2003 정보 및 제어 학술회의 (2003.11.21-22)

자바 카드에서의 UICC 파일 시스템 설계 및 구현

UICC File System Design and Implementation for Java Card

김 학 두*, 전 성 익**

* 한국전자통신연구원 IC카드연구팀(전화:(042)860-3812, 팩스:(042)860-5611, E-mail : rexxion@etri.re.kr)
** 한국전자통신연구원 IC카드연구팀(전화:(042)860-5562, 팩스:(042)860-5611, E-mail : sijun@etri.re.kr)

초 록 : UICC는 애플리케이션, 파일 시스템, 보안 메커니즘, 암호 알고리즘 등을 포함하고 있는 일종의 스마트 카드이다. UICC의 파일은 ISO7816-4 표준에서 정의한 파일들과 애플리케이션의 최상 위 파일인 Application Dedicated File등이 있다. UICC가 자바 카드 기반에서 구현되었을 때 모든 파일은 객체로서 구체화될 수 있다. 이때 각 파일 객체는 파일에 대한 정보들과 정보 조작과 관련된 메소드를 제공하게 된다. 자바 카드는 일반 컴퓨터와 유사하지만 제한적인 메모리와 처리속도 등으로 인하여 구현 상 많은 제약사항이 따른다. 자바 카드는 저장 공간을 ROM, RAM, EEPROM로 구분할 수 있으며, 파일이나 애플릿, 데이터 등은 EEPROM 영역에 저장된다. 하지만 자바 카드가 지원하는 EEPROM영역이 2.2.1 버전에서 확장되었다 할지라도 여전히 많은 데이터를 저장하기에는 부족한 메모리 공간을 갖는다. 이것은 메모리 사용에 있어 신중을 기해야 한다는 것을 의미하며 효율적인 메모리 사용은 카드 사용자에게 보다 많은 가용 메모리를 제공할 수 있다는 점에서 중요하다. 본 논문에서는 이러한 점을 고려하여 자바 카드에서 UICC의 파일 시스템을 Linked List 방식을 이용하여 구현하는 방법과 배열을 이용하여 구현하는 방법을 제시하고 파일 시스템의 전체적인 구조를 효율적으로 구성하는 방법을 제시한다..

주요어 : UICC, File System, Java Card

Ⅰ. 서론

스마트카드는 일종의 소형화된 컴퓨터라고 할 수 있으며 연산 장치(CPU), 저장장치, I/O 장치 등을 내장하고 있다. 스마트카드에서는 일반 컴퓨터와 마찬가지로 데이터를 저장하기 위해서 파일이라는 개념을 사용한다. 스마트카드에서 사용되는 파일의 종류는 Master File, Dedicated File, Elementary File이 있으며 Elementary File은 다시 Transparent File, Linear Fixed File, Cyclic File로 그 형태에 따라 구분 된다[3]. UICC는 자바 카드를 기반으로 구현될 수 있다. 자바 카드는 스마트카드의 환경에 맞게 개량된 가상머신(Virtual Machine)을 탑재하고 있다. 이 가상머신은 일반 자바 가상 머신에 비해 상당히 경량이며 그 내부의 기능면에서도 약간의 차이를 보인다. 자바 카드는 메 RAM, ROM, EEPROM등과 같은 메모리를 사용하고 있으며 파일이나 기타 보존이 필요한 데이터, 그리고 애플리케이션은 EEPROM에 저장한다. 최근의 2.2.1버전의 자바 카드에서는 64Kbyte의 EEPROM을 지원하고 있다 [5]. UICC 카드에서 생성되는 파일은 USIM(Universal Subscriber Identity Module)애플리케이션에서 정의도어 있는 것만 100여개를 초과하고 2G를 위한 파일까지 포함하게 되면 약 200여개가 된다[4]. 여기에 애플리케이션과 기타 데이터까지 포함한다면 64Kbyte의 메모리는 여전히 넉넉하지 않은 양이다. 이런 문제점을 해결하기위해서는 메모리를 사용하는데 있어 신중을 기해야 하며 효율적인 파일 시스템의 설계는 가용 메모리 확보에 직결된다고 볼 수 있다.

