ÇOMAR Mimari Belgesi

Gürer Özen (gurer @ uludag.org.tr)

1 Ağustos 2005

İçindekiler

1 Giriş

Bu belge, Ulusal Dağıtım'ın yapılandırma yönetim sistemi olan ÇOMAR'ın, ne amaçla geliştirildiğini, hangi sorunlara çözüm getirdiğini, yapısını ve bileşenlerini anlatır. Hedef kitlesi, ÇOMAR'ı yakından tanımak isteyen kullanıcılar ve sistem yöneticileridir. Bilişim okuryazarı seviyesine göre yazılmış olmakla birlikte, bazı tartışmaları takip edebilmek için, bir işletim sisteminin bileşenleri, ve uygulamaların nasıl ayarlandığı gibi sistem yönetimi konularında bilgi sahibi olmak gerekebilir.

2 Sorun

Çeşitli uygulamalar bir sistem içinde bir araya getirildiklerinde, birbirleriyle uyumlu çalışabilmeleri için ayarlanmaları gerekmektedir. Kurulan bir uygulamanın masaüstü menüsüne eklenmesi, açabildiği dosya tiplerini sisteme bildirmesi, yeni kurulan bir spam (istenmeyen eposta) filtreleyicinin mevcut eposta sunucusuna bağlanması gibi çok sayıda entegrasyon işlemi bulunmaktadır. Kullanıcı, bu ayarları yapabilmek için, kendi yapmak istediği işin dışındaki teknik konularda bilgi kazanmak zorunda kalmakta ve zaman kaybetmektedir.

2.1 Belgeler

Özgür yazılım camiası, bu işleri kolaylaştırmak için "nasıl" (ing. howto) belgeleri adıyla çeşitli belgeler hazırlamıştır. Bunlar bir işi yapabilmek için neler yapılması gerektiğini adım adım anlatan kısa belgelerdir. Kullanıcıların belge okumak istememeleri ve belgelerin kısıtlı sayıda senaryoyu kapsaması yüzünden faydalı olamamaktadırlar.

Burda aklımıza, madem bir işi adım adım belgeleyebiliyoruz, bunu bir program haline getirip otomatik olarak yapılmasını sağlayamaz mıyız? sorusu gelmektedir. Bu yapılabilirse, kullanıcının zaman tasarrufu yanında, bu belgelerin çeşitli dillere tercüme edilmesi gibi işler de gereksiz hale gelecektir.

2.2 Diğer Linux Dağıtımları

Linux dağıtımları gelişme süreçleri içerisinde bu tür entegrasyon problemleri ile karşılaştıkça bunlara ayrı ayrı çözümler üretip kendi sistemlerine (özellikle paket yönetici yazılımlarının içine) dahil etmişlerdir. Bu çözümler, kurulu uygulamalar (menü), fontlar, açılış işlemleri (initscripts) gibi tek tek alt sistemler bazındadır.

Genellikle, 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.

Bir de uygulama kurulur, kaldırılır ve güncellenirken çalışan özel betikler bulunmaktadır. Bunlarla güncelleme sırasında eski ayarların taşınması gibi işler yapılmaktadır. Bazı sistemler (örneğin Debian'ın debconf'u) kurulum anında bu betiklerin kullanıcıya soru sorabilmeleri ve uygulamayı cevaplara göre ayarlamalarını sağlamaktadır.

Burda gördüğümüz noksanlıklar:

- Her sorun için ayrı bir çözüm kullanılması benzer işlerin tekrar tekrar yapılmasına yol açmaktadır. Genel bir ayar profili oluşturmayı ve yönetmeyi zor kılmaktadır. Birbirleriyle ilişkileri eksik kaldığı için yeterli entegrasyonu sağlayamamaktadır.
- Yapılandırma ile paket kurulumu iç içe geçmiştir. Bir paket (özellikle bir sunucu uygulaması) kurulduğunda çalışacak şekilde yapılandırılmasının yanlış olduğunu, uygulamanın ancak kullanıcı bir emir verdiğinde, verilen emre göre yapılandırılmasının anlamlı olduğunu düşünüyoruz. Yapılandırma, kurma ve kaldırma ile ilgisi olmayan ve uygulama sistemde durduğu sürece her an ihtiyaç duyulabilecek bir iştir.
- Bir sürü veri biçimi ve dil kullanılması, öğrenme ve hata düzeltme süreçlerini çok güçleştirmektedir.
- Bir sorun çıktığında sistemi çalışabilir bir konuma getirmek, uygulamalar güncellenirken kullanıcı ayarlarını ve sistemin tutarlılığını korumak çok zor olabilmektedir.

