaması kullanan X.509 sertifikalarına dayalıdır.)
Buna karşılık, RKY çalışması geniş çapta uygulanmadı. Dolayısıyla, göreli teknik değerleri ne olursa olsun, DER, BER'in kanonik kodlamaları için fiili standart haline geldi ve CER muhtemelen öldü!

15 Semantik model ve hepsi - 1990'ların sonlarında ASN.1

Sözdizimsel olarak izin verilen bazı ASN.1 yapılarının yasallığı hakkında her zaman sorular olmuştur, ancak gerçekten izin verilen bir şey olabilir veya olmayabilir. Bu sorunların ana alanı, bir değer referansı ile onun düzenleyicisi arasındaki "tip eşleştirme" kurallarıdır. Örneğin, ile:

> İnsanlar sadece basit ve bariz ASN.1 yazar. Ancak aptal aptal bilgisayarlar, sözdiziminin izin verdiği en anlaşılmaz ifadelerin yasallığını bilmek ister.
>
> Ve bilgisayarların araç satıcılarında önemli bir yeri var! Dinlenmeleri gerekiyor!

```
intval TAM SAYI ::= 7
```

Bir dizinin öğesi olarak yasal olarak yazıp yazamayacağınızı sorabilirsiniz:

```
[27] INTEGER DEFAULT aralığı
```

veya

```
INTEGER (0..127) DEFAULT intval
```

Elbette bunların yasal olmasını beklersiniz, değil mi? Ancak "[27] INTEGER" ve "INTEGER (0..27)" kesinlikle "INTEGER" ile tam olarak aynı tür değildir. Üç tür de tam olarak aynı değerleri içermez ve ortak değerlerinin kodlaması, BER ve PER'de veya her ikisinde de farklılık gösterir.

Yine, bir değer referansı belirli (oldukça karmaşık) bir tip tanımı kullanılarak tanımlanırsa ve bu değer referansı, aynı (ancak metinsel olarak farklı) bir tip referansı tarafından yönetildiğinde kullanılırsa, bu yasal mıdır? Ve ikinci metin oluşumu birincisiyle tamamen aynı değilse, metin ASN.1 yasa dışı hale gelmeden önce ne kadar sapma gösterebilir?

Bu örneklere uzantı işaretleyicisinin kullanımını ekleyin ....

Bunlar, 1990'ların sonlarında uğraşılan ve muhtemelen değerleri içeren kovalar olarak türlerin modellerinin (resimlerinin) standardına ve metinsel olarak tanımlanan türler arasındaki "değer eşlemelerinin" dahil edilmesine yol açacak olan sorunlardır. ayrı notasyon parçaları.
Üç nokta ve/veya uzantıları olan türleri kapsamak için benzer modeller/resimler gereklidir.

Tüm bu çalışmadaki yol gösterici ilke, herhangi bir anlam ifade ettikleri takdirde (yalnızca en bariz şekilde doğru olan şeyleri yasal hale getiren katı bir spesifikasyondan ziyade) şeyleri yasal hale getirmek, ancak sonunda neyin yasal olduğuna dair çok eksiksiz bir spesifikasyon elde etmektir ASN .1.

Elbette okuyucu, bu iş için baskının araç satıcılarından geldiğini tahmin edecektir. Aklı başında hiçbir protokol belirleyicisinin asla yazmayacağı şeylerin yasallığı veya başka türlüsü hakkında yargıya varmak için gerekli olan kodu yazmaları gerekir!

16 Ne kaçtı?

ASN.1 geliştirmesinin mevcut standarda girmeyen birkaç özelliği olmuştur. Diriltilebilirler, ama muhtemelen olmayacaklar!

ASN.1 daha da iyi olabilir mi? Tartışılan başka iyileştirmeler de kesinlikle var. Ancak eklenen karmaşıklık, kazanımlara değer mi? Konsensüs "HAYIR" dır.

Hafif Ağırlık Kodlama Kuralları (LWER), Kısım III Bölüm 4'te tam olarak tartışılmıştır ve burada tekrar değinilmeyecektir.

Muhtemelen en büyük kayıp, KARAKTER DİZİSİ DİZİSİ için verimli bir kodlama sağlamamak ve her sütunun bir dizi olası türün seçimi olabileceği bir tablonun kodlaması içindi.

CHARACTER STRING durumunda (hatırlarsanız, bu türdeki her kodlamada iki nesne tanımlayıcı değeri taşır), orijinal konsept, herhangi bir zincirdeki her kodlamanın aynı değere sahip olduğu CHARACTER STRING türündeki kodlama zincirlerine izin vermekti. nesne tanımlayıcı değerleri. Bu değerler, her zincirin başlangıcında iletilecek ve daha sonra, ağ protokolündeki sanal devreler gibi, her kodlamayı kendi zincirine bağlamak için kısaltılmış bir tanımlama olacaktır. Ne yazık ki, bu zincirleme kavramında (uzantılarla etkileşim nedeniyle) ciddi hatalar bulundu ve ilk yayınlanmasından sonraki günler içinde çok hızlı bir şekilde geri çekildi.

O zamanlar, başka bir özellik olan "çalışma zamanı parametreleri"nin ("dinamik kısıtlamalar" olarak da adlandırılır, çünkü çalışma zamanı parametreleri yalnızca kısıtlamalarda kullanılabilir) aynı verimlilik gereksinimini destekleyebileceği düşünülüyordu, ancak çalışma zamanı parametreleri (dinamik kısıtlamalar) sonunda terk edildi.

Yaklaşım, herhangi bir yapısal sorun nedeniyle değil, yalnızca pazar yerinin (ASN.1 kullanıcıları) bunu gerçekten talep ediyor gibi görünmemesi ve ASN.1'e oldukça karmaşık bir özellik eklemenin zahmete değmemesi nedeniyle terk edildi.

Bu çalışma zamanı parametreleri nelerdi? Buradaki fikir, bir türün parametreleştirilmiş bir tür olabileceği, ancak gerçek parametrelerin, tür referans alındığında belirtilmek yerine bir iletişim örneğinde iletileceğiydi. Bu, bir SEQUENCE OF için ortak olan herhangi bir bilginin (örneğin, Sequence Of CHARACTER STRING'in nesne tanımlayıcıları veya bir tablonun her sütunu için türlerin tanımlanması), dizinin her bir öğesi yerine yalnızca bir kez iletilmesini sağlar. DİZİSİ.

Bir diğer terkedilmiş özellik ise "global parametreler"di. Parametreleştirilmiş bir türünüz varsa, parametrelerin soyut sözdizimi tanımından birçok tür tanımı düzeyinden geçerek sonunda kullanıldıkları noktaya kadar iletilmesi oldukça yaygın bir durumdur.

Genel parametreler çalışması, soyut sözdiziminin bir parametresinden kullanılacağı noktaya esas olarak doğrudan bir yol sağlayarak, netliği artırmayı ve belirtimlerin ayrıntılarını azaltmayı amaçlıyordu.

Bu fikirlerden bazılarını beğendiyseniz, standardizasyon oyununa katılın ve onları geri getirip getiremeyeceğinize bakın! Standardizasyon oyununa girmek istemiyorsanız, ASN.1'in olduğu gibi harika olduğunu kabul edin ve bu bölümü bitirelim!

BÖLÜM SONU

Bölüm 2
ASN.1 Uygulamaları

(Veya: ASN.1 kodlamaları yapan bir yazılım mı kullanıyorsunuz?)

Özet:

Bu bölüm:

ASN.1'in kullanıldığı uygulama alanlarının bir göstergesini sağlamaya çalışır.

Seçtikleri belirtim dili olarak ASN.1'i kullanmış olan bazı kuruluşları belirlemeye çalışır.

Uygulamaların ve kuruluşların tartışılması için kısmi bir tarihsel çerçeve kullanır.

1. Giriş

Bu kısa bölüm, ASN.1'in uygulandığı bazı alanların ana hatlarını vermektedir. Hiçbir şekilde kapsamlı olduğunu iddia etmez ve bazı gruplar kendilerinden bahsedilmediği için rahatsız olurlarsa özür dilerim!

Dahil edilenden daha fazla dışarıda kaldınız - Üzgünüm!

Aynı şekilde, ASN.1 tanımlarını içereceklerini söyleyen Web sayfaları da gördüm, ancak bu belirli uygulama için ASN.1 kullanımının terk edildiğine güvendiğim kişiler tarafından güvence verildi! Bundan sonraki kısımlarda umarım çok fazla hata yoktur ama eminim ki ciddi eksiklikler vardır.

Vurgu farklı uygulamalar üzerinde olsa da, ele alma kısmen tarihseldir ve ASN.1 kullanımının tek bir uygulamadan (X.400) günümüzde geniş bir uygulama yelpazesine kademeli olarak genişletildiğini göstermektedir.

Böylece bu bölüm, önceki tarihsel bölümü tamamlar.

Bu bölüm, ISO Standart numaralarının ve ITU-T Tavsiyelerinin ve İnternet RFC'lerinin ayrıntılı bir listesini içermez , bunun yerine uygulama alanlarının geniş bir taslağını verir ve örnek olarak ara sıra gerçek bir belirtimden söz eder.

İlgilenen herkes için, ASN.1'i kullanan spesifikasyonlara ilişkin daha eksiksiz bir ayrıntılı referans seti, Ek 5'teki URL aracılığıyla veya Olivier Dubuisson'un ek metninde (Ek 5'te de atıfta bulunulmaktadır) bulunabilir.

Bu bölümdeki kısaltmaların çoğu, Web arama motorlarına girdi olarak kullanılabilir ve genellikle ilgili kuruluşların veya spesifikasyonların ana sayfalarında isabetle sonuçlanır. bu en iyi yol

Ek 5 işinize yaramazsa daha fazla bilgi almak için! (Web URL'lerinin değişme alışkanlığı vardır!)

ITU-T ve ETSI ve ECMA için size özellikleri hakkında çok daha fazla bilgi verecek Web siteleri (Ek 5 veya bir arama yoluyla erişim) ve ITU-T söz konusu olduğunda ASN kullanan Önerilerin bir listesi de vardır. 1. (ITU-T Önerilerinden herhangi biri ilginizi çekerse, dikkatli olun - bunların tümü çevrimiçi olarak satın alınabilir ve teslim edilebilir, ancak bu size ciddi paraya mal olur!)

Bu bölüm kaçınılmaz olarak pek çok kısaltma içerir - her protokolün ve her organizasyonun kendi kısaltması vardır. Daha önceki metinde kullanılmadıysa kısaltmayı hecelemeye çalışırım, ancak bazen çabaya pek değmez çünkü kısaltma genellikle tam başlıktan çok daha iyi bilinir!

Birçok durumda, bir arama yoluyla bulduğunuz bir belgenin tam adını vermeden kısaltmayı kullandığını göreceksiniz. Pek çok insan bu kısaltmaları biliyor, ancak size tam adı vermek için çok düşünmek zorunda kalacak ve muhtemelen o zaman yanlış anlayacaktır! (Bazı durumlarda, farklı Web ve diğer belgeler aynı kısaltmalar için farklı tam adlar verir - ancak aynı şeyi tanımlamayı amaçladıkları açıktır!)

