空間OS

@空気を読む家

2019年10月7日

先端IT活用推進コンソーシアム
ビジネスAR研究部会
日本総合システム株式会社　中川雅三

空間OS

- 身の回りのコンピュータ（エージェント）を相互連携させるプラットフォーム

実現したいこと1

- たとえば屋内での熱中症防止
  - 空間OS　　：「窓をあけませんか？」
    外気温が低く晴れており…
  - ありがちな解決策
    - 電機メーカー：　エアコンを自動的につけます。
      窓は開けないでね。
    - 住宅メーカー：　窓を自動的にあけます。
      窓のまわりには物を置かないでね。

様々な情報から
総合的判断

自社製品だけで
解決しようとする

実現したいこと2

- 一生使うシステム
  - ・人の老いや技術の変化・進歩へ対応する
  - ・機械を入れ替えても存在し続ける

  - 40歳　エアコン自動制御
  - 60歳「窓を開けませんか」
  - 80歳　家事ロボット:「窓をあけますね」

年月をかけて作り上げてゆくしくみ

窓をあけませんか

- 実現に必要なこと
  - 機器の連携
    - 状況＋手段＋（知識orルール）
  - 人との連携
    - 家庭や社会の一部としてシステムをデザイン
  - 過去・未来との連携
    - データを中心にデザイン
  - プラットフォーム化・インフラ化
    - その場所でのビジネス
      「窓をあけませんか？」という動作のスポンサーは誰？

世の中とも連携

永続化できるのはデジタルデータだけだから

稼げるようにすることが最大の課題

現状

- 連携できない
  - 「エッジ」同士
    - 「クラウド」の介在が必要
  - 異なるメーカー
    - 囲い込みから脱却できず…
  - 異なる業界
    - 異なる分野
    - 異なる規模
    - 新興ビジネス

    } 規格を作れない、間に合わない
  - 異なる世代・時代
    - 製品・事業・企業・業界の寿命は人生より短い

空間OSの目標

- フィールド(現場)での情報共有
  - 共通のことばでデータ即時共有
    - プロトコルからオントロジーへ
  - 受け継ぐデータ
    - デジタルデータはハードウエアの寿命を超えられる

フィールド:
「エッジ」より主体的

- フィールドとクラウドのすみ分け
  - プライバシー保護
  - 即応
  - 広域災害・障害耐性
  - データ品質の実現
  - データ量の縮減

フィールドの役割: クラウドにはできないこと

- フィールド基盤のインフラ化
  - 「規格」を作る方法の規格化
  - 異業種連携・新興ビジネス

稼げるようにすれば
様々な産業が興る

空間OSの実装

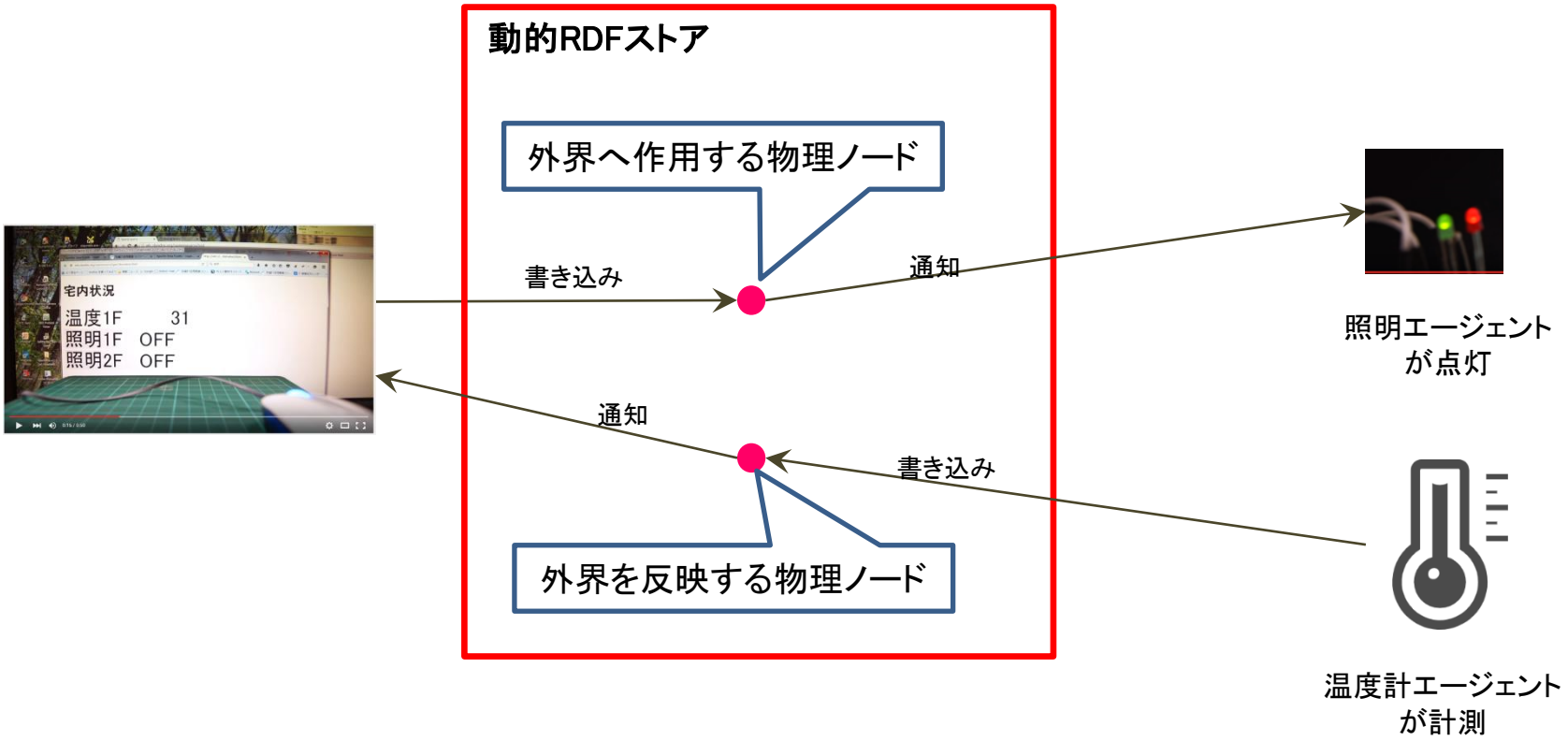
- アーキテクチャ：RDFストア＋拡張機能＋規格制定基盤
  - RDFストア
    - グラフデータベース → 任意の構造データの共有
    - RDF → データ表現の統一＋名前空間の分離
    - HTTP → プロトコルの統一

