2007－OS－104 (10)
2007／1／31

XML Web サービスのための分散型オペレーティング・システム

大木 薫[†] 新城 靖[†‡] 佐藤 聡[†] 板野 肯三[†‡] 馬渕 充啓[†]

[†]筑波大学システム情報工学研究科コンピュータサイエンス専攻
[‡]科学技術振興機構, CREST

要旨

この論文では、XML Web サービスのための分散 OS(Operating System)について述べる。本分散 OS は Web サービスのためのファイル、パイプ、シェル、およびコンソールを提供する。本分散 OS の特徴は、アプリケーションとして動作する Web サービスのサーバが本分散 OS の機能を用いて直接連携できることにある。本分散 OS では、オブジェクトへのアクセス制御の仕組みとしてケーパビリティを用いている。また、利用者はコンソールを通じて Web サービスのサーバを対話的に利用することができる。

A Distributed Operating System for XML Web Services

Kaoru Oki[†] Yasushi Shinjo[†‡] Satoshi Sato[†]

Kozo Itano[†‡] Mitsuhiro Mabuchi[†]

[†]Department of Computer Science, University of Tsukuba
[‡]CREST, Japan Science and Technology Agency

Abstract

This paper describes a distributed operating system for XML Web Services. This distributed operating system provides files, pipes, a shell and a console for web services. By using these functions, web services can interact. This operating system uses access control based on capabilities. By using the console, users can use server of web services interactively.

1. はじめに

XML Web サービスは、Web サービス(Web of services)ともよばれ、XML 形式のデータ交換を行うことでソフトウェア・コンポーネントをネットワークを通じて利用可能にするものである[5]。Web サービスの利点としては、使用する言語、およびプラットフォームから独立していることがあげられる。現在公開されている代表的な Web サービスとしては、Amazon Web Services[2]、Google SOAP Search API[6] および、Yahoo! Search Developer Network[15]などがある。また、既存のサービスを組み合わせ、新しい Web サービスを提供するマッシュアップと言う考え方が広まりつつある。

従来の Web サービスの利用形態では、主に 1 つのクライアントが 1 度に 1 つのサーバを利用している。マッシュアップを行う場合でも、新たなサーバが元になるサーバのクライアントとなり、複数の元になるサーバを逐次的に呼び出している。今後は複数のサーバが直接連携することが重要になっていくと思われる。

本論文では、XML Web サービスのための分散

OS(Operating System)について述べる。本分散OSはWebサービスのためのファイルとパイプを提供する。また、利用者との対話、およびスクリプトの実行を行うシェルを提供する。これにより、Webサービスのサーバの連携が容易になる。本分散OSアクセス制御のためにケーパビリティを用いている。これにより、インターネットのような開かれた環境においてオブジェクトの保護を行える。

2. 関連研究

2.1 Webサービス

Webサービスとは、XML形式のメッセージを用いたRPCによりオブジェクトにアクセスができるようにする技術である。Webサービスで利用される代表的なプロトコルとしてはSOAPがある。SOAPは下位プロトコルにHTTPなどを使用している。

この論文ではサービスを提供するプログラムをサーバ、利用するプログラムをクライアントと呼ぶことにする。サーバはWSDL(Web Service Description Language)という XML 形式で記述されたインタフェース情報を公開する[14]。クライアントはこのWSDL記述を得ることで、スタブを生成し、サーバを利用する。

複数のWebサービスのサーバを連携させるための仕様として、コレオグラフィ記述言語がある。コレオグラフィ記述言語は複数のWebサービスのワークフローを記述し、連携させるためのXMLに基づく言語である。コレオグラフィの仕様として代表的なものに WSBPEL(Web Services Business Process Execution Language)、WSCI(Web Services Choreography Interface)がある[9][10]。コレオグラフィ記述言語を用いることで、複数のサーバの連携を定義できるが、実行方法はクライアントがそれぞれ1度に1つのサーバを利用する形である。本分散OSでは、ファイルやパイプを用いることで、サーバがクライアントを介さず直接連携できる。

