# JXTAを利用したP2Pによる分散開発環境

## — ソフトウェアエージェントを使ったアジャイルソフトウェア開発環境の構築 —

長尾雄行
産業技術大学院大学　産業技術研究科
小山裕司
実践女子大学　人間社会学部

**概　要**

ソフトウェアエージェントを利用して分散プログラム開発環境を構築する。ファイルシステムの変更監視を行うFAMエージェントとコンパイルとテストを行う構築エージェントを作成する。これらのエージェントを単純なメッセージ機構により接続することで疎結合のプログラム開発環境を構成する。メッセージの通信路としてJXTAによるP2Pオーバーレイネットワークを利用する。このような開発環境を利用することで、1人のプログラマが地理的に分散した多数のビルドサーバを同時に利用してテストコードの実行と動作確認を行うことができる。その結果、プログラマ1人当たりの生産性が向上し、頻繁なテストが必要なアジャイルソフトウェア開発に有意義な結果をもたらす。

### 1.　はじめに

ソフトウェアの開発中に行われる、ソースコードの編集、コンパイル、テストという基本工程はプログラマが何度も繰り返し実行する手順である。アジャイルソフトウェア開発では、リファクタリングや機能強化を行いながら、これらの手順をより頻繁に実行することが求められる。したがって、この一連の工程からプログラマの手による操作を排除して、自動化することは生産性の向上に必要不可欠である。

本稿では、前述の工程のうち、ソースコードの編集後にプログラマがキーボードやマウスなどの操作を行わずに、自動的にコンパイルやテストが実行され、その結果が目視確認できる開発環境の構築を行う。複数のホストで様々な設定の下でテストコードが実行できるように、テスト環境の分散化も行う。

### 2.　ソフトウェアエージェント

ソフトウェアエージェントは人工知能の研究の中で見出されたものであり、複雑なソフトウェ

アシステムを自律機能するソフトウェアエージェントの集合体として構成する手法がソフトウェア工学においても研究されている［12］、［13］。ただし、ソフトウェアエージェントには様々な種類のものが考案されており、その定義や分類方法に関してはコンセンサスの形成には至っていないのが現状である［16］、［18］。

ソフトウェアエージェントの導入によって開発環境の構成要素を疎結合にすることで、上記の目的が達成されることを示すのが本稿の目指すところである。まず着目するのは、コンパイラの起動を自動化することである。そのためには、プログラマがソースコードを編集して保存した瞬間を捉えればよい。

Eclipse［4］のような統合開発環境（IDE）では、エディタとコンパイラが１つのソフトウェアにまとめられている。そのため、ソースコードの変更が行われると、IDE はすぐにコンパイラを起動することができる。エディタとコンパイラが別々のソフトウェアとなっている場合でも、ソースコードの変更を自動検出することができれば、エディタからコンパイラへと制御を移すことができる。

我々が採用するのは後者の方法である。つまり、ソフトウェアエージェントによりソースコードの変更を監視し、変更通知をコンパイル用のソフトウェアエージェントに送信する。コンパイル用のソフトウェアエージェントは変更通知を受信後に適切なビルドツールを起動してコンパイルとテストを行う。この方法に、プログラマの手を介さずにコンパイルの自動化を行うのが本稿の方針である。

## 3.　ソースコードの変更検出と FAM エージェント

ソースコードの変更を検出するためには OS の FAM（File Alteratrion Monitor）機構を利用するのがもっとも確実であるが、この機能は必ずしもすべての OS が備えているわけではない。重要なのはファイルの保存開始ではなく、保存終了を検出することである。非常に長いソースファイルを保存する場合には、ファイルを書き込みモードで開いてから内容を書き込んで閉じるまでには時間がかかる。この場合、ソースファイルの保存開始を検出した直後にコンパイラに制御を移すと不完全な内容のファイルを扱うことになり、適切な処理が行えない。

OS に用意された FAM 機構としては、Linux（カーネル 2.6.13 以降）の場合はカーネルの inotify サブシステムがある。inotify では、CLOSE_WRITE イベントを利用するとファイルの保存終了を検出できる。それ以前には dnotify サブシステムというものもあったが、こちらはディレクトリしか監視できなかった。Windows の場合には Win32API に用意されている FindFirstChangeNotification 関数などを利用することができると思われる。

また、ソースコードを編集するためのエディタに用意されているフックを利用することも良い方法である。たとえば、最近の VIM［6］には BufWritePost というフックが用意されており、これを利用するとソースコードの保存完了を検出してスクリプトを実行することができる。

