ソフトウェア工学 112－8
（1996．11．15）

連邦アーキテクチャを用いた会議予約エージェントシステムの試作と評価

井上 健　根岸雅子　日野泰臣　秋本綾子
横河電機株式会社 計測制御アルゴリズム研究室

米国SRI社が開発した連邦アーキテクチャ型エージェントシステム開発支援環境である，OAA(Open Agent Architecture)を用いて会議予約システムを試作した．本稿では，システムの概要を紹介し，ソフトウェア工学の観点から，開発の流れと評価を行う．

Using Federation Architecture for constructing Meeting Scheduling System

Takeshi Inoue, Masako Negishi, Yasutaka Hino, Ayako Akimoto
Measurement Computing Laboratory, Yokogawa Electric Corporation

We constructed a Meeting Scheduling System using OAA(Open Agent Architecture), which, developed by SRI, supports constructing federation architecture based agent systems. In this report, we introduce our system and evaluate the development process and OAA from the views of software engineering.

1 はじめに

エージェントシステム構築環境として，米国SRI社がOAA(Open Agent Architecture) を開発している．これは，Facilitator と呼ばれる一種の黒板を介してエージェントどうしが通信するシステムを構築支援するものである．我々はOAAを導入し，最初のエージェントシステム開発として，会議予約システムを選んだ．ドメイン固有の知識を特に必要としないこと，エージェントらしさの作り込みが継続して行えることにより，評価や試作を続けられること，これを基盤に産業分野への応用を考察できることなどがその理由である．

本稿ではOAAと会議予約システムの紹介をし，機能や開発手法の観点から評価を行う．

2 OAAの概要

OAAは，米国SRI(Stanford Research Institute) で開発された，エージェントシステム構築支援環境である．これは簡単に述べると，分散ネットワークをまたがってお互い通信し，協調する独立プロセスを作るための支援を行う環境であり，実現は，いわゆる連邦アーキテクチャに基づいている [1, 2]．OAAを用いると，さまざまな言語で書かれたプログラムがさまざまなプラットフォームでFacilitator（協調促進器）と呼ばれる一種の黒板を介して行われ，これによりネットワーク上に分散したプログラムが協調して作業を行うことが可能になる．

エージェント開発を支援するために，そこにはいくつかの工夫がなされており，それらの工夫された機能をフルに利用して，より知的で自律的なシステムを開発できることがOAAの真の目的である．

OAAを使ったエージェントシステムの動作を一般的に述べると，各エージェントが動き始めるとき，まずFacilitatorに接続される．そのとき自分が実行可能な機能—これをソルバブルと呼ぶ，が何であり，そのソルバブルの名前やパラメータをICL(Interagent Communication Language) とよばれる形式で，Facilitator に登録する．このFacilitatorに接続されたエージェントは，Facilitatorに登録されているすべてのソルバブルを自由に利用することができるし，エージェントが依頼する作業をどのエージェントが実行するかは，依頼元は全く知らなくてよい．依頼がFacilitatorに届くと，その依頼をどのエージェントに送るかは，Facilitator自身が判断して行う．登録後，エージェントはFacilitatorからのイベント待ち状態にはいる．

図1には，PrologとCで書かれたエージェントがFacilitatorに登録される様子を示す．図中，接続と同時に，SolveP1, SolveP2, SolveC1,SolveC2など，エージェントが処理可能なソルバブルをFacilitatorに登録している．Facilitatorに接続されたエージェントは，普通はイベント待ち状態になり，イベントが来ると，図中に示されたdo_eventルーチンが起動されることになる．このため，各エージェントの各機能の実装はFacilitatorからのイベントを処理するdo_eventの中身をソルバブルごとに作り込むことになる．

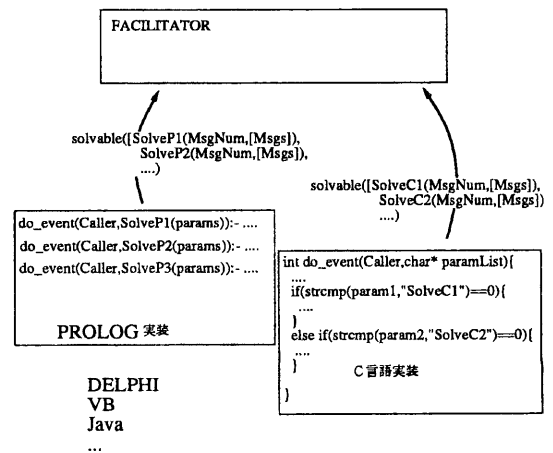

図1. Facilitatorに二つのエージェントを接続

OAAを使ったエージェントシステムの開発は

i. ICLによって，そのエージェントが実行可能なソルバブルを定義する．
ii. 各ソルバブルを，実装する．

という作業が基本である．Facilitatorに接続するルーチンや，メッセージを送るためのプログラムはすべてOAAのライブラリに用意されており，開発者は面倒なsocketプログラミングなどから解放される．また，実

