空間 OS によるエージェントのデータ共有と相互運用

Agent data sharing and interoperability by Space-OS

中川雅三 [1,2]

Masami Nakagawa[1,2]

[1] 先端 IT 活用推進コンソーシアム　ビジネス AR 研究部会
[1] *Advanced IT Consortium* to Evaluate, Apply and Drive Committee on Business AR

Abstract: 空間 OS は RDF の表現力と仕様の安定性を利用することで、コンピューター応用機器メーカー間の規格の相違を埋め、人の一生や世代を超えた時間レンジでの相互運用を目指すプラットフォームである。第 39 回 SWO 研究会では、実世界のモデルをリアルタイムで扱えるようにした動的な RDF ストアとしての空間 OS を提案した[1]。本発表では、空間 OS の上にマルチエージェントシステムを構築するための機能の詳細と課題について発表する。また、空間 OS の機能を応用することで、異なるアーキテクチャ間で、データ利用だけでなく、API 制御を含めての相互運用を実現する手法についても考察する。空間 OS は、エージェントの WEB サービス化、黒板モデルによる連係、包摂アーキテクチャ、WebSocket による即時通知、エージェント間交信言語の統一、Toulmin モデルによる行為の説明などのアーキテクチャをサポートする。

はじめに

身の回りのあらゆるコンピューター —情報家電、HEMS、ロボット、クラウドサービスなど— を連携させ、数十年以上にわたって生活の場を支え続けるインフラストラクチャとするアーキテクチャとして、空間 OS を提案[1]した。

本発表では、試作中の空間 OS へ新たに追加した機能と、空間 OS が目指すマルチエージェントシステムの構成方法について述べる。

空間 OS は、エージェント群を連携させるための機能を追加した LOD（Linked Open Data）である。CPLOD（Cyber Physical LOD）と呼ぶ動的 RDF ストアと、コンパイラを含むツールで構成する。

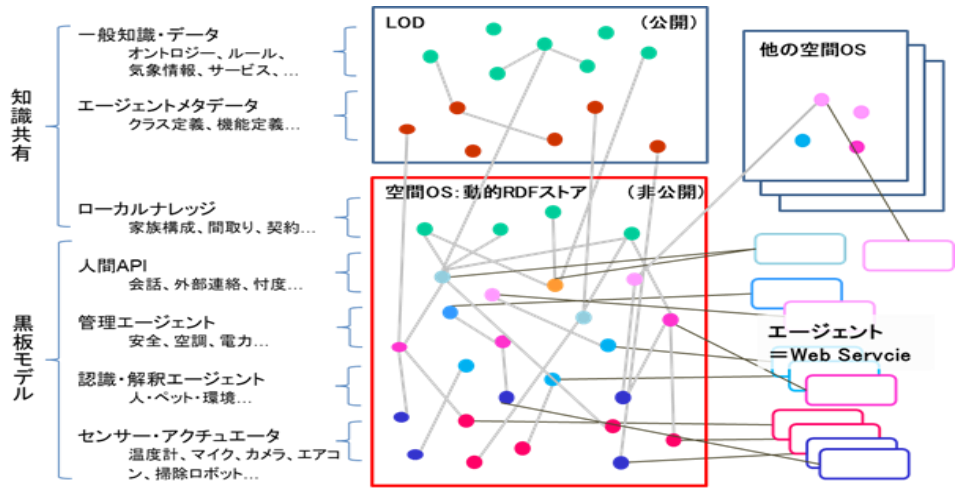

図 1. 空間 OS を介したエージェント連携

[2] 日本総合システム株式会社ユニファイド・テクノロジーグループ（Nippon Sogo Systems, Inc.）

動的 RDF ストアを通して、空間の中のエージェント群が、データやサービスをメタデータと一緒に公開し、エージェント同士でこれらを共有して連携動作する。

エージェントは物理ノードとよぶノードを通してリアルタイムにデータを授受する。外界にある LOD や空間 OS ともリンクし、動作に必要な知識を得たり、サービスを利用したりする。

CPLOD は、黒板モデルや包摂アーキテクチャによる実装をサポートする。

空間 OS に LOD を採用したのはつぎの特性を持つからである。

1. HTTP、RDF、SPARQL によるアクセス・通信方法・語彙表現の統一
   簡単・確実に実装・接続できる。RDF は十分に単純であるため、将来にわたって仕様が保たれることを期待できる。

2. 名前空間の分離
   他の規格との衝突を防ぎながら、独自の規格を作ることができ、バージョンの異なる規格を分離できる。また複数の規格のインタフェースへ同時に接続することもできる。