Linked Open Dataで使われている

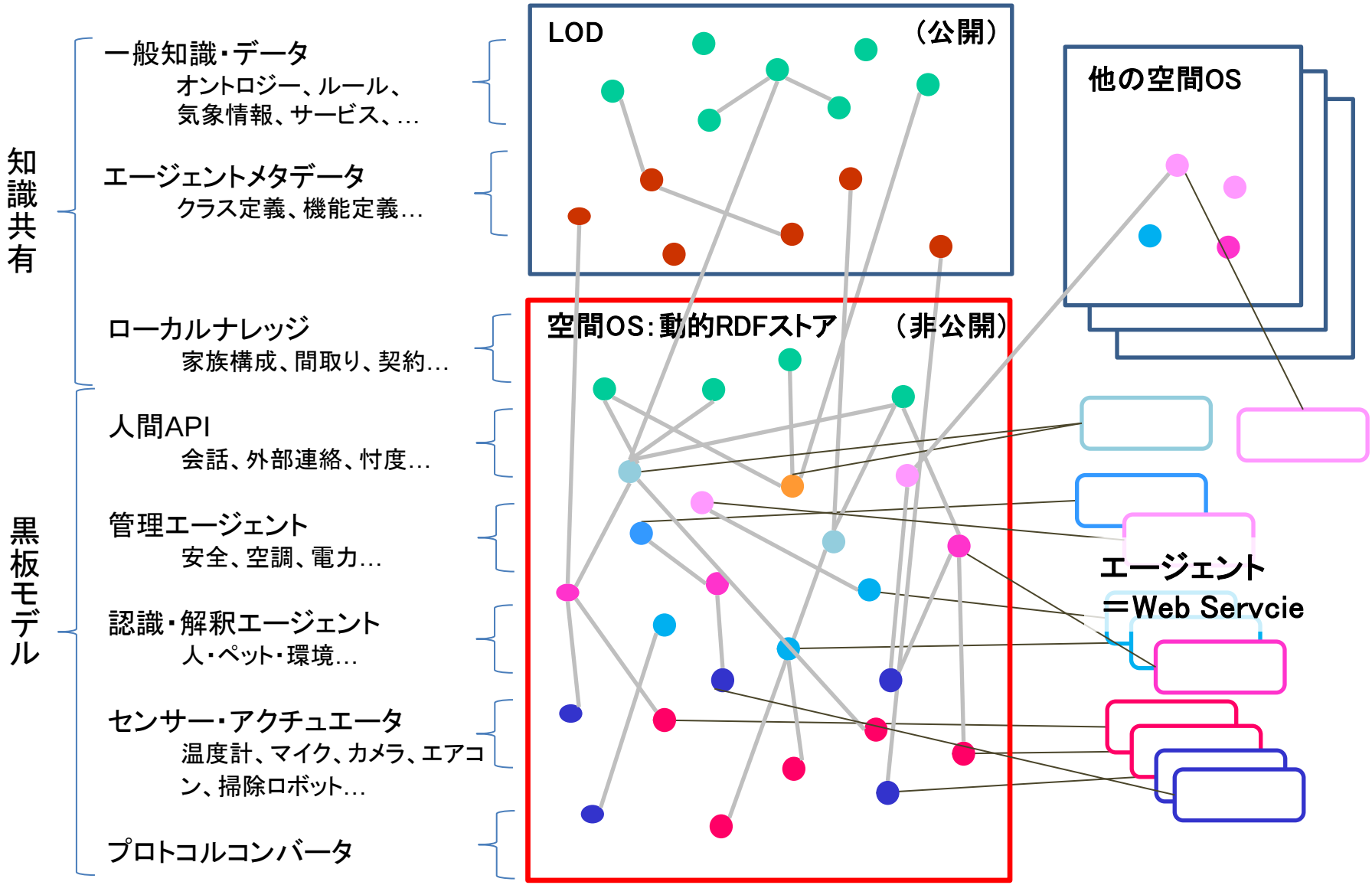
  - 拡張機能
    - 物理ノード：現実世界の反映
      - REST的：それぞれがサービスポータル
      - WebSocket：リアルタイム＋データプッシュ
      - セキュリティ：データごとのアクセス認可
  - 規格制定基盤
    - OWL＋上位の構造 → 機械可読な規格
    - 隔離された名前空間 → 複数規格の共存

