

Sun SPOT における SSRMI の実装

大 宮 佑 介^{†1} 杉 山 安 洋^{†1}

無線センサネットワークを容易に構築する事の出来るデバイスとして Sun SPOT がある。Sun SPOT を使用すれば Java 言語を用いて容易にプログラムを作成し動作させる事が可能である。しかし、Sun SPOT ではネットワーク上に存在するリモートオブジェクトのメソッドを呼び出す技術が実装されていない。そこで、本研究では Sun SPOT 向けのリモートメソッド呼び出しフレームワークとして SSRMI を提案し、そのアーキテクチャと実装方法について述べる。

Implementation of SSRMI on Sun SPOT

YUSUKE OHMIYA^{†1} and YASUHIRO SUGIYAMA^{†1}

Sun SPOT is a device designed to build wireless sensor networks. Sun SPOT includes a Java virtual machine to allow programs written in Java to run on it. However, the current Sun SPOT technology does not allow remote method calls over networks. We propose a remote method invocation framework SSRMI for Sun SPOT. This paper will show its architecture and implementation.

1. はじめに

近年、無線技術の発展に伴い無線技術を使ったネットワークの構築が盛んに行われている。無線を使用したネットワークを構築することのできるデバイスの一つに Sun SPOT(Sun Small Programmable Object Technology)¹⁾がある。Sun SPOT とはサン・マイクロシステムズ(Sun Microsystems)の開発した、図1に示すような、小型の無線センサーネットワークデバイスである。これは、Java 言語により書かれたプログラムを実行させることが

可能で、各デバイスを使用して無線によるネットワークを構築することができる。

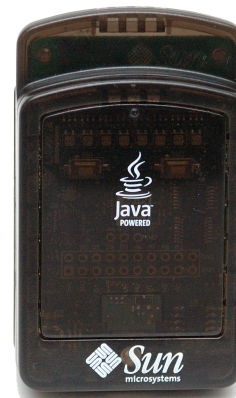


図1 SunSPOT
Fig.1 SunSPOT

Sun SPOT には標準で温度、照度、加速といった種類のセンサが用意されている。他にも汎用の IO ピンやアナログの入力ピンなどが用意されているため、ユーザによるセンサの追加などにも対応し、センサ類だけでなくモーターの制御やその他、高度な制御に用いることも可能である。これらを無線ネットワークを利用して活用することが可能で、様々な分野に応用が期待されている。

一般的にネットワークを使用したプログラムは複雑になることが多く、開発者にとって大きな負担となる場合が多い。Java SE(Standard Edition)ではネットワークを通してのメソッド呼び出しを容易に実現することができる技術の一つである RMI(Remote Method Invocation)²⁾と呼ばれる技術が用意されている。

しかし、Sun SPOT ではモバイル機器向けの Java MIDP2.0³⁾(Mobile Information Device Profile 2.0)を採用しておりネットワークを通してのメソッド呼び出しが実装されていない。本稿では、Sun SPOT 向けの RMI として SSRMI(Sun SPOT RMI)を提案し、そのアーキテクチャと実装について述べる。

本稿の構成は以下の通りである。まず、2節で RMI の概要と、RMI を Sun SPOT へ実装する場合の問題点について述べる。3節で SSRMI の概要、4節で実装、5節でその使用方法を述べる。6節で SSRMI の評価と7節で関連技術について述べる。

2. RMI の概要

RMI とは、リモートホスト上にあるオブジェクトのメソッドを実行する処理を、プログラマが容易に記述することのできるシステムである。つまり、一つの Java 仮想マシン (VM) 上のオブジェクトが、別の VM 上に配置されたオブジェクトのメソッドを呼び出し、実行させる事が可能である。また、プログラマは特にネットワークの知識がなくともネットワーク上にある別の VM 上に設置されたオブジェクトのメソッドを呼び出す処理を構築することができる。

^{†1} 日本大学工学部 情報工学科
Department of Computer Science, College of Engineering, Nihon University

2.1 RMI のアーキテクチャ

図2を用いてRMIの基本的なアーキテクチャを説明する。RMIではNamingクラスが用意されており、サーバオブジェクト上に置かれたオブジェクトを使用するための登録(bind)や検索(lookup)を行う。

RMIを使用するにはあらかじめいくつかの準備をする必要がある。その一つにrmicと呼ばれるツールを使用してStubとSkeletonと呼ばれるネットワークを使用するための部品を生成する必要がある。現在のJavaのバージョン1.6ではStubとSkeletonは廃止されているが、ここでは後述するSSRMIの説明の関係上Stub・Skeletonについても説明する。

