모바일 GUI를 지원하는 WFC에서 포커싱 가능한 테이블 컴포넌트의 설계 및 구현

전종찬* · 김정익* · 강영만** · 한순희**

Design and implementation of the focusable table component for mobile application using the WFC

Jong-chan Jun* · Jeong-ik Kim* · Young-man Kang** · Soon-hee Han**

본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음
(NIPA-2010-C1090-1021-0012)

요 약

본 연구는 모바일 기기 상에서 애플리케이션 작성 시 GUI를 지원하는 윈도우 기반의 컴포넌트 패키지 소프트웨어 모듈인 WFC에서, 보다 다양한 콘텐츠의 표현과 조작을 지원하기 위한 포커싱 가능한 테이블 컴포넌트를 제안한다. 모바일 기기의 특성상 작은 디스플레이에서 테이블 특성을 가지는 콘텐츠의 표현은 쉽지 않다. 자바와 같은 관련 연구들에서 몇 가지 방법들이 제안되었으나, 이들은 모바일 기기의 특성을 고려하여 제안된 것들은 아니다. 본 연구에서는 모바일 기기의 디스플레이상에서 테이블 특성을 가지는 콘텐츠의 알맞은 출력을 지원하고, 사용자가 지정하는 콘텐츠에 포커싱 조작이 가능하게 함으로써, 일반적인 테이블에 비해 보다 폭넓은 표현을 제공하는 확장된 테이블 컴포넌트를 설계하고 구현한다. 또한, GUI 관점에서 성능 평가를 위해 제안된 컴포넌트와 자바 Swing의 JTable과의 비교를 제시한다.

ABSTRACT

In this paper, we propose a focusable table component for mobile application to support table form representation of various contents and manipulation such as focusing on the contents using the WFC which is a software module to support components of the window based GUI package. It is not easy for us to express contents with table attributes on mobile devices. Java provides several table components, but these are not suitable on the mobile environment. So we design and develop the extended table component to provide properly table form representation and manipulation on the mobile devices. Also, we provide a performance comparison between th supposed table component and Java Swing's JTable.

키워드
테이블 컴포넌트, 모바일 GUI, WFC, 포커싱 테이블

Key word
Table component, Mobile GUI, WFC, Focusable table

* 전남대학교 디지털컨버전스
** 전남대학교 문화콘텐츠학부

접수일자 : 2009. 07. 31
심사완료일자 : 2009. 09. 16

Ⅰ. 서 론

모바일 기기 상에서의 콘텐츠 표현에는 일반적인 환경에서와는 달리 많은 제약사항들을 가진다. 적은 용량의 메모리, 입력키의 수 외에도, 특히 작은 디스플레이 장치로 인한 표현 공간에 대한 제약이 주된 요인으로 작용한다. 모바일 기기 상에서 이러한 제약사항들을 가지고서 일반적인 콘텐츠를 표현하고자 하는 경우 사용자가 의도하는 바와 유사한 출력물을 얻기 위해 콘텐츠의 적절한 가공이나 불필요한 부분의 생략 등의 처리과정을 거치게 된다. 콘텐츠에 따라서는 날씨, 주식, 일정 등 반복된 일련의 수치 값의 표현과 같이 그 특성상 내용을 가공하거나 생략하기가 어려운 것들이 있다. 이 경우 보통 리스트나 테이블과 같은 형태를 통하여 쉽게 표현하게 된다[1].

테이블 특성을 가지는 콘텐츠를 모바일 기기 상에서 표현하고자 하는 경우 테이블 컴포넌트의 지원이 필요하다. 만약 테이블 컴포넌트 없이 이를 표현하고자 한다면, 개발자가 일일이 상황에 알맞게 행과 열에 맞추어 내용을 배치하고, 각 셀의 구분에 해당하는 라인을 긋는 등의 행위를 매번 해야 하는 번거로움이 있을 뿐만 아니라 콘텐츠가 동적으로 변하게 되는 경우 이에 대한 처리는 매우 복잡하게 될 것이다.

테이블 콘텐츠의 표현에는 단순한 출력 외에도 필요에 따라서는 테이블 내에 포함된 콘텐츠에 포커스 제어가 가능하도록 복합적인 표현을 고려할 수 있다. 이것은 테이블 공간 내에 표현과 입력 제어를 함께 제공하므로 작은 디스플레이 장치를 가지는 모바일 기기에서 더욱 유용하다. 테이블 내 특정 셀의 내용으로 또 다른 테이블 콘텐츠를 표현하거나 제어가 필요한 콘텐츠를 배치하여 사용자의 입력 요구의 표현을 한 테이블 내에서 표현할 수도 있도록 하기 위해서는 일반 테이블의 기능 확장이 필요하다.