物理ノード

外界とつながる論理的な実体
- リアルタイム通知
- 時系列記録

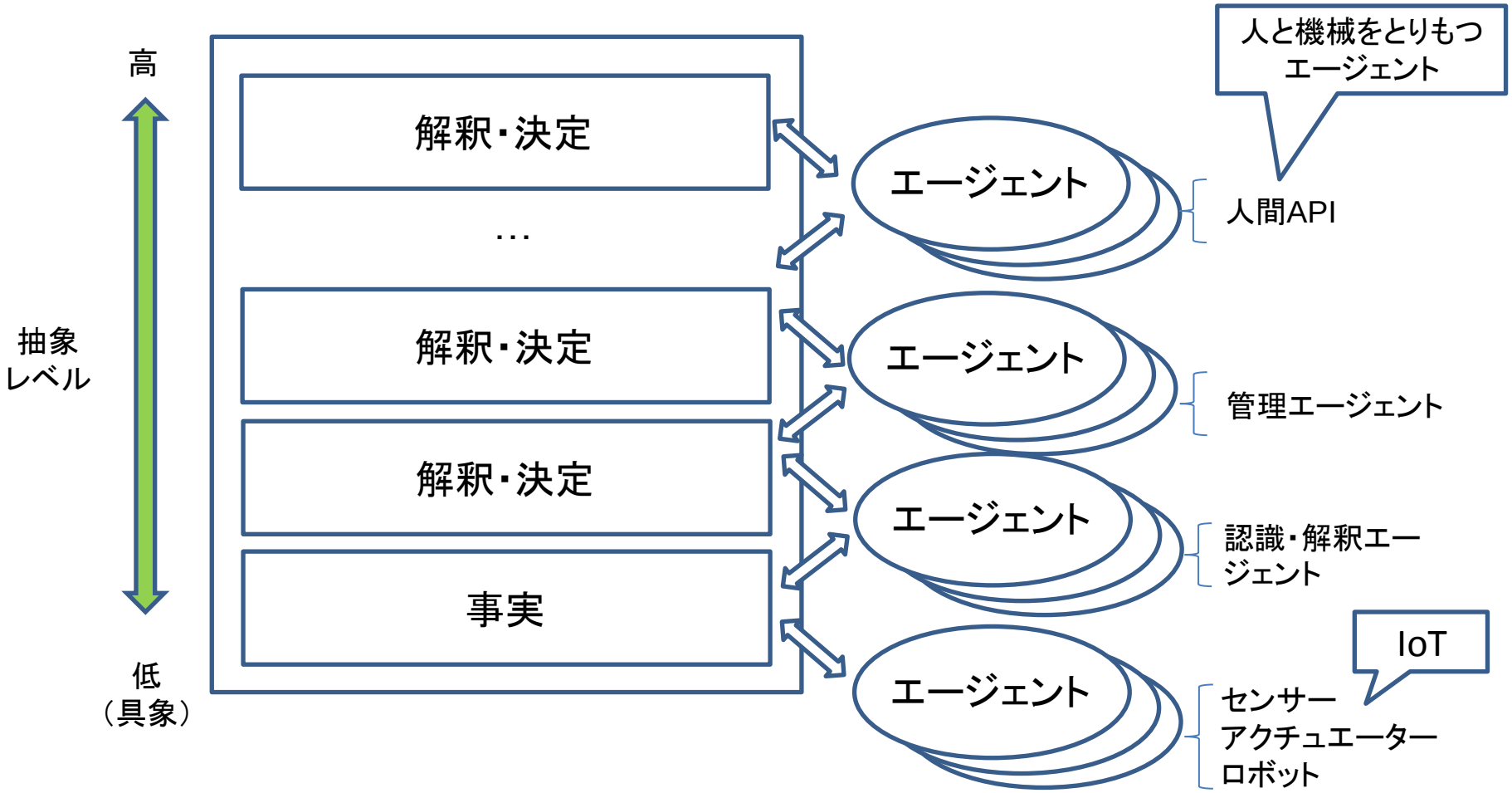


「家」での空間OS



プログラミングモデル

動的RDFストア　＝　マルチレイヤー黒板（レイヤー境界は曖昧）



空間OSの機能

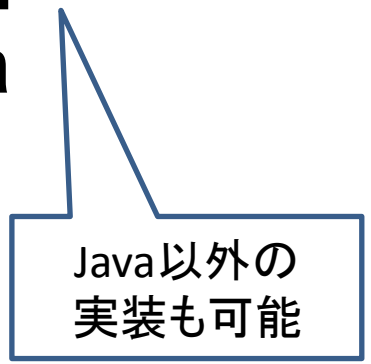
- 空間OSのユーザはエージェント
  - 様々なセンサー、ロボット(エアコン、掃除機)…
  - クラウド上のサービス、他の空間OS
  - 人間

- エージェント基盤としての空間OS
  - アーキテクチャの統一　　黒板モデル
  - プロトコルの統一　　HTTP+WebSocket
  - データフォーマットの統一　RDF

- 規格制定基盤としての空間OS
  - 規格の機械語
    → 規格のコンパイラを作りたい

空間OS 2019新機能

- ツール
  データ構造をプログラミング言語で表現
  - コンパイラ　　：Java → OWL+RDF
  - 逆コンパイラ　：OWL+RDF → Java

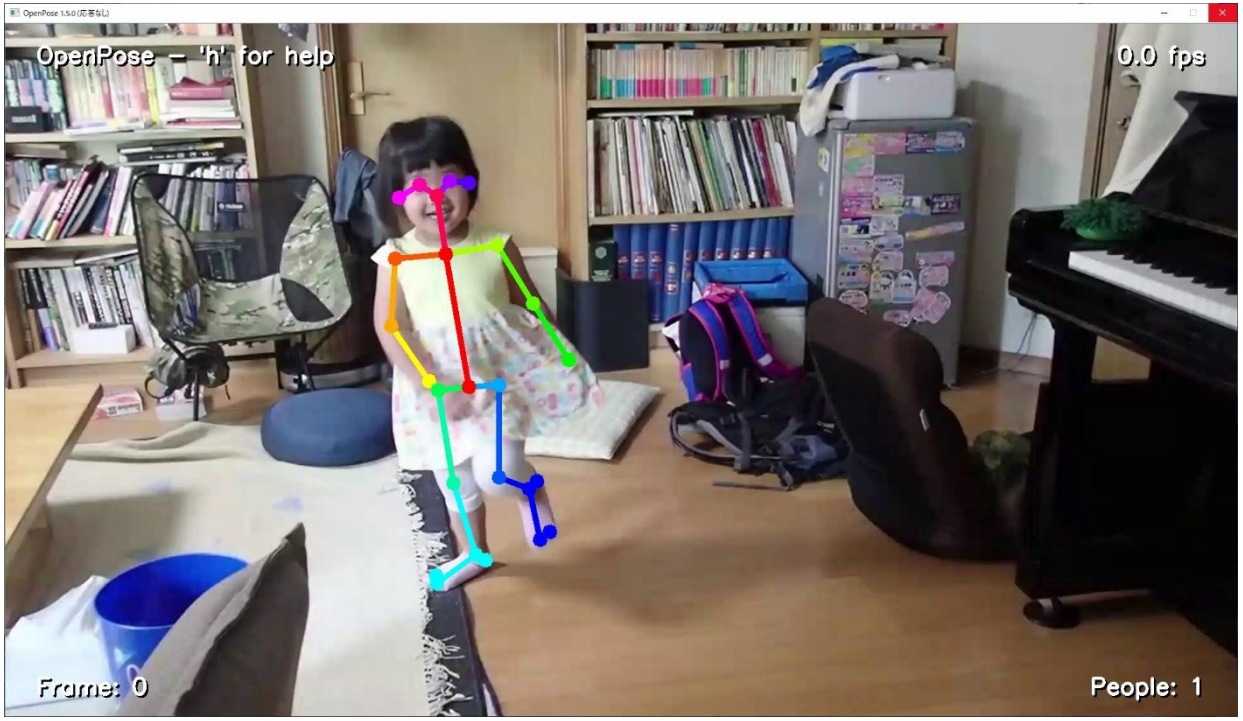


- 物理ノードの構造化
  - 入れ子クラス構造
  - メッセージキュー

- インスタンスの生成管理
  - クラス定義からのインスタンス生成

使用例

- 姿勢骨格データの取り込みとリアルタイム中継



データ例

- 1フレーム分（毎秒10～フレーム）で発生

{"persons":[{"bodyParts":[{"part":0,"pos":{"x":0.380,"y":0.773},"score":0.687},{"part":1,"pos":{"x":0.368,"y":0.790},"score":0.346},{"part":2,"pos":{"x":0.336,"y":0.802},"score":0.712},{"part":3,"pos":{"x":0.391,"y":0.908},"score":0.782},{"part":4,"pos":{"x":0.442,"y":0.974},"score":0.820},{"part":5,"pos":{"x":0.430,"y":0.794},"score":0.325},{"part":6,"pos":{"x":0.479,"y":0.830},"score":0.123},{"part":7,"pos":{"x":0.460,"y":0.818},"score":0.225},{"part":8,"pos":{"x":0.180,"y":0.835},"score":0.253},{"part":9,"pos":{"x":0.182,"y":0.871},"score":0.233},{"part":10,"pos":{"x":0.134,"y":1.023},"score":0.252},{"part":11,"pos":{"x":0.097,"y":0.994},"score":0.129},{"part":12,"pos":{"x":0.166,"y":0.806},"score":0.188},{"part":14,"pos":{"x":0.097,"y":0.990},"score":0.175},{"part":15,"pos":{"x":0.371,"y":0.757},"score":0.621},{"part":16,"pos":{"x":0.391,"y":0.749},"score":0.640},{"part":17,"pos":{"x":0.354,"y":0.761},"score":0.482},{"part":18,"pos":{"x":0.412,"y":0.745},"score":0.322},{"part":22,"pos":{"x":0.072,"y":0.953},"score":0.152},{"part":23,"pos":{"x":0.074,"y":0.953},"score":0.157},{"part":24,"pos":{"x":0.086,"y":0.974},"score":0.113}]}],"fos_warrant":"OpenPose-1.5.0-CPU"}

