システムソフトウェアと
オペレーティング・システム 81－24
（1998．5．7）

Javaによる非同期メッセージ配送フレームワーク ECJ

宮澤 隆幸　海邊 裕
（株）東芝　研究開発センター

イベント駆動型の分散 Java アプリケーションを構築するためのフレームワーク ECJ(Event Centric for Java)を開発した。ECJ は、Java 標準の RMI を補完する、イベント駆動型の非同期通信モデルを提供する。ECJ はイベント駆動型システムの基本要素であるイベントディスパッチャとイベントハンドラに加えて、各種デバイスなどの外部システムとのイベント交換機能を提供する ECJ ドライバにより構成される。この構成により、多様な通信プロトコルへの対応や他のイベントシステムの統合が容易になった。

ECJ: Asynchronous Messages Delivery Framework for Java

Takayuki Miyazawa　Hiroshi Kaibe
Research and Development Center, Toshiba Corporation

We have developed ECJ(Event Centric for Java), which is a framework for developing event driven distributed Java applications. ECJ provides an asynchronus event driven communication model, which complements Java RMI. It consists of an event dispatcher, event handlers and ECJ drivers. ECJ facilitates to use various kinds of transport protocols and integrate with other event driven systems.

1. はじめに

SunMicrosystems 社が開発し普及・標準化活動を展開している Java 言語環境は、そのプラットホーム独立性により分散システムのベース環境として注目されている。

現在 Java の主流の開発環境である JDK-1.1 および Java2 では、分散システムを構築するための機構として RMI[1]が利用できる。RMI はオブジェクト呼び出しの位置透過性を提供する。RMI では遠隔手続き呼び出し型の通信が用いられる。遠隔手続き呼び出し型の通信は、呼び出される手続きを提供する通信相手の応答を待つ必要がある。この通信相手の応答を待つ通信を同期通信とする。同期通信は、応答を待つ相手を知る必要があるため、不特定多数との通信が困難である。

一方、同期通信に対して通信相手の応答を待たない非同期通信がある。非同期通信は、応答を待つ相手を知る必要がないため不特定多数との通信も容易である。さらに非同期通信は、同期通信と比較して送信間隔を短くできる。しかし、Java では非同期通信を使用するためには、アプリケーション開発者が通信に必要なすべての処理を記述する必要があり、開発者に大きな負担がかかる。そこで我々は、RMI を補完する非同期通信機構が必要であると考える。

また、GUI などに利用されているプログラミングモデルとして、イベント駆動型モデルがある。このモデルは、イベントの送信者と受信者が分離されており、非同期通信のプログラミングモデルとして利用できる。またこのモデルは、Java において AWT や JavaBeans のコンポーネント間の通信として使用されているため、アプリケーション開発者にとって理解が容易であると思われる。そこで、我々は非同期通信とイベント駆動型モデルを融合したイベント駆動型非同期通信モデルを提案する。

本稿では、イベント駆動型の非同期通信モデ

ルと、その実装である ECJ(Event Centric for Java)について述べる。以下第2節でイベント駆動型非同期通信モデルについて提案を行う。第3節で ECJ の概要について述べる。第4節で評価を行い、第5節で特徴を述べる。最後にまとめと今後の課題を述べる。

2. イベント駆動型非同期通信モデル

2.1. 非同期通信

本稿では、同期通信を、通信相手の応答を待つランデブー通信と定義し、非同期通信を、通信相手の応答を待たない通信と定義する。この非同期通信は、送信者が応答を待つ相手を知る必要がないため、一対一だけでなく、一対多や多対一、あるいは多対多の通信を使用できる。さらに、この非同期通信は、受信者の応答を待たないため、同期通信と比較して送信間隔を短くすることができる。

2.2. イベント駆動型モデル

GUI システムや、制御デバイスの割り込みモデルで利用されるプログラミングモデルとして、イベント駆動型モデルがある。このモデルは、特定の形式のメッセージであるイベントを送信者から受信者へ引き渡すことで、プログラムを記述する。

