特集論文 「エージェント」

ビヘイビア記述に基づく自己適応システム実装フレームワークの提案

An Implementation Framework for Self-adaptive Systems Based on Agent Behaviors Description

中川 博之 Hiroyuki NAKAGAWA
電気通信大学
The University of Electro-Communications
nakagawa@is.uec.ac.jp, http://www.is.uec.ac.jp/staff/217.html

大須賀 昭彦 Akihiko OHSUGA
（同　上）
ohsuga@is.uec.ac.jp, http://www.ohsuga.is.uec.ac.jp/~ohsuga/

本位田 真一 Shinichi HONIDEN
国立情報学研究所，東京大学
National Institute of Informatics, The University of Tokyo
honiden@nii.ac.jp, http://research.nii.ac.jp/~honiden/

keywords: self-adaptive systems, agent platforms, component-based architecture, implementation frameworks

Summary

The complexity of current software systems requires the ability to adapt at run-time, and the development of self-adaptive systems is one of the recent challenges for realizing dynamic adaptation. In this paper, we focus on the adaptation based on the components connection as a fundamental adaptation, and present our implementation framework for constructing self-adaptive systems on the basis of an agent platform. We reinforce the agent platform by adding some application program interface for behavior cooperation and its autonomic activation, and connect the behaviors with components in self-adaptive systems. We also introduce an implementation guideline: a way to identify the responsibilities for control loops and implementation patterns for these responsibilities. We also demonstrate the effectiveness of our framework and guideline through the results from our implementation experiments and show how they can be used to construct self-adaptive systems by using agent platforms.

1. は じ め に

近年ソフトウェアの大規模・複雑化に伴い，環境の変化に対しても人が介在すること無く，動的に環境に適応するソフトウェアが求められるようになっている．特に，環境の変化に対しても各種サービスの継続的な提供が必要となる大規模サーバや，環境の変化を扱うことが前提となるユビキタスコンピューティングの分野においては，環境変化への動的な適応が強く求められている．このような背景から，**自己適応システム**（self-adaptive systems）[SAS 07, Cheng 08] と呼ばれる，環境に適応し，みずからの動作や構成を変化させることのできるシステムの実現に対する期待が高まっている．

自己適応システムに対するアーキテクチャとしては，Kramer らの 3 層アーキテクチャモデル [Kramer 07] などのコンポーネント構成による適応メカニズムの実現が有効と考えられている．ただし，その構築手段や実装フレームワークに関しては，いくつかの研究がなされているものの，汎用的な提案には至っていない．特に，自己適応システムには**制御ループ**（control loop）[Oreizy 99, Dobson 06, Kephart 03] と呼ばれる，システム自らが動的に状況を分析し，適応のための振舞いや構成変更を実現する意思決定メカニズムが組み込まれている必要があるが，これを実現するためのアプローチについて言及されているものは少ない．

そこで本研究では，制御ループ内の各アクティビティに対する責務に着目し，関与するコンポーネントに対してそれぞれの責務を割当てるとともに，並行動作するこれらのコンポーネント間で制御ループが実現可能な自己適応システムの実装基盤フレームワークを提案する．本研究では特に，並列する複数のプロセスをビヘイビアとして記述することのできるエージェントプラットフォーム JADE [Telecom Italia] を拡張利用した実装フレームワークを実現する．ただし，現段階の JADE ではビヘイビア記述により並列する複数のプロセスを実現可能であるものの，その連携に関する記述は十分ではないため，コンポーネントの並行動作を実現するために JADE に拡張クラスを導入する．また，制御ループを実現するために，制御ループ内の各責務の決定法と各責務の実装を支援する実装パターンを導入することで，自己適応システムの実

装ガイドラインを提供する．本手法により，アーキテクチャモデルからの系統的なシステム実装が可能となるとともに，コンポーネント連携による制御ループの構築が容易になることから，様々な適応を同時に扱うことのできる自己適応システムの系統的な構築が期待できる．

本論文は以下のような構成となっている．まず2章で，自己適応システムの実装フレームワークに求められる要件と，本研究で拡張利用するJADEの概要を述べる．続く3章では，本研究で提案する実装フレームワークとして，制御ループの責務決定法と，JADEの拡張，さらに制御ループを実現するための実装パターンについて説明する．4章では本研究で実施した自己適応システムを想定した2種類のシステム構築実験の結果を示し，5章で実験結果に基づいた本手法の有効性と適用範囲について論じる．最後に6章で関連研究について言及した後，7章でまとめを述べる．

2. 自己適応システム

2·1 実装フレームワークに求められる要件

自己適応システムの実現に向けては，様々なアーキテクチャモデルが提案されているが，Kramerらの3層アーキテクチャモデル [Kramer 07] やOreizyらのモデル [Oreizy 99]，Garlanらの Rainbow [Garlan 04] に代表されるように，複数のコンポーネントの接続によるシステム構成とその接続の動的変化による環境への適応アプローチを取るものが多い．しかしながら，自己適応システムの実現に関しては，ドメイン固有の構築例が報告され始めている一方で，自己適応システムを実装するための汎用的な実装フレームワークや自己適応システム構築に有効な実装方針が確立されるには至っていない．

この大きな要因として，制御ループ（control loop）[Oreizy 99, Dobson 06, Kephart 03] の構築手段を提供するフレームワークが少ないことが挙げられる．制御ループとは，環境やシステム自身の状態に関する情報の収集（Collect），得られた情報を基にした分析（Analyze），分析結果による振舞いや構成変更のための意思決定（Decide），変更後の実行（Act）の4つのプロセスが繰り返される制御構造を指し，自発的な適応を実現するために必要なメカニズムである．コンポーネントの結合により適応を実現するためには，環境の変化を検知するセンサ，環境に作用を与えるアクチュエータ，そしてこれらを制御するための制御ループを実現する必要があるが，制御ループは並行に動作するコンポーネントとそれらの結合により実現する必要がある．つまり，コンポーネントを実装する際には，これらの4つのプロセスに対して，どのコンポーネントが責務を持つかを決定し，これらの責務を過不足なくコンポーネント上に実装しなければならない．しかしながら，既存のアーキテクチャモデルは制御ループを明示的に示したものは少なく，開発者は適切なスコープで作用する制御ループを自身で設計・実装するか，システムを通して与えられた制御ループ機構を利用することとなる．しかし実際には，例えばRainbow [Garlan 04] のように1つの制御ループによりシステム全体の適応を実現することになると，コンポーネント数の増加によりその記述が急激に難しくなる．

また，自己適応システムを構築するにあたっては，新規にコンポーネントをすべて実装するのではなく，既存のモジュールを利用した拡張的な開発が想定される場合が多い．従って，自己適応システムの実装フレームワークはこれらの既存モジュールとの連携も考慮されたものでなければならない．

これらの背景から，本研究では自己適応システムを実現するための実装フレームワークに求められる要件として以下を定義する．

［定義］自己適応システムの実装フレームワークに求められる要件

- **要件1**（コンポーネントの並行実行）：複数のコンポーネントが並行動作可能な実行環境が提供されている
- **要件2**（制御ループの実現容易性）：複数のコンポーネントにまたがる制御ループの機能を容易に実現することができる
- **要件3**（既存モジュールとの親和性）：すでに存在するモジュールを組込んだコンポーネント実装が可能である