装のための言語には，C,C++, Prolog, Visual Basic, Delphi, Java などを使うことができるし，TCP/IP をサポートするプラットフォームであればどこでもエージェントが実行可能である．

エージェントらしさの実装のために工夫されている機能としては,

- ダイナミックにエージェントを Facilitator に接続/切り離し可能にできること．
- Facilitator に対するリクエスト方式に同期，非同期，タイムアウト，ブロードキャストなど，単なる関数コール以上の協調方式を持たせたこと．
- エージェントは単にイベント待ち状態になるだけではなく，状況やデータの変化に対応して自動的に起動するしくみを持たせたこと．
- Facilitator に，エージェントがデータの書き込み/参照を自由に行える,「黒板」としての機能を持たせていること．
- Facilitator の階層化による，システム間交信が可能であること．

などが挙げられる．

また，開発環境としての OAA としては，ICL を定義して，プログラミング言語を指定すると自動的に言語のスケルトンを生成する ADT(Agent Development Tool)，複数エージェントの実行をマネージおよび動作をモニタリング可能にする実行マネジャなどが挙げられる．

3 OAA を用いて開発した会議予約システム

3.1 概要

OAA の評価および，エージェントシステム構築手法を確立するために,「会議予約システム」の試作を行った．これは，エージェントごとに役割担当を与え，ユーザからの要求を複数のエージェントが処理をしながら会議予約を行うシステムである．主な機能は

- 日付，時間，場所，参加者を指定すると，スケジュールの空き領域を見つけ出し，候補を表示したり，予約を行う．
- 指定する項目の内容は，曖昧な表現を使うことができる．
- 会議室だけでなく，人のスケジュールも予約する．
- 開催通知を行う
- 予約のキャンセルを行う
- 簡単な自然言語（英語）を使った予約が可能である．

などである．

入力となるアイテムの指定には曖昧表現が許される．数例あげると，日付に対しては「来週後半」,「毎週水曜日」，時間に対しては「午後 2 時頃から 2 時間くらい」,「一日のうちどこでもいいから 2 時間」，場所に関しては,「OHP が設置されている会議室」「15 人以上収容可能なところ」，参加者に対しては「○○チームの人」などである．

3.2 構成

我々が試作段階で用意したエージェントは，図 2 のものである．尚，この図ではエージェント間で直接通信しているように描いてあるが，実際にはすべてのエージェント間通信は Facilitator 経由で行っている．

