(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) 。Int. Cl. [6]
G06F 3/14

(45) 공고일자 2002년09월10일
(11) 등록번호 10-0351535
(24) 등록일자 2002년08월23일

(21) 출원번호 10-1999-0027671
(22) 출원일자 1999년07월09일

(65) 공개번호 특2000-0011602
(43) 공개일자 2000년02월25일

(30) 우선권주장 09/119,427 1998년07월20일 미국(US)

(73) 특허권자 인터내셔널 비지네스 머신즈 코포레이션
미국 10504 뉴욕주 아몬크

(72) 발명자 시크사메리엘렌
미국플로리다33607템파파빌리온빌딩웨스트닥터마틴루터킹주니어블러바드3405
포소븐스티븐
미국플로리다33607템파파빌리온빌딩웨스트닥터마틴루터킹주니어블러바드3405

(74) 대리인 허정훈
김성택

심사관 : 이정숙

(54) 그래피컬 유저 인터페이스 콘트롤 인핸서를 제공하는 시스템 및 방법

요약

컴파일러 또는 GUI 툴킷(toolkit)에 의해 제공되는 서브-클래싱(버튼들, 리스트 상자들, 편집 콘트롤들 등)이 적절하지 않는 객체 지향 개발 환경에서 신속한 그래피컬 유저 인터페이스(GUI) 개발을 돕기 위한 소프트웨어 애플리케이션 시스템 및 방법이 제공된다. 본 발명은 상기한 바와 같은 환경에 의해 의도되었으나, 어떠한 객체 지향 GUI 프로그래밍 환경에서도 사용될 수 있다. 본 발명은 데이터 또는 다른 설정들로 콘트롤을 초기화하는 것; 콘트롤로부터 데이터를 검색(retrieve)하는 것; 데이터를 포맷하는 것; 입력된 데이터를 검증하는 것; 파일들 또는 데이터베이스들 내에 검증된 데이터를 저장하는 것; 단일 콘트롤로 보이도록 복수의 콘트롤들을 핸들링하는 것; 콘트롤들간의 관계를 설정하는 것을 포함하지만 이들에 제한되지 않는 네이티브 콘트롤들(native controls)에 기능성을 부가하기 위한 방법을 제공한다. 본 발명의 시스템 및 방법은 콘트롤 인핸서들로 언급되는 빌딩 블록들의 집합을 포함하며, 각 콘트롤 인핸서는 데이터 저장소, 데이터 초기화자, 데이터 종료자, 관련 콘트롤들에 대한 콘트롤 인핸서들 사이의 콘트롤 관계들을 유지하고 그에 기초하여 동작하는 수단을 포함한다.

대표도

도 1

색인어

GUI, 유저 인터페이스, 콘트롤 인핸서, 이벤트 핸들링

명세서

도면의 간단한 설명

도 1은 본 발명에 따른 시스템을 설명하기 위한 도면.

도 2는 본 발명 시스템에 의해 핸들되는 콘트롤 관계를 나타내는 도면.

도 3a 및 도 3b는 윈도우 코드 내에 콘트롤 인핸서들을 프로그래밍하기 위한 대표적인 프로세스 흐름을 나타내는 도면.

도 4a 및 도 4b는 본 발명 시스템에 의한 이벤트 핸들링을 위한 대표적인 프로세스 흐름을 나타내는 도면.

< 도면의 주요 부분에 대한 부호의 설명

100…윈도우

110…콘트롤 인핸서 리스트

111…GUI(graphical user interface) 콘트롤

112…콘트롤 인핸서(control enhancer; 제어 인핸서)

113…데이터 초기화자(data initializer)

114…데이터 종료자(data finalizer)

115…데이터 저장소 핸들러(data storage handler)

116…디폴트 파인더(default finder)

117…제한 디렉터(limitation director)

118…확인자(validator)

119…관계

120…룰(rule)

130…동작(action)

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

발명의 분야

본 발명은 컴퓨터: 유저 상호작용에 관한 것으로, 특히 모델-뷰-콘트롤러 디자인을 향상시키는 그래피컬 유저 인터페이스 개발에 관한 것이다.

발명의 배경

객체 지향 환경(object-oriented environments)에서 그래픽 유저 인터페이스(GUI)를 개발함에 있어서, 콘트롤을 구현하기 위한 다양한 방법들이 개발되어 왔다. 미국 특허 제5,555,365호에서, 그래피컬 유저 인터페이스(이하, GUI라 함) 객체와 애플리케이션 객체간의 관계를 특정하는 테이블을 생성한다. 다른 예에서, 프로그래머는 디스플레이, 포맷, 확인, 데이터 저장 및 콘트롤 관계들의 관리를 위하여 각 윈도우 상에 각 콘트롤을 위한 모든 코드를 작성할 수 있다. 그러한 방식은 프로그래머 시간, 공간 및 운용 시간 면에서 비효율적이며, 특히 서로 다른 애플리케이션에서 많은 콘트롤들이 "재-사용"될 수 있다는 사실에 비추어 더욱 그러하다.

C++ 프로그래머들은 컴파일러 또는 GUI 개발 툴킷 또는 프레임워크로써 포함된 콘트롤(control; 제어)들을 표현하는 클래스들의 집합을 서브클래스한다. 특정 목적들을 위해 제공된 콘트롤 클래스들의 집합을 서브클래스화하는 것은 객체 지향 그래피컬 환경에서 특정된 콘트롤들을 생성하기 위해 일반적으로 행해지고 있다. 예를 들어, 프로그래머는 데이터베이스로부터/로 데이터를 판독/기록하는 리스트 상자의 서브클래스를 생성하고; 테이블로부터/로 그 데이터를 판독/기록하는 리스트 상자의 또 다른 서브클래스 등을 생성할 것이다. 이 예에서, 테이블로부터 데이터를 읽은 다음 데이터베이스에 선택된 엔트리를 기록하는 데 리스트 상자가 필요하면, 네 번째 서브클래스가 작성되어야만 한다. 또한, 이 네 번째 리스트 상자와 윈도우 상의 다른 콘트롤들간의 관계들을 관리하기 위하여 윈도우 코드가 작성되어야만 한다. 예를 들어, 유저가 월(月)의 리스트 중에서 엔트리 "4월"을 선택하면, 당해 월내에 일(日)들의 수를 표시하는 스크린 상의 또 다른 콘트롤이 "30"으로 변경되어야만 한다.

서브클래싱 방법은 다음과 같은 2가지의 단점이 있다:

(1) 큰 애플리케이션에서, 콘트롤들간의 관계들을 관리하기 위하여 작성되어야만 하는 상당량의 코드와 긴 개발 시간을 요구할 정도로 서브클래스들의 수가 매우 방대해질 수 있으며; 그리고 (2) 서브클래싱 콘트롤들로 인해 몇 가지의 테스트 툴을 사용할 수 없게 된다. 서브클래싱에 의해 콘트롤로부터 서브클래스된 방대한 수의 클래스들을 야기할 수 있는데, 이 콘트롤은 혼합 및 정합 능력을 갖춘 소수의 클래스로 귀착될 수 있는 것이다. 어떤 프로그램은 "완벽히 기능적"-추가 코딩이 필요없이 요구되는 모든 것을 수행한다는 의미, 즉 수행해야할 동작에 대한 모든 코딩을 다 포함하고 있다는 의미임- 인 콘트롤들을 제공하지만, 그러한 콘트롤들은 모든 프로그래밍 과업에 대해 수용되기는 극히 어렵고 직접적인 적합화(customize) 또한 불가능하다.

