임베디드 OCR시스템 개발을 위한 소프트웨어 아키텍쳐

김세호° 박재화
중앙 대학교 컴퓨터 공학부
seho@hilab.cau.ac.kr°, jaehwa@cau.ac.kr

Software Architecture for Embedded OCR System Development

Seho Kim° Jaehwa Pack
Chung-Ang Univ, Computer Engineering Dept.

요 약

최근 임베디드 환경에서는 정보 처리를 위한 문자 인식 기술이 많이 요구되고 있다. 하지만 임베디드 환경에서의 문자인식 시스템(Optical Character Recognition)은 제약적인 자원으로 인하여 플랫폼에 크게 의존하는 문제점을 안고 있어 재사용성을 기대하기 힘들다. 그렇지만 임베디드 환경에서 플랫폼에 독립적인 즉, 재사용이 가능한 모범적인 소프트웨어 아키텍쳐는 없다. 따라서, 본 논문에서는 임베디드 환경에서의 문자 인식 시스템 개발시 플랫폼에 독립적인 즉, 재사용이 가능한 소프트웨어 아키텍쳐를 제안하였다. 또한 제안한 아키텍쳐를 바탕으로 실제 임베디드 환경(WIPI, Qt)에 문자인식 시스템에 적용시켜보았으며, 더 이상 플랫폼에 의존적이지 않음을 확인 해 볼 수 있다.

1. 서 론

최근 임베디드 환경에서 정보처리를 위한 문자 인식 기술이 많이 요구 된다. 하지만, 임베디드 시스템에서의 문자인식 시스템 개발은 많은 어려움을 가지고 있다. 제약적인 환경 즉, 느린 CPU, 낮은 메모리, 낮은 배터리 파워 게다가 라이브러리 지원 유무로 인하여 한 번 개발된 문자 인식 시스템은 플랫폼에 의존적인 가장 큰 문제점이 있다. 또한 실수 연산을 위한 FPU(Float Pointing Unit)의 사용으로 연산시간이 오래 걸리고, 에러 발생시 디버깅이 데스크탑 보다 훨신 어렵고 시간이 많이 걸리게 되는 단점을 가지고 있다. 하지만, 아직 임베디드 환경에서 문자 인식 시스템 개발을 위한 모범적인 아키테쳐는 없다. 따라서, 개발된 문자 인식 시스템은 유지 보수 시 많은 시간과 비용이 들고, 재사용은 아예 기대하기 힘들다.

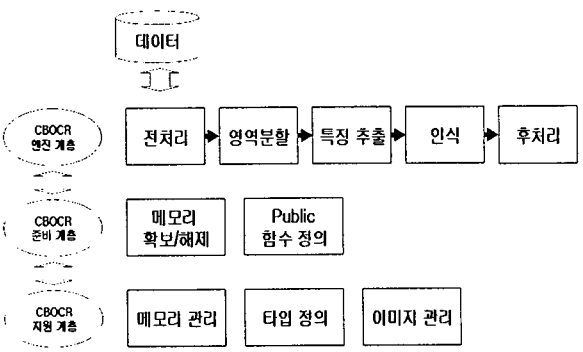

그림 1 임베디드 문자인식 시스템(OCR) 아키텍쳐

본 논문은 임베디드 환경의 오프라인 문자인식 시스템 개발을 위한 소프트웨어 아키텍쳐를 제안하였고, 실제로 임베디드 환경의 플랫폼(WIPI, Qt)에 적용해 보았다. [그림 1]에서는 우리가 제안한 임베디드 문자인식 시스템(OCR) 아키텍쳐를 보여주고 있다.

본 논문에서 제안한 소프트웨어 아키텍쳐는 임베디드 시스템의 의존성을 최소화하기 위해 크게 3개의 계층(지원 계층, 준비 계층, 엔진 계층)으로 나누었다.

