マルチメディア通信と分散処理　94－18
分散システム/インターネット運用技術　14－18
（１９９９．７．１６）

ADIPS Framework に基づく Java Beans のエージェント化手法

関場 治朗† 今野 勝之* 菅沼 拓夫† 木下 哲男† 白鳥 則郎†
† 東北大学電気通信研究所 / 情報科学研究科
* 富士通 (株) アウトソーシング事業本部
E-mail: jir@shiratori.riec.tohoku.ac.jp,katsuyuki_konno@mail7.dddd.ne.jp
{suganuma,kino,norio}@shiratori.riec.tohoku.ac.jp

あらまし　ADIPS フレームワークにおいて実際に利用者にサービスを提供するエージェント (プリミティブエージェント) の開発は主に,(1) エージェント化の対象となるプロセス固有のインターフェースに関する知識,(2) 様々な環境や利用ドメインでその対象プロセスを効率的に利用・運用するための機能・性能に関する領域知識, の 2 種類の知識を記述する必要があり, この作業が本フレームワークを用いた応用システム開発全体の作業負担の中で大きな割合を占めている. 本稿では, 上記の (1) の知識記述負担軽減に着目し, エージェント化する計算機プロセスを Java Beans に限定することで, 計算機プロセスのインターフェースに関する知識を自動的に生成し, エージェント開発者を支援する手法を提案する. また, 提案に基づき JAVA Beans を ADIPS フレームワーク上で動作するエージェントとしてエージェント化を行う実装例を示す.

キーワード　マルチエージェントシステム, JavaBeans, ADIPS フレームワーク

Java Beans agentification method based on ADIPS Framework

Jiro Sekiba†, Katsuyuki Konno*, Takuo Suganuma†,
Tetsuo Kinoshita† and Norio Shiratori†
†Research Institute of Electrical Communication /
Graduate School of Information Sciences, Tohoku University
***Fujitsu Outsourcing Business Group**
E-mail: jir@shiratori.riec.tohoku.ac.jp,katsuyuki_konno@mail7.dddd.ne.jp
{suganuma,kino,norio}@shiratori.riec.tohoku.ac.jp

Abstract　To write primitive agents on ADIPS Framework which offer service to users require main two knowledge, such are (1)knowledge of interface of each processes which is target for agentification,(2)domain knowledge related to functionality or capability to utilize target process efficiently on various environment and used domian.It is pointed to that describing these two knowledge are large part of devellopingprocess and developer's load to build applied system using ADIPS framework. In this paper, we focus on that helping (1), propose method for automated knowledge description to help agent developer. Futhermore, we show example of Java Beans agentification process based on the propose method.

key words　multi agents system, JavaBenas, ADIPS Framework

1　はじめに

これまでに, 利用者の多種多様な要求に対応する利用者指向のやわらかい分散システム・情報システムの概念やアーキテクチャに関する研究が進められてきている [1, 2]. これを実現するための手法として, エージェント指向の概念に基づいたエージェント型分散処理システムである ADIPS フレームワークが提案されている [3, 4] が,ADIPS フレームワークに基づく応用システムの開発時のエージェント記述作業負担が大きいことが問題として指摘されている.

ADIPS フレームワークでは, エージェントどうしが互いに協調動作することで, 利用者の多種多様な要求に応じたサービスを動的に構成・提供する. 本フレームワークにおいて実際に利用者にサービスを提供するエージェント (プリミティブエージェント) は, 既存の計算機プロセスに, エージェントとして動作させるための機構をつつみこみ手法により付加すること (エージェント化) により構築される.

しかしながらこの開発においては主に

1. エージェント化の対象となるプロセス固有のインターフェースに関する知識

2. 様々な環境や利用ドメインで,その対象プロセスを効率的に利用・運用するための,機能・性能に関する領域知識

の2種類の知識を記述する必要があり,この作業が本フレームワークを用いた応用システム開発全体の作業負担の中で大きな割合を占めている.そこで本研究では,ADIPS フレームワークにおけるプリミティブエージェントの開発効率を向上させるためのエージェント化支援手法を考案し,その手法に基づく支援システムの設計開発を行うことで,応用システム開発者の負担を軽減させることを目的としている.

