(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2020년04월23일
(11) 등록번호 10-2092721
(24) 등록일자 2020년03월18일

(51) 국제특허분류(Int. Cl.)
G06F 8/30 (2018.01) *G06F 11/36* (2006.01)
G06F 8/34 (2018.01) *G06F 8/35* (2018.01)
G06F 8/40 (2018.01) *G06F 9/448* (2018.01)
G06F 9/48 (2018.01)
(52) CPC특허분류
G06F 8/31 (2013.01)
G06F 11/3664 (2013.01)
(21) 출원번호 10-2018-7030472
(22) 출원일자(국제) 2017년03월23일
심사청구일자 2018년10월22일
(85) 번역문제출일자 2018년10월22일
(65) 공개번호 10-2018-0122018
(43) 공개일자 2018년11월09일
(86) 국제출원번호 PCT/US2017/023911
(87) 국제공개번호 WO 2017/165712
국제공개일자 2017년09월28일
(30) 우선권주장
62/312,106 2016년03월23일 미국(US)
(뒷면에 계속)
(56) 선행기술조사문헌
KR1020080012265 A
KR1020130130706 A

(73) 특허권자
포그혼 시스템스 인코포레이티드
미국 캘리포니아 94040 마운틴 뷰 스위트 100 웨스트 엘 카미노 레알 800
(72) 발명자
루카스 제이슨
미국 워싱턴 98059 렌턴 키트삽 플레이스 노스이스트 1800
샤르마 압히섹
미국 캘리포니아 94040 마운틴뷰 아파트먼트 230 컨티넨털 서클 707
(74) 대리인
특허법인우인

전체 청구항 수 : 총 19 항 심사관 : 지정훈

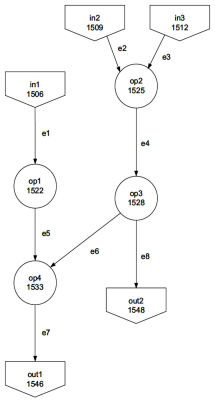
(54) 발명의 명칭 **실시간 데이터플로우 프로그래밍에서 패턴 구동형 반응의 구성**

(57) 요 약

기술은 다수의 데이터 입력의 스트림을 취하는 데이터플로우 그래프를 구현하며, 이러한 입력을 다수의 출력 스트림으로 변환한다. 데이터플로우 그래프는 패턴 매칭을 실행할 수 있다. 기술은 입력 데이터의 결합된 스트림에 매칭하는 패턴의 구성을 통해 반응을 구현한다. 입력 시퀀스를 특정한 입력패턴에 매칭시키는 완벽성은 적어도

(뒷면에 계속)

대 표 도 - 도15



차가운 상태 (아직 매칭되지 않음), 미지근한 상태 (예를 들면, 최소한 매칭됨), 및 뜨거운 상태 (예를 들면, 최대한 매칭됨) 와 같은 세가지 상이한 정도를 갖는 것으로 특징지을 수 있다. 매칭될 입력패턴은 0인 길이 내지, 제한이 없거나, 임의로 긴 길이를 포함하여 다양한 길이를 가질 수 있다. 데이터플로우는 푸시 베이스 또는 풀 베이스 이거나, 이들의 조합일 수 있으며, 상태에 의존하여 변할 수 있다.

(52) CPC특허분류
G06F 11/3668 (2013.01)
G06F 8/34 (2013.01)
G06F 8/35 (2019.02)
G06F 8/40 (2013.01)
G06F 9/4498 (2018.02)
G06F 9/48 (2013.01)

(30) 우선권주장
62/312,187 2016년03월23일 미국(US)
62/312,223 2016년03월23일 미국(US)
62/312,255 2016년03월23일 미국(US)

명 세 서

청구범위

청구항 1

센서에 결합되며, 각각 센서로부터 데이터를 수신하는 복수의 에이전트; 및

데이터 버스를 포함하며, 상기 복수의 에이전트에 결합되며, 일련의 상호 접속된 변환을 형성함으로서 입력 스트림을 출력 스트림으로 변환하는 데이터 처리 구성요소를 포함하며, 각 변환은,

외부 데이터 소스 또는 변환기의 적어도 한 개의 기준으로서 주어지는 한 개 이상의 입력;

외부 데이터 싱크 또는 변환기의 적어도 한 개의 기준으로서 주어지는 한 개 이상의 출력; 및

각 입력에 대해, 입력에 적용되고, 원래 입력을 필터링하고, 수집하고, 더 유용하고, 부분적으로 처리된 형태로 조직하고, 문제가 있는 입력을 제거하기 위한 패턴을 포함하는 것을 특징으로 하는 시스템.

청구항 2

제1항에 있어서,

적어도 한 개의 에이전트는,

컴퓨터 메모리 내의 제1 메모리 위치를 포함하는 입력큐, - 여기서 상기 입력큐는 처리될 토큰의 선입선출 시퀀스와 시간 스탬프를 상기 제1 메모리 위치에 저장하고, 상기 입력큐에서 각 토큰은 데이터를 포함하거나 포함하지 않으며, 상기 시간 스탬프는 상기 토큰이 상기 입력큐로 들어가는 시간을 나타내며, 상기 토큰은 네트워크를 통해 상기 입력큐에 의해 수신됨-;

상기 입력큐에 접속되며, 역추적 없이 상기 입력큐에서 상기 토큰을 처리하며, 한 개 이상의 기설정된 입력 패턴과 일치하는 토큰의 시퀀스에서의 패턴을 식별하며, 일치되는 입력 패턴을 식별시에, 출력되는 이벤트 출력을 생성하는 드라이버 구성요소;

상기 드라이버 구성요소에 접속되며 상기 컴퓨터 메모리 내에 제2 메모리 위치를 포함하며, 상기 드라이버 구성요소에 의해 생성된 출력되는 이벤트의 선입선출 시퀀스를 상기 제2 메모리 위치에 저장하는 출력큐;

상기 드라이버 구성요소에 접속되며, 상태 테이블 포맷으로 기설정된 입력 패턴을 저장하는 상태 테이블 구성요소; 및

상기 드라이버 구성요소에 접속되며, 상기 컴퓨터 메모리 내의 제3 메모리 위치를 포함하는 상태 스택 구성요소, - 여기서 상기 상태 스택 구성요소는 상기 제3 메모리 위치에 프레임의 후입선출 시퀀스 스토리지를 저장하고, 상기 프레임은 번역상태 번호, 기호, 및 데드라인을 포함함- 를 포함하는 것을 특징으로 하는 시스템.

청구항 3

제1항에 있어서,

각 변환은 필터링 식을 평가할 때 판단하기 위하여 사용되는 트리거링 식을 포함하는 것을 특징으로 하는 시스템.

청구항 4

제1항에 있어서,

트리거링 식은 노드들의 트리로서 정의되며,

각 노드는

입력중 한 개에 대한 기준,

시간 단위로, 시간 간격으로서 주어지는 타임아웃,

각각은 상이한 트리거링 식 노드로 표현되는 적어도 두 개의 자손을 포함하는 2진 공접 연산자 (binary conjunction operator), 및

각각 또 다른 트리거링 식 노드에 의해 표현되는 적어도 두 개의 자손을 포함하는 2진 이접연산자 (binary disjunction operator) 중 적어도 하나를 가지는 것을 특징으로 하는 시스템.

청구항 5

제3항에 있어서

각 변환은 필터링 식을 포함할 수 있으며, 상기 필터링 식은 매칭된 입력의 도메인에서 지정되고, 상기 필터링 식은 변환이 출력을 생산할 때를 결정하는 불린(Boolean)결과를 산출하는 것을 특징으로 하는 시스템.

청구항 6

제1항에 있어서,

입력패턴은 0인 길이 또는 무제한의 길이를 포함하는 가변 길이의 입력 시퀀스를 매칭시키는 것을 특징으로 하는 시스템.

청구항 7

제1항에 있어서,

입력패턴을 입력 시퀀스에 매칭시키고자 하는 시도의 결과는 적어도 세 개의 상이한 상태를 가질 수 있는 것을 특징으로 하는 시스템.

청구항 8

제1항에 있어서,

입력패턴을 입력 시퀀스에 매칭시키고자 하는 시도의 결과는 적어도 세 개의 상이한 상태를 가질 수 있으며, 상기 적어도 세 개의 상이한 상태는,

입력 패턴의 최소한의 입력이 매칭될 때까지, 매칭을 위한 시도인 첫번째 상태,

최소한의 입력이 매칭된 후 및 최대한의 입력이 매칭되기 전, 매칭을 위한 시도인 두번째 상태, 및

최대한의 입력이 매칭된 후, 매칭을 위한 시도인 세번째 상태를 포함하는 것을 특징으로 하는 시스템.

청구항 9

제8항에 있어서,

첫번째 상태는 "차가운" 상태라고 명명되며, 두번째 상태는 "따뜻한" 상태라고 명명되며, 세번째 상태는 "뜨거운" 상태라고 명명되는 것을 특징으로 하는 시스템.

청구항 10

제1항에 있어서,

데이터플로우는 센서로부터 에이전트까지의 푸시 기반인 것을 특징으로 하는 시스템.

청구항 11

제8항에 있어서,

데이터플로우는 센서로부터 에이전트가지 푸시 기반이며, 데이터플로우는 에이전트의 모든 입력이 적어도 두번째 상태에 있으며, 센서의 한 개 이상의 입력이 세번째 상태에 있을 경우, 센서로부터 에이전트에 의해 풀 기반이 되며, 트리거링 식의 결과는 두번째 상태에 있는 것을 특징으로 하는 시스템.

청구항 12

제11항에 있어서,

상기 에이전트는 두번째 상태에 있는 센서를 가속화하여, 그러한 센서가 패턴이 완전히 매칭되었다고 간주하여 상기 세번째 상태가 되게 하며, 그 출력을 생산하도록 강제하는 것을 특징으로 하는 시스템.

청구항 13

제12항에 있어서,

명확히 길거나, 오픈 엔트 패턴이 변환에 유용하게 구현되는 것을 특징으로 하는 시스템.

청구항 14

제1항에 있어서,

한 개 이상의 센서는 물리적 속성을 스트림인 디지털 양으로 변환시키는 하드웨어 장치인 것을 특징으로 하는 시스템.

청구항 15

다수의 에이전트와 센서를 서로 연결하는 단계, - 여기서 상기 에이전트는 상기 센서에 연결되며, 상기 센서로부터 데이터의 스트림을 수신함-; 및

상기 센서로부터의 입력 스트림을 일련의 접속된 변환을 통해 출력 스트림으로 변환하는 단계를 포함하며, 각각의 변환은

기준으로서 외부 데이터 소스 또는 변환기에 주어지는 한 개 이상의 입력,

기준으로서 외부 데이터 싱크 또는 변환기에 주어지는, 한 개 이상의 출력,

각각의 입력에 대하여, 입력에 적용되고, 원래 입력을 필터링하고, 수집하고, 더 유용하고, 부분적으로 처리되며 매칭된 형태로 조직하고, 문제가 있는 입력을 제거하기 위한 패턴, 및

필터링 식을 평가할 때를 판단하기 위하여 사용되고, 노드들의 트리로서 정의되는 트리거링 식을 포함하며, 각각의 노드는

입력중 한 개에 대한 기준,

편리한 시간 단위로 특징지어지는 시간 간격으로서 주어지는 타임아웃;

각각이 상이한 트리거링 식 노드로 표현되는 적어도 두 개의 자손을 포함하는 2진 공접 연산자 (binary conjunction operator), 및

각각 또 다른 트리거링 식 노드에 의해 표현되는 적어도 두 개의 자손을 포함하는 2진 이접연산자 (binary disjunction operator),

상기 매칭된 입력의 도메인에서 정의되고 변환이 출력을 생산할 때를 결정하는 불린(Boolean)결과를 산출하는 필터링 식, 및

상기 매칭된 입력의 도메인에서 정의되고 변환의 출력을 생산하는 유형의 임의의 조합의 결과를 산출하는 한 개 이상의 식 중 적어도 하나를 포함하는 것을 특징으로 하는 방법.

청구항 16

제15항에 있어서,

입력 패턴은 길이가 0 또는 무제한의 길이를 포함하는 가변길이의 입력 시퀀스를 매칭시킬 수 있으며, 최소한의 입력이 매칭될 때까지는 차가운 상태, 최소한의 입력이 매칭된 후, 최대한의 입력이 매칭되기 전은 따뜻한 상태, 최대한의 입력이 매칭된 후는 뜨거운 상태라고 각각 간주되는 것을 특징으로 하는 방법.

청구항 17

제15항에 있어서,

매칭을 위한 변환기의 준비상태의 완전성 모델은 두 개 이하의 상이한 정도 보다는 적어도 세 개의 상이한 정도에 의해 특징지워지는 것을 특징으로 하는 방법.

청구항 18

제17항에 있어서,

상기 세 가지 정도의 완전성 모델은 패턴 구동형 반응이 가변 길이 패턴을 조절하는 것을 가능하도록 하는 것을 특징으로 하는 방법.

청구항 19

제16항에 있어서,

생산자로부터 소비자로의 데이터플로우는 푸시기반이며, 에이전트의 모든 입력이 적어도 따뜻한 상태에 있는 경우와 센서의 한 개 이상의 입력이 뜨거운 상태에 있을 경우, 및 상기 트리거링 식의 결과는 따뜻한 상태인 것을 제외하고,

상기 에이전트는 따뜻한 상태에 있는 센서를 가속화하여, 그러한 센서가 그것의 패턴이 완전히 매칭되었다고 간주하여 뜨거운 상태가 되게 하며, 그것의 출력을 생산하도록 강제하며,

상기 센서가 거기에 연결된 다른 생산자에게 데이터를 끌어당기도록 하며,

명확히 길거나, 오픈 엔드 패턴이 변환에 유용하게 채용하도록 하는 것을 특징으로 하는 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 컴퓨팅 분야에 관한 것이며, 더욱 상세하게는 산업기계에 의해 생성된 다량의 데이터를 처리하기 위해 에지 컴퓨팅에서 사용될 수 있는 실시간 데이터흐름 프로그래밍에서 패턴으로 구동되는 반응의 구성에 관한 것이다.

배 경 기 술

[0002] 전통적인 기업 소프트웨어 애플리케이션 호스팅은 데이터 센터 또는 "클라우드" 인프라에 의존하여 규모의 경제 및 시스템 효율을 활용한다. 하지만, 이러한 데이터센터는 기업이 대부분의 비즈니스 운영을 하는 물리적인 운영 지점 (예를 들면, 공장, 창고, 소매점, 및 기타)으로부터 임의로 떨어져있을 수 있다. 산업용 사물 인터넷 (IIoT) 은 매우 높은 빈도의 이벤트를 추적하는 센서로 물리적인 운영의 기기에 의존하는 장치나 사용사례의 수집을 의미한다.

[0003] 제조, 석유 및 가수, 광업, 운송, 전력 및 물, 재생 에너지, 헬스케어, 소매, 스마트 빌딩, 스마트 시티, 및 연결된 차량을 포함하는 사물 인터넷 (IoT) 아래에 여러 분야의 산업용 기계가 있다. 클라우드 컴퓨팅의 성공에도 불구하고, 많은 단점이 존재한다. 연결이 항상 존재하는 것이 아니고, 대역폭이 충분하지 않으며, 대기시간의 변화가 너무 길고, 대역폭이 존재한다고 하더라도 비용이 너무 높기 때문에, 모든 데이터를 클라우드 스토리지로 보내는 것이 실용적이지 않다. 연결성, 대역폭, 및 비용이 문제가 되지 않더라도, 실시간 의사결정 및 예측 유지관리가 존재하지 않아 기계에 심각한 손상을 초래할 수 있다.

[0004] 그러므로, 산업기계에서 생성된 다량의 데이터를 처리하기 위해 개선된 컴퓨팅 시스템, 아키텍처, 및 개선된 에지 분석 및 데이터 흐름 프로그래밍을 포함하는 기술 이 필요하다.

발명의 내용

해결하려는 과제

[0005] 기술은 다수의 데이터 입력의 스트림을 취하는 데이터플로우 그래프를 구현하며, 이러한 입력을 다수의 출력 스트림으로 변환한다. 데이터플로우 그래프는 패턴 매칭을 실행할 수 있다. 기술은 입력 데이터의 결합된 스트림에 매칭하는 패턴의 구성을 통해 반응을 구현한다. 입력 시퀀스를 특정한 입력패턴에 매칭시키는 완벽성은 적어도 차가운 상태 (아직 매칭되지 않음), 미지근한 상태 (예를 들면, 최소한 매칭됨), 및 뜨거운 상태 (예를

들면, 최대한 매칭됨) 와 같은 세가지 상이한 정도를 갖는 것으로 특징지을 수 있다. 매칭될 입력패턴은 0인 길이 내지, 제한이 없거나, 임의로 긴 길이를 포함하여 다양한 길이를 가질 수 있다. 데이터플로우는 푸시 베이스 또는 풀 베이스 이거나, 이들의 조합일 수 있으며, 상태에 의존하여 변할 수 있다.

과제의 해결 수단

[0006] 특별한 일 구현에서, 데이터플로우 프로그래밍 기술이 에지 컴퓨팅 시스템에서 사용된다. 방법은 에지에서 지능화를 가능하게 한다. 특징은, 게이트웨이 장치 또는 내장된 시스템상에 호스팅된 소프트웨어 계층에서 센서데이터에 의해 트리거링하는 것을 포함한다. 소프트웨어 계층은 근거리 통신망에 접속된다. 서비스, 애플리케이션, 및 데이터처리 엔진의 리포지토리 (repository) 는 소프트웨어 계층에 의해 접근가능하다. 소프트웨어 계층에 의해 가능한 센서데이터를 표현 언어를 통한 특정 조건의 등장의 구문적 기술(descriptions)과의 매칭. 계속적으로 표현식을 실행함으로서 패턴 이벤트의 자동발견, 게이트 장치를 가로지르는 서비스 및 애플리케이션 및 애플리케이션 및 해석 표현식을 체인연결하는 소프트웨어 계층에 의해 관리되는 네트워크 상의 내장 시스템을 지능적으로 결합, 리소스 유용성에 기초한 애플리케이션 및 분석의 레이아웃의 최적화, 소프트웨어 계층의 건강의 모니터링, 원시 센서 데이터 또는 로컬 타임 계열 데이터베이스 또는 클라우드 스토리지에 센서 데이터 또는 표현식의 결과의 저장을 포함한다. 서비스 및 구성요소는 컨테이너로 운반되어 임의의 게이트웨이 환경에서의 매끄러운 실행을 보증한다.

[0007] 에지 인텔리전스는 사물인터넷 (IoT) 데이터의 소스에서 가능하다. 시스템은 실시간 에지 분석 및 애플리케이션에 대한 IoT 장치 젠서 데이터로의 풍부한 액세스 (스트림 또는 배치모드, 또는 둘다) 를 제공한다. 시스템은 낮은 메모리 풋프린트 기계에서 동작하는 고성능 분석엔진을 통해, 해석 함수 및 식을 실행하는 고도로 효율적이고 표현적인 컴퓨터 언어를 포함한다. 시스템은 클라우드로 집계 데이터를 게시하여, 추가의 머신 러닝을 가능하게 한다. 시스템은 에지 앱을 개발하기 위한 소프트웨어 개발 키트를 포함한다. 클라우드 기반 매니지먼트 콘솔은 이제 전개, 구성, 응용, 및 해석표현의 관리를 가능하게 한다.

[0008] 에지 인프라 및 플랫폼의 특정한 구현은 FogHorn Systems, Inc. 포그혼(FogHorn) 에 의해 행해진다. FogHorn 웹사이트 www.foghorn-systems.com, 출판물 (백서, 사용자 가이드, 튜토리얼, 비디오, 기타를 포함) 및 FogHorn 기술에 관한 다른 출판물, 및 제품이 참조로서 포함된다.

[0009] FogHorn 은 산업용 및 상업용 사물 인터넷(IoT) 데이터에 대한 에지 지능화를 가능하게 하기 위한 플랫폼을 제공한다. 수십억의 산업용 및 상업용 사물 인터넷(IoT)장치에 의해 생성된 데이터의 양은 전체 인터넷을 제압할 수 있을 만큼 충분히 거대하다. FogHorn 플랫폼은 그것이 유래된-네트워크의 에지에서 IoT 데이터를 처리하고, 분석하며, 반응한다. FogHorn의 "intelligent edge" 소프트웨어 플랫폼은 전례없는 수준의 자동화, 운영효율, 비용절감 등을 가능하게 한다.

[0010] 산업용 사물인터넷(IIoT)은 센서, 기계, 및 컴퓨터 등과 같은 상호접속된 산업용 및 상업용 장치로 이루어진다. IIoT의 목표는 더 큰 장치제어, 데이터 매니지먼트, 기계 자동화, 및 분산된 기업에서의 동작효율을 더 크게하는 것이다. 회사들은 에지에서, 포크연산을 적용하여 실시간 분석 및 자동화된 반응을 사용하여, 개발된 적이 없는 IIoT 기회를 포착할 수 있지만, 도한, 시스템 폭 매니지먼트 및 최적화를 위한 클라우드 연산을 저울질하고 있다. 만약 추가 컴퓨팅 리소스를 추가하는 것이 가능하지 않다면, FogHorn 에지 컴퓨팅 플랫폼은 또한 기존의 프로그램가능한 논리 콘트롤러 (PLCs)(예를 들면, 브라운필드(brownfield) 기회)에서 실행하도록 디자인되었다. 브라운필드는 설립된 시스템을 차지하는 동안, 정보기술(IT) 문제 분야를 해결하기 위한 새로운 시스템의 구현을 말한다. 새로운 소프트웨어 아키텍처는 기존의 구동중인 소프트웨어를 고려한다.

[0011] 에지 인텔리젼트 플랫폼은 IIoT 장치가 상주하는 에지에 가까운 데이터 처리 및 분석을 늘이는 포그 연산 개념에 의존하는 소프트웨이 기반 솔루션이다. 모든 데이터를 멀리 떨어진 중앙집중형 클라우드에 보내는 것보다는 에지장치와 아주 가깝게 유지하는 것이 최대성능, 더 빠른 반응시간, 및 더효율적인 매인터넌스 및 동작정책을 가능하게 한다. 그것은 또한 전체적인 밴드폭 요구조건 및 넓은 분산 네트워크를 관리하는 비용을 현저하게 감소시킨다.

[0012] 에지에서의 IIoT 동작에 촛점을 맞추는 것은 모든 밴드폭 요구조건을 감소시키며, 시간에 민감한 조건에 반응하여 즉시 자동화가 가능하게 된다. 산업분야에서, 10억개의 새로운 IIoT 장치를 총괄적으로 추가하고, 이러한 장치는 매일 많은 페타바이트의 데이터를 생성한다. 이러한 모든 데이터를 클라우드에 보내는 것은 엄청난 비용이 들뿐 아니라, 더 큰 보안상의 위험을 야기한다. 에지에서의 동작은 더 빠른 동작시간, 줄어든 위험, 및 전체적으로 더 낮은 비용을 확보해준다.

[0013] 2015년 8월27일에 출원된 미국특허출원 제62/210981호 및 2016년 8월29일에 출원된 제15/250720호는 참조로서 포함되며 에지컴퓨팅 환경 및 플랫폼을 기재한다. 2017년 3월23일에 출원된 미국특허출원 제15/467,306호는 참조로서 포함되며 실시간 데이터플로우 프로그래밍을 위한 효율적인 상태 머신을 기재한다. 2017년 3월23일에 출원된 미국특허출원 제15/467,318호는 참조로서 포함되며 실시간 데이터플로우 프로그래밍 언어를 위한 도구 및 방법을 기재한다.

[0014] 일 구현에 있어서, 시스템은 센서에 접속되며, 센서로부터 데이터 (예를 들면, 스트림 데이터)를 수신하는 다수의 에이전트 및 에이전트가 접속되는 데이터버스를 포함하는 데이터 처리 구성요소를 포함한다. 데이터 처리 구성요소는 일련의 상호접속된 변환을 형성함으로서 입력 스트림을 출력 스트림으로 변환한다. 변환은 패턴매칭, 연산, 및 다른 동작을 포함할 수 있다. 각 변환은 기준으로서 외부 데이터 소스 또는 변환기 중 적어도 하나가 주어지는 한 개 이상의 입력, 외부 데이터 싱크 또는 변환기 중 적어도 하나에 기준으로서 주어지는 한 개 이상의 출력을 포함할 수 있다. 입력패턴을 입력 시퀀스에 매칭시키고자 하는 시도의 결과는 적어도 세 개의 상이한 상태 또는 완벽도를 가질 수 있다.

[0015] 또 다른 구현에 있어서, 방법은 다수의 에이전트와 센서를 서로 연결하는 단계로서, 에이전트는 센서에 연결되며, 센서로부터 데이터의 스트림을 수신하는 단계, 및 센서로부터의 입력 스트림을 일련의 접속된 변환을 통해 출력 스트림으로 변환하는 단계를 포함한다. 각 변환은 기준으로서 외부 데이터 소스 또는 변환기 중 적어도 하나가 주어지는 한 개 이상의 입력, 외부 데이터 싱크 또는 변환기 중 적어도 하나에 기준으로서 주어지는 한 개 이상의 출력을 포함한다. 각 입력에 대해, 입력에 적용되고, 원래 입력을 필터링하고, 수집하고, 더 유용하고, 부분적으로 처리되거나, 매칭된 형태로 조직하고, 문제가 있는 입력을 제거하기 위한 패턴이 존재한다. 변환에서 트리거링 식은 필터링 식을 평가할 때를 판단하기 위하여 사용될 수 있다.

[0016] 본 발명의 다른 목적, 특징, 및 장점은 이하의 상세한 설명 및 첨부도면을 참조하면 더욱 명백해지며 도면 전체에 걸쳐서 유사한 참조번호는 동일한 특징을 나타낸다.

도면의 간단한 설명

[0017] 도1은 클라이언트 서버 시스템 및 네트워크의 블록도를 도시한다.

도2는 클라이언트 또는 서버의 더 상세한 도면을 도시한다.

도3은 컴퓨터 시스템의 블록도를 도시한다.

도4는 센서 스트림과 클라이드 사이의 에지 컴퓨팅 플랫폼의 블록도이다.

도5는 에지 분석을 포함하는 에지 컴퓨팅 플랫폼의 더 상세한 블록도를 도시한다.

도6은 에지 인프라와 클라우드 인프라 사이의 동작흐름을 도시한다.

도7은 결정성 유한 오토마톤 (DFA) 및 상태 감소된 기계로 변환된 강화된 비결정성 유한 오토마톤 (NFA) 을 도시한다.

도8을 토큰 알파의 수신하에 상태 A로부터 B로의 전이를 도시한다.

도9는 추가 상태 전이인 상태 X를 거친 상태 A로부터 상태B로의 전이를 도시한다.

도10은 구문해석에 의해 형성된 추상구문트리의 일 예를 도시한다.

도11은 교대에 대한 서브그래프를 도시한다.

도12는 결합에 대한 서브그래프를 도시한다.

도13은 구조를 갖는 클로저를 도시한다.

도14는 센서 표현 언어 엔진을 통해 일부 가상 센서를 생성하기 위해 물리적 센서를 사용하는 예를 도시한다.

도15는 흐름 그래프의 예를 도시한다.

도16은 노드의 종류의 계층을 도시한다.

도17은 에지의 종류 및 노드에 대한 카디널리티 (cardinality) 의 계층을 도시한다.

발명을 실시하기 위한 구체적인 내용

[0018] 도1은 본 발명의 일실시예를 구체화하는 분산 컴퓨터 네트워크 (100) 의 개략 블록도이다. 컴퓨터 네트워크(100)는 복수의 통신 링크 (128)를 통해 통신 네트워크(124)와 연결된 다수의 클라이언트 시스템 (113, 116, 110) 및 서버시스템 (122)를 포함한다. 통신 네트워크(124)는 분산 네트워크100의 다양한 구성요소가 서로 통신하고 정보를 교환할 수 있게 하는 매커니즘을 제공한다.

