5ZC-2

携帯ゲームアプリケーション開発を支援する オブジェクト指向を用いたフレームワークの開発

橋山　牧人 †　　中鉢　欣秀 ‡　　大岩　元 ††
慶應義塾大学　政策・メディア研究科 †　　産業技術大学院大学 ‡　　慶應義塾大学　環境情報学部 ††

1　はじめに

1.1　携帯ゲームの規模の拡大による開発手法の変化

携帯ゲームアプリケーション（以下，携帯ゲームと書く）を搭載した携帯電話が日本国内に普及しはじめた 2001 年頃は，アプリケーションサイズが 10KB（キロバイト）に制限されており，サイズを小さく抑えて開発する必要があった．そのため，携帯ゲーム開発者は開発効率よりも軽量化を重視したコーディングスタイルをとっていた．

しかしながら，2006 年頃からメガアプリと呼ばれる容量が 1MB（メガバイト）を超える携帯ゲームが実行可能になり，ソースコードの軽量化を行なう必要はなくなった．また，携帯アプリの規模が大きくなったことで，過去には 1 人で可能であった開発作業を複数人で行なうことも多くなった．

ところが，携帯ゲームの開発現場では過去の習慣が抜けておらず，オブジェクト指向による再利用性を意識したコーディングとは無縁であり，構造化も行わず，ソフトウェア工学的な視点から携帯ゲームの開発効率を向上させる試みを行うことは少ない．さらに，携帯ゲームでは習慣として曖昧な仕様書や口頭でのやり取りのみで開発することも多い．そのため，ソースコードのみが唯一の仕様書であるといったケースもあり，可読性の低いソースコードは開発や保守の作業を妨げる原因となっている．

1.2　携帯ゲーム開発の問題点と解決策の提案

ゲームアプリケーションの多くは，ゲーム内の進行状況に応じて，1) ユーザの入力に対する反応，2) 必要な計算，3) 画面表示，という処理を切り替える必要がある．しかし，現在の携帯ゲームの開発では，これらの処理がゲームの画面と関連してまとまっておらず，ソースコード上での線引きが曖昧である．その結果，ゲーム全体の進行を管理する処理と，個々の画面で行う処理が密接に結合しているため，画面の拡張性に乏しい．

また，画面の数が増加するに連れてソースコードが複雑になっていき，開発や保守の効率が悪化する．さらに，ゲーム全体の動作構造も開発者に依存しており，複数のゲームで同じ構造を共有したり，再利用したりすることが難しい．

我々は，これらの問題を解決するために，オブジェクト指向と状態遷移を用いることでゲーム全体の構造を抽象化し，外部化した．また，煩雑なゲーム画面の管理と遷移を行うために，ゲーム内の個々の画面を分割して管理することができる仕組みを考案し，実装した．

本稿では，それらの機能を持つアプリケーションフレームワーク Bridge*について述べる．

2　Bridge

2.1　ゲーム全体の構造の外部化

携帯ゲームのプラットフォーム仕様の多くは，イベントドリブン方式で設計してある．イベントドリブン方式は，処理の発生するタイミングをイベントに委ねるため，プログラマが意図しない間隔でイベントを通知することがある．そのため，常に描画しつづけており，ユーザのキー操作に敏感に反応する必要がある携帯ゲームでは，キー操作に描画が追いつかずにコマ落ちの問題が発生することがある．

そこで，携帯ゲーム開発者はこの問題を回避するために，一般的にゲームループ方式を利用する．ゲームループ方式では，プログラムがポーリングによりハードウェアの状態を断続的に問い合わせ，特定のタイミングにおけるアクションの発生有無を確認し，条件分岐によって実行する．ゲームループ方式では処理の実行間隔をプログラマが制御できるので，描画とキー処理の順番が食い違うことで発生するコマ落ちの問題は解決できる．

我々は，携帯ゲームのプラットフォーム仕様が提供するイベントドリブン構造を，携帯ゲームに適したゲームループ構造に書き換えた．また，Eclipse[1] のプラグインによるスケルトンコード生成機能を開発し，抽象化したゲームループ構造を簡単に利用できるように外部化した．

Object-Oriented Framework for Mobile Game Application
Makito Hashiyama †, Yoshihide Chubachi ‡, Hajime Ohiwa ††
†Graduate School Media and Governance, Keio University
‡Advanced Institute of Industrial Technology
††Faculty of Environmental Information, Keio University

