LISPを用いた知的エージェントシステムの構築

Building an Intelligent Agent System Using Lisp Language

川村旭　北島弘伸　寺本良明　佐藤陽　益岡竜介
Akira Kawamura　Nobuhiro Kitajima　Yoshiaki Teramoto　Akira Satoh　Ryusuke Masuoka

富士通研究所　知能システム研究部
Intelligent Systems Laboratory, Fujitsu Labs.

我々は、知的エージェントによって異種情報処理システムの統合を行なう環境であるSAGE(Smart AGent Environment)の研究開発を行なっており、その成果として、富士通製品INTERSTAGEのオプションとしてAGENTPROを製品化している。我々は、エージェント、特に情報統合の要となるFacilitatorの実装にLISPを用いた。本論文では、知的エージェントシステムの構築についての説明を行なうとともに、その実装にLISPを採用した理由や、実装上の問題点と、それに対してCLOSを導入した解決方法について報告する。

１．背景

インターネット及びイントラネット等は、共通の規約即ちインターネットプロトコルに従う限り、如何なるプラットフォームのコンピュータ及びネットワークの接続も可能とするオープンなネットワークとして構築されており、情報システム同士のオンラインでのデータ交換が一般的なものになるのに大きな役割を果たしている。

また、WWWブラウザがインターネット及びイントラネット等上の情報へのアクセス手段として広く普及しているので、それを標準インタフェースとして用いることで、新規に構築する情報処理システムへのアクセス手段を幅広い想定ユーザにとって簡便で受け入れられ易いものとすることが可能となっている。

それらのインターネット関連技術の発展・普及と表裏一体の関係を持ちながら、これまで分散・局在していた情報・知識の流通・共有・結合を行なうことによって、従来の業務の効率化・高度化や新たな事業分野の創造を行なうことを目指す動きが活発になっている。

情報・知識の流通・共有・結合を行なう情報処理システムを構築するためには、情報・知識が電子化されたものについては、それらが載っている情報処理システム同士を接続し、連携を行なわせる作業が必要となり、電子化されていない情報・知識については、情報処理システムと接続・連携させるために、それらを電子化する作業が必要となる。そのとき、良い情報システムを構築するためには、以下の条件を満たすことが重要となる。

- インターネット関連技術及びその影響を受けて発達中の事業分野の激しい状況の変化に対応する必要があるので、構築する情報処理システムは、固定的なものとして一から作り込むのではなく、既存資産をできるだけ生かして、後で流用可能な共通部品の組合せとなるように構築し、既存システムからのスムーズな移行や、システムの一部の動的変更等が容易なものとしなければならない。
- インターネット及びイントラネット等上に構築された既存の情報処理システム同士の連携については、一般的に、ネットワーク上に分散して存在する情報源・情報処理システムの各々の運営主体の運営ポリシー・利害関係が異なる。また、連携自体についても、期間や内容に関する限定があることも有り得る。したがって、情報システム間連携の運用形態として、情報源やその内容を固定的・一元的なものとして扱い、集中管理する旧来のものではなく、分散したものを分散したまま、かつ、個々の情報源・情報処理システムの各々の運用の主体性やその内容の多様性を保ったまま、流動的・多元的なものとして扱わなければならない。

上記の条件を満たした情報処理システムを構築するために、我々は、エージェント系による異種情報処理システムの統合を行なう環境であるSAGE(Smart AGent Environment)の研究開発を行なっており、その成果として、富士通製品INTERSTAGE [1]のオプションとしてAGENTPROを製品化している。

２．SAGE解説

2.1 SAGEに於けるエージェントとは

SAGEに於けるエージェントの概念は、ネットワーク上に分散して存在し、各々独立して動作し、ひとまとまりとして運用・管理される機能の提供を行なうプログラムのことである。他カテゴリのエージェントと区別するときは、知的エージェントまたは会話型エージェントと称する。

エージェントシステム内の各エージェントは、他のエージェントとメッセージを交換することで、全体として目的の情報処理を実現するように協調動作を行なう。それらのメッセージの規約としては、ACL(Agent Communication Language)[2]を採用している。ACLは、エージェント間通信言語であり、KQML(Knowledge Query and Manipulation Language)[3]とKIF(Knowledge Interchange Format)[4]の２階層で定義されている。KQMLは型付けられたメッセージ記述言語であり、KIFはメッセージの内容そのものを記述するための一階述語論理に基づいた言語である。

2.2 典型的なエージェントシステム構成

図１にSAGEのエージェントシステム構成の典型例を示す。このエージェントシステムは、User-Agent，DB-Agent，Facilitatorという３種類のエージェントで構成さ

れている。User-Agentは、ユーザの検索要求・結果表示をWWWブラウザ上等で受け付けるためのユーザインタフェース機能と、その検索要求・結果をACLのメッセージとして送受信する機能とを提供する。DB-Agentは、データベース等の情報源と結合し、検索要求メッセージを受信し、情報源を検索した結果のメッセージを回答送信する機能と、どのような検索要求を受理するかの情報を宣伝メッセージとして送信する機能とを提供する。Facilitatorは、User-AgentとDB-Agentとのメッセージ交換の仲介機能を提供する。

