F-011

DASH-JADE 間におけるエージェント相互運用機構の開発
Development of Agent Interoperability Mechanism between DASH and JADE

奥村 文雄† 打矢 隆弘† 内匠 逸†
Fumio Okumura Takahiro Uchiya Ichi Takumi

1. はじめに

近年のネットワークの高度化に伴い，ユーザが様々な環境からネットワークを用いたサービスを利用する機会が増えている．そのため，ユーザの環境や要求の変化に動的に対応する技術としてエージェントが注目されている．

エージェントを動作させる基盤となるソフトウェアの事をエージェントプラットフォーム(以下 AP)と呼び，用途や目的に合わせて様々な特徴を持ったものが開発されている．しかし，それぞれの AP は通信やメッセージ形式の仕様が異なっているために，他の AP 上のエージェント(以下異種エージェント)同士が直接連携する事は困難となっている．

そこで本稿では異種エージェント同士の連携を可能とするための相互運用機構を提案する．異種エージェントの連携が可能となると，自 AP 上のエージェントでは解決が困難なタスクに対する包括的な解決が望める．具体的には，各 AP の長所に合わせたタスクを依頼する事で効率的なタスクの実行が可能となる．また，それぞれの AP 上には既に開発されている様々なエージェントが存在しており，代替として利用可能なエージェントの選択肢が増える事によって，より状況に適したエージェントの利用が図れる．

本稿で提案する相互運用機構はそれ自体を各 AP 上で動作するエージェント(以下相互運用エージェント)として実装し，異種エージェントとの通信の仲介を行う．各 AP の仕様が変更された際は，相互運用機構におけるエージェント動作部分のみの変更によって追従が可能となるために，対応が容易となる．また，各エージェントは自 AP 上のエージェントとの通常の通信と同様に異種エージェントとの通信を行う事ができる．

本稿において相互運用の対象とする AP は DASH と JADE である．DASH はリポジトリと呼ばれるエージェントの集積所を有し，エージェントの動的な組織化，組織再構成を特徴としている．また，エージェントの動作はルールベースの推論システムによる実装であり，知的な処理を長所としている．JADE はエージェント標準化団体 FIPA が規定した標準仕様に準拠しており，現在最も普及している．そのため，相互運用に向いており，既存のエージェントも豊富である．

相互運用機構の概略図を図 1 に示す．

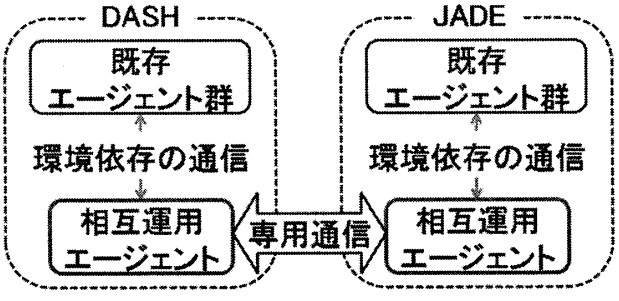

図 1：相互運用機構

†名古屋工業大学 大学院 工学研究科 情報工学専攻
Nagoya Institute of Technology

2.相互運用

異種エージェントの相互運用において必要となる機能として以下の 3 つの項目が挙げられる．

(R1)通信路レベルの相互運用

各 AP 上の相互運用エージェント間でデータのやり取りを行うため，専用の通信路を実装する必要がある．

(R2)メッセージレベルの相互運用

エージェントはメッセージ交換によりタスクの処理を行うが，メッセージの仕様は各 AP によって異なっている．そのため，双方のメッセージ形式へ変換を行い，メッセージ形式の差異を吸収する必要がある．

(R3)プロトコルレベルの相互運用

異種エージェントの協調動作をより堅固に行うために，インタラクションプロトコルに従った通信を行う必要がある．インタラクションプロトコルとはエージェント間通信に用いられる代表的なメッセージ交換手順の事であり，用途に合わせてあらかじめ定義されている．インタラクションプロトコルに従う事によってメッセージの意図が明確化され，相手エージェントの目的，目標の把握が容易となる．これにより，開発者の異なるエージェント間の通信を効率的に行うことが可能となる．本研究では FIPA によって規定されている FIPA プロトコルを用いる．

