X Window System Architecture Overview HOWTO

ManriqueDaniel

roadmr@entropia.com.mx

도운이

고경진

zero2one@netian.com

이 문서는 X윈도우 시스템 아키텍처와 윈도우 매니저, 툴킷, 위젯 라이브러리, 데스크탑 환경 등 구성요소에 대한 선택들 그리고 X에 포함된 요소들과 그래픽컬한 환경을 위한 작업에 관한 개괄을 설명합니다.

고친 과정

고침 1.0.1	2001-05-22	고친이 dm

Some grammatical corrections, pointed out by Bill Staehle

고침 1.0	2001-05-20	고친이 dm

Initial LDP release

1. 들어가는 말

이 문서는 X에 포함된 구성요소들과 그래픽컬한 환경을 위한 조합 그리고 이러한 구성요소들을 다루면서 선택할 수 있는 것들, 이러한 아키텍처가 설계된 방법과 이유에 대한 보다나은 이해를 위해 X윈도우 시스템의 아키텍처에 관한 개괄을 제공하기 위해 쓰여졌다.

몇몇 개념들, 위젯이나 툴킷, 윈도우 매니저, 데스크탑 환경 등등은 자주 언급되어 지지만 초보자들에게는 낯선 단어일 것이다. 이러한 요소들이 어떻게 작용하는지는 차츰 읽어나가면서 예를 들어 설명할 것이다.

이 문서는 기술적인 서술은 피하려고 한다. 단지 저자의 주제에 관한 경험적인 사실에 바탕을 두고 서술하고자 한다. 비록 평이한 소개문이지만 몇몇 논평들과 상세한 보기와 설명, 기술적인 서술 등은 유용하리라 생각된다. 저자는 이 문서에 관한 어떠한 질문이나 논평도 환영하며 roadmr@entropia.com.mx. 주소로 연락할 수 있다.

2. 소개

UNIX가 1970년경 새로운 물건으로 인식될 때 대학 연구실에서 동작되던 GUI환경은 신비로운 또 하나의 사실이었다(제록스의 PARC 플랫폼에서). 하지만 요즘 들어서는 GUI는 경쟁력 있는 OS가 되기 위해서는 필수 불가결한 요소가 되고 있다. GUI는 사용하기 쉬워야 함에도 UNIX머신에서는 전통적으로 사용의 편이성보다는 다양성을 추구하므로 그 동안 등한시 되어왔다. 그러나 UNIX도 GUI가 필요로 한 데에는 몇 가지 이유가 있다. 예를 들면 UNIX의 멀티태스킹은 동일한 시간에 여러 프로그램을 돌려야 하고 GUI는 동일한 시간에 한 화면에 여러 프로그램을 운영하여 좀더 나은 편이성을 제공하기 때문이다. 또한 일부 자료형은 그래픽컬 환경에서 더 잘 표현된다(graphical data와 pr0n등은 오직 이 환경에서만 표현할 수 있다).

역사적으로 볼 때 UNIX는 대학으로부터 발전되어 왔다. 그 한 예가 1970년대 후반 사용되어진 Berkeley대학 연구 성과물이었던 BSD networking code이다. 이것이 마찬가지로 MIT대학의 아데나(Athena)프로젝트로 개발된 X Window System (X라고도 하지만 X Windows는 아님)이 리눅스와 BSD를 포함하여 현대의 Unix가 사용하는 GUI 시스템의 기초가 된 것이다.

UNIX는 처음부터 다중사용자, 다중작업, 시분할(timesharing)의 운영체제이다. 또한 네트워크 기술에 기반하여 사용자가 원격에서 접속하고 실행할 수 있는 능력이 있다. 전통적인 telnet이나 시리얼 터미널이 이러한 기반 위에서 작동되었던 것이다.

Unix에서 본질적으로 작동되는 GUI 시스템이 개발되면서도 이러한 개념들이 유지되었고 설계에 반영되었다. 사실은 X는 매우 복잡한 설계구조를 가지고 있다. 이것은 종종 불리한 점으로 언급되어 지지만 이러한 설계구조가 한편으로는 진정한 유연성 시스템인 것이다. 이것은 Unix에서의 GUI를 구성하는 모든 요소가 어떻게 조합되는가를 설명하면서 더욱 분명해질 것이다.

