情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-DPS-145 No.9
Vol.2010-GN-77 No.9
Vol.2010-EIP-50 No.9
2010/11/25

アスペクト指向フレームワークを用いた複製オブジェクトの振る舞い同期の実現

鈴本　悟[†1]　高田秀志[†2]

リアルタイムな協調作業支援アプリケーションを実現する手段として，各端末に配置した複製オブジェクトの振る舞いを同期する P2P 型複製オブジェクト環境がある．この振る舞い同期は，代理オブジェクトを介してメソッド呼び出しを行うことによりにより実現可能である．代理オブジェクトを介して振る舞い同期をするには，開発者が代理オブジェクトの導入に関するソースコードの変更や再コンパイルをしなければならない．これにより，開発者が代理オブジェクトを介したメソッド呼び出しに変更し忘れたことにより振る舞い同期が行われないバグが発生し得る．本稿では，開発者が複製オブジェクトを意識することなく複製オブジェクトの振る舞い同期を実現することを可能にするアスペクト指向フレームワークを用いた複製オブジェクトの振る舞い同期手法を提案する．

Synchronization of Behaviors on Replicated Objects Using the Framework of Aspect-Oriented Programing

Satoru Suzumoto[†1] and Hideyuki Takada[†2]

As a method to realize real-time cooperative work systems, there is the P2P-based replicated objects environment which synchronizes the behaviors of replicated objects allocated on every terminal. Synchronization of behaviors on replicated objects is realized by method invocation via the proxy object. In order to apply proxy object, developers must change source codes and recompile them. If developers forget to change source codes so that methods can be invoked via proxy object, other terminals can't synchronize replicated objects. This research proposes a method of synchronizing behaviors on replicated objects using a framework of Aspect Oriented Programming in order to solve the problems of invocation with proxy object. This method allows developers to write source codes synchronizing replicated objects without paying attention to replicated objects.

1. はじめに

リアルタイムな協調作業支援アプリケーションは, 教育や会議など幅広い分野で利用されはじめている．このようなアプリケーションの実現手法の一つとして，協調作業を行う各端末へ複製したオブジェクトを配置する複製計算モデル[1] に基づいた P2P 型複製オブジェクト環境が挙げられる．P2P 型複製オブジェクト環境上に構築されたリアルタイムな協調作業支援アプリケーションでは，各端末におけるアプリケーション状態の同期を複製オブジェクトの振る舞い同期により実現する．我々は，この P2P 型複製オブジェクト環境を構築するためのフレームワーク「CUBE(Collaborative Universal Basic Environment)」[2] の構築を行っている．CUBE では，代理オブジェクトを用いて複製オブジェクトの振る舞いを同期している．これは，Java の Proxy クラスのように代理オブジェクトを実装するための機能が提供されている場合，Android[⋆1]が搭載された携帯端末と PC が混在しているような環境でも，複製オブジェクトの振る舞い同期が可能である．

P2P 型複製オブジェクト環境の利点として，スタンドアロンアプリケーションを協調作業支援アプリケーションへ拡張することが容易になることが考えられる．この拡張をクライアント・サーバモデルで実現しようとした場合は，開発者がアプリケーションの機能をサーバ側の機能とクライアントがの機能へ分離する必要がある．P2P 型複製オブジェクト環境の場合は，各端末に同一のプログラムを配置するので，開発者がアプリケーションの機能をクライアントサーバモデルのように分離する必要がない．特に，P2P 型複製オブジェクト環境では，プログラムの本質的な機能を変更することなく，複製オブジェクトの振る舞い同期が行われるようにプログラムを変更するだけでよい．

CUBE で採用している代理オブジェクトを介した振る舞い同期の問題点として，複製オブジェクトの振る舞い同期をするためには，代理オブジェクトの生成や代理オブジェクトを介したメソッド呼び出しを行う必要があるので，開発者によるアプリケーションのソースコードの変更・再コンパイルが必要になることが挙げられる．加えて，開発者が，代理オブ