Stubは呼び出す側であるクライアントホスト上に設置される。クライアントオブジェクトがサーバホスト上にあるサーバオブジェクトを間接的に呼び出すためのオブジェクトである。クライアントオブジェクトは、サーバオブジェクトを呼び出すためにStubを呼び出す事でネットワーク越しでのメソッド呼び出しを行っている。Skeletonはサーバホスト上に設置され、サーバオブジェクトの呼び出しを行う。前述で説明したStubからネットワークを通してメソッド呼び出し情報を受け取り、サーバオブジェクトの呼び出しを行っている。

また、サーバオブジェクトの情報を登録・検索するレジストリサーバとしてrmiregistryがある。このrmiregistryはサーバホスト上で起動し、サーバオブジェクトの登録を行う。クライアントオブジェクトは、このサーバオブジェクトが登録されたrmiregistryに検索をかけ、サーバオブジェクトの情報を取得し、その情報を元にサーバオブジェクトのメソッド呼び出しを行う。

2.2 Sun SPOT と MIDP

Javaは、使用される環境に合わせて複数の仕様を用意している。一般的なPC向けのJ2SE(Java 2 Standard Edition)、サーバ向けのJ2EE(Java 2 Enterprise Edition)、そして携帯電話などの携帯端末向けのJ2ME(Java 2 Micro Edition)が存在する。

J2MEは、家電製品や携帯電話など小型の機器向けに用意された仕様である。様々な機器に対応させるため、Javaの実行環境仕様であるコンフィギュレーションとクラスライブラリのセットであるプロファイルが用意されている。J2MEではコンフィギュレーションとしてCLDC(Connected Limited Device Configuration)とCDC(Connected Device Configuration)がある。CLDCは携帯電話のような処理能力の低い機器を対象とし、Java VMの代わりにKVM(The K Virtual Machine)と呼ばれる小型の機器専用のVMが用意されている。J2SEとは互換性が無く、最小限の機能で動作する。CDCはカーナビゲーションシステムなど、ある程度処理能力を持つ機器を対象としておりJ2SEと比較してGUI関

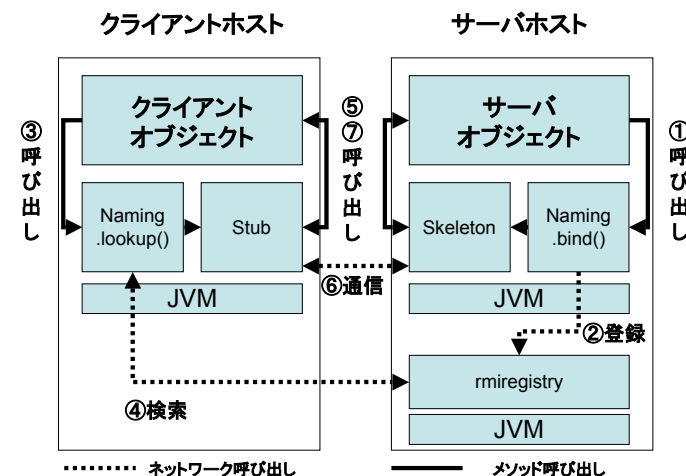


図2 RMIの概要
Fig.2 An Overview of RMI

係を除くほぼ全てのライブラリが含まれてる。

CLDCと組み合わせるプロファイルにはMIDP、IMP(Information Module Profile)、Dojaなどがある。MIDPは携帯電話やPDAなどのような組み込み機器で使用するプロファイルである。IMPは自動販売機や組み込み向け産業機器などディスプレイを持たないデバイスなどに用意されているプロファイルである。また、DojaはNTTドコモ社の携帯電話上で動作するJavaアプリケーション向けで、同社の提供するiアプリなどで使用されているプロファイルである。

Sun SPOTではCLDCと組み合わせて使用するMIDPが採用されている。そのため、Java SEで使用されていたRMIなどのライブラリが用意されておらず、最小限のライブラリのみ用意されている。また、VMにはKVMの代わりにCLDCに準拠したSquawk VMが実装されている。Squawk VMは小型の組み込みデバイス向けに開発されたオープンソースのVMで非常にコンパクトなVMである。

