エネルギー科学研究科

エネルギー社会・環境科学専攻修士論文

題目：

オブジェクト指向に基づく仮想空間構築手法に関する研究

指導教官： 吉川 榮和　教授

氏名： 小牧 大輔

提出年月日： 平成13年2月7日(水)

論文要旨

題目： オブジェクト指向に基づく仮想空間構築手法に関する研究

吉川榮和研究室, 小牧 大輔

要旨：

近年のコンピュータの計算処理能力の飛躍的な発展と、インタフェース技術の発展により、従来より活発に利用されてきた、ゲームなどのエンターテイメントの分野だけでなく、様々な分野で VR の実用化に対する試みがなされている。VR を用いた訓練環境も応用分野の 1 つである。原子力発電プラント等の大規模エネルギー機器の保守作業や運転作業を体験できる環境を構築することで、安全・安価に多様な訓練を実施可能となり、訓練機会の増加、エネルギー利用技術の向上、エネルギー機器取扱上の安全性・信頼性の向上を期待できる。

しかし、従来の仮想空間の構築方法では、構築対象が複雑で大規模になったときに、仮想物体の運動や仮想物体間の関係、および仮想人間の動作を効率的に扱うことが困難であり、構築労力が膨大になる問題が存在した。そのため、原子力プラントの様な複雑で大規模な機器構成を仮想空間内に構築するにはプログラム規模が大きくなり、大規模ソフトウエア開発に共通なプログラミング生産性の困難さがあった。そこで、本研究では、大規模ソフトウエアの開発の効率化に貢献しているオブジェクト指向の概念に注目し、この概念を仮想空間の構築手法に適用して、大規模仮想空間を効率的に開発する手法を考案することを目的に研究を進めた。

オブジェクト指向とは、世界を「もの」を中心に捉える考え方である。本研究では、この概念を採り入れた仮想空間構築手法である OCTAVE (Object-Oriented Technique for Constructing Interactive Virtual Environment) 手法を考案した。OCTAVE 手法は、オブジェクト指向の「世界」を「仮想空間」に「もの」を「仮想物体」に対応させ、仮想空間を仮想物体中心に捉えて構築する手法である。そして、仮想物体間の衝突などのインタラクションも、「仮想物体」の内在する性質によって生じる挙動と規定した。このため、仮想物体を構築する時に、個々の仮想物体が持っている性質のみを考慮すればよいようになり、仮想空間内に配置する他の仮想物体との関係を考慮する必要がなくなった。すなわち、大規模な仮想空間を構築する際に問題となる、仮想物体間の関係の記述が複雑になる問題を回避できた。

本研究では、次に、この OCTAVE 手法に基づいて仮想空間を構築し、実際にシミュレーションを実行するために仮想空間シミュレーションシステム OCARINA (Octave Based Virtual Environment Simulation System) を開発した。そして、仮想空間内の仮想人間が実際の人と同様の動作で作業を実演できる機能を備えた作業実演システムを、OCTAVE 手法に基づいて OCARINA を用いて構築する作業を行い、大規模仮想空間の構築に必要な労力が、従来手法と比較して、少ないことを検証した。

目 次

図 目 次

表 目 次

第 1 章 序論

MATRIX、2199 年の世界で、巨大コンピュータが擬似的に作り出すこの仮想都市空間で、そこがコンピュータの作り出した虚構と気がつくこともなく、多くの人間が何の疑問も感じずに生活している。映画 MATRIX の描く 1 シーンである。

映画の中の世界では、人間はコンピュータに支配される存在であり、コンピュータの発展の負の側面を描き出していた。しかし、現実と区別のつかないほどの仮想社会を作り出すことができれば、その効果は計りしれない。

人は、現実の世界では移動することなく、インターネット上に構築された仮想都市に集うことにより仕事をし、また、買物をすることができる。また、現実には移動することなく、世界各国を旅行することができ、世界中の人と同じ仮想空間内で自由にコミュニケーションすることができる。このような技術が実現すれば、エネルギー問題、CO_2 の排出問題、都市への人口の集中問題、先進国と発展途上国の経済格差問題等の現在社会が抱えている、様々な問題の解決に役立つことが予想される。

近年のコンピュータ技術の発達により、3 次元コンピュータグラフィックス (3DCG) は、コマーシャルフィルムや映画、テレビゲーム等、我々の身の回りのありとあらゆる場面で利用されている。しかし、これらの技術は、3 次元映像をみせる技術であり、MATRIX のように 3 次元空間で擬似体験できる仮想空間を構築する技術はまだまだ実用的とはいえない段階である。

実用的でない理由の 1 つとして考えられるのが、大規模で自由度の高い仮想空間が存在しないことである。現在、構築されている仮想空間は、仮想空間内に少数の物 (仮想物体) が配置され、最初から決められた物を、決められた方法で操作することしかできないものが大半である。これは、そのような仮想空間が、その中に配置されている仮想物体の動き方や、仮想物体と仮想物体の間のインタラクション、仮想物体と仮想人間の間のインタラクションを一々定義しているため、仮想空間を大きくしたり、自由度を大きくすると急速に構築作業が複雑になるからである。

本研究では、仮想空間の構築にオブジェクト指向の概念を導入することにより、これらの問題の解決をはかる。オブジェクト指向とは、ソフトウエア開発の分野で活発に用いられている概念であり、この分野では複雑大規模なソフトウエアの開発の効率化に大きな効果をあげている。

以上により、本研究ではMATRIXのような仮想空間を作るための第一段階として、オブジェクト指向の概念を仮想空間の構築に適用し、複雑大規模な仮想空間を少ない労力で構築することができる手法を提案し、その手法に従って構築された仮想空間をシミュレーションすることができる仮想空間シミュレーションシステムを開発することを目的とする。

以下に、本論文の構成について述べる。まず、第2章では、本研究の背景について述べた後、本研究に関する従来研究についてまとめ、本研究の目的を明らかにする。第3章では、オブジェクト指向の概念について説明し、それを適用した仮想空間を構築するための手法を提案する。そして、提案した手法で仮想空間を構築することによる利点をまとめる。第4章では、提案した手法に従って仮想空間を構築し、シミュレーションするために開発したシステムについて説明する。第5章では、提案した手法に従って実際に仮想空間を構築し、提案する手法の効果を評価し、今後の展望について述べる。最後に、第6章で本研究の結論を述べる。

第 2 章 研究の背景と目的

本章では、まず、Virtual Reality の現状について展望し、次に、Virtual Reality を様々な分野に応用するにあたり、重要な要素技術となる仮想空間の構築手法について、その研究開発の現状と問題点、課題等を述べ、最後に本研究の目的を述べる。

2.1 Virtual Realityとその応用分野

仮想現実感 (Virtual Reality:VR) とは、「計算機によって作り出された合成情報を人間の感覚器官に直接提示することにより、仮想世界 の能動的な疑似体験を可能にする」ための技術である [1][2]。近年のコンピュータの計算処理能力の飛躍的な発展と、インタフェース技術の発展により、従来より活発に利用されてきた、ゲームなどのエンターテイメントの分野だけでなく、様々な分野で VR の実用化に対する試みがなされている。

VR の主な応用例としては、設計段階の家やビルなどの中を仮想的に歩き回ることで、実物を構築することなく使用感を評価できるシステム [3]、インターネットを使用したオンラインショッピングで、実際の製品が手元になくても製品の使用感を確かめることができるシステム等がある。

VR を用いた訓練環境もその応用の 1 つであり、実機を利用した訓練環境と比べ、以下のような利点が期待される。

1. プラントのような大規模な訓練環境を、限られた広さの場所に構築できる。
2. 実機を用いると危険な訓練でも、安全に実施できる。
3. 訓練環境に訓練生を支援するための各種機能を付加することで訓練効率を高めることができる。

このように、VR を用いて原子力発電プラント等の大規模エネルギー機器の保守作業や運転作業を体験できる環境を構築することで、安全・安価に多様な訓練の実施が可能となり、訓練機会の増加、エネルギー利用技術の向上、エネルギー機器取扱上の安全性・信頼性の向上を期待できる。

従ってこれまでに、様々な種類の訓練環境が研究開発されている。

これまでに開発されたVRを用いたプラント運転・保守作業の訓練システムとしては、南雲らによる原子炉ドライウェルにおけるウォークスルー環境[4]、阿部らによる機器の修復支援システム[5]、新井らによる変電所保守員向け体感型シミュレータ[6]、吉川らによるスイング式逆止弁の分解・組み立て作業の訓練システム[7]等をあげることができる。

これらの訓練システムは、訓練生がシステムを利用する際のユーザインタフェースに応じて以下のように分類できる。

分類1 ウォークスルー式訓練システム

ウォークスルー式訓練システムは、訓練生がマウスやキーボード、ジョイスティック等を用いて、仮想空間内を歩き回ることにより、訓練対象機器の構造や各種名称を学習する訓練システムである。訓練対象機器の3次元モデルを作成し、訓練生の視点を変更する機能を構築することにより比較的に容易に実現できるが、作業手順や機器の操作方法等の学習に用いることはできない。

分類2 メニュー選択式訓練システム

メニュー選択式訓練システムは、ウォークスルー式訓練システムに、仮想空間内に配置された物体間の物理的拘束関係を論理的にモデル化してシミュレーションする機能を付加したもので、訓練生はマウスやキーボードで訓練対象機器に対する作業の種類を選択することで、機器の分解や組立作業を行う。主に作業手順の学習に用いることができる。

分類3 ジェスチャ式訓練システム

ジェスチャ式訓練システムは、メニュー選択式訓練システムで、マウス等を用いて作業の種類を選択する代わりに、データグローブや3次元マウス等を用いて、実際の作業に近いジェスチャで機器を操作して作業を進める訓練システムである。仮想物体間の物理的拘束関係を実世界の物理法則に近い形でシミュレーションし、結果を視覚的に表現する必要がある。作業手順に加え、具体的な機器の操作方法も学習できる。

以上に述べた訓練システムの分類の内、分類1の訓練システムは、その構築のために必要な労力が他の分類と比べて最も少なく、技術的にも最も容易に構築できるが、訓練生自らが作業を体験できないために、訓練が受け身となり、訓練効率が最も低い訓練システムである。一方、分類3の訓練システムは、最も構築の労力が大きく、技術的

にも最も構築が困難であるが、実機を用いた訓練に最も訓練環境が類似しており、訓練効率が最も高い訓練システムである [7]。

2.2 本研究室における研究の経緯

2.1 節において、VR を用いた訓練システムの研究開発の現状を述べたが、本研究室では、2.1 節で述べた 3 種類の訓練環境より、効率的に訓練することを目的として、「協調作業型訓練システム」の構築を目指している。協調作業型訓練システムでは、図 2.1 に示すように、仮想作業空間内に実際の人と同じ形状をもつ仮想人間を配置し、その仮想人間と共同作業したり、仮想人間が訓練生に作業内容を教示することで、これまでの訓練システムより効率的に訓練を進めることを目指したシステムである。

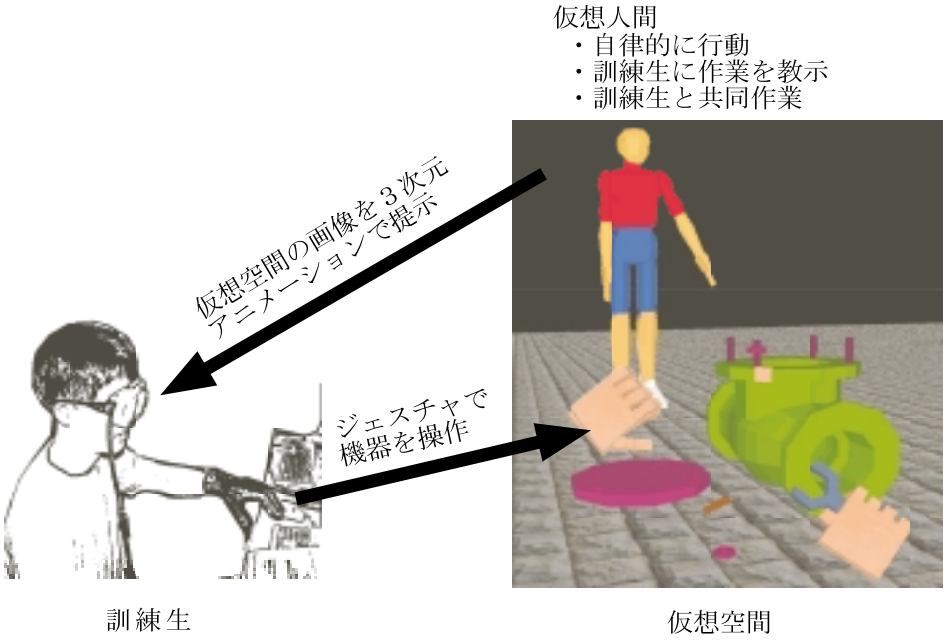


図 2.1: 協調作業型訓練システム

このようなシステムを構築することで、インストラクタが必要な高度で複雑な作業に対する訓練や、複数の訓練生の参加が必要となるチーム連携での訓練などを、訓練生が単独で実施できるようになるため、訓練の機会が増加し、従来の訓練システムより高い訓練効果が期待される。

本研究室では、このような協調作業型訓練システムを実現するために、図 2.2 に示すような構成の訓練システムを考案し、次の 4 つの要素技術に分けて研究を進めてきた。

1. 訓練生の動作の計測技術

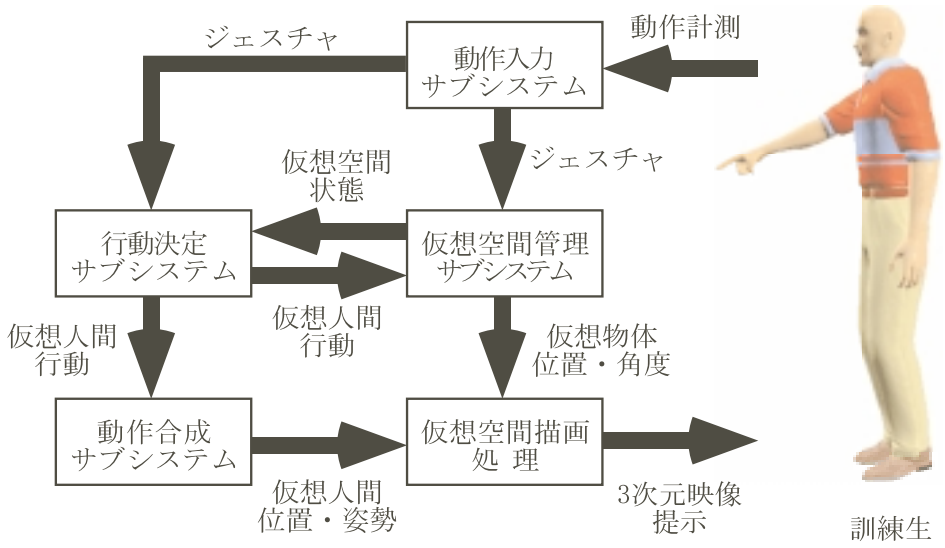


図 2.2: 協調作業型訓練システムの概要

図 2.2 における動作入力サブシステムを実現するための技術で、訓練生のジェスチャを認識する。本研究室では、主に、訓練生の腕の動きを計測するための技術を開発してきた [8]。

2. 仮想人間の行動内容を決定する技術

   図 2.2 における行動決定サブシステムを実現するための技術で、訓練生と仮想人間との協調作業を実現するためには、仮想空間の状況 (作業の進捗状況) と訓練生のジェスチャ等に応じて、適切に行動内容を決定する必要がある。本研究室では主に、原子力発電プラントの制御室における異常診断作業を対象に、運転員のヒューマンモデルを開発する研究を行ってきた [9]。

3. 仮想人間の動作を合成する技術

   図 2.2 における動作合成サブシステムを実現するための技術で、行動決定サブシステムで決定された仮想人間の行動内容を基に、3 次元アニメーションを合成し、訓練生に提示する。本研究室では、これまで、アフォーダンス [10] の概念を参考に人体モーション合成システム AHMSS を開発し、仮想人間が仮想空間内に配置された物体 (仮想物体) から動作合成に必要な情報をリアルタイムに取得し、アニメーションを合成することを可能にしている [11]。

4. 仮想空間の構築および管理技術

   図 2.2 における仮想空間管理サブシステムを実現するための技術で、エネルギープラント等の保守訓練を実施可能にするためには、実世界の物理法則等をシミュ

レーションする必要がある。

本研究では、これらの要素技術のうち、主に 3. および 4. の技術に着目し、少ない労力で効率的に仮想空間管理サブシステムと動作合成サブシステムを構築可能にするためのフレームワークを確立することを目指す。

2.3 仮想空間を構築するための従来手法とその問題点

VR を使用した訓練システムは、現状ではその多くが、VR に精通した専門家によるプログラミングで構築されていて、(1) 専門家でなければ構築できない、(2) 大規模な仮想空間の構築が難しい、(3) 新たに訓練環境を構築する際に、プログラムを修正または作り直す必要があるため、その労力が大きい等の問題がある。そのため、大規模プラントを対象にした訓練システムなど、実用規模のシステムを構築するには、膨大な時間と労力および人件費が必要となり、VR システムの普及の大きな妨げになっている。

これらの問題を解決するためには、主に次の 3 つの方法が考えられる。

1. 仮想空間を構築するための Graphical User Interface(GUI) を用意し、プログラミングすることなく構築できるようにする。

2. 複雑なものを作る際に、対象を分割して個々に構築することを可能にする。

3. 一度作成したものを再利用可能にする。

1. を実現することにより、VR の専門家以外も仮想空間を構築することが容易になり、2. を実現することにより大規模な仮想空間の構築が容易になる。また、3. を実現することにより、新たに仮想空間を構築する際に、過去に仮想空間を構築した時のソフトウエア資産が再利用できるため、作業効率を大幅に向上できると考えられる。

この中で、1. の GUI を介して仮想空間の構築作業を支援するシステムとしては、これまでに「EON[12]」や「dVISE[13]」等が開発されている。「EON」は米国 EON 社が開発したシステムで、プログラミングを行わなくてもデータグローブ等を介して物体の操作ができる仮想空間の構築を可能にしている。また、「dVISE」は英国 Division 社が開発したシステムで、メニューによる対話入力で、プログラミングなしに仮想空間の構築、操作を可能にしている。これらのシステムを用いることにより、大規模仮想空間の構築作業労力はある程度削減されたが、VR を用いた訓練システムを実用化するためには、まだ改善が必要である。例えば、「EON」を用いて、仮想物体の総数が約

20 個、作業工程が約 40 工程の訓練システムを構築した事例では、1 人のシステムエンジニアが構築して、5ヶ月〜6ヶ月の時間が必要であった。これは、1 つの訓練環境を構築する際に、人件費だけで 500 万〜1000 万円程度の費用が必要になることを意味する。従って、実際の訓練の現場で必要となるような数十種類にのぼる訓練環境を構築するためには、膨大な費用が必要となる。

これは、上記の支援システムでは、

- 仮想空間を構築する場合に、仮想空間を仮想物体ごとに分割して作成することができないために、1 つの仮想空間を 1 度にまとめて構築する必要がある。
- 過去に作成した仮想物体に関する情報を、新たな仮想空間を構築する際に再利用することが難しい。

という問題点があるためである。

同様の問題は、1980 年代の大規模コンピュータソフトウエアの開発現場でも発生していた。当時、ソフトウエアを開発する場合には、1 つのソフトウエア全体を一括して開発していたため、ソフトウエアが複雑大規模化するにつれ、開発作業が急速に複雑化し、また、一度作成したプログラムを他のソフトウエアの開発に利用することも困難であった。しかし、近年ソフトウエア開発の分野では、この問題の解決策としてオブジェクト指向 [14] の概念を利用してソフトウエア開発効率を大幅に改善することに成功している。すなわち、ソフトウエアをその機能単位毎に分割して開発し、後に個々の機能単位を組み合わせて複雑大規模なソフトウエアを比較的容易に開発している。また、その分割した機能単位を他のソフトウエアに再利用可能な形にして、ソフトウエア開発の生産性を向上させることに成功している。機能単位毎に分別するソフトウエアのことをソフトウエア工学では、「オブジェクト」と呼んでいる。また、「オブジェクト指向」については、次章で詳しく述べる。

そこで本研究では、このオブジェクト指向に基づいて、複雑大規模なシステムを開発する方法を、仮想空間の構築に利用することを考える。すなわち、大規模複雑な仮想空間の構築においても、仮想空間を構成する仮想物体をオブジェクト指向の概念に基づいて分割して構築しておいて、それらを後で組み合わせて様々な仮想空間を容易に構築できると考えた。

2.4 本研究の目的と方法

本研究では、2.3 節で述べた大規模仮想空間を構築する際の各種問題を解決し、2.2 節で述べた協調作業型訓練システムを、少ない労力で構築可能にするための新たな手法として、オブジェクト指向を用いて、仮想空間を設計・構築する手法を提案し、その手法に従って実際に仮想空間を構築可能なシステムを試作した後、提案する手法の効果を評価することを目的とする。

具体的には、オブジェクト指向を用いて、仮想空間を設計・構築する手法である OCTAVE (Object-Oriented Technique for Constructing Interactive Virtual Environment) 手法を提案し、その手法を基に仮想空間を構築、管理できる試作システムとして OCARINA (Octave Based Virtual Environment Simulation System) を開発する。なお、OCARINA は、現在の訓練システムの主な構築者である、VR の専門家が使用して仮想空間を構築することを想定して設計する。そして、OCARINA を用いた仮想空間の構築例として、仮想空間内に配置された仮想人間が作業を実演できる機能を備えた作業実演システムを構築し、従来手法を用いた構築と比較することにより、OCTAVE 手法による仮想空間の設計・構築の有効性を確認する。

第 3 章 オブジェクト指向に基づく仮想空間構築手法 (OCTAVE) の提案

本章では、まずオブジェクト指向の概要について説明する。その後、仮想空間を構築するための従来手法を説明し、その手法の問題点を指摘した後、問題を解決する手段として、オブジェクト指向に基づいて仮想空間を構築する手法である OCTAVE 手法について説明する。そして、具体的に OCTAVE 手法を用いた仮想空間の設計例について述べた後に、OCTAVE 手法と従来手法を比較検討する。

3.1 オブジェクト指向とは

本節では、まずオブジェクト指向の概要を説明する。その後、オブジェクト指向を適用することで、大規模システム開発の作業効率を改善することに成功した例として、大規模なソフトウエア開発におけるオブジェクト指向の適用方法について述べた後、ソフトウエア開発にオブジェクト指向を適用することの利点についてまとめる。

3.1.1 オブジェクト指向の概要

オブジェクト指向とは、ソフトウエアプログラムをより人間の思考に近い形で記述可能にすることを目指して徐々に確立されてきた概念である。オブジェクト指向の技術体系の大筋を確立したのは、1970 年にゼロックスのパロアルト研究所で開発された Smalltalk というプログラミング言語であるといわれている。その開発の目標は「幼児にもプログラミングできる言語」であり、そのために人間が実世界を頭の中で理解する単位を参考にしたと言われている。

人間は、言葉を通して実世界を理解していると考えられる。例えば、人がリンゴを理解するときには、「林檎」という名詞に、様々な性質を言葉として結合して理解している。すなわち、「皮、果肉、種」などのように林檎を構成する要素や、「甘い、丸い、おいしい」などの性質、そのほかにも「果物」といった分類名も「林檎」という単語に結合することでリンゴを理解していると考えられる。これは、ある特定の「もの」に意味的に近いものがリンクを張っている「意味ネットワーク」として、人が頭の中で、

「もの」を理解していることを説明した物である。

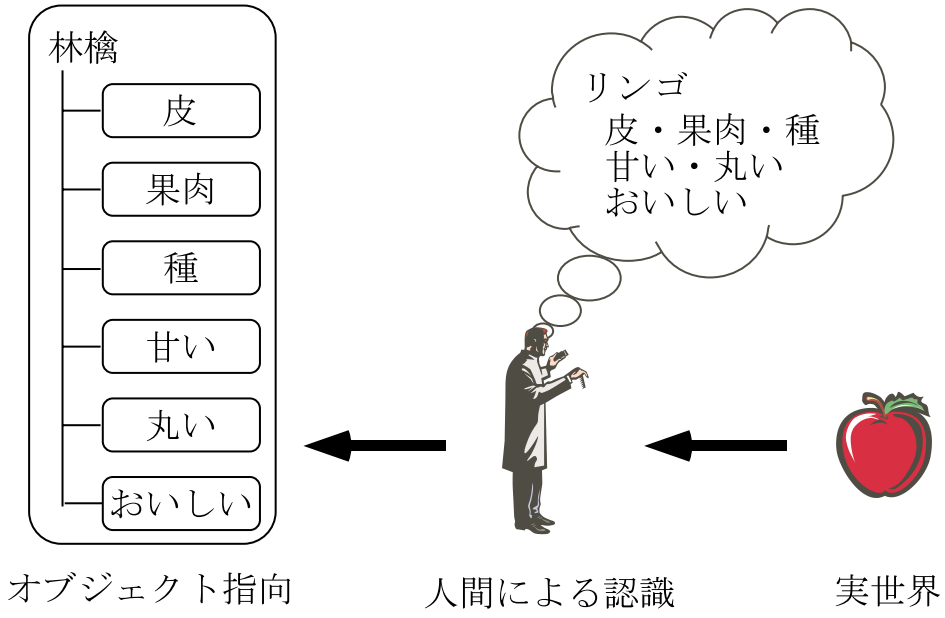


図 3.1: オブジェクト指向の概念

オブジェクト指向での「オブジェクト」とは「もの」のことであり、すなわち、人間が「言葉で理解できるもの」であるといえる。例としてリンゴをオブジェクト指向で構成する場合の概念図を図 3.1 に示す。「林檎」というオブジェクトに、「皮」「果肉」「種」というリンゴを構成するオブジェクトを加え、さらに「甘い」「丸い」「おいしい」という性質を示すオブジェクトを加えることで、リンゴという「もの」を実現することができる。これは、1 つの「もの」に関する知識を、その構成要素や「性質」などの「属性」を中心に見たもので、人工知能での「フレーム」という概念に沿って記述したものである。同じものでも人間の「もの」の捉え方は様々で、人間の思考形態と同様、オブジェクト指向のオブジェクトの構成方法にも決まりはなく、リンゴを対象とした場合、前述の方法のではなく、「原子」というオブジェクトを、多数「林檎」というオブジェクトに追加することにより、リンゴを構成することも可能である。

3.1.2 オブジェクト指向の基本用語

現在、プログラミング等で一般に使用されているオブジェクト指向の概念では、現実世界に対する人間の思考を、なるべく理解しやすく、より効率的に表現するために、いくつかの基本用語を設定している。これが、「オブジェクト」「クラス」「インスタンス」「継承」「メッセージ」である。以下でそれぞれの用語について説明する。

オブジェクト

オブジェクトとは3.1.1項で述べたように、人間が言葉で理解できる「もの」であるが、オブジェクト指向では何を「もの」とすべきかという明確な定義はない。人間が「もの」と理解できるものであれば、すべてオブジェクトということができる。オブジェクト指向に従ってプログラミングするときには、「何をオブジェクトとして扱うか」というオブジェクト指向方法論が数多く提案されている[15][16]。

一方、複数のオブジェクトを定義した場合、その間には「関係」が生じる。オブジェクト指向では、オブジェクト間の関係として次の2種類を定める。

- 階層的な関係
- 並列的な関係

階層的な関係とは、あるオブジェクトが他のオブジェクトに属している場合の関係である。リンゴの例を図3.2に示す。3.1.1項で述べたように、「林檎」というオブジェクトに「皮」や「甘い」などの構成要素や、性質を表すオブジェクトが属していることで、様々な性質を合わせ持つリンゴを表している。この場合の、「林檎」オブジェクトと「皮」などの属しているオブジェクトとの関係が階層的な関係である。このように、オブジェクトに対し階層的な関係を定めることで、「林檎」のように属されているオブジェクト(親オブジェクト)に、「皮」のように属しているオブジェクト(メンバオブジェクト)の性質を追加することができる。

またこの時、親オブジェクトはメンバオブジェクトがなければ機能しないが、メンバオブジェクトは親オブジェクトがどのようなオブジェクトであっても機能する。例えばリンゴの場合、「林檎」は「甘い」という性質を表すメンバオブジェクトがなければリンゴとして機能しない。しかし「甘い」オブジェクトは「林檎」ではなく、「蜜柑」オブジェクトに加わって、みかんに甘いという性質を加えることもできる。すなわち、オブジェクト指向では、親オブジェクトは、メンバオブジェクトに依存関係があるが、メンバオブジェクトは親オブジェクトに依存していない。

並列的な関係とは、あるオブジェクトが他のオブジェクトとは独立に存在している場合の関係である。例えば、図3.3に示すように、「林檎」オブジェクトと「蜜柑」オブジェクトが存在する場合、これら2つの間には、階層関係はなく、双方とも相手のオブジェクトが存在しなくても、それぞれリンゴとみかんとして機能する。この場合、双方のオブジェクトには依存関係がない。

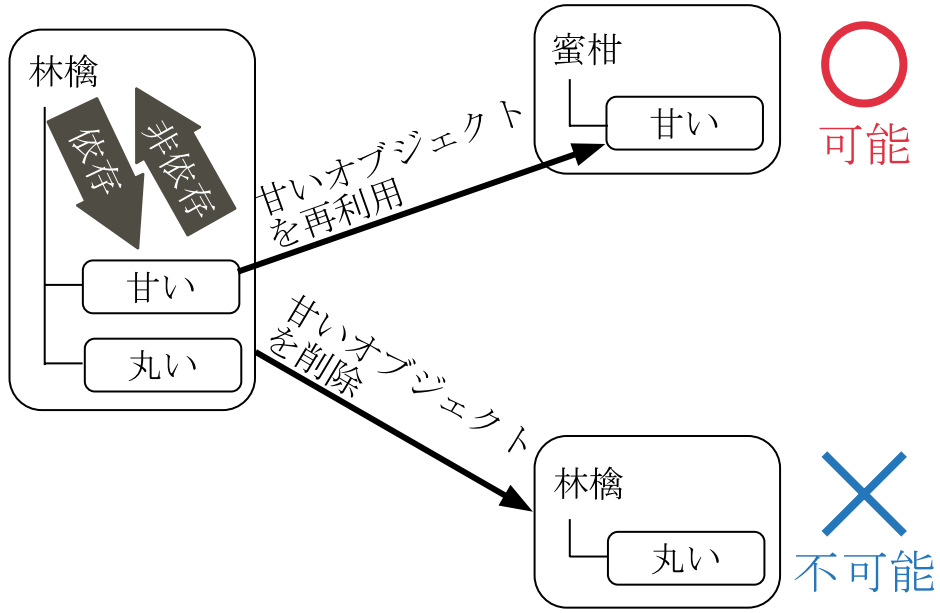


図 3.2: オブジェクト間の階層的関係

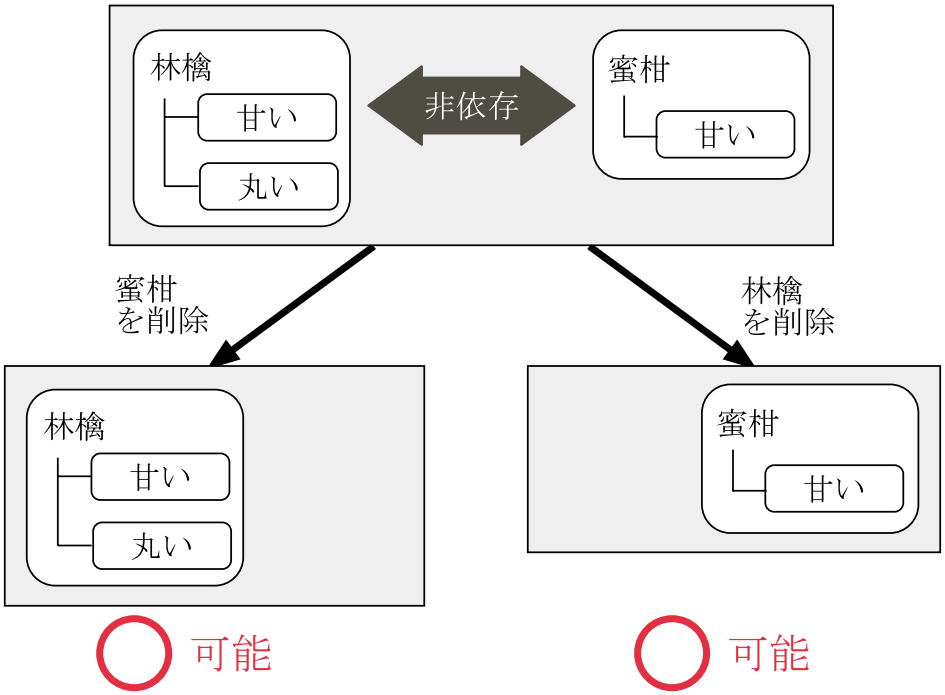


図 3.3: オブジェクト間の並列的関係

クラスとインスタンス

「クラス」と「インスタンス」は、一般にオブジェクト指向に従ってプログラミングする場合に用いられる用語である。クラスは、オブジェクトの性質を定める「型」であり、インスタンスは、オブジェクトの性質を定める型であるクラスを用いて作成された実体である。

