■7 群（コンピュータ-ソフトウェア）－ 7 編（分散協調とエージェント）

5 章 エージェントの設計・開発・処理系

【本章の構成】

本章では以下について解説する.

5-1 エージェント開発実行環境概説
5-2 エージェント開発支援環境・フレームワーク
5-3 モバイルエージェント実行環境
5-4 マルチエージェント実行環境
5-5 エージェント実行環境の相互運用性
5-6 エージェント設計手法・方法論

■7 群－7 編－5 章

5-1 エージェント開発実行環境概説

（執筆者：田原康之）［2010 年 11 月 受領］

エージェントは高度なオブジェクトである，という観点を考慮すると，エージェントアプリケーションの設計と開発の支援環境，及び処理系としての実行環境は，オブジェクト指向アプリケーションに対するものをベースとし，そのうえでエージェント固有の機能を実現する，という方向に沿ったものがほとんどであると考えられる．本節では，このような観点からエージェント開発実行環境を分類して説明する．

まず開発支援環境としては，テキストベース，あるいはよりビジュアルなユーザインタフェースにより，システムモデル，プログラムコード，及び関連するデータの記述編集機能，ならびにそれらを含んだファイル群全体を管理するための構成管理機能などを有する，統合開発環境（Integrated Development Environment：IDE）が主に研究，構築されている．このような環境では，エージェントに固有な機能で利用するデータとして，推論や学習に利用する知識表現，エージェント間協調に利用するためにエージェント間通信言語で記述されるメッセージ，及びモバイルエージェントの移動先情報などを扱うことが必要である．したがって，それぞれ独自の手法により，これらのデータを扱う様々な開発環境が提案されている．

実行環境についても，広く利用されているもののほとんどは Java 言語で記述されており，Java のオブジェクト指向言語としての特徴を活かしたものとなっている．すなわち，エージェントの基本機能を実現したライブラリー，コンポーネント，あるいはミドルウェアを，再利用や変更が容易とするためのフレームワークとして構築している．エージェントの基本機能としては，例えば次のようなものが提供されている．

- モバイルエージェントのための，エージェントの可搬データ化機能，移動のためのネットワーク通信機能，及び移動先の場所（ネットワークノードや論理的に定義された場所情報など）を扱う機能．
- エージェントの知的処理のための，推論機能，学習機能，及び知識管理機能．
- エージェント間協調のための，ネットワーク通信機能，メッセージ作成・処理機能，及び協調プロトコル実行機能．

また，以上の機能を容易に利用するための，専用の記法や記述言語を用意していることも多い．

このように，実行環境に盛り込まれる機能は多岐にわたり，必要な要素技術も様々であるため，これまでに構築されているものは，API，及び専用の記法や記述言語などについて，様々に異なっている．そのため，1 種類の実行環境だけでは実現できないような，複雑なアプリケーションの開発のために，複数の異なる実行環境を利用したい場合でも，それぞれの環境上で構築された部分システムを統合することは困難である．そこで，エージェント実行環境の相互運用性を高める手法の研究も行われており，そのための標準化を図っている団体もある．代表的な標準化活動として，OMG（Object Management Group）による MASIF（Mobile Agent System Interoperability Facility）[1]，及び FIPA（Foundation for Intelligent Physical Agents, http://www.fipa.org/）がある．

以上で取り上げた開発支援環境，及び実行環境は，開発プロセスにおいてコーディングを中

心とした下流工程に関するものである．しかし，エージェントの実用化を進めるうえで，大規模かつ複雑なアプリケーションの開発を容易にするためには，ソフトウェア工学の観点から，上流工程から保守運用に至る，開発プロセス全体の支援が必要となる．そこで，エージェントアプリケーションの設計手法，また更に開発プロセスを広範囲にカバーする開発方法論の研究が行われている．これらの研究においても，前述のようにオブジェクト指向をベースとすることが多い．すなわち，オブジェクト指向開発における標準的なモデリング言語である UML（Unified Modeling Language），及びそれに類する記法を利用したうえで，エージェント固有の概念を，やはりオブジェクト指向方法論の概念をベースとして設計する手法を盛り込んでいる．エージェント固有のものとして，例えば以下のような概念や手法がある．

