ソフトウェア工学 123－3
（1999．5．14）

共有オブジェクト空間を利用したプログラミングスタイル

飯島 正
慶應義塾大学理工学部

本論文は，共有オブジェクト空間を利用したプログラミングスタイルを分類し，さらにシナリオ主導スタイルを提案するものである．共有オブジェクト空間は，タプル空間や黒板アーキテクチャといった共有データ空間の拡張である．シナリオ主導プログラミングスタイルは，データ主導とタスク主導の両方を統合するものである．

Programming Styles based on Shared Object Spaces

Tadashi Iijima,
Faculty of Science and Technology,
Keio University

In this paper, programming styles based on shared object spaces are classified, and scenario centered programming style is proposed. The shared object space is an extension of a shared data space, such as Tuple Space or Blackboard Architecture. The scenario centered programming style integrates both of data centered style and task centered one.

1. はじめに

タプル空間モデルに基づくLindaシステム[1]-[5]や黒板アーキテクチャ[6]-[8]など，共有データ空間に基づくソフトウェアアーキテクチャは従来から，幅広く使われてきた．その中には，そうした共有データ空間上で読み書きするデータをオブジェクトに拡張する試みがある．本論文では，そのように拡張されたものを共有オブジェクト空間（Shared Object Space; 略して SOS）と呼び，その共有オブジェクト空間でのプログラミングスタイルについて論じる．そうした共有オブジェクト空間を提供する仕掛けとしては，最近では，Java 言語のための Jini 技術上のサービスである JavaSpaces[9]がある．

共有オブジェクト空間という仕掛けが与えられたとき，それを利用するプログラミングのスタイルには，いろいろなものが考えられる．例えば，共有空間にデータを置き，それがきっかけとなってデータ主導的に計算が進むようなものであるとか，ゴールやリクエストというようなむしろタスクをそこにおいてタスク主導で計算を進めるものなどがある．そこでポイントとなるのは協調ないし連携動作[10]である．そうしたプログラミングスタイルを分類するにあたっていろいろな分類観点が考えられる．

本論文では，その空間に何を置くのか，どのようにプロセスを制御するか，といった点から，プログラミングスタイルの分類を試みる．

さらに，データ主導とタスク主導を統合するシナリオ主導スタイルを提案する．このスタイルではタスクやデータに関するメタデータを，共有空間に置くことで実行時により深い協調動作や連携動作が導入できることを目的としている．

2. 共有オブジェクト空間

2.1. 共有オブジェクト空間とは

共有オブジェクト空間とは，従来より幅広く使われてきた共有データ空間に基づくソフトウェアアーキテクチャ(例えば，タプル空間モデルに基づくLindaシステムや黒板アーキテクチャなど)の「データ」を「オブジェクト」に拡張したものである．そうしたアプローチとして具体的には，最近では，Java 言語のためのJini 技術上のサービスであるJavaSpacesがある．

JavaSpaces サービスでは，タプルからオブジェクトに拡張されているので，以下に示すような幾つかの相違がある．

(1) データタプルとプロセスタプルといった種別はない．eval 操作もない．
(2) 属性の型や，継承を考慮したパターンマッチングが行われる
(3) メソッドを持つ．

他に，JavaSpaces サービスでは，分散ネットワーク上に複数の JavaSpace サーバを配置して，分散トランザクションで操作し合う，分散共有オブジェクト空間に拡張されている．こうした拡張は，共有オブジェクト空間どうしを構造化して連携させるメタ空間の構成に展開可能だと期待できる．次節では，そうした連邦化(federalization)も含めた共有オブジェクト空間のモデル化を試みる．

2.2. 共有オブジェクト空間のモデル化

以下では，オブジェクト指向概念に添った形で共有オブジェクト空間のモデル化を試みる．このモデル化は，メソッドのパラメータ渡しを値渡しを前提とするなど，JavaSpaces のAPI からの影響が大きいが，トランザクションやリースに関して簡略化してある．

2.2.1. 単一の共有オブジェクト空間

まず，単独の共有オブジェクト空間 SOS を以下のようにモデル化する．

$SOS = \langle OS, AS \rangle$.