2.3 Diğer İşletim Sistemleri

Windows, OS/2, MacOS X gibi 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 ile birlikte; çokluortam, ağ protokolü, donanım yöneticisi gibi parçalar için parçaların yerleşebileceği modül yapıları bulunmaktadır.

Bu yaklaşımda 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 kodu üzerinde büyük değişiklikler yapmayı, ve yapılan değişikliklerin yeni sorunlara yol açmadığını kapsamlı olarak analiz etmeyi gerektirmektedir. Bu da büyük zaman ve emek harcamasına yol açacaktır. Kodunu değiştirme olanağı olmayan uygulamaların sisteme entegre edilebilmesini önleyecektir.

- Bu API lerin her uygulamadan kullanılabilmesi ya CORBA, COM gibi karmaşık çözümler ya da çok sayıda “wrapper” gerektirmektedir.
- 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.

2.4 Özel Yönetim Uygulamaları

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 araçlarda gördüğümüz yanlışlar:

- Üst düzey bir model değil, alt düzey ayarlar yöneticiye sunulmaktadır.
- Yapılandırma problemlerini görev tabanlı değil, uygulama tabanlı ele almaktadırlar.
- Uygulamalara görevi yaptırmak için gereken bilgi uygulamalarda değil merkezde (yönetici yazılımın içinde) bulunmaktadır. Bu bilgiler, bir arada oldukları ve birden fazla parçaya ait detayları içerdikleri için, öngörülmeyen durumlarda hata ortaya çıkarma ihtimalleri artmaktadır.

3 Gerekler

Bir yapılandırma sisteminin iki müşterisi olacaktır. Biri sistemin çalışacağı bilgisayardaki kullanıcılar, diğeri ise bu sisteme paket veya yönetim araçları yapacak olan geliştiricilerdir.

3.1 Kullanıcı Gerekleri

1. Yapılandırma sorunları mümkün olduğunda otomatik biçimde çözülmeli, kullanıcıdan ihtiyacı olmayan teknik detayları bilmesi ve ayarlaması istenmemelidir.
2. Otonom olarak çalışabilmeli, gerektiğinde gömülü sistemlerde de kullanılabilmesi için grafik arayüzlerden bağımsız olmalıdır.
3. Görevler koşutzamanlı (concurrent) çalışmalı, biri diğerini yavaşlatmamalıdır.
4. Kullanıcının karar vermesi gereken durumlar kullanıcıya görev tabanlı bir yaklaşımla ve bilişim okuryazarına yönelik terimlerle sunulmalıdır. Görev yerine uygulama bazında ayarların sunulması, soruların teknik detaylara ait terimlerle sorulması istenmemektedir.
5. Otomatik yapılacak yapılandırmalar veya kullanıcının yapılandırma istekleri, sistemi asla iç tutarlılığını kaybetmiş hale getirmemelidir. Başka bir sebepten bu duruma gelen sistemi tekrar tutarlı hale getirebilmek mümkün olmalıdır.
6. Kullanıcının istekleri farklı adlar altında birer profil olarak gruplanabilmeli, sistem bir profilden diğerine rahatça geçebildiği gibi, gerektiğinde bu profiller bir sistemde kaydedilip, diğer bir sisteme taşınabilmelidir.
7. Sistem yöneticisi diğer kullanıcılara belirli kriterlere göre yapılandırma kararı verme yetkisi dağıtabilmelidir. Böylece kullanıcıların gerektiğinde “kök” (ing. root) kullanıcı olmadan da sistemin belirli bir bölümünde (örneğin ağ iletişimi, paket kurulumu, ya da donanım) yapılandırma ve yönetim yapabilmesi mümkün olacak, kök kullanıcıya olan bağlılık ortadan kaldırılabilecektir.

