4F-7

組込み Java™ 向け開発環境へのアスペクト指向ツールの適用

○井奥 章† 上原 一郎‡ 川崎 進一郎† 森本 義章† 磯部 竜雄‡

(株)日立製作所 中央研究所† 日立ソフトウェアエンジニアリング（株）‡

1. 背景 組込み Java™ 向け開発環境への期待

カーナビやデジタルテレビなどの組込み機器へのソフトウェアにおいては、開発効率の低下、品質の劣化によって、搭載製品の市場価値を損なう事例が目立ってきた。背景には、機種ごとに異なる様々なプラットフォーム(異なる CPU、OS、ミドルウェアの環境)で統一的に使用できる操作性の良い開発ツールが調達しにくく、作業効率が低下している状況が存在する。

Java によるソフトウェア開発においても、組込み機器の実機デバッグやチューニングは必要であり、この実機上での統一したデバッグ環境の調達が容易でない。こうしたニーズに応えるため、組込み Java アプリケーションの開発支援ツールの利便性向上への期待は大きい。

２ 組込み Java™ 向け開発環境の問題点の抽出

Java アプリケーションのデバッグに活用できるツールとして、JDK（JavaDeveloperKit）付属の JDB（Java Debugger）や Eclipse などの統合開発環境に組み込まれたデバッガが存在し、PC や WS の Java アプリケーションの開発で広く使われている。一般にこれらデバッガを活用するには、JVMDI(JavaVM Debug Interface) [4]に準拠した JavaVM の利用が前提となる。

２．１ 問題の提示

組込み機器の Java 開発に JVMDI を適用すると、性能面での問題が顕在化する。

（１）組込み機器側の Java 実行環境（JavaVM）に JVMDI 対応の機能追加による、デバッグ情報の収集のためのオーバーヘッドが発生する。ハードウェア資源に制約のある組込み機器では無視できない程になる場合がある。

（２）組込み JavaVM では、メモリサイズ上の制約などの理由で、高速化機構（JIT コンパイラなど）を優先するため JVMDI 対応を除外せざるをえない場合がある。

以上、（１）（２）の性能問題を抱えているため、運用時の実行性能を優先せざるをえず、組込み Java の各種製品は、JVMDI 非対応であることが多い。そのためデバッグ環境の不便さが問題となっている。

２．２ 現行の対策

組込み機器の実機デバッグでは JVMDI 対応の一般的な Java デバッガを用いるかわりに、次のような対処がなされている。前者は実行環境側、後者はアプリケーション側での工夫であるが、依然として問題を残している。

（１）運用時は性能を優先するためにデバッグ時にのみ JVMDI 対応の Java 実行環境を用いる。→この方法では、運用時と実機デバッグ時を比べると、アプリケーションの実行速度の乖離が拡大する。運用時と異なる挙動がデバッグ時に発生する頻度が多くなるため、デバッグ作業の精度・効率の低下要因となる。

また、組込みの分野ではこの方法を使えない場合が多いのも難点である。OS や CPU が多様で少量多品種である組込み機器の分野では、JavaVM 提供主体の経済性の事情で、デバッグ用の JavaVM が多プラットフォーム展開されているわけでないからである。そのため、組込み機器を対象とする場合に、次に示す旧来からの方法を相変わらず強いられることが少なくない。

（２）デバッグ時にも運用時の Java 実行環境（JVMDI 非対応）をそのまま用いて、アプリケーションに対してデバッグ用の仕掛けを入れる（printf デバッグなど）。
→この方法では、JVMDI 準拠のデバッグ環境の長所である高機能性や、PC/WS ともデバッガを共用できるような汎用性を失う。結果としてデバッグ作業の効率が下がる。

３ アスペクト指向を適用した組込み Java™ 向けデバッグ環境の実現

報告者らは、以上の問題の改善を目的に、組込み Java 向け開発環境の改善を目的に、アスペクト指向の考え方に基づく PC／WS 向けのツールを適用した。

その結果、JavaVM 側に特別な仕掛けを必要とせずに済み、多様な組込み機器の実行環境に即応しやすい汎用性を獲得できた。また、JVMDI 対応のデバッガを用いるよりも、Java アプリケーションの実行速度の低下を抑制しうることがわかった。運用中における不具合発生時の動作タイミングを維持できるので、組込みソフトウェアのテストで問題となる不具合再現の困難さを低減できる。

以下、この新たに実現したデバッグ環境の概要、ならびに性能評価の内容を報告し、その有用性について示す。

３．１ 構成

図１に機器構成とソフトウェア構成を示した。典型的な組込み機器のリモートデバッグの機器構成である。アスペクト指向の考え方に基づく PC／WS 向けのツールである、Java 用ソフトウェア開発環境 Bugdel[1][2]を採用し、さらに、デバッガとしての機能を補完するために機能追加を行った（step 実行や break を始め変数参照/設定等）。追加分を DCT（デバッグ用クラスツール群）として用意した。本システム向けに特別なハードウェア構成（ＪＴＡＧツール等々）の追加は不要である。

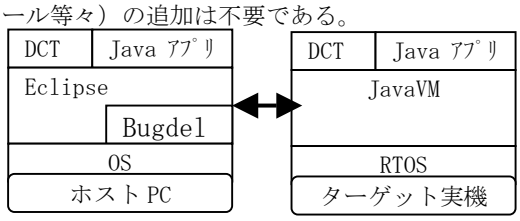

図１ 当該システムの機器構成とソフトウェア構成

Applying an Aspect-Oriented Tool to a Development Environment for Embedded Java™

Akira Ioku†, Ichiro Uehara‡,Shin-ichiro Kawasaki†, Yoshiaki Morimoto†, Tatsuo Isobe‡

