

イベント付随型 RMI によるサービス連携機構

谷川 善紀[†] 西尾 信彦[‡]

[†] 立命館大学大学院理工学研究科 〒561-2741 滋賀県草津市野路東 1-1-1

[‡] 立命館大学情報理工学部 〒561-2741 滋賀県草津市野路東 1-1-1

E-mail: [†] nori@ubi.cs.ritsumei.ac.jp, [‡] nishio@cs.ritsumei.ac.jp

あらまし 様々な機器やアプリケーションが相互に接続される分散サービス環境において、イベントの利用はサービスの連携を定義する上で有効な手段の1つである。イベントはサービスの状態やセンサ情報の変化を通知するために利用される。サービスの実行の通知も有効なイベントの1つと考えられるが、サービスごとにそのための記述をするのは負担である。そこで、分散サービスプラットフォームのリモート通信部を拡張し、サービスが実行される度にイベントを発生させる機能を持たせる方法を提案する。また、このイベントを利用したサービス連携機構を構築した。この機構ではサービスの連携をスクリプトで定義することで記述の自由度を実現している。本稿では、それぞれの設計と Jini での実装について述べる。

キーワード サービス連携, イベント, Jini, RMI

Event-Trigering RMI for Service Cooperation

Yoshinori Tanigawa[†] Nobuhiko Nishio[‡]

[†] Graduate School of Science and Engineering, Ritumeikan University 1-1-1 Nojihigasi, Kusatsushi, Shiga, 561-2741 Japan

[‡] Department of Infomation Science and Engineering, Ritumeikan University 1-1-1 Nojihigasi, Kusatsushi, Shiga, 561-2741 Japan

E-mail: [†] nori@ubi.cs.ritsumei.ac.jp, [‡] nishio@cs.ritsumei.ac.jp

Abstract Ubiquitous computing environment is composed of various types of applications and devices whose services are provided via networks. In this environment, the event-driven programming is an appropriate approach for service cooperation. Events are generated to notify the state of service and sensor. However, it is necessary to programming the code in services to generate events. Then, we propose a method to extend RMI system to generate an event when a specified method of service is invoked. We also enabled to script the service cooperation using the third party's scripting language as an independent module of our system. This article explains its design and implementation on Jini and Groovy.

Keyword Service Cooperation, Event, Jini, RMI

1. はじめに

近年、家庭やオフィスへのインターネットの普及とともに、情報家電など、ネットワークに対応したさまざまな機器が登場している。情報家電化により、携帯電話を利用した遠隔からの情報家電の操作や、一括ボタンによる情報家電の連動、インターネットから得られる情報との連携動作などのサービスが提供されている。また、家電だけではなく、PCやPDA、住宅設備、センシング技術などとも連携して、より高次のサービスが創出されていくと期待される。

このような技術を支える基盤として、ECHONET[1]やUPnP[2]、Jini[3]などが存在する。Jiniは、さまざまな機器をネットワークに接続し、管理するためのJava技術を基盤としたプラットフォームである。Jiniは、プリンタやAV機器、アプリケーションなどをサービスとして定義し、サービスの登録や発見、利用などの機能を提供する。

さまざまなサービスが遍在する中で、それらを互いに連携させるために、イベントを連携の条件に利用することは有効な手段の1つである。イベント発生のきっかけとしては、サービスの状態変化や、センサからの情報などが考えられる。また、サービスが実行された時にイベントが発生するようにすれば、それを使ったサービス連携が実現できると考えられる。例えば、DVDが再生されると部屋の照明を落したり、TVがつけられるとPCに番組表が表示されるようにするなどできる。しかし、全てのサービスで実行時にイベントが発生するようにプログラムを書き換えるのは、開発者にとって負担になると考えられる。そこで本研究では、RMI[4]などのサービスへのリモート通信を監視し、イベントを発生させるという手法を提案する。RMIはJavaで実装された多くの分散アプリケーションで使用されており、適用範囲が広いと考えられる。また、このイベントを利用してサービスを連携させるための

