(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2022년11월07일
(11) 등록번호 10-2464337
(24) 등록일자 2022년11월02일

(51) 국제특허분류(Int. Cl.)
G06F 9/46 (2006.01) *H04L 47/762* (2022.01)
H04L 65/40 (2022.01)
(52) CPC특허분류
G06F 9/461 (2013.01)
H04L 47/762 (2022.05)
(21) 출원번호 10-2017-7009555
(22) 출원일자(국제) 2015년09월25일
심사청구일자 2020년09월25일
(85) 번역문제출일자 2017년04월07일
(65) 공개번호 10-2017-0059451
(43) 공개일자 2017년05월30일
(86) 국제출원번호 PCT/US2015/052469
(87) 국제공개번호 WO 2016/049582
국제공개일자 2016년03월31일
(30) 우선권주장
62/055,068 2014년09월25일 미국(US)
(56) 선행기술조사문헌
US20110213870 A1*
(뒷면에 계속)

(73) 특허권자
오라클 인터내셔날 코포레이션
미국, 캘리포니아 94065, 레드우드 쇼어스 엠에스 5오피7, 오라클 파크웨이 500
(72) 발명자
사후 산제브
미국 캘리포니아주 94065 레드우즈 쇼어 오라클 파크웨이 500
티아가라잔 시바쿠마르
인도 벵갈루루 560075 지반 비마 나가르 팀마 레디 콜로니 라 레지던스 플랫 207
(74) 대리인
박장원

전체 청구항 수 : 총 21 항 심사관 : 이후락

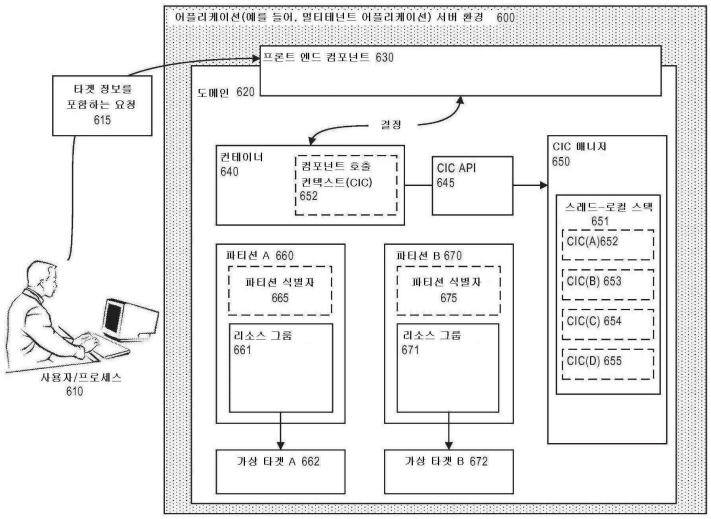
(54) 발명의 명칭 **멀티테넌트 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 위한 시스템 및 방법**

(57) 요 약

일 실시예에 따른, 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 위한 시스템 및 방법이 본 명세서에서 설명된다. 예시적인 방법은 하나 이상의 컴퓨터들에서, 어플리케이션 서버 환경 내에서 사용될 수 있는 복수의 배치가능한 리소스들, 하나 이상의 파티션들 및 컴포넌트 호출 컨텍스트 매니저를 함께 제공하는 것으로 시작될

(뒷면에 계속)

대 표 도



수 있으며, 상기 하나 이상의 컴퓨터들은 상기 하나 이상의 컴퓨터들 상에서 실행되는 상기 어플리케이션 서버 환경을 포함하며, 각각의 파티션은 도메인의 관리 및 런타임 서브디비전을 제공하며, 상기 컴포넌트 호출 컨텍스트 매니저는 스택을 포함한다. 상기 방법은 하나 이상의 컴포넌트 호출 컨텍스트들을 설정할 수 있다. 상기 파티션-인식 컨테이너는 현재 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에 등록시키는 것, 또는 상기 컴포넌트 호출 컨텍스트 매니저에서 현재 컴포넌트 호출 컨텍스트를 검색하는 것 중 하나를 수행할 수 있다. 상기 현재 컴포넌트 호출 컨텍스트는 현재 파티션과 관련될 수 있다. 상기 방법 및 시스템은 멀티테넌트뿐만 아니라 비-멀티테넌트 어플리케이션 서버 환경들에서 활용될 수 있다.

(52) CPC특허분류
H04L 67/10 (2022.05)

(56) 선행기술조사문헌
KR1020140043068 A*
KR1020080089672 A
KR1020060004909 A
KR1020050057024 A
KR1020040076583 A
KR1020020047218 A
*는 심사관에 의하여 인용된 문헌

명 세 서

청구범위

청구항 1

어플리케이션 서버 환경에서 파티션 식별자들(partition identifiers)의 결정을 위한 시스템에 있어서, 상기 시스템은:

상기 어플리케이션 서버 환경을 포함하는 하나 이상의 컴퓨터들과, 상기 어플리케이션 서버 환경은

상기 어플리케이션 서버 환경 내에서 사용될 수 있는 복수의 배치가능한 리소스들, 및

타겟 파티션을 포함하는 하나 이상의 파티션들을 포함하고, 각 파티션은 도메인의 관리 및 런타임 서브디비전을 제공하고 파티션 식별자와 연관되며, 상기 타겟 파티션은 상기 타겟 파티션과 사용하기 위한 상기 배치가능한 리소스들의 콜렉션에 대한 구성 데이터의 값들을 정의하고, 상기 타겟 파티션과 사용하기 위해 상기 배치가능한 리소스들의 콜렉션을 상기 정의된 값들과 바인딩하고;

요청의 타겟 리소스가 상기 타겟 파티션의 상기 배치가능한 리소스들의 콜렉션에 포함된다고 결정하고, 상기 타겟 파티션의 파티션 식별자와 연관된 컴포넌트 호출 컨텍스트를 설정하는 파티션-인식 컨테이너와;

컴포넌트 호출 컨텍스트 매니저를 포함하고, 상기 컴포넌트 호출 컨텍스트 매니저는 컴포넌트 호출 컨텍스트들을 유지하기 위한 스택을 포함하며;

상기 파티션-인식 컨테이너는 상기 요청을 표현하는 상기 컴포넌트 호출 컨텍스트를 상기 스택에 푸싱함으로써 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 등록하는 것 또는 상기 스택으로부터 현재 컴포넌트 호출 컨텍스트를 요청함으로써 상기 컴포넌트 호출 컨텍스트 매니저에서 현재 컴포넌트 호출 컨텍스트를 검색(look up)하는 것 중 하나를 수행하는, 시스템.

청구항 2

청구항 1에 있어서,

상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 등록하는 것은:

프론트 엔드 컴포넌트에서, 타겟 정보를 포함하는 상기 요청을 수신하는 것을 포함하며, 상기 타겟 정보는 상기 타겟 파티션을 표시하며;

상기 타겟 파티션의 파티션 식별자와 연관된 컴포넌트 호출 컨텍스트를 상기 스택에 푸싱하는 것은 상기 파티션-인식 컨테이너에 의해 상기 컴포넌트 호출 컨텍스트 매니저의 어플리케이션 프로그래밍 인터페이스에 대한 호출을 하는 것을 포함하는, 시스템.

청구항 3

청구항 2에 있어서,

상기 파티션-인식 컨테이너는 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에서 등록한 이후, 상기 요청의 완료시, 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에서 등록 취소(unregister)하도록 구성되며, 상기 등록 취소는:

상기 어플리케이션 프로그래밍 인터페이스를 통해, 상기 파티션-인식 컨테이너에 의해, 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에서 등록 취소하는 것을 포함하는, 시스템.

청구항 4

청구항 1에 있어서,

작업 매니저(work manager)를 더 포함하고,

상기 파티션-인식 컨테이너에 의해 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에 등록하는 것은 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에 등록하기 위해, 상기 파티션-인식 컨테이너에 의해 명령어들을 상기 작업 매니저로 전달(pass)하는, 시스템.

청구항 5

청구항 1에 있어서,

상기 현재 컴포넌트 호출 컨텍스트를 검색하는 것은:

상기 파티션-인식 컨테이너 내에서 상기 컨텍스트로서, 상기 현재 컴포넌트 호출 컨텍스트를 설정하는 것을 포함하는, 시스템.

청구항 6

청구항 1에 있어서,

상기 타겟 파티션은 글로벌 파티션이며, 상기 글로벌 파티션은 상기 도메인과 연관되는, 시스템.

청구항 7

청구항 1에 있어서,

상기 어플리케이션 서버 환경은 멀티-테넌트 어플리케이션 서버 환경(multi-tenant application server enviroment)을 포함하고, 상기 시스템은 테넌트에 의한 사용을 위해 상기 하나 이상의 파티션들과 상기 테넌트를 연관시키는, 시스템.

청구항 8

하나 이상의 컴퓨터들에서 실행되는 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 위한 방법에 있어서, 상기 방법은:

상기 하나 이상의 컴퓨터들에 의해, 어플리케이션 서버 환경에서,

상기 어플리케이션 서버 환경 내에서 사용될 수 있는 복수의 배치가능한 리소스들,

타겟 파티션을 포함하는 하나 이상의 파티션들, 각 파티션은 도메인의 관리 및 런타임 서브디비전을 제공하고 파티션 식별자와 연관되며, 상기 타겟 파티션은 상기 타겟 파티션과 사용하기 위한 상기 배치가능한 리소스들의 콜렉션에 대한 구성 데이터의 값들을 정의하고, 상기 타겟 파티션과 사용하기 위해 상기 배치가능한 리소스들의 콜렉션을 상기 정의된 값들과 바인딩하고, 및

컴포넌트 호출 컨텍스트 매니저를 제공하는 단계, 상기 컴포넌트 호출 컨텍스트 매니저는 컴포넌트 호출 컨텍스트들을 유지하기 위한 스택을 포함하며;

파티션-인식 컨테이너에서, 요청의 타겟 리소스가 상기 타겟 파티션의 배치가능한 리소스들의 콜렉션에 포함된다고 결정하는 단계;

상기 어플리케이션 서버 환경의 상기 파티션-인식 컨테이너에서, 상기 타겟 파티션의 파티션 식별자와 연관된 컴포넌트 호출 컨텍스트를 설정하는 단계; 및

상기 어플리케이션 서버 환경의 상기 파티션-인식 컨테이너에 의해,

상기 요청을 표현하는 상기 컴포넌트 호출 컨텍스트를 상기 스택에 푸싱함으로써 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 등록하는 것, 또는

상기 스택으로부터 현재 컴포넌트 호출 컨텍스트를 요청함으로써 상기 컴포넌트 호출 컨텍스트 매니저에서 현재 컴포넌트 호출 컨텍스트를 검색하는 것 중 하나를 수행하는 단계를 포함하는, 방법.

청구항 9

청구항 8에 있어서,

상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 등록하는 것은:

프론트 엔드 컴포넌트에서, 타겟 정보를 포함하는 요청을 수신하는 것을 포함하며, 상기 타겟 정보는 상기 타겟 파티션을 표시하며; 그리고

상기 타겟 파티션의 파티션 식별자와 연관된 컴포넌트 호출 컨텍스트를 상기 스택에 푸싱하는 것은 상기 파티션-인식 컨테이너에 의해 상기 컴포넌트 호출 컨텍스트 매니저의 어플리케이션 프로그래밍 인터페이스에 대한 호출을 하는 것을 포함하는, 방법.

청구항 10

청구항 9에 있어서,

상기 파티션-인식 컨테이너는 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에서 등록한 이후, 상기 요청의 완료시, 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에서 등록 취소(unregister)하도록 구성되며, 상기 등록 취소는:

상기 어플리케이션 프로그래밍 인터페이스를 통해, 상기 파티션-인식 컨테이너에 의해, 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에서 등록 취소하는 것을 포함하는, 방법.

청구항 11

청구항 8에 있어서,

상기 하나 이상의 컴퓨터들에 의해, 상기 어플리케이션 서버 환경에서, 작업 매니저를 더 제공하는 단계를 더 포함하고, 그리고

상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에서 등록하는 것은 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에 등록하기 위해, 상기 파티션-인식 컨테이너에 의해 명령어들을 상기 작업 매니저로 전달(pass)하는 것을 포함하는, 방법.

청구항 12

청구항 8에 있어서,

상기 현재 컴포넌트 호출 컨텍스트를 검색하는 것은:

상기 파티션-인식 컨테이너 내에서 상기 컨텍스트로서, 상기 현재 컴포넌트 호출 컨텍스트를 설정하는 것을 포함하는, 방법.

청구항 13

청구항 8에 있어서,

상기 타겟 파티션은 글로벌 파티션이며, 상기 글로벌 파티션은 상기 도메인과 연관되는, 방법.

청구항 14

청구항 8에 있어서,

상기 어플리케이션 서버 환경은 멀티-테넌트 어플리케이션 서버 환경을 포함하고, 상기 방법은:

테넌트에 의한 사용을 위해 상기 하나 이상의 파티션들과 상기 테넌트를 연관시키는 단계를 더 포함하는, 방법.

청구항 15

비일시적 컴퓨터 판독가능한 저장 매체로서, 상기 저장 매체는 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 위한 상기 저장 매체 상에 저장된 명령어들을 포함하며, 상기 명령어들은 하나 이상의 컴퓨터들에 의해 판독 및 실행시, 상기 하나 이상의 컴퓨터들로 하여금 단계들을 수행하도록 하며, 상기 단계들은:

어플리케이션 서버 환경에서,

상기 어플리케이션 서버 환경 내에서 사용될 수 있는 복수의 배치가능한 리소스들,

타겟 파티션을 포함하는 하나 이상의 파티션들, 각 파티션은 도메인의 관리 및 런타임 서브디비전을 제공하고 파티션 식별자와 연관되며, 상기 타겟 파티션은 상기 타겟 파티션과 사용하기 위한 상기 배치가능한 리소스들의 콜렉션에 대한 구성 데이터의 값들을 정의하고, 상기 타겟 파티션과 사용하기 위해 상기 배치가능한 리소스들의 콜렉션을 상기 정의된 값들과 바인딩하고, 및

컴포넌트 호출 컨텍스트 매니저를 제공하는 단계, 상기 컴포넌트 호출 컨텍스트 매니저는 컴포넌트 호출 컨텍스트들을 유지하기 위한 스택을 포함하며;

파티션-인식 컨테이너에서, 요청의 타겟 리소스가 상기 타겟 파티션의 배치가능한 리소스들의 콜렉션에 포함된다고 결정하는 단계;

상기 파티션-인식 컨테이너에서, 상기 타겟 파티션의 파티션 식별자와 연관된 컴포넌트 호출 컨텍스트를 설정하는 단계; 및

상기 어플리케이션 서버 환경의 상기 파티션-인식 컨테이너에 의해,

상기 요청을 표현하는 상기 컴포넌트 호출 컨텍스트를 상기 스택에 푸싱함으로써 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 등록하는 것, 또는

상기 스택으로부터 현재 컴포넌트 호출 컨텍스트를 요청함으로써 상기 컴포넌트 호출 컨텍스트 매니저에서 현재 컴포넌트 호출 컨텍스트를 검색하는 것 중 하나를 수행하는 단계를 포함하는, 비일시적 컴퓨터 판독가능한 저장 매체.

청구항 16

청구항 15에 있어서,

상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 등록하는 것은:

프론트 엔드 컴포넌트에서, 타겟 정보를 포함하는 요청을 수신하는 것을 포함하며, 상기 타겟 정보는 상기 타겟 파티션을 표시하며; 그리고

상기 타겟 파티션의 파티션 식별자와 연관된 컴포넌트 호출 컨텍스트를 상기 스택에 푸싱하는 것은 상기 파티션-인식 컨테이너에 의해 상기 컴포넌트 호출 컨텍스트 매니저의 어플리케이션 프로그래밍 인터페이스에 대한 호출을 하는 것을 포함하는, 비일시적 컴퓨터 판독가능한 저장 매체.

