

“ Grid ” の生理学

分散システムインテグレーションのためのオープングリッドサービスアーキテクチャ

Ian Foster^{1,2} Carl Kesselman³ Jeffrey M. Nick⁴ Steven Tuecke¹

¹ Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439

² Department of Computer Science, University of Chicago, Chicago, IL 60637

³ Information Sciences Institute, University of Southern California, Marina del Rey, CA 90292

⁴ IBM Corporation, Poughkeepsie, NY 12601

foster@mcs.anl.gov carl@isi.edu jnick@us.ibm.com tuecke@mcs.anl.gov

概要

e-ビジネスおよび e-サイエンスにおいて、分散した、複雑かつ動的な「仮想的な組織」にまたがって、サービスの統合が必要とされることが多くなっている。このような「仮想的な組織」は、一つの企業内の分散された資源から構成されたり、外部のリソース共有やサービスプロバイダーなどを通じて構成される。この統合は、本質的に異なるプラットフォーム上で実行される場合にも様々な QoS (Quality of Service) を達成する必要があるため、技術的にチャレンジングなものとなる。本稿では、このような課題に取り組むための Open Grid Service Architecture について述べる。グリッドとウェブのコミュニティから概念と技術の上に、このアーキテクチャは均一で明確なサービスセマンティクス (グリッドサービス) を定義し; 変化するグリッドグリッドサービスインスタンスを作成、名前づけし、discovery するための標準的な機構を定義し; サービスインスタンスに対して、位置透明性と複数プロトコルのバインディング提供し; 基盤となるそれぞれのプラットフォームの設備の統合をサポートする。Open Grid Service Architecture は、また、ウェブサービス記述言語 (WSDL) インターフェイスと関連する記法を用いて、life time 管理、変更管理、告知などの洗練された分散システムを生成し、構成するために必要とされる機能を定義する。サービスバインディングによって信頼性の高いサービスの起動、認証、権限委譲を必要に応じて行うことが可能である。本論文は先行論文「The Anatomy of the Grid」を補足するものであり、組織内もしくは組織間のドメインにまたがって、どのようにしてグリッド機能を用いてサービス指向アーキテクチャを実装するかについて述べ、どのようにしてグリッド機能をウェブサービスのフレームワークに組み込むか、分散システムインテグレーションの基本としての商用コンピューティングの分野で、どのようにして本アーキテクチャを適用するかについて、明らかにする。

1.はじめに

近年まで、アプリケーションの開発者は同質的で信頼度、安全性が高く集中管理された環境を対象として仮定することができた。しかし、次第に、コンピューティングは、共同作業やデータ・シェアリングなど、分散されたリソースを含むを使った新たな形でのインタラクションに関係するようになってきている。その結果、インテリジェントネットワーク、スイッチ機器、キャッシュサービス、アプライアンスサーバ、ストレージシステム、ストレージエリアネットワーク管理システムなどいろいろな形で、企業内及び企業間でのシステムの連結が注目されている。加えて企業はその IT 環境の中で重要でない要素を様々なサービスプロバイダーにアウトソースすることで大幅なコスト削減が達成できることに気付きつつある。

このような状況の変化により、分散アプリケーション開発と設置に新たな技術開発が求められている。今日、アプリケーションとミドルウェアは、そのアプリケーションを実行するためのホスト環境を提供する特定のプラットフォーム (Windows NT, Unix, mainframe, J2EE, Microsoft.NET など) で開発されているのが普通である。それらのプラットフォームで提供されている機能は、統合されたリソース管理機能からデータベース統合から、クラスタリングサービス、安全性、仕事量管理、問題確定まで、様々なものがあり、それらにはまた様々な実装方法や semantic behaviors があり、それぞれのプラットフォームではそこでの機能を使うための API など提供されている。しかしながら、このように様々な機能が提供されているにもかかわらず、ソフトウェア、ハードウェア、そして人的な資源が次第に分散化されていくにつれ、--- 一般的なセキュリティ構造、分散された仕事の処理、資源管理パフォーマンス、フェイル・オーバー (coordinated fail-over)、問題確定サービスなど、どの指標で測られるにせよ --- 企業システム、サービスプロバイダシステム、そして消費者システムより有機的に集められたリソースによって求められている QoS(QoS, quality of service)を達成することは重要になっている。分散した、広域ネットワーク (WAN) をまたがるリソースやサービスにアプリケーションがアクセス、共有できるようにするための新しい抽象化と概念が必要になっている。

このような問題は、大規模科学研究のための分散システムの開発者の中心的な関心事であった。この分野での研究がグリッド技術[33,34]の開発をもたらしした。グリッド技術はまさに、このような問題に関するものであり、科学技術計算で広く普及しているのを見ることができる。

既出の論文において、我々は Grid 技術とそのインフラを動的で、分散した仮想的な組織 (VOs, virtual organizations) におけるいろいろなリソースの共有と協調をサポートするものと定義した[34]。さらに、グリッドの本質的な性質を定義し、プロトコルとサービスに対し必要な条件を明確にして、コミュニケーションと認証に関する connectivity protocol、個々のリソースへ協調してアクセスするための resource protocol、そして複数のリソースの協調的な使用に関する collective protocol に明らかにした。幅広い主要な e-サイエンスのプロジェクトをサポートする重要な Grid プロトコルのオープンソースの参考実装として、Globus Toolkit[29]についても述べた。

本論文では、どのように Grid が働くのか、そして Grid 技術がどのように実際に実装、適用されるのかということをも 3つの観点からより正確に定義していく。はじめに、Grid が VO 要素間のインタオペラビリティを持つために必要なプロトコルという観点から[34]が構成されていたが、ここではプロトコルに対応したサービスの性質に焦点を当てる。我々は、Grid を、VO のニーズに対応するために様々な方法で集約された Grid サービスの拡張された集合と見なし、それぞれの VO 自体はそれらが動作、共有するサービスの一部として定義される。そこで、分散システムの統合をサポートするためにそのようなグリッドサービスが持つべき挙動について定義する。機能 (すなわち、“生理学”) に重点を置くことによって、Grid に対するこの見方は以前に述べたプロトコル指向の (すなわち、“解剖学”) 記述を補足するものになる。

次に、サービス記述と discovery、サービス記述からのクライアントとサーバーコードの自動生成、インターオペラブルなネットワークプロトコルへのサービス記述のバイディング、最近の高レベルのオープンな規格やサービス、ツールとの互換性、様々な企業サポートなどの、望ましいウェブ・サービスを利用するための web サービスの技術[40,47]を Grid 技術とどのように連携しうるか説明する。この連携を OGSA(open Grid services architecture)と呼ぶことにする。ここでアーキテクチャという用語は有用なシステムを構築する well-defined な基本インターフェイスの集合、そして open は拡張性、ベンダー中立性、そしてコミュニティの標準化プロセスにコミットしていくことを意味する。このアーキテクチャでは Web Services Description Language(WSDL)を使用することで、自己記述される発見可能なサービスと互換性のあるプロトコルを実装し、複数の整合性のあるインターフェイスをサポート、管理を変更するといった拡張性を備えることができるようになる。OGSA は Globus Toolkit で培った経験から、グリッドサービスについての規約や WSDL のインタフェースや、(必要などときには)信頼性の高く安全な起動をサポートする(おそらく変化していく)状態を持ったサービス、life time 管理、通知、ポリシー管理、仮想化について定義している。OGSA はまた、グリッドサービスの発見や変化していくグリッドサービスのインスタンスに対するインタフェースを定義している。結果として、最近の企業や多くの組織同士で計算するような環境で求められているような洗練された分散処理システムの構築をサポートする、標準化に基づく分散サービス(我々はその意味が重複しているため、あえて分散オブジェクトシステムという用語を避けている)となっている。

3 つ目のポイントとして、[30,34]で重視された科学や技術利用よりもむしろ、商業利用を目的とした議論に焦点を当てる。基本的にはどちらでも同じ規則と仕組みが適用できうると考えている。しかし、商業的利用には、特に現在あるリソースとアプリケーションのシームレスな統合、そして仕事量、リソース、安全性、ネットワーク QoS,そして可用性管理のためのツールの、シームレスな(きれめのない)統合が必要となる。OGSA においていろいろな性質をもつサービスを発見する機能をサポートすることによって、元のプラットフォームにおいて提供されている機能に対して高レベルな Grid サービスのマッピングもしくは「適用」ができるようになる。OGSA のサービス志向はまた、リソースを複数のレベルで仮想化することを可能とし、いくつものドメインにまたがった共同作業をサポートする分散グリッド内であろうと、単一の IT ドメイン内で複数の階層からなるホスト環境内であろうとも同じ抽象化と仕組みが同じように使えるようになる。共通のインフラストラクチャとは、相互にインタラクションする仕組みが違うのではなく、リソースの所有権やプライバシーやセキュリティに関するポリシーの制御が異なることに起因することを意味する。それゆえ、今の企業のシステムが個別の計算資源から統合した階層化された分散システムへと変化しているように、複数のサービスコンポーネントが組織内はもちろん組織の境界を越えて、動的に柔軟に統合することができる。

本稿では以下について述べる。2 章では Grid 技術の商業利用の有用性について検討する。3 章では Globus toolkit と web サービスを概観し、4 章において、OGSA の動機と概要について述べる。5 章から 8 章では例をあげて実際のプロトコル運用と高レベルのサービスについて述べる。9 章では関連研究について述べ、10 章で全体のまとめを行う。

ここで、OGSA とそれにとまなうグリッドサービスの仕様は global Grid forum や Globus project 内、その他での研究によりいまだ改良され続けているものであることを断っておく。したがって、本論文とこれまでの概説の技術的な内容は現在進行中の研究のごく一部を扱ったものである。

2、Grid 技術の必要性

Grid 技術は動的に変化する VO の様々な資源の共有、協調した利用をサポートする。それはつまり、地理的にも組織的にも広範囲にわたる要素から、必要とされる QoS[34]を提供するための十分統合された、仮想的なコンピューティングシステムの構築をサポートする。

Grid 技術とそのコンセプトは当初、別々の科学技術の共同研究においてリソース共有するために開発された[18,19,28,30,46,64]。その応用としては、巨大な科学技術データの協調可視化技術(専門的知識の共用)、大規模な計算が必要なデータ解析についての分散コンピューティング(コンピュータの能力と記憶装置の共有)、実験装置をリモートのコンピュータやアーカイブの結合(機能性及び有益性の向上)[45]などがある。同様なアプリケーションが商用的にも重要になるであろう。まずは、科学技術に用いられるアプリケーション(そのあたりはもう成功しているとすでに指摘している)はそうであるし、次に企業内でのアプリケーションの統合化やインターネット上のビジネス間(B2B)パートナーコラボレーションなどの、商業分散コンピューティングアプリケーションが重要となってくると思われる。それは world wide web が共同研究用に始まり、それが e-ビジネスに応用されたように、似たような軌道を Grid 技術もたどることを期待している。

しかしながら、我々は、Grid 技術は本質的に能力を高める手段としてだけではなく、高信頼でスケラブルでかつ安全な分散システムの構築に関する新たなチャレンジに対するソリューションとして非常に重要であると考え。このチャレンジは技術のトレンドから商用面からの要請からおこった現在の流れからきたもので、これから述べるように、これまでのモノリシックで集中的なホストでのサービスを分解し、分散化しようというものである。

2.1 エンタプライズコンピューティングの進化

これまで、コンピューティングは高度に統合されたホストからなる企業のコンピュータセンターにより行われるものであった。洗練された分散システム(command and control system や reservation システムや Domain Name System)などが存在しても、これらは依然として、特殊で、限られたものであった。

しかし、インターネットの普及と e-ビジネスの登場により、企業の IT インフラも外部のネットワークやリソースやサービスと密接に関連していることがしだいにわかってきた。当初は、システムを複雑化するこのような要素は、ネットワークに特有な現象と取られていて、“周辺のサーバー”だけで従来の企業の IT データセンターとやり取りを行う“インテリジェントネットワーク”を構築しようという試みが行われていた。例えば、企業の存在を示すためだけの web や企業内のネットワークを外部のプロバイダにつなげるバーチャルプライベートネットワークサーバはその一つである。したがって、企業の核なる IT インフラに対する e-ビジネスとインターネットの影響はうまく管理でき、一線を引くことことができるものであると考えられていたのである。

一般的には、その試みは失敗した。なぜなら、IT サービスのコンポーネント化はまさに企業内の IT システムの内部でも起こっていたからである。新たなアプリケーションが、ベースとなるコンピューティングプラットフォームからアプリケーションを分離し、複数のプラットフォームにポータブルに配置できるようにするプログラミングモデル(Enterprise JAVA Beans component model[65]など)で、開発されるようになってきている。このようなポータビリティにより、どのような OS がサポートされているかではなく、パフォーマンスを実装するための価

格や QoS という観点からから、プラットフォームを選ぶことができるようになった。したがって、例えば web サーバやよく使われるアプリケーションは旧来のメインフレームでなく市販のサーバを対象としている。Unix や NT のサーバが増殖するにつれ、旧来のメインフレームのアプリケーションやデータ資産などに対するコネクションが必要となった。このような処理の増加に伴い、企業はそれほど重要でないデータベース処理などの問題を、中枢の業務処理システムから、中間のサーバに移動した。その後しばらくして、企業のリソースへの web からのアクセスが行われるようになると、以前にも増して早くサービスすることが求められるようになり、さらにコンテンツを周辺のネットワークに近いところに分散化したりキャッシュしたりすることが必要となってきた。結果として、高度に統合された内部 IT インフラは、ヘテロな断片化されたシステムの集合に分解されることになった。そして、企業は外部ネットワークでよりもさらに、企業内でデータ検索、分散セキュリティ、コンテンツ分散についての課題に取り組み、それらの分散されたサーバやリソースを再統合しなくてはならなくなった。