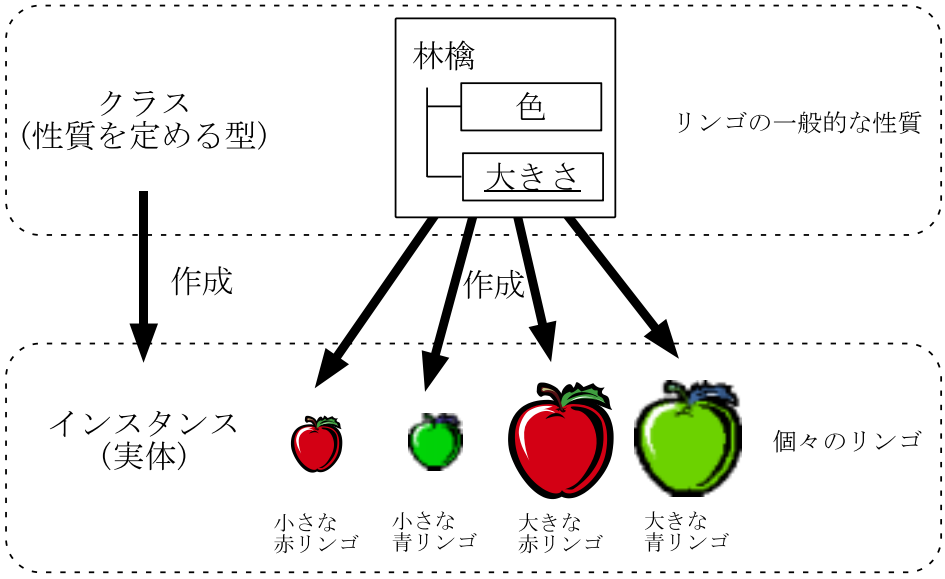


図 3.4: クラスとインスタンス

リンゴの例におけるクラスとインスタンスの関係を図 3.4 に示す。ここに示すように、クラスとはリンゴの設計図であり、大きさに関する性質や色に関する性質等が定義されているが、具体的な大きさや色は定められていない。インスタンスとは、このクラスから作られる実際の林檎オブジェクトで、作られたインスタンスの種類に応じて、具体的な大きさや色が定められている。

継承

継承とは、クラスを定義する際に、良く似た性質をもつクラスを効率的に再利用するために考え出されたオブジェクト指向における機能である。再利用される側のクラスを基本クラスとよび、再利用する側のクラスを派生クラスと呼ぶ。

例えば、人間はリンゴについて甘さや色といった性質とともに、果物という分類を考える。これは、リンゴが果物に共通の性質を持っているためである。ここで、果物に甘いという共通の性質があるとする。図 3.5 に示すように、この果物というグループに属する「りんご」や「蜜柑」は「甘い」という共通の性質を受け継いでいる。この場

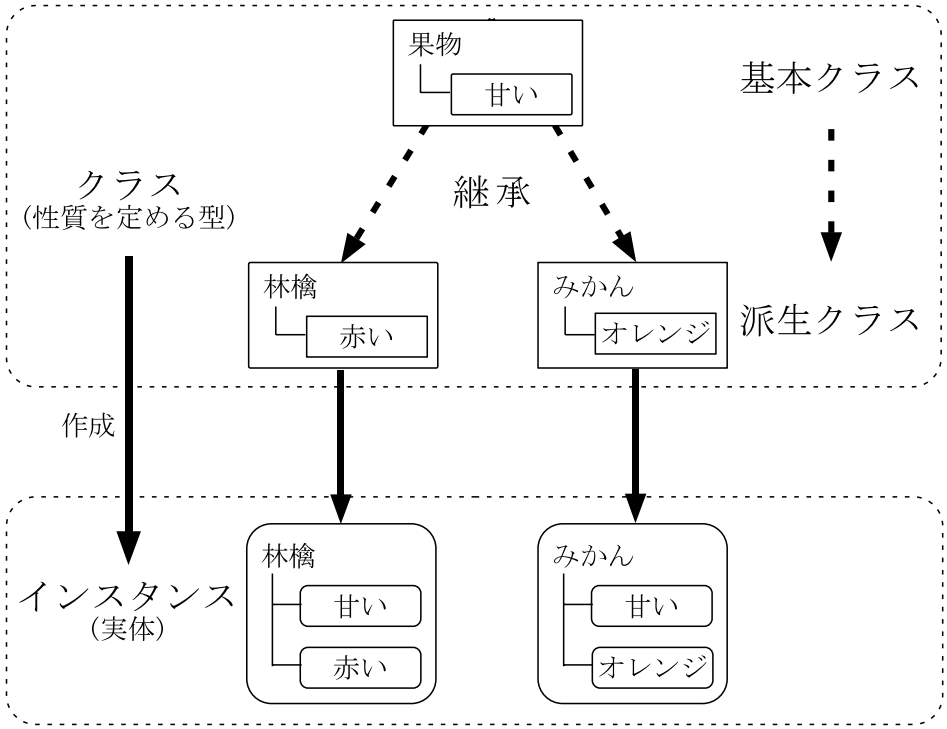


図 3.5: 継承

合、「林檎」や「蜜柑」は「果物」を継承して定義することができ、「果物」が持っている性質は、すべて「林檎」や「蜜柑」も持つことになる。

メッセージ

前述したように、オブジェクトは階層的な関係と、並列的な関係を持っている。オブジェクト指向において、2つのオブジェクト間の依存関係は、階層構造としての親クラスからメンバクラスへの依存だけであり、並列的な関係の場合は、依存関係を定めることができない。

そのため、並列関係の 2 つのオブジェクトは、お互いにどのような作用を及ぼすこともできない。これを解決するのがメッセージである。メッセージとは、双方の性質による作用を監視し、なんらかの作用を及ぼすことが判明したときに、双方の性質にそのことを伝えることで、依存関係がある場合と同様に、相互に作用を及ぼすことができる機能である。

例えば、ともに丸いという形状をもつ「林檎」と「蜜柑」があった場合、2つには依存関係がないため、丸い形状という性質を相手が持っていることはわからない。しかし、図 3.6 に示すように、メッセージが双方の位置と形状を監視し、衝突すると双方に

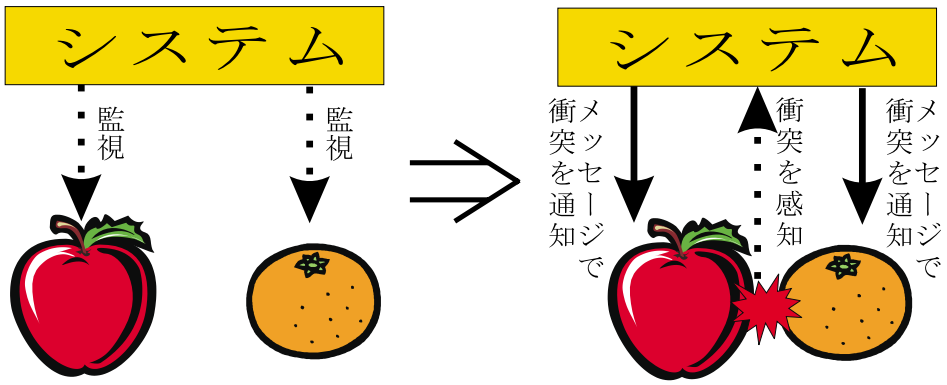


図 3.6: メッセージ

衝突を通知することにより、衝突したときの相互作用が可能となる。

3.1.3 ソフトウエア開発におけるオブジェクト指向の適用方法

オブジェクト指向を利用することの利点を、オブジェクト指向がよく利用されている分野の 1 つであるソフトウエア開発の分野を例に述べる。

オブジェクト指向の様々な特徴のうち、最も大きな効果を生み出しているのは、「オブジェクト間の関係を階層的関係と並列的関係という 2 種類の関係だけに限定することで、オブジェクト間の依存関係を限定することが可能である」ということである。この特徴により、次のような利点が生じる。

1. 新たなクラスを定義する際、依存関係のないクラスは意識する必要がないため、同時に意識するべき範囲が限定され、クラスを定義する際の認知負荷を軽減できる。

2. クラス間に依存関係がないので、クラスの定義を再利用しやすい。

3. クラス間に依存関係がないので、クラス定義を変更しやすい。

例えば、ソフトウエア開発でクラスを定義する場合、クラスの中に数多くの関数を作成する。これらの関数は、依存関係のないクラスの中で定義されている関数からは使用されることはなく、依存関係のないクラスの関数を呼び出すこともない。このため、関数を構築する際に、依存関係のないクラスの関数は意識しなくてよいので、意識する範囲が限定され、認知負荷を軽減できる。また、クラス内の関数から依存関係のないクラスの関数を呼び出さないため、たとえ依存関係のないクラスが存在しなくても、個々のクラスは実行可能である。また、依存関係がないクラスの関数から、ク

ラス内の関数が呼び出されることはないので、関数を変更しても影響の及ぶ範囲が限定される。このため、個々のクラスの定義を変更しやすい利点がある。

また、その他の機能により、次のような利点が生み出されている。

1. 継承機能を用いることにより、クラスを定義する労力が削減できる。
2. ソフトウエアをオブジェクトに切り分けて作成することにより、人間の思考に近い形でソフトウエアを構築することが可能なため、複雑大規模化しても全体像を把握しやすい。

これらの利点により、オブジェクト指向を使用することで、大規模なソフトウエア開発の作業効率が向上する。また十分な数のクラスが既に定義されているときには、小規模なソフトウエアを開発する際にも、既存のクラスを組み合わせる手法での開発が可能となるため、開発労力の削減効果が大きい。

3.2 訓練環境としての仮想空間が備えるべき機能

2.2 節で述べたように、本研究では協調作業型訓練システムを含めた仮想空間を効率的に構築するための新たな手法を提案することを目的とする。

協調作業型訓練システムを構築するためには、以下のような機能を備えた仮想空間を設計する必要がある。

機能 1 仮想物体が実世界と同様の物理法則に従って振舞う。

機能 2 仮想人間が実際の人間と同様の動作で行動する。

機能 3 訓練生が実世界と同様に仮想物体を操作できる。

機能 4 仮想人間が実世界と同様に仮想物体を操作できる。

機能 5 仮想人間は、仮想空間の状況や訓練生の行動に応じて適切に判断し、自律的に行動する。

上記の機能のうち機能 5 は、仮想人間の頭脳に相当する AI(Artificial Intelligence) を作成することにより実現可能であるが、2.2 節で述べたように本研究の対象外であるので、ここでは言及しない。

本研究で対象とするのは、機能1〜4を備えた仮想空間を少ない労力で効率的に構築できる手法を提案することである。以下で、これらの機能の内、特に機能1を実現するための従来手法とその問題点について述べる。

3.2.1 仮想空間内における物理法則のシミュレーション手法

現実世界では、すべての物体が物理法則に従って動いている。例えば、手が鉛筆を握っているのであれば、鉛筆には、重力、手から直接受ける力など様々な力が働き、その結果、鉛筆は手に追随する。

仮想空間内でも同様に、全ての物体 (仮想人間も含めて) に対して、物理法則を正確に適用し、シミュレーションを実行すれば、上記の機能1〜4は実現できるが、実際には形状データの精度や計算機の処理速度等の問題により、リアルタイムにシミュレーションを実行するのは不可能である。

従って、このようなシミュレーションを行うときには、通常、仮想物体の「状態」という概念を導入する [17]。「状態」の概念とは、次のようなものである。

1. 個々の仮想物体に、複数個の離散的な状態を設定する。
2. 個々の状態に応じて、仮想物体の動き方を設定する。
3. ある状態から別の状態に遷移する条件となる事象 (以下では、イベントと呼ぶ) を設定する。
4. 仮想物体は、定義されている状態のうち、常にどれか1つの状態をとり、その状態に定義されている動き方に従って動きが合成される。イベントが発生すると状態が遷移する。

例えば、鉛筆について状態、イベント、イベントの発生による状態遷移と状態に応じた動き方を図3.7(以下ではこのような図を状態遷移図と呼ぶ) のように設定したとする。この場合、手が鉛筆を握っているのであれば、鉛筆は「手に握られている状態」にあり、その状態に設定されている「手に追随して移動」という動き方をする。また、その状態で鉛筆が「手に放されるイベント」が発生すると、鉛筆は「落ちつつある状態」に遷移し、その状態に設定されている「鉛直下向きに速度を上げながら移動」という動き方をする。同様に、「机の上にある状態」の場合は、「静止」という動き方をし、「手に握られるイベント」が発生すると、「手に握られている状態」になって、再び「手に追随して移動」という動き方をする。

このように、「手に追随して移動」、「鉛直下向きに速度を上げながら移動」等の仮想物体の動き方を、仮想物体の状態に応じて、あらかじめ定められた方法に従って制御すれば、仮想物体の動きをシミュレーションするための計算量を削減できる。また、イベントの監視については、状態に応じて監視するイベントを限定すれば、それも計算量を削減することにつながる。例えば、鉛筆が「机の上にある状態」の場合、この鉛筆は「手に握られるイベント」の発生によって、「手に握られている状態」に遷移する可能性があるために、「手に握られるイベント」が発生するかどうかを常に監視する必要がある。しかし、鉛筆が「手に握られている状態」の場合には、この鉛筆は新たに他の手で握られることはないため、「手に握られるイベント」が発生するかどうかを監視する必要はない。

このように「状態」という概念を導入して、

- 仮想物体の状態に応じて、あらかじめ定めた処理を実行することによって仮想物体の動きを合成する。
- イベントの発生に応じて、あらかじめ定めた決まりに従って仮想物体の状態を遷移させる。

という 2 つの処理だけを行えば計算量が削減され、現在の計算機の処理速度でも現実世界を模した仮想空間のシミュレーションが可能となる。このため、これまで構築された仮想空間は、若干の手法の違いはあるが上記の「状態」の概念を用いて構築されている [18]。

3.2.2 状態を用いた仮想空間の管理手法の問題点

以上で述べたように、仮想物体に状態の概念を導入することで、現在一般に販売されているパーソナルコンピュータ程度の性能の計算機でも現実世界を模した仮想空間のシミュレーションが可能になるが、この手法を用いて実際のプラントの保守訓練に使用可能な規模の仮想空間を構築する場合、新たな問題が発生する。

例えば、仮想空間内に鉛筆と手が存在するときに、状態とイベントの定義は図 3.8 に示すようになる。ここで、新たに仮想空間内に、机を追加すると、状態とイベントの定義は図 3.9 に示すようになり、新たに 3 つのイベントを加える必要がある。つまり、それぞれの仮想物体の間で発生する相互干渉をイベントとして個々に定義する必要があり、仮想物体の数が増えると定義しなければならないイベントが急速に増加し、定義が困難になる。

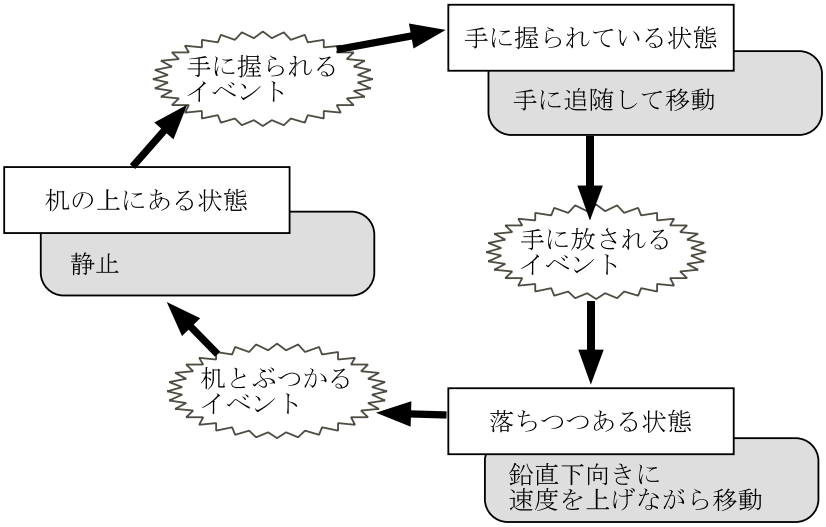
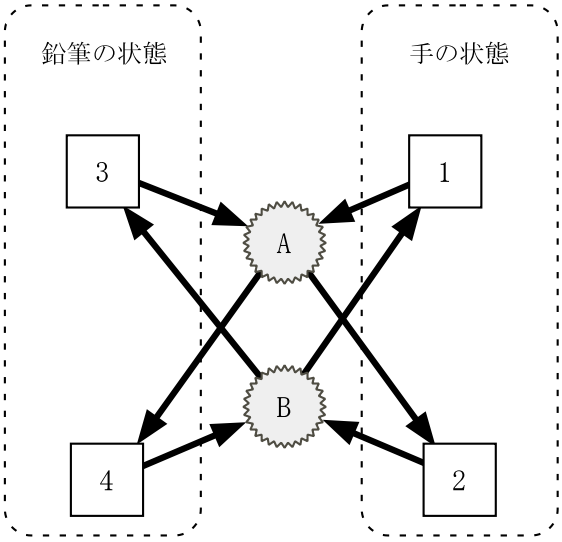


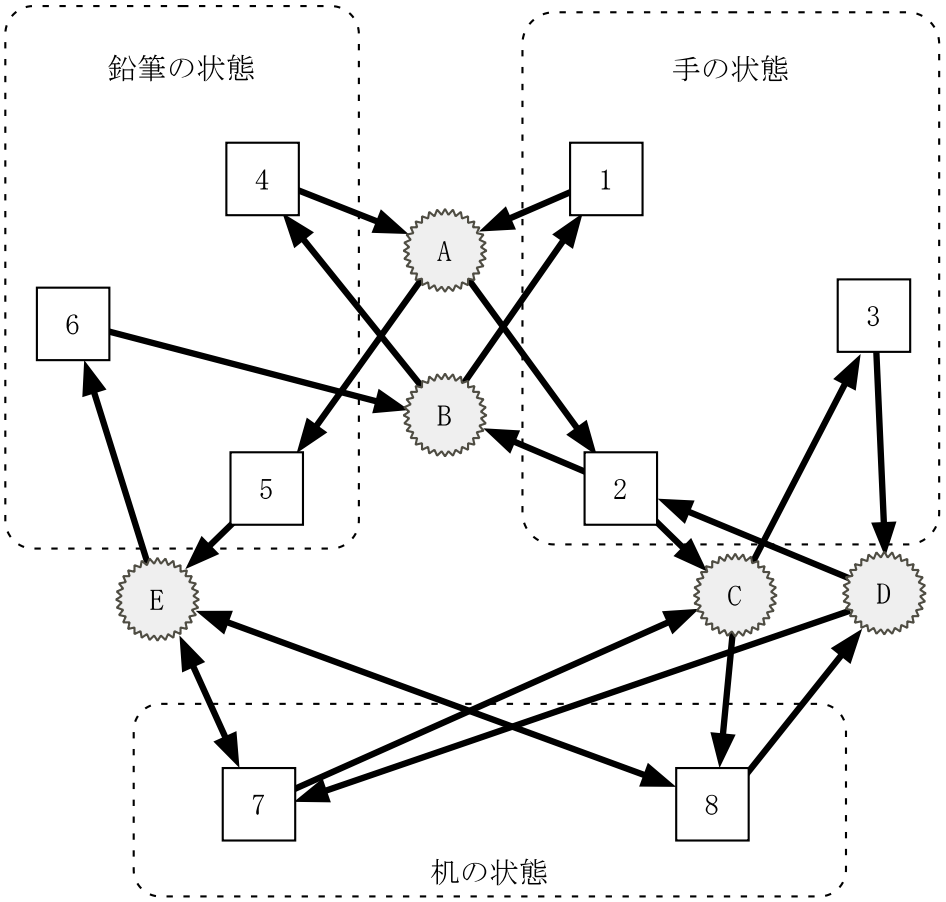
図 3.7: 状態の概念による鉛筆の定義



1:鉛筆を握っている状態
2;鉛筆を握っていない状態
3:手に握られている状態
4:手に握られていない状態

A:鉛筆が手に放されるイベント
B:鉛筆が手に握られるイベント

図 3.8: 状態による定義手法の問題点 1



1:鉛筆を握っている状態
2;何も握っていない状態
3:机を握っている状態
4:手に握られている状態
5:落ちつつある状態
6:机の上にある状態
7:手に握られていない状態
8:手に握られている状態

A:鉛筆が手に放されるイベント
B:鉛筆が手に握られるイベント
C:机が手に握られるイベント
D:机が手に放されるイベント
E:鉛筆が机に衝突するイベント

図 3.9: 状態による定義手法の問題点 2

また、図3.9は図3.8より非常に複雑になっている。このように、仮想物体とイベントが増加すると共に、状態遷移図も急速に複雑になり、全体を理解することが困難になる。

さらに、「イベント」によって、仮想物体は互いが直接つながっており、これを切り分けることはできない。そのため、一度作成した仮想物体を、他の仮想物体から切り離して、新たな仮想空間に追加することは困難である。

以下に、「状態」の概念を用いて仮想空間を構築する場合の問題点をまとめる。

問題点1 仮想空間を構築するときに定義しなければならない項目が、仮想物体の数の2乗に比例するので、大規模な仮想空間を構築する労力が大きい。

問題点2 仮想空間全体を1つの状態遷移図で管理しなければならず、仮想空間が大規模複雑化したときに、理解が困難になる。

問題点3 新たに仮想空間を構築する際に、過去に仮想空間を構築した際の仮想物体の定義の再利用が困難である。

このような問題は、従来の仮想空間の構築手法では、仮想物体間の関係や、仮想物体の性質を定義する際に、個々の仮想物体同士を互いに依存しあう形で定義しているため、結局は1つの仮想空間内に配置される全ての仮想物体を、ひとまとめにして扱っているために生じている。問題点2や3に対する個々の解決策は、多くの仮想空間構築手法で試みられているが、仮想空間内の仮想物体をひとまとめに扱うという原因が解決されていないため、それらの個々の解決策を適用することによって、すべての問題点を解決することは難しい。

そこで本研究では、以上に述べた仮想空間構築上の問題の解決を、オブジェクト指向の概念を適用した仮想空間の構築手法を提案することにより試みる。すなわち、オブジェクト指向では、3.1.2項で述べたように、オブジェクト間の依存関係を、階層的な関係と並列的な関係のみに制限し、全てのオブジェクト間が互いに依存することがない。そこで、個々の仮想物体や、仮想物体の性質を、オブジェクト指向におけるオブジェクトに対応させれば、仮想空間内の仮想物体を分割して扱うことが可能であり、上記問題を解決可能と考えた。

次節以降では、本研究で提案するオブジェクト指向に基づく仮想空間の構築手法について、その詳細を述べる。

3.3 OCTAVE手法の概要

本節では、オブジェクト指向に基づいて仮想空間を構築する具体的な手法について述べる。オブジェクト指向とは、人間の思考に近い形でプログラミングすることを目的として開発された概念である。そのため、人間の思考が非常に多様であるのと同様、オブジェクト指向に基づいてプログラミングする場合も、その手法は非常に多様であり、明確な決まりはない。

仮想空間をオブジェクト指向に基づいて構築する場合も同様で、考え得る手法は非常に多い。例えば、仮想空間全体を1つのオブジェクトとみなせば、3.2節で述べた従来の構築手法もオブジェクト指向に基づいていると捉えることが可能である。すなわち、オブジェクト指向を仮想空間構築に適用すれば、必ず仮想空間の構築労力が削減できるわけではなく、その適用方法が重要となる。本節では、適用方法まで含めたオブジェクト指向型仮想空間構築手法である、OCTAVE手法について述べる。

以下では、まず、OCTAVE手法へオブジェクト指向の適用の基本方針とOCTAVE手法における仮想物体の構成について説明したのち、OCTAVE手法におけるクラスの構成について説明する。そして最後に、オブジェクト指向のメッセージ機能を利用して、仮想物体同士のインタラクションを実現する方法について述べる。

3.3.1 オブジェクト指向適用の基本方針

人間は、世界を物体の集合として把握し、物体を性質の集合として把握している。なお、ここで「性質」とは、形状、動き方など、各物体が個々に持っている特徴である。世界を仮想空間に対応させると、物体は仮想物体に、物体の性質は仮想物体の性質に対応する。従って、人間が仮想空間を把握する場合は仮想物体の集合として、仮想物体を把握するには仮想物体の性質の集合として認識すると、把握しやすいであろう。また、仮想空間をオブジェクト指向に従って構築する際には、仮想空間を分割して構築する必要があるが、仮想空間を仮想物体の単位まで分割し、仮想物体を性質の単位まで分割することは、人間にとって直感的であり、また、個々の仮想物体を他の仮想物体と独立に定義することが可能となるため、設計が容易になると考えられる。

そこで、OCTAVE手法では、仮想空間を仮想物体の単位まで分割し、さらに仮想物体をその性質の単位まで分割する。そして、仮想物体あるいは仮想物体の様々な性質を、オブジェクト指向におけるクラスに対応させる。図3.10に、OCTAVE手法における仮想物体の構成を示す。仮想物体は、1つの仮想物体クラスと、そのメンバクラスで

ある複数の性質クラスで定義される。仮想物体クラス自身は、仮想物体の具体的な性質は持っておらず、メンバクラスである性質クラスを制御することにより、個々の仮想物体の性質をシミュレーションする。

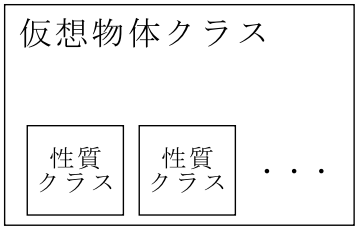


図 3.10: 仮想物体の構成

3.3.2 クラスの構成

OCTAVE 手法では、仮想物体を定義する仮想物体クラスと仮想物体の性質を定義する性質クラスは、共に、図 3.11 に示すように、

- クラスの離散的な状態を管理するペトリネット [19][20]
- クラスのペトリネットだけでは管理できない連続的な状態を管理する変数
- クラスに様々な性質を追加するメンバクラス

で構成される。以下で、各構成要素について説明する。

クラスの離散的な状態を管理するペトリネット

OCTAVE 手法では、クラスは内部状態を持ち、イベントの発生に伴って、その内部状態が変化する。個々の内部状態には、仮想物体の性質をシミュレーションするための処理が設定され、クラスの状態が変化すると、実行する処理の仕方が変化する。例えば、仮想物体の「動き」に関する性質を表すクラスの内部状態が「持たれている状態」であれば、仮想物体が「持っている相手の動きに追随する」ための処理を実行し、また「静止している状態」であれば、仮想物体の位置・姿勢を一定にするための処理を実行する。このようにクラスの内部状態に応じて実行する処理の仕方を変化させることで、仮想物体の状態の変化に応じた性質の変化をシミュレーションする。

OCTAVE 手法に基づいて、クラスを新たに作成する場合には、以下に示す 4 つの項目を設定することになる。

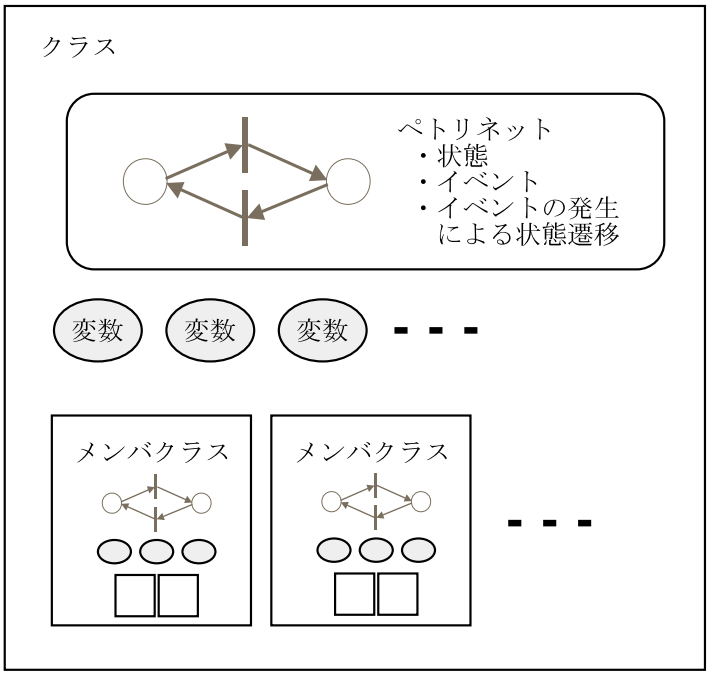


図 3.11: クラスの構成

1. クラスが取りうる状態の種類と各状態において実行する処理の方法

2. クラスの状態遷移の仕方とクラスの状態遷移の原因となるイベント

3. 状態遷移が発生した場合に新たに発生させるイベント

4. あるイベントの発生によってクラスの状態が遷移した場合に、イベントを発生させた他のクラスから受け取る情報

1. は、クラスが取りうる状態の名前と、各状態に置いて実行する処理方法の名前を指定する。2. は、イベント発生前の状態とイベント発生後の状態、状態遷移させる原因となるイベント名を指定する。ここで指定するイベントは後に述べるメンバクラスに所属するイベントのみが指定可能であるという制約条件を課す。この制約により、階層関係にあるクラス以外のクラスに対する依存関係を排除することができる。3. は、クラスの状態遷移が発生した場合に新たに発生させるイベント名を指定する。ここでも指定するイベントはメンバクラスに所属するイベントのみに制約する。4. は、後述するように、複数の仮想物体間のインタラクションを実現するために、仮想物体間での情報のやりとりをオブジェクト指向におけるメッセージの形で実現するための仕組みである。具体的なインタラクションの実現方法は 3.3.3 項で述べる。

OCTAVE 手法では、以上の 4 つの項目をペトリネットによって設定する。ペトリネットは、カール・アダム・ペトリ (Carl Adam Petri) 博士が提唱した離散的、並行的な状態遷移のモデル化に用いる有向グラフである。ペトリネットの特徴は、「グラフィカルな視覚的ツール」、「シミュレーションツール」、「数学的方法論」の三つの機能を同時に備えていることである。本研究では、この 3 つの機能のうち、「グラフィカルな視覚的ツール」という機能や「シミュレーションツール」という機能に注目し、ペトリネットを採用した。ペトリネットを用いるとクラスの状態遷移の仕方を視覚的に定めることができる。