2.3 Sun SPOT への実装時の問題点

Sun SPOTではMIDPが採用されており、RMIをそのまま移植することはできない。RMIで使用されている技術であるTCP・UDPを使用した通信やリフレクションなどがサポートされていないのが大きな原因である。これは、プロファイルの違いによる機能の制限

によるものが大きい。図3はプロファイルと機能の比較を行ったものである。なお、Dojaについては一部の項目が未公開のため不明と記載している。

Sun SPOT では、TCP や UDP を使用する Socket に代わり、Radiogram, Radio Stream と言った無線ネットワーク用の通信方式が用意されている。これらの通信方式は、TCP や UDP との互換性がないため、PC ネットワークと直接通信することは不可能である。

また、Sun SPOT は SPOT 自身のアドレスを指定して通信する事ができない。これは、RMI ではローカルホストに対しての通信が可能であったため、サーバホスト上に置かれたサーバオブジェクトを同一ホスト上で起動している rmiregistry へ bind することが可能であったが、Sun SPOT ではローカルホスト間での通信が出来ないため、同一ホスト上のサーバオブジェクトからレジストリへの登録が困難である。

リフレクションとはプログラム自身が実行過程でその構造を解析、書き換えする技術で、この技術を使用することにより動的な処理の実行が可能となる。RMI では実行中に、リモートで呼び出されるオブジェクトの解析などに使用している。

また、Sun SPOT 上では複数のプロセスの起動が制限されているため、サーバホスト上でレジストリを立ち上げる事が困難になっている。これは、Sun SPOT 上では Squawk VM が用意されているが、Java SE などと違い、VM を複数動作させることが出来なくなっている。それにより複数のプロセスを別々に起動・実行することが出来ない。そのため、Java SE の RMI で可能であった rmiregistry とサーバオブジェクトを別の VM 上で起動させることが不可能である。

	Java SE	MIDP	Doja	Sun SPOT
VM(Virtual Machine)	JVM	KVM	KVM	Squawk VM
Network	TCP/IP	HTTP	HTTP	Radiogram/RadioStream
リフレクション	○	×	×	×
複数のVM起動	○	×	×	×
同一ホストとの通信	○	×	不明	×

図3 Java プロファイルの比較
Fig.3 Comparing Java Profiles

3. SSRMI のアーキテクチャ

SSRMI とは、Sun SPOT 向けのリモートメソッド呼び出しフレームワークである。基本的な使用方法とプログラムの配置を RMI とほぼ同じような構造としユーザが特に違いを意識せず使用できる。

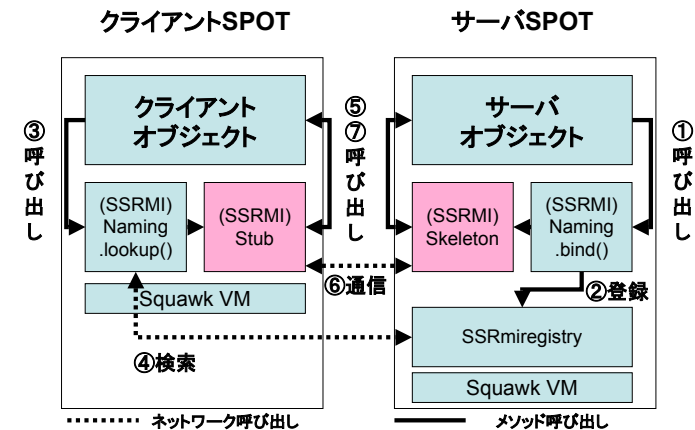


図4 SSRMI の概要
Fig.4 An Overview of SSRMI

SSRMI は RMI と比較して、基本的な使用方法やクラス、プログラムの配置は RMI とほぼ同じような構造になっているが、いくつかの相違点がある。図4は SSRMI のアーキテクチャを図にしたものである。

3.1 SSRmiregistry

J2SE では、rmiregistry と呼ばれるリモートオブジェクト検索用のディレクトリサーバが用意されていた。しかし、Sun SPOT では、通信方式やプロファイルの違いからそのまま rmiregistry を起動させることは出来ない。そこで、SSRMI 専用の SSRmiregistry を用意している。SSRmiregistry には次に示す特徴がある。

まず、SSRmiregistry は、従来の rmiregistry のようなツールではなく、Naming クラスと同じように API として提供される。これは、Sun SPOT では複数の VM を起動することが出来ないため、1 つの VM 上でサーバオブジェクトと SSRmiregistry を起動させる方式