3. データ構造のグラフ表現
   任意のデータ構造を表現・実装できる。

4. URL による外部参照
   世界中の空間 OS を連携させ、その空間だけでは実現できないサービスやサイバースペースとの連携を実現できる。

住宅への適用

本発表では、空間 OS の適用例として住宅の制御を挙げて説明する。

空間 OS が目指す動作は、たとえばつぎのようなものである。

ユースケース A:「外が涼しそうなので、窓を開けませんか」と家が住人に提案する。

空間 OS が目指すのは単なる自動化ではなく、現場のニーズに合ったサービスの実現である。これまではサプライヤの都合でサービスが提供されているように見える。エアコンメーカーのソリューションに「窓を開ける」はなく、住宅メーカーのソリューションは「窓開けを自動化する」になりがちであろう。空間 OS は、その空間にあるすべての事物をデータベースに持ち、インタフェースを共通化することで、その場で利用可能なサービスを組み合わせてのソリューション提供を可能とするものである。これにより、現場の多様なニーズに対応したサービスをデザインできるようにすることを目指している。

ユースケース A が「住人への提案」になっていることには意味がある。窓を開けて良いかどうかの判断と操作を人に任せているのである。空間 OS では、人間を介在させることで妥当な解決を行えるようにする。

現在の技術で窓開け操作を自動化するとしたら、エキスパートシステムということになろう。エアコン、扇風機、窓開けといった空調手段から、条件によって適切なものを選ぶルールを列挙したものになる。しかし、風雨の被害防止、防犯、窓際に置いた物の落下防止といった「常識」すべてをルール化して窓開けの判断をするのは難しい。また、この動作のためだけに窓の自動開閉装置を設置するといったことも現実的ではないだろう。人の介在を仰ぐことでこれらの問題を回避できる。適切なコミュニケーションがとれるなら、UX の向上につなげることもできるだろう。

空間 OS は数十年以上の連続運用を目指す。現在若い住人が高齢になり身体が不自由になった未来において、このユースケースを、たとえば、

ユースケース B:「(常識を持った) AI が「窓を開けますね」といってロボットに窓を開けさせる」

へと変化できるようにしておく。

長い運用期間においてハードウエアは寿命やモデルチェンジで入れ替わる。ソフトウエアも同様である。しかし、データは永続的に存続させて使うことができる。空間 OS では、すべてのコンテキストをデータ化し、それをコピーして引き継ぐことで永続的に存在し、エージェントを置きかえたり包摂（後述）させたりすることで機能を更新する。

空間 OS の機能

前回の発表[1]では CPLOD における LOD からの 3 個の拡張を紹介した。

1. 物理ノード
   リアルタイムにプロパティ値を変化させる

ことができるノード。エージェントは WebSocket を通して物理ノードへ接続することで、リアルタイムに値の書き換えを行ったり変化通知を受信したりできる。

2. 履歴記録
   物理ノードの値変化と、RDF ストアのグラフ構造変化を時系列記録する。

3. アクセス制御
   RDF ストア全体ではなく、RDF ストア内のグラフの部分ごとにアクセス権限を制御する。(実現可能性を実証する実験は行ったが、まだ実装していない)

本発表までに、試作システムへ新たにつぎの機能を追加した。

4. 包摂機能つきメッセージキュー
   複数のエージェントからの物理ノードへの書き込みで競合が発生しても個別に対応できるようにするために、メッセージキューを内蔵する物理ノードを設けた。
   メッセージキューには包摂機能を加えた。包摂とは、あるエージェント A がサービスの受付に使っているメッセージキューを別のエージェント B が横取りし、エージェント A へのリクエストをエージェント B がすべて処理し、エージェント B のみがエージェント A へメッセージ送信できるようにする機能である。これによって包摂アーキテクチャ（後述）の実現や互換性を保ちながらのエージェントのバージョンアップを可能とする。

