大規模エージェントサーバ MobidgetLite *

4U−03

小山和也、富沢伸行、藤田悟、山之内徹†
NEC ‡
e-mail: k-koyama@ax.jp.nec.com

1　はじめに

Web サービスを始めとして、インターネットを介して共通プロトコルにより様々なサービスを利用するための環境が整いつつある。これに伴い、エージェントによる様々なサービスの利用の自動化、支援などの代行処理が可能となるが、このような処理は極めて多数・多種多様でかつ実行に信頼性が求められる。本稿では、このような処理を実現し実行・管理するための実行基盤、エージェントサーバ MobidgetLite について述べる。MobidgetLite はエージェントの実行、管理、保存、メモリ管理などの機能を備え、GUI によるエージェントの運用管理やサーバ障害時のエージェント復旧、少ないメモリでの大量エージェント実行を可能にする。

2　Web サービスとエージェント

従来より、人手によるネットワークサービスの利用を自動化・支援など代行する技術としてエージェントが注目されてきた。これに加えて、近年 XML や HTTP、SOAP などをベースとした Web サービスを巡る開発が活発である。Web サービスでは、従来 WWW ブラウザを介してしか利用出来なかった各種インターネット上のサービスが XML により容易にプログラム処理可能になる事から、これによりエージェントが操作可能なネットワークサービスが多数出現し、エージェントを実用で利用できる環境が整うと考えられる。

エージェントが可能な代行処理は、例えばユーザに代わってオークションへの自動入札を行うようなサービス利用の自動化、ユーザの作業内容に応じて関連情報を表示するようなサービスの利用支援、鉄道やホテルの予約サービスを利用して旅行予約サービスを提供するようなサービス統合、SCM など企業間サービス連携など様々なものが考えられる (図 1)。さらに利用するサービスや利用場面、処理の目的、利用するユーザなどが変わると処理の内容も変わる事から、可能な処理は極めて大量・多種多様になると考えられる。

しかし、実用では多くの処理の実行には高い信頼性が求められる。例えば、商品の購入要求を行ったエージェントがその結果をユーザに通知する前にマシン障害で消失してしまえば大きなトラブルに繋がる。このような代行処理を行うシステムの実装は、個別には従来のプログラム言語などでも実現可能であるが、数多くの処理を構築し実行・管理するのは非常に困難である。

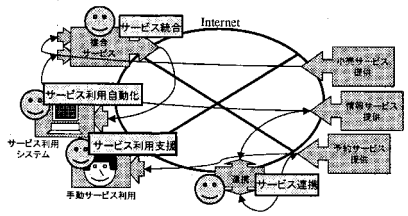

図 1: エージェントによるサービス処理

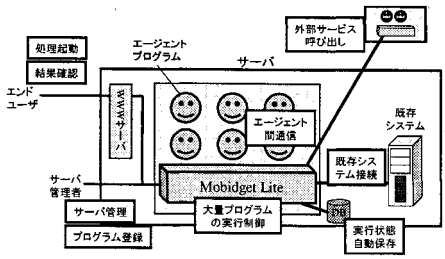

図 2: MobidgetLite 概要

3　エージェントサーバ MobidgetLite

我々は、上記のような多数・多種多様な処理を高信頼で実行するためのプログラム実行基盤として、エージェントサーバ MobidgetLite を作成した。MobidgetLite はユーザによって記述されたエージェントプログラムを多数実行管理するための機構を提供し、サービスの自動化・支援を行うシステムの構築を容易にする。

MobidgetLite はサーバ上で動作するエージェントプログラムの実行基盤である (図 2)。エージェントはユーザの何らかの処理を代行するためにサーバ上で動作する任意のプログラムであり、サーバ内外の他のシステム・サービスの呼び出しやエージェント間通信などを行い動作する。エージェントのプログラムはアトミックな小さなタスクの遷移として記述する。エンドユーザはアプリケーション毎の GUI を介してエージェントと対話し、処理の依頼や結果の確認を行う。またこれとは別にサーバ管理者は管理 I/F を介してエージェントプログラムやユーザの登録管理、実行中のプログラムの管理等を行う。MobidgetLite はこれら大量のプログラムの実行制御を行うとともに、実行状態を自動的に DB に保存し、サーバダウンなどの障害後のエージェントの復旧を可能にする。また、エージェントプログラム用の通信など

