名古屋工業大学博士論文
甲第385号(課程修了による)
平成１４年３月２２日授与

博士論文

知的モバイルエージェントの構築環境に関する研究

2002年1月

福田 直樹

論文要旨

本論文では，知的モバイルエージェントの構築環境の実現に関する研究成果を示す．知的モバイルエージェントとは，知的エージェントとモバイルエージェントの性質を併せ持つソフトウェアである．知的エージェントは，相互に協調して動作する能力を持つソフトウェア主体である．知的エージェントは，中央からの制御が与えられないような分散環境における，ソフトウェアシステム構築のモデルとして用いられる．モバイルエージェントは，ネットワークで接続された計算機間を移動しながら，計算を継続することのできるソフトウェアとしてモデル化される．モバイルエージェントの特長は，複数の計算機を利用するソフトウェアを単一のソフトウェアとしてモデル化可能な点，およびソフトウェアの導入や削除などのソフトウェア運用方法に関する制御をソフトウェア自身に組み込むことが可能な点である．

知的モバイルエージェントを適用する環境は，異種エージェントおよび異種計算機が存在するという点で開放性を持つといえる．開放的な環境は，必然的に可変性を持つ．異種エージェントが相互にサービスを提供するという点で環境は対称性を持ち，それらのエージェントおよび計算機の存在に偏りがあるという点で環境は資源偏在性を持つ．環境の持つこれら４つの性質をエージェントが適切に扱うためには，エージェントは自律性，即応性，相互操作性，および移動性を持つ必要がある．これら４つの性質を持つ知的モバイルエージェントの構築環境を実現することが課題となる．

本論文では，これらの課題を解決するために，論理型言語に基づく知的モバイルエージェントの構築環境 *MiLog* フレームワークを提案する．論理型言語は，自己言及が可能で記号処理の記述にも適しているため，エージェントの自律性の実現にとって有用である．エージェントの相互操作性を実現するために，論理型言語上にエージェント間通信の記述を実現する．エージェントの即応性を実現するために，論理型言語を割り込み処理の記述が可能となるように拡張する．エージェントの移動性を実現するために，強モビリティを実現する．割り込み処理と強モビリティを組み合わせた，任意時刻モビリティを提案する．任意時刻モビリティを実現するための機構として *MiLogEngine* を設計する．*MiLogEngine* をソフトウェアコンポーネントとして構成し，アプリケーションへの埋め込み，及びオブジェクト指向に基づく拡張を実現する．本フレームワークの特筆すべき点は，これらの機能をもつエージェント記述言語処理系を単に実現するだけでなく，それをコンパクトで拡張性の高いアプリケーションフレームワークとして実現している点である．本フレームワークを，GUI アプリケーションフレームワーク *iML*，WWW アプリケーションフレームワーク *MiPage* の実装，および知的エージェントに基づく電子商取引支援システム *BiddingBot* の構築に適用することにより，本フレームワークの有用性を示す．

本論文の構成は次のようになっている．

第1章では，序論として本研究の動機と目的を述べる．

第2章では，本研究の背景を述べ，モバイルエージェントと知的エージェントに関する研究を調査し，本研究の課題を明らかにする．

第3章では，本研究で実現する知的モバイルエージェントフレームワーク *MiLog* の基本的な設計を示す．知的モバイルエージェントが持つべき性質について検討するために，それを適用する環境が持つ特性を考察する．環境が，開放性，可変性，対象性，および資源偏在性を持つことから，知的モバイルエージェントが持つべき性質として，自律性，即応性，相互操作性，および移動性が重要となることを述べる．エージェント記述言語 *MiLog* における，エージェント間通信，割り込み処理，およびモビリティの記述とその意味論を定義する．ここで定義したエージェント記述言語を用いて，アプリケーションフレームワークを構築する．本フレームワークの実行環境を，複数インタプリタの並行実行と相互作用に基づく構成により実現する．本実行環境の利用方法を示し，本実装の課題について議論する．

第4章では，論理型言語上における任意時刻モビリティの実装として，*MiLogEngine* を示す．*MiLogEngine* では，プログラムの実行状態をコンパクトに保存可能とするために，Box モデルに基づく実行状態分割手法を新たに提案した．本手法を用いることにより，任意時刻モビリティが効果的に実現される．本実装を，既存のモバイルエージェントフレームワークの実装および論理型言語処理系の実装と比較し，本実装手法の有効性を評価する．

第5章では，本フレームワークの応用事例として，2つのフレームワークと1つのシステムの構築について述べる．アプリケーション開発支援フレームワーク *iML*，および WWW アプリケーションフレームワーク *MiPage* の実現について述べる．電子商取引システム *BiddingBot* の実装における *MiLog* フレームワークの有用性を示す．

第6章では，これまで述べてきた結果を総括し，本研究の成果および今後の課題を示す．

目 次

図目次

表目次

第1章

序論

1.1 本研究の動機

XEROX PARCで開発されたALTOマシンによるブレークスルーにより，コンピュータの持つ計算能力は我々の日常的な活動にも向けられるようになった[64]．1990年代にTim Berners Leeによって開発されたWWW(World Wide Web)によって，計算機のネットワーク化が飛躍的に進み，あらゆる情報やサービスが電子化され，ネットワークを介して利用できるようになってきている[15]．

これらの電子化された多量の情報を前に，我々はまさに知的な電子秘書の支援を必要とする状況に置かれるようになっている．電子化された大量の情報やサービスを効果的に扱うことを可能とするための技術として，情報エージェントが提案されている[79]．情報エージェントは，インターネット上などに電子的に蓄えられた情報や利用可能なサービスを，ユーザにとって扱いやすい性質のものへ変換し，状況に応じてユーザの代理として操作を行うソフトウェアである．

情報エージェントに関する先駆的な研究に，Sycaraらの分散知的エージェントに関する研究[122]がある．Sycaraらは，研究室の来客応対支援システムおよび，ポートフォリオ管理システムに情報エージェントの技術を適用した．Sycaraらが構築した来客対応システムでは，得られた来客情報を元に，エージェントが研究室の教官らの研究分野とスケジュールから，その来客の興味に会った最も適切な教官を担当として選び出し，その教官とのスケジュール交渉を行う．本システムを用いることにより，来客は，興味に近い研究内容についての話を聞くことができ，担当教官側も，自身の研究に興味を持っている人に優先的に研究をプレゼンテーションすることができる．さらに，本システムを用いるためのコストはコンピュータ1台分で済み，特別な秘書を雇う必要がないため，教官は研究のためにより多くの予算を割くことができる．ポートフォリオ管理システムでは，ポートフォリオ(性質の異なる複数の有価証券等を組み合わせたもの)を管理するのに必要となる株価等の値動きやニュース記事等の情報を，エージェントが自律的に収集し，市場動向が特定のパターンの動きを見せたときにすばやく利用者に知らせたり，ポートフォリオの構成方法を検討するために必要となる価格変動情報などを効果的にユーザに提示することが可能である．すなわち，ポートフォリオ管理システムは，ユーザがポートフォリオの管理必要な情報を，状況に合わせて適切にかつタイムリーに提示する．これら2つのシステムの特長は，

ユーザに対し，適切な情報を適切なタイミングで提示するとともに，特定の作業をユーザに代わって行う機能を持っている点である．

情報エージェントに関連した他の研究には，複数の主体によって提供される情報源の統合を行う異種データベース統合に関する研究 [13]，自律的な情報エージェントを用いて電子商取引やその支援への適用を目指した研究 [31]，HTML 等の半構造データから高速かつ高精度に情報抽出を行う手法に関する研究 [80]，記述論理を用いた情報へのメタ情報の付与に関する研究 [17]，ユーザに合わせた情報収集を実現するために学習やユーザプロファイルを適用する研究 [86] などが多岐にわたって行われている．情報エージェントに関する研究は，異なる複数の分野における研究成果が統合的に用いられる，応用的研究分野である．

情報エージェントに関連した研究の中で，特に注目されているのが，電子商取引への適用である．インターネットの普及によって大量の取引が電子取引に置き換わりつつあり，それらは法人間での取引 (Business to Business, B2B) にとどまらず，法人-個人間 (Business to Consumer, B2C) および個人-個人間 (Consumer to Consumer, C2C) に広がりつつある．例えば，B2B 取引では，IBM が，WWW を用いた B2B マーケットプレース [68] の開発を進めている．B2C では，Amazon.com[3] がインターネットを用いた書籍販売を行い，その業績を伸ばしつつある．また，C2C では，Yahoo Auctions[6] や，eBay[32] 等のオンラインオークションサイトを通じて，多数の財がやり取りされている．すなわち，電子商取引は多くの人たちに利用され，その重要度も今後高まることが予想される．電子商取引に情報エージェントを適用するためには，少なくとも次の 2 つの課題を解決する必要がある．1 つは，通信遅延の問題であり，もう 1 つは，商取引に十分に適用可能なエージェントの自律性の実現である．商取引では，「時は金なり」であり，状況に応じて遅延なく取引を行うことが利益をもたらす．商取引を取り巻く状況は不確実性を持ちながら常に変化を続けるため，エージェントには状況に応じた高度に自律的な行動をとることが求められる．商取引は地球レベルで分散しており，これらの動的に変化する環境を単一のエージェントで扱うには限界がある．したがって，エージェントの自律性は，他のエージェントとの協調的な振る舞いに昇華されなければならない．

ネットワークの遅延，帯域の制約，通信路の切断を効果的に扱うための技術として，モバイルエージェントが提案されている [131, 108, 116]．モバイルエージェントは，ネットワークで相互接続された計算機上を，計算を継続しながら移動可能なソフトウェアである [108]．モバイルエージェントは，ネットワークの遅延，帯域の制約，切断に対して効果的に振舞うことのできるソフトウェアを構築するためのソフトウェア構築パラダイムとして与えられる．同時に，モバイルエージェントはネットワークで接続された分散環境におけるアプリケーションソフトウェア開発，および保守を容易にするための基礎技術を提供する [116]．

モバイルエージェントの先駆的な研究は White らによって行われ，その成果は商用ソフトウェア Telescript として発表された [131]．Telescript では，エージェントはプレースと呼ばれる実行環境上で動作する．プレースはネットワークで接続された複数の計算機上で動作させることができ，エージェントは，プレース上を移動しながらタスクの遂行を行うことが可能である．エージェントは，JAVA に似たオブジェクト指向言語 (High Telescript) で記述され，より低レベルな言語 (Low Telescript) にコンパイルされた後，プレース上で実行される．Telescript では，セキュリティの確保のために，オーソリティやエージェント

の寿命の概念が取り入られれている．Telescript システムは，AT&T の PersonaLink サービスに適用された．Telescript の特徴的な点は，その言語仕様に至るまでモバイルエージェントの実現に焦点を絞って開発された点である．

情報エージェントがユーザに代わって情報の収集，選別を行い，さらにユーザに代わって（例えば，オンラインオークションへの入札など）情報源への特定の操作を行うことを可能とするためには，情報エージェントは情報源から得た情報と，事前にユーザから与えられた知識に基づいて，自律的に推論し，行動する必要がある．自律的なソフトウェアを実現するための方法論は，知的エージェント関連の研究で議論されている [73]．知的エージェントの実現では，エージェントの自律性の実現が重要視されている．エージェントの自律性の実現のために，BDI アーキテクチャに基づくアプローチが提案されている [57, 94]．BDI アーキテクチャでは，エージェントの認識行動サイクルを Belief, Desire, Intention の 3 つの要素に分けて扱うことにより，不確実性を持ち動的に変化する状況におけるエージェントの行動を規定するための理論的な基盤を与えている．

知的エージェントとモバイルエージェントの性質を併せ持つ「知的モバイルエージェント」の実現は，情報エージェントの実現にとって有用であると考えられる．しかし，知的エージェントとモバイルエージェントにおける性質の違いから，両者の性質を併せ持つソフトウェアの実現は容易ではない．モバイルエージェントの構築では，エージェントの移動におけるオーバヘッドを低減するために，本質的にエージェントをいかに簡潔に構成するかが重要となる．したがって，モバイルエージェントの記述には，手続き的で構造が単純な言語が用いられる場合が多い．一方で，知的エージェントは，エージェントに自律的な振る舞いをさせる目的から，高度で複雑な内部処理機構を持つ必要があるため，エージェントの構成が複雑化する傾向がある．知的エージェントの記述では，記号処理などを簡潔に記述できる宣言型言語が用いられる．記述言語の性質の違いを克服することが，知的モバイルエージェントの実現にとって重要な課題の 1 つとなる．

記号処理を簡潔に記述するのに適した言語として，論理型言語がある．論理型言語では，宣言的な知識の記述によってプログラミングを行う．論理型言語は，特に知的エージェントの Belief の表現形式に適しており，知的エージェントの知識表現，およびシステム構築への適用が研究されている [19]．論理型言語は，第 5 世代コンピュータプロジェクトにおいて主要な研究課題とされ，並列論理プログラミング [55, 129]，制約論理プログラミング [91]，帰納論理プログラミング [113] など多くの発展的な研究成果を生み出した．これらの研究成果の多くは（例えば，大型並列計算機のような）豊富な計算機資源の存在を前提としたものであり，その処理系は複雑な内部構造をもつため，そのままではモバイルエージェントに適用することが困難である点が課題となる．

1.2 本研究の目的

本研究では，知的モバイルエージェントの構築に関連して，以下の 3 つの目的により研究を行う．

目的 1: 知的モバイルエージェントの記述言語と構築環境（フレームワーク）の設計

目的 2: 知的モバイルエージェント言語処理系の実装と高性能化

目的3: 知的モバイルエージェントのアプリケーション構築への適用

1.2.1 目的1:知的モバイルエージェントの記述言語と構築環境の設計

本研究の第1の目的は，知的モバイルエージェントを効果的に実現するためのエージェント記述言語と構築環境（フレームワーク）を設計することである．設計を効果的に進めるために，最初に知的モバイルエージェントが対象とする問題領域のモデル化を行う．このモデル化では，まず，知的モバイルエージェントの置かれる環境の特性を明らかにする．次に，知的モバイルエージェントの置かれる環境の特性を考慮した場合に，知的モバイルエージェントが持つべき特性を明らかにする．次に，知的モバイルエージェントとその問題領域の特性から，知的モバイルエージェントを記述するための言語および構築環境で実現すべき機能と性質についてまとめる．知的モバイルエージェントの記述を行うためのエージェント記述言語を定義する．エージェント記述言語では，エージェント自身の振る舞い，エージェントの移動，および他のエージェントとの通信の記述方法を定義する．エージェントの協調的な振る舞いを可能とするためには，他のエージェントとの通信，および動的な環境変化への即応的な振る舞いの実現が必要となる．本記述言語では，問い合わせ（Query）に基づくエージェント間通信，およびエージェントの即応的な移動を実現する任意時刻モビリティの実現の必要性について議論し，それらを実現するエージェント記述言語の設計を示す．エージェント記述言語を効果的に利用可能とするために，本エージェント記述言語に基づく知的モバイルエージェント構築フレームワーク *MiLog* を設計する．本フレームワークによる知的モバイルエージェントの効果的な構築手法を提供する．

1.2.2 目的2:知的モバイルエージェント言語処理系の実装と高性能化

第2の目的は，目的1で設計を行ったエージェント記述言語および構築環境を実際に利用可能とするために，知的モバイルエージェント言語処理系の実装，およびその高性能化を行うことである．本処理系の実装では，任意時刻モビリティの実現方法が主な課題となる．本処理系の高性能化では，論理型言語処理系をコンパクトに実装して，モバイルエージェントの低オーバヘッドによる移動を可能とすることが課題となる．既存の論理型言語処理系では，プログラム実行途中の状態を保存する機能を提供しなかったり，仮に提供していてもそのサイズが大きいため必ずしもモバイルエージェントへに適さない点が課題である．この課題を解決するために，モバイルエージェントの構築に適した論理型言語処理機構（*MiLogEngine*）を新たに構築する．本処理機構では，任意時刻モビリティを実現するために，プログラムの実行状態を実行途中の状態まで含めて移動可能とする．本処理機構では，プログラムをその意味論を保ったまま処理機構中にコンパクトに保存し，エージェント移動時の実行状態転送オーバヘッドの低減を実現する．本処理機構では，これらの特徴を実現しながら，実用アプリケーションの構築には十分な実行速度を確保する．本処理機構は，より多くの環境で利用可能とするために JAVA 言語により実装する．本処理機構を JAVA 言語で実装する場合における課題と，その解決方法を示す．本機構を，目的1で設計した知的モバイルエージェントフレームワークの構築に適用する．

1.2.3 目的3:知的モバイルエージェントの応用

第3の目的は，目的1および目的2で構築した知的モバイルエージェント構築フレームワークをアプリケーション開発に応用可能とするために，本フレームワークを用いてアプリケーション構築フレームワークを実現し，さらに電子商取引アプリケーションの構築に適用する．

ここで実現するアプリケーションフレームワークでは，アプリケーション構築時のユーザインタフェースに2種類のものを考慮し，それらを用いたアプリケーションソフトウェアの構築支援環境を構築する．1つは，WWWに基づくユーザインタフェースの構築支援である．WWWブラウザによる操作は，他のGUIプログラミングと比較してその構築が比較的容易であり，HTMLによるユーザインタフェースは機種ごとの差異がなく遠隔地からでも利用できるという利点がある．本支援環境では，WWWに基づくユーザインタフェースの構築を支援し，さらにモバイルエージェントとWWWの統合までを視野に入れ，これらを実現するための機構 *MiPage* を実現する．*MiPage* では，HTMLによって記述されたユーザインタフェース中に論理プログラムを埋め込むことを可能とし，WWWサービスを提供するモバイルエージェントとして運用する機構を実現する．もう1つは，GUIに基づくユーザインタフェースの構築支援である．HTMLを用いてWWWブラウザで実現できるユーザインタフェースには制約があり，例えばユーザに対して情報を送信（プッシュ）する場合には，従来のGUIに基づくユーザインタフェースのほうが適している．本研究では，本フレームワークを用いてGUIを持つモバイルエージェントの構築を支援するための機構 *iML* を実現する．*iML* では，視覚的な設計ツールによりGUIの開発を支援し，GUIを持ったモバイルエージェントの移動を実現するための機構を提供する．

これら2つのアプリケーション構築フレームワークを実装し，学生のプログラミング演習への適用を試みる．プログラミング演習では，限られた時間内でプログラミング環境の扱い方を習得させる必要がある．プログラミング演習への適用は，本構築支援環境の扱いやすさを判断する指標となりうる．また，プログラミング演習に最新の技術を持ち込むことにより，学習者の興味と好奇心を刺激し，より高い学習効果をもたらすことが期待される．

電子商取引アプリケーション構築への適用として，本研究と並行して行われた研究の1つである，協調的なマルチエージェントに基づくオンラインオークション入札支援システム *BiddingBot* の構築に本フレームワークを適用する．*BiddingBot* では，実際に存在するオークションサイトからの情報抽出および入札を可能としており，本フレームワークの実アプリケーションへの適用可能性を評価する上で適切であると考えられる．*BiddingBot* の構築における様々な経験から，本フレームワークの適用可能性および限界を明らかにする．

1.3 本研究の貢献

本研究は，モバイルエージェント，論理プログラミング，知的エージェント，およびプログラミング教育の分野に貢献している．本節では，各分野における本研究の貢献を示す．

モバイルエージェントの分野 モバイルエージェントの分野では，論理プログラミングという新たなプログラミング言語の選択肢を提供する点，および，モバイルエージェントの

マルチエージェントシステムへの適用の可能性を示す点で貢献する．モバイルエージェントシステムの研究では，オブジェクト指向言語をプログラミング言語として用いる研究が多く行われているが，記号処理言語，特に論理プログラミングの適用についての研究は今後の発展が期待される部分である．本研究では，論理プログラミングをモバイルエージェントの構築に効果的に利用することにより，知的なモバイルエージェントの実現をより行いやすいものとする．本研究で実現する枠組みを用いて，モバイルエージェントを協調的エージェントに基づくアプリケーションの構築に適用する．

論理プログラミングの分野 論理プログラミングの分野では，マルチエージェントシステムの構築への適用可能性を示す点，論理プログラミングを利用したGUIの構築手法を示す点，任意時刻モビリティをメモリに関して効率よく実現するための手法を示す点で貢献する．従来の論理型言語の実装では，マルチスレッドやスレッド間通信などの，マルチエージェントシステムの構築に必要な機構の実装例が少ないため，論理プログラミングによるマルチエージェントシステムの構築を行う研究は今後の発展が期待される分野である．論理プログラミングでは，記号処理に特化した実装は数多くあるが，GUI等のアプリケーションの実現に必要な機能についての考察はこれからである．一部実用化されているシステムもあるが，使い勝手の面などで発展の余地が大きい．本研究では，学部の学生のプログラミング教育にも適用可能な使い勝手のよいシステムを提案することで，論理プログラミングの新たな活用方法を示すことが貢献となる．

知的エージェントの分野 知的エージェントの分野では，知的エージェントの構築に論理プログラミングを適用することでシステムの低コストでの構築を実現可能とする点，および知的エージェントによるシステムの構築にモバイルエージェントを適用することにより，新たな知的エージェントに基づくシステムの構築手法を示す点で，貢献している．従来は，知的エージェントによるシステムの構築を行う際に，エージェントそのものの実現やエージェント間通信の実現に多大な構築コストをかける必要があった．これまでに，これらのエージェント構築基盤に関する研究がいくつか行われているが，エージェントにモビリティを持たせる部分を統合したエージェント構築環境は今後発展が期待されるものであり，本研究によって知的エージェントの新たな構築環境を提案することが貢献となる．

プログラミング教育の分野 プログラミング教育の分野では，論理プログラミング，マルチエージェントおよびモバイルエージェントという最新の技術に対して，プログラミングを学ぶ人たちが理解しやすく興味を持って学習するための基盤を提供する点から貢献する．これらの最新技術をプログラミング教育に適用した事例を作ることは現在必要とされていることである．本研究で実現する枠組みをプログラミング教育に適用可能とすることで，プログラミング教育の発展に貢献する．

1.4 本論文の構成

本論文の構成を以下に示す．

第2章では，本研究の背景を述べ，モバイルエージェントと知的エージェントに関する研究を調査し，本研究の課題を明らかにする．

第3章では，本研究で実現する知的モバイルエージェントフレームワーク *MiLog* の基本的な設計を示す．知的モバイルエージェントが持つべき性質について検討するために，それを適用する環境が持つ特性を考察する．環境が，開放性，可変性，対象性，および資源偏在性を持つことから，知的モバイルエージェントが持つべき性質として，自律性，即応性，相互操作性，および移動性が重要となることを述べる．エージェント記述言語 *MiLog* における，エージェント間通信，割り込み処理，およびモビリティの記述とその意味論を定義する．ここで定義したエージェント記述言語を用いて，アプリケーションフレームワークを構築する．本フレームワークの実行環境を，複数インタプリタの並行実行と相互作用に基づく構成により実現する．本実行環境の利用方法を示し，本実装の課題について議論する．

第4章では，論理型言語上における任意時刻モビリティの実装として，*MiLogEngine* を示す．*MiLogEngine* では，プログラムの実行状態をコンパクトに保存可能とするために，Box モデルに基づく実行状態分割手法を新たに提案した．本手法を用いることにより，任意時刻モビリティが効果的に実現される．本実装を，既存のモバイルエージェントフレームワークの実装および論理型言語処理系の実装と比較し，本実装手法の有効性を評価する．

第5章では，本フレームワークの応用事例として，2つのフレームワークと1つのシステムの構築について述べる．アプリケーション開発支援フレームワーク *iML*，および WWW アプリケーションフレームワーク *MiPage* の実現について述べる．電子商取引システム *BiddingBot* の実装における *MiLog* フレームワークの有用性を示す．

第6章では，これまで述べてきた結果を総括し，本研究の成果および今後の課題を示す．

図1.1に，本論文の構成を各章ごとの関連図として示す．

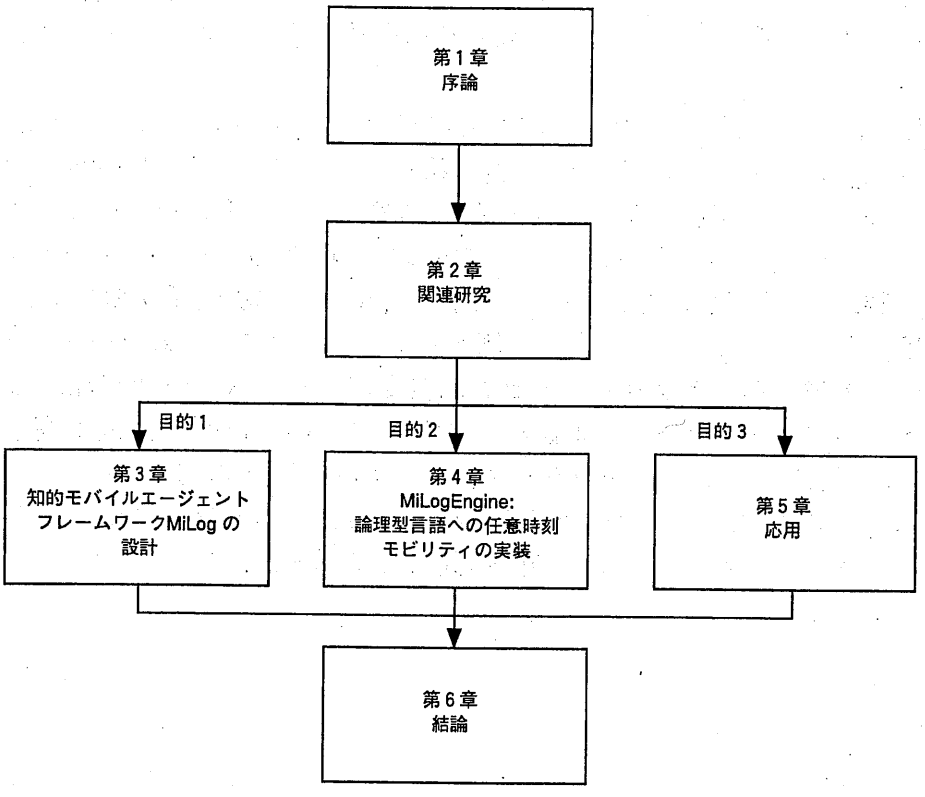

図 1.1: 本論文の各章ごとの関連

第2章

関連研究

2.1 はじめに

本章の目的は，本研究の土台となった研究および既存の関連研究との差分を明確にし，本研究の位置付けを行うことである．最初に，本研究のおおまかな位置付けを示す．

モバイルエージェントと関連しており，その背景にもなっている研究に，プロセスマイグレーション [90] に関する研究がある．プロセスマイグレーションは，分散コンピューティングにおける動的負荷分散手法として，主にオペレーティングシステムの分野において研究された．プロセスマイグレーションでは，主に同種の複数の計算機をネットワーク接続した環境において，プロセスのネットワーク透過な移動を実現する．すなわち，プロセスの利用するメモリ領域およびプロセス中のレジスタやプログラムカウンタ等の情報を保存し，ネットワークを経由して他の計算機上へ転送し，復元する．プロセスマイグレーションで主に問題となったのは，プロセス転送を実現するための機構の設計自身が非常に複雑であり，複雑な構造をもつソフトウェアを実装したため，そのオーバヘッドが負荷分散による性能向上を相殺してしまう場合がある点である．移動を行う対象を，自身の移動を必ずしも考慮しない汎用のプロセスから，それ自身が移動することを強く意識して設計されたソフトウェアとしたのが，モバイルエージェントである．プロセスマイグレーションでは，環境が対称に近い構造を持っていることを前提としていたが，モバイルエージェントでは偏りがあって非対称な環境，すなわち，異なるアーキテクチャで異なる性能の計算機が，異なる帯域や構造を持つネットワークで相互接続された環境を扱う点が異なる．本研究で扱う知的モバイルエージェントの適用環境は，モバイルエージェントで扱うのと同様に，資源の偏りを前提とした環境である．

モバイルエージェントと類似した研究として，分散環境における位置透過性の実現がある．例えば，UNIX の NFS(Network File System) では，複数の計算機上から共有された単一のファイルシステムを提供している．位置透過性の利点は，位置の違いを意識せずに統一的にサービスを利用できる点である．モバイルエージェントでは，位置の違いをむしろ明確に区別し，位置ごとの特性を最大限に利用することに重点を置いている点が異なる．本研究で扱う知的モバイルエージェントも，モバイルエージェントに関する研究と同様に，エージェントが現在存在する「位置」の違いを明確に区別し，その位置ごとの特性を最大限に利用することに焦点を当てる．

知的エージェントの背景となった研究に，人工知能における記号処理がある．人工知能における記号処理では，環境に対する知的な振る舞いを実現するにあたって，その知的な主体の内部に環境についての明示的な表現を持たせ，その表現に対して明示的な操作を行うことによって知的な振る舞いをさせる．背景となる研究には，プランナーおよび推論エンジンに関する研究がある．エージェントという記述が頻繁に使用されるようになったのは，分散人工知能 (Distributed Artificial Intelligence, DAI) の分野におけるマルチエージェントの研究による．マルチエージェントの研究では，複雑で困難な問題解決に，複数の単純なソフトウェア（エージェント）を用い，それらのエージェント間の協調により問題解決を行う点が特徴である．知的エージェントでは，分散人工知能における狭い範囲での協調動作をより汎用的なものと捉え，制約された資源のもとでの自律的に振舞うエージェントについて，それらの間の協調的な振る舞いの実現について議論している．知的エージェントの研究では，環境はエージェントにとって未知なものであり，エージェントはそれ自身が自律的な振る舞いをする主体であるとしている点が，分散人工知能におけるエージェントの捉え方と異なる点である．本研究で実現を目指す知的モバイルエージェントは知的エージェントの延長線上にあるエージェントであり，自律的で移動能力を持ったエージェントである．

論理型言語に関する研究では，論理型言語の持つ処理の潜在的な並列性や宣言的な記述能力を拡張する言語が多く提案されている．第5世代コンピュータプロジェクトで主に研究された並列論理型言語は，単一化という操作を基礎として並列処理の記述に特化した言語である．並列論理型言語における「並列」の目指すところは，処理の高速化である．一方で，本研究で実現するエージェント記述言語では，並列性を利用した高速な処理の記述を目的としていない．本研究におけるエージェント記述言語で目指すのは，プログラミングの容易化と，計算機の柔軟な利用である．すなわち，複数の計算機を利用するプログラムを容易に構築可能とし，計算機をその場の目的に合わせて柔軟に利用することを目的としている．論理型言語における記述力の拡張として，制約処理機構を備えた制約論理型言語や，線形論理の適用により資源管理を効果的に記述できる線形論理型言語が提案されている．本研究では，制約論理型言語や線形論理型言語のように論理型言語の記述力そのものを向上させることは意図しない．本研究で意図するのは，論理型言語の適用範囲をモバイルエージェントへ広げることである．

本章の以降は，次のようになっている．2.2節で，モバイルエージェントについての基礎的な事項をまとめる．2.3節で，モバイルエージェントシステムにおいて本研究に関連する研究をまとめ，本研究との相違点を示す．2.4節で，論理型言語の実装技術について基礎的な事項をまとめるとともに，関連研究を調査し本研究との相違点を示す．2.5節で，知的エージェントフレームワークについて基本的な事項をまとめるとともに，関連研究を調査し本研究との相違点を示す．

2.2 モバイルエージェント

2.2.1 モバイルエージェントの定義

モバイルエージェントは，計算機間を移動しながら計算を行うソフトウェアである．モ

バイルエージェントという言葉の厳密な定義については，様々な定義がなされており，十分な同意が得られていない．それらの定義において共通している点は，モバイルエージェントが，その特徴的な性質として，計算機間を移動しながら計算を継続する能力（モビリティと呼ぶ）を持つことである．

モバイルエージェントと関連する分野には，分散コンピューティング，ソフトウェア設計パラダイム，知的エージェント（人工知能），およびプログラミング言語設計がある．それぞれの関連分野ごとに，モバイルエージェントという言葉に対する定義が異なっている．

分散コンピューティングの分野では，モバイルエージェントは計算機間を**移動する能力を持つオブジェクト**である．モバイルエージェントの役割は，CPUおよびネットワークの負荷を低減することである．モバイルエージェントの機能は，ORB(Object Request Brokering)の一部として統合的に扱われる．

知的エージェントの分野では，エージェントは動的な環境において自律的に動作する主体である．モバイルエージェントが動作する環境を，動的な環境の具体例としてとらえ，**知的エージェント技術の応用を行う対象**としてモバイルエージェントをとらえている．

ソフトウェア設計パラダイムの分野では，モバイルエージェントはクライアントサーバパラダイムと並ぶ**ソフトウェアの構成パラダイム**の１つである．モバイルエージェントというパラダイムの特性を解析し，その特性を生かしたアプリケーションドメインを明らかにすることを目標としている．

プログラミング言語設計の分野では，モバイルエージェントの効果的な記述方法および実現方法の探求が研究対象である．モバイルエージェントは，**計算機間を移動することのできるオブジェクトあるいはスレッドのあつまり**であり，処理系ごとにその形体が定義される．たとえば，Telescript[131]では，モバイルエージェントはそれ自身がスレッドを持っていて計算機間を移動可能なオブジェクトであると定義されるが，Aglets[84]ではモバイルエージェント自身がスレッドを必ずしも内包せず，外部からのイベント通知により駆動される．プログラミング言語設計の分野では，モビリティの実現方法，およびモビリティのプログラミングからの効果的な利用方法が議論される．モビリティをコードレベルで記述可能とする手法は，**コードモビリティ**と呼ばれ，モバイルエージェントとは区別される．コードモビリティを実現し，コードモビリティを用いて記述したソフトウェアを実行可能とするための環境を提供するシステムは，**モバイルエージェントシステム**と呼ばれる．

本論文では，広義のモバイルエージェントを，「実行環境間の移動の概念を用いることで，複数の実行環境にまたがる処理の制御が可能であり，なおかつそれらの実行環境や通信路の性質の違いおよび変化を主体的にかつ明示的に扱うことのできるソフトウェア主体」と定義する．また，狭義のモバイルエージェントを，「あるモバイルエージェントシステム上で移動可能な最小限の主体」と定義する．本節では，以後は特に断りがない限りモバイルエージェントとは，広義のモバイルエージェントを意味する．

この定義に従えば，広義のモバイルエージェントは，ある局所的な範囲でクライアントサーバモデルを用いたモジュールから構築されることもありうる．ただし，位置透過性を実現する分散システムは，実行環境（位置）ごとの性質の違いが利用者および設計者から隠蔽されているため，広義のモバイルエージェントには含まれない．

モバイルエージェントは「インターネット上など大規模な範囲に適用可能で，かつオープンな多数の実行環境に対して匿名で移動が可能なソフトウェア」と定義される場合もある

が，本論文では，モバイルエージェントを開放的な環境上での運用に限定せず，たとえアプリケーションとして閉じた環境内での運用であっても，先に定義したモバイルエージェントの特性を利用して構築したソフトウェアを，モバイルエージェントとみなすことにする．

2.2.2 モバイルエージェントの利点

ソフトウェアの設計モデルとしての利点 モバイルエージェントをソフトウェアの設計モデルとして用いる利点は，設計モデルの内部にそのソフトウェアの稼働環境の性質（計算速度，他のエンティティとの通信速度や遅延など）を明示的に表現可能な点，および複数の稼働環境の性質の違いを考慮し，それらを適切に扱うためのソフトウェアを，単体のソフトウェアとしてモデル化可能な点である．例えば，サーバ-クライアントモデルでは，複数の稼働環境にまたがるソフトウェアは，単体のソフトウェアとしてはモデル化されず，少なくともサーバとクライアントの2つのソフトウェア（エンティティ）によってモデル化される．モバイルエージェントを用いると，クライアントとサーバをあわせて1つのソフトウェアとして扱うことが可能となる．

モバイルエージェントのモデルを用いることで，ソフトウェアをより高度に抽象化された主体としてモデル化することが可能となり，すなわちソフトウェアのモデルをより簡潔にすることに貢献する．ソフトウェアのモデルが簡潔になれば，モデルの理解が容易になることが期待される．ソフトウェアのモデルの理解が容易になれば，そのソフトウェアを実装する開発者が短期間で正確にモデルを理解可能となることが期待される．開発者が，実現すべきソフトウェアのモデルを短期間で正確に理解できるようになれば，開発者はモデルの理解のために使っていた労力をモデルの実現に集中することが可能となり，最終的に実現されるソフトウェアシステムの品質[1]が向上することが期待できる．また，ソフトウェアのモデルの理解が簡単になることにより，プログラミングの学習者にとってもモデルの理解が容易になるという利点があり，ソフトウェア開発コストの低減に役立つことが期待される．ソフトウェアのモデルが簡潔になることで，ユーザがソフトウェアのモデルを容易に理解できるようになり，ユーザがそのソフトウェアの持つ機能および特性を十分に活用可能になることが期待される．すなわち，モバイルエージェントのモデルは，ソフトウェアの設計開発者および利用者の双方に利益をもたらす可能性を持っている．

ソフトウェアの運用モデルとしての利点 モバイルエージェントをソフトウェアの運用モデルとして用いる利点は，ソフトウェアの運用者（所有者）とそのソフトウェアを動作させるハードウェアの運用者（所有者）を分離して扱うことが可能な点である．ソフトウェアの運用者は，ハードウェアの運用者に制約されることなく，そのハードウェアで運用するソフトウェアの決定，変更，導入および削除を行うことが原理的に可能となる．すなわち，他の主体が運用するハードウェア上（たとえば，インターネット上のサービスプロバイダが持つサーバ上）で，ソフトウェア運用者（たとえば，小規模な企業や個人）の必要性にあわせた仕様を持つソフトウェアを，ソフトウェア運用者自身の手で運用可能となる．さらに，そのソフトウェアの運用方法をそのソフトウェア自身が管理することが可能となる．これまででも，例えばユーザレベルでインターネットサービスプロバイダが提供する

[1] ここでいう品質とは，仕様と実装との差分（バグ）が少ないことを意味する．

ディスクスペースにPerl等で記述したCGIプログラムをアップロードして動作させることが可能な場合もあったが，セキュリティ機構などソフトウェアの高度な運用を実現するための基盤が十分に提供されていなかったため，ユーザが行えることは限定的なものであった．モバイルエージェントは，ソフトウェアの高度な運用（例えば，状況に応じて複数の実行環境を使い分けるなど）のソフトウェア自身による制御を実現するための，基盤となるモデルを提供する．

モバイルエージェントに基づくソフトウェアの運用が実現されると，計算機資源の提供（あるいは販売）が容易になり，そこに新たな市場が形成されることが期待される．この市場を通じて，例えば，余ったCPUパワーをユーザが他へ売却したり，あるいは企業が高度な計算を行うためのCPUパワーを多くのユーザから購入することが可能となる．計算機資源が流動的に扱われることにより，計算機資源の効果的な運用が可能となるため，必要な計算機資源をより安価に入手できるようになることが期待される．

モバイルエージェントの具体的な利点 モバイルエージェントの利点については，文献[85]および文献[111]が詳しい．ここでは，モバイルエージェントの利点を簡単にまとめる．

分散プログラミングの容易化 サーバ／クライアント方式のプログラム設計の場合に必要な通信プロトコルの決定および通信プログラムの記述の必要がないため，プログラミングが容易になる．

通信遅延への対処 エージェントが通信先との通信遅延が小さい場所へ移動することにより，通信遅延が軽減される．電子商取引のように，通信遅延が結果に大きな影響を与える分野において，効果が期待できる．

通信路の切断への対処 コストや通信路の特性から通信路の切断を余儀なくされる場合がある．このような場合に，エージェントを切断される通信路の向こうに移動させておくことにより，通信路が切断された状態でもエージェントの実行を継続することができる．

ソフトウェアの保守管理の容易化 計算機には，モバイルエージェントプラットフォームのみを事前にインストールしておけばよく，計算機ごとにソフトウェアのインストールおよびアップグレード作業を行う必要がないため，保守管理が容易となる．

ソフトウェアの柔軟な運用 計算機のメンテナンスや使用環境の変化にあわせて，ソフトウェアを実行させたまま，他の計算機上へ移動させることができる．

文献[36]では，モバイルエージェントの利点として，サービスのカスタマイズ，運用における柔軟性向上，耐故障性向上，データ管理の柔軟性向上，プロトコルの内包による通信性能と柔軟性の向上を挙げている．モバイルエージェントの応用領域としては，分散情報検索，アクティブドキュメント，遠隔コミュニケーションサービス，遠隔機器制御，ワークフロー管理，アクティブネットワーク，電子商取引を挙げている．文献[82]では，サービスのカスタマイズの観点から分散情報検索への応用可能性について指摘している．文献[111]では，通信遅延特性の向上という観点から電子商取引への応用にモバイルエージェン

トが適していることを指摘している．これ以外に，エンターテイメント分野での利用も期待される．モバイルエージェントを利用したアプリケーションを携帯電話上で利用できるようにするサービスが，すでに開始されている．

2.2.3 セキュリティ

モバイルエージェントを安全に実行するために求められるセキュリティには，i) 通信路の保護，ii) エージェントの保護，iii) 計算機の保護，iv) エージェントの破損や消失への対処の4つがある．

i) 通信路の保護では，移動中のエージェントが悪意ある第三者から盗聴／改ざん／複製されないようにする必要がある．エージェントの移動に用いる通信路にSecure Socket Layer等の暗号化通信路を用いることで，盗聴等を防ぐことができるが，暗号化のオーバヘッドによる性能低下が生じる．状況に応じて通常の通信路と暗号化通信路をどのように使い分けるかが課題となる．

ii) エージェントの保護では，エージェントを悪意ある計算機（ホスト）あるいは他のエージェントからの不正アクセス（解読，改ざん等）から防ぐことを目的とする．他のエージェントからの不正アクセスを防ぐには，モバイルエージェントシステム上においてアクセス制限機構を実装する必要がある．最も多く用いられるのは，Java言語のセキュリティーマネージャを用いる手法である．悪意ある計算機からのエージェントの保護には，難読化を施した特殊なハードウェアを用いない限り実現が困難であるという見解もあるが，エージェントのインタプリタそのものを移動させる方法や，エージェントを複数に分割して難読化する方法が提案されている．多くのモバイルエージェントシステムでは，信頼できる計算機間のみでエージェントを移動可能とすることにより，この問題を回避している．

iii) 計算機の保護では，悪意あるエージェントから移動先の計算機に対する，サービス停止攻撃 (Denial of Service, DoS) あるいは不正アクセスから保護することを目的とする．エージェントからの不正アクセスに対して最も一般的に用いられるの手法は，エージェントの行動が制限されたサンドボックス上でエージェントを実行させる手法である．他のモバイルエージェントシステムでは，信頼できる計算機から移動してきたエージェントをすべて信頼できるエージェントとしている場合もあるが，エージェントがたとえ信頼できるものでも，それが暴走する可能性があるため，運用に注意が必要となる．

iv) エージェントの破損や消失への対処では，エージェントの移動をトランザクション処理として扱う手法 [134]，あるいはエージェントの処理を多重実行して結果を照合する手法が考えられる．これらの手法は記憶容量や実行速度とのトレードオフにあるため，エージェントの用途やエージェントを動作させる環境に応じてこれらの手法を使い分ける必要がある．

2.2.4 コードモビリティ

モビリティを実現するためには，エージェント移動前の実行状態保存，移動先へのエージェントプログラムの転送，およびエージェント移動後の実行状態復元処理が必要となる．モビリティを実現するための処理の実現には複雑なプログラム記述が必要であり，アプリ

ケーションごとにこれらの処理を記述するのはエージェントの開発者にとって大きな負担となる．モビリティに必要な処理の記述を再利用可能とし，モビリティをプログラミング言語中で直接利用可能にする技術として，コードモビリティが開発された．

コードモビリティを，移動可能な部分をもとに分類する（図2.1）．コードモビリティの分類についてはいくつかのものがあるが，ここでは文献[36]による．前提として，モバイルエージェントは，コード領域，データ領域および，実行状態領域から構成されることとする．コード領域には，エージェントの振る舞いを決定するプログラムが格納される．データ領域には，エージェントの持つデータあるいはオブジェクトが格納される．実行状態領域には，局所変数や呼び出し関係を保存するスタック，およびプログラムの現在の実行個所を示すプログラムカウンタが格納される．(Java言語等におけるスレッドは，実行状態領域に相当する．) **強モビリティ(Strong mobility)** では，コード領域，データ領域および実行状態領域のすべてが移動される．エージェントは，移動前の状態をそのまま引き継いで実行を継続することができる．**弱モビリティ(Weak mobility)** では，コード領域およびデータ領域は移動されるが，実行状態領域は移動されない．エージェントは，移動の直前に実行を継続するためのフラグ等をデータ領域に明示的に格納しておく．[2]

強モビリティは，エージェントの移動を含んだプログラムを自然にかつ簡潔に記述することができる．一方で，既存のプログラミング言語処理系に適用することが難しく，実行速度を向上させることが難しいという課題がある．弱モビリティは，プログラムの記述がやや煩雑になるが，機構が単純であるため既存のプログラミング言語処理系上に実現することが容易であり，また実行状態領域を移動しないためエージェントの移動性能の向上，すなわち移動速度の高速化および移動に必要な通信データ量の削減が期待できる．

次に，コード領域，データ領域および実行状態領域それぞれについて，コードモビリティの実現方法の詳細を示す．

コード領域： コード領域を移動させるためには，i) そのエージェントによってどのコードが利用されているかの識別，ii) エージェントのコードの置き場所，および iii) コードの転送時期が重要となる．i) については，言語処理系上に用意されたコードへのアクセスAPI (たとえば，Java言語でのクラスローダ) を利用する方法と，モバイルエージェントシステム内部に言語処理系を実装してしまう方法がある．前者の方法では，利用する言語処理系上においてAPIが用意されている必要があるが，新たな言語処理系を実装する場合と比較して容易に実現可能であり，機構も簡単となる．後者の方法では，強モビリティを実現するなど他の目的から言語処理系をすでに再構成している場合に有効であり，場合によっては前者の方法よりも高速なコードの識別や，コード表現の縮小化を行うことが可能となる． ii) では，コードを中央のサーバ上に登録しておき，そこからダウンロードする方法と，移動元のホストから直接ダウンロードする方法がある．前者の方法では移動先のホストが移動時にコードサーバ上にアクセス可能である必要があるが，コードの一貫性保持やコードの署名等のセキュリティ上の課題を解決しやすい．後者の方法では，中央にサーバを用意する必要がなく，移動時にも移動元のホストにさえアクセスできればコード領域の移動が可能となる．iii) では，移動時に必要なコードを一括して転送する方法と，移動後に必要なコードを随時ダウンロードする方法がある．前者では，事前に必要なコードをすべ

[2] ここで示したコードモビリティの実装モデル以外に，ダウンローダブルソフトウェア (Downloadable Software) あるいは遠隔実行 (Remote Evaluation) がある [36].

