# Rose-ZIPC++連携による携帯電話アプリケーション開発

パナソニックMSE株式会社　ユビキタスネットワーク事業部
ICT開発グループ　ICT推進チーム
小林　圭介

## 1．移動体端末のソフト開発の現状

近年の移動体端末は、多機能化が進んでおり、移動体端末を動かしているソフトウェアの開発量も膨大となってきている。そのため、膨大な量のソフトウェアをベースとして開発を行う必要があり、解析やテストにより多くの工数が必要となる。また、1モデルの開発期間がこれまでと変わらない現状では、機能追加に要する工数をより削減することが求められている。

移動体端末における開発では、上流工程では仕様が確定しづらいのが現状である。また、仕様が決まったとしても後工程で仕様変更が発生することを考慮し、仕様変更を見越した開発の進め方を行っていく必要がある。近年では一定期間内で複数の機種を市場に投入していくことが求められているため、実際の開発では複数機種の差分部分のみを同時並行開発している。そのため、ある開発で発生した問題が流用した他機種の開発に影響を与えてしまうリスクに関しても今まで以上に増加している。複数の体制が並行している状態で開発を行うためには、成果物のルールの徹底や、統合のしやすさなどが開発を行っていく上でポイントとなる。

市場が飽和しつつある移動体端末業界では『ブランド』が重要であり、ユーザが求めている機能をいち早く機種へ反映し、かつ、高品質な状態で市場へ送り出すことが課題となってくる。

## 2．ツール導入の目的

今回の開発で設計ツールを使用した目的を以下に記述する。

### （1）IBM Rational Roseを利用した目的

複数人で開発を行っていく際に、各メンバのスキルによって手動で生成されたソースコードの品質にばらつきがある問題が以前にあった。そのため、ソースコードレビューの効率が悪くなってしまうことが起きていた。

また、開発ではMicrosoft Word [1]（以下、Wordと記す）による設計ドキュメントの作成作業がある。開発現場からは、設計ドキュメントをクラス設計と連動してほしいという声もあった。その結果、今回の開発では、クラス設計、状態遷移設計にIBM Rational Rose [2]（以下、Roseと記す）を使用し、ドキュメントの作成はRoseと連携可能なドキュメント生成ツールであるIBM Rational SoDA for Word（以下、SoDAと記す）を使用することに決めた。

### （2）ZIPC++を利用した目的

移動体端末のアプリケーションは、割り込み動作が多く、アプリケーション間の連携やミドルウェアの制御が非常に複雑となる。また、大量の状態を扱っているため、処理のもれや抜けのチェックは人の手でやるのではなく、状態遷移表を作成して効率的にチェックしていく必要があった。さらに、状態設計からソースコードを生成する場合でも、安定した品質のソースコードが必要となる。以上のことから、状態設計はRose、状態設計のチェックは、Roseと連携可能なZIPC++を使用することに決めた。

## 3．ツール導入事例

今回ツールを適用したのは、移動体端末の1アプリケーションである。適用したアプリケーションは、パターンであるMVCモデルを基本とした機能ブロックで構成されている。(図1参照)このうち、Roseを利用してクラス設計を行った箇所は、Model、View、Controllerの3つとな

1　Microsoft（R）は、米国及びその他の国における米国Microsoft Corporationの登録商標です
2　IBMは、米国及びその他の国における米国International Business Machines Corporationの登録商標です

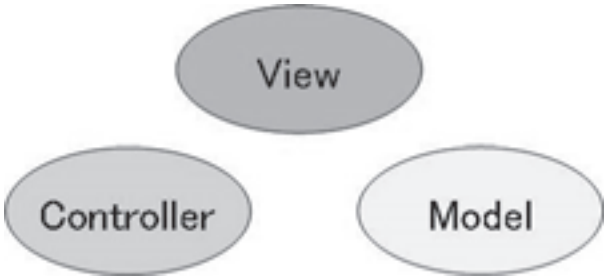



図1　機能ブロック

る。一方、ZIPC++とRoseを連携させた箇所は、左下のControllerブロックのみとなる。

### （1）適用箇所について

Rose-ZIPC++連携を適用したControllerブロックには、大きく分けて状態を管理する部分と関数呼び出し部分の2つから構成される。（図2参照）状態管理部分とは、アプリケーションの様々な状態を管理している。一方、関数呼び出し部分では、ある状態になった時にどういうアクションを実行するかといった状態の振り分けを行う部分である。2つのうち、状態管理部分について、ZIPC++を使用しC++ソースコードの生成まで行った。関数呼び出し部分については、今回はツールを使わずに手動コーディングで実装を行った。