바람직한 것은, 각각의 네이티브 클래스(native class)에 콘트롤들을 맡겨두면서, 빌딩 블록들의 세트를 사용하여 콘트롤들에 기능성을 부가함으로써, 각 콘트롤 기능을 위해 특정 코드를 작성하기 위해 걸리는 것보다 적은 시간 내에 GUI 애플리케이션을 구축하게 되는 시스템 및 방법이다. 그러한 발명적인 시스템 및 방법을 사용함으로써, 프로그래머는 주어진 애플리케이션의 특별한 요구에 특별히 맞추어진 재사용 가능한 툴들의 세트를 개발할 수 있다. 현존하는 콘트롤들은, 그것이 원래의 것이건 컴파일러 벤더 또는 제3자에 의해 강화된 것이건 역시 디자인에 관련될 것이다.

발명이 이루고자 하는 기술적 과제

컴파일러 또는 GUI 툴킷에 의해 제공되는 서브클래싱 GUI 콘트롤 클래스들(버튼들, 리스트 상자들, 편집 콘트롤들, 등)이 충분하지 않은 객체 지향 개발 환경에서 그래피컬 유저 인터페이스(GUI)의 조속한 개발을 돕는 소프트웨어 애플리케이션 시스템 및 방법이 개시된다.

발명의 구성 및 작용

본 발명은 어떠한 객체 지향 GUI 크로스-플랫폼 애플리케이션에서도 사용될 수 있다. 본 발명은 네이티브 콘트롤들에 기능성을 추가하기 위한 방법을 제공한다. 여기서, 네이티브 콘트롤은 데이터 또는 다른 설정들로 콘트롤을 초기화하는 것: 콘트롤로부터 데이터를 검색하는 것; 데이터를 포맷하는 것; 입력 데이터를 검증하는 것; 검증된 데이터를 파일들 또는 데이터베이스 내에 저장하는 것; 단일 콘트롤인 것처럼 보이도록 복수의 콘트롤들을 핸들링하는 것; 및 콘트롤들 간의 관계들을 설립하는 것을 포함하지만 이에 제한되는 것은 아니다. 본 발명 시스템 및 방법은 매우 짧은 시간에 복잡한 기능들을 갖는 GUI 윈도우들을 생성하는데 사용하기 위한 빌딩 블록들의 세트를 포함한다.

이어서 첨부한 도면들을 참조하여 본 발명의 바람직한 실시예를 상세히 설명하기로 한다.

GUI 윈도우 코드는 약간의 예외가 있긴 하지만 이벤트 구동 형태로 작성된다. "이벤트"는 마우스 클릭, 버튼 클릭, 마우스 이동(mouse movement), 키조작(keystroke), 음성 인식 하드웨어로의 구두 명령 또는 사용자와 프로그램간의 인식 가능한 다수의 상호 작용들 중의 어느 하나이다. 창의적인 콘트롤 인핸서를 특징으로 하는 본 발명은 GUI 윈도우들이 이벤트 구동 애플리케이션들로서 디자인되어 있는 시스템에 관한 것이다. 이어지는 설명들은 본 발명을 이벤트-구동 애플리케이션들[예컨대, Windows(상표명), OS/2(상표명) 또는 UNIX 상에서 동작함]을 위해 구현된 것으로서 상술하지만, 본 발명을 변형함으로써 메뉴 구동 애플리케이션을 위해서도 역시 구현될 수 있음을 알 수 있다. 본 발명의 기본 개념은 이벤트가 윈도우 상의 콘트롤들에 속할 때 그 윈도우 객체에 연관된 콘트롤 인핸서로 이벤트를 전달하는 것이다(예를 들어, 라디오 버튼상의 마우스 클릭은 라디오 버튼의 제어를 의도한 것인 반면, 윈도우-사이즈드 이벤트는 윈도우 객체 자체에 의해 핸들되도록 함). 디자인 자체는 어떠한 플랫폼 상의 어떠한 애플리케이션을 위해서도 사용될 수 있다. 이 디자인에 따라 쓰여진 코드는 그것이 쓰여진 플랫폼 상의 어떠한 GUI 애플리케이션을 위해서도 사용될 수 있으나, 이 플랫폼들 상에서 가능한 네이티브 콘트롤들에서의 변경 및 데이터가 애플리케이션 내에서 어떻게 핸들되는가에 따라, 플랫폼들을 변경할 때 약간의 변형을 요구할 수도 있다. 예를 들어, 서로 다른 GUI 애플리케이션들은 데이터의 서로 다른 내부 표현(representation)을 가지며 GUI 콘트롤 인핸서들은 적절한 포맷을 판독/기록하기 위해서 커스토마이즈될 필요가 있을 것이다. 최적의 것은, 플랫폼의 변경은 플랫폼들에 걸쳐서 일관된 인터페이스를 제공하기 위해 사용되는 크로스-플랫폼 GUI 콘트롤 라이브러리를 사용할 때 네이티브 콘트롤들 내의 차이점들을 위해 어떠한 코드 변경도 요구하지 않는 것이다. 라이브러리는 각 플랫폼에 따라 코드가 특별히 쓰여지는 것이 요구되지 않도록 서로 다른 플랫폼들 상에서 이용 가능한 네이티브 콘트롤들간의 차이점들을 감춘다.

GUI 개발의 숙련자들은 모델-뷰-콘트롤러(MVC) 디자인에 익숙하다. MVC 구현에서, 모델은 데이터를 표현(represent)하고 뷰(예컨대, 스프레드시트, 리스트뷰, 테이블, 등)는 데이터를 표출(display)한다. GUI 애플리케이션에서, 동일 데이터 상에 몇 가지 활성 뷰(active Views)(예를 들어, 스프레드시트 및 그래프)가 있을 수 있다. 뷰가 특정 모델에 연관된 데이터에 의해 영향을 받을 수 있는 경우 뷰는 그 모델에 연관된 콘트롤러로 자신을 등록한다. 뷰는 콘트롤러를 통해 데이터의 변경을 요구한다. 그러면 콘트롤러는 모델에게 변경을 지시하고 이어서 데이터가 변했음을 모든 등록된 뷰들에게 알린다. 본 발명의 콘트롤 인핸서들은 MVC-기반 시스템에서 구현된다. 콘트롤 인핸서 디자인은 상세히 후술하며 도 1에 도시되어 있는 다음과 같은 콤포넌트들: 콘트롤 인핸서, 데이터 저장소, 데이터 초기화자(data initializer), 데이터 종료자(data finalizer), 콘트롤 관계, 룰들 및 동작들로 구성된다.

