Open Grid Services Architecture I. Foster, Argonne & U.Chicago (Editor)
http://forge.gridforum.org/projects/ogsa-wg H. Kishimoto, Fujitsu (Editor) Authors:
A. Savva, Fujitsu (Editor)
D. Berry, NeSC
A. Djaoui, CCLRC-RAL
A. Grimshaw, UVa
B. Horn, IBM
F. Maciel, Hitachi
F. Siebenlist, ANL
R. Subramaniam, Intel
J. Treadwell, HP
J. Von Reich, HP

オープン・グリッド・サービス・アーキテクチャ　Version 1.0

本文書の位置づけ
本文書はオープン・グリッド・サービス・アーキテクチャ（OGSA）の仕様に関する情報を伝えるものであり、標準事項や技術的推奨事項を定義するものではない。配布は自由に行ってもよい。

概要
分散型システムインテグレーションや仮想化、管理を行うために幅広く活用、採用されるフレームワークとして、オープン・グリッド・サービス・アーキテクチャ（OGSA）の実現に成功するには、インタフェース、挙動、リソースモデル、バインディングのコアセットを定義する必要がある。OGSA ワーキンググループがグローバル・グリッド・フォーラム（GGF）の範囲内で作成した本文書は、こうした OGSA の定義に関して最初の説明を与えるものである。本文書では、e-サイエンスや e-ビジネスにおけるグリッドシステムやアプリケーションをサポートするために必要な重要な機能とその要件に焦点を当てている。この機能とは、実行管理、データ、リソース管理、セキュリティ、自己管理、情報などである。機能の記述は高レベルにあり、機能間の相互関係もある程度含まれている。

目次
（原文参照ください）

ogsa-wg@ggf.org

1 はじめに

グリッドシステムおよびそのアプリケーションは、異機種分散型の動的「仮想組織」[Grid Anatomy][Grid Physiology]上でリソースやサービスを統合、仮想化、操作することを目的とするものである。このためには組織内および組織間の異種コンピュータシステム間に立ちふさがる多くの障壁を崩し、コンピュータやアプリケーションサービス、データ、その他のリソースに対して、それらがどの場所にあろうとも必要に応じていつでもアクセスできなければならない。

このグリッドシステムを実現するために重要なポイントとなるのが標準化である。標準化を行えば現代のコンピュータ環境を構成している多様なコンポーネントに対し、検出、アクセス、アロケーション、モニタリング、会計、課金などを行うことができる。また一般に、１つの仮想システムとして操作を行うこともできる。その際、各コンポーネントが異なるベンダーが提供したものであり、しかも異なる組織で使用されているものであっても構わない。共同利用でき、ポータブルでしかも再利用可能なコンポーネントやシステムを構築するのであれば、標準化は重要なポイントとなる。さらに、標準化により適正な使用形態が促進され、安全かつ強固で拡張可能なグリッドシステムの構築にも寄与することになる。

本文書では、サービス指向アーキテクチャであるオープン・グリッド・サービス・アーキテクチャ（OGSA）を紹介する。OGSA は、グリッドシステムの重要な問題点に対処するために中心的な役割を果たす機能や動作を定義することで、上述の標準化のニーズに応えるものである。この問題には次のような点が含まれる。いかにアイデンティティを作り上げて認証を取り決めるか。ポリシーをいかに表し取り決めるか。サービスをいかに見つけるか。サービスレベル・アグリーメントをいかに取り決め、モニタするか。仮想組織のメンバーシップや仮想組織内の通信をいかに扱うか。信頼性が高く、スケーラビリティのあるサービスのセマンティクスを供給するには、いかにサービスコレクションを階層的に組織化すればよいか。データリソースをコンピューテーションにいかに組み入れるか。サービスコレクションをいかにモニタし取り扱うか。

本文書は大きく分けて２つの部分から構成されている。１つは必要条件、もう１つは機能に焦点を当てている。２章では OGSA に求められる一連の要件に関し、大まかな定義を示している。この分析は、要件、技術的問題、使用例、過去の経験、最新の関連事例に基づいている。その抽象的概念は、基盤となるインフラストラクチャに対していかなる仮定を行おうとも制約を受けることがなく、むしろ「グリッド」に関する議論を組み立て、ソリューションを定義することを目的としている。

３章では、２章の要件を、OGSA を総合的に定義する整合性のある機能に読み替える。最初に、OGSA の設計開発の制約となるインフラストラクチャのサービスと条件について紹介する。とくに OGSA が、新しく生まれつつある Web サービス・アーキテクチャ

[WS-Architecture]の増大する技術仕様群に基づきどのように構成されているか、同時にOGSA が Web サービスの発展にどのように寄与しているかについて解説する。さらに、実行管理、データ、リソース管理、セキュリティ、自己管理、情報サービスなど、必要とする機能内容の改良についても言及する。

本文書では、文中の説明だけでは意味のわかりにくい用語を多く用いている。用語の明確な定義については、付属の「OGSA 用語集」[OGSA Glossary]を参照されたい。

本文書の今後の版では、特定のサービスに関する説明を行い、既存のサービス仕様をそのままの形で使用することができるのはどのような場合か、そしてサービス仕様の修正あるいは新規のサービス仕様が必要となるのはどのような場合かを、明確にする予定である。さらに、そうした拡張あるいは定義を決める作業が進められている現状についても解説する。

グローバル・グリッド・フォーラムの OGSA ワーキンググループが作成したこの情報文書 GWD-I は、OGSA version 1.0 を定義するものである。OGSA ワーキンググループは、アーキテクチャの標準的内容を示すために、一連の提言文書 GWD-R のリリースを今後予定している。

2. 要件

この OGSA version 1.0 の定義は、機能として定義できる要件と定義できない要件から導出されたものである。これらの要件は、表１および付属文書[OGSA Use Cases][OGSA Use Cases Tier 2]に示した使用例から読み取ることができる。これらの使用例は、商用および学術目的のインフラストラクチャやアプリケーションのシナリオを扱っている。これらの使用例を用いて正式な要件の解析を行うことはできないが、アーキテクチャを決める過程において有用な情報となりうるものである。

表１　OGSA の使用例

使用例	概要
商用データセンター(CDC)	データセンターは、管理コストの削減やリソースの利用を増やしつつ、サーバ、ストレージ、ネットワークを含む数千もの IT リソースを管理しなければならない。
暴風雨のモデル化	天候をリアルライムで広範囲にわたって計測する装置と、データモデルによる大規模シミュレーションを組み合わせて、暴風雨の正確な位置予想を行うものである。
オンラインメディア、オンラインエンターテインメント	エンターテインメント・エクスペリエンスを、消費または相互のやりとりのために配信する。
核融合グリッド（National	核融合研究を行う仮想組織を定義し、アプリケーショ

