부록 E

공개소프트웨어 관련 연구

1. 스케줄러 선택기반의 실시간 리눅스의 성능분석

요 약

본 논문에서는 스케줄러 선택방식 기반의 실시간 리눅스 시스템에서 비율단조(RMS)와 마감시간우선(EDF) 중에서 사용자가 하나를 선택함으로서, 개선된 스케줄링 검사가 가능하고 태스크 특성에 맞는 스케줄링 알고리듬을 제안하였다. 스케줄러 선택방식의 성능분석을 위해 다양한 프로세서 이용률을 갖는 태스크의 평균 응답 시간과 마감시간에 따라 효율적인 태스크 스케줄링 방식의 성능을 분석하였다.

1.1. 서론

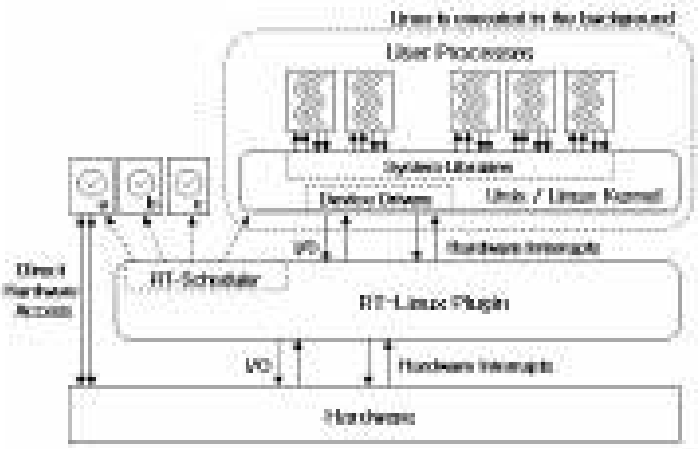
실시간 리눅스 시스템(RTOS)은 일반적인 시스템과는 달리 태스크의 수행결과의 정확도뿐만 아니라 태스크가 마감시간 내에 실행될 수 있는 지가 중요한 성능의 척도가 된다. 기존의 일반 운영체제의 스케줄링 방식은 시분할 환경에 적합하도록 고안 되었으며 스케줄러는 프로세스들의 우선순위를 주기적으로 재계산하여 각 프로세스들이 프로세서를 공평하게 공유할 수 있도록 설계하였다[1]. 그러나 실시간 시스템에서 이러한 비 실시간스케줄링 방법으로는 시간의 제약성을 만족시킬 수 없다. 실시간 시스템의 스케줄링은 시간 제약성과 정확성을 보장하기 위해서 우선순위에 따르는 선점 스케줄링이 필요하다[2]. 실시간 태스크들은 마감시간 내에 수행되는 것이 보장되고 수행이 될 수 있다는 것이 예측 가능해야 한다. 현재 실시간 리눅스를 위한 스케줄러로는 비율단조(RMS) 스케줄러와 마감시간우선(EDF)스케줄러 두 가지가 별도로 구현되어 있다. 이 두 가지 스케줄러 중에서 사용자는 두 가지방법 중 하나를 선택하여 사용하고 있다[3]. 이로 인해 실시간 시스템의 스케줄링 가능성검사의 미수행으로 마감시간 실

패율을 증가시키는 결과를 초래한다[4]. 또한 현재 실시간 리눅스에서는 스케줄 불가능한 태스크들을 스케줄 함으로서 시스템 자체가 멈춰버리는 정지현상이 발생할 수 있다. 이러한 현상은 실시간 시스템에서는 매우 치명적이다[5]. 본 논문에서는 이러한 현상을 고려하여 보다 유연성 있고 개선된 스케줄링 가능성 검사를 통하여 태스크의 특성에 맞는 스케줄링 방법을 선택하도록 하여 마감시간을 보장하고 효율적인 태스크 스케줄링 방법을 제안한다.

1.2. 실시간 리눅스에서 스케줄링 비교

1.2.1. 실시간 리눅스 구현

실시간 리눅스에서 실시간 태스크를 스케줄링 할 수 있는 메커니즘을 제공해 주고 있다. 리눅스가 실시간 기능을 갖추려면 실제로 커널부터 다시 설계되어야 하겠지만, 이는 커널을 구성하는데 많은 시간을 필요로 하며, 리눅스 커널이 업그레이드 될 때마다 이를 신속히 반영하기 어렵고, 버그가 생길 가능성이 더 커지게 된다. 이에 실시간 리눅스는 커널을 다시 설계하지 않고 리눅스 소스 변경을 최소화하는 방향으로 진행되었다. 리눅스 아래에 실시간 리눅스 커널을 올리고 기존 리눅스 커널을 실시간 리눅스 커널에서 돌아가는 하나의 프로세스로 만들어서 실시간 태스크와 기존의 리눅스 커널이 공존하는 형태를 만들었다. 실시간 리눅스 커널은 실시간 프로세스를 생성하고 이들을 스케줄링하며, 인터럽트가 발생하였을 때 이를 처리하며 리눅스와의 통신을 담당하는 역할을 한다. 기존 리눅스 커널은 일반 리눅스 실시간 리눅스는 [그림 1]과 같이 구성되어 있다[1].



[그림 1] 실시간 리눅스의 커널 구성

실시간 리눅스의 구현 특징은 다음과 같다[1].

⊙ 리눅스 커널 태스크 : 실시간 리눅스는 기존 리눅스 커널을 실시간 리눅스 상에서 돌아가는 하나의 태스크로 생각한다. 여기에는 가장 낮은 우선순위가 부여되므로, 실행할 수 있는 실시간 태스크가 하나도 없을 때 실행된다.

⊙ 스케줄링 : 실시간 리눅스는 현재 두 가지 스케줄링 방법을 제공한다. 하나는 우선순위 기반 선점 스케줄러로서 모든 태스크에 우선순위를 부여하고, 실행할 수 있는 태스크 중에서 가장 우선순위가 높은 태스크를 수행하며, 이보다 더 높은 우선순위를 갖는 태스크가 등장하면 이를 곧바로 수행한다.

다른 스케줄러는 마감시간 우선(EDF, Earliest Deadline First) 알고리듬을 사용하며, 우선순위가 아니라 마감시간을 기준으로 스케줄링 한다[2].

1.2.2. 실시간 스케줄링과 선택알고리즘 비교

스케줄링 측면에서 주어진 태스크 집합에 대하여 마감시간을 만족할 수 있는 지의 여부를 조사하는 것을 스케줄링 가능성 검사라고 한다. 만약 주어진 태스크 집합의 모든 태스크들이 마감시간 내에 실행을 완료할 수 있다면 그 태스크 집합은 스케줄링 가능하다고 한다. 스케줄링 가능성 검사를 예측하기 위해서는 간단한 수식을 통하여 판단 가능한 분석적 방법을 많이 사용한다. 분석적 방법을 통한 스케줄링 가능성 검사는 우선순위 구동 방식의 알고리듬들에 유용하게 사용할 수 있는 기법이다. 시스템 설계자는 설계 당시 시스템에 적합한 스케줄링 기법을 선택하고 아래 표와 같은 태스크 주기, 실행 시간, 마감 시간 등을 결정할 때, 분석적 방법을 이용하면 쉽게 시스템의 스케줄링 가능성 여부를 판단할 수 있다[1].

〔표 1〕 스케줄링 선택방법의 분석위한 표기법

표기법	의미
n	태스크 집합의 태스크 개수
t_i	태스크 집합의 태스크 (1 ≤ i ≤ n)
C_i	t_i의 수행 시간
T_i	t_i의 주기
D_i	t_i의 마감시간
W_i	최악의 경우의 수행 시간
U	프로세서 이용률(U)
S	태스크 전환 시 발생하는 오버헤드

가. 비율 단조 스케줄링 (RMS)

비율 단조 스케줄링(RMS, Rate Monotonic Scheduling) 은 최적의 정적 알고리듬으로 알려졌다. 주기를 기초로 태스크에 정적 우선순위를 할

당하고 짧은 주기의 태스크에 높은 우선순위를 할당하며 태스크가 도착할 때 우선순위가 할당되므로 우선순위는 재 계산될 필요가 없다.

이것은 단일 프로세서 상에서 고정 우선순위 기반 선점형 스케줄링 방식으로 다음과 같은 가정을 하고 있다[3].

1) 태스크들은 주기적이고, 주기와 마감 시간은 같으며, 실행 중에 스스로 중지 되지 않는다.
2) 태스크들은 선점될 수 있고, 문맥 교환과 태스크 스케줄링에 드는 비용은 무시한다.
3) 모든 태스크들은 서로 독립이다. 즉, 선행 제약은 존재하지 않는다[4].

it의 최악의 경우의 수행 시간 (iW)은 다음 [식1]과 같이 계산할 수 있다[3].

$$W_i(n+1) = C_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i(n)}{T_j} \right\rceil C_j, \quad W_i(0) = 0 \qquad [식1]$$

이 때, Wi(n + 1)= Wj(n)이고 Wi(n − 1)≤ Ti(n)이면 ti는 스케줄링 가능하고, 그렇지 않은 경우에는 ti는 마감시간을 놓치게 된다.

나. 마감 시간 우선 스케줄링 (EDF)

마감 시간 우선 스케줄링(EDF, Earliest Deadline Firs) 알고리듬은 동적 우선순위 스케줄링 방법으로 Liu와 Layland에 의해서 제시되었다. EDF는 동적 알고리듬 중에서 선점 알고리듬으로는 최적인 것으로 알려져 있다. 태스크의 우선순위는 마감시간에 따라서 할당된다. 즉, 마감시간이 짧을수록 높은 우선순위가 할당되므로 임의의 순간에 실행되는 태스크는 실행이 완료되지 않은 태스크들 중에서 마감시간에 가까운 것이 선택된다.

또 정적 알고리듬과 달리 태스크의 우선순위가 시간에 따라 변하게 된다.

Liu와 Layland는 EDF 알고리듬에 대해서도 스케줄링 가능성 기준을 제시하였다. 주기적으로 발생하는 독립적인 n개의 태스크에 대해, EDF 알고리듬의 스케줄링 가능판단에 대한 프로세서이용률(u)의 충분조건은 [식2]와 같다[3].

$$U = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \le 1 \qquad [식2]$$

이 식에서 보여주듯이 프로세서 이용률(u)이 최대 1(100%)까지 가능함을 알 수 있다. 따라서 태스크 전환시 발생하는 오버헤드(S)를 고려할 경우 태스크 집합의 프로세서 이용률(U)은 [식3]과 같다.

$$U = \sum_{i=1}^{n-1} \frac{C_i + 2S + I}{T_i} + \frac{C_j - 2S}{T_j}, \quad j = i + 1 \qquad [식3]$$

주어진 태스크 집합에 대하여 RMS 알고리즘으로 스케줄링 가능하고 EDF 알고리즘보다 나은 성능을 이끌어내기 위한 조건은 다음 [식4]와 같다.

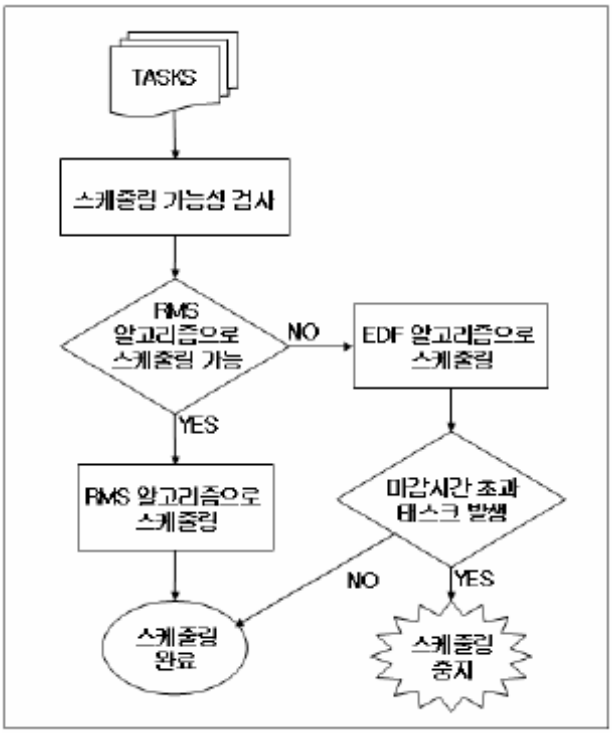
$$U = \sum_{i=1}^{n-1} \frac{C_i + 2S + I}{T_i} + \frac{C_j - 2S}{T_j} \le n(2^{1/n} - 1),$$
$$n = 1,2,\ldots, \; j = i + 1 \qquad [식4]$$

또한, EDF 알고리듬은 동적 우선순위 알고리듬들 중에서 최적으로, EDF 알고리듬으로 스케줄링될 수 없으면 다른 동적 우선순위 알고리듬으로도 스케줄링될 수 없다. 그러나 새로운 태스크를 할당할 스케줄링 오버

헤드가 큰 단점이 있다. 마감시간이 주기와 동일하거나 주기에 비례하여 작을 경우 RMS와 효율이 같고 마감시간이 주기보다 작을 때 최적이다.

1.2.3. 선택 스케줄러 알고리듬 설계

본 논문에서 스케줄러 선택은 스케줄링 가능성 검사 알고리듬에서 얻어지는 프로세서 이용률을 기반으로 수행하며, 시스템에 영향을 미치지 않도록 그림과 같이 간결하게 설계하도록 제안하였다[6].



[그림 2] 스케줄링 관리자의 동작 흐름도

1.2.4. EDF 스케줄러 확장 알고리듬 설계

RMS 알고리듬으로 스케줄링이 가능한 충분조건은 693.0=U으로서, 프로세서 이용률이 0.693 이하일 경우 RMS 알고리듬을 선택하고, 그 이상일 경우 별도로 최악의 수행 시간을 계산하지 않고 EDF 알고리듬을 제안하였다[6].

```
U = 태스크 집합의 프로세서 이용률 계산
 if( U : 0.693 )
      RMS로 스케줄링
  else
     EDF로 스케줄링
```

RMS 알고리듬으로 안정적인 스케줄링을 보장할 수 없는, 프로세서 이용률이 0.693을 초과하는 상황에 대해서 EDF 알고리듬을 이용하여 스케줄링한다. 하지만, EDF 알고리듬으로도 프로세서 이용률이 100%를 넘어갈 경우 안정적인 스케줄링을 보장할 수 없다. 따라서 EDF 알고리듬으로 스케줄링 할 때, 각각의 태스크가 실행에 들어가기 전에 마감시간이 초과되는 지를 체크하여 마감시간을 초과하는 상황이 예상될 경우 스케줄링을 중지한다.

1.3. 시뮬레이션 및 결과

고찰실시간 리눅스 상에서 실시간 태스크들의 스케줄링 가능성을 검사하여 특성에 맞는 스케줄러를 선택하는 스케줄러 관리자의 스케줄링 결과를 기존의 RMS와 EDF 알고리듬과 비교 평가한다. 선택적 알고리듬의 성

능분석을 시뮬레이션으로 위한 운영체제로 RT Linux 3.1과 Linux 커널 버전 2.4.32를 사용하였으며, 스케줄러는 실시간 리눅스에서 기본적으로 제공하는 RMS 스케줄러와, Practicia Balbastre와 Ismael Ripoll에 의해 구현된 EDF 스케줄러를 인스톨함으로서 프로세서 이용률에 따른 평균응답시간의 성능을 분석하였다.

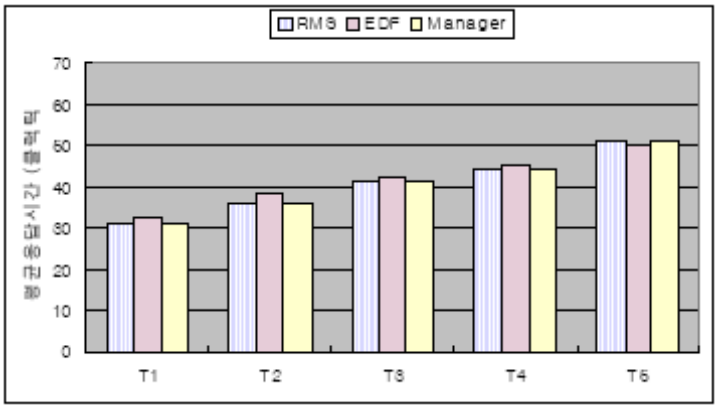
1.3.1. 프로세스 이용률에 따른 성능분석

스케줄러는 실시간 리눅스에서 제공하는 RMS 스케줄러와, EDF 스케줄러를 인스톨하여 사용한다. 시뮬레이션을 마감시간과 주기가 다른 경우로 나누어 각각 다양한 프로세서 이용률을 가지는 태스크 집합을 준비하고, 성능분석을 위한 평균 응답 시간과 마감시간을 비교한다.

가. 프로세스 이용률:0.626(주기=마감시간)

[실험 1]에서는 기존의 RMS와 EDF 알고리듬으로 스케줄링을 한 경우의 평균 응답시간을 구하고 스케줄러 선택 알고리듬이 스케줄러를 선택하여 스케줄링을 한 경우를 비교하였다. RMS 알고리듬의 성능이 EDF 알고리듬 보다 약간 좋음을 알 수 있다.

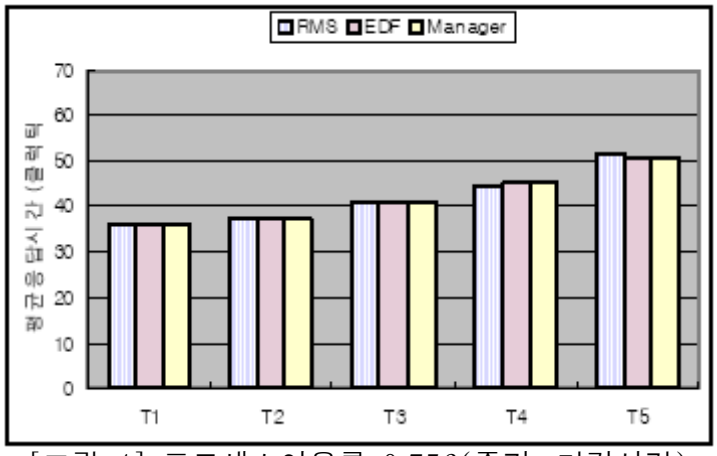
RMS의 결과와 매니저(Manager)의 결과가 동일함을 알 수 있다. 이는 스케줄링 가능성 검사에서 RMS 알고리듬의 충분조건인 $U \le n(2^{\frac{1}{n}}-1)$ 이 넘지 않기 때문에, RMS 알고리듬을 선택하여 스케줄링 되었다고 볼 수 있다.



[그림 3] 프로세스이용률 0.626(주기=마감시간)

나. 프로세스 이용률:0.756(주기=마감시간)

[실험 2]에서는 태스크 집합의 프로세서 이용률이 RMS 알고리듬의 충분조건인 0.693을 넘었기 때문에, RMS로 스케줄링 할 경우 태스크들이 마감시간 내에 수행을 완료하는 것을 보장하지 못한다.



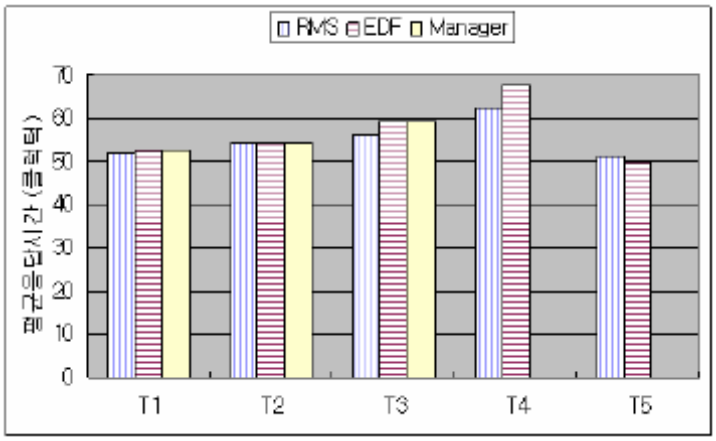
[그림 4] 프로세스이용률 0.756(주기=마감시간)

그러나 RMS 스케줄링 경우에도 태스크 집합에서 마감시간을 넘겨 스케줄링 된 태스크는 없었다. 스케줄러 선택 알고리듬은 EDF 알고리듬을

선택하여 스케줄링 하였다. EDF 알고리듬의 경우 스케줄링 가능한 구간이 10≤≤U이기 때문에 충분히 스케줄링 안정성을 보장하며, RMS 알고리듬에 비하여 크게 성능이 떨어지거나 하는 일은 없었다.

다. 프로세스 이용률:1.101(주기=마감시간)

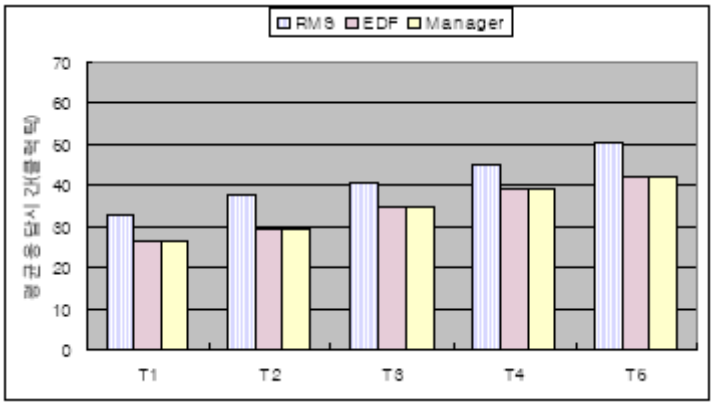
[실험 3]에서는 태스크 집합에 대한 프로세서 이용률이 100%를 초과하고 있다. 이 경우 RMS와 EDF 알고리듬 모두 태스크들의 마감시간 내 스케줄링을 보장할 수 없다. RMS 알고리듬과 EDF 알고리듬 모두 태스크 4와 5에서 마감시간 위반이 발생한다. 스케줄러 선택 알고리듬은 마감시간을 위반하는 태스크가 발생할 경우 시스템의 안정성 확보를 위해 스케줄링을 중지한다. 이를 위해 EDF 스케줄러를 확장하였기 때문에 마감시간을 위반하게 되는 태스크 4 이전의 태스크의 경우 EDF 알고리듬과 동일한 성능을 보여주게 된다.



[그림 5] 프로세스이용률 1.101(주기=마감시간)

라. 프로세스이용률:0.626(주기≠마감시간)

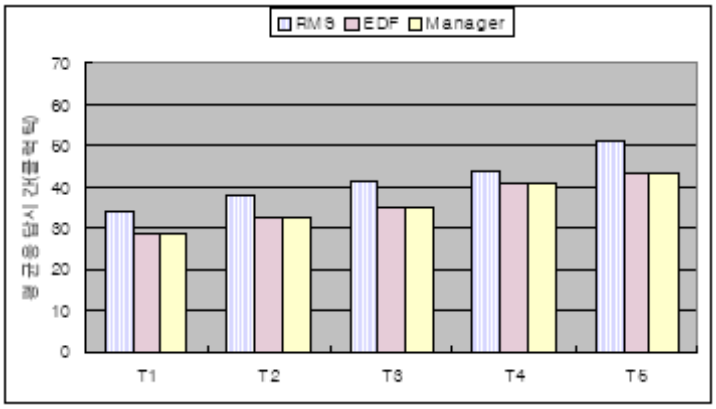
[실험 4]에서는 주기와 마감시간이 다르므로 [그림 6]에서의 결과와 같이 EDF 알고리듬이 RMS 알고리듬 보다 더 좋은 평균 응답시간을 보였다. 따라서 스케줄러 선택 알고리듬에서는 주기와 마감시간이 다른 태스크 집합에 대해서는 EDF 알고리듬으로 스케줄링을 할 수 있으므로 평균 응답 시간도 EDF 알고리듬과 동일하게 나타났다.



[그림 6] 프로세스이용률 0.626(주기≠마감시간)

마. 프로세스이용률:0.756(주기≠마감시간)

[실험 5]는 태스크 집합의 프로세서 이용률이 실험 2에서처럼 0.760인 경우 RMS 알고리듬으로는 태스크들의 마감시간 내에 수행을 보장하지 못한다. 하지만 여기서도 [실험 2]에서와 같이 RMS 알고리듬으로 태스크는 없었다. [실험 5]에서 스케줄러 선택 알고리듬은 태스크 집합의 프로세서 이용률이 RMS 알고리듬의 충분조건인 0.693을 넘었고, 주기와 마감시간이 다르게 설정되었으므로 [그림 7]과 같이 EDF 알고리듬을 선택하여 스케줄링할 수 있다.



[그림7]프로세스이용률0.756(주기≠마감시간)

1.3.2. 스케쥴러 선택 알고리듬의 성능고찰

RMS 알고리듬은 마감시간이 주기와 같은 태스크 모델에서 최적이지만, 마감시간이 주기보다 작은 경우에는 최적이지 못하고, 프로세서 이용률도 평균 88%라는 제약이 존재한다. EDF 알고리듬은 마감시간이 주기와 동일한 경우 RMS 알고리듬의 효율과 같고, 마감시간이 주기보다 작은 경우 최적이며, 프로세서 이용률을 거의 100%까지 이용할 수 있다. 하지만, 동적 우선순위 스케줄링 방법인 EDF 알고리듬은 스케줄링 오버헤드가 존재하기 때문에 마감시간이 주기와 동일할 경우 정적 우선순위 스케줄링 방법인 RMS 알고리듬보다 효율이 떨어질 수 밖에 없다. 따라서 마감시간과 주기가 같지 않은 경우와 스케줄링 오버헤드를 고려하여 스케줄링 가능성 검사 알고리듬을 구현하였다. 이를 이용하여 RMS 알고리듬으로 스케줄링이 가능한 경우 RMS 알고리듬으로 스케줄링을 하고, 그렇지 못할 경우 EDF 알고리듬으로 스케줄링을 태스크 집합이라도 마감시간을 보장하여 스케줄이 가능하다. [표 2]는 RMS과 EDF 알고리듬의 비교와 본 논문에서 제안하는 스케줄링 관리자의 성능분석 결과이다.

[표 2] 프로세스 이용률에 따른 스케줄링 관리자의 성능분석 결과

	이용률	주기: 마감시간	RMS	EDF	관리자가 선택한 알고리듬
실험1	0.626	=	○	△	RMS
실험2	0.756	=	○	△	EDF
실험3	1.101	=	X	X	-
실험4	0.626	≠	△	○	EDF
실험5	0.756	≠	△	○	EDF
○ 스케줄링 가능, 최적의 성능 △ 스케줄링 가능 X 안정적인 스케줄링 불가 - 스케줄링 중					

1.4. 결론

논문에서는 실시간 태스크들을 스케줄링 하기 위해서 스케줄링 관리자를 설계하였으며, 마감시간 내에 태스크의 수행을 보장하기 위해 EDF 스케줄러를 확장하여 시스템 정지와 같은 위험한 상황을 미리 예측 가능하도록 구현하였다.

또한, 실시간 리눅스가 보다 동적인 상황에 적응이 가능하도록, RMS 알고리듬과 EDF 알고리듬을 태스크 집합의 특성에 따라 유연하게 선택함으로써 안정적인 스케줄링과 더 나은 성능을 이끌어 낼 수 있도록 선택 알고리듬을 제안하였다. 제안한 선택 알고리듬의 결과로 실행시간 오버헤드를 가지는 동적 스케줄링 알고리듬인 EDF 알고리듬보다 정적 스케줄링 알고리듬인 RMS 알고리듬을 선호하도록 하였고, RMS 알고리듬으로 스케줄링이 불가능한 경우 EDF 알고리듬을 선택하도록 하였다. 스케줄링 오버헤

드와 마감시간이 항상 주기의 의한 성능분석을 위해 컴퓨터 시물레이션을 통한 결과를 분석결과 본 논문에서 제안한 스케줄링을 선택하는 방법은 어떠한 특성의 태스크 집합이라도 마감시간을 보장하는 스케줄이 가능하게 되었다.

향후 스케줄링 가능성 검사 알고리듬은 스케줄러 선택 알고리듬의 오버헤드는 무시할 수 있는 수준이었지만, 오버헤드를 줄일 수 있는 최적화 기법의 연구가 계속되어야 할 것이다.

참 고 문 헌

[1] Fredette A. and Cleaveland R., "A Generalized to Real-Time Schedulability Analysis.", 10th workshop on real-time operating system and software, 1993.

[2] John L., Sha L., Strosnider K. and Tokuda H., "Fixed Priority Scheduling Theory for Hard Real-Time Systems.", Foundations of real-time computing : Scheduling and Resource Management, pp.1-30., Mar., 1990.

[3] Warren C., "Rate Monotonic Scheduling.", IEEE Micro, pp.34-38, Jun., 1991. [4] Liu J. W. S., "Real-Time Systems", Prentice Hall, 2000.

[5] Kevin M., "Preemptible Linux-a reality check.", Articles and whitepapers about Linux in embedded applications of LinuxDevice.com, Sep., 2001.

[6] Daniel P. and Marco C., "Understanding the Linux Kernel.", O'Reilly, 2001.

[7] 심형용, 김지환, 선동국, 김성조,"RTOS를 위한 TCP/IP 프로토콜 스택의 구현," 한국정보과학회 추계 학술발표논문집, 제29권제2호, pp427-429, 한국정보과학회, 2002.10

2. 오픈소스 컴포넌트 활용에 있어서의 소프트웨어 개발방법론 적용에 관한 탐색적 연구

김종배, 김두연, 류성열

2.1. 서 론

최근 소스 코드의 공개를 통해 사용, 복제, 수정, 배포가 자유로운 오픈소스 소프트웨어가 상용 소프트웨어의 독과점에 의한 폐해를 방지하고, 유지보수 및 업그레이드에 투여되는 예산과 절차를 감축할 수 있다는 장점을 바탕으로 전 세계적으로 그 영역이 확대되고 있고[1, 2], 기업들도 기존 소프트웨어 개발의 한계점들, 즉 소프트웨어의 품질, 개발 속도 및 개발비용 등의 문제를 해결하기 위한 새로운 대안으로서 오픈소스를 활용한 소프트웨어 개발 접근방법의 적용을 시도하고 있다[3]. 이에 따라, 그 동안 오픈소스 소프트웨어에 대한 다양한 분석들이 이루어졌지만, 실제 산업현장에서 소프트웨어 개발에 오픈소스를 활용하기 위한 절차나 방법에 대한 적절한 연구결과가 미흡한 실정이다. 오픈소스는 개발 도구로서의 사용, 완성된 소프트웨어 패키지의 변경 도입, 컴포넌트의 도입 등 그 활용 수준이 다양할 수 있고, 각 활용수준에 따라 절차와 방법이 다를 수 있다.

따라서 본 연구에서는 컴포넌트 수준에서의 활용 절차를 중심으로 관련 문헌들에 대한 검토를 바탕으로 오픈소스 컴포넌트의 활용을 위한 핵심 절차를 도출하고, 한국전자통신연구원에서 개발한 컴포넌트 기반의 시스템 개발방법론인 마르미-Ⅲ[5]의 개발 프로세스에 적용, 실제 프로젝트에서 활용한 사례를 통해 그 유효성을 검증하였다.

2.2. 관련 연구

2.2.1. 오픈소스 소프트웨어의 개념 및 활용현황

오픈소스 소프트웨어는 라이센스 요금이 무료이면서 소스코드를 개방한 소프트웨어로 누구나 자유롭게 사용·활용·개선할 수 있다. 따라서 오픈소스 소프트웨어는 프로그램을 복제하여 배포할 수 있는 권리, 소프트웨어의 소스코드에 접근할 수 있는 권리, 프로그램을 개선할 수 있는 권리들을 개발자 또는 사용자에게 보장한다. 이렇게 소스코드를 수천, 수만 명의 프로그래머, 사용자와 공유함으로써 오픈소스 소프트웨어는 자유롭게 버그를 해결하고, 성능을 강화하고, 자신의 목적에 따라 코드를 수정, 사용할 수 있도록 하여 공동체의 창조력과 협력을 이끌어낼 수 있다는 장점들이 여러 연구들에서 언급되고있는데[1, 4, 6, 12], 이러한 오픈소스 소프트웨어 활용의 장단점을 제시하면 <표 1>과 같다[8, 4, 10, 13].

〈표 1〉 오픈소스 소프트웨어 활용의 장단점

구분	장단점	내용
장점	융통성	라이센스 비용이나 허락에 제한을 받지 않고, 다양한 오픈소스 컴포넌트들을 데스크 탑 상 최선의 것을 선택할 수 있음
	기술 지원	신속한 문제해결, 빨라진 성능개선 프로세스, 기술의 공동 습득이 가능
	기술혁신촉진	유료일 경우 사용이 힘었을 컴포넌트의 실험적 적용을 촉진
	재활용	소스코드 접근이 가능하여 재활용이 용이
	품질	이미 개발되고 검증된 컴포넌트를 사용함에 따라 개발이 빨라지고 유연해질
	표준	독점소프트웨어보다 표준에 더 충실하고 상호 운영성이 더 뛰어나, 개발 과정에서 새로운 표준이 형성될 수 있음
단점	응용의 부족	윈도우즈 기반의 응용프로그램에 비해 리눅스 기반의 응용프로그램이 부족
	문서화 미비	상용 프로그램에 비해 체계적인 문서를 갖고 있지 못함
	불확실한 계획	영리보다 자발적 참여로 개발되기 때문에 상용제품과 같은 로드맵을 기대하기 힘듦

〈표 2〉 오픈소스 소프트웨어 제품 사례

제품	특성
Linux	운영체제
Apache	웹 서버
Sendmail	인터넷 메일 유틸리티
Bind	웹 Domain Name Server를 운영하는 데 사용되는 소프트웨어
Perl, Python	프로그래밍 언어
Gnu software	다양한 컴파일러, 유틸리티, EMACS editor
GNOME, KDE	Linux GUI 인터페이스
Mozilla	Netscape Navigator의 오픈소스 소프트웨어 버전
Jikes	IBM이 개발한 Java 컴파일러
GIMP	Image Workshop
Darwin	Mac OSX 서버 시스템
SAMBA	마이크로소프트의 SMB 프로토콜과 통합을 허용
Ghostscript	Aladdin Enterprises PDF 유틸리티
Doom	게임

오픈소스 소프트웨어 제품들은 <표 2>에서 보는 바와 같이 주로 개발도구, 하부 구조형 소프트웨어, 네트워킹 유틸리티들인데, 이들은 주로 성능과 신뢰성이 중요시되는 제품들이다[1, 7, 11].

2.2.2. 오픈소스 소프트웨어를 활용한 시스템 개발방법론

오픈소스를 활용한 소프트웨어 개발은 크게 두 가지 모델로 구분할 수 있다. 첫 번째는 독점 라이센스 하에 판매되는 소프트웨어를 오픈소스 라이센스로 전환(아웃바운드)하는 것이고, 두 번째는 오픈소스를 기업에 적용하는 것, 즉 현존하는 오픈소스 소프트웨어를 쓰거나 그것을 제품의 부분으로, 또는 IT프로젝트 안에서 사용되게 통합하는 과정(인바운드)이다[4].

먼저, 아웃바운드는 기업의 인지도 향상, 제품의 판매 촉진, 유지비용 절감 등을 위해 소프트웨어를 오픈소스 커뮤니티에 릴리즈하는 것인데, 이를 위해서는 <표 3>에서 보는 바와 같이 사전에 비즈니스 측면의 이점과 단점을 신중히 고려하여야 한다[4].

〈표 3〉 아웃바운드의 장단점

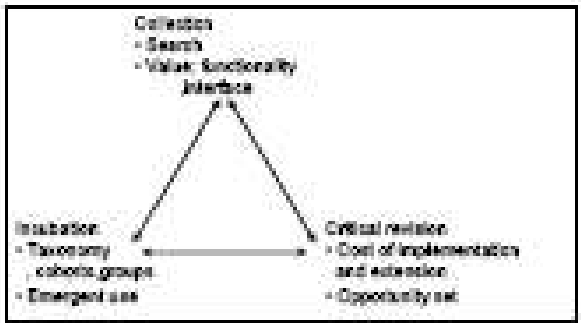
구분	장단점	내용
장점	편재	다른 제품이나 솔루션의 기반이 되는 핵심기술을 보유하고 있다면, 산업 표준으로 확립되도록 할 수 있음
	비용 절감	커뮤니티와의 공동 작업으로 솔루션 개발 비용을 절감
	하드웨어	하드웨어에 내장된 소프트웨어의 경우 소스의 오픈은 하드웨어 판매를 촉진
	서비스와 지원	주요 비즈니스 모델이 서비스 공급에 있을 경우 효과를 볼 수 있음
	파트너 역할	큰 규모의 프로젝트에서 파트너나 경쟁자와의 협력을 촉진할 수 있음
단점	통제 시점	제품이 경쟁을 통제(진입장벽)하고 있는 경우 오픈을 고려해서는 안 됨
	제품의 진부화	진부한 제품을 유지하기 위해 오픈하는 것은 오히려 역효과
	부정적인 매출예상	소스의 오픈은 지속적인 투자와 활동을 요구하므로 수익이 예상되지 않는다면 실행해서는 안됨
	커뮤니티와의경쟁	유사한 프로젝트가 이미 존재한다면, 경쟁보다는 협력하여 길을 놓여야 함

마찬가지로, 현존하는 오픈소스 소프트웨어를 쓰거나 그것을 제품의 부분으로 또는 프로젝트 안에서 사용되게 통합하는 과정(인바운드)은 기존의 비즈니스와 근본적으로 다르다. 오픈소스 소프트웨어를 가져오거나 내부 프로젝트에 통합할 때 얻을 수 있는 이점과 피해야하는 경우(단점)를 살펴보면 <표 4>와 같다[4].

