# ユビキタス環境における動的なプロセス配備のためのプログラミング支援フレームワーク

松崎 和賢[†] 本位田 真一[†,††]

モバイル機器を利用して様々なアプリケーションソフトウェアによる処理が行われるようになってきている．ユビキタス環境では，モバイル機器の画面サイズなどのリソース制約を補うため，外部の機器を仮想的にモバイル機器の拡張として利用することが考えられる．こうしたモバイル機器の仮想的な拡張を行う際には，効率化のためプロセスを分散配置させ並行に動作させる状況が生じる．しかし，現状ではこうしたアプリケーションが効率的に実行されるためのプログラミング支援が不足している．たとえば，ユーザの位置に応じてモバイル機器以外の計算機上でプロセスの一部を実行させる処理を記述する場合，位置条件処理と分散・並行化にともなう記述負担が増加する．本論文では，効率化にともない複雑化する開発を，記述系とその処理系から支援するフレームワークを提案する．記述系はプロセス配置戦略を宣言的・抽象的に定義する支援を，処理系は状況に応じた分散・並行化処理の付加を担う．また，提案手法によりプログラマの記述負担が軽減されることを具体的なアプリケーションの例を通して示す．

# A Programing Support Framework for Dynamic Process Deployment in Ubiquitous Environment

Kazutaka Matsuzaki[†] and Shinichi Honiden[†,††]

Various kinds of application software are getting to be processed on mobile devices. In ubiquitous environments, in order to compensate resource restrictions of mobile devices such as monitor size, processes may use external service appliances as a virtual extension (VE) of mobile devices. When a VE is performed for efficiencies, distributed and concurrent execution of processes is often required. However, proper programming supports are lacking for this situation. For instance, to put a subset of a process on some remote host depending to its user's position requires programmers to handle location condition managements, distribution of the process, and concurrency of the process, which can be heavy burden for them. In this paper, we propose a framework that supports to achieve efficient development for VE in two points. The first point is a description part that allows defining a strategy for allocations and scheduling of a process in a declarative and abstract way. The second point is a processing part that helps to realize the requests written in the form of the strategy. We also show how the framework contributes to lesson the burden of programmers.

## 1. Introduction

近年のモバイル機器・ネットワーク関連技術の進歩は，スマートビルに代表されるようなユビキタス環境のシナリオを現実的にしつつある．同時に，モバイルユーザは様々なアプリケーションソフトウェアをモバイル機器上で実行するようになっている．ユビキタス環境では，モバイル機器の画面サイズなどのリソース制約を補うため，外部の機器を仮想的にモバイル機器の拡張として利用することが考えられる．ペインタのパレット問題（Painter's palette problem）[13),32)] はモバイル機器のリソース制約により生じるもので，支援を必要とする 1 つの例である．この問題は，一般的な描画系アプリケーションソフトウェアはカラーパレットとカンバスをそこに同時に出力するため，カンバスが隠れてしまうというものである．Migratable User Interfaces [13)]，PCOM [3)]，Plan B [2)] などはこうしたモバイル機器のリソース制約を，周辺のネットワーク接続されたサービス機器（ディスプレイ，タッチパネル，スピーカなど）に処理を委譲することで解決を図るものである．本研究では，アプリケーションのプ

† 東京大学大学院情報理工学系研究科
The Graduate School of Infromation Science and Technology of the University of Tokyo
†† 国立情報学研究所
National Institute of Informatics

ロセス☆が周辺サービス機器のサービスを動的に構成して利用することを，モバイル機器の仮想的な拡張（Virtual Extension: VE）☆☆を行うと定義する．

VE を行うプロセスを開発する場合，ユーザの体感を考慮した効率化を施す必要があると考えられる．従来のユビキタス環境向け開発支援（Olympus[24]，CML[14]，PCOM，Plan B など）では，抽象的な環境のモデルを与えることで，具体的な環境の差異を吸収させることが行われてきた．しかし，抽象度を高めたことでプログラミング支援は実現されたものの，依然として実行効率を高めるためのプログラミング支援が考慮されていないという問題がある．これには，効率化のためのプロセス分散・並行化をアプリケーションレベルの条件に応じて行う状況が含まれる．

本論文では，VE を行うプロセスの開発支援フレームワークを提案する．このフレームワークは実行状況に応じた動的配備（Deployment）を宣言的な記述から実行できるという特徴を持つ．実行状況に応じた VE を効率的に実行するためには，プロセスを実行時に判断・処理して分散・並列処理する必要がある．この実現のために，既存の開発支援手法やフレームワークでは開発者の記述負担を著しく増加させてしまう．提案するフレームワークでは，プロセスの配備に関する宣言的な記述から，必要になる分散・並列配に関わる処理をプロセスにコンパイル時・実行時に付加することとでプログラミングの負担を軽減する．なお，配備（Deployment）は一般にソフトウェアを動作させるために必要な作業全般[4]とされるが，本論文では特に，プロセスを適切なタイミングで効率的な実行場所で動作させるための作業，という意味で用いる．

本論文の構成は以下のとおりである．2 章では想定環境・動作の説明を通して本論文の動機の説明をする．3 章では本研究の提案するフレームワークの説明をする．4 章ではフレームワークの設計と実装について示す．5 章では例題に対する本提案手法の有効性を評価・議論する．6 章では関連研究について触れる．最後に 7 章で本研究の結論をまとめる．

## 2. 背　　景

この章では，本研究の想定環境および問題となる状況の説明を行い，そのうえで本論文の基本アプローチを述べる．

☆ 本論文では，プロセスという用語をアプリケーションの処理の実体としての意味で利用する．

☆☆ 標準として利用されている用語ではないが，この論文では可読性のため以下 **VE** と表記する．

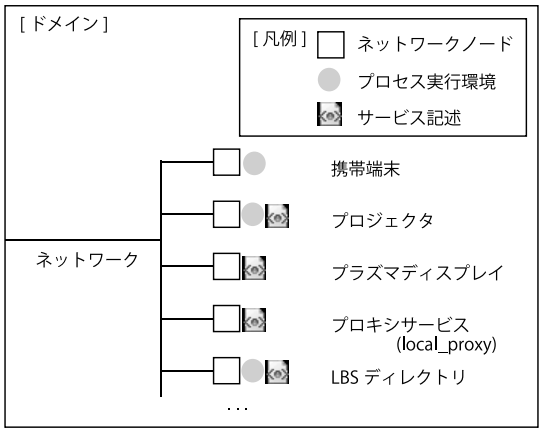



**図 1** VE が行われるドメインモデル．スマートビルの内部を想定
Fig. 1 Target environment example for VE.

### 2.1 想 定 環 境

まずはじめに，本論文の可読性を向上させるため，想定環境の特徴とアプリケーション（プロセス）モデルについて言及する．

#### 2.1.1 モバイル機器の仮想的拡張への対応

**図 1** は本研究の想定環境を簡易モデル表現したものである．この想定環境はスマートビルを念頭に置いている．アプリケーションプロセスに対して実行環境・サービスを提供する，同一ネットワークからアクセス可能なノードの集合は**ドメイン**（domain），と表現される．ここでは，サービス機器類，およびプロセス実行環境（図 1，円形）☆☆☆を提供する計算機資源（図 1，方形）が有線または無線でネットワークに接続されている．アプリケーションプロセスは，位置条件を満たすサービス記述を動的に取得して各種サービス機器を VE のために利用する．また，様々な種類のサービス機器がその機能を統一されたサービス記述（例：WSDL[30]）に則って公開しているものとする．xBPEL[5]，PSI[28]，3PAC[29] などは同様の想定に基づいている．このドメイン内のディレクトリサービス（図 1，*LBS* **ディレクトリ**）により，サービス機器の位置とサービス記述が管理されるものとする．同様に，モバイルユーザの位置情報も管理対象とすることで，位置情報に基づいたサービス（LBS: Location-based Service[7),26)]）をモバイルユーザに提供することが可能となる．