Fusion Collaboratory, NFC）	ン・サービス・プロバイダ（ASP）モデルを使ってこの組織が開発、実行するソフトウェアのニーズに応えるものである。
サービスをベースとした分散型クエリ・プロセッシング	サービスをベースとした分散型クエリ・プロセッサは、１つあるいは複数の既存のサービスに対し宣言型言語で表記されたクエリの評価を行うものである。
グリッド・ワークフロー	ワークフローは、既存のサービスを再構成することで新しいサービスを構築する便利な方法である。ワークフローの定義をワークフロー・エンジンに登録することで、新しいサービスを作成、使用することができる。
グリッドリソース・リセラー	グリッドリソースの所有者とエンドユーザの間にサプライ・チェーンを導入することで、リソースの所有者が中核業務に集中できるようになり、エンドユーザはリセラーがパッケージ化したリソースを購入することができる。
インターグリッド	インターグリッドは、グリッドと非グリッドが混在したデータセンターや複数の企業にまたがったグリッドなど、グリッド化できず変更も困難な大量のアプリケーションに重点を置いた形で上述の CDC 使用例を拡張したものである。また、ユーティリティ・コンピューティングの一般的概念も目に見えるようになる。
インタラクティブグリッド	オンラインメディアの使用例と比較し、この使用例では分散実行の高い精度に着目している。
グリッドライト	PDA、携帯電話、ファイアウォールなどの小型デバイスにグリッドを使用し、小型デバイスをグリッド環境の一部にするために不可欠なサービスを決めるものである。
仮想組織（VO）グリッドポータル	VO は、さまざまな計算データや機器関連データ、その他のリソースに対し、VO のメンバーがアクセスできるようにするものである。グリッドポータルでは、VO のメンバーが使用できる数々のリソースをエンドユーザに見せることができる。
永続的アーカイブ	認証記録を保ちながら新旧のプロトコル、ソフトウェア、ハードウェア間のマッピングを管理するため、適切なアブストラクション・レイヤを提供して技術の進歩に対処する保存環境である。
相互認証	CDC や NFC の使用例を改良し、ジョブの実行先となる

	リソースをオーソライズするジョブの送り手のシナリオを導入する。
リソース使用サービス	異機種分散環境におけるアプリケーション、ミドルウェア、オペレーティングシステム、物理的（計算およびネットワーク）リソースが生み出すリソース使用メトリクスの媒介を容易にする。
IT インフラストラクチャと管理*	ジョブの実行、サイクル分割、プロビジョニングのシナリオ。
アプリケーション使用例*	ピアツーピアの PC グリッドコンピューティング、ファイル共有、コンテンツデリバリのシナリオ。
リアリティグリッド*	操縦型コンピューティングやハイエンドのオンライン可視化を通じた、パラメータ空間の分散型および協調型探索。
ラーニンググリッド*	仮想組織の枠組みでユビキタス・ラーニングに対し、ユーザを中心として文脈的で経験に基づいたアプローチを行う。
HLA をベースとした分散型シミュレーション*	管理ドメインを横断するシミュレーションを開発、実行するための分散型協調環境。
GRID をベースとしたビジネス向け ASP*	グリッド技術に基づいた異なるビジネスモデルをサポートするアプリケーション・サービス・プロバイダ(ASP)のインフラストラクチャ
グリッド・モニタリング・アーキテクチャ*	広範囲ネットワークに拡張でき、多数の動的異機種リソースを網羅することのできるグリッドモニタリングシステム。

*[OGSA Use Cases Tier 2]のドキュメントに登場する使用例

これらの使用例およびその他の関連情報を解析することで、グリッド環境とそのアプリケーションの特徴や、重要かつ幅広い関連性を持つ機能上・非機能上の必要条件を明らかにすることができる。以下の節で明らかになった点をまとめる。

2.1 動的異機種環境の相互運用性およびサポート

いくつかの使用例には、専門的に特化したプロファイルを与えることになる制約の多い環境もあれば均質な環境もある。しかしグリッド環境は一般に、さまざまなベンダーが提供する多様なホスティング環境（J2EE や.NET など）、オペレーティングシステム（UNIX、Linux、Windows、埋め込みシステム）、デバイス（コンピュータ、機器、センサ、ストレージシステム、データベース、ネットワーク）、サービスなどを網羅した異種混在型かつ分

散型の環境であるという明らかな傾向がある。さらにグリッド環境は、長い期間にわたって継続し、動的なものとして構築されることが多く、そのため当初予想していなかった方向に進化する可能性がある。

OGSA は、それらの多様な異機種分散型リソースやサービスの間の相互運用性を可能にし、異機種システムの複雑な管理を容易にするものでなければならない。さらにセキュリティやリソース管理など、分散型環境において必要とされる多くの機能が安定で信頼性の高いレガシ・システムによってあらかじめ実装されていることもある。そうしたレガシ・システムを変更することは容易ではない。したがってグリッド内にレガシー・システムを組み込む必要がある。

異機種システムを維持する必要性からは、次のような要請が生まれる。

- リソースの仮想化：異機種システムの複雑な管理を容易にし、統一した手法で多様なリソースを取り扱うために重要な要素である。
- 共通管理機能：異機種システムの管理を簡素化するには、リソースを一貫性をもって統一的に管理するメカニズムが必要である。最低限の容易な共通管理機能が必要となる。
- リソースの検索とクエリ：要求する属性を持ったリソースを見つけ出し、その特性を引き出すメカニズムが必要である。高度に動的な異機種システムを、その検索とクエリが処理できなければならない。
- 標準プロトコルとスキーム：これらは相互運用性にとって重要なものである。さらに標準プロトコルは、それによりグリッドへの移行をより簡単なものにすることができることから、とりわけ重要である。

2.2 組織間のリソース共有

グリッドは一枚岩的なシステムではなく、さまざまな組織が所有、管理するリソースから構成される場合が多い。OGSA の大きな目的の１つは、管理ドメインが１つの会社内で別々の装置として分かれている場合や、異なる機関にまたがって存在している場合、それらの管理ドメインを横断してリソースを共有、利用することにある。それには、組織の境界を超えてユーザ、リクエスト、リソース、ポリシー、アグリーメントを互いに関連付ける際に用いるコンテクストを与える機構が必要となる。組織間のリソース共有からは、さまざまなセキュリティ上の要件が発生する。この点については 2.7 節で取り上げる。