청구항 17

청구항 16에 있어서,

상기 파티션-인식 컨테이너는 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에서 등록한 이후, 상기 요청의 완료시, 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에서 등록 취소(unregister)하도록 구성되며, 상기 등록 취소는:

상기 어플리케이션 프로그래밍 인터페이스를 통해, 상기 파티션-인식 컨테이너에 의해, 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에서 등록 취소하는 것을 포함하는, 비일시적 컴퓨터 판독가능한 저장 매체.

청구항 18

청구항 15에 있어서,

상기 하나 이상의 컴퓨터들에 의해, 상기 어플리케이션 서버 환경에서, 작업 매니저를 더 제공하는 단계를 더 포함하고, 그리고

상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에서 등록하는 것은 상기 타겟 파티션의 파티션 식별자와 연관된 상기 컴포넌트 호출 컨텍스트를 상기 컴포넌

트 호출 컨텍스트 매니저에 등록하기 위해, 상기 파티션-인식 컨테이너에 의해 명령어들을 상기 작업 매니저로 전달(pass)하는 것을 포함하는, 비일시적 컴퓨터 판독가능한 저장 매체.

청구항 19

청구항 15에 있어서,

상기 현재 컴포넌트 호출 컨텍스트를 검색하는 것은:

상기 파티션-인식 컨테이너 내에서 상기 컨텍스트로서, 상기 현재 컴포넌트 호출 컨텍스트를 설정하는 것을 포함하는, 비일시적 컴퓨터 판독가능한 저장 매체.

청구항 20

청구항 15에 있어서,

상기 타겟 파티션은 글로벌 파티션이며, 상기 글로벌 파티션은 상기 도메인과 연관되는, 비일시적 컴퓨터 판독가능한 저장 매체.

청구항 21

비일시적 컴퓨터 판독가능 저장 매체에 저장된 컴퓨터 프로그램으로서, 상기 컴퓨터 프로그램은 하나 이상의 컴퓨터 시스템들 상에서의 실행을 위한 프로그램 명령어들을 포함하며, 상기 프로그램 명령어들은 실행시, 상기 하나 이상의 컴퓨터 시스템들로 하여금 청구항 8 내지 14 중 어느 한 항의 방법을 수행하도록 하는, 컴퓨터 프로그램.

청구항 22

삭제

발명의 설명

기 술 분 야

[0001] [저작권 공지]

[0002] 이 특허 문서의 개시 부분에는 저작권 보호 타겟인 자료가 포함되어 있다. 저작권 소유자는 특허청 및 상표국의 특허 파일 또는 기록에 나타나있는 특허 문서 또는 특허 개시 내용에 대한 사본 재생에 대해서는 이의를 제기하지 않지만, 이외에 다른 행위에 대해서는 모든 저작권 권리를 보유한다.

[0003] [기술분야]

[0004] 본 발명의 실시예들은 일반적으로 어플리케이션 서버들 및 클라우드 환경들에 관한 것으로, 특히 멀티테넌트 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 위한 시스템 및 방법에 관한 것이다.

배 경 기 술

[0005] 어플리케이션 서버들은 일반적으로 소프트웨어 어플리케이션들이 배치되고 실행될 수 있는 관리 환경을 제공한다. 클라우드-기반 환경들은 어플리케이션들로 하여금 클라우드에 의해 제공된 분배된 리소스들 내에서 실행되고, 상기 리소스들의 장점을 취할 수 있게 한다. 이러한 환경들은 많은 수의 사용자들 또는 테넌트들을 지원할 수 있고, 그 중 일부는 해당 사용자 또는 테넌트에게 특정한 특정 요구 사항들을 가질 수 있다.

발명의 내용

[0006] 일 실시예에 따른, 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 위한 시스템 및 방법이 본 명세서에서 설명된다. 예시적인 방법은 하나 이상의 컴퓨터들에서, 상기 하나 이상의 컴퓨터들 상에서 실행되는 어플리케이션 서버 환경을 포함하는 것을 제공하는 것으로 시작될 수 있으며, 상기 어플리케이션 서버 환경은 상기 어플리케이션 서버 환경 내에서 사용될 수 있는 복수의 배치가능한 리소스들, 하나 이상의 파티션들, 각각의 파티션은 도메인의 관리 및 런타임 서브디비전을 제공하며, 그리고 컴포넌트 호출 컨텍스트 매니저를 포함하고, 상기 컴포넌트 호출 컨텍스트 매니저는 스택을 포함한다. 상기 방법은 파티션-인식 컨테이너에서 하나 이상의 컴

포넌트 호출 컨텍스트들을 설정할 수 있다. 상기 파티션-인식 컨테이너는 현재 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에 등록시키는 것, 또는 상기 컴포넌트 호출 컨텍스트 매니저에서 현재 컴포넌트 호출 컨텍스트를 검색하는 것 중 하나를 수행할 수 있다. 상기 현재 컴포넌트 호출 컨텍스트는 현재 파티션과 관련될 수 있다.

[0007] 일 실시예에 따른, 본 명세서에서 설명된 멀티-테넌트 운영 환경의 컨텍스트 내 시스템들 및 방법들은 또한, 서블릿이 EJB 컨테이너를 호출하는 예시에서 아래에 설명된 바와 같이, 비-멀티-테넌트 운영 환경 내에서 사용될 수 있다

[0008] 일 실시예에 따라, 파티션(예컨데, WebLogic 환경의 도메인 내의 파티션)은 도메인 내의 테넌트로서 포지션(position)될 수 있다. 본 시스템 및 방법은 일반적인 방식으로 테넌트 요청들의 빠른 식별(early identification)을 허용할 수 있다(즉, 시스템 및 방법이 멀티-테넌트 어플리케이션 서버 환경뿐만 아니라 비-멀티-테넌트 어플리케이션 서버 환경 내에서도 테넌트 요청들을 식별할 수 있음을 일반적으로 의미한다).

도면의 간단한 설명

[0009] 도 1은 일 실시예에 따른, 어플리케이션 서버, 클라우드 또는 다른 환경에서 멀티-테넌시(multi-tenancy)를 지원하는 시스템을 도시한다.

도 2는 일 실시예에 따른, 어플리케이션 서버, 클라우드 또는 다른 환경에서 멀티-테넌시를 지원하는 시스템을 추가로 도시한다.

도 3은 일 실시예에 따른, 어플리케이션 서버, 클라우드 또는 다른 환경에서 멀티-테넌시를 지원하는 시스템을 추가로 도시한다.

도 4는 일 실시예에 따른, 예시적인 멀티-테넌트 환경과 함께 사용하기 위한 도메인 구성을 도시한다.

도 5는 일 실시예에 따른, 예시적인 멀티-테넌트 환경을 추가로 도시한다.

도 6은 일 실시예에 따른, 멀티테넌트 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 도시한다.

도 7은 일 실시예에 따른, 멀티테넌트 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 도시한다.

도 8은 일 실시예에 따른, 멀티테넌트 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 도시한다.

도 9는 일 실시예에 따른, 멀티테넌트 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 위한 방법을 흐름도를 통해 도시한다.

발명을 실시하기 위한 구체적인 내용

[0010] 일 실시예에 따른, 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 위한 시스템 및 방법이 본 명세서에서 설명된다. 예시적인 방법은 하나 이상의 컴퓨터들에서, 상기 하나 이상의 컴퓨터들 상에서 실행되는 어플리케이션 서버 환경을 포함하는 것을 제공하는 것으로 시작될 수 있으며, 상기 어플리케이션 서버 환경은 상기 어플리케이션 서버 환경 내에서 사용될 수 있는 복수의 배치가능한 리소스들, 하나 이상의 파티션들, 각각의 파티션은 도메인의 관리 및 런타임 서브디비전을 제공하며, 그리고 컴포넌트 호출 컨텍스트 매니저를 포함하고, 상기 컴포넌트 호출 컨텍스트 매니저는 스택을 포함한다. 상기 방법은 파티션-인식 컨테이너에서 하나 이상의 컴포넌트 호출 컨텍스트들을 설정할 수 있다. 상기 파티션-인식 컨테이너는 현재 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에 등록시키는 것, 또는 상기 컴포넌트 호출 컨텍스트 매니저에서 현재 컴포넌트 호출 컨텍스트를 검색하는 것 중 하나를 수행할 수 있다. 상기 현재 컴포넌트 호출 컨텍스트는 현재 파티션과 관련될 수 있다.

[0011] 일 실시예에 따른, 본 명세서에서 설명된 멀티-테넌트 운영 환경의 컨텍스트 내 시스템들 및 방법들은 또한, 서블릿이 EJB 컨테이너를 호출하는 예시에서 아래에 설명된 바와 같이, 비-멀티-테넌트 운영 환경 내에서 사용될 수 있다

[0012] 일 실시예에 따라, 파티션(예컨데, WebLogic 환경의 도메인 내의 파티션)은 도메인 내의 테넌트로서 포지션될 수 있다. 본 시스템 및 방법은 일반적인 방식으로 테넌트 요청들의 빠른 식별을 허용할 수 있다(즉, 시스템 및 방법이 멀티-테넌트 어플리케이션 서버 환경뿐만 아니라 비-멀티-테넌트 어플리케이션 서버 환경 내에서도 테넌트 요청들을 식별할 수 있음을 일반적으로 의미한다).

[0013] **어플리케이션 서버(예를 들어, 멀티-테넌트, MT) 환경**

[0014] 도 1은 일 실시예에 따른, 어플리케이션 서버, 클라우드 또는 다른 환경에서 멀티-테넌시를 지원하는 시스템을 도시한다.

[0015] 도 1에 도시된 바와 같이, 일 실시예에 따르면, 어플리케이션 서버(예를 들어, 멀티-테넌트, MT) 환경(100) 또는 소프트웨어 어플리케이션들의 배치 및 실행을 가능하게 하는 다른 컴퓨팅 환경은, 런타임(runtime)시 어플리케이션 서버 도메인을 정의하는데 사용되는 도메인(102) 구성에 따라 포함하고 동작하도록 구성될 수 있다.

[0016] 일 실시예에 따르면, 어플리케이션 서버는 런타임시 사용하기 위해 정의된 하나 이상의 파티션들(104)을 포함할 수 있다. 각각의 파티션은 글로벌 고유 파티션 식별자(globally unique partition identifier)(ID) 및 파티션 구성과 관련될 수 있고, 리소스 그룹 템플릿(resource group template)(126) 및/또는 파티션-특정 어플리케이션들 또는 리소스들(128)에 대한 참조와 함께 하나 이상의 리소스 그룹들(124)을 더 포함할 수 있다. 또한, 도메인-레벨 리소스 그룹들, 어플리케이션들 및/또는 리소스들(140)은, 옵션적으로(optionally) 리소스 그룹 템플릿에 대한 참조와 함께, 상기 도메인 레벨에서 정의될 수 있다.

[0017] 각각의 리소스 그룹 템플릿(160)은 하나 이상의 어플리케이션들(어플리케이션 A(162) 및 어플리케이션 B(164)), 하나 이상의 리소스들(리소스 A(166) 및 리소스 B(168)) 및/또는 다른 배치가능한 어플리케이션들 또는 리소스들(170)을 정의할 수 있고, 리소스 그룹에 의해 참조될 수 있다. 예를 들면, 도 1에 도시된 바와 같이, 파티션(104) 내 리소스 그룹(124)은 리소스 그룹 템플릿(160)을 참조(190)할 수 있다.

[0018] 일반적으로, 시스템 관리자(system administrator)는 파티션들, 도메인-레벨 리소스 그룹들 및 리소스 그룹 템플릿들 및 보안 영역들(security realms)을 정의할 수 있다; 예를 들면, 파티션 관리자는 파티션-레벨 리소스 그룹들을 생성하거나, 어플리케이션들을 파티션에 배치하거나, 또는 파티션에 대한 특정 영역들을 참조함으로써, 그들 자신의 파티션의 양태들을 정의할 수 있다.

[0019] 도 2는 일 실시예에 따른, 어플리케이션 서버, 클라우드 또는 다른 환경에서 멀티-테넌시를 지원하는 시스템을 추가로 도시한다.

[0020] 도 2에 도시된 바와 같이, 일 실시예에 따르면, 파티션(202)은 예를 들면, 리소스 그룹 템플릿(210)에 대한 참조(206)를 포함하는 리소스 그룹(205), 가상 타겟(예를 들어, 가상 호스트) 정보(virtual target information)(207) 및 플러거블 데이터베이스(PDB: pluggable database), 또는 다른 데이터 소스, 정보(208)를 포함할 수 있다. 리소스 그룹 템플릿(예를 들어, 210)은, 예를 들면, 리소스들(예컨데, 자바 메시지 서버(JMS) 서버(213), 축적 전송(SAF) 에이전트(215), 메일 세션 컴포넌트(216) 또는 자바 데이터베이스 연결(JDBC) 리소스(217))과 함께 복수의 어플리케이션들(어플리케이션 A(162) 및 어플리케이션 B(164))을 정의할 수 있다.

[0021] 도 2에 도시된 리소스 그룹 템플릿은 예시로서 제공된다. 다른 실시예들에 따라 서로 다른 타입의 리소스 그룹 템플릿들과 엘리먼트들이 제공될 수 있다.

[0022] 일 실시예에 따르면, 파티션(예를 들어, 220) 내의 리소스 그룹이 특정 리소스 그룹 템플릿(예를 들어, 210)을 참조(220)할 때, 특정 파티션과 관련된 정보는 파티션-특정 정보(230)(예를 들면, 파티션-특정 PDB 정보)를 표시하기 위해, 참조된 리소스 그룹 템플릿과 결합하여 사용될 수 있다. 그 후, 파티션-특정 정보는 파티션에 의한 사용을 위한 리소스들(예를 들면, PDB 리소스)를 구성하기 위해, 어플리케이션 서버에 의해 사용될 수 있다. 예를 들면, 파티션(202)과 관련된 파티션-특정 PDB 정보는, 어플리케이션 서버에 의해, 적절한 파티션(228) 또는 해당 파티tus에 의한 사용을 위한 다른 데이터 소스를 갖는 컨테이너 데이터베이스(CDB: container database)(236)를 구성(232)하는데 사용될 수 있다.

[0023] 유사하게, 일 실시예에 따르면, 특정 파티션과 관련된 가상 타겟 정보는, 해당 파티션(예를 들어, baylandurgentcare.com)에 의한 사용을 위해, 파티션-특정 가상 타겟(240)을 정의(239)하는데 사용될 수 있고, 이후 URL(http://baylandurgentcare.com)을 통해 액세스 가능하게 만들어질 수 있다.

[0024] 도 3은 일 실시예에 따른, 어플리케이션 서버, 클라우드 또는 다른 환경에서 멀티-테넌시를 지원하는 시스템을 추가로 도시한다.

[0025] 일 실시예에 따르면, config.xml 구성 파일과 같은 시스템 구성은 파티션을 정의하는데 사용되며, 해당 파티션과 관련된 리소스 그룹들에 대한 엘리먼트들 및/또는 다른 파티션 속성들(properties)을 포함한다. 값들(values)은 속성 이름/값 쌍들을 사용하여 파티션마다 특정될 수 있다.

[0026] 일 실시예에 따르면, 복수의 파티션들은 관리 서버/클러스터(242) 또는 CDB(243)로의 액세스를 제공할 수 있고 웹 티어(web tier)(244)를 통해 액세스 가능한 유사한 환경에서 실행될 수 있다. 예를 들면, 이는 도메인 또는 파티션으로 하여금(CDB의) 하나 이상의 PDB들과 관련되도록 한다.