<표 4> 인바운드의 장단점

구분	장단점	내용
장점	표준	조직에 필요한 주요 산업 표준의 오픈소스 도구가 있다면 이 표준에 근거한 제품을 만드는 것은 유의미함
	현존기술활용	활용할 수 있는 기술이 이미 개발되었다면 그것을 사용하는 것이 프로젝트를 진척시키는 가장 효과적인 방법임
	자원집중	오픈소스가 프로젝트의 기초로 효과적으로 사용될 수 있다면, 더 높은 부가가치 창출에 자원을 집중시킬 수 있음
단점	전략적 방향 불일치	오픈소스 프로젝트가 당장의 요구를 만족시키더라도, 장기적 방향이 비즈니스 목표와 맞지 않을 수 있음
	기술적 동의	조직의 프로젝트 관련 주요 기술자가 아키텍처의 부분으로 오픈소스 소프트웨어의 사용에 동의하지 않는다면, 새로운 시도에 대한 구조적 문제(예를 들면, 문화적 현호)에 봉착할 수 있음
	릴리즈 시기 불일치	커뮤니티의 소프트웨어 출시 시기가 통제되지 않을 수 있음(커뮤니티와의 계약이나 공헌을 통한 합의가 필요)

인바운드 오픈소스의 경우, 오픈소스 소프트웨어의 활용 수준에 따라 그 절차나 방법이 다를 수 있다. 즉, 오픈소스를 패키지 수준에서 활용할 것인지, 라이브러리 수준에서 사용할 것인지, 컴포넌트 수준에서 사용할 것인지, 도구나 개발 환경으로 사용할 것인지에 따라 전체 프로젝트에서 오픈소스 적용의 범위가 달라질 수밖에 없고, 이에 따라 적절한 활용 전략(모델)과 절차(프로세스)를 선택하여야 한다. 특히, 본 연구의 범위인 컴포넌트 수준에서의 활용에 있어 핵심적인 전략과 절차는 '적절한 오픈소스 컴포넌트의 선정'에 관한 것으로, 이를 위해 기존 연구 중에서 참조할 수 있는 모델로 제시되고 있는 것이 T.R. Madanmohan and Rahul De'(2004)의 '오픈소스 컴포넌트 선정 모델'인데, 이 모델은 [그림 1]과 같이 '수집(Collection), 배양 (Incubation), 개정(Critical Revision)'의 세 단계(Basic Steps)로 구성된다[9].



[그림 1] 오픈소스 선정 모델

첫째, '수집(Collection)'은 개발자들이 관련 정보를 검색하고 새로운 아이디어와 컴포넌트를 탐색하는 단계이다. 즉, 커뮤니티를 조사하고 프로젝트 로드맵에 기초하여 가능성 있는 컴포넌트(후보 컴포넌트)들을 식별한다.

둘째, '배양(Incubation)' 단계에서는 컴포넌트에 대한 평가 정보를 수집, 작성하고 브레인스토밍(Brainstorming)과 가정(what-if) 분석을 통해 컴포넌트의 유용성(Usability)을 평가한다. 이 과정에서 아이디어나 컴포넌트가 서로 결합되기도 한다.

셋째, '개정(Critical Revision)'은 재검토, 가치측정 등의 비판적 검토를 통해 적절치 않은 후보들을 탈락시키는 단계이다. 물론 이러한 절차들은반복적으로 수행된다. T.R. Madanmohan and Rahul De'(2004)의 '오픈소스 컴포넌트 선정 모델'은 활용 가능한 오픈소스 컴포넌트의 식별과 평가를 위한 모델을 제시하고 있다는 점에서 의미있는 연구이지만, 실제 오픈소스 컴포넌트를 활용한 개발에 적용하기 위해서는 세부적인 절차(프로세스)의 보완과 컴포넌트 기반 개발프로세스(Component Based Development, CBD)와 같은 전체적인 절차로의 적용 과정이 필요하다.

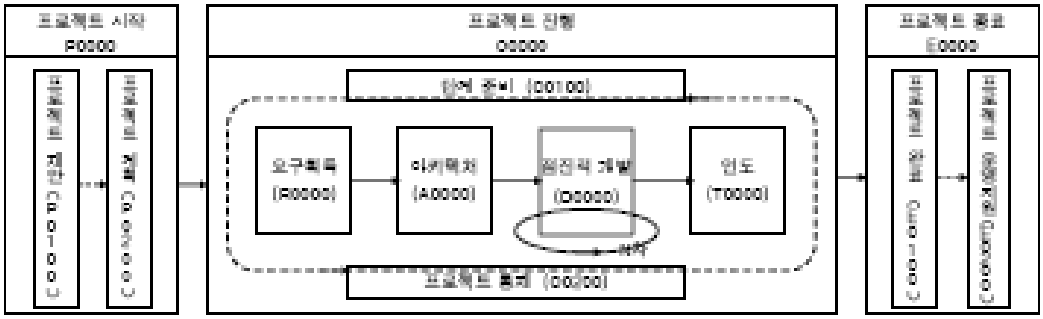
2.3. 오픈소스 컴포넌트를 활용한 소프트웨어 개발 절차

오픈소스 컴포넌트를 활용한 소프트웨어 개발은 오픈소스 활용 계획의 전략적 측면 이외에도, 후보 컴포넌트의 식별 및 평가를 위한 정보의 수준, 획득 및 변경의 과정 등 여러 가지 면에서 기존의 상용 컴포넌트 기반의 소프트웨어 개발 절차를 그대로 적용하기에는 한계가 있다.

따라서, 본 장에서는 T.R. Madanmohan and Rahul De'(2004)가 제시한 모델을 보완, 확장하고 이를 컴포넌트 기반 개발 프로세스(마르미-Ⅲ)에 적용하여 실제 개발 과정에 적용할 수 있는 실용적인 오픈소스 컴포넌트 활용 프로세스를 도출하고자 한다. 한국전자통신연구원은 기존의 정보공학 및 구조적 방법론 기반의 마르미(MaRMI, Magic and Robust Methodology Integrated)와 객체지향 기반의 마르미-II 방법론에 이어, 학계 및 산업계와 협력하여 컴포넌트 기반의 마르미-III 개발 방법론을 개발하였다. <표 5>와 [그림 2]에서 보듯이, 마르미-III는 컴포넌트의 개발 및 컴포넌트 기반의 시스템 개발에 필요한 작업과 이에 필요한 기법 및 작업별 산출물을 정의하고, 작업에 따른 상세한 개발 절차와 지침을 제공 한다.

〈표 5〉 마르미 방법론 현황

구분	마르미	마르미-II	마르미-III
개발 기간	1994.10 – 1997.9	1998.11 – 1998.10	1999.7 – 2003.10
지원 방법	구조적 방법, 정보공학	객체지향방법	컴포넌트 기반
특징	• 국제표준 수용(ISO 12207) • 개발공정의 계층화/상세화 • 산출물의 간소화	• UML 기반 • 반복적/점진적 개발 • 위험관리	• EJB기반,COM+, CORBA, .NET, 웹서비스 지원 • 아키텍처중심, 프로젝트/품질관리 지원 • 마르미-II 데타모형 공유, 사용자 방법론 개발/조정지원



[그림 2] 마르미-III 공정

마르미-III는 절차서, 기법서, 양식정의서 및 적용사례서로 구성되어 있다. 개발 공정은 컴포넌트 개발 공정과 이미 개발된 컴포넌트를 활용하여 시스템을 개발하는 공정으로 구분할 수 있는데, 마르미-III는 컴포넌트 기반 시스템 개발 공정을 중심으로 컴포넌트 개발 공정을 함께 기술하였다.

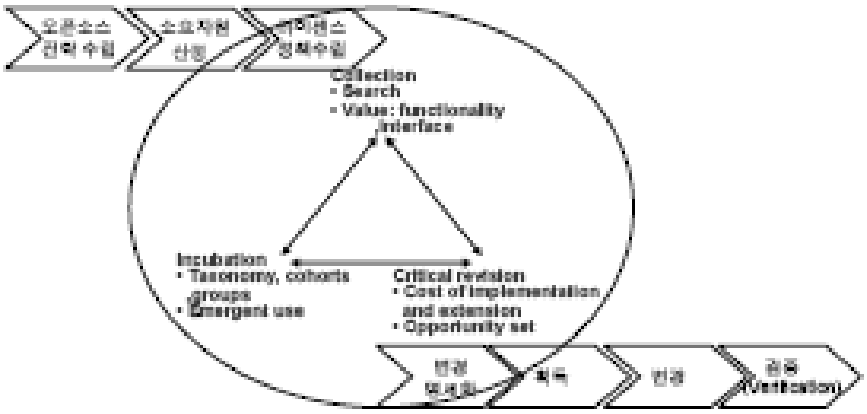
개발 절차는 크게 계획단계, 아키텍처단계, 점진적 개발단계, 인도단계의 4개 단계로 이루어져 있고, 각 단계는 논리적으로 서로 연관된 작업을 하나로 묶은 활동으로 구성되어 있다. 각 작업은 하나 이상의 산출물을 만들어 내며, 이를 위한 상세한 세부 절차가 정의되어 있다.

2.3.1. 오픈소스 컴포넌트 활용 절차

오픈소스의 활용은 많은 장점에도 불구하고 응용 의 부족, 문서화의 미비, 불확실한 계획 등과 단점(Risk)도 존재하기 때문에 초기 프로젝트 계획수립 단계에서부터 전략적 의사결정의 과정이 필요하다. 따라서, T.R. Madanmohan and Rahul De'(2004)의 '오픈소스 컴포넌트 선정 모델'에 오픈소스 활용 전략 수립, 소요자원산정, 라이센스 정책수립 등의 선행 절차를 추가할 필요가 있다. 또, 오픈소스 컴포넌트 선정을 위해서는 후보 오

픈소스 식별 방법과 적합성 평가(심사)의 기준 등이 보완되어야 하며, 선정된 오픈소스 컴포넌트의 획득 및 활용을 위해서는 변경 명세화, 획득, 변경 등의 절차가 추가되어야 한다.

본 연구에서 T.R. Madanmohan and Rahul De'(2004)의 '오픈소스 컴포넌트 선정 모델'을 보완하여 새롭게 제시된 '오픈소스 컴포넌트 활용 프로세스'를 도식화하면 [그림 3]과 같다.



[그림 3] 변경된 오픈소스 컴포넌트 활용 절차

제시된 절차의 각 활동과 주요 내용은 다음과 같다.

가. 활동 1 : 오픈소스 활용 전략 수립

활동 1은 오픈소스 프로젝트를 통한 비즈니스 목표를 결정하는 활동이다. 이 활동의 입력물은 프로젝트정의서와 프로젝트제안서이고 출력물은 프로젝트수행전략기술서, 위험분석서, 프로젝트수행계획서 등이다. 오픈소스 활용 비즈니스의 목표는 조직과 프로젝트의 특성에 따라 결정되어야 하는데, 이때 '어떤 프로젝트에 오픈소스 컴포넌트를 사용할 것인가? 어떤 라

이센스 정책을 가진 오픈소스 컴포넌트를 사용할 것인가? 조직내 자원이 아닌, 오픈소스 라이센스 하에서 제공되는 컴포넌트를 다룰 때 팀원들을 어떻게 통제할 것인가? 오픈소스를 포함하는 개방된 컴포넌트를 사용하기 위해 어떤 프로세스를 적용할 것인가?' 등을 고려하여야 한다.

나. 활동 2 : 소요자원 산정

활동 2는 오픈소스를 활용할 수 있는 훈련된 조직을 구성하고, 지속적인 통제와 교육을 진행하는데 필요한 조직의 소요 예상 자원을 분석, 투입계획을 수립하는 활동이다. 이 활동의 입력물은 프로젝트정의서와 프로젝트제안서이고 출력물은 프로젝트수행계획서, 프로젝트예산내역서 등이다.

공수산정은 프로젝트 접근방법을 포함한 많은환경적 요인에 좌우된다. 특히, 프로젝트의 비용은 초기 도입 비용 뿐 아니라, 시스템 수명에 따른 유지 관리 비용을 함께 고려하여야 한다. 따라서, 오픈소스의 도입으로 인한 자원의 감소만을 생각해서는 안되며, 시스템을 유지 관리하는 과정에서 오픈소스 커뮤니티로부터 지원을 받을 수 없는 경우를 고려하여 예상 자원을 산정하여야 한다.

다. 활동 3 : 라이센스 정책 수립

오픈소스를 활용하기 위해서는 상용 소프트웨어와 마찬가지로 반드시 해당 오픈소스의 라이센스에 대한 준수가 필수적이다. 자칫 잘못하면 라이센스 위반을 양산하는 상황을 초래할 수 있다. 활동 3은 오픈소스 라이센스 관련 문제를 사전에 방지하기 위한 위험 분석과 이에 대한 위험 회피 계획을 수립하는 활동이다. 이 활동의 입력물은 프로젝트정의서와 프로젝트제안서이고 출력물은 프로젝트수행계획서, 위험분석서이다. 이 계획에는

개발 단계별로 준수해야 할 라이센스 관련 사항들을 반드시 명시하여야 한다.

라. 활동 4 : 수집(Collection)

이 활동은 기능 요구사항을 충족시킬 수 있는 오픈소스 컴포넌트들에 대한 판별 과정(screening)을 수행하는 과정이다. 즉, 오픈소스 커뮤니티를 조사하고 프로젝트 로드맵에 기초하여 가능성 있는 후보 컴포넌트들을 식별한다. 이 활동의 입력물은 요구사항기술서, 유스케이스모형기술서이고, 출력물은 검색된 오픈소스들의 리스트와 기능을 분석한 기능분석서이다. 후보 오픈소스는 프레쉬밋닷넷(Freshmeat. net)[14], 소스포지닷넷(SourceForge.net)[15]과 같이 잘 알려진 오픈소스 리포지터리를 통해서 찾을 수 있다. 이러한 사이트들은 카테고리(Software Categories) 형태와 주제(Topic)에 대한 검색 등 다양한 방식으로 원하는 소스를 찾을 수 있게 되어 있고, 해당 오픈소스별 간략한 기능 설명(About)뿐 아니라, 개발 상태(Development Status), 라이센스(License), 등록일(Registered), 활동성(Activity Percentile) 등 프로젝트 팀의 신뢰성을 판단할 수 있는 정보와 운영 체제(Operating System), 개발언어(Programming Language), 사용자 인터페이스(User Interface) 형태 등 적용기술에 대한 상세정보(Project Details)가 함께 제공된다. 또한 관련 포럼(Forums), 뉴스(News)와 같은 커뮤니티와 프로젝트 팀이나 회사가 운영하고 있는 홈페이지를 링크하고 있어서 해당 소스에 대한 판단의 근거로 참조할 수 있다. 이때 '우리가 원하는 핵심 부분이 이들의 프로젝트에 포함돼 있는가? 우리에게 익숙한 프로그래밍 언어로 작업돼 있는가? 프로젝트가 현재 진행 중인 것들인가? 우리가 달성하려는 목표와 유사성을 갖고 있는 라이센스에서 개발되고 있는가?'를 고려하여 적절한 후보들을 선정하여야 한다. 그리고, 국내외 여타의 오픈소스 사이트나 커뮤니티들을 통해 해당 오픈소스

사용 경험자들의 평가와 의견을 참조할 필요가 있다.

마. 활동 5 : 배양(Incubation)

이 활동에서는 컴포넌트에 대한 평가 정보를 수집, 작성하고 브레인스토밍(Brainstorming)과 가정(what-if) 분석을 통해 컴포넌트의 유용성(Usability)을 평가한다. 이 과정에서 아이디어나 컴포넌트가 서로 결합되기도 한다. 이 활동의 입력물은 오픈소스리스트, 기능분석서이고, 출력물은 오픈소스후보리스트이다. 오픈소스 컴포넌트의 소스 코드를 손쉽게 해독하고 이용할 수는 있으나 선정의 과정에서는 그러한 소스 코드를 세밀히 볼 필요는 없다. 컴포넌트의 유용성은 소스를 보거나 직접 수정하지 않고도 재사용 가능하다는 것이기 때문이다. 수정이 꼭 필요한 경우가 아니면, 소스 코드 보다는 사용 가능성과 지원에 대한 보증(광범위한 개발자 커뮤니티의 존재 여부 등)을 고려할 필요가 있다.

바. 활동 6 : 개정(Critical Revision)

이 활동은 재검토, 가치 측정 등의 비판적 검토를 통해 적절치 않은 후보들을 탈락시키는 활동이다. 이 활동의 입력물은 오픈소스후보리스트, 위험분석서이고, 산출물은 오픈소스선정리스트이다. 처음부터 요구사항에 알맞은 후보 오픈소스를 찾기보다는 우선 적절하지 못한 후보를 탈락시키는 것이 효과적인데, 이때 '프로젝트의 목적과 요구사항(기능, 성능, 또는 기타 비기능적 요구사항)의 만족 정도', '호환성', '지속적인 업그레이드 계획(비전)', '지원 가능한 공급자(개발 그룹), 사용자(개발자) 층의 존재유무', '변경 또는 재공학(Reengineering)의 용이성', '명세화(문서화) 정도', '오류의 유무', '적용 사례', '사용자(개발자)들의 평판' 등의 심사 기준을 적용하여 적절하지 못한 후보를 제거한다.

사. 활동 7 : 변경 명세화

이 활동은 오픈소스를 획득하기에 앞서 오픈소스와 요구사항 사이의 격차를 분석한 후, 해당 오픈소스의 도입이 타당하다고 판단되는 경우에 요구사항에 대한 오픈소스의 기능적 혹은 기술적 부족분(격차)을 해소하기 위한 방법과 범위, 내용 등을 명세화하는 활동이다. 이 활동의 입력물은 오픈소스후보리스트, 변경요구사항기술서, 위험분석서, 오픈소스선정리스트이고, 산출물은 재사용여 부비교표와 변경계획서이다. 오픈소스의 기능과 요구하는 기능이 맞지 않는 경우(기능적 불일치, 즉 요구사항에 기술된 기능이 없거나 업무 규칙을 처리하는 방식이 다른 경우)에 그 변경의 범위나 방법, 항목등을 정한다. 경우에 따라 원하는 기능 이상의 것을 제공하는 경우도 기능적 불일치가 발생한다고 볼 수 있지만, 이것은 기능적인 측면만을 보았을 경우에는 오픈소스의 도입에 큰 영향을 끼치지 못한다. 또, 업무에 필요한 데이터를 관리하지 않거나 필요 없는 데이터를 관리하는 경우(데이터 불일치)나 기존 소프트웨어와 도입 대상 오픈소스 또는 두개 이상의 도입 대상 오픈소스가 다루는 데이터가 서로 중복되는 경우에는 이들 데이터 사이에 동기화가 필요하다. 그리고, 오픈소스의 플랫폼이 목표 시스템의 플랫폼과 다르거나 데이터베이스가 다른 경우, 또는 이를 변경하는데 기술적으로 많은 노력이 필요한 경우(기술적 이슈)에 이러한 격차를 제거하거나 줄이기 위한 변경을 명세화(Specifying Changes)하여 오픈소스 도입에 따른 위험을 최소화할 필요가 있다.

아. 활동 8 : 획득

이 활동의 입력물은 오픈소스선정리스트, 변경 계획서이고, 산출물은 오픈소스컴포넌트와 재사용 오픈소스 컴포넌트 명세서이다. 선정된 소스코드를 다운로드 하는 것으로 획득 과정이 끝나지 않는다. 기존의 코드에 대

한 소유 권뿐만 아니라, 커뮤니티의 기여로부터 나온 부분에 대한 소유권도 결정해야 한다. 이러한 의사결정은 계획단계에서 선택한 비즈니스 모델에 근거할 것이다.

물론, 오픈소스와 독점적 라이센스라는 두 가지 서로 다른 라이센스 하에서 소프트웨어를 개발, 출시할 수도 있다. 만일, 이중라이센스 비즈니스 모델을 선택하였다면, 프로젝트에 기여한 사람으로부터 해당 부분에 대한 저작권을 반환 받을 것인지 아니면 하위 라이센스로 진행할 것인지를 확실히 결정해야 한다.

자. 활동 9 : 변경

이 활동은 실질적인 개선과 통합의 과정으로, 다수의 오픈소스 컴포넌트를 통합해야 하는 경우, 사전에 설계한 아키텍처에 의거하여 오픈소스 컴포넌트 간의 인터페이스를 정의한다. 이 활동의 입력물은 변경계획서이고 출력물은 재사용오픈소스컴포넌트이다. 오픈소스 컴포넌트의 아키텍처 및 소스 코드를 분석한 후, 필요한 경우 품질 향상을 위한 코드 개선을 수행하거나 새로운 인터페이스를 구현한다. 대부분의 변경 과정은 컴포넌트의 재사용이나 조립과정과 크게 다르지 않다. 하지만, 오픈소스는 획득하여 적용하는 중에도 계속 변경되거나 확장될 수 있다. 따라서 오픈소스의 변경을 기 적용한 프로젝트에 지속적으로 반영할 것인지를 판단하여야 하며, 반영할 경우를 고려하여 설계 및 개발시에 기존의 작업을 반복하지 않기 위한 사전 검토가 중요하다.

차. 활동 10 : 검증(Verification)

개발이 완료 되었으면, 개발 결과물인 소스 코드에 대해 실질적인 검증

작업이 필요하다. 이 활동에서는 단위 시험 및 통합 시험 외에도 라이센스 검증 작업이 필수적이다. 개발계획서 그 자체로는 라이센스 이슈가 없었더라도 실제 구현 과정에서 오픈소스 코드에 대한라이센스 이슈 검증없이 사용된 경우가 있을 수 있기 때문이다.

2.3.2. 마르미-III 기반 개발 프로세스 적용 : 사례연구

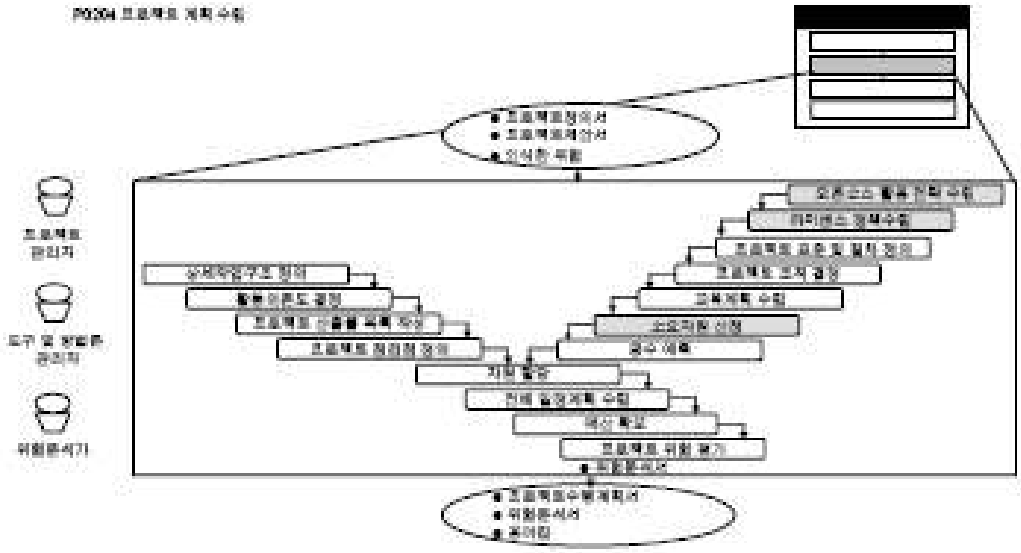
위에서 제안한 절차를 컴포넌트 기반의 개발방법론인 마르미-Ⅲ에 적용하고, 이를 오픈소스 컴포넌트 기반의 소프트웨어 개발 프로젝트에 활용하여 그 유효성을 검증하고 개선의 과제를 도출하였다. 연구 사례는 오픈소스 컴포넌트를 사용하여 웹리포팅툴을 개발하는 프로젝트였다. 웹리포팅툴은 출력할 리포트의 서식을 제작하고, 이를 웹에서 데이터베이스나 기타 데이터 소스와 연동하여 프린트할 수 있는 툴이다.

가. 프로젝트 시작 단계(P0000)

마르미-III의 프로젝트 시작 단계는 프로젝트의 수행목적과 목표, 범위를 결정하여 프로젝트기안서 등을 기준으로 프로젝트 수행목적, 범위, 프로젝트 제약사항 및 접근법을 확정하고, 상세한 프로젝트 수행계획을 수립하는 단계이다. 사례에서는 [그림 4]와 같이 마르미-III의 '프로젝트 계획수립(P0204)' 절차에 '오픈소스 활용 전략 수립', '소요자원산정', '라이센스 정책수립' 등의 활동을 추가하였다. 이 활동들에서는 프로젝트정의서를 토대로 프로젝트수행전략을 수립하고, 위험분석을 수행하였으며, 이를 프로젝트수행계획서와 위험분석서에 반영하였다. 리포팅툴은 기업에서 매우 필요한 소프트웨어로서 전혀 새로운 것이 아니며, 오래 전부터 많은 연구 개발이 이루어져왔기 때문에 화려하고 다양한 사용자 인터페이스를 가지며 웹 환경에서 사용 가능한 제품들이 개발, 판매되고 있었다. 그러나 리포트

의 폼 포맷이 아직까지 특정 제품에 종속적인 단점이 있어 표준화된 리포트 폼에 대한 요구가 높아, 이에 대한 개선이 요구되어졌다. 따라서 우리는 리포트 폼을 정의하는 도구로 XML을 사용하고, 플랫폼 독립적인 언어인 자바로 개발할 필요가 있었다. 오픈소스에 대한 사전 지식으로 기존의 오픈소스 커뮤니티들 중에 이러한 요구에 맞춰 몇 가지 오픈소스 그룹이 형성되어 있었기 때문에, 우리는 오픈소스를 기반으로 하여 프로젝트를 수행하기로 기본 전략을 정하고, 이에 대한 기초적인 조사를 거쳐 계획을 수립하였다.

조사과정에서 이러한 오픈소스들이 상당한 수준의 컴포넌트로 제공되고 있었으며, 지속적인 업데이트와 이를 도입한 제품들이 파생되고 있었다는 사실에 주목하여, 개발 완료후의 기능 보완을 염두하여, 컴포넌트 기반으로 프로젝트를 수행하기로 하였다. 한편, 몇 가지 오픈소스에서 이를 상용화하여 자유롭게 판매하는 것을 허용하고 있었기 때문에 라이센스에 대한 위험은 크지 않았으나, 소요자원에 대한 개략적인 산정과 추가적인 위험요인에 대한 검토과정에서 오픈소스 제품들이 다국어 지원, 특히 한글에 대한 지원이 만족할 만한 수준이 아닐수 있다는 것과 국내 사용자들이 특수하게 요구하는 정형화된 문서 서식들이나 다양한 그래프들에 대한 요구를 충족할 수 있는 소스를 찾을 수 있을 것인가라는 문제는 가장 큰 위험요인으로 파악되었기 때문에, 이 부분을 해결하기 위한 추가 개발비용을 고려하여야 했다.

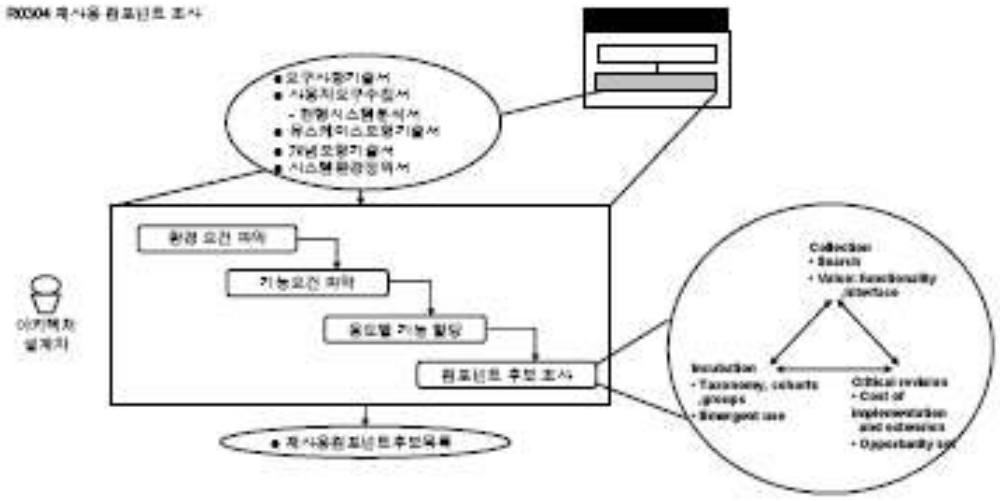


[그림 4] 변경된 프로젝트 계획 수립 절차

나. 요구획득 단계(R0000)

마르미-III의 요구획득 단계는 상위수준의 요구사항을 수렴, 개념적인 초기 시스템비전을 개발한다. 개발 시스템의 범위를 명확히 한 후, 사용자 요구사항을 유스케이스 모형과 개념 모형으로 정리, UI 프로토타이핑을 통해 사용자와 함께 검증하고, 이를 기반으로 시스템 아키텍처 초안을 정의, 재사용 가능한 컴포넌트를 조사한다. 사례에서는, [그림 5]와 같이 이 단계에 앞의 연구에서 보완한 '수집(Collection)-배양(Incubation)-개정(Critical Revision)'으로 구성된 '오픈소스 컴포넌트 선정 모델'을 마르미-III의 '재사용 컴포넌트 조사(R0304)' 절차에 적용하여 가능성 있는 컴포넌트(후보 컴포넌트)들을 식별, 평가, 검토하는 절차로 활용하였고, 이때 후보 오픈소스 컴포넌트의 적합성 평가의 기준으로 기능분석표와 기능비교표를 활용하여 오픈소스 후보를 선정하였다. 먼저, 재사용할 오픈소스 컴포넌트의 조사를 위해 도출된 요구사항을 기반으로 <표 6>과 같은 기

능분석표를 작성하였다.



[그림 5] 확장된 재사용 컴포넌트 조사 절차

〈표 6〉 기능분석표

구분		설명	비고
기능	F01.리포트 생성 및 관리	GUI(Graphic User Interface) 방식의 리포트 제작, 출력, 보관 기능을 제공하여야 한다.	
	F02. 파일 저장	조회된 보고서를 텍스트, 엑셀, PDF, HTML 등의 파일로 저장하는 기능을 제공하여야 한다.	
	F03.차트	다양한 유형의 차트 생성 및 출력 기능을 제공하여야 한다.	
	…(중략)	…	
비기능	NFR01. 성능	대용량의 보고서 출력시 속도를 보장하여야 한다.	
	NFR02. 확장성	추후 시스템의 확장이 용이해야 한다.	
	…(생략)	…	

그리고, 가장 잘 알려진 오픈소스 사이트인 소스포지닷넷(http://sourceforge.net)[15]을 통해 기능분석표의 기능들을 키워드로 하고, 개발언어(Programming Language)를 자바로 제한하여 검색하였다.

검색된 오픈소스들의 기능을 분석하여 만들어진 <표 7>과 같은 기능비교표를 기반으로 하여 후보군을 선정하였다. 요구사항의 모든 기능을 포함하고 있는 오픈소스 후보가 없었기 때문에 선정기준은 기능일치율 70% 이상으로 정하였다.

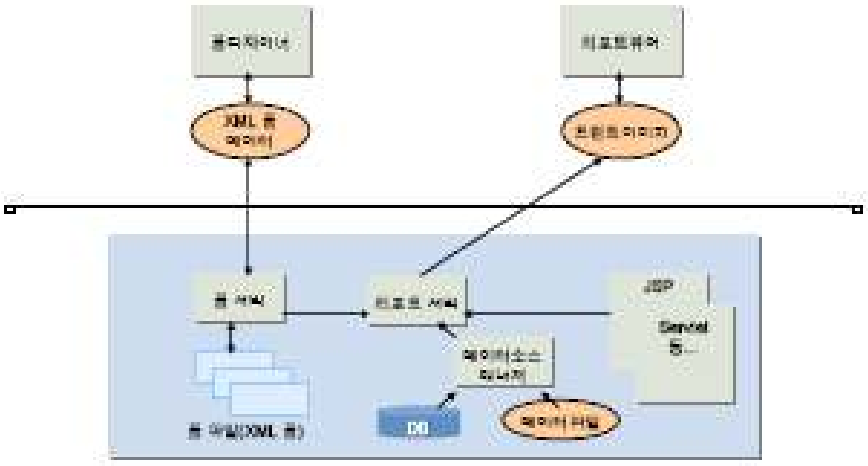
〈표 7〉 기능비교표

오픈 소스	개발할 소프트웨어 기능								일치율
	F01	F02	F03	…			F29	F30	
BPT01	V	V	V				V	V	83%
BPT02							V	V	67%
BPT03		V						V	50%
…(중 략)									…
CRT01	V	V	V				V		80%
CRT02	V	V						V	67%
…(중 략)									….
BPD01			V					V	77%
BPD02	V							V	53%
…(생략)									….

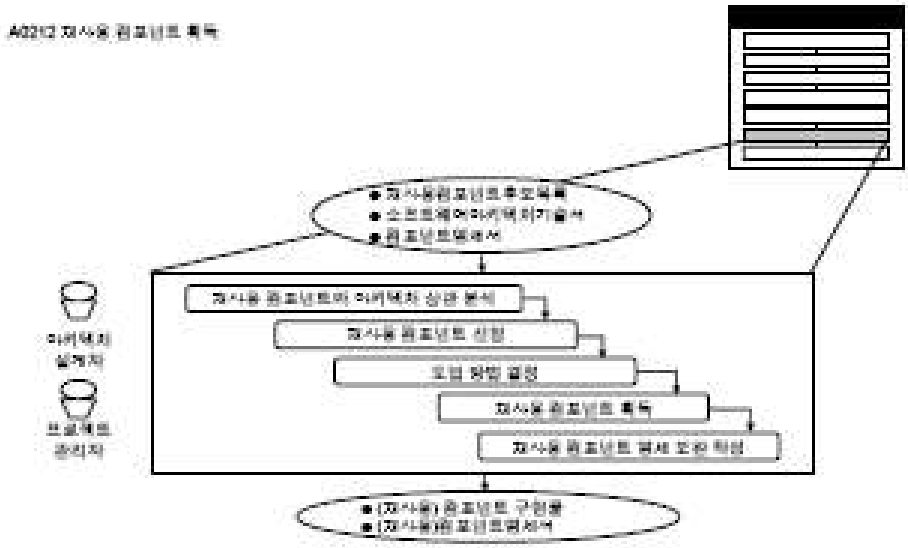
다. 아키텍처 단계(A0000)

마르미-III의 아키텍처 단계에서는 유스케이스 모형과 개념모형을 기반으로 유스케이스 분석을 하고, 시스템 수준의 객체 및 클래스를 도출하고 객체 모형을 작성한다. 또한, 업무 컴포넌트, 소프트웨어 및 기술 아키텍처를 설계하고 재사용 가능한 컴포넌트를 획득하여 시스템 아키텍처를 정의한다. 사례에서는 [그림 6]과 같은 소프트웨어 아키텍처를 설계하였고, [그림 7]과 같이 마르미-III의 '컴포넌트획득(A0212)' 절차를 보완하여, 오픈

소스 후보군에 대한 상세한 정보 수집과 명세작업을 통해 각 오픈소스 후보군에 대한 변경요구사항기술서를 작성하였다.



[그림 6 리포팅툴 아키텍처



[그림 7] 재사용 컴포넌트 확득 절차

<표 8>은 오픈소스 후보군 중의 하나인 RPT01 의 변경요구사항기술서의 예시이다.

〈표 8〉 RPT01의 변경요구사항기술서

구분		일치/불일치	변경요구사항
기능	FR1.리포트생성 및 관리	파라미터 불일치	파라미터 통일
	FR2.파일저장	수정 오피레이션 불일치	수정 오피레이션 변경
	...	...	...

라. 점진적 개발 단계(D0000)

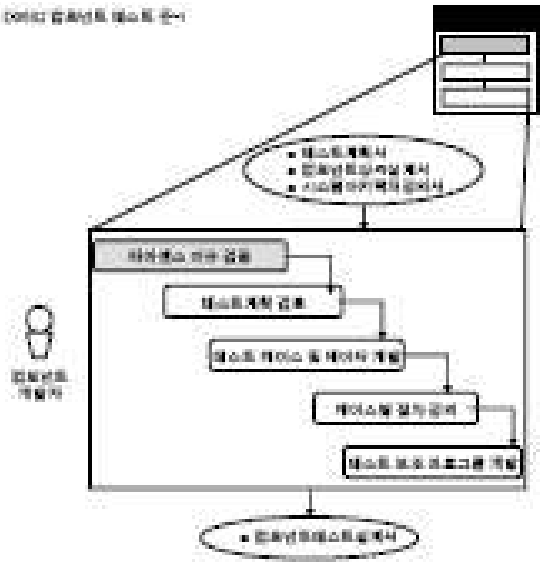
마르미-III의 점진적 개발 단계에서는 미니프로젝트 수행계획에 따라 컴포넌트, 데이터베이스, GUI 등을 구현하여 점진적으로 시스템을 개발한다. 계획된 모든 미니프로젝트를 수행하고 나면, 통합된 시스템의 기능성 및 성능이 사용자 요구사항을 만족하는지 확인하기 위한 시스템 테스트를 수행하고, 그 결과를 평가한다. 이 단계에서는 마르미-III에서 제시하고 있는 컴포넌트의 재사용이나 컴포넌트 조립과정을 적용하지만, 개발과정에서 오픈소스의 변경을 기 적용한 프로젝트에 지속적으로 반영할 것인지에 대한 고려가 함께 이루어져야 한다. 적용 사례에서는 각각의 오픈소스 후보군을 기반으로 하여 변경에 따른 위험이나 비용을 산정하여 오픈소스 후보군 중에 재사용될 오픈소스를 결정하였다. 후보군인 RPT01를 핵심적인 리포트 서버 기능에 적용하고, 서식 디자인 기능을 가지고 있는 후보군인 RPD01를 수정, 통합하기로 하였다. 그리고, 차트 컴포넌트는 CRT01을 사용하기로 하였고, 이를 위해 <표 9>와 같이 후보 오픈소스에 대한 재사용여부비교를 통해 재사용형태를 결정하였다. 선택한 오픈소스는 파일 저장 기능 측면에서의 요구를 완전하게 충족시킬 수 없었다. 그러므로, 획득

한 오픈소스의 아키텍처와 소스를 분석하여, 변경 계획에 따라 수정한 후, 리포트 서식 작성기와의 인터페이스를 정의하고 새로운 기능 및 개선된 코드를 통합시켰다.

