ハイパフォーマンス
コンピューティング 91−33
（２００２．８．２３）

動的なアプリケーション開発実行を可能にするグリッドポータルアーキテクチャ

鈴村　豊太郎[†2,†4] 中田　秀基[†2,†1]
松岡　聡[†2,†3] 関口　智嗣[†1]

近年，グリッドポータルはグリッドに対する高位のインタフェースとして重要な役割を担っている．我々は，グリッドポータルの開発を支援する為のツールとして，ポータルに必要となるユーザインタフェースの自動化とバックエンドのグリッドアプリケーションの開発を支援するツール Ninf Portal の開発を進めている．本稿では，グリッドポータルにおける動的なアプリケーションの開発実行環境のアーキテクチャの提案を行なう．具体的には，Ninf-G のクライアント API に Java バインディングを実装し，そのバインディングを基盤に，スクリプト言語 Python のインタフェースを実装した．次に，ポータル上でそのスクリプトインタフェースを用いてグリッドアプリケーションを記述し，ページのフォームにユーザインタフェースの情報を入力することによって，動的にユーザに特化したアプリケーションの生成を支援する環境のアーキテクチャの設計を行なった．

Dynamic Application Developement and Execution Environment for Grid Portals

Toyotaro Suzumura,[†2] Hidemoto Nakada,[†2,†1]
Satoshi Matsuoka[†2] and Satoshi Sekiguchi[†2]

As the Grid proliferates as the next-generation computing infrastructure, a user interface in the form of "Grid Portals" is becoming increasingly important, especially for computational scientists and engineers. Although several Grid Portal toolkits have been proposed, the portal developer still must build and deploy both the user interface and the application, which results in considerable programming efforts. We aim to ease this burden by generating the portal frontend (that constitutes of JSP and Java Servlets) from a XML document for the former, and a GridRPC system, Ninf-G for easily "gridifying" existing applications for the latter, and realizing their seamless integration. The resulting system, which we call the Ninf Portal, allowed concise description and easy deployment of a real Grid application with greatly small programming efforts. This paper describes the off-the-self architecture which automatically generates an application portal by developing a Grid application by the use of an scripting language in an interactive way and giving user interface information on the web page. This allows portal users to utilize a large variety of applications including default applications defined by portal administrators as well as user-defined applications generated by this architecture.

1. はじめに

近年，グリッド技術が成熟するにつれて，より高位のインタフェースである Grid ポータルと呼ばれる機構の必要性が認識されつつある．Grid ポータルとは，特別なソフトウェアサポートを持たないクライアントから Grid 上のアプリケーションを使用することを可能にするシステムである．特別なソフトウェアを使用しないという要請から，インタフェースとしては Web ブラウザを用い，プロトコルには http (もしくは https) を用いるのが一般的である．ユーザは，Web ブラウザでポータルにログインし，Grid アプリケーションと使用資源，使用データを指定して実行する．Grid アプリケーションの実行状態の監視と制御や，使用データや設定ファイルのアップロードや結果のダウンロードもできる．グリッドポータルは，通常の Web アプリケーションの機能に加えて，シングルサインオンや資源の検索など Grid 特有の機能を実装しなければならない．これまでに，これらの機能を構築する為のツールキットが，HotPage や GPDK 等で提案されている．しかし，これらのシステムでは，Grid アプリケーションのユーザインタフェースとなるデータ入力のページの記述や，実際に起動される Grid アプリケーションの記

†1 産業技術総合研究所
National Institute of Industorial Science and Technology(AIST)
†2 東京工業大学 Tokyo Institute of Technology
†3 国立情報学研究所 National Institute of Informatics
†4 日本学術振興会特別研究員
Japan Society for the Promotion of Science

述に関しては，サポートしていない．我々は，これまでにこれらの機能も提供する統合的なグリッドポータル構築ツールキット Ninf Portal[5)3)] を提案した．Ninf Portal は，グリッドポータルのフロントエンドの構築を XML ベースのユーザインタフェース生成でサポートし，バックエンドの構築を Grid RPC システムである Ninf-G でサポートする．