X의 아키텍처를 살펴보기 전에 우선 간략한 역사와 이것이 GNU/Linux에서 동작하게 된 경위를 알아보고자 한다.

X는 아데나 프로젝트로 개발되었으며 1984년에 발표되었다. 1988년에 'X 컨소시엄'이라는 단체가 이 X를 넘겨받아 오늘날까지 개발과 배포를 담당하고 있다. X는 자유로이 사용 가능하고 이로 인해서 X가 대중적으로 사용되어져 왔고 XFree86도 이렇게 해서 나왔다. XFree86은 우리가 Linux 컴퓨터를 사용할 때의 X를 말하며 (X의 인텔 버전) BSD계열이나 OS/2 또는 몇몇 다른 운영체제에서도 작동한다. XFree86 이라는 이름임에도 이것은 다른 CPU 아키텍처에서도 사용할 수도 있다.

3. X윈도우 시스템 아키텍처: 개괄

X는 클라이언트-서버 아키텍처로 설계되었다. 어플리케이션 그 자신이 클라이언트이고 서버와 서로 통신하며 정보요구(request)를 발생시키고 서버로부터 정보를 얻는다.

X서버는 클라이언트의 화면표시와 서비스 정보요구(request)에 대한 배타적인 제어권을 갖는다. 여기에서 이러한 모델에서의 장점이 분명함을 알 수 있다. 어플리케이션(클라이언트)는 단지 어떻게 서버와 통신할 것인가를 알기만 하면 되고 화면표시장치에 어떻게 정보를 전달할 것인가는 알 필요가 없는 것이다. 아주 기초적인 레벨에서 클라이언트는 '여기서 저기까지

선을 그려라' 또는 ' 이 글꼴을 사용하여 이 지점에 문자열을 출력하라'등의 명령만을 서버에게 전달하면 되는 것이다.

이것은 어플리케이션을 작성할 때 그래픽 라이브러리를 사용한다 하여도 아무런 차이도 없음을 말한다. 게다가 X모델은 한 단계 더 진보적이다. 이것은 클라이언트가 서버와 동일한 컴퓨터에서 작동되는 것을 강요하지 않는다. 서버와 클라이언트간에는 프로토콜로, 프로세스간 상호 통신 메커니즘은 믿을만한 octet stream을 제공하므로서, 서로 통신하기 때문이다. 짐작하겠지만 이것은 TCP/IP 프로토콜을 사용하므로 구현될 수 있다. 우리가 알고 있는 한 X모델은 강력하다. 이것의 전형적인 좋은 예는 Cray computer에서 프로세서 의존적인 어플리케이션을 구동한다던가 솔라리스 서버에서의 데이터베이스를 small BSD 메일 서버에서의 이메일 어플리케이션을 SGI 서버에서 비주얼 프로그램을 그리고 내 GNU/Linux 워크스테이션 화면에 이 모든 것을 표시할 수 있다는 것이다.

지금까지 우리는 X 서버가 그래픽 표시를 담당하는 장치라는 것을 알았다. 또한 실제적으로 사용자가 작업하는 물리적인 컴퓨터를 구동하는 것도 X 서버이기 때문에 사용자와의 상호작용 또한 X의 몫이다. 마우스나 키보드의 입력을 받는 것도 X이고 이 정보를 클라이언트에게 전달해야 한다. 당연히 클라이언트는 이 정보에 대하여 반응할 것이다.

X는 Xlib이라는 저 수준의 클라이언트-서버 통신을 제어하는 라이브러리를 제공한다. 다시 말해서 클라이언트는 어떠한 일을 하기 위해서는 Xlib에 포함된 함수를 호출해야 한다는 것을 의미한다.

이 시점에서 우리는 시각적인 출력과 데이터 입력, 클라이언트 프로그램, 그리고 이것들이 서로 통신하는 방법을 서버가 담당한다는 것을 알게 되었다. 서버와 클라이언트가 서로 작용하는 것을 상상해 본다면 클라이언트는 서버로부터 화면에 사각 영역이 할당되는 것을 요구할 것이라고 생각할 수 있다. 사실 클라이언트 입장에서 보면 자신이 어디에 표시되고 있는가는 관심 밖의 일이다. 단시 서버로부터 X와 Y만큼의 크기만 요구하면 되고 '여기에서 저기까지 선을 그려라'라든가 '사용자가 마우스를 자신의 영역으로 이동하였는지 알려달라'등 함수만 호출하면 되는 것이다.