図2　Controllerブロックについて

### （2）実施内容

図3が今回実施したツール適用の関連図である。

Roseでは、クラス図作成とステートチャート図の作成を行った。作成したクラス図のクラス定義については、全てRoseとSoDAを連携させて設計ドキュメントを作成した。さらにクラス図からRoseを使ってヘッダファイルおよび関数のプロトタイプ定義を自動で生成した。続いて、Rose-ZIPC++連携により、クラス図とステートチャート図を元に状態遷移表の自動生成を行った。最後に自動生成された状態遷移表を元にZIPC++でC++ソースコードを自動生成した。

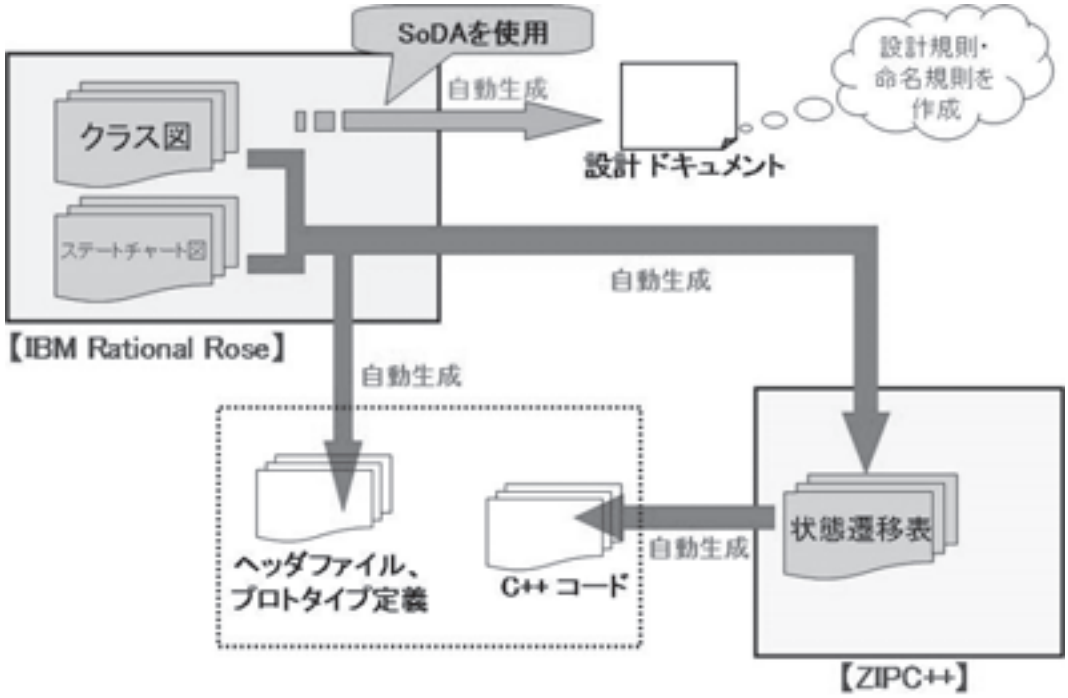



図 3　Rose-ZIPC++連携図

今回、ツール連携を行うにあたり、下記のようなルールを決めて実施した。

①詳細情報まで必要な内容については、全てRoseで作成したUML図に記述し、その後、SoDAを使用して設計ドキュメントを自動で生成するようにした。

②設計規則や命名規則を作成し、設計ドキュメントについても品質を一定にすることを目指した。また、SoDAのテンプレート機能を使用して、作成したクラスが設計規則や命名規則に沿っているかを半自動でチェックするようにした。

