텍(TeX)을 이용한 한글 문헌 처리

김 강 수*

1. 텍과 레이텍

텍(TeX)[1]은 미국 스탠포드 대학의 전산학자인 도널드 커누스(Donald Knuth) 교수에 의해 개발된 문서 조판 언어이다. 문서 조판 언어로서 텍은 다른 프로그래밍 언어와 마찬가지로 연산과 처리 등을 할 수 있다(이 프로그램의 이름은 〈그림 1〉과 같이 표기하여야 하지만 이렇게 하기가 어려운 경우 e를 소문자로 써서 'TeX'으로 쓴다). 다만 최종 출력이 오브젝트 코드가 아니라 "문서"라는 점이 다르다. 그래서 텍의 처리도 주로 가상의 평면에 문자와 그림 등이 들어간 박스를 배치하는 일에 집중되어 있다[2]. 텍이 취하고 있는 출력물인 문서의 포맷은 dvi(Device Independent)라는 것으로, pdf(Portable Document Format)나 ps(PostScript)와 같은 보편적인 전자출력물 형식으로 쉽게 변환할 수 있다[3]. 1980년대 초에 개발이 거의 완료된 텍은 오늘날의 발전한 프로그래밍 언어와 비교하면 한계가 뚜렷하지만, 자체적으로 매크로를 정의하고 처리하는 능력은 탁월하며, 문서 생성 프로그래밍 언어로는 유일하다시피 하기 때문에 직접적인 문서와 서적의 출판 이외에도 웹 인터페이스, 데이터 처리 등 다양한 방면에서 출력엔진으로 활용되고 있다.

〈그림 1〉 TeX Logo

TEX

한편 레이텍(LaTeX)은 레슬리에 람포트(Leslie Lamport)에 의하여 만들어진 텍 매크로 집합이다. 이 매크로의 의도는 사전에 표준적인 문서 형식을 텍을 통하여 정의해둠으로써 문서작성자가 직접 고단한 문서 프로그래밍에 마음을 빼앗기지 않고 최소한의 마크-업(mark-up)만으로 잘 정비된 문서 출력물을 얻어낼 수 있도록 하는 것이었다. 이와 더불어 마크-업을 통하여 문서의 구조화를 가능하게 한 데에도 특징이 있다. 레이텍은 대단한 성공을 거두어, 오늘날 많은 나라에서 단행본 서적, 학

* 한국텍학회 부회장(e-mail: karnes@ktug.or.kr)

술지, 프로시딩 등을 제작하는 데 있어서 실제로 활용되고 있다. 주로 복잡한 수식이 포함된 문서의 작성에 사용되었으나 현재는 대부분의 과학 분야에 있어서 표준적인 문서 작성 도구로 인정받게 되었다.

텍 및 레이텍과 관련 시스템의 활용 분야는 급격히 증가하여, 인문학 및 사회과학의 다양한 분야에서도 사용되기 시작하였으며, 음악을 포함한 예술 분야에서까지고 이용되고 있다. 또한 학술적 저술을 포함한 광범위한 분야의 상업 출판계에서도 텍과 관련 시스템을 적극적으로 활용하고 있다.

텍이 아무리 훌륭한 프로그램이라 하여도 한글을 구현할 수 없다면 우리에게는 제한적인 의미밖에 가지지 못한다. 원래부터 텍이 다국어 문자와 언어를 처리하기 위해 제작된 프로그램이 아니었으며 유니코드가 일반화되기 훨씬 이전에 개발이 거의 종료된 프로그램이었기 때문에[4] 한글의 구현은 한글 코드의 문제, 한글 폰트의 문제 등의 장애를 거쳐야 하는 과정이었다. 초창기 헌신적인 연구자들의 노력 덕분에 현재 텍과 관련 엔진에서 한글을 구현하고 처리하는 데는 어려움이 없으며 심지어 유니코드 전 영역(BMP를 넘어서는 영역)까지 한글의 식자가 가능하도록 확장되어 있다.

이 글에서는 텍과 레이텍에서 한글을 구현하고자 하는 노력과 성과를 간략히 소개하고, 한글 문헌을 텍으로 처리할 필요성에 대해서 알아본다. 특히 한글 문서의 전자화 방향과 관련하여 전자 문서 생성 처리의 기본 도구로 텍을 채택하는 데 정책적 지원의 필요성을 제기하고자 한다.

2. 텍의 한글화

1) 한글 문자의 표현

텍이 처음 만들어졌을 때는 7-비트 아스키(ASCII) 코드만을 위한 것이었다. 이것은 처음부터 영어 이외 언어의 처리에 상당한 제약이 있었음을 의미하나, 그 후 이 제약이 점진적으로 철폐되어 갔다.

처음에 나타난 변화는 유럽어를 구현하고자 하는 것이었다. 1990년대 초 EuroTeX의 노력에 의해서 복잡한 악센트 부호가 붙은 유럽 각국어 표기를 텍으로 처리할 수 있게 하기 위해 텍의 7-비트 체계를 8-비트 체계로 바꾼 것이다. 이것은 텍 자체의 발전에 의하여 이루어졌다. 동시에 폰트의 인코딩도 7-비트 OT1 인코딩에서 8-비트 T1 인코딩으로 바뀌게 되었다. 그러나 동양어, 특히 한중일 문자를 포함하는 부분은 8-비트 코드 체계로도 표현할 수 없었다. 1980년대 후반부터 90년대 초까지 이루어진 '텍의 한글화'에 관련된 작업은 이와 같은 텍의 본래적 한계를 안고 시작한 것이었다.

문제가 되었던 것은 한글 코드였다. 당시 조합형과 완성형의 논쟁이 한창이었고, 마침내 소위 완성형이 KS C 5601:1987(후의 KS X 1001) 표준 코드로 정착하였으므로 텍의 한글화에 있어서도 완성형으로 입력된 한글을 구현하는 데 바쳐졌다. 그 과정을 개략적으로 스케치하면 다음과 같다.

(1) 완성형 한글 코드가 2바이트로 이루어져 있음에 착안하여, 첫 번째 바이트를 \active[5]하게 만들고 두 번째 바이트를 첫 번째 바이트의 인자로 취하게 하여 하나의

매크로처럼 처리하게 한다.

(2) 이것을 텍 폰트의 리거처(ligature, 합자)에 대응시켜서 한글을 식자한다.

(3) 이 처리를 처음에는 "전처리" 유틸리티를 이용하도록 하다가, 나중에 텍 매크로 수준에서 처리하도록 발전하였다.

(4) 적절한 자유 폰트를 찾기가 어려웠으므로 상업용 폰트를 연구 목적으로 이용하였는데 이를 일반에 공개할 때는 저작권 제약 때문에 품위가 대단히 낮은 pk 비트맵 폰트만을 제공하거나, 이에 불만족하여 새로운 한글 폰트를 처음부터 디자인하는 (HLaTeX) 어려움을 겪었다. 폰트 문제는 지금도 매우 치명적인 문제가 되어 있으나, 이러한 노력의 결과 "은 글꼴"이라는 자유 소프트웨어 라이선스(GNU GPL)의 폰트 한 벌을 가지게 된 것은 특기해야 할 사항이다[6]. 은 글꼴을 확보함으로써 높은 품질의 pdf 제작과 같은 텍의 한글화가 급격히 발전한 것은 다른 나라의 텍 사용자들로부터 부러움을 사고 있는 사실이기도 하다.

1990년대 중반에 오면 완성형 한글을 식자하는 한글 텍 시스템이 대략 그 모습을 드러내게 된다. 차재춘의 hLaTeXp, 은광희의 HLaTeX이 이들이며, 일부에서는 이를 상용화하려는 시도도 있었다[7].

한편, 한글 윈도 운영체계가 CP949라는 확장완성형 코드를 취하게 되면서 텍으로 표현하지 못하는 문자가 문제시되게 되었다.

