

特別研究報告書

マルチエージェントインタラクションに
おける並行シナリオの協調

指導教員 石田 亨 教授

京都大学工学部情報学科

田仲 理恵

平成18年2月10日

マルチエージェントインタラクションにおける 並行シナリオの協調

田仲 理恵

内容梗概

人間と音声発話やテキストを介して対話を行うエージェントの研究は様々なところで行われている。仮想空間内に体を持つ対話エージェントの研究では、言語に加えて指差しなどの非言語的動作を行いながら、ユーザと1対1で対話を行い、学習やタスクを達成するエージェントが構築されている。

しかし、現実世界での対話は、店員が複数の客に対応する場合や、グループで話し合いを行う場合のように、1対1ではなく複数人で行う場合が多い。本研究では、複数人を相手にしても問題なく対応ができることを目的とする。

本研究では、仮想都市環境 FreeWalk/Q を用いる。FreeWalk/Q では、インタラクションを記述したシナリオによって、仮想都市空間に配置したエージェントを制御する。シナリオには、相手の行動などインタラクションのきっかけとなるイベントと、イベントが観測された際に行うアクションを記述する。相手が複数になると、相手の数だけのイベントを観測しなければならないため、シナリオの記述が非常に複雑になる。よって、相手ごとにインタラクションシナリオを用意し、複数のシナリオを並行実行する方法を提案する。

複数のシナリオを並行実行すると、以下のような問題が生じる。

1) アクションの実行時に競合が起きる

各シナリオは独立に処理され、互いの内容は考慮されないため、複数のアクションを同時に実行すると競合が起きることがある。競合とは、例えば、言葉を発話するアクションを複数同時に実行するなどの状況を指す。

2) シナリオの協調の仕方が指定できない

複数の相手に対応する方法は様々である。ある相手とのインタラクションが終わるまで別の相手は待たせておく方法もあれば、どの相手も関係なく交互にインタラクションを行う方法もある。シナリオを並行実行するだけでは、どのように協調させるかを指定することができない。

以上の問題点を解決するために、インタラクションシナリオと通信を行うことでアクションの実行を制御する協調シナリオを導入する。制御は、1) 各シナリオは、アクションを実行する前に協調シナリオにメッセージを送り、返答が

あるまで実行せずに待つ，2) 協調シナリオは，アクションを実行して良いか判断して返答する，3) 各シナリオは協調シナリオからの返答に従う，という順序で行う．協調シナリオを導入した上で，問題点を次のような方法で解決した．

1) アクションを調べ，競合が起きないように制御する

複数のアクションを並行実行する際に起きる競合には、アクションを実行する前，実行中，実行後に満たすべき条件に矛盾があるデータ競合と，複数のアクションが同じリソースを用いるリソース競合の2種類がある．競合の解決のため，アクションの定義を用意し，定義を用いてこれらの競合が起こる条件を示した．協調シナリオは，シナリオからメッセージを受け取ると条件を調べ，実行してよいかを判断する．

2) シナリオの協調方法を，ポリシーとして記述可能にした

アクションをどのような場合に実行するかというメタ的な内容を，状態遷移機械モデルを用いて協調ポリシーとして記述する．そして，協調ポリシーを元に協調シナリオを作成する．協調シナリオは，協調ポリシーに従っており，かつ解決方法の1に述べたような競合を検出する機能を持ったシナリオということになる．

本研究では，解決方法を実現するために，協調ポリシーの書式を定め，シナリオコーディネータを実装した．シナリオコーディネータは，協調ポリシーとシナリオを入力すると，協調シナリオと，協調シナリオと通信ができるよう変更したシナリオを自動生成する．生成されたシナリオはFreeWalk/Qでそのまま実行可能な形式である．実装したシナリオコーディネータを用いて，実際に作成した協調ポリシーとシナリオを変更し，得られたシナリオをFreeWalk/Qに読み込ませ，動作の確認を行った．

Coordination of Concurrent Scenarios in Multi-Agent Interaction

Rie TANAKA

Abstract

Many studies about agents which interact with humans by voice utterance or text have been conducted. In the studies about conversational agents, agents which have their bodies in the virtual space and have purposes to learn or achieve tasks have been constructed. They interact with users one-on-one, using verbal and nonverbal communications.

However, interactions in the real world are not only conducted in one-on-one fashion. For example, when a clerk waits on two or more customers, interactions occur between plural persons. This study aims to enable agents to successfully interact with plural persons.

I use FreeWalk/ Q , in which agents in the virtual space are controlled by scenarios which describe interactions. In the scenario, we describe events that trigger interaction, like actions of the person, and actions executed when events are observed. If agents interact with a single person, description of the scenario is not so complicated. If agents interact with two or more person, however, it is very difficult, because agents have to observe all events of all persons. To resolve this difficulty, I propose a way of describing scenarios for each person and executing them concurrently.

When we execute multiple scenarios concurrently, the following problems arise.

1) Conflict occurs in executing actions

Each scenario is analyzed and executed independently and the difference of each effect is not considered. For that reason, conflict may occur when multiple actions are executed at the same time. Conflict means, for example, a case that agent executes multiple utterances concurrently.

2) Lack of method to specify how to coordinate scenarios

There are various ways to interact with plural persons. One way is to keep a person waiting until the interaction with another person finishes. We cannot describe such coordination in interaction scenarios.

In order to solve these problems, I propose the use of coordination scenario, which coordinates interaction scenarios by communicating with each scenario. It controls execution of actions in the following process. 1) Interaction scenarios send messages to the coordination scenario before executing action, and wait for response. 2) The coordination scenario examines whether each action is executable and responds the result to interaction scenarios. 3) Interaction scenarios follow the responses.

The solutions to the problems are presented below using the coordination scenario.

1) Execution of actions were controlled so as not to raise contradictions

There are two kinds of conflict of actions. One is a data conflict, which is a conflict of conditions needed to be satisfied in executing actions. And the other is a resource conflict, which means multiple actions use the same resource. For the purpose of solving conflicts, I defined actions, and showed conditions of these conflict. Coordination scenario examines whether these conditions are fulfilled when it receives a message from interaction scenario, and determines whether the action is executable.

2) A method of coordinating scenarios as coordination policy was proposed

In this method, meta context which specifies when to execute actions is described as the coordination policy. The policy is based on state machine model. The coordination scenario is a scenario which follows policy and has functions to examine competing actions.

In this study, to realize the way to solve problems, I defined commands used in the coordination policy and implemented the scenario coordinator. Inputs of the scenario coordinator are the policy and interaction scenarios, and outputs are the coordination scenario and interaction scenarios which are modified so as to communicate with the coordination scenario. The scenario coordinator automatically generates these output scenarios which are applicable to FreeWalk/ Q . I actually ran the implemented scenario coordinator, applied generated scenarios to FreeWalk/ Q and checked agents' behaviors.

マルチエージェントインタラクションにおける 並行シナリオの協調

目次

第1章	はじめに	1
第2章	シナリオの協調	3
2.1	関連研究	3
2.2	本研究の扱う問題と解決方法	3
2.2.1	本研究の扱う問題	3
2.2.2	問題の解決方法	5
第3章	競合の検出と解決	7
3.1	アクションの定義	7
3.2	競合の条件	9
3.2.1	データ競合の条件	9
3.2.2	リソース競合の条件	11
3.3	検出と解決方法	11
第4章	協調方針の記述	13
4.1	協調ポリシーの書式	13
4.2	インタラクションシナリオの変更	15
第5章	シナリオコーディネータの実装と動作	17
5.1	シナリオコーディネータの動作と構成	17
5.2	シナリオの例	20
5.3	実行結果と考察	23
第6章	おわりに	23
	謝辞	26
	参考文献	26

第1章 はじめに

人間と音声発話やテキストを介して対話を行うエージェントの研究は様々なところで行われている。その中で、仮想空間内に体を持ち、言語に加えて指差しなどの非言語的動作を行う ECAs(Embodied Conversational Agents)[1] の研究も進められている。例えば、言葉やさまざまな表情を用いて不動産の案内をする Rea[2] や、ネットワークについて空間内の物体を指差しながら教えてくれる擬人化学習エージェント Cosmo[3] などがある。また、Extempo¹⁾の Web サイトには個性豊かなエージェントがあり、ユーザはエージェントと対話を行いながらロールプレイやコーチングを行うことにより学習を行うことができる。これらの対話エージェントは、それぞれ学習、タスクなどの目的を持ち、ユーザと1対1で対話を行うことでそれを実現しようとするものである。

