(19)대한민국특허청(KR)
(12) 공개특허공보(A)

(51) 。Int. Cl.
G06F 3/03 (2006.01)
G06F 3/023 (2006.01)
G06F 3/033 (2006.01)
G06F 3/00 (2006.01)

(11) 공개번호 10-2007-0001223
(43) 공개일자 2007년01월03일

(21) 출원번호 10-2006-7021095
(22) 출원일자 2006년10월11일
심사청구일자 없음
번역문 제출일자 2006년10월11일
(86) 국제출원번호 PCT/IB2005/000614
국제출원일자 2005년03월11일
(87) 국제공개번호 WO 2005/088431
국제공개일자 2005년09월22일

(30) 우선권주장 10/800,204 2004년03월12일 미국(US)

(71) 출원인 반 데르 호에벤 스티브
모나코 모나코 98000 애비뉴 드 그랑드 브르타뉴 7/9

(72) 발명자 반 데르 호에벤 스티브
모나코 모나코 98000 애비뉴 드 그랑드 브르타뉴 7/9

(74) 대리인 김태홍
신정건

전체 청구항 수 : 총 23 항

(54) 데이터 입력 인터페이스 장치, 방법 및 시스템

(57) 요약

제한된 입력 채널의 세트를 갖는 장치 상에 효율적인 데이터 입력을 위한 장치, 방법 및 시스템을 개시한다. 이들 장치, 시스템 및 방법은 효율적인 데이터 입력을 가능하게 하는 사용자 인터페이스를 채용할 수 있다. 이 인터페이스는 그 인터페이스를 이용하는 장치에 따라 그리고 그 인터페이스와 연계된 내용적 의미에 따라 맞춤화될 수 있다. 효율적인 데이터 입력을 지원하기 위해, 이들 장치, 시스템 및 방법은 사용자 입력을 명확하게 하거나, 입력을 정확하게 지정할 수 있는 효율적인 방법을 제공하거나, 또는 사용자 입력을 예측할 수 있다. 또한, 이들 장치, 시스템 및 방법은 어떤 사용자의 경향에 적응할 수 있고, 빈번하게 사용되는 입력에 대해 사용자 정의된 축약어 및 동의어를 통합할 수 있다.

대표도

도 2

특허청구의 범위

청구항 1.

데이터 입력을 위한 인터페이스 방법으로서,

최초 누름을 검출하는 단계와;

누름 해제를 검출하는 단계와;

상기 누름과 누름 해제 사이의 움직임을 검출하는 단계로서, 하나 이상의 존 세트에 대하여 진입 또는 이탈하는 것을 검출하는 단계를 더 포함하는 움직임 검출 단계와;

상기 최초 누름, 움직임, 누름 해제를 개별 메시지로 정규화화는 단계

를 포함하는 데이터 입력 인터페이스 방법.

청구항 2.

제1항에 있어서, 상기 존 세트는 키 사이 존 세트와 키 존 세트를 포함하고, 2개의 키 존은 접촉하지 않으며, 각각의 키 존은 적어도 하나의 키 사이 존과 접촉하는 것인 데이터 입력 인터페이스 방법.

청구항 3.

제2항에 있어서, 상기 키 존은 행 세트에 배치되는 것인 데이터 입력 인터페이스 방법.

청구항 4.

제3항에 있어서, 상기 행 세트는 적어도 하나의 동심 곡선을 형성하는 것인 데이터 입력 인터페이스 방법.

청구항 5.

제3항에 있어서, 각각의 상기 행은 각 종단에서 키 존을 갖고, 상기 행의 각 키 존 사이에는 키 사이 존이 있는 것인 데이터 입력 인터페이스 방법.

청구항 6.

제5항에 있어서, 각각의 상기 키 사이 존은 그 키 사이 존이 접촉하는 적어도 2개의 인접 키 존과 겹치는 것인 데이터 입력 인터페이스 방법.

청구항 7.

제6항에 있어서, 각각의 상기 키 사이 존의 모든 부분은 그 부분이 접촉하는 적어도 2개의 인접 키 존의 하나와 연계되는 것인 데이터 입력 인터페이스 방법.

청구항 8.

제7항에 있어서, 상기 연계는 상기 움직임에 기초한 것인 데이터 입력 인터페이스 방법.

청구항 9.

제1항에 있어서, 상기 개별 메시지는 위치 및 방향을 포함하는 것인 데이터 입력 인터페이스 방법.

청구항 10.

제9항에 있어서, 내용적 의미(semantic meaning)를 상기 개별 메시지와 연계시키는 단계를 더 포함하는 데이터 입력 인터페이스 방법.

청구항 11.

제10항에 있어서, 상기 최초 누름은 제1 존에서 일어나고, 상기 누름 해제는 제2 존에서 일어나는 것인 데이터 입력 인터페이스 방법.

청구항 12.

데이터 입력을 위한 인터페이스 시스템으로서,

최초 누름을 검출하고, 누름 해제를 검출하며, 상기 누름과 누름 해제 사이의 움직임을 검출하도록 동작하는 센서로서, 상기 움직임의 검출은 하나 이상의 존 세트에 대하여 진입 또는 이탈하는 것의 검출을 더 포함하는 것인 센서와;

상기 최초 누름, 움직임, 누름 해제를 개별 메시지로 정규화하도록 동작 가능한 로직

을 포함하는 데이터 입력 인터페이스 시스템.

청구항 13.

제12항에 있어서, 상기 존 세트는 키 사이 존 세트와 키 존 세트를 포함하고, 2개의 키 존은 접촉하지 않으며, 각각의 키 존은 적어도 하나의 키 사이 존과 접촉하는 것인 데이터 입력 인터페이스 시스템.

청구항 14.

제13항에 있어서, 상기 키 존은 행 세트에 배치되는 것인 데이터 입력 인터페이스 시스템.

청구항 15.

제14항에 있어서, 상기 행 세트는 적어도 하나의 동심 곡선을 형성하는 것인 데이터 입력 인터페이스 시스템.

청구항 16.

제14항에 있어서, 각각의 상기 행은 각 종단에서 키 존을 갖고, 상기 행의 각 키 존 사이에는 키 사이 존이 있는 것인 데이터 입력 인터페이스 시스템.

청구항 17.

제16항에 있어서, 각각의 상기 키 사이 존은 그 키 사이 존이 접촉하는 적어도 2개의 인접 키 존과 겹치는 것인 데이터 입력 인터페이스 시스템.

청구항 18.

제17항에 있어서, 각각의 상기 키 사이 존의 모든 부분은 그 부분이 접촉하는 적어도 2개의 인접 키 존의 하나와 연계되는 것인 데이터 입력 인터페이스 시스템.

청구항 19.

제18항에 있어서, 상기 연계는 상기 움직임에 기초한 것인 데이터 입력 인터페이스 시스템.

청구항 20.

제12항에 있어서, 상기 개별 메시지는 위치 및 방향을 포함하는 것인 데이터 입력 인터페이스 시스템.

청구항 21.

제20항에 있어서, 상기 로직은 내용적 의미를 상기 개별 메시지와 연계시키도록 동작 가능한 것인 데이터 입력 인터페이스 시스템.

청구항 22.

제21항에 있어서, 상기 최초 누름은 제1 존에서 일어나고, 상기 누름 해제는 제2 존에서 일어나는 것인 데이터 입력 인터페이스 시스템.

청구항 23.

데이터 입력을 위한 인터페이스 시스템으로서,

최초 누름을 검출하고, 누름 해제를 검출하며, 상기 누름과 누름 해제 사이의 움직임을 검출하도록 동작하는 센서로서, 상기 움직임의 검출은 하나 이상의 존 세트에 대하여 진입 또는 이탈하는 것의 검출을 더 포함하는 것인 센서와;

상기 최초 누름, 움직임, 누름 해제를 각각의 존과 연계된 콘텍스트에 기초하여 내용적 의미로 정규화하도록 동작 가능한 로직

을 포함하는 데이터 입력 인터페이스 시스템.

명세서

기술분야

관련 출원

본 출원은 반 데르 호에벤(Van der Hoeven)이 2004년 3월 12에 출원한 발명의 명칭이 "Method and System for Disambiguation and Predictive Resolution"인 미국 특허 출원 번호 10/800,203에 관한 것이며, 이 단락 내에 인용하는 모든 출원은 참조문헌으로서 본 명세서에 그 전체가 포함된다.

본 발명은 개괄적으로 데이터 입력에 관한 것이며, 보다 구체적으로는 장치에의 데이터 입력을 효율적으로 하기 위한 인테페이스 및 로직에 관한 것이다.

배경기술

컴퓨터 시대가 도래한 이래로 가장 변하지 않으면서 계속되는 경향 중 하나가 소형화 지향이다. 이러한 경향은 모든 산업의 양상에서 나타나고 있다. 휴대 전화, 개인 휴대 정보 단말기(PDA; Personal Digital Assistant), 휴대용 컴퓨터 등은 모두 놀라운 속도로 사이즈가 축소되고 있다. 그러나, 반대로 이들 장치가 축소화됨에 따라 또한 이들은 보다 강력해지고 있다. 이러한 양분 현상은 다소 문제가 된다. 이 장치와 함께, 디스플레이 장치 및 데이터 입력 인터페이스가 이용할 수 있는 실 면적이 자연히 축소되면서, 사용자가 정보를 전달하고 정보를 수집할 필요성도 장치의 능력과 비례하게 증가한다.

데이터 입력 및 디스플레이에 대한 2개의 하위시스템 각각에 있어서, 각 하위시스템이 비효율적이거나 무용하게 되기 전에 가능한 사이즈 축소에 대한 본래의 물리적인 제한이 있다. 사용자에게 통신에 대해 부과되는 물리적인 제한은 데이터 입력 인터페이스에 대한 부과되는 것보다 훨씬 많다. 이러한 제한은 하드웨어 제한, 생리적 제한, 및 대부분의 경우 데이터를 사용자에게 시각적으로 전달되어야 한다는 점을 포함할 수 있다. 이에, 통신 및 데이터 입력을 위해 예약된 장치의 공간 크기(footprint)의 대부분을 시각적 디스플레이 장치가 차지하게 된다. 이러한 시각적 디스플레이 장치는 데이터 입력 인터페이스를 위한 공간을 거의 남기지 않는다.

이러한 문제를 해결하기 위한 방안이 수기(handwriting) 소프트웨어를 장치에 통합함으로써 고안되었다. 이러한 방안을 이용하기 위해서, 사용자는 접촉식 디스플레이 장치 상에 기록함으로써 텍스트를 직접 입력할 수 있다. 그리고, 이렇게 수기 작성된 데이터는 수기 인식 소프트웨어에 의해 디지털 데이터로 변환된다. 이러한 방안에 의해 하나의 디스플레이 장치가 사용자에 대한 정보 전달 및 데이터 수신을 모두 할 수 있지만, 수기 인식 소프트웨어의 정확성 및 속도는 알려진 바와 같이 나쁘다. 또한, 스타일러스(stylus)를 이용한 인쇄 또는 기록은 일반적으로 타이핑, 즉 키를 이용한 입력보다 속도가 느리며, 2개의 손을 이용해야 한다.

그에 따라, 한 손으로 작업할 수 있는 소량의 데이터를 입력하기 위한 인터페이스를 이용하는 것이 바람직하다. 그러한 키보드에서의 종래의 시도에 따라 키 수가 축소된 키보드가 개발되었다. 도 1은 종래의 장치용의 축소형 키패드(100)를 도시하고 있다. 키패드(100)는 10개의 숫자키(110-128), 별표(*)키(130), 파운트(#)키(140)를 갖는 표준형 전화 키패드이다. 영어 및 그외 많은 알파벳 언어의 경우, 전화기의 숫자 키패드가 알파벳 키패드와 오버레이되어 3개의 문자가 각 숫자키(110-128)과 연계된다. 예컨대, 숫자 2 키(114)는 문자 a-b-c와 연계된다. 이들 축소형 키 장치 중 일부를 이용함에 있어서, 사용자는 복수개의 키를 눌러서 단어 또는 이름을 입력할 수 있다.

그러나, 이 방안에 있어서도, 중국어 및 일본어 간지 등의 다른 문자계 언어의 경우 숫자 키패드 상에 오버레이될 수 있는 처리 가능한 문자 수를 가질 수 없기 때문에 문제가 된다. 추가 문제는 각각의 키 누름이 복수의 문자를 포함할 수 있어, 키스트로크의 시퀀스에 중의성이 있을 수 있다는 것이다.

이 중의성 문제에 대한 방안은 각 문자를 지정하기 위해 사용자가 2회 이상의 키스트로크를 행해는 것이며, 키스트로크는 동시에(코딩(chording)) 또는 순서대로 행해질 수 있다. 그러나, 이들 방법은 다수회의 키스트로크를 필요로 하기 때문에 비효율적이다. 이상적으로 데이터 입력 방법은 문자당 1 키스트로크를 필요로 한다.

종래에, 단어 레벨의 중의성 해소(disambiguration)는 수신된 키스트로크의 시퀀스를 사전에 있는 필적할 만한 것과 비교함으로써 단어 전체를 명확하게 하는데 이용되고 있다. 그러나, 디코딩의 제한 때문에, 단어 레벨의 중의성 해소로, 문자당 1 키스트로크의 효율성을 갖는 구속없는 데이터의 디코딩에 에러가 없을 수는 없다.