[0019] 통신 네트워크(124) 자체는 서로 연결된 많은 컴퓨터 시스템 및 통신 링크로 이루어질 수 있다. 통신 링크(128)는 하드웨어 링크, 광 링크, 위성 또는 다른 무선 통신 링크, 파동 전파 링크, 또는 정보의 통신을 위한 다른 임의의 매커니즘일 수 있다. 통신 링크 (128) 은 DSL, 케이블, 이더넷, 또는 다른 하드웨어 링크, 수동 또는 능동 광 링크, 3G, 3.5G, 4G 및 다른 이동성, 위성, 또는 다른 무선 통신 링크, 파동 전파 링크, 또는 정보 통신을 위한 다른 임의의 메커니즘일 수 있다.

[0020] 다양한 통신 프로토콜이 도1에 도시된 다양한 시스템 사이의 통신을 가능하게 하기 위하여 사용될 수 있다. 이러한 통신 프로토콜은 VLAN, MPLS, TCP/IP, 터널링, HTTP 프로토콜, 무선 응용 통신 프로토콜 (WAP), 벤더 고유의 프로토콜, 맞춤형 프로토콜 등을 포함할 수 있다. 일 실시예에서, 통신 네트워크(124)는 인터넷이지만, 다른 실시예에서, 통신 네트워크(124)는 근거리 통신망 (LAN), 광역 네트워크 (WAN), 무선 네트워크, 인트라넷, 전용 네트워크, 공용 네트워크, 교환망 및 이들의 결합을 포함하는 임의의 적절한 통신 네트워크 일 수 있다.

[0021] 도1의 분산 컴퓨터 네트워크(100) 은 본 발명을 구체화하는 일실시예의 일 예일 뿐이며 청구항에서 기재된 본 발명의 범위를 제한하는 것은 아니다. 본 발명이 속하는 기술 분야의 당업자는 다른 변형, 변경, 및 수정을 인식할 것이다. 예를 들면, 한 개 이상의 서버 시스템(122)가 통신 네트워크(124)에 접속될 수 있다. 다른 예로서, 다수의 클라이언트 시스템(113, 116, 119)가 접속 제공자(미도시) 또는 다른 서버 시스템을 통해 통신 네트워크(124)에 접속될 수 있다.

[0022] 일반적으로, 클라이언트 시스템(113, 116, 119)는 정보를 제공하는 정보시스템으로부터 정보를 요청한다. 이 때문에, 일반적으로 서버 시스템은 클라이언트 시스템 보다 연산 및 저장 용량이 더 크다. 하지만, 특정한 컴퓨터 시스템은 컴퓨터 시스템이 정보를 요청하는지 또는 제공하는 지에 따라 클라이언트 또는 서버 양자로서 기능할 수 있다. 또한, 본 발명의 태양이 클라이언트 서버 환경을 사용하는 것으로 기재되었지만, 본 발명은 또한 독립형 컴퓨터 시스템으로도 구현될 수 있다는 것은 명백하다.

[0023] 서버(122)는 클라이언트 시스템(113,116, 119) 로부터 정보요청을 수신하고, 요청을 만족시키기 위해 필요한 처리를 수행하며, 요청하는 클라이언트 시스템으로 요청에 대응되는 결과를 포워딩하는 역할을 한다. 요청을 만족시키기 위해 필요한 처리는 서버시스템 (122)에 의해 행해질 수 있으며, 그 대신에 통신 네트워크(124)에 접속된 다른 시스템에 위임될 수도 있다.

[0024] 클라이언트 시스템(113, 116, 119)는 사용자가 서버시스템(122)에 의해 저장된 정보에 접속하고 질의할 수 있도록 한다. 특정한 실시예에서, 클라이언트 시스템은 데스크탑 애플리케이션, 모바일 스마트폰, 또는 테블릿 애플리케이션과 같은 독립형 애플리케이션으로서 동작할 수 있다. 또 다른 실시예에서, 클라이언트 시스템 상에서 실행하는 "웹 브라우저" 애플리케이션은 사용자에게 서버 시스템(122)에 의해 저장된 정보를 선택, 접속, 검색, 또는 질의할 수 있도록 한다. 웹브라우저의 예로는 마이크로소프트의 인터넷 익스플로러 브라우저 프로그램, 모질라의 파이어폭스 브라우저, 구글의 크롬 브라우저, 애플의 사파리 브라우저 등이 있다.

[0025] 클라이언트 서버 환경에서, 일부 자원 (예를 들면, 파일, 음악, 비디오, 또는 데이터)은 클라이언트 측에 저장되지만, 다른 자원은 서버와 같이 네트워크의 다른 곳에 저장되거나 전달되며 네트워크 (예를 들면, 인터넷) 를 통해 접속가능하다. 그러므로 사용자의 데이터는 네트워크 또는 "클라우드"에 저장될 수 있다. 예를 들면, 사용자는 클라우드 (예를 들면, 서버)에 원격으로 저장된 클라이언트 장치상에서 문서작업을 할 수 있다. 클라이언트 장치 상의 데이터는 클라우드와 동기화 될 수 있다.

[0026] 도 2는 본 발명의 예시적인 클라이언트 또는 서버 시스템을 도시한다. 일 실시예에서, 사용자는 도2에 도시된 바와 같이 컴퓨터 워크스테이션 시스템을 통해 시스템과 인터페이스한다. 도2는 모니터(203), 스크린(205), 엔클로저(207) (또한, 시스템 유닛, 캐비넷, 또는 케이스라고도 함), 키보드, 또는 다른 인체 입력장치 209, 및 마우스 또는 다른 포인팅 장치(211) 을 포함하는 컴퓨터 시스템(201)을 도시한다. 마우스(211)은 마우스 버튼(213)과 같은 한 개 이상의 버튼을 갖는다.

[0027] 본 발명이 특정한 폼 팩터 (예를 들면, 데스크 탑 컴퓨터의 폼 팩터) 를 갖는 임의의 컴퓨팅장치로 제한되는 것

이 아니며, 다양한 폼 팩터의 모든 유형의 컴퓨팅 장치를 포함할 수 있다는 것을 이해해야 한다. 사용자는 데이터를 수신 또는 전송할 수 있는 스마트폰, 개인용 컴퓨터, 랩탑, 전자 태블릿 장치, GPS 수신기, 휴대용 미디어 플레이어, 개인용 정보 단말 (PDA), 다른 네트워크 접속 장치, 및 다른 처리 장치를 포함하는 임의의 컴퓨팅 장치와 인터페이스 할 수 있다.

[0028] 예를 들면, 특별한 구현으로서, 클라이언트 장치는 애플 아이폰 (예를 들면, 애플 아이폰6), 애플 아이패드(예를 들면, 애플 아이패드 또는 애플 아이패드 미니), 애플 아이팟 (예를 들면, 애플 아이팟 터치), 삼성 갤럭시 제품 (예를 들면, 갤럭시 S 시리즈 제품 또는 갤럭시 노트 시리즈 제품), 구글 넥서스 장치 (예를 들면, 구글 넥서스 6, 구글 넥서스 7, 또는 구글 넥서스 9), 및 마이크로소프트 장치 (예를 들면, 마이크로소프트 서피스 태블릿) 과 같은 스마트 폰, 또는 테블릿 장치일 수 있다. 일반적으로, 스마트폰은 터치스크린 디스플레이를 통해 접속가능한 전화부분 (및 관련된 라디오) 및 컴퓨터 부분을 포함한다.

[0029] 전화부분 (예를 들면, 연락처 및 전화번호) 및 컴퓨터 부분 (예를 들면, 브라우저, 사진, 게임, 비디오, 및 음악을 포함하는 애플리케이션 프로그램) 의 데이터를 저장하기 위한 비휘발성 메모리가 제공된다. 일반적으로, 스마트폰은 사진 및 비디오를 촬영하기 위한 카메라 (전방을 향하는 카메라, 후면 카메라, 또는 양자모두)를 포함한다. 예를 들면, 스마트폰 또는 테블릿은 한 개 이상의 다른 장치로 스트리밍 될 수 있는 라이브 비디오를 촬영하기 위해 사용될 수 있다.

[0030] 엔클로저(207)는 프로세서, 메모리, 대용장 저장장치(217) 등과 같은 친숙한 컴퓨터 구성요소 (이들 중 일부는 도시하지 않음) 를 수용한다. 대용량 저장장치(217)는 대용량 디스크 드라이브, 플로피 디스크, 자기 드시크, 광디스크, 광자기 디스크, 고정형 디스크, 하드 디스크, CD-ROM, 기록가능 CD, DVD, 기록가능 DVD (예를 들면, DVD-R, DVD+R, DVD-RW, DVD+RW, HD-DVD, 또는 블루레이 디스크), 플래시 및 기타 비휘발성 고체 저장장치 (예를 들면, USB 플래시 드라이브 또는 고체상태 드라이브 (SSD), 배터리 백업된 휘발성 메모리, 테이프 저장장치, 판독장치, 및 다른 유사한 매체, 및 이들의 조합을 포함할 수 있다.

[0031] 본 발명의 컴퓨터로 구현되거나 컴퓨터로 실행가능한 버전 또는 컴퓨터 프로그램 제품은 컴퓨터로 판독가능한 매체에 저장되거나 관련되어 구현될 수 있다. 컴퓨터로 판독가능한 매체는 실행을 위한 한 개 이상의 프로세서에 명령을 제공하는데 참여하는 임의의 매체를포함할 수 있다. 그러한 매체는 비휘발성, 휘발성, 및 전송형 매체를 포함하지만, 이에 한정되지 않는 많은 형태를 취할 수 있다. 예를 들면, 비휘발성 매체는 플래시 메모리, 광 디스크 또는 자기 디스크를 포함한다. 휘발성 매체는 캐쉬 메모리 또는 RAM 과 같은 정적 또는 동적 메모리를 포함한다. 전송 매체는 동축 케이블, 구리선, 광섬유선, 및 버스 내에 배열된 배선을 포함한다. 전송매체는 라디오파 또는 적외선 데이터 통신 동안 생성된 것과 같은, 전자기, 무선 주파수, 음향 또는 광파의 형태를 취할 수 있다.

[0032] 예를 들면, 본 발명의 소프트웨어의 이진의 기계로 실행가능한 버전이 RAM 또는 캐시 메모리, 또는 대용량 저장장치(217)에 저장되거나 상주할 수 있다. 본 발명의 소프트웨어의 소스 코드는 또한 대용량 장치(217) (예를 들면, 하드디스크, 자기 디스크, 테이프, 또는 CD-ROM) 상에 저장되거나 상주할 수 있다. 또 다른 예로서, 본 발명의 코드는 유선, 라디오파, 또는 인터넷과 같은 네트워크를 통해 전송될 수 있다.

[0033] 도3은 본 발명의 소프트웨어를 실행하기 위해 사용되는 컴퓨터시스템(201)의 블록도를 도시한다. 도2에 도시한 바와 같이, 컴퓨터 시스템(201)은 모니터(203), 키보드(209), 및 대용량 저장장치(217)을 포함한다. 컴퓨터 시스템(201)은 또한 중앙처리장치(302), 시스템 메모리(304), 입/출력(I/O)제어기(306), 디스플레이 어댑터(308), 직렬 또는 범용 시리얼 버스(USB) 포트(312), 네트워크 인터페이스(318), 및 스피커(320)와 같은 서브 시스템을 더 포함한다. 본 발명은 또한 추가의 또는 더 적은 수의 서브 시스템을 갖는 컴퓨터 시스템을 사용할 수 있다. 예를 들면, 컴퓨터 시스템은 한 개 이상의 프로세서(302) (예를 들면, 멀티프로세서 시스템)을 포함하거나 시스템은 캐시메모리를 포함할 수 있다.

[0034] 322와 같은 화살표는 컴퓨터 시스템(201)의 시스템 버스 아키텍처를 나타낸다. 하지만, 이러한 화살표는 서브 시스템을 연결하도록 기능하는 임의의 상호접속 방식의 예시이다. 예를 들면, 스피커는 포트를 통해 다른 서브 시스템에 접속될 수 있으며, 중앙 프로세서(320)에 내부적으로 직접 접속될 수 있다. 프로세서는 정보를 병렬로 처리할 수 있는 복수개의 프로세서 또는 한 개의 멀티코어 프로세서를 포함할 수 있다. 도2에 도시된 컴퓨터 시스템(201)은 본 발명에서 사용하기에 적합한 컴퓨터 시스템의 일 예에 불과하다. 본 발명에서 사용하기에 적합한 서브시스템의 다른 구성은 당업자에게 이미 명백할 것이다.

[0035] 컴퓨터 소프트웨어 제품은 C, C++, C#, 파스칼, 포트란, Perl, Matlab (from MathWorks, www.mathworks.com),

SAS, SPSS, 자바스크립트, AJAX, 자바, Python, 얼랑(Erlang), and Ruby on Rails 와 같은 다양한 적절한 프로그래밍 언어 중 일부로 기록될 수 있다. 컴퓨터 소프트웨어 제품은 데이터 입력 및 데이터 디스플레이 모듈을 갖는 독립적인 애플리케이션일 수 있다. 혹은, 컴퓨터 소프트웨어 제품은 분산 개체들로서 예시될 수 있는 클래스일 수 있다. 컴퓨터 소프트웨어 제품은 또한 Java Beans (Oracle 사의 제품) 또는 Enterprise Java Beans (Oracle 사의 EJB) 와 같은 컴포넌트 소프트웨어 일 수 있다.

[0036] 본 시스템용 연산 시스템은 Microsoft Windows® 패밀리 (예를 들면, Windows 95, 98, Me, Windows NT, Windows 2000, Windows XP, Windows XP x64 Edition, Windows Vista, Windows 7, Windows 8, Windows 10, Windows CE, Windows Mobile, Windows RT), Symbian OS, Tizen, Linux, HP-UX, UNIX, Sun OS, Solaris, Mac OS X, Apple iOS, Android, Alpha OS, AIX, IRIX32, or IRIX64 중 한 개 일 수 있다. 다른 오퍼레이팅 시스템도 사용될 수 있다. Microsoft Windows는 마이크로소프트 사의 상표이다.

[0037] 또한, 컴퓨터는 네트워크에 접속될 수 있으며, 이 네트워크를 사용하여 다른 컴퓨터와 인터페이스 할 수 있다. 이 네트워크는 인트라넷 또는 인터넷 일 수 있다. 네트워크는 (예를 들면, 구리를 사용하는) 유선 네트워크, 전화 네트워크, 패킷 네트워크, (예를 들면, 광섬유를 사용하는) 광 네트워크, 또는 무선 네트워크 또는 이들의 임의의 조합일 수 있다. 예를 들면, 데이터 및 다른 정보는 컴퓨터와 Wi-Fi (단지 몇가지 예를 들면, IEEE 표준 802.11, 802.11a, 802.11b, 802.11e, 802.11g, 802.11i, 802.11n, 802.11ac, 및 802.11ad), 근거리 무선 통신 (NFC), 무선 주파수 인식 (RFID), 모바일 또는 셀룰러 무선 (예를 들면, 2G, 3G, 4G, 3GPP LTE, WiMAX, LTE, LTE Advanced, Flash-OFDM, HIPERMAN, iBurst, EDGE Evolution, UMTS, UMTS-TDD, 1xRDD, 및 EV-DO) 과 같은 프로토콜을 사용하는 무선 네트워크를 사용하는 본 발명의 시스템의 구성요소 (또는 단계) 사이를 통과할 수 있다. 예를 들면, 컴퓨터로부터의 신호는 적어도 부분적으로, 무선으로 구성요소 또는 다른 컴퓨터로 전송될 수 있다.

[0038] 일 실시예에서, 컴퓨터 워크스테이션 시스템상에서 실행시키는 웹브라우저로, 사용자는 인터넷과 같은 네트워크를 통해 월드와이드웹 (www) 상에서 시스템에 접속한다. 웹브라우저는 웹 페이지 또는 HTML, XML, 텍스트, PDF, 및 포스트스크립트와 같은 다양한 포맷의 다른 컨텐츠를 다운로드 하기 위해 사용되며, 정보를 시스템의 다른 부분으로 업로드하기 위해 사용될 수 있다. 웹브라우저는 URI (uniform resource identifiers) 를 사용하여 웹 상의 리소스와 웹 상의 파일을 전송하는 HTTP (hyper transfer protocol) 을 식별할 수 있다.

[0039] 다른 구현들에서, 사용자는 네이티브 및 넌네이티브 애플리케이션 중 하나 또는 둘 모두를 통해 시스템에 접속한다. 네이티브 애플리케이션은 특정한 컴퓨팅 시스템상에 국부적으로 설치되며, 연산 시스템 또는 컴퓨팅 시스템의 한 개 이상의 하드웨어 장치, 또는 이들의 조합에 특유하다. 이러한 애플리케이션 (때때로 "앱" 이라고 부르기도 함) 은 직접 인터넷 업그레이드 패칭 매커니즘 또는 애플리케이션 스토어 (예를 들면, 애플 아이튠스 및 앱 스토어, 구글 플레이 스토어, 윈도우즈 폰 스토어, 및 블랙베리 앱 월드 스토어)를 통해 (예를 들면, 주기적으로) 업데이트 될 수 있다.

[0040] 시스템은 플랫폼 독립적인 넌네이티브 애플리케이션에서 구동된다. 예를 들면, 클라이언트는 한 개 또는 복수개의 서버와의 네트워크 접속을 사용하여 한 개 이상의 서버로부터 웹 애플리케이션을 통해 시스템에 접속할 수 있고, 웹 브라우저에서 웹 애플리케이션을 로드할 수 있다. 예를 들면, 웹 애플리케이션은 웹 브라우저에 의해 인터넷을 통해 애플리케이션 서버로부터 다운로드 될 수 있다. 넌네이티브 애플리케이션 또한 디스크와 같은 다른 소스로부터 얻을 수 있다.

[0041] 도4는 일반적으로 센서(409) 및 클라우드(412) 사이인 에지 게이트웨이 또는 등가물 상에서 실행되는 에지 컴퓨팅 플랫폼(406)의 블록도를 도시한다. 에지 컴퓨팅 플랫폼은 산업기계 및 다른 산업용 사물용 인터넷 (IoT) 의 관리 및 최적화에 중요한 에지 인텔리전스를 이끌어낼 수 있다. 에지 게이트웨이의 구성요소는 이하의 것을 포함한다. 인제스천(ingestion, 421), 보강(enrichment, 425), 복합 이벤트 처리(CEP) 엔진(429), 애플리케이션(432), 표현 언어(435)를 통한 해석, 및 전송(438). 클라우드는 에지 프로비저닝 및 오케스트레이션 443 및 클라우드 및 에지 분석 및 앱 이식성 446을 포함 할 수 있다.

[0042] 상술한 바와 같이, 에지 컴퓨팅 플랫폼의 특정한 구현은 FogHorn 으로부터 구현된다. FogHorn 은 급속히 부상하고 있는 "에지 인텔리전스" 분야의 리더이다. FogHorn 의 획기적인 솔루션은 제어 시스템 및 물리적인 센서에 가까운 고성능 처리, 분석, 및 이종 애플리케이션을 호스팅함으로써, 폐쇄 루프 장치 최적화를 위한 에지 인텔리전스를 가능하게 한다. 이것은 제조, 석유 및 가스, 전력 및 물, 운송, 광업, 재생가능 에너지, 스마스 시티 등의 산업고객을 위해 현장에서의 빅 데이터 및 실시간 처리를 가지고 온다. FogHorn 기술은 클라우드 컴퓨팅, 고성능 에지 게이트웨이, 및 IoT 시스템 통합에 있어서 세계 유수의 산업 인터넷 혁신기업 및 주요 업체에 의해

받아들여지고 있다.

[0043] Foghorn 은 FogHorn 은 스트림 및 배치(batch) 모드 모두에서 에지 앱에 대한 강화된 IoT 장치 및 센서 데이터 접속을 제공한다. 분석기능을 실행하기 위한 매우 효율적이고 표현력있는 DSL, 풋 프린트가 적은 기계상에서 실행될 수 있는 강력한 소형 분석 엔진, 추가적인 기계 학습을 위한 집계된 데이터를 클라우드로 보내기 위한 게시 기능, 에지 앱을 전개하기 위한 SDK (폴리글롯), 구성, 앱, 및 분석 표현의 에지 전개를 관리하기 위한 관리 콘솔.

[0044] FogHorn 은 산업용 기계로부터 실시간으로, 현장에서 센서 데이터의 스트리밍 처리를 가능하게 하는 효율적이고 확장성이 뛰어난 분석 플랫폼을 제공한다. FogHorn 소프트웨어 스택은 에지 및 클라우드 상에서 실행되는 서비스의 조합니다.

[0045] "에지" 솔루션은 처리되지 않은 데이터를 오프라인 분석을 위해 클라우드 환경에 게시하기 위해 로컬 저장소 리포지토리로의 센서 데이터의 수집을 지원할 수 있다. 하지만, 많은 산업 환경 및 장치는 인터넷 연결성의 부족으로 인해 이러한 데이터를 사용할 수 없게 만든다. 인터넷 연결이 된다고 하더라도, 생성 된 엄청난 양의 데이터가 사용 가능한 대역폭을 초과하거나 클라우드로 전송하는 데 너무 많은 비용이 발생할 수 있다. 또한, 데이터가 클라우드에 업로드되고, 데이터 센터에서 처리되고, 그 결과가 에지로 전송될 때까지, 어떤 조치를 취하기에는 너무 늦을 수 있다.

[0046] FogHorn 솔루션은 분석 엔진이라고도 알려진 고도로 소형화 된 CEP (complex event processing) 엔진과 데이터 수신 센서 스트림의 다수에 대한 규칙을 표현하는 강력하고 표현적인 도메인 특정 언어 (DSL)를 제공함으로써 이 문제를 해결한다. 이들 표현으로부터의 출력이 즉각적으로 사용되어 산업 운영 및 프로세스의 효율과 안전을 실시간으로 개선할 뿐만 아니라 비용이 많이 드는 기계오류 또는 가동중단을 방지할 수 있다.

[0047] FogHorn 플랫폼은 높은 처리량 또는 게이트웨이 환경뿐만 아니라 낮은 풋프린트 환경에서 실행할 수 있는 능력, 수신되는 스트리밍 센서 데이터에 작용할 수 있는 확장성이 뛰어난 고성능 CEP 엔진, 강화된 데이터 접속을 갖는 에지상의 이종 앱 발전 및 전개, 클라우드와 에지를 통한 애플리케이션의 이동성, 고급 기계 학습 (ML) 및 클라이드 및 에지 간의 모델 전송 등을 포함한다. 발군의 FogHorn 은 다른 데이터 전송 프로토콜뿐만 아니라 주된 산업용 데이터 인제스천 프로토콜 (예를 들면, OPC-UA, Modbus, MQTT, DDS 및 기타)를 지원한다. 또한, 사용자는 맞춤형 프로토콜 어댑터를 FogHorn 의 데이터 인제스천 계층으로 쉽게 플러그인 할 수 있다.

[0048] FogHorn 에지 서비스는 IIoT 장치가 상주하는 네트워크의 에지에서 동작한다. 에지 소프트웨어 스택은 센서 및 산업용 장치로부터 고속 데이터 버스로 데이터를 인제스천 한 다음 스트리밍 데이터에 대한 사용자 정의 분석 표현식을 실행하여 통찰을 얻고 장치를 최적화하는 기능을 수행한다. 이러한 분석 표현식은 FogHorn 의 확장성이 뛰어나고 작은 풋프린트 복합 이벤트 처리 (CEP) 엔진에 의해 실행된다.

[0049] FogHorn 에지 서비스는 또한 시간 기반 센서 데이터 쿼리에 대한 로컬 시계열 데이터베이스와 스트림 모드와 배치 모드 양자에서 데이터를 사용할 수 있는 애플리케이션 개발을 위한 다중 언어 SDK를 포함한다. 선택적으로, 이러한 데이터는 또한 고객이 선택한 클라우드 저장 목적지에 게시될 수도 있다.

[0050] FogHorn 플랫폼은 또한 클라우드 또는 온프레미스 (on-premises) 환경에서 실행되어 원격으로 에지를 구성 및 관리하는 서비스를 포함한다. FogHorn 의 클라우드 서비스는 분석 표현식을 개발 및 전개하고, 도커 (www.docker.com) 라고 알려진 애플리케이션을 사용하여 에지에 애플리케이션을 전개하며, 고객의 ID 접속 관리 및 지속성 솔루션으로 서비스 통합을 관리하기위한 관리 UI 를 포함한다. 플랫폼은 또한 클라우드에서 개발된 기계학습 모델을 에지에서 실행될 수 있는 센서 표현식으로 변환 할 수 있다.

[0051] 예를 들면, 애플리케이션은 실시간 데이터 모니터링 및 분석, 예측 유지보수 스케줄링, 및 자동화된 흐름의 방향수정을 적용하여 공동화현상의 발생으로 인해 펌프에 비용이 많이 드는 손상이 발생하는 것을 방지한다. 다른 예는, 전력발생을 최대화 하고, 장비수명을 연장하며, 정확한 에너지 예측을 위한 과거의 분석을 적용하기 위하여 포그혼 에지 인텔리전스 소프트웨어를 사용하는 풍력에너지 관리 시스템이다.

[0052] 도5는 에지 컴퓨팅 플랫폼의 더 상세한 블록도를 도시한다. 이 플랫폼은 데이터 인제스천(512), 데이터 처리(515), 및 데이터 게시(518)의 세 개의 논리적 계층들 또는 섹션들을 갖는다. 데이터 인제스천 구성요소는 센서들 또는 데이터를 생성하는 장치들(523)에 접속되는 에이전트(520)을 포함한다. 에이전트는 각각 프로토콜 서버로부터 한 개 이상의 프로토콜을 통해 센서로부터 데이터를 수집 또는 인제스천한다. 에이전트는 MQTT, OPC UA, Modbus, DDS 와 같이, 프로토콜을 위한 클라이언트들 또는 브로커들일 수 있다. 센서에 의해 제공되고 출력되는 데이터는 일반적으로 이진 데이터 스트림이다. 센서로부터 이들 데이터의 전송 또는 전달은 푸시 또는 풀 방식

으로 행해질 수 있다.

[0053] 푸시는 주어진 트랜젝션을 위한 요청이 송신자(예를 들면, 센서) 에 의해 초기화되는 통신 스타일을 나타낸다. 풀 (또는 get) 은 정보의 전송을 위한 요청이 수신자(예를 들면, 에이전트) 에 의해 초기화되는 통신 스타일을 나타낸다. 또 다른 통신기술은 폴링 (polling) 이며, 수신자나 에이전트가 주기적으로 센서가 보낼 데이터를 가지고 있는지를 문의하거나 체크하는 것을 말한다.

[0054] MQTT (이전의 MQ Telemetry Transport)는 TCP/IP 프로토콜의 최상위에서 사용하기 위한 ISO 표준 게시-구독 기반 "라이트웨이트(lightweight)" 메시징 프로토콜이다. 또 다른 프로토콜로는 고급 메시지 큐잉 프로토콜, IETF 제한 애플리케이션 프로토콜, XMPP, 및 WAMP(Web Application Messaging Protocol)가 있다.

[0055] OPC UA (OPC Unified Architecture) 는 OPC 재단에 의해 개발된 상호운용성을 위한 산업용 M2M 통신 프로토콜이다. 이것은 OPC (Open Platform Communications) 의 후속버전이다.