本稿では,特に上記 (1) の知識記述負担軽減に着目し,エージェント化する計算機プロセスを Java Beans に限定することで,計算機プロセスのインターフェースに関する知識を自動的に生成し,エージェント開発者を支援する手法を提案する.これにより,エージェント開発者はより知識集約的な作業が必要となる (2) の領域知識記述に専念でき,より知的で柔軟なエージェントの開発が可能となる.

以下,2章では ADIPS フレームワークの概要と本稿で取り上げる問題点を述べる.3章において,Java Beans の概要と特徴を述べ問題点を解決する機能を説明し,4章でその機能の設計と実装を示す.5章では,本システムのプロトタイプを用いたエージェント化の実験について述べる.6章は本稿のまとめである.

2 ADIPS フレームワーク

2.1 ADIPS フレームワークの概要

ADIPS フレームワークとは種々の計算機プロセスを定式化 (エージェント化) することによって構成されるエージェント指向情報分散処理システム (Agent-based Distributed Information Processing System: ADIPS) の設計・開発方法論である.

ADIPS フレームワークでは利用者指向分散システムの実現に適した次の様な特徴が得られる.すなわち,

1. 利用者駆動で自律的にシステムが構成される

2. イベント駆動で自律的にシステムの再構成が行なわれる

3. 自律的なシステム構成 / 再構成のために,エージェントは設計者・運用技術者の知識を利用する

4. エージェント化により既存プロセスの系統的な再利用ができる.

これらの特徴を利用することで,利用者要求に基づいて分散処理システムを動的に構築し,利用者要求の変化や分散環境の変化に応じた,柔軟なサービスを提供し続けるが可能である.ADIPS フレームワークの概念図を図1に示す.

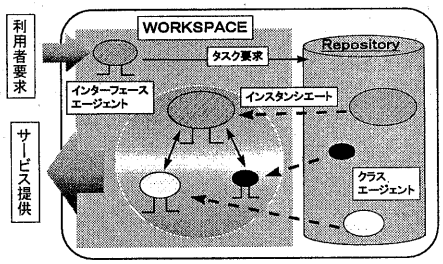

図 1: ADIPS フレームワーク

ADIPS フレームワークでは,利用者要求に基づいて,リポジトリ (Repository) と呼ばれるエージェントの保管庫内で,拡張契約ネットプロトコル [5] を用いて動的に分散処理システムを設計する.設計された分散処理システムは,動作環境 (Workspace) と呼ばれるネットワーク上の計算機内に存在するエージェント実行環境上に構成される.このとき,リポジトリ内のエージェント (クラスエージェント) からインスタンシエートされ,分散処理システムを制御 / 監視するエージェントをインスタンスエージェントと呼ぶ.

動作環境上で計算機プロセス (ベースプロセス) を直接制御し,分散処理システムの部品として利用する機能を持つエージェントをプリミティブエージェントという.

2.2 計算機プロセスのエージェント化とその問題点

計算機プロセスのエージェント化とは,上記のプリミティブエージェントを開発することにあたる.現在の ADIPS フレームワークにおいてエージェント化が可能な既存プロセスは,(1)Unix コマンド,(2) 外部インターフェースを持ったアプリケーション, (3)Java によるプログラム,の三つがある.

プリミティブエージェントのアーキテクチャを図2に示す.ADIPS エージェントは大きく分けて三つ部分に分けられる, すなわち CM(Cooperation Mechanism: 協調機構),DK(Domain Knowledge: ドメイン知識) 及び,TPM(Task Processing Mechanism: タスク処理機構) である.CM により ADIPS エージェントは他のエージェントと協調し,DK により推論を行い,TPM によりベースプロセスの制御を行う.