도 1에 나타낸 바와 같이, 윈도우(100) 상의 각 GUI 콘트롤(111)은 콘트롤 인핸서(112)에 할당된다. 콘트롤 인핸서

(112)는 콘트롤 (예를 들어, 텍스트 엔트리 필드, 리스트 상자, 콤보 상자 등)의 성질에 관계없이, 윈도우 객체를 위해, GUI 콘트롤(111)에 대한 일관된 인터페이스를 제공한다. 예를 들어, 콘트롤 인핸서가 셋데이터(SetData)를 포함하는 인터페이스를 제공하는 중일 때, 콘트롤 인핸서 서브클래스는 텍스트 객체, 스핀 버튼, 또는 다른 어떤 객체를 관리할 수도 있다. 외부 코드는 단지 콘트롤 인핸서의 셋데이터 메쏘드(SetData method)만을 알면되며 각 콘트롤을 위한 특정 메쏘드들은 인식할 필요가 없다.

윈도우용 GUI 콘트롤에 대한 인터페이스를 제공하는 것 외에, 콘트롤 인핸서는 GUI 콘트롤에게 콘트롤 인핸서의 내부 콤포넌트들(즉, 이하에서 상세히 설명할 데이터 초기화자(113), 데이터 종료자(114), 데이터 저장소 핸들러(data storage handler)(115))에 대한 일관된 인터페이스를 제공한다. 각종 콘트롤 또는 밀접하게-관련된 콘트롤들의 패밀리를 핸들하기 위해 상이한 콘트롤 인핸서가 제안되어 있다.

각 GUI 윈도우 객체는 그 윈도우 상의 콘트롤들을 위한 콘트롤 인핸서들의 집합을 유지한다. 윈도우 객체는 이벤트들을 전달되는 콘트롤 인터페이스들의 리스트를 유지한다. 콘트롤 인핸서 리스트(110)는 콘트롤 인핸서들 상으로 이벤트들을 전달하는 객체이다. 각 콘트롤 인핸서는 인스턴스화(instatiate)될 때 콘트롤 인핸서 리스트에 등록된다. 각 콘트롤 인핸서가 콘트롤 인핸서 리스트에 등록될 때, 또한 어떤 이벤트들이 그것에 영향을 미칠 것인지가 등록된다. 윈도우가 이벤트를 수신할 때, 윈도우는 이벤트를 콘트롤 인핸서 리스트로 전달하고 콘트롤 인핸서 리스트는 그러한 이벤트에 관계된 것으로서 등록되어 있는 콘트롤 인핸서들에게만 그 이벤트를 전한다. 데이터 저장소 핸들러(115)가 각 콘트롤 인핸서에게 할당된다. 데이터 저장소 핸들러는 "저장(save)" 이벤트가 수신될 때 데이터를 어디에 저장해야하는지를 결정하고 그 데이터의 저장소을 핸들한다. 예를 들어, 데이터는 데이터베이스, INI 파일에 저장되거나 또는 이후의 처리를 위한 어떤 메모리 공간에 저장될 수도 있다. 새로운 데이터 저장소 핸들러가 데이터가 프로그램에 의해 저장될 수 있는 각각의 방식을 핸들하도록 디자인된다.

윈도우 코드가 콘트롤 인핸서를 인스턴스화할 때, 윈도우는 또한 원하는 형태의 데이터 초기화자(113)를 인스턴스화하고 그것을 콘트롤 인핸서로 전한다. 새로운 데이터 초기화자가 각 형태의 데이터 초기화를 핸들하기 위해 고안된다. 예를 들어, 콘트롤을 위한 초기 데이터 값들 또는 설정치들은 테이블들, INI 파일들, 또는 데이터베이스로부터 얻어질 수 있으며 프로그램은 어떤 한 포맷으로 데이터를 저장할 수도 있으나 다른 포맷으로 그것을 디스플레이할 필요가 있을 수도 있다. 서로 다른 데이터 초기화자들은 서로 다른 초기 데이터를 핸들할 것이다. 각 데이터 초기화자는 초기화시 요구되는 어떠한 데이터 조작도 핸들한다. 콘트롤 인핸서는 초기화 이벤트(Initialize event)가 수신될 때 데이터 초기화자를 활성화한다. 디폴트 파인더(116) 및 제한 디렉터(117)는 나중에 좀더 상세히 설명되는 바와 같이 데이터 초기화자에 연관된다.

데이터 종료자(114)는 데이터 저장소 핸들러에 의해 저장될 데이터를 준비하기 위해 각 콘트롤 인터페이스에 연관되어 있다. 콘트롤 인핸서가 저장(save) 이벤트를 수신하면, 데이터 저장소 핸들러에게 알리고, 데이터 저장소 핸들러는 다시 어떤 데이터 또는 설정을 저장해야하는지를 결정하기 위해서 데이터 초기화자와 상의한다. 새로운 데이터 종료자는 프로그램에 의해 필요로 될 수 있는 각종 데이터 준비를 핸들하도록 고안된다. 예를 들어, 프로그램은 "A"값을 저장하고 그것을 "모든 값들"로서 표시할 필요가 있을 수 있다. 데이터 종료자는 콘트롤로부터의 데이터가 저장되기 전에 수행되어야 할 어떠한 데이터 조작도 핸들한다. 확인자(118) 및 상기한 제한 디렉터(117)는 데이터 종료자에 관련되어 있다.

몇 개의 윈도우 상에서, 프로그래머는 하나의 콘트롤을 다른 콘트롤의 설정에 근거하여 변경할 필요가 있음을 발견할 수도 있다. 그러므로 콘트롤 관계(119)는 피영향 콘트롤 인핸서(Affected control Enhancer)에 대해 보관되고 있는 포인터와 함께, 콘트롤 인핸서들중 하나[소유 콘트롤 인핸서(Owning Control Enhancer)로 알려진]로 할당된다. 각

콘트롤 인핸서는 필요하면 콘트롤 관계들의 세트를 보관한다. 소유 콘트롤 인핸서가 변경 이벤트를 수신하였을 때, 그 관계들 내에서 임의의 피영향 콘트롤 인핸서들로 이 이벤트를 전한다. 콘트롤 인핸서는 프로그래머에 의해 결정된 바대로 하나 이상의 관계들을 가질 수 있다.