따라서, 각종 데이터의 효율적인 한 손 입력을 가능하게 하며, 문자 중의성의 해소 및 예측 입력 완성을 위한 로직을 이용하여 데이터 입력에 필요한 스트로크 수를 단축할 수 있는 능력을 가능하게 하는 인터페이스 장치, 방법 및 시스템이 필요하다.

발명의 상세한 설명

제한된 입력 채널의 세트를 구비한 장치 상에 효율적인 텍스트 입력을 위한 장치, 방법 및 시스템을 개시한다. 이들 장치, 시스템 및 방법은 한 손의 엄지로 조작될 수 있는 키 레이아웃을 이용하여 효율적인 한 손 작업이 이루어지는 사용자 인터페이스를 채용할 수 있다. 이 인터페이스는 그 인터페이스를 이용하는 특정 장치에 맞춤화될 수 있으며, 어떤 경우에는 복수의 문자가 각 키와 또는 키의 일부와 연계될 수 있다. 데이터의 효율적인 입력을 지원하기 위하여, 이들 장치, 시스템 및 방법은 사용자가 입력한 데이터가 여러개의 해석을 가질 수 있는 경우에 사용자 입력을 명확하게 할 수 있거나, 입력을 정확하게 지정할 수 있는 효율적인 방법을 제공하거나, 또는 사용자 입력을 예측하여 이러한 예측 결과를 사용자에게 제시할 수 있다. 또한, 이러한 예측 시스템 및 방법은 어떤 사용자의 경향에 적응할 수 있고, 빈번하게 사용되는 입력에 대한 생략, 변환, 동의어를 포함할 수 있다.

일 실시예에서, 최초 누름이 검출되고, 누름 해제가 검출되며, 누름과 누름 해제 간의 움직임이 존 세트를 이용하여 검출되고, 최초 누름, 누름 해제 및 움직임이 정규화되어 개별 메시지가 된다.

다른 실시예에서, 존 세트는 키 사이 존 세트과, 키 존 세트를 포함하고, 2개의 키 존은 접촉하지 않으며, 각각의 키 존은 적어도 하나의 키 사이 존과 접촉한다.

또 다른 실시예에서, 존 세트는 행 세트에 배치되어 있다.

또 다른 실시예에서, 각 행은 각 단부에 키 존을 갖고, 그 행에 있는 각각의 키 존 사이에는 키 사이 존이 있다.

또 다른 실시예에서, 각각의 키 사이 존은 그 키 사이 존이 접촉하는 적어도 2개의 인접 키 존과 겹쳐진다.

또 다른 실시예에서, 각각의 키 사이 존의 모든 부분은 그 키 사이 존이 접촉하는 적어도 2개의 인접 키 존 중 하나와 연계된다.

일부 실시예에서는 내용적 의미(semantic meaning)가 개별 메시지와 연계된다.

본 발명의 전술한 양상 및 다른 양상들은 이어지는 설명 및 첨부하는 도면을 참조하여 보다 더 잘 알게 될 것이다. 다음의 설명에 있어서, 본 발명의 다양한 실시예 및 수많은 특정 상세를 나타내는 이어지는 설명은 예시적이며, 제한적이지 않다. 다양한 대체, 변경, 추가 또는 재구성이 본 발명의 기술적 사상 내에서 이루어질 수 있으며, 본 발명은 이러한 모든 대체, 변경, 추가 또는 재구성을 포함한다.

실시예

본 발명 및 본 발명의 다양한 특징 및 유용한 상세 내용은 첨부하는 도면에 도시하며 이어서 상세히 설명하는 비제한적인 실시예를 참조하여 보다 충분하게 설명한다. 잘 알려진 개시 요소, 처리 기술, 구성 및 장비에 대한 설명은 편의상 생략하였다. 그러나, 당업자라면, 본 발명의 양호한 실시예를 개시하지만, 상세한 설명 및 특정예는 예시적인 것일 뿐 제한적이지 않다는 것을 이해하여야 한다. 본 발명의 기본 원리의 범주 내에서 다양한 대체, 변경, 추가 또는 재구성이 본 명세서를 이해한 후에 당업자에게 분명해질 것이다.

이제 본 발명의 예시적인 실시예에 대해 상세히 설명하기로 하며, 이 예들은 첨부 도면에 도시되어 있다. 가능하다면, 동일한 도면 부호는 도면 전체를 통해 동일하거나 유사한 부분(요소)을 나타내도록 사용될 것이다.

본 명세서 전체를 통해 사용되는 용어의 이해를 돕기 위해 몇가지 용어를 정의하여 분명하게 하였다.

용어 "문자(character)"는, 문자가 글자(letter), 판별 가능한 마크, 간지의 일부, 필적(writing) 및 인쇄 등의 분야에서 함축적 의미를 가진 기호(symbol)이든지 임의의 기호를 의미하는 것이다.

용어 "키(key)"는 터치를 비롯한 사용자 동작에 응답하는 인터페이스의 존을 의미하는 것이다. 내용적 의미는 키 또는 키의 일부와 연계될 수 있으며, 이 내용적 의미는 하나 이상의 문자로 구성되거나 정보를 제어할 수 있다.

용어 "키보드(keyboard)"는 키의 그룹화를 의미하는 것이다.

용어 "예측 결과(prediction)"란 기호 시퀀스의 완성을 예측하려는 시도의 결과를 의미하는 것이다. 이 예측 결과는 개연적, 수학적, 언어적 또는 그외의 것을 포함하는 임의의 타입의 시스템에 기초할 수 있다.

용어 "동의어(synonym)"는 예측 결과의 가능한 대안을 의미하는 것이다. 이들 동의어는 예측 결과와 연계될 수 있으며, 예측 결과에 대응하는 데이터 구조에, 즉 예측 결과에 대응하는 단말 노드에 기억될 수 있다.

용어 "제안(proposition)"은 하나 이상의 문자의 시퀀스, 즉 이러한 시퀀스의 세트를 의미하는 것이다.

용어 "프리픽스(prefix)"는 사용자 입력, 중의성 해소의 결과 또는 예측 결과 등으로 확장되거나 될 수 없는 하나 이상의 문자의 시퀀스를 의미한다.

용어 "후보(candidate)"는 하나 이상의 문자의 시퀀스 및 관련 평점(score)을 의미하는 것이다. 이 평점은 개연적, 수학적, 언어적 또는 그외의 것을 포함하는 임의의 모델에 기초할 수 있다.

용어 "활성 존(active zone)"은 사용자로부터 입력을 수신하는데 이용된 인터페이스의 부분으로 지정되는 인터페이스의 존을 의미하는 것이다.

용어 "존(zone)"은 인터페이스의 영역 또는 센서의 범위 내에서의 영역을 의미하는 것이다.

용어 "중의적이다(ambiguous)"는 어떤 것이 몇가지 가능한 의미를 가질 수 있음을 의미하는 것이다.

용어 "완성(completion)"은 최초의 기호의 세트(보통 프리픽스라고 함)와 연결되는 경우에 원하는 최종적 또는 중간적 기호의 시퀀스를 형성할 수 있는 기호의 세트를 의미한다. 많은 경우에, 완성은 원하는 최종적 또는 중간적 시퀀스를 형성할 가능성이 높도록 계산되는 것이 좋다.

용어 "메시지 발송(send a message)"은 정보의 전송, 교환 또는 사용을 의미하는 것이다. 중요한 것은, 메시지가 어떻게 발송되는지에 대해서는, 즉 메시지의 발송이 임의의 특정 실시예에서 필요할 경우에도, 고려할 필요가 없다는 점이다.

용어 "시퀀스(sequence)"는 순서의 개념이 있는 유한 개수의 요소를 의미하는 것이다. 시퀀스는 비어있거나 단 하나의 요소만 포함할 수 있다. 이것은 수학에서 일반적으로 인정되는 정의이다. 우리는 그것을 $(e_1,\cdots,e_n)$로서 나타내기로 한다.

용어 "세트(set)"는 순서의 개념이 없는 유한 개수의 요소를 의미하는 것이다. 세트는 비어있거나 단 하나의 요소만 포함할 수 있다. 이것은 수학에서 일반적으로 인정되는 정의이다. 우리는 그것을 $\{e_1,\cdots,e_n\}$로서 나타내기로 한다.

용어 "타이프(type)"는 값 또는 등급의 범위를 의미하는 것이다. 변수 또는 값을 타이프하는 것은 그것을 열거하는 것이며, 즉 그것의 값 또는 등급을 지정하는 것이다. 타입 T를 갖는 요소 E의 타이프는 다음과 같이 "E:T"로서 작성된다.

용어 "구조(structure)"는 정렬된 요소의 세트 $(e_1,\cdots,e_n)$를 의미하는 것이다. 각각의 요소 e_i는 타입 t_i를 갖게 타이프될 수 있다. 그리고 구조의 타입은 시퀀스 (t1,…,tn)이다.

용어 "필드(field)"는 구조의 요소를 의미하는 것이다. 필즈(fields)라고 명명될 수 있다. P(e:t)는 구조 P가 타입 t의 필드 e를 가짐을 의미한다. p의 타입이 P라면(전술한 바와 같이), p.e는 요소 p의 필드를 나타낸다.

이제, 제한된 입력 채널의 세트를 갖는 장치 상에 텍스트의 효율적인 입력을 위한 장치, 방법 및 시스템에 대해 설명하기로 한다. 이들 장치, 방법 및 시스템은 한 손의 엄지로 조작될 수 있는 키 레이아웃을 이용하여 효율적인 한 손 작업을 가능하게 하는 사용자 인터페이스를 채용할 수 있다. 이 인터페이스는 이 인터페이스를 이용하는 특정 장치에 맞게 맞춤화될 수 있으며, 어떤 경우에는 복수의 문자가 각 키 또는 키의 부분과 연계될 수 있다. 데이터의 효율적인 입력을 지원하기 위해,

이들 장치, 방법 및 시스템은 사용자가 입력한 데이터가 몇가지 해석을 가질 수 있는 경우에 이들 키로부터 사용자 입력을 명확하게 할 수 있는데, 즉 사용자가 입력을 정확하게 지정할 수 있는 효율적인 방법을 제공할 수 있다. 또한, 이들 시스템 및 방법은 사용자가 전체 텍스트를 수동으로 입력할 것을 요구하는 대신에, 사용자 입력을 예측하여 사용자가 선택할 수 있도록 이 예측 결과를 사용자에게 제시할 수 있다. 또한, 이 예측 시스템 및 방법은 어떤 사용자의 경향에 맞게 적응할 수 있고, 빈번하게 사용된 입력에 대해서는 사용자 정의된 축약어 및 동의어를 통합할 수 있다.

본 발명의 예시적인 실시예에 있어서, 컴퓨터 실행 가능한 명령어는 어셈블리 코드 또는 컴파일 C++, 자바 또는 그외 언어 코드의 라인일 수 있다. 다른 아키텍쳐도 이용할 수 있다. 또한, 그러한 코드를 갖는 컴퓨터 프로그램 또는 소프트웨어 구성요소는 복수의 컴퓨터에서 복수의 컴퓨터 시스템 판독 가능 매체에 구현될 수 있다.

이제 도 2를 참조하면, 본 명세서에 개시하는 장치, 방법 및 시스템을 구현하여 편성하기 위한 아키텍쳐(200)의 일실시예가 도시된다. 본 명세서를 이해한 후에는, 그외 많은 아키텍쳐가 본 발명의 실시예로 이용될 수 있음을 이해할 것이다. 일실시예에서, 이들 아키텍쳐는 본체 센서(210), 추가 센서(212), 키스트로크 모듈(220), 의미화 모듈(240), 코어 로직(250), 언어 모델(260) 및 용법 모델(270)을 포함한다. 키스트로크 모듈(220)은 본체 센서(210)로부터 데이터를 수신할 수 있다. 키스트로크 모듈(220)은 이 데이터를 처리한 다음, 그 결과를 의미화 모듈(240)에 보낸다. 의미화 모듈(240)은 키스트로크 모듈(220)로부터 그리고 임의의 추가 센서(212)로부터 결과를 취득한다. 그런 다음, 의미화 모듈(240)은 사용 콘텍스트(290), 텍스트 요소(280), 메뉴 요소(282) 및 출력 장치(284)를 이용하여 코어 로직(250)에 의해 사용되도록 이 데이터를 처리하여 데이터를 형식화할 수 있다. 코어 로직(250)은 의미화 모듈(240), 언어 모델(260) 또는 용법 모델(270)로부터의 입력을 이용해, 사용자의 입력을 예측하여 완성하거나 중의성을 해소할 수 있다. 코어 로직(250)은 그 결과를 의미화 모듈(240)에 제공한다. 그리고 의미화 모듈은 사용 콘텍스트(290), 텍스트 요소(280), 메뉴 요소(282) 및 출력 장치(284)의 조합을 통해 결과를 출력할 수 있다.

일반적으로, 이 아키텍쳐는 한 손의 엄지로 사용하도록 설계된 구성을 비롯하여, 입력 장치에 대해 다양한 구성을 가능하게 함으로써 사용자가 장치 상에 데이터를 입력할 수 있는 능력을 강화시키도록 설계된다. 또한, 실시예들은 입력을 명확하게 하고, 사용자에게 정확한 입력을 명확하게 지정할 수 있는 효율적인 방법을 제공하며, 추가 수동 입력을 피하기 위하여 사용자에게 예측 완성을 제시할 수 있다. 당업자라면, 본 명세서를 이해한 후에, 이 아키텍쳐는 설명의 용도로만 제시된 것일 뿐이며, 설명하는 모든 기능성은 더 많거나 더 적은 모듈을 포함하는 다양한 방식으로, 전달받은 메시지가 있거나 없거나, 소프트웨어, 하드웨어 등으로 구현될 수 있음을 이해할 것이다.

