ソフトウェア基礎論　37－18
プログラミング言語　27－18
（1990. 12. 14）

自律的な協調を行う分権型計算モデルKemari

矢野博之*　武宮 博**　布川博士***　野口正一*

*　東北大学応用情報学研究センター
**　日立東北ソフトウェア
***東北大学電気通信研究所

あらまし　本稿では，自律・分散・協調概念に基づいた計算システムのための，計算モデルKemariを提案する．Kemariにより，従来の分散処理でみられた機能分割に基づく構成系と異なる，構成要素が自律的に協調を行うような分散処理系の解釈が容易に与えられる．また，現実世界での自律分散協調処理の例として蹴鞠を取り上げ，本計算モデルを用いて，蹴鞠のシミュレートを行うシステムの記述を行った．この記述により，自律性や協調性が自然に表現できることが示された．

Autonomous decentralized cooperative computation model : Kemari

Hiroyuki YANO*, Hiroshi TAKEMIYA**, Hiroshi NUNOKAWA***, Shoichi NOGUCHI*

*　Research Center for Applied Information Sciences, Tohoku University.
**　Hitachi Tohoku Software Co.,Ltd
***　Research Institute of Electrical Communication, Tohoku University

Abstract　In this paper, we propose a computation model : *Kemari* for computation systems based on a autonomous decentralized cooperative concept. Using *Kemari*, we are able to interpret the autonomous decentralized cooperative system easily and differently from conventional distributed systems. We think the Japanese classic ball game : KEMARI is a good environment for presenting features of our computational model. We describe the simulation system for KEMARI using *Kemari*, and show that *Kemari* is a computation model suitable for representing autonomy and cooperation naturally.

1 はじめに

近年の高度情報化によって情報処理システムの分散化が進んでいる．これは，扱うデータの増加や処理やシステム自身の多様化・複雑化，また信頼性などにより従来の集中型の処理方式では対応することができなくなったからである．しかし，現在の分散処理は

・機能の性質に着目して分割を行なった分割システム

・機能の階層性に着目して分割を行なった階層システム

になっている．

これらのシステムを構成するサブシステムは，それぞれほかのサブシステムが不稼働になったときの自分の処理の実行可能性(自律可制御性)[1]や，ほかのサブシステムとの協調性(自律可協調性)に欠ける．我々は，自律可制御性と自律可協調性を共に満たすようなサブシステムにより構成されるシステムは，自律性と協調性を持つサブシステムが分散された環境にあると考える．

一方，問題解決においても分散型による解決法がシステムの分散化にしたがって進んでいる．この方法は，「疎に結合された問題解決ノードが互いに協力して1つの問題を解く」[2]もので，特に各ノード間での協調の方法が問題になる．この分散問題解決は一分散システム上で実行されるために，この考え方を分散システムの構成にも適用することが出来る．

現在，分散型のシステムや問題解決のための計算モデル[3][4][5]が提案されているが，自律性や協調性が十分に実現されていない．特に協調という観点でみると，Lindaでは全域的な黒板を唯一つしか持たないために，協調のメカニズムの説明が明確にならない．Cellulaでは，親子関係のような階層関係によって協調処理を行っているが，協調のために新しくサブシステムを生成しなければならないため分散システムでの階層システムに対応して，既存のサブシステムが自律的な協調を行なうシステムを記述できない．これは，Cellulaが分散協調問題解決を従来の分散システム上で解決することを対象としたことにも原因があると考えられる．また，Kamui88ではオブジェクト指向の独立性に基づいてサブシステムを独立に構成し，協調をグループへの出入りにより自然に表現しているが，送られてきた情報を処理する通信形態を取っているために，サブシステムの自律的な処理を十分に実現することはできない．

我々が提案する計算モデルが対象とする分散処理システムは，自律可制御性と自律可協調性を共に満たすようなサブシステムの自律的な分散協調処理を実行できるものである．このように，各サブシステムがもつ機能が全体の機能の分割でもなく，他との階層関係によって形成されるものでもないような分散処理形態を，分権型と呼ぶことにする．分権型システムとは，従来のような全体としてある機能を持つものを構成するために，どの様に各サブシステムに機能を割り振るのかという，「全体→サブシステム」の発想ではなく，逆にそれぞれ十分な機能を持った各サブシステムが集まったときに，全体としてシステムがある機能を持つという「サブシステム→全体」の発想に基づいて構成されたシステムで，各サブシステムは自律性と協調性を有する．