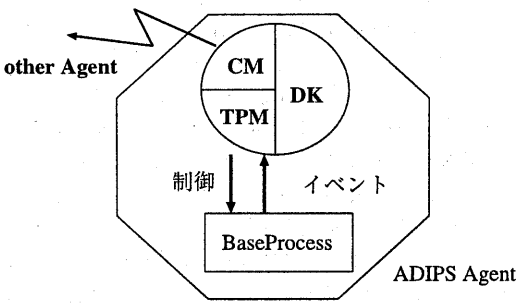

図 2: プリミティブエージェントの
アーキテクチャ

現在の ADIPS フレームワークでは, 多種多様な既存プロセスをエージェント化の対象として利用可能である. しかしながらその反面,

(P1) プロセスを制御するためのインターフェースに関する知識記述が, プロセス毎に異なっている為, エージェント開発者の負担が大きい

(P2) ベースプロセスを直接 TPM と接続し, 制御するという制約上, ベースプロセスが外部とのインターフェースを持たない場合, プログラムの改造などが必要となる

という問題が指摘されている.

(P1) の問題は,TPM 開発時の問題であり, この知識を記述するためにはベースプロセスに対しどのような入力を行い, その結果どのような出力が得られるかといったベースプロセスのインターフェースに関する深い知識が各ベースプロセス毎に必要になることが原因である.

また (P2) の問題はベースプロセス側の対応の問題であり,TPM から直接ベースプロセスを制御できない場合に, 図 3のようにプロセス内に TPM とのインターフェースを埋め込み TPM から制御可能とする必要があるため, プログラムに関する深い知識が必要になることに起因する.

第 3章において, これら問題に対する解決案として利用する Java Beans の概要を説明し, そのエージェント化手法について述べる.

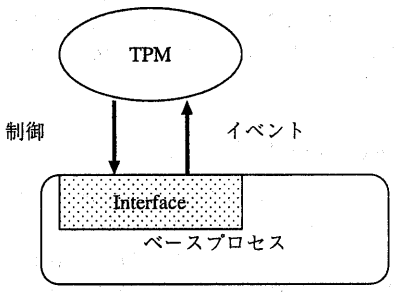

図 3: ベースプロセスの改造による
エージェント化

3 Java Beans のエージェント化

3.1 Java Beans

Java Beans とは,Java 言語で書かれたソフトウェアコンポーネントである. ソフトウェアコンポーネントとは, ある処理のために必要なプログラムの集合体で, 外部とのインターフェースや, 必要なデータ (画像等) を含む. このため非常に独立性が高く, プログラムの部品として再利用が容易である. 一般的に Java Benas は図 4に示す通り, 複数のクラスやデータを含んだものを指す. 個々の Java Beans を単に Bean と呼ぶ. 一般に Bean はこれらのクラスやデータを jar という Java アーカイブ形式で単一ファイルにアーカイブされる.

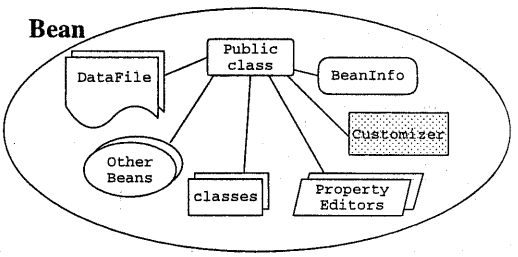

図 4: Java Beans 構成

Java Beansの特徴は多数あるが,特に本稿で注目する特徴を以下に挙げる.

BeanInfo クラス:

Bean自身の情報や,Beanの持つメソッド,イベントなど,そのBeanに関する情報を管理するクラスである.このクラスはそのBeanの開発者によって外部へのインタフェースとして製作される.

Introspection 機構:

あるBeanに着目し,そのBeanのBeanInfoクラスを取得する機構である. BeanInfo自体の定義は任意であるので,そのBeanInfoクラスが存在しない場合には,Beanを独自に解析し,そのBeanのBeanInfoクラスを動的に作成する.

これらの特徴を活用し2.2章で述べた(P1),(P2)の問題点を解決する.