4. 윈도우 매니저

아직까지 우리는 X서버가 클라이언트의 화면(윈도우)영역에서의 처리를 어떻게 담당하는지 설명하지 않았다. GUI를 사용해본 사람이라면 클라이언트의 화면영역에 컨트롤이 필요하다는 것을 짐작할 것이다. 사용자는 보통 크기를 달리한다거나 옮기는 등의 작업을 한다. 그러면 X서버는 어떻게 이러한 일을 처리할까? 답은 '하지 않는다'이다.

X의 기본적인 원칙 중 하나는 '메커니즘은 제공하지만 수법은 아니다(we provide mechanism, but not policy)'라는 것이다. 그래서 X서버가 윈도우 조작에 필요한 방법은 제공하지만 정작 이러한 조작이 어떻게 작동하는가는 말하지 않는다.

이러한 메커니즘/수법의 기묘한 사실은 기본적으로 다음과 같이 요약될 수 있다 : 화면상의 공간을 통제하는 것은 다른 프로그램의 몫이다. 이 '다른 프로그램'이 윈도우를 어디에 위치시킬 것인가를 결정하고 윈도우의 겉모습이나 위치 크기 등을 사용자가 다룰 수 있는 메커니즘과, 우리가 윈도우를 제어할 수 있게 윈도우의 틀과 단추 제목 등의 장식을 제공해 준다. 이러한 윈도우를 다루는 프로그램을 '윈도우 매니저'라고 부른다.

윈도우 매니저는 X의 또다른 클라이언트이다 - 이것은 이러한 특별한 이점이 있음에도 X 윈도우 시스템의 일부는 아니다. 또한 유일한 윈도우 매니저가 있는 것도 아니다. 사용자가 서로 다른 방식으로 윈도우와 상호작용하고 상이한 배치와 장식, 그리고 키보드와 칼라맵을 바꿀 수 있다.

X 아키텍처는 윈도우 매니저가 이러한 모든 윈도우 상의 일을 처리할 수 있게 해 주지만 X는 어떠한 윈도우 매니저도 제공하지 않는다.

이런 이유로 많은 윈도우 매니저가 있다. 윈도우 매니저가 X의 외부 요소이므로 당신이 윈도우가 어떻게 보이고 어떻게 행동할 것이며 어디에 있을 것인가 등은 당신의 취향에 달렸다. 어떤 윈도우 매니저는 극히 단순하고 추하며(twm), 또 어떤 것은 화려하고 모든 요소를 갖추었고(enlightenment) 이들 사이에 fvwm, amiwm, icewm, windowmaker, afterstep, sawfish, kwm등 많은 것이 있다. 각 취향에 맞는 윈도우 매니저가 있는 것이다.

윈도우 매니저의 기본적인 임무는 다른 클라이언트를 제어하는 "meta -client"이다. 대부분의 윈도우 매니저는 몇 가지의 추가적인 편이요소를 제공하고 몇몇은 상당한 편이요소를 제공한다. 대부분의 윈도우 매니저가 제공하는 기능은 클라이언트(응용프로그램)를 구동시키는 것이다. 이 중 일부는 당신이 표준적인 명령을 입력할 수 있는 명령박스를 제공하고 어떤 것은 체계화된 메뉴를 제공하기도 한다. 하지만 이것들이 표준은 아니다. 그것은 X가 클라이언트는 어떻게 구동되야 한다고 정하지 않기 때문이며(no policy) 단지 이것은 클라이언트 프로그램에서 구현되는 편이요소일 따름인 것이다. 그럼에도 윈도우 매니저의 전형적인 임무는(서로 다르게 구현되지만) 다른 클라이언트 프로그램을 구동시키는 ’클라이언트 프로그램’이라는 것이다. 프로그램 런칭 패드를 생각해 보라. 이렇게 해서 사람들은 많은 프로그램 런칭 프래그램을 갖게 된 것이다.

5. 클라이언트 어플리케이션

잠깐동안 클라이언트 프로그램을 이야기하려 한다. 단지 X만을 가지고 클라이언트 프로그램을 만든다고 생각해 본다면 Xlib을 사용한다는 것이 매우 스파르타적이고 화면에 단추를 넣고 글씨를 쓰고 스크롤 바나 라디오 박스를 넣는 일 등은 사용자 입장에서는 매우 복잡한 일일 것이다.