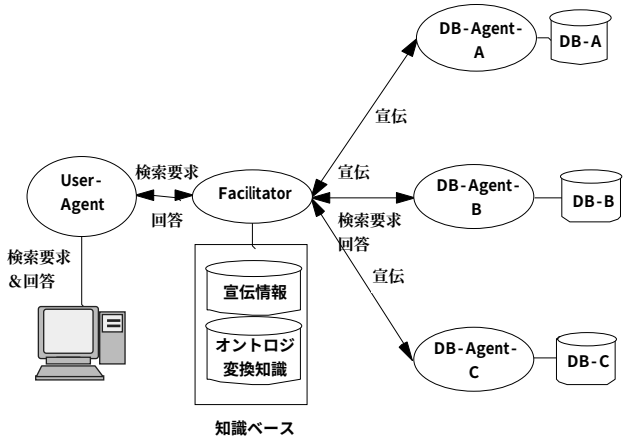

図 1．エージェントシステム構成図

2.3 Facilitatorの機能とメッセージ例の説明

図１のエージェントシステムに於て、エージェント間のメッセージ交換の要であるFacilitatorの機能の説明を、具体的なメッセージがどのように処理されるかを述べることで行なう。

図 2にFacilitatorのユースケース図を示す。

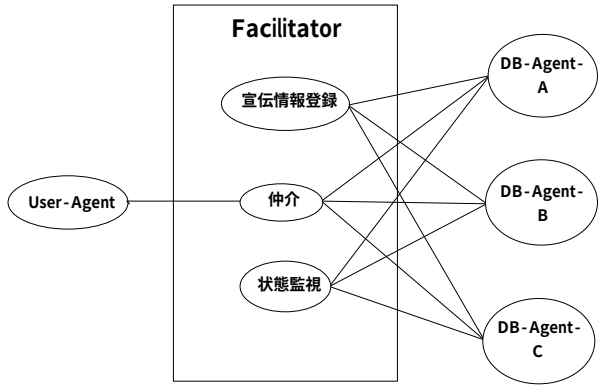

図 2．Facilitatorのユースケース図

Facilitatorは、DB-Agentからの宣伝メッセージの情報を、内部に持つ知識ベースに登録する機能と、ユーザエージェントからの検索要求メッセージに対して、それに相応しいDB-Agentを知識ベース内の宣伝情報を参照することで選択して、検索要求メッセージを回送し、それらの回答を待ち合わせて収集し、まとめたものをUser-Agentに送信するという仲介機能とを持つ。仲介の際には、必要に応じて、知識ベース内のオントロジ変換知識を用いて、DB-Agentが処理できるようにメッセージの内容の変換を行なう。また、DB-Agentの状態を監視して、検索要求の回送の制御に用いる機能も持つ。

2.3.1 宣伝情報登録機能とオントロジ

Facilitatorの知識ベースに登録するために、DB-AgentからFacilitator宛に送信される宣伝メッセージの説明と共に、エージェント間のメッセージ交換のために必要となるオントロジについての説明を行なう。

2.3.1.1 宣伝メッセージ

エージェント間で交換されるメッセージ一般はKQMLの規約に則っている。メッセージはリストであり、その文字列としての記述はCOMMON LISPのリーダで読み取り可能である。リストの最初の要素は、performativeと呼ばれる、メッセージを送信したエージェントが意図する行為を表わすシンボルである。２番目以降の要素は、メッセージの各パラメタを表わす「:」で始まるシンボルと、そのパラメタの値との対の並びが順不同で続く。

また、パラメタの一つである:contentの値は、KIF文である。KIF文は、リレーションと呼ばれるリストの論理式として構成される。リレーションのリストの最初の要素は、リレーション名と呼ばれ、リストの残りの要素間の関係を表わす。また、リレーションは述語である。即ち、それを評価した値が真または偽となる。

ここで、特に宣伝メッセージの:contentの値は、DB-Agent自体に関する属性・能力と、DB-Agentがラッピングしているデータベース等の情報源が取り扱う情報の内容に関するリレーションで構成されており、受信したときの宣伝情報登録作業として、その各々のリレーションはそのままの形式で、Facilitatorの知識ベースにassertされる。ただし、どのような形式・内容のKIF文が受理可能かを表わすhandlesリレーションのみは、以下のようにメッセージの仲介時のマッチング判定のための推論に用いられるルールの形式に変換された後に知識ベースにassertされる。

宣伝メッセージ中の形式：

```
(handles momochihama-dba momochihama
         '(カテゴリ?x1 ?x2)
         '(kif-variable ?x2))
```

知識ベースへのassert時の形式：

```
(<= (handles momochihama-dba ?p)
    (and (== ?p (カテゴリ?x1 ?x2))
         (kif-variable ?x2)))
```

このように変換してassertすることで、handlesリレーションの３番目の引数に指定した任意のKIF文の形式と、４番目の引数に指定した任意のKIF変数に対する制約条件（例えば、'(and (kif-variable ?x2)(< 10 ?x2)(> 40 ?x2))等）とを、DB-Agentが扱えることを推論で判定可能となる。