これらの進歩と平行して、企業は e-ビジネスにより前向きに取り組むようになり、高度に頑強な IT インフラが予期できない問題に対処したり、急激な拡大に対処するために必要であると認識するようになってきている。企業はまた、顧客関係の管理、統合流通、既存の核のシステムなどをうまく統合しようと、企業内の在庫管理プロジェクトの対象と規模を拡大させている。これらの進歩は、企業の IT インフラに関する重要な圧力になっている。

それらを統合することによって得られる効果とは“従来、メインフレームでホスト重視のコンピューティングにより行われていたサービスの質を提供することは、企業の外部に対してはもちろ内部に対しても、分散されたコンピュータ上の e-ビジネスを有効に進める上で重要である”ということである。例えば、平均とピークの時では仕事の量は大幅に違ってくるにもかかわらず、企業は消費者の要望に対して一定の時間で対処しなければならない。そのため、与えられた仕事の需要と優先度によって、柔軟な資源割り当てが必要となる。企業は複数の異なるサーバで処理される分散トランザクション処理に安全で信頼できる環境を提供しなくてはならず、エンドユーザにもいつでも連続して利用可能で、分散されたデータサーバとアプリケーションの間で流れている仕事に対して災害時の復旧を提供しなくてはならない。依然として、特定のプラットフォームにあるコンポーネントとサービスを垂直統合してアプリケーションに QoS を提供する方式は現在の分散された環境では動作しない。すなわち、モノシリックな IT インフラをコンポーネントを分解しても、与えられたプラットフォームでサービスを垂直統合して得られる QoS とは同等のものにならない。また、分散リソース管理は有効でもないし、個々の所有物であるということや、プラットフォームの資源にアクセスできなかったり、分散環境にある類似のリソースの一貫性のなさにより制限されている。

このような動向の結果として、IT システムインテグレータにとって、分散されたリソースを要求された QoS を実装するように再統合することが、重荷となっている。しかしながら、適切なインフラのツールがなくては、分散コンピューティングのワークフローを管理するのは、次第に人力を必要とし複雑で脆弱なものとなっている。特定のプラットフォームを保守しているスタッフはシステム全体の可用性や性能に問題ないか絶えず監視し、異なるプラットフォームで適切なオペレーションを行うために連絡して共同作業を行わなくてはならなくなっている。このような状況は、計算環境やアプリケーションの変化に対しては、スケーラブルでも費用効果が良いわけでもなく、何らかの改善を行わなくてはならないようになってきているのだ。

2.2 サービス・プロバイダーと B2B コンピューティング

もう一つの重要な流れは様々な種類のサービスプロバイダ (SP) の出現で、その種類は、web のホスティング、コンテンツの分配、アプリケーション、記憶装置などがある。スケールメリットを利用して、web ポータルの構築などの SP は標準の e-ビジネスプロセスを得ることを目的とし、複数の顧客によりよいコストパフォーマンスよいサービスを提供することである。昔からある自分の IT インフラを持っている企業さえも、このようなプロセスはコモディティな機能としてみているため、これらを外部委譲している。

これらの e-utility (この用語は継続的なオンデマンドアクセスを提供するサービスプロバイダのことを指す) の出現により、定量制の利用や登録サービスなどを通じて、高グレードの IT リソースのモデルを提供し始めている。昔のオフラインでバッチ処理を提供しようとしていた昔の計算サービスと違い、e-utility により提供されるリソースは、大抵、企業のインフラと密接に統合され、自社とアウトソーシングされるリソース双方にわたるビジネスプロセスに用いられる。そのため、e-utility の構造により可能となる、スケールメリットによる価値とは、企業レベルの計算機能がより細分化され分散されたものとなることである。E-utility のプロバイダはそれぞれの技術的な問題に直面している。スケールメリットを利用するために、e-utility のプロバイダはサーバのインフラが顧客の要望に応じて簡単にカスタマイズできるようにしなければならない。そのため IT インフラには以下のような要請がある：(1) サービスレベルでの契約方針、設備レベルでの IT インフラの効率的な共有と再利用、末端のネットワークからアプリケーションやデータサーバに至るまでの分散されたセキュリティにしたがって、動的なリソースの割り当てをサポートしなければならない。(2) 一定以上のレスポンスタイム及び高レベルの可用性を提供する - それには末端から末端までの性能監視、リアルタイムで再構成が必要となる。

もう一つの重要な IT のトレンドとして、複数の組織に跨るサプライチェーンマネジメントや、仮想的な web モール、電子オークションなどの、企業間のビジネス間(B2B)のコラボレーションがある。B2B の関係は上に述べたように事実上の仮想的な集団だが、セキュリティ、監査能力、可用性、サービスレベル合意、複雑なトランザクションフロー、など特に厳しい要求がある。そのため B2B コンピューティングは、分散システムの統合に対する別の観点からの要請を代表しており、その特徴は異なる組織に配置された IT 技術がしばしば大きく異なっていることが多いことである。

3. 背景

OGSA を定義するために、その基礎となる 2 つの技術を概観する：一つは Globus Toolkit で、今まで学術や技術計算で Grid 技術に広く適用されてきた。もう一つは web サービスで、ネットワークアプリケーションにアクセスするための一般的な標準規格ベースのフレームワークとして登場した。

3.1 Globus Toolkit

Globus Toolkit[29,34]とは、Grid とそのアプリケーションをサポートする、コミュニティベースの、オープンなアーキテクチャを持つ、オープンソースのサービスとソフトウェアのライブラリである。この Toolkit は、セキュリティや、information discovery、リソース管理、デ

ータ管理、通信、障害検出、移植性についての問題について取り組んでいる。Globus Toolkit の機構は、何百ものサイト、いくつもの世界規模の有名な Grid のプロジェクトで実際に使われている。

Toolkit の中でも特に OGSA と関係が有るのは以下のものである。(1)GRAM(Grid Resource Allocation and Management) プロトコルと、その”gatekeeper”サービスはセキュアで信頼の置けるサービスの作成及び管理を提供する[25]。(2)MDS-2 (meta discovery service) [24]は soft state registration による情報発見、[59,69]、データモデリング、local registry を提供する。(3)GSI (Grid security infrastructure、) は single sign on、委任、credential mapping をサポートする。図 1 に描かれているようにこれらのコンポーネントはサービス指向のアーキテクチャに必要な要素を提供するが、OGSA のアーキテクチャに比べると一般的ではない。

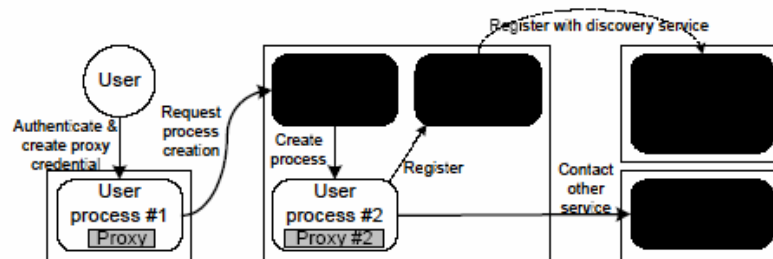


図 1 Globus tool kit の一部。最初に initial credential を生成し、次にリモートの gatekeeper サービスに対して認証されたリクエストを発行すると、関連する（おそらく制限された）proxy credential を持つユーザプロセス # 2 が生成され、さらに、それが他のサービスをリクエストする様子を示している。また、MDS-2 による soft-state サービス登録についても示す。

GRAM プロトコルは任意の計算についての高信頼で、セキュアなりモート起動および管理の機能を提供する。それは本文では transient service instance (一時的なサービスインスタンス) と呼ぶ。GSI の機能はリモートの計算の認証、認可、credential の委譲[38]などに使われる。two-phase commit プロトコルは Condor system[52]で用いられた技術をベースに、reliable な起動にもちられる。サービスの作成は小さな reliable な”gate keeper”プロセス(本稿では factory という用語を用いる)により行われ、GRAM reporter はローカルな計算の状態や認識を監視し(registry)、公開する。

MDS-2 [24]は、システム構成情報や計算サーバの構成などの状態情報や、ネットワークの状態や、複製したデータがどこにあるかなどの情報を発見したり、それにアクセスするための統一したフレームワーク(これを discovery インタフェースと呼ぶ)を提供する。MDS-2 は soft-state プロトコルと Grid notification プロトコル[44]を、公開情報の life time 管理に使う。

公開鍵暗号に基づく GSI (Grid Security Infrastructure) プロトコル[33]は、single sign on の認証、通信保護、及び制限された委譲の際の初めの処理の一部をサポートしている。簡単に言うと、single sign on により一旦認証されると、プログラムがユーザに代わって任意のリモートサービスについて認証することができる proxy credential の作成することができるようになる。Delegation (委譲) というのは、ユーザーに代わってリモートサービスが使用できる委譲

された proxy credential のリモートサービスを作成と通信を許可するものである。ただし、この場合多くの制約があることがある：このような機能は、ネストされた操作をするのに重要である。(似たようなものに Kerberos[63]があるが、両者は潜在的に違う性質を持っている)

GSI では X.509 という広く使われている PKI (公開鍵インフラ) 認証系をユーザー認証の基本として使っている。GSI は X.509 の proxy certificate[67]を single-sign on と委譲のために定義している。(Kerberos の forwardable ticket と似ているが純粋な公開鍵暗号の技術に基づいている。)他の公開鍵暗号からなるプロトコルも X.509 認証に使うことができるが、GSI は典型的な場合には認証のための TLS(Transport 層レイヤー)のセキュリティプロトコル(SSLの後継)を使っている。リモートから委譲するような X.509 の proxy certificate のリモート委譲プロトコルは、TLS の上のレイヤーとして構築されている。IETF (Internet Engineering Task Force) のドラフトでは X.509 Proxy certificate 拡張について定義している[67]。Globus Toolkit Forum のドラフトは、X.509 Proxy Certificate を生成する委譲プロトコル[67]や、Grid プログラミングに使いやすいようにした GSS-API の拡張について定義している。

このような制約された委譲サポートはプロトタイプとして実証実験されており、提案されている X.509 Proxy Certificate Profile[67]の主要な一部分になっている。制約付の委譲を使えば、あるエンティティは全ての特権の一部のみを他のエンティティに委譲することを可能とする。このような制約は、故意にしる偶然にしる委譲された credential の誤用の危険性を減少させるために重要である。

3.2 web サービス

web サービスという用語は注目されつつある重要な分散計算のパラダイムのことであるが、ヘテロな分散コンピューティングに対して単純で、インターネットベースの標準的な技術(例えば XML[14,27])に焦点を当てており、DCE、CORBA、Java RMI など他のアプローチとは異なるものである。Web サービスは、アクセスするソフトウェアのコンポーネント、それらのコンポーネントにアクセスするメソッド、関連性のある SP(service provide)を識別するような discovery メソッドを記述する技術を定義している。web サービスはプログラミング言語、プログラミングモデル、システムソフトウェアの仲立ちをするものである。

web サービスの規格は W3C やその他の標準化団体において制定作業が進められており、Microsoft(.NET)、IBM(Dynamic e-business)、Sun (Sun ONE) などのメジャーな新規ビジネスを主導する活動のベースとなっている。我々は特に SOAP、WSDL、WS-Insection という 3 つの標準化技術について関心をもっている。

- SOAP (The Simple Object Access Protocol)[4]はサービスの提供者と利用者間のメッセージのやり取りの手段を提供する。SOAP は RPC (remote procedure call:遠隔手続き呼び出し)やメッセージのやり取りについての規約を定義する XML の payload についての簡潔な梱包機構である。SOAP は transport プロトコルとは独立している。すなわち、SOAP payload は、Http、Ftp、JMS (Java Message Service)などにより運ばれる。ここで重要なのは web サービスが、その下にあるソフトウェアコンポーネントに対し複数のアクセス機構を記述することができることである。SOAP は web サービスの起動の形式を与える一つの手段に過ぎない。

- WSDL (The Web Service Description Language)[22]は web サービスをドキュメント指向 (message による) もしくは RPC Payload を含むメッセージに作用する endpoint の集合を定義するための XML ドキュメントである。サービスのインタフェースはメッセージ構造と一連の簡単なメッセージ交換 (又は WSDL の用語でいえば、オペレーション) により抽象的に定義され、endpoint を定義する具体的なネットワークプロトコルやデータのエンコードフォーマットなどに結合される。関連する具体的なエンドポイントは、まとめて抽象的なエンドポイント (又はサービス) として定義される。WSDL は色々なメッセージのフォーマットやネットワークプロトコルに対してそのメッセージについて具体的な表現やエンドポイントを記述することができるように拡張することができる。数々の標準化されたバインディング規則は、WSDL をどうやって、SOAP1.1, HTTP GET/POST,MINE などと組み合わせるかについて記述し定義されている。
- WS-Inspection [15]は単純な XML 言語と、どのようにしてサービスプロバイダから公開されているサービス記述を見つけるかについての関連する規約からなっている。WSIL (WS-instruction language) のドキュメントには、サービスについての幾つかの記述や、他のサービスについての記述に対するリンクを含めることができる。サービスの記述は通常 WSDL ドキュメントへの URL である。すなわち、場合によってはサービスの記述は UDDI (Universal Description, Discovery, and Integration) [5]レジストリへの参照である。リンクは、通常、他の WS-Inspection ドキュメントへの URL である。すなわち、たいていはリンクは UDDI エントリへの参照である。WS-Instruction によりサービスプロバイダは WSIL ドキュメントを作り、ネットワークからアクセスできるようにする。サービス要求は (HTTP GET などの) web ベースのアクセスの仕組みを使い、このドキュメントを回収しサービスプロバイダの告知したサービスを見つけ出す。WSDL は違ったインデックスの形式でも構成されることがある。