Javaで構造を記述

```
package com.example.fos.samples.pose2;
…
@FOSobject
@Data
public class Frame {
        private List<Person> persons;

        @FOSobject
        @Data
        public static class Person {
                private List<BodyPart> bodyParts;
        }
        @FOSobject
        @Data
        public static class BodyPart {
                private String id;
                private float score;
                private Position2d pos;
        }
        @FOSobject
        @Data
        public static class Position2d {
                private float x;
                private float y;
        }
}
```

Javaを選んだのはJavaを使っていたから他の言語でも可能

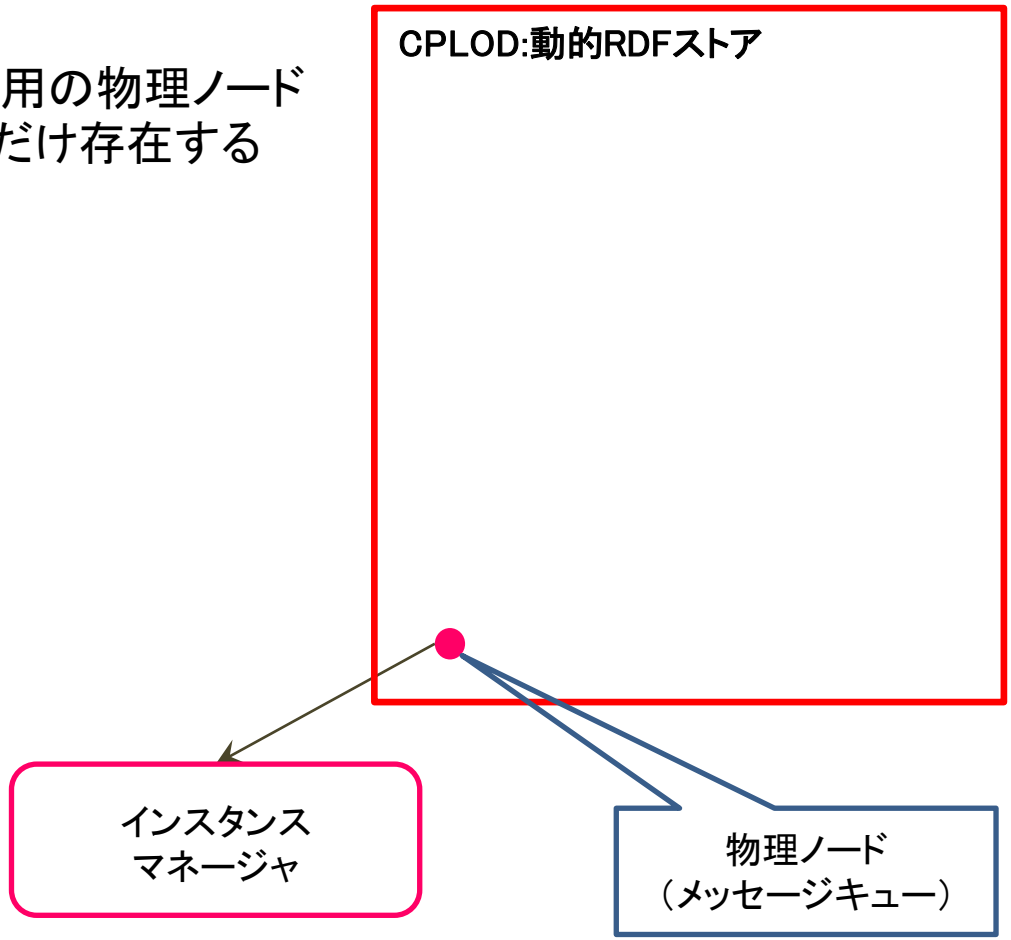
OWL-RDF表現へコンパイル

```
.....

@prefix owl:      <http://www.w3.org/2002/07/owl#> .
@prefix fos:     <http://bizar.aitc.jp/ns/fos/0.1/> .

fos:Class         rdfs:subClassOfowl:Class          .

@prefix :         <http://pose2.samples.fos.example.com/Frame#>                    .

<http://pose2.samples.fos.example.com/Frame>
                  rdf:type          fos:Class;
                  .
:persons
                  rdf:type          rdf:Property;
                  rdfs:domain       <http://pose2.samples.fos.example.com/Frame>;
                  rdfs:range        _:Seq_Person;
                  .
_:Seq_Person
                  rdfs:subClassOf rdf:Seq;
                  rdfs:subClassOf _:Restrict_Seq_Person;
                  .
_:Restrict_Seq_Person
                  rdf:type          owl:Restriction;
                  owl:onProperty  rdfs:ContainerMembershipProperty;
                  owl:allValuesFrom                 <http://pose2.samples.fos.example.com/Frame/Person>;
                  .

@prefix :         <http://pose2.samples.fos.example.com/Frame/Person#>          .

<http://pose2.samples.fos.example.com/Frame/Person>
                  rdf:type          fos:Class;

....
```