[0027] 일 실시예에 따르면, 복수의 파티션들 각각은, 상기 예시적인 파티션 A(250) 및 파티션 B(260)에서, 해당 파티션과 관련된 복수의 리소스들을 포함하도록 구성될 수 있다. 예를 들면, 파티션 A는 PDB A(259)와 관련된 데이터소스 A(257)과 함께 어플리케이션 A1(252), 어플리케이션 A2(254) 및 JMS A(256)를 포함하는 리소스 그룹(251)을 포함하도록 구성될 수 있고, 상기 파티션은 가상 타겟 A(258)를 통해 액세스 가능하다. 유사하게, 파티션 B(260)는 PDB B(269)와 관련된 데이터소스 B(267)과 함께 어플리케이션 B1(262), 어플리케이션 B2(264) 및 JMS B(266)를 포함하는 리소스 그룹(261)을 포함하도록 구성될 수 있고, 상기 파티션은 가상 타겟 B(268)를 통해 액세스 가능하다.

[0028] 상기 예시들 중 몇몇은 CDB 및 PDB들의 사용을 도시하지만, 다른 실시예들에 따라 다른 타입의 멀티-테넌트 또는 비-멀티-테넌트 데이터베이스들이 지원될 수 있고, 예를 들어 스키마들(schemas)의 사용 또는 서로 다른 데이터베이스들의 사용을 통해, 특정 구성이 각각의 파티션에 대해 제공될 수 있다.

[0029] **리소스들**

[0030] 일 실시예에 따르면, 리소스는 시스템 리소스, 어플리케이션 또는 환경의 도메인에 배치될 수 있는 다른 리소스 또는 객체(object)일 수 있다. 예를 들어, 일 실시예에 따르면, 리소스는 어플리케이션, JMS, JDBC, JavaMail, WLDF, 데이터 소스 또는 서버, 클러스터 또는 다른 어플리케이션 서버 타겟에 배치될 수 있는 다른 타입의 객체일 수 있다.

[0031] **파티션들**

[0032] 일 실시예에 따르면, 파티션은 파티션 식별자(ID) 및 구성과 관련될 수 있는 런타임 및 관리상의 서브디비전(subdivision) 또는 도메인의 슬라이스(slice)이며, 리소스 그룹들 및 리소스 그룹 템플릿들의 사용을 통해 어플리케이션들 및/또는 도메인-전체(domain-wide) 리소스들에 대한 참조를 포함할 수 있다. 또한, 파티션은 테넌트 또는 플랫폼 테넌트와 관련될 수 있다.

[0033] 일 실시예에 따르면, 파티션은 어플리케이션을 포함할 수 있고, 리소스 그룹 템플릿들을 통해 도메인 전체 어플리케이션들을 참조할 수 있고, 그 자신의 구성을 가질 수 있다.

[0034] 일 실시예에 따르면, 파티션 내의 리소스 그룹은 옵션적으로 리소스 그룹 템플릿을 참조할 수 있다. 파티션은 다수의 리소스 그룹들을 가질 수 있으며, 각각의 리소스 그룹은 리소스 그룹 템플릿을 참조할 수 있다. 각각의 파티션은 파티션의 리소스 그룹들이 참조하는 리소스 그룹 템플릿들에 특정되지 않은 구성 데이터에 대한 속성들을 정의할 수 있다. 이는 해당 파티션에서 사용하기 위한 값들을 특정하기 위해, 상기 파티션으로 하여금 리소스 그룹 템플릿에 정의된 배치 가능한 리소스들의 바인딩(binding)으로서 역할하도록 할 수 있다. 일부 경우들에서, 파티션은 리소스 그룹 템플릿에 의해 특정된 구성 정보를 오버라이드(override)할 수 있다.

[0035] 일 실시예에 따르면, 예를 들면 config.xml 구성 파일에 의해 정의된 파티션 구성은 복수의 컴포넌트(예를 들면: 파티션을 정의하는 속성들과 이들에 속하는 엘리먼트들을 포함하는 "파티션"; 파티션에 배치된 어플리케이션들과 리소스들을 포함하는 "리소스-그룹"; 해당 템플릿에 의해 정의된 어플리케이션들과 리소스들을 포함하는 "리소스-그룹-템플릿"; 데이터베이스-특정 서비스 이름, 사용자 이름 및 패스워드를 포함하는 "JDBC-시스템-리소스-오버라이드"; 및 리소스 그룹 템플릿들에서 매크로 대체에 사용되기 위한 속성 키 값들을 포함하는 "파티션-속성들")를 포함할 수 있다.

[0036] 시작할 때에, 시스템은 리소스 그룹 템플릿으로부터 각각의 리소스에 대한 파티션-특정 구성 엘리먼트들을 생성하기 위해, 특정 파일에 의해 제공된 정보를 사용할 수 있다.

[0037] **리소스 그룹들**

[0038] 일 실시예에 따르면, 리소스 그룹은 도메인 또는 파티션 레벨에서 정의될 수있는 배치가능한 리소스들의 지정된, 완전히 적격인(fully-qualified) 콜렉션이며, 리소스 그룹 템플릿을 참조할 수 있다. 리소스 그룹 내 리소스들은 관리자가 이러한 리소스들을 시작하거나 상기 리소스들에 연결하기 위해 필요한 모든 정보(예들 들면, 데이터 소스에 연결하거나, 어플리케이션에 대한 정보를 타켓팅하는 크리덴셜들(credentials))를 제공했다는 점에서 완전히 적격인 것으로 고려된다.

[0039] 파티션 레벨에서, 관리자는 임의의 보안 제한들에 따라 파티션에서 0 이상의 리소스 그룹들을 구성할 수 있다. 예를 들면, SaaS 유스 케이스(use case)에서, 다양한 파티션-레벨 리소스 그룹들은 도메인-레벨 리소스 그룹 템플릿들을 참조할 수 있다; 반면에 PaaS 유스 케이스에서는, 리소스 그룹 템플릿들을 참조하지 않고 대신에 해당 파티션에서만 이용가능 하도록 만들어진 어플리케이션들과 이들의 관련 리소스들을 나타내는 파티션-레벨 리소스 그룹들이 생성될 수 있다.

[0040] **리소스 그룹 템플릿들**

[0041] 일 실시예에 따르면, 도메인은 옵션적으로 리소스 그룹 템플릿을 포함할 수 있다. 리소스 그룹 템플릿은 리소스 그룹에서 참조될 수 있는 도메인 레벨에서 정의된 배포가능한 리소스들의 콜렉션이며, 이들의 리소스들을 액티베이트(activate)하는데 요구되는 정보 중 일부는 템플릿 자체의 일부로서 저장되지 않을 수 있어서 이는 파티션 레벨 구성의 사양(specification)을 지원한다. 도메인은 임의의 수의 리소스 그룹 템플릿들을 포함할 수 있고, 각각의 리소스 템플릿은 예를 들면, 하나 이상의 관련된 Java 어플리케이션들과 이러한 어플리케이션들이 의존하는 리소스를 포함할 수 있다.

[0042] 일 실시예에 따르면, 특정 리소스 그룹 템플릿은 하나 이상의 리소스 그룹들에 의해 참조될 수 있다. 일반적으로, 임의의 주어진 파티션 내에서, 리소스 그룹 템플릿은 한 번에 하나의 리소스 그룹에 의해 참조될 수 있다. 즉, 동일한 파티션 내의 다수의 리소스 그룹에 의해 동시에 참조될 수는 없다; 그러나, 리소스 그룹 템플릿은 서로 다른 파티션의 다른 리소스 그룹에 의해 동시에 참조될 수 있다. 리소스 그룹(예를 들어, 도메인 또는 파티션)을 포함하는 객체는 리소스 그룹 템플릿 내 임의의 토큰들(tokens) 값을 설정하기 위해, 속성 이름/값 할당들을 사용할 수 있다. 시스템이 참조 리소스 그룹을 사용하여 리소스 그룹 템플릿을 액티베이트하면, 이러한 토큰들을 상기 리소스 그룹의 포함 객체에 설정된 값들로 바꿀 수 있다. 일부 경우들에서, 시스템은 또한 각각의 파티션/템플릿 조합에 대한 런타임 구성을 생성하기 위해, 정적으로-구성된 리소스 그룹 템플릿들과 파티션들을 사용할 수 있다.

[0043] **테넌트들**

[0044] 일 실시예에 따르면, 테넌트는 파티션 ID와 관련될 수 있다. 예를 들면, 테넌트들은 별개의 사용자 조직들(예컨데, 서로 다른 외부 회사들 또는 특정 기업 내의 서로 다른 부서들(예를 들어, 인사부 및 재무 부서들))과 관련될 수 있다.

[0045] 일 실시예에 따르면, 시스템은 서로 다른 테넌트들(즉, 파티션들)의 관리 및 런타임을 서로 분리한다. 예를 들면, 허가된 사용자(예를 들어, 파티션 사용자 조직들의 허가된 사용자)는 관련된 파티션들의 어플리케이션들의 일부 행동들, 및 이들이 액세스하는 리소스를 구성할 수 있다. 시스템은 테넌트들 사이의 분리를 보장할 수 있고; 런타임시, 특정 테넌트를 대신하여 작업하는 어플리케이션들이 다른 테넌트들과 관련된 리소스들이 아닌 해당 테넌트와 관련된 리소스들만을 참조하는 것을 보장할 수 있다.

[0046] **예시적인 도메인 구성과 멀티-테넌트 환경**

[0047] 일 실시예에 따르면, 어플리케이션들은 도메인 레벨에서 리소스 그룹 템플릿으로, 또는 파티션으로 범위가 지정되거나 도메인으로 범위가 지정된 리소스 그룹으로 배치될 수 있다. 어플리케이션 구성은 응용 프로그램별로 또는 파티션별로 특정된 배치 계획들(deployment plans)을 사용하여 오버라이드될 수 있다. 또한, 배치 계획들은 리소스 그룹의 일부로서 지정될 수 있다.

[0048] 도 4는 일 실시예에 따른, 예시적인 멀티-테넌트 환경과 함께 사용하기 위한 도메인 구성을 도시한다.

[0049] 일 실시예에 따르면, 시스템이 파티션을 시작할 때, 상기 시스템은 제공된 구성에 따라, 각각의 데이터베이스 인스턴스들(instances)에 대한 각각의 파티션에 대해 하나의 파티션을 포함하는 가상 타켓들(예를 들어, 가상 호스트들) 및 연결 풀들(pools)을 생성한다.

[0050] 일반적으로, 각각의 리소스 그룹 템플릿은 하나 이상의 관련 어플리케이션들과 이러한 어플리케이션들이 의존하는 리소스를 포함할 수 있다. 각각의 파티션은 리소스 그룹 템플릿들의 배치가능한 리소스들의 바인딩을 상기 파티션과 관련된 특정 값들에 제공함으로써, 상기 파티션이 참조하는 상기 리소스 그룹 템플릿들에서 특정되지 않은 구성 데이터를 제공할 수 있다; 일부 경우들에서, 리소스 그룹 템플릿에 의해 특정된 특정한 구성 정보를 오버라이드하는 것을 포함한다. 이는 시스템으로 하여금 각각 파티션이 정의한 속성 값들을 사용하여, 각각의 파티션에 대해 리소스 그룹 템플릿에 의해 나타나는 어플리케이션을 다르게 액티베이트 할 수 있게 한다.

[0051] 일부 경우들에서, 파티션은 리소스 그룹 템플릿들을 참조하지 않거나 자신의 파티션-범위의 배치가능한 리소스

들을 직접 정의하는 리소스 그룹들을 포함할 수 있다. 파티션 내에 정의된 어플리케이션들과 데이터 소스들은 일반적으로 해당 파티션에서만 이용가능하다. 파티션:<partitionName>/<resource JNDI name> 또는 도메인:<resource JNDI name>을 사용하여 파티션들을 통해 리소스들이 액세스될 수 있도록, 리소스들이 배치될 수 있다.

[0052] 예를 들면, MedRec 어플리케이션은 복수의 자바 어플리케이션들, 데이터 소스, JMS 서버 및 메일 세션을 포함할 수 있다. 다수의 테넌트들에 대해 MedRec 어플리케이션을 실행시키기 위해, 시스템 관리자는 하나의 MedRec 리소스 그룹 템플릿(286)을 정의할 수 있고, 상기 템플릿에서 배치가능한 리소스들을 선언할 수 있다.

[0053] 도메인-레벨 배치가능한 리소스들과는 달리, 리소스 그룹 템플릿에서 선언 된 배치가능한 리소스들은 템플릿에 완전히 구성되지 않거나, 일부 구성 정보가 없기 때문에 그대로 액티베이트될 수 없다.

[0054] 예를 들면, MedRec 리소스 그룹 템플릿은 어플리케이션들에 의해 사용되는 데이터 소스를 선언할 수 있지만, 데이터베이스로 연결하는 URL을 특정하지 않을 수 있다. 서로 다른 테넌트들과 관련된 파티션들(예를 들어, 파티션 BUC-A(290)(Bayland Urgent Care, BUC) 및 파티션 VH-A(292)(Valley Health, VH)은 MedRec 리소스 그룹 템플릿을 참조(296 및 297)하는 MedRec 리소스 그룹(293 및 294)을 각각 포함함으로써, 하나 이상의 리소스 그룹 템플릿들을 참조할 수 있다. 이후, 상기 참조는 Bayland Urgent Care 테넌트에 의한 사용을 위해 BUC-A 파티션과 관련된 가상 호스트 baylandurgentcare.com(304); 및 Valley Health 테넌트에 의한 사용을 위해 VH-A 파티션과 관련된 가상 호스트 valleyhealth.com(308)을 포함하여, 각각의 테넌트에 대한 가상 타겟들/가상 호스트들을 생성(302 및 306)하는데 사용될 수 있다.

[0055] 도 5는 일 실시예에 따른, 예시적인 멀티-테넌트 환경을 추가로 도시한다. 도 5에 도시된 바와 같이, 2개의 파티션들이 MedRec 리소스 그룹 템플릿을 참조하는 본 예시에 계속하여, 일 실시예에 따르면, 서블릿 엔진(servlet engine)(310)은 복수의 테넌트 환경들을 지원하는데 사용될 수 있고, 본 예시에서 상기 복수의 테넌트 환경들은 Bayland Urgent Care Physician 테넌트 환경(320)과 Valley Health Physician 테넌트 환경(330)이다.

[0056] 일 실시예에 따르면, 각각의 파티션(321 및 331)은 해당 테넌트 환경에 대한 인커밍 트래픽(incoming traffic)을 수신하기 위한 서로 다른 가상 타겟, 및 상기 파티션과 그 리소스들(324 및 334)을 연결하는 서로 다른 URL(322 및 332)을 정의할 수 있고, 본 예시에서 Bayland Urgent Care 데이터베이스 또는 Valley Health 데이터베이스를 각각 포함한다. 동일한 어플리케이션 코드가 두 데이터베이스 모두에 대해 실행될 것이므로, 데이터베이스 인스턴스들은 호환가능한 스키마를 사용할 수 있다. 시스템이 파티션들을 시작할 때, 상기 파티션은 각각의 데이터베이스 인스턴스들에 대한 가상 타켓들 및 연결 풀들(pools)을 생성할 수 있다.

[0057] **파티션 식별자들**

[0058] 일 실시예에 따르면, 어플리케이션 서버 환경 내에서, JNDI(자바 네이밍 디렉토리 인터페이스), JMX(자바 관리 확장들), 로깅(logging), 웹 컨테이너 및 RCM(자원 소비 관리)과 같은 컨테이너는 파티션-특정 프로세싱, 로깅 및 분리 요구들에 사용하기 위해, 현재의 또는 실행중인 요청의 파티션 ID를 결정한다.

[0059] 일 실시예에 따르면, 컴포넌트 호출 요청(component invocation request) 또는 어플리케이션 서버에서 일반적으로 수신된 임의의 다른 요청에 대해, 상기 요청의 타겟 리소스는 도메인의 파티션 내에 상주할 수 있다. 타겟 리소스(즉, 상기 요청의 타겟 리소스)가 상주하는 파티션은 상기 요청에 대한 파티션 컨텍스트(partition context)를 결정할 수 있다.