†1 立命館大学大学院 理工学研究科
Graduate School of Science and Engineering, Ritsumeikan University
†2 立命館大学 情報理工学部
College of Infromation Science and Engineering, Ritsumeikan University
⋆1 http://www.android.com/

Vol.2010-DPS-145 No.9
Vol.2010-GN-77 No.9
Vol.2010-EIP-50 No.9
2010/11/25

ジェクトを介さないメソッド呼び出しを誤ってソースコードに記述してしまうと，各端末間で複製オブジェクトの振る舞いが同期されない問題が発生する．

本稿では，開発者が複製オブジェクトを意識することなく，複製オブジェクトの振る舞い同期を実現させるために，アスペクト指向プログラミングを実現するフレームワークを用いた複製オブジェクトの振る舞い同期手法を提案する．

2. P2P 型複製オブジェクト環境

2.1 複製オブジェクト

P2P 型複製オブジェクト環境では，複製オブジェクトを各端末に配置し，その振る舞いを同期することによって各端末のアプリケーションの状態を同期する．複製オブジェクトのインスタンスは，オブジェクト ID によって各端末間で対応付けがされ，各端末のオブジェクトマネージャで管理されている．この複製オブジェクトは，振る舞い同期対象となるメソッドが定義されたインタフェースをインプリメントしている．これにより，Android が搭載された携帯端末と PC でオブジェクトの実装が異なっているような場合でも，インターフェースを介して振る舞い同期が可能である．

CUBE では，複製オブジェクトの振る舞い同期が代理オブジェクトを介したメソッド呼び出しにより行われる．本稿では，この仕組みをオブジェクトミラーリングと呼ぶ．このオブジェクトミラーリングを利用した振る舞い同期は，図 1 で示したように実行される．図 1 中の端末 A で代理オブジェクトを介したメソッド呼び出しが発生すると，代理オブジェクトは，メソッドが呼び出された端末の複製オブジェクトのメソッドを呼び出す．それと同時に，代理オブジェクトは他端末に対しメソッドの呼び出しを通知する．他端末のオブジェクトマネージャは，このメソッド呼び出しの通知を受信すると該当する複製オブジェクトに対し，同じメソッド呼び出しを行う．

P2P 型複製オブジェクト環境で協調作業支援アプリケーションを実現する場合は，クライアントサーバ・モデルのようにアプリケーションの機能をサーバとクライアントに分離する必要がない．また，複製オブジェクトの振る舞い同期により協調作業を実現するため，スタンドアロンアプリケーションを協調作業支援アプリケーションに拡張する場合は，アプリケーションの変更箇所を振る舞い同期の実現部分のみにとどめることが可能である．

2.2 P2P 型複製オブジェクト環境上のアプリケーション例

P2P 型複製オブジェクト環境の適用例として，CUBE を基盤とした児童向け協調学習環境 SnowBoy[3] がある．SnowBoy は，児童らが 3D グラフィックの作成と作成した 3D グラフィックに対するプログラムの付与を協調作業により行う．SnowBoy による協調作業では，図 2 で示すように児童らが同じ部屋で各端末を操作し協調作業を行う．CUBE の適用アプリケーションとしては，このように各々が端末を持ち寄り同じ部屋で協調作業を行うためのアプリケーションを想定している．

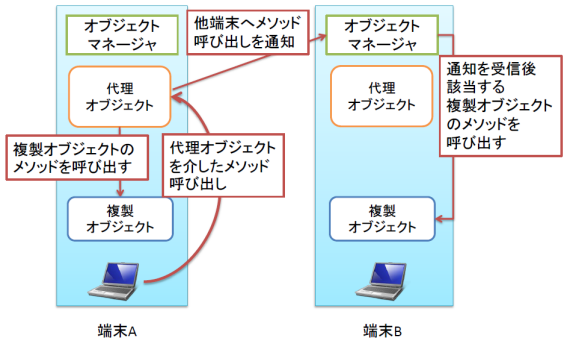

図 1 オブジェクトミラーリング

図 2 SnowBoy による協調作業

2.3 代理オブジェクトの問題点

代理オブジェクトを介した複製オブジェクトの振る舞い同期では，開発者が代理オブジェクト生成コードの追加をしたり，振る舞い同期対象のメソッド呼び出しを代理オブジェクト経由のメソッド呼び出しへ変更したりしなければならない．そのため，振る舞い同期対象の