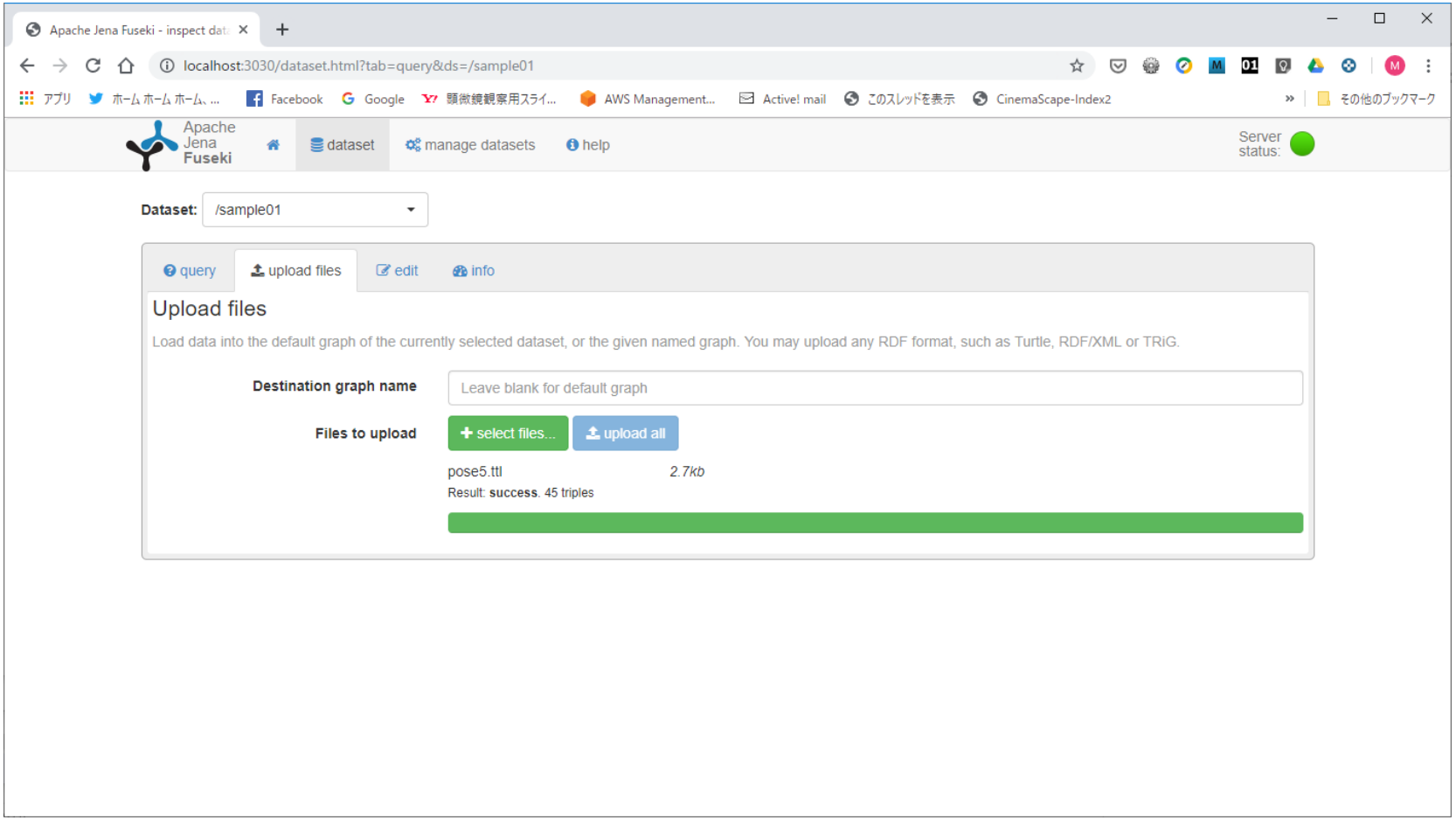
現在は
データ構造のみ変換

拡張RDFストアの初期状態

システム用の物理ノード
が1個だけ存在する

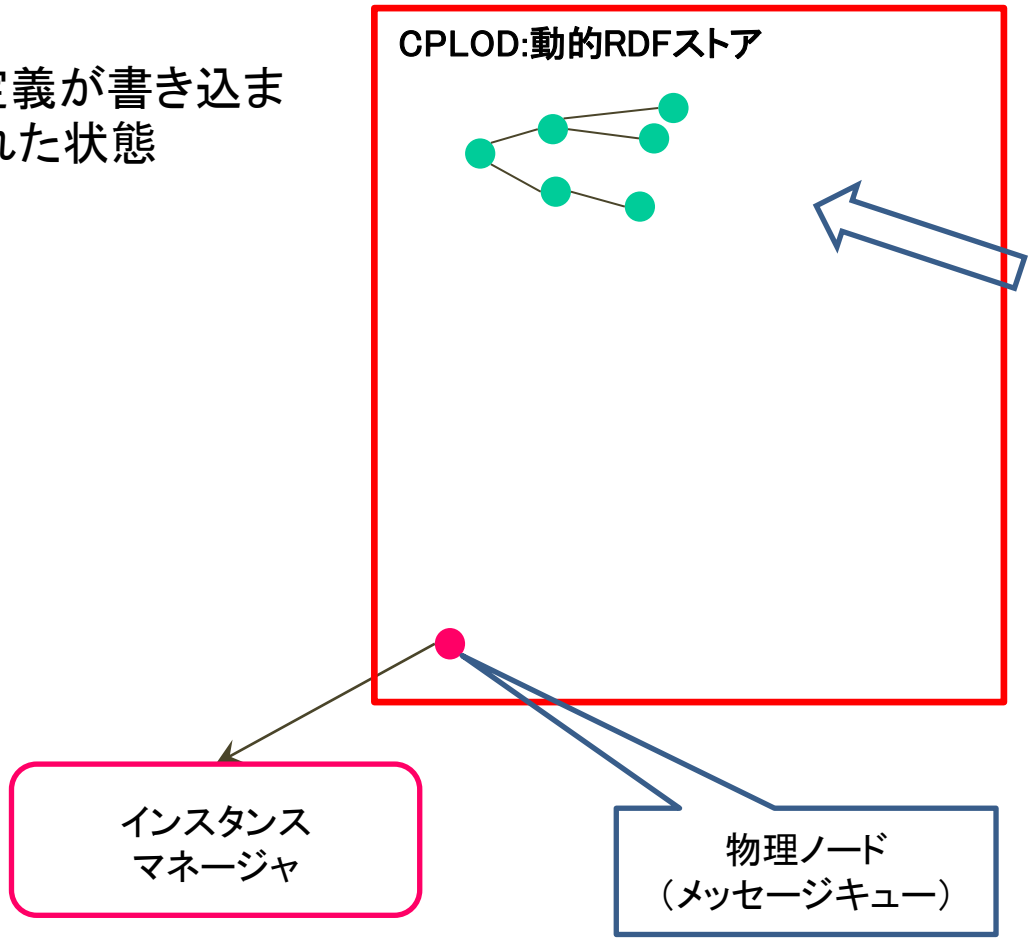


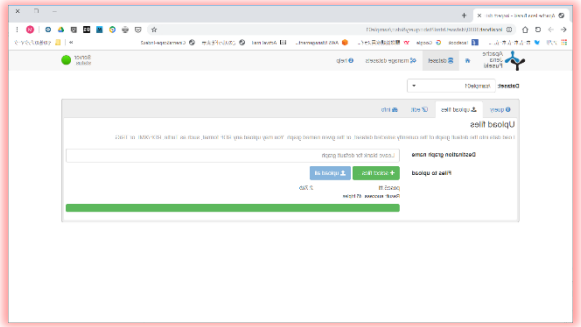
クラス定義のアップロード



クラス定義アップロード

クラス定義が書き込まれた状態





デモでは
Fusekiコンソールから

インスタンスの生成

SPARQL（RDFストア操作言語）で
空間OSのメッセージキューへリクエスト書き込み

```
PREFIX :        <http://bizar.aitc.jp/ns/fos/0.1/local/label#>
PREFIX :        <http://bizar.aitc.jp/ns/fos/0.1/local/label#>
PREFIX fos:        <http://bizar.aitc.jp/ns/fos/0.1/>
PREFIX moyo:       <http://fos.bizar.aitc.jp/>
INSERT {
       ?system fos:agent :test .
       ?system fos:arg_type moyo:CreateNewInstance .
       ?system fos:arg '{class_name: "http://pose2.samples.fos.example.com/Frame", as_object:[["http://samples.fos.example.com/住宅#居間",
"http://samples.fos.example.com/住宅#設置"]], nick:"abc" }' .
}
where {
      ?system fos:tag fos:_system .
}
```

