VÇTD

(VÇTD Çomar Tasarımı Değildir)

Serdar Köylü, A. Murat Eren, Gürer Özen

24 Kasım 2004

İçindekiler

1 Sorun

Bir sisteme kurulan uygulamaların, birbirleriyle uyumlu çalışabilmeleri için bazı ayarlarının yapılması gereklidir. Bunu otomatik olarak yapabilecek bir sistem olmadığında, kullanıcı kendi yapmak istediği işin dışında bir takım teknik konularda bilgi kazanmak ve uygulamaları tek tek ayarlamak için zaman kaybetmektedir. Bu konudaki nasıl (howto) belgeleri, az sayıda belirli senaryo için yazıldıklarından kullanıcıya yardımcı olamamaktadır.

1.1 Dağıtımların Yaklaşımı

Varolan Linux dağıtımları, bu sorunu örneğin kurulu uygulamalar (menu), fontlar, açılış işlemleri (initscripts) gibi tek tek alt sistemler bazında çözmeye çalışmaktadır.

Genelde, uygulama paketleri, dosya sistemi üzerinde sabit bir dizine, söz konusu alt sisteme neler sağladıklarını kaydetmekte; bu alt sistemi kullanacak uygulamalar ise, buraya önceden belirlenmiş biçimde kaydedilen dosyaları tarayarak, sağlanan hizmetleri bulmaktadır. Uygulamaların entegrasyonu için, ya uygulamalar buradaki standartları bilecek biçimde değiştirilmekte, ya da gerekli çevrimi yapacak üçüncü bir yönetici uygulama araya sokulmaktadır. Kayıt ve çevrim işlemleri için özel veri biçimleri, kabuk, Perl ya da Python betikleri, bazen de bunların bir karışımı kullanılmaktadır.

Bu yöntemde gördüğümüz noksanlıklar:

- Alt sistemler, bir sistem modeli içinde toplu biçimde tasarlanmadıklarından, birbirleriyle ilişkileri eksik kalmakta ve uygulamaları paketleyenlerin dikkatinden kaçabilmektedir. Birçok uygulama, bağımsız tek başına çalışan sistemler olmayıp, bir işletim sistemi içerisinde bir takım görevleri yerine getiren bileşenlerdir. Sistem modeli olmayınca uygulamaların bu görevlerini birbirine ekleyerek tek tek uygulamaların toplamından daha güçlü bir sistem elde etmek mümkün olmamaktadır.
- Kapsayıcı bir sistem modeli olmaması kullanıcı ve sistem profilleri oluşturmayı ve yönetmeyi zorlaştırmaktadır.
- Dosya sistemi üzerinden gidilmesi, bazı servislerin ve alt sistem yönetici uygulamalarının uzak makinalarda çalışabileceği esnek bir tasarıma olanak vermemektedir.
- Kabuk ve Perl/Python betiklerinin paketleyicinin veya alt sistemin tercihine göre karışık olarak kullanılması, paketlerin doğruluğunu otomatik olarak denetleyecek programlar yapmayı ve sorunlu paketlerin incelenip, entegrasyon hatalarının giderilmesini güçleştirmektedir.
- Özellikle kabuk betikleri içerisinde değişikliklerin atomik yapılması ve o an bir nedenden dolayı yapılamayan işlemlerin yapılabilecekleri ana kadar bekletilmesi çok güç olduğundan sık sık alt sistemler hatalı ayarlanmakta ve uygulamalar arasında sorunlar çıkmaktadır.

1.2 İşletim Sistemleri

Windows, OS/2, MacOS X gibi ticari işletim sistemlerinde, sistemin parçaları ve kullanıcının çalışma ortamını oluşturan uygulamalar genellikle tek bir merkezden çıktıkları için, uyum sorunları işletim sisteminin çağrıları (API si) üzerinden çözülmektedir. Ayarları toplu halde tutan merkezi bir kütük; çokluortam, ağ protokolü, donanım yöneticisi gibi parçalar için parçaların yerleşebileceği modül yapıları bulunmaktadır.