2.2 LANで動作する分散OS

関連研究として、LAN(Local Area Network)で接続されたコンピュータを対象とした分散OSについて述べる。Machはカーネギーメロン大学で開発された分散OSである[1]。Machではポートへのアクセスをカーネルにより保護されたケーパビリティで制御している。Amoebaは1980年後半から1990年前半にかけて開発された分散OSである[8]。Amoebaでは、クライアントがプログラムを実行する際、遠隔のオブジェクトを利用するためにケーパビリティを用いている。このケーパビリティは、利用者プロセスによって、直接操作されるため、偽造や改ざんを防ぐための乱数を含んでいる。また、利用者プロセスはケーパビリティから操作が限定された弱いケーパビリティを生成することができる。

これらの分散OSはLAN内の閉じた環境を対象としている。本分散OSはインターネットなど、開かれた環境を対象とする。

3. Webサービス連携のための分散OSに対する要件要求

集中型OSにおけるアプリケーションの連携と比較しながら、Webサービスにおけるアプリケーションを連携させるための分散OSに必要とされる機能を明らかにする。

3.1 集中型のOSにおけるアプリケーションの連携

集中型OSでは、アプリケーションは他のアプリケーションとファイルを介して連携することができる。この場合、ファイルを名前で指定することもできる。また、UNIXのパイプのようにファイルと互換性のあるプロセス間通信を用いて、他のアプリケーションの処理の結果を直接利用することもできる。

現状のWebサービスで、複数のサーバを連携させる場合、クライアントが、個々のサーバを呼び出す処理を記述しなければならない。現在、Webサービスでは集中型OSのファイルのような、全てのサービスで利用可能な基盤は存在しない。したがって従来のWebサービスのクライアントは、特定のWebサービスのサーバだけを利用するか、非標準的なファイルを利用することしかできない。また、ファイルをファイル名のような参照でやりとりできず、データは全て値でやりとりをしている。

集中型OSでは、プロセスファイルシステムのようにいくつかのサービスをファイルと言う一般的なインタフェースで提供している。文献[4][13]で述べられているシステムでは、ディレクトリを通じて、ファイルの内容を検索する機能が利用できる。この場合、検索のための特別なアプリケーションを使うことなく、ls 等のファイルをブラウズするコマンドにより検索を行うことができる。

RPCはもともと分散システムを構築するための技術であるが、集中型OSにおいても、RPCは活用されている。そのようなRPCの例として、ToolTalk[12]、Windows COM[7]があげられる。そのようなRPCに対するアクセス制御はインターネットのような開かれた環境では、そのまま利用できない。たとえば、ToolTalkが利用しているSunRPCでは、Kerberosを利用した利用者認証機能を利用できるが、クライアントとサーバで同じKerberosサーバを信頼する必