특정 실시예에서, 사용자의 동작은 센서(210, 212)에 의해 포착되어 의미화 모듈(240)에 통보된다. 일부 실시예에서, 본체 센서(210)로부터의 정보는 의미화 모듈(240)에 통보되기 전에 키스트로크 모듈(220)에 의해 처리될 수 있다. 의미화 모듈(240)은 사용자에 대한 유일한 인터페이스일 수 있으며, 모든 입력 및 출력 메시지를 변환 및 형식화하는 데에 책임이 있을 수 있다. 의미화 모듈(240)의 변환은 콘텍스트 종속적일 수 있거나 시스템으로의 통합을 위해 파라미터화(parameterization)를 제공할 수 있다. 코어 로직(250)은 사용자 입력의 예측 및 중의성 해소를 비롯하여, 의미화 모듈(240)과 사용자 간의 상호 작용을 관리하기 위한 원칙(rule)을 포함할 수 있다.

코어 로직(250)은 사용자 입력을 명확하게 하기 위한 언어 모델(260) 및 사용자 입력을 예측할 수 있는 사용자 모듈(270)을 비롯한 데이터 구조를 사용할 수 이다. 텍스트 모듈(280), 메뉴 요소(282) 및 그외 선택적 출력 장치(287)가 의미화 모듈(240)에 의해 사용되어 코어 로직의 결과 또는 시스템의 상태를 나타낼 수 있다. 텍스트 요소(280)가 데이터 입력 인터페이스일 수 있기 때문에 사용자는 기호 시퀀스를 삽입하거나 입력할 수 있거나 기존의 기호 시퀀스를 확장할 수 있고, 기호 시퀀스가 텍스트 요소를 통해 표시될 수 있다.

일 특정 실시예에서, 텍스트 요소(280)는 사용자에 의해 기호 시퀀스를 삽입하거나 기존의 시퀀스를 확장할 수 있는 장치이다. 텍스트 요소(280)는 가능한 프리픽스 또는 다른 사용자 입력 텍스트에 대한 정보를 포함하는, 메시지(EditText)를 시스템 또는 의미화 모듈(240)에 보낼 수 있다. 이 때, 텍스트 요소(280)는, EditText 메시지에 응답하여 의미화 모듈(240)로부터 기호 시퀀스(S)를 포함하는 SubstituteText 메시지를 수신한다. 또한, 텍스트 요소(280)는 사용자가 기호 시퀀스의 입력을 종료하는 경우에 EditText 메시지의 완성을 신호로 전송하는 EndEdit 메시지를 수신할 수 있다. 텍스트 요소(280)가 이 SubstituteText 메시지를 수신하는 경우에, 텍스트 요소(280)는 그에 따라 응답할 수 있다. 만약 그 메시지가 최종 EditText가 보내진 후에 수신된 첫번째 SubstituteText 메시지이고 그 최종 EditText가 프리픽스를 갖고 있다면, SubstituteText 인수는 텍스트 요소(280) 내의 프리픽스에 대한 기호 시퀀스에 치환된 다음에 사용자에게 표시된다. 만약 그 메시지가 최종 EditText가 보내진 후에 수신된 첫번째 SubstituteText 메시지이고 그 최종 EditText가 프리픽스를 갖고 있지 않다면, 텍스트 요소(280)는 사용자에게 표시된 시퀀스에 SubstituteText 인수를 단지 삽입할 수 있다. 2개의 연속 SubstituteText 메시지에 있어서, 그 사이에 EndEdit가 없다면, 텍스트 요소는 제1 SubstituteText 메시지의 텍스트 대신에 제2 SubstituteText의 인수를 대체하여 그 정보를 사용자에게 표시한다.

메뉴 요소(282)는 시스템의 출력 또는 상태에 따라, 사용자에게 일련의 선택 또는 옵션을 제시할 수 있다. 메뉴 요소(282)는 일부 실시예에서는 텍스트 요소(280)와 조합될 수 있다. 메뉴 요소(282)는 사용자가 선택할 수 있도록 선택 리스트 또는 선택 트리를 표시할 수 있다. 메뉴 요소(282)는 의미화 모듈(240)로부터 인수를 포함하는 SetMenu 메시지를 수신할 수 있다. 이 인수는 메뉴가 리스트인 경우에 단순 선택 리스트(flat list)일 수 있고, 트리 메뉴인 경우에 트리 또는 내포 선택 리스트(nested list)에서의 선택의 위치를 나타내는 인수일 수 있다. 선택은 기호 또는 상수의 시퀀스일 수 있다. 사용자가 제시된 리스트로부터 선택하는 경우, SelectItem 메시지가 인수로서 선택치와 함께 의미화 모듈(240)에 보내질 수 있다.

사용 콘텍스트(290)는 센서(210, 212)의 특성 및 레이아웃을 제어하고 사용자에게 소정의 이벤트를 통지할 수 있다. 일 실시예에서, 사용 콘텍스트(290)는 TextInputQuery를 이용하여 사용자와의 상호 작용을 위해 텍스트 요소(280) 또는 메뉴 요소(282)를 준비할 수 있다. 텍스트 요소(280)는 그 텍스트 요소(280)가 편집이 준비됨을 나타내는, QueryFlag와 EditText 메시지를 의미화 모듈에 출력함으로써 응답할 수 있다.

또한, 사용 콘텍스트(90)는 편집 시작 시에 StartEdit 메시지를, 또는 편집 정지 시에 EditStop 메시지를 인터셉트하여 텍스트 편집이 텍스트 요소(280)에서 수행될 때를 알 수 있다. 또한, 사용 콘텍스트(290)는 본체 센서(210)의 다양한 특성을 제어하고 사용자에게 소정의 이벤트를 통지할 수 있다.

이들 이벤트는 통지의 타입에 관한 정보를 포함할 수 있는 UserNotify 메시지를 통해 트리거될 수 있다. 시스템은 각 타입의 통지마다 감당해야 할 동작을 해석할 수 있다. 이 해석은 의미화 모듈(240)에 의해 행해질 수 있다.

본체 센서(210)는 환경에서 변수의 세트를 모니터하고 그 결과를 개별 메시징을 통해 키스트로크 모듈(220)에 보낼 수 있다. 또한, 본체 센서(210)는 본체 센서(210)의 범위 내에서 사용자가 특정 영역 지정하기, 사용자에 의해 이루어진 일련의 움직임, 및 사용자의 움직임 정지를 포함할 수 있는, 사용자의 행동을 검출할 수 있다. 대개 이러한 행동들은 사용자와 센서 간의 직접적인 물리 접촉, 스타일러스, 펜 등의 툴을 이용한 간접적인 접촉, 또는 눈 움직임, 레이저 포인터 등의 물리적 접촉을 수반하지 않는 방법을 수반한다. 당업자라면 본체 센서(210)를 구현하는데 다양한 옵션이 존재함을 인식할 것이다. 이러한 옵션의 예로는 눈 추적 시스템, 터치 스크린, 조이스틱, 그래픽 사용자 인터페이스, 전자 또는 기계식 등의 모든 전통적 키보드가 있다. 마찬가지로, 이들 감지 장치는 하드웨어로, 소프트웨어로, 또는 이들의 조합으로 구현될 수 있다. 또한, 본 명세서를 이해한 후에는, 다양한 레이아웃이 본체 센서(210)로 이용되어 데이터 입력 인터페이스를 제공할 수 있음이 분명해질 것이다.

아키텍처(200)의 구성요소의 상호 작용을 설명하기 전에, 접촉을 검출할 수 있는 본체 센서(210)의 타입으로 이용될 수 있는 레이아웃의 일례가 도 3에 도시되어 있다. 본체 센서(210) 내에는 겹치지 않는 존(300-390)의 세트가 정의될 수 있다. 이들 존(300-390)은 사용자와의 상호 작용을 돕기 위해 설계된 키 존(300, 310-390)일 수 있다. 이들 키 존(300, 310-390)은 하나 이상의 행(312, 314)을 따라 배열될 수 있으며, 각 행은 하나 이상의 키 존(300, 310-390)을 포함하며, 행은 동심 곡선에 순서대로 배열될 수 있다. 키 존(300, 310-390)의 행(312, 314) 외부에 있는 존은 키가 없는 존(302)이다. 각각의 키 존(300, 310-390)은 고유 식별자와 연계되고, 이 고유 식별자는 행 사이에서 키 존(300, 310-390)의 위치를 수단으로 하여 키 존(300, 310-390)의 정렬을 나타낸다.

예컨대, 행(312, 314)은 최내측 행에서 시작하여 올림차순으로 번호가 매겨질 수 있다. 도 3에 나타내는 예에서, 행(314)은 행 번호 1이 되고, 행(312)은 행 번호 2가 된다. 또한, 키 존(300, 310-390)은 그 각각의 행 내에서 정렬될 수 있다. 도 3에서 번호가 좌에서 우로 키 존(300, 310-390)에 지정되어, 그 결과 존(300, 350)은 1로 번호 매겨지고, 존(320, 360)은 2로, 등등 이런 식으로 번호 매겨질 수 있다. 이들 행과 존 정렬을 이용하여, 고유 식별자는 본체 센서(210)를 이용하여 구현되는 각 존마다 어셈블될 수 있다. 이 예에서, 키 존(300)은 (2,1)이 되고, 키 존(350)은 (1,1)로서, 키 존(360)은 (1,2)로서, 등등 이런 식으로 식별될 수 있다. 당업자에게는 쉽게 이해되겠지만, 본체 센서(210) 내에서 존(300-390)의 세트를 고유하게 식별하는데 많은 방식을 이용할 수 있으며, 이들 존(300-390)은 사용자, 장치, 특정 센서의 사이즈 또는 그외 다른 기준에 적합할 수 있는 각종 토폴로지로 정의될 수 있다.

본체 센서(210)는 환경을 모니터하여 사용자에 의한 접촉이 이루어지고 있는 겨냥 존 또는 접촉 존을 결정할 수 있다. 이 겨냥 존은 접촉이 이루어지는 점 또는 위치만으로 구성될 수 있거나, 실제 접촉 위치를 둘러싸는 정의된 영역으로 구성될 수 있다. 접촉이 이루어지는 경우, 본체 센서(210)는 존(300-390)이 겨냥 존과 겹치는지의 여부를 결정할 수 있다. 겨냥 존과 겹치는 각각의 존(300-390)에 대하여, 본체 센서(210)는 메시지를 키스트로크 로직(220)에 보낸다. 각 메시지는 접촉 개시 및 겹치는 존(300-390)에 대한 각각의 고유 식별자를 식별하는 메시지(StartContact)일 수 있다. 겨냥 존이 이동하여 임의의 다른 존(300-390)에 겹치기 시작하는 경우, 존(300-390)이 그 존(300-390)의 고유 식별자와 함께 진입하고

있음을 나타내는 메시지가 키스트로크 로직(220)에 보내진다. 접촉이 중지하면, 본체 센서(210)는 접촉이 중지되었음을 감지하고, 이것을 키스트로크 모듈(220)에 알린다(StopContact). 따라서, 본체 센서는 접촉의 개시, 이 접촉의 움직임, 및 접촉의 해지를 감지할 수 있다. 이러한 타입의 메시지는 본체 센서(210)와 키스트로크 모듈(220) 사이에서 전술한 타입의 메시지를 이용하여 교환될 수 있다. 모든 타입의 접촉이 본체 센서(210)에 의해 결정될 수 있지만, 이러한 모든 접촉에 관한 메시지는 키스트로크 모듈(220)에 전달될 수 없으며, 즉 사용자의 접촉을 평가하는데 사용될 수 있다는 점이 중요하다.

존(300-390)은 그 존(300-390)의 레이아웃, 사용자의 사양, 존(300-390)과 연계된 내용적 의미 등에 따라 활성 또는 비활성으로서 지정될 수 있다. 존(300-390)이 비활성으로서 지정되는 경우, 본체 센서(210)는 비활성 존과 부합하는 본체 센서의 영역을 겨냥 존과 비교할 수 없기 때문에, 키스트로크 로직(220)에 보내질, 비활성 존의 고유 식별자를 포함하는 메시지는 없을 것이다. 마찬가지로, 키스트로크 로직(220)에 보내질 비활성 존의 고유 식별자를 포함하는 메시지는 없을 것이지만 비교는 행해질 수 있다.

키스트로크 로직(220)은 본체 센서(210)로부터 일련의 접촉 개시/정지 메시지를 구별하는 역할을 할 수 있다. 키스트로크 로직(220)은 본체 세서(210)로부터 생성된 메시지를 취득하여, 접촉이 일어나고 있는 키 존(300, 310-390)의 행 번호와, 그 행에 있는 키 존(300, 310-390)의 번호를 결정하기 위해 이들 메시지를 해석할 수 있다. 또한, 키 존(300, 310-390)은 사용자의 접촉의 움직임 또는 사용자의 해제점에 따라 여러개의 모드로 트리거될 수 있고, 키스트로크 로직은 키 존(300, 310-390)이 트리거되는 모드를 결정할 수도 있다. 이들 모드는 KeyCenter, KeyLower, KeyUpper, RowLower 및 RowUpper로 표시될 수 있다. 이들 메시지는 사용자와의 접촉이 최초로 개시된 위치에 관하여 사용자의 해제점에 대응할 수 있다.