いろいろなほかの web サービスの規格が、すでに制定済み、もしくは制定中である。例えば WSFL (Web Service Flow Language) [6]は web サービスの orchestration, すなわち、より単純な web サービスを組み立てることで洗練された web サービスを構築する技術について取り組んでいる。

我々の目的に関して、Web サービスのフレームワークは 2 つの利点がある。まず、我々は、ヘテロな環境において様々なサービスの動的な発見と構築をサポートする必要があり、そのためには、インタフェースの定義とエンドポイントでの実装の記述を登録・発見するための機構ならびに、特定のインタフェースに対する (複数であることがある) バインディングに基づいた複数の proxy を動的に生成するための機構が必要となる。WSDL は特定のバインディング (transport プロトコルやデータの符号化形式) の中で、それら具体的な中身とインタフェースを分離して定義するための標準的な機構を提供することにより、これらの要求をサポートしている。第二に、web サービスを広く適用することは、web サービスに基づくフレームワークが、多くのツールと関連するサービスを利用できることを意味する。そのようなツールとしては、様々な言語に対して言語バインディングを生成することができる WSDL プロセッサ (例として WSIF, Web Service Invocation Framework[53]) や WDSL 上に構築されるワークフローシステム、 (.NET や Apache Axis のような) web サービスのホスティングの環境がある。但し、ここで、web サービスの使用が SOAP の使用を全ての通信で使用することを意味するわけでない。もし必要ならば、例えばさらに性能を向上させるために他の通信方法も使えるし、特定の

ネットワークプロトコルで動作させることも可能である。

4 An Open Grid Services Architecture

我々は IT インフラストラクチャー企業内や、SP により強化された IT インフラや複数組織でのグリッドの内部において、コンピューティングは資源やサービス（そして人的な資源）の動的な集合体 - これを *virtual organization* と呼ぶ - の生成、管理、応用に次第に関連しつつあることについて述べてきた。ここでは、これらの集合体は小規模であることもあるし、大規模であることもある。また、長期的なこともあるし短期的な場合もある。さらに、単一の組織のこともあるし、複数の組織に跨ることもある。そして、均一なこともあるしヘテロな場合もある。個々の集合体は小さなシステムから階層的に構成されているかもしれないし、メンバーが重複していることもありうる。

我々はこれらの相違にかかわらず、VO に使われるアプリケーションの開発者は、QoS - これは、共通のセキュリティ機構、分散環境での workflow や資源の管理、統合された fail-over 機構、問題決定サービス、もしくは他の指標で推し量れるもものであるが - を提供しようとしている時に、ヘテロでかつ往々にして他の動的な特性を持つ様々な資源に跨って、共通の要請に直面していると考ええる。

そこで、我々は、これらの要請の性質と、実際の場面においてそれらに取り組むために必要な仕組みに目を向けることにする。[34]で述べた我々の分析を発展させて、VOs によって整備されるサービスの集合体の生成、保守、そしてアプリケーションをサポートする Open Grid Services Architecture を導入することにしよう。

まず、サービス指向のグリッドアーキテクチャの機能に関する一般的な考察とグリッドサービスを仮想化することができる重要性、そして本質的なサービスの性質を述べることから始めることにする。そして、我々がグリッドサービスと呼ぶものの定義での標準化しようとしている重要な点について述べる。より技術的な説明については 6 章で示す。

4.1 Services 指向と仮想化

VOs について述べる時には、我々は（[34]で述べたように）共有されている物理資源もしくはこれらの資源によってサポートされるサービスに注目することができる。（サービスとはなんらかの機能を提供するネットワークから利用可能な entity である。オブジェクトという言葉は議論の中では使われることがあるが、他の意味で使われることがあるためこの用の使うことを避けた）OGSA において、我々は services に注目する：すなわち、計算資源、ストレージ資源、ネットワーク、プログラム、データベース、それらの類似もものはすべて、サービスとして表現される。

我々の見方に関わらず、分散された、複数の組織にまたがるグリッド環境における重要な要請とは、相互運用ができるメカニズムに対してのものである。サービス指向の観点からすると、我々は相互運用の問題を 2 つの問題に分けることができる。すなわちサービスインタフェースの定義と特定のインタフェースを起動するために用いることができるプロトコルの識別である - 概念的には、そのようなプロトコルの標準的な集合に関する合意である。

サービス指向の観点に立つことによって、標準的なインタフェースのメカニズム定義や

Local/remote 透過性、ローカルOS サービスへの適応、そして均一なサービス semantic の必要性に取り組むことができるようになる。すなわち、様々な実装の共通なインタフェースをカプセル化する必要がある。仮想化することにより、ローカルもしくはリモートでのロケーションの透過性により複数のヘテロなプラットフォームに跨って一貫性のあるアクセスを提供し、複数の論理資源インスタンスから同一の物理資源へのマッピングを可能にし、低レベル資源から構成されるVO内での資源のマネージメントが可能になる。仮想化は一連の構成されたサービスの集まりをより高機能なサービスからの構成することを可能にする。構成されるサービスがどのように実装されるかにはよらない。グリッドサービスの仮想化はまた、共通サービスの semantic の振る舞いをネイティブなプラットフォームの機能の上へシームレスにマップする機能を支えるものである。

仮想化はもしサービスの機能が標準的な形式で表現されうるのであれば、簡単であり、どのようなサービスの実装も同じように起動することができる。我々がこの目的のために WSDL を用いるが、WSDL は、サービスの起動に用いられる代用されるプロトコルバインディングとは別に、サービスインタフェース定義をサポートしている。WSDL は、同一ホスト上での要求やサービスプロセス間の相互作用のために局所的に最適化されたバインディング（例：local IPC）はもちろん、分散環境での通信プロトコル（例：HTTP）など、一つのインタフェースに対して複数のバインディングを可能にしている。他のバインディングの属性として、認証や credential の委譲はもちろん、信頼性（と様々な QoS の属性）を持つことができる。バインディングの選択は常にサービス起動 semantics という観点では、要求側に対して透過性を持つものでなくてはならない - しかし、ほかの点、例えば、要求する側は性能を理由に確かなバインディングを選ぶことができるようにするなどについてはユーザが知る必要がある。

サービスインタフェースの定義とアクセスバインディングはまた、サービスの機能の実装から分離されている。サービスは異なるプラットフォーム上での複数の実装をサポートし、ネイティブなプラットフォームの機能だけでなく、サービスの実装を多重に使うことにより、資源の仮想的な集まりに対しても、シームレスに重ね合わせることを可能にする。プラットフォームや場合に依りて、我々は以下のような実装のアプローチを使うことができるであろう。

1. サービスをホスティングする実行環境（container）をサポートするために複数のプラットフォームに跨る完全な可搬性のために構築した参照実装を使うことができる。
2. サービス機能を提供するために特化されたネイティブなプラットフォームにおいて、サービスインタフェースの定義からネイティブなプラットフォームの機能をマップすることができるであろう。
3. 高レベルなサービスが複数の低レベルのサービスの集合から構成され、その低レベルのサービスそれぞれ自身がネイティブな機能にマップされるか、さらに分解されるように、これらの機構を再帰的に適用することもできる。したがって、サービスの実装は低レベルなサービスの操作に振り分けられる（4.4 も参照のこと）。

例として、リポジトリにトレースレコードを記録する分散トレース機構を考えてみよう。ロバスタなトレース機構をサポートしていないプラットフォーム上では、オンデマンドにトレースレコードを格納、検索するためのサービス実行環境において、参照実装が生成、ホスティングされる。しかし、ロバスタなトレース機能をすでに保持しているプラットフォーム上では、

我々はネイティブなプラットフォームの機構によってトレースサービス機能を構築することができる。すなわち、分散トレースサービスを通じて論理的なトレースストリームを意味的に保持しつつ、既存のトレース管理ツールや、付加的なオフロード機能、dump/restore、そして類似の機能を活用する。最終的には、高レベルのサービスの場合、低レベルサービスから取得したトレースレコードは結合され、そのサービスのための構築されたトレース機能として動作するであろう。

資源の振る舞いのこの仮想化の中心となるものは特定のホスト上でのオペレーティングシステムの機能へ橋渡しする能力である。これらのマッピングを実現するときの重要なチャレンジはネイティブな機能 - パフォーマンスのモニタリング、ワークロード管理、問題決定、ネイティブなプラットフォームセキュリティポリシーの施行に関する事柄 - を十分に活用することができるようにすることであり、グリッド環境はその構成する一つ一つの最小公約数とはならない。グリッドサービスメカニズムの discovery は、この点において重要であり、特定のインタフェースの実装によってサポートされる能力が何かを発見するために高レベルサービスを提供する。例えば、ネイティブなプラットフォームが予約機能をサポートしているならば、資源管理インタフェースの実装（例えば、GRAM[25, 31]）は、これらの機能を活用できるようになっている。

したがって、我々のサービスアーキテクチャはサービスが提供される場所と起動に関してローカル、リモートにかかわらず位置透過性をサポートする。また、このアーキテクチャは複数のプロトコルバインディング (multiple protocol bindings) を提供しており、サービスをリクエストする側でローカルにサービスがホストされる時にはサービス起動に対してローカルな最適化ができるようにする。もちろん、これによって、いろいろな interGrid プロトコルがあり、それらが異なる目的で最適化されている組織間の境界においては、その間のネットワークフローに対してプロトコルのネゴシエーションが可能となる。最後に、特定のグリッドサービスインタフェースの実装はネイティブな、分散化されていない、プラットフォームの機能や能力にマップされることがあることを注意しておく。

4.2 サービス semantics : Grid サービス

サービスの仮想化や構築する能力は標準的なインタフェース定義以外のものにも依存する。我々はまた、例えば、異なるサービスがエラー通知のために同じ規約に従うように、サービス間のやり取りのための標準的な semantics を必要とする。この延長上において、OGSA は我々が Grid サービスと呼んでいるものを定義する。Grid サービスとは明確に定義された一連のインタフェースを提供する Web サービスであり、特定の規約に従うものである。これらのインタフェースは discovery、動的なサービスの生成、ライフタイムの管理、通知、管理に関するものであり、規約とはネーミングや更新に関するものである。我々はまた、OGSA を高度化していくにつれて、認可 (authorization) や並列性制御にも取り組もうと考えている。認証と reliable な起動の 2 つの重要な課題は、サービスプロトコルバインディングとして捕らえることができ、したがって OGSA のコアの定義の範疇外である。この問題の切り分けによって、機能について妥協することなくアーキテクチャの一般性を高めることになる。

グリッドサービス定義するインタフェースや規約は、特に、一時的なサービスインスタンス (transient service instances) の管理に関連する振る舞いと関係している。VO に参加している人は通常、クライアントからの複雑な処理要求を扱う、永続性サービスの静的な集合を保持しているだけではない。これらは動的に新たな一時的なサービスのインスタンスをインスタン

ス化する必要があることが多く、そのような場合には、インスタンスは特定の要求された処理の状態に対応するやり取りや管理を扱う。その処理の状態がもはや必要でなくなった時には、そのサービスは破棄することができる、例えば、テレビ会議システムでは、ビデオ会議セッションの確立には、QoS の制限に従った end-to-end のデータフローを管理するために中継地点でのサービスインスタンスの生成を行うであろう。また、Web サーバー環境では、サービスインスタンスは、動的に処理能力を増やしアプリケーションのワークロードを管理することによって一定のユーザレスポンス時間を保証するために、動的に生成されることがあるかもしれない。一時的なサービスインスタンスのその他の例としては、データベースに対するクエリやデータマイニング操作、ネットワークバンド幅割り当て、データ転送、処理能力の確保のための事前予約などがある。(これらの例はサービスインスタンスが非常に軽量の entity でよい場合であり、十分に短い時間の処理に対してもインスタンスが生成される例となっている) 一時的 (transience) であることはサービスがどのように管理され、名前付けされ、discovery されるか、そして使用されるかにとって重要な意味を持つ。

4.2.1 Upgradeability の規約とトランスポートプロトコル

複雑な分散システム内のサービスはそれぞれが独立に upgradeable でなくてはならない。したがって、クライアントは特定のサービスのバージョンだけでなく互換性のあるサービスを discovery できるようにするために、サービス間のバージョンと互換性は、管理、記述できなくてはならない。さらに、サービス(と、それが実行されるホスティング環境)は、クライアントの操作を妨げることなしにアップグレードできなければならない。例えば、ホスティング環境のアップグレードはサービスと通信に使われることがある、一連のネットワークプロトコルを変更するかもしれないし、サービス自身のアップグレードは、エラーを訂正したり、インタフェースを改良することもある。したがって、OGSA はいつサービスが変更されたか、これらの変更がインタフェースや semantics (しかし、かならずしも、ネットワークプロトコルということではない) に関していつの時点でバックワード互換性がある変更されたかを確かめることができる規約を定義する。OGSA はまた何の操作がサポートされるのか、どのネットワークプロトコルがサービスとの通信に用いることができるのかなどの、サービスに関するクライアントのサービスの知識を更新する機構を定義している。