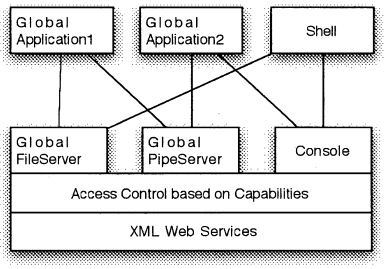

図 1:本分散 OS の構成

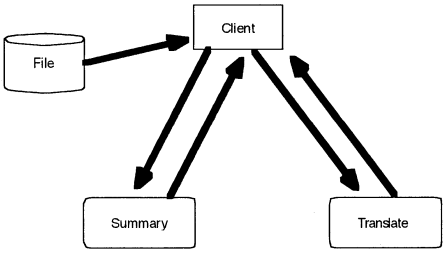

図 2-a:従来の方法での Web サービスの連携

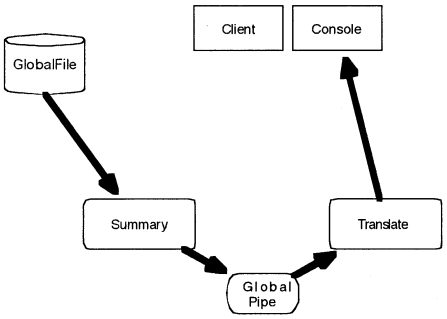

図 2-b:本分散 OS による Web サービスの連携

要がある。インターネットのような開かれた環境ではそのような単一のサーバを信頼するような仕組みを利用することはできない。

また現在、Web サービスにおけるサーバの利用はバッチ的な利用が一般的である。すなわちクライアントは全ての引数をあらかじめ用意してサーバを呼び出す。サーバが必要に応じて利用者の入力を受け取るような、対話的な利用はできない。

3.2 分散 OS に対する要求要件

3.1 節で述べた問題を解決するために、次のような仕組みが必要である。

(1) 集中型 OS におけるファイルと同等の仕組み
- 複数アプリケーションで標準的に利用できる。
- 参照によるデータの受け渡しが可能である。
- パイプのようなファイルと互換性のある通信手段を提供する。

(2) サービスの起動方法の標準的な仕組み。

(3) インターネットのような開かれた環境でも利用可能なアクセス制御の仕組み。

(4) 利用者と対話を行う仕組み。

本研究では、このような要件を満たす XML Web サービスのための分散 OS を設計し、実装する。

4. XML Web サービスを連携させるための分散 OS

本分散 OS の構成を図 1 に示す。本分散 OS は以下の要素から構成される。

(1) グローバル・ファイルのサーバ : Web サービスのサーバであり、ファイル、およびディレクトリのサービスを提供する。

(2) グローバル・パイプのサーバ : Web サービスのサーバであり、ファイルと互換性のあるパイプのサービスを提供する。

(3) シェル : 利用者との対話、およびスクリプトの実行を行う。

(4) コンソール : キーボード、および画面を通じてユーザとの文字による入出力を行う。

本分散 OS では、オブジェクトの保護にはケーパビリティを用いている。本分散 OS では統一されたインタフェースを持った Web サービスのサーバを提供する。このような Web サービスのサーバを**グローバル・アプリケーション**と呼ぶ。

図 2 に 2 つのアプリケーションが連携している様子を示す。クライアントは Summary を利用してファイルの要約を行い、その結果に対して Translate で翻訳を行っている。

図 2-a は従来の方法、および図 2-b は本分散 OS を用いた Web サービスの連携の様子をデータの流れに着目して示したものである。これらの図において矢印はデータの流れを表す。従来の方法では、クライアント自身がファイルからデータを取り出し、データを引数にしてサーバを呼び出し、結果を受け取っている。本分散 OS では、クライアントはファイルやパイプへの参照(ケーパビリティ)をアプリケーションに渡しているだけであり、アプリケーションの呼び出しと、最終的なデータの取得のみを行っている。