現在の Ninf Portal では，ポータルの管理者が事前に提供したアプリケーションのみがポータル上で使用可能である．しかし，使用可能なアプリケーションが限定されてしまい，すべてのユーザの要求を満たすポータルを作成することは管理上困難である．よって，ポータル上でユーザが動的にグリッドアプリケーションとそのユーザインタフェースの作成できる環境が必要である．

本稿では，グリッドポータルにおける動的なアプリケーションの開発実行環境のアーキテクチャの提案を行なう．具体的には，Ninf-G のクライアント API に Java バインディングを実装し，そのバインディングを基盤に，スクリプト言語 Python のインタフェースを実装した．次に，ポータル上でそのスクリプトインタフェースを用いてグリッドアプリケーションを記述し，ページのフォームにユーザインタフェースの情報を入力することによって，動的にアプリケーションポータルの生成を支援する環境のアーキテクチャの設計を行なった．

本稿の構成は次のとおりである．2 章では，Ninf Portal の概要を示す．3 章では Grid RPC システム Ninf-G の Java API の実装について述べる．4 章では、動的なアプリケーション開発実行環境のアーキテクチャを述べ，最後にまとめを述べる．

2. Ninf Portal の概要

Ninf Portal[5)3)] は，グリッドポータルの構築ツールキットである．Ninf Portal では，既存のポータル構築ツールキットの機能に加え，ユーザインタフェース部と Grid アプリケーション部の記述をサポートする．ユーザインタフェースに関しては，Grid Application IDL と呼ばれる XML ベースの Grid アプリケーションの記述言語で書かれたインタフェース情報から自動的に JSP ファイルを生成し，ユーザインタフェースページを作成する．Grid アプリケーションに関しては，Grid RPC システム Ninf-G を提供することで，記述を容易にする．

Ninf Portal のアーキテクチャを図 1 に示す．ユーザから入力されたデータを処理するコンポーネントは，汎用データ処理サーブレットが行う．このサーブレットは，ユーザインタフェースの JSP プログラムがセッションに保存した入力情報のメタデータを取り出し，その情報に基づいて，サブミットされてきたデータを解釈し，Ninf-G で作成されたバックエンドプログラムを起動する．Grid アプリケーションの実行時には，ログイン時に MyProxy から取得したユーザの証明書を使用する．現在，データ処理サーブレットは，証明書をテンポラリディレクトリに書き出して，環境変数に証明書ファイルのパスを設定してから Grid アプリケーションを起動しているが，これはセキュリティ上好ましくない．この問題は，次章で述べる Ninf-G の Java API を用いて Grid アプリケーションを開発することによって解決できる．Java ベースの アプリケーションであると，サーブレットが直接アプリケーションに代理証明書を渡すことができる．

Ninf Portal のバックエンド部となる Ninf-G では，Globus の認証機構である GSI (Grid Security Infrastructure) を用いて認証を行う．GSI は，ユーザの証明書によって署名された証明書 (代理証明書) がユーザと同じアイデンティティを持つとみなす，証明書の委譲と呼ばれる機構によってシングルサインオンを実現している．Ninf Portal では，MyProxy を用いて，グリッドポータルからのシングルサインオンを実現している．MyProxy は，代理証明書のリポジトリとなるサーバである．これを用いると，代理証明書を安全な第三者のサーバに預けておき，他のホストからそれをユーザ名とパスフレーズで取り出すことができる．