[0056] 모드버스(Modbus)는 원래 프로그래머블 로직 컨트롤러들(PLCs; Programmable Logic Controllers)과 함께 사용하기 위해 1979년에 Modicon(현재 Schneider Electric)에 의해 공개된 직렬 통신 프로토콜이다. 이것은 단순하고 견고하며, 이후 모든 의도들과 목적들을 위해 표준 통신 프로토콜이 되었다. 이것은 현재 산업용 전자장치를 접속하는 수단으로서 보편적으로 사용된다.

[0057] 데이터 처리(515)는 데이터 인제스천 계층의 에이전트(520)에 연결된 데이터 버스(532)를 포함한다. 데이터 버스는 모든 접속된 구성요소들 간의 데이터 및 제어 메시지를 위한 가장 중요한 백본이다. 구성요소들은 데이터 버스를 통해 흐르는 데이터 및 제어 메시지를 구독한다. 분석엔진(535)은 그러한 중요한 구성요소 중 하나이다. 분석엔진은 표현언어(538)로 발전된 분석표현식에 기초하여 분석데이터의 분석을 수행한다. 데이터 버스에 접속된 다른 구성요소들은 구성서비스(541), 매트릭스 서비스(544), 및 에지 메니저(547)을 포함한다. 또한, 데이터 버스는 원시 이진 데이터를 소모가능한 데이터 포맷 (예를 들면, JSON) 으로 디코딩하며 또한 추가적으로 필요하고 유용한 메타데이터로 데코레이팅 (decorating), 함으로서 센서로부터 유입데이터를 보강하는 "디코더 서비스"를 포함한다. 또한, 보강은 데이터 디코딩, 메타 데이터 데코레이션, 데이터 정규화 등을 포함하지만, 이에 한정되는 것을 아니다.

[0058] JSON (때때로, JavaScript Object Notation 라고 언급됨) 는 사람이 읽을 수 있는 텍스트를 사용하여 속성 값 페이를 구성하는 데이터 오프젝트를 전송하는 개방형 표준 포맷이다. JSON 은 비동기 브라우저 또는 서버 통신 (AJAJ) 또는 둘다에 사용되는 공통 데이터 포맷이다. JSON 의 대체가능한 예로서 AJAX 에 의해 사용되는 XML 이 있다.

[0059] 에지 매니저는 클라우드(412)에 접속하며, 특히 클라우드 매니저(552)에 접속한다. 클라우드 매니저는 또한 클라우드에 존재하는 고객 아이덴티티 및 엑세스 관리 (IAM)(555) 및 사용자 인터페이스 콘솔(558)용 프록시에 접속된다. 또한, 클라우드를 통해 접속가능한 앱(561)이 존재한다. 아이덴티티 및 엑세스 관리는 올바른 개인이 적절한 시기와 적절한 이유로 적절한 리소스에 접속할 수 있게 해주는 보안 및 비지니스 규율이다.

[0060] 데이터 처리(515)내에서 소프트웨어 개발 키트(SDK)(564) 구성요소는 또한 데이터 버스에 접속되어 에지 게이트웨이 상에서 전개될 수 있는 애플리케이션의 생성을 가능하게 한다. 이러한 데이터 개발 키트는 또한 로컬 시계열 데이터베이스에 접속되어 데이터를 인출한다. 애플리케이션은 도커(Docker) 와 같은 컨테이너 기술을 사용하는 것과 같이 컨테이너화 될 수 있다.

[0061] 도커 컨테이너는 코드, 런타임, 시스템 도구들, 및 시스템 라이브러리들과 같이 서버 상에 설치될 수 있는 어떠한 것을 실행하기위해 필요한 모든 것을 포함하는 완전한 파일 시스템 내로 소프트웨어 조각을 랩업(wrap up)한다. 이것은 소프트웨어가 그것이 실행되는 환경에 관계없이 동일한 것을 항상 실행할 수 있도록 보장한다.

[0062] 데이터 퍼블리케이션(518)은 클라우드에서 저장 위치(573)에 접속된 데이터 퍼블리셔(data publisher)(570) 을 포함한다. 또한, 소프트웨어 개발 키트(564)의 애플리케이션(567)은 시계열 데이터베이스(576)에서 데이터에 접속할 수 있다. 시계열 데이터베이스(TSDB)는 시계열 데이터, 시간 (예를 들면, 날짜-시간 또는 날짜 시간 범위)에 의해 색인된 숫자들의 배열을 처리하기 위해 최적화된 소프트웨어 시스템이다. 시계열 데이터베이스는 일반적으로 새로운 정보가 데이터 베이스에 추가되고, 오래된 데이터가 제거되는 롤링 또는 순환적인 버퍼 또는 큐이다. 데이터 퍼블리셔(570)는 또한 데이터 버스에 접속되며 로컬 시계열 데이터 베이스 또는 클라우스 스토리지에 저장될 필요가 있는 데이터를 구독한다.

[0063] 도6은 에지(602)와 클라우드 인프라들 간의 동작흐름을 도시한다. 일부 특정 에지 인프라는 상술하였다. 데이터

는 센서들(606)로부터 수집된다. 이들 센서는, 산업용, 소매, 헬스케어, 또는 의료기기, 또는 전력 또는 통신 애플리케이션, 또는 이들의 조합을 위한 것일 수 있다.

[0064] 에지 인프라는 데이터처리(612), 로컬 시계열 데이터베이스(615), 클라우드 싱크(618), 분석 복합 이벤트 처리 엔진(CEP)(621), 분석 실시간 스트리밍 도메인 특정 언어(DSL)(624)(예를 들면, 포크혼의 Vel 언어), 및 실시간 집계 및 액세스(627)을 포함한다. 플랫폼은 이하에 더 자세하게 후술할 가상 센서(630)를 포함할 수 있다. 가상 센서는 보강된 실시간 데이터 액세스를 제공한다.

[0065] 플랫폼은 소프트웨어 개발 키트 또는 SDK를 사용하여 개발될 수 있는 앱 또는 애플리케이션(1, 2, 3)과 같은 한 개 이상의 앱(633)을 통해 접속가능하다. 앱은 기계학습을 수행할 뿐 아니라, 이종 (예를 들면, 다수의 다른 언어로 개발됨) 이며 복합 이벤트 처리 엔진(621)을 활용할 수 있다. 앱은 에지 플랫폼 개발자 또는 에지 플랫폼의 고객 (파트너라고 언급될 수 있음) 에 의해 제공될 수 있는 앱스토어(637)을 사용하여 배포될 수 있다. 앱스토어를 통해, 사용자는 앱을 다운로드 하거나 다른 사람과 공유할 수 있다. 앱은 기계학습, 원격 모니터링, 예측가능한 보수유지, 동작가능한 인텐리전스, 및 이들의 임의의 조합을 포함하는 분석 및 애플리케이션(639)일 수 있다.

[0066] 이러한 앱의 경우, 에지와 클라우드 사이에 동적 앱 이동성이 존재한다. 예를 들면, 포그혼 소프트웨어 개발 키트를 사용하여 개발된 애플리케이션은 에지상에서 또는 클라우드에서 전개될 수 있으며, 그에 의하여 에지와 클라우드 사이의 앱 이동성을 얻을 수 있다. 앱은 에지의 일부 또는 클라우드의 일부로서 사용될 수 있다. 일 구현에서, 이러한 특징은 앱들이 컨테이너와 되어 있기 때문에 가능한 것이며, 따라서 앱들은 앱들이 실행되는 플랫폼과 독립적으로 작동할 수 있다. 이것은 분석 표현에 대해서도 동일하다.

[0067] 클라우드 또는 사설 데이터 센터(644)에 데이터의 모니터링 또는 저장을 포함하는 통합 경영 및 관리(640)를 위해 허용하는 데이터 앱이 존재한다.

[0068] 물리적인 센서는 아날로그 또는 디지털 측정으로서 그러한 환경의 일부 특성을 측정하는 전자 변환기이다. 아날로그 측정은 일반적으로 아날로그-디지털 변환기를 사용하여 디지털 양으로 변환된다. 센서 데이터는 필요에 따라 측정(폴링)되거나 균일한 속도로 스트림으로서 사용가능하다. 일반적인 센서 사양은 범위, 정확도, 해상도, 드리프트, 안정성, 및 기타 속성이다. 대부분의 측정 시스템 및 애플리케이션은 처리, 운송, 및 저장을 위해 센서 데이터를 직접적으로 사용하거나 전달한다.

[0069] 시스템은 분석 표현 언어를 사용하여 만들어진 소프트웨어 기반 센서인 가상센서라고도 불리는 "프로그램가능한 소프트웨어 정의 센서"를 갖는다. 일 구현에 있어서, 분석 표현 언어는 포그혼의 분석표현 언어이다. 이러한 표현언어는 Vel 이라고 알려져있다. Vel 언어는 실행 지연이 낮고 제약이 적은 풋프린트 환경에서 실시간 스트리밍 분석을 지원하도록 효율적으로 구현된다. 예를 들면, 시스템의 지연은 약 10 밀리초 또는 그 이하일 수 있다.

[0070] 일 구현에 있어서, 프로그램가능한 소프트웨어 정의 센서는 "센서표현언어" 또는 "SXL" 이라고 불리는 선언형 애플리케이션 프로그램 인터페이스(API)로 생성된다. SXL 언어의 특별한 구현은 포그혼의 Vel 이다. Vel 센서는 이러한 구성을 통해 생성된 Vel 센서이며, 물질적인 Vel-센서를 포함하는 다수의 소스에 의해 생성된 처리 데이터로부터 파생된 측정을 제공한다. 이러한 응용에서, Vel 과 SXL 은 서로 교환가능하게 사용된다.

[0071] Vel 센서는 이러한 세 개의 소스 중 하나, 또는 세 개의 소스의 조합으로부터 파생된다.

[0072] 1. 한 개의 센서 데이터

[0073] 1.1. 한 개의 물리적인 센서로부터 파생된 가상 또는 Vel 센서는 임의의 조합의 동적 캘리브레이션, 신호 처리, 수학식, 데이터 압축, 또는 데이터 분석을 사용하여 유입하는 센서 데이터를 변환할 수 있다.

[0074] 2. 복수개의 물리적 센서 데이터

[0075] 2.1. 가상 또는 Vel 센서 또는 다수의 이종 물리 센서로부터의 변환(상술한 방법을 사용) 으로서 파생됨.

[0076] 3. Vel 센서장치의 구현으로 가능한 물리적 센서 데이터 및 가상 센서 데이터의 조합.

[0077] Vel 센서는 도메인에 특화되며 특정한 애플리케이션을 염두에 두고 만들어졌다. Vel 프로그래밍 인터페이스의 특별한 구현은 애플리케이션이 변환 (예를 들면, 수학식) 및 집계를 통한 데이터 분석을 정의하게 한다. Vel 은 일반적으로 프로그래밍 언어를 기초로 하는 한 조의 수학 연산자를 포함한다. Vel 센서는 Vel 구성 또는 프로그램을 실행함으로써 런 타임에 데이터 상에서 동작한다.

[0078] Vel 센서의 생성 Vel 센서는 소프트웨어 장치로서 디자인되어 데이터를 실시간으로 이용가능하게 한다. 이것은 내장된 계산 하드웨어 상에서 실시간으로 Vel 과 함께 개발된 애플리케이션을 실행하여 애플리케이션에 의해 요구되는 속도로 Vel 센서 데이터를 생성하여야 한다. 이 시스템은 이것을 달성하기 위해 고효율의 실행 엔진을 포함한다.

[0079] Vel 센서의 장점은 이하의 것을 포함한다.

[0080] 1. 프로그램가능성 Vel 은 Vel 센서를 프로그램하도록 하여, 데이터 품질, 빈도, 및 정보와 관련된 특정한 애플리케이션 요구사항에 맞도록 데이터를 합성한다. Vel 센서는 물리적 센서 및 기타 (예를 들면, 기존의) Vel 센서로부터 유래된 데이터로 플러그 인 되도록 무선(over-the-air) 소프트웨어 업그래이드로서 널리 배포될 수 있다. 즉, 애플리케이션 개발자는 물리적인 인프라의 레이아웃과 독립적인 비지니스 로직을 효율적으로 실행할 수 있는 디지털 인프라를 생성할 수 있다.

[0081] 2. 유지보수성 또는 투명성 Vel 센서는 애플리케이션과 물리적 센서간의 디지털 추상계층을 생성하며, 이것은 물리적인 센서에 대한 업그레이드 및 서비스로 인한 물리적 인프라의 변화로부터 개발자를 보호한다.

[0082] 3. 효율성 Vel 센서는 물리적 센서로부터의 원시 데이터를 그 안에 포함된 정보의 정확한 표현으로 변환하여 정보 관리의 효율성을 만들어낸다. 이러한 효율성은 애플리케이션에서의 연산, 네트워킹, 및 스토리지 다우스트림과 같은 IT 리소스의 효율적인 활용으로 변화된다.

[0083] 4. 실시간 데이터: Vel 센서는 실제 세계 또는 물리적 센서 데이터 스트림으로부터 연산된 실시간 센서 데이터를 제공한다. 이것은 데이터가 최소한의 시간 지연으로 애플리케이션을 위해 활용될 수 있게 한다.

[0084] 구현. 시스템은 Vel 인터페이스 기반의 Vel 센서의 확장가능한 실시간 구현을 설계하여왔다. Vel 은 Java 언어로 지원되는 연산자를 포함하며 물리적 센서와 그 프로토콜과 잘 결합된다.

[0085] 시스템은 실행된 물리적 센서의 데이터에 대한 연산을 정확하게 표현하기 위한 새로운 방법을 제시한다. 이 선언적인 표현은 물리적 센서상의 구현으로부터 디지털 추상의 정의를 분리시킨다.

[0086] 다양한 타입의 데이터 스트림 셋트와 이러한 스트림 내에서 데이터의 특별한 패턴에 반응하고 조절하기 위한 기능적 수단의 세트가 주어지면, 본 발명은 이러한 기능을 데이터가 스트림에 도착하면 적절하고 효율적으로 호출되도록 이러한 기능을 기술하고 변환하는 기술이다.

[0087] 이러한 종류의 문제를 해결할 필요성은 모든 형태의 데이터플로우 프로그래밍에 공통적으로 나타난다. 이것은 내장된 장치내의 이벤트의 흐름과 같은 아주 소규모의 아키텍처뿐만 아니라 기업 데이터 센터들 내부 또는 기업 데이터 센터들 간의 데이터의 흐름과 같은 아주 대규모의 아키텍처에 적용가능하다.

[0088] 본 발명은 데이터플로우 프로그래밍의 모든 도메인에 적용가능하지만, 일치가 감지될 수 있는 속도와 적용되는 핸들러 함수가 가장 중요하고 실행에 전념할 수 있는 제한된 저장소 및 컴퓨팅 리소스가 존재하는 상황에서 가장 적절하다.

[0089] 일 예. 주어진 정수 스트림으로부터, 우리는 한 개 이상의 0 이 뒤따르는, 한 개 이상의 0이 아닌 값을 일치시키고자 한다. 이러한 패턴이 일치되면, 우리는 0이 아닌 값들의 합을 연산하고 그 결과를 다른 스트림에 기입하고자 한다.

[0090] 우리는 정규식 표기법으로 이러한 문제의 패턴 일치부분을 기입한 후, 산술식으로 합의 연산을 별도로 기입할 수 있다. 이러한 일이 발생하면, 에지 컴퓨팅에서 데이터플로우 애플리케이션에서 사용하도록 디자인된 Vel 프로그래밍 언어는 통일된 표현법으로 전체적인 변환을 기입할 수 있게 한다.

표 1

[0091]
```
stream("output") =
( a:{!= 0} .. {>0}, :0 .. {>0} -> sum(a) )
from stream("input")
```

[0092] 이 기술은 상기의 함수 매개변수화를 상태 머신으로 번역한다. 그런다음, 상태 머신을 기반으로 결정성있는 유한 자동화로서의 일치를 구현하고 결과 매치들을 합계 표현식에 공급한다. 이러한 플로우는 도7에 도시된다. 이것은 상태 0 (705), 상태 1 (710), "리스트 a" 로부터 블록(715) 및 "푸시합계(a)" 블록(720) 이다.

[0093] 이러한 문제는 각 핸들러 함수에 대한 매칭함수를 생성함으로서 해결될 수 있다. 매칭함수는 입력으로서 스트림으로부터의 데이터 윈도우를 입력으로서 받아들이고, 매치에 대한 "참" 및 비매치에 대해 "거짓"을 반환한다. 데이터가 윈도우를 통해 흐름에 따라, 매칭 함수는 매치가 발견될 때까지 반복적으로 적용되어야 한다. 일단 매치가 발견되면, 핸들러 함수가 적용된다.

[0094] 이러한 해결책은 핸들러 함수가 데이터베이스 쿼리에 대하여 사용된 것과 유사한 방식으로 지정되었기 때문에 생기는 것이다. SQL과 유사한 WHERE 절은 매칭 조건을 기술하는 Boolean 표현식을 제공하며 매칭 함수는이 표현식을 직접 컴파일한다.

[0095] 새로운 데이터가 스트림 버퍼로 흐를 때, 개별적인 매칭 함수가 개별적으로 평가되어야한다. 매치는 각 함수에 대해 독립적으로 판단된다.

[0096] 매치를 수행하기 위해 상태 머신을 사용하는 것이 복수개의 임의의 Boolean 연산식을 반복적으로 적용하는 것보다 더 효율적이다.

[0097] 본 발명은 함수의 매개변수를 선언하는 패턴 기술언어로부터 상태 머신을 도출한다. 도출된 상태 머신은 종래 Boolean 연산식 매칭 함수보다 데이터 스트림에서 더 효율적으로 매치를 검출한다.

[0098] 도출된 상태 머신은 또한 데이터 스트림에서 검출된 매치를 위한 한 세트의 핸들러 함수를 구현할 수 있다. 복수개의 매칭 및 대응하는 핸들러함수는 임의의 핸들러 함수에 대한 매치를 효율적으로 인식하는 한 개의 상태 머신으로 결합되거나 감소될 수 있다.

[0099] 도출된 상태 머신은 또한 증가되어 상태 머신에 의해 인식된 시퀀스를 변경시키지 않고 추가적인 노드를 통한 자유 (엡실론) 변환을 포함할 수 있다.

[0100] 그러한 추가적인 노드를 통한 변환은 데이터에 대한 다양한 액션을 트리거할 수 있다. 예를 들면, 결정성 유한 오토마톤(DFA: Deterministic Finite Automaton) 또는 스택 머신의 시프트 버퍼로의 데이터의 수집을 대기영역으로 트리거할 수 있다. 이들 데이터는 나중에 함수 응용에 대한 인수의 기본을 형성할 수 있다.

[0101] 이 애플리케이션은 DFA라는 용어를 사용하지만, 이러한 오토마톤 또는 유닛은 스택머신이라고 부를 수도 있다. 엄격하게 말하면, 결정성 유한 오토마톤은 공간에서의 유한한 성능을 암시한다. 하지만, 본 특허에서 오토마톤은 유한할 필요가 없으며, 비유한하고 간단할 수 있다. 그러므로, 본 특허에서 기술된 바와 같은 DFA 는 유한하지 않다.

[0102] 그러한 추가적인 노드를 통한 변환은 또한 이전 노드에서 함수 애플리케이션 인수로서 캡처된 데이터를 사용하여 핸들러 함수의 호출을 트리거할 수 있다.

[0103] 정규 표현식 및 값 표현식의 태양을 결합하는 스크립트로부터의 변환은 패턴을 효율적으로 매치시키고 값을 연산할 수 있는 확장된 상태 머신 또는 DFA를 야기시킨다.

[0104] 그 결과로서 결합된 매칭 또는 연산 알고리즘은 패턴 매칭 또는 값 연산의 개별적인 조직보다 더 효율적이다.

[0105] 어휘 소스 (lexical source) 로부터 DFA 또는 상태 머신을 구성하고, 비결정적 유한 오토마톤(NFA: Nondeterministic Finite Automaton) 로 시작한 다음 그것을 최소 DFA로 줄이는 방법. DFA의 목적은 입력데이터의 시리즈 내에서 패턴을 인식하는 것이다. 본 논의를 위해, 우리는 상태 머신 토큰을 통해 흐르는 데이터 및 토큰의 언어로서 DFA 에 의해 인식된 특정한 패턴을 호출할 것이다.

[0106] 도8에서 NFA 의 부분을 고려한다. 이 부분은 DFA 일 수도 있지만, 본 예의 목적에 의하면 중요하지는 않다. 토큰 알파 수신 하에 상태A(805)로부터 상태B(810) 으로 변환한다.

[0107] 도9에 도시된 바와 같이, 엡실론 변환(920)으로 추가적인 노드를 추가함으로서 NFA 를 증가시킬 수 있다. 엡실론 에지는 언제라도, 자유롭게, 그 입력 상태에 관계없이 그 상태대로 뒤따를 수 있다.

[0108] 한 개 이상의 엡실론 에지의 존재는 상태기계를 비결정성으로 만들지만, 엡실론 에지는 이 수단에 의해 테이블로 구동되는 방법에 의해 효율적으로 구현될 수 있는 동등한 DFA로 감소된 NFA로, 알고리즘에 의해 제거될 수 있다. 그러므로, 우리는 효율적인 구현을 위한 정책을 여전히 남겨둔 채 이러한 추가적인 엡실로 변환을 소개할 수 있다.

[0109] 도9의 상태 머신은 토큰 알파(925)의 수신시에 상태A(905)로부터 상태X(915)로 변환될 것이며, 임의로 상태X로부터 상태 B(910) 으로 임의로 진행할 수 있다. 알파의 추진동력은 도8의 단순한 기계가 하는 것과 같이, 여전

히 상태A로부터 상태B로 변환되며, 이러한 변환을 성취하기 위해 추가적인 입력은 필요하지 않다. 그러므로, 도9의 NFA 는 도8에서의 동일한 언어를 번역하는 것으로 보일 수 있다. 그것은 그렇게 하기 위해, 단순히 상태X를 통해 추가적인 상태변환을 한다.

[0110] 추가적인 상태는 우리가 부작용의 성능을 연관시킬 수 있다는 점에서 유용하다. 이러한 부작용이 상태 머신의 정의 및 상태 머신을 통해 흐르는 데이터를 변경시키는 한, 추가적인 노드는 언어의 인식에 영향을 미치지 못하지만, 부작용은 추가적인 작업을 할 수 있다.

[0111] 데이터플로우 반응 구현에 있어서, 추가적인 작업은 데이터에 대한, 또는 데이터를 사용하는 임의의 수의 유용한 동작을 포함할 수 있다. 한 개의 예시적인 구현에 있어서, 작업은 이하의 것을 포함한다.

[0112] 1. 노드를 통해 흐르고 그것의 사본을 외부 수집기로 보내는 데이터를 검사.

[0113] 2. 데이터가 노드를 통해 흐르는 경우 데이터로의 변환을 적용하고 변환된 데이터를 임시버퍼에 저장, 또는

[0114] 3. 수집된 데이터를 임시버퍼로부터 추가적인 변환으로 플러시하고 그 결과를 다른 DFA 또는 스택머신으로 푸시.

[0115] 일 예로서, 소스 조각을 고려한다.

표 2

[0116]

```
( a:{!= 0} .. {>0}, :0 .. {>0} -> sum(a) )
```

[0117] 이 조각은 두 개의 용어로 이루어지는 패턴을 기술한다. (1) a 라고 불리는, 한 개 이상의 0이 아닌 값의 반복과 일치하는 첫번째 용어 (2) 한 개 이상의 0의 반복과 일치하는, 이름이 주어지지 않은 두번째 용어.

[0118] 이러한 것을 반응에 대한 기본으로서 사용하기를 바란다고 가정한다. 우리는 호출된 소스로부터 값을 독취하며, 우리가 입력 중에서 단편의 패턴을 인식할 경우, 단면의 오른쪽을 평가하고, 최종 호출로 결과를 푸쉬함으로서 반응할 수 있다.

[0119] 예를 들면, 값[101, 202, 303, 0, 0] 으로 이루어진다면, 우리는 첫번째 세 개의 값을 a와 결합하고 마지막 두 개의 값을 익명의 제2 용어와 결합함으로서 패턴을 일치시킬 수 있다. 그러고 나서, 우리는 a에 결합된 값의 리스트 [101,202, 303] 에 합함수를 적용하고 606으로 돌아감으로서 오른쪽을 평가한다. 그 후, 우리는 606을 바깥쪽으로 푸시한다.

[0120] 본 발명에 따라 본 실시예에서와 같은 기능적 패턴의 번환은 컴퓨터로 실행되는 변환 프로그램을 통해 구현될 수 있다. 프로그램은 두 개의 상이한 형태의 번역을 수행할 수 있다. 기능 지향 부분 "sum(a)" 를 연산을 실행할 실행가능 명령문들의 블록으로 번역하고, 패턴 지향 부분 "a:{!= 0} .. {>0}, :0 .. {>0}" 을 패턴을 인식하고, 인수를 캡쳐하고, 함수를 호출할 DFA 또는 스택머신으로 변환한다. 이들을 첫번째 작업함수 변환과 두번째 작업함수 변환이라고 부르기로 한다.

[0121] 함수 번역은 컴파일러와 해석기의 작성을 전문으로 하는 컴퓨터 프로그래머들은 잘 이해할 수 있다. 패턴번역, 함수번역과 패턴번역의 맞춤, 및 패턴 인식과 함수 발송의 후속적인 자동화가 본 발명의 주제이다.

[0122] 소스 텍스트의 구문내용을 기술하는 추상구문 트리 (AST: abstract syntax tree)의 잎을 형성하도록, 함수번역은 소스텍스트의 수신, 텍스트를 토큰으로 분해 후, 문법따라 안내하여 토큰을 정렬하는 것으로 이루어진다. 그 후, 추상구문트리는 소스에 의해 기술될 함수를 평가하기 위하여 요구되는 명령어 블록을 최종적으로 생성하는 일련의 알고리즘에 의해 트리버스된다.

[0123] 패턴번역은 상술한 구문해석에 의해 형성된 추상구문트리로 시작한다. 추상구문트리는 패턴선언의 뿌리를 형성하는 한 개 이상의 노드를 포함할 수 있다. 예를 들면, 상술한 패턴은 두 개의 자식을 갖는 한 개의 루트노드로 이루어질 수 있으며, 도10의 좌하단에 도시된 바와 같이, 각각의 자식은 패턴의 한 용어를 기술한다. 도10에, 반응루트 노드(1005), 패턴루트노드(1010), sum(a) 노드(1015), 노드(1020), 및 <노네임>노드(10) 이 존재한다.

[0124] 일치하는 예제와 그것과 일치시키기 위한 반복을 지정하는 패턴 용어 노드가 정규 표현식의 항과 동일한 정보를 전달한다는 것을 인식해야 한다. 또한, 함께 및 순서대로 취해지는 자식노드의 시퀀스는 정규 표현식 항의 선형결합과 동일한 정보를 지정한다. 정규 표현식 또는 정규 표현식 항의 선형결합은 NFA로 번역된 항일 수 있다.

본 발명자들은 동일한 알고리즘이 본 발명에서 사용될 수 있으며, 패턴 항은 정규 표현식 항을 나타낸다는 것을 발견하였다.

[0125] 일단 기본 NFA 가 그렇게 형성되면, 패턴 용어로 동작이 요구되는 위치에 추가적인 부작용 유발 상태를 주입 할 수 있고, 수용 상태 후에 반응 기능을 호출 할 수있다.

[0126] 예제를 계속하기 위하여, 용어 a는 그것과 일치되는 값의 리스트를 수집하고 그것을 인수로서 반응 함수에 전달할 것을 요구한다. 즉, 도9에 도시된 변환을 용어 a 의 결과인 NFA 상태에 적용하고 일치하는 용어를 수집하는 작업을 행하기 위한 새로운 상태를 사용한다. 그 후, 변환을 다시 NFA 가 수용된 상태로 적용하고, 수집된 값들을 사용하여 반응 함수를 호출하고, 반응 소비자에게 결과를 푸시하고, 수집 버퍼를 비운다. 이러한 강화된 NFA 가 DFA 및 축소된 상태로 변환 된 후, 도7에 도시된 기계로 남겨진다.