3.2 Java Beansのエージェント化

2.2章の問題点を解消するために,ADIPSフレームワークに基づいたエージェント化支援機能を考案した.この機能は,問題点となっている(P1)プロセスを制御するためのインターフェースに関する知識記述が,プロセス毎に異なっている為,エージェント開発者の負担が大きい,および(P2)ベースプロセスを直接TPMと接続し,制御するという制約上,ベースプロセスが外部とのインタフェースを持たない場合,プログラムの改造などが必要となる,の二つの問題点をJava Beansをベースプロセスとして利用することを前提に解消する機能である.

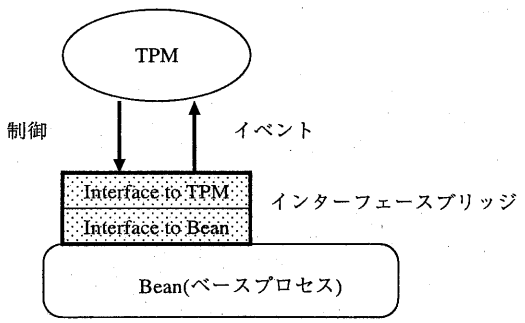

図5: インターフェースブリッジ概念図

3.1章で説明したように,Java Beansは再利用の促進を目的とした独立性の高いコンポーネントである.Beanを利用するプログラムはBeanInfoクラスに記述されているBeanの外部へのインターフェースの情報を基にBeanの制御を行う.また,このBeanを利用する開発者はBeanのプログラムのソースコードやリファレンスを必要とすることなく,BeanInfoの情報を基に,このBeanを利用することが可能である.

従って上記のBeanの特性を利用し,図3の既存プロセス内部に埋め込んでいたTPMとのインターフェース部分を,TPMからBeanの外部インターフェースへのブリッジとなるようにする.このTPMのインターフェースから,Beanの外部インターフェースのブリッジとなる部分(インターフェースブリッジ)を,外部部品として図5のようにBeanの外へ取り出す.これにより,既存プロセスであるBeanを変更すること無くTPMと接続可能となる.

このインターフェースブリッジを自動的に生成するのがエージェント化支援機能である.この機能によりエージェント開発者はBeanのプログラムを変更すること無くエージェントのベースプロセスとしてBeanを利用できる.また,インターフェースブリッジとなる部分を自動生成すると同時に,このブリッジ部分を制御するための知識記述の雛型(TPM+DKの一部)を自動生成する.これにより,エージェント開発者はBeanをエージェント化することが容易になり負担が軽減される.

4 エージェント化支援機能の設計と実装

4.1 エージェント化支援機能の設計

エージェント化支援機能は,入力としてJava Bean(ABean)を受け取り, 3.2章で述べたインタフェースブリッジ(ABeanAdapter)を生成する.また,Adapterの制御はTPMが行うが,TPMもAdapterクラス生成時に同時に生成される.更にこのTPMを制御するための知識をDKの一部分として生成する.これらの知識を知識ファイルの雛型としてエージェント記述言語であるADIPS/Lの形式に従い出力する.図6にエージェント化支援機能の概念図を示す.

BeanInfoクラスの情報はIntrospection機構を利用し取得することができる.そこで,Introspection機構を活用し,Beanの制御に必要となるMethod名,引数の型,戻り値,また,Beanから発生し得るイベントなどの情報を取得する.

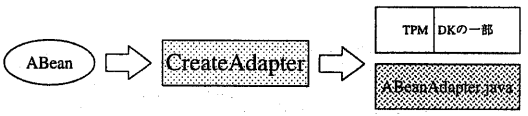

図 6: エージェント化支援機能概念図

対象となるベースプロセスがJavaプログラムの場合adips97.BPというクラスを継承したクラスがTPMとのインターフェースとして働く.このTPMとのインタフェースと上記のBeanの情報を基に,図7に示すように,インタフェースブリッジとして働くAdapterクラスのソースコード(ABeanAdapter.java)を自動生成する.TPMからAdapterクラスに制御を行うと,それに従いAdapterクラスがBeanの制御を行う. また,Beanからイベントが発生するとAdapterクラスがそのイベントを受取り,TPMにイベントを伝える.