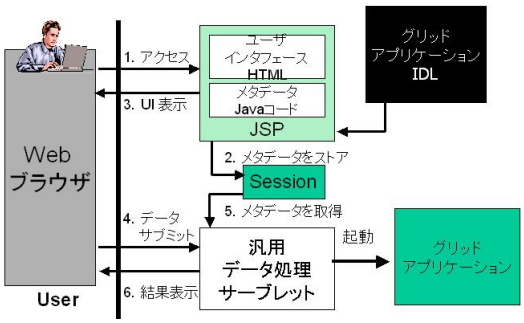

図 1 Ninf Portal のアーキテクチャ

3. Ninf-G の Java API の実装

本章では，Ninf-G の Java API について述べる．Java API を提供することによって，Ninf Portal の汎用データ処理サーブレットは，外部コマンドとして，Grid アプリケーションを実行させるのではなく，Java で書かれた Grid アプリケーションのインスタンスを生成するだけでよい．また，ユーザの代理証明書も外部ファイルに書き出す必要がなくなり，サーブレットは，アプリケーションのインスタンス時に，予め取得した代理証明書を渡せばよい．4.1 章では，この Java バインディングを用いて，Ninf-G のスクリプト言語のインタフェースを実装する．はじめに Ninf-G の概要を述べた後，Java API の実装の詳細及びサンプルコードを述べる．

3.1 Ninf-G の概要

Ninf-G[6] は，旧電総研が中心となって開発した Grid RPC システム Ninf[2] を Globus Toolkit を用いて再実装したものである．Ninf/Ninf-G の基本的な特徴を以下に示す．

- クライアント・サーバ型の計算システムであり，サーバに存在する計算ルーチンをクライアント側のプログラムから容易に実行できる．
- サーバ側での IDL 記述が容易である．基本的に C における関数のインタフェース定義に入出力モードを追加したものである。IDL 記述を IDL コンパイラで処理することで，make ファイルとスタブメインが得られ、make ファイルを実行することで，実行バイナリが得られる．

実装に用いられている Globus Toolkit は，グリッドソフトウエアに必要とされる資源管理機構やユーザ認証システム，通信ライブラリなどの要素技術を実装する為の API やコマンドを提供する低レベルのツールキットであり，グリッドのソフトウェアインフラストラクチャを構成する要素の事実上の標準になりつつある．我々が以前に開発した Ninf システムでは、サーバやインタフェースの取得などは、独自のコンポーネントを作成していたが，Ninf-G では，それらのコンポーネントを Globus の各種コンポーネントに置換することで実装している．

3.2 Java API の実装

実装は、Java CoG Kit (Commodity Grid Kits)[4] を用いて実装した．CoG は、Globus の各種サービスに対する Pure Java の ライブラリである．Ninf-G の Java API の実装では，CoG を用いて，MDS から関数インタフェース情報を取得し，GRAM へリモートライブラリの起動要求を行う．また，CoG は，GSI による通信のライブラリも含まれており，このライブラリを用いて Globus の各コンポーネント及び リモートライブラリへの安全な通信を実現する．

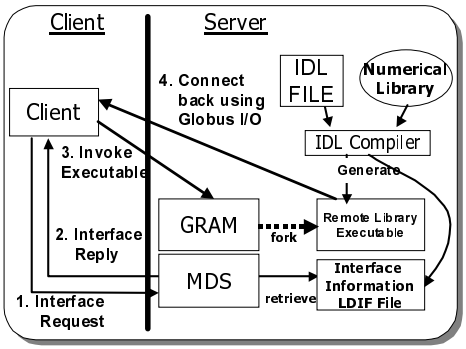

図 2 Ninf-G のリモートライブリの実行プロセス

3.2.1 リモートライブラリの実行

Ninf-G のリモートライブラリ実行は図 3.2 に示す過程で行われる．

(1) クライアントが MDS に引数情報とパス情報をリクエスト
(2) 引数情報とパス情報を取得
(3) パス情報を用いて GRAM にリモートライブラリの起動をリクエスト
(4) リモートライブラリからクライアントに Globus I/O を用いて接続

3.2.2 引数情報とパス情報の取得

Ninf-G ではリモートライブラリのパスと引数情報は MDS に登録されている．Ninf-G の Java クライアントは, CoG の MDS ライブラリを用いて，これらの情報の検索、取得を行う．MDS ライブラリは， JNDI (Java Naming and Directory Interface) をベースに実装されており，MDS のサーバへは GSI を用いてアクセスする．MDS から得られた検索結果はクライアント側でキャッシュされ，可能な限り再利用される．これによって比較的大きい MDS 検索のコストを低減している．