これらのサービスに，プロセス実行環境を提供するプロキシサービス（図 1，`local_proxy`）の存在を想

☆☆☆ プロセスは，プロセス管理フレームワークの提供する実行環境の上でのみ実行が可能であるものとする．

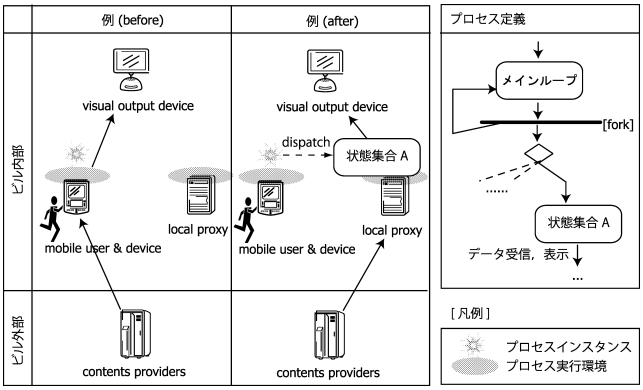



**図 2** プロセスの分散・並列化が有効な例
Fig. 2 An allocation example.

定している．プロキシサーバはモバイル機器の負荷分散に利用される．

本論文ではこれらの想定の上に VE が実現されるものとする．

### 2.1.2 アプリケーションモデル

本論文では，プロセス表記に基づいたアプリケーションモデルを利用する．これは，アプリケーションの動作を表現するうえで標準的に用いられる UML 状態遷移図・アクティビティ図に代表されるものである．また，プロセス定義は XML 形式などで宣言的に記述されるものとする☆．実行の際には，各状態に関連づけられているアクションハンドラ☆☆を実行し，条件に沿った状態遷移を行う．

## 2.2 モチベーション

VE を行うプロセスで実行の効率化を考える際に，分散・並行化を必要とする状況が生じる．たとえば，モバイル機器を用いてマルチメディアコンテンツを受信し，そのデータを VE により外部映像表示サービス（`visual output device`）へと転送する状況を考える（**図 2**-例（before））．このとき，モバイル機器以外の処理力のあるプロセス実行環境を利用して映像表示サービスへのデータ転送処理を行うという選択肢が考えられる．ユーザと映像表示サービスからアクセス可能なプロキシホスト（図 2，`local_proxy`）が存在する際にそれを利用するという状況である．VE を用いる動機は，モバイル機器がユーザの他の動作やリクエストに対して即応できる状態にしておくためといえる．しかし，ビデオ会話など大量のデータ転送処理はオーバヘッドも大きく，その妨げになると考えられる．そのため，プロセス中の VE を利用する部分をプロキシホストに配備して実行を行うことは，効率的な実行につながると考えられる．この例において，プロセスがユーザからのリクエストを受信する状態（図 2，メインループ）はユーザのモバイル機器上で実行されるのに対し，負荷分散に関わる効率化処理（図 2，状態集合 A）部分はローカルプロキシ上へと送信され，そこで実行される（図 2-例（after））．

問題は，こうした負荷分散などの効率化を実現するために，既存研究では支援されていない作業がプログラマに要求されることである．ここで，VE を行うプログラムを開発するための，従来研究によるアプローチを 2 通りに分類する．これらをプログラミングモデルアプローチと OS アプローチとする．

- プログラミングモデルアプローチは，物理層・ネットワーク層など下層の操作を隠蔽するミドルウェアを利用し，その上で高水準な API を提供する．たとえば，Olympus は Gaia ミドルウェア[25] を利用し，仮想エンティティ☆☆☆ の操作を API から容易に行うことができる．特に，逐次実行型のタスクを実行するうえで有用である．
- OS アプローチの場合，プログラマは下層の変化を意識することなくプログラミングが可能となる．プログラマは自動的に VE をすることになるプログラムを，Java などのプログラミング言語を用いて通常のデスクトップ環境用と同様に記述することが可能となる．

双方に共通して，図 2 に示すような，効率化のため

☆ 本論文では説明のうえで，JBoss jBPM [16] のモデル・記述形式を利用する．付録 A.1 に簡単な記述例を示す．

☆☆ Java スレッドクラスとして動作する．

☆☆☆ Application，ApplicationComponent，Device，Service，Person，PhysicalObject，Location，ActiveSpace などの仮想的な要素に対する操作をプログラム中で抽象的に定義し，実行時に実体化した具体的な要素に対し操作を適用していく．

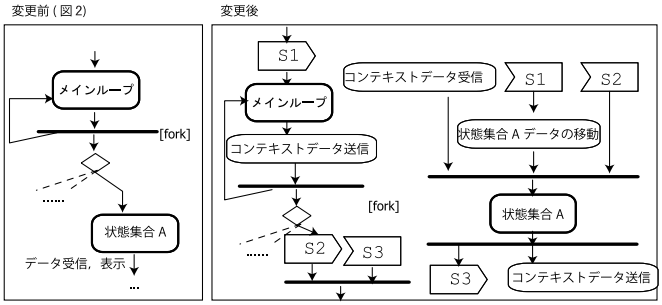



**図 4** VE の効率的な実行のための分散・並行化の例
Fig. 4 An example of distributed and concurrent execution for VE.

```
    //user1 の存在する ActiveSpace をインスタンス化
1   ActiveSpace as1;
2   as1.hasProp("containsPerson", "user1");
3   as1.instantiate();
    //user1 の側の画像出力サービスをインスタンス化
4   Service service1;
5   service1.hasProp("class", "VisualOuput");
6   service1.hasProp("location",
          user1.getProp("location", "room"));
7   service1.hasProp("resolution", "800*600");
8   service1.hasMetric("distance", user1, "ascending");
// プロキシサービスの存在する ActiveSpace をインスタンス化
9   ActiveSpace as2;
10  as2.hasProp("class", "LocalProxy");
11  as2.hasMetric("somevalue");
12  as2.instantiate();
    // 効率化のためのコンポーネントを移動させる
13  AplicationComponent ac1;
14  ac1.migrate(as2);          =  状態集合 A
15  ac1.start();
```

**図 3** Olympus における図 2 例（after）の記述例
Fig. 3 An example description with the Olympus.

に分散・並行実行するプロセスに対するプログラミング支援が不足していると考えられる．ここで，プログラミングアプローチの Olympus を利用した場合を考える．図 2-例（after）の記述例は**図 3** のようになる．アプリケーションコンポーネント（`ApplicationComponent`，13 行目）を状態集合 A に相当するものとする．1–3 行目でユーザ（user1）のいる空間（ActiveSpace）を，4–8 行目で利用したいユーザのいる空間映像表示サービスをそれぞれ属性値を用いて抽象的に指定できる．これらの空間やサービスは実行時に具体化される．同様にして，9–12 行目でアプリケーションコンポーネントを送信するプロキシホストを指定し，13–15 行目でアプリケーションコンポーネントを送信先で実行する．

このように，記述としては簡潔であるが，プロセスの部分集合を，いつ，どのような条件下で，どこのホストに配備するのか（動的配備，スケジューリング，配備場所の条件），またプロセスの他の部分との同期（実行・データ）をどうするのか，という処理記述を提供された API の枠組みでは記述できず，別の方法で処理を追加しなければならない．また，OS アプローチの場合も同様に，これらの処理がプログラマに委ねられる．そのため，本来記述したいプロセスのロジックではないこれらの処理にプログラミングの負担がかかることが問題となる．

実際に図 2-例（after）のシナリオを実現するためには，たとえば **図 4** に示す手続き拡張を必要とする．この例では，まずメインループに入る直前に状態集合 A のデータをプロキシホストに送信する処理（シグナル S1 の送受信による同期）を行う．次いで，状態集合 A を実行する前にコンテキストデータ（プロセス内の共有変数）を同期する処理，およびメインループから状態集合 A に状態遷移することを通知する処理（シグナル S2 の送受信による同期）が加えられる．最後に状態集合 A の処理が終了した際にコンテキストデータの送信と終了の通知が送られる（シグナル S3 の送受信による同期）．状態集合 A をローカルプロキシに送出するための位置情報処理・移動処理・同期処理などを扱う必要があり，本論文ではプログラマのための支援を要する問題と考える．

### 2.3 基本アプローチ

これに対し，本研究ではプロセス実行環境側のフレームワークと，位置情報に基づいたサービス（LBS）ディレクトリとの双方に従来のプロセス実行環境からの拡張を加えることで対処させる☆．本論文の主要な提案となるクライアント側の拡張は，プロセス配備戦略を宣言的に定義し，フレームワークで状況に応じた処理支援をするというものである．戦略はプロセスをいつ，どこで実行するかといった配備・スケジューリングに関するものである．処理支援は，プログラマが細かい条件分岐を書くことを回避するためのものであり，位置条件によって処理が切り替えられる．これには，図 2

☆ 以下，区別のためプロセス実行環境側のフレームワークをクライアント側，LB S ディレクトリをサーバ側と呼ぶ．

の例のように，プロセスの一部分を動的に遠隔ホスト上に配備するための処理を静的・動的に追加すること，およびその追加を行うべきかどうか判定することとが含まれる．サーバ側の拡張は LBS ディレクトリサービスの配備戦略に対応する拡張であり，抽象的・宣言的な配備条件をリクエストを受けた時点での情報に基づき解決するというものである．