Yani elimizden gelenin en iyisini yapıyoruz. Ancak bir meydan okuma istiyorsanız, aşağıdaki kısaltmalar hakkında neler bulabileceğinize bakın (ASN.1 bağlamında). Belirli bir sırayla verilmezler. Bazılarından bu bölümde bahsediliyor, çoğundan bahsedilmiyor. Hepsinin, ASN.1'i belirtim dili olarak kullanan protokoller veya kuruluşlarla ilgili olduğuna inanılmaktadır. Aşağıdaki konularda kendinizi test edin:

SET, SNMP, TCAP, CMIP, PKCS, MHS, ACSE, CSTA, NSDP, DPA, TDP, ETSI, DMH, ICAO, IMTC, DAVIC, DSS1, PKIX, IIF, LSM, MHEG, NSP, ROS(E), FTAM , JTMP, VT, RPI, RR, SCAI, TME, WMtp, GDMO, SMTP.

%100 alamıyorsanız (tabii ki bazıları yanlış yazıyor olabilir!), bir ağ gurusu değilsiniz ve ağ konularındaki tavsiyeleriniz için saat başına $$£££$$ ücret alamazsınız!

Avrupa ve ABD arasında gidip geliyorsanız ve her iki toplumda da aktifseniz, Atlantik göletinin yalnızca bir tarafında çalışanlardan daha fazla zorlukla başa çıkma şansınız daha yüksektir. Elbette ASN.1 araç sağlayıcıları, tüm bu kısaltmaların ne anlama geldiğini KESİNLİKLE biliyorlar, çünkü araçlarını onları desteklemek için satıyorlar. Ama söyleyecekler mi?

Pekala, biraz araştırmadan sonra yukarıdaki listenin yaklaşık %95'ini kapsayabileceğimi dürüstçe kabul ediyorum (bu bölümde %95'ten çok daha azını anlattım), ama kesinlikle hepsini değil!

Eğer herhangi bir okuyucu konuyu anlatabilirse (ve tercihen daha fazla bilgi için bir URL verebilirse), o zaman Ek 5'teki bağlantı yoluyla adresime bir e-posta memnuniyetle karşılanacaktır - ancak bu kitap için çok geç, belki de ikinci baskı?

2 X.400'deki kökenler

X.400, orijinal olarak X.400 ila X.430'u kapsayan (boşluklarla birlikte) ilgili bir CCITT Tavsiyeleri setiydi. X.400 belirtimlerinin dünya için (OSI) fiili e-posta sistemi olması amaçlandı.

Her şeyin bir başlangıcı vardır!

X.400, İnternet posta protokolüne göre pek çok avantajla başladı (o zamanlar Basit Posta Aktarım Protokolü'ydü (SMTP), gösterişsizdi - Çok Amaçlı İnternet Posta Uzantıları (MIME) gibi gösterişler daha sonra eklendi).

X.400, en başından beri, çeşitli farklı "vücut parçası" türlerini destekleyerek, postaya multimedya eklerinin gönderilmesine izin verdi ve 1998 sürümünde, Askeri İleti İşleme Sistemleri (MMHS) spesifikasyonlarının (güvenlik özellikleri) neredeyse tüm güvenlik özelliklerini bir araya getirdi. SMTP'de hala çok daha fakirler).

Ancak SMTP, isteğe bağlı eklerin aktarımını sağlamak için MIME uzantılarıyla geliştirildi (her ne kadar X.400'ün yaklaşık iki katı bant genişliğinde olsa da) ve günümüzde İnternet posta uygulamaları genellikle kendi alanlarının dışından gelen postaları kabul etmeyerek (ama ortadan kaldırmaz) maskeli balo risklerini. (Bu çalışmaların hiçbiri ASN.1 tabanlı değildir.) Ancak teknik özellikleri ne olursa olsun, SMTP tabanlı e-postanın artık dünyanın fiili standardı olduğunu biliyoruz, ancak X.400 hala aralarındaki ağ geçitlerinde rol oynuyor. farklı posta sistemlerinde ve askeri iletişimde ve diğer azınlık takipçileri var.

ASN.1 orijinal olarak yalnızca bu X.400 belirtimini desteklemek için üretilmiştir ve elbette devam eden tüm X.400 çalışmalarında hala kullanılmaktadır.

Orijinal olarak yalnızca X.400'ü desteklemek için üretilmiş olan bir diğer önemli özellik, başlangıçta yalnızca "ROS" olarak adlandırılan Uzaktan İşlemler Hizmet Öğesi (ROSE) belirtimiydi. ASN.1 gibi, bu da daha genel bir fayda olarak kabul edildi ve X.200 Öneriler serisine taşındı. (ROSE, Bölüm II Bölüm 6'da daha ayrıntılı olarak ele alınmıştır). ROSE tamamen ASN.1 tabanlıydı (ve öyledir) ve telekomünikasyon alanındaki pek çok uygulamanın temelidir. Gereksinimleri, Bilgi Nesnesi konseptinin geliştirilmesinde ve "delikleri" ele alma ihtiyacının tanınmasında çok etkili olmuştur. (ASN.1'in geçmişi ile ilgili önceki bölüme bakın.)

3 Açık Sistemler Ara Bağlantısı (OSI) ve ISO'ya geçiş

OSI aracılığıyla dünyayı ele geçirmek için hızlı genişleme - güya! Ancak diğe bazı ISO Teknik Komitelerinde taraf alınmıştır.

1980'lerin başlarında, konferanslardaki makalelerin "OSI'ye karşı SNA" (SNA, IBM'in "Sistem Ağ Mimarisi" idi) gibi başlıkları olacaktı ve çoğu insan OSI çalışmasının sonunda dünya çapında ağ iletişimi için fiili standart olacağına inanıyordu. SNA'yı devirmek için bir savaş verirdi. Yine, tarihsel olarak, OSI bir bütün olarak asla başaramadı, ancak ASN.1'i tek uygulamalı bir dil olmaktan çıkarıp birçok protokol belirtici tarafından kullanılan bir araç haline getiren, ASN.1'in ana akım OSI'ye girişiydi.

CCITT'den (o zamanlar olduğu gibi) ISO'ya dahil edilmesinden çok kısa bir süre sonra ASN.1, OSI'nin Uygulama Katmanı ve OSI ile ilgili diğer birçok standart için spesifikasyonlar üreten her bir grup tarafından tercih edilen spesifikasyon dili olarak benimsendi. Bu standartların çoğunun uygulamaları bugün hala kullanılmaktadır, ancak çoğu durumda azınlıkta olduklarını söylemek doğru olur.
kullanmak.

ASN.1'in OSI uygulamalarının çoğu, ISO/JTC1/SC16 tarafından geliştirilen ve daha sonra (yeniden düzenlemenin ardından) ISO/JTC1/SC21 tarafından geliştirilen OSI'nin sözde "Uygulama Katmanı"ndaki standartlar içindi. Bunlar, diğerlerinin yanı sıra, uzaktan veritabanı erişimi, işlem işleme, dosya aktarımı, sanal terminaller vb. için standartları kapsıyordu.

Özet ve aktarım sözdiziminin ayrılmasına ilişkin ASN.1 kavramları, OSI yığını üzerinde çalışan ve herhangi bir özet için kullanılacak aktarım sözdizimini müzakere etmek için Sunum Katmanını kullanan protokoller için OSI'nin sözde "Sunum Katmanı" ile çok iyi uyum sağladı. sözdizimi.

Bununla birlikte, ilginç bir şekilde, ASN.1 aynı zamanda Sunum Katmanı protokolünün kendisini tanımlamak için de kullanıldı - muhtemelen ASN.1'in OSI Sunum Katmanı üzerinde çalışmayan bir protokol için ilk kullanımı (birçoğu takip edecekti).

Hatta OSI Oturum Katmanının (Sunum Katmanının altındaki katman) ASN.1 kullanılarak nasıl tanımlanabileceğini (daha net ve makine tarafından okunabilir bir biçimde) gösteren bir taslak bile dağıtılmıştı . Buna, BER'de küçük bir değişiklik olan ve ASN.1 tanımına uygulandığında, Oturum Protokolü Standardının şu anda belirttiği satırdaki bitleri tam olarak üretecek olan bir "Oturum Katmanı-BER" taslağı eşlik etti. Ancak Oturum Katmanı belirtimleri o zamana kadar eksiksiz ve kararlıydı, bu nedenle taslak hiçbir zaman ilerletilmedi.

Benzer bir durum, Yönetilen Nesnelerin Genel Tanımı (GDMO) ile ortaya çıktı - Japonya'dan dağıtılan bir taslakta Bilgi Nesnesi Sınıfları ve "SİZDİZİMİ İLE" kullanan eşdeğer bir notasyonun tanımlandığı, ancak GDMO nedeniyle asla ilerleme kaydedilmediği aşağıdaki 8. Maddeye bakın. iş o zamana kadar istikrarlı ve oldukça olgundu.

ASN.1, bankacılık, güvenlik, otomatik üretim hatlarının kontrolü için protokoller gibi alanlarda ve en son olarak "akıllı karayolları" için ulaşım alanındaki protokollerin geliştirilmesinde diğer birçok ISO Teknik Komitelerinde kullanılmıştır. Bu protokoller genellikle (genellikle) OSI yığını üzerinde taşınmazlar ve OSI çalışmasındaki ilk köklerine rağmen ASN.1'in OSI'den bağımsızlığını göstermeye hizmet etmişlerdir.

Bu tür bir kullanıma yeni bir örnek, bir Entegre Devre kredi kartı ile kart kabul eden cihaz arasında geçen mesajların (ISO/TC68 ile) tanımlanması içindir.

4 Protokol testi topluluğu içinde kullanın

Protokol spesifikasyonlarının yanı sıra OSI dünyası, protokol uygulamalarının standartlaştırılmış testleri fikrini başlattı. Bu test dizileri, elbette, kendi başına protokollerdir; burada bir test sistemi, test edilen bir uygulamaya mesajlar gönderir ve aldığı yanıtları değerlendirir. Ağaç ve Tablo Kombine Notasyonu (TTCN) bu amaç için en sık kullanılan notasyondur ve ASN.1, veri yapılarının tanımlanması için bu notasyona gömülüdür.

Sadece standartlaştırılmış bir protokol değil, standartlaştırılmış uygulama testleri de istiyorsunuz!

TTCN uygulamasıyla yakından ilgili olan, ASN.1'in başka bir ITU-T resmi tanımlama tekniği olan Sistem Açıklama Dili (SDL) içinde kullanılmasıdır.

Avrupa Telekomünikasyon Standartları Enstitüsü (ETSI), bu gösterimleri kullanarak test spesifikasyonlarının geliştirilmesinde önemli bir aktör olmuştur.

5 Entegre Hizmetler Dijital Ağı (ISDN) içinde kullanım

80'lerde, Entegre Hizmetler Dijital Ağı (ISDN) en çok konuşulan konuydu. Telefon ağının sayısallaştırılmasından doğdu.

> Muhtemelen ASN.1'in ana OSI çalışması dışındaki ilk uygulaması.