を SSRMI が採用しているためである。Sun SPOT 上で bind メソッドを使用してサーバオブジェクトの登録を行う前に、ユーザがプログラム中に SSRmiregistry の起動を記述する。

また、RMI では同じホスト上に置かれた rmiregistry に対して自身の IP アドレスを指定してネットワークを通して登録を行っていた。しかし、Sun SPOT では自身のアドレスを指定しての通信をサポートしていない。SSRMI でも、SSRmiregistry へサーバオブジェクトを登録する際に SSRmiregistry のある SPOT のアドレスを指定する。そのアドレスを見て、同じ SPOT 上にある SSRmiregistry に対して登録が行われた場合には、ネットワークを経由せず、SSRmiregistry オブジェクトのメソッドを直接呼び出すことにより登録を行っている。

また、RMI ではセキュリティの面から、リモートホスト上にあるレジストリへの bind、rebind などの処理は禁止されている。しかし、Sun SPOT では一般的な PC ネットワークとの互換性がないことや SPOT 自身の処理能力の面から、柔軟な使用を考え、図 5 のように別の SPOT 上にあるレジストリへの登録を可能としている。この仕様により、SSRmiregistry だけを設置した SPOT を起動することも可能になっている。SPOT 自身の処理能力はそれほど高くないため、負荷のかかる処理がある場合はサーバオブジェクトと SSRmiregistry を別々の SPOT 上で起動させるといったことも可能となっている。また、セキュリティの面から他の SPOT からの SSRmiregistry への登録を禁止することも可能である。

3.2 スタブ・スケルトンコンパイラ

また、Stub と Skeleton の生成についても RMI で使用されている rmic に代わって ssrmic を用意して対応している。これにより、Sun SPOT の通信方式である Radiogram、Radio Stream に対応し、SSRMI 向けの Stub と Skeleton を生成させている。また、RMI で使用されていた Naming クラスについては、基本的な lookup メソッドや bind メソッドといったメソッド名は従来のものとほぼ共通で、プログラムを書くにあたって、あまり違いを意識せずにプログラムを書くことができるようになっている。

ただし、SSRMI は RMI とは通信での互換性がない。これは、RMI で使用されている通信方式である TCP・UDP に Sun SPOT が対応していないためである。そのため、SSRMI からの RMI の呼び出し、RMI からの SSRMI の呼び出しなどには対応していない。

4. SSRMI の実装

SSRMI を実装するにあたって、通信方式の違いやリフレクションの使用ができないため RMI とは一部異なる方式で実装されている。基本的な使用方法に関しての互換性は保たれ

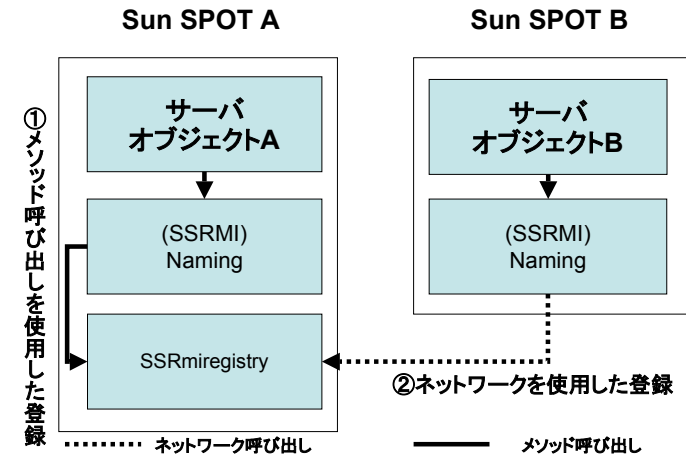


図 5 SSRmiregistry への登録
Fig.5 Registration to SSRmiregistry

ているが、実装面では全く別ものである。

4.1 メソッド呼び出しの実現

SSRMI でのリモートメソッド呼び出しを図 6 を使用して説明する。

- (1) クライアントはレジストリの指定されたポート番号に対してリモートオブジェクトの検索をかける。
- (2) 検索をかけられたレジストリは検索結果を元に Stub の生成に必要なサーバのアドレスやポート番号と言った情報をクライアントに送信し、その情報を元にクライアントでは Stub が生成される。Stub はリモートオブジェクトと同じ引数・返り値のメソッドを保有している。
- (3) クライアントは、リモートオブジェクトの代わりに Stub のメソッドを呼び出し、Stub は Skeleton へメソッド呼び出しを行うためのポート番号を要求する。
- (4) Stub からポート番号を要求された Skeleton は使用していないポート番号を検索し、Stub へ接続先のポート番号を送信する。
- (5) Stub は受け取ったメソッド呼び出し用ポート番号へメソッド名・引数を送信する。
- (6) Skeleton は受け取ったメソッド名・引数を元にリモートオブジェクトのメソッドを呼び出す。