## 3. プロセス配備戦略のためのフレームワーク

この章では，プロセスの配備戦略（Deployment strategy）に関する拡張を加えたプロセス管理フレームワークを提案する．このフレームワークは，従来のプロセス管理フレームワーク[16] に対し配備戦略に関する記述系と処理系に拡張が加えられている．

記述系の支援により，開発者は宣言的にプロセス配備戦略を記述することができる．ここで，位置選好（Location Preference: LP ☆）という概念を導入する．LP はプロセス $P$ の各部分集合 $Sub_i(P)$（i は識別のための整数）を配備する場所に関して抽象的に指定するためのものである☆☆．また，配備戦略 $St_i(P)$ を，プロセス $P$ の部分集合 $Sub_i(P)$ が処理を行う場所（LP）と時間を指定する宣言的な情報と定義する．

処理系の支援により，従来プログラマの負担となっていた処理記述部分を自動化する．さらに実行時の状況に応じて，オーバヘッドを抑制する解析を行う．

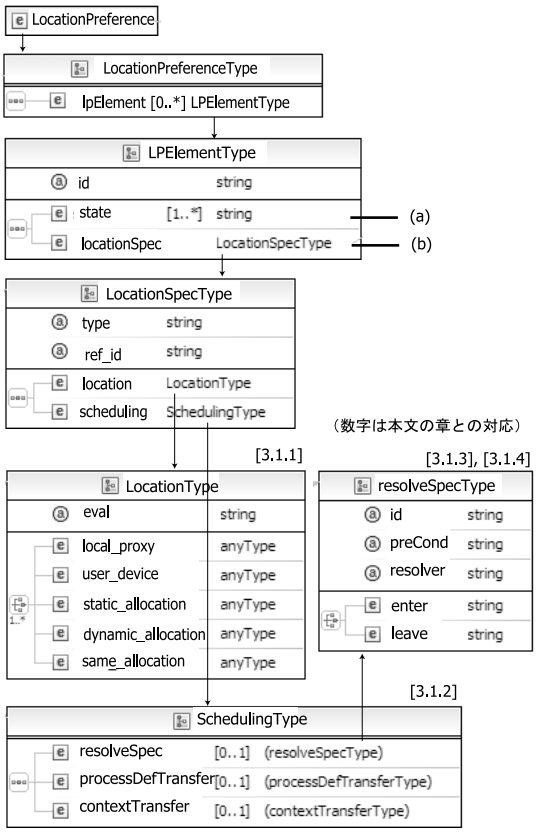
### 3.1 記 述 系

記述系では，VE を行う際の実行効率化のためのプロセス配備戦略を定義する枠組みが提供される．プログラマは，プロセス定義の各部分集合 $Sub_i(P)$ に対してその配備を効率化するために以下の情報を定義する．

- どのような実行環境に配備されるべきか（→3.1.1）
- 何に関する情報を扱うのか（→3.1.2）
- どのタイミングで処理を行うのか（→3.1.3）
- 何によって宛先を決定するのか（→3.1.4）

位置選好（LP）はプロセスの部分集合（$Sub_i(P)$）と抽象的な位置分類とを関連づけるものである（**表 1**-*LPElementType*-(*a*)，(*b*)）☆☆☆．**図 5** は，図 2 と同じ状況における LP のモデルを示している．LP を用いることにより，開発時には抽象的な位置分類を定義しておき，実行時に状況に合わせて適切なプロキシサービスを用いて行うことが可能となる．以下で，リストにあげた項目について説明をする．

**表 1** 配備戦略記述系
Table 1 Descriptions for allocation strategy.




#### 3.1.1 位 置 分 類

XML スキーマのモデルである表 1-*LocationSpecType* は，フレームワークで提供される位置分類を示している．この位置分類は配備場所を抽象的に記述し，特定の実行環境に束縛されることを避けるために利用される．表 1 中の *local_proxy, user_device* は図 1 中の項目と対応する．*static_allocation* は具体的なアドレスを指定する際に，*dynamic_allocation* はユーザ定義クラスを用いて実際のアドレス解析をする際に，それぞれ用いる．また，*same_allocation* は，プロセスの部分集合 $Sub_l(P)$ が別のプロセスの部分集合 $Sub_k(P)$ と同じ位置で実行することを指定する．

#### 3.1.2 配備スケジューリングの対象とする情報

表 1-*SchedulingType* は配備戦略におけるスケジューリングの対象を示している．これらの対象は，プロセス実行時の情報を反映した配備戦略の実行のために必要となる．配備場所を解析するタイミング（*resolveSpec*）は，たとえばユーザから近くのタッチパネルサービス

☆ 以下，位置選好を LP と記す．
☆☆ jBPM などのプロセス管理フレームワークでは，プロセスはグラフ表現される．プロセス $P = G(V, E)$ において V はプロセスを構成する実行状態，E は実行状態間の状態遷移である．各部分集合は $Sub_i(P) = \{G'(V', E')|V'$ は V の部分集合，E' は E の部分集合 $\}$ と表現される．
☆☆☆ 記述例を付録 A.1 に記載する．

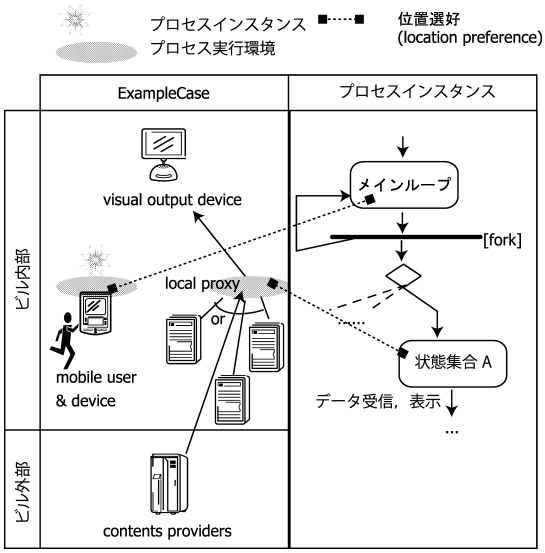



**図 5** 配備戦略を利用した際の実行イメージ
Fig. 5 A sample case with an allocation strategy.

を指定する際に，できるだけ必要とする状況に合致したタイミングで具体的な配環境を決定したい際に利用する．また，プロセス実行に必要なデータ（プロセス定義と実行に必要なクラス）を送信するタイミング（*processDefTransfer*），コンテキスト変数（同一プロセス内のクラスで共有する変数）の同期・転送タイミング（*contextTransfer*）を指定する．

#### 3.1.3 配備スケジューリング

プロセスの実行がある状態に遷移した場合（*enter*）と，ある状態から遷移した場合（*leave*）を指定することで，スケジューリング対象の実行タイミングを指定する．特別な状態を示すキーワードとして，配備直後（*deploy*），実行直後（*start*），終了時（*end*）も指定可能とする（表 1-*SchedulingType*）．

#### 3.1.4 位 置 解 決

表 1-*SchedulingType-resolveSpec* の属性として，*resolver* を指定する．これは，LBS ディレクトリかユーザ定義クラスを実際のアドレス解決に利用する．

### 3.2 処 理 系

処理系では，定義された配備戦略を実際に実行に反映させるための機構が組み込まれている．ここではまず，フレームワークによる支援が開発手順のどの段階で行われ，何が自動化されることになるのかを示す（3.2.1 項）．そして，定義された配備戦略を実行するうえで，一部の無駄な処理を省くための配備戦略分析について述べ（3.2.2 項），最後にプログラミング支援として自動的に実行される処理について説明する（3.2.3 項）．

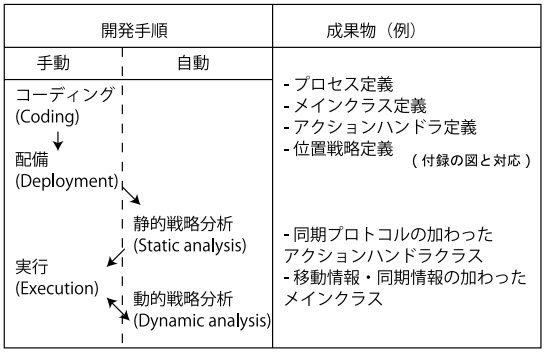



**図 6** 配備戦略を導入した際の開発手順と成果物
Fig. 6 A development process and architects with allocation strategies.

#### 3.2.1 開 発 手 順

**図 6** に配備戦略を導入した際の開発手順の概要を示す．開発手順のうち，従来どおり手動で行われる作業はコーディング・配備・実行である．これに対し，静的戦略分析を配備直後に，動的戦略分析を実行時に，それぞれ自動化された処理として追加する．コーディングの段階における成果物の例は，付録-**図 13**，A.2 と対応する．続く 3.2.2 項に示す配備戦略分析により，必要な処理の自動化の可否を判定する．