Gelişmiş ülkelerin çoğunda telefon şebekesi, evler ile yerel telefon santrali arasındaki ve çoğu durumda analog kalan sözde "yerel döngü" dışında artık tamamen dijitaldir.

ISDN, evler ve yerel bir telefon santrali arasındaki mevcut yerel döngüleri kullanarak, her biri bir telefon araması veya 64 Kbps veri bağlantısı taşıyabilen iki sözde "B-kanalı" ve bir "D-kanalı" (sinyalleşme için kullanılır) sağladı. abone ile santral arasında). ISDN, telefon aboneleri tarafından geniş çapta erişilebilir hale geldi, ancak ana uygulaması, telefon ağı üzerinden video konferans için 128 Kbps veri kanalı sağlamak için iki B kanalının birlikte kullanılmasıydı (ve bugün hala - 1999).

ISDN içinde, birçok sözde "tamamlayıcı hizmet" (örneğin, Meşgul Aboneye Geri Arama) D-kanalı kullanılarak uygulandı ve bu hizmetler için protokolü tanımlamak üzere ASN.1 (BER kodlamalı) seçildi.

6 ITU-T ve multimedya standartlarında kullanım

ASN.1, elbette, ITU-T'ye ilk olarak X.400 ve OSI aracılığıyla tanıtıldı, ancak ITU-T (daha sonra CCITT) içindeki diğer birçok standardizasyon grubu tarafından hızla benimsendi.

> ASN.1'in ITU-T'nin birçok yerinde yaygın kullanımı bugüne kadar devam etmektedir.

ASN.1'in ITU-T içindeki kullanımları şurada bulunabilir:

- Konuşma kodlama ve sessizlik sıkıştırma için G serisi tavsiyeleri.
- Düşük bit hızlı iletişim için hareketli video kodlaması ve Etkileşimli Multimedya Telekonferans Konsorsiyumu (IMTC) tarafından uygulanan spesifikasyonlar dahil multimedya (görsel-işitsel) iletişim için H serisi.
- ATM'de test yönetimi için M serisi.
- ISDN ve Akıllı Ağlar (IN) ile ilgili bir dizi özellik için Q serisi.
- Grup 3 faks ve MHEG iletişimleri için T serisi.
- Görsel-işitsel terminal iletişimi için V serisi.
- SDL (yukarıda açıklanmıştır) ve GDMO içinde (Madde'de açıklanmıştır) kullanım için Z serisi 8 aşağıda).
- Ve tabii ki, OSI çalışmasından kaynaklanan Tavsiyeler için X serisinde.

H-serisi ile ilgili olarak, bu Önerilerin belki de en önemlisi, İnternet üzerinden ses, video ve veri iletişimi (video konferans, etkileşimli alışveriş, ağ oyunu ve diğer birçok multimedya uygulaması dahil -) için H.323 serisidir. daha fazla ayrıntı için H.323 Web sitesine bakın). H.320 serisindeki diğer spesifikasyonlar, hem dar bant hem de geniş bant (ATM) ISDN ve PSTN iletişimleri üzerinden multimedya iletişimini ele alır.

Bu Öneriler, çok çeşitli ağ altyapıları üzerinde çalışacak çoklu ortam iletişimi için fiili standartlar haline gelecek gibi görünüyor.

Bilinen birçok ürünün içlerinde gömülü ASN.1 (bu durumda PER) kodlayıcılara sahip olmasına neden olan bu Önerilerdir, dolayısıyla bu ürünlerden herhangi birini kullanırsanız , ASN.1 (kodlamalar) kullanıyorsunuz demektir! Bu tür ürünlere örnek olarak Microsoft NetMeeting, Intel VideoPhone, PictureTel yazılımı vb. verilebilir.

7 Avrupa ve Amerika standardizasyon gruplarında kullanın

Gerçekten uluslararası aktörler olan birçok alt uluslararası (bir tabir türetecek olursak) grup ASN.1'i kullanmıştır.

ASN.1'in oldukça yoğun bir şekilde kullanıldığı üç Avrupa standardizasyon grubu vardır (şüphesiz başkaları da vardır). İlk ikisi başlıklarında "Avrupalı" adını taşır, ancak hepsi dünya çapındaki topluluğa standartlara katkıda bulunur. Bunlar, Avrupa Bilgisayar Üreticileri Derneği (ECMA), Avrupa Telekomünikasyon Standartları Enstitüsü (ETSI) ve daha yakın tarihli Dijital Görsel-İşitsel Konsey'dir (DAVIC). (DAVIC Avrupa merkezlidir, ancak haklı olarak dünya çapında bir konsorsiyum olduğunu iddia eder.)

ECMA, OSI'ye giriş için OSI ile ilgili standartlar üzerinde uzun süredir çalışmaktadır (ancak daha geniş alanlarda da - örneğin, ilk IEEE 802 Standardına önemli girdileri olmuştur). Ayrıca, telefon anahtarları ve son kullanıcı bilgisayarları arasındaki iletişim için ASN.1 tabanlı Bilgisayar Destekli Telekomünikasyon Uygulamaları (CSTA) spesifikasyonunu da üretmiştir. CSTA'nın ilk kurulumu, 1990'ların sonlarında iletişimde önemli bir gelişme olan büyük Çağrı Merkezlerini desteklemek için olmuştur. ECMA spesifikasyonlarında normal olduğu gibi, çalışma uluslararası standardizasyon için ISO'ya girildi.

ETSI, öncelikle ITU-T Önerilerinin Avrupa varyantları ve ITU-T'ye girdi için telekomünikasyon spesifikasyonlarının geliştirilmesi ile ilgilidir. Ayrıca TTCN'ye (içinde gömülü ASN.1 bulunan) dayalı spesifikasyonların geliştirilmesinde de aktif olmuştur. Telekomünikasyon standartları konusunda ECMA ve ETSI ile ITU-T arasında yakın bir ilişki vardır.

DAVIC, video konferansı destekleyen 25 ülkeden 157 şirket ve devlet kurumundan oluşan bir konsorsiyumdur. Spesifikasyonları, uluslararası standardizasyon için ISO'ya girdi niteliğindedir.

Ayrıca ABD'de spesifikasyonlarında ASN.1'i kullanan bir dizi standart grubu ve konsorsiyum vardır. Sıklıkla, ancak her zaman değil, bu tür çalışmalar uluslararası standartlaştırmayı besler.

Bahsetmeye değer (ancak bu liste çok eksik ve biraz rastgele - duyduklarım bunlar):

ANSI X9 komiteleri, ISO/TC68'i besleyen Finansal Sektör Standardizasyonu (örneğin, Fon Transferi ve EDI) ile ilgilidir.

Kimyasal bilgi ve DNA dizilerinin değiş tokuşu için Amerikan Kimya Derneği (Web sitesi için, Ek 5 aracılığıyla Ulusal Biyolojik Bilgi Merkezi'ne (NCBI) giden bağlantılara bakın).

Güvenlik konularıyla ilgili birçok Federal Bilgi İşleme Standardı (FIPS), örneğin, Bilgi Aktarımı için Standart Güvenlik Etiketleri hakkında FIPS PUB 188 - Standart Güvenlik Etiketi bir ASN.1 türü olarak tanımlanır: "NAMEDTagSet"in "NamedTagSet" olduğu "SET OF NamedTagSet" .... vb.

SET konsorsiyumu (aşağıdaki 9. Maddeye bakınız).

8 Bilgisayar kontrollü sistemleri yönetmek için kullanın

Bir bilgisayar tarafından kontrol ediliyorsa, muhtemelen ASN.1 tabanlı değişimleri kullanarak uzaktan yönetirsiniz.

OSI çalışmasından bir başka önemli "buluş", "yönetilen nesneler" kavramıydı (uzaktan iletişim yoluyla sorgulanan, test edilen, yapılandırılan, sıfırlanan vb. cihazlar). Bu, özniteliklere (diğer ASN.1 tarafından tanımlanan ASN.1 türleri olan) sahip bu tür nesnelerin (ASN.1 nesne tanımlayıcıları tarafından tanımlanan) bir modelini üreten Ortak Yönetim Bilgi Hizmetleri/Protokolü (CMIS/CMIP) üzerindeki çalışmadan çıktı. nesne tanımlayıcıları). "Yönetim" esas olarak yönetilen nesnelerin yüzeyinde bulunan ve nesnenin harici görünürlüğünü ve kontrolünü sağlayan bu "özniteliklerden" (CMIP kullanılarak) okunarak veya bunlara yazılarak gerçekleştirildi.

CMIP standardı ilk yayınlandığında, "deliklerle" dolu bir protokoldü - tek bir yönetilen nesne değil ve nitelikleri o aşamada tanımlanmıştı! İnsanların yönetilen nesneleri (tercihen makine tarafından okunabilir bir şekilde) tanımlamasına izin vermek için açıkça bir gösterime ihtiyaç vardı. Bu gösterimi tanımlamak için bir ASN.1 makrosu kullanılmış olabilir, ancak o zamana kadar yeni makroların yazılmasına yönelik bir ambargo vardı ve Bilgi Nesnesi Sınıfının yerini alacak çalışma henüz başlangıç aşamasındaydı. Bu nedenle, Yönetilen Nesnelerin Genel Tanımı (GDMO), GDMO içinde yerleşik bir gösterim olarak ASN.1 ile yönetilen nesneler hakkında gerekli ayrıntıları belirtmek için bir gösterim olarak (İngilizce olarak) tanımlandı.

İnternet dünyasında CMIS/CMIP kavramları benimsenmiş ve CMIS/CMIP'in geliştirilmesine yönelik çalışmalar devam ederken Basit Ağ Yönetim Protokolü (SNMP) için bir RFC üretilmiştir. Başlangıçta bunun CMIS/CMIP olgunlaşana kadar geçici bir çözüm olduğu belirtildi, ancak çoğu geçici çözüm gibi, oldukça kalıcı hale geldi ve bugün uzak cihazların yönetiminde CMIS/CMIP'den daha büyük bir pazar payına sahip.

CMIS/CMIP gibi, SNMP de ASN.1'i kullanır, ancak çok kısaltılmış bir biçimde ve yönetilen nesnelerde ayarlanacak veya okunacak değerleri tanımlamak için kullanılabilecek ASN.1 türlerinin biçiminde önemli kısıtlamalar vardır. Ancak bu, ASN.1'in İnternet standardizasyon topluluğuna ilk gerçek girişini temsil ediyordu.

CMIS/CMIP başlangıçta ağ anahtarlarındaki ve uzak ana bilgisayarlardaki OSI yığınının uygulamalarını kontrol etmek için tasarlanmıştır, ancak (SNMP gibi) günümüzde bilgisayar kontrollü her şeyi uzaktan yönetmek için giderek daha fazla kullanılmaktadır. Dolayısıyla yönetim protokollerinin uygulamaları, teleskopların veya radar çanaklarının yönlendirilmesini ve hatta çamaşır makinelerinin veya fırınların açılıp kapanmasını içerebilir! (Ama ikincisinin henüz bir gerçeklik olduğundan emin değilim.)