また，この計算モデルにより，分散処理を各サブシステムが自律的に協調しながらおこなうシステムが自然に記述できる．本計算システムはネットワーク・アーキテクチャ等のハードウェアから完全に独立したものとして与えられる．

本稿ではこの計算モデルの基本構成を示し，本モデルの適用例として蹴鞠のシミュレートをおこなうシステムの記述を行なう．また，我々はこの計算モデルをKemariと呼ぶ．

以下，2章では分散問題解決と分散システムを通して，我々の考える自律・分権・協調の概念とそれぞれの関係について述べ，3章で，その概念をプログラミング言語の世界でモデル化するために必要な機能について述べる．4章では，計算モデルKemariの基本構成について述べる．5章は，Kemariの適用例として蹴鞠のシミュレーションについて説明する．6章は，まとめと今後の課題である．

2 自律·分権·協調概念

自律分散協調の概念は，経済学，生物学など最近多くの分野で用いられているが，従来から研究として取り組んでいる分野は制御工学の分野である．しかしその制御工学でさえ，「自律分散システムは概念だけが先行してその理論的展開はまだほとんど行なわれておらず，どの様な構成原理によれば自律分散システムが実現できるかがわかっていない．」[6]という状態で，この言葉を用いる場合には，適用しようとする分野での概念をしっかりと定める必要がある．

この章では，まず自律・分権・協調概念とそれぞれの関係を定義する．

2.1 分権

従来から行なわれている分散問題解決は，最初の大きな問題をより小さなサブ問題に分割し，それらのサブ問題で得られた結果をうまく組み合わせることにより，最初の問題の解を得るものである．この分割の手順は，まず統括の機能を有するもの(統括者)を作成し，その配下にサブ問題を配置していく．統括者はサブ問題の制御とサブ問題で得られた結果の統合をおこなう．この分散問題解決の方法を従来の分散型計算機システムの形態に投影して考えた場合，このシステムはマスターとしての統治者が一切の統治権をもち，各サブ問題はスレーブとして機能し，この関係が永遠に持続する．またサブ問題への分割の度

合は，システムに内在する統治者の数に影響しその数は動的に変化する．また，複数回の分割は，システムの階層性に対応する．

以上のような従来型の分散システムに対して分権型の分散システムとは，上で述べたような永続的な統治者が一切いない，すなわちすべてのサブ問題が対等なシステムのことを言う．このシステムでは，ある一時点ではサブ問題の間にマスター-スレーブ関係が存在してもよいが，その関係は永続的なものではない．また，このシステムでは各サブ問題を統括するものが存在しないために，他のサブ問題の結果を参照したり，自分の機能を他のサブ問題に利用させたりするような機能，すなわち協調のための機能が必要になる．また，その協調の機能を統括者がいないために自分自身で行なわなくてはならない．したがって，自律のための機能も必要になる．

以上のことから，分権型の分散システムが全体としてうまく動作するためには，以下のことが成立しなくてはならない．

分権ならば，統治者がいないので，周りの様子をみて自らの行動様式を決めなくてはならない．(1-1)

分権ならば，統治者がいないので，自分自身で単独に自分の行動を決めなくてはならない．(1-2)

2.2 自律

自律とは，各サブ問題がそのサブ問題自身で自分の行動を定められることをいう．すなわち，それ自身が他から強制的に干渉される事なく動くことが可能になっている．したがって，サブ問題を統治する永続的な統治者が存在しない．これに対して従来の分散システムは，統治者がサブ問題を制御，結果の収集をするといった点で，自律的ではない．また，自律的なサブシステムが他からの干渉を受けることを自分で定めてもよい．したがって，次のことが成り立つ．

自律的なサブ問題は，それ自身，分権型の分散システムにおけるサブ問題である．(2-1)

複数の自律的なサブ問題が集まって全体の問題を解こうとしたとき，それらがなんらかの方法で互いに情報を交換する必要がある．また，自律的なサブ問題は情報交換の結果を利用して，自らの行動を自ら定めることが出来る．よって，以下のことが成り立つ．

自律的なサブ問題が，統括者なしにシステムに与えられた問題を解くためには自分自身で他と情報交換をしなければならない．そして，その情報交換の結果を利用して自らの行動を定めなければならない．(2-2)