3.2.3 リモートライブラリの起動

リモートライブラリの起動は CoG の GRAM ライブラリを用いて行う．この際に，MDS から取得したリモートライブラリのパス情報を使用する．クライアント側では，GassServer, HandlerManager, Observer の 3 つのクラスのオブジェクトが生成され，それぞれリモートライブラリ及び GRAM からのコールバックを受ける．各クラスの役割を以下に示す．

- GassServer: リモートライブラリが標準出力，標準エラーに出力したメッセージがクライアント側に転送される
- HandlerManager: 実際にリモートライブラリと通信する．XML ベースのコマンド、入出力パラメータの送受信を行う
- Observer: リモートライブラリのモニタリングを行う．サーバ側で何らかの異常が合った時にリモートライブラリから通知を受ける

上記のオブジェクトが待機するポート番号は，基本的に動的に空いているポートを用いるが，クライアントの設定ファイルで静的に指定することもできる。また，使用可能なポート番号の範囲を制限しているサイトなどのために、ポートの割り当て範囲を指定することも可能である．リモートライブラリの起動を GRAM に要求する際には，引数として、HandlerManager と Observer のオブジェクトのポート番号を指定する．

3.2.4 クライアントとリモートライブラリの通信

GRAM により リモートライブラリが起動されると，引数からクライアントのアドレスとポートを取得する．次に，リモートライブラリは，Globus I/O を用いてこのアドレスとポートに接続し，互いに通信を開始する．この際，クライアント側でオープンされているポートは認証と認可に制約を受け，自分自身の証明書を持っているプログラム以外からは接続できない

ようになっている．従って，コールバックを装った第三者が接続してしまうことはない．

3.3 Java API 及びサンプルコード

次に，Java の API と図 4 のサンプルコードを用いてアプリケーションの記述の仕方について述べる．

サンプルコード中で注意すべき行は try から catch までの文である．まず，GrpcClient クラスのオブジェクトを生成し，activate メソッドを用いてクライアントアプリケーションの初期化を行なう．activate メソッドには，クライアントの設定ファイルを文字列として渡す．この設定ファイルには，使用する MDS のホスト名，ポート番号などの情報を書く．

次に，GrpcClient クラスの getHandler メソッドで、リモートライブラリが存在するホスト名と名前を指定し，GrpcHandler クラスのオブジェクトを取得する．GrpcHandler クラスは，ある特定のサーバ上のライブラリへのコネクションを抽象化したクラスで，一度起動したリモートライブラリのコネクションを使い回すことができる．このクラスの call 又は，callWith メソッドを用いてリモートライブラリにジョブの要求を行なう．

図 3 に GrpcHandler が実装するインタフェース GrpcInterface を示す．call メソッドは，引数に入出力パラメータの List のオブジェクトを受け取り，リモートライブラリにジョブ要求を送る．返り値の GrpcExecInfo クラスは，計算を実行したホストの情報、入出力パラメータの転送時間、計算の実行時間等を表すインタフェースである．Java では，C のように可変個の引数をとるメソッドをを定義することはできないので，この他に，callWith メソッドを用意した．これは，ユーザが明示的に List インタフェースを実装した Vector クラスを作らなければならず、煩雑である為である．現在，引数の数が 12 個までのメソッドを定義している．また，Java では，基本型 (int, double) はオブジェクトではないので，ラッパークラス (Integer, Double) に変換して，メソッドに引数を代入する．call 及び callWith メソッドはすべてブロッキングの呼び出しである．C の API とは異なり，ノンブロッキングの呼び出しを用意しなかったのは，Java が言語レベルでスレッドをサポートしており，マルチスレッドを用いてノンブロッキングの機構をユーザが容易に実現できる為である．

GrpcHandler のメソッド destract はハンドラを終了するメソッドである．また，GrpcClient の deactivate メソッドを用いてアプリケーションを終了する．