ここで，OS はオブジェクト空間を意味するオブジェクト，AS はエージェントの集合である．

$OS = \{O \mid O \text{ はオブジェクト}\}$

$AS = \{A \mid A \text{ はエージェント}\}$

エージェントAは，OSに対してOを読み書きするといった操作を行うことができるプロセス(中のオブジェクト)である．A は，OS の外部のプロセスであるのか一般的である．但し，ここで，

$$O \supseteq A$$

とする(但し，集合の包含ではなく，⊇の意味は概念順序とする)．すなわち，エージェントはオブジェクトとしての操作を受け付けるものとし，共有空間(OS)に書き込むことができる．

基本的な OS の操作メソッドには以下のものがある．

・書き込み操作
　write : O→⊥
・ブロッキング読み出し操作
　read : O→O
・非ブロッキング読み出し操作
　readIfExists : O→O
・ブロッキング取り出し操作
　take : O→O
・非ブロッキング取り出し操作
　takeIfExists : O→O

これらの意味付けは，JavaSpaces に準じる．

書き込み操作は，オブジェクトの参照(JavaSpaces ではコピー)を共有空間に書き

込む操作である. ここで, 共有空間に書き込まれてからオブジェクトがその属性値を変更することも許される.

読み出しと取り出しの操作の引数はテンプレートであり, パターンマッチングでマッチする属性値を持つオブジェクト(のリファレンス)が返却される. ブロッキング操作はマッチするオブジェクトが共有空間中に存在しなければ, それが書き込まれるまで実行を中断し, マッチするオブジェクトが書き込まれると再開する. あるテンプレートにマッチするオブジェクトが複数書き込まれている場合, 取り出し操作の繰り返しで順次, 取り出されるが, その順序は保証されない.

一方, 非ブロッキング命令は, マッチするオブジェクトが共有空間中に存在しなければ, 特別なオブジェクト(リファレンス) $\perp$ を直ちに返す. これは, テンプレートにマッチするオブジェクトの有無をチェックする

・存在検査操作

checkExists : O→Boolean

とブロッキング操作との組合せを一つの原子操作とできるならば, その組合せとして表現できる(但し, チェック時にマッチしたオブジェクトと実際に読み出されるオブジェクトは一致するとは限らない).

他に, 与えたテンプレート O_1 にマッチするオブジェクトの書き込みを検知してエージェント A にイベント Ev の発生を通知することもできる. その通知依頼として以下の操作がある.

・書き込み通知依頼操作

notify : $O_1 \times A \rightarrow Ev$

その依頼の取り消し操作などもあるがここでは省く. また, A で実装されるイベントハンドラメソッドを

event : Ev→ $\perp$

とし, パラメータの Ev で発生したイベントの種類を識別できるものとする.

2.2.2. 連邦化

共有空間を分散ネットワーク上に複数個配置し, その間に仲介エージェント(ブローカ)を設けることで連邦化することができる. その仲介の仕方によって, いろいろな構造化が可能になる. たとえば, 特定の条件を満たすオブジェクトが書き込まれたら, 転送したり, コピーするようなブローカを作ることができる.

3. 何を共有空間に置くか?

共有オブジェクト空間に基づくいろいろなプログラミングスタイルを特徴づける分類観点として, まず, 共有空間に置かれるものは何なのか, そして, それがどのように計算されていくのか, について考えてみる.

共有データ空間の基本的発想からすれば, そこに置くものとして典型的なものは, データ・オブジェクトである. その場合, データを共有空間に置き「データ主導」で計算を進めることになる. しかし, その「データ」としてクライアントからの「リクエスト」(を表すデータ)を置き, サーバがそれを取得して代わりに「結果」を置くような構成を考えれば, その振る舞いは「タスク主導」であり, 共有オブジェクト空間自体が一種のメッセージキューとして機能することになる. そうしたタスク主導とデータ主導の中間にイベント主導が位置付けられる. イベントはタスクほど処理と密接に結合しておらず, またデータのように直接の処理対象でもない.

3.1. データ主導プログラミングスタイル

これは, Hearsay-II などの黒板モデルでも採用されていた典型的な共有データ空間のプログラミングスタイルである. 共有空間に置かれるのはデータである.