・モバイルエージェントの移動をモデル化するための記法やデザインパターン
・エージェントの推論や学習に用いる知識の抽出とモデル化：ゴール指向分析やオントロジー
・エージェント間協調プロトコルのモデル化：シーケンス図や状態遷移モデル
・エージェントアーキテクチャのモデル化：BDI（Belief, Desire, and Intention）

そして，これら様々な概念や手法をモデル化するための，UML の拡張である AUML（Agent UML，http://www.auml.org/）が FIPA から提案され，その一部が随時 UML の改訂に反映されるに至っている．

■参考文献

1) Object Management Group：“Mobile Agent System Interoperability Facility Specification,” http://www.omg.org/cgi-bin/doc?orbos/97-10-05, 1997.

■7 群－7 編－5 章

5-2 エージェント開発支援環境・フレームワーク

（執筆者：菅原研次）［2011 年 2 月 受領］

ソフトウェア開発におけるフレームワークとは，抽象的で典型的な設計仕様やプログラムコードを用意し，フレームワーク利用者がそれらを選択的に上書き，修正，拡張をしながらシステムを開発することを支援する一般的な枠組みのことを言う．フレームワークの利用者は，ライブラリーの利用のように明確に定義されている API（Application Program Interface）を持つコードを再利用するだけでなく，システムのアーキテクチャを含めた設計ノウハウを再利用できる環境が提供されるため，よりソフトウェアの開発効率が向上すると言われている．

エージェントシステムの開発を支援するためのエージェントフレームワークは，この特性に加えて，自律性の高いエージェントを部品とすることにより，ネットワーク環境に分散した部品間の協調が期待できる点に大きな特長がある．部品を要求する利用者あるいはほかの部品が，要求する部品の名前，場所，あるいは，部品の個数を意識することなく，部品の機能を利用できる開発運用環境が提供される．これらの環境を実現するために，一般にファシリテータあるいはブローカと呼ばれる機能が環境に存在し，部品の利用・再利用を促進している．

エージェントフレームワークは，研究を目的としたものから，実用性を指向したものまで多くのフレームワークと開発環境が提供されている．

OAA（Open Agent Architecture）は SRI International から提供されている実用的なエージェントシステムを開発するためのエージェントフレームワークである [1]．OAA は，(1)フレキシブルなシステムを構成するための汎用の開発環境とランタイムシステムを提供する，(2)人間指向のユーザインタフェースを提供する，(3)既存の様々なプログラミング言語で書かれたプログラムをラップ(包み込む)することにより，実用的で効率的なシステム開発環境を提供する，ことを主要な目標としている．OAA を利用することにより，画像処理，音声認識，協働作業支援システム，テキスト理解，プランニング，仮想空間などのアプリケーションが開発されている．

AgentBuilder は，エージェント型アプリケーションを開発するためのツールキットとランタイムシステムから構成される．AgentBuilder は，Java 仮想マシン上で動作するので，多くのコンピュータシステムで利用できる．オブジェクト指向部品再利用環境の CORBA との互換性を持ちエージェント通信言語の KQML との互換性がある．AgentBuilder を利用したメールサービス，オークションサイト，エネルギー資源の制御・管理システムなど多くのアプリケーションが開発されている．

次世代のエージェントシステム開発を支援するための研究指向のエージェントフレームワークとしては，現在も多くの研究が行われている．

ADIPS フレームワークは，リポジトリを中心とした情報や部品の再利用を支援するエージェントフレームワークである [2]．リポジトリとは，**図 2･1** で示すように，エージェント部品のクラスを格納する部品庫であり，エージェント部品は，自分の機能，エージェント通信，相互利用関係に関する仕様を知識として保持し，拡張された契約ネットプロトコルを用いて，リポジトリに対して要求された機能を持つ部品の組織（サブシステム）を動的に構成し，ランタイムシステムに実行部品（サブシステム）を生成するサーバ機能である．既存のデータやプログラ

ムは，インスタンスエージェントにラッピングされ，エージェントの機能として再利用される．開発環境 IDEA とランタイムシステム DASH を利用して，分散知識ベース，やわらかいネットワークシステムやユビキタス型のケアシステムの実験が行われている．