リソース共有のための要件としては以下の点が挙げられる。

- グローバルなネーム空間：データやリソースへのアクセスを容易にするために必要となる。OGSA のエンティティは、他のエンティティに対して、場所や複製か複製でないかにかかわらず、セキュリティ上の制約の範囲内でユーザが意識することなくアクセスできなければならない。
- メタデータ・サービス：これは、OGSA のエンティティを見つけ出し、呼び出し、追跡

する上で重要なものである。メタデータ・サービスは、管理ドメインを横断してエンティティのメタデータにアクセスしたり、メタデータを伝播、集約、管理したりすることを可能にする。

- サイトの自律性：ローカルなコントロールやポリシーに従いながらサイトを横断してリソースにアクセスするためのメカニズムが必要である（2.7 節のデリゲーションを参照）。
- リソース利用データ：会計や課金などのために、組織間でリソース利用データ（リソース消費データ）を収集、交換するためのメカニズムおよび標準スキームが必要となる。

2.3 最適化

最適化は、コンシューマやサプライヤのニーズに合うよう、リソースを効果的に割り振る技術を指す。最適化は、リソースやサービスのサプライヤ（供給側）とコンシューマ（需要側）の両方に適用されるものである。供給側の最適化によくあるケースは、リソースの最適化である。たとえば、リソースの割り振りは最悪のシナリオ（たとえば予想最大負荷や故障対策としてのバックアップ）を見越して行うことが多く、リソースを十分に使い切らない場合が生じる。限られた時間内でリソースを事前予約したり、バックアップ用リソースをプーリングしたりするなど、リソースの割り振りに関して柔軟なポリシーを作成することでリソースの利用は改善される。

需要側の最適化は、予測が困難なアグリゲートの作業負荷に対する需要を含め、多様な作業負荷を管理することができなければならない。この点で重要な要件となるのが、全体的なサービスレベルの方針に合致するよう、作業負荷の優先度を動的に調整する能力である。リソース利用を測定、監視、ログに残すなど、利用を追跡するためのメカニズムや、リソースの割り振りを変更するメカニズム、あるいは要求に応じてリソースを提供するメカニズムが、需要側最適化の土台として必要となる。

2.4 サービスの品質（QoS）保証

ジョブの実行サービスやデータサービスなどのサービスは、取り決めどおりの QoS を提供しなければならない。重要な QoS としては、これに限られるわけではないが、アベイラビリティ、セキュリティ、性能があげられる。QoS に関する要請は、測定可能な項目を用いて行うべきである。

QoS 保証に関する要件には以下のものが含まれる。

- サービスレベル・アグリーメント：QoS は、サービスをリクエストする側と提供する側が、サービスの実行前に取り決めを行い、そのアグリーメントによって決まるものである。アグリーメントを作成、管理するための標準的メカニズムが必要である。
- サービスレベルの達成：アグリーメントにサービスレベルの達成が必要な場合、求められている QoS を維持するためにそのサービスが使用するリソースを調整する必要がある。したがって、サービス品質のモニタリング、リソース利用の推定、リソースの使用

計画と使用調整のためのメカニズムが必要となる。

・ マイグレーション：パフォーマンスやアベイラビリティを向上させるために作業負荷を調整する上で、サービスやアプリケーションの実行を移動させることが可能でなければならない。

2.5 ジョブの実行

OGSA では、ユーザが定義した作業（ジョブ）の実行に対し、ジョブが終了するまでのマネージャビリティが与えられる必要がある。多数の異機種リソースにジョブが分散された場合でも、スケジューリング、プロビジョニング、ジョブ管理、ジョブの例外処理などの機能がサポートされていなければならない。

ジョブの実行に関しては、以下の要件が含まれる。

・ 多様なジョブのサポート：作業負荷や複合サービスのような単純なジョブや複雑なジョブを含め、さまざまな種類のジョブの実行がサポートされていなければならない。

・ ジョブ管理：ジョブが存在している間は、それを管理できるということが重要である。ジョブは、マネージャビリティのためのインタフェースをサポートしてなければならない。またそのインタフェースは、作業負荷やジョブアレイなど、多様なジョブの集まりとともに機能するものでなければならない。各ジョブステップの実行管理や、オーケストレーションやコレオグラフィといったサービスの管理を行うためのメカニズムが必要である。

・ スケジューリング：決められた優先度やリソースの現在の割り振り等の情報に基づいて、ジョブの実行やスケジューリングを行う機能が必要となる。さらに、複数のスケジューラを用いて管理ドメインをまたがってスケジューリングを行うメカニズムを構築することも必要である。

・ リソースのプロビジョニング：リソースの割り振り、デプロイメント、設定などの複雑なプロセスを自動化させるための機能が必要となる。ジョブの実行に要する環境を準備するために、必要であれば OS やミドルウェアなどのホスティング環境をデプロイ、再設定し、必要なアプリケーションやデータをリソースに自動的にデプロイ、設定することができなければならない。計算リソースだけでなく、たとえばネットワークやデータのリソースなど、あらゆる種類のリソースをプロビジョニングできなければならない。

2.6 データサービス

非常に大量のデータに効率的にアクセスし、それを移動する機能を必要とする科学技術の分野が増えている。またデータ共有も重要であり、たとえば個別に管理、運営されているデータベースに保存された情報に対し、アクセスできる必要がある。ビジネスの分野では、データの保管や管理が重要な要件となっている。

データサービスの要件には以下のようなものがあげられる。

- データアクセス：さまざまな種類のデータ（データベース、ファイル、ストリームなど）に対し、その物理的な場所やプラットフォームに関係なく、根底にあるデータソースを抽出することで簡単かつ効率的にアクセスできなければならない。さまざまな粒度レベルにおけるアクセス権を管理するためのメカニズムも必要である。
- データ整合性：OGSA では、キャッシュ内のデータや複製されたデータが修正を受ける場合に、整合性が保たれる必要がある。
- データ保持性：データや、そのメタデータに関連するものは、それらが存続する限りは維持管理が行われなければならない。複数の永続性モデルを使用することができなければならない。
- データインテグレーション：OGSA は、フェデレートされた異機種分散型データを統合するメカニズムを提供するものでなければならない。また、さまざまなフォーマットで提供されるデータを統一的な方法で検索することができなければならない。
- データロケーション管理：必要なデータは、いずれの場所からも取得できなければならない。OGSA は、データの特性に応じて、移行、複製、キャッシュ化するなど、いろいろな方法で選別する機能を有していなければならない。

