(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2015년08월03일
(11) 등록번호 10-1541664
(24) 등록일자 2015년07월28일

(51) 국제특허분류(Int. Cl.)
*G06F 9/46* (2006.01)
(21) 출원번호 10-2014-7008357
(22) 출원일자(국제) 2012년08월16일
심사청구일자 2014년03월28일
(85) 번역문제출일자 2014년03월28일
(65) 공개번호 10-2014-0074319
(43) 공개일자 2014년06월17일
(86) 국제출원번호 PCT/US2012/051203
(87) 국제공개번호 WO 2013/032727
국제공개일자 2013년03월07일
(30) 우선권주장
13/224,198 2011년09월01일 미국(US)
(56) 선행기술조사문헌
JP평성07073035 A
US6253252 B1
US20050268302 A1
JP2007199811 A

(73) 특허권자
**퀄컴 인코포레이티드**
미국 92121-1714 캘리포니아주 샌 디에고 모어하우스 드라이브 5775
(72) 발명자
**가르가쉬 노먼 에스**
미국 92121 캘리포니아주 샌디에고 모어하우스 드라이브 5775
**비자야라잔 비노드**
미국 92121 캘리포니아주 샌디에고 모어하우스 드라이브 5775
(74) 대리인
**특허법인코리아나**

전체 청구항 수 : 총 44 항 심사관 : 유진태

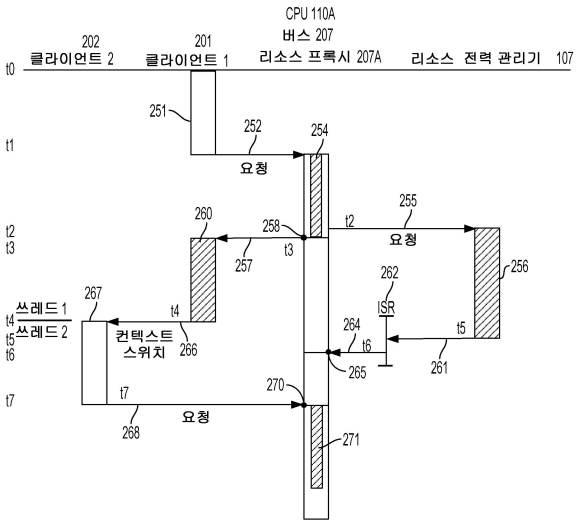
(54) 발명의 명칭 **휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법 및 시스템**

## (57) 요 약

휴대형 컴퓨팅 디바이스 ("PCD") 에서 병렬 리소스 요청들을 관리하는 방법 및 시스템이 설명된다. 본 시스템 및 방법은 제 1 클라이언트로부터, 제 1 실행 쓰레드의 컨텍스트에서 발해지는 제 1 요청 (252) 을 발생시키는 것을 포함한다. 제 1 요청은 리소스 (255) 로 포워딩될 수도 있다. 리소스는 제 1 요청을 확인응답하

(뒷면에 계속)

*대 표 도* - 도5



고 비동기 프로세싱을 개시할 수도 있다. 리소스는 제 1 클라이언트로 하여금 제 1 실행 쓰레드에서 계속해서 프로세싱하도록 (260) 하면서, 제 1 요청 (256) 을 프로세싱할 수도 있다. 리소스는 제 1 요청 (264) 의 프로세싱의 완료를 시그널링할 수도 있으며, 제 2 요청 (268) 을 수신할 수도 있다. 제 2 요청은 제 1 요청 (270) 의 프로세싱의 완료를 초래한다. 제 1 요청의 프로세싱의 완료는 리소스의 로컬 표현을 새로운 상태로 업데이트하는 것, 및 임의의 등록된 콜백들을 호출하는 것을 포함할 수도 있다. 리소스는 제 2 요청을 서비스하는데 이용가능하게 될 수도 있으며, 제 2 요청을 프로세싱할 수도 있다.

# 명 세 서

## 청구범위

### 청구항 1

휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법으로서,

제 1 클라이언트로부터 제 1 요청을 발생시키는 단계로서, 상기 제 1 요청은 제 1 실행 쓰레드의 컨텍스트에서 발해지는, 상기 제 1 요청을 발생시키는 단계;

상기 제 1 요청을 리소스로 포워딩하는 단계;

상기 제 1 클라이언트가 상기 리소스로 하여금 비동기 프로세싱을 허용할지 여부를 결정하도록 허용한다는 것을 상기 리소스에게 통지하는 디폴트 선호사항을 상기 제 1 클라이언트에 의해 패스하는 (pass in) 단계를 포함하고,

상기 비동기 프로세싱은,

상기 리소스에 의해 상기 제 1 요청을 확인응답하고, 상기 리소스가 상기 비동기 프로세싱을 허용하도록 결정하는 경우 상기 비동기 프로세싱을 개시하는 단계;

상기 제 1 클라이언트로 하여금 상기 제 1 실행 쓰레드에서 계속해서 프로세싱하는 것을 허용하면서 상기 리소스에서 상기 제 1 요청을 프로세싱하는 단계;

상기 리소스에 의해, 상기 제 1 요청의 프로세싱의 완료를 시그널링하는 단계로서, 상기 제 1 요청의 프로세싱의 완료는 상기 리소스의 로컬 표현을 새로운 상태로 업데이트하는 것 및 임의의 등록된 콜백들을 호출하는 것 (invoking) 을 포함하며, 상기 리소스는 제 2 요청을 서비스하는데 이용가능하게 되는, 상기 제 1 요청의 프로세싱의 완료를 시그널링하는 단계;

상기 리소스에서 상기 제 2 요청을 수신하는 단계; 및

상기 제 2 요청을 상기 리소스에서 프로세싱하는 단계를 포함하는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법.

### 청구항 2

제 1 항에 있어서,

상기 제 2 요청은 상기 제 1 클라이언트에서 발신되는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법.

### 청구항 3

제 1 항에 있어서,

상기 제 2 요청은 제 2 클라이언트에서 발신되는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법.

### 청구항 4

제 1 항에 있어서,

상기 제 1 요청은 리소스 프록시를 통해서 상기 리소스로 포워딩되는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법.

### 청구항 5

제 1 항에 있어서,

상기 제 1 요청을 확인응답하는 것은, 상기 제 1 클라이언트로 하여금, 상기 리소스가 상기 제 1 요청을 프로세

싱하기 전에 상기 제 1 실행 쓰레드에서 계속해서 프로세싱하도록 허용하는 단계를 더 포함하는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법.

**청구항 6**

제 1 항에 있어서,

상기 리소스에 의해, 상기 제 1 요청의 프로세싱의 완료를 시그널링하는 단계는, 상기 리소스를 결합가능하게 (joinable) 하며,

상기 리소스가 결합가능하다는 것은:

상기 리소스가, 상기 제 1 요청을 서비스하였으며 새로운 상태로 이동하였으며 그리고 상기 새로운 상태를 리소스 프록시에 표시하였다는 상황을 나타내는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법.

**청구항 7**

제 1 항에 있어서,

상기 리소스의 상기 로컬 표현은 리소스 프록시 및 실행 프레임워크 중 임의의 것을 포함하는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법.

**청구항 8**

제 1 항에 있어서,

상기 리소스에 의해 상기 제 1 요청을 확인응답하고 비동기 프로세싱을 개시하는 단계는 상기 리소스를 인코히어런트 상태 (incoherent state) 에 두며,

상기 제 2 요청은 상기 리소스로 하여금 코히어런트 상태로 되돌아가게 하여, 상기 제 2 요청을 프로세싱하는데 이용가능하도록 하는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법.

**청구항 9**

제 1 항에 있어서,

상기 리소스는 상기 제 2 요청을 수신함이 없이 이용가능하도록 되는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법.

**청구항 10**

제 1 항에 있어서,

상기 제 2 요청은 상기 제 1 실행 쓰레드의 컨텍스트에서 발해지는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법.

**청구항 11**

제 1 항에 있어서,

상기 제 2 요청은 제 2 실행 쓰레드의 컨텍스트에서 발해지는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법.

**청구항 12**

휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템으로서,

제 1 클라이언트로부터, 제 1 실행 쓰레드의 컨텍스트에서 발해지는 제 1 요청을 발생시키는 것;

상기 제 1 요청을 리소스로 포워딩하는 것;

상기 제 1 클라이언트가 상기 리소스로 하여금 비동기 프로세싱을 허용할지 여부를 결정하도록 허용한다는 것을 상기 리소스에게 통지하는 디폴트 선호사항을 상기 제 1 클라이언트에 의해 패스하는 것;

상기 리소스에 의해 상기 제 1 요청을 확인응답하고, 상기 리소스가 상기 비동기 프로세싱을 허용하도록 결정하는 경우 상기 비동기 프로세싱을 개시하는 것을 위해 동작가능한 프로세서를 포함하고,

상기 비동기 프로세싱은,

상기 제 1 클라이언트로 하여금 상기 제 1 실행 쓰레드에서 계속해서 프로세싱하는 것을 허용하면서 상기 리소스에서 상기 제 1 요청을 프로세싱하는 것;

상기 리소스에 의해, 상기 제 1 요청의 프로세싱의 완료를 시그널링하는 것으로서, 상기 제 1 요청의 프로세싱의 완료는 상기 리소스의 로컬 표현을 새로운 상태로 업데이트하는 것 및 임의의 등록된 콜백들을 호출하는 것을 포함하며, 상기 리소스는 제 2 요청을 서비스하는데 이용가능하게 되는, 상기 제 1 요청의 프로세싱의 완료를 시그널링하는 것;

상기 리소스에서 상기 제 2 요청을 수신하는 것; 및

상기 제 2 요청을 상기 리소스에서 프로세싱하는 것을 포함하는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 13**

제 12 항에 있어서,

상기 제 2 요청은 상기 제 1 클라이언트에서 발신되는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 14**

제 12 항에 있어서,

상기 제 2 요청은 제 2 클라이언트에서 발신되는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 15**

제 12 항에 있어서,

상기 제 1 요청은 리소스 프록시를 통해서 상기 리소스로 포워딩되는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 16**

제 12 항에 있어서,

상기 제 1 요청을 확인응답하는 것은, 상기 제 1 클라이언트로 하여금, 상기 리소스가 상기 제 1 요청을 프로세싱하기 전에 상기 제 1 실행 쓰레드에서 계속해서 프로세싱하도록 허용하는 것을 더 포함하는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 17**

제 12 항에 있어서,

상기 리소스에 의해, 상기 제 1 요청의 프로세싱의 완료를 시그널링하는 것은, 상기 리소스를 결합가능하게 하며,

상기 리소스가 결합가능하다는 것은:

상기 리소스가, 상기 제 1 요청을 서비스하였으며 새로운 상태로 이동하였으며 그리고 상기 새로운 상태를 리소스 프록시에 표시하였다는 상황을 나타내는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 18**

제 12 항에 있어서,

상기 리소스의 상기 로컬 표현은 리소스 프록시 및 실행 프레임워크 중 임의의 것을 포함하는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 19**

제 12 항에 있어서,

상기 리소스에 의해 상기 제 1 요청을 확인응답하고 비동기 프로세싱을 개시하는 것은 상기 리소스를 인코히어런트 상태에 두며,

상기 제 2 요청은 상기 리소스로 하여금 코히어런트 상태로 되돌아가게 하여, 상기 제 2 요청을 프로세싱하는데 이용가능하도록 하는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 20**

제 12 항에 있어서,

상기 리소스는 상기 제 2 요청을 수신함이 없이 이용가능하게 되는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 21**

제 12 항에 있어서,

상기 제 2 요청은 상기 제 1 실행 쓰레드의 컨텍스트에서 발해지는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 22**

제 12 항에 있어서,

상기 제 2 요청은 제 2 실행 쓰레드의 컨텍스트에서 발해지는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 23**

휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템으로서,

제 1 클라이언트로부터 제 1 요청을 발생시키는 수단으로서, 상기 제 1 요청은 제 1 실행 쓰레드의 컨텍스트에서 발해지는, 상기 제 1 요청을 발생시키는 수단;

상기 제 1 요청을 리소스로 포워딩하는 수단;

상기 제 1 클라이언트가 상기 리소스로 하여금 비동기 프로세싱을 허용할지 여부를 결정하도록 허용한다는 것을 상기 리소스에게 통지하는 디폴트 선호사항을 상기 제 1 클라이언트에 의해 패스하는 수단;

상기 리소스에 의해 상기 제 1 요청을 확인응답하고, 상기 리소스가 상기 비동기 프로세싱을 허용하도록 결정하는 경우 상기 비동기 프로세싱을 개시하는 수단을 포함하고,

상기 비동기 프로세싱은,

상기 제 1 클라이언트로 하여금 상기 제 1 실행 쓰레드에서 계속해서 프로세싱하는 것을 허용하면서 상기 리소스에서 상기 제 1 요청을 프로세싱하는 수단;

상기 리소스에 의해, 상기 제 1 요청의 프로세싱의 완료를 시그널링하는 수단으로서, 상기 제 1 요청의 프로세싱의 완료는 상기 리소스의 로컬 표현을 새로운 상태로 업데이트하는 것 및 임의의 등록된 콜백들을 호출하는 것을 포함하며, 상기 리소스는 제 2 요청을 서비스하는데 이용가능하게 되는, 상기 제 1 요청의 프로세싱의 완료를 시그널링하는 수단;

상기 리소스에서 상기 제 2 요청을 수신하는 수단; 및

상기 제 2 요청을 상기 리소스에서 프로세싱하는 수단을 포함하는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 24**

제 23 항에 있어서,

상기 제 2 요청은 상기 제 1 클라이언트에서 발신되는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 25**

제 23 항에 있어서,

상기 제 2 요청은 제 2 클라이언트에서 발신되는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 26**

제 23 항에 있어서,

상기 제 1 요청을 리소스 프록시를 통해서 상기 리소스로 포워딩하는 수단을 더 포함하는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 27**

제 23 항에 있어서,

상기 제 1 요청을 확인응답하는 수단은, 상기 제 1 클라이언트로 하여금, 상기 리소스가 상기 제 1 요청을 프로세싱하기 전에 상기 제 1 실행 쓰레드에서 계속해서 프로세싱하도록 허용하는 수단을 더 포함하는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 28**

제 23 항에 있어서,

상기 리소스에 의해, 상기 제 1 요청의 프로세싱의 완료를 시그널링하는 것은 상기 리소스를 결합가능하게 하며,

상기 리소스가 결합가능하다는 것은:

상기 리소스가, 상기 제 1 요청을 서비스하였으며 새로운 상태로 이동하였으며 그리고 상기 새로운 상태를 리소스 프록시에 표시하였다는 상황을 나타내는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 29**

제 23 항에 있어서,

상기 리소스의 상기 로컬 표현은 리소스 프록시 및 실행 프레임워크 중 임의의 것을 포함하는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 30**

제 23 항에 있어서,

상기 리소스에 의해 상기 제 1 요청을 확인응답하고 비동기 프로세싱을 개시하는 수단은 상기 리소스를 인코히어런트 상태에 두며,

상기 제 2 요청은 상기 리소스로 하여금 코히어런트 상태로 되돌아가게 하여, 상기 제 2 요청을 프로세싱하는데 이용가능하도록 하는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 31**

제 23 항에 있어서,

상기 리소스는 상기 제 2 요청을 수신함이 없이 이용가능하게 되는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 32**

제 23 항에 있어서,

상기 제 2 요청은 상기 제 1 실행 쓰레드의 컨텍스트에서 발해지는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 33**

제 23 항에 있어서,

상기 제 2 요청은 제 2 실행 쓰레드의 컨텍스트에서 발해지는, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 컴퓨터 시스템.

**청구항 34**

컴퓨터 판독가능 프로그램 코드가 내부에 저장된 컴퓨터 판독가능 저장 매체로서,

상기 컴퓨터 판독가능 프로그램 코드는 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법을 실행하여 구현되도록 적응되며,

상기 방법은,

제 1 클라이언트로부터 제 1 요청을 발생시키는 단계로서, 상기 제 1 요청은 제 1 실행 쓰레드의 컨텍스트에서 발해지는, 상기 제 1 요청을 발생시키는 단계;

상기 제 1 요청을 리소스로 포워딩하는 단계;

상기 제 1 클라이언트가 상기 리소스로 하여금 비동기 프로세싱을 허용할지 여부를 결정하도록 허용한다는 것을 상기 리소스에게 통지하는 디폴트 선호사항을 상기 제 1 클라이언트에 의해 패스하는 단계;

상기 리소스에 의해 상기 제 1 요청을 확인응답하고, 상기 리소스가 상기 비동기 프로세싱을 허용하도록 결정하는 경우 상기 비동기 프로세싱을 개시하는 단계를 포함하고,

상기 비동기 프로세싱은,

상기 제 1 클라이언트로 하여금 상기 제 1 실행 쓰레드에서 계속해서 프로세싱하는 것을 허용하면서 상기 리소스에서 상기 제 1 요청을 프로세싱하는 단계;

상기 리소스에 의해, 상기 제 1 요청의 프로세싱의 완료를 시그널링하는 단계로서, 상기 제 1 요청의 프로세싱의 완료는 상기 리소스의 로컬 표현을 새로운 상태로 업데이트하는 것 및 임의의 등록된 콜백들을 호출하는 것을 포함하며, 상기 리소스는 제 2 요청을 서비스하는데 이용가능하게 되는, 상기 제 1 요청의 프로세싱의 완료를 시그널링하는 단계;

상기 리소스에서 상기 제 2 요청을 수신하는 단계; 및

상기 제 2 요청을 상기 리소스에서 프로세싱하는 단계를 포함하는, 컴퓨터 판독가능 저장 매체.

**청구항 35**

제 34 항에 있어서,

상기 제 2 요청은 상기 제 1 클라이언트에서 발신되는, 컴퓨터 판독가능 저장 매체.

**청구항 36**

제 34 항에 있어서,

상기 제 2 요청은 제 2 클라이언트에서 발신되는, 컴퓨터 판독가능 저장 매체.

**청구항 37**

제 34 항에 있어서,

상기 제 1 요청은 리소스 프록시를 통해서 상기 리소스로 포워딩되는, 컴퓨터 판독가능 저장 매체.

**청구항 38**

제 34 항에 있어서,

상기 제 1 요청을 확인응답하는 단계는 상기 제 1 클라이언트로 하여금, 상기 리소스가 상기 제 1 요청을 프로세싱하기 전에 상기 제 1 실행 쓰레드에서 계속해서 프로세싱하도록 하는 단계를 더 포함하는, 컴퓨터 판독가능 저장 매체.

**청구항 39**

제 34 항에 있어서,

상기 리소스에 의해, 상기 제 1 요청의 프로세싱의 완료를 시그널링하는 단계는 상기 리소스를 결합가능하게 하며,

상기 리소스가 결합가능하다는 것은:

상기 리소스가, 상기 제 1 요청을 서비스하였으며 새로운 상태로 이동하였으며 그리고 상기 새로운 상태를 리소스 프록시에 표시하였다는 상황을 나타내는, 컴퓨터 판독가능 저장 매체.

**청구항 40**

제 34 항에 있어서,

상기 리소스의 상기 로컬 표현은 리소스 프록시 및 실행 프레임워크 중 임의의 것을 포함하는, 컴퓨터 판독가능 저장 매체.

**청구항 41**

제 34 항에 있어서,

상기 리소스에 의해 상기 제 1 요청을 확인응답하고 비동기 프로세싱을 개시하는 단계는 상기 리소스를 인코히어런트 상태에 두며,

상기 제 2 요청은 상기 리소스로 하여금 코히어런트 상태로 되돌아가게 하여, 상기 제 2 요청을 프로세싱하는데 이용가능하도록 하는, 컴퓨터 판독가능 저장 매체.

**청구항 42**

제 34 항에 있어서,

상기 리소스는 상기 제 2 요청을 수신함이 없이 이용가능하게 되는, 컴퓨터 판독가능 저장 매체.

**청구항 43**

제 34 항에 있어서,

상기 제 2 요청은 상기 제 1 실행 쓰레드의 컨텍스트에서 발해지는, 컴퓨터 판독가능 저장 매체.

**청구항 44**

제 34 항에 있어서,

상기 제 2 요청은 제 2 실행 쓰레드의 컨텍스트에서 발해지는, 컴퓨터 판독가능 저장 매체.

## *발명의 설명*

### *배 경 기 술*

[0001] 휴대형 컴퓨팅 디바이스들 ("PCDs") 은 점점 더 인기를 끌고 있다. 이들 디바이스들은 셀룰러 전화기들, 휴

대형/개인 휴대정보 단말기들 ("PDAs"), 휴대형 게임 콘솔들, 휴대형 네비게이션 유닛들, 팜탑 컴퓨터들, 및 다른 휴대형 전자 디바이스들을 포함할 수도 있다.

[0002] PCDs 는 여러 기능들 및 특성들을 제공하는 여러 유형들의 소프트웨어를 실행할 수도 있다. 예를 들어, PCDs 는 비디오들을 시청하기 및 비디오 게임들을 플레이하기와 같은 기능들을 제공할 수도 있는 엔터테인먼트 소프트웨어를 실행할 수도 있다. PCDs 는 또한 스프레드시트들, e-메일, 및/또는 워드 프로세싱 소프트웨어와 같은, 비즈니스 소프트웨어 또는 작성 (writing) 소프트웨어와 같은 다른 유형들의 소프트웨어를 지원할 수도 있다.

[0003] 대개, PCD 상에서 실행하는, 위에서 설명한 소프트웨어는 함께 링크된 여러 하드웨어 엘리먼트들로부터의 액션들을 필요로 한다. 소프트웨어와 하드웨어 엘리먼트들 사이의 상호작용들은 링크된 노드 구조로서 간주될 수 있는 전체 동작 프레임워크에 의해 제어될 수 있다. 일부의 경우, 이들 엘리먼트들 사이의 상호작용은 동기적으로 발생하며, 여기서, 특정한 리소스에 대한 요청은 요청이 승인되고 인정될 때까지 엘리먼트의 동작을 중지한다. 다른 경우, 이들 엘리먼트들 사이의 상호작용은 비동기적으로 발생하며, 여기서, 특정한 리소스에 대한 요청은 요청이 프로세싱될 때까지 엘리먼트의 동작을 중지하지 않는다.

[0004] 그러나, 요청이 프로세싱되는 동안 리소스가 동작을 계속하도록 허용되는 지 여부를 그 요청하는 엘리먼트가 결정할 수 있는 것이 바람직할 것이다.

