

協調プロトコルの混在を目的としたADIPSフレームワークにおけるエージェントアーキテクチャの設計

藤田 茂

E-mail: fujita@cs.it-chiba.ac.jp

千葉工業大学 情報工学科

あらまし

これまで分散処理システムのエージェント指向アーキテクチャとして研究開発してきたADIPSフレームワークに対して、複数の協調プロトコルを実装するためのエージェントアーキテクチャの設計を行った。この拡張によって、ADIPSフレームワーク上で動作するエージェントを作成する際に、利用しない協調プロトコルに対する記述を行うことが不要になり、また利用者が新たに協調プロトコルを導入することが可能になった。本稿で提案するエージェントアーキテクチャは、プロトコルエンジン、コントローラ、インタプリタの3要素を主な構成要素として設計された。このプロトコルエンジンが協調プロトコルを逐次解釈実行することで、目的である複数協調プロトコルの混在を実現している。

キーワード エージェントアーキテクチャ ADIPS フレームワーク 協調プロトコル メタアーキテクチャ

A Design of the Agent Architecture on ADIPS Framework for Multiple Cooperation Protocol Handling

Shigeru Fujita

E-mail: fujita@cs.it-chiba.ac.jp

Department of Computer Science, Chiba Institute of Technology

Abstract

In this paper, a new agent architecture in ADIPS framework is proposed. We design an ADIPS framework to develop a distributed agent oriented information processing system. Previous our agent architecture makes agent programmer to use only one protocol to cooperate among the agents. It's make to hard to use our environment, hence, we design a new agent architecture which is consisted by "protocol engine", "controller" and "interpreter". The protocol engine interprets cooperation protocol which described by agent programmer. A new agent architecture of ADIPS framework is able to handle multiple cooperation protocols by own protocol engine.

key words Agent Architecture ADIPS Framework Cooperation Protocol Meta-architecture

1 はじめに

我々はこれまで、分散処理システムのエージェント指向アーキテクチャによって構成するための枠組である“ADIPS フレームワーク”の研究開発を行って来た [1]. ADIPS フレームワークでは、リポジトリと呼ばれるエージェントの保管庫の中で、動的に利用者要求を充足する分散システムを構築し、また利用者に対してサービスを提供している途上での、計算機環境の変化や利用者要求の変化に追従して動的に分散システムを再構成する. このときエージェント間で利用される協調プロトコルは, “ADIPS 構成プロトコル”として全てのエージェント間で共通して用いられる協調プロトコルである.

これまでの ADIPS フレームワーク利用経験から, この共通プロトコルに基づいて全てのエージェントの動作を記述することが, エージェントプログラムの負担となっており, また, “ADIPS 構成プロトコル”以外の協調プロトコル導入のためにエージェントプログラムにプロトコル設計を行わせる必要があるという問題点が明らかになってきた.

これまで ADIPS フレームワーク上で動作するエージェントのアーキテクチャは, 協調機構, 知識処理機構, タスク実行機構の 3 要素から構成されていた. 協調機構はエージェント間メッセージを取り扱い, エージェントの処理可能なメッセージを知識処理機構に対して伝達する. 知識処理機構はエージェントプログラムのインタプリタであり, エージェント間メッセージとエージェント自体の状態に応じて, タスク実行機構に対する動作指示, あるいは協調機構を介してのエージェント間メッセージ送信を行う. タスク実行機構は, 分散処理システムを構成する計算機プロセスあるいはオブジェクトに対する制御監視を行い, 分散システムの構成要素として利用者に対するサービスの構成要素となる.

このアーキテクチャに基づいて構成されるエージェントに対するプログラムからみると, エージェント間協調プロトコルは, 協調機構の実装に際してハードコーディングされており, ひとたび ADIPS フレームワークの利用が開始された後の変更は困難であった.

本稿では ADIPS フレームワーク上で動作するエージェントのアーキテクチャに対する拡張設計について述べ, このアーキテクチャによってフレームワークの提供後に, エージェントプログラマが新たなエージェント間協調プロトコルを定義利用できることを示す.

2 ADIPS フレームワーク

“ADIPS フレームワーク”の概念図を図 1 に示す. ADIPS フレームワークは, エージェントプログラマ (開発者) が登録したエージェントの保管庫である “リポジトリ” と, そのリポジトリからコピー/移動によってエージェント組織を構成し, 利用者にサービスを提供するための “実行環境” から構成される.

