成蹊大学理工学研究報告
J. Fac. Sci.Tech., Seikei Univ.
Vol.51 No.1 (2014)pp.15-20

AgentSphereにおける各種機能エージェントを利用したアプリケーションエージェントの構築手法

大久保　秀[*1]，甲斐　宗徳[*2]

A New Constructing Method for Application Agents Using Various Functional Agents for AgentSphere

Shu OKUBO[*1], Munenori KAI[*2]

ABSTRACT：AgentSphere is a platform for an autonomous parallel distributed processing system based on strong migration mobile agent system. In general, it is very difficult to manage load balance and effectiveness of total processing performance on parallel distributed processing system. Our approach is that mobile agents help us to perform effective parallel distributed processing by agents' autonomy. In our conventional AgentSphere, one has to describe all of agent's code necessary to functions of it by oneself. In this paper, we propose a mechanism for creating a new agent by using various functional agents, which have been prepared on the AgentSphere network a priori. According to this mechanism, when a user describe an agent's source code, one can write it easily and effectively without writing all of its code by means of selecting existing functional agents and delegating them to process the parts of functions.

Keywords：parallel distributed processing system, mobile agent system, application programming interface

(Received March 31, 2014)

1．はじめに

1．1　研究背景

近年，金融，公共，流通など幅広い分野で分散処理システムの普及，応用が進んでいる。また，一般家庭や会社においても，複数のコンピュータからなるホームネットワークや社内ネットワークが存在し，そのネットワーク内のコンピュータ群を利用することによって分散処理を行うことができる環境がある。分散処理により，一つのシステムとしての処理を複数の小さな処理の集まりとして捉え，それらの処理を複数のマシンで分担して行なう。これによる利点として，構成しているノードに障害が発生しても処理を続けられるといった耐故障性，構成するノードを柔軟に変更できるスケーラビリティ，複数マシンで処理を行うことによる処理速度の向上などを見込むことができる。

しかし，これらの利点を十分に発揮させられるように分散処理システムを構築するためには，分散処理に関する高度な知識と技術が必要となる。これが分散処理システムの導入を困難にさせている。そこで我々は，ネットワークのリアルタイム情報を用いたプログラム制御をユーザ自身が行わなくても分散処理の恩恵を受けられるようにするため，自律型の分散処理システムのプラットフォーム「AgentSphere」を開発してきた。

1．2　自律型モバイルエージェントシステムAgentSphere

我々は分散処理を実現するためのプラットフォームとして，自律的に振る舞うことが可能なモバイルエージェントシステムを採用した。モバイルとは「移動可能な」という意味をもち，エージェントとは「プログラム」そのものを意味する。つまり，モバイルエージェントシステムとは，ネットワークで繋がれた複数のマシンを，プログラム自身が実行中に移動し，移動した先のマシンで処理を継続することができるシステムである。更に，このモバイルエージェントが，自身が実行中のマシンにか

[*1]：理工学研究科理工学専攻博士前期課程学生
[*2]：理工学研究科理工学専攻教授 (kai@st.seikei.ac.jp)

かっている負荷状況を読み取り，負荷が高ければ負荷の低い他のマシンへ移動するということを自律的に行うことによって，負荷分散を実現している。

また，AgentSphereはP2P方式を採用している。これにより，分散処理システムのためのサーバを用意する必要がなく，一般的なホームネットワークや社内ネットワークで繋がれた個々のマシンを用いて分散処理を実現させることができる。

更に，AgentSphereはエージェントの移動性に強マイグレーションをサポートしている。これにより，エージェントは自身の実行状態を保持したままマシン間の移動を行い，移動先で中断した箇所から処理を再開することができる。

1．3　AgentSphereの現状

AgentSphereは自律分散を実現するための様々なモジュールで構成されている。従来までの研究で，エージェントが移動するためのネットワーク，エージェント同士が通信するためのメッセージング機構，エージェントの起動や復帰を行うための未知クラスの読み込みも行える階層型クラスローダ，高信頼性を保証するバックアップ機構，エージェントの「移動」に関する自律性をサポートするスケジューラ，AgentSphere内のオブジェクトへの動的なアクセスを可能にするシェル，そして，AgentSphere間で情報共有するための機構である分散ハッシュテーブル「Mercury」が実装された[1]。

