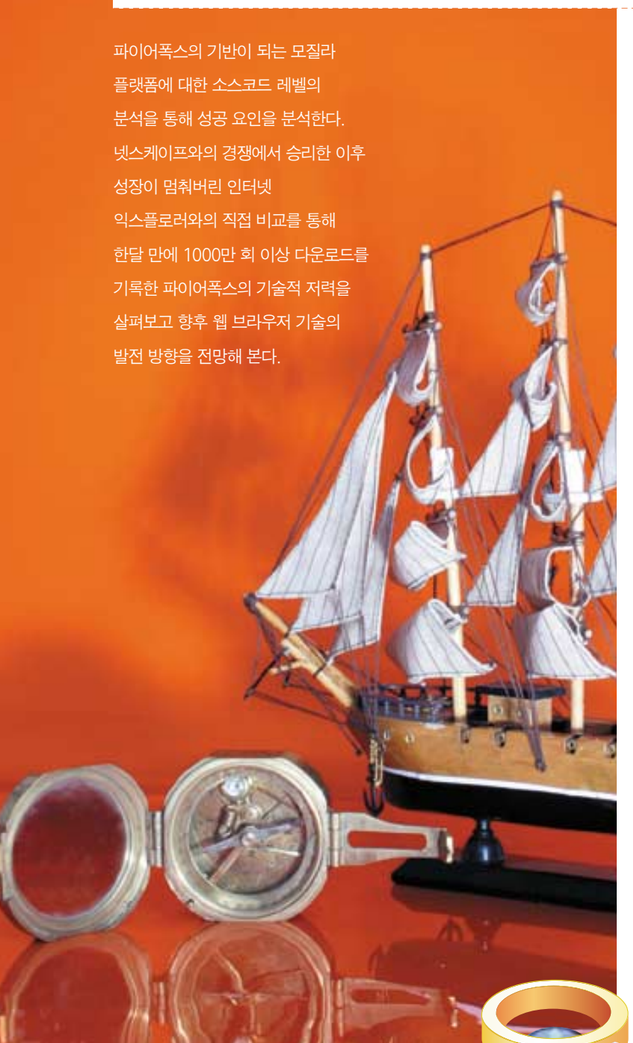
파이어폭스, 그 안에 숨은 더 큰 가능성

파이어폭스의 기반이 되는 모질라 플랫폼에 대한 소스코드 레벨의 분석을 통해 성공 요인을 분석한다. 넷스케이프와의 경쟁에서 승리한 이후 성장이 멈춰버린 인터넷 익스플로러와의 직접 비교를 통해 한달 만에 1000만 회 이상 다운로드를 기록한 파이어폭스의 기술적 저력을 살펴보고 향후 웹 브라우저 기술의 발전 방향을 전망해 본다.



윤석찬 | channy@daumcorp.com

현재 다음커뮤니케이션 R&D 센터에서 사내 기술전략을 수립하는 일을 하고 있으며 한글 모질라 포럼을 이끌고 있다. 웹 표준 기술과 보급에도 관심이 많으며 ZDNet과 블로그에 글쓰는 것이 취미다. 일용할 양식을 받을 수 있는 오픈소스 분야의 테크니컬 에반젤리스트를 꿈꾸고 있다.

신정식 | jshin@i18nl10n.com

유닉스와 야구와 '큰 사과'를 '사랑'하며, 평화(ㅍ), 사랑(ㅅ), 정의(ㅈ)와 같은 단어를 좋아한다. 재외 한국인이 컴퓨터와 인터넷에서 한글을 사용할 때 생기는 문제에 관심을 갖기 시작해 오픈소스 프로젝트의 다국어 지원과 국제화에 틈틈이 기여하고 있다. 'ㅍ, ㅅ, ㅈ'이 넘치는 세상을 꿈꾸고 있고 그를 실현하기 위해 일하고 있다.

파이어폭스가 기반으로 하고 있는 모질라는 넷스케이프 시절부터 본다면 10년이 넘는 세월 동안 수백 명의 개발자들의 손을 거쳐 수백 만행이 넘는 방대한 소스코드로 발전해 왔다.

그러나 넷스케이프 소스가 초기에 공개되었을 때만 해도 현재의 모질라와는 많이 달랐다. 소스코드 공개 후 1~2년간 렌더링 엔진으로서 게코(Gecko)를 선택하고, XUL(eXtensible User interface Language)을 기반으로 하는 UI를 개발하는 등 개발 로드맵의 중요한 부분을 결정했다. 또한 다양한 운영체제(OS)를 지원하면서도 OS 의존적이지 않기 위해 XPCOM이라는 별도의 컴포넌트 모델을 만드는 데 오랜 시간을 투자했다. 모질라가 단순한 웹 브라우저가 아니라 개발 플랫폼으로 조명 받는 이유도 OS 의존적이지 않은 기반 기술과 웹 표준 기술을 애플리케이션에 적절하게 조화시켜, 개발자는 소프트웨어를 쉽게 확장할 수 있고 사용자는 이런 확장기능을 맘껏 누릴 수 있는 환경을 제공하기 때문이다.

파이어폭스, 무엇이 다른가?

파이어폭스의 기반인 모질라 소스는 매우 방대하다. 이 소스코드는 모듈 형태로 구분해 각 모듈마다 소유자(owner)나 담당자(peer)가 정해져 있다. 이들은 모듈에 입력되는 대부분의 소스코드 수정과 추가 사항을 처리한다. 각 모듈은 100여 개가 넘으며, 핵심 모듈과 확장 모듈로 구분해 관리된다. 소스코드를 이루는 파일 형식은 대부분 컴파일 언어인 C++로 구성돼 있지만 네이티브 코드 내에 객체를 활용하는 데는 인터프리터(interpreter) 언어인 자바스크립트를 사용해야 한다. 자바스크립트는 웹 상에서 기능 구현이 쉬운 언어로서 동적인 데이터 처리가 가능한 장점이 있다. 그 밖에 메시지 파일 처리를 위한 DTD 파일, 외양을 결정하는 CSS 파일, UI를 결정하는 XUL 파일, 인터페이스를 정의하는 IDL 파일 등으로 구성돼 있다.

모질라가 컴파일돼 실행되면 C++ 컴포넌트가 실행되고 자바스크립트와 연결을 담당하는 XPConnect라고 불리는 기술을 사용할 수 있다. 그러나 이 자바스크립트는 내부 객체를 사용할 수 있다는 점에서 웹 페이지에 있는 자바스크립트와 구별된다. 후자는 웹 페이지 렌더링시 실행되며 내부 객체를 호출할 수 없도록 격리된다. 모질라 내부 코드에서 사용되는 자바스크립트는 사용자 인터페이스 상호 작용에 쓰이는 사용자 이벤트(event)를 처리하는 데 주로 이용된다.

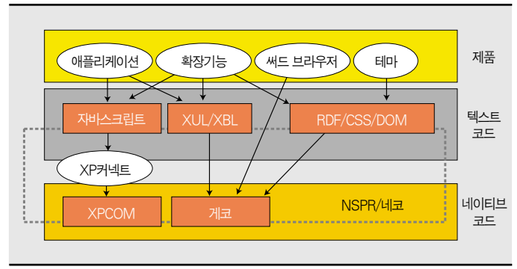
모질라의 소스코드는 철저하게 객체 지향 프로그래밍(Object Oriented Programming) 원칙을 따르며, 각 컴포넌트들은 철저히 모듈화돼 있다. 이들이 서로 통신할 때도 자체의 XP 환경 하에 컴포넌트 객체 모델(COM)에 기초한 잘 정의된 인터페이스만을 사용해 통신한다. 인터페이스의 주요 아이디어는 코바(CORBA)에서 왔으며, XPIDL(CROSS Platform Interface Definition Language)이라고 하는 언어를 이용해 사용한다.