본 논문의 2절에서는 자바카드에서 파일 시스템을 구성할 때 파일의 상속관계와 파일 시스템을 구성하는 방법 2가지를 살펴보고 3절에서는 실험을 통하여 두 가지 방법을 비교하며 마지막으로 4절에서는 최종 결론과 향후 계획을 서술하였다.

Ⅱ. 자바카드에서의 UICC 파일 시스템

1. 파일 클래스의 상속관계

<그림 1>는 파일을 클래스로 나타내었을 때 파일 간 상속관계를 나타낸다. 파일 시스템의 최상위에는 File이라는 추상 클래스가 존재하며 모든 파일 클래스의 부모 클래스가 된다. 파일은 자신의 부모 파일 내에서 유일하게 자신을 구별할 수 있게 하는 FID(File Identifier)와 파일의 크기, 파일에 대한 접근 권한 등과 관련된 정보를 제공한다[3]. Dedicated File은 일반적으로 컴퓨터에서 말하는 Directory를 상징하는 파일 클래스이다. Dedicated File은 자식 파일에 대한 검색을 제공하며 Application Dedicated File의 부모 클래스가 된다. Application Dedicated File은 Dedicated File의 특성을 갖고 있으며 추가로 Application Identifier 라는 속성을 가지고 있어서 SELECT Command 시에 DF Name에 의한 선택을 지원하게 된다. DF Name에 의한 선택이 이루어 질 경우에는 특정 애플리케이션이 활성화 되며 Application Dedicated File이 Root 파일로서 선택된 상태가 된다. Elementary File은 File을 상속하며 Linear Fixed File과 Transparent File의 부모 클래스이다. Linear Fixed File과 Transparent File의 가장 큰 차이점은 내부의 데이터 저장 방식에 있다. Linear Fixed File은 데이터를 일정한 길이를 가진 여러 개의 Record로 구분하여 저장하지만 Transparent File은 데이터에 어떤 구분을 두지 않고

연속적인 값을 순차적으로 저장하는 가장 기본적인 Elementary File이라고 할 수 있다. Linear Fixed File은 레코드 검색, 읽기, 쓰기를 지원하며 Transparent File은 단순한 데이터 읽기, 쓰기를 지원하도록 설계한다. Cyclic File은 Linear Fixed File을 상속하게 되는데 Linear Fixed File과의 구조적인 차이로 인하여 몇몇 Linear Fixed File의 메소드를 Overriding하여 구현하도록 한다. 이러한 상속관계를 이용하여 클래스를 구현하는 이유는 하위 클래스가 부모 클래스의 메소드나 데이터를 재사용할 수 있다는 장점이 있으며 이는 코드의 중복을 줄여 메모리를 보다 효율적으로 사용할 수 있기 때문이다.[3]

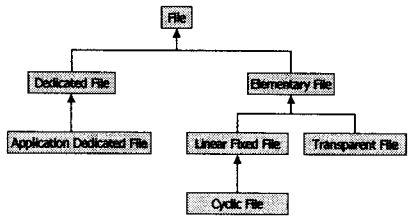


그림 1. 파일 클래스의 상속 관계

2.FileSystem구조

2.1 Linked List 구조를 이용한 파일시스템

자바는 변수에 대한 참조에서 일반 변수와 같은 경우는 Call by Value형태를 취하고 있으며 객체와 같은 경우는 Call by Reference 형태를 취한다[6]. 자바 카드에서도 마찬 가지로 Object형 변수는 Object에 대한 참조를 가지게 된다. <그림 2>는 파일 클래스의 구조를 나타내고 있다. 파일은 자신의 다음 파일을 가리키는 참조(*next*)와 자신의 부모 Dedicated File(*parent*), 그리고 만약 클래스가 Dedicated File이라면 자신의 첫 번째 자식 파일에 대한 객체 형 변수(*child*)로 가지고 있다. 이는 Linked List 방식을 파일 구조에 적용하기 위해 설계된 것이다.