[0060] 일 실시예에 따르면, 내부 서버 액티베이션들은 다르게 취급될 수 있다. 예를 들어, 내부 서버 액티베이션들은 도메인 내의 특정/개별 파티션들(예를 들어, 파티션에 속한 로그 파일의 순환)을 위해 특별히 수행될 수 있다. 이러한 내부 서버 액티베이션들에 대해, 파티션 컨텍스트는 상기 타겟 리소스가 상주하는 파티션 대신 상기 액티비티가 수행되는 파티션이 될 수 있다.

[0061] 일 실시예에 따르면, 다른 내부 서버 액티베이션들은 임의의 개별/특정 파티션에 기인하지 않는다. 이러한 내부 서버 액티베이션들은 서버 런타임의 초기화 및/또는 셧다운(shutdown), 컨테이너들의 로딩 등을 포함하나, 이에 제한되지 않는다. 특정 파티션에 기인하지 않는 이러한 액티베이션에 대해, 파티션 컨텍스트를 일반 값으로 설정할 수 있다.

[0062] 도 6은 일 실시예에 따른, 멀티테넌트 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 도시한다. 도 6에 도시된 실시예에서, 어플리케이션 서버 환경(600)은 도메인(620)을 포함한다. 도메인(620)은 프론트 엔드 컴포넌트(front end component)(630)와 관련될 수 있고, 컨테이너(640), CIC(컴포넌트 호출 컨텍스트) API(645),

컴포넌트 호출 컨텍스트 매니저(CIC 매니저)(650), 파티션 A(660), 파티션 B(670), 가상 타겟 A(662) 및 가상 타겟 B(672)를 포함할 수 있다. 프론트 엔드 컴포넌트는 인커밍 요청들(예컨데, 예를 들면, Oracle™ Traffic Director)을 처리할 수 있는 임의의 프론트 엔드 컴포넌트일 수 있다. 상기 컨테이너(640)는 파티션 인식 컨테이너(예컨데, 예를 들면, JNDI, JMX, 로깅, RMC 등)일 수 있다. 파티션 A 및 파티션 B는 리소스 그룹들(661 및 671)을 각각 포함할 수 있다. 리소스 그룹들(661 및 671)은 가상 타겟 A 및 가상 타겟 B와 각각 관련될 수 있다. CIC 매니저(650)는 스레드-로컬 스택(thread-local stack)(651)과 같은 스택을 포함할 수 있다.

[0063] 일 실시예에 따르면, 사용자 또는 프로세스(610)는 요청(615)을 개시할 수 있고, 상기 요청은 도메인(620) 내의 요청의 타겟에 대한 타겟 정보를 포함할 수 있다. 예를 들면, 파티션 A(660)의 허가된 사용자는 파티션 A 내의 특정한 어플리케이션이 실행될 것을 요청할 수 있다. 이 상황에서, 요청(615)과 관련된 타겟 정보는, 적어도, 파티션 A를 요청의 타겟으로서 식별할 것이다.

[0064] 일 실시예에 따르면, 타겟 정보(615)를 포함하는 요청과 같은 요청이 도메인(620)으로 향하는 경우, 트래픽 디렉터와 같은 프론트 엔드 컴포넌트(630)로 라우트(route)될 수 있다. 일단 프론트 엔드 컴포넌트(630)에서 수신되면, 컨테이너(640)는 요청의 타겟 컴포넌트/리소스의 다양한 세부 사항들을 결정할 수 있다. 이러한 세부 사항들은 예를 들어, 상기 요청내에 포함된 타겟 정보를 검토함으로써 획득될 수 있다. 일단 컨테이너(640)가 요청의 타겟 컴포넌트/리소스를 결정하면, 컨테이너는 호출을 나타내는 객체로서 리턴(return)될 수 있는 컴포넌트 호출 컨텍스트(652)를 생성 할 수 있다. 일 실시예에 따르면, 결정된 컴포넌트 호출 컨텍스트(652)는 요청에 따라 파티션 식별자(665 또는 675)와 관련될 수 있다.

[0065] 인커밍 요청의 세부 사항들을 결정하고 컴포넌트 호출 컨텍스트 객체를 생성하는 컨테이너의 예로서, 인커밍 요청(615)이 HTTP 요청이라고 가정한다. 인커밍 HTTP 요청의 URI를 통해, 웹(HTTP) 컨테이너는 HTTP 요청을 타겟으로 하는 파티션 ID, 응용 프로그램 ID, 모듈 및 컴포넌트 ID를 결정할 수 있다. 상기 결정은, 예를 들면, 상기 요청의 타겟이 식별될 수 있도록 충분한 정보가 상기 요청에서 판독된 이후에, 소켓 판독기 스레드(socket reader thread)에서 HTTP 프로토콜 핸들러에 의해 발생할 수 있다. 상기 정보에 기초하여, 컨테이너는 인커밍 HTTP 요청에 대한 객체, 컴포넌트 호출 컨텍스트를 생성할 수 있다.

[0066] 일 실시예에 따르면, 요청이 도메인(620)으로 향하는 경우, 이는 트래픽 디렉터와 같은 프론트 엔드 컴포넌트(630)로 라우트될 수 있다. 일단 프론트 엔드 컴포넌트(630)에서 수신되면, 컨테이너(640)는 요청의 타겟 컴포넌트/리소스의 다양한 세부 사항들을 결정할 수 있다. 이러한 세부 사항들은, 예를 들어, 상기 요청내에 포함된 타겟 정보를 검토함으로써 획득될 수 있다. 일단 컨테이너(640)가 상기 요청의 타겟 컴포넌트/리소스를 결정하면, 컨테이너는 호출을 나타내는 객체 일 수 있는 컴포넌트 호출 컨텍스트(652)를 생성할 수 있다. 일 실시예에 따르면, 결정된 컴포넌트 호출 컨텍스트(652)는 파티션 식별자(665 또는 675)와 관련될 수 있다.

[0067] 일 실시예에 따르면, 일단 컨테이너가 결정을 내리고 인커밍 요청(즉, 객체)의 컴포넌트 호출 컨텍스트를 생성하면, 컨테이너는 컴포넌트 호출 컨텍스트 매니저(650)에 컴포넌트 호출 컨텍스트를 등록시킬 수 있으며, 상기 컴포넌트 호출 컨텍스트 매니저(650)는, 예를 들면, 다양한 컴포넌트 호출 컨텍스트들(예를 들면, 컴포넌트 호출 컨텍스트(A)-(D)(652 내지 655))을 차례대로 포함하는 스레드-로컬 스택(651)을 포함할 수 있다. 스레드-로컬 스택(651)은 임의의 수의 컴포넌트 호출 컨텍스트들을 포함할 수 있고, 도 6에 도시된 바와 같은 4개의 컴포넌트 호출 컨텍스트들로 제한되지 않는다는 것을 유의해야 한다. 컴포넌트 호출 컨텍스트 매니저(650)는 현재 컴포넌트 호출 컨텍스트의 검색(look-up), 등록 및 등록 취소를 허용할 수 있다.

[0068] 일 실시예에 따르면, 컨테이너는 CIC API(645)를 통해 컴포넌트 호출 컨텍스트 객체를 컴포넌트 호출 컨텍스트 매니저에 직접적으로 등록시킬 수 있다. 컨테이너는 상기 요청에 대해 결정된 컴포넌트 호출 컨텍스트를 푸시(push)하기 위해, pushComponentInnvocationContext와 같은 커맨드를 사용하여 컴포넌트 호출 컨텍스트의 상기 직접적인 등록을 달성할 수 있다. 새로운 컴포넌트 호출 컨텍스트를 컴포넌트 호출 컨텍스트 매니저에 등록시키기 위해 이러한 방법을 활용하는 컨테이너는 또한, 일단 상기 요청과 관련된 호출이 완료되면 컴포넌트 호출 컨텍스트의 설정 해제/등록 취소를 처리할 수 있다. 호출이 성공적인 완료 이후에만, 컨테이너가 컴포넌트 호출 컨텍스트를 컴포넌트 호출 컨텍스트 매니저에서 등록 취소/설정 해제하는 것이 요구사항이 아니다. 실패적인 호출(예컨데, 예외가 발생한 호출)은 또한, 컴포넌트 호출 컨텍스트를 컴포넌트 호출 매니저에서 설정 해제/등록 취소하는 컨테이너를 야기할 수 있다. 컴포넌트 호출 컨텍스트 매니저에서 컴포넌트 호출 컨텍스트의 설정 해제/등록 취소는 컴포넌트 호출 컨텍스트 매니저 상에서 popComponentInvocationContet과 같은 명령을 통해 수행될 수 있다.

[0069] 일 실시예에 따르면, 컨테이너는 CIC API(645)를 통해 컴포넌트 호출 컨텍스트 객체를 컴포넌트 호출 컨텍스트

매니저에 직접적으로 등록시킬 수 있다. 컨테이너는 실행시키기 위해 callabe를 전달함으로써, runAs 방법을 사용할 수 있다. 이 상황에서, 컴포넌트 호출 컨텍스트 매니저는 컴포넌트 호출 컨텍스트를 설정할 수 있다. 이는 컨테이너로 하여금 컴포넌트의 호출 컨텍스트의 컨텍스트에서 액션을 동작시키고, 상기 액션을 실행하고, 호출이 완료된 이후에 컴포넌트의 호출 컨텍스트를 설정 해제할 수 있도록 한다.

[0070] 일 실시예에 따르면, 컨테이너는 CIC API(645)를 통해 컴포넌트 호출 컨텍스트 객체를 컴포넌트 호출 컨텍스트 매니저에 직접적으로 등록시킬 수 있다. 컨테이너는 try-with-resources 스타일 블록 내의 setCurrentComponentInnvocationContext 방법에 의해 리턴된 오토클로저블(AutoCloseable)을 사용할 수 있다. 이 경우, CIC 매니저는 오토클로저블을 통해 try 블록의 끝에서 호출 컨텍스트를 자동으로 설정 해제할 수 있다. 이 접근법은 호출자(caller)가 런어블(Runnable) 또는 클로저블로 작업을 수정하지 않고, CIC 매니저가 작업을 실행하는 동안 올바른 CIC 컨텍스트를 설정하는 것과 같은 장점들을 제공할 수 있다.

[0071] 도 7은 일 실시예에 따른, 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 도시한다. 도 7에 도시된 실시예에서, 어플리케이션 서버 환경(600)은 도메인(620)을 포함한다. 도메인(620)은 프론트 엔드 컴포넌트(630)와 관련될 수 있고, 서비스 제공자 인터페이스(720)를 포함할 수 있는 컨테이너(640)를 포함할 수 있다. 도메인은 작업 매니저(work manager)(730), 컴포넌트 호출 컨텍스트 매니저(CIC manager)(650), 파티션 A(660), 파티션 B(670), 가상 타겟 A(662) 및 가상 타겟 B(672)를 더 포함할 수 있다. 프론트 엔드 컴포넌트는 인커밍 요청들(예컨데, 예를 들면, Oracle™ Traffic Director)을 처리할 수 있는 임의의 프론트 엔드 컴포넌트일 수 있다. 상기 컨테이너(640)는 파티션 인식 컨테이너(예컨데, 예를 들면, JNDI, JMX, 로깅, RMC 등)일 수 있다. 파티션 A 및 파티션 B는 리소스 그룹(661 및 671)을 각각 포함할 수 있다. 리소스 그룹들(661 및 671)은 가상 타겟 A 및 가상 타겟 B와 각각 관련될 수 있다. CIC 매니저(650)는 스레드-로컬 스택(651)과 같은 스택을 포함할 수 있다.

[0072] 일 실시예에 따르면, 사용자는 요청(615)을 개시할 수 있고, 상기 요청은 도메인(620) 내의 요청의 타겟에 대한 타겟 정보를 포함할 수 있다. 예를 들면, 파티션 A(660)로의 액세스를 가진 사용자, 프로세서 또는 다른 어플리케이션은 파티션 A 내의 특정한 어플리케이션이 실행될 것을 요청할 수 있다. 이 상황에서, 요청(615)과 관련된 타겟 정보는, 적어도, 파티션 A를 요청의 타겟으로서 식별할 것이다. 요청이 다른 파티션(동일한 도메인 내의 다른 파티션, 또는 다른 도메인으로부터의 파티션)에서 발생하는 상황에서는, 발생된 파티션의 컴포넌트 호출 컨텍스트가 상기 요청과 함께 포함되지 않는다.

[0073] 일 실시예에 따르면, 타겟 정보(615)를 포함하는 요청과 같은 요청이 도메인(620)으로 향하는 경우, 트래픽 디렉터와 같은 프론트 엔드 컴포넌트(630)로 라우트될 수 있다. 일단 프론트 엔드 컴포넌트(630)에서 수신되면, 컨테이너(640)는 요청의 타겟 컴포넌트/리소스의 다양한 세부 사항들을 결정할 수 있다. 이러한 세부 사항들은 예를 들어, 요청내에 포함된 타겟 정보를 검토함으로써 획득될 수 있다. 일단 컨테이너(640)가 요청의 타겟 컴포넌트/리소스를 결정하면, 컨테이너는 호출을 나타내는 객체일 수 있는 컴포넌트 호출 컨텍스트(652)를 생성할 수 있다. 일 실시예에 따르면, 결정된 컴포넌트 호출 컨텍스트(652)는 요청에 따라 파티션 식별자(665 또는 675)와 관련될 수 있다.

[0074] 일 실시예에 따르면, 일단 컨테이너가 결정을 내리고 인커밍 요청(즉, 객체)의 컴포넌트 호출 컨텍스트를 생성하면, 컨테이너는 컴포넌트 호출 컨텍스트로 하여금 컴포넌트 호출 컨텍스트 매니저(650)에 간접적으로 등록되도록 할 수 있으며, 상기 컴포넌트 호출 컨텍스트 매니저(650)는, 예를 들면, 다양한 컴포넌트 호출 컨텍스트들(예를 들면, 컴포넌트 호출 컨텍스트(A)-(D)(652 내지 655))을 차례대로 포함하는 스레드-로컬 스택(651)을 포함할 수 있다. 스레드-로컬 스택(651)은 임의의 수의 컴포넌트 호출 컨텍스트들을 포함할 수 있고, 도 7에 도시된 바와 같은 4개의 컴포넌트 호출 컨텍스트들로 제한되지 않는다. 컴포넌트 호출 컨텍스트 매니저(650)는 현재 컴포넌트 호출 컨텍스트의 검색, 등록 및 등록 취소를 허용할 수 있다.

[0075] 일 실시예에 따르면, 컴포넌트 호출 컨텍스트의 간접적인 등록은 작업 매니저(730)에 의해 수행될 수 있다. 예를 들면, 컨테이너(640)는 작업 매니저(730)를 통해, 별도의 스레드에서 비동기적으로 상기 요청의 다음-레벨 프로세싱을 발송(dispatch)할 수 있다. SPI는 스레드에서 컴포넌트 컨텍스트 설정을 지원(facilitate) 할 수 있다. 컴포넌트 요청 인터페이스(710)는 컨테이너(640)에 의해 제공될 수 있으며, 상기 컨테이너(640)는 요청과 관련된 작업으로 하여금 자신의 컴포넌트 호출 컨텍스트에 따라 실행되도록 한다. 또한, 컴포넌트 요청 인터페이스(710)는 컨테이너가 타겟 컴포넌트 세부 사항들(즉, 컴포넌트 호출 컨텍스트 리턴 값)을 제공하게 하기 위해, getComponentInnvocationContext와 같은 커맨드를 포함할 수 있다. 컴포넌트 요청(710) 인스턴스가 실행되기 전에, 작업 매니저(730)는 컴포넌트 호출 컨텍스트 매니저(650)를 검색하고 getComponentInvocationContext

명령에 의해 제공된 컴포넌트 호출 컨텍스트를 사용하여 새로운 컴포넌트 호출 컨텍스트를 등록할 수 있다. 일단 컴포넌트 요청(710) 인스턴스의 실행이 완료되면, 작업 매니저(730)는 컴포넌트 호출 컨텍스트 매니저로부터 컴포넌트 호출 컨텍스트를 자동으로 제거할 수 있다.