- (7) リモートオブジェクトの処理が終了すると処理結果が返る。
- (8) Skeleton は処理の結果を Stub へ送信する。

以上が SSRMI でのメソッド呼び出しの実装である。また、それぞれの通信には SSRMI の独自プロトコルを使用して通信エラーや整合性・同期を取っている。プロトコルのヘッダなどが追加されているために、通常の通信と比べてオーバーヘッドが増えている。

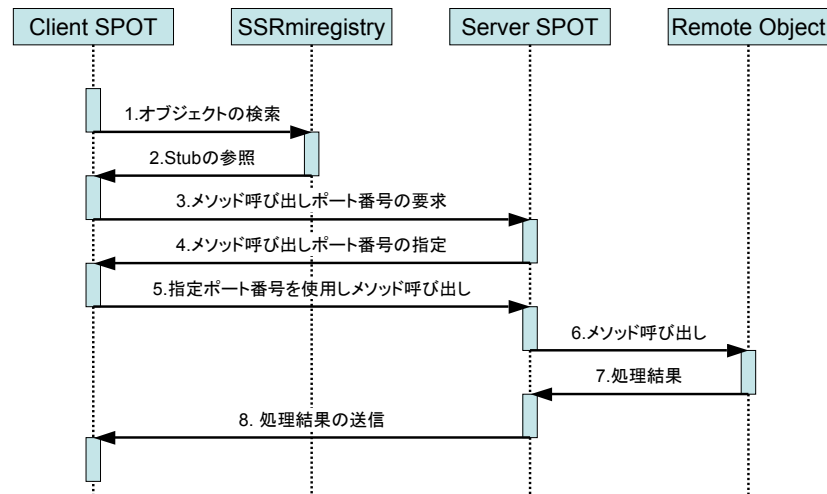


図 6 SSRMI のシーケンス図
Fig. 6 SSRMI sequence diagram

4.2 SSRMI での通信

Sun SPOT では TCP・UDP をサポートしておらず、無線通信方式である IEEE 802.15.4 を使用した通信を行うため Java の API である Socket クラスの代わりに Radiogram・Radio Stream と言ったネットワークを使用するクラスが用意されている。これは、Sun SPOT で使用する無線通信方式である IEEE 802.15.4 に準拠したネットワークを使用するための通信方式で、TCP・UDP などとの通信の互換性はない。基本的に IP アドレスではなく MAC アドレスを指定して通信を行うため、SSRMI 上でのアドレスの指定も Sun SPOT 自身の MAC アドレスを指定して行うようになっている。SSRMI では任意のポート番号を使用して通信を行っている。使用できるポート番号は 1~250 番までとなっており、これはハード

ウェアの制限によるものである。

4.3 リフレクション

リフレクションとは、プログラム自身が実行中にその構造を解析する技術である。この技術を使用する事で実行中に動的なコードを生成する事が可能である。

Sun SPOT では MIDP を採用している事からリフレクションに対応していないため、リフレクションを使用する事が不可能である。現在の RMI では、Stub・Skeleton を生成する代わりに、リフレクションを使用して動的にリモートオブジェクトのメソッド呼び出しを行っているが、Sun SPOT ではリフレクションが使用できない。そこで、SSRMI では `ssrmic` を実行した際に Stub・Skeleton として静的なコードを実行させることでリモートメソッドの呼び出しを行っている。

4.4 `trmic` を使用した `ssrmic` の実装

SSRMI を使用するにあたって、Stub と Skeleton というネットワーク通信をする為のクラスが必要になる。RMI には `rmic` と呼ばれる Stub と Skeleton を生成するツールが存在する。ユーザが用意したリモートオブジェクトを解析し、解析した結果を元に Stub と Skeleton を生成するツールが `rmic` である。SSRMI では、Sun SPOT 用に Stub と Skeleton を生成するツールとして `ssrmic` を用意している。これは Sun SPOT 用に書かれたリモートオブジェクトのクラスファイルを解析し、解析結果に合わせて Stub と Skeleton を生成するツールである。