以下の節では本分散 OS の要素のうち、シェルと除いたものについて述べる。これらはサーバとして動

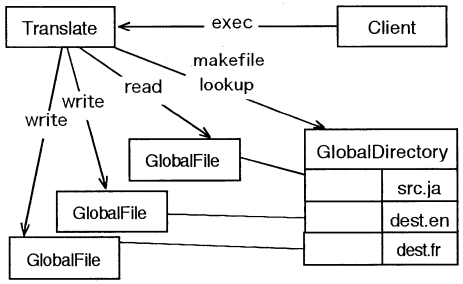

図 3:グローバル・ファイルとグローバル・ディレクトリ

作する。シェルについては 5 章で述べる。

4.1 ケーパビリティによるアクセス制御

本分散 OS ではオブジェクトへのアクセス制御としてケーパビリティに基づくものを用いる。ケーパビリティは次のような要素を持つ。

(1) オブジェクトへの参照
(2) オブジェクトに対するアクセス権
(3) 偽造や改ざんを防ぐための乱数や電子署名

Mach のように、カーネルでケーパビリティを保護すれば(3)の要素は必要ない。

分散 OS ではケーパビリティとして、WSDL による記述を用いている。WSDL は、RPC おけるインタフェース記述言語と同様に、そのオブジェクトのインタフェースを記述する。それに加えて WSDL による記述は、サーバやオブジェクトの位置を示す情報を含んでいるため、それだけでオブジェクトの手続きを呼び出し、オブジェクトを操作することが可能である。したがって WSDL による記述はオブジェクトへの参照として働く。本分散 OS では、予測を不可能にするために、WSDL の記述のオブジェクトの位置を示す URL に乱数を含ませている[11]。アクセス制御にケーパビリティを用いることで、グローバル・アプリケーションの間でオブジェクトを参照の形でやりとりすることが可能になる。

また、本分散 OS では権限の強いケーパビリティから権限の弱いケーパビリティを生成することを可能にする。たとえば、ファイルへの書き込み、および読み出しの両方を行う権限を持つケーパビリティから、読み出しのみを行う権限を持つケーパビリティを生成できるようにする。弱いケーパビリティを用いて、潜在的に悪意を持つ可能性のある遠隔の Web サービスのサーバのアクセス権を制限することができる。そして、万一攻撃を受けたとしても被害の拡大を小さくすることができる。

4.2 グローバル・ファイルとグローバル・ディレクトリ

本分散 OS はグローバル・アプリケーションが利用できるファイル、およびディレクトリを提供する。そのようなファイル、およびディレクトリをそれぞれ**グローバル・ファイル**、および**グローバル・ディレクトリ**と呼ぶ。グローバル・ファイルのインタフェース GlobalFile は、Java の File クラスのオブジェクトを参考にして設計した。GlobalFile は、ファイルへのストリームに基づく入力端、および出力端を生成するメソッドを持つ。グローバル・ディレクトリのインタフェース GlobalDirectory は、ファイル、ディレクトリ、および 4.4.1 項で述べるコマンド型のグローバル・アプリケーションの名前と、それらのケーパビリティのマッピングを行う。グローバル・ディレクトリは、新たなディレクトリの作成、ファイルの作成、およびリンクの作成などのメソッドを持つ。

グローバル・ファイル、およびグローバル・ディレクトリの動作を図 3 に示す。図 3 においてグローバル・アプリケーション Translate はあるファイルを読み込んで複数カ国語に翻訳し、それぞれの結果を同じディレクトリに保存している。クライアントは日本語のファイルを含むディレクトリのケーパビリティ、および、Translate を利用するためのケーパビリティを持っている。クライアントは、グローバル・ディレクトリのケーパビリティ、およびファイルの名前 src.ja を引数として、Translate を実行する。Translate は、オブジェクト名に対応するケーパビリティを得るメソッドである lookup メソッドを用いて、src.ja のケーパビリティを得、データを入力する。その後、グローバル・ディレクトリ対して makefile メソッドを実行し、dest.en、および dest.fr を作成し、結果を出力している。

グローバル・ファイルには、ストリームに基づく入出力を行うためのインタフェースがある。これらをそれぞれ**グローバル・インプット・ストリーム**、および**グローバル・アウトプット・ストリーム**と呼ぶ。これらのインタフェースはそれぞれ Java の InputStream、および OutputStream を元に作成した。それぞれ GlobalInputStream、および GlobalOutputStream と呼ぶ。これらのインタフェースは Java の同名のインタフェースと同じメソッドを持つ。ただし、read メソッドの一部は Java では引数に値を書き込む方法と取っており、メモリを共有していない Web サービスではそのまま利用できない。そのため、本分散 OS のストリームと Java のストリームの間には完全な互換性はない。互換性が必要な場合には本分散 OS が提供しているラッパを