†Hitachi,Ltd., Central Reserch Laboratory

‡ Hitachi Software Engineering Co., Ltd., 2nd Software Development Division

Bugdel の処理系では、Javassist[3]が提供する機能を呼び出すことによってアスペクト指向の考え方を実現している。たとえば、一般的なアスペクト指向システムと同様に、デバッグコードとソースコードを別々に記述したり、デバッグコードの挿入位置を一度に複数指定したりという操作が可能になっている。クラスファイルにデバッグコードを挿入する方式なので、ソースコードに手を入れることなく、Java アプリケーションをデバッグできる。デバッグコードはクラスファイルに埋め込まれるため、生成されたクラスファイルを Eclipse 以外の環境で実行することも可能になる。

一般のアスペクト指向システム（AspectJ など）は、汎用的なシステムであるためデバッグには不向きな点がある。例えば、特定の位置にはデバッグコードを挿入できない、advice コードから pointcut 位置にある局所変数へアクセスできないなどの問題がある。これに対して、Bugdel ならではの特徴として、通常のアスペクト指向の要素に加えて、ソースコードの行番号で挿入箇所を指定できる機能等々、デバッガに近い機能が拡張されている。この点に、報告者らは注目した。

３．２　操作方法

Bugdel は Eclipse のプラグインとして稼動し、ユーザは操作の全てをこの GUI を通して行う。

（１）設定　ホスト PC 上で Bugdel を用いて、デバッグ対象（クラスファイル）に対して必要な加工（デバッグ用の追加処理の埋め込み）を行う。GUI ベースのガイドに沿って簡単な操作で一連の該当箇所に行うことができる。埋め込みを行う際に DCT（デバッグ用クラスツール群）からも必要な部品が組み込まれる。

（２）実行　ターゲット実機上の Java 実行環境（組込み JavaVM）が、ホスト PC で加工済みのデバッグ対象（クラスファイル）をリモートロードして、実行を開始する。すると Bugdel のエディタにメソッドのソースが現れ、指定箇所でブレイク状態になる。この状態でステップ実行、変数の参照・設定等が可能になる。

４　性能評価

ベンチマークプログラムに、組込み分野向けの jumcoMark を用いて性能評価を実施した。ターゲット（計測対象）となる組込み機器の CPU は、（株）ルネサステクノロジ製 SuperH™ RISC engine[5]、OS は Linux である。Java™ 実行環境としては J2ME/CDC[6]を用いた。

結果を図２に示す。縦軸がベンチマークのスコアの数値（相対量）である。数値が大きいほど、高速である。図２では、左から順に JVMDI 搭載なしの運用時の Java 実行環境、JVMDI 搭載ありのデバッグ用の Java 実行環境であり、共に Bugdel によるデバッグコードの挿入（埋め込み）がない状態である。右の３つのグラフは Bugdel による埋め込みが存在し、挿入内容の違いで区別した（右ほど挿入量が少ない）。

挿入量にもよるが、挿入対象と内容を選別することで JVMDI 搭載の実行環境よりも速度低下を抑制できることが示唆された結果であるといえる。また、この結果は、アプリケーション開発者自身が性能を制御可能であることを意味しており、運用環境などの条件設定が細やかに効いてくる組込み機器においては、意義深いと考えられる（一方 JVMDI 対応の Java 実行環境では、インタプリタの命令実行ごとにデバッグ情報を取得する挙動が前提であり、カスタマイズ性は低い）。

５　まとめ

本稿で構築した開発環境は、実行環境に左右されない汎用性を実現するとともに、デバッグ時の性能問題の解決にも貢献することを示すに至った。

６　謝辞

本研究を進めるにあたり、東京工業大学情報理工学研究科の千葉滋氏、および、Bugdel の作者でもある東京工業大学情報理工学研究科の薄井義行氏、には、ご討論いただきました。ここに深謝いたします。

７　参考文献

[1]薄井義行、千葉滋,コンピュータソフトウェア、vol.22, no.3, pp.229-234, ソフトウェア科学会、2005
[2] Bugdel Home Page.
http://csg.is.titech.ac.jp/~usui/bugdel/
[3] Chiba, S.: Load-time structural reflection in Java, in *Proc. ECOOP 2000*,2000,pp.313-336.
[4]http://java.sun.com/j2se/1.5.0/ja/docs/ja/guide/jpda/jvmdi-spec.html
[5]http://www.renesas.com/jpn/products/mpumcu/32bit/sh/
[6]http://java.sun.com/products/cdc/

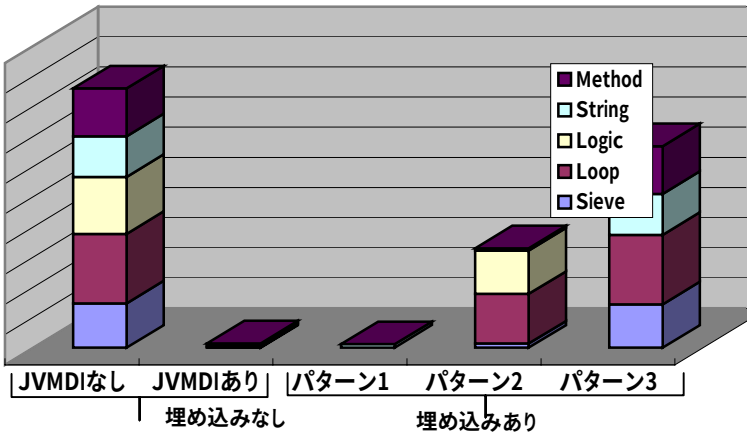

図２　実行環境ごとのベンチマーク結果