TRMI⁴⁾ と呼ばれるスタンドアローンソフトウェアを自動的に分散処理に対応させる技術がある。この技術は Java 言語で書かれたスタンドアローンで動作するソフトウェアを自動的に解析し解析結果を元にネットワークを使用して分散処理を行うことのできるソフトウェアに変換する技術である。この TRMI で使用されているツールに `trmic` がある。これは、Java で書かれたクラスファイルを解析し、ネットワークを使用する部品や、分散処理を実行する部品などを自動で生成するツールである。`trmic` には java のクラスファイルを解析し、クラスファイルを生成する技術が使用されているため、今回 `ssrmic` を実装するにあたって `trmic` の技術をそのまま利用して `ssrmic` の実装を行った。

4.5 メソッド呼び出しにおける制限

メソッド呼び出しの際に、必須となってくるもののひとつに引数や戻り値などが挙げられる。SSRMI では、メソッド呼び出し時にいくつかの制限が設けられている。メソッドを呼び出す際にメソッド呼び出しに使用する型の宣言と戻り値に使用する型の宣言に制限がある。基本的に対応している引数・戻り値の型はプリミティブ型(int, long, float, double,

boolean) と String 型の 2 つである。この型以外の変数は現時点では指定することは出来ない。これは、Sun SPOT で採用されている MIDP の制限によるものである。Java SE で実装されている RMI ではシリアライズという技術を使用してオブジェクトや特殊な型の変数をネットワークを通してリモートオブジェクトに渡している。シリアライズは、Java で扱われるオブジェクトをバイト列に変換して送信する技術である。これによりオブジェクトなどのプリミティブ型以外の情報をすべてバイト列にすることでネットワークを通してのデータの通信を可能としている。Sun SPOT ではこれらのシリアライズ技術が実装されていないため、SSRMI ではプリミティブ型、String 型以外の型の引数、返り値の使用が制限されている。

また、RMI ではメソッドの返り値としてリモートオブジェクトの Stub を渡すことも可能である。これは、Stub の生成に必要な情報を送信してスタブを生成し、その参照を渡している。リモートオブジェクトの Stub の参照を渡す事は SSRMI でも実装可能である。しかし、現段階の SSRMI での実装では RMI の Sun SPOT への適用を探る段階のため、実装は行っていない。

5. SSRMI の使用

SSRMI では、基本的な使用法は RMI とほぼ変わらないが、SSRMI 独自の設計や機能により一部の API や使用法が異なっている。ここでは SSRMI を使用して開発をする場合の、手順に沿って API とその使用法を図 7 を使用して説明する。図 7 はクライアント SPOT からサーバ SPOT 上のリモートオブジェクトのメソッド hello() を呼び出し文字列 "Hello World" を受け取るプログラムである。

5.1 SSRMI の準備

SSRMI を使用するためには、メソッド呼び出しを行うリモートオブジェクト、リモートオブジェクトのインターフェースを用意する必要がある。また、リモートオブジェクトをレジストリに登録するサーバオブジェクト、リモートオブジェクトのメソッドを呼び出すクライアントオブジェクトを用意する必要がある。

5.2 Stub と Skeleton の生成

SSRMI では、ネットワークを使用するための部品として、Stub と Skeleton を準備する必要がある。これは以下のようにメソッド呼び出しを行うリモートオブジェクトに対して ssrmic を実行することで自動生成される。

```
>ssrmic HelloWorld
```

ssrmic を実行するクラスファイルは必ず SSRMI で用意されている UnicastRemoteObject クラスを継承し、オブジェクトのインターフェースを implements する。リモートオブジェクトに対して ssrmic コマンドを使用し、生成された Stub と Skeleton をそれぞれサーバ SPOT とクライアント SPOT に設置し、リモートオブジェクトとそのインターフェースをそれぞれの SPOT に設置する。

5.3 SSRmiregistry の起動とオブジェクトの登録

サーバオブジェクトの置かれるサーバ SPOT 上では、SSRmiregistry を起動し、リモートオブジェクトを SSRmiregistry へ登録する必要がある。

まず、SSRmiregistry クラスを使用し、

```
SSRmiregistry registry = new SSRmiregistry(250);
```

という処理で、ポート 250 番を使用して SSRmiregistry のインスタンスを生成する。続いて SSRmiregistry を起動するために、

```
reg.serverRun();
```

という処理を行い、SSRmiregistry を開始する。さらに、起動した SSRmiregistry に

```
Naming.bind("ssrmi://レジストリ SPOT アドレス:250/hello", hello);
```