WFC(Wireless Foundation Classes)는 모바일 애플리케이션 작성 시 GUI 표현을 효과적으로 처리하기 위해 GUI를 지원하는 윈도우 기반의 컴포넌트 패키지 소프트웨어 모듈이다[2]. 이 WFC에는 기본적인 기능을 제공하기 위해 여러 컴포넌트들을 제공하고 있지만, 테이블 표현에 관한 컴포넌트는 별도로 제공되는 것이 없어 테이블 특성을 가지는 콘텐츠를 표현하기에는 어려움이 있다. 이를 해결하기 위해 본 연구에서는 모바일 애플리케이션의 GUI기능을 지원하는 WFC에서, 테이블 내에 사용자의 입력에 따른 포커싱이 가능하게 하여 콘텐츠에 대한 보다 폭넓은 표현을 지원하는 테이블 컴포넌트를 제안한다.

2장에서는 GUI관점으로 자바에서 제안된 테이블 표현에 관련한 연구들을 기술한다. 3장에서는 테이블 표현에 있어서 포커싱을 지원하기 위해 요구되는 사항들과 모바일 기기상의 제약을 고려한 사항들을 바탕으로 테이블의 설계와 구현에 대해 기술하고, 구현의 예를 제시한다. 4장에서는 자바 Swing에서 제안하고 있는 테이블 컴포넌트와 WFC의 테이블 컴포넌트에 대한 성능 비교를 제시한다. 마지막으로 5장에서는 결론과 함께 향후 연구에 대해 기술한다.

Ⅱ. 관련 연구

WFC에서 클래스 정형화 모델을 위한 단순 참조로 활용되었던, 자바의 AWT(Abstract Window Toolkit)는 GUI 지원을 위한 다양한 컴포넌트들을 클래스기반으로 지원하고 있지만 테이블 컴포넌트에 대해서는 지원하고 있지 않다[3]. AWT의 부족한 컴포넌트들과 보다 다양한 기능들을 제공하기 위해 만들어진 Swing에서는 JTable라고 명명된 테이블 컴포넌트를 지원하고 있다[4]. 그리고 어느 시스템에서나 일관된 GUI를 제공하는 Swing과는 달리 시스템 고유의 GUI 스타일을 지원함으로써 이질감을 낮추고, 2D 그래픽 출력속도를 향상 시킨 SWT(Standard Widget Toolkit)에서도 Table 위젯이라는 컴포넌트를 제공하고 있다[5]. 이 컴포넌트들은 모두 포커싱이 가능한 테이블 구성이 지원되지만, 일반 PC상에서 널리 사용될 수 있는 범용성과 확장성을 가지는 형태로 구성되어 있어 모바일 환경에는 부적합하다. 그리고 모바일 기기 적용에 있어서 자바 VM(Virtual Machine)상에서 구동됨으로 인한 성능저하, 리소스 사용 등의 제약사항들을 고려해야만 하기 때문에 적용 가능성 여부에 대한 별도의 검토가 필요하다[6]. 이런 점들 외에도 개발언어, 구현방법 등에 있어서도 본 연구와는 접근 방법이 다르다. 또한 4장에서는 성능비교 관점의 실험결과를 제시한다.

III. 포커싱 가능한 테이블 컴포넌트

본 장에서는 테이블 컴포넌트가 부재한 WFC에서 표현의 확장과 조작의 편의성을 지원하기 위해 포커싱 기능을 가지는 2차원 격자(grid) 형태의 테이블 컴포넌트를 제안한다. 먼저 테이블 컴포넌트에 있어서 포커싱 지원을 위해 요구되는 사항들과 모바일 환경을 고려한 사항들에 대해 기술한다. 그리고 이 사항들을 고려한 구현 방안을 제시하고, 구현의 적용 예시를 통하여 올바른 구현이 이루어졌는지를 보인다.

