成蹊大学理工学研究報告
J. Fac. Sci.Tech., Seikei Univ.
Vol.53 No.1 (2016)pp.29-34

AgentSphere へのセキュリティの導入と並列分散処理向けファイルシステムの試作

蓮見 建太[*1], 甲斐 宗徳[*2]

Introduction of A Security Mechanism into AgentSphere and Prototype Design of A File System for Parallel Distributed Processing System

Kenta HASUMI[*1], Munenori KAI[*2]

ABSTRACT : The authors of this study are using mobile agents with capability of strong migration to develop AgentSphere, a platform for an autonomous parallel and distributed processing system that significantly reduces loads of the complex execution control for its users. In AgentSpheres on a network, agents can migrate autonomously among PCs and execute their own processes. In this paper, we introduce two kind of securities for AgentSphere, one for the protection of agents and another for the protection of the resource of each PC from the attacks by agents sent onto it. In addition, we reconsider the file access mechanism on AgentSphere and implement a new file system in order to access files on any AgentSphere from any AgentSphere safely and transparently.

Keywords : Parallel and distributed systems, mobile agent, strong migration, autonomous system

(Received June 30, 2016)

1. 序論

1. 1 研究背景

近年、PCの性能が向上することによって解決可能な問題が増えつつあるが、解決困難な計算量を持つ問題は依然多く存在している。このような問題を解決するために更なるハードウェア面の性能向上を行う方法もあるが、その場合には物理的限界や高いコストがかかってしまう問題が存在する。そこで、このような問題へのソフトウェア面からの解決方法として並列分散処理が存在する。

並列分散処理は、ネットワーク上の複数台のPCに処理を分割して与え、並列的に実行することで問題解決の高速化を図るものである。しかし、このようなシステムの構築を行うためには、特別な知識や技術、経験が必要となる。そこで、特別な知識や技術を必要とせずに並列分散処理を利用するための分散処理環境が求められている。

*1：理工学研究科理工学専攻博士前期課程学生
*2：理工学研究家理工学専攻教授 (kai@st.seikei.ac.jp)

1. 2 研究目的

本研究は、特別な知識や技術を必要とせず、簡単な操作やコーディングで並列分散処理を行うことができるシステムのプラットフォームにAgentSphereと名付け、開発を進めている。

AgentSphereでは、エージェントと呼ばれるプログラムが各PC間を移動しながら処理を行う。即ち、外部のPCに自らのエージェントを預けたり、外部のPCから送信されてくるエージェントを受け入れて自らのPC上で実行させたりといったことが頻繁に発生する。こうした一連の流れの中で、エージェントの持つデータに対する盗聴・改竄や、送信されてきたエージェントから自分のPC上のファイルやリソースに対する攻撃的な操作などが考えられる。既存のモバイルエージェントシステムにおいても、こうしたセキュリティ問題への対策は必須となっている。

また、AgentSphere上で実行されるエージェントは通常のJavaで書かれたプログラムと同様のファイルストリームを利用できるが、そのエージェントが実行中のPCの持

つローカルファイルシステムにしかアクセスすることはできなかった。それどころか、そのPCの持つファイルシステムへ自由なアクセスが可能であった。そこで、従来のファイルアクセス機能の制限を行い、その代替となり、自律的に移動するエージェントの挙動に適したファイルアクセス機能が必要となっている。

本研究では、昨年度までセキュリティ対策の存在しなかったAgentSphereに、基本的なセキュリティを導入し、ユーザができる限り安心して利用できる態勢を整える。同時に、エージェントのファイルアクセス方法の見直しを行い、いずれのPC上に移動した後でもユーザがローカルファイルにアクセスするかのように透過的に利用できることを目的としたファイルシステムの試作を行う。

1. 3 AgentSphereの概要

我々は分散処理を実現するためのプラットフォームとして、モバイルエージェントシステムを採用した。モバイルエージェントシステムとは、実行中のプログラムがネットワークで繋がれた各PC上を自律的に移動し、移動した先のマシンで処理を継続することのできるシステムである。このプログラムを本研究ではエージェントと呼ぶ。エージェントが自身を実行中のPCにかかっている負荷状況を読み取り、負荷が高ければより負荷の低いPCへ移動することを自律的に行うことで、負荷分散を実現している。

AgentSphereはマルチプラットフォームを目指しており、プラットフォームに依存しない言語としてJavaを採用している。また、エージェントの移動性には移動後に移動前と同じ位置から再開するために強マイグレーションを採用している。

2. AgentSphereに要求されるセキュリティ

2. 1 エージェントの機密性

モバイルエージェントシステムでは、ネットワーク上の他者のPCに自らの作成したエージェントを送り込む性質上、そのエージェントの持つ情報が安全であるように配慮しなくてはならない。

