웹 기반의 Ad Hoc 리포팅을 위한 Fat Client를 갖는 리포팅 툴

(Reporting Tool using Fat Client for Web-based Ad Hoc Reporting)

최 지 웅 † 김 명 호 ††
(Jeewoong Choe) (Myungho Kim)

요 약 최근 들어 기업을 포함한 각 조직에서는 그들이 그 동안 축적한 데이타를 의미 있는 정보로써 활용하기 위해 데이타의 자유로운 포맷팅이 가능한 리포팅 툴을 의사 결정을 위한 데이타 분석 툴로서 확대 사용하고자 하는 요구가 증가하고 있다. 기존에는 하나의 동적 문서에 대해서 다수의 조회자가 발생하는 성격의 동적 문서들을 생성 및 배포하기 위하여 리포팅 툴을 사용해 왔다. 이러한 용도에 적합하도록 기존의 리포팅 툴은 서버 측에서 동적 문서의 생성을 담당하는 구조를 취하고 있다. 또한 서버 프로그램은 동적 문서의 미리 생성, 정기적 갱신 등을 담당하는 스케줄러와 반복된 생성을 피하기 위한 캐쉬 기능을 통하여 효율적 운영을 꾀하고 있다. 그러나 데이타 분석 용도의 동적 문서의 생성은 다수의 조회자를 고려한 동적 문서의 생성이 아니며 사용자가 다양한 값의 파라미터를 입력하는 방식으로 짧은 시간 간격으로 반복적인 동적 문서의 즉시 생성을 요구하는 특징이 있다. 이와 같이 리포팅 툴의 사용 범위 확대는 기존 리포팅 툴의 서버 측에 처리 부하를 증가시키고 있다. 본 논문에서는 제한된 리소스 환경에서도 대량의 데이타를 가공하여 동적 문서를 생성할 수 있는 리포트 뷰어를 통해 배포 목적이 아닌 데이타 분석 목적의 동적 문서의 경우 클라이언트 측에서 생성하도록 하여 서버의 부하를 분산시키고자 한다.

키워드 : 리포팅 툴, 팻 클라이언트, 비즈니스 인텔리전스(BI), 동적 문서

Abstract Recently, a variety of organizations including enterprises tend to try to use reporting tools as a data analysis tool for decision making support because reporting tools are capable of formatting data flexibly. Traditional reporting tools have thin-client structure in which all of dynamic documents are generated in the server side. This structure enables reporting tools to avoid repetitive process to generate dynamic documents, when many clients intend to access the same dynamic document. However, generating dynamic documents for data analysis doesn't consider a number of potential readers and increases requests to the server by making clients input various parameters at short intervals. In the structure of the traditional reporting tools, the increase of these requests leads to the increase of processing load in the server side. Thus, we present the reporting tool that can generate dynamic documents at the client side. This reporting tool has a processing mechanism to deal with a number of data despite the limited memory capacity of the client side.

Key words : reporting tool, fat client, business intelligence(BI), dynamic document

1. 서 론

리포팅 툴(Reporting Tool)[1,2]은 데이타베이스 혹은 CSV(Comma Separated Value) 형식의 텍스트 파일을 데이타 소스로 하여 이들의 데이타를 반영하는 동적 문서를 생성할 수 있는 툴이다. 리포팅 툴은 웹 상에서 PHP, JSP 등의 특정 스크립트로 작성한 서버 측 코드를 실행시킴으로써 동적 페이지를 생성하는 일반적인 방식과 달리 워드프로세서와 유사한 GUI 환경의 편집기를 통해 디자인한 템플릿 문서를 기반으로 동적 문서를 생성한다. 이러한 방식은 전산 인력의 도움 없이 신속한 동적 문서의 생성 및 배포를 가능하게 한다. 리포팅 툴은 스크린상에 디스플레이를 목적으로 하는

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음
† 학생회원 : 숭실대학교 컴퓨터학과
jwchoi@ss.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
kmh@computing.ssu.ac.kr
논문접수 : 2005년 4월 14일
심사완료 : 2006년 4월 7일

HTML 형식의 웹 페이지와는 달리 워드프로세서를 통해 생성한 문서처럼 특정 인쇄 용지 크기 내에서 정교하게 포맷팅한 문서를 웹 환경으로 출력할 목적으로 사용한다. 따라서 이러한 문서의 웹 환경을 통한 사용자 조회 및 인쇄를 제공하기 위해 리포팅 툴은 Microsoft의 ActiveX 컨트롤, Java Applet 등의 형태로 제작한 별도의 뷰어를 웹 페이지에 포함시켜 실행시키는 형태를 취하고 있다.

클라이언트/서버 구조의 응용 프로그램에서 클라이언트 측에 위치한 프로그램은 해당 응용에서 수행하는 역할의 비중에 따라 thin client와 fat client로 구분할 수 있다[3]. 예를 들어, 대량의 데이타를 기반으로 이미지를 생성하는 프로그램을 가정하면 가장 많은 리소스와 처리 시간이 소요되는 이미지 생성을 위한 모듈이 서버 측에 위치하고 클라이언트는 서버에서 처리한 결과를 전달 받아 화면에 이미지를 출력하는 기능 만을 담당한다면 이러한 구조에서의 클라이언트를 thin client라 하며 클라이언트가 직접 이미지를 생성하고 결과를 출력하는 역할을 담당하는 구조를 fat client라 한다.

기존의 리포팅 툴[4-6]은 동적 문서를 일괄적으로 다수의 사용자에게 배포하는 것을 목적으로 하기 때문에 전형적인 thin client 구조를 가지고 있다. 이 구조에서 서버는 동적 문서를 생성한 후 일정기간 동안 저장하며 동적 문서의 내용에 영향을 미치는 데이타의 갱신 주기에 따라 다시 생성한다. 이러한 구조는 동일한 동적 문서의 조회 요청에 대한 반복된 생성을 피하기 위함이다. 반면 클라이언트는 단순히 동적 문서를 조회하는 역할을 한다. 그러나 최근 들어 리포팅 툴은 단순한 리포팅 수단에서 의사 결정을 위한 분석 수단으로 그 역할이 확대되고 있다[7]. 그 이유는 동적 문서가 데이타를 사용자가 이해하기 용이하도록 포맷팅한 것이므로 데이타베이스에 직접 접근하여 테이블 형태로 데이타를 분석하는 것에 비해 직관적이며 SQL 질의문을 작성할 수 없는 사용자들도 분석을 수행할 수 있기 때문이다. 데이타 분석 행위 동안 사용자는 분석 데이타의 범위 결정을 위한 반복된 조건 입력을 수행하며 이것은 배포를 목적으로 하는 동적 문서와는 달리 서로 다른 데이타 셋을 반영하는 일회성의 반복적인 동적 문서의 실시간 생성을 요구한다. 이러한 요구들이 증가할 경우 thin client 구조에서의 서버는 동적 문서의 생성을 위한 처리의 부하가 증가한다. 기존의 thin client 구조를 갖는 리포팅 툴들은 서버의 개수를 늘림으로써 로드를 분산시킨다. 그러나 이러한 방법은 많은 리소스를 요구하며 추가적인 하드웨어와 소프트웨어를 위한 비용이 발생한다는 단점이 있다. Thin client 구조의 이러한 단점은 클라이언트 측에서 직접 동적 문서를 생성하는 fat client 구조를 취함으로써 극복할 수 있다. 그러나 fat client 구조는 실행 환경이 클라이언트에 위치하므로 사용할 수 있는 리소스에 제약이 따르기 때문에 대량의 데이타로 인해서 많은 페이지를 생성해야 하는 문서의 경우 동적 문서의 생성을 위한 처리가 완료되지 못한 채로 수행을 멈출 수 있다.