リポジトリはネットワークを介して互いに通信可能な状態にあり, 実行環境からはサーバプログラムとして見え, 常時利用可能な状態におかれる. リポジトリ内部では実行環境からエージェント間メッセージを介して伝達される利用者要求を充足するために, エージェント組織構成が行われる. このときエージェント間協調プロトコルが利用される. エージェント組織構成は, 利用者に対してサービスを提供するために, 実行環境上にコピーとして生成される. もしくは, 以前に生成した履歴のあるエージェント組織であれば, リポジトリに格納された過去のエージェント組織を実行環境に移動させることで, 利用者にサービスを提供する.

実行環境は, リポジトリよりコピー/移動してくるエージェント組織の活動の場であり, 利用者に対してサービスを提供し, 実行環境とその動作しているネットワーク環境の状況監視を行い, 常に利用者要求を充足するために, エージェント組織の再構成/調整を実施する. このとき, エージェント組織内部のエージェント間で再構成や調整を実施する. このときエージェント間協調プロトコルが利用される.

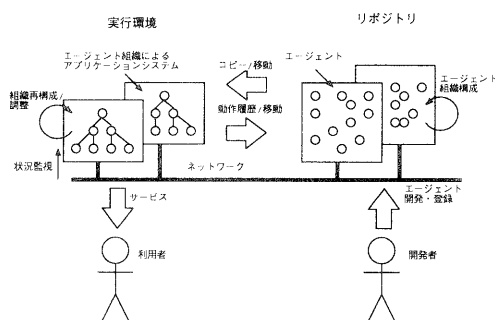


図 1: ADIPS フレームワーク概念図

3 エージェントアーキテクチャ

ADIPS フレームワークで動作するエージェントは, 全て同一のアーキテクチャより生成され, 規定フォー

マットに従ったエージェント間メッセージを交換して、目的のサービスを提供する。以下、本節では新しく定義した ADIPS フレームワーク上でのエージェントアーキテクチャを示す。まず、比較のためにこれまでのエージェントアーキテクチャを概説する。

3.1 エージェントアーキテクチャ(旧)

図 2 にこれまでのエージェントアーキテクチャを示す。協調機構はエージェントが処理可能なメッセージを取得し、メッセージハンドラへ、該当するメッセージを引き渡す。メッセージハンドラは、メッセージのパフォーマティブにより起動するべきエージェントプログラムを選択し、エージェントプログラムを処理機構へ引き渡す。この結果、エージェントプログラムの処理機構は、内部状態を更新し、(1) タスク実行機構に対する処理を実行する、(2) 他のエージェントに対するメッセージを生成し、協調機構に伝達する、(3) 内部状態を更新する、のいずれかのアクションを実行する。

このエージェントアーキテクチャでは、エージェント間の協調プロトコルは、協調機構とメッセージハンドラに対して分離して実装されており、リポジトリや実行環境に対して動的に協調プロトコルを追加するためには、一般的な問い合わせである *ask*, *tell* 等のメッセージに対するコンテンツをエージェントプログラマ間で整備し、疑似的なプロトコルとして利用する手法が用いられていた。この手法は、エージェントプログラマに対して、タスク実行機構を制御監視するという状態の他に、独自に導入されたプロトコルに対する記述も要求するという点で負担となっていた。また、一般的な問い合わせを独自に拡張していることから、本来の問い合わせを行ったときの処理が却って繁雑になる、独自プロトコルを導入しているエージェントと、導入していないエージェント間での協調が事実上不可能になっているという問題があった。

この問題に対して、本稿で提案するアーキテクチャでは、AgenTalk[2] による、エージェントアーキテクチャ独立な、エージェント間協調プロトコルの記述法を導入し、エージェントの実装と協調プロトコルを分離する。

また、エージェントプログラムはルールベース言語により行う必要があり、必ずしも知識処理言語に慣れていない分散処理システムの設計者にとっては、記述の負担が大きかった。この問題に対しては、エージェント組み込みの言語処理系としてエージェント

プログラマに提供するのではなく、複数の言語処理系を提供可能な形式でエージェントアーキテクチャを設計することで対処している。

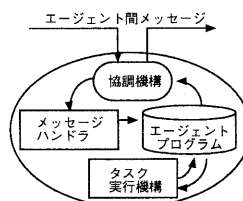


図 2: エージェントアーキテクチャ(旧)

3.2 エージェントアーキテクチャ(新)