도 2에 도시된 바와 같이, 관계들은 지정된 룰들(120 및 121)이고, 룰들은 지정된 동작들(130, 131) 등이다. 룰들은 두 콘트롤들(즉, 소유 콘트롤 인핸서의 콘트롤 및 그와 관계있는 다른 콘트롤 인핸서의 콘트롤)의 데이터 값들을 고찰하고 비교하는 객체들이다. 만일 비교 결과가 참이면, 룰을 만족한다고 한다. 룰을 만족할 때, 동작이 트리거된다. 동작은 피영향 콘트롤 인핸서의 양태를 변경하기 위해서 피영향 콘트롤 인핸서로 메시지들을 보낸다. 룰들은 콘트롤들간에 필요한 각 비교 유형에 대해 규정되어야 한다. 예를 들어, 초과 룰(Greater Than Rule)은 피영향 콘트롤 인핸서의 데이터보다 소유 콘트롤 인핸서의 데이터가 큰지를 알아보기 위해서 소유 콘트롤 인핸서의 데이터를 체크한다. 만일 그렇다면, 동작이 트리거된다. 동작들은 프로그램에 의해 필요한 각종 행위 변경을 핸들하도록 고안되어야 한다. 예를 들어, 트리거될 때, 디스에이블 동작(Disable Action)은 그 콘트롤을 디스에이블 하도록 피영향 콘트롤 인핸서에게 명령할 것이다. 전례에서 디스에이블 동작이 구현될 수 있는 방법은 다음과 같다: 윈도우 상의 체크 박스는 소유 콘트롤 인핸서를 가지며 피영향 콘트롤 인핸서는 텍스트 엔트리 필드를 관리한다고 가정한다. 텍스트 엔트리 필드는 체크 박스가 선택[예를 들어, "체크(checked)"]되면 디스에이블될 것이다. 체크 박스를 위한 소유 콘트롤 인핸서가 "체크" 이벤트를 수신할 때, 소유 콘트롤 인해서는 그의 룰에게 적절한 비교(즉, 룰에 의해 요구되는 상태를 "체크"하는, 상태 비교)를 하라고 요구한다. 결과적으로 이벤트가 "체크" 중인 박스였으므로, 룰은 만족되었고 동작은 트리거되어, 그 결과 디스에이블 메시지가 피영향 콘트롤 인핸서로 보내진다.

각 콘트롤은 윈도우 상의 복수의 콘트롤과 관계들을 가질 수도 있다. 관계는, 복수의 룰을 가질 수 있다. 각 룰들에 대해서, 할당된 복수의 동작들(예를 들어, 동작 1 - 피영향 콘트롤 인핸서로 디스에이블 메시지 보내기, 동작 2 - 피영향 콘트롤 인핸서로 컬러 변경 메시지 보내기)이 있을 수 있다. 소유 콘트롤 인핸서는 그 관계들의 각각에게로 변경 이벤트를 전하고, 적절하다면 각 관계에서 각각은 그의 룰들 모두를 체크하고 하나 이상의 후속되는 동작들을 트리거한다.

도 3a 및 도 3b를 참조하여, 윈도우 코드 내에 콘트롤 인핸서들을 프로그래밍하기 위한 대표적인 프로세스 흐름을 살펴보기로 한다. 본 발명의 콘트롤 인핸서의 구현은 원하는 결과들을 얻기 위해 다양한 방식들로 다양한 콤포넌트들을 조립하는 것으로 이루어진다. 기본적으로, 콘트롤 인핸서의 조립은 다음 방식으로 수행된다: 윈도우는 단계 301에서 GUI 콘트롤을 생성한 후 단계 302에서 콘트롤 인핸서를 생성한다. 다음에 윈도우는 단계 303에서 GUI 콘트롤에 대한 포인터를 콘트롤 인핸서로 전하고 단계 304에서 콘트롤 인핸서 리스트에 콘트롤 인핸서를 부가한다. 단계 305에서 특별 데이터 저장소 핸들러가 요구될 것인지에 관하여 결정한다. 만일 필요하다면, 단계 306에서 윈도우는 생성된 인핸서에 적절한 형태의 데이터 저장소 핸들러 객체를 할당한다. 다음에, 단계 307에서, 특별 초기화가 필요한지에 관하여 결정한다. 만일 필요하다면, 데이터 초기화자가 단계 308에서 윈도우에 의해 인핸서에 할당된다. 다음에, 단계 309에서, 특별 데이터 종료자가 요구될 것인지를 결정한다. 단계 309에서의 결정에 기초하여, 단계 310에서 적절한 형태의 데이터 종료자를 할당한다.

만일 단계 311에서 결정된 바와 같이, 콘트롤이 스크린 상에 다른 콘트롤들과 관계가 있다면, 다음을 포함하는 추가 단계들이 윈도우에 의해 이루어진다: 단계 312에서, 각 관계를 인스턴스화하고, 인스턴스화된 관계를 콘트롤 인핸서에 할당하고, 소유 및 피영향 콘트롤 인핸서들 양쪽에서 포인터들을 설정하고; 단계 313에서, 관계에 적절한 룰을 인스턴스화하고; 단계 314에서, 적절한 동작들을 인스턴스화하고 룰들에 동작들을 할당한다. 단계 305, 307, 309, 311이 실행되는 순서가 중요하지 않음을 알 수 있을 것이다.

일단 전술한 기초적인 빌딩 블록들이 완료되고 테스트되면, 그들은 거의 모든 GUI 윈도우를 핸들하기 위하여 다양한

방법으로 결합될 수 있다. 그러면, GUI를 프로그래밍하는 것은 콘트롤 인핸서들로써 관리되는 몇 개의 GUI 콘트롤들을 갖는 공통 베이스 윈도우 클래스(common base window class)를 규정하는 문제로 귀결된다. 코더는 베이스 윈도우 클래스로부터, 특정 윈도우를 위한 고유한 GUI 콘트롤들을 부가하는 서브클래스들을 유도할 수 있다. 이 방식으로, 코더는 각 서브클래스에 대해 코드의 공통 특징들을 여러번 다시 코딩할 필요가 없게 된다.

전술한 바와 같이, 콘트롤 인핸서가 조립될 때, 신뢰할 수 있는 데이터를 특정하는 데 사용되는 데이터 식별자가 할당된다. 할당된 데이터 식별자는 콘트롤 인핸서에 연관된 객체들 모두에 의해 사용된다. 더미 식별자가 어떠한 초기 데이터를 로드 하거나 또는 완료시 데이터를 저장할 필요가 없는 콘트롤들(예를 들어, 파일로부터의 메시지를 단순히 표시하는, 상태 바 콘트롤 인핸서)에 대해 사용될 수 있다.

윈도우가 콘트롤 인핸서를 생성할 때, 초기화 메쏘드을 호출한다. 이어서 콘트롤 인핸서는 데이터 초기화자의 초기화 메쏘드를 호출한다. 각종 데이터 초기화자가 그 형태에 특정한 초기화를 수행한다. 어떤 데이터 초기화자가 데이터 식별자에 기초하여 디폴트 데이터를 찾기 위해 디폴트 파인더 객체(116)를 사용할 수 있다. 다른 데이터 초기화자는 또 다른 어떤 데이터 초기화자가 데이터베이스로부터 데이터를 읽는 반면, INI 파일로부터 데이터를 읽을 수도 있다. 초기화가 지시될 때, 각 데이터 초기화자는 적절한 소스로부터 데이터를 읽은 다음 콘트롤 내에 데이터를 디스플레이하기 위해서 콘트롤 인핸서의 셋데이터 메쏘드(SetData method)을 호출한다.