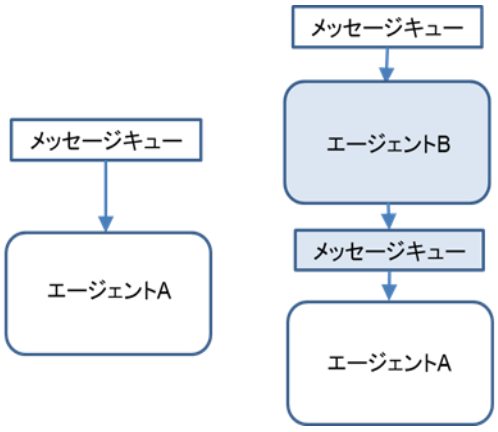

図 2. エージェント B による A の包摂

5. インスタンス管理
   RDF で表現されたクラス情報から物理ノードのインスタンスを生成できるようにした。インスタンスの生成や消滅を、同種のインスタンスを監視するエージェントへ通知することもできる。

6. 空間 OS コンパイラ
   CPLOD でのアプリケーション開発を容易にするために、空間 OS 用のコンパイラを作りはじめた。このコンパイラは、一般のプログラミング言語で書かれたクラスのソースコードを RDF スキーマと OWL によるプロパティ制約の表現へ変換する。
   現在の実装ではソースコードの言語として Java に対応する。コンパイル結果は空間 OS がインスタンス生成するための制約を守ったスキーマである。コンパイル結果から Java プログラムへ逆変換するトランスレータも試作した。
   コンパイル結果はプログラミング言語に依存しない。だから、Java 以外の言語からのコンパイラや、Java 以外の言語へ逆変換するトランスレータを作ることが可能である。
   現状では、データ構造のみを RDF へ変換する。

コンパイラが生成したスキーマを CPLOD へアップロードし、CPLOD へインスタンス生成を要求すると、CPLOD のインスタンス管理モジュールがクラス構造を照会して対応する構造をもつ物理ノードのインスタンスを生成する。エージェントは、このインスタンスの URL をポータルとする WEB サービスとして動作する。

マルチエージェントアーキテクチャ

空間 OS は、マルチエージェントシステムの実装基盤として以下のアーキテクチャを使えるようにしている。

1. WEB・REST 的サービス
2. 黒板モデル
3. 包摂アーキテクチャ
4. WebSocket による即時通知
5. 統一されたエージェント間交信言語
6. 行為の説明

WEB・REST 的サービス

センサー、アクチュエーター、ロボットなど、すべてのエージェントは、自らのサービスエンドポイントを CPLOD の物理ノードとして空間内へ公開する。物理ノードは URL をもち、CPLOD が公開するポートからこの URL へ HTTP 接続あるいは WebSocket 接続することで、サービス開設エージェントとサービス利用エージェントが物理ノードへの値の読み書きをすることができる。データは物理ノードのもつ RDF スキーマに相当する JSON テキストとして送受する。

これらの機能は、いわゆる WEB サービスや REST サービスに似ている。SPARQL での問い合わせによる、メタデータによる探索やインタフェース定義の取得は、WEB サービスにおける UDDI や WSDL にあたる。

エージェント群を仲介する OS として ROS(Robot Operation System)[2]があるが、これと比べて空間 OS における運用の特徴は、エージェント群の構成が事前に決められていないことである。エージェントは設置、入れ替え、廃止などを繰り返しながら長年月にわたって機能しつづける。空間 OS は、ローカルな構成や設定の情報、LOD に機器の仕様やエージェントのコードを置き、SPARQL によって検索できるようにしておくことで、これらの動作に必要な要求に応えることができる。

黒板モデル

上記のようなインタフェースで接続したエージェント群は、全体として RDF ストアを黒板とした黒板モデル[3]によるエージェント連携を行っているとみなすことができる。

エージェントはその場で利用できるすべての情報―コンテキスト―　を共有して動作する。パラメータのみに基づいてサービスを起動する一般的な API に比べて、多くの情報に基づいたふるまいが可能となる。