따라서 본 논문에서는 클라이언트 측의 제한된 리소스 환경에서도 효과적으로 문서 생성을 할 수 있는 구조와 처리 방식을 갖는 리포트 뷰어(fat client)를 제안한다. 이를 위해 문서 생성에 필요한 데이타를 클라이언트에서의 가용 메모리 크기를 고려하여 효과적으로 관리할 수 있는 구조를 제시한다. 또한 대량의 데이타로 인해서 긴 생성 시간을 가지면서 많은 페이지를 생성해야 하는 경우에도 문서 생성과 사용자의 페이지의 조회가 병행될 수 있는 문서 생성 방식도 제시한다.

본 논문의 구성은 다음과 같다. 제2장에서는 리포팅 툴의 구성 요소와 리포팅 툴이 동적 문서를 생성하는 과정을 기술한다. 또한 기존 리포팅 툴에 대해서 분석한다. 제3장에서는 동적 문서에 데이타를 적용하는 패턴의 종류를 분류하여 기술한다. 제4장에서는 리포트 뷰어의 데이타베이스로부터 가져온 데이타를 효과적으로 관리하기 위한 구조를 기술한다. 제5장에서는 동적 문서의 생성 시간에도 사용자의 조회가 동시에 이루어질 수 있도록 한 처리 방식을 기술한다. 제6장에서는 구현 내용을 기술하고 마지막으로 제7장에서는 결론 및 향후 연구에 대하여 기술한다.

2. 리포팅 툴

이 장에서는 리포팅 툴의 구성 요소와 리포팅 툴이 동적 문서를 생성하는 과정을 기술하며 기존 리포팅 툴에 대해서 분석한다.

2.1 리포팅 툴의 구성 요소와 동적 문서 생성 과정

리포팅 툴은 다음과 같은 요소들로 구성되어 있다.

- 포맷 편집기(Format Editor)
- 리포트 생성기(Report Generator)
- 리포트 뷰어(Report Viewer)

포맷 편집기는 동적 문서의 템플릿 파일인 포맷 파일을 디자인하기 위해 사용한다[1]. 일반적으로 포맷 편집기는 워드프로세서와 유사한 편집 환경을 가지고 있으며 리포팅 툴의 특성상 데이타베이스로 질의할 수 있는 기능과 결과 데이타 셋을 필드 단위로 문서의 필요한 곳에 맵핑할 수 있는 기능을 갖추고 있다.

리포트 생성기[1]는 포맷 파일을 해석하여 필요한 데이타베이스 데이타를 수집한 후 이를 포맷 파일에 정의되어 있는 규칙에 따라 위치를 정하여 배치하는 방식으로 동적 문서를 완성한다. 이때 사용자가 입력하는 파라

미터의 값에 따라 수집할 데이타베이스 데이타의 범위에 영향을 준다.

리포트 뷰어는 최종적으로 생성한 동적 문서를 조회할 수 있는 도구로써 웹 환경에서는 Microsoft의 ActiveX 컨트롤, Java Applet 등의 형태로 제작하여 웹 페이지에 포함시켜 실행시킨다. 따라서 리포팅 툴은 그림 1과 같은 순서로 동적 문서를 생성한다.

2.2 기존의 리포팅 툴

기존의 리포팅 툴[4,5,6,8]들을 앞에서 살펴본 리포팅 툴의 구성 요소 별로 갖는 특징에 대해서 기술한다.

2.2.1 포맷 에디터

포맷 에디터가 동적 문서를 디자인하기 위한 접근 방식은 밴드(band) 방식과 프리폼(free-form) 방식으로 나눌 수 있다. 대부분의 리포팅 툴[4-6]은 밴드 방식을 취하고 있으나 Microsoft의 SQL Server 2000 reporting services[8]은 프리폼 방식을 취하고 있다.

밴드 방식으로 완성한 포맷 파일은 밴드의 조합으로 볼 수 있다. 밴드 방식으로 포맷 파일을 디자인하는 작업 방식은 헤더(header), 풋터(footer), 디테일(detail) 등으로 명명된 각각의 고유한 역할과 행동 양식이 미리 정의된 밴드들을 필요에 따라 나열한 후 각각의 밴드 안에 필요한 텍스트 상자, 차트, 이미지, 표 등의 구체적인 표현 컴포넌트를 배치하는 방식이다.

프리폼 방식은 워드 프로세서와 같이 인쇄 용지에 대응하는 디자인 페이지 위에 바로 차트, 표 등의 구체적인 표현 컴포넌트를 배치하는 방식이다.

밴드 방식은 프리폼 방식에 비해 포맷 파일의 디자인 시간을 단축할 수 있는 장점이 있으나 포맷 파일의 디자인 유연성 측면에서는 프리폼 방식이 더 우수하다. 하지만 프리폼 방식은 각각의 표현 컴포넌트에 대해서 동적 문서로 변환될 시의 행동 규칙을 스크립트 방식의 프로그램 코드를 작성하여 정교하게 지정해 주어야 하므로 디자인 시에 디자이너에게 더 많은 책임을 요구한다. [8]의 경우 비주얼 스튜디오 닷넷 2003을 포맷 파일을 위한 디자인 인터페이스로 사용한다.

2.2.2 리포트 제너레이터

기존의 리포팅 툴이 제공하는 리포트 제너레이터의 동적 문서를 생성하기 위한 두 가지 주요 기능은 데이타 수집 기능과 수집한 데이타의 조작 기능이다. 데이타 수집 기능은 다양한 데이타 소스 즉, CVS 타입의 텍스트 파일, 이기종의 DBMS에 접근하여 데이타를 추출해 올 수 있는 기능이다. 수집한 데이타의 조작 기능은 일반적인 DBMS에서 제공하는 데이타 필터링, 집계, 산술 연산, 통계 등의 역할을 수행하는 함수와 동일하다. 데이타 소스가 DBMS일 경우 리포팅 툴의 입장에서는 가장 근원적인 데이타는 SQL 질의문의 결과 셋이다. 따라서 포맷 파일에 기술된 SQL 문이 리포팅 툴의 데이타 조작 기능을 최소한으로 사용하도록 정교하게 기술되어 있다면 동적 문서의 생성 시간을 절약할 수 있다. 그러나 하나의 동적 문서 안에 서로 다른 데이타 소스로부터 가져온 데이타들을 연관시켜 처리해야 할 경우가 발생할 수 있으므로 대부분의 리포팅 툴은 데이타 조작 기능을 필수적으로 내장하고 있다. 이 외에 리포팅 툴의 서버는 동적 문서를 미리 정해진 시간 또는 시간 간격으로 재생성 시키는 스케줄러와 생성한 문서를 저장하는 동적 페이지 리파지토리를 포함한다. 대부분의 리포팅 툴들은 리포트 제너레이터가 다양한 플랫폼에서 동작하도록 구현되어있다. 그러나 [8]의 경우 리파지토리를 위해 Micosoft사의 SQL Server 2000 DBMS를 필요로 하며 리포트 제너레이터의 실행 환경 또한 Windows 계열의 운영체제만을 지원하고 있다.