2·2 JADEを用いたコンポーネント制御の実現

本研究では，各コンポーネントの実装フレームワークとして，エージェントプラットフォームであるJADE [Telecom Italia] の拡張利用を検討する．JADEはJava言語によるエージェント構築が可能なエージェントプラットフォームであり，複数のビヘイビアを記述することで，エージェントが動作するために必要となる複数の並行タスクを記述可能な開発・実行環境を提供するものである．

JADEではビヘイビアを実装するためにBehaviourクラスとその派生クラスが提供されている．開発者はビヘイビアの利用形態に応じて適切な派生クラスを選択し，その派生クラスを継承するビヘイビアを実装することで，エージェントの多様な振舞いを定義することができる．図1にJADE上でのビヘイビア記述例を示す．ビヘイビアの定義には通常，繰り返し実行する処理を記述する *action* メソッドと，終了条件を記述する *done* メソッドを利用する．この例では，標準的なAPIを提供するSimpleBehaviourを継承し（8行目），繰り返し実行する処理として自身の名前の出力とカウント変数のインクリメントを（16〜19行目），終了条件としてカウント変数値のチェック（20行目）を記述している．ビヘイビアは *addBehaviour* メソッドにより，エージェントの実行プロセスとして登録・起動され（3, 4行目），この例では2つのビヘイビアインスタンスが終了条件が満たされるまでそれぞれ並行に

```
 1: public class SampleAgent extends Agent{
 2:   protected void setup(){
 3:     addBehaviour(new Printer(this,"p1"));
 4:     addBehaviour(new Printer(this,"p2"));
 5:   }
 6: }
 7:
 8: class Printer extends SimpleBehaviour{
 9:   int n=0;
10:   String name;
11:
12:   public Printer(Agent a, String st){
13:      super(a);
14:      name = st;
15:   }
16:   public void action(){
17:     System.out.println("I am "+name+".");
18:     n++;
19:   }
20:   public boolean done(){return n>=5;}
21: }
```

図 1 JADE におけるビヘイビア記述例

実行される．

このように，JADE は複数のビヘイビアを記述することでシステムに割当てられるべき各プロセスの並行実行が実現可能であることから，コンポーネントをビヘイビアにより実装することで，2·1 節で定義した実装フレームワークに求められる要件の 1 つ「コンポーネントの並行実行」を満たすことができると考えられる．しかしながら，現状の JADE では次に挙げるビヘイビア制御の難しさにより，各コンポーネントに分離される制御ループ機能の実現は容易であるとは言えない．

ビヘイビアの自発的起動：状況に応じてみずから振舞いや構成を変化させる必要のある自己適応システムにおいては，状況の変化に応じてコンポーネントが自発的に動作を駆動する必要があり，実装に利用するプラットフォームにはコンポーネントが自発的に起動するメカニズムが求められる．しかし JADE では，ビヘイビアの終了条件を指定できる *done* メソッドが提供されている一方で，ビヘイビアの起動に関しては条件を指定するための API が提供されていない．

ビヘイビアの連携：自己適応システムにおいては，環境や自らの振舞いを監視するプロセスや目的の遂行状況を管理するプロセス，振舞いを実現するプロセスなど複数のプロセスが相互に連携することでシステムが構成される．これに対し，JADE では複数ビヘイビアによりプロセスの並行実行を実現することができるが，他ビヘイビアのタスク実行状態を能動的に確認する手段は提供されていない．また，他ビヘイビアの動作を制御する手段として，ビヘイビアを強制的に中断させる *block* メソッドが提供されているが，処理を中断する際の適切な退避処理などを記述することができない．その結果，単に JADE 上でビヘイビアを用いてコンポーネントを実装すると，二つ以上のコンポーネントが同一コンポーネントのサービスを利用する際にサービス提供側コンポーネントで競合が発生したり，環境の変化などによりコンポーネントのサービス利用を中断した後に再開した際に，適切な退避処理が記述できないことから誤動作を引き起こすといった，コンポーネント連携による問題が発生することになる．

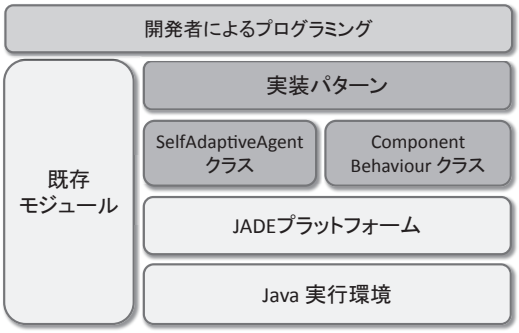

図 2 提案手法のプログラミングモデル

自己適応システムの実装に JADE を利用するためには，これらの問題は解消されていなければならない．

3. ビヘイビア記述による自己適応システムの実装

本研究では 2·1 節で定義した要件を満たす実装フレームワークとして，JADE を拡張利用した実装フレームワークと，本フレームワークを用いた実装ガイドラインを提案する．本手法では，まずコンポーネントの接続関係から，制御ループ実現のために各コンポーネントに割り当てられる責務を同定し，その後，本研究で提供する拡張クラスと制御ループ実現のための実装パターンを利用することで，JADE 上で自己適応システムの構成要素となるコンポーネントを実装する．提案手法におけるプログラミングモデルは図 2 に示すとおりであり，本章では以降，文献 [Morandini 08] などでも自己適応システムの構築例として用いられている清掃ロボットの構築を例に挙げ，提案手法による実装プロセスを説明する．

3·1 責務の同定

提案手法では，3 層アーキテクチャモデルにおいてもアーキテクチャ記述として用いられているコンポーネント接続図をアーキテクチャ構成図として用い，本構成図中で用いられる拡張 Darwin モデル [Hirsch 06] の接続形態から，各コンポーネントが持つべき制御ループに対する責務を同定する．図 3 は拡張 Darwin モデルを用いて記述した清掃ロボットのアーキテクチャ構成図である．拡張 Darwin モデルの特徴として，各コンポーネントがサービス提供ポートとサービス要求ポートを持ち，それらを接続することでコンポーネントの相互関係を定義する点と，mode と呼ばれる状態を外部に可視化する変数を持つ点が挙げられる．例えば図 3 では，清掃ロボット（Cleaning

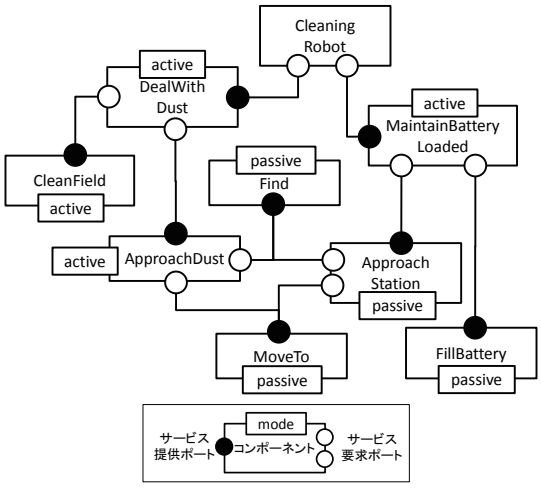

図 3 清掃ロボットのアーキテクチャ構成図

Robot）はごみを処理する DealWithDust とバッテリーを管理する MaintainBatteryLoaded の 2 つのコンポーネントのサービスにより構成され，DealWithDust コンポーネントはそのサービスを実現するために，ごみに近づく ApproachDust と床を清掃する CleanField の 2 つのコンポーネントのサービスを利用することが分かる．