住宅を管理する空間 OS 上で動作するエージェント群は、つぎのような機能の階層を持つだろう。

1. 実世界＝事実と直結する低レベルエージェント。
   温度センサー、ドアロック、エアコン、映像からの人の姿勢抽出などを行う。いわゆる IoT 機器がこれらに相当する。
2. 実世界エージェントから報告される事実群を解釈し、データの統計操作や、人・ペット・ロボットなどの行動把握、状況を推測したりするエージェント。
   センサーやアクチュエーターのヘルスチェック、家の中での行動追跡、火災・防犯状況の認識などを行う。
3. 上記のエージェントからの情報をもとに、与えられたミッションを遂行するエージェント。
   見守り、空調、電力制御などを行う。
4. 人間 API
   人や社会との対話・連絡を行うエージェント。他のエージェントへ API を公開する。例えばあるエージェントが異常通報 API へメッセージを送信すると、適切な方法を選んでそれを必要とする人へ伝える。
   現在の試作ではまだ実装へ至ってはいないが、ショートメッセージの送信機能程度の最低限の実装から始めて、会話や忖度などの機能を持ったものへ進化させることになるだろう。完成した人間 API は、様々な情報から空間内の人間やペットなどのモデルを生成し、気分や健康などの状態を把握するような存在になるだろう。

抽象度の高い処理を行う上位のエージェントは、抽象度の低い下位のエージェントから得たデータを解釈し、さらに上位のエージェントに解釈を提供したり、下位のエージェントに命令を送ったりすることで、全体として高度な連携動作を生み出す。

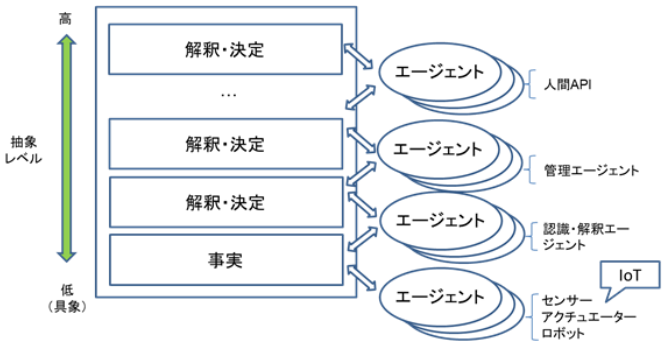

図 3. 黒板モデル

包摂アーキテクチャ

黒板モデルで述べた階層は主に役割の分担であっ

たが、空間 OS は、同系統の役割を持ちながら階層を分ける「包摂アーキテクチャ(Subsumption Architecture)」[4]もサポートする。

たとえば、配下の照明を制御する照明スイッチエージェントを上位の照明コントロールエージェントが包摂し、無人になったらスイッチの状態にかかわらず消灯するといった機能を作ることができる。

包摂アーキテクチャでは、上位のエージェントは下位のエージェントの機能をコントロールして、より高度な機能を実装する。また、クラウドとの通信が断たれるなど、上位のエージェントの機能が失われた場合には、下位のエージェントのみで動作するような実行形態へ縮退するような実装も可能である。

包摂アーキテクチャの階層が事前に設計できる場合には、下位のエージェントは上位のエージェントが利用するためのインタフェースを提供する。上位のエージェントは、下位のエージェントが提供するインタフェースを通して、下位エージェントを制御する。

下位エージェントが事前に上位エージェントに対するインタフェースを用意しないことがある。このときには、CPLOD メッセージキューのフック機能を使い、現行エージェントのインタフェースを上位エージェントが横取りすることで包摂を行う。

WebSocket による変化通知

物理ノードへの読み書きは、WebSocket による接続を通して高速に実行することができる。

データを利用するエージェントはデータ源の物理ノードへ WebSocket で接続してデータの発生を待つ。データ源となるエージェントは物理ノードの WebSocket へのメッセージとしてデータを送信する。CPLOD は、受信したデータのバリデーションを行い、正常であれば時系列記録すると同時に、そのデータを待つすべての WebSocket へコピーを送信する。

現状の CPLOD の実装は、動画ストリームから抽出された人や物体の振る舞い程度の情報を、並みのハードウエアで処理することを想定している。

動画ストリームなど、性能の面で CPLOD が直接扱えないデータについては、エージェントが CPLOD を介さずに授受するための外部サービスの URL やメタデータを提供することで、エージェント同士で直接授受する。

統一されたエージェント間交信言語

エージェントは空間 OS を介して RDF スキーマで定義されたメッセージ構造体を使って交信する。これは、低レベルではあるが、交信言語の統一といえる。メッセージ構造体のクラスやプロパティなどに関するオントロジーを整備することで、より上位の統一言語をつくることができるだろう。

上位層で知的エージェントが実装できるようになり、たとえば FIPA-ACL[5]のようなデータ表現が必要になった場合には、この基盤を使ってメッセージ構造体をそれらの要件に合わせた構造にし、アダプタエージェントによってプロトコルを合わせることも可能であろう。