## *발명의 내용*

### *과제의 해결 수단*

[0005] 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법 및 시스템이 설명된다. 일 실시형태에서, 방법 및 시스템은 제 1 클라이언트로부터 제 1 요청을 발생시키는 것을 포함하며, 제 1 요청은 제 1 실행 쓰레드의 컨텍스트에서 발해진다. 제 1 요청은 리소스로 포워딩될 수도 있다. 리소스는 제 1 요청을 승인하고 비동기 프로세싱을 개시할 수도 있다. 리소스는 제 1 클라이언트로 하여금 제 1 실행 쓰레드에서 프로세싱하는 것을 계속할 수 있도록 하면서, 제 1 요청을 프로세싱할 수도 있다. 리소스는 제 1 요청의 프로세싱의 완료를 시그널링할 수도 있으며, 제 2 요청을 수신할 수도 있다. 제 2 요청은 제 1 요청의 프로세싱의 완료를 일으킨다. 제 1 요청의 프로세싱의 완료는 리소스의 로컬 표현을 새로운 상태로 업데이트하는 것 및 임의의 등록된 콜백들을 호출하는 (call) 것을 포함할 수도 있다. 리소스는 제 2 요청을 서비스하는 것이 이용가능해질 수도 있으며, 제 2 요청을 프로세싱할 수도 있다.

### *도면의 간단한 설명*

[0006] 도면들에서, 유사한 도면부호들은 여러 도면들에 걸쳐서 달리 언급하지 않는 한, 유사한 부재들을 지칭한다. "102A" 또는 "102B" 와 같은 문자 부호 명칭들을 가진 도면 부호들에 있어, 문자 부호 명칭들은 동일한 도면에 존재하는 2개의 유사한 부재들 또는 엘리먼트들을 구분할 수도 있다. 도면부호들에 대한 문자 부호 명칭들은, 그 도면부호가 모든 도면들에서 동일한 도면부호를 갖는 모든 부분들을 포괄하도록 의도될 때 생략될 수도 있다.

도 1 은 휴대형 컴퓨팅 디바이스 ("PCD") 에서 병렬 리소스 요청들을 관리하는 시스템의 예시적인 엘리먼트들을 예시하는 기능 블록 다이어그램이다;

도 2 는 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법 및 시스템을 위한 예시적인 동작 환경을 예시하는 기능 블록 다이어그램이다;

도 3 은 단일 리소스 상에서 실행하는 동기적 쓰레드를 예시하는 다이어그램이다;

도 4 는 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법 및 시스템의 일 실시형태의 동작을 나타내는 다이어그램이다;

도 5 는 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법 및 시스템의 일 실시형태의 동작을 나타내는 타임라인 다이어그램이다;

도 6a 및 도 6b 는 집합적으로, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법 및 시스템의 일 실시형태의 동작을 설명하는 플로우차트를 예시한다;

도 7a 는 도 1 의 휴대형 컴퓨팅 디바이스의 리소스들을 관리하는 노드 아키텍처의 제 1 양태의 다이어그램이다;

도 7b 는 도 1 의 휴대형 컴퓨팅 디바이스의 리소스들을 관리하는 노드 아키텍처의 제 2 양태의 일반적인 다이어그램이다;

도 7c 는 도 1 의 휴대형 컴퓨팅 디바이스의 리소스들을 관리하는 노드 아키텍처의 제 2 양태의 특정의 다이어그램이다;

도 7d 는 휴대형 컴퓨팅 디바이스의 리소스(들)를 관리하는 노드 아키텍처를 생성하는 방법을 예시하는 플로우차트이다;

도 7e 는 휴대형 컴퓨팅 디바이스의 리소스(들)를 관리하기 위한 노드 아키텍처를 생성하는 방법을 예시하는 도 7d 의 연속 플로우차트이다;

도 8 은 휴대형 컴퓨팅 디바이스에서 소프트웨어 아키텍처에서의 노드 구조 데이터를 수신하는 도 7d 의 하위-방법 또는 루틴을 예시하는 플로우차트이다;

도 9 는 휴대형 컴퓨팅 디바이스에 있어 소프트웨어 아키텍처에서의 노드를 생성하는 도 7d 내지 도 7e 의 하위-방법 또는 루틴을 예시하는 플로우차트이다;

도 10 은 휴대형 컴퓨팅 디바이스의 소프트웨어 아키텍처에서 클라이언트를 생성하는 도 9 의 하위-방법 또는 루틴을 예시하는 플로우차트이다;

도 11 은 휴대형 컴퓨팅 디바이스에 있어 소프트웨어 아키텍처에서 리소스에 대한 클라이언트 요청을 생성하는 방법을 예시하는 플로우 차트이다.

## *발명을 실시하기 위한 구체적인 내용*

[0007] 단어 "예시적인" 은 "일 예, 사례, 또는 예시로서 기능하는 것" 을 의미하도록 본원에서 사용된다. 본원에서 "예시적인" 으로 설명하는 임의의 양태는 반드시 다른 양태들에 비해 바람직하거나 또는 유리한 것으로 해석되지는 않는다.

[0008] 본 설명에서, 용어 "애플리케이션" 은 또한 오브젝트 코드, 스크립트들, 바이트 코드, 마크업 언어 파일들, 및 패치들과 같은, 실행가능 콘텐츠를 가진 파일들을 포함할 수도 있다. 게다가, 본원에서 언급되는 "애플리케이션" 은 또한 성질상 실행가능하지 않은 파일들, 예컨대 열려져야 할 수도 있는 문서들 또는 액세스되어야 하는 다른 데이터 파일들을 포함할 수도 있다.

[0009] 용어 "콘텐츠" 는 또한 오브젝트 코드, 스크립트들, 바이트 코드, 마크업 언어 파일들, 및 패치들과 같은 실행가능 콘텐츠를 가진 파일들을 포함할 수도 있다. 게다가, 본원에서 언급되는 "콘텐츠" 는 또한 성질상 실행가능하지 않은 파일들, 예컨대 열려져야 할 수도 있는 문서들 또는 액세스되어야 하는 다른 데이터 파일들을 포함할 수도 있다.

[0010] 본 명세서에서 사용하는, 용어들 "구성요소", "데이터베이스", "모듈", "시스템" 등은 하드웨어, 펌웨어, 하드웨어와 소프트웨어의 조합, 소프트웨어, 또는 실행 중 소프트웨어 중 어느 것이든, 컴퓨터-관련 엔터티를 지칭하도록 의도된다. 예를 들어, 구성요소는 프로세서 상에서 실행하는 프로세스, 프로세서, 오브젝트, 실행가능한 것, 실행 쓰레드, 프로그램, 및/또는 컴퓨터일 수도 있지만 이에 한정되지 않는다. 실 예로서, 컴퓨팅 디바이스 상에서 실행하는 애플리케이션 및 컴퓨팅 디바이스 양자는 구성요소일 수도 있다. 하나 이상의 구성요소들이 프로세스 및/또는 실행 쓰레드 내에 상주할 수도 있으며, 구성요소는 하나의 컴퓨터 상에 로컬라이즈되거나 및/또는 2개 이상의 컴퓨터들 사이에 분산될 수도 있다. 게다가, 이들 구성요소들은 여러 데이터 구조들을 안에 저장하고 있는 여러 컴퓨터 판독가능 매체들로부터 실행할 수도 있다. 구성요소들은 로컬 및/또는 원격 프로세스들의 방법에 의해, 예컨대, 하나 이상의 데이터 패킷들을 갖는 신호 (예컨대, 분산 시스템의 로컬 시스템에서의 또 다른 구성요소와 상호작용하는 하나의 구성요소로부터의, 및/또는 그 신호에 의해 다른 시스템들과의 인터넷과 같은 네트워크를 가로지르는 데이터) 에 따라서, 통신할 수도 있다.

[0011] 본 설명에서, 용어들 "통신 디바이스", "무선 디바이스", "무선 전화기", "무선 통신 디바이스", 및 "무선 핸드셋" 은 상호교환가능하게 사용된다. 3세대 ("3G") 및 4세대 ("4G") 무선 기술의 도래에 따라, 더 큰 대역폭 이용가능성은 아주 다양한 무선 능력들을 가진 더 많은 휴대형 컴퓨팅 디바이스들을 가능하게 하였다.

[0012] 본 설명에서, 용어 "휴대형 컴퓨팅 디바이스" ("PCD") 는 배터리와 같은, 제한된 용량 전원 상에서 동작하는 임의의 디바이스를 기술하는데 사용된다. 배터리 구동되는 PCDs 가 수십 년간 사용되어 왔지만, 3세대 ("3G") 및 4세대 ("4G") 무선 기술의 도래와 결합된, 재충전가능한 배터리들에서의 기술적 진보들은 다수의 능력들을 가진 매우 많은 PCDs 을 가능하게 하였다. 따라서, PCD 는 특히, 셀룰러 전화기, 위성 전화기, 페이저, 개인 휴대정보 단말기 ("PDA"), 스마트폰, 네비게이션 디바이스, 스마트북 또는 리더, 미디어 플레이어, 전술한 디바이스들의 조합, 및 무선 접속을 가진 랩탑 컴퓨터일 수도 있다.

[0013] 도 1 은 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법들 및 시스템들을 구현하는 무선 전화기의 유형에서 PCD (100) 의 예시적인, 비한정적인 양태의 기능 블록 다이어그램이다. 나타낸 바와 같이, PCD (100) 는 멀티-코어, 중앙 처리 유닛 ("CPU") (110A), 그래픽스 프로세서 (110B), 및 아날로그 신호 프로세서 (126) 를 가진 온칩 시스템 (102) 을 포함한다. 이들 프로세서들 (110A, 110B, 126) 은 당업자들에 알려진 바와 같이, 하나 이상의 시스템 버스들 또는 또 다른 상호접속 아키텍처 상에 함께 커플링될 수도 있다.

[0014] CPU (110A) 는 당업자가 주지하고 있는 바와 같이, 0번째 코어 (222), 첫번째 코어 (224), 및 N번째 코어 (226) 를 포함할 수도 있다. 대안적인 실시형태에서, CPU (110A) 및 그래픽스 프로세서 (110B) 를 이용하는 대신, 하나 이상의 디지털 신호 프로세서들 ("DSPs") 이 또한 당업자가 주지하고 있는 바와 같이 채용될 수도 있다. 또, 2개 이상의 멀티-코어 프로세서들이 또한 사용될 수도 있다.

[0015] PCD (100) 는 프로세서들 (110) 에 의해 실행되는 내부 칩 버스 ("ICB") 드라이버 모듈들 (103) 을 포함할 수도 있다. 당업자는, 각각의 ICB 드라이버 모듈 (103) 이 본 개시물로부터 일탈함이 없이 여러 부분들로 분할되고 상이한 프로세서들 (110, 126) 에 의해 실행될 수도 있는 하나 이상의 소프트웨어 모듈들을 포함할 수도 있다는 것을 알 수 있을 것이다.

[0016] 2개의 유형들의 ICB 드라이버 모듈들 (103), 즉, 상부 계층 ("UL") 유형 (103A); 및 하부 계층 ("LL") 유형 (103B) 이 있을 수도 있다. 일반적으로, UL ICB 드라이버 유형들 (103A) 은 여러 애플리케이션 모듈들 (105) 을 지원할 수도 있는 하나 이상의 프로세서들 (110, 126) 에 의해 대개 실행될 것이다. LL ICB 드라이버 유형들 (103B) 은 리소스 전력 관리기 (107) 로서 지칭되는 하나의 하드웨어 엘리먼트에 의해 대개 실행될 것이다.

[0017] LL ICB 드라이버 (103B) 를 실행하는 리소스 전력 관리기 (107) 는 대역폭 값들을 적용하고 설정하는 것을 일반적으로 담당할 것이다. 이들 대역폭 값들은 리소스 전력 관리기 (107) 에 의해, 도 2 와 관련하여 아래에서 설명되는 하나 이상의 버스들 및/또는 스위치 패브릭들에 적용될 것이다. 리소스 전력 관리기 (107) 는 스위치 패브릭들 및 버스들에 대한 클록 속도들 뿐만 아니라 슬레이브들에 대한 클록 속도들을 설정하는 것을 일반적으로 담당한다. 슬레이브들은 일반적으로 애플리케이션 프로그램들 (105) 을 실행하는 마스터 프로세서들 (110) 로부터의 요청들을 지원하는 하드웨어 구성요소들이다.

[0018] 리소스 전력 관리기 (107) 와 함께, ICB 드라이버들 (103A, B) 은 유사한 스위치 패브릭들 내에 및/또는 상이한 스위치 패브릭들을 가로질러서 존재할 수도 있는 하드웨어 구성요소들에 대해 런타임에서 마스터-슬레이브 쌍들의 동적 생성을 가능하게 한다. ICB 드라이버들 (103A, B) 및 리소스 전력 관리기 (107) 는 스위치 패브릭들 및 버스들에 대한 대역폭들을 그때 그때 또는 실시간으로 계산하고 조정할 수도 있다.

[0019] 특정한 양태에서, 본원에서 설명한 방법 단계들 중 하나 이상은 ICB 드라이버들 (103A, B) 을 포함하는 메모리 (112) 에 저장된 실행가능한 명령들 및 파라미터들에 의해 구현될 수도 있다. ICB 드라이버들 (103A, B) 을 형성하는 이들 명령들은 CPU (110A), 아날로그 신호 프로세서 (126), 및 리소스 전력 관리기 (107) 에 의해 실행될 수도 있다. 또, 프로세서들 (110A, 126), 리소스 전력 관리기 (107), 메모리 (112), 그 안에 저장된 명령들, 또는 이들의 조합이 본원에서 설명한 방법 단계들 중 하나 이상을 수행하는 수단으로 기능할 수도 있다.

[0020] 도 1 에 예시된 바와 같이, 디스플레이 제어기 (128) 및 터치스크린 제어기 (130) 는 멀티코어 CPU (110A) 에 커플링된다. 온칩 시스템 (102) 의 외부에 있는 터치스크린 디스플레이 (132) 는 디스플레이 제어기 (128) 및 터치스크린 제어기 (130) 에 커플링된다.

[0021] 도 1 은 또한 멀티코어 CPU (110A) 에 커플링된, 비디오 코더/디코더 ("코덱") (134), 예컨대, PAL (phase-alternating line) 인코더, SECAM (sequential couleur avec memoire) 인코더, NTSC (national television system(s) committee) 인코더, 또는 임의의 다른 유형의 비디오 인코더 (134) 를 예시한다. 비디오 증폭기 (136) 는 비디오 인코더 (134) 및 터치스크린 디스플레이 (132) 에 커플링된다. 비디오 포트 (138) 는 비디

오 증폭기 (136) 에 커플링된다. 범용 시리얼 버스 ("USB") 제어기 (140) 는 CPU (110A) 에 커플링된다. 또한, USB 포트 (142) 는 USB 제어기 (140) 에 커플링된다. 가입자 식별 모듈 (SIM) 카드 (146) 가 또한 CPU (110A) 에 커플링될 수도 있다. 또, 도 1 에 나타낸 바와 같이, 디지털 카메라 (148) 는 CPU (110A) 에 커플링될 수도 있다. 예시적인 양태에서, 디지털 카메라 (148) 는 전하 결합 소자 ("CCD") 카메라 또는 상보형 금속 산화물 반도체 ("CMOS") 카메라이다.

[0022] 또한, 도 1 에 예시된 바와 같이, 스테레오 오디오 코덱 (150) 이 아날로그 신호 프로세서 (126) 에 커플링될 수도 있다. 더욱이, 오디오 증폭기 (152) 가 스테레오 오디오 코덱 (150) 에 커플링될 수도 있다. 예시적인 양태에서, 제 1 스테레오 스피커 (154) 및 제 2 스테레오 스피커 (156) 는 오디오 증폭기 (152) 에 커플링된다. 도 1 은 마이크로폰 증폭기 (158) 가 스테레오 오디오 코덱 (150) 에 커플링될 수도 있다는 것을 나타낸다. 게다가, 마이크로폰 (160) 이 마이크로폰 증폭기 (158) 에 커플링될 수도 있다. 특정한 양태에서, 주파수 변조 ("FM") 무선 튜너 (162) 가 스테레오 오디오 코덱 (150) 에 커플링될 수도 있다. 또한, FM 안테나 (164) 는 FM 무선 튜너 (162) 에 커플링된다. 또, 스테레오 헤드폰들 (166) 이 스테레오 오디오 코덱 (150) 에 커플링될 수도 있다.

[0023] 도 1 은 또한 무선 주파수 ("RF") 송수신기 (168) 가 아날로그 신호 프로세서 (126) 에 커플링될 수도 있다는 것을 나타낸다. RF 스위치 (170) 가 RF 송수신기 (168) 및 RF 안테나 (172) 에 커플링될 수도 있다. 도 1 에 나타낸 바와 같이, 키패드 (174) 가 아날로그 신호 프로세서 (126) 에 커플링될 수도 있다. 또한, 마이크로폰 (176) 을 가진 모노 헤드셋이 아날로그 신호 프로세서 (126) 에 커플링될 수도 있다. 또, 진동기 디바이스 (178) 가 아날로그 신호 프로세서 (126) 에 커플링될 수도 있다. 도 1 은 또한 전원 (180), 예를 들어, 배터리가 온칩 시스템 (102) 에 커플링된다는 것을 나타낸다. 특정한 양태에서, 전원 (180) 은 AC 전력 소스에 접속된 교류 전류 ("AC") 대 DC 변환기로부터 유도된 재충전가능한 DC 배터리 또는 DC 전원을 포함한다.

[0024] 도 1 에 도시된 바와 같이, 터치스크린 디스플레이 (132), 비디오 포트 (138), USB 포트 (142), 카메라 (148), 제 1 스테레오 스피커 (154), 제 2 스테레오 스피커 (156), 마이크로폰 (160), FM 안테나 (164), 스테레오 헤드폰들 (166), RF 스위치 (170), RF 안테나 (172), 키패드 (174), 모노 헤드셋 (176), 진동기 (178), 및 전원 (180) 은 온칩 시스템 (322) 의 외부에 있다.

[0025] 도 2 는 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법 및 시스템을 위한 예시적인 동작 환경 (200) 을 예시하는 기능 블록 다이어그램이다. 동작 환경 (200) 은 이 예에서, CPU (110A) 및 리소스 전력 관리기 (107) 일 수 있는 2개의 프로세서들을 포함한다. 이 실시형태에서, 시스템 버스 (207) 일 수 있는, 리소스는 리소스 전력 관리기 (107) 에 의해 관리된다. 이 실시형태에서, 시스템 버스 (207) 를 위한 리소스 프록시 (207A) 일 수 있는 또 다른 리소스는 CPU (110A) 상에 위치되며 그에 의해 관리된다. 리소스 프록시는 원격 리소스의 로컬 표현으로 정의되며, 원격 리소스의 로컬 표현의 상태는 원격 리소스의 상태를 반영한다.

[0026] 또 다른 실시형태에서, 리소스 프록시 (207A) 는, CPU (110A) 상에 함수들의 라이브러리로서 구현되는, 클라이언트들과 리소스들 사이의 인터페이스로서 기능하는 프레임워크로 대체될 수도 있다. 이런 프레임워크의 일 예가 아래에 설명된다.

[0027] 위에서 설명한 바와 같이, CPU (110A) 는 0번째 코어 (222), 제 1 코어 (224) 및 N번째 코어 (226) 를 포함한다. 그러나, CPU (110A) 는 또한 단일 코어 프로세서일 수 있으며, 여기서, CPU (110A) 는 오직 프로세서일 것이다. 뒤따르는 설명에서, CPU (110A) 는 복수의 코어들을 포함하지만, CPU (110A) 는 단일 프로세서로서 지칭될 것이다. 이와 유사하게, 리소스 전력 관리기 (107) 는 하나 이상의 코어들을 포함할 수도 있다.

[0028] CPU (110A) 상의 각각의 코어가 하나 이상의 쓰레드들을 실행할 수도 있다. 쓰레드는 프로세서 상에서의 실행의 단위이며, 오직 하나의 쓰레드가 주어진 코어 상에서 임의의 주어진 시점에서 활성화될 수도 있다. 클라이언트들이 생성되며 요청들이 하나 이상의 쓰레드들의 컨텍스트 (context) 에서 발해진다. 아래의 설명에서, 용어들 클라이언트 (또는, 클라이언트들) 와 쓰레드 (또는, 쓰레드들) 는 상호교환가능하게 사용된다. 클라이언트가 요청을 발할 때, 활성 쓰레드의 컨텍스트에서 실행하는 클라이언트가 요청을 발했다는 것을 의미한다. 이와 유사하게, 클라이언트는 진행하기 전에 요청에 대한 응답을 대기한다는 의미에서 동기적이라고 하며, 클라이언트가 요청을 발한 쓰레드가 응답이 수신될 때까지 중지된다는 것을 의미한다. 다수의 클라이언트들이 단일 쓰레드의 컨텍스트에서 생성될 수도 있으며, 그러나 이 설명의 목적들을 위해, 오직 하나의 클라이언트가 주어진 쓰레드에서 주어진 시점에 활성화된다는 점에 유의한다.

[0029] 도 2 에 나타낸 예에서, 제 1 클라이언트 (201) 및 제 2 클라이언트 (202) 는 각각 0번째 코어 (222) 상에서 실행하고 있다. 제 3 클라이언트 (204) 는 첫번째 코어 (224) 상에서 실행하고 있는 것으로 도시되며, N번째 클라이언트 (206) 는 N번째 코어 (226) 상에서 실행하고 있는 것으로 도시된다. 그러나, 여러 코어들에의 클라이언트들의 커플링은 임의적이며, 오직 예시적인 목적들을 위해 도 2 에 도시된다. CPU (110A) 가 단일 코어 프로세서인 일 실시형태에서, 클라이언트들의 모두는 CPU (110A) 상에서 실행하고 있을 것이다. 또, 뒤따르는 설명에서, 클라이언트들과 CPU (110A) 내 여러 코어들 사이의 동작 (operative) 상호작용들 중 임의의 상호작용은 또한 CPU (110A) 와 각각의 클라이언트들 사이에서 직접 발생하는 것으로 설명될 수 있다.

[0030] CPU (110A) 는 시스템 버스 (207) 에 리소스 프록시 (207A) 를 통해서 커플링되는 것으로 점선 211 로 도시되며, 반면 시스템 버스 (207) 는 리소스 전력 관리기 (107) 에 커플링되는 것으로 실선 212 를 통해서 도시된다. 이 접속 토폴로지는 리소스 전력 관리기 (107) 가 시스템 버스 (207) 를 관리하고, CPU (110A) 가 리소스 프록시 (207A) 를 경유해서 시스템 버스 (207) 에 논리적 접속을 하지만 능동적으로 관리하지는 않는다는 것을 나타내는 것이다.