2. 연구 배경

소프트웨어 아키텍쳐의 목표는 컴포넌트와 연결자들을 나타내는 높은 수준의 구조를(청사진)제공하는데 있다. 아래 [그림2.2] 에서는 소프트웨어 아키텍쳐로서 가져야 할 특징을 그림으로 표현해 주고 있다. [1][2][3]

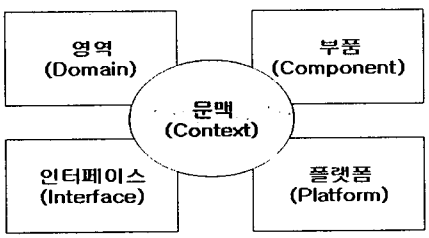

그림 2.1 아키텍쳐 특징

일반적인 오프라인 문자 인식 시스템의 구조는 [그림 2.1] 과 같이 스캐너, 카메라 혹은 디지타이저에 의해 이미지 파일로 저장되어진 영상을 바탕으로 전처리 과정, 영역 분할 과정, 특징 추출 과정, 인식 과정, 후처리 과정을 거쳐 인식되어진 문자들의 집합을 결과로 얻게 되는 전체의 과정을 의미 한다.[4]

3. 임베디드 문자인식 시스템 소프트웨어 아키텍쳐

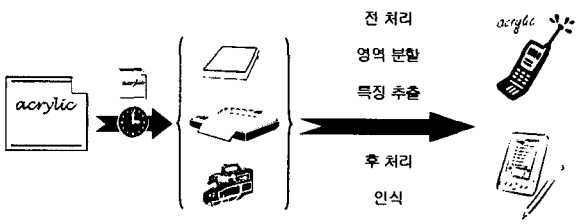

그림 2.2 문자 인식 시스템(OCR)

소프트웨어 개발 주기를 보면, 실제적인 구현 보다는 아키텍쳐 설계 쪽에 큰 비중을 두고 있다. 그 이유는 어떻게 아키텍쳐를 설계하느냐에 따라 개발되어진 시스템의 재사용성이 달라진다고 할 수 있기 때문이다. 특히, 임베디드 환경에서의 문자인식 시스템 개발은 기본적으로 다음 아래와 같은 모듈의 요건을 고려하여 아키텍쳐를 설계 하여야 한다.

1) 느린 CPU에서의 빠른 처리속도
2) 한정된 메모리 자원 내에서도 메모리를 효과적으로 조절 할 수 있는 메모리 조정능력
3) 대부분의 플랫폼은 소수점 연산용 FPU를 내장하지 않으므로 소수점 연산을 정수형 연산으로 변환 사용
4) 다양한 플랫폼에 대하여 이식성이 높도록 코드 작성.
5) 다양한 플랫폼마다 다른 데이터 저장 방식(Endian)에 대응 가능.

3.1 지원계층
(확장성, 고려사항의 분리)

문자인식 시스템(OCR) 지원계층은 시스템 개발 시 플랫폼이 변경될 때마다 의존적일 수밖에 없는 메모리 관리, 이미지 관리 그리고 데이터 타입 같은 부분을 따로 분리하여 정의한 부분이다.

이렇게 플랫폼에 의존적인 부분을 따로 분리함으로써, 각각의 플랫폼이 지원해 주는 메모리 할당 및 해제 방법의 의존성, 그리고 지원해 주는 데이터 타입에 따른 의존성, 지원해 주는 이미지 로딩 및 저장, 표시의 방법의 의존성을 해결함으로써, 전체 소스의 추가/수정 즉 확장이 용이하도록 하여야 한다. 다시 말해 이렇게 플랫폼에 의존적인 부분을 따로 분리함으로써, 플랫폼이 변경 될 때마다 각 플랫폼에서 제공하는 메모리, 데이터 타입에 따른 바이트 수에 맞게 전체 소스 코드를 찾아가며 하나 하나 수정해 줄 필요 없이 각각 하나의 파일만을 수정함으로써 전체 소스 코드가 수정되는 효과를 얻을 수 있다.

3.2 준비계층
(확장성, 추상화, 캡슐화, 인터페이스 구현의 분리)

오프라인 문자인식 시스템(OCR)은 시스템과 이 시스템을 호출하는 사용자는 오브젝트가 어떻게 호출되고, 내부 데 이터가 어떻게 저장되며, 어떻게 구현되어 있는가 하는 문제로부터 완전히 격리시켜야 한다. 그리고 객체에 포함된 정보의 오용과 손상을 방지시키고, 독립성 및 이식성을 증가시키기 위한 컴포넌트 캡슐화를 구현해야 한다.

즉, 사용자에게는 시스템의 내부 데이터를 숨기고, 하나의 Public 헤더 파일만을 제공하여, 접근 가능하도록 하여야 한다. 따라서 내부 구현을 숨기고 하나의 인터페이스만을 제공하여, 사용자가 사용하기 쉽게 해주고, 각 컴포넌트(모듈)간의 결합을 약하게 해 줌으로써 유지 보수 비용을 줄일 수 있다. 또한 여기서 Public 헤더 파일은 최소한의 완벽한 인터페이스를 제공해야 하고, 불필요한 함수 및 변수는 삽입하지 말아야한다.