を実行し、リモートオブジェクトの登録を行う。引数にはレジストリの置かれている SPOT の MAC アドレス、ポート番号、リモートオブジェクト名、リモートオブジェクトのインスタンスを指定する。

5.4 SSRmiregistry へのオブジェクトの検索とリモートメソッド呼び出し

クライアント SPOT では、SSRmiregistry へリモートオブジェクトの検索をかける必要がある。検索の結果、リモートオブジェクトの Stub のインスタンスが取得できる。

まず、SSRmiregistry の起動されているサーバ SPOT に対して、

```
HelloWorldIf hello =
```

```
(HelloWorldIf)Naming.lookup("ssrmi://レジストリ SPOT アドレス:250/hello");
```

を実行し、リモートオブジェクトの Stub のインスタンスを取得する。引数にはレジストリの置かれている SPOT の MAC アドレスとポート番号、登録したリモートオブジェクト名を指定する。

SSRmiregistry への lookup が成功すると、クライアントオブジェクトからリモートオブジェクトを呼び出すことが可能となる。lookup() により取得したインスタンスに対して、

```
hello.hello();
```

を実行し、リモートメソッド呼び出しを行う。

5.5 Exception

SSRMI はネットワークを使用したリモートメソッド呼び出しフレームワークである。その性格上、ネットワークの異常を含め、プログラムの処理が正常に行われない可能性があるため、SSRMI では RemoteException を始め様々な Exception を用意している。SSRMI を使用するユーザは Exception に合わせたエラー処理を記述する必要がある。

```
protected void startApp() throws MIDlet State Change Exception{
    HelloWorldIf hello;
    hello = (HelloWorldIf)Naming.lookup("ssrmi://レジストリ SPOT アドレス:250/hello");
    System.out.println(hello.hello()); // リモートメソッドの呼び出し
}
```

Client SPOT

```
protected void startApp() throws MIDlet State Change Exception{
    SSRMiregistry reg = new SSRMiregistry(250);
    reg.serverRun (); // レジストリの起動
    HelloWorld hello = new HelloWorld(); // リモートオブジェクト
    Naming.bind("ssrmi://レジストリ SPOT アドレス:250/hello", hello); // レジストリへ登録
}
```

Server SPOT

```
public class HelloWorld extends UnicastRemoteObject implements HelloWorldIf{
    public String hello() throws RemoteException{
        return "Hello World"; // 文字列を返す
    }
}
```

Remote Object

```
public interface HelloWorldIf extends Remote{
    public String hello() throws RemoteException;
}
```

Remote Object Interface

図 7 SSRMI サンプルコード
Fig. 7 SSRMI Sample code

6. SSRMI の評価

評価を得るために、今回は図 7 で用意した SSRMI を使用したリモートメソッド呼び出し

プログラムと同じ動作をする "Hello World" という文字列を Radiogram を使って直接送受信するプログラムを用意し、その実行速度を比較・評価を行う。この評価については、実機の Sun SPOT を用意し、同じ SPOT にそれぞれプログラムを実装し同じ条件下で速度を計測しその速度の差について検討する。

実際にそれぞれのプログラムを 5 回実行した場合の実測結果を図 8 に記載する。計測を行う方法は Java で用意されている API である System.currentTimeMillis(); を使用し計測する。この API は現在の時刻をミリ秒まで取得する API で、処理の開始、終了時にそれぞれ計測し、その差を実行時間とする。今回得られる実行時間は、処理の要求、処理の実行、処理結果の返信の時間の和である。処理の実行時間について、それぞれ同じ文字列を送信するという処理内容のため処理に必要な時間は同一とし、処理時間の差は通信に消費されたものとする。

	1	2	3	4	5	平均
Radiogramによる送受信	480	480	474	481	479	478.8
SSRMI	704	718	730	710	711	714.6

(msec)

図 8 評価プログラムの実行結果
Fig. 8 Result of the evaluation program

今回の実行結果から SSRMI の方が 1.5 倍程度実行に時間が必要であることがわかった。これは、SSRMI の実装においてヘッダ情報などの付加によるオーバーヘッドの増加と考えられる。

しかし、ソースコードの記述量を比較すると SSRMI で記述した場合は 45 行、radiogram で直接記述した場合は 65 行と、radiogram で記述した方が SSRMI の 1.5 倍必要となり、処理が複雑になればより大きな差となる事が予想される。このことから SSRMI を使用する事でプログラムの開発速度が向上すると考えられる。