Vol.2010-DPS-145 No.9
Vol.2010-GN-77 No.9
Vol.2010-EIP-50 No.9
2010/11/25

メソッド呼び出しがソースコードの各所で行われていると，開発者によるソースコードの変更箇所が多くなる．これにより開発者は，ソースコードの再コンパイルが必要になるほか，開発者が代理オブジェクトを介したメソッド呼び出しに変更し忘れると，複製オブジェクトのメソッド呼び出しが他端末に通知されないといったバグが発生する．したがって，開発者が，複製オブジェクトを意識しなくても複製オブジェクトの振る舞い同期が行われるようにするための仕組みが必要とされる．

3. 振る舞い同期手法

複製オブジェクトには，そのインスタンスのオブジェクト ID があらかじめ決定されている静的な複製オブジェクトと，オブジェクト ID があらかじめ決定されていない動的な複製オブジェクトが存在する．静的な複製オブジェクトは，インスタンス生成時に他端末へその生成を通知する必要がない．一方，動的な複製オブジェクトについては，オブジェクト ID がオブジェクトのインスタンス生成時に決定されるため，他端末へ複製オブジェクトのインスタンス生成を通知する必要がある．各端末に配置された複製オブジェクトは，ある端末で呼び出された振る舞い同期対象のメソッドが他端末に通知されることで同期される．

P2P 型複製オブジェクト環境では，上記で挙げた複製オブジェクトの登録・メソッド呼び出しの通知が，さまざまなオブジェクトを横断して必要とされる．このようなさまざまなオブジェクトを横断して実行される処理を扱うものとして「アスペクト指向プログラミング」[4)5)] がある．本稿では，これを用いて複製オブジェクトの振る舞い同期を実現する手法を提案する．

3.1 アスペクト指向プログラミング

アスペクト指向プログラミングでは，複数の機能により利用される機能を横断的関心事として分離することで，保守や再利用を容易にする．この横断的関心事が，メソッドやコンストラクタなどに織り込まれることでそれらの本質的な機能を変更することなく．機能の追加や変更が可能となる．横断的関心事として実行時に具体的な処理として組み込まれる処理は，インターセプタあるいはアドバイスと呼ばれる．このインターセプタを組み込むことのできる場所をジョインポイントと呼び，ジョインポイントとしては，コンストラクタやメソッドなどがある．どのジョインポイントにどのインターセプタを組み込むのかの定義は，ポイントカットと呼ばれる．ジョインポイントにインターセプトを組み込む処理は，ウィービングと呼ばれる．このウィービングを行うタイミングには，コンパイル時，クラスロード時，実行時の 3 種類がある．

3.2 アスペクトを利用した振る舞い同期

アスペクト指向プログラミングを用いた複製オブジェクトの振る舞い同期は，複製オブジェクトとなるクラスのコンストラクタや振る舞い同期対象のメソッドへインターセプタが組み込まれることにより実現される．これにより，代理オブジェクトの生成や代理オブジェクトを介したメソッド呼び出しがソースコードに記述されることなく，複製オブジェクトのインスタンス生成や振る舞い同期対象のメソッド呼び出しをスタンドアロンアプリケーションと同様に行うだけで，複製オブジェクトの振る舞い同期が可能になる．