다행히도 어떤 이가 이러한 제어에 대한 프로그래밍의 어려움을 알고 우리에게 사용할만한 ’형식’을 만들었다. 이것이 라이브러리이다. 이 것들은 흔히 위젯(widgets)으로 알려졌고 위젯 라이브러리라고 부른다. 이렇게 해서 단지 몇몇 매개변수(parameters)로 라이브러리의 함수만 호출하면 화면에 버튼을 그릴 수있다. 이러한 위젯의 예를 들자면 메뉴, 버튼, 라디오버튼, 스크롤바, 캔버스(canvas)가 있다.

캔버스(canvas)라는 위젯은 조금 별난 위젯이다. 이것은 기본적으로 클라이언트 안에서 그릴 수 있는 영역(sub-area)이다. 이 곳은 위젯 라이브러리가 간섭하기 때문에(위젯 라이브러리로 그림을 그림) Xlib을 직접적으로 사용하지 않아도 되고 때문에 라이브러리 자체가 임의대로 그래픽을 그릴 수 있는 방법을 제공하여 준다.

사실 실제적으로 화면에 요소들을 그리는 것은 위젯 라이브러리이기 때문에 사용자의 반응(키보드 입력과 마우스 행동)만을 입력으로 간주하는 한 라이브러리는 각 클라이언트의 면면과 행동을 결정짓는다고 할 수 있다. 개발자의 입장에서 보면 위젯 라이브러리는 일률적인

API(함수들의 세트)를 갖게 되고 따라서 이러한 요소가 어떤 위젯을 사용할 것인가를 결정짓게 된다.

6. 위젯 라이브러리(툴킷)

아데나 프로젝트로 개발되었던 원형 위젯 라이브러리는 '아데나 위젯'으로 불리는 아데나 위젯 라이브러리이다. 이것은 매우 간단하고 추하며 사용법은 오늘날의 표준으로 여겨지는 것같이 직관적이지 않다. 예를 들면 스크롤바를 움직이기 위해 끌 기하는 대신 위로 스크롤하기 위해 오른쪽 버튼을 클릭하고 내리기 위해서는 왼쪽 버튼을 클릭한다. 알다시피 이것들은 요즘에 사용하지 않는다.

윈도우 매니저와 함께 운영되기 때문에 많은 툴킷이 저마다의 다른 설계목적을 가지고 생겨났다. 최초의 잘 알려진 툴킷 중 하나는 Motif이고 이것은 Open Software Foundation의 윈도우 매니저와 툴킷을 포함한 Motif 그래픽컬 환경의 일부였다. OSF의 역사에 대하여는 이 문서의 범주에 벗어나므로 생략하고 어쨌든 아데나 위젯보다 뛰어났던 Motif 툴킷은 80년대와 90년대 초에 많이 이용되었다.

요즘에는 Motif가 그리 대중적인 툴킷은 아니다. 우선 Free가 아니고 개발자가 자신의 프로그램을 이것으로 컴파일하고자 할 때 개발자 라이센스로 돈을 지불해야 한다는 것이다. 그럼에도 바이너리 배포는 사용할 수 있다. 아마도 가장 잘 알려진 Motif 어플리케이션은, 적어도 리눅스 사용자들에게는, Mozilla의 전신인 Netscape Navigator/Communicator일 것이다.

한동안 Motif는 사용가능한 유일한 툴킷이었고 많은 Motif 소프트웨어가 있었다. 그래서 많은 사람들이 이를 대체할 수 있는 툴킷을 개발하였고 XForms나 FLTK와 몇몇 툴킷이 있었다.

요즘에는 Motif를 많이 사용하지 않는다. 특히나 자유 소프트웨어 세계에서는 더욱 그러하다. 보다 좋은 것들이 나왔고 라이센스 문제도 그러하고 성능면(Motif는 살찐 돼지에 간주된다)에서도 좋지 않기 때문이다.

이러한 툴킷 중 잘 알려진 것은 Gtk이고 GIMP 프로젝트 일환으로 Motif를 대체하기 위해 만들어졌다. (Gtk 가 GIMP ToolKit로 알려지고 사용되어지나 GNU ToolKit으로 해석될 수 있다.) 이것은 상대적으로 가볍고(lightweight) 구성이 풍부하고 확장성과 무엇보다 Free이기 때문에 매우 인기 있다. 0.6판의 chagelog에 "Bloatif has been zorched"라는 말은 Motif가 비대해졌음을 말하는 것이다.