예컨대 "똠"이나 "뙿"과 같은 글자는 실용적으로 사용될 가능성이 크지 않다고 하나 아무래도 표현의 제약이란 그다지 바람직한 것이 아니었다. 그래서 텍의 한글화를 주도한 개발자들은 유니코드에 주목하게 된다.

텍 자체에서도 유니코드를 처리하는 능력에 대한 고민이 시작되었다. 그 결과가 Omega 프로젝트인데, 이것은 8비트로 디자인된 텍의 언어 코드 자체를 16비트로 확장하여 텍이 내부적으로 BMP[8]에 해당하는 유니코드 문자를 다룰 수 있는 능력을 부여하려는 것이었다. 만약 이 프로젝트가 성공하여 텍을 대체하는 정도로까지 갔다면 지금과는 다른 결과를 얻을 수 있었을지도 모른다. 오메가는 상상 이상으로 복잡한 문제를 야기하였고 텍이 주로 사용되는 서구에서 그다지 흥미를 보이지 않은데다가 유니코드 자체가 2바이트에 제한되지 않고 확장되어 있는 등의 이유가 작용한 것인지, 현재는 개발이 사실상 중단되어 있다. 일반적 목적에 사용하지 못할 정도는 아니지만, 오메가에 의존하지 않고도 레이텍에 다중 바이트 문자 코드를 처리하는 방법이 개발됨으로써 그 실익이 줄어들기도 하였다. 오메가에서 한글 문자의 처리는 비교적 간단하였다. 2000년대 초반에 오메가에 의한 한글 문자 처리에 상당한 진척이 이루어졌으나 이제는 실험적인 시도로만 남게 되었다.

레이텍 프로젝트의 일부인 babel 패키지에서는 utf-8 입력된 텍스트를 처리하는 방법을 부분적으로 제시하였고 이것은 곧 레이텍의 일부가 되었다. 이와는 별도로 Dominique Unruh는 unicode 패키지[9]를 개발함으로써 utf-8 인코딩[10]의 유니코드 문자를 레이텍 자체로 모두 처리할 수 있게 되었다. 이와 더불어 unicode 패키지의 한글처리방법을 원용하여 이를 한글 폰트와 대응시키고 한글 문서에 알맞은 여러 가지 기능을 갖춘 Hangul-ucs 패키지[11]

가 2004년에 개발되어 현재 한글 사용의 새로운 표준으로 자리잡아가고 있다. 현재의 Hangul-ucs는 D. Unruh의 unicode 패키지에 대한 초창기의 의존성을 제거하여 좀 더 레이텍 표준에 가까운 방식으로 동작한다[12]. 이에 이르러 한글 문자를 텍과 레이텍에서 식자하는 문제 자체는 거의 완전히 해결되었다고 평가할 수 있다.

2) 한글 문서의 타이포그래피

타이포그래피란 하트만과 제임스에 의하면 '텍스트를 인쇄된 페이지들의 형태로 나타내는 것과 관련된 실용적인 분야'를 말한다[13]. 즉, 타이포그래피란 책의 판형, 판면 배치, 글꼴의 종류와 크기 등 인쇄물의 형태를 결정짓는 모든 절차와 과정이다. 인쇄물 또는 출력물은 그 텍스트에 담긴 정보와 더불어, 그 정보를 전달하는 매체인 시각적 요소로 이루어지는데, 이 가운데 정보 전달의 시각적 매체의 특성을 총칭하여 넓은 의미에서 타이포그래피라 하기도 한다.

텍과 레이텍이 문서 작성 언어 또는 시스템인 점을 감안하면 타이포그래피로부터 자유로울 수 없다. 초창기에는 한글 코드를 성공적으로 "찍는" 데 주력하였다면 한글 레이텍의 발전을 위해서는 타이포그래피에 대한 진지한 고려가 필수적이었다. 이를 위해 타이포그래피 디자인의 경향과 활판 인쇄 시절의 타이포그래피 규칙들을 연구하여 다양한 방식의 한글 레이텍 매크로로 구현하는 노력이 경주되었다.

타이포그래피의 기본이 되는 폰트는 GPL 글꼴인 은 글꼴이 채택되었으나 곧 다른 상업용 폰트를 자유롭게 사용할 수 있는 기술이 발전하였다[14]. 폰트의 특성에 대한 연구가 시작되고 이를 통하여 한글 문서에 적절한 배치 요소, 자간, 행간, 문장부호의 처리, 문단의 배치 등에 대한 연구와 시도가 시작되었다. 디자이너가 자신의 의도를 더 쉽게 구현할 수 있도록 레이텍 스타일을 지원하는 작업이 이루어졌다. 이 성과들은, 한글 텍의 관심을 문자의 식자에서 문서 및 문헌 조판의 실제에 적용하기 위한 것들로, 한글 텍 시스템의 일부를 이루게 되었다. 텍 자체가 주로 과학 문헌에 사용되었기 때문에 이에 적합한 요소들, 예컨대 영문자와의 조화, 한글에서의 이탤릭 및 강조 방법의 문제, 한글 문장부호와 ASCII 문장부호의 일치 등의 문제가 집중적으로 관심의 대상이 되었는데, 그 성과의 일부를 ko.TeX에서 찾아볼 수 있다[15].

3) 실제 출판에의 응용

한글 레이텍의 기술적 발전과 더불어 레이텍을 이용하는 서적의 출판도 이루어졌다. 주로 수학 및 경제학 서적이 일부 출판사에 의하여 시도된 것인데, 레이텍의 대중화의 한계로 인하여 충분한 시스템이 갖추어졌다고 보기에는 이르지만 적어도 수식 등이 많은 서적에 관한 한 나쁘지 않은 성과를 얻게 되었다. 실제 현장의 출판 경험을 가진 한 보고자의 경험에 따르면 한글 레이텍의 발전에 따라 레이텍 출판된 서적의 질도 함께 향상되어 왔다고 한다[16].

3. 전자 텍스트와 텍

1) 전자 텍스트와 마크업 문서

정보 기술이 발달하기 시작하면서 손으로 쓰였거나 인쇄된 종이 텍스트 이외에 방대한 양의 '전자 텍스트' (electronic text)가 생산되기 시

작하였다. 또한 인터넷의 발달은 전자 텍스트의 폭발적인 증가를 불러왔다. 전자 텍스트란 컴퓨터에서 읽고 쓸 수 있는 기록 매체에 저장된 텍스트를 말하는 것이다. 전자 텍스트의 초기 용도는 앞서 설명한 전산 조판 및 탁상 출판의 재료였다. 그러나 오늘날 전자 텍스트는 텍스트에 담긴 정보의 추출, 검색, 요약, 가공 등의 절차를 통해 다양한 분야에서 다양한 용도 활용되고 있다. 언어학 쪽에서는 말뭉치 언어학이나 디지털 인문학 쪽에서 전자 텍스트를 직접적으로 취급하고 있으며, 다른 학문 분야에서도 전자 텍스트 활용성이 이에 못지않다.

2) 상용 워드프로세서 파일의 전자문서로서의 문제점

전자 텍스트는 다양한 전산 도구로 처리되어야 하며, 다양한 플랫폼에서 활용되어야 하므로 이식성(portability)이 높은 형식으로 작성되어야 한다.

가장 이식성이 높으며 중립적인 전자 텍스트의 형식은 순수한 문자열 텍스트 이외의 정보는 일체 포함하지 않은 '날 텍스트'(raw text)이다[17]. 그러나 텍스트에는 조판 명령은 아니더라도 다른 정보가 부가되어야 할 필요가 있는 경우가 많다. 장, 절, 단락 구분 등의 책의 논리적 구조는 그러한 부가 정보가 대표적인 예이며, 데이터베이스 형식의 전자 텍스트라면 담긴 자료의 구조 역시 부가되지 않으면 그 전자 텍스트의 가치가 별로 없을 수도 있다. 즉, 현대에 요구되는 전자 텍스트는 '구조화 문서'(structured document)인 것이다. 구조화 문서를 작성하기 위해서는 날 텍스트에 더하여 구조적 정보를 부가하여야 하는데, 이 부가 작업을 마크업(Mark-up)이라고 부른다. 마크업 역시 여러 형식으로 이루어질 수 있는데, 이 또한 표준화가 필요하다. 이를 위하여 1970년대에 IBM에서 개발된 전자 텍스트 형식이 SGML(Standard Generalized Mark-up Language)이다. SGML은 일종의 메타언어로써 전자 텍스트에 각종 정보를 부가할 수 있는 언어를 정의할 수 있게 해 준다. 인터넷에서 널리 쓰이는 HTML(Hypertext Mark-up Language)은 SGML 텍스트의 한 예이다[18].