3.1 설계

WFC에서의 테이블 컴포넌트는 셀 내부에 또 다른 컨트롤 컴포넌트를 삽입 가능하게 함으로써, 일반적인 콘텐츠의 표현을 위한 기본적인 처리 외에도 확장된 컴포넌트 컨트롤을 제공하도록 하는 것이 단순히 출력만을 위한 일반적인 테이블과 다른 점이다.

확장된 컴포넌트 컨트롤로, 포커싱 처리를 지원하기 위해서는 다음과 같은 사항들을 고려해야만 한다. 첫 번째는 우선적으로 각 셀 내에 여러 컴포넌트의 수용이 가능한 테이블 형태를 위해 알맞은 자료구조의 선정이 필요하다. 여기서 수용 가능한 컴포넌트에는 사용자가 제어 가능한 컨트롤 컴포넌트들도 포함된다. 자료구조의 선정에는 모바일 GUI특성에 알맞게 자료에 대한 메모리 사용량도 함께 고려되어야 하므로, 가능한 각 셀에 삽입되는 컴포넌트들은 외부에서 구성되고 할당되는 것으로 하며, 테이블 컴포넌트는 단지 참조만할 뿐 셀 데이터의 자료 자체에 대한 별도의 메모리 공간은 사용하지 않는 것이 바람직하다.

두 번째는 테이블 셀에 포함된 컨트롤 컴포넌트들에 있어서 사용자 입력에 대한 포커스 처리 정책이 필요하다. 일반적인 포커스 처리는 사용자의 입력방식에 대해 의존적이다. 예를 들면, 입력장치가 키가 아닌 마우스나 터치스크린 형태로 사용자가 테이블의 어느 지점이나 입력할 수 있다면 포커스 처리 정책은 필요하지 않을 수도 있다. 하지만 키 입력의 경우 사용자가 제어를 원하는 특정 셀 내의 컴포넌트까지 순차적인 경로를 가지고 이동해야 하기 때문에 포커스의 제어 경로를 고려해야만 한다. 이것은 모바일 기기상의 대한 제약사항과 함께 고려되어야 하는 것으로, 본 연구에서는 주로 4개의 방향키에 의한 조작을 기본으로 고려하였다.

대부분의 모바일 기기(특히, 모바일 휴대전화 단말장치)의 디스플레이 장치가 작은 것과 적은 수의 조작키를 감안하여, 콘텐츠의 구조가 디스플레이의 너비를 고정으로 하고, 높이를 가변으로 하여 표현하는 경우가 많다. 이 경우 상하키의 배정은 스크롤에 대한 처리가 대부분으로 사용되고 있어서, 좌우키에 대해 포커스를 이동하도록 한다. 그림 1은 3x3 셀을 가지는 테이블에서 컨트롤 컴포넌트가 삽입된 경우의 테이블 구조에서 포커스 이동 경로를 나타낸 것이다. 포커스의 경로는 컨트롤 컴포넌트 1에서 시작하여 컴포넌트 9까지 이동 후에 테이블 자체 포커스로 전환하는 순환 흐름을 가진다.

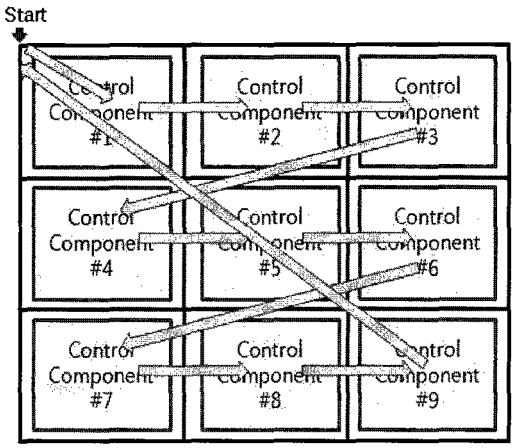


그림 1. 3x3 컨트롤 컴포넌트들을 가지는 테이블에서의 포커스 이동 경로
Fig. 1 The moving path of focus in the table with 3 by 3 control components

마지막으로 포커스 된 컴포넌트의 표현에 관한 고려가 필요하다. 이것은 컴포넌트의 전체 배경색의 변화나 테두리의 포커스 라인을 그려냄으로써 일반 컴포넌트와 구분이 가능하다. 셀 내에 배정된 컨트롤 컴포넌트들은 자신이 포커스 될 경우 이러한 구분된 표현을 위해 설계 시 별도의 공간이나 추가적인 속성 정보를 함께 고려되어야 한다. 그림 2는 테이블 내에 표현될 하나의 셀에 대한 구조로서 포커스 라인 외에도 추가적인 여백정보들을 함께 나타낸 것이다.