9 PKCS ve PKIX ve SET ve diğer güvenlikle ilgili protokollerde kullanım

Başlıktaki kısaltmaları aradan çıkaralım! PKCS, Açık Anahtar Şifreleme Standartlarıdır, PKIX, Açık Anahtar Altyapısıdır (X.509) ve SET, Güvenli Elektronik İşlemlerdir (bunlarla ilgili biraz daha ayrıntılı bilgi aşağıda verilmiştir).

X.509 (ASN.1 tabanlı) sertifikalarının geniş çapta benimsenmesi, ASN.1'i güvenlik işlerinde baskın belirtim tekniği haline getirdi.

X.500, hala önemli desteği olan OSI Standartlarından biridir ve OSI çalışmasında ASN.1'i kullanması, ASN.1'in neredeyse tüm güvenlikle ilgili protokollerde benimsenmesine yol açmıştır.

X.500, bir ISO ve ITU-T Standardı ve Önerisiydi (ve öyledir), ancak X.500'ün işlevsel bir alt kümesi olan Hafif Dizin Erişim Protokolü (LDAP), bir İnternet RFC'sidir ve hızla geliştirilmektedir. Dizin hizmetlerine erişim için facto standardı, X.500'ü dünya çapında bir Dizin hizmeti sağlamak üzere yerel LDAP sunucularını bağlamak için "perde arkasında" kullanıma uygun hale getirir.
LDAP, mesajlarını tanımlamak için ASN.1 gösterimini kullanır, ancak kullandığı ASN.1'in (sınırlı) alt kümesinin değerleri için bir metin kodlaması belirtir (İnternet belirleyicileri arasında metin tabanlı protokoller için tercihler hakkında daha sonra Madde 10'daki tartışmaya bakın).

X.500 öncelikle dünya çapında bir Dizin hizmeti sağlamak için tasarlanırken, dünya çapında bir arama ile çok çeşitli bilgilerin aranmasına olanak tanırken, aynı zamanda sertifikalar için ilk standardı (X.509) sağladı (bunlar - ve elbette bir ASN.1 tipidir).

Temel sertifika kavramı, bir Sertifika Yetkilisinin (CA) bir başvuru sahibine bir genel ve özel anahtar çifti (genellikle bazı ticari ücretler karşılığında) sağlaması ve ayrıca genel anahtar kullanılarak şifrelenmiş bir elektronik bit modeli (bir sertifika) sağlamasıdır. CA'nın anahtarı. Sertifika, başvuru sahibine verilen genel anahtar ile başvuru sahibinin bazı özellikleri (isim, şirket sicil numarası, vb.) arasında bir ilişki sağlayan bir ASN.1 türüdür.

CA'nın kendi özel anahtarını güvende tutması koşuluyla sertifikalar taklit edilemez. Bununla birlikte, CA'nın genel anahtarını bilen (kesin olarak) herhangi biri, verdiği sertifikaların şifresini çözebilir ve dolayısıyla sertifikanın içerdiği kuruluşun veya kişinin genel anahtarına "inanabilir" - ve dolayısıyla bir dereceye kadar "güven" uygulayabilir. o kuruluşa veya kişiye (ve bu genel anahtarı kullanarak geçerli karma değerler üretmek için şifresini çözen mesajlara veya imzalara). Tabii ki, CA'nın genel anahtarı genellikle "daha yüksek" bir CA tarafından verilen başka bir sertifikadan alınır ve bu sertifikanın açık anahtarı .... tarafından verilen başka bir sertifikadan alınır ve bu böyle devam eder, ta ki, ... peki, ... Netscape ortak anahtarı genellikle Web tarayıcı yazılımınızda yerleşiktir! (Elbette güvenilir bir kaynaktan aldınız!).

Başka bir sertifikanın kilidini açan bir genel anahtarı almak için başka bir sertifikanın kilidini açmak üzere bir sertifikadan bir ortak anahtar alma işlemine sertifika zincirleme denir ve başlangıçta insanlar tüm dünyada genel anahtarlarıyla yalnızca bir veya iki üst düzey CA bekliyordu. gerçekten halka açık - belki de gazetelerde günlük olarak ilan edilir!

Ancak daha sonra hemen hemen her ulusal hükümet, ajanslarından birinin üst düzey CA olmasını istediğine karar verdi ve birçok şirket de dahili kullanım için kendi CA'ları olmaya karar verdi. Ve aniden ortak anahtarların ve güven derecelerinin dağıtımı sorunu çok daha karmaşık hale geldi.

PKIX, Açık Anahtar Altyapısı (X.509) anlamına gelir ve CA'ların nasıl çalışması gerektiğini belirten bir dizi İnternet RFC'si ve Taslak RFC'dir. Örneğin, PKIX 4, tüm uyumlu CA'ların halka sunması gereken bir Sertifikasyon Politikası Beyanı (CPS) biçimini belirtir. CPS, örneğin, (bir sertifika vermeden önce) CA'nın bireysel isimleri şu şekilde doğrulaması gerektiğini söyler:

bir pasaportun veya gerçek bir pasaportun veya bir doğum belgesinin fotokopisini gerektirmesi veya (Birleşik Krallık'taki bir şirket için) Şirketler Evi'ne kayıtlı Kayıtlı Ofisin var olup olmadığını kontrol etmesi veya ... Fikri anladınız . Verdikleri sertifika, içerdiği genel anahtar ile bir kişi veya şirket hakkında bazı ek bilgiler arasında bir ilişki olduğunu iddia eder.
Bu iddiaya ne kadar güvenebilirsin? CPS, bunu belirlemenize yardımcı olur.

PKIX'in birçok parçası ASN.1'i tamamen ve doğrudan kullanır.

PKCS, Açık Anahtar Şifreleme Standartları anlamına gelir. Bunlar, RSA Data Security konsorsiyumu ve Microsoft, Apple, Lotus, Sun, Novell ve MIT dahil olmak üzere başlıca lisans sahipleri tarafından üretilen standartlardır. PKCS, veri yapılarını ve bunların kodlamasını tanımlamak için notasyonu olarak ASN.1'i kullanır.

Güvenlikle ilgili bir diğer önemli protokol, MasterCard, Visa ve bilgisayar ve bankacılık endüstrilerinin diğer bölümlerinden oluşan bir konsorsiyum tarafından üretilen Güvenli Elektronik İşlemlerdir (SET). SET, elektronik ticareti tamamen güvenli bir şekilde desteklemek için tasarlanmıştır ve bu nedenle X.509 sertifikalarını kullanır ve kendisi yaklaşık 60 sayfalık ASN.1'dir (daha birçok destekleyici metin sayfasıyla birlikte).

SET sertifikaları akıllı kartlarda depolandığında (akıllı kartlarda bulunan sınırlı bellek nedeniyle) PER kodlaması, sıkıştırılmış sertifika adı verilen bir ASN.1 veri türüyle birlikte kullanılacaktır .

Genel olarak, ASN.1'in X.509'da kullanılması, güvenlikle ilgili protokollerin çoğunun ASN.1'i kullanmasına yol açmıştır.

10 Diğer İnternet spesifikasyonlarında kullanım

PKCS ve PKIX ve SNMP'yi zaten tartıştık.
ASN.1'in (PER ile) HTTP'nin en son sürümünde kullanılması düşünüldü, ancak bunun yerine "pseudo-C" adı verilen ASN.1 benzeri bir notasyon icat edildi.

Evet, burada bile ASN.1'in bazı kullanımlarını görüyoruz!

Genel olarak, İnternet belirticileri, protokol belirtimlerini olabildiğince basit tutmaya ve uygulayıcıların özel araçlar olmadan veya yalnızca kamu malı olan araçları kullanarak çalışmasını kolaylaştırmaya çalışır.

Bu, sonunda basit ASCII metin satırları olan (genellikle BNF kullanılarak tanımlanır) veya ASN.1 kullanılıyorsa, ASN.1 notasyonunun bir alt kümesini kullanan protokollere yol açar .

Web, İnternet'in büyük ölçüde bir parçasıdır, ancak World-Wide Web Consortium'un (W3C) artık kendine ait bir hayatı vardır.

XML Kodlama Kuralları'nın (XER) tanımı aracılığıyla XML ve ASN.1'i birleştirme çalışmaları W3C forumu içinde devam etmektedir. Bu çalışma yenidir ve Kısım III Bölüm 3'te de bahsedilmiştir.

11 Büyük kurumsal işletmelerde ve acentelerde kullanım

Bazı hanehalkı şirketlerinin ve ulusal ve uluslararası ajansların, kendi şirket ve ajansları içindeki iletişim faaliyetlerini desteklemek için ASN.1'i ve onun kodlama kurallarını kullandığı (ve halen kullanmakta olduğu) bilinmektedir.

Üzgünüm - Sana bunları anlatamam!

Ancak, bu kitapta yayınlanmak üzere daha fazla ayrıntı elde etme girişimleri, başvuruların ticari gizliliğiyle ilgili endişeler nedeniyle neredeyse evrensel bir ret ile karşılaştı. Bu nedenle, üzülerek, ticari bir kuruluşun ASN.1 kullanımıyla ilgili bilgiler Web'de görünmediği sürece herhangi bir özel adından bahsetmemeye karar verdim.

Ancak bir kurumdan bahsedeceğim ve bu da Uluslararası Sivil Havacılık Örgütü'dür (ICAO).

Paket Kodlama Kurallarını benimseyen (ve geliştirilmesine yardımcı olan) ilk kuruluş olduğu için ICAO'dan bahsetmeye değer. PER kodlamaları, gerçek ASN.1 spesifikasyonları nihai olarak onaylanmadan çok önce ICAO spesifikasyonlarında tanımlanmıştır ve ASN.1 ve PER'nin kullanımı, Havacılık Telekomünikasyon Ağı (ATN) için temeldir.

12 Sonuç

ASN.1, yalnızca bir uygulama (X.400) için destek sağladığı günlerden bu yana çok yol kat etti.

Dünya çapında bir felaket hikayesiyle bitireceğiz!

Artık protokollerin tüm ana belirticileri tarafından önemli ölçüde kullanılmaktadır ve bazı durumlarda (ancak hepsinde değil) baskın belirtim dilidir. Genellikle notasyonun kullanımı, birkaç istisna dışında ASN.1 tanımlı kodlamaların kullanımıyla ilişkilendirilir.

Sihirli bir değnek sallar ve ASN.1 tanımlı değerlerin kodlamaları olan tüm mesajları dünyadan ortadan kaldırırsanız, felaket kesinlikle Y2K'nin (yıl) olası etkileri için en karamsarların tanımladığı ölçeğin çok ötesinde bir ölçekte gerçekleşir. 2000) bilgisayar hataları. (Ya da bu kitabı 2000 sonrası okuyorsanız gerçekte olanlardan herhangi biri!)

Uçaklar çarpışır, cep telefonları çalışmaz hale gelir, neredeyse tüm telekomünikasyon ve ağ anahtarları yönetilemez ve sürdürülemez hale gelir ve yavaş yavaş ölür, elektrik dağıtım sistemleri çalışmayı durdurur ve biz sihirli değneğimizi sallamadan önce biraz daha ileriye bakmak, akıllı kart tabanlı elektronik işlemler tamamlanamayabilir ve çamaşır makineniz çalışmayabilir! Ama en kötüsü, yeni nişanlınızla yapacağınız NetMeeting bir anda çöker ve hayatınız bir ızdıraba dönüşür!