[0127] 이러한 단계는 상태작업 테이블을 사용하여 상태 머신 엔진을 구동하는 알고리즘과 마찬가지로 NFA 를 DFA로 변혼하고 상태를 DFA로 축소시키기 위해 사용되며, DFA 를 상태 작업 테이블로서 변경는데 사용된다.

[0128] 본 발명의 기술에 의해 생성된 NFA 는 테이블로 변환 및 렌더링된다. 하지만, 이러한 결과 테이블은 각 상태를 통과할때 싱행될 부작용 람다로 이루어진 추가 컬럼을 포함한다. 그러한 상태 동작 람다 테이블을 사용하는 자동화 엔진은 다른 기술과 달리 전이를 실행할 때마다 추가 람다를 실행한다.

[0129] 데이터플로우 연산 환경을 사용하기 위한 반응 함수를 기술하고 번역하는 방법은 (i) 반응함수를 식별하는 단계, (ii) 함수에 입력을 제공하는 매개변수의 패턴을 식별하는 단계, (iii) 함수를 거친 인수에 기반하여 평가될 연산식을 식별하는 단계, (iv) 매개변수의 패턴을 패턴과 일치하는 입력의 시퀀스를 인식할 수 있는 상태 머신으로 번역하는 단계, (v) 입력 데이터를 수집하고 변환하는 작업을 행하는 추가적인 상태로 상태 머신을 증강하여 함수에 대한 인수로서 사용하도록 준비하는 단계, 및 (vi) 간단한 소프트웨어 또는 하드웨어로 자동화할 수 있는 상태 동작-효과 테이블로 상태 머신을 감축시키는 단계를 포함한다.

[0130] 인수로서 함수의 세트 및 값들의 시퀀스가 주어지면, 본 발명은 인수가 일치하는 함수에 실행을 디스패치하거나 인수가 함수 중 어느 것과도 일치하지 않는다고 판단하는 방법이다. 본 방법은 값 표현식, 유형 표현식, 및 정규 표현식을 결합함으로서 유형 시스템에서 표현가능한 임의의 값의 시퀀스를 모호함이 없이 일치시킬 수 있는 신규한 방법이다.

[0131] 이러한 형태의 문제를 해결하기 위한 필요성이, 번역기, 해석기, 및 컴퍼일러의 발전 단계에서 제기되고 있으며, 이것은 다형성 디스패치의 개념과 밀접한 관련이 있다. 만약 누군가가 한 개의 오브젝트(튜플)을 구성하기 위하여 시퀀스의 임의의 접두사(prefix)를 형성하는 요소들을 고려한다면, 올바른 함수로 디스패치하는 작업이 튜플의 클래스의 방법의 다형성 디스패치와 동등한 것으로 생각될 수 있다.

[0132] 본 발명은 이러한 종류의 다형성 디스패치가 요구되는 임의의 상황에 적용가능하다. 이것은 프로그램의 외부로부터 발생하는 데이터 스트림에 응답해야 하는 모든 방식의 이벤트로 구동되거나 반응하는 프로그램을 포함한다. 본 발명은 특히, 에지 또는 포그 컴퓨팅 또는 네트워킹 환경에서 종종 발생하는 것과 같은 복수의 데이터 스트림의 실시간 처리와 관련한 애플리케이션에 유용할 것이다.

[0133] 정규 표현식은 특정한 패턴에 일치되는 문자열을 검출하기 위하여 공통적으로 사용된다. 다수의 정규 표현식 언어와 이것에 기초하여 효율적인 매칭 엔진을 실행하는 가장 밀접하게 관련된 많은 수의 도구가 존재한다. 이것은 일반적으로 문자의 시퀀스를 일치시키는 것으로 한정된다.

[0134] 문자열 이외의 도메인 상에서 동작하는 다른 패턴 기반의 표기법도 존재한다. 일 예로는 XML 문서에 패턴을 기술하는 XPATH 가 있다. 이러한 표기법은 종종 정규 표현식보다 덜 완전하고 덜 강력하며 특정한 도메인에 맞추어져 있다.

[0135] 일부 프로그래밍 언어는 형태 기반의 패턴 매칭 시스템을 통해 런타임 다형성 디스패치를 구현한다. 함수의 복수의 과부하가 정의 되며, 각각은 형태와 값이 상이한 패턴을 취한다. 디스패치는 함수 매개변수의 패턴에 대한 인수의 형태와 값을 매칭시킴으로서 실행시간에 해결된다. 하스켈(haskell) 은 그러한 프로그래밍 언어중 한가지이다.

[0136] 언어-명세언어는 일련의 생산 규칙으로서 문맥자유 문법(context-free grammer)를 기술한다. 이러한 규칙은 언어의 구문을 구성한다. 컴파일러-컴파일러는 이러한 규칙을 언어의 예들을 인식할 수 있는 테이블로 구동되는 결정성 있는 유한 상태 머신으로 번역한다. 비손(bison) 은 그러한 언어-명세 언어의 일 예 및 그와 연관된 컴

파일러-컴파일러이다.

[0137] 정규 표현식과 같은 문법으로 구동되는 패턴 일치 시스템은 결정성 있는 유한 자동화(DFA) 또는 상태 머신과 같은 단순한 머신으로 표현될 수 있기 때문에 효율적인 실행이라는 장점을 가지지만, 완전한 형태의 시스템의 넓은 모델링 능력은 부족하다. 하스켈에서 사용되는 것과 같은 형태로 구동되는 패턴 매칭 시스템은 모델링 능력이 더 풍부하지만, 합리적으로 효율적인 구현을 위해 표현가능한 것을 희생하지만, DFA에 기초한 고속 매칭 시스템만큼 효율적이지는 않다.

[0138] 본 발명은 그 유형 중에서 표현가능한 모든 상태에 대해 일치할 수 있지만 여전히 상태 기계만큼 효율적으로 구형될 수 있는 형태 기반 매칭 시스템을 취급한다. 유형 및 상태의 일반화된 패턴은 패턴의 일 예들을 효율적으로 인식할 테이블로 구동되는 상태 머신으로 번역된다.

[0139] 이러한 패턴에 기초하여 함수 매개변수를 정의하는 것은 함수가 데이터의 임의의 패턴을 정확하게 매칭시키도록 하며, 매칭시에 매칭 데이터 요소들 중에서 그 인수를 바인딩한다. 함수의 조합에 대한 매칭 패턴을 기술하는 상태 머신은 멤버 함수의 상태 머신을 통합한 후 그 결과를 최소수의 상태로 축소함으로서 형성된다. 과부하들 간의 명확성 또는 전체적인 불일치의 검출은 시퀀스에서 가능한한 조기에 발생하여 함수 애플리케이션의 해상도를 가속화한다. 매칭은 가능한 한 많은 입력을 받아드릴 수 있는 함수의 "greedy" 버전을 생성하여 시퀀스 내에서 가능한한 늦게까지 지연될 수 있다.

[0140] 방법은 값 표현식, 유형 표현식, 및 정규 표현식을 결합함으로서 유형 시스템에서 표현가능한 임의의 값의 시퀀스를 모호함이 없이 일치시킨다. 이 방법은 함수 애플리케이션을 해소하고 최소수의 판단으로 정정 과부하를 디스패치한다. 문법적 하위 구성요소를 재귀적으로 인식하고 변환함수를 적용함으로서 특정 언어를 인식하도록 하여, 이 방법은 과부하 함수 애플리케이션이 문맥 자유 그래머와 동일한 작업을 수행하게 한다.

[0141] 본 방법은 복수의 상이한 유형을 포함하는 유형시스템과 결합하여 적용가능하다. 예를 들면, (1) 정수, 실수, 및 문자열과 같은 한 조의 기초적인 단형성 유형. (2) 한 조의 다형형 유형 및 그들의 생성자(constructor), 특히 우리가 간단히 논의하여야 하는 임의의 속성을 갖는 다형성 세트 유형. (3) 합산 유형 (sum type) (4) 기록형태의 곱 유형 (product type). (5) 이 필드의 반복을 포함하는 튜플의 일반화 인, 패턴 형태의 곱 유형. (6) 패턴 유형을 임의의 유형에 매핑하는 람다유형. 및 (7) 람다의 리스트를 구성하는 폴리람다 유형 (poly-lamda type).

[0142] 세트는 요소의 한 개 이상의 범위를 구성하는 다형성 유형이다. 예를 들면, 한 세트의 정수가 한 세트의 문자열로부터부터의 구별되는 형식이 되도록, 세트 유형은 포함하는 요소의 유형에 대해 매개변수화된다. 세트 유형은 이 내용에 대한 한계에 의해 특징지어질 수 있다. 특히, 세트 유형은 유한 또는 무한하도록, 또는 좌측 또는 우측으로 폐쇄 또는 개방되도록, 또는 이들의 임의의 조합으로 제한될 수 있다. 정수 세트의 다음 예를 다음과 같이 고려한다.

표 3

[0143] 테이블 A

표기법	길이	폐쇄성	의미
[1]	1	좌측 및 우측에서 폐쇄됨	한 개의 정수 1로 구성된 세트.
[1, 2, 3]	3	좌측 및 우측에서 폐쇄됨	1, 2, 및 3 세 개의 정수로 이루어진 세트.
[5000 .. 6000]	1001	좌측 및 우측에서 폐쇄됨	5000 부터 6000 까지의 정수를 포함
[10 ..]	무한	좌측 폐쇄, 우측 개방	10 이상의 모든 정수
[.. 10]	무한	좌측 개방, 우측 폐쇄	10 이하의 모든 정수
[> 5]	무한	좌측 폐쇄, 우측 개방	5보다 큰 모든 정수. [6..] 과 같음.
[>= 5]	무한	좌측 폐쇄, 우측 개방	5이상의 모든 정수. [5..] 와 같음.
[< 5]	무한	좌측 개방, 우측 폐쇄	5 미만의 모든 정수.[.. 4] 와 같음.
[<= 5]	무한	좌측 개방, 우측 폐쇄	5 이하의 모든 정수.[.. 5] 와 같음.
[!= 5]	무한	좌측 개방, 우측 폐쇄	5 이외의 모든 정수.
[>= 1] and[<= 3]	3	좌측 및 우측에서 폐쇄됨	1, 2, 및 3의 세 개의 정수로 이루어진 세트. [1, 2, 3] or [1 .. 3] 과 같음.
[<= -10] or [>= 10]	무한	좌측 개방, 우측 폐쇄	10 이상의 절대값을 갖는 모든 정수
not [1 .. 3]	무한	좌측 개방, 우측 폐쇄	1, 2, 및 3 을 제외한 모든 정수.

[0144] 요소가 정수형태이고 정수는 명확하게 셀 수 있으므로 [>= 1] 과 [> 0] 사이에는 구분이 없다. 요소가 실수 또는 문자열과 같이 셀수 없는 유형인 경우라면, 명백한 포함 또는 종점의 포함이 필요하게 된다. 예를 들면, 세트 [>= "cat"] 는 문자열 "cat" 및 "cat" 이후에 사전편찬식으로 정렬되는 모든 문자열로 이루어질 수 있다.

[0145] 유형으로서 세트의 일 예를 사용할 수 있다. 그러한 유형의 일 예는 세트의 구성요소일 수 있다. 예를 들면, 유형으로서 사용되는 세트 [>0] 은 값과 같은 양의 정수만을 허용할 수있다. 사실상, 이런 방식으로 모든 유형을 생각할 수 있다. 예를 들면, 단형성 정수유형이 모든 정수의 세트로 이루어진 세트 유형으로 고려될 수 있다.

[0146] 합 유형은 다른 유형의 단순한 합집합이다. 예를 들면, 유형 정수(int) 또는 스트링은 두 개의 구성 유형의 합이다. 합 유형의 임의의 구성 유형의 임의의 예는 합 유형의 일 예일 수 있다. 예를 들면, 이것은 정수 또는 문자열일 수 있는, 값들의 리스트인 유형 리스트 (정수 또는 문자열)을 기술하도록 할 수 있다. 합집합의 합집합은 평평해지므로, 유형 표현 (정수 또는 문자열) 또는 (정수 또는 실수) 은 정수, 실수, 또는 문자열과 동일하다. 합집합 내에서 유형의 순서는 중요하지 않지만, 표준화를 위해서, 그 구성요소가 알파벳 순서가 되도록 모든 합집합 유형을 나타낸다.

[0147] 레코드 유형은 명명된 필드를 사용하고 각 필드를 유형과 연관시킨다. 예를 들면, {생일: 날짜; 이름: 문자열; 성: 문자열}. 레코드 유형은 항상 유한한 수의 필드를 가지며, 각 필드는 유형 내에서 고유의 이름을 갖는다. 필드의 순서는 중요하지 않다. 즉, {x: int; y: int} 는 {y: int; x: int} 와 같다. 하지만, 합집합에 대해 행했던 것처럼, 알파벳 순서로 구성요소를 갖는 레코드 유형을 나타낼 수 있다.

[0148] 레코드의 이러한 유형은 레코드 자체라는 것에 주목하여야 한다. 값 {x: 3; y: 4} 은 유형 {x: int; y: int} 을 갖는다.

[0149] 패턴 유형은 유형의 시퀀스로서 정의된 튜플과 유사하지만, 튜플은 암시적으로 각 요소가 정확하게 한번 만 나타나는 것으로 가정하고, 패턴은 각 요소가 반복성을 갖는 것을 허용한다. 반복성은 정수 세트로서 주어진다. 예를 들면, 패턴 <a: int # [1 .. 3]; b: string # [1 .. 3]> 은 한 개 내지 세 개의 문자열이 뒤따르는 한 개 또는 세 개의 정수와 매치된다.

[0150] 람다의 매개변수로서 사용될 경우, 패턴의 필드는 람다의 평가 내에서 바인딩되는 인수를 발생시킨다. 예를 들면, 이전 단락에서 주어진 패턴에 일치시킨 후, 두 개의 로컬 식별자 a 와 b 를 범위내에서 가질 수 있다. A 의 값은 한 개 내지 세 개의 정수의 리스트일 수 있고, b의 값은 한 개 또는 세 개의 문자열의 리스트일 수 있다.

[0151] 패턴 내의 한 개 이상의 필드가 이름을 갖지 않는 것도 또한 유효하다. 이름을 갖지 않는 필드도 매칭되지만, 값이 없는 것은 인수로서 바인딩된다. 예를 들면, <a: int # [1 .. 3]; string # [1 .. 3] > 에 매칭이 되면, 이전처럼, 한 개 이상의 문자열이 뒤따르는 한 개 이상의 정수로 매칭하며, 정수들은 a 라고 불리는 리스트로서 바인딩하지만, 문자열은 바인딩하지 않는다.

[0152] 패턴의 길이는 무한할 수 있다. 예를 들면, 패턴은 <a: int # [1 ..]> 상한이 없는 한 개 이상의 정수와 매칭될 것이다. 이러한 것도 유효하며, 끝이 없는 입력 스트림을 처리하기 위하여 사용된다면, 무한한 패턴이, 값을 수집하는 것을 중단해야 할 때를 나타내는 시간 간격과 같이, 다른 트리거의 일부와 페어링되어야 한다.

[0153] 일반적으로 패턴은 그것과 매칭되는 데이터를 소모하지만, 해당 데이터의 서브셋만을 소모하거나 전혀 사용하지 않을 수 있다. 패턴은 "peek point" 라고 불리는 마크를 포함할 수 있으며, 이 시점을 넘어서 데이터를 매칭시키고, 인수를 바인딩하지만, 입력 스트림으로부터 소비되는 것은 아니다. 예를 들면, 패턴 <a: int; b: int; peek; c: int> 세 개의 정수와 세 개의 로컬 식별자를 매칭시키지만, 입력으로부터 오직 두 개의 정수만을 소비한다.

[0154] 필드를 갖지 않는 레코드 또는 필드를 갖지 않는 패턴을 갖는 것은 유효하다. 이들 두 경우는 곱 유형을 나타내기 때문에, 서로 의미있게 구별할 수 없다. 구문적으로, 이러한 컨셉을 키워드 보이드(void) 라고 지정한다. 보이드는 고유의 값이며 그 자체의 유형이다. 합집합에서 사용되면, 보이드는 만약 존재할 경우에는 int 이지만, 존재하지 않을 수도 있는 값을 의미하는 int 또는 void과 같은 선택적인 유형의 개념을 발생시킨다.

[0155] 우리의 목적을 위해, 유형 매칭은 구조적이지만, 지명되지 않는다. 유형은 이름은 가지지 않고, 단지 기술되기만 한다. 동일하게 기술되는 두 개의 유형은 동일한 유형이다. 기술이 다른 유형의 기술의 서브세트인 유형은 그 유형의 일반화이다. 예를 들면, 유형 {x: int; y: int} 및 {x: int; y: int; z: int} 을 고려해보자. 두 개의 필드 x 와 y를 갖는 유형이 세 개의 필드 x, y, z 를 갖는 유형의 서브세트이면, 전자는 후자의 일반화라고

고려될 수 있다. 이것은 또한 패턴에 대해서는 참이다. 다른 것의 접두사인 패턴 또한 그것의 일반화이다.

[0156] 람다 유형은 입력 패턴을 다른 유형에 매핑한다. 예를 들면, 세 개의 정수 중 한 개를 취하고 정수를 되돌려 놓는 함수의 유형인 <int # [1 .. 3]> → int 이다. 폴리 람다 유형은 람다 유형의 리스트로 이루어진다. 람다들의 순서는 여기에서는 중요하다. 폴리 람다 애플리케이션은 해결하는 경우, 매칭되는 구성요소인 람다의 처음으로 디스패치 할 것이다.

[0157] 이런 방식으로 정의된, 폴리 람다를 디스패치하기 위하여 요구되는 패턴 매칭은 DFA 로 축소될 수 있다. 방법을 설명하기 위해, 비교를 위한 기초로서 상태 머신 구성의 방법을 사용하고 필요에 따라 그것을 증강시킬 것이다. 설명은 먼저 비결정적 유한 오토마톤 (NFA) 을 구성하고 나서, 그것은 DFA, 로 축소시키는 것과 관련되지만, 실제로, 이것은 일반적으로 단일의 단계에서 행해질 수 있다.

[0158] 앞서 논의한 바와 같이, 이 애플리케이션은 DFA라는 용어를 사용하지만, 이러한 오토마톤 또는 단위는 스택머신이라고 부를 수도 있다. 엄격하게 말하면, 결정성 유한 오토마톤은 공간에서의 유한한 성능을 암시한다. 하지만, 본 특허에서 오토마톤은 유한할 필요가 없으며, 비유한하고 간단할 수 있다. 그러므로, 본 특허에서 기술된 바와 같은 DFA 는 유한하지 않다.

[0159] 우선, 개별적인 람다 패턴인, 폴리 람다의 구성요소는 교대의 요소로서 간주되어야 한다. 정규 표현식을 번역함에 있어서, 구문 a|b (a 또는 B) 는 교대의 표현이다: a 를 1105에 매칭시키거나 b 를 1110 에 매칭시킨다. 이 경우에, a AND b 는 각각 람다 패턴이다. 도11에 교대에 대한 서브그래프를 구성하였다.

[0160] 개별패턴의 필드를 결합으로 표현하였다. 정규 표현식을 번역함에 있어서, 구문 ab 1210 은 결합을 나타낸다. a 를 1205에 매칭시키고 나서 b를 1215에 매칭시킨다. 이 경우에, a AND b 는 패턴의 각 필드이다. 도12에 결합에 대한 서브그래프를 구성하였다.

[0161] 필드의 반복 팩터는 종래에 a+ or a* 또는 a{n:m} 로 기재되는, 정규 표현식의 폐쇄(closure) 와 동일하다. 다시 한번, 도 13에 도시된 바와 같은 구조를 갖는 폐쇄를 도시한다. 이 경우에, 반복 세트의 값에 기초하여 서브그래프 내의 일부 변수가 필요할 것이다. 예를 들면, 노드 i (1305) 로부터 노드 j (1310) 순방향 엡실론은 세트가 0을 포함하지 않는 경우에만 포함될 수 있다. 이러한 변형들은 대체로 명백하며, 여기에서 제시되는 동일한 기본적인 아이디어와 함께 계속된다.

[0162] 중간 NFA 가 완료된 후, 그것은 DFA 로 축소시키고 나서 최소 DFA 에 도달할 때까지 DFA 의 상태를 감소시킨다. 그 후, 자동화 상태 머신에서 채용되는 일반적인 종류의 소프트웨어 또는 하드웨어에 의해, 자동화에 적합한 상태-동작 테이블로서 DFA 를 렌더링한다. 이러한 테이블의 수용 상태는 폴리 람다로의 진입지점을 표시하고 중간 상태는 인수를 바인딩하기 위하여 사용된 데이터의 수집을 제공한다.

[0163] DFA가 이렇게 자동화되고 입력 스트림을 제공할 경우, 스트림 및 디스패치로부터 입력의 접두사와 올바른 과부하를 매칭시켜서 그것을 조절하고 계산된 결과를 산출한다. 이러한 과정이 반복되는 것이 허용된다면, 결과는 입력 스트림으로부터 매치당 하나씩 산축된 결과의 시퀀스이다. 이것은 데이터 스트림에서 검출된 다양한 유형의 인수의 대응 패턴에 의해 트리거되는 다형성 함수에 의해 입력 데이터 스트림의 효율적인 실시간 처리를 제공한다.

[0164] 값과 유형 식별자의 혼합을 포함하는 복수의 종류의 함수 인자를 포함하는 데이터 스트림에 대응하는 다형성 함수의 실행을 디스패칭하는 방법은 (i) 실행될 다형성 함수로서, 상기 다형성 함수는 상이한 종류의 인수의 패턴과 각각 관련된 복수의 과부하를 갖는 다형성 함수를 식별하는 단계; (ii) 과부하의 인수 패턴을 매칭시킴으로서 입력 스트림으로부터 바인딩된 인수 값 세트상으로 평가될 출력 표현식을 각각의 고부하에 대해 식별하는 단계; (iii) 각각의 과부하의 인수 패턴을 입력 스트림 내의 패턴에 대한 매치를 효율적으로 인식할 DFA 로 번역하는 단계; (iv) 개개의 과부하의 DFA를 개개의 DFA 에 의해 매칭될 임의의 패턴을 매칭시킬 수 있는 결합된 DFA 로 전체적으로 다형성 함수에 대한 단일한 DFA 로 결합하고 매칭 입력을 처리해야 하는 과부하를 선택하는 단계; (v) 데이터 스트림을 결합된 DFA 에 적용하고, DFA 는 매칭 또는 매칭의 부재를 판단하기 위하여, 필요에 따라 스트림으로부터 데이터를 검사 또는 소비 또는 양자를 행하고, 매칭인 경우에, 입력 인수값을 적절하게 바인딩하며 평가될 적절한 출력표현식을 선택하는 단계; (vi) 출력표현식의 평가를 디스패칭하며 그 결과를 되돌리는 단계를 포함한다.

[0165] 반응함수에 의해 생성되는 뚜렷한 유형의 데이터 스트림 세트가 주어질 경우, 본 발명은 이러한 스트림이 그 출력이 통합된 유형의 단일 스트림으로 효율적으로 구성되게 하는 기술이다.

[0166] 이러한 종류의 문제를 해결할 필요성은 모든 형태의 데이터플로우 프로그래밍에 공통적으로 나타난다. 이것은 내장된 장치내의 이벤트의 흐름과 같은 아주 소규모의 아키텍처뿐만 아니라 기업 데이터 센터들 내부 또는 기업 데이터 센터들 간의 데이터의 흐름과 같은 아주 대규모의 아키텍처에 적용가능하다.

[0167] 본 발명은 데이터플로우 프로그래밍의 모든 도메인에 적용가능하지만, 일치가 감지될 수 있는 속도와 적용되는 핸들러 함수가 가장 중요하고 실행에 전념할 수 있는 제한된 저장소 및 컴퓨팅 리소스가 존재하는 상황에서 가장 적절하다.

[0168] 일 예. n 개의 개별 입력 스트림 Ai:0<k<n 으로 이루어지는 유입(inflow)을 고려한다. 각 스트림은 유형Ti 의 요소의 큐로 이루어진다. 각 스트림이 소비되며, 유형Ti -> Ui 의 반응함수 fi 에 의해 변형되며, n 개의 스트림 Bi 의 유출(outflow)이 존재하며, 각 스트림은 유형 Ui 의 요소의 큐로 이루어진다. 유형Σ Tk→ ΣUk 의 병합함수 m 을 사용하여, 모든 스트림 Bi 를 단일의 스트림 C 로 병합하고자 한다.

[0169] Vel 언어로 기재된 세 개의 스트림 사이에서 발생하는 그러한 병합의 일 예다.

표 4

[0170]
```
B0 = f0 from A0
B1 = f1 from A1
B2 = f2 from A2
C = B0 or B1 or B2
```

[0171] 스트림 C 는 B0, B1, B2로부터의 값으로 이루어지며, 생성될 대 인터리브된다. B 스트림은 C 스트림을 구성하는데만 사용하기 때문에, 그 스트림의 내용을 실현하는 것은 의미가 없다는 점에 주목하여야 한다. 그것은 익명의 임시 서브 표현식으로서 쉽게 표현될 수 있다.

표 5

[0172]
```
C = (f0 from A0) or (f1 from A1) or (f2 from A2)
```

[0173] 본 발명은 각 변환함수 fi 의 결정성 유한 오토마톤(DFA) 로의 번역 및 이러한 DFS 의 합집합으로서 병합함수의 한 개의 최소 DFA 로의 변환을 기술한다. 그 결과는 중간 플로우 Bi 의 내용을 실현할 필요없이, 인플로우 Ai 를 아웃플로우 C로 병합하는 최대로 효율적인 수단이다. 이 기술은 반복적으로 적용될 수 있으며, 중간 플로우의 후속 계층을 단일한 반응함수로 융합시킨다. 이것은 Vel 의 경우와 같이 선언적인 데이터플로우 언어에서의 중위(infix) 또는 연산자로 표기되는 병합의 개념과 일치된다.

[0174] 병합함수가 중간 플로우의 유일한 소비자인 경우라도, 이러한 문제는 브루트 포스 (brute force), 즉, 중간 플로우를 실현시키고, 그것들을 소비함으로서 해결될 수 있다.

[0175] 병합함수는 유입 및 유출 모두가 동일한 유형일 것을 필요로 하는 경우가 종종 있으며, 유형이 없는 시스템의 경우는 미분화된 유형이어야 한다. 이것은 유형 시스템의 합집합 유형 (합 유형이라고도 함)이 부족하기 때문이다. 데이터플로우 시스템으로의 참 병합의 존재는 합집합 유형의 사용을 필요로 한다.

[0176] 일부 데이터 시스템은 다중 입력 단일 출력 반응 변환을 구현하는 대신, 진정한 병합이 부족하다. 이러한 것들은 자신의 권한 내에서 유용한 구조이지만, 진정한 병합함수만큼 단순하고 일반적이지 않으며, 완벽하게 최적화될 수 없다.

[0177] 매칭함수를 DFA 로서 나타내는 것은 그것은 불린(Boolean) 유형의 임의의 표현식으로 표현하는 것보다 더 효율적이다. 각각 자신의 드라이빙 유입을 갖는 다중 매칭 함수의 DFS 는 단일 아웃플로우를 갖는 병합함수를 나타내는 단일하고 효율적인 DFA 를 형성한다. DFA 의 병합은 그 결과가 가능한 조기에, 또는 가능한 한 늦게 매칭되어, 두 개의 상이하고, 잠재적이며, 바람직한 행동을 가져올 수 있도록 행해질 수 있다. 복수의 반응을 단일한 DFA 로 구성하면 최소의 기계가 생성되며, 즉, 최소한의 판단을 사용하여 모든 매칭을 수행하는 알고리즘으로 귀결된다. 최소의 기계는 작은 플랫폼을 위한 복수의 반응의 가장 적절한 구현이다. 최소의 기계는 매칭함수의 복수개의 개별적인 평가에 대해 알고리즘적인 장점을 가지며, 따라서 다른 모든 것이 동일하면, 더 효율적으로 수행된다.