図 3 に、本稿で定義した新しいエージェントアーキテクチャを示す。エージェントを特徴づける主な構成要素は、プロトコルエンジン、コントローラ、インタプリタである。

通信器はエージェント間メッセージとして受取可能なメッセージを全て受け取り、コントローラの明示的な指示により該当するメッセージをプロトコルエンジンへ引き渡す。また、コントローラを介して送信が指示されたメッセージをエージェントの動作環境である実行環境、リポジトリを介して他のエージェントに送信する。

タスク実行機構は旧アーキテクチャと同様であり、計算機プロセス、オブジェクトを制御監視し、その出力や状態をコントローラを介してインタプリタ、もしくはプロトコルエンジンへ受け渡す。

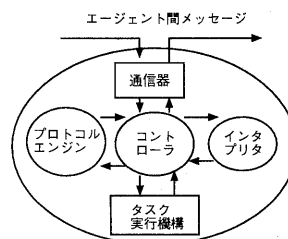


図 3: エージェントアーキテクチャ(新)

3.2.1 プロトコルエンジン

プロトコルエンジンは、図 4 に示すように、プロトコルインタプリタとプロトコル記述ファイルによ

て構成される。

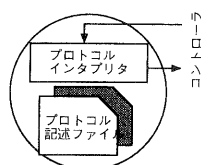


図 4: プロトコルエンジン

プロトコルエンジンは全てのエージェントがそれぞれインスタンスの形式で持つが、動作モデルは全て同一である。後述する“エージェント型”の定義により、それぞれのエージェントは処理すべき協調プロトコルを宣言されるので、その宣言に対応するプロトコル記述ファイルを持って、実行環境、リポトリ上で動作することになる。

このプロトコル記述ファイルは、新たに定義されたエージェントアーキテクチャを利用する ADIPS フレームワーク上で、これまでのエージェントプログラムと同様な形式で閲覧可能であり、新たにエージェントを記述するプログラマが利用可能な協調プロトコルがあるかを適宜検査することを可能にしている。

3.2.2 コントローラ

コントローラの内部機構は図 5 に示すように、他のエージェント内部構成要素からの内部メッセージを取り扱う弁とそれらの弁のコントローラからなる。弁の制御を実施するメカニズムの実装は現時点では実装言語である Java 言語のメカニズムに依存して設計している。

コントローラは、各内部モジュールからのメッセージをそれぞれのエージェント型に応じて適切な他の内部モジュールへ振り分ける。このコントローラの設計が、後述する“エージェント型”を決定する。

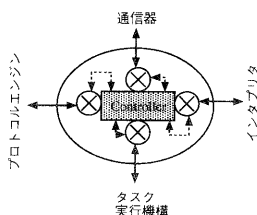


図 5: コントローラ

3.2.3 インタプリタ

インタプリタは、コントローラを介して、他のエージェント内部機構よりメッセージを伝達され、記述されたプログラムに基づいて動作する。エージェントプログラミングに先だって、プログラマが利用する情報は、エージェントの型とそのエージェントの利用する協調プロトコルである。この 2 つの情報はエージェントプログラムに対して、エージェントの振舞を示すとともに、付随して利用可能なインタプリタの仕様が明らかになることから、利用可能なプログラミングのモデルが明らかになる。現時点の想定では、エージェントプログラムのモデルとして、これまでの ADIPS フレームワーク上で用いられて来たルール型言語、手続き型言語、その他、必要に応じて論理型言語の導入を予定している。図 6 にインタプリタのモデル図を示す。

インタプリタ API は全てのエージェントに共通であるが、インタプリタモジュールは後述するエージェント型によって宣言されたモジュール（ルール型、手続き型、論理型）によって異なる。エージェントプログラムはそれぞれのモジュールによって解釈可能な形式で記述されて、リポトリに格納される。

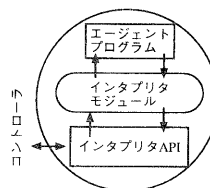


図 6: インタプリタ

4 協調プロトコル

本稿で述べたエージェントアーキテクチャに基づいて、エージェントをプログラミングすると、それぞれのプログラマが独自の協調プロトコルを勝手に定義し、結果としてそれぞれ作成したエージェント間での協調が促進されないという事態が予測される。これに対して、本フレームワークでは、あらかじめ必要と思われる複数の協調プロトコルセットをリポトリに準備し、この利用を推奨するという方式で対処している。