3.2 Geliştirici Gerekleri

1. Her yapılandırma problemi için aynı ortak alt yapının kullanılması, sistemin bir bütün olarak modellenmesi istenmektedir. Bu geliştirme işlerini kolaylaştıracaktır.
2. Uygulamalar, üzerlerinde büyük değişiklikler yapılarak değil, gerekli yapılandırma bilgisini taşıyan ufak aracılar eklenerek sisteme entegre edilmelidir.
3. Her bir uygulama pakedinin yapılandırma bilgisi tamamen kendi içinde olmalı, ve tek bir dil ve veri biçimi ile tanımlanmalıdır.

4. Yapılandırma sistemi, ilerde ortaya çıkabilecek değişik özellikte uygulamaların entegrasyonunda sorun çıkartmayacak esnek bir görev modeline sahip olmalıdır.
5. Yapılandırma sistemi ile kurulacak iletişimde dil veya veri biçimi bağımlılığı problemi olmamalıdır.
6. Sistem, kendi ile iletişim kuran uygulamalara, sürekli bir sorma gereksinimi bırakmadan, bir takım olayların oluştuğunu haber verebilmelidir.

4 ÇOMAR

Bu gerekleri sağlayacak bir sistem oluşturabilmek için ilk adım, yapılandırma sorunlarının tarif edilebileceği bir model oluşturmaktır. Genel olarak iki tip sorun vardır.

Birinci tip sorunlar iki uygulamanın birbiriyle uyumlu çalışmasının gerektiği yerlerde çıkarlar. Bunları çözebilmek için her uygulamanın,

1. Diğer uygulamaların bilgilerine erişebilmesini ve kendi bilgilerini diğer uygulamalara sunabilmesini,
2. Önceden belirlenmiş görevler içinden neleri yapabildiğini sisteme bildirebilmesini,
3. Kendi görevleri dışındaki işlere karışmamasını, bunları ilgili uygulamalardan istemesini,
4. Bilgileri değiştiğinde, ilgilenen uygulamaları haberdar edebilmesini sağlamak gerekir.

Böylece uygulamalar kendilerini birbirlerine göre ayarlayabileceklerdir.

İkinci tip sorunlar ise tek bir uygulamanın yapılandırılmasında ortaya çıkarlar. Aynı görevleri yapabilen uygulamalar fonksiyonel bir sınıf oluşturmaktadır. Değişen şey, her birinin bu görevi yapmak için farklı şekilde ayarlanması gerekeceğidir.

Buradan, uygulamaların eş görevleri olan fonksiyonel sınıflar olarak sınıflandırılabileceği, ve bu sınıflar ortogonal olarak tasarlandığında, uygulamalar arası yapılandırma problemlerinin çözümü için gereken şartları sağlayabileceğimiz sonuçlarına varıyoruz.

Ortogonal olarak tasarlanmış sınıflardan oluşan kapsayıcı bir sistem modeli oluşturduktan sonra, karşımıza modelden istenen görevleri uygulamalara aktarma sorunu çıkmaktadır.

Uygulamaları bize ait API leri kullanacak biçimde değiştirmek gereklerimize uygun değildir. Uygulamaların çalışmalarını yönetecek bilgiler ve ayarlanması gerektiği düşünülen seçenekler genellikle uygulama yazarları tarafından önceden düşünülüp, bir takım ayar dosyalarından, komut satırı parametrelerinden, ve benzer yollarla elde edilip kullanılacak duruma getirilmiştir. O halde en uygun yol, birer aracı programcık ile modeldeki görevleri, uygulamaların kendi ayar ve komutlarına çevirmektir.

Bu programcıkları bir arada ve düzenli olarak çalıştırabilmek, yapılandırmaları profiller bazında yönetebilmek, kullanıcılara görevlere erişim yetkisi verebilmek gibi gerekleri kolayca sağlayabilmek için, bir yönetici servis programı gerekmektedir.

Yapılandırma grafik arayüzleri, paket yöneticisi, üst düzey yönetim programları bir iletişim kanalı aracılığı ile bu yönetici programa bağlanıp, model üzerindeki görevleri kullanabileceklerdir.