サービス記述はサービスとの通信に使用することができるプロトコル・バインディングを示す。以下の2つの属性がそのようなバインディングで必要になることが多い。

- ・ **Reliable なサービス起動** サービスはメッセージを交換することによってお互いに相互作用する。しかし、部分的な障害が起こり易い分散システムでは、メッセージが正しく到着したことを保障できない。内部状態の存在する場合には、サービスがそれまでメッセージを受け取ったかもしくは全く受け取っていないかどうかを保障することが重要となる。この基本的な属性から、トランザクションなど、広範囲の高レベルの操作ごとの semantics を構築することができる。
- ・ **認証** 認証メカニズムにより、個人の認識を可能にし、ポリシーに従ってサービスが確立されることを可能にする。したがって、proxy credential の委譲はもちろん、クライアントやサービスインスタンスの相互認証のために提供される transport プロトコルを必要とすることが多い。この基本的な属性から、いろいろな高レベルの認証メカニズムを構築することができる。

4.2.2 標準インタフェース

グリッドサービスを定義するインタフェース (WSDL の用語や、プロトタイプ) を表 1 に示す。ここで紹介したものは、6 章で詳しく説明する ([66]参照)。OGSA は様々な動作や対応するインタフェースを定義されているが、これらのインタフェース (グリッドサービス) 一つだけ除いてすべてオプションである。

表 1 : OGSA が提案するグリッドサービスインスタンス (詳細はテキスト参照)。与えられた名前は将来変更される可能性がある。認証、ポリシー運用、管理可能性、そのほかの否フェースはまだ定義されていない。

PortType	Operation	説明
GirdService	FindServiceData	グリッドサービスインスタンスについての様々な情報のクエリ。基本の内省情報 (handle、reference, primary key, home handleMap:定義すべき用語) さらに詳細な否フェースごとの情報、サービスに特化された情報 (例えば、レジストリにあるべき情報) を含む、様々なクエリ言語に対する拡張サポート
	SetTerminationTime	グリッドサービスのための終了時間のセット (および、取得)
	Destroy	グリッドサービスインスタンスの終了
NotificationSource	SubscribeToNotificationTopic	サービスに関係されるイベントの告知を subscribe する。通知は、メッセージタイプと関連する記述に基づく。third party のメッセージサービス経由での配信も可。
NotificationSink	DelicerNotification	非同期の告知メッセージの配信を実行
Registry	RegisterService	グリッドサービスハンドルの soft-state 登録を行う。
	UnregisterService	グリッドサービス handle の登録を取り消す
Factory	UnregisterService	新たなグリッドサービスインスタンスを作り出す
HandleMap	CreateService	与えられたグリッドサービスハンドルに現在対応しているグリッドサー

		ビス参照を返す。
--	--	----------

発見 (Discovery) アプリケーションは、サービスが適宜、それ自身とこれらのサービスへクエストを構成できるように、実行可能なサービスを発見し、それらのサービスの特性を決定するためのメカニズムが必要となる。我々は以下の定義することによってこの要求を扱う。

- ・ サービスデータ (service data) の標準的な表現。すなわち、グリッドサービスインスタンスについての情報であり、標準的なフォーマットでカプセル化された、service data elements と呼ばれる名前と型がついている XML の集合として構成される。
- ・ 標準的な操作。個々のグリッドサービスインスタンスからのサービスデータを検索するための (グリッドサービスインタフェースが必要とされる中の) FindServiceData。 (pull モードアクセス; 下にある NotificationSource インタフェースの部分参照)
- ・ レジストリサービス(Registry)を用いて、グリッドサービスインスタンスについての情報を登録するための標準的なインタフェースと、handle から reference へのマッピング (HandleMap-6章で、その名前について議論する時に述べる) のための標準的なインタフェース

動的なサービスの生成 (Dynamic service creation) 新たなサービスインスタンスを動的に生成し、管理する機能は OGSA モデルの基本であり、サービス生成サービスがなくてはならない。OGSA モデルは任意のサービス生成サービスが提供しなくてはならない、標準的なインタフェース (factory) と semantics を定義する。

life time 管理 (life time management) どのような分散システムでも避けられない故障を扱うことができなくてはならない。過渡的な、内部状態を持つサービスインスタンスを組み込んでいるシステムでは、失敗した操作に対応するサービスや内部状態を回収するためのメカニズムが提供されなくてはならない。例えば、ビデオ会議のセッションの終了には、通信フローを管理するために中継ポイントで生成されているサービスの終了も必要とするであろう。我々は Destroy と SetTerminationTime (必要となるグリッドサービスインタフェース中にある) の2つの標準の操作の定義により、この要求を扱う。これらの操作は、それぞれ、明示された破棄とグリッドサービスインスタンスの life time の管理についてのものである。(Soft state protocols[59,69]は、もし後続の keepalive メッセージのストリームによるリフレッシュがない限り、リモート側で確立された状態の破棄が起こりうることを許す。そのようなプロトコルは、故障からの回復 - 一つのメッセージの消失が回復できない害を引き起こすことがない - と reliable な "discard" プロトコルメッセージが必要とされないため、簡便になるという利点がある。)

告知 (Notification) 動的な分散されたサービス群は、その状態の関知しなくてはならない変化について、お互いに非同期的に告知できなければならない。OGSA は、そのような告知について、subscription したり (NotificationSource) や配信したり (NotificationSink) するための、共通の抽象化とサービスインタフェースを定義しており、単純なサービスの組み合わせによって構成されるサービスは標準的な方法で告知 (例えばエラーなど) を扱うことができるようになっている。NotificationSource インタフェースはサービスデータに統合されており、告知要求はそれに続く "push" モードのサービスデータの配信の要求として表される。(これらのイ

インタフェースによって供給される機能をイベントサービスと呼ぶことができる[10]が、あえてその言い方をさせている。)

その他のインタフェース (Other interfaces) 我々は認証や、ポリシー管理、大規模になりうるグリッドサービスインスタンス群のモニタリングや管理などの課題に取り組むために、近い将来、付加的な標準的なインタフェースを定義しなくてはならないと考えている。

4.3 ホスティング環境の役割

OGSA はグリッドサービスインスタンスの semantics を定めている：すなわち、インスタンスはどのように作られるのか、名前付けられるのか、life time はどうやって決まるのか、どのように通信するのか、などを定義している。しかしながら、OGSA は基本的な不振舞いに関しては規定しているが、サービスが何をするものなのか、あるいはサービスがどのように実行されるのかについてはなにも規定していない。言い換えると、OGSA は実装するプログラミングモデル、プログラミング言語、実装ツール、もしくは実行環境に関しては何も述べてはいない。

実際、グリッドサービスは特定の実行環境、すなわちホスティング環境 (hosting environment) 内でインスタンス化される。特定のホスティング環境は実装プログラミングモデル、プログラミング言語、開発ツール、そしてデバッグツールを定めるだけでなく、どのようにグリッドサービスの実装がグリッドサービス semantics における制約を満たすのかについて定義する。

今日の e-science グリッドのアプリケーションは一般的にはこれらのホスティング環境としてネイティブ なオペレーティングシステムのプロセスによるものである。例えば、新しいサービスインスタンスを作ることは新しいプロセスの生成を伴う。そのような環境では、サービス自体が C,C++, Java, もしくは Fortran のような様々な言語で実装されているかもしれない。グリッド semantics はサービスの一部として直接実装されているかもしれないし、もしくはリンクライブラリを経由して提供されているかもしれない。典型的な semantics はオペレーティングシステムによって供給されるサービスを越えて、外部のサービス経由では提供されることはない。したがって、例えば、life time を管理する機能は必要とあればアプリケーション自身の内部で扱わなくてはならない。

一方、Web サービスは J2EE, Websphere, .NET, そして Sun One のようなより洗練された コンテナ もしくは コンポーネントベース のホスティング環境上で実装することができる。そのような環境では、環境が定義する標準インタフェースに従ったコンポーネントをインスタンス化し、複雑なアプリケーションを構築するために組み合わせるためのフレームワークが定義されている。ネイティブなホスティング環境によって提供される低レベルの機能に比べて、container/component を基本とするホスティング環境は優れたプログラマビリティ、管理可能性、柔軟性、安全性を提供してくれる。それゆえ、component/container ベースのホスティング環境は e-business サービスの構築に広く利用されているのを見ることができる。OGSA のコンテキストでは、container(ホスティング環境)は、サポートするサービスは、グリッドサービス semantics に従うことを保障する基本的な役割をし、したがって OGSA は container/component インタフェースの変更もしくは追加のきっかけになるかもしれない。

サービスの semantics を定義することにより、OGSA はいくつかのホスティング環境に独立した形でサービス間のやり取りを指定する。しかしながら、ここまでの議論で明らかになって

いるように、全てのホスティング環境が保持しなければならない基本的な特性の仕様を定め、グローバルなグリッド環境への内部インタフェースを定めることにより、グリッドサービスの良い実装が可能になる。そして、これらの特性は様々な実装技術(例: J2EE、共有ライブラリ)により実現されるであろう。

ホスティング環境特性の詳細な議論はこの論文の範囲を越えている。しかしながら、我々はホスティング環境が、Grid-wide 名前づけ(例えば グリッドサービスハンドル)から特定の実装における entities(C pointers, Java object references などに)へのマッピング; グリッド起動やイベントの告知を特定の実装での動作(events, procedure calls)への振り分け; ネットワーク転送のためのプロトコル処理やデータのフォーマット; グリッドサービスインスタンスのライフタイム管理; サービス間の認証など、の諸課題について、扱うことができるのではないかと考えている。

4.4 VO 構築のための OGSA メカニズムの利用

アプリケーションやユーザは一時的なサービスを作り、利用可能なサービスの属性の discovery や決定ができなければならない。OGSA Factory, Registry, GridService, そして HandleMap インタフェースは、一時的なサービスインスタンスを作り出すことやサービスの discovery、VO と関連するサービスインスタンスの特性づけをサポートしている。(実際、レジストリサービスとは、登録のための Registry インタフェースと、discovery のために、適当なサービスデータで GridService インタフェースの FindServiceData 操作をサポートしているサービスインスタンスであり、VO と関連する定義されているサービスセットを定義する) これらのインタフェースは様々なサービス構造を構成するのに用いることができる。図 2 に示し、以下で説明する。

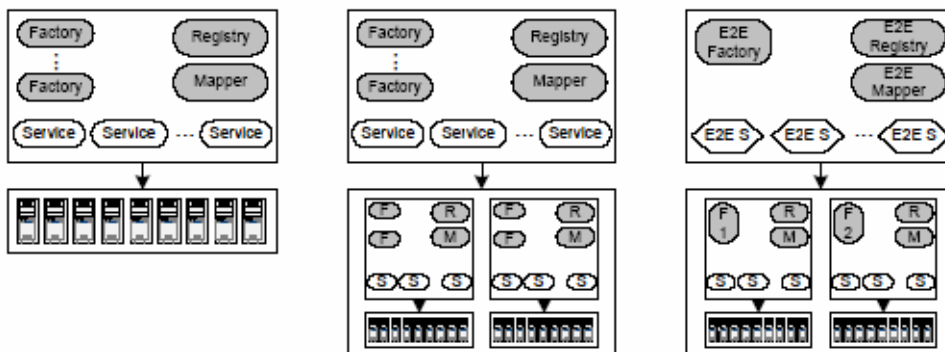


図 2 : 3つの異なる VO 構造。左から、単純なホスティング環境、仮想的なホスティング環境、集合操作

単純なホスティング環境 (Simple hosting environment) 単純な実行環境は、単一の管理上のドメイン内部に設置される資源やサービス運用のためのネイティブな機能をサポートしている資源のセットである; 例として、J2EE アプリケーションサーバー、Microsoft .NET システム、もしくは Linux cluster がある。OGSA では、そのような環境のためのユーザインタフェースは典型的には、レジストリと1つ以上の Factory, handleMap サービスから、構成される

だろう。それぞれの factory は、レジストリに登録されており、クライアントが利用可能な factory を discovery できるようになっている。クライアントの新しいインスタンスを作り出すための要求を factory が受け取った時、factory はそのホスティング環境での機能を起動し、新しいインスタンスを作り、それをハンドルを割り当て、レジストリにインスタンスを登録し、そして、その handle を handleMap サービスで利用可能にする。これら様々なサービスの実装はローカルな操作へ直接にマッピングされる。