---

* MobidgetLite: A large scale agent server

† Kazuya Koyama, Nobuyuki Tomizawa, Satoru Fujita, Toru Yamanouchi

‡ NEC Corporation

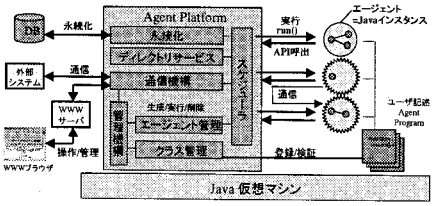

図 3: MobidgetLite 構成

の API や、サーバ管理者用の管理 GUI などを提供し、エージェントプログラムの作成や実行管理を容易にする。

MobidgetLite の構成は図3の通りである。MobidgetLite は Java 上で動作する。エージェントは独自のスレッドを持たない一つの Java インスタンスとして表現され、プログラムはそのメソッドとして記述する。AgentPlatform と呼ぶ実行管理機構が多数のエージェントインスンタンスを管理し、そのメソッドを繰り返し呼び出す事でエージェントの処理を実行する。さらに、AgentPlatform は下記の機能を提供している。

- エージェント管理: 生成、削除、一覧、検索
- プログラム管理: クラスの登録 / 削除、コード検証
- スケジューリング: 実行、一時停止、タイマ設定、イベント通知
- 永続化: エージェントの状態を定期的に保存
- メモリ管理: エージェトの swap-out/in
- 通信: Agent 間 / 外部システムとの非同期メッセージ通信
- 運用管理: Web 画面によるエージェント、プログラム、ユーザ、サーバなどの管理

永続化では、1 回のエージェントのメソッド呼出しをアトミックなタスクとし、1 呼出し毎にエージェントと AgentPlatform の状態を DB に保存し、サーバ障害時の保存状態からのシステム復旧を可能にする。また外部システムと高信頼メッセージによって通信する場合、エージェント状態保存と通信を 1 トランザクションで扱い矛盾の発生を回避する。さらにメモリ管理として、DB に保存後に休止中のエージェントをメモリ上から削除し、必要時に該当エージェントのみ DB から復旧する事で、少ないメモリでの大量エージェントの処理を可能にする。

図 4はエージェントプログラムの記述例である。プログラムには通信や自身の実行管理などの処理を記述できる。エージェントの実行状態はインスタンス変数としてプログラムが明示的に保存する。

4 性能評価

大量のエージェントを生成した場合の実行性能、特に DB への状態保存を主とした実行オーバーヘッドの測定実験を行った。実験は空の処理を行うエージェントを n 個生成し、そのうち 10 個だけを active エージェントとして連続実行した場合の合計実行時間を測定した。ま

```
public class HelloAgent extends Agent {
  int counter; //エージェントとして保存される状態

  public void init() { //生成時に一度だけ実行
    counter = 0;
  }

  public void run() { //定期的に実行
    Message recvMsg = receive(); //メッセージ受信
    AgentName sender = m.getSender();
    switch (counter) {
      case 0:
        SampleContent c = new SampleContent("Hello Agent.");
        Message sendMsg = new Message(c, null);
        send(sender, sendMsg); //メッセージ送信
        break;
      case 1:
        ....
      case 10:
        exit(); //エージェント終了
    }
    counter++;
    sleep(0); //エージェント一時停止
  }

  public void destroy() { //削除時に一度だけ実行
  }
}
```

図 4: エージェント記述例

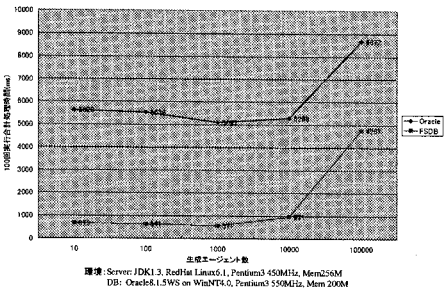

図 5: 性能評価実験結果

たエージェントの保存先として Oracle DB と、独自のファイル形式 (FSDB) の 2 通りを測定した。

結果は図 5の通りで、生成エージェント数 1 万程度までは変化なく、１０万程度になると性能劣化が見られた。また、1 エージェントの 1 メソッド実行あたりの実行時間はおよそ Oracle で 5ms、FSDB で 0.6ms 前後であった。

5 考察

本稿ではエージェントサーバ MobidgetLite の概略について述べた。MobidgetLite を用いる事で大規模・高信頼なエージェントシステムが容易に実現できる。

同様なシステムとしては Caribbean [1] があるが、Caribbean は状態監視などサーバ内部に閉じた処理を目的としているのに対して、MobidgetLite は他システムのサービスを利用・連携する事を目的としており、永続化機構が高信頼メッセージによる他システムとの通信と連動するなどの機能を備える点が異なる。

現状の MobidgetLite ではプログラムが自身で実行状態を変数に保存するように記述する必要があるが、今後は MobidgetLite 上で状態遷移図など別の記述形式をサポートすることで実行状態の扱いを簡易化し、プログラムの記述をより容易化していきたい。

参考文献

[1] G.Yamamoto,H.Tai: “Architecture of an agent server capable of hosting tens of thousands of agents”, Proc. of the fourth international conference on Autonomous agents.