4. 動的なアプリケーション開発実行環境の設計

本章では，ポータル上で動的にアプリケーション及びそのユーザインタフェースの開発ができる環境の設計を行う．

```
public interface GrpcInterface {
 public void destract() throws GrpcException;
 public GrpcExecInfo call(List args)
                          throws GrpcException;
 public GrpcExecInfo callWith(Object a1)
                          throws GrpcException;
 public GrpcExecInfo callWith(Object a1,
                      Object a2)
                        throws GrpcException;
 ....
}
```

図 3 ハンドラが実装するインタフェース

```
import java.io.*;
import java.util.Vector;
import org.apgrid.grpc.client.GrpcException;
import org.apgrid.grpc.client.GrpcHandler;
import org.apgrid.grpc.client.GrpcClient;
import org.apgrid.grpc.client.GrpcExecInfo;

public class Test{
 public static void main(String args[]){
  GrpcHandler handler;
  GrpcExecInfo execInfo;

  String configFile = "./config.cl";
  double[] a, b;
  int i, j, k, result;
  int arraySize = 5;
  int ittr      = 2;

  a = new double[arraySize];
  b = new double[arraySize];

  for (i = 0; i < arraySize; i++){
    a[i] = (double) (i * 1.1);
    b[i] = 0.0;
  }

  try {
    GrpcClient client = new GrpcClient();
    client.activate(configFile);
    handler = client.getHandler("test/plus");
    for(i = 0; i < ittr; i++){
       execInfo =
         handler.callWith(
          new Integer(arraySize),a, b);
     }
    }
   handler.destract();
   client.deactivate();
  } catch (GrpcException e){
    e.printStackTrace();
    System.exit(1);
  }
 }
}
```

図 4 Ninf-G の Java API を用いたサンプルコード

4.1 スクリプト言語によるアプリケーション開発

まず，ユーザがポータル上でインタラクティブにアプリケーション開発できるように，Ninf-G のクライアントインタフェースに スクリプト言語 Python[1] を提供する．

Python は，オブジェクト指向型のスクリプト言語である．現在，実行処理系には，C 言語で書かれた CPython と，Java で書かれた Jython がある．Jython は 実行時にソースコードから Java のバイト

```
import sys
from java.util import Vector
from java.lang import String, Integer
from org.apgrid.grpc.client
  import GrpcClient, GrpcHandler, GrpcExecInfo
from jarray import zeros,array

N=5
input = zeros(N, 'd')
output = zeros(N, 'd')
for i in range(0, N):
 input[i] = i * 1.1

client = GrpcClient()
client.activate("config.cl")
handler = client.getHandler("test/plus")
execInfo = handler.callWith(Integer(N),
                            input, output)
handler.destract()
client.deactivate()

for i in range(0, N):
  print output[i]
```

図 5 Jython を用いた Ninf-G のサンプルコード

コードを生成し，Java の仮想マシン (JVM) 上でそのバイトコードを実行する．また，JVM 上で実行する為，Java の標準のライブラリや，ユーザが作成したライブラリを Python のスクリプトコードに組み込むことができる．一方，Java のプログラムからも容易に Jython のインタプリタを起動することができる．

Python の利点としては，Java 言語で記述するよりも記述量が少ないこと，コンパイルする必要がなくポータル上でもインタラクティブなプログラム開発ができること，などが挙げられる．また，Ninf Portal では，処理系に Jython を用いる．これによって，前章で実装した Ninf-G の Java API を利用することができる．

図 5 に，前章で記述した Java のサンプルコードを Python を用いて記述した例を示す．

4.2 アプリケーションポータル作成ページ

ユーザは，図 6 のアプリケーションポータル☆ 作成ページを用いて，アプリケーション及びユーザインタフェースの記述を行う，ユーザが記述すべき情報は以下の通りである．

(1) アプリケーションのメタ情報 (名前，説明，作成者，アクセス権等)
(2) アプリケーションのパラメータ情報 (名前，型，説明)
(3) アプリケーションのスクリプトコード