콘트롤 인핸서 초기화 코드는 또한 테이블-구동 제한 디렉터(117)와 상의하며, 테이블-구동 제한 디렉터(117)는 데이터 상에 놓인 임의의 적절한 제한들을 찾기 위해 데이터 식별자를 사용한다. 제한들의 예에는 데이터에 대한 최대 및 최소 길이들, 데이터에 대한 최대 및 최소값들, 데이터 형태(예를 들어, 모든 숫자, 날짜, 시간, 등)를 포함한다. 적절하다면, 콘트롤 인핸서는 콘트롤의 사용을 통해서 사용자가 적절히 가이드 되도록, 그 행위를 조절하기 위해 그의 콘트롤로 메시지를 보내기 위해 이 정보를 사용할 수도 있다.

초기화 후, 콘트롤 인핸서는 적절한 이벤트들에 응답하기 위해 준비한다. 도 4a 및 도 4b를 참조하면, 단계 401에서 윈도우 이벤트가 검출될 때, 윈도우는 단계 402에서 콘트롤 인핸서 리스트를 액세스하고 단계 403에서 만일 그 형태의 이벤트에 관심이 있는 것으로 등록된 어떤 콘트롤 인핸서들이 있는지를 결정한다. 만일 콘트롤 인핸서들이 그 형태의 이벤트에 대해 등록되어 있지 않다면, 단계 404에서 이벤트는 무시된다. 만일 단계 403에서 그러한 윈도우 이벤트에 의해 영향을 받는 것으로 리스트에 등록된 적어도 하나의 콘트롤 인핸서가 있다고 결정되면, 관심 있는 콘트롤 인핸서들이 단계 405에서 이벤트를 취한다. 콘트롤 인핸서에서, 단계 410에서 이벤트가 디스플레이 이벤트인지를 결정한다. 그렇지 않다면, 단계 420에서 이벤트가 저장 이벤트인지를 결정한다. 그렇지 않다면, 단계 430에서 이벤트가 초기화 이벤트인지를 결정한다. 그렇지 않다면, 핸들될 수 있는 다른 형태의 이벤트들(도시하지 않음)에 연속된 것일 수 있다.

단계 410에서 결정된 바와 같이, 이벤트가 디스플레이 이벤트이면, 시스템은 단계 411에서 콘트롤 내의 새로운 데이터를 디스플레이하거나 또는 선택한다. 단계 412에서 결정된 바와 같이, 체크할 어떤 관계들이 있다면, 룰들이 단계 413에서 (예를 들어, 비교에 의해) 체크된다. 어떤 룰이 참으로 평가되면, 지정된 동작이 단계 414에서 실행된다.

단계 420에서 결정된 바와 같이, 이벤트가 저장 이벤트이면, 이벤트는 단계 421에서 데이터 종료자로 전달된다. 데이터를 확인하는 선택적인 단계가 단계 422에서 구현될 수 있다. 확인을 위해, 프로그래머는 데이터를 확인해야 하는지 및 어떻게 확인해야 하는지에 관한 정보를 제공하기 위하여, 데이터 초기화자들 상에 맞추어, 테이블들을 설정하여야만 한다. 확인 객체(Validation object)가 데이터가 부정확하게 들어왔음을 발견하면, 단계 424에서 오류 윈도우로 오류 메시지 수를 전달하며, 오류 윈도우는 사용자에게 적절한 메시지를 디스플레이한다. 통상, GUI 환경에서, 콘트롤은 무효 선택이 이루어질 수 없도록 사용자의 선택을 제한한다. 그러나, 그러한 것이 불가능한 경우에, 콘트롤 인핸서는 확인

자 객체(Validator object)(도 1의 118)를 생성할 수도 있다. 오류 메시지가 생성케한 콘트롤에 대한 데이터 식별자를 추적함으로써, 윈도우 코드가 그 정보를 이용하여 에러 상황을 바로잡을 콘트롤 인핸서를 찾을 수 있다. 일단 데이터가 확인되면, 단계 423에서 적절한 곳으로 데이터를 기록하는 데이터 저장소 핸들러로 데이터를 전달한다.

단계 430에서 결정된 바와 같이, 이벤트가 초기화 이벤트이면, 이벤트는 단계 431에서 데이터 초기화자로 전달되고 단계 432에서 초기 데이터가 검색되고 디스플레이된다. 단계 414의 동작 실행 또는 단계 423의 데이터 저장 후뿐만 아니라 단계 432에서의 디스플레이 후에도, 메쏘드는 다른 윈도우 이벤트를 기다리기 위해 단계 406에서 리턴한다.

프로그래머가 단일 엔터티로서 콘트롤들의 그룹을 핸들할 필요가 있는 GUI 개발 환경들이 있다. 예를 들어, 필드들의 날짜 엔트리 세트는 년을 위한 스핀 버튼, 월(月)을 위한 드롭-다운 리스트, 일(日)을 위한 또 다른 스핀 버튼으로 구성될 수 있다. 날짜가 단일 필드로서 데이터 베이스 내에 저장되고, 단지 하나의 데이터 식별자를 갖더라도, GUI가 날짜의 각 콤포넌트에 대해 적절한 간편 사용(easy-to-use) 콘트롤을 통해 날짜의 각 콤포넌트를 사용자가 선택하도록 경우가 많다. 전례에 있어서, 각 콤포넌트마다 하나씩, 3개의 콘트롤 인핸서들이 생성되는 것이 가장 바람직하다. 그러나, 데이터 저장은 3개 콘트롤 인핸서들 모두를 인식할 필요가 있으므로, 멀티-콘트롤 인핸서가 사용되어야만 한다. 멀티-콘트롤 인핸서는 개별 콤포넌트를 단일 엔터티 및 복수의 개별 콤포넌트들 모두로서 핸들하기 위해 작성된다. 멀티-콘트롤 인핸서는 콤포넌트들(즉, 데이터 초기화자 등)과 연관된 다른 모든 콘트롤 인핸서들과 동일한 인터페이스를 제공하지만 적절하게 자신의 고유한 데이터를 핸들한다. 상기한 바와 같이 이것은 어떤 데이터 저장, 관계들 등에 할당될 수도 있으나 이것의 초기화자 및 종료자는 그 업무들에 매우 특정된다.

멀티-콘트롤 인핸서는 그 자신의 콘트롤 인핸서들로 콘트롤들을 관리할 필요가 있을 수도 있고 그렇지 않을 수도 있다. 어떤 경우에는 콘트롤들을 직접 관리하는 것이 보다 쉬울 수도 있다; 그러나, 콘트롤 인핸서들을 사용함으로써 프로그래머로 하여금 멀티-콘트롤 인핸서 코드에 영향을 주지 않고 개별 데이터 콘트롤들을 나중에 변경할 수 있도록 한다. 멀티-콘트롤 인핸서는 그들 자신이 콘트롤 인핸서들의 복합체일 수도 있는 현존하는 콘트롤 인핸서들의 복합체일 수 있다. 예를 들어, 6개의 콘트롤들을 다음에 나타낸다:

월(月)을 선택하기 위한 콘트롤(1-12로 제한됨);

일(日)을 선택하기 위한 콘트롤(선택된 달 및 년에 기초하여 1-28/29/30/31로 제한됨);

년(年)을 입력하기 위한 콘트롤(4 디지트 값);

시간을 위한 콘트롤(0-23으로 제한됨);

분을 위한 콘트롤(0-59로 제한됨);

초를 위한 콘트롤(0-59로 제한됨).