Toparlarsak, gereksinimlerimizi karşılayabilecek ve diğer benzer sistemlerdeki sorunlara düşmeyecek bir yapılandırma yönetim çerçevesi için ihtiyacımız olan bileşenler,

bir sistem modeli, uygulamaları bu modele oturtmak için gerekli aracı programcıklar ve bu programcıkları işletecek bir uygulamadır. Şimdi bu bileşenlere detaylı olarak bakalım.

4.1 Sistem Modeli

Sistem modeli, eş görevleri olan uygulamaların fonksiyonel sınıflarından oluşmaktadır. Burdaki sınıf kavramı, nesne tabanlı programlamadaki (OOP) sınıf kavramına yakınlık taşıdığından, aynı isimlendirmeleri kullanmak öğrenme kolaylığı sağlayacaktır.

Buna göre, bir sınıf, bir veya birden fazla uygulamaya ait aracı programcıklar (nesneler) tarafından sağlanan, ve üzerinde yapılacak görevlere karşılık gelen metotlar içeren bir tanımlamadır.

Bu nesnelerin, metotlar dışında OOP anlamında özniteliklere sahip olmaları gerekli görülmemiştir. Bunlar işletilirken sistemin doğası gereği birer fonksiyon olarak çağrılacak ve tek parametreli bir metottan farklı olmayacaklardır.

Ayrı yapılandırma problemlerine yönelik olduklarından dolayı, sınıflar arasında herhangi bir kalıtım (inheritance) ilişkisi yoktur. Bununla birlikte yakın amaçlara yönelik sınıflar (örneğin iletişimle ilgili sınıflar, donanım tanımayl ilgili sınıflar, vb) bir grup ismi altında bir araya toplanmıştır. Grup kavramı, model üzerinde yetki denetimi tanımlarken kolaylık sağladığı gibi, modeli mantıksal olarak düzenli tuttuğu için tercih edilmiştir. Bir grup yalnızca, kendine ait sınıfları bir arada tutar, metot ya da nesneleri yoktur.

Benzer biçimde, aracı programcıklar arasında kod paylaşımı söz konusu değildir. Bunların ihtiyaç duyacağı ortak yordamlar, API olarak ÇOMAR tarafından kendilerine sunulacaktır.

Böylece her bir nesne diğerinden yalıtılmış olacağından, nesnelerin birbirlerinin iç detaylarını bilmesi, ya da başka bir nesneye direk olarak bağlı olması gibi durumlar oluşmayacaktır.

Her uygulama sağladığı sınıflara ait nesneleri yanında taşır ve kurulum sırasında ÇOMAR'a kaydeder. Bu nesneler model üzerinde ait oldukları sınıflara yerleştirilir.

İsimlendirmeler

Model üzerinde gruplar direk adlarıyla gösterilir, sınıflar grup adı ile birlikte,

Grup.Sınıf

biçiminde gösterilirler, her sınıf mutlaka bir grubun içindedir. Sınıf metotları ise

Grup.Sınıf.metotAdı

biçiminde yazılır.

Tasarım Kuralları

Sistem modeli tasarlanırken bazı noktalara dikkat edilmesi gerekmektedir.

1. Belli uygulamaların değil, bu uygulamaların yaptığı görevlerin yapılandırılması gözetilmeli, modelin genelliği yitirilmemelidir.
2. Modelin gelişen teknolojilerle birlikte eskiyip, kullanışsız hale gelmemesi için, esnek olması gözetilmelidir.
3. Bununla birlikte, ucu açık, tanımlanmamış bilgi ve görevler modele sokulmamalıdır.
4. Burda ayrımı doğru yapabilmek için, görev ve bilgilerin genel kullanıma mı, yoksa özel kullanıma mı yönelik olduğu bir kriterdir. Bir nesnenin bir görevi eğer üst katmandaki her nesne tarafından kullanılabiliyorsa geneldir, açıkça ve kesin olarak tanımlanmalıdır. Eğer görevin kullanımı sadece özel bir üst nesne tarafından yapılabiliyorsa, özeldir ve bunun bilgisi tanımlanmaya çalışılmak yerine, üst nesneye hedef olarak verilip, kendi aralarındaki ilişkileri kendilerinin kurmaları desteklenmelidir.
5. Model, kullanıcı ve görev tabanlı tasarlanmakla birlikte, görev uygulamalarının ihtiyaçlarına yönelik teknik bilgiler de taşıyacaktır. Bu durumların modelde açıkça belirtilmesi önemlidir.