メタ情報中のアクセス権は，定義したアプリケーションポータルに対してどのユーザへ使用許可を与えるかを定める．(1) すべてのユーザに公開する，(2) あるグループに所属するユーザのみ公開する，(3) 公開しない，の 3 つのオプションを用意する．これによって，例えば，物理の研究者のコミュニティで，あるユーザが作成したアプリケーションポータルを共有するといったことが可能になる．

☆ 本稿では，ある特定のアプリケーションに特化したポータルをアプリケーションポータルと呼ぶ

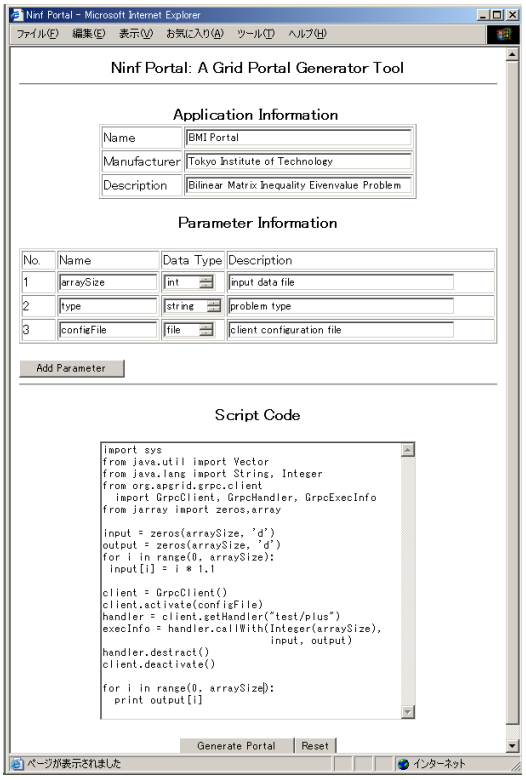

図 6 アプリケーションポータル作成ページ

パラメータは，ユーザがフォームで入力データを与え，アプリケーションに渡すものである．パラメータの型 (int, double, string, inputfile 等) は，プルダウン型のメニューを用いて選択する．パラメータの名前は，次のスクリプトコードで使用する．

アプリケーションのスクリプトコードは，4.1 章の Ninf-G の Python インタフェースを用いて記述する．コード中でユーザがポータル中から入力するパラメータにアクセスするには，前述したパラメータの名前を用いる．図 6 の例では，一番目で定義したパラメータ arraySize をスクリプトコード中で用いている．

ユーザがこれらの情報をすべて入力した後，ページ下の”Generate Portal” ボタンを押すことによって，ユーザが定義したアプリケーションポータルが作成される．スクリプトコードでシンタックスエラーなどがある場合には，エラーを発生した個所が表示される．

4.3 アーキテクチャ

図 7 に，前章で示した仕組みのアーキテクチャを述べる．

4.3.1 ユーザインタフェースの生成

アプリケーションポータル作成ページでユーザが入力した情報を受け取り，アプリケーションポータルを生成するコンポーネントが，図 7 中の Portal Genera-

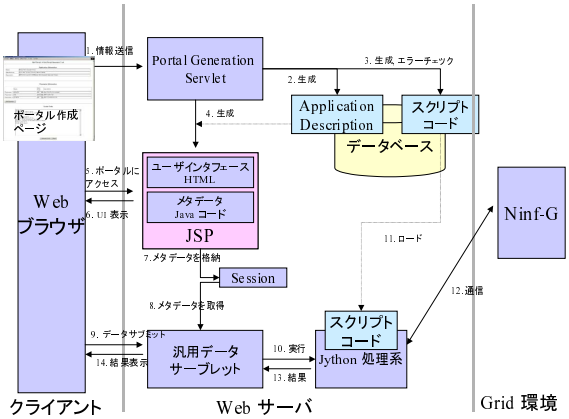

図 7 アーキテクチャ

tor Servlet (PGS) である．PGS は以下の２つのファイルを生成し，それらをデータベースに格納する。

(1) スクリプトコード

(2) アプリケーション情報ファイル (Application Description File)
スクリプトコードの場所，アプリケーションのメタ情報，パラメータ情報を基に，Grid Application IDL を用いて記述した XML の文書