[0178] 변환 DFA 의 세트를 단일한 DFA 로 병합하기 위해, 정규 표현식에서의 교대를 고려하여야 한다. 정규 표현식을 번역함에 있어서, 구문 a|b 는 교대의 표현이다: a 를 매칭시키거나 또는 b 를 매칭시킨다. 이 경우에, a AND b 는 각각 변환 함수로부터의 DFA이다. 도11에 교대에 대한 서브그래프를 구성하였다.

[0179] 중간 NFA 가 완료된 후, 그것은 DFA 로 축소시키고 나서 최소 DFA 에 도달할 때까지 DFA 의 상태를 감소시킨다. 그 후, 자동화 상태 머신에서 채용되는 일반적인 종류의 소프트웨어 또는 하드웨어에 의해, 자동화에 적합한 상태-동작 테이블로서 DFA 를 렌더링한다. 이러한 테이블의 수용한 상태는 병합된 데이터 요소가 출력 스트림으로 방출되는 지점을 나타낸다.

[0180] DFA 가 이와 같이 자동화되고 입력 스트림의 세트를 제공할 경우, 그 입력과 관련된 원래의 변환함수에 다라 각 입력을 변환시키고, 단일한 출력상으로 함께 인터리브되는 결과를 얻게 될 것이다.

[0181] 입력 데이터의 복수개의 독립적인 스트림을 출력 데이터의 단일한 스트림으로 병합하는 방법은 (i) 복수개의 잠재적인 입력 데이터 스트림을 식별하는 단계, (ii) 입력 스트림 당 한 개의 변환함수로서, 각 입력 스트림에서 데이터 상에서 실행되며, 그 결과가 서로 병합되는 복수개의 변환 함수를 식별하는 단계, (iii) 복수개의 스트림으로부터 입력 데이터 요소를 동시에 수신하며, 데이터 요소를 단일한 출력 스트림으로 인터리브하는 병합함수를 식별하는 단계, (iv) 각 변환함수를 그 변환을 효율적으로 수행할 DFA 로 번역하는 단계, (v) 변환 DFA 를 변환을 효율적으로 수행하며 그 결과를 단일한 스트림으로 인터리브할 단일한 결합된 DFA 로 병합하는 단계, (vi) 데이터 스트림을 병합된 DFA 에 적용하며, DFA 는 변환 및 병합 작업을 수행하는 단계, 및 (vii) 병합된 출력을 사용을 위한 수신지로 디스패칭하는 단계를 포함한다.

[0182] 본 발명은 Vel 프로그래밍 언어에서의 소프트웨어를 개발하기 위한 도구 및 관련 방법이다. Vel 은 데이터플로우 프로그램을 표현하기에 유용한 프로그래밍 언어이다. 올바른 데이터플로우 프로그래밍은 많은 도전을 제시한다. 어떤 것은 컴퓨터 프로그래밍의 모든 형태에 대해 공통되는 도전이지만, 다른 것들은 데이터플로우 파라다임에 특유한 것들이다. 이러한 도구는 Vel 프로그램의 많은 영역을 어드레싱한다. (1) 구문론적 및 의미론적 정확성 검사, (2) 논리적 정확성 검사, (3) 지원의 디버깅, (4) 소스코드의 안전하고 휴대가능한 형태 (즉, 패키지화된 코드) 로의 번역, 소스코드 또는 패키지화된 코드의 다양한 컴퓨팅 플랫폼, 특히, 작은 플랫폼에 적합한 원시 및 최적의 이진 형태로의 번역, (6) 패키지화된 코드의 기술 및 그의 서명의 확인, (7) 패키지화된 코드의 배치모두 해석, Vel 소스의 상호적인 해석, (9) 패키지화되거나 원시코드를 실행하기 위한 데이터플로우 환경의 시뮬레이션, 및 (10) 라이브 데이터플로우 환경에서 이진 코드의 원격 실행, 모니터링, 및 제어를 포함한다.

[0183] 이것들은 Vel 언어내의 소프트웨어를 개발하는 누군가가 완수해야 하는 과제이다. 본 발명은 모든 이러한 영역들에 대해 충분한 지원을 제공하여, Vel 프로그래밍에 능숙한 사람들이 올바르고 유용한 소프트웨어를 생성할 수 있도록 한다.

[0184] 구문론적 및 의미론적 정확성을 검사하는 것은 모든 형태의 자동화 소프트웨어 번역에 공통되는 작업이다. 논리적 정확성을 검사하기 위한 도구는 일반적으로 번역도구에 포함되지 않는다. 이러한 종류의 도구는 일반적으로 독립적으로 존재하며, 종종 테스트중인 코드에 대한 불완전한 통찰력을 가지고 존재한다.

[0185] 디버깅은 소프트웨어 개발에 있어서 공통적인 작업이지만, 대부분의 디버깅 도구는 명령형 프로그래밍이 촛점을 맞춘다. 기능성 및 반응성 프로그래밍의 디버깅은 명령형 디버깅과 매우 다른 도전을 제시하기 때문에 훨씬 덜 공통적으로 어드레싱된다. 특히, 이 값들이 종종 디버거 (및 디버깅 프로그래머) 가 살짝 엿볼 수 있는 주소를 가지고 있지 않기 대문에, 이들 언어내의 "in flight" 연산을 검사하는 것이 어려울 수 있다.

[0186] 복수개의 원시 플랫폼 아키텍처를 목표로 하는 능력은 C 와 같은 시스템 언어의 컴파일러에게는 드문 일은 아니지만, 스크립트 수준의 언어들 사이에서 흔히 발견될 수 있는 능력은 아니다. 스크립트 언어는 컴파일되지 않거나, 부분적으로 컴파일 되거나, 그들의 호스트에 대해서 적시에 (just-in-time) 컴파일(jitted) 되는 경향이 있지만, 교차 컴파일 (한 개의 아키텍처 상에서 실행되지만, 다른 것에 대해서는 코드를 생성하는 컴파일러) 는 흔하지 않다. 특히, 작은 플랫폼상에서 실행을 위한 스크립트 수준의 언어를 컴파일링하는 것은 극히 드물다.

[0187] 상호작용하는 쉘은 스크립트 언어의 일반적인 특징이다. 예를 들면, 파이썬(python) 은 셀을 구현한다. 실제 또는 시뮬레이션된 데이터플로우 환경에 연결된 셀은 훨씬 덜 일반적이다.

[0188] 컴파일된 코드의 원격 실행은 일부 운영 시스템의 특징이며 오픈소스이며 상업적인 몇몇개의 제3자 도구에서도 활용가능하다. 이러한 경향은 특히 작은 플랫폼을 목적으로 하지 않지만, 작은 플랫폼을 위한 원격 실행도구의 일부 예가 존재한다. 그것들은 데이터플로우 프로그래밍에만 국한되지 않으며, 원격으로 실행된 프로그램을 개

발하기 위하여 사용되는 도구에 포함되지 않는다.

[0189] Vel 코드를 개발하기 위한 단일한 집적화된 도구는 Vel 언어로 작업하는 소프트웨어 개발자들에게는 유용하고 편리하다. 도구는 주로 Vel 언어를 번역하는 컴파일러이지만, 또한 Vel 프로그래밍과 관련된 다른 몇몇의 함수 세트도 제공한다. 이 도구로 구문론적 및 의미론적 정확성 테스트와 함께 논리적 정확성 테스트를 실행하는 것은 개발자가 효율적으로 코드의 정확성을 더 높일 수 있다. 논리 테스트는 코드에 대한 컴파일러의 인사이트라는 장점을 가지고 있으며, 진단 메시지가 더 완벽해 질 수 있다. 상호 쉘은 개발자가 코드를 테스트하고 즉각적인 반응을 얻을 수 있도록 한다. 이것은 디버깅 뿐만 아니라 개발에도 유용하다. 쉘은 또한 데이터플로우 환경에 대한 프로그래머 가시성을 제공한다.

[0190] 작은 플랫폼에서의 사용에 적합한 독립적인 이진의 실행가능한 코드를 생성하는 것은 종종 다양한 소형장치의 복잡한 연산을 실행하는 것에 의존하는 사물인터넷 (IoT) 의 사용사례를 가능하게 한다. 시뮬레이션된 데이터플로우 환경을 제공하는 것은 개발자가 그 코드에서 버그를 해결하고, 논리적인 정확성에 대한 테스트와 함께 패키지가 올바르게 작동한다는 것을 입증한다. 컴파일된 패키지의 원격 실행은, 특히, 원격 플랫폼이 소형인 경우, 대상 플랫폼이 그가 개발중인 것이 아니라고 하더라도, 프로그래머가 단일한 명령내의 대상 하드웨어에 대한 프로그램을 컴파일링하고 테스트하는 프로그램을 대해 신속하게 반복할 수 있게 한다.

[0191] 어휘적 표현으로부터 중간의 상징적인 표현으로 언어를 번역하고 (1단계 컴파일) 나서 이 중간 표현을 하드웨어를 연산함으로서 실행될 수 있는 형태로 변환하는 단계 (2단계 컴파일).

[0192] Vel 의 제1 단계 번역도구는 컴파일러에 공통인 일반적인 전력을 따른다. 특히, (1) 입력 문자열을 분석하여 토큰의 시퀀스로 분해. (2) 토큰의 시퀀스를 분석하여 구문트리를 형성. (3) 트리 내에서 상징적인 선언을 식별. (4) 트리 내에서 상징적인 기준을 식별 및 해결. (5) 공통 하위 표현식 제거 및 상수 폴딩과 같은 초기 최적화. (6) 유형 체크. (7) 최적화 및 심볼 성숙의 추가 단계. (8) 심볼의 완성 및 중간 표현의 방출.

[0193] Vel 제1단계 번역기의 구별되는 특징 중 하나는 결정성 유한 오토마톤 또는 DFA 의 사용으로 함수 애플리케이션에 필요한 패턴 매칭을 실행하고 반응을 트리거하는 것이다. 제1단계 번역도구는 (1) 입력 언어를 구문 트리로 변환하는 구문 분석기, (2) 번역 중인 프로그램이 자기 참고를 할 수 있도록 하여, 분석 중인 언어가 DSL 또는 매크로 분석기 방식으로, 분석기에 의해 변조될 수 있는 어휘적인 바인딩 구성요소, (3) 바인딩된 구문 트리를 데이터플로우, 패턴, 반응, 기능적 표현식, 타이머, 및 입/출력 매개변수화를 나타내는 심볼로 번역하는 구문 해석 알고리즘, (4) 표현식 트리를 마이크로프로세서 ALU 명령으로의 다소의 직접 번역에 적합한 스택으로 변환하는 표현 번역기, (5) 반응의 패턴 및 표현을 잠재적으로 최소가 아닌 DFA 의 중간적인 집합으로 번역하는 DFA 생성기, 및 (6) DFA 의 중간 집합으로부터 통일된 최소의 DFA 를 생성하는 DFA 결합 및 축소 알고리즘을 포함한다.

[0194] 제1단계 번역도구의 출력은 (1) 각각이 복수개의 스트림 중에서 고유한 참조일 수 있도록 번역에 관계된 스트림 각각의 논리적인 아이덴티티, (2) 내부를 향하거나 (반응을 향해, 즉, 외부소스의 구독), 외부를 향하거나 (반응으로부터 벗어남. 즉, 외부 수신지로의 공개), 내부 및 외부를 향하거나 (공개/구독 쌍), 또는 내적인 (다른 반응의 중간 단계로서만 사용되고 공개 또는 구독과 같이 표면화되지 않음) 각각의 스트림 내의 데이터의 플로우의 기재, (3) 스트림으로 삽입되거나 스트림으로부터 추출되는 데이터가 유형 정확성을 위해 정적으로 확인될 수 있도록 유한한 용어로 기술되는, 각 스트림에서 흐르는 데이터의 유형의 기재, (4) DFA 의 상태 및 전이를 기술하는 테이블 세트, (5) 반응동안 실행될 계산을 기술하는 표현식 스택 세트, (6) 스트림 입력을 DFA 입력으로 매핑하는 테이블, (7) 시간이 정해진 이벤트를 DFA 입력으로 매핑하는 테이블, DFA 출력을 동작쌍으로 매핑하는 테이블로서, 각 쌍은 표현식 스택 및 스트림 출력에 대한 기준으로 구성되며, DFA 의 출력이 주어진 표현식에 의해 변환된 후, 주어진 스트림으로 푸시될 것을 나타낸다.

[0195] Vel 해석기 및 데이터플로우 시뮬레이터는 제1단계 번역의 출력을 직접 사용한다. 해석기는 코드의 실행시에 하드웨어 플랫폼을 모방하고, 데이터플로우 시뮬레이터는 스트리밍 데이터 환경을 모방하여, 입력을 Vel 스트림에 제공하고, Vel 스트림으로부터 출력을 수집한다. 이러한 두 작업을 명령 해석 및 데이터플로우 모방이라고 부르기로 한다.

[0196] 명령어 해석은 컴파일러와 해석기 작성을 전문으로 하는 컴퓨터 프로그래머에게 잘 이해되는 작업의 카테고리이다. 이 작업은 런타임 변수의 상태가 저장될 수 있는 실행 컨텍스트를 구성하는 단계와 프로그램의 명령어를 통해 한번에 한 단계씩 나아가는 단계, 실행 컨텍스트로부터 데이터에 접근하는 단계, 및 필요에 따라 그것을 업데이트 하는 단계를 포함한다.

[0197] Vel 의 경우에, 실행 컨텍스트는 또한, 변환 과정에서 데이터의 스트림을 유지하기 위한 큐의 세트 및 DFA 에 의해 기술된 변환을 실행하기 위한 테이블로 구동되는 상태 머신 엔진을 포함해야 한다. 큐는 데이터의 흐르는 채널을 기술하는 Vel 소스에서의 선언으로 인해 야기된다. 이들 중 일부는 Vel 프로그램의 외부 입력 또는 출력이며, 다른 것은 입력 및 출력간의 중간 상태를 기술하는 순수한 내부 채널이다.

[0198] 데이터플로우 모방은 외부 소스 및 파일 또는 소켓과 같은 데이터용 싱크에의 액세스, 이러한 외부 시스템들 사이에서 데이터를 교환하기 위하여 필요한 프로그래밍, 및 해석중인 Vel 프로그램으로 이루어진다. 이것은 외부 소스로부터 데이터를 읽고, 그것을 프로그램 입력을 나타내는 큐로 푸시하는 주입기 (injector) 기능 및 프로그램 출력을 나타내는 큐로부터 데이터를 출력하여 외부 싱크에 기입하는 추출기 (extractor) 기능을 가질 것이다.

[0199] 본 발명에 따른 Vel 해석이 표준과 다른 것은 DFA 가 관여하는 방식에 있다. 상태 머신 엔진은 큐로부터 데이터를 읽고 그것을 사용하여 그들의 DFA 의 상태를 진전시킨다. DFA 테이블은 DFA 가 그 상태를 통해 이동할 때 실행되는 부작용의 칼럼을 포함한다. 이러한 부작용은 명령해석을 호출하여 연산을 실행하며, 그 결과가 다른 큐로 푸시되며 이것이 다른 DFA 를 트리거한다.

[0200] 이러한 방식으로, 본 발명에 따른 해석중인 Vel 프로그램이 우선 빠르고 소형인 상태 머신 세트에 의해 표현되고 필요할 때 일반적인 명령 해석으로만 복귀한다. 모든 것이 명령 해석으로만 모두 처리되는 것보다 더 효율좋게 프로그램이 실행될 수 있도록 한다.

[0201] Vel 의 제2단계 번역도구는 대부분이 Vel 언어에만 특유한 것이 아니라, 실행 대상인 플랫폼이다. 제2단계 번역기의 Vel 언어 관련 구성요소는 (1) 제1단계에 의해 생성되는 중간 표현의 초기 섭취량, (2) 반응 시스템을 생성하기 위한 제2단계 코드 생성의 전반적인 조직, (3) 데이터 포맷 또는 실시간 클록의 내부 규칙의 외부 부호화 및 복호화를 실행하는 것과 같은, 런타임 지원 구성요소의 라이브러리의 제공이다.

[0202] 다중 소스, 다중 목적지 데이터플로우 환경에서 데이터 스트림의 실시간 처리를 위한 프로그램을 만드는 도구는 (1) 복수개의 잠재적인 데이터 스트림을 식별하는 단계, (2) 스트림 내부의 데이터의 패턴에 대응하는 반응 함수 및 매개변수의 세트를 식별하는 단계, (3) 선언된 패턴과 일치하는 데이터를 변환하기 위한 처리함수 및 매개변수 세트를 식별하는 단계, (4) 데이터가 수집되거나 버려지는 시간 간격 또는 데이터가 수집되거나 버려지기 전 또는 후의 특정한 시각과 같이 데이터플로우의 패턴이 비교되는 시간이 정해진 이벤트 세트를 식별하는 단계, (5) 식별된 스트림, 반응, 함수, 및 시간이 정해진 이벤트를 기술하는 데이터플로우 프로그램을 생성하는 단계, 대응하는 DFA 로 Vel 프로그램 선언을 번역하는 DFA 생성기를 포함하는 제1단계의 번역도구 및 플랫폼 상에서의 실행을 위한 번역된 Vel 선언에 대응하는 플랫폼 특유의 하드웨어 명령을 생성하는 제2단계의 번역도구를 포함하는 2단계 번역도구에 입력으로서 프로그램을 제공하는 단계, 및 (7) 번역도구의 각 단계의 출력을 수신하는 단계를 포함한다.

[0203] 제1단계의 번역도구의 출력은 해석기 구성요소에 의해 사용될 수 있으며, (1) 코드의 실행시에 하드웨어 플랫폼을 모방하는 명령 해석기, (2) Vel 스트림에 입력을 제공하고, Vel 스트림으로부터의 출력을 수집하는 스트리밍 데이터 환경을 모방하는 데이터플로우 시뮬레이터를 포함한다.

[0204] 제1단계 번역도구의 출력은 제2단계 번역도구에 대한 입력으로서 사용될 수 있으며, (1) 중간 표현으로부터의 명령을 대상 하드웨어 플랫폼에 의한 실행에 적합한 형태로 번역하는 하드웨어 명령생성기, (2) 데이터플로우환경에서의 반응 프로그램으로서 사용하기에 적합한 형태로 출력의 생성을 지시하는 프로그램 구성모듈, (3) 실행에 필요한 실시간 지원 구성요소의 라이브러리를 포함한다. 제2단계의 번역도구의 출력은 대상 하드웨어 플랫폼에서 사용하기에 적합한 실행프로그램이다.

[0205] 도14는 입력으로부터 가상 센서를 생성하기 위해 사용되는 Vel 센서 표현 언어 엔진(1407)을 도시한다. 센서 표현 언어 엔진은 물리적 센서 또는 다른 가상 센서로부터 입력을 받는다. 입력의 일부 예는 주입구 압력(1411), 출구압력(1414), 온도(1417), 및 흐름(1420) 이다. 임의의 수의 입력과 입력의 조합이 가상 센서로의 입력으로서 사용될 수 있다. 입력에 기초하여, 센서표현 언어 엔진은 차동 압력(1431), 온도(1434) (켈빈으로 나타냄), 및 증기압(1437)과 같은 출력을 갖는 가상 센서를 생성할 수 있다. 임의의 수의 가상센서와 출력이 존재할 수 있다. 상술한 바와 같이, 출력은 가상센서에의 입력의 수학적 함수일 수 있다.

[0206] 도14는 가상 센서의 대표로 상자(예를 들면, 1431, 1434, 1437) 를 도시하고 있다. 가상센서는 복수개의 출력을 가질 수 있다. 예를 들면, 가상센서(1431 및 1434)는 두 개의 출력을 갖는 한 개의 가상센서로 결합될 수 있다. 예를 들면, 가상센서(1431, 1434, 1437)는 세 개의 출력을 갖는 한 개의 가상센서로 결합될 수 있다.

[0207] 도14에서 Vel 센서 표현 언어 엔진(1407)의 내부는 이하에 기술될 데이터플로우 프로그래밍에 의해 특징지어질 수 있다. 입력으로서 기능하는 뚜렷한 형태의 데이터 스트림 세트 및 이들 입력을 데이터플로우 프로그램을 통해 출력 스트림으로 변환하고자 하는 요구가 주어지면, 본 발명은 그래프의 주된 동작으로서 패턴-매칭 동작을 용이하게 하고, 데이터 변환을 호출하는 주된 매커니즘으로서 패턴-매칭을 통합하는 방식으로 데이터플로우 그래프를 구현하는 기술이다.

[0208] 이러한 종류의 문제를 해결할 필요성은 모든 형태의 데이터플로우 프로그래밍에 공통적으로 나타난다. 이것은 내장된 장치내의 이벤트의 흐름과 같은 아주 소규모의 아키텍처뿐만 아니라 기업 데이터 센터들 내부 또는 기업 데이터 센터들 간의 데이터의 흐름과 같은 아주 대규모의 아키텍처에 적용가능하다.

[0209] 본 발명은 데이터플로우 프로그래밍의 모든 도메인에 적용가능하지만, 패턴 매칭이 감지될 수 있는 속도와 적용되는 핸들러 함수가 가장 중요하고 실행에 전념할 수 있는 제한된 저장소 및 컴퓨팅 리소스가 존재하는 상황에서 가장 적절하다.

[0210] 일부 백그라운드 정보가 데이터 프로그래밍에 대하여 이하에 제시되며, 패턴 구동되는 반응에 대한 이하의 논의에 대한 기초로서 작용할 것이다. 데이터플로우 프로그래밍은 프로그램의 입력과 출력을 구체화하는 외부 노드를 갖는, 흐름 그래프라 불리는, 방향성 그래프로서 컴퓨터 프로그램을 모델링하고, 각 내부 노드는 실행될 동작을 구체화한다. 각 노드로 흐르는 데이터는 그 동작에 입력을 제공하고, 그 동작의 결과는 노드로부터 흐르는 데이터를 제공한다. 한쌍의 노드들 사이의 방향성 에지는 한 개의 노드의 출력이 다른 것의 입력이 되도록 한다. 즉, 데이터가 그래프로 들어가서, 다양한 동작을 통해 그래프 내부에서 이동하며, 계속적인 흐름 내에서 유용한 방식으로 변형된 그래프를 빠져 나간다.

[0211] 흐름그래프 구조. 도15는 흐름 그래프의 예를 도시한다. 데이터는 세 개의 입력노드 (in1 1506, in2 1509, 및 in3 1512)를 통해 그래프로 들어가고, 네 개의 독립된 동작노드 (op1 1522, op2 1525, op3 1528, 및 op4 1533)에서 변형되며, 두 개의 출력노드 (out1 1546 및out2 1548)를 통해 빠져 나간다. 에지e1 내지 e7 는 데이터가 흐르도록 허용된 경로를 나타낸다.

[0212] 입력노드는 주입기라고 명명될 수 있고, 출력노드는 추출기라고 명명될 수 있다. 이들 노드는 그래프를 외부의 데이터 시스템과 연결시키도록 기능하며, 일부 구현에서 그것을 임시적으로 버퍼링할 수는 있지만, 어떤 방식으로 데이터를 변형시키지 않는다. 도15에서 주입기와 추출기는 뒤집어진 집 모양의 노드로 표현된다.

[0213] 동작이 실행되는 내부 노드는 변환기라고 명명된다. 변환기에 도달한 각 입력에 대해, 변환기는 0 이상의 출력을 생성할 수 있다. 도15에서 변환기는 원모양의 노드로 표현된다.

[0214] 출력을 전달하는 노드는 생산자라고 명명되며, 입력 데이터를 수신하는 노드는 소비자라고 명명된다. 주입기는 일종의 생산자이며, 추출기는 일종의 소비자이다. 변환기는 생산자 및 소비자 모두 이다. 도16은 이러한 관계를 도시한다.

[0215] 도16은 노드의 종류의 계층을 도시한다. 주입기(1608) 및 변환기(1613) 이 생산자(1619)에 입력한다. 변환기(1613) 및 추출기(1622)는 소비자(1625)에게 입력한다. 생산자(1619) 및 소비자(1625)는 노드(1634)에 입력한다.

[0216] 각 생산자는 그것에 부착된 0 개 이상의 아웃고잉 에지 (outgoing edge)를 가질 수 있다. 각각의 밖으로 나가는 에지는 생산된 데이터를 생산자로부터 멀리 전달한다. 각 소비자는 그것에 부착된 0 개 이상의 인커밍 에지 (incoming edge)를 가질 수 있다. 각각의 인커밍 에지는 소비될 데이터를 소비자에게 전달한다.

[0217] 각각의 고유한 에지는 정확하게 한 명의 생산자를 한명의 소비자에게 연결하며, 임의의 주어진 생산자 및 소비자 쌍에 대해, 최대한 한 개의 고유의 에지가 존재할 수 있다. 또 다른 방식으로, 임의의 주어진 생산자 및 소비자쌍이 고유한 에지에 의해 서로 연결되거나 연결되지 않는다.

[0218] 에지의 생산자의 관점으로 볼때, 에지가 생성된 데이터를 멀리 전달하므로 각각의 고유한 에지는 일종의 아웃고잉 에지이다. 또한, 에지의 소비자의 관점으로 볼때, 에지가 소모될 데이터를 내부로 가져오므로 각각의 고유한 에지는 일종의 인커밍 에지이다.

[0219] 도17은 에지의 종류 및 노드에 대한 카디널리티 (cardinality) 의 계층을 도시한다. 이 도면은 에지의 종류와 생산자 및 소비자와의 카디널 관계를 도시한다. 생산자(1708) 및 고유한 에지(1714)는 아웃고잉 에지(172)으로 입력한다. 고유한 에지(1714) 및 소비자(1723)은 인커밍 에지(1726)으로 입력한다. 아웃고잉 에지(1720) 및 인

커밍 에지(1726)은 에지(1735)로 입력한다.

[0220] 변환. 상기한 바와 같이, 주입기와 추출기는 그들의 데이터를 변경하지 않는다. 데이터의 모든 변경은 변환기에 의해 행해진다. 함수 애플리케이션과 같이, 변환은 입력값을 받아 출력값으로 변환하지만, 함수와는 달리, 변환은 복수의 입력값을 받아 그것들을 유지하면서, 나중에 0개 이상의 출력값을 생성할 수 있다. 변환의 출력들은, 비록 한꺼번에 생산된다고 하더라도, 각각 출력 스트림으로 들어간다.

[0221] 입력 데이터는 한 개 이상의 생산자로부터 변환기에 도달한다. 새로운 데이터가 도달할 때마다, 변환기가 작동하는데, 그것은 작동할 수 있는 기회가 주어진다는 것을 의미한다. 동작은 새로운 데이터를 폐기하거나, 나중까지 내부적으로 그것을 버퍼링하거나 또는 이전에 버퍼링된 데이터와 함께 새로운 데이터를 사용하여 한 개 이상의 새로운 출력을 생성하는 것을 선택할 수 있다.

[0222] 예를 들면, 도15에서 op1 이 (에지 e1을 따라) 입력으로서 단일한 정수 스트림을 소비하고, (에지 e5를 따라) 출력으로서 정수 스트림을 생성하는 변환기라고 가정한다. 또한, op1 은 두 개의 입력을 내부적으로 버퍼링하고, 제3 입력의 도착시에, 세 개의 입력의 평균과 동일한 출력을 생성하도록 프로그래밍되어있다고 가정한다.