3. 相互運用機構の実装

3.1 相互運用機構の設計

相互運用機構を実現するために先に述べた各レベルに応じた機能をモジュールとして実装した．

(M1)通信路管理モジュール

このモジュールは以下の 3 つの要素により各 AP 上の相互運用エージェント間での専用通信に用いる通信路を実現する(図 2)．

- **通信路**
  DASH，JADE 共に Java 言語による実装であるため，Java の API で提供されているソケット通信を用いて実現する．
- **相互運用専用メッセージキュー**
  各相互運用エージェントには，各 AP が提供する通常のメッセージキューとは別に，他の相互運用エージェントから受信したメッセージを格納する専用メッセージキューが必要となる．エンキューはエージェントのスレッドによって，デキューは通信管理スレッドによって不定期に行われるため，用意するキューは同

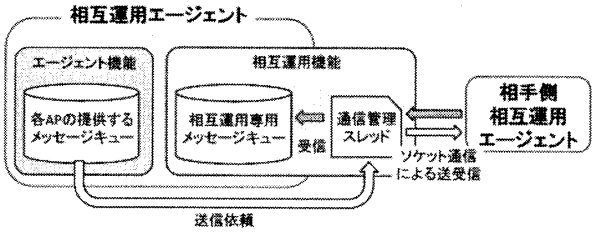

図 2：通信路管理モジュール

期されている必要がある．

- **通信管理スレッド**
  エージェントは AP によって実行権が管理されており，メッセージ送信時に相手エージェントのスレッドが停止している可能性もある．そのため，確実な通信を行うためにはソケット通信を管理するスレッドをエージェントのスレッドとは別に動作させなくてはならない．

(M2)メッセージ管理モジュール

このモジュールでは以下の 3 つの処理により他の相互運用エージェントから転送されたメッセージを自 AP におけるメッセージ形式へ変換する処理を行う．

- **各メッセージ形式に合わせたパラメータの移行**
  DASH と JADE ではメッセージ形式が異なっているが，主要なパラメータはそのまま値を移行する事によって用いる事が可能である．以下に対応関係のあるパラメータを表で示す(表 1)．異種エージェント間通信の際は，送信(受信)エージェントアドレスには仲介を行う相互運用エージェントのエージェントアドレスが格納され，通常の通信時と同様に通信を行う．
- **新たなパラメータの導入**
  通信時に相互運用エージェントが仲介するに伴い，本来の送信(受信)エージェントアドレスを表すパラメータを導入する．新たなパラメータの導入には，ユーザ定義パラメータを用いる．ユーザ定義パラメータとは AP がシステムのために用いるパラメータ以外に，エージェント開発者が独自に定義して使用できるパラメータであり，DASH，JADE 共に利用する事ができる．本来の送信(受信)エージェントを表すパラメータをそれぞれ“dashAddress”，“jadeAddress”と定義して用いる．
- **メッセージストレージの導入**
  本稿で提案する手法では相互運用エージェントが仲介を行うために，返信メッセージの取り扱いが通常とは異なる．通常，返信メッセージを送信する際は，送られてきたメッセージから正常にパラメータを引き継ぐ必要がある(図 3)．

表 1：パラメータの対応関係

パラメータの種類	DASH	JADE
メッセージの種類	performative	performative
送信エージェントアドレス	from	sender
受信エージェントアドレス	to	receiver
メッセージ ID	replyWith	reply-with
返信元メッセージ ID	inReplyTo	in-reply-to
メッセージ内容	content	content