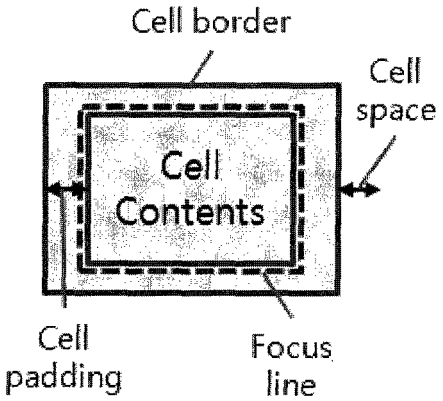


그림 2. 포커스 라인 및 여백정보를 나타내는 테이블 셀의 구조
Fig. 2 The table cell structure showing focus line and margin information

WFC에 추가될 테이블 컴포넌트의 설계에는 모바일 환경의 특성을 고려하여 작성되어야 한다. 이들 고려에는 작은 디스플레이, 적은 수의 키, 적은 메모리, 비교적 낮은 프로세스 성능 등이 여러 요소들이 있다. 여기서 가장 많은 영향을 미치는 것이 디스플레이에 해당하는 것으로, 디스플레이의 크기에 비해 많은 양의 콘텐츠의 출력을 어떻게 처리할 것인가 고민해야 한다. 일반적인 방법인 스크롤을 사용할 것인지, 스크롤을 사용한다면 메모리 소요량을 감안했을 때 어떤 크기까지 허용할 것인지, 디스플레이상 출력공간이 부족한 경우 콘텐츠 표현에 대한 잘림 현상을 감수하면서도 출력 처리를 허용할 것인지, 각 셀 내에 표현될 컴포넌트의 최소화 처리를 수행함으로써 보다 알맞은 표현을 유도할 것인지 등 여러 표현 방법들이 디스플레이의 크기에 의존성을 가지게 된다. 한편 기기에 할당될 키의 수에 대해서도 포커스 제어 처리 시 고려되어야 할 대상으로 이미 앞서 설명한 바대로 최소키의 사용으로 제어가 가능하도록 해야 한다. 메모리 사용량을 최소화하기 위한 테이블 표현과 셀 내의 데이터를 중복 처리하지 않고 이를 분리하여 참조 값만 가짐으로써 최소화 하도록 한다. 또한 저성능 프로세서에서 원활한 랜더링 속도를 위해 섬세함과 복잡성을 위주로 하기 보다는 모바일 환경에 적합한 단순하면서도 기본 기능을 지원하는 테이블의 구조가 되도록 한다.

이 외에도 사용자가 일일이 셀 내의 데이터에 대한 크기를 지정하지 않더라도 테이블 크기가 자동 계산되도록 하는 알고리즘의 적용이 필요하며, 리소스 관리에 대한 정책 또한 중요하다.

3.2 구현

이전 절에서 기술한 여러 가지 요구사항들을 고려하여 포커싱 가능한 테이블 컴포넌트를 구현하고 기존 WFC내에 추가하였다. 테이블 클래스는 기존의 FocusManager의 상속을 통하여 포커싱 처리에 대한 지원이 가능하도록 구현한다. 여기서 FocusManager 클래스는 각 자신의 하위에 가지는 모든 컨트롤 가능한 컴포넌트들의 포커스를 관리하는 역할을 한다.

사용자의 각 셀의 데이터들은 동적 객체 포인터 변수 배열을 사용하여 단순 참조로 처리하고, 데이터들에 대한 리소스들은 테이블의 외부에서 관리되도록 구현한다.

사용자 입력키에 대한 포커스 이동 처리는 이전 절에서 언급한 바와 같이 모바일 기기 특성을 고려하여 상하키는 스크롤 이동이, 좌우 키는 포커스 이동이 적용되도록 한다. 따라서 테이블은 너비에 대해서는 디스플레이 너비의 범위 내에서만 허용되며, 높이에 대해서는 스크롤바에 의해 메모리가 허용하는 데까지 제한 없이 가변적일 수 있다. 테이블 컴포넌트에서의 포커스 표현은 점선으로 된 사각형 표시로 처리한다.