提案手法では，まず，拡張 Darwin モデルの 2 種類のポートから制御ループを実現するために各コンポーネントに割当てられるべき責務を決定する．サービス提供ポート（図 3 で●として表現されているポート）は，他コンポーネントに自身のコンポーネントのサービスを提供するためのポートであり，自己適応システムにおいては，センサ・アクチュエータのサービスやそれらを包含したサービスを提供する側の役割を担っていると考えることができる．従って提案手法では，同ポートを持つコンポーネントには，該当するサービスの実行（Act）の役割と共に，収集（Collect）すべき情報を提供する役割を付与する．一方のサービス要求ポート（図 3 で○として表現されているポート）は，他コンポーネントのサービスを利用するためのポートであり，自己適応システムにおいては，サービスを利用し統合する，つまり制御する側の役割を担っていると考えることができる．従って提案手法では，同ポートを持つコンポーネントには，該当するサービスの状況を収集（Collect）し，分析（Analyze）した結果から振舞いの変化を決定（Decide）する役割を付与する．

3·2 拡張ビヘイビアによるコンポーネント実装

提案手法では，アーキテクチャ構成図に基づいて各コンポーネントに対応するビヘイビアを JADE 上で実装する．本研究では，自己適応システムを実装するために JADE を拡張し，Agent クラスを継承する SelfAdaptiveAgent クラスと，SimpleBehaviour クラスを継承する ComponentBehaviour クラス（図 4）を導入する．特にコンポーネント実装に関しては，開発者は ComponentBehaviour クラスを継承したコンポーネント，つまり JADE におけるビヘイビアを実装することで，2·1 節で定義した要件を満たすコンポーネントの実現が可能となる．以下，ComponentBehaviour クラスの概要について説明する．

まず，ComponentBehaviour クラスは拡張 Darwin モデルの特徴の 1 つである mode を変数として持ち，各コンポーネントの状態を mode 変数を通じて外部に公開する．しかしながら，単に拡張 Darwin モデルに従って mode 変数を導入するだけでは，コンポーネント間で制御ループを実現するための制御インタフェースとしては十分ではない．そこで本研究では，コンポーネントの制御を容易にするために，図 5 に示す状態遷移をコンポーネントのライフサイクルとして導入し，状態遷移のためのメソッドを導入する．このライフサイクルは，筆者らが文献 [中川 09] で定義したライフサイクルをもとにビヘイビア記述による実装を考慮して拡張したものであり，まず，各ビヘイビアにより実装されたコンポーネントは活性状態（active），待機状態（waiting），達成状態（achieved），達成不能状態（not achieved）の 4 つの状態を mode 変数の値として取り得る．また，図 5 中の *activate*，*passivate* メソッドはそれぞれコンポーネントの状態を活性状態あるいは待機状態へと遷移させるメソッドであり，この 2 つのメソッドによりコンポーネントの動作を制御する．さらに，*create*，*finalize* メソッドはそれぞれ，コンポーネントを生成して待機状態にするメソッド[*1] と，コンポーネントを終了させるメソッドである．これらの拡張により，状態遷移制御がメソッドとして分離され，状況の変化に即応するコンポーネント制御手段が外部のコンポーネントに提供されることとなる．

さらに提案手法では，コンポーネントの自発的起動メカニズムも ComponentBehaviour クラスに導入する．通常 JADE ではビヘイビアの処理内容を *action* メソッド内に記述するが，ComponentBehaviour クラスにおいてはライフサイクルに連携した状態遷移に関する条件分岐を *action* メソッド内に埋め込み，実質の処理内容部を記述するメソッドとして *perform* メソッドを新たに定義する．これにより，コンポーネント実装時にはコンポーネントが活性化したときの処理内容のみを *perform* メソッドに切り出して記述することができる．その一方で，活性条件を記述するための *activateCondition* メソッドを導入し，本メソッドに記述された判定式が `true` を返す場合に，*perform* メソッドが実行されるよう関連付ける．これにより，開発者は *activateCondition* メソッドと *perform* メソッドをオーバーライドすることによりコンポーネントが活性化される条件と活性状態における処理内容を定義することが可能となり，結果として自発的な起動が実現される．

∗1 実体は JADE の *addBehaviour* メソッドである．

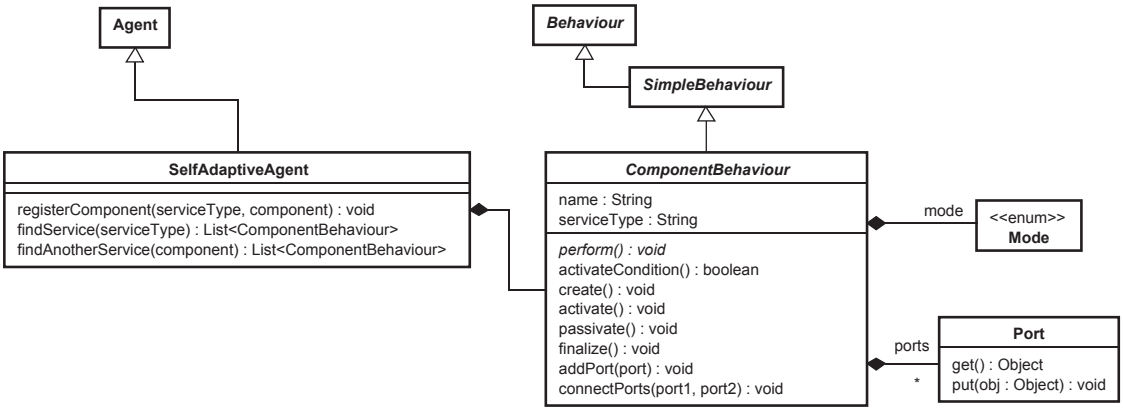

図 4 提案手法における Agent クラスと Behaviour クラスの拡張（メソッドは追加した主要 API のみ記載）

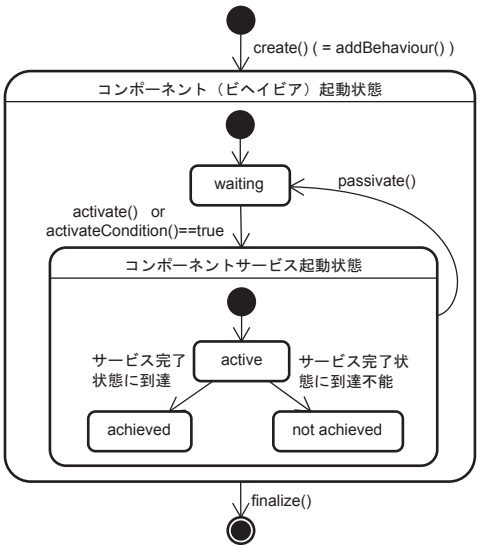

図 5 ComponentBehaviour のライフサイクル

3.3 実装パターン

提案手法ではさらに，制御ループの機能をコンポーネント上に実装するために，事前に抽出した制御ループの各責務に対応する 3 種類の実装パターンを導入する．以下，制御ループの各アクティビティに対応する実装パターンを示す．