SGML은 ISO에 의해 국제 표준으로 정해졌으나 SGML에 의해 마크업이 이루어진 문서의 형식적 구조가 너무 복잡하여 실질적으로 응용되기에는 무리가 있었다. 이에 따라 1980년대 후반부터 SGML을 간략화하기 위한 연구 진행되어 1990년대 후반에 이르러 SGML의 부분집합이면서 훨씬 간략한 문법을 지닌 XML(eXtensible Mark-up Language)이 등장하였다. XML은 폭발적으로 그 수요가 늘고 있으며, 인터넷 상의 문서들도 급속히 XML로 전환되고 있다[19].

이 점에서 볼 때 워드 프로세서 등 상업용 프로그램의 자료 파일 형식이 갖는 문제점은 다음과 같다.

(1) 텍스트와 조판 지시의 분리 불가능

본질적으로 전자 텍스트는 내용(contents)을 구성하는 텍스트와 이 텍스트를 조판하기 위해 사용하는 지시로 분리 가능해야 한다. 그러나 일반적인 워드 프로세서 문서는 이러한 분리가 불가능하거나 매우 어려운데 그 이유는 워드 프로세서의 원래 목적이 텍스트를 전자적으로 처리하는 것이라기보다는 출판에 적합한 최종 판

면을 만들어내는 데 있기 때문이다. 이 문제를 처리하기 위한 근본적인 방안은 텍스트를 되도록 건드리지 않으면서 언제라도 제거하거나 치환할 수 있는 조판 및 주석 표지(mark)를 붙여두는 것이 될 것이다. 마크업 언어들이 개발된 것은 바로 이런 필요성 때문이었다.

(2) 이진 파일 형식의 폐쇄성과 비이식성

워드 프로세서 파일은 대부분 이진 파일 형식으로 인코드되어 있어서 직접 그 내용을 볼 수 없고 반드시 해당 워드 프로세서를 통하거나 뷰어를 통해야만 가능하다. 이것은 외부의 다른 프로그램이나 도구로써 텍스트를 조작하거나 변형하는 것을 힘들게(불가능하게) 만드는 것이며, 상업용 프로그램으로서는 당연히 요청되는 기능이라 할 수 있다. 그러나 텍스트의 공유와 연구를 위해서는 텍스트의 개방성과 이식성이 확보되어야 한다. 이 점에서 워드 프로세서 파일 형식은 적절하지 못하다[20]. 워드 프로세서 파일로 데이터베이스를 구축해보아야 기껏 할 수 있는 것은 텍스트 자체에 대한 검색이나 접근이 아니라 제목과 저자 등 외부 정보(meta data)에 대한 것에 그칠 것이다.

(3) 상업 프로그램에 대한 의존성의 문제

상업 프로그램은 제작사의 자산이다. 그런데 우리는 특정인의 사적 소유물이 아니라 사회 전체의 공유물로서의 전자 텍스트에 주목하고자 한다. 상업 프로그램을 이용하여 전자 텍스트를 제작하거나 열람하면 안 된다고 말할 것은 없겠지만 그 이외에 다른 대안이 없다는 것은 문제가 된다. 원칙적으로 공유된 텍스트는 누구라도 보고 참고할 수 있도록 제공되어야만 할 것이라고 믿는다. 만약 일종의 공유재산인 문화적 유산으로서의 한글 텍스트들이 특정한 기업의 상업용 판매물에 의존해야만 보거나 처리할 수 있다면 이것은 치명적인 손실이 아니라고 할 수 없다. 즉 콘텐츠는 공유자산인데 상업용 프로그램에 의하지 않고는 그 콘텐츠에 접근할 수 없게 한다면 이것은 당연한 권리에 대한 제한이 되는 것이다. 그 대표적인 예가 흔글과 같은 워드 프로세서이다. 흔글은 매우 탁월한 프로그램이지만 그 문서 형식이 개방적이 아니므로 이 프로그램에만 전적으로 의존하였을 때 향후 특정한 문서를 읽지 못하게 되는 경우도 없으란 법이 없다.

(4) 구조화의 취약성

오늘날 대부분의 워드 프로세서들은 문서를 구조화할 수 있는 도구를 제공한다. 그러나 여전히 그 사용의 유인은 낮은 편이며, 장절 정보와 같은 문서 구조 정보를 포함하지 않고 다만 시각적 치환만으로 이를 처리하는 예를 왕왕 볼 수 있다. 구조화되지 않은 문서를 전자적으로 처리하는 것이 매우 어려워질 가능성이 큰 까닭에 워드 프로세서 포맷의 파일을 그대로 사용하는 것은 이러한 위험에 노출될 가능성을 보여주는 것이다.

2) 마크업 언어 처리도구로서의 텍의 가능성

마크업 텍스트는 텍스트 자체는 날 텍스트와 동일한 인코딩으로 만들어지므로 그 내용에 쉽게 접근하고 조작할 수 있으며, 텍스트 부분과 지시어(mark-up) 부분이 분리되어 처리되므로 조판 지시 등의 문서 구조화 요소와 컨트롤을 텍스트 손상 없이 취급할 수 있다. 레이텍 문서

도 본질적으로 마크업 문서와 그 개념이나 형식이 유사하다. 레이텍의 경우에는 텍스트 자체와 텍스트를 조판하는 지시어(mark-up)가 함께 쓰이지만 이를 분리하여 다룰 수 있는 것이다[21].

텍의 급속한 확산에는 그 출력물의 품질이 우수한 이유도 있지만, 더욱 중요한 것은 플레인 텍스트를 기반 형식으로 하는 특성으로 인해 방대하고 복잡한 자료를 다루는 분야에서 이용되는 여러 자료 처리 도구와의 결합이 자유롭다는 점일 것이다[22]. 마크업 언어 처리 도구인 텍은 아름다운 문서를 작성할 수 있으며 개방성과 확장성을 갖춘 자유소프트웨어이다.

텍의 개방성은 전자 텍스트 처리에 중요한 가능성을 제시해 준다. 대표적인 자유 및 공개 소프트웨어 가운데 하나인 텍은 그 내부 구조가 공개되어 있으며, 다양한 모듈들을 사용자의 요구에 맞게 조절할 수 있는 유연성을 지니고 있다. 즉, XML 형식으로 마크업된 텍스트를 처리함에 있어서도 텍을 "출력 엔진"으로 사용할 수 있다는 것이다. 실제로 TeXML이나 XML stylesheet 형식으로 텍을 출력 엔진으로 활용하는 전자 텍스트 작성도구들이 개발되어 있다.

텍의 확장성은 주목할 만한 특성이다. 텍은 다른 스크립트 언어와 결합하여 함께 동작하게 하거나 다른 언어를 텍 콘트롤 시퀀스 정의에 활용할 수 있다. 현재 lua 언어와 텍을 연계한 luaTeX이 개발 중이며[23], perl 언어를 텍 매크로 정의에 활용할 수 있도록 하는 perlTeX[24]이나 데이터베이스를 직접 조작할 수 있게 하는 sqlTeX이 시험 중이거나 실용화되어 있는 상태이다. 전자 텍스트의 양이 방대하고 이에 대하여 검색 치환 등의 조작을 가하려면 필수적으로 데이터베이스나 스크립트 언어를 적용하지 않을 수 없는데 이것을 더 쉽게 가능하게 해준다[25].