動作の説明

複数のエージェントの相互運用では、システムの改良や発生した事象の原因解析のために、システムが行った行為を説明する機能が必須であると考える。

行為の説明とは、パフォーマンス低下や異常な動作などが発生したときに、その事象に至るエージェント群の動作について、動作の決定理由を説明することである。

空間 OS は、論証における Toulmin モデル[6][7]を援用し、「意思決定の正当性の主張」という形式で説明することを支援する。Toulmin モデルにおける「根拠（Data)」「結論（Claim)」「理由付け（Warrant)」の 3 要素に注目し、

エージェントが「入力＝根拠」から「理由付け」に基づいて「出力＝結論」を出した

を動作の説明の基本単位とする。

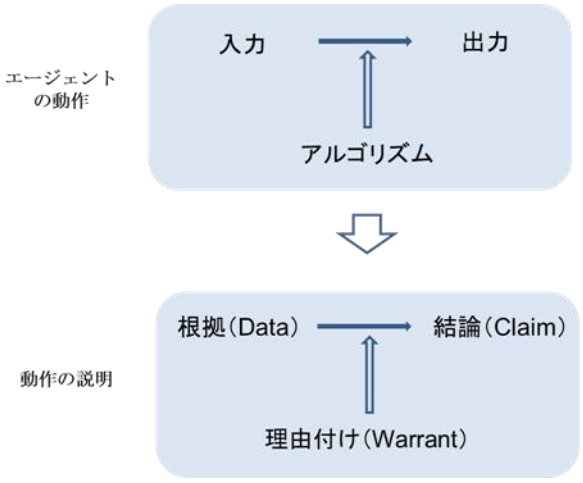

図 4. Toulmin モデルによる動作説明

たとえば、ユースケース A の動作の説明はつぎのようなものだろう。

根拠　　：室温が 30 度。外気温が 26 度。
理由付け：評価関数 X が手段「住人への窓開け依頼」を選択。手段候補と評価値のリストは…。
結論　　：人間 API へ窓開け依頼のメッセージを送信。

空間 OS は、動作の説明のために、エージェント出力の記録につぎの 3 要素を付加して記録する。

1. 時刻
2. エージェント ID（書き込みを実行した）
3. 理由付け（オプション）

時刻はエージェントへの入力を特定する。空間 OS ではすべてのデータ書き込みを記録するため、時刻を指定すればそのときの入力値を特定できる。

エージェント ID はエージェントを特定することで、決定の主体となったプログラムを特定する。

理由付けが含むべき内容とその表現形式は今後の研究課題であるが、空間 OS は任意の JSON データとして理由づけを記録できる。理想的にはアルゴリズムの動作経路の記述が望ましい。しかし少ないオーバヘッドでの実現は困難だろう。たとえばエージェントがルールエンジンであれば適用されたルールや評価値、結果を説明できないタイプの機械学習モデルによるものであれば判定に使った学習データのタイムスタンプなどが記録されるべきであろう。また、不特定の入力を扱うエージェントであれば、判断に使った入力のリストも理由付けに含まれるだろう。

理由付けがオプションであるのは、エージェントが理由付けの出力に対応しないケースがあるからである。温度センサーのような自明な機能をもつエージェントや、説明へ配慮していない既存のプログラムでは、理由付けは空欄となる。分析対象の問題の原因となったエージェントが特定され、そのエージェントが理由付けの報告に対応していないときには、必要に応じて当該エージェントのアルゴリズムを分析することになる。

空間 OS におけるプログラミング

住宅でのユースケース A を例として、空間 OS におけるエージェントのプログラミングの概要を説明する。

データ構造の定義

ユースケース A では、たとえば部屋の快適さを検出するために温湿度センサーエージェントが空間 OS へ計測値を報告する。

Java で記述した温湿度データ構造は、つぎのようなものになる。

```
class SensorValue {
  Optional<Float> celsius;
  Optional<Float> humidity;
}
```

この例では、欠測を表現するために計測値を Optional としている。空間 OS のコンパイラはこれをつぎのような RDF 表現へ変換する。

```
@prefix : <http://.../SensorValue#>      .
<http://....SensorValue>
        rdf:type    fos:Class;
        rdfs:subClassOf       _:SensorValue_celsius;
        rdfs:subClassOf       _:SensorValue_humidity;
        .
:celsius
        rdf:type    rdf:Property;
        rdfs:domain           <http://.../SensorValue>;
        rdfs:range xsd:float;
        .
_:SensorValue_celsius
        rdf:type    owl:Restriction;
        owl:onProperty        :celsius;
        owl:minCardinality  0;
        owl:maxCardinality  1;
        .
……
```