機構のプロトタイプを作成した。実装には Jini を使用し、通信プロトコルには Java RMI の Jini 版実装である JERI を使用した。また、プロトタイプではサービスの連携を定義するための記述言語に、Java 実装のスクリプト言語である Groovy[5]を採用した。Groovy は平易に記述できる文法をもつ反面、ほぼ全ての Java 文法もサポートしている。また、Java のライブラリーのインポートや、逆に Java プログラムからのロードにも対応している。したがって、簡易性と自由度を両立したサービスの連携定義の記述が可能となる。

関連研究として、uBlocks[6]がある。uBlocks では、サービスがアイコンで表示され、それらを矢印で視覚的に接続して、容易にイベントドリブンのアプリケーションを構築できる。しかし、イベントを利用するためには、サービスにイベント発生プログラミングがされている必要がある。本研究は、RMI にイベントを発生する機能を持たせるため、サービスにイベント発生のための記述がほとんど必要ない。また、サービスの連携を外部のスクリプトファイルで記述できるので、自由度の高いサービス連携の指定が可能である。

2. 設計

2.1. システムの概要

本システムでは、サービスへの RMI 通信が行われるごとにイベントが発生し、それらのイベントを利用してユーザが容易にサービスの連携を定義できる環境を提供する。本システムは、イベントを監視し連携を行うためのサービス連携管理サービス、連携を定義するためのツールである RelationEditor から構成される。まず、サービスがサービス連携管理サービスに自身の持つ機能の情報を登録する。ユーザは RelationEditor を使ってサービス連携管理サービスにアクセスすると、利用可能なサービスの情報が GUI により表示される。そこで、連携開始のトリガーとしたいイベントと、連携させるサービスを選ぶことにより、ユーザの好みのサービス連携を定義することができる。

2.2. イベント発生機構

サービスが実行されるごとにイベントを発生する機構を設計する。Jini, CORBA[7], Web サービス[8]などに対応した分散サービスを構築するためのフレームワークの多くでは、サービスをロジック部分と通信部分に分離しており、プログラマがロジックを記述すれば、通信を行うための部分はフレームワークにより自動生成される。そこで、通信部分に、サービスの実行を監視してイベントを発生させる機構が組み込まれるように、フレームワークを拡張する。

2.3. サービス連携機構

2.3.1. サービス連携管理サービス

サービス連携管理サービスは、イベントドリブンのサービス連携を実現するための機構であり、図 1 にも示すように、主に次の 3 つの機能から構成される。

サービス管理機能

現在利用可能なサービスを把握と、サービス情報の管理をおこなう。サービス情報には、利用可能な機能やそのサービスから発生するイベントの簡単な説明を格納する。これは、ユーザによる即興的なサービスの連携定義を支援するためである。

連携定義管理機能

サービスの連携定義の登録を管理し、登録された連携定義をもとに、サービスの実行を行う。

イベント管理機能

イベントを待ち受ける。

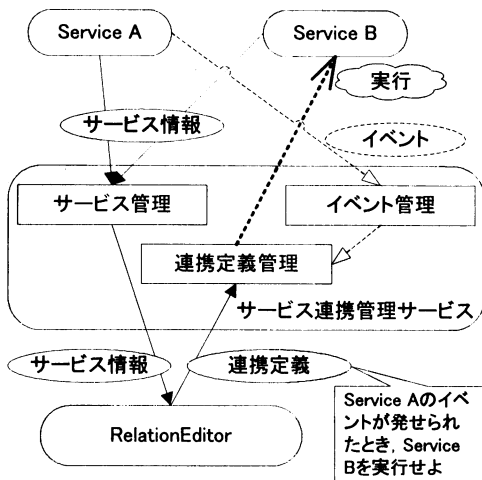


図 1. サービス連携管理サービス

2.3.2. 連携エディタ

サービスの連携定義はスクリプト言語により記述する。しかし、エンドユーザにはプログラミングの敷居が高いといえる。RelationEditor は、エンドユーザによるサービス連携の定義を支援するためのツールである。RelationEditor はまず、サービス連携管理サービスからサービス情報を得て、ユーザが利用可能なサービスと、サービス連携を開始するトリガーとして利用できるイベントの一覧を表示する。ユーザはそれらを GUI で自由に選択することにより、簡単にサービスの連携定義を作成することができる。

3. 実装

Java の標準開発キットである J2SE SDK では、java.rmi.以下のパッケージにより RMI の実装が提供され、サービスのサーバ化はこのパッケージに含まれる