문자인식 시스템(OCR) 시스템을 위한 인터페이스(즉, Public 헤더파일 함수)는 총 4개의 함수가 필요하다. <표3.2>는 문자 인식을 위한 최소한의 API 함수들을 보여주고 있다.

먼저, 응용 프로그램에서 시스템을 이용하고자 한다면, 시스템이 동작하기 위한 필요한 초기화 작업을 수행해 주어야 한다. 따라서, Construct 함수를 호출하여 인식에 필요한 전체 메모리를 확보하고, 인식시 필요한 데이터가 템플레이트 파일에 저장 되어 있을 경우 TemplateFile 함수를 호출하여 템플레이트 파일을 읽고 해당 메모리를 할당한 후 포인터를 넘겨 받는다. 이렇게 되면, 인식을 위한 각 처리 단계별 초기화 함수가 호출되어 각 처리 과정에 필요한 준비가 완료된 것이고, 이제 Recognition 함수를 통해서 인식기를 호출하여 인식할 수 있다.

즉, 응용프로그램을 통하여 인식기의 호출이 이루어지면, 인식을 위해 필요한 모든 메모리 영역을 확보하고, 필요한 데이터를 로딩시킨다음 그 영역 안에서 인식이 이루어지도록 해야 한다. 또한 응용프로그램 종료가 이루어지면 할당 받았던 모든 메모리 영역을 반환하도록 구조적으로 인식기를 구성해야 한다.

함수 명	설명
Construct	엔진을 초기화 및 메모리 할당
Destruct	할당된 메모리를 해제
Recognition	인식 엔진을 호출
TemplateFile	템플레이트 파일을 읽고 해당 메모리를 할당

<표 3.2> Public헤더 파일 함수 설명

3.3 엔진계층 (모듈화)

모듈화는 프로그램을 작성할 때 큰 시스템을 한 번에 작성하는 것이 아니라 기능별로 나누어 우선 부분별 작성 한다. 그리고 각각의 작은 프로그램들을 서로 연결시켜 하나의 완성된 프로그램을 만드는 방법이다. 이렇게 기능별로 나누어진 각 모듈은 하나의 기능을 수행하며 그 기능을 수행하기 위하여 필요한 모든 코드와 변수를 포함하도록 하는 프로그래밍 방식이다. 모듈화는 구조화 프로그래밍과 마찬가지로 프로그램의 구조가 논리적으로 구성되어 있어 유지 보수가 용이하고, 이해하기가 쉬운 장점이 있다. 따라서 OCR시스템 또한 [그림3.1]과 같이

각 단계별 프로세스(전처리, 영역 분할, 특징추출, 인식, 후처리)를 분리시켜야 한다. 그리고 하나의 인터페이스를 통해서만 접근 하도록 모듈화 해야 한다. 만약, 각 단계가 모듈화 되어 있지 않다면, 하나의 프로세스에 문제가 있거나, 새로운 기능이 추가 또는 필요 없는 기능이 삭제 될 때 많은 비용 들고, 재사용을 기대하기 힘들어 지므로, 각 단계를 모듈화 해야 하고, 각 모듈은 인터페이스만을 제공하게 하여, 독립성을 유지시켜 주어야 한다.

3.4. 템플레이트 파일

문자 인식기 에서는 인식을 위해 데이터를 저장하여 사용하는 경우가 있다. 데이터를 저장할 경우 데이터가 정해져 있고, 그것에 맞게 메모리를 할당하고 개수에 맞게 연산을 수행하도록 정적인 구조로 프로그램이 되어 있다면 데이터의 수정, 추가, 삭제 등 여러 경우에 문제가 생길 수 있다. 따라서 아래 [그림3.1]의 템플레이트 파일의 구조와 같이 헤더를 두어 이 헤더 정보를 읽어 오면서 그에 맞게 메모리 할당하고 연산을 하도록 구조화 하는 방법을 사용하여야 한다. 따라서 하나의 플러그인 아키텍처 파일을 통해 모든 기본적 구조가 잡히고 전체 프로그램 구조를 바꾸지 않고 이 파일만 수정함으로써 업데이트와 확장이 가능하도록 해야 한다.