날짜를 입력하기 위해, 월(月) 및 년(年)에 기초하여 일(日)을 제한하기 위한 모든 관계들을 가지고, 처음 3개의 콘트롤들을 제어하는 날짜 멀티-콘트롤 인핸서가 구축된다. 처음 3개의 콘트롤들은 자신들을 제어하는 간단한 콘트롤 인핸서들-데이터 액세스를 위해 날짜 멀티-콘트롤 인핸서에 의해 사용됨-을 가진다. 날짜 멀티-콘트롤 인핸서는 이 3개의 콘트롤 인핸서들로부터의 데이터를 어떻게 얻고 이 값들을 하나의 값으로 어떻게 결합하는지를 알고 있다. 이런 식으로, 그 정보의 모두가 날짜시간 콘트롤 인핸서(DateTime Control Enhancer)에 대해 재-코딩될 필요는 없다.

마찬가지로, 후반 3개의 콘트롤들에 대한 시간 멀티-콘트롤 인핸서가 구축된다. 그러므로, 날짜시간 멀티-콘트롤 인핸서를 구축하기 위해서, 날짜 멀티-콘트롤 인핸서 및 시간 멀티-콘트롤 인핸서들의 복합체가 구축된다. 새로운 날짜시간 콘트롤 인핸서는 단일의 날짜시간 값을 구축하기 위해 날짜 콘트롤 인핸서와 시간 콘트롤 인핸서로부터의 값들을 간단히 결합한다. 보다 복잡한 콘트롤 인핸서를 만들기 위해 현존하는 콘트롤 인핸서들을 취하고 그들을 결합하는 것이 멀티-콘트롤 인핸서의 기능이다.

각 콘트롤 인핸서는 멀티-콘트롤 상황에 있을 때 동일한 인터페이스를 갖는다. 프로그래머가 단일 콘트롤로서 복수의 GUI 콘트롤 기능을 갖는 것이 바람직하게 되는 조건을 만족시킬 필요가 있는 각 인스턴스에 새로운 멀티-콘트롤 인핸서가 생성된다. 도 3을 참조하면, 멀티-콘트롤 인핸서 셋업을 위해 윈도우 코드로 프로그래밍할 때, 프로그래밍의 차이는 단지 단계 303에 있으며, 복수의 포인터들이 생성되도록 한다.

본 발명은 몇 개의 특정 실시예들에 관해서 설명하였다. 그러나 이 기술 분야의 숙련자는 첨부한 특허청구범위에 나타낸 바와 같은 본 발명의 사상 및 범주를 벗어나지 않고서 변경들이 만들어질 수 있음을 인식할 것이다.

발명의 효과

상술한 바와 같이 본 발명에 의하면, 매우 짧은 시간에 복잡한 기능들을 갖는 GUI 윈도우를 생성할 수 있다는 이점이 있다.

(57) 청구의 범위

청구항 1.

복수의 윈도우 콘트롤들을 갖는 윈도우 객체에 의해 수신된 적어도 하나의 이벤트의 각 이벤트를 핸들하기 위해 강화된 기능성을 제공하는 시스템에 있어서,

각각이 상기 윈도우 객체를 위한 하나의 특정 콘트롤에 대한 인터페이스를 제공하는 복수의 콘트롤 인핸서들과;

상기 복수의 콘트롤 인핸서들 중 어느 것을 상기 수신된 이벤트를 핸들하기 위해 호출할 것인지를 결정하기 위한 상기 콘트롤 인핸서들의 리스트

를 포함하는 시스템.

청구항 2.

제1항에 있어서,

상기 복수의 콘트롤 인핸서들 각각은 적어도 하나의 데이터 저장소 핸들러를 포함하는 것인 시스템.

청구항 3.

제1항에 있어서,

상기 복수의 콘트롤 인핸서들 각각은 적어도 하나의 데이터 초기화자를 포함하는 것인 시스템.

청구항 4.

제1항에 있어서,

상기 복수의 콘트롤 인핸서들 각각은 적어도 하나의 데이터 종료자를 포함하는 것인 시스템.

청구항 5.

삭제

청구항 6.

삭제

청구항 7.

삭제

청구항 8.

삭제

청구항 9.

복수의 윈도우 콘트롤들을 갖는 윈도우 객체에 의해 수신된 적어도 하나의 이벤트의 각 이벤트를 핸들링하는 강화된 기능성을 제공하는 시스템에 있어서,

각각이 상기 윈도우 객체를 위한 하나의 특정 콘트롤에 인터페이스를 제공하는 복수의 콘트롤 인핸서들과 - 여기서, 상기 콘트롤 인핸서들의 각각은 적어도 하나의 데이터 저장소 핸들러와 적어도 하나의 데이터 초기화자 및 적어도 하나의 데이터 종료자를 포함함 - ;

상기 복수의 콘트롤 인핸서들중 어느 것을 상기 수신된 이벤트를 핸들하기 위해 호출할 것인지를 결정하기 위한 상기 콘트롤 인핸서들의 리스트

를 포함하는 시스템.

청구항 10.

제1항 또는 9항에 있어서,

상기 윈도우 콘트롤들중 제1 윈도우 콘트롤은 상기 윈도우 콘트롤들 중 적어도 하나의 제2 윈도우 콘트롤에 관련되어 있고,

상기 제1 윈도우 콘트롤을 위한 상기 콘트롤 인핸서는

상기 제2 윈도우 콘트롤을 위한 상기 콘트롤 인핸서에 대한 적어도 하나의 포인터와;

상기 제1 콘트롤 인핸서에서의 동작이 상기 제2 콘트롤 인핸서에 영향을 주는지를 결정하기 위한 적어도 하나의 수단과;

상기 제2 콘트롤 인핸서와 통신하기 위한 수단을 더 포함하는 것

인 시스템.

청구항 11.

제1항 또는 제9항에 있어서,

적어도 하나의 상기 콘트롤 인핸서들은 상기 콘트롤 인핸서에 관련된 데이터 상에 적용될 제한들을 결정하기 위한 수단을 더 포함하는 것인 시스템.

청구항 12.

제4항 또는 제9항에 있어서,

적어도 하나의 상기 콘트롤 인핸서들은 상기 데이터 종료자에서 데이터를 확인(validate)하기 위한 수단을 더 포함하는 것인 시스템.

청구항 13.

제1항 또는 제9항에 있어서,

적어도 하나의 상기 콘트롤 인핸서들은 상기 적어도 하나의 콘트롤 인핸서들의 윈도우 콘트롤에 관련된 데이터를 식별하기 위한 수단을 더 포함하는 시스템.

청구항 14.