OS やエディタの機能を使わずにファイルの変更を検出する方法は、ファイルのタイムスタンプを一定期間で監視する方法である。この方法は我々の目的であるソフトウェアエージェントを

使った疎結合化に最適である。しかし、この方法では、ファイルの保存開始は検出できるが、保存終了は検出できないという重大な欠点がある。我々はこの欠点をエディタのフックや OS の FAM 機構を併用することによって補う方法を採用する。

プログラマがソースコードの編集を行うホスト上に、ソースツリーを監視するソフトウェアエージェントを準備する。これを**FAM エージェント**と呼ぶことにする。FAM エージェントはエディタとは独立したプログラムであり、ソースファイルの変更を検出すると他の機能を担うソフトウェアエージェントにソースファイルの変更通知を送信する。

## 4. ビルドエージェント

ソースファイルの変更通知を受けてビルドを実行するためのソフトウェアエージェントを用意する。それが**ビルドエージェント**である。ビルドエージェントはソースファイルの変更通知を受信後に GNU Make, Apache Ant や Apache Maven などのビルドツールを実行する。ソースコードをコンパイルする作業と、ビルドによって生成されたバイナリなどをテストする作業はビルドツールによって一括して実行できるので、ビルドエージェントではテストの実行と動作確認もビルドと併せて行うものとする。

## 5. 同期エージェント

前述のように我々は複数のホストでテストを実行することを考えているので、複数のホストでファイルを同期する仕組みを用意する。そのためのソフトウェアエージェントを**同期エージェント**と呼ぶことにする。同期エージェントは他のエージェントからメッセージを受け取ると、リポジトリにあるディレクトリツリーを取得して動作中のホストの作業ディレクトリの内容を最新の状態に更新する。更新後に、リポジトリの同期が完了したという事実を他のソフトウェアエージェントに通知する。その通知はトリガメッセージを後述のオーバーレイネットワークに送信することにより行う。

## 6. 情報表示エージェント

複数のホストで動作するソフトウェアエージェントの活動状況を収集して、ソースコードを編集しているプログラマに表示する機能が必要である。そのために、**情報表示エージェント**を用意する。このソフトウェアエージェントは純粋にログの表示だけを担当し、プログラマが操作する端末上で動作する。プログラマはリアルタイムにビルドやテストの進捗状況を目視で確認できることが望ましい。

本稿のソフトウェア開発環境では、プログラマがソースコードを保存すると自動的にビルドとテストが複数のホストで並列実行される。情報表示エージェントは多数のソフトウェアエージェントからのログ出力を収集してプログラマが目視確認を行いやすい形式に表示する。このことにより、１人のプログラマが効率よく多数の環境下でテストとデバッグを行うことができる。

## 7.　ソフトウェアエージェントの通信路

以上で説明したソフトウェアエージェントの通信路として、JXTA による P2P オーバーレイネットワークを利用する。分散処理を行うために、ソフトウェアエージェント同士のメッセージが一対多に配信される仕組みが望ましい。JXTA ではピアグループの管理とピア間の通信に利用するパイプの管理が Advertisement を用いて比較的簡単に行える。また、JxtaMulticastSocket という一対多の通信方式が用意されているため、本稿では JXTA を利用してメッセージの交換を行うこととした。ファイアーウォールやプロクシサーバを越えた接続が比較的簡単であるというのも JXTA を選ぶ理由である。

## 8.　実装例

### 8.1.　実験環境

本稿で利用した実験環境は以下の通り。

| | |
|---|---|
| **PC1** | Xeon2.8GHz　2 基/HDD80GB/メモリ 1 GB/WindowsXP Pro |
| | JDK6.0 |
| | VMWareServer1.0 (ゲスト OS は VM1) |
| **PC2** | Xeon2.8GHz　2 基/HDD80GB/メモリ 1 GB |
| | Linux Slackware10.0/Kernel 2.6.16.9 |
| | JDK6.0/Apache Ant 1.6.5/VIM 6.3 |
| | VMWareServer1.0（ゲスト OS は VM2） |
| **VM1** | CentOS-4.3/subversion/JDK1.6.0/Apache Ant 1.6.3 |
| **VM2** | CentOS-4.3/subversion/JDK1.5.0/Apache Ant 1.7.0 |

