RM-003

Javaバイトコード変換による組込み機器連携システムの提案

A Proposal of Cooperation System for Embedded Devices by Java Bytecode Instrumentation

綾木 良太 †　門脇 恒平 ‡　小板 隆浩 †　佐藤 健哉 ‡
Ryota Ayaki　Kohei Kadowaki　Takahiro Koita　Kenya Sato

1 はじめに

近年，情報家電機器や携帯情報端末，PDA など，様々な組込み機器に通信機能が搭載されてきている．また，将来的には通信機能を搭載した組込み機器同士で，自律的に，且つ動的にネットワークを構成すると考えられる．ネットワークを介して，複数の組込み機器の機能を連携させることで，新たな機能を動的に構築することが可能となる．

携帯電話とテレビが連携する場合を例に挙げる．テレビは単体で「動画を出力する」,「テレビ番組一覧を提供する」等の機能を持つが，携帯電話と連携することで，携帯電話に格納されている動画ファイルをテレビ画面から出力したり，テレビ番組表を携帯電話に転送することで，携帯電話からテレビ番組の録画予約を行うことが可能となる．

例えば，固有のハードウェアや OS で構成される組込み機器でも，Jini[1] を利用することで，ネットワークを介した連携が可能となる．しかし，Jini では，ユーザが利用する可能性のある機能のインタフェースを，クライアント機器の実行プログラム (以後，バイトコード) 開発時に，予め全て組み込む必要がある．このため，Jini は将来新たに登場する機能を利用できない．また，ある機器の入力操作と他の機器が提供する機能を動的に対応付けられない．

本研究では，Jini の抱える問題点を解決するために，バイトコードを動的に変換する Jini の拡張システムについて提案する．これにより，将来新たに登場する機能についても利用可能となり，入力操作と機能の動的な対応付けが可能となる．

2 Jini について

2.1 Jini の概要

Jini とは，Sun Microsystems 社が開発した機器間の連携を実現するためのミドルウェアである．Jini は，Java により実装されているため，プラットフォーム依存性が低いことが特徴として挙げられる．Jini では，機器が提供する機能群を “ サービス ” と呼び，サービスを管理することで，機器間の連携を実現する．

Jini は，主にサービスの提供機器である “ サービスプロバイダ ”，サービスの利用機器である “ クライアント ”，サービスの管理機器である “ Lookup サービス ”，サービスを連携させる上で必要なファイルを格納するための HTTP サーバである “ Codebase ” の 4 要素から構成される．

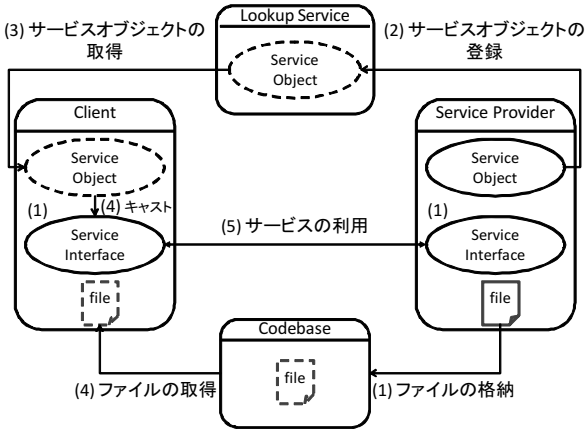

図 1　Jini の構成要素と動作手順

次に，サービスプロバイダが提供するサービスをクライアントが利用するまでの動作手順について，図 1 を用いて説明する．

1. 事前準備
   クライアントは，サービスプロバイダの提供するサービスを利用する際に，サービスプロバイダ固有のインタフェース (以後，サービスインタフェース) を保持する必要がある．また，サービスプロバイダは，クライアントがサービスを利用するために必要なファイルを Codebase に格納する．
2. サービスオブジェクトの登録
   サービスプロバイダは Lookup サービスに対し，自身が提供するサービスをオブジェクト化したサービスオブジェクトを登録する．
3. サービスオブジェクトの取得
   クライアントは Lookup サービスから，利用したいサービスオブジェクトを取得する．
4. サービスオブジェクトの再構築
   サービスオブジェクトは，ネットワークを介して，サービスプロバイダからクライアントに転送される際に直列化される．このため，クライアントがサービスを利用するためには，サービスオブジェクトを再構築する必要がある．クライアントは，Codebase から必要なファイルを取得し，サービスインタフェースを用いてキャストすることで，サービスオブジェクトを再構築する．
5. サービスの利用
   クライアントとサービスプロバイダは RMI(Remote Method Invocation) を利用して通信する．そして，クライアントはサービスインタフェースで定義されている機能を用いて，サービスプロバイダの提供するサービスを利用する．