空間 OS のクラススキーマは RDF スキーマと OWL に準拠するようにしているが、微妙に異なる部分もあるため、クラスのタイプは rdfs:Class ではなく独自の名前空間の Class として区別できるようにしている。たとえば OWL のプロパティ制約で MinCardinality=0 はプロパティの不在を示すが、空間 OS では、プロパティが存在するが値が無い（null である）ことを示している。

インスタンスの生成

このコンパイル結果を CPLOD へアップロードしてインスタンス化を指示すると、温湿度センサーの

データ報告先となる物理ノードを CPLOD が生成する。

インスタンス化は、システムノードと呼ぶ物理ノードへリクエストを書き込むことで実行する。この例では、CreateNewInstance という CPLOD のコマンド名とパラメータの JSON 文字列を与えている。パラメータは、インスタンス化するクラス名と、インスタンスが組み込まれるグラフ構造を指定している。この例では、「<…居間> <…設置> <このインスタンス>」というトリプルでセンサーが居間に設置されることを表している。

```
PREFIX 住宅: ….
…
INSERT {
 ?system fos:arg_type fos:CreateNewInstance .
 ?system fos:arg '{
       “class_name”:"http://.../SensorValue",
       “as_object”:[[ ”住宅:居間”,”住宅:設置”,]]}' .
}
where {
        ?system fos:tag fos:_system .
}
```

CPLOD では、インスタンス生成をつぎの 3 通りの方法で実行できる。

1. SPARQL でのリクエスト（上記例）
2. システムノードの URL へのリクエストの POST
3. システムノードの URL へ接続した WebSocket へのリクエスト送信

エージェントの発見

この物理ノードの値を参照するエージェントは、SPARQL クエリで物理ノードを発見して値を取得する。

たとえば、「住人 X がいる部屋にある温湿度センサーの物理ノード」の URL を取得するクエリはつぎのようなものである。

```
SELECT { ?sensor }
WHERE {
        <uri://住人 P> 住宅:在室 ?room .
        ?room 住宅:設置 ?sensor .
        ?sensor a <http://....SensorValue> .
}
```

ここで取得した「?sensor」に格納されている IRI から、ある変換規則に従って URL を得ることができる。IRI を URL そのものにしていない理由はつぎのとおりである。

- IRI が示すサービスへアクセスする URL を変更できるようにしておきたい。
- 空間 OS サーバ内のデータを、サーバの URL から独立したものにしたい。

データの取得

エージェントは、前セクションのクエリで取得した IRI あるいは URL を使って、つぎの 3 通りのうちいずれかの方法で温湿度の値を取得する。

1. SPARQL クエリ
   （IRI を subject としてプロパティの値を参照する。）
2. URL への HTTP による GET
3. URL へ WebSocket で接続して連続読み出し

HTTP GET と WebSocket による読み出しでは、追加パラメータ指定をすることで、センサーが過去に記録した測定値の時系列情報を取得することもできる。

パフォーマンス

このセクションでここまでに紹介した機能は、試作実装で動作している。

WebSocket で JSON データを送受信する実験では、AWS EC2 の t2.micro インスタンスで、1KB のデータを毎秒 1000～2000 個中継することができた。このとき、受信した JSON データをデシリアイズしてからふたたびシリアライズし、履歴記録した後に送信している。

住宅での応用において、最もデータ量が多いのは画像センサーからの動画像であろう。前述のように、CPLOD は動画ストリームを直接中継する能力はないが、画像認識ソフトウエアが抽出したデータをリアルタイム中継できることを目標としている。たとえば人体ポーズ認識ソフトウエアが 1 コマの画像から抽出する 1 人の人物のポーズの特徴点情報が毎秒 10 コマで 1 コマ 1～2kB 程度として、空間 OS は人物の動きを 100 チャネル程度同時に中継・蓄積する能力がある。たとえば 10 部屋に 4 台ずつカメラがある家でも約半分の能力を消費するだけである。