1対1ではなく複数の生徒を相手とし、対話を行いながらタスクを達成するエージェントに Steve[4] がいる。Steve は、仮想空間内でユーザの操作するアバター群とともに、チームで一つのタスクを行う。チームのメンバーには役割が与えられており、役割に応じて要所所所であらかじめ定められた発話を行うことで協調作業を行う。

しかし、現実世界での対話は常に1対1とは限らないし、タスクを行う協調作業のように、行動の内容と役割が明確とは限らない。店員が複数の客を相手にするときやグループで話し合いを行うときのように、複数人の間で対話が生じるような状況では、1対1の対話を記述したエージェントも、発話参加者の全員が一つのタスクを行うことを前提として協調方法を持つエージェントも対応ができない。よって、本研究では、複数人の間で対話が生じるときにも問題なく対応ができることを目指す。

本研究は、仮想都市環境 FreeWalk/Q[5] のような、インタラクションプロトコルを状態遷移機械モデルのシナリオで記述し、シナリオによってエージェントを制御するプラットフォームを用いることを想定して行う。シナリオには、インタラクションのきっかけとなる相手の行動や周囲の状況などをイベントとして記述し、イベントを観測した際に行う外界への作用をアクションとして記述する。インタラクションプロトコルが用意されている相手が複数人いる場合、すべての相手に対応できるようにするには、複数のプロトコルを一つにまとめて

¹⁾ Extempo : <http://www.extempo.com/>

記述しなければならず，非常に複雑になる．具体的には，シナリオは状態遷移機械モデルであるので，相手が n 人いて，一人の相手とのインタラクションに平均 m 個の状態がある場合，記述が必要な状態数は m^n という非常に大きな値になる．起こりうる状態の組み合わせはすべて網羅しなければならないためである．よって，記述を容易にするため，相手ごとにインタラクションを記述したシナリオを n 個用意し，それらを並行に実行することを考える．

複数のシナリオを並列実行すると，以下のような問題が生じる．

1) アクションの実行時に競合が起きる

各シナリオは独立に処理され，互いの内容は考慮されないため，複数のアクションを同時に実行すると競合が起きることがある．競合とは，例えば，言葉を発話するアクションを複数同時に実行するなどの状況を指す．

2) シナリオの協調の仕方が指定できない

複数の相手に対応する方法は様々である．ある相手とのインタラクションが終わるまで別の相手とはインタラクションを行わず待たせておく方法もあれば，どの相手も関係なく交互にインタラクションを行う方法もある．複数人との対話を目的としてシナリオを書くときには，対応方法，つまりシナリオの協調方法も指定できなければならないが，並行実行するシナリオを用意するだけでは指定することができない．

本研究では，以上の問題点を解決するために，シナリオと通信を行い同期制御を行う協調シナリオを導入する．そして，そのまま実行すれば問題なく複数人への対応ができるようなシナリオを生成するシナリオコーディネータを実装する．実装の際には，協調する対象を FreeWalk/ Q で用いられる Q シナリオ (Q 言語 [6] で記述されたシナリオ) とする．

以下，第 2 章では本研究の目的であるシナリオの協調について，関連研究を示し，本研究での問題点を整理した上で，協調シナリオを用いて同期を行う方法について説明する．次に第 3 章では，問題点の一つ目であるアクションの競合の解決のために，アクションの定義を与え，定義を用いた競合の条件を述べ，検出と解決方法を述べる．第 4 章では，二つ目の問題点の解決のために導入した協調ポリシーについて内容の定義を与え，協調ポリシーを元に協調シナリオを生成する方法を示す．第 5 章では実装したシナリオコーディネータについて述べ．実際に 1 対 1 対応用のシナリオと協調ポリシーを作成して動作を確認したことについて詳説する．最後に第 6 章で本研究のまとめを行う．

第2章 シナリオの協調

2.1 関連研究

対話エージェントに関する研究はすでに述べたとおりである．本研究では1対1の対話ではなく複数人での対話を目的とし，複数のシナリオを持つ対話エージェントを扱う．

本研究では一人のエージェントが複数のシナリオをもつことを想定しているが，関連研究として，各自一つずつのプランを持った複数のエージェントの協調を行うマルチエージェントプランニングの研究がある [7]．[7] では，プランを持ったエージェントが複数存在するとき，各エージェントのプラン内に通信機能を埋め込むことによって協調を図る方法が述べられている．エージェントのプランを比較して安全性解析を行い，他のエージェントと排他的に行わなければならない部分を検出する．これをプラン中の臨界領域と言う．そして，臨界領域の最初と最後に通信コマンドを発行することによって同期を実現する．通信コマンドは CSP プロセス記述言語 [8] を用いて記述される．同期は，エージェントと同期プログラム間で通信を行い，同期プログラムが他のエージェントのプランが実行されていないときに実行許可を出すことで行う．

本研究で扱うシナリオとプランは同一のものではない．プランは行われるアクションの順番が予め分かっているのに対し，シナリオでは複数のイベントと対応するアクションがあり，イベントが起こりアクションが実行される順番が分からないため，シナリオ全体を調べて臨界領域を設けることができない．しかし，どちらも実行単位がアクションであり，一つのシナリオ・プランで同時に実行されるアクションが常に一つであるなど類似点が多い．よって，同期プログラムを設け，シナリオと通信を行って同期を取るというアイデアを利用する．

2.2 本研究の扱う問題と解決方法

2.2.1 本研究の扱う問題

本研究は FreeWalk/ Q で使用する Q シナリオのように，状態遷移機械モデルで記述されたシナリオを協調することを目的とする． Q シナリオは図 2.1 の形式に従っている．

シナリオには初期状態を含めて複数の状態があり，状態の中に複数のルールがある．ルールはイベントを観測したことを示すキューと呼ばれる適用条件と，

```
シナリオ：(scenario-name (state1 {rule}*)
                (state2 {rule}*)
                .....)
ルール：(cue {action}* {transition})
```

図 2.1: シナリオの形式

適用時に行う任意個のアクション，状態遷移からなる．状態遷移は記述される場合も記述されない場合もあり，記述されない場合はそこでシナリオの実行が終了する．

エージェントは周囲の状況を観測し，現状態に含まれるルールの適用条件が満たされているかどうかを照合する．適用条件が満たされることをキューが発火すると言う．キューが発火するとアクションと状態遷移を行い，新たな状態で再び観測結果とルールを照合する，というプロセスの繰り返しを行う．

問題であるアクションの競合が生じるのは，同じ状況に対し複数のシナリオでキューが発火する場合である．窓口で客に対応し，データと照合して切符を売るエージェントを例にする．エージェントのシナリオ内に，切符があり，購入要求があった場合，切符のデータから 1 減らし、『ありがとうございます』と言って切符を渡すルールがあったとする．全く制御をしなければ，切符が一枚しかないときに二人の客に同時に切符の購入を要求されると，二人に同時に一枚ずつ切符を売ろうとする．また，その後の『ありがとうございます』という言葉で二人に同時に話そうとする．前者は切符は一枚しかないわけであるから起こってはいけない状況であり，後者は物理的に不可能である．シナリオの協調を行うためには，このような競合を検出し，同時に実行できないと判断された場合に実行しないように実行を制御することが必要となる．

複数人とインタラクションを行う際には，競合が起きないようにするだけでなく，相手への対応順序も問題となってくる．現実世界の対応を考えると，声をかけてきた順に一人ずつ対応する場合もあれば，相手に関係なく何か言われるたびに返答する場合もある．店員なら，道を訊いてきた相手より商品について訊いてきた相手を優先したいと思うであろう．そのような対応の仕方，つまりシナリオの協調の仕方をどのように指定するかも問題となってくる．

まとめると，本研究の扱う問題は，1) アクションの競合が検出・解決でき，か

つ2) シナリオの協調方針が指定できるように、複数シナリオの協調方法を考えることである。

2.2.2 問題の解決方法

シナリオの協調のために、シナリオと通信を行い同期制御を行う協調シナリオを導入する。制御対象はアクションであり、協調シナリオはアクションを実行する前に逐一実行命令を出す。この協調シナリオを用いて二つの問題を解決する。

一つ目の問題の解決は、アクションの競合を定義しておき、協調シナリオが競合が起きているかどうか調べ、結果にもとづいて命令を出すことで行う。

二つ目の問題の解決のために、シナリオの協調方針を協調ポリシーとして記述できるようにする。協調シナリオは、アクションの競合を検出する機能を持っており、かつ協調ポリシーに従うものとなる。

概念図を図 2.2 に示す。図中のシナリオコーディネータは、1対1の対応用に記述されたインタラクションシナリオと協調ポリシーを元に、協調シナリオと、通信が可能なように変更したインタラクションシナリオを自動生成する。変更されたインタラクションシナリオを変更後シナリオと呼ぶことにする。