[0076] 일 실시예에 따르면, 요청은 일반적으로 하나의 스레드와 관련되기 때문에, CIC 매니저는 현재 호출 요청에 대한 스레드-로컬 스택을 사용함으로써 스레드 기반 상으로 CIC 상태를 유지할 수 있다. 요청에 대한 작업이 복수의 스레드에 의해 처리되는 상황들에서는, 서로 다른 스레드에서 작업을 실행하는 컨테이너는, 상기 설명된 바와 같이, 스레드의 CIC가 새로운 스레드 상으로 전달되는 것을 보장하기 위해 getComponentInvocationContext 명령을 활용할 수 있다.

[0077] 도 8은 일 실시예에 따른, 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 도시한다. 도 8에 도시된 실시예에서, 어플리케이션 서버 환경(600)은 도메인(620)을 포함한다. 도메인(620)은 프론트 엔드 컴포넌트(630), 컨테이너(640), CIC API(645), 컴포넌트 호출 컨텍스트 매니저(CIC manager)(650), 파티션 A(660), 파티션 B(670), 가상 타겟 A(662) 및 가상 타겟 B(672)를 포함한다. 프론트 엔드 컴포넌트는 인커밍 요청들(예컨데, 예를 들면, Oracle™ Traffic Director)을 처리할 수 있는 임의의 프론트 엔드 컴포넌트일 수 있다. 상기 컨테이너(640)는 파티션 인식 컨테이너(예컨데, 예를 들면, JNDI, JMX, 로깅, RMC 등)일 수 있다. 파티션 A 및 파티션 B는 리소스 그룹들(661 및 671)을 각각 포함할 수 있다. 리소스 그룹들(661 및 671)은 가상 타겟 A 및 가상 타겟 B와 각각 관련될 수 있다. CIC 매니저(650)는 스레드-로컬 스택(651)과 같은 스택을 포함할 수 있다.

[0078] 일 실시예에 따르면, 컨테이너(640)는 getCurrentComponentInvocation과 같은 명령을 통해 API(645)를 통해 컴포넌트 호출 매니저에서 현재 컴포넌트 호출 컨텍스트를 검색할 수 있다. 예를 들면, 도 8에 도시된 바와 같이, 컴포넌트 호출 컨텍스트 매니저는 가능한 컴포넌트 호출 컨텍스트들(예를 들면, 컴포넌트 호출 컨텍스트(A)-(D)(652 내지 655)) 중 컴포넌트 호출 컨텍스트(652) 상에 현재 설정된다. 컨테이너(640)는 API(645)를 통해 컴포넌트 호출 컨텍스트 매니저(650)로부터 현재 컴포넌트 호출 컨텍스트(810)를 얻을 수있다.

[0079] 도 9는 일 실시예에 따른, 멀티테넌트 어플리케이션 서버 환경에서 파티션 식별자들의 결정을 위한 방법을 흐름도를 통해 도시한다. 예시적인 방법(900)은, 단계(910)에서, 하나 이상의 컴퓨터들에서, 상기 하나 이상의 컴퓨터들 상에서 실행되는 어플리케이션 서버 환경을 포함하는 것을 제공하는 것으로 시작될 수 있으며, 상기 어플리케이션 서버 환경은 상기 어플리케이션 서버 환경 내에서 사용될 수 있는 복수의 배치가능한 리소스들, 하나 이상의 파티션들, 각각의 파티션은 도메인의 관리 및 런타임 서브디비전을 제공하며, 그리고 컴포넌트 호출 컨텍스트 매니저를 포함하고, 상기 컴포넌트 호출 컨텍스트 매니저는 스택을 포함한다. 상기 방법은, 단계(920)에서, 파티션-인식 컨테이너에서 하나 이상의 컴포넌트 호출 컨텍스트들을 설정할 수 있는 방법을 계속한다. 단계(930)에서, 상기 예시적인 방법은 상기 파티션-인식 컨테이너에 의해 현재 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에 등록시키는 것, 또는 상기 컴포넌트 호출 컨텍스트 매니저에서 현재 컴포넌트 호출 컨텍스트를 검색하는 것 중 하나를 수행하는 것을 계속한다. 상기 예시적인 방법은, 단계(940)에서, 상기 현재 컴포넌트 호출 컨텍스트를 현재 파티션과 관련시킴으로서 완료될 수 있다.

[0080] **컴포넌트 호출 컨텍스트의 전파**

[0081] 일 실시예에 따르면, 컴포넌트 호출 컨텍스트 매니저는 관련된 요청의 지속 기간 동안 컴포넌트 호출 컨텍스트 세부 사항들의 전파를 담당할 수 있다. 파티션 컨텍스트 정보는 로컬 호출을 통해 전파될 수 있다. 컴포넌트 호출 컨텍스트의 다른 서브-컨텍스트들은 요청을 처리함에 있어 어떤 컴포넌트가 관련되는지에 따라, 각각의 컨테이너들에 의해 결정될 수 있다. 컴포넌트 호출 컨텍스트는, 새로운 API가 스레드를 생성하는데 사용되는 것과 관계없이, 새로운 스레드가 생성될 때마다 자동으로 상속(inherite)될 수 있다.

[0082] **컴포넌트 호출 컨텍스트 검색**

[0083] 전술한 바와 같이, 일 실시예에 따르면, 컴포넌트 호출 컨텍스트 세부 사항들(예를 들어, 파티션 ID)을 요구하는 임의의 내부 코드 또는 컨테이너(예컨데, JNDI)는 현재 컴포넌트 호출을 결정하기 위해, getCurrentComponentInvocation과 같은 방법을 통해 컴포넌트 호출 컨텍스트 매니저를 검색할 수 있다. 컴포넌트 호출 컨텍스트의 파티션 ID는 글로벌 파티션 ID 및 사용자 파티션 ID를 포함하여 여러 값들을 가질 수 있다.

[0084] 일 실시예에 따르면, 글로벌 파티션 ID는 다음 상황들에서 사용될 수 있다. 서버가 요청의 타겟 컴포넌트가 글로벌 파티션(즉, 도메인)에 상주한다고 결정하면, 컨테이너는 파티션 ID를 글로벌 파티션의 ID로 설정할 수 있다. 추가적으로, 글로벌 파티션 ID는 사용자 파티션에 특정되지 않은 내부 서버 액티비티들(예를 들면, 서버 런타임의 초기화/셧다운, 컨테이너의 로딩 등)에 사용될 수 있다. 서버 런타임에서 이러한 액티비티들이 파티션

대신 특별히 수행되지 않으면, 컨테이너는 파티션 ID를 글로벌 파티션의 ID로 설정할 수 있다. 다른 실시예들은 파티션 ID가 공유 시스템 파티션을 나타내도록 함으로써, 글로벌 파티션 작업과 공유 작업을 구별할 수 있다.

[0085] 일 실시예에 따르면, 사용자 파티션 ID는 다음 상황들에서 사용될 수 있다. 요청이 사용자 파티션에 있는 컴포넌트를 타겟으로 하는 경우, 서버는 파티션 ID를 사용자 파티션의 ID로 설정할 수 있다. 추가적으로, 만일 임의의 서버 액티비티가 파티션(예를 들어, 특정 파티션에 대한 로그 파일들의 자동적인 로테이션(rotation))에 대해 특별히 수행되는 경우, 파티션의 ID는 컴포넌트 호출 컨텍스트에서 설정될 수 있다.

[0086] 일 실시예에 따르면, 현재 컴포넌트 호출 컨텍스트는 non-null 값이다. 예를 들면, 스레드가 설정된 컨텍스트를 갖지 않은 상황에서, 글로벌 파티션을 나타내는 non-null 컴포넌트 호출 컨텍스트(즉, 어플리케이션, 모듈 및 컴포넌트 ID와 이름들에 대해 null 값들을 갖는 non-null 컴포넌트 호출 컨텍스트)가 리턴될 수 있다 또 다른 예로서, 스레드가 생성되고 해당 스레드에 새로운 컨텍스트가 설정되지 않은 상황에서, 상기 스레드는 생성 스레드의 컨텍스트를 자동으로 상속할 수 있다.

[0087] **컴포넌트 호출 컨텍스트 스택**

[0088] 일 실시예에 따르면, 요청이 다수의 컴포넌트들에 걸친(span) 상황들(예를 들어, 다른 컴포넌트에 대한 로컬 호출시-예를 들면, 엔터프라이즈 자바빈즈(enterprise JabaBean)(EJB)를 호출하는 서블릿)에서, 타겟 컨테이너(본 예시에서, EJB 컨테이너)는 EJB 호출을 처리하기 전에 새로운 모듈 및 컴포넌트 ID를 포함하는 새로운 컴포넌트 호출 컨텍스트를 설정할 수 있다. 또한, 타겟 컨테이너는 EJB 호출의 성공적인 완료시, 컴포넌트 호출 컨텍스트를 설정 해제할 수 있다. 컴포넌트 호출 컨텍스트 매니저는 컴포넌트 호출의 목록을 스택으로서 내부적으로 관리할 수 있다. 스택은 호출의 컨텍스트 내에서 만들어진 로컬 호출들과 관련된 컴포넌트 호출 컨텍스트들을 나타낼 수 있다.

[0089] 예시로서, 서블릿 S1이 로컬 EJB E1를 호출하는 상황에서, 컴포넌트 호출 컨텍스트 매니저는 서블릿 S1 및 EJB E1에 대응하는 컴포넌트 호출 컨텍스트들의 스택을 내부적으로 유지할 수 있다. EJB E1의 실행 중, componentinvocationcmanager.getCurrentComponentInvocationContext()에 대한 호출은 E1에 대응하는 컴포넌트 호출 컨텍스트를 현재 컴포넌트 호출 컨텍스트로서 리턴할 수 있다. EJB E1의 실행이 완료되자마자, EJB 컨테이너는 E1에 대응하는 컴포넌트 호출 컨텍스트를 설정 해제할 수 있고, 이후 컴포넌트 호출 컨텍스트 매니저는 S1에 대응하는 컴포넌트 호출 컨텍스트를 componentinvocationcontextmanager.getCurrentComponentInvocationContext()에 대한 임의의 후속 호출들에 대한 현재 호출 컨텍스트로 설정할 것이다.

[0090] 일 실시예에 따르면, 스택 내 모든 호출 객체들을 검색하기 위한 내부적인 방법이 또한 이용가능하게 될 수 있다.

[0091] **API**

[0092] 일 실시예에 따른, API 클래스들 및 인터페이스들의 예시들이 본 명세서에 나열된다:

[0093] package weblogic . invocation;

[0094] /**

[0095] * Information about the current request executed by a component in the server.

[0096] * Currently the information exposed in this interface is related to the

[0097] * target component which is executing the request. In future,

[0098] * more information can be exposed via this interface.

[0099] */

[0100] public interface ComponentInvocationContext {

[0101] /**

[0102] * Returns the ID of the Partition Id that the component that's being

[0103] * invoked resides. If the component belongs to no partition or belongs to the

```
[0104]      * GLOBAL partition, this method must return the GLOBAL partition's uuid.
[0105]      *
[0106]      * @ return the Partition UUID
[0107]      */
[0108]      String getPartitionId() ;
[0109]      /**
[0110]      * This method is similar to #getPartitionId except that it returns the
[0111]      * human readable name instead of ID.
[0112]      *
[0113]      * @return the Partition name
[0114]      */
[0115]      String get.PartitionName();
[0116]      /**
[0117]      * Return if the partition is global or not
[0118]      *
[0119]      * @return boolean value
[0120]      */
[0121]      boolean isGlobalRuntime ();
[0122]      /**
[0123]      * Returns the ID of the application that, contains the component
[0124]      * being invoked currently. It is possible that the component does not
[0125]      * belong to any application in which case it returns null. This can
[0126]      * happen while accessing WLS MBeans for example.
[0127]      *
[0128]      * When there are multiple versions of the same application
[0129]      * or when an application is deployed to multiple partitions,
[0130]      * application name remains same for all those instances where as
[0131]      * application ID is unique for each instance.
[0132]      *
[0133]      * @return the Application ID
[0134]      */
[0135]      String getApplicationId();
[0136]      /**
[0137]      * This method is similar to #getApplicationId except that it returns the
[0138]      * human readable name instead of ID,
[0139]      * Application name can't uniquely identify an application instance.
```

[0140] *

[0141] * @return the Application name

[0142] */

[0143] String getApplicationName();

[0144] /**

[0145] * Return the version of he application containing the component being

[0146] * invoked.

[0147] *

[0148] * @return the Application Version

[0149] */

[0150] String getApplicationVersion();

[0151] /**

[0152] * Return the name of the application module containing the component

[0153] * being invoked. In case of standalone module deployments,

[0154] * this is same as the name of the application.

[0155] *

[0156] * <p>

[0157] * This method returns null whenever #getApplicationId returns null.

[0158] * It can return null, even if #getApplicationId returns non-null. This

[0159] * happens when the invocation is in the context of the application, but

[0160] * not a particular module. For instance, when an

[0161] * ApplicationLifecycleListener receives callbacks.

[0162] *

[0163] * @return the Module Name

[0164] */

[0165] String getModuleName();

[0166] /**

[0167] * Return the name of the component being invoked.

[0168] *

[0169] * @return the Component Name

[0170] */

[0171] String getComponentName();

[0172] }

[0173] 일 실시예에 따른, API 클래스 ComponentInvocationContextManager의 예시적인 구현이 아래에 제공된다:

[0174] package weblogic.invocation;