일 실시예에서, 본체 센서(210)로부터의 메시지는 다음의 알고리즘에 따라 처리될 수 있다.

StartContact 위치에 키가 없다면, 키스트로크 모듈은 임의의 다른 활성없이 다음 StartContact 메시지를 기다린다.

2개의 변수, Row와 Key가 있다. 그 각각은 다음의 값, Upper, Lower 및 Same을 가질 수 있다.

각 StartContact($k_{r1,n1}$)에 대해서, (r1은 키가 속하는 행이고, n1은 그 행에서의 키 번호이다):

Row를 Same으로, Key를 Same으로 설정한다.

각 EnterZone($k_{r2,n2}$)에 대하여:

r2>rl이면, Row를 Upper로 설정한다

r2<rl이면, Row를 Lower로 설정한다

r2=rl이면:

Row를 Same으로 설정, 그리고

n2>nl이면 Key를 Upper로 설정한다

n2<nl이면 Key를 Lower로 설정한다

n2=nl이면 Key를 Same으로 설정한다

StopContact에 대하여:

Row가 Same이면:

Key가 Same이면 Input(rl,nl,KeyCenter)을 출력한다

Key가 Lower이면 Input(rl,nl,KeyLower)을 출력한다

Key가 Upper이면 Input(rl,nl,KeyUpper)을 출력한다

Row가 Lower이면 Input(rl,nl,RowLower)을 출력한다

Row가 Upper이면 Input(rl,nl,RowUpper)을 출력한다

이제 도 4를 참조하면, 본체 센서(210)의 실시예들에 이용되는 다른 레이아웃이 도시되어 있다. 도 3에 대해 설명한 레이아웃이 더할 나위없이 적절하지만, 많은 존(300-390)이 설치되는 경우에, 접촉이 2개의 존 경계에 가까운 경우에, 사용자의 접촉 위치를 결정하기가 곤란할 수 있다. 이러한 어려움을 해결하기 위해 본체 센서(210) 상의 레이아웃(400)은 래칭 메커니즘(latching mechanism)을 구현하기 위해 키 사이 존(400-470)을 포함할 수 있다. 키 사이 존(400-470)은 2개의 키 존(300, 310-390)이 접촉하지 않도록 키 존(300, 310-390) 사이에 개재될 수 있다. 또한, 이들 키 사이 존(400-470)은 고유 식별자를 가질 수 있고, 본체 센서(210)는 키 사이 존(410-470)에서 접촉이 일어나는 경우를 감지하고 대응하는 메시지를 키스트로크 로직(220)에 보낸다. 키 사이 존(400-470)이 설치되는 경우, 본체 센서(210)로부터의 메시지는 다음의 알고리즘에 따라 처리될 수 있다.

행 번호 x는 이제 다음의 라벨 ($k_{x,1}$,$i_{x,1}$,$k_{x,2}$,$i_{x,2}$,…,$i_{x,y-1}$,$k_{x,y}$)을 갖는 존으로 구성되며, 여기서 y는 행 번호 x의 키의 번호이다. StartContact 위치에 키가 없거나 그 위치가 키 사이 존이라면, 키스트로크 모듈은 임의의 다른 활성없이 다음의 StartContact 메시지를 기다린다.

각 StartContact($k_{r1,n1}$)에 대해서:

Row를 Same으로, Key를 Same으로 설정한다

각 EnterZone($i_{r2,n2}$)에 대하여

r2=rl이면

n2<nl-1이면 Row를 Same으로, Key를 Lower로 설정한다

nl>nl이면 Row를 Same으로, Key를 Upper로 설정한다

r2<r이면 Row를 Lower으로 설정한다

r2>r이면 Row를 Upper로 설정한다

각 EnterZone($k_{r2,n2}$)에 대하여

r2=rl이면

n2=nl이면 Row를 Same으로, Key를 Same으로 설정한다

n2<nl이면 Row를 Same으로, Key를 Lower로 설정한다

n2>nl이면 Row를 Same으로, Key를 Upper로 설정한다

r2<rl이면 Row를 Lower로 설정한다

r2>rl이면 Row를 Upper로 설정한다

첫번째 StopContact에 대하여

Row가 Same이면

Key가 Same이면 Input(rl,nl,KeyCenter)을 출력한다

Key가 Lower이면 Input(rl,nl,LeyLower)을 출력한다

Key가 Upper이면 Input(rl,nl,KeyUpper)을 출력한다

Row가 Lower이면 Input(rl,nl,RowLower)을 출력한다

Row가 Upper이면 Input(rl,nl,RowUpper)을 출력한다

당업자에게는 키 사이 존이 임의의 2개의 존 또는 존 세트 사이에서 래칭 메커니즘을 구현하기 위해 채용될 수 있음이 자명할 것이다. 예컨대, 키 사이 존은 행(312, 314) 사이에서 래칭 메커니즘을 구현하기 위해 채용될 수 있다.

도 5에는 본 발명의 실시예들에 사용되는 사용자 인터페이스(500)의 상세 레이아웃이 도시되고 있다. 사용자 인터페이스(500)는 본체 센서(210)와, 사용자에게 정보를 표시하기 위한 텍스트 윈도우(510)를 포함할 수 있다. 본체 센서(210)는 존(520-542)의 세트를 포함할 수 있다. 각각의 존 세트는 기호 세트를 사용자에게 제공할 수 있다. 하나의 존(520-528) 세트는 영어의 문자에 대응하며, 그외 존(530-542)은 콘텍스트 또는 사용자 인터페이스(500)가 이용되고 있는 시스템의 아키텍처에 종속하는 그외 기능에 대응할 수 있다. 이러한 그외 기능은 커맨드 기능 또는 메뉴 기능을 포함할 수 있다.

도 6은 영어와 관련된 존(520-528)의 세트를 보다 구체적으로 도시하고 있다. 존(520-528)이 곡선으로 레이아웃될 수 있어, 존(520-528)은 사용자 인터페이스(500)를 이용하고 있는 장치를 잡고 있는 손의 엄지로 작동될 수 있다. 도시하는 실시예에서, 존(520-528)은 오른손잡이가 보다 효율적으로 작동할 수 있도록 배치되어 있다. 그러나, 전술한 바와 같이, 키의 가로 방향은 왼손잡이가 효율적으로 작동할 수 있는 포맷을 비롯한 임의의 원하는 포맷으로 변경될 수 있다. 각각의 존(520-528)은 상측 쌍(610), 중심 쌍(620) 및 하측 쌍(630)으로 그룹지어지는 최대 6개의 문자쌍을 포함한다. 사용자는 존(520-528)을 활성화할 수 있고, 본체 센서(210)는 이 활성을 검출하여 이 활성에 대응하는 메시지를 키스트로크 로직(220)에 전달하고, 이 로직에서는 이들 메시지를 입력 메시지로 변환할 것이다. 이어서, 의미화 로직(240)은 이 입력 메시지를 적절한 문자쌍과 연계시킬 것이다.

예컨대, 중앙 쌍(620)을 선택하기 위하여, 사용자는 원하는 쌍을 갖는 존을 터치하여 해제할 수 있다. 임의의 하측 쌍(630)을 선택하기 위하여, 사용자는 원하는 쌍을 갖는 존을 터치하여 원하는 쌍의 방향에 있는 임의의 존으로 슬라이딩할 수 있으며, 그 반대의 동작은 상측 쌍(610)을 선택하는 데에 적용된다. 이 예를 추가 설명하기 위해서 원하는 문자 쌍을 (s,h)라고 한다. 이 선택은 (g,m)를 포함하는 존(526)을 터치한 다음 쌍 (b,j)까지 슬라이딩하여 움직임으로써 달성될 수 있다. 중요한 것은 이 슬라이딩은 원하는 쌍 (s,h)을 포함하는 존(526) 아래에 있는 임의의 존(520, 524)에서 멈출 수 있다는 점이다. 그래도 의미화 로직(240)은 이 움직임을 원하는 쌍 (s,h)으로 변환할 수 있다.

의미화 로직(240)은 키스트로크 모듈(220) 또는 추가 센서(212)로부터 메시지를 수신한 다음, 이 메시지를 메시지가 수신되는 콘텍스트에 따라 기호의 세트로 또는 커맨드의 세트로 변환한다. 일 특정 실시예에서, 의미화 로직(240)은 키스트로크 로직(220)으로부터 입력 메시지를 수신한 다음, 이것을 콘텍스트에 따라 가능한 기호의 세트로 변환한다. 예컨대, 도 6에 대하여 전술한 사용자 동작은 문자 쌍 (s,h)으로 변환된다. 그리고, 이 변환 결과는 코어 로직(250)으로 전달될 수 있다. 또한, 의미화 로직(240)은 시스템의 콘텍스트에 따라 시스템의 입력 및 출력을 조정하는 책임이 있을 수 있다.

보다 구체적인 실시예에 있어서, 의미화 모듈(240)은 입력 메시지를 기호 또는 커맨드로 변환하는 데에 책임이 있을 수 있다. 이 변환은 콘텍스트 종속적일 수 있는데, 즉 의미화 모듈이 필요로 하는 콘텍스트의 부분을 SemantifyContext라고 할 수 있다. 의미화 모듈(240)은 콘텍스트 (SemantifyContext)에 따른 기호를 나타내는 입력 메시지에 대해, 기호 세트를 포함하는 SymbolMessage로 응답할 수 있다.

입력 메시지가 커맨드를 나타내는 경우에, 커맨드의 고유 식별자를 포함하는 메시지 (CommandMessage)가 출력된다. 이 고유 식별자 정보는 BackSpaceCmd, EndLexiconCmd, ItemSelectionCmd, NextCharCmd 및 PrevCharCmd를 비롯한 원하는 커맨드에 대응하는 값을 가질 수 있다. 입력 메시지가 콘텍스트 (SemantifyContext)에 대하여 어떤 것도 나타내지 않는다면, 어떤 동작도 이루어질 수 없다.

기호 세트를 표시하기 위하여 SymbolMessage를 CommanddMessage에 앞서 보낼 수 있는 경우에, 입력 메시지는 커맨드와 기호 세트 양쪽을 모두 나타낼 수 있다. 또한, 의미화 모듈(240)은 센서(210, 212)의 구성에 책임이 있을 수 있다.

일부 실시예에서, 텍스트 요소(280), 메뉴 요소(282) 또는 사용자 콘텍스트(290)로부터의 메시지는 다음과 같이 취급된다.

SelectItem은 커맨드 식별자로서 ItemSelectionCmd를 그리고 인수로서 SelectItem를 갖는 CommandMessage의 출력을 트리거한다.

최종 EndEdit 메시지가 최종 StartEdit보다 최근의 것이라면, StartEdit의 수신은 EditText 메시지를 트리거한다. 최종 StartEdit 메시지가 최종 EndEdit보다 최근의 것이라면, EndEdit의 수신은 인수 EndLexiconCmd를 갖는 CommandMessage의 코어 로직(250)으로의 출력을 트리거한다.

최종 EndEdit 메시지가 최종 StartEdit보다 최근의 것인 경우 의미화 모듈(240)이 막 SymbolMessage를 발송하려고 한다면, 의미화 모듈(240)은 SymbolMessage의 발송을 일시 정지하고, 메시지 TextInputQuery를 사용자 콘텍스트(290)에 발송한다. 수신된 다음 메시지가 QueryFlag와 관련되는 EditText라면, 의미화 모듈(240)은 EditText 메시지를 코어 로직(250)에 포워드하고 일시 정지된 SymbolMessage를 발송한다.

SemantifyContext는 (아키텍쳐 구성요소 또는 외부 중 어느 한 곳에서부터) 최종 수신된 메시지와 인수로서 이전의 SemantifyContext를 갖는 호출된 ContextGeneration 함수의 증분 결과일 수 있다. 임의의 2개의 메시지 사이에서 SemantifyContext가 일치하게 된다.

이제 의미화 모듈(240)에 의해 채용된 코어 로직(250)에 대해 설명하면, 도 7은 코어 로직(250)이 채용한 기본 방법론을 나타내고 있다. 코어 로직(250)은 입력 세트를 취득하고(블록 710), 처리 원칙을 적용한 다음(블록 720), 필요하다면 출력을 반환한다(블록 730). 코어 로직(250)은 입력 세트를 취득할 수 있고(블록 710), 이들 입력은 코어 로직(250)에 의해 내부에서 생성될 수 있거나, 본체 센서(210)를 통한 사용자로부터의 것일 수 있다. 또한, 입력은 의미화 모듈(240)로부터 코어 로직(250)에 의해 수신될 수 있다(블록 710). 전술한 바와 같이, 일부 상황에서는 의미화 모듈(240)은 수신한 메시지와, 작업하고 있는 콘텍스트에 따라 메시지 세트를 생성할 수 있다. 의미화 모듈(240)은 이들 메시지를 코어 로직(250)에 전달할 수 있고, 코어 로직에서는 이들 메시지를 입력 세트로서 수신할 수 있다(블록 710). 일부 실시예에서는 이들 입력이 기호 또는 커맨드를 나타내는 메시지일 수 있다. 또한, 이들 메시지는 프리픽스 및 프리픽스에 관련된 기호 시퀀스를 포함할 수 있다.