*http://www.crew.sfc.keio.ac.jp/ hashiyaman/bridge/

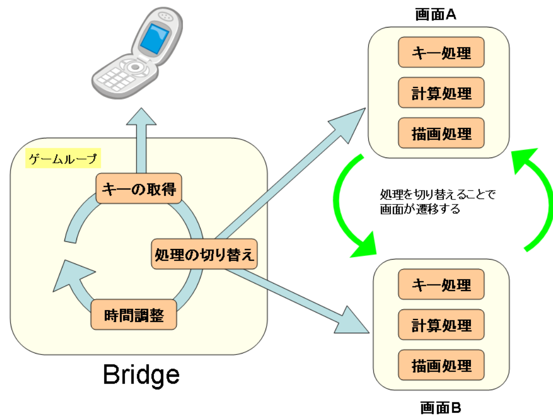

図 1: Bridge のアーキテクチャ

2.2 状態遷移による画面管理

ゲームは通常複数の画面で構成してあるため，各画面で行う描画やキー操作の処理を管理する必要がある．複数の画面を管理する最も単純な方法は，あるクラスに現在の画面を表す変数を用意し，それに応じた分岐処理を記述することである．しかし，この方法では各画面で行う処理や画面数が増大した場合に，1 つのクラスにすべての画面の処理が集中するため，保守性が著しく低下する．また，画面を追加する際に既存の分岐処理を見直さなければならないので，拡張が難しい．

Bridge では画面を状態オブジェクトとして捉えて，State パターン [2][3] を用いることでこの問題を解決した．図 1 は Bridge のアーキテクチャを表している．State パターンにより画面に依存する処理を画面を表すクラス内に隠蔽することで，煩雑な条件分岐文をなくし，コードの可読性を向上させた．各画面は，Bridge が呼び出すインターフェースで定義してあるメソッドに従って画面固有の処理を記述する．これにより，定数管理では 1 つのクラスに集中していた画面の処理を画面ごとに分散することができ，ポリモーフィズムによって画面を動的に切り替えることが可能である．画面を新しく追加する場合も，既存のソースコードは修正する必要はないため，拡張が容易である．

3 評価と考察

Bridge を用いて，300 ステップほどの規模の携帯ゲームを開発する試用実験を行なった．被験者には，携帯ゲームの仕様書を配布し，仕様書どおりのゲームが完成するまでにかかった時間を測定した．なお，実装時間が 3 時間を超えた場合は実装を打ち切り，実装不可とした．

表 1 は被験者の携帯ゲーム開発経験と，試用実験における実装にかかった時間を表したものである．被験者 A~D はまず Bridge を利用しないで実装を行い，その後で Bridge を利用して実装を行なった．被験者 E~H はその逆順で実装を行なった．この結果から，Bridge を利用した場合の実装時間が平均 30 分程度短くなった．

表 1: 試用実験における実装時間の比較

被験者	携帯ゲーム開発経験	Bridge 未利用時	Bridge 利用時
A	なし	81 分	37 分
B	あり	135 分	88 分
C	なし	-（実装不可）	116 分
D	なし	-（実装不可）	70 分
E	なし	70 分	63 分
F	なし	163 分	101 分
G	なし	110 分	67 分
H	なし	89 分	87 分
I	あり	114 分	90 分
平均		108.9 分	79.9 分

また，被験者から「Bridge を使うことで実装時間が大幅に短縮できただけでなくプログラム自体も見やすくなり，変更も容易に行なえると思う」という感想もあり，Bridge の画面管理機能によって開発効率や保守性が向上していることが伺えた．

4 まとめ

試用実験を通じて，携帯ゲームの開発経験に関わらず，Bridge を利用することで小規模の携帯ゲームを短時間で開発できたため，携帯ゲーム開発の敷居を下げることにも十分な効果があることが分かった．Bridge が普及することで，誰にでも高品質のゲームが効率良く開発できるようになれば，携帯ゲーム市場のさらなる活性化が期待できる．

最後に，Bridge の問題提起と研究の支援をしていただいた，株式会社インテムの金澤徹社長に感謝します．

参考文献

[1] Eclipse Project. Eclipse. http://www.eclipse.org/, 2008.

[2] Erich Gamma, et al. オブジェクト指向における再利用のためのデザインパターン. ソフトバンククリエイティブ, 2006.

[3] Ralph E. Johnson, et al. パターンとフレームワーク. 共立出版, 1999.