1．4　目標

1.3 に示したように，AgentSphereにはすでに自律分散に必要な最低限の機能が実装されている。そこで本研究では，AgentSphereを利用するユーザのエージェント作成効率の向上を目的として，AgentSphereネットワーク上に，他のエージェントから利用可能な処理モジュールを存在させておくための機構を構築した。これによって，エージェントの作成者はエージェントに行わせたい処理を自ら記述するだけにとどまらず，AgentSphereネットワーク上に存在する処理モジュールを選択していくことによって，エージェントの処理を遂行させることができる。また，処理モジュール自体もAgentSphereを利用するユーザなら誰でも作成可能にすることによって，より多くの機能がAgentSphereネットワーク上に存在するようになると考えられる。例えば，現在ライブラリとして研究・開発が進められている機能がAgentSphereネットワーク上に存在しているならば，ユーザはそのライブラリのダウンロードなどをしなくても，エージェントが利用できる機能として，そのライブラリを利用することができる。

このように個別の処理モジュールをエージェントが利用するためには，エージェントが利用できる機能としてAgentSphereに記述しておく必要がある。しかし，例えばあるAgentSphereのユーザが処理モジュールを増やしたとすると，そのAgentSphereは他のAgentSphereにはない固有の処理モジュールを持つことになる。そのため，エージェントがその処理モジュールを利用する場合，その処理モジュールが存在するAgentSphereに移動しなければならない。ここで，その処理モジュールへの要求が多すぎると，大量のエージェントがその処理モジュールが存在するAgentSphereに移動してくることによって，マシンに大きな負荷がかかってしまう。また，処理モジュールがAgentSphereに記述されてあるものだとしたら，その処理もそのAgentSphere内で行われる。この場合，処理実行中に何らかの不測の事態によってそのAgentSphereが存在するマシンが落ちてしまうと，処理が中断され，エージェントに結果を返せないという問題が生じる。

そこで本研究では，あるユーザが作成したエージェントが利用できる処理モジュール自体をエージェント化し，どのAgentSphereで生まれたエージェントでもこの特定の機能を持ったエージェントに処理を依頼することを可能にした。こうしてそれぞれ異なった機能を持つエージェントが大量にAgentSphereネットワークに存在すれば，ユーザはエージェントに行わせたい処理を自ら記述するのではなく，ユーザの行いたい処理に適した機能を持つエージェントを選択し，処理を依頼することによってエージェントに行わせたい処理を完了させることができる。更に，特定の機能を持ったエージェントも一つのエージェントに変わりないので，メッセージングやマシン間の自律的な移動が可能である。そのため複数のエージェントから処理を要求されても，特定の機能を持ったエージェントが複製されることによってそれぞれの要求に対応でき，複製されたエージェントがそれぞれ自律的に移動を行なうので，一つのマシンに負荷が集中するという事態も起こらなくなる。

2．特定の機能を持ったエージェント

エージェントが利用できる処理モジュールをエージェント化することによって，ネットワーク上にこのエージェントが常に存在し続けるような状態となる。そして，他のエージェントから処理を依頼されたときにだけ，自らの機能を実行する。また，処理を依頼されていないときは，リソースの消費をできるだけ抑えるため休止状態

となっている。これによりユーザは，ネットワーク上に存在する特定の処理を行うエージェントを選択していくことによって，プログラムを作成することができる。

図1は，エージェントが自ら処理を行なうのではなく，特定の機能を持ったエージェントに処理を依頼する例を表している。ユーザは，図1のエージェントにおける，データ収集，集計，結果算出といった処理を記述するのではなく，ネットワーク上に存在する特定の機能を持ったエージェントを選択する記述を行なう。

図1　特定の機能を持ったエージェントへの処理依頼

特定の機能を持ったエージェントは処理を依頼されていないときは休止状態となっているが，定期的に起きてマシン間の自律的な移動を行う。これは，特定の機能を持ったエージェントが処理を依頼された時点でのAgentSphereで稼働を始めるので，これによってマシンが許容できる負荷の高さを超えないために行われる。

2．1　要求と解決手法

今回開発する機構を満たすための要求は大きく分けて二つある。一つ目は処理モジュールをエージェント化する方法，二つ目はエージェント化された処理モジュールの利用方法である。

