海外 W-CDMA 基地局装置への ZIPC ツールの適応

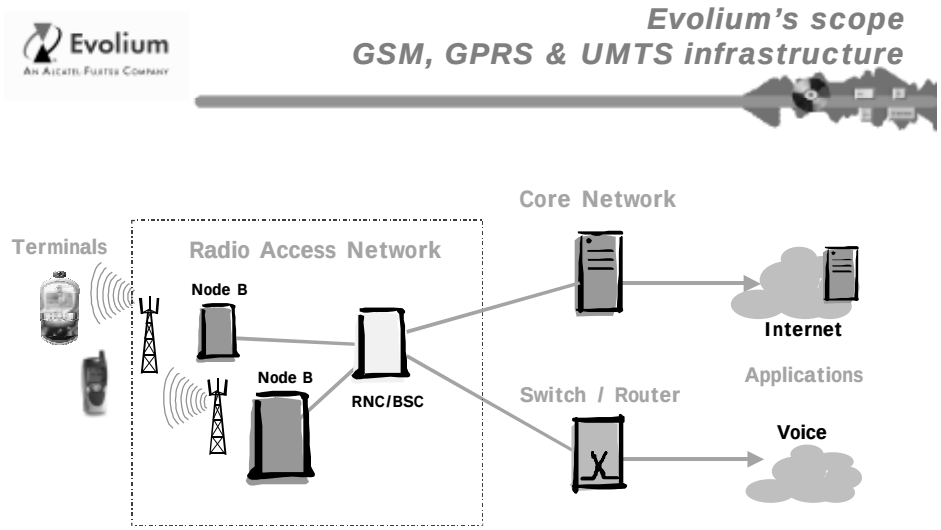
エボリウム・ジャパン株式会社
第一開発部

和久田 達也

1．はじめに

当社 Evolium Japan㈱ は、富士通㈱とフランスの通信機メーカ Alcatel 社との合弁会社で、2000 年 11 月に設立されました。この会社は、富士通が持つ W-CDMA（Wide band CDMA: Code Division Multiple Access）の開発技術をベースに Alcatel 社が持つ欧州の販売チャネルを有効活用し、欧州エリアで展開される次世代携帯電話システムビジネスに参入しようと試みております（システムについては **図 1** を参照）。

既に国内では、(株)ドコモ殿が FOMA システムとして、次世代携帯電話のサービスを既に運用していることは皆様ご承知のことと思いますが、富士通もこの装



置のサプライヤーとしてエントリされており、ここで培ったノウハウを海外にも適応させていこうと手がけております。

基本的には、この国内システムで開発されております RNC と NodeB と言われている両装置のソフトウェア・ハードウェアを極力流用し、海外用として機能のモディファイを加えております。私は、この中で NodeB のソフトウェアの開発を担当し、現在は、一通りの開発を終了し、欧州にて実施されている各種テスト（フィールドテスト）の対応をしております。

2. ZIPC の本製品への適応

上記のような背景の中、ZIPC が我々の製品の中にどのように適応されているかを以下に説明いたします。

NodeB の装置は数種類のパッケージに機能分割され、それぞれのパッケージにソフトウェアを搭載しております。ZIPC はその中で呼処理の機能を担当するアプリケーション部分やファームウェア部分のソフトウェア、また保守監視系のアプリケーション部分に用いられております（全パッケージに搭載されるソフトウェアの約 60% の部分に ZIPC を適応）。

このような ZIPC 適応の背景には、国内のソフトウェア資産の有効利用があります。国内用の別 Project で開発された NodeB 装置にもこれと同じソフトウェアを搭載しており、これらは ZIPC を使用して開発されております。従い、我々の開発するソフトウェアも必然的にこの開発環境一式を踏襲することが、コスト面的には有効な手法となります。