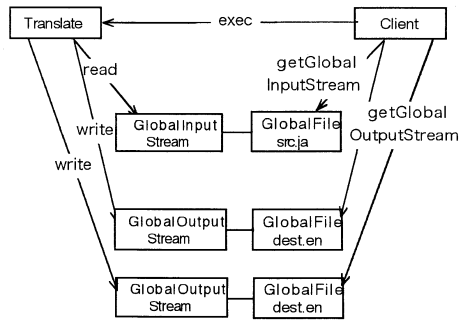

図 4:GlobalInputStream と GlobalOutputStream による アクセスの制限

用いることができる。

GlobalFile のケーパビリティは、読み出しおよび書き込みの権限を持つ強いケーパビリティと考えることができる。クライアントは GlobalInputStream または GlobalOutputStream のケーパビリティのみをサーバに渡すことで、読み出しのみ、または書き込みのみの権限を持つ弱いケーパビリティを生成して、サーバに渡すことに相当する。

図 3 では、Translate は親のグローバル・ディレクトリ、および、それに含まれている全てのファイルに対して任意のメソッドを実行することができる。Translate が悪意を持っているグローバル・アプリケーションであった場合、これら全てのオブジェクトが破壊されてしまう。この問題を解決するために GlobalInputStream、および GlobalOutputStream を利用することができる。

図 4 では、クライアントは Translate を実行する前にあらかじめ dest.en および、dest.fr のファイルを作成し、src.ja を含めた 3 つのグローバル・ファイルのケーパビリティを得る。次にそれぞれの getGlobalInputStream メソッド、または getGlobalOutputStream メソッドを実行し、入力専用の GlobalInputStream、または出力専用の GlobalOutputStream のケーパビリティを得ている。そしてこれらのケーパビリティを引数として Translate を実行している。その結果、Translate は src.ja の読み出しのみ、dest.en、および dest.fr の書き込みのみのアクセス権を持つ。

4.3 グローバル・パイプ

本分散 OS はファイル入出力と互換性のあるプロセス間通信を行う機能として、**グローバル・パイプ**を提供する。グローバル・パイプは、UNIX のパイプと類似の機能を Web サービスのサーバとして提供する。グローバル・パイプのインタフェース

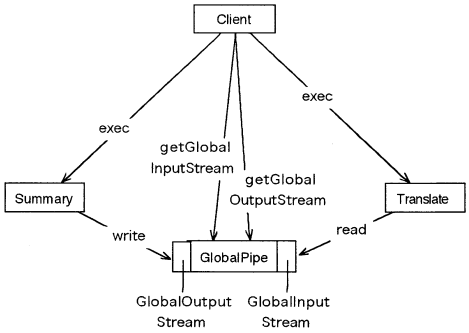

図 5:グローバル・パイプの利用

GlobalPipe は入力端、および出力端として、それぞれインタフェース GlobalInputStream、および GlobalOutputStream のオブジェクトを生成するメソッドを持つ。グローバル・パイプは内部にバッファを持つ。

グローバル・パイプを用いたグローバル・アプリケーション間の通信について、図 5 を用いて述べる。図 5 においてクライアントはグローバル・アプリケーション Summary の処理の結果をグローバル・アプリケーション Translate で翻訳している。クライアントはグローバル・パイプにアクセスし、パイプの入力端と出力端である、GlobalInputStream、および GlobalOutputStream のオブジェクトを作成し、そのケーパビリティを得る。その後、それぞれのケーパビリティを引数としてグローバル・アプリケーションを実行している。Summary および、Translate は、ファイルに対する入出力と同じ方法でパイプを利用する。GlobalInputStream、および、GlobalOutputStream のオブジェクトはグローバル・パイプ内のバッファを介して通信を行う。

グローバル・パイプを用いることで、クライアントが仲介することなく、グローバル・アプリケーション間でデータのやり取りが行えるようになる。また、ファイルと同じインタフェースを用いることで、グローバル・アプリケーションはファイル入出力と同じ方法でデータを送受信することができる。

4.4 分散 OS で利用されるアプリケーション

本分散 OS で動作するグローバル・アプリケーションは、以下の 2 つのいずれかに分類される。

(1) コマンド型

(2) ファイル型

以下でこれらのグローバル・アプリケーションについて述べる。

4.4.1 コマンド型グローバル・アプリケーション

コマンド型グローバル・アプリケーションは、メ

引数：

```
{stdin=inputCapability,
 stdout=outputCapability,
 arg1="jp-en",
 config=configFileCapability}
```