테이블에 대한 디스플레이 출력공간이 부족한 경우, 각 셀의 데이터에 해당하는 컴포넌트의 최소 사이즈 정보를 획득한 후 표현에 알맞은 사이즈로 맞추어 출력되도록 하며, 이보다 작은 경우 잘림 현상을 허용하도록 한다. 그리고 셀의 여백관련 정보에 대한 공간조차도 없는 경우 테이블 자체는 출력되지 않는다. 사용자는 셀의 폭과 높이에 대해 별도의 지정이 가능하나, 미지정한 경우 이러한 셀들의 내용에 대해서는 테이블 출력 직전에 reshape처리를 통하여 테이블의 크기를 확정 짓게 된다.

3.3 구현의 예

구현의 검증으로 데스크톱 PC 에뮬레이터 상에서 몇 가지 테이블 컴포넌트를 활용한 데모 WFC 애플리케이션을 실행함으로써 확인한다. PC 에뮬레이터는 모바일

GUI 환경과 유사한 형태의 디스플레이 버퍼, 제한된 키 입력 등 제한된 리소스 상에서 WFC 및 이를 활용한 애플리케이션의 기능 구현을 검증하기 위해 Visual C++로 작성된 별도의 프로그램이다.

그림 3에서는 각 셀 내에 포커싱 가능한 컴포넌트인 CheckBox를 포함하는 테이블 컴포넌트 사용에 대한 예시 코드의 일부를 나타낸다. 테이블 생성과 함께, 해당 컴포넌트에 포커싱이 될 경우 이벤트를 처리하기 위해 별도의 이벤트 핸들러 등록 처리를 하게 된다.

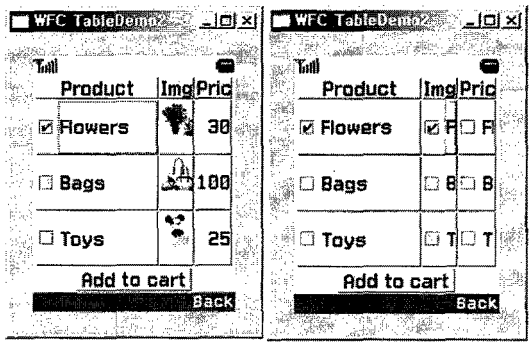
```
WObject* tableInfo[ROW_CNT][COL_CNT];
WObject* aa, fcCard1, table;
<중략>
aa=newActionAdaptor(obj);
implements_actionPerformed(aa,
                    actionPerformed_appShopping);
fcCard1=newActionAdaptor(obj);
implements_actionPerformed(fcCard1,
                    actionPerformed_card1FocusControl);
for(i=0;i<PRODUCT_CNT;i++)
{
    tableInfo[i+1][0]=newCheckBox(name1, checked);
    setBorderStyle(tableInfo[i+1][0], style);
    setBgColor(tableInfo[i+1][0], __WHITE);
    addActionListener(tableInfo[i+1][0], aa);
    <중략>
}
table=newTable("Demo2",ROW_CNT,COL_CNT,
                    (WObject***)tableInfo);
addActionListener(table, fcCard1);
useFocusBorder(table, TRUE);
setSize(table,SCREEN_WIDTH-2,
                MAIN_ZONE_HEIGHT-22);
setCellSpacing(table, 1);
setRowHeight_Table(table, 0, 18);
setColumnWidth_Table(table, 1, 30);
setColumnWidth_Table(table, 2, 32);
...
```

그림 3. 포커스를 가지는 테이블 컴포넌트 사용의 예제 코드 일부

Fig. 3 A part of sample code for the table component with a focus

그림 4(a)는 컨트롤 컴포넌트를 가지는 테이블 의 출력 화면이다. 3x3 셀을 가지는 테이블로, 각각 Label 컴포넌트와 아이콘 컴포넌트 외에 사용자의 선택이 가능한 CheckBox 컴포넌트를 포함한다. 현재 포커스는 점선으로 표기된 Flowers에 해당하는 CheckBox 컴포넌트를 가리킨다.

이 시점에서 사용자는 좌우키를 통하여 포커스를 Bags, Toys 등으로 이동할 수 있으며, 해당 컨트롤 컴포넌트에 포커싱이 되어 있는 경우 좌우키를 제외한 모든 키 이벤트는 지정된 컴포넌트에서 우선적으로 처리하게 된다. 그림 4(b)는 포커싱 이동과 셀의 공간이 셀 내용을 표현하기에 부족한 경우 잘림 현상에 대한 처리를 나타낸다. 테이블의 속성은 그림 4(a)와 동일하나 2, 3, 4 행의 각 첫 번째 셀의 내용을 행의 모든 셀에 동일하게 적용한 경우이다. 이 그림에서 현재 포커스는 제목표시 행을 제외한 첫 행의 두 번째 열에 위치한다.