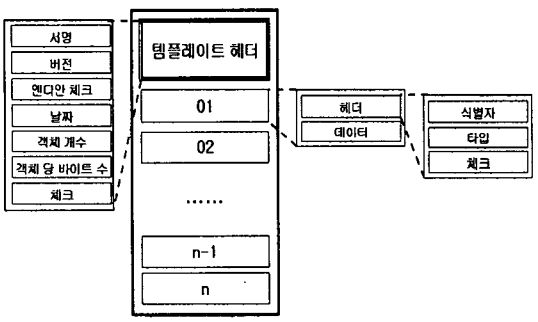

그림 3.1 템플레이트 파일

4. 실험

실험 환경은 WIPI, Qt 환경에서 테스트를 하였다. 메모리 환경은 최대 힙 사이즈 1M 에 플래쉬 메모리 8M 환경, FPU 와 라이브러리는 지원이 안 된다는 가정 하에서 실험 하였다. 아래 [그림 4.1], [그림 4.2] 는 각각 Qt 환경과 WIPI 환경에서 테스트한 결과를 나타내주고 있다.

5. 결론

다양한 임베디드 환경에서의 문자 인식 시스템은 일반적으로 데스크탑용 컴퓨터에 비해 여러 가지 이유에서 최소한의 자원만을 가지게 된다. 즉, 느린 CPU, 낮은 메모리, 낮은 배터리 파워 등과 같은 열악한 환경 속에 좋은 성능을 요구하기 때문에, 문자인식 시스템 개발에 많은 어려움을 가지고 있다. 게다가 라이브러리 함수의 지원여부 또는 실수 연산을 위한 FPU의 사용으로 인하여 실시간 처리에 문제가 있으며, 한 번 개발 되어진 시스템은 높은 이식성 즉 재사용은 더욱 기대하기 힘들다.

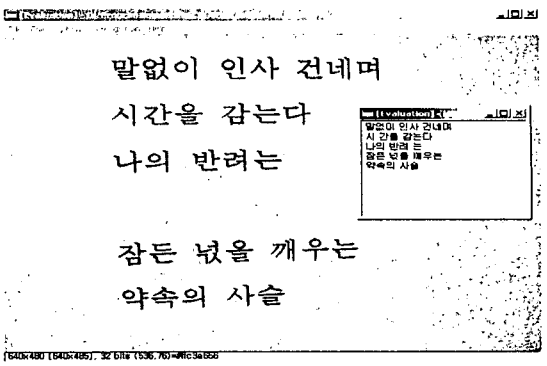

그림 4.1 Qt에서 인식되어진 결과

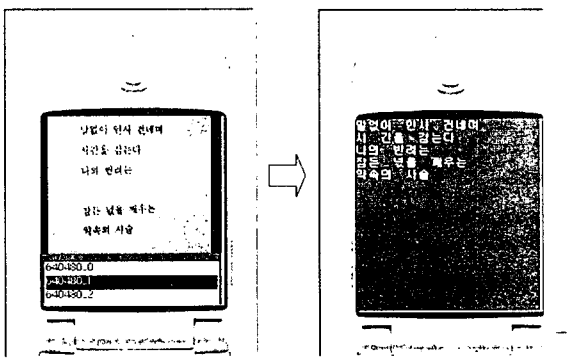

그림 4.2 WIPI 에서의 결과

따라서 본 논문에서는 임베디드 환경하에서의 카메라 기반 오프라인 문자 인식 시스템 개발을 위한 아키텍쳐를 제안하였다. 그리고 제안한 아키텍쳐를 적용하여 개발되어진 시스템을 실제 임베디드 환경인 WIPI, Qt 에 적용해 본 결과 더 이상 플랫폼에 의존적이지 않았음을 확인 할 수 있었다.

참고문헌

[1] David Garlan and Mary Shaw, "An Introduction to Software Architecture" in Advances inSoftware Engineering Volume 1 January 1994

[2] G.Zarayaraz and Dr.PaulRodrigues and Dr.P.Thambidurai and Dr.S.Duppuswami "A New Approach to Software Architecture" ACM SIGSOFT Software Engineering Notes, Volume 28 No2 March 2003.

[3] Margaret(Maggie)J. Davis and Roger B. Williams "Software Architecture Characterization" ACM SIGSOFT Software Engineering Notes, Proceedings of the 1997 symposium on Software reusability Volume 22 Issue3 May 1997.

[4] J. Park, "Hierarchical Character Recognition and Its Use in Handwritten Word/Phrase Recognition," Ph. D. thesis, State University of New York at Buffalo, 1990.