텍이 자유 소프트웨어라는 점은 중요한 장점이다. 텍의 소스는 완전히 표준 코드를 사용하는 플레인 텍스트로서 어떤 상황에서도 이를 처리하지 못할 가능성이 없는 것이라는 점이 텍과 같은 자유 소프트웨어를 사용해야 할 필요성을 제기한다 하겠다.

텍의 강력한 문서 작성 능력은 당연히 고려되어야 한다. 처리된 전자 텍스트를 일반이 읽을 수 있게 하려면 고도의 타이포그래피가 적용된 아름다운 출력물을 생산할 필요가 상당하다. 한글 텍에서 발전한 많은 기술이 이것을 가능하게 해준다[26].

4. 21세기 세종계획의 예: 텍의 활용 가능성[27]

21세기 세종 계획은 우리말과 우리글을 바탕으로 하는 정보 사회 건설을 위하여 세계 수준의 국어 기초 언어 자료 기반 및 표준화된 전자사전 등의 대규모 국어 자료 구축, 한민족 언어 정보화 및, 전문 용어 정비 등을 내용으로 하는 국어 정보 기반 구축 사업과 비표준 문자 등록, 국어 정보화 인력 양성 등을 내용으로 하는 국어 정보화 여건 조성으로 구성되어 10년간의 발전 계획을 가지고 1998년부터 시작된 국가 주도의 중장기 국어 정보화 사업이다.

21세기 세종 계획을 통하여 구축된 국어 기초 자료의 핵심은 대규모의 국어 자료인 말뭉치'로서 현대 국어는 물론, 훈민정음 창제 이후의 중요 문헌들의 전자화되었다. 말뭉치에는 원문

의 내용만 전자화된 원시 말뭉치 이외에 형태론적 정보와 통사론적 정보 등의 언어 정보가 부가된 주석 말뭉치가 있다. 이들 말뭉치는 전자 텍스트의 한 형태로 컴퓨터를 이용한 분석과 처리를 전제로 구축된 것이다.

그러나 현재의 세종말뭉치는 이 전제를 만족시키기에는 부적합한 부분이 있다. 첫 번째 문제는 현재 채택된 전자 텍스트 구조화 방식의 한계이다. 세종말뭉치의 구조화에는 TEI(Text Encoding Initiative)에서 제정한 SGML 기반의 말뭉치 구조화 규격을 국어 말뭉치에 맞게 확장 및 변경하여 사용하고 있다. SGML은 표현력은 우수하지만 전산 처리 등의 부담이 커서 최근에는 XML로 대부분 대체되었으며, 말뭉치의 구조화도 그 예외는 아니다. TEI에서도 최근 XML 기반의 말뭉치 구조화 규격을 발표한 바 있다. 따라서 현재의 SGML 기반 마크업은 XML 기반으로 변환하여야 할 것이다. 또한 현재의 마크업은 말뭉치의 원본 문헌 정보 등의 메타 정보와 원본 문헌의 논리적 구조(문단 구분 등)의 표시에 사용되고 있을 뿐, 주석 말뭉치의 경우에는 XML 문법에 따른 마크업이 되어 있지 않다. 이는 정보 표현의 일관성 문제뿐만 아니라 말뭉치의 적극적인 이용에도 장애 요소로 작용할 수 있으므로마크업의 확장이 시급히 요청된다.

두 번째 문제는 더욱 심각한 문제이다. 현재 세종말뭉치는 상용 워드프로세서인 흔 글 파일 형식으로 구축되어 보급되고 있다. 상용 워드프로세서가 말뭉치 구축 도구로 사용되었다 하더라도 보급되는 말뭉치는 컴퓨터의 하드웨어 및 소프트웨어 플랫폼에 구애받지 않는 중립적인 '날 텍스트' 형태로 보급되는 것이 지극히 당연한 것임은 길게 논의하지 않아도 될 일이다. 날 텍스트 형식으로 보급될 경우에도 국제 및 국내 표준인 유니코드 인코딩을 사용해야 할 것이다. 이는 옛한글 말뭉치의 경우 더욱 중요한 문제이다. 흔글에 사용된 옛한글 코드는 유니코드 표준에 맞지 않는 사용자 영역 코드로서, 흔글에서 말뭉치를 유니코드 텍스트로 저장한다 하더라도 여전히 비표준 코드로 저장된다. 비표준 코드로 표현된 옛한글 음절은 자소 분리 등의 처리가 불가능하기 때문에 제대로 된 텍스트 처리를 위해서는 유니코드 표준으로 변환하는 과정을 거쳐야만 한다.

요약하면, 세종말뭉치의 전산적 분석, 가공, 인쇄 등의 폭넓은 활용을 위해서는 하드웨어 및 소프트웨어 중립적인 표준 텍스트 형식이 필수적이며, 부가 정보의 표현력을 높이기 위한 적극적인 마크업이 요구된다.

우리는 그 대안으로 텍을 이용한 문헌 처리 방법을 제시하려 한다. 설령 흔글로 작성되고 처리된 텍스트라 하더라도 이를 가공 및 이용이 가능한 형태로 변환하기만 하면 텍의 활용 가능성은 훨씬 커질 것이다.

5. 텍을 이용한 한국어 문헌 처리의 제 문제

1) 입력 코드의 표준 확립

현재 한국어 전자 텍스트의 활용성을 높이기 위해서는 입력 코드의 통일이 시급한 문제라고 생각한다. 한글 코드는 이론의 여지가 없이 유니코드가 되어야 하겠지만[28], 비록 유니코드를 다룰 수 있다고는 해도 윈도 운영체제의 사용자 인터페이스가 완성형에 기초한 확장완성형이라

는 엄연한 사실은 특별한 주의를 요한다. 만약 텍을 이용하는 전자 문서 처리 시스템을 개발한다면 이 점을 충분히 반영하여 사용자 인터페이스를 구성해야 할 것이다. 반면 리눅스나 맥과 같은 다른 시스템은 이미 유니코드 기반의 운영체제로 이행한 지 오래다.

또한, 옛한글의 입력 코드 문제는 여전히 미해결 문제로 남아 있다. 현재 훈글의 옛한글 처리 방식은 이른바 "폰트 의존적"[29] 방법으로서, 한컴바탕, 새 바탕과 같은 옛한글을 표현할 수 있는 글꼴의 사용자 영역(PUA)에 엄선된 5299자의 옛한글 문자를 완성형으로 넣어두고 그 코드를 불러쓰는 방식이다[30]. 이것은 유니코드 표준에도 어긋나며 특정 기업의 저작물인 특정 코드에만 의존해야 옛한글을 처리할 수 있다는 것을 의미한다[31].

옛한글을 표현하는 국제 표준은 유니코드 [U+1100]부터 배정된 한글 자모를 조합하여 표현하는 이른바 "첫가끝" 방식이다. 초성 90자, 중성 66자, 종성 82자와 2자의 채움 문자로 이루어지는 이 코드체계는 모든 자모에 코드를 부여하고 완성된 음절 하나를 일련의 자모 코드의 연속으로 표현하는 조합형인데, 초성, 중성, 종성이 모두 다른 코드 영역으로 되어 있으므로 한글 표현의 한계가 거의 없다. 이 방식은 텍스트의 코드량이 늘어난다는 단점을 제외하면 한글을 가장 잘 표현할 수 있는 코드체계라고 생각한다. 이 코드를 충분히 잘 다룰 수 있는 부가도구(에디터 등)가 많지 않은 현실을 생각하더라도, 향후 텍스트 정보의 손실 없이 유지할 수 있는 거의 유일한 코드 체계가 아닌가 한다.