코어 로직(250)은 이 입력 세트를 수신하고(블록 710), 이들을 특정 원칙 세트에 따라 처리한다(블록 720). 이 처리는 기호, 단어, 임의적인 문자 시퀀스 또는 임의의 조합일 수 있는 제안의 형식화를 포함할 수 있다. 이 제안은 입력의 중의성 해소, 예측 완성이나 예측 결과, 또는 이 2개의 임의의 조합과 연계될 수 있거나, 이들로부터 도출될 수 있다.

코어 로직(250)이 입력 세트를 처리하는데(블록 720) 이용하는 원칙은 언어 모델(260)과 용법 모델(280)을 이용하여 후보 또는 예측 결과의 세트를 생성하고, 이들 후보 또는 예측 결과를 평가하여 제안 세트를 생성하도록 설계될 수 있으며, 그 하나 이상의 제안이 사용자에게 제시될 수 있다. 일 실시예에서, 코어 로직(250)은 평점 시스템을 이용하여 입력을 처리한다(블록 720). 코어 로직이 입력을 수신하는 경우, 그 입력에 프리픽스가 없다면, 하나 이상의 제안이 생성되어 평점 시스템에 따라 평가된다. 그러나, 프리픽스가 있다면, 새로운 입력을 그 프리픽스에 첨부하여 시퀀스를 생성하고 이 시퀀스에 기초하여 하나 이상의 후보를 생성하여 평가하려고 시도한다. 생성되는 후보가 없거나, 생성된 후보가 임의의 임계 평점을 넘지 않으면, 입력을 지정하기 위한 방법이 제공될 수 있고, 후보가 있다면, 후보를 평가하여 예측 결과를 생성하며, 이 평가에 기초하여 제안이 형성되어 사용자에게 제시될 수 있다. 또한, 제안은 원 입력의 중의성 해소라는 점에서 형식화될 수 있다.

소정의 실시예에서는, 코어 로직(250)이 입력에 기초하여 후보를 형식화하고, 하나 이상의 후보로부터 예측 결과를 결정한 다음, 생성된 후보 및 예측 결과의 세트로부터 사용자에게 제시하기 위하여 제안을 형식화할 수 있다.

일부 실시예에서는, 코어 로직(250)이 이용하는 원칙 및 알고리즘은 3개의 카테고리, 즉 추측 원칙, 중의성 해소 원칙, 및 내부 원칙으로 그룹화될 수 있다. 중의성 해소 및 추측 원칙은 코어 로직(250)을 위한 메인 드라이버일 수 있으며, 내부 원

칙은 코어 로직(250) 배후에서 제어 기능성과 같은 역할을 할 수 있고, 추측 및 중의성 해소 원칙 세트, 이들 2개의 세트의 평가 또는 이들 원칙 세트로부터의 응답을 이용하는 데에 책임이 있는 기능성을 제공할 수 있다. 또한, 내부 원칙은 사용자에게 제시되는 프리픽스 또는 제안을 작성하는 책임이 있을 수 있다.

이들 내부 원칙은 원하는 기능성과, 원칙이 이용되고 있는 콘텍스트에 따라 중의성 해소 원칙 및 추측 원칙과 함께 작용할 수 있다. 이들 원칙은 시스템의 작동을 다양한 콘텍스트로 기술할 수 있고, 사용자의 희망 또는 내부 상태에 기초하여 채용될 수 있다. 중의성 해소 원칙은 사용자로부터, 즉 의미화 모듈(240)로부터 수신된 입력이 애매한 경우에 또는 적절한 후보가 없는 경우에 작동을 기술할 수 있다. 소정의 실시예에서, 이들 중의성 해소 원칙은 언어 모델(260)만 채용한다. 추측 원칙은 관련 후보를 발견한 경우의 작동을 기술할 수 있고, 다른 실시예에서 언어 모델(260)과 용법 모델(280) 모두를 채용할 수 있다.

일 특정 실시예에서, 코어 로직 모듈(250)은 의미화 모듈(240)로부터 CommandMessage 또는 SymbolMessage를 수신한다. 코어 로직(250)은 원칙 세트에 따라 그 메시지에 응답한다. 코어 로직(250)의 원칙이 사용자와의 통신을 지시하는 경우에, 의미화 모듈(250)은 사용자와의 인터페이스에 이용된 대응 메시지를 전달하는데 이용된다.

이 실시예에서, 코어 로직(250)의 나머지 원칙은 3가지 방향, 1) 시스템이 입력에 관한 관련 제안을 갖는 경우의 코어 로직(250)의 작동을 기술하는 추측 원칙, 2) 입력이 애매하고, 시스템이 입력에 관하여 관련 제안을 추측할 수 없는 경우에 시스템의 작동 방법을 기술하는 중의성 해소 원칙, 및 3) 중의성 해소 원칙에서 추측을 평가하는데 필요한 몇가지 함수를 포함하는 내부 원칙 하에 편성될 수 있다.

이들 원칙에 대해서는 각각 추가로 후술하기로 한다.

모든 원칙의 공통 변수는 다음과 같다.

CurrentRun 기호 시퀀스

CurrentPropositionSet 데이터 구조에 저장될 수 있는 제안 세트로서, 다음과 같은 코드로 표현될 수 있다.

typedef struct {

Char* run;

Int count:} Proposition;

InputSeqs는 세트 기호의 세트 시퀀스이다.

EditionState는 2개의 값: GuessingState와 UnambiguousState를 가질 수 있다.

FinalSeq는 기호의 시퀀스 또는 Undefined이다.

CurrentSym는 정수이다.

코어 로직(250)은 EditText 메시지를 수신하는 경우, 다음과 같이 응답한다.

StartEdit 메시지를 텍스트 요소에 발송한다.

EditText 프리픽스가 empty라면

CurrentPropositionSet는 empty set로 설정된다;

InputSeqs는 empty set로 설정된다;

원칙 ShowPropositions이 평가된다;

그렇지 않다면,

NewPrefix가 인수 EditText 프리픽스에 의해 평가된다;

CurrentPropositionSet가 empty라면

EditionState는 UnambiguousState로 설정된다;

InitUnambiguous가 평가된다;

그렇지 않다면,

원칙 ShowPropositions가 평가된다;

이 알고리즘은 새로운 입력마다 코어 로직(250)의 상태를 초기화한다. 초기화는 프리픽스가 없는 경우에는 간단한다. 프리픽스가 제공되는 경우, 알고리즘은 언어 모델(260)을 이용하여, 가능한 프리픽스의 완성이 되는 최장 부분 스트링을 룩업하려고 시도하도록 설계된다. 언어 모델(260)이 그러한 부분 스트링을 포함하지 않는다면, 알고리즘은 명확한 입력 모드로 전환된다.

i. 추측 원칙(Guessing Rules)

추측 원칙은 EditionState=GuessingState인 경우에 적용된다. 이 장의 원칙은 고유한 상수 Add를 갖는다. 코어 로직(250)에 도착하는 메시지 타입마다 원칙이 있다.

SymbolMessage는 기호 리스트로서 S=(Si,…,St)를 가지며, t>1이다.

InputSeqs를 InputSeqs와 (S)의 연결로 설정한다

CurrentRun가 empty가 아니라면, newPrefix (CurrentRun)를 평가하고 CurrentRun를 empty로 설정한다

CurrentPropositionSet가 empty라면

EditionState는 UnambiguousState로 설정된다;

InitUnambiguous가 평가된다;

그렇지 않다면

CurrentPropositionSet가 empty라면:

L은 타입 노드 $\{L_1,\cdots,L_n\}$의 비일치 요소의 최대 세트이고, 여기서 각 i는 {1,…,n}의 부분이다:

L_i.c=S_j이고, j는 {1,…,t}의 부분이다

IsInSubtree(L_i.location,root)

P는 proposition $(P_i,\cdots,P_m)$의 최소 세트이며, 각 i는 {1,…,m}이다:

P_i.run=(L_i.c) 및 P_i.count=L_i.count

PF=FilterProposition(P)

PF가 empty라면,

EditionState를 UnambiguousState로 설정한다

InitUnambiguous를 평가한다

그렇지 않다면:

CurrentPropositionSet를 PF로 설정한다

ShowPropositions를 평가한다

인수로서 click을 갖는 UserNotify 메시지를 출력한다.

그렇지 않다면:

CurrentProposiontSet가 세트 $\{CP_1,\cdots,CP_m\}$이다.

TempLocationSet는 비일치 노드 요소 $\{(L_1,X_1,Y_1,Count_1),\cdots,(L_n,X_n,Y_n,Count_n)\}$, 여기서 각 i는 {1,…,n}의 부분이다:

$L_i.c=S_j$, 여기서 j는 {1,…,t}의 부분이다

Y_i는 CP_j, 여기서 j는 {1,m}이다

X_i는 노드 $(N_1,\cdots,N_m)$의 최장 시퀀스, 여기서:

$(N_1.c,\cdots,N_m.c)$는 Y_i.run의 서픽스 낫 널(suffix not null)을 형성한다

IsInSubtree(N_1,root)

IsInSubtree(L_i,N_m)

$Count_i$=ExtendProposition(L_i,CP_j.count)

PF는 proposition$(PFP_1,\cdots,PFP_n)$의 시퀀스, 여기서 각 i는 {1,…,n}이다

PFP_i.run는 Y_i.run와 L_i.c을 연결한 것이다

PF2=FilterProposition(PF)

PF2가 empty라면:

EditionState를 UnambiguousState로 설정한다

InitUnambiguous를 평가한다

그렇지 않으면

CurrentPropositionSet는 세트 PF2, 여기서 각 i는 {1,…,n}이다:

ShowPropositions를 평가한다

인수로서 클릭을 갖는 UserNotify 메시지를 출력한다

주의 : SymbolMessage를 위한 이들 원칙은 사용자 입력을 처리할 수 있고, 사용자 입력이 애매한지의 여부에 관계없이 최종 EditText 메시지 이후에 수신한 모든 정보에 기초하여 사용자가 입력하고자 하는 기호 시퀀스를 발견하려고 시도할 수 있다.

CommandMessage

메시지에 포함된 커맨드 식별자에 따라, 다음의 동작이 취해진다:

BackSpaceCmd

InputSeqs이 empty 시퀀스라면 EditStop 메시지가 UseContext로 발송된다

그렇지 않다면,

Replay는 입력 시퀀스 (S_i,…,S_n)에 따른 시퀀스 (EditText(),SymbolMessage(S_i),…,SymbolMessage(S_{n2}))이다. CurrentPropositionSet 및 InputSeqs는 empty 세트로 설정된다. 코어 로직은 인수가 SymbolMessage인 Replay의 요소를 차례로 그 자신에게만 발송하고 그 자신이 다른 모듈에 메시지를 출력하는 것을 방지한다. 결국 SymbolMessage(S_{n-1})가 발송된다.

주의 : 이들 원칙은 코어 로직(250)을 최종 미폐기된 SymbolMessage가 발생하기 전의 상태로 복귀하게 할 수 있다.

EndLexiconCmd

CurrentRun가 Add와 같으면,

EditionState를 UnambiguousState로 설정한다

InitUnambiguous를 평가한다

그렇지 않으면

CurrentRun가 empty이고 CurrentPropositionSet가 폼 (P,…)에 속하면 CurrentRun를 P.run로 설정한다

인수 CurrentRun를 갖는 SubstituteText를 발송한다

EditEnd 메시지는 사용자 콘텍스트에 그리고 Semantization 모듈에 발송된다

인수로서 CurrentRun를 갖는 StoreWord 원칙을 평가한다

주의 : 사용자가 메뉴에서 제안 (Add)을 선택하였다면, 이 원칙은 코어 로직(250) 원칙을 중의성 해소 모듈로 전환하며, 그렇지 않다면 이 원칙은 환경을 정화하고 사용자에 의한 최종 StartEdit 이후에 구성된 기호 시퀀스를 고려한다.

ItemSelectionCmd

메뉴에서 강조 처리된 아이템이 커맨드 Add이면:

CurrentRun을 Add로 설정한다

인수 P.c를 갖는 SubstituteText를 발송하고, CurrentPropositionSet는 (P…)이다

그렇지 않으면

CurrentRun를 PropositionSelect의 인수로 설정한다

CurrentRun를 갖는 SubstituteText 메시지를 텍스트 필드로 발송한다

ShowRules를 평가한다

주의 : 사용자는 TextElement에 표시하기를 원하는 입력으로서 제안 중 하나를 선택할 수 있다.

중의성 해소 원칙(Unambiguous Rules)

EditionState=UnambiguousState인 경우에 추측 원칙 하에서 인덱싱된 원칙이 적용된다.

InitUnambiguous

는 UserNotification(Error)를 출력한다

CurrentSym를 1로 설정한다

InputSeqs는 폼 (S_1,…,S_n)를 갖고, FinalSeq를 n개 요소의 시퀀스로 설정한다

Undefined

ShowUnambiguous를 평가한다

StartAmbiguousInput를 의미화 모듈로 발송한다

SymbolMessage

는 시퀀스 S를 인수로서 갖는다

InputSeqs는 폼 (S_i,…,S_n)을 갖고, InputSeqs를 시퀀스 (S_i,…,S_n,S)로 설정한다

FinalSeq는 폼 (F_1,…,F_n)을 갖고, FinalSeq를 시퀀스 (Fl,...,Fn,Undefined)로 설정한다

ShowUnambiguous를 평가한다

CommandMessage

BackSpaceCmd

InputSeqs이 폼 (S)라면:

인수로서 empty 시퀀스를 갖는 SubstituteText를 발송한다.