(a) (b)

그림 4. 포커스를 가지는 테이블 컴포넌트의 출력 화면

Fig. 4 The output of table component with a focus

Ⅳ. 성능 비교

이 절에서는 포커싱 가능한 컴포넌트의 구현에 대한 성능 평가로 자바의 Swing에서 제공하는 JTable과 본 연구에서 제안하는 포커싱 가능 테이블 컴포넌트를 비교한다. 실제 모바일 기기상의 동일한 환경에서 검증이 이루어져야 하나 개발 환경의 설정에 따른 어려움으로 인해, 동일한 데스크톱 상에서 같은 콘텐츠 유형의 테이블 표현 시 출력에 소요되는 시간 측정으로 검증하였다. 서로 개발언어와 소프트웨어적인 실행환경이 다르고, 동일한 데스크톱 상에 동작한다고 하더라도 프로세스 점유의 차이를 보일 수 있어 정확한 수치의 측정이 어렵지만, 두 테이블 컴포넌트의 성능 분석에 있어서 테스트 편차를 줄이기 위해 테이블 컴포넌트 자체 출력에 소요되

는 시간만 측정하도록 신중히 코드를 선택하고, 테스트 시점에 타 프로세서의 CPU 점유를 최소화 하였다.

테스트 환경의 데스크톱 컴퓨터는 펜티엄 E2140 1.6GHz (DualCore) CPU, 2GB RAM, RADEON X550XT 128M 비디오 카드, 운영체제는 윈도우즈 XP(SP3)를 사용한다. 테스트에 사용된 자바 Swing의 버전은 JDK1.6.0 (JavaTM SE Development Kit 6 Update 13)이며, WFC은 제안된 테이블 컴포넌트를 포함한 버전이다[7].

먼저 테이블 컴포넌트의 출력에 따른 소요 시간을 측정하기 위해 시간 측정 코드를 별도로 작성한다. 본 연구에서 제시된 테이블 컴포넌트의 경우 테이블 자체를 그래픽 버퍼에 출력해내는 draw함수를 제공한다. 테이블 컴포넌트의 인스턴스를 획득한 후 draw함수를 호출하는 코드 전후에 시간 측정코드를 삽입하고, 이 호출을 반복하도록 설정하여 측정한다.

자바의 Swing에서의 테이블 컴포넌트에서 출력 소요 시간 측정을 위해서는, JTable의 출력시점에 대한 간접적인 접근을 확보하기 위해 JTable을 상속한 JTableEx 클래스를 생성한 후, paint 메소드를 오브라이드(override)하였다[8]. 그리고 이 paint 메소드 내에서 부모에 대한 paint 메소드를 호출하는 코드 전후에 시간 측정코드를 삽입하고, 이 호출을 반복하도록 설정하여 측정한다.

```
#define REPEAT_CNT 100000   /* 1 ~ 1,000,000 */

table=newTable("Table",ROW_CNT, COL_CNT,
                (WObject***)tableInfo);
<중략>
time_startval = clock();
for(i=0;i<REPEAT_CNT; i++)
    draw(table);
printf("Draw table time =%ld\n", clock() - time_startval);
...
```

그림 5. 제안된 테이블 컴포넌트의 출력 시간 측정을 위한 예제 코드의 일부
Fig. 5 A part of sample code for output time measurement of the suggested table component

그림 5와 그림 6은 앞서 설명한 내용과 같이 각각 성능 분석을 위한 두 테이블 컴포넌트의 사용 예제 코드의 일부이다. 그리고 그림 7은 이 예제 코드에 대한 출력된 테이블 실행화면을 나타낸다.