〈표 9〉 재사용여부비교표

구분		대상	재사용형태	재사용 여부
기능	F01.리포트생성 및 관리	RPT01	프레임워크/소스코드	O
	F02.파일 저장	RPT01	소스코드	X
	F03.차트	CRT01	컴포넌트	O
	…	…	…	O

또한, [그림 8]과 같이 '컴포넌트 테스트 준비(D0602)' 과정에 컴포넌트 소스 코드에 대한 라이센스 이슈 검증의 절차를 추가하여 수행하였다.



[그림 8] 변경된 컴포넌트 테스트 준비 절차

마. 인도(T0000) 및 종료(E0000) 단계

마르미-III의 인도 단계에서는 개발자 환경에서 개발된 결과물이 컴포넌트인 경우에는 컴포넌트 리포지터리에, 컴포넌트 기반 시스템인 경우에는 실제 시스템이 운영될 사용자 환경에 설치, 승인을 얻고 프로젝트의 모든 전달물을 사용자에게 전달 인계하고, 종료 단계에서는 프로젝트 결과를 평가하고 종료를 보고한다. 이 단계에서는 추가적으로 종료 후 사후관리를 고려할 수 있다. 예를 들어, 개발된 프로젝트를 공개(아웃바운드)하여 기존 프로젝트 커뮤니티의 관심을 유도하고, 그들의 코드에 기반해 상업용 애플리케이션을 개발하였지만, 이 프로젝트에 대해 그들이 규정한 라이센스를 준수할 것이라는 사실을 각인시키면서 지속적인 지원과 확산을 이끄는 '커뮤니티 기반 개발 전략'을 활용할 수 있다.

2.3.3. 적용 평가

지금까지 오픈소스 컴포넌트를 활용하기 위한 절차를 10개의 활동으로 정의하고, 이를 컴포넌트 기반의 개발방법론인 마르미-Ⅲ에 적용, 실제 프로젝트에 활용한 예를 제시하였다. 제안한 절차(활동)는 <표 10>과 같이 마르미-Ⅲ의 단계, 활동 및 산출물을 보완하여 오픈소스 컴포넌트를 효과적으로 조사, 선정, 변경하여 적용할 수 있는 실용적인 프로세스로 활용될 수 있다. 또한, 오픈소스 컴포넌트를 기반으로 시스템을 구축하는 것은 비용을 절감하며, 일정을 앞당기고, 제품의 품질을 개선시켜준다. 사례로 제시한 프로젝트에서는 당초 4명 이상의 개발자와 6개월 이상의 인력(24인/월) 비용이 예상되었으나, 오픈소스 컴포넌트를 활용함으로써 12인/월의 노력으로 고성능의 제품을 개발할 수 있었다. 반면, 사용 가능한 오픈소스의 선정 및 이의 분석과 수정, 통합에 따르는 비용과 일정 등을 비롯한 많은 부분에서 다양한 변수들이 존재하고, 이러한 변수들은 오픈소스 커뮤니

티와 같은 외부적인 요인들에 영향을 받는 만큼 개발 절차의 복잡성을 더욱 심화시켰다. 특히, 오픈소스의 선정시 제공되는 문서들이 많지 않고, 그 명세들을 신뢰할 수 없는 경우가 종종 있었기 때문에 이 정보들만으로 선택하기에는 무리가 있었다. 따라서, 오픈소스에서 제공되는 소스를 분석하여 설계 명세를 추출할 수 있는 역공학 도구들에 대한 활용과 다양한 적용 사례들을 통한 절차와 기법의 상세화가 필요하다.

〈표 10〉 제안한 절차와 마르미III의 연관 관계

마르미III 단계	마르미III 활동	제안 절차(활동)	관련 산출물
P000 프로젝트 시작	P010 프로젝트 준비	-	-
	P020 프로젝트 계획	활동 1 : 오픈소스활용 전략 수립	프로젝트수행 전략기술서, 위험분석서, 프로젝트수행계획서
		활동 2 : 소요자원산정	프로젝트수행계획서
		활동 3 : 라이센스정책수립	프로젝트수행계획서, 위험분석서
R000 요구 획득	R010 요구사항 이해	-	-
	R020 요구사항 정의	-	-
	R030 개발전략 수립	활동 4 : 수집	오픈소스리스트, 기능분석서
		활동 5 : 태깅	오픈소스후보리스트
		활동 6 : 개정	오픈소스선정리스트
A000 아키텍처	A010 요구사항 분석	-	-
	A020 아키텍처 설계	활동 7 : 변경명세화	재사용여부비교표, 변경계획서
		활동 8 : 획득	오픈소스컴포넌트, 재사용오픈소스컴포넌트명세서
	A030 아키텍처 프로토타이핑	-	-
	A040 아키텍처 전개	-	-
	A050 테스트 및 견본계획	-	-
D000 점진적 개발	D010 요구사항 및 아키텍처 정제	-	-
	D020 컴포넌트기본설계	-	-
	D030 컴포넌트상세설계(NET)	-	-
	D030 컴포넌트상세설계(J2EE)	-	-
	D040 컴포넌트 및 시스템구현	활동 9 : 변경	재사용오픈소스컴포넌트
	D050 지원서 개발	-	-
	D060 컴포넌트 테스트	활동 10 : 검증	컴포넌트테스트결과보고서
	D070 통합테스트	-	-
	D080 시스템테스트	-	-
T000 인도	T010 사용자 교육	-	-
	T020 시스템 설치	-	-
	T030 인수테스트	-	-

2.4. 결 론

본 연구에서는 오픈소스 컴포넌트를 활용한 소프트웨어 개발의 절차를 제시하고, 이를 컴포넌트 기반 개발 프로세스(마르미-Ⅲ)에 적용하였다. 또한, 실제 프로젝트에 적용하여 비용을 절감하고 기간을 단축함으로써, 제시한 절차의 적용 가능성을 확인할 수 있었다. 그동안 소프트웨어 개발에 오픈소스를 활용하기 위한 절차나 방법에 대한 연구결과가 미흡한 상황에서, 본 연구는 실제 산업현장에서 활용 가능한 실용적인 지침이 될 것이다. 그러나, 오픈소스는 아키텍처(프레임워크), 소프트웨어 패키지, 라이브러리, 컴포넌트 등 그 유형과 레벨이 다양하게 존재하고, 소프트웨어 패키지를 활용하는 경우와 컴포넌트를 활용하는 경우의 절차가 서로 다를 수 있다. 따라서 각각의 경우에 적합한 세부적인 활용 프로세스를 연구할 필요가 있다. 또한, 보다 정확하고 객관적인 오픈소스 선정을 위한 메트릭에 대한 연구가 필요하다.

향후, 좀더 구체적이고 실용적인 오픈소스 소프트웨어 재사용 프로세스의 개발과 적용에 대한 추가적인 연구를 수행할 것이다.

참 고 문 헌

[1] 송위진, "오픈소스 소프트웨어의 기술혁신 특성 : 리뷰", 「한국기술혁신학회지」, 제5권, 제2호(2002), pp.212-227.

[2] 한국소프트웨어진흥원, 「공개소프트웨어 도입 가이드라인 연구」, 2003.

[3] 김덕수, "효과적인 오픈 소스 소프트웨어 개발을 위한 프로젝트 관리 프로세스", 「한국정보과학회지」, 제20권, 제12호(2002), pp.30-42.

[4] Martin Fink, 「리눅스와 오픈소스의 비즈니스와 경제학」, 조광제 역, 영진닷컴, 2005.

[5] 한국전자통신연구원, 「마르미III-v3.0」, 2002.

[6] Research, FLOSS(Free/Libre and Open SourceSoftware : Survey and Study), FINAL REPORT, 2002.

[7] Feller, J. and B. Fitzgerald, A Framework Analysis of the Open Source Software Development Paradigm, Proceedings of the 21st International Conference in Information Systems, 2000.

[8] LEF REPORT, Open Source : Open for Business, Leading Edge Forum, 2004.

[9] Madanmohan T. R. and Rahul De', Open Source Reuse in Commercial Firms, IEEE. November/December, 2004.

[10] Michel Ruffin and Christof Ebert, Alcatel, Using Open Source Software in Product Developmen t : A Primer, IEEE, January/February 2004.

[11] James W. Paulson, Giancarlo Succi, Armin Eberlein, "An Empirical Study of Open-Source and Closed-Source Software Products", IEEE TRANSACTIONS ONSOFTWARE ENGINEERING, Vol.30, No. 4(2004).

[12] MICHAEL J. KARELS, Commercializing Open Source Software, ACM QUEUE, July/August 2003.

[13] Linux, http://linux.org/

[14] Freshmeat.net, http://www.freshmeat.net

[15] Sourcefoge.net, http://www.sourceforge.net

3. 공개소프트웨어 활성화를 위한 수요창출 사례연구

한국소프트웨어진흥원 남일규·이재덕·김태열

3.1. 개 요

공개소프트웨어는 많은 장점에도 불구하고, 그동안 국내에서는 활성화가 많이 늦어진 것이 사실이다. 그 원인은 무엇보다도 사용자가 대부분의 시스템을 유닉스나 윈도우로 구축한 관성을 유지하고 있었기 때문이다.

한국소프트웨어진흥원에서는 적극적인 홍보활동을 통하여 리눅스를 기반으로 한 시스템 구축을 권고하였다. 이에 대한 결실로 교육부의 신NEIS 시스템 단독서버 2300여대를 성공적으로 리눅스로 구축 운영[2006.3]할 수 있었다. 또한, 예산처에서는 예산절감과 효율적인 정보시스템 구축을 위한 수단으로 리눅스를 인식하여 2006년 정보화예산에 37개 사업을 리눅스로 구축하게 유도하였다.

레퍼런스 구축을 목적으로 하는 시범사업을 통하여서는 정부부처 공공기관에 리눅스를 기반으로 하는 시스템을 구축하여 리눅스 성공사례를 확보함으로써 공공기관에서 리눅스를 꺼려하는 분위기를 일소하기 위하여 노력하였다. 시범사업의 한 예로 인터넷 수능방송은 고화질(600K) 시스템을 리눅스로 구축 운영함으로써 기존에 유닉스로 구축된 저화질(300K) 시스템과 병행하여 사용함으로써 리눅스의 강점(TCO, 안정성, 호환성)을 각인시켜 주고 있다. 또한, 데스크탑 분야의 윈도우 고착화를 해소하기위한 연구의 일환으로, 한국소프트웨어진흥원 전체가 리눅스 데스크탑을 사용하여, 장단점 파악과 확산을 위한 기초 확립에 분주하고 있다. 그 예로 데스크탑 분야는 웹 접근성, 오피스 호환성등 주요 문제를 해결하기 위한 노력을 하고 있다. 데스크탑은 범용으로 사용하는데 문제가 되는 한계를 극복

하기 위한 다각적인 노력과 더불어, 단순기능을 사용하는 분야에 우선 적용하여 그 사용성을 먼저 검증받는 순차적인 접근이 필요한 것이 현실이다.

이외에도 그간 특정시스템에 익숙한 사용자들이 습관적으로 사용하여 공정경쟁에 위배되어 왔던 제도 등 을 개선하는데 노력을 하고 있다. 이러한 제반 환경개선과 소프트웨어의 패러다임이 바뀌면서, 공개소프트웨어는 앞으로 새로운 개념의 사업을 태동시키는 핵심키워드가 될 것이다.

3.2. 공개소프트웨어 활성화 정책의 추진 배경

리눅스로 대변되는 공개소프트웨어는 국가적 차원의 필요성과 세계적인 기술발전 추세에 발맞추어 국내에서도 2003년 이후 정부의 적극적인 추진으로 활성화 되기 시작하였다[1]. 소프트웨어산업은 그 자체로도 시장규모가 크고 성장률도 높은 고부가가치산업이기에 IT839정책의 핵심적인 역할을 수행하고 있으며 21세기 국가경쟁력의 핵심으로 부각되고 있다. 그러나 국내 소프트웨어산업은 해외 글로벌 기업에 의존하고 있고, 자체 수익구조가 매우 열악한 상황이다. 이는 지난 수년간 HW와 통신 인프라에 치중되어 왔던 정책적 편향성과 소수 글로벌 기업에 의한 시장선점과 진입장벽 등이 주요한 원인으로 작용한 것으로 볼 수 있다.

시장조사 전문기관인 IDC(2005년)[2]에 따르면 리눅스의 신규 서버시장 점유율이 '09년 26.7%에 이르는 등 MS 등 소수 글로벌 기업이 주도하던 기반소프트웨어 시장이 리눅스 등 공개소프트웨어의 약진으로 재편되는 추세를 보일 것으로 예측하고 있다. 공개소프트웨어 정책과 관련해서 이미 세계 각국에 서는 특정 벤더에 대한 편향성을 완화하고 경제·기술적 장점을 활용하기 위하여 전자정부사업 등 공공분야에 공개소프트웨어 이용을 적극 장려하고 있으며, 유럽 각국은 소프트웨어기술 주도권 확보 차원

에서 공개소프트웨어 활성화 전담기관을 설립하고[3]1) 관련 이용활성화, 기술개발 등을 추진 중에 있다. 또한 중국, 일본, 동남아시아 국가들도 IT 예산절감, 자국 산업보호 등을 목적으로 전자정부 구축 등에 공개소프트웨어 활용을 적극 추진 중에 있다[4].

국내 공개소프트웨어 보급·확산 정책은 국내업체의 소프트웨어원천기술 확보 및 국제경쟁력 제고를 위한목적으로 추진되고 있다. 세부적으로 보면 첫째, 소프트웨어가 글로벌 소수 업체에 의해 시장이 지배되는 독점현상을 해소하여 공정경쟁 환경조성 둘째, 공개된 소스코드를 통한 최신기술 습득이 가능해 선진국 및 기업과의 기술격차 해소 셋째, 독점제품에 종속되지 않아 소비자 선택에 다양성 강화로 소프트웨어제품 경쟁을 통한 고품질, 합리적 가격구매가 가능 넷째, 특정업체의 시스템에 종속되어 기밀유출 위험에 노출되지 않는 투명하고 안정된 기술 보안성 강화등에 주요한 목적을 갖고 있다.

3.3. 공개소프트웨어 수요창출 추진 사례

3.3.1. 공개소프트웨어 활성화 정책의 추진 과제

국내에서는 공개소프트웨어가 갖는 장점 및 세계적인 변화추세에도 불구하고 공개소프트웨어에 대한 신뢰도 부족, 레퍼런스 부족 등으로 인해 시장에서의 확장에 어려움을 겪고 있었으며 이러한 문제점을 해결하기 위하여 2004년부터 본격적으로 추진된 공개소프트웨어 활성화 지원 사업은 1) 인식개선 홍보 활동, 2)이용활성화 제고를 위한 best practices 산출, 3) 기반 공정 환경 조성을 위한 제도 개선 활동, 4) 국제협력강화 등을 주요 내용으로 공개소프트웨어활성화를 위한 수요창출에 초점을 두고 추진되고 있다.

3.3.2. 추진 사례

가. 인식개선 홍보 활동의 전개

국내에서 공개소프트웨어 확산이 느리게 진행된 가장 큰 이유는 공개소프트웨어에 대한 막연한 불신과 오해 즉, ‘공개소프트웨어는 성능이 미흡하다.’, ‘공개소프트웨어는 기술지원체계가 불완전하다.’, ‘공개소프트웨어는 성공적인 도입사례가 부족하다.’, ‘공개소프트웨어는 보안성이 약하다.’ 등의 인식이 만연해 있기 때문이다. 이는 공개소프트웨어에 대한 정확한 정보 전달의 부족 즉, 불완전한 정보로 인한 것이기 때문에 이를 해소하기 위해 공개소프트웨어를 제대로 알리기 위한 다양한 마케팅 활동이 필요하였다.

이를 위해, 공개소프트웨어에 대한 편견과 오해를 종합적으로 해소하기 위해 ‘공개소프트웨어 문답집’을 발간하고, 공개소프트웨어가 성공사례가 부족하다는 오해를 해소하기 위해 대표적 공개소프트웨어인 리눅스 도입 사례에 대한 성공사례집을 발간하였으며, 발간된 사례집은 약 2,500명의 공공기관 정보화담당자를 대상으로 배포하여 그간의 부족했던 정보를 해소하는데 주력하였다. 또한, 공개소프트웨어 정책 필요성 및 도입 방법론 전파를 위한 컨퍼런스 등 마케팅 활동과 함께, 공개소프트웨어 기업을 알리기 위한 “공개소프트웨어 기업과 제품정보 가이드”를 발간하여 기존 공개소프트웨어 인식을 전환할 수 있는 기반을 마련토록 추진되어 왔다.

또한, 공개소프트웨어 도입 이후 공개소프트웨어 활용 선례부족과 기술지원에 대한 불안감 등에 따른 도입 저해요소를 제거하고자 ‘공개소프트웨어 기술지원센터’ 를 설치(2005년)·운영하여 기술적 지지 기반의 마련과 기술지원 체계 안정화를 추진하였다. 공개소프트웨어 기술지원센터는 공개소프트웨어 도입 공공기관에 대한 기술지원, 공개소프트웨어 및 공개소프트웨어기반 솔루션 개발업체에 대한 기술개발 지원 등 공개소프트웨어 공

급자 및 수요자에 대한 기술지원을 동시에 추진함으로써 안정적인 공개소프트웨어 수요기반을 확보 하는데 주력하고 있으며 또한, 이를 통해 공개소프트웨어 개발기업 및 기술지원 기업의 육성의 기반을 마련하게 되었다.

나. Best Practices 구축

공개소프트웨어는 소스코드가 공개되어 대선진국 기술격차를 해소할 수 있는 기회를 제공할 수 있다는 점에서 매우 중요한 의의를 갖고 있으나, 수요기반이 매우 부족하여 기업의 공개소프트웨어에 대한 투자가 매우 부족하다. 특히, 공개소프트웨어 기반 정보시스템 구축 시장은 시장형성단계에 있으므로 공공기관을 중심으로 한 선도적인 수요창출이 요구되어 진다[5].

1) 대규모 공공 정보화사업 공개소프트웨어 도입 확산

대규모 공공 정보화사업은 국가정보화를 위해 각 부처에서 의욕적으로 추진하는 사업으로 과거에는 대부분 유닉스 기반의 시스템 도입을 추진하는 관례로 특정 스팩에 제한적이며 우수 국산소프트웨어가 접근하기에 어려움을 보여 왔던 영역이었다[6]. 이러한 공개소프트웨어 및 국산소프트웨어의 진입장벽을 없애기 위해 특정 스팩의 명시를 없애고 공개소프트웨어인 리눅스 기반의 시스템 도입을 유도함으로 국가적인 예산 절감 효과, 안정성 확보 및 우수 국산소프트웨어기업 제품이 활동되는 공정 경쟁 환경을 유도토록 추진되었다.

교육부에서 추진된 '새로운 NEIS 시스템 구축 사업' 추진시 공개소프트웨어 관련 기술자문을 실시하여 리눅스가 도입('05. 8월)되었다. 당시 중소형 서버라는점과 기술지원 문제, 보안성 등이 핵심 이슈사항이었다. 중소형 서버문제는 64bit용 국산리눅스를 앞당기는 촉매가 되었으며, 기술지

원문제는 대형 OS업체와 대형SI업체의 지원으로 그 우려를 불식하였으며, 보안성은 오히려 강화될 수 있다는 인식개선이 이루어졌다. 이러한 과정을 거쳐 단일 정보시스템으로는 세계 최대 규모인 2,300여대 서버 운영체계에 국산 리눅스가 도입[7]되었다는 점에서 세계적 주목을 받게 되었다. 또한, 미들웨어 및 애플리케이션이 100% 국산소프트웨어로 도입되어 공개소프트웨어기반의 국산소프트웨어육성정책이 가시적 성과를 나타내었다. 그러나, 이러한 많은 의의에도 불구하고 저가입찰의 문제가 발생하였으며, 이는 또한 소프트웨어의 생태계를 재조명하고, '소프트웨어 제값주기'에 좀 더 힘을 싣는 계기가 되기도 하였다.

또한, 정보화예산혁신을 위해 기획예산처 주도로 2006년 각 부처 추진 정보화사업 중 37개 사업(예산 규모 약750억원)을 리눅스로 도입토록 예산처와 TFT를 구성하여 추진하였다. 비록 전체예산에서 차지하는 비중은 미미하나, 리눅스가 가격대비 성능이 뛰어나서 예산절감차원에서 추진한다는 그 의미가 있다고 하겠다. 신규 사업 위주로 주로 리눅스 채택에 검토가 있었으며, 이를 통하여 절감된 예산은 각 부처 IT분야 예산으로 전용할 수 있게 하여, 예산을 효율적으로 활용할 수 있게 하였다.

2) 시범사업을 통한 안정성 검증 및 이용확대 기반 조성

공개소프트웨어 시범사업은 공개소프트웨어 도입에 소극적인 공공기관의 도입확대를 통한 성공사례를 확보하는 목적으로 추진되었으며 '04년 8개, '05년 11개의 시범사업이 추진되었고, '06년에도 8개전후로 추진 예정이다. '04년 시범사업의 경우는 서버 시스템 구축으로 일반적인 웹 기반의 서버환경구축이 주를 이루었으며, 대표적인 것은 한국교육방송공사 인터넷 수능방송, 전북 소방본부 119 구조시스템, 영천시청의 자료관 시스템, 대전시청의 보육ASP서비스 등이다. 한국교육방송공사의 경우는 국내 최대 규모인 온라인 교육시스템을 구축하여 기존의 저화질(300K) 시스템은 유닉

스로, 고화질(600K) 시스템은 리눅스로 구축하여 현재 운영 중에 있으며, TCO와 안정성, 호환성 등에 장점이 있는 것으로 자체 분석되고 있다. 또한, 동영상 국제표준인 MPEG을 사용함으로써, 특정업체의 파일형식에 디지털 콘텐츠가 종속되지 않도록 하는 목적도 갖고 있었다. 전북소방본부의 경우도 서버를 이전함으로써 TCO와 안정성에 더 효율적인 구축과 운영이 실현되어 있다. 또한, 영천의 자료관시스템은 유사 시스템을 타기관에서도 적용하는 예가 확보되었다.

[표 1] '05년도 공개소프트웨어 시범사업 추진내역

순번	대상 기관	시범사업명
1	공군본부	공개소프트웨어기반 국방정보체계 통합툴 개발 및 시범체계 구축
2	보건복지부	공개소프트웨어기반 전자문서 협업시스템 및 모바일 GW 구축
3	통일부	공개소프트웨어기반 통일업무정보 공유시스템 구축
4	민주평통 사무처	공개소프트웨어를 활용한 디지털멀티미디어 통합 홍보시스템 구축
5	전라북도청	공개소프트웨어기반의 공문서 자동집계 및 DB통계분석 시스템 구축
6	창원대학교	공개소프트웨어기반의 대학캠퍼스 시설물 및 조경관리 시스템 구축
7	식품의약품안정청	리눅스 사용자를 위한 홈페이지 환경 개선
8	환경부	리눅스 사용자를 위한 홈페이지 환경 개선
9	한국소프트웨어진흥원	데스크탑 리눅스 기능 개선
10	정통부 지식정보센터	공개소프트웨어기반의 우체국 인터넷뱅킹 시스템 구축
11	원주시청	행정정보포탈시스템 구축

[표 1]의 '05년 시범사업 경우 우정사업본부의 '우체국 인터넷뱅킹 시스템 구축'을 통해 윈도우 운영환경과 인터넷 익스플로러 환경에서만 동작하던 인터넷 뱅킹업무를 리눅스 PC 사용자도 활용할 수 있도록 하여 리눅스

PC 도입장벽을 제거하는 등 공개소프트웨어에 대한 단순한 불안감이나 기술적인 안정성에 대한 우려를 종식시킬 수 있는 대표적인 사례를 확보하였다. 물론, 아직 시장에서 리눅스 데스크탑 사용자가 많지 않기에 소수사용자의 정보이용 평등권을 확대하는 필요성을 확인하는 성과도 있었다. 또한 한국소프트웨어진흥원의 업무용PC에 리눅스를 시범 적용함으로 사무환경에 리눅스 도입 가능성 제시로 향후 타 기관 및 기업에 도입할 수 있는 계기를 마련하였다. 특기할 사항으로는 '05년 추진 시범사업의 경우 OS와 소프트웨어가 국산화되는 등 국산소프트웨어 제품의 적용으로 우수성 및 안정성에 대한 확신을 가져오는 계기가 되었다.

'06년 시범사업은 이외에도 지자체나 학교 전체 시스템을 공개소프트웨어로 도입하려는 의지를 갖고 있는 지자체나 학교에게 일부 비용을 지원하는 공개타운이나 공개대학을 지원한다. 지자체나 학교의 전체시스템을 리눅스에서 구축 운영되기 위해서는 최소 5년 이상의 기간이 소요될 것으로 예측하고 있다. 그리고, 데스크탑 시범사업은 범용 데스크탑의 보급보다는, 단위 기능을 리눅스 데스크탑으로 구현하려는 기관에게 시스템 구축을 주로 지원한다. 단위기능에서 안정성과 신뢰성을 확보하여 많은 성공사례를 만들고, 이를 통해 범용데스크탑을 확산하는 단계적인 전략이 필요한 것으로 보고 있다. 즉, 각 기관의 특수사용 목적의 기능을 우선 리눅스 데스크탑을 사용하고, 리눅스 데스크탑이 갖는 한계인 네트워크 효과와 LOCKOUT 효과를 해결하는 다각적인 노력이 범용 데스크탑을 리눅스로 활성화하는 사전준비가 될 것이다.

다. 공개소프트웨어 이용촉진을 위한 법·제도 개선 및 가이드 마련

국내 공개소프트웨어 활성화 추진시 제도적 제약문제는 다양한 법률, 규칙, 지침 등에서 간접적으로 도입을 가로막는 요인이 있어 왔다는 점이

다. 예를 들면, 정보화시스템 구축 등 RFP 상의 유닉스, 오라클 등 특정 제품 및 기술스펙을 지정하는 관행 등이 그것이며, 이는 불공정 경쟁 환경으로 볼 수 있다. 즉, RFP 상에 관계형 DB로 표현하면 될 것을 사용자가 시장점유율이 높은 특정회사의 DB 제품명을 직접 거론하는 것이 그 예 일 수 있는데, 특수한 경우는 그 DB만 가능한 경우일 수도 있으나, 대부분의 경우는 관성적인 행태일 경우가 많은 것으로 판단되어 지고 있다.

따라서, 전자정부를 위시한 공공부문 정보화 사업 구축시 공개소프트웨어 진입 장애요인으로 작용하는 제도 및 관행을 개선 토록하는 노력과 나아가 정보화 수준 평가지표 등에 적극적으로 공개소프트웨어 관련지표를 반영함으로써 적극적 활용을 유도하는 노력이 추진되었다. 공정경쟁 촉진을 위하여 정부혁신지방분권위원회에서 각 부처의 전자정부사업 추진 시 공개소프트웨어에 대한 도입을 권고하는 권고안을 마련하였고('04. 12)이의 안내서라 할 수 있는 '공개소프트웨어 기반 정보시스템 구축 사용자 가이드(행정자치부, 정보통신부)'를 작성·배포하였다.

또한, 윈도우 데스크탑 브라우저에 적합토록 구성된 각 행정기관의 홈페이지에 대해서도 리눅스 데스크탑 브라우저나 멕켄토시용 브라우저 등에서도 웹 접근성을 보장토록 규정하는 지침 마련(05.5)과 함께 '행정기관 홈페이지 평가지표'에 구현여부를 평가토록 반영(05.10)하였다. 이와 함께 홈페이지 개발자 대상의 '실전 웹 표준 가이드' 제작(05.12)[8]을 통해 다양한 운영체계 환경에서 유사한 접근성을 갖도록 표준을 준수하는 개발 방법을 제시하였다.

라. 공개소프트웨어 이용촉진을 위한 국제협력 강화

국내 공개소프트웨어 활성화 추진과 더불어 한국, 중국, 일본 3국은 공개소프트웨어한중일 포럼을 정기적으로 갖고, 3국의 공개소프트웨어정보

교류와 활동을 하고 있다. 3국은 특히, 민간 3국포럼을 진행하여 워킹그룹을 구성하고 있으며 기술개발테스트분과(Technolgy Development & Assessment), 인력양성분과 (Human Resource Development), 표준화분과(Standardization)를 통하여 실질적인 협력을 구축해나가고 있다. 기술개발테스트분과에서는 리눅스 데스크탑, 서버 테스트, SecureOS등이 연구되고 있으며, 인력양성분과에서는 3국 공동 커리큐럼 연구, 공개소프트웨어 경진대회 등이 수행되었으며, 표준화분과에서는 입력방법(Input Method), 웹 접근성 연구 등이 수행되었다. 향후 이러한 연구를 토대로 3국 공동 프로젝트가 다각적으로 펼쳐질 것이다. 이외에도, 유럽, 동남아, 기타 나라와 공개소프트웨어 국제 정보교류를 통하여 상호협력과 공개소프트웨어에 대한 기여를 지속적으로 추진하고 있다.

3.4. 공개소프트웨어 수요창출 추진 결과 평가

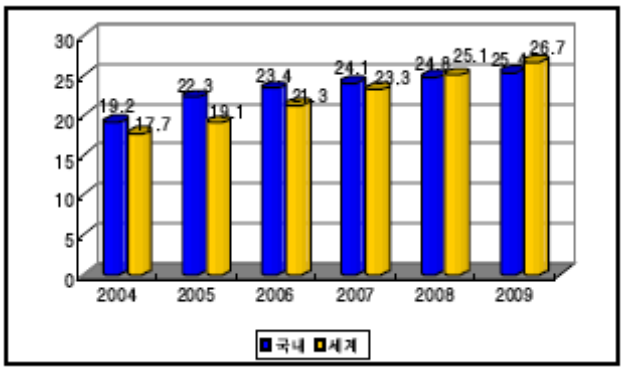
3.4.1. 그간의 추진 성과

서두에서 언급하였듯이 공개소프트웨어 활성화 정책의 주요한 추진 목적 중 하나는 국내 소프트웨어시장이 소수의 특정 글로벌 벤더에 의한 시장독점과 그에 따른 공정경쟁을 저해하는 문제를 해결하는 것이며 이러한 문제의식에서 출발한 1차 활성화 정책은 공개소프트웨어로 대표되는 리눅스 서버시장 확대, 리눅스에 대한 인식전환, 그리고 이를 위한 공공부문에서의 이용사례 확보에 있어 소기의 목적달성을 이루었다고 보여 진다.

'04~'05년 공개소프트웨어 활성화 정책 추진 결과 국내 리눅스 시장 성장율은 그림 1과 같이 세계시장 성장률을 추월하고 있으며 지속적인 정책추진에 따라 공개소프트웨어 시장의 주도적 위치 확보가 가능케 되었다. 공개소프트웨어 도입의 가장 큰 걸림돌로 작용되어온 공개소프트웨어에 대

한 막연한 불신과 오해를 해소하는 성과는, 공공부문을 타켓으로 정부정보화사업을 통한 리눅스 도입사례를 확대해 나감으로써 국내 및 세계 최대 규모의 리눅스 레퍼런스를 구축하는 등 세계적으로 급부상하는 공개소프트웨어 시장에 대한 이니셔티브를 확보하는 전기를 마련하게 되었다. 이는 공개소프트웨어의 안정성과 비용절감 등 막연한 불신을 해소하는 역할을 하였으며, 또한 공개소프트웨어 기술지원센터 운영을 통해 보다 신뢰감 있는 공개소프트웨어 도입을 촉발할 수 있게 되었다. 이러한 인식전환의 노력은 아직은 열악한 리눅스 데스크탑에 대한 전망 [9,10]에 대한 관심을 불러일으키는 계기가 되었다.

또한, 공공 정보화사업 및 시범사업 등을 통해 서버 영역에서의 국산 하드웨어, 국산소프트웨어의 도입 상승 효과를 가져왔다. 즉, 기존의 유닉스, 윈도우 체계의 시스템에서 리눅스 기반으로 유도됨으로 DBMS, 미들웨어, 서버용 솔루션에서 상대적으로 국산 도입비중이 증가하여 국내소프트웨어 기업의 새로운 시장영역을 만들어주는 역할을 이루었다.



[그림 1] 리눅스 서버 운영체제 시장 점유율 전망

3.4.2. 최근 동향 및 과제

공개소프트웨어 서버 분야에서는 그림 2와 같이 중소형 서버에서 중대

형 서버로 리눅스가 시장을 확대할 수 있는 가능성을 볼 수 있다. 그 첫째가 4way 서버에서 8way, 16way로 그 성능이 비약적으로 발전하고 있으며, 중대형 처리의 필수라 할 수 있는 64bit 컴퓨팅이 상용화 되었다. 또한, Dual Core 기능이 상용화되어 짐으로써 성능이 일취월장하였다. 이러한 제반 여건이 인프라스트럭쳐 분야에 집중되어 있던 리눅스가 DB서버등과 같이 대형서버에 사용되는 조건이 조성되고 있고 실질적으로 최근에 DB서버로 리눅스상에서 구축되는 경우가 많아지고 있다. 따라서 앞으로 지속적인 서버분야의 리눅스의 약진은 계속 될 것이며, 그 방향은 중소형서버에서 대중소 서버 전체를 아우르는 형태로 진행될 것이다. 이는 기존에 중대형의 영역이었던 유닉스에 많은 영향을 줄 것이며, 이는 메인프레임까지 일정정도 영향을 미칠 것으로 판단되어 진다. 규모면과 더불어 기능면에서는 비즈니스 어플리케이션 서버와 데이터베이스 서버로 약진할 것이며, 중소형 서버에서는 GUI기능이 강화되면 윈도우시장에 위협을 줄 것으로 보고 있다.

반면 공개소프트웨어 데스크탑 분야에서는 가트너 그림을 참고하면, 엔트리용으로 사용이 안정화되어 있으나, 범용 데스크탑 사용은 아직까지는 일부 기술 선구자가 사용하고 있다. 범용으로 데스크탑이 사용되기 위해서는 오피스 호환부분, 웹 접근성해소 부분, 각종디바이스 드라이버 지원 등 많은 산적한 문제의 해결 방안이 사전에 준비되어야 할 것이다. 이러한 현실에서 우리는 국가사이버안전센터에서 마이크로소프트가 2006년 7월1일자로 윈도98에 대한 기술지원서비스를 중단함을 발표함에 따라 이에 대응하기 위하여 분주하게 움직이는 것을 주의 깊게 봐야한다. 국가정보원 국가사이버안전센터에 따르면 국내PC가운데 약 350만대가 아직도 윈도98을 사용하고 있다고 한다. 이런 현실에서, 일방적인 업체의 통보에 국가정보시스템이 보안에 노출될 위험을 원천적으로 안고 있는 상황인 것이다.

편리성과 익숙함 때문에, 관성에 의한 시스템 독점은 결국 한 국가의 시스

템이 한 업체의 이익에 의해 좌우되는 위험성을 항상 경계하고 그에 대한 대안을 준비하여야 하는 필요성을 느끼게 하는 좋은 사례이다.

[그림 2] Hype Cycle for Open Source software

서비스 측면에서는 그간의 공개소프트웨어 비즈니스모델에 방관하던 SI업체와 솔루션업체 등이 비즈니스모델로 소프트웨어, 서비스를 위한 준비를 할 것이며,이는 아직까지 기술지원에 대한 부족 등이 큰 활성화 저해요소로 남아있는데 이에 대한 해결의 열쇠를 갖고 있을 것이다. 그럼에도 불구하고, 아직까지 리눅스 관련 시장이 충분히 형성되지 않았으며 서비스 모델로써 공개소프트웨어가 정착하지 못한 것으로 분석하고 있다. 따라서, 이러한 모델을 발굴하는 노력과 더불어 제도로써 발 빠르게 정착하는 실행력이 필요한 시점이다. 더불어 수요자 및 SI업체 등 국산 서버로 시스템 구성을 하는 사례가 늘어남으로써, 국내 소프트웨어, 하드웨어 산업이 수직계열화를 통한 체질개선의 가능성을 열어 놓았다. 그럼에도 불구하고 외산위주의 시스템 구축을 선호하는 경향이 아직도 대세임을 부인 할 수 없다. 이는 전반적으로 기존 시스템에 익숙해진 사용자의 관행, 국산소프트

웨어에 대한 낮은 신뢰도 등 복합적인 요인 등이 작용하는 것으로 판단되어 진다.

3.5. 결 론

소프트웨어산업은 지식기반경제에 있어 기업과 정부의 혁신 및 글로벌 경쟁력을 좌우하는 핵심인프라로써의 중요성과 IT분야는 물론 각종 기기에 내장되어 제품의 가치 혁신을 촉진하는 등 국민경제에 미치는 영향뿐 아니라 그 자체로도 시장규모와 성장률이 높은 고부가가치산업 임에는 틀림없다. 그럼에도, 아직 국내 소프트웨어산업은 원천기술의 부족, 글로벌 기업의 시장선점 및 내수시장의 불합리 등으로 인해 세계적인 경쟁력을 갖춘 기업이 없는 상태이다. 흔히, 소프트웨어는 네트워크 효과와 lock-in 효과에 기인하여 후발 국가에서는 상당히 어려운 시장구조를 가질 수밖에 없다.