ただし、本稿では同一サブネット内のピアについてのみ実験を行った。つまり、これらのホストは同じサブネットに属する。

### 8.2.　実装の詳細

ソフトウェアエージェントの実装には主に Java 言語を利用した。また、XML ファイルにより各ソフトウェアエージェントの動作設定を行うことができるようにした。主なソフトウェアエージェントと Java クラスについて以下で詳しく説明する。

#### 8.2.1.　FileAlterationMonitor（FAM エージェント）

ファイルシステムを監視して、ファイルの変更・削除・追加を検出する FAM エージェントが FileAlterationMonitor である。FileAlterationMonitor はファイルの状態変化を検出後、設定ファイルに記述されたコマンドを実行する。コマンドが成功すると、JxtaMulticastSocket を利用して他のピアに対して変更通知を送信する。コマンドが失敗した場合には他のピアには何も通知されない。

FileAlterationMonitor は Java 言語で実装されたコマンドラインツールである。Java 言語では FAM 機能は用意されていないので、定期的に指定ディレクトリ配下のファイル一覧を作成して、前回作成したファイル一覧との差分を取ることによって、どのファイルが追加・削除・変更され

たかを監視する。

FileAlterationMonitor はプログラマがファイル編集を行う編集サーバで起動される。起動時に JXTA のオーバーレイネットワークへの接続が行われ、他のピアとのメッセージ交換が始まる。FileAlterationMonitor はこの初期化処理の後、ファイル監視を行うループへと制御を移す。ファイルの監視は監視対象ファイルのタイムスタンプを定期的に調査することによって行う。

### 8.2.2. fam-config.xml（FAM エージェントの設定ファイル）

FileAlterationMonitor の動作設定を行う設定ファイルが fam-config.xml である。fam-config.xml では、監視項目を以下の(1)－(3)のように設定可能である。(1)監視対象のディレクトリを指定する。(複数可) (2)ファイル名を正規表現で指定する。(3)状態変化を検出した際に起動する外部コマンドを指定する。fam-config.xml に複数の監視項目を設定することもできる。fam-config.xml の設定例を以下に挙げる。

```
<?xml version="1.0" encoding="UTF-8"?>
<FAMConfig xmlns="http://fam.ed.jxta.aiit.ac.jp/xmlns/fam" >
 <item id="test-make">
  <fileset>
   <directory>/dat3/develop/jxta/test</directory>
   <filePattern>^(Makefile|.*.(pm|pl))$</filePattern>
  </fileset>
  <actionset>
   <action><![CDATA[
     cd   /dat3/develop/jxta/test   &&   make   &&   svn   commit   -m
"automatic"   ]]></action>
  </actionset>
 </item>
</FAMConfig>
```

図 1　fam-config.xml

### 8.2.3. Message クラスとメッセージ交換

ピア同士のメッセージ交換に Message クラスを利用する。Message クラスは本システムで使われるメッセージオブジェクトの基本クラスである。その役割は、メッセージに識別用の ID を紐づけて管理することにある。実際にメッセージ交換を行う際には、文字列情報をやり取りする必要があるので、この Message クラスを継承した StringMessage クラスを利用する。StringMessage クラスでは、Java 言語の String クラスのインスタンスを保持することができる。

メッセージ交換の通信路として JXTA のオーバーレイネットワークを使う。IP マルチキャストを利用することも可能だが、JXTA を利用することでルータやファイアーウォールを越えたピアの接続が容易となる。また、IP マルチキャストを利用した場合、マルチキャストグループの割り当てと管理を行う必要があるが、JXTA を利用するとこの種の操作が Advertisement の生成と発行という手続きによって簡単になる。

実装コードでは、送信側のピアは StringMessage クラスのインスタンスを直列化して、JxtaMulticastSocket を使って受信側のピアに送信する。受信側のピアでは、受信したデータグラムパケットから StringMessage クラスのインスタンスを復元して、適切なメッセージ処理機構に処理を移す。今回はメッセージ処理機構が簡単になるように文字列のやり取りのみを行うこととした。

### 8.2.4. ShellCommunicator（ビルド・テストエージェント）

他のピアからのメッセージを受信して、メッセージに応じたコマンドを実行するソフトウェアエージェントが ShellCommunicator である。このソフトウェアエージェントもコマンドラインツールである。ShellCommunicator は起動時にオーバーレイネットワークに接続し、他のピアからのメッセージ受信待ちループに入る。そして、メッセージを受け取ると適切な外部コマンドを実行する。メッセージと外部コマンドの対応は XML ファイルで設定を行う。ShellCommunicator の設定ファイルを切り替えることで、コンパイル用、テスト用のビルドエージェントを実装することが可能である。設定ファイルの例を以下に示す。

```
<?xml version="1.0" encoding="UTF-8"?>
<FAMConfig
xmlns="http://fam.ed.jxta.aiit.ac.jp/xmlns/fam">
 <handler hook="test-make"><![CDATA[
   svn update && make test
 ]]></handler>
</FAMConfig>
```