출력 성능 비교에 있어서는 일반적인 테스트와 같이 출력 반복횟수의 증가에 따른 소요시간을 측정하였다. 그림 8과 표 1은 반복횟수가 1에서 1,000,000회까지 증가할 때, WFC의 테이블 컴포넌트와 자바 Swing의 JTable의 출력 소요시간 차이를 나타낸 것이다.

```
public class JTableEx extends JTable {
    <중략>
    public void paint(Graphics g) {
        int i, REPEAT_CNT = 100000 /* 1 ~ 1,000,000 */

        long time_startval = System.currentTimeMillis();
        for(i = 0; i < REPEAT_CNT; i++)
            super.paint(g);
        System.out.println(System.currentTimeMillis()
                                - time_startval);
    }
}
...
```

그림 6. 자바 Swing의 JTable에 대한 출력 시간 측정을 위한 예제 코드의 일부
Fig. 6 A part of sample code for output time measurement of JTable

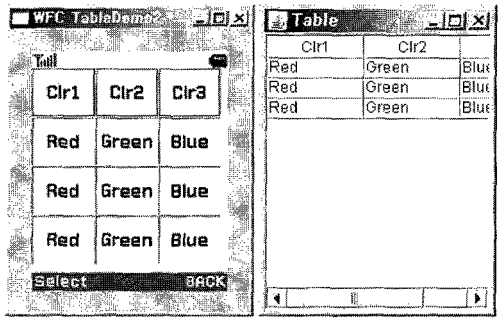


그림 7. 출력 시간 측정 예제에 사용된 테이블들의 출력 화면 (왼쪽이 제안된 테이블 컴포넌트, 오른쪽이 자바 Swing의 JTable)
Fig. 7 Output of tables in the examples for output time measurement(on the left, the supposed table component, on the right, Java Swing's JTable)

JTable의 경우 효과적인 표현을 위해 실행초기 시점과 실행완료 직후 시점에서 출력이 두 단계로 이루어지기 때문에 이를 구별하여 첫 번째 것을 FC(First View)로, 두 번째 것을 CV(Complete View)로 표기한다. 이 실험의 비교 결과에서 WFC의 테이블 컴포넌트가 1백만 번 반복 출력 시에 약 30%정도의 속도 향상을 보인다.

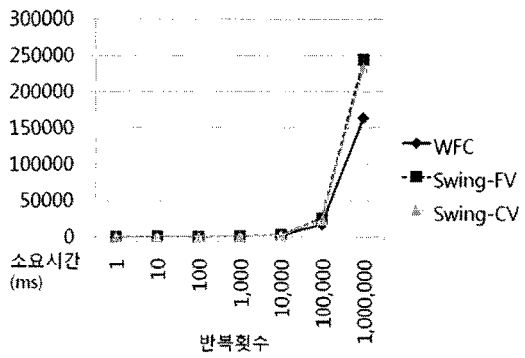


그림 8. WFC와 Swing의 테이블 컴포넌트 출력 소요시간 그래프 비교

Fig. 8 A comparison graph of output time for the table component in the WFC and Swing

표 1. WFC와 Swing의 테이블 컴포넌트 출력 소요시간 비교

Table. 1 A comparison of output time for the table component in the WFC and Swing

반복횟수	WFC	Swing -FV	Swing -CV
1	1	10	10
10	10	20	10
100	20	200	170
1,000	160	661	340
10,000	1,652	2,934	2,384
100,000	16,344	24,886	22,753
1,000,000	163,705	244,782	233,536

본 연구에서는 자바 SWT에서 제공하는 Table 위젯에 대한 비교분석은 제외되었다. 이것은 SWT의 Table 위젯이 자체 출력에 대한 메소드를 공용으로 제공하지 않을 뿐 아니라 상속을 통한 간접적인 메소드의 접근도 허용되지 않아서 출력 소요시간에 대한 정확한 측정이 어렵기 때문이다. 또한, GUI 재출력을 위한 메소드는 지원되나 반복 테스트 시에 이 메소드의 요청이 모두 수용되지는 않음으로 인해 이를 사용할 수 없었다. [9]에서는 Swing과 SWT에 대한 성능 평가를 기본 2D 그래픽 출력과 위젯의 성능 비교에 대해 소개하고 있다. 여기서는 Table 위젯의 경우 셀 데이터 설정과 스크롤 처리에 대한 소요 시간을 측정한 것으로 테블 자체 출력에 대한 성능은 알 수 없으며, 또한 각 테스트 환경에 따라 많은 편차를 보인다.