[0031] 리소스 프록시 (207A) 는 또한 조건 변수 (condition variable) 의 상태를 나타내는 플래그 (213) 를 포함하는 것으로 도시된다. 조건 변수는 주어진 시점에 참 (true) 이거나 참이지 않을 수도 있는 어떤 조건을 표시하거나 또는 나타낸다. 조건 변수는 그 표현된 조건이 현재 참이라는 것을 나타내기 위해 시그널링될 수도 있으며, 그 조건이 참이 될 때까지 대기하게 될 수 있다. 이 예에서, 플래그 (213) 에 의해 표현되는 조건 변수는 버스 (207) 로의 주어진 요청이 서비스되었는지 여부를 지칭한다. 그 요청이 서비스되면, 그 조건 변수가 이 상태를 나타내기 위해 시그널링된다 (CV/S) (또는, 플래그 (213) 가 설정된다). 그때까지, 조건 변수는 시그널링되지 않는다 (CV/NS). 플래그 (213) 의 동작이 아래에서 좀더 자세히 설명된다.

[0032] 일반적으로, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법 및 시스템은, 리소스, 이 예에서는, 시스템 버스 (207) 를 요청하는 클라이언트들 (201, 202, 204 및 206) 중 임의의 하나의 일 예를 이용하여 설명된다. CPU (110A) 가 시스템 버스 (207) 를 능동적으로 관리하지 않기 때문에, 클라이언트 요청을 리소스 프록시 (207A) 를 경유해서 시스템 버스 (207) 를 관리하는 프로세서로 포워딩한다. 이 예에서, 시스템 버스 (207) 를 관리하는 프로세서는 리소스 전력 관리기 (107) 이다. 따라서, 예를 들어, 클라이언트 (201) 로부터의 요청이 라인 208 로 나타낸 바와 같이, CPU (110A) 로부터 리소스 전력 관리기 (107) 로 전송된다. 그 요청이 서비스될 때, 인터럽트의 유형으로, 응답이, 도면부호 (209) 로 나타낸 바와 같이, 리소스 전력 관리기 (107) 로부터 CPU (110A) 로 전송된다. 리소스 전력 관리기 (107) 로부터 CPU (110A) 전송된 인터럽트는 CPU (110A) 상에서 인터럽트 서비스 루틴 (ISR) 에 의해 처리된다. ISR 은 변수 플래그 (213) 의 조건을 시그널링하고, 이에 의해, CV/NS (condition variable-not signaled) 로부터 CV/S (condition variable-signaled) 로 상태를 변화시킬 것이다. 인터럽트의 효과는 아래에서 좀더 자세히 설명된다.

[0033] 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법 및 시스템의 동작은 도 3, 도 4, 도 5 및 도 6 을 참조하여 추가로 설명된다.

[0034] 도 3 은 프로세서, 이 예에서는, CPU (110A) 상에서 실행하는 클라이언트를 예시하는 다이어그램이다. 216 에서, 클라이언트가 실행 중에 있으며 리소스 요청을 발하며, 이 리소스 요청은 도면부호 (217) 로 나타낸 바와 같이, CPU (110A) 상의 리소스 프록시 (207A) 에 의해 리소스 전력 관리기 (107) 로 포워딩된다. 이 요청은 리소스 전력 관리기 (107) 에 의해 관리되는 임의의 리소스에 대한 것일 수 있다. 지점 215 에서, 클라이언트는 CPU (110A) 상에서 실행하기를 중지하고 (또는, 정지되거나 또는 차단되고), 반면 그 요청이 도면부호 (218) 로 나타낸 바와 같이, 리소스 전력 관리기 (107) 에 의해 프로세싱된다. 일단 그 요청이 리소스 전력 관리기 (107) 에 의해 프로세싱되면, 확인응답 (acknowledgment) 이 (예를 들어, 인터럽트의 유형으로) 도면부호 (219) 로 나타낸 바와 같이, 리소스 전력 관리기 (107) 로부터 CPU (110A) 로 전송되며, 이에 의해, 지점 220 에서 클라이언트는 도면부호 (221) 로 나타낸 바와 같이 CPU (110A) 상에서 실행하는 것을 재개한다. 이런 요청은, 그 요청이 리소스 전력 관리기 (107) 에 의해 프로세싱되는 동안 CPU (110A) 상에서 실행하는 클라이언트 쓰레드 (216) 가 지점 215 에서 중지되고 확인응답 (219) 이 CPU (110A) 에 의해 수신될 때의 지점 220 까지 재개할 수 없기 때문에, 동기적 요청으로서 지칭된다. 그러나, 그 요청이 프로세싱되기를 대기하는 시간 동안 일부 다른 작업을 수행하는 것이 클라이언트에게 바람직한 일부 경우들이 있다. 또한, 클라이언트가 리소스 전력 관리기 (107) 로부터의 확인응답을 대기할 필요가 없으며 간단히 계속해서 실행할 수 있는 것도 가능하다. 이런 요청은 "파이어-앤드-포겟 (fire-and-forget)" 요청으로 지칭될 수도 있다. 파이어-앤드-포겟 요청의 일 예는 클라이언트가 리소스를 턴오프하는 요청일 것이다. 클라이언트 (그리고, 더 나아가, 클라이언트가 실행하고 있는 쓰레드) 가 이 액션이 계속하기 전에 완료될 때까지 대기할 필요가 없다.

이런 방법론의 일 예가 도 4 에 설명된다.

[0035] 도 4 는 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법 및 시스템의 일 실시형태의 동작을 나타내는 다이어그램이다. 도 4 에서, 제 1 쓰레드 (231) 는 CPU (110A) 에서 시작된다. 지점 232 에서, 이 쓰레드에서 클라이언트로부터의 요청이 리소스 프록시 (207A) 에 의해, CPU (110A) 로부터 리소스 전력 관리기 (107) 로 포워딩된다. 일 실시형태에서, 이 요청은 예를 들어, 시스템 버스 (207) 상의 어떤 양의 대역폭을 요청하는 제 1 클라이언트 (201) 로부터 기인할 수 있다. 예를 들어, 시스템 버스 (207) 는 초당 100 megabits (MB/s) 의 대역폭에서 동작하고 있을 수도 있지만, 클라이언트 (201) 는 그것을 200 MB/s 에서 동작하기를 원할 것이다. 따라서, 도면부호 (234) 로 표시된 요청은 시스템 버스 (207) 를 관리하는 리소스 전력 관리기 (107) 로의, 제 1 클라이언트 (201) 로 하여금 더 높은 대역폭에서 동작가능하게 하도록 하는 시스템 버스 (207) 상에서의 추가적인 대역폭에 대한, 요청이다.

[0036] 지점 232 에서, 요청이 CPU (110A) 상의 리소스 프록시 (207A) 로부터 리소스 전력 관리기 (107) 로 포워딩된다. 그러나, 제 1 클라이언트 (201) 가 계속해서 동작하기 위해 시스템 버스 (207) 상의 가용 대역폭이 증가될 때까지 대기할 필요가 없기 때문에 (리소스 전력 관리기 (107) 가 시스템 버스 (207) 상에서 이용가능한 대역폭을 증가시킬 때까지 제 1 클라이언트 (201) 가 더 낮은 대역폭에서 실행할 수 있으며, 그 지점에서, 제 1 클라이언트 (201) 는 더 높은 대역폭을 자동적으로 이용하기 시작한다), 제 1 클라이언트 (201) 는 선호사항 (preference) 으로 전달함으로써, 이것을 CPU (110A) 상의 리소스 프록시 (207A) 에 표시할 수 있다.

[0037] 선호사항은 제 1 클라이언트 (201) 에 의해 규정될 수 있으며 그 리소스가 병렬, 또는 비동기적, 프로세싱을 가능하게 하도록 분기될 (fork) 수 있는지 여부를 결정한다. 선호사항들의 예들은 허용 (ALLOWED), 불허 (DISALLOWED) 및 디폴트 (DEFAULT) 를 포함하지만 이에 한정되지 않는다. 선호사항들 허용 및 불허는 명시적으로 위에서 설명한 바와 같이 병렬 프로세싱을 허용하거나 또는 불허한다. 선호사항 디폴트는, 클라이언트가 리소스로 하여금 클라이언트 거동에 영향을 주지 않는 한, 분기할지 여부에 관하여 스스로 결정을 하도록 허용한다는 것을 리소스에게 통지한다. 예를 들어, 선호사항 디폴트가 규정되면, 리소스는 리소스를 턴오프하거나 또는 이전에 요구된 요청을 취소하는 모든 요청들을 분기하는 것을 선택할 수도 있다. 리소스를 턴오프하거나 또는 이전 요청을 취소하라는 요청은 클라이언트가 이런 요청 이후 리소스의 상태에 관심을 갖지 않는다는 것을 대개 통지하며, 리소스는 이 요청을 분기하고 클라이언트로 즉시 복귀할 수 있다. 그러나, 그 요청이 특정의 값에 대한 것이었고 선호사항이 디폴트이면, 클라이언트는 제어가 클라이언트로 복귀되기 전까지 리소스가 (적어도) 이 특정한 상태에 있을 것으로 예상할 수도 있기 때문에 이런 요청을 분기할 수 없다.

[0038] 선호사항은, 그 요청이 서비스되고 있는 동안에 실행 중에 있는 제 1 클라이언트 (201) 또는 쓰레드가 중지되지 않도록, 그 요청을 리소스 전력 관리기 (107) 로 비동기 방식으로 포워딩하기 위해, CPU (110A) 상의 리소스 프록시 (207A) 에 의해 사용된다. 이런 경우에, CPU (110A) 는 계속해서, 231 에서 시작된 쓰레드를 실행하고 도면부호 (237) 로 나타낸 바와 같이, 동작하는 것이 가능하다. 리소스 요청이 따라서 발해졌지만 아직 서비스되지 않았거나 또는 그 요청이 발해졌고 서비스되었지만 새로운 상태가 CPU (110A) 상의 리소스 프록시 (207A) 에 의해 아직 인식되거나 또는 클라이언트들에게 보이지 않을 때 (즉, 클라이언트가 리소스의 상태를 쿼리할 수 있었으면, 여전히 예전 상태를 "볼" 수 있을 것이다), 그 요청 및 리소스는 "분기된" 으로 표기된다.

전문용어 "분기된" 은 리소스가, 리소스에 대한 최종 요청이 발해졌지만 아직 서비스되지 않거나, 또는 서비스되더라도, 리소스 프록시 (207A) 에 의해 유지되는 것으로, 리소스들의 로컬 표현이, 새로운 상태로 아직 업데이트되지 않는다는 의미에서, 인코히어런트 상태 (incoherent state) 에 있는 조건을 지칭한다. 리소스 프록시 (207A) 는 또한 그 요청이 실제로 서비스된 후에 호출되는 콜백 (callback) 을 등록할 수도 있다. 이 콜백에서, 리소스 프록시 (207A) 는 새로운 (요청되는) 상태에 있는 리소스에 의존하는 다른 액션들을 수행하도록 선택할 수 있다. 예를 들어, 시스템 버스 (207) 의 대역폭이 증가된 후 클록 리소스 (미도시) 가 새로운 값으로 설정되는 것이 가능하다. 이런 경우, 리소스 프록시 (207A) 는 그 콜백에서 클록 리소스 (미도시) 에 대한 그 요청을 발할 것이며, 따라서, 오직 시스템 버스 (207) 에 대한 요청이 서비스된 후에만 클록 리소스 (미도시) 에 대한 요청이 실행된다는 것을 보장할 것이다. 이와 동시에, 리소스 전력 관리기 (107) 는 도면부호 (235) 로 표시된 바와 같이 그 요청을 프로세싱하고, 도면부호 (236) 로 표시된, 확인응답을 CPU (110A) 로 전송한다. 확인응답 (236) 은 (도 2 에 나타낸 바와 같은) 접속 (209) 을 통해서 통신되는 인터럽트에 대응한다. CPU (110A) 상에서 이 인터럽트를 처리/서비스하는 인터럽트 서비스 루틴 (ISR) 은 플래그 (213) 와 연관된 조건 변수를 "시그널링된 (signaled)" 상태, 즉, CV/S 로 설정한다. 이 액션은 대상 리소스가 이제 "결합가능 (joinable)" 하다는 것을 나타낸다. 전문용어 "결합가능" 은 리소스, 예컨대, 시스템 버스 (207) 가 그 요청을 서비스하였으며 새로운 상태로 이동하였으며 이 새로운 상태를 그 요청하는 프로세

서 상에서의 리소스 프록시 (207A) 에 표시하였다는 조건을 지칭한다. 시스템 버스 (207) 의 후속 요청이 CPU (110A) 상의 클라이언트들 (201, 202, 204 및 206) 중 하나에 의해 발해질 때, 리소스가 "결합된다". 전문용어 "결합된" 은 리소스 프록시 (207A) 가 리소스 (이 예에서, 시스템 버스 (207)) 의 로컬 표현을 새로운 상태로 업데이트함으로써, 리소스를 코히어런트하게 하며 새로운 요청을 서비스할 수 있는 액션을 지칭한다.

리소스 프록시 (207A) 는 또한 위에서 설명한 바와 같이 분기에 등록된 임의의 콜백들을 실행할 것이다. 그후, 그 새로운 요청을 처리할 것이다.

[0039] 도 4 에 추가로 나타낸 바와 같이, 도면부호 (241) 로 나타낸 쓰레드에서 실행하는 제 2 클라이언트, 또는 새로운 쓰레드에서 실행하는 동일한 클라이언트 (201) 는, 지점 242 에서 시스템 버스 (207) 에 새로운 요청을 행한다. 따라서, 지점 242 에서, 리소스 (이 예에서, 시스템 버스 (207) 및 그의 로컬 리소스 프록시 (207A)) 는 "결합되며", 이것은 위에서 설명한 바와 같이, 플래그 (213) 와 연관된 조건 변수가 (CV/S) 로 설정되고 리소스를 "결합가능" 으로 표기하기 때문에 가능하다. 지점 242 에서, 제 2 요청이 리소스 프록시 (207A) 에 도달하면, 시스템 버스 (207) 는 아직 "결합가능"하지 않으며, 실행 중에 있는 요청 및 쓰레드는 리소스 전력 관리기 (107) 로부터의 인터럽트가 리소스를 "결합가능" 으로 표기할 때까지 차단할 것이다. 이 새로운 쓰레드에서의 프로세싱은 그후 도면부호 (244) 로 나타낸 바와 같이 속행한다.

[0040] 도 5 는 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법 및 시스템의 일 실시형태의 동작을 나타내는 타임라인 다이어그램이다. 시간 t0 에서, CPU (110A) 상에서 실행하는 제 1 클라이언트 (201) 는 도면부호 (251) 를 이용하여 예시된 바와 같이 작업을 시작한다. 시간 t1 에서, 제 1 클라이언트 (201) 는 도면부호 (252) 로 나타낸 바와 같이, 시스템 버스 (207) 와 같은, 원격 리소스에 대한 요청을, 로컬 리소스 프록시 (207A) 를 통해서 발한다. 요청이 로컬 리소스 프록시 (207A) 에서 수신되며, 도면부호 (254) 를 이용하여 예시된 바와 같이 CPU (110A) 상에서 수행되기 시작한다. 시간 t2 에서, 리소스 프록시 (207A) 는 도면부호 (255) 를 이용하여 예시된 바와 같이, 그 요청을 제 2 프로세서 (리소스 전력 관리기 (107)) 로 포워딩한다. 그 요청이 리소스 전력 관리기 (107) 에 의해 수신될 때, 리소스 전력 관리기 (107) 는 도면부호 (256) 를 이용하여 예시된 바와 같이, 그 요청을 실행하여 프로세싱하기 시작한다. 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법 및 시스템의 일 실시형태에 따르면, 제 1 클라이언트 (201) 가 그 요청의 '분기' 를 허용하였다고 가정하면, 시간 t3 에서, 리소스 프록시 (207A) 는 도면부호 (257) 를 이용하여 나타낸 바와 같이, 그 요청을 비동기 방식으로 (즉, 그 요청을 원격 프로세서 (리소스 전력 관리기 (107)) 로 발하지만, 그 요청이 서비스되고 인식될 때까지 대기하지 않음으로써) 프로세싱하고, 제 1 클라이언트 (201) 로 복귀한다. 제 1 클라이언트 (201) 는 이제 도면부호 (260) 로 나타낸 바와 같이 작업을 계속한다. 타임라인으로 나타낸 바와 같이, 260 으로 나타낸 기간 동안 제 1 클라이언트 (201) 에 의해 수행된 작업은 256 으로 표시되는 시간 기간 동안 리소스 전력 관리기 (107) 에 의해 수행되는 작업과 병렬로 발생한다. 요청이 리소스 프록시 (207A) 에 의해 제 1 클라이언트 (201) 로 반송되는 지점 258 은 "분기" 지점으로서 지칭된다.

분기 지점은 지점 258 에서, 리소스 또는 요청이 리소스 전력 관리기 (107) 에서 그리고 클라이언트 (201) 에서 동시에 병렬 프로세싱을 허용하도록 분기한다는 것을 나타낸다.

[0041] 시간 t4 에서, 제 2 쓰레드로의 컨텍스트 스위치 (context switch) 가 도면부호 (266) 를 이용하여 나타낸 바와 같이 발생한다. 시간 t5 에서, 리소스 전력 관리기 (107) 는 도면부호 (261) 를 이용하여 나타낸 바와 같이, CPU (110A) 상의 리소스 프록시 (207A) 로 인터럽트를 전송한다. CPU (110A) 에서 이 인터럽트를 처리하는 인터럽트 서비스 루틴 (ISR) 은 도면부호 (264) 를 이용하여 나타낸 바와 같이 조건 변수 플래그 (213) (도 2) 를 "시그널링된" (CV/S) 상태로 설정한다. 지점 265 에서, 시스템 버스 (207) 는 "결합가능" 한 것으로 간주되며 후속 요청들을 프로세싱하는데 이용가능하다.

[0042] 시간 t7 에서, CPU (110A) 상에서의 제 2 쓰레드에서 실행하는, 제 2 클라이언트 (202) 는 도면부호 (268) 를 이용하여 나타낸 바와 같이 리소스 프록시 (207A) 를 통해서 시스템 버스 (207) 로 요청을 발한다. 그 요청 (268) 은 시스템 버스 (207) 의 관점으로부터 "후속" 요청인 것으로 간주될 수 있다. 후속 요청이 수신될 때, 리소스 프록시 (207A) 는 체크하여, 시스템 버스 (207) 가 결합가능 상태에 있고 "결합될" 수 있다고 결정한다. 지점 270 에서, 시스템 버스 (207) 는 제 2 쓰레드의 컨텍스트에서 결합된다. 이것은 시스템 버스 (207) 가 후속 요청 (268) 을 제 2 클라이언트 (202) 로부터의 제 2 쓰레드에서 프로세싱하기 위해, 지점 265 에서 "결합가능"하지만, 지점 270 에서 "결합되는" 것으로 간주된다는 것을 의미한다. 리소스 (이 예에서, 시스템 버스 (207)) 는 따라서 3개의 상태들 - 분기된, 결합가능, 및 결합된 중 하나일 수 있다. 리소스가 분기되는 지점으로부터, 인코히어런트 상태에 있는 것으로 간주되며, 요청이 수신되었지만 아직 프로세싱되지 않았거나 또는 프로세싱되었으면, 리소스의 로컬 표현이 아직 실제 (새로운) 리소스 상태로 업데이트되지

않았다는 것을 의미한다. 이런 상태에서, 리소스는 후속 요청들을 프로세싱할 수 없다. 따라서, 리소스는, 후속 요청 (268) 을 프로세싱할 수 있기 전에, "결합된다", 즉, 지점 270 에서 코히어런트로 된다. 271 에 나타낸 시간 기간은 제 2 클라이언트 (202) 로부터의 후속 요청 (268) 이 리소스 프록시 (207A) 에 의해 프로세싱되는 동안의 시간을 예시한다. 이 요청은 또한 "분기" 할 수도 있으며, 아래에서 설명하는 시퀀스를 반복할 수도 있음에 유의한다.

[0043] 후속 요청이 상이한 클라이언트로부터 유래할 필요는 없지만, 일 실시형태에서, 또한 제 1 클라이언트 (201) 로부터 후속 요청의 유형으로 유래할 수 있다는 점이 언급되어져야 한다. 중요하게는, 시스템 버스 (207) 의 임의의 클라이언트가 후속 요청을 개시하고 그 결과로, 시간 t7 에서 "결합" 할 수 있다. 제 1 클라이언트 (201) 의 컨텍스트에서 (그리고, 이 실시형태에서, 상이한 쓰레드에서) 발해진 이전 요청은 따라서, 위에서 언급한 바와 같이, CPU (110A) 상에서의 임의의 쓰레드 상의 임의의 클라이언트로부터 유래할 수도 있는 후속 요청의 프로세싱 동안 (로컬 리소스 표현 (리소스 프록시 (207A)) 을 새로운 상태로 업데이트함으로써, 리소스 (시스템 버스 (207)) 를 코히어런트로 다시 만든다는 의미에서) 가 "완료된다". 이것은 추가적인 관리 없이 발생하며, 시스템 버스 (207) 로의 임의의 요청은 지점 270 에서 그 리소스를 이용가능성 및 액션으로 되돌려 줄 수 있다.

[0044] 도 6a 및 도 6b 는 집합적으로, 휴대형 컴퓨팅 디바이스에서 병렬 리소스 요청들을 관리하는 방법 및 시스템의 일 실시형태의 동작을 설명하는 플로우차트를 예시한다. 블록 (603) 에서, 리소스에 대한 요청이 수신된다. 상기 예에서, 요청은 리소스 프록시 (207A) 에 의해 수신된다. 그러나, 요청은 또 다른 엔터티에 의해 수신될 수 있다. 이 실시형태에서, 제 1 클라이언트 (201) 는 리소스 전력 관리기 (107) 와 같은 원격 프로세서에 의해 관리되는 시스템 버스 (207) 를 요청한다. 설명되는 예에서, 요청은 CPU (110A) 상에서의 로컬 리소스 프록시 (207A) 를 통해서 이루어진다.