또 하나의 인기있는 툴킷은 Qt이다. 이것은 KDE 프로젝트가 GUI를 구현하면서 사용함에 따라 알려지게 되었다. 여기서는 Qt의 라이센스 문제와 KDE/GNOME 비교하는 지리한 논박은 하지 않겠다.

마지막으로 언급할만한 것이 하나 있는데 LessTif라는 툴킷이다. Motif에서 따온 이름이며(more와 less) free를 위해 만들어졌다. Motif와 호환되며 새로운 개발로 계획되었다기보다는 Motif 코드로 어플리케이션을 포팅할 때 대체할 수 있게 만들어진 것이다.

7. 지금까지 우리가 알고 있는 것

여기까지 설명하면서 우리는 X가 어플리케이션 프로그램을 클라이언트로 하는 서버-클라이언트 아키텍처를 가지고 있다는 것을 알 것이다. 이러한 서버-클라이언트 시스템에서 우리는 몇 개의 사용가능한 윈도우 매니저를 알고 있으며 이것이 화면을 관리한다는 것을 안다. 우리가 원하는 일을 수행하기 위해서 우리는 몇 가지 다른 툴킷을 이용하여 프로그래밍한 클라이언트 프로그램이 필요하다는 것도 알것이다.

이 시점에서 아마 혼란이 시작될 것이다. 각각의 윈도우 매니저는 서로 다른 방법으로 클라이언트에 접근하고 행동양식과 장식도 각기 다르며 또한 클라이언트가 어떠한 툴킷을 이용하는가에 따라 달라진다. 개발자가 어플리케이션을 만들기 위해 하나의 툴킷을 이용해야 한다는 법이 없기 때문에 사용자는, 물론 가정이지만, 서로 다른 툴킷을 사용한 외양과 행동이 조금씩 다른 여러 개의 프로그램을 구동하는 것이 가능하게 된다. 어플리케이션 사이의 일관된 양식이 없으므로 이것은 혼란을 일으키게 된다. 만약에 당신이 아데나 위젯으로 짜여진 프로그램을 사용해 본 적이 있다면 Gtk로 짜여진 것과는 많이 다르다는 것을 알 수 있을 것이다. 또한 외양과 느낌이 다른 어플리케이션을 사용한다는 것이 혼란스러울 것이다. 이것은 무엇보다도 GUI환경의 장점에 반하는 것이다.

기술적인 관점에서 본다면 많은 툴킷을 사용하는 것은 자원을 낭비하는 것이다. 현대의 운영체제는 동적공유라이브러리(dynamic shared libraries)를 지원하는데 이것은 만약 내가 Gtk를 이용한 2-3개의 어플리케이션과 Gtk의 동적 공유 라이브러리 버전을 갖고 있다면 이 2-3개의 어플리케이션은 디스크와 메모리에서 Gtk를 공유할 수 있다는 것이다. 자원을 절약하는 것이다. 그럼에도,예를 들어 설명하자면, 내가 Gtk 어플리케이션과 Qt 어플리케이션과 아데나 기반 프로그램과 Netscape같은 Motif기반 프로그램, FLTK나 XForms를 이용한 프로그램을 갖고 있다면 나는 6개의 서로 다른 라이브러리를 메모리에 적재해야 한다. 자원을 낭비하게 되는 것이다. 한가지 기억해야 할 것은 이 모든 툴킷이 기본적으로 같은 기능을 갖고 있다는 것이다.

또 다른 문제가 더 있다. 윈도우 매니저마다 프로그램을 구동하는데 방법이 다르다는 것이다. 어떤 것은 메뉴를 갖고 있고 어떤 것은 그렇지 않으며 명령박스를 열어야 하는 것과 키를 조합해야 하는 것도 있고 명령을 입력하기 위해 xterm을 열어야 하는 것도 있다. 다시 말해서 표준이 없기 때문에 혼란스러워지고 있는 것이다.