모질라를 컴파일할 때는 이러한 인터페이스 정의(IDL) 파일을 자동으로 xpidl이라는 IDL 컴파일러를 통해 C/C++ 헤더 파일로 변환한다. 인터페이스는 UUID라는 식별 가능한 고유번호를 갖고 있으며, 이는 자바스크립트 코드에서 인터페이스에 접근할 수 있음을 의미한다. 그러나 이러한 활용은 자바스크립트 런타임 범위 내에서 유효한 파라미터를 위해 데이터 타입을 사용하는 경우로 한정돼 있다.

모질라에서 코드 실행에 실패하면 C++ 부분이 아닌 자바스크립트에서 예외 처리를 한다. 자바스크립트의 try-catch를 사용해 C++ 내의 반환값을 표시한다. 모질라는 타입 라이브러리와 실행 가능한 컴포넌트의 내부 레지스트리와 인터페이스에 종속돼 있다. 애플리케이션이 시작되면 레지스트리의 최신성을 확인한 후 변경된 부분이 있으면 레지스트리를 갱신한다. 그밖에 객체 지향 프로그래밍에서 컴포넌트 간의 상태를 관찰해 응답하기 위한 nsIObserver를 이용해 내부 메시지 전달 체제를 갖추고 있는 것도 특징이다.

모질라 소스코드에는 텍스트 메시지를 소스에 하드 코딩할 수 없으며 별도의 메시지 파일에 넣어 관리와 지역화가 용이하도록 소스와 내용을 분리하고 있다. 이 메시지 파일은 실행시에 변수 등을 포함해 구현할 경우 Properties 파일에 기록하며 일반적인 메시지는 모두 DTD 파일에 정의해 XUL 파일에서 참조할 수 있다(이런 내용들은 3부에서 실제 확장기능을 개발해 보며 자세하게 살펴볼 예정이다).

<그림 1> 모질라의 기술 기반 구조



파이어폭스의 주요 기반 기술

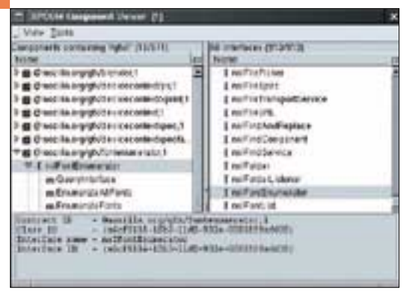
모질라에서 사용하는 기반 기술은 크게 형식에 따라 네이티브 코드와 텍스트 코드로 구별된다. 네이티브 코드는 컴파일되어 바이너리 행태로 존재하고 텍스트 코드는 컴파일 후에도 읽을 수 있는 것이 특징이다.

모질라 개발시 제 1원칙은 특정 OS에 제한되지 않는 크로스 플랫폼(XP, CROSS Platform) 소프트웨어여야 한다는 점이다. 이를 위해 모질라에는 <그림 1>과 같이 NSPR(Netscape Portable Runtime)이라는 라이브러리가 있다. NSPR은 원래 넷스케이프 제품군에 폭넓게 사용됐는데, 여러 운영체제 환경에서 독립적인 쓰레드(thread)와 동기화, 파일 입출력, 메모리 관리 등을 처리할 수 있는 함수들의 API이다. 특히 NSPR은 멀티 쓰레딩이 가능한 OS 독립적인 기능을 제공해, 데이터 전송을 하고 있는 동안에도 사용자 인터페이스가 응답할 수 있는 상태로 쓰레드 작업을 하는 것이 특징이다. NSPR은 모질라 소스의 ns/nspr에서 찾을 수 있으며 모질라 내부 프로젝트로 지속적으로 개발되고 있다(www.mozilla.org/projects/nspr/). 네코(Necko)라고 불리는 네트워크 라이브러리는 외부와의 통신과 캐시, 암호화, 인증 처리 등을 담당한다. 이밖에 모질라에 사용된 기술은 <표 1>과 같다.

운영체제를 넘나드는 비밀, XPCOM

XPCOM은 모질라를 기반으로 크로스 플랫폼 환경의 모듈 소프트웨어를 개발하는 플랫폼이다. XPCOM은 MS의 COM과 매우 유사하며 여러 종류의 애플리케이션 컴포넌트와 통신하기 위한 프록시 메커니즘을 제외하고는 거의 같다고 할 수 있다. 양쪽 모두 인터페이스 기반이며 모든 인터페이스를 세 개의 메쏘드(method), 즉 QueryInterface, AddRef, Release가 정의된 기본 인터페이스에서 만들 수 있다(그러나 이런 기본적인 공통점을 제외하면 MS의 COM과 XPCOM은 컴포넌트상 호환이나 데이터 교환이 사실상 불가능하다).

XPCOM의 각 인터페이스에는 향후 자바스크립트에서 참조할 수 있는 ContractID와 IDL을 통해 정의해서 사용하는 ClassID를 포함하고



<화면 1> XPCOM 뷰어로 본 COM 인터페이스

있으며 이를 통해 식별한다. XPCOM에서는 몇 가지 특징이 있는데 먼저 객체에서 메모리를 정확히 사용하고 반납하게 하기 위해 nsMemory라는 할당 및 반납 체계를 사용한다. 다음처럼 호출 받은 쪽에서 메모리를 할당하고 사용한 쪽에서 반납하는 규칙을 지켜야 한다.

```
void GetString(char **aResult) {
   Const original_message[]= "hello, world";
   char *message = (char*)nsMemory::Alloc(sizeof(char) *sizeof(original_message));
   strcpy(message, original_message);
   *aResult = message;
}
...
char *result;
GetString(&result);
Process(result);
nsMemory::Free(result);
```

이를 통해 메모리 리스크를 줄여 소프트웨어 오류를 줄인다. 이것은 각 객체들을 참조, 반납하는 데도 엄격하게 적용되는데 예를 들면 다음과 같은 방식을 사용한다.

```
void GetSample(nsISample **aResult) {
    *aResult = someobject;
    NS_ADDREF(*aResult);
}
```

<표 1> 모질라에 사용된 주요 기술

주요 기술	소개
XPCOM	운영체제에 영향을 받지 않도록 제작된 자체 컴포넌트 모델
XPConnect	XPCOM을 스크립터블하게 사용할 수 있도록 하는 인터페이스
XUL	UI를 동적으로 만들 수 있는 XML 기반 UI 정의
XPInstall	운영체제에 영향을 받지 않는 소프트웨어 설치 지원 프로그램
네코	외부와의 통신과 캐시, 암호화 등을 담당하는 네트워크 라이브러리
게코	각종 웹 표준과 XUL 등을 렌더링할 수 있는 그래픽 엔진

```
void doSomething() {
    nsISample *sample;
    GetSample(&sample);
    ProcessSample(sample);
    NS_RELEASE(sample);
}
```

그러나 참조와 반납에 드는 효율성 문제 때문에 각 인스턴스에 참조 횟수를 포인터로 저장해 사용한다. 인터페이스를 사용하는 모든 클래스는 참조 횟수 저장 기능과 자동 파기 기능을 갖춘 nsISupports라는 공통 기초 클래스를 공유하고 있다. 매번 이런 메쏘드를 호출해야 하는 번거러움을 줄이고 코드 가독성을 높이고 return과 같은 여러 출구에서도 객체 참조 반납을 쉽게 하기 위해 nsCOMPtr라는 COM 객체 포인터를 사용한다.

```
void doSomething() {
    nsCOMPTr<nsISample> sample;
    GetSample(getter_AddRefs(sample));
    ProcessSample(sample);
}
```

nsCOMPTr는 사용이 끝나면 객체를 자동으로 반납한다. C++ 코드에서는 이러한 규칙을 엄격하게 지키기 위해 nsCOMPTr을 항상 사용해야 한다. 그러나 이러한 동작은 자바스크립트 같은 스크립트 언어에서는 가비지 콜렉션이기 때문에 필요할 때 참조를 자동으로 줄여준다.