UnicastRemoteObject クラスを継承することにより実現される。プロトタイプの実装にはこれを利用するほうが、多くの Java の分散アプリケーションにそのまま適用できるという意味で、有効であると考えられる。しかし、本システムのプロトタイプ実装には、Lookup サービスや、リース、Discovery/join、分散イベントなど、分散サービス構築のための環境がより強化された Jini をプラットフォームとして使用した。Jini の RMI は、Jini Extensible Remote Invocation (JERI) で実装されており、サービスのサーバ化には BasicJeriExporter が使用される。

3.1. イベント発生機構

3.1.1. MethodInvocationEvent クラス

サービスが実行、すなわち、サービスのメソッドが実行されたときに発生するイベントを MethodInvocationEvent クラスとして図 3 のように実装した。MethodInvocationEvent クラスはフィールドとして、イベントの発生源への参照をあらわす source、メソッド名をあらわす methodName、メソッドの引数の型をあらわす argTypes をもつ。

```
public class MethodInvocationEvent
    extends net.jini.core.event.RemoteEvent {
    protected String methodName;
    protected Class[] argTypes;
    /**
     * @param source イベントの発生源への参照
     * @param methodName メソッド名
     * @param argTypes 引数の型の配列
     */
    public MethodInvocationEvent(Remote source,
        String methodName, Class[] argTypes){
        super(source, 0, 0, null);
        this.methodName = methodName;
        this.argTypes = argTypes;
    }
}
```

図 2. MethodInvocationEvent ソースの一部

3.1.2. RMI の監視とイベント発生

Jini の通信プロトコルでは、Java RMI の実装である Jini Extensible Remote Invocation (JERI) が使用される。Jini フレームワークでは、プログラマが作成したサービスのロジックを Jini のライブラリーに含まれる BasicJeriExporter クラスに渡して Export することにより、サービスのサーバ化が行われ、同時にプロキシが生成される。プロキシとはサーバにアクセスするための、スタブにあたるものである。図 3 は Export の様子、及び JERI のプロトコルスタックをあらわしている。Export されたサービスはレイヤー構造を持つが、ServiceLogic 以外の全てが Jini のフレームワークによ

り自動生成されるクラスで構成される。実装では、RMI により呼び出されるメソッドの監視機能と MethodInvocationEvent クラスを発生する機能をもった InvocationDispatcher を生成するように、Exporter を拡張した。

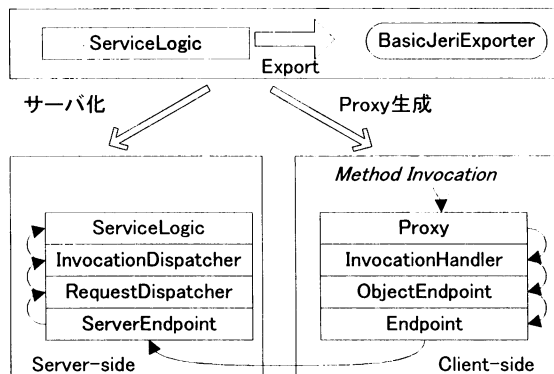


図 3. JERI プロトコルスタック

3.2. サービス連携機構

3.2.1. サービス連携管理サービス

サービス情報

サービス連携管理サービスに登録するために、サービス情報を格納するパラメータクラス（図 4）を定義した。

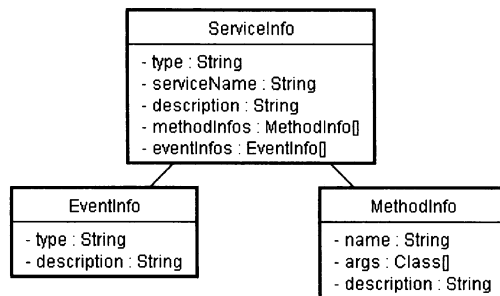


図 4. サービス情報を格納するクラス

サービス連携定義

サービス連携定義の記述言語には Groovy スクリプトを採用した。Groovy スクリプトでは、変数の動的型付けや例外処理の省略などが実現されており、コーディングの簡略化を実現できる。また、Groovy スクリプトは Java との親和性が高く、自由度の高いプログラミングが可能である。