#### 3.2.2 配備戦略分析

**図 7** に配備戦略の分析手順を示す．この分析の目的は，配備戦略の実行における非効率な処理を抑制することである．これは，十分な効果が得られない 2 つの状況において配備戦略の実行を行わないということである．1 つ目の状況は，同じホスト上で実行している $Sub_1(P)$ と $Sub_2(P)$ の間で配備戦略に関係する同期メッセージを送受信する状況である．2 つ目の状況は，すべての配備戦略実行をプロセス実行時になるまで行わない状況である．$Sub_n(P)$ の配備場所が静的に定まっている場合，配備戦略の実行は適切なスケジューリングの対象となるべきと考えられる．

これらの状況下で，VE の効率的実現という本来の目的を達成できなくなる懸念がある．本フレームワークによる分析は，この 2 つの状況への対処を行うものである．

静的分析・動的分析とも基本的な手順は同じである．ここでは，図 7 中の番号に従って静的分析の説明をする．

(1) 配備戦略のうち，プロセスの配備直後に配備場所が確定するものの存在を調べる．これには，*local_proxy* を利用し，かつそのアドレス解決を配備直後に行う場合，およびユーザが静的に *local_proxy* のアドレスを指定している場合が含ま

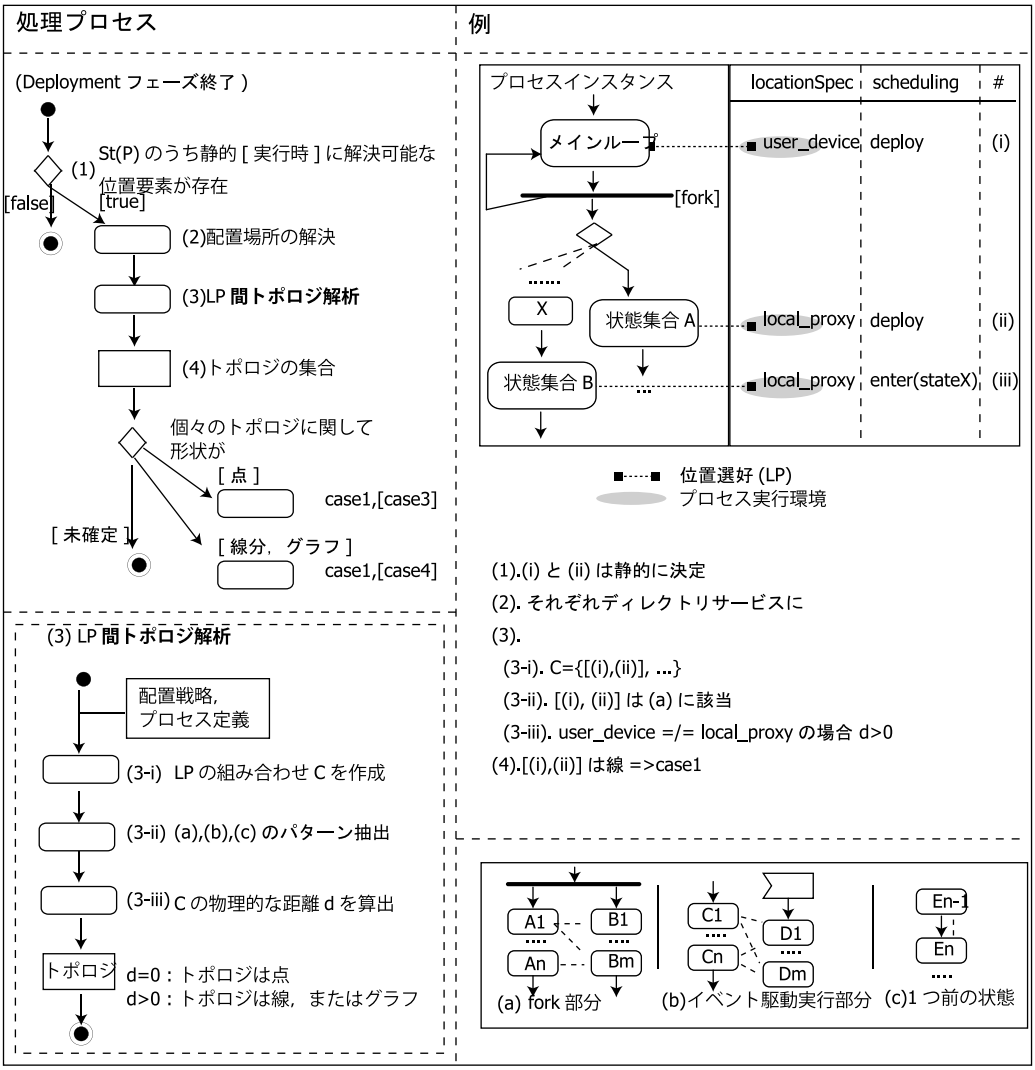



**図 7** 配備戦略分析
Fig. 7 Analysis phase of allocation strategies.

れる（3.1.1 項，3.1.2 項-*resolveSpec*）．

(2) 指定された位置解決手法により配備場所を解決する（3.1.4 項）．

(3) LP 間の位置関係を判定する．これはある実行状態において並列に実行される $Sub_k[P]$ と $Sub_l[P]$ とのネットワークに関する位置関係となる．この位置関係を LP 間のトポロジ形状と定義し，その解析を以下の手順で行う．

3- i. LP とプロセス定義を入力とし，LP の組合せを抽出する．LP の指定のないものは前の実行状態と同じ LP が割り当てられているものと仮定して扱う．

3- ii. 図中 (a)，(b)，および (c) の条件を満たす組合せを抽出する．(a) は並列実行の箇所（fork の後），(b) はイベント駆動実行（イベントの有効条件に明示的に C1,...Cn が含まれている場合のみ），(c) は連続する関係にある実行状態 $E_n$，$E_{n+1}$ ☆．

3- iii. 各組合せに対し，ネットワーク上での位置関係を表現したトポロジ形状を取得す

☆ この 3 条件を利用する理由としては，並行実行になりうる基本的な部分であるということと (a)，(b)，配備場所の切替えにともなうオーバヘッドを起こしうるため支援を要すること (c) とからである．これらは基本的な処理方針を示すためのものであり，網羅性に関しては本論文では扱わない．また，プロセス定義を XML などで記述する場合，状態間の関係が静的に取得可能となる．

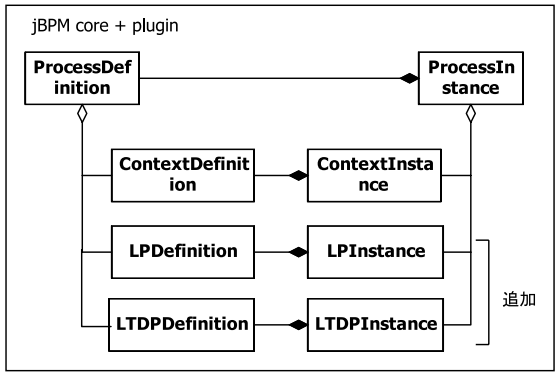



**図 8** jBPM を利用した実装モデル
Fig. 8 Implementation model using jBPM.

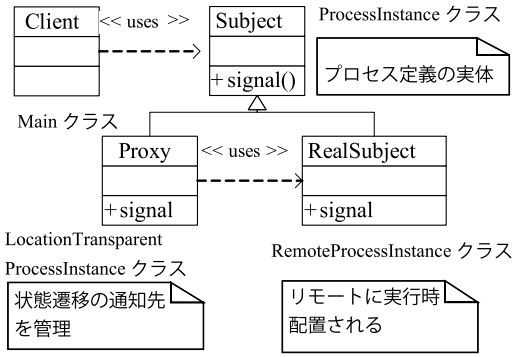



**図 9** プロキシパターンによる戦略の処理系
Fig. 9 Processing phase of allocation strategies by proxy pattern (remote proxy).

る．たとえば，図 5 において，**メインループ**と**状態集合** $A$ は同一の fork の後に存在するためトポロジ形状解析の対象となり，仮に状態集合 A の LP に含まれるプロキシホストのアドレスが**メインループ**の LP であるユーザのモバイル機器以外の場合，ネットワーク上の別のノードに存在することからこれらの組のトポロジ形状は「線」となる．

(4) 状態（状態集合）の組とそのトポロジ形状からなる集合を得る．

以下，個々のトポロジ形状が点であるときを Case1，それ以外の形状時（線分・グラフ）を Case2 として処理する．また，いずれかの端点の配備場所が未定の場合，トポロジ形状が決定されないため特に処理されない．動的戦略分析の場合も同様にして，LP の解析がスケジュールされている際にそのつど行われ，トポロジ形状が点のときを Case3，線のときを Case4 として処理する．以上の分類結果の内容と，フレームワークによる処理を整理すると以下のとおりになる．