結果

```
{status="success"}
```

図 6:グローバルアプリケーションのインタフェースの例(Translate)

ソッド exec を持つ。このメソッドは、連想配列を引数に取り、別の連想配列を結果として返す。この連想配列には、文字列、または、ケーパビリティを含めることができる。たとえば、図 6 に翻訳を行うグローバル・アプリケーションの引数と結果の連想配列の構成を示す。図 6 において stdin に入力ファイルのケーパビリティ、stdout に出力ファイルのケーパビリティを与えている。arg1 はフィルタ固有の引数である。config に設定ファイルのケーパビリティを与えている。結果には成功や失敗を示す値を持つ status の要素がある。このようなインタフェースを用いることにより、統一された方法でグローバル・アプリケーションを実行することが可能になる。

コマンド型のグローバル・アプリケーションは主に 5 章で述べるシェルにより利用される。

4.4.2 ファイル型グローバル・アプリケーション

ファイル型グローバル・アプリケーションはインタフェース GlobalFile、および GlobalDirectory を通じて機能を提供する。これは集中型の OS におけるプロセスファイルシステムと類似の機能である。このような方法は、固定されたキーワードによる Web ページの検索エンジンや天気予報の提供などのサービスを利用する場合に有用である。

4.4.3 ラッパ

本分散 OS で動作するグローバル・アプリケーションには 4.4.1 項、または 4.4.2 項で示すようなインタフェースを通じて機能を提供する必要がある。このようなインタフェースを持たない従来の Web サービスのサーバは、ラッパを用いることで本分散 OS のアプリケーションとして利用する。

本研究では Google SOAP Search API を対象としてファイル型のラッパを実装した。Google SOAP Search API は Google の Web 検索の機能を Web サービスとして提供する。このラッパは、検索に必要なキーワード等のパラメタを受け取り、検索の結果をディレクトリ、およびファイルの形で返す。ディレクトリこのラッパにより提供されるファイルは、read メソッドが呼ばれると Google SOAP Search API の doSearch メソッドを実行し結果を返す。

4.5 コンソール

本分散 OS では、利用者のキーボード入力、およびアプリケーションからの画面出力を行うための機能として、コンソールを提供する。コンソールはインタフェース GlobalInputStream および、GlobalOutputStream を通して利用できる。利用者が入力したデータは、コンソールのバッファに格納され GlobalInputStream のインタフェースから読み出すことができる。またグローバル・アプリケーションは GlobalOutputStream のインタフェースを通じて利用者の画面に出力できる。

本分散 OS では、グローバル・アプリケーションはコンソールを用いて利用者と対話することができる。従来の Web サービスのサーバはクライアントからの引数を受け取るだけであり、このように利用者と対話することはできない。

5. シェル

シェルは利用者と対話することで、ファイルシステムを操作する機能、およびグローバル・アプリケーションを実行する機能を提供する。

5.1 文法

この節では、シェルの文法について述べる。シェルのコマンドは、基本的には以下の形式で与えられる。

シェル変数名 = コマンド名　引数

本シェルは、UNIX のシェルと同様にシェル変数を持つ。シェル変数は$で始まる文字列で表現される。シェル変数は文字列型、または連想配列型のデータを保持することができる。文字列型は、通常の文字列以外にケーパビリティを保持するためにも用いる。連想配列の要素を参照するには演算子"."を用いる。たとえば連想配列$a の中のキーkey1 で指定される要素は$a.key1 と表現される。シェル変数への代入は演算子"="を用いて表現する。シェル変数は代入を行うことで作成する。変数への代入が行われなければ、シェルは評価した値を表示する。演算子"@"は、名前からファイル、またはディレクトリのケーパビリティを得るものである。ことができる。演算子"@>"、および"@<"を用いることでそれぞれファイルへの入力、および出力のケーパビリティを得ることができる。また、UNIX のシェルと同様に入力行の末尾に"&"を付けることで、バックグラウンドでの実行を行うことができる。