윈도우 - 복수의 콘트롤 인핸서들을 포함함 - 에서 수신된 적어도 하나의 이벤트에 응답하여 윈도우 콘트롤들의 강화된 기능성을 제공하는 방법에 있어서,

상기 윈도우에서 이벤트를 수신하는 단계와;

상기 복수의 콘트롤 인핸서들로부터 상기 이벤트를 위한 적어도 하나의 관심 있는 콘트롤 인핸서를 로케이팅하는 단계;

상기 적어도 하나의 관심 있는 콘트롤 인핸서로 상기 이벤트를 전달하는 단계와;

상기 적어도 하나의 관심 있는 콘트롤 인핸서에서 상기 이벤트를 핸들링하는 단계

를 포함하는 윈도우 콘트롤들의 강화된 기능성을 제공하는 방법.

청구항 15.

제14항에 있어서,

상기 윈도우는 리스트된 콘트롤 인핸서들 각각에 영향을 주는 이벤트들의 콘트롤 인핸서 리스트를 포함하고,

상기 로케이팅하는 단계는

상기 이벤트들의 리스트를 액세스하는 단계와;

상기 이벤트들의 리스트와 상기 수신된 이벤트를 비교하는 단계와;

상기 비교에 기초하여 관심 있는 콘트롤 인핸서들을 결정하는 단계

를 포함하는 윈도우 콘트롤들의 강화된 기능성을 제공하는 방법.

청구항 16.

제14항에 있어서,

상기 윈도우 콘트롤들은 관계들을 가지며,

상기 수신된 이벤트는 디스플레이 이벤트이고,

상기 디스플레이 이벤트가 상기 관계들중 적어도 하나에 영향을 주는지를 결정하는 단계와;

상기 적어도 하나의 관계를 위한 적어도 하나의 룰이 참인지를 평가하는 단계와;

상기 적어도 하나의 룰이 참이면 적어도 하나의 동작을 실행하는 단계

를 더 포함하는

윈도우 콘트롤들의 강화된 기능성을 제공하는 방법.

청구항 17.

윈도우 상에서의 이벤트 핸들링을 위한 강화된 콘트롤을 제공하는 방법에 있어서,

상기 윈도우 상에 콘트롤을 생성하는 단계와;

상기 콘트롤을 위한 콘트롤 인핸서를 인스턴스화하는 단계와;

상기 콘트롤 인핸서로 상기 콘트롤을 위한 포인터를 전달하는 단계

를 포함하는 방법.

청구항 18.

제17항에 있어서,

특별한 데이터 핸들링이 요구되는지를 결정하는 단계와;

특별한 데이터 핸들링이 요구되면 적어도 하나의 데이터 핸들러를 인스턴스화하는 단계와;

상기 콘트롤 인핸서에게 상기 데이터 핸들러를 할당하는 단계

를 더 포함하는 방법.

청구항 19.

제17항에 있어서,

특별한 초기화가 요구되는지를 결정하는 단계와;

특별한 초기화가 요구되면 적어도 하나의 데이터 초기화자를 인스턴스화하는 단계와;

상기 콘트롤 인핸서에게 상기 적어도 하나의 데이터 초기화자를 할당하는 단계

를 더 포함하는 방법.

청구항 20.

삭제

청구항 21.

제17항에 있어서,

특별한 데이터 종료가 요구되는지를 결정하는 단계와;

특별한 데이터 종료가 요구되면 적어도 하나의 데이터 종료자를 인스턴스화하는 단계와;

상기 콘트롤 인핸서에게 상기 적어도 하나의 데이터 종료자를 할당하는 단계

를 더 포함하는 방법.

청구항 22.

삭제

청구항 23.

삭제

청구항 24.

삭제

청구항 25.

제17항에 있어서,

상기 콘트롤이 상기 윈도우 상의 적어도 하나의 다른 콘트롤와의 적어도 하나의 관계를 갖는지를 결정하는 단계와;

상기 적어도 하나의 관계를 인스턴스화하는 단계와;

상기 콘트롤 인핸서로 상기 적어도 하나의 관계를 할당하는 단계와;

상기 적어도 하나의 다른 콘트롤의 각각으로 포인터를 전달하는 단계

를 더 포함하는 방법.

청구항 26.

제25항에 있어서,

상기 적어도 하나의 관계에 대한 적어도 하나의 룰을 인스턴스화하는 단계와;

상기 적어도 하나의 관계로 상기 적어도 하나의 룰을 할당하는 단계

를 더 포함하는 방법.

청구항 27.

제26항에 있어서,

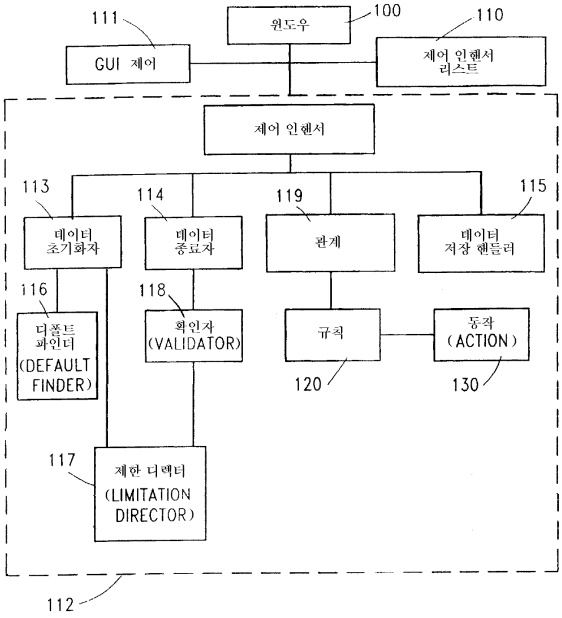
상기 적어도 하나의 룰에 대한 적어도 하나의 동작을 인스턴스화하는 단계와;

상기 적어도 하나의 룰으로 상기 적어도 하나의 동작을 할당하는 단계

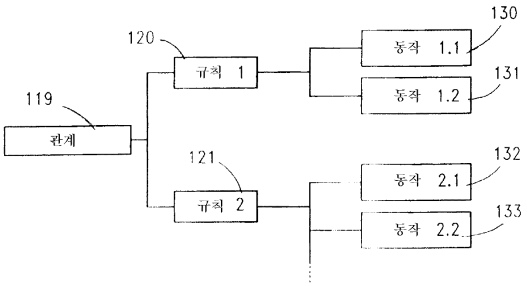
를 더 포함하는 방법.