**Case1［Case3］** トポロジ形状が点と Deploy 終了時［実行時］に判明 → 何もしない．

**Case2［Case4］** トポロジ形状が点でないと Deploy 終了時［実行時］判明 → 戦略のための処理（分散・並行化処理）を静的［動的］に付加

#### 3.2.3 配備戦略実行

分析の結果を利用してフレームワークは一部の処理を自動化する．アスペクト指向プログラミングで利用されている実装手法を利用し，プロセスの配備に必要なクラスの拡張・インタフェースの実装をフレームワークにより行う[1),9),15),23)]．自動化される箇所は，**図 9** の *RealSubject*（$Sub_i(P)$，およびその複製）の生成と呼び出しに関する処理である．フレームワークの処理を受けたプロセスは，複数存在する可能性のある *RealSubject* のうち実行時の状況に応じて実際に通知すべきノード上のものを呼び出す．そのため，プログラマからみてプロセスの配備は位置透過となる．また，遠隔地に配備されたプロセス間の通信部分もフレームワークにより支援される．

位置条件が利用可能になった結果，効率的な配備を実行するためのプロセスのプログラムは複雑になってしまう．提案するフレームワークによる自動化は，プログラムが複雑化になる負担を軽減する意味を持つ．

## 4. 設計と実装

この章では，フレームワークの設計と実装について述べる．3 章で記述した機能を実現するために，プロセスの実行環境・戦略処理機構としてのフレームワーク（クライアント側）と LBS ディレクトリサービス（サーバ側）が必要となる．

### 4.1 クライアント側

フレームワークはビジネスプロセス管理ツールの Java 言語によるオープンソース実装である，jBPM[16)] の拡張となっている（**図 8**）．jBPM では，機能が複数のモジュールに分割され，各モジュールが定義部分と実行部分とに分かれている．追加モジュールにより，状態遷移モデルの主要部分であるグラフモジュール☆を利用・拡張できる．本研究では *LP*（*Location Preference*），*LTDP*（*Location Transparent Dynamic Proxies*）という 2 つのプラグインを作成した．これらは 3.2 節に示す処理を実現するためのものであり，LP は配備戦略の記述・分析処理に関するモジュールである．LTDP

☆ ProcessDefinition はプロセス定義 XML に対応したクラス，ProcessInstance はプロセス定義をインスタンス化したものである．

メインクラス

```
public class MyApplication implements
                                    MessageListener{
    public void main(){
        ...
        processInstance.signal();
        /* 状態遷移を管理する箇所に，位置情報の
        問い合わせなどのロジックが混合する */
        URL local_proxy = queryProxyAddress(
                getProperties(``spec'' ));
        if (local_proxy.equals(
                getProperties(``mobile_device'' )){
                processInstance.signal();
        } else {
                sendProcessSubset(local_proxy,...);
                /* リモートホストに状態遷移を行う
                ため，*/
                processInstance.signal(local_proxy);
        }
        processInstance.signal();
        processInstance.signal();
    }

    public void onMessage(Message msg){
        if (msg instanceof SignalThree){
       /* 同期ロジックが元々のメッセージ処理
        ロジックと混合する．*/
        }
    }
}
```

状態集合 A 中の状態と関連付けられたアクションハンドラ

```
public class MyAction implements ActionHandler
                                    ,MessageListener{
  public void execute(){
      /* 同期ロジックが混合 */
      waitSignal(SignalOne, SignalTwo);

      originalLogic();

      /* リモートで実行している場合には，
      コンテキスト情報を送信するなどの処理
      が最後に追加される */
      sendContextData();
  }

  public void onMessage(Message msg){
      // 同様に混合
      if (msg instanceof SignalOne){
      }
      else if (msg instanceof SignalTwo){
      }
  }
}
```

**図 10** 配置戦略を実現するためのプログラム記述モデル．網掛け部分は配置戦略のために追加記述する必要のあるロジックである
Fig. 10 A description example to realize an allocation strategy.

は配備戦略の実行処理に関するモジュールである．これらのモジュールではリモートのプロセス実行環境との通信に JBoss Remoting [17] を利用している．

処理実現のため，デザインパターン [11] の Proxy パターン（Remote Proxy）を利用し，メインクラスから状態遷移を次の状態に通知する際に，通常同一ホストの状態に通知するところで，戦略に基づいて配備されたホスト上の状態に通知する（図 9）．この際，(1) 実行に必要なプロセス定義・クラスファイルの配備をデータのシリアライズ転送により行う（図 9，*RemoteProcessInstance* クラスなど），および (2) リモート呼び出しによる配備先のプロセスへの状態遷移（図 9，*LocationTransparentProcessInstance* クラスによる状態遷移シグナル転送の判断）を行っている．プログラマは，状態遷移先ホストの具体的な情報をプログラム中に記載せず，抽象的に戦略として指定する．これにより，実行状況によって異なる配備でプロセスの実行が進み，プログラマから見て具体的な配備が透過となる．

### 4.2 サーバ側

サービスディレクトリはプロセス（クライアント）からのリクエストを受け付け，配備戦略を解析し適切なアドレス情報を返す．RFID を利用して，Object Name Service [7] の例として定義されている，“あるオブジェクトを指定して，関連づけられている WSDL ファイルを返す”，という動作を簡易的に実装した．また，“あるユーザ ID から最もそばにあるサービス機器と関連づけられている WSDL ファイル” の取得も可能となっている．本論文の LBS ディレクトリにはこの機能を利用し，プロセスにとって適切な実行環境（プロキシサービス）の解析を行う機能を追加した．配備戦略に関しては，物理的な距離，およびネットワーク的な距離とに限定してクライアントは条件を追加できる．クライアントから物理的に最も近いサービス，およびある特定のサービス（たとえば，プラズマディスプレイ）からネットワーク的に最も近い（遅延が最も少ない）プロキシサービス，という指定のみをプロトタイプ実装では定義可能である．

## 5. 評価

この章では，2.2 節で述べた問題に対する提案するフレームワークの有効性を評価する．以下の 3 つの評価項目に対して検証を行う．

- 効率化のプログラミング時に負担が増加することを，提案フレームワークの配置戦略記述系，処理系サポートによりどのように抑止可能か（ミクロ・定性評価）．
- 同様にどの程度抑止可能か（マクロ・定量評価）．
- 配備戦略という抽象記述を開発者に提供した際に生じる，実行速度とのトレードオフについて．

### 5.1 ミクロ・定性評価

図 4 に示す動作をすべて手作業でプログラミングする際のコードモデルを**図 10** に示す．状態遷移を管理するプロセスのメインクラス（*MyApplication*）と，（状態集合 A に含まれる）状態と関連づけられて実行