3. 我が Project での ZIPC 活用方法

上記までにご説明した様に、我々はすべてのソフトウェアを新規開発したわけでは無く、ZIPC を採用したソフトウェアをなるべく少ない工数で改造・流用・保守していく為に ZIPC を使用しております。さらに今回の改造では状態遷移までを変更するような改造等は行わなかったので主に保守の立場で ZIPC を使用しました。従って、新規開発・及び改造での ZIPC を適応したときの利点･欠点については詳細に述べることができません（それらについては、既に今まで各方面の方々が語られてきたと思います）。ここからは、ZIPC で開発されたソフトウェアを保守していく中で感じたことを述べていきたいと思います。我々も今まで ZIPC を使ったことは無く初めての経験であり、さらに、開発の初期（初期検討～設計工程）から使ったわけではなく、既に作成されたソフトウェアのデバッグをメインとして、これらのソフトウェアを作成した部隊とは全く異なる部隊として実施していたため、ZIPC の優れた機能を十二分に活用できていない部分もあるかと思いますが、そこはご容赦願います。

3.1 ソフトウェア問題発生時の検証（デバック）での ZIPC

装置単体試験やシステム総合試験で発生した問題について、その現象を解析するときに、ZIPC でかかれた資料を我々は参照します。大前提として、これらの ZIPC ドキュメントは最新のソースファイルとリンクが取れていることが重要です。この場合、複雑なメッセージシーケンスに対し、各タスクの状態を把握することが非常に容易く、そこから問題点を見つけることが可能となります。実際に何点かの問題をプログラム設計者なしで検出し我々で改修することが出来ました。ただし、これは状態遷移に関連する問題の時は効果があるのですが、そうでない場合の問題では我々は以下の点で解析に困難をきたしました。

我々は、多くの解析に実際のソースコード（C 言語プログラム）をトレースしながら検証を実施します。その時に ZIPC で書かれた状態遷移がどのように C ソースに展開されるかを理解していないとその展開系は複雑で、解析者にはそのロジック解析に多くの時間を費やすことになりました。実際、フラグ等で状態遷移を制御していたり、ZIPC のオリジナルな関数が組み込まれていたり、等でそれらの解析に時間をかけたところなどは我々がデバック初期段階で直面した問題でした。このような展開時のソフト構造等を示す資料等（キャッツ殿では用意されていたのかもしれませんが、不運にも我々は手探りの状態でした…）があれば効率的にデバックが出来たものと思われます。

また、ZIPC の特徴の一つに状態遷移の中に別の状態遷移を記述できる。といったネスト構造が可能でありますが、ソフトウェア設計段階でしっかりとソフトウェア構造を考慮していないと、別の解析者が、この ”STM ” の先の ”STM ” のそのまた先の ”STM ” の… となると手に追えなくなり、ソーストレースが不可能な状態になってしまうといったことも今回感じました。さらに ZIPC の性格上難しいことかも知れませんが、個々の状態遷移の中身は非常に分かり易くなるのですが、各状態遷移間のインタフェースのリンクが取り辛く感じました。具体的には、状態遷移のトリガとなるイベントは、別の STM の処理から発生するメッセージ等になってくると思いますが、それら関係が一目して分かるようなインタフェースのリンク（例，インタフェース一覧が自動生成される。とか送信元のメッセージをクリックすると自動的に次の STM に飛んでイベント表示される。等々）が本ツールで可能になると、さらにデバック効率があがるのではと期待します。

3.2 流用元ソフトの改版を取り入れる作業での ZIPC

我々の作業のもう一つの重要な作業として、流用元ソフトの改修部分の盛り込み作業があります。

我々が使用しているソフトウェアは、ある時期に流用元ソフトウェアを切り出して、それをベースに我々独自の機能を盛り込んでいます。しかし、流用元ソフトウェアも開発途中のソフトウェアであるために、それらのソフトウェアもデバッグされて当然、改版作業が発生しています。

我々は、定期的にその改修された流用元ソフトを受け取り、改版部分を我々のベースソフトウェアに盛り込む必要があります（基本的にはソースレベルでの比較差分をとって、その差分をベースソースに盛り込みをしています）。

軽微な修正であれば、我々のソフトウェアと流用元ソフトの差分比較がソースレベルやコメント分等で可能なのですが、STM に関わるような修正が施されていると、展開結果後のソースが元のそれとかなり異なるものとなり場合があり、単純に差分比較とならないケースがあって調査に時間を取られました。

4. まとめ

今回は主にソフトウェアの保守（メンテナンス）工程での ZIPC を使用するということで、本来使われるべき工程でないところ（？）の観点で述べさせていただきました。我々も ZIPC を熟知していない状態で、作業上感じたことを書かせて頂いたかも知れませんが、そこのところはご容赦頂き、まとめとさせていただきます。