§1 Collect パターン

自己適応システムにおいては環境情報とシステムの内部状態に関する情報の 2 種類の情報を扱う必要がある．このうち，環境情報の収集に対しては，センサモジュールの出力を収集する形態となり，センサのサービスを提供するコンポーネント内に情報取得のための処理を記述する．提案手法における Collect パターンは，システムの内部状態に関する情報のやり取りに対して定義する．システムの内部状態としては，本フレームワークにおいては，各コンポーネントの状態と各コンポーネントのサービス実行結果が該当するが，これらの情報の格納・取得手段を次のように実現する．まず，コンポーネントの状態収集に関しては，拡張 Darwin モデルに従い，サービス提供側コンポーネントで状態遷移の時点で mode 変数に変更後の状態値をセットし，サービス要求側コンポーネントで，

< コンポーネント名 >.mode

という形態で状態を参照（収集）する．一方のサービス実行結果の収集には，拡張 Dawin モデルが持つポートの概念を利用する．サービス提供側がポートに実行結果を *put* メソッドにより格納し，サービス要求側はポートに結果が存在すれば *get* メソッドにより取得する．

§2 Analyze & Decide パターン

状況の分析（Analyze）と意思決定（Decide）はサービス要求側のコンポーネントが果たすべき責務であり，提案手法ではこれらを統合した Analyze & Decide パターンを提供する．制御ループでは，情報収集（Collect）により得られた結果をもとに状況を判断・分析し，適応のための変化を決定するため，Analyze & Decide パターンとして以下の構造をもつ if-文を用いる．

```
if (< 分析結果 >){
    < 適応手段 >
}
```

ここで < 分析結果 > には，Collect パターンにより得られた情報に基づく分析結果を記述する．例えば，ごみへ近づいた状態であるかを分析するには，ごみへ近づくというサービスが完了しているかどうかを確認する

```
approachDust.mode==Mode.ACHIEVED
```

といった記述を用いる．一方，< 適応手段 > に対しては，主にサービス提供コンポーネントの動作を制御する *activate*，*passivate* メソッドを用いたコンポーネント動作制御や，SelfAdaptiveAgent が提供する findAnotherService メソッドを利用して同様のサービス提供コンポーネントを取得して，コンポーネントを生成，切替えるといった処理を記述する．

§3 Act パターン

変更後の実行（Act）は，サービス提供側の責務である．Act パターンは，適応に伴うコンポーネントのサービス提供開始と退避処理の記述手段に該当するが，実はこれらはそれぞれ ComponentBehaviour クラスの *perform* メ

ソッドと *passivate* メソッドの呼び出しにより実現可能である．従って Act パターンは，サービス内容を *perform* メソッドに，退避処理を *passivate* メソッドに記述するというものになる．もし環境に対して影響を与える動作があれば，同メソッド内に既存のアクチュエータモジュールを駆動する記述を加える．

提案手法では，以上の実装パターンを用いて各コンポーネントを実装するが，上に述べたように，既にセンサやアクチュエータなどの機能を実現するモジュールが存在する場合には，そのモジュールをラッピング（wrapping）する形態でサービス提供コンポーネントを実装する．

4. 実　　験

提案手法の有用性を検証するために，提案手法に従って清掃ロボットと Web サーバ管理システムの 2 種類のシステム構築を想定した実装実験を実施した．本章では実験内容とその結果を示し，提案手法の有効性を評価する．

4·1 実験 1：清掃ロボット構築実験

実験 1 における実験内容は大きく 2 種類に分類される．まず，提案手法による自己適応システムの実装が可能であることを確認するために，提案手法に従って自己適応システムを想定したプログラムを実際に構築し，その挙動とコンポーネントの記述方法を分析した（実験 1-1）．続いて，同様の機能を持つプログラムを JADE が提供する既存のビヘイビアにより実装した場合と，拡張 Darwin のコンポーネントモデルに従って実装した場合の 2 通りの方法により構築し，これらと実験 1-1 で構築したプログラムを比較することで提案手法の有効性と実装効率を評価した（実験 1-2）．

実験 1 において対象とするシステムは，自身のバッテリー残量に配慮しながら与えられたフィールドに散在するごみを清掃する清掃ロボットである．本実験では，清掃ロボットを取り巻く環境をシミュレータ上にあらかじめ構築し，また，指定したオブジェクトを検出するセンサモジュールや，指定した対象に対して移動する駆動モジュールをあらかじめ実装し，これらの機能をコンポーネントから呼び出すという実装方法で清掃ロボットを構築した．なお，本実験では，清掃ロボットのアーキテクチャ構成として図 3 の構成を用いた．実験では，ごみとバッテリーステーションのいくつかの配置パターンを用意し，それぞれにおいて清掃ロボットがごみを収集するプロセスを観測した．図 6 は実験で用いた配置パターンの 1 つである．

§1 実験 1-1：実現可能性

まず，提案手法にもとづいて構築した清掃ロボットの挙動を観測した．図 6 の配置パターンに対する実験結果として，清掃ロボットが出力したログの一部を図 7 に示す．図 7 からは，清掃ロボットがごみのある座標 (5, 5) に向かって移動中（79〜84 行目）にバッテリーが低下したため（85 行目），バッテリー管理の責務を持つ MaintainBatteryLoaded コンポーネントが活性化し（86 行目），座標 (2, 3) に存在するバッテリーステーションへの移動へと動作を切り替えたことが分かる（87〜99 行目）．その後，バッテリーステーションに到着して充電した後に（100〜102 行目），ごみ収集を再開し（103 行目），バッテリーステーションから最も近いごみを見つけて移動（104 行目〜）していることも確認できる．このように，清掃ロボットはバッテリーの残量がある間はごみの収集を続け，残量が少なくなるとバッテリーステーションへ移動するというプロセスを繰り返し，最終的にすべての配置パターンにおいてごみをすべて収集することが確認できた．

次に，提案手法に従って実装したコンポーネントの記述方法を分析した．まず実験結果から，清掃ロボットはごみ，

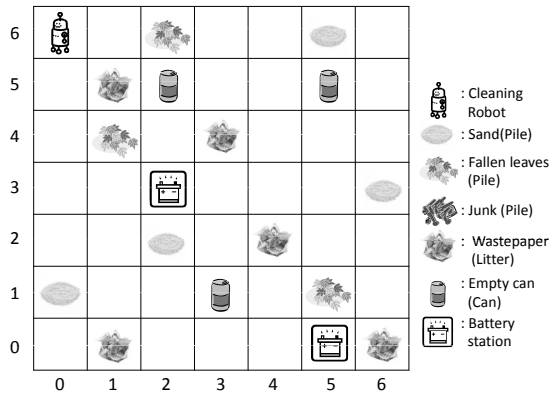

図 6 実験で用いた配置パターンの例

```
          ...
 79:    [java]   Dust is found at (5,5).
 80:    [java] Component: Move
 81:    [java]   Moved to (4,4).
 82:    [java]     Rest battery: 32
 83:    [java]   Moved to (4,5).
 84:    [java]     Rest battery: 28
 85:    [java] Battery is low ...
 86:    [java] Component: MaintainBatteryLoaded
 87:    [java] Component: ApproachStation
 88:    [java] Component: Find
 89:    [java]     Rest battery: 25
 90:    [java]   Station is found at (2,3).
 91:    [java] Component: Move
 92:    [java]   Moved to (4,4).
          ...
 98:    [java]   Moved to (2,3).
 99:    [java]     Rest battery: 9
100:    [java] Component: FillBattery
101:    [java]   Fill battery ...
102:    [java]   Battery is full
103:    [java] Component: DealWithDust
104:    [java] Component: ApproachDust
105:    [java] Component: Find
106:    [java]     Rest battery: 97
107:    [java]   Dust is found at (2,2).
108:    [java] Component: Move
109:    [java]   Moved to (2,2).
          ...
```