마지막으로 그 동안에 다루지 않았지만 GUI 환경에서 생각할 수 있는 부가요소들(niceties)이 있다. 설정 유틸리티나 컨트롤 패널 또는 그래픽컬한 파일 매니저 같은 것들로 당연히 이러한 것들은 클라이언트 애플리케이션으로 만들어질 수 있다. 그래서 자유소프트웨어 세계에서는 정말 많은 파일 매니저와 시스템 설정 프로그램이 상이한 소프트웨어 구성으로 기억하기에도 벅찰 만큼 많이 있다.

8. 해결 방안으로의 데스크탑 환경

여기에서는 데스크탑 환경이라는 개념에 대하여 이야기하고자 한다. 이것은 데스크탑 환경은 이전에 기술되었던 여러 문제들을 최소화하고자 모든 요소에 표준을 제시하는 가이드라인과 편이요소를 제공한다는 것이다.

데스크탑 환경이라는 개념은 처음 리눅스에 도입될 때 새로운 그 무엇이였다. 그것은 Windows나 MacOS와 같이 타 운영체제가 갖고 있던 그 무엇이었기 때문이다. 예를 들면 일찍부터 GUI를 사용한 MacOS는 일관된 look-and-feel을 모든 세션에서 제공한다. 다시 말해 앞에

서 제기되었던 부가요소들(기본 파일 매니저(the finder)와 시스템 전체에 걸친 컨트롤 패널) 과 모든 어플리케이션이 사용하는 단 하나의 툴킷을 제공한다. 어플리케이션 윈도우는 시스템에 의해 관리되고(엄밀히 말하면 여기에도 윈도 매니저는 있다) 무엇보다 개발자가 자신의 어플리케이션이 어떻게 행동해야 하는지 알 수 있고 외형과 위치를 제어하는 것이나 시스템 안에서 이루어지는 프로그램들간의 행동을 규약하는 가이드라인을 가지고 있다. 이 모든 것이 쉬운 사용법과 일관성을 위해 이루어진다.

여기에서 다음과 같은 의문을 갖을 수 있을 것이다. "왜 처음부터 X 개발자는 그렇게 하지 않았지?" 충분히 그렇게 생각할 수 있다. 만약 그랬다면 이전에 제기되었던 많은 문제들이 발생하지 않았을 것이다. 답을 하자면 그것은 X를 설계하면서 개발자들이 가능한 한 유연하게 만들기 위해 그랬던 것이다. mechanism/policy 전형으로 말한다면 MacOS는 대부분의 수법(policy)를 제공하는 것이다. 메커니즘도 있지만 이러한 OS는 사용자가 이것을 조작하는 것을 원하지 않는다. 결과적으로 다양성을 잃게 되는 것인데 만약 당신이 MacOS의 윈도우 관리 방식이나 당신이 원하는 기능을 툴킷이 제공하지 않는다면 당신은 무척이나 불행하게 될 것이다. 이러한 일이 X에서는 없다. 앞에서 살펴보았듯이 유연성의 대가는 복잡 그 이상인 것이다.

Linux/UNIX와 X에서 이러한 문제를 인식하고 ’해결책’을 내놓았다. KDE를 예로 들자면 KDE는 윈도우를 관리하고 제어하는데 하나의 윈도우 매니저를 갖는다(kwm). 특정한 툴킷(Qt)를 사용하므로 모든 KDE 어플리케이션은 화면상에서 같은 모습을 하고 있게 된다. KDE는 메뉴, ’about’상자, 프로그램 툴바를 만들고 프로그램들 간에 통신과 프린팅, 파일선택 등의 일을 수행하기 위해 환경-규정(environment-specific) 라이브러리인 kdelibs를 제공하여 Qt를 확장할 수 있다. 이것은 프로그래머가 일을 쉽게 할 수 있고 프로그램의 행동 양식을 표준화할 수 있다. 또한 KDE는 일련의 디자인과 프로그래머에게 가이드라인을 제공함으로 KDE에서 구동되는 프로그램은 유사한 모양과 행동을 보이게 된다. 마지막으로 KDE는 환경의 일부로서 구동 패널(kpanel)과 표준 파일 매니저(지금은 Konqueror로 바뀜), 설정 유틸리티(control panel)를 제공하여 화면배경이나 타이틀바와 같은 설정에서부터 하드웨어 설정에 이르기까지 컴퓨터 환경을 다양하게 설정할 수 있게 해준다.