Bu kitabı bu mutlu notla bitireceğiz!

EKLER

1 Wineco protokol senaryosu

Bu kitaptaki örneklerin çoğu, "Wineco protokolünün" geliştirilmesine dayanmaktadır. Bu, yalnızca ASN.1'in çeşitli bölümlerini göstermek için kullanılan hayali bir protokoldür. İlk bölümleri Kısım 1 Bölüm 2'nin Şekil 13'ünde yer almaktadır ve son protokolün tam bir kopyası aşağıdaki Ek 2'de verilmiştir.

Wineco, çeşitli satış noktalarından şarap satan ve biri kuzeyde diğeri güneyde olmak üzere iki deposu olan bir şirkettir. Başlangıçta tüm satış noktaları yalnızca Birleşik Krallık'taydı (bir satış yerinin adının ASCII karakter seti tarafından desteklenebildiği), ancak daha sonra Wineco, daha büyük bir karakter setinin gerekli olduğu denizaşırı bölgelere yayıldı.

Şekil 13'te, protokolde kullandığımız mesajlardan biri olan "Sipariş-stok" mesajını, belirli bir aciliyete sahip belirli şarap türlerinin bir dizi vakasını talep etmek için görüyoruz. Ayrıca bir "Şube kimliği" türü biçimini de görüyoruz.

Bölüm 1 Bölüm 3'te, gerekli modül başlıklarını ve sonunda olmayan bir ekleme noktası olan bazı genişletilebilirlik işaretlerini ekliyoruz. Daha sonra, bir modülde "ortak tipler", diğerinde üst seviye tip ve üçüncüsünde sipariş protokolü mesajı "Stok için sipariş" ile çok modüllü bir spesifikasyona dönüştürüyoruz. Ayrıca Şekil 21'de, son dönemde yapılan satışların raporunu sağlayan "Satışların Getirisi" adlı ikinci bir üst düzey mesajı kullanıma sunduk.

Bölüm 1, Bölüm 4'te, "Satışların dönüşü" mesajını makul bir şekilde, ancak gerçekten yalnızca kalan ASN.1 temel veri türlerini göstermek için doldurduk! İstisna belirteçleri ve istisna işleme bu Bölümde tanıtılmaktadır. C için Ek 3'te ve Java için Ek 4'te verilen "Return-of-sales" ve "Rapor kalemi" türü, bir ASN.1-derleyici aracından çıktının gösterimi için ana örnek olarak kullanılmıştır.

Bölüm I Bölüm 4'te (Şekil 23) ASN.1 değer gösterimini göstermek için "Satışların geri dönüşü" de kullanılmıştır.

Örneğimizin bir sonraki kullanımı, Bölüm II, Bölüm 3'te, hem sipariş hem de satışların iadesi için "tam sınıf" protokolümüzün katı bir alt kümesi olarak bir "temel sınıf" protokolü tanımlamaya karar verdiğimiz zamandır. Dijital nakit çağına girerken üçüncü bir üst düzey mesajı da buraya ekledik! Geliştirilmiş bir protokol kullanana kadar elektroniğimizin içeriğini yüklüyoruz.

Son ana genişletme, karar verdiğimiz zamandır (Kısım II Bölüm 6'da, tanımlanmış dört işlemle birlikte Uzaktan İşlemler metaforunun kullanımına geçmek için. Bu, iki modüle daha yol açar - biri Uzaktan İşlemler PDU'yu tanımlamak için (gerçek dünyada Remote Operations Service (ROSE) Standardından içe aktarılmış olacaktı ve biri Wineco işlem Bilgi Nesnelerini tanımlayacaktı.

2 Wineco için tam protokol

Bu ek, Wineco protokolünün belirtiminin son sürümünü sözdizimsel olarak doğru ve eksiksiz bir biçimde verir.

```
Wineco-ortak-üst düzey
        {joint-iso-itu-t uluslararası kuruluş(23) seti(42)
            set-vendors(9) wineco(43) modüller(2) top(0)}
    TANIMLAR
        OTOMATİK ETİKETLER ::=
    BAŞLAMAK

    İHRACAT ;
    İTHALATLAR Stoka göre sipariş
        Wineco-ordering-protocol {wineco-OID modülleri(2)
        siparişi(1)}
            DAN satış iadesi
        Wineco-dönüş protokolü {wineco-OID modülleri(2)
        dönüşü(2)};

    wineco-OID NESNE TANIMLAYICI ::=
        {joint-iso-itu-t uluslararası kuruluş(23) seti(42)
            set-vendors(9) wineco(43)}

    wineco-soyut-sözdizimi ÖZET-SYNTAX ::=
                {Wineco-protocol TANIMLAYAN
                                    {wineco-OID özet sözdizimi(1)}
                                        HAS PROPERTY {geçersiz
                                        kodlamaları işler}
                                    --Madde 45.6'ya bakın --          }

    Wineco protokolü ::= SEÇİM {sipariş [UYGULAMA 1] Stok
        siparişi, satış [UYGULAMA 2] Satış iadesi, ... ! PrintableString : "45.7 maddesine bakın" }



SON
--Yayınlanan özelliklerde yeni sayfa.
Wineco sipariş protokolü
  {joint-iso-itu-t uluslararası organizasyon(23) set(42) set-vendors(9) wineco(43)modules(2) sipariş(1)}

    TANIMLAR
        OTOMATİK ETİKETLER ::=
    BAŞLAMAK

    İHRACAT Stoka göre sipariş;

    İTHALATLAR OutletType, Address, Security-Type FROM
                    Wineco-ortak türleri {wineco-OID
                modülleri(2) ortak (3)};

    wineco-OID NESNE TANIMLAYICI ::=
                {joint-iso-itu-t uluslararası kuruluş(23) seti(42)
                    set-vendors(9) wineco(43)}

    Stok siparişi ::= SIRALI {sipariş no
                                INTEGER, ad-
        adres BranchIdentification, ayrıntılar
                                DİZİSİ
```

```
                                        SEKANS
                                {öğe                NESNE TANIMLAYICI,
                                  vakalar           TAM SAYI},
                        aciliyet                    NUMARALI
                                {yarın(0), üç günlük(1),
                                  hafta(2)} VARSAYILAN
                                  hafta, doğrulayıcı Güvenlik-Türü}


    BranchIdentification ::= AYARLA
            {unique-id NESNE TANIMLAYICI, ayrıntılar
                                SEÇİM {uk
                        [0] DİZİ {isim
                                                    GörünürDize,
                                Adresi yazın}, denizaşırı[1] SEQUENCE {ad
                                                    UTF8Dizesi, tür
                                                    OutletType, yer Adresi},

                        depo [2] SEÇİM {kuzey [0] BOŞ,

                                güney [1] NULL} }                   }

        END --
Yayınlanan özelliklerde yeni sayfa.
    Wineco iade protokolü {joint-iso-itu-t uluslararası
     organizasyon(23) set(42) set-vendors(9) wineco(43) modüller(2) iade(2)}

      TANIMLAR
            OTOMATİK ETİKETLER ::=
      BAŞLAMAK

      İHRACAT Satışların geri dönüşü;

      İTHALATLAR OutletType, Address, Security-Type FROM
                        Wineco-ortak türleri {wineco-OID
                modülleri(2) ortak (3)};

      wineco-OID NESNE TANIMLAYICI ::=
            {joint-iso-itu-t uluslararası organizasyon(23) set(42) set-vendors(9) wineco(43)}


      Satış getirisi ::= DİZİ {versiyonu
                                BİT DİZİSİ
                        {sürüm1 (0), sürüm2 (1)} VARSAYILAN {sürüm1},
          bildirilen gün sayısı INTEGER {hafta(7), ay (28), maksimum (56)}
            (1..56) VARSAYILAN hafta, rapor tarihi ve saati SEÇİMİ {iki haneli- yıl UTCZamanı,


                        dört haneli yıllık GeneralizedTime},
                -- Sistem saati dört basamaklı bir yıl sağlıyorsa, -- ikinci alternatif kullanılacaktır. -- ilk alternatif ile zaman, --
                kayan bir pencere olarak yorumlanacaktır. Gecikme nedeni SAYILANDIRILMIŞ {bilgisayar arızası, ağ arızası,
                diğer} İSTEĞE BAĞLI, -- Bu alanı yalnızca ve yalnızca -- rapor edilen gün sayısı yediyi geçiyorsa dahil edin.




          ek bilgi İSTEĞE BAĞLI PrintableString DİZİSİ,
                -- Bu alanı, ancak ve ancak -- gecikme nedeni "diğer" ise dahil edin.

          satış verileri SET OF Rapor öğesi, ... ! PrintableString : "Wineco
           kılavuzunun 15. bölümüne bakın" }
```

```
Report-item ::= SEQUENCE {item NESNE
 TANIMLAYICISI, item-description ObjectDescriptor İSTEĞE BAĞLI, -- Yeni
  stoklanan herhangi bir ürüne eklenecek.

  barkod verileri                    OCTET DİZGİSİ, --
        Wineco kılavuzunun 29. bölümünde belirtilen -- öğenin barkodunu temsil eder. stok tükendi
        BOOLEAN DEFAULT FALSE,

        -- Öğe için stok, rapor edilen dönem boyunca herhangi bir zamanda tükenirse DOĞRU gönderin.
        minimum stok seviyesi GERÇEK, maksimum stok seviyesi GERÇEK, ortalama stok seviyesi GERÇEK



     -- Dönem boyunca minimum, maksimum ve ortalama seviyeleri normal hedef stok seviyesinin yüzdesi olarak
     verin-- }

  wineco-items NESNE TANIMLAYICISI ::= {joint-iso-itu-t International-
   organization(23) set(42) set-vendors(9) wineco(43)stock-items (0)}

SON

Wineco ortak türleri
          {joint-iso-itu-t uluslararasıRA(23) seti(42)
              set-vendors(9) wineco(43) modüller(2) ortak(3)}
   TANIMLAR
          OTOMATİK ETİKETLER ::=
   BAŞLAMAK

   İHRACAT ÇıkışTürü, Adresi, Güvenlik-Tipi;
-- İTHALAT EDEN Güvenlik Tipi
--              SET modülü
--               {joint-iso-itu-t uluslararasıRA(23) set(42) modülü(6) 0};
--İçe aktarmaya gerek kalmaması için bu ek kaldırıldı, --ve aşağıdaki türle değiştirildi.


Güvenlik Tipi ::= SIRALI{
              algoritma NESNE TANIMLAYICISI, kodlama OCTET
              STRING}

--OutletType ana metinde doldurulmamış--
   OutletType ::= SEQUENCE {type
          SAYILANDIRILMIŞ{mail-order, perakende}, açıklama KARAKTER DİZİSİ}


--Ana metinde adres doldurulmamış--
   Adres ::= SEQUENCE {ad UTF8String,
          kasaba UTF8String, ülke
            UTF8String}


   SON
```