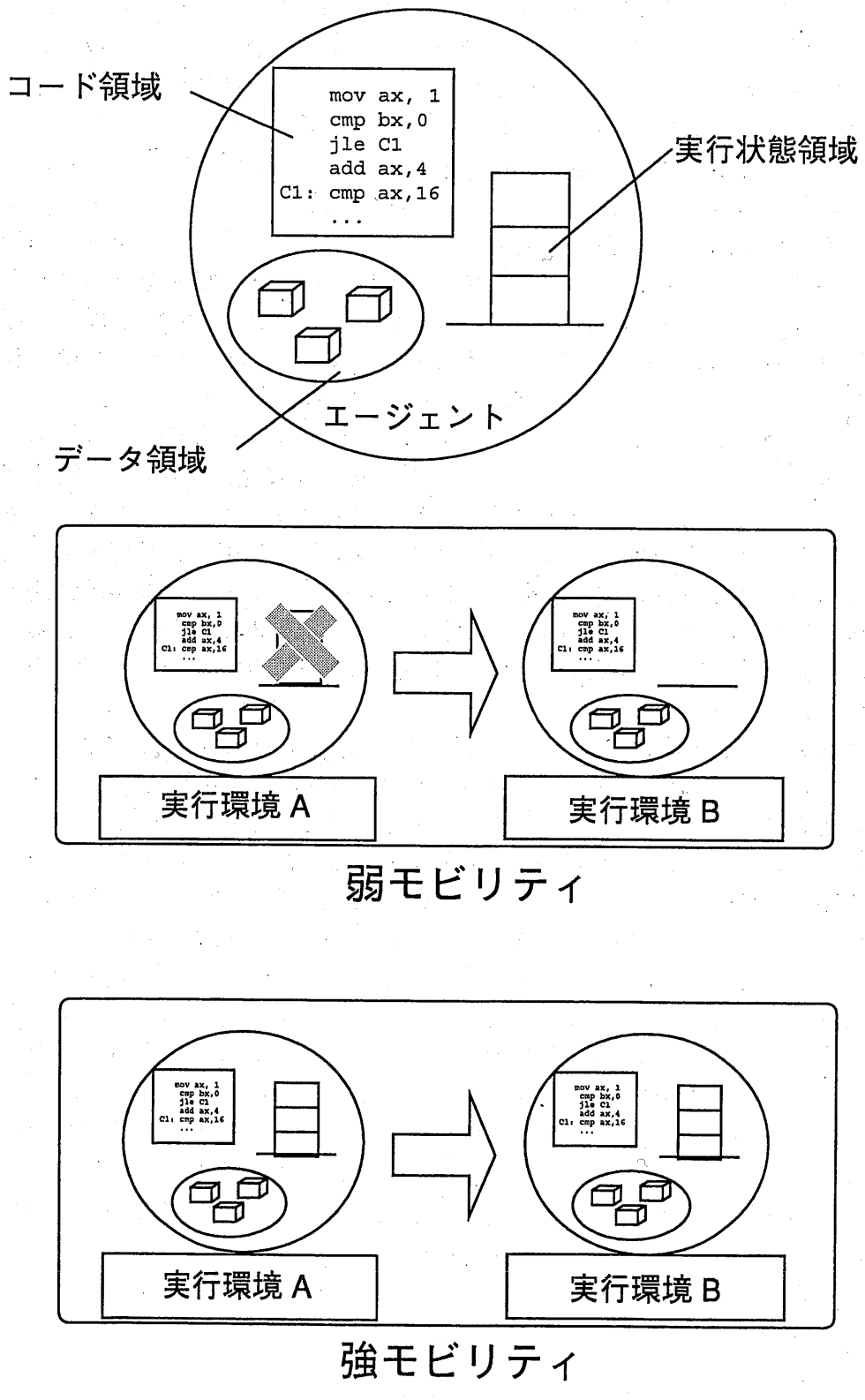

図 2.1: モビリティの分類

て列挙できるようにしておく必要があるが，移動後に移動元ホストとの通信を切断することができ，通信遅延の影響も受けにくい．後者では，必要なコードのみをダウンロードするため，あまり利用しないコードの転送を抑制でき，必要となるコードを事前に列挙しておく必要も省かれる．

データ領域：データ領域の移動では，データ領域の取り込みにシリアライズ（あるいはマーシャリング）機能を用いることが一般的である．データ領域の移動で課題となるのは，データ領域中にファイルハンドルなどの本質的に移動できないオブジェクトが存在する場合，およびオブジェクトが複数のエージェントによって共有される場合である．ファイルハンドル等の移動不可能なオブジェクトの扱いに関しては，移動先で同等のオブジェクトを新たに生成する方法，および移動元のオブジェクトへのリファレンスを移動して，移動先から遠隔操作する方法がある．前者の方法はユーザインタフェース等に対して有効であり，後者の方法はファイルアクセス制御等に対して有効である．共有されたオブジェクトの扱いに関しては，移動元のオブジェクトへのリファレンスを持ったエージェントが移動する方法，オブジェクト自身を持ってエージェントが移動するが他のエージェントにはその移動後のオブジェクトへのリファレンスを渡す方法，およびオブジェクトのコピーを移動先に生成する方法がある．どの方法が最適化はアプリケーションに依存しており，アプリケーションが行う処理の性質を考慮して設計者が適切な方法を選択する．

実行状態領域：Java に代表される既存のプログラミング言語処理系の多くは，実行状態領域を参照／改変するための API が，性能上やセキュリティ上の理由から用意されていない．このため，実行状態領域の移動では，実行状態領域上のデータをどのように取り込み復元するかが課題となる．実行状態領域の移動機構の実装には，A) 実行状態領域にアクセス可能なプログラミング言語処理系をモバイルエージェントシステム上に実装する手法と，B) 既存のプログラミング言語処理系を改変して，実行状態を参照／改変するための API を拡張する手法がある．B) の手法には，B1) 実行系自身に直接手を加える手法，B2) 処理系と上位互換性のある実行系を新たに実装する手法，B3) プログラム変換による手法がある．手法 A) では，モバイルエージェントの特性を踏まえて大胆な設計を行うことができ，ネットワーク透過性を持つ入出力ライブラリや，資源割り当て機構の提供が行いやすい．手法 B) では，既存のプログラミング言語処理系を基にしているので，過去のプログラム資産の再利用が行いやすい．手法 B1) では，基となる処理系上の特長（マルチプラットフォーム性，セキュリティ，実行速度）の一部が犠牲となるが，基となる処理系を新たに構築するよりも開発が容易である．手法 B2) では，開発の負担が大きく，動作する環境も限られるが，既存のプログラムのバイナリコードがそのまま利用できるため，既存のプログラミング資産の再利用が容易である．手法 B3) では，変換後のコードの肥大化や実行効率面での課題があるが，機構が比較的単純であり，既存の実行系がそのまま利用できる．

実行状態領域の移動に関するこれまでの研究は，移動の実現が目的となっている場合がほとんどである．しかし，実際には多くの処理系において，実行状態領域は実行速度の高速化に偏った最適化がされており，移動の効率を考慮した実行状態領域の設計など，移動の効率化に関する議論はまだ十分ではない．

2.3 モバイルエージェントシステム

2.3.1 モバイルエージェントシステムの利点

モバイルエージェントシステムは，コードモビリティを用いて記述したソフトウェアを実際に実行可能とするための環境を提供するシステムである[3]．モバイルエージェントシステムは，モバイルエージェントが活動するために必ず必要となるものではない．例えば，コンピュータウィルスの多くは，オペレーティングシステムや特定のアプリケーションソフトウェアの機能[4]（さらに，必要であれば人間の手）を利用して自身を他の計算機へ拡散させ増殖することが可能である．モバイルエージェントも，オペレーティングシステムの機能を利用して自身を移動させるためのモビリティ機構をエージェントプログラム自身に埋め込むことにより，特別な実行環境なしに他の計算機へ移動することが可能となる．たとえモバイルエージェントの実装にコードモビリティを用いていたとしても，エージェントプログラムをコンパイルしてコードモビリティをエージェントに内包させることにより，モバイルエージェントシステムなしでモバイルエージェントを運用することは可能である．

モバイルエージェントシステムを用いる利点は，モビリティの実現に必要なモビリティの実装をエージェント自身のコードから切り離し，モビリティとモバイルエージェントを独立して設計および運用を行うことが可能となる点である．モビリティの実装をモバイルエージェント自身から独立させることにより，エージェント開発者はモビリティの詳細な実装から開放され，モバイルエージェント自身の開発に専念できる．モビリティの実装をモバイルエージェント間で共有することにより，モバイルエージェントの運用に必要な資源を効率的に利用可能となるという利点と，モバイルエージェントの移動時に移動元から移動先へ転送する情報からモビリティの実装を省略することによるエージェントの移動の効率化が可能となるという利点が得られる．

モバイルエージェントシステムを用いるもう1つの利点は，オペレーティングシステムでは用意されないセキュリティ機構や互換性を向上させる機構などを，モバイルエージェントシステム上に追加可能な点である．モバイルエージェントシステムに，オペレーティングシステム上では用意されないセキュリティ機構や資源割り当て機構を実装することにより，モバイルエージェントの運用のために十分な機能を持たないオペレーティングシステム上であっても，モバイルエージェントを運用可能となる．モバイルエージェントシステムにオペレーティングシステムや計算機を抽象化する役割を持たせることにより，オペレーティングシステムやプロセッサのアーキテクチャ等の差異をモバイルエージェントシステムに吸収させることが可能となる．これにより，異なるオペレーティングシステムや異なるアーキテクチャのプロセッサを持つ計算機間を移動しながら動作するモバイルエージェントを統一された設計方法で実現可能となり，エージェントプログラム開発者の負担が軽減される．また，実行環境ごとの差異に対応するための実装の多くの部分をモバイルエージェントシステム自身に持たせることにより，エージェントの移動時にこれらの実装の移送を省略することが可能となるため，エージェント移動のオーバヘッド低減，および移動のための通信データ量の削減が可能となる．

---

[3] 本論文では，モバイルエージェントシステムとは，モバイルエージェントを用いて実現されたシステムではなく，そのようなシステムはモバイルエージェントに基づくシステムとして区別される．

[4] これは，機能というよりもバグと呼ぶほうが適切かもしれない．

実際には，モバイルエージェントの開発および運用には，ほとんどの場合にモバイルエージェントシステムが用いられる．これは，ほとんどの既存のオペレーティングシステムがモバイルエージェントの運用に十分な機能を提供しておらず，モバイルエージェントシステムを用いることによる他の利点も十分大きいからであると考えられる．

2.3.2 モバイルエージェントシステムの機能と性質

モバイルエージェントシステムには，上述の2つの利点を実際にユーザや開発者が享受可能なものにするために，次の5つの性質あるいは機能の実現が求められる．

1) コードモビリティ：コードモビリティの実現には，モバイルエージェントのプログラムおよび実行状態を移動元で取り込み，移動先に転送し，復元する機能が必要となる．エージェントのどの部分を移動させることができるか，どのような方法でいつ転送するか，および移動させる部分の情報をどのように取り込むかなどの違いから，多種のモバイルエージェントシステムが試作および実装されている．

2) セキュリティ：モバイルエージェントに求められるセキュリティには，通信路の保護，エージェントの保護，計算機の保護，エージェントの破損や消失への対処がある．必要とされるセキュリティの強度に応じて，様々な種類の実装が行われている．

3) マルチプラットフォーム性：モバイルエージェントはあらゆる計算機上で実行できることが望ましい．また，過去の資産の利用およびプログラマの教育コストの観点から，既存のプログラミング言語との親和性が高いことが望まれる．モバイルエージェントシステムの提供方法に応じて，様々な種類の実装が行われている．

4) ネーミングサービス：エージェントは場合によっては不特定多数の計算機間を移動するため，エージェントを一意に特定するための機構がプログラミングを行う上で必要となる．

5) 資源割り当て：不特定多数のエージェントが計算機を利用する場合，CPU資源，記憶領域，およびその計算機上でのローカルなサービス（ファイル入出力など）を適切にその計算機上のエージェントに割り当てる必要がある．

これら1)から5)以外に，コードモビリティを拡張したものとして**エージェントの永続化**，あるいはセキュリティと資源割り当てを拡張した**エージェントの寿命**の概念が実装される場合もある．

2.3.3 Telescript

Telescript[131]は，最も初期の段階に開発されたモバイルエージェントシステムである．Telescriptは初期に開発されたシステムであるが，アカデミックな研究としてではなく，商用のソフトウェアとして開発された．Telescriptでは，コードモビリティ，セキュリティ，エージェントの寿命の概念が実装されている．Telescriptプログラムは，C++に似たオブジェクト指向言語(High Telescript)で記述され，より低レベルな言語(Low Telescript)にコンパイルされる．Low Telescriptプログラムは，Telescript Engineと呼ばれるインタプリタ上で実行される．Telescriptが実現するコードモビリティは強モビリティであり，プログラムの実行状態は移動に伴って保存される．Telescriptにおける強モビリティは，実行状

態を保存可能とした独自のインタプリタによって実現される．Telescriptエージェントは，プレース(Place)と呼ばれる実行環境で動作する．プレースは1台の計算機上に複数存在することができ，内部にミーティングプレースなどの別プレースを入れ子構造で含むことができる．Telescriptエージェントおよびプレースにはテレネーム(Telename)という識別子が与えられる．テレネームは，その所有者を識別するためのオーソリティ(Authority)と，そのエージェントあるいはプレースそのものを識別するためのアイデンティティ(Identity)から構成される．Telescriptエージェントには，リソースを使用するための権利（パーミット）が与えられ，その範囲でのみリソースを使用できる．Telescriptでは，エージェントのマイグレーションメソッドgoを含むプログラムを次のように記述する．

```
1  SimpleAgent : class ( Agent, MeetingAgent ) = (
2    public
3      initialize: op(key: copied String; val: copied Telename; Permit;
4                     Permit|nill; Integer|Nil ) = {^();};
5      live: sponsored op(Exception|Nil) = {
6        aPlace: Telename;
7        aPlace = here@LookupPlace.returnValue("place1");
8        try {
9          self.go(Ticket(aPlace));
10       } catch error: Exception {
11         "Can't go!".dump;
12       };
13     };
14 );
```

左側の数字はここでの説明のためにつけた行番号であり，実際のプログラムでは入力しない．プログラムの1行目で，SimpleAgentクラスの定義を宣言している．このクラスでは，initializeメソッド（3行目から4行目）およびliveメソッド（5行目から13行目）を定義している．最初に，インスタンスオブジェクト生成時にinitializeメソッドが呼ばれ，次にliveメソッドが呼ばれる．liveメソッド中では，まず移動先のテレネームを取得し（7行目），そのプレースへの移動を試みる（9行目）．移動に失敗した場合には，Can't Go!が出力される（11行目）．

Telescriptは非常に先進的な概念を導入したシステムであり，学ぶべきところが多い．Telescriptがモバイルエージェントを運用するために必要となる機能および概念を提案しているのに対して，本研究は記号処理に基づくエージェントの記述およびアプリケーション開発のためのフレームワークの提案に焦点を当てている点で異なる．

2.3.4 Grayらの D'Agents

D'Agents(旧名 AgentTcl)[59]は，Dartmouth大学のGrayらによって開発されたモバイルエージェントシステムである．D'Agentsシステムの特長は，エージェントの記述言語とエージェントマイグレーションエンジンが独立しており，それぞれ異なる言語で記述さ

れたエージェントを1つのマイグレーションエンジンで移動させることが可能な点ある．エージェントの記述には，JAVA, SafeTcl, Scheme を用いることが可能である．エージェントは，それぞれの言語のインタプリタ上で実行される．D'Agents システムが提供するコードモビリティは，強モビリティであり，エージェントの実行状態は移動先で復元される．JAVA における強モビリティの実装は，JAVA 仮想機械を独自に拡張する方法で実現される．D'Agents システムでは，エージェント間通信を実現するための機構，および階層的なネームサーバ機能を実現している．D'Agents システムでの，Tcl によるモバイルエージェントの簡単な記述例は次のようなる．

```
1  agent_begin  # register with the local agent server
2
3  set output {}
4  set machineList {place1 place2}
5
6  foreach machine $machineList {
7     agent_jump $machine          # jump to each machine
8     append output [exec who]     # any local processing
9  }
10
11 agent_jump $agent(home)         # jump to home
12
13 #display results
14
15 agent_end
```

ここで，左側の数字はここでの説明のためにつけた行番号であり，実際のプログラムでは入力しない．エージェントの処理は agent_begin（1行目）から agent_end（15行目）の間に書く．このプログラムでは，machineList（4行目）に含まれる計算機上を順に agent_jump コマンド（7行目）を用いて移動し，それぞれの移動先で who コマンドを実行（8行目）し，ホームに戻って（11行目）結果を表示する．

D'Agents は，言語のインタプリタ全体をマイグレーションさせるアプローチを用いた最初のモバイルエージェントシステムである点で興味深い．各インタプリタが1つのプロセスで表現されることから，複数のエージェントが動作した際にシステムに対する負荷が高くなる傾向がある点，および同一計算機上に存在する他のエージェントとの通信がプロセス間通信となるため通信オーバヘッドが大きい点が課題となる．D'agents が複数の言語によるエージェントの記述および実行時のセキュリティに焦点を当てているのに対し，本研究は協調的なエージェントの記述および運用時のマルチプラットフォーム性の実現に焦点を当てている点で異なる．

2.3.5 IBM Aglets

IBMで開発されたAglets[84]は，JAVAベースのモバイルエージェントシステムである．Agletsでは，エージェントをJAVA言語で記述し，コードモビリティとして弱モビリティが提供される．Agletsは，実行環境がJAVA上で動作可能なため，非常に広範囲のオペレーティングシステム上で動作可能となっている．Agletsでは，弱モビリティおよびセキュリティ機構の実装に，JAVA処理系が持つ動的クラスロード機能を利用している．Agletsでは，エージェント間の通信機構を備えるが，Telescript等に見られるような名前管理機構は標準では用意されない．Agletsエージェントは，Tahitiエージェントランチャ上からロードされ，Dispatchコマンドにより実行が開始される．簡単なモバイルエージェントのAgletsによる記述例は，次のようになる．

```
1  public class Test1 extends Aglet {
2    URL home,dest;
3
4    SimpleMobilityAdapter ma;
5    int counter;
6
7    public void onCreation(Object init) {
8      try{
9        dest = new URL("atp://host1:434/");
10       home = new URL("atp://host2:10000/");
11     } catch( MalformedURLException me ) {
12       System.out.println( me );
13     }
14     ma = new SimpleMobilityAdapter(this);
15     addMobilityListener( ma );
16     counter = 2;
17     goThere(dest);
18   }
19   public void run() {}
20   public void onDisposing() {}
21
22   void goThere(URL url) {
23     try{
24       dispatch(url);
25     } catch( RequestRefusedException rre ) {
26       System.out.println(rre);
27     } catch( IOException ioe ) {
28       System.out.println(ioe);
29     }
30   }
```

```
31
32   public void onArrival() {
33     counter--;
34     System.out.println(counter);
35     if( counter == 1 ) {
36       goThere(home);
37     }
38   }
39 }
40
41 class SimpleMobilityAdapter extends MobilityAdapter {
42   Test1 aglet;
43
44   public SimpleMobilityAdapter(Test1 aglet) {
45     this.aglet = aglet;
46   }
47
48   public void onDispatching() {
49   }
50
51   public void onArrival(MobilityEvent e) {
52     aglet.onArrival();
53   }
54 }
```

ここで，左側の数字はここでの説明のためにつけた行番号であり，実際のプログラムでは入力しない．Agletsのエージェントプログラムは，この例では2つのJAVAクラスから構成される．プログラム中の1行目から39行目までがエージェント本体であり，41行目から54行目までが，イベント受信部分のプログラムである．エージェントが生成されたときに，onCreationメソッドが呼ばれる．ここで示している例では，onCreationメソッド内で，移動先の定義（8行目から13行目），イベントリスナの登録（14，15行目），初期値の設定（16行目），および移動の開始（17行目）を順に行っている．エージェントのマイグレーション（22行目から30行目）では，dispatchメソッドによりエージェントのマイグレーション開始を指示する．

Agletsのエージェントプログラミングでは，提供されるモビリティの種類が弱モビリティであるため，移動後にプログラムの実行状態は完全には再現されない．移動後のエージェントの制御を行うために，エージェントが移動先に到着したときに，イベントonArrivalが発生するようになっている．イベントの発生によりTest1クラスのonArrivalメソッド（32行目から39行目）が呼ばれる．移動後の状態を制御するために，この例ではフラグcounterを用いている．ここでは，フラグcounterの値が1の時にエージェントが移動先にいることを示している．エージェントが移動先に到着した場合には，元の場所に戻る（35行目から37行目）．

Agletsでは，扱いやすいJAVA言語をエージェントプログラミングに利用できる点が利点である．Agletsは汎用性および実用性を重視して，モビリティの実装を弱モビリティにとどめているのに対して，本研究はモビリティの簡潔な記述方法の実現に焦点を当てている点で異なる．

2.3.6 TOSHIBA Plangent

TOSHIBAで開発されたPlangent[98]は，プランニング機能を備えたモバイルエージェントを実現するシステムである．Plangentシステムでは，エージェントの知的な振る舞いの実現を容易とするために，プランニング機構をエージェント自身に組み込んでいる．Plangentシステムは JAVA言語上に実装されるが，エージェントの記述には，主にPlangent独自の言語により記述される．エージェントの記述は，プラン生成のためのルールの記述（アクション定義），プランで行う手続き的操作の記述（スクリプト定義），およびスクリプト定義で用いる機能を拡張するためのJAVAプログラムの記述（拡張コマンド定義）の3つから構成される．Plangentにおけるモバイルエージェントは，最初にアクション定義からプランニングを行い，次に生成したプランに基づいてスクリプト定義による動作を行う．Plangentにおけるアクション定義の記述例を示す．

```
1  action(_,_,_,
2         [say_hello(N,Message)],
3         [sayHello(Nodes,Message)],
4         [sayHello([N|Nodes],Message)],
5         []
6  ).
7  action(_,_,_,
8         [],
9         [],
10        [sayHello([],Message)],
11        []
12 ).
```

ここで，各行の左側の番号はここでの説明のために付けた行番号であり実際のプログラムには記述しない．ここで示すアクション定義は，2つの節（1行目から6行目，および7行目から12行目）からなる．最初の節では，アクション定義は，実行するアクション（2行目），事前条件（3行目），事後条件（4行目）から構成されている．最初の節は，sayHello([N|Nodes], Message) という目標を達成したいときは（4行目），say_hello(N, Message) というスクリプト定義を実行し（2行目），かつ sayHello(Nodes, Message) という条件を満たせればいい（3行目）と解釈する．Plangentにおけるスクリプト定義の記述例を示す．

```
1  say_hello($node,$message) {
2    goto($node);
```

```
3     print($message);
4  }
```

ここで，各行の左側の番号はここでの説明のために付けた行番号であり実際のプログラムには記述しない．ここで示すスクリプト定義は，コマンド宣言部（1行目および4行目）と組み込みコマンド等による手続き的操作の記述（2行目および3行目）から構成される．コマンドの宣言は，コマンド名と引数から構成される（1行目）．ここでは，$で始まる単語が変数を示している[5]．コマンドgotoおよびprintはシステムに組み込みのコマンド(組み込みコマンド）であり，スクリプト定義中で利用可能である．コマンドgotoは引数に指定された実行環境（ノード）への移動を行う．コマンドprintは，引数の内容を出力する．

Plangentでは，エージェントを知的に振舞わせるためにプランニング機能を備えている点で他のモバイルエージェントシステムと大きく異なっており，知的モバイルエージェントの実現に関して非常に重要な研究である．Plangentではフレームワークとしてプランの記述方法を1つ示しているが，本研究は様々なプランニング機能を実現するためにプランナ自身の記述を行えるようなモバイルエージェントシステムを目指している点で異なる．

2.3.7 SuriらのNOMADS

Suriらによって開発されたNOMADS[121]は，JAVAに基づくモバイルエージェントシステムである．NOMADSの特長は，JAVAによるエージェントの記述で強モビリティを実現する点，および各エージェントごとのリソース管理機構を実行環境レベルで実現する点である．NOMADSでは，強モビリティを実現するために，新たにJAVA互換の仮想機械（Aroma VM）を構築している．AromaVMは，スレッドの実行状態を取り込む機能を備える．実行環境（Oasis）上で複数のAroma VMが動作し，それぞれのAroma VM上でエージェントが動作する．NOMADSにおけるエージェントの記述例を示す．

```
1  public class sampleAgent extends Agent {
2    public static void main(String args[]) {
3      for( int i = 0; i < args.length; i++ ) {
4        go(args[i]);
5        System.out.println("Hello");
6      }
7    }
8  }
```

ここで，各行の左側の番号はここでの説明のために付けた行番号であり，実際のプログラムでは記述しない．NOMADSにおけるエージェントでは，JAVAプログラムの文法を用いてエージェントを記述する．エージェントの移動には，goメソッドを用いる．NOMADSでは強モビリティを実現するため，IBM Agletsのように移動後に動作を復元するためのフ

[5] Plangentでは，変数の表記をスクリプト言語Perlに似た文法で記述する．変数を示す接頭語には$以外に@等が使用可能である

ラグ等を用意する必要はなく，エージェントの移動後はgoメソッド実行後の状態から動作が再開される．

NOMADSにおけるエージェント実行環境 Oasis では，エージェントごとの資源割り当てを制御する機構を持つ．例えば，エージェントが使用可能なディスク容量，あるいはネットワーク通信における利用可能な帯域幅を，エージェントごとに制限する機能を持つ．

NOMADS では，記述言語がJAVA言語と互換性を持つため，プログラミングが行いやすい点が利点となる．NOMADS では，エージェントの実行にJAVA言語の実行環境とは異なる専用の実行環境を必要とするため，利用できる環境が制限される点，HotSpot等の高速化手法が適用されないことによる低速な実行速度，およびGUI等のアプリケーションプログラミングに必要なライブラリの整備が十分でない点が課題となる．

2.3.8 その他のモバイルエージェントシステム

佐藤の Mobile Spaces[109] は，JAVA に基づくモバイルエージェントシステムである．Mobile Spaces では，エージェントの内部にモビリティ機構を入れ子構造で持たせることが可能となっている．エージェントの内部にモビリティ機構を持たせることにより，エージェント内部の特定の部分のみを選択的に移動させることが可能となる．

MITSUBISHI Concordia[134] は，JAVA に基づく商用モバイルエージェントシステムである．Concordia では，エージェント移動時のセキュリティと耐故障性に着目し，エージェント移動を双方向コミットメント法によるトランザクション処理として実現している．

Mobile Spaces および Concordia は，Aglets と同様に弱モビリティを実現しており，Aglets と同様に知的モバイルエージェントの記述ではモビリティの記述力不足が課題となる．

TOSHIBA Bee-gent[77, 78] では，モバイルエージェントにおけるプロトコルの隠蔽に着目し，既存のシステム間の協調機構としてモバイルエージェントを利用する手法を提案している．Bee-gent では，エージェントの振る舞いの記述に状態と状態間の遷移を用いている．一般に知的エージェントの振る舞いは少数の状態とその間の遷移として定式化することが困難であるため，本研究で扱う知的エージェントの構築には適さないと考えられる．

モバイルエージェントシステムとしては，前述のもの以外に，Mobidget[37], MOBA[118], Pathwalker[38], Gossip[127], Jumon[81], Voyager[97], TACOMA[75], ARA[66], Obliq[24], muCode[102], Messanger[16, 133], Mole[10], Correlate[126] が提案されているが，いずれも本研究で扱う知的エージェントの構築を意図したものではない．

2.3.9 分散コンピューティングと分散オブジェクト

分散コンピューティングの分野では，遠隔での手続き呼び出しとして RPC (Remote Procedure Call) が提案されている．RPC では，スタブあるいは動的手続き呼び出しを用いて，遠隔での手続き呼び出し機構を実現している．RPC の代表的な実装に，SUN の ONC がある [76]．RPC をさらに高度に発展させた技術に，ORB(Object Remote Brokering) がある．ORB は，従来個別の計算機環境で用いられていた RPC をオブジェクト技術を用いて統合化し，異機種計算機間での相互運用を可能とした通信機構である [76]．ORB の実装には，OMG による CORBA[99]，Microsoft による COM があり，JAVA 上の ORB の実

装としては，平野らのHORB[65]，およびJava RMIがある．ORBの発展的規格として，オブジェクトの内部に計算過程を含めるための仕組みが検討されており，CORBAに対するMAF(Mobile Agent Facility)の追加[25]，およびFIPA (Foundation of Intelligent adn Physical Agents)におけるモバイルエージェントの規格化[35]が行われている．これらの試みは，いずれも分散環境における位置透過性の実現の延長としてモバイルエージェントを捉えており，ここで扱われるモバイルエージェントの定義は，単にコードモビリティを持つオブジェクトであり，本論文で議論するモバイルエージェントとは本質的に異なるものである．

Java AppletおよびJava Servletは，アプリケーションソフトウェアを動的にダウンロードして利用するためのフレームワークである．Java Appletでは，アプリケーションはサーバ側からユーザ側にダウンロードされ，Java Servletではサーバ間でのアプリケーションのダウンロードが行われる．Java AppletおよびJava ServletがORBと異なる点は，その実装がインターネット等の大規模ネットワーク上での運用を前提としている点である．アプリケーション運用の柔軟性を向上させている点でモバイルエージェントと類似している．Java AppletおよびJava Servletでのアプリケーション設計方針は，分散環境における位置透過性の実現の延長として実現されるため，本論文で議論するモバイルエージェントの設計上の利点は得られにくい．また，コードモビリティとこれらを比較した場合，コードモビリティではコード自身に移動の主体があるのに対し，Java AppletおよびJava Servletではコードは外部からの要求があった場合に受動的にダウンロードされ，初期状態から実行を開始される点が異なる．すなわち，Java AppletおよびJava Servletは，それ自身がエージェントのモビリティを実現するものではない．

オブジェクトへの遠隔アクセスを提供するフレームワークに，SOAP (Simple Object Access Protocol) [18]に基づくフレームワークがある．SOAPでは，CORBAのような複雑な構造を持たず，単純なオブジェクトへのアクセスのみがプロトコルとして規定される[112]．SOAPでは，インターネット標準 (HTTP, XML)に基づいており，インターネット等の広範囲な分散環境での適用が期待されている点で，モバイルエージェントと類似している．SOAPでは，単純なオブジェクト間のアクセスを提供することに焦点を当てており，コードモビリティの実現は意図していない．

2.4 論理型言語処理系の実装技術

2.4.1 用語の定義

論理型言語は，一階述語論理を計算モデルの基礎とするプログラミング言語である[22]．論理型言語の最も一般的なものに，Prologがある．Prologに関する詳細は，文献[119]が詳しい．論理型言語には，Prolog以外に制約論理型言語，並列論理型言語，帰納論理型言語，線形論理型言語等がある．本節では，Prologを中心とした論理型言語とその実装に関連する基本的な概念を説明し，用語の定義を行う．

プログラムはホーン節 (Horn clause)によって表現される．ホーン節は単に節とも呼ばれる．ホーン節は，`h :- b1,...,bx.` という形式で表現される．ここで，`h` を節のヘッド **(head)**, `b1,...,bx` をボディ **(body)**と呼ぶ．ホーン節の各要素 `h`, `b1`, ..., `bx` はそれ

ぞれ原始論理式であるが，便宜的に述語 (predicate) 呼ぶこともある．述語 h(x,y) について，h を**述語名**，x および y を**引数**，引数の数を**アリティ** (arity) (この場合のアリティは2) と呼ぶ．引数のない述語は，引数を示す括弧を省略して記述する．例えば，h. のように記述する．述語を表現する場合に，述語名とアリティをあわせて**述語名/アリティ**と表記する場合がある．たとえば，h(x,y) の場合，h/2 と表記される．述語の引数には，述語と同様の複数の引数を持つ関数表現を含めることができる．引数を持った述語のように，名前と複数の引数を括弧でかこんだ形式で表現される関数を**関数子** (functor) と呼ぶ．関数子はその引数にも関数子を含めることが可能である．述語の引数となっている関数子を特別に構造と呼ぶ場合がある．記号 ':-' は左向きの矢印記号を意味し，その形からめだかと読む．ボディが空でヘッドだけからなる節を，**ファクト** (fact) と呼び，h(x,y). のように記述する．プログラムは，**節データベース**に格納される．節データベースへの節の挿入操作を**アサート** (assert)，節の削除操作を**リトラクト** (retract) と呼ぶ．プログラムの節データベースへの読み込み操作を**コンサルト** (consult) および**リコンサルト** (reconsult) と呼ぶ．リコンサルトの場合は，以前に同じ述語名を持つ節定義が上書きされる．プログラムの実行は，節データベースに対する**問い合わせ**として与えられる．節データベースに対する問い合わせは，**クエリ** (query) と呼ばれることもある．プログラムの読み込み時に特定の問い合わせを実行する方法として，ヘッドのない節を記述する方法があり，これを**ディレクティブ節** (directive clause)，あるいは**コマンド** (command) と呼ぶ．例えば，':-do(Something).' のように記述する．問い合わせの実行のためにホーン節のヘッド部との単一化が試みられるとき，このホーン節が**呼ばれた**，**呼び出された**，あるいは**実行された**という．このホーン節のボディ要素すべてが呼び出され，妥当な変数の値が求められた場合に，この述語の実行は**成功した**といい，そうでない場合を**失敗した**という．**変数**は，英大文字あるいは_(アンダースコア) から始まる文字列として表記される．変数に値を代入する操作を**束縛する** (bind) と呼ぶ．変数は，それが用いられるホーン節内でのみ有効であり，同名の変数は同一の値をとる．節定義で用いられる変数はすべて局所変数である．大域変数に相当する機能の実現は，その変数の値を節データベースにアサートすることで代用する．すなわち，論理型言語では，大域変数という種類の変数はなく，すべての変数が局所変数である．変数をまったく含まない節のことを，**基底節** (ground clause) と呼ぶことがある．変数は，それが用いられるホーン節が呼び出されるごとに，別の内容を示す変数であることを明確にする目的で変数名が付け替えられる．ホーン節内の変数の名前を別の名前に置き換える操作を，**変数のリネーミング**という．たとえば，f(X,Y) :- g(X), h(Y). という節があったとき，述語 h/2 が実際に呼び出される際には，f(_1,_2) :- g(_1), h(_2). のように変数名が置き換えられた後，ヘッド部と単一化される．**単一化**とは，2つの述語について，それらが同一の内容となるようなもっとも緩い変数束縛 (Most General Unifier, MGU) を決定する操作のことをいう．変数は，述語の呼び出し時に引数の値を受け取る用途 (**入力**)，および呼び出しの結果を返す用途 (**出力**) の2通りの用途で用いられる．変数が入力と出力のどちらに用いられるかの種別を，その変数の**モード** (mode) と呼ぶ．モードを示す記号を変数の前につけて表記することを，**モード宣言** (mode declaration) と呼ぶ．一般に，入力変数を示すのに'+'，出力変数を示すのに'-' が用いられる．モード宣言は，プログラムの実行の最適化を助けるためにプログラム中に記述されるほか，組み込み述語の定義での変数のモードを説明するためにも用いられる．ここで，**組み込み述語**とは，特

別な入出力等の操作を実現するためにシステムが内部に持つ特別な述語である．述語の呼び出しに失敗した場合に，プログラムを後戻りさせて別の候補を探す動作をバックトラック (backtrack) と呼ぶ．バックトラックのうち，述語の呼び出しに失敗した原因となる節に直接戻る動作を**依存駆動バックトラッキング (dependency-directed backtracking)**，別の候補を持つ節で最も最近に呼び出された節へ戻る動作を**クロノロジカルバックトラッキング (chronological backtracking)** と呼ぶ．論理型言語では，一般にクロノロジカルバックトラッキングが用いられる．述語の実行に関して，現在得られた呼び出し結果（あるいは途中経過）以外に別の候補を持つ場合，その述語の実行箇所を**選択点 (choice point)** と呼び，プログラムの実行時に選択点を記録しておく構造を**選択点スタック (choice point stack)** と呼ぶ．バックトラック時には，バックトラックする箇所以後に行われた変数束縛を元に戻す必要がある．バックトラック時に変数束縛を元に戻す動作を，**変数束縛の解除**，変数束縛を解除するための情報を**変数のトレイル (trail)** と呼ぶ．実行時にトレイルを蓄えておく構造のことを**トレイルスタック (trail stack)** と呼ぶ．変数や定数以外の記号のことを**オペレータ**と呼ぶ．オペレータは，四則演算のような一般的なもの以外に，プログラム中で述語 `op/3` を用いて自由に定義することが可能である．バックトラックの動作を制御するための特別なオペレータとして，**カットオペレータ**があり，`!` と表記する．カットオペレータは，呼び出されたときに，そのカットオペレータが埋め込まれた節のヘッド部の呼び出し以降に記録されたすべての選択点を選択点スタックから削除した後，呼び出しに成功する．たとえば，`f(X) :- g(X,Y), !, h(Y).` `f(X) :- i(X,Y,Z).` と２つの節が定義されていたとき，問い合わせによって１つめの節のカットオペレータが実行されると，その直前の `g(X,Y)` の呼び出しの内部，および２番目の節定義へのバックトラックが行われなくなる．カットオペレータを用いることにより，手続き的な動作を論理プログラムとして記述することが可能となる．

2.4.2 知的エージェント構築における論理型言語の利点

本節では，知的エージェント構築という観点から見た，論理型言語の利点を述べる．

知的エージェントの構築には，エージェントの信念の表現および推論の実現に関する部分（熟考部分）と，ある特定の目標を実現するための手続き的な処理の記述（手続き部分）が必要となる．多くのエージェントフレームワークでは，これら２つの部分の記述を，それぞれの目的に特化した言語によって記述している．複数の言語を用いたプログラミングでは，特に言語間での表現の違いから，データの相互変換を常に意識したプログラミングが必要となる．データの相互変換処理のコーディングは時として，それらのデータに対する処理のコーディングそのものよりも複雑で大きくなってしまう．単一のプログラミング言語によってあらゆる処理が記述できれば，データの相互変換処理の問題が解決される．しかし，JAVA 等の多くのオブジェクト指向言語は手続き型言語に基づいており，記号処理の記述には十分な記述力を持たない．Prolog は，単一化と節表現による強力な記号処理記述能力を持ちながら，カットオペレータを用いることにより手続き的な処理の記述も効果的に行うことが可能である．すなわち，Prolog という単一の言語によって，高度な記号処理と手続き的な処理の両者を混在させた状態で記述可能である．Prolog には，数値演算処理において実行速度が低下する傾向があるが，近年の計算機が持つ演算速度は非常に高速

になっており，ライブラリで補助することでこの問題はたいていの場合に解決可能である．Prologの，論理型言語としての高い表現力と手続き的処理の記述力を併せ持つという性質は，知的エージェントの構築に好都合である．

Prologには，メタインタプリタと呼ばれるPrologインタプリタのProlog自身による記述が非常に容易に行えるという特徴がある．すなわち，Prologは対象とするエージェントの構築に適するようにPrologを拡張することが容易にできる．他のエージェントフレームワークと比較して，Prologの持つフレームワークとしての柔軟性は知的エージェントの構築に適している．

Prologでは，プログラムおよび静的データはすべて節データベースに格納される．節データベースはそれ自身が演繹データベースになっており，エージェント内部に容易にデータベースを持たせることが可能となる．節データベース中のプログラムは，そのプログラム自身で参照および改変が可能であるため，状況に応じて動的にプログラムを変更することが可能である．この性質は，開放的な環境で動作する知的エージェントの構築に都合がよい．

Prologには上述の特長があるが，その処理系の実装には単一化やバックトラック処理などの複雑な機構が必要となる．次節では，論理型言語処理系の実装技術についてまとめる．

2.4.3 論理型言語処理系の実装技術

Prologの実装技術に関する調査研究としては，文献[105]が詳しい．本節では，Prologの実装技術を概観する．

Prologの初期の実装は主にインタプリタであった．その後，初のPrologコンパイラDEC-10 Prologが開発された．DEC-10 Prologの仕様は，後のほとんどのProlog処理系の手本となっている．1980年半ばに，D.H.D. WarrenによってWAM(Warren's Abstract Machine)に基づく高機能なProlog処理系の構築方法[130]が提案された．WAMは非常に高機能であったため，その後，多くの研究はWAMの拡張を目指して行われた．WAMの基本原理は，Prologコードを手続き的な逐次コード(WAMコード)に置き換え，それをWAMという仮想機械で実行することにより，高速なPrologプログラムの実行を実現する．例えば，次のPrologコード

```
append([],L,L).
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).
```

は，WAMコードに変換されると

```
1  append/3:  switch_on_term V1,C1,C2,fail
2        V1:  try_me_else V2
3        C1:  get_nil A1
4             get_value A2,A3
5             proceed
6        V2:  trust_me_else fail
7        C2:  get_list A1
8             unify_variable X4
```

```
 9            unify_variable A1
10            get_list A3
11            unify_value X4
12            unify_variable A3
13            execute append/3
```

となる．ここで，A1,...,A4 は引数レジスタ，X4 は一時変数レジスタ，C1，C2，V1，V2，append/3 はコードのアドレスを示す．引数レジスタや一時変数レジスタを管理する単位を環境と呼び，環境は述語の最初の呼出し時に１つづつ生成される．１行目では，append/3 の第１引数が，変数，定数，リスト，および構造のとき，それぞれ V1, C1, C2, および fail に分岐する．ここで，fail への分岐は述語の実行失敗を意味する．２行目では，現在の実行位置を選択点スタックにプッシュしている．３行目では，第１引数を 空リスト (nil) と単一化している．４行目では，第２引数と第３引数の単一化を行っている．５行目は，述語の呼び出し元へ戻る動作を示している．すなわち，append/3 の最初の節定義の適用が成功したことを意味する．３行目および４行目で単一化に失敗した場合には，６行目の V2 に分岐する．６行目は，選択点スタックの先頭をポップして取り除くことを意味している．７行目では，第１引数をリストとして取り出すことを示している．８行目では，第１引数のリストの先頭要素を一時変数 X4 に移動させている．９行目では，第１引数のリストの残りの要素を次に呼び出す述語の第１引数に設定している．１０行目では，第３引数をリストとして取り出すことを示している．１１行目では，第３引数のリストの先頭を先ほど一時変数 X4 に退避した第１引数の先頭要素と単一化している．１２行目では，第３引数のリストの残りの要素を次に呼び出す述語の第３引数に設定している．１３行目では，述語 append/3 の呼び出しを行う．ここで，１３行目の execute は末尾呼び出し最適化 (Last Call Optimization) がこの位置で可能であることを示しており，現在の環境をそのまま用いて次の述語呼び出しを行うことにより，環境のためのメモリ消費を抑制することが可能となっている．通常の述語呼び出しには execute の代わりに call が用いられる．WAM は実在するハードウェアではなく仮想的な機械であるため，WAM の動作をエミュレートするインタプリタか，あるいは WAM コードを CPU ネイティブコードにコンパイルする手法が用いられる．

1990 年代初めまでに，WAM に基づいた多数の商用 Prolog 処理系が開発された．最も代表的な商用 Prolog 処理系に，SICStus Prolog と Quintus Prolog がある．Quintus Prolog は，WAM の考案者である D.H.D. Warren らによって設立された Quintus 社によって開発され，ISO 標準や SICStus Prolog の開発に非常に大きな影響を与えた．Quintus Prolog の重要な点は，２つある．１つは，Prolog で初めて他のプログラミング言語とのインタフェースを実装した点である．このインタフェースでは，他のプログラミング言語で作成したプログラムへの埋め込みと他のプログラミング言語による述語の拡張の２つを同時に実現した．他言語とのインタフェースを実装することにより，Prolog プログラミングを広範囲のアプリケーション開発に利用することが可能となった．もう１つは，Prolog で初めて自己改変コードにおける明確な意味論を提案した点である．節の処理対象となる候補を，その節が最初に呼ばれたときに存在する候補と定めた．それ以外に，Quintus Prolog には生成されるコードが非常に小さいという利点もある．SICStus Prolog は，SICS(Swedish Institute of Computer Science) によって開発された．SICStus Prolog の開発動機の１つに，Quintus

は高機能だが非常に高価であったことがある．SICStus Prologは商用であるが比較的安価であるため，Quintusと並んで広く用いられた．SICStus Prologの特徴的な技術に，変数の参照経路を短縮する手法 (variable shunting) がある．Variable Shunting法では，単一化処理時の変数束縛内容の参照にかかるコストを低減するために，同一選択点区間にある変数間の連続参照部分を1つの参照に変換する．これらのProlog処理系は，その基本設計が1990年代と古く，近年のプログラミングで頻繁に利用されるマルチスレッドおよびネットワークプログラミング等の機能は実装されていない．

WAMの動作を高速化するための拡張として，単一化をコンパイルする際に有効なTwo-Stream アルゴリズム，大域最適化手法，および単一化への型の導入が提案されている[105]．WAMとは異なった実行モデルによる実装もいくつか試みられている． Krall らの VAM(Vienna Abstract Machine)[83] では，引数部分の構築とマッチング処理を並列して行うことにより，実行速度の飛躍的な高速化を実現している．たとえば，WAMでは，`p(X,[a,b,c],Y)` と `p(A,_,B)` をマッチングする場合に，最初に `p(X,[a,b,c],Y)` の引数である `[a,b,c]` を項として構築した後に，無名変数とのマッチングを行う．VAMでは，マッチングと項の構築が並行して行われるため，無名変数とのマッチング対象となる `[a,b,c]` の項の構築をスキップすることが可能となる．Tarauの BinWAM[124] は，節定義をボディ部分が高々1つとなるような節（バイナリ節）に変換し，バイナリ節の実行に特化して簡単化されたWAM(BinWAM) を用いて実行する．たとえば，

```
p(X,X).
p(A,B) :- q(A,C),r(C,D),s(D,B).
```

という節定義がある場合，これを

```
p(X,X,Cont) :- call(Cont).
p(A,B,Cont) :- q(A,C,r(C,D,s(D,B,Cont))).
```

と書き換える．すなわち，継続 (Continuation) を引数として渡していく．BinWAMでは，処理対象が単純であるという利点を生かして非常に高度な最適化が行われており，プログラムの実行を飛躍的に高速化することに成功している．BinWAMは，商用 Prolog処理系 BinPrologの実装に用いられている．

WAMに基づく手法の課題は2つある．1つは，WAMに基づくインタプリタやコンパイラの実装は非常に複雑であり，コードサイズが大きい点である．もう1つは，WAMコードが本来のPrologプログラムとは異なる意味論で構成されるため，そのコードに対して直接リフレクティブな操作を実現することが困難であり，例えば プログラムリストを表示する `listing` や節データベースを参照する `clause/2` といった述語を利用可能とするために，WAMコードとは別に ホーン節で表現された Prolog コードを保持しておく必要がある点である．これらの課題は従来の Prolog プログラミングではさほど問題にならないが，コードモビリティの実現では，プログラムや実行状態を移動させる際のオーバヘッド増加につながる場合がある．

2.4.4 論理型言語に基づくエージェント構築環境

RXF[101] は，マルチエージェントシステムを構築するためのエージェント記述言語である．RXF では，複数の論理プログラムを並行動作させる機能を持ち，制約解消系とリフレクション機構を実現している点が特長である．さらに，RXF では，プログラムの構成単位（エージェント）間において，オペレーティングシステムに非依存な通信機構を実現している．RXF に関する詳細は，文献 [101] を参照されたい．RXF は本研究の先行研究となっており，RXF における研究成果は本研究に多大な影響を与えている．例えば，複数のエージェントが複数の Prolog 処理系の並行動作によって駆動される点，エージェント間の通信機構を持つ点が類似している．本研究が RXF と本質的に異なる点は，RXF が制約解消系やリフレクションを用いてエージェント記述言語の記述力を向上させる点を重視しているのに対して，本研究はアプリケーションフレームワークとしての利用を重要視している点，および効果的なモビリティの実装についてより深く踏み込んで扱っている点である．

Tarau の Jinni[123] は，マルチスレッド処理を実現した軽量な論理プログラミング言語である．Jinni では黒板モデルに基づくデータ共有機構を持ち，スレッドは Prolog のサブセットを実現した簡単なインタプリタによって制御される．Jinni のスレッドは Tarau らの主張するライブコードモビリティ（live code mobility）を実現しており，遠隔でのプログラム実行を実現している．Jinni では，オペレータパーザおよびカットオペレータを除いた単純な文法を採用しており，バックトラックの制御には `once/1` および `if/3` を用いる．Jinni でのマルチスレッドの扱いは，エンジンという単位で行われる．エンジンの生成は `new_engine/3` で行い，エンジンへの問い合わせは `ask_engine/2` を用いる．Jinni のモビリティでは，述語 `move/0` および `return/0` の間にはさまれた述語列の遠隔評価を実現する．例えば，

```
?- there,move,println(on_server),member(X,[1,2,3]),return,println(X),fail.
```

という問い合わせを実行した場合，リモートのサーバ上で `println/1` および `member/2` が実行された後，ユーザの手元で `println(X)` の実行結果として 1 のみが表示され，実行を終了する．すなわち，リモートのサーバ上で実行した問い合わせについてはバックトラックを行わない．Jinni と本研究で提案する *MiLog* では，詳細な実装方法や文法のレベルではかなり異なっているが，実装する機能の種類は非常に似ている．*MiLog* と Jinni は，JAVA 上に構築されている点，マルチスレッドを実現している点，スレッド間の通信機構を実現している点，およびモビリティに相当する機能を実現している点で類似している．最も大きく異なる点は，実現するモビリティの種類である．Jinni のライブコードモビリティは基本的に遠隔評価に基づくものであり，移動中および移動終了後に移動元の計算機上でプログラムが稼動している必要がある．*MiLog* ではエージェントの完全な移動が行われるため，エージェント移動後には移動元の環境が動作している必要はない．また，*MiLog* では任意時刻モビリティ，すなわちエンジンでの問い合わせ実行中における同期を伴ったマイグレーションを実現している点が異なっている．

Nakashima らの GAEA[95] は，有機的プログラミングに基づく論理型言語処理系である．GAEA では，セルと呼ばれる単位をサブサンプションアーキテクチャに基づいて相互接続することにより，即応性を実現している．GAEA と *MiLog* は，エージェントの即応性に着

目している点で類似している．GAEAでは，モビリティの実装は行われていないが，*MiLog*ではモビリティの実装に焦点を当てている点が異なっている．

2.5 エージェントフレームワーク

2.5.1 エージェントフレームワークの定義

エージェントの定義と同じように，フレームワークとはどういうものかという点について，十分な合意が得られた定義はまだ存在しない[2]．ソフトウェア開発における**フレームワーク構築の目的は，抽象化が容易でないようなソフトウェアの設計問題に対して，その設計指針を与えること**である．フレームワークは，ソフトウェア設計の指針のみを与え，その抽象化された内容に対する**具体的な実装は実装者にゆだねる**．例えば，オブジェクト指向フレームワークでは，“ソフトウェアをデータの集まりとそれに対する操作をペアにし，それらを隠蔽したオブジェクトとしてモデル化する”という設計指針を与えるが，それらのオブジェクトの具体的な実装はフレームワークからは提供されず，開発者の手によって実装される．フレームワークは，そこで提供される抽象化の指針に沿って設計されたソフトウェアに対して，共通に利用できる機能をライブラリ，あるいはプログラミング言語処理系などによって提供する．本論文では，**フレームワークをソフトウェア設計の指針を与え，その指針に沿って設計されたソフトウェアに共通的に利用できる機能を提供するもの**と定義する．

フレームワークに関連する用語に，プラットフォームがある．ソフトウェアにおけるプラットフォームとは，ある特定の種類のソフトウェアを実際に動作可能とするためのソフトウェアである[6]．プラットフォームはソフトウェアの動作環境を提供するが，そのソフトウェアの設計方針には積極的に関与しない点が，フレームワークとは異なる．

エージェントを利用したフレームワークの１つに，エージェント指向アプリケーションフレームワークがある．エージェント指向アプリケーションフレームワークでは，アプリケーションをエージェントの集まりとしてモデル化し，エージェント間の相互作用によりアプリケーションを構築する．Shohamは，エージェント指向プログラミング[117]の概念を提唱し，そのための実装としてAgent0を示した．Agent0では，エージェント間の協調によってアプリケーションが構築される．エージェント指向アプリケーションフレームワークの特長は，オブジェクト指向フレームワークのようにオブジェクト間のリンクを静的で呼び出し関係が明確化されたリンクとして定義するのではなく，エージェント間の動的で対等な関係によるリンクを用いてアプリケーションを構築する点である．

エージェントを利用するフレームワークだけでなく，エージェント自身を実現するためのフレームワークも多数提案されている．これらは，エージェント構築フレームワークである．知的エージェントの実現では，エージェントの自律性を実現するために，プランニング，スケジューリング，推論，信念管理，学習などといった，多くの要素技術が必要となる．エージェント構築フレームワークでは，エージェントの構築に必要な要素技術の効果的な組み合わせかた，およびそれらの要素技術用いてエージェントを容易に構築できる

[6] プラットフォームとなるソフトウェアのみではなく，それを動作させるハードウェアまで含めてプラットフォームと呼ぶ場合もある．

ようにするためのライブラリ（あるいは，再利用可能なコード）が提供される．

エージェントの構築は，それ自身が目的である場合は少なく，他の何らかの目的を実現するための手段として構築される場合が多い．多くのエージェント構築フレームワークでは，エージェント自身を構築するためのフレームワークと，それを用いて構築されたエージェントを用いて何らかのアプリケーションを構築するためのフレームワークを混在して用意している．本研究で実現を目指すエージェントフレームワークは，エージェントを用いたフレームワークであり，かつそのエージェントを構築するためのフレームワークも含むものである．

2.5.2 SycaraらのRETSINA

Sycaraらはエージェントに基づく再利用可能なアプリケーション構築フレームワークとしてRETSINAを提案している[122]．RETSINAフレームワークでは，基本コンポーネントとして，エージェント間の協調と調整のためのコンポーネント，プランニングコンポーネント，スケジューリングコンポーネント，および実行監視コンポーネントを提供する．実際のエージェントアーキテクチャの構築は，基本コンポーネントを組み合わせることで行う．例えば，RETSINAフレームワークでは，BDIエージェントアーキテクチャの構築に次の基本コンポーネント用意している．

タスクスキーマライブラリ (Task Schema Library) タスクスキーマライブラリには，プランの断片が目的ごとに索引付けされている．プランは，入力に応じて随時検索され，利用される．

信念データベース (Belief Database) 信念データベースは，ファクト，制約あるいは他の知識の形式を用いて，エージェントの現在の環境および実行状態を格納する．

スケジュール (Schedule) スケジュールには，実行すべき行動の列が格納される．

RETSINAは，すべてのアプリケーションの共通の基盤となる必要はなく，アプリケーションから必要なライブラリを利用する形式を採用している．例えば，CORBAではアプリケーションに対して共通の基盤となることを強制するため，CORBAを利用するすべてのアプリケーションはCORBAに基づいてトップダウンに設計される必要がある．一方で，RETSINAでは，ライブラリを任意に利用すればよいため，アプリケーションのすべてをRETSINA上で再構築する必要性を排除し，アプリケーションの再利用性を向上させている．

本論文の著者の指導教官は，CMUのSycaraの研究室に滞在していた際にRETSINAの開発グループに所属していたため，本論文の成果の一部はRETSINAにもフィードバックされている．

2.5.3 GrahamらのDECAF

Grahamらは，分散環境を指向したエージェントフレームワークDECAF (Distributed Environment Centered Agent Framework) [58]を提案している．DECAFは，知的エージェントの設計，開発および実行のためのソフトウェアツールキットを提供する．DECAFは，比較的規模の大きな知的エージェントの構築に必要な機構として，通信，プランニング，スケジューリング，実行モニタリング，および協調機構を提供しており，さらに，学習，およ

び自己認識のための機構が提供される．

DECAF エージェントの制御はプランとして表現され，*PlanEditor* と呼ばれる GUI ツールを用いる．*PlanEditor* では，実際に実行可能な基本要素はモジュール化されており，それらをつなぎ合わせて階層的なタスクツリーを形成することによって，より複雑な振る舞いをプログラミングできる．

DECAF で構築したシステムは，エージェントネームサーバ (ANS, Agent Name Server)，DECAF フレームワーク，および，作成したプランを格納したプランファイルで構成される．エージェントの振る舞いはプランファイルからプランナにより自動生成され，エージェント間の通信はエージェントネームサーバを仲介する形で行われる．

DECAF では，研究開発における利用を考慮し，短期間でのプロトタイプ開発に焦点を当ててフレームワーク設計を行っている．そのため，DECAF には RETSINA と比較してプログラミングの負担を軽減するためのツール群が多数用意されている．一方で，DECAF はシステムの構築の容易化の代償として実行速度が生じるため，商用アプリケーション等の実行速度が重要な要素となるシステムの開発に使用するには不向きな面もあるが，研究開発用途での利用に適しており，プログラミング教育への適用と評価も行われている [58]．

DECAF で主に支援を行っているのは，プラン等のエージェントの内部構造の設計である．本研究で目標とするアプリケーション構築環境には，エージェントの内部状態ではなくアプリケーションの外観の設計を支援することが必要とされる．

2.5.4 Bergamaschi らの MOMIS

MOMIS[12] は，異種データベースの統合を出発点に開発が進められ，その後に知的モバイルエージェントエージェントフレームワークとしての機能が拡張された．異種データベースの統合には様々なアプローチがあるが，MOMIS では，意味アプローチ，すなわちデータベース中の各レコードの意味に基づく情報統合アプローチを用いている．意味アプローチでは，意味を統一的に扱うための共通な基盤が必要となる．MOMIS では，統一的に意味を扱うために，共通シソーラスをフレームワーク内に用意する．本フレームワークは，エージェントの利用目的が情報統合と明確に定められている点が，他のフレームワークと大きく異なる．エージェントは，利用者からの問い合わせに対して，複数の情報源からの情報を統合することにより返答を行う．エージェントの情報源へのアクセスは，ラッパーエージェントを介して行われる．ラッパーエージェントは，それが対応する情報源に特化した情報抽出機構を持ち，統一したフォーマットによる情報源へのアクセスを提供する．エージェントは，ラッパーエージェントから得た情報を共通シソーラスに基づいて統合し，ユーザに返答として返す．

MOMIS では，問い合わせ自身があたかもエージェントのように扱われ，エージェント自身は問い合わせが終了すれば消滅する．すなわち，MOMIS では実質的に遠隔問い合わせの発行をモビリティとして扱っている．これは，エージェントの永続性に基づくモビリティとは考え方が少し異なる．すなわち，本論文で扱う知的モバイルエージェントと MOMIS でのエージェントは異なったものである．

2.5.5 Berganti らの PARADE

自律性と相互操作性は，ソフトウェアエージェントの特徴的な性質である．Berganti らは，自律性と相互操作性を持ったエージェントを JAVA で実現するための開発ツールキットとして，PARADE [14] を提案している．PARADE では，FIPA 互換の目標指向（Goal-oriented）エージェントアーキテクチャによるエージェントの開発を可能とするための機能を提供する．目標指向のエージェントは，個々のイベントに対してどう対処するかを毎回記述しなくて済むため，自律性を持ったエージェントの実現を容易とする．その拡張として，PARADE ではプランニングエンジンを搭載することにより，エージェントが持つ目標を達成するためのプランを自動生成させることが可能である．目標指向は FIPA ACL の基本設計と親和性が高いため，エージェントの相互操作性の実現にも役立つと Berganti らは主張している．

PARADE では，抽象度の異なる 2 つの方法でエージェントを構築可能である．1 つは，エージェントレベルでのエージェントの設計であり，エージェントの振る舞いは UML による図表現として設計され，PARADE によって JAVA の雛型プログラムに変換される．エージェントレベルでの設計では，エージェントの信念や目標などを意識した設計を行うことが可能である．PARADE におけるもう 1 つの設計抽象度は，JAVA プログラムレベルである．JAVA プログラムレベルでは，PARADE はあたかも JAVA のライブラリであるかのように使用することが可能であり，アプリケーションに特化した最適化を行うことが可能である．PARADE 以外にも，BDI アーキテクチャに基づくエージェントの設計を支援するフレームワークは提案されているが，PARADE では FIPA ACL のレベルでの相互操作性を実現している点が異なっている．

目標指向のエージェントの実現は，知的エージェントの自律性を実現するために重要な課題であり，本研究で扱うエージェントも目標指向の振る舞いを実現することを目指す．本研究では，必ずしも FIPA ACL レベルでのエージェント間通信を実現することを意図していない．本研究では，FIPA ACL のような頑健で汎用の通信基盤ではなく，扱いが容易で記述が簡潔なエージェント間通信機構を提供することで，エージェント設計時のプロトタイプ作成を容易にすることに焦点を当てる．

2.5.6 Huber の JAM

Huber は，知的モバイルエージェントアーキテクチャJAM[67] を提案している．JAM におけるエージェントは，BDI アーキテクチャ[57] に基づいて実装される．JAM では，エージェントの推論のためのアーキテクチャをエージェントに実装することに焦点をあてて，エージェントフレームワークを構築している．

JAM フレームワークでは，エージェントのプログラムは，ゴール，プラン，ファクト，およびオブザーバによって与えられる．ゴールには，エージェントのトップレベルの目標が記述され，推論によって更新される．エージェント生成時には，初期ゴールを与える．初期ゴールは，例えば次のように記述される [67].

```
GOALS:
  PERFORM wander_lobby;
```

```
ACHIEVE initialize:UTILITY 300;
MAINTAIN charge_level "20%";
MAINTAIN safe_distance_from_obstacles 50.0;
```

この記述では，4つの初期ゴールが与えられており，それぞれロビーを歩き回る (wander_lobby) という動作を行う (PERFORM)，効用 (UTILITY) を 300 に初期化する処理を行う (ACHIEVE)，charge_legel と safe_distance_from_obstacles をそれぞれ '20%' と '50.0' に保つ (MAINTAIN) ことを示している．ここで，PERFORM, ACHIEVE, MAINTAIN の違いは，PERFORM が具体的な動作や振る舞いを行うことを示しているが，MAINTAIN では PERFORM のような具体的な動作ではなく特定の状態を保つように振る舞うことを示しており，ACHIEVE は完遂することによって削除されるような特定の動作を示している点である．

プランには，エージェントが特定のゴールを遂行するための手続き的な知識が記述される．プランは，例えば次のように記述される [67].

```
PLAN: {
  NAME: "Plan 1:Gather and process information";
  GOAL: ACHIEVE information_exploited $user_query $result
    $recursed;
  BODY:
    EXECUTE com.jrs.jam.primitives.GetHostname.execute
     $hostname;
    EXECUTE print "Currently at " $hostname "\n";
    OR
     { // Check to see if we are done
       TEST (querySatisfiedp $user_query $solution);
       EXECUTE print "Done working on query.\n";
     }
     { // We are not done, so figure out what to do next
       EXECUTE determineNextInfoSource $user_query
         $nextHostname $nextPort $result;
       EXECUTE agentGo $nextHostname $nextPort;
       EXECUTE gatherAndProcessInfo $query $result;
       ACHIEVE information_exploited $user_query $result
         "true";
     };
    WHEN: TEST( == $recursed "false")
     {
       agentGo $hostname $port;
     };
}
```

この例では，ゴール駆動 (goal-directed) 型のプランとして，情報収集を行うためのプランを記述している．1つのプランは，名前 (NAME), 目標 (GOAL), プラン本体 (BODY) から構成される．NAME には，プランを人間が識別するための便宜的な名前を記述する．GOAL には，そのプランを用いることによって達成可能な目標を記述する．BODY には，その目標を達成するための具体的なプラン（手続き）を記述する．GOAL および BODY 部分では，先頭に $ がついたリテラルがあるが，これは変数を意味する．BODY 部分では，

手続きの実行 (EXECUTE), 条件分岐 (OR, TEST, WHEN) などの記述を用いることが可能となっている.