ペトリネットの要素名称	形状	クラスの要素
プレース	○	状態
アーク	→	状態遷移
トランジション	|	イベント
トークン	●	現在の状態

図 3.12: ペトリネットと状態、イベントの対応

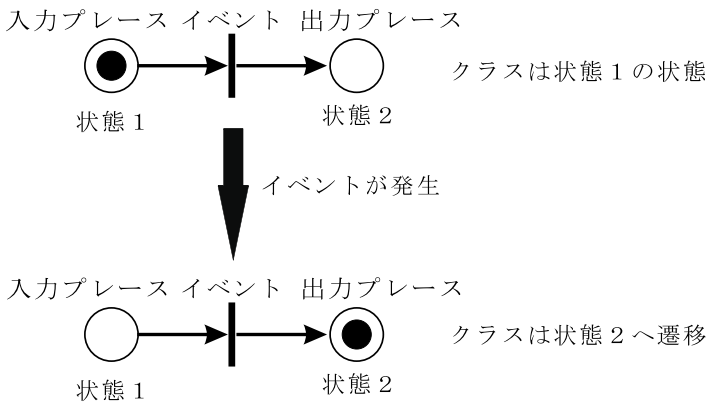


図 3.13: ペトリネットの例

クラスの状態およびイベントとペトリネットの各要素との対応関係を図 3.12 に示す。ペトリネットにおけるプレース 1 つは、クラスの状態 1 つに対応し、トランジションはイベントに対応する。図 3.13 に、OCTAVE 手法において、クラスの状態遷移の仕方をペトリネットを用いて定義した例を示す。図 3.13 に示すように、プレースにトークンが存在する場合、クラスは、そのプレースに対応づけられた状態にあることを意味する。仮想空間のシミュレーションを実行することにより、何らかのイベントが発生すると、対応するトランジションが発火し、トークンが入力プレースから出力プレース

に遷移する。これにより、トークンが存在するプレースの種類が変化し、クラスの状態遷移がシミュレーションされる。

変数

仮想物体の離散的な状態変化は、前述の状態とイベントを用いることによってシミュレーション可能であるが、仮想物体の位置や姿勢など、その値が連続的に変化する場合には、状態とイベントだけではシミュレーションすることができない。

そこで、OCTAVE 手法では、クラスに変数という概念を導入する。これは、C 言語や BASIC などのプログラミング言語における変数の概念とほぼ同等であり、シミュレーションの実行によって自由にその値を変更することができる。変数の値を自由に変更することによって、仮想物体の位置や姿勢などの連続的変化のシミュレーションが実現できる。

メンバクラス

OCTAVE 手法では、3.1.2 項で述べた場合と同様に、メンバクラスは親クラスに対して、性質を付加するために使用する。図 3.14 に親クラスとメンバクラスの関係を示す。図 3.14 において、クラス A,B,C は、それぞれ性質 1,2,3 を持つとする。この場合、クラス A が、そのメンバクラスとしてクラス B とクラス C を持つ場合、クラス A は、性質 1,2,3 をすべて兼ね備えたクラスとなる。OCTAVE 手法では、親クラスになるのは、仮想物体クラスであり、メンバクラスになるのは性質クラスである。これにより、仮想物体クラスに各種性質クラスを加えることで、仮想物体の様々な性質を定義できる。

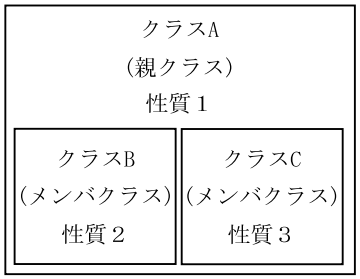


図 3.14: 親クラスとメンバクラスの関係

3.3.3　仮想物体間のインタラクションの実現方法

OCTAVE 手法では、各仮想物体は、他の仮想物体の存在を意識せずに構築できるようにするために、仮想物体をオブジェクト指向のオブジェクトに対応させ、他の仮想物体との関係をオブジェクト指向の並列の関係、つまり依存関係のない関係としている。このため、衝突などの仮想物体間のインタラクションを定義する際に、インタラクションする相手を指定することはできない。

そこで、本研究では、オブジェクト指向のメッセージの概念を導入し、仮想物体間のインタラクションを実現する。OCTAVE 手法では、メッセージを次のように実現する。

- すべてのオブジェクトと独立したインタラクション監視システムを想定する。
- メッセージはインタラクション監視システムとオブジェクトの間で双方向にやりとりすることができる。
- インタラクション監視システムからオブジェクトへのメッセージは、特定のイベントを発生させる形で送られる。
- メッセージにはデータを含めることができる。

例えば、2 つの仮想物体間で、衝突というインタラクションを実現する場合、すべての仮想物体と独立のインタラクション監視システムである衝突監視システムを用意し、双方の仮想物体に、衝突したというメッセージを受け取ることができるイベントを用意する。そして、以下のような流れで、衝突するというインタラクションが実現される。

1. 衝突監視システムが、双方の仮想物体の動きを監視する。
2. 衝突監視システムは、双方の仮想物体が衝突する位置関係になったと判断すると、双方の仮想物体に、衝突相手の名称というデータとともに、衝突を通知するメッセージを送信する。
3. 双方の仮想物体がメッセージを受信することにより、それぞれの仮想物体の衝突イベントが発生し、仮想物体の状態が遷移する。
4. 仮想物体の動き方が変わり、衝突の現象を実現する。

3.4 OCTAVE手法を用いた仮想物体の定義例

本節では、鉛筆と手を例に、OCTAVE 手法に基づいて仮想物体を定義する方法を説明する。また、その定義を基に、実際に仮想空間をシミュレーションする場合の仮想物体間のイベント発生関係を説明する。

鉛筆には、「持たれる」という性質と「衝突する」という性質があり、手には、「持つ」という性質と「衝突する」という性質があるとする。

鉛筆を OCTAVE 手法に基づいて設計する場合、仮想物体クラスである鉛筆クラスに、性質クラスをメンバクラスとして加える。鉛筆クラスに追加する性質クラスとしては、鉛筆の「持たれる」性質を定義する被把持クラスと、「衝突する」性質を定義する衝突判定クラスの 2 つを定義する。

また、同様の方法で、手は、手クラスとそのメンバクラスである「把持クラス」と「衝突判定クラス」で構成される。衝突判定クラスはメンバクラスであり、オブジェクト指向の階層的関係で述べたように、親クラスには依存しないので、鉛筆のために定義したものを再利用して、手クラスにメンバクラスとして加えることができる。

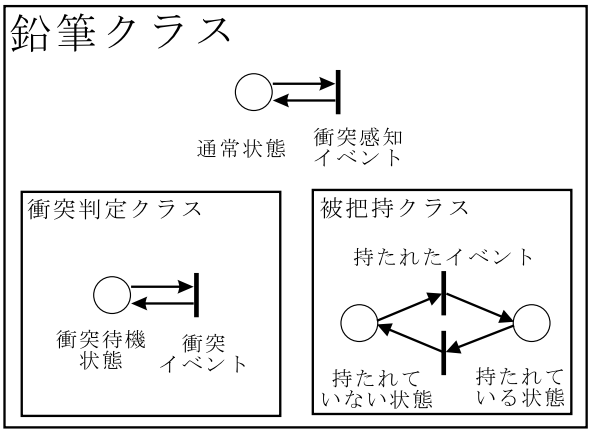


図 3.15: 鉛筆クラスの定義

鉛筆クラスの定義を図 3.15 に、手クラスの定義を図 3.16 に示す。ここで、各クラスの状態遷移の仕方を以下ように定義する。

- 被把持クラスには、「持たれている状態」と「持たれていない状態」の 2 つの状態を定義し、さらに「持たれたイベント」を定義する。そして、「持たれたイベント」が発生すると、被把持クラスの状態は「持たれていない状態」から「持たれている状態」へと遷移すると定義する。「持たれている状態」の時には、持たれる相手の仮想物体に追随する動きをシミュレーションする処理を定義する。

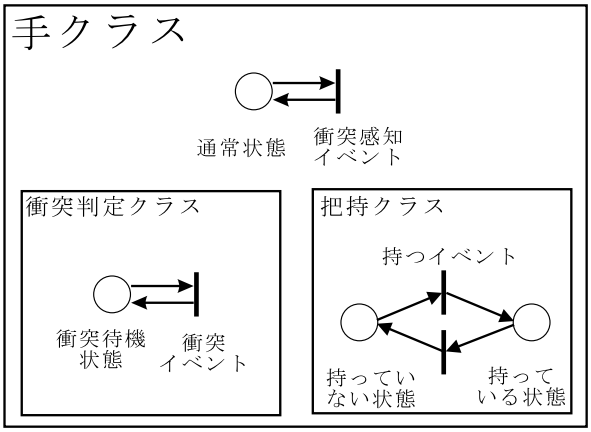


図 3.16: 手クラスの定義

- 衝突判定クラスには、「衝突待機状態」と「衝突イベント」を定義し、衝突したというメッセージを衝突監視システムから受け取ったら、「衝突イベント」が発生するように定義する。

- 鉛筆クラスは、「通常状態」と「衝突感知イベント」を定義し、衝突感知イベントは、イベント発生条件として、メンバクラスである衝突判定クラスの「衝突イベント」を定義し、イベント発生時には、被把持クラスの「持たれたイベント」を発生させるように定義する。

- 把持クラスには、「持っている状態」と「持っていない状態」の 2 つの状態を定義し、さらに「持つイベント」を定義する。そして「持つイベント」が発生すると、「持っていない状態」から「持っている状態」へと遷移すると定義する。

- 手クラスには、「通常状態」と「衝突感知イベント」を定義し、「衝突感知イベント」は、イベント発生条件として、メンバクラスである衝突判定クラスの「衝突イベント」を定義し、イベント発生時には、被把持クラスの「持たれたイベント」を発生させるように定義する。

次に、鉛筆が手に握られるという現象が、上記の定義に従って、どのようにシミュレーションされるのかを説明する。なお、上記の鉛筆と手の定義では、仮想物体の性質を定める段階であり、実体が存在しないので、一般のオブジェクト指向の用語の使用方法に従い「クラス」という言葉を使用したが、以下の仮想空間のシミュレーション時の説明では、クラスを用いて仮想物体を作成した後の、実体としての仮想物体間の関係であるため、「インスタンス」という言葉を使用する。

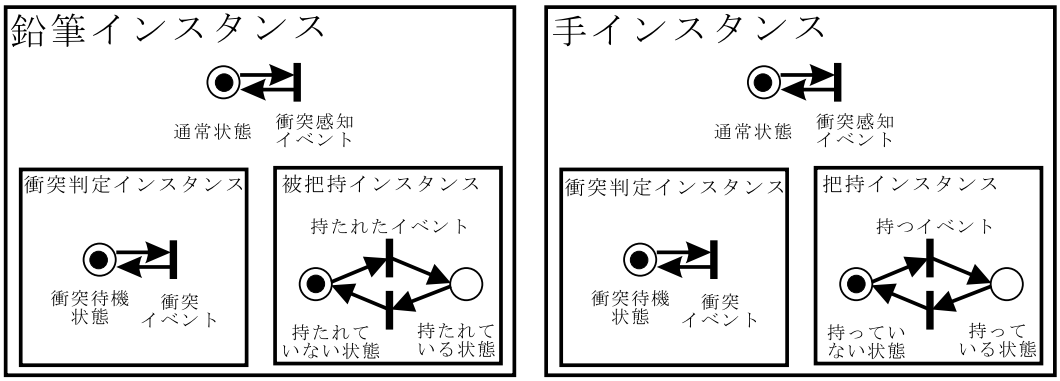


図 3.17: 鉛筆が手に持たれていないときの状態

図 3.17 に示すように、仮想物体の初期状態として、鉛筆の披把持インスタンスは「持たれていない状態」であり、静止しているとする。手が鉛筆に徐々に近づき、手と鉛筆が接触すると、衝突したというメッセージが衝突監視システムからインスタンスへ送られ、鉛筆と手の双方の衝突判定インスタンスの「衝突イベント」が発生する。この時、鉛筆インスタンスでは、「衝突感知イベント」の定義に従い、衝突判定インスタンスの「衝突イベント」、鉛筆インスタンスの「衝突感知イベント」、被把持インスタンスの「持たれたイベント」が連続して発生する。これにより、仮想物体の状態は図 3.18 に示す状態へ遷移する。この時、被把持インスタンスは、「持たれている状態」になり、手に追随して動く処理が実行される。

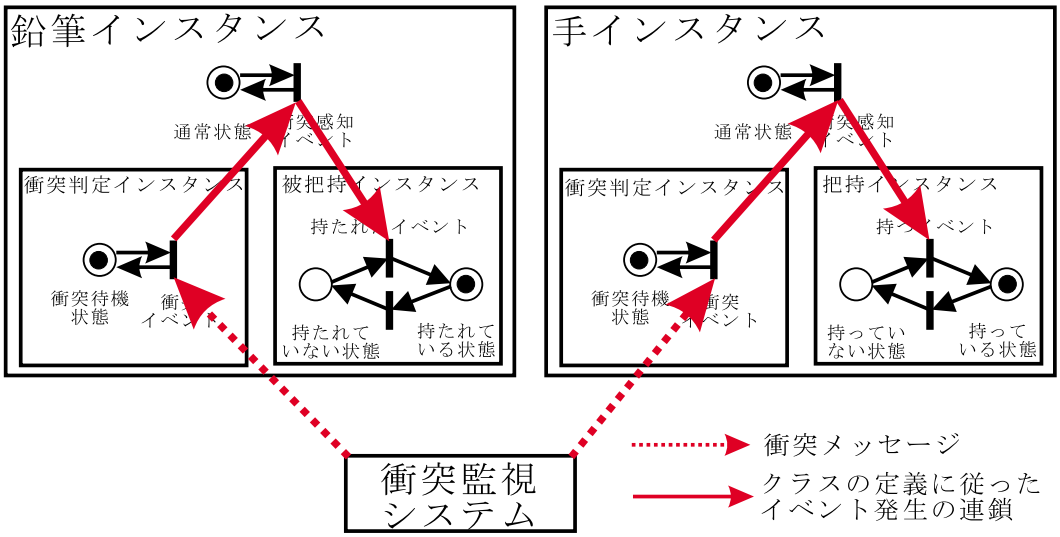


図 3.18: 鉛筆が手に持たれているときの状態

OCTAVE 手法で構築した仮想物体は、以上に述べたようにシミュレーションが実行され、「手に衝突したときに手に追随して動き出す」という鉛筆の動きを仮想空間内に

シミュレーションで実現する。

3.5 OCTAVE手法と従来手法の比較

本節では、OCTAVE手法を用いて仮想空間を構築することにより、従来手法で仮想空間を構築する場合と比較して、どのような利点が生じるかを述べる。

まず、他の仮想物体との直接の依存関係が無くなったことで、他の仮想物体を意識せずに個々に仮想物体を構築することが可能となる。従来手法で構築した仮想空間のイメージを図3.19に示す。この図に示すように、従来手法では、新しく仮想物体を追加する際に、仮想空間内に既に配置されている仮想物体の数の2乗に比例する数の仮想物体間の関係を考慮しながら構築しなければならなかった。そのため、仮想空間内の仮想物体の数が増すにつれて、構築労力は急速に増加することとなった。それに対し、図3.20に示す、OCTAVE手法を用いた仮想空間では、仮想物体間の関係は、メッセージで一括管理するので、それぞれの仮想物体間を、他の仮想物体との関係を考慮することなく、メッセージを受けたときの処理のみを考慮して構築することができるようになった。これにより、構築労力を大幅に節約することができる。

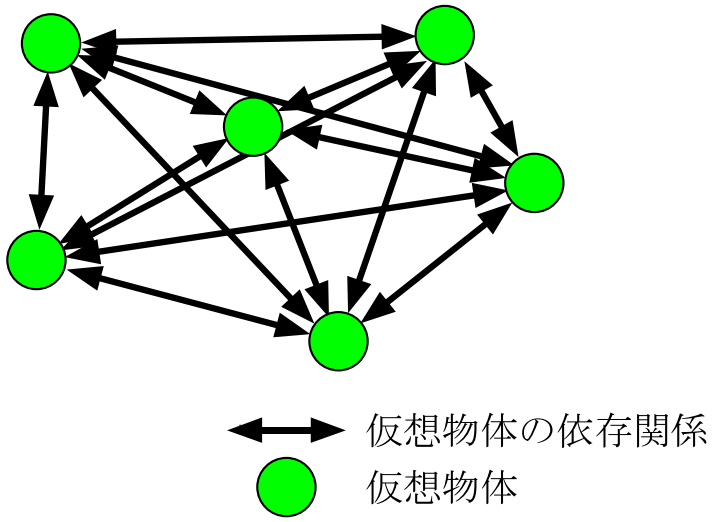


図 3.19: 従来手法

さらに、従来手法では、仮想物体の状態遷移の仕方を定める際に、他の仮想物体を直接指定して定義していたために、個々の仮想物体を、他の仮想物体から切り離して、別の仮想空間に再利用することが難しかった。このため、新たに仮想空間を構築する場合、過去の構築資産を活用することが難しかった。これに対し、OCTAVE手法では、

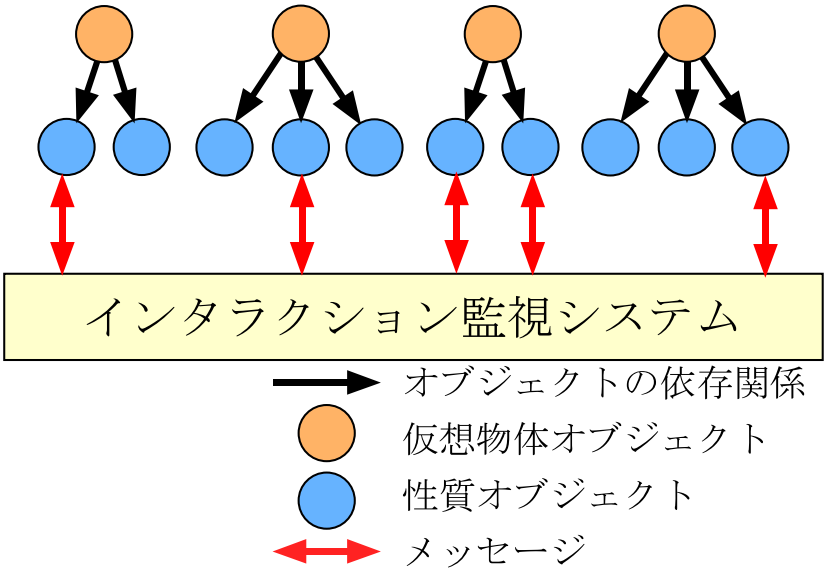


図 3.20: OCTAVE 手法

個々の仮想物体は、他の仮想物体との間に依存関係はなく、他の仮想空間を構築する場合に、仮想物体の定義を変更することなく使うことが可能となった。さらに、仮想物体の性質は、他の性質や、親の仮想物体に依存していないので、仮想物体間で、同じ性質の定義を共有することも可能である。これらの再利用性の向上により、構築の労力が大幅に節約できる。

このように、OCTAVE 手法を使用することにより、仮想空間構築の労力を大幅に削減することが可能である。

第 4 章 仮想空間シミュレーションシステム (OCARINA) の開発

本章では、3 章で述べた OCTAVE 手法を用いて設計する仮想空間を、実際に構築し、シミュレーションするためのシステムとして開発した、仮想空間シミュレーションシステム OCARINA について述べる。まずはじめに、2.2 節で述べた協調作業型仮想空間を実現するシステムとして OCARINA が備えるべき機能について述べた後、開発した OCARINA の概要について説明し、次に OCARINA を構成する各サブシステムの役割を説明する。その後、OCARINA を実行する際の処理の流れについて説明し、最後にシステムのユーザが OCTAVE 手法に基づいて仮想空間を設計し、OCARINA を用いてシミュレーションを実行するまでの作業手順をまとめる。

4.1 OCARINA が備えるべき機能

3 章で述べた OCTAVE 手法は、仮想空間をオブジェクト指向に基づいて設計する手法であるが、本節では、この OCTAVE 手法に基づいて設計した仮想空間の情報を用いて、2.2 節で述べた協調作業型訓練環境のシミュレーションを実行可能にするために、OCARINA が備えるべき機能についてまとめる。

OCARINA を用いて、協調作業型仮想空間のシミュレーションを実行可能にするためには、主に、仮想空間の設計情報をシステムに読み込む機能、仮想空間の設計情報に従って仮想空間をシミュレーションする機能、仮想空間と訓練生との間のユーザインタフェースを提供する機能等が必要である。以下でそのそれぞれの機能について要求仕様をまとめる。

要求仕様 1 仮想空間の設計情報をシステムに読み込む機能

本研究で開発する OCARINA では、OCTAVE 手法を用いて設計した仮想空間の情報を、仮想空間の設計者が予めファイルに記述しておく。そして、OCARINA がそのファイルを読み込むことで、仮想空間の設計情報をシステムに読み込む機能を実現する。その際、本研究では、OCTAVE 手法を用いて仮想空間をファイル

に記述することを可能にするために、独自のスクリプト言語を開発した。そのスクリプト言語の詳細は 4.2.2 項で述べる。

要求仕様 2 仮想空間をシミュレーションする機能

OCTAVE 手法では、クラスは内部状態を持ち、その内部状態はペトリネットを用いて管理されている。また、個々の仮想物体の間ではメッセージの交換によって、相互のインタラクションを実現する。従って、OCTAVE 手法に従って設計した仮想空間の情報を用いて、実際に仮想空間のシミュレーションを実行するためには、以下の処理を行う必要がある。

要求仕様 2-1 ペトリネットのシミュレーションによってクラスの状態を管理する処理

要求仕様 2-2 クラスの状態に応じて仮想物体の性質をシミュレーションする処理

要求仕様 2-3 クラス間のメッセージ交換の実行で仮想物体間のインタラクションを実現する処理

要求仕様 3 ユーザインタフェースを提供する機能

VR を用いてプラントの保守訓練等を実施する場合、効率的に訓練成果を上げるためには、訓練生に適切なユーザインタフェースを提供することが重要である。しかし、訓練を実施する際の最も適切なユーザインタフェースは、訓練の内容や、訓練生の習熟度、訓練を実施する際の状況に応じて大きく異なる。従って、訓練を実施する際のユーザインタフェースは柔軟に変更できることが望ましい。そこで、本研究では訓練生に提供するユーザインタフェースを、システムを再構築することなく自由に変更できる仕組みとして、プラグインを介してユーザインタフェースを提供する機能を実現する。プラグインの詳細に関しては 4.2.6 項で述べる。

また、協調作業型訓練環境では、訓練生が仮想空間内で作業できる機能を実現するだけでなく、共同作業する相手である仮想人間が、実際の人間と同様に作業できる機能を実現する必要がある。そこで本研究では、OCARINA に以下の機能を実現する。

要求仕様 4 仮想人間の動きを 3 次元アニメーションとして合成する機能

要求仕様 5 仮想人間に対して行動を指示する機能

以上に検討した機能の他に、協調作業型訓練環境として訓練の現場で使用することを考慮した場合、OCARINA は以下の特徴を備える必要がある。

要求仕様 6 安価に入手できる計算機上でリアルタイムシミュレーションを実行可能にする機能

要求仕様 7 システムに新たな機能を追加する場合にシステムを再構築する必要がない機能

本研究では、上記 7 つの要求仕様を満たす仮想空間シミュレーションシステム OCARINA を開発した。

4.2 OCARINA の概要

本節では、4.1 節で述べた OCARINA の要求機能の検討を基に開発したシステムの概要を述べる。

4.2.1 OCARINA のシステム構成

前節で述べた各種の要求機能を実現するために、本研究では図 4.1 に示す構成の仮想空間シミュレーションシステム OCARINA を開発した。OCARINA は、主に以下に示す 6 つの部分から構成される。

- OCTAVE 手法に基づいて作成したクラスの定義情報を格納するクラスデータベース
- クラスの定義を基に仮想空間をシミュレーションするシミュレーションサブシステム
- 仮想物体の動きの合成、仮想物体のインタラクション判定や、入出力の処理をする各種プラグイン
- 3 次元アニメーション映像を提示するために必要である、仮想物体の 3 次元形状データと表面の材質の模様を設定するテクスチャを格納する形状データベース及びテクスチャデータベース
- シミュレーションサブシステムから送られてくる情報を基に 3 次元アニメーションを合成するディスプレイサブシステム
- ユーザと仮想空間の間のユーザインタフェースを提供する入力サブシステム

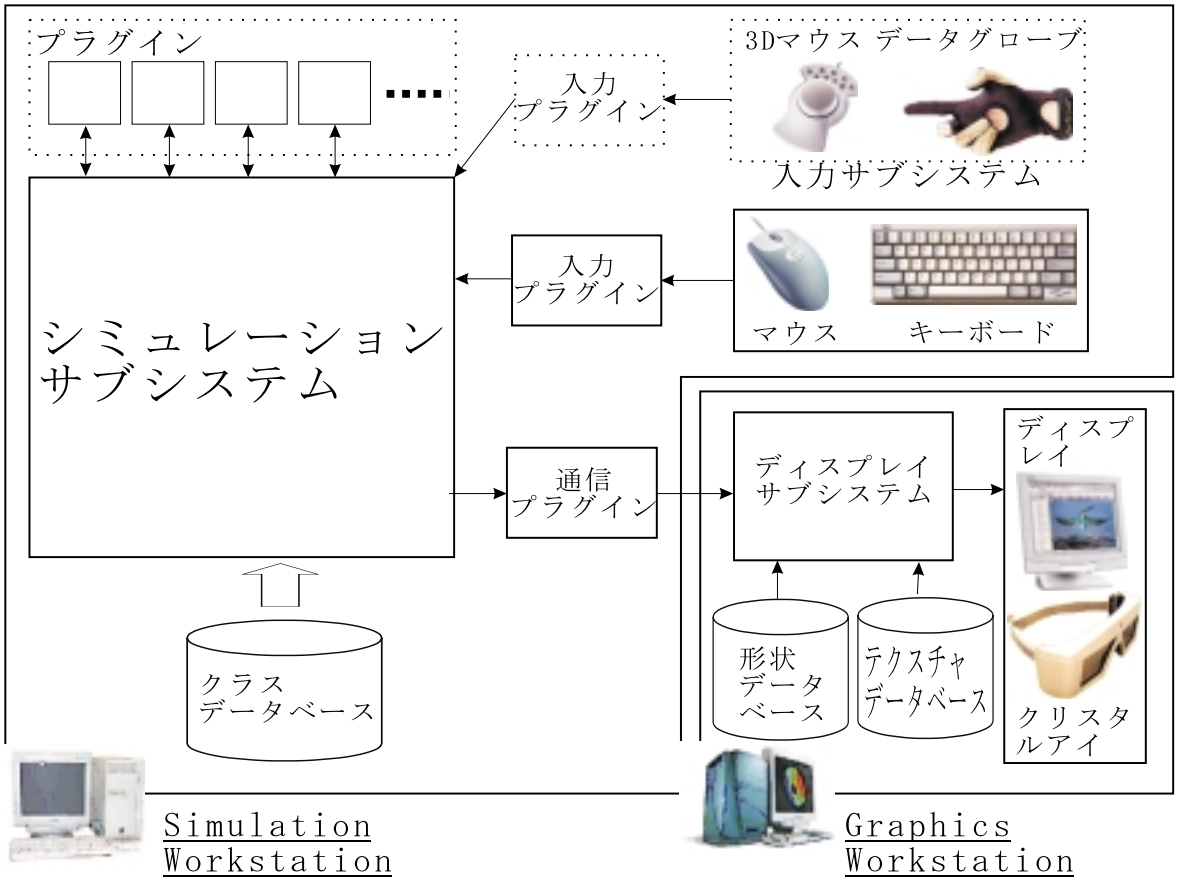


図 4.1: OCARINA のシステム構成

本研究では、4.1 節で述べた [要求仕様 6] を実現するために、上記の OCARINA の構成要素の内、シミュレーションサブシステムとディスプレイサブシステムを異なる計算機上に実装して、計算負荷の分散を試みる。シミュレーションサブシステムは数値計算の負荷が高く、ディスプレイサブシステムは 3 次元描画処理の負荷が高いため、本研究ではこの計算負荷の特性の違いも考慮して、シミュレーションサブシステムは、計算処理能力が高い Simulation Workstation(DOS/V 機:OS Linux:CPU PentiumIII 733MHz×2) 上に実装し、ディスプレイサブシステムは、描画処理能力が高い Graphics Workstation(SGI 社製 Octane:OS IRIX6.5:CPU MIPS-R12000 400MHz×2) 上に実装した。なお、ディスプレイサブシステムとシミュレーションサブシステムの間はソケット通信により、必要なデータの送受信が行われる。

次項以降では、OCARINA の構成要素のうち、クラスデータベース、シミュレーションサブシステム、プラグイン、ディスプレイサブシステムについて述べる。

4.2.2 クラスデータベース

3.3 節で述べたように、OCTAVE 手法では、個々のクラスは、1 つのペトリネット、複数の変数及び複数のメンバクラスで構成される。また、個々のクラスはオブジェクト指向の継承の機能により継承元のクラスから仮想物体の性質を引き継ぐ。このように、OCTAVE 手法を用いて仮想空間を設計する場合、ペトリネットの構造、クラスが持つ変数の種類、クラスの継承構造等を定義する必要がある。本研究では、これらの定義を仮想空間の設計者がファイルに記述することを可能にするために、クラスを定義するためのスクリプト言語を独自に設計した。クラスを定義するためのスクリプト言語は、図 4.2 に示すようにテキスト形式の記述言語であり、一般に市販・配布されているテキストエディタで記述することが可能である。なお、スクリプト言語の詳細を付録 A に示す。

OCTAVE 手法では、3.3.2 項で述べたように、仮想物体の状態に応じた性質の変化をシミュレーションするために、クラスに内部状態を持たせ、また、その内部状態に応じて仮想物体の性質をシミュレーションするために実行する処理内容を変更できる仕組を考案した。本システムでは、実行する処理の内容を OCARINA が予め備えるプログラムの関数名として指定することで、この機能を実現した。このプログラムの関数名の指定は、クラスファイルを作成する際に各クラスの内部状態毎に行われる。ここで指定する関数名は、4.2.6 項で後述するように、OCARINA にプラグインを追加することによって、システムを変更することなく柔軟に追加できる。

図 4.2: クラス定義ファイル

4.2.3 シミュレーションサブシステム

4.2 節で述べたように、シミュレーションサブシステムは、クラスデータベースからクラスの定義を読み取り、その定義に従って仮想空間をシミュレーションする。

シミュレーションサブシステムは、図 4.3 に示すように、制御部、インスタンス管理部、クラス定義読み取り部、ペトリネットシミュレーション部、プラグイン制御部から構成される。以下でシミュレーションサブシステムを構成する各部分の役割について説明する。

- 制御部
  シミュレーションサブシステム全体の処理の流れを制御する。

- クラス定義読み取り部
  4.2.2 項で説明した、スクリプト形式のクラス定義ファイルを読み取り実体としての仮想物体 (インスタンス) を作成し、インスタンス管理部に格納する。