通信による制御プロセスを以下に示す。キューが発火するたびに、変更後シナリオと協調シナリオの間で同じプロセスが繰り返される。

まず、変更後シナリオは、アクションを実行する前に協調シナリオに実行許可依頼をメッセージとして送り、命令が送られてくるまでアクションを実行せずに待つ。協調シナリオは、送られてきた実行許可依頼メッセージに対し、アクションの競合を調べ、実行するかどうかを判断して結果をメッセージとして送る。変更後シナリオは受信したメッセージに従う。

協調シナリオは、競合が起きなければ実行するようメッセージを、競合が起きる場合には実行しないようメッセージを送る。変更後シナリオはそれに従うが、後者の場合にはただ実行しなければ良いわけではない。一度発火したキューはいずれかのタイミングで実行されなければいけないからである。しかし、インタラクションの相手の行動はエージェントにとって予想できないものである。実行できるようになるまで待ち続けると、その間に相手は何からの行動を起こして状況が変わってしまう恐れがある。そのため、実行しないよう命令が送られてきた場合には、キューが発火したことを記憶しておいて何もせず元の状態に戻る。元の状態で新たなキューが発火すれば対応するアクションの実

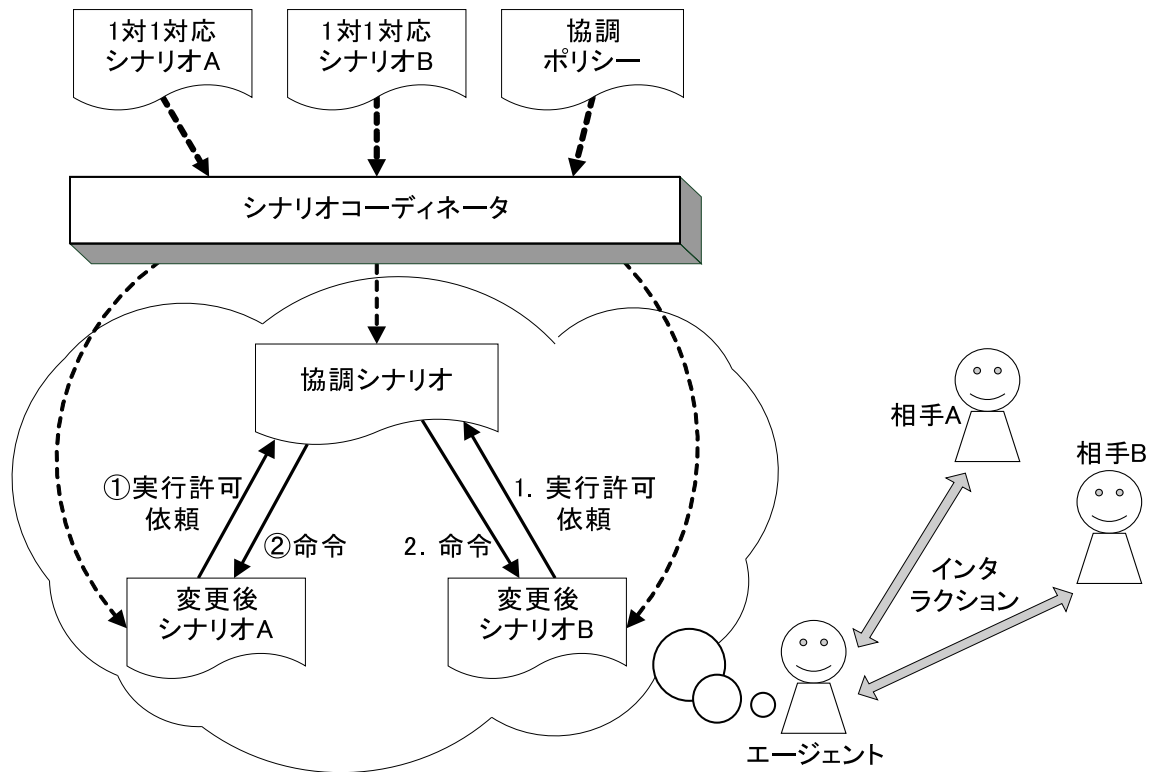


図 2.2: シナリオの協調の概念図

行許可依頼を送り，発火しなければ記憶しておいたキューに対する実行許可依頼を再度送ることで，新たな状況にも対応でき，新たな状況が生じなければ発火したキューが実行できるようになる．

キューを記憶するのは次のような理由による．インタラクションに使用されるキューの中には，相手のいる場所を観測するなど何度でも発火するキューと，言葉を聞き取るなど一度しか発火しないキューがある．後者のようなキューは，一度発火すると，再度元の状態に戻って観測しても発火しないため，記憶が必要になる．

以上のような処理が出来るようにシナリオコーディネータによって自動生成された変更後シナリオと協調シナリオをエージェントに与えると，エージェントはおのこのシナリオを並行実行する．キューが発火するかどうか変更後シナリオ上で観測しながら，同時にアクションの実行許可依頼が送られていないか協調シナリオ上で観測を行う．そしてキューが発火したときは自分自身にメッセージを送り，判断してメッセージを自分自身に送り返し，それに従ってアク

ションを実行することになる．人間の思考プロセスにたとえて言えば，相手が来て対応を考える部分が変更後シナリオ，どの相手から対応するかを決めるのが協調シナリオということである．

第3章 競合の検出と解決

複数のアクションの競合に関しては，プロダクションシステムにおいて，高速化を図るためルールの並列実行を行う際に起こる競合について研究が行われている [9]．プロダクションシステムのルールの動作部は，ルールの実行に必要なデータを追加・削除するものであり，条件部が満たされるとすぐに実行が終了する．これに対し，インタラクションに使用されるルールのアクションは，言葉を話すものや相手の前まで歩いていくものなど，一定時間継続して行われるものが多く，プロダクションシステムで定義されている競合の条件だけでは不十分である．また，実行に手や足などのリソースを使用するため，プロダクションシステムにはなかったリソースの競合と言う新たな競合も生じる．

本章では，プロダクションシステムでの競合の定義を参考にして，まず競合の検出のためにアクションの定義を与え，それをを用いて本研究で生じる競合の条件を述べ，検出と解決方法について説明する．

3.1 アクションの定義

各アクションを，前提条件，事後条件，継続条件，使用するリソースで定義する．前提条件はアクションの実行前に満たされているべき条件，事後条件は実行後に満たされているべき条件である．『歩く』『話す』と言ったアクションは一定時間継続して実行されるため，継続条件にはその間に満たされているべき条件を記述する．リソースはアクションの実行中に使用する資源のことで，行動に関するアクションであれば手や足などがリソースとなる．

形式的な定義を図 3.1 に示す．条件の各要素 condition は，立ち止まっている，歩いている，相手 A の前にいる，などの基本的な状態で，リソースの各要素 resource は手や足などの基本的なリソースである．アクションの条件とリソースは，基本となる状態・リソースの組み合わせで定義する．図 3.1 ではすべて連言で定義しているが，アクションの実行系によって『右手または左手の開いているほうを検出して使用』するようにアクションが実装されている場合な

どには、選言を用いて定義する。

この形式で、指定した方向を向くアクション `turn` と指定したルートを歩くアクション `walk` を定義したものが図 3.2 である。

アクションの競合検出は、定義した各条件が矛盾するかどうかを元に行う。そのために、条件が矛盾すると言うのはどういうことかを定義しておく。

状態を表すリテラルには、『立っている』のように変数を含まないものと、『相手 X の前にいる』のように変数を含むものの二種類がある。リテラル A と B が矛盾するのは、以下のいずれかが成り立つときである。

- (条件 1) A と B がともに変数を含む同じリテラルで、変数が異なっている
- (条件 2) A と B がともに変数を含まないリテラル、または一方が変数を含むリテラルで、相反する意味を持つ

```
(def_cond_of_action action-name
  :pre-condition condition condition ...
  :post-condition condition condition ...
  :during-condition condition condition ...
  :resource resource resource ...)
```

図 3.1: アクションの定義

```
(def_cond_of_action turn
  :pre-condition notmotion
  :post-condition notmotion toward(ID)
  :during-condition motion
  :resource leg)
(def_cond_of_action walk
  :pre-condition notmotion
  :post-condition notmotion position(ID)
  :during-condition motion
  :resource leg)
```

図 3.2: アクションの定義例

条件2については、アクションの条件の定義は実行系によって異なるので、条件を定義する際に、相反する意味を持つとみなす条件の組を作成しておき、それと照らし合わせることで矛盾を判断する。