† 同志社大学 理工学部 情報システムデザイン学科
‡ 同志社大学大学院 工学研究科 情報工学専攻

2.2 Jini の問題点

Jini を用いることで，ネットワーク上のサービスを管理し，機器間の連携が可能となる．しかし，現状の Jini には大別して二つの問題点がある．

一つ目の問題点として，クライアントが，利用する機器のサービスインターフェースを全て事前に保持しなければならない点が挙げられる．これは，Jini では，直列化されたサービスオブジェクトを再構築する必要があり，前節の動作 1 で述べたように，サービスプロバイダ毎にサービスインターフェースが固有であることに起因する．また，クライアントが開発された時点よりも，後に開発されるサービスは，サービスインタフェースを保持しないため利用できない．本研究では，この問題を“ 未定義サービスへの対応問題 ”と呼ぶ．

二つ目の問題点として，クライアント機器の入力操作とサービスプロバイダ機器が提供する機能を対応付ける方法がないことが挙げられる．同一ネットワーク内にクライアントとして携帯電話，サービスプロバイダとしてテレビが存在する場合を例として挙げる．ここで，携帯電話には，入力装置としてボタン A，B があり，テレビには「動画を出力する」と「テレビ番組一覧を提供する」機能があると仮定する．Jini を利用することで，機器間の連携は実現可能であるが，ボタン A に対して，「動画を出力する」と「テレビ番組一覧を提供する」のどちらの機能が対応付けられたかをユーザは事前に知ることはできない．このため，ユーザが直感的に行った入力操作と実際に対応付けられている操作に差が生じる可能性がある．本研究では，この問題を“ 入力操作と機能の対応付け問題 ”と呼ぶ．

3 提案システム

3.1 システムの概要

本研究では，バイトコード変換を用いて，動的にサービスを利用するために必要な実行可能プログラムとサービスインタフェースを生成することで，Jini が抱える二つの問題点の解決を図る．

提案システムは，最小で三つの機器から構成される．例として, 図 2 に示すようにクライアント機能を持つ“ 機器 C ”, サービスプロバイダ機能を持つ“ 機器 S ”, Lookup サービス機能を持つ “ 機器 L ” を用いて説明する．

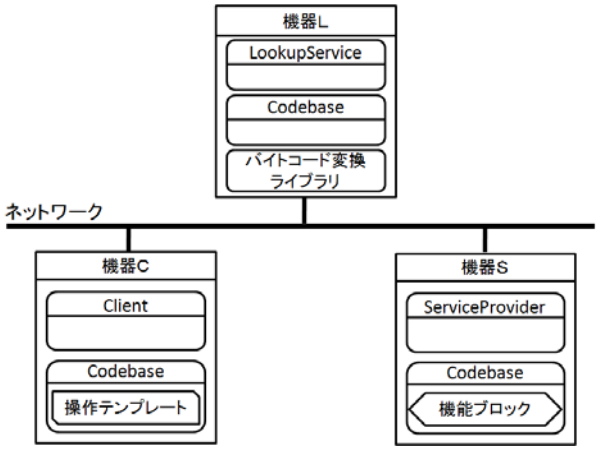

図 2　提案システムの構成

提案システムでは，全ての機器がファイルを受け渡す必要がある．そのため，全ての機器に対して，Jini における Codebase 機能を持たせる．

また，機器連携の準備として，機器 C は入力装置に対応するプログラム (以後，操作テンプレート) を，機器 S は，提供機能を定義したプログラム (以後，機能ブロック) をそれぞれの Codebase 内に用意する．

操作テンプレートには，機器 C の入力装置を操作した時に，呼び出されるメソッド群を定義する．例えば，機器 C の入力装置として，複数のボタンが存在する場合，操作テンプレートにはボタンの数だけ，メソッドを定義する必要がある．そして，ボタンを操作された際のイベントハンドラとして，各メソッドとボタンを一対一に対応させる．操作テンプレートを用いることで，機器 C の入力装置に関する情報を他の機器，またはユーザに提示可能となる．