宣伝メッセージの例（DB-Agent-A）：

```
(advertise+
  :reply-with DBA-A-20000518-1234
  :protocol sage_advertise_1.0
  :language-encoding euc-jp
  :language KIF
  :ontology ebisu
  :content
```

```
  (and
    (handles-ontology DB-Agent-A ebisu)
    (needs-class-expansion DB-Agent-A)
    (handles-protocols DB-Agent-A
                       '(sage_ask-all_1.0
                         sage_management_1.0
                         sage_advertise_1.0
                         sage_unadvertise_1.0
                         sage_check-status_1.0))
    (handles-logical-combination-of
              -acceptable-sentences DB-Agent-A)
    (handles DB-Agent-A ebisu '(= ?x1 ?x2) 'true)
    (handles DB-Agent-A ebisu '(< ?x1 ?x2) 'true)
    (handles DB-Agent-A ebisu '(> ?x1 ?x2) 'true)
    (handles DB-Agent-A ebisu '(=< ?x1 ?x2) 'true)
    (handles DB-Agent-A ebisu '(>= ?x1 ?x2) 'true)
    (handles DB-Agent-A ebisu '(SITEMX ?x1) 'true)
    (handles DB-Agent-A ebisu
                      '(CTGRYDAIKANJI ?x1 ?x2)
                      '(kif-variable ?x2))
    (handles DB-Agent-A ebisu
                      '(ITEMKANJI ?x1 ?x2)
                      '(kif-variable ?x2))
    (handles DB-Agent-A ebisu
                      '(DISCORPKANJI ?x1 ?x2)
                      '(kif-variable ?x2))
    (handles DB-Agent-A ebisu
                      '(PRICE ?x1 ?x2)
                      '(kif-variable ?x2))
    (handles DB-Agent-A ebisu
                      '(GRADE ?x1 ?x2)
                      '(kif-variable ?x2))
    (handles DB-Agent-A ebisu '(畜産 ?x1) 'true)
  ) ;; and
 :sender DB-Agent-A
 :receiver Facilitator
 )
```

宣伝メッセージの例（DB-Agent-B）：

```
(advertise+
  :reply-with DBA-B-20000518-4567
  :protocol sage_advertise_1.0
  :language-encoding euc-jp
  :language KIF
  :ontology momochi
  :content
    (and
      (handles-ontology DB-Agent-B momochi)
              :
      (handles DB-Agent-B momochi '(商品リスト ?x1) 'true)
      (handles DB-Agent-B momochi
                      '(取引先名 ?x1 ?x2)
                      '(kif-variable ?x2))
      (handles DB-Agent-B momochi
                      '(商品名 ?x1 ?x2)
                      '(kif-variable ?x2))
      (handles DB-Agent-B momochi
                      '(カテゴリー ?x1 ?x2)
                      '(kif-variable ?x2))
      (handles DB-Agent-B momochi
                      '(商品価格 ?x1 ?x2)
                      '(kif-variable ?x2))
      (handles DB-Agent-B momochi
                      '(和牛 ?x1)
                      'true)
    ) ;; and
 :sender DB-Agent-B
 :receiver Facilitator
 )
```

2.3.1.2 オントロジ

これらのDB-Agentの宣伝メッセージに於て、handlesリレーション記述中の個々のリレーションは、各々のDB-Agentがラッピングしている情報源が取り扱うの情報の内容にどのようなものが有るかを表わしている。

ここで、一般的には情報源毎に、その提供する情報の記述に使われるオントロジ（用語の体系）は異なっているので、同一の内容に対して異なる用語を割り当てている。

Facilitatorでは、それらのオントロジの違いを吸収するために、複数のオントロジの情報と、オントロジ間での用語の対応関係（オントロジ変換ルール）の情報も、メッセージ処理を開始する前に知識ベースにassertする。

以下に、２つのオントロジmomochiとebisuとをFacilitatorに登録するためのリレーション（の抜粋）を示す。

オントロジmomochiについては、検索要求メッセージで使用するデータベース（表）名・項目名・項目値についての記述の例を示すが、オントロジebisuについては、項目値の特別なものであるカテゴリーの構成（ツリー構造）に関する記述の例のみを示す。

オントロジmomochi(User-AgentとDB-Agent-B用)：

```
(class 商品リスト)
(single-valued-slot 商品リスト 取引先名)
(single-valued-slot 商品リスト 商品名)
(single-valued-slot 商品リスト カテゴリー)
(single-valued-slot 商品リスト 商品価格)

(unary-function 取引先名)
(slot-value-type 商品リスト 取引先名 string)
(range-of string 取引先名)

(unary-function 商品名)
(slot-value-type 商品リスト 商品名 string)
(range-of string 商品名)

(unary-function カテゴリー)
(slot-value-type 商品リスト カテゴリー string)
(range-of (union 畜産物 農産物) カテゴリー)

(unary-function 商品価格)
(slot-value-type 商品リスト 商品価格 number)
(range-of number 商品価格)

(class 畜産物)
(subclass-partition 畜産物 (setof 牛))

(instance "畜産物")
(direct-instance-of "畜産物" 畜産物)

(class 牛)
(direct-subclass-of 牛 畜産物)
(subclass-partition 牛 (setof 和牛 輸入肉))

(instance "牛")
(direct-instance-of "牛" 牛)

(class 和牛)
(direct-subclass-of 和牛 牛)

(instance "和牛")
(direct-instance-of "和牛" 和牛)

(class 輸入牛)
(direct-subclass-of 輸入牛 牛)
```

```
(instance "輸入牛")
(direct-instance-of "輸入牛" 輸入牛)

(instance "農産物")
        :
        :
```