その共有空間をとりまくエージェントの性質によっていろいろなスタイルができる. その一つは一方のエージェントが生産者, もう一方

が消費者というように，役割の分担がなされているタイプである．生産者と消費者の間に，一般には，複数の中間加工業者が存在し得る．こうしたデータ変換の流れにとって縦に並ぶ役割分担とは直交して，横方向の役割分担もある．こうした役割分担は，エージェントが処理できるデータの種類が異なるという専門性に他ならない．　そのデータを処理できるエージェントが各自，自分の専門範囲のデータを共有空間から取り出して処理しては結果を戻すということを繰り返すことで，計算が進んでいく．しかし，データがあらかじめ与えられた手順に沿って，同期的に変換されていくのであれば，共有空間を使うメリットは小さい．

Hearsay-II では，雑音や発声の揺らぎを含むような音声認識を対象にしているため，はじめに与えられている情報が不完全である．そのため，音素，単語，句，文というような階層構造にそって必ずしも処理することができず，他のすでに認識できている「島」からの階層を越えた相互作用が，その不完全性を補完するのに役立つ．こうした予め制御の順序を規定できないような場合の，機会主義的な振る舞いが黒板システムのような共有データ空間の典型である．そうした，機会主義的なアプローチは，たとえば，プランニングにも適用されている．

Linda[1][2]では，共有空間上に置くデータの配置に着目した分散データ構造体を中心に，共有空間上にデータを内包するプロセスを配置する形にプログラムを構成して通信が明示化されるメッセージパッシングと，データの中にプロセスを内包させて通信を暗黙化する活性データ構造体という二つのスタイルに展開して具体的にプログラミングできることが示されている．

3.2. タスク主導プログラミングスタイル

タスク主導のプログラミングのもっとも典型的な形は，クライアントがサーバへのサービスリクエストを共有空間においてサーバがそれを取得してサービスを遂行してくれるのを待つというパターンである．共有空間を介すことにより，サーバとクライアントの結合を弱めることで具体的には2つのメリットが得られる．一つは，同等のサーバが複数あるとき，空いているサーバがリクエストを取得していくので，負荷分散がなされるという点である．このときに，とくにタスク分配をおこなうマネージャを設ける必要がない．マネージャにエージェント(ワーカプロセス)の情報を集中させる必要がないので，エージェントの追加交換が容易である．．もう一つは，共有空間に置かれたリクエストにとって最適の専門性をもったサーバが自らリクエストを見つけてサービスしてくれるという点である．これも集中管理するマネージャを置かずにすむというメリットにつながり，システム全体のスケーラビリティを向上させる．

もっとも，このスタイルにおいて，共有空間に置かれるのは「サービスリクエスト」であるが，これをデータとして捉えるならば，データ主導スタイルに含まれることになる．しかし，処理対象のデータというよりも，作業指示として解釈できる場合もあり，その場合には，このタスク主導スタイルに含めることとする．その場合，共有リクエストキューによく似たものとなるが，基本的な共有空間自体にはキュー(待ち行列)のように優先順位が組み込まれていないことになる(必要があればリクエストに埋め込むか，そうしたメカニズムを共有空間の操作に埋め込めばよい)．

また，この仕掛けは，受け手を指定しないメッセージになっているため，マルチキャストのための一種のドメインを規定するものとしても捉えることができる．したがって，この共有空間上上で，いわゆる契約ネットプロトコル[13]のタスク告示や，入札なども行うことが可能である．他にも，いろいろなタイプのオークションプロトコルがこの上で実行可能である．

契約ネットプロトコルの場合に重要なことは，

単純なリクエスト／リプライの対ではなく，サブタスク，サブサブタスクへとタスク分解を繰り返していく点である．この点で，契約ネットプロトコルは単なるコミュニケーションパターンではなく問題解決手法と呼ばれている．こうした契約ネットや，プランニングのように問題を部分問題に分割し，それを解決する手順に割り当てていくエージェント(群)(言い換えると，手順生成器)と，その共有空間に置かれた手順を実行するエージェント(群)の両方を共有空間の周りに配置することで，より深くタスク主導の協調動作を実現することができる．