- インスタンス管理部
  クラス定義から作成されたすべてのインスタンス情報を格納する。

- ペトリネットシミュレーション部
  ペトリネットシミュレーションによりイベントの発生及びインスタンスの状態遷移を管理する。

- プラグイン制御部
  インスタンスと、後述するプラグインとの間の情報の受け渡しを行う。

なお、シミュレーションサブシステムは、GLib ライブラリを用いて C 言語で構築した。Glib ライブラリとは、文字列やリスト等の基本的なアルゴリズムを提供する C 言語のライブラリであり、GTK(Gimp Tool Kit) プロジェクトが作成し、配布している。

4.2.4 ディスプレイサブシステム

ディスプレイサブシステムでは、シミュレーションサブシステムから受けた情報に従って、仮想空間を 3 次元映像として描画し、ディスプレイに出力する。シミュレーションサブシステムから受け取る情報は、個々の仮想物体の位置、姿勢と形状に関する情報であり、これらの情報を基に仮想物体の描画を行う。また、[要求仕様 6] で述べた

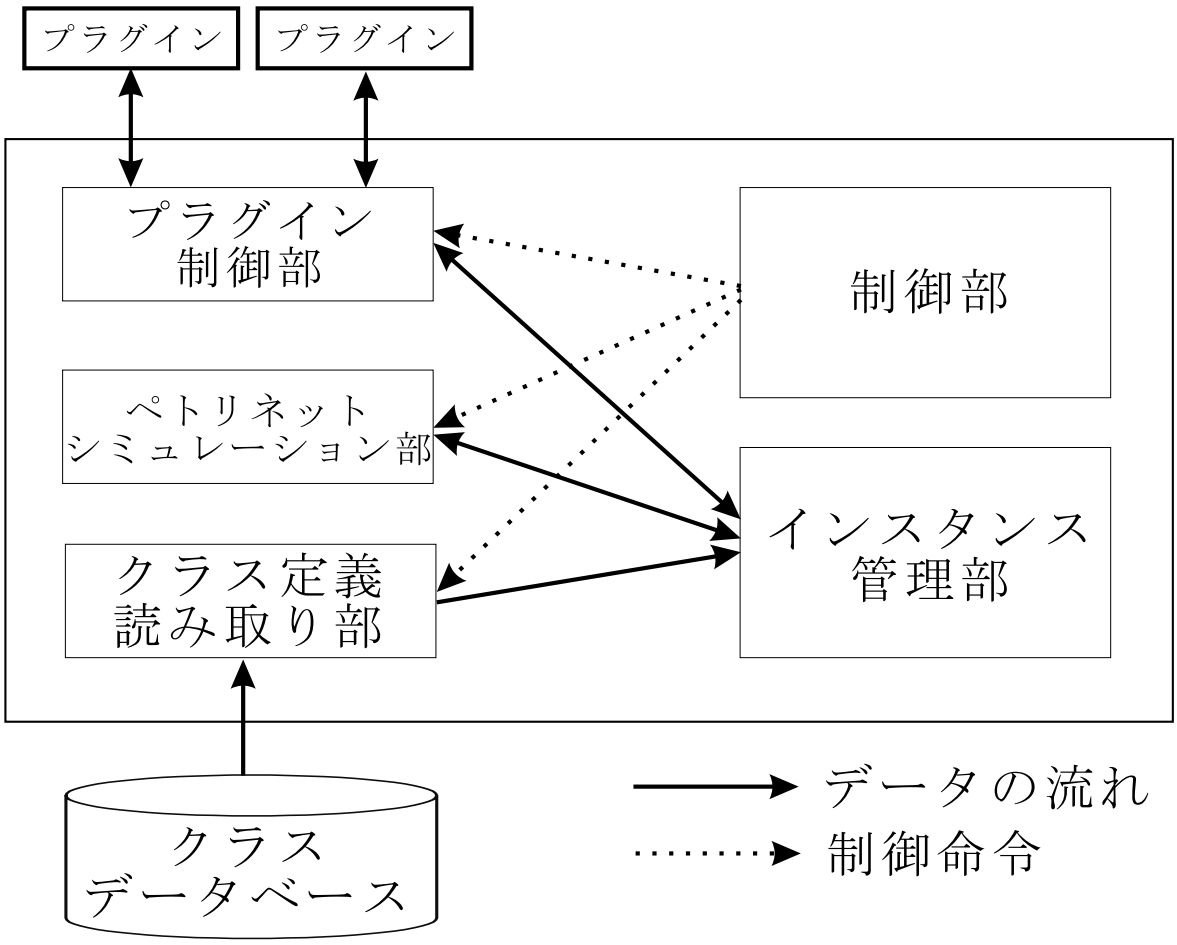


図 4.3: シミュレーションサブシステムの構成

ように、リアルタイムでシミュレーションすることが望ましい。そのため、OCARINAでは、通信による時間的遅延を減少させるため、仮想物体の位置や姿勢、形状等の情報がシミュレーションサブシステムによるシミュレーションによって更新されたときのみ、送信するという手法を採用した。

ディスプレイサブシステムは、OpenGLライブラリ用いてC言語で構築した。OpenGLライブラリは、3次元画像処理のために開発されたC言語ライブラリである。

なお、OCARINAでは、クリスタルアイズやヘッドマウントディスプレイを用いれば、表示された3次元映像で立体視することも可能である。

4.2.5 入力サブシステム

入力サブシステムは、OCARINA上でシミュレーションされている仮想空間へ、外部から何らかの働きかけ(情報入力)を行うための手段を提供する。外部から仮想空間への情報入力としては、仮想空間を体験する訓練生の動作の入力だけでなく、仮想空間内に配置する仮想人間への行動の指示や、仮想空間内への仮想物体の追加・削除等が含まれる。

具体的には、本研究では、次項で説明するプラグインを介して、仮想空間への情報入力を可能にするための仕組を構築した。これにより、OCARINAを再構築することなくプラグインを変更するだけで、仮想空間への情報入力手段を自由に変更できる。この仕組を利用すれば、例えば、以下のようなプラグインを構築することが可能である。

1. キーボードやマウスを介して仮想物体の位置・姿勢を変えるプラグイン
2. データグローブからの入力を仮想物体の位置・姿勢に反映させるプラグイン
3. キーボードから仮想人間の行動内容を指定するプラグイン
4. ネットワークを介して仮想人間の行動内容を指定するプラグイン

上記2のプラグインを構築すれば、訓練生がデータグローブを介して仮想空間内で作業することが可能になり、また、上記3のプラグインを構築すれば、訓練生が直接仮想物体を操作するのではなく、仮想人間を介して仮想空間内の物体を操作するインタフェースを実現することも可能である。さらに、仮想人間の頭脳に相当するヒューマンモデルを構築し、上記4のプラグインを介して仮想人間の行動を制御することで、仮想空間内に自律的に行動する仮想教師を構築することも可能である。

4.2.6 プラグイン

プラグインとは、一般的には、既存のアプリケーションソフトウエアを変更することなく、そのアプリケーションソフトウエアに機能を追加するためのプログラムのことである。Netscape Navigator や Photoshop などのアプリケーションで採用されており、Netscape Navigator の場合には、動画の再生機能や音声再生機能などの機能を追加することができ、Photoshop の場合には、画像の加工アルゴリズムを追加することができる。

本研究でも、Netscape Navigator や Photoshop などと同様に、プラグインを追加すれば、システムを変更することなく、OCARINA に各種機能を追加できる仕組みを構築した。OCARINA に追加可能なプラグインは実行する処理の種類によって以下の 4 種類に分類できる。

- 動作合成プラグイン
- インタラクション判定プラグイン
- 入力プラグイン
- 出力プラグイン

以下、それぞれのプラグインについて説明する。

- 動作合成プラグイン
  OCARINA では、4.2.2 項で述べたように、クラスに内部状態を持たせ、その内部状態に応じて実行する処理プログラムを変化させて、仮想物体の状態に応じた性質をシミュレーションする。動作合成プラグインは、このクラスの状態に応じて実行する処理プログラムを提供するもので、仮想物体の動きや仮想人間の動作の合成を行うプラグインである。例えば、「手に握られた状態のペン」の、手に追随する動作の合成や、「ボルトにはまった状態のナット」の、ボルトを中心にした回転運動を合成する処理等は、動作合成プラグインをシステムに追加することによって利用可能になる。

- インタラクション判定プラグイン
  インタラクション判定プラグインは、衝突などの仮想物体間のインタラクションの発生を判定し、3.3.3 項で述べた仮想物体間のメッセージ交換を実現するプラグ

インである。このプラグインは、仮想物体間で衝突等のインタラクションが発生すると、インタラクションを定義する性質クラスの特定のイベントを発生するという手法で、メッセージを送信する。

- 入力プラグイン
  入力プラグインは、4.2.5 項で述べたように、OCARINA 上でシミュレーションされている仮想空間へ、外部から情報入力する際に使用するインタフェースを提供するためのプラグインである。入力プラグインの追加で、訓練の実施状況に応じて訓練生に適切なインタフェースを提供したり、将来登場する新しいインタフェースにも柔軟に対応することが可能になる。

- 出力プラグイン
  出力プラグインは、入力プラグインとは逆に、仮想空間のシミュレーション結果をディスプレイなどの機器へ出力するためのプラグインである。出力プラグインの主なものとしては、仮想空間を 3 次元アニメーションとして、ディスプレイに出力するプラグインがある。また、協調作業型訓練システムを開発する場合には、作業の進捗状況に応じて仮想教師に教示内容を発話させる機能等を追加することも可能である。

5 章でも述べるように、現在までに、動作合成プラグインとして、仮想物体の動きを直線や弧に限定するプラグインや、仮想人間が移動する動作を合成するプラグイン、仮想人間が物を握る動作を合成するプラグイン等を開発している。また、入力プラグインとしては、キーボードを介して仮想人間に行動を指示するプラグインを、出力プラグインとしては、仮想空間を 3 次元アニメーションとして描画するプラグイン等を開発している。

4.3 OCARINA 実行時の処理の流れ

OCARINA の実行時の処理の流れを図 4.4 に示す。OCARINA は起動すると、まずクラスデータベースよりクラス定義ファイルを読み込む。次に、クラス定義ファイルの内容を基に仮想空間へ仮想物体を追加し、シミュレーションを開始する。

仮想空間のシミュレーション処理では、入力プラグインに関する処理から出力プラグインに関する処理までを、1 単位のシミュレーションとし、この単位シミュレーションを繰り返すことにより、仮想空間をシミュレーションする。

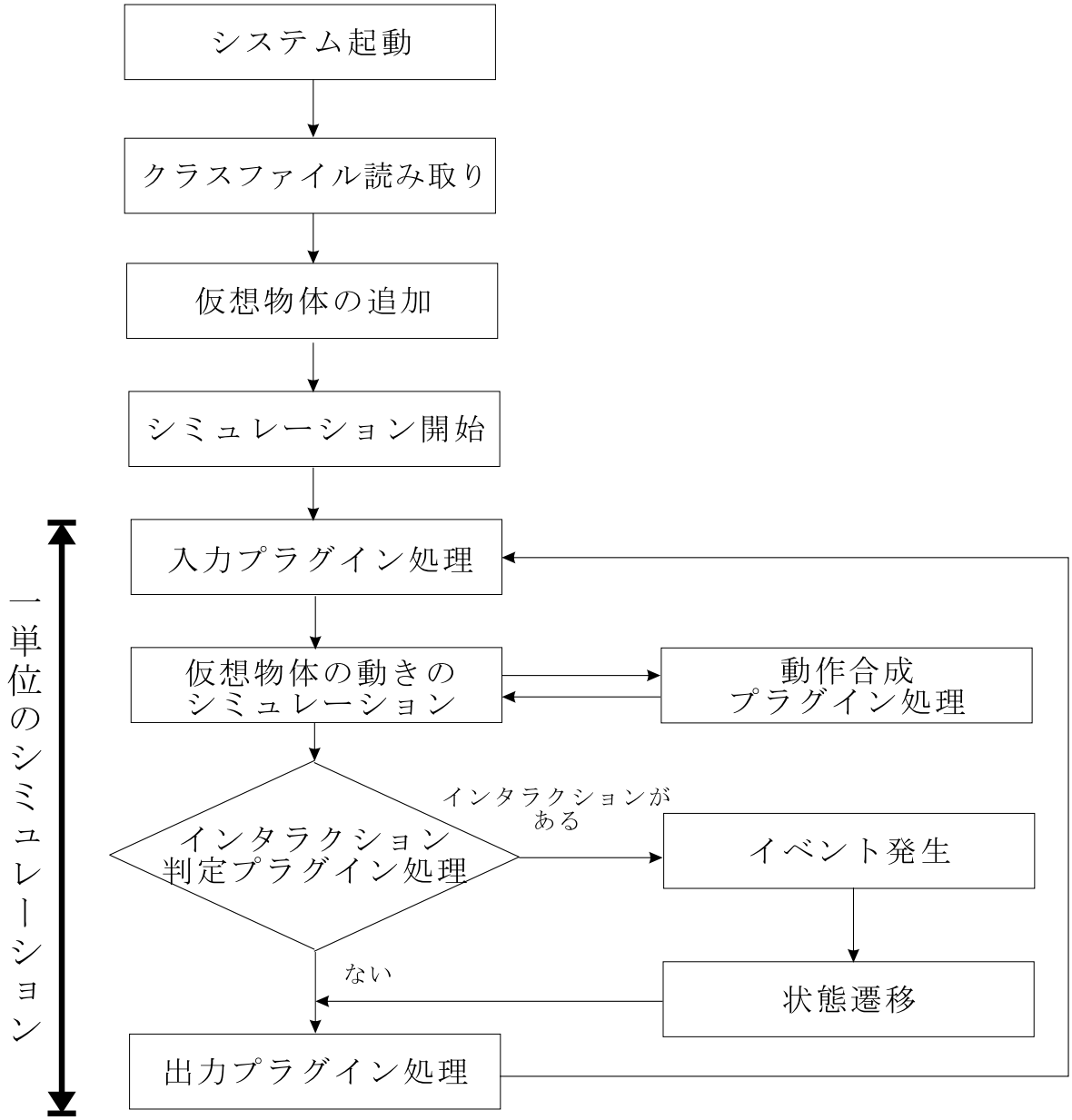


図 4.4: OCARINA 実行時の処理の流れ

単位シミュレーションでは、まず入力プラグインによってユーザからの指示の有無を判断する。指示がある場合には、適切なイベントを発生することにより、仮想物体の状態を遷移させ、仮想空間のシミュレーションに反映させる。例えば、仮想人間に対して「ペンを握る」指示が入力された場合には、仮想人間の握る動作を開始するイベントを発生させることにより、仮想人間は握る動作状態となり、仮想人間に握る動作を開始させることができる。

入力プラグインに関係する処理の終了後、すべての仮想物体に対して、仮想物体の動きを合成するためのシミュレーションを実行する。この際、必要に応じ動作合成プラグインに関する処理を実行する。

次にインタラクション判定プラグインに関係する処理を実行し、インタラクションがある場合には、プラグインがイベントを発生させ、仮想物体を状態遷移させる。

そして最後に、出力プラグインによる映像出力が行われる。

4.4 OCARINA を用いた仮想空間の構築手順

OCTAVE 手法と OCARINA を使用した場合の、仮想空間の構築手順を図 4.5 に示す。仮想空間を構築する際には、まず目的の仮想空間を構築するために必要な仮想物体の種類を検討する。必要とする仮想物体が既に定義済みなら、それを使用し、なければ、仮想物体クラスを作成する。

仮想物体クラスを作成するには、まず、仮想物体が備えるべき性質を検討する。既存の性質クラスの中で、仮想物体が備えるべき性質を定義しているものがあれば、それを仮想物体に追加することにより仮想物体クラスを作成する。なければ性質クラスを新たに作成する。

性質クラスを作成する際、性質クラスで定義している性質 (動き方等) をシミュレーションするためのプラグインがなければ、プラグインを作成する。その後、4.2.2 項で述べた独自のスクリプト言語の書式に従って、性質クラスを記述し定義する。

以上に述べたように、OCARINA を使用して仮想空間を構築する場合には、クラスとプラグインについて必要なものの内、まだ作成されていないもののみを作成し、他は既存のものを再利用することで、仮想空間を作成する。

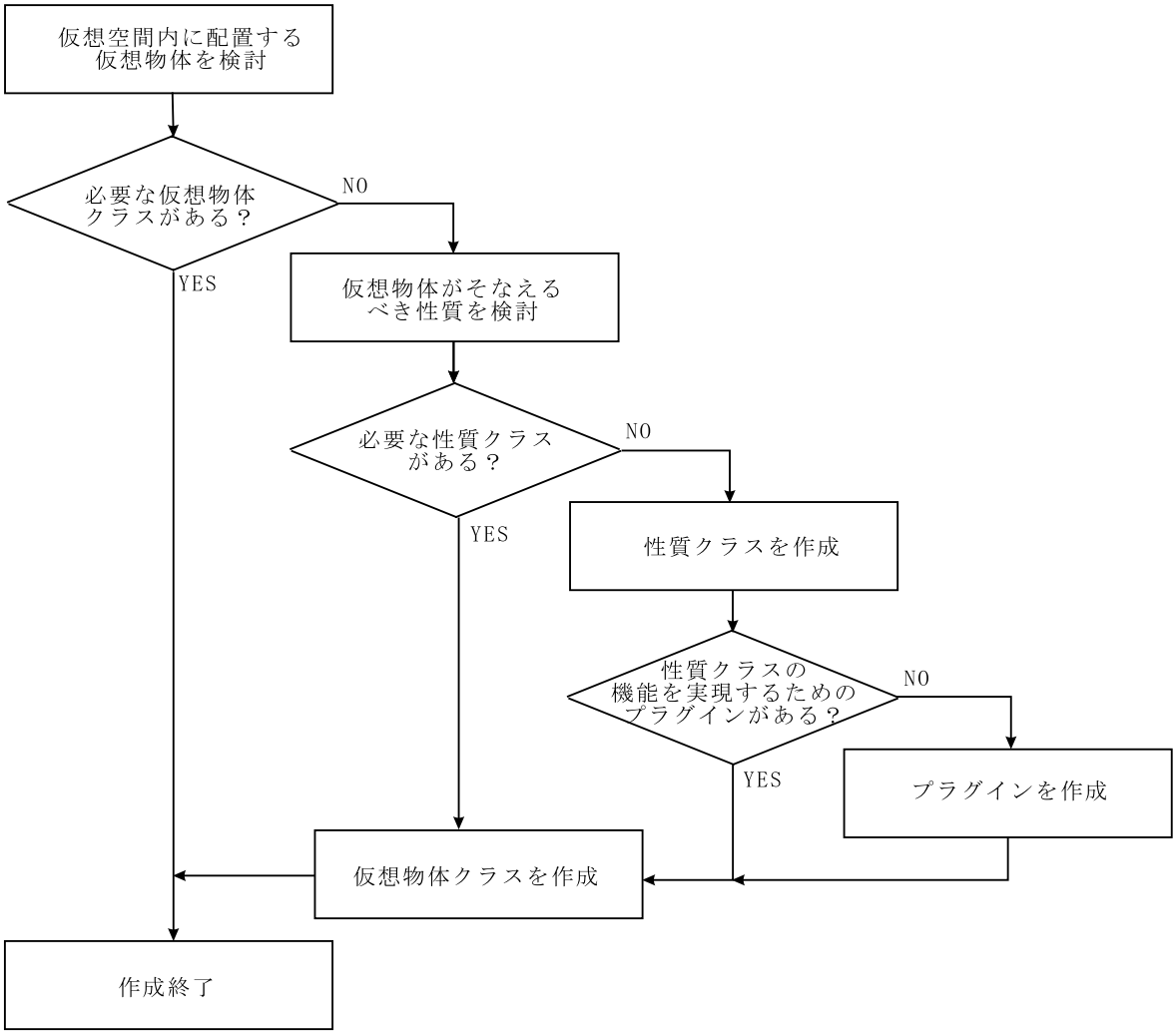


図 4.5: OCARINA を用いて仮想空間を構築する手順

第 5 章 仮想空間の構築例と評価および考察

2 章で述べたように、本研究室では、仮想作業空間内に実際の人と同じ形状をもつ仮想人間を配置し、実際の人間が訓練生として仮想空間を共有して、その仮想人間と共同作業したり、仮想人間が訓練生に作業内容を教示することができる協調作業型訓練システムの構築を目指している。本研究では、協調作業型訓練システムを実現するための要素技術の 1 つとして、少ない労力で仮想空間の構築および管理が可能な OCARINA を開発した。

本章では、まず、OCARINA を使用して協調作業型訓練システムを構築する方法について説明する。その後、協調作業型訓練システムの機能の一部の実現例として、OCARINA を用いて仮想教師が実際の人と同様の動作で作業を実演できる機能を備えた作業実演システムを構築する手法について説明する。その後、作業実演システム構築に要した作業量について考察する。最後に今後の研究の展望を述べる。

5.1 OCARINA を用いた協調作業型訓練システムの構築方法

協調作業型訓練システムは、2.2 節で述べたように、訓練生のジェスチャを認識する動作入力サブシステム、仮想空間の状況に応じて仮想人間の行動内容を決定する行動決定サブシステム、仮想人間の動作を 3 次元アニメーションとして合成する動作合成サブシステム、仮想空間の 3 次元映像を訓練生に提示する仮想空間描画サブシステムと仮想空間内の状態を管理する仮想空間管理サブシステムから構成される。

OCARINA は、4 章で述べたように、クラスデータベース、クラスの定義に従って仮想物体や仮想物体の性質の状態をシミュレーションするシミュレーションサブシステム、標準装備のプラグインとして実装される仮想空間描画サブシステム等からなる。

協調作業型訓練システムを、OCARINA を使用して構築する場合には、図 5.1 に示す構成となる。OCARINA を用いた協調作業型訓練システムの各部分の実現方法を以下に示す。

- 仮想空間管理サブシステム

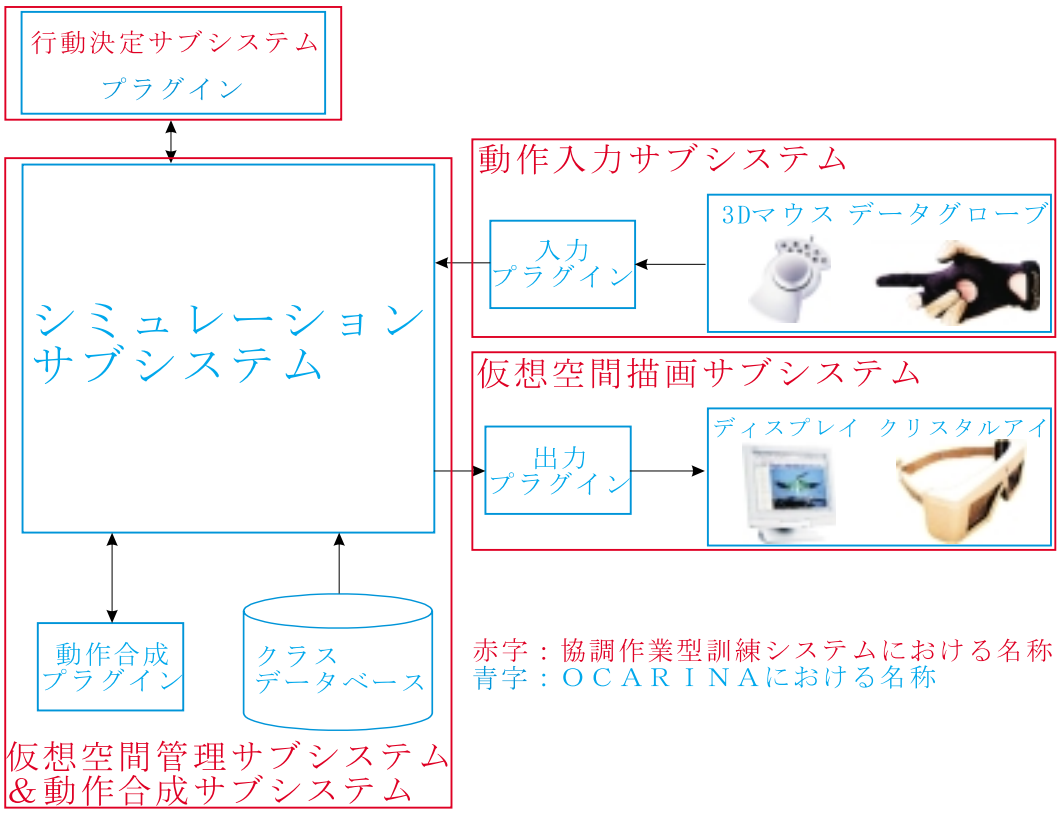


図 5.1: OCARINA で実現する協調作業型訓練システムの概要図

協調作業型訓練システムにおける仮想空間管理サブシステムは、動作入力サブシステムや行動決定サブシステムからの入力を基に、仮想物体の状態を管理し、実世界に近い仮想空間をシミュレーションする機能を担う。この機能は、OCTAVE 手法に基づいて設計された仮想空間の情報をもとに、OCARINA が仮想空間をシミュレーションする機能を用いて実現する。

- 動作入力サブシステム
  動作入力サブシステムは、4.2.6 項で述べた OCARINA のプラグインを自由に追加できる機能を用いて実現する。すなわち、データグローブや 3 次元マウスからの入力を、OCARINA 上でシミュレーションされている仮想空間内の仮想物体の位置や姿勢に反映させる入力プラグインを構築し、訓練生と仮想空間との間のインタフェースを実現する。

- 行動決定サブシステム
  入力プラグインと出力プラグインの両方の機能を備えるプラグインを構築することで実現する。すなわち、OCARINA が管理している仮想物体の状態を出力プラ

グインの機能を用いて取得し、その内容に応じて仮想人間の次の行動を決定した後に、入力プラグインの機能で仮想人間に対する行動の指示情報を OCARINA に入力する。

- 動作合成サブシステム
  動作合成サブシステムは、他の仮想物体と同様に、OCTAVE 手法に基づいて、仮想空間内に仮想人間を設計して実現する。すなわち、人間の各種性質を備えた仮想人間クラスと、仮想人間クラスが利用する動作合成プラグインを構築することによって、実際の人と同様の動きで作業を行う仮想人間を設計する。仮想人間クラスは、仮想人間がどの動作を実行中かを表す内部状態と、イベント発生に応じた内部状態の状態遷移の仕方、そして各内部状態で仮想人間の動作を合成する際に使用する動作合成プラグインの種類を設定して構築が完了する。これにより、実際の人に近い動作で行動する仮想人間を設計する。

- 仮想空間描画サブシステム
  4.2.4 項で述べた OCARINA のディスプレイサブシステムが協調作業型訓練システムの仮想空間描画サブシステムに相当する。

以上に述べたように、4 章で述べた OCARINA に必要なプラグインとクラスを追加すれば、OCARINA 自体を変更することなく、協調作業型訓練システムを構築することが可能である。

以下では、協調作業型訓練システムの機能の一部の実現例として、外部からの指示に従って、仮想人間が実際の人間と同様の動作で保守作業を行うシステムを、OCARINA を用いて構築する具体的な方法と、実際に構築した結果について述べる。

本来は、OCARINA を用いて、人と仮想人間が共同作業可能な協調作業型訓練システムの全体を構築することが望ましいが、本研究では人が仮想空間に参加できる機能の構築は将来課題とし、上記のように、外部からの指示に従って、仮想人間が実際の人間と同様の動作で保守作業を行うシステムを試作する。

5.2 OCARINA を用いた仮想空間の構築例

本研究では、協調作業型訓練システムの一部の機能の実現例として、2.2 節で述べたシステム構成の内の動作合成サブシステムに相当するシステムを OCARINA を用いて実際に構築し、それを用いて仮想人間が仮想空間内で作業を実演する作業実演システ

ムを構築した。本節では、まず構築の対象とする作業実演システムの概要について述べる。次に構築の対象とする仮想空間内に配置する仮想物体と仮想人間、及び、それらを構成するクラスについて説明し、最後に作業実演システムを実現するために必要となるプラグインについて説明する。

5.2.1 構築する仮想空間の概要

5.1 節で説明したように、OCARINA を用いて協調作業型訓練システムを構築するためには、動作入力サブシステム、行動決定サブシステム、動作合成サブシステムを構築する必要がある。しかし、行動決定サブシステムを構築するためには、仮想人間が自律的に次に行う動作を決定するために、仮想人間の頭脳にあたる AI モデルを作成する必要がある。しかし、本研究では、仮想空間を容易に構築可能にすることを目的としているので、仮想人間の頭脳にあたる部分は構築せず、仮想人間の行動の決定は、行動決定サブシステムの代わりに、訓練生が行うこととし、訓練生の指示に従って、仮想人間が作業を実演する作業実演システムを試作する。

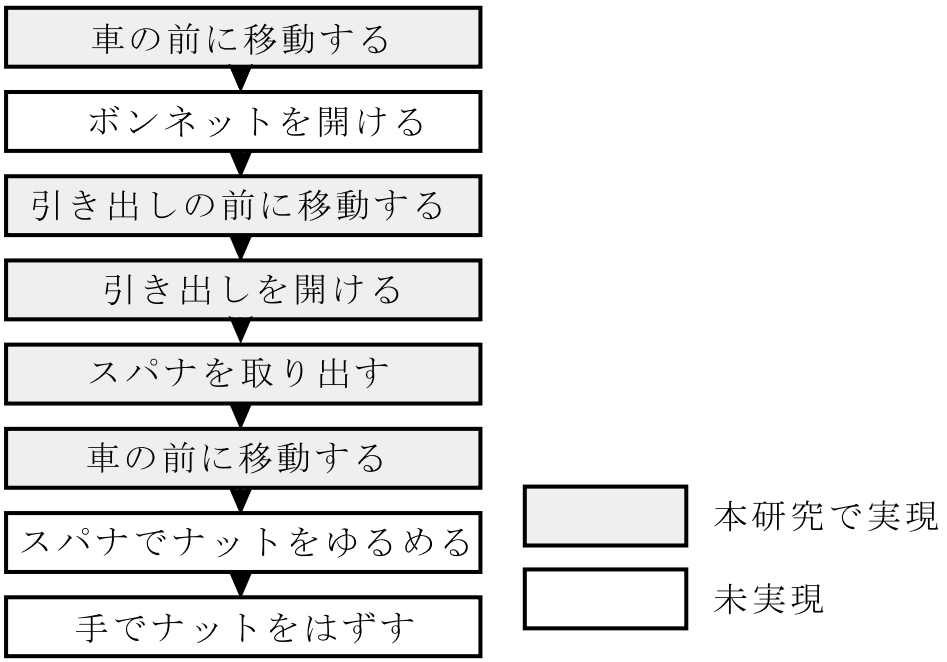


図 5.2: 車のメンテナンス作業の流れ

構築する作業実演システムの訓練対象は、図 5.2 に示す手順で作業する車メンテナンス作業とする。本研究では図に示した作業の内、

- 作業対象の仮想物体の前まで移動する。

- 引き出しを開ける。
- スパナを手に持つ。

の 3 つの動作を仮想人間が実演できるシステムを、OCARINA を用いて構築した。

この作業実演システムを構築するにあたり必要となる仮想物体を表 5.1 に示し、仮想空間内での配置を図 5.3 に示す。

表 5.1: 作業実演システムの仮想空間を構成する仮想物体の一覧

	仮想物体名		仮想物体名
1	車	6	ボルト
2	ボンネット	7	机
3	スパナ	8	引き出し
4	ナット	9	たんす
5	バッテリー		

5.2.2 仮想人間の設計

本項では、まず、仮想空間内に配置する仮想人間の動作を合成する手法について述べた後、仮想人間のクラス構成について述べ、最後に仮想人間の動作を合成する際の処理の流れを述べる。