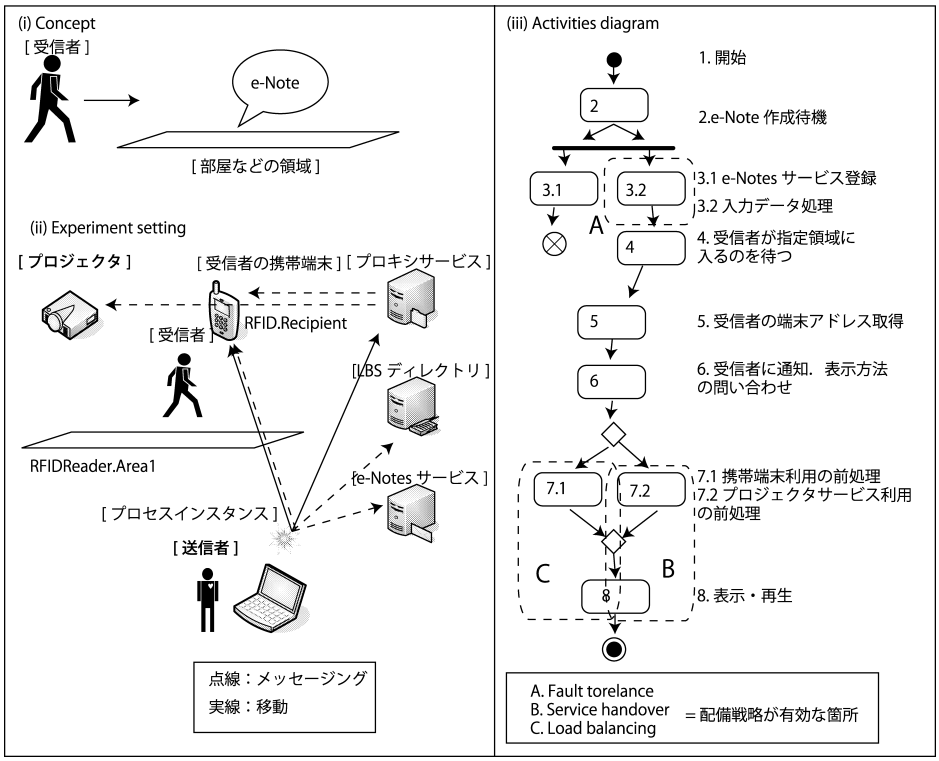



**図 11** 評価アプリケーション（e-Notes）
Fig. 11 e-Notes: a sample application for evaluation.

されるアクションハンドラクラス（*MyAction*）は，同期や配置のためのコードと混在してしまう．プログラマが分散・並列化の処理に不慣れな場合は特に，問題としてあげたようなプログラミング負担が発生すると考えられる．さらに，テストや更新，別の環境での再利用のため起こるプロセスの修正にともなうプログラミング負担も考えられる．メインクラス中の状態遷移を管理する箇所に，位置情報の問合せなどのロジックが混在することになる．これはプロセス本体のみの修正などを考えた際に，作業の障害となりうる．また，メッセージ処理部分に関しても（*onMessage()*）同期ロジックが元々のメッセージ処理ロジックと混在することになる．これは，誤動作や，デバッグの困難さを招くことにつながる．アクションハンドラに関しても同様に，本来のロジックを記述する場所（*execute()*）に同期ロジックが混在することになり，本来のロジックのメンテナンスを行う際にも双方の情報を修正する必要が生じるなど，作業コストをかけてしまう．

本提案手法では，こうしたプログラミングの負担に関する部分を，戦略という形でメタプログラミングと同等の機能を提供し，自動化している．この点で問題に対する有効な解であると考えられる．特に，位置透過性を考慮する点で，位置条件の動的な変化の予測される VE のシナリオに対しても有益であると考えられる．

### 5.2 マクロ・定量評価

ここでは，実際にフレームワークの機能を利用するアプリケーションの構築事例を通してフレームワークの評価を行う．本研究では，ユビキタス環境向けのアプリケーション “e-Notes” を実装した[8),21),27)]．これは，モバイルユーザがある受信者に対し特定の領域に電子的なメッセージを残すことができるというものである（**図 11**-(i)）．動作概要は以下のとおりである（図 11-(iii)）：(1) ユーザが e-Notes プロセスを起動する．(2) ユーザのデータ入力を待つ．ここでの入力は 3 種類ある．受信者を識別する RFID，メッセージを残したい場所に存在する RFID リーダの ID，および送信するデータである☆．(3) 入力データの処理，および e-Notes サービスへの登録．(4) 受信者が指定した領域に入るのを待つ．(5) 領域内の受信者端末アドレスを取得．(6) 受信者に表示方式を問い合わせる．(7)

☆ RFID は ID カードや携帯端末につけられていることを想定している．4.2 節の位置管理システムを実現するうえで，RFID リーダが各所に配置されていることを想定している．

**表 2** 通常の実装と提案手法による実装とのコード量比較
Table 2 A comparison of lines of code between normal implementation and implemetation with our framework.

| | 通常 | 提案手法 | 増減 | |
|---|---|---|---|---|
| ソースコード (Java) | 1,402 | 1,224 | -178 (-12.6%) | (MLOC) |
| 設定ファイル (XML) | 86 | 160 | +74 (+86.0%) | (行) |

表示端末（サービス）の前処理．(8) 表示・再生．

このアプリケーションにおいて，提案手法が有効な状況は，図 11-(iii)-A，B，C に例として示す，アプリケーションの適応動作と考えられる．(A) 冗長性を持たせる（fault tolerance）：データの消失を防ぐために $Sub_i(P)$ のクローンとデータのコピーをプロキシサービスなどに待避させる．(B) サービス切替え時のオーバヘッド削減（service handover）：ユーザが表示先の指定を切り替えた際の処理オーバヘッドを下げるため他のサービス候補の検索などを行う．(C) 携帯端末による処理の負荷分散（load balancing）：2.2 節であげた例と同じく，受信者端末のリソース制約への対応を VE により行う．

また，アプリケーションは以下のサービスを随時利用する．(3) では e-Notes サービス，(5) では LBS ディレクトリサービス，(A)，(C) ではプロキシサービスをそれぞれ利用する（図 11-(ii)）．e-Notes サービスではユーザから入力されたデータを保存し，受信者が指定した領域に入った際にアプリケーションに通知する．LBS ディレクトリサービスは 4.2 節で述べたものである．プロキシサービスはプロセスの実行環境を提供する．

e-Notes アプリケーションのプロトタイプを実装したコード量☆に関して**表 2** に示す．配備戦略 (A)，(B)，(C) を実現する際に，提案手法を用いることで 12.6% コード量を抑制することができた．設定ファイルに関しては，戦略を記述する分だけ 86% 増加した．

今回の実装は，評価目的であり GUI 機能やマルチメディア処理機能は必要最低限しか実装していない．そのため，実用目的のアプリケーションでは，全体のコード量が増加し提案手法によるコード量の抑制効果は落ちることが考えられる．しかし，適応動作への要求とソースコードのメンテナンス性とを考慮すると，提案手法の意義は保たれると考えられる．適応動作への要求に関して，ユーザの体感を向上させるためにはユーザの移動にともなう環境の変化への適応動作も増え，結果的に戦略利用箇所が増えると考えられる．

☆ プログラム部分（Java 言語）は Method Lines of Code (MLOC)[19]，設定ファイル（XML）は行数．

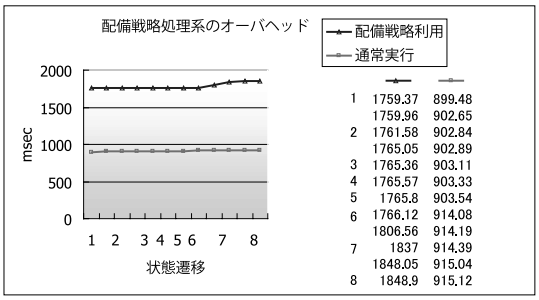



**図 12** 配備戦略処理系のオーバヘッド評価
Fig. 12 An experiment result for processing overhead of our deployment strategy framework.

### 5.3 配備戦略処理系のオーバヘッド評価

配備戦略という抽象記述を提供することによるオーバヘッドとして，戦略の解析にかかる時間があげられる．解析としては，(1) 状態遷移の際に処理を行う必要があるか判定すること，(2) 3.2.2 項に述べた非効率処理を回避する分析を行うこと，(3) 抽象的な位置条件から具体的な情報を得ること，に分類できる．これを配備戦略に関して従来の処理系と比較すると，(1)，(2) がオーバヘッドとして該当する．(3) に関しては，LBS ディレクトリサービスの処理となるため開発者がマニュアルで記述した場合にも同様の時間を消費すると考えられるためである．**図 12** にオーバヘッドを比較したグラフを示す．グラフ中の状態番号は，図 11 と対応する．実験には，CPU：Intel Pentium M Processor 1 GHz，メモリ：1.24 G というスペックのノート PC を用いた．この結果，通常の処理系よりもほぼ 2 倍の時間を消費することが示され，配備戦略の初期読み込み（状態 1）と並列に配備戦略の解析が要求される場合（状態 7–8）に特に処理時間を要することが分かる．しかし，ほぼ 2 倍とはいえおよそ 2 秒程度であり，例にあげた負荷分散のようなシナリオを適切に行う場合，全体として容易に相殺されうる範囲の時間消費であると著者らは考えている．

## 6. 関 連 研 究

ユビキタス環境向けソフトウェアの開発支援の研究がいくつか存在する．2 章で述べたように，プログラマ支援は OS によるものと抽象的なプログラミングモデルを提供するものとに分類できる．Plan B[2] は OS レベルのアプローチである．これはミドルウェアアプローチに存在した，ネイティブ OS へのタスクを委譲

するなどの依存性を排除したものとなっている．このアプローチは低層（ハードウェア，ネットワークなど）を抽象化することに成功している．そのためプログラマは，デスクトップ環境におけるプログラミングと変わらないプログラミングが可能となる．Jadabs[10]もモバイル環境に特化した同様のアプローチである．しかし，これらのアプローチで負荷分散の例などを実現するためには，結局低層の情報を使う必要があり，コンポーネントの分散を隠蔽することに主眼を置くこれらのモデルではプログラミングのサポートに欠け困難となる．プログラマがアプリケーションレベルの効率化を試みる場合，非常に具体的で分散・並行性に対応したプログラムを記述しなくてはならない．これは非常な負担であると考えられる．