[0045] 블록 (604) 에서, 그 요청을 발한 제 1 클라이언트 (201) 가 이 요청을 동기적으로 완료하기를 원하는지 여부 또는 제 1 클라이언트 (201) 가 리소스 프록시 (207A) 로 하여금 이 요청을 "분기" 하여 제 1 클라이언트 (201) 로 즉시 복귀하도록 허용하는지 여부가 결정된다. 제 1 클라이언트 (201) 가 분기를 불허하고 리소스 요청이 동기적으로 완료되기를 원하면, 블록 (606) 에서, 리소스 프록시 (207A) 는 그 요청을 리소스 전력 관리기 (107) 로 포워딩하고, 프로세싱 쓰레드를 차단하고, 그리고, 리소스 전력 관리기 (107) 가 그 요청을 서비스할 때까지 대기한다.

[0046] 블록 (608) 에서, 제 1 클라이언트 (201) 가 그 요청을 분기하도록 허용하였으면, 리소스 프록시 (207A) 는 그 요청을 리소스 전력 관리기 (107) 로 포워딩하고, 그 요청이 서비스되기를 대기하지 않고, 클라이언트로 즉시 복귀한다.

[0047] 블록 (612) 에서, 제어는 제 1 클라이언트 (201) 로 복귀하며, 쓰레드는 그 요청이 리소스 전력 관리기 (107) 에 의해 아직 서비스되지 않았음에도 불구하고, 계속해서 실행한다. 리소스는 "분기된" 으로 표기되며, 임의의 후속 요청을 서비스할 수 있을 때까지 결합되어야 한다.

[0048] 블록 (614) 에서, 리소스 전력 관리기 (107) 는 그 요청의 프로세싱을 완료하고 CPU (110A) 로 인터럽트를 전송하여 이것을 나타낸다. 이 인터럽트를 처리하는, CPU (110A) 상에서의 인터럽트 서비스 루틴 (ISR) 은 조건 변수를 시그널링하며, 플래그 (213) 가 (CV/S) 로 설정됨으로써, 리소스가 '결합가능' 상태로 설정되게 한다. 블록 (614) 에서 설명되는 단계는 예시된 바와 같이 직렬로, 병렬로, 또는 블록 (612) 에서 설명된 단계 이후 및 블록 (628) 에서 설명된 단계 이전에 임의의 시간에 일어날 수도 있다.

[0049] 블록 (618) 에서, 후속 요청이 개시된다. 후속 요청이 동일한 쓰레드에서 또는 제 2 쓰레드에서 개시될 수도 있다. 게다가, 후속 요청이 동일한 클라이언트에 의해 또는 상이한 클라이언트에 의해 개시될 수도 있다. 이 예에서, 후속 요청은 상이한 쓰레드에서 상이한 클라이언트, 즉, 제 2 클라이언트 (202) 에 의해 개시된다.

[0050] 또 다른 실시형태에서, 후속 요청은 동일한 쓰레드로부터 제 1 요청으로서 개시될 수도 있다. 초기 쓰레드에서의 프로세싱이 제 1 요청에 의해 차단되지 않았기 때문에, 초기 쓰레드는 다른 작업을 한 후 리소스에 대한 후속 요청을 행할 수도 있다. 예를 들어, 쓰레드는 로컬 리소스 프록시에 의해 분기된 원격 리소스를 턴오프하도록 요청을 발할 수도 있다. 어떤 시간 기간 이후, 쓰레드 (또는, 좀더 구체적으로는, 쓰레드에서의 클라이언트) 는 리소스를 턴오프하기를 원할 수도 있다. 이런 경우, 후속 요청은 동일한 쓰레드에서 개시된다.

[0051] 블록 (620) 에서, 후속 요청이 리소스 프록시 (207A) 에서 수신된다.

[0052] 블록 (624) 에서, 리소스 (시스템 버스 (207)) 가 결합가능한지 여부가 결정된다. 블록 (624) 에서 리소스가 결합가능하지 않다고 결정되면, 프로세스는 블록 (626) 로 진행하며, 여기서 쓰레드는 리소스가 결합가능하게 될 때까지 대기한다. 블록 (624) 에서 리소스가 결합가능하다고 결정되면, 또는 리소스가 블록 (626) 에서 결합가능하게 된 후, 프로세스는 블록 (628) 으로 진행하며, 여기서 리소스는 "결합되며", 리소스 프록시 (207A) 는 이전 분기된 요청으로부터 임의의 계류중인 프로세싱을 완료하고 리소스의 로컬 표현을 새로운 상태로 업데이트한다. 이러한 방법으로, 리소스가 '결합된' 상태로 이동된다. 리소스는 이제 제 2 요청을 프로세싱하기 시작한다.

[0053] 후속 요청이 "분기된" 또는 "결합가능" 상태에서 리소스에 도달할 때를 보장하는 암시적인 결합에 대한 대안은 "명시적인 결합 (explicit join)" 으로 알려져 있다. 위에서 설명한 바와 같이, 분기된 리소스는 리소스에 대한 후속 요청이 리소스 프록시 (또는, 프레임워크) 에 도달할 때는 언제나 암시적으로 결합한다. 그러나, 클라이언트가 후속 요청을 발함이 없이, 리소스에 명시적으로 결합하기를 원하는 경우들이 있을 수도 있다. 이런 경우, 분기된 리소스가 결합되어 코히어런트로 되게 할 수도 있는 수단은 "명시적인 결합" 으로서 지칭된다. 이런 명시적인 결합 동작은 조건 변수가 시그널링되기를 대기하고, 로컬 리소스 표현을 업데이트하여 새로운 리소스 상태를 반영하고, 클라이언트로 복귀할 것이다. 클라이언트는 이 호출이 복귀될 때까지 중지될 것이다.

[0054] 원격 리소스들에 대한 "분기된" 요청들에 대한 대안은 분기된 '로컬' 리소스들/요청들이다. 상기 예에서는, 요청이 원격으로 서비스되고, 그리고, 그 클라이언트 (또는, 호출자) 가 그 요청하는 프로세서 상에서 다른 동작들을 수행하거나 또는 어쩌면, 그의 요청이 서비스되는 동안 다른 동작들을 수행하기 위해 다른 클라이언트들 또는 다른 쓰레드들에게 양보하기를 원한다고 가정된다. 이것은 항상 사실일 필요는 없다. 요청이 CPU (110A) 상에서의 또 다른 쓰레드에 의해 서비스되는 시나리오를 고려한다. 이런 경우, 클라이언트는 (운영 시스템에 의해 스케쥴링될 때는 언제나) 다른 쓰레드에 의해 선택되며, 서비스되고, 호출하는 쓰레드에 대한 확인응답이 반환되는 이러한 리소스에 요청을 발한다. 원격 프로세서를 이 요청 프로세싱 쓰레드로 대체함으로써 상기 방법 및 시스템을 이 시나리오에 적용하는 것이 가능하다.

[0055] "결합" 에 대한 대안은 "열망하는 결합 (eager join)" 상태로서 지칭된다. 위에서 설명한 실시형태에서, 분기된 리소스들은 후속 요청이 리소스 (또는 그의 프록시) 에 도달할 때는 암시적으로, 또는 클라이언트가 요청들 그것을 요청할 때는 명시적으로, 결합된다. 또 다른 실시형태에서, 후속 요청 또는 명시적인 결합과는 독립적으로, '결합가능' 리소스를 결합하는 표현 목적을 위해 생성된 다수의 작업자 쓰레드들이 있을 수 있다.
이런 시나리오에서, 원격 프로세서로부터 완료의 통지를 수신하는 ISR 은 조건 변수를 시그널링하고 리소스를 "결합가능" 으로 표기할 뿐만 아니라, 결합 액션 (결합 콜백) 을 큐 상에 작업 아이템으로서 할당할 것이다.
큐 상에서 대기하고 있는 작업자 쓰레드들은 이 작업 아이템을 조사하고, 이를 꺼집어 내어, 그들의 컨텍스트들에서 이것을 실행할 것이다; 결합 액션이 수행되며 리소스가 가능한 한 빨리 '결합된' 상태로 설정되는 것을 보장한다.

[0056] 위에서 언급한 바와 같이, 대안적인 실시형태에서, 클라이언트와 리소스 사이의 인터페이스는 리소스 프록시 대신, 프레임워크일 수도 있다. 이런 프레임워크의 설계 및 구조가 아래에 설명된다.

[0057] 아래에 설명된 바와 같은 도 7a 내지 도 10 은 어떻게 도 2 의 노드 아키텍처가 설정되거나 유지되는지를 설명하기 위해 제공된다. 도 7a 는 도 4 에 예시된 노드 아키텍처를 설정하고 유지하는 소프트웨어 아키텍처 (500A) 의 제 1 양태의 다이어그램이다.

[0058] 도 7a 는 소프트웨어 또는 하드웨어 (또는, 양자) 를 나타내는 기능 블록들을 포함하는 다이어그램이다. 도 7a 는 ICB 드라이버 모듈 (103); 또한 일반적으로 제 1 하드웨어 엘리먼트 (하드웨어 엘리먼트 #1) 로도 지칭되는 중앙 처리 유닛 (110); 또한 일반적으로 제 2 하드웨어 엘리먼트 (하드웨어 엘리먼트 #2) 로도 지칭되는 CPU (110) 를 위한 클록 (442); 또한 일반적으로 제 3 하드웨어 엘리먼트 (하드웨어 엘리먼트 #3) 로도 지칭되는 버스 아비터 (arbiter) 또는 스케쥴러 (422); 또한 일반적으로 제 1 소프트웨어 엘리먼트 (소프트웨어 엘리먼트 #1) 로도 지칭되는 버스 프로그램 A - 444A; 또한 일반적으로 제 2 소프트웨어 엘리먼트 (소프트웨어 엘리먼트 #2) 로도 지칭되는 버스 프로그램 B - 444B; 일반적으로 제 3 소프트웨어 엘리먼트 (소프트웨어 엘리먼트 #3) 로 지칭되는 클록 프로그램 AHB; 키누름 (448) 으로 일반적으로 표시되는, 소프트웨어 엘리먼트에 의해 모니터링되는 액션 또는 기능; 및 소프트웨어 엘리먼트 또는 하드웨어 엘리먼트 또는 양자를 포함하는 레거시 엘리먼트 (450) 와 같은, 그러나 이에 한정되지 않는, 복수의 하드웨어 및 소프트웨어 엘리먼트들에 커플링된 아키텍

처 또는 프레임워크 관리기 (440) 를 예시한다.

[0059] 레거시 소프트웨어 엘리먼트의 일 예는 동적 환경 관리기 (DEM) 를 포함하지만 이에 한정되지 않을 수도 있다. 이것은 프로세서 슬립 이벤트들의 프로세서 간 통지를 처리하는 소프트웨어 모듈이다. 예를 들어, 제 1 프로세서 (A) 는 DEM 을 이용하여, 제 2 프로세서 (B) 가 휴지로 진행하였거나/휴지로부터 복귀하였다는 통지를 수신한다. 더 새로운 하드웨어에 있어, 이 소프트웨어 기능이 루트 프로세서 모듈 (RPM) 서브시스템/통신 프로토콜에 포함되었다. 다른 레거시 소프트웨어 엘리먼트들이 존재하며 본 발명의 범위 내에 포함된다.

[0060] 레거시 하드웨어 엘리먼트의 일 예는 AMBA (진보된 마이크로제어기 버스 아키텍처) 높은-성능 버스 (AHB) 를 포함할 수도 있지만 이에 한정되지 않는다. 더 오래된 PCDs (100) 에 있어, AHB 는 1차 시스템 버스를 포함할 수도 있으며, 반면, 더 새로운 PCDs (100) 에 있어, 시스템 버스 패브릭 (107) 은 완전히 상이하며, AHB 버스는 그 새로운 시스템 버스 패브릭을 통해서 통신하도록 아직 업데이트되지 않은 모듈들과 통신하기 위해 오직 특수 애플리케이션들에만 사용된다. 다른 레거시 하드웨어 엘리먼트들이 존재하며 본 발명의 범위 내에 포함된다.

[0061] 프레임워크 관리기 (440) 는 전술한 하드웨어 및 소프트웨어 엘리먼트들의 각각과 통신하는 노드들과 같은, 데이터 구조들을 관리하는 컴퓨터 명령들의 라이브러리를 포함할 수도 있다. 프레임워크 관리기 (440) 는 도 7a 의 파선 A 의 우측면 상에 예시된 바와 같이, 노드들 (602, 622, 642, 및 646) 을 형성할 수도 있는 하나 이상의 리소스들을 생성하는 것을 담당할 수도 있다.

[0062] 프레임워크 관리기 (440) 는 CPU (110) 상에 상주하는 각각의 ICB 드라이버 모듈 (103) 과 직접 통신할 수도 있다. 도 7a 의 우측면 상에서 각각의 노드 (602, 622, 642, 및 646) 는 도 7a 의 파선 A 의 좌측면 상에서의 각각의 소프트웨어 또는 하드웨어 엘리먼트의 표현 또는 모델이다. 본 개시물의 나머지에 있어, 일반적인 또는 비-특정의 노드는 도 7b 에 예시된 바와 같이, 도면부호 (601) 로 지정될 것이다.

[0063] 앞에서 언급한 바와 같이, 도 7a 의 각각의 예시적인 노드 (602, 622, 642, 및 646) 는 하나 이상의 리소스들을 포함할 수도 있다. 리소스는 소프트웨어 엘리먼트 또는 하드웨어 엘리먼트 또는 양자를 포함할 수도 있다. 예를 들어, 제 1 노드 (602) 는 일반적으로 제 1 하드웨어 엘리먼트 또는 중앙 처리 유닛 (110) 에 대응하는 단일 리소스를 포함한다. 본 개시물에서 설명된 소프트웨어 아키텍처에 의하면, 노드 (601) 의 각각의 리소스에는 하나 이상의 영숫자 문자들을 포함하는 고유한 이름이 제공될 수도 있다. 도 7a 에 예시된 예시적인 실시형태에서, 제 1 노드 (602) 의 리소스에는 "core/cpu" 의 리소스 이름이 할당되었다. 이 예시적인 리소스 이름은 일반적으로 당업자에게 알려진 종래의 파일 명명 (naming) 구조들에 대응한다. 그러나, 당업자가 주지되는 바와 같이, 영숫자 문자들 및/또는 심볼들의 임의의 다른 조합을 포함하는 다른 유형들의 리소스 이름들은 완전히 본 발명의 범위 내이다.

[0064] 도 7a 의 예시적인 실시형태에서, 제 2 노드 (622) 는 복수의 리소스들을 포함한다. 구체적으로 설명하면, 이 특정한 예시적인 실시형태에서, 제 2 노드 (622) 는 버스 아비터 또는 스케쥴러 (422) 에 대응하는 단일 하드웨어 엘리먼트를 포함하는 제 1 리소스를 갖는다. 제 2 노드 (622) 의 제 2 리소스는 버스 프로그램 A (444A) 의 제 1 소프트웨어 엘리먼트에 일반적으로 대응하는 소프트웨어 엘리먼트를 포함한다. 제 2 노드 (622) 의 제 3 리소스는 버스 프로그램 B (444B) 의 제 2 소프트웨어 엘리먼트에 일반적으로 대응하는 또 다른 소프트웨어 엘리먼트를 포함한다. 당업자는 임의의 조합 및 임의 개수의 리소스들 및 주어진 노드 (601) 에 대한 리소스 유형들이 완전히 본 발명의 범위 이내임을 알 수 있다.

[0065] 노드들 (601) 을 생성하는 것에 더해서, 프레임워크 관리기 (440) 는 또한 마커들 (650) 을 생성하거나 또는 인스턴스화할 수도 있다. 마커는 그 자신들을 용이하게 맵핑하지 않거나 또는 프레임워크 관리기 (440) 에 의해 관리되는 소프트웨어 아키텍처와 용이하게 호환할 수 없는, 하드웨어 엘리먼트 또는 소프트웨어 엘리먼트 (또는, 양자 뿐만 아니라 복수의 이들 엘리먼트들) 와 같은, 하나 이상의 레거시 엘리먼트들을 포함할 수도 있다. 마커 (650) 는 노드 (601) 의 리소스를 지원할 수 있으며, 노드 (601) 의 리소스가 마커 (650) 에 의존할 수도 있다는 것을 의미한다. 마커 (650) 의 일 예는 스트링 드라이버를 포함할 수도 있다. 스트링 드라이버는 도 7a 와 관련하여 설명한 아키텍처 내에 용이하게 적응되지 않을 수도 있다. 마커 (650) 는 노드 (601) 및 도 8 의 블록 (1125) 에서 수집된 그의 의존성 어레이 데이터에 의해 참조될 수도 있다.

[0066] 도 7a 는 또한 2개의 소프트웨어 엘리먼트들 (448, 450) 의 액션 또는 기능에 일반적으로 대응하는 제 1 클라이언트 (648) 를 예시한다. 도 7a 에 예시된 예시적인 실시형태에서, 제 1 클라이언트 (648) 는 휴대형 컴퓨팅 디바이스 (100) 에 의해 지원되는 특정한 애플리케이션 프로그램 모듈 (105) 내에서 일어날 수도 있는 키누

름 액션에 일반적으로 대응한다. 그러나, 당업자는 키누름들 이외의 소프트웨어 엘리먼트들의 다른 액션들 및/또는 기능들이 완전히 본 발명의 범위 이내임을 알 수 있다. 클라이언트 요청들 (648) 및 그들의 각각의 생성에 관한 추가적인 세부 사항들은 도 10 을 참조하여 아래에서 설명된다.

[0067] 도 7a 는 또한 특정한 아키텍처 엘리먼트들 사이의 관계들을 예시한다. 예를 들어, 도 7a 는 클라이언트 (648) 와 제 1 노드 (602) 사이의 관계를 예시한다. 구체적으로 설명하면, 제 1 클라이언트 (648) 는 리소스 "/core/cpu" 를 포함하는 제 1 노드 (602) 에 의해 관리되거나 또는 처리되는, 파선들로 예시된, 클라이언트 요청 (675A) 을 발생할 수도 있다. 일반적으로, 클라이언트 요청들 (675) 의 유형들의 미리 결정된 또는 설정된 개수가 존재한다. 클라이언트 요청들 (675) 은 도 10 과 관련하여 아래에서 좀더 자세히 설명된다.

[0068] 도 7a 에서 디스플레이되는 다른 관계들은 파선들 (680) 로 예시된 의존성들을 포함한다. 의존성들은 또 다른 노드 (601) 의 각각의 리소스들 사이의 관계들이다. 의존성 관계는 제 1 리소스 (A) 가 제 1 리소스 (A) 에 정보를 제공할 수도 있는 제 2 리소스 (B) 에 의존한다는 것을 대개 나타낸다. 이 정보는 제 2 리소스 (B) 에 의해 수행되는 동작의 결과일 수도 있거나 또는 단지 제 1 리소스 (A) 또는 이들의 임의의 조합에 의해 요구되는 상태 정보를 포함할 수도 있다. 제 1 리소스 (A) 및 제 2 리소스 (B) 는 동일한 노드 (601) 의 부분일 수도 있거나 또는 상이한 노드들 (601) 의 부분일 수도 있다.

[0069] 도 7a 에서, 제 1 노드 (602) 는 제 1 노드 (602) 에서 발신하여 제 2 노드 (622) 로 연장하는 의존성 화살표 (680B) 로 나타낸 바와 같이 제 2 노드 (622) 에 의존한다. 도 7a 는 또한 제 1 노드 (602) 가 또한 의존성 화살표 (680A) 로 예시된 바와 같이 제 3 노드 (642) 에 의존한다는 것을 예시한다. 도 7a 는 또한 제 2 노드 (622) 가 의존성 화살표 (680C) 로 예시된 바와 같이 제 4 노드 (646) 에 의존한다는 것을 예시한다. 당업자는 도 7a 의 점선 화살표들로 예시된 의존성들 (680) 이 단지 성질에 있어 예시적이며, 그리고 각각의 노드들 (601) 사이의 의존성들의 다른 조합들도 본 발명의 범위 이내임을 알 수 있다.

[0070] 아키텍처 또는 프레임워크 관리기 (440) 는 도 7a 에 예시된 클라이언트 요청들 (675) 및 의존성들 (680) 을 포함하지만 이에 한정되지 않는, 위에서 설명한 관계들을 유지하는 것을 담당한다. 프레임워크 관리기 (440) 는 임의의 주어진 노드 (601) 에 대한 의존성들 (680) 이 완료되기만 하면, 가능한 한 많은 노드들 (601) 을 인스턴스화하거나 또는 생성하려고 시도할 것이다. 의존성 (680) 은 의존성을 지원하는 리소스가 존재하고 있거나 또는 의존성 (680) 에 관련된 정보를 처리하려고 준비 상태에 있을 때 완료된다.

[0071] 예를 들어, 제 1 노드 (602) 와 제 3 노드 (642) 사이에 존재하는 의존성 관계 (680A) 때문에 단일 리소스 "/clk/cpu" 를 포함하는 제 3 노드 (642) 가 생성되지 않았으면, 단일 리소스 "/core/cpu" 를 포함하는 제 1 노드 (602) 가 프레임워크 관리기 (440) 에 의해 생성되거나 또는 설정되지 않을 수도 있다. 일단 제 3 노드 (642) 가 프레임워크 관리기 (440) 에 의해 생성되었으면, 프레임워크 관리기 (440) 는 의존성 관계 (680A) 때문에 제 2 노드 (602) 를 생성할 수도 있다.

[0072] 프레임워크 관리기 (440) 가 특정한 노드 (601) 를, 그의 의존성들 (680) 중 하나 이상이 불완전하기 때문에, 생성하거나 또는 인스턴스화할 수 없으면, 프레임워크 관리기 (440) 는 프레임워크 관리기 (440) 에 의해 성공적으로 생성된 그들 노드들 (601) 에 대응하는 단계들을 계속해서 수행하거나 또는 실행할 것이다. 프레임워크 관리기 (440) 는 의존적인 리소스들이 생성되지 않는 불완전한 의존성들로 인해 존재하지 않을 수도 있는 특정한 노드 (601) 에 대한 호출을 보통은 건너 뛰고, 그 불완전한 상태를 반영하는 메시지들을 그 호출에 반송할 것이다.

[0073] 도 1 에 예시된 바와 같은, 멀티코어 환경에서, 프레임워크 관리기 (440) 는 도 1 의 0번째, 첫번째 및 N번째 코어들 (222, 224, 및 226) 과 유사하게, 별개의 코어들 상에서 노드들 (601) 을 생성하거나 또는 인스턴스화할 수도 있다. 노드들 (601) 은 일반적으로, 노드들 (601) 이 서로 의존하지 않기만 하면, 그리고, 특정한 노드의 대응하는 의존성들 모두가, 이하에서 설명하는 바와 같이, 완료되면, 멀티코어 환경에서 별개의 코어들 상에서 그리고 병렬로 생성될 수도 있다.