ごみに向かって移動
←バッテリー低下を検知
ステーションに向かって移動
充電
最寄りのごみに向かって移動

図 7 清掃ロボットの実行ログ（抜粋）

```
 1: public void perform() {
      ...
 8:   if(find.mode==Mode.ACHIEVED){
 9:     if(moveTo.mode==Mode.WAITING){
10:       try{
11:         nextDust = findPort.get();
12:         if(nextDust!=null){
13:           connectPorts(moveToPort,
                           pairOfMoveToPort);
14:           moveToPort.put(nextDust);
15:           moveTo.activate();
16:         }
              ...
```

図 8 実装パターンの適用例（ApproachDust コンポーネントの perform メソッド）

あるいはバッテリーステーションを見つけ，その後目標に向かっての移動を開始していることが確認できる．この一連の動作を実現するコンポーネントは図 3 のアーキテクチャ構成図における ApproachDust と ApproachStation コンポーネントである．図 8 は，ApproachDust コンポーネントの *perform* メソッドの一部を示したものであるが，ここでは周囲の物体を探索する責務を持つ Find コンポーネントの mode 値が “ACHIEVED” となったことを判定して（8 行目），移動の責務を持つ MoveTo コンポーネントを *activate* メソッドを用いて活性化させる（15 行目）という記述により，上記の動作を実現していることが確認できる．これは，本研究で提案する Collect パターンおよび Analysis& Decide パターンを利用した記述であり，またその直後に MoveTo コンポーネントの *perform* メソッドに記述された動作が開始することが確認できることからも，提案する実装パターンに従った実装により制御ループが実現可能であることが分かった．

一方で図 7 のログからは，バッテリーの残量が少なくなるとバッテリー管理を責務とする MaintainBatteryLoaded コンポーネントが活性化したことも確認できる（86 行目）．図 9 は，同コンポーネントの *activateCondition* メソッドの記述内容である．このように，提案手法では ComponentBehaviour クラスの同メソッドをオーバーライドすることで，バッテリー残量の監視によるコンポーネントの自動起動が可能である．さらに，図 7 のログからはバッテリーステーションへの移動の前後で，目標とするごみが座標（5,5）の空き缶から座標（2,2）の砂へと変わっていることも確認できる．これは，ロボットの移動を実現する MoveTo コンポーネントを制御するコンポーネントが ApproachDust から ApproachStation に切替わる際に，ApproachDust の *passivate* メソッドが呼び出されることで適切な退避処理が実行されたことによるものである．図 10 は ApproachDust コンポーネントの *passivate* メソッドの実装を示したものであるが，本メソッドではサービスを利用している Find，MoveTo ビヘイビアを待機状態（waiting）に遷移させ（4，7 行目），目標地点であるごみの座標をリセットしている（9 行目）ことがわかる．つまり，ComponentBehaviour クラスの *passivate* メソッドをオーバーライドすることで，コンポーネントをタスク実行途中で中断させる場合にも，待機状態への遷移前に適切な退避処理が記述可能であることが確認できたといえる．

```
1: public boolean activateCondition(){
2:    return (robot.getBattery()
3:       < CleaningRobot._BATTERY_THRESHOLD_);
4: }
```

図 9 MaintainBatteryLoaded コンポーネントの activate-Condition メソッド

```
 1:  public void passivate(){
 2:     super.passivate();
 3:     if(find!=null){
 4:        find.passivate();
 5:     }
 6:     if(moveTo!=null){
 7:        moveTo.passivate();
 8:     }
 9:     nextDust = null;
10:  }
```

図 10 ApproachDust コンポーネントの passivate メソッド

§2 実験 1-2：実装効率

続いて提案手法のコンポーネント実装効率と実装方法の有用性を評価するために，実験 1-1（Case 1 とする）で構築した清掃ロボットと同様の機能を持つロボットを，JADE が提供している既存ビヘイビアを利用した場合（Case 2）と，拡張 Darwin モデルに従った場合（Case 3）の 2 通りの手段によってそれぞれ実装し，得られた 3 種類の実装コードを比較した．具体的には，Case 2 ではごみ清掃とバッテリー管理のタスクをそれぞれ別ビヘイビアに分離し，それぞれのタスクを遂行するために，有限状態機械を実現する FSMBehaviour クラスを継承したビヘイビアを実装する形でビヘイビア階層を構築した．一方 Case 3 では，提案手法における ComponentBehaviour と同様に，拡張 Darwin モデルを想定したスーパークラスビヘイビアを新たに構築し，このビヘイビアを拡張継承することで，図 3 のアーキテクチャ記述に基づいたコンポーネントを実装した．ここで，拡張 Darwin モデルのスーパークラスビヘイビアには，mode による状態管理・参照機構は実装したが，外部から状態を遷移させる *activate*, *passivate* などのメソッドは実装していない．

表 1 は，本実験で構築した 3 種類の清掃ロボットを実装効率の観点から比較したものである．まず，提案手法（Case 1）と拡張 Darwin モデル利用（Case 3）の場合は，

表 1 清掃ロボット構築における 3 種類の実装手法の比較

評価項目	Case 1	Case 2	Case 3
- ビヘイビア数	8	16	8
- コード行数	626	849	830
- 分岐数 (if 文, switch 文 / 状態遷移)	47 (47/0)	60 (41/19)	73 (73/0)

図 3 のアーキテクチャ構成に従って清掃ロボットを構築したため実装ビヘイビア数は共に 8 であったが[*2]，従来手法（Case 2）においては 16 個のビヘイビアを実装する必要があった．これは FSMBehaviour の利用に伴うビヘイビアの細分化によるものである．FSMBehaviour は順序だったタスクを実現する標準的な手段として用いられるが，状態ごとにサブビヘイビアを実装する必要があり，また，これらのサブビヘイビアは再利用が難しいことから，結果として本実験においてもビヘイビア数の増加につながったといえる．

一方，プログラムの規模を示す指標となるコード行数（Lines of code）に関しては，提案手法による実装では従来手法よりも記述が効率化されているのに対し，拡張 Darwin モデルを適用した実装では従来手法とほぼ同等の結果となった．これは，単に拡張 Dawrin モデルを適用しただけでは，自身の実行処理の中断や再開のタイミングを各コンポーネントがそれぞれで判断する必要があり，これらの処理を追加で記述する必要があったことによるものである．また，他ビヘイビアを制御できないことから，本実験では特にごみ処理とバッテリー管理の責務を持つコンポーネントが共通利用する Find ビヘイビアや MoveTo ビヘイビアの制御が難しく，結果として記述すべきコード数が増加してしまったことも要因として挙げられる．