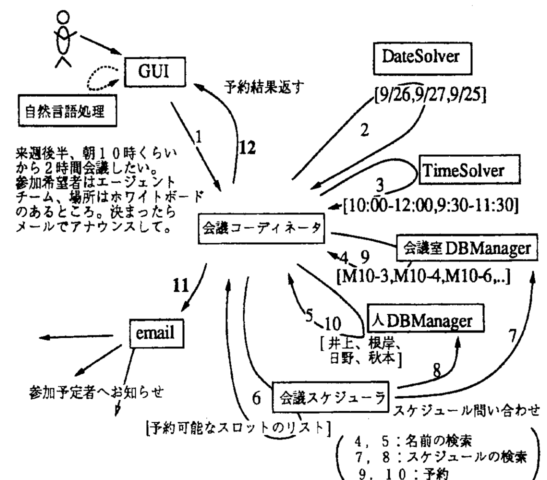

図 2. 会議予約エージェントと，処理の流れ

GUIから会議を予約する過程を図2に合わせてエージェント通信の流れから追ってみると，

i. ユーザがGUIを使って，希望の日付，時間帯，場所，参加者，を指定する.（図中の1）（このとき，予約後メールで参加者に通知したり，予約がとれない場合の処理を指定可能）．尚，今回は，自然言語処理エージェントとして，簡単な英語を解釈するエージェントをDCGを使ったProlog表現で開発している．これを使う場合はGUI上に英語のテキスト形式で入力すると，自然言語エージェントが予約形式フォーマットに変換してGUIに返し，それをGUIエージェントが直接使う．
ii. コーディネータエージェントが，場所/日付/時間/参加者のさまざまな表現をシステム内部で共通に使える形式に変換するため，会議室DBマネージャ，日付ソルバ，時間ソルバ，人情報DBマネージャという各エージェントにそれぞれの情報を変換してもらう.（図中の2，3，4，5）
iii. 内部形式になった情報をスケジューラに渡し（6），スケジューラエージェントは，指定日付のスケジュールを会議室DBマネージャ，人情報DBマネージャに聞きながら（7，8）空きスロットのリストを得る
iv. 空きスロットからコーディネータが選んで，会議室および人のスケジュールの予約を行う.（9，10）
v. ユーザが希望するなら，参加予定者に，会議の通知を行い（11），コーディネートの結果をGUIに返す（12）．

構成エージェントの実装言語は，GUIがDelphi，自然言語処理，コーディネータ，スケジューラ，会議室DBマネージャ，人情報DBマネージャがProlog，日付ソルバ，時間ソルバがCである．またメールエージェントはOAA環境が予め用意しているもの（Prolog記述）を使用した．

4 開発の流れ，OAAの評価

4.1 開発手法

我々の開発の流れを紹介しながら，エージェントシステム開発手法に関する評価/考察を行う．

我々にとって今回はエージェントシステムの試作であり，OAAの習得および評価を目標にしていたために，綿密な開発スケジュールやプロジェクト管理は行わなかった．しかし開発メンバはオブジェクト指向の分析・設計手法，特にOMT法[5]の経験があり，それが開発に役立った．エージェント指向とオブジェクト指向の違いをふまえて，方法論などを考察するのも興味深いところである．

今回の開発は，

- エージェントをどう作ればよいのか
- システムにどんな「エージェントらしさ」を組み込めるか，それはどのように行うのか
- 拡張性や保守性をどう高めるか
- OAA自身が上の3つの要求に耐えられるか，足りない点，問題点は何か

ということを念頭に行っており，システムをどう作れば一番よいものができるか，という一般的なシステムの開発とは趣がずれている部分もあるが，開発の中で，OAAのアーキテクチャに基づくエージェント構築手法を考察してきた．

開発作業は

i. シナリオ作り
ii. 機能分けと，役割分担によるエージェントの洗い出し
iii. エージェントのインタフェース（ICL）決定
iv. 実装
v. 単体テスト
vi. 結合テスト

という流れであり，各フェーズ間での手戻りはあったがおおよそウォーターフォールモデルに基づくオーソドックスな開発の流れであった．

[1] シナリオ作り，エージェントの洗いだし

シナリオ作りは，オブジェクト指向分析でも使われているユースケースを用いて行った．システムに対してユーザが会議予約を行う場合を大きく

- 予約可能な会議のスロットを検索する

● 会議の予約を行う

の二つに分けて考えた．図3にユースケース図を示す．

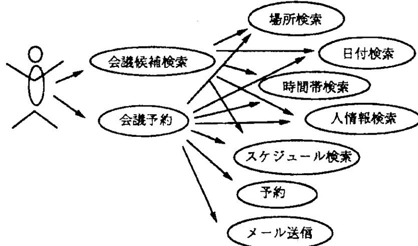

図3. 会議予約のユースケース

図からもわかるとおり，与えられた日付と時間帯に対する会議室と参加者のスケジュールを検索するところはいずれのユースケースでも共通な部分である．会議スロット検索では予約可能な部分をユーザが指定する候補数だけ表示する（そこからユーザは予約できる），会議予約の場合は，予約可能な会議スロットから一つを選んで予約を行う．

ユースケースからエージェントの洗い出しの部分が，開発の作業，以降のフェーズに対して最も重要なものであった．エージェントの単位の定義は未だに定説となっているものはないが，ここでは，人間が役割分担してシナリオを実行すると想定したときに最も自然と思われる役割単位を決め，それにエージェントを割り振っていった．オブジェクト指向の世界における，オブジェクトよりも当然粒度の大きな単位となっている．