上記のコンピューターで空間 OS が扱う情報の総

量は毎秒 1〜2MB 程度である。記録された JSON 情報は 1/10〜1/20 程度に zip 圧縮できるため、1 年分のデータは 4TB の記録メディアに収まる程度となる。

これらの結果から、廉価な PC 並みのコンピューターで 1 軒の住宅の管理が可能であろうと言える。

上位エージェント

これまでのセクションで、温度、人の所在などを CPLOD 内へ蓄積し共有する方式を述べた。

たとえば「人がいる」という情報と様々な情報を総合して「居間には X さんがいる」という情報を導くようなエージェントも同様の方法で生成したデータを共有する。

このセクションでは、ユースケース A を実現する層のエージェント　―まだ実装したことがない―が、素朴な実装であれば、現在の技術で実現可能であろうことを考察する。まず動作するものを作り、構成エージェントを置き換えながら理想的なシステムへ近づけて行くことが空間 OS の目指すアプリケーションのライフサイクルであり、最低限の実装が可能であることが重要だからである。

ユースケース A:「外が涼しそうなので、窓を開けませんか」と家が住人に提案する。

は、たとえばつぎの 3 段階の動作で実現できる。

1. 課題の発見
   「住人のいる部屋が快適でない高温だ」
2. 課題の解決策の探索と起動
   気温を下げる方法を見つけて、起動する
3. 解決の確認
   起動した方法が正常に機能したことを確認する。

最も単純な実装のひとつは、つぎのようなものである。

1. 温度が規定値を超えたら、課題「高温」発生と判定する。

2. 「高温」に対してあらかじめ列挙された、「機械が実行してよい解決策と、その評価関数（現在の状況をパラメータに持つ）」について評価値を計算し、評価が最大の解決策を起動する。
   ユースケース A では、「外気温が XX 度低かったときにその部屋にいる住人に依頼するメッセージ文」が解決策として列挙されている。エアコンを起動する、ブラインドを閉めるなどの解決策に対して、消費電力などから導かれる評価値が最大になることから A を解決策として選択する。
   このエージェントは人間 API へ、メッセージをその部屋に在室している住人へ伝えるように依頼する。

3. 指定された時間だけ待って、室温が下がり始めたら正常に機能したと判定する。

このレイヤのエージェントの実装では、「時間の経過」の扱いが難しいだろう。なにかを起動して結果が出る前に状況が変化するといったことに対しても安定した動作を行わなければならないからである。一般的な論理値の計算やファジィ論理におけるメンバシップ関数値の計算では、状態遷移に要する時間経過はゼロである。時間経過に伴って変化する内部状態を持つモデルを作る必要があるだろう。

人間 API

人間 API は、人や社会との対話・連絡を行う API を他のエージェントへ提供するエージェントである。全エージェントが使う機能であるため、OS として標準 API の提供が必要だと考えている。

初期の最低限の実装は、他のエージェントに託された自然言語メッセージを、対象の人物やサービス窓口へメールやショートメッセージで伝える程度のものとなる。

将来はつぎのような機能を持つようになるだろう。

1. 「エージェントの意図」を表すデータ構造を与えるとメッセージ表示や音声合成などの現在使える手段に応じた表現形式へ変換して伝達し、対話によって伝達を補強したり確認したりする。

2. 空間内の人間の行動予定や気分などのデータを把握し、行動を促すときや、緊急事態のときに、もっとも確実な対応方法を選択する。

3. 外部のサービスについて、どのくらい信頼できるか、品質はどの程度かといった情報を収集し、望ましい委託先を選択する。

包摂によるユースケース B への置き換え

将来、先に述べたユースケース B を実行できるエージェント B ができたときには、ユースケース A を

実行するエージェントAをエージェントBに包摂させることができる。

今後の機能拡張

メソッドのコンパイル

空間OSコンパイラに、メソッド（アルゴリズム）のコンパイル機能を追加しようとしている。メソッドの構文木からアルゴリズム記述とコメントを抽出し、RDFのグラフ（以後メソッドグラフと呼ぶ）へ変換する機能である。

この機能に対応して、メソッドグラフをプログラミング言語に変換するトランスレータを実装する。入出力がCPLODに限定されるので、コンパクトな実行時ライブラリで実装できるだろうと考えている。