2.7 セキュリティ

安全管理には、与えられたセキュリティポリシーにしたがって、しっかりしたセキュリティプロトコルによるサービスへのアクセス管理が不可欠である。たとえば、手に入れたアプリケーションプログラムをグリッドシステムにデプロイするには、認証と認可が必要である。さらに、ユーザがリソースを共有するためには、何らかの分離メカニズムが必要となる。これに加えて、管理ドメイン間でリソースを安全に共有しながらグリッドシステムを保護するには、そのためにデプロイすることのできる安全で標準的なメカニズムが求められる。

セキュリティ要件としては以下の点があげられる。

- 認証と認可：個人とサービスのアイデンティティを定めるために認証メカニズムが必要となる。サービス・プロバイダは、それぞれのサービスの使われ方に関するポリシーを施行するため、認可メカニズムを導入することが重要である。グリッドシステムは、各ドメインのセキュリティポリシーに準拠すると同時に、ユーザのセキュリティポリシーも確認する必要がある。認可は、さまざまなアクセス管理モデルやアクセス管理の遂行に適応するものでなければならない。
- マルチセキュリティ・インフラストラクチャ：分散型オペレーションを行うということは、複数のセキュリティ・インフラストラクチャと融合し、相互運用する必要性があることを意味する。OGSA は、既存のセキュリティ・アーキテクチャやセキュリティ・モデルと融合し、相互運用する必要がある。
- 境界セキュリティ・ソリューション：組織の境界を越えてリソースにアクセスする必要

性が出てくることがある。OGSA では、ファイアウォール・ポリシーや侵入検知ポリシーといったローカルなセキュリティメカニズムに悪影響を及ぼすことなくドメイン間の相互作用を可能にする一方で、組織を保護するための標準的なセキュリティメカニズムを配備する必要がある。

- 分離：ユーザの分離、パフォーマンスの分離、同じグリッドシステム内でのコンテンツ・オファリング間での分離など、いろいろな分離が必要となる。
- デリゲーション：サービスの要求元から提供側へのアクセス権のデリゲーションに関し、これを許可するメカニズムが必要である。デリゲートされた権利を誤用するリスクは、対象とするジョブにデリゲーションを通じて権利を移行することを制限したり、権利の期限を設定するなどして、最小限にとどめなければならない。
- セキュリティ・ポリシー交換：サービスの要求側と提供側とは、互いの間でセキュリティ・コンテクストの合意を取りつけるために、セキュリティ・ポリシー情報を動的にやりとりする必要がある。
- 侵入の検知、保護、安全なロギング：ウイルスやワームによる攻撃などの侵入の検知、さらには誤用や悪意ある利用の検出のためには、強力な監視機構が要求される。また、重要な領域や機能を攻撃から遠ざけることにより、これらを保護することも必要である。

2.8 管理コストの低減

分散型の巨大な異機種システムの複雑な管理は、管理コストと人的ミスの危険性を高めることにつながる。管理作業の自動化や仮想化リソースの一貫した管理など、管理作業に対する支援が求められる。

グリッドシステムの自動管理には、ポリシーに基づいた管理が必要である。これにより、管理作業が、グリッドシステムを操作、利用する組織の目的に合致するようになる。リソースの監視や管理のしかたを決める低レベルポリシーから、課金等のビジネスプロセスの管理を決める高レベルポリシーまで、システムにはレベルに応じたポリシーがある。ポリシーには、アベイラビリティ、パフォーマンス、セキュリティ、スケジューリング、ブローカリングなども含まれる。

アプリケーションコンテンツ管理メカニズムは、アプリケーションのあらゆる関連情報を１つの論理的ユニットとして指定、管理することで、複雑なシステムのデプロイメント、コンフィグレーション、メンテナンスを容易にすることができる。このアプローチを取ることで、管理者はアプリケーションに関する専門知識がなくても、アプリケーションコンポーネントを簡潔かつ信頼できる方法で維持することが可能となる。

問題が発生した場合に管理者が素早く認識、対処できるように、問題判定メカニズムが必要である。

2.9 スケーラビリティ

大規模グリッドシステムは、ジョブ所要時間（経過時間）の大幅な短縮や多数のリソースの利用などの付加価値を生み出すことができ、これにより新たなサービスが可能となる。しかし管理インフラストラクチャに対する新たな必要性が生じることから、こうした大規模システムにも問題が発生することがある。

管理アーキテクチャは、きわめて多様な特性を持った数千ものリソースに対応する能力が必要である。その管理は、階層的にあるいはピアツーピア的に（フェデレートされた方法またはコラボラティブな方法にて）行われなければならない。

1 つ 1 つの計算を最適化すると同時に、計算プロセス全体のスループットを向上させるためにジョブの並列処理を調整、最適化させるには、高スループット・コンピューティング・メカニズムが必要となる。

2.10 アベイラビリティ

高いアベイラビリティは、多くの場合、高価でフォルトトレラントなハードウェアや複雑なクラスタシステムがあれば実現できる。重要な公的インフラストラクチャ・サービスを提供する IT システムが幅広く利用されるようになったことで、多くのシステムが高いレベルのアベイラビリティを達成することを要求されるようになってきている。グリッド技術があれば、組織内および組織間で、より幅広いリソースプールへ透過的ににアクセスすることが可能になる。このため、グリッド技術を１つのビルディングブロックとして使うことで、安定で信頼性の高い実行環境を実現することができる。ただし、グリッド内に異機種が混在していることから、信頼性の高い既存のシステムで一般的に使用されているコンポーネントよりも、平均修復時間（MTTR）の長い、あるいは予測不可能なコンポーネントをグリッドシステムでは用いなければならず、この点が難しい問題である。

こうした複雑な環境では、ポリシーに基づいた自立的管理（2.8 節のポリシーに基づいた管理を参照）と動的プロビジョニング（2.5 節のプロビジョニングを参照）が、フレキシビリティとリカバビリティの高いシステムを実現する上で重要となる。

ディザスタ・リカバリ・メカニズムは、自然災害や人的災害の発生時に長時間にわたるサービス停止を避け、グリッドシステムの動作を復旧させるために必要となる。リモートバックアップや復旧作業を簡素化、自動化させることが必要である。