図２　build-agent.xml

### 8.2.5. InfoAgent（情報表示エージェント）

複数のソフトウェアエージェントからのログ出力を収集して表示するためのツールが InfoAgent である。

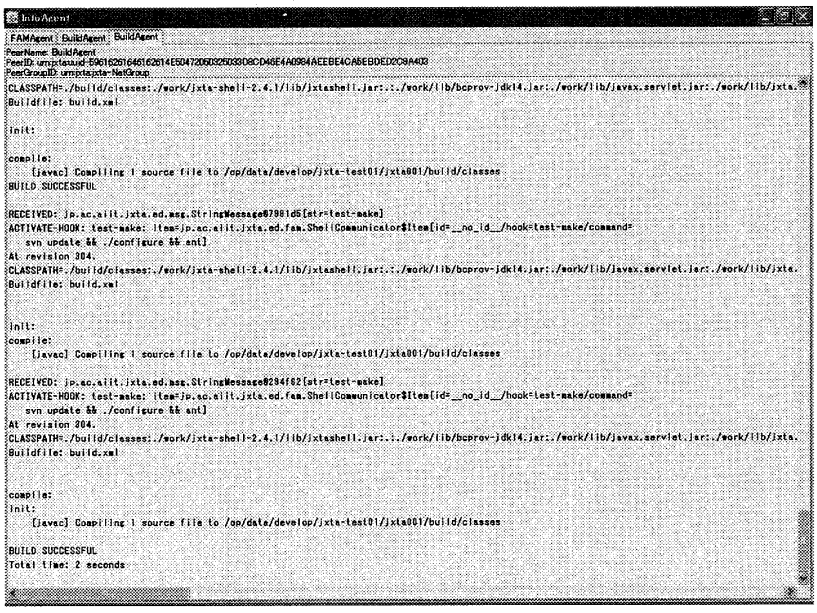


図３　Swing アプリケーションとして実装された InfoAgent

InfoAgent は Swing アプリケーションであり、複数のソフトウェアエージェントからのログ出力を JXTA のオーバーレイネットワーク経由で受信する。ログの表示は 1 つのソフトウェアエージェントにつき 1 つのタブを利用して行う。

### 8.2.6. 同期エージェント

今回の実装では、同期エージェントは前述の FileAlterationMonitor と subversion [8] を利用して実装した。変更をリポジトリに反映する機能は FileAlterationMonitor の設定ファイル fam-config.xml で svn commit を発行するように設定することで実現し、リポジトリから作業ディレクトリを取り出す機能は、ShellCommunicator の設定ファイルで svn update を実行することにより実現した。

### 8.2.7. エディタのフックを利用したファイルの保存終了検出

実験環境で利用するエディタとしては Linux 上の VIM 6.3 を利用した。前述のように、Java 言語で実装された FileAlterationMonitor ではファイルの保存終了を検出することができない。そこで、以下のような工夫を行った。

1．適切なディレクトリに fam.fifo という名前で名前つきパイプを作成する。
2．FileAlterationMonitor では fam.fifo ファイルの内容を読み続けるスレッドを作成し、fam.fifo が変更されると監視中のディレクトリツリーを再走査する。
3．VIM の設定ファイルの vimrc において BufWritePost フックに関数を登録し、その関数の中身で fam.fifo にトリガメッセージを書き出すようにする。

この工夫により、FAMAlterationMonitor の欠点である、ファイルの保存終了を検出できない点を改善することができた。

## 9. 動作検証

実装の動作検証として、本システムの Java 言語によるソースツリーを利用した。プログラマは編集サーバ(Linux)にログインし、svn リポジトリから作業ディレクトリをチェックアウトする。続いて、ソフトウェアエージェントの起動を行う。まず、編集サーバ上でコマンドラインから FAMAlterationMonitor を起動する。次に、テストサーバ上で ShellCommunicator を起動して、ビルドエージェントを準備しておく。エディタとして利用する VIM には前述のフック設定する。また、名前つきパイプの準備も行う。ログの収集と表示を行う InfoAgent も起動しておく。