プログラムの複雑さや可読性に関与する分岐数に関しても同様で，拡張 Darwin モデルを適用した場合（Case3）は，他コンポーネントを制御出来ないことに起因して各コンポーネント内で状況に応じた振舞いの変更内容を記述する必要が生じることから，結果として条件分岐が増え，コードが複雑になる傾向にあることが分かった．一方，従来手法の場合（Case2）は，状態を表現するサブビヘイビア個々の記述内容が簡潔になることから if 文・switch 文の利用は少ないものの，これらのビヘイビア間の遷移を記述する必要があり，プログラム全体としての分岐数は少なくはなかった．これに対して，提案手法を用いた実装（Case1）では，多くは図 8 に示したような Analysis & Decide パターンにおける状況判断や，図 10 のような退避処理時における利用コンポーネントとの接続確認に用いられていることが分かった．

また，3 種類の清掃ロボットを実行した結果からは，Darwin 拡張モデル適用の清掃ロボットにおいて，実行状況によってはロボットがバッテリー切れを起こすことが分かった．これは，バッテリー残量低下により，ごみ処理を扱う DealWithDust コンポーネントから移動停止のシグナルが ApproachDust コンポーネントを経由して MoveTo コンポーネントに送られるが，このシグナルが到達する前に MoveTo がごみに向かって移動することでバッテリーが消費され，その結果バッテリーステーションに到達できなくなることによるものである．既存の構築手法では割込み制御の実装が困難であり，コンポーネントが多段構成になることでシグナルの遅延が発生し，開発者の意図しない現象を引き起こす可能性がある．その一方で，提案手法ではコンポーネントの状態遷移をメソッド呼び出しにより実現するため，たとえコンポーネントが多段構成になったとしても，このような遅延は発生しない．

*2 図 3 中の CleanerRobot は清掃ロボット本体を示したものであり，ビヘイビアとして実装はしていない．

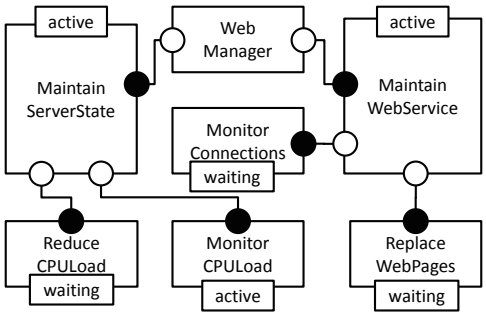

図 11 Web サーバ管理システムのアーキテクチャ構成図

4·2 実験 2：Web サーバ管理システム

本研究ではさらに，実世界における自己適応システム構築を想定した Web サーバ管理システム実装実験を実施し，その構築過程と動作結果を評価した．本システムでは 2 種類の適応を扱い，クライアント接続数が一定数を超えた場合に，コンテンツに含まれる画像を一時的に低解像度のものに切り替えるという適応と，Web サーバの CPU 利用率が過負荷になった場合に Web アプリケーションサービスを一時的に停止するという適応を実現する．本実験では，Web サーバ管理システムのコンポーネント接続関係として図 11 に示す構成を用いた．また，Web サーバとして Apache [Apache Software Foundation] を，同時接続数の増減には Apache に付属しているベンチマーク・ツール ab (Apache Bench) を利用した．

実験 2 の実験結果として，実装した Web サーバ管理システムの構成と実装パターンの適用数を表 2 に示す．まずシステム実装にあたっては，図 11 のアーキテクチャ構成をもとに制御ループの責務を同定し，MaintainServerState と MaintainWebService コンポーネントが関与する 2 つの制御ループを実装した．これらの制御ループに対しては，表 2 に示すように，抽出された責務に対して適時実装パターンを適用することで実現できることが確認できた．また，CPU・クライアント接続数のモニタリングや，Apache 再起動，Web ページ置き換えに関しても，表 2 に示すそれぞれのモジュールを利用し，センサおよびアクチュエータの責務を持つコンポーネントともにラッピングによる実装が可能であることが確認できた．

続いて動作実験では，実装した制御ループに従って，CPU の負荷が一定の割合を越えると，MaintainServer-

表 2 実装した Web サーバ管理システムの構成

実装プログラム	
– 構築コンポーネント数	6
– コード行数	484 行
パターン適用箇所	
– Collect パターン（mode 参照 / port 参照）	2 / 2
– Analyze & Decide パターン	6
– Act パターン（perform / passivate）	6 / 2
利用モジュール	
– CPU ログ読込みモジュール：Java（で実装したモジュール） → MonitorCPULoad コンポーネントで利用	
– Apache 停止・起動モジュール： XAMPP モジュール + MS-DOS バッチファイル → ReduceCPULoad コンポーネントで利用	
– アクセスログ読込みモジュール： Apache モニタリングモジュール + wget + Java → MonitorConnections コンポーネントで利用	
– Web ページ置換えモジュール：shell script + Java → ReplaceWebPages コンポーネントで利用	

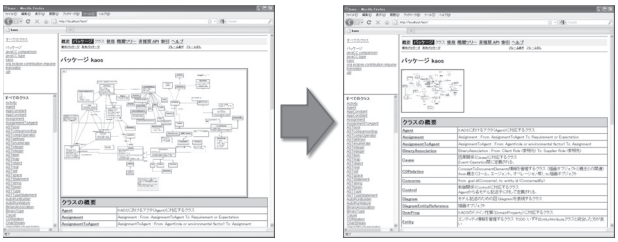

図 12 クライアント接続数増加に伴う Web ページの動的置換

State コンポーネントが MonitorCPULoad コンポーネントの収集する環境変化を検知して ReduceCPULoad コンポーネントを活性化し，同様にクライアント接続数が一定値を越えると，MaintainWebService コンポーネントが MonitorConnections コンポーネントの収集する環境変化を検知して ReplaceWebPages コンポーネントを活性化することが確認できた．図 12 は，本実験において確認したクライアント接続数の増加に伴う Web サイトの変化を示したものである．

5. 考　　察

4 章の実験結果をもとに，2·1 節で定義した要件に対して本手法を評価し，本手法の適用範囲について論じる．

5·1 コンポーネントの並行実行

提案手法では実行基盤として JADE を用いることで，ビヘイビアとして実装したコンポーネントの並行実行を実現している．また，実験結果が示すように，提案手法では，自発的起動と相互参照・制御の観点から JADE を拡張することで，コンポーネント並行実行のための記述能力を向上させ，自己適応システムにおけるコンポーネント実装を支援しているといえる．その一方で，提案手法で拡張利用する JADE においては実行の最小単位がビヘイビアであり，ラウンドロビンでビヘイビアの実行スケジュールが制御されるため，適用時には各コンポーネントの実行サイクルに留意する必要がある．例えば，非常に細かい時間単位でのスループット監視が要求される場合などは，情報の収集，つまり Collect のアクティビティが，保証すべき単位時間内に必ず実行されなければならない．このような制約に対しては，そもそも本フレームワークの適用が困難な場合もあるが，不要なコンポーネントを *finalize* メソッドにより終了させたり，コンポーネントの粒度を詳細化し，各スレッドの実行時間を短縮するなどの手段を検討する必要があると考えられる．

5·2 制御ループの実現容易性

提案手法では，自己適応システムのアーキテクチャ構成図から制御ループの適用範囲と各コンポーネントの責務を同定し，制御ループ実現のための拡張ビヘイビアクラスと実装パターンを提供することで，その実装を系統的に支援しているといえる．また，実験結果が示すように，制御ループの実装を開発者に委ねることにより，システム内に複数の制御ループを構築することが可能となり，関心事ごとに閉じた制御ループを構成できる．この特徴は，コンポーネント数の増加に対しても実装の複雑さを軽減することができるという点において有効であるといえよう．