オントロジebisu(DB-Agent-A用)：

```
(class 畜産)
(subclass-partition 畜産 (setof 牛肉 豚肉))

(instance "畜産")
(direct-instance-of "畜産" 畜産)

(class 牛肉)
(direct-subclass-of 牛肉 畜産)

(instance "牛肉")
(direct-instance-of "牛肉" 牛肉)

(class 豚肉)
(direct-subclass-of 豚肉 畜産)

(instance "豚肉")
(direct-instance-of "豚肉" 豚肉)
```

以上の２つのオントロジ間で共通する用語同士の対応関係を付けて、メッセージ中に表れるそれらの用語を変換するための関数の指定と共にFacilitatorに登録するためのリレーション記述を列挙したものが以下のオントロジ変換ルールである。

オントロジ変換ルール：

```
(translation ebisu momochi
  (sentence-translation '(SITEMX ?x1) '(商品リスト ?x1)))

(translation ebisu momochi
  (sentence-translation
    '(CTGRYDAIKANJI ?x1 ?x2) '(カテゴリー ?x1 ?x2)))

(translation ebisu momochi
  (sentence-translation
    '(ITEMKANJI ?x1 ?x2) '(商品名 ?x1 ?x2)))

(translation ebisu momochi
  (sentence-translation
    '(DISCORPKANJI ?x1 ?x2) '(取引先名 ?x1 ?x2)))

(translation ebisu momochi
  (sentence-translation
    '(PRICE ?x1 ?x2) '(商品価格 ?x1 ?x2)))

(translation ebisu momochi
  (sentence-translation '(畜産 ?x1) '(畜産物 ?x1)))

(translation ebisu momochi
  (instance-translation "畜産" "畜産物"))

(translation ebisu momochi
  (instance-translation "牛肉" "牛"))

(translation ebisu momochi
  (sentence-translation '(牛肉 ?x1) '(牛 ?x1)))
```

2.3.2 仲介機能とメッセージ例の説明

Facilitatorの仲介機能を、下のUser-Agentからの検索要求メッセージの処理を具体例として説明する。ここで、:contentの値は、検索要求しているデータのレコード自体を?xとして、その?xが満たすべき条件を表わすKIF文となっている。:aspectの値は、User-Agentへの回答メッセージの:contentである（複数の）レコードの一つ一つに含まれるべき項目の並びを表わしている。また、:meta-info の値として、複数のDB-Agentから集めた回答の（複数の）レコードをどのように処理するかを指定する。下の場合は、ソート関数による並べ替え処理を指定している。

検索要求メッセージ（User-Agent→Facilitator）：

```
(ask-all
  :reply-with UA-20000518-3591
  :protocol sage_ask-all_1.0
  :language-encoding euc-jp
  :language KIF
  :ontology momochi
  :aspect (?カテゴリー ?商品名 ?商品価格)
  :content (and
             (商品リスト ?x)
             (カテゴリー ?x ?カテゴリー)
             (牛肉 ?カテゴリー)
             (商品名 ?x ?商品名)
             (取引先名 ?x ?取引先名)
             (= ?取引先名 "九州物産")
             (商品価格 ?x ?商品価格)
            ) ;; and
  :meta-info ((ordered-post-processes
                (sort-content ascending-numerical-order
                              ?商品価格)
              ))
  :sender User-Agent
  :receiver Facilitator
  )
```

上のような検索要求メッセージを受信したFacilitatorは登録されているオントロジ変換を必要に応じて:contentの値のKIF文に適用した後、そのKIF文を取り扱うことが可能なDB-Agentを、知識ベースに登録されている宣伝情報を元に、推論によるマッチングを行なうことでリストアップする。そのときの推論の一つとして、以下のように、直接対応するカテゴリが無くても、カテゴリツリーを上または下に辿ることで次善のDB-Agentを探す拡張カテゴリマッチングを行なう。

拡張カテゴリマッチングルール：

```
(<= (extended-handles ?dba (?class ?instance))
    (or (extended-handles-upward ?dba (?class ?instance))
        (extended-handles-downward ?dba (?class ?instance))))

(<= (extended-handles-upward ?dba (?class ?instance))
    (subclass-of-upward ?class ?superclass)
    (handles ?dba (?superclass ?instance)))

(<= (extended-handles-downward ?dba (?class ?instance))
    (subclass-of-downward ?subclass ?class)
    (handles ?dba (?subclass ?instance)))

(<= (extended-handles ?dba ?p) (handles ?dba ?p))
```