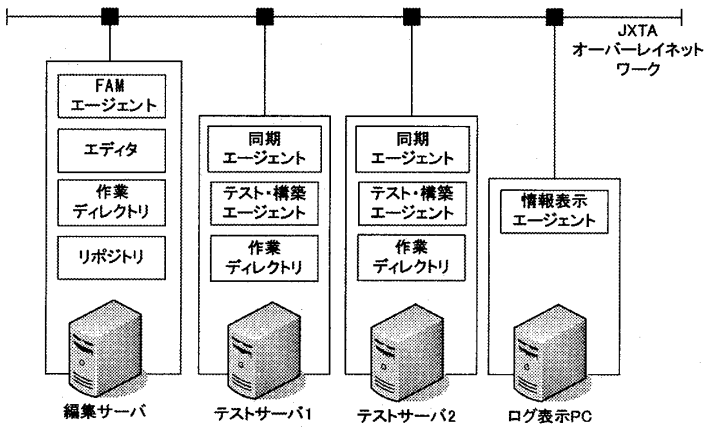


**図4　ソフトウェアエージェントの構成図**

以上の準備の下で本システムの動作確認を行った。プログラマはソースツリーを VIM で編集し、変更内容を保存すると、即座に FAMAlterationMonitor がファイルの変更を検出し、編集サーバ上でビルドが開始された。ビルドが成功すると、FAMAlterationMonitor は、設定にしたがってソースツリーを svn リポジトリにコミットして、コミット終了のトリガメッセージをオーバーレイネットワークに送信することが確認された。２つのテストサーバ上で待機していたビルドエージェントはそのトリガメッセージを受信して、svn リポジトリから作業ディレクトリをチェックアウトして、ビルドを実行してその結果を InfoAgent に送信することが確認された。

以上の手順を繰り返すことで、２つのテストサーバ上で同時にビルドを実行しながら、プログラマはソースコードの編集に集中できることを確認した。この動作検証では、実験環境の仮想マシン VM1 と VM2 をテストサーバとした。VM1 と VM2 では JDK のバージョンが異なっており、この２つの JDK のもとでの動作検証が同時に行える利点が確認できた。

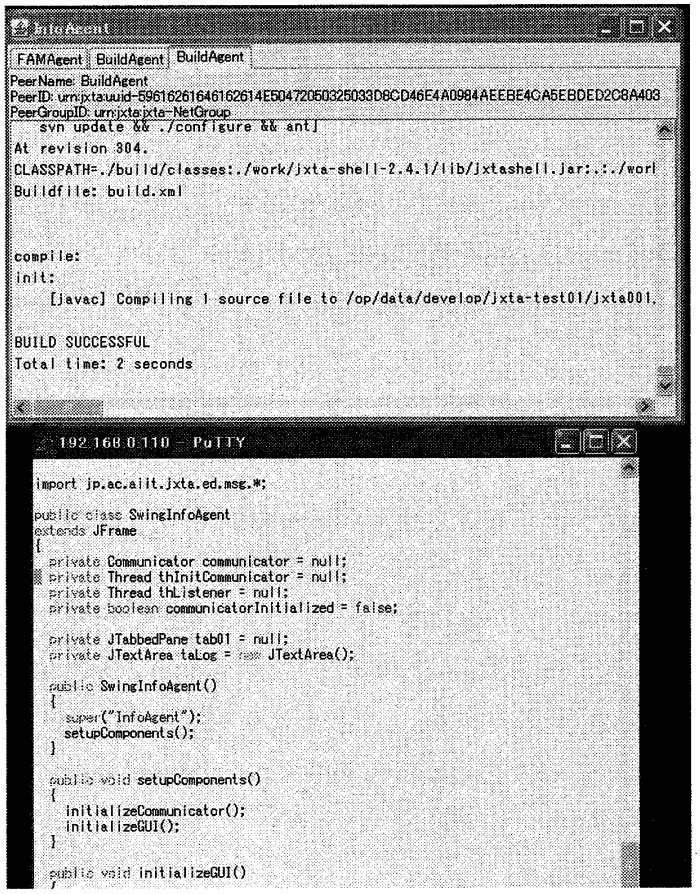


**図５　ソースコード編集の様子**

## 10.　考　察

本稿の開発環境を用いることで、ソフトウェア開発を行うプログラマはソースファイルを編集することに集中することができる。それは、コンパイルからテストまでの工程に、コンパイラを起動したりテストコードを実行するためにコマンドを入力するといった、人の手による操作が必要なくなるからである。ソフトウェアエージェントによってソースファイルの変更を自動検出す