リソース・フォルトで実行中のジョブが失われないよう、フォルト管理メカニズムが求められる。また、フォルトの監視や検知、あるいはフォルトの原因や実行中ジョブへの影響の解析を行うためのメカニズムも必要となる。さらに、チェックポイント・リカバリなどの技術を使って、フォルト処理の自動化も求められている。

2.11 使いやすさと拡張性

OGSA を使うことで、必要であればユーザが環境の複雑さを覆い隠すことも可能でなければならない。ランタイム機能に呼応して作動するツール類により、できるかぎりユーザ

環境が管理され、有用な抽象化が必要なレベルで提供されなければならない。低レベルの決定や低レベルのシステム・メカニズムとの連動を要請、要求する厳しいアプリケーションを持った「パワーユーザ」の存在を知ることが、使いやすさという目的を左右することになる。したがって、エンドユーザがシステムとやりとりするレベルを選択することが可能でなければならない。

ユーザの持っている数多くの多様なニーズをすべて予測することは不可能である。そのため、OGSA が時間とともに進化し、ユーザが独自のメカニズムやポリシーを特定のニーズに合うように作ることができるよう、拡張性のある交換式コンポーネントを使ってメカニズムやポリシーを構成しなければならない。さらに、中核となるシステムコンポーネント自体が、拡張性があり、交換可能でなければならない。こうした拡張性により、開発や利用を目的とする付加価値のついたサービスを提供するサードパーティ（あるいはサイトローカル）の実施が可能になる。拡張性とカスタマイズは、相互操作性を損なうことがない方法で提供される必要がある。

3 機能

つぎに、上で紹介した要件に合致する特定の機能に目を向けることにする。

3.1 概要

OGSA は、分散型かつ異機種リソースのシームレスな利用、管理を容易にすることを目的としている。このアーキテクチャでは、「分散型」「異機種型」「リソース」といった言葉が広い意味で使われている。たとえば「分散型」は、位置的に隣接し、何らかの接続構造により互いにリンクしたリソースから、グローバルでマルチドメイン型の、大まかで間欠的なリンクを持つリソースまで、幅広い意味を持つ。「リソース」は、アーチファクト、エンティティ、システム内あるいはシステム上で作業を完結するために必要な知識などを指す。こうしたインフラストラクチャが提供するユーティリティは、一連の機能として実現される。図 1 は、これらの機能のうちのいくつかを論理的かつ抽象的に半階層構造で示したものである。3 つの主要な論理的、抽象的階層が図示されている。

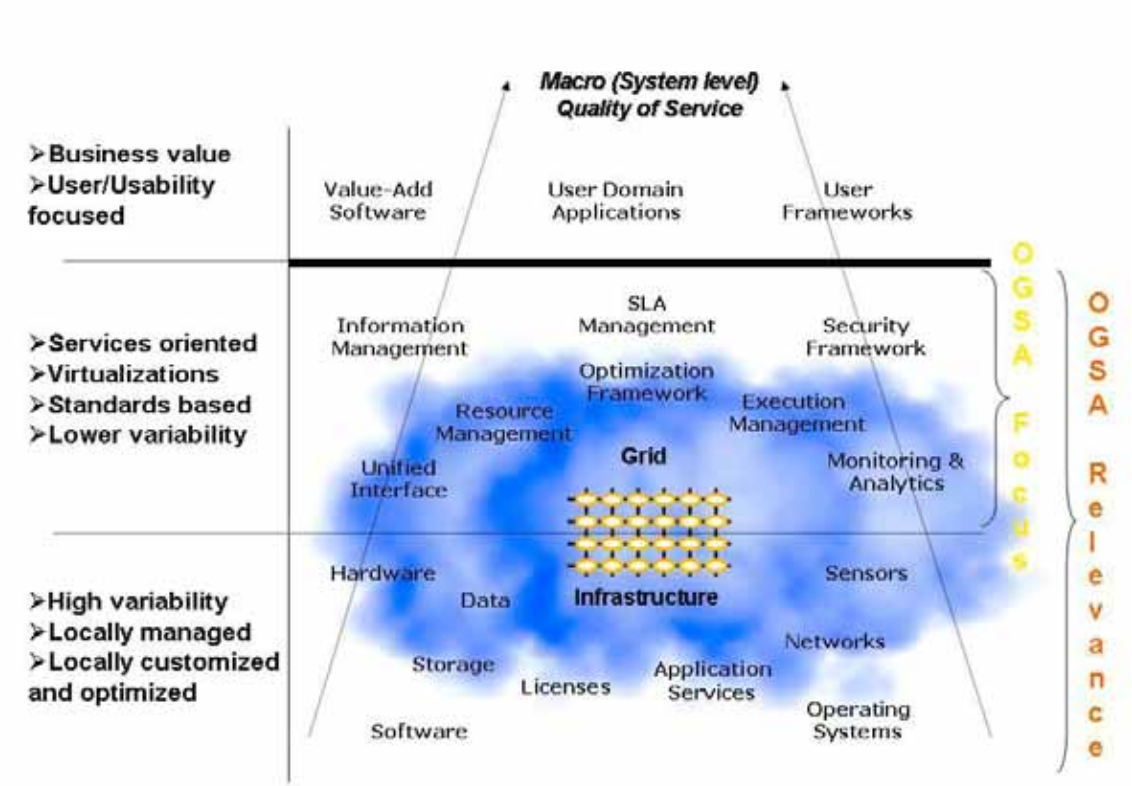


図 1：グリッド・インフラストラクチャの概念図
（注：図示した機能やリソースは、すべてを網羅しているわけではなく、わかりやすくするために最小限にとどめてある。）

図 1 に示した最初の階層（下段）では、ベース・リソースが表示されている。ベース・リソースは、基盤となる物理的あるいは論理的なエンティティやアーチファクトにより支えられたリソースを表している。これらのエンティティやアーチファクトは、OGSA のコンテクストの外部とも関連したものである。そうした物理的エンティティの例としては、CPU、メモリ、ディスクがあげられ、論理的アーチファクトの例としては、ライセンス、コンテンツ、OS プロセスなどがある。仮想化は、仮想化されるエンティティと密接に結びついており、基盤となるエンティティやアーチファクトに名前を付ける際に使用する語句が、それらの仮想化を名付ける際にも使用される。これらのリソースは通常ローカルに所有、管理されているものの、リモートで共有されることもある。配置やカスタマイズもローカルに行われる。実際のエンティティやアーチファクトは急激に変化することがあり、さらにそれらは複数のソースから成り立っていることがあるため、リソースはその特性、サービスの質、バージョン、アベイラビリティなどの点において大きく変わりうる。