下に、オントロジ変換と拡張カテゴリマッチングによって、Facilitatorから複数のDB-Agentに回送された検索要求メッセージを示す。

検索要求メッセージ（Facilitator→DB-Agent-A）：

```
(ask-all
  :reply-with FA-20000518-9861
  :protocol sage_ask-all_1.0
  :language-encoding euc-jp
  :language KIF
  :ontology ebisu
  :aspect (?カテゴリー ?商品名 ?商品価格)
  :content (and
            (SITEMX ?x)
            (CTGRYDAIKANJI ?x ?カテゴリー)
            (= "牛肉" ?カテゴリー)
            (ITEMKANJI ?x ?商品名)
            (DISCORPKANJI ?x ?取引先名)
            (= ?取引先名 "北海物産")
            (PRICE ?x ?商品価格)
           ) ;; and
  :meta-info ((ordered-post-processes
               (sort-content ascending-numerical-order
                             ?商品価格)
              ))
  :sender Facilitator
  :receiver DB-Agent-A
 )
```

検索要求メッセージ（Facilitator→DB-Agent-B）：

```
(ask-all
  :reply-with FA-20000518-1358
  :protocol sage_ask-all_1.0
  :language-encoding euc-jp
  :language KIF
  :ontology momochi
  :aspect (?カテゴリー ?商品名 ?商品価格)
  :content (and
            (商品リスト ?x)
            (カテゴリー ?x ?カテゴリー)
            (or
              (= "牛" ?カテゴリー)
              (= "和牛" ?カテゴリー)
              (= "輸入牛" ?カテゴリー)
             ) ;; or
            (商品名 ?x ?商品名)
            (取引先名 ?x ?取引先名)
            (= ?取引先名 "北海物産")
            (商品価格 ?x ?商品価格)
           ) ;; and
  :meta-info ((ordered-post-processes
               (sort-content ascending-numerical-order
                             ?商品価格)
                ))
  :sender Facilitator
  :receiver DB-Agent-B
 )
```

上記のように（複数の）DB-Agent宛に検索要求のメッセージを回送したのち、FacilitatorはDB-Agentからのメッセージのタイムアウト付きの待ち合わせを行なう。このときの待ち合わせとしては、非同期の待ち合わせが望ましく、また、待ち合わせている回答メッセージ同士以外に、User-Agentからの新規検索要求メッセージや、以前の検索要求のキャンセル処理要求メッセージの並列処理が可能であることが望ましい。

回答メッセージ（DB-Agent-A→Facilitator）：

```
(reply
  :reply-with DBA-A-20000518-9765
  :in-reply-to FA-20000518-1358
  :protocol sage_ask-all_1.0
  :language-encoding euc-jp
  :language KIF
  :ontology ebisu
  :content (("牛肉" "牛ロース" 2000)
            ("牛肉" "牛もも肉" 1000))
  :sender DB-Agent-A
  :receiver Facilitator
 )
```

回答メッセージ（DB-Agent-B→Facilitator）：

```
(reply
  :reply-with DBA-B-20000518-6684
  :in-reply-to FA-20000518-1358
  :protocol sage_ask-all_1.0
  :language-encoding euc-jp
  :language KIF
  :ontology momochi
  :content (("和牛" "和牛バラ" 1500))
  :sender DB-Agent-B
  :receiver Facilitator
 )
```

各々のDB-Agentから上記のような回答メッセージをタイムアウト時間までに受信した場合には、:contentの値の（複数の）レコードを集めてまとめの処理を行ない、User-Agent宛の回答メッセージの:contentの値を生成する。この場合は、元の検索要求メッセージの:meta-infoに価格による並べ替えが指定がされていたので、後処理としてソート関数の適用が行なわれる。

回答メッセージ（Facilitator→User-Agent）：

```
(reply
  :reply-with FA-20000518-2648
  :in-reply-to UA-20000518-3591
  :protocol sage_ask-all_1.0
  :language-encoding euc-jp
  :language KIF
  :ontology momochi
  :content (("牛肉" "牛もも肉" 1000)
            ("和牛" "和牛バラ" 1500)
            ("牛肉" "牛ロース" 2000))
  :sender Facilitator
  :receiver User-Agent
 )
```

2.2.3 状態監視機能の説明

検索要求の回送の制御のために、FacilitatorからDB-Agent自体の状態の問合せを行ない、状態を監視する。

この状態問合せメッセージの処理を、一般の検索要求メッセージの仲介処理と比較すると、タイムアウト付き待ち合わせを行なう点は共通であるが、管理方針によっては、問合せのタイムアウト処理として、何らかの回答がDB-Agentから得られるまで再問合せをし続けることや、Facilitator内部のDB-Agent上階管理情報・知識ベース内の宣伝登録情報の更新処理が伴う場合が有る点が異なる。

３．LISPのメリットと課題

3.1 LISPのメリット

エージェント（Facilitator）の実装にLISPを用いた理由は以下の通りである。

・エージェント間でやり取りされるKQMLやKIFの規約に則ったメッセージがS式であり、かつ、その枠内で適用対象毎にメッセージの構造が再帰的構造を含めて自由に定義可能になっており、個々の具体的なメッセージのレベルでもメッセージ中の各項目の値の構造・大きさは一定ではないので、メッセージの解釈やメッセージ内容の保持・処理は簡単ではない。しかし、LISPのリーダの流用や豊富なリスト処理の機能を用いることで、構造が複雑で自由度の高いメッセージの扱いが容易なものとなる。