7. 関連技術

J2ME 向けの RMI の仕様としては、J2ME RMI Optional Package(以下 RMI OP)⁵⁾ が定められている。RMI OP は、J2SE RMI のサブセットで、J2ME の CDC に準拠する機

器で使用可能である。RMI OP でサポートされていない機能の代表的なものに、HTTP トンネリングやアクティベーション、スタブやスケルトンコンパイラがある。RMI OP の標準実装は、CDC に基づくプロファイル Foundation Profile, Personal Basis Profile, Personal Profile などの機器で使用できる。しかし、SunSPOT は、CLDC に準拠しているため、RMI OP は使用できない。CDC は、J2ME に分類されるが、通信機能のある程度保有する機器向けであり、CLDC は、制限された通信機能しか持ち合わせない機器向けということになる。CLDC の方が規模が小さい。

CLDC のプロファイルである MIDP 向けの RMI 関連技術としては、MeRMI⁷⁾ や Middleman Architecture⁶⁾ が提案されている。MIDP は、CLDC のネットワーク機能を向上させるため、HTTP プロトコルをサポートしている。しかし、CLDC も MIDP もソケットやデータグラム通信機能は持っていない。もちろん、RMI も使用できない。従って、ソケットあるいは RMI ベースのサーバアプリケーションを MIDP 準拠の機器から呼び出すことはできない。この問題の解決法として、MeRMI や Middleman Architecture がある。

MeRMI では、J2ME のクライアントが J2SE のサーバのメソッドをネットワーク経由で呼び出せるように、RMI と同様のリモートメソッド呼び出しの機能を実現している。MeRMI の標準実装では、シリアル化されたデータを HTTP プロトコルで通信する手法で実装されている。

Middleman Architecture では、サーブレットを用いて、ソケットベースの接続や RMI のメソッド呼び出しの中継機能を行わせる。MIDP デバイスは、Middleman サーブレットと HTTP で通信する。サーブレットはクライアントからのリクエストを解析して、ソケット通信や RMI メソッド呼び出しを行い、その結果をクライアントへ返す。

MeRMI と Middleman Architecture は、MIDP デバイスと一般の J2SE サーバとの接続性を確保することが目的である。一方、SSRMI は SunSPOT 同士のネットワーク接続を実現しようとするものであり、その利用分野は異なる。また、SunSPOT では HTTP プロトコルはサポートされていないため、HTTP を用いた実装も適用できない。

8. まとめと今後の課題

本稿では、Sun SPOT 向けに動作する RMI として SSRMI を提案し、SSRMI についての基本的なアーキテクチャを挙げ、実際に SSRMI の Sun SPOT への実装を行い、実装においての問題点や、その問題の解決方法、実装方法を挙げた。また、SSRMI の使用方法や SSRMI の速度評価などを行い、SSRMI の有用性を検討した。

今後の課題は、SSRMI の仕様拡張について、4.5 で挙げられているメソッド呼び出しでの戻り値での Stub の扱いについて拡張を検討する。この拡張により、SSRMI での利用方法の幅が広がる事が考えられる。また、SSRMI の実行速度についても最適化を図り実行速度の向上も検討していく。現段階では実用に耐える速度をマークしているがリアルタイムでの処理などで処理速度が要求される場合がある可能性もあるためである。また、様々なプログラムに対しての SSRMI の有用性についても検討していき、多種多様なプログラムに対して SSRMI を使用し正常に使用できるか、また不具合などの有無なども確認していく。以上が今後の課題である。

参 考 文 献

- 1) Sun Microsystems : 無線センサーネットワークデバイス,
<http://jp.sun.com/products/software/sunspot/>
- 2) Sun Microsystems : Java Remote Method Invocation,
<http://java.sun.com/j2se/1.5.0/docs/guide/rmi/index.html>
- 3) Sun Microsystems : MIDP,
<http://java.sun.com/products/midp/>
- 4) 杉山安洋 : TRMI によるオブジェクトの分散化, コンピュータソフトウェア, Vol.19, No.5, pp.40-59, 2002.
- 5) Sun Microsystems : J2ME RMI Optional Package,
<http://java.sun.com/products/rmiop/>
- 6) Sun Microsystems : Advanced MIDP Networking, Accessing Using Sockets and RMI from MIDP-enabled Devices,
<http://developers.sun.com/mobility/midp/articles/socketRMI/>
- 7) MeRMI, <https://mermi.dev.java.net/>