Bu yöntemde şu noksanlıkları görüyoruz:

- Uygulamaların ayarlarına merkezi erişim sunulması, tek başına istenen faydayı getirmemektedir. Bir genel model olmadığı için, bu bilgileri kullanmak isteyen kullanıcı yada diğer uygulamaların, bilgiyi sunan uygulama ve ayarları hakkında detaylı bilgiye sahip olması sorunu hala ortadadır.
- Uygulamalar ve yönetim sistemi arasında API düzeyinde bir ilişki, iki grubu iç içe geçirip direkt bağlantı sağlayacağı için, parçaların bağımsızlığını azaltacaktır. Bu da, ayrı ayrı parçaların geliştiricilerinin, adam/ay modelinde bağımsız çalışmak yerine, bir araya gelip karşılıklı iletişim ve senkronizasyon ile çalışmasına, dolayısıyla geliştirme işlerinin ölçeklenebilirliğinin azalmasına yol açmaktadır.
- Parçaların farklı ellerden çıktıkları ve alternatiflerin bol olduğu özgür yazılım modeline uymamaktadır.
- Dağıtımımıza
  girecek uygulamaları yeni API leri kullanacak şekilde değiştirmek, uygulama kodunu çok iyi incelemeyi, yapılan değişikliklerin yeni sorunlara yol açmadığını kapsamlı olarak analiz etmeyi gerektirmektedir. Bu da büyük zaman ve emek harcamasına yol açacaktır.
- Bir alt sistemin yetersiz kaldığı görülüp yeni bir alt sistem yapısı geliştirildiğinde, API değişikliğine yol açmamak için API üzerindeki değer ve çağrılara kapsamları dışında anlamlar ve görevler yüklenmekte, ve API yi öğrenmek ve kullanmak isteyenlerin işi çok zorlaşmaktadır. Ya da API değişikliği yapılmakta, ve varolan uygulamaların yeni API yi taşınması, eski ve üçüncü parti uygulamalar için uyumluluk katmanları hazırlanması gibi fazladan sorunlar çıkmaktadır.
- Her dile API desteği verebilmek için ya CORBA gibi karmaşıklığı arttıracak teknolojiler ya da bakım işlerini yükseltecek çok sayıda “wrapper” hazırlanması gereklidir.

1.3 Uzman Sistemler

Linux’un masaüstü ve iş dünyasında kullanımının artmasıyla, bir takım genel yapılandırma ve yönetim araçları da geliştirilmiştir. YaST, LinuxConf, WebMin gibi bu

araçlar kullanıcıya üst seviye bir arabirim sunup, kullanıcının burada yaptığı seçimleri uygulamaların alt seviye ayarlarına taşımaktadır. İki seviye arasında geçiş yapabilmek için gereken bilgiler araçların içinde bir dizi kural olarak kodlanmıştır.

Bundan başka, bilgisayar ağlarının yaygınlaşmasıyla birlikte, birden fazla bilgisayarın merkezi yönetimini yapabilecek IBM Tivoli, HP OpenView, CIM, SNMP, OSI CMIP gibi ürün ve çerçeveler ortaya çıkmıştır. Ayrıntılarda farkları olmakla birlikte, genel mimarileri, yönetilecek bilgisayarda bulunacak ajanlar, yönetim bilgisayarında bir yönetici yazılım, bunlar arasında bir iletişim protokolü ve yönetilecek görev ve ayarları belirten bilgi modellerinden oluşmaktadır. Yalnızca yapılandırma ile sınırlı kalmayıp, hata bulma, performans ve güvenlik değerlendirmeleri, kullanım hesaplama gibi işleri de yapmaktadırlar.

Bu sistemler de, kapsayıcı genel bir model yerine, tüm olası ayar ve görevleri sunmaya çalışmakta, yönetici yazılımı kullanacak kişinin yönetilecek her birimi ve bunlar arasındaki ilişkileri çok iyi bildiğini varsaymaktadır. Sorun kullanıcıdan alınıp yöneticiye taşındığında, artan miktardan dolayı başedilmez hale gelmektedir. Bu noktada yöneticiye yardımcı olacak uzman sistemler sunulmaktadır.

Her iki grup yazılım da, kolay kullanılır ve detaylarda boğulmamış bir yapılandırma arabirimi sunabilmek için, alt ve üst düzey bilgiler arasındaki ilişkileri tanımlayan kural dizileri kullanmakta ve bu kurallar, işin uzmanları tarafından, bir elden yönetici yazılımın içinde kodlanmaktadır. Yönetici yazılımın bir takım API ler ile bu bilgileri üst seviye, bazen üçüncü parti, uygulamalardan alması, bilginin parçalarda değil, merkezde bulunduğu gerçeğini değiştirmemektedir.