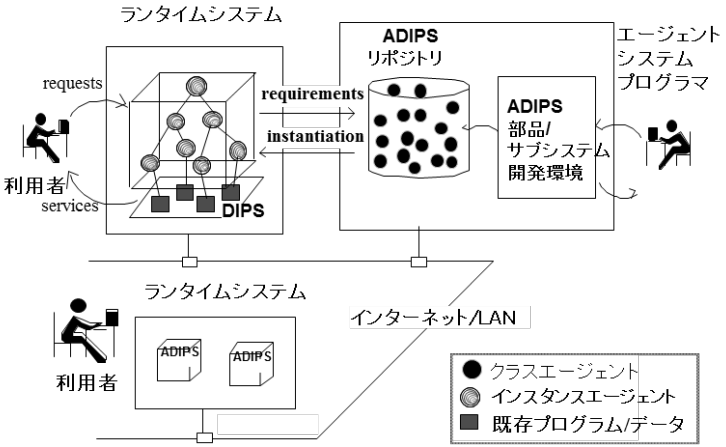


図 2・1　リポジトリ型フレームワーク（ADIPS）

更に，FIPA のフレームワークが 5-4 節の項目，モバイルエージェントのフレームワークが 5-3 節の項目で述べられている．

■参考文献

1) A. Cheyer and D. Marti：“The Open Agent Architecture,”AUTONOMOUS AGENTS AND MULTI-AGENT SYSTEMS, vol.4, No.1-2, pp.143-148, 2001.
2) T. Kinoshita and K. Sugawara：“ADIPS Framework for Flexible Distributed Systems,”Multiagent Platforms, Lecture Notes in Artificial Intelligence 1599, pp.18-32, Springer, 1998.

■7 群－7 編－5 章

5-3 モバイルエージェント実行環境

■7 群－7 編－5 章

5-4 マルチエージェント実行環境

（執筆者：田原康之）［2010 年 11 月 受領］

マルチエージェントシステムの研究が開始された当初，研究者は既存のプログラミング言語を用いて個別にアプリケーションを実装していた．しかし，1990 年代の初頭に，ショーハム（Shoham）が「エージェント指向プログラミング」の概念[1]，及び専用のプログラミング言語 AGENT0 とその処理系に関する論文 2）を発表して以来，マルチエージェントシステムのための実行環境の提案が，プログラミング言語の処理系を中心に行われるようになった．その後，Java などのオブジェクト指向プログラミング言語の普及に伴い，オブジェクト指向の特徴を活かした実行環境も多数登場するようになった．

現在，マルチエージェントシステムの実行環境は，それぞれ汎用，及び専用のプログラミング言語を用いるものとの 2 つに大別される．前者については，Java を用いたフレームワークがほとんどである．後者については，マルチエージェントシステムの機能を記述するのに有効と考えられる特徴を持った，様々な言語が提案されている．いずれにしても，マルチエージェントシステムの基本機能，すなわちエージェントの知的かつ自律的な振る舞い，及びエージェント間協調を容易に実現することを特徴としている．また，Java 実装のものについては，Java のオブジェクト指向言語としての特徴を活かし，フレームワークとして実装されたプログラミングの枠組み，あるいはカスタマイズ機能を有している．

Java を用いたフレームワークの代表的なものとして，まず FIPA（本章 5-1 節参照）仕様に準拠した，FIPA-OS（http://fipa-os.sourceforge.net/index.htm），及び JADE（Java Agent Development Framework, http://jade.tilab.com/）が挙げられる．両者に共通する特徴は以下の通りである．

（1）オープンソースライセンス
（2）諸機能をコンポーネントとして実装
（3）Java で記述された，広く使われている推論エンジン JESS（http://www.jessrules.com/）との統合が可能
（4）各種開発支援環境の提供．

FIPA-OS に固有の特徴は以下の通りである．

（1）FIPA 標準の参照実装として登場
（2）Conversation クラスにより通信プロトコルをカプセル化
（3）商用のデータベースやメッセージングシステムとの統合が可能