2.3 協調

協調とは，周囲の様子をみて自らの行動様式を変更することを言う．これを，分権的な分散システムで考えれば以下のようになる．すなわち，自律的なサブ問題はその行動を，周囲のサブ問題との情報交換により変更する．すなわち，式2-2は以下のように言い替えることが出来る．

自律的なサブ問題が統括者なしにシステムに与えられた問題を解くためには他のサブ問題と協調しなくてはならない．(3-1)

また，協調するためには，自らの情報収集機能，自分自身で行動を定められる機能を有していなければならないから，以下のことが言える．

自律的でなければ協調できない．協調するためには自律的でなければならない．(3-2)

以上の事から，我々は以下の系を得ることが出来る．この系は，分権自律協調が三位一体であることを示す．

系

分権的な分散システムはシステムに与えられた問題を解くためには，各サブ問題は自律的に働く．そして，それらサブ問題間で協調しなくてはならない．

すなわち自律分権協調システムとは自律，分権，協調が集まったものではなく，自律分権協調という一語で語られる，特定のシステムを指す．

3 計算モデルでの自律･分権･協調

この章では2章で定めた自律・分権・協調の概念をプログラミング言語の世界でモデル化するときに必要な機能について考える．このとき2章での1つのサブ問題，サブシステムが計算モデルでの計算主体，実行主体に対応する．

3.1 協調性

協調とは，「周囲の様子をみて自らの行動様式を変更すること」であったが，これは周囲の実行主体とデータのやり取りをすることにより周囲の様子を知り，さらに実行主体自身に内在する協調のメカニズムにより自らの行動様式を変更すると考え

る．このときの実行主体の動作は協調のためのプロトコルにしたがう．

この協調性を計算モデルの上で実現するために，

・他の実行主体との情報交換

・協調するためのプロトコル

の2つを必要な機能として考える．

まず，他の実行主体との情報交換では，協調している各実行主体はそれ自身1つの自律分権協調システムであると考えられるので，それらを1つにまとめるためにグルーピングの概念を導入する．すなわち，協調している実行主体は1つのグループの中にいて，グループがまた1つの実行主体としての機能を持つと考える．また，協調しようとする実行主体はグループの中に入り，協調を終えた実行主体はグループから出る(図1)．このグループへの出入りは，各実行主体が自律性を持っていることから，実行主体自身が決める．また，このときグループは，各実行主体が互いに相手を指定しなくても，情報交換を自由に出来るための機能を持つ．

協調のためのプロトコルは，上で述べたグループの出入りを定めたものである．また，このプロトコルはグループの構造の動的な変更が可能であるための機能を有する．

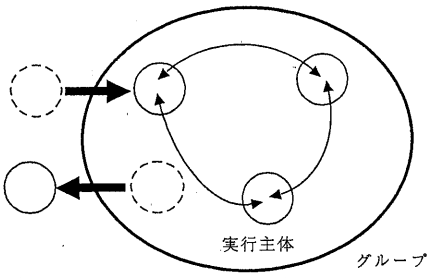


図1　協調のためのグループ

3.2　自律性

実行主体が自律性を持つことは，単に自律性を持つための新しい機能を用いる事では十分に実現されない．例えば，「自律性のために新しい通信形態を追加した」ということは，データの受け渡しという実行主体の持つ1つの機能が，自律性を持っているというだけで，実行主体自身が自律性を持っているとは言えない．したがって，実行主体が自律性を持つということは，実行主体の持つ機能が自律性を持っているということと同値であると考える．

2章で自律性とは，「各サブ問題がそのサブ問題自身で単独に自分の行動を定められること」と定めたが，実行主体が持つ機能でこの自律性を実現できるものとして，ここでは

・データの受け渡し

・仕事の依頼関係

の2つの機能を考える．

データの受け渡しでは，実行主体の独自の判断により通信形態を変更できるものとする．すなわち，1対1通信やブロードキャストなどのような1種類の通信プリミティブを持つのではなく，複数の通信形態を実行主体の判断により選択できるものとする．もちろん，他の実行主体からのデータからの指示にしたがって通信形態を変更してもかまわない．