後者のアプローチは，ユビキタス環境向けのミドルウェア層の上に作られている．ミドルウェアの例としては，Ninja[12]，Gaia[25]などがあげられる．PCOM[3]は本論文であげた想定環境と同様の環境下で，契約に基づくコンポーネントの動的な組合せを実現する．コンポーネントとしてキーボードなどを考慮しており，本研究の想定する VE の概念と同一である．PCOM のモデルはパーベイシブ環境でアプリケションを開発・実行するのが非常に困難であるという考え方に基づく．PCOM はアプリケーションから利用されるデバイスやサービスの動的な変化（追加・消失）を吸収するように設計されている．この契約の機構は，VE を実現するが，やはり効率化の例などを実現するためには不十分である．コンポーネント間のデータフローまで定義することはできないためである．

CML[14]は，コンテキストモデリング言語であり，プログラマに対しコンテキストモデル，選好モデルと管理，そしてプログラミングモデルを提供する．CML はコンテキストアウェア開発の次の段階として設計された．これは，コンテキスト情報の取得・処理・コンテキスト情報の管理に関しては一定の水準に達しているため，ソフトウェアの開発支援にシフトしているというものである．コンテキストファクト・状況（Context fact and situation）がモデリングされ，高度なプログラミングモデルを与える．これにはブランチング・トリガリング（branching and triggering）が含まれる．ブランチングはすべてのサービスの中からシステムで算出されるランクの値に基づいてサービスを選択するためのものであり，トリガリングは高レベルのコンテキスト情報を利用したイベント駆動プログラミングモデルを適用する．CML にはツールによる支援も追加されている．しかし，例にあげる効率化に対する支援はされていない．

Olympus[24]は Gaia ミドルウェアの上で動作し，高水準のプログラミングモデルを提供する．これは，アクティブスペースという概念に基づいている．実行例としては，図 3 に示したように，まずユーザ A のいるアクティブスペースを指定し，これをインスタンス化する．これは，実行時にユーザがいる場所とバインディングするという作業である．そして，アクティブスペースの中にあるアプリケーションを指定し，同じくインスタンス化する．このアプリケーションのインスタンスに対してさらにアプリケーションを起動するなどの操作を加える．本研究との差異は，Olympus モデルが能動的・静的なモデルに基づいているという点にある．開発者に十分なアクセス権とアプリケーションの実行される場所に関する情報が与えられていて，しかもリソースの分布に偏りがない場合に，Olympus モデルは開発者の要求を満たすと考えられる．しかし，Olympus は並行して実行されるタスクを想定しておらず，ユビキタス環境におけるアプリケーションの効率化には適していない．

ユビキタス環境向けアプリケーションのメンテナンス性を高めるために，AOP を利用するものも存在する．Jadabs は Spring[9]，JBoss AOP[15]などの標準的な AOP コンテナの機構をモバイル機器向けのアプリケーションに導入したものであり，アクションハンドラクラスを他の関心事から分離することが可能となる．Decorator パターンを利用してアプリケーションを構築する手法は本提案フレームワークも共通である．AOP コンテナによる支援はプログラムのロジックを保存をするために有用であるが，Jadabs では汎用的なサーバサイドの場合と同様のロジックのためにコンテナの機能を利用している．本研究では，実行状況の変化に対応する位置透過性のために利用する点で，新規性がある．

アプリケーションのプロセス（プログラム）は何種類かの関心事として分解できる．特に本研究での関心事には，リモート関心事，または分散関心事といえるものを含む．AspectJ[6),18)]に代表される，汎用的な AOP フレームワークは横断的関心事をモジュール化する機能を提供する．しかし，分散関心事に対する支援はなく，ローカルホスト上のポイントカット☆のみ定義

☆ ポイントカット（pointcut）は，プログラムの実行フロー中のある実行地点を意味するジョインポイント（join points）の集合である．ポイントカットで定義されたジョインポイントのいずれかにプログラムの実行が到達した場合，関連づけられたコード断片（アドバイス：advice）が実行される．

可能である．開発者がリモート関心事を定義する際には，特別なモジュールが分散環境への対応に必要となる．こうしたリモート関心事は，オブジェクトシリアライズ，リモートメソッド呼び出し（RMI）など分散環境向け技術で扱われる．これらの中には JAC[22]，DJ-Cutter[20]，Workflow- awareness モデル（WFA）[31] が含まれる．JAC は一貫性・同期・トランザクションをアスペクトとし，RMI を利用して織り込む．DJ-Cutter はリモートポイントカット（remote pointcut）の機構を提供する．これはリモートホストのポイントカットに透過的にアクセスするものである．これらのアプローチは，リモート関心事が事前にリモートホストに配備されている場合，有効である．実行時にプロキシサービスを決定する本研究のモデルでは，これらの手法を利用するためには制限がある．

WFA は先にあげたように，事前にリモート関心事が配備されていない状況で機能する．WFA においてリモート関心事はモバイルエージェントとして定義される．このエージェントにコミュニケーションプロトコルをすべての状態遷移に追加し，主要関心事を含む実行主体とコミュニケーション・同期を行う．互いにメッセージングを通して状態遷移の状況（Workflow）を把握する（Aware）ことができ，それをイベントとして実行を開始することなどが可能になる．これは，主要関心事を含む側のプロセスの状態遷移モデルを抽象的な形で保持し，メッセージングに応じて最新の状況を反映するように更新している．WFA の元々のコンセプトは，WFA であるエージェントを直接記述するための API を提供するというものであった．これにより，リモート関心事を処理するうえで十分と考えられる機構を提供した．しかし，つねに別のエージェントとしてモデリングすることで，非常にアドホックな作りになりがちであるということと，ローカルホスト上でも無駄なコミュニケーションを行ったりするなどパフォーマンスの問題があった．また，これを回避するための明確なモデリングプロセスに欠けていた．

本フレームワークでは，位置配置戦略が導入され静的・動的な解析が行われる．これはプロセス配置に関するモデリングの支援となる．そのうえで，不必要なコミュニケーションを回避し，リモート関心事処理がより低いオーバヘッドで行われるものと考えられる．

## 7. ま と め

本論文では，ユビキタス環境におけるプロセス配備の戦略を扱うフレームワークを提案し，プロセスの実行効率化により増加するプログラミングの負担の軽減を図った．プロセスに配備戦略（位置選好，およびスケジューリング）という概念を導入し，宣言的にプログラミングを可能にし，プログラミングの負担を軽減することを示した．特に，VE における効率化というシナリオを念頭に置いているため，位置選好情報から明確に実行効率の改善を見込める箇所に関して，分析フェーズにより改善処理を追加した．これらの拡張を実際のアプリケーションに適用した際の効果を計測・評価した．

今後の課題として，戦略記述系に関しては，パターンライブラリなどの実践的な形でプログラマの支援を追加することがあげられる．処理系に関しては，より複雑な戦略指定における処理効率化の精度を高めることと，数学的な検証とが考えられる．たとえば，LP の選好が複数候補存在する場合の扱いの精度が相当する．

## 参 考 文 献


1) Baker, J. and Hsieh, W.: Runtime aspect weaving through metaprogramming, *AOSD '02: Proc. 1st international conference on Aspect-oriented software development*, New York, NY, USA, pp.86–95, ACM Press (2002).
2) Ballesteros, F.J., Soriano, E., Leal, K. and Guardiola, G.: Plan B: An Operating System for Ubiquitous Computing Environments, *PerCom'06: International Conference on Pervasive Computing and Communications* (2006).
3) Becker, C., Handte, M., Schiele, G. and Rothermel, K.: PCOM—A Component System for Pervasive Computing, *PERCOM '04: Proc. 2nd IEEE International Conference on Pervasive Computing and Communications* (*PerCom'04*), Washington, DC, USA, p.67, IEEE Computer Society (2004).
4) Carzaniga, A.: A Characterization of the Software Deployment Process and a Survey of Related Technologies, Technical Report 97-84, Dipartimento di Elettronica e Informazione, Politecnico di Milano (1997).
5) Chakraborty, D. and Lei, H.: Pervasive Enablement of Business Processes, *PerCom'04: International Conference on Pervasive Computing and Communications* (2004).
6) eclipse.org: AspectJ. http://www.eclipse.org/aspectj/
7) EPCglobal: Object Name Service (ONS) 1.0 (2004). 7.1 Finding a WSDL file for a product.
8) Espinoza, F., Persson, P., Sandin, A., Nyström, H., Cacciatore, E. and Bylund, M.: GeoNotes: Social and Navigational Aspects of Location-Based Information Systems, *Ubi-*