2.2.3 리포트 뷰어

동적 문서를 HTML 형식으로 생성할 경우 웹 브라우저가 리포트 뷰어의 역할을 수행한다. 또한 대부분의 리포팅 툴들은 HTML 형식으로 생성한 동적 문서를 Adobe 사의 PDF 형식, Microsoft 사의 Excel 형식, 이미지로 변환하는 기능을 가지고 있다. 그러나 각 기업 및 조직들 중 그들이 이미 사용하던 문서 양식을 리포팅 툴을 사용하여 그대로 웹 환경을 통해 조회 및 인쇄하는 방식으로 대체하고자 하는 요구 사항이 발생할 경

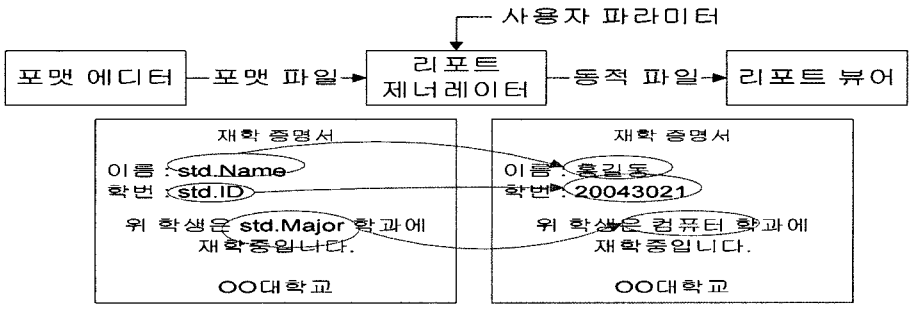

그림 1 동적 문서의 생성 과정

우, HTML 언어는 페이지 사이즈를 고려하여 표현 컴포넌트들을 정교하게 포맷팅하기 어려운 문서 양식이기 때문에 Microsoft의 ActiveX 컨트롤, Java의 Applet 같은 기술을 사용하여 구현한 별도의 뷰어를 제공한다. 이 경우 각 벤더별로 제작한 테이블, 텍스트 레이블, 차트 등의 컴포넌트를 사용하여 동적 문서를 구성한다. 특히 국내의 경우 문화적 차이로 인해서 서양 국가에 비해 복잡한 표가 삽입된 문서 양식을 선호하며 이를 인쇄하고자 하는 요구가 빈번하게 발생하기 때문에 별도의 뷰어가 웹 브라우저에 삽입된 형태가 많이 사용되고 있다.

3. fat client 구조의 리포팅 툴을 위한 요구 사항 분석

이 장에서는 리포팅 툴이 동적 문서에 데이타베이스의 데이타를 적용하는 패턴의 종류를 기술한다. 또한 이를 바탕으로 fat client 구조의 리포트 뷰어가 효과적으로 동작하기 위한 요구 사항을 도출한다.

3.1 리포팅 툴의 데이타베이스 데이타의 적용 패턴의 분류

이 절에서는 데이타베이스로부터 수집한 데이타가 동적 문서에 반영되는 패턴의 종류를 구분하여 기술한다.

3.1.1 단순 치환 타입

단순 치환 타입은 그림 1과 같이 포맷 파일에서 디자인한 문서의 레이아웃이 동적 문서에서도 동일하게 유지되며 단순히 질의 결과 셋의 실제 값을 1:1로 동일한 위치에 치환하여 완성하는 경우에 해당한다. 그림 1의 예에서는 사용자 파라미터가 학번을 나타내는 '20043021'가 입력된 결과이다. 다른 예로 사용자 파라미터가 학과 단위로 입력될 경우 복수개의 동적 문서를 얻을 수 있다.

3.1.2 리스트 타입

리스트 타입을 위한 포맷 파일에서의 디자인은 해당 리스트의 반복의 단위를 구성함으로써 이루어진다. 동적 문서에서는 이 반복의 단위가 포맷 파일에서 지정한 결과 필드들로 구성된 레코드의 개수만큼 반복하여 나타난다. 따라서 레코드의 개수에 따라서 페이지의 수가 결정된다. 또한 전체 레코드의 개수는 사용자 파라미터에 의해 그 범위가 결정된다.

그림 2는 어느 대학 도서관의 '학생별 대여도서 리스트'이다. 이러한 종류의 리스트를 마스터 디테일(Master-Detail) 리스트라고 한다. 이 리스트는 학생 정보를 의미하는 'std'로 명명된 질의 결과 셋이 마스터 역할을 하며 도서 정보를 의미하는 'Book'로 명명된 질의 결과 셋이 디테일 역할을 한다. 마스터와 디테일은 두 결과 셋에 공통으로 존재하는 필드들 중 사용자가 지정한 키(key) 필드로 연관되어 있다. 그림 2에서는 학번을 의미하는 'ID' 필드가 key 필드가 된다. 'Book' 질의 결과 셋에서의 'ID'필드는 디스플레이 하지 않을 뿐이다. 따라서 마스터 디테일 리스트를 포함하는 동적 문서를 생성하는 방식은 마스터의 한 레코드가 출력되고 해당 레코드의 키 값과 일치하는 값을 갖는 디테일 레코드들이 출력되는 방식으로 마스터의 레코드 개수만큼 반복한다. 이러한 마스터 디테일 리스트는 하나의 마스터 질의 결과 셋이 복수개의 디테일 질의 결과 셋을 갖는 형태와 하나의 마스터에 종속된 디테일 결과 셋이 그 자신의 디테일 결과 셋을 갖는 형태도 가능하다.

3.1.3 질의 결과 데이타 분석 타입

차트(chart)나 피봇 테이블(pivot table)을 사용해 질의 결과 셋의 데이타를 시각적으로 분석할 목적으로 작성한다. 그림 3은 '지점별 판매종목당 판매량'을 보여주는 피봇 테이블이다. 그림 3의 좌측은 포맷 파일의 디자인 결과이며 우측은 동적 문서에서의 결과 모습이다. 피봇 테이블의 행과 열에 할당한 각 필드들의 중복을 제거한 레코드의 개수에 따라 피봇 테이블 전체의 행과 열의 수가 결정된다. 따라서 이 경우는 피봇 테이블을 한 페이지의 정해진 영역 안에 담지 못할 경우 종(縱) 방향은 물론 횡(橫) 방향으로도 페이지가 추가된다.

3.2 fat client 방식의 리포트 뷰어를 위한 요구사항

리포팅 툴이 현실에서의 다양한 요구를 만족시키기 위해서는 이러한 각기 다른 패턴들을 하나의 동적 문서 안에서 함께 표현할 수 있어야 한다.

따라서 본 논문에서 제안하는 fat client 방식의 리포트 뷰어는 각각의 패턴을 처리하기 위한 그리고 클라이언트의 제한된 리소스 환경을 고려한 구조와 처리 방식

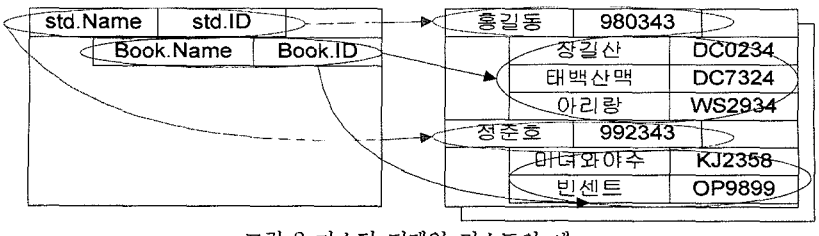

그림 2 마스터 디테일 리스트의 예

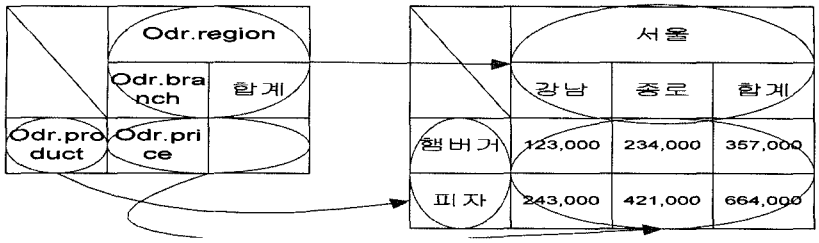

그림 3 피봇 테이블의 예

이 요구된다.