XPCOM의 또 다른 주요한 특징은 향후 자바스크립트에서 참조할 수 있도록 IDL로 인터페이스를 정의한다는 점이다. 먼저 CID를 기초로 주고 다음과 같은 IDL을 정의한다.

```
[scriptable, uuid(76f216a0-6c2f-11d9-9669-0800200c9a66)]
interface nsISample : nsISupports {
   attribute string value;
   void writeValue(in string aPrefix);
   void poke(in string aValue);
}
```

C++ 코드에서 IDL로 정의한 nsISample은 다음과 같이 구현한다. IDL 파일은 별도의 xpidl을 통해 컴파일해 xpt라는 헤더 파일을 생성한다.

```
class nsISample : public nsISupports {
   NS_IMETHOD SetValue(…) = 0;
   NS_IMETHOD GetValue(…) = 0;
   NS_IMETHOD WriteValue(…) = 0;
   NS_IMETHOD Poke(…) = 0;
}
```

이렇게 자바스크립트에서 만들어진 xpt 헤더 파일을 참조해 생성한 인터페이스는 XPCOM에서도 사용할 수 있다. XPCOM 인터페이스는 'nsI' 처럼 중간에 I를 넣게 되며 이를 구현하는 메쏘드는 I가 빠지고 Impl을 붙인다. 컴포넌트는 URI 방식으로 표현하고 레지스트리에 저장해 필요에 따라 설치한다. 만약 어떤 로컬파일이 있어 nsILocalFile에 제공되는 컴포넌트를 사용하면 '@mozilla.org/file/local;1'을 사용해 참조할 수 있다. 다음을 실행하면 파일 컴포넌트가 검색되어 aFile에 저장된다.

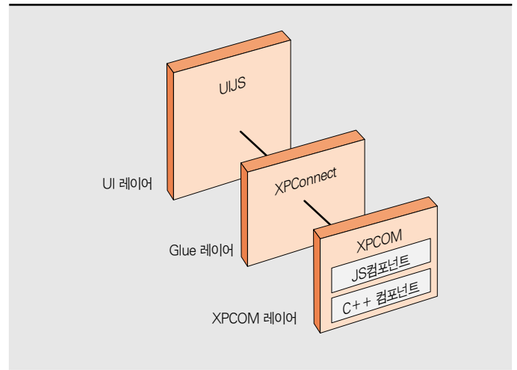
```
var aFile = Components.classes["@mozilla.org/file/local;1"].createInstance();
if (!aFile) return false;
aLocalFile=aFile.QueryInterface(Components.interfaces.nsILocalFile);
if (!aLocalFile) return false;
aLocalFile.initWithPath("/mozilla/testfile.txt");
aLocalFile.delete(false);
```

components는 컴포넌트에 관련되는 몇 개의 함수를 제공하는 일반적인 객체를 참조하고 classes 자산에서 컴포넌트를 취득한다. 함수를 호출하려면 인터페이스 중 하나를 호출한다. QueryInterface는 모든 컴포넌트에 제공되는 함수로 이를 사용하면 컴포넌트로부터 지정된 인터페이스를 취득할 수 있다.

XPCOM 인터페이스는 다른 인터페이스를 상속 받을 수 있다. 이 인터페이스는 그 안의 모든 함수를 가지게 되는데, 최상위 인터페이스인 nsISupports는 모든 인터페이스가 상속 받아 사용할 수 있다(nsISupports에 제공되는 함수가 바로 QueryInterface로, 모든 컴포넌트에서 사용할 수 있다). 단 어느 객체로 QueryInterface를 호출하든 그 객체가 필요로 하는 인터페이스가 지원되지 않을 경우 null 여부를 항상 확인해야 한다. 그 이후에 호출한 인터페이스를 이용해 특정 파일을 삭제할 수 있다.

XPCOM 컴포넌트 안에는 서비스라고 불리는 특수한 컴포넌트가 있다. 서비스는 글로벌 데이터를 취득하거나 설정, 다른 객체를 실행하는 역할을 하는데 일반적인 createInstance 대신 getService를 호출한다. 북마크가 대표적인 사례로 다음과 같이 하면 현재의 웹 사이트를 북마크에 추가할 수 있다.

<그림 2> XPConnect의 아교 역할




```
var bmarks = Components.classes["@mozilla.org/browser/bookmarks-service;1"].getService();
bmarks = bmarks.QueryInterface(Components.interfaces.nsIBookmarksService);
bmarks.AddBookmark("http://www.mozilla.org","Mozilla");
```

XPCOM은 오랫동안 심혈을 기울여 작업해 온 수천 개의 인터페이스로 구성돼 있으며 다양한 운영체제에서 사용할 수 있다는 점에서 모질라의 핵심 코드다. XPCOM에 대한 더 많은 정보는 www.mozilla.org/projects/xpcom에서 찾을 수 있다.

UI와 XPCOM의 중계자, XPConnect

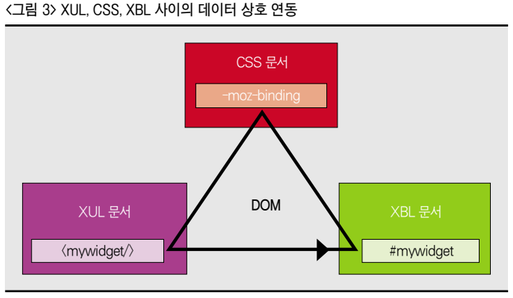
XPConnect는 이미 언급한 대로 자바스크립트로 XPCOM을 조작할 수 있는 기술이다. XPCOM 형식의 인터페이스 한쪽에서 통신하는 객체가 반대쪽 객체의 실행 언어에 영향을 받지 않게 함으로써 자바스크립트뿐만 아니라 어떤 언어에서도 참조할 수 있도록 한다(실제로 파이썬을 위한 pyXPCOM이라는 프로젝트가 진행중이며, 자바스크립트에 파이썬이나 Perl을 바인딩하여 사용할 수도 있다). 이를 위해 네이티브 코드에서 IDL 컴파일러(idlc)를 사용해 C 소스코드를 대량으로 생성한 후 JSAPI를 사용하는 자바스크립트 런타임 객체에 반영한다. 그 소스는 컴파일돼 모질라 바이너리가 된다. 이로써 훨씬 적은 코드로도 더 동적인 구현이 가능하다.

XPConnect는 MSCOM과 IDispatch 인터페이스를 지원하지 않는다. 오직 XPCOM의 규약을 지키고 XPIDL에서 선언된 인터페이스만 사용할 수 있다. 현재 자바스크립트 외에는 다른 언어를 지원하지 않으나 XPIDL 컴파일러가 자바스크립트 맵핑을 위해 C++ 고유 헤더를 통해 typelib를 생성하는 것은 가능하다. 누군가 컴파일러를 확장하면 다른 포맷을 생성하거나 다른 언어에 맵핑할 수 있으나, 현재는 자바스크립트만 지원한다.