EditEnd 메시지를 사용자 콘텍스트에 그리고 의미화 모듈에 발송한다.

그렇지 않으면:

InputSeqs은 폼 (S_1,…,S_n)이고 InputSeqs를 (S_1,…,S_{n-1})로 설정한다

FinalSeq은 폼 (F_1,…,F_n)이고 FinalSeq를 (F_l,…,F_n)로 설정한다

EndLexiconCmd

는 CurrentRun를 갖는 SubstituteText를 텍스트 필드로 발송한다

EditEnd 메시지는 사용자 콘텍스트에 그리고 의미화 모듈에 발송된다.

인수로서 CurrentRun를 갖는 StoreWord 원칙을 평가한다

ItemSelectionCmd

는 인수 A를 갖는다

A가 Done이면,

CurrentRun를 갖는 SubstituteText 메시지를 텍스트 필드에 발송한다

EditEnd 메시지는 사용자 콘텍스트에 그리고 의미화 모듈에 발송된다

인수로서 CurrentRun를 갖는 StoreWord 원칙을 평가한다

그렇지 않으면:

폼 (F_l,…,$F_{Currentsym-1}$,$F_{Currentsym}$,$F_{CurrentSym+1}$,…,F_n)이 (F_1,…,$F_{CurrentSym-1}$,A,$F_{CurrentSym+1}$,…,F_n)로 설정된다.

CurrentSym를 CurrentSym+ 1로 설정한다

ShowUnambiguous를 평가한다

PrevCharCmd

CurrentSym=1이면 UserNotify (Error)를 출력, 그렇지 않다면 CurrentSym=CurrenSym-1

NextCharCmd

CurrentSym note가 InputSeq의 길이와 같으면, UserNotify (Error)를 출력, 그렇지 않으면,
CurrentSym=CurrenSym+ 1

iii 내부 원칙(Internal Rules)

이 장에 기술하는 원칙은 "중의성 해소 원칙" 및 "추측 원칙"을 평가할 때에 명백하게 호출된다.

ShowUnambiguous

InputSeq는 폼 (($S_{1,1}$,…,$S_{1,M(1)}$),…,($S_{n,1}$,…,$S_{n,M(n)}$))이고, 여기서 M(x)는 InputSeq의 x번째 요소의 길이를 넘겨주는 함수이다. FinalSeq는 폼 (F_l,…,F_n)이고, 여기서 각 Fi는 Undefined이거나, ($S_{1,1}$,…,$S_{1,M(1)}$)의 요소 중 하나이다. C는 시퀀스 (C_l,…,C_n) 중 하나이고, 여기서 i는 {1,…,n}이다:

$F_i \neq$ Undefined이면 $C_i=F_i$ 그렇지 않으면 Ci는 $(S_{i,1},\cdots,S_{i,M(1)})$,

$Quotation_n$가 최대:

Ci=L.c이면, 여기서 IsInSubtree(L,root) $Quotation_l$=L.count 그렇지 않으면 $Quotation_l$=0

I는 {2,…,n}, $Quotation_n$=Accumulate($Quotation_{i-1}$,L.count) 여기서 L=NodeFromPath $(C_{i-j},\cdots,C_i)$, J는 최대

CurrentRun를 c로 설정한다

인수 $(S_{CurrentSym,1},\cdots,S_{CurrenSym,M(l),}$,Done)를 갖는 MenuSet 메시지를 발송한다

Accumulate 및 Invariant는 임플리멘테이션(implementation) 종속 방법으로 정의될 수 있으며, 본 명세서를 이해한 후에 당업자에게 명백할 것이다.

NewPrefix

는 인수로서 s = $(C_1\cdots C_r)$를 갖는다

InputSeqs를 시퀀스 $(\{C_1\},\cdots,\{C_r\})$로 설정

P는 다음과 같은 제안이다:

p.run=s, p.count=Invariant

최장 $(N_1,\cdots,N_m)$ 시퀀스

여기서:

IsInSubtree (N_i,N_{i+1}), i는 [l,m-1]이다

N_m=root

각 N_i에서 i는 [l,m-1], $N_i.c=C_{r-m+i}$

P가 존재하면

CurrenPropositionSet를 FilterProposition ({P})로 설정한다

그렇지 않으면

CurrentPropositionSet를 empty으로 설정한다

ShowPropositions 원칙

CurrentPropositionSet는 폼 $\{P_1,\cdots,P_n\}$이다.

UP는 세트이고, 여기서 i는 {1,…,n}이다:

시퀀스 PredA ($(PrA_l,CA_l,SymA_l),\cdots,(PrA_m,CA_m,SymA_m)$)는 GetSugestions ($P_i$.run)과 같다

시퀀스 PredB는 최단 시퀀스 ($(PrB_1,CB_1,SymB_1),\cdots,(PrB_k,CB_k,SymB_k)$), 각 j는 {1,…,k-1}, $CB_j \geq CB_{j+1}$

각 j는 {1,…,m}, Relevant(PrA_j) Adjust($PrA_j,CA_j,SymA_j$)가 PredB의 부분이면

U는 GetUNodeFromPath (Pi.run)이다

U가 UnknownPath이면

((P_i.run,P_i.c))는 UP의 부분이다

그렇지 않으면

Adjust((P_i.run,P_i.c,U),($PrB_l,CB_1,SymB_i$),…,($PrB_k,CB_k,SymB_l$))가 UP의 부분이다

메뉴가 트리를 보여줄 수 있다면 인수 FilterSortExtentions(U)를 갖는 SetMenu 메시지를 발송한다, 그렇지 않으면:

U는 폼 (((DS,DC)((PS,PC)((SS,SC)…)…)…)

U2는 DC,PC,SC를 따라 정렬된 U로부터 모든 요소 (DS,DC),(PS,PC) 및 (SS,SC)로 이루어진 최단 리스트이다

인수 U2를 갖는 SetMenu message를 발송한다

주의 : 이들 원칙은 사용자 입력에 대한 중의성 해소/예측을 위해 현재 평가되는 모든 관련 제안을 사용자에게 보여줄 수 있다.

Adjust는 임플리멘테이션 종속적이며 예측 결과의 가능성을 중의성 해소 제안의 가능성과 관련시킬 수 있다. 이것은 사용자 입력이 대체로 최소이게 한다.

ExtendProposition 및 FilterProposition는 임플리멘테이션 종속적일 수 있는 함수이다. FilterProposition은 사용자에게 보여지는 제안을 결정하는 임계 평점 레벨을 조절할 수 있다. 임계치가 높으면 사용자에게 보여지는 제안에서의 비상관수를 낮다.

StoreWord도 역시 임플리멘테이션 종속적이며, 문자 세트에 대해 유지하는 위치일 수 있으며, 언어 모델(LanguageModel) 및 용법 모델(UsageModel) 구조에 할당된 리소스양을 관리할 수 있다. 또한, 사용자의 행동에 맞게 모델 적응을 제어하여 차후 이용을 위해 예측/중의성 해소를 개선할 수 있다.

이제 도 8을 참조하면, 코어 로직(250)이 언어 모델(260)을 이용할 수 있는 방법의 일례가 도시되어 있다. 언어 모델(260)은 J.G. Cleary 및 J.H. Witten의 논문 "Data Compression Using Adaptive Coding and Partial String Matching"[IEEE Communications journal(Clearyl)에서 1984년 4월 4일자로 발행된 Volume Com-32]에 개시된 모델의 확장형 및 개선형일 수 있으며, 이 논문은 여기에서의 참조에 의해 본 명세서에 그 전체가 포함된다. 코어 로직(250)은 언어 모델(260)을 채용하여 프리픽스 또는 입력을 수신하고(블록 810), 후보 세트를 형식화하며(블록 820), 그 후보 세트를 정렬할 수 있다(블록 830). 후보 로직(250)은 프리픽스 및 입력을 수신할 수 있다. 프리픽스는 빈 세트를 포함한 임의의 긴 기호 세트일 수 있으며, 입력은 의미화 모듈(240)에 대하여 기술된 타입의 입력 메시지일 수 있다. 그리고 후보 세트는 프리픽스와 입력에 기초하여 계산된다(블록 820). 프리픽스가 비어 있다면, 입력에만 기초하여 후보가 계산될 수 있다. 그러나, 프리픽스가 비어 있지 않다면, 코어 로직(250)은 프리픽스와 관련된 스트링 세트를 룩업하고 그 프리픽스에 기초하여 후보를 생성한다(블록 820).

일부 실시예에서는 코어 로직(250)이 언어 모델(260)을 이용하여 이들 후보를 형식화할 수 있으며(블록 820), 언어 모델은 긴 부분 스트링 및 관련 평점을 임의대로 저장하기 위해 트리 구조를 채용할 수 있다. 일 실시예에서는, 코어 로직(250)

은 프리픽스와 관련된 언어 모델(260)에서 최장 스트링을 찾는다. 관련 실시예에서, 코어 로직(250)은 언어 모델(260)을 이용하여 후보 리스트를 작성할 수 있는데, 이 리스트는 임플리멘테이션에 따라 단순 리스트 또는 내포 리스트일 수 있다. 코어 로직은 언어 모델(260)에서 구현된 트리의 각 노드와 관련된 평점에 기초하여 후보와 연계시키는 평점을 결정할 수 있다. 이들 평점은 차례로 개연적 모델에 기초할 수 있다.

당업자라면 각종 알고리즘이 평점을 계산하여 후보의 평점을 매기는 데에 이용될 수 있음을 이해할 수 있을 것이다. 예컨대, 사용자가 단어 "test"를 입력하기를 원한다고 하자. 도 6에서 설명한 레이아웃을 이용하여, 사용자는 쌍 (t,z)을 선택할 수 있고, 코어 로직(250)은 "t"의 가능성과 "z"의 가능성을 계산할 수 있다. 이어서, 사용자는 쌍 (e,f)을 선택할 것이다. 코어 로직(250)은 언어 모델을 이용하여 이전 후보 (t,z) 세트과 관련된 각 문자에 이어 문자 "e"의 평점과 "f"의 평점을 계산하고, 그것과 최초 문자 쌍 (t,z)에서의 각 문자의 가능성을 곱한다. 이런 식으로 2개 문자의 각 시퀀스의 가능성을 계산할 수 있다.

일 특정 실시예에서, 이런 타입의 언어 모델(260)을 저장하기 위한 구조 및 입력을 처리하고 가능성을 계산하기 위한 관련 로직은 다음과 같이 구현될 수 있다.

언어 모델(260)은 다음과 같이 나타내는 타입 LanguageModel의 구조에 포함될 수 있다.

```
Typdef struct {

Char c;

Int counter;

NodeStruct* next;

NodeStruct* subtree;} NodeStruct

Typedef NodeStruct* Node;

Node root;
```

노드 구조는 실시예에서 [Clearyl]의 구조, 즉 모듈로 신택스(modulo the syntax)일 수 있다. Root는 언어 모델(260)에 대한 코어 로직의 입력점일 수 있다.

다음의 특성 및 알고리즘은 언어 모델(260)을 지원하는 구조에 대하여 정의될 수 있다.

IsInSubtree(Node N_2,Node N_1)

N_1.subtree가 무효 기준이면

IsInSubtree(N_2,N_1)는 거짓이다

그렇지 않으면

N_2=N_1.subtree

IsInSubtree(N_2,N_1)는 참이다

그렇지 않으면

IsInSubtree(N_2,N_1)=IsNeighbor(N_2,N_1.subtree)

주의 : 이 특성은 N_2가 N_1의 자식일 때 참이고, 다른 경우에는 거짓이다.

IsNeighbor(Node N_2,Node N_1)

N1.next가 노드로의 무효 기준이면 IsNeighbor(N_2,N_1)는 거짓이다

그렇지 않으면

N_2=N_1.next

IsNeighbor(N_2,N_1)는 참이다

그렇지 않으면

IsNeighbor(N_2,N_1)=IsNeighbor(N_2,N_1.next)

주의 : 이 특성은 N_2가 트리에서 N_1와 동등 레벨인 경우에 참이고 다른 경우에는 거짓이다.

NodeFromPath((S_1,…,S_n))

각 i는 {1,…,n}, S_i는 기호이다

N은 노드 (n_1,…,N_n)의 시퀀스, 여기서 i는 {1,…,n}, N_i.c=S_i

i는 {l,…,n-1}, IsInSubtree(N_{i+1},N_i)는 참이다

IsInSubtree(N_1,root)

N이 NodeFromPath((S_1,…,S_n))에 존재하면 값을 N_n로 설정, 그렇지 않으면, 값을 UnknownPath로 설정한다

도 9는 코어 로직(250)이 용법 모델(280)을 이용할 수 있는 방법의 일 실시예를 도시하고 있다. 코어 로직(250)은 용법 모델(280)을 이용하여 스트링 세트를 수신하고(블록 910), 예측 결과 세트를 형식화하며(블록 920), 예측 결과의 세트를 정렬할 수 있다(블록 930). 코어 로직(250)은 스트링 세트를 구성할 수 있는 기호의 세트를 수신할 수 있으며(블록 910), 이들 스트링은 사용자로부터의 입력과 프리픽스일 수 있다. 일부 실시예에서는 이들 기호가 언어 모델(260)을 이용하여 코어 로직에 의해 어셈블된, 즉 코어 로직에 반환된 후보 세트로부터 코어 로직(250)에 의해 도출 또는 추출될 수 있다.