・データとプログラム（関数）を一体のものとして扱えるので、プログラム（関数）をデータとして保存しておいて、後で、そのデータをプログラムを実行（関数を評価）する方式の実装にすることで、オントロジ変換指定・回答メッセージ内容の後処理等の指定を柔軟で自由度の高いものとすることが可能である。

・メッセージ中で使用する用語の体系であるオントロジや、オントロジ間の相互変換の定義や、データベースエージェントの能力の宣伝情報も再帰的構造を含めて複雑かつ自由度の高い構造を持ち、それらを用いたオントロジ変換や質問内容に相応しいデータベースエージェントの選択のための処理も複雑である。それらの処理を柔軟かつ見通しよく処理するためには、推論エンジンを用いたルールベースの処理として扱うことが必要となる。そのためのProlog処理系（推論エンジン）を実装するには、LISPが適している。我々は、LISPで実装されたProlog処理系（推論エンジン）であるFrolic [5]に、エージェント（Facilitator）の実装に必要な機能（メッセージ並列処理のためのマルチプロセス化・オントロジ毎にリレーションの登録を行なうためのマルチモジュール化・類似処理の高速化のためのキャッシュ機能）の追加して使用している。

・上記の処理は、非常に頻繁なガベージコレクション処理を必要とするが、LISPは枯れた言語であり、安定した動作が期待できる。

・LISP上でのオブジェクト指向プログラミングシステムCLOS(Common Lisp ObJect System)を使用した実装を行なうことで、コードの共有・整理が容易なものとなる。具体的には、次節で説明を行なう。

3.2 エージェント実装上の問題点と解決方法

3.2.1 エージェント実装上の問題点

エージェント、特にFacilitatorのように複雑なメッセージのやり取りを行なうエージェントの実装を考えると以下の（実装に使用する言語に非依存の）課題がある。

受信したメッセージに対して、受信したメッセージの種類や内容によって異なる処理を行う必要がある。

送信したメッセージに対する回答メッセージの待ち合わせの処理の必要がある。特にFacilitatorの場合は、User-Agentからの検索要求メッセージに対して、複数のDB-Agent宛に検索要求メッセージの回送を行なって、それらに対する複数の回答メッセージを待ち合わせる処理が必要となる。また、応答性向上のために、User-Agentからの検索要求メッセージの並列処理、即ち、１つの検索要求メッセージへの回答処理が終わる前に、次の検索要求メッセージ等を受信するように処理することの要求を考慮する必要がある。

ここで、回答メッセージ待ちの処理に対して、詳しく考察すると、単に回答メッセージの受信を待つ処理だけではなく、タイムアウトやキャンセル要求に対してもそれらに合わせた処理が必要である。その処理は、設定や場合により、これまでに受信したDB-Agentからの回答メッセージの蓄積だけをまとめて、User-Agentへは成功したものとして回答メッセージを送信する場合や、検索処理が失敗したとする回答メッセージを送信する場合の両方が有り得る。また、受信メッセージがどの待ち合わせに対応するかを判定する処理も必要となる。その処理には、受信時には既にタイムアウトまたはキャンセルされてしまったために該当する待ち合わせの存在しない場合のことも考慮する必要がある。

さらに、実装に必要となる資源の使われ方から、回答メッセージ待ちの処理の特徴を考えると、最終的な回答を送信するまでに、受信メッセージの蓄積等の作業用の一時的記憶用領域が多種・大量に必要となり、かつ、タイムアウトを長時間にして運用する場合には、ほとんどの時間は何らかのイベント待ちで、何ら処理が行なわれないという状態となる。

ここで、受信メッセージの処理等の実装について生じ得る問題点について具体的に考察するために、エージェントを単一プロセスとして実装する、または、User-Agentからの受信メッセージ毎にプロセスまたはスレッドを起こして、そのプロセスまたはスレッドが受信メッセージに対する回答メッセージをUser-Agentに回答するまでの一貫した処理を行なう場合を想定すると以下のようになる。

プロセスまたはスレッドの生存時間の大半がイベント待ちであり、かつ、その間、大量の一時的記憶用領域を必要とするので、CPU時間・記憶容量等の資源の浪費が問題となる。また、システムが落ちたときの回復処理のために、それらの一時的記憶用領域の内容の一部の定期的バックアップを行なうコード明示的に記述する必要があり、また、処理状況の監視のために定期的にモニタするために、該当部分の一時的記憶用領域をモニタ出力させるコードを明示的に記述する必要がある。

また、プロセスまたはスレッドは無限ループを回して定期的に、受信メッセージの有無のチェックする必要が有り、受信メッセージが有ったときは、その種類と内容等によって異なる処理を行なう必要があり、かつ、受信メッセージの有無のチェックと同じタイミングでタイムアウトやキャンセルの処理要求がないかどうかのチェックも同時に行なう必要があり、それらが有ったときの処理は、それまでの状況等の組み合わせの各々に応じて処理を変えなければならない。したがって、エージェントの実装は複雑なものとなり、メッセージの処理などの変更や、他の似た情報処理システムのエージェントに流用するなどの作業は複雑で難しいものなる。