2．1．1　処理モジュールをエージェント化する方法

特定の機能を持ったエージェントに要求されることは，常にネットワーク上に存在すること，必要なときにだけ呼び出されて特定の処理を行なうこと，完了した処理の結果を依頼したエージェントに返すこと，AgentSphere上のすべてのエージェントが利用可能なこと，複数の要求が集中したときにそのエージェントの複製を行なうことである。

これらの振る舞いは特定の機能を持ったエージェントすべてに共通される振る舞いである。そこで，これらの振る舞いをする抽象的なエージェントのコードをあらかじめAgentSphereに準備しておく。ユーザはその抽象的なエージェントを継承して特定の機能を持ったエージェントを作成することができる。これによって，ユーザは特定の機能を持ったエージェントすべてに要求される振る舞いを考慮することなく，処理モジュールの作成にのみ集中することが可能となる。

2．1．2　特定の機能を持ったエージェントの利用方法

特定の機能を持ったエージェントを利用するためには，まずネットワーク上にどのような機能を持ったエージェントが存在しているのか知っていなければならない。それを知ったうえで，目的のエージェントを選択して，そのエージェントに処理を依頼する。そして，処理を行ってもらい，その結果を受け取る。これが特定の機能を持ったエージェントの利用の流れである。

この利用の流れをユーザが意識せず，一つのメソッドの実行によって処理の依頼を行うことができ，処理依頼中も自律的なマシン間の移動を行なう抽象的なエージェントを，あらかじめAgentSphereに準備した。ユーザはエージェント作成の際，この抽象的なエージェントを継承することによって，特定の機能を持ったエージェントに処理を依頼できるエージェントを作成することができる。

今回，処理の依頼にはAgentSphereのモジュールの一つである，メッセージング機構[1]を利用した。このメッセージング機構は，エージェント同士が通信する際，お互いが別のマシンに存在する場合，通信したい方のエージェントがメッセンジャーというエージェントを生成し，そのメッセンジャーに情報の伝達を行わせるという仕組みをとっている。これによって，メッセージングを行なう際も，エージェントは処理を継続することができる。すなわち，特定の機能を持ったエージェントに処理を依頼する際，メッセンジャーを通して処理を依頼することによって，処理を依頼したエージェントは自律的なマシン間の移動を行なうことが可能となる。また，このメッセージング機構によってなんらかのデータを渡す際，そのデータの型や数は問わずに渡すことが可能である。これは処理を依頼する際や，処理が終わった後の結果を受け渡す際に有用である。

2．1．3　ユーザの記述部分

図2は特定の機能を持ったエージェントを作成する際にユーザが記述する部分と，そのエージェントに処理を依頼する際に記述が必要なメソッドを示している。

特定の機能を持ったエージェントを作成するには，抽象的なエージェントに準備された”特定の処理を行う”という抽象メソッドをオーバーライドし，具体的な処理を記述する。この処理が，依頼してきたエージェントに渡

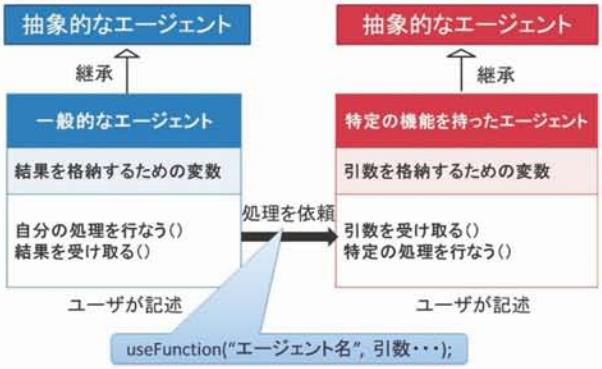

図２　ユーザの記述部分

されたデータをもとに行うものならば，そのデータを格納するための変数を準備する必要がある。通常のメソッドを記述する際は，メソッド名の直後の括弧内に仮引数を指定する。しかし，特定の機能を持ったエージェントはそれぞれ，異なる型，異なる数のデータを受け取るため，”特定の処理を行う”抽象メソッドは仮引数を一つに決めることができない。そのため，この抽象メソッドは仮引数を持たず，データを受け取らないようにした。その代わりにデータを受け取る方法として，処理を依頼されたときに呼び出されるメッセージング機構のメソッド内で，メッセージとしてデータを受け取るようにした。メッセージング機構で扱われるメッセージに型や数の制限はないので，汎用的な引数の渡し方となる。受け取ったデータを格納するための変数は，”特定の処理を行う”メソッドで使用するので，メンバ変数として宣言する。また，特定の処理を行なうメソッドの最後には，処理を依頼したエージェントに結果を返すために，抽象的なエージェントに準備されたメソッドを，結果を引数にして呼び出す必要がある。