<화면 2>
XUL을 이용한 다양한 모질라 테마

<화면 3>
XUL을 이용해
간단하게 만든 UI

<그림 3> XUL, CSS, XBL 사이의 데이터 상호 연동



다양한 테마를 지원하는 비밀, XUL

대부분 운영체제에서는 독자적인 그래픽 사용자 인터페이스(GUI)를 가지고 있으나, 모질라와 같은 XP(크로스 플랫폼) 프로그램은 OS 의존성을 제거하기 위해 독자적인 GUI 레이아웃을 정의했다. 무엇보다 기존의 GUI 라이브러리에서 사용하는 절대 위치를 지정하는 방식보다는 더 유연한 것이 필요했다. 모질라는 이를 위해 일단 UI는 논리적으로 특정 위치에 미리 디자인하고 여기에 글꼴을 맞추어 UI를 동적으로 생성하는 방법을 사용했다.

이것은 웹 브라우저가 웹 페이지를 표시하는 방법과 유사하다. 모질라는 웹이 그렇듯 독자적인 UI를 정의하기 위해 웹 표준인 XML 기반 문법에 따라 XUL을 만들었다. 현재는 UI 편집기가 별도로 없지만 사람들은 XML 문법만으로 직관적으로 메뉴나 윈도우 등을 설계할 수 있도록 지원한다. XUL은 브라우저의 외양을 매우 간단하게 바꿀 수 있는 기초가 됐다. 예를 들면 〈화면 3〉과 같이 윈도우 툴바에 File, Edit라는 메뉴를 넣은 UI는 다음과 같이 구현할 수 있다.

```
<toolbox flex="1">
   <menubar id="sample-menubar">
      <menu id="file-menu" label="File">
      <menupopup id="file-popup">
      <menuitem label="New" />
      <menuitem label="Open" />
      <menuitem label="Save" oncommand="Save();" />
      <menuseparator/>
      <menuitem label="Exit" oncommand="window.close();" />
   </menupopup>
   </menu>
   <menu id="edit-menu" label="Edit">
   </menu>
</menubar>
</toolbox>
```

XUL은 XHTML, MathML 또는 SVG와 같은 다른 XML 언어를 XUL 내에 삽입할 수 있으며 스타일시트(CSS)를 이용해 인터페이스의 외양을 수정할 수 있어서 스킨과 테마를 제작하는 데도 용이하다. 필요한 대부분의 UI 요소를 생성할 수 있으며 어떤 기기의 특별한 필요에도 맞출 수 있을 만큼 보편적이다. 대표적인 UI 요소는 다음과 같다.

- 텍스트 박스와 체크 박스와 같은 입력 컨트롤
- 버튼이나 다른 컨텐츠를 갖고 있는 툴바
- 메뉴 바 또는 팝업 메뉴상의 메뉴
- 탭 대화상자
- 구조적 또는 테이블화된 정보의 트리
- 키보드 단축키

XUL 파일에서 데이터를 제공해 그것으로 내용을 구성하는 것도 가능하다. 활용할 수 있는 데이터는 메일함, 북마크, 검색 결과는 물론 동적으로 생성된 데이터나 RDF 파일에서 제공된 데이터로 표현할 수도 있다. 이렇듯 XUL 파일은 메뉴, 윈도우, 툴바 같은 UI 요소 뿐만 아니라 사용자 이벤트에 대한 자바스크립트 함수를 정의할 수 있다. XPCOM 객체를 정의해 만든 자바스크립트도 호출할 수 있다.

또한 XUL은 로컬 파일이나 원격 사이트에서 호출해 사용하거나 다운로드해 설치할 수 있는 패키지로 배포할 수 있다. 대개 XUL 확장자를 가지는 파일에 저장되는데 다른 파일처럼 모질라로 열어 볼 수 있으며, 이때 정의된 UI가 동적으로 생성되어 나타난다. 원격 사이트에서 XUL 컨텐츠를 제공할 때는 웹 서버가 'application/vnd.mozilla.xul +xml' 형식으로 XUL 파일을 전송할 수 있도록 설정하면 된다.

한편 정해진 UI 대신 XBL(eXtensible Binding Language)을 사용하면 보다 동적인 UI를 구성할 수 있다. 〈그림 3〉은 XUL에서 고유하게 정의한 UI 요소를 CSS에서 표현하고 XBL로 정의해 사용하는 방법을 보여준다.

〈리스트 1〉은 nextback이라는 박스 클래스를 정의하고 이를 CSS를 통해 별도 XBL로 바인딩한 예제이다. nextback은 Next와 Back 버튼을 포함하고 있다.

이런 방법은 DOM을 이용해 객체를 동적으로 움직이고 싶을 때 사용한다. 예를 들어 슬라이드쇼와 같은 경우 Next, Back 버튼을 이용해 특정 DOM 객체를 동적으로 사용할 수 있다.

이처럼 XUL은 UI 분리에 효과적이고 여러 운영체제를 지원하며 지역화가 쉬운 장점이 있다. XML, RDF, CSS 등의 웹 표준 기술에 기반하고 있어 이미 개발자에게 친숙한 기술이므로 개발 시간도 크게 단축시킬 수 있다. XUL에 대한 더 많은 정보는 모질라 레이아웃 프로젝트(www.mozilla.org/newlayout) 혹은 XULPlanet(www.xulplanet.com)에서 찾을 수 있다.

모질라 네트워크 라이브러리, 네코

흔히 네코(Necko)라고 불리는 모질라 네트워크 라이브러리는 확장된 기능의 네트워크와 전송 기능을 제공하는 OS 독립적인 API다. URL을 인식해 http, ftp, file, chrome 등의 기본적인 프로토콜 통신을 수행한다. 특히 비동기적인 I/O를 지원하고 디스크 및 메모리 캐시, 비동기 DNS 캐시, 프록시 및 HTTPS 암호 통신 등의 역할도 한다. Necko의 구조는 〈그림 4〉와 같다.

특히 개인 보안 도구(Personal Security Manager), 네트워크 보안 서비스(Network Security Service) 등은 SSL을 기반으로 하는 표준 암호화 알고리즘과 인증서 관리 등의 기능을 모두 지원한다. 최근 한국소프트웨어진흥원에서 지원한 모질라 기반 인터넷 뱅킹 솔루션 프로젝트(project.oss.or.kr/LinuxPKI)에 따르면, 전자 서명용 국내 암호 표준인 SEED를 모질라의 NSS와 아파치 OpenSSL에 각각 삽입해 표준 SSL 통신을 통해 공인 인증 서명 데이터를 교환하는 데 성공했다고 한다.

모질라 애플리케이션 설치 필수품, XPInstall

XPInstall은 모질라 애플리케이션들을 운영체제와 상관없이 설치하고 확장, 업그레이드하는 데 필요한 기술이다. 이 기술은 XPI라고 부르는 ZIP 형식의 압축 파일을 자바스크립트 혹은 RDF 기반으로 설치하는 것이다. 브라우저 스킨이나 패치, 확장기능, 게코를 기반으로 하는 애플리케이션 등을 설치할 수 있다. XPInstall을 이용하면 버전 확인, 설치에 대한 로깅(Logging), 레지스트리 설정 업데이트 등의 작업을 할 수 있어 개발한 소프트웨어를 쉽게 배포할 수 있다. 다음은 xpinstall에서 사용되는 주요 함수이다.

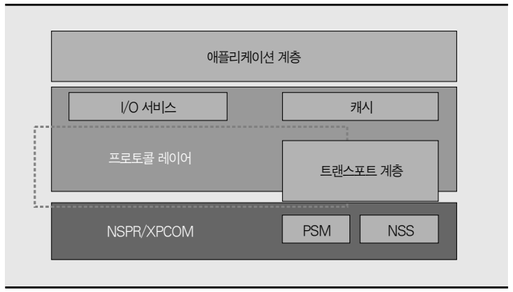
```
// Register chrome
registerChrome(PACKAGE | DELAYED_CHROME, getFolder("Chrome","xmlterm.jar"), "content/xmlterm/");
registerChrome(SKIN | DELAYED_CHROME, getFolder("Chrome","xmlterm.jar"), "skin/modern/xmlterm/");
registerChrome(LOCALE | DELAYED_CHROME, getFolder("Chrome","xmlterm.jar"), "locale/en-US/xmlterm/");
if (getLastError() == SUCCESS)
   performInstall();
else {
   alert("Error detected: "+getLastError());
   cancelInstall();
}
```

자바스크립트로 작성된 install.js에서 registerChrome에 설치할 파일을 읽어 특정 디렉토리에 설치하도록 지원한다. XPInstall에 대한 자세한 정보는 www.mozilla.org/scriptable을 참고하면 된다.