4.2 Aracı Programcıklar (CSL)

Ne yazık ki basit bir tanımlama dili, görevleri uygulamalara taşımaya yetmemektedir. Çünkü aracının birçok durumda kendi içinde birden fazla işlem yapması, çeşitli kriterlere göre işi nasıl yaptıracağına karar vermesi, gerektiğinde uygulamayı yönetebilmek için, genel API lerin sağlayabileceğinin dışında fonksiyonlar kullanması gerekmektedir. İterasyon, karar verme, karşılaştırma, aritmetik ve string işlem yapma özellikleri olan bir dil gereklidir.

Bu dilin seçimi özgür bırakılabilir, ancak bu durumda yapılandırma sisteminin bağımlılıkları artmaktadır. En önemlisi diller arası uyum, hata giderme işlemleri ve öğrenme süreci çok güçleşmektedir. Bu nedenle tek bir dil kullanılmalıdır.

Genel bir programlama dilinde bulunan çoğu modül ve kitaplık (özellikle grafik arayüze yönelik olanlar) gerekmeyecektir. Sorun çıkmaması için dilde bu destekler hiç olmamalı ya da kapatılabilmelidir. İhtiyaç duyulacak API ler, ayar dosyalarını okuyup yazma, programları çalıştırıp durdurma gibi işlere yönelik olacaktır.

Bu iş için en uygun dil olarak gördüğümüz Python'u temel aldık. Python işleticisini (VM), bizim belirlediğimiz bazı modülleri (string, re, config modülleri, vb...) alıp, bunun üstüne ihtiyacımız olan diğer fonksiyonları bir modül olarak ekleyerek CSL (Çomar Scripting Language) adını verdiğimiz bir alt dil oluşturduk. Nedenlerimiz:

- PİSİ paketlerinin hazırlama ve derleme betiklerinde de Python kullanıldığı için paket yapıcı tek bir dil öğrenip kullanarak tam bir Pardus pakedi hazırlayabilecektir.
- Python VM, hız ve kaynak kullanımı olarak çok uygundur. Bir işletici program içinden rahatça ayarlanıp kullanılabilen bir kitaplık halindedir.
- Program yazarken sıkça karşılaşılan yapıların (design patterns) çoğu Python'da temel özellik olarak bulunduğu için kod temiz ve anlaşılır olmakta; implementasyon, mantığı gölgelememektedir.
- Minimal ve temiz sentaksı dolayısıyla kodların boyutu kısa, okunabilirliği yüksek olmaktadır.

4.3 Yapılandırma Yöneticisi

ÇOMAR işletici uygulaması (comard), kullanıcı arayüzleri, ÇOMAR destekli uygulamalar ve çeşitli araçlardan gelen görev isteklerini sistem modeli üzerindeki uygulama nesnelerine yaptıran bir sistem servisidir.

Bu istekleri almak, ve olup biten yapılandırma olaylarını bağlanan uygulamalara aktarabilmek için bir iletişim kanalı gereklidir. ÇOMAR'ın ön tanımlı iletişim kanalı sistemde sabit bir UNIX soket olmakla birlikte, yerel bağlantılar için DBus, uzak bağlantılar için HTTP, SSH gibi protokoller, hatta e-posta ya da SMS gibi iletişim kanalları modüler olarak kullanılabilir.

Her bir iletişim modülü, ÇOMAR çağrılarını iletmek, ve gelen çağrıların hangi kullanıcıdan geldiği, iletişim hattının şifreli olup olmadığı, iletinin elektronik imzayla doğrulanıp doğrulanmadığı gibi bilgilere bakarak ÇOMAR'ın yetki denetim mekanizmasından geçirmekle sorumludur. İşletici elindeki nesnelerle sisteme kullanıcı eklemek, alt düzey ayarları değiştirmek gibi işler yapabilmekte, bunları yapabilmek için en yüksek yetki seviyesinde çalışmaktadır. Güvenlik açıklarına yol açmamak için, iletişim modüllerinden gelen isteklerin yetki denetiminden geçmeden işleticiye geçmesine izin verilmemelidir.