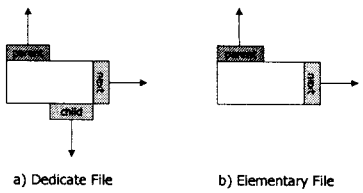


그림 2. Linked List를 위한 파일 클래스 구조

Linked List는 자신의 다음 원소에 대한 참조를 가리키는 변수를 가지고 있는 단순 Linked List와 이전 원소에 대한 참조까지 포함하고 있는 이중 Linked List로 구분할 수 있으며 본 논문에서는 단순 Linked List를 이용하여 메모리의 사용을 줄이도록 하였다. 앞에서도 설명하였듯이 DF는 EF와 DF를 자식 파일로 포함할 수 있다. <그림 3>에서 DF1은 EF1, DF2, EF2을 자식 파일로 포함하고 있으며 DF2는 EF3을 자식 파일로 포함하고 있다. DF1의 자식 파일 중 가장 끝에 연결되어 있는 자식 파일인 EF2의 *next*가 Null을 가리키고 있어서 파일을 검색할 때 자신이 끝이라는 것을 나타낸다. 또한 DF1은 자신의 첫 번째 자식 파일인 EF1의 참조(*child*)만을 가지고 있어서 모든 자식 파일에 대한 링크를 가지고 있는 것보다 메모리 사용이 적다. 현재 선택된 파일이 어떤 파일이건 간에 현재 파일에서 특정 파일을 검색하기 위해서는 자신의 부모 Dedicated File부터 시작한다[1]. 이런 이유 때문에 모든 자식 파일은 자신의 부모 파일에 대한 참조(*parent*)를 가지고 있어야 한다.

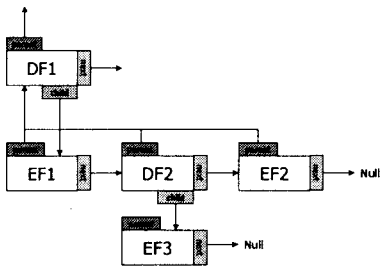


그림 3. Linked List를 이용한 파일 시스템

<그림 4>은 MF와 ADF와의 관계를 나타낸다. MF는 자식 파일로서 EF1, ADF1, ADF2들을 가지고 있다. EF1의 *parent*는 MF를 가리키고 있는 반면 ADF1이나 ADF2는 MF를 가리키고 있지 않다. 이는 UICC의 파일 검색 범위에 대한 고려 사항을 만족시키기 위해 제거한 것이다. 즉 ADF1은 애플리케이션의 최 상위 부모 파일로서 MF의 자식 파일에 대한 직접 선택이 금지되기 때문이다[1]. 만약 ADF1의 부모를 MF로 설정하게 되면 UICC 표준에 따라 MF는 ADF의 부모가 되고 EF1은 MF의 자식 파일이기 때문에 접근이 가능하게 된다.

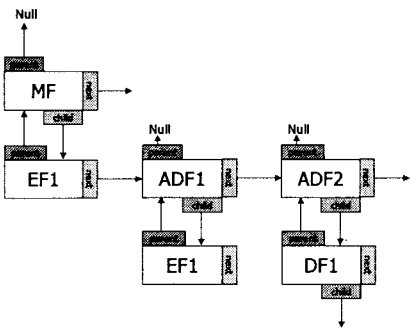


그림 4. Linked List를 이용한 파일 시스템. ADF는 MF를 부모 파일로 설정하지 않음.

<그림 5>은 Linked List 구조로 이루어진 파일 시스템에 새로운 파일을 추가할 때의 변화 모습을 보여준다. a)는 초기 파일 시스템의 상태를 보여주는데

DF1이 단지 하나의 Elementary File인 EF1을 자식 파일로 가지고 있다. b)는 DF1에 새로운 파일인 EF2를 추가하기 위해 EF2를 생성하는 것을 보여준다. c)는 EF2의 *next*에 EF1을 대입하고 *parent*에 DF1을 대입하는 것을 보여준다. d)는 DF1의 *child*에 EF2를 대입하는 것으로 파일의 생성 과정이 끝나게 된다.