첫째, 다양한 패턴을 기술할 수 있는 포맷 파일의 구조가 필요하다.

둘째, 각기 고유한 패턴을 표현할 수 있는 독립된 표현 컴포넌트들이 필요하다.

셋째, 각기 다른 패턴을 갖는 표현 컴포넌트들이 데이타를 얻기 위해 질의 결과 셋에 다양한 방식으로 접근한다. 따라서 질의 결과 셋은 다양한 방식으로 접근할 수 있는 인터페이스를 제공해야 한다. 또한 제한된 클라이언트의 환경으로 인해 많은 레코드를 갖고 있는 결과 셋의 운영도 함께 고려해야 한다.

마지막으로 빠른 동적 문서의 조회를 위한 각각의 패턴에 적합한 처리 방식을 갖추어야 한다. 특히 리스트의 경우처럼 대량의 페이지를 생성해야 경우에도 사용자의 페이지 조회를 위한 대기 시간을 최소화 해야 한다.

4. fat client 방식에서의 리포트 뷰어의 구조

이 장에서는 fat client 방식에서의 리포트 뷰어의 구성 요소들인 포맷 파일, 표현 컴포넌트, 질의 결과 셋의 구조를 간략히 기술하며 마지막으로 질의 결과 셋과 표현 컴포넌트 사이의 관계에 대해서 중점적으로 기술한다.

4.1 포맷 파일의 구조

본 논문에서 제안하는 fat client 방식의 리포트 뷰어에서 동적 문서를 생성하기 위한 입력 파일인 포맷 파일의 형식으로 XML을 선택하였다. 포맷 파일은 자체 개발한 GUI 환경의 포맷 에디터[3]에서 작성할 수도 있으며 일반 텍스트 편집기에서도 작성 및 수정이 가능하다.

포맷 파일은 그림 3과 같은 구조를 가지고 있다. 최상위의 전역 문서 속성에는 문서 전체에 일관되게 적용되는 속성들이 위치한다. 사용자 파라미터 속성은 사용자 파라미터의 데이타 타입, 기본값 등의 속성으로 구성되어있으며 사용자 파라미터의 개수만큼 반복된다. 질의문 결과 셋 속성은 데이타베이스 연결 정보, 질의문, 질의문 이름 등의 내용을 포함하며 질의문 결과 셋의 개수만큼 반복된다. 페이지 속성은 페이지 전체에 일관되게 적용되는 속성들로 구성되어 있으며 포맷 에디터에서 디자인한 페이지 수만큼 반복된다. 표현 컴포넌트는 한

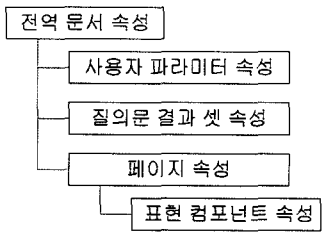

그림 4 포맷 파일의 구조

페이지에 포함되어 있는 표현 컴포넌트의 개수만큼 반복되며 각각의 표현 컴포넌트는 각각의 고유한 데이타베이스 데이타를 반영하는 패턴을 가지고 있다.

4.2 표현 컴포넌트의 구조

동적 문서의 페이지 위에 구체적 형상을 가지고 디스플레이 되는 표현 컴포넌트는 정적 컴포넌트와 동적 컴포넌트로 구분할 수 있다. 정적 컴포넌트는 포맷파일에서 디자인한 레이아웃과 데이타가 동적 문서에 그대로 반영되는 컴포넌트이다. 동적 컴포넌트는 데이타베이스로부터 추출한 데이타에 영향을 받는 컴포넌트이다. 그림 5의 기본 도형에 속하는 컴포넌트들은 정적 컴포넌트로서만 동작한다. 표, 텍스트 상자, 이미지 컴포넌트는 기본적으로 정적 컴포넌트로써 동작한다. 그러나 질의 결과 셋의 필드가 이들 컴포넌트에 할당될 경우 동적 컴포넌트로써 동작한다. 이때에도 레이아웃에는 변화가 일어나지 않는 단순 치환 타입으로 동작한다. 리스트 컴포넌트는 할당된 질의 결과 셋의 개수에 따라 마스터 디테일 타입을 포함한 리스트 타입으로 동작하며 차트와 피봇 테이블 컴포넌트는 질의 결과 데이타 분석 타입으로 동작한다.

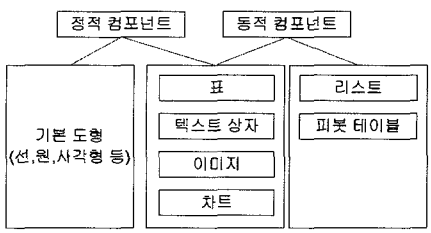

그림 5 표현 컴포넌트의 구조

4.3 질의 결과 셋의 구조

질의 결과 셋은 하나의 클래스로 맵핑되며 'RVQuery'라는 이름으로 명명하였다. RVQuery 클래스의 객체는 포맷 에디터에서 생성한 질의 결과 셋의 개수만큼 리포트 제너레이터에서 인스턴스화 된다.

RVQuery 클래스의 주요 멤버 변수는 질의를 요청하기 위한 데이타베이스의 연결정보, 질의문(Query), 결과 셋에 대한 레코드 수, 필드 이름, 필드 타입 등의 메타데이타, 이차원 배열 타입의 질의 결과 데이타를 저장할 장소 등으로 구성되어 있다.

RVQuery 클래스의 메소드들의 역할은 이차원 배열 형태로 저장한 질의 결과 데이타 셋에 대한 메타데이타와 데이타를 동적 표현 컴포넌트들이 사용할 수 있도록 하는 것이다.

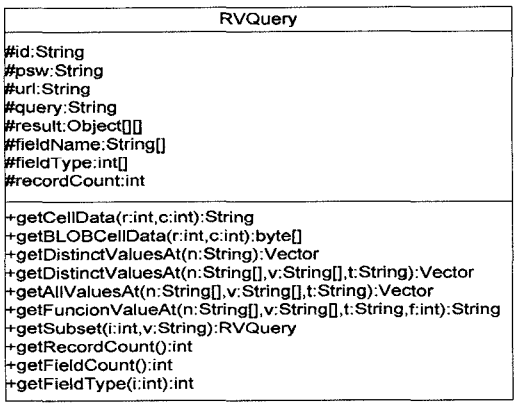

그림 6 RVQuery 클래스의 주요 멤버 변수와 메소드

4.4 RVQuery와 동적 표현 컴포넌트

앞서 기술한 세가지 타입의 동적 컴포넌트들은 각각의 타입에 따라 RVQuery의 서로 다른 메소드를 사용하며 RVQuery 또한 각각에 대비하여 내부적으로 다른 처리 방식을 가지고 있다.

4.4.1 단순 치환 타입

그림 7과 같이 단순 치환 타입의 동적 컴포넌트로 구성된 하나의 포맷 파일에 대응하는 하나의 동적 문서 인스턴스를 생성하기 위해 필요한 질의 결과 데이타의 의미상의 단위는 하나의 레코드이다. 그러나 데이타를 추출하는 단위는 그 레코드에 포함된 필드 단위이다. 따라서 단순 치환 타입의 동적 컴포넌트들은 행 인덱스와 열 인덱스를 사용하여 이차원 배열에 저장된 한 배열요소에 접근할 수 있는 RVQuery의 getCellData 메소드를 호출하여 문자 데이타를 추출하며 getBLOBCellData 메소드를 호출하여 이미지 데이타를 추출한다.