本稿におけるイベント駆動型モデルを図 1 に示す。このモデルは、一つ以上の種類を持つイベントと、一つ以上のイベント供給者と、一つのイベント配送者と、一つ以上のイベント消費者から構成される。イベントには形式と種類がある。イベントの形式とは、イベントを表現するデータの構造である。イベントの種類とは、イベントの意味に基づいてイベントを分類したものである。イベント供給者は、ある一つの形式の、一つ以上の種類のイベントを生成する。イベント配送者は、イベント供給者からイベントを受け取りイベント消費者に引き渡す。イベント消費者は、イベントを受け取って処理を行う。またイベント消費者は、ある一つの形式の、一つ以上の種類のイベントを供給することを可能とする。よって、多段のイベント駆動システムを構成できる。

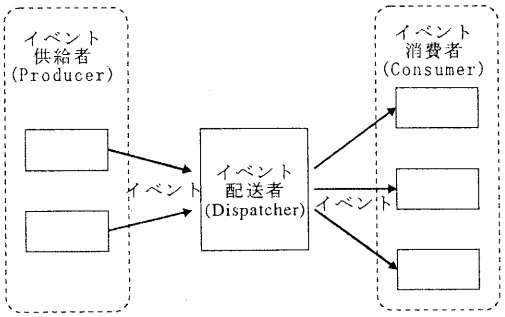

図 1 イベント駆動型モデルの基本構成

このモデルでは、あらかじめイベント配送者にどの種類のイベントをどのイベント消費者に配送するのかを登録しておき、その登録情報に従ってイベント配送者がイベントの配送を行う。このためイベント供給者はイベント配送者のみを知っていればよく、最終的な通信相手であるイベント消費者については知る必要がない。また逆にイベント消費者はイベント配送者からイベントを受け取るだけで、そのイベントの供給者については知る必要がない。つまり、イベント駆動型モデルでは、イベント配送者によってイベント供給者とイベント消費者が分離されている。

2.3. イベント駆動型非同期通信モデル

イベント配送者をイベント供給者からイベントを受け取る部分である A と、イベント消費者へイベントを引き渡す部分である B の 2 つに分け、A,B 間をネットワークで結ぶことによりイベント駆動型モデルを分散拡張できる。このモデルを図 2 に示す。この図において、実線の矢印はイベントの流れを表し、点線はネットワーク上のデータの流れを表す。

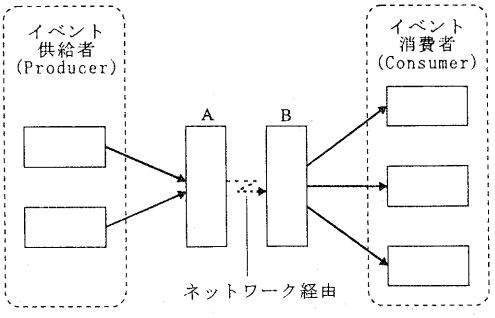

図 2 イベント駆動型モデルの分散拡張

図 2 のモデルでは、A,B はイベント配送者ではない。そこでこれらをイベント配送者にしたモデルを考える。A からイベントを受け取り、このイベントの情報をネットワークに送信するイベント送信者 S(Sender)と、送信されたイベントの情報を受信して B に引き渡すイベント受信者 R(Receiver)とを考える。この S,R を使用して、イベント供給者を P (Producer)、イベント消費者を C (Consumer)として図 2 を書き直すと、図 3 のように書ける。

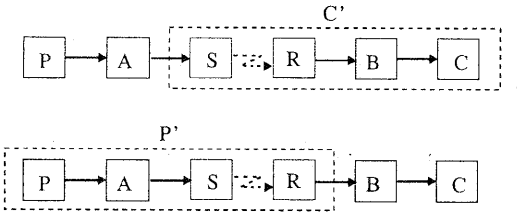

図 3　分散拡張モデル

ここで、図に点線で示したように、S,R,B,C をまとめて C'とし、P,A,S,R をまとめて P'とすることにより、これらはいずれもイベント駆動型の基本モデルと同じになる。すなわち、A,B をイベント配送者(ED)と見なすことができる。そこで、R の代わりにイベントの情報をネットワークから受信する通信機能を持つイベント供給者(P")を、S の代わりにイベントの情報をネットワークに送信する通信機能を持つイベント消費者(C")を考えることにより、イベント駆動型の通信モデルを作ることができる。このモデルを図 4 に示す。これが、我々の考えたイベント駆動型非同期通信モデルである。