Bu uzman sistem yaklaşımında şu noksanlıkları görmekteyiz:

- Parçaların dinamik olarak değişmesi, eski kurallarda değişikler ve yeni kurallar gerektirmekte, kural dizilerini oluşturmayı zorlaştırmaktadır.
- Çok sayıda ve çeşitli parçalardan oluşan bir sistem için çok sayıda kural gerekmekte; kuralların işletilmesi, güncellenmesi, değiştirilmesi gittikçe büyüyen bir sorun olmaktadır.
- Kurallar birden fazla parçanın detaylı bilgilerini bir arada içerdiği için, kural yazmak çok zorlaşmakta, kuralların hata içermesi, öngörülmeyen durumları ihmal etmesi olasılıkları artmaktadır.

2 Gereksinimler

Uygulamalar arası uyumlu çalışmayı sağlamak için her uygulama,

1. Diğer uygulamaların bilgilerine erişebilmeli,
2. Kendi bilgilerini diğer uygulamalara sunabilmeli,

3. Önceden belirlenmiş görevler içinden neleri yapabildiğini sisteme bildirebilmeli,
4. Kendi görevleri dışındaki işlere karışmamalı, bunları ilgili uygulamalardan istemeli,
5. Bilgileri değiştiğinde, ilgilenen uygulamaların haberdar edilmesini sağlamalıdır.

Diğer çözümlerin noksanlıklarına takılmadan bunları sağlamak için,

1. Uygulamaların çalışmalarını yönetecek bilgiler ve ayarlanması gerektiği düşünülen seçenekler genelde uygulama yazarları tarafından önceden düşünülerek bir takım ayar dosyalarından okunup kullanılacak duruma getirilmiştir. En az emekle entegrasyonu sağlamak ve kapalı kodlu üçüncü parti yazılımların da sisteme konabilmesini olanaklı kılmak için, uygulamaları oldukları gibi kabul edip, bu ayar dosyalarını değiştirmek yoluyla entegre etmek gereklidir.
2. Ayar dosyasından değil de, komut satırından veya benzeri değişik yollarla değiştirilen seçenekler ve türlü ayar dosyası biçimleri (ini, XML, vs) bulunmaktadır. Ayrıca uygulamalar arası uyum düzeyinde bir ayar, alt düzeylerde birden çok ayara karşılık gelebilir. Bu tür implementasyon farklılıklarının enkapsüle edilebildiği nesnel bir yaklaşım gerekmektedir.
3. Bir uygulamanın sistemde yapabileceği görevlerin ve alt düzey ayar işlemlerinin nasıl yapılacağının bilgisi uygulamanın kendisi ile birlikte gelmeli ve uygulamayı en iyi bilen kişi olan paketleyici tarafından hazırlanmalıdır. Bu, nesnel yaklaşımla enkapsüle edilmiş ayar işlemlerinin, her türlü uygulamanın alt düzey ihtiyaçlarına çevrilebilmesini sağlayacaktır.
4. Nesnel yaklaşımda, uygulamaların bilgilerini ve görevlerini sunacakları nesneleri, ve birbirleriyle ilişkilerini gösteren bir model gerekmektedir. Bu nesne modelinin hiyerarşik olması, daha kolay kavramayı ve öğrenmeyi sağlayacaktır.
5. Nesne modelinde, diğer bilgisayarlardan gelen nesnelerin, birer uzak nesne olarak yer alabilmesi ve kullanıcı tarafından aynen yerel nesneler gibi yönetilebilmesi, bir ağ yönetim sistemi oluşturmayı çok basit bir iş haline getirecektir. Hiyerarşik model burda erişim yetki denetimini belli alt sistemler bazında ayarlamayı kolaylaştıracaktır.
6. Aynı görevi yapabilecek birden fazla uygulama olabilir. Duruma göre görevlerin hepsine birden yaptırılması, veya içlerinden birinin seçilmesi gerekebilir. Seçim için uygulamaların yeteneklerinden başka, o anda hangilerinin çalıştığı, hangilerinin çalıştırılabilir olduğu gibi kriterlere de bakılması gerekecektir. Uygulama görevi aldığında ise, çalışmaya başlayabilmesi için kendi içinde birden fazla işlem yapması ve duruma göre nasıl çalışacağına karar vermesi gerekebilir. Bütün bu dinamik bilgileri tanımlayabilmek için, iterasyon, karar verme, karşılaştırma özellikleri olan bir dil gerekecektir.