4.4.2 리스트 타입

리스트 타입의 동적 컴포넌트는 리스트의 한 행을 구성하기 위해서 질의 결과 셋의 한 레코드의 데이타를 요구한다. 따라서 리스트 타입의 동적 컴포넌트는 레코드의 개수만큼 반복하며 단순 치환 타입의 동적 컴포넌트와 마찬가지로 추출하고자 하는 데이타 타입에 따라 getCellData와 getBLOBCellData를 사용하여 리스트를 완성한다. 두 개 이상의 질의 결과 셋이 필요한 마스터 디테일 리스트는 마스터 리스트의 경우 질의 결과 셋이 한 개 할당된 단순한 리스트와 마찬가지로 자신에게 할당된 질의 결과 셋의 레코드에서 순차적으로 데이타를 추출할 수 있으나 디테일 리스트의 경우 자신의 마스터 리스트의 키 값에 따라 범위가 한정되므로 이 키 값을 파라미터로 하여 자신에게 할당된 RVQuery 객체의 getSubset 메소드를 호출하여 해당 범위의 레코드 셋을 가진 RVQuery 객체를 얻은 후 그 안에 포함된 레코드에서 데이타를 추출한 후 제거하는 방식을 사용한다.

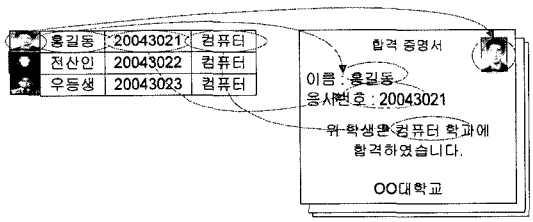

그림 7 단순 치환 타입 동적 컴포넌트의 RVQuery의 사용

4.4.3 질의 결과 데이타 분석 타입

질의 결과 데이타 분석 타입의 대표적인 동적 컴포넌트는 피봇 테이블이다. 피봇 테이블은 분석하고자 하는 데이타의 기준이 되는 데이타가 동일 테이블의 행, 열의 헤더 부분과 분석하고자 하는 데이타가 위치할 테이블의 내용 부분으로 구성되어 있다. 각각의 데이타를 얻기 위해 사용하는 RVQuery 객체의 메소드에 차이가 있다. 그림 3을 예로 들어 설명하면 행 헤더는 'product' 필드가 할당되었으며 열 헤더는 'region'과 'branch'의 두 필드가 할당되었다. 마지막으로 분석하고자 하는 데이타는 'price' 필드이다. 우선 테이블의 헤더 부분은 헤더에 할당된 필드의 중복을 제거한 모든 값의 종류를 얻어야 한다. 이를 위해 할당된 필드를 파라미터로 지정하여 getDistinctValuesAt 메소드를 호출한다. 이 결과 그림 3의 행 헤더는 '햄버거'와 '피자'라는 값을 얻었으며 열 헤더는 '서울'이라는 값을 얻었다. 'branch'필드는 'region'필드에 종속적이므로 '서울'이라는 값을 갖는 레코드 중에서 중복을 제거한 모든 값의 종류를 얻기 위해 getDistinctValuesAt('region', '서울', 'branch') 형식으로 메소드를 호출하여 '강남'과 '종로'라는 값을 얻었다. 테이블의 내용 부분의 각 셀들의 값은 getFunction-

ValueAt 메소드를 사용하여 얻는다. 이 경우 그림 3의 3행 2열의 '123,000'라는 값은 'region'필드 값이 '서울'이고 'branch'필드 값이 '강남'이고 'product'필드 값이 '햄버거'인 레코드들의 'price'필드 값들의 합을 의미한다.

4.4.4 RVQuery의 메소드 처리 방식

RVQuery는 클라이언트의 제한적인 메모리 환경을 고려한 두 가지 동작 방식을 가지고 있다.

첫째, 이차원 배열 형태로 질의 결과 데이타를 저장하고 있는 result 멤버 변수가 한 번에 저장할 수 있는 레코드의 개수를 제한한다. 한 번에 저장할 수 있는 레코드의 개수는 포맷 파일의 '질의문 결과 셋 속성' 부분의 해당 RVQuery의 속성으로 지정되어있다. 따라서 단순 치환 타입과 리스트 타입의 컴포넌트에서 호출하는 getCellData와 getBLOBCellData 메소드는 접근하고자 하는 레코드가 result 변수에 없다면 포맷 파일에 지정된 레코드 개수 단위로 필요한 레코드가 포함된 결과 데이타를 가져와 result 변수에 저장한 후 동적 컴포넌트에 제공한다.

둘째, 질의 결과 데이타 분석 타입의 컴포넌트가 호출하는 RVQuery의 메소드들의 반환값의 성질은 질의 결과 데이타의 모든 레코드를 필요로 한다는 점이다. 따라서 이러한 메소드의 처리 방식은 첫 번째 방식과 같이 result 변수를 참조하는 것이 아니라 해당 RVQuery를 위한 질의문을 저장하고 있는 query 멤버 변수의 질의문을 메소드 내부에서 일시적으로 수정한 후 그 질의문을 질의한 결과를 메소드의 반환값으로 한다. 그림 3의 피봇 테이블은 다음과 같이 처리한 결과이다. 이름이 'Odr'인 RVQuery의 query 멤버 변수에 저장된 질의문을 S_{Odr} 라고 하면 getDistinctValuesAt('product') 메소드는 질의문 SELECT DISTINCT product FROM (S_{Odr});의 결과를 반환하며 getDistinctValuesAt('region', '서울', 'branch') 메소드는 SELECT DISTINCT branch FROM (S_{Odr}) WHERE region='서울';의 결과를 반환한 것이며 3행 2열의 '123,000'라는 값은 getFunctionValueAt 메소드를 호출하여 질의문 SELECT SUM(price) FROM(S_{Odr}) WHERE region = '서울' AND branch = '강남' AND product = '햄버거' ; 를 수행한 결과의 반환값이다.

이 장에서는 fat client 방식의 리포트 뷰어에서 데이타베이스 데이타를 효과적으로 관리하기 위한 구조를 살펴보았다. RVQuery의 내부 구조는 클라이언트의 제한된 메모리 환경에서도 대량의 질의 결과 셋을 다룰 수 있도록 설계하였다. RVQuery는 뷰어 내의 모든 동적 컴포넌트에 데이타베이스로부터 가져온 데이타를 전달할 수 있도록 설계하였다. 따라서 복수 개의 동적 컴포넌트들이 하나의 RVQuery 객체를 공유할 수 있으며 이러한 구조는 중복된 질의 결과 데이타로 인한 메모리 낭비를 방지할 수 있다. 리포트 뷰어의 전체적인 구조에서 볼 때 RVQuery 객체는 데이타베이스 데이타를 가져올 수 있는 유일한 객체이며 뷰어 내의 모든 동적 컴포넌트들은 RVQuery 객체의 메소드를 호출함으로써 그 데이타를 제공받는다. 이러한 구조로 인해 각 동적 컴포넌트들의 구현이 데이타베이스와 관련된 코드로부터 독립적일 수 있게 한다. 마지막으로 각 RVQuery 객체는 서로 다른 데이타베이스에 대하며 질의한 결과 셋을 가질 수 있다. 따라서 이처럼 서로 다른 데이타 소스에 존재하는 데이타라 하더라도 한 동적 문서 안에 반영할 수 있다.