変換先の言語は、コンパイル前の言語と異なる言語を選択できるように、メソッドグラフをプログラミング言語に中立なものとしたい。空間OSがプログラミングモデルとしてRDF上のクラスとインスタンスを採用しているため、オブジェクト指向言語でないプログラミング言語や、クロージャのような複雑なメモリ管理を含む構文のトランスレータは作りにくいかもしれない。

メソッドグラフの第一の目的は、実行プログラムの永続化である。数十年以上の連続稼働を目指す空間OSにおいて、プログラミング言語仕様の経年変化や、作成者の退場が、深刻な問題となる。メソッドグラフ化によって処理プログラムを言語独立とし、かつ人間にも可読とすることで作成者不在でもメンテナンスできるようにする。

規格の記述

空間OSのRDFで表現したスキーマによるデータ構造記述とメソッドグラフを、一般の情報処理に関する規格の記述の一部を機械実行可能とするために使うことができると考える。

空間OSコンパイラの出力はプログラミング言語独立でありながら一般のプログラミング言語のソースプログラムへ変換して実行可能であることを利用する。規格記述はプログラミング言語から独立していることが望ましいからである。

規格記述のうち、つぎの部分に応用することが可能であろう。

1. データ構造とアルゴリズムの記述
2. データのバリデータの作成
3. サンプルデータの生成
4. サービスモデルのスタブの作成

現在多くの規格は規格自体に2〜4を含まないが、これらを含めれば実装への参入ハードルを下げ、互換性を高めるために役立つものと考えている。

空間OSでは、空間OS自体の規格の記述に空間OSコンパイラのアウトプットを使いたいと考えている。

CPLOD内蔵エージェント

CPLODにメソッドグラフの実行機能を組み込むことで、エージェントをCPLODサーバ内で実行できるようになる。また、RDFストアへアクセスするライブラリを整備することで、リレーショナルデータベースにおけるストアドプロシージャに相当する機能を作ることもできる。

プロトコルスタックと組み合わせることで、アダプタエージェントや、既存APIと空間OSのエージェントインタフェースとのコンバーターを作ることができるだろう。

空間OS内にモデルを構築することで、内部状態をもち、リクエストや状態変化によってメソッドグラフを実行することができる。これにより、様々なAPIのシミュレートや変換が可能となると考える。

他の空間OSとの連携とセキュリティ

物理ノードのURLをインターネットへ公開すれば、空間OS同士を連携させることができる。プログラム内のデータ参照が他の処理系の中にあるオブジェクトを指すようなプログラミングモデルを実装できるだろう。

このような実装も可能となるようなセキュリティモデルを開発する必要がある。

おわりに

空間OSが取り込むアーキテクチャのいくつかは、かつてもてはやされ、しかし期待されたほどは使われなかったものである。今、いわゆるAIやIoTなどの技術を連携させるものとして、これらのアイデアを再評価する良いタイミングであろうと筆者は考えている。

本稿で適用例とした住宅を管理するエージェント群の実現には、ビジネス化の方法などのハードルによりまだ時間がかかりそうである。

空間 OS の実用化に適用可能な題材の提案などをいただければ幸いである。

謝辞

筆者に活動の場を与えている日本総合システム株式会社、空間 OS の構想と設計をともにしてきた先端 IT 活用推進コンソーシアムのビジネス AR 研究部会とコンテキストコンピューティング部会のメンバー、「空気を読む家」プロジェクトで空間 OS の実験実装を使っていただいている他の部会の方々に感謝いたします。

参考文献

[1] 中川雅三: LOD の物理世界拡張と空間 OS, 第 39 回 SWO 研究会 (2016)

[2] Morgan Quigley, et al.: ROS: an open-source Robot Operating System, ICRA workshop on Open-Source Software, 2009.

[3] Erman, L.D., F. Hayes-Roth, V.R. Lesser, and D.R. Reddy,:The Hearsay-II Speech Understanding System: IntegratingKnowledge to Resolve Uncertainty:, ACM Computing Surveys,Vol. 12, No. 2 (June, 1980).

[4] Brooks, R. A.: A robust layered control system for a mobile robot. IEEE Journal on Robotics and Automation, 2, 14-23.(1986)

[5] The Foundation for Intelligent Physical Agents (FIPA): Agent Communication Language Specifications:, http://www.fipa.org/repository/aclspecs.html

[6] Stephen E. Toulmin.: 議論の技法, 東京図書(2011)

[7] 福澤一吉.:議論のレッスン,NHK 出版(2002)