アクションの実行制御は、一つのキューに対し記述されたアクションをまとめたアクションブロックの単位で行う。一つの事象を観測し、対応する複数のアクションを行うことが実行の単位になっているからである。よって、図3.1の形式で定義されたアクションの前提条件・事後条件・継続条件・リソースを元に、アクションブロックの前提条件・事後条件・継続条件・リソースを作成する。

前提条件は、基本的には各アクションの前提条件の連言をとる。ただし、先行するアクションの事後条件によって後続アクションの前提条件が満たされる場合、後続アクションの前提条件はアクションブロックの前提条件には加えない。また、アクションブロックの前提条件はアクションブロック全体を実行する前の前提条件であるから、前提条件同士で矛盾するものがあつた場合には、先に実行されるアクションのものを採用する。たとえば、前提条件がそれぞれ『相手Aのところにいる』と『場所Bのところにいる』であれば、『相手Aのところにいる』を採用する。

事後条件も各アクションの事後条件の連言を取り、矛盾があつた場合には後に実行されるアクションのものを採用する。ただし、FreeWalk/Qのように、終了を待ってから次のアクションを実行する同期アクションと終了を待たない非同期アクションが存在する場合、非同期アクションについては事後条件の代わりに継続条件を用いる。

継続条件も同様に連言を取り、矛盾があつた場合には後に実行されるアクションのものを採用する。

リソースには矛盾は生じないので、各アクションの連言を取ればよい。

このようにして記述されたアクションブロックについて、次節に示す二種類の競合を検出する。これ以降、単にアクションと記述された場合はアクションブロックのことを指すこととする。

3.2 競合の条件

3.2.1 データ競合の条件

アクションの競合の一つ目はデータ競合である。複数のアクション間にデータ競合が起こるとは、アクションの条件に着目して考えたとき、並行に実行す

ると矛盾する状態を帰結することを言う。条件の連言が矛盾するとは、連言を構成する基本条件要素の中に矛盾するリテラルの組が一組以上あることを示す。データ競合の条件を以下のように定義する。

[定義] データ競合の条件

アクション A, B があり, それぞれの前提条件の連言を pre-A, pre-B, 事後条件の連言を post-A, post-B, 継続条件の連言を during-A, during-B と表すとき, アクション A と B が並行実行できないのは, 以下のいずれかが満たされる場合である。

- (条件 1) post-A と post-B が矛盾する
- (条件 2) post-A と pre-B が矛盾する, かつ post-B と pre-A が矛盾する
- (条件 3) during-A と during-B が矛盾する
- (条件 4) post-A と during-B が矛盾する, または post-B と during-A が矛盾する
- (条件 5) during-A と pre-B が矛盾する, かつ during-B と pre-A が矛盾する

各条件について説明する。

まず, 条件 1 と条件 2 は, ルールの干渉 (interference) を示すものである。干渉とは, 複数のルールを並行実行させたとき, 逐次的にいかなる順番で実行した結果とも異なる結果が得られることを言う。並行実行する際には, 並行実行の結果がいずれかの順で逐次実行された結果と一致する直列化可能性が保障されていなければならない。条件 1 が満たされているとき, 逐次実行を行えば, 実行後の状態は後に実行したアクションの事後条件となる。しかし, 並列実行を行うと, 矛盾が生じるために, どちらのアクションの事後条件とも異なる結果になる可能性がある。次に, 条件 2 が満たされているとき, 逐次実行を行うと, 先行アクションの事後条件と後続アクションの前提条件が矛盾するために後続アクションが実行できない。並行実行を行うと, 結果は両方のアクションを実行した結果となるので, 逐次実行の結果と異なる可能性がある。

条件 3 は, 実行中に満たすべき条件に矛盾があるアクションは並行に実行できないことを示す。

条件 4 は, 先に終了したアクションの事後条件と, まだ終了していないアクションの継続条件が矛盾してはいけないことを示す。どちらが先に終了するか

は分からないため、A が先に終了した場合、B が先に終了した場合の両方を考え、いずれかに矛盾があれば競合とみなすこととする。

条件 5 は、すでに実行を開始しているアクションの継続条件と、新たに開始しようとしているアクションの前提条件が矛盾してはいけないことを示す。

3.2.2 リソース競合の条件

競合の二つ目はリソース競合である。リソース競合は、複数のアクションが同じ資源を使用する場合には並行、つまり同時に実行できないことを言う。例えば、『右を向く』と『左を向く』は、どちらも体という資源を使用するため、並行に実行することはできない。リソース競合の条件は以下の通りである。

[定義] リソース競合の条件

アクション A, B があり、それぞれのリソースの連言が resource-A, resource-B と表されるとき、アクション A と B が競合するのは以下の場合である。

(条件) resource-A と resource-B の中に共通のリソースが一つ以上存在する

3.3 検出と解決方法

競合の検出は、協調シナリオが実行時に行う。解決方法は、2.2.2 節に述べたように、競合が起こる場合は実行しないようにすることである。協調シナリオはアクションの定義データを持っており、アクションの実行許可依頼が送られてくるたびに競合を調べ、結果に従って命令を返信する。

実行許可依頼を送ってきたアクションと競合するかどうか調べる対象は、すでに実行中のアクションである。協調シナリオはまず、実行中のアクションの集合を取得する。集合が空ならば、つまり実行中のアクションがなければ、依頼されたアクションに実行命令を出す。空でないときの処理は次のようになる。

実行許可依頼を送ってきたアクションを X, 実行中のアクションの集合を $A = \{A_1, A_2, \dots, A_n\}$ とする。X と集合 A 中の各々のアクション $A_i (i=1, 2, \dots, n)$ について、図 3.3 に示すアルゴリズムで競合が起きているかどうかを調べる。

データ競合の条件 5 に対応する部分については、すでに実行開始しているアクション A_i の継続条件とこれから開始するアクション X の前提条件のみを調べればよく、 A_i の前提条件と X の継続条件を調べる必要はない。

集合 A 中のすべてのアクションについて結果が偽となれば、競合は起きず並行実行可能であるので実行命令を出す。いずれかのアクションについて結果が

真となれば，競合が起き並行実行不可能であるので，その時点で集合中の残りのアクションについては調べるのをやめ，実行しないよう命令を出す．

実行許可を依頼されたアクションを X ，実行中のアクションを A_i とする．アクション X の前提条件を $\text{pre-}X$ ，事後条件を $\text{post-}X$ ，継続条件を $\text{during-}X$ ，リソースを $\text{resource-}X$ と表す．条件，リソースはそれぞれ基本的な条件，リソースの連言である．

条件の連言が矛盾するとは，条件を構成する基本条件要素の中に矛盾するものが一組以上あることを言う．基本条件要素の矛盾は，以下のいずれかを満たすことを示す．

- (1) 条件がともに変数を含む同じリテラルで，変数が異なっている
- (2) 条件がともに変数を含まないリテラル，または一方が変数を含むリテラルで，相反する意味を持つ

```
if
  post- $X$  と post- $A_i$  が矛盾する
  or
  post- $X$  と pre- $A_i$  が矛盾する and pre- $X$  と post- $A_i$  が矛盾する
  or
  during- $X$  と during- $A_i$  が矛盾する
  or
  post- $X$  と during- $A_i$  が矛盾する or during- $X$  と post- $A_i$  が矛盾する
  or
  pre- $X$  と during- $A_i$  が矛盾する
  or
  resource- $X$  と resource- $A_i$  の中に共通のリソースが一つ以上存在する
then
  結果を真とする
else
  結果を偽とする
```

図 3.3: アクションの競合検出アルゴリズム

第4章 協調方針の記述

本章では，シナリオの協調方針を記述する協調ポリシーの書式と，協調ポリシーを元に協調シナリオを生成する方法について述べる．また，協調シナリオと通信を行えるように，1対1の対応用のインタラクションシナリオに通信機能を付加する方法についても述べる．

4.1 協調ポリシーの書式

協調ポリシーは状態遷移機械モデルを用いて記述する．状態遷移機械モデルにするのは次のような理由による．協調ポリシーに記述する内容は，インタラクションシナリオ内の個々のアクションに関する内容ではなく，アクションを制御するものである．アクションの制御は，協調シナリオとインタラクションシナリオとの間でやり取りされるメッセージを用いて行われる．つまり，協調ポリシーには，インタラクションシナリオからメッセージを受信したことを観測し，それに対してどのようなメッセージを返信するかを記述する．記述するレベルがインタラクションシナリオよりメタレベルになっただけで，周囲の状況を観測しアクションを行うシナリオと考え方の根本は変わらない．よって，シナリオと同じ状態遷移機械モデルを用いるのが良いと考えた．協調ポリシーの状態としては，アクションの実行許可依頼を観測するとすべてに実行命令を出す状態，常に一つのアクションしか実行できないように制御する状態，常に同じシナリオのアクションにのみ実行命令を出す状態などが記述でき，これらを組み合わせることで協調の方針を記述する．