アスペクト指向プログラミングを利用した複製オブジェクトのインスタンス生成は，図 3 のように行われる．以下にその手順を説明する．

(1) 複製オブジェクトのインスタンス生成を行うため，端末 A で複製オブジェクトのコンストラクタが呼び出される．
(2) 元のコンストラクタの処理が終了したのち，生成されたインスタンスがコンストラクタへ組み込まれたインターセプタによってオブジェクトマネージャに登録される．静的な複製オブジェクトのインスタンスが生成される場合は，ここで処理が終了する．
(3) 複製オブジェクトのインスタンス生成通知がコンストラクタへ組み込まれたインターセプタによって他の端末へ通知される．
(4) 通知を受信した端末 B は，受信した通知から複製オブジェクトのインスタンスの生成を行い，生成されたインスタンスをオブジェクトマネージャに登録する．このとき，複製オブジェクトのコンストラクタが実行されるため，コンストラクタに組み込まれたインターセプタの処理が端末 B でも同様に実行されてしまう．したがって，インターセプタは，他端末からの通知による実行かどうかの判別を行い，他端末からの通知であれば生成されたインスタンスの登録および他端末への通知は実行しない．

このようにして生成された複製オブジェクトの振る舞い同期は，図 4 のように行われる．以下にその手順を説明する．

(1) 端末 A で振る舞い同期を行う複製オブジェクトのメソッドが呼び出される．
(2) 元のメソッドの処理が終了したのち， 振る舞い同期を行うメソッドに組み込まれたインターセプタが他の端末へ同じメソッド呼び出しの実行を通知する．
(3) メソッド呼び出しの実行通知を受信した端末 B は，受信した通知から同じメソッド呼び出しを行う．このとき，振る舞い同期対象となるメソッドの実行が行われるため，端末 B のメソッドに組み込まれたインターセプタが端末 B でも同様に実行されてしまう．したがって，インターセプタは，コンストラクタが呼び出された場合と同様に

Vol.2010-DPS-145 No.9
Vol.2010-GN-77 No.9
Vol.2010-EIP-50 No.9
2010/11/25

他端末からの通知による実行かどうかの判別を行い，他端末からの通知であれば他端末へのメソッド実行通知はしない．

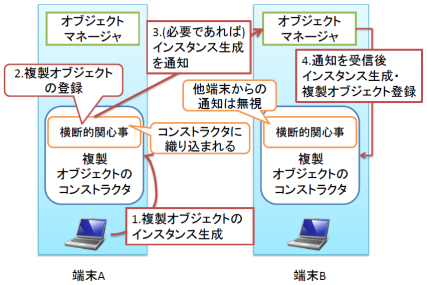

図 3 AOP による複製オブジェクトの生成

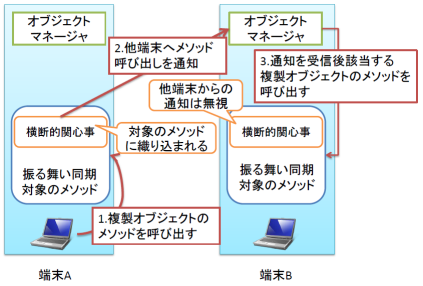

図 4 AOP による振る舞い同期

4. 実　　装

4.1 振る舞い同期の実現

アスペクト指向プログラミングを実現する手段として，言語としてアスペクト指向をサポートしている AspectJ[5] やフレームワークの機能の一部として実装されている Seaser2[⋆1] などがある．今回は，コンストラクタに対してジョインポイントを持ち，クラスロード時にウィービングを行うためソースコードの再コンパイルが不要な SamuraiAOP を利用して，CUBE における複製オブジェクトの振る舞い同期を実現した．SamuraiAOP では，ポイントカットとなるメソッドやコンストラクタに対して組み込むインターセプタの指定を XML ファイルに記述する．インターセプタの指定では，各ポイントカットに対応したインターセプタのクラスから生成されたインスタンスの取得方法を記述する．

本稿でこれらのインスタンスを生成するクラスは，AOPAlliance[⋆2]が提供するインタフェースをインプリメントしている．インターセプタのインスタンスは，cmlab.cube.interceptors.CUBEInterceptors に定義されているメソッドにより取得される．CUBE を利用するために必要なインターセプタは，以下のようになる．