ることの利点がここにある。

プログラマは常にソースコードを編集しながら、他のホストで自動実行されているビルドエージェントの進捗状況を確認することができる。ソースコードの編集とテストがほぼ同時に行えるので、ソースコードに混入する欠陥を早い段階で検出することが可能である。これは、アジャイルソフトウェア開発にとって非常に有利である。

ソフトウェアエージェント化を行うことにより、コンパイルとテストは多数のホストで同時に実行することができる。たとえば、複数の処理系で動作するソフトウェアを開発する際に、プログラマ１人あたりの生産性を向上させることが可能である。また、開発現場で余っている計算機を利用してテスト工程を１つ追加するといったことが可能となる。

本稿の実装では、Java 言語で実装された Message クラスのインスタンスを直列化して通信路に送信する方法でソフトウェアエージェント間の通信を行った。この方法では、ソフトウェアエージェントを Java 言語以外の言語で実装することが難しい。ソフトウェアエージェント間の通信は SOAP などを使って他の言語での実装が容易となるようにすることが今後の課題である。そうすることで、ソフトウェアエージェントがより疎結合となり、開発環境の拡張性が高まると考えられる。

エディタ、コンパイラ、テストツールなど、統合開発環境の構成要素をソフトウェアエージェントにより疎結合とすることで、新しい処理系を追加することも簡単となる。通常、統合開発環境を新しい処理系に対応させる場合には、その統合開発環境に対するプラグインを作成しなければならない。これは、プラグインの仕様を詳しく知っている開発者のみが行えることである。しかし、本稿で示したようにエディタとコンパイラを疎結合となっていれば、新しい処理系に対応する場合にも、ビルドエージェントに新しい処理系のコンパイラを起動するコマンドを追加すればよい。この作業はプラグインの開発に比べて非常に簡単である。

## 11. おわりに

本研究では、ソフトウェアエージェントを利用したプログラム開発環境の構築を行い、コンパイルとテストの実行を自動化することができることを確認した。そして、ソフトウェアエージェント化によりテストを複数のホストで並列して実行可能であることも併せて確認した。

今後、本研究の成果をより発展させ、ソフトウェアエージェントを利用したビルドツールを完成し、オープンソースの成果物として公開したい。そうすることで、アジャイルソフトウェア開発を利用している開発者がより高性能のソフトウェアをより効率よく開発できるようになると考えている。

## 参考文献

[1] JUnit http://www.junit.org/

[2] Apache Ant http://ant.apache.org/

[3] Apache Maven http://maven.apache.org/

[4] Eclipse http://www.eclipse.org/

[5] JXTA http://www.jxta.org/

[6] VIM http://www.vim.org/

[7] inotify http://www.kernel.org/pub/linux/kernel/people/rml/inotify/

[8] subversion http://subversion.tigris.org/

[9] J. C. Brustoloni, Autonomous Agents: Characterization and Requirements, Carnegie Mellon Technical Report CMU-CS-91-204, Pittsburgh: Carnegie Mellon University.

[10] FIPA (Foundation for Intelligent Physical Agents) http://www.fipa.org/

[11] JADE (Java Agent DEvelopment Framework) http://jade.cselt.it/

[12] N. R. Jennings, An agent-based approach for building complex software systems, Communications of the ACM 44, 2001, pp. 35－41.

[13] N. R. Jennings, On agent-based software engineering, Artificial Intelligence 117, 2000, pp. 277－296.

[14] KQML ( Knowledge Query and Manipulation Language) http://www.cs.umbc.edu/kqml/

[15] S. Liu, P. Kungas and M. Matskin, Agent-Based Web Service Composition with JADE and JXTA, http://ww1.ucmss.com/books/LFS/CSREA2006/SWW3144.pdf

[16] H. Nwana, D. Ndumu, A Brief Introduction to Software Agent Technology, Unicom Seminar on "Real-World Applications of Intelligent Agent Technology", London UK, pp.278－292.

[17] M. Wooldridge, Agent-based software engineering, IEE Proceedings of Software Engineering 144, 1997, pp. 26－37.

[18] M. Wooldridge, N. R. Jennings, Agent Theories, Architectures and Languages: a Survey, in Wooldridge and Jennings Eds., Intelligent Agents, Berlin: Springer-Verlag, 1995, pp. 1－22.