[0074] 도 7b 는 도 1 의 PCD (100) 의 리소스들을 관리하는 시스템을 위한 소프트웨어 아키텍처 (500B1) 의 제 2 양태의 일반적인 다이어그램이다. 이 일반적인 다이어그램에서, 각각의 노드 (601) 의 하나 이상의 리소스들에는 고유한 이름들이 제공되지 않았다. 도 7b 의 노드 또는 리소스 그래프 (500B1) 는 단지 노드들 (601), 마커들 (650), 클라이언트들 (648), 이벤트들 (690), 및 아키텍처 또는 프레임워크 관리기 (440) 에 의해 지원되는 쿼리 기능들 (695) 을 포함한다. 각각의 노드 (601) 는 계란형 형태로, 그리고, 노드 (601) 내 리소스들 사이의 각각의 의존성들을 나타내는 특정의 방향들을 가진 화살표들 (680) 로 예시되어 있다.

[0075] 도 7a 내지 도 7b 에 예시된 노드 아키텍처 내 호출들은 노드 (601) 내 리소스의 별칭, 또는 실제 리소스 이름에 대해 이루어질 수도 있다. 예시적인 일 실시형태에 따르면, 클라이언트들 (648) 과 마커들 (650) 사이에 어떤 인터페이스도 없기 때문에 마커 (650) 에 대해 클라이언트 요청 (675) 을 행할 방법이 없으며, 따라서, 이것은 일반적으로 마커들 (650) 과 교환되는 정보가 대개 노드 (601) 또는 리소스로부터 발신하고 클라이언트 (648) 로부터 발신하지 않는다는 것을 의미한다.

[0076] 예를 들어, 도 7b 의 제 1 노드 (601A) 는 제 1 노드 (601A) 가 제 2 노드 (601B) 의 2개의 리소스들 (리소스들 #2 및 #3) 에 의존한다는 것을 나타내는 의존성 화살표 (680A) 를 갖는다. 이와 유사하게, 제 1 노드 (601A) 는 제 1 노드 (601A) 가 또한 하드웨어 또는 소프트웨어 또는 이들의 조합의 레거시 엘리먼트를 일반적으로 포함하는 제 1 마커 (650) 에 의존한다는 것을 나타내는 의존성 화살표 (680B) 를 갖는다.

[0077] 도 7b 는 또한 어떻게 제 1 노드 (601A) 의 클라이언트 (648) 가 클라이언트 요청 (675) 을 제 1 노드 (601A) 로 발할 수도 있는지를 예시한다. 이들 클라이언트 요청들 (675) 이 발해진 후, 제 2 노드 (601B) 는 이벤트 (690) 를 트리거하거나 또는 쿼리 (695) 에 대한 응답을 제공할 수도 있으며, 여기서, 이벤트 (690) 및 쿼리 (695) 에 대응하는 메시지들이 클라이언트 (648) 로 되돌아 흘러간다.

[0078] 도 7c 는 도 1 의 PCD (100) 의 리소스들을 관리하는 시스템을 위한 소프트웨어 아키텍처 (500B2) 의 제 2 양태의 특정의 다이어그램이다. 도 7c 는 도 7a 의 쿼리 기능들에 대응하는, 클라이언트들 (648), 이벤트들 (690), 및 쿼리 기능들 (695) 뿐만 아니라, 특정적인, 또한 예시적인 리소스 이름들을 가진 노드들 (601) 만을 오직 포함하는 노드 또는 리소스 그래프 (500B2) 를 예시한다. 각각의 노드 (601) 는 계란형 형태로, 그리고 노드 (601) 내 리소스들 사이의 각각의 의존성들을 나타내는 특정의 방향들을 가진 화살표들 (680) 로 예시되어 있다.

[0079] 예를 들어, 제 1 노드 (602) 는 제 1 노드 (602) 가 제 2 노드 (622) 의 3개의 리소스들에 의존한다는 것을 나타내는 의존성 화살표 (680B) 를 갖는다. 이와 유사하게, 제 2 소프트웨어 엘리먼트 (444B) 를 포함하며, 일반적으로 도 7c 에서 참조 문자 "C" 로 지정된 제 3 리소스 "/bus/ahb/sysB/" 는, 이 제 3 리소스 (C) 가 제 4 노드 (646) 의 단일 "/clk/sys/ahb" 리소스에 의존한다는 것을 나타내는 의존성 화살표 (680C) 를 갖는다.

[0080] 도 7c 는 또한 하나 이상의 이벤트들 (690) 또는 쿼리 기능들 (query functions; 695) 을 포함할 수도 있는 노드들 (601) 로부터의 출력 데이터를 예시한다. 쿼리 기능 (695) 은 이벤트 (690) 와 유사하다. 쿼리 기능 (695) 은 고유하거나 또는 고유하지 않을 수도 있는 쿼리 핸들을 가질 수도 있다. 쿼리 기능은 일반적으로 외면적으로 식별되지 않으며, 일반적으로 한 상태를 갖지 않는다. 쿼리 기능 (695) 은 노드 (601) 의 특정한 리소스의 상태를 결정하는데 사용될 수도 있다. 쿼리 기능 (695) 및 이벤트들 (690) 은 설정된 클라이언트 (648) 와 관계들을 가질 수도 있으며, 이들 관계들은 각각의 이벤트 (690) 및 쿼리 기능 (695) 으로부터 정보가 특정한 클라이언트 (648) 로 전달된다는 것을 나타내는 방향 화살표들 (697) 로 표현된다. 도 7c 는 또한 어떻게 도 7c 의 제 2 노드 (622) 가 의존성 화살표 (680D) 를 통해서 제 1 마커 (650) 에 의존하는지를 예시한다.

[0081] 도 7b 내지 도 7c 의 노드 또는 리소스 그래프들 (500B) 은, 메모리 내에 존재하며 프레임워크 관리기 (440) 에 의해 관리되는 관계들, 및 노드들 (601) 을 포함할 수도 있는 관련된 데이터 구조들을 나타낸다. 노드 또는 리소스 그래프 (500B) 는 프레임워크 관리기 (440) 에 의해 관리되는 각각의 엘리먼트들 사이의 관계들을 식별하는데, 그리고 소프트웨어 팀에 의한 분쟁중재에, 유용한 툴로서 프레임워크 관리기 (440) 에 의해 자동적으로 발생될 수 있다.

[0082] 도 7d 는 PCD (100) 의 리소스(들)를 관리하는 소프트웨어 아키텍처를 생성하는 방법 (1000A) 을 예시하는 플로우차트이다. 블록 (1005) 은 PCD (100) 의 리소스들을 관리하는 방법 또는 프로세스 (1000) 의 제 1 루틴이다. 루틴 블록 (1005) 에서, 루틴은 노드 구조 데이터를 수신하기 위해 프레임워크 관리기 (440) 에 의해 실행되거나 수행될 수도 있다. 노드 구조 데이터는 특정한 노드 (601) 가 다른 노드들 (601) 과 가질 수도 있는 의존성들을 약술하는 의존성 어레이를 포함할 수도 있다. 노드 구조 데이터 및 이 루틴 또는 하위방법 (705) 에 관한 추가적인 세부 사항들은 도 8 과 관련하여 아래에서 자세히 설명될 것이다.

[0083] 다음으로, 블록 (1010) 에서, 프레임워크 관리기 (440) 는 블록 (1005) 에서 수신된 노드 구조 데이터의 부분인 의존성 데이터를 검토할 수도 있다. 결정 블록 (1015) 에서, 프레임워크 관리기 (440) 는 노드 구조 데이터가 리프 노드 (leaf node) (601) 를 정의하는지 여부를 결정할 수도 있다. 리프 노드 (601) 는 일반적으로 노드 구조 데이터에 기초하여 생성되는 노드가 임의의 의존성들을 갖지 않는다는 것을 의미한다. 결정 블록

(1015) 에 대한 문의 (inquiry) 가 긍정이면, 현재의 노드를 생성하기 위한 노드 구조 데이터가 임의의 의존성들을 갖지 않는다는 것을 의미하며, 프레임워크 관리기 (440) 는 루틴 블록 (1025) 으로 속행한다.

[0084] 결정 블록 (1015) 에 대한 문의가 부정이면, "아니오" 브랜치에 뒤이어서 결정 블록 (1020) 이 뒤따르며, 여기서, 프레임워크 관리기는 노드 구조 데이터 내 강한 (hard) 의존성들의 모두가 존재하는지 여부를 결정한다. 강한 의존성은 하나를 포함할 수도 있으며, 여기서, 리소스는 그것 없이 존재할 수 없다. 한편, 약한 (soft) 의존성은 하나를 포함할 수도 있으며, 여기서 리소스는 의존적인 리소스를 선택사항적인 단계로서 이용할 수도 있다. 약한 의존성은, 약한 의존성이 존재하지 않을 때에도, 약한 의존성을 갖는 노드 (601) 또는 노드 (601) 의 리소스가 노드 아키텍처 내에서 생성되거나 또는 인스턴스화될 수도 있다는 것을 의미한다. 마커 (650) 는 위에서 설명한 바와 같이 약한 의존성으로서 참조될 수도 있다.

[0085] 약한 의존성의 일 예는 다수의 리소스들을 포함하는 리소스 지향 노드 (601) 에 대한 동작에 중요하지 않은 최적화 특성을 포함할 수도 있다. 프레임워크 관리기 (440) 는 약한 의존성이 생성되지 않은 약한 의존성들을 가지는 그들 노드들 또는 리소스들에 대해서 존재하지 않을지라도 존재하는 모든 강한 의존성들에 대해 노드 또는 리소스를 생성하거나 또는 인스턴스화할 수도 있다. 콜 백 특성은, 약한 의존성이 프레임워크 관리기 (440) 에 이용가능하게 될 때, 프레임워크 관리기 (440) 가 약한 의존성을 참조하는 각각의 콜백에게 약한 의존성들이 현재 이용할 수 있다고 통지할 수 있도록, 약한 의존성을 참조하는데 사용될 수도 있다.

[0086] 결정 블록 (1020) 에 대한 문의가 부정이면, "아니오" 브랜치에 이어서 블록 (1027) 이 뒤따르며, 여기서, 노드 구조 데이터는 메모리와 같은 임시 스토리지에 프레임워크 관리기 (440) 에 의해 저장되며, 프레임워크 관리기 (440) 는 이 비-인스턴스화된 노드와 연관된 콜 백 특성을 생성한다.

[0087] 결정 블록 (1015) 에 대한 문의가 긍정이면, "예" 브랜치에 이어서 루틴 (1025) 이 뒤따르며, 여기서, 노드 (601) 는 루틴 블록 (1005) 에서 수신된 노드 구조 데이터에 기초하여 생성되거나 또는 인스턴스화된다. 루틴 블록 (1025) 의 추가적인 세부 사항들은 도 9 를 참조하여 아래에서 설명된다. 다음으로, 블록 (1030) 에서, 프레임워크 관리기 (440) 는, 다른 노드들 (601) 이 정보를 그 새로 생성된 노드 (601) 로 전송하거나 그로부터 정보를 수신할 수 있도록, 새로 생성된 노드 (601) 를 그의 고유한 리소스 이름(들)을 이용하여 공표한다.

[0088] 이하 도 7d 의 연속 플로우 차트인 도 7e 를 참조하면, 블록 (1035) 에서, 프레임워크 관리기 (440) 는 새로 생성된 노드 (601) 에 의존하는 다른 노드들 (601) 에게, 새로 생성된 노드 (601) 가 인스턴스화되어 있고 정보를 수신하거나 또는 송신할 준비가 되어 있다고 통지한다. 하나의 예시적인 양태에 따르면, 통지들은 의존적인 노드가, 도 7b 의 노드 (601B) 와 유사하게, 생성될 때 즉시 트리거된다, 즉, 통지들은 회귀적으로 수행된다. 따라서, 도 7b 의 노드 (601B) 가 구성되면, 노드 (601A) 가 즉시 통지받는다. 이 통지는 (노드 (601B) 가 노드 (601A) 의 최종 의존성이었기 때문에) 노드 (601A) 가 구성되도록 할 수도 있다. 노드 (601B) 의 구성은 다른 노드들 (601) 로 하여금 통지 받는 등등을 할 수 있게 할 수도 있다. 노드 (601B) 는 노드 (601B) 에 의존하는 최종 리소스가 완료될 때까지 완료되지 않는다.

[0089] 약간 더 복잡한, 제 2 구현예는 통지들의 모두를 별개의 통지 큐 상에 할당한 후, 단일 시점에서 시작하는 큐를 통해서 실행하는 것이다, 즉, 통지들이 반복하여 수행된다. 따라서 도 7b 의 노드 (601B) 가 구성될 때, 노드 (601A) 로의 통지가 리스트 상으로 푸시된다. 그후, 그 리스트가 실행되며, 노드 (601A) 가 통지를 받는다. 이것은 다른 추가적인 노드들 (601) (노드 (601A) 에 더해서, 도 7b 에 미예시) 로의 통지가 동일한 리스트 상에 할당되게 하며, 그후 노드 (601A) 로의 통지가 전송된 후에 그 통지가 전송된다. 노드 (601B) 및 노드 (601A) 와 연관된 모든 작업이 완료되기 이후까지, 다른 노드들 (601) 로의 통지들 (노드 (601A) 로의 통지에 더해) 이 일어나지 않는다.

[0090] 논리적으로, 이들 2개의 구현예들은 동등하나, 구현될 때 상이한 메모리 소비 성질들을 갖는다. 귀납적인 실현은 간단하지만 임의의 양의 스택 공간을 소모할 수 있으며, 이때 스택 소비는 의존성 그래프의 깊이 (depth) 의 함수이다. 반복 구현예는 약간 더 복잡하며 조금 더 많은 정적 메모리 (통지 리스트) 를 필요로 하지만, 스택 사용량은 도 7b 에 예시된 것과 같은 의존성 그래프의 깊이와 관계없이, 일정하다.

[0091] 또한, 블록 (1035) 에서의 노드 생성의 통지는 다른 노드들에 한정되지 않는다. 또한, 별칭 구성을 위해 내부적으로 사용될 수도 있다. 시스템 (500A) 에서의 임의의 엘리먼트는 동일한 메커니즘을 이용하여, 단지 다른 노드들이 아닌, 노드 (또는, 마커) 가 이용가능하게 될 때 통지를 요청할 수도 있다. 노드들 및 비-노드들 양자는 동일한 통지 메커니즘을 이용할 수도 있다.

[0092] 결정 블록 (1040) 에서, 프레임워크 관리기 (440) 는 다른 노드들 (601) 또는 약한 의존성들이 현재의 노드 (601) 의 생성에 기초한 생성 또는 인스턴스화를 위해 현재 해제되어 있는지 여부를 결정한다. 결정 블록 (1040) 은, 일반적으로, 어떤 의존성 관계들 (680) 이 생성 또는 인스턴스화를 최근에 겪은 현재의 노드에 의해 수행되었기 때문에, 리소스들이 생성될 수도 있는지 여부를 결정한다.

[0093] 결정 블록 (1040) 에 대한 문의가 긍정이면, "예" 브랜치에 이어서 루틴 블록 (1025) 이 뒤따르며, 여기서, 막 생성된 노드 (601) 에 의한 의존성의 충족 때문에 해제된 노드 (601) 가 이제 생성되거나 또는 인스턴스화될 수도 있다.

[0094] 결정 블록 (1040) 에 대한 문의가 부정이면, "아니오" 브랜치에 이어서 블록 (1045) 이 뒤따르며, 여기서, 프레임 작업 관리기 (440) 는 도 2 에 예시된 바와 같이, 아키텍처의 엘리먼트들 사이의 통신들을 관리할 수도 있다. 다음으로, 블록 (1050) 에서, 프레임워크 관리기 (440) 는 특정한 리소스와 연관된 리소스 이름들을 이용함으로써 리소스들에 의해 취해진 액션들을 계속해서 로그하거나 또는 기록할 수도 있다. 블록 (1045) 은 리소스들, 노드들 (601), 클라이언트들 (648), 이벤트들 (695), 및 쿼리 기능들 (697) 과 같은, 프레임워크 관리기 (440) 또는 프레임워크 관리기 (440) 에 의해 관리되는 엘리먼트들 중 임의의 엘리먼트에 의해 취해지는 임의의 액션 이후, 프레임워크 관리기 (440) 에 의해 실행될 수도 있다. 블록 (1045) 은 프레임워크 관리기 (440) 가 노드 (601) 의 리소스와 같은 특정한 엘리먼트를 생성한 작자들 (authors) 에 의해 제공되는 그들의 고유 식별자 또는 이름에 따라서 각각의 엘리먼트에 의해 수행되는 액션들을 리스트하는 활동의 실행 로그 (running log) 를 유지할 수도 있는 노드 아키텍처의 또 다른 양태를 나타낸다.

[0095] 종래기술에 비해, 블록 (1050) 에서, 시스템의 각각의 리소스에 할당된 고유한 이름들을 리스트하는 이 활동의 로깅은 고유하며, 디버깅 및 에러 트러블슈팅 (error troubleshooting) 에 사용되는 것과 같이 현저한 이점들을 제공할 수도 있다. 노드 아키텍처 (500A) 의 또 다른 고유한 양태는, 별개의 팀들이 상이한 하드웨어 및/또는 소프트웨어 엘리먼트들 상에서, 서로 독립적으로, 작업할 수도 있다는 것이며, 여기서, 각각의 팀은 다른 팀들 및/또는 OEM (original equipment manufacturer) 에 의해 할당된, 덜 의미 있고 그리고 대개 혼란시키는 리소스 이름들을 변환하기 위해 테이블들을 생성할 필요 없이, 고유하며 추적하는데 용이한 리소스 이름들을 사용할 수 있을 것이다.

[0096] 다음으로, 결정 블록 (1055) 에서, 프레임워크 관리기 (440) 는 프레임워크 관리기 (440) 에 의해 기록된 활동의 로그가 요청되었는지 여부를 결정한다. 결정 블록 (1055) 에 대한 문의가 부정이면, "아니오" 브랜치에 이어서 프로세스의 종료가 뒤따르며, 여기서, 프로세스는 루틴 (1005) 으로 되돌아 간다. 결정 블록 (1055) 에 대한 문의가 긍정이면, "예" 브랜치에 이어서 블록 (1060) 이 뒤따르며, 여기서, 프레임워크 관리기 (440) 는 의미 있는 리소스 이름들 및 그 리소스 이름들에 의해 수행되는 각각의 액션들을 포함하는 활동 로그를, 프린터 또는 디스플레이 스크린 및/또는 양자와 같은 출력 디바이스로 전송한다. 프로세스는 그후 위에서 설명한 루틴 블록 (1005) 으로 되돌아 간다.

[0097] 도 8 은 PCD (100) 의 소프트웨어 아키텍처에서 노드 구조 데이터를 수신하는 도 7d 의 하위-방법 또는 루틴 (1005) 을 예시하는 플로우차트이다. 블록 (1105) 은 도 7d 의 하위 방법 또는 루틴 (1005) 에서 제 1 단계이다. 블록 (1105) 에서, 프레임워크 관리기 (440) 는 소프트웨어 또는 하드웨어 엘리먼트, 예컨대 도 7d 의 CPU (110) 및 클록 (442) 에 대한 고유한 이름을 수신할 수도 있다. 전술한 바와 같이, 노드 (601) 는 적어도 하나의 리소스를 참조해야 한다. 각각의 리소스는 시스템 (500A) 에서 고유한 이름을 갖는다. 시스템 (500A) 내 각각의 엘리먼트는 고유한 이름으로 식별될 수도 있다. 각각의 엘리먼트는 문자 관점 (character perspective) 에서 고유한 이름을 갖는다. 즉, 일반적으로, 시스템 (500A) 내에 동일한 이름을 갖는 어떤 2개의 엘리먼트들도 존재하지 않는다. 시스템의 예시적인 양태들에 따르면, 노드들 (601) 의 리소스들은 일반적으로 시스템 전반에 걸쳐서 고유한 이름들을 가질 수도 있으며, 그러나, 클라이언트 또는 이벤트 이름들이, 원하는 바에 따라 고유할 수도 있지만, 고유할 것을 필요로 하지는 않는다.

[0098] 편의를 위해, 고유한 이름들을 생성하기 위해 순방향 슬래시 "/" 문자들을 채용하는 종래의 트리 파일 명명 (naming) 구조 또는 파일 명명 "메타포" 가 예컨대, CPU (110) 에 대해 "/core/cpu" 및 클록 (442) 에 대해 "/clk/cpu" 과 같이, 채용될 수도 있지만 이에 한정되지 않는다. 그러나, 당업자가 주지하고 있는 바와 같이, 영숫자 문자들 및/또는 심볼들의 임의의 다른 조합을 포함하는 다른 유형들의 리소스 이름들은 완전히 본 발명의 범위 이내이다.

[0099] 다음으로, 블록 (1110) 에서, 프레임워크 관리기 (440) 는 생성되고 있는 노드 (601) 의 하나 이상의 리소스들과 연관된 하나 이상의 드라이버 기능들에 대한 데이터를 수신할 수도 있다. 드라이버 기능은 일반적으로

특정한 노드 (601) 에 대한 하나 이상의 리소스들에 의해 완료되는 액션을 포함한다. 예를 들어, 도 7a 내지 도 7b 에서, 노드 (602) 의 리소스 /core/cpu 에 대한 드라이버 기능은 요청되어진 프로세싱의 요청된 양을 제공하기 위해 필요로 하는 버스 대역폭의 양 및 CPU 클록 주파수를 요청할 수도 있다. 이들 요청들은 노드들 (642) 및 노드 (622) 에서 리소스들의 클라이언트들 (예컨대, 도 2 에서 클라이언트들 (201, 202, 204 및 206)) 을 통해서 이루어질 것이다. 노드 (642) 에서의 /clk/cpu 에 대한 드라이버 기능은 노드 (602) 의 /core/cpu 리소스로부터 수신된 요청에 따라서 물리적인 클록 주파수를 실제로 설정하는 것을 대개 담당할 것이다.