さらに、エージェントシステムは、エージェント間の

メッセージのやり取りによって、協調作業を行ない、その結果、一連の情報処理を実現するものであるが、他のエージェントは独立して動作しているので、エージェント内の処理と、エージェント間のメッセージのやり取りの処理の流れが分断されて、一連の処理の全体が理解し難いものとなる。

3.2.2 エージェント実装上の問題点の解決方法

注目しているエージェントの他エージェントとのメッセージのやり取りを、特定の目的を達成するための一連の流れ毎に整理・分類して、それをメッセージ送受信等の処理の過程に沿って状態遷移を行なう有限状態機械（Protocol）オブジェクトのクラスとしてまとめることでメッセージ処理の実装の部品化を行なう。

その際に、複数の状態に於ける処理で共通に必要である情報は、有限状態機械（Protocol）オブジェクトのスロットとして格納することで値の受け渡しを行なう。

メッセージ受信処理要求・タイムアウト処理要求・キャンセル処理要求の各々に対応した処理を行なうために、有限状態機械（Protocol）オブジェクトのメソッドを個別に作成する。また、メッセージ受信処理要求・タイムアウト処理要求・キャンセル処理要求に対して、それらの処理対象となる有限状態機械（Protocol）オブジェクトインスタンスを探し出すための対応付け機構を要求の種類毎に用意し、回答メッセージを必要とするメッセージを送出する際には、必要に応じて各対応付け機構への登録を行ない、状態遷移に伴い登録した対応付けが必要なくなった場合には削除を行なう。

具体例として、仲介機能を実現する有限状態機械（Protocol）オブジェクトの状態遷移図を、図３に示す。

特定の目的を達成するための一連の流れを構成する、エージェント内のメッセージ処理過程または、エージェント間のメッセージのやり取りの過程を、有限状態機械（Protocol）オブジェクトとすることで、特定の目的達成に関連したメッセージ処理等の実装を一塊のものとして扱うことができ、かつ、他の目的達成に関連したメッセージ処理等の実装と明確に区別して取り扱うことが可能となる。ここで、それらに共通部分が有るならば、それをスーパークラスとして定義し、各々の相違点を加えたものをサブクラスとして定義すれば良い。そのような整理・系統化を行なうことで、動作の制御パラメタ等の設定の変更によって、複数のメッセージ処理サービスで共通して用いることのできるメッセージ処理の実装の部品化が可能となる。また、サービス定義オブジェクトとして、受信メッセージに対してどのクラスの有限状態機械（Protocol）オブジェクトを用いて処理を開始するかを決めるメソッドや、有限状態機械（Protocol）オブジェクトの動作の制御パラメタの格納用スロット等を持つものを導入することで、メッセージ処理サービスの定義の部品化も可能となる。さらに、エージェント内だけでなくエージェント間のメッセージのやり取りに関しても、特定の目的を達成するための一連の流れ毎に整理・分類して、メッセージ送受信等の処理の過程に沿って状態遷移を行なう有限状態機械（Protocol）オブジェクトのクラスとしてまとめることで、複数エージェントが関わるメッセージ処理の実装の部品化を行なうことができる。

また、有限状態機械（Protocol）オブジェクトの状態遷移の前後で受け渡す必要のある情報は、有限状態機械（Protocol）オブジェクトのスロットとして格納されている、即ち、特定の目的達成のための一連の処理過程が、どの段階まで進んでおり、どのような状態であるかは、有限状態オブジェクトの適当なスロットを参照することで知ることができる。したがって、システムが落ちたときの回復処理のための定期的にバックアップや、処理状況の監視のためには、有限状態機械（Protocol）オブジェクト全体や、必要な任意のスロットの値を参照するだけでよいので、メッセージ処理のメインのコードに事前に手を入れておく必要が無い。また、資源管理上も、目的の達成の如何によらず、有限状態機械（Protocol）オブジェクトが任意の状態から終了状態に遷移した後にオブジェクト毎消滅することで、それまで確保してきた作業用の一時的記憶の消去・記憶領域の開放を自動的に行なうことが可能である。

並列処理、即ち、マルチプロセスまたはマルチスレッド化の観点から、有限状態機械（Protocol）オブジェクトを用いた実装を考察すると以下のようになる。有限状態機械（Protocol）オブジェクトは状態遷移の度に、次以降の状態に於ける処理を行なうのに必要な情報をスロットの値として持っているので、１つのプロセスで最低限１つずつの状態遷移の処理を行なえば、複数のプロセスで協調して１つの有限状態機械（Protocol）オブジェクトを動作させることが可能である。そのとき、受信メッセージ処理要求・タイムアウト処理要求・中止処理要求に対してどの有限状態機械（Protocol）オブジェクトのインスタンスを動作させるべきかは、その有限状態機械（Protocol）オブジェクトが状態遷移時のメッセージ送信等の際に自分自身を登録した各対応付け機構を参照して決定され、各処理要求に対応するメソッドが適用される。ここで、各処理要求の待ち合わせの処理を１つのプロセスで行なう必要が無くなったので、全処理要求の有無のチェックを同時にしなくてよくなり、各処理要求に対するコードを互いに独立にできるので、メッセージ処理の実装の部品化を容易にする要因となる。また、状態遷移の操作を完了したプロセスは、まったく別の有限状態機械（Protocol）オブジェクトの処理のために再利用することが可能であり、計算機の資源の節約となる。