協調ポリシーもシナリオと同様に，キュー，アクションおよび状態遷移で記述する．協調ポリシーで用いるキューとアクションを定義する前に，まず，シナリオ間の通信で用いられるメッセージを定義する．

インタラクションシナリオから協調シナリオへ送られるメッセージ（依頼メッセージと呼ぶ）は，アクション実行許可依頼，アクション終了報告，待機報告，シナリオ終了報告の4種類である．これに対し，協調シナリオからインタラクションシナリオに送られるメッセージ（返答メッセージと呼ぶ）は，実行命令，取消命令，待機命令の3種類である．メッセージの意味を表4.1に示す．

メッセージ中の待機処理は，相手に『お待ちください』と言うなどして，待っていてもらうための処理である．協調ポリシーの最も単純な記述は，一状態で

表 4.1: 通信に用いるメッセージの意味

	メッセージ	意味
依頼 メッセージ	実行許可依頼	キューが発火し、実行できるアクションがあることを伝える
	アクション終了報告	アクションが終了したことを伝える
	待機報告	アクションの待機処理が終了したことを伝える
	シナリオ終了報告	シナリオが終了したことを伝える
返答 メッセージ	実行命令	アクションを実行するよう伝える
	取消命令	アクションを実行せず元の状態に遷移するよう伝える
	待機命令	アクションを実行せず待機処理をするよう伝える

『実行許可依頼があれば実行し、元の状態に遷移する』もので、その場合には待機処理は必要ない。しかし、ある相手とのインタラクション中は他の相手を待たせておくなどのポリシーを記述する際、より自然なインタラクションのために、待っていてもらうよう伝えたい場合もある。待機処理はそのために用意した処理である。具体的にどの様な言葉を話し行動をするのかは協調ポリシーとは別に記述する。

メッセージの定義を元にした、協調ポリシーで使用されるキューとアクションを表 4.2 に示す。協調ポリシーはシナリオと同じ形式で記述されるため、協調ポリシーから協調シナリオを作成する際には、キューとアクションを実行可能なコマンドに変換するだけでよい。ただし、協調ポリシーにはアクションの競合に関する内容は記述されないため、実行命令を送るアクション(!SendPermit :action \$action) については補足が必要である。実行許可依頼に対し実行命令を出すときは、図 3.3 に示したアルゴリズムに従って、並行実行可能な場合には実行命令を、不可能な場合には取消命令を送る。取消命令を送るときには、送ってすぐに元の状態に戻るようにする。アクションの後に状態遷移が記述されている場合は、協調ポリシーとしては実行命令を送って次の状態に遷移したいということであるが、実行命令を送らなかった場合は、次の状態に遷移してはいけなからである。変更後シナリオからはアクションが実行できるようになるまで実行許可依頼が送られ続けるので、いずれは実行命令を送ることができるよう

表 4.2: 協調ポリシーで使用されるキューとアクション

	書式	意味
キュー	(?GetRequest :action \$action)	アクションの実行許可依頼を観測した
	(?GetActionEnd :action \$action)	アクションの終了報告を観測した
	(?GetWait :action \$action)	アクションの待機報告を観測した
	(?GetScenarioEnd :scenario \$scenario)	シナリオの終了報告を観測した
アクション	(!SendPermit :action \$action)	アクションに実行命令を送る
	(!SendCancel :action \$action)	アクションに取消命令を送る
	(!SendWait :action \$action)	アクションに待機命令を送る

になり、そのとき初めて次状態に遷移する。待機報告に対し実行命令を出すときは、インタラクションシナリオは待機報告を送った状態で実行命令を待ち続けているので、実行再開するアクションが実行中のアクションと競合しなくなるまで待ってから実行命令を送る。

4.2 インタラクションシナリオの変更

インタラクションシナリオには、複数の状態内に複数のルールが記述されており、各ルールはキューとアクション、状態遷移で構成されている。シナリオの協調を行うためには、各ルールに通信を行うための機能、および受信したメッセージに従った処理を追加する。また、第2章に述べたように、受信したメッセージが取り消し命令であった場合には、キューが発火したことを記憶して元の状態に戻る処理を記述する。そして、状態内のどのキューも発火しなければ記憶したキューに対する処理を行うような記述を追加する。

追加の例を示す。図 4.1 は 1 対 1 対応用に記述されたインタラクションシナリオの中のある状態で、機能が追加される前の通常の Q シナリオの記述である。状態 State1 において、cue1 が観測されると action1 と transition1 を行う。cue1 が観測されなければそれで終わりなのではなく、cue1 が観測されるまで

```
(State1
  (cue1 action1 transition1))
```

図 4.1: インタラクションシナリオ内の状態

```
(State1
  (cue1
    (協調シナリオに実行許可依頼を送る)
    (返答を待つ)
    (返答が『実行命令』のとき
      (action1 を実行する)
      (協調シナリオにアクション終了報告を送る)
      (transition1 を実行する))
    (返答が『待機命令』のとき
      (『お待ちください』と言う等, 待たせるための処理をする)
      (協調シナリオに待機報告を送る)
      (実行命令が送られてくるのを待つ)
      (『お待たせしました』と言う等, 待たせた後の処理をする)
      (action1 を実行する)
      (協調シナリオにアクション終了報告を送る)
      (transition1 を実行する))
    (返答が『取消命令』のとき
      (cue1 が発火したことを記憶する)
      (協調シナリオにアクション終了報告を送る)
      (元の状態 State1 に遷移する)))
  (otherwise
    (cue1 が発火したと記憶されているとき
      (cue1 が発火したときと同じ処理を行う))
    (そうでなければ
      (State1 に遷移する))))
```

図 4.2: 機能を追加した変更後シナリオ内の状態

観測を続ける。

機能を追加した状態を図 4.2 に示す。cue1 が観測されると、実行許可依頼を送り、メッセージを受信するまで待ち、受信したメッセージの種類によって条件分岐する。最後の otherwise 節は、cue1 が観測されなかったときに実行する文を記述する節で、cue1 が発火したと記憶されているときは cue1 が発火したときと同じ処理を行う。そうでないときは、cue1 が発火するまで観測を行わなければならないので、元の状態 State1 に遷移する。

第5章 シナリオコーディネータの実装と動作

シナリオコーディネータの実装は Scheme を用いて行った。これは、解析の対象となるシナリオが Scheme を母言語とする Q 言語 [6] で書かれているためである。また、実装に使用した計算機の性能は表 5.1 の通りである。

本章では、実装したシナリオコーディネータについて、具体例を挙げながら動作を説明する。

5.1 シナリオコーディネータの動作と構成

シナリオコーディネータの入力ファイルと出力ファイルは表 5.2 の通りである。

policy.q には、協調ポリシー、および待機処理を行う際の前処理と後処理を記述する。前処理とは、待機状態に入る前に相手に対して行う動作のことで、相手の方を向いて『お待ちください』と言う動作などを指定する。後処理とは、待機状態が終わり実行再開するときに行う動作のことで、相手の方を向いて『お

表 5.1: 使用した計算機の性能

OS	WindowsXP
CPU	Penitum4 3.0GHz
メモリ	2.0GB

表 5.2: シナリオコーディネータの入出力ファイル

入力ファイル	出力ファイル
<ul style="list-style-type: none">1対1対応シナリオ (Qシナリオ)policy.qusefiles.q	<ul style="list-style-type: none">変更後シナリオ (Qシナリオ)協調シナリオ (Coordination.q)

待たせました』と言う動作などを指定する．指定しなければ，何もせずに待機状態に入り，何もせずに実行を再開する．

最も簡単な協調ポリシーは，4.1 節にも述べたように，一状態で『アクションの実行許可依頼があれば実行する』ものであるが，それ以外には相手への対応の型として，一人の相手に続けて対応し，別の相手が来た場合も対応が終わるまで待たせておく逐次型，呼びかけられるたびに相手に関係なく応答する八方美人型などが考えられる．典型的な型はコーディネータ内部に用意しており，使用する型を指定するだけでキューやアクションを用いた記述をしなくても協調が可能である．