この議論では、OGSA の概念とリソースの従来の概念とを結びつけるためにベース・リソースを区別したが、今後本文書では、ベース・リソースと他のリソース・サービスとを

とくに区別することなく、リソースの一般的概念のみ用いていくことにする。

2 つめの階層（中段）には、高いレベルの仮想化と論理的抽象化が示されている。仮想化と抽象化は、OGSA グリッドに関連したさまざまな機能を定義する目的で行われる。こうした機能は、単独で使用することもできるが、高レベルのアプリケーションや「ユーザ」ドメインプロセスを支えるために必要なインフラストラクチャを提供できるように適切に組み合わせて使うこともできる。これら一連の機能は、OGSA で定義されているように、比較的変わることが少なく、標準的なものである。これらの機能がどのように実現、実行され、いかにしてユーザ・ドメイン・アプリケーションによりさらに構成、拡張されるのかが、より大きなインフラストラクチャのマクロな（システムレベルでの）QoS を決定づけるのである。そしてこのことはエンドユーザが経験していることである。なお、図に示した機能は、単に OGSA の機能の一例を表しているに過ぎず、全機能については本章の残りの部分で議論することを留意されたい。

図 1 の中段と下段の階層間の関係について、もう少し詳しく見ていくことが必要である。OGSA のサービス指向の特性から次のことが言える。すなわち、サービスとして表現される仮想化されたリソースが、アーキテクチャ内の他のサービス（たとえば中段および上段の階層におけるサービス）に対するピアとなっている。このピアとしての関係により、アーキテクチャ内のいずれのサービスも、サービス間の相互作用を起動することができる。さらに、2 つめの階層におけるサービスは、個々のサービス（あるいはサービスの集合や組み合わせ）が提供することになっている機能を遂行するため、下段の階層における仮想化（リソース）を利用、管理する必要がある。図 1 では、この密接な関係が、下段と中段の間の細い境界線で表されている。このため下段の階層におけるエンティティは、OGSA の議論に関連するもの、あるいはその一部とみなすことができる。

図の論理表示における 3 番目（上段）の階層には、ビジネスプロセスなどのユーザ指向あるいはドメイン指向の機能やプロセスを実現するために OGSA の機能を利用するアプリケーションやエンティティなどがある。これらの多くは OGSA の範囲外にあるが、インフラストラクチャの対応した使用例からアーキテクチャの定義を導き出すものである。（現在知られている使用例のリストは表 1 を参照。このリストは使用例のすべてを示しているわけではなく、将来その項目が増えると思われる点に注意されたい。）

これらの階層はいずれも、求められているサービスの質（QoS）を提供するために、相互に運用し、相乗的に動作する必要がある。これがアプリケーション・ティア（少なくとも特定のユーザ・シナリオに組み入れられたサービス）を含むシステム全体の QoS であることから、「サービスのマクロ品質」として定められるものである。図 1 では点線矢印でこれを表現している。

3.2 OGSA の枠組み

OGSA は図 1 の中段の論理階層を構成しており、サービスやそのサービスが表に出すイ

ンタフェース、サービスに属するリソースの個々のステートあるいは集団的ステート、サービス指向アーキテクチャ（SOA）におけるそれらのサービス間の相互作用などから成る。図 2 と図 3 に OGSA サービスの枠組みが示されている。図に表示された円柱は、個々のサービスを表している。これらのサービスは、グリッドに関係する動作、追加機能、拡張機能、修整を付け加えた上で Web サービスの規格の上に構築されている。

以下に重要な点をいくつか示す。

- OGSA を構築する重要な動機は、構成パラダイムやブロック構築といったアプローチである。このアプローチでは、ニーズに見合うように、必要最小限の初期機能から一連の性能や機能を必要に応じて構築、採用する。ニーズをあらかじめ知る必要はない。これにより、アーキテクチャに求められる変化に対する適応性、柔軟性、堅牢性が実現される。
- OGSA は、サービスやそのインタフェース、さらには、サービスの動作／振る舞い、相互作用を表現している。なお、これらのサービスの細かな実行をつかさどるソフトウェア・アーキテクチャは、OGSA ワーキンググループの対象外であることを注意しておく。
- さらに、アーキテクチャはレイヤ構造となっておらず、1 つのサービスの実行は、それが論理的に依存しているレイヤ上で構築され、そのレイヤのみと互いに作用しあうものである。すなわち、多くの概念がオブジェクトベースであるように見えるが、実際にはオブジェクト指向である。