⋆1 http://www.seasar.org/
⋆2 http://aopalliance.sourceforge.net/

CreateObjectWithNotification
ConstructorInterceptor をインプリメントしており，オブジェクトマネージャへの登録を行うとともに，他端末に対してインスタンスの生成通知を行う．このインスタンスは，getCreateObjectWithNotification() により取得される．

CreateObjectWithoutNotification
ConstructorInterceptor をインプリメントしており，オブジェクトマネージャへの登録を行うが，他端末に対してインスタンスの生成通知を行わない．このインスタンスは，getCreateObjectWithoutNotification() により取得される．

MethodInvocation
MethodInterceptor をインプリメントしており，振る舞い同期を行うために他端末へメソッド呼び出しを通知する．このインスタンスは，getMethodInvocationInstance() により取得される．

StartCUBE
MethodInterceptor をインプリメントしており，CUBE のオブジェクトマネージャやネットワーク通信を開始する．このインスタンスは，getStartCUBEInstance() により取得される．複製オブジェクトの振る舞い同期をするには，ネットワークなどの機能があらかじめ起動されている必要があるので，main() メソッドへ CUBE のオブジェクトマネージャやネットワーク通信を開始するインターセプタを組み込む．

4.2 アプリケーションへの適用

本節では，4.1 節で実装した複製オブジェクトの振る舞い同期の適用方法を簡単なグラフ構造エディタにより説明する．図 5 のクラス図は，適用したグラフ構造エディタのクラス図である．今回の適用で複製オブジェクトとして扱うクラスは，**MouseCheck** であり，振る舞い同期の対象となるメソッドは，**mouseClicked()** および **MouseDragged()** である．

リスト 1 で示された XML ファイルは，複製オブジェクトのクラスと振る舞い同期を行うメソッドの指定，および，CUBE の機能をあらかじめ起動させるために main() メソッドの指定を行っている．5 行目から 12 行目は，複製オブジェクトと振る舞い同期に関する指定である．まず，複製オブジェクトとして扱うクラスは，5 行目の aspect タグの属性である target で指定を行う．次に，振る舞い同期対象となるメソッドの指定は，6 行目の pointcut タグの属性である name で指定を行う．この pointcut タグで指定されたメソッドに対して振る舞い同期を行うインターセプタの指定は，7 行目で行われている．複製オブジェクトの生成・登録を行うインターセプタを組み込むため，ポイントカットにコンストラクタを指定

Vol.2010-DPS-145 No.9
Vol.2010-GN-77 No.9
Vol.2010-EIP-50 No.9
2010/11/25

する場合は，9 行目のように pointcut タグの属性である name へ **this** と指定する．今回のアプリケーションでは，MouseCheck のオブジェクト ID があらかじめ決定されていてもよいので，10 行目で複製オブジェクトの登録のみを行うインターセプタを指定している．13 行目から 17 行目は，CUBE を利用するために CUBE の初期化処理を main() メソッドへ組み込む記述である．このアプリケーションでは，MindMap クラスに main() メソッドがあるため 13 行目でクラス，14 行目でメソッド，および 15 行目でインターセプタの指定を行っている．

このように XML ファイルで複製オブジェクトのクラスや振る舞い同期対象のメソッドを指定することで，開発者は，ソースコードを変更することなくスタンアロンアプリケーションを協調作業支援アプリケーションに拡張できる．