이와 관련하여, 겹치는 코드의 우선순위 문제를 생각해야 한다. 예컨대 "각"이라는 글자는 한글음절영역의 [U+AC01]로도 표현할 수 있지만, 첫가끝 방식 [U+1100] [U+1161] [U+11A8]의 코드열로도 표현할 수 있는 것이다. 음절영역 코드를 쓰면 2바이트 표현이 가능하다. 첫가끝 방식의 경우에는 그 세 배에 달하는 코드량을 가지기는 하지만(UTF-8로 인코딩하면 바이트열로는 한글 한 글자를 표현하는 데 9바이트가 소요된다) 이것이 치명적인 문제가 된다고 보기는 어렵다. Mac OS X 운영체제가 한글 처리에 첫가끝 유니코드를 사용하고 있음을 생각건대 전산처리에 있어 바이트량 때문에 무리가 간다고 볼 수는 없는 것이다. 이 두 표현 방법 중에 어느 것을 우선할 것인가? 두 가지 방법을 다음과 같이 정리하자.

방법 1. 현대 한글 음절 영역에 정의된 문자는 최대한 음절 영역 코드로 치환하여 처리한다.

방법 2. 전체를 첫가끝 코드로 처리한다. 음절영역 코드로 입력된 것도 첫가끝으로 치환한다.

앞서 언급한 바이트량이 문제라면 방법 1이 더 나아 보인다. 그러나 코드의 일관성이라는 측면에서는 방법 2가 더 나은 방법일 것이다. 이와 관련하여 고려대학교 민족문화연구원 문자코드연구센터에서 제시한 시안[32]은 방법 1을 원칙적으로 따르게 하고 있는데, 이것은 마이크로소프트 워드 2002의 방식과 유사하다. 그러나 이 방법을 따르면 혼선을 피할 길이 없다고 본다.

방법 2를 채택하더라도 문제가 되는 것이 두 가지가 있다. 유니코드 한글 자모 영역에 정의

되어 있는 "복합자모"의 코드화와 관련된 문제이다. 이에 대응하는 두 가지 방안을 다음과 같이 정리할 수 있다.

방법 2-1. 복합자모가 정의되어 있는 경우는 복합자모 코드를 우선적으로 하고 복합자모가 정의되어 있지 않은 "U+YEO"와 같은 경우는 단자모 조합으로 처리하는 방안.

방법 2-2. 복합자모 정의를 사실상 무시하고 한글 단자모 정의만을 채택하여 복합자모 코드로 입력되어 있더라도 이를 단자모 조합으로 바꾸어 처리하는 방안.

방법 2-1이 문자코드연구센터가 제시하는 안이지만 현재의 복합자모 정의가 완전하지 않은 데에다 앞으로 어떤 문자가 더 발견될는지 알 수 없는 상황에서 바람직한 방안이라고 보기 어렵다. 방법 2-2는 코드열의 양이 부담이 되기는 하지만 이를 문제 삼지 않는다면 문제의 소지를 최소화한 가장 깔끔한 방안이다. 방법 2-2의 방식을 취한다면 추가적인 한글 복합자모를 유니코드에 새로운 영역으로 할당받으려는 시도도 불필요할 것이며 복합자모 정의 자체가 그다지 필요하지 않게 될 것이다. 또한, 예컨대, 옛한글 텍스트 검색을 한다고 가정해보자. 이 때는 일단 복합자모나 완성형 음절 문자를 사용하는 텍스트라 하더라도 이를 전부 일관된 코드 체계인 단자모 조합으로 변환한 다음, 검색 문자열 자체도 같은 코드 형식으로 통일하고 나서 시행해야 제대로 된 결과를 얻을 수 있다. 서로 다른 코드 체계를 하나의 문서에 섞어 넣음으로써 검색과 치환 등에 오히려 부담을 가중하게 하는 것은 바람직한 방안이라고 할 수 없다.

한글 텍(ko.TeX)에서는 두 경우 모두 원칙적으로 두 번째 방안 즉 방법 2와 2-2의 방안을 표준으로 고려하고 있으며, 방법 1이나 2-1, 그 밖의 방식으로 입력된 텍스트를 변환할 도구를 마련하고 있다.

2) 마크업의 표준

한글 전자 텍스트는 XML을 이용할 가능성이 가장 많고 그것이 현실적이다. 한글 전자 텍스트의 마크업 표준이 마련되어 공개된다면 이를 바탕으로 한 여러 변형 및 출력 도구들이 개발될 수 있을 것이다. 이를 위하여 마크업에 포함되어야 할 기초 정보를 선정하는 작업이 선행되어야 할 것이다.

여러 기관과 연구자들이 다양한 방식으로 설정한 마크업 방식들을 비교 종합하여 표준을 확립할 필요성이 크다고 하겠다. 구축된 전자 텍스트는 실용의 목적뿐 아니라 각 학문 분야의 연구 소재로도 활용될 것이므로, 사용되는 수준과 층위에 따라 필요한 정보 요소가 달라질 것임도 고려해야 한다.

3) 편집, 출력, 조판 도구

전자 문서의 전자적 출력은 pdf를 이용하는 것이 일반적이다. pdf는 Adobe사의 전자문서 표준 포맷이다[33]. 어떤 플랫폼에서도 읽을 수 있고 Adobe사에서 제공하는 뷰어가 무료이며 그 외에도 다양한 오픈소스 뷰어들이 많이 제공되고 있으므로 pdf 형식의 전자 문서에 대한 접근 가능성은 매우 높은 편이다. 또한 화면상의 시각적 형태 그대로 출력이 가능하며 텍스트에 대한 검색도 가능하다는 등의 장점으로 인해 사

실상의 표준으로 자리 잡았다. 텍 역시 pdf 출력을 기본적으로 지원한다. dvi의 pdf 변환 도구인 dvipdfmx와 같은 드라이버 유틸리티, 소스를 직접 pdf로 제작하는 pdfTeX 등의 발달은 텍을 매우 pdf 친화적인 프로그램으로 만들어 놓은 것이다.

pdf의 바탕이 되는 PostScript는 페이지 기술 언어(page description language)이다. 마크업 언어가 텍스트와 분리되는 문서 구조 등을 추상적으로 기술하는 것인데 비해, 페이지 기술 언어는 특정의 오브젝트(박스, 인쇄요소, 그림 등)를 해당 페이지에 출력하는 방식에 대해 기술하는 인쇄 전문 언어이다. pdf는 폰트 정보와 페이지 상의 위치 정보, 그리고 텍스트 정보를 가지고 있으므로 텍스트만을 별도로 추출하거나 검색하는 것이 가능하다는 중요한 장점이 있다[34].

pdf는 적어도 다음과 같은 요소를 갖추어야 한다.

(1) 텍스트의 검색과 추출이 가능해야 한다.
(2) 인쇄본을 얻을 수 있을 정도의 품위를 갖추어야 한다.

pdf 출력본을 만들 때 사용되는 폰트의 문제가 있다. 흔글을 이용하는 경우[35] 상업용 폰트인 한컴바탕을 내장(embed)하게 되는데, 이것은 흔글의 이용권 자체와는 별도로 라이선스 문제를 야기할 수 있다[36]. 즉, 원칙적으로 최종출력물에도 폰트의 문제가 발생하는 것이다. 이를 회피하기 위해서는 은 글꼴과 같은 자유 라이선스의 폰트를 사용할 수밖에 없다. 김도현(2007)에 의하면, GPL 라이선스인 '은 글꼴'을 pdf에 임베드하더라도 그것이 텍스트와 콘텐츠 자체에까지 저작권의 감염성이 미치지는 않는다.

한글 글꼴은 한글 운영체제에서만 탑재되어 있을 것이므로 이를 임베드하지 않고 배포하는 것은 다른 시스템에서 해당 문서를 읽을 수 없게 되는 등 약간의 호환성 문제를 야기할 수 있을 것이다.

한글 텍스트의 검색과 추출은 pdf로부터 정보를 얻어내기 위해서도 필요하고 시각장애인이 pdf에 접근할 수 있게 하기 위해서도 필요하다. pdf를 읽어주는 프로그램이 동작하려면 텍스트 추출이 불가능해서는 곤란할 것이다. 일부 흔글로부터 작성된 pdf가 텍스트 검색 추출이 불가능한 경우가 많다.