Jade に固有の特徴は以下の通りである．

（1）Behaviour クラスにより，通信も含めたエージェントの振る舞いをカプセル化
（2）携帯機器の組込みプログラムなど，計算資源が乏しい環境で動作させるためのライブラリーLEAP（Lightweight Extensible Agent Platform）の提供

Java を用いたその他のフレームワークは，商用・非商用，有償・無償，またオープンソースのものなどを含め多数存在するが，そのなかから Bee-gent（http://www.toshiba.co.jp/rdc/beegent/）を取り上げる．Bee-gent の特徴は以下の通りである．

（1）一部 FIPA 仕様に準拠
（2）非エージェントシステムをエージェントとして扱えるようにするためのエージェント

ラッパー（Agent Wrapper）と，ネットワーク上移動機能を有し，エージェントラッパー間の協調を実現する仲介エージェントとの 2 種類のエージェントから構成

（3）開発支援環境 IPEditor の提供

次に，専用のプログラミング言語を用いる実行環境も多数にのぼる．それらの多くは，エージェントの推論機能の実装を用意にするために，論理型言語を用いている．そのなかから，Jason（http://jason.sourceforge.net/Jason/Jason.html）と 3APL（http://www.cs.uu.nl/3apl/）を取り上げる．両者とも，BDI アーキテクチャに基づいた論理型言語を用いている，という共通の特徴がある．

Jason は，以前 AgentSpeak と呼ばれていたプログラミング言語の拡張に対する，Java で実装された処理系である．その特徴は以下の通りである．

（1）エージェント間協調は，環境（Environment）に対する動作を通じて実現

（2）Java 言語の API を利用したカスタマイズが可能

（3）ソースコード編集ツール jEdit とデバッガ mind inspector の提供

3APL の特徴は以下の通りである．

（1）実行環境として，Java 版のほかに，関数型プログラミング言語 Haskell で実装したものも提供

（2）FIPA 仕様準拠のメッセージ通信をサポート

（3）プラグイン作成によるカスタマイズが可能

（4）実行中にエージェントの状態を表示する機能や，メッセージ通信状況を時系列で視覚的に表示する機能を備えた，ユーザインタフェースの提供

以上で紹介したもののほかにも，それぞれ独自の特徴を有する各種の実行環境が提案されているが，主要なものをまとめたものとしては，ボルディーニ（Bordini）らによる 2005 年の文献 3）と 2009 年の文献 4）がある．

■参考文献

1) Y. Shoham：“Agent-oriented programming,” Artif. Intell., vol,60, no.1, pp.51-92, 1993.
2) Y. Shoham：“AGENT0: A Simple Agent Language and Its Interpreter,” in Proc. of the Ninth National Conference on Artificial Intelligence, eds. T.L. Dean and K. McKeown, pp.704-709, AAAI Press, Menlo Park, 1991).
3) R.H. Bordini, M. Dastani, and A. El Fallah Seghrouchni：“Multi-Agent Programming: Languages, Platforms and Applications,” Springer, Berlin, 2005.
4) R.H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni：“Multi-Agent Programming: Languages, Tools and Applications,” Springer, Berlin, 2009.

■7 群－7 編－5 章

5-5 エージェント実行環境の相互運用性

（執筆者：桑原和宏）［2011 年 3 月 受領］

異なる設計者によって設計されたエージェントから成るマルチエージェントシステムを構成する場合には，エージェント間の相互運用性を確保することが重要になる．

5-5-1 標準に基づいた相互運用性の確保

相互運用性を実現するためには，何らかの標準に基づいてエージェントシステムを構築することが考えられる．エージェントシステムの代表的な仕様として，Foundation for Intelligent Physical Agents（FIPA）が策定したものがある [1]．FIPA の仕様は自由度が大きく，FIPA の仕様に準拠していても，そのままエージェント間で相互にメッセージのやり取りを可能にするとは限らない．例えば，エージェント通信言語 FIPA-ACL では，メッセージの通信行為の意味を定式化しているが，メッセージの内容を記述する言語ならびに語彙は通信行為とは独立に指定するようになっている（7 編 4 章 4-4 節参照）．エージェント間でメッセージをやり取りするためには，これらを同じものとするか，あるいは違いを吸収することが必要になる．