[0100] 블록 (1115) 에서, 프레임워크 관리기 (440) 는 노드 속성 데이터를 수신할 수도 있다. 노드 속성 데이터는 (노드가 사용자 공간 애플리케이션들을 통해서 액세스될 수 있는) 보안성, (노드가 시스템 내 다른 프로세서들로부터 액세스될 수 있는) 원격성 (remotability), 및 (리소스가 다수의 병행 클라이언트들을 지원할 수 있는) 접근성과 같은, 노드 정책들을 정의하는 데이터를 포함한다. 프레임워크 관리기 (440) 는 또한 리소스로 하여금, 요청 평가 또는 로깅 정책과 같은, 디폴트 프레임워크 거동을 오버라이드가능하게 하는 속성들을 정의할 수도 있다.

[0101] 후속하여, 블록 (1120) 에서, 프레임워크 관리기 (440) 는 생성중인 특정한 노드 (601) 에 대한 커스터마이즈된 사용자 데이터를 수신할 수도 있다. 사용자 데이터는 "C" 프로그래밍 언어에 대해 당업자가 주지하고 있는 바와 같이, 보이드 (void) "star" 필드를 포함할 수도 있다. 사용자 데이터는 또한 "trust me" 필드로서 당업자에게 알려져 있다. 예시적인 커스터마이즈된 사용자 데이터는 주파수 테이블들, 레지스터 맵들 등과 같은, 테이블들을 포함할 수도 있지만 이에 한정되지 않는다. 블록 (1120) 에서 수신된 사용자 데이터는, 커스터마이제이션이 프레임워크 관리기 (440) 에 의해 인식되거나 또는 전적으로 지원되지 않으면, 시스템 (500A) 에 의해 참조되지 않으나, 리소스의 커스터마이제이션을 고려한다. 이 사용자 데이터 구조는 특정한 또는 구체적인 사용들을 위해 확장시키려고 의도된 "C" 프로그래밍 언어에서 베이스 클래스이다.

[0102] 당업자는 특정 클래스의 구체적인 사용들을 확장하는 다른 종류의 데이터 구조들도 본 발명의 범위 이내 임을 알 수 있다. 예를 들어, "C++" (C-플러스-플러스) 의 프로그래밍 언어에서, 등가 구조는 노드 (601) 내 리소스에 대해 확장 메커니즘이 될 키 워드 (key word) "public" 을 포함할 수도 있다.

[0103] 다음으로, 블록 (1125) 에서, 프레임워크 관리기 (440) 는 의존성 어레이 데이터를 수신할 수도 있다. 의존성 어레이 데이터는 생성중인 노드 (601) 가 의존하는 하나 이상의 리소스들 (601) 의 고유한 및 특정적인 이름들을 포함할 수도 있다. 예를 들어, 도 7c 의 제 1 노드 (602) 가 생성중이었으면, 이 블록 (1125) 에서, 의존성 어레이 데이터는 제 1 노드 (602) 가 의존하는, 제 2 노드 (622) 의 3개의 리소스들의 리소스 이름들 및 제 3 노드 (642) 의 단일 리소스 이름을 포함할 수도 있다.

[0104] 후속하여, 블록 (1130) 에서, 프레임워크 관리기 (440) 는 리소스 어레이 데이터를 수신할 수도 있다. 리소스 어레이 데이터는 이 제 1 노드 (602) 가 생성중이었으면, 도 7b 내지 도 7c 의 제 1 노드 (602) 에 관련된 파라미터들과 같은, 생성중인 현재의 노드에 대한 파라미터들을 포함할 수도 있다. 리소스 어레이 데이터는 다음 데이터 중 하나를 포함할 수도 있다: 다른 리소스들의 이름들; 유닛; 최대 값; 리소스 속성들; 플러그-인 데이터; 및 블록 (1120) 의 커스터마이즈 사용자 데이터와 유사한 임의의 커스터마이즈된 리소스 데이터. 플러그-인 데이터는 일반적으로 소프트웨어 라이브러리로부터 취출된 기능들을 식별하고, 생성중인 특정한 노드 또는 복수의 노드들에 의해 지원될 수도 있는 클라이언트 유형들을 일반적으로 리스트한다. 플러그인 데이터는 또한 클라이언트 생성 및 파괴의 커스터마이제이션을 고려한다. 블록 (1130) 이후, 프로세스는 도 7d 의 블록 (1010) 으로 되돌아간다.

[0105] 도 8 에서, 속성 데이터 블록 (1115), 커스터마이즈 사용자 데이터 블록 (1120), 및 의존성 어레이 데이터 블록 (1125) 은 이들 특정한 단계들이 선택사항적이며 임의의 주어진 노드 (601) 에 대해 요구되지 않는다는 것을 나타내기 위해 파선들로 예시되어 있다. 한편, 고유한 이름 블록 (1105), 드라이버 기능 블록 (1110), 및 리소스 어레이 데이터 블록 (1130) 은 루틴 (1005) 의 이들 단계들이 노드 (601) 를 생성하는데 일반적으로 필수적임을 나타내기 위해 실선들로 예시되어 있다.

[0106] 도 9 는 PCD (100) 에 대한 소프트웨어 아키텍처에서 노드를 생성하는 도 7d 의 하위-방법 또는 루틴 (1025) 을 예시하는 플로우차트이다. 루틴 블록 (1205) 은 하나의 예시적인 실시형태에 따라 노드 (601) 를 인스턴스화하거나 또는 생성하는 하위-방법 또는 루틴 (1025) 에서 제 1 루틴이다. 루틴 블록 (1205) 에서, 인스턴스화중인 노드 (601) 와 연관되는 하나 이상의 클라이언트들 (648) 이 이 단계에서 생성된다. 루틴 블록 (1205) 에 관한 추가적인 세부 사항들은 도 10 과 관련하여 아래에서 좀더 자세히 설명된다.

[0107] 블록 (1210) 에서, 프레임워크 관리기는 블록 (705) 의 노드 구조 데이터에 대응하는 하나 이상의 리소스들을 생성하거나 또는 인스턴스화할 수도 있다. 다음으로, 블록 (1215) 에서, 프레임워크 관리기 (440) 는 루틴 블록 (1005) 의 루틴 블록 (1110) 에서 수신된 드라이버 기능들을 활성화할 수도 있다. 하나의 예시적인 양태에 따르면, 드라이버 기능들은 루틴 블록 (1005) 의 리소스 어레이 데이터 블록 (1130) 에서 수신된 최대 값들을 이용하여 활성화될 수도 있다. 또 다른, 바람직한, 예시적인 양태에 따르면, 각각의 드라이버 기능은 루틴 (1005) 으로부터 노드 구조 데이터와 함께 전달된 선택사항적인, 초기 값으로 활성화될 수도 있다. 초기 데이터가 제공되지 않으면, 드라이버 기능은 0 내지 최소 값으로 초기화된다. 드라이버 기능은 또한 초기화중인 것이 알려지는 방법으로 대개 활성화된다. 이것은 리소스로 하여금 초기화에 고유한 임의의 동작들을 수행가능하게 하지만, 정상 또는 루틴 동작 동안 수행될 필요가 없다. 프로세스는 그후 도 7d 의 단계 (1030) 로 되돌아간다.

[0108] 도 10 은 PCD (100) 의 소프트웨어 아키텍처에서 클라이언트 (648) 를 생성하는 도 9 의 하위-방법 또는 루틴 (1205) 을 예시하는 플로우차트이다. 블록 (1305) 은 루틴 블록 (1205) 의 제 1 단계이며, 여기서, 하나 이상의 리소스들 (601) 의 클라이언트 (648) 가 생성된다. 블록 (1205) 에서, 프레임워크 관리기 (440) 는 생성중인 클라이언트 (648) 에 할당된 이름을 수신한다. 리소스 이름들과 유사하게, 클라이언트 (648) 에 대한 이름은 임의 종류의 영숫자 및/또는 심볼들을 포함할 수도 있다.

[0109] 다음으로, 블록 (1310) 에서, 커스터마이즈된 사용자 데이터가 이 생성중인 클라이언트 (648) 에 대해 임의의 특정한 커스터마이제이션들이 있으면, 프레임워크 관리기 (440) 에 의해 수신될 수도 있다. 블록 (1310) 은 단계가 선택사항적임을 나타내기 위해 파선들로 예시되어 있다. 블록 (1310) 의 커스터마이즈된 사용자 데이터는 노드들 (601) 에 대한 리소스들의 생성과 관련하여 위에서 설명한 커스터마이즈된 사용자 데이터와 유사하다.

[0110] 블록 (1315) 에서, 프레임워크 관리기 (440) 는 생성중인 특정한 클라이언트에 할당된 클라이언트 유형 카테고리를 수신한다. 이 작성 중인 현재, 클라이언트 유형 카테고리는 다음 4개의 유형들 중 하나를 포함할 수도 있다: (a) 요구된 (required), (b) 임펄스 (impulse), (c) 벡터 (vector), 및 (d) 등시성 (isochronous). 클라이언트 유형 카테고리 리스트는 시스템 (101) 에 의해 관리중인 리소스들에, 그리고, 노드들 (601) 의 리소스들에 의존하는 애플리케이션 프로그램들에 따라서 확장될 수도 있다.

[0111] 요구된 카테고리는 일반적으로 그 요구된 클라이언트 (648) 로부터 특정한 리소스 (601) 로 전달되는 스칼라 값의 프로세싱에 대응한다. 예를 들어, 요구된 요청은 일정수의 MIPs (millions of instructions per second) 를 포함할 수도 있다. 한편, 임펄스 카테고리는 일반적으로 시작 시간 또는 정지 시간의 임의의 지정 없이, 어떤 시간 기간 내에 일부 활동을 완료하라는 요청의 프로세싱에 대응한다.

[0112] 등시성 카테고리는 일반적으로 일반적으로 재발생하고 있고 잘-정의된 시작 시간 및 잘-정의된 종료 시간을 갖는 액션에 대한 요청에 대응한다. 벡터 카테고리는 일반적으로 일반적으로 직렬로 또는 병렬로 요구되는 다수의 액션들의 부분인 데이터의 어레이에 대응한다.

[0113] 후속하여, 블록 (1320) 에서, 프레임워크 관리기 (440) 는 클라이언트 (648) 가 동기적 또는 비동기적으로 지정되어 있는지 여부를 나타내는 데이터를 수신한다. 동기적 클라이언트 (648) 는 프레임워크 관리기 (440) 에게, 리소스 (601) 가 동기적 클라이언트 (648) 로부터의 그 요청된 작업을 완료하는 것을 종료하였다는 데이터 및 표시를 반송할 때까지, 노드 (601) 의 리소스를 잠그도록 일반적으로 요구하는 클라이언트이다.

[0114] 한편, 비동기적 클라이언트 (648) 는 프레임워크 관리기 (440) 에 의해 액세스되는 하나 이상의 쓰레드들에 의해 병렬로 처리될 수도 있다. 프레임워크 관리기 (440) 는 쓰레드에 대한 콜백을 생성할 수도 있으며, 콜백이 각각의 쓰레드에 의해 실행되었을 때 값을 반환할 수도 있다.

[0115] 블록 (1320) 이후, 결정 블록 (1325) 에서, 프레임워크 관리기 (440) 는 클라이언트 (645) 에 의해 식별되는 리소스가 이용가능한지 여부를 결정한다. 결정 블록 (1325) 에 대한 문의가 부정이면, "아니오" 브랜치에 이어서 블록 (1330) 이 뒤따르며, 여기서, 클라이언트 (648) 가 이때에 생성될 수 없다는 것을 나타내는 널 값 또는 메시지가 사용자에게 반송된다.

[0116] 결정 블록 (1325) 에 대한 문의가 긍정이면, "예" 브랜치에 이어서 결정 블록 (1335) 이 뒤따르며, 여기서, 프레임워크 관리기 (440) 는 클라이언트 (648) 에 의해 식별되는 각각의 리소스가 블록 (1310) 에서 제공된 클라이언트 유형을 지원하는지 여부를 결정한다. 결정 블록 (1335) 에 대한 문의가 부정이면, "아니오" 브랜치에 이어서 블록 (1330) 로 되돌아 가며, 여기서, 클라이언트 (648) 가 이때에 생성될 수 없다는 것을 나타내는

널 값 또는 메시지가 반송된다.

[0117] 결정 블록 (1335) 에 대한 문의가 긍정이면, "예" 브랜치에 이어서 블록 (1340) 이 뒤따르며, 여기서, 프레임워크 관리기 (440) 는 메모리에 클라이언트 (648) 를 생성하거나 또는 인스턴스화한다. 다음으로, 블록 (1345) 에서, 선택사항적인 인수들 (arguments) 과 같은, 임의의 커스터마이즈된 사용자 데이터가 블록 (1310) 에서 수신되면, 이들 선택사항적인 인수들은 그들 각각의 리소스들 및 특정한 노드들 (601) 과 맵핑될 수도 있다. 다음으로, 블록 (1350) 에서, 새로 생성된 클라이언트 (645) 가 위에서 설명한 도 10c 에서 예시된 바와 같이 휴지 상태로 또는 요청된 상태로 그의 대응하는 하나 이상의 리소스들에 커플링된다. 프로세스는 그후 도 12 의 블록 (1210) 으로 되돌아간다.

[0118] 도 11 은 PCD (100) 에 대한 소프트웨어 아키텍처에서 리소스 (601) 에 대해 클라이언트 요청 (675) 을 생성하는 방법 (1400) 을 예시하는 플로우 차트이다. 방법 (1400) 은 일반적으로 도 7d 내지 도 7e 및 도 10 과 관련하여 위에서 설명한 바와 같이 클라이언트 생성 및 노드 생성 후 실행된다.

[0119] 블록 (1405) 은 리소스 (601) 에 대해 클라이언트 요청 (675) 을 생성하는 방법 (1400) 에서 제 1 단계이다. 이 방법 (1400) 은 어떻게 다음 3개의 유형, 즉, (a) 요구된, (b) 임펄스, 및 (c) 벡터의 요청들 (675) 이 프레임워크 관리기 (440) 에 의해 처리되는지를 기술할 것이다. 위에서 언급한 요청들 (675) 의 이름들이 시사하듯이, 클라이언트 요청들 (675) 은 일반적으로 위에서 생성되고 설명된 클라이언트 유형들에 대응한다.

[0120] 블록 (1405) 에서, 프레임워크 관리기 (440) 는 위에서 언급한 3개, 즉 (a) 요구된, (b) 임펄스, 및 (c) 벡터 중 하나와 같은, 특정한 클라이언트 요청 (675) 과 연관된 데이터를 수신할 수도 있다. 요구된 요청과 연관된 데이터는 일반적으로 그 요구된 클라이언트 (648) 로부터 특정한 리소스 (601) 로 전달되는 스칼라 값을 포함한다. 예를 들어, 요구된 요청은 일정수의 MIPs (millions of instructions per second) 를 포함할 수도 있다. 한편, 임펄스 요청은 시작 시간 또는 정지 시간의 임의의 지정 없이, 어떤 시간 기간 내에 일부 활동을 완료하라는 요청을 포함한다. 벡터 요청에 대한 데이터는 일반적으로 직렬로 또는 병렬로 완료되도록 요구되는 다수의 액션들의 어레이를 포함한다. 벡터 요청은 임의의 길이의 값들을 포함할 수도 있다. 벡터 요청은 대개 사이즈 값 및 값들의 어레이를 갖는다. 노드 (601) 의 각각의 리소스는 벡터 요청을 지원하기 위해 포인터 필드 (pointer field) 를 갖도록 확장될 수도 있다. "C" 프로그래밍 언어에서, 포인터 필드는 당업자에 의해 주지되는 바와 같이 집합 기능 (union function) 에 의해 지원받는다.

[0121] 다음으로, 블록 (1410) 에서, 프레임워크 관리기 (440) 는 그 요청을, 도 10 과 관련하여 위에서 설명한 방법에 의해 생성된 클라이언트 (648) 를 통해서 발한다. 후속하여, 블록 (1415) 에서, 프레임워크 관리기 (440) 는 그 요청이 요구된 유형 또는 벡터 유형이면, 클라이언트를 통과중인 그 요청 데이터를 이중 버퍼링한다. 그 요청이 임펄스 유형이면, 블록 (1415) 은 프레임워크 관리기 (440) 에 의해 스킵된다.

[0122] 요구된 요청들에 대해, 이 블록 (1415) 에서, 이전 요청으로부터의 값들은 프레임워크 관리기 (440) 가 요청된 값들의 현재의 세트에서 이전 요청된 값들 사이에 어떤 차이가 있는지 여부를 결정할 수 있도록, 메모리에 유지된다. 벡터 요청들에 있어, 이전 요청들은 일반적으로 메모리에 유지되지 않지만, 노드 (601) 의 리소스는 특정한 구현예에 대해 원하는 바에 따라 유지할 수도 있다. 따라서, 블록 (1415) 은 요청들의 벡터 유형들에 대해 선택사항적이다.

[0123] 블록 (1420) 에서, 프레임워크 관리기 (440) 는 요청된 값들의 현재의 세트에서 요청된 값들의 이전 세트 사이의 델타 (delta) 또는 차이를 계산한다. 결정 블록 (1425) 에서, 프레임워크 관리기는 요청된 값들의 현재의 세트가 요청된 값들의 이전 세트에 동일한지 여부를 결정한다. 즉, 프레임워크 관리기 (440) 는 요청된 값들의 현재의 세트와 요청된 값들의 이전 세트 사이에 차이가 존재하는지 여부를 결정한다. 요청된 값들의 현재의 세트와 이전 세트 사이에 어떤 차이도 없으면, "예" 브랜치에 이어서 (블록 (1430) 내지 블록 (1470) 을 건너 뛰고) 블록 (1475) 이 뒤따르며, 여기서 프로세스가 종료한다.

[0124] 결정 블록 (1425) 에 대한 문의가 부정이면, 요청된 값들의 세트가 사전-이전 요청된 값들의 세트에 대해 상이하다는 것을 의미하며, "아니오" 브랜치에 이어서 결정 블록 (1430) 이 뒤따른다.

[0125] 결정 블록 (1430) 에서, 프레임워크 관리기 (440) 는 현재의 요청이 비동기적 요청인지 여부를 결정한다. 결정 블록 (1430) 에 대한 문의가 부정이면, "아니오" 브랜치에 이어서 블록 (1440) 이 뒤따르며, 여기서, 클라이언트 요청 (675) 에 대응하는 리소스 (601) 가 프레임워크 관리기 (440) 에 의해 잠겨진다. 결정 블록 (1430) 에 대한 문의가 긍정이면, 현재의 요청이 비동기적 요청 유형임을 의미하며, "예" 브랜치에 이어서 블록 (1435) 이 뒤따르며, 여기서, 요청이 또 다른 쓰레드 상으로 푸시될 수도 있으며, 도 1 의 것과 유사한, 멀티코

어 시스템이 프레임워크 관리기 (440) 에 의해 현재 관리되면, 또 다른 코어에 의해 실행될 수도 있다. 블록 (1435) 은, PCD (100) 가 단일 코어 중앙 프로세싱 시스템이면, 이 단계가 선택사항적일 수도 있다는 것을 나타내기 위해 파선들로 예시되어 있다.

[0126] 후속하여, 블록 (1440) 에서, 요청 (675) 에 대응하는 리소스들 (601) 은 프레임워크 관리기 (440) 에 의해 잠겨진다. 다음으로, 블록 (1445) 에서, 리소스 (601) 는 일반적으로 도 8 의 블록 (1130) 에서 수신된 리소스 어레이 데이터의 플러그-인 데이터에 대응하는 업데이트 기능을 실행한다. 업데이트 기능은 일반적으로 새로운 클라이언트 요청을 고려하여 새로운 리소스 상태를 담당하는 기능을 포함한다. 업데이트 기능은 클라이언트 요청에서 그의 이전 상태를 요청된 상태와 비교한다. 그 요청된 상태가 이전 상태보다 더 크면, 업데이트 기능은 클라이언트 요청을 수행할 것이다. 그러나, 그 요청된 상태가 현재의 상태와 동일하거나 더 작고 리소스가 그 요청된 상태에서 동작 중이면, 구 (old) 상태가 요청된 상태를 달성하거나 또는 만족하므로 효율을 증가시키기 위해 클라이언트 요청은 수행되지 않을 것이다. 업데이트 기능은 클라이언트로부터 새로운 요청을 취하여 모든 다른 활성 요청들과 종합하여, 그 리소스에 대한 새로운 상태를 결정한다.

[0127] 일 예로서, 다수의 클라이언트들이 버스 클록 주파수를 요청하고 있을 수도 있다. 버스 클록에 대한 업데이트 기능은 일반적으로 모든 클라이언트 요청들의 최대치를 취하고 그것을 버스 클록에 대한 새로운 원하는 상태로서 이용할 것이다. 다수의 리소스들에 의해 이용될 일부 업데이트 기능들이 있지만, 모든 리소스들이 동일한 업데이트 기능을 이용할 것이라는 것은 사실이 아니다. 일부 공통 업데이트 기능들은 클라이언트 요청들의 최대치를 취하고, 클라이언트 요청들의 최소치를 취하고, 클라이언트 요청을 합할 것이다. 또는, 리소스들은 그들의 리소스가 요청들을 어떤 고유한 방법으로 집합할 필요가 있으면, 그들 자신의 맞춤 업데이트 기능을 정의할 수도 있다.

[0128] 다음으로, 블록 (1450) 에서, 프레임워크 관리기 (440) 는 리소스가 노드 (601) 의 리소스에 고유한 드라이버 기능을 실행할 수 있도록, 데이터를 클라이언트 (648) 에 대응하는 리소스로 전달한다. 드라이버 기능은 리소스 상태를 업데이트 기능에 의해 계산된 것처럼 적용한다. 이것은 하드웨어 설정들을 업데이트하고, 의존적인 리소스들에 대한 요청들을 발하고, 레거시 기능들 또는 상술한 것의 일부 조합을 호출하는 것을 수반할 수도 있다.

[0129] 이전 예에서는, 업데이트 기능이 요청된 버스 클록 주파수를 계산하였다. 드라이버 기능은 그 요청된 주파수를 수신할 수도 있으며, 그 주파수에서 실행하기 위해 클록 주파수 제어 HW 를 업데이트할 수도 있다. 종종 업데이트 기능이 계산되었던 정확한 요청된 상태를 드라이버 기능이 만족시키는 것이 불가능하다는 점에 유의한다. 이 경우, 드라이버 기능은 그 요청에 최선으로 만족시키는 주파수를 선택할 수도 있다. 예를 들어, 버스 클록 HW 은 오직 128 MHz 및 160 MHz 에서 실행할 수도 있지만, 그 요청된 상태는 150 MHz 일지도 모른다. 이 경우, 드라이버 기능은 그것이 그 요청된 상태를 초과하므로, 160 MHz 에서 실행해야 한다.