仕事の依頼関係については，従来の分散問題解決で用いられてきたような，「マスターから依頼された処理を必ず実行しなければならない」，というものではなく，自分で処理の取捨選択が出来るということが必要であると考える．この機能を実現するためには，2種類の機能を持っていなければならない．1つは，自分に送られてきた処理を取捨選択できる機能で，もう1つは自分から処理を選択して取りに行く機能である．ここで，先の機能は一時的に実行主体がスレーブ状態になって協調させられているときの自律性，後の機能は協調しようとしているときの自律性のために必要な機能である．

3.3　分権性

これまでに述べてきた協調性と自律性のための機能を持った実行主体は，協調のために自律的にグループを構成し，またそのグループ自身が実行主体として行動することが出来る．さらに，グループへの出入りも，実行主体自身の単独の判断で決めることが出来た．

また，分権型の分散システムとは，「各サブシステムを永続的に統治する統括者が一切存在しない，すべてのサブシステムが対等なシステム」であった．したがって，自律性と協調性を併せもった実行主体は，それ自身，分権性のための機能を持っているといえる．

先に，グループ自身も1つの実行主体としての機能を持つことを述べた．また，分権型の分散システムでは，各サブシステムが対等なので，グループと実行主体がそれぞれ複数個集まって構成されたものも，1つのグループであると考える．

前章の終わりで述べたように，自律・分権・協調概念は三位一体でそれぞれを分離して語ることはできない．さらに，式(1-1)，(1-2)，(3-2)で示されるように，各概念をモデル化するために必要な機能の間にも包含関係が存在する．したがって，本章

で述べた機能をすべて持つような実行主体の集まりが自律分権協調システムの計算モデルになると考える．

4 Kemariの基本構成

前章では，自律分権協調概念をモデル化するときに，必要な機能について述べた．本章では，その機能をもった実行主体によって構成される計算モデルについて，その基本構成と情報交換法を述べる．我々は，現実世界での自律分権協調処理の例として日本古来の遊びである「蹴鞠」を取り上げ，これを我々の提案する計算モデルでシミュレートすることを目的とした．この計算モデルを，Kemariと呼ぶ．

4.1 基本構成

Kemariは実行主体として機能するプロセス，交換される情報を蓄えるプールから構成される．このプロセスとプールを合わせてファンクショナルセル(F_C)と呼ぶ．

プロセスは識別子としてのタグと，自分が知っているF_Cを記憶している内部状態と，自分の処理できる機能を情報交換を単位として記述した機能記述部と，協調のためのプロトコルを記述した協調記述部から構成される(図2)．

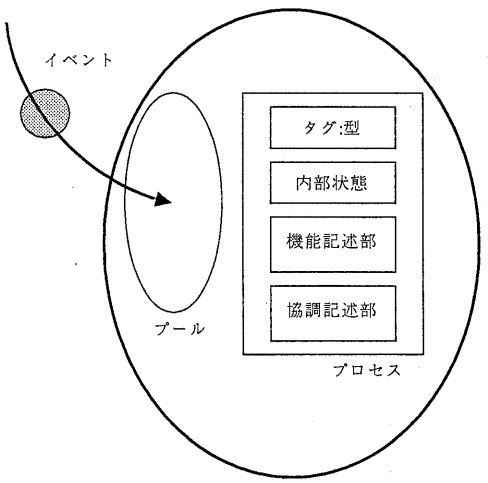


図2 Kemariの基本構造

タグは識別子としての名前と型を持つ．同じ型を持つF_Cは，協調できる機能を持つことを表す．これらのF_Cは，グループを作り協調処理を行うことが出来る．また，型により持っている機能を表すことになるので，特定のF_Cの名前を知らなくても，型の指定により処理の依頼が出来る．

内部状態には，自分が知っているF_Cを名前または型を用いて記述する．ここに名前を記述されたF_Cとは，直接通信が可能になる．この内部状態は，情報交換により動的に変化する．以下にplayer1というタグの付いたF_Cについての例を示す．

```
タグ
F_C     player1 : player   ; F_Cplayer1はplayer型
  内部状態
        known-about : ball, player
                          ; ball型とplayer型のF_Cを知っている
```

F_Cがやり取りする情報をイベントと呼ぶ．このイベントの処理とイベントのやり取りを記述したのが機能記述部である．ここでは，1回のイベントのやり取りと処理を基本単位として記述される．