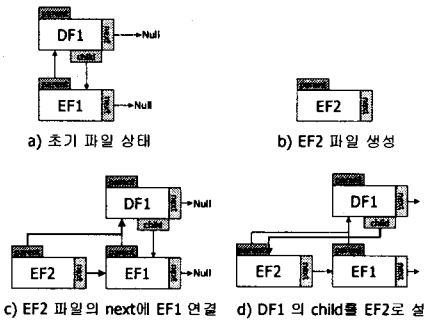


그림 5. Linked List를 이용한 파일 시스템에서의 파일 추가

<그림 6>은 파일시스템에서 파일을 제거하였을 때의 과정을 나타내는 것으로 EF1, EF2, EF3가 연결되어 있는 상황에서 EF2를 제거하는 과정이다. 최초 EF1의 *next*에 EF2의 *next*를 대입하여 EF1의 *next*가 EF3를 가리키게 한다. 이후 EF2 을 가리키고 있는 변수에 *Null*를 대입하고 Garbage Collection[5]을 요청하여 EF2에 할당된 메모리를 회수하는 과정을 거치면 모든 삭제 과정이 끝나게 된다.

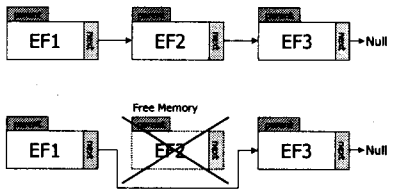


그림 6. Linked List를 이용한 파일 시스템에서의 파일 삭제

2.2 배열 구조를 이용한 파일 시스템

<그림 7>는 파일 시스템을 배열 방식을 사용하여 설계한 것을 나타낸다. Dedicated File은 자식 파일들의 참조를 배열을 사용하여 저장하게 되고 자식 파일들은 부모의 참조를 가지게 된다. 배열은 원소의 검색이 쉽고 원소를 생성할 때나 삭제할 때 별도의 작업이 필요치 않으며 배열에 추가만 하면 되기 때문에 조작이 쉽다는 장점이 있다. 하지만 배열은 처음 생성 때의 크기를 변경할 수 없으므로 배열의 원소가 모두 사용 중인 상태에서 원소를 추가하였을 경우 배열의 크기를 늘려야 하는데 이런 과정은 상당한 부하를 초래하게 된다. 또한 원소가 삭제되었을 때 배열의 중간 원소가 비게 되고 차후 객체 원소를 추가하였을 때 사용 가능한 원소를 찾아 그곳에 객체의 참조를 연결해야 하는데 이런 과정 또한 Linked List방식을 사용한 것에 비해 비효율적이다.

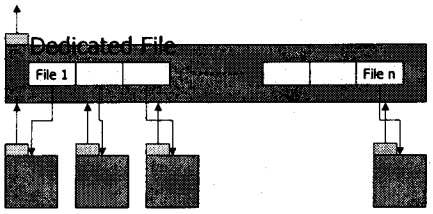


그림 7. 배열을 이용한 파일 시스템

<그림 8>은 자식 파일을 가리키는 배열 *child*가 모두 사용 중일 때 새로운 파일이 추가된 경우 배열이 어떻게 변경 되는지 보여주고 있다. 먼저 기존의 배열의 크기를 N이라고 하였을 때 원소의 개수가 $N + m$ (m>0 인 정수) 인 새로운 배열을 만든다. 이 때 m이 1인 경우, 차후에 새로운 파일이 추가되었을 때, 다시 전체 배열의 크기를 증가시켜야 하며 이 작업은 상당한 부하를 초래하게 된다. 또한 m값이 너무 큰 경우, 메모리의 낭비를 초래하게 된다. 그러므로 m값을 결정할 때는 이 점을 고려해야 한다.