5. fat client 방식에서의 동적 문서의 생성

이 장에서는 리포트 뷰어에서 동적 문서를 생성하기 위한 절차를 기술한다. 리포트 뷰어는 다음과 같은 처리 과정을 거쳐 동적 문서의 각 페이지들을 생성한다. 첫 번째 단계는 준비 단계로서 포맷 파일로부터 포맷 에디터에서 디자인한 페이지와 같은 상태의 원본 페이지를 복원한다. 두 번째 단계는 동적 페이지 생성을 위한 전처리 단계로서 준비 단계에서 복원한 원본 페이지들이 각각 몇 페이지의 동적 페이지를 생성하게 되는지를 계산한다. 마지막 단계는 동적 페이지 생성 단계로서 사용자가 화면상에 디스플레이 또는 인쇄를 목적으로 요청한 동적 페이지를 생성한다.

4.1 준비 단계

리포트 뷰어의 실행은 ReportViewer 객체에서 시작한다. ReportViewer 객체는XML 타입의 포맷 파일로부터 원본 페이지를 복원하는 역할을 RVXML 객체에게 위임한다. RVXML 객체는 다음과 같은 순서로 원본 페이지를 생성한다.

① RVParameter 객체를 생성한다.
② RVQuery 객체를 생성한다.
③ RVPage 객체를 생성한다.
④ RVPage 객체에 포함되어있는 RVObject 객체를 생성한다.

RVParameter 객체는 포맷 파일의 '사용자 파라미터 속성' 부분과 대응하며 사용자가 뷰어에서 직접 입력한 값이 있을 경우 이 값으로 변경된다. RVQuery 객체는 포맷 파일의 '질의문 결과 셋 속성' 부분과 대응하며 RVParameter 값에 따라 질의 결과 셋의 범위가 결정된다. RVPage 객체는 원본 페이지를 의미하며 포맷 파일의 '페이지 속성' 부분과 대응한다. 마지막으로 각 페이지에 포함된 표현 컴포넌트들을 생성함으로써 원본 페이지들의 복원을 완료한다.

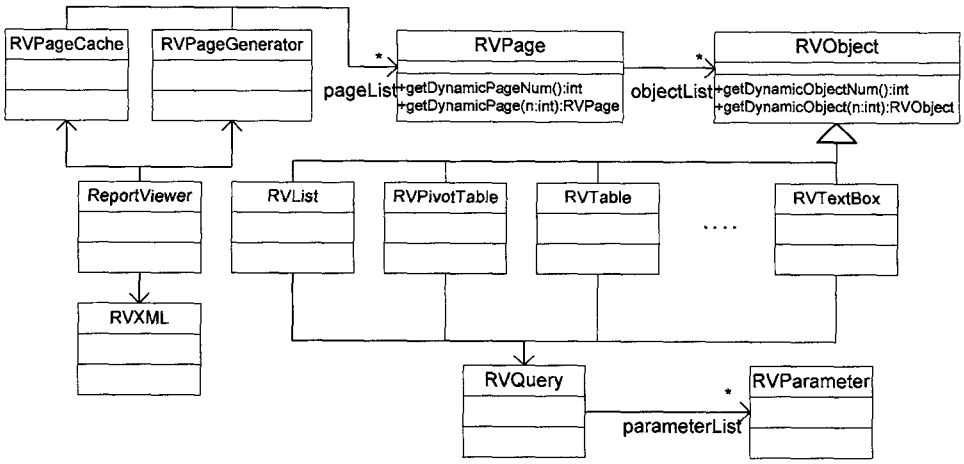

그림 8 리포트 뷰어의 클래스 다이어그램

5.2 전처리 단계

전처리 단계를 통해 두 가지 메타데이타를 얻는다. 첫째는 각각의 원본 페이지가 몇 페이지의 동적 페이지를 생성하는가에 대한 정보이다. 두 번째는 생성될 각 동적 페이지가 구체적으로 질의 결과 셋의 어떤 데이타를 포함하게 되는가에 대한 것이다. 이러한 메타데이타를 통해 사용자가 임의의 페이지를 요구할 경우 빠르게 동적 페이지를 생성할 수 있다. 그러나 전처리 단계는 질의 결과 셋의 레코드 개수 혹은 동적 컴포넌트의 종류에 따라 처리 시간이 가변적이다. 따라서 경우에 따라 장시간의 사용자 페이지 조회를 위한 대기 시간이 발생할 수 있다. 이러한 경우를 방지하기 위해 전처리 단계는 별도의 쓰레드를 발생시켜 처리한다. 이러한 처리 방식으로 인해 사용자는 전처리 단계가 완료되기 이전에도 생성된 메타데이타 범위 안에서 동적 페이지를 조회할 수 있다.

5.2.1 동적 페이지 개수의 산출을 위한 처리

ReportViewer 객체가 RVPageGenerator의 start 메소드를 호출함으로써 전처리 단계를 위한 쓰레드가 생성된다. RVPageGenerator는 원본 페이지가 몇 페이지의 동적 페이지를 생성하는가를 묻기 위해 원본 페이지를 의미하는 RVPage의 *getDynamicPageNum* 메소드를 호출한다. getDynamicPageNum 메소드의 호출은 연쇄적으로 해당 원본 페이지가 담고 있는 표현 컴포넌트들의 getDynamicObjectNum 메소드를 호출한다. 표현 컴포넌트의 getDynamicObjectNum 메소드의 리턴값은 자신이 몇 페이지의 동적 페이지에 나타나는지를 의미한다. 따라서 getDynamicPageNum 메소드의 리턴값 즉 하나의 원본 페이지에서 생성될 동적 페이지 개수는 각 표현 컴포넌트들의 getDynamicObjectNum 메소드의 리턴값들의 최대값이다. 이 과정은 원본 페이지 개수만큼 반복되며 각 원본 페이지의 *getDynamicPage-Num* 메소드의 리턴값들의 합이 동적 문서의 최종 페이지 수이다.

표현 컴포넌트의 종류에 따른 getDynamicPageNum 메소드의 처리 방식을 살펴보면 단순 치환 타입의 컴포넌트는 맵핑된 질의 결과 셋의 레코드의 개수가 get-DynamicPageNum 메소드의 리턴값이다. 질의 결과 데이타 분석 타입의 컴포넌트인 피봇 테이블은 자신이 차지할 수 있는 한 페이지 내에서의 영역을 고려하여 테이블 행과 열의 개수만을 계산하면 필요로 하는 페이지 수를 산출할 수 있다. 리스트 타입은 리스트의 한 행이

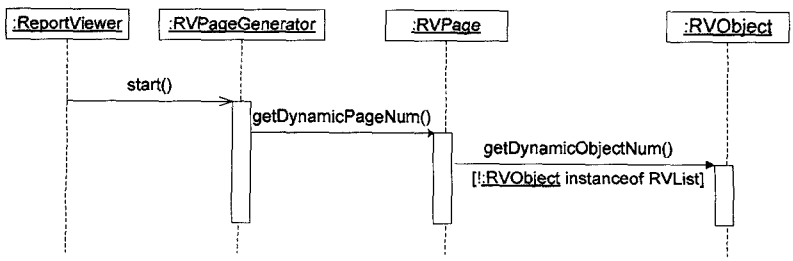

그림 9 동적 페이지 개수를 계산하기 위한 시퀀스 다이어그램

질의 결과 셋의 한 레코드에 맵핑된다. 따라서 리스트 컴포넌트가 페이지 수를 산출하는 기본적 방법은 한 페이지 내에서 차지할 수 있는 높이와 리스트 행들의 높이를 고려하여 필요한 페이지 수를 계산하는 것이다. 그러나 마스터 디테일 리스트는 하나의 마스터 행에 종속적인 디테일 행들의 개수를 통해 디테일 행들의 높이를 계산할 수 있으므로 해당 디테일 레코드 셋을 얻기 위해 질의문을 한 번 수행해야 한다. 따라서 마스터 행의 개수에 비례하여 처리 시간이 길어지므로 리스트 타입의 동적 컴포넌트는 getDynamicPageNum 메소드 수행을 별도의 쓰레드를 생성하여 처리한다. 이러한 처리 방식은 리스트의 getDynamicPageNum 메소드 수행이 완료되기 이전에도 계산한 페이지 범위 내에서 리스트 컴포넌트를 제공하기 위함이다.