[0130] 다음으로, 블록 (1455) 에서, 프레임워크 (440) 는 블록 (1450) 에서 드라이버 기능을 실행했던 리소스로부터 상태 제어를 수신한다. 후속하여, 블록 (1460) 에서, 리소스에 대해 정의되면, 이벤트들 (690) 은 데이터가 이벤트 (690) 에 대응하는 클라이언트 (648) 로 되돌아 전달되도록, 트리거될 수도 있다. 이벤트들은 또 다른 쓰레드에서 프로세싱될 수도 있다. 이것은 리소스들이 잠겨지는데 소요되는 시간의 양을 최소화할 수도 있으며, 도 1 에 예시된 바와 같은 멀티코어 시스템에서 병렬 동작을 가능하게 한다. 하나 이상의 이벤트들 (690) 은 요청이 이 방법 (1400) 에서 설명되는 바와 같이 리소스에 대해 정의될 수도 있는 방법과 유사한 방법으로 리소스에 대해 정의될 수도 있다. 즉, 이벤트 생성 프로세스는 클라이언트 생성 프로세스와 대개 병렬일 수도 있다. 이벤트들과 상이한 한가지 사항은 어떤 임계치들이 교차될 때에만 오직 트리거되게 하는 이벤트들을 정의하는 것이 가능하다는 것이다.

[0131] 오직 임계치들에 기초하여 트리거되게 하는 이벤트들의 이 정의는, 시스템 오버로딩 (overloading) 상태를 나타내는, 리소스가 과신청되고 있을 때 (리소스가 지원할 수 있는 사용자들보다 더 많은 병행 사용자들을 갖는다), 또는 리소스가 낮아지거나/오프될 때의 통지를 가능하게 하며, 이것은 다른 것들이 정지될 수 있게 하여, 시스템이 과신청되어질 때 기타 등등에 의해 디스에이블되었던 기능을 복구할 수도 있다. 이벤트 등록이 임계치들과 함께 이루어질 수도 있기 때문에, 실제로 필요한 어떤 것이 있을 때에만 오직 일어나도록 하기 위해 시스템이 이벤트 통지에 대해 행해야 할 작업의 양을 감소시킨다. 또한, 모든 상태 변화에 대한 이벤트를 등록하는 것도 가능하다.

[0132] 다음으로, 선택사항적인 블록 (1465) 에서, 프로세싱되고 있는 요청이 벡터 요청이면, 이 선택사항적인 블록 (1465) 이 일반적으로 수행된다. 선택사항적인 블록 (1465) 은 일반적으로 사용자가 벡터에 전달한 동일한

데이터 상에 벡터 포인터가 여전히 위치되어 있는지 여부를 평가하는 체크 또는 결정을 포함한다. 이 선택사항적인 블록 (1465) 에 대한 문의가 긍정이면, 포인터가 사용자에 의해 벡터로 전달되었던 동일한 데이터를 여전히 가리키고 있다는 것을 의미하며, 구 데이터에 대한 참조들이 유지되지 않도록, 포인터가 비워진다. 이 선택사항적인 블록 (1465) 은 임펄스 요청 및 요구된 요청에 비해, 벡터 요청이 프로세싱되고 있을 때, 위에서 설명한 이중 버퍼링 블록 (1415) 을 고려하기 위해 일반적으로 수행된다.

[0133] 후속하여, 블록 (1470) 에서, 프레임워크 (440) 는 다른 클라이언트 요청들 (648) 이 현재이지만 지금 해제된, 특정한 노드 (601) 의 요청된 리소스에 의해 처리될 수 있도록, 그 요청된 리소스를 해제한다. 프로세스는 그후 다음 클라이언트 요청을 수신하는 제 1 블록 (1405) 으로 되돌아간다.

[0134] 상기 본 개시물을 고려하여, 프로그래밍에서의 당업자는 컴퓨터 코드를 기록하거나 또는 적합한 하드웨어 및/또는 회로들을 식별함으로써, 예를 들어, 본 명세서에서의 플로우 차트들 및 연관된 설명에 기초하여, 어려움 없이 개시된 발명을 구현할 수 있다. 따라서, 프로그램 코드 명령들의 특정한 세트 또는 상세한 하드웨어 디바이스들의 개시물은 본 발명을 실시하고 이용하는 방법의 적절한 이해에 필수적인 것으로 간주되지 않는다. 청구된 컴퓨터 구현된 프로세스들의 신규한 기능성은 상기 설명에서, 그리고 여러 프로세스 흐름들을 예시할 수도 있는 도면들과 함께, 좀더 자세하게 설명된다.

[0135] 하나 이상의 예시적인 양태들에서, 설명된 기능들은 하드웨어, 소프트웨어, 펌웨어, 또는 이들의 임의의 조합으로 구현될 수도 있다. 소프트웨어로 구현되는 경우, 그 기능들은 안에 저장되거나 또는 컴퓨터-판독가능 매체 상의 하나 이상의 명령들 또는 코드로서 송신될 수도 있다. 컴퓨터-판독가능 매체들은 한 장소로부터 또 다른 장소로 컴퓨터 프로그램의 전송을 용이하게 하는 임의의 매체를 포함한, 컴퓨터 저장 매체들 및 통신 매체들 양자를 포함한다. 저장 매체들은 컴퓨터에 의해 액세스될 수도 있는 임의의 가용 매체들일 수도 있다. 일 예로서, 이에 한정하지 않고, 이런 컴퓨터-판독가능 매체들은 RAM, ROM, EEPROM, CD-ROM 또는 다른 광디스크 저장, 자기디스크 저장 또는 다른 자기 저장 디바이스들, 또는 원하는 프로그램 코드를 명령들 또는 데이터 구조들의 형태로 전달하거나 또는 저장하는데 사용될 수도 있으며 컴퓨터에 의해 액세스될 수도 있는 임의의 다른 매체를 포함할 수도 있다.

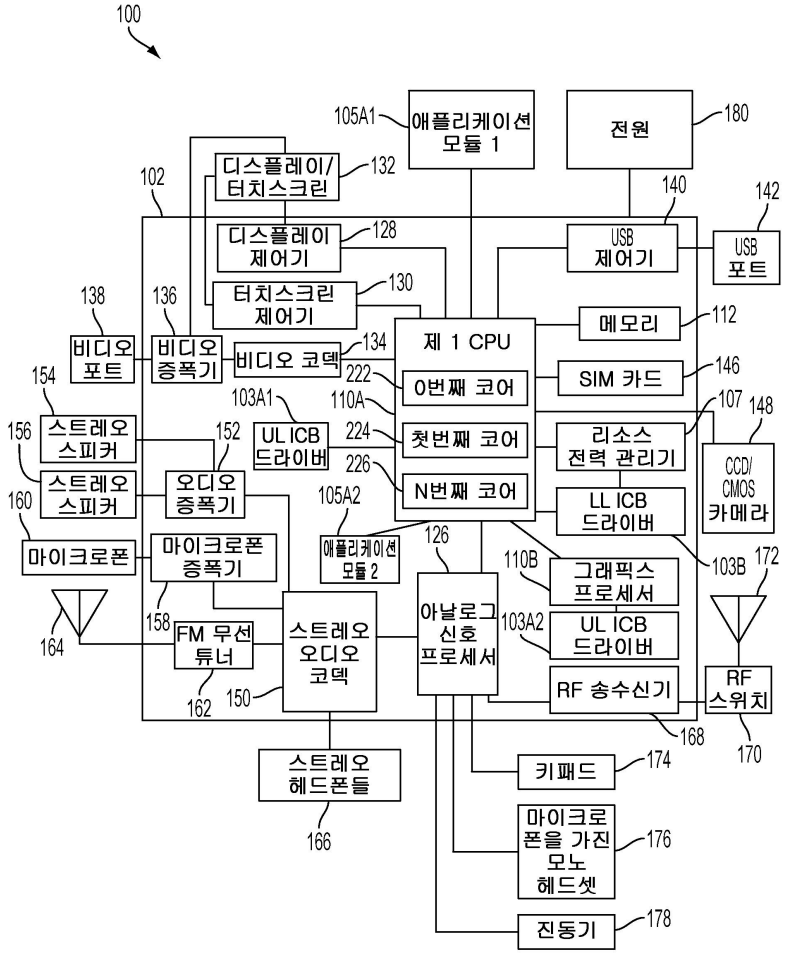
[0136] 또한, 임의의 접속이 컴퓨터-판독가능 매체로 적절히 지칭된다. 예를 들어, 소프트웨어가 웹사이트, 서버, 또는 다른 원격 소스로부터 동축 케이블, 광섬유 케이블, 이중 권선, 디지털 가입자 회선 ("DSL"), 또는 무선 기술들, 예컨대 적외선, 무선, 및 마이크로파를 이용하여 송신되면, 동축 케이블, 광섬유 케이블, 이중 권선, DSL, 또는 무선 기술들, 예컨대 적외선, 무선, 및 마이크로파가 그 매체의 정의에 포함된다.

[0137] 디스크 (disk) 및 디스크 (disc) 는, 본원에서 사용할 때, 컴팩트 디스크 ("CD"), 레이저 디스크, 광 디스크, 디지털 다기능 디스크 ("DVD"), 플로피 디스크 및 블루-레이 디스크를 포함하며, 디스크들 (disks) 은 데이터를 자기적으로 보통 재생하지만, 디스크들 (discs) 은 레이저로 데이터를 광학적으로 재생한다. 앞에서 언급한 것들의 조합들이 또한 컴퓨터-판독가능 매체들의 범위 내에 포함되어야 한다.

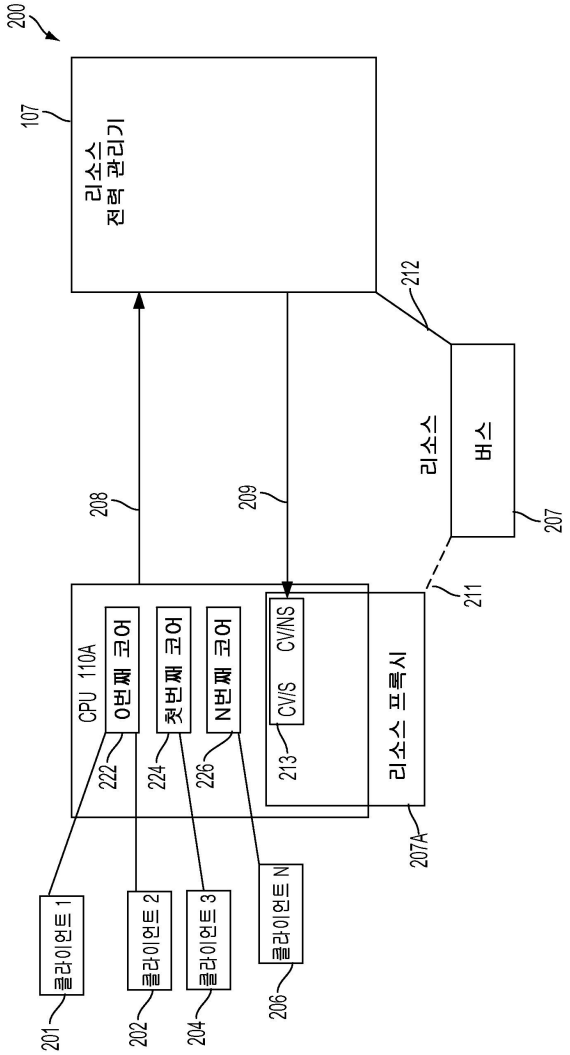
[0138] 선택된 양태들이 자세히 예시 및 설명되었지만, 다음의 청구항들에 의해 정의되는 바와 같이, 본원에서 여러 대체들 및 변경들이 본 발명의 정신 및 범위로부터 일탈함이 없이 이루어질 수도 있는 것으로 이해되어야 할 것이다.

## 도면

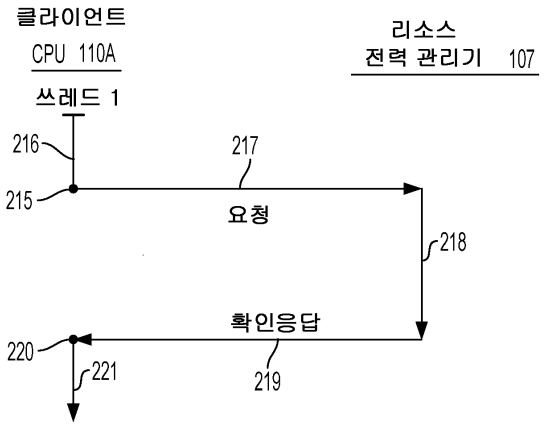
### 도면1


100
105A1
애플리케이션 모듈 1
전원
180
디스플레이/ 터치스크린
132
102
140
142
디스플레이 제어기
128
USB 제어기
USB 포트
터치스크린 제어기
130
138
136
메모리
112
비디오 포트
비디오 증폭기
비디오 코덱
134
제 1 CPU
SIM 카드
146
154
222
0번째 코어
103A1
110A
107
148
156
UL ICB 드라이버
224
첫번째 코어
리소스 전력 관리기
CCD/ CMOS 카메라
152
스트레오 스피커
226
N번째 코어
160
오디오 증폭기
105A2
LL ICB 드라이버
마이크로폰
마이크로폰 증폭기
애플리케이션 모듈 2
126
110B
그래픽스 프로세서
103B
172
164
158
FM 무선 튜너
스트레오 오디오 코덱
아날로그 신호 프로세서
103A2
UL ICB 드라이버
162
150
RF 송수신기
RF 스위치
168
170
스트레오 헤드폰들
키패드
174
166
마이크로폰을 가진 모노 헤드셋
176
진동기
178

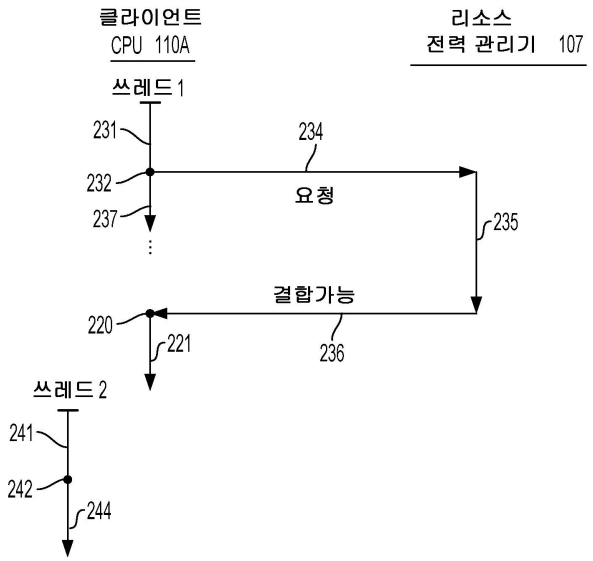
도면2


200
107
리소스 전력 관리기
212
리소스 버스
207
208
209
211
CPU 110A
0번째 코어
첫번째 코어
N번째 코어
CV/IS
CV/INS
리소스 프록시
207A
222
224
226
213
클라이언트 1
클라이언트 2
클라이언트 3
클라이언트 N
201
202
204
206

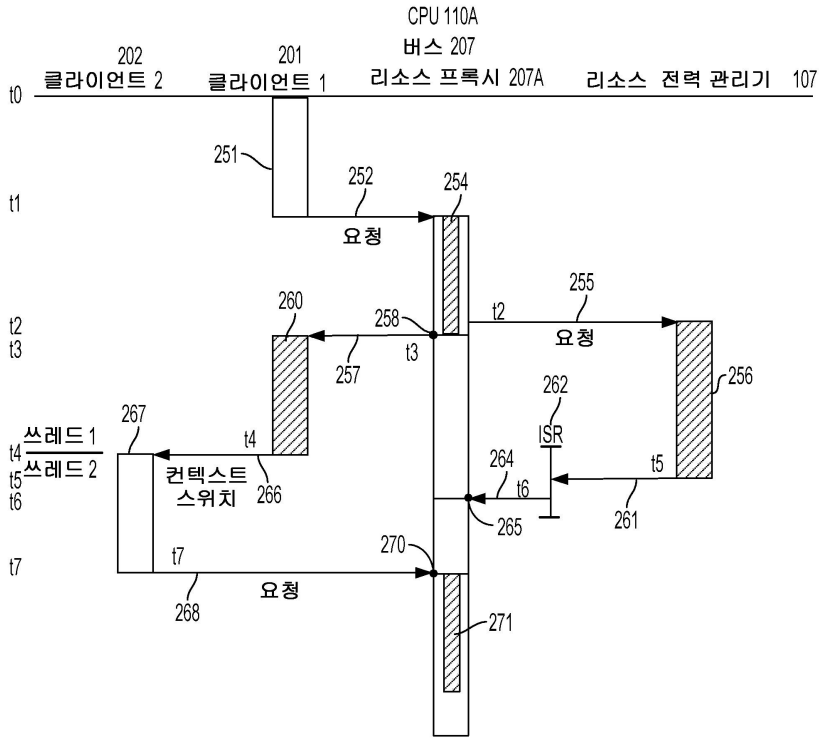
도면3


클라이언트
CPU 110A
쓰레드 1
리소스
전력 관리기 107
216
217
215
요청
218
확인응답
220
221
219

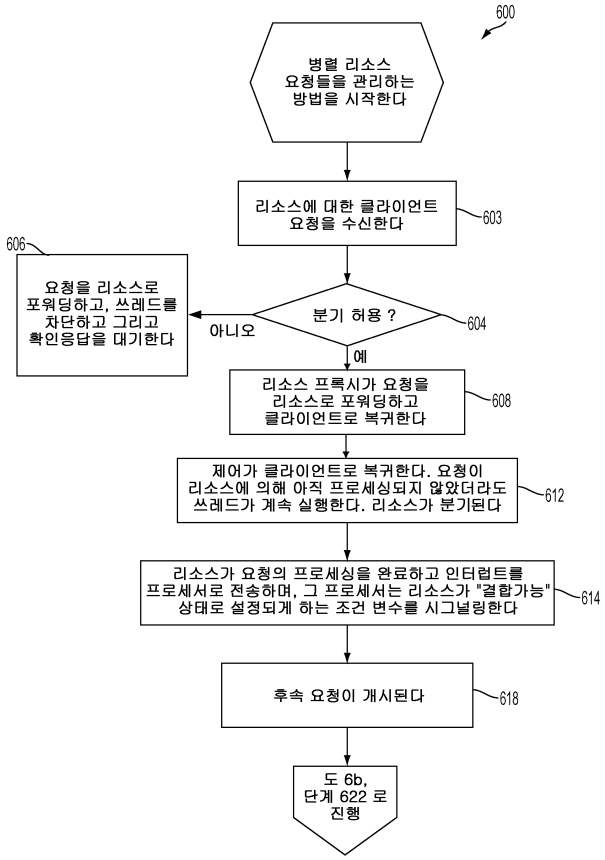
도면4


클라이언트
CPU 110A
리소스
전력 관리기 107
쓰레드 1
231
234
232
요청
237
235
결합가능
220
221
236
쓰레드 2
241
242
244

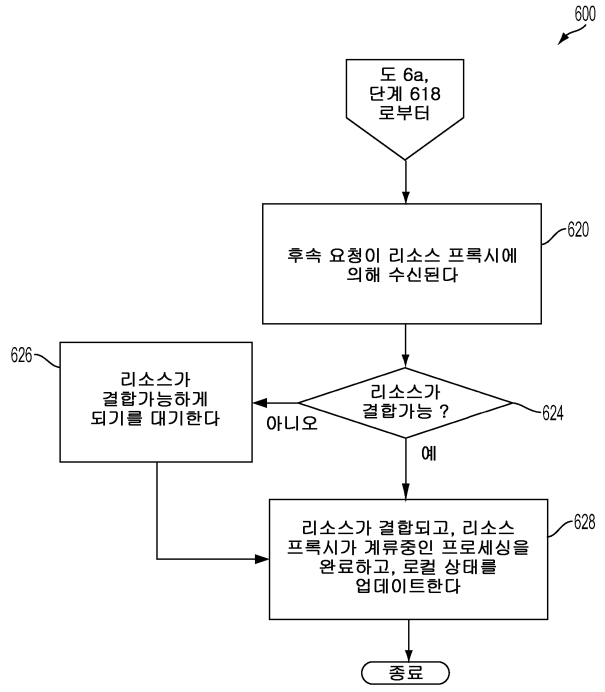
도면5


CPU 110A
버스 207
202
클라이언트 2
201
클라이언트 1
리소스 프록시 207A
리소스 전력 관리기 107
t0
251
252
254
t1
요청
255
260
258
t2
t2
t3
요청
257
t3
256
262
267
ISR
쓰레드 1
t4
t4
t5
t5
264
쓰레드 2
컨텍스트
스위치
266
t6
t6
265
261
270
t7
t7
요청
268
271