이러한 조건을 갖춘 양질의 pdf 출력물을 얻는 가장 쉬운 방법은 한글 텍 시스템을 이용하는 것이다. 적어도 현대 한글 텍스트를 처리하는 데 폰트 등의 문제는 없다. 옛한글 폰트가 아직 자유 소프트웨어 라이선스로 배포되는 것이 없기 때문에 한국텍학회를 중심으로 이 분야의 해결을 위한 연구방안을 모색하고 있다.

6. 텍을 이용한 한글텍스트 처리의 실례

이 절에서는 21세기 세종계획 웹사이트에 공개된 옛한글이 포함된 문서를 레이텍을 통하여 가공하고 조판하여 pdf 전자문서를 얻는 과정을 보이려 한다[37]. 이 시험은 한국텍학회와 한글텍사용자그룹(KTUG)[38]에서 제공하는 KTUG Collection 배포판을 이용하였다.

먼저 21세기 세종계획 사이트[39]에 접속하여 자료실에서 역사자료 중 초간 두시언해 권19 자료를 다운로드하였다. 이것은 hwp 확장자를 가

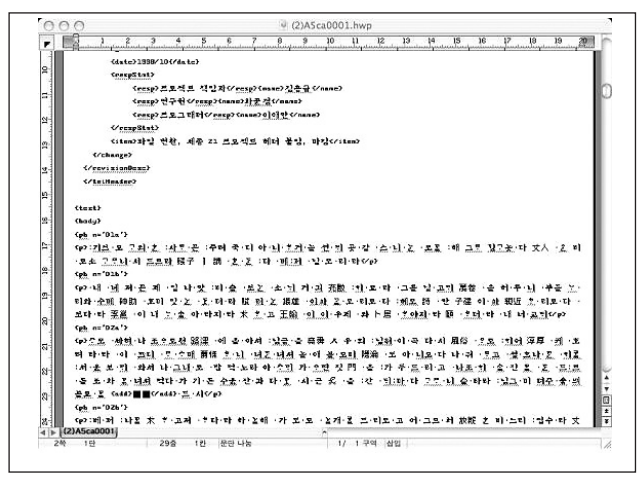
진 훈글 문서 파일이지만 이것을 해당 워드 프로세서로 열어보면 〈그림 2〉와 같이 훈글 이진 문서 포맷의 내용이 TEI 태그를 붙인 SGML mark-up 문서 형식임을 알 수 있다. 이것은 매우 실망스러운데 그 이유는 mark-up 형식의 문서라고 해야 할지 아니면 hwp 형식의 문서라고 해야 할지 잘 판단이 되지 않기 때문이다. 21세기 세종계획 사이트에는 이 태그를 없애는 도스 유틸리티(오래된 도스 프로그램이라서인지 Windows XP에서 잘 실행되지 않을 때가 많다)를 제공하고 있으나, 어차피 그럴 것이면 왜 유니코드 인코딩 텍스트 파일로 제작하지 않았는가 하는 문제를 제기할 수 있다.

또한 여기에는 한 글자의 결자(〈add〉태그가 있는 곳)가 있는데 이것은 아마도 훈글의 완성형 사용자 영역 옛한글 문자이거나 UCS2 영역에서 찾을 수 없는 한자일 가능성이 높다. 옛한글 문자라면 "첫가끝" 코드로는 입력이 가능하였을 것이나 훈글이 이 코드를 지원하지 않기 때문에 표현이 불가능해진 것일터이고, 한자라면 UCS4의 확장 한자(Ext-B) 영역까지 고려하면 아마도 찾아볼 가능성이 있을지 모른다. 그러나 우리 고문헌의 특성상 그래도 표현할 수 없는 글자도 있을 수는 있는데[40] 그것은 고문헌을 처리하면서 발견하여 이를 보고하고 확장된 유니코드 체계에 반영하도록 노력하여야 할 것이다. 옛한글 문자의 경우라면 한마디로 비표준 옛한글 코드 체계를 선택한 대가라고 하지 않을 수 없다. 이 한 글자는 향후 추가 문자 목록을 참고하지 않으면 아주 결락되어 활용이 불가능해질 수도 있는 것이다.

이 문서가 채용하고 있는 마크업 방식(SGML)은 보다 범용성이 높은 XML 형식으로 바뀌는 것이 바람직하다고 판단된다. 그러면 이 자료들에 대한 적당한 스타일시트를 제작하여 문서의 처리나 조판 등을 자동화할 수 있을 것이다.

이 글에서는 pdf의 제작을 중심으로 테스트하고자 한다. 그러므로 SGML 태그들은 일단 고려하지 않기로 한다.

〈그림 2〉 한글로 열어본 문서 파일



〈그림 3〉 텍스트 변환의 결과

이기황(2007b)에서 제작한 레이텍 스타일 문서를 먼저 작성하고, 이것을 출력용 wrapper로 사용하기로 하자. 그 내용은 대략 다음과 같다.

```
\documentclass[11pt,a4paper]{oblivoir}
\usepackage{dhucs-midkor}
\begin{document}
\input a5ca0001.txt
\end{document}
```

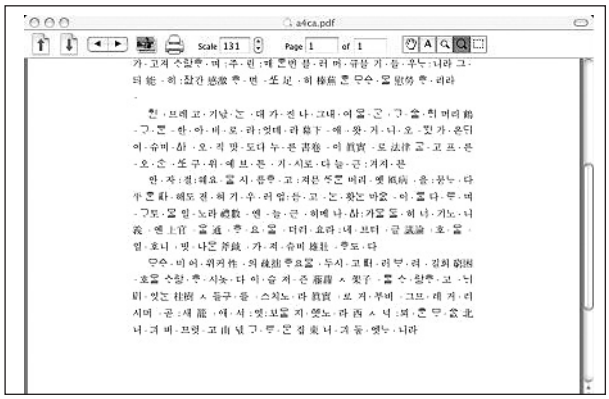
이제 같은 디렉토리에 첫가끝 코드로 입력된 텍스트 파일 a4ca0001.txt를 만들어 두기만 하면 pdf를 제작할 준비가 갖추어졌다. 그런데 여기서 훈글로 열어본 원본 문서에서 태그를 제거한다 하더라도 여전히 옛한글 코드가 소위 한양 PUA라 불리는 폰트 의존적인 옛한글 음절문자 사용자 영역 액세스 코드이므로[41] 그것을 옛한글 "텍스트"라고 하기는 어려운 상황이다. 이것을 훈글의 유니코드 텍스트 저장기능을 이용하여 저장하고 태그를 제거한 다음 자모 조합형 코드(단자모)로 일괄 변환하는 유틸리티 hypua2jamo를 적용하기로 하자[42]. 이 과정을 거친 후의 a5ca0001.txt 일부를 유니코드 편집이 가능한 에디터로 연 결과는 〈그림 3〉과 같다.

우리가 준비해둔 wrapper tex 파일에 대하여 pdflatex 또는 latex을 적용하여 컴파일한다. 그렇게 얻어진 pdf 파일이 〈그림 4〉이다.

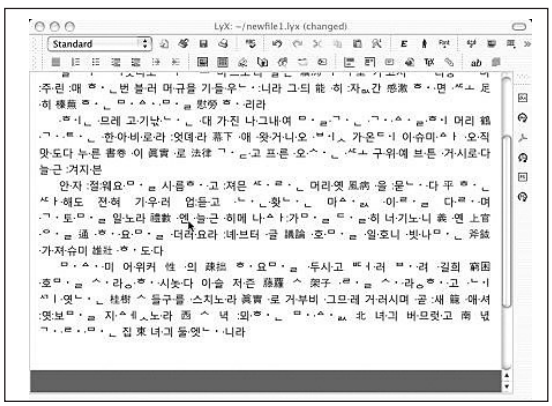
이 pdf 파일에 대해서는 자모 한글을 입력할 수만 있다면 자모 조합 코드의 텍스트 열을 검색할 수 있다. 이것은 매우 중요한 성과이며 종래의 방법으로 일부밖에는 검색이 불가능하던 상황을 극적으로 개선한 것이다. 이런 종류의 pdf로 데이터베이스를 구축하더라도 그 텍스트를 활용할 가능성이 있는 것이라고 판단된다.