プランにおけるBODY部分の記述には, エージェントの移動コマンド agentGo を利用することが可能である. JAMでは, プランの実行状態を保存したままエージェントを移動されることが可能である. すなわち, JAMでは, プランの実行部分をプログラミング言語とみなした場合に, 強モビリティを持つ. JAMにおける agentGo コマンドはプラン単位での実行のみでなく, プラン中での実行状態も移動先で復元される.

ファクトには, エージェントの外界の現在の状態についてのエージェント自身が持つモデル (世界モデル) を記述する[7]. 例えば, 積み木の世界についてのファクトは次のように記述される [67].

```
FACTS:
  FACT ON "Block5" "Block4";
  FACT ON "Block4" "Block3";
  FACT ON "Block1" "Block2";
  FACT ON "Block2" "TABLE";
  FACT ON "Block3" "TABLE";
  FACT CLEAR "Block1";
  FACT CLEAR "Block5";
  FACT CLEAR "Table";
  FACT initialized "False";
```

ファクトは, エージェント初期化時に与えられ, 推論に伴って随時更新される.

オブザーバには, エージェントにおけるプランの各ステップ実行ごと起動される簡単な手続きを記述し, 主に非同期のメッセージ受信に用いる. オブザーバには, サブゴールを生成せず短時間で実行が終了するような短い処理を記述する.

JAMにおけるエージェントプログラムの記述は一見して宣言的に見えるが, その動作はむしろ手続き的であるといえる. JAMにおけるエージェントプログラムの実行の効率性, およびモビリティの実装の効率性という観点については, 文献 [67] では議論されていない.

2.5.7 その他のエージェントフレームワーク

Jeonらの JATLite[74] は, JAVAによるエージェントフレームワーク開発を支援するためのライブラリパッケージである. JATLiteは, 基本的なエージェント間通信機能とそのテンプレートを提供する. JATLiteはエージェント間通信機構の構築支援に焦点を当て, KQML[34] に基づくエージェント間通信を支援するための強力なライブラリ群を用意している. JATLiteでは通信部分をライブラリとして提供しており, 本研究ではエージェント間通信の記述をエージェント記述言語の一部として実現しようとしている点で異なっている.

d'Invernoらの dMARS[29] は, PRS (Procedual Reasoning System)[56] を発展させたエージェントフレームワークである. dMARSは, BDIアーキテクチャに基づくエージェントの実装を支援する. dMARSエージェントでは, BDIモデルはエージェントの持つプラン

[7] 文献 [67] 中では'ファクト'という言葉を使っているが, 実際の意味はむしろ'信念'に近い

として表現される．dMARSエージェントはプランライブラリを内部に持ち，BDIアーキテクチャに基づいてプランの展開および実行を制御する．dMARSはエージェントにBDIに基づく推論機構を実現するためのフレームワークであり，本研究では推論機構自身を記述するためのフレームワークを提供しようとしている点で異なっている．

NwanaらのZEUS[96]は，JAVA上で開発されたツールキットであり，主に，熟考型のエージェントを対象としている．ZEUSでは，プランニングやエージェント間通信等のライブラリ，エディタやコードジェネレータ等のエージェント開発環境，および，ネームサーバや視覚化支援等のユーティリティエージェントから構成される．ZEUSではエージェントの開発を主に支援しており，本研究ではエージェントの開発のみでなくそのアプリケーション化まで含めて支援しようとしている点で異なる．

エージェントフレームワークとしては，前述のもの以外に，JADE[11], LEAP[1], JACK[23], AgentBuilder[125]が提案されている．JADEはParADEと同様にFIPA ACL互換のエージェント間通信機構を実現しており，LEAPはJADEのPDA(Personal Digital Assistant)端末上での実装である．JACKはRETSINAに類似したJAVA上のツールキットを提供する．AgentBuilderは知的エージェントの実装を支援するための商用ソフトウェアである．本研究は，知的エージェントの構築の支援を行うための環境を提供することを目標としている点で，これらの研究と異なっている．

第3章

知的モバイルエージェントフレームワーク *MiLog*の設計

3.1 エージェントの持つべき性質

知的モバイルエージェントの実現について議論するためには，まず知的モバイルエージェントの特性，およびそれが用いられる環境の特性を明らかにする必要がある．本節では，以降で知的モバイルエージェントのことを単にエージェントと呼ぶ．まず，エージェントと外界をモデル化する．

エージェントと環境のモデル化 エージェントは，エージェントの外界とインタラクションを行うことにより，目的を遂行する．ここで，エージェントの外界のことを**環境**と呼ぶ．**環境**は，エージェントを動作させるための計算機，オペレーティングシステム，ファイル，ネットワーク，ネットワークを通じて扱うことのできる外部のシステムをすべて包含したものである．例えば，Webページおよび他のエージェントは，この**環境**に含まれる．エージェントは，環境上で何らかの**サービス**を提供することを目的とする．ここでのサービスとは，他のエージェントあるいは利用者に何らかの価値のある情報を提供することと定義される．エージェントは，環境にある**資源**を利用してサービスを提供する．**資源**とは，エージェントがサービスを提供するために必要なもので，サービスの定義に含まれないものと定義する．例えば，CPU，メモリ，記憶媒体，およびネットワークは，資源である．エージェントが持つべき性質とは，エージェントが**環境**を適切に扱う，すなわち環境と適切なインタラクションを行い目的を遂行するために必要となる性質である．

3.1.1 エージェントが扱う環境の持つ性質

本節では，エージェントが扱おうとする**環境**自身が持つ性質を明らかにする．

開放性 開放性を持つとは，環境が複数の主体によって協調的に設計，管理，運営されることである．開放性を持つ環境には，複数の設計者によって個別に設計されたシステムが存在し，それらのシステムは，ある特定の標準にしたがって相互にインタラクションを行

うことで個々のシステムがその目的を達成する．開放性を持つ環境の例に，WWW(World Wide Web) がある．WWW では，W3C の策定する標準 (HTTP,HTML など) にしたがって，複数の設計者によって設計された WWW サーバと WWW クライアントが相互にインタラクションを行うことで成り立っている．システムが大規模で複雑なものになるにつれて，システムを特定の主体が独占的に設計および管理することが困難となる．システム全体の大規模化および複雑化が今後さらに進むと考えられることから，開放性を扱うことは重要である．

可変性 可変性を持つとは，環境が事前に予期しないような変化をする可能性を持つことである．可変性を持つ環境では，ある時点まで成り立っていた性質が，その次の瞬間に成り立たなくなる現象が起きる．環境に含まれているシステムのいくつかが，サービスの提供を中止したり，サービス提供の方針を変更することが起きる可能性がある．環境が開放性を持てば，必然的にその環境は可変性を持つことになる．なぜなら，開放性を持つ環境は複数の主体によって運営されるため，それらの運営主体の一部が運営方針を変更することは本質的に妨げられないからである．

対称性 対称性を持つとは，環境に存在するシステムが，相互にサービスを提供および利用可能であることである．対称性を持つ環境では，サービスの提供者と利用者は明確に区別されず，他のサービスの利用に付加価値をつけることで新たなサービスの提供が行われる．たとえば，WWW 上で提供されるメタサーチエンジンサービスでは，WWW 上で提供される他のサーチエンジンサービスを複数利用し，それらの出力結果を統合しフィルタリングを行うことで，検索結果の精度や再現率を向上させている．対称性は，サービスの提供を低い費用で実現するために必須の性質である．

資源偏在性 資源偏在性[1] を持つとは，環境に存在するサービス，およびそれらのサービスの提供に必要な資源が偏って存在していることである．開放性を持つ環境では，サービスの提供主体が個々にどのサービスを提供するかを決定可能なため，環境上で提供されるサービスは多様性を持つ．開放性を持つ環境では，サービスの提供に必要な CPU，メモリ，外部記憶装置，ネットワークなども複数の主体から提供されることが想定される．たとえば，インターネット上にある複数のオンラインオークションサービスは，eBay, Yahoo, Amazon などの複数の主体によって提供される．提供されるサービスは類似しているが，利用料金や利用可能なオークションの種類など，それぞれ異なる特徴をもつ．それらのサービスの提供に必要な資源は，各主体から提供されるものであり，異なるネットワーク経路を通じて提供される．開放性を持つ環境は，潜在的に資源偏在性を持つ．

3.1.2 エージェントの持つべき性質

エージェントは，3.1.1 節で述べた環境の持つ性質をうまく扱う必要がある．エージェントがこれらの性質を扱うために持つべき性質を，次にまとめる．

[1] 局所性とも呼ばれるが，ここでは偏りの意味を強調するためにこの表現を用いた

自律性 開放性を持つ環境では，環境が複数の主体によって運営されるため，それらすべてに対してトップダウンに制御を行うことは，本質的に困難である．開放性を持つ環境上で動作するエージェントは，他のエージェントやユーザからのトップダウンな制御を必要とすることなく，自らを制御する必要がある．ここで，自身の制御を他からのトップダウンな制御なしで行う能力を持つことを，自律性を持つと定義する．自律性を実現するために，エージェントは，自身を制御するために必要となる機能を，すべて自身で持つことが必要となる．例えば，エージェントは，あるサービスの提供に必要とする別のサービスが停止した場合に，ハングアップしてしまうのではなく，そのサービスの停止を認識して，代替サービスを利用したり，あるいはサービスの提供そのものを一時停止するなどの行動を取ることができる必要がある．

即応性 可変性を持つ環境では，変化の速度や度合いが多様性を持つ場合がある．特定の主体によってトップダウンに管理されるような閉鎖的な環境では，変化の多様性を限定することが可能であるが，ここで扱う環境は開放的な環境であるため，変化の多様性を限定することが困難である．エージェントが環境の変化を扱う能力を持つことを，適応性を持つという．環境の変化には，エージェントの設計時において事前に予測することが困難な変化と，事前にその変化の可能性が予測可能であるが，変化に対して短時間で反応する必要がある変化の２種類がある．例えば，ある移動ロボットがあるとして，そのロボットが利用すべき階段が工事中で一時的に使用できなくなるような変化は前者であり，その階段から落ちそうになったときにすばやく姿勢を立てなおすのが，後者である．環境のゆっくりした変化に対応するためには，エージェントの設計後にエージェントの振る舞いの動的変更を可能とするための仕組みが必要となる．環境のゆっくりした変化へは，人間が介入する余地が残される．すなわち，代替エージェントを人間が設計することでサービスの継続性を維持することは可能である．一方で，即応的な対応を求められるような環境の変化に対しては，人間の介入がそもそも間に合わない状況があるため人間が介入できる余地が小さい．したがって，エージェント自身が，急な環境の変化に対して即応的に反応する能力を持つ必要がある．エージェントが，環境の変化に対して即応的に振舞うためには，エージェントが同時に複数のサブゴールを持つことができ，それらのサブゴールの遂行優先順位を動的に変更できる必要がある．

相互操作性 環境が対称性を持つとき，エージェントが相互にサービスを提供しあうための仕組みが必要となる．すなわち，エージェントは他のエージェントに対して明確に意図を伝えられる必要があり，他のエージェントからの表明を理解できる必要がある．複数の主体によって個別に設計されたエージェントが，相互にサービスを利用および提供する能力を持つことを，相互操作性を持つという．相互操作性を実現するための基盤には，エージェントの発話行為に共通性を持たせるための基盤，およびエージェントが発話行為に使用するオントロジーを共通化するための基盤があり，ここではこれらをそれぞれ発話基盤およびオントロジー基盤と呼ぶことにする．エージェントは，発話基盤およびオントロジー基盤を持つ必要がある．

移動性 環境が資源偏在性を持つとき，エージェントは可能な限り資源から独立な存在としてモデル化され，それらの資源を効果的に利用できる必要がある．ここで，エージェントが資源に束縛されず，エージェントが利用する資源をエージェント自身が自由に選択できる能力を持つことを，エージェントが移動性を持つと呼ぶ．ここで，エージェントは移動しなくとも，他のエージェントを媒介して他の資源を利用可能ではないかという疑問が生じる．ここで，環境の持つ開放性を考慮する．開放性のある環境では，エージェントはそれぞれ別個の主体によって管理され運営される．すなわち，他のエージェントの振る舞いはエージェントの制御の対象外であり，エージェントが他のエージェントを通じて資源の利用を制御することには限界がある．したがって，エージェントが資源を効果的に利用するためには，そのエージェント自身が移動する必要がある．ここで，**エージェントが移動するとは，エージェントが利用する資源をエージェント自身の判断で動的に変更すること**と定義する．ここでは，エージェントが他の計算機上に自身の複製（クローン）を生成可能であることも，(広義の) エージェント移動性に含める．エージェントが移動性を持つためには，エージェントの移動性を実現するための何らかの基盤が環境に必要となる．エージェント自身には，環境が持つ基盤を利用して自身を移動可能とするための機構が必要となる．

エージェントの扱うべき環境の持つ性質と，エージェントがその環境を扱うために持つべき性質を表3.1にまとめる．

3.2 エージェントフレームワークが提供すべき機能

本研究では，3.1節で述べた性質を持つエージェントの構築を容易にするためのフレームワークを提供することを目的としている．本節では，エージェントの構築を支援するために，エージェントフレームワークが提供すべき機能と持つべき性質について述べる．本節では，最初に，3.1節で述べたエージェントの持つべき性質の実現を支援するために必要な機能について述べる．次に，エージェントを用いたシステムの開発を支援するために必要となる機能について述べる．

3.2.1 エージェントの実現のための機能

3.1節で述べた，エージェントが持つべき性質とは，自律性，即応性，相互作用性，および移動性であった．本節では，これら4つの性質を持つエージェントの構築を支援するために，フレームワークが提供すべき機能を，それぞれの性質ごとに議論する．

自律性の実現の支援 自律性の実現を支援する際に，まず議論すべきこととして，自律性を実現するための機構そのものを提供するべきか，それとも自律性を実現するための機構の実現を支援すべきかがある．エージェントの自律性を実現するための機構には，例えばプロダクションシステム，ベイジアンネット，あるいはプランナーなどがある．アプローチとしては，プロダクションシステムやプランナーを部品として提供しそれらを自由に組み合わせてエージェントを構築できるようにするアプローチ（部品提供アプローチ），お

エージェント周囲の環境が持つ性質	エージェントが持つべき性質
開放性	自律性
可変性	即応性
対称性	相互操作性
資源偏在性	移動性

表 3.1: 環境の持つ性質とエージェントの持つべき性質

よびプロダクションシステムやプランナーを容易に記述できるようなプログラミング言語処理系を提供するアプローチ（基盤提供アプローチ）がある．

部品提供アプローチでは，プロダクションシステムやプランナーなどの部品は，既存のプログラミング言語から利用可能なライブラリとして提供される．エージェント開発者は，提供されたライブラリを組み込んだプログラムとしてエージェントを開発する．部品提供アプローチの利点は，部品を利用することでコーディング量を削減できる点である．部品提供アプローチの課題は，開発者が必要とする機能と部品が提供する機能が一致しない場合にそれらの調整を行う負担が意外に大きい点である．基盤提供アプローチでは，開発者が必要とする機構を少ない工数で実現するためのプログラミング言語が提供され，プロダクションシステムやプランナーなどをそのプログラミング言語処理系を用いて記述する．基盤提供アプローチの利点は，開発者の必要性に合致した機構を確実にすばやく構築できる点である．基盤提供アプローチの課題は，開発者は機構の実現のために，ある程度の工数をかけて開発を行うことが必要となる点である．

本研究で扱うエージェントは，自律性以外に移動性も持つ．移動性を持つエージェントは，過大な機能を持つことによる移動時オーバヘッドの増加を防ぐために，必要最小限の機能で構成されることが望ましい．エージェントを必要最小限の機能で構成することを支援するために，本研究では基盤提供アプローチを取る．すなわち，エージェントの自律性を実現するための基盤としてプログラミング環境を提供する．エージェントの自律性を保つために，本プログラミング環境は，エージェントごとに独立したプログラミングを実現すべきである．エージェントの記述を簡潔にかつ容易に行えるようにするためには，エージェントの記述に宣言型言語などの高水準言語が利用可能であることが必要とされる．

即応性の実現の支援　即応性の実現とは，すなわちエージェントが行う目標の優先順位付けの実現である．即応性の実現には，エージェントが行う目標の優先順位の変更を実現し，なおかつその変更要求が発生してから短時間で変更を完了することが必要になる．即応性は，エージェントが持つ個々の目標の遂行過程内ではなく，それらの目標そのものへのメタなレベルでの操作として実現される．即応性を実現するための機構は，エージェントを構成する基本要素としてエージェント自身に組み込む必要がある．

最も簡単な即応性の実現方法は，割り込み処理の実現である．割り込み処理とは，現在行っている処理を一時的に中断して別の処理を行い，処理完了後に中断した処理を継続する機能である．割り込み処理の実現では，実行中の処理の中断および再開の機能を実現する必要がある．割り込みで行う処理は，中断した処理に対して影響を与えることはできるが，中断した処理自身を破壊して再開不能にしてはならない．即応性を実現するためには，割り込み処理の要求が発生してから実際に割り込み処理が開始されるまでの時間（割り込み待ち時間）が重要となる．処理速度などの他の要素への影響を最小限にしながら，割り込み待ち時間をいかに小さくできるかが課題となる．

相互操作性の実現の支援　相互操作性の実現では，エージェントの発話基盤とオントロジー基盤を実現することが重要である．オントロジー基盤の実現は重要な課題であるが，比較的小規模なシステムの構築には統一したオントロジーを用いることはそれほど困難ではなく，また，オントロジー基盤を十分に扱えるだけのシステムが十分に普及していないため，

それらのシステムとの対話に必要となるような大掛かりなオントロジー基盤の提供は，本研究の目的を超えるものであると考えられる．オントロジー基盤の提供は，DAML[63] 等の関連研究に譲り，本研究では，相互操作性の実現の支援をエージェントの発話基盤の提供に集中する．

エージェントの発話基盤を実現するには，エージェントが互いを認識し，確実に発話行為を相手に届けるための基盤（通信基盤）が必要となる．エージェントの通信基盤には，エージェントに固有の名前を与える名前サービス，およびその名前を用いてエージェント間でのメッセージ交換を実現するメッセージ通信が必要となる．メッセージ通信は処理の単位として抽象度が低く，これを用いてエージェント間での相互作用を実現するには開発者の負担が大きい．本研究では，単純なメッセージ交換よりも抽象度の高い通信プリミティブとして，問い合わせを用いる．問い合わせは，それ自身が発話行為としての質問の意味を含んでおり，問い合わせに対して返答を行うことが期待される．問い合わせに基づくエージェント間通信機構の実現，およびその機構を用いた相互操作性の実現の具体的な支援方法の検討が課題となる．

移動性の実現の支援 移動性の実現の支援手法には，移動に必要な機能の一部のみを提供し，残りの処理を開発者自身に記述してもらう手法と，移動を完全な機能として提供し，開発者にはエージェントの移動のトリガーのみを記述してもらう手法がある．これらの手法で実際に提供されるモビリティは，前者が弱モビリティ，後者が強モビリティとなる．

強モビリティは，エージェント開発者の負担を軽減できる点で有用である．しかしながら，不十分な強モビリティの実装は，逆にエージェント開発者の負担を増大させる．例えば，エージェント移動時にファイルハンドラやネットワークソケットの退避および復元処理を開発者が記述する必要がある場合，開発者の実質的な負担は弱モビリティを利用した場合と変わらなくなってしまう．そこで，弱モビリティをあえて採用する方法もあるが，その方法にはすでに多くの関連研究が存在するため，弱モビリティの有効性の検証は他の研究に譲る．本研究では，強モビリティを持つエージェントプログラミング言語処理系の構築を目標とする．すなわち，開発者はエージェントの移動のトリガーのみを記述すればよく，エージェントの移動によって生じる問題を可能な限り未然に防ぐような設計とする．強モビリティの実現方法が課題となる．

3.2.2 エージェントシステムの開発のための機能

エージェントを用いてシステムを開発するためには，3.2.1 節で述べたエージェントの持つべき性質の実現を支援するだけでなく，エージェントの開発自身を支援するための枠組みが必要になる．本節では，エージェントを用いたシステムを開発するために，本研究で提供すべき機能について述べる．

エージェントプログラムの実行可能性 エージェントの振る舞いを記述したプログラムを実際に動作させて試行錯誤するための環境を，フレームワークで提供するべきである．エージェント記述のための枠組みを与えることは重要であるが，エージェント開発者の利便性を考慮し，記述したエージェントを即座に試動作できることは重要である．エージェント

を実際に動作させるための機構（エンジンと呼ぶ）を，どのように実現するかが課題となる．そのエンジンは，エージェント記述言語の解釈実行機能を持つ言語処理系によって構成される．エンジンは，3.2.1節で述べた機能を実現する必要がある．

エージェントの拡張容易性 エージェントの実現には，本フレームワークで提供されるエージェント記述言語のみでは記述が困難なモジュールが必要となる場合が考えられる．例えば，音声認識にはFFT（高速フーリエ展開）などの計算が必要であり，これらは既存の手続き型プログラミング言語を利用して記述したほうが効果的である．エージェントおよびエージェント記述言語に拡張性を持たせることが必要となる．エージェントの拡張を現実的に使用可能とするには，単に拡張方法を提供するだけでなく，拡張方法自身が容易に利用できる必要がある．コードの再利用性の観点から，拡張は個々のエージェントに対して機能の追加を個別に行うのではなく，拡張を行ったエージェント自身を，1つのエージェントのクラスとして扱えるようにするべきである．

エージェントプログラムの改変容易性 エージェントを用いたシステム開発は新しい領域であり，システムの実現のために多くの試行錯誤が必要になると考えられる．システム開発における試行を効果的に行えるようにするために，エージェントプログラムは，単に実行可能であるだけでなく，その改変が容易である必要がある．エージェントプログラムの改変容易性を高めるには，フレームワーク上でのエージェントプログラム修正およびその反映の容易化が1つの解決法となる．

エージェントの振る舞いの理解容易性 複数のエージェントを並行に動作させることでシステムを構成する場合，その開発は並行システムの開発の一種であるといえる．並行システムの開発では，実行シナリオの違いによる動作の非再現性の課題があり，システム自身の挙動を把握することが容易ではない[110]．フレームワークには，エージェントの振る舞いに関する情報を開発者に適切に伝えるための機能が必要である．エージェントの基本的な振る舞い（例えば，実行中，停止中，どの環境で動作中かなど）の情報を開発者に提示するのみでは不十分である．開発者自身がエージェントの振る舞いに関する情報の収集基盤として（例えば，エージェントの振る舞いの表現をプログラミング可能とするなどの手段を提供するなど）本フレームワークを利用可能とする必要がある．

エージェントのメタ操作性 分散して動作する複数のエージェントを用いたシステムの開発を行う場合，システムの動作に必要なエージェントにプログラムのロードなどの必要な準備を行い，それらを適切な実行環境に配置する作業が必要となる．この準備作業をフレームワーク内部で記述可能とすることにより，準備作業の自動化が可能となる．準備作業にの記述は，エージェントの生成，およびエージェント外部からのエージェントの移動など，エージェントに対するメタな操作が必要となる．

外界との通信可能性 開発したシステムを実際に運用するためには，ユーザおよび他のシステムに対して，何らかのインタラクションを行う必要がある．これらのインタラクションは複数のエージェントに対して共通的に利用可能なものである．フレームワークは，ユーザお

よび他のシステムとのインタラクションを行うための機構を提供する必要がある．ユーザや他のシステムとのインタラクションの手段の1つに，HTTP(Hypertext Transfer Protocol)がある．HTTPでは，WWWブラウザを用いてユーザとのインタラクションが可能となるほか，他のWWWサーバとのインタラクションにも利用可能である．HTTPは，インタラクションの手段として有用であると考えられる．しかしながら，HTTPに基づく通信は，特にユーザとのインタラクションを行う際に，ユーザからの要求なく能動的に情報を送ることには適さない．ユーザへの能動的な情報提供のために，GUI(Graphical User Interface)に基づくユーザインタフェースの実現も検討する．

以上で議論した，エージェントフレームワークが必要とする機能を，表3.2にまとめる．

3.3 *MiLog* エージェント記述言語の設計

3.2節で述べた性質を備えたエージェントフレームワークを実現するために，本節では，エージェント記述言語の定義を行う．自律性，即応性，相互操作性および移動性を持ったエージェントの振る舞いを記述可能な，エージェントの記述言語を定義する．

3.3.1 *MiLog* エージェント記述言語の定義

本節では，*MiLog* エージェントフレームワークにおけるエージェントの振る舞いを記述するための言語として，*MiLog* エージェント記述言語を定義する．

エージェント記述言語の定義

MiLog エージェント記述言語（以後，*MiLog* 言語と呼ぶ）は，論理型言語に基づいて，エージェントの振る舞いを記述する．エージェントの振る舞いの記述は，エージェント自身の推論，他のエージェントとの通信，およびエージェント自身のマイグレーションの記述によって構成される．*MiLog* 言語 $\mathcal{L}_{\mathcal{M}}$ は，ホーン節 $\mathcal{H}$ によって表現される．

$$\mathcal{L}_{\mathcal{M}} = \{\mathcal{H}_0, ..., \mathcal{H}_n\} \quad (n \geq 0)$$

ここで，ホーン節 $\mathcal{H}$ は，

$$\mathcal{H} = head \leftarrow body_1 \wedge ... \wedge body_n \quad (n \geq 0)$$

と定義される．ここで，ヘッド要素 $head$ およびボディ要素 $body_i$ は，いずれも原始論理式 $\boldsymbol{p}$ である．

$$head \in \boldsymbol{p}, \quad body_i \in \boldsymbol{p} \quad (0 \leq i \leq n)$$

述語 $\boldsymbol{p}$ は，述語名 p および項 $t_0, ...t_n$ を用いて

$$p(t_0, ..., t_n) \quad (n \geq 0)$$

と表現される．なお，$n = 0$ の場合は，単に p とも表記される．項 t は，変数 ($X, Y, ...$)，定数アトム ($p, q, ...$)，および $n \geq 1$ 個の項を引数に持つ関数 $f(t'_0, ..., t'_n)$ のいずれかであ

表 3.2: エージェントフレームワークが提供すべき機能

目的	提供すべき機能
自律性実現の支援	宣言型言語によるエージェントの振る舞いの記述
即応性実現の支援	割り込み処理
相互操作性実現の支援	問い合わせに基づくエージェント間通信
移動性実現の支援	強モビリティ
エージェントの実行可能性	エージェントプログラム解釈実行エンジン
エージェントの拡張容易性	手続き型言語によるエージェントの機能拡張
エージェントの改変容易性	フレームワーク上でのエージェントプログラム修正
振る舞いの理解容易性	エージェントの挙動の表現と表現のカスタマイズ
エージェントのメタ操作	メタ操作の記述
外界とのインタラクションの実現	HTTP 通信機構，GUI

る．ここで，$list(t''_0, list(t''_1, ...))$ という形をとった，2引数で入れ子構造になった特別な種類の関数を，単純に $[t''_0, ..., t''_n]$ と表記する（リスト表記）．

エージェントの定義

エージェント $\mathcal{A}$ は，その実行環境 $\mathcal{E}$，プログラム $\mathcal{P}$，実行状態 $\mathcal{S}$ を用いて，

$$\mathcal{A} = \langle \mathcal{E}, \mathcal{P}, \mathcal{S} \rangle$$

と定義される．ここで，プログラム $\mathcal{P}$ は，

$$\mathcal{P} = \{H_{\mathcal{A}_0}, ..., H_{\mathcal{A}_n}\} \subset \mathcal{L}_{\mathcal{M}} \quad (n \geq 0)$$

である．すなわち，エージェントのプログラムは *MiLog* 言語 $\mathcal{L}_{\mathcal{M}}$ で定義される．

次に，エージェント $\mathcal{A}$ の実行状態 $\mathcal{S}$ について，その構造を定義する．実行状態 $\mathcal{S}$ は，問い合わせ q とその問い合わせに対する推論過程 s の対 (q, s) により，次のように定義される．

$$\mathcal{S} = \langle (q_0, s_0), ..., (q_n, s_n) \rangle \quad (n \geq 0)$$

問い合わせ q は，q_0 から順に実行され，実行が終わるとそれに対応する (q, s) が S から取り除かれる．ここで，問い合わせ q には複数の解 $(\theta_1, ..., \theta_n)$ が存在する可能性があることから，何をもって実行を終了とするかについて，明確に定義を行う必要がある．問い合わせの実行終了を定義する．問い合わせ q と節集合 P について，$P \vdash_\theta q$ とは，節集合 P に対する問い合わせ q が導出され，そのための最小の変数束縛の方法 θ が存在することを示し，$P \nvdash q$ とは，節集合 P から問い合わせ q が導出されないことを示すとする．

定義 3.3.1 (問い合わせの実行終了) あるエージェント $\mathcal{A}_i = \langle \mathcal{E}_i, \mathcal{P}_i, \mathcal{S}_i \rangle$ において，'問い合わせ q_j の実行が終了するとき'とは，次の命題が真となることが明らかになったときである．

$$\exists \theta_i (\mathcal{P}_i \vdash_{\theta_i} q_j) \quad \vee \quad \mathcal{P}_i \nvdash q_j$$

すなわち，実行が終わるときとは，解が見つからないときか，あるいは1つの解が見つかったときである[2]．エージェントは S が空になると，新たな問い合わせを与えられるまで待機する．

他エージェントへの問い合わせの定義

次に，他のエージェントへの問い合わせを定義する．他のエージェント $\mathcal{A}_i$ への問い合わせ $\mathcal{Q}(\mathcal{A}_i, q_j)$ は，$\mathcal{A}_i = \langle \mathcal{E}_i, \mathcal{P}_i, \mathcal{S}_i \rangle$ なるエージェント $\mathcal{A}_i$ において，$\mathcal{P}_i \vdash_\theta q_j$ を得ることと定義される．すなわち，

$$\mathcal{Q}(\langle \mathcal{E}_i, \mathcal{P}_i, \mathcal{S}_i \rangle, q_j) \Rightarrow \mathcal{P}_i \vdash q_j$$

[2] この定義を用いると，トップレベルの解を求める過程でのみ解の完全性が保証されなくなる．あえてこの定義を用いたのは，実行の終わりを容易に判定可能とすることで，次に実行すべき問い合わせに迅速に取り掛かることを可能とするためである．また，エージェントのモビリティにより，別解探索の有無をユーザに問い合わせること自体が困難な場合が考えられ，その場合に全解探索を行うことが必ずしも効果的な解決方法とはいえない．解の完全性が必要な場合には，全解探索の仕組みを問い合わせそのものに持たせるようにする．全解探索の仕組みの記述は比較的容易であるため，実用上の問題はほとんど生じないと思われる．

である．ここで，$\mathcal{S}$ の変化に着目すると，他のエージェントへの問い合わせは，

$$\begin{aligned}\mathcal{Q}(\mathcal{A}_i, q_j) &= \mathcal{Q}(\langle \mathcal{E}_i, \mathcal{P}_i, \mathcal{S}_i \rangle, q_j) \\ &= \mathcal{Q}(\langle \mathcal{E}_i, \mathcal{P}_i, \langle (q_{i_0}, s_{i_0}), ..., (q_{i_n}, s_{i_n}) \rangle \rangle, q_j) \\ &\Rightarrow \langle \mathcal{E}_i, \mathcal{P}_i, \langle (q_{i_0}, s_{i_0}), ..., (q_{i_n}, s_{i_n}), (q_j, s_{q_j}) \rangle \rangle\end{aligned}$$

と定義できる．すなわち，問い合わせでは，現在実行されている問い合わせ列の状態を保存したまま，その末尾に新たな問い合わせ q_j を追加する．なお，s_{q_j} は，問い合わせ q_j に対応する初期実行状態である．

割り込み問い合わせの定義

次に，エージェントへの割り込み問い合わせを定義する．割り込み問い合わせ実行 $\mathcal{I}(\mathcal{A}_i, q_j)$ は，次のように定義される．

$$\begin{aligned}\mathcal{I}(\mathcal{A}_i, q_j) &= \mathcal{I}(\langle \mathcal{E}_i, \mathcal{P}_i, \mathcal{S}_i \rangle, q_j) \\ &= \mathcal{I}(\langle \mathcal{E}_i, \mathcal{P}_i, \langle (q_{i_0}, s_{i_0}), ..., (q_{i_n}, s_{i_n}) \rangle \rangle, q_j) \\ &\Rightarrow \langle \mathcal{E}_i, \mathcal{P}_i, \langle (q_j, s_{q_j}), (q_{i_0}, s_{i_0}), ..., (q_{i_n}, s_{i_n}) \rangle \rangle\end{aligned}$$

すなわち，割り込み問い合わせでは，現在実行されている問い合わせ列の状態を保存したまま，新たな問い合わせを，最も高い優先順位の問い合わせとして，実行状態 $\mathcal{S}$ に追加する．なお，s_{q_j} は，問い合わせ q_j に対応する初期実行状態である．

エージェントのクローンの定義

次に，エージェントのクローンを定義する．あるエージェント $\mathcal{C'}_j$ が別のエージェント $\mathcal{A}_i$ の（広義の）クローンであることを $\mathcal{C}'(\mathcal{A}_i, \mathcal{C'}_j)$ と表現する．ここで，2つのエージェント $\mathcal{A}_i = \langle \mathcal{E}_{\mathcal{A}_i}, \mathcal{P}_{\mathcal{A}_i}, \mathcal{S}_{\mathcal{A}_i} \rangle$ および $\mathcal{C'}_i = \langle \mathcal{E}_{\mathcal{C'}_i}, \mathcal{P}_{\mathcal{C'}_i}, \mathcal{S}_{\mathcal{C'}_i} \rangle$ について，

$$\mathcal{C}'(\mathcal{A}_i, \mathcal{C'}_j) \Rightarrow (\mathcal{P}_{\mathcal{A}_i} = \mathcal{P}_{\mathcal{C'}_j})$$

が成り立つ．すなわち，(広義の) クローンとは，同一の節データを持つように作り出された特別なエージェントを意味する．

ここで，エージェントに対する節データの追加操作 $assert(\mathcal{A}, \mathcal{H})$ および $retract(\mathcal{A}, \mathcal{H})$ を定義するとき，(狭義の) クローン $\mathcal{C}$ は，

$$\begin{aligned}\mathcal{C}(\mathcal{A}_i, \mathcal{C}_j) \Leftrightarrow\ & (\forall \mathcal{H}_j \in \mathcal{H})[assert(\mathcal{A}_i, \mathcal{H}_j) \Leftrightarrow assert(\mathcal{C}_i, \mathcal{H}_j)] \\ & \wedge (\forall \mathcal{H}_k \in \mathcal{H})[retract(\mathcal{A}_i, \mathcal{H}_k) \Leftrightarrow retract(\mathcal{C}_i, \mathcal{H}_k)]\end{aligned}$$

と定義される．すなわち，エージェントの（狭義の）クローン $\mathcal{C}$ とは，その元となるエージェント $\mathcal{A}_i$ と同一の節データを共有するエージェントである．

エージェントの移動の定義

次に，エージェントの移動を定義する．エージェント $\mathcal{A}_j = \langle \mathcal{E}_j, \mathcal{P}_j, \mathcal{S}_j \rangle$ の環境 $\mathcal{E}_j$ から他の環境 $\mathcal{E}_{j_{next}}$ への移動 $\mathcal{M}(\langle \mathcal{E}_j, \mathcal{P}_j, \mathcal{S}_j \rangle, \mathcal{E}_{j_{next}})$ は，次のように定義される．

$$\mathcal{M}(\langle \mathcal{E}_j, \mathcal{P}_j, \mathcal{S}_j \rangle, \mathcal{E}_{j_{next}}) \Rightarrow \langle \mathcal{E}_{j_{next}}, \mathcal{P}_j, \mathcal{S}_j \rangle$$

すなわち，エージェントの移動では，エージェントが実行される環境のみが変化し，他の要素（プログラムおよび実行状態）は保存される．

3.3.2 エージェントプログラムの記述

本節では，3.3.1 節で定義した言語に基づいた，実際のエージェントプログラムの記述方法について述べる．

エージェントのプログラムの記述

エージェントプログラムの文法を BNF 記法で定義したものを，図 3.1 に示す．エージェントプログラムの文法は，おおむね Prolog 言語の文法と同じである．例えば，

$$h \leftarrow b_1 \wedge b_2 \wedge b_3$$

というホーン節は，プログラム上では

$$\mathtt{h} :- \mathtt{b1}, \mathtt{b2}, \mathtt{b3}.$$

と記述する．このプログラムの手続き的な意味は，h を満たすためには $b1$, $b2$, $b3$ を実行すればよいと与えられる．実際には，変数や関数，リストなどを伴って，例えば

$$\mathtt{h(X,[Y,Z])} :- \mathtt{b1(X,Z)}, \mathtt{b2(Z,A)}, \mathtt{b3(A,f(Z,Y,a))}.$$

のように記述される．ここで，a は定数アトム，X,Y,Z,A は変数，f(Z,Y,a) は構造，[Y,Z] はリストを意味する．なお，ホーン節のボディが空の節（基底節）は，

$$\mathtt{h(a,b)}.$$

のように，ヘッド要素の直後に '.'(ピリオド) を記述する．

エージェントに対する問い合わせ (Query) は，?- （あるいは，:-）を先頭につけて記述する．例えば，

$$?- \mathtt{h(X,Y)}.$$

のように記述する．問い合わせの実行では，問い合わせの終了の定義（定義 3.3.1）に基づいて，最初の解が求められた段階で解を表示し，解の表示後のバックトラックは行わない．すなわち，解の完全性を保証したい場合には，全解探索を行う機構（'findall/3' など）を問い合わせ内に含ませるようにする．

```
program ::= clause | clause program
clause ::= head ':-' body '.' | head '.' | ':-' body '.'
head ::= literal
body ::= literal | literal ',' body
literal ::= predicate | cut |variable '=' literal | numericalOperation
cut ::= '!'
predicate ::= atom | atom '(' terms ')'
atom ::= [a-z,0-9] [a-z,A-Z,0-9,_]* | ''' (any chars without '') '''
terms ::= term | term ',' terms
term ::= atom | atom '(' terms ')' | list
list ::= '[' ']' | '[' listItems ']' | '[' listItems '|' variable ']'
listItems ::= term | term ',' listItems
variable ::= [A-Z] [a-z,A-Z,0-9,_]* | [_] [a-z,A-Z,0-9,_]*
numericalOperation ::= vnumber 'is' exp
vnumber ::= number | variable
number ::= [0-9]*
exp ::= vnumber | '(' exp ')' | exp ['+','-','*','/'] exp
```

図 3.1: *MiLog* 言語の BNF による記述

エージェントプログラムは，エージェントごとに独立したプログラムとして記述する．たとえば，2つのエージェント $\mathcal{A}_1$,$\mathcal{A}_2$ が存在する場合，それぞれ対応するエージェントプログラム $\mathcal{PROG}_{\mathcal{A}_1}$，および $\mathcal{PROG}_{\mathcal{A}_2}$ を記述する．2つのプログラム $\mathcal{PROG}_{\mathcal{A}_1}$，および $\mathcal{PROG}_{\mathcal{A}_2}$ が，それぞれ，

```
use(tora,macintosh).
```

および

```
use(fukuta,beos).
```

として与えられたとする．このとき，エージェント $\mathcal{A}_1$ に対する問い合わせ ?- use(USER,OS). では，

```
?- use( USER, OS ).
Yes
USER = tora,
OS = macintosh
```

という答えが得られるが，エージェント $\mathcal{A}_2$ に対する同一の問い合わせ ?- use(USER,OS) では，

```
?- use( USER, OS).
Yes
USER = fukuta
OS = beos
```

となり，異なる答えが得られる．

エージェントの生成および削除

エージェントの生成には，メタ述語 'new/1' を用いる．ここで，'/1' は1引数を持つ述語であることを示している．以後は，特に断りなくこの表記法を用いる．エージェントは，それぞれ固有の識別子としてエージェント名を持つ．例えば，'a' という名前をもつエージェントの生成には，問い合わせ

$$?-\texttt{new(a)}.$$

を行う．エージェントの名前を自動的に決めたい場合には，エージェント名を変数として

```
?- new(X).
Yes
X = a_jr_0
```

のように記述することも可能となっている．

本研究では，エージェントの持つ能力は均一ではなく，それぞれが独自の能力を持つことを可能としている．エージェントは，異なる組み込み述語を持つエージェントごとにそ

れぞれ型が定義される．述語'new/1'によって生成されるエージェントは，その問い合わせを実行したエージェントと同じを型持つ．エージェントの生成時に型を明示的に指定する場合には，メタ述語'new2/2'を用いる．例えば，情報エージェントの実装に特化したエージェント型WeblogIRのエージェントを，エージェント名bで生成したい場合には，

$$?-\texttt{new2(b,'WeblogIR')}.$$

という問い合わせを実行する．

エージェントの削除には，メタ述語 delete/0 を用いる．例えば，

$$?-\texttt{delete}.$$

のように用いる．

クローンの生成

クローンの生成には，述語'clone1/1'あるいは'clone2/1を用いる．述語'clone1/1'では，その問い合わせを実行したエージェントの（広義の）クローンが生成され，clone2/1では（狭義の）クローンが生成される．クローンは，エージェントのように識別子として名前を持つ．例えば，'c'の名前をもつクローンを生成する問い合わせは，

$$?-\texttt{clone1(c)}.$$

あるいは

$$?-\texttt{clone2(c)}.$$

となる．

クローンは，それ自身が独立したエージェントプログラム解釈実行エンジンをもつため，その生成元のエージェントが移動等を行っても，それに伴っての移動は行わず，その場に存在しつづける．

（狭義の）クローンは，その生成元のエージェントと節データベースを共有する．(狭義の）クローンは，その生成元のエージェントが移動を行った際に，自動的に（広義の）クローンに変化するか，あるいはネットワーク経由の通信により節データベースの共有を保つことが可能である．

エージェント間問い合わせの記述

エージェント間の問い合わせの記述は，アクティブオブジェクトモデル[21]に従う．すなわち，エージェントの問い合わせを受け取る側は，他エージェントからの問い合わせ要求を受け取るための待ち受けループを明示的に記述する必要がなく，問い合わせのための処理スレッドは自動的にエージェントプログラム解釈実行エンジンによって管理される．エージェント間の問い合わせでは，問い合わせの実行終了に関する定義（定義3.3.1）により，最初の解のみを求め，別解探索は暗黙的には行われない．

エージェント間の問い合わせには，同期方法の違いから，4種類の基本問い合わせ述語 (request/2 query/2, requestF/2, queryF/2) が用意される．以後，2つのエージェント 'a' および b' 間での問い合わせについて考える．エージェント 'a' およびエージェント ' b' には，プログラムとして，それぞれuse(tora,macintosh). およびuse(fukuta,beos). が与えられているとする．

同期問い合わせ 同期問い合わせ述語では，他のエージェントへの問い合わせ結果が返るのを待ち，その結果が問い合わせと単一化される．問い合わせ結果が失敗であれば，問い合わせ述語の実行そのものが失敗する．同期問い合わせの実行には，同期問い合わせ述語 'query/2' あるいは 'queryF/2' を用いる．例えば，述語 'query/2 を用いて，エージェント 'a' がエージェント 'b' に問い合わせ?- use(USER,OS). を行う場合，エージェント 'a' 上での問い合わせの実行およびその結果は

```
?- query( b, use(USER,OS) ).
Yes
USER = fukuta,
OS = beos
```

となる[3]．ここで，エージェント 'b' では，問い合わせの待ち受けループを予め実行しておく必要はなく，問い合わせの起動はエージェントプログラム解釈実行エンジンによって自動的に行われる．

述語 'query/2' では，問い合わせ先のエージェントが他の問い合わせを実行していない状態（アイドル状態）のときにのみ問い合わせを行い，アイドル状態以外の場合には実行に失敗する．一方で，述語 'queryF/2' では，問い合わせ先のエージェントがアイドル状態になるのを待って問い合わせを行う．

例えば，エージェント 'b' 上で有限時間で終わる問い合わせが実行中であるとき，述語 'query/2' では，エージェント 'a' からの問い合わせが

```
?- query( b, use(USER,OS) ).
No
```

と失敗するのに対し，述語 'queryF/2' では，

```
?- queryF( b, use(USER,OS) ).
(エージェント b での問い合わせの実行が終わるまで待つ...)
Yes
USER = fukuta
OS = beos
```

となり，述語の実行が成功する．すなわち，エージェントは問い合わせ先のエージェントの状態を考慮に入れて問い合わせを行うことが可能となっている．

[3] 問い合わせ述語は，?-によるトップレベルでの問い合わせ以外にも，エージェントプログラム中で自由に利用できる．

非同期問い合わせ 非同期問い合わせでは，他のエージェントへの問い合わせの実行を開始した直後に述語の実行が成功し，問い合わせ先のエージェントが問い合わせの実行をしている間に別の問い合わせを並行して実行することが可能となっている．非同期問い合わせの実行には，非同期問い合わせ述語'request/2'あるいは'requestF/2'を用いる．例えば，エージェント'a'がエージェント'b'に問い合わせ'?- use(USER,OS).'を実行させながら，エージェント'a'自身が別の問い合わせ'?-use(USER,OS) 'を並行して実行したい場合には，次のような問い合わせとして記述することができる．

```
?- request( b, use(USER1,OS1) ), use(USER,OS).
Yes
USER1 = _
OS1 = _
USER = tora
OS = macintosh
```

述語'request/2'では，問い合わせの実行開始のみをおこない，その問い合わせの実行結果は束縛として反映されない．述語'request/2'による問い合わせ結果の受け取りには，述語wait/2を用いる．たとえば，次のような問い合わせとして記述する．

```
?- requst( b, use(USER1,OS1) ),
   use(USER,OS),
   wait( b, query(use(USER1,OS1))).
Yes
USER1 = fukuta
OS1 = beos
USER = tora
OS = macintosh
```

ここで，述語'wait/2'の第2引数を構造'query(...)'で囲っているが，これは述語'wait/2'を別の述語で指定ミリ秒待機を行う述語'sleep/1'と誤解して，第2引数に誤って数値を指定してしまった場合に，無限ループに陥るのを未然に防ぐ役割を持っている．また，問い合わせ結果以外の情報（例えば，他エージェントの到着など）を待つための述語として拡張できるようにという意図もある．

ここで，先に述べた同期問い合わせ述語'query/2'は，述語'request/2'および'wait/2'を用いることにより，次のように書き換えることが可能である．

```
query( AGENT, QUERY ) :-
  request( AGENT, QUERY),
  wait( AGENT, query(QUERY) ).
```

述語'requestF/2'は，同期問い合わせでの述語'queryF/2'に対応し，問い合わせ先のエージェントが既に別の問い合わせを実行中の場合に，その問い合わせの終了を待ってから，問い合わせの実行を開始させる．

次に，エージェント間の協調プロトコルとして代表的な，契約ネットプロトコルの記述を示す．契約ネットプロトコルでは，タスクの依頼者が協調対象となる他のエージェントに対してタスクの請負コストを問い合わせ，そのコストの最も小さいエージェントにタスクの依頼を行う．ここで，タスクの依頼者と協調関係にあるエージェントを b1，b2，b3 とすると，タスク依頼者の処理は，述語 request/2，query/2，および wait/2 を用いて次のように記述される．

```
contractNet(Agent,Price,Task) :-
  request( b1, price(A,Task)),
  request( b2, price(B,Task)),
  request( b3, price(C,Task)),
  wait( b1, query(price(A,Task))),
  wait( b2, query(price(B,Task))),
  wait( b3, query(price(C,Task))),
  sortAZ([[A,b1],[B,b2],[C,b3]],
          [[Price,Agent]|_]),
  !,query( Agent, Task ).
```

ここで，述語 sortAZ/2 は第1引数をリストの先頭要素の値が小さなものから大きなものへ順にソートしたものが第2引数となる述語である．また，タスクの遂行コストは price/2 によって各エージェントの知識として表現されるものとする．契約ネットプロトコルは，わずか１０行足らずのプログラムとしてコンパクトに記述可能である．

エージェントの移動の記述

エージェントの移動は，3.3.1 節で定義したように，エージェントの推論過程には影響を与えず，エージェントの実行環境のみを変更する操作である．実行環境は，識別子として名前（アドレス）を持つ．マイグレーションの記述として，述語 'move/1' を用いる．引数には，実行環境のアドレスを指定する．述語 'move/1' は，エージェントのマイグレーションに成功した場合に真となる．例えば，エージェントが自分自身をアドレス '133.68.14.183:17008' の実行環境へ移動させる場合には，問い合わせ

$$?-\mathtt{move('133.68.14.183:17008')}.$$

を実行する．エージェントは永続性を持っているため，この例の場合では，エージェントは実行環境 '133.68.14.183:17008' 上に移動した後，待機状態になる．

次に，実行環境を移動しながら情報を収集するエージェントの記述例を示す．

```
1  gatherInfo(_,[],[]).
2  gatherInfo(Key,[Host|REST],[Info|RestInfo]) :-
3    move(Host),                    % Host へ移動
4    searchInfo(Key,Unfiltered),    % Key を用いて情報検索
5    filter(Unfiltered,Info),       % 検索結果をフィルタリング
```

```
6    gatherInfo(Key,REST,RestInfo). % 残りも同様に...
7  gatherInfo(Key,[_|REST],[[]|RestInfo]) :-
8    gatherInfo(Key,REST,RestInfo).
```

この例で，左側の数値は行番号を示している．このプログラムでは，情報収集を行う述語 'gatherInfo/3' を定義している．述語 'gatherInfo/3' では，第1引数に検索キーワード，第2引数に実行環境のリストを指定し，第3引数に検索結果を受け取る．述語 'searchInfo/2' および 'filter/2' は，それぞれ検索およびフィルタリングを行う述語であり，ここでの定義は省略している．例えば，エージェントに問い合わせ

```
?- gatherInfo( 'Information Agent',
               ['133.68.14.183:17008',
                '133.68.14.152:17008',
                '133.68.14.179:17008'], RESULT ).
```