協調記述部は，グループを作ってその中で協調を行うときの処理について記述する．F_Cは，グループを構成するために演算を用いることができる．この演算では，名前または型によって協調するF_Cを指定する．また，協調するF_Cは，自分が協調をやめてグループから出るときに，自分の機能を他のF_Cに渡し，自分の機能を肩代わりしてもらう機能も有する．このときにも，グループ構成に関する演算が用いられる．

F_Cは，F_Cを生成できる親F_Cにイベントを送るか，または自分自身のコピーを作ることによって動的に生成することができる．

4.2 Kemariでの交信形態

Kemariは，計算モデルなので物理的な情報交換は考えず，論理的な情報交換を考える．この2つの情報交換を区別するために物理的な情報交換を通信，論理的な情報交換を交信と呼ぶ．以下では，Kemariでの交信とイベントの構造について述べる．

Kemariの交信で扱うデータであるイベントは，名前とデータの有限列で構成されるリスト構造をしている．

イベント：(名前，data1，data2，・・・，datan)

イベントはF_C内のプロセスで生成されイベントを蓄えるプールに送られ，プールを仲介して交信がなされる．このときのプールは，自分のプール，他のF_Cのプール，グループのプールである．自分のプールや，自分が属しているグループのプールへは自由に読み書きできるが，他のF_Cのプールには，F_Cが常にすべてのF_Cを知っているわけではないので，他のF_Cの存在を知らなければ読み書きが出来ない．また，イベントの名前

により，プールの中のイベントを指定する．Kemariでは，この書き出す先のプールの違いにより様々な交信形態が実現される．以下にそれを示す(図3).

・自分のプールに書き出す場合

これは，自分自身または自分の存在を知っている他のF_Cから，自律的に直接にイベントを読み出されることを表す．

・他のF_Cのプールに書き出す場合

これは，1対1の直接交信を表す．このときは，あきらかに，相手のタグの名前を知っていなければならない．

・グループのプールに書き出す場合

これは協調のために用いられる．F_Cは協調するときにグループを作るが，このとき，グループとしてのプール(グループプール)が1つ作られる．このグループプールの中のイベントは，グループ内のF_Cからは自由に読み書きされるので，グループ内でのブロードキャストに対応する．また，このイベントは，グループの存在を知っていれば，グループの外のF_Cからも読み書きされる．

プールに置かれたイベントは，他の(自分自身も含む)F_Cから自律的に読み出される．F_Cがイベントを読み出すとき，プールからイベントを消去しないで，イベントの後に自分のタグを付ける．例えばイベント1が"F_C1:h"というタグを持つF_Cに読み出された場合，イベント1は(イベント1, F_C1:h)という構造に変わる．また，読み込んだイベントを処理しない場合は，自分のタグを付けない事もできる．

イベントを発生したF_Cは，自分に書き出す場合は，そのイベントがどのF_Cから参照されるべきか知っているので，プール内のイベントの構造を見て，読み出されるべきすべてのF_Cから参照されたイベントがあればそれを消滅する．また，直接相手F_Cのプールに書き出した場合は，相手F_Cが読み出した時に，相手がイベントを消す．また，グループプールに書き出した場合は，不特定多数のF_Cを対象としているので，イベントを読み出そうとするF_Cを特定できない場合がある．したがって，グループプール内のイベントはそのイベントを書き出したF_Cによって，消滅することにする．

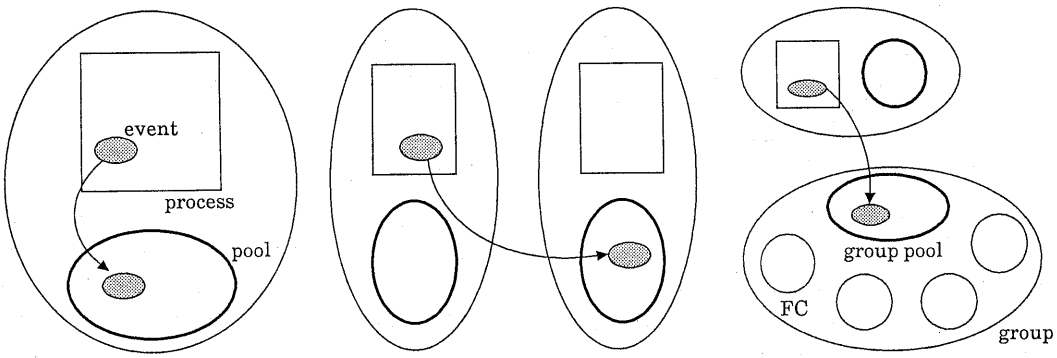
4.3 交信の記述と処理の実行

KemariでのF_Cの機能の記述は交信を基本単位として静的に記述される．すなわち，1回の交信に対する処理を1つの式で記述する．ここでは，処理の記述をλ-式の形を用いて行なう．ここでλ-式を用いたのは型の明示的な記述，機能の静的な記述，継続により逐次的な処理の表現が可能なためである．以下に機能記述式の構造を示す．