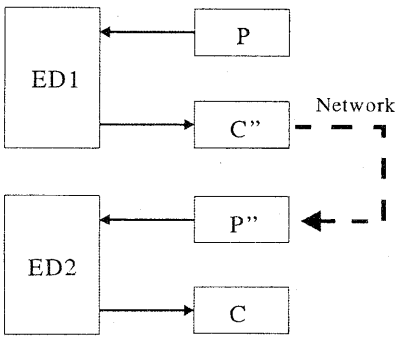

図 4　イベント駆動型通信モデル

このイベント駆動型通信モデルでは、P が送信者、C が受信者となる。P は C の応答を待たず、また P が C を知る必要がないことから、これは非同期通信である。

2.4.　イベント駆動型通信モデルの拡張

図 4 のモデルでは同じ形式のイベントを扱うイベント駆動システム間の通信を対象としているが、P"または C"でイベントの形式を対象となるイベント配送者の形式に変換することにより、異なる形式のイベントを扱うイベント駆動システム間の通信も可能である。このイベント形式変換機能を持つ、イベントの供給者および消費者を P"', C"'とする。ここで、イベント形式変換を持つということは他のイベント配送者との通信機能を持つということであるから、通信機能だけを持つ P", C"は P"', C"'の中でイベント形式変換を行わないものと考えることができる。イベント消費者はイベント供給者になることができることと合わせて、図 4 を書き直したものが図 5 である。これが、拡張したイベント駆動型非同期通信モデルである。

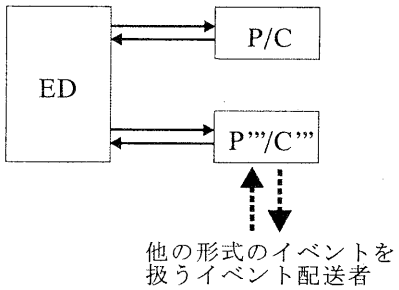

図 5　イベント駆動型通信モデルの拡張

3.　ECJ の概要

本節では、前節で述べたモデルを元に実装したイベント駆動型非同期通信機構 ECJ の概要を述べる。

3.1.　構成

ECJ は以下の要素から構成される。

- ECJ イベント
- ECJ コア
- ECJ ドライバ

- ECJ イベントハンドラ

ECJ の全体構造を図 6 に示す。

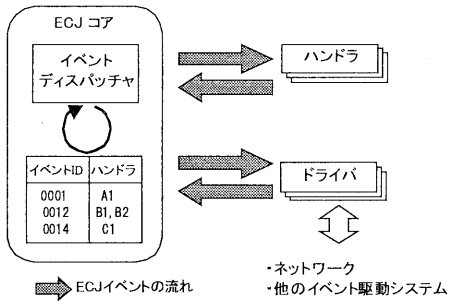

図 6　ECJ の全体構造

我々は Java を用いて ECJ を実装した。ECJ の各構成要素の機能はそれぞれ ECJEvent クラス、ECJLocalManager クラス、ECJDriver-Interface クラス、ECJEventHandler クラスが提供する。ECJ-LocalManager クラスと、ECJ-DriverInterface 、 ECJEventHandlerInterface のクラス関連図を図 7 に示す。我々は、多様な ECJ ドライバを扱うために、Java インタフェースを利用した。

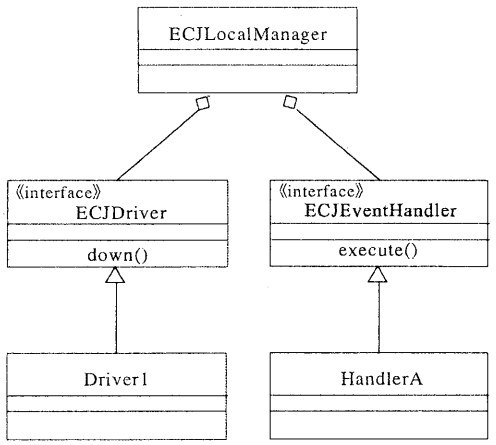

図 7　ECJ のクラス関連図

以下に ECJ の各クラスについて述べる。

3.2.　ECJEvent クラス

ECJEvent クラスはイベント ID とデータを属性として持つ。データは、イベント ID ごとに任意の利用方法が許されている。この ECJEvent クラスとそのサブクラスが ECJ イベントとなる。