*Comp '01: Proc. 3rd international conference on Ubiquitous Computing*, London, UK, pp.2–17, Springer-Verlag (2001).
9) Spring Framework. http://www.springframework.org
10) Frei, A. and Alonso, G.: A Dynamic Lightweight Platform for Ad-Hoc Infrastructures, *PerCom'05: International Conference on Pervasive Computing and Communications*, pp.373–382 (2005).
11) Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns: Elements od Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series, Addison-Wesley Publishing Company, New York, NY (1995).
12) Gribble, S.D., Welsh, M., von Behren, R., Brewer, E.A., Culler, D., Borisov, N., Czerwinski, S., Gummadi, R., Hill, J., Joseph, A., Katz, R.H., Mao, Z.M., Ross, S., Zhao, B. and Holte, R.C.: The Ninja architecture for robust Internet-scale systems and services373423, *Comput. Networks*, Vol.35, No.4, pp.473–497 (2001).
13) Grolaux, D., Roy, P. V. and Vanderdonckt, J.: Migratable User Interfaces: Beyond Migratory Interfaces, *International Conference on Mobile and Ubiquitous Systems: Networking and Services* (2004).
14) Henricksen, K. and Indulska, J.: A Software Engineering Framework for Context-Aware Pervasive Computing, *PerCom'04: International Conference on Pervasive Computing and Communications* (2004).
15) JBoss.com: JBoss AOP. http://www.jboss.com/products/aop
16) JBoss.com: JBoss jBPM. http://www.jboss.com/products/jbpm
17) JBoss.com: JBoss Remoting. http://labs.jboss.com/jbossremoting
18) Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M. and Irwin, J.: Aspect-Oriented Programming, *Object-Oriented Programming 11th European Conference* (1997).
19) Martin, R.: OO Design Quality Metrics-An Analysis of Dependencies, *Proc. Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOPSLA'94* (1994).
20) Nishizawa, M., Chiba, S. and Tatsubori, M.: Remote pointcut: a language construct for distributed AOP, *AOSD '04: Proc. 3rd international conference on Aspect-oriented software development*, New York, NY, USA, pp.7–15, ACM Press (2004).
21) Pascoe, J.: The stick-e note architecture: extending the interface beyond the user, *IUI '97: Proc. 2nd international conference on Intelligent user interfaces*, New York, NY, USA, pp.261–264, ACM Press (1997).
22) Pawlak, R.: JAC - A Framework for Aspect-Oriented Programming in Java. http://jac.objectweb.org/
23) Popovici, A., Alonso, G. and Gross, T.: Just-in-time aspects: efficient dynamic weaving for Java, *AOSD '03: Proc. 2nd international conference on Aspect-oriented software development*, New York, NY, USA, pp.100–109, ACM Press (2003).
24) Ranganathan, A., Chetan, S., Al-Muhtadi, J., Campbell, R.H. and MickunasOl, M.D.: Olympus: A High-Level Programming Model for Pervasive Computing Environments, *PerCom'05: International Conference on Pervasive Computing and Communications* (2005).
25) Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H. and Nahrstedt, K.: Gaia: a middleware platform for active spaces, *SIGMOBILE Mob. Comput. Commun. Rev.*, Vol.6, No.4, pp.65–67 (2002).
26) Romer, K., Schoch, T., Mattern, F. and Dubendorfer, T.: Smart Identification Frameworks for Ubiquitous Computing Applications, *PerCom'04: International Conference on Pervasive Computing and Communications* (2004).
27) Syukur, E., Cooney, D., Loke, S.W. and Stanski, P.: Hanging Services: An Investigation of Context-Sensitivity and Mobile Code for Localised Services, *MDM'04: International Conference on Mobile Data Management* (2004).
28) The Printer Working Group: Print Service Interface Version 1.0 Working Draft (2004). ftp://ftp.pwg.org/pub/pwg/ps/wd/wd-psi10-20040113.doc
29) van Bemmel, J., Wegdam, M. and Lagerberg, K.: 3PAC: Enforcing Access Policies for Web Services, *IEEE International Conference on Web Services* (*ICWS'05*) (2005).
30) W3C: Web Service Definition Language (WSDL). http://www.w3.org/TR/wsdl
31) 松崎和賢，吉岡信和，本位田真一：モバイルエージェントによる動的環境下のサービス利用の効率的実現手法，人工知能学会誌，Vol.19 (2004).
32) 綾塚祐二，松下伸行，暦本純一：HyperPalette：PDA を利用する複合計算機環境，WISS：インタラクティブシステムとソフトウェア VII，pp.109–

118，近代科学社 (1999).

## 付　　録

### A.1 jBPM 表現

JBoss jBPM において，プロセスは XML 形式で定義され（図 13-(i)，`processdefinition.xml`），アプリケーション本体（`MyApplication`）からインスタンス化される．`ProcessInstance` クラスの `signal` メソッドの実行により，状態遷移が行われる．このプロセスは UML 状態遷移図のように状態と状態遷移を持つ．アクションハンドラ（図 13-(ii)，`MyActionHandler.java`）は，通常の Java スレッドを実装したクラスである．本文中では単純化のため，アクションハンドラは 1 つの状態に対し 1 つだけ定義されるものとしている．

### A.2 配置戦略の記述例

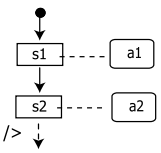
本文中の例（図 2 など）で示した動作を実現するための配置戦略の記述例を**図 14** に示す．

Main Class ( メインクラス )

```
public class MyApplication {
      public void main(){

      // プロセス定義を読み込む
      ProcessDefinition processDefinition = ProcessDefinition
            .parseXmlString("...processdefinition.xml");      ---- (i)

      // プロセスをインスタンス化する
      ProcessInstance processInstance =
                  new ProcessInstance(processDefinition);

      // 初期状態から状態 's1' に遷移する
      processInstance.signal();
      // アクションハンドラ'a1'が実行される
      // 状態 's1' から状態 's2' に遷移する
      processInstance.signal();
      // ...
      }
}
```

(i)Process Definition (processdefinition.xml)

```
<process-definition name="MyMain">
  // 右図のようなプロセス定義を行う
  <start-state name="start">
    <transition name="tr" to="s1" />
  </start-state>
  <state name="s1">
    <action name="a1"
      class="jp.ac.nii.jbpm.MyActionHandler" />
    <transition name="tr1" to="s2" />
  </state>                                          ---- (ii)
      ...
  <end-state name="end" />
</process-definition>
```




(ii)ActionHandler Definition (MyActionHandler.java)

```
public class MyAction implements ActionHandler{
      public void execute(){
            // 実行するタスクを記述
      }
}
```

**図 13** JBoss jBPM の表記によるプロセス記述例
Fig. 13 Application model (the JBoss jBPM notation).

```
1  <?xml version="1.0" encoding="Shift_JIS"?>
2  <tns:locationPreference>
3    <lpelement id="lpe-1">
4  <!-- 状態集合に含まれる状態A -->
5      <state>state_k</state>
6      <state>state_k_1</state>
7  <!-- 位置指定 -->
8      <location id="1">
9        <local_proxy>
10 <!-- 実行開始直後に具体的なアドレスを解析 -->
11         <resolveSpec id="rs1" resolver="default">
12           <enter>start</enter>
13         </resolveSpec>
14 <!-- アドレス解析終了後にプロセス定義を転送 -->
15         <processDefTransfer id="pdt1" preCond="rs1" />
16 <!-- プロセス転送が済み，かつ状態に入った時にコンテキスト変
   数の同期を行うk -->
17         <contextTransfer preCond="pdt1">
18           <enter>state_k</enter>
19         </contextTransfer>
20       </local_proxy>
21 <!-- 複数定義可能．投機的実行などに利用．優先度は記述順 -->
22       <local_proxy>...</local_proxy>
23 ...
```

**図 14** 配置戦略の記述例
Fig. 14 An example description of the deployment strategy.

(平成 18 年 4 月 3 日受付)
(平成 18 年 10 月 3 日採録)

**松崎　和賢**

1979 年生．2002 年東京大学理学部情報科学科卒業，2004 年東京大学大学院情報理工学系研究科コンピュータ科学専攻修士課程修了，同年同博士課程進学，文部科学省国立情報学研究所知能システム研究系リサーチアシスタント，エージェント技術の研究に従事．現在に至る．日本ソフトウェア科学会学生会員．

**本位田真一**（正会員）

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 年より同研究所研究主幹を併任，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併任，現在に至る．2002 年 5 月〜2003 年 1 月英国 UCL ならびに Imperial College 客員研究員（文部科学省在外研究員）．2005 年度パリ第 6 大学招聘教授．早稲田大学客員教授．工学博士（早稲田大学）．1986 年度情報処理学会論文賞受賞．ソフトウェア工学，エージェント技術，ユビキタスコンピューティングの研究に従事．IEEE，ACM，日本ソフトウェア科学会等各会員．情報処理学会理事．