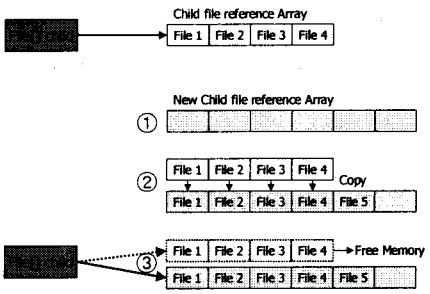


그림 8. 배열을 이용한 파일 시스템에서의 파일 추가

배열을 이용하여 파일시스템을 구성하는 방법은 애플리케이션을 설계할 때 생성될 파일의 정확한 개수와 부모와 자식 파일 간의 종속관계를 파악할 수 없을 때 또는 차후에 파일의 추가가 예상되는 경우에 효과적인 방법이 될 수 없다.

III실험 및 결과

이 절에서는 배열로 구성한 파일 시스템과 Linked List로 구성한 파일 시스템과의 실험을 통하여 각 구성 방법의 장단점을 살펴본다. 실험은 실제로 ARM 32bit Processor를 사용하는 스마트카드 애뮬레이터를 이용하여 실시하였다. 실험에서 사용되는 파일 시스템은 배열 구조와 Linked List구조와 같이 전체 구성 방식에 대해서만 차이를 보일 뿐 파일에 대한 정보를 나타내는 FCP(File Control Parameter)[3]를 구성하는 방법이나 보안 관련 부분 등 파일 객체를 구성하는 다른 요소는 동일한 조건을 사용하여 실시하였다. <그림 9>는 실험에서 생성될 파일 시스템을 나타낸 것으로 Master File과 10개의 Elementary File로 구성되어 있

다. 실험에서 생성될 Elementary File은 Transparent 형태의 파일이며 저장할 수 있는 데이터의 크기는 10byte이다. 모든 Elementary File들은 FID를 제외한 모든 정보나 내용이 동일하다.

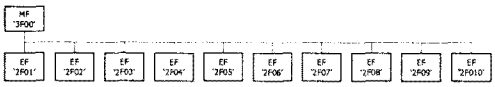
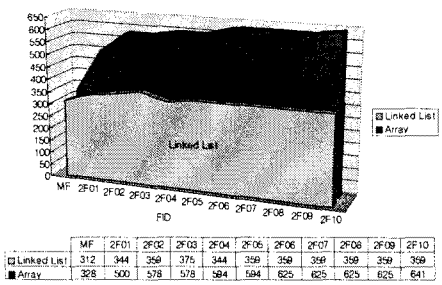


그림 9. 파일 시스템 예제

파일 생성 시간 측정은 Command APDU(Application Protocol Data Unit)[3]를 전송할 때의 시간과 Response APDU를 수신하고 나서의 시간을 측정하여 그 차이를 계산하는 방법을 사용하였다. 카드 내에서의 부하로 인한 오차나 PC와 리더기간의 전송시간 지연 또는 PC내부에서의 지연으로 인한 오차 등을 고려하여 Linked List와 배열을 이용한 방법에 대해 각 10회씩 측정한 시간의 평균을 구하는 방법으로 값을 구하였다. <그림 10>은 실험의 결과 그래프이며 Y축은 결과시간을 X축은 생성파일의 FID를 나타낸다.



	MF	2F01	2F02	2F03	2F04	2F05	2F06	2F07	2F08	2F09	2F10
Linked List	312	344	359	375	344	359	359	359	359	359	359
Array	328	500	578	578	594	594	625	625	625	625	641

그림 10. 파일 생성 시간 그래프

실험 결과에서 알 수 있듯이 Linked List 방법을 이용한 파일 시스템 구성이 배열을 이용한 파일 시스템 구성에 비해 파일 생성시간이 짧다는 것을 알 수 있다. 또한 파일을 생성해 감에 따라 Linked List를 이용한 파일 시스템에서는 생성시간에 변화가 없는 반면 배열을 이용한 파일 시스템에서는 점점 파일 생성 시간이 오래 걸리는 것을 알 수 있다. 이는 배열을 이용한 파일 시스템에서는 파일을 추가할 때 크기가 증가된 배열을 새로 만들고 이전의 배열에서 새로 생성된 배열로 각 원소를 복사하는 연산에서 상당한 부하를 초래하기 때문이다.