この例では、
インスタンスに「居間に設置」
というグラフをくっつけている

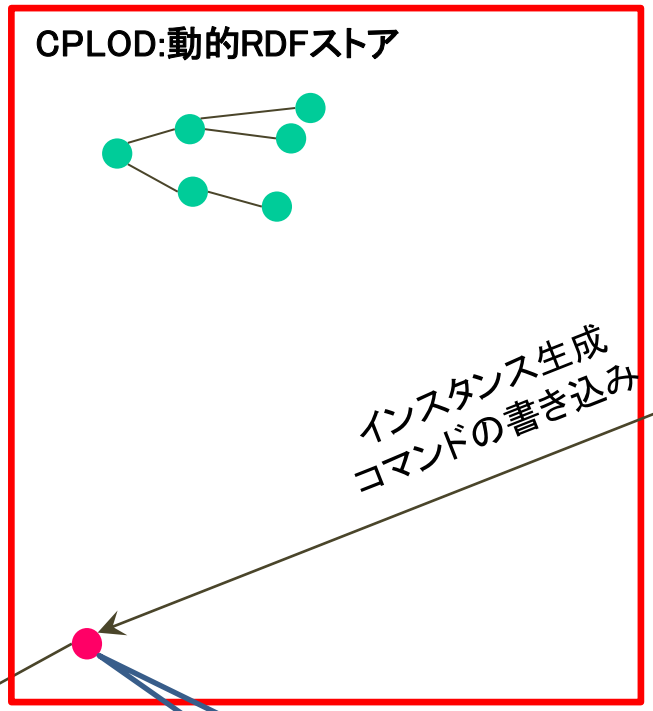
QUERY RESULTS

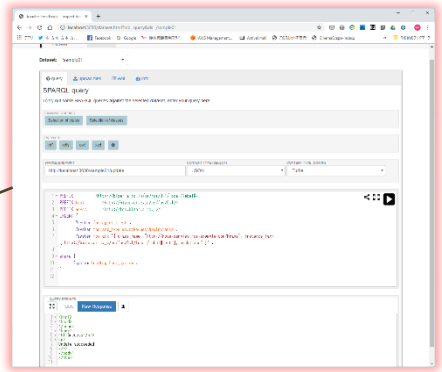
Table | Raw Response

```
<html>
<head>
</head>
<body>
<h1>Success</h1>
<p>
Update succeeded
</p>
</body>
</html>
```