仮想的なホスティング環境 (Virtual hosting environment) より複雑な環境では、VO に対する資源は、ヘテロで地理的に分散された "ホスティング環境" に跨っているであろう。(例えば、図 2 では、これらの資源は 2 つの単純なホスティング環境に跨っている。) それにもかかわらず、この "仮想的なホスティング環境" (おそらく B2B パートナースHIPに関連する資源のセットに相当する) は、今述べたホスティング環境に用いられていたのとまったく同じインタフェース経由で、クライアントからアクセスできるようにする。我々は作成要求を低レベル factory に委任する、一つ以上の "高レベル" な factory を作る。同様に、我々は高レベル factory とそれらが作成するサービスインスタンスについて知っている高レベルのレジストリを作る。もちろん、いくつかの VO サービスの使用を統括する VO-specific ポリシーも同様である。したがって、クライアントは factory を見つけることや VO に関連した他のサービスインスタンスを見つけるために、VO レジストリを使うことができ、これらのサービスインスタンスに直接通信するために、レジストリから返された handle を使うことができるようになる。高レベルな factory やレジストリは標準的なインタフェースを実装しており、したがって、ユーザからみれば、他の factory やレジストリからと見分けのつかないものとなる。

ここに示すのは、以前の例のように、レジストリ handle は VO によって管理されているサービスセットに対するグローバルなユニークな名前として用いることができる。資源運用ポリシーはホスティング VO サービスプラットフォーム上で定義され、従わなくてはならない。また、このユニークな名前によって VO が指定される。

集団的操作 (Collective operations) 我々はまた、より洗練され、仮想的で、"集団的" もしくは "end-to-end" のサービスを VO 参加者に提供する「仮想的なホスティング」も構築することができる。この場合、レジストリは高レベルなサービスインスタンスを生成する factory の追跡し、告知する。そのようなインスタンスは低レベルな factory に複数のサービスインスタンスを作成を依頼し、これらの複数の低レベルなサービスインスタンスの動作を単一で高レベルサービスインスタンスになるように組み合わせることにより実装される。

これら 3 つの例、そしてこれまでの議論により、どうやってグリッドサービスメカニズムが仮想な複数の組織境界間もしくは、商用の内部 IT インフラにおいて、分散された資源を統合するのに使用することができるかを示した。どちらの場合でも、適当な discovery サービスに登録されたグリッドサービスの集まりは、分散した資源のプールに跨って相互の QoS 提供する機能をサポートすることができる。アプリケーションやミドルウェアは、リモートとローカルによらず、局所的に最適化されたフローを持つヘテロなプラットフォーム間で、分散資源の運用のためにこれらのサービスを利用することができる。

ネイティブなプラットフォームの資源や API にマッピングされるグリッドサービスの実装に

より、基本的なプラットフォームコンポーネントから、いま丁度説明したような高レベルのグリッドサービスのシームレスな統合が可能になる。さらに、複数の VO に対応するサービスセットは同じ下位の物理的な資源にマッピングされるが、あるレベルで論理的には1つのサービスとして表現されるが低いレベルでは物理資源システムを共有するサービスにより行われる。

5 応用例

図3にデータマイニングを一連の動きを追って、次に示すステージを説明し、基本的な遠隔サービス起動、life time の管理、そして通知機能の働きについて明らかにする。

- 1 . 環境は始めは4つの単純なホスティング環境（左から右へ）から成る：そのうちの1つはユーザアプリケーションを実行し、もう一つは計算と記憶資源（これは2つの factory サービスをサポートしており、そのうちの1つは記憶装置の予約のためのもの、もう一つは名前づけサービスを作り出すものである）をカプセル化する。そして、のこりの2つはデータベースサービスをカプセル化するものである。” R ” はローカルレジストリサービスを表している；追加された VO レジストリサービスはおそらく全ての図にあるサービスの位置についての情報を提供する。
- 2 . ユーザアプリケーションが2番目のホスティング環境の中の2つの factory 上に要求する ” create Grid service ” を起動し、代わってデータマイニング演算を実行する ” data mining service ” の生成と、計算により利用される一時的な記憶の配置を要求する。それぞれの要求も、ユーザと関連した factory (factory のサービス内容に記載されている認証メカニズムを使用する) の相互の認証が行われ、要求の確認が行われる。それぞれの要求が成功し、結果としてある初期 life time を持つグリッドサービスインスタンスが作られる。新しいデータマイニングサービスにはまた、ユーザに代わってさらなるリモート操作を実行することができるようにするための委譲された proxy credential が与えられる。
- 3 . 新たに作られたデータマイニングサービスは、その proxy credential を使って、2つのデータベースサービスからデータの取得要求を開始し、その中間結果をローカル記憶装置に収納する。データマイニングサービスはまた、通知メカニズムも使用し、アプリケーションに定期的にその状態についての更新を知らせる機能を提供する。そのうち、ユーザアプリケーションはそれが作った2つのグリッドサービスへ、定期的な ” keepalive ” 要求を生成する。
- 4 . ユーザアプリケーションが何らかの理由で機能しなくなる。データマイニング計算はそれまで継続しているが、しかしその結果に関する party はないので、これ以上 keepalive メッセージは生成されない。
- 5 . (図にはない) アプリケーションの障害が原因で、keepalive メッセージが消滅し、したがって、2つのグリッドサービスインスタンスがそのうちタイムアウトし終了し、消費していた記憶装置や計算資源が解放される。

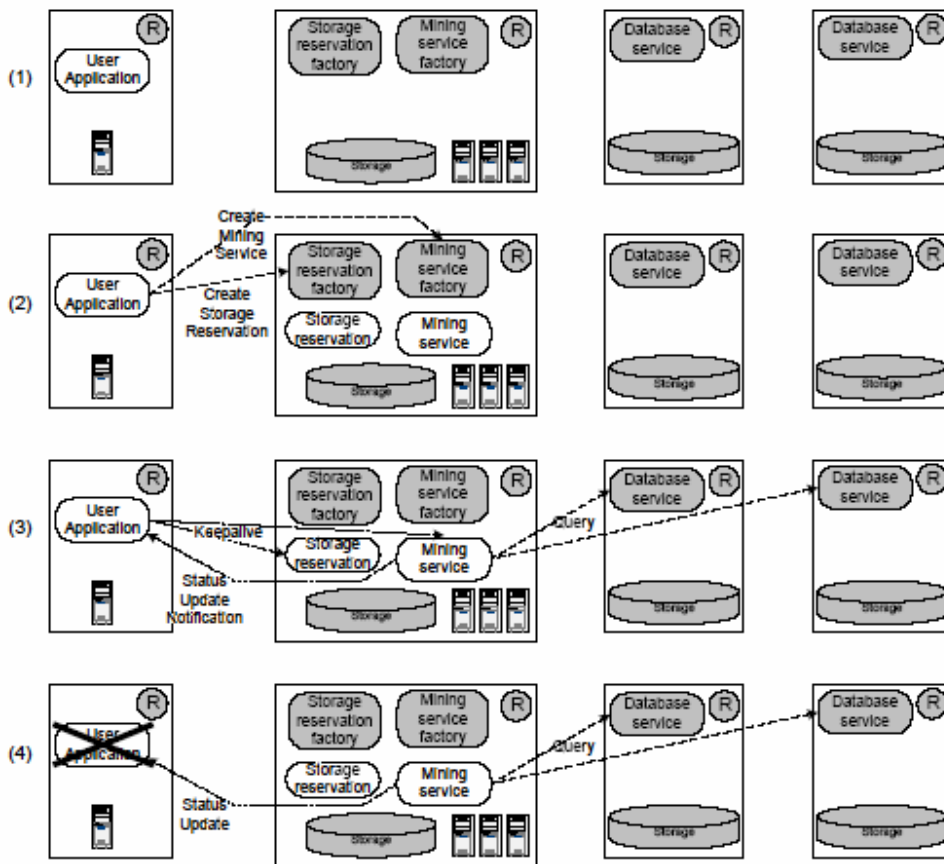


図 3 : グリッドサービスの例

6 . 技術の詳細

ここでは、グリッドサービスの抽象化、関係するインタフェース、規約についての詳細について説明する。

6.1 OGSA のサービスモデル

OGSA の基本的な前提は全てがサービスとして表されることである。サービスとは、メッセージの交換によりその機能を提供する、ネットワークによって可能になる entity である。計算資源、記憶装置資源、ネットワーク、プログラム、データベースなどは全てサービスである。このようなサービス指向のモデルを取り入れることは、環境中のすべての要素が、仮想化されるということの意味する。

より具体的には、OGSA は全てを Grid サービスとして表現する。Grid サービスとは、規約に従い、life time 管理などのための標準的なインタフェースをサポートした web サービスである。この一貫したコアのインタフェースにより、すべてのグリッドサービスが実装されるが、

これらのインターフェースは抽象化の各レイヤで同じように扱うことができる、階層化された、高次のサービスの構築を可能にする。

Grid サービスは彼らの提供する機能により特徴づけられる。Grid サービスでは一つ以上のインタフェースを実装し、それぞれのインタフェースは定義されたメッセージ列を交換することにより起動される一連の操作を定義する。Grid サービスのインタフェースは WSDL の portTypes に対応する。Grid サービスのサポートされる portType の集合は、versioning に関連する付加的な情報と共に、OGSA により定義された WSDL の拡張要素であるグリッドサービスの ServiceType において指定される。

Grid サービスは、サービスの life time において内部状態を保持できる。状態があることにより、あるサービスのインスタンスが同じインタフェースを提供する他のものと区別される。我々は、Grid サービスインスタンスという用語を、Grid サービスの特定のインスタンスに用いる。

サービスのインタフェースと関連するプロトコルの binding は、例えば信頼性などに関する、配送の semantics を定義することができる。サービスはメッセージを交換することで互いに協調動作する。しかし、その構成要素に障害が起こる可能性がある分散システムでは、送られたメッセージが届いていることを保障されない。内部状態があることにより、たとえリトライなどの障害回復機構が使われていたとしても、サービスがメッセージを受け取っていたか、あるいはまったく受け取っていなかったかを保障できることが重要と成りうる。このような場合、一回だけ配送を保障するプロトコルまたは類似の semantics のプロトコルが望ましいと考えるかもしれない。他の要求されることが多いプロトコル binding の振る舞いとしては、通信中の相互の認証がある。

OGSA のサービスは動的に生成、破棄することができる。サービスは明示的に破棄されることもあれば、OS のクラッシュやネットワークの分離などのシステムの障害によって、破棄又はアクセス不能になることもある。いくつかのインタフェースは life time の管理について定義されている。

Grid サービスは動的で状態を持つため、ある動的に作成されたサービスを他のサービスと区別する方法が必要となる。したがって、全ての Grid サービスインスタンスは、グローバルでユニークな名前、GSH (Grid service handle) を割り当てられる。これは、ある Grid サービスインスタンスを、それまであった、今ある、もしくは、未来において存在する Grid サービスインスタンスと区別するものである。(もしある Grid サービスが故障し、その状態を保持されるように再スタートした場合は、それは本質的に同じインスタンスであり、同じ GSH を使うことができる)。

Grid サービスは life time 中にも、例えば、新しいバージョンのプロトコルをサポートしたり、他のプロトコルを加えたり、アップグレードされうる。そのため、GSH は、ネットワークアドレスやサポートされてる binding プロトコルなどの、プロトコルやインスタンスなどに特有の情報を保持しない。かわりに、そのような情報は、あるサービスとやりとりする時に必要となるインスタンスに特有なその他の情報と一緒に、GSR (Grid Service Reference) 呼ばれるものにカプセル化される。不変である、GSH と違って、Grid サービスインスタンスの GSR はサービスの life time 中に変化しうる。GSR は、明確な期限切れの時間を持ち、サービスの life time 中に無効になることもある。OGSA は更新された GSR を取得するために、以下で述べる mapping 機構を定義している。

生存期限が終了した GSR を使った結果は未定義である。ここで注意したいのは、有効な GSR を持っていることが、Grid サービスインスタンスへのアクセスを保障するものではない。ローカルポリシーやアクセス制御制限（例えば、一度の最大リクエスト数、などの）は、リクエストへのサービスを禁止することがある。さらに、参照された Grid サービスインスタンスは失敗していることがあり、GSR の使用が妨げられる。

OGSA のすべては Grid サービスであるので、Grid サービスやハンドル、OGSA モデルを定義する参照の抽象化を扱うための Grid サービスがなくてはならない。ある一連のサービスを定義することにより、それに対応するある OGSA のサービスモデルを提供するであろう。そこでより柔軟なアプローチとり、サービスモデルの抽象化を扱うため OGSA の基本的なインタフェイス（WSDL や portType など）の集合を定義する。これらのインタフェイスは様々な方法で組み合わせて、広範囲の Grid サービスを作り出す。表 1 は Grid サービスインタフェイスの名前と定義について述べている。ただし、全ての Grid サービスにより、Grid サービスインタフェイスだけはサポートされていなくてはならない。

6.2 一時的なサービスの作成 : Factories

OGSA は、新しい Grid サービスインスタンスを生成するインタフェイスを実装する、Grid サービスのクラスを定義している。それは Factory インタフェイスと呼び、このインタフェイスを実装するサービスは factory と呼ぶ。Factory インタフェイスの CreateService 操作は要求された Grid サービスを生成し、新しいサービスインスタンスの GSH と初期 GSR を返す。

Factory インタフェイスはどうやってサービスインスタンスが生成されるのか指定するものではない。一つの共通のシナリオとしては、新しいサービスインスタンスを生成する（そして、それを管理する）ための標準的な機構を提供するホスティング環境（.NET や J2EE など）で、Factory インタフェイスが実装される。ホスティング環境は、どのようにサービスが実装されるか（言語などを）について定義することができるが、OGSA ではサービスを要求する側には透過的であり、要求側は factory インタフェイスしか見ない。また、リクエストを他の factory に委譲することによりサービスを生成する、“高レベルな”factory を生成することができる（4.4 を参照）。例えば web サービスの環境で、適切な web サービス factory に暇なコンピュータ上での “web サービス” のサービスのインスタンスを作成するように依頼することにより、新しいコンピュータをアクティブなプールに統合することができるであろう。