図4に初期のエージェント構成図を示す．今回の開発は約4ヶ月という短期集中開発で，エージェント構築の評価を行うことが目的であったが，最初のエージェント構成図では機能の絞り込みは行わず，会議予約システムが持つべきエージェントたちを洗い出した．

ユーザインタフェースに関しては，GUIだけでなく，WWWを使った予約，電子メールを使った出張先などからの予約，電話などによる音声での予約などを考えた．このため，音声認識や自然言語処理用エージェントが必要になった．次にユーザインタフェースエージェントからの指定項目を一貫して受け付ける，コントローラエージェントを考えた．また，予約が競合したときなどの処理を知的に解決する競合処理エージェントも必要であった．

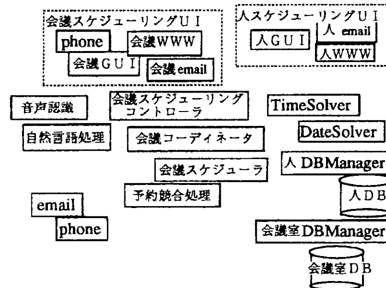

図4. 開発初期に考えたエージェント構成

最終的なエージェント構成図はさきに図2に示したとおりである．図4の時から抜けているものの多くは，今回時間と優先度から開発できなかったものも多いが，UIからの入力を一括して受け取るコントローラは，コーディネータの機能として作り込むことにしたため省略した．

エージェントの洗い出し作業は，4人のメンバで議論を行い，変更は設計にはいってからも行われた．洗い出しの際に問題となり議論の中心となったのは，「エージェントの守備範囲」である．一つのエージェントが何を知っていてどこまで担当するのかということは，初期の段階では詳細までは決められなかった．図2のコーディネータが受け取れるメッセージのフォーマットを現在は各UIエージェントが知っているが，将来さまざまなUIエージェントが開発されると，コーディネータのことは知らずにFacilitatorにメッセージを投げてみるような実装が出てくるかもしれない．こうなると入力を解釈する，図4のコントローラの役割をはたすエージェントが必要になるであろう．

シナリオからのエージェント洗い出しでもう一つ問題になったのは，伝統的な開発手法と協調エージェント開発の関連である．OAAの環境を使うと「誰が処理してもよいから，この問題を解いてほしい」という要求をFacilitatorに投げる方式が実現され，一つのエージェントは，Facilitatorに接続されたすべてのエージェントと自由にメッセージ交信可能である．かたやロバストなシステム作りとして構造化プログラミングやオブジェクト指向などで推奨されているのは，システムの管理，拡張，再利用性のために，関数コールやメッセージの流れを明確かつ整理された形でまとめること

である.(図 5 参照)

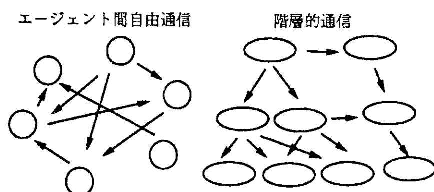

図 5. 自由な通信と階層化された通信

今回の我々のシステムをシナリオの流れ通りに動くシステムとして開発してしまうと，これまでの手法で関数コールの流れとして実現できてしまう. OAA を使ったエージェントのみで実現できる，ダイナミックな処理が実現できない. 具体的には，スケジュールがわからないときは UI からコーディネータに頼んで調整をしてもらいたいが，そうでない時やとにかく会議室の予約だけしたいときには，コーディネータを介することなく，直接予約したい. この要求を UI エージェントがいちいち判断しなくても,「処理できるエージェントが自由に処理して返す」方式がほしい. これが可能であれば,「知的協調エージェント」という次のステップが見えてくる. しかしながら，各エージェントが勝手に処理を用意して実装し，Facilitator に接続していってしまうと，システムとして収集がつかなくなるし，畢竟，保守性や拡張性も悪くなる.

我々の結論は，基本的な流れはシナリオ通りに開発し，add_on 的にしかも整合性に注意を払ってソルバブルを加えていくことである. 多数のエージェントが Facilitator を介して協調していくシステムといっても，全体の流れの中では既存システム的な要素は多い. その部分をしっかり伝統的手法で開発し，エージェントらしさ，知的協調部分などをエージェント独特の手法で埋め込んでいく方法が効率的かつ確実な方法であると思う.