5.2.1 項で述べたように、作業実演システムにおける仮想人間は、訓練生からの指示を基に行動する。

このため仮想人間は、次のような仕様を満たす必要がある。

仕様 1 OCTAVE 手法に基づいて定義できる

仕様 2 訓練生の指示に従って行動する

仕様 3 訓練作業に必要となる動作を合成できる

仕様 4 リアルタイムに動作を合成できる

作業実演システムでは、上記 2.〜4. の仕様を満たす仮想人間を設計する手法として、本研究室の先行研究で市口によって提案された人体モーション合成手法 [11] (以下では

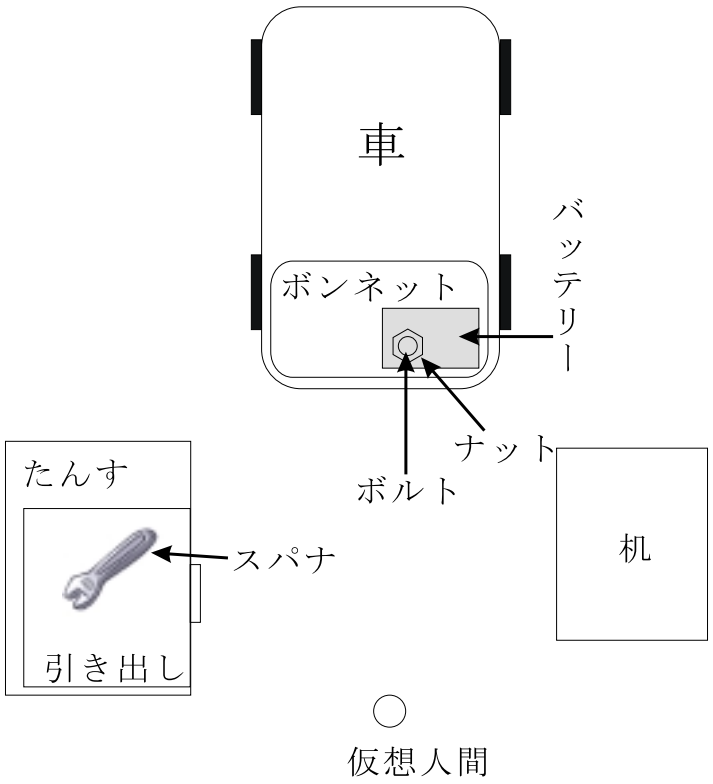


図 5.3: 仮想物体の配置

市口手法とする) を参考にする。さらに、市口手法を改良して OCTAVE 手法に適用することにより、作業実演システムの仮想人間を設計する。以下では、まず市口手法の概要について説明した後、市口手法の OCTAVE 手法への適用方法について説明する。

市口手法の概要

市口手法は、仮想人間の動作を合成するために、認知科学や人工知能の分野で注目されている、“アフォーダンスの概念” を採り入れたものである。アフォーダンスとは、知覚心理学者であった James J.Gibson により提唱された概念であり、「観察者との関係で存在する環境の特性であり、環境が観察者に提供するもの」と定義される、観察者に内在するのではなくて環境自身に内在する情報を指している。このアフォーダンスの概念に従えば、「人の動作は環境に内在する情報に誘発されて生じるものであるとされ、人があらかじめ備えている環境のモデルに従って行動しているのではない」とされる。

市口手法は、この概念を参考にし、仮想人間の動作を合成する際に必要となる各種情報 (アルゴリズムやアルゴリズムを働かせるために必要なデータ (以下では動作情報

とする)) は、仮想人間自身に定義するのではなく、動作対象の仮想物体に内在させ、必要に応じて、それらの情報を仮想物体から仮想人間に伝達し、仮想人間の動作を合成する仕組みを新たに提起したものである。

この手法は、仮想空間に新たに仮想物体を追加する際に、仮想人間自身に動作情報を定義しなおさなくてはならない場合に比べて、仮想人間の定義を変更することなく、その仮想物体に対する動作を合成できる利点がある。

市口手法の OCTAVE 手法への適用方法

市口手法では、仮想人間の動作に関する各種情報はすべてその動作対象の仮想物体に持たせるので、仮想物体が存在しない場合には、仮想人間は仮想物体に対する行動を実行できない。すなわち、仮想人間は仮想物体に対して依存関係を持つ。

このため、市口手法では、仮想人間を新たに定義し、仮想空間に追加する場合に、その仮想人間の動作を合成するためのアルゴリズムを、既存の仮想物体全部に追加しなければならなかった。例えば、健常な仮想人間が歩く動作を合成するアルゴリズムが仮想物体に既に内在しているとする。ここで、新たに足の不自由な仮想人間を仮想空間に追加した場合に、従来の歩くアルゴリズムでは、足の不自由な人の動きは合成できない。そこで、従来から仮想空間に存在するすべての仮想物体に、新たに足の不自由な人の動きを合成するアルゴリズムを定義する必要がある。

本研究では、このような市口手法に存在する問題点を解決するために、仮想物体の動作を合成するために必要な各種情報の内、アルゴリズムと動作情報を分離し、仮想人間自身の動き方を決定するアルゴリズムは仮想人間自身が、仮想人間の動作を誘発する動作情報は仮想物体が持つこととする。そして、動作情報をメッセージを使用して、仮想人間に渡すことで、仮想人間は自分の持っているアルゴリズムを使用して行動する。

例えば、移動する動作を合成するアルゴリズムとして、健常者の仮想人間は通常の歩行アルゴリズムを持ち、足の不自由な仮想人間は車椅子で移動するアルゴリズムを持つとする。また、歩く動作を誘発する仮想物体は、「移動する」という動作情報のみを保有し、この動作情報をメッセージを使用して仮想人間に送るとする。そして、仮想人間は動作情報に関するメッセージを受け取ると、自分自身が持つアルゴリズムを使用して、歩く動作を合成する。すなわち、動作情報に関するメッセージを受け取ると、健常者の仮想人間は通常の歩行動作を行い、足の不自由な仮想人間は車椅子で移動することとなる。このように仮想人間に動作を合成するためのアルゴリズムを仮想

物体に、仮想人間の動作を誘発する動作情報を仮想物体に定義することにより、仮想物体と仮想人間は、互いに依存関係がなくなり、OCTAVE 手法で定義することが可能となる。

人体モデルの構成

作業実演システムでは、仮想人間として上半身のみの人体モデルを使用する。これは、本訓練システムで想定する作業は、すべて上半身のみで行うことができる作業であり、下半身は必要ないためである。

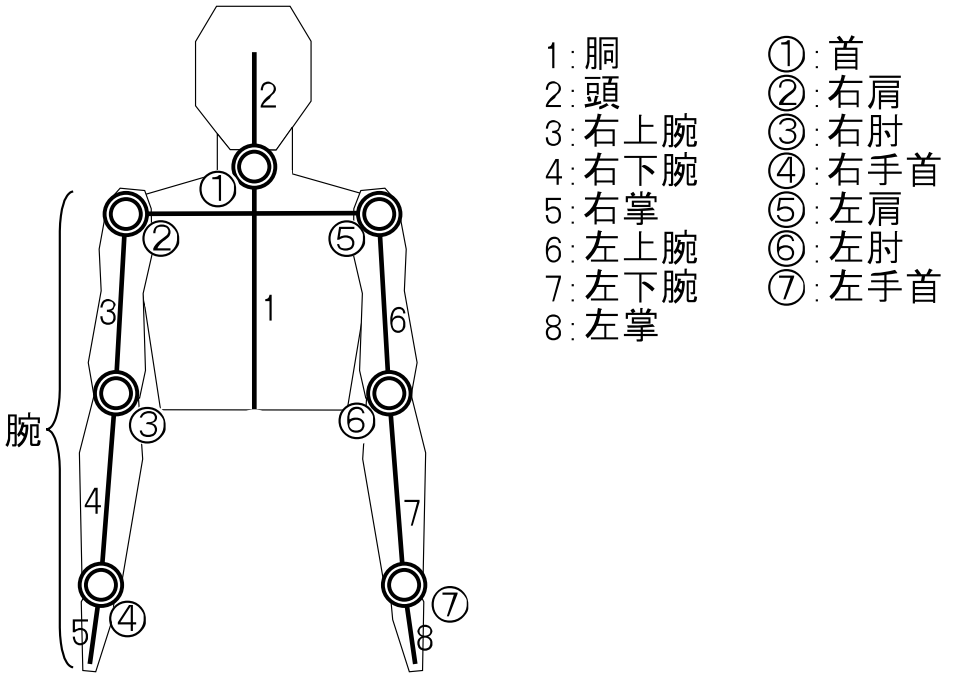


図 5.4: 人体モデルと各部名称

作業実演システムで用いる仮想人間の人体モデルの構成を図 5.4 に示す。このモデルでは、人の上半身の動きを表現するのに最低限必要と思われる 7 個の関節と 8 個の部位 (リンク) から構成されている。図中、二重丸 (◎) が関節の位置を示す。それぞれの関節では、接続されている 2 つのリンクのうち、一方のリンクが他方のリンクに対して x, y, z 軸周りの回転の 3 つの自由度を持つ。

ここで、このモデルに図 5.5 に示すような、胴を頂点とする階層構造を設定する。このような階層構造を設定することにより、ある階層の姿勢を 1 つ上の階層から見た相対的な姿勢として表すことが可能となる。

また、各リンクには、図 5.6 に示すように、上位リンクとの接続部分である関節を原

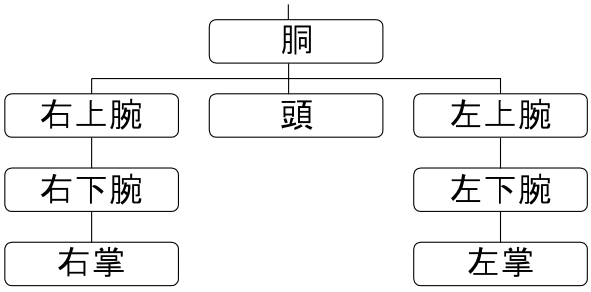


図 5.5: 人体モデルのリンクの階層構造

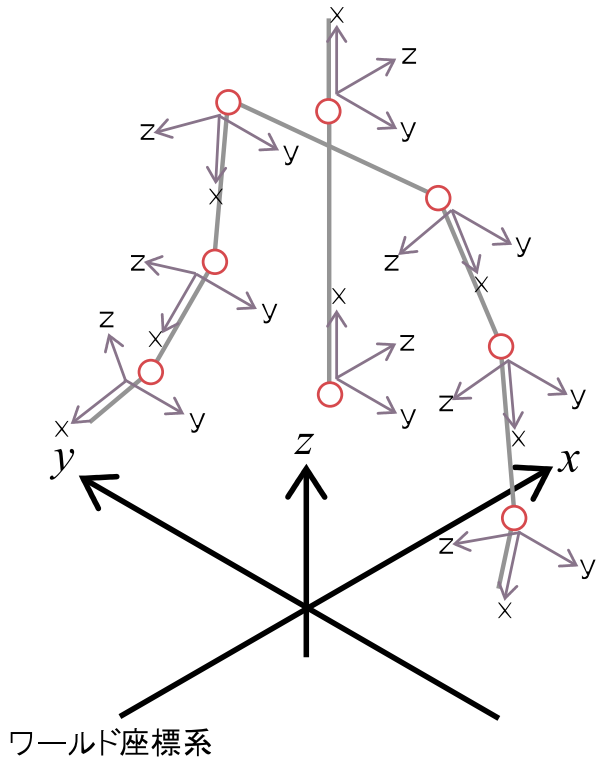


図 5.6: ワールド座標と各リンクに設定したローカル座標

点とし、そのリンクに固定されるローカル座標系を設定する。また、ローカル座標系とは別に、上方向を z 軸とするワールド座標系を設定する。

このように、人体モデルに階層構造を設定し、各リンクにローカル座標系を設定することによって、人体モデルの姿勢を、ワールド座標系から胴のリンクのローカル座標系への変換と、隣接する 2 つのリンクのローカル座標系の変換の合計 8 個の変換を用いて記述することが可能となる。

仮想人間のクラス構成

作業実演システムで設計する仮想人間は図 5.2 で示した動作を、すべて合成できる必要がある。これらの動作は、

1. 目的地点まで移動する動作
2. 腕を伸ばして対象物を触る動作
3. 手で対象物を握る動作
4. 手を特定の軌跡に沿って動かす動作

の 4 つの動作を組み合わせて、すべての動作を合成することが可能である。

例えば、引き出しを開ける動作は、1. の動作で引き出しの前まで移動し、2. の動作で腕を伸ばして取っ手を触り、3. の動作で取っ手を握り、4. の動作で引き出しが開く方向の直線に沿って手を動かすことで合成できる。

作業実演システムでは、前述のように、仮想人間は動作対象の仮想物体から動作情報をメッセージとして受け取り、その動作を仮想人間自身に定義されているアルゴリズムで合成する。

作業実演システムでは、実際には、1.〜3. の動作は、「対象物の前まで移動し、腕を伸ばして対象物を握る」という１つの動作として扱う。このため、仮想物体が仮想人間に対して送る動作情報は、

1. 握る
2. 手を軌跡に沿って動かす

の 2 つだけで、仮想人間にすべての動作を合成させることが可能である。

すなわち、仮想人間は、「握る」という動作情報を含んだメッセージを受け取ると、そのメッセージの送り元の仮想物体に対し、仮想物体の前まで移動し、腕を伸ばして対

象物を触り、手で対象物を握るという一連の動作を合成する。また、「手を軌跡に沿って動かす」という動作情報を含んだメッセージを受け取ると、メッセージで与えられた軌跡に沿って手を動かす動作を合成する。

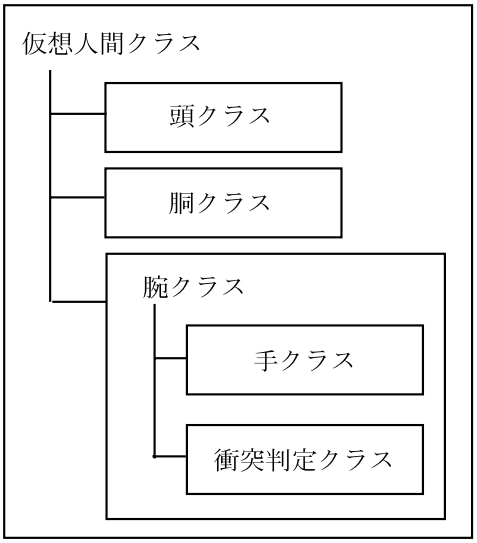


図 5.7: 仮想人間のクラス構成

以上に述べた仕様を実現するための仮想人間のクラス構成を図 5.7 に示す。仮想物体クラスである仮想人間クラスに、頭、胴、腕の動きを合成する性質を定義された性質クラスである、頭クラス、胴クラス、腕クラスがメンバクラスとして属し、さらに腕クラスには衝突判定クラスと手クラスが属する構造となる。

頭クラスは、仮想人間が特定の仮想物体の方向を見る動作を合成する性質を提供する。実際の人間は作業する際に、作業の対象物を見て行動するのが自然であり、頭クラスは、仮想人間の動作が自然に見えるようにするために必要である。胴クラスは、仮想人間の移動する動作を合成する性質を提供する。腕クラスは、仮想人間が腕を伸ばして仮想物体に触る動作と手を特定の軌跡に沿って動かす動作を合成する性質を提供する。仮想人間クラスは、動作情報のメッセージを受け取るイベントを持ち、そのイベントの発生に伴って、メンバクラスのイベントを発生させることにより、動作を開始させる。

図 5.8〜図 5.12 に頭クラス、胴クラス、手クラス、腕クラス、仮想人間クラスのクラスの定義を示す。図中で、メンバクラスのイベントから親クラスのイベントへの矢印が引かれている場合、メンバクラスのイベントが発生すると、連続して親クラスのイベントが発生することを示す。また、親クラスからメンバクラスへ矢印が引かれている場合、親クラスのイベントが発生すると連続してメンバクラスのイベントが発生す

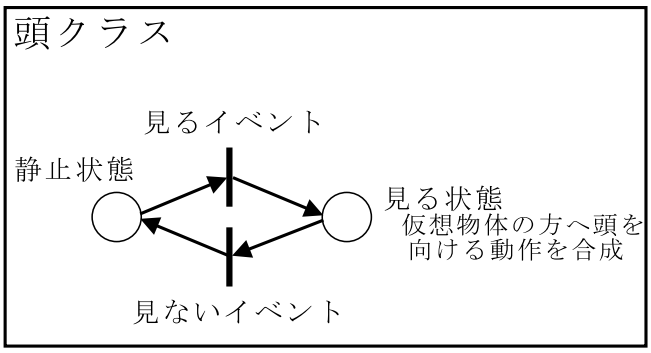


図 5.8: 頭クラスの定義

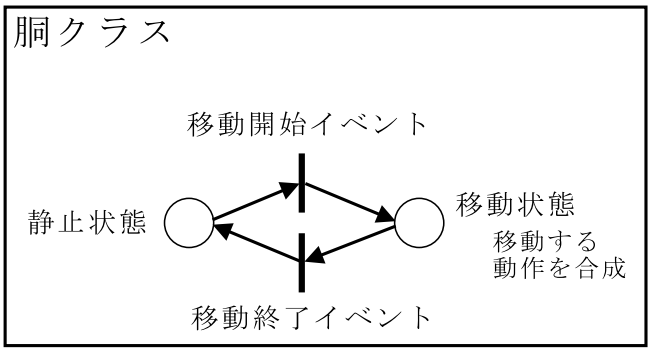


図 5.9: 胴クラスの定義

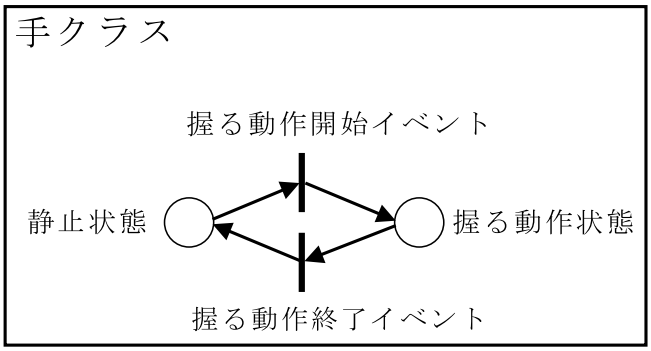
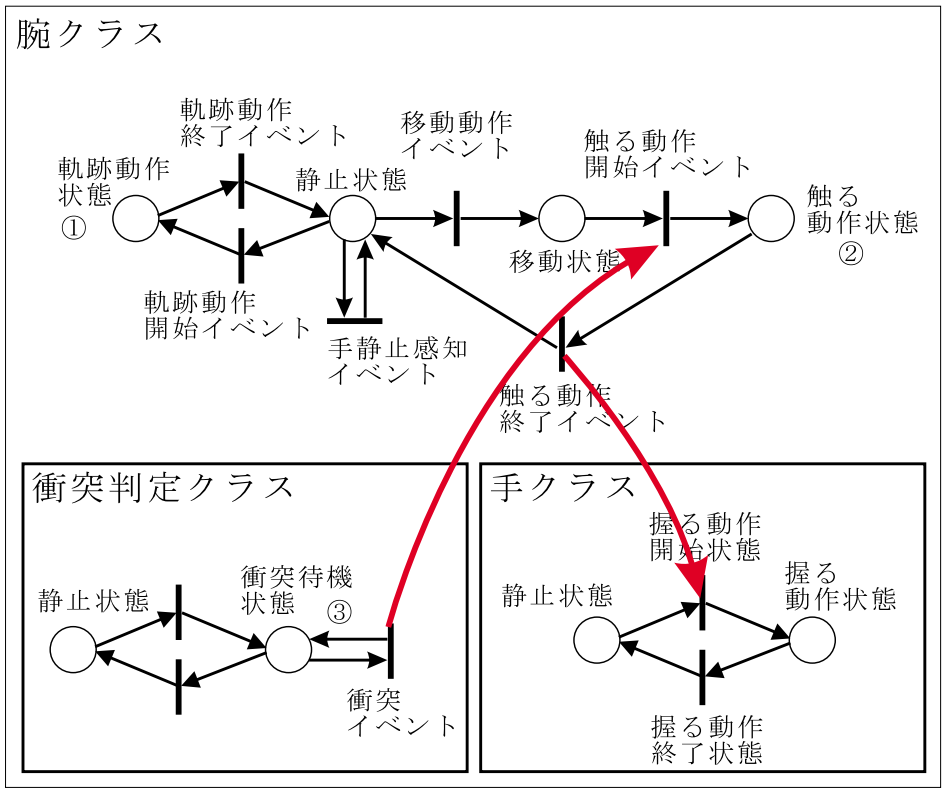


図 5.10: 手クラスの定義



①…手を軌跡に従って動かす動作を合成
②…腕を伸ばして仮想物体に触る動作を合成
③…特定の仮想物体との衝突判定を行う

図 5.11: 腕クラスの定義

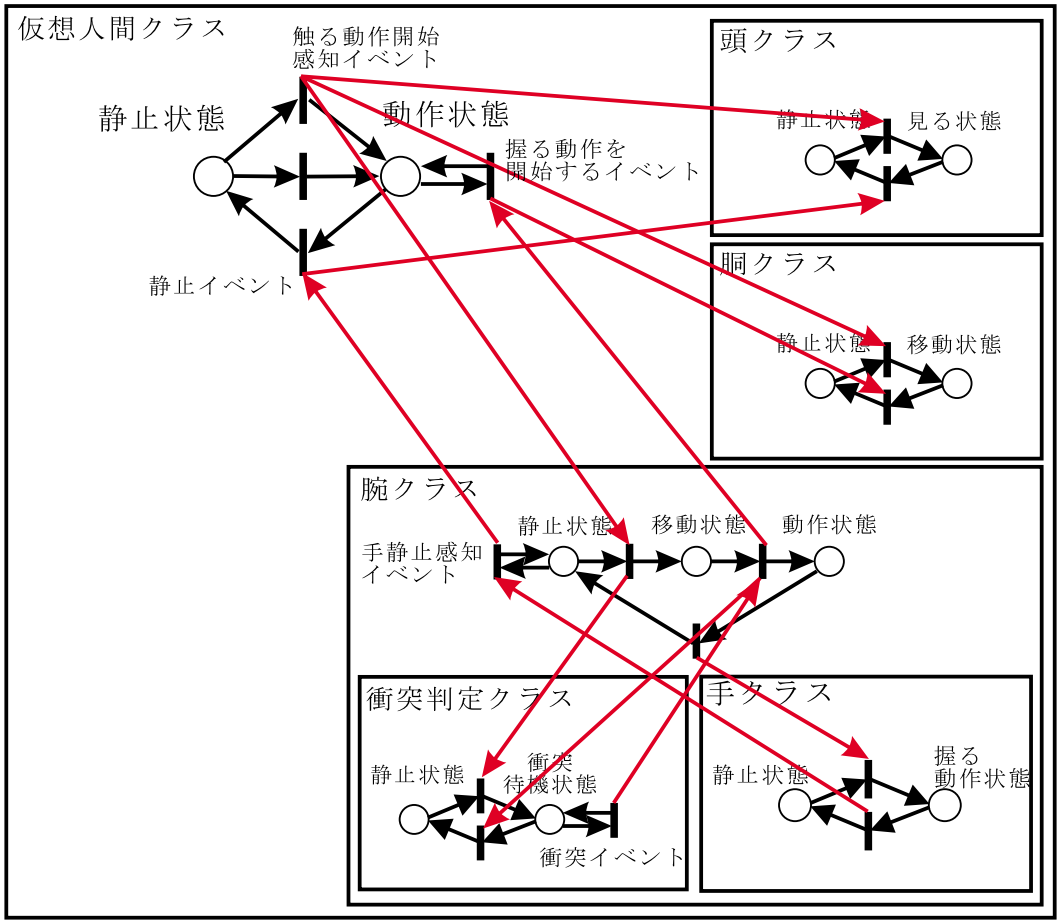


図 5.12: 仮想人間クラスの定義

ることを示す。また、仮想人間クラスは、人体モデルの姿勢を記述するために、ワールド座標系から胴のリンクのローカル座標系への変換と、隣接する 2 つのリンクのローカル座標系の変換の合計 8 個の変換を変数として持つ。

仮想物体を握る動作のシミュレーション

次に、仮想人間が仮想物体を握る動作が、上記の定義に従って、どのようにシミュレーションされるのかを説明する。なお、上記の仮想人間の定義では、仮想物体の性質を定める段階であり、実体が存在しないので、一般のオブジェクト指向の用語の使用方法に従い「クラス」という言葉を使用したが、以下の仮想空間のシミュレーション時の説明では、クラスを用いて仮想物体を設計した後の、実体としての仮想物体間の関係であるため、「インスタンス」という言葉を使用する。

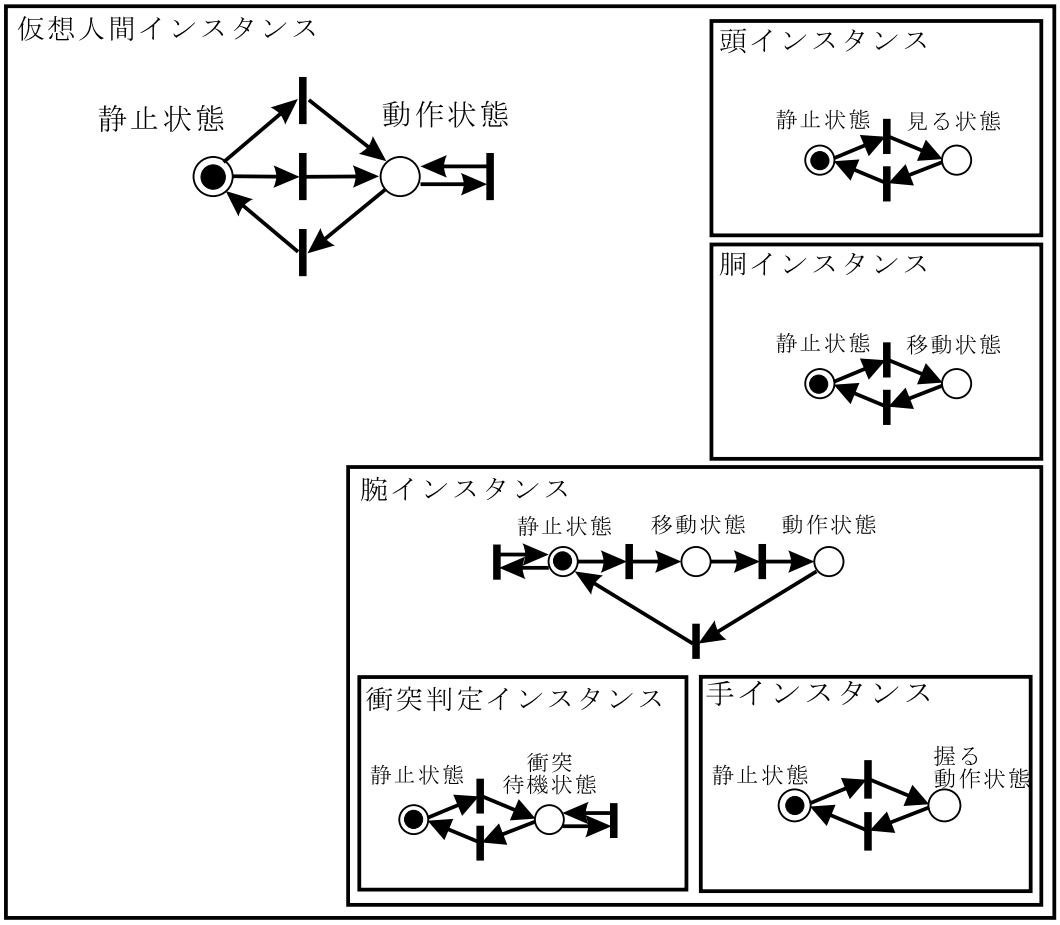


図 5.13: 握る動作シミュレーション (静止状態)

図 5.13 に示すように、仮想人間の初期状態として、すべてのインスタンスは「静止状態」であり、仮想人間を構成する各部位は、静止しているとする。仮想物体から「握

る」という動作情報を含んだメッセージがシステムを通して仮想人間に送られると、仮想人間インスタンスの「握る動作を開始するイベント」が発生する。この時、仮想人間インスタンスの「握る動作を開始するイベント」の定義に従って、胴インスタンスと腕インスタンスの「移動開始イベント」と頭インスタンスの「見るイベント」が発生する。さらに腕インスタンスの「移動開始イベント」の定義に従って、衝突判定インスタンスの「衝突判定開始イベント」が発生する。これらの一連のイベントの発生に伴って、仮想人間の状態は図 5.14 に示す状態に遷移する。この際、胴インスタンスは、「移動状態」で仮想物体に向かって移動する動作を合成し、頭インスタンスは、「見る状態」で仮想物体の方向を向く動作を合成する。以上の処理により、握る対象の仮想物体の方向を向き、その仮想物体の方向へ移動するという仮想人間の動作を実現する。

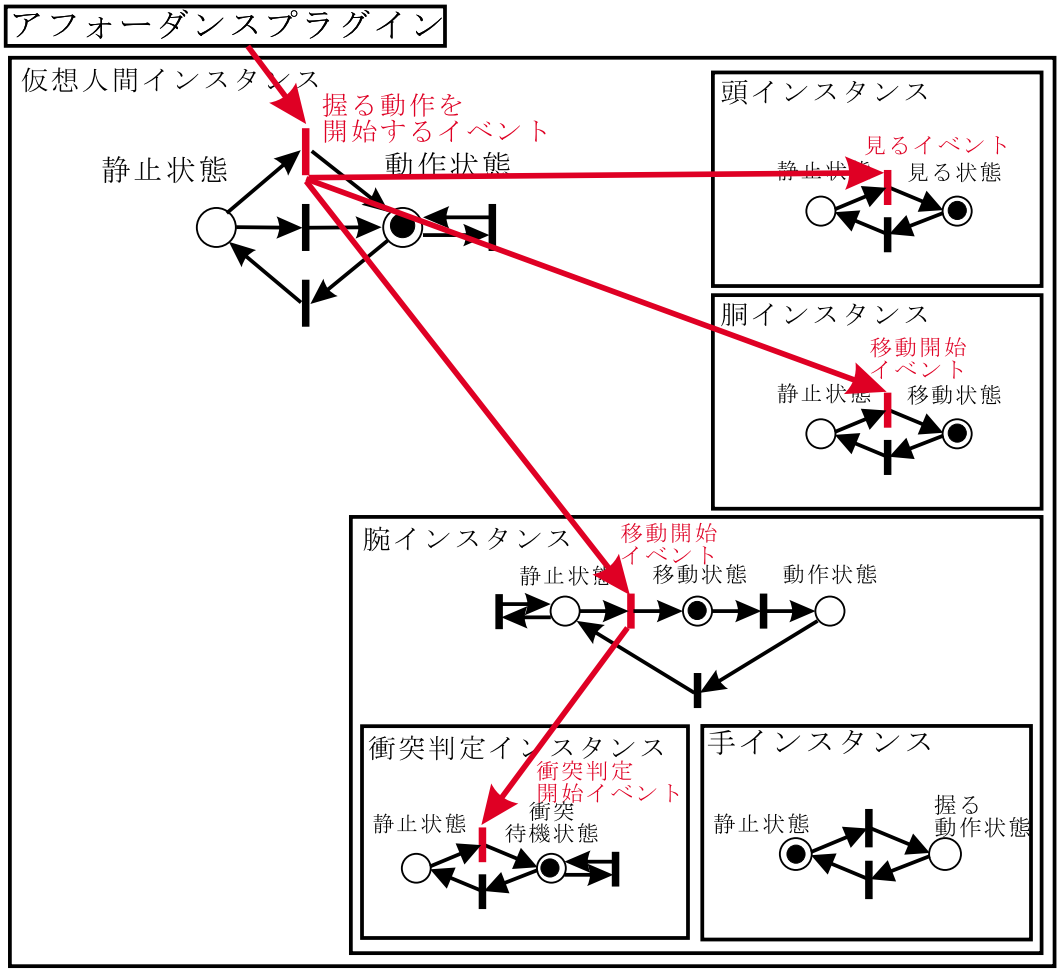


図 5.14: 握る動作シミュレーション (移動状態)