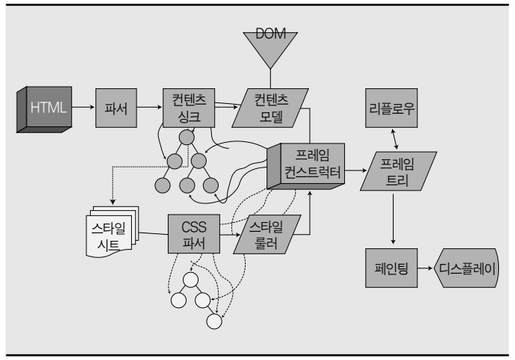
〈리스트 1〉 XUL, CSS, XBL을 이용한 UI 만들기

```
XUL (example.xul):
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<?xml-stylesheet href="chrome://example/skin/example.css" type="text/css"?>
<window
   xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
   <box class="nextback" />
</window>

CSS (example.css):
box.nextback {
   -moz-binding: url('chrome://example/skin/example.xml#nextback');
}

XBL (example.xml):
<?xml version="1.0"?>
<bindings xmlns="http://www.mozilla.org/xbl"
      xmlns:xul="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
   <binding id="nextback">
   <content>
      <xul:button label="Back" />
      <xul:button label="Next" />
   </content>
   </binding>
</bindings>
```

<그림 4> 네코 라이브러리의 기본 구조



<그림 5> 게코 내부의 HTML 데이터 처리 모식도



렌더링 엔진, 게코

게코는 모질라 플랫폼에 기반이 되는 렌더링 엔진이다. 게코는 1998년 넷스케이프 소스가 공개된 이후 기존의 마리너 렌더링 엔진을 다시 개발한 것으로, 광범위한 웹 표준 기술 즉, HTML, XHTML, CSS, DOM, XML, RDF 등을 표시할 수 있는 기반 플랫폼이다. 현재 게코를 비롯해 MS의 트리덴트, 애플의 KHTML, 오페라의 렌더링 엔진 등이 나와 있다. 〈표 2〉와 같이 이 엔진들에 대한 기본 비교를 보면 크로스 플랫폼 엔진이 적용 비용이 저렴한 것은 물론 여러 가지 측면에서 우수하다는 사실을 알 수 있다.

실제로 이 때문에 게코는 에피파니(Epiphany), 뷰(Nvu, HTML 편집기), 갈레온(Galeon), 카미노(Camino) 같은 웹 브라우저의 기본 렌더링 엔진으로 탑재돼 있다. 모든 개발 환경에서 쉽게 임베딩할 수 있으며 가능한 크기를 줄여 메모리 점유율이 낮고 속도를 빠르게 하는 최적화 작업을 거쳐 왔다. 이러한 중요한 기술적 진전을 통해 파이어폭스라는 작고 빠른 브라우저를 선보일 수 있었고, 최근에는 노키아에서 지원하는 미니모(Minimo)라는 모바일 브라우저까지 이어지고 있다.

이제 게코를 통해 데이터가 실제로 어떻게 처리되는지 알아보자. 인터넷이나 로컬 파일로부터 HTML 데이터가 게코로 들어오면 HTML 파서가 구문 분석을 시작한다. 컨텐츠 모델(contents model)이 분석

<표 2> 각 렌더링 기술에 대한 비교

	게코	트리덴트	KHTML	오페라
오픈소스 여부	○		○	
브라우저 확장	+++	+	++	+
표준 기술	W3C	MS	W3C	W3C
메모리 점유율	3~40M	30~100M	6~10M	3~13M
플러그인 지원	++	+++	+	+
플러그인 기술	XPCOM	액티브X	NS플러그인	NS플러그인
UI 기술	+++	+++	+	+
기술 적용 비용	무료	OS에 부과	무료	39달러
크로스 플랫폼	○			○

된 데이터를 큰 트리 구조로 늘어 놓는데 이 구조는 W3C 문서 객체 모델(DOM)에 근거하고 있다. 그리고 CSS 프레임 구조체(frame constructor) 데이터를 프레임 내에 놓는다. 이것은 HTML의 프레임이 아니라 DOM의 요소가 표시되는 추상적인 박스로, 여기서도 프레임 트리가 만들어져 데이터를 표시하기 위해 그림과 글자의 위치를 계산한다. 실제 화면 상의 위치는 CSS 규칙에 따라 계산되는데 이때 모니터 화면과 프린터가 다른 '표현(presentation)' 모드를 가지게 된다.

〈그림 5〉는 이러한 일련의 과정을 도식화한 것이다. 위치 계산이 계속 되는 동안 리플로우(reflow) 과정을 통해 새 정보가 시스템 안으로 계속 유입되는데, 시스템 트리를 변경시킨 항목은 '조잡(dirty)'이라고 표시하고 재작업해 완료(clean)될 때까지 계속 항목을 처리한다. 프레임 트리 안의 모든 아이템은 그 컨텐츠 모델 안에 일치하는 아이템으로 되돌리는 포인터를 가지고 있다. 즉 특정 요소를 보이지 않은 상태에서(hidden) 보이도록(visible)할 때 DOM API를 사용해 컨텐츠 모델 안의 변경이 프레임 트리에서 같은 변화를 가져온다.

다음 단계는 표시 관리자(view manager)로, 작은 예외 사항을 처리하거나 프레임 구조체가 그림을 읽고 처리하는 기본 과정이다. 디스플레이에 데이터를 표시하는 데는 GFX나 WIDGET 같은 그래픽 엔진을 이용한다. 모질라 소스에서 컨텐츠 모델의 코드는 /mozilla/content 폴더를, 프레임 구조체와 CSS 리플로우 코드는 mozilla/layout, 표시 관리자 코드는 /mozilla/view, DOM API 코드는 /mozilla/dom 폴더를 참고하면 된다.

게코로 간단한 애플리케이션 만들기

게코는 기본적으로 여러 가지 개발 환경에 임베딩해 자체적인 브라우저를 개발할 수 있다. 게코를 이용해 새로운 웹 브라우저를 만들 때 일반적으로 사용하는 몇 가지 인터페이스가 있는데, 이 가운데 게코 초기화와 종료에는 초기화 함수(NS_InitEmbedding)와 셧다운 함수(NS_TermEmbedding) 등 2개의 C++ 함수가 사용된다.

초기화 동안에 nsIWebBrower 인터페이스를 사용하면 전형적인 브라우저의 윈도우를 표시하면서 전체 프로그램의 chrome과 연결시킬 수 있고 DOM 객체를 얻는 데도 사용할 수 있다. nsIWebBrowserSetup은 브라우저가 열릴 때 그림을 표시할지 여부 등과 같은 일반적인 설정을 하는 데 사용하며 nsIWebNavigation은 URI 방문 기록에서 앞 또는 뒤로 가는 제어를 담당한다. nsIBaseWindow는 윈도우와 기본적인 명령(크기, 위치 , 타이틀 검색 등)을 표현하며 nsISHistory는 방문 기록을, nsIWebBrowserFind는 검색 및 실행을 제어한다.

nsIWebBrowserChrome은 게코 웹 브라우저의 가장 기초 인터페이스로 nsIWebBrowser를 이용해 웹 브라우저로 만들 수 있는 필수 인터페이스이다. nsIEmbeddingSiteWindow는 게코에 윈도우 크기를 변경하거나 숨기거나 윈도우 타이틀 등을 변경할 때 사용하며 nsIContextMenuListener는 사용자가 마우스 오른쪽 버튼을 클릭했을 때 나타나는 메뉴를 알고 싶을 때 사용하는 인터페이스이다.