機能ブロックには，機器 S が提供する機能を呼び出すメソッド群を定義する．例えば，サービスプロバイダとして，スピーカーサービスが存在する場合について説明する．ここで，スピーカーサービスは機能として“ 音を鳴らす ”, “ 音を止める ” という機能を提供する．サービスプロバイダは，クライアントから RMI 通信により，メソッドを呼び出されることで，機能を提供する．ここで, “ 音を鳴らす ” 機能が play メソッド, “ 音を止める ” 機能が stop メソッドにより，実行されるとすると，機器 S の機能ブロックには，play メソッド，及び stop メソッドを定義する必要がある．機能ブロックを用いることで，機器 S の提供する機能についての情報を，他の機器，またはユーザに提示可能となる．

この二つのプログラムは，機器 C の入力操作，及び機器 S の提供する機能を定義するもので，単体では実行できない．内部にバイトコード変換ライブラリを持つ機器 L が，ユーザが決定した対応付けに従い，動的に操作テンプレートのメソッド定義に対して，機能を組込むことで，実行可能プログラムを生成する．この実行可能プログラムを用いることにより，ユーザは，機器 C の入力装置を操作した際に，機器 S に対応付けた機能を利用可能となる．なお，機器 L はユーザに対し，入力操作と機能一覧を提示し，ユーザが決定した対応付けを取得するため，ディスプレイ等の出力装置と，マウスやキーボード等の入力装置を備える必要がある．

3.2 システムの動作

Jini では，Lookup サービスを介して，サービスプロバイダが提供するサービスオブジェクトをクライアントに渡すことで連携を実現する．これに対し，提案システムでは，サービスオブジェクトの受け渡しを行う必要はない．代わりに，クライアントとサービスプロバイダは，Lookup サービスに対し，自身の Codebase に格納された操作テンプレート，または機能ブロックを指す URL を登録する．なお，URL は，Java 標準 API の URL クラスを用いて実装する．そのため，ネットワークを介す際に直列化された URL オブジェクトは，常に同一の URL クラスを用いてキャストし，再構築できる．

提案システムによる機器連携は次のような動作手順により実現する．

1. URL の登録
   クライアントは，機器 C の Codebase 内にある操作テンプレートを指す URL を Lookup サービスに登録する．同様に，サービスプロバイダは機器 S の Codebase 内にある機能ブロックを指す URL を登録する．
2. ファイルのロード
   Lookup サービスは, 登録された URL を基に, 操作テンプレートと機能ブロックを各 Codebase からロードする．ロードしたファイルは機器 L の Codebase 内に格納する．
3. 入力操作と機能の対応付け
   操作テンプレートと機能ブロックをロードした機器 L は，ディスプレイを通して，ユーザに GUI 画面を表示する．GUI 画面には，操作テンプレートから取得した機器 C の操作一覧と，機能ブロックから取得した機器 S の機能ブロック一覧を表示する．ユーザは画面に表示された情報を基に，機器 C の入力操作と機器 S の機能を対応付ける．
4. プログラムの動的生成
   ユーザが設定した機器 C の入力操作と機器 S の機能ブロックの対応を基に，機器 L は操作テンプレートの適切な箇所に機能を組込み，実行可能プログラムを生成する．また，機能ブロックを基に，クライアントがサービスプロバイダと RMI 通信する際に必要なサービスインタフェースを生成する．そして生成した実行可能プログラムとサービスインタフェースを機器 C の Codebase 内に格納する．
5. サービスの利用
   機器 C は，実行可能プログラムを起動し，機器 S と RMI 通信を行う．この時，ユーザが機器 C の入力装置を操作すると，サービスインタフェースで定義されている機器 S の機能を呼び出すことができる．

4 性能評価

4.1 システムの実装

提案システムに基づいた実装を行う．システムの構成として，3 章で説明したように機器 C，機器 S，機器 L の機能を持つ三台のコンピュータを同一ネットワーク上に接続する．

実装では，機器 C は入力装置としてボタン A，B を持ち，「ボタン A を押す」,「ボタン B を押す」という入力操作を行えるように設計した．また，機器 S には，『“ Hello ”を出力する」,『“ World ”を出力する」という文字出力サービスを提供するように設計した．機器 L は，ユーザが決定した入力操作と機能ブロックの対応付けを基に，バイトコード変換を行い，実行可能プログラムを生成する．機器 C の操作テンプレート，機器 S の機能ブロックと機器 L により動的に生成される実行可能プログラムの関係を図 3 に示す．