入力において、"[]"で囲まれた式はその時点の値

```
1 >$ws = makews {input=$stdin}
2 $p = pipe
3 $s=summary {input=$args.input, output=$p.input, config=[@~/lib/summary.conf]}&
4 $t=translate {input=$p.output, output=$args.output, config=[@~/lib/trans.conf]}
5 $result.sstatus=$s.status
6 $result.status=$t.status
7 end
8 >link { dir=@~/bin, name=sumtrans, capability=$ws.capability}
9 >$result=sumtrans {input=@./src.ja out=$stdout}
```

図7:シェルにおけるグローバル・アプリケーションの作成と利用

として評価される。この機能は後述する makews コマンドにおいてケーパビリティを埋め込むために用いる。

5.2 コマンド

コマンドには、内部コマンドと外部コマンドがある。外部コマンドは、あるグローバル・ディレクトリに登録されたコマンド型のグローバル・アプリケーションの名前である。シェルの内部コマンドのうち、代表的なものを以下にしめす。

・pipe
引数：{}
結果：{*input,output,status*}
パイプを作成し、入力端、および出力端を返す内部コマンドである。status に成否が格納される。

・makews
引数：{*input*}
結果：{*capability,status*}
複数のコマンド列をまとめて実行するグローバル・アプリケーションを生成する内部コマンドである。引数 input にコマンド列のソースを読み込むための GlobalInputStream のケーパビリティを与える。結果として capability 要素には生成されたグローバル・アプリケーションのケーパビリティが格納される。

・link
引数：{*dir,name,capability*}
結果：{*status*}
インタフェース GlobalDirectory の link メソッドを用いてグローバル・ディレクトリにエントリを追加するメソッドである。dir は対象となるディレクトリのケーパビリティを与える。name、および capability にそれぞれオブジェクトの名前、およびケーパビリティを与える。

5.3 動作例

シェルが 2 つのグローバル・アプリケーション Summary、および Translation を実行している様子

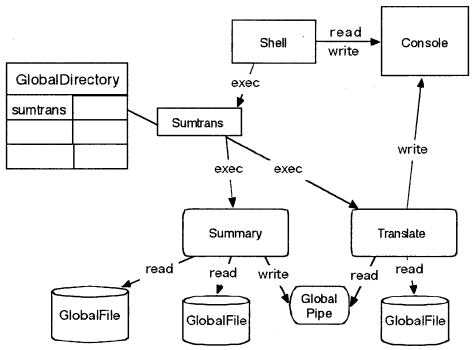

図8:シェルの動作

を図7、および図8に示す。グローバル・アプリケーション Summary は引数として、入力元のケーパビリティ、出力先のケーパビリティ、および設定ファイルのケーパビリティを取る。グローバル・アプリケーション Translate は引数として、入力元のケーパビリティ、出力先のケーパビリティ、設定ファイルのケーパビリティ、および翻訳のタイプを与える。行頭の">"はシェルのプロンプトである。1 行目で makews コマンドで一連の操作を行うグローバル・アプリケーションをキーボード入力で定義している。定義中は通常のプロンプト">"は表示されない。2行目でパイプを作成している。3行目、および4行目で、グローバル・アプリケーションを実行している。演算子"@"を含む部分はその場で評価されて返り値であるケーパビリティが埋め込まれている。5行目、6行目でそれぞれのグローバル・アプリケーションの成否の結果を得ている。$args、および$result はそれぞれ定義中のグローバル・アプリケーションの引数、および結果を表している。8行目で新しく定義したグローバル・アプリケーションを外部コマンドのディレクトリである~/bin に"sumtrans"と言うコマンド名で登録している。9行目で入力のケーパビ