KDE 패널은 MS Windows의 작업표시줄과 같다. 어플리케이션을 구동하는 시작점을 제공하고 애플릿이라는 패널 안에서 표시되는 작은 어플리케이션도 있다. 이것은 대부분의 사용자가 없어서는 안될 시계와 같은 기능들을 포함할 수 있다.

9. 데스크탑 환경 몇 가지

KDE를 예로 들었는데 이것은 Unix 시스템에서 초기의 데스크탑 환경은 아니었다. 아마도 최초의 것 중 하나는 OSF의 자매격인 CDE(Common Desktop Environment)일 것이다. CDE FAQ에 보면 "CDE는 UNIX를 위한 표준 데스크탑이고 최종사용자와 시스템 관리자들과 어플리케이션 개발자들에게 플랫폼에 상관없는 일관된 서비스를 제공한다" 라는 말이 있는데 이것은 데스크탑의 키워드는 일관성이라는 것을 말해 주고 있다. 하지만 CDE는 구성요소가 풍부하지 않았고 그렇게 쉽지도 않았다. 더 우수한 환경이 나타나면서 Motif와 함께 CDE는 자유소프트웨어 세계에서 사라지게 되었다.

리눅스에서 가장 인기 있는 두 가지 환경은 KDE와 GNOME이지만 이것만이 다는 아니다. 인터넷 검색을 한다면 대여섯개의 데스크탑을 볼 수 있다(GNUStep, ROX, GTK+XFce, UDE

등등). 이것들은 모두 앞서 말했던 기본적인 편이요소들을 제공한다. KDE와 GNOME은 단체와 산업체로부터 지원을 받고 있으며 이용자나 어플리케이션에 상당한 서비스를 제공하는 가장 진보된 환경이다.

앞서 우리는 KDE와 그 구성요소가 그 환경에서 제공하는 서비스들을 살펴보았다. 하나의 좋은 데스크탑 환경으로서 GNOME도 이러한 면에서 어느 정도 비슷하다. 가장 두드러진 특징은 GNOME은 특정 윈도우 매니저에 구애받지 않는다는 것이다(KDE는 kwm을 쓴다). GNOME 프로젝트는 항상 윈도 매니저에 개방적이고 또한 대부분의 사용자가 자신의 윈도 매니저에 어느 정도 집착한다는 것과 다양하게 동작되는 윈도우 관리방식을 한 매니저로 강요하는 것은 좋지 않다는 것을 인정한다. GNOME은 초기에 Enlightenment 윈도우 매니저를 즐겨 사용하였고 최근에는 Sawfish를 주로 사용한다. 이와는 상관없이 GNOME의 컨트롤 패널에는 윈도우 매니저 선택상자를 갖고 있다.

또 다른 차이는 GNOME은 Gtk 툴킷을 사용하고 고수준의 기능과 편이성을 위해 라이브러리로 gnome-libs 세트를 제공한다. 어플리케이션 사이의 일관된 행동을 보장하는 프로그래밍 가이드라인을 가지고 있고 패널을 제공하며(그냥 '패널' 임) 파일 매니저(gmc, 아마도 이것은 Nautilus로 대체될 것이다)와 컨트롤 패널(the gnome control center)도 제공한다.

10. 어떻게 이것들이 작동하나

어떤 데스크탑 환경이 가장 좋게 느껴지는가는 사용자 각자의 자유이다. 만약 당신이 모든 KDE와 GNOME 시스템을 사용하였다면 최종결론은 이러한 환경의 look and feel이 매우 일관성을 갖고 있다는 것이고 어플리케이션이 매우 근사하게 서로 반응한다는 것이다. 이것은 서로 다른 여러 툴킷으로 쓰여진 어플리케이션에서는 불가능한 일이다. 또한 요즘의 리눅스 데스크탑 환경은 몇 가지의 편이요소를 더 제공한다. 컴포넌트 아키텍처(KDE는 Kpart를 GNOME은 Bonobo component framework를 갖고 있다)가 그것인데 이것은 스프레드시트나 도표를 워드 프로세싱 문서 내에서 사용할 수 있게 하고 포괄적인(global) 프린트 편리성(Windows에서와 같은 프린팅 환경)을 제공하며 사용자가 어플리케이션에 덧붙여 상호작용하고 보완할 수 있는 스크립트 언어를 가능하게 해준다.

