スレッド冬眠技術を用いたワークフローエンジン

1. 背景及び目的

既存の複数のシステムを組み合わせて、より高度な自動化を図る、という事が最近はやっている。大げさには、このような統合はワークフロー、ビジネスプロセスオーケストレーションと呼ばれるが、手近なところでは、メーリングリストに加入する時の次のような電子メールによる本人照合などもこれに含まれる。

1. ユーザーがウェブサイト訪れメーリングリストへの加入を希望する
2. システムがユーザーへ確認のメールを送る
3. ユーザーがメールに返信し再度加入意思を表明する
4. システムが返信を受け取り、ユーザーを追加する

一般に、ワークフローには、一つの「会話」が始まってから完結するまでは比較的長い時間を要し、また1つのシステムが同時に多くの会話を平行して処理する事が多いという特性がある。現在、ワークフローの開発にはBPELなどの専用言語を用いる方法と、イベント駆動的にプログラムを書くという2つの方法がある。

専用言語方式は敷居が高く、通常は編集するために専用ツールが必要になる程複雑である。更にそれに加え、プログラマは手続き部分をJava等の汎用言語で記述し、これら2つを関連付けなくてはならない。このような複雑さは生産性を大きく損なう。

一方、イベント駆動的にプログラムを書くには、会話の各ステップの間（上の例では2と4の間）に、会話に関する情報（上の例ではユーザーのメールアドレスなど）を、プログラマが明示的に永続的な記憶領域から入出力しなくてはならない。また、ステップ4では届いた電子メールから進行中の会話を特定する作業もプログラマが記述する必要がある。更に、会話の各ステップはプログラムの違う場所に記述され、見通しが悪い。このような、処理の本質とは何ら関係のない部分に時間を費やさねばらないため、この方法も生産性を大きく損なう。

もし、上述のようなシステムを、Javaを使って次のように記述できれば、先に述べたような生産性のロスを防ぐことが出来る。

```
void subscribe(MailingList ml, String userEmailAddress) {
        sendConfirmationEmail(userEmailAddress);
        try {
        msg = waitForResponse();
                if(validateMessage(msg))
                        ml.add(userEmailAddress);
        } catch( TimeoutException e ) {
                // abort
```

```
        }
}
```

なぜ現在このように記述することが出来ないかというと、waitForResponse の部分でプログラムの実行が数日から数週間ブロックするからである。各々の会話にスレッドを割り当てていたら計算機の資源が持たないし、またシステムがシャットダウンする時にデータが失われてしまう。

本提案では、実行中の Java スレッドを「冬眠」させる研究 を使って、waitForResponse で返信待ちとなったスレッドの実行状態を、ローカル変数・スタック等を全て含めてディスクに退避し、後で再開できるようにする。この処理はアプリケーションには透過的に行う。

これにより、開発者には上のような単純なプログラムでも、複数の会話を少数のスレッドで処理でき、またサーバーがダウンしても会話の情報が失われることはない。開発者の生産性を損なうことなく、同じ処理を実現できる。

2. 開発の内容

本プロジェクトのソフトウェアは、大きく分けて次の部分から構成される。

1. 継続(continuation)を Java 上に実装する為のライブラリ javaflow の開発
2. これを用いてワークフローを実行する dalma engine
3. dalma engine 上でワークフローアプリケーションと外部世界の入出力操作を仲介する endpoint 群の実装
4. 複数のワークフローアプリケーションを実行・管理・運用する為の dalmacon
5. dalmacon をウェブ上で操作する為の dalma webui
6. これらの開発を効率化するツール maven dalma plugin と dalma ant task
7. 更に、これら開発成果の実用性を検証するため、実際にワークフローアプリケーションを開発

http://dalma.dev.java.net/ を参照のこと。以下、注目に値する幾つかの要素技術について説明する。

javaflow は、関数型言語に見られる「継続」の概念を Java で実現するライブラリである。このプロジェクトは Apache Jakarta commons の sandbox コンポーネントとして開発が開始されていたが、Dalma プロジェクトを開始した頃には開発は停滞していた。このプロジェクトに committer として開発に参加し、以降の開発を中心になって行った。

Javaflow の実装で注目されるべき点は、まず、Java バイトコードを書き換えて継続の機能を実現するバイトコード変換機構である。

バイトコードの変換機能の実装に際しては、Java のバイトコードに関する様々な制約を切り抜ける為にデータフロー解析や様々な書き換えを行う必要があり、またデバッグが難しく、ここが一つの難所であった。この部分に関しては、単体テストを開発して機能退行を自動検出している。