텍과 레이텍에 익숙하지 않은 사용자들의 경우에는 LyX[44]이라는 레이텍 front-end 에디터를 사용하면 더 쉽게 같은 작업을 할 수 있다. 준비한 a5ca0001.txt의 내용을 (코드 손실 없이) LyX으로 붙여넣기만 해도 같은 pdf 결과물을 얻을 수 있기 때문이다. 아직 이 분야에서 LyX을 위한 스타일 파일 등이 제작되지 않았으

〈그림 4〉 최종결과물 pdf



〈그림 5〉 LyX 작업



나 이를 위한 추후 연구가 필요하다고 하겠다.

〈그림 5〉는 LyX을 이용하여 더 간단히 편집 작업을 행하는 것을 예시한 것이다. 한편 LyX을 비롯하여 이 테스트에서 사용된 모든 도구들과 프로그램은 자유 소프트웨어들인 점을 상기시키고자 한다.

7. 결론

이상에서 보인 대로 텍과 그 관련 도구들은 한글 문서의 전자화에 충분히 활용될 수 있을 정도로 발전하였다. 텍을 한글 문서 전자화에 활용함으로써 한국어 문헌의 전자적 활용 가능성을 높이고 일반과 연구자의 텍스트에 대한 접근성을 현저히 개선할 수 있을 것이다. 텍의 개방성, 이식성, 확장성을 효율적으로 활용함으로써 특정 워드 프로세서에 고착됨으로써 발생하는 여러 문제를 피해갈 수 있다.

그러나 텍 및 관련 자유 소프트웨어의 발전은 현재까지 소수의 헌신적인 개발자와 연구자

의 개인적이고 희생적인 기여에 의존해오고 있는 실정이다[45]. 최근 한국텍학회가 설립되어 이러한 연구 개발을 좀 더 체계적으로 지원하게 되었지만 우리나라에서 텍에 대한 관심은 아직 미미한 수준이다.

사회 공동의 자산이 되어야 할 전자 문헌과 전자 텍스트의 생산에서 탁월한 문서작성도구이자 자유소프트웨어인 텍이, 공공성 측면에서, 할 수 있는 가치 있는 역할을 생각해보건대, 활용 방법의 복잡성을 넘어설 수 있는 편리한 인터페이스의 개발이라든가 한글 텍 시스템 자체의 발전 등에, 인적 물적 자원이 더 배분될 수 있는 사회적 관심과 정책이 요구된다 하겠다.

【주】

1. 이 프로그램의 발음을 우리말로 표기하면 '텍' 이다. LaTeX의 경우에는 '레이텍' 또는 '라텍' 이라 읽는다.
2. TeX에 관한 자세한 기술적 정보와 사양, 전체 소스 코드는 Knuth(1986)에 기술되어 있다.
3. 현재는 dvi를 거치지 않고 직접 pdf를 얻을 수 있는 pdfTeX 엔진이 널리 쓰인다. 이에 대해서는 조진환(2007)을 보라.
4. TeX은 1980년대에는 거의 개발이 종료된다. 그 이후의 발전은 Knuth의 손을 떠나서 TeX-확장의 형태로 e-TeX, pdfTeX 등 새로운 엔진의 제작으로 이어졌다. 현재 우리가 일반적으로 텍이라고 부르는 것은 Knuth의 오리지널 텍이 아니라 pdf-e-TeX을 의미하는 수가 많다. 조진환, 같은 글.
5. 텍에서 매크로 명령처럼 다루어지는 문자(열)를 \active라고 한다. 즉 한 문자이지만 실제로는 텍 명령(control sequence)으로 처리되는 것이다.
6. 폰트의 디자인은 HLaTeX과 함께 은광희에 의하여 이루어졌다. http://project.ktug.or.kr/hlatex. 이 폰트를 트루타입으로 변환한 것은 일군의 오픈소스 개발 진영에 의해서였고, 주로 박원규에 의하여 완성되었다. 은 글꼴은 현재 리눅스 등 오픈소스 운영체제에서 광범위하게 활용되고 있다.
7. 한글과 컴퓨터사가 1995년 출시했던 '호TeX이라는 상용 프로그램이 제작되어 판매되기도 했다.
8. 유니코드의 기본 다국어 평면. 2바이트로 표현할 수 있는 코드 영역이 여기에 속한다. 이 영역에 한글은 현대한글 음절문자 11172자, 한중일 호환 한자, 한글 자모, 한글 호환 자모 들이 군데군데 배치되어 있다. BMP에서 가장 많은 평면을 차지하는 것은 한자이고 그 다음이 한글이다. UCS2라 함은 기본 다국어 평면에 속하는 2바이트 코드로 정의된 문자들을 이르는 말이다. http://www.unicode.org.
9. http://www.unruh.de/DniQ/latex/unicode.
10. utf-8 인코딩이란 유니코드 문자정의를 바이트열로 나타내는 규칙 중의 하나로서, BMP 영역 유니코드 문자들을 4바이트 이내의 가변 바이트열로 표현한다. 이 인코딩의 특징 중 하나는 ASCII 영역이 1바이트로 인코딩되므로 이 영역 문자에

대해서는 별도의 인코딩이 불필요하여 utf-8 유니코드와 ASCII가 호환된다는 점이다. 텍과 레이텍에서 utf-8 인코딩을 주로 쓰는 이유가 여기에 있다. 이 인코딩에 의하면 한글 음절문자 영역 코드들은 3 바이트 바이트열로 표현된다.

11. 김도현 · 김강수. http://faq.ktug.or.kr/faq/Hangul-ucs.
12. Hangul-ucs의 새 버전은 ko.TeX이라는 한국텍학회의 표준 한글 텍 시스템에 포함되었다.
13. Hartman, R. R. K. and Gregory James(1998), Dictionary of Lexicography, London and New York: Routledge. 이기황(2007a)에서 재인용.
14. 이 과정에서 조진환은 DVIPDFMx라는 유틸리티를 제작하여 한글 트루타입 사용에 결정적 기여를 하였다. http://project.ktug.or.kr/dvipdfmx.
15. 한국텍학회의 표준 한글 텍 시스템인 ko.TeX은 유니코드 한글뿐 아니라 euc-kr에 대한 지원까지 갖추고 레이텍, 플레인텍, 컨텍스트 등 다양한 텍 매크로에서 한글을 거의 완벽하게 구현하도록 지원할 계획이다. ko.TeX의 타이포그래피적 요소에는 김강수, "레이텍에서 한글 유니코드 문서 작성하기," Hangul-ucs 사용자 안내서 (온라인 문서) 및 김강수, "한글 문장부호의 식자 관행에 관하여," (The Asian Journal of TeX, 제1권 1호, 2007) 등의 연구결과가 많이 반영되었다.
16. 이주호, "TeX과 책," (온라인 문서) http://faq.ktug.or.kr/faq/Progress.
17. 이지영 (1999), p.11에서는 "아스키 데이터"라고 하였는데 이것은 엄밀히 말하면 ASCII가 아니다. 그러므로 "플레인 텍스트"라고 부르는 것이 옳을 것이다.
18. SGML에 대해서는 Goldfarb · Rubinsky(1991)을 보라.
19. XML을 빠르게 진화하고 있는 규격으로 자세한 내용은 http://www.w3.org/XML을 참조하라. 말뭉치 언어학에서의 XML의 이용에 대해서는 이기황(2007a)을 참고할 수 있다.
20. 물론 표준적인 XML 등의 마크업 문서로 export하는 기능을 갖추고 있으나 그것은 저자가 의도적으로 제작하지 않는 한 사용할 수 없을 것이다.
21. 레슬리에 람포트는 레이텍의 가장 큰 장점을 디자인과 저술의 분리라고 하였다. 여기에 마크업 언어의 철학이 반영되어 있다고 본다. Lamport (1994)참조.
22. 국내에서도 텍과 관련 시스템은 다양한 학계와 산업계에서 이용되고 있으며 2006년 텍과 관련 시스템의 보급과 자유로운 한글 표현을 위한 연구, 개발을 포함한 학술 활동을 목표로 '한국텍학회'가 설립되었다.
23. http://www.luatex.org.
24. CTAN:macros/latex/contrib/perltex.
25. 텍과 XML의 연동 가능성에 대한 내용은 Goossens and Rahtz (1999)에 개괄적으로 제시되어 있다.
26. 원래 레이텍은 그 출력물의 아름다움으로 유명하였다. 그러나 한글 레이텍의 경

우에는 자유 폰트의 부족, 개발 인력과 자원의 부족, 관심의 미흡함 등으로 인하여 이러한 정도로까지 고품위 타이포그래피를 구현하는 데 상당한 시간이 들지 않을 수 없었다. 최근 이루어진 한글 레이텍의 발전은 이에 필적할 정도라고 판단한다.