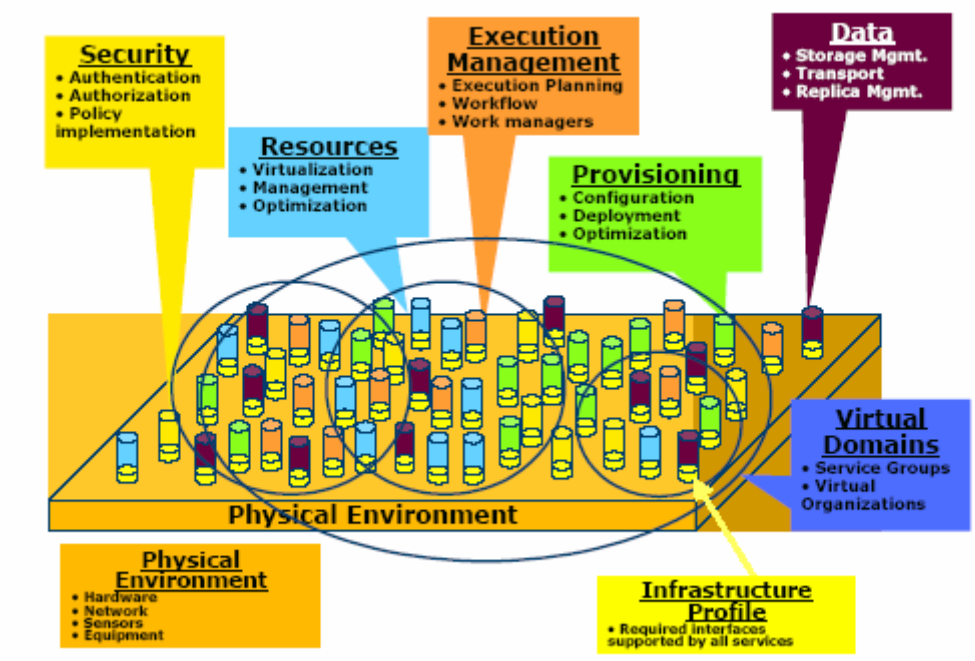


図 2：OGSA の枠組み

（注：すべての機能がここに示されているわけではない。詳細は本章後半を参照のこと。）

サービスは、実行、合成、他のサービスとの相互作用により、単独で、あるいは互いに作用しあっているサービス集団の一部として、OGSA の機能を実現するピアである（図 2 および図 3 参照）。ピア同士は柔軟に結合している。たとえば、「オーケストレーション」を実現するには、サービス集団を構築し、そのうちの一組のサービスがオーケストレーションを行い（すなわち「オーケストレータ」として機能し）、その集団の他のサービスがオーケストレーションするインタフェースやメカニズム（すなわち「オーケストレーションされる側」）を提供する。ある特定のサービスは、複数の機能を実現させるために、複数のサービスや相互作用を実施したり、さらに（あるいは）、そこに参加することができる。一方で、特定の機能を実現する際にすべてのサービスが参加する必要はない。

これらのサービスは、仮想ドメイン（図 2）と呼ばれる仮想的なサービス集団に関与、参加することがある。たとえばサービスグループなどでは機能の実現を目的とし、仮想組織では集団的コンテクストやマネジャビリティの枠組みの共有を目的としている。

OGSA のサービスは、計算ハードウェアやネットワークなどのよく知られた物理的コンポーネントや相互接続、さらには望遠鏡のような物理的機器までをも含む物理環境を必要とし、前提としている。

サービスが OGSA グリッドの一部となるには、Null でないインタフェース、基準、共通知識／ブートストラップといったコアセットをサービスが実装する必要があると思われる。OGSA を支えるこうした共通の実装やマニフェステーションは、インフラストラクチャ・サービスあるいはグリッド・ファブリックと呼ばれる。次の 3.3 節に示すように、Web サービス・リソース・フレームワーク（WS-RF）の仕様は、現在開発途中にあるが、初期のグリッド・ファブリックに欠かせない要素となるはずである。3.3 節では、さらに規格について解説するが、それらはグリッド・ファブリックの一部となりうるものである。

図 3：サービスの相関

- ソースは、クライアントにサービスを提供するためにターゲットとなる機能を使用する。
- サービスは、より高いレベルの機能を提供するために、基盤となるサービスの機能を組み合わせる。
- サービスは、リクエストをその実現のために関連するサービスにデリゲートする。
- サービスは、リクエストを具体化する（検証、具現化する）ために、関連するサービスを参照する。
- サービスは、関連するサービスの機能を拡張する（組み込む、増補する）。

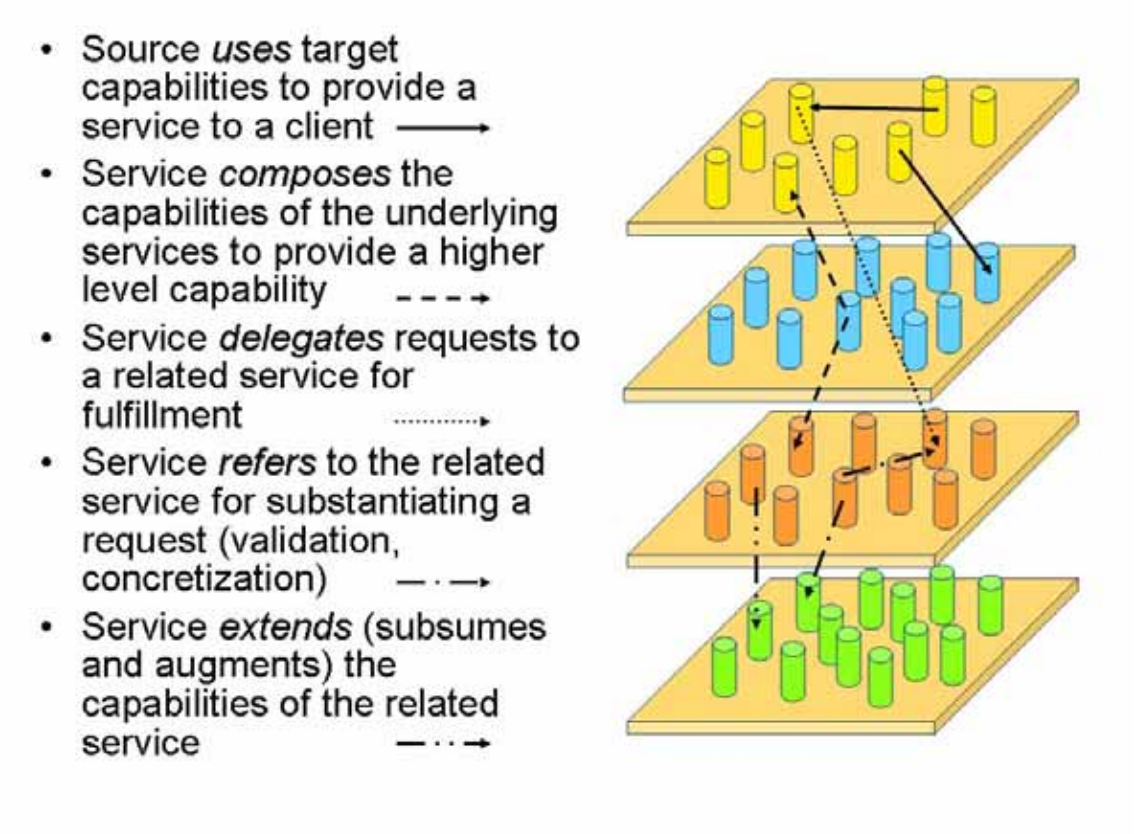


図 3 は、図 2 とは異なる視点から、OGSA のサービス間にあると思われる数多くの関係や相互作用に注目したものである。円柱は、物理的インフラストラクチャのレイヤ上の個々のサービスを表している。図の各「レベル」は、1 つの機能を表現するためのサービスの集合を表している。たとえば、あるレベルは実行管理の機能を実現するものであり、別のレベルはデータ管理機能を行うものであるかもしれない。（ただし、それぞれの「レベル」間には、階層的あるいは積層的関係はないものとする。）図に示した関係は、特定のサービス間に存在し、機能を補うものである。サービス間に存在するこれらの関係としては、たとえば、「使用する」、「組み合わせる」、「デリゲートする」、「参照する」、「拡張する」、などの例がある。管理やマネージャビリティ、サービス・プロファイルの宣言的仕様といった他の目的のためのサービス間の相互関係についても、そのアスペクトをこうした関係を用いてモデル化することが可能である。