リティ、および画面出力を引数として sumtrans を実行している。

6. 実装

本分散 OS で提供する Web サービスのエンジンには Apache Axis(以下 Axis)を用いた[3]。また、Axis が動作するアプリケーションサーバとして、Apache Tomcat を用いた。現在ファイル、パイプ、および一部のグローバル・アプリケーションが動作しており、シェルを実装中している。

本分散 OS ではファイル、ディレクトリ、およびパイプなどの一つ一つのオブジェクトが Web サービスとして公開(deploy)され、ケーパビリティを用いて管理している。そのため、多数オブジェクトを Web サービスのサーバとして動的に公開する。Axis には公開できるオブジェクト数に限界があるという問題点がある。この問題を解決するために、仮想記憶と同様に要求により公開する機能を Axis に導入した。Axis には、公開しているオブジェクトの名前とオブジェクトの対応をとるハッシュ表がある。このハッシュ表に含まれていなければ通常、オブジェクトは存在しないものとして扱われる。本研究では Axis の URL を解析する部分に手を加えて、要求されたオブジェクト名がハッシュ表に登録されていないときに、ハッシュ表にオブジェクトを登録する機能を追加した。オブジェクトの情報をデータベースに保存しておき、要求されたオブジェクトがデータベースに登録されているオブジェクトであった時にはその場でハッシュ表に登録する。長い時間使われないエントリはハッシュ表から削除する。

7. まとめ

本論文では XML Web サービスのための分散 OS について述べた。この分散 OS は Web サービスのためのファイルとパイプを提供する。また、本分散 OS は利用者との対話を行うシェルを提供する。アクセス制御のためにケーパビリティを用いている。

今後の課題は、分散 OS を利用するためのユーティリティプログラムを作成することである。また、スタッカブルオブジェクトの生成による弱いケーパビリティの生成を行う機能、およびグローバル・アプリケーション間の同期を行うための機能を提供することを考えている。

参考文献

[1] Accetta, M., R. Baron, W. Bolosky, D.Golub, R.Rashid, A.Tevanian and M. Young : "Mach : A New Kernel Foundation for UNIX Development",Proceedings of USENIX Summer Conference, pp. 93-112(1986).

[2] Amazon : Amazon Web Services(2006). http://developer.amazonwebservices.com/

[3] The Apache Web Services Project: Axis Reference Guide Version 1.2 (2005). http://ws.apache.org/axis/java/reference.html

[4] David K. Gifford, Pierre Jouvelot, Mark A, James W. O'Toole, jr.: Semantic File Systems. In Proceeding of thirteenth ACM symposium on Operating system principles, pp. 16-25 (1991)

[5] Glass, G.: Web Services: Building Blocks for Distributed Systems, Prentice-Hall (2001)

[6] Google : Google SOAP Search API (2005). http://code.google.com/apis.html

[7] Microsoft Corporation: Inside COM+ Base Services (1999).

[8] Mullender, S. J., van Rossum, G., Tanenbaum, A. S., van Renesse, R. and van Staveren, H.: Amoeba: A Distributed Operating System for the 1990s, IEEE Computer, Vol. 23, No.5, pp. 44-53 (1990).

[9] OASIS: Web Services Business Process Execution Language Version 2.0 (2006).

[10] The World Wide Web Consortium: Web Service Choreography Interface (WSCI) 1.0 (2002). http://www.w3.org/TR/wsci/

[11] 新城 靖, 阿部 聡, 板野 肯三: "XML Web サービスのための大域的ファイル・サービスの提案", 情報処理学会研究会報告 2004-05-96-03, pp.15-22 (2004).

[12] SunSoft: The ToolTalk Service: An Interoperability Solution (1993).

[13] 滝田 裕, 多田 好克:"全文検索エンジンを利用したファイルしシステムの名前空間拡張", 情報処理学会論文誌, Vol.47, No. SIG3, pp. 16-26(2006).

[14] The World Wide Web Consortium: Web Services Description Language (WSDL) 1.1 (2001).

[15] Yahoo: Yahoo! Developer Network (2006). http://devloper.yahoo.net/