[0223] 트레이스(1)는 op1 주위의 흐름 그래프의 부분의 동작의 가능한 예를 도시한다. 입력 데이터는 12시에 도착하기 시작하여, 12시5분에 op1 이 세 개의 입력 [12, 16,5] 를 보았고, 그것들의 평균은 11이다. 동작노드 op1 은 12시5분에 출력으로서 평균을 생성하고, 동일한 순간, 마지막 입력을 수신한다. 이러한 패턴이 12시6분에 시작되어 반복하며, op1 은 12시10분에 또 다른 출력을 생성한다. 기타 등등.

표 6

[0224]

```
1.      e1 [ 12 @ 12:00, 16 @ 12:02, 5 @ 12:05,
          18 @ 12:06, 22 @ 12:08, 3 @ 12:10 ... ]
        ==> op1 ==>
        e5 [ 11 @ 12:05, 14 @ 12:10 ... ]
```

[0225] 인바운드 에지를 통해 도착하는 공식적인 입력 외에, 변환기는 또한 타임아웃의 형태로 비공식적인 입력을 선언할 수 있다. 타임 아웃은 변환기가 그 활성화 사이에 만료되기를 희망하는 최대 시간이다. 타임아웃이 만료되기 전에 새로운 입력이 도달하지 않으면, 변환기는 활성화되지만, 타임아웃이 되었다는 것을 나타내는 특별한 상태에 있게 될 것이다. 이러한 방식으로, 변환기는 데이터의 존재뿐만 아니라 데이터가 존재하지 않을 때에도 동작할 수 있다. 변환기가 작동 될 때 마다, 새로운 타임아웃을 선언하기 위한 옵션을 갖는다.

[0226] op1 의 알고리즘을 수정하여 10분의 타임아웃을 갖는다고 하자. 만약 그 시간 내에 세 개의 데이터가 도착하지 않으면, op1 은 가지고 있는 값이 무엇이든 그 값의 평균을 생성할 것이다.

[0227] 트레이스(2)는 새로운 시간 인식 op1 주위의 흐름 그래프의 부분의 동작의 가능한 예를 도시한다. 입력 데이터가 op1 가 10분의 타임아웃을 선언한 시점인 12시에 도착하기 시작한다. 또 다른 입력이 2분 후인 12시 2분에 도착하지만, 타임아웃이 12시10분에 만료되기 전에 더 이상의 데이터가 도착하지 않는다. 12시10분에, op1 은 보여지는 두 개의 입력의 평균을 생성하고 더 이상의 타임아웃은 없다고 선언한다. 기타 등등.

표 7

[0228]

```
2.      e1 [ 12 @ 12:00, 16 @ 12:02, timeout @ 12:10 ... ]
        ==> op1 ==>
        e5 [ 14 @ 12:10 ... ]
```

[0229] 샘플링 대 트리거링. 변환기가 두 개 이상의 입력을 가질 경우, 때대로, 그 입력 중 일부는 항상 변환기가 출력을 생성하도록 하지만, 다른 입력은 출력을 생성하도록 하지 않는 경우가 있다.

[0230] 예를 들면, 도14에서 op4 이 (에지 e5 및 e6을 따라) 입력으로서 두 개의 정수 스트림을 소비하고, (에지 e7를 따라) 출력으로서 정수 스트림을 생성하는 변환기라고 가정한다. 또한, 입력이 e6을 따라 도착할 때마다, op4가 입력의 사본을 내부적으로 저장하는 방식으로 op4가 프로그래밍되었다고 가정한다. 이러한 방식으로, op4는 항상 e6 으로부터 가장 최근의 값을 알게 되지만, e6을 따라 값이 도착한다고 하더라도, op4가 무엇인가를 생성하

게도록 하지는 않는다.

[0231] 또한, 입력이 e5를 따라 도착할 때마다, op4는 입력과 e6으로부터의 가장 최근의 값의 합과 동일한 e7을 따른 출력을 생성할 것이다. 트레이스(3)은 op4 주위의 흐름 그래프의 부분의 동작의 가능한 예를 도시한다.

표 8

[0232]

```
3.      e5 [ 11 @ 12:05, 14 @ 12:10, 17 @ 12:20 ... ],
        e6 [ 20 @ 12:00, 30 @ 12:15 ... ]
        ==> op4 ==>
        e7 [ 31 @ 12:05, 34 @ 12:10, 47 @ 12:20 ... ]
```

[0233] 12시에, op4는 값 20이 e6을 따라 도착한 것을 본다. 이것을 e6의 가장 최근의 값으로서 캐시에 저장한다. 12시 5분에, e5 를 따라 값이 도착한다. 각각의 경우에, op4는 e5의 생생한 값과 e6의 캐시에 저장된 값을 사용하여 출력을 생성한다. 12시15분에, op4는 e6으로부터 새로운 값 (30)을 캐시에 저장하며, 또 다른 입력이 e5를 따라 도착할 때, 그것을 사용하여 12시20분에 출력을 생성한다.

[0234] 이 예에서, op4는 e6을 샘플링하고 있지만, e5에 의해 트리거되고 있다고 말한다. 샘플링은 변환기 내부의 데이터의 축적과 관련된 적극적인 동작이고, 트리거링은 새롭고 축적된 데이터를 활용하여 출력을 생성하는 동작이다.

[0235] 때때로, 변환기가 입력 모두를 샘플링하고 타임아웃까지만 트리거링 하는 경우가 있다. 상술한 op1 의 시간 인식버전은 그러한 변환기의 일 예다.

[0236] 변환기는 또한 전혀 인바운드 되지 않은 에지를 가지며 여전히 출력을 생성할 수 있다. 그러한 변환기는 타임아웃시에만 트리거링하는 단순한 타이머이다. 고정된 스케쥴 상에서 생성된 안정된 데이터 시리즈를 갖기를 원하는 애플리케이션에 유용할 수 있다.

[0237] 트리거링 조건을 갖지 않는 변환기는 쓸모가 없고 잘못된 것이다.

[0238] 스케줄링. 변환기는 입력을 수신할 때마다 또는 타임아웃이 만료될 때, 또는 양자의 경우에 변환기는 작동된다. 각각의 작동시에, 변환기는 0 개 이상의 출력을 생성한다. 아직 소비되지 않는 출력을 생한한 생산자는 핫(hot)할 수 있고, 그렇지 않으면 차갑다(cold).

[0239] 뜨거운 생산자의 출력을 조절하는 기술을 선택하는 것은 그래픽 엔진에 달려있다. 두 개의 공통적이고 공평한 유효한 두 개의 전략이 존재하는 데, 하나는 생산자에 의해 구동되고 나머지 하나는 소비자에 의해 구동되는 것이다.

[0240] 생산자에 의해 구동되는 그래프에서, 생산자는 가능한 한 빨리 작동된다. 그것이 생산하는 출력은 외부 에지를 따라 이동하고 소비자측에서 대기열에 놓인다. 생산자가 데이터가 준비되자 마자 데이터를 소비자에게 푸시하기 때문에 이것은 푸시모델이라고 불린다. 이 모델에서, 생산자는 최소한의 시간 동안 핫(hot)한 상태로 존재한다.

[0241] 소비자에 의해 구동되는 그래프에서, 생산자는 소비자가 소비할 준비가 되었을 때 작동한다. 소비자가 이미 그것을 기다리고 있기 때문에 생산자가 생산하는 출력은 대기열에 놓이지 않는다. 소비자가 생산자로부터 데이터를 끌어당기기 때문에, 이것은 또한 풀 모드 (pull mode) 라고 불린다. 이 모델에서, 생산자는 무한정 핫하게 존재할 수 있다.

[0242] 데이터플로우의 방향은 생산자로부터 소비자로, 어떤 전략에서든 동일하다는 것에 주목하여야 한다. 차이는, 생산자나 소비자 중 어떤 쪽에 데이터플로우를 시작하려는 충동이 상주하느냐 이다.

[0243] 데이터플로우 프로그램의 구현. 거의 모든 프로그래밍 언어로 데이터플로우 컨셉을 구현할 수 있지만, 일부 언어는 다른 것보다 더 적합할 수 있다.

[0244] 스칼라 또는 헤스켈과 같은 기능언어는 부작용 없이 평가될 일련의 수학식으로서 프로그램을 모델링하기 때문에, 언어모델 프로그램 데이터플로우 프로그램은 종종 스칼라 또는 헤스켈과 같은 기능 언어로 기재된다.

[0245] 기능언어로 구현된 그래프에서, 각 변환기의 동작은 수식으로 정의되며, 각 노드의 런타임 동작은 단순하게 그 수식의 평가이다. 부작용으로부터의 자유는 실행 모델을 단순화하여 노드가 반복적으로 독립적으로 작동하도록 보장합니다. 수식을 결합하고 단순화하기 위하여 사용되는 대수학이 그래프 노드에 응용될 수 있기 때문에, 그

래프 최적화도 또한 단순화된다.

[0246] 하지만, 순수한 기능언어에서, 획득된 상태에 의존하는 변환을 실행하는 것은 어려울 수 있다. 그러한 경우에, 일련의 명령과 (내부 버퍼링과 같은) 부작용을 허용함에 따라 그러한 언어가 프로그램을 모델링하기 때문에, Java 또는 C++과 같은 명령형 언어가 바람직할 수 있다.

[0247] 명령형 언어로 구현된 그래프에서, 각 변환기의 동작은 입력이 도달할 때마다, 타임아웃이 만료될 때, 또는 양자의 경우에 호출되는 이벤트 핸들러 (event handler) 함수로 정의된다. 함수는 적합하다고 생각되는 내부 동작을 자유롭게 취할 수 있으며, 출력을 생산하거나 타임아웃을 선언하거나 양자 모두를 할 수 있다. 이벤트 핸들러의 동작에 대해 외부적으로 판단하기 어렵기 때문에, 이것은 구현자에게 큰 유연성을 부여하지만, 그래프 최적화를 매우 더 어렵게 한다.

[0248] 언어와 관계없이, 그래프의 자동화 프레임워크(framework) (그래프엔진), 그 노드의 상세, 및 그 엔진의 접속이 코드 라이브러리를 사용하여 일반적으로 다루어진다. 일반적으로 라이브러리는 클래스와 같은, 변환기를 구성할 추상화(abstractions), 변환기를 에지와 접속하는 방법, 및 이벤트 구동 시스템으로서 그래프를 실행할 메인 루프를 제공한다. 언어의 유연성 및 라이브러리는 그래프로 표현가능한 제한 및 그래프가 실행될 효율을 정의한다.

[0249] 패턴 구동되는 반응. 본 발명의 주된 개념은 패턴 매칭작업 주변의 변환기의 동작을 체계화하는 것이다. 본 논의의 목적을 위해, 패턴 매처(또는 짧게 말해서, "매처") 는 구문적이고 시간 관련된 규칙 세트가 주어질 경우, 데이터를 갖는 이벤트의 인커밍 스트림을 일반적인 데이터 구조로 체계화할 수 있다. 시간상 전방 최우측 (timed forward right-most)(1) 또는 TFR(1) 패턴 매칭이라고 불리는 기술이 그러한 알고리즘의 일 예다.

[0250] 논리적 프로그래밍. 패턴 선언은 논리점 프로그래밍의 한 형태이며, 명령형 프로그래밍 및 기능적 프로그래밍과는 다른 모델이다. 논리적 프로그래밍에서, 한 개 이상의 자유변수가 선언되며 어떻게 값들이 그 변수에 할당될 것인지에 대한 규칙이 주어진다. 또한, 한 개 이상의 종속 식이 정의되며, 그것은 자유변수에 대한 참조를 만든다. 실행시에, 자유 변수는 규칙에 따른 값을 획득하고, 종속 식이 평가된다.

표 9

[0251]

```
4.      a = b + 1 for b in [ 1, 2, 3 ]
```

[0252] 정의(4)에서, b 는 자유 변수이며, b+1 은 b에 관계된 종속식이다. 값을 b에 할당하는 규칙은 단순하다. b 는 순서대로 리스트 [1, 2, 3]의 값이라고 가정한다. 그러므로, 실행이후, a 는 [2, 3, 4] 와 같다.

[0253] 패턴 규칙. 패턴규칙은 입력 스트림에서 예상될 패턴을 선언하는 논리 식의 형태이다. 그것은 람다식과 유사해 보이지만, 대신에 아래첨차 파이 (π) 를 도입하여 구분한다.

표 10

[0254]

```
5.      p1 = π a, b, c → (a + b + c) / 3
```

[0255] (5) 에서, 패턴 p1 이 순서대로 세 개의 자유 변수, a, b, c 가 가 될 것이라고 선언된다. 입력 스트림에 적용될 경우, 각 자유변수는 입력으로부터 한 개의 값에 바인딩되며, 그것은 각각의 자유변수가 패턴이 적용되는 스트림과 동일한 유형이라는 것을 의미한다. 패턴을 적용한 결과가 종속식에 의해 화살표 우측으로 주어지고, 이 경우에, 세 개의 입력의 평균과 같아진다. 사실상, 이러한 패턴이 (1)에서 도시된 변환이다.

[0256] 함수가 값에 적용되는 방식으로 p1 을 스트림에 적용할 수 있다. 예를 들면, x가 정수 스트림이라면, p1(x) 는 세 개로 그룹화되고 평균화된 x 의 값으로 이루어진다. p1(p1(x)) 은 평균의 평균일 수 있다.

표 11

[0257]

```
6.      p2 = π d[3] → sum(d) / 3
```

[0258] (6) 에서는, 약간 짧아지고 더 일반적일 수 있도록 (5)를 수정한다. 세 개의 입력을 (a,b,c) 로 개별적으로 나열하는 대신, 그것들을 세 개의 구성요소의 길이를 갖는 한 개의 입력(d) 으로 모은다. d 는 단지 한 개 인 대

신에, 입력스트림으로부터 세 개의 값을 매칭한다. 이러한 종류의 스트림이 T 라면, d 의 유형은 리스트(T) 가 될 것이다. 리스트의 구성요소를 합하는 내장함수 합의 도움으로 평균을 연산한다. p2 는 기능적으로 p1 과 동일하다.

표 12

[0259]

```
7.      p3 = π d[1..3 or 10(min)] → sum(d) / len(d)
```

[0260] (7) 에서, 타임아웃을 갖는 가변 길이 매칭의 개념을 도입하도록 (6) 을 수정한다. d 의 길이는 간격 1..3 으로 주어지고 그것은 입력으로부터 1과 3 사이의 값에 매칭한다는 것을 의미한다. d 의 유형은 리스트(T) 로 존재하며, 한 개의 입력만 매칭한다면, 한 개의 구성요소의 목록이 될 수 있다. 매처는 도달할 최대 3 개의 값에 대해 최대 10분동안 기다릴 수 있으며, 시간이 만료되면, 한 개 또는 두 개의 값을 받아들일 것이다. 리스트에 있는 구성요소의 갯수를 반환하는 추가적인 내장함수 len 의 도움으로 평균이 연산된다. 사실상, 이러한 패턴이 (2) 에서 도시된 변환이다.

표 13

[0261]

```
8.      p4 = π d[1.. or 10(min)] → sum(d) / len(d)
```

[0262] (8) 에서, 타임아웃을 갖는 개방 종단 매칭의 개념을 도입하도록 (7) 을 수정한다. d 의 길이는 상한이 없다. 이것은 적어도 한 개의 값에 매칭되며 10분 이내라면 가능한한 많이 매칭될 수 있다. 아 패턴은 타임아웃에 의해서만 트리거링된다.

표 14

[0263]

```
9.      p5 = π d[1..] → sum(d) / len(d)
```

[0264] (9) 에서, 타임아웃을 갖지 않는 개방 종단 매칭의 개념을 도입하도록 (8) 을 수정한다. (8) 에서와 같이, d의 길이는 상한을 갖지 않지만, 아무것도 패턴을 트리거링 하지 않는다. 그 자체로 적용된다면, 이 패턴은 잘못된 것이다. 이후 단락의 주제인 조인(join)의 컨텍스트에서만 의미가 있다.

[0265] 플로우 그래프의 관점에서, 패턴 규칙의 각 응용은 한 개의 입력을 갖는 변환기를 정의한다. (패턴이 타임아웃을 갖는다면, 변환기도 또한 타임아웃을 가지게 될 것이다). 변환기는 패턴규칙으로부터 유도된 매처의 정의를 포함한다. 입력값이 도달하면, 매처로 보내진다. 매처가 매칭을 생성하면, 매칭은 종속식을 평가할 인수로서 사용되고, 그 결과는 변환기에 의해 생성된다.

[0266] 아직 만족되지 않은 패턴은 찬 변환기를, 완벽하게 만족된 패턴은 뜨거운 변환기를 생성한다. 하지만, 가변길이의 패턴이 부분적으로 만족되면, 변환기를 기술하기 위하여 새로운 상태인 따듯한 상태가 요구된다. 따뜻한 변환기는 출력을 생산할 준비가 되어있지만, 생산할 필요는 없다. 이것은 준비가 되어 있어야 하고, 가능한 한 빨리 생산을 해야 하는 뜨거운 변환기와는 반대된다.

표 15

[0267]

```
10.     p6 = π d[1..3] → sum(d) / len(d)
```

[0268] (10)에서 정의되며, 변환기에 의해 표시되는, 입력 스트림에 적용되는 p6 을 고려한다. 임의의 입력이 도착하지 전에, 변환기는 차가울 것이다. 한 개의 입력이 도달한 후, 패턴이 부분적으로 만족되기 때문에, 변환기는 따뜻해 질 것이다. 2 개 이상의 입력을 받아들 일 수 있으며, 이미 가지고 있는 것은 매칭될 수 있다. 2개 이상의 입력이 도착한 후, 변환기는 뜨겁게 될 것이다. 입력을 먼저 생성하지 않는다면, 더 이상의 입력을 받아들일 수 없다.

[0269] 변환기의 준비상태에 대한 2가지 상태(뜨겁고, 차가운 상태) 모델 대신에, 3가지 상태 (뜨겁고, 따뜻하고, 차가운 상태) 모델은 본 발명의 주된 특징이다. 이것은 패턴구동 반응을 직접적으로 가능하게 한다. 그것이 없으면, 가변 길이 패턴은 사용불가능할 것이다.

[0270] 조이닝(joining). 조인은 한 개의 출력 스트림을 생성하기 위하여, 서로 평가될 두 개 의상의 입력 스트림을 서로 얽매는 것이다. 예를 들면, 냉장고에 두 개의 센서-냉장고 문용 및 온도용-가 있다면, 문이 닫혀 있을 때 평균 온도를 생산하기 위해 두 개의 스트림을 서로 결합한다.

표 16

[0271]

```
11.     z1 = a + b for a in x or b in y
```

표 17

[0272] 테이블 B

Time	x	y	z1
12:00	11	22	33
12:05	12		34
12:10		24	36
12:15	13	26	39

[0273] 입력 중 하나가 변화할 때 출력이 생산되며, 입력의 변화에 의해 그 평가가 트리거링된다는 것에 주목하여야 한다.

표 18

[0274]

```
12.     z2 = a + b for a in x or b in y or every 3(min)
```

[0275] (12)에서, 타임아웃이 3분 이라는 추가적인 트리거링 조건을 추가한다. 다음 테이블은 이러한 새로운 트리거링 조건의 관점에서, x, y, 및 z2 의 임의의 가능한 값들을 보여준다.

표 19

[0276] 테이블 C

Time	x	y	z2
12:00	11	22	33
12:03			33
12:05	12		34
12:08			34
12:10		24	36
12:13			36
12:15	13	26	39

[0277] 값이 타임아웃으로 인해 생산된 것인지 입력의 변화로 생산된 것인지 관계없이, 각각의 값이 생산되면, 타임아웃 간격이 다시 시작된다. 트리거링을 위해, 타임아웃은 다른 입력으로 취급되고 있다.

표 20

[0278]

```
13.     z3 = (a, b) for a in p5(x), b in y
```

[0279] (13)에서, (9)에서 정의된 오픈 엔드 (open-ended) 패턴 p5 와 관련된 조인을 보여준다. (p5 가 상한없이 1개 이상의 값을 모은다는 것을 상기한다). 다음 테이블은 이러한 새로운 트리거링 조건의 관점에서, x, y, 및 z3 의 임의의 가능한 값들을 보여준다.

표 21

[0280] 테이블 D

Time	x	y	z3
12:00	11		
12:05	12		
12:08	13	0	([11, 12, 13], 0)
12:10		1	
12:13	15		([15], 1)
12:15	16	2	([16], 2)

[0281] 오픈엔드 패턴p5가 그 자체로는어떤 것도 트리거링할 수 없기 때문에 출력생산이 x 에 의해 결코 트리거링되지 않는다는 점에 주목하여야 한다. 하지만, y의 변화가 생산을 트리거링할 수 있으며, 과정중에 적어도 한 개의 값을 수집했다고 가정하면, p5 를 클로즈시킨다. 12시부터 12시08분까지, a 는 x로부터 값을 수집하고 있다. 12시08분에, y의 변화가 수집을 종료시키고 출력이 생산되게 한다. 12시10분에 y가 변화되지만, 수집a가 비어있기 때문에, 출력이 생산되지 않는다. 하지만, 그 후 첫번째 기회에서 (12시13분), a에 대한 값이 제공되고 또 다른 출력이 생산된다. 기타 등등.

[0282] (11), (12), 및 (13)에서, 식들 a in x, b in y, a in p5(x), 및 every 2(min) 은 트리거링 식의 예이다. 이것은 종속식이 평가될 조건을 정의한다.

[0283] 트리거링 식의 값은 뜨겁거나, 따듯하거나, 차갑다이다. 입력 스트림에 연결된 트리거링 식의 경우, 값은 스트림의 생산자의 준비의 반영이다. 시계에 연결된 트리거링 식의 경우, 타임아웃이 경과되면 그 값은 뜨겁다이고, 그렇지 않으면 차갑다이다. (시간 기반 트리거링식은 결코 따뜻하게 되지 않는다.)

[0284] 두 개의 트리거링 식이 논리 합 연산자 (logical or operator) 와 결합되어 새로운 트리거링 식을 생산할 수 있다. 결합된 트리거링식의 새로운 값은 다음 진리표에서 정의된 바와 같이, 좌측 및 우측 피연산자에 의해 결정된다.

표 22

[0285] 테이블 E

Lhs	rhs	out
cold	cold	cold
cold	warm	warm
cold	hot	hot
warm	cold	warm
warm	warm	warm
warm	hot	hot
Hot	cold	hot
Hot	warm	hot
Hot	hot	hot

[0286] 두 개의 트리거링 식이 논리 곱 연산자 (logical and operator)와 결합되어 새로운 트리거링 식을 생산할 수 있다. 결하된 트리거링 식의 새로운 값은 다음 진리표에 나타난바와 같이, 좌측 및 우측 피연산자에 의해 결정된다.

표 23

[0287] 테이블 F

Lhs	rhs	out
cold	cold	cold
cold	warm	cold
cold	hot	cold
warm	cold	cold
warm	warm	warm
warm	hot	hot

Hot	cold	cold
Hot	warm	hot
Hot	hot	hot

[0288] 플로우 그래프의 관점에서, 패턴 규칙의 각 응용은 두 개 이상의 입력을 갖는 변환기를 정의한다. (패턴이 타임아웃을 갖는다면, 변환기도 또한 타임아웃을 가지게 될 것이다). 변환기는 종속 식과 트리거링 식의 정의를 포함한다.

[0289] 조인의 트리거링 식이 뜨겁게 될 경우, 아직 닫히지 않은 모든 입력 패턴은 닫히게 되며, 종속식이 평가된다.

[0290] 조인의 트리거링 식이 따뜻하면, 변환기는 생산자가 가속되기를 요구하는 메시지를 생산자 각각에게 보낸다.

[0291] 가속화된 변환기는 가능한한 빨리 매칭을 클로즈할 것이다. 그것이 이미 따뜻하다면, 뜨겁게 될 것이다. 그것이 차갑다면, 임의의 가능한 중간정도의 따뜻한 상태를 우회하여, 가능한 한 빨리 뜨겁게 될 것이다. 출력을 생산하자마자, 변환기는 자동적으로 감속하여, 다시 가속화될 때까지 정상적인 동작을 재개할 것이다.

[0292] 가속화된 주입기는, 가능하지만, 가혹화되지 않은 주입기와 다르지 않게 행동한다. 다음에 값을 주입할 때 감속된다.

[0293] 가속된 생산자에 의해 생산된 출력이 가속을 요청한 소비자뿐만 아니라, 모든 소비자에게 전송된다는 점에 유의하여야 한다. 이것은 모든 관찰자에게 데이터 흐름의 일관된 관점을 유지한다.

[0294] 생산자를 가속화시키는 소비자의 능력은 본 발명의 주된 특징 중 하나이다. 이것은 패턴구동 반응을 직접적으로 가능하게 한다. 그것이 없으며, 오픈 엔드 패턴은 사용불가능할 것이다.

[0295] 필터링. 조인이 트리거되고 종속 식이 평가된 후, 여전히 출력을 생산하지 않을 수 있다. 선택적인 필터는 바람직하지 않은 결과를 억제할 수 있다.

표 24

[0296]
```
14.     z4 = a + b for a in x, b in y if a + b > 0
```

[0297] (14)에서, 필터의 예를 볼 수 있다. 키워드 "if" 이후의 식은 출력으로서 생산될 종속식의 값에 대한 진리를 평가하여야 한다.

[0298] 필터의 개념은 플로우그래프의 구조에 변화를 유도하지 않는다. 그것은 트리거링된 후에 변환기에 의해 취해지는 추가적인 단계일 뿐이다. 하지만, 활성화 (입력의 도착) 과 트리거링(평가를 위한 전체조건의 만족) 사이에 구분이 없기 때문에, 패턴 매칭이 없이, 변환기의 모든 활성화가 이것과 같은 식의 평가를 요구하는 것은 가치가 없다.

[0299] 식의 평가는 종종 패턴 매칭보다 훨씬 덜 효율적이다. 잘 디자인된 패턴매칭 알고리즘은 매칭단 한번씩만 각 입력을 고려하지만, 식은 조건이 만족되기 전에 각 입력을 여러번 재평가해야 한다. 순수한 식 기반의 변환기는 생산을 하는 것보다, 생산할 지 말지를 결정하는데 더 많은 시간을 소비할 수 있다.

[0300] 가속와와 트리거링 사이의 구분은 본 발명의 주된 특징 중 하나이다. 성능강화를 가능하게 하는 것은 패턴구동되는 접근의 주된 장점 중 하나이다.

[0301] 샘플링. 앞서 진술한 바와 같이, 잘못되지 않은 각 변환기는 일정한 형태의 트리거를 가져야 하며, 패턴 구동되고 결합되는 변환기는 트리거를 선언적으로 정의한다. 샘플링을 채용하는 변환기를 선언하는 것도 또한 가능하다.

표 25

[0302]
```
15.     z5 = a + (latest y) for a in x
```

[0303] (15)를 위해, x 및 y 가 각각 정수 스트림이라고 가정한다. 여기에서 정의된 변환기는 x 에 의해 트리거되지만, y 를 샘플링만 한다. 이것은 x 는 트리거 식에 나타나지만, y 는 종속식에서만 나타나기 때문이다. y 가 정수 스트림이기 때문에, 그 유형은 시퀀스(int) 이며, 전치연산자 (prefix operator) latest 는 식을 평가할 때, 이

시퀀스에서의 맨 앞의 값을 참조한다.

[0304] 플로우그래프의 관점에서, 변환기는 x 로부터 한 개, y 로부터 한 개의 두 개의 입력을 갖는다. 하지만, 입력이 y로부터 도착하면, 값은 간단히 변환기에 의해 캐시에 저장된다. x로부터의 입력의 도착은 y의 캐시값을 사용할 종속식의 평가를 트리거링한다.