5·3 既存モジュールとの親和性

提案手法ではラッピングによる既存モジュールとの連携を想定している．特に，提案手法においては制御ループの責務を複数のコンポーネントに分割することが可能であることと，コンポーネントに対して共通のインターフェースを定義していることにより，実験結果が示すように，既存モジュールに対しての機能単位でのコンポーネント化が可能となる．これは実装の観点からだけでなく，複数コンポーネントからのコンポーネントサービス共通利用の観点からも有益であると考えられる．

5·4 適 用 範 囲

最後に提案手法の適用範囲について論じる．まず，提案する実装プラットフォームは JADE 上で動作するため，その適用は JADE が動作する環境上に制限される．ただし，JADE は Java 言語により構築されたプラットフォームであり，PC 環境およびサーバ環境の Java である JavaSE, JavaEE が動作する環境上での適用が可能である．また，JADE には，リソースが制限されたデバイス用の Java である JavaME に対応した，Leap [Bellifemine 07] と呼ばれる軽量版が存在する．従って，本研究で拡張した各クラスを同様に軽量化することにより，Leap が動作する環境上での自己適応システムの実装も可能である．

一方で，提案手法のようなビヘイビア記述でコンポーネントを実装した場合，待機状態においてもコンポーネントのプロセスは継続することとなる．このため，5·1 節

での議論のように実行サイクルへの考慮が必要であるとともに，携帯端末等の主記憶容量に制限のある環境でコンポーネント数の多いシステムを実装する場合は，*finalize* メソッドを利用した同時起動ビヘイビア数の抑制を検討する必要がある．

また，提案手法では制御ループを複数実装することが可能であるが，これらの制御ループが関連した複雑な適応を扱う場合は，期待した適応が実現されることを保証するための検証プロセスが必要であろう．このような適応動作は再現性が低いため，検証プロセスとしてはテスティングによる網羅的なチェックではなく，形式手法 [中島 07] を用いた検証が有効であると考えられる．

6. 関 連 研 究

自己適応システムの実現に向けては，様々なアーキテクチャモデルが提案されている．Kramer らの 3 層アーキテクチャモデルは，ゴール管理層，変更管理層，コンポーネント制御層の 3 層により構成され，環境の変化のレベルに応じて異なった適応を実現するものである．3 層アーキテクチャモデルに関しては，実現に向けた関連研究が文献 [Sykes 08, Foster 07] などいくつか挙げられるものの，各層間での無矛盾性の保証などの研究課題も多く，本アーキテクチャを実現するための実装フレームワークは現段階では確立されていない．3 層アーキテクチャモデルの他にも，コンポーネントの結合による適応性実現のアプローチは，Oreizy，Taylor らの研究 [Oreizy 99] や Rainbow [Garlan 04] など多くの研究がみられる．ただし，Oreizy，Taylor らの研究に関しては，文献 [Georgas 08] などでロボットを対象とした実装の試みがなされているものの，実装フレームワークの提案には至っていない．一方，Rainbow はアーキテクチャモデルだけでなく実装フレームワークも提案したものであるが，1 つの制御ループがシステム全体を管理するという集中的な制御機構を採用しているため，複数の適応を扱う場合にはその記述は急激に複雑化する．

自己適応システムの実装フレームワークとしては，Rainbow の他に，StarMX [Asadollahi 09] や Adaptive Server Framework (ASF) [Gorton 06] などが提案されている．StarMX は，Java 言語とポリシー記述言語で動作を記述する実装フレームワークである．StarMX の特徴としては，Java のアプリケーション管理フレームワークである Java Management Extensions (JMX) [Sun Microsystems] を利用して，センサやイフェクタ，アプリケーションを監視する点にある．しかしながら，StarMX は振舞いの制御を基本的にはポリシー言語で記述するため，複数の適応や複数のオブジェクトを扱う場合にポリシー記述部が急激に複雑化する可能性がある．一方の ASF も JMX を用いてアプリケーションを監視するが，ASF はサーバシステムに特化したフレームワークである．

エージェントプラットフォームを利用した自己適応システムの実装フレームワークとしては，Morandini らの提案 [Morandini 08] がある．Morandini らは，Belief–Desire–Intention の 3 つの心的モデルにより適切な行動を決定する BDI モデル [Rao 95] を実装基盤とした Jadex [Pokahr 03] を自己適応システムの実装プラットフォームとして利用している．Jadex ではビヘイビアを実装するのではなく，実行アクションであるプランを記述するスタイルを取るが，プランを起動するためのゴールとの関係を定義する必要があり，また，ゴールの状態により動作するプランが決定するため，並行プロセスの記述は困難である．

JADE におけるビヘイビア記述の拡張やビヘイビア記述方法に言及した開発方法論に関しても，いくつかの既存研究がある．Griss らは [Griss 02] において，階層化されたステートマシンモデルを導入することで，エージェント間の会話（通信）機能の実装を支援するイベント駆動型アーキテクチャSmartAgent を提案している．SmartAgent では中央に各イベントを処理するコントローラを設置し，コントローラが各ビヘイビアにイベントを配分するモデルを採用しているため，本機構を拡張することで，本論文で取り上げた自発的な起動メカニズムを実現できる可能性はある．しかしながら並列実行するビヘイビア間の連携については未解決であり，コンポーネント間の連携を実現できるものではない．Moraïtis ら [Moraïtis 03] は，設計フェーズまでを支援するエージェント指向開発方法論 Gaia [Zambonelli 03] による設計結果を JADE 上で実装する方法について提案し，Nikraz ら [Nikraz 06] はユースケースによるエージェントの同定から JADE を用いたエージェント実装までを包括する方法論を提案している．しかしながら，いずれも FSMBehaviour や SimpleBehaviour を用いた実装を推奨するにとどまり，並行動作するビヘイビアの実装方法に関しては言及していない．

最後に，筆者らは文献 [中川 09] において，本手法で利用するアーキテクチャ構成図を要求分析結果を用いて生成する手法を提案している．この手法は，システムに対する要求をゴール指向要求分析法 KAOS [Dardenne 93] により記述し，その構造から自己適応システムのシステムアーキテクチャを決定するものである．この手法と本論文の提案とを統合することで，システム開発の上流工程から実装までの開発環境が提供されることとなり，要求分析結果に合致した自己適応システムの実現が期待できる．

7. ま と め

本研究では自己適応システムの実現を目的として，エージェントプラットフォーム JADE を拡張した実装フレームワークと制御ループの実現方法に着目した実装ガイドラインを提案した．また，提案する実装フレームワークとガイドラインに従った 2 種類の自己適応システム構築

実験を実施し，その有効性を検証した．本手法により，制御ループの責務とその実装法が明確化され，様々な適応を同時に扱うことのできる自己適応システムの系統的な構築が期待できる．

今後は，実行中の要求や目的の変化に対応するための動的なコンポーネント導入機能を本フレームワークに導入する予定である．また，複雑な適応に対してもシステムの振舞いを保証するために，形式仕様を利用した検証プロセスについても検討を進めたい．

◇ 参 考 文 献 ◇

[Apache Software Foundation] Apache Software Foundation, : Apache – HTTP SERVER PROJECT, http://httpd.apache.org/

[Asadollahi 09] Asadollahi, R., Salehie, M., and Tahvildari, L.: StarMX: A framework for developing self-managing Java-based systems, in *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS '09. ICSE Workshop on*, pp. 58–67 (2009)