공개소프트웨어 활성화 정책은 이러한 국내 소프트웨어 시장구조의 개편을 통해 소프트웨어원천기술을 확보하는 차원에서 추진되어 왔으며, 적극적인 공공시장에서의 리눅스 도입 추진으로 새로운 수요시장의 창출과 더불어 국산소프트웨어의 시장진입 가능성을 확대해 왔다. 지금까지의 수요창출 노력과 함께 첫째, 리눅스 서버의 안정성 및 성능에 대한 인식 제고로 서버시장 확대를 더욱 가속화할 필요성이 있다. 공개소프트웨어 관련 기술적 혁신 또한 시장을 뒷받침 할 것으로 보인다.

이러한 기회를 잘 활용하여 국내 하드웨어, 소프트웨어산업이 국제적 수준으로 진입할 수 있는 체질개선의 기회로 활용하여야 한다. 둘째, 시범사업을 통해 확보된 데스크탑 PC 적용가능성은 향후 점진적인 확대로 이루어질 수 있을 것으로 보인다. 특히, 단순 목적용, POS용, 특수목적용 등 접근 용이성을 고려한 단계적 접근을 통한 확대 노력이 필요하다. 마지막

으로 모바일 기기, 통신분야 등 임베디드 리눅스 활용 시장은 점차 확대되어 가는 추세에 있어 이에 대한 접근 또한 중요한 과제로 남아 있다.

참고문헌

[1] 한국소프트웨어진흥원, 공개소프트웨어 중장기 기반기술기획 연구, 2003.12.

[2] IDC, Worldwide and U.S. Server 2005-2009 Forecast, 2005.

[3] UN , Policies of united nations system organizations towards the use of open source software(OSS) in the secretariats, 2005.

[4] 한국소프트웨어진흥원, 오픈소스 소프트웨어 연구 보고서, 2002. 12.

[5] STEPI, 공개소프트웨어와 정부정책, 2003.

[6] 한국전산원, 공공부문 정보화지원사업에서 Open Source Software 적용관련 연구, 2003.6.

[7] 한국소프트웨어진흥원, 공개소프트웨어 이용활성화 지원 사업, 2005.12.

[8] 한국소프트웨어진흥원, 실전 웹 표준가이드, 2005. 12

[9] IDC, 국내OS별 데스크탑 시장규모(신규도입율), 2004.6.

[10] Gartner, Hype Cycle for Open-Source Software, 2005.

4. I/O Benchmark Using x345/GPFS/Linux Clients and x345/Linux/NSD File Servers

4.1. Executive overview

The primary focus of these four tests was to evaluate General Parallel File System (GPFS) performance for HPC applications on Linux clusters. The clusters in this test were IBM Eserver xSeries 345-based (x345) systems using IBM TotalStorage FAStT900 disk controllers and EXP700 disk drawers. Each test was conducted on a different cluster network configuration (Gigabit Ethernet (GbE), bonded GbE, 100 Megabit Ethernet (MbE), and Myrinet). Some of these test configurations are officially supported, and others are not.

In this paper, we summarize and interpret the results of these benchmark tests. We also propose untested alternative designs that were suggested by these results. Overall, the results of the tests were positive, most meeting or exceeding expectations. A few others, although below expectations, were acceptable. Only one test, in an unsupported, infrequently-used configuration, yielded unacceptable results.

Equally important to the performance results is that there were no system failures attributable to GPFS or the GPFS/Linux combination. We did experience several node failures on one of the storage server nodes, but the apparent cause was an intermittent adapter failure under heavy loads. This presented no significant

setbacks to the benchmark tests and had the added value of validating the failover features of the storage subsystem.

4.2. System configuration

This benchmark consisted of four tests. Each was associated with a unique cluster network configuration. The node and storage subsystem configuration remained constant across all completed tests.

4.2.1. Components defined

The General Parallel File System (GPFS) for Linux is a high-performance shared-disk file system that can provide data access from all nodes in a Linux cluster environment. Parallel and serial applications can readily access shared files using standard UNIX® file system interfaces, and the same file can be accessed concurrently from multiple nodes. GPFS provides high availability through logging and replication and can be configured for failover from both disk and server malfunctions

High Performance Computing (HPC) is a cluster configuration designed to provide greater computational power than one computer alone could provide. HPC clusters connect the computing powers of the nodes involved to provide higher scalability and more computing power.

IBM Eserver xSeries 345 is an expandable, high-availability,

high-performance server for e-business applications and mail and collaborative applications in a rack-mounted package. The x345 2-way server delivers computing power with high availability and expansion capability. The 2U x345 has the latest Intel® Xeon processors, DDR memory, and Dual Ultra320 SCSI controllers to deliver high performance.

IBM TotalStorage FAStT900 Storage Server delivers breakthrough disk performance and outstanding reliability for demanding applications in compute intensive environments. The FAStT900 offers investment protection with advanced functions and flexible features, including up to 32 TB of Fibre Channel (FC) disk storage capacity with the EXP700. It also offers advanced replication services to support business continuance and disaster recovery.

IBM TotalStorage FAStT EXP700 Storage Expansion Unit is a 14-bay, rack-mountable Fibre Channel disk drive enclosure, designed for use with many of the FAStT Storage Servers. It provides a full end-to-end 2 Gbps FC highly scalable, high-performance storage solution. Used with the FAStT900 Storage Server1 in particular, this option provides incremental support for more than 1 TB of disk storage per unit. It supports four new, slim-profile, dual-loop, hot-swappable disk drive modules.

4.2.2. Node, disk, and file system configurations

The test cluster consisted of 32 nodes as follows:

⊙ xSeries 345, dual CPU, 2.8 GHz, 4 GB RAM
⊙ Red Hat Linux 8.3, Kernel 2.4.20-19.8 pok
⊙ GPFS 1.3
⊙ PVM 3.4

The nodes were divided into two sets; there were 28 client nodes and four storage server2 nodes. All 32 nodes were part of the same GPFS node set.

The storage subsystem consisted of four server nodes (x345), two disk controllers (FAStT900), 12 disk drawers (EXP700, six per controller), 16 FC adapters (two per server, four per controller, all at 2 Gbps) in a Network Shared Disk (NSD) configuration. See the test configuration diagrams throughout the paper for illustrations.

The disks attached to each FAStT900 were configured as 16 LUNs (4+P RAID groups). The FAStT900 parameters were set as follows.

⊙ Read-ahead multiplier = 0
⊙ Write caching = off
⊙ Read caching = off
⊙ Segment size = 32 KB
⊙ Cache block size = 16 KB

The GPFS settings were left at their default settings for the most part. The only exceptions were the page pool size, which was set to 128 MB, and the block size, which was set to 256 KB.

4.2.3. Cluster network configurations for each test

Each test had a different test cluster network configuration, each offering alternative price and performance profiles. GPFS, in an NSD configuration, makes extensive use of the cluster network for passing file data/transactions between the client and server nodes.

1) Test #1 – GPFS/NSD over 100 MbE for client nodes

In this test, each client node was connected to the cluster network using 100 MbE (Megabit Ethernet), and each server node was connected using 1 GbE (Gigabit Ethernet or 1000MbE). This is not an officially supported configuration by GPFS, but it configured quite naturally, and all tests using it completed normally. See Figure 1 on page 7.

2) Test #2 – GPFS/NSD over 1 GbE on servers and clients

There were two related but distinct configurations evaluated in this test. In the first test (#2A), each of the client nodes and server nodes was connected to the cluster network with a single GbE adapter. This configuration is officially supported and commonly used by GPFS.

In the second test (#2B), each of the client nodes was connected to the cluster network with a single GbE adapter. However, each of the server nodes was connected to the cluster network using two

bonded GbE adapters forming a single channel (IP address). Although it worked, and all tests performed on it completed normally, this configuration is not officially supported for use by GPFS at this time. The reason for considering it is that it can compensate for the communication load imbalance presented by having a large number of client nodes sharing a small number of server nodes. See Figures 2 and 3 beginning on page 8.

3) Test #3 – GPFS/NSD over Myrinet on servers and clients

In this test (#3), each client and server node was connected to the cluster network using Myrinet adapters and the Myrinet switch. This configuration is officially supported and commonly used by GPFS. A Linux cluster with the Myrinet switch is architecturally similar to an SP system. See Figure 4 on page 12.

4) Test #4 – Miscellaneous tests

There were two miscellaneous tests conducted, more as a proof of concept than a rigorous benchmarking test, to examine using the Linux storage server nodes for multiple purposes. The Myrinet-based configuration was used for these tests, but extended as necessary.

In the first test (#4A), client tasks were executed both on the server nodes and the client nodes at the same time. This is very common on SP systems. The system configuration in this case was identical to Test #3. See Figure 4 on page 12.

In the second test (#4B), the NSD server nodes for GPFS were also used as NFS server nodes, although it was the Linux ext3 file system that was exported, not GPFS.3 The NFS clients used the 100 MbE network to communicate with the server nodes, and GPFS used the Myrinet network. A client node with an NFS mount did not have access to the GPFS file system and vice versa This was done for reasons of convenience, not necessity. See Figure 5 on page 14.

4.2.4. Benchmark code description

The benchmark codes used in this study are custom codes that we designed and programmed to evaluate common storage I/O paradigms used in seismic processing and most HPC customers. These codes have been used many times to evaluate I/O designs for various customers. They run on either AIX® or Linux systems. They use either PVM or MPI for parallelism when it is needed. The name of this benchmark is ibm.v3e (I/O Benchmark, version v3e). Contact the author for further details.

4.3. Analyzing the results

We explain the measurements and access patterns that we used, and then we examine the test results.

4.3.1. Metrics used to assess results

Bandwidth in units of megabytes per second (MB/s) is the fundamental measure of performance in these tests. To measure bandwidth, one must therefore measure data volume and time.

For all of the reported tests, each task in a job accessed at least 1 GB of data with the total data for the job being the number of tasks times the data per task. For example, a four-task job accesses 4 GB of data.

Time is assessed using wall clock time starting at a beginning and terminating at an end. Thus no distinction is made between user, system, and real time. Because the test jobs are parallel, time is collected on a per task basis, and no task starts its timer until all tasks have been spawned, their parameters initialized, and their file opened; because the timer starts immediately following a barrier, all tasks start at approximately the same time. The task''s timer is stopped after the last record has been accessed and the file has been closed. Because of the stochastic (involving chance or probability) nature for the various queuing systems associated with parallel tasks, under normal circumstances, tasks can and do terminate with some variance,4 although on the whole they remain uniformly active.

Three different measures of bandwidth are used to assess performance:

⊙ Natural aggregate: Total data divided by the time of the longest task
⊙ Harmonic mean: Total data divided by the sum of the task

times

⊙ Harmonic aggregate: The harmonic mean multiplied by the number of tasks

The harmonic aggregate has the effect of smoothing out the variance across the tasks of a job and gives a more typical rate as viewed from a system perspective. (Note: It is very consistent with performance monitors such as iostat measuring I/O rates at the LUN level). It cancels the effects of outliers. The natural aggregate is the I/O rate as viewed from a job perspective. From a system perspective, this rate is often lower than the actual system rate because it is heavily biased by outliers.

4.3.2. Access patterns

An access pattern is determined by the size of the records and the order in which records are read or written in a file. Within this benchmark, two record sizes and several patterns are used:

⊙ Large record: 1 MB or 1024 KB.
⊙ Small record: 16 KB.
⊙ Sequential: The file is partitioned into N subsets (N = number of tasks) and each task accesses its records contiguously.
⊙ Strided: Skip 100 records, access the next record, and repeat the pattern, wrapping around the file till all records are accessed once.
⊙ Semi-random: The next four records are randomly accessed within a working set of 32 records.
⊙ Random: Each record within the file is accessed once and only

once using a uniform random distribution.

⊙ Hint: Use the GPFS Multiple Access Hint mechanism to pre-fetch records.

Access patterns have a significant impact on performance. Generally speaking for GPFS, large record patterns, especially sequential, produce the best results. However, due to caching algorithms in GPFS, a small record sequential access pattern can often perform nearly as well. These patterns are frequently used in evaluating file system performance.

Unfortunately, many applications cannot adopt these optimal access patterns and thus other access patterns are tested. At the other extreme in terms of performance is small record irregular patterns (random or semi-random). These are patterns for which no optimizations exist to improve their performance and which also violate working set rules rendering caching algorithms ineffective. In many cases, however, the application can be written in such a manner as to predetermine when records will be accessed. If this can be done, this information can be passed to the file system and GPFS can pre-fetch the records asynchronously, significantly improving performance.

Strided access patterns lie in between the extremes of small record random and large record sequential. GPFS has optimizations that can recognize this pattern and pre-fetch records, for example, to improve performance without the explicit use of hints. This access pattern commonly occurs in seismic applications, for

example.

4.3.3. Results and analysis for each test

This section summarizes key results and their analysis from each of the various tests. Complete results for each experiment are listed in a spreadsheet. See “"Associated documents"" on page 18.

1) Test #1 – NSD over 100 MbE for client nodes

100 MbE is an older technology, but provides “"good enough"" performance in a cost effective manner for many HPC applications with low storage I/O performance requirements. For example, it is commonly used to provide file access through NFS in very large Linux clusters. Because of its popularity, GPFS was evaluated using this technology, even though it is not officially supported. See [Figure 1].

[Table 1] lists the results for a large record sequential access pattern test. For a single 100 MbE adapter, 7 MB/s of effective throughput is a reasonable target. Looking at the harmonic mean in the read test, you can easily see that this was achieved and that aggregate performance scales linearly. Because the servers'' access to the cluster network was over GbE, they were not a bottleneck. Write performance, however, was much lower for reasons that are not clear. It is felt that a configuration anomaly or some other artificial problem is the cause. Further investigation is needed.

Table 1 Large record sequential using 100 MbE for the client nodes

		Natural aggregate		Harmonic aggregate		Harmonic mean	
Number of client nodes	Number of tasks per client node	Write rate MB/s	Read rate MB/s	Write rate MB/s	Read rate MB/s	Write	Read
1	1	0.141	7.87				
2	1	0.209	15.44	0.209	15.49	0.105	7.74
4	1	0.296	30.59	0.307	30.74	0.077	7.68
8	1	0.357	56.27	0.393	60.22	0.049	7.53
16	1	0.429	111.89	0.498	116.41	0.031	7.28

Because of the intrinsically low access rate for 100 MbE, time only allowed one more test, a small record sequential access test. The data rate per task dropped down to a little over 4 MB/s for reads; writes were not tested. See the spreadsheet listed in “"Associated documents”" on page 18 for further details.

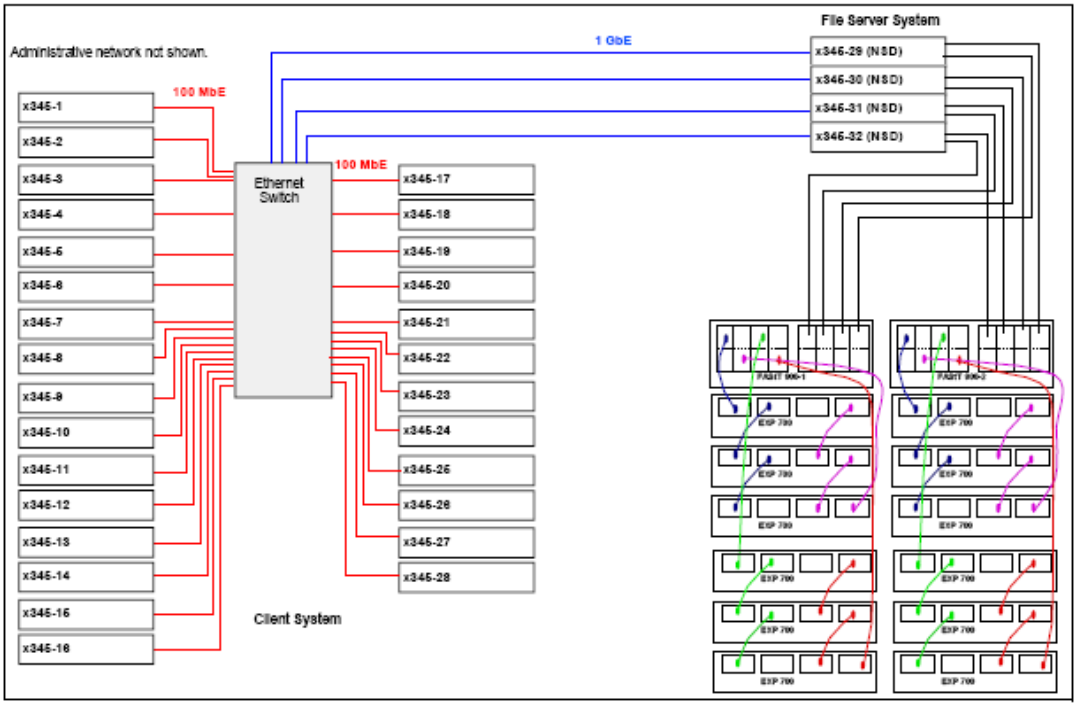


Figure 1 Test #1 configuration

2) Test#2 – GPFS/NSD over GbE on servers and clients

Two variations of this test were performed. The first, Test #2A, was the very standard configuration where all nodes, both server and client, provided a single GbE adapter for GPFS traffic. See Figure 2. In the second, Test #2B, the server nodes had dual bonded GbE adapters in order to compensate for the large number of clients per server; this configuration is not yet officially supported in GPFS.

First consider a large record sequential test for the single GbE adapter per server configuration. It is used to establish baseline peak performance. See Table 2. A reasonable upper limit estimate of GbE adapter performance is 75 MB/s. As the single and dual client tests show, this was significantly surpassed. However, as the task counts increased, performance leveled off somewhere around 350 MB/s, and the harmonic mean steadily dropped rather than remaining constant. Part of the problem is that the servers are gating the performance. There are only four servers, each with a single GbE adapter, but there are 28 clients in the configuration, which in aggregate are capable of yielding 7X the bandwidth that the servers can provide. Using the 75 MB/s target, the aggregate performance exceeds expectation for what this configuration can provide given that there are only four servers. However, using a 100 MB/s target, suggested as feasible by the single and dual task jobs, one would expect an aggregate closer to 400 MB/s. A possible reason that we are below this is attributable to network congestion being sorted out by the various software layers (such as inevitable parallel overhead). For example, there are 16 clients dumping packets to only four servers. In any case, scaling appears to be linear in the number of servers, being O(n). See “"Linear scaling in the number of storage servers”" on page 14.

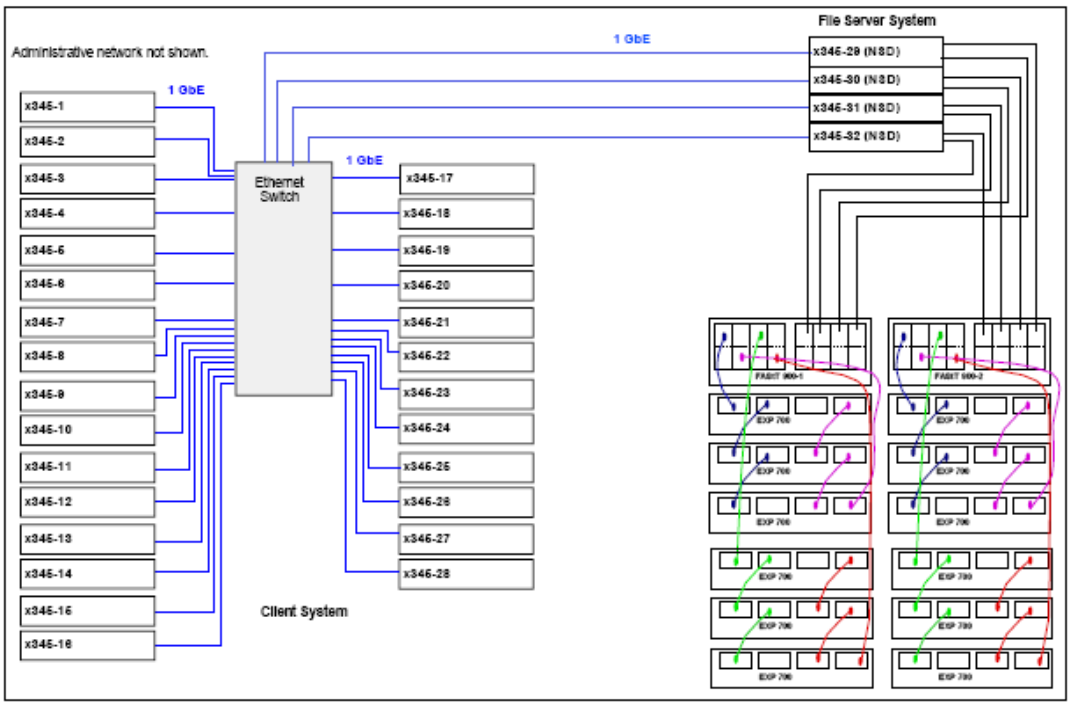


Figure 2 Test #2A configuration

Table 2 Large record sequential in the single GbE adapter/server configuration

		Natural aggregate		Harmonic aggregate		Harmonic mean	
Number of client nodes	Number of tasks per client node	Write rate MB/s	Read rate MB/s	Write rate MB/s	Read rate MB/s	Write	Read
1	1	92.26	111.27				
2	1	178.02	209.94	178.03	220.60	89.02	110.30
4	1	292.54	320.46	296.06	337.92	74.02	84.48
8	1	383.69	338.26	383.86	360.21	47.98	45.03
16	1	315.12	347.32	335.75	357.07	20.98	22.32

In order to diminish the effect of the single GbE adapter/server gating factor, we tried another set of tests using a set of dual bonded GbE adapters per server; see Figure 3. Table 3 summarizes the large record test. As a target, one would expect to double the bandwidth per server node, yet this is not the case. Performance is roughly the same for the writes and slightly improved for the reads. A plausible explanation for this is that these x345 systems have only two CPUs, so there is insufficient horsepower to keep up with the load. If this is true, perhaps a four-CPU x360 could improve the performance.

A good validation of this assertion could be based on CPU utilization. Unfortunately, the CPU utilization record was unavailable for this test. However, the CPU utilization for the single GbE adapter tests was as high as 86% for peak bandwidth jobs. It is, therefore, quite plausible that having dual GbE adapters per server, albeit bonded into a single channel, required more than the additional 14% CPU bandwidth to fully use their bandwidth. Further testing is needed to fully validate this point.

Under Test #2A (single GbE adapter per server), a number of other access patterns were also tested and performed according to expectation. For example, the small record random pattern yielded the worst performance; the 16 KB record size only used 6% of the GPFS block and the random pattern thwarted cache optimization algorithms (see Table 4). In spite of low absolute performance, reads have the useful property of scaling linearly in the number of

Table 3 Large record sequential in dual bonded GbE adapter/server configuration

		Natural aggregate		Harmonic aggregate		Harmonic mean	
Number of client nodes	Number of tasks per client node	Write rate MB/s	Read rate MB/s	Write rate MB/s	Read rate MB/s	Write	Read
1	1	109.67	110.29				
2	1	212.36	214.51	212.36	215.50	89.02	110.30
4	1	344.26	398.28	365.75	398.94	74.02	84.48
8	1	326.76	414.49	342.01	426.82	47.98	45.03
16	1	308.77	388.94	312.33	424.90	20.98	22.32
28	1	192.20	206.89	235.88	207.65	8.42	7.42

clients instead of leveling off at four tasks, as in the sequential pattern case being gated by the servers. This yields a relatively acceptable aggregate performance across a larger number of client nodes. This can be seen very clearly by looking at the harmonic mean. However, due to the RAID penalty, small record writes performed significantly worse than reads did, and they did not scale linearly. By using the Multiple Access Hint feature in GPFS, the absolute performance can be increased significantly for random reads (for example, 5X for eight client tasks; see the spreadsheet listed in “"Associated documents"" on page 18 for details).

Table 4 Small record random in a single GbE adapter/server configuration

		Natural aggregate		Harmonic aggregate		Harmonic mean	
Number of client nodes	Number of tasks per client node	Write rate MB/s	Read rate MB/s	Write rate MB/s	Read rate MB/s	Write	Read
1	1	2.86	2.09				
2	1	1.49	4.16	1.58	4.16	0.79	2.08
4	1	2.09	8.37	2.35	8.43	0.59	2.11
8	1	1.80	16.78	2.03	16.95	0.25	2.12
16	1	1.98	32.42	2.12	32.87	0.13	2.05

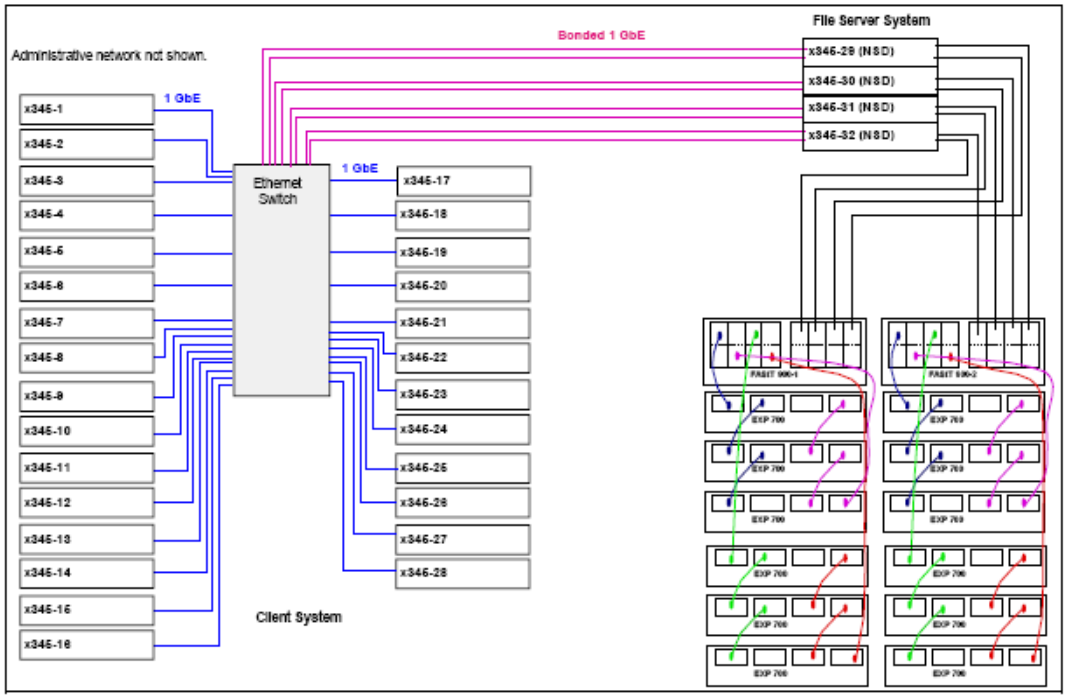


Figure 3 Test #2B configuration

The small record strided access pattern yielded performance rates in between the two extremes of the small record random and

large record sequential patterns (see Table 5). Its performance is degraded because it is only using 6% of the block for each record, but it gets a performance uplift from optimization algorithms based on the fact that GPFS recognizes the pattern; for example, it can pre-fetch records in a read job. Due to the RAID write penalty for small records, the write performance is not as good as the read performance.

Table 5 Small record strided in a single GbE adapter/server configuration

		Natural aggregate		Harmonic aggregate		Harmonic mean	
Number of client nodes	Number of tasks per client node	Write rate MB/s	Read rate MB/s	Write rate MB/s	Read rate MB/s	Write	Read
1	1	8.46	24.73				
2	1	8.83	30.72	8.84	44.89	4.42	22.44
4	1	13.22	51.02	13.25	79.13	3.31	19.78
8	1	18.14	86.32	18.44	117.54	2.31	14.69
16	1	22.08	76.30	22.67	91.95	1.42	5.75

3) Test#3 - GPFS/NSD over Myrinet on servers and clients

The Myrinet switch is a very standard Linux xSeries cluster network configuration for GPFS. It is architecturally similar to SP systems, although it uses only the IP stack for communication. This test configuration used the first generation Myrinet technology. Therefore, single adapter target bandwidth is 125 to 150 MB/s, although testing suggests that this might be slightly pessimistic. Compared to GbE, Myrinet delivers higher

performance. However, the general shape of the performance profiles is very similar; large record sequential is faster than small record strided is faster than small record random. Although the absolute price for Myrinet is higher, fewer servers may be needed from a storage standpoint, especially for the faster second generation Myrinet technology, thus improving its price/performance characteristics.

Consider the large record sequential access pattern (see Table 6 and Figure 4). As in the GbE case, the four-server configuration was used, and each server and each client had a single Myrinet adapter. And as with the GbE adapter case, performance was gated by the limited number of servers, each with one Myrinet adapter. However, what is different is that the write performance is significantly less than the read performance, although in absolute terms, it is quite acceptable. For example, the relative error between the read and write performance in the Myrinet case is roughly 30% for 8 or 16 client tasks, although it is only 6% for the GbE case.

Table 6 Large record sequential in Myrinet adapter configuration

		Natural aggregate		Harmonic aggregate		Harmonic mean	
Number of client nodes	Number of tasks per client node	Write rate MB/s	Read rate MB/s	Write rate MB/s	Read rate MB/s	Write	Read
1	1	159.75	184.63				
2	1	273.85	324.94	288.82	341.13	144.41	170.56
4	1	320.76	394.52	420.34	483.16	105.09	120.79
8	1	382.95	536.70	385.03	559.73	48.13	69.97
16	1	388.43	516.92	396.57	584.77	24.79	36.55
28	1	380.49	560.52	385.31	563.83	13.76	20.14

The performance profiles of the other Myrinet test cases follow expectation in a similar manner to the GbE test cases. Small record strided access is less than large record sequential, but it is better than the GbE case. Random and semi-random patterns using hints is slower than small record strided, but again, it is better than the GbE case. The one slight surprise is the small record random with no hints case. Although it is the slowest access pattern for Myrinet, its rate relative to GbE is nearly the same. Because of these similarities, these test cases are not considered in detail in this report, but the spreadsheet (listed in “"Associated documents"" on page 18) documents them.

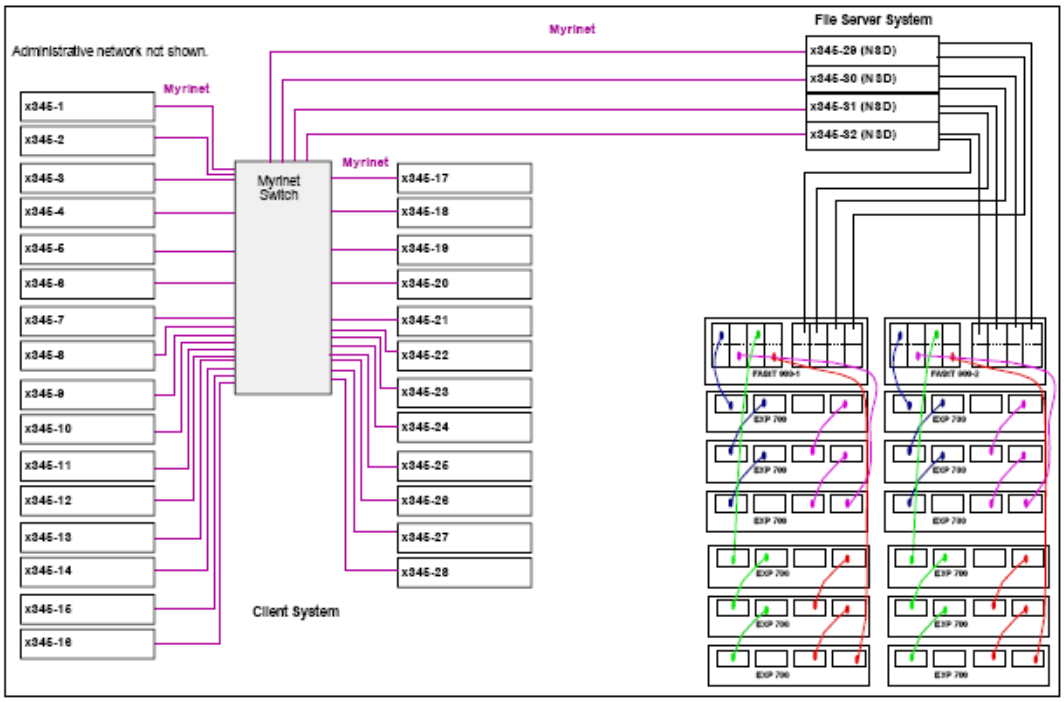


Figure 4 Tests #3 and #4A configuration

4) Test #4 – Miscellaneous tests

Several smaller, restricted scope tests5 were conducted as time allowed. The purpose of these tests was to evaluate multiple simultaneous uses of the servers. They were completed using the Myrinet test environment, extending it as necessary. The significance of this test is that NSD servers do a lot of work for GPFS clients, especially in the case where there are a small number of servers relative to a large number of clients. However, the NSD servers do not always use all of their available bandwidth. Therefore, can this bandwidth be devoted to other tasks without adversely affecting NSD service?

Test #4A (see Table 7 and Figure 4) shows the result of one experiment running “"client-on-server tasks"" at the same time that regular client-on-client tasks are running (these are all large record sequential read tests). This is common strategy in an SP environment, but there are typically a large number of servers relative to the clients, unlike this case. The first three lines of Table 7 are various baseline measurements. First compare the client-on-client and client-on-server baseline cases. The client-on-server case is significantly slower, probably because these nodes are doing both client and server work. Therefore, local client activities appear to interfere with local NSD service activity. But then compare the 8-way client-only baseline with the mixed client and client-on-server case, eight tasks in total). The results are not significantly different. This test was repeated several times, and the results were generally similar. (See the spreadsheet listed in “"Associated documents"" on page 18.)

Table 7 Client on server tests

Client task on NSD server node		Client task on client node				
Number of tasks	Read rate MB/s	Number of tasks	Read rate MB/s	Total number of tasks	Aggregate rate	
4	336	0		4	336	Client on server baseline
0		483		4	483	Client on client baseline
8	560			8	560	8-way client only baseline
4	212	4	364	8	576	Mixed client and client on server

Test #4B (see Table 8 and Figure 5) is similar in spirit to the previous test, except that the NSD servers are given NFS service duties instead of GPFS client duties (these are all large record sequential read tests). Also note that the NFS service is based on the local ext3 file system. In this test, because 100 MbE is used, the NFS client rates are quite low. Comparing the GPFS client rate in the mixed NSD/NFS test (third row, fourth column) to the GPFS client only baseline rate (second row), the difference is not significant. However, the NFS client rate in the NSD/NFS mixed test (third row, second column) showed a 23% drop in performance compared with its baseline (first row). Because the absolute value of this rate is small relative to the GPFS client rates, it is hard to draw definitive conclusions; however, the results show enough promise to warrant further testing.

Table 8 Simultaneous NFS and NSD server tests

NFS client tasks		GPFS client tasks				
Number of tasks	Read rate MB/s	Number of tasks	Read rate MB/s	Total number of tasks	Aggregate rate	
4	22.84				22.84	NFS clients only - baseline
0			413.88		413.88	GPFS clients only - baseline
4	17.92		428.33		446.25	Mixed NSD/NFS server test

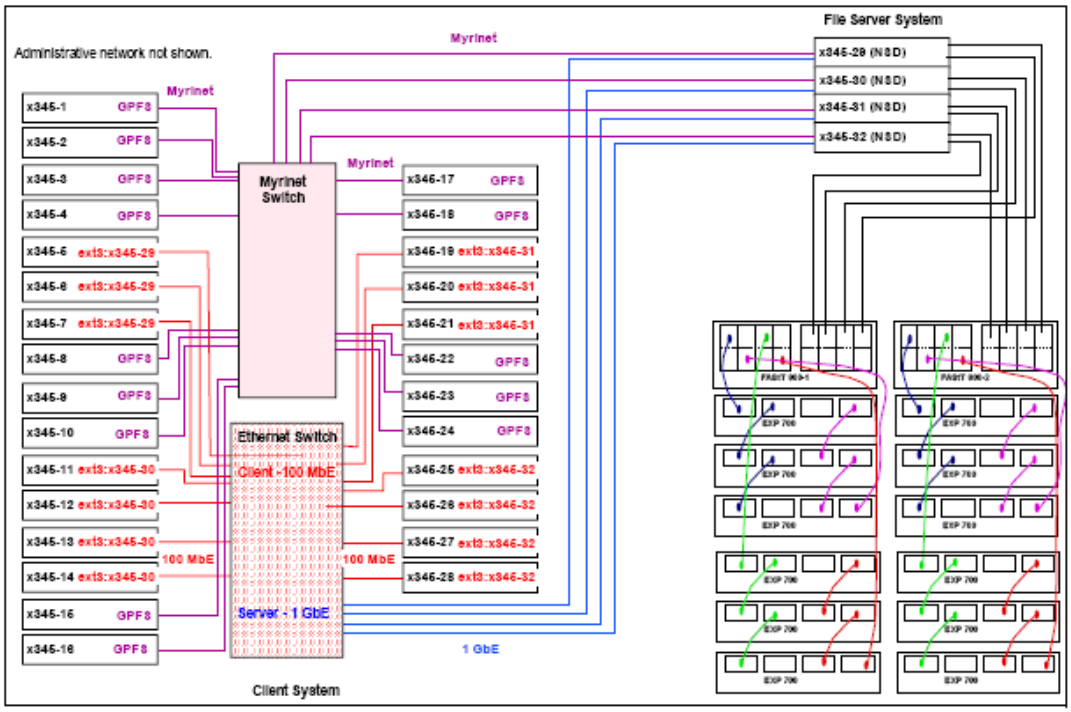


Figure 5 Test #4B configuration

4.3.4. Linear scaling in the number of storage servers

One of the well-known properties of GPFS in AIX systems is its ability to scale linearly in the number of VSD server nodes. Mathematically, this property is represented as O(n), where n is the number of servers. In more specific terms, if X is the I/O rate for one server, then the rate for n servers is n * X - kn where kn is an overhead term. kn can be unique for each n, but it is not necessarily linear in n.