その後、仮想人間の手が仮想物体に届く位置まで移動すると、システムは、衝突判定プラグインの処理により仮想人間の手が届く範囲の性質を表すインスタンスと仮想

物体が衝突したことを判定し、仮想人間と仮想物体の双方にメッセージを送る。このメッセージにより、衝突判定インスタンスの衝突イベントが発生し、腕インスタンスの「触る動作開始イベント」が、その定義に従って発生する。さらにあらかじめ定めておいたイベント伝搬の定義に従って、仮想人間インスタンスの「触る動作開始感知イベント」と、胴インスタンスの「移動終了イベント」が発生する。この一連のイベントの発生により、仮想人間の状態は図 5.15 に示す状態に遷移する。この時、胴インスタンスは「静止状態」になり、仮想人間は静止する。そして、腕インスタンスが「触る動作状態」となり、腕を仮想物体の方向へ伸ばしていく動作を合成する。

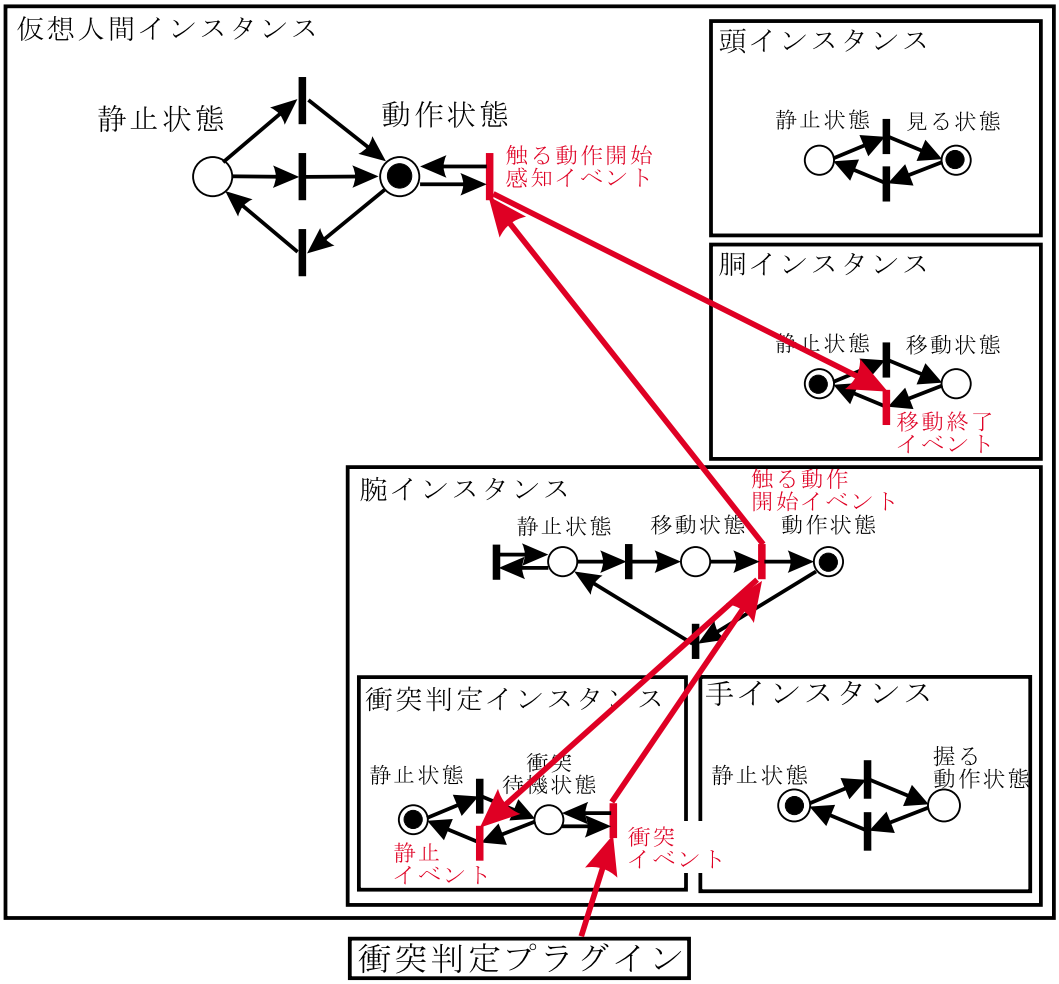


図 5.15: 握る動作シミュレーション (触る動作状態)

次に、この状態で仮想物体に手が触れる位置になると、腕インスタンスの「触る動作終了イベント」が発生にする。するとこのイベントの定義に従って、手インスタンスの「握る動作開始イベント」が発生する。これらのイベントの発生により、仮想人間の状態は図 5.16 に示す状態へ遷移する。この時、腕インスタンスは「静止状態」へ

遷移し、手インスタンスは「握る動作状態」となり、手が仮想物体に触れている状態で、指を曲げて仮想物体を握る動作を合成する。

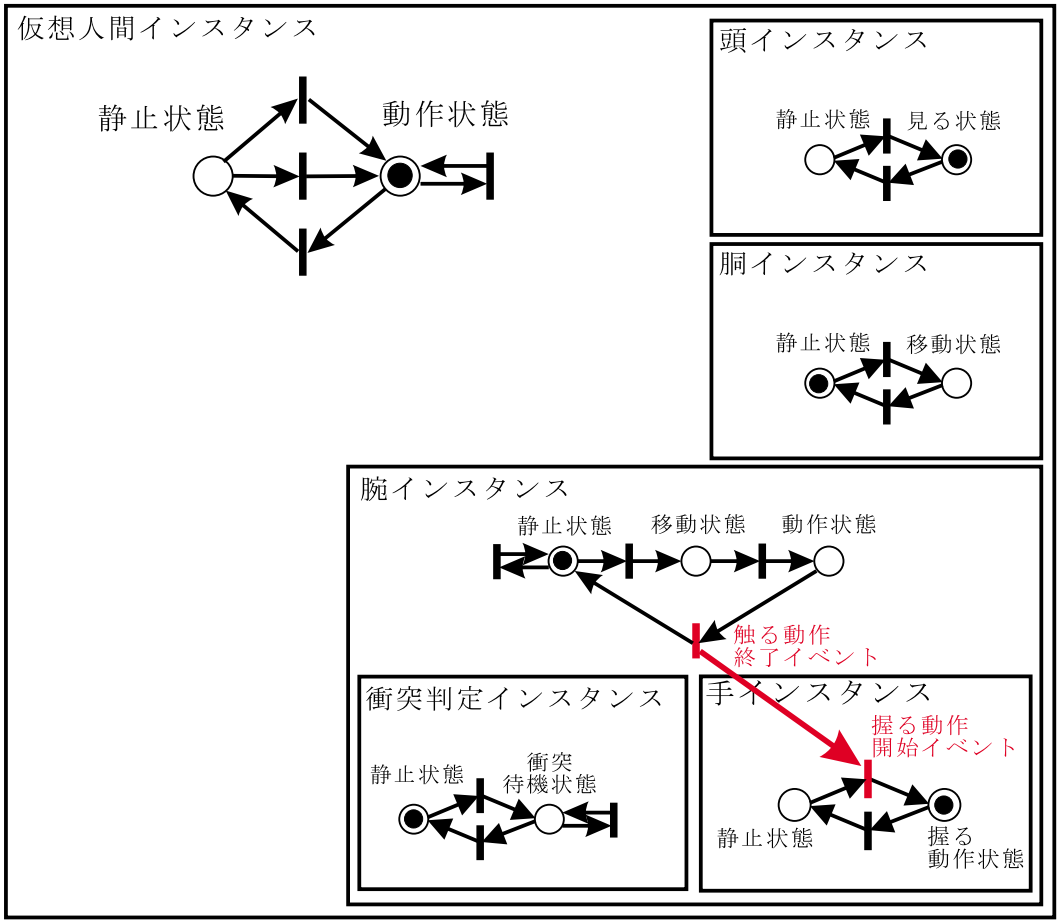


図 5.16: 握る動作シミュレーション (握る動作状態)

指を曲げ終わると、手インスタンスの握る動作終了イベントが発生し、各イベントの定義に従い、腕インスタンスの「握る動作終了イベント」、仮想人間インスタンスの「静止イベント」、頭インスタンスの「動作終了イベント」が発生し、仮想人間の状態は図 5.13 に示す状態に復帰し、仮想人間が静止し、一連の「握る」動作のシミュレーションが終了する。

5.2.3 仮想物体の設計

本項では、作業実演システムの仮想空間内に配置する仮想物体を設計する方法について述べる。

仮想物体とその訓練環境における性質

作業実演システムにおける仮想物体の動きをシミュレーションするためには、

1. 仮想物体の動きを合成する処理

2. 他の仮想物体とのインタラクションを感知する処理

3. 仮想物体の形状を 3 次元で描画する処理

4. 仮想人間に対して、動作を誘発する動作情報をメッセージとして送る処理

という 4 つの処理を定義する必要がある。作業実演システムでは、これらの処理を仮想物体の性質と捉え、個々の処理を 1 つの性質クラスに定義する。そして、仮想物体クラスにこれらの性質クラスを追加することにより、仮想物体がこれらの処理を実現できるようにする。

例えば引き出しの場合は、仮想人間に持たれて引っ張られることにより、引き出しが動き得る方向に直線運動する。そのため、1. の仮想物体の動きを合成する処理として、仮想人間の手に追随して移動する動きを合成する処理と、引き出しが動き得る直線から離れないように、引き出しの位置を直線上に限定する処理が必要であり、それぞれ、「被把持クラス」と「直線動作クラス」に定義する。また、2. の他の仮想物体とのインタラクションを感知する処理としては、手に握られたことを感知する処理が必要であり、「衝突判定クラス」に定義する。3. の性質は、訓練生に提示する仮想物体の 3 次元形状であり、「形状クラス」を定義する。

また、5.2.2 項で述べたように、動作対象の仮想物体は、仮想人間のこれらの動作を誘発するために動作情報をメッセージを使用して仮想人間に送信する必要がある。仮想人間は、引き出しに対して「握る」という動作と、引き出しが動く方向の「直線の軌跡に従って手を動かす」という 2 つの動作を行う。このため、引出しは、仮想人間に対して、「握る」という動作情報と、「軌跡に従って手を動かす」という動作情報を送る処理が必要であり、それぞれ「把持動作クラス」と「直線動作クラス」に定義する。

これらの 4 つの処理を実現する性質クラスを、引き出しの仮想物体クラスである引き出しクラスに加えることにより、引き出しが仮想空間内で実現すべきすべての処理を実現することができる。

作業実演システムでは、図 5.2 に示した作業を実現するために、車、ボンネット、スパナ、ナット、ボルト、机、引き出し、たんす、バッテリーの 9 つの仮想物体が必要となる。これらの仮想物体についても引き出しの場合と同様に、仮想物体の行う処理を

4種類に分けて考え、これらの仮想物体をシミュレーションする方法を定義する仮想物体クラスとそれに属する性質クラスを表5.2に、各性質クラスが行う処理を表5.3にまとめる。

表 5.2: 仮想物体クラスと、それに属する性質クラス

仮想物体クラス	性質クラス
車クラス	形状クラス、衝突判定クラス
ボンネットクラス	形状クラス、衝突判定クラス、被把持クラス、回転運動クラス、把持動作クラス、弧状動作クラス
スパナクラス	形状クラス、衝突判定クラス、被把持クラス、回転運動クラス、把持動作クラス、弧状動作クラス
ナットクラス	形状クラス、衝突判定クラス、被把持クラス、回転運動クラス、把持動作クラス、手回転動作クラス
ボルトクラス	形状クラス、衝突判定クラス
机クラス	形状クラス、衝突判定クラス、置動作クラス
引き出しクラス	形状クラス、衝突判定クラス、被把持クラス、直線運動クラス、把持動作クラス、直線動作クラス
たんすクラス	形状クラス、衝突判定クラス
バッテリークラス	形状クラス、衝突判定クラス、被把持クラス、把持動作クラス

仮想物体の定義例

ここでは、まず引き出しの定義例について述べた後、その定義に従って引き出しの動きをシミュレーションする方法を述べる。

表5.2に示したように、引き出しは、仮想物体クラスである引き出しクラスに、形状クラス、衝突判定クラス、被把持クラス、直線運動クラス、把持動作クラス、直線動作クラスがメンバクラスとして属する構造となる。図5.17〜図5.23に各クラスの定義を示す。

以下では、それぞれのクラスの定義と働きについて述べる。

- 形状クラス

  形状クラスは、仮想物体の3次元形状に関する情報を保持するクラスである。図5.17

表 5.3: 性質クラスが行う処理

クラス名	シミュレーションする仮想物体の処理
形状クラス	仮想物体の形状を表示する処理
衝突判定クラス	衝突を判定する処理
被把持クラス	人間に握られる処理
回転運動クラス	軸を中心として回転する処理
直線運動クラス	平行移動する処理
把持動作クラス	人間に「持つ」動作を誘発する処理
弧状動作クラス	人間に「弧に沿うように手を動かす」動作を誘発する処理
直線動作クラス	人間に「直線に沿うように手を動かす」動作を誘発する処理
手回転動作クラス	人間に「手をまわす」動作を誘発する処理
置動作クラス	人間に「置く」動作を誘発する処理

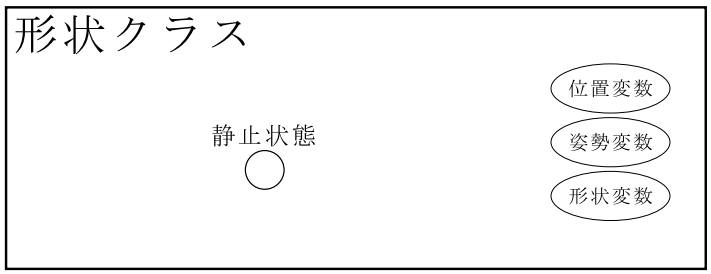


図 5.17: 形状クラスの定義

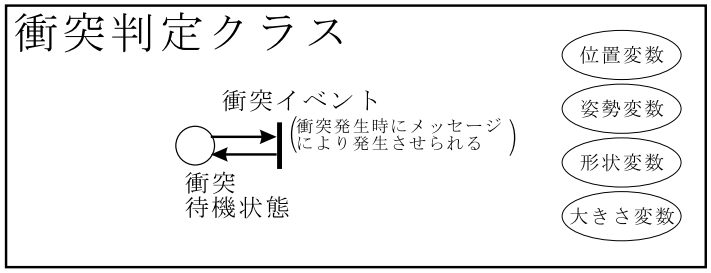


図 5.18: 衝突判定クラスの定義

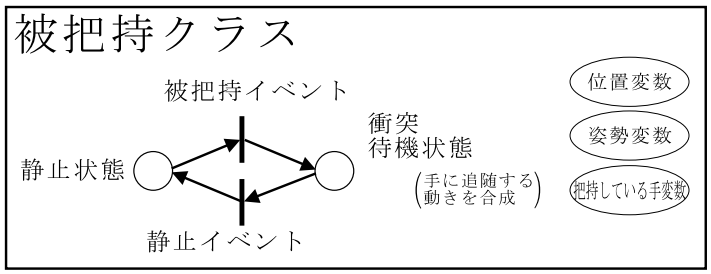


図 5.19: 被把持クラスの定義

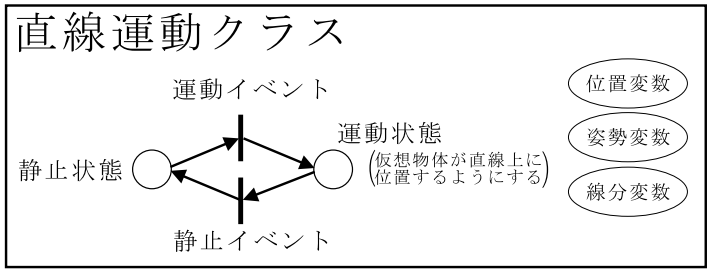


図 5.20: 直線運動クラスの定義

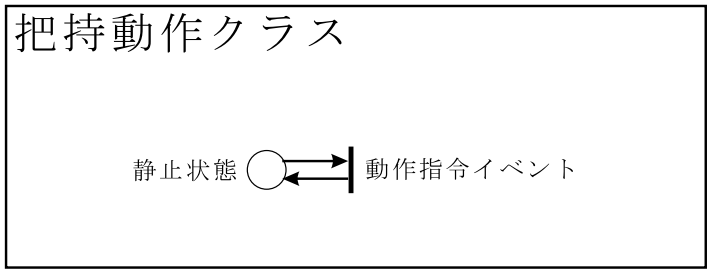


図 5.21: 把持動作クラスの定義

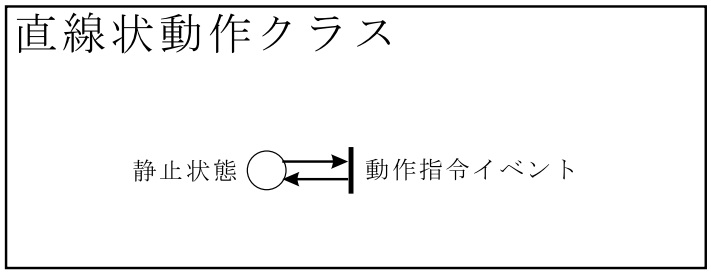


図 5.22: 直線動作クラスの定義

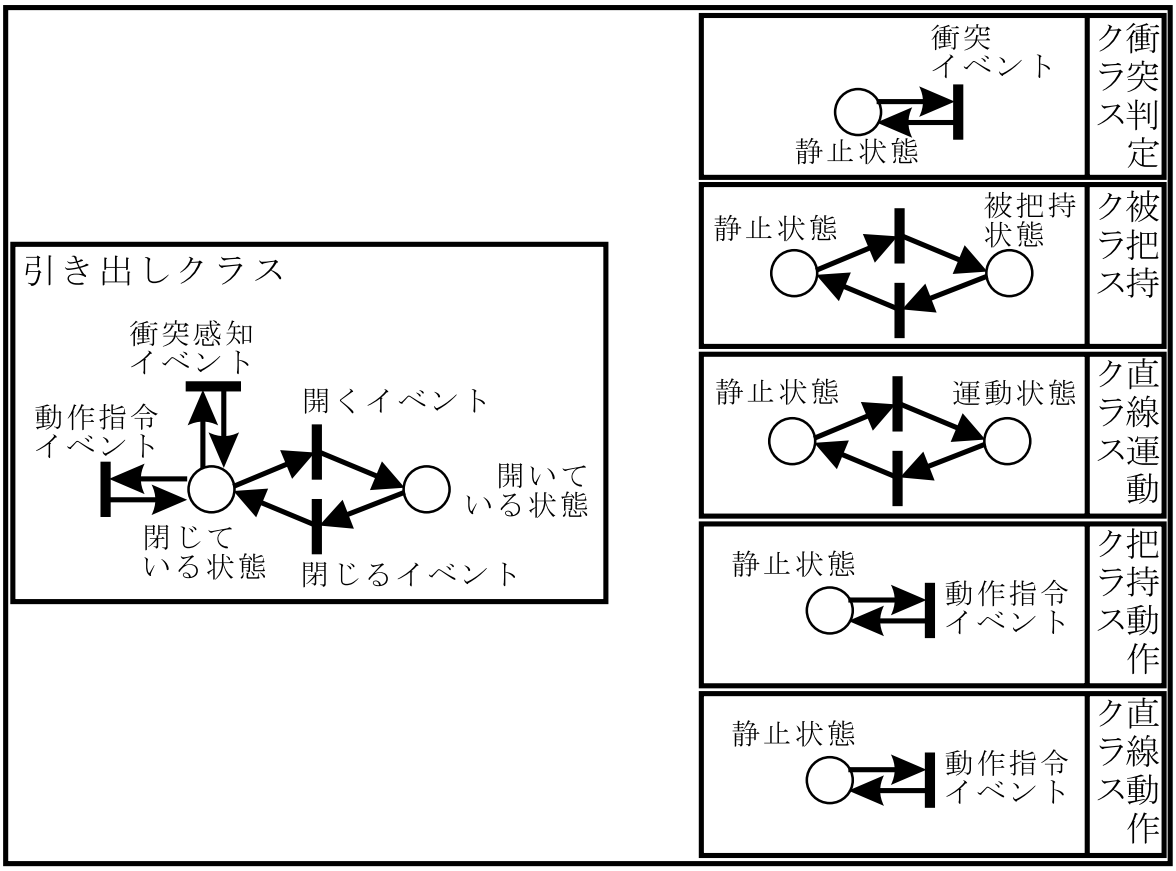


図 5.23: 引き出しクラスの定義

にクラスの定義を示す。形状クラスは「静止状態」という 1 つの状態のみが定義され、「位置」「姿勢」「形状」の 3 つの変数を持つ。後で 5.2.5 項で述べる 3 次元アニメーションプラグインが、これらの変数をメッセージ交換で取得し、形状クラスがメンバクラスとして属している仮想物体の 3 次元映像を合成可能にする。

- 衝突判定クラス

  衝突判定クラスは、仮想物体に他の仮想物体と衝突する性質を提供する。図 5.18 にクラスの定義を示す。衝突判定クラスには「静止状態」という 1 つの状態と、「衝突イベント」というイベントが定義されれ、「位置」「姿勢」「形状」「大きさ」という 4 つの変数を持つ。後で 5.2.5 項で述べる衝突判定プラグインが、この 4 つの変数と他の仮想物体クラスに属している衝突判定クラスが保持している 4 つの変数と比較して、衝突したか否かを判定する。衝突判定の結果、衝突した場合には、メッセージが双方の衝突判定クラスに送られ、衝突イベントが発生する。なお、衝突のメッセージには衝突した相手の仮想物体の情報が含まれる。

- 被把持クラス

  被把持クラスは、仮想物体が手に追随して動く性質を提供する。図 5.19 にクラスの定義を示す。被把持クラスには、「静止状態」と「被把持状態」の 2 つの状態と、「位置」「姿勢」「把持している手」の 3 つの変数を持つ。「被把持状態」のときには、予め定義された「把持している手」に追随して動きを合成する処理が行われ、処理結果が「位置」と「姿勢」の変数に格納される。

- 直線運動クラス

  直線運動クラスは、特定の線分上に仮想物体の動きを限定する性質を提供する。図 5.20 にクラスの定義を示す。直線運動クラスには、「静止状態」と「運動状態」の 2 つの状態を定義し、さらに「位置」「姿勢」「線分」に関する情報を変数として定義する。「運動状態」の時には、仮想物体が線分上にあるかを判定し、線分上になければ、線分上に強制的に移動させる。

- 把持動作クラス

  把持動作クラスは、仮想物体を握るという動作を仮想人間に誘発する性質を持つ。図 5.21 にクラスの定義を示す。把持動作クラスには、「静止状態」という 1 つの状態と、「動作指令イベント」という 1 つのイベントが定義される。「動作指令イベント」が発生すると、仮想人間に「握る」という動作情報をメッセージで送信

する。

- 直線動作クラス
  直線動作クラスは、手を線分に沿って移動させる動作を仮想人間に誘発する性質を持つ。図 5.22 にクラスの定義を示す。直線動作クラスには、「静止状態」と、「動作指令イベント」が定義されている。「動作指令イベント」が発生すると、仮想人間に「線分に沿って手を移動させる」動作情報をメッセージで送信する。

- 引き出しクラス
  引き出しクラスは、引き出しの仮想物体クラスであり、引き出しのすべての性質を持つ。図 5.23 にクラスの定義を示す。引き出しクラスには、「閉じている状態」と「開いている状態」の 2 つの状態を定義する。さらに「開くイベント」と「閉じるイベント」の 2 種類の状態遷移を起こさせるイベントと、「衝突感知イベント」、「動作指令イベント」の 2 つのイベントを定義する。

5.2.4 仮想物体のシミュレーション例

本項では、前項で定義した引き出しがどのようにシミュレーションされ、引き出しが仮想人間に開けられる現象を実現するかを説明する。なお、仮想人間の動作の合成については、5.2.2 項で説明したので省略する。

図 5.24 に示すように、引き出しの初期状態として、引き出しインスタンスは「閉じている状態」にあり、その他のすべてのインスタンスは「静止状態」にあり、引き出しは静止しているとする。

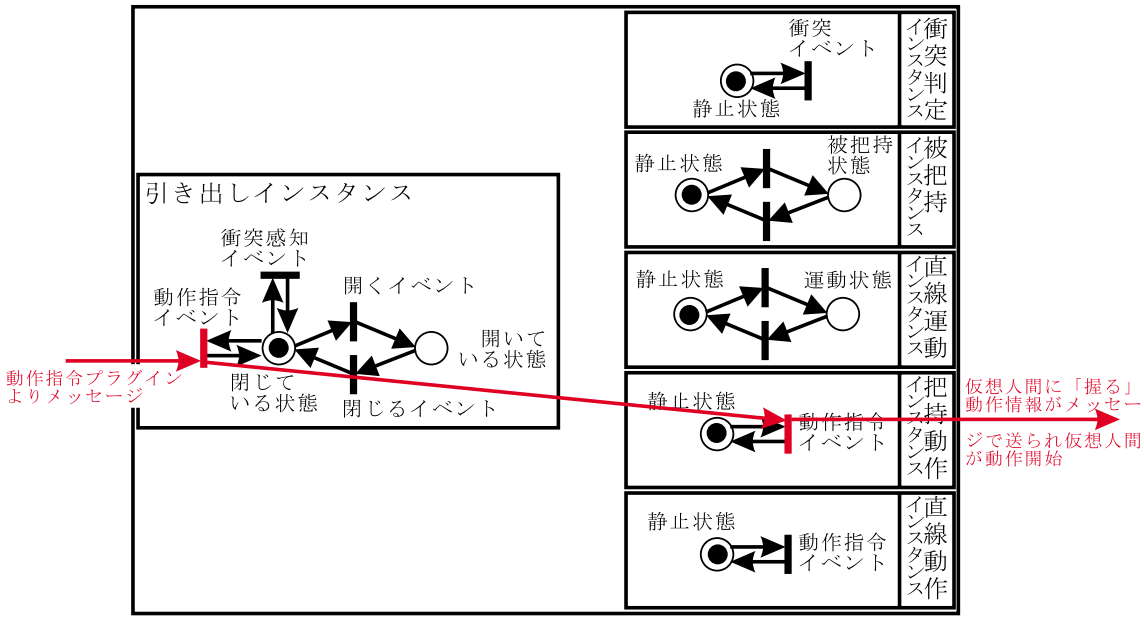


図 5.24: 引き出しが開くシミュレーション (閉じた状態)

作業実演システムでは、仮想人間が引き出しを開ける動作のシミュレーションは、まず、訓練生から仮想人間への動作指令により開始される。訓練生が、5.2.5 項で説明する動作指令プラグインのインタフェースを使用して、仮想人間に「引き出し」を「開ける」動作を指令する。この時、動作指令プラグインは、行動の対象として指定された仮想物体である「引き出しインスタンス」へメッセージを送り、引き出しインスタンスの動作指令イベントを発生させる。これにより、動作指令イベントの定義に従って、把持動作インスタンスの動作指令イベントが発生し、仮想人間に「握る」動作情報がメッセージで送られ、仮想人間が握る動作を開始する。

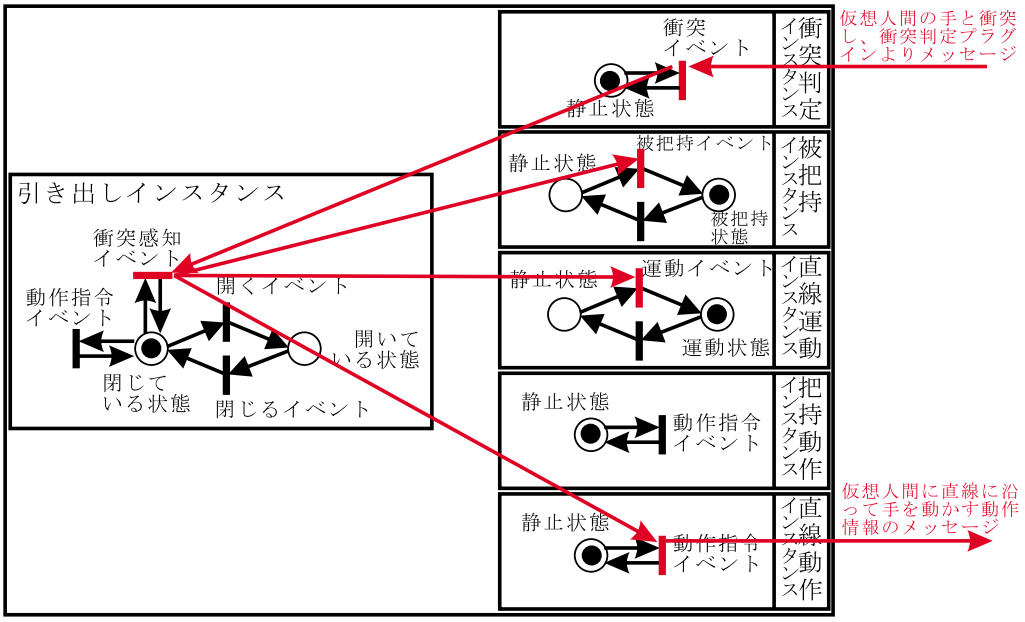


図 5.25: 引き出しが開くシミュレーション (開く途中の状態)

仮想人間が引き出しを握るための動作を行い、仮想人間の手と引き出しが接触すると、衝突判定プラグインから、引き出しの接触判定インスタンスに衝突のメッセージが送られ、「衝突イベント」が発生する。この時、引き出しインスタンスの定義に従い、引き出しインスタンスの「衝突感知イベント」、被把持インスタンスの「被把持イベント」、直線運動インスタンスの「動作イベント」、直線動作インスタンスの「動作指令イベント」が連続して発生する。この一連のイベントの発生により、引き出しの状態は図 5.25 に示す状態に遷移する。また、直線動作インスタンスの「動作指令イベント」の発生により、仮想人間に対して、引き出しが開く方向の直線に沿って手を移動させる動作を誘発する情報をメッセージで送り、仮想人間は、手を直線に従って移動させ、引き出しを開ける動作を開始する。この時、被把持インスタンスは「被把持状態」で

あり、仮想人間の手の動きに追随する動きをシミュレーションし、直線運動インスタンスは「運動状態」であり、引出しの動きを直線上に制限する。つまり、引き出しは、仮想人間の手の動きに従って、直線方向に移動する。