AgentSphereと同様にJavaを利用したモバイルエージェントシステムは多く存在し、Javaの仕様上不正なメモリアクセスが発生せず、実行中のエージェントの持つprivateな情報を読み取ることはできない。

一方で、Javaのオブジェクトをシリアル化して送信する場合には盗聴・改竄の可能性が高まる。シリアル化オブジェクトは機密性の高くないデータなので、暗号化を施すことが望ましい。

AgentTCLでは、マシン間で送信されたエージェントやメッセージを暗号化する機能を用意し、エージェントのプライバシーを維持している。

AgentSphereにおいても必要に応じてエージェントの持つ変数やメッセージの暗号化が可能になっている。

これにより、現状でも十分にエージェントの機密性は保たれていると考えられる。

2. 2 不正なエージェントからの攻撃に対するAgentSphereの保護

2. 2. 1 エージェントの権限

AgentSphereのエージェントは通常のJavaで記述されたスレッドと同様であり、制約は一切存在しなかった。その場合、エージェントを受け入れたAgentSphereを実行しているPCの持つシステムプロパティやファイルシステムへのアクセスが容易に可能である。この状況では、ユーザは安心してエージェントを受け入れることができない。

他のモバイルエージェントシステムにおける対応策の一例として、Voyager[1]ではJavaのSecurityManagerを拡張し、エージェントの実行可能な操作を制限している。また、Aglets[2]では自らが生成したAgletsと外部から送り込まれたAgletsを区別し、それぞれに対してファイルや通信ポートの利用許可を設定することができる。

今回、本研究ではJavaのSecurityManagerを利用することによってエージェントの権限操作を導入した。Javaアプリケーション上でSecurityManagerを有効にすると各クラスのアクションを制御できる。この機能によって制御可能なアクションは、システムプロパティへのアクセスやファイル操作、セキュリティ設定の変更などがある。

Javaアプリケーションに与える権限はPolicyクラスによって定義することで、明確に設定することができる。権限の設定は適用する範囲を指定して行う。AgentSphere上で実行するエージェントの所在はnullとして扱われる。AgentSphere本体に権限を与えたとしても送られてきたエージェントには権限が与えられない。こうして、外部から送られてきたエージェントからのみシステムプロパティへのアクセスやファイル操作の権限を失わせることができ、不正なエージェントからのAgentSphereへの攻撃を防ぐことができる。

しかし、エージェントから権限を奪うということは、エージェントに記述できる処理の自由度が損なわれてしまうということになる。エージェントは自身に記述され

たファイル操作命令だけでなく、AgentSphere本体の機能を通じたファイル操作も行うことができなくなる。この問題の解決のために、特権ブロックを利用することができる。特権ブロック内のアクションは、呼び出し元のエージェントではなくその特権ブロックの書かれたAgentSphere本体の権限として実行されることになる。よって、認証情報や生成元AgentSphere、グループなどの情報を元にして制御できるメソッドを通じてアクションを実行することで、エージェント自身は自由度を失わずにセキュリティを保つことができる。

現在、制限された機能の中でAgentSphereのエージェントにとって重要となる機能は、ファイルへのアクセス機能である。

2. 2. 2　エージェントの受け入れ制限

モバイルエージェントシステムは他者に計算リソースを公開し、他者のエージェントを実行させるものである。外部から目的も詳細も不明なエージェントが送信されてきたとしても、そのエージェントを無条件で受け入れることは難しい。したがって、外部から送られてきたエージェントの持ち主が誰であり、信用できるプログラムであるかどうかを認証する必要がある。

このための対策として、AgentTCLではマシン間で送信されたエージェントやメッセージを電子署名化する機能を備えており、JampingBeansでは公開鍵、秘密鍵と証明書の利用によって認証を行っている。

AgentSphereで扱うエージェントは、現状、ネットワーク上の全てのAgentSphereへ自由に移動することができる。通常はエージェントの自律性に任せて移動先を決定するが、必要であればユーザが特別に移動先を指定して移動させることもできる。この時、移動先のAgentSphereは送られてくるエージェントを拒否することはできない。そのため、信頼できるエージェントのみを受け入れるようにしなくてはならない。

また、例えば図1のようにAgentSphereネットワーク内に小さなグループAがあったとすれば、一時的にグループAの計算リソースをグループAのAgentSphereで生成されたエージェントのみで使用したいといった場合も考えられる。このような動的な受け入れ制限にも対応する必要がある。そこで、送信されてくるエージェントが受け入れ可能であるかどうかを認証する機能を実装した。