を与えた場合，先のプログラム3行目での述語 'move/1' の実行により，エージェントはアドレス '133.68.14.183:17008' に移動する．次に，4行目で述語 'searchInfo/2' によって検索が行われる．次に，5行目が実行され，検索結果に対するフィルタリングが行われる．フィルタリング結果が述語 'gatherInfo/3' の第3引数の要素に束縛され，6行目で残り2つの実行環境についての検索が行われる．ここで，3行目での移動が失敗した場合，7行目が実行され，失敗が発生した実行環境上での検索がスキップされる．4行目あるいは5行目で失敗した場合には，3行目のバックトラックによる再実行はされず，その前の述語 'gatherInfo/3' にバックトラックする．すなわち7行目が実行される．7行目の実行は，その直前に 'move/1' により移動した実行環境上で行われる．実行結果は，実行環境 '133.68.14.179:17008' 上で表示される．

割り込み処理の記述

割り込み処理の記述は，割り込み問い合わせ述語を用いて行う．割り込み問い合わせ述語には，'interruptAndQuery/2' および 'interruptAndRequest/2' があり，それぞれ問い合わせ述語の 'query/2' および 'request/2' に対応している．割り込み問い合わせ述語では，問い合わせを実行中の別のエージェントに対して，その問い合わせの実行を中断し，別の問い合わせを実行させることを可能としている．例えば，エージェント 'b' が，問い合わせ

```
?- loop,write(hello),nl,fail.
```

を実行しているとする．この問い合わせは，無限に 'hello' と改行を出力する．ここで，エージェント 'a' がエージェント 'b' に対して割り込み問い合わせ

```
?- interruptAndQuery(b,write(ogenkidesuka)).
```

を実行すると，エージェント 'b' での問い合わせ実行が中断され，割り込みで問い合わせ write(ogenkidesuka) が実行された後，元の問い合わせが再開される．エージェント 'b' の出力は次のようになる．

```
...
hello
hello
% suspend!
ogenkidesuka
% resume!
hello
hello
hello
...
```

割り込み問い合わせ述語では，他のエージェントからの割り込み問い合わせを受け付けることができる．エージェント自身で環境の変化を観測して即応的な振る舞いを行わせるには，割り込み問い合わせ述語と環境観測用のクローンを組み合わせて用いる．エージェントは事前にクローンを生成し，そのクローンに環境の変化を観測させる．クローンは，環境の変化に対して事前に与えられた条件が満たされたとき，エージェントに対して割り込み問い合わせを行う．

クローンへの問い合わせ

これまでに定義したエージェント間の問い合わせ述語では，原則として問い合わせ先のエージェント自身が問い合わせの実行を行っていた．これは，エージェントに対する問い合わせに対して，節データの厳密な同期を行うためである．実際にシステムを構築する際には，節データの厳密な同期が必要ない場合もある．節データ同期の厳密性よりもシステム構築時の利便性が重要となる場合のために，エージェントのクローンへの問い合わせ述語 (queryC1/2, queryC2/2, requestC1/3, requestC2/3) を定義しておく．クローンへの問い合わせ述語では，まず，問い合わせ先のエージェントのクローンが生成され，次にクローンへの問い合わせが行われ，クローンによる問い合わせの実行が行われ，その結果が問い合わせ元に返された後，クローンが削除される．述語名の末尾の'C1'あるいは'C2は，問い合わせの返答のために生成するクローンの種類を示しており，それぞれ'clone1/1','clone2/1'に対応する．述語名の先頭の'query'および'request'は，問い合わせの同期および非同期の種別を示している．'requestC1/3'および'requestC2/3'では，引数が3に拡張されており，述語'wait/2'による問い合わせ結果の受け取りに必要なクローン名が第3引数に束縛される．

任意時刻マイグレーションの記述

割り込み処理の記述とモビリティの記述を組み合わせることにより，状況に応じて動的に移動を行うエージェントを記述することが可能となる．本研究では，これを任意時刻マイグレーション (Anytime Migration) と呼ぶ．例えば，エージェントを環境にかかる負荷に応じて移動させるプログラムは次のようになる．

```
nameOfOriginalAgent(a).  % エージェント名は 'a'
```

```
autoMove :-
  loadIsHigh,                                  % 環境への負荷が高いとき
  getAnotherHost(HOST),                        % 他の移動先を見つける
  nameOfOriginalAgent(AGENT),
  interruptAndQuery(AGENT,move(HOST)),         % エージェントを移動
  move(HOST),                                  % クローンも移動
  !,autoMove.                                  % 監視を続ける

autoMove :-
  sleep(1000),              % 少し間を開けて，さらに監視を続ける
  !,autoMove.
```

このプログラムを持つエージェントに対し，問い合わせ

$$?-\texttt{clone1(X)},\texttt{request(X,autoMove)}.$$

を実行すると，クローンによって環境の監視が開始される．クローンが負荷を観測し，負荷に応じてエージェントを移動させる．任意時刻マイグレーションでは，割り込みでマイグレーションが実行されても，中断していた他の問い合わせの実行状態は保存される．クローンによって移動されたエージェントは，移動先で問い合わせの実行を継続することが可能である．

3.4 *MiLog* 実行環境の設計

本節では，3.2 節で示した要求を満たす知的モバイルエージェントフレームワークである，*MiLog* フレームワーク [45] の実行環境の設計について述べる．本実行環境では，3.3.1 節で示した *MiLog* 言語に基づいてエージェントの記述を行い，記述したエージェントを実際に動作させることを可能とする．

3.4.1 *MiLog* 実行環境の構成

実行環境の実装方法には，実行環境を 1 つのプロセスとして，その実行環境上でエージェントを動作させる方法（マルチスレッドによる実装）と，各エージェントごとに実行環境のプロセスを用意し，それらの実行環境を制御するメタ制御機構を実現する方法（マルチプロセスによる実装）の 2 つが考えられる．本実行環境では，実行環境の運用の容易化，エージェント間での通信の高速化，およびオペレーティングシステムにかかる負荷の低減を重視し，マルチスレッドによる実装を採用した．

MiLog 実行環境の構成を図 3.2 に示す．*MiLog* 実行環境は，1 つのアプリケーション（プロセス）として構成される．実行環境は，エージェントプログラムを解釈実行するためのインタプリタ（*MiLogEngine*）を内部に持つ．各インタプリタは，独立した節データベー

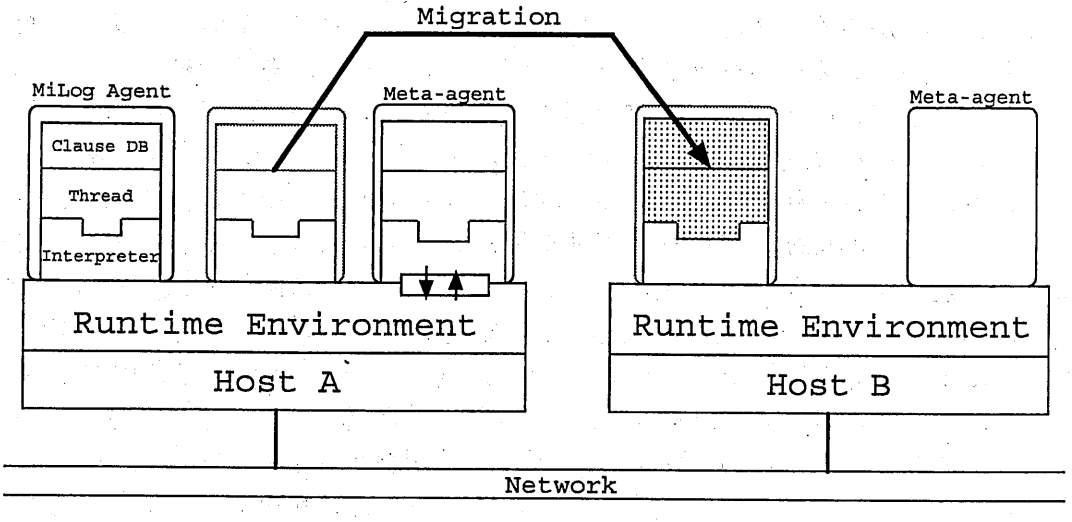

図 3.2: *MiLog* 実行環境の構成

スを持つ．各インタプリタには実行環境内でスレッドが割り当てられる．１つの実行環境上で，複数のインタプリタが並行に動作可能である．同一実行環境上で動作する複数のインタプリタは，同一プロセス上で実行されているため，インタプリタを実現するプログラムをメモリ上で共有可能であり，メモリ利用効率の点で本アーキテクチャは優れている．

エージェントの移動は，移動するエージェントに割り当てられたインタプリタの実行状態（図3.2のThreadとClause DB）を保存し，移動先の実行環境上で復元することによって実現される．すなわち，エージェントのインタプリタが持つスレッドは，実行環境上から保存可能な状態で保持される．

次に課題となるのは，実行環境上で動作するエージェントに対する環境固有の資源へのアクセス手段の提供である．環境固有の資源はエージェントに伴って移動させることが困難であるため，エージェント移動時に適切な方法で資源を解放する必要がある．資源へのアクセスの実現には，エージェント個別に環境へのアクセスを管理する方法（個別アクセス制御）と，フレームワークから環境に対する統一的なアクセス手段を用意する方法（統一アクセス管理）がある．個別アクセス制御では，環境へのアクセスを直接記述可能なため，アクセスにかかるオーバヘッドが小さく済むという利点がある．統一アクセス管理では，エージェントごとのアクセス制御の実現が比較的容易であり，資源の解放に必要となるプログラムの記述が単純になるという利点がある．本フレームワークで課題となるのは，エージェントが割り込み処理によって即応的に移動する可能性がある点である．本フレームワークでは，基本的に統一アクセス管理を利用し，特別に必要な場合にのみ，即応性を犠牲にして個別アクセス制御を記述可能とした．個別アクセス制御では，資源へのアクセスは割り込み不可能な単一の組み込み述語として実現され，組み込み述語内で資源の確保，アクセス，および資源の解放が一括して行われる．統一アクセス管理を利用する場合には，メタエージェントを利用する．メタエージェントは，実行環境上に存在する特殊なエージェントである．メタエージェントの利用には，問い合わせ述語を利用する．メタエージェントは，複数の問い合わせを効果的に処理するために，問い合わせをクローンに委譲する機構を持つ．本フレームワークが提供する問い合わせ述語は記述が平易であり，他のエージェントへの問い合わせと統一した方法で資源へのアクセスを記述可能な点が利点となる．

MiLog 実行環境の実装には，JAVAを用いた．JAVA言語を用いて実装を行うことにより，次のa), b), およびc)の3つの利点が得られた．

a) マルチプラットフォーム性の実現: モバイルエージェント実行環境としては，多数のオペレーティングシステム上で動作することが重要となる．JAVA言語の実行環境は，Windows, Macintosh, Linux, Solaris等の多くのオペレーティングシステム上で稼動する．JAVA言語を用いて *MiLog* 実行環境を実装することにより，*MiLog* 実行環境を多数のオペレーティングシステム上で動作可能とすることができた．本実装の時点で，JAVA言語による実装は最も現実的な実装手段であった．

b) 開発効率の向上: JAVA言語は広く普及しているため，多数のJAVAプログラミング環境が利用可能であった．これらの洗練されたデバッガや統合開発環境等のツールを利用することにより，*MiLog* 実行環境の開発効率を向上させることが可能となった．JAVA言語の持つ特長である，スレッド安全性およびメモリ管理の自動化機能により，プログラミングの効率が向上した．

c) 拡張手段としての利用: JAVA言語を用いて *MiLog* 実行環境を構築したことにより，

*MiLog*実行環境の拡張手段としてJAVAプログラミングを利用することが容易となった．JAVA言語には，プログラミング初心者にも理解しやすく，広く普及し多数の参考書が出版されているため情報が手に入りやすいという利点がある．この利点によりJAVA言語によるプログラミングを習得する人が増加しつつある．JAVA言語を*MiLog*実行環境の拡張手段として利用可能とすることにより，より多くの利用者の手によって*MiLog*実行環境を拡張可能とした．JAVA言語による拡張方法については3.4.3節で述べる．

3.4.2 プログラミング環境

*MiLog*実行環境では，エージェントプログラムの開発を行うためのユーザインタフェースとして，各エージェントにエージェントウィンドウを持たせることを可能とした．図3.3にエージェントウィンドウの例を示す．エージェントウィンドウは，対応するエージェントの移動に伴って移動し，エージェントプログラムの編集，問い合わせ実行，出力結果の表示，およびデバッグを行うためのユーザインタフェースとなる．エージェントウィンドウは，編集モードおよび実行モードの2つのモードから構成される．図3.3では，上のウィンドウが実行モード，下のウィンドウが編集モードであり，図3.3のEのボタンによりモード切替を行う．編集モードでは，エージェントプログラムの作成および編集を行う．エージェントプログラムを図3.3のFの領域に記述する．記述したプログラムは，図3.3のHのボタンによりファイルへの保存，およびファイルからの読み込みを行う．記述したプログラムは，図3.3のGのボタンによりエージェントに読み込ませる(reconsult)ことが可能である．また，emacs等の外部エディタプログラムを用いて編集したプログラムも，図3.3のIボタンにより読み込ませる(reconsult)ことが可能である．実行モードでは，エージェントへの問い合わせの実行，出力結果の確認およびトレース実行の切り替え操作を行う．エージェントへの問い合わせは，図3.3のAのフィールドに入力する．入力された問い合わせはエージェント上で実行され，出力結果は図3.3のBのテキスト領域に表示される．図3.3のJにより，過去に入力した問い合わせの履歴を参照することが可能である．エージェントプログラムのデバッグには，プログラムのトレース機能の利用が有用である．しかしながら，モバイルエージェントは複数の実行環境上を移動することから，単一の実行環境上ではプログラムに対するすべてのトレース結果が得られないため，モバイルエージェントに固有なトレース機能の実装方法の開発が課題となる．本フレームワークでは，エージェントプログラムの解釈実行を行う*MiLogEngine*内にトレース機能を実装し，トレースの実行状態をエージェントの実行に伴って移動可能とした．図3.3のCのチェックボックスにより，トレース実行との切り替えを行う．プログラムのトレース結果は図3.3のBに表示されるほか，図3.3のDのボタンにより，ファイルに保存することも可能である．

各エージェントの動作状態の把握を容易とするために，実行環境ごとにエージェントモニタが用意される．エージェントモニタ上では，その実行環境上で動作するエージェント，それらのエージェントの動作状態の確認，およびエージェントの削除を行うことが可能である．図3.4にエージェントモニタの表示例を示す．図3.4のAの丸いアイコンが，エージェントを示しており，アイコンの下の文字がエージェント名を示している．図3.4のBのように，プログラム中でデバッグ用の述語を使用することにより，エージェントは吹き出しの中に特定の文字を表示させることが可能である．この吹き出し表示は簡単なアイデ

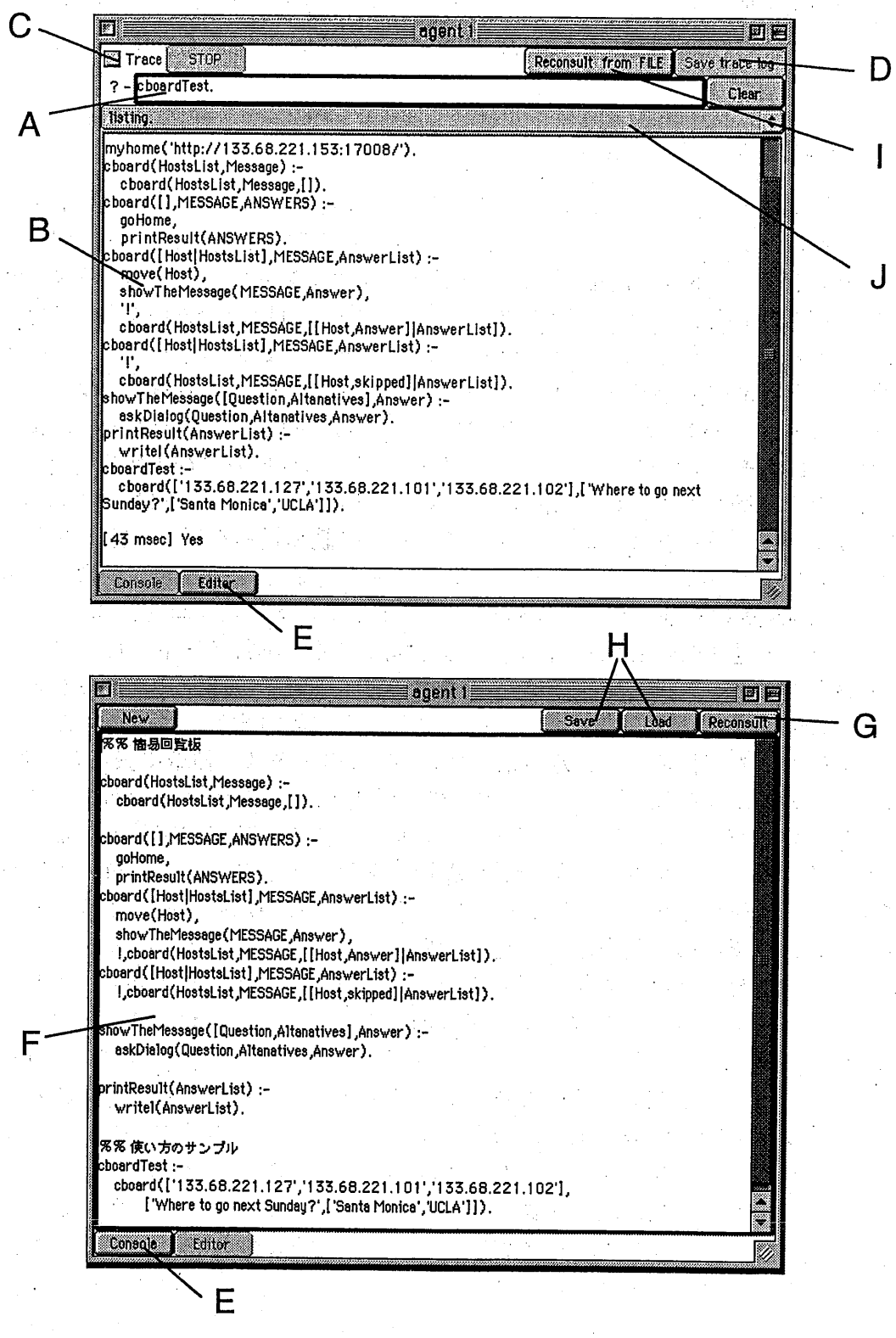

図 3.3: エージェントウィンドウ

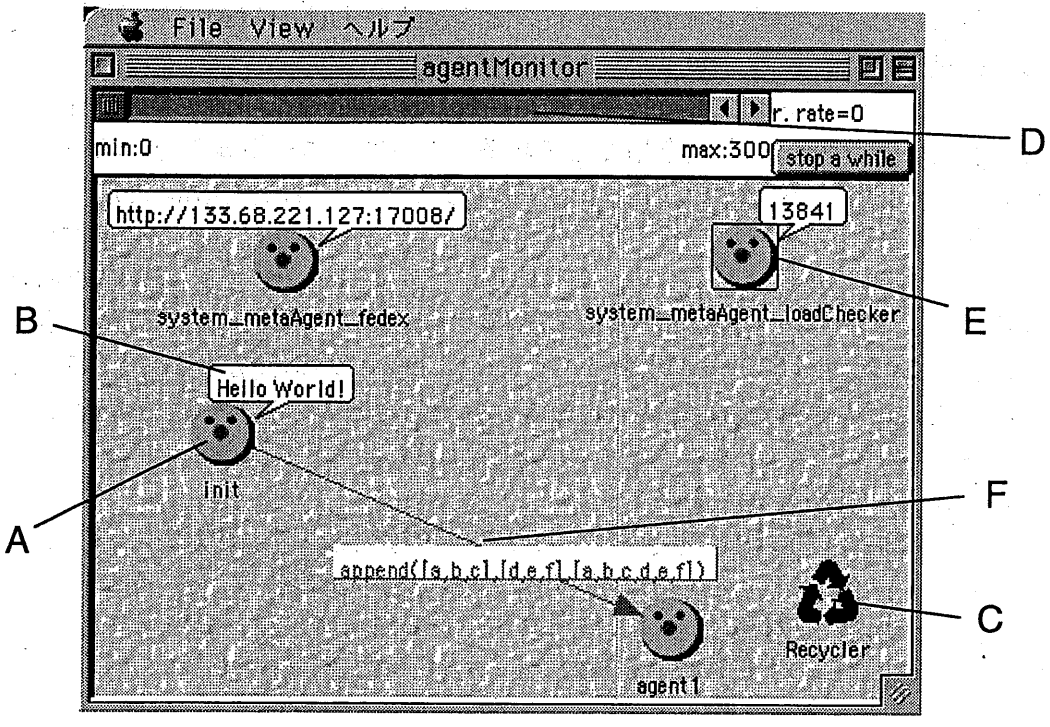

図 3.4: エージェントモニタ

アではあるが，プログラムのデバッグの際に大きな手助けとなる．問い合わせを実行中のエージェントは，図3.4のEのように，アイコンの外周が黒い四角で覆われる．エージェント間の通信は，図3.4のFのようにアイコン間の矢印として表示される．この例では，エージェント `init` からエージェント `agent1` へ問い合わせを行い，結果として `append( [a,b,c], [d,e,f], [a,b,c,d,e,f] ).` を得たことを示している．エージェントの動作速度が観測できないほど速い場合には，図3.4のDのスライダーを動かすことにより，動作速度を一時的に遅くすることが可能である．

本フレームワークでは，エージェントプログラムの開発を補助するためのライブラリとして，エージェント間の通信機構，WWWアクセス支援機構を提供する．エージェント間の協調動作をより簡潔に記述できるようにするために，エージェント間における問い合わせ機能が提供される．WWW上の情報へのアクセスおよび得られた情報の解析処理を簡単に記述できるようにするために，WWWアクセス述語およびHTML解析述語が組み込み述語として提供される．

MiLog では，モバイルエージェントの特性を考慮して，エージェントのユーザインタフェースを構築するために2通りの方法を用意している．1つは，エージェントをWWWアプリケーションとして構築する方法であり，WWWブラウザ（WWW Browser）によってどこからでも利用可能なユーザインタフェースを構築するための手段として利用する．もう1つは，エージェントに独自のGUI(Grahpical User Interface)を持たせる方法であり，より詳細な操作およびユーザへの能動的な情報提供を行うための手段として利用する．前者の実現のために，*MiLog* にはWWW(World Wide Web)サーバ機能が内蔵されており，エージェントはあたかもCGI(Common Gateway Interface)プログラムのようにWWWブラウザからの要求に応答することができる．本機構を発展させた，*MiPage* フレームワークの実現については，5.2節で詳細を述べる．

後者の実現のために，GUI開発支援ツール *iML*[92] を *MiLog* 上に実装した．*iML* を用いることにより，エージェントのGUIレイアウトを画面上で確認しながら編集を行うことが可能となり，*iML* により構築したGUIは，エージェントの強マイグレーションに伴って移動を行うことが可能である．*iML* の詳細については，5.1節で述べる．

3.4.3 実行環境の拡張手段の実現

拡張性の実現はフレームワーク実現における1つの課題である．本フレームワークに対する拡張には，1) 本フレームワーク，およびそれを用いて作成したエージェントの，JAVAプログラムへの埋め込み，2) フレームワーク自身のJAVA言語を用いた拡張 の2つが考えられる．

JAVAプログラムへの埋め込み

本フレームワーク，およびそれを用いて作成したエージェントを他のソフトウェアから利用可能とする利点は，本フレームワークをシステムの実装手段の一部として部分的に利用可能となる点である．すなわち，目標とするシステムの全体は本フレームワークによる実装に適していなくても，適した部分の実装にのみ本フレームワークを用いることが可能

となり，システム構築の柔軟性が向上する．JAVAプログラムから本フレームワークを利用するには，*MiLog*実行環境をバックグラウンドで起動し，HUBエージェントを用いて*MiLog*実行環境上のエージェントと通信を行う．すなわち，HUBエージェントを仲介して*MiLog*実行環境上のエージェントを制御する．ここで，*MiLog*実行環境上のエージェントとの仲介を行う機構を単なるJAVAオブジェクトとして設計せずにそれ自身を*MiLog*エージェントとしたのは，*MiLog*フレームワークの持つエージェント間問い合わせ機構を最大限利用可能とするためである．本設計方法では，その副次的な利点として，移動の必要がなく単に論理プログラミングの手段として*MiLog*を利用したい場合に，HUBエージェントをその手段として利用可能となる点がある．

JAVAプログラムからの*MiLog*実行環境の起動は，次のように記述する[4]．

```
1  String[] args = { "-server", "17008","-agents" };
2  weblog.Boot.main(args);
3  while( weblog.Weblog.getTarget("_boot") != null ) {
4    Thread.yield(); // wait until bootup the runtime environment
5  }
```

ここで，各行の左側の数字は行番号であり，実際のプログラムには入力しない．最初に，1行目で*MiLog*の実行環境に渡す起動パラメータとして，サーバのポート番号の指定，および実行環境起動時での初期エージェントの生成の抑制の指定を行っている．次に，2行目の実行により，実際に*MiLog*の実行環境が起動する．ここで，*MiLog*実行環境の起動時に渡すことのできる起動パラメータは，扱いやすさを考慮し，*MiLog*実行環境を直接JAVAアプリケーションとして起動する場合と同一のパラメータが指定可能となっている．*MiLog*実行環境の起動はバックグラウンド処理として行われるため，*MiLog*実行環境をJAVAプログラムから利用する前に，3から5行目のように実行環境の起動完了を確認する．*MiLog*実行環境の起動後は，JAVA実行環境上で静的なオブジェクトとして動作する．*MiLog*実行環境を利用するJAVAプログラムからは，*MiLog*実行環境への参照等を持つ必要がないため，既存のJAVAプログラムへの*MiLog*組み込みが容易となっている．HUBエージェントの生成は，*MiLog*エージェントのインスタンスオブジェクトをJAVAプログラム上で生成することにより行う．例えば，'agent'という名前のHUBエージェントを生成する場合には，JAVAプログラム中で

```
weblog.Weblog agent = new weblog.Weblog("agent");
```

と記述する．HUBエージェントの生成により，エージェントはJAVAプログラム中からJAVAオブジェクトとして参照可能となり，同時に*MiLog*実行環境からのアクセスも可能となる．生成したHUBエージェントには，queryメソッドを用いて

```
agent.query("append([a,b,c],[d,e,f],X).");
```

のように問い合わせを実行させることが可能となっている．問い合わせは非同期で実行され，その問い合わせ結果は，問い合わせ終了後にgetAnswerメソッドを用いて*MiLog*内部形式で得ることができるほか，getAnswerAsObjectsメソッドを用いてJAVAオブジェクトに変換することも可能である．例えば，先の例では，

[4] ここで，'Weblog'という名称は，*MiLog*の過去の開発コード名であり*MiLog*と同義である．

```
Object result = agent.getAnswerAsObjects("X");
```

とすることにより，resultに，String型の6つのオブジェクト "a", "b", "c", "d", "e", "f" を要素として含むVector型オブジェクトを得ることができる．getAnswerAsObjectsメソッドで変換できる *MiLog* 内部オブジェクトの形式はリスト，アトム等に限られているが，HUBエージェント上で実行させるプログラム上で恣意的に利用するオブジェクトの形式を制限すればよいため実用上の問題は生じず，複数言語を用いたプログラミングにおけるオブジェクトの形式変換の問題を効果的に軽減することが可能となっている．

MiLog 実行環境上のエージェントをHUBエージェントの仲介により利用するには，エージェント間問い合わせ述語を利用する．例えば，新たなモバイルエージェントmobileAgent1を生成するには，HUBエージェントを用いて

```
agent.query("new(mobileAgent1).");
```

と問い合わせの実行を行う．生成したモバイルエージェントへの問い合わせ実行は，HUBエージェント上から問い合わせ述語を利用して行う．例えば，先に生成したモバイルエージェントmobileAgent1に対して問い合わせ ?- doSomeTask(RESULT). を実行させる場合には，例えば問い合わせ述語queryF/2を利用して

```
agent.query("queryF(mobileAgent1,doSomeTask(RESULT)).");
```

と記述する．HUBエージェントから利用可能な問い合わせ述語には，ここで示したqueryF/2以外に，request/2等の他の問い合わせ述語もすべて含まれる．モバイルエージェントmobileAgent1は，問い合わせ実行中に移動することが可能であり，移動した場合でも問い合わせ結果は（通信路が確保されていれば）HUBエージェントに返される．すなわち，HUBエージェントを用いて間接的なモバイルエージェントの利用が可能となっている．

JAVAによる *MiLog* の拡張

本フレームワークでは，*MiLog* 言語処理系をJAVA言語を用いて拡張するためのインタフェースを用意している．論理型言語処理系の他言語による拡張インタフェースとしては，述語（機能）単位での拡張機能が提供される場合が多い．しかしながら，述語単位での機能拡張では，述語内部に静的にデータを持たせた場合に，その述語を利用する複数の処理系から共有データとして扱われてしまうため，セキュリティ上の問題およびモビリティとの相性問題が生じる．本フレームワークでは，エージェント内部に自由にJAVAオブジェクトを持たせることができるように拡張可能とする．なぜなら，エージェント内部にその処理に特化したデータ構造を持つオブジェクトとそのオブジェクトに対する操作を行う述語群を用意することによって，*MiLog* 言語上で利用可能なデータ形式との相互変換にかかるオーバヘッドを削減できるからである．これにより，例えば，エージェント内部にJAVAオブジェクトの形式で行列データを持たせて，組み込み述語により行列演算を高速に実行するという設計が可能となる．

上述の拡張手法を実現するための具体的な手法として，本フレームワークでは *MiLog* エージェントのJAVAクラスの継承に基づく手法を用いた．本手法では，拡張された *MiLog* 言

語処理系は，*MiLog*エージェントのJAVAクラスを継承することで実現される．典型的な拡張方法は，次のようになる．

```
1  import weblog.*;
2  public class myClass extends Weblog {
3    public myClass(String name) {
4      super(name);
5      stringProperty = name + " is myClass";
6    }
7    String stringProperty;
8    public PrologObject sys_sampleBuiltin( PrologObject  args ) {
9      if( arity(args) == 1 && (
10       args.binding().typeVar() || args.binding().typeAtom())  ) {
11       PrologObject p = new PrologObject( PrologObject.typeAtom );
12       if( match( p, args ) ) {
13         return(success);
14       }
15     }
16     return(null);
17   }
18 ...
19 }
```

各行左側の数字は行番号であり，実際のプログラムでは入力しない．ここでは，*MiLog*エージェントのJAVAクラスとして`weblog.Weblog`を利用し，それを拡張して`myClass`を定義している．`myClass`の定義では，1つの内部オブジェクト`stringProperty`（7行目）と1つの組み込み述語`sampleBuiltin`（8から17行目）を追加している．エージェントが内部に持つことのできるオブジェクトは，`java.lang.String`等のシリアライズ可能なオブジェクトに限られる．なぜなら，拡張した*MiLog*エージェントは，JAVAのシリアライズAPIを用いてシリアライズ可能とする必要があるからである．例えば，`java.lang.Thread`クラスのオブジェクトはシリアライズ可能ではないため，エージェントの内部に持たせることはできない．組み込み述語の拡張は，`PrologObject`型の返り値と引数1つを持ち`sys_`から始まる名前を持つ`public`なメソッドを追加することで行う．この条件を満たしたメソッドは，*MiLogEngine*内部でJAVAリフレクション機能を用いて自動的に認識され，組み込み述語として利用可能となる．具体的には，`java.lang.Class`クラスの持つ`getMethods`メソッドを利用して，*MiLogEngine*コンストラクタ中で`public`なメソッドの一覧を取得し，ハッシュテーブルに格納することで実現している．本手法で拡張可能な組み込み述語は，バックトラックを行わない述語に限定している．すなわち，本手法で拡張された組み込み述語は，1度述語の実行に成功した後には，バックトラックにより再実行されることなく失敗となる．この仕様は，バックトラックのための記述を取り除くことにより，組み込み述語のJAVAプログラムによる定義をより簡潔なものとする点で有効である．組み込み述語の定義では，引数として渡された`PrologObject`型のオブジェクト内部に述語とし

て参照する引数の数と値が含まれている（8行目）．引数の数は arity メソッドにより取得することができ（9行目），引数の型は PrologObject クラスのインスタンスメソッドにより得られる（10行目）．なお，PrologObject クラスの説明については，ここでは省略する．引数に対する変数束縛には，match メソッドを用いる（12行目）．match メソッドは weblog.Weblog クラスで定義されている．match メソッドは，PrologObject 型で表現された2つのオブジェクトの単一化を行い，単一化に成功すれば true を返す．メソッドの返り値には，非 null なオブジェクトを返した場合に述語の実行が成功したものとみなされる．例えば，13行目では述語の成功時に weblog.Weblog クラス中で定義されている非 null な定数 success を返している．同様に，16行目では，null を返すことにより述語の実行を失敗させている．

拡張された *MiLog* エージェントは，JAVA の動的クラスローディングを利用して，動的に本フレームワーク内にロードすることが可能である．たとえば，先に定義した myClass クラスに基づく *MiLog* エージェントは，*MiLog* 実行環境から参照可能なクラスパス中に JAVA クラスファイルを置いた状態で，組み込み述語 new2/2 を用いて，

```
?- new2( a, 'myClass' ).
```

のように生成することが可能である．この例では，'a' という名前のエージェントが生成される．ここで，新しくロードされたクラスは，既存の他のエージェントには影響を与えない．すなわち，*MiLog* 実行環境上では，異なる JAVA クラスに基づく *MiLog* エージェントを混在させることが可能となっている．

拡張された *MiLogEngine* にもモビリティが実現され，必要なクラスファイルがエージェントの移動に伴って移送される仕組みを実現している．

3.5 議論

本章では，エージェントが扱う環境の性質として開放性，可変性，対称性，資源偏在性について指摘し，そのためにエージェントが持つべき性質として，自律性，即応性，相互操作性，移動性を挙げた．エージェントの自律性の実現を支援するために，自律性の実現に必要となる機構を最小限の機能でコンパクトに実装可能なエージェント記述言語の必要性について述べた．この目的を達成するために，論理型言語に基づくエージェント記述言語 *MiLog* の設計を示した．論理型言語の強力な記述力は，エージェントの自律性実現のための機構をコンパクトに実装するのに有用であると考えられる．エージェント記述言語のみでエージェントの自律性が実現されるわけではないため，本エージェント記述言語を用いて自律的なエージェントの記述が可能かどうかをさらに考察する必要がある．エージェントの即応性の実現を支援するための1つの方法として，割り込み処理の有用性について述べた．割り込み処理をエージェントの実装に利用可能とするために，割り込み処理のエージェント記述言語 *MiLog* 上での記述と意味論を定義した．割り込み処理と移動性を組み合わせた任意時刻モビリティの記述と意味論を示した．割り込み処理および任意時刻モビリティがエージェントの即応性実現手法として機能面および性能面から十分なものであるかどうかは，さらに議論を行うべき点である．エージェントの相互操作性の実現を支援するために，問い合わせに基づくエージェント間通信の有用性について述べた．問い合わせに

基づくエージェント間通信の記述と意味論を，エージェント記述言語 *MiLog* 上で定義した．本エージェント記述言語を用いることにより，エージェント間の典型的な協調プロトコルの記述が簡潔に行えることを示した．エージェントが頻繁に移動する場合のエージェント間通信の実現については，ヒューリスティックな解決方法でもある程度は実現可能であるが，本質的な議論を行う必要がある部分の１つである．より開放的な環境における相互操作性の実現を考慮した場合，既存のエージェント間通信の枠組みとの相互運用性，オントロジーの問題，およびセキュリティやプライバシの配慮について，さらに議論する必要がある．エージェントの移動性の実現を支援するために，強モビリティの有用性を検討する必要性について述べた．強モビリティの記述および意味論を，エージェント記述言語 *MiLog* において定義した．本エージェント記述言語では，強モビリティによりエージェントの移動の記述が簡潔となることを示した．強モビリティを割り込み処理と組み合わせることで，即応的なエージェントの移動が記述可能となることを示した．強モビリティの詳細な実装方法および高性能化については４章でさらに議論する．

エージェントを用いてシステムを開発するための枠組みに必要となる性質として，エージェントプログラムの実行可能性，エージェントの拡張容易性，エージェントプログラムの改変容易性，エージェントの振る舞いの理解容易性，エージェントのメタ操作性，および外界との通信可能性があることを述べた．エージェントプログラムの実行可能性を実現するために，エージェントフレームワークにエージェントプログラムの実行環境を含ませた．実行環境をインタプリタベースの設計とすることにより，ユーザの作成したプログラムを迅速に試行可能とした．エージェントプログラムの改変容易性は，実行環境をインタプリタベースとすること，および論理型言語に基づくエージェント記述言語を用いることから自然に実現された．エージェントプログラムは実行時にも動的に変更可能であり，エージェントプログラムの改変はエージェントごとに用意されたユーザインタフェースを用いて迅速に行うことが可能である．エージェントの振る舞いの理解容易性を実現するために，エージェントごとにユーザインタフェースを用意し，エージェントの移動に伴って移動しながら，エージェントプログラムの出力，およびトレースを表示可能とした．エージェントの移動およびエージェント間通信に関する振る舞いをユーザが理解するための手段として，プログラマブルなエージェントモニタによる支援方法について述べた．エージェントのメタ操作性を実現するために，エージェントへのメタ操作述語をエージェント記述言語 *MiLog* 上に定義した．メタ操作述語により，システムを動作させる前準備の処理をエージェント記述言語自身で記述可能とした．エージェントの外界との通信可能性を実現するために，WWW へのアクセスインタフェース，WWW 上での情報発信手段，エージェントへの GUI の付与について概観を述べた．より詳細な内容については，５章で議論する．本章で示したエージェント実行環境は，エージェントプログラムの開発に特化したものである．エージェントプログラムの実行に特化し，実行速度およびセキュリティを向上させることは，今後さらなる議論が必要となる点である．

第4章

MiLogEngine: 論理型言語への任意時刻モビリティの実装

4.1 はじめに

本章では，*MiLog*エージェント記述言語の一部として定義した任意時刻モビリティの実装方法の詳細について述べる．最初に，任意時刻モビリティの実装に関して本章で用いる用語の定義を示す．

本節では，以後の議論で必要となる用語の定義を行う．**任意時刻モビリティ**は，**任意の時刻において，複数の問い合わせの実行を同期させた状態でかつ同時に強マイグレーション**させる機能である．ここで，**マイグレーション**はエージェントの移動という動作そのものを意味し，その**機能あるいは能力をモビリティ**と呼ぶことにする．任意時刻モビリティが満たすべき性質は，**マイグレーションの任意時刻性，強モビリティ，マイグレーションの自発性，および複数問い合わせの同期マイグレーション**である．**マイグレーションが任意時刻性を持つ**とは，マイグレーションが予めプログラム中で予期された場面に限定されず，任意の時刻に行われる可能性を持つことである．**強モビリティ**とは，エージェントのプログラムを格納する領域（コード），データを格納する領域（ヒープ），およびプログラムの実行過程を格納する実行状態領域（スタックとプログラムカウンタ）の3つを保存するマイグレーションである．**マイグレーションが自発性を持つ**とは，マイグレーションの制御がその移動対象である**エージェント自身の判断**によって行われることである．**複数問い合わせの同期マイグレーション**とは，プログラムに対する複数の問い合わせが並行に実行された状態で，その問い合わせ間の実行の同期を損なうことなく同時にマイグレーションされることである．ここで，複数の問い合わせが**同時にマイグレーション**されるとは，その中の任意の問い合わせがマイグレーション先で実行を再開するより以前に，他のすべての問い合わせのマイグレーションが完了していることと定義する．マイグレーションの任意時刻性に関連して，**プログラムの基本実行単位**を定義しておく．プログラムの基本実行単位とは，ある整合性の取れた実行状態から次に整合性の取れた実行状態に移るまでのプログラムの実行単位である．ここで，**整合性の取れた実行状態**[1] とは，その実行状態がプログラムの実行を継続するための完全な情報を持つ状態，すなわち強マイグレーション可

[1] ここでいう整合性の概念は，TMS等における知識の整合性の概念とは異なるものである．

能な状態である.

本章の以降では，任意時刻モビリティの特性について議論し，その具体的な実装方法を示す.

4.2 既存のモビリティ実現手法との比較

本節では，任意時刻モビリティを既存のモビリティの実装手法と比較し，任意時刻モビリティの実現における課題を明確にする.

強モビリティは実装上の負担が大きいため，強モビリティと弱モビリティの中間の性質を持つような弱モビリティの実装方法が提案されている．強モビリティと類似した特性を持つものに，遠隔評価 (Remote Evaluation) がある．遠隔評価では，プログラムの一部の実行を遠隔地で行い，その結果を通信で受け取る．主に分散プログラミングの拡張として実装されるモビリティの実装では，遠隔評価にコードのマイグレーションを含ませることにより実現される．たとえば，Jinni[123] で提案されている Live Code Mobility は，遠隔評価にコードの要求時ダウンロードを組み合わせたものである．遠隔評価では，関数などの呼び出し関係などのスタック上の情報は保存されないが，プログラムの実行位置（プログラムカウンタ）を移動先で復元可能であり，モビリティを往復という単純な形に限定した場合には移動後のプログラム実行の継続も可能となる[2] ため，強モビリティの代替としての利用が考えられる．遠隔評価の拡張によるモビリティでは，エージェントが移動可能な時期が，プログラム中の移動コマンドの記述位置に限定されるため，任意時刻モビリティで必要な任意の時刻におけるマイグレーションが困難となり，任意時刻モビリティの実現には適さない.

強モビリティの手法の１つに，強制マイグレーションがある．強制マイグレーションは，プロセスマイグレーション [90] の分野で考案されたマイグレーション手法である．強制マイグレーションでは，マイグレーション対象は集中的な管理システム（以後，単にシステムと呼ぶ）によって管理され，システムが判断する任意の時刻にシステムの指定した実行環境へマイグレーションされる．強制マイグレーションと任意時刻マイグレーションは，強モビリティおよびマイグレーションの任意時間性という２つの共通な性質を持つため，非常に似ている．強制マイグレーションでは，マイグレーションの主導権はシステムにあり，マイグレーションの自発性は実現されない点が，任意時刻マイグレーションとの違いである.

強モビリティの実装方法の１つに，チェックポインティング法 [133] がある．チェックポインティング法では，プログラムの実行がどこまで進んだかを示すチェックポイントをプログラム中に埋め込むことにより現在の実行位置（プログラムカウンタ）に相当する情報を得る手法である．チェックポインティング法では，チェックポイントの区間を単位として移動先にプログラムの実行位置を復元することが可能である．チェックポインティング法に関数やメソッドの呼び出し関係および局所変数の値を記録する能力を持たせることで，スタックの保存を実質的に可能とする手法 [106] も提案されている．従来のチェックポインティング法では，チェックポイントはプログラムに対する特定の操作として実行コード中に埋め込まれるため，コードの肥大化，およびチェックポイントの処理によるオーバヘッドが

[2] 移動によってスタックが分断されるため，強モビリティと同一ではない.

課題となる．チェックポイントの埋め込み間隔を短くするほどマイグレーションを実行できる機会が増えるが，コードの肥大化およびチェックポイントの処理によるオーバヘッドは顕著化する．任意時刻モビリティの実装にチェックポインティング法を用いた場合，チェックポイントの埋め込み間隔を相当短くとる必要があり，チェックポイントの埋め込みによるオーバヘッドの影響が大きいと考えられる．以上の理由から，チェックポインティング法を任意時刻モビリティの実装に用いるのは，あまり効果的でないと予想される．

強モビリティの別の実装方法として，処理系の持つ実行状態へのアクセス手段を実現する方法がある．ここでは，既存の言語処理系の実装に強モビリティを追加する方法について考える．既存の処理系に強モビリティを追加する場合，処理系の多くは実行状態に対するアクセス手段を提供していない場合が多いため，処理系の内部を改変する必要が生じる．処理系のもつ実行状態領域にプログラムからのアクセスが可能となったとしても，その表現がモビリティを考慮した設計になっていない場合がある．たとえば，実行状態の表現がその処理系を実行するCPUアーキテクチャに依存したものである場合，実行状態をそのまま他のCPUアーキテクチャにマイグレーションしても，プログラムの実行は継続不可能である．すなわち，単に既存の言語処理系において実行状態領域のメモリダンプを出力可能とし，それを移送するだけでは，強モビリティは実現されない．強モビリティの実現では，実行状態領域を特定のCPUアーキテクチャ等に依存しない形式で保存可能にする必要がある．既存の処理系に対して実行状態領域の変換処理を実現する負担を考慮した場合，新たに強モビリティを実現した処理系を構築することと比較して，既存の処理系上に強モビリティを追加することが開発の負担軽減になるとはいえない．

任意時刻モビリティでは，割り込み前の処理および割り込み処理の少なくとも2つの問い合わせを同期してマイグレーションさせる必要がある．しかし，既存の強モビリティに関する研究では，複数の問い合わせの実行を同期させて強マイグレーションさせる手法の実現について，ほとんど議論されていない．複数問い合わせを同期したマイグレーション方法を実現するためには，その実現を考慮をしたプログラム解釈実行系を新たに構築する必要がある．複数問い合わせの同期マイグレーションを満たすマイグレーション方法として，3章で割り込み処理に基づく同期マイグレーションを示した．割り込み処理に基づく同期マイグレーションの実現には，プログラムの実行状態を自身で管理可能なプログラム解釈実行系を新たに実現することが必要と考えられる．本目的を達成するために，*MiLog*エージェント記述言語の解釈実行系 *MiLogEngine*を実装する．*MiLogEngine*には，プログラムの実行状態を自身で管理可能とするための機構を実現する．本章の以降では，*MiLogEngine*の設計および実装を示す．

4.3 *MiLogEngine*の動作モデル

本節では，*MiLogEngine*における割り込み処理の実現，および割り込み処理中におけるマイグレーション（任意時刻マイグレーション）の実現方法を示す．*MiLogEngine*の構成を図4.1に示す．*MiLogEngine*は，**インタプリタコア，実行状態，実行状態レジスタ，実行状態スタック，サスペンドフラグ，およびプログラム格納領域**[3] から構成される．ここ

[3] *MiLog*言語では大域的なデータはプログラム格納領域（節データベース）に格納されるため，データ格納領域はプログラム格納領域に含まれるものとみなすことができる．

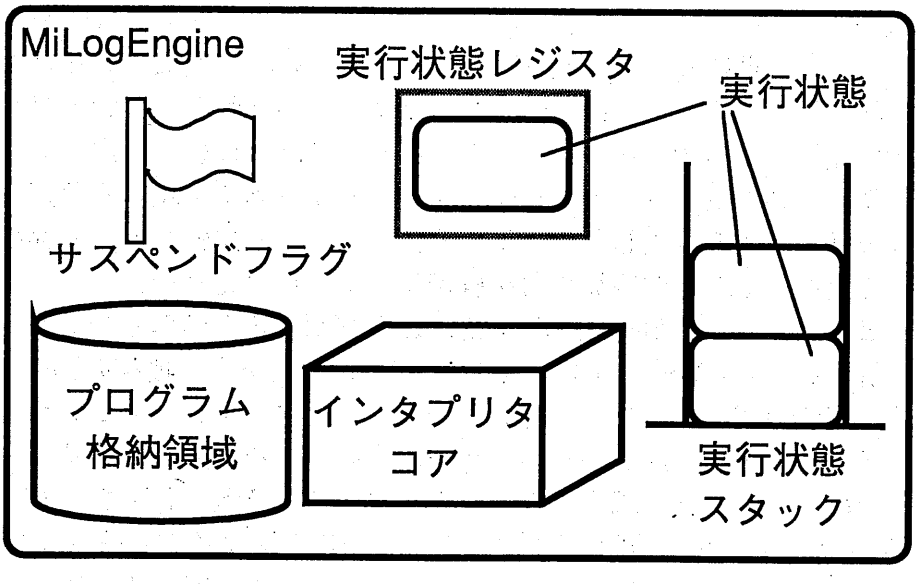

図 4.1: *MiLogEngine* の構成

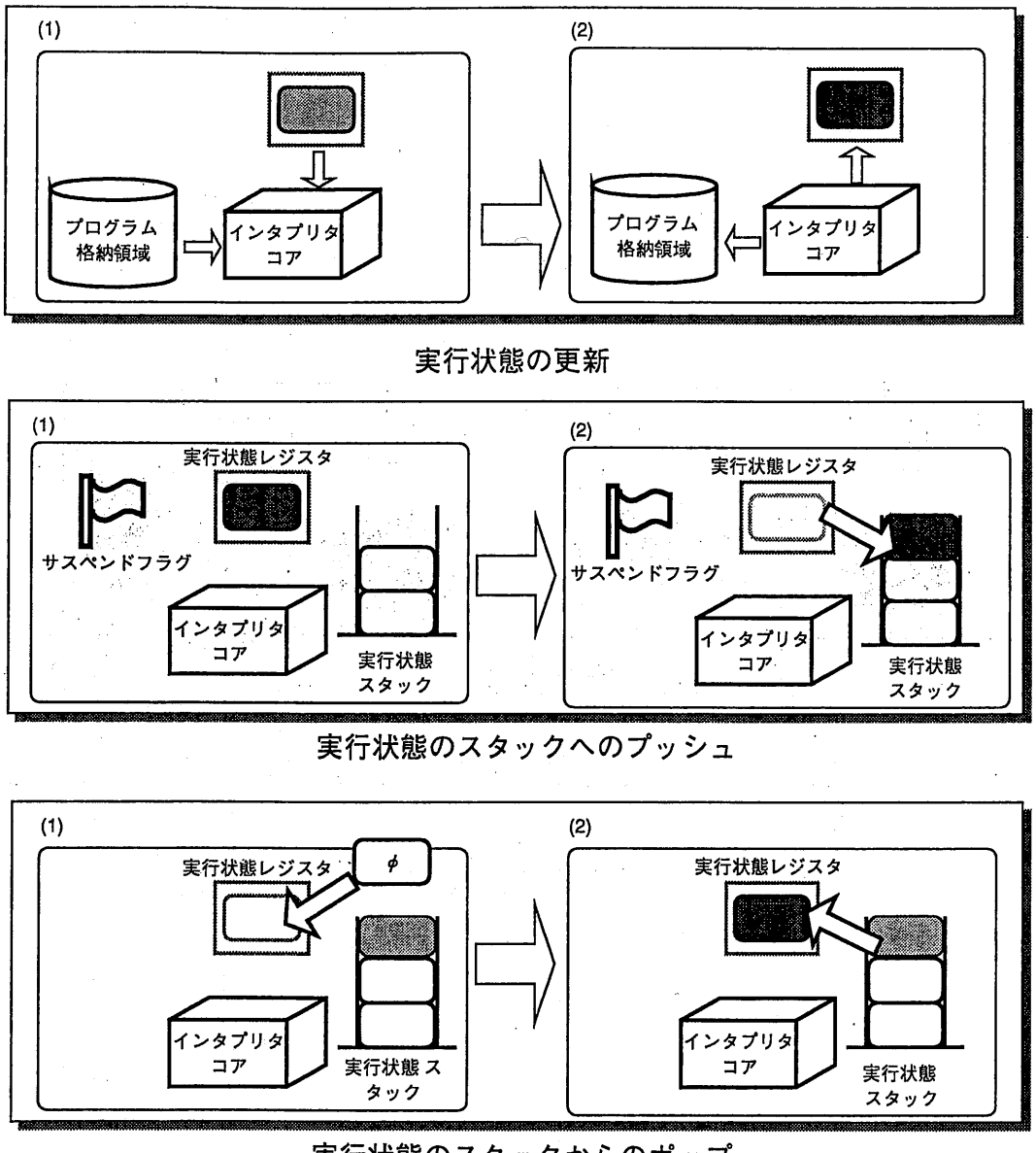

図 4.2: *MiLogEngine* の基本動作

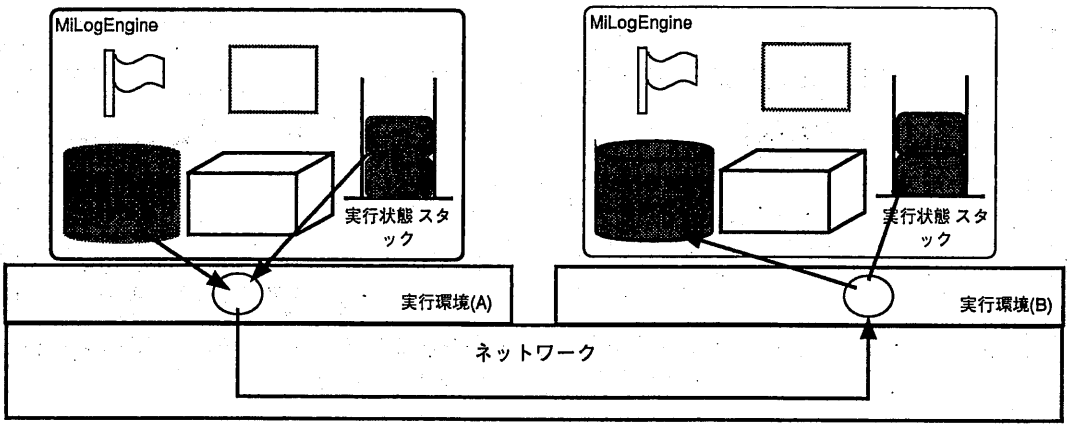

図 4.3: *MiLogEngine* における任意時刻マイグレーション

での実行状態とは，LISP 言語でいうところの継続 (Continuation) に相当するものであり，プログラムが今後実行すべき内容を表現している．インタプリタコアは，ある実行状態を入力として受け取り，新たな実行状態を出力するブラックボックスとしてモデル化される．インタプリタコアは，実行状態レジスタに実行状態が設定されると動作を開始し，実行状態レジスタに設定された実行状態が終了状態となるか，あるいはサスペンドフラグが立てられるまで，実行状態レジスタの値を更新しつづける．実行状態レジスタの値が終了状態となると，終了状態の実行状態は実行状態レジスタから取り除かれ，インタプリタコアは動作を停止する．インタプリタコアにおける処理の（単純な）強マイグレーションは，出力された実行状態およびコード格納領域を，移動先の *MiLogEngine* 上にマイグレーションすることで実現される．