[2] ICL の決定から実装

各エージェントが，どういうインタフェースを他のエージェントに提供するかを決定していった. 機能としてどういうものを提供するかはエージェント洗いだしの段階で明らかであったため，メッセージ名は簡単に決められたが，必要な引数やその形式については実装にはいるまで変更が続いた.

特に曖昧表現を受け付けるため，来月前半，午前中，毎週水曜，などという表現をフォーマットとして決定するところに時間がかかったし，実装にはいってからの細かい点の変更もあった.

実装言語は，GUI に関しては開発のしやすさの点から，Delphi を選んだ. 他のエージェントは開発者自身が慣れている言語ということで，C と Prolog が使われた. 特に機能の点から言語に対する要求はなかったが，OAA 自体が Prolog で書かれており，OAA 開発初期においては，Prolog でのエージェント開発を想定していたために，ライブラリの使いやすさなどから Prolog が便利であった.

実装時に注意する必要があったことは，入力に対するチェックと例外処理をしっかり行うことである. 前節でも述べた通り，エージェント間の自由な通信が可能であるためには，いつ誰からどんな要求が来ても，正しく対処できる信頼性が必要とされる.「この処理は，このエージェントから依頼が来るはず」などという仮定のもとにエラーチェックを省略することは禁物である.

[3] テスト

一つの単体テストであっても，他のエージェント機能を利用しながらテストすることが必要である. 初期段階ではスタブに相当する機能を単体の中に作り込みテストしたが，他のエージェントの開発が進行すると共に，それを使いながらテストを行った. その際，StartIt と呼ばれるエージェント監視プログラムや，インタフェースエージェントが有用であった.

自分が開発したエージェントのみデバッガを通し，他のエージェントと共に走らせながらデバッグ作業ができるところは効率よいテストを可能にしている.

[4] 開発形態について

少人数とはいえ，ICL 定義は開発と同時進行で変更が頻繁に行われたため，ICL 定義は HTML で記述して，他人が開発したエージェントを使うときは Web 上で定義を参照しながら行った. これによりペーパーレスは勿論，最新の情報を効率よく参照することが可能になった. 但し，ICL 定義変更と同時に HTML ファイルに変更反映することはグループ内で徹底した.

4.2 OAAの評価

OAAの機能，開発環境，開発手法に関して評価を行う．

[1] 機能

ICLというインタフェースをエージェント間で共有することによって，効率よくエージェント開発が行えることが確認できた．

いくつか注目すべき利点を挙げる．

- 言語やプラットフォームを選ばない
  言語や動作環境を選ばずにシステム構築できることは，以下の利点がある．
  - 運用環境に沿った開発が可能
  - 特定の場所，マシン上，言語でしか動かないライブラリがあっても，エージェント社会に組み込める
- ダイナミックなエージェント接続
  システム全体を止めたり再コンパイルすることなく，一部のエージェントを改良することができた．これによりユーザの見えないところで機能のアップグレードなども可能である．
- トリガ機能
  何かを監視するようなエージェントに対しては「あることが起きたら」または「この条件が満たされたら」処理を開始するという指定が可能であり，エージェントの自律性を実現しやすい．
- Facilitatorの黒板としての役割
  黒板を用意して，エージェント間で共通に使える知識を書き込める所も効率的な通信には有用であった．知識を蓄積することにより，エージェントの学習機能も作りやすくなると思われる．書き込んだエージェントがFacilitatorから切り離された時点でその知識も消滅するところも一貫性がある．
- 既存のアプリケーションのエージェント化
  既存アプリケーションのエージェント化は，今回の我々の実装では経験しなかったが，プログラミング言語とのAPIを持つアプリケーションであれば，エージェント化が可能であることはOAAの強力さの一つである．各種データベースや，音声インタフェースなど，必要なものは他から導入し，エージェント化作業を行うだけで自分達のエージェント社会に既存システムを組み込むことが可能である．
  特にヒューマンインタフェースに関しては，今回は簡単な自然言語解釈エージェントをPrologで用意しただけであるが，文章の内容を理解してICL形式に変換し，Facilitatorに投げるだけで処理可能であった．市販のさまざまな自然言語解釈，音声認識ソフトなどとの組合せも容易であると考えられ，優れたヒューマンインタフェースを開発すれば，既存のアプリケーションに付加して使用することも可能になる．