Given the broad architectural similarities between

GPFS/VSD/AIX/pSeries and GPFS/NSD/Linux/xSeries, it is reasonable to expect that this behavior would likewise be evident in the latter system. However, this assertion needs to be validated. Given the limited time that this test system was available, there was insufficient time to rigorously test this.

However, some proof of concept tests give support to this assertion, as do several other indirect measures. During initial testing to validate optimum FAStT900 parameters, we conducted a two-server test. The results of this test can then be compared with the baseline four-server tests used for the bulk of this study. These results are shown in Table 9. These were write tests on the Myrinet configuration (see ""Test#3 - GPFS/NSD over Myrinet on servers and clients"" on page 11) writing 1 GB per task to a common file. The numbers are consistent with the O(n) relationship.

Table 9 Ad hoc test evaluating scaling in the number of servers

			Natural aggregate	Harmonic aggregate
Number of server nodes	Number of client nodes	Number of tasks per client node	Write rate MB/s	Write rate MB/s
2	4	1	215	231
4	4	1	356	384

Another measure providing evidence that is consistent with this linear scaling observation can be seen by considering Tables 2 and

6. Consider the read performance in particular. Referencing a single client task job, the data rate is 111 MB/s for the GbE adapter case, and it is 185 MB/s for a single Myrinet adapter case. From this, we can infer for the GbE case that the performance is gated by the adapter at 111 MB/s, which would also be an upper bound data rate on a single server test where the server had only a single GbE adapter.

Next consider the multitask jobs in Table 2. This test uses four servers, each with a single GbE adapter. In this test, the peak read rates are approximately 360 MB/s, obeying the O(n) relationship to the 111 MB/s for a single server. Again, this assertion needs more rigorous testing. But based on these preliminary tests and inferences, as well as the architectural similarity GPFS on Linux/xSeries to GPFS on AIX/pSeries systems where this relationship is well understood, this assertion seems plausible enough to warrant further investigation.

4.3.5. Failover validated by unexpected server failures

During these tests, one of the server nodes experienced several failures, forcing it to shut down and be rebooted. This problem appears to have been caused by intermittent adapter failures under heavy load, although the exact nature of these problems was never determined. But more to the point, the failover features of this system performed according to expectation in each case. The jobs continued to run, albeit at a reasonably reduced data rate.

4.4. Alternative GbE configurations that are balanced

Ideally, the aggregate bandwidth of the various components of a storage subsystem (disk servers, disk controllers, disk drawers, and adapters) should be balanced. For example, if the aggregate bandwidth of the GbE adapters in the disk servers can sustain a data rate of 360 MB/s, yet the controllers can yield 700 MB/s, the controllers are being under-used and their cost wasted. The one place where this rule regarding balance might not apply is if the total disk volume needed requires a large number of physical disks (pdisks) to achieve it, yet the data rates needed by the application do not require the peak bandwidth that this number of pdisks can deliver. In such cases, the number of pdisks attached to the controllers will exceed the bandwidth capability of the controllers.

According to this design criterion, the configurations tested in this benchmark are unbalanced (see Figures 1 through 5). In earlier tests using four GPFS/VSD/AIX/p655 servers with the exact same FAStT900 configuration described in this paper, sustained peak aggregate data rates were measured as high as 740 MB/s. However, due to the gating effect of having either a single GbE or Myrinet adapter per server, the peak sustained bandwidths measured in this benchmark was much less. To circumvent this gating factor and thereby balance the design, a configuration using dual bonded GbE adapters was also tested. However, the performance of this configuration appears to have been gated by having insufficient CPU resource in the x345 servers to fully harvest the available bandwidth, thus leaving the design

unbalanced.

This observation suggests two alternative configurations that can theoretically eliminate this gating effect and balance the design.

1) A 4:1 server/controller ratio

The configurations tested in this benchmark used a 2:1 server/controller ratio delivering at best 50% of the potential bandwidth from controllers, being gated by the single GbE adapter in each x345 server. Because it appears that an x345 server cannot fully drive more than 1 GbE adapter (see “"Test#2 – GPFS/NSD over GbE on servers and clients"" on page 7), then by increasing the server/controller ratio to 4:1, the bandwidth can be potentially balanced while maintaining a single GbE adapter per server. However, this requires attaching eight FC adapters to each controller in order to support the customary failover design (see Figure 6).

Based on extractions from the results of this benchmark, it is reasonable to expect that four x345 servers, each with a single GbE adapter, should be able to harvest most of the bandwidth that a FAStT900 can deliver. But what is not tested is whether the FAStT900 has the processing power to effectively manage eight FC adapters, although only four FC adapters are needed for bandwidth purposes. Using four-CPU x360 servers As previously observed, the x345 servers do not appear to be able to handle the load of dual bonded GbE adapters. However, because an x360 can

be configured with up to four CPUs (twice as many as an x345), as well as having a more robust I/O capability, then perhaps they can be used in place of the x345 servers and achieve bandwidth balance with a 2:1 server/controller ratio (see Figure 7).

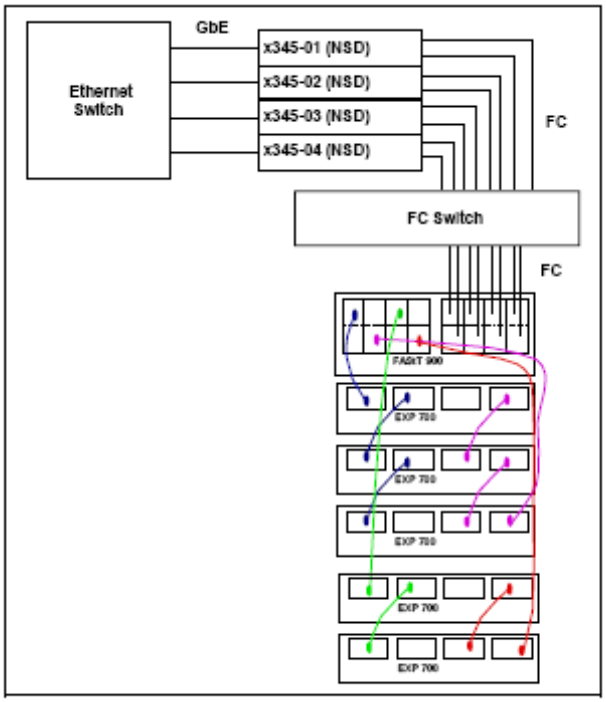


Figure 6 Configuration using a 4:1 server/controller ratio

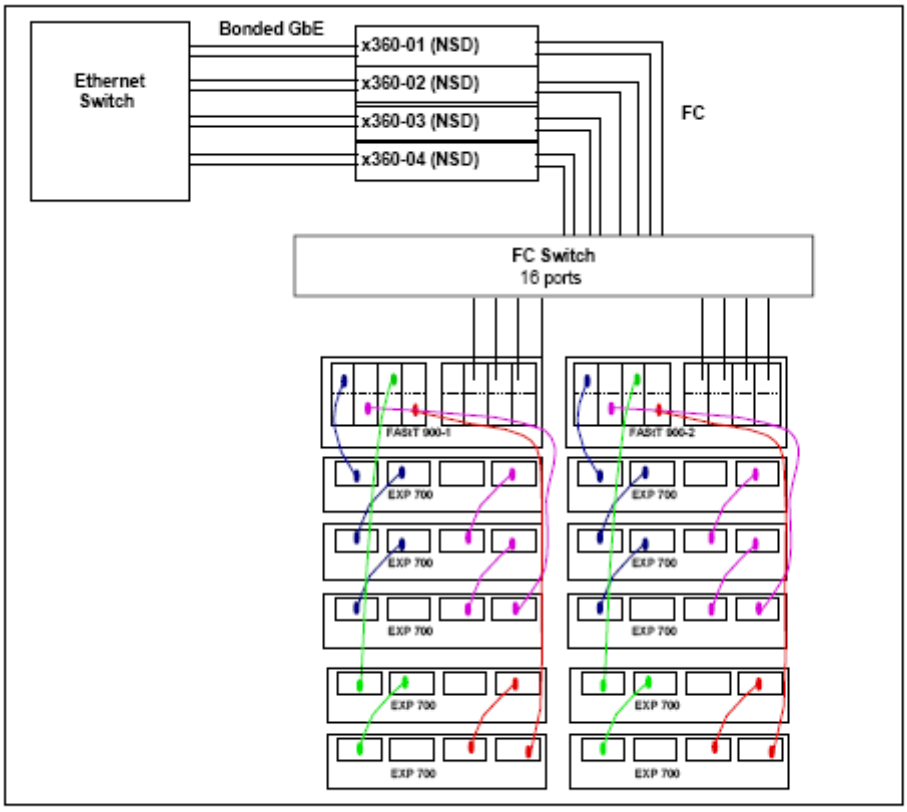


Figure 7 Balanced configuration using bonded GbE with a 4-CPU x360 in a 4:1 server/controller ratio

4.5. Summary

The GPFS file system in a Linux/x345 cluster configured in an NSD mode is shown to provide effective and efficient file service when using single GbE adapters per node or single Myrinet switch adapters per node. Alternative designs based on presently unsupported configurations (100 MbE and bonded Ethernet) show promise but require further investigation. The configurations in this

study were not, however, bandwidth balanced. Therefore, alternative configurations are proposed, subject to validation.

Associated documents

More details about this test are available in the following documents:

- Benchmark Results: GPFS on Linux/x345 Cluster See spreadsheet file: BM_results.xls
- xSeries/Linux/GPFS: Benchmark Plan
- See file: BM_Plan.pdf

These documents can be downloaded from the IBM Redbooks™™ Web server at:

ftp://www.redbooks.ibm.com/redbooks/REDP3909

At the Web site, click Additional Material on the right side of the window. Right-click the .zip file (REDP3909.ZIP). Select the Save file as option to download the file. Unzip into a directory of your choice.

5. 프리 소프트웨어 프로젝트에서의 품질 관리 관행과 문제점

요약

프리 소프트웨어와 공개 소스 프로젝트의 결과물은 품질이 좋은 것으로 인식되고 있다. 일부 프리 소프트웨어에서 관찰되는 높은 수준의 품질은 다른 개발자에 의한 소스 검토를 권장하는 공개적인 개발 모델과 관계가 있다고 여겨진다. 일부 프리 소프트웨어의 품질이 비공개 상용 소프트웨어의 품질에 필적(더 좋지는 않아도)할만하다 할지라도 모든 프리 소프트웨어 프로젝트가 성공적이며 품질이 높다고 볼 수는 없다. 완성도가 높고 성공적인 프로젝트들조차 품질 문제에 직면한다. 이들 문제점 중 일부는 대부분이 자원 개발자에 의해 진행되는 분산 개발 모델을 따르는 프리 소프트웨어와 공개 소스 프로젝트의 고유 특성과 연관되어 있다. 이 연구에서는 프리 소프트웨어, 공개 소스 개발자와의 인터뷰를 통하여 실제적인 품질 문제와 여러 가지 품질 관리를 위한 개발 관행들이 확인되었다. 이 논문 에 제시된 인터뷰 결과는 현재의 프리 소프트웨어 프로젝트의 품질 현황을 점검하고 품질 개선 절차 개발의 출발점으로 삼을 수 있을 것이다.

5.1. 서론

최근 프리 소프트웨어와 공개 소스 개발 모델은 다른 개발 모델들을 대치할 수 있을 정도의 경쟁력을 갖추었다. Linux, Apache, Samba와 같은 많은 대중적 제품들이 공개 소스 모델 을 따라 만들어졌으며, 프리 소프트웨어 응용 프로그램의 수도 꾸준히 증가하고 있다. 2001년에 행한 프리 소프트웨어 연구에서는 5천 5백만 라인 이상의 소스를 찾아냈는데 이의 가

치는 COCOMO 모델[4]을 따라 평가할 때, 19억 달러에 이르는 것으로 추정되며, 그 규모 는 그 이후에도2배 이상 증가했다[5].

프리 소프트웨어 모델은 주요 소프트웨어들의 창조에 기여했으며, 많은 프리 소프트웨어 응용 프로그램은 공개 소스가 아닌 독점 소유의 소프트웨어를 능가하거나 버금가는 품질을 갖추고 있다[6][14]. Raymond는 그의 저서 “성당과 시장 (The Cathedral and the Bazaar)”에서 이러한 높은 수준의 품질이, 부분적으로는 프리 소프트웨어 프로젝트에서 흔히 이루어지는 많은 다른 개발자에 의한 소스 검토와 사용자의 참여 덕분이라고 말하고 있다[13]. 이러한 가설은 다른 개발자에 의한 소스 검토가 소프트웨어의 품질에 중대한 영향을 끼친다는 전통적인 소프트웨어 엔지니어링 기술 연구 결과와 일치하는 것이다.

Raymond 의 가설은 거의 맞는 것처럼 들리긴 하지만 여전히 실험에 근거한 자료 분석을 통하여 엄격하게 증명되고 검사되어야 한다. 보통 프리 소프트웨어의 품질에 관한 언급의 대부분은 실험적 증거보다는 단편적인 일화에 근거한 것들이 많다. 많은 프리 소프트웨어 응용 프로그램이 이룬 최근의 성공들을 고려하면, 어느 정도는 이런 소프트웨어들이 사용자들의 요구와 품질 기준을 만족시킨다고 여길 수도 있다. 럼에도 불구하고, 리 소프트웨어의 품질에 대한 연구와 독점 소프트웨어와의 품질 비교를 위한 실험적 연구가 필요하다고 본다. 특히 프리 소프트웨어 모델과 관련된 문제점들을 도출하는 작업은 유익한 것으로 사료되며, 이러한 연구의 결과물은 앞으로 개발 절차 개선을 위한 전략을 개발하는데 사용될 수 있을 것이다.

5.2. 연구 배경

보통 프리 소프트웨어 프로젝트는 품질 확보와 중요한 관계가 있는 두 가지 특징을 가지고 있다. 첫째, 많은 경우 프리 소프트웨어 프로젝트는 전 세계에 분산되어 진행된다. 둘째, 프리 소프트웨어 프로젝트의 구성원들은 보통 보수를 받지 않는 자원 개발자들이다. 프리 소프트웨어를 설치한다는 것으로 나타나는 프리 소프트웨어에 대한 관심은 일반 사용자들을 위한 성공적이고 완성도 높은 제품들이 분산 개발 방식으로 자원 개발자들 의해 만들어질 수 있다는 것을 분명하게 보여 주고 있다. 프리 소프트웨어 개발 모델은 독점 소프트웨어 개발 모델보다 나은 어떤 이점들을 제공하는데, 그 이점은 프리 소프트웨어 모델이 다른 개발자들에 의한 진지한 소스 검토를 받을 가능성이 높다는 것과 전 세계로부터 뛰어난 프로그래머들을 흡수할 수 있다는 것이다. 그러나 프리 소프트웨어도 개발 모델의 특수성으로 인한 여러 문제들을 가지고 있다. 예를 들어, 프리 소프트웨어 프로젝트가 갖는 자원 봉사의 속성 때문에 개별 프로젝트 구성원들을 지속적 참여를 전적으로 기대하는 것이 불가능하다[11]. 이런 문제는 프로젝트의 분산 개발 속성에 따라, 자신에게 할당된 작업을 소홀히 하는 자원 개발자를 확인하는 것과, 더 많은 개발자가 필요한 개발 영역에 관한 판단을 더욱 어렵게 만든다[10].

공개 소스에 관한 연구 대부분이 Apache[12]와 GNOME[9]와 같이 대중적이고 성공적인 프로젝트에 초점을 맞추고 있지만, 모든 프리 소프트웨어 프로젝트가 완성도가 높거나, 성공적이거나, 아니면 고품질에 이르는 것은 아니라는 우려도 증가하고 있다. 95,000개가 넘는 프리 소프트웨어와 공개 소스 프로젝트를 호스트 하는 가장 대중적인 사이트인 SourceForge는 유지 보수가 잘 되고 있는 프리 소프트웨어 응용 프로그램을 찾기 위한 훌륭한 소스이기도 하지만, 같은 사이트에는 많은 중단된 프로젝트들과 저품질의 소프트웨어들도 함께 있다[7]. 중단된 프로젝트 중 일부는 더 높은

가능성을 가진 흥미로운 프로젝트가 분명히 더 많은 자원 봉사자를 끌어들인다는 선택 과정의 측면에서 설명될 수 있지만, 한편으로는 프로젝트의 실패가 프로젝트 관리 기술의 부족과 연관이 있다는 것을 제시하는 연구도 있다[15]. 또, 규모가 크고 성공적인 프로젝트라 해도 자원 개발자의 지속적인 개발 지원의 어려움[10][11], 버그에 대한 개발자들의 지속적인 대처 부족[17] 등, 품질과 관련된 중요한 문제들에 직면하고 있다.

프리 소프트웨어 개발 모델이 기업 업무 및 높은 안정성을 요구하는 업무에 적합한 수준의 완성도와 고품질의 소프트웨어를 창조하기 위한 경쟁력 있는 모델로 남기 위해서는, 프리 소프트웨어의 품질 확보와 프로젝트 관리에 있어, 앞서 언급한 문제들과 다른 품질 관련 문제들을 모두 고려한 해결 방법을 찾아 내야한다. 프리 소프트웨어 프로젝트의 품질을 확보하고 더 개선하기 위한 개발 관행과 개발 절차를 확보하기 위한 첫 단계는, 현재 이용되는 품질 유지를 위한 여러 개발 관행과 품질 문제들을 명확하게 확인하는 것이다. 그러나 지금까지는, 프리 소프트웨어 프로젝트에서의 품질 유지와 관련된 초기 조사만이 일부 실시되었다[19][20]. 이 논문의 나머지 부분은 다양한 분야의 프로젝트들에서 활동하고 있는 프리 소프트웨어 개발자들과의 인터뷰 결과를 담고 있다. 이 인터뷰 결과는 현 시점에서 품질 관리를 위한 개발 관행들과 문제점의 상황을 더욱 정확하게 판단하는데 도움이 될 것으로 기대된다.

5.3. 연구 방법

이 연구를 위해 프리 소프트웨어와 공개 소스 개발자7명과 인터뷰를 실시하였다. 인터뷰 대상 개발자들의 프로젝트들이 매우 다양했기 때문에, 인터뷰 대상의 수가 비교적 적었음에도 불구하고 다양한 종류의 프리 소프트웨어 프로젝트들을 대변할 수 있는 결과를 얻었다고 믿는다. 한 프로젝

트는 대학에서 학부 학생들에게 소프트웨어 개발 프로젝트에 대한 경험을 주기 위해서 시작된 뒤, 나중에 이 대학 학생들이 외부 기여자로 참여하는 커뮤니티 프로젝트로 바뀐 것이며, 다른 하나는 대기업에 의해 이루지는 연구 활동의 일부이다. 또 다른 하나는 대학에서 이루어진 연구 프로젝트의 결과이며, 나머지 다른 나머지 프로젝트들은 커뮤니티 프로젝트로 시작되었다. 프로젝트의 규모는 Apache, GNOME과 Debian 같은 대규모 프로젝트부터 VideoLAN과 Dasher 같은 규모가 좀 더 작은 프로젝트까지 다양하다.

이 인터뷰는 인터뷰 대상자가 속해 있는 프로젝트의 전반적인 품질 문제에 관한 탐구를 위하여 정형화된 형태를 갖추지는 않았으며, 래와 같은 항목들이 직접적으로 질문되었고, 다른 주제들에 대한 자유로운 인터뷰 방법을 함께 사용하였다:

⊙ 당신이 참여한 프로젝트에 품질에 관한 이슈들이 있습니까? 만약 그렇다면, 그 이슈들은 무엇이며, 것을 어떻게 방식으로 처리합니까?
⊙ 프리 소프트웨어와 공개 소스 프로젝트의 품질을 보장하기 위해 어떤 기술이 적용될 수 있습니까?
⊙ 참여한 프로젝트에는 품질을 보장하기 위해 어떤 장치가 있습니까?
⊙ 당신은 공개 소스와 상용 소프트웨어의 품질을 어떻게 비교할 수 있다고 보십니까? 언제 어떤 소프트웨어가 다른 소프트웨어보다 품질이 높다고 이야기할 수 있습니까?

5.4. 결과

인터뷰 결과는 인터뷰에서 확인된 현재의 개발과 품질 유지를 위한 관행들에 대한 검토와 공개 소스 프로젝트가 직면한 품질 문제들에 관한 요

약 등 두 영역으로 나누어 설명한다. 이 두 영역을 다루기 전에 먼저 프리 소프트웨어 프로젝트 구성원들이 자신들의 , 프로젝트 품질을 어떻게 생각하는지에 대해 개괄적으로 살펴본 후에 상용 소프트웨어 품질 대비 공개 소스 소프트웨어 품질에 대한 질문 결과에 대해 이야기할 것이다. 인터뷰 대상자들이 이 질문에 대해서 전혀 편견이 없는 답변을 했다고 기대되지는 않기 때문에, 프리 소프트웨어와 공개 소스 개발자들이 가진 품질에 대한 주관적인 생각을 알 수 있을 것이다. 인터뷰 대상자 중 몇 명은 프리 소프트웨어든 비공개 상용 소프트웨어든 어느 한 쪽의 품질 이 더 높다고 말했다. 그러나 많은 인터뷰 대상자들은 프리 소프트웨어가 더 훌륭한 품질을 가질 수 있는 가능성을 더 많이 가지고 있고, 보안 버그와 같은 중요한 이슈에 더 빠르게 반응할 수 있다고 느꼈다. 이렇게 느끼는 데는 여러 이유가 있다.

첫째, 공개 소스라는 속성이 피드백을 더욱 촉진하여 소프트웨어를 개선하는데 도움이 될 수 있다는 것이다. 피드백은 버그 리포트의 형태나 기능에 대한 요구 형태로 나타날 수 있다. 둘째, 많은 인터뷰 대상자들은 자원 개발자들이 그들이 원하는 일을 한다는 점 때문에, 프리 소프트웨어 프로젝트에서 더 높은 동기 부여가 일어난다고 느꼈다. 다른 개발자들과의 공개적인 협력 및 다른 사람들의 의견 제시를 받는 것 등이 소프트웨어의 한 부분을 담당한다는 자신의 일에 더 높은 동기를 부여한다. 일부 인터뷰 응답자는 높은 동기 부여가 소프트웨어의 품질에 긍정적으로 작용한다고 생각했다. 셋째, 한 인터뷰 응답자는 프리 소프트웨어의 분산 개발 속성 때문의 더 많은 인적 자원을 끌어들일 수 있다고 말했다. 그는 “프리 소프트웨어는 보통 전통적인 소프트웨어 회사가 어떤 문제에 대해 보통 구할 수 있는 해결책보다 더 폭넓은 전문적 기술과 지식의 덕을 볼 수 있다”고 주장했다. 상업적 개발과 비교하여 커뮤니티 프로젝트의 부족한 점은 자원과 인프라의 부족이었다. 마지막으로, 한 인터뷰 응답자는 이 두 모델의 근본 원리가 서로 반대되기 때문에 공개 소스 개발 모델과 폐쇄적인 개발 모델

을 비교하는 것은 매우 어려운 일이라고 답했다. 소스를 공개하지 않는 회사들은 대개 자신들의 소프트웨어 결함과 소스 코드를 숨기기 때문에,공개 소스와의 직접적인 비교는 어렵다. 이 차이는 분명히 학문적 연구가 채워 줄 수 있는 영역이다.

5.4.1. 개발과 품질 유지를 위한 관행

인터뷰 결과 얻어진 놀라운 관찰 가운데 하나는 여러 개발 관행과 절차가 여러 프로젝트들에 대하여 어떻게 다르게 채택되고 있는가이다. 이 절에서는 인터뷰를 통해 확인된 여러 개발 관행들에 대해 요약해서 설명한다. 거의 모든 프로젝트가 이런 개발 절차의 일부분만을 채택하는 반면에, 몇몇 소수의 프로젝트는 모든 부분을 따른다. 왜 이런 현상이 일어나는지와 프로젝트들이 이러한 개발 관행들을 더 많이 도입함으로써 결국 이득을 볼 수 있을지는 분명하지 않다. 다만 흥미로운 사실 하나는 모든 인터뷰 응답자가 개발 절차와 특히 훌륭한 인프라의 중요성을 강조하였다는 점이다. 확인된 개발 관행들은 대체로 다음 세 영역으로 나누어 설명할 수 있다.

가. 인프라

프리 소프트웨어 프로젝트는 분산, 협력 개발을 가능케 하는 인프라에 많이 의존한다. 인프라의 중요 부분은 다음과 같다:

⊙ 버그 추적 시스템(예, BugZilla): 이는 사용자의 피드백을 수집하는 데 사용된다. 이 시스템은 실제 버그 리포트뿐만 아니라 기능에 대한 요구를 저장할 때도 자주 사용된다.

⊙ 버전 관리 시스템 (예, CVS, SVN, Arch) : 이 시스템은 여러 사람이 동시에 같은 코드기반에서 일하는 것과 누가 어떤 것을 변경했는지 계속 추적하는 것을 가능하게 한다.
⊙ 자동 빌드 (예, tinderbox): 버전 관리 시스템 내의 최신 코드가 계속 성공적으로 빌드 되는 것을 보장한다. 시험 빌드는 다른 여러 하드웨어나 소프트웨어 환경에서도 진행될 수 있다.
⊙ 메일링 리스트: 개발자, 사용자들 사이의 의사소통에 사용된다.

많은 프로젝트가 이런 인프라를 사용하는데 반해, 그 사용 방법은 프로젝트마다 상당히 다르다. 이 인프라 사용 방법, 즉 관행의 차이는 프로젝트의 품질과 밀접한 관계를 가질 수 있다. 예를 들어, 일부 프로젝트는 자동 빌드 시스템을 이용하여 결함이 제거될 때까지 향후 개발을 중지하는 매우 엄격한 정책을 채택하고 있다. 또 버전 관리 시스템의 사용에 따른 관행 또한 매우 다양하다. 일부 프로젝트에서는 첫 기여 이후에 그 기여자에게 바로 소스 등록 권한을 주기도 한다. 또 다른 프로젝트에서 개발자는 다른 개발자의 신뢰를 얻기 위해 몇 달 동안 일을 해야 하는 경우도 있고, 주 개발자만이 소스 등록 권리를 가지는 프로젝트도 있다. 프로젝트 품질에 이러한 다른 개발 관행이 미치는 영향은 앞으로 더 세밀한 연구가 필요하다고 사료된다.

나. 프로젝트의 절차

프리 소프트웨어의 개발에는 많은 단계적 절차들이 개입되지만, 대부분은 어디에도 문서화 되어 있지 않으며, 개발자들이 그 절차들을 묵시적으로 따른다.

⊙ 참여: 프로젝트들은 잠재적 구성원들이 프로젝트에 참여하기 위해서는, 대부분 문서화되어 있지 않은, 그 프로젝트에서 정한 절차를 따를 것을 요구한다. 이러한 절차는 프로젝트에 따라 상당히 다양하다. 몇몇 프로젝트는 높은 품질의 결과물을 기대할 수 있는 개발자들에게만 참여를 허용하는 것을 명시적으로 밝히고 있는 반면에, 다른 프로젝트들은 더 개방적이다. [18]에서 프로젝트 참여를 위한 가입 양식 하나를 볼 수 있다.

⊙ 릴리즈: 릴리즈 정책은 프로젝트에 따라 다르지만 많은 경우 프리즈 단계를 거친다. 기능 프리즈는 새 기능을 코드에 추가하지 않고 버그를 고칠 충분한 시간을 벌기 위하여 한다. 또한 최근에는 문자열 프리즈가 인기를 얻어가고 있다. 그것은 다른 나라의 언어로 소프트웨어를 번역하는 개발자들이 소프트웨어 릴리즈 전에 사용된 문자열에 대한 번역을 할 수 있는 기회를 제공한다.

⊙ 브랜치: 이는 안정적 버전과 개발 버전을 유지하는 것처럼 프로그램 버전들 간의 차별화를 위해 사용된다. Arch와 같이 브랜치 관리를 쉽게 만들어주는 새로운 개발 도구는 개발 프로세스에 막대한 영향을 끼쳤다.

⊙ 다른 개발자에 의한 소스 리뷰: 전형적으로 버전 관리 시스템을 통한 소스의 수정은 다른 프로젝트 구성원들에 의해 검토되지만, 많은 경우에 이런 형태의 피어 리뷰가 공식화 되어있지는 않다 즉 개발자들은 자신이 수정한 것을 다른 프로젝트 구성원들이 검토해 주기를 원하지만 실제로 그렇게 하는 것을 보장할 방법이 없다. 다른 한편, 일부 프로젝트는 이 절차를 상당히 엄격히 실행하며 소스 등록을 하기 전에 반드시 코드를 검토할 것을 요구한다.

⊙ 테스트: 새로운 릴리즈가 프로젝트의 기준을 만족시키는지, 주요한 퇴보가 없는지를 확인하기 위하여 일부 프로젝트는 테스트 체크리스트를 가지고 있다. 이 리스트는 가장 중요한 기능들을 포함하고 있으며, 어떻게 실험할 것인가를 간략하게 설명하고 있다. 릴리

즈는 테스터들이 다른 플랫폼에서 체크 리스트의 내용을 하나하나 검사하고 새 버전이 중요한 프로그램의 중단이나 이전 기능이 동작하지 않는 퇴보 현상이 나타나지 않았음을 확증하고 난 이후에 이루어진다. 자동 시험 도구를 가지고 있는 프로젝트는 거의 없기 때문에, 테스트 체크 리스트를 가지고 있는 것은 적어도 중요한 구성 요소와 기능의 작동을 보장하기 위한 좋은 해결책이 될 수 있다.

⊙ 품질 보증: 일부 프로젝트는 드러난 오류들을 제거하기 위해 버그 데이나 버그 잡기 파티 등을 개최한다. 이 과정에서 중복된 버그 리포트를 확인하고, 또 수정된다.

다. 문서화

많은 인터뷰 응답자가 자신이 참가하고 있는 프로젝트의 개발 관행에 대한 문서화가 미흡하다고 비판했다. 프로젝트에 참가하고 기여하는 방법에 대한 명확히 설명한 문서를 가지고 있는 프로젝트는 별로 없다. 어떤 경우에는, 개발자가 제출한 소스 코드가 프로젝트 주 개발자에 의해 검토된 후 정해진 코딩 스타일을 따르지 않았다고 거부당하기도 했는데, 사실 그 코딩 스타일이라는 것이 문서로서 명확히 되어있는 것도 아니었다. 그러나 기여자의 수가 많은 일부 프로젝트들의 대부분은 훌륭한 문서를 가지고 있다.

⊙ 코딩 스타일: 이 문서는 개발자들이 따라야 하는 소스 코드 스타일을 설명하는 문서이다.

⊙ 소스 등록: 누가, 언제 프로젝트의 버전 관리 시스템에 수정된 소스를 등록할 수 있는지 나타내는 문서이다.

5.4.2. 품질 문제

인터뷰에서 확인된 품질 문제들은 대개, 프리 소프트웨어 커뮤니티에서, 해결되어야 할 것으로 생각되어 왔던 이슈들이었기 때문에 특별한 관심을 요한다. 앞부분에서 언급된 많은 개발 관행들이 프로젝트에 쉽게 채택되고 있지 못하고 있는 한편, 앞으로 언급될 품질 문제에 대한 훌륭한 해결책들도 역시 개발되어야 한다.

- ⊙ 지원이 없는 코드: 아직 해결되지 않은 문제들 중 하나는 과거에 코드가 제공되었지만 지금은 유지 보수가 되지 않는 코드를 어떻게 하느냐이다. 한 기여자가 잘 알려지지 않은 하드웨어에 소스를 포팅하거나, 특별한 기능을 추가한 소스 코드를 등록을 할 수 있다. 원래 소스에 대한 수정이 다른 개발자들에 의해 되면서, 이런 특별한 기능이나 포팅 결과도 지속적인 개정을 해야만 계속 사용될 수 있다. 그러나 불행히도 소스에 기여한 개발자 중 일부는 사라질 수 있고,그 코드는 기술 지원과 유지 보수 없이 남겨진다. 주 개발자들은 이런 상황을 어떻게 다룰 것인지 결정해야 하는 어려운 상황에 직면한다.
- ⊙ 설정 관리: 많은 프리 소프트웨어와 오픈 소스 프로젝트들은 높은 수준의 커스터마이징이 가능하도록 개발된다. 이는 사용자에게 많은 유연성을 제공하는 반면, 테스트 문제를 야기한다. 주 개발자는 모든 경우의 수를 테스트해 보는 것이 매우 어렵거나 불가능하기 때문에 흔히 사용되는 환경에서만 테스트해보는 경향이 있다. 따라서 새로운 버전의 소프트웨어가 릴리즈 되면, 새 버전이 자신의 환경에서 동작하지 않는다는 버그 리포트가 많은 것이 일반적인 일이다.
- ⊙ 보안 업데이트: 많은 경우에 보안 업데이트는 적시에 이루어지나

때로는 몇 주 혹은 몇 달 동안 이루어지지 않는 경우도 있다.

⊙ 사용자가 버그 리포팅 방법을 모르는 경우 : 전문적인 기술이 별로 없는 사용자들이 프리 소프트웨어를 더 많이 사용함에 따라 개발자들은 쓸모없는 버그 리포트를 더 받게 된다. 많은 경우 사용자는 버그 리포트를 할 때, 충분한 정보를 포함하지 않거나 중복된 버그 리포트를 제출한다. 그런 버그 리포트는 개발자들의 시간을 빼앗을 따름이다. 일부 프로젝트에서는 버그 리포팅 방법에 대한 더 상세한 설명서를 만들기도 했지만, 많은 사용자들은 버그 리포트 전에 그 설명서를 읽지 않는다.

⊙ 자원 봉사자들의 영입: 일부 프로젝트-특히 아주 인기 있는 프로젝트가 아닐 경우가 직면한 문제 중 하나는 자원 봉사자들을 참여시키는 문제이다. 보통 프로젝트에 기여하는 방법에는 코딩, 테스팅, 버그 선별 등 여러 가지가 있다. 그러나 많은 잠재적 구성원들은 새로운 소스 코드 개발에만 관심이 있을 뿐, 테스트나 버그를 선별하는 일에 관심 있는 기여자는 별로 없다. 그 결과, 개발자들은 그들의 시간 중 많은 부분을 다른 사람들이 쉽게 해결할 수 있는 작업에 사용해야만 한다.

⊙ 문서화의 미흡: 앞의 문제점은 문서화의 미흡과 관련이 있다고 말할 수 있다.자원 봉사자들은 어떤 한 영역에 기여를 하고 싶어 하는데 어떻게 시작해야 하는지 모를 수 있다. 실제 잠재적 기여자들이 받을 수 있는 도움은 거의 없고, 문서도 거의 존재하지 않는다. 또한 개발자 문서의 부족은 또한 모든 사람이 같은 기술과 진행 과정을 따르고 있는지 확신할 수 없다는 것을 암시한다.

⊙ 조정과 의사소통의 문제: 일부 프로젝트에는 조정과 의사소통의 문제가 있으며 이는 프로젝트 품질에 나쁜 영향을 줄 수 있다. 때로는 어떤 영역에 대한 책임이 누구에게 있는지 명확하지 않기 때문에 버그에 대해 적절한 의사소통을 할 수 없는 경우도 있다. 또한 작업이 중복될 수도 있고 치명적인 버그를 없애기 위한 조정이

미흡할 수도 있을 것이다.

요약하면, 이 인터뷰는 프리 소프트웨어 프로젝트에서 프로젝트 품질에 잠재적 영향력을 가지고 있는 몇 가지 두드러진 이슈들을 확인했다.

5.5. 검토와 향후 연구 방향

이 연구에서 행한 인터뷰는 프리 소프트웨어의 품질에 관한 많은 몇 가지 흥미로운 관점을 제시해 주었다. 눈에 띄는 관찰 결과 하나는 프리 소프트웨어 프로젝트들에 사용된 개발 관행의 다양성이다. 인터뷰에 따르면 모든 프로젝트가 따르는 하나의 프리 소프트웨어 개발 모델이 있다고 주장하기 어렵다. 모든 프로젝트는 일반적 속성을 공유하지만, 각 개발 관행 사이에는 분명한 차이들이 존재한다. 다른 개발 관행들이 프로젝트의 품질에 서로 다른 영향을 줄 수 있기 때문에, 이런 개발 관행들과 그 결과 만들어진 소프트웨어의 품질 사이의 연관성에 대해서는 향후 실험적 연구가 요구된다 그러한 연구를 통하여 . 특정 프로젝트에 가장 좋은 개발 관행은 무엇인지, 또는 개발 관행의 적용에 영향을 주는 요인들이 무엇인지 를 제시할 수 있을 것이다.

또한 인터뷰는 분명히 프로젝트들이 많은 문제에 직면하고 있다는 것을 보여주었다. 이 연구의 대상이 되었던 모든 프로젝트는 성공적인 프리 소프트웨어 프로젝트라고 볼 수 있지만,그래도 분명히 품질 개선의 여지가 있다. 기술 지원이 없는 코드의 문제점은 현재 있는 기능의 제거가 사용자에게 영향을 줄 수 있기 때문에 대단히 중요하다. 한편으로는, 자원 개발자에 의존하는 프로젝트들은 개별 기능의 유지 보수뿐만 아니라 프로젝트 그 자체의 유지에 대한 보장조차도 명확하지 않다. 이것은 보수를 받지 않는

자원 개발자를 중심으로 진행되는 프로젝트가 높은 품질의 상품을 만들어 낼 뿐 아니라 높은 수준의 품질을 유지하기 위하여 지속적으로 유지 보수 될 것인지에 대한 일반적인 의문을 더욱 증폭시킨다.