実行状態レジスタが空でない状態でサスペンドフラグを立てると，インタプリタコアは，実行状態レジスタに格納された実行状態を実行状態スタックにプッシュし，停止する．実行状態スタックが空でなく，かつ実行状態レジスタが空の状態で，実行状態レジスタに空の問い合わせを意味する実行状態を設定すると，実行状態スタックから実行状態がポップされ，実行状態レジスタに設定される．実行状態レジスタが空でなくなるので，インタプリタコアは動作を開始する．割り込み処理は，インタプリタコアから出力された実行状態を実行状態スタックにプッシュし，割り込みで実行すべき問い合わせを表現した実行状態を，インタプリタコアの入力として与えることにより実現される．すなわち，*MiLogEngine* の行うことのできる基本的な処理は，実行状態レジスタの更新，実行状態の実行状態スタックへのプッシュ，実行状態スタックからの実行状態のポップの 3 つである（図 4.2）．任意時刻マイグレーションは，単純なマイグレーションの場合における実行状態のかわりに，実行状態スタック全体をマイグレーションさせることで実現される（図 4.3）．マイグレーションの任意時刻性は，実行状態の更新を基本実行単位として実現される．

プログラムの実行状態を移動先の実行環境へ転送する処理には，メタエージェントを利用する．移動元のメタエージェントは，エージェント（の持つ *MiLogEngine*）からの要求に応じて，エージェント（の持つ *MiLogEngine*）のプログラムと実行状態を移動先のメタエージェントに転送する．移動先のメタエージェントは，新たなエージェント（とそれに対応する *MiLogEngine*）を生成し，受け取ったプログラムと実行状態をエージェント（の持つ *MiLogEngine*）に復元する．

ここまでに述べたモデルは，対象言語を特に *MiLog* 言語に限定しない一般的なモデルである．本モデルを *MiLog* 言語の解釈実行系として実現するためには，*MiLog* 言語の実行に特化した実装方法を示す必要がある．本章の以降では，*MiLog* 言語に特化した本モデルの実装について示す．

4.4 実行状態のデータ構造

次に課題となるのは，本モデルに基づいて *MiLog* 言語の解釈実行を行うためのインタプリタコアの実現である．インタプリタコアを実現するためには，インタプリタコアに解釈実行可能であり，なおかつプログラムの基本実行単位ごとに整合性を保つことが可能となるような，実行状態のデータ構造を設計する必要がある．本節では，*MiLogEngine* における実行状態のデータ構造を示す．本節では，実行状態のデータ構造を示すための準備とし

て，最初に節や項などのプリミティブのデータ構造を示す．次に，プリミティブのデータ構造を用いて，実行状態のデータ構造を示す．

4.4.1 プリミティブの表現

プリミティブとは，節，項，あるいはもっと詳細なアトムやリスト等の基本的な *MiLog* プログラムの要素である．本節では，プリミティブのデータ構造を示す．すべてのプリミティブは，内部プリミティブオブジェクトの組み合わせによって表現される．内部プリミティブオブジェクトの構造を示す．

```
1  public class PrologObject implements Serializable {
2    public PrologObject car;
3    public PrologObject cdr;
4          int type;          // オブジェクトの種類
5    public int value;        // 値
6    public String name;      // 名前
7    public Object bin;       // 任意のオブジェクトを入れておける
8    ...
9  }
```

ここで，各行の先頭にある数字は行番号を示しており，記述にはJAVAの文法を用いている．オブジェクト PrologObject は，シリアライズ可能なオブジェクトである（1行目）．PrologObject は1種類のオブジェクトでアトム，リストおよび関数子等を表現する．アトムや変数は1つのオブジェクトで表現され，リスト，関数子，および節は複数のオブジェクトのリスト構造で表現される．リスト構造の表現のために， car 要素（2行目）および cdr 要素（3行目）が用いられる[4]．オブジェクトがどの種類のプリミティブを表現しているかは，type 要素を用いて表現する（4行目）．プリミティブが名前を持つ場合には，name 要素が用いられる（6行目）．数値の表現はその数値の文字列が name 要素に格納され，value 要素は処理の最適化のためのフラグとして用いられる．JAVAオブジェクトを効率よく処理できるようにするために，JAVAオブジェクト自身をプリミティブに格納することもできるようになっている（7行目）．

アトム a は，PrologObject を用いて次のように表現される．

```
PrologObject atom = new PrologObject(typeAtom);
atom.name = "a";
```

ここでは，a という名前を持つアトムを atom オブジェクトが表現しているときに，その atom オブジェクトの name 要素の値が文字列 "a" であり，type 属性がアトム (typeAtom) であることを示している．なお，type 属性の設定は，コンストラクタに引数を与えることで設定される．

リストは， type 属性に typeList を持つ PrologObject を用いて次のように表現される．

[4] 'car' と 'cdr' の表現は Lisp 言語のリスト操作関数に由来する

```
// [a, b, c] の表現
PrologObject list = new PrologObject(typeList);
list.car = new PrologObject(typeAtom);
list.car.name = "a";
list.cdr = new PrologObject(typeList);
list.cdr.car = new PrologObject(typeAtom);
list.cdr.car.name = "b";
list.cdr.cdr = new PrologObject(typeList);
list.cdr.cdr.car = new PrologObject(typeAtom);
list.car.cdr.car.name = "c";
list.cdr.cdr.cdr = null;
```

ここでは，3つの要素をもつリスト [a, b, c] が list というオブジェクトを先頭として表現されているとき，list の car 要素がリストの1番目の要素である a というアトムを表現する PrologObject への参照であり，残りの要素を格納したリストを表現する PrologObject への参照が cdr 要素に格納されることを示している．リストの末尾は cdr 要素が空 (null) のリストによって表現され，要素が空のリストは，car 要素および cdr 要素がともに空 (null) のリストである．

述語あるいは関数子は，type 属性に typeFunc を持つ PrologObject を用いて次のように表現される．例えば，引数が1つの場合には次のようになる．

```
// f(x) の表現
PrologObject func = new PrologObject(typeFunc);
func.name = "f";
func.cdr = new PrologObject(typeAtom);
func.cdr.name = "x";
```

引数が複数の場合には，**and 型 (typeAnd)** という特殊な型を用いて表現される．この and 型は，末尾要素の表現がリスト型と異なっており，要素が1つの場合にはその要素自身が格納され，格納される要素が2つ以上ある場合には，末尾の and 型の cdr 要素に最後の要素が格納される．例えば，引数が3つの場合には次のようになる．

```
// f(x, y, z) の表現
PrologObject func3 = new PrologObject(typeFunc);
func3.name = "f";
func3.cdr = new PrologObject(typeAnd);
func3.cdr.car = new PrologObject(typeAtom);
func3.cdr.car.name = "x";
func3.cdr.cdr = new PrologObject(typeAnd);
func3.cdr.cdr.car = new PrologObject(typeAtom);
func3.cdr.cdr.car.name = "y";
func3.cdr.cdr.cdr = new PrologObject(typeAtom);
func3.cdr.cdr.cdr.name = "z";
```

引数にand型を導入したのは，より少ないオブジェクトで引数を表現するためである．引数を表現するオブジェクトを少なくすると，節のコピーが高速に行えるという利点がある．

変数はとその変数束縛の表現を次に示す．

```
// 変数 X (と束縛値 a) の表現
PrologObject var = new PrologObject(typeVar);
var.name = "X";
var.type = typeVar;
var.cdr = new PrologObject(typeAtom);
var.cdr.name = "a";
```

変数は，その束縛内容を cdr 要素に格納する．未束縛の変数は，cdr 要素が空 (null) になる．なお，ユーザがデバッグ作業の際に参考にしてもらえるように，変数名を name 要素に格納してある．

ホーン節は，ヘッドとボディから構成される．ヘッドは関数型の表現として格納され，ボディは複数の関数型の表現を and 型で表現している．例えば，次のように表現される．

```
// ホーン節 "f(X,Z) :- g(X,Y),h(Y,Z)." の表現
PrologObject hornCl = new PrologObject(typeClause);
hornCl.car = new PrologObject(typeFunc);

... // 関数 f(X,Z) の表現は省略

hornCl.cdr = new PrologObject(typeAnd);
hornCl.cdr.car = new PrologObject(typeFunc);
hornCl.cdr.cdr = new PrologObject(typeFunc);

... // 関数 g(X,Y) と h(Y,Z) の表現は省略
```

節データベースは，複数のホーン節からなるリストとして，PrologObject を用いて表現される．ただし，節データベースはプログラムの実行速度を決める鍵になる部分なため，効率よく処理が行えるようにいくつかの工夫がしてある．節の検索処理を効率よく行うために，節データベースは同じ名前を持つヘッド毎の小さなリスト (coTable) に分割されて保存される．各 coTable の先頭には，要素の末尾への追加が高速に行えるように，末尾要素への参照が格納される．これらの構造を利用して，節データベースへのアクセスは高速に行われる．

節データベースは，クローンによって共有される場合がある．他のエンジンから直接節データベースのオブジェクトが共有可能となるように，節データベースおよび内部のリストの先頭は，ハンドル表現になっている．すなわち，節データベースの先頭は空の要素と実体の節データベースへの参照を持つリスト型の要素であり，実体の節データベースが改変された場合でも正しい内容を参照できるようになっている．

4.4.2 単一化処理と内部表現

MiLog プログラムの実行は，目標となる節と，それに単一化可能なヘッドを持つホーン節との単一化の繰り返しによって実現される．単一化の手続き的な意味は，目標となる節（目標節），およびその候補となるホーン節（候補ホーン節）における妥当な変数束縛の決定である．

単一化の手続きは，候補ホーン節の複写，候補ホーン節のヘッド部と目標節との単一化，および単一化の後処理の3つの段階から構成される．

候補ホーン節の複写段階では，候補ホーン節に対する単一化が可能なように複写される．候補ホーン節は単一化前の段階では節データベース内に格納されており，単一化によって節データベース内の情報が壊れないようにするために複写される．複写では，候補ホーン節中の変数について，同一の名前の変数が単一のオブジェクトとなるように複写される（変数のリネーミング処理）．たとえば，`f(X,Z) :- g(X,Y), h(Y,Z).` という節が複写される場合，変数 `X`, `Y`, および `Z` はそれぞれ単一のオブジェクトとして構築される．例えばヘッド部 `f(X,Z)` から参照される変数 `X` のオブジェクトと，ボディ部 `g(X,Y)` から参照される変数 `X` のオブジェクトは同一のオブジェクトとなるように複写される．候補ホーン節中の同一の変数を単一のオブジェクトで表現することで，ヘッド部に対する単一化による変数束縛結果は特別な操作なしにボディ部に反映される．目標節が仮に同名の変数を含む場合でも，変数を表現するオブジェクトが異なるため，変数名によらずこれらの変数は別のものとして扱われる．

候補ホーン節のヘッド部と目標節との単一化段階では，単一化による変数束縛の変化がトレイルスタックに記録される．例えば，`f(a,b)` という目標節と `f(X,Z) :- g(X,Y), h(Y,Z).` という候補ホーン節のヘッド部 `f(X,Z)` との単一化を行う場合には，変数 `X` および `Y` にそれぞれ `a` および `b` が束縛されると同時に，トレイルスタックの先頭に変数 `X` および `Y` に対する参照が追加される．先に示したように，候補ホーン節中の同一の変数は単一のオブジェクトで表現されるため，即座に候補ホーン節のボディ部に変数束縛結果が反映され，ボディ部は `g(a,Y), h(Y,b)` となる．

単一化の後処理の段階は，単一化が途中で失敗した場合，および単一化が成功した場合についてそれぞれ異なる処理を行う．単一化が途中で失敗した場合には，トレイルスタックの先頭がポップされ，そこから参照されるすべての変数について，変数束縛が解除される．単一化が成功した場合には，単一化が成功したことを示すと同時に，候補ホーン節以後に定義されていて単一化を行っていない候補ホーン節の集合を返す．

これらの処理をアルゴリズムとしてまとめたものを，図4.4に示す．図4.4では，文法にJAVAに似た表現を用いて，目標節の展開および単一化のアルゴリズムの概要を示している．

4.4.3 実行状態の表現

1つの実行状態は，トレイルスタック，制御スタック，および選択点スタックから構成される．トレイルスタックの構造については既に4.4.2節で述べた．本節では，制御スタックおよび選択点スタックの構造と内部表現について述べる．

```
PrologObject 目標節の展開( 目標節, 候補ホーン節のリスト ) {
  while( 候補ホーン節のリスト != null ) {
    PrologObject 候補ホーン節 =
          候補ホーン節のリストの先頭要素;
    候補ホーン節のリスト = 候補ホーン節のリスト.cdr;
    候補ホーン節 = 複写と変数リネーミング( 候補ホーン節 );
    トレイルスタックに空のリストをプッシュ;
    if( 単一化( 目標節, 候補ホーン節 ) == 成功 ) {
      return( 候補ホーン節のリスト );
    } else {
      トレイルスタックの先頭をポップ;
      そこから参照されるすべての変数の束縛を解除;
      return(失敗)
    }
  }
}
PrologObject 複写と変数リネーミング( 候補ホーン節 ) {
  使用変数の一時リストを初期化;

  再帰的に複写と変数リネーミングを行う.
  変数の複写時には, 使用変数の一時リストを確認
    リストにない変数なら, 新たにコピーを生成してリストに追加
    リストにある変数なら, リストの内容をコピー内容とする

  return( 複写後の候補ホーン節 );
}
PrologObject 単一化( 目標節, 候補ホーン節 ) {

  再帰的に目標節と候補ホーン節を単一化する.
  もし変数に値を束縛する場合には,
    トレイルスタックにその変数への参照を追加

  return( 単一化の成否 );
}

PrologObject 使用変数の一時リスト;

PrologObject トレイルスタック;
```

図 4.4: 単一化に関連する処理のアルゴリズムの概要

制御スタックは，Boxを要素とした木構造として表現される．各Boxはそれぞれ目標節の実行状態を表現している．Box間は1) 目標とサブ目標というリンクによって相互接続される．個々のサブ目標はさらにそのサブ目標を持つため，制御スタックは全体としては木構造になる．Boxのデータ構造を示す．

```
public class Box
{
  public PrologObject goal;              // 目標節
  public PrologObject currentClauses;    // 候補ホーン節リスト
  public Box          body[];            // サブ目標節のリスト(配列)
  public Box          parent;            // 親の目標節
  public int          nParent;           // 親の目標節の body[] での順位
  public Box          next;              // (選択点スタックの構成時に使用)
  public int          nBody;             // サブ目標節中での実行位置
  public int          id;                // Box の ID
  public int          result;            // Box の実行結果 or 実行途中
  ...
}
```

goalには，そのBoxの目標節が格納される．たとえば，目標節がf(a,b)なら，f(a,b)のPrologObjectによる表現がgoalに格納される．currentClausesには，そのBoxの目標節に対する候補ホーン節リストが格納される．候補ホーン節リストとは，その目標節に対して別解が存在する可能性がある場合の，残りの候補ホーン節のリストである．例えば，f(X,Y) :- a(X). f(X,Y) :- b(X). という節が定義されており，現在の目標節と単一化された節がf(X,Y) :- a(X). である場合，残りの候補ホーン節は，f(X,Y) :- b(X). となる．body[]には，Boxの目標節から展開されたサブ目標節を表現するBoxが格納される．例えば，目標節がf(a,b)で単一化された候補ホーン節がf(X,Z) :- g(X,Y), h(Y,Z). の場合，g(a,Y)およびh(Y,b)を目標節とするBoxがそれぞれ生成され，body[0]とbody[1]に格納される．body[]が配列として表現できるのは，ホーン節のボディ部分の表現形式を','による連言に限定しているからである．parentには，そのBoxをサブ目標として持つBoxへの参照が格納される．たとえば先の例で，body[0].parentおよびbody[1].parentには，f(a,b)を目標節に持つBoxへの参照が格納される．nParentは，parentでのbody[]における自身の配列要素番号が格納される．たとえば先の例では，body[0].nParent == 0およびbody[1].nParent == 1となる．nBodyは，そのBoxのサブ目標節の何番目を実行中かを表現している．たとえば先の例でg(a,Y)のサブ目標節を実行中である場合には，nBody == 0となる．idは，そのBoxのIDを格納する．ここで，BoxのIDとは，そのBoxが生成された時刻を示すものであり，新しく生成されたBoxほど大きなIDの値を持つ．BoxのIDは，トレイルスタックの最適化等に利用される．resultは，そのBoxの目標節に対する実行結果を示す．まだ実行中で結果が出ていない場合にはNOTSET，実行が成功した場合にはTRUE，実行が失敗したがまだ候補ホーン節リストが空でないときにはREDO，実行が失敗し候補ホーン節リストも空の場合にはFAILがそれぞれresultに格納される．トップレベルの(根ノードとなる)Boxは，*MiLogEngine*に対する問い合わせを

束ねるためのダミー節である．例えば，問い合わせとして `?- a(X,Y), b(Y,Z), c(Z,X).` という問い合わせが与えられた場合，`top_level :- a(X,Y), b(Y,Z), c(Z,X).` という節が節データベースに一時的に追加された後，問い合わせ `?- top_level.` がエンジンで実行される．ここでの `top_level` という名前は，実際にはユーザ定義述語と重複しない適当な節になるようにシステムで自動的に名前が付けられる．ダミー節表現を用いることにより，複数の節の連言を問い合わせとして扱うことが可能となっている．

制御スタックは，トップレベルの Box (`topBox`) および現在実行中の Box (`currentBox`) の2つの参照を用いて利用される．`topBox` は，問い合わせの実行処理過程全体へのポインタになっている．`currentBox` はプログラム中の現在の実行位置を示し，インタプリタコアによるステージの処理ごとに更新される．各 Box は，`nBody` と `result` の値から，対応するインタプリタコアのステージが一意に決まるようになっている．すなわち，次に処理すべき Box が定まれば，インタプリタコアで次に行うべき処理は一意に定まる．これら2つの参照を用いることで，プログラム全体の実行状態および現在の実行位置の両方が参照可能なオブジェクトの集合として表現可能となっている．

選択点スタックは，制御スタック中で次にバックトラック可能な Box を格納するスタックである．ここで，選択点とは，バックトラック可能な候補ホーン節をもつ Box を意味する．*MiLogEngine* の制御スタック表現では，選択点スタックなしでも次にバックトラックすべき選択点を探索可能であるが，選択点スタックを追加することにより，選択点の探索コストを低減し，動作速度を高速化することが可能となる．選択点スタックには，Box がスタック状に積まれる．選択点スタックの表現には Box そのものをデータ表現として用いており，実際に選択点スタックとして存在するのは，選択点スタックの先頭を示すポインタのみである．選択点スタックの先頭のポインタは，次に選択点としてバックトラック可能な Box への参照を持っている．次の次に選択点としてバックトラック可能な Box への参照は，先の Box の定義中にあった `next` 要素に格納される．選択点スタックは以上の構造をもつため，数パーセント程度の実行状態の増加を代償として，定数の処理オーダで次の選択点を決定することができる．

なお，カットオペレータの動作については，カットオペレータの Box を `cb` とすると，`cb.parent.id` で示される ID よりも大きな ID をもつ Box を選択点スタックから削除する操作として定義できる．

以上で示したように，制御スタック，選択点スタックおよびトレイルスタックの内容はすべて JAVA オブジェクトとして表現可能となっており，その中にすべての実行の継続に必要な情報を含んでいる．これら3つのスタックの内容を節データベースとあわせてビット列に変換することにより，実行状態が保存および復元可能となる．

4.5 インタプリタコアの設計

4.5.1 インタプリタコアのアーキテクチャ

インタプリタコアの内部は，Box モデル [27] に基づいて，12の処理段階（ステージと呼ぶ）に分割されている．入力された実行状態は，インタプリタコア内の適切なステージへ

送られ，そのステージにおけるの処理によって新たな実行状態が出力される[5]．

MiLogEngine のインタプリタコアの処理の概要を図4.5に示す．インタプリタコアのメインループ（doIteratively メソッド）は，1つの while ループによって構成され，内部で12のステージに分岐している．現在実行中の Box は，プログラムカウンタの b から参照される（1行目）．現在実行中の Box の処理に対応するステージは，nBody 属性値（現在実行しているボディの位置／負の値のときは前処理中を意味する），および result 属性値（Box の実行成功失敗等を意味する）により決定される．1回のループで1段のステージが実行され，Box の内容とプログラムカウンタが更新される．インタプリタコアのループは，すべての Box について処理を終える（問い合わせ成功あるいは失敗）か，あるいは外部からサスペンドフラグ（signal の値）を書き換えられるまで続く（3行目）．

任意時刻モビリティを実現するためには，インタプリタの実行の中断（サスペンド）およびサスペンド中での別の問い合わせの実行を可能とする必要がある．サスペンドフラグは，インタプリタの実行をサスペンドするために用意される．サスペンドフラグを立てると，インタプリタコアは，実行中の問い合わせについて現在のステージにおける処理を終えたあと，実行状態を実行状態スタックへプッシュして，待機状態となる．インタプリタコアは，サスペンド後の待機状態において，サスペンドした実行状態を破壊することなく，サスペンドした問い合わせとは別の問い合わせを実行可能である．中断した問い合わせを再開するには，待機状態の *MiLogEngine* に空の問い合わせを送る．*MiLogEngine* は，空の問い合わせを受け取ると，実行状態スタックから実行状態を1つポップし，取り出した実行状態を入力として実行サイクルを再開する．

MiLogEngine の問い合わせ起動部分の処理の概要を図4.6に示す．現在の問い合わせの実行状態は，エンジンの属性値として参照可能となっている（1行目）．1行目で，goal は現在の問い合わせの実行状態をトップレベルのゴールを頂点としたオブジェクトのツリーによって表現しており，lastResult には，最後に実行した問い合わせの問い合わせ結果（変数束縛内容など）が保存される．エンジンへの問い合わせの起動は，スレッドを起動することにより行われる．通常の問い合わせ実行時には，'queryString' に問い合わせを文字列として代入し，スレッドを起動する．問い合わせ実行時には，まず問い合わせ文字列が解釈され（8行目），次に解釈された問い合わせが 'doIt' メソッド[6] により実行される（16行目）．'queryString' が null の場合にはレジュームであると判断し，実行状態スタックから実行状態を復元し（21行目），その実行を継続する（23行目）．実行後にフラグを調べ（25行目），中断でない場合には，実行結果を表示するための最小限の情報以外を開放する（26行目）．サスペンドフラグの操作による実行中断の場合には，現在の実行状態を実行状態スタックにプッシュする（28行目）．

エージェントの強モビリティは，待機状態の *MiLogEngine* の実行状態スタックおよび節データベースの内容を保存し，それを移動先の *MiLog* 実行環境上に用意した *MiLogEngine* 上に復元することにより実現される．

---

[5] ここでのステージとは，CPU でのパイプライン処理におけるステージのような複数のステージを並列に動作させる性質のものではなく，単に処理段階を意味している．

[6] 実際のソースコードでは別名のメソッドであるが，そのメソッド名はエンジンのバージョン番号等を含んだ直感的にわかりにくい名前のため，本文中では仮に doIt という名前を用いた．

```
1  Box b; int signal; // プログラムカウンタとサスペンドフラグ
2  void doIteratively() {
3    while( b != null && signal == 1 ) {
4      if( b.result == b.NOTSET ) {
5        if( b.nBody == -1 ) {
6          //... 実行前処理
7        } else if( b.nBody == -2 ) {
8          // モードチェック
9        } else if( b.nBody == -3 ) {
10         //... サブゴールの展開
11       } else if( b.nBody == -4 ) {
12         //... 組み込み述語実行処理
13       } else if( b.nBody == -5 ) {
14         //... カットの処理
15       } else if( b.nBody == -6 ) {
16         //... call(X) の処理
17       } else if( b.nBody >= 0 && b.nBody < b.bodyLength ) {
18         //... ボディの実行
19       } else if( b.nBody >= b.bodyLength ) {
20         //... 全ボディの実行成功
21       } else { // ...内部例外処理 }
22     } else if( b.result == b.TRUE ) {
23       // ... 実行成功
24     } else if( b.result == b.FAIL ) {
25       // ... 実行失敗
26     } else if( b.result == b.REDO ) {
27       // ... バックトラック準備
28     }
29   }
30 }
```

図 4.5: インタプリタコアのメインループ

```
1   PrologObject goal,rhs,lastResult; String queryString;
2   public final void run() {
3     if( queryString != null ) { // 問い合わせ実行
4       String queryString = this.queryString;
5       this.queryString = null;
6       init(); // 変数領域等の初期化
7       try{
8         goal = PrologParser.parser(":- !," + queryString);
9       } catch( Exception e ) {
10        out("\nSyntax Error!!\n" + queryString + "\n");
11        return;
12      }
13      rhs = goal.car.cdr;
14      // 問い合わせが正しく解釈できたらそれを実行する
15      if( rhs != null && rhs.cdr != null ) {
16        lastResult = doIt(rhs);
17      } else {
18        return;
19      }
20    } else { // レジューム
21      popCurrentStatus();
22      signal = 1;
23      lastResult = doIt(rhs);
24    }
25    if( signal >= 1 ) {
26      freeCurrentStatus(); // 実行終了時
27    } else if( signal == 0 ) {
28      pushCurrentStatus(); // サスペンド時
29    }
30  }
```

図 4.6: 問い合わせ起動部分の処理

4.5.2 インタプリタコアにおける各ステージの動作

インタプリタコアの設計の特長は，各ステージごとの処理を，できるだけ単純かつ重複のない処理となるように設計した点である．各ステージを単純な処理にすることで，各ステージごとでの処理の最適化を行いやすいようにした．本設計では，各ステージに重複した処理を持たせれば，ステージ間の移行を減らすことが可能であが，*MiLogEngine* をできるだけ小さなコードサイズで実現することを重視し，ステージ間の移行の削減は行わなかった．図4.7に，インタプリタコアにおけるステージとステージ間の移行関係を示す．以下では，インタプリタコアにおける各ステージの動作と各ステージ間での処理の移行の流れを示す．

実行の開始

トップレベルの問い合わせが，初期状態の Box としてインタプリタコアに与えられる．初期状態の Box の処理は，実行前処理ステージ (Init) から開始される．

実行前処理ステージ (Init)

実行前処理ステージ (Init) では，Box の初期化処理として，単一化の候補節に節データベース全体を設定する．実行前処理の終了後は，モードチェックステージ (ModeChk) に移行する．

モードチェックステージ (ModeChk)

モードチェックステージ (ModeChk) では，現在の Box で実行すべき問い合わせ（カレントゴール）の種類について判別する．モードチェックステージによる判別後に，組み込み述語 (Built-in)，カットオペレータ (Cut)，call 実行 (BCall)，および通常のサブゴール展開 (MT CDB) のうちで適切なステージに移行する．

サブゴールの展開ステージ (MT CDB)

サブゴールの展開ステージ (MT CDB[7]) では，カレントゴールと単一化可能なヘッドを持つ節を候補節から検索し，単一化処理を行う．単一化に失敗した場合には，現在の Box に失敗フラグをつけ，実行失敗ステージ (Fail) に移行する．単一化処理が成功した場合には，候補節の残りを現在の Box の候補節に設定し，現在の Box を選択点スタックにプッシュする．単一化処理が成功した場合で，単一化したのがファクト（ボディを持たない）の場合には，現在の Box に成功フラグをつけ，実行成功ステージ (True) に移行する．単一化したのがボディを持つ通常の節の場合には，ボディを新たなゴールとするために Box を生成し，サブゴールの実行位置を示す nBody に先頭を示す '0' を設定して，ボディの実行ステージ (CC) へ移行する．

[7] Match To Clause DataBase の略である．

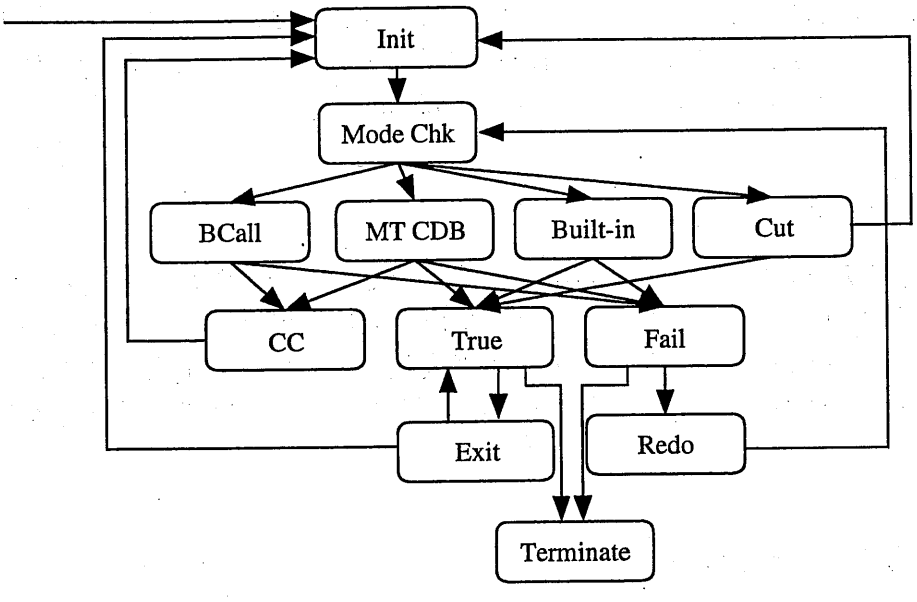

図 4.7: インタプリタコアにおけるステージ間の関係

組み込み述語実行処理ステージ (Built-in)

組み込み述語実行処理ステージ (Built-in) では，カレントゴールに相当する組み込み述語の実行を行う．組み込み述語の実行が成功した場合には，現在の Box に成功フラグをつけて実行成功ステージ (True) に，失敗した場合には失敗フラグをつけて実行失敗ステージ (Fail) へそれぞれ移行する．

カットの処理ステージ (Cut)

カットの処理ステージ (Cut) では，現在の Box の親の Box 以降に生成された選択点を，選択点スタックからポップする．現在の Box と同一の親を持つ Box で現在の Box 以前に実行されたもの（兄 Box）は，この後の処理では参照されないので，Box の内容を破棄する．トレイルスタックの最適化および，末尾呼び出し最適化の適用を行う．末尾呼び出し最適化の適用により次に実行すべき Box が変更された場合には，その Box を次に処理する Box として設定して実行前処理ステージ (Init) に移行する．そうでない場合には，現在の Box に成功フラグをつけて実行成功ステージ (True) に移行する．

call 実行ステージ (BCall)

call 実行ステージでは，call の第1引数として指定されたゴールが未束縛の変数でない場合に，そのゴールを持つ Box を新たに生成して次に処理すべき Box に設定し，ボディの実行ステージ (CC) に移行する．もし未束縛変数あるいはリストなどの実行不可能なゴールが指定されていた場合には，現在の Box に失敗フラグをつけて実行失敗ステージ (Fail) に移行する．

ボディの実行ステージ (CC)

ボディの実行ステージ (CC[8]) では，実行位置を示す nBody で示された Box を次に処理すべき Box に設定し，実行前処理ステージ (Init) に移行する．

全ボディの実行成功ステージ (Exit)

全ボディの実行成功ステージ (Exit) では，現在の Box に成功フラグをつけて実行成功ステージ (True) に移行する．

実行成功ステージ (True)

実行成功ステージ (True) では，次に処理すべき Box に親の Box を設定するとともに，親の Box で次に処理すべき Box があればそれを初期化する．親 Box の状態に応じて，ボディの実行ステージあるいは全ボディの実行成功ステージ (Exit) に移行する．もし現在の Box

[8] Call Current の略である．

がトップレベルのBoxであった場合には，次に処理すべきBoxは空となり，トップレベルの処理ループから抜ける (Terminate).

実行失敗ステージ (Fail)

実行失敗ステージ (Fail) では，選択点スタックから先頭要素をポップする．取り出した選択点を次に処理すべきBoxに設定する．選択点以後に生成されたBoxのツリーがあれば削除する．選択点のBoxにバックトラックフラグをつけて，バックトラック準備ステージ (Redo) に移行する．もし，選択点スタックがすでに空の場合には，トップレベルのBoxに失敗フラグをつけ，次に処理すべきBoxを空にすることにより，トップレベルの処理ループから抜ける (Terminate).

バックトラック準備ステージ (Redo)

バックトラック準備ステージ (Redo) では，現在のBox以後に行われた変数束縛を解除し，モードチェックステージ (Mode Chk) に移行する．

問い合わせ終了ステージ (Terminate)

問い合わせ終了ステージでは，インタプリタコアの処理ループを抜け，終了処理を行って問い合わせ結果を返す．

4.6 末尾呼び出し最適化 (Last Call Optimization)

4.6.1 *MiLogEngine* における末尾呼び出し最適化

末尾呼び出し最適化 (Last Call Optimization) は，WAMの実装において，節のボディ末尾の呼び出し時に，環境を再利用する最適化手法である．末尾呼び出し最適化は末尾再帰最適化をより一般化したものであり，再帰プログラミングを基本とするProlog処理系では非常に有用な最適化手法である．WAMにおける末尾呼び出し最適化の実装では，PrologコードのWAMコードへのコンパイル時に，プログラムの流れの決定性を解析することにより，最適化の適用可能性を判別する．最適化可能と判別されたコードは，最適化を伴った呼び出し命令 (exec) としてWAMコード中で表現され，実行される．すなわち，WAMにおける末尾呼び出し最適化は，コンパイル時のコード解析および実行時の最適化命令の実装によって実現される．*MiLogEngine* は，コードを別の意味論を持つコードへコンパイルせず，ソースコードと同一の意味論で保持する．*MiLogEngine* では，WAMの実装のようにコンパイル時における最適化命令の埋め込みは行われず，実行時にコードを動的に解析して最適化を行う．実行時におけるコードの動的解析では，コード解析にかかるコストと最適化によって得られる性能向上とのトレードオフがある．コード解析を頻繁に行った場合，最適化によって得られる性能向上以上にコード解析に多くの時間を割いてしまう可能性がある．末尾呼び出し最適化は実行速度の高速化よりむしろメモリ使用量の抑制に重点

を置いており，マイグレーション時の実行状態転送のコスト削減のために積極的にコード解析を行うことには利点がないわけではないが，実行速度はより高速であるほうが好ましい．*MiLogEngine*では，カットオペレータ適用時にコード解析を限定することにより，コード解析にかかるコストと最適化による利点とのバランスを取っている．

*MiLogEngine*における末尾呼び出し最適化は，以下のプログラム変換規則に基づく．

変換規則1 カットオペレータでない任意の述語 A, C, T，および カットオペレータを含む任意の述語 $B_0, ..., B_n, D_0, ..., D_n$ から構成される次の形式のホーン節

$$A : -B_0, ..., B_n, !, C.$$

$$C : -D_0, ..., D_n, !, T.$$

によって構成される論理プログラムに対して，問い合わせ

$$? - A.$$

が与えられ，その実行が T の手前のカットオペレータ（!）に到達したとき，それ以後のプログラムの実行は，

$$A : -B_0, ..., B_n, !, T.$$

における T の位置からの実行として変換可能である．

変換規則2 カットオペレータでない任意の述語 A, C, T，およびカットオペレータを含む任意の述語 $B_0, ..., B_n, D_0, ..., D_n, X_0, ..., X_n$ から構成される次の形式のホーン節

$$A : -B_0, ..., B_n, !, C, !, X_0, ..., X_n.$$

$$C : -D_0, ..., D_n, !, T, !.$$

で構成される論理プログラムに対して，問い合わせ

$$? - A.$$

が与えられ，その問い合わせの実行が T の手前のカットオペレータ（!）に到達したとき，それ以後のプログラムの実行は，

$$A : -B_0, ..., B_n, !, T, !, X_0, ..., X_n.$$

における T の位置からの実行として変換可能である．

変換規則1および2は，Boxモデルに基づく制御フロー解析によって正しいことが容易に証明可能である．ただし，変換規則1および2では，変換の対象は制御スタックのみであり，変数束縛の処理等は対象外とする．

本変換規則に基づく最適化は，インタプリタコアのCutの処理ステージ内で，図4.8に示すアルゴリズムにより実装される．図4.8で最初のif文中が変換規則1の実装であり，次のelse if文中が変換規則2の実装である．

```
if(
 (b.nParent == b.parent.bodyLength-2) &&              // 最後から2番目
 (isCut( b.parent.body[b.nParent+1].goal)==false) &&  // 次が ! 以外
 (b.parent.parent != null) &&                         // 親がtop でない
 (b.parent.nParent > 0) &&                            // 親が先頭要素以外
 (isCut( b.parent.parent.body[b.parent.nParent-1].goal )) // 親の前が !
 ) {
 Box youngerBrother = b.parent.body[b.nParent+1]; // 弟
 Box parent = b.parent;                           // 親
 // 弟を親の代わりにする
 youngerBrother.nParent = parent.nParent;
 youngerBrother.parent  = parent.parent;
 parent.parent.body[parent.nParent] = youngerBrother;
 // 弟を次の実行目標Boxにする
 b = youngerBrother;
} else if(
 (b.nParent == b.parent.bodyLength-3) &&      // この ! が最後から3番目
 (isCut( b.parent.body[b.nParent+2].goal ) && // 最後が !
 (b.parent.parent != null) &&                 // 親が top でない
 (b.parent.nParent > 0) &&                    // 親が先頭以外
 (isCut( b.parent.parent.body[b.parent.nParent-1].goal)) && // 親の前が!
 (b.parent.nParent+1 < b.parent.parent.bodyLength) &&   // 親の後が存在
 (isCut( b.parent.parent.body[b.parent.nParent+1].goal)) && // 親の後が!
 (isCut( b.parent.body[b.nParent+1].goal ) == false) // 次が!でない
       ) {
 Box youngerBrother = b.parent.body[b.nParent+1];
 Box parent = b.parent;
 // 弟を親の代わりにする
 youngerBrother.nParent = parent.nParent;
 youngerBrother.parent  = parent.parent;
 parent.parent.body[parent.nParent] = youngerBrother;
 // 弟を次の実行目標Boxにする
 b = youngerBrother;
}
```

図 4.8: 末尾呼び出し最適化のアルゴリズム

4.6.2 最適化の効果とオーバヘッドの評価

末尾呼び出し最適化アルゴリズムの実装を評価するために，2つの実験を行った．1つは，本最適化アルゴリズムの実装によるコード解析オーバヘッドの評価である．もう1つは，本最適化アルゴリズムの実装による制御スタック縮小化の評価である．実験は，500MHzで駆動する Intel Mobile Celeron CPU と 320MBytes のメモリを搭載するノート型PC上で，オペレーティングシステムに Windows 2000 ServicePack2，JAVA 実行環境に SUN JAVA2 1.3.1_02 HotSpot ClientVM を用いて行った．

実行時におけるコード解析オーバヘッドの評価を行うために，いくつかのベンチマークプログラムに対して，本最適化を有効にした場合，本最適化におけるコード解析処理のみを有効（最適化自身は無効）にした場合，および本最適化を無効にした場合での実行時間を計測した．なお，制御スタック縮小化の計測では，同一計算機上に2つの *MiLog* 実行環境を起動して計測を行った．表4.1に，計測結果を示す．ベンチマークプログラムには，3つの代表的なベンチマークと1つの恣意的に作成したベンチマークプログラムを用いた．3つのベンチマーク nreverse, 8queen(first), 8queen(all) はそれぞれ，リストの反転，8クイーン問題の初期解探索，8クイーン問題の全解探索を行う．これ以外に，恣意的に多くのカットオペレータを含ませたプログラム (bench4) をベンチマークに加えた．bench4 のプログラムを図4.9に示す．

表4.1で，括弧の中の数値は，最適化が無効の場合を1とした場合の実行時間の比を示している．最適化が無効の場合とコード解析のみを有効にした場合では，実行速度の差は高々1%程度であった．最適化が有効の場合には，nreverse で 6% 高速になっているがそれ以外では高々 3% 程度高速になった．最適化が有効になった際に実行速度が高速化されるのは，使用メモリが減少することによってガベジコレクタの負担が減る点と，最適化によって実行すべきステージ数がわずかに減少することが原因と考えられる．ここでの実行速度の差は非常に小さいものであり，通常のプログラミングでは実質的な実行速度の差はないといえる．カットを意図的に多く含ませたベンチマーク (bench4) では，最適化を有効にした場合に，他の結果とは逆に 1% 程度動作速度が低下している．これは，本最適化の適用によって実行速度が低下する場合があることを示している．本ベンチマークは恣意的に最適化に不利な構造としたにもかかわらず，それによる実行速度の低下は 1% と十分に小さい．本最適化で用いたカットオペレータ適用時におけるコード解析のオーバヘッドは，プログラム全体の実行速度にはほとんど影響を与えない程度の小さなものであるといえる．

制御スタック縮小化の効果を計測するために，2つの計算機間を往復する単純な *MiLog* プログラムを作成し，本最適化を有効にした場合と無効にした場合について，往復ごとのエージェントマイグレーションの転送データサイズを計測した．往復プログラムは，*MiLog* 言語で以下のように記述される．

```
goAndBack(0) :- !.
goAndBack(N) :-
  there(THERE),!,move(THERE),!,M is N-1,!,goHome,!,goAndBack(M).
```

ここで，述語 there/1 は，移動先のアドレスを格納した述語であり，計測開始時に節データベースに適切な値が挿入される．計測結果を図4.10に示す．図4.10のグラフで，縦軸はエージェントの転送データサイズ（単位は bytes），横軸はエージェントの往復移動回数

```
bench4 :-
  bench3,!,bench3,!,bench3,!,bench3,!,bench3,!,
  bench3,!,bench3,!,bench3,!,bench3,!,bench3,!.
bench3 :-
  bench2,!,bench2,!,bench2,!,bench2,!,bench2,!,
  bench2,!,bench2,!,bench2,!,bench2,!,bench2,!.
bench2 :-
  bench1,!,bench1,!,bench1,!,bench1,!,bench1,!,
  bench1,!,bench1,!,bench1,!,bench1,!,bench1,!.
bench1 :-
  bench0,!,bench0,!,bench0,!,bench0,!,bench0,!,
  bench0,!,bench0,!,bench0,!,bench0,!,bench0,!.
bench0 :-
  bench01,!,bench01,!,bench01,!,bench01,!,bench01,!,
  bench01,!,bench01,!,bench01,!,bench01,!,bench01,!.
bench01 :- my_append([1,2,3,4,5,6,7,8,9,0],[a],X),!.

my_append([],X,X) :- !.
my_append([X|L],Y,[X|Z]) :- !,my_append(L,Y,Z).
```

図 4.9: 意図的にカットを多く含ませたベンチマーク (bench4)

表 4.1: 末尾呼び出し最適化とコード解析オーバヘッドの計測結果

	最適化無効 [msec]	コード解析のみ [msec]	最適化有効 [msec]
nreverse	26614(1.00)	26347(0.99)	24982(0.94)
8queen(first)	762(1.00)	767(1.01)	737(0.97)
8queen(all)	12840(1.00)	13024(1.01)	12663(0.99)
bench4	57963(1.00)	57978(1.00)	58641(1.01)

(単位は「往復回数」) を示している．本プログラムは再帰を用いて定義されているため，最適化が無効の場合には，移動回数が増えるごとに再帰呼び出しによる制御スタックの増加が発生し，転送データサイズが増加している．最適化が有効な場合には，制御スタックが縮小化されるため，転送データサイズが移動回数に対してほとんど変化していない．本最適化の適用により，効果的に制御スタックが縮小化されている．

4.7 APIの実装

本節では，*MiLogEngine*における，エージェント間通信，割り込み処理，および任意時刻モビリティの機能をユーザプログラムから利用可能にするため機能を，*MiLog*言語上から利用可能な述語として実装するための方法を示す．本節の以降では，説明上の混乱を避けるため，**エージェント間通信，割り込み処理および任意時刻モビリティ等の機能をユーザプログラムから利用可能にする述語のことを，API (Application Programming Interface) 述語，あるいは単に API と呼ぶことにする．**

API述語の実装で解決すべき課題に，API述語実行時におけるモビリティの実現がある．具体的には，API述語の実行中に任意時刻マイグレーションを可能とすること，およびAPI述語の実行の一部分としてマイグレーションを含ませることを可能とすることである．API述語として実現すべき機能には，問い合わせ述語 query/2 のように実行が即座に終了しない機能が含まれている．これらの述語の実行によってマイグレーションの即時性が著しく制限されると，任意時刻マイグレーションを実現する利点が薄れてしまう．また，マイグレーションに付加的な機能（たとえば，ユーザインタフェースの保存など）を追加した述語や，頻繁に使う状況での引数の省略機能（たとえば，エージェントが生成された実行環境に帰還する場合など）を持ったマイグレーション述語が提供されると，ユーザにとってプログラミング上の利便性が向上すると考えられる．これら2つの目的を達成するために，*MiLogEngine*では，API述語の実装そのものを*MiLog*言語でプログラミング可能とするアーキテクチャを実現する．

4.7.1 APIの実装アーキテクチャ

*MiLogEngine*では，APIをコアプログラム層，システムプログラム層，およびユーザプログラム層の3つの層によって構成する（図4.11）．コアプログラム層では，少数の組み込み述語（コアAPI）がJAVA言語を用いて実装され，システムプログラム層では，実際にエージェントプログラム開発者が利用する述語群（システムAPI述語）が*MiLog*プログラムによって実装される．システムAPI述語は，インタプリタコア内部に含まれるもう1つのインタプリタ（Co-Interpreter）によって解釈実行される．Co-Interpreterが内部に持つプログラムは，ユーザプログラム層から隠される．Co-Interpreterの実行状態は，親となっているインタプリタコアの実行状態に付随したものとして扱われ，サスペンド時には親のインタプリタコアの実行状態に付随して実行状態スタックに格納される．本アーキテクチャにより，システムプログラム層で記述されている述語の実行中にも任意時刻モビリティが可能となり，モビリティを内包した述語の設計も容易となる．本アーキテクチャに

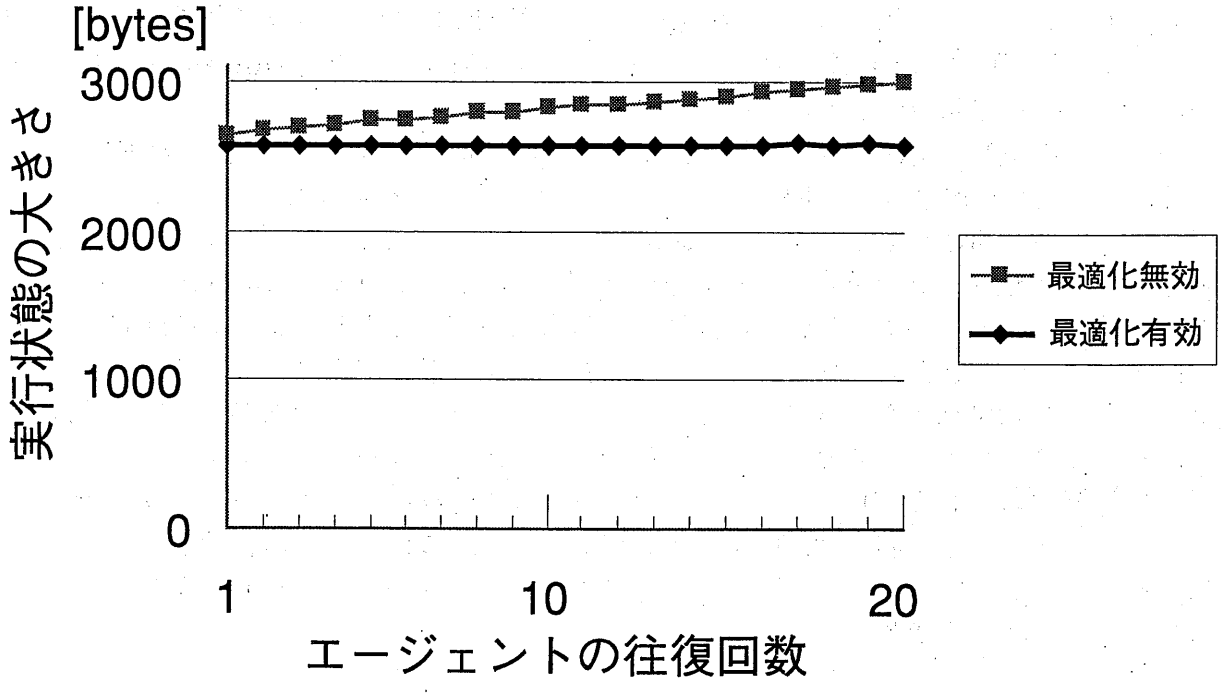

図 4.10: 末尾呼び出し最適化による制御スタックの縮小化

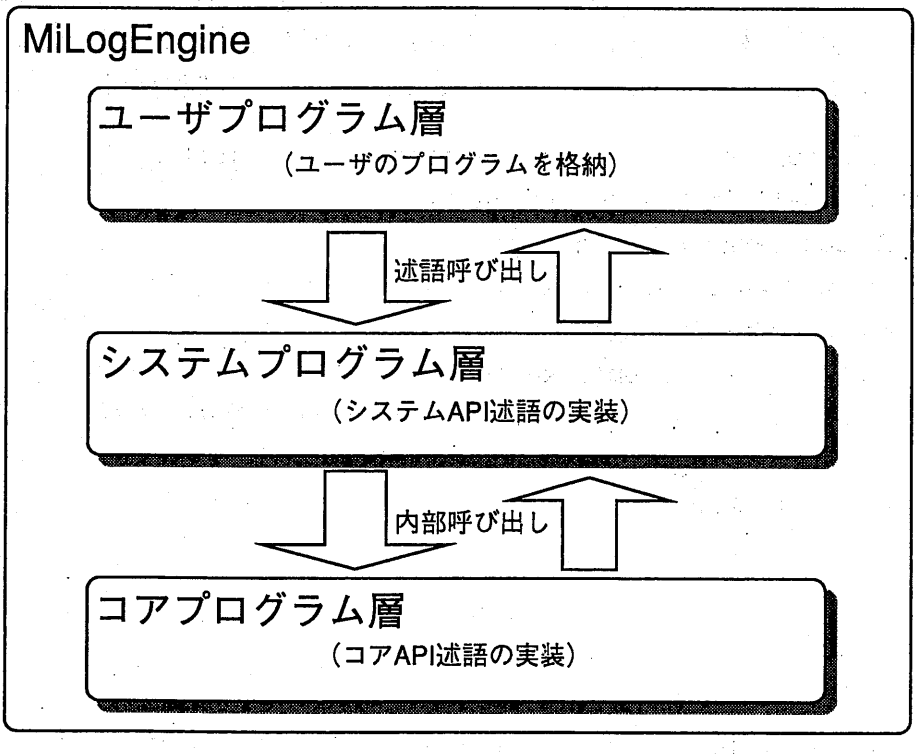

図 4.11: *MiLogEngine* における API の階層構造

は別の利点として，インタプリタコアで用意するコアAPIの実装を，3.3.1節で定義した述語群の実装に必要な最小限の機能のみに限定することにより，インタプリタコアの設計が簡単となる点がある．本節では，以降でコアAPI述語とシステムAPI述語の実装の詳細を示す．

4.7.2 コアプログラム層の設計

コアプログラム層が実装する代表的なコアAPI述語のリストを表4.2に示す[9]．表4.2中の1から4のコアAPI述語（'system_request/2'，'system_remoteAssert/2'，'system_myname/1'，'system_thereExists/1'）は，主にエージェント間問い合わせの実現に利用される．述語'system_request/2'は，他のエンジンに対して問い合わせの起動を指示する．第1引数にエンジン名，第2引数に問い合わせ内容を指定する．エンジンは，システム内部にある名前テーブルを参照することで，対象となるエンジンを特定し，そのエンジンが待機状態であれば，問い合わせの実行を開始させる．問い合わせの実行が開始されれば，述語は真となる．述語'system_request/2'の実装の概要は，次のようになる．

```
1  public PrologObject sys_system_request(PrologObject goal) {
2   if( arity(goal) != 2 ) return(null);
3   PrologObject targetAgent,query;
4   targetAgent = goal.car.binding();
5   query = goal.cdr.binding();
6   if( targetAgent.typeAtom() && (query.typeVar() != true) ) {
7     String queryString = query.toString(); // 文字列に変換
8     if( query(targetAgent.name,queryString) ) {
9        return(success);
10    }
11  }
12  return(null);
13 }
```