[0175] /**

```
[0176]  * Allows containers in WebLogic to lookup, register and unregister the
[0177]  * current component invocation context.
[0178]  */
[0179]  public abstract class ComponentInvocationContextManager {
[0180]  /**
[0181]  * Returns the instance of the <code>ComponentInvocationManager</code> .
[0182]  * The implementation of the ComponentInvocationContextManager is loaded
[0183]  * through the Java SE Service Provider mechanism
[0184]  */
[0185]  public static ComponentInvocationContextManager getInstance() {...};
[0186]  /**
[0187]  * Returns the instance of the <code>ComponentInvocationManager</code> . The
[0188]  * implementation of the ComponentInvocationContextManager is loaded through
[0189]  * the Java SE Service Provider mechanism.
[0190]  *
[0191]  * Since there is a relatively expensive security check while calling this
[0192]  * method, containers should cache the instance of
[0193]  * <code>ComponentInvocationContextManager</code> returned by this method for
[0194]  * performance reasons.
[0195]  *
[0196]  * @param subject An <code>AuthenticatedSubject</code> representing the Kernel
[0197]  * identity
[0198]  * @return A <code>ComponentInvocationContextManager</code> instance.
[0199]  *       A read-only instance is returned if the subject does not
[0200]  *       represent the kernel identity
[0201]  * @throws SecurityException when the supplied subject does not represent
[0202]  *                                          the kernel identity.
[0203]  */
[0204]  public static ComponentInvocationContextManager getInstance(Principal subject) {
[0205]  try {
[0206]  SingletonHolder.INSTANCE.checklfKernel(subject);
[0207]  return SingletonHolder.INSTANCE;
[0208]  } catch (SecurityException se) {
[0209]  se.printStackTrace();
[0210]  throw se;
[0211]  }
```

```
[0212]      }
[0213]      /**
[0214]      * Factory method for creating a ComponentInvocationContext.
[0215]      * This is available as a convenience only. Users of this class are free
[0216]      * to create their own implementation of ComponentInvocationContext.
[0217]      * @param applicationId
[0218]      * @param moduleName
[0219]      * @param componentName
[0220]      * @return
[0221]      */
[0222]      public abstract ComponentInvocationContext createComponentInvocationContext(
[0223]      String applicationId,
[0224]      String moduleName,
[0225]      String componentName
[0226]      );
[0227]      /**
[0228]      * Factory method for creating a ComponentInvocationContext.
[0229]      * This is available as a convenience only. Users of this class are free
[0230]      * to create their own implementation of ComponentInvocationContext.
[0231]      * @param partitionName
[0232]      * @param applicationName
[0233]      * @param applicationVersion
[0234]      * @param moduleName
[0235]      * @param componentName
[0236]      * @return
[0237]      */
[0238]      public abstract ComponentInvocationContext createComponentInvocationContext(
[0239]      String partitionName,
[0240]      String applicationName,
[0241]      String applicationVersion,
[0242]      String moduleName,
[0243]      String componentName
[0244]      );
[0245]      /**
[0246]      * Factory method for creating a ComponentInvocationContext.
[0247]      * This is available as a convenience only. Users of this class are free
```

```
[0248]      * to create their own implementation of ComponentInvocationContext.
[0249]      *
[0250]      * calling this method has the same effect of calling
[0251]      * the other factory method with all parameters except partitionName as
[0252]      * null.
[0253]      */
[0254]     public ComponentInvocationContext createComponentInvocationContext (String partitionName);
[0255]     /**
[0256]      * Returns the Component Invocation Context associated with the current
[0257]      * thread. For threads where a CIC is not established, a non-null
[0258]      * ComponentInvocationContext instance representing the global partition
[0259]      * is returned.
[0260]      */
[0261]     public abstract ComponentInvocationContext
[0262]     getCurrentComponentInvocationContext();
[0263]     /**
[0264]      * Allows a container to push a new Component Invocation Context for the
[0265]      * current request.
[0266]      * @param ci The component invocation context to be pushed. This must not be null.
[0267]      */
[0268]     public abstract void pushComponentInvocationContext (ComponentInvocationContext
[0269]     ci);
[0270]     /**
[0271]      * Allows a container to unset the current Component Invocation Context in the
[0272]      * top of the invocation context stack for the current request.
[0273]      */
[0274]     public abstract void popComponentInvocationContext () ;
[0275]     /**
[0276]      * Allows a container to run an action in the context of a
[0277]      * ComponentInvocationContext.
[0278]      *
[0279]      * @param ci     The context in which the action must be performed.
[0280]      * @param action         The action to be performed
[0281]      * @throws ExecutionException when the supplied action aborts by
[0282]      *                                  throwing any exception. Use #getCause to see the
[0283]      *                                  underlying exception.
```

```
[0284]      */
[0285]      public static <T> T runAs (ComponentInvocationContext ci,
[0286]                                 Callable<T> action) throws ExecutionException;
[0287]      /**
[0288]      * Allows a container to run an action in the context of a
[0289]      * ComponentInvocationContext. This method is similar to {#runAs (Callable)}
[0290]      * except that this accepts a Runnable instead of a Callable.
[0291]      *
[0292]      * @param ci     The context in which the action must be performed.
[0293]      * @param action         The action to be performed
[0294]      * @throws ExecutionException when the supplied action aborts by throwing
[0295]      *                                  any exception. Use #getCause to see the
[0296]      *                                  underlying exception.
[0297]      */
[0298]      public static void runAs (ComponentInvocationContext ci,
[0299]      Runnable action) throws ExecutionException;
[0300]      /**
[0301]      * Establishes the provided Component Invocation Context for the current request.
[0302]      *
[0303]      * As this method returns an <code>AutoCloseable</code> instance, this method
[0304]      * may be used in a try-with-resources expression to have the
[0305]      * {@link #ComponentInvocationContextManager() } automatically manage the
[0306]      * establishment and unsetting of the context.
[0307]      *
[0308]      * For example:
[0309]      * <pre>
[0310]      * { @code
[0311]      * ComponentInvocationContextManager cicm =
[0312]      ComponentInvocationContextManager.getInstance(Principal);
[0313]      * ComponentInvocationContext cic = //CIC to establish ...
[0314]      * try (ManagedInvocationContext mic =
[0315]      cicm. setCurrentComponentInvocationContext(cic)) {
[0316]      * ...
[0317]      *       }
[0318]      * }
[0319]      * </pre>
```

[0320] *

[0321] * In this case, the <code>cic</code> is automatically popped at the end of the

[0322] * try block, and an explicit pop/close is not required.

[0323] *

[0324] * @return An <code>AutoCloseable</code> <code>ManagedInvocationContext</code>

[0325] instance .

[0326] */

[0327] public abstract ManagedInvocationContext setCurrentComponentInvocationContext(

[0328] ComponentInvocationContext

[0329] * Adds a listener to the list that is notified every time the

[0330] * component invocation context changes.

[0331] *

[0332] * @param ic the invocation context change listener

[0333] */

[0334] public void

[0335] addInvocationContextChangeListener(ComponentInvocationContextChangeListener ic) ;

[0336] /**

[0337] * Removes a listener from the list that is notified every time the

[0338] * component invocation context changes.direc

[0339] *

[0340] * @param ic the invocation context change listener

[0341] */

[0342] public void

[0343] removeInvocationContextChangeListener(ComponentInvocationContextChangeListener ic);

[0344] }

[0345] 예시적인 API가 여기에 추가적으로 제공된다;

[0346] package weblogic.invocation;

[0347] /**

[0348] * Represents a managed Component Invocation Context and may be used in

[0349] * a try-with-resources statement to establish a component invocation

[0350] * context within the try block, and have it automatically restored at the

[0351] * end of the try block.

[0352] *

[0353] */

[0354] public class ManagedInvocationContext Implements AutoCloseable {

[0355] private final ComponentInvocationContext invocationContext;

```
[0356]  private final Object token;
[0357]  /**
[0358]  * Constructs a Managed Invocation Context
[0359]  *
[0360]  * @param cic the Component Invocation Context to establish
[0361]  * @pararn token an opaque Token provided by a
[0362]  <code>ComponentInvocationContextManager</code>
[0363]  * implementation, so that subsequent pops through
[0364]  * {@link.
[0365]  ComponentInvocationContextManager#_managedPop(ManagedInvocationContext) }
[0366]  * may be verified, to prevent unsetting of inappropriate Contexts.
[0367]  /*
[0368]  public ManagedInvocationContext(ComponentInvocationContext cic, Object token) {
[0369]  this.invocationContext = cic;
[0370]  this.token = token;
[0371]  }
[0372]  /**
[0373]  * Unset the current invocation context from the top of the component
[0374]  * invocation stack of the current request.
[0375]  *
[0376]  * @see java.lang.AutoCloseable#close()
[0377]  * @throws An IllegalStateException if the invocation context cannot be unset,
[0378]  * because of an inappropriate request or an empty stack.
[0379]  */
[0380]  @Override
[0381]  public void close() {
[0382]  //_managedPop verifies the token in this object before popping
        ComponentInvocationContextManager.getInstance() ._managedPop(token);
[0383]  }
[0384]  /**
[0385]  * @see java.lang.Object#toString()
[0386]  */
[0387]  @Override
[0388]  public String toString() {
[0389]  return "ManagedInvocationContext [invocationContext=" + invocationContext + " , token=" + token + "]";
[0390]          }
```

[0391] }

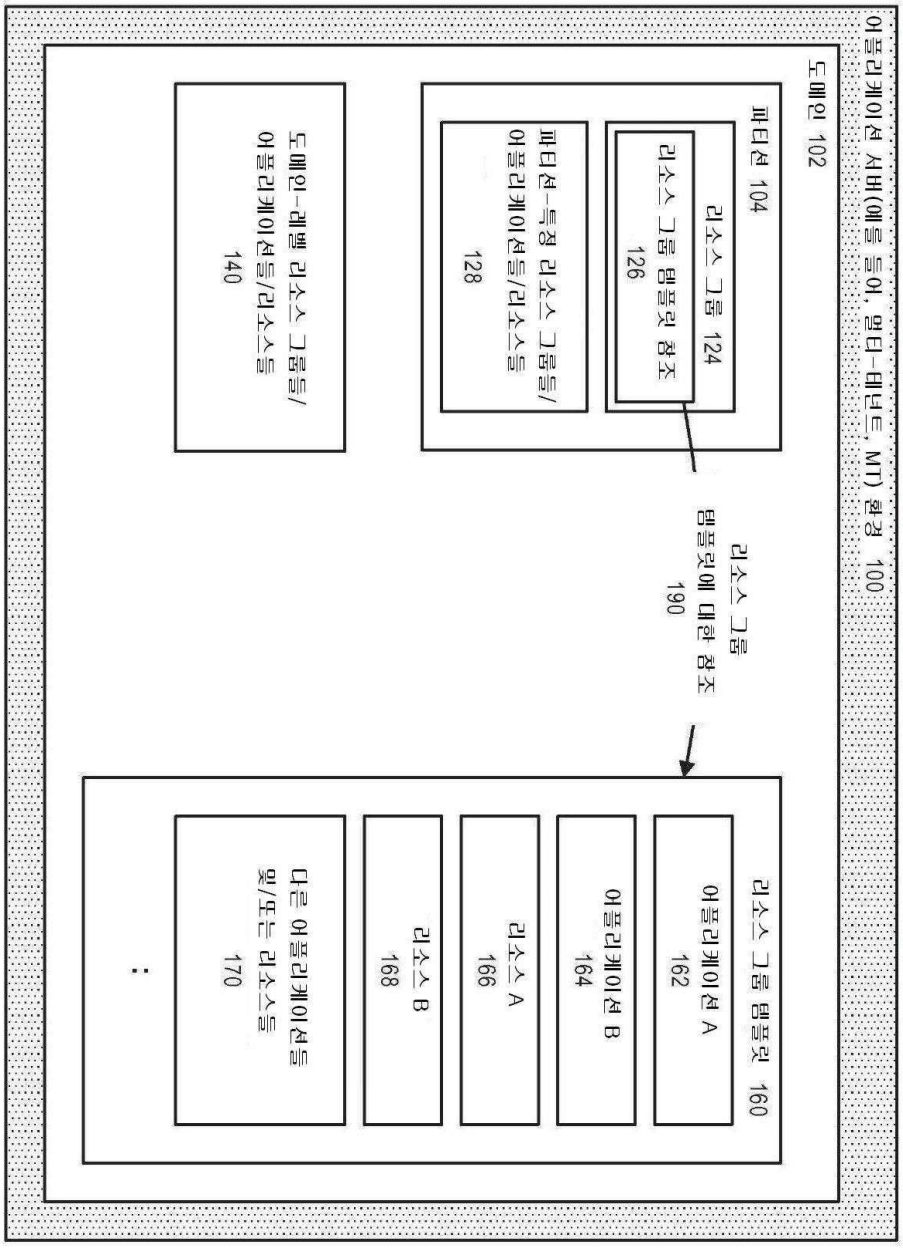
[0392] 본 발명은 본 개시 내용의 교시들에 따라 프로그래밍 된 하나 이상의 프로세서들, 메모리 및/또는 컴퓨터 판독 가능한 저장 미디어를 포함하는 하나 이상의 종래의 범용 또는 특수 디지털 컴퓨터, 컴퓨팅 장치, 머신 또는 마이크로프로세서를 사용하여 편리하게 구현될 수 있다. 적절한 소프트웨어 코딩은 소프트웨어 분야의 통상의 지식을 가진자에게 명백한 바와 같이, 본 개시 내용의 교시들에 기초하여 숙련된 프로그래머들에 의해 용이하게 준비될 수 있다.

[0393] 일부 실시예들에서, 본 발명은 본 발명의 프로세스들 중 임의의 프로세스를 수행하도록 컴퓨터를 프로그래밍하는데 사용될 수 있는 명령어들을 저장하는 비-일시적인 저장 매체 또는 컴퓨터 판독 가능 매체(미디어)인 컴퓨터 프로그램 제품을 포함한다. 상기 저장 매체는 플로피 디스크들, 광디스크들, DVD, CD-ROM들, 마이크로드라이브 및 광-자기 디스크들, ROM들, RAM들, EPROM들, EEPROM들, DRAM들, VRAM들, 플래시 메모리 디바이스들, 광 또는 자기 카드들, 나노시스템들(분자 메모리 IC를 포함한다)을 포함하는 임의의 유형의 디스크, 또는 명령어들 및/또는 데이터를 저장하기에 적합한 임의의 유형의 미디어 또는 디바이스를 포함할 수 있으나, 이에 제한되지 않는다.

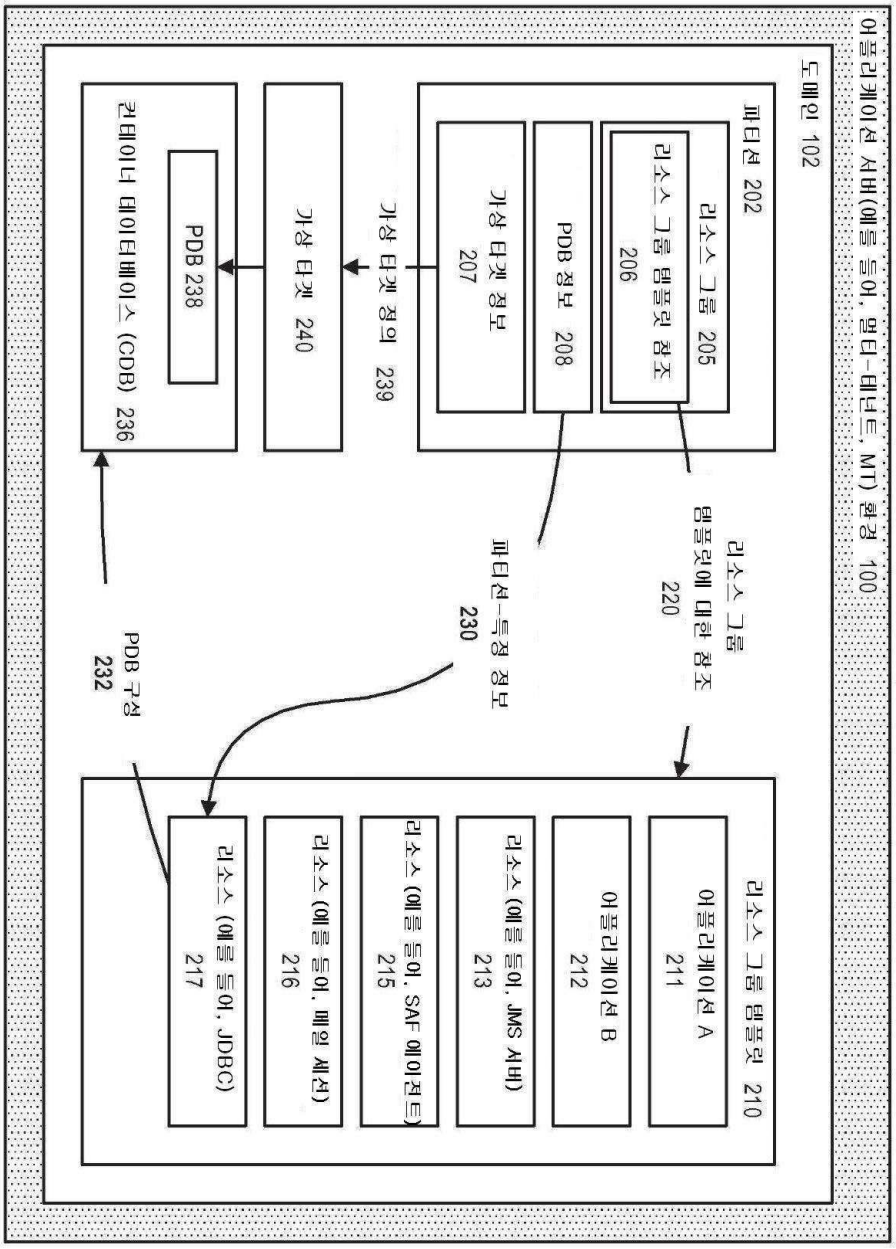
[0394] 본 발명의 상기 설명은 예시 및 설명의 목적으로 제공되었다. 이는 본 발명을 개시된 정확한 형태로 제한하거나 포괄하려는 것은 아니다. 많은 수정들 및 변형들이 당해 기술분야에서 통상의 지식을 가진자에게 명백할 것이다. 수정들 및 변형들은 개시된 구성들의 임의의 관련 조합을 포함한다. 실시예들은 본 발명의 원리들과 그 실제 응용을 가장 잘 설명하고, 당해 기술분야에서 통상의 지식을 가진자가 다양한 실시예들과 의도된 특정 용도에 적합한 다양한 수정들을 통해 본 발명을 이해할 수 있도록 선택 및 설명되었다. 본 발명의 범위는 다음의 청구범위들과 이들의 균등물에 의해 정의되는 것이 의도된다.