Λ<event> : tag. (process, reception-tag)

これは，tagで指定されたF_Cから<event>で指定された名前を持つイベントを読み込み，processで指定された処理を行ない，得られたイベントを reception-tagで指定されたF_Cのプールへ送ることを表す(図5). tag, reception-tag は，それぞれタグの名前または型で指定する．また，processが()の場合はデータを読み取って何も処理をしないことを表す．

また，協調処理では，協調のグループから出る場合のように，他からイベントをもらうことなしに，自律的にイベントを書き出すことがある．このために，協調記述部では，



自分のプールへの書き出し　　他のF_Cのプールへの書き出し　　グループプールへの書き出し

図3　Kemariの交信形態

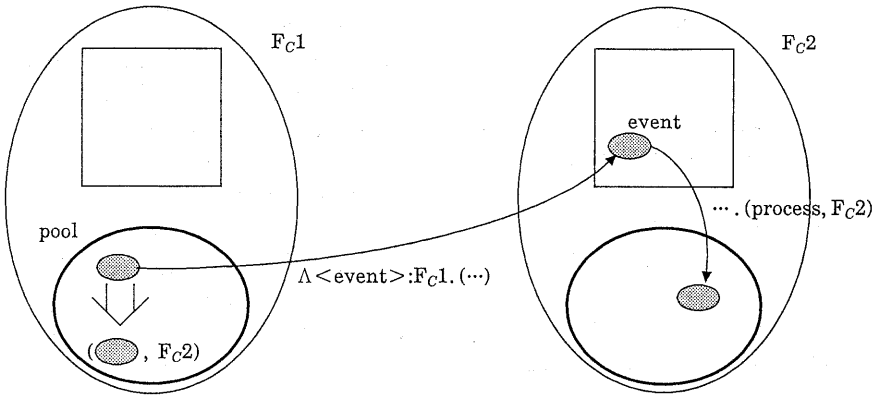


図4　交信の記述　$\Lambda<\text{event}>:F_C1.(\text{process}, F_C2)$

((event), reception-tag)

で，自律的なイベント書き出しを表すことにする．

5　蹴鞠のシミュレーション

5.1　蹴鞠とは

蹴鞠は，複数の人間が円陣を組み1個の鞠をできるだけ長く蹴りあう遊びである．鞠が自分の所に来たときに自分が蹴っていいかどうかを自分の周りの人間の状況を把握して独自に決定しなければならない(図6)．即ち，蹴鞠を行なう一人の人間を一つのサブシステムと考えると，このサブシステムは自律的に周りの状況を把握して他のサブシステムと協調しながら処理を進めなければならない．従って，人間をプロセスとみなすことによりKemariによるシミュレートが可能になる．

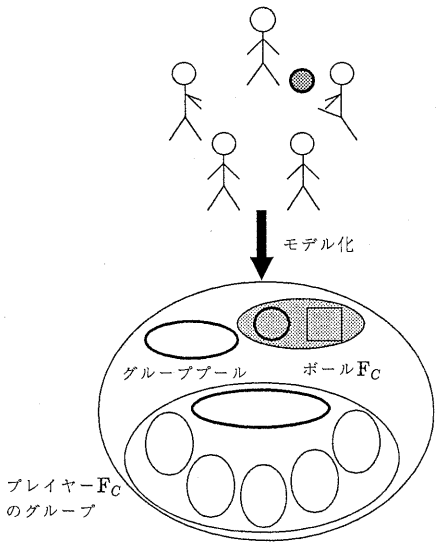


図5　蹴鞠のモデル化

5.2　Kemariによる蹴鞠のシミュレーション

本節では，5人のプレイヤーにより行われる蹴鞠を，本計算モデルで記述した例を示す．

蹴鞠の実行は，協調記述部で記述されているように，1つのプレイヤーF_Cが，プレイヤーF_CとボールF_Cを併せたグループのグループプール(all-pool)へ蹴鞠開始のイベントを書き込むこと(式 C)により開始される．そのイベントを見て，蹴鞠に参加したいF_Cは，イベントを読み込んで，自分のタグを付加する．蹴鞠の参加定員に達すると，イベントを書き込んだF_Cがall-poolのイベントを消去し，参加F_Cをグルーピング(式 G)する．さらに，参加メンバのリストを参加者にブロードキャストし(式O，O')，自分の内部状態を設定する(式 M)．その後，蹴鞠が開始される．