③ツールが生成したソースコードには、手動で修正を加えないようにした。ソースコードに修正の必要が生じた場合は、必ずRose側でステートチャート図を修正してからZIPC++と連携させて、ソースコードを自動生成する方法を取った。

④Roseで作成したステートチャート図では、1状態1処理までしか記述しないようにし、生成されるソースコードの複雑度を下げるようにした。また、ソースコードと設計ドキュメントの整合性を取るため、ソースコードに表示させたくない情報は、できるだけステートチャート図に記述しないようにした。

## 4．ツール導入の成果

ツール導入を行った結果として、以下のような成果が挙げられる。

### （1）状態管理について

これまで、ツールを使用しなかった場合では、大量の状態を管理するのが大変だったが、今回ツールを使用したことにより、大量の状態を管理可能なレベルにすることができた。また、1状態1処理の実現が可能となり、ある状態に遷移したら1処理を行うレベルまで状態を分解することで、手動コーディングを行った部分（関数呼び出し部分）の複雑度を下げることができた。この結果、状態遷移表を自動生成した部分のモジュールテスト工程は、簡単なチェックで済ますことができ、短期間で次の結合テストに移ることが可能となった。

### （2）資産構築について

状態管理以外の効果としては、流用開発に向けた資産を構築することができた。これは、差分並行開発を行っている移動体端末の開発にとって大きなメリットとなる。以降のモデルでもRose-ZIPC++連携を実施した際には、より少ない開発メンバで実施することが可能となる。

## 5．今後の課題

### （1）UML図について

開発中には、どうしても開発するアプリケーションの状態が増えてしまうことがあった。ステートチャート図の状態が増えた場合、人間が見ると見づらくなってしまう。また、SoDAを使用して設計ドキュメントを自動生成するために、ソースコードとして出力させたくないものは極力記述しないようにした。結果、人間が参照する設計としては、読みづらいものとなってしまった。

これは、通常Wordなどで設計書を作成した場合では、その設計に至った過程を記述することができるが、今回、ソースコードと設計ドキュメントの整合性を高めるためにステートチャート図に余計な記述を省いたために起きた弊害と言える。今後は、設計ドキュメントとソースコードをいかに同期させるかが課題となる。

### （2）状態の増加に付随する課題

状態が多い場合、イベントが一つ追加された際に各状態時のアクションの決定やアクションが発生した際のUIについて全て検討する必要があった。結果として、イベント追加時の作業工数が増えてしまった。今後は、イベント追加時の効率的な対応を検討していく必要がある。

### （3）リソースマネジメントに関する課題

今回の開発の場合、各メンバは担当した１ブロックに関する専門性はつくが、逆にその専門性しかつかなくなってしまうということがあった。そのため、緊急対応時に別ブロックから応援依頼が入った時に、引継ぎ業務に時間がかかってしまうことが起きてしまった。

## 6．あとがき

今回、ツールサポートとして開発に携わってきた。初回のツール導入だったということもあり、試行錯誤に工数がかかる部分もあった。ツール適用に関しては、ツールに実装されている機能を全て使用するのではなく、そのプロジェクトで適用できると判断された機能を適材適所で使用するのが効果的であると感じた。

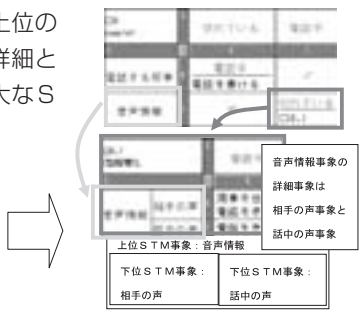
CATS様には、今後も開発がより効率化されるCASEツールを提供していただきたいと思う。

### ご存知ですか？ EHSTM/ZIPC 豆知識

ＳＴＭの事象(E型)階層化とはなんですか。

事象階層化とは、上位のＳＴＭで概要の事象としてとらえ、上位のＳＴＭはその事象を下位のＳＴＭに伝達し、下位の事象では詳細としてとらえる手法です。事象を階層化することで、複雑で巨大なＳＴＭを小さく理解しやすいＳＴＭにすることができます。下位のＳＴＭは上位ＳＴＭのアクションセルから呼び出します。

図A　事象(E型)階層化前




図B　事象(E型)階層化後