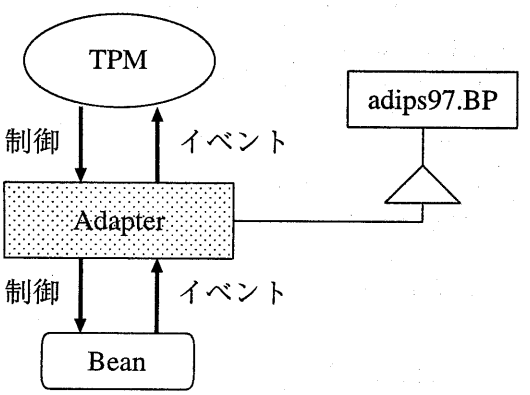

図 7: Adapter の設計

4.2 エージェント化支援機能の実装

4.1章の設計に基づきエージェント化支援機能をCreateAdapterとして実装した.CreateAdapterにより,ベースプロセスであるABeanを入力としインタフェースブリッジであるAdapterクラスのソースファイル(ABeanAdapter.java) ,及びAdapterの制御知識の雛方が生成される.実装にはJava言語を使用し,クラスは10クラス,コード量は約4,000行であった.また,Java Beansの特性を活かし, Beans自体をネットワーク上からロードすることを可能にした.以下に例として生成されたjavaのプログラムコード及び,知識記述の雛型をしめす(図9,10).

5 実験

CreateAdapterにより生成されたJavaプログラム及び,知識ファイルの雛型を用い,エージェントの開発を行った.実験で利用したJava Beansは横軸を時間とし縦軸に値を入力することでパルスを表示するPulseGraph Bean,及びJava Virtual Machine(JVM)の空きメモリ率を観測するMemoryChecker Beanである. これらのBeanをエージェント化し,JVMの空きメモリ率を表示する簡単なアプリケーションを構築した.

図8の後方のウィンドウがエージェントモニタである.またその前のグラフが実験で構築したアプリケーションの概観である.

エージェント化にあたり,実際に必要であった記述は数行であった.エージェント化支援機能が無い場合,これらの知識を全て記述する必要がありあきらかに知識記述が容易になっている.また,Beansのプログラムに手を加えること無くエージェント化できることにより,エージェント開発とベースプロセスの開発が完全に分離できる.これらのことより2.2章の問題点が解決されたことが確認できたと言える.

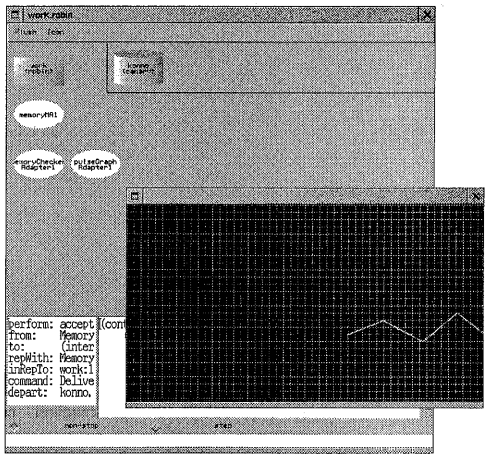

図 8: 実験

6 まとめ

本稿では,ADIPS フレームワーク上でプリミティブエージェントを開発する際, エージェント化の対象となるプロセス固有のインタフェースに関する知識を書くことがエージェント開発者の負担になっていることを述べ, その解決策として Java Beans をベースプロセスとして利用することで, エージェント化が容易になるを示した. 今後,Java Beans のみをベースプロセスとして利用するフレームワークを構築することで, よりエージェント開発の負担が軽減すると考えられる.

参考文献

[1] Shiratori, N., Sugawara, K., Kinoshita, T. and Chakraborty, G.: Flexible Networks: Basic concepts and Architecture, *IEICE Trans. Comm.*, Vol. E77-B, No. 11, pp.1287-1294,1994.

[2] N. Shiratori, T. Suganuma,S. Sugiura, G. Chakraborty, K. Sugawara, T. Kinoshita and E.S.Lee.: Framework of a flexible computer communication network, Computer Communications, Vol,19, pp.1268-1275,1996.