6.3 サービスの life time 管理

一時的なサービスインスタンスというものを導入した結果、サービスの life time を決定しなくてはならないという問題を扱わなくてはならない。つまり、関連する資源を回収できるように、いつになったらサービスを終了することができるか、又は終了するべきか消してもいいか、ということを決めなくてはならない。通常の動作条件では、一時的サービスインスタンスはある特定のタスクを行うために作られ、そのタスクの完了により終了する、サービスを要求した側からの明示的な要求、又はサービスを要求した側により作られたサービスからの要求により終了する。しかしながら、分散システムではコンポーネントが故障しているときもあればメッセージがなんらかの原因で到着しない時もある。その結果、サービスが想定している明示的な終了要求を受け取ることができないことがあり、資源をずっと消費するしてしまうことになる。

OGSA は soft state のアプローチによりこのような問題に対処する[23,69]。すなわち、Grid サービスインスタンスはその life time を指定し、生成される。初期の life time は、クライアント又はクライアントにより作られた Grid サービスによって明示的に延長することができる（それはサービスポリシーの問題だが）。もしクライアントから何のアクションも受け取らずに期限がきた場合には、ホスト環境又はサービスインスタンス自身が、自由に、サービスインスタンスを終了して関連リソースを開放することができる。

このようなアプローチの Grid サービスの life time 管理は以下の 2 つの性質をもつことが望ましい。

- クライアントはいつ Grid サービスインスタンスが終了するかについて、知っているかあるいは決めることができる。それを知ることにより、たとえシステム障害（例えばサービス、ネットワーク、クライアントの障害など）に直面しても、クライアントが、確実にいつサービスインスタンスが終了し、資源が回収されたかを判断することができるようになる。クライアントはサービスインスタンスからの最終状態を要求したり、サービスの life time の延長リクエストをするために、サービスが正確にどのくらい期間なのかを知る。さらに、もしシステム障害が起こったなら、終了時間以降はサービスにコンタクトしようとし続ける必要がないことを知ることができるし、（他のクライアントがセ life time を延長に成功しない限り）サービスに関係する全てのリソースはその時間以降はすべて開放されることも知ることができる。すなわち、life time 管理により、サービスインスタンスの life time の semantics を明確に定義することにより、ロバストな終了と故障検知が可能になる。
- ホスティング環境は、たとえ制御できないシステム障害に直面したとしても、リソース消費に制限されることを保障する。サービスの終了時間がきたならばホスティング環境の全ての関連するリソースを回収することができる。

我々は、グリッドサービスのインタフェース内で SetTerminationTime 操作により、Soft state の life time 管理の実装をする。この操作は、新しいサービスインスタンスの初期のセ life time について交渉したり、life time の延長の要求、life time が切れた時のサービスインスタンスの回収をするための操作を定義するものである。

初期の life time の交渉 factory を通して新しい Grid サービスインスタンスの生成を要求するときには、クライアントは初期の life time の最大と最小の許容範囲を提示する。Factory は初期 life time を選び、クライアントに返す。

life time の延長の要求 クライアントは life time の延長の要求を、Grid サービスインスタンスに対する SetTerminationTime メッセージで行う。そのメッセージには許容できる最大と最小の新しい life time を指定する。サービスインスタンスは新しい life time を選び、クライアントに返す。ただし、keepalive メッセージは実際、冪等(idempotent)である。つまり、サービスインスタンスの life time が期限切れで、多くのメッセージが行方不明になり限り、中間のメッセージの順序が変わろうが失われても、一連のリクエストの結果は同じとなる。

Keepalive メッセージの周期は、サービスインスタンスの交渉により決まった初期の life time と（多分、それ以降の keepalive メッセージでも交渉が行われる）ネットワークの信頼性についての情報により、クライアントにより決定される。初期サイズはオーバーヘッドと情報

流れの間のトレードオフで決定される。

ただし、このような life time 管理のアプローチは重要な自立性 (autonomy) を提供する。クライアントによる life time 延長の要求は必須のものではない。サービスは、このような要求を保障する際には、自分のポリシーを適用することができる。サービスは、クライアントによる life time 延長の要求に対する返信、又はその他の理由により、いつでもその life time の延長するかどうかについて決めることができる。例えばリソースの制限や優先度制御により、リソースを放棄する場合など、サービスインスタンスはそれ自体をいつでもキャンセルできる。それ以降のクライアントのこのサービスへの参照は失敗するであろう。

SetTerminationTime において絶対時間を用いており、Grid サービスの情報要素やセキュリティの credential など一般的に使われているが、それは十分に同調しているグローバルクロックの存在を意味している。NTP (Network Time Protocol) は、クロックの同期についての標準な機構を提供しており、ほぼクロックを最大数十ミリ秒以内で同期させることができる。その精度は、life time 管理の目的に対して十分に適切である。ただし、これらの記述が、event の順序づけのために必要事項だと意味しているのではないことだ、尤もこのような仕組みが将来のバージョンでは導入されることを期待している。

6.4 Handle と Reference の管理

上で述べたように、factory の要求の結果として、GSH と GSR が得られる。GSH は作られたグリッドサービスインスタンスの照合の永続性を保障するが、GSR は有限の life time とともに作られ、サービスの life time 中に变化しうる。この戦略は、Grid サービスプロバイダーから見れば、柔軟性が大きく利点となるが、サービス生成操作により返された GSR の有効期限が切れてしまったら、有効な GSR をどうやって得るかということが問題になる。本質的にはこれは bootstrapping の問題である。すなわち、それに対する GSH のみで、どうやってグリッドサービスとコネクションを確立するのかということである。ここでは、2002 年 6 月の Grid サービスの仕様において、この問題がどのように取り扱われたかを述べるが、しかしここで注意しないといけないのは、最低でも複数の handle 表示と handle mapping サービスにおいては、この部分の仕様はおそらく将来的に改善される点であると思われる。

OGSA でのアプローチは handle と reference を mapping するインターフェイス(handle Map)を定義することである。このインターフェイスにより提供される操作は、GSH を受け取り、有効な GSR を返す。この操作は、アクセスコントロールされることがあり、mapping 要求は拒否されることがある。Handle Map の実装では、どのグリッドサービスインスタンスが実際に存在するか把握するためのものであり、すでに消去されたと分かっているインスタンスへの参照を返すものではない。しかしながら、有効な GSR を所持しているからといって、Grid サービスインスタンスに連絡できるとは限らない。サービスはすでに機能しないかもしれないし、GSR が作成されてから用いられるまでの間に明示的に消去されていたかもしれない。(明らかに、もしサービスの終了がスケジュールされていたなら、GSR の life time にそれを表現するようにすべきであるが、しかしそれは必須条件ではない。

Handle Map インタフェイスを導入することにより任意のサービスについて GSR を習得する上での、一般的な問題を 2 つのもっと特化された部分的な問題に分けて考える。

- 1) 指定された GSH に対するマッピングを含む、handle Map サービスを識別することと、

2) 所望の GSR を得るために Handle Map にコンタクトすること、である

それでは、この二つの問題について考えてみる。いつでも GSH から GSR に確実にマッピングできることを保障するために、全ての Grid サービスインスタンスが、最低 1 つの handle Map に登録されていることを必要とする。その 1 つの handleMap をここでは home handleMap と呼ぶ。GSH に home handleMap の ID を含むように構成することで、拡張可能かつ簡単に、どの handleMap にコンタクトすれば特定の GSH に対する GSR を得ることができるか、判断できる。したがって、ユニークな名前がローカルに決定でき、従って中央的な名前割り当てサービスに関連するスケラビリティの問題を避けることができる - しかしながら、DNS(Domain Name System) [52] に依存することになるが。ただし、GSH マッピングは、他の handle Mapping にあってもよい。しかしながら、すべての GSH は厳密に一つの home handleMap を持たなければいけない。

どうやって GSH 中の home handleMap を識別するのか。HandleMap インタフェイスを実装する、どのようなサービスでもグリッドサービスの一つであり、それらは全て GSH を持つ。しかし、GSH を構成するときに、この名前を使えば、handleMap サービスの GSH から、GSR を取得しようとするのと同じ立場に戻るることとなる。この bootstrapping の問題を解決するために、ある handle Map の GSR を handle Map がなしでも、得る方法が必要となる！すべての home handleMap が URL により識別できるようにし、知られたプロトコル、すなわち、HTTP (又は HTTPS) にバインドした bootstrapping 操作をサポートすることにより、これを行う。それゆえ、handle Map サービスにコンタクトするために何のプロトコルを使うかを記述しているための GSR を使うかわりに、home handle Map を指す URL について、HTTP GET 操作を用い、handle Map への GSR が、WSDL 形式で、返される。

ここで注意は、handleMap と factory インタフェイスを実装するサービスの間にある関係が存在することである。特に、factory の要求により返された GSH は、必ず home handleMap の URL を含まなければならない、GSH/GSR のマッピングは handleMap サービスに必ず入力され更新されなければならない。Factory の実装においては、どのサービスを home handleMap として用いるか決めなければならない。実際、単一のサービスは、factory と handle Map インタフェイスの両方を実装しうる。

現在の GGF の仕事は、現在の handle と resolver を簡単になるように、この Grid サービスのコンポーネントが、違う形式の handle と mapper/resolver を持つことができるように見直すことである。

6.5 サービスデータとサービス discovery

それぞれの Grid サービスインスタンスに対して、サービスデータの集合、すなわち、サービスデータ要素としてカプセル化された XML 要素の集まりがある。それぞれの要素のパッケージには、Grid サービスインスタンス固有の名前、タイプ、受信者が life time 管理をするための time-to-live 情報、が含まれている。

必須の GridService インタフェイスは、サービスデータを検索し取り出すための、標準的な WSDL 操作 FindServiceData を定義する。この操作は、単純な名前による検索言語を必要とし、用いられている検索言語に対して拡張可能である。その言語としては、例えば Xquery[20] でもよい。

Grid サービスの仕様は、それぞれの Grid サービスインタフェースに対して、そのインタフェースをサポートする任意の Grid サービスインスタンスによりサポートされなければならない 0 個以上のサービスデータ要素を定義する。従って、Grid サービスインタフェースと対応して、Grid サービスインスタンスがサポートしなくてはならないのは、GSH、GSR、primary key、及び home handleMap などの、Grid サービスインスタンスについての基本的な情報を含む要素の集合である。

Grid サービスインタフェースの FindServiceData 操作の一つの使い道が、サービスの discovery である。これまでの議論では、使いたいサービスに対する GSH をもっていることを前提としてきた。しかし、初めにどうすれば GSH を得ることができるであろうか？それがサービス discovery の本質的な点であり、提供されるインタフェースについての GSH の属性、サービスをすでに提供しているリクエスト数、サービスの負荷などの集合、あるいはいくつまでのリクエストが許可されるかについてのポリシー情報などを指定して、それから、GSH の部分集合を特定するプロセスとして、ここでは定義する。

サービスの discovery をサポートする Grid サービスが registry である。Registry サービスは 2 つのことで定義される。ひとつは registry インタフェースで、その提供する操作と registry サービスを用いて GSH を登録できる。もうひとつは、GHS に登録された情報を含む、それに対応するサービスデータ要素である。したがって、registry インタフェースは GSH を register するために用いられ、GridService インスタンスの FindServiceData 操作は登録された GSH に関する情報を検索するのに用いられる。

registry インタフェースにより、付随するものと考えられる GSH の集合を付加してレジ registry サービスに GSH を登録することができる。MDS-2[24]で行われているように、サービス(又は VO)はこの操作を使って、VO 内の関連のあるグループのみに自分の存在と提供するサービスを知らせることができる。関連のあるグループとは大抵いろいろな形式のサービスの discovery をするサービスを含んでおり、それはサービスを discovery するリクエストに効果的に答えるためにサービス情報を収集し構築する。OGSA のほかの状態を持つインタフェースと同じで、GSH の登録は soft-state な操作で、周期的にリフレッシュされなくてはならない。したがって、discovery サービスは自然に動的なサービスの可用性を扱うことができることになる。

ただし、GHS と関係のある属性の仕様は、GridService インタフェースを実装するサービスと GHS の登録を結びつけているわけではない。このことは重要で、なぜなら、属性の値は動的に変わりえるもので、属性値を得るにはいろいろな方法がありえるからである。例えば、GridService インタフェースを実装している他のサービスに問い合わせるなどの方法がある。

6.6 告知(notification)

OGSA の notification のフレームワークは、クライアントが、特定のメッセージによる告知を受け取ることを登録し、(NotificationSource インタフェース)、告知などの非同期な、一方方向の配送(NotificationSink)をサポートするものである。もし特定のサービスが告知メッセージを配送予約をサポートしたい場合、そのサービスは配送予約を管理するための NotificationSource インタフェースをサポートしなければいけない。告知メッセージを受け取りたいサービスは、NotificationSink インタフェースを実装しておかなくてはならず、そのインタフェースはメッセージの配送に使われる。特定のサービスからの告知を開始するには、