[0305] 변환기를 트리거링 하는 입력과 변환기가 단순히 샘플링하는 입력을 사이를 선언적으로 구별하는 능력은 본 발명의 주된 특징 중 하나이다. 변환기의 정의는 부작용과 관련된, 샘플 캐싱처럼, 변환기가 동작을 실행하는 경우라도, 순수하게 선언적으로 존재할 수 있게 한다.

[0306] 기능적-논리적 최적화. (종속 및 필터링 식의) 기능적 프로그래밍과 결합된 (패턴 매칭과 결합의) 논리 프로그래밍의 조합은 순수하게 선언적인 방식으로 광범위한 범위의 변환을 표현할 수 있다는 것을 지금까지 알게 되었다. 복합적인, 기능적-논리적 프로그램은 기능적 프로그래밍의 많은 장점을 가지며, 그 많은 제한과 복잡성을 완화시킨다. 명령형 프로그램을 재분류하지 않고, 대부분의 실제 의도를 표현할 수 있다.

[0307] 기능적 논리적 프로그램의 기능식은 기능적 프로그램이 최적화되는 것과 동일한 방식으로 최적화될 수 있다. 이러한 종류의 최적화는 생산자와 소비자의 긴 사슬을 한 개의 변환기로 융합하여, 그래프를 동시에 작고 빠르게 실행시킬 수 있다.

[0308] 또한, 규칙이라고 표현된 논리패턴은 통일된 상태 머신으로 결합될 수 있다. 이러한 결합의 정확한 속성은 사용되는 매칭알고리즘에 의존한다. 예를 들면, TFR(1)의 경우에, 한 개의 소스에 적용되는 모든 패턴이 결합되어 한 개의 상태 머신을 형성할 수 있으며, 그 스트림으로의 모든 입력이 단 한번만 평가될 것을 보증한다.

[0309] 최적화를 위한 넓은 범위의 기회를 보유하면서, 데이터플로우 프로그래밍이 채용하는 대부분의 유용한 프로그램을 표현하기 위한 기능적-논리적 선언의 능력은 본 발명의 주된 특징의 하나이다. 간결하게 선언된 프로그램은 더 작고 더 빠른 플로우그래프를 가져온다.

[0310] 일 구현에 있어서, 시스템은 각각 센서에 연결되며, 센서로부터 데이터(예를 들면, 스트림 데이터)를 수신하는 다수의 에이전트 및 데이터버스를 포함하며, 에이전트에 연결된 데이터 처리 구성요소를 포함하며, 데이터 처리 구성요소는 서로 연결된 일련의 변환기를 형성함으로서, 입력의 스트림을 출력의 스트림으로 변환한다. 각각의 변환은 기준으로서, 적어도 한 개의 외부 데이터 소스 또는 변환기를 가지는 한 개 이상의 입력, 기준으로서 외부 데이터 싱크 또는 변환기 중 적어도 하나를 가지는, 한 개 이상의 출력, 및 각각의 입력에 대하여, 입력에 적용되고, 원래 입력을 필터링하고, 수집하고, 더 유용한, 부분적으로 처리된 형태로 조직하고, 문제가 있는 입력을 제거하기 위한 패턴을 포함한다.

[0311] 적어도 한 개의 에이전트는, 제1 메모리 위치에, 처리될 토큰의 선입선출 시퀀스를 저장하고, 큐에서 각 토큰과 관련된 시간 스탬프는 관련 토큰이 입력큐로 들어가는 시간을 나타내며, 토큰은 네트워크를 통해 입력큐에 의해 수신되는, 컴퓨터 메모리 내의 제1 메모리 위치를 포함하는 입력큐, 입력큐에 접속되며, 역추적 없이 입력큐에서 토큰을 처리하며, 한 개 이상의 기설정된 입력 패턴과 일치하는 토큰의 시퀀스에서의 패턴을 식별하며, 일치되는 입력 패턴을 식별시에, 출력되는 이벤트 출력을 생성하는 드라이버 구성요소, 드라이버 구성요소에 접속되며 컴퓨터 메모리내에 제2 메모리 위치를 가지며, 드라이버 구성요소에 의해 생성된 출력되는 이벤트의 선입선출 시퀀스를 제2 메모리 위치에 저장하는 출력큐, 드라이버 구성요소에 접속되며, 상태 테이블 포맷으로 기설정된 입력 패턴을 저장하는 상태 테이블 구성요소, 드라이버 구성요소에 접속되며, 컴퓨터메모리의 제3 메모리 위치를 가지며, 제3 메모리 위치에 프레임의 후입선출 시퀀스 스토리지를 저장하는 상태 스택 구성요소를 포함한다. 프레임은 번역상태 번호, 기호, 및 데드라인을 포함한다.

[0312] 각 변환은 필터링 식을 평가할 때 판단하기 위하여 사용되는 트리거링 식을 포함할 수 있다. 트리거링 식은 노드들의 트리로서 정의되며, 각 노드는 입력중 한 개에 대한 기준, 시간 단위로, 시간 간격으로서 주어지는 타임아웃, 각각은 상이한 트리거링 식 노드로 표현되는 적어도 두 개의 자손을 포함하는 2진 공접 연산자 (binary conjunction operator), 및 각각 또 다른 트리거링 식 노드에 의해 표현되는 적어도 두 개의 자손을 포함하는 2진 이접연산자 (binary disjunction operator) 중 적어도 하나를 가진다.

[0313] 각 변환은 필터링 식을 포함할 수 있으며, 필터링 식은 매칭된 입력의 도메인에서 지정되고, 필터링 식은 변환이 출력을 생산할 때를 결정하는 불린(Boolean)결과를 산출한다. 입력패턴은 0인 길이 또는 무제한 (또는 무한정 긴) 길이를 포함하는 가변 길이의 입력 시퀀스를 매칭시킬 수 있다.

[0314] 입력패턴을 입력 시퀀스에 매칭시키고자 하는 시도의 결과는 적어도 세 개의 상이한 상태 또는 완벽도를 가질

수 있다. 입력패턴을 입력 시퀀스에 매칭시키고자 하는 시도의 결과는 적어도 세 개의 상이한 상태를 가질 수 있다. 세 개의 상이한 상태로는 입력 패턴의 최소한의 입력이 매칭될 때까지, 매칭을 위한 시도가 첫번째 상태이며, 최소한의 입력이 매칭된 후 및 최대한의 입력이 매칭되기 전, 매칭을 위한 시도가 두번째 상태이며, 최대한의 입력이 매칭된 후, 매칭을 위한 시도가 세번째 상태이다. 첫번째 상태는 "차가운" 상태라고 명명할 수 있다. 두번째 상태는 "따뜻한" 상태라고 명명할 수 있다. 세번째 상태는 "뜨거운" 상태라고 명명할 수 있다.

[0315] 데이터플로우는 센서로부터 에이전트까지의 푸시 기반일 수 있다. 데이터플로우는 에이전트의 모든 입력이 적어도 두번째 상태에 있으며, 센서의 한 개 이상의 입력이 세번째 상태에 있을 경우, 센서로부터 에이전트에 의해 폴 기반으로 변화될 수 있으며, 트리거링 식의 결과는 두번째 상태에 있다. 에이전트는 두번째 상태에 있는 센서를 가속화하여, 그러한 센서가 패턴이 완전히 매칭되었다고 간주하여 세번째 상태가 되게 하며, 그 출력을 생산하도록 강제한다. 이것은 에이전트가 그 센서로부터 데이터는 효과적으로 끌어당기도록(pull) 한다.

[0316] 명확히 길거나, 오픈 엔트 패턴이 변환에 유용하게 구현될 수 있다. 한 개 이상의 센서는 물리적 속성을 디지털 양으로 변환시키는 하드웨어 장치일 수 있다. 디지털 양은 스트림된다.

[0317] 또 다른 구현에 있어서, 방법은 다수의 에이전트와 센서를 서로 연결하는 단계로서, 에이전트는 센서에 연결되며, 센서로부터 데이터의 스트림을 수신하는 단계, 및 센서로부터의 입력 스트림을 일련의 접속된 변환을 통해 출력 스트림으로 변환하는 단계를 포함한다. 각각의 변환은 각각은 기준으로서, 외부 데이터 소스 또는 변환기에 주어지는 가지는 한 개 이상의 입력, 기준으로서 외부 데이터 싱크 또는 변환기 중 적어도 하나에 주어지는, 한 개 이상의 출력, 각각의 입력에 대하여, 입력에 적용되고, 원래 입력을 필터링하고, 수집하고, 더 유용하고, 부분적으로 처리되며 매칭된 형태로 조직하고, 문제가 있는 입력을 제거하기 위한 패턴; 및 필터링 식을 평가할 때를 판단하기 위하여 사용되고, 노드들의 트리로서 정의되는 트리거링 식을 포함한다.

[0318] 각각의 노드는 입력중 한 개에 대한 기준, 편리한 시간 단위로 특징지어지는 시간 간격으로서 주어지는 타임아웃, 각각이 상이한 트리거링 식 노드로 표현되는 적어도 두 개의 자손을 포함하는 2진 공접 연산자 (binary conjunction operator), 및 각각 또 다른 트리거링 식 노드에 의해 표현되는 적어도 두 개의 자손을 포함하는 2진 이접연산자 (binary disjunction operator) 중 적어도 하나를 가진다. 매칭된 입력의 도메인에서 정의되고 변환이 출력을 생산할 때를 결정하는 불린(Boolean)결과를 산출하는 필터링 식이 존재한다. 매칭된 입력의 도메인에서 정의되고 변환의 출력을 생산하는 유형의 임의의 조합의 결과를 산출하는 한 개 이상의 식이 존재한다.

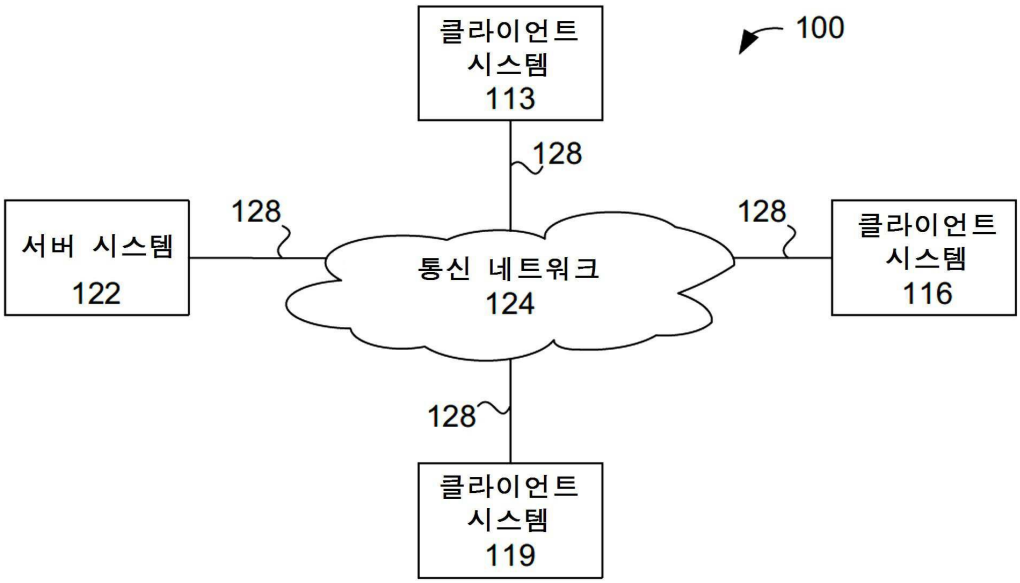
[0319] 입력 패턴은 길이가 0 또는 무제한의 길이를 포함하는 가변길이의 입력 시퀀스를 매칭시킬 수 있으며, 최소한의 입력이 매칭될 때까지는 차가운 상태, 최소한의 입력이 매칭된 후, 최대한의 입력이 매칭되기 전은 따뜻한 상태, 최대한의 입력이 매칭된 후는 뜨거운 상태라고 간주된다. 매칭을 위한 변환기의 준비상태의 완전성 모델은 두 개 이하의 상이한 정도 보다는 적어도 세 개의 상이한 정도에 의해 특징지워진다. 세 가지 정도의 완전성 모델은 패턴 구동형 반응이 가변 길이 패턴을 조절하는 것을 가능하도록 한다.

[0320] 생산자로부터 소비자로의 데이터플로우는 에이전트의 모든 입력이 적어도 따뜻한 상태에 있으며, 센서의 한 개 이상의 입력이 뜨거운 상태에 있을 경우를 제외하고, 푸시기반이며, 트리거링 식의 결과는 따뜻한 상태에 있다. 에이전트는 따뜻한 상태에 있는 센서를 가속화하여, 그러한 센서가 패턴이 완전히 매칭되었다고 간주하여 뜨거운 상태가 되게 하며, 그 출력을 생산하도록 강제한다. 이것은 센서가 거기에 연결된 다른 생산자에게 데이터를 끌어당기도록 한다. 이것은 명확히 길거나, 오픈 엔드 패턴이 변환에 유용하게 채용하도록 한다.

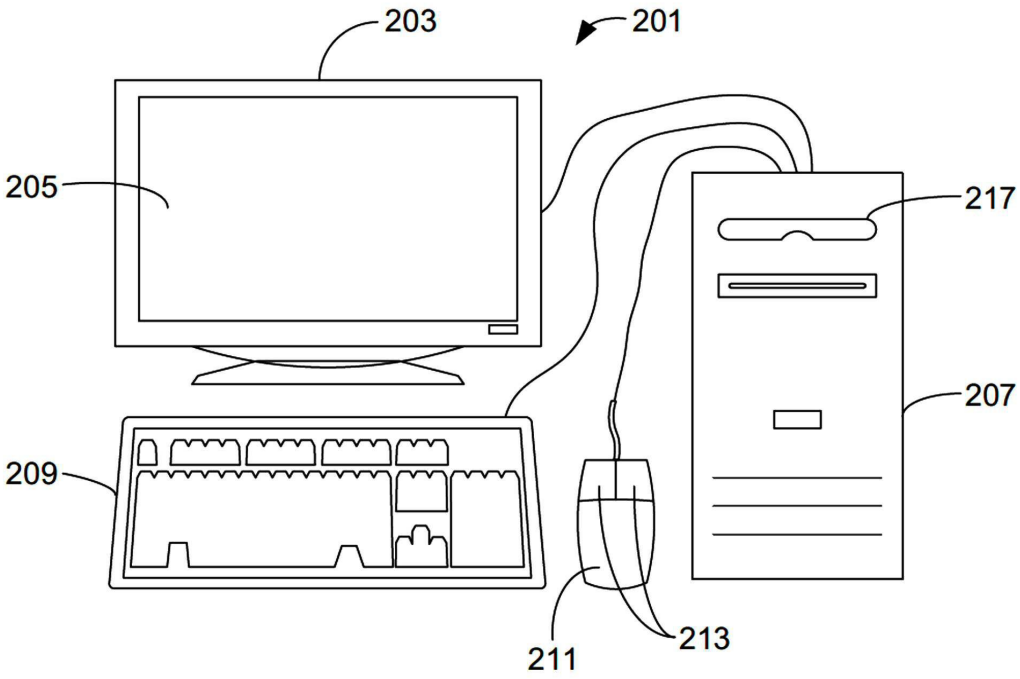
[0321] 본 발명의 기재는 예시 및 설명을 위해 제시되었다. 본 발명을 상술한 정확한 형태로 한정하고자 하는 의도는 아니며 다양한 수정과 변형이 상기 교시에 비추어볼 때 가능하다. 실시예는 본 발명의 원리 및 그 실용적인 응용을 가장 잘 설명하기 위하여 선택되고 기술되었다. 본 기재는 당업자가 특정한 용도에 적합한 다양한 변형과 함께 다양한 실시예로 본 발명을 가장 잘 활용하고 실시할 수 있게 할 것이다. 본 발명의 범위는 이하의 청구범위에 의해 정의된다.

도면

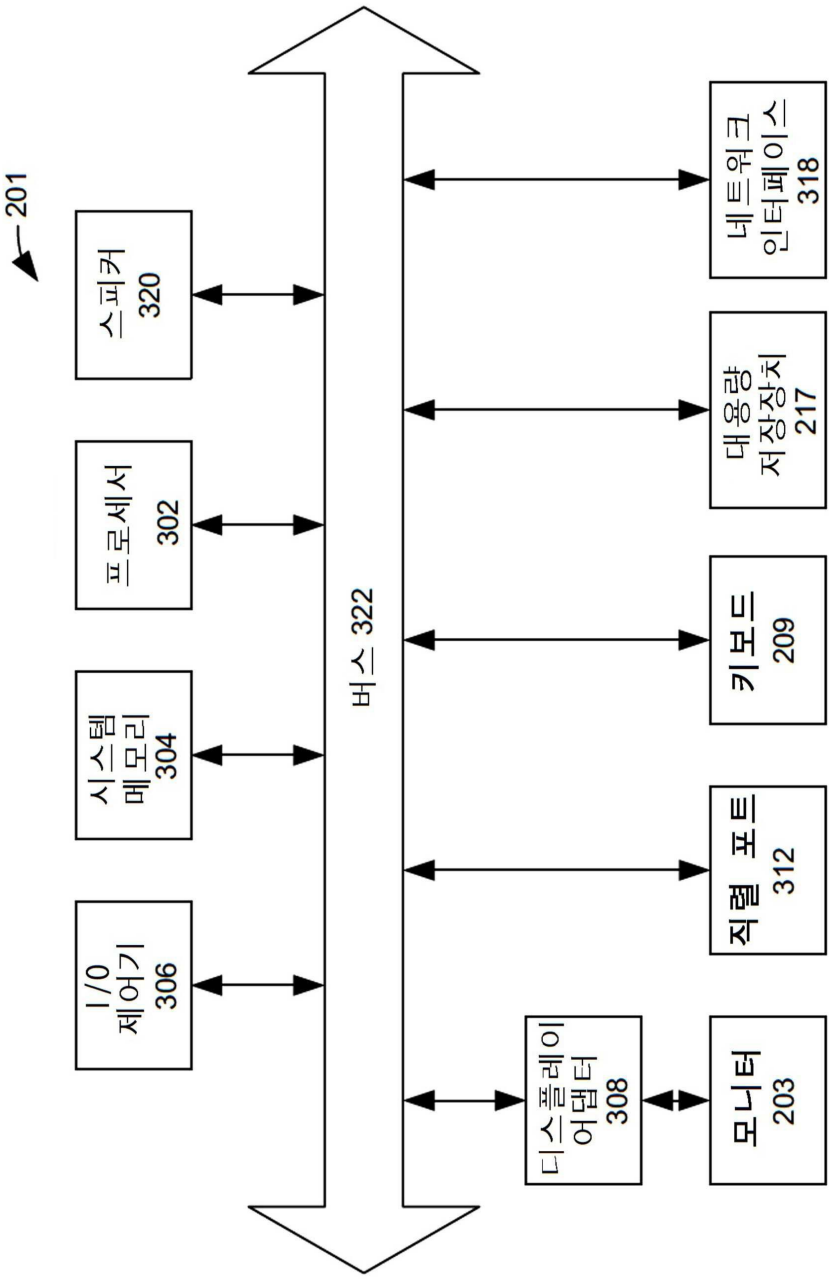
도면1


클라이언트
시스템
113
100
128
서버 시스템
122
128
통신 네트워크
124
128
클라이언트
시스템
116
128
클라이언트
시스템
119

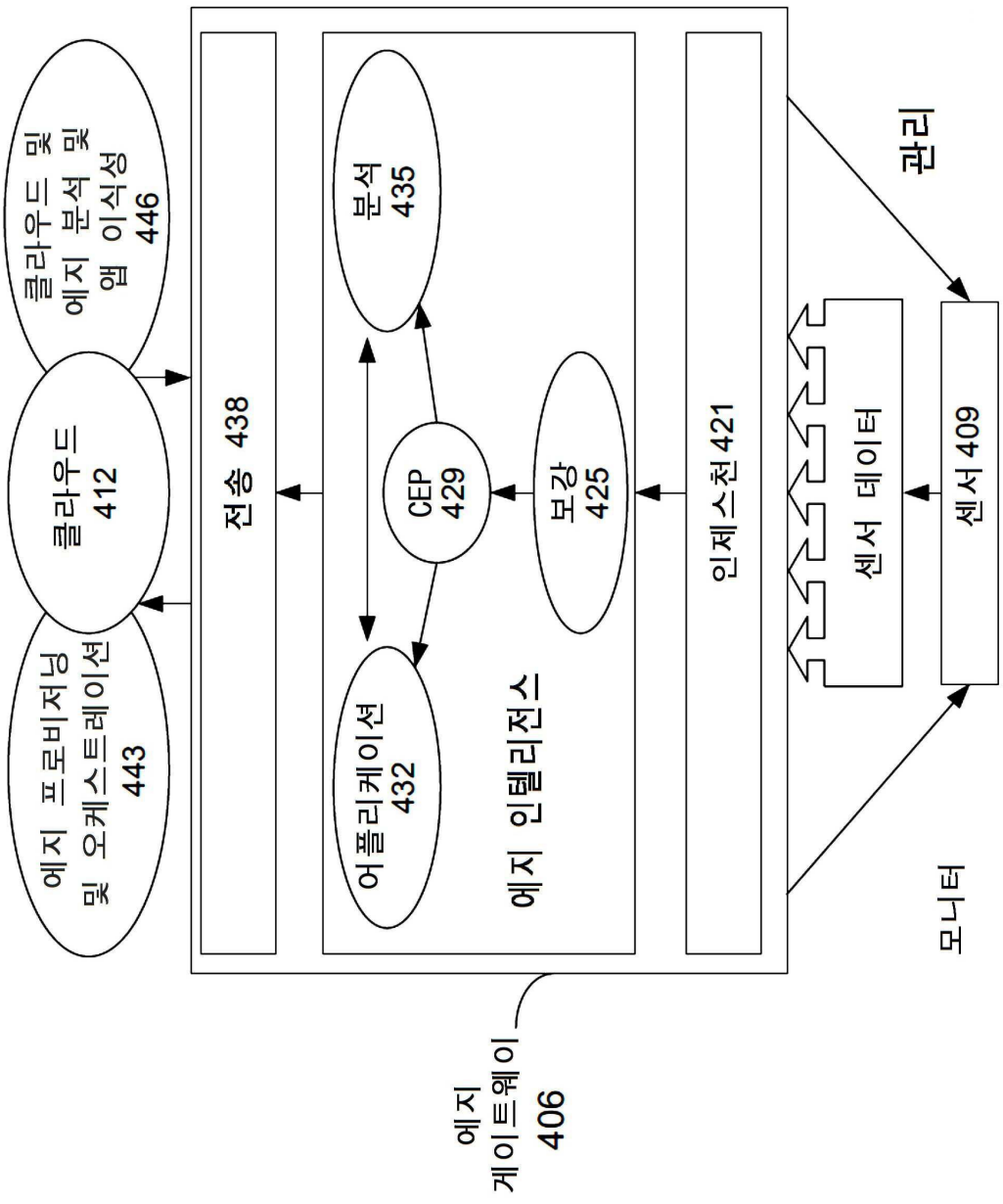
도면2


203
201
205
217
207
209
211
213

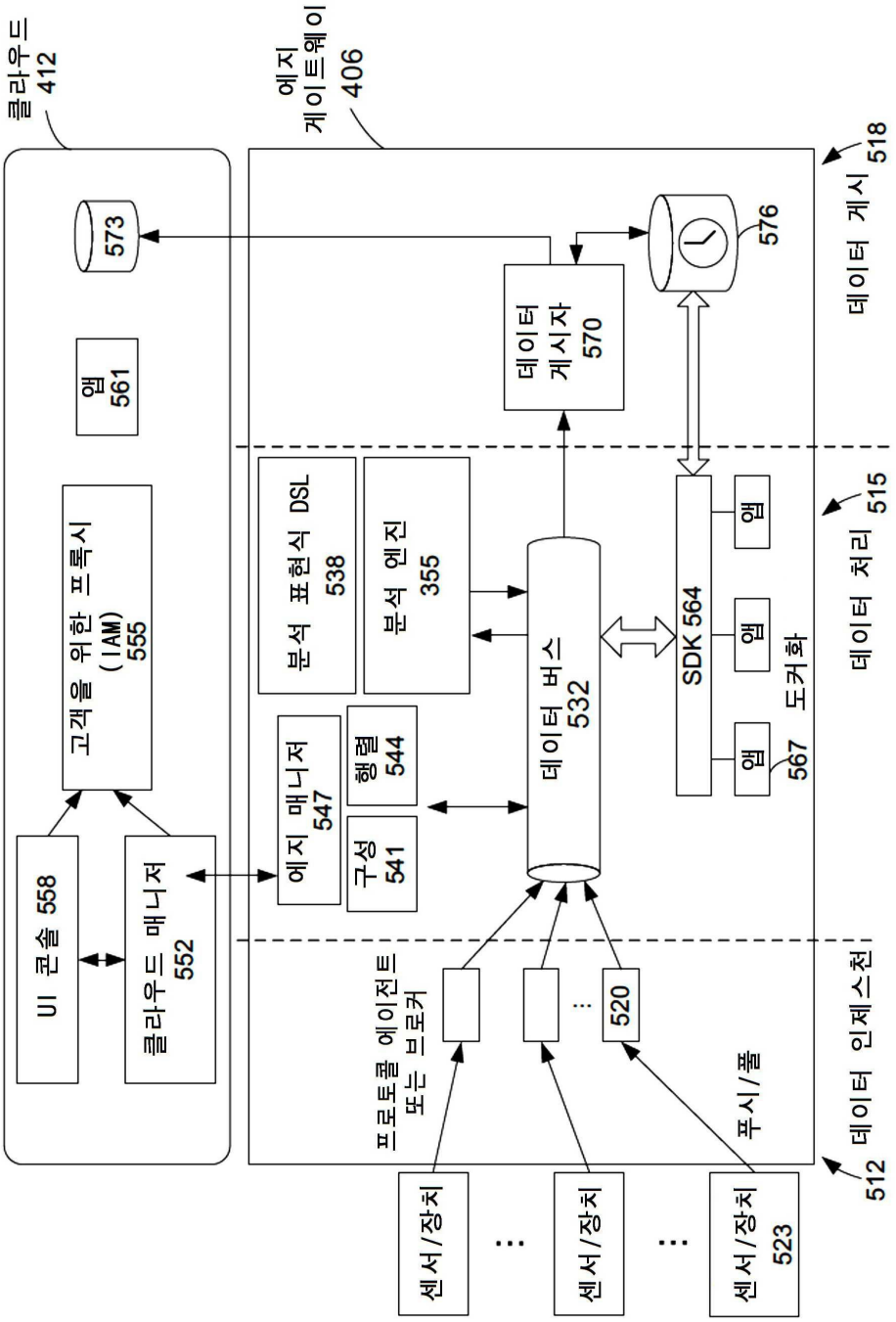
도면3


201
스피커 320
프로세서 302
시스템 메모리 304
I/O 제어기 306
버스 322
네트워크 인터페이스 318
대용량 저장장치 217
키보드 209
직렬 포트 312
디스플레이 어댑터 308
모니터 203