ISO 8402는 품질 보증을 품질에 대한 요구를 만족시키는 방향으로 나아가는 모든 “계획적이고 체계적인 활동들”이라고 정의한다. 많은 프리 소프트웨어 활동에서 볼 수 있는 자원 개발자의 변동성, 무계획적인 개발 등을 고려하면, 프리 소프트웨어 프로젝트가 품질에 대해 계획적이고 체계적인 접근을 할 수 있는지, 그 결과 고품질을 보장할 수 있는지 분명하지 않다. 프리 소프트웨어가 기업 업무 환경과 높은 수준의 안정성을 요구하는 응용에 점점 더 많이 이용됨에 따라, 이러한 의문에 대한 관심은 더 증가될 것이다. 품질에 대한 체계적인 접근의 부족으로 인한 문제들 중 일부는 이미 해결된 것도 있다. 많은 중요한 결점들, 특히 보안과 관련된 이슈들은 매우 빠르게 고쳐진 (종종 1시간 안으로) 반면, 일부는 몇 주 후나 몇 달 후에도 미해결 상태로 남아있다는 것이 인터뷰 중에 지적되었다. 결함 제거에 걸리는 시간은 경우에 따라 상당한 차이가 있으며, 더 체계적인 품질 보증 활동이 적절한 시간 내에 결함을 수정할 수 있게 만들 수 있을 것이다.

최근에 일부 프리 소프트웨어 프로젝트는 품질 보증의 중요성을 깨닫고 전담팀을 구성했다. GNOME는 오랫동안 다양한 품질 보증 관련 작업을 수행했으며[17], Debian은 잘 확립된 QA 팀을 운영하고 있고[2][10], KDE는 2004년 3월에 품질 향상을 위한 작업에 착수했다[8]. 이러한 노력들은 주로 결함이나 더 이상 기술 지원을 받지 못하는 구성 요소를 없애는 것 같은 품질 보증의 기본적 수준에 초점을 맞추고 있다. 그 다음 단계는 최소 수준을 보장하는 것에서 결함 방지와 품질 개선 전략의 이행으로 나아가는 것이다.

품질과 품질 개선에 대한 연구를 위해서는, 실제로 품질 측정을 위한 도구

와 평가 기준이 필요하다. 품질 개선을 위한 기술에 대한 평가는 반복적인 품질 측정을 통해서만 가능하다. 불행히도 품질은 정확하게 정의하기가 매우 힘든 개념이다. 많은 사람들이 무엇이 품질을 수반하는지 직관적으로 느낀다. 그러나 정확한 방법으로 품질을 확인해야 할 때, 그 개념은 명확히 정의하기가 매우 어렵다는 것을 알게 된다[1]. 품질은 다른 많은 요소들로 구성된 개념이다 [16]. 어떤 사람에게는 고품질 제품인 것이 다른 사람에게도 반드시 같은 식으로 받아들여지는 것은 않는다. 예를 들어, 품질에 대한 사용자의 견해는 소프트웨어 개발자와 다를 수 있다. 자원 개발자들은 대개 자신이 사용할 목적으로 소프트웨어를 만들기 때문에 기술적으로 미비한 다른 사용자들의 요구와 품질 , 수준을 고려하지 않는 경향이 있으며, 이 때문에 품질에 대한 사용자와 개발자 사이의 견해 차이는 프리 소프트웨어에서 중요한 의미를 가지고 있다. 상용 소프트웨어 프로젝트가 철저히 유료 고객의 요구에 의해 진행되는 반면, 대부분 자원 개발자에 의해 진행되는 프로젝트에서는 사용자와 개발자 사이에 조정이 거의 없다. 품질에 대한 바람직한 접근은 넓은 시각을 유지하며 소프트웨어에 대한 여러 다른 요구들과 품질의 속성에 대한 고려를 하는 것이다.

이 연구는 어떤 부분을 개선할 수 있는가에 관한 몇 가지 시작점들을 제시하였다. 프리 소프트웨어 프로젝트의 품질과 관련된 다른 문제들에 대하여는 더 많은 샘플을 가진 연구가 앞으로 필요하다. 또한 품질 개선 전략을 만들기 위해서는 품질의 여러 측면들을 측정하고 평가하기 위한 도구와 평가 기준이 필요하다. 이 부분은 학계와 프리 소프트웨어 커뮤니티의 협력으로 상승효과를 얻을 수 있는 영역이다. 앞으로의 품질에 관한 실험적인 연구는 품질을 측정하는 도구와 평가 기준의 개발에 달려있다. 이러한 도구들은 다시 프리 소프트웨어의 개발자들에게 제공되어 자신들의 프로젝트의 품질을 평가하고, 그 결과를 효과적인 품질 개선 전략 개발의 출발점으로 사용할 수 있을 것이다.

5.6. 결론

이 논문은 프리 소프트웨어 개발자와 실시한 인터뷰를 통하여, 품질 측면에서 프리 소프트웨어 프로젝트를 관찰하였다. 또한 이 논문에서는 품질과 관련된 소프트웨어 개발 관행이 어떤 것인지와 더불어 프리 소프트웨어 프로젝트가 직면하고 있는 몇 가지의 품질 문제에 대해 기술하였다. 이 연구에서 제기된 여러 이슈들은 품질 개선 전략의 요소와 함께 프리 소프트웨어 프로젝트의 품질에 대한 앞으로의 연구를 위한 출발점으로 사용될 수 있을 것이다.

보수를 받지 않는 자원 개발자들에 의한 프로젝트가 높은 수준의 품질을 보장할 수 있는지에 대한 의문이 남아있기 때문에, 프리 소프트웨어의 품질에 관한 향후 연구는 매우 중요하다. 향후 연구로 이 연구에서 발견되지 않은 다른 품질 문제들을 확인하는 것과 프로젝트의 품질에 대한 여러 다른 개발 관행들의 영향을 평가할 것을 제안하였다. 마지막으로, 품질을 측정하고 평가할 수 있는 도구들이 필요하다는 것을 주장하였으며, 그 도구는 프리 소프트웨어의 품질에 대한 향후 학계의 연구와 프로젝트 품질을 개선하기 위한 프리 소프트웨어 개발자 모두를 위해 도움이 될 것으로 사료된다.

참고 문헌

[1] B. Dale and H. Bunney, Total Quality Management Blueprint. Oxford, UK: Blackwell Business, 1999.
[2] "Debian quality assurance team," http://qa.debian.org/, accessed March 31, 2005.
[3] M. E. Fagan, "Design and code inspections to reduce errors in program development," IBM Systems Journal, vol. 15, no. 3, pp. 182 211, -1976.
[4] J. M. Gonz´alez-Barahona, M. A. Ortu˜o P´erez, P. de las Heras Quir´os, J.Centeno Gonz´alez, and V. Matell´an Olivera, "Counting potatoes: the size of Debian 2.2," Upgrade, vol. II, no. 6, pp. 60-6, December 2001.
[5] J. M. Gonz´alez-Barahona, G. Robles, M. Ortu˜o P´erez, L. Rodero-Merino, J.Centeno Gonz´alez, V.

Matellan-Olivera, E. Castro-Barbero, and P. de-las Heras-Quir´os, "Analyzing the anatomy of GNU/Linux distributions: methodology and case studies (Red Hat and Debian)," in Free/Open Source Software Development, S.Koch, Ed. Hershey, PA, USA: Idea Group Publishing, 2004, pp. 27-8.
[6] T. J. Halloran and W. L. Scherlis, "High quality and open source software practices," in Proceedings of the 2nd Workshop on Open Source Software Engineering. Orlando, FL, USA: ICSE, 2002.
[7] J. Howison and K. Crowston, "The perils and pitfalls of mining SourceForge," in Proceedings of the International Workshop on Mining Software Repositories (MSR 2004), Edinburgh, UK, 2004, pp. 7-1.
[8] "KDE quality team," http://quality.kde.org/, accessed March 31, 2005.
[9] S. Koch and G. Schneider, "Effort, cooperation and coordination in an open source software project: GNOME," Information Systems Journal, vol. 12, no. 1, pp. 27-2, 2002.
[10] M. Michlmayr, "Managing volunteer activity in free software projects," in Proceedings of the 2004 USENIX Annual Technical Conference, FREENIX Track, Boston, USA, 2004, pp. 93-02.
[11] M. Michlmayr and B. M. Hill, "Quality and the reliance on individuals in free software projects," in Proceedings of the 3rd Workshop on Open Source Software Engineering. Portland, OR, USA: ICSE, 2003, pp. 105-09.
[12] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," ACM Transactions on Software Engineering and Methodology, vol. 11, no. 3, pp. 309-46, 2002.
[13] E. S. Raymond, The Cathedral and the Bazaar. Sebastopol, CA: O'Reilly & Associates, 1999.
[14] D. C. Schmidt and A. Porter, "Leveraging open-source communities to improve the quality & performance of open-source software," in Proceedings of the 1st Workshop on Open Source Software Engineering. Toronto, Canda: ICSE, 2001.
[15] A. Senyard and M. Michlmayr, "How to have a successful free software project," in Proceedings of the 11th Asia-Pacific Software Engineering Conference. Busan, Korea: IEEE Computer Society, 2004, pp. 84-1.
[16] I. Sommerville, Software Engineering. Essex, England: Pearson Education Limited, 2001.
[17] L. Villa, "Large free software projects and Bugzilla," in Proceedings of the Linux Symposium, Ottawa, Canada, 2003.
[18] G. von Krogh, S. Spaeth, and K. R. Lakhani, "Community, joining, and specialization in open source software innovation: a case study," Research Policy, vol. 32, no. 7, pp. 1217 1241, -2003.
[19] L. Zhao and S. Elbaum, "A survey on quality related activities in open source," Software Engineering Notes, vol. 25, no. 3, pp. 54-7, 2000.
[20] - "Quality assurance under the open source development model," The Journal of Systems and Software, vol. 66, pp. 65-5, 2003.

6. 리눅스 활성화 정책방향

정보통신부 임종태

6.1. 서 론

현재 정보통신부에서 프리웨어(Freeware)인 리눅스를 기반으로 정보통신 산업을 활성화시키기 위한 방안으로 국내에서 리눅스 사용을 확대하고 기술 개발을 촉진시키기 위한 방안을 계획하고 있다. 본 고를 통해 이 방안의 추진 배경, 목표 및 계획 등에 대해서 설명하고자 한다.

'91년 핀란드의 Linus Torvalds라는 한 대학원생에 의해 개인적 취미로 개발되기 시작한 PC 기반 운영체제 리눅스는 지난 10년 동안 눈부신 발전을 거듭하여 지금은 세계 PC 소프트웨어 시장을 석권하고 있는 마이크로소프트사 Windows 운영체제의 경쟁자로 부각될 만큼의 괄목할 발전을 이룩했다. 초기의 리눅스는 인텔 프로세서 전용으로 개발되었으나, 현재는 Alpha, Sparc, PowerPC 등 거의 모든 프로세서용 버전이 개발 되었고, 활용분야도 초기에 서버용에 국한되었으나 점차 거의 모든 정보통신 분야로 확산되어 가고 있다. 이러한 리눅스의 활성화로 인해 다가오는 21세기에는 마이크로소프트사가 독주해 왔던 세계 소프트웨어 시장 및 정보통신 분야 판도에 커다란 변화 가능성이 예측되고 있다. 이러한 변화에 대처하기 위해 현재 각 국의 컴퓨터 관련 업체들은 리눅스 기반 기술 및 새로운 응용 개발 연구를 통해 리눅스 시장 선점을 위한 활발한 노력을 경주하고 있는 중이다. 외국에 비하면 아직 부족하지만 국내에서도 최근 학계, 산업체, 연구소 및 일반인 등 각 계에서 리눅스에 대한 관심이 빠른 속도로 증가하고 있고, 활성화 노력이 다양한 각도에서 진행되고 있다. 정보통신부에서는 리눅스의 활성화가 한국과 같이 아직 소프트웨어 기술력이 선진국에 비해 떨어지고 정보화사업에 많은 외국 소프트웨어 제품에 의존해

야 하는 국가에게는 세계 수준의 첨단기술의 습득, 막대한 소프트웨어 비용의 절감, 그리고 소프트웨어 수출국으로 발돋음 하는 기회를 제공할 수 있을 것이라는 판단아래 정부 차원에서의 활성화 방안을 마련하여 추진하고 있다.

6.2. 추진목표 및 전략

6.2.1. 추진목표

정보통신부의 리눅스 기반 정보통신 산업 활성화 방안의 추진목표는 크게 세 가지로 나뉘어 볼 수 있다.

첫째, 프리웨어인 리눅스의 대중화를 통한 보편적 이용 환경 조성이다. 저가격·고성능의 리눅스를 이용한 PC 보급 확산으로 인터넷 이용환경 구축 예산을 절감하고 교육정보화 등 정보화 사업에 리눅스를 적용하여 국민의 보편적 이용환경을 조성하고자 한다.

둘째, 기반 기술 확보를 통한 소프트웨어 기술 경쟁력 강화이다. 아직까지 국내의 소프트웨어 개발 능력은 선진국에 비해 많이 미흡한 수준이다. 리눅스에서는 원천코드를 포함한 충분한 기술 자료가 공개되어 있고 외국의 시스템 소프트웨어 전문가들과의 기술교류 기회를 손쉽게 가질 수 있어, 충분한 연구 및 기술 개발 노력이 경주 된다면 국제적 수준의 첨단 소프트웨어 기술을 확보할 수 있을 것으로 보여 진다.

셋째, 애로 기술 개발 지원을 통한 상품 경쟁력 강화이다. 리눅스의 보급 활성화와 함께 데스크탑 응용 등 리눅스 관련 제품의 경우 조기 개발이 되면 우리와 비슷한 문화권에 속한 중국, 싱가폴 등 아시아 시장을 선점할 수 있고 더 나아가 21세기 수출 산업화가 가능할 것으로 기대 된다.

6.2.2. 추진전략

활성화 방안의 추진전략은 다음과 같다.

첫째, 리눅스 보급 확산을 위한 기초 환경을 조성한다. 이를 위해 원활한 한글 적용이 가능하도록 다양한 한글 폰트를 확보・보급하고 시스템 메시지 및 용어에 대한 표준화를 지원한다. 또한 리눅스 기반 기술에 대한 연구개발을 연구소 및 학계를 중심으로 추진하여 핵심 기반 기술을 확보함과 동시에 소프트웨어 기술 인력을 양성하도록 한다.

둘째, 다양한 응용프로그램 개발을 촉진한다. 편리하게 사용할 수 있는 워드프로세서, 프리젠테이션 도구 등과 같은 응용프로그램들이 개발되도록 지원한다. 이를 위해 리눅스 응용제품 개발에서 요구되는 기반기술을 갖고 있는 국내 대학 및 연구소들과 응용 제품 개발을 수행할 민간기업 간의 효과적 협력 체제를 구축하여 기반기술이 관련 산업체에 신속하게 이전될 수 있도록 한다.

셋째, 공공분야에 대한 리눅스 보급을 촉진한다. 국가정보화사업 추진 시 리눅스 적용이 적극 검토될 수 있도록 하고, 교육정보화 사업의 기본 장비로 리눅스를 활용하는 시범사업을 추진한다. 그리고 리눅스 관련 세미나, 워크샵 및 리눅스 포럼 등의 개최를 통해 리눅스를 홍보한다.

6.3. 추진계획

추진계획은 크게 표준화, 연구개발 및 기술지원, 보급지원, 홍보 및 교육 등 네 가지 분야로 나뉘어 볼 수 있다. 각각을 구체적으로 살펴보면 다음과 같다.

6.3.1. 표준화

지금까지 시스템 메시지 및 용어의 한글화는 여러 업체 및 사람들에 의해 개별적으로 추진되어 왔다. 그 결과 동일한 시스템 메시지 및 용어가 다르게 번역되어 사용자의 혼란을 야기 시켰다. 이러한 문제점을 시정하기 위해 한글 용어에 대한 표준화를 추진한다. 이러한 표준화에 함께 리눅스 시스템에서 한글을 유연하게 표현할 수 있는 미려한 폰트(font)를 확보하여 보급한다. 또 한 전문가 위주로 되어 있는 리눅스 관련 정보를 일반 사용자들이 이해하기 쉽고 활용하기 쉬운 사용자 지침서, 운영자 지침서 형태로 개발하여 보급한다.

6.3.2. 연구개발 및 기술 지원

리눅스 기술개발에 관한 지원은 두 가지 관점에서 추진될 예정이다.

첫째, 핵심 기반기술을 연구소 및 대학 등을 통해 개발함으로 산업체에서 이를 활용하여 상용화할 수 있게 한다. 이러한 관점에서 산업체에서 주문하고 있는 대표적인 리눅스 기반 기술은 여러 리눅스 서버를 연결하여 하나의 대용량 서버처럼 작동하게 하는 클러스터링(Clustering) 기술과 클러스터에서 일부 컴퓨터에 장애가 발생하여도 계속해서 서비스를 제공할 수 있게 하는 고가용성(High Availability) 기술이다. 이러한 기술들은 현재 고가의 유닉스 서버들에서만 제공되는 기능인데 리눅스에서 제공될 경우 고부가가치의 리눅스 서버 개발이 가능해 진다. 현재 국내에서 일부 몇 개의 중소기업에서 추진하고 있으나 아직 확장성, 안정성, 고성능 등 여러 관점에서 많은 개선이 요구되는 수준에 있다.

이러한 기술들 외에도 리눅스에서 시급히 요구되는 기능은 효과적인 서버 관리 환경이다. 대표적인 PC 기반 운영체제 중의 하나인 Windows

NT에 비교하면 표준화된 서버 관리도구들이 존재하지 않고 존재하는 관리도구들도 사용하기 어렵다. 이러한 문제점은 서버 관리 전문가들이 아닌 사람들이 서버를 관리하는 것을 매우 어렵게 한다. 이 문제를 해결하기 위해 효과적인 시스템 진단 및 관리 소프트웨어에 관한 연구 및 기술개발이 요구된다.

둘째, 산업체 기술 자문 및 애로 기술 개발을 지원하는 것이다. 국내 리눅스 벤처기업을 대상으로 기술적인 자문과 애로 기술을 공동으로 개발하는 것이다. 예를 들면, 리눅스 시스템용 주변장치를 만드는 기업이 자체적으로 장치 구동기를 만들 수 없을 때 연구소와 공동으로 장치 구동기 개발하는 것 등이 될 것이다. 또한, 리눅스 유틸리티 개발을 적극 유도하여 배포판에 포함될 수준의 프로그램을 작성하는 기반을 조성한다. 예를 들어, 국내 대학 컴퓨터 프로그래밍 관련 동아리들이 대부분 컴퓨터 게임에 관심이 많은 현실인데 이들 동아리들이 리눅스 유틸리티 프로그램 개발에도 관심을 가질 수 있는 환경을 조성한다. 가능한 방법으로는, 리눅스 유틸리티 공모전등이 추진될 수 있다. 이러한 활동을 통해 리눅스 전문가가 양성될 수 있을 것으로 기대된다. 데스크탑 개발 지원은 리눅스 매니아, 리눅스 회의론자, 리눅스 단순 상용자 모두가 인정하는 리눅스의 커다란 취약점은 다양한 데스크탑 응용 프로그램들의 부족이다. 특히 표계산, 문서편집기, 프리젠테이션 도구, 그래픽도구 등 일상 업무에서 빈번하게 요구되는 도구들은 PC 환경에 비해 많이 미흡한 형편이다. 이는 PC 환경에 친숙한 사용자들이 리눅스로의 전환을 고려할 때 곧바로 직면하는 문제이어서, 일부 컴퓨터 전문가들을 제외한 대부분의 일반 사용자들 사이에서 리눅스를 활성화하는데 커다란 장애요인이 되고 있다. 이러한 문제는 기술적으로 특별하게 응용 프로그램 개발이 어렵기 때문이 아니라 리눅스가 처음에 만들어질 때부터 컴퓨터 전문가들을 위한 운영체제로 개발되었고 그러한 방향에서 지금까지 발전되어온 데서 기인한다. 따라서 이 문제는 기업들이 적극

적으로 리눅스용 응용 프로그램 개발에 나서면 해결될 문제이지만 아직 리눅스 응용프로그램 시장은 규모가 작고 전망이 불투명하여 적극적으로 개발투자를 하는 기업들이 많지는 않다. 그러나 최근에 여러 소프트웨어 업체들에서 응용 프로그램 개발을 시작하고 있어 곧 상황은 많이 개선되리라 예상된다. 리눅스 응용프로그램 시장은 이미 정보화가 상당히 진행된 선진국보다는 앞으로 정보화를 추진해야 하는 정보화 개발도상국들(대표적으로 아시아 국가들)에서 활성화될 것으로 보인다. 이는 정보화가 막대한 투자비를 요구하는데 리눅스로 추진할 경우 엄청난 비용절감 효과가 있기 때문이다. 이러한 시장들은 위와 비슷한 문화권에 속해국내에서 데스크탑 응용 등 리눅스 관련 제품이 조기 개발되면 시장을 선점할 수 있어 21세기 수출산업화가 가능할 것으로 보인다. 국내의 리눅스 관련 응용프로그램 개발 업체들은 대부분 소규모 중소기업들이여서 막대한 개발비와 장기적 안목의 제품개발을 감당하기 어렵다. 이러한 기업들이 대학 및 연구소에서 개발된 리눅스 관련 기반기술들을 활용하여 많은 개발 비용이나 인력 없이 신속하게 응용제품 개발을 진행할 수 있게 한다. 또한 개발된 제품들의 홍보기회를 최대한 제공하여 상품화를 촉진한다.

6.3.3. 보급지원

최근에 각 분야에서 다양한 활성화 노력들이 경주되고 있으나 아직 이러한 노력들이 효율적인 협조체제를 이루지 못하고 있다. 이러한 문제점을 해결하기 위해서 한국정보통신진흥협회내에 순수 민간차원의 리눅스 협의회를 구성해서 운영하고 있다. 이 협의회는 표준화, 연구개발, 보급지원, 홍보 및 교육의 4개의 분과위원회를 두어 산・학・연의 리눅스 전문가들이 참여해서 리눅스 보급 확산을 위한 활동을 추진하고 조율하게 될 것이다.

교육정보화 사업에 리눅스를 활용하여 교육예산을 절감시키고 컴퓨터

교육이 특정 제품들에종속되는 것을 막도록 한다. 이를 위해 우선 리눅스 장비를 활용한 교육정보화 시범사업을 추진하려고 한다. 또한 공공기관의 정보화사업 수행시 리눅스 서버 활용이 활성화될 수 있도록 시스템 도입 RFP에 다른 OS와 같이 리눅스 규격을 포함할 예정이며, 공공기관에서 리눅스 서버의 시범적 사용(웹서버, 메일서버, 프린터 서버 등)을 권고할 계획이다. 또한 최근에 보급되기 시작한 인터넷PC의 기본운영체제로 MS Windows와 함께 리눅스를 선택사양으로 채택하여 소기의보급효과를 거두고 있다.

6.3.4. 홍보 및 교육

리눅스 관련정보 홈페이지를 운영을 통해 사용자가 원하는 정보를 One-Stop으로 제공하며 의견 수렴의 장으로서 활용하려고 한다. 이를 통해리눅스 관련 통계자료, 정부 정책소개, 성장 예측치, 용량산정 자료 등을 제공하게 될 것이다. 국내외에 현재 존재하고 있는 리눅스 관련 홈페이지들과는 차별되면서도 최대한 협조체제를 유지하도록 추진될 계획이다. 예를 들어, 공신력이 있는 정보들만을 선별하고 체계적으로 분류하여 제공하는 데 주력할 수 있을 것이다.

리눅스 전문 기술인력 양성을 지원할 예정이다. 대학 교육 과정에서 리눅스 교육이 활성화되도록 권장할 것이며 국가인정 자격증 시험에 다른 OS와같이 리눅스에 관한 시험과목을 삽입하는 방안도 추진할 계획이다. 리눅스 관련기술 및 국제적 동향 소개, 정보공유 등을 위한 리눅스 포럼 및 워크샵을 산·학·연 공동으로 개최하여 리눅스 관련정보와 기술교류를 통해 리눅스 개발 방향을 선도할 예정이다. 이러한 기회들을 통해 리눅스 현안들에 대해서 공동 대처가 가능할 수 있을 것으로 기대한다.

리눅스에 대한 초·중·고교생, 대학생 및 일반인 등의 관심을 증폭시

키기 위한 경진대회 개최도 추진할 계획이며 국내 리눅스 기업들의 국제적 교류 활성화를 위한 국제행사의 개최도 계획하고 있다.

6.4. 요약

이상에서 소개된 정보통신부 리눅스 활성화 방안을 요약하면 다음과 같다. 1991년 개발된 리눅스가 최근 저비용・고효율・안정성 등의 여러 가지 이점 때문에 대표적인 PC 서버용 운영체제로 부각되고 있다. IBM, Intel, Sun 등 세계적인 하드웨어 및 소프트웨어 업체들이 공식적으로 리눅스 지원을 시작하면서 Windows NT에 대적하는 새로운 서버 운영체제로 부각되었고, 이에 따라 다른 여러 업체들도 리눅스 시장을 선점하기 위해 활발한 개발활동을 전개하고 있다.

본 활성화 방안은 국내에서 리눅스 활성화를 통해 다음의 목표를 달성하기 위해 마련되었다. 추진목표는 첫째, 저가격·고성능의 리눅스를 이용한 PC 보급확산으로 인터넷 이용환경 구축 예산을 절감하고 교육정보화 등 정보화사업에 리눅스 적용을 통해 국민의 보편적 이용환경을 조성하는 데 있고, 둘째, 리눅스 기반기술 개발을 통해 소프트웨어 기술을 국제수준으로 향상시키고, 셋째, 리눅스의 보급 활성화를 통한 국내 소프트웨어 산업을 육성하는 데 있다.

구체적 추진전략은 다음과 같다. 첫째, 리눅스 한글환경 표준화 및 리눅스 기반기술 개발 등과 같은 원활한 리눅스 보급을 위한 연구개발을 추진한다. 둘째, 리눅스 환경하의 다양한 응용 개발 촉진체계를 구축한다. 이를 위해 산·학·연 협력 체계를 구축한다. 셋째, 공공분야에 대한 리눅스 보급을 추진한다. 정보화사업 추진 시 리눅스 활용을 적극 추진하고 교육정

보화 사업의 기본장비로 활용하는 시범사업을 추진한다. 또한 산・학・연의 리눅스 전문가들로 리눅스 협의회를 구성하여 리눅스 보급 확산을 위한 활동을 추진하게 될 것이다. 이 과정에서 기업들의 요구사항 및 이용 활성화에 대해서 협의가 이루어질 것이다.

새 천년에는 시작부터 국가 간에 치열한 정보화 경쟁이 있을 것이다. 정보화 환경이 얼마나 저비용・고효율로 잘 구축되었느냐, 그리고 정보화 인력을 얼마나 갖고 있느냐가 그러한 국제사회에서의 경쟁력을 좌우하게 될 것이다. 이러한 관점에서 리눅스의 활성화는 매우 중요하다. 본 활성화 방안이 성공적으로 수행되어 우리나라가 아시아의 리눅스 메카로 떠올라 새천년에 우리나라가 정보화 선진국에 진입하도록 정부에서는 최대의 노력을 기울일 방침이다.

7. 리눅스 확대를 위한 공개소프트웨어 기술 지원센터의 역할

한국소프트웨어진흥원 이영재

7.1. 개 요

한국소프트웨어진흥원 공개소프트웨어지원센터에서는 공공기관을 필두로 공개소프트웨어를 확산시키는 활동을 수행하고 있다. 지금은 많이 나아졌지만 2005년도 초기만 해도 공개소프트웨어와 리눅스 기반으로 웹서버와 DB를 연동하는 시스템 구축에 대해서조차도 그 안정성은 둘째 치고 기술적인 가능성에 대한 의구심을 떨쳐버리지 못하였다. 많은 언론 보도와 경험의 축적다양한 프로젝트의 성공에 따라 2005년도 말에 와서 는 대부분의 공공기관 전산 담당자들은 공개소프트웨어 기반에서 시스템을 구축하는 것에 대한 기술적 제약은 없음을 인정하고 있다. 하지만 여전히 공개소프트웨어를 사용하였을 때 발생하는 문제에 대한 기술지원 가능성에 대한 불안감을 떨쳐 버리지 못하고 있는 것이 사실이다. 한국소프트웨어진흥원은 공개소프트웨어 활성화의 일환으로 이러한 기술지원의 문제를 해결하고자 2004년도부터 공개소프트웨어 기술 지원센터를 구축하고 공개소프트웨어와 공개소프트웨어 기반의 솔루션들에 대한 테스트 및 공공 사용자에 대한 기술지원 서비스도 제공하고 있다. 또한 윈도우나 유닉스 환경의 솔루션들이 공개소프트웨어 기반위에서 실행되도록 포팅하도록 유도하고 개발된 솔루션의 고도화를 돕기 위한 테스트 등을 수행하고 있다.

공개소프트웨어 비즈니스모델의 중심에는 기술지원 서비스가 있다. 장기적으로 보아서는 예산을 사용해 민간 기업의 핵심 비즈니스 모델인 기술

지원 서비스를 무상으로 제공하는 것은 바람직하지 않다. 기업이 기술지원을 체계적으로 수행할 수 있는 수익과 인력을 확보하고 Know-How를 습득하며 시장에 원활히 접근해 서비스를 판매할 수 있는 마케팅 능력이 생긴다면 당연히 민간 기업의 유료화 된 서비스화가 되어야 한다. 이에 따라 기술 지원센터 구축 초기에 한국소프트웨어진흥원의 직원으로 기술 지원센터를 구성해 직접 운영할 것인지 기업들의 컨소시엄을 통해 구축할 것인지를 놓고 고민을 하게 되었다.

현재 기술지원센터의 구성은 공개소프트웨어 관련 기업들의 컨소시엄을 통한 사업자들로 구성되고 해당 컨소시엄의 인력을 파견 받아 기술 지원 활동을 수행한다. 그 과정에서 엔지니어들은 다양한 솔루션을 접해 보고 테스트하며 기술지원 체계를 익혀 향후 귀사 했을 때 기존 접했던 기술지원 체계를 자사에 적용하고 다양한 기업의 엔지니어들 간의 협력 통해 공개소프트웨어 분야 기술지원에 있어 시너지를 창출할 수 있는 네트워크를 만들 수 있는 기회를 가지게 되었다. 또한 전국적 기술지원망을 각 지역의 공개소프트웨어 관련 기업을 지정하여 구축함으로써 지역의 기술지원과 함께 공개소프트웨어 비즈니스 활동의 주도적인 역할을 할 수 있는 기회를 가질 수 있었다.

이와 같이 공개소프트웨어 기술 지원센터는 공공기관에 대한 공개소프트웨어 사용상 문제에 대한 실질적인 기술 지원, 각종 솔루션의 포팅과 성능 고도화 지원 그리고 기업의 기술지원 역량 강화를 통한 공개소프트웨어 활성화의 역할을 목적으로 다양한 과제를 수행하여 왔다.

7.2. 공개소프트웨어 현황

7.2.1. 리눅스의 역사

[표 1]은 리눅스가 발전되어 온 과정을 간략히 정리한 것이다. 1991년 사용자 수 1은 물론 리누스 토발즈를 의미한다. 10년간 천 만명이 사용하는 발전을 거쳐 2001년도에 커널 2.4가 발표되었고 2003년도에 2.6이 발표된다. 1999년도에 레드햇이 기업을 공개한 이후로, 그리고 커널2.4와 2.6을 발표되어 엔터프라이즈 급의 기업에서 리눅스를 사용할 수 있게 되면서부터 리눅스는 이제 일부 커뮤니티의 매니아나 앞선 기술 수용자들의 전유물이 아니다. 리눅스는 표 2와 같이 이제 서버시장에서 윈도우에 이어 두 번째의 OS를 차지하고 있다.

표 1 연도별 리눅스 발전

연도	버전	소스 라인 수	사용자 수
1991년	0.01	10,000	1
1992년	0.96	40,000	1,000
1993년	0.99	100,000	20,000
1995년	1.2	250,000	500,000
1997년	2.1	800,000	3,500,000
1998년	2.1.11	1,500,000	7,500,000
1999년	2.2	1,600,000	10,000,000

표 2 세계 클라이언트 및 서버 OS 점유율(2002-2007)

(단위:%)

구 분		2002	2003	2004	2005	2006	2007	CAGR
클라이언트 OS	윈도우	93.8	93.9	93.6	93.2	92.8	92.2	7.1
	맥	2.9	2.6	2.4	2.2	2.0	1.8	-2.2
	리눅스	2.8	3.2	3.8	4.4	5.1	6.0	25.4
	기타	0.5	0.3	0.2	0.2	0.1	0.0	-35.2
서버 OS	윈도우	55.1	58.1	60.1	60.5	60.0	58.8	10.5
	리눅스	23.1	23.9	25.3	27.2	29.5	32.3	16.6
	유닉스	11.0	10.0	8.8	7.7	6.6	5.7	-4.5
	넷웨어	9.9	7.4	5.4	4.3	3.6	2.9	-14.4
	기타	0.9	0.6	0.4	0.3	0.3	0.3	N.A

리눅스 커널의 발전과 함께 미국시장에서는 1999년과 2000년도에 리눅스 붐이 형성되었다. 1999년 8월에 Redhat은 미국 나스닥에 상장하며 $53까지 달하는 실적을 올렸다. 1999년 12월에 IPO를 단행한 VA리눅스의 경우 최대 $30의 IPO 가격에 대해 $279에 이르는 10배 가까운 실적을 올려 세계를 놀라게 하기도 하였다. 하지만 2001년 VA리눅스가 리눅스 사업을 포기를 발표하면서 그 다음날 주가는 $2.61달러에 이르는 부침을 겪기도 한다. 이러한 부침은 우리나라에서도 비슷하게 있었다. 1999년 말에서 2000년에 이르러 리눅스코리아, 리눅스원, 와우리눅스, 한컴리눅스 등 많은 리눅스기업과 알짜, 미지, 한컴리눅스, 터보, 액셀 같은 다양한 배포판들이 만들어지게 된다. 커뮤니티에서 활동하는 인원이 폭발 하였으며, 리눅스 발표회에는 1000명에 이르는 관객이 참여하기도 하였다. 그러나 그 열기는 미국에서와 같이 오래갈 수 없었다. 리눅스 확산을 지속할 수 있는 인프라가 부족했고 시장 확대에 장애가 되는 다양한 문제점들이 해결 되어야만 했다.

7.2.2. 공개소프트웨어 확산의 장애요인

[그림 1]은 공개소프트웨어 도입의 저해 요인을 분석한 내용이다. 위의 조사에 의하면 기술지원의 부족이 가장 큰 문제로 인식되고 있다. 이 자료는 그 대상이 10억달러 이상의 대규모 기업을 대상으로 하였음에 따라 저해요인의 우선순위는 국내의 경우와 다소 다를 수 있더라도 공개소프트웨어에서 기술지원 부족 문제의 해결 필요성에 대하여 잘 설명해 주는 자료가 될 것이다.

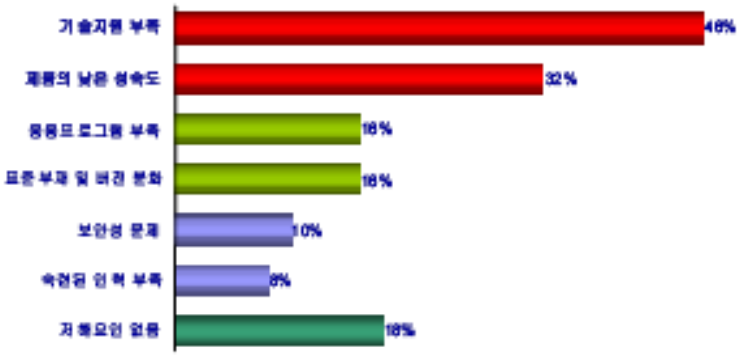


그림 1 리눅스 도입 저해 요인

제품의 낮은 성숙도 또한 공개소프트웨어 확산의 주요한 장애 요인으로 분석되어 있다. 글로벌 기업의 상용 솔루션들은 리눅스를 타켓으로 이미 오래 전부터 리눅스 기반으로 포팅되어 시장을 확산해 가고 있는 상황이다. 그러나 소프트웨어 기업의 규모가 작은 국내 기업 상황에서는 응용프로그램의 질을 높이고 완성도를 높이는 데에 많은 투자를 하기 어려운 것이 현실이다. 리눅스 OS를 제품화하고 리눅스 기반에서 비즈니스를 하고 있는 기업들의 개별적인 수익성을 본 기고에서 상세히 논할 것은 아니나, 아직 성공적으로 리눅스 비즈니스를 추진하고 있는 기업에서조차 공개소프트웨어 분야에서 수익을 내고 있는 상황은 아니다. 향후의 수익을 위해 지속적인 투자를 진행하고 있는 상황이다. 더욱이 아직 그 규모가 작은 관계로 공개소프트웨어 프로젝트의 주 사업자는 대부분 SI 기업이 수주하게 되고 공개소프트웨어 기업은 그 하청 역할을 하고 있다. 리눅스 OS만으로는 수익성이 없고 특화된 솔루션을 함께 가지고 있어야만 생존을 위한 기업의 존속 가능성이 있다 하겠다.

기술지원 부족과 제품의 낮은 성숙도라는 두 가지 중요한 문제점을 해결하기위해 정부에서는 2004년 7월 한국소프트웨어진흥원 내에 10월 공개소프

트웨어 기술 지원센터를 공식 개소하였다.

7.2.3. 레드햇의 기술지원 모델

리눅스 비즈니스에서 가장 성공적인 비즈니스를 진행하고 있는 기업이 레드햇이다. 레드햇은 기술지원을 그 기본적인 서비스로 활용하고 있고 기업의 수익 확보를 위해 그 모델을 참조 할 필요가 있다. Redhat의 서비스 핵심에는 온라인과 오프라인상의 기술지원(주로 온라인 기술지원)과 Redhat 리눅스 OS 제품에 대한 패치서비스(온라인 기술지원의 핵심)가 있다.