V. 결론 및 향후 연구

모바일 기기에서 콘텐츠의 표현은 여러 가지 제약들이 있으며, 이 중에서 테이블 표현에 관한 어려움으로 테이블 컴포넌트의 지원이 필요하다. 그리고 테이블 출력을 위한 단순한 표현 외에도, 상황에 알맞은 사용자의 입력 제어가 가능하도록 테이블 내에 포함된 콘텐츠에 대해 사용자의 입력 제어가 상황에 알맞게 가능하도록 함으로써 작은 디스플레이 장치에서 유용한 복합적인 표현이 가능하다. 이를 위해 본 연구에서는 포커싱이 가능한 테이블 컴포넌트를 제안하고 이를 구현하고자 할 경우 검토되어야 할 요구사항들을 제시하였다. 그리고 이런 여러 가지 사항들을 고려하여 WFC 환경 하에서 제안한 테이블 컴포넌트를 구현하고, 또한 검증을 위한 테스트로 PC 에뮬레이터 상에서 실행되는 것을 제시하였다. 이로써 WFC 개발 환경 하에서도 테이블 표현에 대한 확장을 가질 뿐만 아니라 사용자 입력 제어에 대한 보다 폭넓은 지원을 제공할 수 있게 된다.

구현된 WFC 테이블 컴포넌트는 모바일 기기에서 운용될 특성을 고려하여 기능에 있어서 많은 부분을 간소화함으로 출력속도에 대한 이점을 가진다. 이를 제시하기 위해 자바 Swing에서 제안된 테이블 컴포넌트와 출력에 대한 성능 비교를 제시하였다. 성능 비교 결과 본 연구에서 제안한 포커싱 가능 테이블 컴포넌트가 약 30% 정도 출력 성능이 빠른 것을 확인할 수 있었다.

현재는 WFC 테이블 컴포넌트를 포함한 WFC 패키지를 ARM7 임베디드 보드에 실장하여 이 컴포넌트에 대한 테스트 및 보완을 진행하고 있다.

참고문헌

[1] 백준상, 김유란, 이선영, 복일근, 황병철, "User Interface Characteristics of Converged Mobile Phone", The Human-Computer Interaction(HCI), 2004.

[2] 전종찬, 김정익, 강영만, 한순희, "효율적인 모바일 애플리케이션 개발을 위한 WFC 구현", 한국통신학회 하계종합학술발표논문집, 2009.

[3] Todd Sundsted, "Introduction to the AWT", JavaWorld.com, July. 1996, http://www.javaworld.com/javaworld/jw-07-1996/jw-07-awt.html.
[4] "Creating a GUI with JFC/Swing", http://java.sun.com/docs/books/tutorial/uiswing.
[5] Barry Feigenbaum, "SWT, Swing or AWT: Which is right for you?", Feb. 2006, http://www.ibm.com/developerworks/java/library/os-swingswt/index.html.
[6] Joshua Engel, "Programming for the Java Virtual Machine", Addison-Wesley Professional, July. 1996.
[7] "Java SE Downloads", http://java.sun.com/javase/downloads/index.jsp.
[8] 이안 F. 다윈, 유경희(역), "자바 프로그래밍 실전 테크닉 300", 한빛미디어, 2002
[9] Igor Križnar, "SWT Vs. Swing Performance Comparison", Oct. 2005, http://cosylib.cosylab.com/pub/CSS/DOC-SWT_Vs._Swing_Performance_Comparison.pdf.

저자소개

전종찬(Jong-chan Jun)

2001년 2월 : 여수대학교 전자계산학과 졸업
2009년 3월 : 전남대학교 디지털컨버전스 석사과정 재학중
※관심분야: 이동통신, 멀티미디어, 임베디드 시스템

김정익(Jeong-ik Kim)

2002년 2월 : 여수대학교 전자계산학과 졸업
2009년 3월 : 전남대학교 디지털컨버전스 석사과정 재학중
※관심분야: 이동통신, 멀티미디어, 임베디드 시스템

강영만(Young-man Kang)

1985년 2월 : 광운대학교 졸업
1987년 2월 : 광운대학교 전자계산학과 석사
2000년 2월 : 광운대학교 전자계산학과 박사
1992년~현재: 전남대학교 문화콘텐츠학부 교수
※관심분야: 이동통신, 멀티미디어

한순희(Soon-hee Han)

1983년 2월 : 경북대학교 전자공학과 졸업
1985년 2월 : 광운대학교 전자계산학과 석사
1993년 2월 : 광운대학교 전자계산학과 박사
1992년~현재: 전남대학교 문화콘텐츠학부 교수
※관심분야: 이동통신, 컴파일러, RFID