一般的なエージェントが特定の機能を持ったエージェントに処理を依頼するには，自分の処理を行なうメソッド内に，

useFunction(“エージェント名”, 引数・・・);

と記述する。これによって，”エージェント名”の特定の機能を持ったエージェントに処理を依頼できる。また，”エージェント名”の後ろに，特定の機能を持ったエージェントに渡したいデータを指定することによって，引数としてデータを渡すことができる。依頼した処理を終えた後の結果は，メンバ変数に結果を格納するための変数を準備しメッセージングを行う際に呼ばれるメソッドで値を格納する。

2．2　一つのエージェントを複数のエージェントが同時に利用する場合

特定の機能を持ったエージェントは必ずしも一つのエージェントにだけ利用されているとは限らない。それよりも同じタイミングで複数のエージェントに処理を依頼され，別々のエージェントのために処理を行なう場合の方が多いと考えるのが自然である。しかし，あるユーザが作成した特定の機能を持ったエージェントがAgentSphereネットワーク上に一つしか存在しない場合，そのエージェントがすでに処理を依頼され実行中であると，他のエージェントが処理を依頼することができなくなってしまう。

そこで，これを解決するために，特定の機能を持ったエージェントがすでに処理を行っているときに他のエージェントが処理を依頼してきた場合，その機能を持ったエージェントが引数を受け取っていない状態でコピーされ，そちらに処理を依頼できるようにした（図3）。このコピーされたエージェントは，依頼された処理を完了し依頼したエージェントに結果を返した後に消滅する。一方オリジナルのエージェントは，依頼された処理を終えても消滅せずに，再び処理の依頼を受けられる状態でAgentSphereネットワーク上に存在し続ける（図4）。このように，オリジナルのエージェントが実行中でないときに処理の依頼をされたら自らが処理を行い，すでに依頼

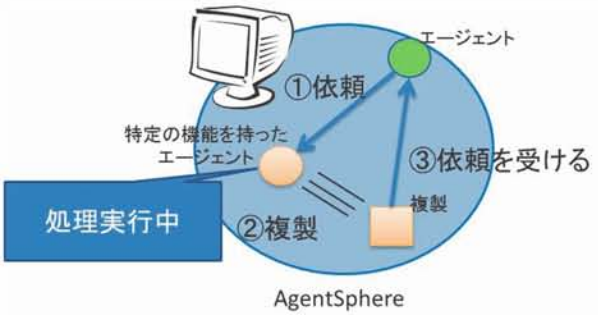

図３　処理実行中の特定機能を持ったエージェントに依頼を行う場合

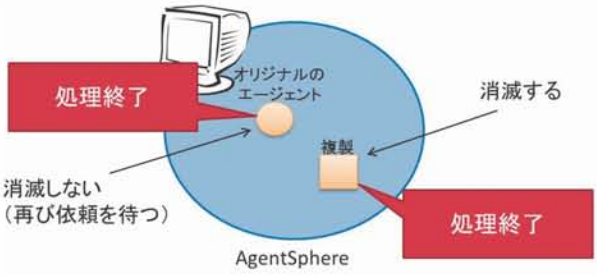

図４　依頼された処理を終え、結果を返した後のオリジナルと複製の挙動

された処理を実行中ならば，自らのコピーを作成しそちらに処理を任せることによって，複数のエージェントが同時に一つの特定の機能を持ったエージェントの機能を利用することが可能になる。

2．3　エージェントのバージョンアップ

ある特定の機能を持ったエージェントをAgentSphere上に放流したとする。その後そのエージェントの作成者が，同じ機能を持った，より精度の高い処理モジュールを開発した場合，すでにネットワーク上に放流した古いエージェントに処理を依頼するより，新たに開発した処理モジュールをエージェント化し，そちらに処理を依頼するほうが良い。これを行うためには，バージョンアップされた機能を持ったエージェントがネットワーク上に放流されたとき，古いエージェントが消滅し，代わりにバージョンアップされたエージェントがネットワーク上に存在するという機構が必要である。