Wineco protokolü için C desteği için 3 derleyici çıktısı

Bu ek, Wineco protokolümüzde "Return-of-sales" ve "Report-item"in C uygulamasına destek sağlamak için "OSS ASN.1 Tools" ürünü tarafından üretilen metni içerir. (Bu metnin bir kısmı yalnızca wineco için oluşturulmuştur, bir kısmı bir içerme dosyasından elde edilen genel tanımlardır, örneğin "GeneralizedTime" ve "ossBoolean"):

```
typedef yapı {kısa
                    yıl;            /* GeneralizedTime için kullanıldığında YYYY formatı */
                                    /* UTCTime için kullanıldığında YY formatı */
   kısa kısa        ay; gün;
                    saat;
   kısa             dakika;
   kısa             ikinci; kısa
   kısa             milisaniye;
   dikkatli; /* UTC +/- dakika farkı kısa
                                                                        */
   anlamına gelir;                      /* DOĞRU, UTC zamanı ossBoolean utc     */
} Genelleştirilmiş Zaman;

typedef GeneralizedTime UTCTime;

typedef yapı Nesne Kimliği {
     imzasız kısa uzunluk; işaretsiz karakter
     *değeri;
} Nesne Kimliği;

typedef yapı Rapor_öğesi { imzasız karakter
     bit_mask; # ran_out_of_stock_present 0x80'i
tanımlayın
     Nesne Kimliği öğesi; *Ürün
     işaretsiz karakter *değeri; } açıklamasi_datamesBoolean ran_out_of_stock; /* ran_out_of_stock_present *uzunluk;
     bit_mask içinde ayarlanmadı, değerin YANLIŞ olduğunu ima eder */ min_stock_level; max_stock_level;
          ortalama_stok_seviyesi;

     çift
     çift kişilik

} Rapor_öğesi;

typedef struct İade_of_sales { unsigned char bit_mask;
     tanımla tanımla tanımla tanımla imzasız karakter
#         sürümünü tanımla;   version_present 0x80
#                             no_of_days_reported_on_present 0x40 Reason_for_delay_present
#                             0x20 ek_bilgi_mevcut 0x10
#

                                   /* version_present bit_mask içinde ayarlanmadı *
                                     değerin { version1 } olduğunu ima eder */
```

```
#               sürüm1 tanımla 0x80 sürüm2 tanımla
#               0x40 imzasız kısa no* of_days_reported_on; /
        no_of_days_reported_on_present ayarlanmadı * bit_mask'ta
                                        değerin hafta olduğunu ima eder */ hafta 7 ay 28 maksimum 56

#               tanımla
#               tanımla
#               tanımla
        struct
                { işaretsiz kısa seçim;
#                       two_digit_year_chosen 1'i tanımlayın four_digit_year_chosen
#                       2'yi tanımlayın birleşim {

                        UTCZamanı               iki_hane_yıl; /* seçmek
                                        için, seçimi * two_digit_year_chosen olarak
                                          ayarla */
                        GeneralizedTime four_digit_year; /* seçmek için,
                                        seçimi şuna ayarla:
                                          * four_digit_year_chosen */
                } u;
        } time_and_date_of_report; enum
        { bilgisayar_hatası = 0, ağ_hatası = 1, diğer
                = 2


        } Reason_for_delay;
                                        /* isteğe bağlı; bit_mask'ta ayarlayın *
                                          Reason_for_delay_present * Varsa */

        yapı _seqof1 { yapı
                _seqof1 *sonraki; *değer;
                karakter
        } *Ek Bilgiler; /* isteğe bağlı; bit_mask
                                        içinde ayarlayın * varsa ek_bilgi_varsa */

        yapı _setof1 { yapı _setof1
                *sonraki;
                Report_item değeri; } *satış verisi;

} İade_of_sales;
```