3.3. イベント主導プログラミングスタイル

イベントは，3.2 節のタスク主導スタイルというほどタスクに密接に結びついておらず，3.1 節のデータ主導スタイルというほど処理対象のデータとは言い切れないというような中間的な位置付けとなる．共有空間をつかったモデルでは，そこに書かれたオブジェクトをどのエージェントが取得するかはエージェントの側で指定するパターンマッチによる．この点で，共有空間を介した通信はイベント通知とよく似た性質を持っている．もっとも 2 章で規定したモデルにおいても，効率上から，エージェント自体が共有空間を現実に(たとえばポーリングループで)監視するようなコーディングは避けるのが一般的であり，共有空間への書きこみをイベントとして検知して更に，それに関心をもっているエージェントへのイベント通知を行う仕掛けを与えている．その意味では共有オブジェクト空間そのものがイベント通知メカニズムに対応するようにはなっているわけではない．

4. シナリオ主導スタイルの提案

第3章でのプログラミングスタイルの分類と分析を踏まえて，新たにシナリオ主導プログラミングスタイルを提案する．

このスタイルは，[10]からインスパイヤされて，本論文で新たにカテゴリ化するものである．データ主導の特徴なども含めつつタスク主導の一つの発展した形と捉えている．

[10]は並行計算モデル GIM(Generic Interaction Model)[11][12]という共有空間上でのオブジェクトの連携動作と相互作用の研究に基づいている．GIM では，タプルの間の相互作用を自由に設計できるようにタプル空間を拡張した GIS(Generic Interaction Space)を持っている．GIM では，これを利用してプロセス間の相互作用を宣言的に記述しており，個々のプロセスの記述と組合せて並行計算を記述できる．共有空間 GIS に書き出されるタプルによって，共有空間上で相互作用がコントロールされる．同モデルでは，あらたなプロセスの追加が既存のプロセス群の独立性を損なわないという開放系にとってきわめて望ましい性質を持っている．このモデルでは，相互作用というメタなレベルの記述が与えれている点が特徴的である．

本論文であたえるシナリオ主導スタイルで共有空間に置くものは，一つのサービスリクエストではなく，よりグローバルな依存関係のパターン(依存関係グラフ)である．ここでは，それをシナリオと呼び，メタレベルの情報に相当する．このスタイルの特徴は，このメタレベルの情報も計算の対象にすることである．

このシナリオは当初キャスト(すなわち依存関係に参加するエージェント)が未定であって，そのシナリオが共有空間上で公開されることで，そのシナリオに参加するキャストが募集される．そこには能力に関する条件が与えられていて，その条件にしたがってキャストが決まる．そのキャスト契約が共有空間のメカニズムの上でなされる．シナリオ遂行の管理も共有空間上で行うことができる．もちろん，その進行管理を監視する監督エージェントを配置してもかまわない．

依存関係グラフは，場合によっては手順に

なり，場合によってはデータ間の計算の依存関係である．

手順を与えるグラフ構造の場合には，前提がそろっているところからグラフ構造を食い潰しつつ計算をすすめる．そうしたグラフ構造の部分的な集まりが，機会主義的に計算できる部分から消化される．同じに，必要に応じて依存グラフ構造を結合したり，共有させるようなエージェントを動かすことで，複数の手順グラフを統合したり，同じ計算の繰り返しを避けることも試みる．

データ主導スタイルに適した問題に，このシナリオ主導スタイルを適用する場合には，少し様子が異なる．例えば，音素，語，句，文といったレベル分けされた構造が，この依存関係を担っていた(音声認識の場合には，この階層分けが明確である)．データ主導スタイルでは，このレベルの間の結合がエージェント内部の処理手順として与えられていたが，シナリオ主導スタイルでは，この依存関係をそのままデータとして共有空間上に置く．すなわちテンプレートとしての部分木構造がシナリオになる．

この場合には，そのシナリオをコピーしつつ，元から与えられているデータに当てはめていき，部分的な木構造を，データとして共有空間上に展開していく．そうして機会主義的に計算を進めると同時に，メタ情報に基づいて，データの間の相互の不完全データの補完や予測といったことを管理する．