また，各F_Cは自分の守備範囲に関するデータをもっていて，協調をやめてグループから出るときに，そのデータをイベントにして，プレイヤーグループのグループプール(p-pool)に書き出すことにより，他のF_Cにブロードキャストする(式 E)．また，そのイベントを受け取ったF_Cは自分の内部状態とグループ構造を変更する(式 R)．このときに，グループに対する演算が用いられる．

以下に，各F_Cの構造を示す．

F_C　ball : b　;タグball(ボールF_C)の型はb
known-about : ( )
function
B＝Λ<蹴りだしdata> : p-pool. ((落下地点の計算), p-pool)
cooperation
S＝Λ<参加メンバ召集> : all-pool. ( )
M＝Λ<参加メンバリスト> : ball.
(内部状態のknown-about作成)
;参加メンバリストによる内部状態の作成

```
F_C  p1 : p    ; タグp1 (プレイヤー F_C) の型はp
     known-about : ( )
     function
      P1 = Λ <落下地点data> : p-pool.
                ((落下点までの移動時間の計算), p-pool)
      P2 = Λ <移動時間data> : p-pool.
                ((自分が蹴ることができるかどうかの判断,
                  蹴りだしdata作成), p-pool)
     cooperation
      C = ((参加メンバ召集), all-pool)
            ; 蹴鞠参加メンバの召集イベントをall-poolに送る
      G = Λ <参加メンバ召集> : all-pool. (グループ作成)
            ; 蹴鞠開始時のグループ作成
      O = ((参加メンバリスト), p-pool)
      O' = ((参加メンバリスト), ball)
            ; 参加メンバのブロードキャスト
      M = Λ <参加メンバリスト> : p-pool.
                (内部状態のknown-about作成)
            ; 参加メンバリストによる内部状態の作成
      E = ((自分の守備範囲のデータ), p-pool)
            ; 自分が協調をやめるときの他のF_Cへの守備範囲
              の分配
      R = Λ <リタイヤリスト> : p-pool.
            (自分の内部状態とグループ構造の変更処理)
            ; 協調をやめたF_Cのイベントを受けたときの内部
              状態と グループ構造の変更

他のプレイヤー F_C はp1と同じ
```

図6　Kemariによる蹴鞠シミュレーションの記述

6 まとめ

本稿では，自律分権協調の概念を定め，その概念をプログラミング言語の世界でモデル化した計算モデルKemariを提案した．また蹴鞠のシミュレーションにより実際の問題への適用が可能であり，Kemariの持つ機能により，自律性や協調性が自然に記述できることが示された．

今後の課題は更にこの計算モデルの記述言語を考え，実験的な方法により自律・分権・協調概念を考察することである．

参考文献

[1] 井原, 大島 : 自律分散システムとその応用, 電気学会雑誌, 昭59 - 3 (1984), pp. 169 - 176.

[2] Lesser, V., Corkill, D. : Distributed Problem Solving, Encyclopedia of Artificial Intelligence, pp. 245 - 251, John Wiley & Sons (1988).

[3] Carriero, N., Gelernter, D : Linda and Friends, IEEE COMPUTER, August 1986, pp. 26 - 34.

[4] 渡辺, 原田, 三谷, 宮本 : 場とイベントによる並列計算モデル-Kamui88, コンピュータソフトウェア, Vol. 6, No. 1 (1989), pp. 41 - 55.

[5] 吉田, 楢崎 : 場と一体化したプロセスの概念に基づく並列協調処理モデル Cellula, 情報処理学会論文誌, Vol. 31, No. 7 (1990), pp. 1071 - 1079.

[6] 伊藤 : 自律分散システムはいかにして構成されるか, 計測と制御, Vol. 29, No. 10(1990), pp. 1-5.