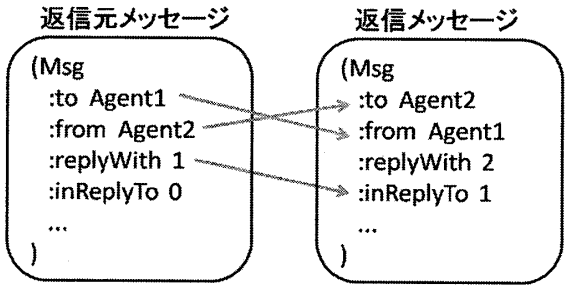

図 3：返信時のパラメータ引き継ぎ

本来の受信エージェントからどのようなタイミングで返信メッセージが送信されるか不明であるため，受信したメッセージを返信作成用に保存しておかなくてはならない．そこでメッセージを保存するための受信メッセージストレージを導入する．本来の受信エージェントから返信メッセージが送信された場合は，そのメッセージが過去に送られてきたどのメッセージに対する返信かを検索し，正常にパラメータの移行を行った上で，相手 AP 上の相互運用エージェントへとメッセージを転送する．

(M3)プロトコル管理モジュール

このモジュールではインタラクションプロトコルを用いた通信を実現するために，メッセージにおいて不足しているパラメータを補完する．本研究では JADE で用いられている FIPA プロトコルを用いるために，パラメータを補完する対象は DASH エージェントから送信されるメッセージである．補完するパラメータはユーザ定義パラメータを用いて導入する．パラメータの補完は，DASH 上のエージェントから相互運用エージェントへメッセージが到着した際に，そのメッセージがプロトコルを開始するメッセージであった場合に行う．

- **conversation-id パラメータの補完**
  インタラクションプロトコルに基づいた一連のメッセージやり取りを“会話”と呼び，それらを識別するユニークな値を表すパラメータが conversation-id である．相互運用エージェントを起動する際に，ID のベースとして，エージェントの実行プログラムのハッシュコードと時刻を設定する．そして新たな ID を生成する度にカウントアップされる数字を末尾に付加する事によりユニークな ID の生成を保証する．
- **protocol パラメータの補完**
  protocol はその会話で用いられているプロトコル名を表すパラメータである．protocol パラメータを補完するには，プロトコルを開始するメッセージの performative パラメータの値から推測を行う．performative の値とインタラクションプロトコルの対応例を表で示す(表 2)．

表 2：performative と protocol パラメータの対応例

performative	protocol
request	FIPA-Request
query	FIPA-Query
subscribe	FIPA-Subscribe
cfp	FIPA-Contract-Net FIPA-Iterated-Contract-Net

一度補完した protocol パラメータは一連の会話が終了するまで同一の値を用いるが，正しく補完できていない可能性もあるため，例外処理が必要となる．1 つ目は，performative が cfp の場合対応する protocol の値が 2 種類取りうる場合である．この場合，プロトコルを開始するメッセージのみでは判断ができないために，protocol の値を変えた 2 種類のメッセージを作成し，相手エージェントからエラーメッセージが返ってきた方の会話を中止する等の場当たり的な対処を行わなくてはならない．2 つ目は，エージェント間通信は必ずしもインタラクションプロトコルに準拠した通信である必要はなく，開発者が独自に定めたメッセージ交換手順による通信も可能となっている点である．本研究ではそのようなプロトコルは unknown プロトコルとして処理を行い，conversation-id を割り当てるのみとし，プロトコルのチェックは行わない．