UNIX의 데스크탑 환경이라는 개념에서는 누구나 한 환경의 프로그램을 다른 환경에서 구동할 수 있다. 나는 GNOME 환경에서 Konqueror를 사용할 수 있고 KDE에서 Gnumeric을 사용할 수도 있다. 결국에는 이것이 단지 프로그램이라는 것이고 보다 넓은 개념에서 데스크탑이라는 것은 일관성을 갖는다는 것이기 때문이다. 따라서 여러분은 자신의 환경에 맞는 어플리케이션을 기대할 수 있다. 만약 어떤 한 어플리케이션을 다루어야 하는데 그것이 조금 다른 모습을 하고 있고 당신의 환경요소와 상호작용(interact)하지 않는다면 당신이 그것을 (환경에 맞게)바꿔 사용할 수가 있는 것이다.

11. X 시스템 생활의 하루

여기에서는 리눅스 시스템의 데스크탑 환경 하에서 전형적인 GNOME 세션이 어떻게 작동하는가를 예를 들어 설명할 것이다. X에서 동작한다면 다른 여타의 환경도 이와 비슷하다.

리눅스 시스템이 X를 시작하면 X서버는 적재되면서 그래픽 장치를 초기화하고 클라이언트로부터 정보요구(requests)를 기다린다. 처음으로 gnome-session이라는 프로그램을 시작하고 작업 세션을 설정한다. 이 세션에는 항상 열리는 프로그램과 그 화면상의 위치 등을 포함하고 있다. 다음으로 패널이 시작된다. 패널은 보통 화면 하단에 위치하고 이것은 윈도우 환경의 계기판과 같다. 이것은 우리가 프로그램을 구동할 수 있게 해주고 어떠한 프로그램이 돌아가는지 알려주며 작업환경을 제어할 수 있게 해준다. 그 다음으로 윈도우 매니저가 뜨는데 우리는 GNOME을 사용한다고 가정했으므로 여러 가지의 다른 윈도우 매니저를 사용할 수 있다. 여기서는 Sawfish를 띄웠다고 가정하자. 마지막으로 파일 매니저가 뜬다(gmc.Nautilus). 파일 매니저는 데스크탑 아이콘 표시를 다루는 일 등을 담당한다. 여기까지 이르면 GNOME 환경은 작업할 준비가 완료된 것이다.

지금까지의 모든 프로그램은 X서버에 연결된 클라이언트이다. 이 경우에는 X가 동일한 컴퓨터에서 구동되지만 , 이전에도 설명했듯이 꼭 그럴 필요는 없다.

이제 몇 가지 명령을 입력하기 위해 xterm을 띄우자. xterm 아이콘을 클릭하면 패널은 xterm 어플리케이션을 구동한다. 이것도 X의 또 다른 어플리케이션이므로 X서버에 연결하여 내용을 출력하여야 한다. X서버가 xterm을 위한 화면상의 공간을 할당하면 윈도우 매니저인 Sawfish로 하여금 그럴싸한 제목줄과 함께 윈도우를 장식하게 하고 화면상의 위치를 결정한다.

이번에는 브라우저를 띄우자. 우리가 Netscape 아이콘을 클릭하면 브라우저가 뜬다. 이 브라우저는 GNOME의 제공된 환경이나 툴킷을 사용하지 않는다는 것을 상기하기 바란다. 이것은 조금 다르게 보인다는 것을 알 수 있을 것이다. 그것은 여타의 것과는 다르게 환경과의 상호작용이 부족하기 때문이다.

다음으로 Gnumeric 스프레드시트를 열어서 몇 가지 일을 한다. 조금 지난 후에 xterm에서의 작업을 위해 xterm을 클릭한다. Sawfish는 이것을 알아채고 윈도우를 관리하게 되며 xterm 창을 화면 최상위에 나타내고 작업을 할 수 있도록 포커스를 준다.

그리고 다시 스프레드시트로 돌아온다. 작업을 마저 끝내고 문서를 출력한다. Gnumeric은 GNOME 어플리케이션이므로 GNOME환경의 편리요소를 이용할 수 있다. 내가 출력명령을 내리면 Gnumeric은 gnome-print library를 호출하고 이것이 프린터와 상호 통신하여 내가 원하는 출력물을 찍어내는 것이다.

12. 저작권과 라이센스