表 1 に、本稿で提案したエージェントアーキテクチャに実装が期待されているパフォーマティブセットと、対応して返答が期待されているパフォーマンス

ブを示す。

このように、エージェントプログラマに対して、ゆ
るい制約を設けることで、作成されるエージェント
間の協調をできるかぎり維持すると共に、これまでの
ADIPS フレームワークのように、エージェント
アーキテクチャと協調プロトコルが一体化されてい
ないことから、新たに協調プロトコルを導入しても、
それまでのエージェントプログラムの枠組を変更す
ることがないという特徴がでる。

5 エージェント型

エージェントの内部構造の中で、エージェント間
の協調を規定するプロトコルエンジン、エージェン
トインタプリタ、またそれらの間の制御を実施する
コントローラがそれぞれモデルとしてプログラマプ
ルになったことで、エージェントプログラマには、記
述しようとしているエージェントの振舞をあらかじめ
理解することが重要になる。

本稿で提案したエージェントアーキテクチャでは、
エージェントプログラム独立にエージェントの振舞
を決定するのは、プロトコルの選択、Java により実
装されているコントローラの選択、インタプリタの
選択による。この中で、プロトコル及び、インタプ
リタはそれぞれ独立して取り扱うことができる。そ
こで、プログラマに“型”として示すことができるの
は、コントローラの実装とその実装に際して利用を
想定したプロトコル、インタプリタの対である。図
7にエージェント型の定義例を示す。

エージェントプログラムを行う際には、この
TypeOfAgent 識別子を明示し、それぞれのインタプ
リタの規定する記述方式に従って、エージェントプ
ログラムを行うことになる。

6 おわりに

本稿では、これまで我々が研究開発してきた ADIPS
フレームワークのエージェントアーキテクチャを拡
張した新しいエージェントのアーキテクチャについ
て述べた。この新しいエージェントアーキテクチャに
より、協調プロトコルの混在を同一フレームワーク
上で実現し、新たなに設計開発される協調プロトコ
ルの自然な導入を実現した。現在、この設計に基づ
いて新しいフレームワークを設計実装中である。本
稿では触れなかったが、この新しいフレームワーク
の設計では、アーキテクチャの変更と、新たに記述対
象となったエージェントのプロトコル、コントロー

```
<agent>
  <identifier>
    TypeOfAgent
  </identifier>
  <controller>
    Protocol-depend-agent.class
  </controller>
  <protocols>
    <item>contract_net_protocol</item>
    <item>simple_request_protocol</item>
  </protocols>
  <interpreter>
    Rule_based_Interpreter
  </interpreter>
</agent>
```

図 7: エージェント型宣言

ラ、インタプリタの保存と利用が新たにリポジトリ
の機能として追加されている。

参考文献

- [1] 藤田, 他, “分散処理システムのエージェント指向
アーキテクチャ”, 情報処理学会誌, Vol.37, No.5,
pp.840-852, 1996 年
- [2] 桑原, 他, “Agentalk: マルチエージェントシス
テムにおける協調プロトコル記述”, 電子情報通
信学会論文誌 B-I, Vol.J79-B-I, No.5, pp.346 -
354, 1996

表 1: 基本プロトコルセット

送信パフォーマティブ	意図	返答を期待している パフォーマティブ	返答の意図
request-information	情報要求	information sorry request-information (null)	情報 返答不能 情報要求 (返答無し)
information	情報	agree disagree complete incomplete (null)	受諾 ¹ 否定 ² 受信 ³ 不完全 ⁴ (返答無し)
task-announce	公募	bid request-information (null)	入札 情報要求 (返答無し)
bid	入札	award request-information (null)	落札通知 情報要求 (返答無し)
cancel-bid	入札取消	agree disagree (null)	受諾 拒否 (返答無し)
award	落札通知	ready-to-do cancel-bid (null)	準備完了 入札取消 (返答無し)
directed-award	指名入札	agree disagree request-information (null)	受諾 拒否 情報要求 (返答無し)
request-action	動作要求	agree disagree request-information request-action (null)	受諾 拒否 情報要求 動作要求 (返答無し)
sorry	不能	(null)	(返答無し)
agree	受諾	(null)	(返答無し)
disagree	拒絶	(null)	(返答無し)
ready-to-do	準備 OK の表明	(null)	(返答無し)
(unknown)	利用者定義	(unknown) (null) unknown sorry	定義依存 (返答無し) 不明通知 不能
unknown	不明通知	(null)	(返答無し)

¹ 情報を肯定することができる² 情報を否定することができる³ 情報を肯定も否定もできないが、情報として受け入れることができる⁴ 情報を肯定も否定もできないし、情報としても不完全であるので、受け入れられない