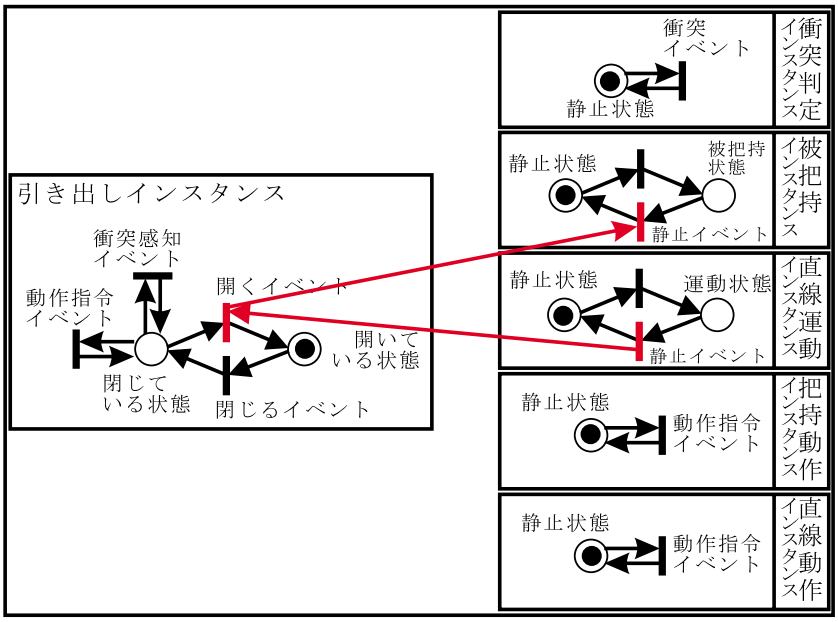


図 5.26: 引き出しが開くシミュレーション (開いた状態)

以上の処理により、引き出しが開き始め、直線運動インスタンスの変数で指定されている線分の端点まで来ると、直線運動インスタンスの静止イベントが発生する。このとき、引き出しインスタンスの定義に従い、引き出しインスタンスの「開くイベント」と、被把持インスタンスの「静止イベント」が発生する。この一連のイベントの発生により、引き出しの状態は、図 5.26 に示す状態となり、静止する。以上により、引き出しが仮想人間に開けられるシミュレーションを実行する。

5.2.5 プラグインの構築

本項では、作業実演システムを構築するために必要なプラグインについて説明する。構築したプラグインを表 5.4 に示す。

作業実演システムを構築するにあたり、これらのプラグインは、GLib ライブラリを用い C 言語で構築した。

個々のプラグインの詳細を以下で説明する。

表 5.4: プラグインとその役割

プラグイン名	役割
タッチ動作プラグイン	仮想人間の対象物に触る動作を合成
握るプラグイン	仮想人間の対象物を握る動作を合成
手直線移動プラグイン	仮想人間の直線に沿って手を動かす動作を合成
移動プラグイン	仮想人間の移動する動作を合成
追随プラグイン	仮想物体の他の仮想物体の動きに追随する動きを合成
直線移動プラグイン	仮想物体の直線に沿って動く動きを合成
衝突判定プラグイン	仮想物体間の衝突を判定
動作情報伝達プラグイン	メッセージを使用して動作情報を仮想物体から仮想人間に送信
動作指令プラグイン	訓練生の仮想人間に対する指示を入力
3次元アニメーションプラグイン	仮想物体と仮想人間の動きを3次元アニメーションとしてディスプレイに表示

タッチ動作プラグイン

タッチ動作プラグインは、腕の動きを合成する性質クラスである腕クラスの「触る動作状態」に定義されている、腕を仮想物体に向かって伸ばしていくという動作の合成を行う。作業実演システムでは、引き出しを握るために腕を引き出しに伸ばす動作の合成等に使用する。

本プラグインでの腕の動作を合成する際の処理の流れを以下に示す。

1. 腕インスタンスから、手で触る対象となる仮想物体の情報を取得する。
2. 手で触る対象となる仮想物体インスタンスが変数として保持している位置、姿勢 (目標位置) と、現在の仮想人間の腕インスタンスが変数として保持している手の位置、姿勢 (現在位置) を取得する。
3. 現在位置と目標位置を用いて手の通る軌跡と、その間の手の姿勢の変化を計算する。
4. 軌跡の長さから動作合成にかける総フレーム数を計算する。
5. 各フレームにおける手の位置と姿勢を計算する。その際、人間の動作として不自然に見えないように、腕の動きはじめと動き終わりの手の移動速度は遅く、途中での手の移動速度は速くなるように計算する。
6. 各フレームにおける手の位置から、次に述べる逆運動学アルゴリズムを使用して、腕全体の姿勢を決定する。

逆運動学アルゴリズムとは、ロボティクスの分野で、ロボットアームの姿勢を制御するために開発されたアルゴリズムである。近年、人体の動きを 3 次元アニメーションで合成する際に、多様な動作を合成可能なアルゴリズムとして注目を集めている。関節数が必要最低限に抑えられているロボットアームと異なり、人間と同様の動作をする仮想人間の腕は自由度が高く、ロボティクスの分野における逆運動学アルゴリズムをそのまま適用することはできない。そこで、逆運動学アルゴリズムを仮想人間の腕に適用する手法がいくつか発表されている。本研究では、その中から、手の位置と姿勢から腕全体の姿勢を決定するためのアルゴリズムとして、Tolani らの手法 [21] を採用する。具体的な手法については、付録 B に示す。

手直線移動プラグイン

手直線移動プラグインは、腕の動きを合成する性質クラスである腕クラスの「軌跡動作状態」に定義されている手を直線に沿って移動させる動作の合成を行う。作業実演システムでは、引き出しを握る際に腕を引き出しに伸ばす動作の合成等に使用する。

本プラグインで腕の動作を合成する際の処理の流れを以下に示す。

1. 腕クラスが保持している変数から、手を移動させる軌跡となる直線と、手を移動させる距離に関する情報を取得する。
2. 移動させる距離の長さから動作合成に掛ける総フレーム数を計算する。
3. 各フレームにおける手の位置と姿勢を計算する。その際、人間の動作として不自然に見えないように、腕の動きはじめと動き終わりの手の移動速度は遅く、途中での手の移動速度は速くなるように計算する。
4. 各フレームにおける手の位置から、次に述べる逆運動学アルゴリズムを使用して、腕全体の姿勢を決定する。

逆運動学はアルゴリズムは、タッチ動作プラグインと同じものを使用した。

移動プラグイン

移動プラグインは、仮想人間の移動動作を合成する性質クラスである胴クラスの「移動状態」に定義されている移動する動作の合成を行う。作業実演システムでは、引き出しの前まで移動する等の動作の合成に使用する。

5.2.2 項で述べたように、作業実演システムでは、上半身のみのモデルを仮想人間として採用している。そのため、このプラグインでは、歩行動作などの合成処理は行わず、仮想人間の胴を目標地点まで直線移動させる。

具体的に移動動作を合成するための処理流れを以下に示す。

1. 移動の目標となる仮想物体クラスが変数として保持している位置、姿勢 (目標位置) と、現在の仮想人間の胴クラスが変数として保持している胴の位置、姿勢 (現在位置) を取得する。
2. 現在位置と目標位置から胴の通る軌跡と、その間の姿勢の変化を計算する。
3. 軌跡の長さから動作合成に掛ける総フレーム数を計算する。

4. 各フレームにおける胴の位置と姿勢を計算する。その際、人間の動作として不自然に見えないように、腕の動きはじめと動き終わりの移動速度は遅く、途中での移動速度は速くなるように計算する。

追随プラグイン

追随プラグインは、仮想物体を他の仮想物体の動きに追随して移動させるプラグインである。被把持クラスの「握られた状態」に定義されている、人の手に追随する動きの合成などを行う。作業実演システムでは、引き出しが仮想人間に握られた際に、仮想人間の手に追随する動作の合成等に使用する。

直線移動プラグイン

直線移動プラグインは、仮想物体が動く範囲を直線上に限定するためのプラグインである。直線運動クラスの「限定状態」に定義されている、仮想物体の動きを直線上に限定する処理などを行う。作業実演システムでは、引き出しが仮想人間に握られた際に、引出しの移動方向を限定するために使用する。

具体的な処理流れを以下に示す。

1. 仮想物体の位置と姿勢を取得する。

2. 仮想物体の位置と姿勢が与えられた直線上からずれている場合には、最も近い直線上の点へ移動させる。

衝突判定プラグイン

衝突判定プラグインは、仮想物体間の衝突を判定するプラグインである。このプラグインは、仮想物体間のインタラクションを判定するプラグインであり、4.3 節で述べた、仮想空間の 1 単位のシミュレーションが終了する毎に実行される。衝突判定プラグインは、仮想物体のメンバクラスとして定義されている全ての衝突判定クラスから、位置と姿勢と大きさの情報を読み取り、他の仮想物体の衝突判定クラスが持つ情報を用いて衝突判定する。衝突判定の結果、2 つの仮想物体が衝突する位置関係にあると判定したときには、双方の衝突判定クラスにメッセージを送り、衝突イベントを発生させる。

現在までに構築した衝突判定プラグインは、直方体と球、球と球、直方体と直方体の各形状の間で衝突判定することが可能であり、作業実演システムを実現する際には、

すべての仮想物体の形状を直方体もしくは球に近似して、衝突判定を行う。

動作情報伝達プラグイン

5.2.2 項で述べたように、仮想人間は仮想物体から動作情報を送られて初めて動作する。このプラグインは、動作情報をメッセージとして仮想物体から仮想人間に送る役割を担う。

作業実演システムでは、握る動作を仮想人間に行わせる握る動作クラスと、軌跡に沿って手を動かす動作を仮想人間に行わせる弧状動作クラス及び直線動作クラスから、それぞれの動作情報を仮想人間に送信する役割を果たす。

動作指令プラグイン

動作指令プラグインは、仮想人間が実行可能な動作の一覧を訓練生に示し、訓練生が選択した動作の種類に従って、仮想人間に対して適切なイベントを発生させるプラグインである。

具体的な処理の流れを以下に示す。

1. 訓練生が入力インタフェースを介して、仮想人間が実行可能な動作の一覧表示を要求する。

2. 動作指令プラグインが、仮想空間内にあるすべての仮想物体クラスに対して「可能動作名取得イベント」を発生させ、クラスから返された情報を基に仮想物体、及びその仮想物体に対して仮想人間が実行できる動作の一覧を作成する。

3. 2. で作成された動作の一覧を訓練生に提示する。

4. 訓練生が仮想人間の動作の対象となる物体と動作の内容を指示する。

5. 選択された内容を基に、動作指令プラグインが適切なイベントを発生させる。

この後、仮想物体の動作指令イベントが、動作情報を仮想人間に対して提供する性質を持つ、性質クラスの動作指令イベントを発生させ、動作情報伝達プラグインを介して、仮想人間に動作情報を届けることにより、仮想人間が、訓練生が行った指令に従って動作を行う。

3 次元アニメーションプラグイン

3 次元アニメーションプラグインは、仮想物体や仮想人間の動きを 3 次元アニメーションとして合成し、訓練生に提示するためのプラグインである。このプラグインは、4.2.6 項で述べた出力プラグインに相当する。

3 次元アニメーションプラグインは呼び出されると、以下に示す処理を行うことで訓練生に 3 次元アニメーションを提示する。

1. 仮想物体の形状に関する性質を定義している形状クラスが変数として持っている位置、姿勢と形状に関する情報を取得する。

2. 1. の処理を仮想空間内の仮想物体がメンバクラスとして所有しているすべての形状クラスに対して行い、仮想空間内にあるすべての仮想物体の位置、姿勢と形状を取得する。

3. 得られた情報から 3 次元アニメーションを合成する。

5.3 OCARINA の動作例とその評価

本節では、まず 5.2 節で説明した作業実演システムで合成した仮想人間による作業の実演のアニメーション例について説明する。その後、OCARINA を使用した仮想空間構築作業について評価を行う。

5.3.1 OCARINA の動作例

5.2 節で述べた方法に基づき構築された作業実演システムをシミュレーションしたときのアニメーションの出力例を図 5.27 に示す。シミュレーションは、訓練生が入力インタフェースを通して仮想人間に動作を指令することによって開始され、5.2.1 項で示した順番に従って仮想人間によって作業の実演が行われた。

5.3.2 仮想空間の構築に必要な作業量の評価

ここでは、5.2 節で述べた作業実演システムを対象に、OCARINA を使用して構築した場合と、3.2.1 項で述べた従来の手法に従ってプログラミングして構築した場合を比較し、構築に要する時間を調べることにより、OCTAVE 手法で仮想空間を構築することの有用性を評価する。

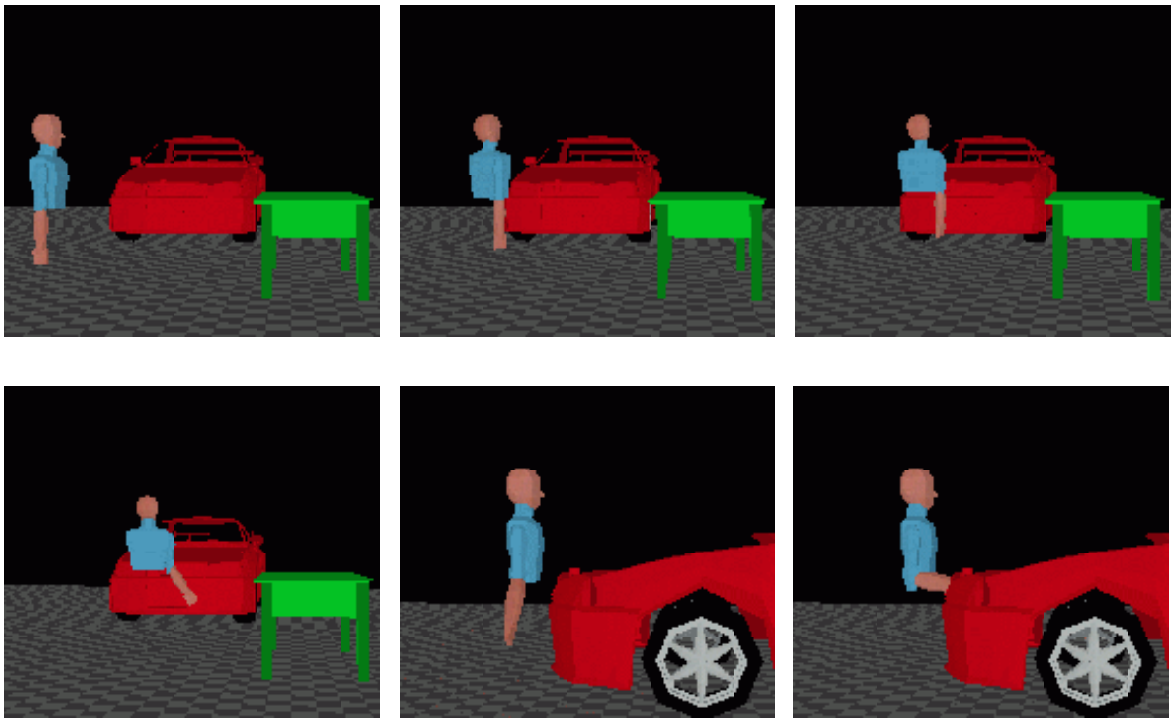


図 5.27: 作業実演システムの動作例

構築方法

表 5.5 に本研究で行った、作業実演システムの構築方法を示す。OCARINA を用いた作業実演システムの構築作業は、OCARINA を開発した本人によって行われた。構築した作業実演システムの詳細については、5.2 節で説明した。なお、プラグインは C 言語によるプログラミングで作成し、仮想物体クラス、性質クラスは独自スクリプトを通常のテキストエディタで編集することにより作成した。

プログラミングによる作業実演システム構築は、プログラミングによる仮想空間構築の経験が豊富なプログラマによって行われた。構築は、3.2 節で述べた従来手法による仮想空間管理手法に従って行われた。

表 5.5: 作業実演システムの構築方法

	構築者	手法
OCARINA を用いた構築	OCARINA を開発した本人	OCARINA を使用
プログラミングによる構築	仮想空間構築の経験者	従来手法に基づいて C 言語によるプログラミング

表 5.6: 作業実演システムの構築時間

構築方法	構築部分		構築所要時間 (日)
プログラミング	全体		30
OCARINA	全体		33
	内訳	仮想物体クラス (9 個)	5
		性質クラス (10 個)	5
		プラグイン (10 個)	20
		仮想人間クラス (5 個)	3

作業時間の比較とその考察

作業実演システムを、OCARINA で構築した場合の作業時間と、プログラミングで構築した場合の作業時間を表 5.6 にまとめる。なお、OCARINA での構築に関しては、仮想物体クラスと性質クラス、仮想人間クラス、プラグインに関する構築時間についても並記する。

表 5.6 より、以下の 2 つのことがわかる。

1. OCARINA を用いた構築は、プログラミングによる構築の場合と比較して、構築作業にほぼ同等の時間を要した。

2. OCARINA の構築では、プラグインと性質クラスを構築する時間が、全構築時間の 75%をしめる。

まず、1. に関して考察する。OCARINA を用いた構築とプログラミングによる構築には、一般的に、それぞれに作業時間が増加する要因が複数存在する。プログラミングによる構築の作業時間を増やす要因として、3.2.2 項で述べたように、「仮想物体同士の依存関係を定義しなければならない」ことや、「作業資産を再利用することが難しい」ことがあげられる。仮想物体同士の依存関係の定義量は、仮想空間内に配置する仮想物体の量の 2 乗に比例して増加し、作業資産の再利用が難しいことによる作業量の増加は、2 度目以降の仮想空間構築に影響する。

それに対して、OCARINA による構築の作業時間を増やす要因としては、OCARINA では仮想物体を性質によって分割して作成するために、「仮想物体をどのような性質に分割するかを考慮する必要がある」ことや、「仮想物体クラスから性質クラスに対する

イベントの伝搬などの依存関係を定義する必要がある」ことがあげられる。これらの作業量は、ともに仮想物体の数に比例して増加する。

すなわち、OCARINA を用いた構築の場合、仮想空間内の仮想物体の数が多いときや、2 度目以降の構築の際に、その構築にかかる作業時間を大幅に短縮できると考えられる。

本研究では、協調作業型訓練システムの機能の一部を実現する、作業実演システムの構築を行ったが、この作業実演システムは、1 人の仮想人間と 9 個の仮想物体から構成される小規模な仮想空間である。また、プログラミングの場合も OCARINA の場合も、初めての仮想空間構築であり、過去の作業資産のない状況から作業を行った。これらは、ともに上述した、OCARINA が作業時間を短縮できる条件に該当しないため、作業時間の短縮につながらず、OCARINA を用いた構築でも、プログラミングによる構築と同等の作業時間を要した。

次に、2. について考察する。2. で示したように、プラグインと性質クラスの構築に要した時間を合計すると、全構築時間の 75%を消費している。しかし、プラグインと性質クラスは、再利用が容易である。仮想物体の性質を定義する性質クラスは、仮想物体の種類が増えてもそれほど増加しないと考えられる。例えば、仮想物体の動き方は、仮想人間に持たれるなどの作業によって生じるものや、自由落下など基本的な物理法則に即して生じるものなど、種類は多くなく、また、仮想物体間のインタラクションも、衝突などに限られている。このため、仮想物体の動き方や、インタラクションを定義する性質クラスの種類もある程度限定された数となる。また、性質クラスで定義している性質を実際に計算し、シミュレーションする機能を持つプラグインの数も、同様に限定される。このため、プラグインと性質クラスに関しては作業資産の再利用性が高い。

これに対し、仮想物体の種類は非常に多いと考えられる。現実世界で物体と認識できるものはすべて仮想物体として仮想空間内に入れることができる。そのため、新たに仮想空間を構築する場合には、多くの場合、仮想物体クラスを新たに作り直さなくてはならない。

以上で述べた理由により、新たに仮想空間を構築する場合は、定義しなければならないクラスやプラグインの量は少なくなると考えられ、2 回目以降の構築では、作業時間を大きく削減できる。

5.3.3 OCARINA で仮想空間を構築する際の問題点

仮想空間を構築するために要した時間から判明したこと以外にも、OCARINA を使用して実際に仮想空間を構築する過程で、OCARINA を用いた仮想空間の構築に関して次のような問題点が明らかになった。

- クラス定義ファイルは、独自に規定したスクリプト言語で記述しなければならないため、初期の学習に要する労力が大きい。
- クラスを設計するためには、オブジェクト指向に関する知識が不可欠であり、オブジェクト指向を知らない人が記述するのは困難である。
- 状態遷移をテキスト形式で記述しなければならないので、ペトリネットという視覚的に状態遷移を記述できるツールを導入する利点を活用できていない。
- 作成したクラスファイルの記述ミスなどを指摘する機能が少なく、仮想空間が予想通りの動作をしないときの、原因究明が難しい。

これらの問題はすべて、クラスの定義を独自のスクリプト言語によって記述しなければならないために発生する問題である。この問題の解決方法として、GUI によるクラス定義支援ツールを作成し、クラスを視覚的に確認しながら構築する方法が考えられる。この方法の具体的な内容については、次節で述べる。

5.4 今後の展望

本節では、仮想空間を簡単に、少ない労力で作成することを可能にするために、今後取り組むべき課題について説明した後、協調作業型訓練システムの実現へ向けた今後の展望について述べる。

5.4.1 仮想空間構築の効率化

本研究では、仮想空間を効率的に設計するための手法と、その手法で設計された仮想空間をシミュレーションするためのシステムを開発した。本手法を用いることにより、大規模で複雑な仮想空間を構築する際には、従来の手法と比較して、大幅な労力削減につながることが予想される。しかし、「クラス定義ファイルの記述が難しい」、「小規模の仮想空間の構築には労力がかかる」、「始めての構築作業の場合に労力がかかる」等の問題点が明らかになった。

ここでは、これらの問題の解決策として、次の 2 つの方法を提案する。

- プラグインと性質クラスのライブラリを充実させる
- 仮想物体クラスの作成を支援する GUI のツールを提供する

以下で、これらの方法について説明し、導入による利点を述べる。

プラグインと性質クラスのライブラリの充実

まず、プラグインと性質クラスのライブラリの作成について述べる。OCTAVE 手法を使用して仮想空間を構築する大きな利点の 1 つに、一度作成したクラスや、プラグインの再利用が容易なことがあげられる。このため、予め十分な数のクラスやプラグインを用意すれば、クラスの定義やプラグインの構築という、OCARINA を用いる仮想空間構築作業で最も労力を要する部分が省略でき、既に用意されているクラスやプラグインを利用して仮想物体を追加していくだけの作業で、仮想空間の構築ができるようになる。

5.2 節で構築した作業実演システムの場合、プラグインがすべて用意されている場合には約 60%の構築労力の削減になり、プラグインに加えて性質クラスが用意されている場合には、約 75%の構築労力の削減になる。5.3.2 項で述べたように、構築する仮想空間によって求められる機能が異なることが多い仮想物体クラスは、再定義する必要があるが、再利用可能な性質クラスとプラグインが十分な数用意されていると、大きな労力の削減につながる。

また、プラグイン作成には高度なプログラミング技術が必要なため、VR の初心者には作成が困難である。しかし、十分な数のプラグインと性質クラスがあらかじめ用意されていれば、そのような人達でも仮想空間を構築することが可能となり、ユーザ層を大幅に広げることができる。

仮想物体クラスの作成を支援する GUI のツールの提供

OCARINA を使用して作成する際のもう 1 つの問題として、クラスを定義する困難さがあげられる。これは、クラスを定義するためには、独自のスクリプト形式のテキストファイルを記述しなければならないことに原因がある。上記のように、ライブラリを用意すれば、性質クラスの定義に要する労力は大幅に削減できるが、仮想物体クラスは構築する必要があり、この記述の難しさが問題となる。すなわち、上記のライ

ブラリの作成により、作業量の削減は可能であるが、それでもまだ、クラス定義ファイルの作成に高度な知識が必要である。

この問題に対する解決手段として、Graphical User Interface(GUI) を用いて簡単に、仮想物体クラスを作成することができる作成支援ツールを提供することが考えられる。3.3.1 項で説明したように、仮想物体クラスには具体的な性質はなく、性質クラスを制御する役割だけが与えられている。そのため、仮想物体クラスに定義しなければならないことは、「状態、イベントと状態の遷移を定義するペトリネット」と「性質クラスと仮想物体クラス間のイベント発生の伝搬」、「各種変数」および「追加するメンバクラス (性質クラス)」の 4 つである。

そのため、GUI を用いたクラス作成支援ツールを使用した場合、次のような手順で仮想物体クラスを構築することが可能となる。

1. 新規作成ボタンを押し、仮想物体クラスの作成を開始する。
2. 性質クラスライブラリに存在する性質クラスの一覧を表示させ、仮想物体が備えるべき性質が定義されている性質クラスを選択し、仮想物体クラスに追加する。
3. 仮想物体クラスの状態と、イベントおよびイベントによる状態遷移の仕方をペトリネットエディタを使用してペトリネットを編集して定義する。
4. 仮想物体に必要となる位置や姿勢などの情報を格納する変数を、変数追加ダイアログボックスを操作して追加する。
5. 性質クラスから仮想物体クラス、仮想物体クラスから性質クラスへのイベント発生の伝搬を、伝搬するイベント間を矢印で結んで定義する。
6. 矢印をダブルクリックして、伝達情報定義ダイアログボックスを表示し、イベント発生により伝達される情報を定義する。

また、GUI を用いてクラス作成支援ツールは、ペトリネットによってクラスの状態を視覚的にわかりやすく表示できる。この機能を利用すると、OCARINA を用いた仮想空間のシミュレーション時の各クラスの状態、イベントの発生とそれに伴う状態遷移を視覚的に確認し、作成した仮想空間の誤動作をチェックすることが可能となり、定義ミスの修正に役立てることも可能である。

これらツールを作成することにより、仮想物体クラスの定義が容易となり、仮想物体クラスの定義と、定義の修正に要する労力も減らすことができると期待される。

5.4.2　協調作業型訓練システムの構築へ向けて

協調作業型訓練システムとは、2.2 節で述べたように、仮想空間内に、実際の人と同じ形状をもつ仮想人間を配置し、その仮想人間と共同作業したり、仮想人間が訓練生に作業内容を教示することにより、これまでの訓練システムより更に効率的に訓練を進めることを目指したシステムである。5.2 節で述べたように、協調作業型訓練システムの一部分は、OCARINA を用いて本研究で構築した作業実演システムで構築済みである。今後、訓練生のジェスチャを認識して仮想物体の動きに反映する「動作入力サブシステム」と、仮想空間の状況 (作業の進捗状況) に応じて、仮想人間の行動を決定する「行動決定サブシステム」を構築する必要がある。以下にそれぞれのサブシステムを OCARINA を用いて構築する場合の方法についてまとめる。

動作入力サブシステム

動作入力サブシステムは、人間の「手」を仮想空間内に配置し、それをデータグローブや 3 次元マウスなどのジェスチャ入力機器を介して、操作することにより訓練生が仮想空間内で作業することを可能にする。入力機器と OCARINA の間は、入力プラグインによって結ばれ、訓練生がジェスチャを行うことで、手の位置や姿勢などの情報が入力プラグインを通して、「手」にメッセージとして送られる。「手」は送られてくる情報に従って動き、仮想物体に対して様々な操作やインタラクションを起こすことが可能となる。

行動決定サブシステム

本研究室では、行動決定サブシステムとして、ペトリネットを用いた仮想人間の行動のモデル化手法を開発している。このモデル化手法の特徴は、仮想空間の現在の状態から最終の状態までの作業手順を探索することができることである。この手法を適用すれば、仮想空間の最初と最後の状態を与えると、最後の状態に至るための作業が決まるので、これに従って仮想人間に自動実演をさせることが可能である。

モデル化の手順を以下に示す。

手順 1 仮想物体の状態 1 つをプレース 1 つに対応させる。

手順 2 仮想物体のイベントの発生を、トランジションの発火に対応させる。

手順 3 トランジションに仮想人間の動作を対応させる。

モデル化手順1〜3に従って、「机と鉛筆があり、鉛筆が机の引き出しの中に置かれ、引き出しから鉛筆を取り出す」ことを目的とする。仮想空間をペトリネットで表現したのが図5.28である。引き出しは、初期状態として「閉まっている」状態にあり、鉛筆は「引き出しの中にある状態」にある。

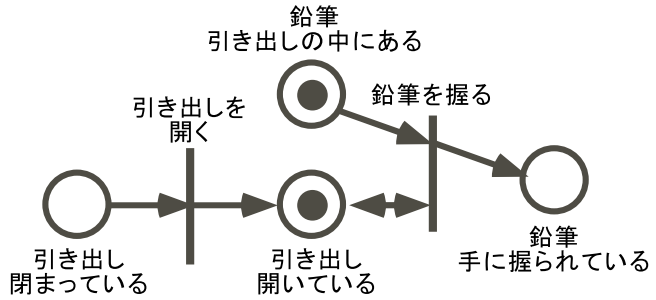


図 5.28: ペトリネットで作成した仮想人間の行動のモデル化例

ここで、訓練の最後の状態として、鉛筆が「手に握られている状態」を設定すると、ペトリネットの発火系列を検索することにより、まず、引き出しのトランジションが発火し、仮想人間はそのトランジションに定義されている「引き出しを開く」動作を行う。次に、鉛筆のトランジションが発火し、仮想人間はそのトランジションに定義されている「鉛筆を握る」という動作を行う。このように、仮想人間に目的の状態までの作業動作を実演させることができる。

さらに、訓練生がジェスチャにより引き出しを開く動作をして、「引き出しが開いている状態」に移ったとする。この時、「鉛筆が手に握られている状態」を最後の状態として仮想人間に設定すると、発火系列の検索により、仮想人間は、「鉛筆を握る」動作だけを行い、「引き出しを開く」動作は行わない。このように、訓練生の行動に応じて、仮想人間はその行動を変化させることができる。

この手法をOCARINAに適用するには、行動決定サブシステムをプラグインとして構築し、行動決定サブシステムで定義されている状態と、OCARINAで定義されている状態を対応させる必要がある。

例えば、OCARINAシステムの引き出しインスタンスの「閉まっている状態」と「開いている状態」を、行動決定サブシステムの「引き出しが閉まっている」状態と「引き出しが開いている」状態を対応させる。こうすることにより、OCARINAにおける仮想空間の状態は、常に行動決定サブシステムに反映されることとなる。そして、上に記した発火系列を検索する方法で、トランジションを発火させることにより、仮想人間に動作を行わせることができる。

第 6 章 結論

本研究では、大規模仮想空間を構築する際に生じる各種問題を解決し、協調作業型訓練システムを少ない労力で構築可能にするための新しい手法として、オブジェクト指向を用いて仮想空間を設計・構築する手法である OCTAVE (Object-Oriented Technique for Constructing Interactive Virtual Environment) 手法を提案した。また、OCTAVE 手法に基づいて設計した仮想空間を、実際に構築しシミュレーションできるシステムとして OCARINA (Octave Based Virtual Environment Simulation System) を開発した。

第 2 章では、まず、機器操作・保守作業に注目して Virtual Reality の研究と応用の現状について述べ、本研究室で構築を目指している協調作業型訓練システムの目標と構成の概要を述べた。そして、協調作業型訓練システム全体における本研究の位置付けを明らかにした。また、VR 技術を用いて訓練システムを構築するための従来手法とその問題点について述べ、その解決策として、ソフトウエア開発の分野で効果をあげているオブジェクト指向を利用することを提案し、本研究の目的を明らかにした。

第 3 章では、まず、オブジェクト指向の概念について説明し、その利点についてまとめた。その後、仮想空間内における仮想物体の動きを定義し、シミュレーションする一般的な方法について述べ、その問題点を明らかにした。そして、オブジェクト指向に従って仮想空間を設計する OCTAVE 手法について説明し、具体的に OCTAVE 手法を用いた仮想空間の設計例について述べた後、OCTAVE 手法を用いて仮想空間を構築することによる利点についてまとめた。

第 4 章では、OCTAVE 手法を用いて設計した仮想空間を、実際に構築し、シミュレーションするためのシステムとして開発した仮想空間シミュレーションシステム OCARINA について述べた。まず、はじめに、協調作業型訓練システムを実現するためのシステムとして OCARINA が備えるべき機能について述べた後、開発した OCARINA の概要について説明し、次に OCARINA を構成する各サブシステムの役割について説明した。その後、OCARINA を実行する際の処理の流れについて説明し、最後にシステムのユーザが OCARINA を用いて仮想空間を構築し、シミュレーションを実行するまでの作業手順についてまとめた。

第 5 章では、OCARINA を使用して協調作業型訓練システムを構築する方法について説明し、その後、協調作業型訓練システムの機能の一部の実現例として、OCARINA

を用いて仮想教師が実際の人と同様の動作で作業を実演できる機能を備えた作業実演システムを構築する手法を説明した。その後、作業実演システム構築に要した作業量を考察し、今後の展望についてまとめた。

本研究では、仮想空間の構築手法にオブジェクト指向の概念を導入した OCTAVE 手法を提案し、従来手法では実現が困難なインタラクティブな大規模仮想環境の実現へ向けて解決策を示した。また、この手法に基づいて設計された仮想空間を、実際に構築し、シミュレーションすることができる仮想空間シミュレーションシステム OCARINA を開発し、その有効性を確認することができた。

しかし、OCARINA を用いて仮想空間の構築するには、クラスを定義するためのスクリプト言語を記述する必要があり、VR に詳しい人以外には困難である。そこで、本研究では、この問題を解決するために、今後更に実装すべき OCARINA の機能をいくつか検討した。これらの機能が実装されれば、仮想空間の構築がさらに容易になり、仮想空間を構築できるユーザ層を大幅に広げることができると考えられる。

OCTAVE 手法の最大の利点は、別々に構築した仮想物体を組み合わせて仮想空間が構築できることである。今後、この利点を活用し、積木を組み合わせて建物を作るような感覚で、用意されている仮想物体を配置するだけで、仮想空間を自由に設計・構築できるようになると期待される。

謝 辞

本研究を進めるにあたり、研究全般にわたってご指導を頂きました吉川榮和教授に深く感謝いたします。

本研究を進めるにあたり、数々の貴重な助言を頂きました下田宏助教授に深く感謝いたします。

本研究室に配属されてより 3 年間、常に研究に協力してくださり、本年度の研究においても研究の進行から論文の執筆に到るまで多大なるご協力を頂いた石井裕剛助手に心より感謝いたします。

本研究を進めるにあたり、プログラミング作業や論文の校正等、様々な面で協力していただいた修士課程 1 回生の社領一将君に深く感謝いたします。

修士課程の 2 年間、楽しい時も苦しい時もその経験を分かち合い、修士論文の執筆にあたっても最後まで共に闘い貫いた、修士 2 回生の伊藤京子さん、大坂融弘君、沖陽三君、笹井寿郎君、米田賀一君に感謝いたします。

最後に、研究を進める上で何かとお世話頂いた谷友美秘書、藤岡美紀秘書、吉川万里子秘書および吉川研究室の学生および宮沢研究室の学生の皆様にも心から感謝いたします。

参 考 文 献

[1] 廣瀬通孝:電子情報通信学会編「ヒューマンコミュニケーション工学シリーズ」バーチャルリアリティ, オーム社 (1995).