4 Wineco protokolü için Java desteği için derleyici çıktısı

Bu ek, Wineco protokolündeki "Return-of-sales" ve "Report-item" türleri için Java desteği metnini içerir. Bu, Wineco modülleri ile beslendiğinde "OSS ASN.1 Tools" ürünü tarafından üretilen çıktının bir parçasıdır. Bu, Ek 3'ten biraz daha hantal - bu bir şey ifade ediyor mu? Hay aksi - KÖTÜ BİLDİRİM - kimse Java'yı eleştiriyor gibi görünemez! Bu Figure 999'dan daha fazlası!. Java kodu daha hantaldır çünkü alanları ayarlamak ve okumak ve SEQUENCE OF içindeki öğeleri eklemek ve silmek için tüm yöntemleri içerir, bu nedenle Ek 3'teki C kodundan daha fazlasını yapar. Java bilmiyorsanız, kesinlikle bileceksiniz bu eki yok saymak istiyorum. Java bilseniz bile , muhtemelen yalnızca birkaç örnek sınıfa ve yönteme bakmak isteyeceksiniz. İşte Java kodu:

```
paket wineco.wineco_returns_protocol; com.oss.asn1.*
dosyasını içe aktarın; com.oss.util.* dosyasını içe aktarın;
wineco'yu içe aktar.*; wineco.wineco_common_types.*
dosyasını içe aktarın;


public class Return_of_sales Sırayı genişletir {

        /**
        Varsayılan kurucu. */ genel İade_of_sales()
        {}


        korumalı ASN1World getASN1World()
                                { wineco.Wineco.cASN1World'ü döndürür; } korumalı int
        getTypeIndex() { dönüş Wineco_returns_protocol.Return_of_sales_PDU; }


        /**
        Bir IAAPI Değer Referansından Oluşturun. */ public
        Return_of_sales(ASN1World dünyası, int dizini)

                        { ASN1Module.getValueReference(dünya, bu, dizin); }

        /**
        Bileşenlerle oluşturun. */ genel Satış
        İadesi( Sürüm sürümü, No_of_days_reported_on
        no_of_days_reported_on,
                Time_and_date_of_report
                time_and_date_of_report, Reason_for_delay Reason_for_delay, Ek_bilgiler
                ek_bilgiler, Sales_data sales_data)



        {
                SetVersion(sürüm);
                SetNo_of_days_reported_on(no_of_days_reported_on);
                RaporunSaatini_ve_tarihi_ayarla(raporun_ve_tarihi_zamanını);
                SetReason_for_delay(reason_for_delay);
                Ek_bilgiyi Ayarla(ek_bilgi);
```

```
        SetSales_data(satış_verileri);
}

/**
Gerekli bileşenlerle oluşturun. */ genel Satış İadesi
(raporun_ve_tarihi_zaman_ve_tarihi_raporun_ve_tarihi,
Satış_verileri_satış_verileri)


{
        RaporunSaatini_ve_tarihi_ayarla(raporun_ve_tarihi_zamanını);
        SetSales_data(satış_verileri);
}

korumalı geçersiz initComponents() {

        mComponents[0] = yeni Sürüm(); mComponents[1] =
        new No_of_days_reported_on(); mComponents[2] = new Time_and_date_of_report();
        mComponents[3] = new Reason_for_delay(); mComponents[4] = yeni Ek_bilgi();
        mComponents[5] = new Sales_data();


}

// Örnek başlatıcı {

        mComponents = new AbstractData[6]; mPresentBits = yeni
        java.util.BitSet(mComponents.length);
}

// "version" alanı için yöntemler public Version
getVersion() {

        dönüş (Sürüm)mComponents[0];

} genel geçersiz SetVersion(Sürüm sürümü) {

        mComponents[0] = versiyon;
        SetComponentPresent(0);

} genel geçersiz SetVersionToDefault() {SetComponentAbsent(0); } genel boolean hasDefaultVersion()
{ dönüş componentIsPresent(0); } genel boolean hasVersion() { dönüş componentIsPresent(0); } genel geçersiz
deleteVersion() { SetComponentAbsent(0); }


genel statik sınıf Sürümü, BitString'i genişletir {
        /**
        Varsayılan kurucu. */ genel Versiyon()
        { süper(yeni bayt[1], 2); }


        /**
        Bir bayt dizisinden ve önemli bitlerden Bit Dizesi oluşturun. @param, bu nesnenin ayarlanacağı bayt
            dizisine değer verir. @param sigBits, önemli bitlerin sayısıdır. */ public Version(bayt[] değer, int
            sigBits) { super(değer, sigBits); }




        korumalı ASN1World getASN1World()
                                { wineco.Wineco.cASN1World'ü döndürür; }

        // Adlandırılmış liste tanımları. genel statik
        nihai int sürüm1 = 0;
```

```
        genel statik nihai int sürüm2 = 1;

} // Sürüm için son sınıf tanımı

// "no_of_days_reported_on" alanı için yöntemler public
No_of_days_reported_on getNo_of_days_reported_on() {

        dönüş (No_of_days_reported_on)mComponents[1];

} genel geçersiz SetNo_of_days_reported_on
            (No_of_days_reported_on no_of_days_reported_on)
{
        mComponents[1] = no_of_days_reported_on;
        SetComponentPresent(1);

} genel geçersiz SetNo_of_days_reported_onToDefault()
                                {SetComponentAbsent(1); }
genel boole hasDefaultNo_of_days_reported_on() { dönüş componentIsPresent(1); }
                                genel boole hasNo_of_days_reported_on() { dönüş
componentIsPresent(1); } genel geçersiz silmeNo_of_days_reported_on()


                                { SetComponentAbsent(1); }

genel statik sınıf No_of_days_reported_on INTEGER'ı genişletiyor {

        /**
        Varsayılan kurucu. */ public
        No_of_days_reported_on() {} public
        No_of_days_reported_on(short value) { super(value);}
        public No_of_days_reported_on(int value) { super(value);} public No_of_days_reported_on(long
        value) { super(value);}


        korumalı ASN1World getASN1World() { dönüş
                        wineco.Wineco.cASN1World; }

        // Adlandırılmış liste tanımları. public static
        final No_of_days_reported_onhafta = new No_of_days_reported_on(7); genel statik
                                        final No_of_days_reported_on ay = new
        No_of_days_reported_on(28); public static final No_of_days_reported_on maximum =
                                        new No_of_days_reported_on(56); özel son statik
        No_of_days_reported_on cNamedNumbers[] =


                                        {hafta, ay, maksimum}; korumalı son
        statik uzun cFirstNumber = 7; korumalı son statik boole cLinearNumbers =
        false; korumalı INTEGER[] getNamedNumbers() { dönüş cNamedNumbers;} korumalı
        boolean hasLinearNumbers() { dönüş cLinearNumbers;} korumalı uzun getFirstNumber() { dönüş
        cFirstNumber;}

} // No_of_days_reported_on için sınıf tanımını sonlandır

// "report_zamanı_tarihi" alanı için yöntemler genel Rapor_ve_tarihi_of_reportu
getTime_and_date_of_report() {

        (Raporun_ve_tarihi_zamanı)mBileşenleri[2] döndürür;

} genel geçersiz SetTime_and_date_of_report
                        (rapor_ve_tarihi_of_rapor saati_ve_tarihi_of_rapor)
{
        mComponents[2] = time_and_date_of_report;
}
```

```
genel statik sınıf Time_and_date_of_report Seçimi genişletir {

        /**
        Varsayılan kurucu. */ herkese açık
        Time_and_date_of_report() {}


        korumalı ASN1World getASN1World() { dönüş
                        wineco.Wineco.cASN1World; }

        /**
        Bir IAAPI Değer Referansından Oluşturun. */ public
        Time_and_date_of_report(ASN1World dünyası, int dizini)

{ASN1Module.getValueReference(dünya, bu, dizin); }

        genel statik final int two_digit_year_chosen = 1; genel statik final int four_digit_year_chosen
        = 2;

        // "two_digit_year" alanı için yöntemler public static
        Time_and_date_of_report
                        createTime_and_date_of_reportWithTwo_digit_year
                                        (UTCTime two_digit_year)
                        {Time_and_date_of_report __nesne = new
                                        Time_and_date_of_report();
                        __object.SetTwo_digit_year(two_digit_year); __nesneyi döndür;


        } genel boole hasTwo_digit_year() {
                dönüş getChosenFlag() == iki_digit_year_chosen;

        } genel geçersiz SetTwo_digit_year (UTCTime two_digit_year) {
                SetChosenValue(two_digit_year);
                SetChosenFlag(two_digit_year_chosen);
        }
        // "four_digit_year" alanı için yöntemler public static
        Time_and_date_of_report
                        createTime_and_date_of_reportWithFour_digit_year
                                        (GeneralizedTime four_digit_year)
                        {Time_and_date_of_report __nesne = new
                                        Time_and_date_of_report();
                        __object.SetFour_digit_year(four_digit_year); __nesneyi döndür;

        }
        genel boole hasFour_digit_year() { dönüş getChosenFlag() ==
                four_digit_year_chosen;

        } genel geçersiz SetFour_digit_year
         (GeneralizedTime four_digit_year) {
                SetChosenValue(four_digit_year);
                SetChosenFlag(four_digit_year_chosen);
        }
        // Belirli bir seçim örneğini yaratma yöntemi korumalı AbstractData
        createInstance(int seçildi) { switch(chosen) { case two_digit_year_chosen: return new
                UTCTime(); case four_digit_year_chosen: yeni GeneralizedTime() döndürür;



                        varsayılan: fırlat
                                        new InternalError("Choice.createInstance()");
                }
        }
 } // Time_and_date_of_report için sınıf tanımını bitir
```

```
// "reason_for_delay" alanı için yöntemler public Reason_for_delay
getReason_for_delay() {

        dönüş (Reason_for_delay)mComponents[3];

} genel geçersiz SetReason_for_delay(Reason_for_delay Reason_for_delay) {

        mComponents[3] = Reason_for_delay;
        SetComponentPresent(3);

} genel boole hasReason_for_delay() { dönüş
                componentIsPresent(3); }
genel geçersiz deleteReason_for_delay()
                { SetComponentAbsent(3); }

genel statik sınıf Reason_for_delay genişletilir Numaralandırılmış {

        /**
        Varsayılan kurucu. */ public
        Reason_for_delay() {süper(cFirstNumber);}
        korumalı Reason_for_delay(uzun değer) { süper(değer);}


        korumalı ASN1World getASN1World() { dönüş
                wineco.Wineco.cASN1World; }

        // Adlandırılmış liste tanımları. genel statik
        final Reason_for_delay computer_failure = new Reason_for_delay(0); genel statik final
                        Reason_for_delay network_failure = new Reason_for_delay(1); public
        static final Reason_for_delay other = new Reason_for_delay(2); private final static Reason_for_delay
                        cNamedNumbers[] = {computer_failure, network_failure, other};




        korumalı son statik uzun cFirstNumber = 0; korumalı son statik boole
        cLinearNumbers = false; korumalı Numaralandırılmış[] getNamedNumbers() { dönüş
        cNamedNumbers;} korumalı boole hasLinearNumbers() { dönüş cLinearNumbers;} korumalı uzun getFirstNumber()
        { dönüş cFirstNumber;}

} // Reason_for_delay için sınıf tanımını sonlandır

// "ek_bilgi" alanı için yöntemler genel Ek_bilgi getAdditional_bilgi() {


        dönüş (Ek_bilgi)mBileşenler[4];

} genel geçersiz SetAdditional_information
        (Ek_bilgi ek_bilgi)
{
        mComponents[4] = ek_bilgi;
        SetComponentPresent(4);

} public boolean hasAdditional_information() { return componentIsPresent(4); }
                genel geçersiz silmeAdditional_information()

                { SetComponentAbsent(4); }

genel statik sınıf Ek_bilgi, SequenceOf'u genişletir {

        /**
        Varsayılan kurucu. */
```

```
        genel Ek_bilgi() {}

        korumalı ASN1World getASN1World() { dönüş
                wineco.Wineco.cASN1World; }

        /**
        Bir IAAPI Değer Referansından Oluşturun. */ genel
        Ek_bilgi(ASN1World dünyası, int dizini)

                {ASN1Module.getValueReference(dünya, bu, dizin); }

        /**
        SEQUENCE OF/SET OF'a bir Eleman ekleyin. */ genel eşitlenmiş
        geçersiz add(PrintableString öğesi) {


            super.addElement(öğe);

        } /**
        SEQUENCE OF/SET OF içinde bir Eleman ayarlayın. */ genel senkronize
        geçersiz küme

                (PrintableString öğesi, int atIndex)
        {
            super.SetElement(element, atIndex);

        } /**
        SEQUENCE OF/SET OF'den bir Eleman alın. */ genel eşitlenmiş
        PrintableString get(int atIndex) {


            dönüş (PrintableString) super.getElement(atIndex);

        } /**
        SEQUENCE OF/SET OF içine bir Eleman ekleyin. */ genel senkronize geçersiz
        ekleme

                (PrintableString öğesi, int atIndex)
        {
            super.insertElement(element, atIndex);

        } /**
        SEQUENCE OF/SET OF dizisinden bir Eleman kaldırın. */ genel eşitlenmiş
        geçersiz kaldır(PrintableString öğesi) {


            super.removeElement(öğe);

        } /**
        Bir SEQUENCE OF/SET OF örneği oluşturun. */ public AbstractData
        createInstance() {


            dönüş ((AbstractData) yeni PrintableString());
        }
    } // Ek_bilgi için sınıf tanımını sonlandır

    // "sales_data" alanı için yöntemler genel Sales_data
    getSales_data() {

        iade (Sales_data)mComponents[5];

    } genel geçersiz SetSales_data(Sales_data sales_data) {
```

```
        mComponents[5] = satış_verileri;
    }

    genel statik sınıf Sales_data, SetOf'u genişletir {

        /**
        Varsayılan kurucu. */ genel Sales_data()
        {}


        korumalı ASN1World getASN1World() { dönüş
                wineco.Wineco.cASN1World; }

        /**
        Bir IAAPI Değer Referansından Oluşturun. */ public
        Sales_data(ASN1World dünyası, int dizini)

                { ASN1Module.getValueReference(dünya, bu, dizin); }

        /**
        SEQUENCE OF/SET OF'a bir Eleman ekleyin. */ genel eşitlenmiş
        geçersiz ekle(Report_item öğesi) {


            super.addElement(öğe);

        } /**
        SEQUENCE OF/SET OF içinde bir Eleman ayarlayın. */ genel eşitlenmiş
        geçersiz küme(Report_item öğesi, int atIndex) {


            super.SetElement(element, atIndex);

        } /**
        SEQUENCE OF/SET OF'den bir Eleman alın. */ genel senkronize
        Report_item get(int atIndex) {


            dönüş (Rapor_öğesi) super.getElement(atIndex);

        } /**
        SEQUENCE OF/SET OF içine bir Eleman ekleyin. */ genel eşitlenmiş geçersiz
        ekleme (Report_item öğesi, int atIndex)


        {
            super.insertElement(element, atIndex);

        } /**
        SEQUENCE OF/SET OF dizisinden bir Eleman kaldırın. */ genel eşitlenmiş
        geçersiz kaldır (Report_item öğesi) {


            super.removeElement(öğe);

        } /**
        Bir SEQUENCE OF/SET OF örneği oluşturun. */ public AbstractData
        createInstance() {


            return ((AbstractData) new Report_item());
        }
    } // Sales_data için son sınıf tanımı
} // İade_of_sales için sınıf tanımını sonlandır
```

```
genel sınıf Report_item Diziyi genişletir {

        /**
        Varsayılan kurucu. */ genel Report_item()
        {}


        korumalı ASN1World getASN1World() { dönüş
                wineco.Wineco.cASN1World; }

        /**
        Bir IAAPI Değer Referansından oluşturun. */ public
        Report_item(ASN1World dünyası, int dizini)

                { ASN1Module.getValueReference(dünya, bu, dizin); }

        /**
        Bileşenlerle oluşturun. */ public
        Report_item( ObjectIdentifier item,
        ObjectDescriptor item_description, OctetString
                bar_code_data, boolean
                ran_out_of_stock, double min_stock_level, double
                max_stock_level, double averaj_stock_level)




        {
                SetItem(öğe);
                SetItem_description(item_description);
                SetBar_code_data(bar_code_data);
                SetRan_out_of_stock(ran_out_of_stock);
                SetMin_stock_level(min_stock_level);
                SetMax_stock_level(max_stock_level);
                Ortalama_stok_seviyesini Ayarla(ortalama_stok_seviyesi);
        }

        /**
        Gerekli bileşenlerle oluşturun. */ public
        Report_item( ObjectIdentifier item, OctetString bar_code_data,
        çift min_stock_level, double max_stock_level, double
                averaj_stock_level)




        {
                SetItem(öğe);
                SetBar_code_data(bar_code_data);
                SetMin_stock_level(min_stock_level);
                SetMax_stock_level(max_stock_level);
                Ortalama_stok_seviyesini Ayarla(ortalama_stok_seviyesi);
        }

        korumalı geçersiz initComponents() {

                mComponents[0] = new ObjectIdentifier(); mComponents[1] = new
                ObjectDescriptor(); mComponents[2] = yeni OctetString();
                mComponents[3] = yeni BOOLEAN(); mComponents[4] = yeni Gerçek();
                mComponents[5] = yeni Gerçek(); mComponents[6] = yeni Gerçek();
```

```
}

// Örnek başlatıcı {

        mComponents = new AbstractData[7]; mPresentBits =
        yeni java.util.BitSet(mComponents.length); mComponents[3] = yeni BOOLEAN();
        mComponents[4] = yeni Gerçek(); mComponents[5] = yeni Gerçek(); mComponents[6] = yeni
        Gerçek();


}

// "item" alanı için yöntemler public
ObjectIdentifier getItem() {

        dönüş (ObjectIdentifier)mComponents[0];

} genel geçersiz SetItem(ObjectIdentifier öğesi) {

        mComponents[0] = öğe;
}

// "item_description" alanı için yöntemler public ObjectDescriptor
getItem_description() {

        dönüş (ObjectDescriptor)mComponents[1];

} genel geçersiz SetItem_description(ObjectDescriptor item_description) {

        mComponents[1] = item_description;
        SetComponentPresent(1);
}
genel boole hasItem_description() { dönüş
                                        componentIsPresent(1); }
genel geçersiz deleteItem_description()
                                        { SetComponentAbsent(1); }

// "bar_code_data" alanı için yöntemler public OctetString
getBar_code_data() {

        dönüş (OctetString)mComponents[2];

} genel geçersiz SetBar_code_data(OctetString bar_code_data) {

        mComponents[2] = bar_code_data;
}

// "ran_out_of_stock" alanı için yöntemler public boolean
getRan_out_of_stock() {

        return ((BOOLEAN) mComponents[3]).booleanValue();

} genel geçersiz SetRan_out_of_stock(boolean ran_out_of_stock) {

        ((BOOLEAN) mComponents[3]).SetValue(ran_out_of_stock); SetComponentPresent(3);

}
genel geçersiz SetRan_out_of_stockToDefault()
                                        {SetComponentAbsent(3); } public
boolean hasDefaultRan_out_of_stock() { dönüş componentIsPresent(3); }

genel boolean hasRan_out_of_stock() { dönüş
                                        componentIsPresent(3); }
genel geçersiz silmeRan_out_of_stock()
```

```
                                        { SetComponentAbsent(3); }

        // "min_stock_level" alanı için yöntemler public double
        getMin_stock_level() {

                return ((Gerçek) mComponents[4]).doubleValue();
        }
        genel geçersiz SetMin_stock_level(double min_stock_level) {

                ((Gerçek) mComponents[4]).SetValue(min_stock_level);
        }

        // "max_stock_level" alanı için yöntemler public double
        getMax_stock_level() {

                return ((Gerçek) mComponents[5]).doubleValue();

        } genel geçersiz SetMax_stock_level(double max_stock_level) {

                ((Gerçek) mComponents[5]).SetValue(max_stock_level);
        }

        // "average_stock_level" alanı için yöntemler public double
        getAverage_stock_level() {

                return ((Gerçek) mComponents[6]).doubleValue();

        } genel geçersiz SetAverage_stock_level(çift ortalama_stok_seviyesi) {

                ((Gerçek) mComponents[6]).SetValue(average_stock_level);
        }
} // Report_item için sınıf tanımını sonlandır
```