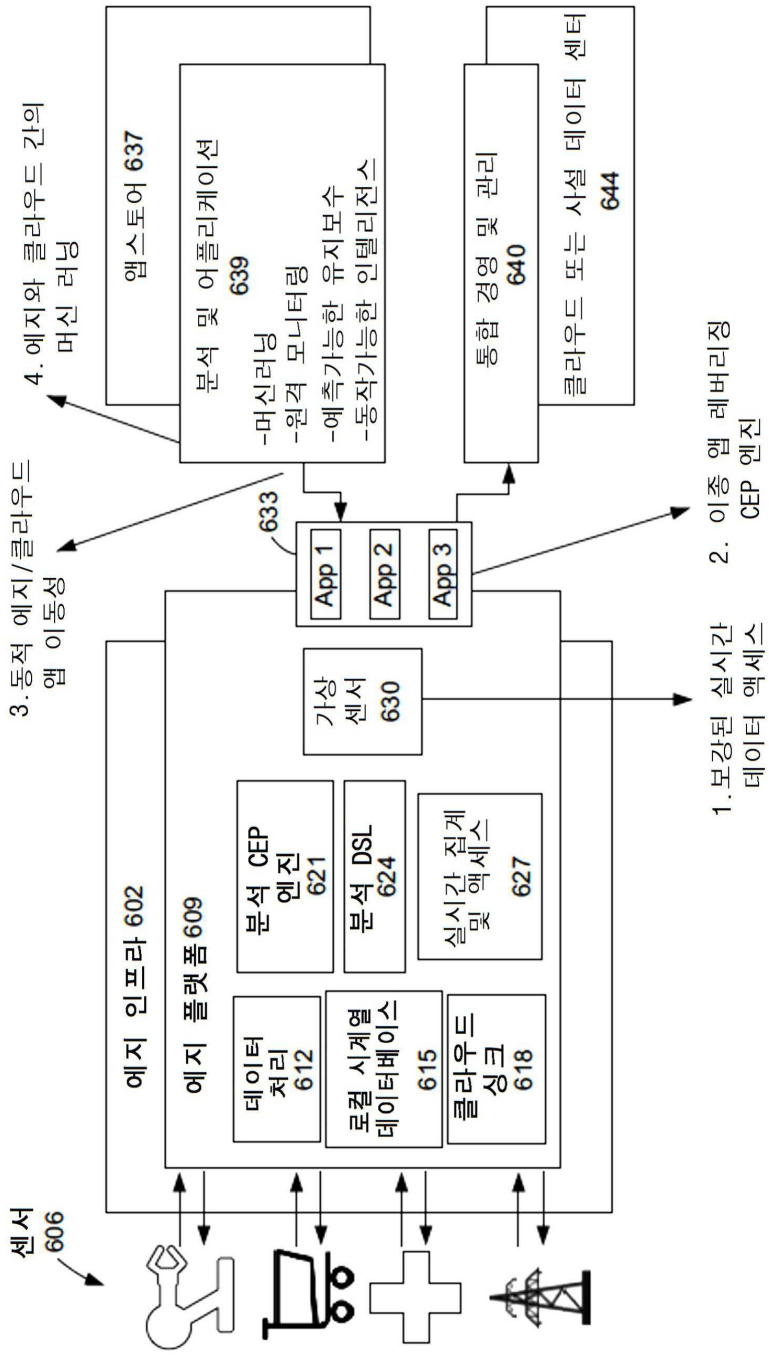
도면4


클라우드 및
에지 분석 및
앱 이식성
446
클라우드
412
에지 프로비저닝
및 오케스트레이션
443
전송 438
분석
435
CEP
429
어플리케이션
432
에지 인텔리전스
보강
425
인제스천 421
에지
게이트웨이
406
관리
센서 데이터
센서 409
모니터

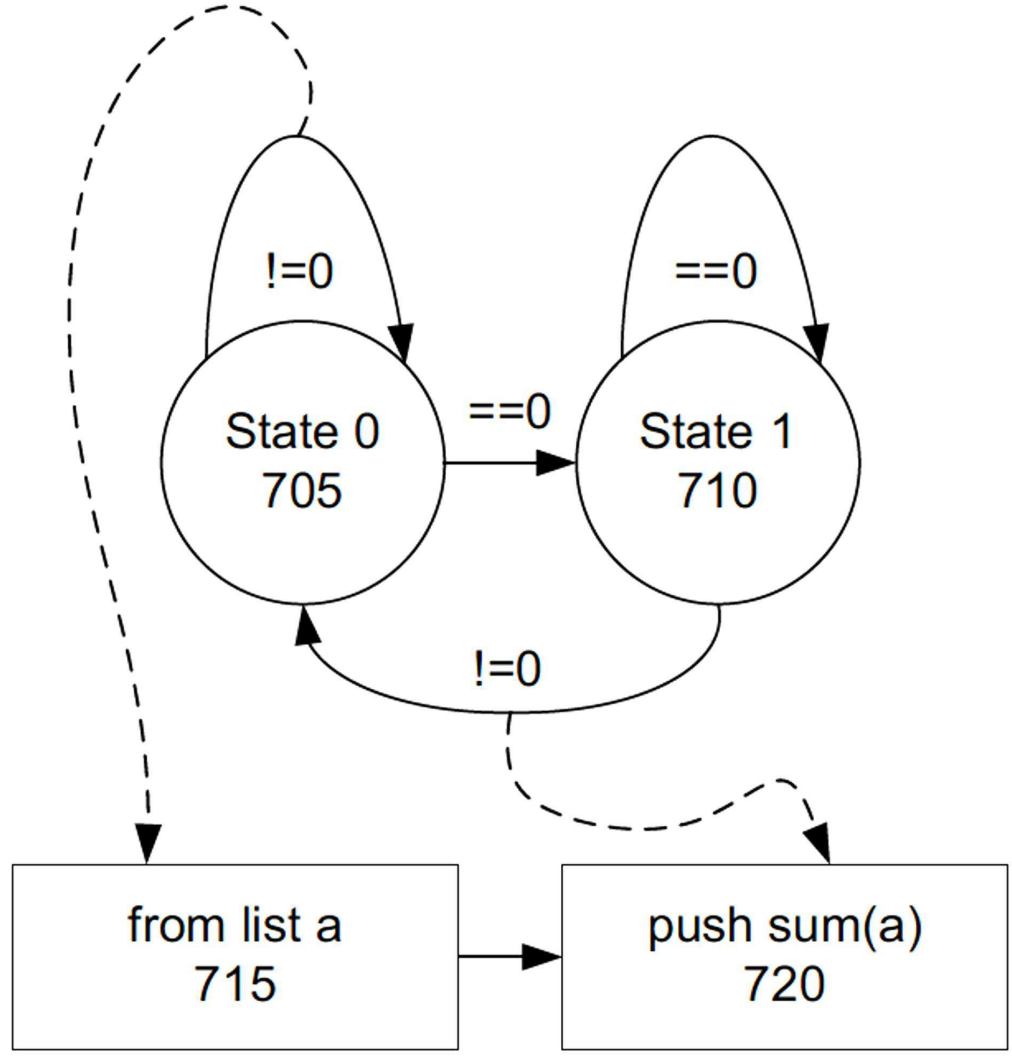
도면5


클라우드
412
에지
게이트웨이
406
데이터 게시
518
데이터 처리
515
데이터 인제스천
512
573
앱
561
고객을 위한 프록시
(IAM)
555
UI 콘솔 558
클라우드 매니저
552
에지 매니저
547
구성
541
행렬
544
분석 표현식 DSL
538
분석 엔진
355
데이터 버스
532
데이터
게시자
570
576
SDK 564
앱
앱
앱
567
도커화
프로토콜 에이전트
또는 브로커
520
푸시/풀
센서/장치
센서/장치
센서/장치
523

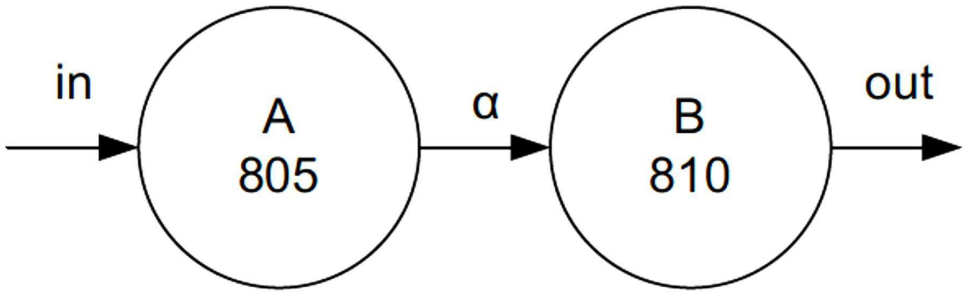
도면6


센서
606
에지 인프라 602
에지 플랫폼 609
데이터 처리 612
로컬 시계열 데이터베이스 615
클라우드 싱크 618
분석 CEP 엔진 621
분석 DSL 624
실시간 집계 및 액세스 627
가상 센서 630
633
App 1
App 2
App 3
앱 스토어 637
분석 및 어플리케이션 639
-머신러닝
-원격 모니터링
-예측가능한 유지보수
-동작가능한 인텔리전스
통합 경영 및 관리 640
클라우드 또는 사설 데이터 센터 644
1. 보강된 실시간 데이터 액세스
2. 이종 앱 레버리징 CEP 엔진
3. 동적 에지/클라우드 앱 이동성
4. 에지와 클라우드 간의 머신 러닝

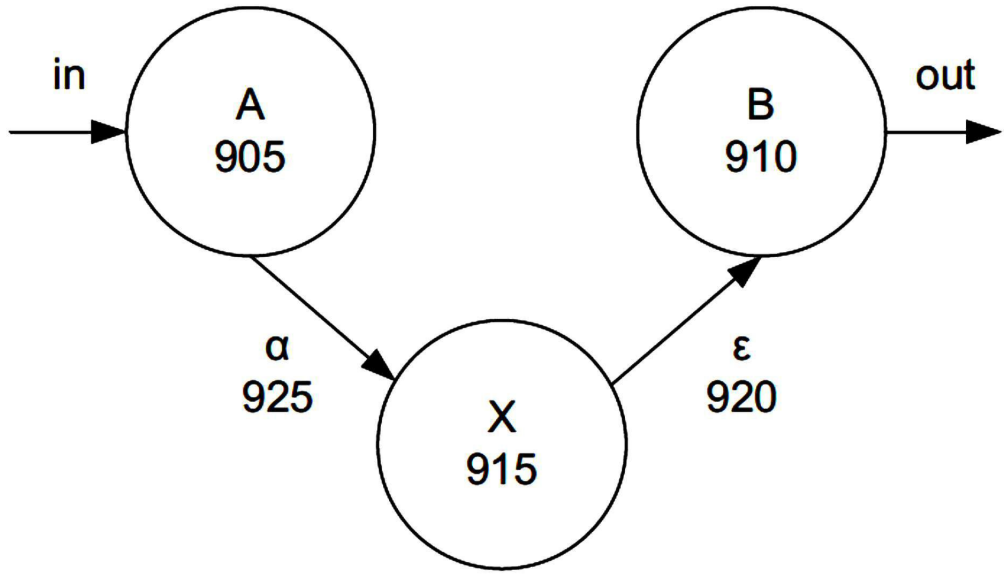
도면7


!=0
==0
State 0
705
==0
State 1
710
!=0
from list a
715
push sum(a)
720

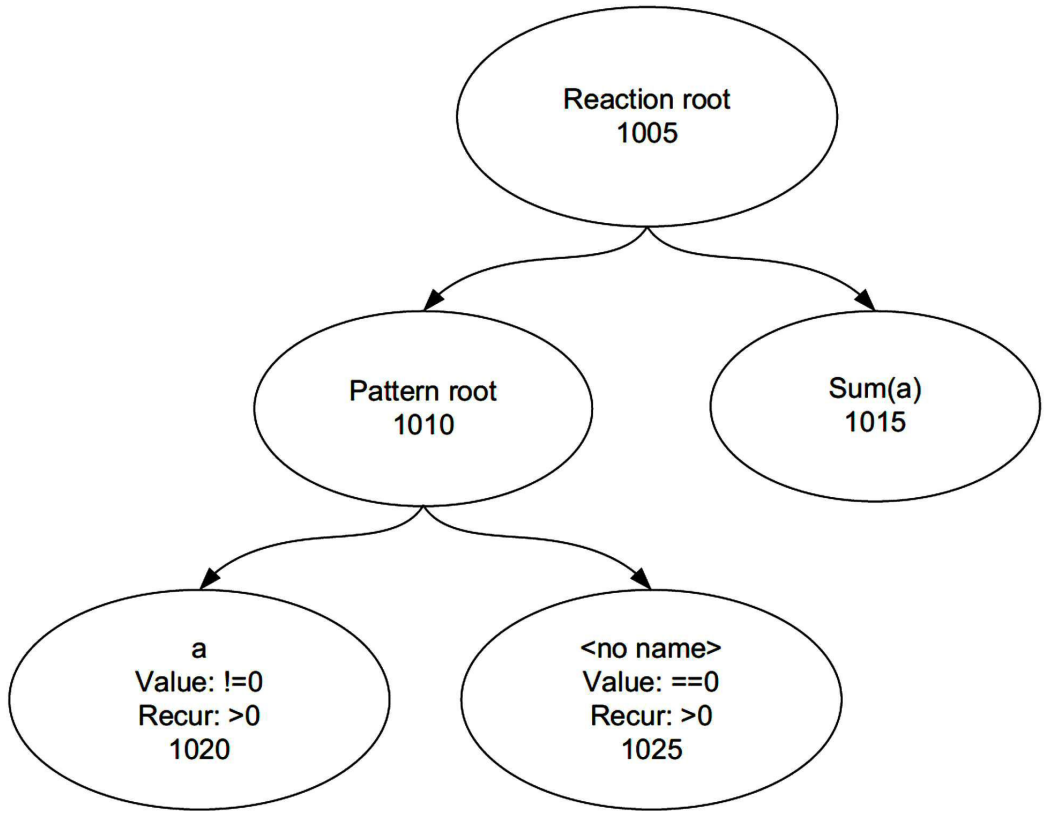
도면8


in
A
805
α
B
810
out

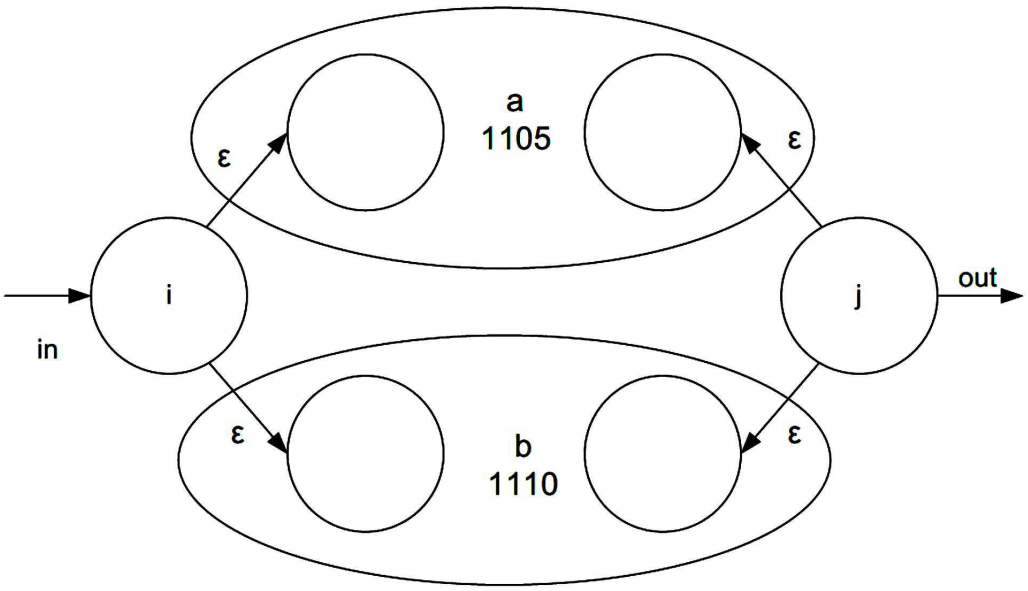
도면9


in
A
905
B
910
out
α
925
X
915
ε
920

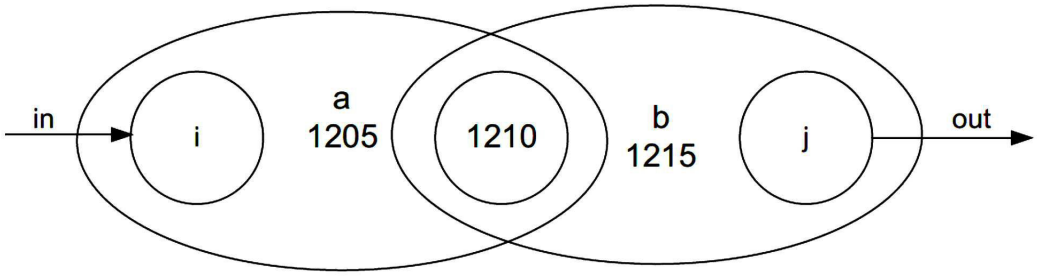
도면10


Reaction root
1005
Pattern root
1010
Sum(a)
1015
a
Value: !=0
Recur: >0
1020
<no name>
Value: ==0
Recur: >0
1025

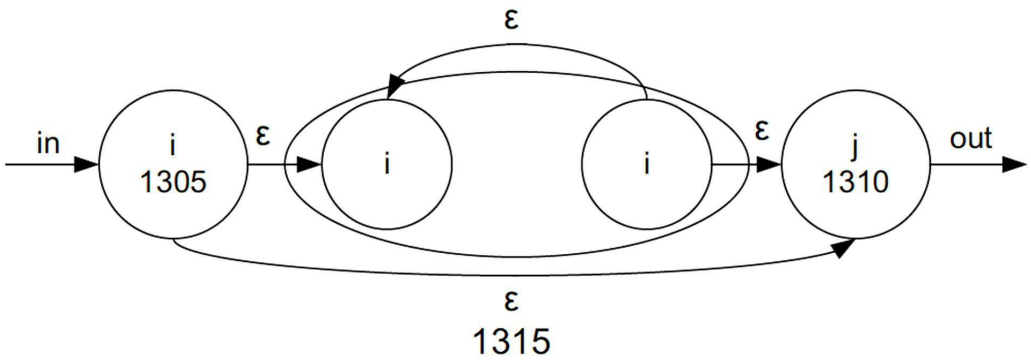
도면11



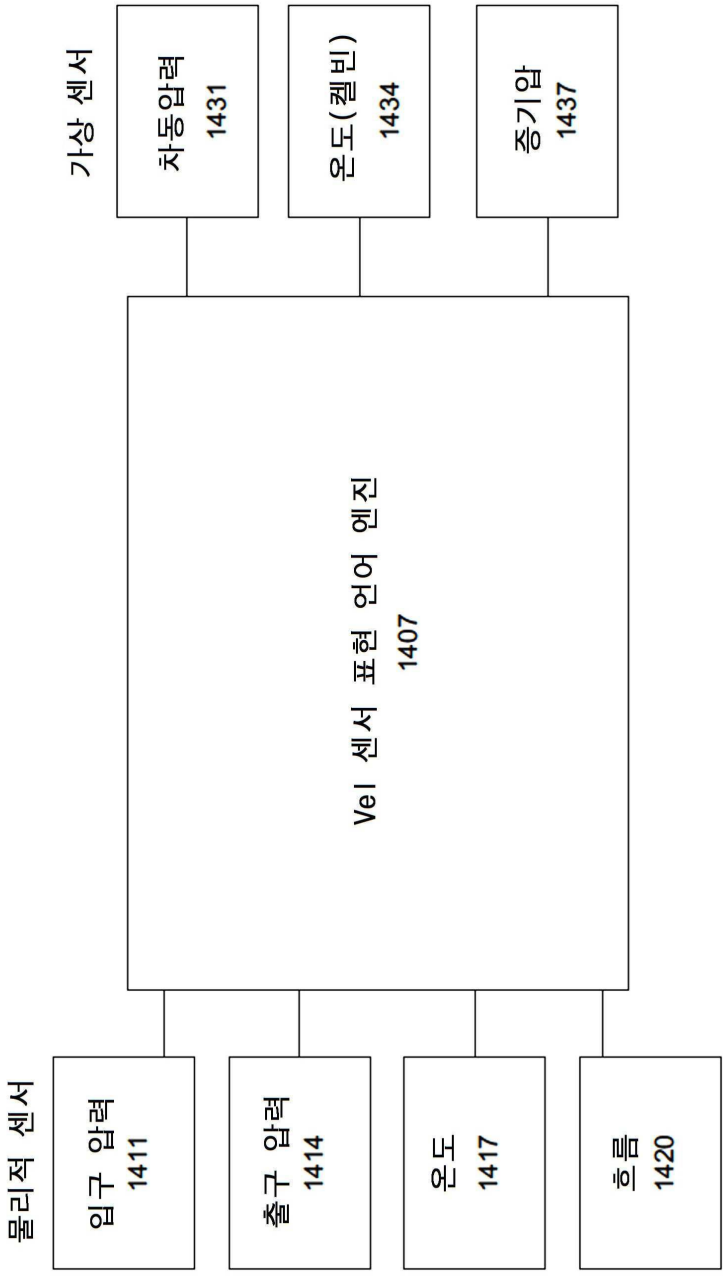
도면12



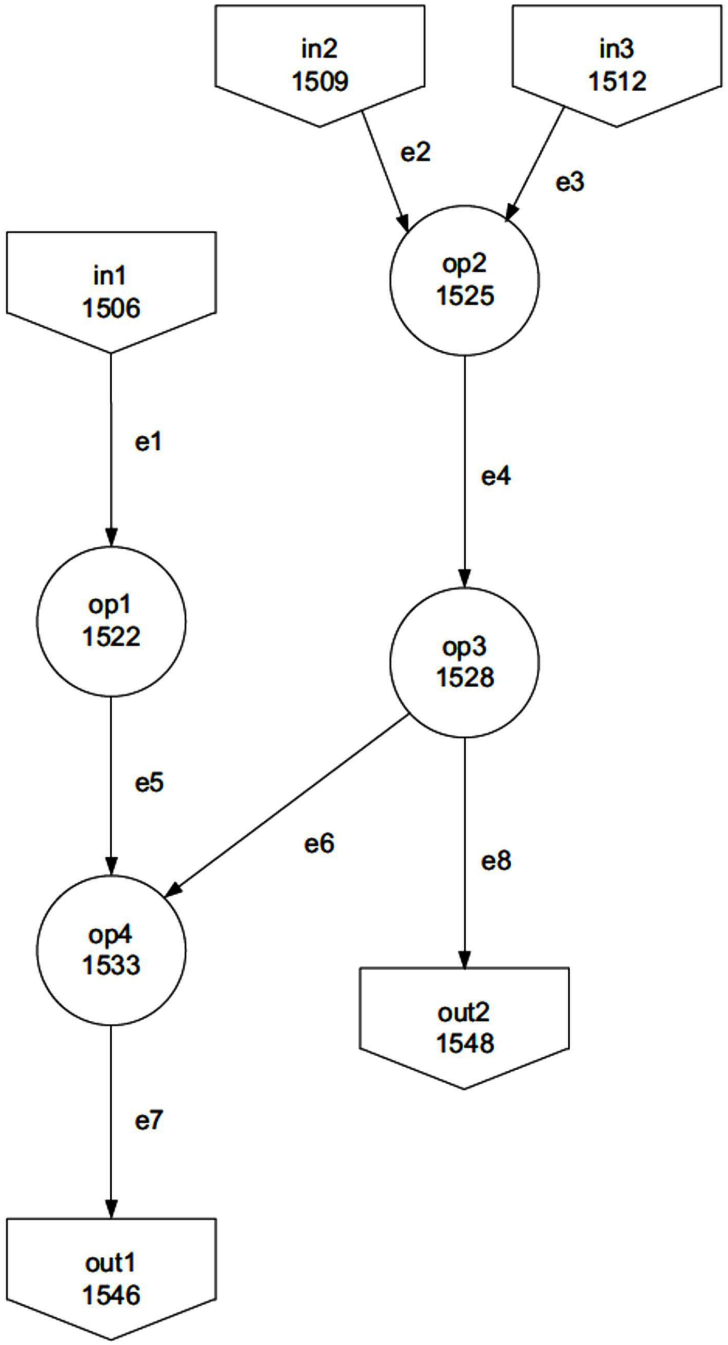
도면13



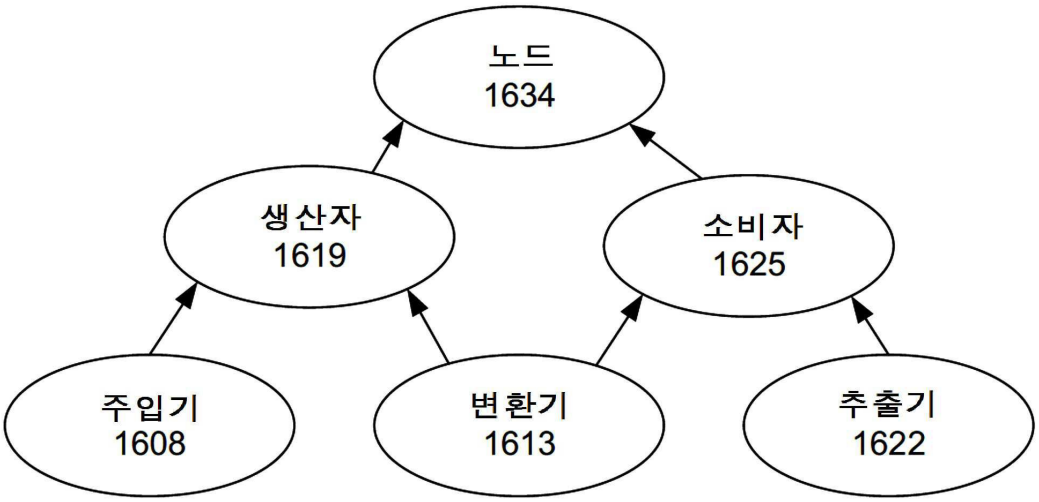
도면14


가상 센서
차동압력
1431
온도(켈빈)
1434
증기압
1437
VeI 센서 표현 언어 엔진
1407
물리적 센서
입구 압력
1411
출구 압력
1414
온도
1417
흐름
1420

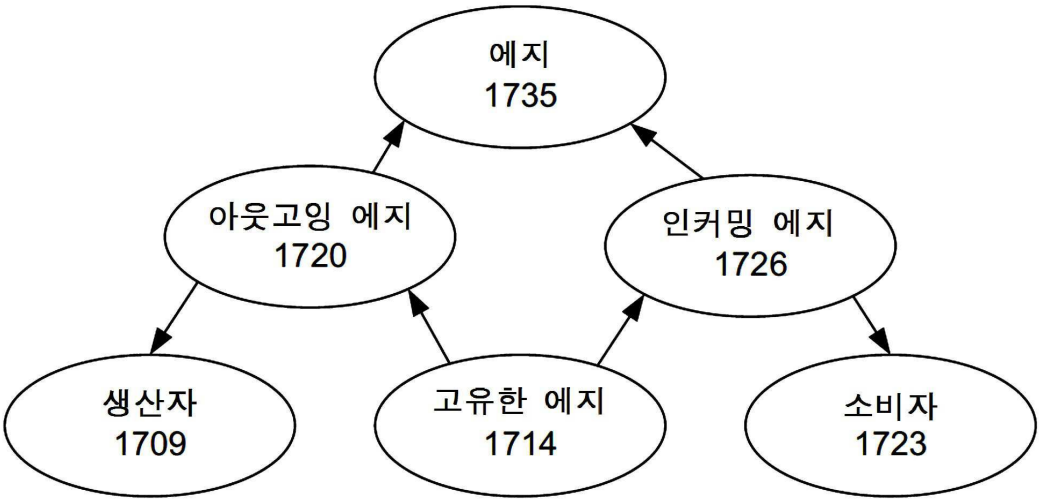
도면15


in2
1509
in3
1512
e2
e3
in1
1506
op2
1525
e1
e4
op1
1522
op3
1528
e5
e6
e8
op4
1533
out2
1548
e7
out1
1546

도면16



도면17



【심사관 직권보정사항】

【직권보정 1】

【보정항목】 청구범위

【보정세부항목】 제2항

【변경전】

상기 상태 테이블 포맷으로

【변경후】

상태 테이블 포맷으로