[2] 廣瀬通孝:バーチャルリアリティって何だろう, ダイヤモンド社 (1997).

[3] 野村淳二, 澤田一哉編著:日本ファジイ学会編「ソフトコンピューティングシリーズ」バーチャルリアリティ, 第 4 章, 朝倉書店 (1997).

[4] 南雲俊喜, 中山功, 甘利治雄, 岡田幹夫:小型円筒面スクリーンによる運転・保守作業訓練環境の構築, 日本バーチャルリアリティ学会第 3 回大会論文集, pp.207-208 (1998).

[5] 天野友博, 田中和明, 鄭絳宇, 安部憲広:仮想機械を用いる機器修復法の教示と誤りの検出・修正機構, 日本バーチャルリアリティ学会第 2 回大会論文集, pp.89-92 (1997).

[6] 新井浩一, 阿部慶子, 上地登:VR 技術を用いた変電所保守員向け集合教育用体感型シミュレータの開発, 日本バーチャルリアリティ学会論文誌, **Vol.2**, No.4, pp.7-16 (1997).

[7] 吉川榮和, 手塚哲央, 柏健一郎, 石井裕剛:仮想空間における機器保修訓練シミュレーション, 日本原子力学会誌, **Vol.39**, No.12, pp.72-83(1997).

[8] 柏健一郎:機器補修訓練へのデータグローブの適用性の研究, 京都大学工学部電気工学第二学科学士論文 (1993).

[9] Wu Wei, 中川隆志, 吉川榮和:プラント運転員の異常診断行動のモデリングとヒューマンモデルシミュレーションによる人間認知信頼性 (HCR) 曲線の導出法の研究, ヒューマンインタフェース学会論文誌, **Vol.1**, No.2, pp.11-24(1999).

[10] 佐々木正人:岩波科学ライブラリー アフォーダンス-新しい認知の理論, 岩波書店 (1994).

[11] 市口誠道:アフォーダンスの概念に基づく人体モーション合成システムの開発, 京都大学大学院エネルギー科学研究科修士論文 (2000).

[12] http://www.eonreality.com/

[13] 村上竜一:dVISE, 第５回産業用バーチャルリアリティ展・セミナー要録, pp.69-80(1997).

[14] P.Wegner 著, 尾内理紀夫訳:オブジェクト指向, 共立出版株式会社 (1992).

[15] I.Jacobson, G.Booch, J.Rumbaugh:UML による統一ソフトウエア開発プロセス・オブジェクト指向開発方法論, 翔泳社 (2000).

[16] J.Rumbaugh:オブジェクト指向方法論 OMT, トッパン (1992).

[17] 三谷拓也:仮想現実感による機器の分解組立シミュレーションに関する研究, 京都大学工学部電気工学第二学科学士論文 (1995).

[18] 廣瀬通孝編:バーチャルリアリティの基礎 3 VR 世界の構成手法, 培風館 (2000).

[19] 村田忠夫:ペトリネットの解析と応用, 近代科学社 (1992).

[20] J.L. ピータースン:ペトリネット入門, シュプリンガー・フェアラーク東京株式会社 (1992).

[21] D.Tolani, N.I.Badler:Real-Time Inverse Kinematics of Human Arm, Presence, **Vol.5**, No.4, pp.393-401 (1996).

付 録 目 次

付 録 図 目 次

付録 A クラス定義ファイル書式

OCARINA で OCTAVE 手法 に従った手法で仮想空間を構築するために記述するクラスファイルの書式の仕様書を以下に示す。

クラス定義ファイル書式

1.概要

・ クラスは命令文の集合で定義する。
・ 1 行に 1 つの命令文を記述するものとする。
・ 命令文は、命令識別子 命令内容 で構成される。
・ #から行末までは、コメントとする。
・ 行頭・行末の空白文字、Tab 文字は無視される。
・ 大文字・小文字を区別しない。
・ 定数は、システム上では、継承されないローカル変数として通常の変数構造体に入れられ、自動的に変数名が付与される。

2.命令識別子

各命令文について説明する。命令文は、”命令識別子 引数”という形式をとる。

ClassNew 命令

書式: ClassNew <クラス名> <仮想物体クラス><継承元クラス名>

新たなクラスと、そのクラスの働きを定義する。次の END_Class 命令までがクラス定義文とみなされる。定義文の中で使用可能な命令は、State 命令、StateNew 命令、Event 命令、EventNew 命令、Function 命令、Variable 命令、MemberClass 命令である。

Class 命令

書式: Class <クラス名><仮想物体クラス>

継承したクラスと、そのクラスの働きを定義する。仮想物体クラスになりうる場合は<仮想物体クラス>に TURE をなり得ない場合は FALSE を入れる。Class 命令から END_Class 命令までがクラス定義文とみなされる。定義文の中で使用可能な命令は、State 命令、StateNew 命令、Event 命令、EventNew 命令、Function 命令、Variable 命令、MemberClass 命令である。

END_Class 命令

書式: END_Class

クラス定義の終了を示す。

StateNew 命令

書式: StateNew <状態名>

新たな状態と、その状態における働きを定義する。次の END 命令までが状態定義文とみなされる。定義文の中で使用可能な命令は、Function 命令、Variable 命令であ

継承した状態と、その状態における働きを定義する。次の END 命令までが状態定義文とみなされる。継承元クラスに存在した「状態」の内、State 命令で継承されたものは消滅する。また、状態が継承された場合、継承元状態に関連し状態遷移を伴わないイベントは自動的に継承されるが、それ以外のイベントは自動的には継承されない。状態定義文の中で使用可能な命令は、Function 命令、Variable 命令である。

EventNew 命令

書式: EventNew <イベント名>

新たなイベントを定義し、そのイベントにおける状態の変化、イベントに伴う処理を定義する。次の END 命令までがイベント定義文とみなされる。定義文の中で使用可能な命令は、Function 命令、Variable 命令、StateChange 命令、Trigger 命令である。

Event 命令

書式: Event <イベント名>

状態を継承することにより、その状態に付随するイベントも自動的に継承される。状態の継承によって、自動的に継承されるイベントの名前は、“継承元イベント名_継承状態名”に限定される。Event 命令文は、そのイベントが発生したときの処理を上書き定義する。次の END 命令までがイベント定義文とみなされる。定義文の中で使用可能な命令は、Function 命令、Variable 命令、Trigger 命令である。

END 命令

書式: END

状態定義文、イベント定義文の終了を示す。

Function 命令

書式: Function <引数を受け入れる変数名> ..., <返値を返す変数名> ...

クラス、状態、イベントの定義文中で使用する。次の END_Function 命令までが Function 定義文とみなされる。新規に作成されるクラス、状態、イベントの定義文中の場合は、返値、引数を自由に設定できる。しかし、継承されたクラス、状態、イベント中で使用する場合は、返値、引数に追加することはできるが、削減することはできない。また、継承元で定義された Function 文は、完全に上書きされる。Function 定義文の中で使用可能な命令は、Variable 命令、Fire 命令、Plugin 命令、Calc 命令であり、継承されたクラス、状態、イベント中の Function 定義文では、InheritedFunction 命令も使用可能である。以下で、Function 命令の役割を、使用する場所別に説明する。引数がない場合は引数のところにNone を、返値がない場合は、返値のところに None を入れる。

Class 定義文中での使用時 クラスのインスタンス生成時の処理を定義する。

State 定義文中での使用時 インスタンスがその状態にあるときに、毎フレーム実行する処理を定義する。

Event 定義文中での使用時イベントが発生したときに、実行する処理を定義す

END_Function 命令
書式: END_Function
Function 定義文の終了を示す。
Variable 命令
書式: Variable <型名> <変数名> <初期値定数>
変数を定義する。変数の型の種類については、後に述べる。変数のスコープについては、後で述べる。
MemberClassNew 命令
書式: MemberClassNew <インスタンス名><クラス名><初期化 Function 命令の引数に渡す定数>…
新たなメンバクラスを定義する。定義文の中では、FirstState 命令で初期状態をかならず設定する必要がある。
MemberClass 命令
書式: MemberClass <インスタンス名><継承元インスタンス名><クラス名><初期化 Function 命令の引数に渡す定数>…
メンバクラスを上書き定義する。インスタンス名は継承元クラスに存在した上書きするメンバインスタンスと同じでなければならない。定義文の中では、FirstState 命令で初期状態をかならず設定する必要がある。
FirstState 命令
書式: FirstState <親インスタンスの初期状態> <メンバインスタンスの初期状態>
親インスタンスの初期状態とメンバインスタンスの初期状態を 1 対 1 で対応させ、親インスタンスの初期状態に合わせてメンバインスタンスの初期状態を設定する。
StateChange 命令
書式: StateChange <イベント前状態名> <イベント後状態名>
イベントの発生による状態の遷移を定義する。
Trigger 命令
書式: Trigger <イベント変数> <引数を受け取る変数名>…
イベントの発生する条件となる、インスタンスのイベントを定義する。他のインスタンスのイベントが発生した時に受け取った引数を受け取ることができる。引数を省略した場合は無視される。
Fire 命令
書式: Fire <イベント変数> <引数に渡す変数名>…, <返値を受け取る変数名>…
インスタンスのイベントを発生させる。引数、返値は、省略可能とする。引数を省略した場合は、各型のデフォルト値が適用され、返値を省略した場合は、無視される。
Plugin 命令
書式: Plugin <プラグイン名> <関数名><引数に渡す変数名>…, <返値を受け取る変数

Calc 命令

書式: Calc <計算式>

計算をする。できる計算については後述する。

InheritedFunction 命令

書式: InheritedFunction <引数に渡す変数名>..., <返値を受け取る変数名>

継承されたクラス、状態、イベントのFunction 定義文中で、継承もとのFunction 命令文を呼び出す。

3.使用できる型

型名	型の内容		定数書式(***に定数)	デフォルト値
String	文字列型	256Byte 文字列	<(STRING)***>	
Integer	整数型	32bit 整数	<(INTEGER)***>	0
Double	実数型	32bit 実数	<(DOUBLE)***>	0
Boolean	論理型	True or False	<(BOOLEAN)***>	TRUE
Vector	ベクトル型	3 次元ベクトル	<(VECTOR)*,*,*>	0,0,0
Quaternion	<(QUATERNION)*,*,*,*>	4 元数型	Quaternion 1,0,0,0	
Instance	インスタンス型	インスタンス	後述	SELF
State	状態型	状態	後述	SELF,SELF
Event	イベント型	イベント	後述	SELF,SELF

Instance, State, Event 型の仕様上の注意

Instance, State, Event 型には SELF 定数が存在する。これは、使用された場所が所属しているインスタンス、状態、イベントをそれぞれ指す。

Instance 型の定数式 ・・・・・・・・・ <(INSTANCE)インスタンス名 > 直接指定できるのは、SELF もしくは、メンバインスタンスのみ

State 型の定数式・・・・・・・・・・・・ <(STATE)インスタンス名 ,状態名> インスタンス名のところで、直接指定できるのは､SELF もしくは、メンバインスタンスのみ。

Event 型の定数式 ・・・・・・・・・・・・ <(Event)インスタンス名,イベント名> インスタンス名のところで、直接指定できるのは､SELF もしくは、メンバインスタンスのみ。

4.変数(Variable 命令)のスコープ

Function で使用される場合は、END_Function までがスコープである。Event(New), State(New), Class(New)で使用される場合は、それ自体と継承先までがスコープである。

5.変数について

変数は、前述したそれぞれの型について宣言可能である。変数に関する制限について述べる。

変数の命名制限

変数名は、アルファベットで始まる 256 文字以内の単語で、アルファベットと_(アンダーバー)と数字からなる文字列で構成されなければならない。ただし、継承元クラスで、一度でも宣言された変数名は、その継承先クラスでは使用できない。また、予約語も使用できない。また、予約変数は、特別な役割を果たす変数であるので、他の用途に使用することはできない。

予約語

上記に挙げた実行命令

ClassNew, Class, END_Class, StateNew, State, EventNew, Event, END, Function, END_Function, Variable, MemberClassNew, MemberClass, StateChange, Trigger, Fire, Plugin, Calc

型宣言命令

String, Integer, Double, Boolean, Vector, Quaternion, Instance, State, Event

と

YES, NO, None, SELF

は予約語である。

命名規則

以下のような命名を推奨する。*の部分は任意の文字列を示すとする。

クラス名	*_Class
状態名	*_State
イベント名	*_Event

6.計算式

計算式で使える記号は =(代入), +(和), -(差), *(積), /(商)である。型ごとに使用可能な演算子と、その効果を示す。ただし=はすべての型で使用でき、常に代入の効果である(以下で説明は省略する)。

String	=(代入), +(文字列の結合)
Integer	=(代入), +, -, *, / 通常の算術演算。/は余りを切り捨てる。
Double	=(代入), +, -, *, / 通常の算術演算。
Boolean	=(代入), +(論理和), *(論理積), -(否定) ただし否定の場合、その前までの演算は意味をなさない。 A=B+C*D-E は A=-E と同等。
Vector	=(代入), +(和), -(差), *(ベクトル積)

Instance　　=(代入)
State　　=(代入), 後述。
Event　　=(代入), 後述。

ただし、計算のフォーマットは、次の 2 種類に限定する。
(変数) = (変数・定数)
(変数) = (変数・定数) (=以外の演算子) (変数・定数)
また、下記の場合を除いて、1 つの計算式で用いられる変数・定数の型はすべて一緒である必要があるとする

State の演算
　　State = Instance　インスタンス部分を代入
Event の演算
　　Event = Instance　インスタンス部分を代入

付録 B 手首の位置から腕の姿勢を計算する逆運動学アルゴリズム

逆運動学アルゴリズムは、操作対象に手を伸ばす際の仮想人間の腕の動作を合成するために利用するアルゴリズムである。本研究では、逆運動学アルゴリズムとして、Tolani らの手法 [1] を採用する。以下に、Tolani らの手法の概要を説明する。

人間の腕は非常に複雑であるが、肩に球状の、肘には外巻きの、そして手首にも球状の関節をもつ、7 自由度の機構としてモデル化する。このモデルでは、肩甲骨の動きや前腕の回内運動を無視しているが、視覚的な要求を満たすのには十分な構造である。肩に固定座標系 0 を設定し、各関節に可動座標系 $i(i = 1 \ldots 7)$ を設定する。$\mathbf{A_i}$ を、可動座標系 $i-1$ から可動座標系 i への 4×4 の同次の座標系変換行列として定義する。この座標変換行列は、関節変数 θ_i の関数である。図 B.1 に示したモデルにおいて、$\mathbf{A_i}(i = 1 \ldots 7)$ は式 (B.1)〜式 (B.7) で与えられる。

$$\mathbf{A_1} = \mathbf{R_z}(\theta_1) = \begin{bmatrix} c1 & -s1 & 0 & 0 \\ s1 & c1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.1}$$

$$\mathbf{A_2} = \mathbf{R_x}(\theta_2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c2 & -s2 & 0 \\ 0 & s2 & c2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.2}$$

$$\mathbf{A_3} = \mathbf{R_z}(\theta_3)\mathbf{T}(0, 0, L1) = \begin{bmatrix} c3 & -s3 & 0 & 0 \\ s3 & c3 & 0 & 0 \\ 0 & 0 & 1 & L1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.3}$$

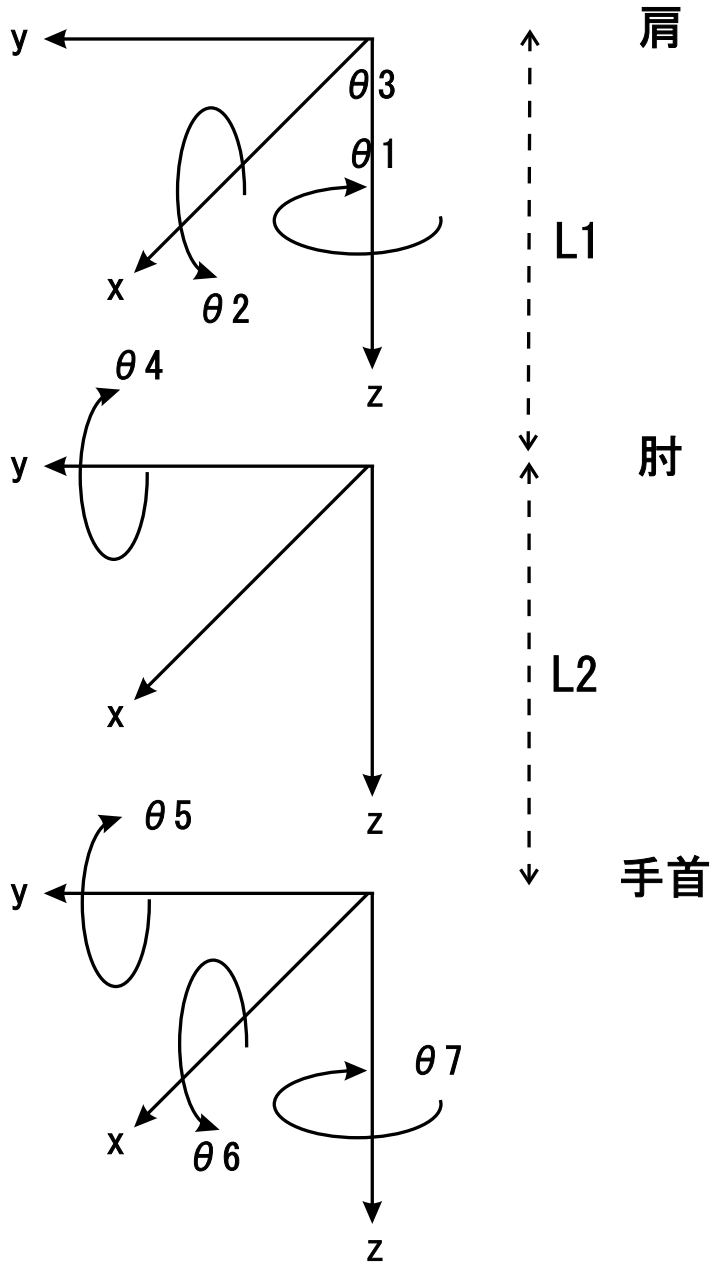


図 B.1: 簡略化された腕のモデル

$$\mathbf{A_4} = \mathbf{R_y}(\theta_4)\mathbf{T}(0,0,L2) = \begin{bmatrix} c4 & 0 & s4 & s4L2 \\ 0 & 1 & 0 & 0 \\ -s4 & 0 & c4 & c4L2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.4}$$

$$\mathbf{A_5} = \mathbf{R_y}(\theta_5) = \begin{bmatrix} c5 & 0 & s5 & 0 \\ 0 & 1 & 0 & 0 \\ -s5 & 0 & c5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.5}$$

$$\mathbf{A_6} = \mathbf{R_x}(\theta_6) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c6 & -s6 & 0 \\ 0 & s6 & c6 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.6}$$

$$\mathbf{A_7} = \mathbf{R_z}(\theta_7) = \begin{bmatrix} c7 & -s7 & 0 & 0 \\ s7 & c7 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.7}$$

ここで、ci と si は、$\cos(\theta_i)$ と $\sin(\theta_i)$ を表し、$L1$ と $L2$ は上腕と下腕の長さを表す定数である。

目的とする手首の座標系の、肩の座標系における位置と角度を表す行列を $\mathbf{A_{wrist}}$ とすると、逆運動学問題は、次式 (B.8) を満たす $\theta_1, \ldots, \theta_7$ の角度の組を導出することである。

$$\mathbf{A_1 A_2 A_3 A_4 A_5 A_6 A_7} = \mathbf{A_{wrist}} \tag{B.8}$$

$\mathbf{A_{wrist}}$ の要素が次式 (B.9) に示すように表せるとすると、

$$\mathbf{A_{wrist}} = \begin{bmatrix} \vartheta 11 & \vartheta 12 & \vartheta 13 & \vartheta 14 \\ \vartheta 21 & \vartheta 22 & \vartheta 23 & \vartheta 24 \\ \vartheta 31 & \vartheta 32 & \vartheta 33 & \vartheta 34 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.9}$$

ベクトル $\mathbf{p} = [\vartheta 14, \vartheta 24, \vartheta 34]^T$ は、固定座標系における手首の位置を表す。手首の位置 $\mathbf{p}$ が決まると、次式 (B.10) によって手首と肩の距離から θ_4 は一意に決定される。

$$\theta_4 = \pi \pm \arccos\left(\frac{L1^2 + L2^2 - \|\mathbf{p}\|^2}{2L1L2}\right) \tag{B.10}$$

θ_4 には２つの解が存在するが、肘の関節が１方向にしか曲がらないことを考慮すると、解は１つに決まる。これより、式 (B.8) において、$\mathbf{A_{wrist}}$ は６つの独立変数によって決まるが、式 (B.8) を満たす $\theta_1, \theta_2, \theta_3, \theta_5, \theta_6, \theta_7$ の組み合わせは無数にある。そこで、有限個の解を導出するために、新たな拘束条件が必要となる。

この拘束条件として、本研究では、肩から手首へのベクトルを中心にした肘の回転角度 (Swivel Angle) ϕ を定める手法を採用する。この手法は Korein ら [2] により提案され、計算機に実装して実行する際の計算量が他の拘束条件と比べて少ないのが特徴である。以下に、Swivel Angle を固定し、$\theta_1, \theta_2, \theta_3, \theta_5, \theta_6, \theta_7$ の値を求める手法について説明する。

図 B.2 に示すように、ベクトル s、e、w をそれぞれ、固定座標系における肩、肘、手首の位置とする。ϕ が変化するに従い、肘の位置は肩から手首に向かうベクトルを法線ベクトルとする平面 N 上の円弧上を移動する。平面 N の法線単位ベクトルを n とすると、ベクトル n は次式 (B.11) で与えられる。

$$n = \frac{w - s}{\|w - s\|} \tag{B.11}$$

また、前腕の可動座標系の z 軸負の方向のベクトルを、平面 N に正射影して得られる単位ベクトル u は次式 (B.12) で与えられる。

$$u = \frac{-z + (z \cdot n)n}{\|-z + (z \cdot n)n\|} \tag{B.12}$$

さらに、平面 N 上の肘の回転中心 c 及び、回転半径 r は、式 (B.13)〜式 (B.17) で与えられる。

$$c = s + \cos(\alpha)L1n \tag{B.13}$$

$$r = L1\sin(\alpha) \tag{B.14}$$

$$\cos(\alpha) = \frac{L2^2 - L1^2 - \|w - s\|^2}{-2L1\|w - s\|} \tag{B.15}$$

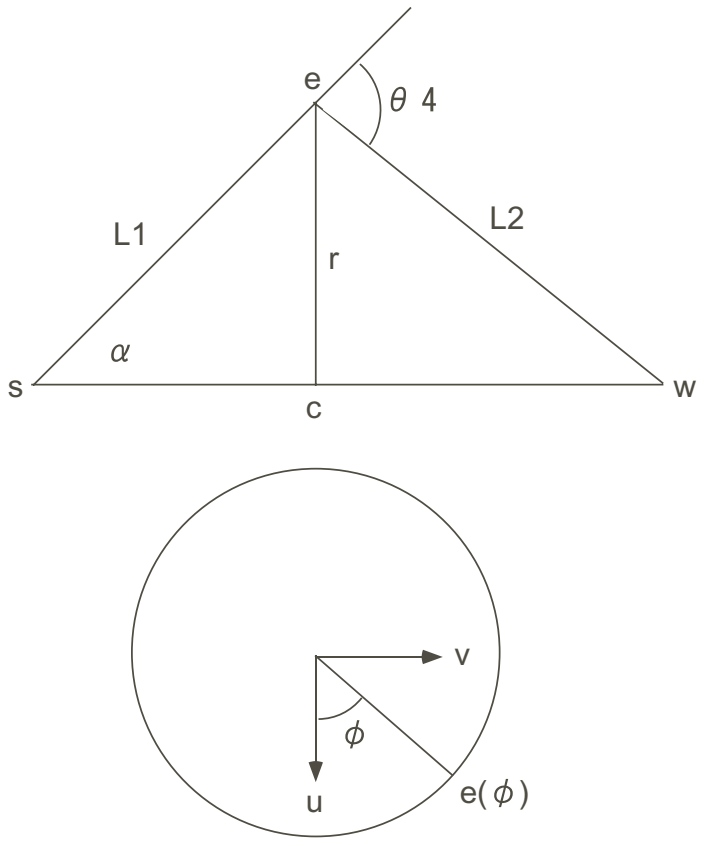


図 B.2: 肩から手首へのベクトルを中心にした肘の回転

$$\sin(\alpha) = \frac{L2\sin(\varphi)}{\|w - s\|} \tag{B.16}$$

$$\varphi = \pi - \theta_4 \tag{B.17}$$

従って、固定座標系における肘の位置 e は ϕ の関数として次式 (B.18) で与えられる。

$$e = r[\cos(\phi)u + \sin(\phi)v] + c \tag{B.18}$$

ただし、ベクトル v は、ベクトル n とベクトル u の外積である。式 (B.18) により、$\phi = 0$ において肘が最も低い位置になり、ϕ が増加するに従って、肘の位置が上昇することが分かる。

ここで適切な ϕ の値を定めると、式 (B.18) を用いてベクトル e を求めることができ、$\theta_1, \theta_2, \theta_3, \theta_5, \theta_6, \theta_7$ を解析的に求めることができる。すなわち、肘の位置は式 (B.19) を展開することにより式 (B.20)〜式 (B.22) が得られ、これらを用いて θ_1 及び θ_2 を求めることができる。

$$\mathbf{A_1 A_2 A_3} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} ex \\ ey \\ ez \\ 1 \end{bmatrix} \tag{B.19}$$

$$\theta_1 = \arctan 2(ex, -ey) \tag{B.20}$$

$$\theta_2 = \arctan 2(s2, \frac{ez}{L1}) \tag{B.21}$$

$$s2 = \begin{cases} \dfrac{-ey}{c1L1} & c1 \neq 0 \\ \dfrac{ex}{s1L1} & otherwise \end{cases} \tag{B.22}$$

また、θ_3 は次式 (B.23) を展開することで式 (B.24) として得ることができる。

$$\mathbf{A_1A_2A_3A_4}\begin{bmatrix}0\\0\\0\\1\end{bmatrix}=\begin{bmatrix}\vartheta 14\\\vartheta 24\\\vartheta 34\\1\end{bmatrix} \tag{B.23}$$

$$\theta_3=\arctan 2(-s1c2\vartheta 14+c1c2\vartheta 24+s2\vartheta 34,c1\vartheta 14+s1\vartheta 24) \tag{B.24}$$

θ_1、θ_2、θ_3 及び θ_4 が決まると、手首の角度は、式 (B.8) を変形することで、次式 (B.25) により計算される。

$$\mathbf{A_5A_6A_7}=(\mathbf{A_1A_2A_3A_4})^{-1}\mathbf{wrist} \tag{B.25}$$

右辺の行列の回転成分を $rij(i=1\ldots3,j=1\ldots3)$ で表し、左辺を展開すると、式 (B.26) に示す９つの等式が得られる。

$$\begin{bmatrix}c5c7+s5s6s7 & -c5s7+s5s6c7 & s5c6\\ c6s7 & c6c7 & -s6\\ -s5c7+c5s6s7 & s5s7+c5s6c7 & c5s6\end{bmatrix}=\begin{bmatrix}r11 & r12 & r13\\ r21 & r22 & r23\\ r31 & r32 & r33\end{bmatrix} \tag{B.26}$$

これより、θ_6 に対して２つの解が求められる。

$$\theta_6=\begin{cases}\arcsin(-r23)\\ -[\pi+\arcsin(-r23)]\end{cases} \tag{B.27}$$

θ_5 と θ_7 は、次式 (B.28) により求められる。

$$\begin{aligned}c6 &\neq 0\begin{cases}\theta_5=\arctan 2\left(\dfrac{r13}{c6},\dfrac{r33}{c6}\right)\\ \theta_7=\arctan 2\left(\dfrac{r21}{c6},\dfrac{r22}{c6}\right)\end{cases}\\ c6 &= 0\begin{cases}\theta_5=\arctan 2(-r31,r11)\\ \theta_7=0\end{cases}\end{aligned} \tag{B.28}$$

参 考 文 献

[1] Deepak Tolani, Norman I. Badler:Real-Time Inverse Kinematics of Human Arm, Presence, **Vol. 5**, No. 4, pp.393-401 (1996).

[2] J. Korein:A Geometric Investigation of Reach, Cambridge, MA, MIT Press (1985).