[Bellifemine 07] Bellifemine, F., Caire, G., and Greenwood, D.: Running JADE Agents on Mobile Devices, in *developing multi-agent systems with JADE*, chapter 8, pp. 145–171, WILEY (2007)

[Cheng 08] Cheng, B. H., Lemos, de R., Giese, H., Inverardi, P., Magee, J., and et al., : Software Engineering for Self-Adaptive Systems: A Research Road Map, in *Dagstuhl Seminar Proceedings 08031* (2008)

[Dardenne 93] Dardenne, A., Lamsweerde, van A., and Fickas, S.: Goal-Directed Requirements Acquisition, *Science of Computer Programming*, Vol. 20, No. 1-2, pp. 3–50 (1993)

[Dobson 06] Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., and Zambonelli, F.: A survey of autonomic communications, *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, Vol. 1, No. 2, pp. 223–259 (2006)

[Foster 07] Foster, H., Uchitel, S., Kramer, J., and Magee, J.: Towards Self-management in Service-Oriented Computing with Modes, in *3rd International Workshop on Engineering Service Oriented Applications: Analysis, Design and Composition - WESOA07*, pp. 338–350, Vienna, Austria (2007), Springer LNCS

[Garlan 04] Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., and Steenkiste, P.: Rainbow: architecture-based self-adaptation with reusable infrastructure, *Computer*, Vol. 37, No. 10, pp. 46–54 (2004)

[Georgas 08] Georgas, J. C. and Taylor, R. N.: Policy-based self-adaptive architectures: a feasibility study in the robotics domain, in *SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, pp. 105–112, New York, NY, USA (2008), ACM

[Gorton 06] Gorton, I., Liu, Y., and Trivedi, N.: An extensible, lightweight architecture for adaptive J2EE applications, in *SEM '06: Proceedings of the 6th international workshop on Software engineering and middleware*, pp. 47–54, New York, NY, USA (2006), ACM

[Griss 02] Griss, M. L., Fonseca, S. P., Cowan, R. M., and Kessler, R. R.: Using UML State Machine Models for More Precise and Flexible JADE Agent Behaviors, in *Agent-Oriented Software Engineering III (AOSE 2002)*, pp. 113–125, Bologna, Italy (2002), Springer

[Hirsch 06] Hirsch, D., Kramer, J., Magee, J., and Uchitel, S.: Modes for Software Architectures, in *EWSA*, pp. 113–126, LNCS (2006)

[Kephart 03] Kephart, J. O. and Chess, D. M.: The Vision of Autonomic Computing, *Computer*, Vol. 36, No. 1, pp. 41–50 (2003)

[Kramer 07] Kramer, J. and Magee, J.: Self-Managed Systems: an Architectural Challenge, *Future of Software Engineering (FOSE '07)*, pp. 259–268 (2007)

[Moraïtis 03] Moraïtis, P., Petraki, E., and Spanoudakis, N. I.: Engineering JADE Agents with the Gaia Methodology, in *Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, Vol. 2592, pp. 77–91, Springer (2003)

[Morandini 08] Morandini, M., Penserini, L., and Perini, A.: Towards goal-oriented development of self-adaptive systems, in *Proc. of the International Workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS2008)*, pp. 9–16, Leipzig, Germany (2008)

[Nikraz 06] Nikraz, M., Caire, G., and Bahri, P. A.: A Methodology for the Analysis and Design of Multi Agent Systems using JADE, http://jade.tilab.com/doc/tutorials/JADE_methodology_website_version.pdf (2006)

[Oreizy 99] Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S., and Wolf, A. L.: An Architecture-Based Approach to Self-Adaptive Software, *IEEE Intelligent Systems*, Vol. 14, No. 3, pp. 54–62 (1999)

[Pokahr 03] Pokahr, A., Braubach, L., and Lamersdorf, W.: Jadex: Implementing a BDI-Infrastructure for JADE Agents, *EXP - in search of innovation (Special Issue on JADE)*, Vol. 3, No. 3, pp. 76–85 (2003)

[Rao 95] Rao, A. S. and Georgeff, M. P.: BDI Agents: From Theory to Practice, in *ICMAS*, pp. 312–319, The MIT Press (1995)

[SAS 07] *The First IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, Boston, MA, USA (2007)

[Sun Microsystems] Sun Microsystems, : Java Management Extensions (JMX) Technology, http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/

[Sykes 08] Sykes, D., Heaven, W., Magee, J., and Kramer, J.: From goals to components: a combined approach to self-management, in *In Proc. of the International Workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS '08)*, pp. 1–8, Leipzig, Germany (2008), ACM

[Telecom Italia] Telecom Italia, : JADE: Java Agent Development Framework, http://jade.tilab.com/

[Zambonelli 03] Zambonelli, F., Jennings, N. R., and Wooldridge, M.: Developing multiagent systems: The Gaia methodology, *ACM Transactions on Software Engineering and Methodology*, Vol. 12, No. 3, pp. 317–370 (2003)

[中川 09] 中川 博之, 大須賀 昭彦, 本位田 真一：ゴール指向要求分析を用いた self-adaptive システムの構築, 情報処理学会論文誌, Vol. 50, No. 10, pp. 2500–2513 (2009)

[中島 07] 中島 震：ソフトウェア工学の道具としての形式手法, Technical report, NII テクニカル・レポート (2007)

〔担当委員：福田 健介〕

2009 年 12 月 18 日　受理

著 者 紹 介

中川　博之

1974 年生．1997 年大阪大学基礎工学部情報工学科卒業．同年鹿島建設（株）に入社．2007 年東京大学大学院情報理工学系研究科修士課程修了，2008 年同大学院博士課程中退．同年より電気通信大学 大学院情報システム学研究科助教，現在に至る．自己適応システム，エージェントシステム開発方法論の研究に従事．情報処理学会，電子情報通信学会，IEEE CS 各会員．

大須賀　昭彦（正会員）

1981 年上智大学理工学部数学科卒．同年 (株) 東芝入社．同社 研究開発センター，ソフトウェア技術センターなどに所属．1985〜1989 年 (財) 新世代コンピュータ技術開発機構 (ICOT) 出向．2007 年より，電気通信大学 大学院情報システム学研究科 教授．工学博士 (早稲田大学)．主としてソフトウェアのためのフォーマルメソッド，エージェント技術の研究に従事．1986 年度情報処理学会論文賞受賞．情報処理学会，電子情報通信学会，日本ソフトウェア科学会，IEEE CS 各会員．

本位田　真一

1953 年生．1978 年早稲田大学大学院理工学研究科修士課程修了．（株）東芝を経て 2000 年より国立情報学研究所教授，2004 年より同研究所アーキテクチャ科学研究系研究主幹を併任，現在に至る．2008 年より同研究所先端ソフトウェア工学・国際研究センター長を併任，現在に至る．2001 年より東京大学大学院情報理工学系研究科教授を兼任，現在に至る．現在，早稲田大学客員教授，英国 UCL 客員教授を兼任．2005 年度パリ第 6 大学招聘教授．工学博士（早稲田大学）．1986 年度情報処理学会論文賞受賞．日本ソフトウェア科学会理事，情報処理学会理事を歴任．ACM 日本支部会計幹事，情報処理学会フェロー，日本ソフトウェア科学会編集委員長，日本学術会議連携会員．