가. 주요 특징

⊙ 소스 레벨은 무료이나 바이너리 코드 제공은 서비스에 해당

⊙ 서비스를 판매하며, 소프트웨어패키지는 서비스에 포함되는 것으로 해석. (소프트웨어 라이센스는 없고 Upgrade와 패치, 기술지원이 비즈니스 모델)

⊙ Subscription

- 온라인 지술지원은 무제한
- 연간 기술지원 서비스 계약
- RHNetwork 접속 (CD 이미지-구버전 포함 + Up-to-Date버전)
- 옵셔널 24 X 7, On Site, Dedicated technical account Manager

⊙ RedHat 현황

- 개발자 300명 Support, Sales 엔지니어 포함 500명 총 950명
- 글로벌 매출 2004년 2억불, 2005년 3.36억불 예상

⊙ 기술지원 현황

- 7년간 해당 버전 OS 지원, 2.5년간은 신기술을 구 버전에 접목
- 12~13개월에 신버전 출시, 분기별 Update Patch

- 초기 신제품 버전을 GA(General Availability) 버전이라 하고 이 버전으로 인증받은 제품이 문제 없이 돌아갈 수 있도록 Patch 제공
- 2.4 ⇒ 2.6일 때 Backward 호환 라이브러리 제공
- 본사에서는 On-Line 지원만을 하고 있고 On-Site 요청에 대하여는 교육을 유도
- 아태 지역만 일부 On-Site 교육을 지원하며 Channel 비즈니스를 하고 있음. 미국의 경우 대형 유통망을 이용한 직접 판매
- 서비스 L1(10분내 처리) L2(1시간내 처리) L1, L2는 총판이 L3은 레드햇만
- Severity 레벨에 따라 S1(개발자까지 stand-by) S2, S3, S4로 구분

국내 리눅스 기업들과 공개소프트웨어 솔루션 기업들도 이와 유사한 모델을 적용할 필요가 있다. 공개소프트웨어인 리눅스에서 최저 가격은 "0"에 가까워질 수 있음에 따라 해당 리눅스를 사용하면서 얻을 수 있는 지원에서 진정한 가치를 인정받을 수 있다. 공개소프트웨어 기술 지원센터에서는 민간 기업이 이러한 모델을 사업에 연계시킬 수 있도록 부족한 부분을 보완하는 역할을 수행하고 있다. 물론 리눅스 뿐 아니라 리눅스 상에서 운용되는 다양한 솔루션을 포함해 그 대상으로 삼고 있다.

7.3. 공개소프트웨어 기술 지원센터

7.3.1. 중앙 기술 지원센터

공개소프트웨어 기술 지원센터는 그림 2와 같이 공개소프트웨어 시장 활성화, 공개S/W기반의 Knowledge Base 구축, 공개소프트웨어 기반 국산 소프트웨어산업 자생력·경쟁력 강화를 목적으로 2004년 10월 한국소프트웨어지흥원내에 개소하였다. 첫째 년도인 2004년~2005년 말까지는 한글

과컴퓨터 컨소시엄에 슈퍼유저코리아, 리눅스원, 케이컴스, NTC코리아, 모아시스 등이 참여 하였고, 2차년도에는 삼성전자 컨소시엄에 엘지엔시스, 아이겟리눅스, 클루닉스, 케이컴스(큐브리드), 와우리눅스, 동해정보통신이 참여해 기술지원센터를 운영하고 있다. 또한 2차년도부터는 공공에 공개소프트웨어 도입 자문역할을 강화하기 위해 아이겟리눅스, 리눅스코리아의 인력을 추가 배치하고 있다.

그림 2 공개소프트웨어 기술지원센터 비전

중앙 기술 지원센터의 우선 기술지원 대상은 공개소프트웨어 비즈니스를 영위하고 있는 기업이다. 사용자에 대한 기술지원은 1차적으로 제품을 판매한 기업이 담당하는 것이 당연하다. 이 과정에서 협력이 필요한 문제나 제품 고도화를 위한 테스트가 필요한 경우 중앙 기술 지원센터의 장비와 인력을 활용한 테스트 지원을 수행한다. 또한 테스트 환경을 구축하고 있는 HW 기업을 연계하는 역할도 하고 있다. 공개소프트웨어 기반 솔루

션을 테스트를 통과한 제품이 있는 기업에 대하여는 공개소프트웨어 Solution Partner 인증을 제공함으로써 중소기업 제품이 시장에서 인지도 제고를 돕는다. Solution Partner 인증을 받은 제품에 대해서는 GS(Good Software) 인증을 보다 쉽고 빠르고, 저렴하게 획득할 수 있는 방안도 강구하고 있다.

기술지원 요청에 대하여는 KMS시스템에 정보를 구축, 저장 하고 해당 내용을 외부의 기업들도 활용할 수 있도록 하였다. 현재 KMS 안에는 4천여 건의 자료가 등록되고, 리눅스 기반의 기업과 기업의 솔루션을 함께 구축하고 있다. 이에 따라 필요시 솔루션별, 기업별로 쉽게 대상을 찾아 활용할 수 있게 함으로써 공개소프트웨어 기반 솔루션 확산에 기여하고 있다. KMS 시스템은 고객의 요청과 업무 분배, 처리 등의 프로세스를 브라우저 상에서 처리토록 시스템화 되어 있으며, 이 프로세스를 따르면 기술지원을 수행 할 수 있도록 하는 체계를 마련하였다. 민간에 공개소프트웨어 기술지원센터의 역할이 이양될 때 해당 시스템을 함께 이양하여 운영할 수 있도록 하였다.

현재 중앙 기술 지원센터에는 12명의 서버 OS, 데스크탑, 응용프로그램, HW, 네트워크, 보안 등의 다양한 분야 전문가로 구성되어 있다. 다양한 분야에 전문가를 함께 확보할 수 없는 공개소프트웨어 관련 중소기업으로서는 중앙 기술 지원센터를 활용함으로써 사업 능력을 배가 할 수 있을 것으로 기대한다. 이들 중 공공기관에 공개소프트웨어 도입을 위한 자문 역할을 하는 인력2인이 포함되어 직접 자문 을 수행하고 있다. 시군구 정보화 사업, 기획 예산처 중소기업 공개소프트웨어 도입 방안, 행정정보 DB 구축 사업, 우정사업본부의 인터넷 뱅킹 등 관련 부처 공개소프트웨어 도입에 기술적인 자문을 수행함으로써 공개소프트웨어 확산을 지원 하고 있다.

중앙 기술 지원센터는 그림 3과 같이 ETRI에서 개발하고 있는 부요

개발에 대한 기술지원도 수행한다.

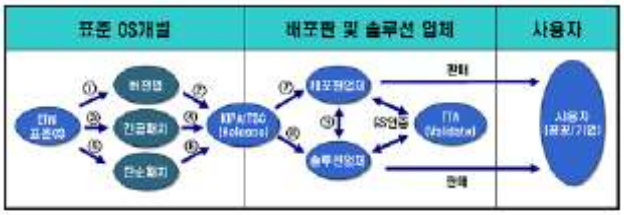


그림 3 OS 기술지원

앞서 RedHat의 기술지원 모델에서처럼 리눅스 OS는 기술지원과 패치 등의 업그레이드 서비스가 주요한 요소이다. 소프트웨어에 있어 테스트는 많을수록 좋다. 물론 중복되지 않고 잘 관리되고 테스트 결과가 제대로 반영되어야 한다. 중앙 기술 지원센터에서는 부요에 대한 테스트를 수행하고 그 결과를 부요 개발팀에 피드백 하며, 개발팀의 패치에 대한 분배의 역할도 담당하고 있다. 향후는 중요 업데이트 및 패치에 대한 글로벌 커뮤니티 결과의 적용을 위한 테스트 활동을 계획하고 있기도 하다.

7.3.2. 지역기술지원

공개소프트웨어 기술지원을 위한 체계로 중앙 기술 지원센터 이외에 지역 기술지원이 있다. 중앙 기술 지원센터와 마찬가지로 본격적인 지역 기술지원 사업이 시작되기 전에는 지역 기술 지원센터를 전국의 주요 거점에 직접 만드는 것을 구상하였다. 그러나 이것은 중앙 기술 지원센터와 마찬가지의 딜레마를 갖게 했다. 기업의 수익 모델이 기술지원이고 이를 공공기관에서 직접 수행하는 것은 바람직하지 않다는 것이다. 이에 따라 각 지역의 기업 중에서 지역 기술지원을 수행할 기업을 모집한 컨소시엄을 선정하는 방식으로 지역 기술지원을 수행하였다. 이에 따라 지역 기술 지원센터는 별도로 구축되지 않았고 지역 기술지원이라는 이름으로 통칭해 사

용한다. 올해에는 전국의 16개 지역 기술지원 기업들이 선정되었다. 작년의 14개에 이어 30여개의 기업이 지역에서 기술지원 활동을 수행하였다. 컨소시엄 주관사는 리눅스 기업으로 선정되었고 이 기업들은 전국에 자사의 기술지원 인프라와 제품 판매 인프라로 활용할 수 있는 기회를 가지게 되었다. 1차년도 한글과 컴퓨터, 올해에는 아이겟리눅스가 선정되었다. 제대로된 기술지원 체계를 전국적으로 확대하기 위해서는 정부의 예산을 통해 운영되는 지역 기술지원 기업으로는 한계가 있다. 전국적인 인프라 확대를 위해 일정한 자격 요건을 갖춘 Support Partner를 전국에 걸쳐 모집하게 되었다. Support Partner의 목적은 기본적인 기술지원 인프라 구축 이외에 기술지역 인력을 육성케 하고 공개소프트웨어 확산 협력을 통해 기업 활동 비중을 자연스럽게 공개소프트웨어 분야로 확장시키도록 유도하는 데에도 있다. 현재 34개의 지역 기술 지원 기업이 선정된 상태이며 올해 50개까지 그수를 확대 하고자 한다. 2010년까지 150개의 지역 기술 지원 기업 확보를 통하여 정부의 공개소프트웨어 정책이 없더라도 기업들의 힘으로 공개소프트웨어 기반 소프트웨어사업에 지장이 없게 되기를 기대한다.

그림 4 지역 기술지원

7.3.3. 데스크탑 기술지원

리눅스 서버의 경우는 글로벌 HW 및 소프트웨어 기업의 활발한 리눅스 개발 참여 그리고 Unix용 응용 프로그램은 리눅스용으로 포팅이 용이함에 따른 소프트웨어의 높은 완성도를 기반으로 리눅스 확산에 많은 기여를 하였다. 데스크탑용 응용프로그램들은 주로 커뮤니티 중심의 공개소프트웨어 개발 모델을 따라 개발되었고 그 종류도 브라우저, 오피스, 멀티미디어, 커뮤니케이션, PIMS, 보안, 게임, 그래픽, 유틸리티 등 서버에 비해 종류가 훨씬 다양하다. KDE와 Gnome으로 양분된 그래픽 사용자 환경은 공개소프트웨어의 다양성 측면에서는 바람직하지만 개발 기업에게는 응용프로그램을 두 가지 그래픽 환경에서 완성도를 높여야 하는 부담을 가중시킨다. 또한 리눅스 버전마다 다른 필수 라이브러리 버전도 응용프로그램 개발에 어려움을 주는 장애 요인이 되었다. 더욱이 공개소프트웨어인 리눅스 데스크탑은 리눅스 배포판 사업자에게 가격 면에서도 큰 장점을 가지기 어렵고 이미 윈도우가 대부분의 데스크탑 OS로 사용되고 있는 상황에서 그 자리를 비집고 새로운 시장을 창출하기란 여간 어려운 일이 아니다. 우리나라의 경우 PC에서 윈도우 점유율은 99%이고 리눅스 데스크탑의 점유율은 0.3%를 밑돌고 있는 실정이다.

이러한 다양한 문제들로 인해 작은 기업 규모와 투자 여력으로 기업들이 리눅스 데스크탑에 투자하는 것이 어렵고 윈도우와 같은 완성도를 요구하는 고객의 요구에 많은 개발비가 들어가는 다양한 솔루션들을 개발하지 못하는 것은 어쩌면 당연한 일인지 모른다. 그렇다고 리눅스 데스크탑의 성공이 전혀 불가능한 일은 아니다. 오픈오피스가 2005년도 10월에 2.0이 발표되면서 그 완성도가 매우 높아졌다. 국내에서는 ThinkFree Office가 출시되어 MS Office와의 파일 호환성 문제와 리눅스에서 사용이 가능해졌다. 커널 2.6이 출시되면서 리눅스 자체의 기능 또한 크게 개선되고,

FireFox가 개선되면서 윈도우에서 가능한 많은 기능들을 리눅스상에서 사용할 수 있게 되었다. OpenDocument 파일 포맷은 OASIS에서 표준으로 제정하고 ISO를 통해 각국에서 표준화를 추진 중이다. 이러한 다양한 소프트웨어 풀과 표준화의 노력이 지속적으로 확대되고 있는 상황에서는 이미 대부분의 시장을 장악하고 있는 현실을 개선할 수 있는 가능성은 증대한다 할 수 있다.

아직까지 데스크탑에 대한 기술지원의 문제 발생이나 요청은 크지 않다. 사용자 수가 적기 때문인데, 실제 기술 지원 요청이 증가하는 시점에서는 서버에 비해 기술지원에 대한 요구가 폭증할 가능성이 많은 분야가 리눅스 데스크탑이라 하겠다. 또한 리눅스 커뮤니티 확산 시점이 바로 데스크탑이 확대되는 시점이 될 것으로 생각한다. 리눅스 데스크탑 기술지원 문제 해결은 기술지원 체계나 인력 확대 이전에 이러한 문제 발생의 근본적인 요인을 해결 할 필요가 있다. 문제 해결의 가장 기본적인 것이 표준화이다. TTA를 통해 리눅스 데스크탑에 대한 표준 스펙 지정을 추진하고 있다. 여기에는 한국소프트웨어진흥원, ETRI, 한글과컴퓨터, 아이겟리눅스 등이 참가하여 2005년 12월 단체 표준으로 지정되었고 현재 두 번째 표준화가 진행 중이다. 보다 바람직한 방향은 윈도우가 한 회사 제품으로 일관성을 갖는다는 측면과 같이 리눅스 데스크탑이 경쟁력을 갖기 위해서는 사용자에 대해 일관성 있는 사용자 환경을 제공하고 응용프로그램도 이러한 인터페이스를 일관성 있게 제공해야 한다. 특히 각사의 라이브러리 버전 차이에 따른 설치 시 호환성 문제나 사소한 문제 발생 여지를 없애는 것에 기업들의 노력이 배가되어야 할 것이다. 리눅스 데스크탑을 커뮤니티로서 자유정신에 따른 협력의 결과물로 바라보는 시각에서는 독창성이나 참여자의 자유의사가 중요할 수밖에 없다. 한국소프트웨어진흥원에서도 커뮤니티의 개발 활동을 적극 지원하고 있다. 그러나 리눅스 기업이 비즈니스 모델로 리눅스 데스크탑을 활용하는 경우 제품으로서의 리눅스와 응용프로그램은 완성도와 사용상의 친밀도 면에서 커뮤니티의 그것과는 달라야 할

것으로 생각한다. 커뮤니티에게도 이러한 점을 이해하고 협력할 수 있기를 바란다. 예를 들어 기업에서는 “휴지통”이라 번역하고 커뮤니티에서는 “쓸모없는 것”이라고 번역한 경우 독창성 면에서는 이해할 수 있지만 상품으로 제품에 적용되기에는 “휴지통”이 더 적절할 것으로 보인다. 사용자에게 거부감이 없을 것이기 때문이다. 글로벌 커뮤니티에서 기업의 번역이 받아들여지지 않는 경우 새 버전이 나올 때마다 기업은 불필요한 재번역 작업을 반복해야 한다. 결국 국내 커뮤니티에서는 글로벌 커뮤니티에서 “휴지통”으로 번역된 내용이 반영되도록 협조하는 것이 결국 목적은 다르지만 리눅스 데스크탑 확산에 주요한 기여를 할 것이다. 응용프로그램 개발 회사의 측면에서는 표준화 문제는 더욱 절실한 실정이다. 다양한 리눅스 OS마다 개발인력을 투입해 포팅을 달리해야 하는 상황에서는 응용프로그램의 확대를 기대하기 어렵다. 실제로 미세한 중요 라이브러리의 차이로 인해 많은 경우 버그의 원인을 찾고 또는 찾지 못하고 미완성의 제품을 시장에 출시하는 경우가 허다하다. 특히 인력이 작고 영세한 국내 개발사 입장에서는 미완성의 가능성 높아 리눅스용 응용프로그램은 완성도가 낮다는 오명을 떨쳐버리기 어렵게 되는 것이 현실이다.

7.4. 결 론

공개소프트웨어의 사업 모델은 서비스를 바탕으로 하고 있다. 레드햇의 주요 비즈니스 모델은 페도라 프로젝트를 통해 개발비용을 절감하고, 서비스를 중심으로 사업을 전개해 나가고 있다. 우리나라에서는 정부의 예산과 기업의 컨소시엄을 통해 부요 표준을 개발하고 공개소프트웨어기술지원센터를 통해 서비스를 지원한다.

장기적인 개발 주체를 확보하기 위하여 부요 프로젝트는 커뮤니티 중심의 한중일 공개 OS 프로젝트로 발전해 나가는 단계에 있으며, 기술 지원센터는 그 뒤를 굳건히 떠받치는 역할을 수행 하고자 한다. 특히 다양하게 발생하는 데스크탑 기술지원 문제 해결은 한 기업이 감당하기에 벅찬 일이 아닐 수 없다. 표준화가 된 배포판에 대해 여러 회사들이 관련 엔지니어를 공유하는 기술지원 협력을 통해서 문제를 해결 하는 것이 보다 바람직한 방향이며 그 장을 제공하는 것이 공개소프트웨어 기술 지원센터이다. 내년도에는 리눅스 데스크탑 활성화를 위해 현재의 서버 중심 기술 지원센터를 데스크탑을 강화하는 쪽으로 무게중심을 이동할 예정이다. 공개소프트웨어 활성화 특히 리눅스 데스크탑 활성화가 멀고도 쉽지 않은 길이지만 공개소프트웨어 기술 지원센터가 확산의 장애들을 제거해 나가는 지뢰 제거반이 될 수 있도록 많은 관심과 조언이 필요하다.

공개소프트웨어 기술 지원센터의 역할은 언젠가는 그 역할을 제대로 수행할 민간 기업이 대신 해야 할 일이다. 그 시점이 빨리 올 수 있어야 한국소프트웨어진흥원에서 추진하는 공개소프트웨어 활성화 정책 전반이 성공했다는 평가를 받을 수 있을 것이다. 시장이 커지도록 그리고 관련 OS와 솔루션 기업이 커지도록 공개소프트웨어 바탕으로 소프트웨어 강국이 될 수 있는 절호의 찬스를 살릴 수 있도록 지원하는 일이 공개소프트웨어 기술 지원센터가 갖는 지상 최대의 임무이다.

참고문헌

[1] “Revolution OS,” Wonderview Productions, 2001

[2] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office

[3] http://www.redhat.com/rhel/compare/server/

[4] Martin Fink, “리눅스와 오픈소스의 비즈니스와 경제학“, p.31, 영진닷컴, 2005.

8. 공개소프트웨어 기반 환경에서의 소프트웨어 개발 교육

국민대학교 황선태·정낙수·허대영

8.1. 서 론

소프트웨어에 대한 교육을 하려면 소프트웨어가 무엇인가에 대해서부터 명확한 인식이 필요하다고 할 수 있다. 이런 경우 직접적인 정의가 어렵다면 존재의 가치를 따져보는 것도 의미 있는 일이라고 할 수 있겠다.

혹 예술 작품이라면 만들어서 전시해 놓고 감상하는 것이 존재의 이유이겠지만 소프트웨어는 그렇지 않은 것이다. 아마 사람의 의사를 컴퓨터에게 전달하는 매체의 역할을 하는 것이 소프트웨어일 수 도 있겠다. 즉 사용자가 어떤 목적을 가지고 사용해야하는 일종의 도구인 것이다. 이런 존재의 가치를 충족시키려면 소프트웨어는 사용자에 의해서 사용되어야 하는 것이다. 사용되지 않는 소프트웨어는 이미 그 존재의 가치를 갖지 못한 것이 되는 것이다. 개발자가 이런 소프트웨어 존재의 본질을 깊이 이해하고 소프트웨어를 만든다면 훨씬 가치 있는 소프트웨어를 만들 수 있을 것이다.

그렇다면 어떻게 하면 소프트웨어가 사용되게 할 수 있을 것인가? 사실 막대한 예산을 들여서 연구, 개발 되어진 소프트웨어도 사용되지 않고 사장돼버린 예는 쉽게 찾을 수 있다. 이런 관점에서 볼 때 소프트웨어 개발의 개념을 약간 바꿀 필요가 있는 것 같다. 즉 소프트웨어를 개발해서 발표한다는 선언적 의미 보다는 우호적 토양에 소프트웨어를 심고 가꾸어서 자라게 한다는 진행형의 의미를 부여하는 것이 훨씬 성공 확률이 높은

것이다. 공개소프트웨어의 활용은 이런 문제에 대해서 어느 정도 답을 주는 것 같다. 이미 어느 정도 사용자층을 확보하고 있으므로 공개소프트웨어를 기반으로 해서 성능 개선이나 기능 추가 등을 위한 개발을 시도한다면 안정되게 사용할 수 있을 정도의 소프트웨어로 완성하는 일이나 사용자를 확보하는 일 등에서 유리할 수 있다. 여기서 관심 있는 것은 실제로 사용되어지는 소프트웨어 개발에 대한 체험적 교육을 어떻게 하느냐이다. 일반적으로 교육에 적절한 환경이 주어질 때 그 교육의 절반 이상이 해결된다고 볼 수 있다. 공개소프트웨어를 교육하려면 그 교육 환경을 이에 가장 적절하도록 구축하는 것이 우선인 것이다. 즉 교육은 처한 환경에 대해서 반응하는 것에서부터 출발한다고 볼 수 있는데 공개소프트웨어로 구성된 교육 환경을 호기심을 가지고 들여다보는 것만으로도 이미 많은 것을 배울 수 있는 것이다. 게다가 대학의 경우 그 구성원 자체가 개발자 이면서 사용자가 될 수 있다. 즉 자체적으로 개발된 소프트웨어를 강제적으로라도 사용할 수 있는 것이다. 이렇게 소프트웨어가 사용되어지면 그 소프트웨어는 생명력을 가졌다고 볼 수 있는데 여러 가지 사용자 요구 사항이 도출되어 후속 개발로 이어질 수 있는 것이다. 또한 공개소프트웨어로 구성된 교육 환경은 경우에 따라서는 매우 복잡한 시스템 통합 이슈와 세심한 운영 관리의 문제가 야기 되는데 이 또한 대학 구성원의 좋은 경험거리가 될 수 있는 것이다.

본 논문에서는 대학에서 소프트웨어 교육을 위해서 공개소프트웨어를 기반으로 어떤 실습 환경을 구축할 수 있는지를 설명하고 이런 환경에서 도출될 수 있었던 시스템 통합, 시스템 운영 및 소프트웨어 개발 교육 관련 이슈들을 언급하고 몇 가지 개발 사례를 설명한다.

8.2. 공개소프트웨어 기반으로 구축된

소프트웨어 교육 환경 일반적으로 대학교의 소프트웨어 교육을 위해서는 높은 수준의 실습 환경 구축을 필요로 한다. 또한 실습 환경을 다양하게 구성하여 다양한 실습을 할 수 있도록 고려되어야 하고, 관리 및 운영이 용이하도록 설계 되어 최상의 환경을 유지하고 장애발생시 신속하게 복구될 수 있어야 한다. 학교의 특징은 이런 실습 환경을 중심으로 필요로 하는 여러 구성원을 자체적으로 확보할 수 있다는 것인데 그 역할에 따라서 다음과 같이 구분할 수 있다.

⊙ 사용자 : 실습 환경에서 프로그래밍과 같은 실습을한다.
⊙ 개발자: 실습 환경 개선에 필요한 소프트웨어 개발
⊙ 관리자 : 실습 환경에 대한 설계 및 구성과 운영

이런 구분이 서로 배타적인 것은 아니며 거의 모든 구성원이 사용자이면서 그 중 일부는 학년이나 개발 능력에 따라서 관리자 또는 개발자의 역할도 하게 되는 것이다.

높은 수준의 다양한 소프트웨어 교육을 위해서는 실습 환경이 다음과 같은 조건을 만족해야 한다.

가. 사용자 및 개발자를 위한 실습 환경 구성 조건

⊙ 멀티 플랫폼을 지원하여 다양한 환경에서의 실습과 개발, 테스트를 진행할 수 있어야 한다.
⊙ 다양한 플랫폼을 이용하는 만큼 단일화 된 인증시스템을 가져야 한다.

⊙ 실습환경은 일관성 있게 제공되어야 한다. 즉, 실습환경에 사용되는 컴퓨터의 구성 및 설치된 프로그램은 모두 동일하여야 한다.
⊙ 사용자는 실습실의 모든 PC에 대해 사용이 가능 하여야 한다.
⊙ 실습환경에 대하여 사용자별로 권한이 할당되어 원활한 실습이 될 수 있도록 하여야 한다.
⊙ 실습환경 하에서 다양한 개발이 가능하도록 여러 개발 소프트웨어가 설치되어야 하고, 테스트가 가능하여야 한다.

나. 관리자를 위한 실습 환경 구성 조건

⊙ 하드웨어와 소프트웨어 모두 관리자의 관리사항이 최소화되도록 구성되어야 한다.
⊙ 실습실의 모든 PC에 대해 동일한 설정이 적용될 수 있는 시스템이 구성되어야 한다.
⊙ 서버들은 유기적으로 동작하여 계정 관리에 대한 비용이 최소화되어야 한다.
⊙ 사용자가 실습 환경을 변경할 수 없도록 하여야 한다.

이와 같은 요구사항을 충족시키기 위해서는 시스템과 네트워크 구성이 일반 엔터프라이즈 환경보다도 더 복잡한 구조가 될 수 있고 다양한 플랫폼을 포함한 시스템 연동은 완성도가 높지 않으면 장애를 일으키기 쉽다. 또한 공개소프트웨어를 적용하여 시스템을 구성할 경우에는 상용 소프트웨어를 이용하여 구성하는 것보다 더 높은 수준의 시스템에 대한 이해를 필요로 한다. 따라서 관리자 역할을 하는 학생뿐만 아니라 사용자의 경우에도 구축된 시스템을 사용하고 관찰하는 것만으로도 시스템 설계 및 통합에 대한 기본적 이해를 할 수 있게 된다. 또한 공개소프트웨어를 사용하기 때문에 자유롭게 코드를 수정하고 개선할 수 있는 이점을 얻을 수 있으며 이미 완성된 공개소프트웨어를 분석함으로써 소프트웨어 개발 능력을 향

상시킬 수도 있다.

앞에서 언급한 실습 환경의 구성은 사용자를 위한 클라이언트 환경 부분과 클라이언트에 서비스를 제공하기 위한 서버 부분으로 구분되어진다. 전체적인 실습 환경 구성은 그림 1과 같다.

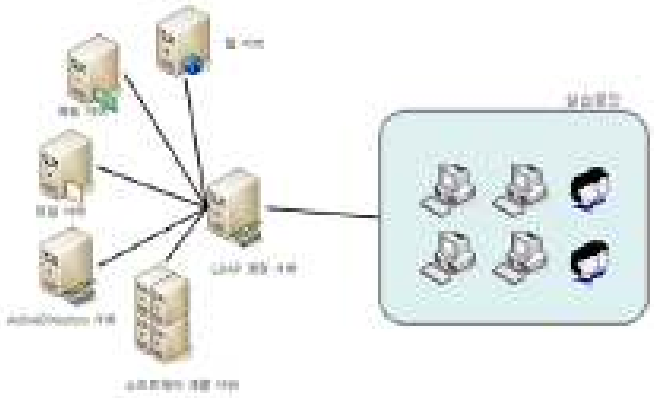


그림 1 실습 환경 구성

8.2.1. 클라이언트 환경 구성

클라이언트 환경은 사용자나 개발자가 임의의 실습환경으로 사용 및 개발을 할 수 있도록 되어야 한다. 기본적인 클라이언트의 환경은 표 1과 같이 구성되어 있는데 기본적으로 공개소프트웨어인 KMU-Linux로 구성되어 있으며 다양한 실습과 개발을 위해 Windows로 멀티 부팅할 수 있다.

표 1 클라이언트 환경 구성

	일반 클라이언트
OS	KMU-Linux[1] WindowsXP[2]
권한	KMU-Linux : 사용자 권한 Windows : 사용자 권한
인증 서버	KMU-Linux : LDAP[3] Windows : ActiveDirectory[4]
용도	수업 및 실습

가. 사용자 관점에서의 클라이언트 환경 구성

사용자는 클라이언트 환경에서 수업 및 실습 을 진행하게 된다. 수업 및 실습은 기본적으로 Linux 클라이언트에서 진행되고 특정 수업의 경우 Windows로 멀티 부팅되게 하여 Windows 환경 하에서도 실습 및 수업이 원활히 진행되도록 구성되었다. 사용자가 클라이언트 환경에서 웹을 사용할 경우에는 특정 포트나 사이트의 접근 금지 등의 제약을 받게 된다. 사용자 및 개발자는 기본적으로 Linux와 Windows 클라이언트를 이용하여 개발을 진행하게 되나 사용권한에 의해 제약을 받을 수 있다. Linux의 경우 root 권한을 얻지 못하므로 커널 프로그래밍이나 KMULinux와 같은 배포판 개발 시에 제약을 받게 된다. Windows의 경우에도 마찬가지의 경우가 발생하게 되는데 이와 같은 경우를 극복하기 위해 일반 클라이언트 실습 공간이 아닌 별도의 개발 공간을 제공한다.

나. 관리자 관점에서의 클라이언트 환경 구성

클라이언트 환경은 관리비용이 최소화 되도록 구성 되어야 한다. 따라서 기본적으로 사용자의 권한을 축소함으로써 클라이언트 환경이 변경될 수 있는 여지를 최소화하고 통합 인증, 네트워크 파일 서비스 등을 통해

각 데스크탑에 불필요한 데이터를 저장하는 것을 최소화하였다.

8.2.2. 통합 인증 서비스

사용자는 실습환경에서 제공하는 웹 서비스, Linux 클라이언트, Windows 클라이언트를 비롯하여 ssh, 네트워크 파일 서비스 등과 같은 인증이 필요한 다양한 서비스에 접근하게 된다. 이 때 당연히 사용자의 인증절차를 거쳐서 접근 권한이 확인된 사용자에 대해서만 서비스를 제공하게 된다. 그러나 서비스간 인증이 통합되지 않을 경우 각각의 서비스 별로 독자적 인증절차를 거쳐야만 해당 정보에 접근이 가능하다. 통합 인증 서비스는 이러한 서비스의 인증 절차를 통합하여 서비스에 대한 접근을 용이하게 한다.

가. 사용자 관점에서의 통합 인증 서비스

사용자는 실습환경의 서비스들에 단일 ID/PAS 소프트웨어D를 통해 접근한다. 단일 ID/PA S소프트웨어D를 사용할 경우 클라이언트에 접속하는 것만으로 SSO(Single Sign On)[5]가 되어 다양한 다른 서비스에 추가 인증절차 없이 편리하게 접근할 수 있게 되고 하나의 서비스에서 패스워드를 변경하면 다른 서비스에도 변경된 패스워드를 이용할 수 있다. 또한 클라이언트 환경의 모든 데스크탑에 부팅된 플랫폼에 상관없이 로그인이 가능하다.

나. 관리자 관점에서의 통합 인증 서비스

계정 관리 서버만 관리함으로써 관리 포인트를 단일화 시킬 수 있으며 관리 비용을 최소화할 수 있다.

다. 통합 인증 서비스의 구성

Linux에서의 대표적인 인증 서비스로는 전통적인 PAM을 들 수 있으며 PAM에서 확장된 NIS, LDAP, Kerberos[6]를 통한 인증 역시 가능하다. 기본적으로 PAM, NIS 등은 Linux 클라이언트를 위해 제공되는 인증 서비스이며 LDAP의 경우 다양한 시스템과 연동이 가능하다. Windows의 경우 SAM과 ActiveDirectory가 있는데 최근의 Windows에서는 ActiveDirectory가 기본적인 인증 서비스이다.[7]

라. LDAP을 통한 인증 서비스

기본적인 Linux 계정 정보는 LDAP으로 재구성되어 제공되어져서[8] 클라이언트의 인증처리도 LDAP을 통해 할 수 있도록 구성되어 있다. 또한 웹 서버와의 연동 역시 LDAP을 통한 인증방식을 사용하고 있으며 웹 서비스를 위한 커넥션 모듈을 개발함으로써 홈 페이지 등 웹 서비스를 외주로 개발할 때에도 제공되는 모듈을 사용하여 SSO를 구성할 수 있다.

마. ActiveDirectory를 통한 인증 서비스

Windows 클라이언트의 경우 LDAP을 통한 인증이 불완전하여 클라이언트에 대한 다양한 제어를 위해 ActiveDirectory를 사용하였다. 일반적으로 Active-Directory와 Linux와의 통합인증에는 Kerberos를 이용한 External Trust 방식의 통합인증 방식[9]과 SFU(Service For Unix)[10]를 이용한 계정 동기화 방식이 있다. Kerberos를 이용한 External Trust 방식은 완벽한 SSO를 구성할 수 있다는 장점이 있으나 계정 정보가 Linux에 있는 이상 사용자에 대한 다양한 제어가 불가능하다는 단점이 있다. 반면 SFU를 이용할 경우 사용자와 컴퓨터에 대한 다양한 제어가 가능

하며 Windows 클라이언트 환경에 대해서 정책을 적용할 수 있다는 장점이 있다. 이 경우에는 Windows 클라이언트 환경에 대하여 적절한 정책을 수립하고 배포하기 위해 SFU를 통한 통합 인증 서비스를 채택하였으며 이를 통해 Linux와의 계정 동기화와 Windows 클라이언트의 완벽한 제어가 가능하게 되었다[11].

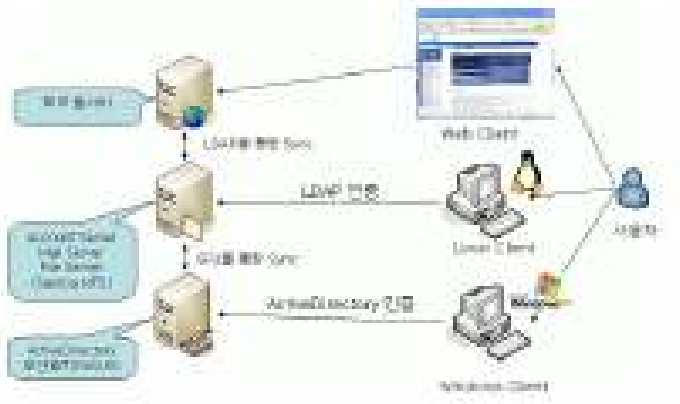


그림 2 통합 인증 시스템 구조

8.2.3. 네트워크 파일 서비스

네트워크 파일 서비스는 서버의 파일 시스템의 일부를 공유하여 해당 데이터를 한 명 이상의 사용자가 사용할 수 있도록 하는 서비스이다. 네트워크 파일 서비스가 필요한 이유는 사용자나 개발자가 클라이언트에서 작성한 데이터를 별도로 보관하지 않고 파일 서버에 저장함으로써 사용자나 개발자에게 편의성을 제공하고 데이터의 안정성을 보장하여 데이터에 대한 신뢰성이 높은 실습 환경을 제공한다.

가. 사용자 관점에서의 네트워크 파일 서비스

사용자는 지정된 데스크탑이 아닌 임의의 데스크탑을 사용하게 된다. 따라서 어느 위치에서나 자신의 데이터에 접근할 수 있도록 파일 서비스가 제공되어야 한다. 또한 데스크탑에 Linux와 Windows 등 다양한 플랫폼이 설치되어 있는 만큼 각각의 플랫폼에서 접근 가능한 파일 서비스가 제공되어야 한다. 그리고 각 사용자에게 할당된 공간은 해당 사용자 이외의 사용자로부터 보호되어야 한다.

나. 관리자 관점에서의 네트워크 파일 서비스

사용자의 데이터가 파일 서버에 집중되어 있으므로 관리가 용이하다. 그러나 백업에 대한 대책이 있어야 하고 보다 강력한 보안 정책을 적용할 필요가 있다. 각사용자별 사용량의 상한선을 관리할 수 있어야하고 사용자 수의 변동에 따른 파일 시스템의 확장 축소가 유연해야 관리가 수월하다.

다. 네트워크 파일 서비스의 구성

네트워크 파일 서비스를 제공하는 서버는 Direct Access Storage로 구성된 Linux 서버이다. 따라서 Linux에서 제공하는 파일 서비스를 통해 클라이언트에 네트워크 파일 서비스를 제공하게 되며 대표적인 공개소프트웨어 네트워크 파일 서비스는 NFS[12]와 Samba[13]가 있다. Linux 기반이므로 LVM을 이용할 수 있다.