また，FIPA の仕様に基づき構築されたエージェントシステムの相互接続性を世界規模で実証するプロジェクトとして AgentCities[2] がある．そこでは，異なるエージェントシステム間でメッセージのやり取りを実現するだけではなく，エージェントが提供するサービスを人間の介在なしにネットワーク上で発見し，利用できる意味的な相互運用性が重要であるとしている．

5-5-2 異なるエージェントプラットフォーム間の相互運用性

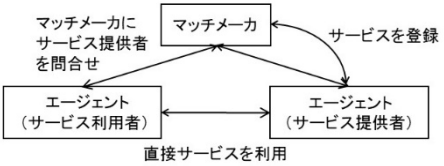
オープンな環境では，必ずしもすべてのエージェントシステムが 1 つの仕様に基づいて構築されるとは限らない．特にエージェントシステムはエージェントプラットフォームを用いて構築される場合が多く，異なるエージェントプラットフォームをまたいでエージェント間の連携を実現する手法が必要になる．

そこで，エージェントプラットフォーム間の仲介役となるエージェントを設ける手法が提案されている．例えば，異なるエージェント通信言語を使用する RETSINA システムと Open Agent Architecture（OAA）システムとの間で，サービスを提供するエージェントを相互に発見し，呼び出す機能を持った RETSINA-OAA InterOperator がある [3]．RETSINA-OAA InterOperator は，それぞれのエージェントプラットフォームからは，スーパーエージェントと呼ばれる一つのエージェントとして見える．同様に，ゲートウェイエージェントを設け，異なるエージェント通信言語（例えば，FIPA-ACL と KQML（Knowledge Query and Manupilation Language））の間でメッセージを変換する試みも行われている [4]．

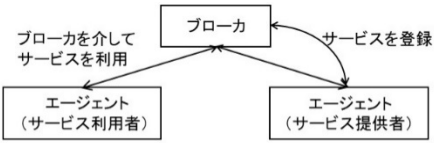
5-5-3 エージェントレベルでの相互運用性

エージェントレベルの相互連携として，協調を促進するミドルエージェントの概念がある [5]．サービスを提供するエージェントとサービスを利用するエージェントの間を仲介するもので，事前にお互いの存在を知ることなくエージェント間の連携を実現する．代表的なものとしてマッチメーカとブローカがある（**図 5･1**）．マッチメーカは，サービス利用エージェントが要求す

るサービスを提供するエージェントを紹介する．サービス提供エージェントが紹介された後は，サービス利用エージェントは，直接サービス提供エージェントとやり取りを行い，マッチメーカは間に入らない．これに対して，ブローカは，サービス利用エージェントとサービス提供エージェントの間に入り，サービス提供エージェントに対する要求はブローカを介して，サービス提供エージェントに伝えられ，その結果は，ブローカを介して，サービス要求側に伝えられる．



(a) マッチメーカ



(c) ブローカ

図 5・1　マッチメーカとブローカ

エージェント通信言語 KQML では，同様の機能がファシリテータ（Facilitator）と呼ばれるエージェントとして実現され，broker，recruit，subscribe，advertise などの仲介を実現するためのエージェント通信言語の遂行語（Performative）が定義されている．

■参考文献

1) Foundation for Intelligent Physical Agents (FIPA)　http://www.fipa.org/
2) J. Dale, B. Burg, and S. Willmott :“The Agentcities Initiative: Connecting Agents Across the World,”in Innovative Concepts for Agent-Based Systems, LNCS 2564, pp.453-457, 2003.
3) J.A. Giampapa, M. Paolucci, and K. Sycara :“Agent Interoperation Across Multagent System Boundaries,”in Proceedings of the Fourth International Conference on Autonomous Agents, pp.179-186, 2000.
4) H. Suguri, E. Kodama, M. Miyazaki, and I. Kaji :“Assuring Interoperability between Heterogeneous Multi-Agent Systems with a Gateway Agent,”in Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering, pp.167-170, 2002.
5) K. Sycara :“Multi-agent Infrastructure, Agent Discovery, Middle Agents for Web Services and Interoperation,”in Multi-Agent Systems and Applications, LNAI 2086, pp.17-49, 2006.

■7 群－7 編－5 章

5-6 エージェント設計手法・方法論