사용 가능 메모리는 Linked List 방법을 이용한 파일 시스템에서 전체 파일일 생성했을 때 42,869byte인 반면 배열을 이용한 파일 시스템에서는 42,859byte이었다. Linked List를 이용한 파일 시스템이 객체를 생성할 때 배열을 이용한 파일 시스템보다 내부 변수가 하나 많음에도 불구하고 사용 가능 메모리의 양이 많다. 이런 결과를 보이는 이유는 배열이 차지하는 메모리의 양이 많기 때문이다. 배열을 이용한 파일 시스템에서 파일을 생성하였을 때, Dedicated File의 내부 변수인 배열의 원소에 파일의 참조를 대입하게 된다. 그러나 그 파일의 원소는 파일형의 객체이기 때문에 Linked List를 이용한 파일 시스템에서 파일 객체가 가지고 있는 *next*와 같은 크기가 된다. 하지만 Linked List를 이용한 파일 시스템에서는 Dedicated File이 자신의 맨 처음 자식 파일에 대한 참조만을 갖고 있는 반면 배열을 이용한 파일 시스템에서는 전체 자식 파일에 대한 참조인 배열 객체를 가지고 있기 때문에 Dedicated File를 생성할 때 Linked List를 이용한 파일 시스템에 비해 많은 메모리를 소비하게 된다.

Ⅳ. 결론

본 논문에서는 UICC 파일 시스템을 자바 카드에서 구현하는 방법에 대해 기술하였다. UICC를 개발할 때는 제한적인 메모리 용량을 고려하여 가능한 한 효율적으로 프로그램을 작성해야 한다. 본 논문에서는 이러한 점을 고려하여 Linked List 와 배열을 이용한 방법을 기술하였다. Linked List를 이용한 파일 시스템 구성은 배열을 이용한 파일 시스템 구성에 비해 파일 시스템의 조작이 유연하며 불필요한 메모리의 낭비를 줄일 수 있고 파일을 생성할 때 속도가 빠르다는 장점을 가지고 있다. 배열을 이용한 구성은 Linked List를 이용한 파일 시스템 구성에 비해 파일 검색이 수월하며 파일을 삭제할 때 Linked List를 이용한 파일 시스템 구성에 비해 불필요한 연산과정이 필요치 않다는 장점을 가지고 있다. 하지만 배열을 이용한 파일 시스템은 배열의 특성으로 인하여 불필요한 메모리의 낭비를 초래하게 될 수 있다. 향후에는 두 가지 파일 시스템에서 파일을 삭제 할 때의 성능 측정과 또한, 파일의 구조뿐만 아니라 파일이 차지하는 저장 공간과 파일의 접근이나 변경 등의 제어와 관련된 FCP를 효율적으로 관리하고 조작하는 방법에 대해 연구하고자 한다.

참고문헌

[1] European Telecommunications Standards Institute ,"UICC-Terminal Interface; Physical and Logical Characteristics", Technical Specification 102 221, 2003.

[2] European Telecommunications Standards Institute ,"UICC Application Programming Interface(UICC API) for Java Card™", Technical Specification 102 241, 2003.

[3] International Organization for Standardization,"Id entification Cards - Integrated Circuit(s) Cards with Contacts. Part 4: Interindustry Commands for Interchange", 7816-4, 1995.

[4] The 3rd Generation Partnership Project," Char acteristics of The USIM Application", Technical Specification 31.102, 2003.

[5] http://java.sun.com/products/javacard/

[6] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha, "The Java™ Language Specification Second Edition", Addison-Wesley, 2000.