配送予約操作を notification source インタフェイスで起動し、告知の配送先になるサービスの GSH を与える。その後、告知メッセージが配送元から配送先に流れ、一方では、配送先から配送元に定期的な keepalive メッセージを送ることにより、配送元にまだ告知を送ってほしいことを伝える。もし信頼性のある配送が必要があるなら、この動作をこのサービスに対する適切なプロトコルバインディングを定義することにより実装できる。

この告知モデルの重要な点は、サービスデータとの密な統合である：配送予約操作は、指定の条件に合うサービスデータを操作後に”push”型で配送されるように要求するものである。（FindServiceData 操作は”pull”モデルである）

このフレームワークは、直接の service-to-service の告知メッセージ配送ならびに、様々な第三者サービスとの統合を可能にする。第三者サービスとして、商業の世界やカスタマーサービスで使われる一般的なメッセージサービスや、配送元に代わりに告知メッセージを、フィルタリングしたり、変形させたり、特に配送したりする特化したサービスがある。告知の semantics はメッセージ配送に使われたプロトコルバインディングの属性による。例えば SOAP/HTTP プロトコル又は直接 UDP バインディングでは、point-to-point で best-effort な notification を提供するであろうし、他のバインディング（たとえば、何らかの固有のメッセージサービスなど）は best-effort の配送よりも良いかもしれない。マルチキャストプロトコルバインディングであれば、複数の受信者をサポートできるであろう。

6.7 変更管理 (Change Management)

Grid サービスの discovery と Change Management をサポートするために、Grid サービスのインタフェイスはグローバルかつユニークな名前が与えられなくてはならない。WSDL においてインタフェイスは、portType により定義され、portType の qname によりグローバルかつユニークな名前を持つ（例えば、WSDL ドキュメントの定義要素の TargetNamespace 属性により定義された XML namespace や portType の要素の name 属性により定義された local な名前）。インタフェイスを変えたり、操作に対して意味的に重要な実装を変えたりして、Grid サービスの定義に変更があった場合には、新しいインタフェイスの名前より反映されなければいけない。（例えば、新しい portType と serviceType）この機能により、クライアントは特定の属性（特定のインタフェイス又は実装の semantics）を持つ GridService に互換性のあるサービスを発見するように要求することができる。

6.8 他のインタフェイス

将来的には一連の manageability 操作をサポートする選択的な (optional な) manageability インタフェイスが定義されるだろう。その操作により潜在的に大きな数の Grid サービスインスタンスを、管理コンソールや自動ツールなどを使って、監視することができる。選択的 (optional) である Concurrency インタフェイスは、並列性機能を提供するであろう。

7 ネットワークプロトコル binding

web サービスのフレームワークは、色々な異なったプロトコルバインディング上でインスタンス化することができる。セキュリティのための TLS 付の SOAP+HTTP はその一つの例であ

るが、他にも可能であり、すでに定義されたものもある。ここでは、OGSA というコンテキストでの問題点について、議論する。

OGSA のコンテキストでネットワークプロトコル binding を選択する場合、以下の 4 つの主要な要請について扱わなくてはならない。

- 信頼性のある転送 前にも述べたように Grid サービスの抽象化において、信頼性のあるサービス起動が必要となることがある。この要求に取り組む一つの方法は、ネットワークプロトコルバインディング (例えば HTTP-R) 内で適切なサポートを組み込むことである。
- 認証と委任 前にも述べたように Grid サービスの抽象化において、リモートサイトへの proxy credential の通信をサポートが必要となることがある。この要求に取り組む一つの方法としては、ネットワークプロトコルバインディング (例えば、proxy credential をサポートするように拡張された TLS 内) で適切なサポートを組み入れることである。
- Ubiquity 分散されたリソースから動的に VO を構築することを可能とするという、Grid の目標は、本質的にサービスの任意のペアは互いにやり取りを可能であることを意味する。
- GSR format GSR では、バインディングに特化したフォーマットを取れることを思い起こして欲しい。一つの可能な GSR format としては WSDL document があり、他にも CORBA IOR がある。

大規模の相互運用可能な OGSA の実装の配置を成功させるためには、Grid サービスの discovery や起動に対して、少ない数の一般的なプロトコル binding の定義を用いたほうがいいだろう。どこでももといえることができるインターネットプロトコルにより原則的にどのような 2 つのエンティティも通信できるようになっているように、このような "InterGrid" プロトコルをどこでも使えるようにすることによって、2 つのサービスが通信することができるようになる。したがって、クライアントは、一つのプロトコルのみ知っていればいいので、特に簡単になる。(ただし、このような基準となるプロトコルが存在している時でも、サービス同士両方がサポートしてさえいれば、違うプロトコルを使うことができる)。いずれにしろ、このような InterGrid プロトコルを定義することできるし、それが広く流通し受け入れられるかはこれからの問題である。ともかく、それらの定義は本稿の範疇外である。

8 高レベルサービス

本稿で述べた抽象化とサービスは、さまざまな高次の Grid サービスを実装するのに使える基本的な部品を提供する。我々は、e-ビジネスや e-サイエンスの広範な要求に取り組む様々なサービスを定義、実装するために、この分野の研究者と密接に作業をしていこうとおもっている。それらは以下のものを含むであろう。

- 分散データ管理サービス このサービスは、分散されたデータに対するアクセスと操作をサポートする [58]。データはデータベースにあってもいいし、ファイルにあってもよい。サービスには、データベースアクセス、データトランスレーション、複製管理 (replica management)、トランザクション、が含まれる

- workflow サービス このサービスは、複数の分散された Grid のリソース上での複数の application の task の協調、実行をサポートする
- 監視 (auditing) サービス このサービスは、使用されたデータを記録し、そのデータの保管の安全を確保し、不正行為と侵入検知をするための分析をしたりする。
- インストルメントとモニタリングサービス このサービスは、分散環境で センサーによる discovery をサポートしたり、これらのセンサーからの情報収集と分析したり、異常が観測された時アラートを出したりするサービス。
- 分散計算に対する問題決定サービス このサービスには、dump、trace、イベントにタグ付け、関連付けるログ機構、などを含む。
- セキュリティプロトコルマッピングサービス このサービスにより、分散セキュリティプロトコルを、分散セキュリティ認証とアクセスコントロール機能を持たないプラットフォームリソースマネージャーを持つネイティブなプラットフォームセキュリティサービスに transparent な mapping することができる。

このフレームワークの柔軟性とはこのようなサービスが色々な方法で、実装や構成できることを意味する。例えば、同時に配置し複数の計算資源を使える協調したサービスは、一つサービスインスタンスとしてインスタンス化でき、アプリケーションをライブラリとしてリンクしたり、さらに高次のサービスに組み込むことができる。

現在の Globus Toolkit 内で使われているリソース管理・データ転送・情報サービスのプロトコルを、これらの共通した基盤機構上に再設計することはごく素直なことに見える (図 4 参照)。実際、似た要素を抜き出し共通性を活用したりして、これらのプロトコルのデザインを再構成することができる。その過程で、現在のプロトコルの機能を改善し、一般的なサービスインフラにたどり着くはずである。その過程により、Globus Toolkit 3.0 ができるであろう。

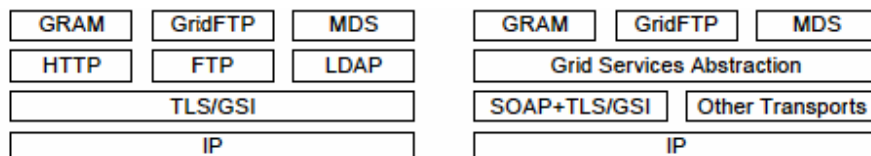


図 4 左が、現在の Globus Toolkit のプロトコル、右が OGSA 機構を利用し再構成したもの

9 . 関連研究

関連性のある先行研究について、特に一時的で状態を持つサービスの安全かつ信頼性を持つ遠隔生成および管理に関する関連研究について焦点を当て、述べる。

第 3.1 章でも述べたように、OGSA の多くの仕組みは、Globus Toolkit v2.0 に基づいている。具体的には、factory(GRAM gatekeeper [25]) registry(GRAM reporter [25] と MDS-2[24]) soft state 登録の利用 (MDS-2[24]) 委譲を利用した安全遠隔起動 (GSI[33]) 信頼性の高い遠隔起動 (GRAM[25]) が挙げられる。Globus Toolkit との主な違いはこれらの仕組みがどのように統合されているかということに関連している。OGSA では、例えば、共通の告知メカニ

ズムをサービス登録とサービス状態に使用するように、主要なデザイン要素を再構成をしている。

OGSA は、それぞれのグリッドサービスインスタンスがシステム内のほかのインスタンス区別される別々のアイデンティティをもっており、各インスタンスは型特有な操作を通じて明らかになる動作とつなぎ合わされた状態により特徴づけることができるという点で、分散オブジェクトシステムと見なすことができる [21]。この点で、OGSA は、先に開発された Eden[7], Augus[48], CORBA[1], SOS[60], Spring[51], Globe[62], Mentat[42], Legion[41, 43]などのシステムのアイデアを利用している。CORBA とは対照的に、OGSA は Web サービスのような安全な相互運用の問題に取り組み、より豊富なインタフェース定義言語を提供している。Grid コンピューティングでは、Legion グループがオブジェクトモデル利用の普及を促進した。一部の OGSA と Legion の構成は似ていると言うことが出来る。具体的には、ファクター (Class object)、handleMap (Binding Agent)、binding におけるタイムアウトが挙げられる。しかし、OGSA は実装の際のオブジェクト技術の利用や、インタフェースレベルにおける継承メカニズムの公開やホスティング技術などの分散オブジェクト技術の主要な問題として扱われる課題に対しては規定していない。

soft state 構造はネットワークエンティティの特定の状態を管理するためにインターネットプロトコル[23, 61, 69]や RMI (“lease” という名で)や Jini [57]で使われている。OGSA では、全てのサービスと情報は soft state 管理できる。分散リファレンスカウンティング[12]よりも簡潔な soft state 技術の方のほうが望ましいと考える。

我々の信頼性のある起動機構は、分散システム分野で先行する研究を元に作成されている Condor[36, 49, 50]で用いられている機構ムから思いついたものである。

第 4.3 章で述べられているように、OGSA のコアのサービスは、持続性の管理、安全性、ライフサイクル管理等によって個々のコンポーネント開発を簡単にするホスティング環境によってサポートされるであろう。ホスティング環境の概念は様々なオペレーティングシステムやオブジェクトシステムに使われている。

Web サービスメカニズムやグリッドコンピューティングのアプリケーションは他の研究により[例：35, 37]研究され、提案されている。それらは最近のワークショップの中で、数多くの関連性のある研究の概要[2]を提供している。Gannon 等[37]は様々な最新の技術のアプリケーションや、e-サイエンスアプリケーションについて議論し、動的にアプリケーションサービスを作成する手段として「アプリケーションファクトリー (WSDL インターフェース)」を提案している。De Roure 等[26]は、「Semantic Grid」Semantic Web[11]の類推により提案し、様々な上級サービスを提議している。サービス提供を中心としたインタフェース、数値ソフトウェアの NetSolve[16,17]や Ninff[55]もまた関連した研究である。

サンマイクロシステムの JXTA システム[3]は Grids が直面したいいくつかの大切な課題に取り組んでいる。その中には、仮想組織 JXTA が「peer group」と呼んでいる - の discovery が含まれている。我々は、この抽象概念が OGSA フレームワークの範囲内で実装可能であると信じている。

これらは、Globus Toolkit メカニズムの上に実装されている分散高性能コンピューティング [8, 13, 68]構成モデルと関連付けられることがあると考えている。

10.まとめ

我々は、標準的なインタフェースと規約により、内部状態を持つ (stateful) 一時的な (transient) サービスの生成、終了、管理を、動的で管理可能な life time を持つ、名前づけられた、管理された entity としてサポートする OGSA について定義した。

OGSA では、全てが Grid サービスとして表される。すなわち、Grid サービスとは、life time 管理、特徴の discovery、告知などのための、幾つかの (WSDL で表記された) 規約に適合した (潜在的に一時的な) サービスである。Grid サービスの実装は、現存する IT インフラとのインテグレーションのために、既存のプラットフォーム設備を対象とする。Grid サービスを、作成・登録・発見する標準的なインタフェースは、色々な形式の VO 構造を作成するために、構成することができる。

このようなサービス指向のモデルの利点は以下のようなものがある。環境の全ての構成要素は仮想化される。そのインタフェースにより全ての Grid サービスが実装されるような一貫性のあるインタフェースのコアな集合を提供することで、抽象的なレイヤ間に跨って一貫した方法で扱うことができる、階層的で高次のサービスを構築を支援する。仮想化することにより、また、複数の論理的な計算資源インスタンスを同じ物理的な資源にマッピングし、実装に関係なくサービスを構築し、低次のリソースからなる VO 内の資源管理ができる。Grid サービスの仮想化とは、共通のサービスの semantics の振る舞いを既存のプラットフォーム設備にシームレスにマッピングすることを可能にするものである。

OGSA の開発は、Globus Toolkit 2.0 の自然な進化を象徴するものである。Globus Toolkit には、factory・登録・信頼性のある安全な起動などの、鍵となる概念は存在するが、ここで論議したことよりは、一般性や柔軟性で劣り、均一性のある Interface definition language の恩恵もない。実際、OGSA では、例えば、共通の告知機構がサービスの登録やサービス状態に使えるように、鍵となるデザイン要素を再設計している。OGSA ではそれらの要素をより抽象化することにより、どのようなレベルの VO にも適用できる。Globus Toolkit は基本的なオープンソース OGSA 実装を提供する予定である。Globus Toolkit 3.0 は www.globus.org/ogsa にあるように、WSDL はもちろん、既存の Globus API をサポートしている。

OGSA の開発はまた、web サービスの自然な進化も意味している。一時的で状態を持つサービスインスタンスを、現存する web サービス技術と統合することにより、現在の技術を少し拡張するだけで、OGSA は web サービスのフレームワークの力を飛躍的に伸ばす。

Acknowledgments

We are pleased to acknowledge the many contributions to the Open Grid Services Architecture of Karl Czajkowski, Jeffrey Frey, and Steve Graham. We are also grateful to numerous colleagues for discussions on the topics covered here and/or for helpful comments on versions of this article, in particular Malcolm Atkinson, Brian Carpenter, David De Roure, Andrew Grimshaw, Marty Humphrey, Keith Jackson, Bill Johnston, Kate Keahey, Gregor von Laszewski, Lee Liming, Miron Livny, Norman Paton, Jean-Pierre Prost, Thomas Sandholm, Peter Vanderbilt, and Von Welch.