また，シナリオ内の協調を行わない状態を指定することができる．詳しくは次節で紹介するシナリオの例を用いて後述する．

usefiles.q には，シナリオコーディネータがシナリオを認識するために，読み込むシナリオを記述したファイル名を列挙する．与えるファイルのうちエージェントの定義部分にのみ変更が必要で，シナリオの協調を行いたいエージェントの読み込むファイルのリストに Coordination を加えておく．すると，シナリオコーディネータはエージェント定義ファイルから対象となるエージェントを検出し，列挙されたファイル内から協調を行うシナリオのみを抽出して処理を行う．シナリオからシナリオが呼び出される場合は，エージェントの定義には最初に読み込まれるシナリオしか列挙されていないため，最初に呼び出されるシナリオを元にシナリオの呼び出し関係を解析し，協調が必要なシナリオをすべて抽出する．

シナリオコーディネータの構成図は図 5.1 のようになっている．内部はシナリオの読み込み部，解析部，競合検出部，協調シナリオ作成部，インタラクションシナリオ変更部からなる．各部は図 5.2 のフローチャートに示した順序で実行される．シナリオ解析の際にはアクションの定義データを，協調シナリオ作成の際には典型的な協調ポリシーの定義データとシナリオ変換データを用いる．シナリオ変換データには，協調ポリシーのキューとアクションを実行可能な Scheme のコマンドに変換する変換方法が記述されている．

シナリオコーディネータにより生成されたシナリオを FreeWalk/Q で実行する際のシステム構成図を図 5.3 に示す．シナリオコーディネータは FreeWalk/Q とは独立している．Q インタプリタは生成されたシナリオを解析し，内容に従って FreeWalk のエージェントにキューの観測依頼とアクションの実行依頼を出す．

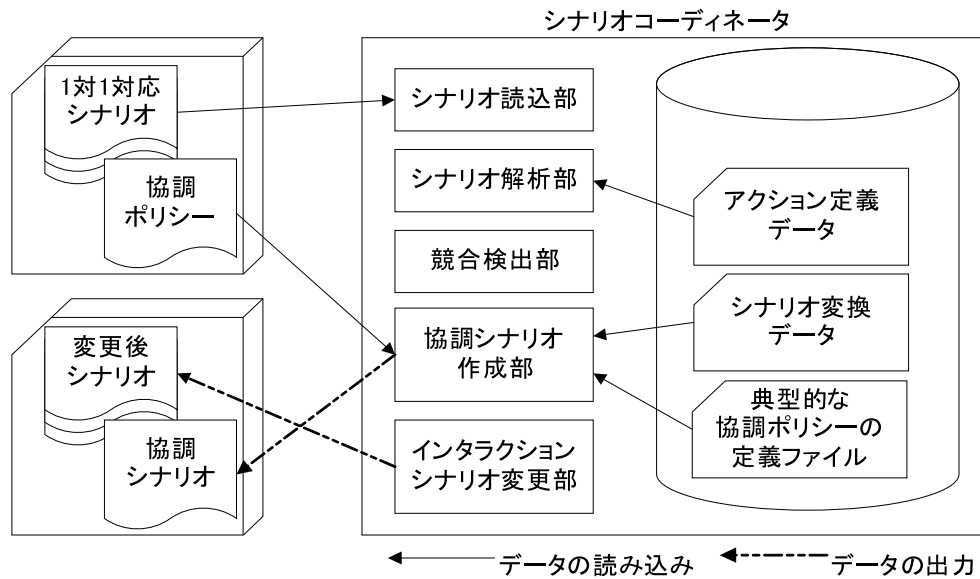


図 5.1: シナリオコーディネータの構成図

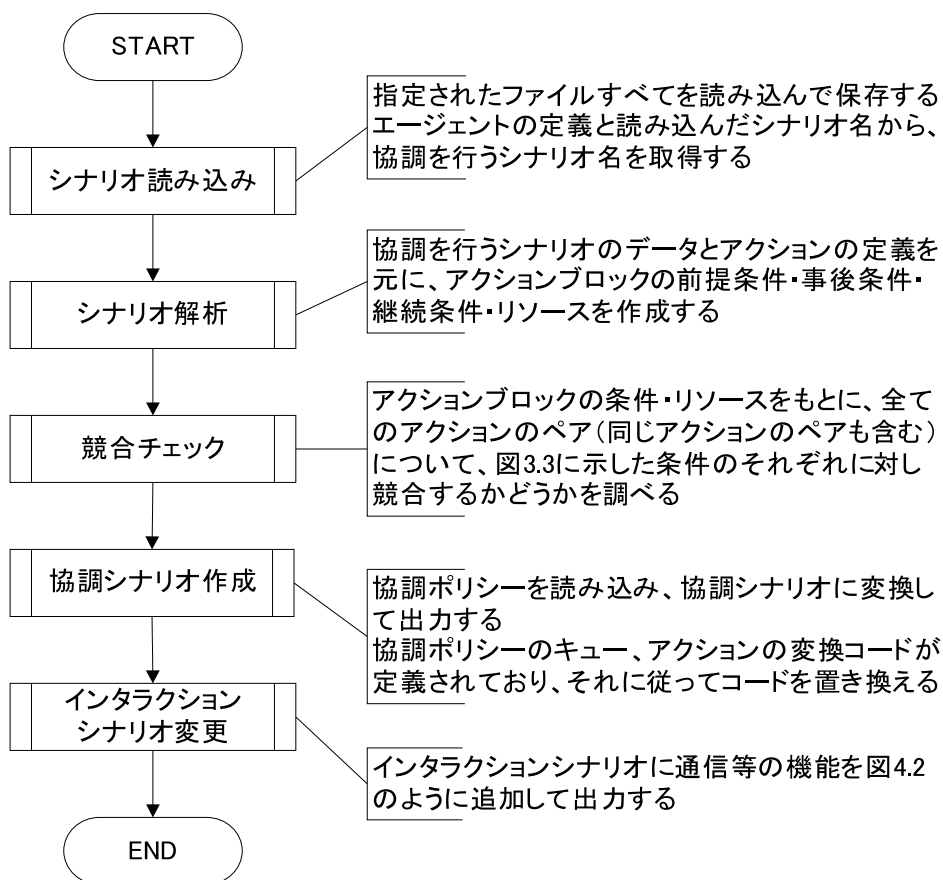


図 5.2: シナリオコーディネータの処理フロー

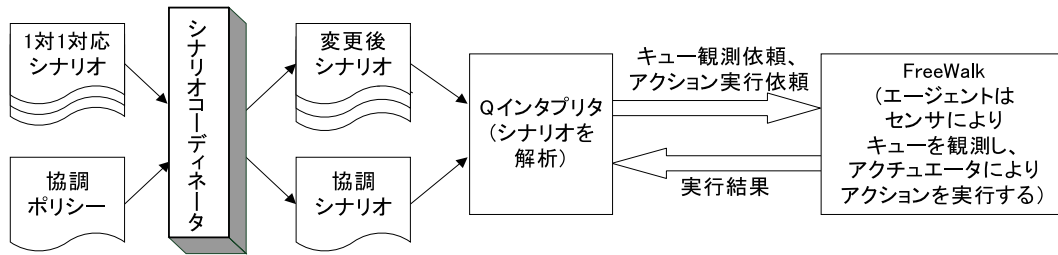
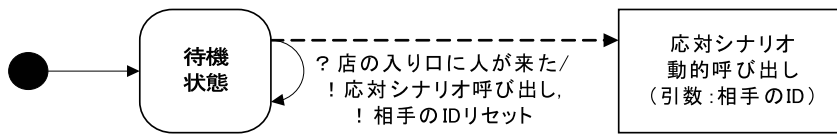