生成コマンド送信

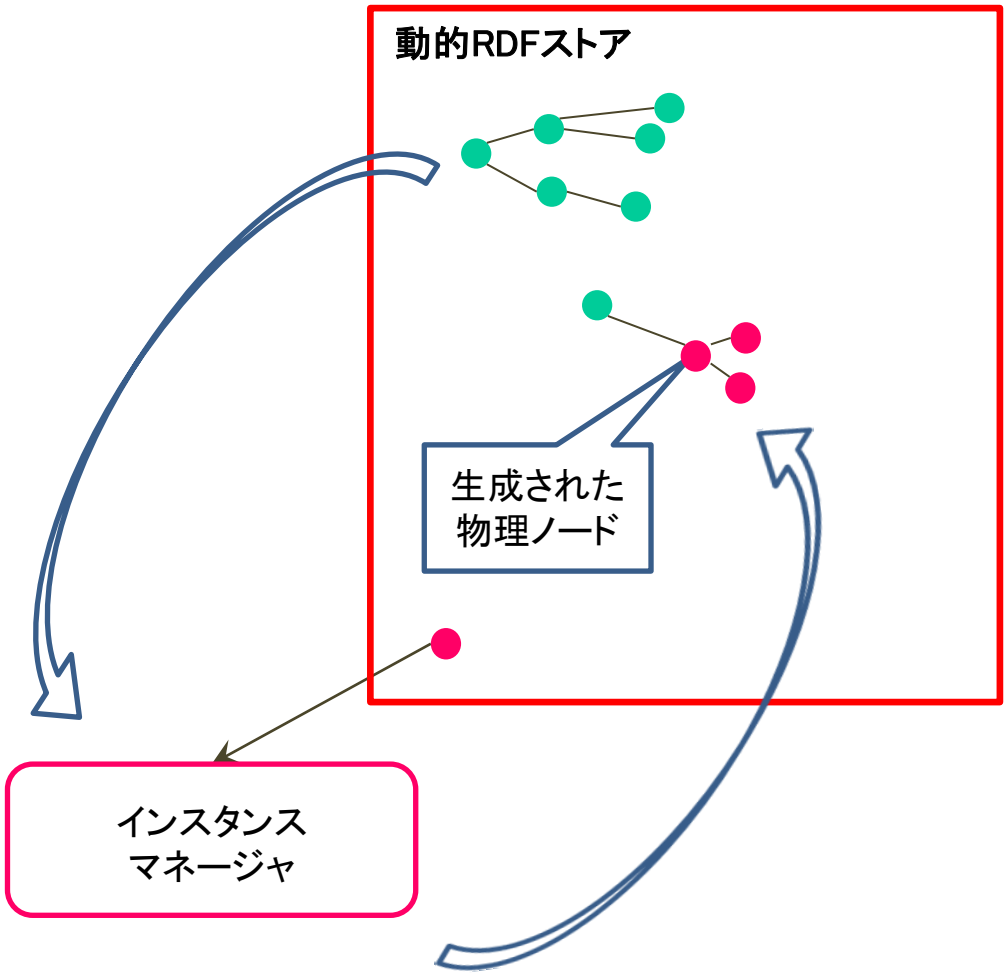
システム用の物理ノードへ
インスタンス生成リクエスト
を書き込む



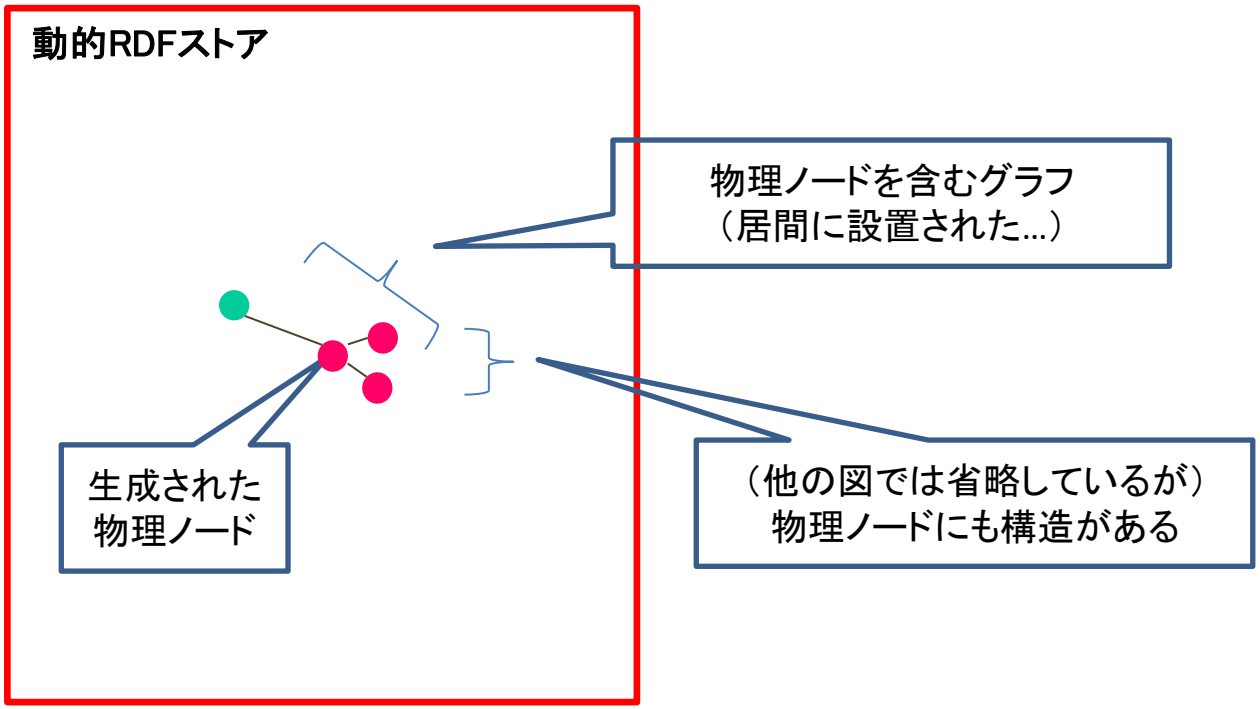


デモでは
Fusekiコンソールから

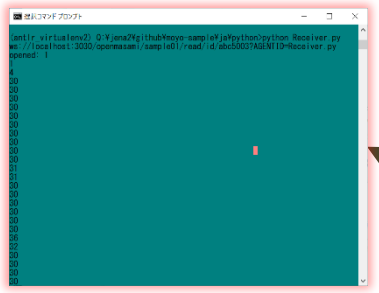
インスタンス生成



生成されたインスタンス



姿勢骨格表示の動作



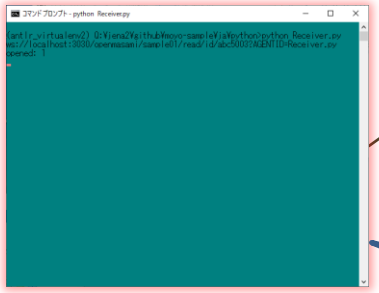
データ消費エージェント
（秒あたりフレーム数カウント）

CPLOD:動的RDFストア

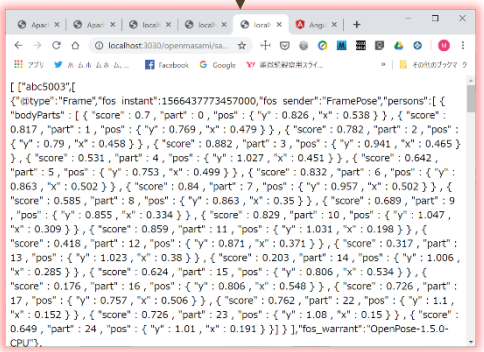
物理ノード

表示エージェント
（ブラウザで実行）

時系列ストア



姿勢骨格データ生成
エージェント



このデモでは生成済データの送信
（実用では画像認識エージェント）

性能

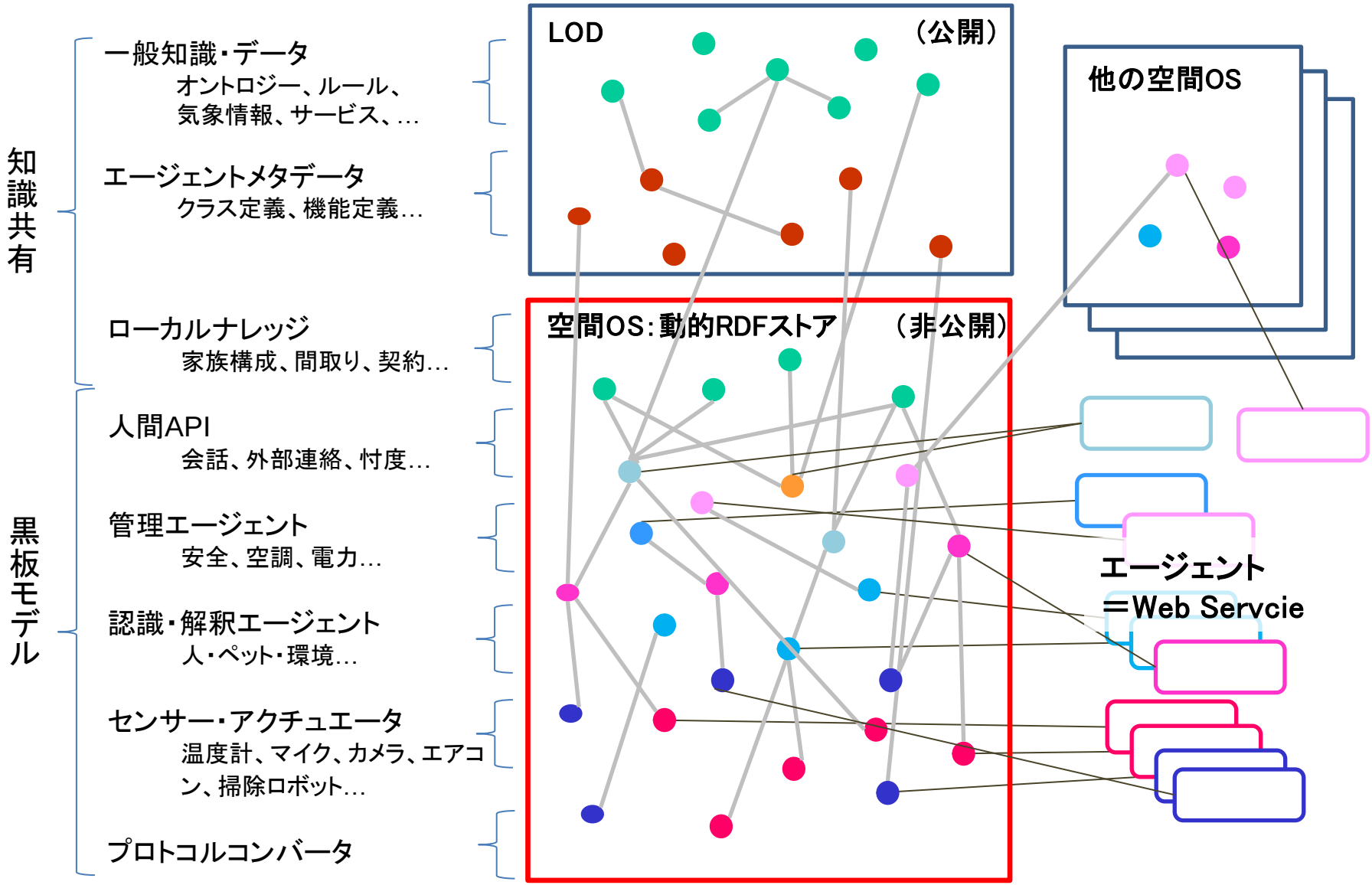
- 100チャネル同時中継程度を目標
  家族8名常時4台のカメラ追跡でも余裕(最大処理能力の半分以下)
  - 10フレーム/s
  - 1～2kB/フレーム
  - JSONフォーマット
    - 受信時にパーズしてValidate
    - 履歴はJSONでテキストファイルとして記録
    - バイナリ転送対応も可能
- 現状実測値:送受信各1チャネルで1200フレーム/s
  - COREi7ノート
  - 空間OSと送受信プログラムを同一PC上で