도면

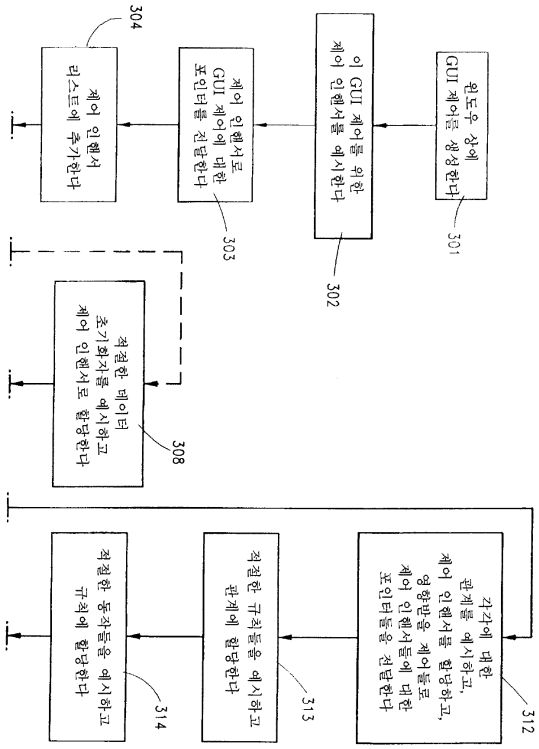
도면 1



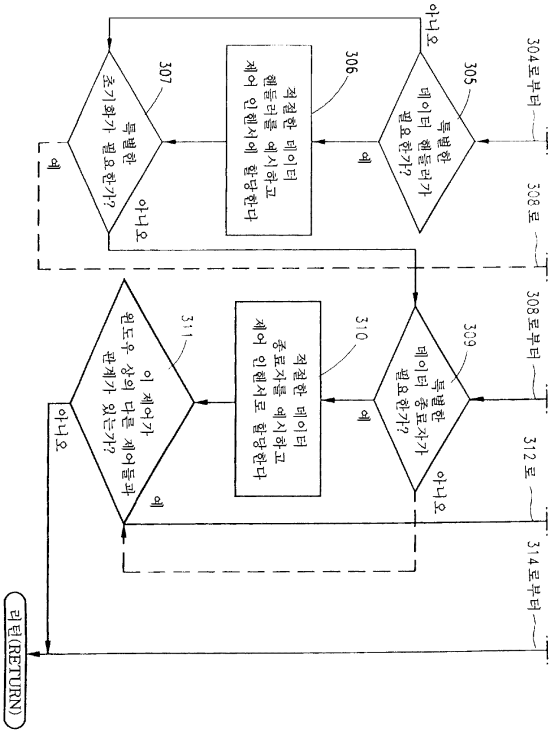
도면 2


119
관계
120
규칙 1
121
규칙 2
130
동작 1.1
131
동작 1.2
132
동작 2.1
133
동작 2.2

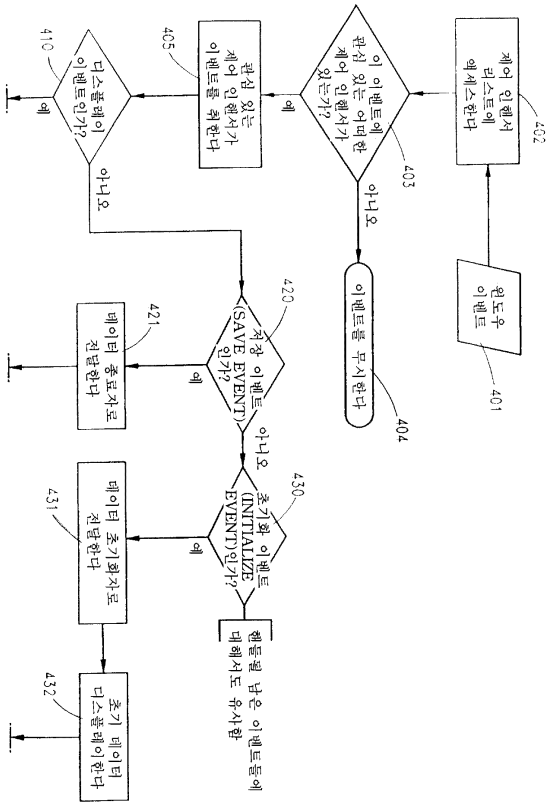
도면 3a


원도우 상에 GUI 제어를 생성한다
301
이 GUI 제어를 위한 제어 인핸서를 예시한다
302
제어 인핸서에 대한 GUI 제어에 대한 포인터를 전달한다
303
제어 인핸서 리스트에 추가한다
304
적절한 데이터 초기화자를 예시하고 제어 인핸서로 할당한다
308
각각에 대한 관계를 예시하고, 제어 인핸서를 할당하고, 영향받을 제어들로 제어 인핸서들에 대한 포인터들을 전달한다
312
적절한 규칙들을 예시하고 관계에 할당한다
313
적절한 동작들을 예시하고 규칙에 할당한다
314

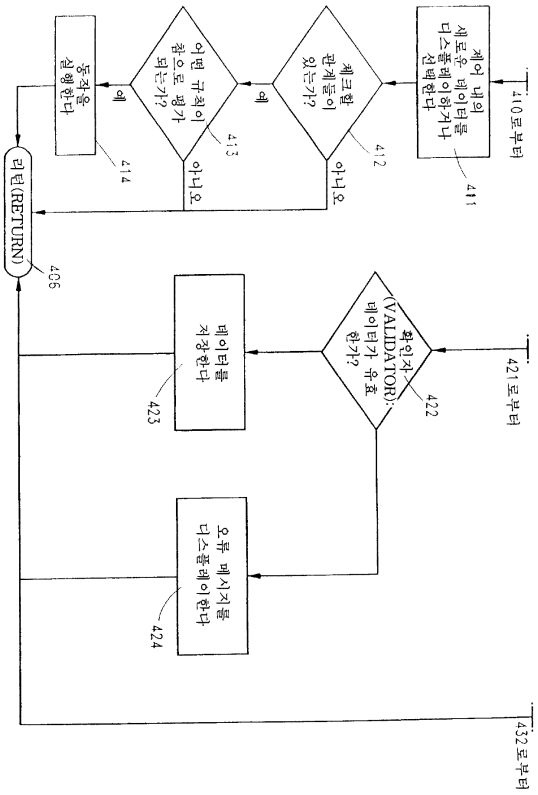
도면 3b


304로부터
305
특별한 핸들러가 필요한가?
아니오
예
306
적절한 데이터 핸들러를 예시하고 제어 인터페이스에 할당한다
307
특별한 초기화가 필요한가?
아니오
예
308로
308로부터
309
특별한 종료자가 필요한가?
아니오
예
310
적절한 데이터 종료자를 예시하고 제어 인터페이스로 할당한다
311
이 제어가 윈도우 상의 다른 제어들과 관계가 있는가?
아니오
예
312로
314로부터
리턴(RETURN)

도면 4a


원도우 이벤트
401
제어 인핸서 리스트에 엑세스한다
402
이 이벤트에 관심 있는 제어 인핸서가 있는가?
403
예
아니오
이벤트를 무시한다
404
관심 있는 제어 인핸서가 이벤트를 취한다
405
디스플레이 이벤트인가?
410
예
아니오
저장 이벤트(SAVE EVENT)인가?
420
예
데이터 종료자로 전달한다
421
아니오
초기화 이벤트(INITALIZE EVENT)인가?
430
예
데이터 초기화자로 전달한다
431
핸들될 수 있는 이벤트들에 대해서도 유사함
초기 데이터 디스플레이한다
432

도면 4b


410로부터
제어 내의
디스플레이할거나
선택한다
411
관계할
있는가?
412
아니오
예
어떤 규칙이
참으로 평가
되는가?
413
아니오
예
동작을
실행한다
414
리턴(RETURN)
416
421로부터
확인자
(VALIDATOR)
데이터가 유효
한가?
422
데이터를
저장한다
423
오류 메시지를
디스플레이한다
424
432로부터