[3] 藤田 茂, 菅原 研次, 木下 哲男, 白鳥 則郎: 分散処理システムのエージェント指向アーキテクチャ, 情報処理学会論文誌, Vol.37, No.5, pp.840-852,1996.

[4] S. Fujita, H. Hara, K. Sugawara, T. Kinoshita and N. Shiratori: Agent-Based Design Model of Adaptive Distributed System, Applied Intelligence, Vol.9,No.1,1998.

[5] Smith, R.G.: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, IEEE Trans. Comp., Vol.C-29, No.12, pp.1104-1113 (1980).


```
(control (bp-input  method getmaxvalue ( getMaxValue (java.lang.String))))
(control (bp-input  method getpreferredsize ( getPreferredSize (java.lang.String))))
(control (bp-input  method getgridsize ( getGridSize (java.lang.String))))
(control (bp-input  method isshowgrid ( isShowGrid (java.lang.String))))
(control (bp-input  method getvalues ( getValues (java.lang.String))))
(control (bp-input  method isdynamicscaling ( isDynamicScaling (java.lang.String))))
(control (bp-input  method getmaxpoints ( getMaxPoints (java.lang.String))))
(control (bp-input  method getgridcolor ( getGridColor (java.lang.String))))
(input   (bp-input  method setmaxvalue ( setMaxValue (java.lang.String))))
(output  (bp-output method setmaxvalue ))
(input   (bp-input  method setgridsize ( setGridSize (java.lang.String))))
(output  (bp-output method setgridsize ))
(input   (bp-input  method setshowgrid ( setShowGrid (java.lang.String))))
(output  (bp-output method setshowgrid ))
(input   (bp-input  method setvalues ( setValues (java.lang.String))))
(output  (bp-output method setvalues ))
(input   (bp-input  method setdynamicscaling ( setDynamicScaling (java.lang.String))))
(output  (bp-output method setdynamicscaling ))
(input   (bp-input  method setmaxpoints ( setMaxPoints (java.lang.String))))
(output  (bp-output method setmaxpoints ))
(input   (bp-input  method setgridcolor ( setGridColor (java.lang.String))))
(output  (bp-output method setgridcolor ))
(input   (bp-input  method newvalue ( newValue ( java.lang.String ))))
(output  (bp-output method newvalue ))
(input   (bp-input  method clear ( clear (  ))))
(output  (bp-output method clear ))
```

図 9: 生成された知識記述 の一部

```
/** Bean のパブリックなメソッドを起動するためのメソッド newValue */
public Object newValue ( String arg ) {
  Object returnVal = null;
  Object[] prm = { new Long(arg) };
  try {
    returnVal = ((Method)methodMapping.get("newValue")).invoke(bean,prm);
  } catch (Exception e) {
    System.err.println(e);
  }
  return returnVal;
}

/** Bean のパブリックなメソッドを起動するためのメソッド clear */
public Object clear ( ) {
  Object returnVal = null;
  Object[] prm = {  };
  try {
    returnVal = ((Method)methodMapping.get("clear")).invoke(bean,prm);
  } catch (Exception e) {
    System.err.println(e);
  }
  return returnVal;
}

/** 外部からプロパティをセットするためのメソッド setValues */
public void setValues ( String[] element) {
  try {
    Object[] prm = new Object[element.length];
    for (int i=0; i<prm.length; i++) {
      prm[i] = new Long (element[i]);
    }
    if (prm != null) ((Method)methodMapping.get("setValues")).invoke(bean,prm);
    else tpm.raiseEvent("NoSuchProperty");
  } catch (Exception e) {
    System.err.println(e);
  }
}

/** 外部からプロパティをゲットするためのメソッド getValues */
public String getValues () {
  Long[] prop = null;
  Object[] prm = {};
  try {
      prop = ( Long[] )((Method)methodMapping.get("getValues")).invoke(bean,prm);
  } catch (Exception e) {
    System.err.println(e);
  }
  StringBuffer buf = new StringBuffer();
  for (int i=0; i<prop.length; i++) {
    buf.append(prop[i].toString()+" ");
  }
  return buf.toString();
}
```

図 10: 生成された Java プログラムの一部