ここでは，機器 C のボタン A を押すと，機器 S で“ Hello ”が表示され，ボタン B を押すと“ World ”を表示されるシステムを実装した．実装システムの評価環境を表 1 に示す．バイトコード変換ライブラリには Javassist[2] を使用した．

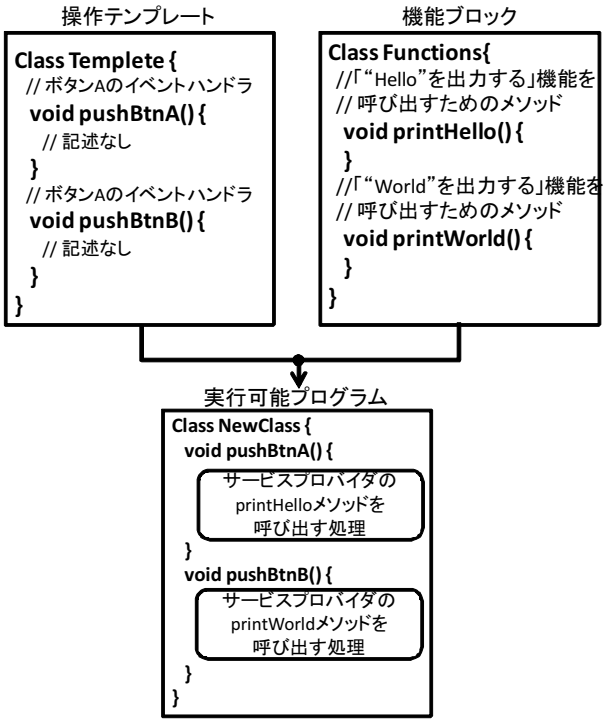

図 3　実行可能プログラムの生成

表 1　評価環境

OS	Windows XP Professional SP2
CPU	Pentium 4，3.00GHz
メモリ	1 GByte
Java	JDK 1.6.0_05
Jini	Jini Starter Kit 2.1

4.2 処理性能

提案システムでは，Jini のみを用いて構成される機器連携システムと比較して，Jini の抱える問題を解決するために，バイトコード変換により，実行可能プログラムとサービスインタフェースを生成する処理が加わっている．そのため，提案システムの実用性を考察するために，Lookup サービス機器で行う実行可能プログラムとサービスインタフェース生成処理について評価する．

操作テンプレートに対して，組込む機能数と処理時間の関係を図 4 に示す．機能数が 1 の場合，処理時間は 238 msec となり，1000 の場合は 656 msec の処理時間を要した．

次に，操作テンプレートに対して，組込ませる機能数とメモリ使用量の関係を図 5 に示す．メモリ使用量を計測するにあたり，JavaVM のメモリ（ヒープ）使用状況を解析するためのツールである jstat[3] を用いた．機能数が 1 の場合，メモリ使用量は 2242 KByte となり，1000 の場合は 4156 KByte となった．main メソッド内に記述がされていない空のプログラムを実行する際のメモリ使用量は 511 KByte であった．

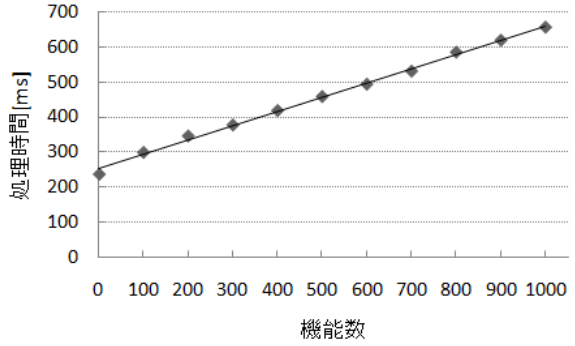

図 4　機能数と処理時間の関係

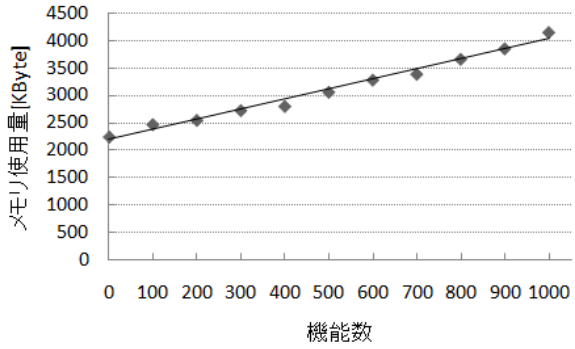

図 5　機能数とメモリ使用量の関係

4.3　考察

提案システムの設計に基づいて実装を行い，システムが提案通り動作することを確認した．

機能数と処理時間の関係を示した図 4 より，機能数を増やした場合，処理時間が長くなっていることがわかる．提案システムでは，機器 C の入力装置に対して，機器 S が提供する機能を割り当てている．一般に，組込み機器は専用目的で開発されているため，入力装置を必要以上に備えていない．提案システムは，組込む機能数が 1000 の場合にも，656 msec と短い時間で処理できるため，処理時間の観点から見ると，実用的であると言える．