さらに、CORBA等の分散オブジェクト技術と組み合わせることで、エージェント間のメッセージのやり取りの有限状態機械（Protocol）オブジェクト化の範囲を、ネットワーク上の異なる計算機上で動作している複数のエージェントで構成されたシステムや、異なる言語で作成された複数のエージェントで構成されたシステムにも拡張することについての技術的可能性が有る。

４．まとめ

エージェント間で交換されるメッセージの構造・大きさの自由度・変化の大きさへの対応や、仲介のための変換・マッチング等の複雑な処理のために必要となる推論エンジンの実現を容易にするために、Facilitatorの実装にはLISPを用いた。また、エージェント間のメッセージのやり取りの一連の流れに対応する有限状態機械（Protocol）オブジェクトの実装のために、CLOSを導入した。

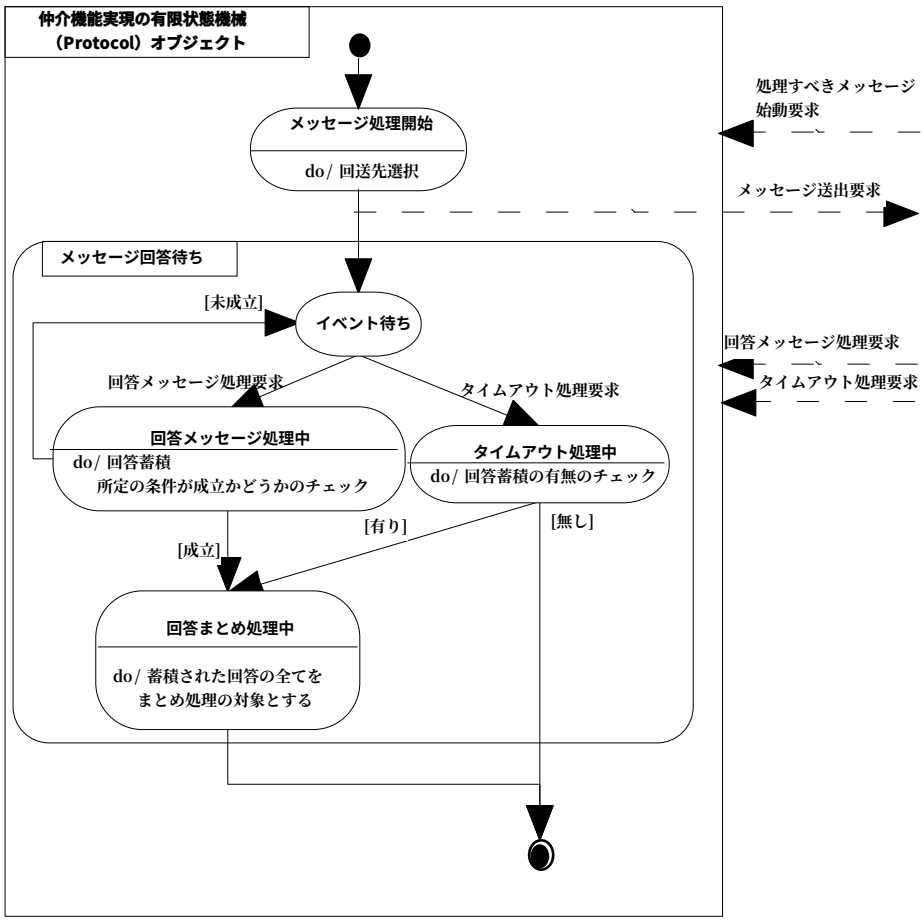

図 3．仲介機能を実現する有限状態機械（Protocol）オブジェクトの状態遷移図

謝辞

富士通研究所知能システム研究部及び、富士通株式会社アプリケーションサーバソフトウェア事業部第２開発部の皆様には、本研究の推進及び、その成果の製品化に対して多大なご協力を戴きました。

また、ユタ大学Dr. Jed KrohnfeldtとDr. Craig Steuryには、快くFrolicの自由な使用の許可を戴きました。

ここに記して感謝の意を表します。

参考文献(URL)

[1] INTERSTAGE, URL:http://www.fujitsu.co.jp/hypertext/softinfo/product/net/INTERSTAGE/

[2] M. R. Genesereth and S. P. Ketchpel: "Software Agents," Comm.ACM Vol.37 No.7, 1994.

[3] The DARPA Knowledge Sharing Initiative External Interfacees Working Group, "Specification of the KQML Agent Communication Language, "1994/2/9, URL: http://logic.stanford.edu/papers/kqml.ps

[4] M. R. Genesereth and R. E. Fikes, "Kowledge Interchange Format Version 3.0 Reference Manual," Technical Report Logic-92-1, Computer Science Department, Stanford Univ., 1992/6, URL:http//logic.stanford.edu/papers/kif.ps

[5] Frolic, URL:http://www.cs.cmu.edu/afs/cs/project/airepository/ai/lang/prolog/impl/prolog/frolic/0.html