5. シナリオ主導スタイルの設計

前章でシナリオ主導スタイルのコンセプトを与えたが，同スタイルを実現するにあたっては，シナリオのデータ表現とその操作に関して設計する必要がある．

基本的にグラフ構造を提供することになるので，共有空間上に置くためのノードとエッジに相当するオブジェクトのクラスを用意することになる．そのノードには，データを意味するオブジェクトが配置されたり，タスクを意味するオブジェクトが配置されたりする．そのグラフ構造の書き換えやそれに従ったマネージメントをするエージェントが存在する．それに従って，実際のデータを書き換えたり処理を遂行するエージェントが起動されたり，同期制御されることになる．

その基本的なノードとエッジからなるグラフ構造の下に，例えば継承を用いて，ペトリネットを表現するようなグラフ構造を与えるなど，多彩なグラフ表現が可能である．特にペトリネットは，同時並行性や機会主義性を元来，備えているグラフ表現である．

6. おわりに

データ主導とタスク主導の両方が表現できる枠組みとして共有オブジェクト空間を捉え，プログラミングスタイルを分類した．さらにその発展としてシナリオ主導スタイルを提案している．この提案は現時点では概念段階であり，実装実験による検証は未だ行っていない．

謝辞

共有データ空間の利用にいろいろな可能性がありそうだと目を開かせていただいたきっかけは1998年8月の宮本衛市先生の御講演[10]です．ここに深く感謝の意を表します．

参考文献

[1] N. Carriero and David Gelernter : How to write Parallel Programs – A First Course, The M.I.T. Press, 1990, 村岡洋一(訳)：並列プログラムの作り方，共立出版，1993

[2] N. Carriero and David Gelernter : How to write Parallel Programs – A Guide to the

Perplexed, ACM COmputing Serveys, Vol.21, No.3, 1989, 寺田実(訳): bit 別冊「コンピュータサイエンス」,共立出版, 1991

[3] Nicholas Carriero and David Gelernter : Linda in Context, CACM, Vol.32, No.4, 1989

[4] David Gelernter : Generative Communication in Linda, TOPLAS, Vol.7, No.1, 1985

[5] Satoshi Matsuoka and Satoru Kawai : Using Tuple Space Communication in Distributed Object-Oriented Languages, OOPSLA'88

[6] L.D. Erman, F, Hayes-Roth, V. Resser and D. Reddy : Hearsay-II speech understanding system : Integrating Knowledge to resolve uncertainty, ACM Computing Serveys, Vol.21, No.3, pp.323-357, 1989, 中島隆之(訳): Hearsay-II 音声理解システム:不確実性の解決のための知識の統合, bit11月別冊「コンピュータサイエンス'80」, 共立出版, 1981

[7] Frank Bushmann, Regine Meunier, Hans Rohnert, Peter Sommerlad and Michael Stal: A SYSTEM OF PATTERNS, PATTERN-ORIENTED SOFTWARE ARCHITECTURE, Wiley, 1996

[8] Mary Shaw and David Garlan : Software Architecture – Perspetives on an emerging Discipline, Prentice-Hall, 1996

[9] Sun Microsystems, Inc. : JavaSpace™ Specification, Revision 1.0, JavaSpace™ の仕様, 1999

[10] 宮本衛市:「招待講演:協調動作のための分散オブジェクト空間の記述」, 電子情報通信学会知能ソフトウェア工学研究会, KBSE98-12, 1998

[11] 渡辺慎哉, 赤間清, 宮本衛市 : Genericな相互作用を有する並行計算モデル – GIL, コンピュータソフトウェア, Vol.12, No.6, 1995

[12] 渡辺慎哉, 宮本衛市 : Generic Interaction Model に基づく並行プログラミング言語GIL/Cによる協調問題の設計, 日本ソフトウェア科学会第 15 回大会, 1998

[13] R.Smith: The Contract Net Protocol : High Level Communication and Control in a Distributed Problem Solving, IEEE, Trans. on Comp., Vol.29, No.12, 1980.