5.2.2 동적 페이지의 빠른 생성을 돕기 위한 전처리

표현 컴포넌트의 getDynamicObjectNum 메소드 수행은 동적 페이지의 개수를 세면서 향후 특정 동적 페이지의 생성 요청이 발생할 경우 그 페이지가 필요로 하는 데이타를 빠르게 가져오기 위한 처리를 동반한다. 이러한 처리는 표현 컴포넌트의 종류에 따라 차이가 있다.단순 치환 타입의 컴포넌트는 별도의 처리가 필요 없다. 같은 원본 페이지를 기반으로 생성되는 동적 페이지의 개수와 그에 대응하는 질의 결과 데이타의 레코드 개수가 같기 때문이다. 질의 결과 데이타 분석 타입의 컴포넌트인 피봇 테이블은 자신의 행과 열의 개수를 결정하기 위해 사용한 테이블의 헤더 부분의 데이타들을 저장함으로써 동적 페이지의 생성 시에는 이 값을 기준으로 테이블의 내용 부분에 해당하는 데이타 만을 필요로 한다. 리스트 타입의 컴포넌트는 리스트의 각 행에 대한 정보를 사용한다. 행 정보란 각 행이 포함될 페이지 번호, 그 페이지 안에서의 시작 위치, 해당 행에 대응하는 질의 결과 레코드 인덱스이다. 마스터 디테일 리스트는 최상위의 마스터 행의 정보만을 저장한다. 그리고 질의 결과 셋의 레코드가 많을 경우를 대비하여 일정 간격의 행 정보만을 저장한다. 행들의 간격은 포맷 파일에 지정된 행 정보를 저장할 장소의 크기에 따라 조정된다. 리스트를 포함하는 동적 페이지의 생성은 행 정보를 저장하는 장소에서 목적 페이지 번호보다 작은 것들 중에서 가장 큰 페이지에 포함될 행 정보를 선택하여 그 행부터 생성을 시작하여 요구한 페이지의 리스트까지 생성한다.

5.3 동적 페이지 생성 단계

동적 페이지의 생성에 대한 이벤트가 발생하면 ReportViewer 객체는 RVPageCache 객체에게 페이지를 요청한다. RVPageCache 객체는 캐쉬 안에 해당 페이지가 존재하면 이것을 전달하며 그렇지 않을 경우 원본 페이지 객체인 RVPage 객체에게 새로운 페이지의 생성을 요청한다. RVPage 객체는 새로운 동적 페이지를 생성하기 위해 해당 페이지의 구성 요소인 RVObject 객체들에게 그 페이지를 위한 표현 컴포넌트들을 요구한다. 각 표현 컴포넌트들은 전처리 단계에서 생성한 메타데이타를 이용해 표현 컴포넌트를 생성한다. 이러한 과정을 거쳐 생성한 동적 페이지는 페이지 캐쉬 안에 새로운 멤버로서 저장된 후 ReportViewer 객체에게 전달된다.

이 장에서는 fat client 방식의 리포트 뷰어에서 동적 페이지를 생성하기 위한 처리과정을 살펴 보았다. 본 논문에서의 처리 과정은 표현 컴포넌트들의 타입을 고려함은 물론 시스템의 확장성을 고려하여 설계하였다. 즉, 각 표현 컴포넌트들의 특성을 고려한 처리는 각 표현 컴포넌트들 내부에 캡슐화 되어 있으며 페이지 컴포넌트와 표현 컴포넌트들은 전처리 단계와 페이지 생성 단계에서 모두 동일한 인터페이스로 통신한다는 것이다. 이것은 새로운 표현 컴포넌트가 필요할 경우 이들간의 인터페이스 규칙을 준수하여 구현 후 기존 시스템에 쉽게 추가할 수 있음을 의미한다.

6. 시스템 구현

본 시스템의 모든 구성 요소는 Java를 사용하여 개발하였다. 본 시스템은 애플릿 형태로 제작한 클라이언트 측의 리포트 뷰어와 서버 측의 서블릿 형태로 제작한 QueryServlet, XMLServlet 그리고 EJB 형태로 제작한 QueryEJB로 구성되어 있다. 서버 측 구성 요소의 역할

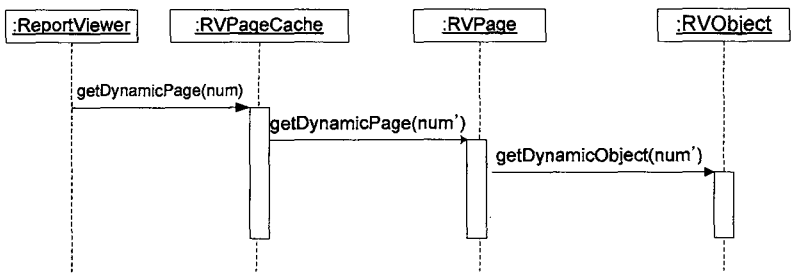

그림 10 동적 페이지를 생성하는 시퀀스 다이어그램

은 다음과 같다. QueryServlet은 클라이언트의 질의문을 QueryEJB에 전달하며 그 결과값을 리포트 뷰어에 되돌려 준다. XMLServlet은 리포트 뷰어의 요청을 받아 서버 측에 저장되어있는 XML 타입의 포맷 파일을 클라이언트에 전해준다. QueryEJB의 기본적 역할은 다양한 종류의 DBMS에 직접 질의문을 요청하는 역할이며 추가적으로 DBMS와의 연결을 감소시키기 위해 질의문 결과를 일시 저장하는 역할도 담당하고 있다.

본 시스템의 실행 순서는 리포트 뷰어에서 필요한 포맷 파일을 XMLServlet을 통해 다운로드 받은 후 필요한 데이타는 QueryServlet을 통해 요청하여 그 결과값을 이용해 동적 문서를 생성한다.

그림 12는 이 논문에서 구현한 리포팅 툴의 실행 화면이다. 동적 문서는 가상의 전자 제품 판매 업체의 지점별 판매 내용을 분석하는 내용이다. 이 동적 문서는 지점별 구체적인 판매 내역을 표현하기 위하여 마스터 디테일 리스트를 사용했으며 지점별 각 제품의 판매 총액을 하나의 피봇 테이블로 표현하였다. 그림 12의 좌측 그림은 해당 동적 문서를 위한 XML 형식의 포맷 파일이다. 중앙에 위치한 그림은 동적 문서의 결과 화면의 첫 페이지이며 우측 그림은 마지막 페이지이다. 우측 그림의 대화 상자는 동적 문서에 포함시킬 데이타의 범위를 결정하기 위한 조건 입력 대화 상자이다. 해당 문서에서의 조건은 '판매 연도'를 의미한다. 따라서 사용자가 입력한 판매 연도에 해당하는 숫자에 따라서 동적 문서의 내용 또한 달라진다. 결과 화면은 2005년 판매 데이타를 분석한 문서이며 총 96페이지가 생성 되었으며 결과 화면 상단이 마스터 디테일 리스트 이며 결과 화면 하단이 피봇 테이블이다. 피봇 테이블은 마스터 디테일 리스트에 비해 적은 페이지에 표현되므로 그림 12의 우측 그림의 경우 결과 화면 상단에 마스터 디테일 리스트 만이 표현 되어 있다.