これを行うにあたり二つの状況が考えられる。それは，バージョンアップされたエージェントがネットワーク上に放流されたときに，同じ機能を持った古いエージェントが仕事をしている場合と，仕事をしていない場合である。仕事をしていない場合は，バージョンアップされたエージェントが，古いエージェントが存在するマシンへ移動し，古いエージェントに停止依頼を行う。これにより，古いエージェントは消滅し，バージョンアップされたエージェントがネットワーク上に存在することになる。一方，古いエージェントが仕事をしている場合は，図 5 のようにエージェントのバージョンアップを行う。まず，古いエージェントが仕事をしていない場合と同様，バージョンアップされたエージェントが，古いエージェントが存在するマシンへ移動する。処理実行中の古いエージェントは，処理を依頼したエージェントから何らかのデータを渡され所持している。また，処理を依頼したエージェントの名前も所持している。そこで，それらをバージョンアップされたエージェントが受け取り，その後に古いエージェントに停止依頼を行う。バージョンアップされたエージェントは受け取ったデータをもとに古いエージェントが依頼された処理を実行する。以上を行うことによって，特定の機能を持ったエージェントのバージョンアップが可能となる。

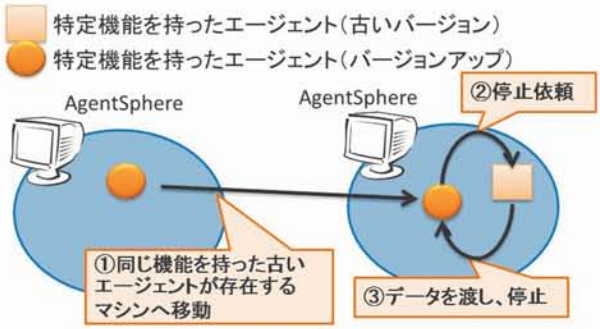

図 5　特定機能を持ったエージェントの更新

3．動作検証

今回はNQueen問題を解く機能を持ったエージェント[2]を作成し，このエージェントに他のマシンで起動された一般的なエージェントが処理を依頼できることを確認した。具体的には図 6 のように実験を行った。

まず，AgentSphere"D"でNQueen問題を解くエージェントを起動しておく。その状態でAgentSphere"A"で起動されたエージェントが，機能を持ったエージェントに処理を依頼する(①)。このエージェントはn=10 からn=16 までの解を求めるため，計 7 回処理を依頼している。NQueen問題を解くエージェントは依頼された時点で，与えられた値nのときのNQueen問題を解くという処理を開始する(②)。そして結果が出るたびにその結果を依頼したエージェントに返すが，nの値が大きいほどNQueen問題を解くのに時間がかかる。特にn=15,n=16 のときはかなりの時間を要する。その段階で，AgentSphere"B"で新たにエージェントを起動し，同様に処理を依頼する(③)。このとき，NQueen問題を解くエージェントはすでに依頼された処理を行っているため，自らのクローンを生成する(④)。AgentSphere"B"で起動されたエージェントはそのクローンに処理を依頼し，クローンは別のAgentSphereに移動してから処理を開始する(⑤)。NQueen問題を解いたクローンは結果をAgentSphere"B"で起動されたエージェントに返し，消滅する。これらの挙動が正常に行われるかどうかの実験を行った。結果は図 7 のようになった。