라. LVM을 이용한 파일 시스템의 확장 및 축소

파일 시스템의 파티션 단위는 입학년도로 구성되어진다. 동일한 입학년도인 사용자들은 동일한 파티션을사용하게 되며 해당 파티션에 개인 사용자 공간이 생성되게 된다. 각 파티션은 LVM으로 구성되며 기본적인 할당

량에서 사용자의 증감에 따라 파티션의 크기를 유동적으로 확장하거나 축소하게 된다. 또한 물리적 Storage 추가 시에도 LVM을 통해 확장함으로써 유연성을 유지할 수 있다.[14]

마. LDAP+NFS를 통한 Linux 클라이언트 네트워크 파일 서비스의 구성

Linux / Unix의 대표적인 네트워크 파일 시스템으로 LDAP을 통해 사용자의 권한 정보를 제공하고 NFS를 통해 공간을 할당한다.[15] NFS는 기본적으로 사용자에 대해 개인 공간 할당을 하게 되며 시스템 구성을 위한 공간과 개발자가 공동 개발을 위한 공간을 제공한다. LVM을 통해 생성된 파티션을 클라이언트 머신에 할당하게 되며 사용자는 LDAP 인증을 통해 자신의 할당 공간에 접근 허용을 허가받아 사용하게 된다. Linux 클라이언트에서는 사용자가 로그인하면 자동으로 자신의 공간을 사용할 수 있도록 AutoMount 를 이용하여 구성하였다.

바. Samba를 통한 Windows 클라이언트 네트워크 파일 서비스의 구성

공개소프트웨어로 개발된 네트워크 파일 시스템 중 Windows를 지원하기 위한 파일 서비스가 Samba이며 Samba를 통해 Linux서버-Windows 클라이언트간 네트워크 파일 서비스를 구성할 수 있게 된다. 클라이언트에는 사용자가 로그인시 자동으로 자신의 공간을 사용할 수 있도록 ActiveDirectory를 통한 Logon Script를 사용하여 구성하였다.

8.2.4. 클라이언트 일관성 유지

사용자 및 개발자가 실습환경을 사용하는 데 있어 중요한 부분은 어떠한 클라이언트에서도 동일한 작업을 수행할 수 있어야 한다는 점이다. 특정 클라이언트에만 특수한 소프트웨어가 설치되어 있거나 특수한 환경이 구성되어 있고 이를 이용해야만 한다면 사용자는 특정 클라이언트에 종속적일 수밖에 없다.

가. 사용자 관점에서의 클라이언트 일관성 유지

사용자는 모든 클라이언트에 대해 사용이 가능하여야 하며 클라이언트가 제공하는 서비스는 동일하여야 한다. 이를 통해 자유롭게 클라이언트를 변경하여 사용및 실습, 수업을 진행할 수 있다.

나. 관리자 관점에서의 클라이언트 일관성 유지

관리자는 사용자나 개발자가 수업 및 실습을 위해 필요로 하는 사항을 최대한 빠른 시간 내에 반영할 수 있어야 한다. 이때 최소의 관리 인력을 투입하여 최단 시간 내에 보다 많은 클라이언트에 대해 변경이 가능 하여야 한다.

다. 일관성 유지 시스템의 구성

일관성 유지는 크게 시스템 초기화 부분과 소프트웨어 설치 부분으로 나뉘게 된다. 시스템 초기화는 실습환경에서 요구하는 기본적인 사항을 충족시키는 환경 구성을 구축하는 것을 뜻하며 소프트웨어 설치는 추가적으로 발생하는 변경사항을 적용하는 것을 뜻한다.

라. Image Tool을 이용한 시스템 초기화

시스템에 심각한 오류 발생시, 혹은 장치의 문제로 인하여 불가피하게 시스템을 교체나 변경해야 하는 경우 기존 시스템과 동일하게 구성하기 위해서는 정상적인 설치방법을 이용할 경우 많은 관리비용이 발생하게 된다. 따라서 관리비용을 최소화하기 위해 주기적으로 정상적인 시스템을 Image Backup하여 이를 이용하여 Image Restore 함으로써 설치에 필요한 관리비용을 최소화 할 수 있다. 그러나 Image Backup과 현재 시스템과는 Backup 시점에 따라 상이할 수 있기 때문에 이에 대한 보완으로서 시스템 동기화를 이용하여 Image Restore 이후에 발생한 변경사항을 적용할 수있다.

마. rsh를 이용한 시스템 동기화

시스템이 초기화 된 경우, 혹은 추가적인 소프트웨어 설치가 필요한 경우 시스템 동기화를 통해 소프트웨어 설치 및 업데이트를 진행할 수 있다. Linux 클라이언트의 경우 rsh과 같은 서비스를 제공하기 때문에 보다 효과적으로 시스템을 동기화 할 수 있다. Linux 클라이언트는 부팅 과정에서 rsh를 이용하여 서버에서 변경된 사항을 체크하고 변경사항이 있을 경우 해당 내용을 반영함으로써 시스템 동기화가 이루어지게 된다. 또한 정상적으로 설치 및 삭제되었는지에 대한 부분을 체크하기 위해 설치에 관한 로그를 남기는 소프트웨어를 개발하고 이를 통해 로그정보를 관리자에게 제공함으로써 보다 완벽한 시스템 동기화를 구성하였다.

바. ActiveDirectory를 이용한 시스템 동기화

Windows 클라이언트의 경우 소프트웨어 설치뿐만 아니라 보안 업데

이트 역시 중요하다. 따라서 시스템 동기화는 소프트웨어 설치와 보안 업데이트로 구분되어지며 소프트웨어 설치의 경우 ActiveDirectory에서 제공되는 GPO(Group Policy Object)중 소프트웨어 설치 부분을 통해 소프트웨어 배포 및 관리가 가능하다. 그러나 GPO를 통해 배포 가능한 소프트웨어는 MSI(Microsoft Installer)에 대해서만 배포 및 삭제가 가능하기 때문에 이를 보완하기 위해 Logon Script를 이용하여 설치 및 삭제하도록 구성하였다. 또한 보안 업데이트의 경우 WSUS[16]를 통해 자동으로 업데이트하도록 하여 보안 업데이트에 대한 동기화도 가능 하도록 구성하였다.

8.2.5. 네트워크 구성

대학교 실습실의 컴퓨터는 특정 사용자가 전용으로 사용하는 것이 아니며 불특정 다수가 사용할 수 있기 때문에 보안에 매우 취약할 수 있다. 그런데 대학의 경우 실습실 자체에 있는 정보를 보호하는 측면 보다는 웜이나 바이러스에 의한 네트워크의 성능 저하나 다른곳을 해킹하기 위한 경유지로 이용되는 것을 막는 것이 더 중요하다. 따라서 네트워크 구성을 적절히 하여 실습 환경을 웜, 바이러스, 해킹으로부터 보호하도록 하여야 한다.

가. 사용자 관점에서의 네트워크 구성

외부의 네트워크 장애에도 불구하고 실습 환경 안에서는 정상적인 수업 및 실습이 이루어질 수 있어야 한다. 특히 클라이언트에는 사용자의 정보나 데이터가 없고 모든 것을 서버에 의존하도록 되어있기 때문에 중요한 이슈가 된다. 또한 실습 환경 외부에서도 사용자 자신의 데이터에는 접근이 가능해야 한다. 그리고 네트워크 관련 프로그램을 개발하고 테스트 할 수 있는 기본 환경이 마련되어 있어야 한다.

나. 관리자 관점에서의 네트워크 구성

관리자는 사용자 및 개발자가 실습 및 개발에 필요한 네트워크 자원을 제공하고 지원하여야 하나 불필요한 네트워크 서비스는 통제하여 미연에 위험을 방지하여야 한다. 또한 사용자가 네트워크 환경의 설정을 변경 못하도록 제한하여야 하며 네트워크 환경을 관리자가 통제할 수 있어야 한다.

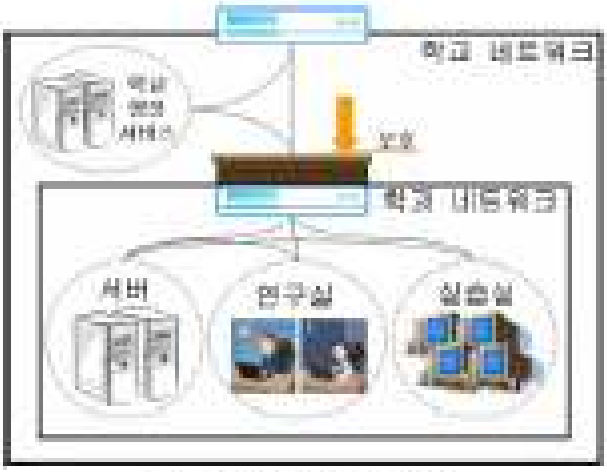


그림 3 네트워크 구조

다. 네트워크 구성

웜, 바이러스, 해킹으로부터 보호하기 위해 네트워크를 섬과 같이 분리하였다. 또한 개발자를 위한 개발 서버를 구성함으로써 네트워크 응용 프로그램 개발이 가능하도록 하였으며 DHCP, DNS 등과 같은 서버를 통해 실습 환경에 대한 네트워크 환경을 관리자가 통제할 수 있도록 구성하였다.

라. IPTable을 통한 네트워크의 분리

대학에서 제공하는 네트워크에서 학부 네트워크를 분리시킴으로써 외부로부터의 위험에 대처가 가능하다. 이는 학부 네트워크를 사설 IP로 구성하고 방화벽을 설치함으로써 가능하며 방화벽은 Linux에서 기본적으로 제공되는 IPTable을 이용하여 구성하였다. 특정 서버의 경우 IPTable에서 NAT(Network Address Translation)를 사용하여 공인 IP를 부여함으로써 외부로부터의 접근이 가능하도록 하였다. IPTable은 제대로 관리하려면 복잡한 커맨드를 자유롭게 쓸 줄 알아야 하지만 대부분의 경우 매우 쉬운 특정 설정만을 변경하면서 관리하게 된다. 이런 단순 관리를 위한 툴을 개발하면 저학년이라도 지침에 의해서 관리의 일부를 맡을 수 있다.

마. POPTOP[17]을 통한 외부 네트워크로부터의 접근

POPTOP은 공개소프트웨어 기반의 Linux용 PPTP(Point-to-Point-Tunneling Protocol) VPN(Virtual Private Network) 서비스이다. 외부의 사용자가 실습 환경 혹은 개발 환경의 자원에 접근하려고 할 경우 네트워크가 분리되어 있기 때문에 NAT로 개방된 서버가 아닌 경우 접근이 불가능하다. 따라서 이와 같이 NAT로 개방된 서버가 아닌 시스템에 접근하고자 하는 사용자에 대해 VPN을 통해 접근이 가능하도록 하고 있다.

바. DHCP/DNS를 통한 네트워크 환경 자동 구성

사용자의 클라이언트는 네트워크 설정은 IP를 지정하지 않고 DHCP로 구성하는 것이 각 클라이언트를 동일한 Image로 관리 할 때 편리하다. 따라서 네트워크 환경을 제공해주는 DHCP 서버가 존재하여야 하며 DHCP

서버 구성을 통해 네트워크 환경 정보를 구성하게 된다. DHCP는 Linux의 dhcpd를 이용하여 구성하였으며 클라이언트의 경우 실습실 공간에 할당받은 네트워크 정보를 제공받게 된다. 클라이언트 환경은 관리자에 의해 관리되는 시스템이며 이동이 발생하지 않는다. 따라서 DHCP를 통해 네트워크 환경을 제공하더라도 네트워크 환경 정보가 유동적이지 않고 고정적이어야 관리가 용이하게 된다. 따라서 DHCP 서버는 MAC(Media Access Control) 고정을 통해 특정 MAC에 대해서는 사전에 정의된 정보를 제공하게 된다. 즉, 클라이언트 A는 항상 동일한 IP Address를 비롯한 동일한 네트워크 정보를 제공받게 되며 구성되게 된다. 그리고 DNS의 경우 Linux 기반의 BIND를 이용하여 구성하였으며 DHCP와의 연동을 통해 클라이언트의 모든 Host, IP 정보를 제공하게 된다.

8.2.6. 보안

대학에서의 사용자는 보안에 대한 개념이 확고하지 않을 수 있으며 실습 환경 또한 외부에 노출되었거나 외부 사용자의 접근이 허용되기 때문에 보안적으로 매우 취약한 구조를 가지고 있다. 또한 의도하지 않았으나 발생할 수 있는 문제에 대해서도 사전에 안전장치가 마련되어야 하며 이를 통해 실습 환경을 보다 견고하게 유지할 수 있다.

가. 사용자 관점에서의 보안

실습환경이 해킹으로부터 보호되어야 하며 스니퍼링과 같은 가로채기에 의해 정보가 유출되지 않도록 시스템이 구성되어야 한다. 또한 응용 프로그램 개발 시 발생할 수 있는 예기치 못한 오류로부터 시스템이 보호되어야 한다. 일반적으로 발생할 수 있는 예기치 못한 오류는 프로그램 오류로 인한 무한 프로세스 생성이나 무한 파일 시스템 점유, 무한 네트워크

시스템 접속 시도 등이 있다.

나. 관리자 관점에서의 보안

관리자는 정책적으로 실습 환경에 대해 불필요한 접근이 차단되도록 하여야 하며 허용된 서버 이외의 서버에 대해 관리자가 아닌 사용자가 접근하는 것을 차단하도록 하여야 한다. 또한 개발서버의 경우 사용자가 원격으로 접속하여 실습할 때 발생할 수 있는 오류에 대하여 방지할 수 있어야 한다.

다. 보안의 적용

KMU-Linux의 경우 충실한 보안을 기본 설계 개념으로 하여 개발되었다. 평문으로 전송되는 서비스를 제공하지 않고 Selinux를 제공하여 최대한 견고하게 구성되었다[18]. 실습환경의 클라이언트의 경우 KMULinux를 사용하기 때문에 telnet이나 ftp와 같은 평문 전송 서비스가 기본 탑재되지 않았으며 설치되지 않도록 구성하였다. 또한 ulimit을 사용하여 자원(CPU, 메모리, 등)의 사용을 제한하였다. ulimit은 생성할 수 있는 프로세스의 수, 메모리 용량, 동시에 열수 있는 파일 수에 대해 제한하도록 구성하였다. 서버의 경우는 관리자가 지정한 IP 이외의 IP에서 접근하지 못하도록 IPTable을 이용하여 접근을 제한하였으며 사용할 수 있는 서비스도 SSH[19]와 같은 안전한 서비스에 대해서만 제한적으로 접속이 허용되도록 구성하였다. 또한 관리자의 IP를 도용할 가능성이 있기 때문에 관리자의 ID 이외에는 접근이 불가능하도록 SSH 구성을 변경하였다. 그리고 네트워크에서 사용하는 모든 정보에 대해 암호화방식을 사용하고 있으며 대표적으로 LDAP의 패스워드 역시 암호화되어 인증을 받도록 되어 있다[20].

8.3. 공개소프트웨어 기반의 소프트웨어 개발

앞에서 설명한 실습 환경은 사용자 입장에서 보면 통합 인증 및 파일 서비스를 제공하는 리눅스/윈도우 통합 환경이며 관리자 입장에서는 구축 및 관리의 난이도가 높지만 클라이언트를 동일하게 구성하고 네트워크 구성 및 보안 정책에 의해 관리 부담을 어느 정도까지는 줄일 수 있는 시스템이다. 이런 시스템에서 시스템 통합을 위해서 또는 사용자 및 관리자의 편의성을 향상시키기 위해서 소프트웨어를 개발할 필요성이 생길 수 있는데 이를 자체 개발하여 바로 이 실습 환경에 적용할 수 있는 것이다. 대학 구성원이 스스로 개발할 수 있는 소프트웨어는 매우 다양하고 범위가 넓은데 이를 유기적으로 묶어주고 개발 동기를 높이기 위해서 중심에 세운 것이 KMU-Linux라는 자체 개발한 Linux 배포판이다. KMU-Linux를 기반으로 했을 때 그림 4와 같은 개발트리가 형성되는데, KMU-Linux 개발, Kernel 개발, 응용 소프트웨어 개발, 모듈 및 라이브러리 개발이 공개소프트웨어 개발 형식으로 진행된다.

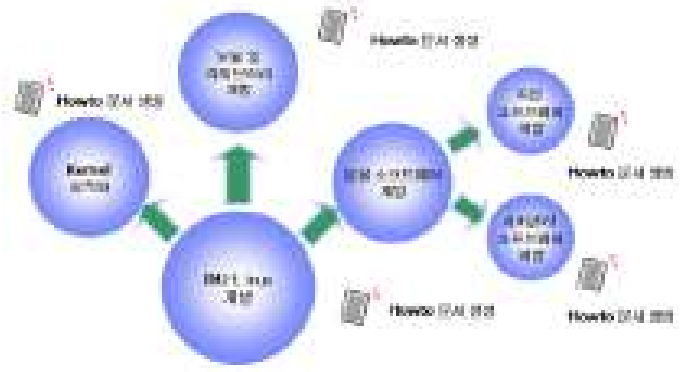


그림 4 공개소프트웨어 개발 구조

8.3.1. 공개소프트웨어 개발 순환 구조

공개소프트웨어 기반으로 구축된 실습 환경을 관리하고 사용하다보면 새로운 소프트웨어 또는 기존 소프트웨어의 업그레이드, 여러 서비스의 기본 설정 변경 등 다양한 요구 사항이 도출 될 수 있다. 이런 요구 사항들은 구성원 안에서 자체적으로 개발팀을 구성하여 공개소프트웨어 기반의 소프트웨어 개발로 이어질 수 있는데, 이렇게 개발된 소프트웨어는 자체 실습 환경에 설치되어 다시 사용 및 관리 되어져서 자연스럽게 테스팅 과정을 거쳐서 안정화 되면서 새로운 요구 사항을 도출하는 그림 5와 같은 순환 구조를 형성하게 된다.[24] 실습 환경에서는 그림 4에서 보는 것처럼 매우 다양한 형태의 소프트웨어가 개발 될 수 있고 그 규모도 몇 줄 안 되는 작은 스크립트에서부터 팀 프로젝트로 해야 할 정도로 큰 소프트웨어까지 다양하다. 하지만 이 다양한 개발 결과물을 모두 KMU-Linux라는 자체 배포판 제작으로 연관을 지으면 소규모이거나 라이브러리 또는 Howto 문서와 같은 형태의 결과물도 다 적절한 개발 동기를 찾을 수 있게 된다.

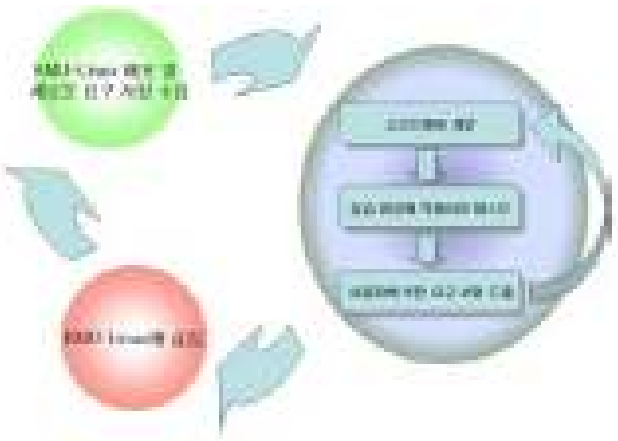


그림 5 KMU-Linux 기반의 공개소프트웨어 개발 순환 구조

8.3.2. KMU-Linux를 기반으로 한 여러 가지 소프트웨어 개발 사례

국민대학교 컴퓨터 학부에서는 지난 몇 년간 KMULinux라는 자체 배포판과 함께 여러 종류의 소프트웨어를 개발해 왔다. 아직 기간이 짧아서 그림 5와 같은 순환 구조가 두드러지게 나타나진 않지만 공개소프트웨어를 기반으로 한 실습 환경이 자체적으로 개발 및 테스트할 수 있는 여건을 잘 제공하고 있으므로 각 응용 소프트웨어의 지속적인 개발에 대한 필요성과 충분한 동기를 가지고 있다.

가. KMU-Linux 개발

KMU-Linux 개발의 경우 KMU-Linux 배포판 자체의 개발뿐만 아니라 배포판 제작 과정을 분석하고 모델링하여서 소프트웨어 도구화하는 작업인 배포판 저작도구의 개발이 함께 진행되고 있다.

1) KMU-Linux

국민대학교 공개소프트웨어 연구소의 발해 프로젝트의 일환으로 시작된 리눅스 배포판 개발 프로젝트의 성과물로서 2004년 KMU-Linux 1.0 발표 이후 현재 1.5까지 지속적으로 기능향상이 이루어지고 있는 리눅스 배포판이며 공개소프트웨어 개발의 베이스가 되고 있다. 또한 IA-64, LiveCD 등 다양한 버전의 KMU-Linux를 개발하여 실험적이고 진보된 Linux배포판을 발표하였다. 가장 큰 특징은 설치 CD를 Add On 형식으로 구성한 것인데 첫 번째 CD만을 설치하면 가장 기본적인 기능으로만 동작하고 여기에 나머지 Add On CD 중 원하는 조합을 설치하게 되어 있다. 현재의 Add On CD는 Desktop, Server, Develop 의 3종류인데 특정 주제에 맞는 Add On CD를 새로 기획해서 추가할 수 있다. 예를 들면 Game Collection 이나 특정 응용 연구자를 위한 소프트웨어 모음 등이새

로운 Add On CD로 만들어질 수 있다.

2) 커널 조정

KMU-Linux가 다양한 하드웨어 환경에서 동작하거나 제한된 환경에서 동작하기 위해서는 커널에 대한 조정이 필수적이다. 따라서 커널 옵션 및 패치 적용을 통해 KMU-Linux의 성능과 안정성을 확보할 수 있으며 궁극적으로 커널에 대한 이해도를 높이고 커널의 성능 개선을 직접 구현하고 테스트를 진행한다. 차후 커널에 대한 기본 이해가 높아진 학생이 늘어나고 좋은 여건이 조성되면 본격적인 커널 프로그래밍으로 이어갈 계획이다. 예를 들면 새로운 네트워크 프로토콜을 구현해서 해당 리눅스 박스를 실습 환경의 네트워크 장비 대신 활용하면 새로운 프로토콜을 테스트할 수 있는 좋은 기회를 얻게 되는 것이다. 이는 짜임새 있는 실습 환경이 조성되었기 때문에 가능한 일이다.

그림 6 KMU-Linux 1.0의 바탕 화면

그림 7 LOBSTER: 리눅스 배포판 저작을 위한 저작 도구

3) 배포판 저작 도구

KMU-Linux 개발 경험을 바탕으로 하여 보다 손쉽게 배포판을 개발할 수 있도록 배포판 개발 프로세스를 모델링하고 이를 바탕으로 배포판 저작 도구(LOBSTER)를 개발하고 있다.

나. KMU-Linux 기반의 응용 소프트웨어 개발

KMU-Linux를 기반으로 하는 응용 소프트웨어는 시스템 관리 프로그램에서부터 일반 응용 프로그램까지 다양하다.

1) BaMTool(Balhae Management Toolkit)

KMU-Linux의 설정을 좀 더 편리하게 하고자 개발한 것으로 리눅스 사용자들에게 현재의 리눅스 설정에 대해 좀 더 사용자 친화적인 환경을 제공하고자 한 것이다. 또한 관리 모듈을 XML로 작성하여 Plug-in 방식으로 구현하여 확장이 용이하고 설정이 간편하도록 설계된 리눅스 관리툴이다.

그림 8 BeMTool: KMU-Linux 관리 도구

2) C-Plus

KMU-Linux와 Windows와의 연동을 위한 Keberos 및 LDAP을 연동한 Identity Management Tool로서 Windows와의 상호 연계가 가능하여 Linux에서 Windows클라이언트에 대한 인증 및 정보제공이 가능 하도록 구현되었다. Keberos 와 LDAP을 Linux에 설치하여 연동시키는 내용의 Howto 문서가 작성되었는데 매우 유용하게 활용되고 있다.

그림 9 C-Plus: LDAP+Kerberos

3) AppleSeed

Bayesian Network 알고리즘을 이용하여 한글 환경에 최적화된 안티 스팸 메일 시스템을 구성하였다. 한글을 형태소 분석을 통해 처리하였기 때문에 한글로 된 이메일의 경우 공개소프트웨어인 스팸 어쌔신 보다 좋은 성능을 보여주고 있다. 차후에 스팸어쌔신과 같은 잘 알려진 공개소프트웨어와 접목하는 것을 계획하고 있다.

4) Emotion Avatar Messenger for Gaim

그림 10 Emotion Avata

메신저를 통해 메시지 전달시 메시지를 자연어 처리를 통해서 분석하여 사용자가 표현하고자 하는 감정을 알아내고 해당 감정에 따라 자신의 아바타 표정을 변화시켜 감정을 표현하는 메신저이다. 공개소프트웨어 기반 메신저인 Gaim과 통합하는 작업을 진행 중이다.

다. 모듈 및 라이브러리 개발

실습 환경에 대한 확장이 가능하도록 실습 환경과 연관된 서비스에 대

한 다양한 모듈과 라이브러리를 개발함으로써 다른 개발자의 개발을 손쉽게 할 수 있다. 대표적인 사례로서 SSO 구현을 위한 LDAP 웹 모듈을 개발하여 제공함으로써 외주에 의한 웹 서비스 개발시 인증과 관련되어 새로운 계정 시스템을 개발하거나 SSO 관련 부분을 개발할 필요가 없게 하여 실습 환경전체 구성과 일관성을 유지하기가 용이하게 하였다.

8.4. 공개소프트웨어 개발 인력 양성

서론에서 언급한 것처럼 교육에 있어서 환경만큼 중요한 것은 없다. 앞에서 기술된 공개소프트웨어 기반의 실습 환경을 기반으로 하여 각 구성원이 사용자, 관리자, 개발자의 역할을 하면서 소프트웨어 개발 교육을 받으면 그야말로 실제로 사용되면서 업그레이드되고 있는 살아있는 소프트웨어를 개발할 수 있는 기회를 갖게 된다. 이미 언급한 공개소프트웨어의 개발 사이클이 좀 더 조직적으로 순환되도록 정규 교과목, 전공 동아리 및 특강 수강자 등의 커뮤니티, 그리고 발전적으로는 산학협력 과제 등을 연계해서 공개소프트웨어 인력 양성이라는 목적을 보다 효율적으로 이룰 수 있다.

8.4.1. 정규 교과 과정과 연계

대학의 풍부한 인력으로 구성된 잠재적인 개발 커뮤니티를 활성화하기 위해서 관련 과목의 커리큘럼을 조정하고 실습 결과를 학점에 일정 부분 반영함으로써 공개소프트웨어에 대한 지식을 습득할 수 있는 기반을 조성하였다.

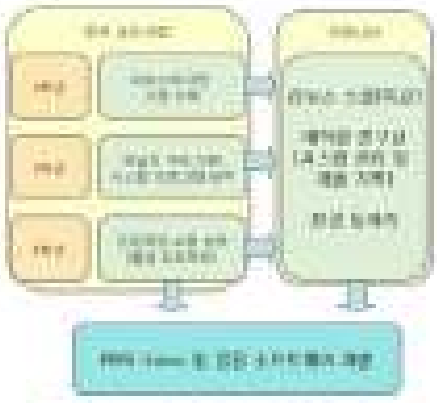


그림 11 인력 양성 구조

1) 2학년

⊙ UNIX 실습 : Linux 관련된 내용도 소개하며 실제 실습 위주의 과목으로 전환하여 Linux 프로그래밍의 기초 지식을 충분히 얻도록 한다.

⊙ 기타 과목 : 반드시 윈도우 환경이 아니어도 되는 경우에는 클라이언트에서 리눅스로 부팅해서 작업하게 한다.

2) 3학년

⊙ 운영체제: 실제 운영체제를 수정하거나 개발하는 실습 위주의 과목으로 전환하여 운영체제의 이론적 개념을 습득할 뿐만 아니라 실제적인 지식도 얻을 수 있도록 한다. 커널의 개념과 구조를 이해한다.

⊙ 리눅스 프로그래밍 : 리눅스 상에서 시스템 소프트웨어를 개발할 때 필요로 하는 기술들을 실제 실습을 통해서 습득하도록 한다.

3) 4학년

⊙ 졸업프로젝트 : 졸업 작품 제작을 목표로 하는 팀 프로젝트 과목인데 작품 주제 중 일정량은 공개소프트웨어를 개발하도록 하도록 유도한다.

이와 같은 구조를 통해 정규 교과목을 수강하는 과정에서 자연스럽게 공개소프트웨어와 친숙해지고 공개소프트웨어 개발을 시도할 수 있는 능력을 가질 수 있다. 게다가 사용하고 있는 실습 환경의 구축과 관리가 자신과 같은 구성원에 의해서 이루지고 있고, 일부 패키지 소프트웨어나 서비스들 역시 같은 소속의 구성원에 의해서 개발되었음에 자부심을 느끼며 자신도 이를 향상하는데 기여할 수 있다는 사실이 커다란 개발 동기가 될 수 있다.

8.4.2. 커뮤니티 연계

공개소프트웨어 개발 과정에서 커뮤니티의 역할은 매우 중요한 것으로 인식되어 왔다. 하지만 국내 공개소프트웨어 관련 산업이 매우 취약한 현 시점에서 활발한 커뮤니티 활동을 기대하기는 매우 어렵다. 즉 학교 외부로부터는 공개소프트웨어 관련 커뮤니티 활동을 하게할 정도의 강한 동기부여가 없는 것이다. 그리고 배포판 제작 정도의 프로젝트는 학부 수준에서 충분히 소화할 수 있는데 문제는 개발 기간이다. 정규 교과 과정의 숙제에 밀려서 학기 중에 많은 일을 하는 것은 기대하기 어려운 일인 것이다. 그래서 도입한 것이 “리눅스 스쿨”이라는 특강이었다. 즉 학기 중에는 교육을 하고 대학원 레벨에서 기획하며, 방학 중에 심도있는 개발을 하는 순환 구조를 만들려고 하였다.

가. 리눅스 스쿨

리눅스 스쿨은 리눅스에 관심있는 학생들을 대상으로 자발적인 참여를 통해 리눅스에 대한 심도있는 지식 공유를 목적으로 하고 있다. 리눅스 스쿨의 실습환경은 리눅스를 최대한 응용하고 자유롭게 실습할 수 있도록 배

려되었으며 이를 위해 관리자권한이 부여되고 다양한 디바이스를 구축하여 학생들의 관심과 부합하는 공개소프트웨어 개발을 유도하고 있다. 또한 다양한 공개소프트웨어를 접목시키고 구성할 수 있도록 교육하고 지도하여 지식뿐만 아니라 경험도 쌓게 된다. 이를 통해 자연스럽게 공개소프트웨어에 대한 관심을 유도하고 공개소프트웨어에 심취할 수 있도록 하였다. 실제로 제 리눅스 스쿨 수강생들은 작은 규모의 커뮤니티처럼 움직였다.

8.4.3. 산학협력

산학협력은 기업의 기술력과 자본, 대학의 인력과 기술력을 바탕으로 실험적인 면과 실용적인 면을 접목시킬 수 있는 좋은 기회이다. 이미 많은 산학 협력 사례가 이를 입증하고 있으며 공개소프트웨어의 경우에 는 수많은 공개소프트웨어 프로젝트가 이러한 산학협력을 통해 개발되어지고 있다. 산업계와 대학이 가지고 있는 장점을 활용할 경우 시너지 효과를 충분히 발휘할 수 있으며 이를 통해 공개소프트웨어 활성화에 기여할 수 있게 된다[22].

그림 12와 같이 대학은 테스트 인력과 초급 개발인력을 보유하고 있는 반면 산업체는 고급 개발 인력을 보유하고 있다. 또한 대학이 우수한 개발 환경과 테스트 환경이 갖춰진 반면 중소산업체는 개발 환경과 테스트 환경이 열악한 대신 고급기술과 노우하우를 가지고 있다. 이러한 산학간 장점을 극대화 시킬 경우 개발 소프트웨어의 완성도를 높일 수 있다. 실제로 KMULinux도 산학 협력 과제에 의해서 탄생하였다. 초기에 배포판 제작에 관한 노하우 등은 해당 업체의 전문가로부터 특강을 통해 전수 받았고 이를 학생들이 소화해서 계속 업그레이드 하고 있는 것이다. 이런 산학 협력에 의한 기술 이전 역시 그 전부터 존재해온 공개소프트웨어 기반의 실습 환경이 있었고 학생들이 이미 어느 정도는 그 환경에 익숙했었기 때문

에 가능한 일이었다.

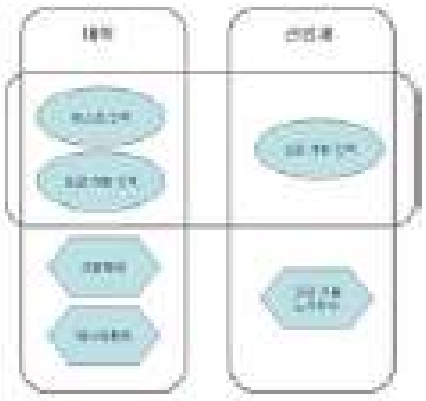


그림 12 산학 협력 구조

8.4.4. 자작 실습 환경 운영의 효과

현재 국내의 산업 현황으로 볼 때에 공개소프트웨어에 대한 관심을 가질 만한 동기를 학교 외부로부터 찾기는 어려운 것 같다. 하지만 해외의 움직임을 볼 때, 그리고 우리나라 소프트웨어 산업의 구조로 볼 때 결코 소홀히 해서는 안되는 분야인 것이다. 표 2는 국민대학교 컴퓨터학부의 지난 7년간의 졸업 작품 주제의 기반을 분류한 것이다. 표에서 보이는 것 처럼 2004년부터 Linux 기반의 작품이 2배로 늘었고 전체 과제수에 대한 점유율도 급증하였다. 실제 KMU-Linux v1.0이 2003년에 개발되기 시작되어 2004년 7월에 공개된 시점과 무관하다고 볼 수 없다. 학생들이 자신의 힘으로 리눅스 배포판을 만들고 그 것을 실제로 실습 환경에 설치해서 사용하고 또 그 위에서 응용 프로그램을 개발하는 과정에서 공개소프트웨어에 대해서 많은 관심을 갖게 되었고 자부심과 함께 개발에 직접 참여하고 싶은 동기부여가 되었다고 볼 수 있다.

표 2 종업 프로젝트 주제 분석

	Linux App	Window APP	Java App	Web App	ToTal	%
1999년	0	9	1	4	14	0.00%
2000년	3	3	2	13	21	14.29%
2001년	4	13	2	1	20	20.00%
2002년	5	12	1	0	18	27.78%
2003년	4	11	2	0	17	23.53%
2004년	8	7	2	0	17	47.06%
2005년	7	10	1	1	19	36.84%

8.5. 결 론

지금까지 공개소프트웨어를 기반으로 한 소프트웨어 교육 환경의 구축 및 이 환경에서 이루어지는 사용자와 관리자의 역할에 대해서 이야기하였고 이런 인프라위에서 같은 구성원이 개발자가 되어서 좋은 모델의 공개소프트웨어 개발 사이클을 순환시킬 수 있음을 설명하였다. 그리고 이런 개발 사이클을 잘 활용할 수 있는 인력 양성의 구조에 대해서도 이야기하였다. 이런 환경이라면 앞에서 언급한 것처럼 사용되어지는, 그래서 살아있고 개선되어질 수 있는 소프트웨어의 개발이 가능하고 이런 소프트웨어를 만드는 양질의 인력을 양성하기가 수월해지는 것이다. 교육은 적절한 환경에서부터 시작한다. 자신이 교육 받고 있는 내용이 바로 자신이 처한 환경이고 그 환경을 구성하는 요소 중에 자신들이 만들어서 기여한 부분도 있다면 거기서 얻는 교육 효과는 매우 높고 생동감이 있을 것이다.

참고문헌

[1] KMU-Linux : http://www.kmulinux.org

[2] WindowsXP : http://http://www.microsoft.com/windowsxp/default.mspx

[3] Openldap : http://www.openldap.org/

[4] ActiveDirectory : http://www.microsoft.com/

windowsserver2003/technologies/directory/activedirectory/default.mspx

[5] Camp, J. and Osorio, C., Kennedy School of Govt., Harvard, "Privacy Enhancing Technologies for Internet commerce", 2002

[6] Kerberos : http://web.mit.edu/kerberos/

[7] Microsoft, "Domain Migration Cookbook", 2000

[8] TomBialaski, "NIS to LDAP Transition: Exploring", Sun Microsystems, 2000

[9] Emmanuel Blindauer, Strasbourg "Kerberos : Linux, Windows et le SSO", JRES 2005

[10] Charlie Russel, Microsoft, "Windows Services for UNIX 3.5 White Paper", 2004

[11] Jun Futagawa, Hosei University "Integrating Network Services of Windows and UNIX for Single Sign-On", Third International Conference on Cyberworlds 2004

[12] NFS : http://nfsv4.org/

[13] Samba : http://www.samba.org/

[14] Layne Churchill, CD/CMS Fermilab "Logical Volume Manager", USCMS 2002

[15] William A. Adamson & Kendrick M. Smith, University of Michigan "Linux NFS Version 4: Implementation and Administration" OLS 2001

[16] WSUS : http://www.microsoft.com/windowsserversystem/updateservices/default.mspx

[17] POPTOP : http://www.poptop.org

[18] K. Sueyasu, T. Tabata, and K. Sakurai, "On the Security of Selinux with a Simplified Policy" Communication, Network, and Information Security 2003

[19] SSH : http://www.openssh.com

[20] Andrew Findlay, "Security with LDAP", Skills 1st 2002

[21] Joel West, "Contrasting Community Building in Sponsored and Community Founded Open Source Projects", The 38th Annual Hawaii International Conference on System Sciences

[22] 이철남, "공개소프트웨어 활성화 정책의 현황과 방향" 정보통신정책 제 15권 5호 통권 320호

[23] 송위진, "한국형 오픈소스 소프트웨어 기술개발 전략" 2002. 8

[24] 오픈소스 소프트웨어 활성화 워킹 그룹, "오픈소스 소프트웨어 연구 보고서" 2002. 12