ここで，各行左側の数値は行番号であり実際のプログラムでは入力しない．述語の実装では，まず渡された引数の数を確認し（2行目），引数の内容としてエージェント名（4行目）と問い合わせ内容（5行目）を得ている．次に，引数の型チェックを行い（6行目），問い合わせ内容を一旦文字列に変換する．問い合わせ内容を一旦文字列に変換することにより，破損あるいは改ざんされた不正なオブジェクトが問い合わせ内容としてエンジンへ渡されるのを防いでいる．実際の問い合わせはqueryメソッド（2引数）により行う（8行目）．引数2のqueryメソッドは，第1引数で指定された名前のエンジンを参照し，そのエンジンが待機中でかつ'system_lock/2'等によりロックされていなければ，問い合わせを行う．queryメソッド内部では，他の問い合わせとの排他処理が行われる．問い合わせが問い合わせ先のエンジンで実行開始されると，queryメソッドは即座に復帰する．問い合わせ先のエージェントが存在しない場合，およびロックされている場合には，問い

[9] 表を見やすくするために，一般的な組み込み述語等（assertやtraceなど）については省略している

表 4.2: 代表的なコア API 述語

	コア API	機能
1	system_request/2	他のエンジンで問い合わせを起動
2	system_remoteAssert/2	他のエンジン上の節データベースに節定義を挿入
3	system_myname/1	エンジンの名前を参照
4	system_thereExists/1	エンジンの存在の確認
5	system_suspend/0,1,2	エンジンを一時停止
6	system_resume/1	エンジンの動作を再開
7	system_lock/1,2	エンジンの他からのアクセスをロック
8	system_unlock/0	エンジンへのロックを解除
9	system_new/3	新しいエンジンの生成
10	system_delete/0	エンジンの削除
11	system_write/1	エンジンに接続されたコンソールに文字列を送信

合わせに失敗する．述語 'system_remoteAssert/2' は，他のエージェントが持つ節データベースに対して，節定義を挿入する．第1引数にエンジン名，第2引数に挿入する節の定義を指定する．述語 'system_request/2' とは異なり，述語 'system_remoteAssert/2' では問い合わせを実行中のエンジンに対しても，節定義の挿入を行うことが可能である．'system_remoteAssert/2' の実装の概要は，'system_request/2' の実装とほぼ同様であり，query メソッドの代わりに assert に相当するメソッドが用いられる点が異なる．述語 'system_myName/1' は，エンジンが自身の名前を参照することを可能とする．第1引数に，エンジン名が束縛されて返される．述語 'system_thereExists/1' は，指定したエンジン名を持つエンジンが，現在の環境に存在するかどうかを調べる．第1引数にエンジン名を指定し，そのエンジン名を持つエンジンが存在すれば真となる．エージェントプログラム開発者が実際に利用する 'query/2' 等の問い合わせ述語の機能，環境間をまたいだエージェント間問い合わせ，および問い合わせのセキュリティ制御等は，すべてシステムプログラム層でコア API を用いて定義される．

表4.2中の5から8のコア API 述語 （'system_suspend/0,1,2' , 'system_resume/0' , 'system_lock/1,2' , 'system_unlock'） は，主に割り込み処理，割り込み問い合わせ，およびモビリティの実現に利用される．述語 'system_suspend/0,1,2' は，指定したエンジンの動作を一時停止させる．引数なしの場合は自身を一時停止し，引数1つの場合は第1引数で指定されたエンジンを一時停止し，引数2つの場合は，エンジンの一時停止と同時に，第2引数で指定されたエンジン名のエンジン以外からのアクセスを排除するようにロックをかける．エンジンに対するロックは，エンジン内部のアクセス制限属性にアクセス可能なエンジン名を代入することにより行う．'system_resume/1' は，指定したエンジンが一時停止していれば，その動作を再開させる．'system_unlock/0' は，エンジンが自身にかかったロックを解除する．'interruptAndQuery/2' 等の割り込み問い合わせ述語，および移動述語は，システムプログラム層でコア API を用いて定義される．

表4.2中の9および10のコア API 述語 （'system_new/3' , 'system_delete/0'） は，主にエージェントの生成および削除を行うメタ述語の実現に利用される．述語 'system_new/3' は，指定された名前のエンジンを新たに生成し，それをエージェントとしてシステムに登録する．述語 'system_delete/0' は，エンジン自身をシステムの登録から抹消する．'new/1' 述語や 'delete/0' 述語等のメタ述語は，システムプログラム層でコア API を用いて定義される．

表4.2中の11のコア API 述語 （'system_write/1'） は，エンジンが持つ開発用ユーザインタフェース（コンソール）に対する文字列の出力を行う．コンソールによって文字列の解釈が行われ，コンソール画面上への文字列の表示等が行われる．

4.7.3 システムプログラム層の設計

システムプログラム層が実装する代表的なシステム API 述語を表4.3に示す．

表4.3の1,2, および3は，3.3.1節で定義したエージェント間問い合わせ述語の実装である．最初に，述語 'request/2' について，その基本的な動作のシステムプログラム層での実装方法を説明する．実際の実装方法は多少複雑であるので，説明を簡単にするために，ここでは基本的な動作に関する実装のみを示す．述語 'request/2' の基本的な動作は，問い

表 4.3: 代表的なシステム API 述語

	システム API	機能
1	request/2,requestF/2	他のエージェントで問い合わせを起動
2	wait/2	他のエージェントでの問い合わせ結果を受け取る
3	query/2,queryF/2	他のエージェントの問い合わせを実行
4	interruptAndRequest/2	他のエージェントへ割り込みで問い合わせを起動
5	interruptAndQuery/2	他のエージェントへ割り込みで問い合わせを実行
6	requestC1/3,requestC2/3	他のエージェントのクローンに問い合わせを起動
7	queryC1/2,queryC2/2	他のエージェントのクローンに問い合わせを実行
8	move/1	エージェントを移動
9	new/1	エージェントを生成
10	new2/2	指定した型のエージェントを生成
11	clone1/1	エージェントの（広義の）クローンを生成
12	clone2/2	エージェントの（狭義の）クローンを生成
13	delete/0	エージェントの消去
14	write/1	コンソールに文字列を出力

合わせ送信側では，コアAPI述語'system_request/2'および'system_myname/1'を用いて次のように定義される．

```
request(AGENT,QUERY) :-
  system_myname(MYNAME),
  system_request( system_macro_requestReceived( MYNAME, QUERY ) ).
```

一方で，問い合わせ受信側では次のプログラムがシステムプログラム層で用意される．

```
system_macro_requestReceived( SENDER, QUERY ) :-
  call(QUERY),!,
  system_myname(MYNAME),
  system_remoteAssert(SENDER,system_macro_know(MYNAME,true,QUERY)).

system_macro_requestReceived( SENDER, QUERY ) :-
  system_myname(MYNAME),
  system_remoteAssert(SENDER,system_macro_know(MYNAME,fail,QUERY)).
```

すなわち，問い合わせ受信側では，受け取った問い合わせを実行し，実行の成否を問い合わせ元のエージェントの節データベースへアサート(assert)する．問い合わせの結果を受け取るための述語'wait/2'は，次のように定義される．

```
wait(AGENT,query(QUERY)) :-
  retract(system_macro_know(AGENT,TF,QUERY)),
  !,TF = true.
wait(AGENT,query(QUERY) :-
  sleep_a_short_time,  % 少しの時間スリープする
  !,wait(AGENT,query(QUERY)).
```

ここで，述語'sleep_a_shor_time/0'は，この説明のために用意した疑似述語で，少しの時間だけスリープすることを意味する．すなわち，述語'wait/2'の実装は，問い合わせ結果が到着するのを待って，問い合わせ結果を節データベースからリトラクト(retract)し，実行の成否を確認している．述語'requestF/2'の基本的な動作は，問い合わせ送信側で，コアAPI述語を用いて次のように定義される．

```
requestF(AGENT,QUERY) :-
  system_myname(MYNAME),
  system_request( system_macro_requestReceived( MYNAME, QUERY ) ).
requestF(AGENT,QUERY) :-
  sleep_a_short_time, % 少しの時間スリープする
  !,requestF(AGENT,QUERY).
```

すなわち，述語'requestF/2'では，エンジンへの問い合わせが成功するまで問い合わせを繰り返すことで実現される．述語'query/2'および'queryF/2'のシステムプログラム層での基本的な実装は，それぞれ次のようになる．

```
query(AGENT,QUERY) :-
  request(AGENT,QUERY),
  wait(AGENT,query(QUERY)).

queryF(AGENT,QUERY) :-
  requestF(AGENT,QUERY),
  wait(AGENT,query(QUERY)).
```

表4.3の4および5は，3.3.1節で定義した割り込み問い合わせ述語の実装である．述語'interruptAndRequest/2'の基本的な動作は，次のように定義される．

```
interruptAndRequest(AGENT,QUERY) :-
  system_myname(MYNAME),
  system_suspend(AGENT,MYNAME),
  system_request(system_interruptRequestReceived(MYNAME,QUERY)).
```

一方で，問い合わせ受信側では，次のプログラムがシステムプログラム層で用意される．

```
system_interruptRequestReceived(SENDER,QUERY) :-
  system_myname(MYNAME),
  system_unlock(MYNAME),
  call(QUERY),!,
  system_remoteAssert(system_macro_know(MYNAME,true,QUERY)),
  system_macro_resume. % 自分自身をレジュームする
system_interruptRequesetReceived(SENDER,QUERY) :-
  system_remoteAssert(system_macro_know(MYNAME,fail,QUERY)),
  system_macro_resume. % 自分自身をレジュームする
```

ここで，述語'system_macro_resume/0'は，ここでの説明のために用意した疑似述語で，問い合わせの終了直後に，エージェント自身に対して述語'system_resume/1'による問い合わせ実行の再開を行う．問い合わせ受信側は，ロックを解除した後に問い合わせの実行を行い，問い合わせ結果を問い合わせ元にアサートした後に，中断した問い合わせの実行を再開する．述語'interruptAndQuery/2'は次のように定義できる．

```
interruptAndQuery(AGENT,QUERY) :-
  interruptAndRequest(AGENT,QUERY),
  wait(AGENT,query(QUERY)).
```

表4.3の6および7は，3.3.1節で定義したクローンへの問い合わせ述語の実装である．述語'requestC1/3'の基本的な動作は，次のように定義される．

```
requestC1(AGENT,QUERY,CLONE) :-
  interruptAndQuery(AGENT,clone1(CLONE)),
  system_myname(MYNAME),
  system_request(CLONE,system_macro_requestAndDelete(MYNAME,QUERY)).
```

一方で，問い合わせ受信側では，次のプログラムがシステムプログラム層で用意される．

```
system_macro_requestAndDelete(SENDER,QUERY) :-
  system_macro_requestReceived(SENDER,QUERY),
  system_delete.
```

ここで，述語 system_macro_requestReceived/2 は，既に定義している．すなわち，問い合わせ受信側は，問い合わせ実行後に自身を削除する．述語 queryC1/2 は，次のように定義できる．

```
queryC1(AGENT,QUERY) :-
  requestC1(AGENT,QUERY,CLONE),
  wait(CLONE,query(QUERY)).
```

述語'requestC2/3'の定義は，'clone1/1'の代わりに'clone2/1'が用いられ，述語'queryC2/2'の定義は，'requestC1/3'の代わりに'requestC2/3'が用いられる点が異なるが，他の部分は同様に定義される．

```
requestC2(AGENT,QUERY,CLONE) :-
  interruptAndQuery(AGENT,clone2(CLONE)),
  system_myname(MYNAME),
  system_request(CLONE,system_macro_requestAndDelete(MYNAME,QUERY)).


queryC2(AGENT,QUERY) :-
  requestC2(AGENT,QUERY,CLONE),
  wait(CLONE,query(QUERY)).
```

表4.3の8は，3.3.1節で定義したエージェント移動述語の実装である．述語'move/1'は，これまでに述べた他の述語とは異なり，そのエージェントのシステムプログラム層のみでなく，メタエージェントのシステムプログラム層で用意されるプログラムを組み合わせることで実現される．エージェント自身のシステムプログラム層で，述語'move/1'の基本的な動作は次のように定義される．

```
1  move(HOST) :-
2    system_macro_hide_console, % コンソールを非表示
3    system_myname(MYNAME),
4    system_lock(MYNAME),
5    system_macro_fedex(META_AGENT),
6    requestF(META_AGENT,move(MYNAME,HOST)),
7    system_suspend,
8    wait(_,query(move(MYNAME,HOST))),
9    system_macro_show_console, % コンソールを再表示
10   !,true. % 移動成功
11 move(HOST) :-
12   system_macro_show_console, % コンソールを再表示
13   !,fail. % 移動失敗
```

ここで，各行の左側に行番号を便宜的につけている．エージェントは移動前に，コンソールを非表示にする（2行目）．述語'system_macro_hide_console/0'は，ここでの説明のために用意した疑似述語で，エージェントのコンソールウィンドウを非表示にする機能を持つ．次に，移動中における他のエージェントからの問い合わせを排除するために，述語'system_lock/0'を用いて自分自身にロックをかける（3から4行目）．次に，述語'requestF/2'を用いて，エージェント自身の移動をメタエージェントに依頼し（6行目），その直後にエージェント自身を述語'system_suspend/0'を用いて一時停止させる（7行目）．メタエージェントは，エージェントの動作停止を待って，エージェントを依頼された環境へ転送する．転送が完了すると，メタエージェントが述語'system_resume/1'を用いて，エージェントの動作を再開させる．移動が成功していれば，メタエージェントへの問い合わせは成功している（8行目）．エージェントは，移動先であらためてコンソールウィンドウを再表示する（9行目）．エージェントが移動に失敗した場合には，コンソールを再表示したあと（12行目），移動述語の実行結果を失敗とする（13行目）．メタエージェントのシステムプログラム層では，述語'move/2'が次のように定義される．

```
1 move(AGENT,HOST) :-
2    system_macro_waitfor_suspend(AGENT),  % エージェントの一時停止を待つ
3    system_serialize(AGENT,DATA),         % エージェントの実行状態をキャプチャ

4    system_macro_fedex(META_AGENT),
5    system_macro_query(META_AGENT,load(AGENT,DATA),HOST),
                                           % 移動先へエージェントの復元を依頼
6    !,system_macro_delete(AGENT).         % エージェントの残骸を処理
7 move(AGENT,HOST) :-
8    system_resume(AGENT),
9    !,fail.
```

ここで，各行の左側に行番号を便宜的に付けている．メタエージェントは，問い合わせ元のエージェントが一時停止するのを待つ（2行目）．述語'system_macro_waitfor_suspend/1'はここでの説明のために用意した疑似述語で，エージェントが一時停止するまで待つという機能を持つ．エージェントが一時停止した後，メタエージェントはエージェントの実行状態をバイト列に変換する（3行目）．ここで，述語'system_serialize/2'は，メタエージェントのみが持つ特別な述語であり，エージェントの実行状態を参照してそれをバイト列に変換する機能を持つ．最後に，エージェントの状態を変換したバイト列を，移動先に転送し，移動先でエージェントの復元を行うように，移動先のメタエージェントに問い合わせを行う（4行目と5行目）．ここで，述語'system_macro_query/3'は，この説明のために用意した疑似述語であり，第3引数に環境のアドレスを指定することにより，異なる環境間でのエージェント間問い合わせを行う機能を持つ．述語問い合わせが成功すれば，移動元に残ったエージェントの残骸を削除し（6行目），問い合わせを成功とする．ここで，述語'system_macro_delete/1'は，ここでの説明のために用意した疑似述語で，第1引数で指定されたエージェントの実体を削除する機能を持つ．エージェントの移動に失敗した場合には，そのエージェントは単に動作を再開され（8行目），問い合わせの実行は失敗と

する（9行目）．エージェントの復元を行う述語‘load/2’は，メタエージェントのシステムプログラム層で次のように定義される．

```
load(AGENT,DATA) :-
  system_deserialize(AGENT,DATA),  % エージェントを復元
  system_resume(AGENT),            % エージェントの動作を再開
  !,true.                          % 成功
```

ここで，述語‘system_deserialize/2’は，メタエージェントのみが持つ特別な述語で，バイト列で表現されたエージェントの状態を復元する機能を持つ．移動先の環境に存在するメタエージェントは，問い合わせで受け取ったエージェントを復元し，その動作を再開させ，述語の実行を成功とする．

表4.3の9,10,11,および12は，3.3.1節で定義したエージェント生成を行うためのメタ操作述語の実装である．これら4つの述語の機能はコアAPI述語‘system_new/3’に統合されている．システムプログラム層では，述語‘system_new/3’へのパラメータ受け渡しを行う述語として，これら4つの述語が定義される．

表4.3の13は，3.3.1節で定義したエージェントの削除を行う述語の実装である．述語‘delete/0’の基本的な実装では，単にコアAPI述語‘system_delete/0’の呼び出しを行う．

表4.3の14は，エージェントの持つコンソールウィンドウへの出力を行う述語の実装である．述語‘write/1’の基本的な実装は，単にコアAPI述語‘system_write/1’の呼び出しを行う．

4.8 実装上の議論

4.8.1 JAVAのシリアライズ機構の問題

JAVAのシリアライズ機能には，JAVAリフレクション機能を利用して，条件を満たしたあらゆる種類のJAVAオブジェクトをシリアライズ可能とするという特長がある．一方で，JAVAリフレクション機能の性能上の問題，およびシリアライズ時のJAVAリフレクション機能の再帰的な呼び出しアルゴリズムの問題が原因となって，低速なシリアライズ速度，およびシリアライズ処理中でのスタックオーバフローの発生という問題が生じる．

*MiLogEngine*の設計では，プログラムの実行状態をJAVAシリアライズ機能を用いてビット列に変換しているため，これらのJAVAシリアライズ機能の問題が生じた．この問題に対して，*MiLogEngine*の設計では，最初に，シリアライズすべきデータそのものをできるだけ小さくすることで解決を試みた．先に述べた実行状態の最適化により実行状態そのものを小さくするということも，この問題の解決方法の1つであるといえる．特に，Last Call Optimizationは実行状態を小さくするのに非常に大きな効果があった．この方法以外に，移動元と移動先に共通して存在するデータの省略，および移動後に計算で復元できる情報の省略を行った．移動元と移動先に共通して存在するデータの1つとして，システム述語の定義がある．システム述語の定義部分をシリアライズ対象から省略し，移動後に移動先にある情報から復元するようにした．移動後に計算で復元できる情報として，組み込み述

語呼び出しを高速化するためのハッシュテーブルがある．組み込み述語呼び出し用ハッシュテーブルをシリアライズ対象から省略し，移動後に計算により復元するようにした．

これらの対策を行ったが，依然としてシリアライズの問題は完全には解決されなかった．そこで，次の 1), 2) および 3) の 3 つの対策を行った．

1) シリアライズ対象オブジェクトの分断: シリアライズ対象となる 1 つづきのオブジェクトを細かなオブジェクトの断片に分解し，対象とするオブジェクトの構造に特化したシリアライズ処理を JAVA プログラムとして記述した．本手法は，構造が比較的単純な制御スタックに対して適用した．具体的には，制御スタックにおける各 Box への参照を JAVA の `transient` 宣言を用いてシリアライズ対象から除外し，すべての Box を配列に格納した後に，参照先の Box ID を記録するようにした．同様の手法をトレイルスタックに対しても適用した．本手法の適用により，制御スタックおよびトレイルスタック関して，シリアライズ処理時のスタックオーバフローが発生しなくなった．

2) 文字列変換処理の利用: シリアライズ対象となるオブジェクトには，別途文字列との相互変換機能が実装されたオブジェクトがある．具体的には，プリミティブの表現に利用される `PrologObject` クラスは，パーザを用いて文字列からオブジェクト列に変換することが可能であり，その逆変換は `write` 述語の実装のために用意した `toString` メソッドにより行うことが可能である．文字列変換処理の利用では，単一のオブジェクトが複数のオブジェクトから参照される際に，オブジェクトの不正な複製が生成されてしまうという問題がある．節データベース上に格納された節定義では，個々の節定義は独立しており，内部で複数オブジェクトからの参照を持つオブジェクトが構造上存在しないため，シリアライズ処理の文字列変換機構による代用が可能であった．節定義の変換に文字列変換機構を用いた場合，JAVA シリアライズ機構を利用した場合と比較して，出力結果の大きさで 10%程度小さくなり，実行速度も高速となった．非常に長いアトム（およそ 5KBytes 以上）を含む節定義では，節定義の文字列変換機構の利用は逆に実行速度の低下が見られた．この問題に対処するために，およそ 5KBytes 以上のアトムを含む節定義を，文字列変換機構の処理対象から除外した．本手法により，JAVA シリアライズと比較して飛躍的に変換速度が向上した．

3) シリアライズ処理の順序制御: シリアライズ処理中でスタックオーバフローが生じる原因は，ネストの浅いところから深いところに向かってシリアライズ処理を行うことにある．JAVA のシリアライズ機構には，一連のシリアライズ処理中では，一度シリアライズされたオブジェクトに対する二度目以降のシリアライス処理が省略されるという特性がある．この特性を利用し，オブジェクトが単方向リンクで構成される場合に，ネストの深いところにあるオブジェクトを一連のシリアライズ処理中の早い段階でシリアライズしてしまうことにより，深くネストしたシリアライズ処理の発生を抑制することが可能である．具体的には，ネストの深いところにあるオブジェクトを深いところから順に，最後にルート (根) となるオブジェクトが配列の末尾に来るように配列に格納して，その配列全体に対してシリアライズ処理を実行する．本手法は，オブジェクトが主に単方向リンクで構成される節データベースの変換処理に適用した．本手法の適用により，節データベース中に多数の節定義が格納された状態で，シリアライズ処理によるスタックオーバフローが発生しなくなった．

以上の工夫により，シリアライズ処理による問題の発生が抑制された．残された課題に

は，特定のJAVA実行環境におけるシリアライズ処理の異常な低速動作への対処，およびカットオペレータを持たない末尾再帰処理への最適化の適用の検討がある．

4.8.2 JAVA特有の機能の利用

多くのProlog実装手法は，ガベジコレクタの実装を内部に持つことを前提として設計されている．一方で，JAVA実行環境が搭載するガベジコレクタは非常に高機能であり，JAVAがもつガベジコレクタは最大限利用したい．*MiLogEngine*の設計では，独自のガベジコレクタを一切持たず，JAVAのガベジコレクタを最大限に利用することで，実装の単純化と実行速度の高速化を狙っている．本設計では，既存のProlog処理系がもつガベジコレクタの機能を利用した最適化機能が利用できなくなるため，新たに別のアルゴリズムによって最適化機能を実装している．アトムや関数の名前などの名前テーブルの管理では，JAVAの機能である`intern`メソッドを利用している．`intern`メソッドは，JAVA内部にある名前テーブルを用いて，同一の文字列が必ず同一のオブジェクトとなるように値を返す．本機構を利用することにより，名前テーブルの管理をJAVAに行わせることが可能となった．

JAVAの実行速度を高速化するための技術として，JIT(Just-In-Time)コンパイラ，および動的にコンパイル箇所を判断するHotSpot Performance Engineが開発されている．これらの高速化手法は非常に効果的であり，純粋なインタプリタと比較しておよそ10倍から30倍程度の高速化が実現される．これらの高速化手法には得手不得手があり，効果的に高速化される種類のプログラムと，あまり高速化が有効に働かない種類のプログラムがある．*MiLogEngine*の設計では，これらの高速化手法の恩恵を受けやすくなるように，一見冗長に見える設計をあえて行っている部分がある．具体的には候補ホーン節の複写部分で，実際には複写する必要があるかどうか決定されていない部分まで投機的に複写処理を実行している．投機的な複写処理は本来必要な複写処理と同一のループ中で処理されるため，JAVA固有の高速化手法が効果的に作用する．逆に，本設計手法をJAVA以外の言語で実装する場合には，これらの設計上の考慮点に注意し，実装言語に適した実装を行う必要がある．

4.9 評価

本節では，まず，*MiLog*におけるProlog処理系としての性能がどの程度実用的であるかを評価する．次に，*MiLog*におけるモビリティの性能を評価する．表4.4に，*MiLog*およびJinni[123]におけるベンチマーク結果を示す．*MiLog*とJinniはともにJAVA言語上で動作し，プログラムをインタプリタ方式によって実行する点で類似しており，*MiLog*の実装モデルの実用性を評価する上で，Jinniとの比較は妥当だと考えられる．`nreverse`は，数値リストの反転により実行速度を計測するベンチマークである．`8queens`は，8クイーン問題を解くプログラムである．表4.4では，最初の解（First Answer）を求めるまでにかかった時間と，すべての解（All Answers）を求めるまでにかかった時間の2つを計測している．実験環境には，AMD社製Athron 700MHzと192MBメモリを搭載したWindows 2000 PC上において，IBM社製JAVA実行環境JRE 1.1.8を用いた．Jinniは評価版のJinni98を用

表 4.4: ベンチマーク結果

Benchmark	Jinni	MiLog
nreverse	19.6KLIPS	**27.5 KLIPS**
8queens(first answer)	8547[msec]	**548.8 [msec]**
8queens(all answers)	151885[msec]	**9065 [msec]**

いた．nreverseでは，Jinniと比較して*MiLog*が40%程度高速である．8queensではさらに大きな差がついているが，これはJinniがカットオペレータをサポートしないために無駄なバックトラックが生じているためだと考えられる．表4.4の結果は，*MiLog*が実装モデルとして実用上十分な性能を持つことを示している．

*MiLog*における実行状態の保存データ縮小化が有効に機能していることを評価するために，K-Prolog[103]との比較を行った．K-Prologは，Prologプログラムの実行状態をファイルに書き出すことができる述語save/1を実装している．本比較では，nreverseプログラム実行時において再帰処理が最も深くなる部分での実行状態を保存し，保存されたデータの大きさを比較した[10]．図4.12では，nreverseプログラムにより反転するリストの大きさを10から100まで10単位で変化させながら，*MiLog*およびK-Prologにおける実行状態の保存データサイズを示している．K-Prologでは，測定値が100KBytes以上となったが，*MiLog*では4KBytesから12KBytes程度の大きさとなった．*MiLog*における実行状態の保存サイズはK-Prologと比較して十分に小さく，本節で提案した手法が有効であることを示している．

*MiLog*のモバイルエージェントシステムとしての性能が実用上十分であるかどうかを評価するために，JAVA上のモバイルエージェントシステムAglets[84]およびBee-gent[77]との比較を行った．AgletsおよびBee-gentは，JAVA言語上で動作可能な弱モビリティを実現した代表的なモバイルエージェントシステムである．AgletsおよびBee-gentとの比較により，JAVA言語上に実装された弱モビリティに基づくシステムと比較して，本実装がモバイルエージェントシステムとしてどの程度実用的かを判断できると考えられる．本比較実験では，2つの実行環境間を100往復する単純なエージェントを作成し，プログラムのソースコードサイズ，移動に必要な転送データ量，および時間を計測した．なお，移動時間の計測では，同一PC上に2つのモバイルエージェント実行環境を起動してその間の移動時間を計測した．表4.5に，計測結果を示す．*MiLog*におけるプログラムのソースコードサイズは，AgletsおよびBee-gentと比較して小さくなっている．これは，*MiLog*における強マイグレーションの実現によってプログラミングが簡潔になったことを示している．エージェントの移動速度については，強マイグレーションによる処理のオーバヘッドから*MiLog*よりもAgletsが，より高速に動作している．しかしながら，*MiLog*の移動速度はBee-gentと比較して高速であり，*MiLog*では十分なエージェントの移動速度が実現できたといえる．エージェントの移動時における転送データサイズは，*MiLog*が最も小さい値を示した．*MiLog*が実行状態をマイグレーションしているにもかかわらず，*MiLog*における転送データサイズが，AgletsおよびBee-gentと比較して小さくなっているのは，*MiLog*におけるプログラム（コード）のサイズが他に比べて小さいという理由もあるが，*MiLog*における実行状態の転送時最適化が有効に機能していることを示している．以上により，*MiLog*を用いた任意時刻モビリティの実装が，十分な性能を持っていることが示された．

[10] K-PrologはCソースコードへのコンパイラを持つが，この実験ではWAMコードインタプリタ版を使用している．

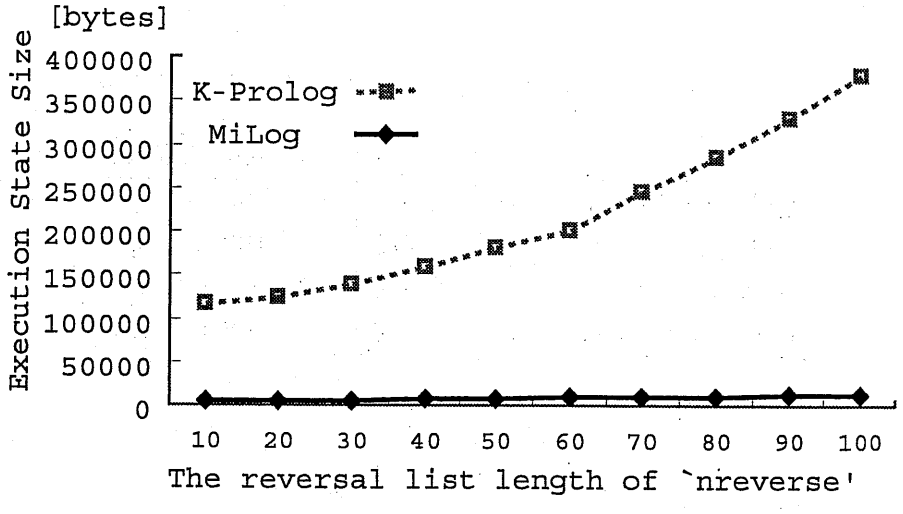

図 4.12: 実行状態の保存サイズ

表 4.5: 他のモバイルエージェントシステムとの比較

	Aglets	Bee-gent	MiLog
code size(lines)	82	91	**14**
code size(bytes)	1,775	2,193	**293**
migration size(bytes)	2,058	11,063	**2,005**
migration speed(msec)	69	313	**166**

第5章

応用

フレームワークは，それ自身が直接的に何かを生み出すわけではないため，評価を行うことが難しい．本論文で述べたフレームワークの評価を行う方法の1つとして，そのフレームワークの応用事例からの評価を試みる．本章では，*MiLog* フレームワークの応用について述べる．*MiLog* フレームワークの応用として，2つのフレームワークと1つのシステムを構築する．1つは，ユーザインタフェース構築フレームワーク *iML* である．*iML* フレームワークでは，モバイルエージェントのためのグラフィカルユーザインタフェース構築フレームワークを，*MiLog* のハイブリッドプログラミングを用いて構築している．もう1つは，WWW とモバイルエージェントとの統合フレームワーク *MiPage* である．*MiPage* では，HTML に基づいて作成された構造化文書に直接アプリケーション制御ロジックを埋め込むことにより，モバイルエージェントを構築する．最後の1つは，オンラインオークション入札支援システム *BiddingBot* システムである．*BiddingBot* システムでは，複数の自律的なエージェントによってシステムが構築されており，オンラインオークションへの入札を効果的に支援する．

5.1 *iML*: GUI アプリケーションフレームワーク

5.1.1 はじめに

モバイルエージェント技術は，分散システムの容易な設計，実装，および保守を実現するための技術として近年注目されている．モビリティによって，エージェントは，ネットワーク負荷を低減，ネットワーク遅延の克服，およびネットワークの切断への対処を行うことが可能となる．[84]. モバイルエージェントシステムによって，ユーザはモバイルエージェントアプリケーションを容易に実装できるようになる [84, 78, 109]. 多くのモバイルエージェントシステムにおいて，そのモビリティは既存のプログラミング言語（たとえば，JAVA 言語）かあるいはモビリティに特化した言語（たとえば，*Telescript*[132]）上に実現される．それらのモバイルエージェントシステムは，そのシステムを用いてモバイルエージェントアプリケーションを構築する際に，特定のプログラミング言語を使用することを強制する．モバイルエージェントシステムのリスト [87] によれば，少なくとも 70 以上のモバイルエージェントシステムが既に開発されている．それにもかかわらず，論理型言語を

サポートするものはごくわずかしかなく，論理型言語上でのGUIプログラミングをサポートするものはこのリストには存在しない．論理型言語の持つ一階述語論理表現は次の2点において，モバイルエージェントの構築に適している:

- **強力な記述力** プログラマは一階述語論理の持つ強力な記述力を用いて，モバイルエージェントの複雑な振る舞いを容易に記述することができる．
- **記述の簡潔性** 一階述語論理によって記述されたプログラムは単純で簡潔であるため，プログラマのコード記述の負担が低減される以外に，エージェントの移動オーバヘッドも低減される．

モバイルエージェント技術は，ネットワークベースの分散知的システムの実装にも適している．我々は，モバイルエージェント技術と論理型言語の一階述語論理の記述力の相乗効果を期待して，論理型言語に基づくモバイルエージェントフレームワーク *MiLog* を提案している．*MiLog* のこれまでの実装では，GUIはJAVA言語によって記述され，その制御はJAVA-*MiLog* 間通信インタフェースを用いて行われている．従来のアプローチでは，利便性，および性能の点で課題がある．GUIプログラミングでは，ユーザの操作により発行される様々なイベントを扱わなければならない．プログラマは，GUIから発行されたイベントを受け取り，それを適切な問い合わせに変換する処理を自ら記述する必要がある．*MiLog* フレームワークは，*MiLog* プログラミングの部分には強モビリティを提供するものの，イベント処理のコーディングに対しては強モビリティを提供していない．もう1つの課題は，Javaシリアライズ機能の性能に関して，GUIのシリアライズ処理の速度面でのオーバヘッドが大きく，特定の部品（Swingコンポーネントなど）をシリアライズするための機能が提供されないことである．本節で，我々は，GUIを持つモバイルエージェントを構築するための，論理型言語に基づくプログラミングフレームワーク *iML* を提案する．*iML* フレームワークでは，GUIは一階述語論理を用いた簡潔な形式で表現される．これらのGUI表現は，論理プログラムによって画一的に扱うことができる．*iML* フレームワークでは，我々は，モバイルエージェントの実行環境，論理型言語に基づくモバイルエージェント記述言語，およびGUIのデザインを視覚的に行うための機構の3つを実現している *iML* フレームワークに対して，マイグレーション性能，およびフレームワークの扱いやすさの2つの視点から評価を行う．

本節の以降の構成は次のようになっている．第5.1.2節で，本フレームワークのモデルと構成を示す．第5.1.3節で，本フレームワークの実装の詳細を述べる．第5.1.4節で，本フレームワークの備えるGUIの視覚的デザインツールの概観を示す．第5.1.5節で，本フレームワークを性能と扱いやすさの2つの指標から評価する．第5.1.6節で，本節における成果をまとめる．

5.1.2 *iML* のモデルと構成

iML では，*MiLog* モバイルエージェントフレームワーク [47] のプログラミングモデルに基づいている．*iML* フレームワークにおける *iML* エージェントとは，*MiLog* フレームワーク上のエージェントを拡張したものである．本エージェントには，GUIコンポーネントお

よびイベントを制御するための拡張がなされている．本エージェントは，GUIコンポーネントの状態の保存を可能とするための機構を備える．図 5.1 に，*iML* エージェントにおける GUI 保存のプログラムを示す．*iML* エージェント上では，述語 move/1 は，図に示されるコードによってオーバーライトされる．述語 moveInterpreter/1 は，*MiLog* フレームワークにおける述語 move/1 と同等の機能を持つ．まず，現在の GUI の状態を保存するために，述語 assertAllGUI/0 が実行される．次に，エージェントの持つインタプリタの状態を移送するために，述語 moveInterpreter/1 が実行される．もし移送が成功した場合，GUI の状態を復元するために，述語 restoreAllGUI/0 が移動先の実行環境で実行される．この段階で，イベント制御機構も移動先の実行環境で同時に復元される．もし移送に失敗した場合，move/1 の次の定義にバックトラックする．ここでは，単純に restoreAllGUI/0 が，移動前の実行環境上で実行される．*MiLog* の提供する強モビリティを用いることにより，*iML* フレームワーク上における GUI の復元機構は十分に単純な構造で実現可能となっている．

5.1.3 実装

概観

iML フレームワークは，JAVA 言語および *Milog* フレームワークを用いて実装されている．*MiLog* フレームワークは JAVA 言語を用いて実装されている．*MiLog* フレームワーク自身の実行形式は十分に小さく（およそ 220Kbytes），多くの利用者にとって容易にダウンロードおよびインストールが可能である．多くの JAVA に基づくモバイルエージェントシステムでは，JAVA シリアライズ機能が利用されている．JAVA の実装上の制約から，JAVA シリアライズ機能では JAVA 上のスレッドをキャプチャする機能は提供されない．*MiLog* では，JAVA 上に新たな論理プログラミングエンジンを構築することで，スレッドのキャプチャを実現している．本エンジンでは，*MiLog* プログラムの実行状態を格納するスレッドを，JAVA オブジェクトのツリー構造として表現することにより，JAVA シリアライズ機能を利用したキャプチャを実現している．

GUI の保存と復元

iML フレームワークでは，GUI は *iML* コンポーネントにより構成される．それぞれのコンポーネントの実体は，ユニークな *ID* を識別子として持つ．それぞれのコンポーネントの定義には，それに対応する *JAVA* クラス（例えば，Java.awt.Button）を型として持っている．GUI の状態は，属性値の集合と包含関係によって表現される．属性値の集合は，そのコンポーネントが JAVA オブジェクトとして持つ属性値の集合である．包含関係は，コンポーネント間の関係を示す．属性値の集合とコンポーネントの包含関係は，*MiLog* エージェント上における節データベース上に表現され，保存される．図 5.2 に，フレーム（ウィンドウ）の内部にボタンを 1 つ持つような簡単な GUI の，節データベース上での表現の例を示す．述語 iml_cpp/3 は，コンポーネントの *ID*，その *JAVA* クラス，そのコンポーネントが持つ属性値の集合を表現している．述語 iml_ctr/2 は，コンポーネントごとの包含関

```
move(X) :-
  assertAllGUI,
  moveInterpreter(X),
  restoreAllGUI,!.

move(X) :-
  restoreAllGUI.
```

図 5.1: GUI保存のためのプログラムコード

係を表現している．

イベント制御

GUIの各コンポーネントを管理するために，*iML*エージェントは*GCM*(GUI Component Manager) を持つ．図 5.3 に，*GCM*とエージェントの構成を示す．*GCM*は，エージェントが管理すべきGUIコンポーネントを束ねており，それらのコンポーネントとエージェントの間の通信手段を提供している．GUIが実体化されたときに，*GCM*はそのGUIのコンポーネントに対する監視役（イベントリスナ）となる．コンポーネントから発行されるすべてのイベントは*GCM*によって監視されている．*GCM*によって観測されたイベントは，それに対応する適切な問い合わせに変換され，そのエージェントが持つインタプリタ（*MiLogEngine*と呼ぶ）上で実行される．例えば，あるボタンが押されたことを示すイベント `actionPerformed` がJAVAレベルで発行されると，そのイベントは*GCM*によって捕捉され，問い合わせ `?- actionPerformed(...).` に変換されてインタプリタに渡される．本機構を機能させるために必要なJAVAレベルでの準備操作（例えば，オブジェクトの初期化，イベントリスナの登録など）はすべて自動的に*GCM*によって行われ，*iML*のユーザはそれらの準備操作を意識することなく*iML*を利用可能である．本機構により，*iML*の利用者の負担を軽減することが可能であり，コードの簡潔性も向上する．

5.1.4 視覚的デザインツールの概観

*iML*の視覚的デザインツールは，3つのモード *sketch mode*, *edit mode*, および *exec mode* を持つ．

*sketch mode*では，システムによってGUIの概観を直感的に設計するためのワークスペースが提供され，設計者はGUIコンポーネント（例えば，ボタンやパネルなど）を自由に配置することができる．図 5.4 に，*sketch mode*におけるユーザインタフェースを示す．図中の左のウィンドウは，*inspector window*と呼び，右のウィンドウを*layout design window*と呼ぶ．設計者は，簡単なマウス操作（たとえば，コンポーネントの端をつかむことでコンポーネントのサイズを調整するなど）により，様々なGUI配置を試すことができる．コンポーネントに対するより正確な値の設定には，*inspector window*を用いる．*layout design window*は*edit mode*でも提供されるが，本モードとの違いは，*sketch mode*のほうが，より軽量で簡便な機能を提供する点である．*sketch mode*では，利用者はGUIのおおまかな設計を迅速に設計することが可能である．

*edit mode*では，コンポーネントの特定のプロパティの値など，GUIの詳細な部分設計を支援するための機能を提供する．図 5.5 に，*edit mode*におけるユーザインタフェースを示す．図中の右にある*layout design window*の外観は*sketch mode*のものと似ているが，本モードではGUIの詳細な属性（たとえば，フォントサイズやラベル，色など）を反映した表示がされている．設計者は，本モード上で（ボタンが押された場合などの）イベント制御コードをプログラミングすることが可能である．

*exec mode*では，設計者が作成したGUIの振る舞いの動作試験を行うための環境を提供する．図 5.6 に，*exec mode*におけるユーザインタフェースを示す．図中，左のウィンドウ

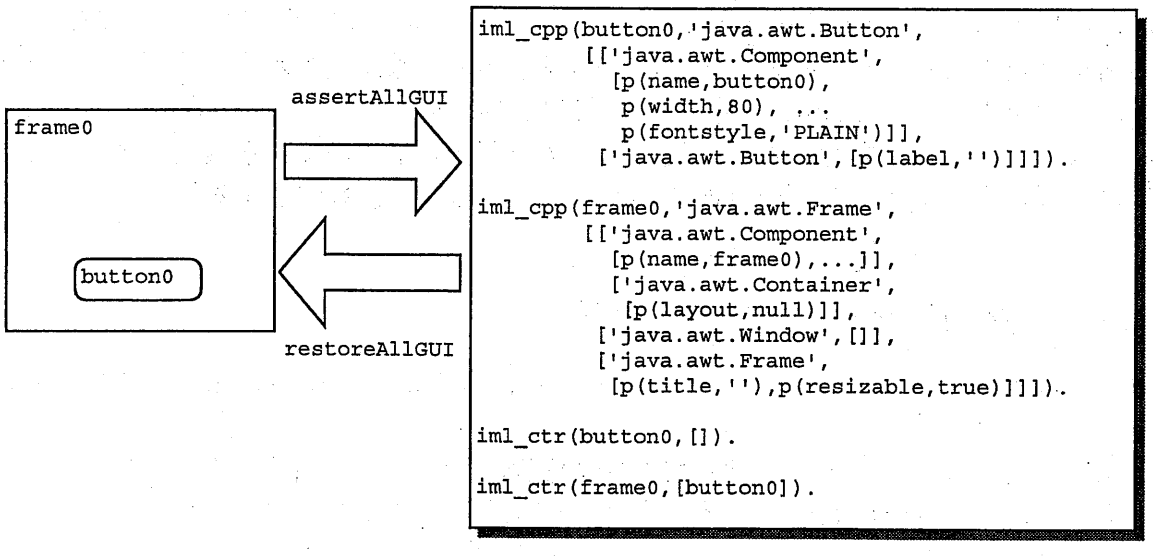

図 5.2: 保存される GUI の節データ表現の例

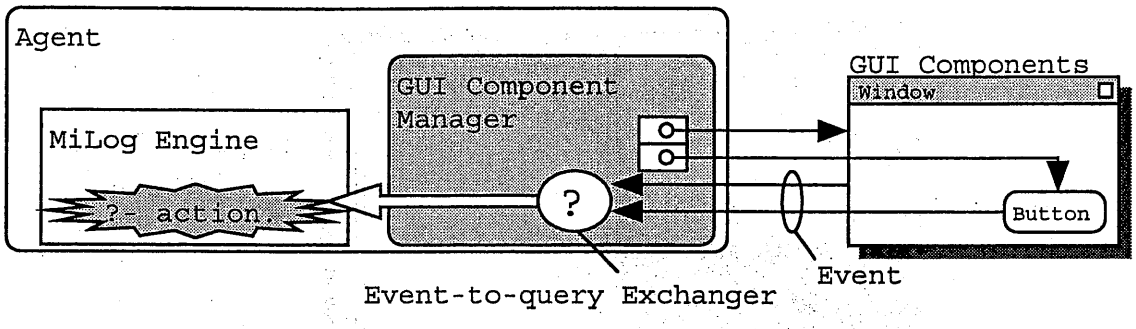
Agent
MiLog Engine
?- action.
GUI Component
Manager
?
GUI Components
Window
Button
Event
Event-to-query Exchanger

図 5.3: *GCM* の構成

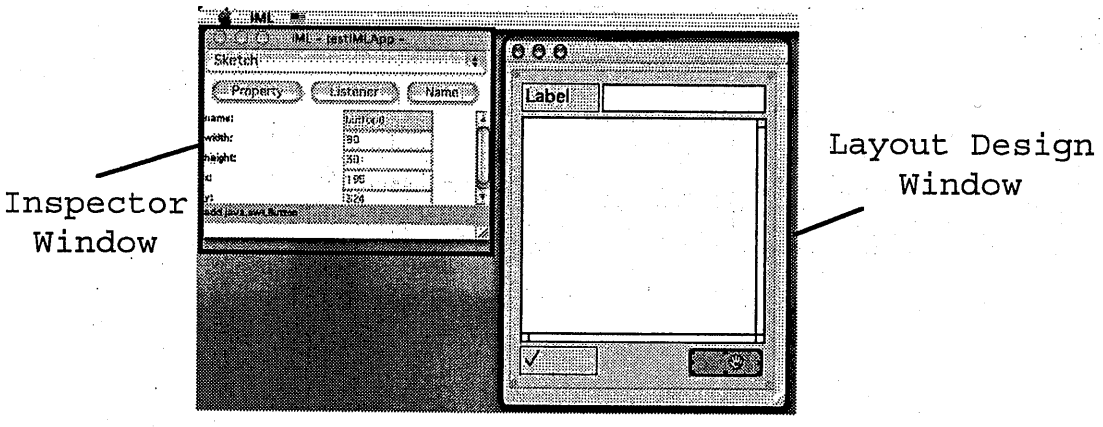

図 5.4: *sketch mode* におけるユーザインタフェース

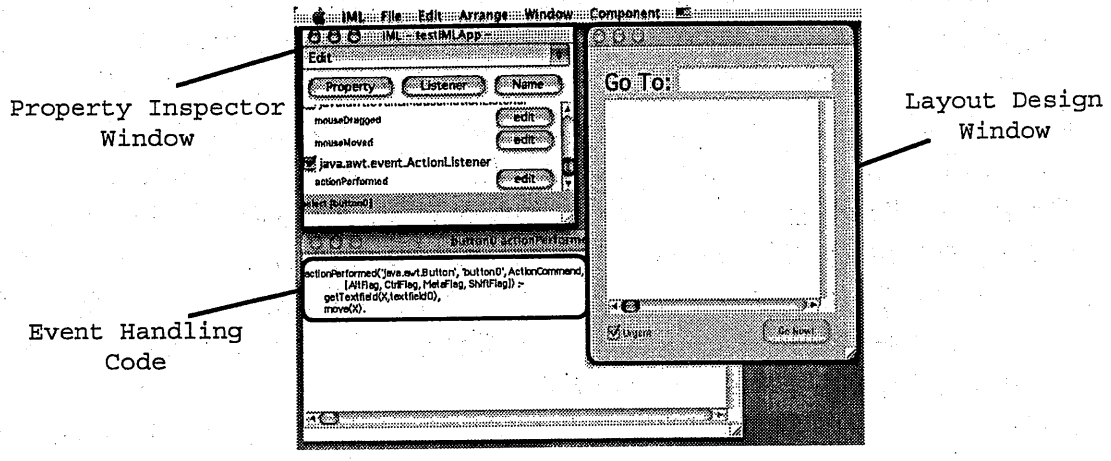

図 5.5: *edit mode* におけるユーザインタフェース

を *debugging console* と呼び，GUI イベント制御プログラム等を直接実行したり，その実行トレースを閲覧することを可能としている．

最終的に，本デザイン環境で設計されたアプリケーションは *MiLog* エージェントとしてコンパイルされ，*MiLog* 実行環境上で動作させることが可能となる．

5.1.5 評価

我々は，*iML* フレームワークを性能と使い勝手の２つの観点から評価する．*iML* のモビリティの性能を評価するために，我々は JAVA シリアライズ機能との比較を行った．JAVA シリアライズ機能は，多くの JAVA に基づくモバイルエージェントシステム上で利用されている．本比較実験では，我々はカレンダーアプリケーションを JAVA および *iML* を用いて作成し，それらの比較を行った．本実験では，Microsoft Windows 上に JAVA2 SE 1.3.0 を HotSpot client VM モードで起動した 750MHz の CPU を搭載するノートパソコン上で行った．実験結果を表 5.1 に示す．表に示した数値は，１００回マイグレーションさせた際の，１マイグレーションあたりの平均値である．本実験結果から，*iML* の手法は，転送サイズにおいて 6.7 倍小さく，移送速度において 6.0 倍高速であった．JAVA シリアライズ機能は，最大の性能を発揮することよりも汎用性を重視した設計となっている．JAVA シリアライズ機能を用いた際の性能の低さは，JAVA の設計思想によるものだと思われる．JAVA シリアライズ機能を用いた多くのモバイルエージェントシステムが，利用されている事例が多数あり，本提案手法は JAVA シリアライズに基づく実装と比較して高い性能を発揮していることから，本提案手法は，プログラミング教育やアプリケーションのプロトタイピングなどの用途では十分な性能を持つといえる．

著者の指導教官が担当する知能プログラミング演習において，本 *iML* フレームワークを実際に利用し，評価を行った.. 本演習の前に，学生は JAVA 等によるプログラミングの知識と，Prolog プログラミングに関する基礎的な知識を習得している．本演習には，60 名の学生が参加し，45 名の学生が本フレームワークを用いてモバイルエージェントを作成することができた．また，16 名の学生は非常に興味深いアプリケーションを本フレームワークを用いて作成した．

5.1.6 まとめ

本節では，我々は，GUI を持つモバイルエージェントを構築するための，論理型言語に基づくプログラミングフレームワークを提案した．本フレームワークの性能と使いやすさを，プログラミング演習に適用することで確認した．現状では，実装は JAVA2 SE のみであり，PDA や携帯電話上で動作する JAVA2 ME 上での動作は今後の課題となっている．これらの限られた計算資源を持った実行環境上へ本フレームワークを拡張することを検討している．

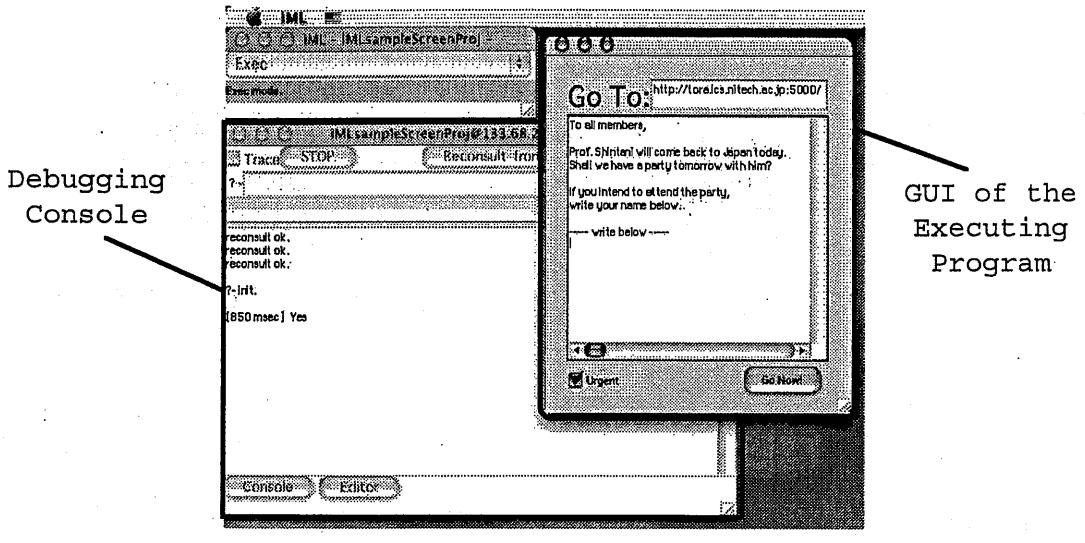

図 5.6: *exec mode* におけるユーザインタフェース

表 5.1: *iML* におけるマイグレーションの性能

	Java serialization	*iML*
transmitting data size [bytes]	59419	8876
elapsed time [msec]	4952	829

5.2 *MiPage*:Web ページ構築フレームワーク

5.2.1 はじめに

本節では，Web ページに埋め込み可能で，かつ移動可能なエージェントを実現するためのフレームワーク *MiPage* を提案する．本フレームワークの特長は，1) Web ページを表現する HTML 文書へプログラムを直接埋め込むことが可能なため，表示結果の設計が容易となる点，2) プログラムの記述にモバイルエージェント記述言語 *MiLog*[47] を利用可能とすることで，強モビリティを利用したモバイルエージェントプログラムの記述が容易に行える点，および 3) HTTP のリダイレクト機構を利用することで，エージェントの移動を意識させることなくプログラムが実行可能である点の 3 つである．

Web ページにモバイルエージェントを埋め込むことを可能とすることにより，新しいスタイルの WWW アプリケーションが生まれる可能性がある．特に，複数サーバを利用するアプリケーションの記述が飛躍的に容易になる点が注目される．従来のサーバクライアント型のアプリケーション構築モデルでは，各ホストごとにサーバプログラムを用意し，それらのサーバプログラム間での通信プロトコルを規定して，サーバプログラム間で同期をとりながら動作させるように，プログラムを設計する必要があった．本フレームワークでは，モバイルエージェントとしてアプリケーションを記述することにより，各ホストにはモバイルエージェントの実行環境のみを動作させ，アプリケーション固有のサーバプログラムの記述，通信プロトコルの規定，およびサーバプログラム間での同期処理を省略することを可能とする．