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the National Science Foundation; by the NASA

Information Power Grid program; and by IBM.

Bibliography

1. Common Object Request Broker: Architecture and Specification, Revision 2.2. Object Management Group Document 96.03.04, 1998.
2. Grid Web Services Workshop. 2001, <https://gridport.npaci.edu/workshop/webserv01/agenda.html>.
3. JXTA. www.jxta.org.
4. Simple Object Access Protocol (SOAP) 1.1. W3C, Note 8, 2000.
5. UDDI: Universal Description, Discovery and Integration. www.uddi.org.
6. Web Services Flow Language. www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
7. Almes, G.T., Black, A.P., Lazowska, E.D. and Noe, J.D. The Eden System: A Technical Review. *IEEE Transactions on Software Engineering*, SE-11 (1). 43--59. 1985.
8. Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L. and Parker, S. Toward a Common Component Architecture for High Performance Scientific Computing. In Proc. 8th IEEE Symp. on High Performance Distributed Computing, 1999.
9. Bal, H.E., Steiner, J.G. and Tanenbaum, A.S. Programming Languages for Distributed Computing Systems. *ACM Computing Surveys*, 21 (3). 261--322. 1989.
10. Barrett, D.J., Clarke, L.A., Tarr, P.L. and Wise, A.E. A Framework for Event-based Software Integration. *ACM Transactions on Software Engineering and Methodology*, 5 (4). 378-421. 1996.
11. Berners-Lee, T., Hendler, J. and Lassila, O. The Semantic Web. *Scientific American*. 2001.
12. Bevan, D.I., Distributed Garbage Collection Using Reference Counting. In PARLE Parallel Architectures and Languages Europe, (1987), Springer Verlag, LNCS 259, 176-187
13. Bramley, R., Gannon, D., Stuckey, T., Villacis, J., Balasubramanian, J., E. Akman, Breg, F., Diwan, S. and Govindaraju, M. Component Architectures for Distributed Scientific Problem Solving. *IEEE Computational Science and Engineering*, 5 (2). 50-63. 1998.
14. Bray, T., Paoli, J. and Sperberg-McQueen, C.M. The Extensible Markup Language (XML) 1.0. 1998.
15. Brittenham, P. An Overview of the Web Services Inspection Language. 2001, www.ibm.com/developerworks/webservices/library/ws-wslover.
16. Casanova, H. and Dongarra, J. NetSolve: A Network Server for Solving Computational Science Problems. *International Journal of Supercomputer Applications and High Performance Computing*, 11 (3). 212-223. 1997.
17. Casanova, H., Dongarra, J., Johnson, C. and Miller, M. Application-Specific Tools. In Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 159-180.
18. Catlett, C. In Search of Gigabit Applications. *IEEE Communications Magazine* (April). 42-51. 1992.
19. Catlett, C. and Smarr, L. Metacomputing. *Communications of the ACM*, 35 (6). 44--52. 1992.
20. Chamberlin, D. Xquery 1.0: An XML Query Language. W3C Working Draft 07, 2001.
21. Chin, R.S. and Chanson, S.T. Distributed Object-based Programming Systems. *ACM Computing Surveys*, 23 (1). 91-124. 1991.
22. Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S. Web Services Description Language (WSDL) 1.1. W3C, Note 15, 2001, www.w3.org/TR/wsdl.
23. Clark, D.D., The Design Philosophy of the DARPA Internet Protocols. In SIGCOMM Symposium on Communications Architectures and Protocols, (1988), ACM Press, 106-114
24. Czajkowski, K., Fitzgerald, S., Foster, I. and Kesselman, C., Grid Information Services for Distributed Resource Sharing. In 10th IEEE International Symposium on High Performance Distributed Computing, (2001), IEEE Press, 181-184

25. Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W. and Tuecke, S. A Resource Management Architecture for Metacomputing Systems. In 4th Workshop on Job Scheduling Strategies for Parallel Processing, Springer-Verlag, 1998, 62-82.
26. De Roure, D., Jennings, N. and Shadbolt, N. Research Agenda for the Semantic Grid: A Future e-Science Infrastructure. UK National eScience Center, 2002, www.semanticgrid.org.
27. Fallside, D.C. XML Schema Part 0: Primer. W3C, Recommendation, 2001, <http://www.w3.org/TR/xmlschema-0/>.
28. Foster, I. The Grid: A New Infrastructure for 21st Century Science. *Physics Today*, 55 (2). 42-47. 2002.
29. Foster, I. and Kesselman, C. Globus: A Toolkit-Based Grid Architecture. In Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 259-278.
30. Foster, I. and Kesselman, C. (eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
31. Foster, I., Kesselman, C., Lee, C., Lindell, R., Nahrstedt, K. and Roy, A., A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In Proc. International Workshop on Quality of Service, (1999), 27-36
32. Foster, I., Kesselman, C., Nick, J.M. and Tuecke, S. Grid Services for Distributed Systems Integration. *IEEE Computer*, 35 (6). 2002.
33. Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S. A Security Architecture for Computational Grids. In ACM Conference on Computers and Security, 1998, 83-91.
34. Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15 (3). 200-222. 2001. www.globus.org/research/papers/anatomy.pdf.
35. Fox, G., Balsoy, O., Pallickara, S., Uyar, A., Gannon, D. and Slominski, A. Community Grids. Community Grid Computing Laboratory, Indiana University, 2002.
36. Frey, J., Tannenbaum, T., Foster, I., Livny, M. and Tuecke, S., Condor-G: A Computation Management Agent for Multi-Institutional Grids. In 10th International Symposium on High Performance Distributed Computing, (2001), IEEE Press, 55-66
37. Gannon, D., Bramley, R., Fox, G., Smallen, S., Rossi, A., Ananthakrishnan, R., Bertrand, F., Chiu, K., Farrellee, M., Govindaraju, M., Krishnan, S., Ramakrishnan, L., Simmhan, Y., Slominski, A., Ma, Y., Olariu, C. and Rey-Cenvaz, N., Programming the Grid: Distributed Software Components, P2P, and Grid Web Services for Scientific Applications. In Grid 2001, (2001)
38. Gasser, M. and McDermott, E., An Architecture for Practical Delegation in a Distributed System. In Proc. 1990 IEEE Symposium on Research in Security and Privacy, (1990), IEEE Press, 20-30
39. Getov, V., Laszewski, G.v., Philippsen, M. and Foster, I. Multiparadigm Communications in Java for Grid Computing. *Communications of the ACM*, 44 (10). 118-125. 2001.
40. Graham, S., Simeonov, S., Boubez, T., Daniels, G., Davis, D., Nakamura, Y. and Neyama, R. Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI. Sams, 2001.
41. Grimshaw, A.S., Ferrari, A., Knabe, F.C. and Humphrey, M. Wide-Area Computing: Resource Sharing on a Large Scale. *IEEE Computer*, 32 (5). 29-37. 1999.
42. Grimshaw, A.S., Ferrari, A.J. and West, E.A. Mentat. In *Parallel Programming Using C++*, MIT Press, 1997, 383-427.
43. Grimshaw, A.S. and Wulf, W.A. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40 (1). 39-45. 1997.
44. Gullapalli, S., Czajkowski, K., Kesselman, C. and Fitzgerald, S. The Grid Notification Framework. Global Grid Forum, Draft GWD-GIS-019, 2001.
45. Johnston, W. Realtime Widely Distributed Instrumentation Systems. In Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 75-103.
46. Johnston, W.E., Gannon, D. and Nitzberg, B., Grids as Production Computing Environments: The

- Engineering Aspects of NASA's Information Power Grid. In Proc. 8th IEEE Symposium on High Performance Distributed Computing, (1999), IEEE Press
47. Kreger, H. Web Services Conceptual Architecture. IBM Technical Report WCSA 1.0, 2001.
 48. Liskov, B. Distributed Programming in Argus. *Communications of the ACM*, 31 (3). 300-312. 1988.
 49. Litzkow, M. and Livny, M. Experience With The Condor Distributed Batch System. In IEEE Workshop on Experimental Distributed Systems, 1990.
 50. Livny, M. High-Throughput Resource Management. In Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 311-337.
 51. Mitchell, J.G., Gibbons, J., Hamilton, G., Kessler, P.B., Khalidi, Y.Y.A., Kougiouris, P., Madany, P., Nelson, M.N., Powell, M.L. and Radia, S.R., An Overview of the Spring System. In *COMPCON*, (1994), 122-131
 52. Mockapetris, P.V. and Dunlap, K., Development of the Domain Name System. In *SIGCOMM*, (1988), ACM, 123-133
 53. Mukhi, N. Web Service Invocation Sans SOAP. 2001, <http://www.ibm.com/developerworks/library/ws-wsif.html>.
 54. Mullender, S. (ed.), *Distributed Systems*, 1989.
 55. Nakada, H., Sato, M. and Sekiguchi, S. Design and Implementations of Ninf: towards a Global Computing Infrastructure. *Future Generation Computing Systems*, 15 (5-6). 649-658. 1999.
 56. Nick, J.M., Moore, B.B., Chung, J.-Y. and Bowen, N.S. S/390 Cluster Technology: Parallel Sysplex. *IBM Systems Journal*, 36 (2). 172-201. 1997.
 57. Oaks, S. and Wong, H. *Jini in a Nutshell*. O'Reilly, 2000.
 58. Paton, N.W., Atkinson, M.P., Dialani, V., Pearson, D., Storey, T. and Watson, P. *Database Access and Integration Services on the Grid*. Manchester University, 2002.
 59. Raman, S. and McCanne, S. A Model, Analysis, and Protocol Framework for Soft State-based Communication. *Computer Communication Review*, 29 (4). 1999.
 60. Shapiro, M. SOS: An Object Oriented Operating System—Assessment and Perspectives. *Computing Systems*, 2 (4). 287-337. 1989.
 61. Sharma, P., Estrin, D., Floyd, S. and Jacobson, V., Scalable Timers for Soft State Protocols. In *IEEE Infocom '97*, (1997), IEEE Press
 62. Steen, M.v., Homburg, P., Doorn, L.v., Tanenbaum, A. and Jonge, W.d. Towards Object-based Wide Area Distributed Systems. In Carbrera, L.-F. and Theimer, M. eds. *International Workshop on Object Orientation in Operating Systems*, 1995, 224-227.
 63. Steiner, J., Neuman, B.C. and Schiller, J., Kerberos: An Authentication System for Open Network Systems. In *Proc. Usenix Conference*, (1988), 191-202
 64. Stevens, R., Woodward, P., DeFanti, T. and Catlett, C. From the I-WAY to the National Technology Grid. *Communications of the ACM*, 40 (11). 50-61. 1997.
 65. Thomas, A. *Enterprise Java Beans Technology: Server Component Model for the Java Platform*. 1998, http://java.sun.com/products/ejb/white_paper.html.
 66. Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S. and Kesselman, C. *Grid Services Specification*. 2002, www.globus.org/research/papers/gsspec.pdf.
 67. Tuecke, S., Engert, D., Foster, I., Thompson, M., Pearlman, L. and Kesselman, C. *Internet X.509 Public Key Infrastructure Proxy Certificate Profile*. IETF, Draft draft-ietf-pkix-proxy-01.txt, 2001.
 68. Villacis, J., M.Govindaraju, Stern, D., Whitaker, A., Breg, F., Deuskar, P., Temko, B., Gannon, D. and Bramley, R., CAT: A High Performance, Distributed Component Architecture Toolkit for the Grid. In *IEEE Intl Symp. on High Performance Distributed Computing*, (1999)
 69. Zhang, L., Braden, B., Estrin, D., Herzog, S. and Jamin, S., RSVP: A new Resource ReSerVation Protocol. In *IEEE Network*, (1993), 8-18

discovery

life time

Termination time=終了時間？実は消去の方が良いかも... 終了はうまく終わった場合で、一応上のほうでも消去と終了使い分けている

WSDL operation=WSDL 演算？ WSDL 操作？ととりあえず操作にしてあるはず、演算の方が良いかも

Register=登録？

notification = 告知にしてある（ととりあえず）

framework=フレームワーク？

委譲としているところは委譲の方が良いかも。一度渡した権限が返してもらえるかにも困るけど。