目標クリア
してそう

ざっくりとまとめ

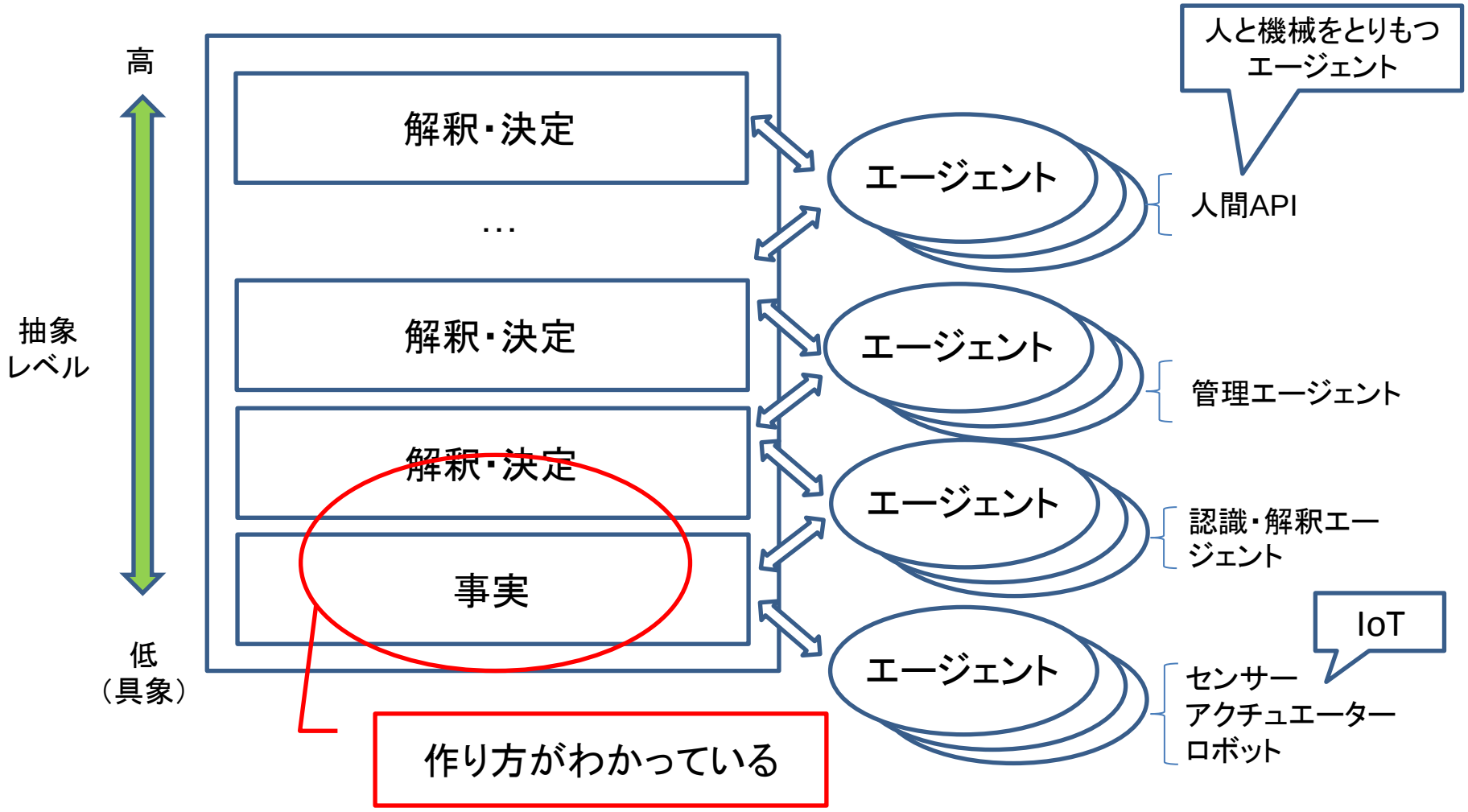
- プログラミング言語でデータ構造を表現
  - コンパイル結果（メタデータ）を空間OS（RDFストア）へアップ
- 空間OSに依頼してインスタンス（物理ノード）生成
  - 物理ノード　＝　WebSocketつきのRESTサービス
    - サブスクライバへ即時中継
    - JSONテキストで人間にも可読
    - 全データを時系列記録
- 空間OSには様々な物理ノードとメタデータが共存
  - 例：　住人Aさんを追跡して、在室中の部屋の温度を監視
    - 室温が28度を超え、外気温が25度以下、窓がある部屋で、天気が晴れ、60歳のAさん今日は運動不足
      → 「窓を開けませんか」

「家」での空間OS



プログラミングモデル

動的RDFストア　＝　マルチレイヤー黒板（レイヤー境界は曖昧）



世の中では…

- データ構造のOWL-RDF表現の利用拡大
  - ISO-8000関連で広がる？
    - 「データ品質」の定義と測定に利用

- HL7-FHIR(医療情報交換規格)
  - OWL-RDFによるリソース定義を公開
  - 空間OS逆コンパイラをテストしてみた
    - クラスやデータ型の表現方法が若干異なることに対応
    - 9万トリプル弱から約900個のクラスを生成

バージョン2020

- アルゴリズムもRDFへコンパイルしたい

  用途：
  - ソフトウエアの永続化　→　論理的な環境すべてをデータとして永続化
    - 使用する時代のプログラミング言語へ変換して実行
    - 人間にも可読にする　→　メンテナンス可能
  - RDFストアのストアドプロシージャ
    - モバイルエージェント
    - 規格・バージョン間コンバータ
  - IT規格の機械実行可能な定義
    - 規格上のアルゴリズム記述をRDF化　→　実行プログラムへ自動変換して検証
    - 規格へのツール同梱
      - サンプルデータ生成器
      - データ検証器
      - シミュレータ
      - コンバータ

AITC
先端IT活用推進コンソーシアム

http://aitc.jp

https://www.facebook.com/aitc.jp

ハルミン
AITC非公式イメージキャラクター