게코의 내부

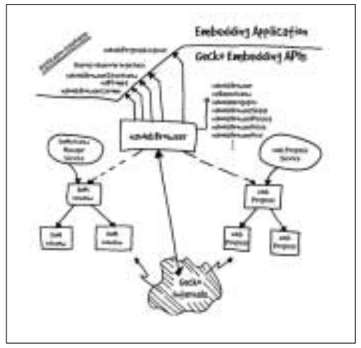
이제 윈도우의 MFC를 이용해 게코를 사용하는 코드를 살펴보자. 이 소스코드는 이미 모질라 테스트 코드에 포함돼 있으며 다른 운영체제나 개발 환경에 대한 예제도 소개돼 있으니 참고하기 바란다(lxr.mozilla.org/seamonkey/source/embedding/tests/mfcembed/).

게코를 사용하기 전에 먼저 임베딩 레이어를 초기화해야 한다. 그러면 XPCOM이 시작되고 컴포넌트가 등록돼 글로벌하게 사용할 수 있다. 임베딩 레이어는 다음과 같이 작업 디렉토리를 가르키는 nsnull과 프로필 파일 위치를 지시하는 provider 파라미터로 호출할 수 있다.

```
nsresult rv;
rv = NS_InitEmbedding(nsnull, provider);
if(NS_FAILED(rv))
{
   ASSERT(FALSE);
   return FALSE;
}
```

임베딩한 BrowserView 객체는 CreateBrowser 메쏘드를 호출한다. 각 브라우저 객체는 단일 브라우저 창을 대표한다. 브라우저 인스턴스를 생성하기 위해 생성한 BrowserView 객체는 메쏘드 Create

<그림 6>
게코 임베딩 API를 이용한 애플리케이션 개발 구조



Browser를 호출한다. 각각의 브라우저 객체(webbrowser)는 단일 브라우저 윈도우를 표현하며 특히 유틸리티 명령 do_Create Instance와 매크로 사용에는 주의해야 한다.

```
HRESULT CBrowserView::CreateBrowser()
{
nsresult rv;
mWebBrowser = do_CreateInstance(NS_WEBBROWSER_CONTRACTID, &rv);
if(NS_FAILED(rv))
return rv;
```

한번 nsWebBrowser 객체가 만들어지면 그 애플리케이션은 nsIWebNavigation 인터페이스 포인터를 mWebNav 멤버 변수로 읽어 오기 위해 do_QueryInterface를 사용한다. 이것은 웹 페이지 네비게이션에서 사용된다.

```
rv = NS_OK;
mWebNav = do_QueryInterface(mWebBrowser, &rv);
if(NS_FAILED(rv))
return rv;
```

그 다음 CBrowserImpl 객체가 만들어지고 개발자는 게코가 애플리케이션과 통신 가능한 여러 인터페이스를 구현해야 한다. 샘플에 있는 CBrowserImpl은 그러한 인터페이스를 요구하는 객체로 구현되어야 하고 SetContainerWindow()로 호출될 때 사용된다.

```
mpBrowserImpl = new CBrowserImpl();
if(mpBrowserImpl == nsnull)
return NS_ERROR_OUT_OF_MEMORY;
```

mWebBrowser 인터페이스 포인터는 CBrowserImpl 메쏘드를

통해서 받는다. 플랫폼에 의존하는 BrowserFrameGlue 인터페이스에 2개 포인터도 건네받아 보존한다. BrowserFrameGlue 포인터에 의해 CBrowserImpl이 상태 바나 진행 바 등을 갱신하기 위해서 메쏘드를 호출할 수 있다.

```
mpBrowserImpl->Init(mpBrowserFrameGlue, mWebBrowser);
mpBrowserImpl->AddRef();
```

이제 개발자는 다음과 같이 webbrowser와 관련된 chrome 객체를 연결할 수 있다.

```
mWebBrowser->SetContainerWindow
   (NS_STATIC_CAST(nsIWebBrowserChrome*, mpBrowserImpl));
nsCOMPtr<nsIWebBrowserSetup>setup(do_QueryInterface(mWebBrowser));
if (setup)
   setup->SetProperty(nsIWebBrowserSetup::SETUP_IS_CHROME_WRAPPER,PR_TRUE);
```

이제 다음과 같이 웹 브라우저 윈도우를 만든다.

```
rv = NS_OK;
mBaseWindow = do_QueryInterface(mWebBrowser, &rv);
if(NS_FAILED(rv))
return rv;
```

만든 웹 브라우저를 다음 코드에서 기본적인 위치를 받는다.

```
RECT rcLocation;
GetClientRect(&rcLocation);
if(IsRectEmpty(&rcLocation))
{
   rcLocation.bottom++;
   rcLocation.top++;
}
rv = mBaseWindow->InitWindow(nsNativeWidget(m_hWnd),
      nsnull,0, 0, rcLocation.right - rcLocation.left,
      rcLocation.bottom - rcLocation.top);
rv = mBaseWindow->Create();
```

m_hWnd는 이와 같은 호출을 통해 InitWindow()로 옮겨진다. CBrowserView는 CWnd로부터 m_hWnd를 상속 받고 임베딩 브라우저의 기본 창으로 사용된다. BrowserImpl 객체는 nsIWebProgressListener에 더해지고 진행 메시지를 받아들일 준비가 된다. 이 callbacks 정보들은 상태 바나 진행 바에 정보를 업데이트하게 된다.

〈화면 4〉
LXR 웹사이트

```
nsWeakPtr weakling
   (dont_AddRef(NS_GetWeakReference(NS_STATIC_CAST(nsIWebProgressListener*, mpBrowserImpl))));
void mWebBrowser->AddWebBrowserListener(weakling, NS_GET_IID(nsIWebProgressListener));
```

이제 마지막으로 웹 브라우저 윈도우를 표시한다.

```
mBaseWindow->SetVisibility(PR_TRUE);
```

```
nsCOMPtr<nsIWebBrowserPrint> print(do_GetInterface(mWebBrowser));
if (print)
   {
   print->GetNewPrintSettings(getter_AddRefs(m_PrintSettings));
   }
return S_OK;
```

웹 페이지 방문 기록을 탐색하기 위해서는 nsIWebNavigation 세션 기록에 저장된 정보를 다음과 같이 사용할 수 있다.

```
void CBrowserView::OnNavBack()
{
if(mWebNav)
   mWebNav->GoBack();
}
```

게코 엔진의 임베딩 예제가 담긴 소스 디렉토리에는 MFC 뿐만 아니라 Win32와 매킨토시, OS/2에서 임베딩하는 소스도 포함돼 있다. 또한 앞서 언급한 모바일 기반의 웹 브라우저인 미니모의 소스코드(lxr.mozilla.org/seamonkey/source/embedding/minimo/)도 참고하기 바란다.

소프트웨어 공학으로서 모질라

모질라 프로젝트는 다수의 개발자가 공동으로 작업하는 개발 시스템이다. 이러한 거대한 프로젝트를 움직이는 데는 필연적으로 각종 규칙과 지원 시스템이 갖추어져 있다. 특히 소스의 열람과 수정이 매우 자유로운 모질라가 다수의 의견을 종합해 올바른 코드로 나가게 하기 위해 사용하는 방법론은 개발 프로젝트를 진행하는 모든 개발자들이 참고할 만하다.