3.2 相互運用機構の動作

各 AP 上に配置した相互運用エージェントは実行権が回ってくる度に以下の 2 種類の動作を行う．

- **他 AP 上の相互運用エージェントへのメッセージ送信（図 4）**

step1　各 AP の提供するメッセージキューから送信するべきメッセージを取り出す．

step2　そのメッセージがプロトコル開始メッセージならば(M3)を用いて conversation-id，protocol パラメータの値を補完する．

step3　(M2)を用いて受信メッセージストレージに保存する．

step4　(M1)を用いて他 AP 上の相互運用エージェントへメッセージを送信する．

- **自 AP 上エージェントへのメッセージ転送（図 5）**

step1　相互運用専用メッセージキューから転送するべきメッセージを取り出す．

step2　そのメッセージを(M2)を用いて自 AP のメッセージ形式へと変換する．

step3　そのメッセージが返信メッセージであるなら，(M2)を用いて受信メッセージストレージを検索し，返信元メッセージから必要なパラメータの引き継ぎを行う．

step4　(M2)を用いて受信メッセージストレージに保存する．

step5　本来の送信先である自 AP 上のエージェントへ通常の通信により送信する．

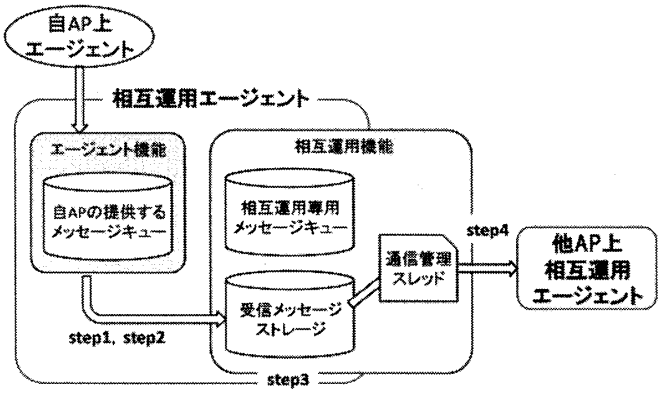

図 4：他 AP 上の相互運用エージェントへの送信

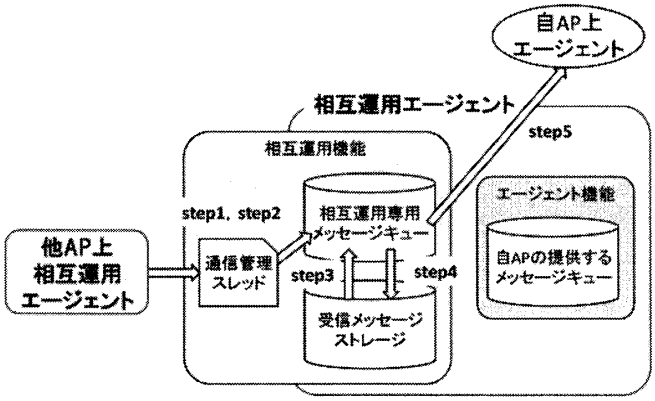

図 5：自 AP 上エージェントへの転送

3.3 GUI の実装

異種エージェントとの連携は通常のエージェント開発以上に複雑さが増す．正しくメッセージの変換が行われたか，意図したインタラクションプロトコルに従った通信が行われたか確認するために専用の GUI を実装した．

通信設定 GUI

他の相互運用エージェントとのソケット通信用に送信先ホスト名，ポート番号，受信用ポート番号を設定する(図 6)．

フローチャート表示 GUI

相互運用エージェントによって判断されたインタラクションプロトコルの種類に合わせたフローチャートを表示する（図 7）．一連のやり取りされたメッセージの conversation-id パラメータの値をチェックする事によって会話ごとに分類してフローチャートを表示する．