그리고, 이 스트링 세트는 스트링 세트로부터의 스트링에 기초하여 예측 결과 세트를 형식화하는데(블록 820) 이용될 수 있다. 용법 모델(280)은 가능한 완성의 통계적 정렬에 기초한 스트링에 대해 일련의 가능한 완성을 결정할 수 있다.

일 실시예에서, 용법 모델(280)은 트리 구조를 이용하여 단어를 표현할 수 있으며, 트리의 각 노드가 문자를 포함하게 된다. 단어의 종료를 지정할 수 있는 각 노드가 구별될 수 있고, 평점은 그러한 단말 노드와 연계될 수 있다. 당업자라면 각종 평점 시스템을 이용하여 평점을 지정할 수 있음을 이해할 것이다.

상기 예를 계속하여, 스트링 "te"가 용법 모델(280)에 수신된다고 하자. 도 10은 용법 모델(280)이 스트링을 완성하기 위해 이용할 수 있는 트리의 일 표현을 도시하고 있다. 스트링 "te"는 "te"로 시작하는 단어의 트리 기반의 표현예(1000)에 인덱스로서 이용될 수 있으며, 이 트리(1000)는 이들 노드가 스트링 "te"에 대해 가능한 완성을 제공하는 인디시아(indicia)를 포함할 수 있는 단말 노드(1010, 1020, 1030, 1040)를 가질 수 있다. 이 인디시아는 평점 관련 단말 노드(1010, 1020, 1030, 1040) 또는 일부 실시예에서는 부울 값일 수 있다. 이 평점은 동일한 스트링을 완성할 수 있는 동일한 깊이의 노드 수에 기초할 수 있다. 또한, 평점은 같은 가지에 있는 얼마나 많은 노드가 스트링에 대한 완성이 아닌가에 기초할 수 있다.

노드와 관련된 평점에 기초하여, 노드는 정렬될 수 있고(블록 830), 트리(1000)를 통해 최고(또는 최하) 순번을 가진 노드로의 경로에서 각각의 노드와 연계된 문자의 시퀀스는 반환될 수 있다. 도 10을 참조하면, 노드(1020)가 최고의 평점을 갖는다면, 문자 시퀀스 "tested"는 반환될 수 있다. 다른 실시예에서, 하위 트리를 통과하는 경로에서의 단말 노드와 연계된 각 문자 시퀀스는 연계된 단말 노드의 평점과 함께 반환될 수 있다.

마찬가지로, 예측 결과의 동의어도 그 연계 평점과 함께 반환될 수 있다. 예컨대, 도 10에서 노드(1020)가 최고 평점이라면, 문자 시퀀스 "tested"가 반환될 수 있다. 또한, 하나 이상의 동의어가 반환될 수 있다. 이들 동의어는 연계 평점에 기초하여 결정될 수 있고, 이들 연계 평점과 함께 또는 없이 반환될 수 있다. 특정 예측 결과와 연계된 동의어는 예측 결과에 대응하는 단말 노드에 저장될 수 있다. 이 예를 계속해서 설명하면, 예측 결과 "tested"에 대응하는 단말 노드(1020)는 "tester", "testes", "tesla" 등을 비롯한 "tested"의 각종 동의어를 저장할 수 있다. 이들 "tested"의 동의어는 예측 결과 "tested"에 대응하는 기호의 시퀀스와 함께 반환될 수 있다. 일 특정 실시예에서, 이러한 타입의 용법 모델(280)을 저장하기 위한 구조와, 입력을 처리하고 평점 또는 가능성을 계산하기 위한 연계 로직이 다음과 같이 구현될 수 있다.

```
Typedef enum {true, false} Boolean;

Typedef struct{

Char* s;

Int count;

} SymListNode

Typedef struct {

Char c;

Unode[] sons;

Boolean terminal;

Int heads;

Int count;

SymListNode[] sym;

} Unode

uNode* Uroot;
```

Uroot는 용법 모델(280)로의 코어 로직(250)을 위한 입력점일 수 있다. 용법 모델(280)에 대하여 다음의 특성이 정의될 수 있다.

IsSon(U_1:UPNode,U_2:UPNode)

IsSon(U_1,U_2)는 U_1가 U_2.sons의 부분이면 참, 그렇지 않으면 거짓이다

주의 : IsSon는 U_1가 U_2의 직계 자식인지를 나타낸다.

GetUNodeFromPath((C_1,…,C_N))

시퀀스 (UP_1,…,UP_N):

IsSon(UP_1,URoot)는 참

각 i는 {l,…,n-1}, IsSon(UP_{i+1},UP_i)는 참이다

각 i는 {1,…,n}, UP_i.c=C_i

UP_n가 GetUNodeFromPath((C_1,…,C_n))에 존재하면, 값은 UP_n, 그렇지 않으면 값은 UnknownPath이다

주의 : GetUpFromPath는 인수가 존재한다면 인수로서 주어진 시퀀스와 연계된 용법 모델에 UNode를 반환하고, 그렇지 않으면 상수 UnknownPath가 반환된다.

GetSuggestions(C)

C는 기호 시퀀스이다

UP는 GetUPFromPath(C)의 결과이다

UPS는 UPNodes (UPS_1,…,UPS_n)의 최소 세트이며, 여기서 각 i는 {1,…,n}:

시퀀스 (T_1,…,T_q)가 존재하고, 여기서:

T_1는 UP이다

T_q는 $UPS_{i\ q>1}$이다

각 j는 {1,…,q-1}이다

IsSon(T_{j+1},T_j)

UPS_i.terminal는 참이다

S는 시퀀스 ((S_1,$Count_1$,Sym_1),…,(S_n,$Count_n$,Sym_n)이며, 여기서 각 i는 {1,…,n}이다: UPS_x=GetUPFromPath(S_i), UPS_x.count=Count 및 Sym_j=UPS_x.sym

GetSugestion(C) 값은 S이다

주의 : GetSuggestions는 단어와 그것의 주어진 프리픽스의 동의어를 검색한다. 각각의 단어와 동의어는 용법 모델(280)에서의 이용 빈도를 나타내는 평점과 연계된다.

도 7을 참조하면, 코어 로직은 전술한 입력의 처리(블록 720)에 기초하여 출력을 형식화할 수 있다(블록 730). 이 출력은 문자 시퀀스 또는 메뉴 옵션 세트를 갖는 문자 시퀀스일 수 있다. 이어서, 이 출력(블록 730)은 추가 삽입을 위해 사용자에게 제시될 수 있다. 일 실시예에서, 코어 로직(250)은 형식화된 출력(블록 730)을 사용자에게 제안으로서 제시하기 위해 의미화 모듈(240)과 상호 작용할 수 있다.

기술한 테이블, 필드, 또는 단계 모두가 반드시 필요하지 않고, 그 테이블, 필드 또는 단계가 필요하지 않을 수 있으며, 추가 테이블, 필드 또는 단계가 설명한 것 외에 부가될 수도 있다. 또한, 각각의 동작의 나열 순서는 그 동작이 수행되는 순서일 필요는 없다. 본 명세서를 이해한 후, 당업자라면 어떤 테이블, 필드 및 정렬이 임의의 특정 목적에 최적인지 결정할 수 있을 것이다.

전술한 설명에 있어서, 본 발명은 특정 실시예를 참조하여 기재되었다. 그러나, 당업자라면 이하의 특허청구범위에서 제시되는 본 발명의 범주에서 벗어나는 일없이 다양한 변형 및 변화가 가능할 수 있음을 이해할 것이다. 따라서, 명세서 및 도면은 제한적인 것이 아니라 예시적인 것으로 간주되어야 하며, 모든 그러한 변형은 본 발명의 범주 내에 포함되는 것으로 한다.

장점, 다른 이점 및 문제 해결 방법을 특정 실시예를 참조하여 전술하였다. 그러나, 그러한 장점, 다른 이점, 문제 해결 방법, 및 임의의 장점, 이점 또는 해법을 발생하게 하거나 보다 분명해 지게 할 수 있는 임의의 구성 요소는 특허청구범위의 일부 또는 전부의 결정적, 필요한 또는 본질적 특징 또는 구성요소로서 해석되어서는 안된다.

도면의 간단한 설명

본 명세서에 첨부되며 명세서의 부분을 구성하는 도면은 본 발명의 소정의 양상을 도시하기 위해 포함된다. 본 발명에 대한 더 분명한 이해, 그리고 본 발명이 제공하는 구성요소 및 시스템의 동작에 대한 보다 분명한 이해는 예시적인, 그러니깐 도면에 도시하는 비제한적인 실시예를 참조하여 보다 자명해 질 것이며, 도면에서 동일한 도면 부호는 동일한 구성요소를 나타낸다. 도면에 도시하는 특징부는 반드시 실측으로 도시하지 않았다.

도 1은 종래의 축소형 키보드를 나타내는 도면이다.

도 2는 개시하는 시스템 및 방법을 구현하기 위한 가능한 일례의 아키텍쳐를 나타내는 도면이다.

도 3은 본체 센서(principle sensor)로 존을 채용하기 위한 레이아웃의 일 실시예를 나타내는 도면이다.

도 4는 본체 센서로 존을 채용하기 위한 레이아웃의 일 실시예를 나타내는 다른 도면이다.

도 5는 장치 상에 데이터 입력을 위한 존을 채용하기 위한 레이아웃의 일실시예를 나타내는 도면이다.

도 6은 도 5에 도시하는 존의 일부를 상세하게 나타내는 도면이다.

도 7은 코어 로직으로 입력을 처리하기 위한 일 방법론을 나타내는 도면이다.

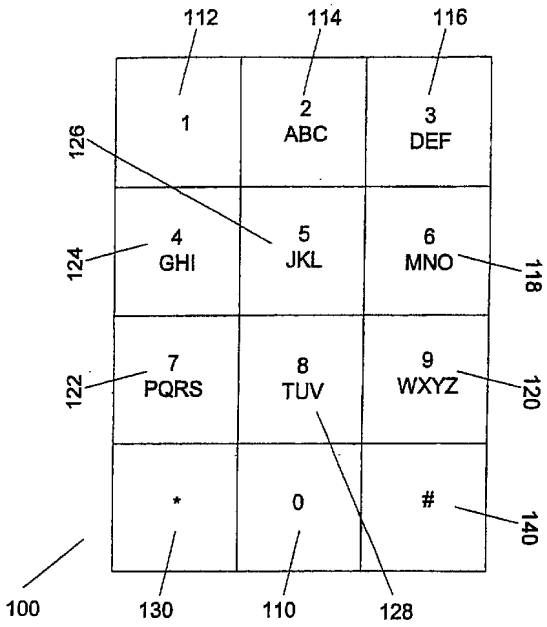
도 8은 입력 처리 시에 코어 로직이 이용할 수 있는 후보들을 형식화하기 위한 일 방법론을 나타내는 도면이다.

도 9는 코어 로직이 이용할 수 있는 예측 결과를 형식화하는 일 방법론을 나타내는 도면이다.

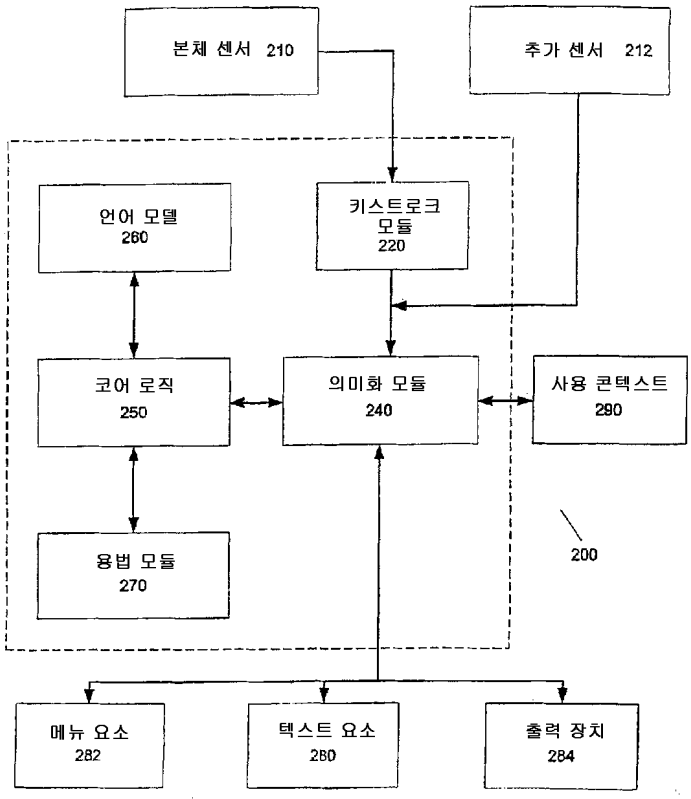
도 10은 언어 모델을 저장하기 위한 데이터 구조의 일 실시예를 나타내는 도면이다.