27. 이 단원은 필자에게 사신으로 제공한 이기황 박사의 정보와 의견에 기초하고 있다. 귀중한 의견을 제시해준 이기황 박사에게 감사한다.
28. 유니코드는 국제 표준이며 한글의 표현에 제약이 없다.
29. 이 표현은 KTUG에서 필자가 처음 썼던 말이다.
30. 국립국어원, 「21세기 세종계획 문자코드 표준화 연구」, 2005년 연구보고서.
31. 반면 마이크로소프트 워드는 워드 2000까지 흔글과 동일한 방식의 옛한글 처리를 하다가 워드 2002부터 옛한글 코드 체계를 자모조합(첫가끝) 방식으로 바꾸었다고 한다. 같은 글, p. 50.
32. 고려대학교 문자코드연구센터, "옛한글 자료 코드 표현의 기본 원칙 및 지침."
33. Adobe Systems(2004)에 pdf에 대한 상세 명세가 나와 있다.
34. 일부 pdf 생성 프로그램은 텍스트 정보를 사상하고 인쇄된 판면 자체를 이미지로 만들어 처리하는 경우가 있다. pdf 문서는 텍스트적 요소와 이미지적 요소를 결합한 것이므로 순전히 이미지만으로도 출력물을 만들 수는 있지만 이렇게 한 경우 텍스트가 사라져버리므로 바람직한 pdf 문서라고 하기 어려워진다.
35. 한글로부터 pdf를 생성하는 방법은 여러 가지가 있으며, 원칙적으로 가능하다. 이를 이용하여 고문헌 디지털화 방안을 제시한 리상용(2000)의 글에 의하면, 옛한글 및 한자를 지원하는 폰트를 이용하면 일부 텍스트에 대한 검색이 가능한 pdf를 제작할 수 있음을 보여주고 있다.
36. 한글과 컴퓨터사가 판매하는 워드 프로세서 흔글을 구매하는 것으로 거기 사용된 폰트의 pdf 임베딩 권한까지 얻게 되는 것은 아닌 것으로 보인다. 또한 상업용으로 판매되는 글꼴(윤 폰트 등)은 그 사용권에 pdf 임베딩을 금지하는 조항이 있는 경우가 있다. 이런 글꼴을 한글에서 사용하여 pdf를 제작하면 본의 아니게 폰트 저작권을 위반하는 상황이 생길 수 있다.
37. 이기황(2007b)에서 이 과정을 좀더 상세히 다루고 있다.
38. 한국텍학회 웹사이트 http://kts.ktug.kr. 한글텍사용자그룹 http://www.ktug.or.kr.
39. http://www.sejong.or.kr/sejong_kr/index.html.
40. 강순애(1999)에 따르면 고문서의 결자를 위한 한자세트를 만들려는 노력이 있었음을 보여주고 있다. 고문서를 처리하기 위해서는 약 5만자에서 7만자의 한자가 필요하다고 한다.
41. 이 코드 체계에서 어떤 규칙을 발견하기는 매우 어렵다. 다만 출현빈도 조사를 통해 선정된 5000여자를 순서대로 배열했을 뿐이므로 이 텍스트들을 전자적으로 처리하기 위해서 자소 분리를 한다든가

하는 데 상당한 노력이 들지 않을 수 없다. 그러므로 이 코드의 날 텍스트는 그대로 다룰 것이 아니라 단자모 자모조합 코드로 변환한 다음 처리하는 것이 오히려 노력이 적게 들 것이라고 생각한다. 이 변환을 위하여 KTUG에서는 많은 노력을 기울여 한양PUA 코드 테이블을 작성한 바가 있다. http://faq.ktug.or.kr/faq/Hanyang PuaTableProject. 이 글에서 언급되는 옛한글 코드 변환 유틸리티는 이 프로젝트의 결과물이다.

42. hypua2jamo와 jamo2hypua는 위의 hanyang pua table project의 결과를 이용하여 김도현이 작성한 perl script이다. 각주 41에서 언급된 웹사이트를 참조하라.
43. 한글 문헌 식자와 관련하여 자유 글꼴인은 글꼴이 가진 문제점은 한자의 부족이다. KS X 1001에 해당하는 4888자의 한자밖에는 들어 있지 않기 때문이다. 이 문제는 많은 한자 자면을 가진 ngulim.ttf 등의 폰트를 사용하도록 설정함으로써 해결할 수 있다. 기술적인 준비는 다 갖추어져 있다.
44. 레이텍 프론트 엔드 에디터이다. http://www.lyx.org. LyX은 1.5.0 버전부터 유니코드를 지원한다.

【참고문헌】

Adobe Systems(2004), *PDF Reference Version 1.6*, 5th edition, Adobe Press. http://www.adobe.com/devnet/pdf/pdf_reference.html.

Goldfarb, Charles F. and Yuri Rubinsky(1991), *The SGML Handbook*, Oxford: Oxford University Press.

Goossens, Michel and Sebastian Rahtz(1999), *The LaTeX Web Companion*, Addison-Wesley.

Knuth, Donald E.(1986), *The TeXbook*, Computers & Typesetting, Volume A, Addison-Wesley.

Lamport, Leslie(1994), *LaTeX: A Document Preparation System*, 2nd edition, Addison-Wesley.

강순애(1997), "한국고문헌정보시스템의 구축 및 전망,"「한국문헌정보학회지」, 제31권 제4호.

고려대학교 문자코드연구센터, "옛한글자료 코드 표현의 기본원칙 및 지침," http://www.code.re.kr.

국립국어원(2005),「21세기 세종계획 문자코드 표준화 연구」, 2005년 연구보고서.

김강수(2004), "레이텍에서 한글 유니코드 문서 작성하기," Hangul-ucs 사용설명서. 온라인 문서. http://faq.ktug.or.kr/faq/Hangul-ucs.

김강수(2007), "한글 문장부호의 조판 관행에 대하여," The Asian Journal of TeX, Vol. 1, No. 1.

김도현(2007), "폰트의 저작권이 TeX 출력물에 미치는 영향," The Asian Journal of TeX, Vol. 1, No. 1.

리상용(2000), "pdf를 이용한 고문헌 외 원

문 디지털화 방안에 관한 고찰,"「한국문헌정보학회지」, 제34권 제1호.

은광희(2005), "한글 라텍 길잡이," HLaTeX 사용설명서, 온라인 문서. http://project.ktug.or.kr/hlatex.

이기황(2007a), "전자 텍스트로서의 사전의 자동 조판에 대하여," 사전학회 발표자료.

이기황(2007b), "Hangul-ucs를 이용한 옛 한글 조판," The Asian Journal of TeX, Vol. 1, No. 1.

이지영(1999), "고서의 디지털화와 색인에 관한 연구,"「서지학 연구」, 제15집.

조진환(2007), "TeX: 조판 그 이상의 가능성," The Asian Journal of TeX, Vol. 1, No. 1.