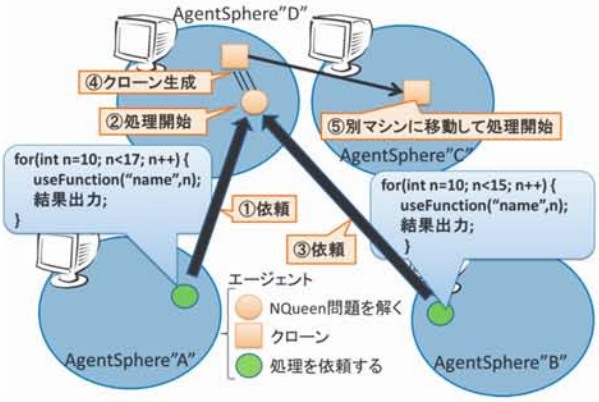

図 6　動作検証

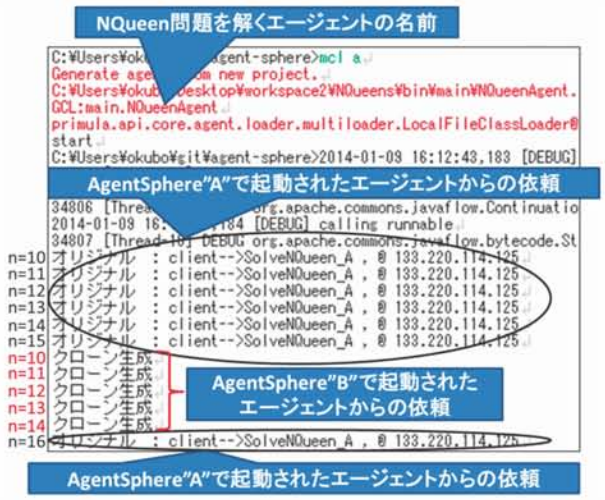

図 7　AgentSphere"D"の実行結果

これを見ると，NQueen問題を解くエージェントはAgentSphere"A"で起動されたエージェントから処理の依頼を受けていることがわかる。また，依頼された処理実行中に，AgentSphere"B"で起動されたエージェントから処理を依頼されクローンを生成していることがわかる。さらに，AgentSphere"A"，"B"における実行結果を見ると結果が返されていること，AgentSphere"C"における実行結果を見るとクローンが移動してきて処理を行っていることを確認できた（図 8，図 9）。

図 8　AgentSphere"A"の実行結果

ここで示した動作検証以外にソート機能を持ったエージェントを作成した。このエージェントにString型の配列，int型の配列を，それぞれ任意の要素数渡すとそれらが正確にソートされて返されることを確認できた。このことから，引数の渡し方が汎用性に対応しているといえる。

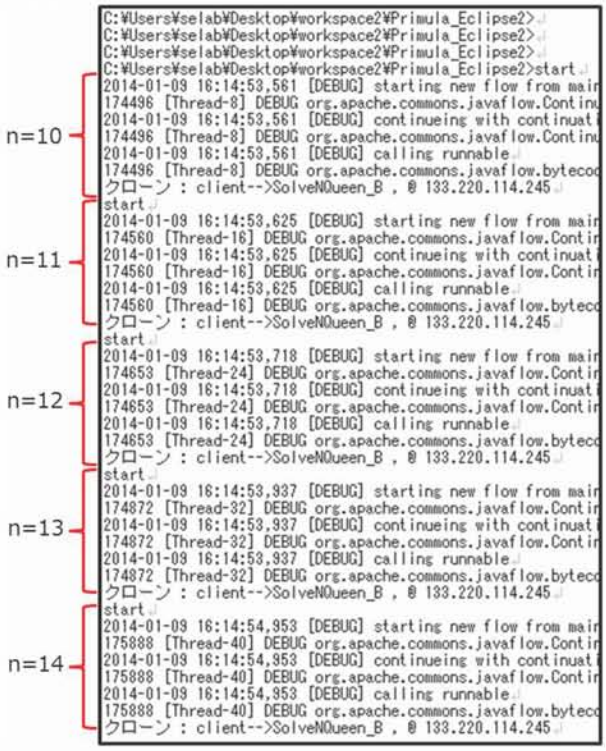

図 9　AgentSphere"C"の実行結果

4．まとめと今後の課題

本研究により，エージェントがAgentSphereネットワーク上に存在する特定の機能を持ったエージェントを選択していくことによって，ユーザが行わせたい処理を完了できるようになった。これによって，ユーザのコード記述の負担を軽減させ，エージェント作成効率を向上させることができると考えられる。

しかし，特定の機能を持ったエージェントに処理を依頼するには，現状ではそのエージェントの名前が必要である。すなわち，AgentSphereネットワーク上にどのような機能を持ったどのような名前のエージェントが存在しているかをユーザが把握していなければならない。これを解決するために，エージェントの名前ではなく，機能による検索を行い，処理を依頼できることが必要である。

引用文献

[1] 鈴木幸祐・甲斐宗徳：「AgentSphere における分散ハッシュテーブルを用いた情報共有機構」，成蹊大学大学院理工学研究理工学専攻修士論文，2012

[2] 山口大介・甲斐宗徳：「AgentSphere における AgentPool の実現と Master/Slave 型並列 API の作成」，成蹊大学大学院理工学研究理工学専攻修士論文，2011