イベント ID は、イベント配送者がイベント消費者を決定するための識別子であり、64 ビットの空間を持つ。この空間は先頭 8 ビットが以下に示す種別を表し、下位 58 ビットは種別に応じて割り当てられる。イベント ID の構造を図 8 に示す。

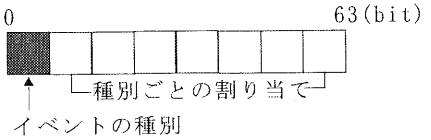

図 8　イベント ID の構造

- ユーザ領域

8~63 ビットは、アプリケーション開発者が自由に割り当てる。

- ドライバベンダ領域

8~31 ビットにはドライバベンダの識別子が割り当てられ、32~63 ビットはドライバベンダが自由に割り当てる。これにより、ドライバベンダごとに独立したイベント ID 空間を持つことが保証される。

- システム領域

ECJ システムが内部で利用する。

- 動的割り当て領域

ECJ が実行時にイベント ID を割り当てる領域である。このイベント ID には、ECJ が動作している計算機の IP アドレスが含まれる。

3.3.　ECJLocalManager クラス

ECJLocalManager クラスは、ECJ イベントの配送機能を提供する。

イベント駆動システムでは、大量のイベントを処理する必要がある。このため、イベント配送機能であるイベントディスパッチャの実装方法による遅延の大小が、システム全体の性能に影響を与える。Java のスレッド切り替えには時間がかかるため、ECJ ではイベント配送をシングルスレッドで処理している。

3.4.　ECJEventHandlerInterface クラス

アプリケーション開発者が、このインタフェ

ースを実装してプログラムロジックを記述する。

3.5. ECJDriverInterface クラス

イベント形式の異なるイベント駆動システムで独自に規定されたイベント群を ECJ イベントに変換する、あるいはその逆変換を行う。

ECJ ドライバの開発者はこのインタフェースを実装して ECJ ドライバを記述する。

3.6. ECJTransportInterface クラス

ECJ イベントをネットワーク経由で転送する機構を提供する ECJ トランスポートのインタフェースである。この ECJTransportInterface クラスは、前述の ECJDriverInterface クラスのサブクラスとして定義される。

ECJ イベントなどのオブジェクトをネットワーク経由で転送する場合、オブジェクトの情報をどのような形にして転送するのかが転送速度に影響を与える。ECJ で対象とするイベントは、単純な属性から構成される。そこで我々は、これらの属性だけを転送する場合は高速に転送することができるように、イベント ID とバイト列のデータのみを転送するようにした。さらに、ECJEvent のサブクラスを作って属性やメソッドを追加した複雑なイベントを転送したい場合にも対応できるように、Java Object Serialization を使用した直列化を行って ECJ イベントを転送することも可能にした。直列化を行うか行わないかの選択は、アプリケーション開発者がトランスポートの生成時に、直列化を行わないモードと直列化を行うモードのどちらかを指定することによって行う。

4. 評価

4.1. 通信速度

ECJ の基本性能として、同一 LAN 上の 2 台の計算機間をイベントが往復するのに要する時間を ECJ の UDP/IP トランスポートで直列化を使用する場合としない場合で測定した。ECJ の UDP/IP トランスポートは、java.net.DatagramSocket クラスを用いて ECJ イベントを転送する ECJ トランスポートである。また、比較のため java.net.DatagramSocket クラスを直接使用して同じ長さの情報を転送する場合も測定した。ここで往復時間とは、2 台の計算機を A,B とすると、A であらかじめ生成しておいたイベントを B へ送信し、B でそのイベントを受けて、イベントハンドラでそのイベントを送り返し、A で B が送ったイベントを受け取ってイベントハンドラが呼ばれるまでの時間である。java.net.DatagramSocket クラスの場合の往復時間は、ECJ イベントと同じ長さのバイト列を 2 台の計算機間で往復させるのに要する時間である。測定環境は以下の通りである。

- 計算機: SPARC AS7000/U2 2170 および SPARC AS7000/U2 2300
- OS: Solaris 2.5.1 および Solaris 2.6
- ネットワーク:100Base-TX