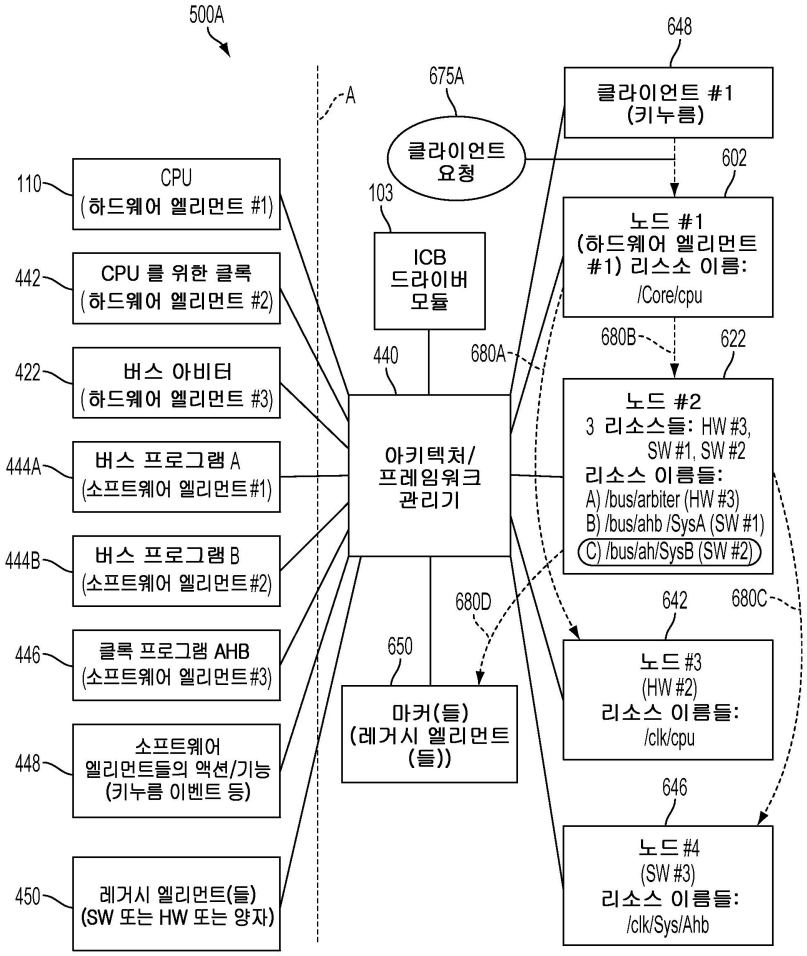
도면6a


600
병렬 리소스 요청들을 관리하는 방법을 시작한다
리소스에 대한 클라이언트 요청을 수신한다
603
606
요청을 리소스로 포워딩하고, 쓰레드를 차단하고 그리고 확인응답을 대기한다
분기 허용 ?
604
아니오
예
리소스 프록시가 요청을 리소스로 포워딩하고 클라이언트로 복귀한다
608
제어가 클라이언트로 복귀한다. 요청이 리소스에 의해 아직 프로세싱되지 않았더라도 쓰레드가 계속 실행한다. 리소스가 분기된다
612
리소스가 요청의 프로세싱을 완료하고 인터럽트를 프로세서로 전송하며, 그 프로세서는 리소스가 "결합가능" 상태로 설정되게 하는 조건 변수를 시그널링한다
614
후속 요청이 개시된다
618
도 6b, 단계 622 로 진행

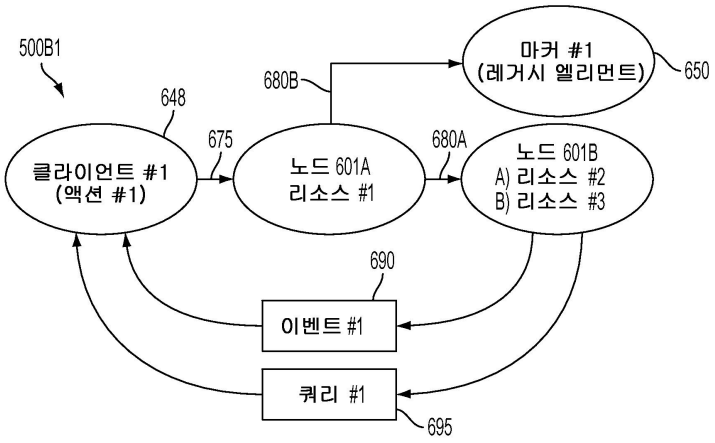
도면6b


600
도 6a, 단계 618 로부터
후속 요청이 리소스 프록시에 의해 수신된다
620
626
리소스가 결합가능하게 되기를 대기한다
리소스가 결합가능 ?
624
아니오
예
리소스가 결합되고, 리소스 프록시가 계류중인 프로세싱을 완료하고, 로컬 상태를 업데이트한다
628
종료

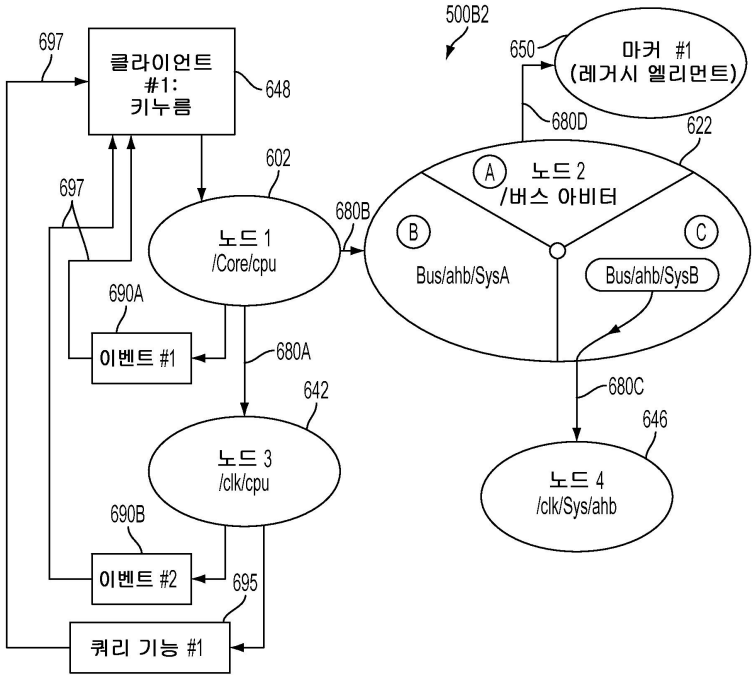
도면7a


500A
A
675A
클라이언트 요청
648
클라이언트 #1 (키누름)
602
110
CPU (하드웨어 엘리먼트 #1)
103
ICB 드라이버 모듈
노드 #1 (하드웨어 엘리먼트 #1) 리소스 이름: /Core/cpu
442
CPU 를 위한 클록 (하드웨어 엘리먼트 #2)
680B
622
422
버스 아비터 (하드웨어 엘리먼트 #3)
440
680A
노드 #2
3 리소스들: HW #3, SW #1, SW #2
리소스 이름들:
A) /bus/arbiter (HW #3)
B) /bus/ahb /SysA (SW #1)
C) /bus/ah/SysB (SW #2)
444A
버스 프로그램 A (소프트웨어 엘리먼트#1)
아키텍처/ 프레임워크 관리기
444B
버스 프로그램 B (소프트웨어 엘리먼트#2)
680D
642
680C
650
446
클록 프로그램 AHB (소프트웨어 엘리먼트#3)
노드 #3 (HW #2) 리소스 이름들: /clk/cpu
마커(들) (레거시 엘리먼트 (들))
646
448
소프트웨어 엘리먼트들의 액션/기능 (키누름 이벤트 등)
노드 #4 (SW #3) 리소스 이름들: /clk/Sys/Ahb
450
레거시 엘리먼트(들) (SW 또는 HW 또는 양자)

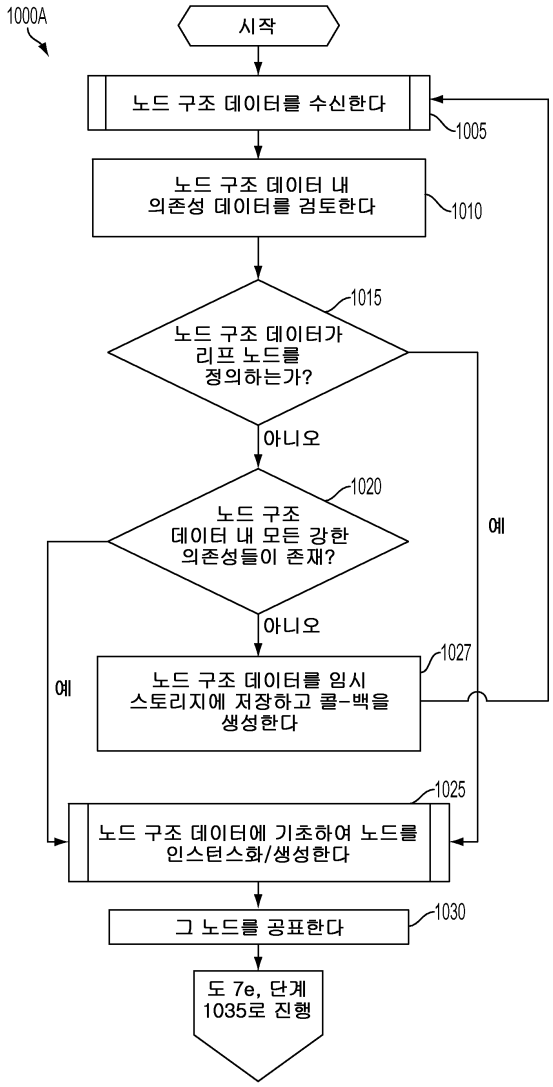
도면7b


500B1
마커 #1 (레거시 엘리먼트)
650
680B
648
675
680A
클라이언트 #1 (액션 #1)
노드 601A 리소스 #1
노드 601B A) 리소스 #2 B) 리소스 #3
690
이벤트 #1
큐리 #1
695

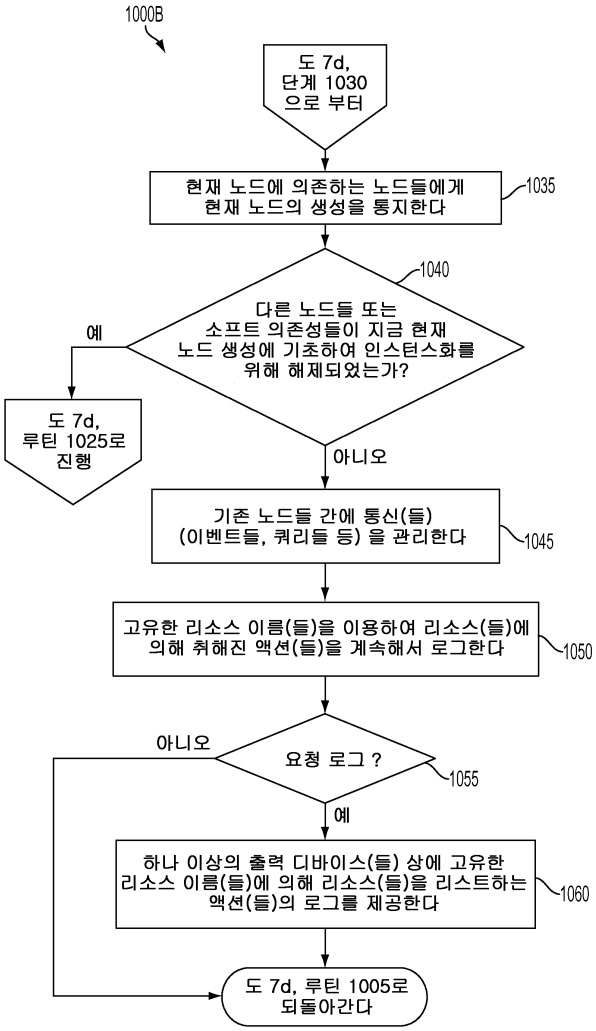
도면7c


500B2
697
클라이언트
#1:
키누름
648
650
마커 #1
(레거시 엘리먼트)
680D
622
602
697
노드 1
/Core/cpu
680B
A
노드 2
/버스 아비터
B
C
Bus/ahb/SysA
Bus/ahb/SysB
690A
이벤트 #1
680A
642
680C
646
노드 3
/clk/cpu
노드 4
/clk/Sys/ahb
690B
이벤트 #2
695
큐리 기능 #1

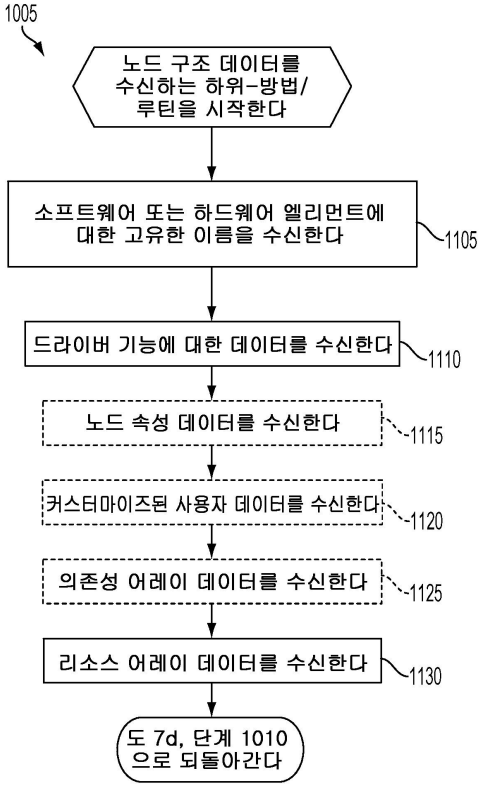
도면7d


1000A
시작
노드 구조 데이터를 수신한다
1005
노드 구조 데이터 내
의존성 데이터를 검토한다
1010
1015
노드 구조 데이터가
리프 노드를
정의하는가?
아니오
예
1020
노드 구조
데이터 내 모든 강한
의존성들이 존재?
아니오
예
1027
노드 구조 데이터를 임시
스토리지에 저장하고 콜-백을
생성한다
1025
노드 구조 데이터에 기초하여 노드를
인스턴스화/생성한다
1030
그 노드를 공표한다
도 7e, 단계
1035로 진행

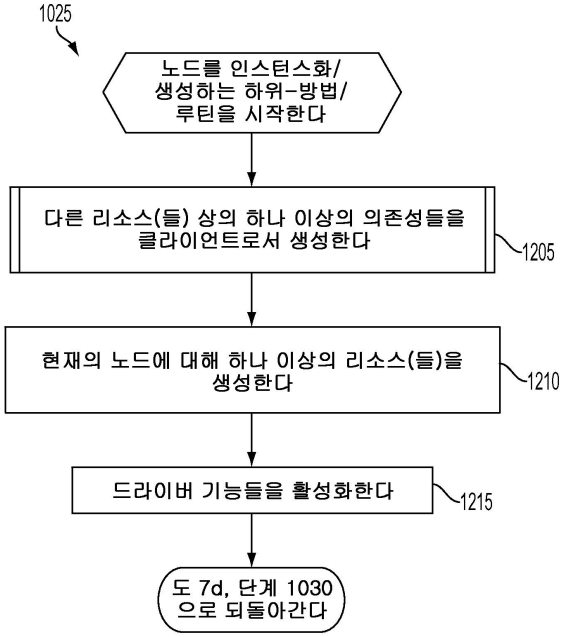
도면7e


1000B
도 7d,
단계 1030
으로 부터
현재 노드에 의존하는 노드들에게
현재 노드의 생성을 통지한다
1035
1040
다른 노드들 또는
소프트 의존성들이 지금 현재
노드 생성에 기초하여 인스턴스화를
위해 해제되었는가?
예
도 7d,
루틴 1025로
진행
아니오
기존 노드들 간에 통신(들)
(이벤트들, 쿼리들 등) 을 관리한다
1045
고유한 리소스 이름(들)을 이용하여 리소스(들)에
의해 취해진 액션(들)을 계속해서 로그한다
1050
아니오
요청 로그 ?
1055
예
하나 이상의 출력 디바이스(들) 상에 고유한
리소스 이름(들)에 의해 리소스(들)을 리스트하는
액션(들)의 로그를 제공한다
1060
도 7d, 루틴 1005로
되돌아간다

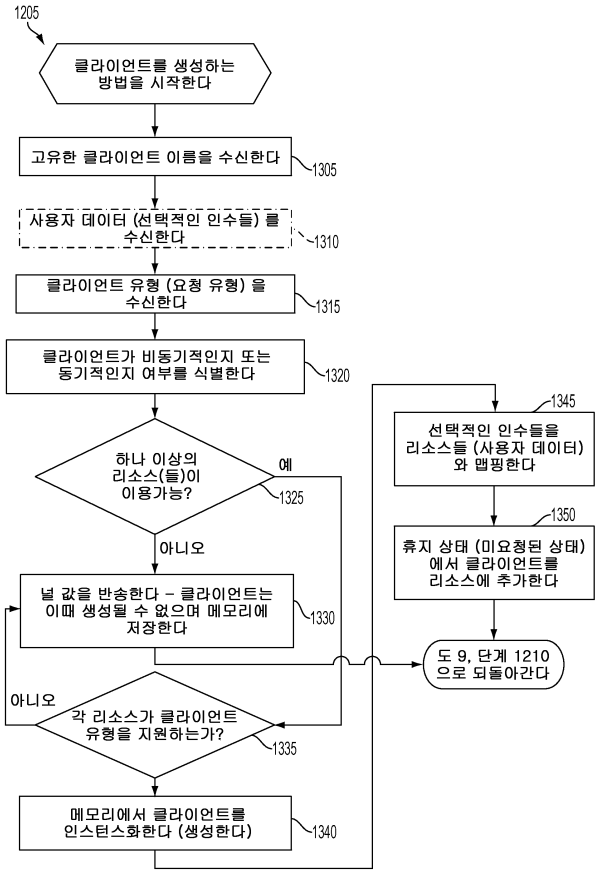
도면8


1005
노드 구조 데이터를 수신하는 하위-방법/루틴을 시작한다
소프트웨어 또는 하드웨어 엘리먼트에 대한 고유한 이름을 수신한다
1105
드라이버 기능에 대한 데이터를 수신한다
1110
노드 속성 데이터를 수신한다
1115
커스터마이즈된 사용자 데이터를 수신한다
1120
의존성 어레이 데이터를 수신한다
1125
리소스 어레이 데이터를 수신한다
1130
도 7d, 단계 1010으로 되돌아간다

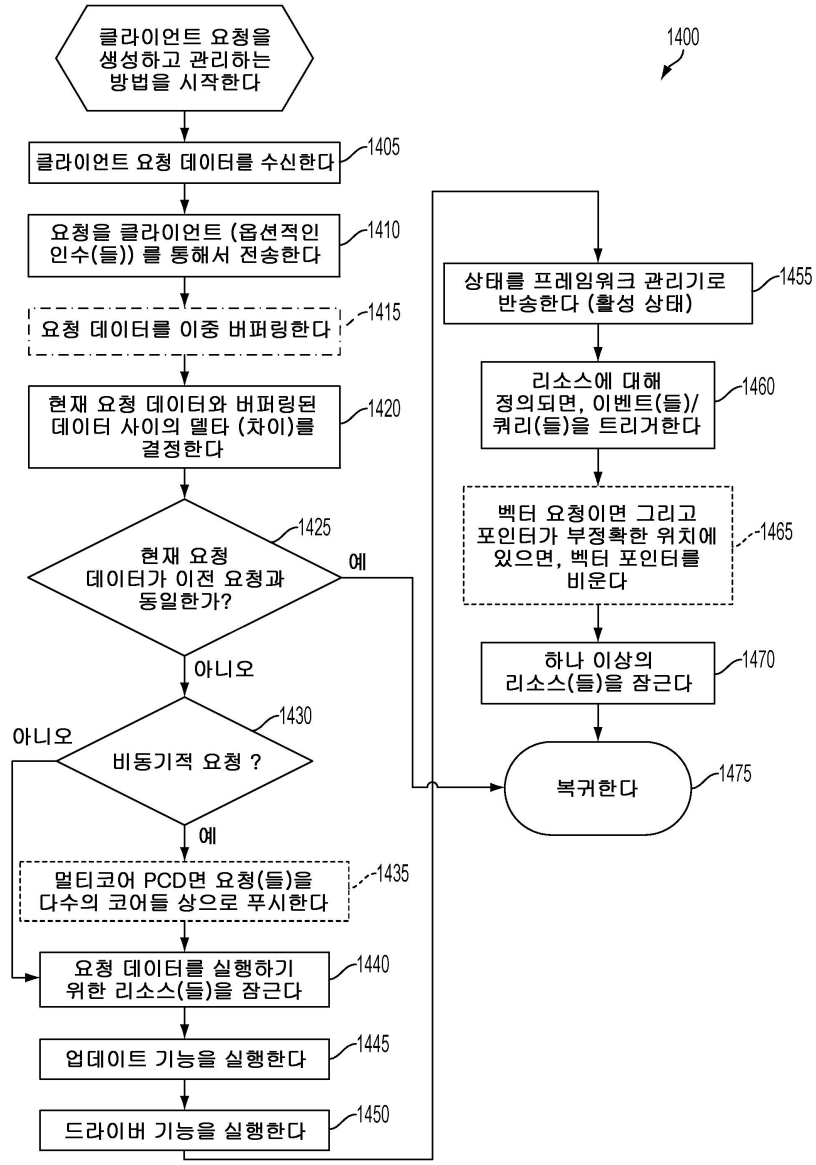
도면9


1025
노드를 인스턴스화/생성하는 하위-방법/루틴을 시작한다
다른 리소스(들) 상의 하나 이상의 의존성들을 클라이언트로서 생성한다
1205
현재의 노드에 대해 하나 이상의 리소스(들)을 생성한다
1210
드라이버 기능들을 활성화한다
1215
도 7d, 단계 1030으로 되돌아간다

도면10


1205
클라이언트를 생성하는 방법을 시작한다
고유한 클라이언트 이름을 수신한다
1305
사용자 데이터 (선택적인 인수들) 를 수신한다
1310
클라이언트 유형 (요청 유형) 을 수신한다
1315
클라이언트가 비동기적인지 또는 동기적인지 여부를 식별한다
1320
하나 이상의 리소스(들)이 이용가능?
1325
예
아니오
널 값을 반송한다 - 클라이언트는 이때 생성될 수 없으며 메모리에 저장한다
1330
아니오
각 리소스가 클라이언트 유형을 지원하는가?
1335
메모리에서 클라이언트를 인스턴스화한다 (생성한다)
1340
1345
선택적인 인수들을 리소스들 (사용자 데이터) 와 맵핑한다
1350
휴지 상태 (미요청된 상태) 에서 클라이언트를 리소스에 추가한다
도 9, 단계 1210 으로 되돌아간다

도면11


클라이언트 요청을 생성하고 관리하는 방법을 시작한다
1400
클라이언트 요청 데이터를 수신한다
1405
요청을 클라이언트 (옵션적인 인수(들)) 를 통해서 전송한다
1410
요청 데이터를 이중 버퍼링한다
1415
현재 요청 데이터와 버퍼링된 데이터 사이의 델타 (차이)를 결정한다
1420
현재 요청 데이터가 이전 요청과 동일한가?
1425
예
아니오
비동기적 요청 ?
1430
아니오
예
멀티코어 PCD면 요청(들)을 다수의 코어들 상으로 푸시한다
1435
요청 데이터를 실행하기 위한 리소스(들)을 잠근다
1440
업데이트 기능을 실행한다
1445
드라이버 기능을 실행한다
1450
상태를 프레임워크 관리기로 반송한다 (활성 상태)
1455
리소스에 대해 정의되면, 이벤트(들)/ 쿼리(들)을 트리거한다
1460
벡터 요청이면 그리고 포인터가 부정확한 위치에 있으면, 벡터 포인터를 비운다
1465
하나 이상의 리소스(들)을 잠근다
1470
복귀한다
1475