PGS は，スクリプトコードのシンタックスを検査し，エラーがある場合には，エラーページを生成する．次に，PGS は，Application Description のファイルを用いてユーザインタフェースである JSP ファイルを生成する．この JSP ファイルの生成の工程は，現在の Ninf Portal の仕組みと同様である．

4.3.2 アプリケーションの実行

ユーザは，4.3.1 で作成された JSP ページにアクセスし，ユーザが定義したアプリケーションをポータル上で使用することができる．ユーザは，フォームに必要なデータを入力した後，その情報がデータ処理サーブレットに送られる．

データ処理サーブレットは，メタデータから取得したアプリケーションの種類を拡張子で判断する．アプリケーションが Python で記述されたコードで，拡張子が .py で終わる場合，スクリプトをデータベースからロードし, Jython のインタプリタ上を起動する．Jython/Python の処理系では，外部のプログラムから，任意のパラメータの値を予め設定して実行することができる．よって，ユーザがフォームに入力する値をメタデータを用いて適切なパラメータに設定した後，スクリプトコードを実行する．

実行後，通常のアプリケーションと同じく，実行結果をウインドウに表示するか、サーバ上のファイルへのリンクとして URL を表示する．

4.3.3 セキュリティ

このアーキテクチャでは，任意のスクリプトコードがサーバ上で実行される．よって，セキュリティについて十分に考慮する必要があるが，スクリプトコードは Java の仮想マシン (JVM) 上で実行されるので，Java の頑健なセキュリティモデルを使用することができる．この時，サーバの管理者は，適切なセキュリティポリシーファイルを JVM に渡す必要がある．

5. まとめと今後の課題

本稿では，グリッドポータルにおける動的なアプリケーションの開発実行環境のアーキテクチャの提案を行なった．このアーキテクチャをグリッドポータルに導入することによって，ユーザは事前に用意されたアプリケーションばかりでなく，ユーザ自身がアプリケーションを開発し，そのアプリケーションポータルを動的に作成することができるようになる．

Ninf-G の Java API は実装済みであるので，近々，Ninf プロジェクトの Web ページ [2] にて公開する予定である．今後の課題としては，本稿の設計に従ってアプリケーション開発実行環境の実装を行うこと，開発環境にデバック機能を設けることが挙げられる．

謝　　辞

本研究の一部は，文部科学省科学研究費補助金（研究番号 06767) によるものである．

参 考 文 献

1) Python. http://www.python.org.

2) Satoshi Sekiguchi, Mitsuhisa Sato, Hidemoto Nakada, and Umpei Nagashima. – Ninf – : Network base information library for globally high performance computing. In *Proceedings of Parallel Object-Oriented Methods and Applications (POOMA)*, Feb. 1996. http://ninf.apgrid.org/.

3) Toyotaro Suzumura, Hidemoto Nakada, Masayuki Saito, Yoshio Tanaka, Satoshi Matsuoka, and Satoshi Sekiguchi. An Automatic Generating Tool for Computing Portals. In *JavaGrande 2002, To be appeared*, Nov 2002.

4) Gregor von Laszewski, Ian Foster, and Jarek Gawor. CoG Kits: A Bridge Between Commodity Distributed Computing and High-Performance Grids,A Java Commodity Grid Kit. In *ACM 2000 Java Grande Conference*, June 2000.

5) 中田 秀基, 斎藤 真幸, 鈴村 豊太郎, 田中 良夫, 松岡 聡, and 関口 智継. Grid ポータル構築ツールキット Ninf-Portal. In *JSPP2002* **論文集**, pages 209–216, 5 月 2002 年.

6) 田中良夫, 中田秀基, 平野基孝, 佐藤三久, 関口智嗣, and 中田秀基. Globus による Grid RPC システム の実装と評価. In *Proceedings of Swopp 2001*, 7 月.