図 5.3: システム全体の構成図

<メインシナリオ>



<応対シナリオ>

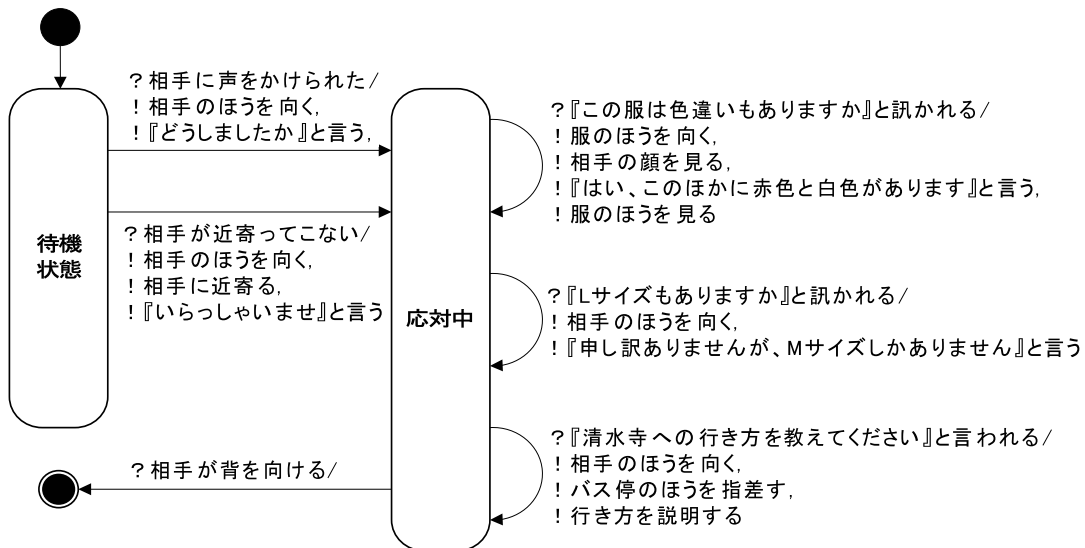


図 5.4: 店員エージェントの1対1対応シナリオの状態遷移図

5.2 シナリオの例

例として、複数の客に対応する店員エージェントのシナリオを生成する。図 5.4 は、一人の客に対してどのように対応するかを記述したインタラクティブシナリオの状態遷移図である。

シナリオからシナリオを呼び出す際、動的呼び出しを行っており、新たに呼び出されたシナリオと元のシナリオは別スレッドで並行して実行される。動的呼び出しを用いるので、最初にメインシナリオを読み込ませておけば、客が新

たに来るたびに対応シナリオが呼び出される．協調シナリオは相手の数に関係なく，実行許可依頼を送ってきたアクションと実行中のアクションが競合するかどうかを調べるので，相手が何人来ても対応が可能である．

対応のメインとなる対応シナリオでは，客から声をかけられれば対応中に遷移する．また，店内の客が近寄ってこない場合にも声をかけて対応中に遷移する．対応中ではいくつかの質問をされれば返答し，客が背を向けて店から出て行こうとすれば対応を終了する．この例では簡単のため，3種類の質問のみに返答できるようにしてあるが，質問の種類を増やしてバリエーションをもたせることも可能である．

次に協調ポリシーを作成する．協調ポリシーは，5.1節で例としてあげた逐次型を採用する．状態遷移図を図5.5に示す．

逐次型の協調ポリシーは，実行中のシナリオ以外のシナリオは待機させるという方針で記述する．図5.5では，準備状態からアクションの実行許可依頼を観測すると実行命令を出して対応中に遷移し，対応中に別のシナリオから依頼があると待機命令を出している．待機命令を出したシナリオは順番に記憶されており，実行中のシナリオが終了したときには，最も早く待機命令を出したシナリオに実行命令を送って実行を再開する．キューの観測には記述した順に優先順位がつくようにしてあり，新たに実行許可依頼を送ってきたシナリオよりも待機中のシナリオが優先して実行される．

この協調ポリシーを用いて，例えば図5.4のインタラクションシナリオの協調を行うと，最初にメインシナリオが実行された時点で協調シナリオは対応中

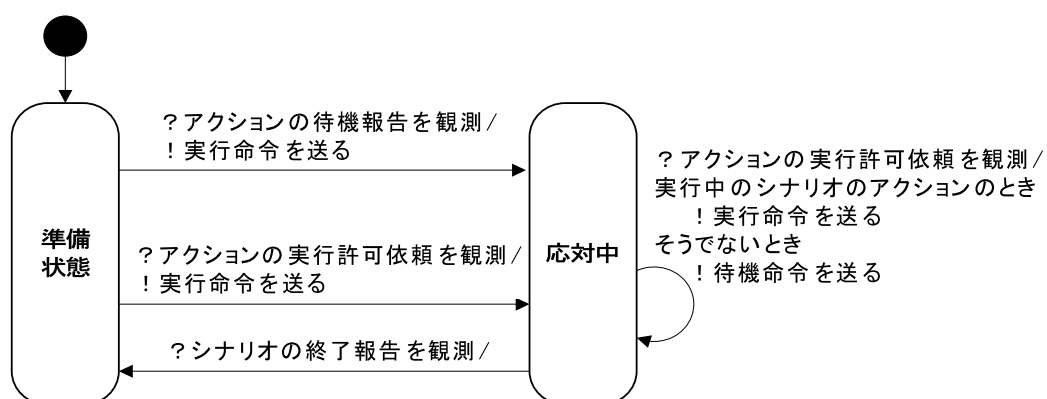


図 5.5: 逐次型の協調ポリシーの状態遷移図

に遷移する。すると、動的呼び出しされた対応シナリオは、メインシナリオとは別のシナリオであるので、初期状態から遷移する段階で待機命令が出される。メインシナリオは終了することがないので、対応シナリオは待機状態に入ったまま永遠に待ち続けることになる。つまり、客は店に来ただけで『お待ちください』と言われ、待たされ続けることになる。インタラクションとしては問題があるため、5.1 節に示したように、協調を行わない状態を指定して協調ポリシーと組み合わせることで、簡単に調整が行えるようになっている。

図 5.5 の状態遷移図をシナリオコーディネータに読み込めるシナリオの形式に変換したものが図 5.6 である。状態遷移図の遷移の枝をそのまま、表 4.2 で定義したアクションとキューを使用するよう書き換えている。実行許可依頼を送ってきたアクションが実行中のシナリオのものかどうかを判断する条件文に関しては、そのような機能を持つ関数 (`active?`) がシナリオコーディネータの内部に定義されており、それをを用いている。

```
(defpolicy Deal-one-by-one
  (Prepare-State
    ((?GetWait :action $action) ; 待機報告を観測
     (!SendPermit :action $action) ; 実行命令を送る
     (go Reception-State))
    ((?GetRequest :action $action) ; 実行許可依頼を観測
     (!SendPermit :action $action)
     (go Reception-State)))
  (Reception-State
    ((?GetRequest :action $action)
     (if (active? $action) ; 実行中のシナリオのアクションであれば
         (!SendPermit :action $action)
         (!SendWait :action $action)) ; 待機命令を送る
     (go Reception-State))
    ((?GetScenarioEnd :scenario $scenario) ; シナリオ終了報告を観測
     (go Prepare-State))))
```

図 5.6: 協調ポリシー (シナリオ形式)

5.3 実行結果と考察

図 5.4 に示したシナリオの状態遷移図を Q シナリオとして記述したものと、図 5.6 に示した協調ポリシーをシナリオコーディネータに入力し、得られた変更後シナリオと協調シナリオを FreeWalk/ Q に与えて実行する。なお、協調を行わない状態には、メインシナリオの初期状態、待機状態と、応対シナリオの初期状態を指定した。図 5.7 は実行時のスクリーンショットである。

中央にいる女性が店員エージェントである。一人目の客（右の女性）が来てインタラクションが始まっているところに、二人目の客（左の男性）が来る（図 5.7 中の a）。すると二人目の客に『お待ちください』と言う（図 5.7 中の b）。一人目の客とのインタラクションが終了すると（図 5.7 中の c）、二人目の客のほうに向き直って『お待たせしました』と言い、中断していたインタラクションを再開する（図 5.7 中の d）。

以上のように、二人の相手に対応ができることが確認されたが、相手の行動によっては問題が生じることもある。

シナリオは相手の行動をある程度予測して記述するが、待機処理を行うような協調ポリシーを用いる場合、協調シナリオから実行命令が送られてくるのを待っている間に相手が店から出て行ってしまう場合もある。シナリオは実行命令を待っている間は観測を行わないので、もし実行再開後に相手のところに歩いて行って話をするようになっていれば、店の外まで追いかけていくことにもなりかねない。対応相手が複数になれば、このような状況変化が起こる可能性も高くなる。この問題は、待機処理を行った後に『待機中』という別の状態に遷移することで解決できそうにも思えるが、待機中に何かキューが発火し再び待機命令が送られてくるようなことになれば、待機中からさらに待機中へ遷移するなどネストが生じることも予想される。待機中の状況変化にどのように対応するかが今後の課題となる。

第6章 おわりに

これまでに行われてきた対話エージェントの研究は、1対1で対話を行うものがほとんどであった。しかし、より現実に即した対話を考えると複数人に対応ができることも求められると考え、本研究では複数人での対話を目標とした。