7. 결 론

이 논문에서는 thin client 방식을 갖는 기존 리포팅 툴에서 다수의 사용자들이 각각의 개별적인 목적으로 데이타 분석을 수행하는 과정에서 즉각적인 동적 문서의 생성 요구가 증가할 경우 서버의 처리량 증가에 따른 성능 저하 문제를 해결하기 위하여 클라이언트에서 직접 동적 문서를 생성하는 fat client 방식의 리포팅 툴을 제시하고 구현하였다. 특히, 클라이언트의 제한된 실행 환경에서도 대량의 데이타를 가공 및 처리할 수 있는 구조와 처리 방식을 갖는 리포트 뷰어를 중심으로 제시하였다.

리포트 뷰어는 질의 결과 데이타를 관리하는 질의 결과 셋 컴포넌트를 따로 둠으로써 질의 결과 데이타를 실질적으로 사용하는 표현 컴포넌트들로부터 질의문과 관련된 코드들을 제거하였다. 이러한 구조는 각각의 표

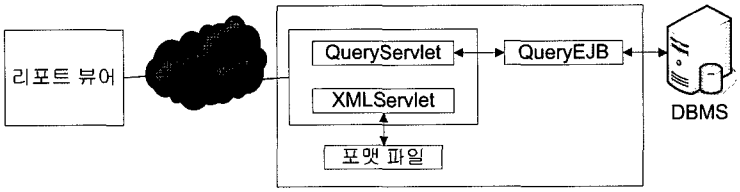

그림 11 시스템 구성도

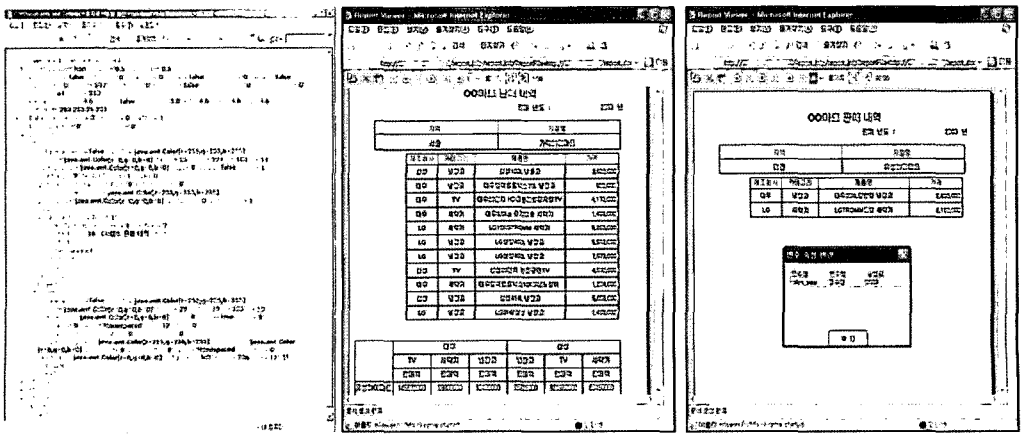

그림 12 포맷 파일과 리포트 뷰어의 실행 화면

현 컴포넌트들이 그들이 필요로 하는 데이타를 직접 서버에 요청한 후 그 결과를 사용할 경우 클라이언트 측에 질의 결과 데이타의 중복이 발생하므로 리소스를 낭비하게 된다. 따라서 이러한 문제를 방지하기 위하여 각각의 표현 컴포넌트들이 필요로 하는 데이타를 메소드 호출 방식으로 해당 질의 결과 셋 컴포넌트에 요청하며 질의 결과 셋 컴포넌트는 필요에 따라 서버에 데이타를 요청하는 방식을 취하도록 하였다. 또한 각 표현 컴포넌트들과 질의 결과 셋 컴포넌트는 동일한 인터페이스를 가지고 통신하므로 인터페이스 규칙만 준수할 경우 필요에 따라 표현 컴포넌트들을 용이하게 추가할 수 있는 구조로 설계 하였다.

리포트 뷰어는 전처리 단계와 동적 페이지 생성 단계의 두 단계를 거쳐 동적 문서를 생성하도록 하였다. 전처리 단계는 동적 문서의 전체 페이지 수를 계산하면서 각 페이지에 대한 메타데이타를 생성한다. 동적 페이지 생성 단계는 사용자가 요청한 페이지 만을 전처리 단계에서 생성한 메타데이타를 사용하여 생성한다. 전처리 단계와 동적 페이지 생성 단계는 동시에 처리된다. 즉, 전처리 단계가 완료하기 이전이라 하더라도 현재 전처리 단계에서 생성한 각 페이지에 대한 메타데이타 범위 안에서 사용자의 요청이 있을 경우 페이지를 생성할 수 있도록 하였다. 이러한 처리 방식은 대량의 데이타로 인한 대량의 페이지를 생성하게 되는 동적 문서라 하더라도 사용자에게 가능한 범위 안에서 빠르게 동적 문서를 제공하기 위함이며 요청이 있는 페이지 단위로 생성하는 것은 클라이언트의 제한된 메모리 환경을 고려한 것이다.

리포트 뷰어는 Java 기반의 애플릿으로 구현하였으므로 웹 브라우저 안에서 실행 가능하며 자동으로 다운로드 되어 실행된다. 따라서 다양한 플랫폼에서 동일하게 동작하며 사용자로 하여금 별도의 설치 과정을 요구하지 않는다.

향후에는 기존의 연결 가능한 관계형 데이타베이스와 CSV 형식의 텍스트 파일 뿐 아니라 OLAP 등 접근 가능한 데이타 소스를 다양화하는 연구가 필요하다. 또한 사용 의도 또는 응답 시간을 고려하여 동적 문서의 생성을 클라이언트와 서버 중 선택적으로 가능하도록 하는 시스템으로 확장할 예정이다.

참 고 문 헌

[1] Tetsuya, M. and Nobuo, T., "A Reporting Tool Using Programming by Example For Format Designation," Proceedings of 5th international conference on Intelligent user interfaces, ACM Press. 2000.

[2] Carlos, A., "Java and Reports, Some Solutions for the Past, Present and Future," ACM SIGUCCS USER SERVICES CONFERENCE, VOL 29, 275-278, 2001.

[3] Mikael J., "THIN vs. FAT Visualization Client," Computer Graphics International pp. 772-788, 1998.

[4] Crystal Reports, "http://www.businessobjects.com/solutions/crystalreports/default.asp."

[5] Style Report, "http://www.inetsoft.com."

[6] JReport, "http://www.jinfonet.com."

[7] Howard, J., "Magic Quadrants for Business Intelligence, 1H04," Gartner Research, 2004.

[8] SQL Server 2000 Reporting Services, "http://www.microsoft/sql/reporting/default.mspx."

최 지 웅

2001년 숭실대학교 컴퓨터학부(학사). 2003년 숭실대학교 컴퓨터학과(석사). 2003년~현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 분산/병렬 컴퓨팅, 그리드, 웹서비스, BI, 보안, 유비쿼터스

김 명 호

1989년 숭실대학교 전자계산학과(학사) 1991년 포항공과대학교 전자계산학과(공학석사). 1995년 포항공과대학교 전자계산학과(공학박사). 1995년 한국전자통신연구소 선임연구원. 1998년~1999년 미국 테네시주립대 교환교수. 1995년~현재 숭실대학교 정보과학대학 컴퓨터학부 교수. 관심분야는 분산/병렬 컴퓨팅, 그리드, 웹서비스, BI, 보안