도면

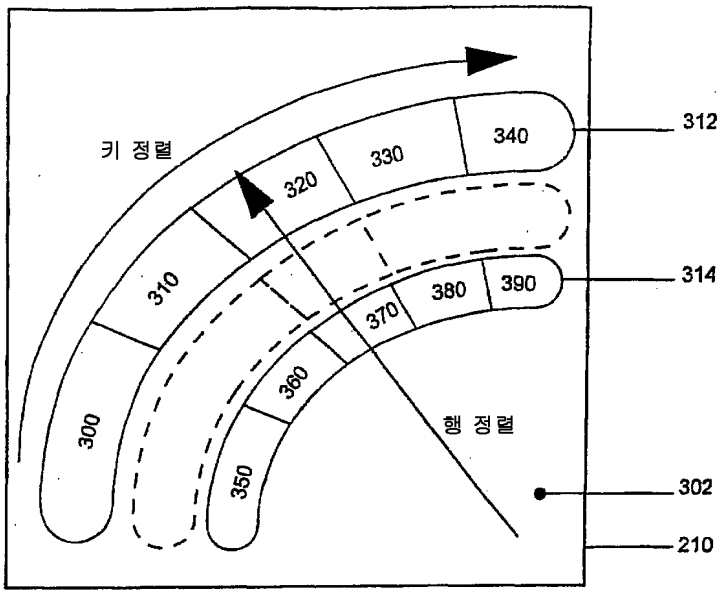
도면1


112
114
116
1
2
ABC
3
DEF
126
4
GHI
5
JKL
6
MNO
124
118
7
PQRS
8
TUV
9
WXYZ
122
120
*
0
#
140
100
130
110
128

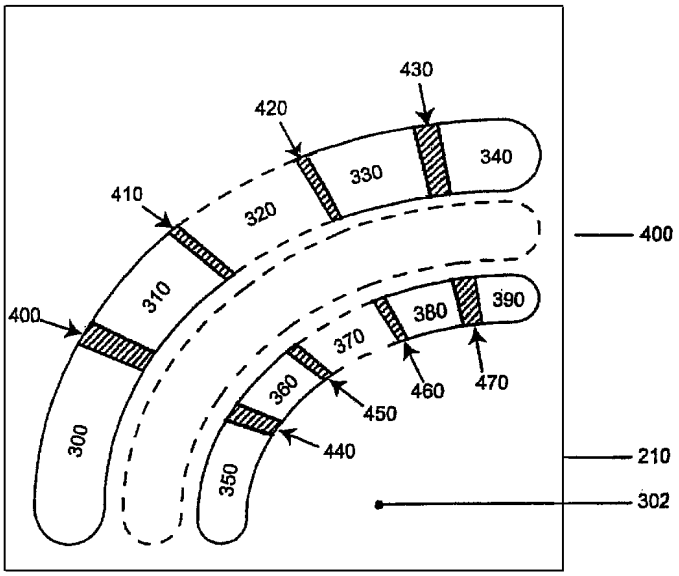
도면2


본체 센서 210
추가 센서 212
언어 모델
260
키스트로크
모듈
220
코어 로직
250
의미화 모듈
240
사용 콘텍스트
290
용법 모듈
270
200
메뉴 요소
282
텍스트 요소
280
출력 장치
284

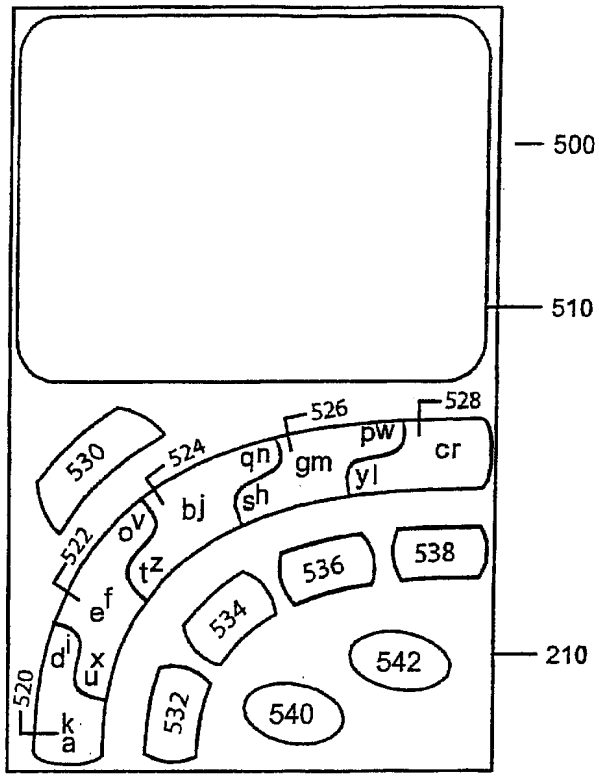
도면3


키 정렬
340
330
320
310
300
350
360
370
380
390
312
314
302
210
행 정렬

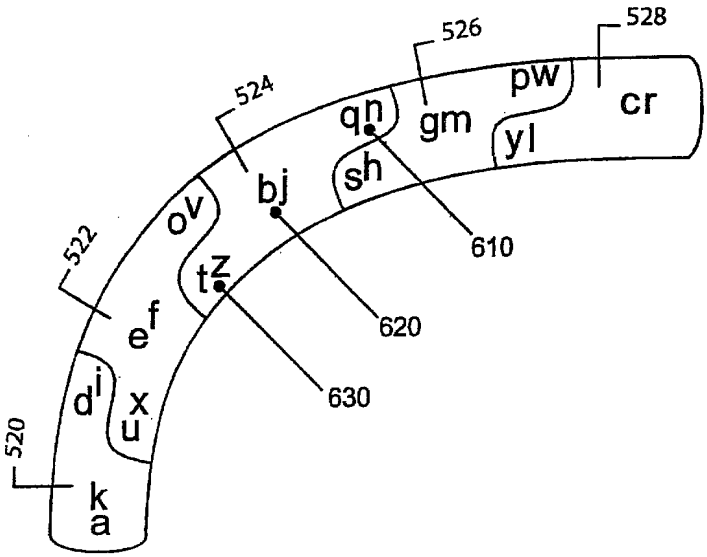
도면4


430
420
340
330
410
320
400
310
400
390
380
370
460
470
450
360
440
300
350
210
302

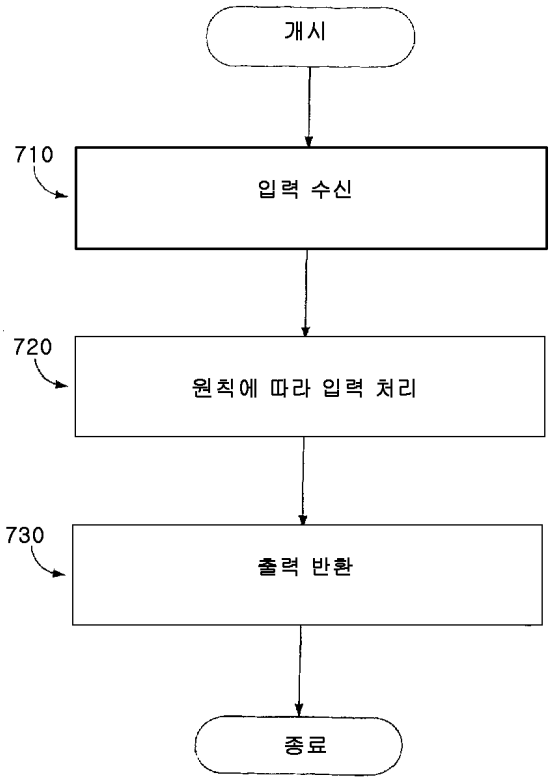
도면5


500
510
526
528
pw
cr
530
524
qn
gm
yl
bj
sh
ov
522
tz
ef
536
538
534
di
ux
542
210
520
ka
532
540

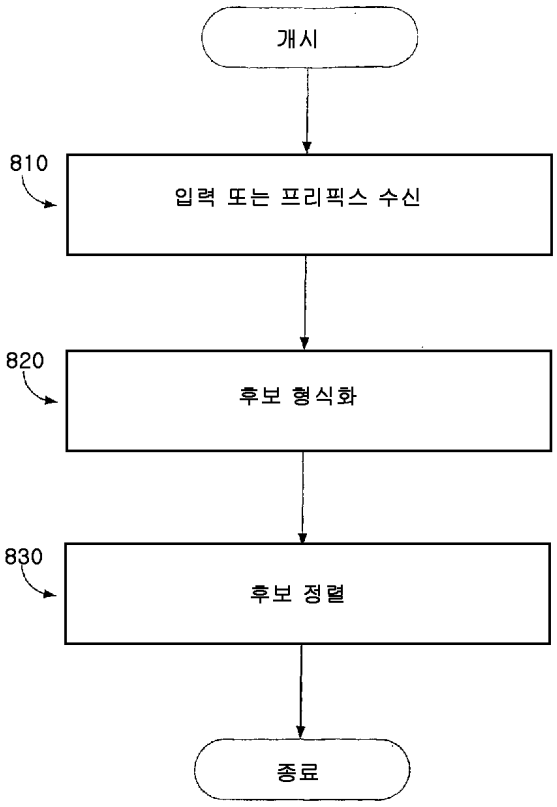
도면6


526
528
524
pw
cr
qn
gm
yl
bj
sh
ov
610
522
tz
620
ef
630
di
ux
520
ka

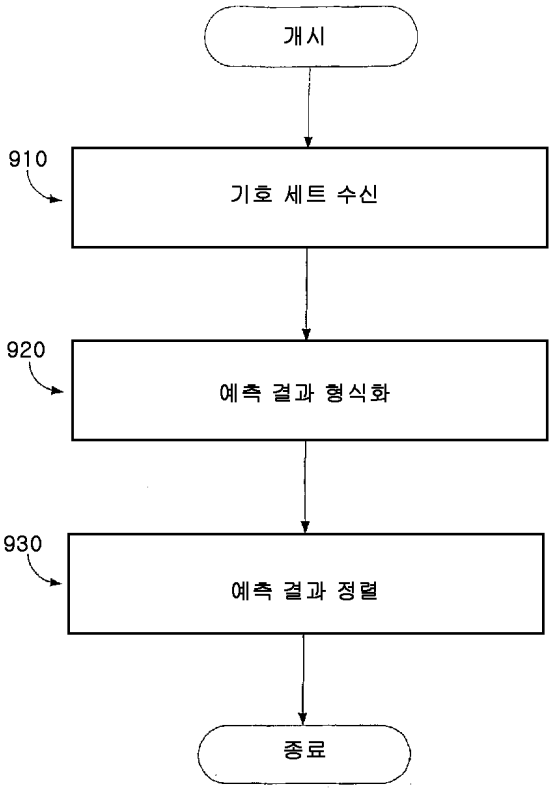
도면7


개시
710
입력 수신
720
원칙에 따라 입력 처리
730
출력 반환
종료

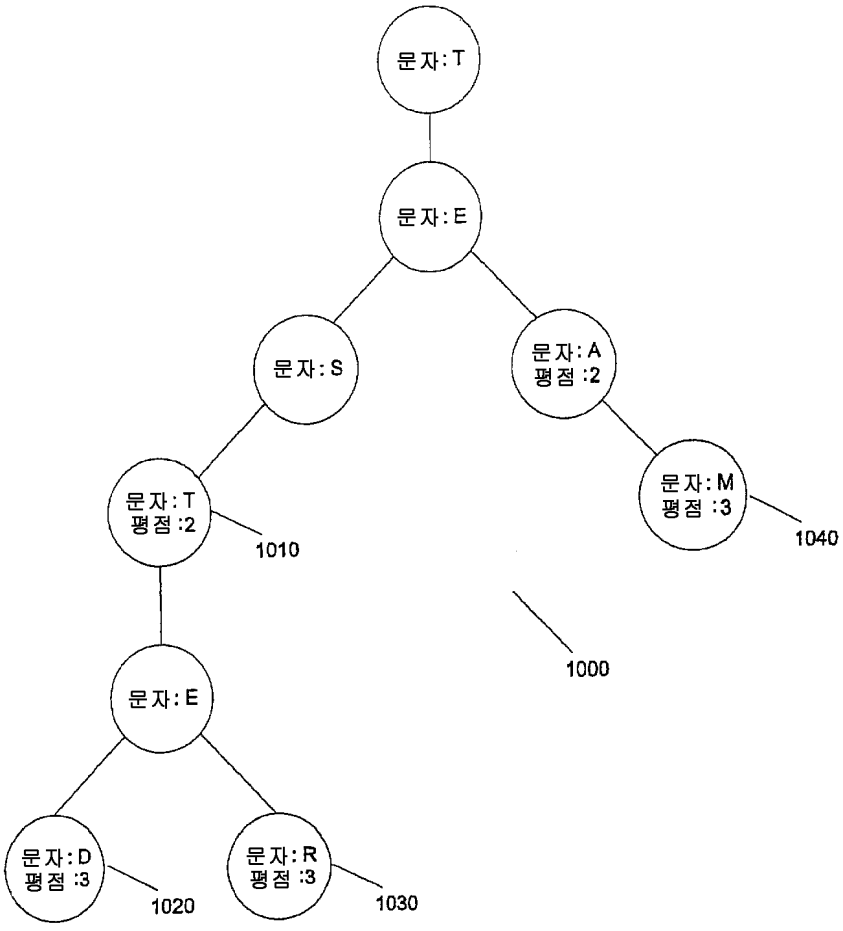
도면8


개시
810
입력 또는 프리픽스 수신
820
후보 형식화
830
후보 정렬
종료

도면9


개시
910
기호 세트 수신
920
예측 결과 형식화
930
예측 결과 정렬
종료

도면10


문자:T
문자:E
문자:S
문자:A
평점 :2
문자:T
평점 :2
1010
문자:M
평점 :3
1040
1000
문자:E
문자:D
평점 :3
1020
문자:R
평점 :3
1030