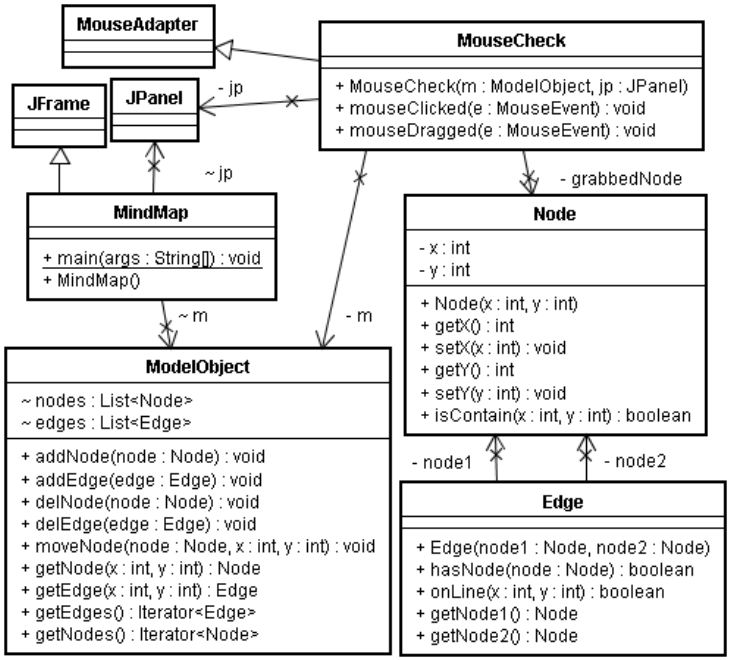

図 5 サンプルアプリケーションのクラス

リスト 1　XML ファイルによる複製オブジェクトの指定

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE aspect-config PUBLIC "-//SAMURAI_AOP//DTD␣samurai-aop␣1.0//EN"
3         "http://dodododo.jp/dtd/samurai-aop10.dtd">
4  <aspect-config>
5    <aspect target="sample.mindmap.MouseCheck">
6      <pointcut name="mouseClicked,mouseDragged">
7        cmlab.cube.interceptors.CUBEInterceptor.getMethodInvocationInstance()
8      </pointcut>
9      <pointcut name="this">
10       cmlab.cube.interceptors.CUBEInterceptor.getCreateObjectWithoutNotificationInstance()
11     </pointcut>
12   </aspect>
13   <aspect target="sample.mindmap.MindMap">
14     <pointcut name="main">
15       cmlab.cube.interceptors.CUBEInterceptor.getStartCUBEInstance()
16     </pointcut>
17   </aspect>
18 </aspect-config>
```

5. 議　　論

5.1 関 連 研 究

開発者が，ソースコードを変更・再コンパイルせずにスタンドアロンアプリケーションを協調作業支援アプリケーションに拡張する研究として Flexible JAMM[6] がある．Flexible JAMM では，シングルユーザ向けであるユーザインタフェースのコンポーネントをマルチユーザ向けのユーザインタフェースのコンポーネントへ置き換えることにより，元のアプリケーションのソースコードを変更することなく，協調作業支援アプリケーションに変更する．したがって，Flexible JAMM では，シングルユーザ向けのコンポーネントを置き換えるマルチユーザ向けのコンポーネントを用意する必要がある．一方，本稿で提案した手法では，特殊なコンポーネントを用意することなく振る舞い同期を行うメソッドを指定するのみで協調作業支援アプリケーションが実現できる．

他の研究としては，Transparent Adaptation(TA) approach[7] による CoWord，CoPPT がある．TA approach では，アプリケーションの API を利用してユーザの操作を捕捉し，ユーザの操作を別の端末で再現することにより，アプリケーション自体のソースコードを変更することなくスタンドアロンアプリケーションを協調作業支援アプリケーションへ拡張す

Vol.2010-DPS-145 No.9
Vol.2010-GN-77 No.9
Vol.2010-EIP-50 No.9
2010/11/25

る．しかし，TA approach では，対象となるアプリケーションがユーザの操作を捕捉可能な API を提供している必要がある．加えて，開発者は，ユーザの操作の捕捉・再現に関して API によるプログラム開発も必要である．本稿で提案した手法では，このような API を利用することなく，XML ファイルに振る舞い同期を行うメソッドを記述するのみで協調作業支援アプリケーションへの拡張が可能である．

5.2 検討事項

本稿で提案した手法では，SamuraiAOP を用いているためリスト 1 で示した XML ファイルを記述しなければならない．これにより，開発者は複製オブジェクトの生成や振る舞い同期を行うメソッドにインターセプタを組み込むことを明示的に設定ファイルへ記述する必要がある．また，この XML ファイルにはアスペクト指向プログラミングに関する情報のみが記述されることから，ネットワークなどの設定は別の設定ファイルに記述しなければならない．この問題に対しては，クラスロード時にバイトコードを動的に変更できる Java の Instrument を用いて独自に振る舞い同期を実現することで，P2P 型複製オブジェクト環境に合わせた形式の設定ファイルの利用が可能になると考えられる．

今回の実装では，複製オブジェクトとなるクラスや振る舞い同期対象のメソッドを指定する設定ファイルは，開発者がテキストエディタによって記述を行う必要がある．これは，設定ファイルの記述ミスが存在してもインターセプタの指定などでエラーチェックが行われないため，バグの発生につながる．したがって，対象アプリケーションのクラスやメソッド一覧を表示して複製オブジェクトとなるクラスや振る舞い同期対象のメソッドを指定したのち，設定ファイルを生成するツールが必要になると考えられる．