7. Bu dilin seçimi özgür bırakıldığında, takip edilmesi zor, çok sayıda bağımlılık ortaya çıkmakta; diller arası uyum ve bilgilerin hata giderme işlemleri ise çok güçleşmekte olduğundan, tek bir dil üzerinde karar verilmelidir.

8. Bu dilin önceki şartları sağlamak için, nesne kavramına sahip olması, kontrol edilemeyecek bağımlılık problemleri çıkarabilecek genişleme mekanizmaları taşımaması gereklidir. Transaction ve asenkron çalışma yapılabilmesi için tüm durumunu (state) içinde taşıyabilen bir script dili olması mecburidir.

9. Basit ve öğrenmesi kolay olması, araçlarla işlenebilmesi, IDE ve RAD araçları yardımıyla üretilebilir olması, paketleme ve hata giderme sürecini çok kısaltacaktır. Uzak işlemler, transaction, profil yönetimi ve asenkron çalışma için sentaks kolaylıkları içermesi de olumlu puanlardır.

10. Transaction, uzak nesneler, profil yönetimi, asenkron çalışma, erişim yetkileri gibi özellikleri uygulayabilmek için bu dildeki betikleri yönetip işletecek bir uygulamaya gerek vardır.

Toparlarsak, gereksinimlerimizi karşılayabilecek ve diğer benzer sistemlerdeki sorunlara düşmeyecek bir yapılandırma yönetim çerçevesi için ihtiyacımız olan parçalar, bir nesne modeli (2-5), bir nesne tanımlama dili (6-9), uygulamalar için bu dilde yazılmış betikler (1,3) ve bu betikleri işletecek bir uygulamadır (10).

3 COMAR

Ulusal Dağıtım için geliştirdiğimiz yapılandırma çerçevesinin adı olarak COMAR (COnfiguration MAnageR) seçtik. Bu çerçeve içinde, yukardaki gereksinimlerden çıkardığımız bileşenler ve ayrıntıları şöyle olacak:

3.1 Nesne Modeli

Bir işletim sisteminde, uygulamalar arasında eşgüdümü sağlayabilmeye yönelik olarak, hangi görev ve bilgilerin bulunduğunu tanımlar. Sistem açılışı, donanım tanıma, ağ yapılandırmasından grafik çalışma ortamı, veri kaynakları, kullanıcı verilerine kadar, bütün bir işletim sistemi haritasındaki yapılandırma ve yönetim sorunlarını kapsar.

3.2 CSL

CSL (COMAR Scripting Language) nesnelerin yazılacağı betik dilidir.

3.3 Betikler

Nesne modelindeki işlemlerin uygulamalara aktarılmasını sağlayan, CSL ile yazılmış nesnelerdir. Her uygulamanın betikleri, uygulamayı dağıtım için paketleyen kişi tarafından hazırlanacak, ve uygulama kurulurken, nesne modeline yerleşecektir.

3.4 comard

Nesne modelini ve üzerindeki betikleri işletecek, profil, transaction, erişim yetkilerini sağlayacak, uzak nesneler için gereken ağ bağlantılarını kuracak uygulamadır. Uygulamalardan ve uzak nesnelerden gelen çağrıları alıp verecek bir iletişim birimi; CSL betiklerini "parse" edecek bir yorumlayıcı; çağrı ve yorumlanmış betikleri işletecek bir mekanizma; gerek betiklerin, gerekse comard'nin, ayar dosyalarına erişim, sistem yönetimi, program çalıştırma, veri depolama gibi işlerini yapacak bir API den oluşmaktadır.

3.5 KGA

KGA (Kullanıcı Grafik Arayüzleri), kullanıcının COMAR ve nesne modeli üzerinde kavramsal olarak bir bütün oluşturan kısımları tek ve basit bir grafik arabirim üzerinde ayarlayabilmesini sağlayan uygulamalardır.