도면

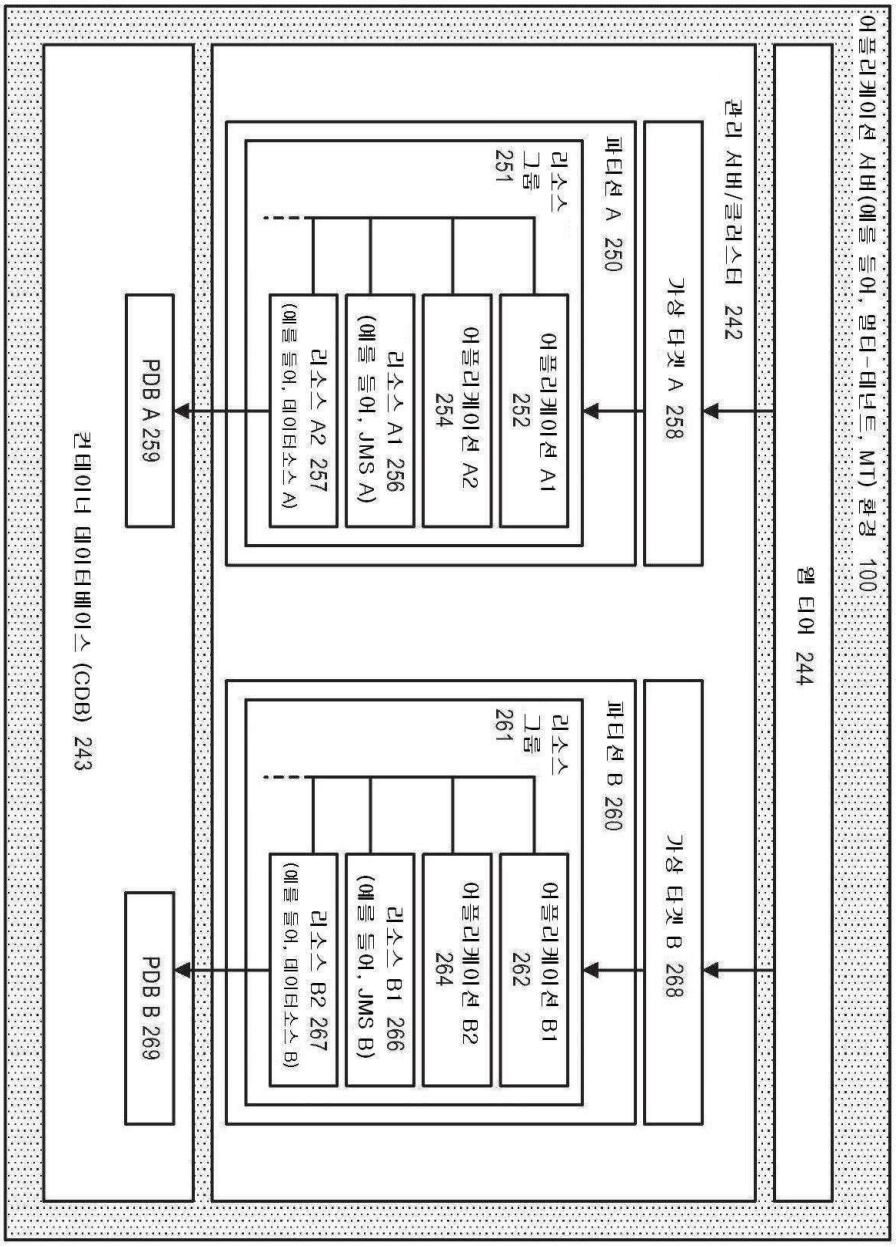
도면1


어플리케이션 서버(예를 들어, 멀티-테넌트, MT) 환경 100
도메인 102
파티션 104
리소스 그룹 124
리소스 그룹 템플릿 참조
126
파티션-특정 리소스 그룹들/
어플리케이션들/리소스들
128
도메인-레벨 리소스 그룹들/
어플리케이션들/리소스들
140
리소스 그룹
템플릿에 대한 참조
190
리소스 그룹 템플릿 160
어플리케이션 A
162
어플리케이션 B
164
리소스 A
166
리소스 B
168
다른 어플리케이션들
및/또는 리소스들
170
:

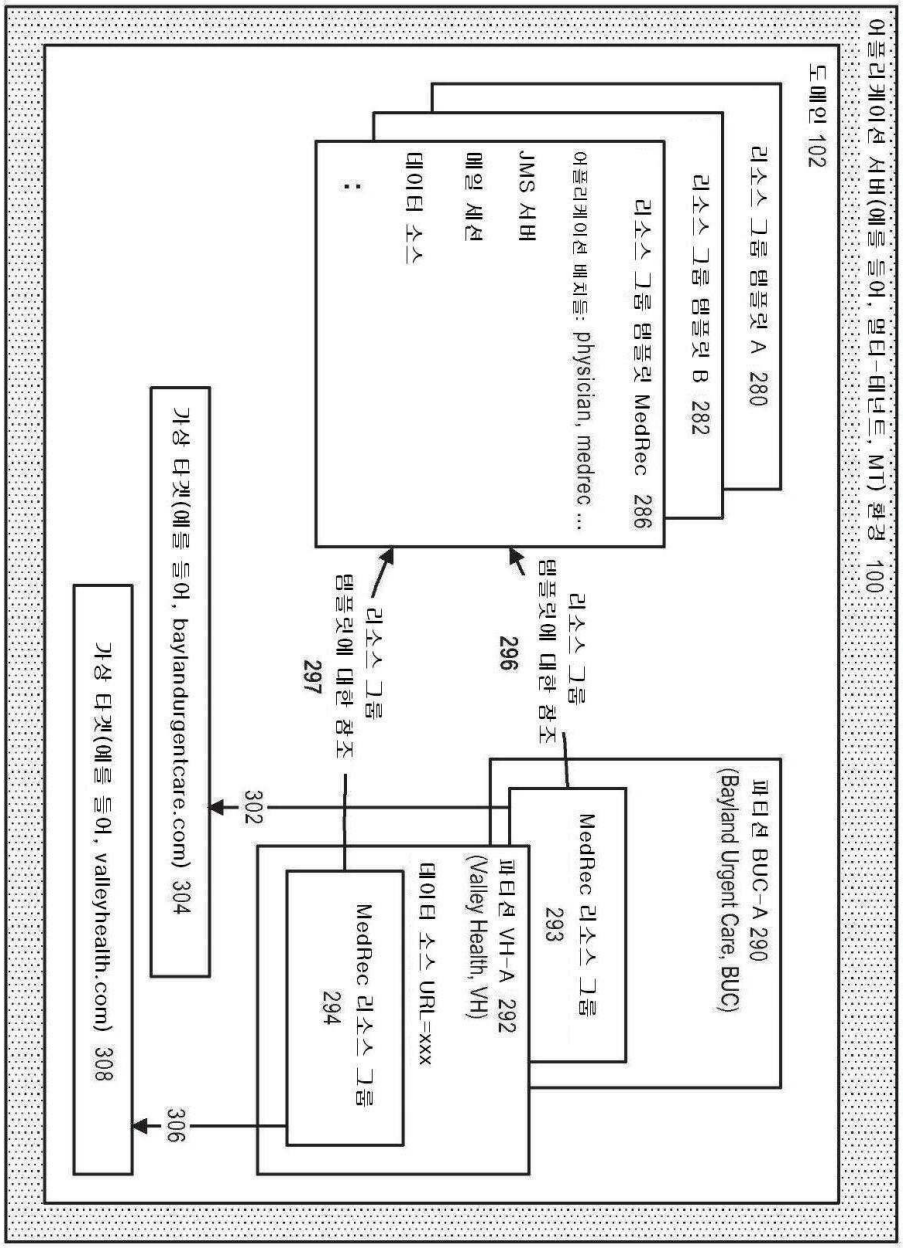
도면2


어플리케이션 서버(예를 들어, 멀티-테넌트, MT) 환경 100
도메인 102
파티션 202
리소스 그룹 205
리소스 그룹 템플릿 참조 206
PDB 정보 208
가상 타겟 정보 207
가상 타겟 정의 239
가상 타겟 240
컨테이너 데이터베이스 (CDB) 236
PDB 238
PDB 구성 232
파티션-특정 정보 230
리소스 그룹 템플릿에 대한 참조 220
리소스 그룹 템플릿 210
어플리케이션 A 211
어플리케이션 B 212
리소스 (예를 들어, JMS 서버) 213
리소스 (예를 들어, SAF 에이전트) 215
리소스 (예를 들어, 메일 세션) 216
리소스 (예를 들어, JDBC) 217

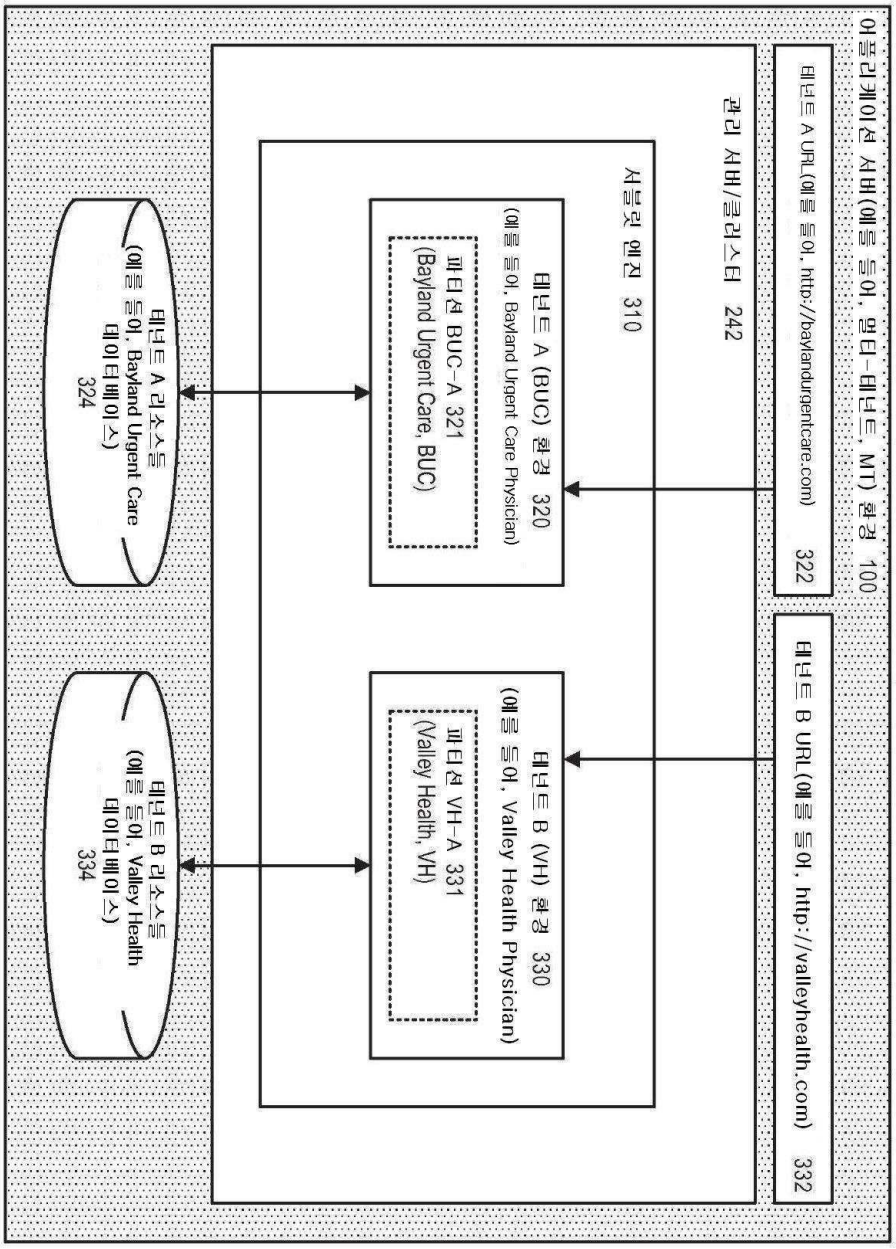
도면3


어플리케이션 서버(예를 들어, 멀티-테넌트, MT) 환경 100
웹 티어 244
관리 서버/클러스터 242
가상 타겟 A 258
가상 타겟 B 268
파티션 A 250
파티션 B 260
리소스 그룹 251
리소스 그룹 261
어플리케이션 A1 252
어플리케이션 A2 254
리소스 A1 256 (예를 들어, JMS A)
리소스 A2 257 (예를 들어, 데이터소스 A)
어플리케이션 B1 262
어플리케이션 B2 264
리소스 B1 266 (예를 들어, JMS B)
리소스 B2 267 (예를 들어, 데이터소스 B)
PDB A 259
PDB B 269
컨테이너 데이터베이스 (CDB) 243

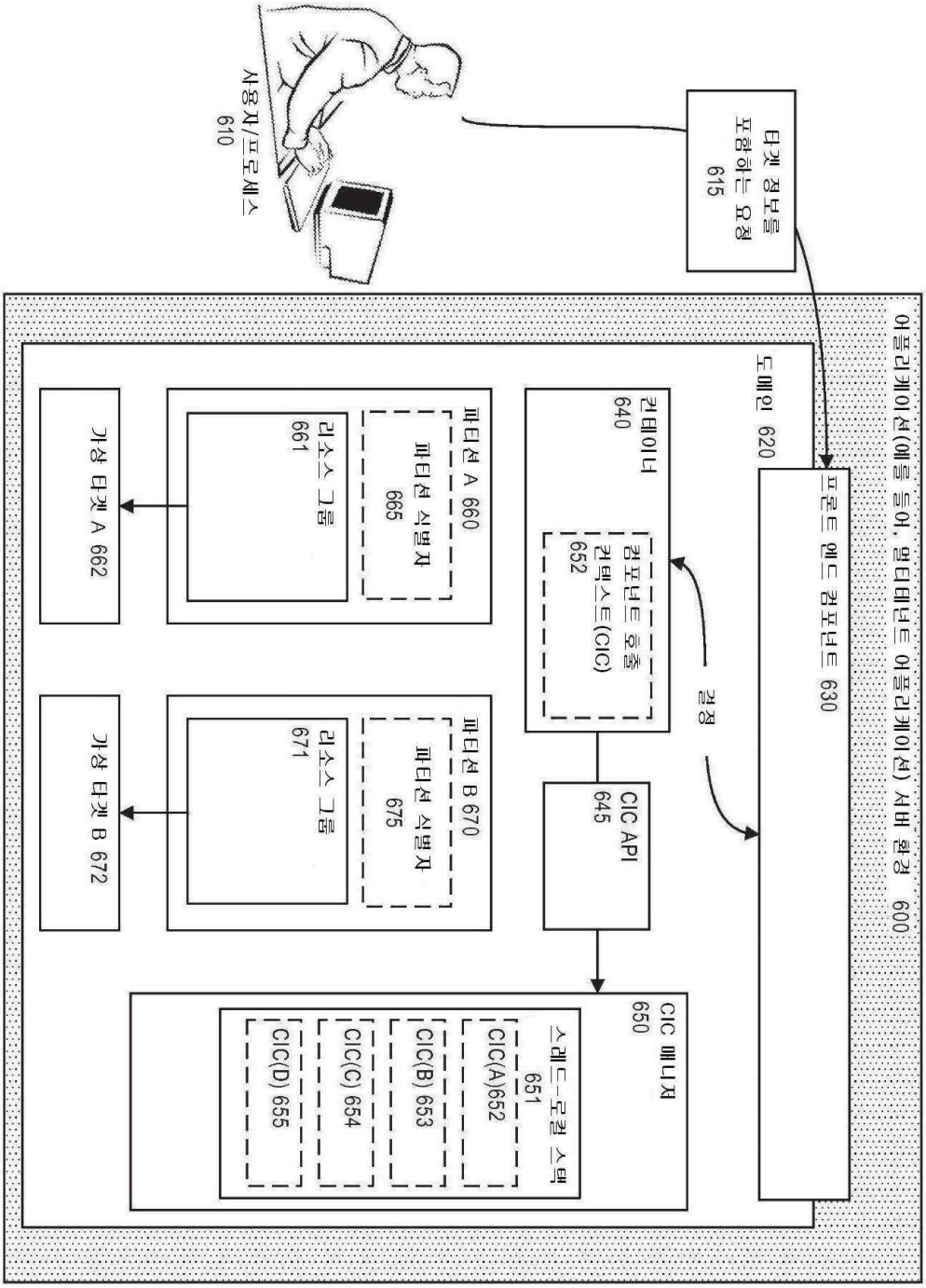
도면4


어플리케이션 서버(예를 들어, 멀티-테넌트, MT) 환경 100
도메인 102
리소스 그룹 템플릿 A 280
리소스 그룹 템플릿 B 282
리소스 그룹 템플릿 MedRec 286
어플리케이션 배치들: physician, medrec ...
JMS 서버
메일 세션
데이터 소스
:
리소스 그룹 템플릿에 대한 참조 296
리소스 그룹 템플릿에 대한 참조 297
파티션 BUC-A 290
(Bayland Urgent Care, BUC)
MedRec 리소스 그룹 293
파티션 VH-A 292
(Valley Health, VH)
데이터 소스 URL=xxx
MedRec 리소스 그룹 294
302
306
가상 타겟(예를 들어, baylandurgentcare.com) 304
가상 타겟(예를 들어, valleyhealth.com) 308