一方，問題点もいくつか指摘したい．複数プロセスの扱いという，もともと複雑な系の管理であるから本質的に難しいこともあるが，改良の余地は充分あると考える．

- エラー処理の作り込みがしにくい．
  もともとPrologを使った開発を想定しているため，ソルバブルの実行が失敗したときに，true/falseのいずれかを返す．falseの場合はPrologの述語が失敗するだけのため，引数にエラーメッセージを入れて返すことができない．我々は，ソルバブルが起動された場合は常にtrueを返し，例外処理はエラーメッセージに代入して返す方式で統一したが，これではせっかくのPrologのバックトラック機能などが有効に活用できない．
- 同種エージェントが複数の場合の処理
  同種のエージェントが一つのFacilitatorに対して複数動作する場合の状態管理は，エージェント開発者側に任されている．Facilitatorは基本的に，一つのソルバブルを処理可能なすべてのエージェントに送るため，同じメッセージを複数のエージェントが受け取る．エンドユーザにとっては柔軟なシステムを作れるが，プログラミングの立場からは注意が必要である．
- パフォーマンス
  Facilitatorを介したTCP/IP通信によって，すべてのエージェント通信が行われる．我々のシステムでは問題にならなかったが，通信が集中する場合のパフォーマンスは問題になりそうである．

[2] 開発環境

エージェントの起動/停止および，モニタリングのためのシェルを起動する StartIt というプログラムが用意されている．これにより，エージェントたちを自由に起動/停止させ，動作状態を監視することができる．また，OAA が予め用意しているエージェントの一つに，(OAA エージェントとの) インタフェースエージェントがあり，このエージェントの GUI を使うことにより，ICL メッセージを直接エージェントに送り，送られたエージェントからの回答も表示することができる．これは，エージェントのテストに非常に有効であった．

OAA はまた，ADT(Agent Developer's Tool)[6] という開発支援ツールを用意している．これを使うと ICL 定義を，GUI を介して行うことができ，その後希望プログラミング言語のスケルトンを自動生成する．ICL の定義だけではなく，他人の作るシステムの ICL 定義を参照することにも役立つ．しかし現段階で PC のみに対応していることや，我々は WWW 環境でドキュメント管理が充分にできていたため，ADT の多用はしなかった．

今後，より分散した環境でさまざまなエージェントが協調動作していくことを考えると，個々のエージェントをグラフィック表現し，メッセージ送信や処理の様子を動画で表せるようなツールができると，より開発環境として使いやすいものになるだろう．

5 まとめ

OAA を使用すると，個々のエージェント構築は開発者自身の考え方，作り方に任されており，自由に好きな言語で開発できるが，そのための環境としての OAA は便利な機能が用意されている．通信自体のプログラミングから解放されて，実質的なエージェント作りの方に注力できることは高く評価したい．

我々自身まだ OAA に携わって間もなく，時間の制約の中で，今回は単純な機能をもつエージェントシステムになったが，今後これを土台にエージェント機能を拡張，充実させていく．特に，学習や知的協調計算に関しては今後注力していく予定である．

OAA を使うといろいろ便利なものを開発できるが，品質管理や拡張性を考えると，ヒューリスティックな方法では信頼性の高いシステム構築は難しい．今後大きなシステムを開発していく上で，エージェントアーキテクチャの発展とともに，しっかりした開発手法が不可欠になると考えられるため，今回述べた問題点などをふまえ，方法論的なものを作り上げていく予定である．

参考文献

[1] 木下哲男，菅原研次「エージェント指向コンピューティング」，SRC(1995)

[2] 西田豊明 「ソフトウェアエージェント」 人工知能学会誌，Vol 10 No.5. pp.44-51. Sept. 1995.

[3] P.R.Cohen, A.J.Cheyer, M.Wang, and S.C.Baeg, "An open agent architecture," in AAAI Spring Symposium, pp. 1-8, March 1994.

[4] D.B.Moran and A.J.Cheyer, "Intelligent agent-based user interfaces," Proceedings of International Workshop on Human Interface Technology 95 (IWHIT'95), pp. 7-10, Oct 1995

[5] J.Rumbaugh, J. et al.:*Object-Oriented Modeling and Design*,Prentice Hall, 1990.(羽生田監訳「オブジェクト指向方法論 OMT」, トッパン,1992)

[6] D.L.Martin, A.Cheyer, and G.L.Lee, "Agent development tools for the open agent architecture,", Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, pp. 387-404, April 1996.