Yetki denetimi çağrıyı yapanın kimlik bilgileri ile, model üzerindeki her noktada yapılır. Böylece bir kullanıcıya ayar değiştirme yetkisi vermeden bilgi sorma metotlarını çağırma yetkisi verilebilmesi ya da bütün bir grubun yönetiminin basitçe tek bir kullanıcıya verilmesi sağlanabilir.

Görevleri sağlayan nesneler paralel olarak veya çağrı bir nesneye yönelikse tek olarak işletilir. Bir nesne içinden başka bir sınıfa yeni bir çağrı yapılabilir. Bir paket kurulduğunda uygulamanın nesnelerini model kaydettiren, kaldırıldığında çıkaran çağrılar da mevcuttur.

Özellikle açılış esnasında bir sürü işlem yapılmaktadır, bu işlemler birbirlerinden bağımsız oldukları, aralarındaki bağımlılıklar çok az olduğu için paralel çalıştırılmaları büyük hız kazancı sağlayacaktır. İsteklerin paralel yürütülebilmesi, kullanıcının interaktif işlemlerine çabuk yanıt verebilmek için de önemlidir. Bu amaçla her bir nesne

ayrı bir süreç olarak işletilecektir. Linux'ta yeni bir süreç yaratan fork çağrısı, bir performans kaybı yaratmayacak kadar hızlı çalışmakta ve süreçlerin bellek alanları copy-on-write metodu ile çoğaltıldığı için gereksiz kaynak israfına da yol açmamaktadır.

Yapılandırma işlemleri sistemde sürekli ve sık biçimde yapılmamaktadır. Yapılacak işler azaldığında ya da iş olmadığında minimum kaynak kullanımına geçilebilmelidir. Nesnelerin ayrı süreçler olarak işletilmesi bunu da kolaylaştırmakta, işler hep ana süreç dışında yapıldığı için, bir iş olmadığında sadece temel takip işlemleri çalışır halde kalmaktadır.

Nesneler belirli bir durumda (bir sistem olayı ya da peryodik zaman olayları) bir metotlarının çağrılmasını isteyebilirler. ÇOMAR işleticisi bu istekleri kaydeder ve ilgili olay meydana geldiğinde ilgilenen nesneleri çağırır.

4.4 Kullanıcı Arayüzleri

ÇOMAR'ı kullanacak en temel uygulama PİSİ'dir. Paketleri kurarken, pakede ait nesneleri ÇOMAR'a verecek ve uygulamanın sisteme entegre edilmesini sağlayacaktır. Paket kaldırılırken ise ÇOMAR'a durumu bildirerek nesnelerin modelden çıkarılmasını sağlar.

Kullanıcının görevleri kullanmasını ve sistemini ayarlayabilmesini sağlayacak uygulama ise TASMA'dır. Bir grafik arayüzü olan TASMA, ÇOMAR'daki bilgileri kullanıcıya sunmak, ve kullanıcının emirlerini ÇOMAR çağrılarına dönüştürmek işlerini yapar.

Bunlar dışında çeşitli arayüzler veya yönetim uygulamaları da ÇOMAR'a bağlanıp hizmetlerinden yararlanabilir.

5 Sıkça Sorulanlar

5.1 ÇOMAR'ın açılımı nedir?

COnfiguration MAnageR. ÇOMAR'ın açılımı ilk olarak "Configuration by Objects, Modify and Restart" idi. Fakat ÇOMAR'ın tasarım sürecinde "Modify and Restart" kısmının ÇOMAR'ın işlevselliğini tam olarak ifade etmez hale geldiği görüldü ve açılımının "Configuration Manager" olmasının daha doğru ve anlamlı olacağında karar kılındı.

5.2 ÇOMAR bana ne fayda sağlayacak?

Kurduğunuz uygulamaları elle ayarlamaktan, sistemin zaten bildiği ve kendi başına bulabileceği bilgileri elle girmekten, bunun için belge okuyup soru sorarak zaman kaybetmekten kurtulacaksınız.