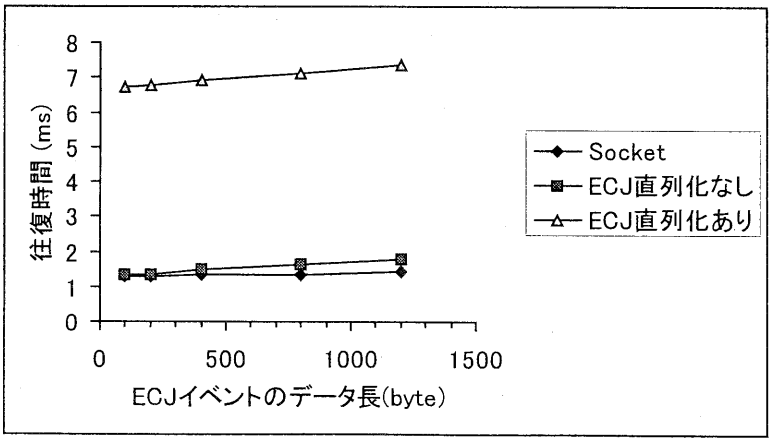

図 9 ECJ の基本性能

2000回の平均往復時間の測定結果を図9に示す。図9において、Socketはjava.net.DatagramSocketクラスを使用した場合(1)、ECJ直列化なしはECJのUDP/IPトランスポートで直列化を行わない場合(2)、ECJ直列化ありはECJのUDP/IPトランスポートで直列化を行う場合(3)を表す。(1)と(2)を比較することによりECJによる遅延が、また(2)と(3)を比較することにより直列化の有無による遅延がわかる。

この結果から、直列化を行わない場合は、ECJはjava.net.DatagramSocketクラスを使う場合と比較して、わずかな遅延でECJイベントを転送できることがわかる。特に、データサイズが200バイト以下の場合はデータグラムソケットクラスを使う場合とほとんど変わらない。また、直列化の処理は通信速度に大きな影響を与えることがわかる。したがって、ECJトランスポートで直列化を行うか否かを選択可能にしたことは有効であるといえる。

5. ECJの特徴

ECJの特徴を以下にまとめる。

通信プロトコルの組み合せ使用

ECJは、通信部分をECJトランスポートとして扱い、ECJイベントハンドラで自由に選択して使用できるようにした。これにより、複数の通信プロトコルを同時に使用でき、例えばあるイベントに対しては到達保証のある通信プロトコルを使用して確実に送り、別のイベントは速度最優先でマルチキャスト通信を行う、といった通信プロトコルの組み合わせ使用が容易に可能となった。

他のイベント駆動システムの統合

ECJはイベント形式変換機能をECJドライバで提供する。これにより、AWTなどの異なる形式のイベントを容易に扱うことができ、イベント駆動システム間のブリッジとして動作させることが可能となった。

記述の容易さ

アプリケーション開発者にとって、新たなプログラミングスタイルを習得することは大きな負担になる。我々は、ECJのプログラミングスタイルをJava標準のAWTと似せて設計した。これにより、アプリケーション開発者は新たなプログラミングスタイルを習得することなくECJを使用することが可能となった。

6. おわりに

本稿では、RMIを補完するイベント駆動型の非同期通信モデルと、その実装であるECJ(Event Centric for Java)について述べた。

ECJは、マルチキャストを容易に扱えることから、多対多の通信を必要とするグループウェアの通信機構として応用できる。また、異なるイベント形式を持つイベント駆動システムを統合できることから、独自のイベント形式を持つイベント駆動システムを採用している既存の制御システムの遠隔監視制御にも応用できる。

現在のECJの実装ではECJイベントの生成および破棄におけるメモリ消費について考慮していないが、組込みデバイスのような、利用できるリソースに制限のある環境ではこれが問題となるため、メモリ消費について今後検討していく。

参考文献

[1] Sun Microsystems, Inc., "Remote Method Invocation Specification", 1996.

[2] Object Management Group, "CORBAservices: Common Object Services Specification", July 1997.

[3] G. Cugola, E. Di Nitto, A. Fuggetta, "Exploiting an event-based infrastructure to develop complex distributed systems", ICSE '98, 20th International Conference on Software Engineering, pp. 261-270, April 1998.