想定されるアプリケーションには，分散型グループ意思決定支援システム，情報収集システム，および情報モニタリングシステムがある．

分散型グループ意思決定支援システムでは，各ユーザごとにモバイルエージェントが割り当てられる．エージェントはユーザと WWW ブラウザにより対話する．ユーザと対話する場面では，ユーザの手元にエージェントを移動させ，ユーザとの対話の応答速度を向上させる．一方で，他のエージェントとの密な通信が必要な場合（交渉を行う場合など）は，エージェントを通信相手の近くに移動させることで，利用ネットワーク帯域の低減を行う．具体的な例としては，分散会議スケジューリングシステム [115] などが考えられる．

情報収集システムでは，エージェントは，情報収集時にネットワークに常時接続された環境に移動して，ユーザと情報源との間の通信帯域に制限されることなく情報収集を行う．情報収集を行うエージェントは *MiPage* を用いて作成され，ユーザはエージェントにブラウザを介してアクセスすることにより情報収集の途中経過および結果を閲覧することが可能となる．同種のシステムに Mobeet[93] がある．

情報モニタリングシステムでは，エージェントはネットワーク上の特定の情報の変化を監視しており，情報が特定のパターンで変化した場合に，ユーザに電子メールを送付するなどの行動を実行する．エージェントが *MiPage* で構成されることにより動作中のエージェントの状態を WWW ブラウザを介して閲覧することが可能となる．エージェントがモバイルエージェントとして構成されることで，システムの負荷およびネットワーク速度等から，適切なホストをエージェント自身が選択して情報の監視を行うアプリケーションの設計が容易となる．具体的な例としては，オンラインオークションにおける商品価格の変化に対応して入札を代行するシステムとして，*BiddingBot* システム [69] がある．

本節の以降の構成は，次のようになっている．5.2.2節では，*MiPage*フレームワークの概観を示し，システムの設計を明らかにする．5.2.3節では，*MiPage*フレームワークを用いて実際にモバイルエージェントを構築する方法を述べる．5.2.4節では，関連研究および関連技術との比較を行い，*MiPage*フレームワークの新規性を明らかにする．5.2.5節で，本研究において今後解決されるべき課題について議論する．5.2.6節で，本研究の成果をまとめる．

5.2.2 *MiPage*フレームワークの構成

システムの構成

システムの構成図を図5.7に示す．本システムは，論理プログラムを含んだHTML文書を*MiLog*プログラムに変換する*MiPage*コンパイラ，*MiLog*プログラムから*MiLog*エージェントを生成しそれを実行するための*MiLog*実行環境，およびHTTPリクエストを対応するエージェントに受け渡すHTTPリクエストルータの3つから構成される．エージェント設計者は，論理プログラムを含んだHTML文書（*MiPage*プログラムと呼ぶ）を記述する．エージェントが利用者によって利用される場合，そのエージェントがまだ実行環境上に存在しない場合には，*MiPage*コンパイラによって*MiLog*プログラムが生成され，その*MiLog*プログラムが新たに生成されたエージェントにロードされる．すでにそのエージェントが存在する場合には，HTTPリクエストルータを経由してそのエージェントに利用者のブラウザから送られたHTTPリクエストが渡される．エージェントによるHTTPリクエストの処理結果はHTTPリクエストルータに戻され，HTTPリクエストルータから利用者のブラウザに処理結果がHTML文書として渡される．

*MiPage*プログラムと*MiPage*コンパイラ

*MiPage*プログラムは，HTML文書中に特定のタグを埋め込むことにより記述される．図5.8に*MiPage*プログラムの例を示す．図5.8のプログラムでは，アクセスされるごとに値を1づつカウントアップする機能を持つWebページを記述している．*MiPage*プログラムでは，`<HEAD>`タグ中の`<SCRIPT>`タグ中にプログラムを格納する（図5.8の(a)の部分）．`<SCRIPT>`タグ中のコードが*MiLog*プログラムであることを明示するために，`<SCRIPT>`タグのオプションとして`LANGUAGE="MiLog"`を指定する．例えば，図5.8では，呼び出されるごとに値を1増加させる述語`currentValue/1`を定義している．述語`currentValue/1`では，初期値を節データベースから取り出し，その値に1を加算した後，加算後の値を節データベースに再格納している．`<SCRIPT>`タグ内に記述したプログラムは，コメントタグを変形した特殊なタグ中に記述された問い合わせによって起動される．例えば，図5.8のプログラムでは，`<!--?-currentValue(X).-->`（図5.8の(b)の部分）が問い合わせの記述である．問い合わせ`?-currentValue(X).`が実行され，変数`X`への束縛値が問い合わせ記述部分と置き換わる．

図5.9に，図5.8のプログラムの*MiPage*コンパイラによるコンパイル結果を示す．コンパイル結果として出力される*MiLog*コードは，*MiPage*プログラム中に記述された論理プ

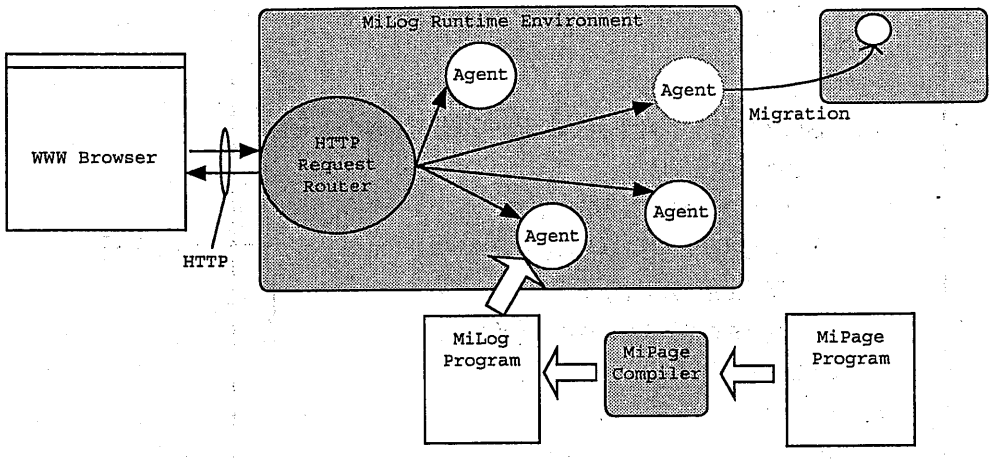
MiLog Runtime Environment
Agent
Agent
Agent
Agent
Migration
WWW Browser
HTTP Request Router
HTTP
MiLog Program
MiPage Compiler
MiPage Program

図 5.7: システムの構成

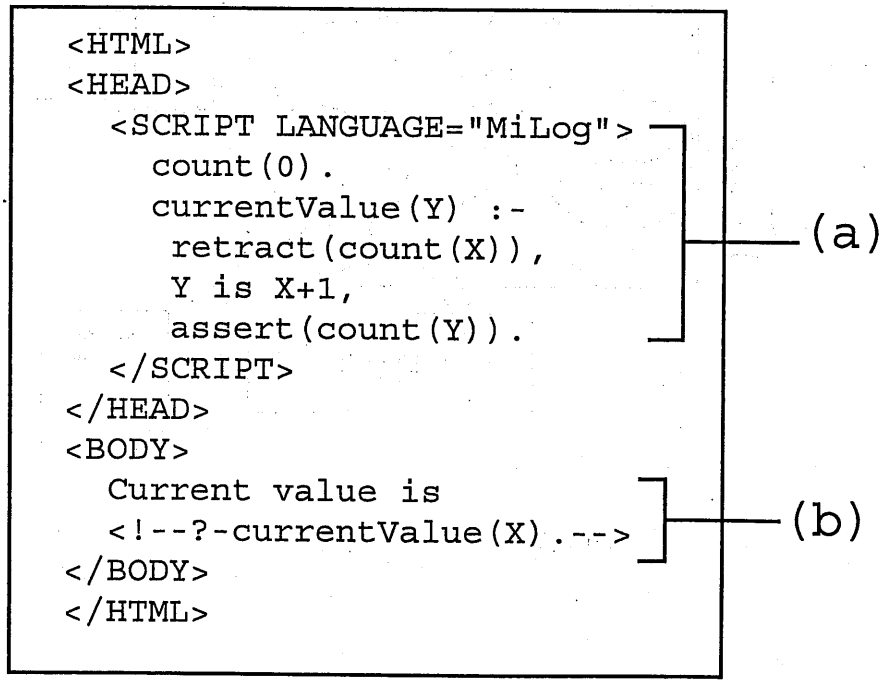

図 5.8: *MiPage* プログラムにおける HTML 文書へのカウンタ機能の埋め込み

ログラム部分（図5.9の(a)の部分），HTTPリクエストハンドラの定義（図5.9の(b)の部分），およびHTTPリクエストハンドラ中の動作（図5.9の(c)の部分）の3つの部分から構成される．*MiLog*エージェントに対するHTTPリクエストは，HTTPリクエストルータによって問い合わせ doService/2 に変換される．問い合わせ doService/2 では，第1引数に，その問い合わせの識別子が入り，第2引数にエージェントによるリクエストの処理結果がリスト構造で束縛される．第2引数への束縛結果はHTTPリクエストルータによってHTML文書に変換され，HTTPレスポンスとして利用者のWWWブラウザへ返される．

図5.10に図5.8のプログラムの実行例を示す．WWWブラウザからのアクセスがあるたびに，カウンタの表示が1づつ増加する．

HTTPリクエストルータ

HTTPリクエストルータは*MiLog*実行環境内部に含まれ，WWWブラウザ等からのHTTPリクエストを*MiLog*エージェントに受け渡す役割を果たしている．HTTPリクエストの例を図5.11に示す．HTTPリクエストでは，WWWサーバに対する要求を，URL，要求のオプションパラメータ，およびそのURLの扱い方（メソッド）として記述する．たとえば，WWWブラウザのアドレスバーからURLとして `http://127.0.0.1:17008/agent?item=ruby&action=delete` を入力した場合，IPアドレス 127.0.0.1 上でポート番号 17008 で動作しているWWWサーバに対して図5.11の要求が送信される．図5.11の要求では，`agent?item=ruby&action=delete` がURLとして指定されている．URL中の '?' 記号より後ろの部分はこの要求に対するパラメータを意味し，ここでは，項目 'item' および 'action' の値がそれぞれ 'ruby', 'delete' であることを示している．図5.11の要求では，このURLに対する処理の方法として，'GET' が指定されている．要求のオプションパラメータには，WWWブラウザの種類，およびWWWブラウザが受け取ることができるデータ型についての定義が含まれている．

HTTPリクエストルータはHTTPリクエストを受け取ると，そのリクエストをURL，要求のオプションパラメータおよびそのURLの扱い方に分離する．URLは，さらにそれをファイルパスおよびパラメータ列に分解する．HTTPリクエストルータは，ファイルパスをエージェント名とみなして，該当するエージェントがその*MiLog*実行環境上に存在するかどうかを調べる．もし，該当するエージェントが存在すれば，そのエージェントに対して，HTTPリクエストの処理を問い合わせる．この問い合わせでは，まず，HTTPリクエスト中のオプションパラメータが `option( ID, NAME, VALUE)` の形式で対象とするエージェントの持つ節データベース中に追加（ assert ）される．次に，HTTPリクエストに対する処理を求める問い合わせ doService/2 が対象とするエージェント上で起動される．問い合わせ doService/2 の実行終了後，追加（ assert ）されていたオプションパラメータが削除（ retract ）され，doService/2 の実行結果がHTTPレスポンスとして整形され，HTTPリクエスト元へ返信される．

URLで複数のオプションパラメータが指定された場合，それぞれのオプションパラメータが個別に節データベースに追加される．例えば，

```
http ://133.68.221.183:17008/agent?opt1=val1&opt2=val2&opt3=val3
```

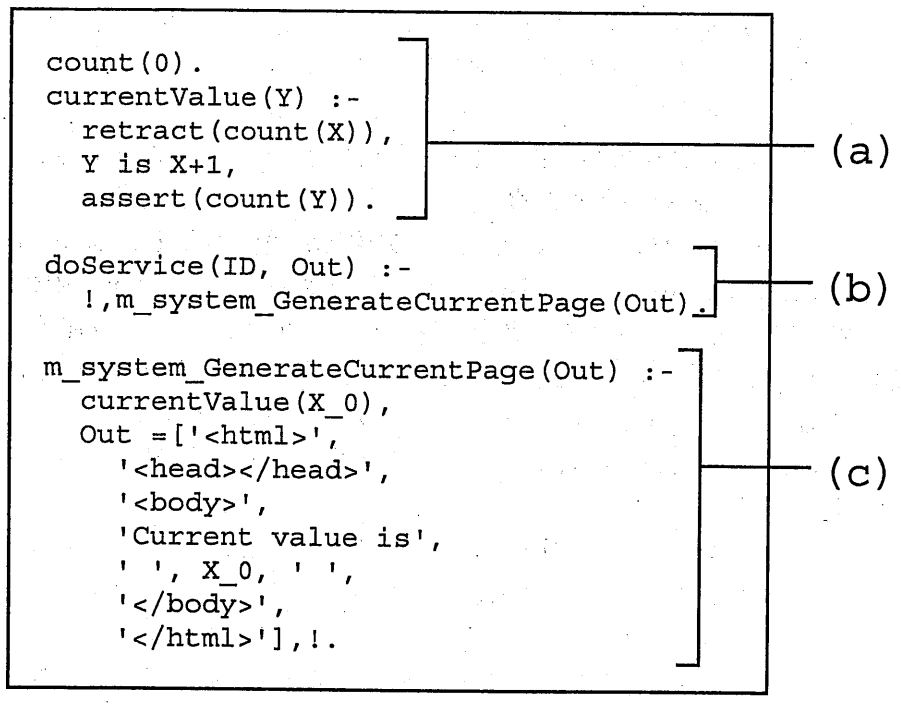

図 5.9: *MiPage* コンパイラによる出力

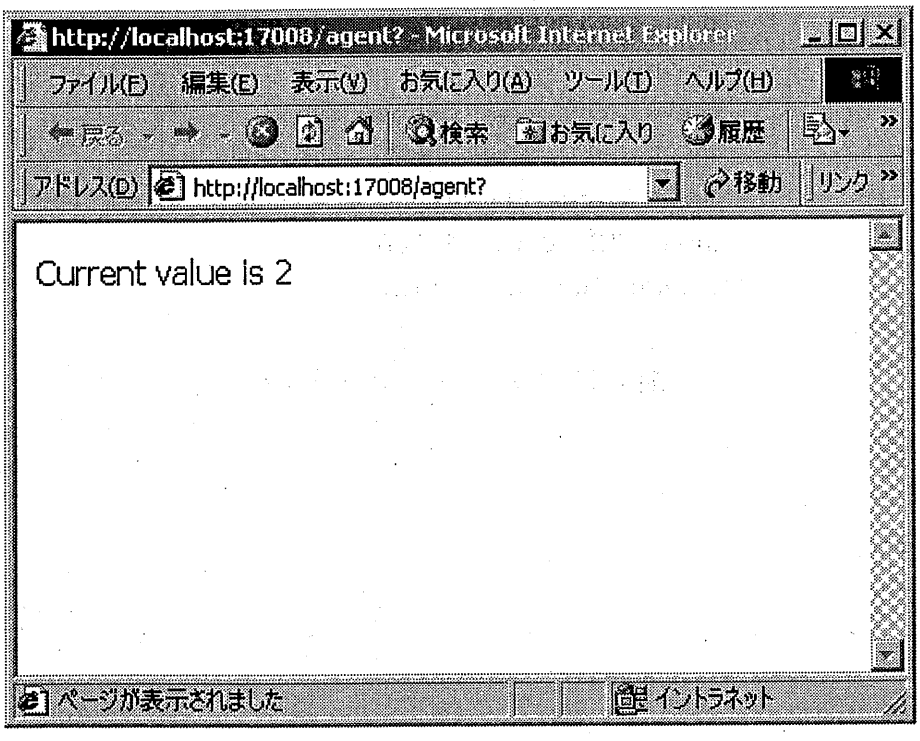

図 5.10: WWW ブラウザ上での表示結果

```
GET agent?item=ruby&action=delete HTTP/1.1
Accept: */*
Accept-Language: ja,en;q=0.5
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Host: 127.0.0.1:17008
Connection: Keep-Alive
```

図 5.11: HTTP リクエストの例

というURLでは，3つのオプションパラメータ (opt1 = val1, opt2 = val2, opt3 = val3) が指定されている．

この場合，それぞれのオプションパラメータに対して，

```
option( agent, opt1, val1).
option( agent, opt2, val2).
option( agent, opt3, val3).
```

という3つの節がデータベースに追加される．ここで， agent はこのHTTPリクエストに割り当てられた識別子を意味し，HTTPリクエストごとに値が変化する場合がある．これらの各オプションパラメータへのアクセスは，

```
?- option( ID, opt1, VALUE).
```

のように，パラメータ名をキーとして問い合わせを実行することで行う．この問い合わせでは，変数 VALUE の束縛として val1 を得ることができる．

図5.12に，HTTPレスポンスの例を示す．HTTPレスポンスは，HTTPリクエストに対する処理の結果，レスポンスのオプションパラメータ，およびレスポンス本体から構成される．図5.12のHTTPレスポンスは，図5.9のプログラムによって生成された*MiLog*エージェントからの応答の例である．

5.2.3 マイグレーションの記述

本節では，*MiPage*プログラム中でのマイグレーションの記述とその実現方法を述べる．*MiLog*エージェントは，HTTPリクエスト処理中に他の*MiLog*実行環境上へ移動することが可能である．ここで課題となるのは，HTTPリクエストの処理結果をどのようにブラウザに返すのか，および移動後のエージェントにブラウザからどのようにアクセスを行うかである．本節では特に，上記2点の実現方法について詳細に述べる．

図5.13に，モビリティを用いた*MiPage*プログラムの例を示す．図5.13の例では，特定のハイパーリンクをブラウザ上でクリックすることにより，そのWWWアプリケーションを特定の計算機上へ移動させることができる．

図5.13のプログラムを実行した場合における処理の過程を示す．まず，ブラウザからのHTTPリクエストをHTTPリクエストルータが受け取る．HTTPリクエストルータはHTTPリクエストを解析し，エージェント間通信を用いて対象とするエージェントにHTTPリクエストの処理を依頼する．処理を依頼されたエージェントでは，Webページをレンダリングするために，問い合わせ ?-currentLocation(X). および?- whereToGo(X). が実行される．問い合わせ ?-currentLocation(X). の実行時における move/1 述語の実行により，エージェントは他の実行環境へ移動する．このとき，移動を行ったことがメタエージェント上で記録される．HTTPリクエストの問い合わせに対する応答は，メタエージェントによって移動先の実行環境からHTTPリクエストを受け取った実行環境へ送られ，WWWブラウザ上に結果が表示される．移動を行ったエージェントに対するアクセスがあった場合には，HTTPのリダイレクト機能を利用してブラウザの接続先をエージェントの移動先ホストに切り替える．なお，図5.13プログラム中の述語 thisHost/1 および concat/2 は

```
HTTP/1.0 200 OK
Content-Type: text/html

<html><head></head><body>
Current value is 1
</body></html>
```

図 5.12: HTTP レスポンスの例

それぞれ，現在いるホストのアドレスを得る，および複数のアトムをつなげて1つの長いアトムを生成する機能を持つ*MiLog*固有の組み込み述語である．

図5.14にリダイレクトを行うためのHTTPレスポンスの例を示す．ブラウザは，図5.14に示されるようなHTTPレスポンスを受け取ると，自動的にリダイレクト先のURLへHTTPリクエストを再発行する．リダイレクト先のURLは，HTTPヘッダフィールド中の 'Location' によって指定される．HTTPリダイレクト機能により，アプリケーションのユーザはエージェントの移動を意識することなくアプリケーションを利用することが可能となっている．

5.2.4 関連研究

本節では，関連研究および関連技術を示し，本フレームワークにおける新規性を明らかにする．

・静的なアプリケーション

WWW上のサービスをモバイルエージェントではなく静的なアプリケーションで構築するための実装方法として，CGIに基づく実装方法，PHP[8],JSP[120],ASP[89]等のページ記述言語を用いる方法がある．*MiPage*では，Webページにモバイルエージェントを埋め込み可能とすることにより，複数のホストを利用したWebアプリケーションの構築を容易としている点が異なる．

・モバイルエージェントの実装技術

これまでに，Telescript[131]やAglets[84]など多くのモバイルエージェントシステムが構築されている．これらのモバイルエージェントシステムの研究成果から，局所的なリソースへのローカルアクセスによる処理の効率化[20]，および既存のアプリケーションの相互接続への適用[77]，が示されている．しかし，これらの研究ではWWWとモバイルエージェントの統合については議論されていない．本フレームワークでは，WWWへのモバイルエージェントの埋め込みについて議論を行っている点が異なる．

・論理プログラミングに基づくWWWの再構築

Lokeらは，文献[88]で論理プログラミングに基づくWWWの再構築について述べている．Lokeらは，ハイパーリンク構造などのWebページ上の情報が論理プログラミングによって宣言的に表現され，WWW上の情報をソフトウェアにとって扱いやすいものとすることで，WWW上の情報を効果的に活用する可能性を示している．本研究ではアプリケーションロジックの記述のためにWebページ中に論理プログラミングを含ませることができるようにしているが，本研究ではアプリケーションを構築するための手法を実現することに重点を置いている点が異なる．

5.2.5 議論

エージェントの移動とセキュリティ

現状の*MiPage*の実装では，エージェントは，移動先と移動元との認証が成功した場合にのみ，移動可能となっている．これは，先行研究である*MiLog*と同様のセキュリティモ

```
<HTML>
<HEAD>
  <SCRIPT LANGUAGE="MiLog">
    currentLocation(X) :-
     option(_,move,LOCATION),
     location(LOCATION,HOSTADR),
     move(HOSTADR),
     thisHost(X),!.
    currentLocation(X) :-
     thisHost(X).

    whereToGo(X) :-
      myname(MYNAME),
      concat(
       ['<A HREF="',MYNAME,
        '?move=1">Go to location 1</A><BR>',
        '<A HREF="',MYNAME,
        '?move=2">Go to location 2</A><BR>',
        '<A HREF="',MYNAME,
        '?move=3">Go to location 3</A><BR>'],
       X).

    location(1,'http://133.68.221.151:17008/').
    location(2,'http://133.68.221.152:17008/').
    location(3,'http://133.68.221.153:17008/').
  </SCRIPT>
</HEAD>
<BODY>
  Current location is
  <!--?-currentLocation(X).-->
  <BR>
  <!--?-whereToGo(X).-->
</BODY>
</HTML>
```

図 5.13: モビリティを用いた *MiPage* プログラムの例

```
HTTP/1.0 302 Found
Content-Type: text/html
Location: http://www-toralab.ics.nitech.ac.jp/
```

図 5.14: リダイレクトを要求する HTTP レスポンス

デルを用いていることによる．*MiLog* では，予め信頼されたホスト同士が共通鍵を持ち，エージェント移動時に移動先のホストと認証を行っている．移動を拒絶する権利は，移動元と移動先の両者が持っており，両者がともに移動を許可した場合にのみエージェントは移動する．エージェント自身は，それを実行するホストからの悪意ある改ざんに対抗するための手段をもっておらず，またホスト側もエージェントの振る舞いに対して制限を加えない．これらのセキュリティ指針は，必ずしも最適なものではないが，プロトタイプアプリケーション作成時に，セキュリティ関連の配慮の負担を最小限とし，アプリケーションの開発の行いやすさを向上させる場合に適していると考えられる．構造的には，*MiLog* エージェントはインタプリタ上で動作するため，サンドボックスモデルに基づく動作の制限は容易であり，移動先環境側でのエージェントの悪意ある行動を制限することは可能である．エージェント自身のセキュリティについては，通信路上での盗聴を防ぐことは可能であるが，悪意あるホストからの盗聴や改ざんを防ぐための手立ては実現されておらず，今後の課題となる．

エージェントの名前管理とエージェント間通信

エージェントの名前は，エージェント自身の名前とそのエージェントが生成されたホストの名前の組み合わせによって表現される．たとえば，ホストの名前を `'133.68.221.183:17008'`，エージェント自身の名前を `'agent'` とすると，エージェントの識別名は `'agent @ 133.68.221.183:17008'` となる．*MiPage* では，各ホストに一意の名前を割り当て，かつ各ホスト上で生成されるエージェント名が一意となるようにすることにより，エージェント名の衝突を避けている．

エージェント間の通信では，*MiLog* で用意されたエージェント間問い合わせ述語，あるいは HTTP アクセスによる通信の 2 つを用いることが可能となっている．*MiLog* では，HTTP サーバ機能のみでなく，HTTP クライアント機能を容易に実現するための組み込み述語を多数用意しており，*MiPage* の提供する HTTP のリダイレクト機能によるスイッチが利用可能となっている．

エージェント間問い合わせ述語では，エージェントの移動履歴に基づいて，問い合わせ先の探索および問い合わせ結果の返信を行う機構が *MiLog* 上において用意されるが，これらの通信はベストエフォート型であり，必ず通信が成功することを保証するものではない．効果的なエージェント間通信，および通信の保証は今後の課題である．

5.2.6 まとめ

本節では，Web ページに埋め込み可能で，かつ移動可能なエージェントを実現するためのフレームワーク *MiPage* を提案した．*MiPage* では，*MiPage* コンパイラを実装することにより，Web ページを表現する HTML 文書へのプログラムの埋め込みを可能とした．*MiPage* では，*MiLog* エージェントフレームワークの拡張として実装することにより，*MiLog* エージェント記述言語をエージェントプログラムの記述に利用可能とした．*MiPage* では，HTTP のリダイレクト機構を利用することにより，エージェントの移動を意識せずにエージェントプログラムの記述を行うことを可能とした．関連研究との差分を示し，今後の課

題として，セキュリティ，およびエージェントの名前管理とエージェント間通信の2点から考察を行った．

5.3 *BiddingBot*: オンライン入札支援システム

5.3.1 はじめに

近年，電子商取引におけるオンラインオークションに対する関心が高まってきている．現在，インターネット上には，eBay[32], Onsale[100], Yahoo! Auctions[6], Amazon.com Auctions[5] など，150以上のオンラインオークションサイトが存在する [30]．エージェント仲介型電子商取引は，近年より高い注目を集めるようになってきている [61]．エージェントは，ユーザの半身として，ネットワーク環境を自律的にかつ協調的に振舞うことが可能である．

オークションサイトは多数あり，扱われる財の数も多数なため，ユーザ自身が多数のオーんラインオークションサイト上でオークションに参加し，財に対して価格の監視および入札を行うことは非常に困難である．ユーザが複数のオークションサイトに並行して参加することを支援するために，*BiddingBot* システム [69] を提案した．本節では，*MiLog* フレームワークのオンラインオークション支援システム *BiddingBot* 構築への適用について述べる．

5.3.2 *BiddingBot* システムの特徴

エージェント仲介型電子商取引には多くの関連研究が存在する．エージェント仲介型電子商取引支援システムは，次の**購買支援システム**および**仮想市場**の2つに分類される．

購買支援システム では，ユーザの希望する商品について，インターネット上の情報の統合および検索機能を提供する．Sherlock 2[4] は，一種のメタサーチエンジンであり，複数のインターネットサーチエンジンを同時に利用することを可能とする．iTrack[72], AuctionWatch[7], および DealTime.com[28] はオンラインサーバであり，オークションサイト上の財の監視を行う機能を提供する．BargainFinder[9], ShopBot[31], および Jango[33] は購入支援エージェントであり，複数のサイト上にある商品に対して価格の比較を行う機能を提供する．これらのシステムは，ユーザに対して，どの商品をいつ購入したらよいかの決定を支援するためのサービス提供を可能としている．

仮想市場 は，エージェントや人間がオークションやその他の市場を作るための場所を提供する．AuctionBot[135], Kasbah[26], FishMarket[104], Tete-A-Tete[60], および eMediator [107] は，これらの仮想市場を実現するシステムの例である．一般的に，これらのシステムが提供するのは，ある種の仮想市場サーバである．仮想市場サーバでは，エージェントは，ユーザの半身として，競合的あるいは協調的に他のエージェントと交渉を行う．ユーザはその仮想市場サーバの提供する API を用いて，自身のエージェントを生成することが可能である．仮想市場サーバは，それぞれに独自のエージェント間交渉プロトコルを提供する．

BiddingBot システムと，上述の購買支援システムとの差異は，上述の購買支援システムがインターネット上からの情報抽出に主眼を置いているのに対して，*BiddingBot* システムは財に対する入札を行うことを可能としている点である．仮想市場では，エージェントはそ

れらの提供する仮想的なオークションや市場に参加することが可能であるが，*BiddingBot* では実際に運営されている既存のオンラインオークションに参加することが可能である点が異なる．

5.3.3 *BiddingBot* の構成

図 5.15 に，*BiddingBot* システムのアーキテクチャを示す．*BiddingBot* は，1つのリーダエージェントといくつかのビッダーエージェントから構成される．各ビッダーエージェントは1つのオークションサイトごとに1つづつ割り当てられる．各エージェントは，*MiLog* フレームワークを用いて実装される．ビッダーエージェントは協調的に動作し，複数のオークションサイトに対して並行して，情報の収集，監視および入札を行う．

エージェント間の通信は，エージェント間問い合わせ，および節データベース共有を用いて実装されている．リーダーエージェントは，ビッダーエージェント間の協調において，マッチメーカーとしての役割を果たし，同時にユーザの希望を受け取り入札情報を提示するユーザインタフェースエージェントとしての役割も果たしている．リーダーエージェントは，ユーザとのインタラクションに WWW を利用している．*BiddingBot* システムのユーザインタフェースを図 5.16 に示す．図 5.16 前面（右側）にあるウィンドウは，ユーザのログイン画面である．ユーザは，ここでユーザネームを入力することにより，システムにログインする．システムにログイン後，ユーザは本システムを用いて，購入を希望する財についての情報収集を行う．本システムでは，複数のオンラインオークションサイトに対するメタサーチエンジンの機能を有する．図 5.16 背面（左側）にあるウィンドウは，オンラインオークションサイトからの財の検索結果を表示している．財の名前，価格，落札までの期限等の情報から，落札すべき財の候補をシステムに入力する．システムは，入札価格の変化に応じて協調的に入札を行い，財への入札に関するユーザの負担を軽減する．*BiddingBot* システムを用いることにより，ユーザは複数のオンラインオークションサイト上にある財を統一的に扱うことが可能となる．

WWW を用いたユーザとのインタラクションの実現には，5.2 節で述べた *MiPage* フレームワークを利用している[1]．協調プロトコルは，有限状態機械として実装され，他のエージェントからの問い合わせの受信によって状態が遷移する．*BiddingBot* における協調プロトコルの1つであるスイッチングプロトコルの *MiLog* による記述の概要を示す．リーダーエージェントのプログラムは，以下のように記述される．

```
1 propose(Agent,NewBid) :-
2  currentBid(CurrentBidder,_),
3  request(Agent,doNegotiate(NewBid,CurrentBidder)).
4
5 propose(Agent,NewBid) :-
6  request(Agent,accept(NewBid)).
7
```

[1] *BiddingBot* と *MiPage* の開発は並行して行ったため，厳密には 5.2 節で述べた機構から *MiPage* コンパイラを除いた，より原始的な機能を *BiddingBot* の実装に用いた

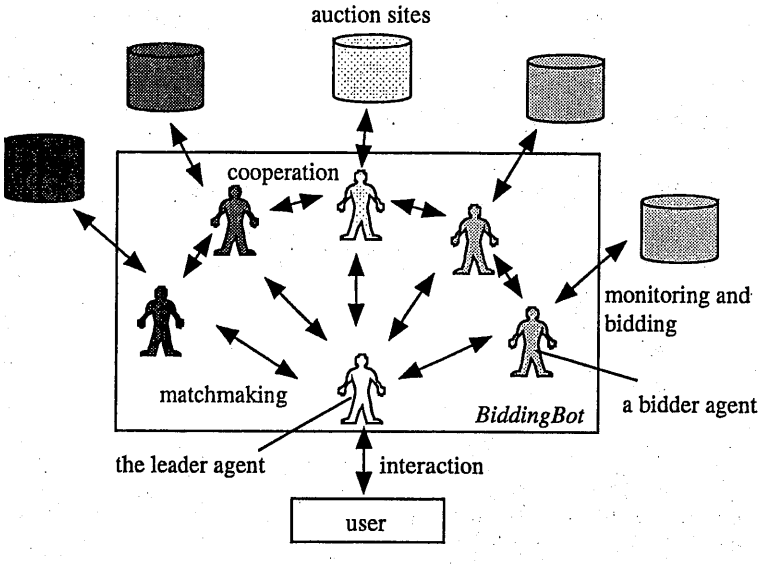

図 5.15: *BiddingBot* システムの構成

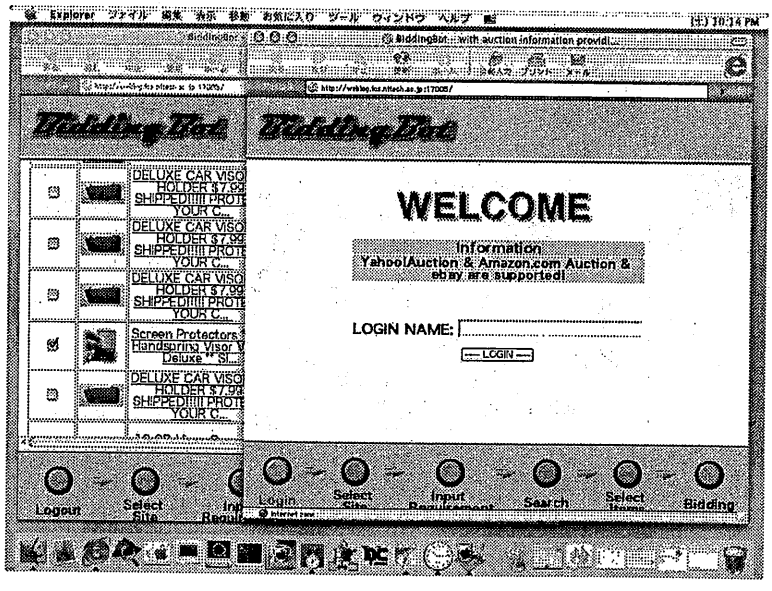

図 5.16: *BiddingBot* システムのユーザインタフェース

```
 8 announce(X) :-
 9  assert(X).
```

ビッダーエージェントのプログラムは，以下のように記述される．

```
 1 do :-
 2  update,doPropose, sleep(10000).
 3
 4 doPropose :-
 5  biddable(NewBid),myname(MyName),
 6  request(leader,propose(MyName,NewBid)).
 7
 8 doNegotiate(Bid,CurrentBidder) :-
 9  negotiate(CurrentBidder,Bid).
10
11 negotiate(CurrentBidder,Bid) :-
12  myname(MyName),request(CurrentBidder,propose(MyName,Bid)).
13
14 accept(propose(NewBid)) :-
15  doBid(NewBid), announce(NewBid).
16
17 announce(Bid) :-
18  myname(MyName),request(leader,announce(MyName,Bid)).
19
20 propose(Agent,ProposedBid) :-
21  currentBidder, betterThanCurrent(ProposedBid),
22  cancelCurrentBid,
23  request(Agent,accept(propose(ProposedBid))),!.
24
25 propose(Agent,ProposedBid) :- !,fail.
```

協調は，ビッダーエージェントが2行目の update/0 の実行により，財の入札情報を更新した後に，doPropose/0 の実行により新たな入札の可能性を発見したところから開始される．新たな入札の可能性が見つかると，ビッダーエージェントはリーダーエージェントに問い合わせ propose/2 を行う．この問い合わせにより，リーダーエージェントのプログラムの1行目 propose/2 が実行される．既に入札を行っている他のエージェントが存在した場合（2行目），入札を提案したビッダーエージェントにそのエージェントとの交渉を行うように問い合わせを行う．ビッダーエージェントはその問い合わせにより，他のビッダーエージェントと交渉を行う（8行目から１２行目）．交渉を持ちかけられたエージェントは，現在の入札結果と提案された入札との間で効用を比較し（２１行目），提案された入札がより効用が大きい場合に，現在の入札をキャンセルし（２２行目），提案を行ったエージェントに提案された入札を遂行するように問い合わせを行う（２３行目）．入札を提案したビッダーエージェントは，提案した入札を遂行した後，リーダーエージェントに交渉

の結果を報告する（１５行目）．*MiLog*の持つエージェント間問い合わせ述語を利用することにより，*BiddingBot*における協調プロトコルが簡潔に記述されている．

*BiddingBot*では，オークションサイトごとの表現の違いを吸収するために，各ビッダーエージェントごとに対応するオークションサイトが決定され，柔軟なラッパーとして振舞う．ラッパーとは，HTML等のタグ付け文書などから，アプリケーションの処理に必要な情報を抽出するソフトウェアプログラムである．ビッダーエージェントにおけるラッパー機能は，*MiLog*におけるパターン記述補助機能を用いて実装されている．ラッパー機能によってオークションサイトから抽出された情報は，*MiLog*を用いて図5.17の表現で格納される．変数のプレフィックスの+および-は変数のモードを示しており，+が入力，-が出力を意味する．各情報の意味は，各行の右側にコメント文で示してある．オークションサイトからの情報抽出は，財に対するキーワードを用いた概要検索，および財ごとの詳細な情報の抽出から構成される．

オークションサイトからの概要検索の実装例を図5.18および図5.19に示す．本実装例は，オンラインオークションサイトBIDDERSにおけるラッパーの実装例である[2]．プログラムが大きいので，概要のみを示す．プログラム本体は，Webページの取得（図5.18の２行目，および５行目から２７行目まで），および取得したWebページからの情報抽出（図5.18の３行目，および２９行目から図5.19の８６行目まで）の２つの部分から構成される，図5.18の９行目から２７行目までは，オークションサイトにアクセスする際に必要となるURLを生成している．生成したURLを用いて，Webページの取得が行われる（図5.18の７行目）．取得されたWebページは，*MiLog*の組み込み述語によりHTMLタグと本文に分離され，リストの形式で格納される．取得したWebページのHTMLソースから，コメント文およびタグの構造に対するパターンマッチングを行い，必要な情報を抽出する（図5.19の４３行目から６２行目）．ここで，述語`extract/2`は，第１引数で指定したリストの要素をすべて連結した結果が第２引数で指定したリストとなるようにパターンマッチングを行う述語であり，１０行程度のPrologプログラムによってバックトラック処理を効果的に利用して記述されている．抽出した情報は，図5.19の６４行目から７４行目のプログラムにより，図5.17の形式で節データベースに格納される．図5.19の７６行目から８６行目までのプログラムは，オークションサイトから得られた検索結果が１つのWebページに表示しきれない場合に，再帰的に残りの検索結果を抽出するためのプログラムである．

本ラッパーの実装の特徴は，論理型言語における単一化とバックトラッキング機構を利用して，パターンマッチングによるヒューリスティックな情報抽出機構を簡潔に記述している点である．本ラッパーの実行速度は，500MHzのCPUを搭載したノート型PC上で実行した場合でもWebページの取得に要する時間よりも十分に小さく（１秒以下），動作速度上の問題は特に生じなかった．

5.3.4 議論

表5.2に，*BiddingBot*システムの実装に要した*MiLog*ソースコードとその大きさを示す．*BiddingBot*システムは，11の*MiLog*プログラムコードと，58のデータファイル(HTMLや

[2] オークションサイトのページ構成はサイトの管理者によって常に更新されていくので，ここで示したプログラムが適用できない形式にページ構成が更新されている可能性がある．

```
getAbstractItemListWithKeywords( +Keywords, -IDList ).
getDetails( +ID ).
% ----- abstract で取ってくる内容 -----
% (required)
itemName( +ID, -Name ).                          % 財の名前
itemPrice( +ID, -Price ).                        % 財の（現時点での）価格
itemURL( +ID, -URL ).                            % 財の詳細を書いたページの URL
% (optional)
itemBids( +ID, -Bids ).                          % 財に対する入札の数
itemEnd( +ID, -End ).                            % 財への入札が終了する時刻
% ----- detail で取ってくる内容 -----
% (required)
itemName( +ID, -Name ).                          % 財の名前
itemPrice( +ID, -Price ).                        % 財の（現時点での）価格
itemURL( +ID, -URL ).                            % 剤の詳細を書いたページの URL
itemBids( +ID, -Bids ).                          % 財に対する入札の数
itemEnd( +ID, -End ).                            % 財への入札が終了する時刻
itemBidder( +ID, -BidderID ).                    % 現在の最高額入札者
itemBidIncrement( +ID, -Increment ).             % 入札金額の増分
itemStartPrice( +ID, -StartPrice ).              % 入札開始価格
% (optional)
itemSeller( +ID, -SellerID ).                    % 売り手
itemID( +ID, -IDonTheSite ).                     % 財の（そのサイトでの）ID
itemAuctionType( +ID, -AuctionType ).            % オークションの種別
itemPaymentMethod( +ID, -PaymentMethod ).        % 支払方法
itemCondition( +ID, -Condition ).                % 財の状態（新品，中古など）
itemDescription( +ID, -Description ).            % 財に対する説明
```

図 5.17: オークションサイトから抽出された情報の表現

```
 1 getAbstractItemListWithKeywords( Keywords, IDList ) :-
 2   getItemListHTML( Keywords, HTML ),
 3   !,w_ripAbstractData( HTML, IDList ),!.
 4
 5 getItemListHTML( Keywords, HTML ) :-
 6   makeQueryURL( Keywords, URL ),
 7   !,getParsedHTML( URL, HTML ).
 8
 9 makeQueryURL( Keywords, URL ) :-
10   concatKeywordsWithBlank( Keywords, Cat ),
11   concat( ['?form_name=key_word&where=auction_status%3D0&',
12            'orderby=isnull%28et.categ_featured%2C0%29+desc',
13            '+%2C+dead_line&categoryid=&check=Y&title=',
14            Cat], Param ),
15   searchURL( SURL ),
16   !,concat( [SURL, Param], URL ).
17
18 siteURL( 'http://www.bidders.co.jp/' ).
19
20 searchURL( URL ) :-
21   siteURL(SiteURL),
22   !,completeURL( SiteURL, '/servlets/NOR1_1', URL ).
23
24 concatKeywordsWithBlank( [Keyword], Keyword ) :- !.
25 concatKeywordsWithBlank( [Keyword|Rest], Cat ) :-
26   concatKeywordsWithBlank( Rest, PCat ),
27   !,concat( [ Keyword, '+', PCat ], Cat ).
28
29 w_ripAbstractData( HTML, IDList ) :-
30   w_ripAbstractDataFromThisPage( HTML, IDList1 ),
31   w_ripAbstractDataFromNextPages( HTML, IDList2 ),
32   append( IDList1, IDList2, IDList ),!.
33
34 w_ripAbstractDataFromThisPage(HTML, IDList) :-
35   cutAbstractBody( HTML, BODY ),
36   !,w_ripAbstractBody( BODY, IDList ).
37
38 w_ripAbstractDataFromNextPages( HTML, IDList2 ) :-
39   w_ripNextPage(HTML,NEXT),
40   !,w_ripAbstractData( NEXT, IDList2 ).
41
42 w_ripAbstractDataFromNextPages( HTML, [] ) :- !.
```

図 5.18: 情報抽出プログラムの例

```
43 cutAbstractBody( HTML, BODY ) :-
44   extract( [ _, [['!',['-- リスト --']]], BODY1], HTML ),
45   !,extract( [BODY2,[['!',['-- /プログラムで生成 --']]],_], BODY1 ),
46   !,extract( [_, [[table|_]], BODY3], BODY2 ),
47   !,extract( [_, [[tr|_]], _, [[tr|_]], _, [['/tr']], BODY ], BODY3 ).
48
49 w_ripAbstractBody( HTML, [ID|IDList] ) :-
50   w_ripAbstractDataItem( HTML, ITEMDESC, REST ),
51   !,storeAbstractData( ID, ITEMDESC ),
52   !,w_ripAbstractBody( REST, IDList ).
53
54 w_ripAbstractBody( _, [] ) :- !.
55
56 w_ripAbstractDataItem( HTML, [URL,NAME,PRICE,ENDTIME,BIDS],REST ) :-
57   extract( [_, [[tr|_]],BODY,[[tr|_]],REST1], HTML ),
58   !,extract( [_, [[td|_]], _,[[td|_]], _, [[a,[href,URL]],NAME], R1],BODY ),
59   !,extract( [_, [[td|_]], _, [[td|_], _, PRICE], R2], R1 ),
60   !,extract( [_, [[td|_]], _, [[td|_],_,ENDTIME], R3], R2 ),
61   !,extract( [_, [[td|_],_,BIDS],R4], R3 ),
62   !,extract( [_, [['/tr']],REST], REST1 ).
63
64 storeAbstractData( ID, [URL,NAME,PRICE,ENDTIME,BIDS] ) :-
65   generateID( ID ),
66   assert( itemURL(ID, URL) ),
67   assert( itemName(ID, NAME) ),
68   assert( itemPrice(ID, PRICE) ),
69   assert( itemEnd(ID, ENDTIME) ),
70   !,assert( itemBids(ID, BIDS) ).
71
72 generateID( ID ) :-
73   generateTmp(N),
74   !,concat([sample,N],ID).
75
76 w_ripNextPage( HTML, NEXT ) :-
77   w_ripNextPageURL( HTML, URL ),
78   getParsedHTML( URL, NEXT ),!.
79
80 w_ripNextPageURL( HTML, URL ) :-
81   extract( [_, [['!',['-- 次の何件/リンク --']]],LINK,
82                [['!',['-- /次の何件/リンク --']]],_    ],HTML ),
83   !,extract( [_, [A,'次の30件'], _ ], LINK ),
84   !,extract( [_, [[href,URL1]], _], A ),
85   !,siteURL(SITEURL),
86   !,completeURL( SITEURL, URL1, URL ).
87
```

図 5.19: 情報抽出プログラムの例 (続き)

GIF ファイルなど) から構成される．プログラムコード全体の大きさは，87281bytes である．最も多くのコードを必要としたモジュールは，協調機構 (約 38KBytes) であった．これは，システムの開発者がシステムの実装上本質的に重要な部分に多くのコードを裂くことができたことを示している．2番目に多くのコードを必要としたモジュールは，ラッパー (約 33KBytes) であった．ラッパーは，WWW ページ等の半構造データからの情報抽出方法を記述したモジュールである．これまでに，ラッパーを自動生成する研究も行われているが，実際の WWW ページの構造は非常に複雑であり，それらの研究を適用できない場合も多い．したがって，本システムではラッパーをプログラムとして記述している．

ユーザインタフェースの構築に必要なコードサイズは，HTML に基づくユーザインタフェースを導入したために，非常に小さくなっている．HTML に基づくユーザインタフェースに必要なデータファイルはある程度大きいが，これらは既存のデザインアプリケーションを用いて生成可能なものであり，プログラマの負担の増加にはつながらない．本システムは，実験目的のアプリケーションとして十分に利用可能であった．

5.3.5 その他の応用事例

BiddingBot システム以外のアプリケーションへの適用事例を簡単にまとめる．

GroupBuyAuction[136]: GroupBuyAuction は，ボリュームディスカウントを可能としたエージェントに基づく仮想市場サーバである．GroupBuyAuction では，買い手エージェントは互いにグループを作って共同購入を行うことが可能であり，売り手エージェントはそれらのグループからの一括購入に対して値引きを行うことが可能である．売り手エージェントと買い手エージェントはともに，*MiLog* を用いて実装されている．エージェントが交渉を行う場 (Market Place) も，*MiLog* によって実装されている．

会議スケジューリングシステム [114][128]: 我々はこれまでに，2種類のスケジューリング機構に基づく会議スケジューリングシステムを開発している．1つは，エージェント間の多重交渉に基づくもの [114] であり，もう1つは，重み付き分散制約充足機構に基づくものである [128]．いずれの機構を用いた場合も，エージェントは移動しながら継続的に推論を行う必要があり，*MiLog* によって効果的に実現される．

Group Choice Design Support System(GCDSS)[62][71]: GCDSS[71] は，エージェントに基づく意思決定支援システムであり，グループにおける代替案選択を支援する．本システムでは，エージェントはそのユーザの半身として互いに交渉を行う．交渉機構として，我々は説得に基づく機構 [71]，および議論に基づく機構 [62] を提案している．これらの交渉機構の実現に，*MiLog* の論理プログラミングの枠組みが活用されている．

表 5.2: *BiddingBot* プログラムのソースコードサイズ

module name	n of files	total size(bytes)
cooperative behavior	3	37914
wrapper	4	33136
user interface(code)	2	16231
user interface(data)	58	92101
other	2	865

第6章

結論

6.1 成果

本節では，本研究で得られた成果を，各章ごとにまとめる．

第3章では，知的モバイルエージェントフレームワーク *MiLog* の設計ついて述べた．知的モバイルエージェントと外界のモデル化を行った．知的モバイルエージェントが対象とする環境の特性を明らかにした．知的モバイルエージェントが環境を適切に扱うために持つべき性質をまとめた．知的モバイルエージェントを実現するために，フレームワークが提供すべき機能について議論した．知的モバイルエージェントが持つべき性質を実現するための機能について述べた．知的モバイルエージェントを用いてシステムを構築するために必要な機能を述べた．知的モバイルエージェントの記述言語として，*MiLog* エージェント記述言語の設計を行った．*MiLog* エージェント記述言語の定義を示した．*MiLog* エージェント記述言語によるエージェントプログラムの記述方法について述べた．知的モバイルエージェントフレームワーク *MiLog* の実行環境の設計を行った．*MiLog* 実行環境の構成を示した．*MiLog* 実行環境におけるエージェントプログラミング環境について述べた．*MiLog* 実行環境の拡張手段の設計と実現について述べた．*MiLog* 実行環境の JAVA プログラムからの利用方法について述べた．JAVA プログラムによる *MiLog* エージェントインタプリタ *MiLogEngine* の拡張方法について述べた．*MiLog* フレームワークの設計ついて，議論をまとめた．本章の初期の成果は，文献 [50],[44],[51],[52],[54] により公表した．本章の主要な成果は，文献 [43],[45],[47],[46] により公表した．

第4章では，論理型言語への任意時刻モビリティの実現を目的として，*MiLog* 言語の解釈実行処理系 *MiLogEngine* を実装した．任意時刻モビリティと既存のモビリティの実現手法との比較を行い，任意時刻モビリティの特性と実装上の課題を明らかにした．*MiLogEngine* の動作モデルを示した．マイグレーション時におけるプログラムの実行継続を実現するために，*MiLogEngine* におけるプログラムの実行状態のデータ構造の設計を示した．プログラムの解釈実行を行うインタプリタコアの設計を示した．インタプリタコアにおける末尾呼び出し最適化手法の実装を示し，その評価を行った．モビリティ等の *MiLog* の機能をプログラム中から利用できるようにするためのアプリケーションプログラミングインタフェース (API) の実装について述べた．*MiLogEngine* の JAVA 上での実装に固有の課題について議論し，その解決法を示した．本実装の評価を行い，本実装の有用性を確かめた．本章の

成果は，文献[43],[53],[46]により公表した．

第5章では，2つのアプリケーション構築フレームワークへの適用，および電子商取引支援システムの実装への適用を行い，本フレームワークの有効性を確認した．

GUIを持つモバイルエージェントを構築するための，論理プログラミングに基づくアプリケーションフレームワーク*iML*を提案した．*MiLog*を用いてGUIを持つモバイルエージェントの構築を行う際の課題は，GUIから発生したイベントに対する処理を論理プログラムの文法で統一的に記述可能とすること，GUIをエージェントに伴って移動させるための機構を実現すること，および，GUIを容易に構築可能な支援環境を実現することである．*iML*では，GUIから発生したイベントを制御するための機構GCMをJAVA言語により実装し，*MiLog*に拡張機能として持たせることにより，*MiLog*によるイベント処理の記述を実現した．*iML*では，GUIの状態をホーン節の形式で節データベースに保存する機構を実現した．本機構と*MiLog*の持つモビリティを組み合わせることにより，GUIをエージェントに伴って移動させる機能を実現した．*iML*では，GUIの設計における負担を軽減するために，3つのモードを持つ視覚的デザインツールを構築した．視覚的デザインツールでは，概観設計，詳細設計，および動作試験を単一の環境内で繰り返し行うことを可能とすることで，GUIの設計における設計者の負担を軽減した．*iML*により構築したエージェントのマイグレーション時の効率を，時間と通信帯域の使用量の2点から計測した．JAVAに基づくシリアライズを用いた手法との比較を行い，本手法が既存の手法と比較して十分な性能を持つことを示した．本フレームワークを，大学におけるプログラミング演習に適用した．プログラミング演習における学習者からのフィードバックなどにより，*iML*フレームワークがプログラミング教育に適用可能であることと，学習者の興味を喚起することで学習効果が高まる可能性の有無を考察した．本評価により，性能と扱いやすさの観点から，本フレームワークの有効性が示された．これらの成果は，文献[41],[49],[92]により公表した．