모질라의 소스코드 수정에는 두 단계의 코드 리뷰 규칙이 있다. 먼저 수정된 패치나 소스가 버그질라를 통해 제출되었을 때, 각 모듈의 소유자가 첫 번째 리뷰를 한다. 이 과정에서 허가가 나면 모질라 프로젝트를 움직이는 사람들로부터 슈퍼 리뷰라는 단계를 거친다. 양쪽의 리뷰를 받고 나면 대부분의 코드는 소스 트리에 체크인된다. 많은 체크인이 수행되고 있기 때문에 모질라는 코드 수정에 따라 애플리케이션이 동작하지 않거나 컴파일하지 못하는 상황에 발생할 수 있다. 소스 트리에 문제가 생기면 체크인은 중단되고 문제 해결에 집중한다.

이러한 소스코드는 몇 주에 한번씩 새로운 스냅샷(snapshot)을 만들어 버그를 수정하게 된다. 개발 로드맵에 따라 제품을 출시해야 할 때는 며칠 동안 드라이버(driver)라고 불리는 몇몇 사람들의 승인을 받아야만 수정할 수 있다. 드라이버들은 소스코드가 목표로 한 개발 로드맵에 따라 제대로 준비되었는지 소스의 상태를 점검하고 일정을 만드는 작업을 한다. 모질라의 제품 출시는 소스코드와 압축 버전, 설치 버전 등으로 나누어 제공되는데, 개발 로드맵에 따라 〈표 3〉과 같은 출시 사이클을 가지고 있다. 특히 일일 버전을 통해 하루 단위로 수정된 사항을 확인하는 버전을 내고 있다.

거대한 프로젝트를 조직적으로 움직이기 위해 필수적으로 모질라 프로젝트에서는 차용되거나 직접 개발하여 사용하고 있는 지원 시스템들이 존재한다. 이들 중 일부는 소스가 공개돼 소프트웨어 개발 프로젝트에 도움을 주고 있다.

〈표 3〉 모질라의 제품 출시 사이클 도표

제품 출시 형식	설명
릴리즈 제품	
소스코드	CVS로 배포
ZIP/Tar.gz(바이너리)	컴파일로 압축 파일 형식(모질라의 경우, 14MB 이하)
인스톨러(네트워크/전체설치)	모질라의 경우 네트워크 설치 1MB 이하, 전체설치 14MB 이하
RPM/디스크 이미지/포트	타 OS를 위한 컨트리뷰션 버전
릴리즈 사이클	
알파(1.0a)	버그 수정과 새 기능에 대한 테스트
베타(1.0b)	버그와 새 기능 탑재 및 지역화 메시지 고정
릴리즈후보(1.0 RC1)	소프트웨어 안정성 검증
릴리즈(1.0, 1.0.1)	완성 버전
나이틀리 빌드	
20040508	매일 단위로 수정된 사항을 확인하는 버전
언어팩	
en-US.jar	DTD, Properites 파일로 메시지 처리
en-win.jar, en-mac.jar, en-unix.jar	각 OS별 메시지 처리

모질라의 소스 관리, CVS

모질라 프로젝트에서는 소스 관리를 위해 CVS를 사용하고 있다. CVS는 버전 관리가 가능하고 소스를 분기해 개발할 수 있다. 로드맵에 따라 제품이 출시될 때는 기존 소스코드에서 분기하는데, 이는 원래 소스코드는 개발을 계속 진행하고 출시를 위한 소스코드가 새로 작업에 들어갈 수 있기 때문이다. CVS는 동시 작업이나 전 세계적인 접근이 가능하기 때문에 많이 사용되는 형상 관리 도구이다.

모질라 소스코드에 대한 검색엔진, LXR

LXR은 모질라 소스코드를 보기 위한 검색 사이트(lxr.mozilla.org)이다. 트리 구조의 내용을 클릭하는 것으로 소스코드를 볼 수 있으며 디렉토리 별로 일, 주, 월 단위의 변경 이력도 바로 확인할 수 있다. LXR의 소스코드는 리눅스 Glimpse 엔진으로 나온 오픈소스로 이를 모질라 프로젝트 내부적으로 수정해서 사용하고 있다.

코드 수정 내역을 추적하는 본사이

본사이(bonsai.mozilla.org)는 소스코드에서 누가 언제 어떤 코드를 수정했는지 그 내역을 볼 수 있는 웹 사이트이다. 변경 이력을 문의할 수 있는 강력한 인터페이스도 제공한다.

운영체제별 자동 컴파일 시스템, 틴더박스

모질라는 완성된 소스코드를 여러 운영체제에 따라 자동으로 컴파일하는 시스템을 갖추고 있다. 틴더박스는 컴파일 과정과 결과에 대해 보고해 주는 웹 사이트이다. 컴파일이 끝났을 때 프로그램이 올바르게 동작하는지 체크하기 위한 몇 가지 자동 테스트가 실행되는데 이러한 테스트의 결과를 틴더박스에서 볼 수 있다. 틴더박스는 세로축이 시간이며 가로축이 OS를 나타내므로 시간에 따라 어떤 변화가 있었는지 바로 알 수 있다.

테스트 결과는 색상으로 나타내는데 초록색은 문제가 없다는 것을 나타낸다. 노란색은 컴파일중인 경우, 오렌지색은 컴파일과 빌드는 끝났지만 자동 테스트에서 실패한 것을 나타낸다. 붉은색은 소스코드 컴파일이 성공하지 않은 것을 나타내며 개발이 중단됐음을 의미한다. 이 기능은 매우 중요한 것으로, 소스코드가 변경된 후에 나타나는 결과를 통해 문제를 쉽게 진단할 수 있게 해 준다. 컴파일이 실패한 경우 실패 로그를 남겨 주고 이와 관련된 체크인 항목도 표시된다.멀티 플랫폼 환경에서 일일 버전을 만들어 내는 프로젝트의 특징상 꼭 필요한 개발 도구이다.

웹 기반 버그 추적 시스템, 버그질라

버그질라는 웹 기반의 버그 추적 시스템이다. 제품에 어떤 문제가 발

〈화면 5〉
본사이 웹 사이트

〈화면 6〉
틴더박스 웹 사이트

생했을 때 사용자는 언제라도 버그를 제출할 수 있다. 버그가 제출되면 번호가 발급되고 문제를 해결해야 하는 사람에게 전달된다. 문제 해결을 위해 코드상의 문제 해결 방법을 알고 있으면 패치를 첨부할 수 있다. 패치가 첨부되면 리뷰 요청(?), 체크인 허가(+), 체크인 금지(-) 등의 방법으로 리뷰와 슈퍼 리뷰를 거칠 수 있다.

이 때 버그란 단지 소프트웨어 내의 에러만을 포함하는 것이 아니다. 특정한 프로세스를 거쳐야 하는 경우나 기능 확장을 요구하는 경우 모두 버그질라를 이용할 수 있다. 버그질라는 모질라 프로젝트에서 간단한 버그 리포팅 도구로 개발됐으나, 소스가 공개돼 있어 현재 IBM, 레드햇 등 주요 IT 기업의 버그 리포팅 도구로도 사용되고 있다.

롱혼은 모질라 플랫폼을 닮았다?

하드웨어와 소프트웨어 시장이 급격한 변화를 겪고 있는 지금 MS는 그 동안 GDI 기반 Win32 API와 NTFS, 그리고 COM/DCOM 및 다이렉트X에 기반해 왔지만 이러한 모델에서 롱혼(Longhorn)이라는 획기적인 모델로 대체하겠다고 밝혔다. 롱혼은 크게 다음과 같은 특징을 가지고 있다.