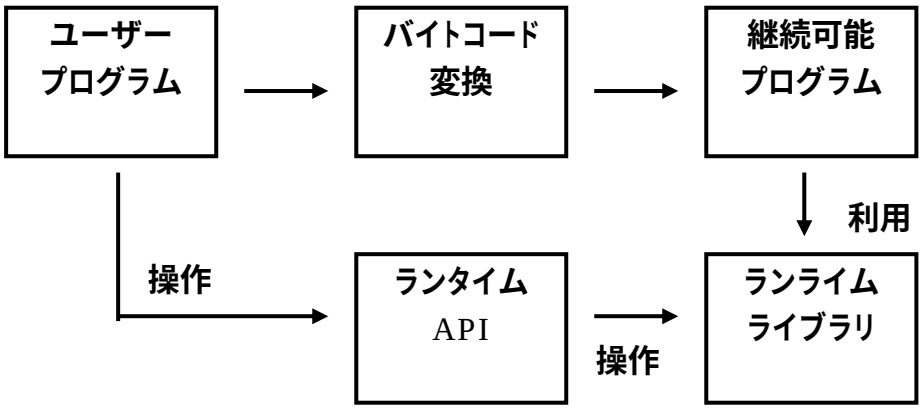

図 1　Javaflow 主要コンポーネント

Dalma engine は、ワークフローの実行をするための中核部分である。

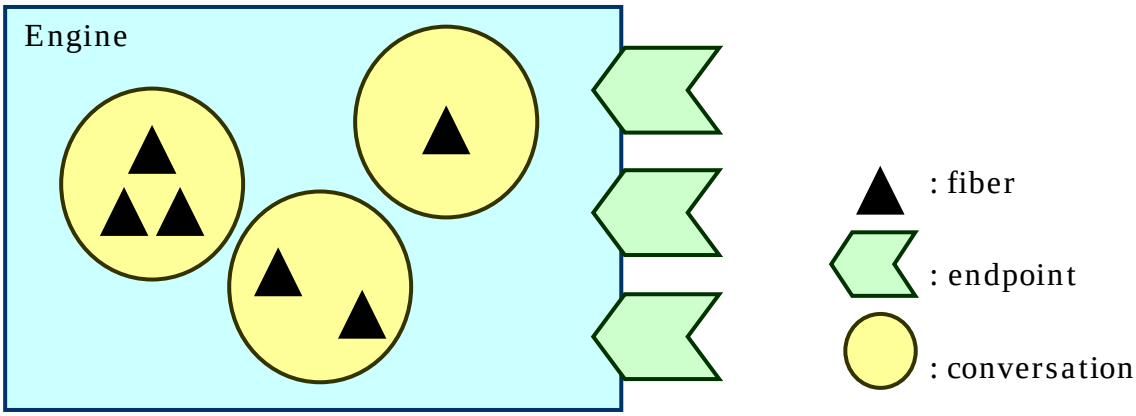

図 2　Dalma Engine 主要コンポーネント

ワークフローの実行インスタンスの一つ一つは conversation と呼ばれる。個々の conversation は独立しており、実行状態にあわせてアプリケーションからは透過的に永続化され、あるいは復元される。夫々の conversation の中にはスレッドに相当する fiber という概念がある。複数の fiber を用いることで、データを共有しながらマルチスレッド的なワークフローアプリケーションを記述できる。

実行中の conversation は、endpoint と呼ばれる入出力レイヤーを通して外界とのデータ入出力を行う（例えば、電子メールを送り、返事を待つなど）。この時に endpoint と engine が協調して conversation を永続化し、また、返事が届いて実行が再開できるようになった時に復元する処理を行う。Endpoint は、個々の入出力の技術に応じて別々のものである（例えば、電子メール、ウェブサービス、タイマー、IRC など）。

Engine は、これら全体を統括し、永続記憶の管理、conversation の開始･終了･スケジューリング、endpoint の起動、ログの管理、エラーに対する対処などを行う。

(開発したソフトウェアの動作環境、構成、機能等を図等を使用して記述)

3. 従来の技術(または機能)との相違

直接的には、継続を陰に用いることによって、プログラムの記述を容易にし、それによって生産性を高める。特に、永続化に関する心配をプログラマからほぼ取り除く事が大きなポイントである。

より本質的なな相違としては、この技術が continuation と呼ばれる計算機科学上の概念に立脚していることにある。この手の処理は continuation をサポートする Scheme のような言語では朝飯前である。しかし、一般の開発者が利用する C, .NET, Java のような汎用言語には今まで continuation が使えなかったし、それが当たり前だと思われていた。

ところが、ここ1年位の間に、次期.NET 2.0 で continuation もどきが実装されたり、ウェブアプリケーションのページフローをコントロールする技術として continuation もどきがサポートされたりして、急速にこの技術の有用性が見直されつつある。しかし、このコンセプトが(汎用言語を使っている開発者達にとっては)あまりに新しいので、まだまだ有効な使い道が多くあるにもかかわらず、それがまだ活用されていない。ここに面白い機会があると私は考える。本提案は、計算機科学では枯れている、この continuation という技術を汎用言語に持ち込んだらどういう事ができるようになるのか、というのを提示したい。

4. 期待される効果

近年エンタープライズ系の開発現場では、ビジネスプロセス統合とよばれるこの種のワークフロー的なニーズが大変高い。Sun でエンタープライズ開発ツールを開発している人達の話を聞くと、ワークフローの特性である会話的なプログラムを記述できるようにするというのは重要な課題であると言う。

上述のようにこれは依然として比較的生産性の低い分野である。Sun も含めてベンダ各社は BPEL を記述するためのツールなど、ツールで複雑さを覆い隠そうという方向に注力しているが、私は見た目の綺麗なだけの GUI ツールでは複雑さを隠すことはできないと思っているし、そういうやり方は嫌いである。本提案によって、こういったニーズをもっと単純に解決するための新しい手法を提案したい。

また、継続は他の分野にも応用の利く技術であるので、このライブラリが新しい分野に転用されるのを期待したい。

5. 普及(または活用)の見通し

内部的に利用しているが、ワークフローエンジンが実際にどの程度使われるかどうかは未知数である。

継続を実装した javaflow ライブラリのほうは、直接のユーザーは他のフレームワークの実装者なので数は限られているものの、それらフレームワークのユーザーに間接的に利便性を提供できると思われる。

6. 開発者名(所属)

川口耕介 (Sun Microsystems, Inc. Web Technologies & Standards group)

(**参考**) http://www.kohsuke.org/
(**参考**) http://dalma.dev.java.net/