知的モバイルエージェントを用いてWeb上のサービスを構築するためのフレームワーク*MiPage*を提案した．本章における課題は，知的モバイルエージェントのWWWとの統合の実現，およびWWWの標準であるHTMLを利用したユーザインタフェースの記述と*MiLog*との融合である．知的モバイルエージェントのWWWとの統合の実現では，エージェントをWWW上のアプリケーションの1つとして実現すること，および，エージェントの移動時におけるWWW上のアドレス(URL)更新を実現すること課題である．エージェントをWWW上のアプリケーションの1つとして扱うために，*MiLog*実行環境にWWWサーバ機能を組み込み，WWWクライアントからの要求をエージェントに受け渡すための機構を実現した．エージェントの移動におけるURLの更新への対処として，エージェントに対するWWWクライアントからの要求に対して，エージェントの移動先を通知しWWWクライアントを再接続させる手法を考案し，実装を行った．これらの機構により，モバイルエージェントをWWW上のアプリケーションとして運用するための基盤が実現された．HTMLを利用したユーザインタフェースの記述と*MiLog*との統合の手段として，HTML文書中に*MiLog*で記述したプログラムの埋め込みを可能とした．*MiLog*プログラムが埋め込まれたHTML文書を*MiLog*のプログラムとして利用可能とするために，*MiPage*コンパイラを実装した．*MiPage*コンパイラは，HTML文書の構造解析を行い，埋め込まれた*MiLog*プログラムに基づいて動作する*MiLog*プログラムの生成を行う．*MiPage*コンパイラによって生成された*MiLog*プログラムに基づき，実際の処理を行うエージェントを

生成する機構を実装した．本フレームワークの評価として，学生のプログラミング演習への適用を行った．本演習における学生からのフィードバックから，考察を行った．本実験結果から，本フレームワークが利用者のプログラミング技術にかかわらず利用されるという特徴が観測された．本フレームワークが，プログラミング初心者にとって受け入れやすい性質を持つことが示された．これらの研究成果は，文献 [48] で公表した．

これまでの章で実装を行ったフレームワークを，オンラインオークション入札支援システム *BiddingBot* の実装に適用し，本研究の実システムへの適用可能性について調査した．*BiddingBot* システムの目的は，オンラインオークション上に多数ある財について，ユーザの入札額決定を効果的に支援し，合理的な判断に基づいて入札作業の代行を実現することである．*BiddingBot* システムは，オークションサイトからの効果的な情報収集，および効果的な入札戦略を実現するために，複数エージェントによる協調機構を用いる．*BiddingBot* の実装における課題は，エージェントに合理的な判断を行うための機構を持たせること，エージェント間での協調機構により効果的な入札戦略を実現すること，および，これらの機構を実現したアプリケーションを構築することである．*MiLog* フレームワークは，論理プログラミングによるエージェントの記述基盤を提供することにより，合理的な振る舞いを行うエージェントを簡潔に記述可能とした．エージェント間の通信基盤を提供することにより，エージェント間の協調を簡潔に記述可能とした．エージェントを WWW 上のアプリケーションとして運用するための機構を実現することにより，アプリケーションの構築におけるユーザインタフェース構築の負担を軽減した．*BiddingBot* の実装で，本フレームワークが開発者にエージェントの合理的な行動および協調動作に労力を集中可能としたことを，*BiddingBot* のソースコードにおけるコード量の分析から示した．*BiddingBot* の実装に必要としたソースコードが比較的小さいことから，本フレームワークが *BiddingBot* を簡潔に実装可能であることを示した．以上の *BiddingBot* システムのソースコード分析から，合理的エージェントに基づくアプリケーションの実現に，本フレームワークが有効であることを示した．これらの成果は，文献 [39],[40],[42],[70],[69] として公表した．

6.2 結論

本論文の研究の背景で，知的エージェントとモバイルエージェントでの記述言語の性質の違いについて述べ，その克服を課題の１つとして示した．エージェント記述言語 *MiLog* の実現により，知的モバイルエージェントに適した記述言語が不足しているという課題に対する解決方法を提案できた．記述言語の実現では，言語としての最適性および実装としての最適性が議論されなければならない．言語としての最適性とは，必要な機能を簡潔に記述可能であることを意味する．実装としての最適性とは，必要な機能が必要な効率で実現されることを意味する．言語としての最適性については，*BiddingBot* の実装における考察結果から，十分な成果が得られた．実装としての最適性では，エージェントの移動に関する効率性の面で十分な結果が得られた．一方で，エージェントプログラムの実行速度の高速性については，コンパイラ方式の実装を検討する価値があり，今後に課題を残した．セキュリティ機構の実現については，本論文では必要最小限の実装にとどめ，今後の課題としている．

序論で，本研究の３つの目的をあげた．本研究の目的は，目的１：知的モバイルエージェ

ントの記述言語と構築環境の設計，目的2：知的モバイルエージェント記述処理系の実装と高性能化，および目的3：知的モバイルエージェントの応用であった．

目的1に関して，知的モバイルエージェントが扱う環境の特性として，開放性，可変性，対称性，および資源偏在性を持つことを明らかにした．次に，知的モバイルエージェントが持つべき性質として，自律性，即応性，相互操作性，および移動性があることを示した．エージェントフレームワークが支援すべき点として，自律性の実現の支援，即応性の実現の支援，相互操作性の実現の支援，移動性の実現の支援を挙げた．エージェントを用いたアプリケーションを開発するためにフレームワークが提供すべき機能として，エージェントプログラムの実行可能性，エージェントの拡張容易性，エージェントプログラムの改変容易性，エージェントの振る舞いの理解容易性，エージェントのメタ操作性，および外界との通信可能性を挙げた．これらを踏まえ，知的モバイルエージェント記述言語として，問い合わせに基づくエージェント間通信，および即応的な移動を実現する任意時刻モビリティを備えるエージェント記述言語 *MiLog* の設計を示した．知的モバイルエージェントの構築環境として，*MiLog* 言語を用いたプログラミングフレームワーク *MiLog* を実現した．目的1における知的モバイルエージェント記述言語と構築環境の設計は，*MiLog* フレームワークとして実現できた．

目的2に関して，エージェント記述言語 *MiLog* の処理機構として，*MiLogEngine* を示した．*MiLogEngine* では，任意時刻モビリティの実現に必要となる割り込み処理と強モビリティを実現するために，実行状態スタックの実現，および基本実行単位での実行状態スタックの保存および復元を可能とした．すなわち，*MiLogEngine* を用いたエージェント記述言語 *MiLog* の実装を示した．*MiLogEngine* ではモビリティの高性能化のための機構を提供した．本機構により，従来の処理系の実装と比較して高いモビリティ性能が実現できた．

目的3に関して，知的モバイルエージェントの応用のためにアプリケーションフレームワーク *iML*，および *MiPage* を実現した．*iML* では，GUIを持つモバイルエージェントアプリケーションの構築を容易とした．*MiPage* では，WWWをユーザインタフェースとするWWWアプリケーションの構築への知的モバイルエージェントの適用を容易とした．これら2つのアプリケーションフレームワークとして，知的モバイルエージェントの応用を支援するための枠組みを実現した．知的モバイルエージェントの応用として，オンラインオークション入札支援システム *BiddingBot* の構築への適用を行った．*BiddingBot* におけるソースコード量の解析結果から，本論文における知的モバイルエージェントフレームワークによってアプリケーションの記述を簡潔に行うことができたことを示した．

以上から，本論文における3つの目的は達成できた．

6.3 今後の課題

本節では，今後の課題について議論し，本研究の他分野への応用の可能性について述べる．

モバイルエージェントの有効性と限界 モバイルエージェントを扱う上で常に起きる問いかけに，モバイルエージェントのキラーアプリケーションは何かという問いがある．これまでに，関連国際会議や研究会など様々な場でモバイルエージェントのキラーアプリケーションに関する議論が行われてきた．この問いについて議論する際に最も重要なのは，モ

バイルエージェントをモデルとして扱うか，それとも実装方法（あるいは，もっと極端な例ではライブラリ）として扱うかである．モバイルエージェントのモデルを実現するアプリケーションソフトウェアは，モバイルエージェントシステムやモビリティを特別に利用することなく，既存の技術のみを用いて実現することが可能である[1]．

モバイルエージェントに基づくソフトウェアの運用が実現されると，計算機資源の提供（あるいは販売）が容易になり，そこに新たな市場が形成されることが期待される．この市場を通じて，例えば，余ったCPUパワーをユーザが他へ売却したり，あるいは企業が高度な計算を行うためのCPUパワーを多くのユーザから購入することが可能となる．計算機資源が流動的に扱われることにより，計算機資源の効果的な運用が可能となるため，必要な計算機資源をより安価に入手できるようになることが期待される．

モバイルエージェントを実装技術として扱う場合の利点は，モバイルエージェントに関連する有用なライブラリの利用が可能な点である．モバイルエージェントは，特定のアルゴリズムや処理そのものの性能を向上させるための技術を提供しないが，それらの設計や実装を容易に行うためのモデルと基盤を提供する．モバイルエージェントの利点が既存の実装手法でも実現可能であるという指摘があるが，その利点の実現を最も容易に行う方法はモバイルエージェントを利用することであると考えられる．しかし，モバイルエージェントを用いたほうが容易に実装が行えるということは，直感的には想像されない．これは，我々の多くが，既存の実装技術と比較して，モバイルエージェントを用いたアプリケーションソフトウェアの実装に関する知識を十分持っていないことが原因であると考えられる．モバイルエージェントは，そのモビリティという特性から，実装が難解であるという印象を受ける．しかし，モビリティという性質は，多くの場合モバイルエージェントシステムやライブラリが提供するため，モビリティの利用者はその実装を考慮する必要はほとんどなく，モビリティの利用自身は，むしろ容易である．適切な設計がなされたモバイルエージェントシステムであれば，アプリケーションソフトウェアの実装におけるモビリティの利用は容易である．プログラミングにおけるモビリティの利用の容易性が広く理解されるようにすることが，今後の課題の１つである．

高速化の余地　言語処理系の実装では，実行速度の高速化は重要な課題の１つである．モビリティの実現には，異なったアーキテクチャの計算機上における移動，およびコードや実行状態の転送の実現を必要とする．異なるアーキテクチャ間で共通のソフトウェアを動作させる方法の１つに，共通のコードで記述されたソフトウェアをアーキテクチャごとに実行時コンパイル (Just In Time Compile) する手法がある．実行時コンパイルにより，実行速度を高めながら，共通のソフトウェアを活用することが可能となる．実行時コンパイルを用いる手法よりも，特定のアーキテクチャに特化した実装を行ったほうが実行速度が高速になる場合がある．例えば，特定のCPUの持つSIMD(Single Instruction Multiple Data)命令を利用した場合，特定の演算を飛躍的に高速化できる可能性がある．このような，アーキテクチャの特性を最大限に生かしたソフトウェアの実現が必要な場合には，アーキテクチャに特化した設計を行うほうが有利である．アーキテクチャ固有の機能を生かしたソフトウェアをモジュールとして用意し，それらをモバイルエージェントから利用可能とする

[1] モバイルエージェントシステムやそれによって提供されるモビリティが，モバイルエージェントの開発の負担を実際に軽減してくれることは，いうまでもない．

ことにより，運用の柔軟性と性能の向上をバランスよく実現可能である．アーキテクチャの種類が特定可能な場合には，モバイルエージェント自身にアーキテクチャ固有のコードを一部持たせる方法も考えられるが，余分にコードを持つことが原因となってエージェントが肥大化し，エージェントの移動処理オーバヘッドの増大を招く可能性がある点，および，実行環境上でアーキテクチャ固有コードを安全に実行可能とするために複雑なセキュリティ機構の実装が必要になる点に注意が必要となる．

強モビリティでは，実行状態を伴ってエージェントが移動するため，アーキテクチャ固有のコードを実行中にエージェントの移動を実現することは困難である．コンパイル後のコードは，メモリ空間の使用よりも実行速度の向上を優先している場合が多いため，コンパイル後のコードの実行状態をキャプチャして転送することはあまり効率的でなく[2]，オペレーティングシステム等の実装方法によってはキャプチャそのものが困難である場合も存在する[3]．これらの理由を考慮し，本研究ではエージェントの実行環境としてインタプリタ方式を選択している．インタプリタ方式は，デバッグの行いやすさおよび処理系の拡張の容易さの点で優れているが，実行速度の点ではコンパイラ方式のほうが優れている．本研究の発展として，実行状態をキャプチャする必要のない部分を選択的に実行時コンパイルする手法の実現が挙げられる．

セキュリティと匿名エージェント 世界中の計算機資源を匿名で利用（共有）できるようにするために，P2P などによる様々な試みが行われている．モバイルエージェントをこの目的で利用可能とするためには，強固なセキュリティ機構および資源割り当て機構の実現が必要になる．本研究では，知的モバイルエージェントを用いたアプリケーションの試作に焦点を当ててフレームワークを設計しており，匿名エージェントを適切に扱うことが可能なセキュリティ機構および資源割り当て機構の実現は今後の課題となっている．

携帯情報機器への適用 ソフトウェアシステムは，専門の技術者によるメインフレーム，ワークステーション上やパーソナルコンピュータ上での利用から，携帯電話，携帯情報端末や据え付け型情報機器等に組み込まれた形で，より一般の利用者によって活用されるようになってきている．この状況の変化に対応し，知的モバイルエージェントシステムをより広い範囲に適用するためには，組み込み型の情報機器への適用が必要となる．情報機器に組み込むためには，より少ない資源での動作が必要であり今後の改良が望まれる．情報機器が持つ限られた資源を有効活用するために，モバイルエージェント技術を活用すること自身も，興味深い研究課題である．

---

[2] この非効率性に関しては，プロセスマイグレーションの研究により実証済みである

[3] 例えば，JAVA では JIT コンパイラの性能向上およびセキュリティ上の理由から実行状態のキャプチャ機能を提供していない

謝辞

本論文の執筆では，多くの方々からの御支援と御協力をいただきました．ここでは，それらの方々への感謝の気持ちを述べます．
本研究を進めるにあたり，指導教官である名古屋工業大学 工学部 知能情報システム学科 新谷虎松教授には，様々な面から非常に多くのご指導をいただきました．心から感謝いたします．
本論文の審査員となることを快く承諾していただき，審査員を務めてくださいました 名古屋工業大学 知能情報システム学科 石井直宏 教授，同 北村 正 教授 に心から感謝いたします．
本研究を進めるにあたり，様々な面で議論し支えていただいた 名古屋工業大学 工学部 知能情報システム学科 大囿忠親助手 に感謝いたします．
名古屋工業大学 工学部 知能情報システム学科 新谷研究室所属のみなさま，そして卒業生のみなさまには，ゼミ等を通じて有益な議論や御指摘を多数頂きました．心から感謝いたします．特に，先輩であり本研究をはじめるきっかけを下さった，現北陸先端科学技術大学院大学 知識科学教育研究センター 伊藤孝行助教授には，研究面だけでなく私生活の面からも励まし支えていただきました．心から感謝いたします．副島大和君（現 松下通信工業所属），と水谷伸晃君には，システムの実装に関連して協力していただき，有益な議論と指摘をいただきました．心から感謝いたします．また，服部宏充君をはじめとした多くの方々に，本研究で開発したシステムを実際に研究で使っていただき，多数の有益なご指摘をいただきました．心から感謝いたします．
本研究での実験を進めるにあたり，名古屋工業大学 工学部 知能情報システム学科 での知能プログラミング演習受講生のみまさなには，実験に参加していただくとともに，有益な指摘を多数頂きました．心から感謝いたします．
友人および知人のみなさんには，私が様々な困難にぶつかったときに，何度も励まし支えていただきました．心から感謝いたします．
最後に，妹（福田愛子），祖父（福田久雄），祖母（福田まつ）および親戚のみなさまには，私が研究活動を行うことを認めていただき，日々の生活で様々な不便をかけてしまったにもかかわらず，辛抱強く支えていただきました．また，私が思い悩んだときに，何度も励ましの言葉を頂きました．心から感謝いたします．亡き母（福田育子，平成９年４月２２日永眠）と父（福田雅文，平成１２年４月２９日永眠）に，この論文をささげます．

関連文献

[1] Giovanni Adorni, Federico Bergenti, Agostino Poggi, and Giovanni Rimassa. Enabling fipa agents on small devices. In Matthias Klusch and Franco Zambonelli, editors, *Cooperative Information Agents V – Proceedings of 5th International Workshop on Cooperative Information Agents(CIA2001)*, pp. 248–257. Springer, 2001.

[2] 鯵坂恒夫, 佐伯元司. ソフトウェアテクノロジーシリーズ 7 – 方法論工学と開発環境. 共立出版, 2001.

[3] Amazon.com. Web page. http://www.amazon.com/.

[4] Apple. Web page. http://www.apple.com/.

[5] Amazon.com Auctions. Web page. http://auctions.amazon.com/.

[6] Yahoo! Auctions. Web page. http://auctions.yahoo.com/.

[7] AuctionWatch. Web page. http://www.auctionwatch.com/.

[8] Stig Saether Bakken, Alexander Aulbach, Egon Schmid, Jim Winsted, Lars Torben Wilson, Rasmus Lerdorf, Zeev Suraski, Andrei Zmievski, and Jouni Ahto. Php manual. (http://www.php.net/manual/en/).

[9] BargainFinder. Web page. http://bf.cstar.ac.com/bf/.

[10] J. Baumann, F. Hohl, K. Rothermel, and K. Strasser. Mole - concepts of a mobile agent system. *Journal of World Wide Web*, Vol. 1, No. 3, pp. 123–137, 1998.

[11] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Jade: a fipa2000 compliant agent development environment. In *Proc. of International Conference on Autonomous Agents 2001 (Agents2001)*, pp. 216–217, May 2001.

[12] S. Bergamaschi, G. Cabri, F. Guerra, L. Leonardi, M. Vincini, and F. Zambonelli. Supporting information integration with autonomous agents. In Matthias Klusch and Franco Zambonelli, editors, *Cooperative Information Agents V – Proceedings of 5th International Workshop on Cooperative Information Agents(CIA2001)*, pp. 88–99. Springer, 2001.

[13] Sonia Bergamaschi and Domenico Beneventano. Integration information from multiple sources of textual data. In Matthias Klusch, editor, *Intelligent Information Agents – Agent-Based Information Discovery and Management on the Internet*, pp. 53–77. Springer-Verlag, 1999.

[14] Federico Bergenti and Agostino Poggi. A development toolkit to realize autonomous and interoperable agents. In *Proc. of International Conference on Autonomous Agents 2001 (Agents2001)*, pp. 632–639, May 2001.

[15] Tim Berners-Lee, Mark Fischetti, and Michael L. Dertouzos. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor.* Harper Collins, 1999.

[16] L.F. Bic, M. Fukuda, and M.B. Dillencourt. Distributed computing using autonomous objects. *IEEE COMPUTER*, Vol. 29, No. 8, pp. 55–61, 1996.

[17] Harold Boley. The rule markup language: Rdf-xml data model, xml schema hierarchy, and xsl transformations. In *Proc. of the 14th International Conference on Applications of Prolog*, pp. 124–139, Oct. 2001.

[18] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol (soap) 1.1. Web page. http://www.w3.org/TR/SOAP/.

[19] Marco Bozzano, Giorgio Delzanno, Maurizio Martelli, Viviana Mascardi, and Floriano Zini. Logic programming and multi-agent systems: A synergic combination for applications and semantics. In Krazysztof R. Apt, Victor W. Marek, Mirek Truszczynski, and David S. Warren, editors, *The Logic Programming Paradigm – A 25-Year Perspective* –, pp. 5–32. Springer-Verlag, 1999.

[20] Brian Brewington, Robert Gray, Katsuhiro Moizumi, David Kotz, George Cybenko, and Daniela Rus. Mobile agents for distributed information retrieval. In Matthias Klusch, editor, *Intelligent Information Agents*, chapter 15, pp. 355–395. Springer-Verlag, 1999.

[21] Jean-Pierre Briot, Rachid Guerraoui, and Klaus-Peter Lohr. Concurrency and distribution in object-oriented programming. *ACM Computing Surveys*, Vol. 30, No. 3, pp. 291–329, Sep. 1998.

[22] Alan Bundy. *Artificial Intelligence Techniques – A Comprehensive Catalogue.* Springer Verlag, 1997.

[23] Paolo Busetta, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. Jack intelligent agents – components for intelligent agents in java. *AgentLink NewsLetter*, Vol. 2, pp. 2–5, Jan. 1999.

[24] Luca Cardelli. A language with distrubuted scope. In *Proc. of the 22nd Annual ACM Symposium on Principles of Programming Languages*, pp. 286–297, Jan. 1995.

[25] Daniel T. Chang and Stefan Covaci. The omg mobile agent facility: A submission. In Kurt Rothermel and Radu Popescu-Zeletin, editors, *LNAI 1219, Mobile Agents – 1st International Workshop, MA'97 Proceedings*, pp. 98–110. Springer-verlag, 1997.

[26] A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proceedings of First International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents (PAAM96)*, pp. 75–90, April 1996.

[27] W.F. Clocksin and C.S. Mellish. *Programming in Prolog.* Springer-Verlag, 1981.

[28] DealTime.com. Web page. http://www.dealtime.com/.

[29] M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification os dmars. In M. P. Singh, A. S. Rao, and M. Wooldridge, editors, *Intelligent Agents IV*, pp. 155–176. Springer-Verlag, 1998.

[30] Yahoo! Auctions Directory. Web page. http://dir.yahoo.com/Business_and_Economy/Companies/Auctions/Online_Auctions/.

[31] R. B. Doorenbos, O. Etzioni, and D. S. Weld. A scalable comparison-shopping agent for the world-wide web. In *Proc. of Autonomous Agents 97*, pp. 39–48, 1997.

[32] eBay. Web page. http://www.ebay.com.

[33] O. Etzioni. Moving up the information food chain: Deploying softbots on the world wide web. *AI magazine*, Vol. 18, No. 2, pp. 11–18, Summer 1997.

[34] T. Finin, R. Fritzson, D. McKay, and R. McEntire. Kqml as an agent communication language. In *Proc. of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pp. 456–463, Gaithersburg, Maryland, 1994. ACM Press.

[35] FIPA. Fipa 98 specification part 11 – agent management support for mobility –. Web page, 1998. http://www.fipa.org/specs/fipa00005/OC00005A.html.

[36] Alfonso Fuggetta, Gian Pietro Picco, and Giovanni Vigna. Understanding code mobility. *IEEE Transactions on Software Engneering*, Vol. 24, No. 5, pp. 342–361, 1998.

[37] S. Fujita, K.Koyama, T.Yamanouchi, S.Jagannathan, R. Kelsey, and J. Philbin. Mobile and distributed agents in mobidget. In *Proc. of the 1st International Symposium on Agent Systems and Applications 3rd International Symposium on Mobile Agents*, 1999.

[38] Fujitsu. Pathwalker. Web page. http://www.labs.fujitsu.com/free/paw/index.html.

[39] N. Fukuta, T. Ito, T. Ozono, and T. Shintani. A framework for cooperative mobile agents and its case-study on BiddingBot. IOS Press, 2002. (to appear.).

[40] N. Fukuta, T. Ito, and T. Shintani. On implementing cooperative e-commerce support agents using mobile intelligent agent framework milog. In *Proc. of the Second International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distrigbuted Computing (SNPD'01)*, pp. 928–935, Aug 2001.

[41] N. Fukuta, N. Mizutani, T. Ozono, and T. Shintani. iML: A logic-based framework for constructing graphical user interfaces on mobile agents. In Oskar Bartenstein, Ulrich Geske, Markus Hannebauer, and Osamu Yoshie, editors, *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2002. (to appear.).

[42] Naoki Fukuta, Takayuki Ito, Tadachika Ozono, and Toramatsu Shintani. A framework for cooperative mobile agents and its case-study on BiddingBot. In *Proc. of the JSAI 2001 International Workshop on Agent-based Approaches in Economic and Social Complex Systems(AESCS2001)*, pp. 91–98, May 2001.

[43] 福田直樹, 伊藤孝行, 大囿忠親, 新谷虎松. モバイルエージェント記述言語 MiLog における anytime migration の実現. 第 6 2 回情報処理学会全国大会論文集, pp. 19–20, Mar. 2001.

[44] 福田直樹, 伊藤孝行, 新谷虎松. Weblog: WWW における情報収集エージェントのための論理型記述言語の実現. 第 8 回マルチエージェントと強調ワークショップ (MACC'99), Dec. 1999.

[45] Naoki Fukuta, Takayuki Ito, and Toramatsu Shintani. MiLog: A mobile agent framework for implementing intelligent information agents with logic programming. In *Proc. of the First Pacific Rim International Workshop on Intelligent Information Agents (PRIIA2000)*, pp. 113–123, August 2000.

[46] Naoki Fukuta, Takayuki Ito, and Toramatsu Shintani. An approach to building mobile intelligent agents based on anytime migration. In R. Kowalczyk, S. W. Loke, N. E. Reed, and G. Williams, editors, *Lecture Notes in Artificial Intelligence*, pp. 219–228. Springer-Verlag, 2001.

[47] Naoki Fukuta, Takayuki Ito, and Toramatsu Shintani. A logic-based framework for mobile intelligent information agents. In *Poster Proc. of the Tenth International World Wide Web Conference(WWW10)*, pp. 58–59, May 2001.

[48] 福田直樹, 伊藤孝行, 新谷虎松. 知的モバイルエージェントによる web ページ構築フレームワーク MiPage の実装. 人工知能学会第 5 3 回知識ベースシステム研究会資料 SIG-KBS-A102, pp. 31–36, Sep. 2001.

[49] Naoki Fukuta, Nobuaki Mizutani, Tadachika Ozono, and Toramatsu Shintani. iML: A logic-based framework for constructing graphical user interface on mobile agents. In *Proc. of the 11th International Conference on Applications of Prolog(INAP2001)*, pp. 152–159, October 2001.

[50] 福田直樹, 新谷虎松. 論理型言語 Weblog に基づく情報探索支援について. 平成10年度 電気関係学会東海支部連合大会講演論文集, p. 280, Sep. 1998.

[51] 福田直樹, 新谷虎松. Weblog: WWW における情報検索エージェント記述言語の実現. 第5 8回情報処理学会全国大会講演論文集 (3), pp. 27–28, Mar. 1999.

[52] 福田直樹, 新谷虎松. エージェント記述言語 Weblog におけるアプリケーション開発支援機構の試作. 第1 3回人工知能学会全国大会論文集, pp. 517–520, June 1999.

[53] 福田直樹, 新谷虎松. モバイルエージェントフレームワーク MiLog におけるトレイルスタックの最適化手法について. 2000年電子情報通信学会情報・システムソサイエティ大会講演論文集, p. 266, Sep. 2000.

[54] 福田直樹, 新谷虎松. 情報エージェント記述言語 Weblog におけるモビリティの実現. 第6 0回情報処理学会全国大会論文集 (3), pp. 79–80, Mar. 2000.

[55] 古川康一, 溝口文雄. (知識情報処理シリーズ6) 並列論理型言語GHCとその応用. 共立出版, 1987.

[56] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proc. of the 6th National Conference on Artificial Intelligence (AAAI-87)*, pp. 677–682, 1987.

[57] M. Georgeff and A. Rao. Rational software agents: From theory to practice. In N. R. Jennings and M. J. Wooldridge, editors, *Agent Technology –Foundations, Applications, and Markets–*, pp. 139–160. Springer-Verlag, 1998.

[58] John R. Graham and Keith S. Decker. Towards a distributed, environment-centered agent framework. In Nicholas R. Jennings and Yves Lespérance, editors, *Intelligent AgentsIV – Proceedings of the International Workshop on Agent Theories, Architectures, and Language (ATAL99)* –, pp. 290–304. Springer, 2000.

[59] Robert S. Gray, David Kotz, George Cybenko, and Daniela Rus. D'Agents: Security in a multiple-language, mobile-agent system. In Giovanni Vigna, editor, *Mobile Agents and Security*, Vol. 1419 of *Lecture Notes in Computer Science*, pp. 154–187. Springer-Verlag, 1998.

[60] R. H. Guttman and P. Maes. Agent-mediated integrative negotiation for retail electronic commerce. In *Proceedings of the Second International Workshop on Cooperative Information Agents (CIA'98)*, 1998.

[61] R. H. Guttman, A. G. Moukas, and P. Maes. Agent-mediated electronic commerce: A survey. *The Knowledge Engineering Review*, Vol. 13, No. 2, pp. 147–159, 1998.

[62] H. Hattori, T. Ito, T. Ozono, and T. Shintani. An approach to coalition formation using argumentation-based negotiation in multi-agent systems. In *Proc. of the 14th International Conference on Industrial &Engineering Applications of Artificial Intelligence & Expert Systems(IEA/AIE-01)*, pp. 687–696, June 2001.

[63] J. Hendler and D. L. McGuinness. Darpa agent markup language. *IEEE Intelligent Systems*, Vol. 15, No. 6, pp. 72–73, 2000.

[64] 平川正人, 安村通晃. (bit別冊) ビジュアルインタフェース–ポストGUIを目指して–. 共立出版, 1996.

[65] Satoshi Hirano. Horb users guide. Web page. http://horb.etl.go.jp/horb/doc/guide/guide.htm.

[66] H.Peine and T. Stolpmann. The architecture of the ara platform for mobile agents. In D. Milojicic, F. Douglis, and R. Wheeler, editors, *Mobility: Processes, Computers, and Agents*, pp. 474–483. Addison-Wesley and the ACM Press, 1999.

[67] Marcus J. Huber. Jam: A bdi-theoretic mobile agent architecture. In *Proc. of International Conference on Autonomous Agents 1999 (Agents99)*, pp. 236–243, May 1999.

[68] IBM. Industry consortium forms global computer, electronics and telecommunications b2b e-marketplace: e2open.com, 2000. IBM Press Release (Available at http://www.ibm.com/Press/prnews.nsf/ jan / 76CE157B41D8F9A9852568F6005B3A5C).

[69] Takayuki Ito, Naoki Fukuta, Toramatsu Shintani, and Katia Sycara. Biddingbot: A multiagent support system for cooperative bidding in multiple auctions. In *Proc. of 4th International Conference on Multi Agent Systems (ICMAS'2000)*, pp. 435–436, July 2000.

[70] Takayuki Ito, Naoki Fukuta, Ryota Yamada, Toramatsu Shintani, and Katia Sycara. Cooperative bidding mechanisms among agents in multiple online auctions. In *Proc. of the Sixth pacific-Rim International Conference on Artificial Intelligence (PRICAI'2000)*, p. 810, Aug 2000.

[71] Takayuki Ito and Toramatsu Shintani. Persuasion among agents : An approach to implementing a group decision support system based on multi-agent negotiation. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 592–597. Morgan Kaufmann, Aug. 1997.

[72] iTrack. Web page. http://www.itrack.com/.

[73] N. R. Jennings and M. J. Wooldridge. Applications of intelligent agents. In N. R. Jennings and M. J. Wooldridge, editors, *Agent Technology –Foundations, Applications, and Markets–*, pp. 3–28. Springer-Verlag, 1998.

[74] Heecheol Jeon, Charles J. Petrie, and Mark R. Cutkosky. JATLite: A java agent infrastructure with message routing. *IEEE Internet Computing*, Vol. 4, No. 2, pp. 87–96, 2000.

[75] D. Johansen, R. v. Renesse, and F.B. Schneider. Operating system support for mobile agents. In *Proc. of the 5th IEEE Workshop on Hot Topics in Operating Systems*, 1995.

[76] 河込和宏, 中村秀男, 大野邦夫, 飯島正. ソフトウェアテクノロジーシリーズ 2–分散オブジェクトコンピューティング. 共立出版, 1999.

[77] 川村隆浩, 田原康之, 長谷川哲夫, 大須賀昭彦, 本位田真一. Bee-gent: 移動型仲介エージェントによる既存システムの柔軟な活用を目的としたマルチエージェントフレームワーク. 電子情報通信学会論文誌 (D-I), Vol. J82-D-I, No. 9, pp. 1165–1180, 1999.

[78] T. Kawamura, N. Yoshioka, T. Hasegawa, A. Ohsuga, and S. Honiden. Bee-gent : Bonding and encapsulation enhancement agent framework for development of distributed systems. In *Proc. of the 6th Asia-Pacific Software Engineering Conference*, 1999.

[79] Matthias Klusch. *Intelligent Information Agents – Agent-Based Information Discovery and Management on the Internet.* Springer-Verlag, 1999.

[80] C. A. Knoblock, S. Minton, J. L. Ambite, N. Ashish, P. J. Modi, I. Muslea, A. G. Philpot, and S. Tejada. Modeling web sources for information integration. In *211–218*, pp. Proc. of AAAI–98, 1998.

[81] 小西秀文, 吉田政臣. エージェントシステム「jumon」の概要とアプリケーション構築例. *Interface*, pp. 150–156, Feb. 2001.

[82] David Kotz and Robert S. Gray. Mobile agents and the future of the internet. *ACM Operating Systems Review*, pp. 7–13, August 1999.

[83] Andreas Krall. The vienna abstract machine. *The Journal of Logic Programming*, Vol. 29, No. 1–3, pp. 85–106, 1996.

[84] D. Lange and M. Oshima. *Programming and deploying Java mobile agents with Aglets.* Addison-Wesley, 1998.

[85] D. Lange and M. Oshima. Seven good reasons for mobile agents. *Comm. of ACM*, Vol. 42, No. 3, pp. 88–89, March 1999.

[86] Henry Lieberman. Letizia: An agent that assists web browsing. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI'95)*, Aug. 1995.

[87] Mobile Agents List. Web page. http://inf.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/mal/preview/preview.html.

[88] Seng Wai Loke and Andrew Davidson. Logicweb: Enhancing the web with logic programming. *The Journal of Logic Programming*, Vol. 36, pp. 195–240, 1998.

[89] Microsoft. Active server pages. (http://msdn.microsoft.com/asp/).

[90] Dejan S. Milojíčič, Fred Douglis, Yves Paindaveine, Richard Whieeler, and Songnian Zhou. Process migration. *ACM Computing Surveys*, Vol. 32, No. 3, pp. 241–299, Sep. 2000.

[91] 溝口文雄, 古川康一, J-L. Lassez. (知識情報処理シリーズ 別巻2) 制約論理プログラミング. 共立出版, 1989.

[92] 水谷伸晃, 福田直樹, 新谷虎松. 言語非依存型アプリケーション開発支援環境の試作. 2000年電子情報通信学会ソサイエティ大会（情報，システム）, p. 31, Sep. 2000.

[93] Mobeet. Web page. http://mobeet.ex.nii.ac.jp/.

[94] Jörg P. Müller. *The Design of Intelligent Agents.* Springer-Verlag, 1996.

[95] H. Nakashima, I. Noda, and K. Honda. Organic programming language gaea for multi-agents. In *Proc. of 1st International Conference on Multi Agent Systems(ICMAS96)*, pp. 236–243, Dec. 1996.

[96] Hyacinth S. Nwana, Divine T. Ndumu, Lyndon C. Lee, and Jaron C. Collis. Zeus: a toolkit and approach for building distributed multi-agent systems. In *Proc. of International Conference on Autonomous Agents 1999 (Agents99)*, pp. 360–361, May 1999.

[97] ObjectSpace. Voyager. Web page, 1995. http://www.objectspace.com/products/voyager/.

[98] A. Ohsuga, Y. Nagai., Y. Irie, M. Hattori, and S. Honiden. Plangent: An approach to making mobile agents intelligent. *IEEE Internet Computing*, Vol. 1, No. 4, pp. 50–57, July/Aug. 1997.

[99] OMG. Corba 2.5 specification. Web page. ftp://ftp.omg.org/pub/docs/formal/01-09-01.pdf.

[100] Onsale. Web page. http://www.onsale.com/.

[101] 大囿忠親. マルチエージェントシステム構築環境に関する研究. 名古屋工業大学 博士論文, 2000.

[102] Gian Pietro Picco. μ code: A lightweight and flexible mobile code toolkit. In *Proc. of the 2nd International Workshop on Mobile Agents(MA98)*, pp. 160–171, Sep. 1998.

[103] KLS Research. K-prolog compiler version 5.0 online manual. Web page, 2000. http://www.kprolog.com/doc/ja/index.html.

[104] J. A. Rodriguez, P. Noriega, C. Sierra, and J. Padget. Fm96.5:a java-based electronic auction house. In *Proceedings of Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM97)*, 1997.

[105] Peter Van Roy. 1983-1993: The wonder years of sequential prolog implementation. *Journal of Logic Programming*, Vol. 19/20, p. The Journal of Logic Programming, May/July 1994.

[106] Takahiro Sakamoto, Tatsurou Sekiguchi, and Akinori Yonezawa. Bytecode transformation for portable thread migration in java. In *Proc. of 2nd International Symposium on Agent Systems and Applications and 4th International Symposium on Mobile Agents(ASA/MA2000)*, pp. 16–28, 2000.

[107] T. Sandholm. emediator: A next generation electronic commerce server. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence(AAAI-99)*, pp. 923–924, 1999.

[108] 佐藤一郎. モバイルエージェントの動向. 人工知能学会誌, Vol. 14, No. 4, pp. 598–605, 1999.

[109] Ichiro Sato. Mobilespaces: A framework for building adaptive distrubuted applications using a hierarchical mobile agent system. In *Proc. of IEEE International Conference on Distriburted Computing Systems(ICDCS2000)*, pp. 161–168, 2000.

[110] Douglas Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-oriented Software Architecture, Volume2 – Patterns for Concurrent and Networked Objects –*. Wiley, 2000.

[111] D. Schoder and T. Eymann. The real challenges of mobile agents. *Comm. of ACM*, Vol. 43, No. 6, pp. 111–112, June 2000.

[112] Kennard Scribner and Mark C. Stiver. (訳:スリー・エー・システムズ) SOAP 技術入門. Pearson Education Japan, 2001.

[113] E. Y. Shapiro. 知識の帰納的推論. 共立出版, 1986. (訳:有川 節夫).

[114] T. Shintani, T. Ito, and K.Sycara. Multiple negotiations among agents for a distributed meeting scheduler. In *Proc. of the 4th International Conference on Multi-Agent Systems (ICMAS-2000)*, pp. 435–436, 2000.

[115] Toramatsu Shintani and Takayuki Ito. Cooperative meeting scheduling among agents base on multiple negotiations. In *Proc. of 6th International Conference on Cooperative Information SYstems(CoopIS2001)*, Sep. 2001.

[116] 新谷虎松, 大囿忠親, 福田直樹. モバイルエージェントの応用–マルチエージェントシステムのためのモビリティの利用–. 人工知能学会誌, Vol. 16, No. 4, pp. 488–493, July 2001.

[117] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, Vol. 60, pp. 51–92, 1993.

[118] K. Shudo and Y. Muraoka. Noncooperative migration of execution context in java virtual machines. In *First Annual Workshop on Java for High-Performance Computing*, pp. 49–57, June 1999.

[119] Leon Sterling and Ehud Shapiro. *The Art of Prolog (second edition)*. The MIT Press, 1994.

[120] Sun. Java server pages quickstart guide, 2001. (http://java.sun.com/products/jsp/docs.html).

[121] N. Suri, J. M. Bradshaw, M. R. Breedy, P. T. Groth, G. A. Hill, R. Jeffers, T. S. Mitrovich, B. R. Pouliot, and D. S. Smith. Nomads: Toward a strong and safe mobile agent system. In *International Conference on Autonomous Agents(AGENTS 2000)*, pp. 163–164, June 2000.

[122] K. Sycara, K. Decker, A Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert/Intelligent Systems & Their Applications*, Vol. 11, No. 6, pp. 36–46, Dec. 1996.

[123] P. Tarau. Jinni: Intelligent mobile agent programming at the intersection of java and prolog. In *Proc. of The Fourth International Conference on The Practical Application of Intelligent Agents and Multi-Agents(PAMM'99)*, pp. 109–123, 1999.

[124] Paul Tarau, Koen De Bosschere, and Bart Demoen. Partial translation: Towards a portable and efficient prolog implementation technology. *The Journal of Logic Programming*, Vol. 29, No. 1–3, pp. 65–83, Nov. 1996.

[125] IntelliOne Technologies. Agentbuilder pro 1.3r0 user guide. Web page. available at http://www.agentbuilder.com/.

[126] E. Truyen, B. Vanhaute, T. Connix, W. Joosen, and P. Verbaeten. Portable support for transparent thread migration in java. In *Proc. of Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents(ASA/MA 2000)*, pp. 29–43, Sep. 2000.

[127] Tryllian. Gossip. Web page, 1999. http://www.tryllian.com.

[128] T. Tsuruta and T. Shintani. Scheduling meetings using distributed valued constraint satisfaction algorithm. In *Proc. of the 14th European Conference on Artificial Intelligence (ECAI-2000)*, pp. 383–387, 2000.

[129] Kazunori Ueda. Concurrent logic/constraint programming: The next 10 years. In Krazysztof R. Apt, Victor W. Marek, Mirek Truszczynski, and David S. Warren, editors, *The Logic Programming Paradigm - A 25-Year Perspective* –, pp. 53–71. Springer-Verlag, 1999.

[130] D.H.D. Warren. An abstract prolog instruction set. Technical Report 309, SRI International, 1983.

[131] J. E. White. Mobile agents make a network an open platform for third-party developers. *IEEE Computer*, Vol. 27, No. 11, pp. 89–90, November 1994.

[132] J. E. White. Mobile agents. In J. M. Bradshaw, editor, *Software Agents*, pp. 437–472. MIT Press, 1997.

[133] Christian Wicke, Lubomir F. Bic, Michael B. Dillencourt, and Munehiro Fukuda. Automatic state capture of self-migrating computations in messengers. In *Second International Workshop on Mobile Agents(MA98)*, pp. 68–79, Sep. 1998.

[134] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, and B. Peet. Concordia: An infrastracture for collaborating mobile agents. In *Proc. of the First International Workshop on Mobile Agents(MA'97)*, pp. 86–97, Apr. 1997.

[135] P. R. Wurman, M. P. Wellman, , and W. E. Walsh. The michigan internet auctionbot: A configurable auction server for human and software agents. In *Proceedings of the Second International Conference on Autonomous Agents (Agents-98)*, 1998.

[136] J. Yamamoto and K. Sycara. A stable and efficient buyer coalition formation scheme for e-marketplaces. In *Proc. of the 5th International Conference on Autonomous Agents (Agents'2001)*, pp. 576–583, 2001.

原著となった発表論文一覧

論文誌

1. **N. Fukuta**, N. Mizutani, T. Ozono, and T. Shintani : "iML: A Logic-based Framework for Constructing Graphical User Interfaces on Mobile Agents", In O. Bartenstein et, al. (Eds.) Lecture Notes in Artificial Intelligence, (採録決定済).

2. **N. Fukuta**, T. Ito, and T. Shintani : "An Approach to Building Mobile Intelligent Agents Based on Anytime Migration", In R. Kowalczyk, S.W. Loke, N.E. Reed, and G. Williams (Eds.) Lecture Notes in Artificial Intelligence, Vol.2112, pp.219–228, Oct. 2001.

3. **N. Fukuta**, T. Ito, T. Ozono, and T. Shintani,"A Framework for Cooperative Mobile Agents and Its Case-Study on BiddingBot", IOS Press, (採録決定済).

4. 福田直樹，大囿忠親，新谷虎松: "知的モバイルエージェントによるWebページ構築フレームワークMiPageの実装", 人工知能学会誌, (投稿中).

5. 新谷虎松，大囿忠親，福田直樹: "モバイルエージェントの応用 – マルチエージェントシステムのためのモビリティの利用", 人工知能学会誌, Vol.16, No.4, pp.488–493, 2001.

International Conferences and Workshops

1. **N. Fukuta**, N. Mizutani, T. Ozono, and T. Shintani,"iML: A Logic-based Framework for Constructing Graphical User Interface on Mobile Agents",Proc. of the 11th International Conference on Applications of Prolog(INAP2001), pp.152–159, Tokyo Japan, Oct. 2001.

2. **N. Fukuta**, T. Ito, and T. Shintani,"On Implementing Cooperative E-commerce Support Agents Using Mobile Intelligent Agent Framework MiLog",Proc. of the Second International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing(SNPD'01),pp.928–935, Nagoya Japan, Aug. 2001.

3. **N. Fukuta**, T. Ito, T. Ozono, and T. Shintani,"A Framework for Cooperative Mobile Agents and Its Case-Study on BiddingBot",The JSAI 2001 International Workshop on Agent-based Approaches in Economic and Social Complex Systems (AESCS 2001),pp.91–98, Matsue Japan, May 2001.

4. **N. Fukuta**, T. Ito, and T. Shintani,"A Logic-based Framework for Mobile Intelligent Information Agents", Poster Proc. of the Tenth International World Wide Web Conference(WWW10),pp.58–59, Hong Kong, May 2001.

5. **N. Fukuta**, T. Ito, and T. Shintani,"MiLog: A Mobile Agent Framework for Implementing Intelligent Information Agents with Logic Programming",Proc. of the First Pacific Rim International Workshop on Intelligent Information Agents (PRIIA2000), pp.113–123, Melbourne Australia, Aug. 2000.

6. T. Ito, **N. Fukuta**, R. Yamada, T. Shintani, and K. Sycara,"Cooperative Bidding Mechanisms among Agents in Multiple Online Auctions",Proc. of the Sixth Pacific-Rim Internatilnal Conference on Artificial Intelligence (PRICAI'2000),pp.810, Melbourne Australia, Aug. 2000.

7. T. Ito, **N. Fukuta**, T. Shintani, and K. Sycara,"BiddingBot: A Multiagent Support System for Cooperative Bidding in Multiple Auctions",Proc. of the Fourth International Conference on Multi Agent Systems(ICMAS'2000),pp.399–400,2000.

研究会（国内）およびワークショップ（国内）

1. **福田直樹**, 伊藤孝行, 新谷虎松 : "知的モバイルエージェントによるWebページ構築フレームワーク MiPageの実装", 人工知能学会 第53回知識ベースシステム研究会資料 SIG-KBS-A102, pp.31–36, 名古屋, 2001.9 N. Fukuta, T. Ito, and T. Shintani: " On Implementing an Intelligent Mobile Agent-based Web Page Constructing Framework MiPage", Proc. of the 53th Special Interest Group on Knowledge-based Systems(SIG-KBS), Japanese Society of Artificial Intelligence(JSAI), pp.31–36, Nagoya Japan, Sep. 2001.

2. **福田直樹**, 伊藤孝行, 新谷虎松: "Weblog: WWWにおける情報収集 エージェントのための論理型記述言語の実現", 第8回マルチエージェントと協調計算ワークショップ（MACC'99）, 1999. N. Fukuta, T. Ito, and T. Shintani: "Weblog: A Logic-based Information Gathering Agent Description Language for WWW", Proc. of the Workshop on Multi-Agent and Cooperative Computation(MACC'99), Kyoto, Dec. 1999.

3. 伊藤孝行, **福田直樹**, 新谷虎松: "マルチエージェント入札支援 システムBiddingBotにおけるエージェント間の協調的入札機構について", 第8回マルチエージェントと協調計算ワークショップ（MACC'99）, 京都, 1999.12

修士論文

1. 福田 直樹: “WWWにおける情報検索エージェント記述言語Weblogの実装に関する研究”, 名古屋工業大学, 1999.

卒業論文

1. 福田 直樹: “リフレクティブな制約論理型言語RXFにおけるマルチエージェントシステム開発支援機構の試作”, 名古屋工業大学, 1997.

全国大会および支部大会での報文

1. **福田直樹**, 伊藤孝行, 大囿忠親, 新谷虎松: ”モバイルエージェント記述言語MiLogにおけるAnytime Migrationの実現”, 第62回情報処理学会全国大会論文集, 情報処理学会, pp.19-20, (2001.3).
2. **福田直樹**, 新谷虎松: ”モバイルエージェントフレームワークMiLogにおける トレイルスタックの最適化手法について ”, 2000年電子情報通信学会情報・システムソサイエティ大会講演論文集, pp.266, 電子情報通信学会, (2000.9).
3. **福田直樹**, 新谷虎松: ”情報エージェント記述言語Weblogにおけるモビリティの実現”, 第60回情報処理学会全国大会論文集 (3), 情報処理学会, pp.79-80, (2000.3).
4. **福田直樹**, 新谷虎松: ”エージェント記述言語Weblogにおけるアプリケーション開発支援機構の 試作”, 第13回人工知能学会全国大会論文集, 人工知能学会, pp. 517-520, (1999.6).
5. **福田直樹**, 新谷虎松: ”Weblog: WWWにおける情報検索エージェント記述言語の実現”, 第58回情報処理学会全国大会論文集 (3), 情報処理学会, pp.27-28, (1999.3).
6. **福田直樹**, 新谷虎松：”論理型言語Weblogに基づく情報探索支援について”, 平成10年度 電気関係学会東海支部連合大会講演論文集, pp.280, (1998.9).
7. **福田直樹**, 藤田隆久, 新谷虎松: ”URL履歴に基づくブラウジングモデルの学習について”, 第56回情報処理学会全国大会論文集 (3), 情報処理学会, pp.177-178, (1998.3).
8. **福田直樹**, 新谷虎松：”論理型言語RXFにおける階層的BDIアーキテクチャの設計”, 平成9年度 電気関係学会東海支部連合大会講演論文集, pp.286, (1997.9).
9. **福田直樹**, 新谷虎松: ”リフレクティブな論理型言語RXFにおけるマルチエージェントシステム開発支援機構の試作について”, 第54回情報処理学会全国大会論文集, 情報処理学会, pp.347-348, (1997.3).
10. **福田直樹**, 新谷虎松: ”論理型言語RXFにおける並行プログラミング支援機構の試作”, 第10回人工知能学会全国大会論文集, 人工知能学会, pp.29-32, (1996.6).

その他の共著報文

1. 伊藤孝行, 服部宏充, 福田直樹, 山田亮太, 新谷虎松: "マルチエージェント協調的入札に基づく複数オークション支援システム BiddingBot の実装", 第 62 回情報処理学会全国大会論文集 (2), 情報処理学会, pp.53-54, (2001.3).

2. 小島修一, 福田直樹, 大囿忠親, 新谷虎松: "モバイルエージェントに基づく研究室情報管理システムについて", 第 62 回情報処理学会全国大会論文集 (2), 情報処理学会, pp.33-34, (2001.3).

3. 後藤将志, 福田直樹, 新谷虎松：" 未知の情報源統合における情報源発見手法について ", 平成 12 年度 電気関係学会東海支部連合大会 講演論文集, pp.301, (2000.9).

4. 林　真暢, 福田直樹, 新谷虎松：" トピック数の単複識別に基づく Web ページの段階的分類によるユーザの興味の推定 ", 平成 12 年度 電気関係学会東海支部連合大会 講演論文集, pp.300, (2000.9).

5. 山田亮太, 伊藤孝行, 福田直樹, 新谷虎松：" 木構造化した HTML 文書の比較に基づく wrapper の生成手法について ", 平成 12 年度 電気関係学会東海支部連合大会 講演論文集, pp.299, (2000.9).

6. 水谷伸晃, 福田直樹, 新谷虎松：" モバイルエージェント記述言語 MiLog における GUI 構築支援環境の試作 ", 平成 12 年度 電気関係学会東海支部連合大会 講演論文集, pp.294, (2000.9).

7. 副島大和, 福田直樹, 大囿忠親, 伊藤孝行, 新谷虎松: " 友好関係を利用したエージェント間のプライベート情報共有手法について ",2000 年電子情報通信学会情報・システムソサイエティ大会講演論文集, 電子情報通信学会,pp.82,(2000.9).

8. 林真暢, 福田直樹, 新谷虎松: " トピック数に基づく Web ページの段階的分類を用いたユーザの興味の推定 ",2000 年電子情報通信学会情報・システムソサイエティ大会講演論文集, 電子情報通信学会,pp.43,(2000.9).

9. 山田亮太, 伊藤孝行, 福田直樹, 新谷虎松: " WWW における複数インスタンス情報源のための wrapper 生成機構について ",2000 年電子情報通信学会情報・システムソサイエティ大会講演論文集, 電子情報通信学会,pp.91,(2000.9).

10. 中野昌弘, 福田直樹, 新谷虎松: " オークション支援システムにおける概念データベースを用いた財のキーワード抽出 ",2000 年電子情報通信学会情報・システムソサイエティ大会講演論文集, 電子情報通信学会, pp.93, (2000.9).

11. 水谷伸晃, 福田直樹, 新谷虎松: " 言語非依存型アプリケーション開発支援環境の試作 ",2000 年電子情報通信学会情報・システムソサイエティ大会講演論文集, 電子情報通信学会, pp.31, (2000.9).

12. 副島大和, **福田直樹**, 大囿忠親, 伊藤孝行, 新谷虎松: "FriendRing: 信頼度関係に基づくモバイルエージェント間の協調手法について", 第 14 回人工知能学会全国大会論文集, 人工知能学会,pp.217–220,(2000.7).

13. 山田亮太, 伊藤孝行, **福田直樹**, 新谷虎松: "複数オークション入札支援システム BiddingBot における情報収集エージェントの wrapper 生成機構について", 第 14 回人工知能学会全国大会論文集, 人工知能学会,pp.557–560,(2000.7).

14. 伊藤孝行, **福田直樹**, 山田亮太, 新谷虎松: "複数オンラインオークションにおけるマルチエージェントの協調的入札戦略について", 第 60 回情報処理学会全国大会論文集 (2), 情報処理学会, pp.19-20, (2000.3).

15. 後藤将志, **福田直樹**, 新谷虎松: "オントロジーに基づく電子メールからのスケジュール情報の抽出", 第 60 回情報処理学会全国大会論文集 (2), 情報処理学会, pp.115-116, (2000.3).

16. 林　真暢, **福田直樹**, 新谷虎松: "Web ページのクラスタリングに基づくユーザの興味の推定", 第 60 回情報処理学会全国大会論文集 (3), 情報処理学会, pp.3-4, (2000.3).

17. 山田亮太, 伊藤孝行, **福田直樹**, 新谷虎松: "オンラインオークション支援システム BiddingBot における入札額決定支援機構について", 第 60 回情報処理学会全国大会論文集 (2), 情報処理学会, pp.21-22, (2000.3).