Web üzerinden 5 ASN.1 kaynağı

Bu ek, hem diğer Web kaynaklarına hem de bu kitabın uzantılarına bağlantılar içeren bir OSS Nokalva sitesine tek bir bağlantı sağlar. Özellikle şunları içerir:

Bu kitabın okuyucularıyla ilgili diğer yayınlara (hem Web tabanlı hem de basılı kopya) yapılan atıflar.

Bu kitapta kullanılan tüm kısaltmalar da dahil olmak üzere ASN.1 ile ilgili terimler sözlüğü. (Burada kullanılan kısaltmaların çoğu, size hızlı bir göz atma ve belki biraz daha fazla bilgi sağlayacak olan dizinde de yer almaktadır.)

Posta listeleri, Olivier Dubuisson'un kitabıyla birlikte sitesi, Unicode sitesi, "OSI'yi Anlamak" kitabımla kendi sitem, Uluslararası Kayıt sitesi, ITU- gibi diğer web tabanlı ASN.1 kaynaklarının ayrıntıları ve/veya bağlantıları T ve ETSI siteleri, Nesne Tanımlayıcı ağacının bazı bölümleri için tahsisat veren bir site vb.

ASN.1 kullanılarak tanımlanan spesifikasyonların daha fazla detayı ve bu spesifikasyonların kamuya açık olduğu bilinen elektronik versiyonlarına bağlantılar.

Bu kitap için yazım hatası sayfaları üretildikleri şekliyle ve ne zaman.

Bu kitabın elektronik bir kopyası.

OSS Nokalva sitesinin URL'si:

http://www.nokalva.com

Lütfen gelin ve ziyaret edin!

Ve her ihtimale karşı - URL'lerin değişme alışkanlığı vardır - şu adreste bir çapraz bağlantı da sağlanır:

http://www.larmouth.demon.co.uk/books

dizin

{

{...}..... 229

7

7 katmanlı model 25

A

soyut sözdizimi ....25 Soyut
Sözdizimi Gösterimi Bir ..... 16, 328 soyut
sözdizimi..... .....345 ÖZET-
SÖZDİZİMİ .... 49, 73, 76 HERHANGİ
BİR 230
HERHANGİ BİR ŞEKİLDE
TANIMLANAN 231
API 117, 122
uygulama için gerekli modüller..... 77
uygulama için gerekli türler 76 ASN.1
uygulamaları..... .....357 ASN.1
modülü..... .....62 ASN.1
araçları..... .....39, 44 ASN.1-
derleyici-aracı ....111 otomatik
etiketleme..... .....68 OTOMATİK
ETİKETLEME..... .....54 OTOMATİK
ETİKETLER..... 66

B

Bacchus-Naur Formu 42
Temel Kodlama Kurallar..... 30, 236,
BER 252 .... 30, 252 İkili
Tabanlı Spesifikasyon ... .24 BİT
DİZİSİ..... 84
BMPString 153
BOOLE.... 80

C

kanonik 34
etiketlerin kanonik sırası..... 291
CGM..... 35
KARAKTER DİZİSİ..... .....233 karakter
dizisi tipi.....97, 100, 149, 338 değer
gösterimi..... .....155 Karakter
Tabanlı Spesifikasyon..... 24
koleksiyon..... ..... 102, 157
iki nokta üst üste 58
yorum ....58 Ortak
Nesne İstek Aracısı Mimarisi.....32
derleyici..... 111
BİLEŞENLERİ 94
Bilgisayar Grafikleri Meta Dosyası.....35
somut sözdizimi..... 34
TARAFINDAN KISITLANMIŞ..... 220
kısıtlamalar ..... .97, 108 alt tür
kısıtlamaları içeriyordu.....166
ÇORBA..... 32
Kurye..... 43, 325, 334

D

tarih/saat türleri..... 91
VARSAYILAN..... 51
tasarım sorunları..... 123
geliştirme süreci..... .....18 ayırt
edici değer ....80 metni
çoğaltma..... 64

E

EDIFACT..... 41
etkili alfabe kısıtlaması.....290 etkili boyut
kısıtlaması.. .....290 üç
nokta 69, 130, 182, 352
GÖMÜLÜ PDV..... 232
Kodlama..... 33
kodlama kuralları..... ..... 16, 30, 236
SAYILANMIŞTIR..... 82
HATA ....345 istisna
işleme .....131, 139 istisna
belirtimi..... 134
istisnalar..... .....104, 181
AÇIK..... 54
açık etiketleme.... .....68 AÇIK
ETİKETLER 66
İHRACAT. 71
genişletilebilirlik28, 60, 66, 70, 97, 104, 129, 139, 181, 282
İMA EDİLEN GENİŞLETİLEBİLİRLİK.....66
HARİCİ..... 231

G

Genelleştirilmiş Zaman 91
GenelDize 153
vali 56
GrafikDizesi..... 153

H

delikler.....26, 97, 105, 188, 190, 275

IA5Dizesi 152
IDL 32
IETF 15
ZIMNİ.. ....54 örtülü
etiketleme ....67 ZIMNİ
ETİKETLER 66
İTHALAT ....71 bilgi nesnesi
sınıfları..... 107, 209 bilgi nesnesi
kümeleri.....201 dahili alt
tipleme..... .....166 ekleme
noktası ... . 70, 184, 308 .....
ÖRNEĞİ .....276 TAM
SAYI ....80 Arayüz Tanımlama
Dili ..... 32, 191 Uluslararası Standartlar
Organizasyonu ..... 15, 25 Int Ulusal Telekomünikasyon
Birliği
Telekomünikasyon Standartları Sektörü.....15
İnternet Mühendisliği Görev Gücü 15
İnternet Protokolü... .....16, 39 IP23, 39
IPv6 ... .28
ISO 15, 25
ISO646Dize ....151 ITU-
T..... ..... 15, 43, 361

L

katmanlama..... 25
düzen ....57 baştaki
bit .... 85, 266 Hafif Ağırlık
Kodlama Kuralları ....316 hat
izleme..... ..... 38, 140 satır
numarası..... 63
LWER..... 316

M

makine tarafından okunabilir sürüm 64
makrolar..... ..... 97, 106, 345 posta
listesi.... ..... .140 yönetim
sorunları.....123
eşleme 117
ÜYE ....319 Mesaj
Yönetim Sistemleri..... 43, 359 Minimum Bit
Kodlama Kuralları.....319

N

adlandırılmış bitler..... 84, 85, 86, 266
ad.. 57
BOŞ..... 88
NumericString..... 150

Ö

NESNE TANIMLAYICI....49, 89, 97, 100, 143, 334
nesne tanımlayıcısı kodlaması.....270 nesne
tanımlayıcısı ağacı..... 144
ObjectDescriptor..... .....90 SEKTÖRLÜ
DİZİ..... 87
RKY..... ....36 Office
Belge Mimarisi .....36, 353 OID
ağacı ....90 Aman
Tanrım.... ....15 Açık
Yönetim Grubu..... .15 Açık Sistemler Ara
Bağlantısı.....25 açık
tipler..... 134
İŞLEM 345
OSI.. ....25 OSI
katmanlaması ....189 OSS
ASN.1 Araçları..... .. 13, 110, 372, 374

P

Paketlenmiş Kodlama Kuralları..... 30, 243,
278 parametrelendirme..... ..... 108, 134, 205, 221
kişi 325
PER..... ..... 30, 243, 278 PER
görünür kısıtlamalar..... .....284 izin
verilen alfabe kısıtlaması..... 163
YazdırılabilirDize 89, 151
protokol ....22 Protokol
belirtimi ....24 Yayın
stili 62

R

aralık kısıtlaması..... 82, 97, 162, 164
GERÇEK..... .....83, 337 ilişkisel
kısıtlama .....108 BAĞIL
OID ....336 uzaktan
işlemler..... 190
GÜL..... 190

S

kapsam kuralları 81
Güvenli Elektronik İşlemler.....15, 365 seçim tipi
gösterimi ....93 semantik
model..... .....109 noktalı
virgül..... 58
SIRALI..... 95
AYAR..... 15, 95, 147, 365 boyut
kısıtlamaları..... .....164 sürgülü
pencere 91
stil..... 128
altküme ....168 alt
tip..... ..... 97, 102, 159

T

T61Dizesi ....152 tablo
kısıtlaması ....108 etiketleme
ortamı..... 66

etiketler 54, 97, 103, 136, 172
TCP /IP 16, 23
TeletexString..... .....152 zaman farkı..... 92
TLV .... 29, 40, 236, 239 üst düzey tip..... .....49 sondaki bit .... 85, 266 aktarım sözdizimi..... .....30, 34 İletim Kontrol Protokolü ....16 tip ataması..... .....56 TİP-TANIMLAYICI..... 225

sen

Unicode..... 342
EvrenselDize 153
Kullanışlı Tip..... .....90 kullanıcı tanımlı kısıtlama 108, 220
UTCZamanı 91
UTF8Dizesi 154

V

değer atamaları.....56 değer referansı ataması .....56 değişken sözdizimi ....216 sürüm parantezleri..... ..... 97, 104, 181
VideotexString 152
VisibleString 151

W

SÖZDİZİMİ İLE.....216

X

X.400 ....43, 358