図 6：通信設定 GUI

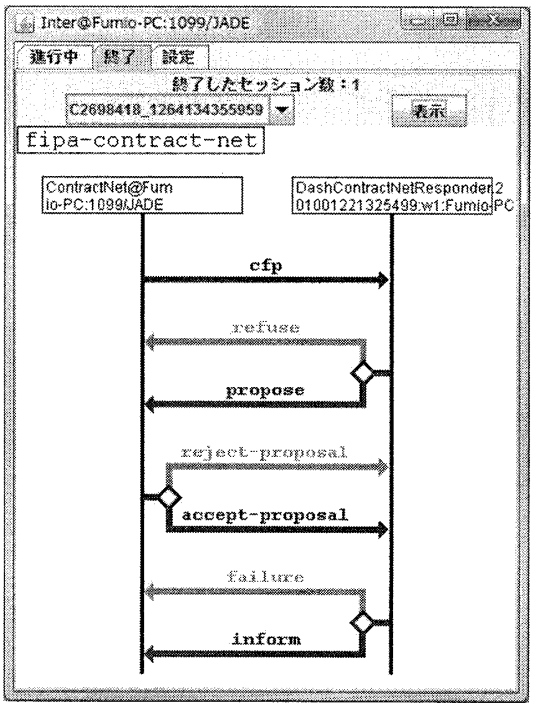

図 7：フローチャート表示 GUI

変換後メッセージ表示 GUI

フローチャート表示 GUI の矢印部を選択する事で，送信したメッセージを相手 AP の形式へ変換したものが表示される（図 8）．

4. 動作確認

作成したシステムの動作を確認するために実験を行った．プロトコル管理モジュールの動作を確かめるために DASH 上エージェントから JADE 上エージェントへの通信を行い protocol，conversation-id パラメータの値が正常に補完されたか確かめる．各 AP 上には，提案した相互運用エージェントを配置し，各プロトコルに合わせて DASH 上にはプロトコルを開始するイニシエータエージェント，JADE 上には返信を行うレスポンダエージェントを配置する．配置した通信エージェントはタスク処理等を行わずメッセージのやり取りのみを行う．

4.1 FIPA-Request プロトコル

FIPA-Request プロトコルは，プロトコル開始メッセージから一意にプロトコルが判別する事ができる．そのため，(M3)プロトコル管理モジュールによって，protocol パラメータに"fipa-request"の値，conversation-id パラメータに"ConvID226290005296_0"というユニークな値を正しく補完する事ができた．また，(M2)メッセージ管理モジュールによって JADE のメッセージ形式に変換する事ができ，その結果 FIPA-Request プロトコルに従った通信を正常に完了する事ができた．

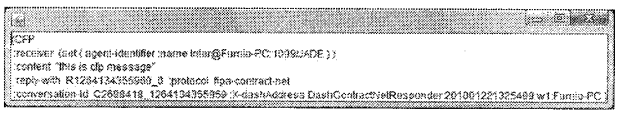

図 8：変換後メッセージ表示 GUI

4.2 FIPA-Contract-Net プロトコル

先に述べたように，プロトコル開始メッセージからでは従っているプロトコルが判断できないために，protocol パラメータの値には"fipa-contract-net"の値を入れたメッセージと"fipa-iterated-contract-net"の値を入れたメッセージの 2 種類を生成して送信する．

通常，エージェントは想定外のメッセージが到着した際に，not-understood メッセージを返信する等の例外処理が発生する．今回の実験でも"fipa-iterated-contract-net"の値を入れたメッセージを受信した際は例外メッセージを返信する．それを利用して，例外メッセージが返信された方のメッセージはパラメータの値が不適であったと判断し，相互運用エージェントによりそのセッションは破棄される．正しくパラメータの補完が行われた方のメッセージは処理を継続する．

結果的に，(M3)プロトコル管理モジュールによって，protocol パラメータに"fipa-contract-net"の値，conversation-id パラメータには"ConvID226290005296_1"というユニークな値を正しく補完する事ができた．また，(M2)メッセージ管理モジュールによって JADE のメッセージ形式に変換する事ができ，その結果 FIPA-Contract-Net プロトコルに従った通信を正常に完了する事ができた．

2 種類のメッセージを送信する方法では，必ず不必要なメッセージの送受信が発生してしまうために，動作するエージェント数が増加した場合に負荷が増大する恐れがある．

5. まとめ

既存エージェントの有効活用，各 AP の長所を生かした処理によるエージェントシステムの向上を目的とし，DASH-JADE 間における相互運用機構を提案した．

今後の課題には以下のものがある．

- protocol パラメータの補完
  補完候補が複数ある場合，現在は場当たり的に対処する方法を取っており改善が必要である．
- 相互運用エージェントへの負荷実験
  異種エージェント間通信は全て相互運用エージェントが仲介を行うため，負荷が集中する設計となっている．そのため，負荷の集中によるメッセージの遅延が発生し，他のエージェントに影響を与える可能性がある．
- 処理を行うエージェントを配した動作実験
  動作実験を行ったエージェントはタスク処理を行っていないため，シンプルなメッセージのやり取りであった．実際に開発されたエージェントではメッセージに独自のパラメータやプロトコルを用いている場合が考えられるため，動作実験を行う必要がある．

参考文献

[1] "DASH マニュアル"
http://uchiya.web.nitech.ac.jp/idea/html/
[2] "JADE –Java Agent Development Framework"
http://jade.tilab.com/