- ◆ **아발론(Avalon)** : 스크린 레이아웃 엔진
- ◆ **인디고(Indigo)** : 고급 수준의 네트워크 라이브러리
- ◆ **윈FS(WinFS)** : 새로운 형태의 파일 시스템
- ◆ **닷넷(.NET)** : COM을 대체할 만한 자바 대응 객체 프로그래밍

MS가 차세대 플랫폼으로 내세운 롱혼은 모질라 플랫폼에 대한 반응이라고도 볼 수 있다. 실제로 아발론은 모질라의 게코 모델과 유사하다. XUL이라는 XML 기반 사용자 인터페이스를 본 따 XAML이라는 표준안을 지원하려고 하고 있다. 인디고 역시 모질라의 네코와 닮은 점이 많다. 윈FS는 모질라에 대응할 만한 플랫폼이 없지만 실제로 MS내에서도 윈FS의 실체에 대해 회의적인 것으로 알려져 있다. 모질라는 닷넷을 지원하지는 않지만 XPCOM이라는 플랫폼을 가지고 있다. 롱혼에서 데이터 표준으로 많이 사용할 예정인 XML 스키마 형식은 이미 모질라에서 RDF와 오버레이 시스템으로 사용되고 있다. 모질라 플랫폼이 몇 년간 주요 웹 기술을 통합 발전시켜 오고 있다는 점을 고려하면 MS의 롱혼 행보는 이에 대한 추격으로 봐도 무방할 것 같다.

〈표 4〉는 윈도우를 표시하는 UI 코드와 모질라를 비교한 것이다. 네임스페이스와 태그는 다르지만 형식과 아이디어는 같다고 볼 수 있다. 개발 코드에서도 유사성을 보이고 있다. 인디고와 네코에서 사용하는 네트워크 스크립팅 코드를 보면 알 수 있다. 〈표 5〉를 보면 XAML 이벤트 핸들러를 처리하는 부분은 제외했지만 Jscript.Net과 네코의 코드가 거의 유사하다는 것을 알 수 있다. 네코의 코드에는 SOAP 객체를 for loop로 불렀지만 인디고의 경우 SOAP를 기본 메시지 문법으로 갖고 있기 때문에 그 소스 모양이나 객체 사용법 등은 거의 같다. 간단한 코드만을 비교해 거대한 운영체제 프로젝트인 롱혼을 모질라와 똑같다고 단정할 수는 없다. 그러나 적어도 롱혼에 들어가 있는 아이디어는 모질라에서 이미 구현돼 성공적으로 사용하고 있는 것들을 많이 차용했음을 알 수 있다.

실제로 롱혼의 프로토 타입이 제대로 세상에 나오지는 않았지만 아발론의 벡터 렌더링은 사실 플래시나 쇽웨이브 등과 비교해 보면 새로운 기술이 아니다. XAML 뿐만 아니라 Laszlo 서버의 경우 플래시를 기반으로 XML 인터페이스 벡터 그래픽 엔진을 이미 선보인 바 있기 때문이다. 이에 비해 모질라 진영에서 이미 70% 가량 구현한 웹 표준 벡터 그래픽 기술인 SVG를 기존의 플랫폼에 장착한다면 더욱 강력한 플랫폼이 될 것이다.

〈표 4〉 아발론과 모질라의 코드 비교

구성	내용
아발론에 사용될 XAML 코드	〈Window xmlns="http://schemas.microsoft.com/2003/xaml"〉 〈TextPanel〉 Some Text〈Button〉Press Me〈/Button〉 〈/TextPanel〉 〈/Window〉
모질라에서 사용중인 XUL 코드	〈window xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"〉 〈description〉 Some Text〈button〉Press Me〈/button〉 〈/description〉 〈/window〉

모질라 2.0을 향해

독립 브라우저인 파이어폭스와 전자메일 클라이언트 썬더버드가 큰 성공을 거두면서 통합 인터넷 플랫폼으로 개발됐던 모질라가 어떻게 바뀔지 기대를 모으고 있다. 먼저 모질라는 현재 형태로 계속 개발될 것으로 보인다. 다만 사용자 인터페이스나 기능의 변화 없이 내부 기반 기술을 계속 축적하는 그야말로 개발 플랫폼(developer platform)의 모양을 띨 것이다. 현재 모질라는 1.8a6 버전이 출시된 이후 계속해서 개선 작업이 진행되고 있다.

모질라 2.0의 주요한 목표는 그래픽 기능을 개선하는 것이다. 현재 사용하고 있는 GFX를 카이로(Cairo)로 대체할 계획을 갖고 있다. 카이로(cairographics.org)는 오픈소스 벡터 그래픽 엔진 개발 프로젝트로서 미려한 그래픽 표현이 가능해 사용자 인터페이스의 혁신을 가져올 것으로 기대된다. 또한 그 동안 진척되어 왔던 SVG를 완벽하게 구현해 각종 벡터 그래픽 포맷을 플러그인 없이 브라우저에 바로 표현할 수 있을 것이다.

특히 2005년에는 XUL을 기반으로 하는 애플리케이션을 개발하기 위해 경량의 XUL Loader(Runner)을 개발하고 libxul.so 형태의 라이브러리를 배포해 개발자들이 쉽게 XUL 애플리케이션을 만들도록 할 계획이다. XUL2로 명명된 이 프로젝트는 템플릿 시스템, 싱글 파일 애플리케이션, X인터넷에 대응되는 원격 인터넷 애플리케이션 개발을 지원하게 될 것이다. 뿐만 아니라 애플의 사파리와 같은 드래그 앤 드롭 기술과 투명 및 비사각형 윈도우에 대한 구현에도 초점을 맞출 예정이다. 모질라와 오페라가 펼치고 있는 웹 애플리케이션 표준안에 따라 WebForm2와 W3C 표준안인 XForm을 구현하는 것도 올해의 목표이다.

모질라는 개발 플랫폼으로 많은 발전을 거듭해 왔지만 아직도 시작 단계에 서 있을 뿐이다. 넷스케이프가 비록 인터넷 익스플로러와의 시장 점유율 싸움에서는 졌지만, 그들이 소스를 오픈함으로써 결과적으로 많은 개발자들이 그로 인해 영감을 받아 기술을 선도할 수 있는 플랫폼으로 발전시키리라고는 생각하지 못했을 것이다. 현재 우리나라에도 많은 애플리케이션 개발자들이 있다. 단지 일로 하는 개발뿐만 아니라 기술을 발전시키고 진보시키는 도구로, 모질라 플랫폼에 관심을 가져 보기를 기대한다.

〈화면 7〉 버그질라 웹 사이트

〈화면 8〉
카이로를 이용한
벡터 데스크탑 화면

〈표 5〉 SOAP 요청을 보내는 코드 비교

구성	내용

Jscript.Net 코드	import System.MessageBus;
	var args, server, method, port, channel;
	args = { data : "Test Data" ; }
	server = new Uri("http://saturn/test");
	method = new Uri("http://saturn/test/test-call", args);
	port = new Port();
	port.Open();
	channel = port.CreateSendChannel(server);
	channel.send(method); // go.

네코 자바 스크립트	var args, server, method, soapcall, soapargs = {};
	args = { data : "Test Data" ; };
	server = "http://saturn/test" ;
	method = "/test-call" ;
	for (var i in args)
	soapargs.push(new SOAPParameter(i, args[i]));
	soapcall = new SOAPCall();
	soapcall.transportURI = server;
	soapcall.actionURI = method;
	soapcall.encode(0, method, null, 0, {}, soapargs.length, soapargs);
	soapcall.invoke(); // go.

정리 | **박상훈** | nanugi@imaso.co.kr