Sistemin sürekli olarak tutarlı bir durumda kalmasını sağlayarak, ayar sorunları yüzünden çalışamayan programlardan sizi kurtaracak.

Sunduğu imkanlar ile tek tek uygulama ayarlamaktan ziyade, görev temelli düşünülmüş grafik arayüzler yazılmasını kolaylaştıracak, bu arayüzler sayesinde bilgisayara kölelik yapmak yerine kendi işinizle uğraşabileceksiniz.

5.3 ÇOMAR desteklemeyen uygulamaları kullanabilecek miyim?

Elbette. Bu uygulamalar ÇOMAR'ın sağladığı avantajlardan faydalanmayacaklar, ama sistemde çalışabilmelerinin önünde bir engel olmayacak.

5.4 Bir uygulamaya ÇOMAR desteği vermek zor mu?

Hayır. Bunun için uygulamayı değiştirmenize gerek yok. Yalnızca CSL ile ÇOMAR modelindeki görevlerin uygulamaya nasıl yaptırılacağını tarif eden betikler (nesneler) yazmanız yeterli.

5.5 CSL yeni bir dil mi?

Aslında hayır. CSL bir Python alt dili. Python'un ihtiyacımız olmayan modülleri çıkarılıp, bazı yeni modüllerin eklenmesiyle oluşturulmuş, ve sistem modelimizdeki sınıflara nesne yazmak için kullanılacak hale getirilmiş hali diyebiliriz. İlk ÇOMAR tasarımı ve prototipinde Javascript/C arası ve çok kısıtlı bir dil olarak tasarlanmıştı, ama bunun yeterli gelmediği ve basitlik sağlamadığı görülünce Python temelli olmasına karar verildi.

5.6 ÇOMAR ile PİSİ arasında nasıl bir ilişki var?

ÇOMAR ve PİSİ, diğer dağıtımlarda bir arada olan kurulum ve yapılandırma işlerini ayırıyor ve her işi kendi sorumluluk sahası içinde düzgünce tarif ediyorlar. Birbirlerine ihtiyaç duyduklarında kullanacakları arabirim ise düzgün bir biçimde tanımlanmış. Böylece temiz ve basit bir çözüm sağlıyorlar.

5.7 ÇOMAR'ı devreden çıkartırsam ne olur?

Otomatik yapılandırma işleri durur, ve ÇOMAR ile çalışan yapılandırma arayüzleriniz (TASMA) artık çalışmaz. Yani artık kendi başınızasınız demektir. ÇOMAR'ı yeniden başlatarak bu durumdan kurtulabilirsiniz.

5.8 ÇOMAR'ın kconfig, gconf, elektra gibi sistemlerden farkı ne?

Bu sistemler "configuration" ismini kullanmalarına rağmen aslen özel bir veri saklama (storage) sisteminden başka bir şey değildirler. Uygulama bazında, belirli anahtar kelimelere karşılık gelen verilerin saklanması ve getirilmesini sağlarlar. Bu anahtarlar sistem çapında tanımlanmamıştır ve her uygulama için farklıdır. Uygulamaların alt düzey ayarlarına erişmenizi sağlarlar, ama bir görevi yapmak için hangi ayarların değişmesi gerektiği, aynı işi yapan farklı bir uygulamanın bilinmeden nasıl ayarlanabileceği, ayarlar karıştığında sistemin tutarlı bir hale nasıl getirilebileceği gibi sorunlara bir çözüm getirmezler.

5.9 Neden başkaları böyle bir çözüm getirmedi?

Diğer dağıtımlar çözümlerini tarihsel gelişme süreçleri içinde adım adım geliştirdikleri ve geçmişe uyumluluk yüküyle yollarına devam ettikleri için bu tür kapsayıcı ve düzenli çözümler getirmeleri zor. Bir çok yeni girişim ise genel bir model oluşturmayı ihmal ederek, sorunu bir ayar deposu (configuration storage) olarak ele almaya devam etmekte.

6 Emeği Geçenler

İlk sürüm:

Serdar Köylü, A. Murat Eren, Gürer Özen

Gözden geçirme:

Barış Metin, S. Çağlar Onur, Onur Küçük

İkinci sürüm:

Gürer Özen, Barış Metin, Eray Özkural