本研究では、エージェントを対話を記述したシナリオによって制御する場合

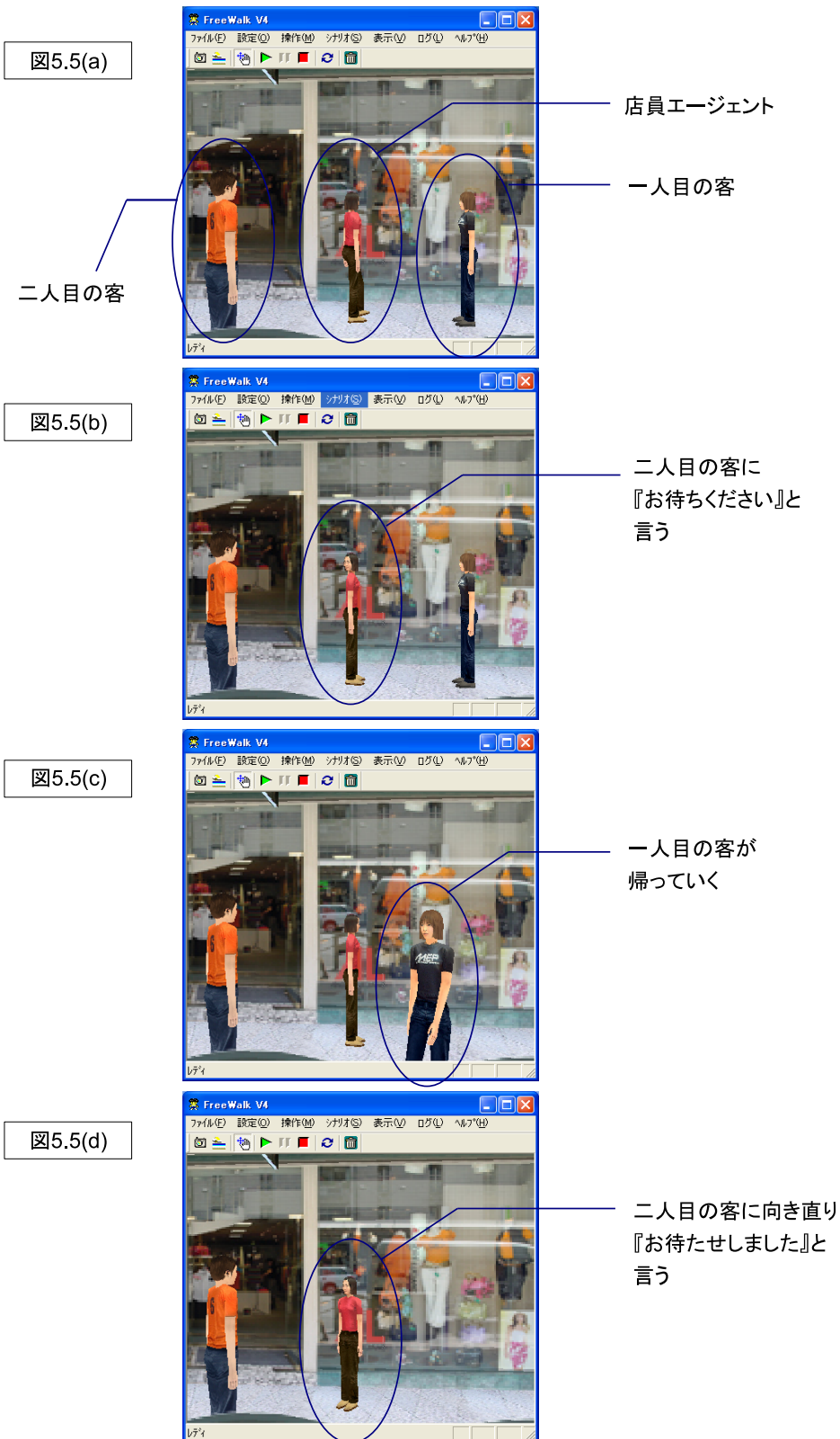


図 5.7: 実行時のスクリーンショット

を対象とする．複数の相手への対応を一つのシナリオにまとめると複雑になることから，相手別に対応を書き分けたシナリオを複数用意し，それらを並行に実行する方法を提案した．

シナリオを並行に実行する際の問題点は，アクションを複数同時に実行するときに競合が起きることと，シナリオの協調方法が指定できないことである．これらを解決するために，シナリオと通信を行うことでアクションの実行を制御する協調シナリオを導入した．協調シナリオは，対応を記述したインタラクションシナリオからのメッセージによりアクションを実行してよいか調べ，結果を返答することで実行を制御する．これにより，問題点を次のように解決した．

1) アクションを調べ，競合が起きないように制御する

複数のアクションを同時に実行したときに起きる競合は，データ競合とリソース競合という二種類の問題に大別できる．データ競合はアクションを実行する前，実行中，実行後に満たすべき条件に矛盾があること，リソース競合は，複数のアクションが同じリソースを用いることをいう．アクションの定義を与えた上でこれらの競合の条件を示した．協調シナリオはシナリオからメッセージを受け取ると競合の条件と照らし合わせ，実行してよいかを返答する．

2) シナリオの協調方法を，ポリシーとして記述可能にした

インタラクションの実行単位はアクションであるため，インタラクションを制御するには，アクションを制御すればよい．アクションをどのような場合に実行するかというメタ的な内容を，状態遷移モデルを用いて協調ポリシーとして記述する方法を示した．次に，協調ポリシーを元に協調シナリオを作成する方法を示した．協調シナリオは，協調ポリシーに従っており，かつ解決方法の1に述べたような競合を検出する機能を持ったシナリオということになる．

以上の解決方法に従い，1対1の対応用のインタラクションシナリオと協調ポリシーから，協調シナリオと通信機能を備えたインタラクションシナリオを自動生成するシナリオコーディネータを実装した．そして，実際にインタラクションシナリオと協調ポリシーを用意し，シナリオコーディネータにより得られたシナリオを FreeWalk/ Q 上でエージェントに与え，問題なく複数人とインタラクションが進められることを確認した．

本研究で提案した方法では，協調を行うシナリオと協調ポリシーを記述する

際に，お互いの内容を知らなくても記述が可能である．インタラクションシナリオは一人に対応できるよう記述すればよく，協調ポリシーについても，すでに挙げた典型的な型や，シナリオ A を B より優先するなどの内容であれば，シナリオの内容の詳細については知らなくてもよい．別々の人によって書かれたインタラクションシナリオを用いて協調を行うことができるという点が，本研究の特徴である．

謝辞

本研究を行うにあたり，熱心なご指導，ご助言を賜りました石田亨教授に厚く御礼申し上げます．また，日頃より多くのご助言をいただきました中西英之助手に感謝いたします．そして，日頃から多くのご助言とご協力をいただきました石田研究室の皆様にも心より感謝いたします．

参考文献

- [1] Cassell, J., Sullivan, J., Prevost, S. and Churchill, E.: *Embodied Conversational Agents*, MIT Press (2000).
- [2] Cassell, J., Bickmore, T., Campbell, L., Vilhjalmsson, H. and Yan, H.: Human Conversation as a System Framework: Designing Embodied Conversational Agents, In Cassell, J., Sullivan, J., Prevost, S. and Churchill, E.: *Embodied Conversational Agents*, MIT Press, pp. 29–63 (2000).
- [3] Lester, J. C., Towns, S. G., Callaway, C. B., Voerman, J. L. and FitzGerald, P. J.: Deictic and Emotive Communication in Animated Pedagogical Agents, In Cassell, J., Sullivan, J., Prevost, S. and Churchill, E.: *Embodied Conversational Agents*, MIT Press, pp. 123–154 (2000).
- [4] Rickel, J. and Johnson, W. L.: Virtual Humans for Team Training in Virtual Reality, In *Proceedings of the Ninth International Conference on AI in Education*, pp. 578–585, IOS Press (1999).
- [5] Nakanishi, H. and Ishida, T.: FreeWalk/Q: Social Interaction Platform in Virtual Space, *ACM Symposium on Virtual Reality Software and Technology (VRST2004)*, pp. 97–104 (2004).
- [6] Ishida, T.: Q: A Scenario Description Language for Interactive Agents, *IEEE*

Computer, Vol. 35, No. 11, pp. 54–59 (2002).

- [7] Georgeff, M.: Communication and interaction in multiagent planning, In *Proceedings of the 3th National Conference on Artificial Intelligence*, pp. 125-129 (1983).
- [8] Hoare, C. A. R.: Communicating sequential processes, *Communications of the ACM*, Vol 21, pp. 666-677 (1978).
- [9] Ishida, T.: Parallel Rule Firing in Production Systems, *IEEE Transactions on knowledge and data engineering*, Vol. 3, No. 1, pp. 11-17 (1991).