次に，機能数とメモリ使用量の関係について考察する．実装結果より，機能数の増加に伴い，メモリ使用量が増大することが判明したが，図 5 から，機能数を増やした場合でも，メモリ使用量が急激に増加することはないとわかる．しかし，組込み機器の多くは，必要最低限の資源で構成されているため，より負荷の少ない実装方法を思案する必要がある．

5　関連研究

組込み機器間の連携を実現するための技術として, Jini 以外に複数の技術や研究が存在する．

UPnP(Universal Plug and Play)[4] は，ネットワークに接続された機器の発見，及び利用するための技術で，プログラムモジュールにより機能を提供する．しかし UPnP では，Jini のようにプログラムモジュールの動的なロードや実行を行うことができない．

HAVi(Home Audio / Video interoperability)[5] は情報家電機器の発見と利用を実現するための技術である．HAVi ではソフトウェアモジュールにより，サービスを実現する．しかし，ソフトウェアモジュールは統一規格に従って定義する必要があるため，新しい機能を追加する時は，ソフトウェアモジュールを更新する必要がある．

Jini を機能拡張した技術として，アダプティブ Jini[6] が提案されている．アダプティブ Jini では，Jini の抱える “ 未定義サービスへの対応問題 ” に対して，サービスを利用する上で必要なプログラムを，サービスプロバイダ機器の Codebase に格納し，必要に応じてクライアントにロードさせることで，問題を解決している．また，“ 入力操作と機能の対応付け問題 ” に対しては，クライアントに GUI を提供し操作させることで，解決を図っている．アダプティブ Jini を利用したシステムでは，クライアントが GUI を用いて入力操作を行うため，クライアント機器がディスプレイ装置を備えていることが前提である．しかし，組込み機器は，外部出力装置としてディスプレイを備えていないこともあり，“ 入力操作と機能の対応付け問題 ” に対しては，Lookup サービス機能を持つ機器にのみディスプレイが必要な提案システムの方が汎用性が高いと言える．

6　まとめと今後の課題

本研究では，ネットワークを介して，組込み機器が提供するサービスを連携させることで，新たなサービスを構築できる組込み機器連携システムを提案した．組込み機器間の連携を実現するためのミドルウェアとして Jini を採用し，Java バイトコード変換による動的プログラム生成機能を追加することで，Jini の問題点である “ 未定義サービスへの対応問題 ” と “ 入力操作と機能の対応付け問題 ” を解決した．

今後の課題として，実際の組込み機器に提案システムを実装し，評価することが挙げられる．それに伴い，必要最低限の資源で構成される組込み機器上でもシステム運用を実現するために，より負荷の少ない実装方法を思案する必要がある．また，入力操作と機能の対応付けを行う際，遠隔地から操作可能にすることや，対応付けを行っていないユーザに対し，対応付け情報を提示するための機能を実装する必要がある．

参考文献

[1] Arnold, K., Wollrath, A., O’Sullivan, B., Scheifler, R., and Waldo, J.: The Jini Specification - Second Edition, Addison-Wesley(2001)

[2] 千葉　滋，立堀　道昭：Java バイトコード変換による構造リフレクションの実現，情報処理学会論文誌，Vol.42，No.11，pp.2752-2760(2001).

[3] jstat: Java Virtual Machine Statistics Monitoring Tool, (2004).

[4] UPnP: UPnP Device Architecture, (2003). http://www.upnp.org/resources/documents/

[5] HAVi, Inc.: Specification of the Home Audio / Video Interoperability (HAVi) Architecture, Version1.1(2001).

[6] Kadowaki, K., Hayakawa, H., Koita, T., and Sato K.: Design and Implementation of Adaptive Jini System to Support Undefined Services, Proceedings of the 6th Annual Conference on Communication Networks and Services Research，pp.577-583(2008).