まず、お互いのAgentSphereに共通鍵を用意する。ここで、エージェントを作成した側をPC1 とし、エージェントを受け入れる側をPC2 とする。共通鍵はAgentSphere外の手段で共有し、それぞれ自らのAgentSphereに登録する。PC1 が共通鍵を登録した状態でエージェントを作成すると、そのエージェントのエージェントIDを共通鍵によって暗号化する。このエージェントが移動する際にPC2 に暗号化されたエージェントIDを送信する。PC2 は送られてきた暗号化されたエージェントIDの複合化を試みて、複合化に成功すればお互いに共通の鍵を持っている既知の相手として認証し、エージェントを受け入れる。

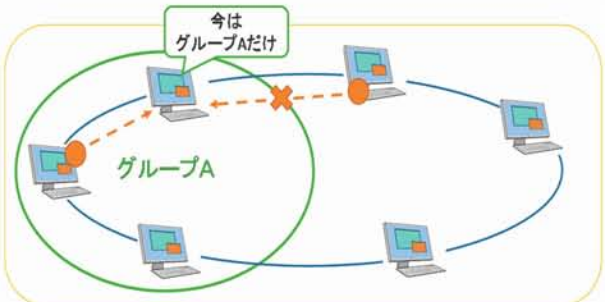

図1　AgentSphereネットワーク上のグループ

2. 3　導入したセキュリティの動作実験

2. 3. 1　エージェントの権限

エージェントの権限設定を確認するため、外部から送られてきたエージェントがファイル操作を行おうとした時に、その操作が制限されるかどうかを動作実験した。SecurityManagerを実行し、送られてきたエージェント上でローカルファイルの操作を行おうとするとエラーが出力され操作できなかった。一方で、AgentSphere本体はファイルの入出力を実行することができた。これにより、エージェントにのみ正しくアクションの制限がかけられていることが確認できた。

2. 3. 2　エージェントの受け入れ制限

エージェントの受け入れ制限が正しく行われているか、また受け入れられなかった時に移動失敗の判定をエージェントが認識し、エージェントの処理の継続が正しく行えるかを確認した。図2 に示すように3 台のAgentSphere

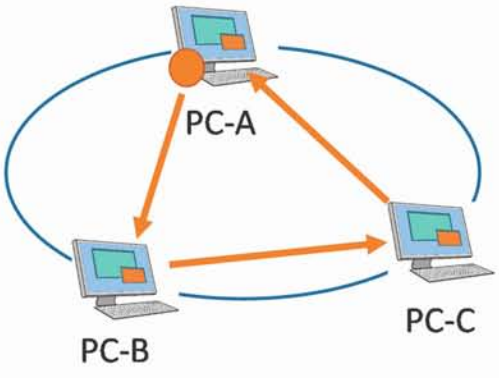

図2　実験に用いたエージェントの移動の様子

（PC-A、PC-B、PC-C）を順に10週移動するエージェントを用意し、PC-Cの受け入れ拒否の有無によってそれぞれ動作実験を行った。

移動先のAgentSphereが受け入れ不可の場合には、移動できなかったことを受けて次の移動を行う。そして、最後にそれぞれのAgentSphereがエージェントを受け入れた回数を表示した。

受け入れ制限をかけない場合には、各PCに10回ずつ移動を行った。PC-Cに受け入れ制限をかけて他のエージェントを受け入れないように設定すると、PC-AとPC-Bは同様に10回ずつ受け入れたが、PC-Cは1回も受け入れなかった。PC-Cの受け入れ制限が正しく機能していたと同時に、移動に失敗したことを認識し処理を継続できていたことがわかった。

2. 3. 3 まとめ

エージェントの権限操作の導入により、AgentSphereで実行されるエージェントは不正なアクションを容易に行うことができなくなった。また、自身のAgentSphereが受け入れるエージェントを制限することが可能となり、信頼できるエージェントのみを受け入れる態勢が整ったといえる。

3．AgentSphere向けファイルシステムの試作

3. 1 AgentSphere向けファイルシステムの目的と条件

AgentSphereは並列分散処理の煩わしい制御を省くことが目的であるため、このファイルシステム自体もエージェントの状態によらない利用が可能でなければならない。そこで、今回作成するファイルシステムは、以下の目的と条件を持つ。

- エージェントがどのAgentSphereに移動したとしても、同様に同じファイルにアクセスすることができる。
- ユーザにとって透過的に利用でき、エージェントの位置や状況によって外見上は特別な操作を必要としない。
- エージェントの権限設定によって利用できなくなったファイルアクセス機能の代替として、AgentSphere内部で管理されたファイルアクセス機能を提供する。