サービス連携管理サービスはサービス連携定義が登録されると、ScriptExecutor クラスをスレッドとして開始する。ScriptExecutor スレッドは、Groovy スクリプトを Java クラスに変換し、インスタンス化の際にスレッドへの参照を引数として渡す。Groovy スクリプト

ではこの参照を用いてイベント処理に関する記述をおこなう。

イベント管理

サービス連携管理サービスのプロキシオブジェクトを、サービスのイベント発生機構にリスナーとして登録することにより、イベントの通知を実現した。受け取ったイベントは、連携定義管理部に通知される。

3.2.2. 連携エディタ

図5に実装した連携エディタのGUIを示す。左側のリストからトリガーとしたいイベントを選び、右側のリストで実行したいサービスを選ぶ。スクリプト生成ボタンを押すと指定した名前で図6のようなGroovyスクリプトファイルを生成する。これをサービス連携管理サービスに登録すると、スクリプトが実行される。

図6で示すスクリプトはメディアプレイヤーでメディアが再生されると、照明を消すといった連携動作をおこなう。MethodInvocationEventTemplateクラスとServiceProxyクラスは記述を簡単化するために独自に作成したクラスである。

```
import relationmanagement.*;
import relationmanagement.script.*;
import net.jini.core.event.RemoteEventListener;
/**
 * ScriptExecutorにより最初に行われるメソッドです
 * @param executor 呼び出し元 (ScriptExecutor)
 */
def startScript(executor){
    // 利用したいMethodInvocationEvent
    // の特徴を指定
    event1 = new MethodInvocationEventTemplate();
    event1.setServiceType("service.MediaPlayer");
    event1.setMethodName("play");
    // 指定した型のサービスを探索し、
    // プロキシを取得
    service1 =
        new ServiceProxy("service.LightService");
    // event1を受け取るまで待機
    executor.setTrigger(event1);
    // サービスのoffPowerメソッドを実行
    service1.invoke("offPower");
}
```

図 5. RelationEditor が生成したスクリプト



図 6. 連携エディタ

3.3. Jini への依存性

実装したプロトタイプの Jini への依存性と、J2SE で提供される標準的な RMI 実装への移植性について述べる。表1はプロトタイプ実装で最低限必要となる機能と、Jini と J2SE SDK それぞれでの実現方法を示す。J2SE では UnicastRemoteObject クラスがサービスのエクスポートをおこなうので、これを拡張すれば RMI を監視してイベントを発生させることができる。また、J2SE にはリモートイベントをサポートしていないため、これに代わるものを作成する必要がある。

表 1. Jini と J2SE での実装の違い

機能	Jini	J2SE
サービスの検索	Reggie サーバ	rmiregistry サーバ
サービスの Export	BasicJeriExporter クラスに渡す	UnicastRemoteObject クラスを継承する
リモートイベント	RemoteEvent クラス RemoteEventListener クラス	用意されていないので、実装が必要

4. まとめと今後の課題

本研究では、RMI 実行時にイベントを発生させ、イベントをトリガーとして、サービスの連携を記述したスクリプトを開始する機構の設計と実装について述べた。現在の実装の問題点として、イベントの取得やサービスの実行に利用権限が考慮されていないという点がある。また、サービス連携開始のトリガーとなるイベントには MethodInvocationEvent しか利用できない。サービスの状態変化や、センサ情報などもイベントとして利用できれば、より自由なサービス連携を記述できるようになると考えられる。これらを今後の課題としたい。

文 献

- [1] ECHONET CONSORTIUM
<http://www.echonet.gr.jp/>
- [2] UPnP FORUM
<http://www.upnp.org/>
- [3] Jini Network Technorogy
<http://www.sun.com/software/jini/>
- [4] JGuru: Remote Method Invocation(RMI)
<http://java.sun.com/developer/onlineTraining/rmi/RMI.html>
- [5] Groovy
<http://groovy.codehaus.org/>
- [6] Masayuki Iwai, Jin Nakazawa, Hideyuki Tokuda, "uBlocks:Enabling Hand-made Distributed Application among Ubiquitous Embedded Devices", IEEE Workshop on Software Technologies for Future Embedded Systems, May. 2003 pp.57-60
- [7] OBJECT MANAGEMENT GROUP
<http://www.omg.org/>
- [8] W3C Web Services Activity
<http://www.w3.org/2002/ws/>