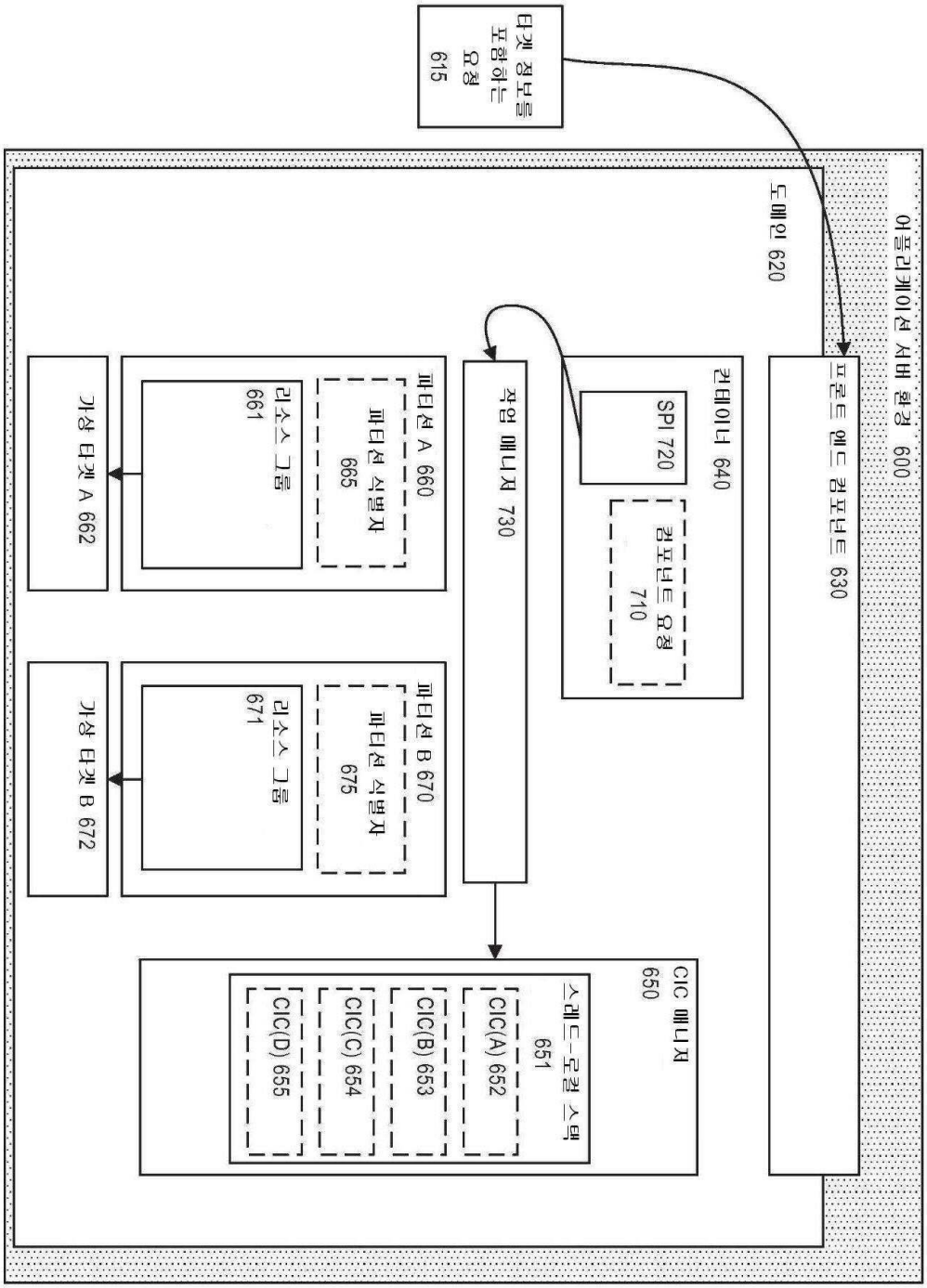
도면5


어플리케이션 서버(예를 들어, 알티-테넌트 MT) 환경 100
테넌트 A URL(예를 들어, http://baylandurgentcare.com) 322
테넌트 B URL(예를 들어, http://valleyhealth.com) 332
관리 서버/클러스터 242
서블릿 엔진 310
테넌트 A (BUC) 환경 320
(예를 들어, Bayland Urgent Care Physician)
파티션 BUC-A 321
(Bayland Urgent Care, BUC)
테넌트 B (VH) 환경 330
(예를 들어, Valley Health Physician)
파티션 VH-A 331
(Valley Health, VH)
테넌트 A 리소스들
(예를 들어, Bayland Urgent Care 데이터베이스)
324
테넌트 B 리소스들
(예를 들어, Valley Health 데이터베이스)
334

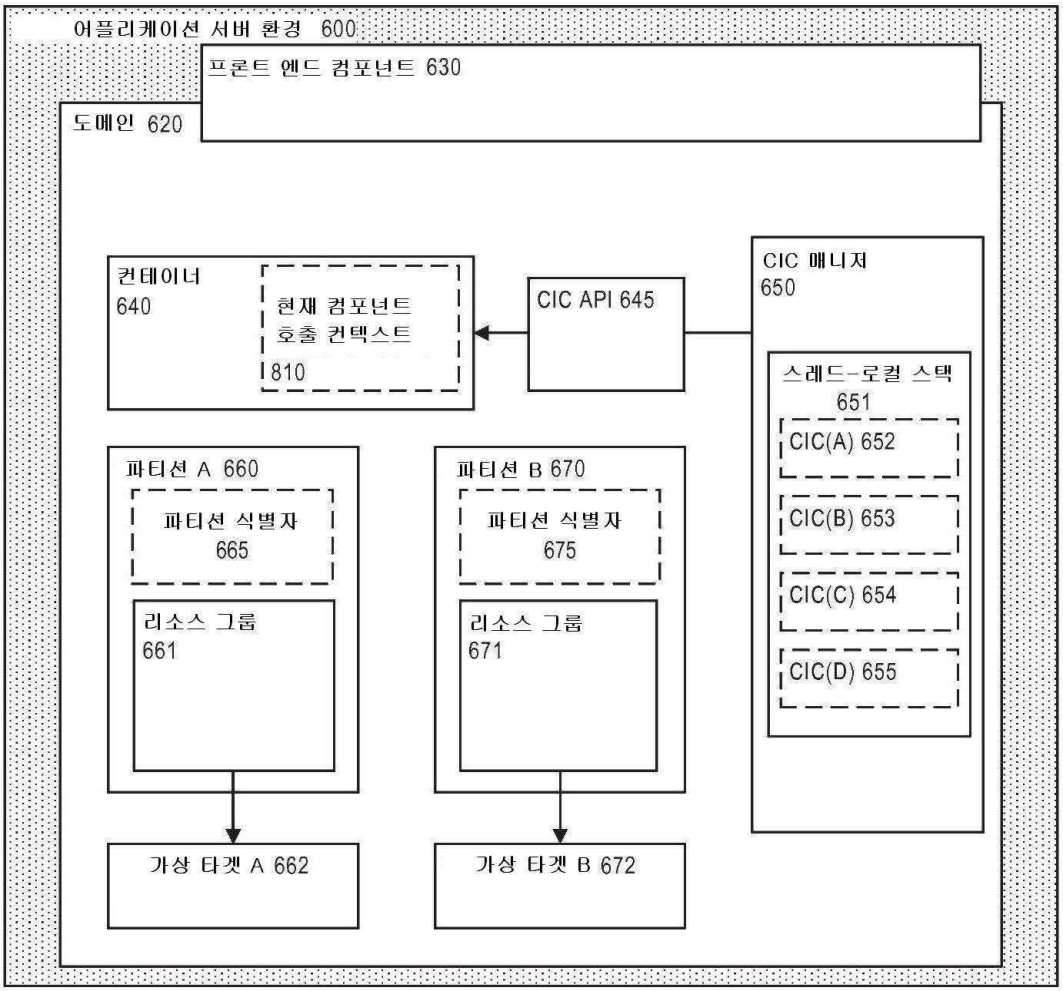
도면6


사용자/프로세스
610
타겟 정보를
포함하는 요청
615
어플리케이션(예를 들어, 멀티테넌트 어플리케이션) 서버 환경 600
도메인 620
프론트 엔드 컴포넌트 630
컨테이너
640
컴포넌트 호출
컨텍스트(CIC)
652
CIC API
645
CIC 매니저
650
스레드-로컬 스택
651
CIC(A)652
CIC(B) 653
CIC(C) 654
CIC(D) 655
파티션 A 660
파티션 식별자
665
리소스 그룹
661
가상 타겟 A 662
파티션 B 670
파티션 식별자
675
리소스 그룹
671
가상 타겟 B 672
결정

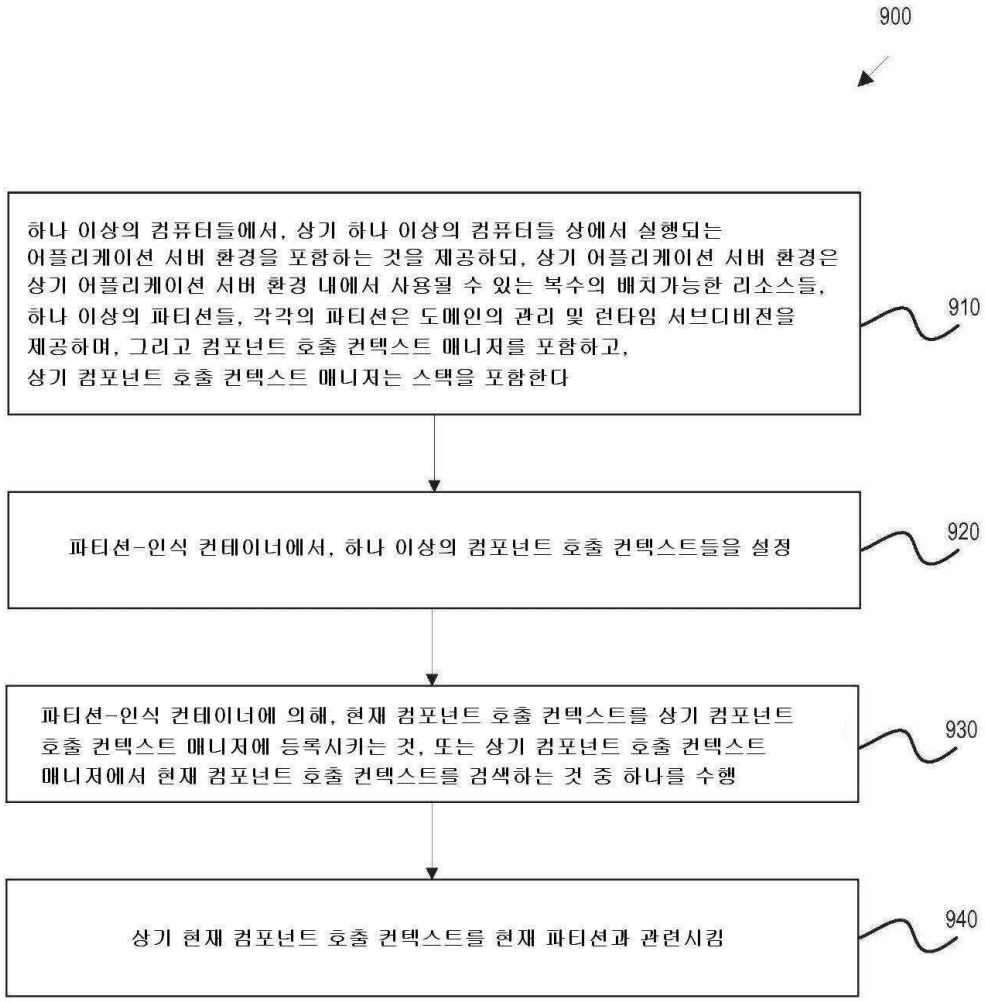
도면7


타겟 정보를 포함하는 요청 615
어플리케이션 서버 환경 600
도메인 620
프론트 엔드 컴포넌트 630
컨테이너 640
SPI 720
컴포넌트 요청 710
작업 매니저 730
CIC 매니저 650
스레드-로컬 스택 651
CIC(A) 652
CIC(B) 653
CIC(C) 654
CIC(D) 655
파티션 A 660
파티션 식별자 665
리소스 그룹 661
가상 타겟 A 662
파티션 B 670
파티션 식별자 675
리소스 그룹 671
가상 타겟 B 672

도면8


어플리케이션 서버 환경 600
프론트 엔드 컴포넌트 630
도메인 620
컨테이너
640
현재 컴포넌트
호출 컨텍스트
810
CIC API 645
CIC 매니저
650
스레드-로컬 스택
651
CIC(A) 652
CIC(B) 653
CIC(C) 654
CIC(D) 655
파티션 A 660
파티션 식별자
665
리소스 그룹
661
파티션 B 670
파티션 식별자
675
리소스 그룹
671
가상 타겟 A 662
가상 타겟 B 672

도면9


900
하나 이상의 컴퓨터들에서, 상기 하나 이상의 컴퓨터들 상에서 실행되는 어플리케이션 서버 환경을 포함하는 것을 제공하되, 상기 어플리케이션 서버 환경은 상기 어플리케이션 서버 환경 내에서 사용될 수 있는 복수의 배치가능한 리소스들, 하나 이상의 파티션들, 각각의 파티션은 도메인의 관리 및 런타임 서브디비전을 제공하며, 그리고 컴포넌트 호출 컨텍스트 매니저를 포함하고, 상기 컴포넌트 호출 컨텍스트 매니저는 스택을 포함한다
910
파티션-인식 컨테이너에서, 하나 이상의 컴포넌트 호출 컨텍스트들을 설정
920
파티션-인식 컨테이너에 의해, 현재 컴포넌트 호출 컨텍스트를 상기 컴포넌트 호출 컨텍스트 매니저에 등록시키는 것, 또는 상기 컴포넌트 호출 컨텍스트 매니저에서 현재 컴포넌트 호출 컨텍스트를 검색하는 것 중 하나를 수행
930
상기 현재 컴포넌트 호출 컨텍스트를 현재 파티션과 관련시킴
940