この条件を満たすため、本研究ではファイル共有プロトコルCIFSを用いてファイルシステムの試作を行った。

3. 2 ファイル共有プロトコルCIFSについて

CIFS(Common Internet File System)は、マイクロソフト社の開発したWindowsOSのファイル共有で使用されるSMBプロトコルを拡張したものである。CIFSはTCP/IPを用いてファイル共有を行う。LinuxやMacOSなどのUNIX系OSでも、Sambaを通じて利用することができる。

3. 3 ファイル共有プロトコルCIFSを用いた共有ファイルシステムの実装

JavaでCIFSを利用できるJCIFSライブラリ[3]が公開されている。このライブラリを利用することで、Windowsファイル共有にアクセスすることができる。しかし、2.2.1項で導入したエージェントの権限設定によってエージェントはファイルシステムへのアクセスが制限されている。そこで、JCIFSを利用するための中継となるクラスを用意した。このクラスをインスタンス化し、JCIFSを利用するために必要な情報を与えておく。必要な情報はユーザ認証情報とファイルサーバのIPアドレス、カレントディレクトリの絶対パス、ファイルの相対パスの4つである。

通常JCIFSを利用する場合は、ユーザ認証情報を本体のプロパティに登録しておくことができるが、AgentSphereのエージェントは別AgentSphereに移動してしまうためこの情報を維持することができない。そこで、このクラスでユーザ認証情報の管理を行う。

カレントディレクトリの絶対パスはローカルファイルにアクセスする場合は必要とならないが、リモートのファイルシステムにアクセスする場合には必要になる。これをこのクラスに持たせておき、自動的に補うこととした。もしエージェントが生成元PCの持つローカルなファイルにアクセスしたい場合は、Java標準のファイルアクセス機能を用いてアクセスする。この時カレントディレクトリやIPアドレスを自動的に記憶しておき、エージェントの移動後はユーザが特別に設定しなくても補えるようにすることで透過的なファイルアクセスを行えるようにした。

3. 4 共有ファイルシステムの動作実験

3. 4. 1 ユーザ情報が正しい場合

PC-Aのローカルファイルで作業を行った後、PC-Bに移動して同様にしてファイルシステムからのアクセスが可能であるか確認した。エージェント内ではローカルアクセスと同等であることを示すためにユーザ認証情報と相対パス（test.txt）のみを指定し、カレントディレクトリとIPアドレスは自動的に取得させた。

エージェントはPC-A上でディレクトリ内のファイルのリストの表示とtest.txtへの出力を行い、PC-Bに移動した後、同じtest.txtの内容を読み込み表示した。PC-A上で

の結果を図 3 に、PC-B上での結果を図 4 に示す。同じアドレス上の同じファイルにアクセスできたことが確認できた。

図３　PC-A上での結果

図４　PC-B上での結果

3．4．2　ユーザ情報が誤りの場合

共有ファイル側の共有設定に含まれないユーザ名とパスワードではアクセスができない。別ユーザ情報を登録して 3.4.1 項と同様の操作を行った。

PC-A上では 3.4.1 項と同様の結果が得られた。これは、生成元のローカルファイルであるためローカルアクセスとしてユーザ情報を用いなかったためである。

一方で、PC-B上では図 5 のように表示された。接続先は正しいが、アクセス権限がない場合のエラーが発生した。

図５　ユーザ情報が誤りの場合の結果

５．結論

本研究によって、AgentSphereはエージェントの挙動の制限と、エージェントの受け入れの制限が可能になった。これにより、不正なエージェントからの攻撃を防ぐことが可能になり、また信頼できる相手や特別に指定した相手のエージェントのみを受け入れることができるようになった。

また、CIFSを用いたファイルシステムの実装によって、エージェントはどのAgentSphereに移動したとしても同様にして同じファイルにアクセスすることが可能となった。この機能を制限されたファイルアクセス機能の代替とし、受け入れた外部のエージェントがPCのファイルに不正にアクセスすることはなくなった。

今後の課題として、不正なエージェントが過剰に計算リソースを消費しようとするケースへの対応が必要である。AgentSphereは例えPCをダウンさせられたとしてもバックアップ機能によって素早い復旧が可能となっている。しかし、そもそものPCのダウンを防ぐべきであると考える。また、こういった悪意のあるエージェントに攻撃されるケースをできる限り少なくすることで、ユーザはより安心してAgentSphereのネットワークを利用することができる。

参考文献

[1] ObjectSpace Voyager 3.1. http://www.inf.fu-berlin.de/lehre/WS99/VS/Misc/Voyager/API/.

[2] Aglets. http://aglets.sourceforge.net/.

[3] JCIFS. https://jcifs.samba.org/.