4.2 節では，マウスの操作を表すオブジェクトを振る舞い同期することで協調作業アプリケーションへの変更を実現した．もし，この振る舞い同期が Node や Edge などアプリケーションの状態を表すオブジェクトで振る舞い同期を行うように変更されたと仮定する．このとき，Node や Edge などのアプリケーション状態を表すオブジェクトはユーザインタフェースを表すオブジェクトへ参照を持たないため，変更を監視する手段が提供されていない場合，ユーザインタフェースへその変更を通知するのが困難になる．これは，アプリケーションの状態とユーザインタフェースに一貫性が取れていないという問題を引き起こす．このように，P2P 型複製オブジェクト環境におけるソフトウェアモデルについても検討を行う必要がある．

6. おわりに

本稿では，アスペクト指向を用いた複製オブジェクトの振る舞い同期の実現方法を提案した．また，アスペクト指向フレームワークの一つである SamuraiAOP により代理オブジェクトを介さない複製オブジェクトの振る舞い同期を実現した．これにより，開発者は，設定ファイルである XML を記述するだけで，複製オブジェクトを意識することなく複製オブジェクトの振る舞い同期を実現することが可能である．

今後は，開発者がより簡単に P2P 型複製オブジェクト環境上へアプリケーションを構築できるように，フレームワークの改良や開発者支援ツールの構築をしていきたいと考えている．

参考文献

1) David P. Reed: Designing Croquet ’s TeaTime - A Real-time, Temporal Environment for Active Object Cooperation OOPSLA 2005, 2005.
2) Shogo Noguchi and Hideyuki Takada: Cube: A Synchronous Collaboative Applications Platform Based on Replicated Computation, Proceedings of the Fifth International Conference on Collaboration Technologies (CollabTech 2009), pp. 19–24, 2009.
3) 取越 翔太郎, 柿内 達真, 植田 亘, 桜打 彬夫, 後藤 清豪, 高田 秀志, 藤原 央樹, 森口 友也, 山本 佑樹: 3D グラフィクスの共同創作機能を備えた子ども向けプログラミング環境 SnowBoy の構築とその評価, インタラクション 2010, PA11, 2010.
4) Popovici, Andrei and Gross, Thomas and Alonso, Gustavo: Dynamic weaving for aspect-oriented programming, Proceedings of the 1st international conference on Aspect-oriented software development(AOSD ’02), pp. 141–147, 2002.
5) Gregor Kiczales, Erick Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm and William G. Griswold: An Overview of AspectJ, European Conference on Object-Oriented Programming (ECOOP 2001), pp. 327–357, 2001.
6) Begole, James and Rosson, Mary Beth and Shaffer, Clifford A: Flexible collaboration transparency: supporting worker independence in replicated application-sharing systems, ACM Transactions on Computer-Human Interaction, Vol.6, No.2, pp. 95–132, 1999.
7) Chengzheng Sun, Steven Xia, David Sun, David Chen, Haifeng Shen and Wentong Cai: Transparent Adaptation of Single-User Applications for Multi-User Real-Time Collaboration, ACM Transactions on Computer-Human Interaction, Vol.13, No.4, pp. 531–582, 2006.