3.3 インフラストラクチャ・サービス

OGSA を定義する目標は、サービス指向アーキテクチャのコンテクストの範囲内で、2 節で示した要件に集団で対処するコンポーネントを、整合性のとれた、統合された集まりとして定義することにある。より高いレベルのサービスについて具体的な記述をするのであれば、基礎となるインフラストラクチャに関して想定をする必要がある。抽象性を求めるあまり目標を達成できない仕様には、数多くの例がある。たとえば、COBRA1.1 は、サ

ービス名が実行によって変わることから相互運用性を確立できなかった。したがって、TCP や DNS、その他のより高レベルの TCP プロトコルやサービスの設計が、その下地となる IP 基盤により特徴づけられているように、OGSA の設計も、基礎となるメカニズムをどう選択するかによって影響を受ける。その選択肢をここに列記し、それぞれある程度の根拠を示すことにする。

Web サービスアーキテクチャ[WS-Architecture]を形作る技術的仕様はますます増加しているが、OGSA における作業はそこに基づいて WS-アーキテクチャの発展に寄与するものと主に想定されている。実際 OGSA は、WS の中心的規格を適用するための、特別なプロファイルの一つと見なすことができる。この選択はサービス指向アーキテクチャの利点に対する強い信念に基づいている。さらにこの選択は、グリッドシステムに必要な機能に対して、広く用いられている業界標準のサービス指向的解釈を行う上で、Web サービス・アーキテクチャがもっとも効率的な道筋であるとの信念にも基づいている。

インフラストラクチャやフレームワークとして Web サービスを選択することは、OGSA システムやアプリケーションがサービス指向アーキテクチャの原則に基づいて組み立てられることを前提としていること、そしてサービスインタフェースが Web サービス記述言語（WSDL）によって定義されることを、それぞれ意味する。現在のところ WSDL1.1 を想定しているが、計画中の WSDL2.0 の仕様が定まった時点で WSDL2.0 に移行する予定である。また、記述と表現の共通語としては XML を、OGSA サービスの主要なメッセージ交換フォーマットとしては SOAP を想定している。（ただし、パフォーマンスが重要になる場面など、コンテクストによっては別の表現が必要になることもあると認識している。）さらに、WS 相互運用性（WS-I）プロセスを通じて定義される相互運用性プロファイルと整合性のあるサービスの定義を作る必要がある。

このように Web サービスの枠組みの中で作業するものの、現在定義されている Web サービスの規格は、グリッドのすべての要件を満たすものではなく、またそれらの要件を満たすように設計されているわけでもないことは明らかである。場合によっては、既存の仕様に修正や拡張が必要になることもあるだろう。そのため OGSA のアーキテクトは、WSDL2.0 の定義に深く関わり、WS セキュリティとその関連仕様を詳しく検討している。本文書では、既存の仕様に拡張が必要な他の箇所について指摘を行っている。別のケースでは、グリッドの要件により、サービスの定義を完全に新しく導入しなおす必要がある。

グリッド要件により既存の仕様を拡張する必要のある主要分野の 1 つが、セキュリティである。セキュリティの問題は、OGSA インフラストラクチャのさまざまなレイヤにおいて発生している。認証、認可、メッセージ保護の目的で、OGSA のサービス・リクエストに適切なトークンを安全に運ばせるため、WS セキュリティ標準プロトコルを使用している。OGSA のインフラストラクチャが扱うシナリオはエンドツーエンドのメッセージ保護を必要とし、したがって OGSA は、TLS や IPsec などのポイントツーポイント伝送レベルのセキュリティに加えて、あるいは取って代わって、XML 暗号化やデジタル署名などのより高

いレベルの保護メカニズムを提供する必要がある。相互運用可能で組み合わせが可能なインフラストラクチャは、メッセージレベルのセキュリティに加え、それ自身がサービスとして提供されるセキュリティ・コンポーネントが必要になる。たとえば、OGSA の認可サービスは、セキュリティ・アサーションやアクセス制御記述を表現するために、SAML や XACML などの発展を続ける OASIS[1] 規格とともに、提案中の WS アグリーメント規格を使用する可能性がある。適切な時と場合に応じて、OGSA はこれらのセキュリティサービスを採用、定義する。

グリッドの要件が新たな仕様（グリッドのシナリオを超えた関係を持つ仕様）を必要とする主要分野は、ステートの表現と操作に関する分野である。とくに、オープン・グリッド・サービス・インフラストラクチャ（OGSI）[OGSI]のリファクタリングである WS リソース・フレームワーク（WSRF）[WS-RF]によって定義されるインタフェースや挙動を構成単位であると考えることにする。WSRF は、ステートのモデル化や管理、ステートへのアクセス、あるいはサービスのグループ化、フォルトの伝達などに対するアプローチを定義する。これらのメカニズムは、いずれもグリッドシステムを構築する上で中心的役割を果たすものである。さらに WSRF は、OASIS WSDM 技術委員会などのリソース・モデリングやシステム管理に関する規格作りにおいて、活用されているし、活用を考えられている。そのため WSRF の上に構築された OGSA 関連の規格は、これらの規格本体の取り組みと折り合う上で、ちょうどいい位置にあるといえよう。

最後に、通知機能やイベンティング機能については、WS 通知仕様[WS-N]の範囲内で定義されたものとする。これらの仕様は、WSRF 同様、OGSI に由来するものであるが、ステート・コンポーネントに対するサブスクリプションやその後の変化に関する通知をサポートするために、WSRF メカニズムに立脚したノーティフィケーション・メカニズムを定義するものである。（WS-イベンティング仕様も類似のメカニズムを提供するが、WSRF へのつながりがまだない。）

3.4 実行管理サービス

実行管理サービス（OGSA-EMS）は、一連の仕事のインスタンスの作成や実行管理を行う問題に関係したものである。一連の仕事の例としては、OGSA アプリケーションあるいはレガシ（非 OGSA）アプリケーション（データベースサーバや Java アプリケーションサーバ・コンテナで動いているサーブレット等）のいずれかが含まれる。

3.4.1 目的

EMS が扱う問題のいくつかを、以下で例示する。アプリケーションにはキャッシュサービスが必要である。それには既存のサービスを使用すべきか、あるいは新しいサービスを作成する必要があるだろうか。新しいサービスを作るのであれば、どこに置くべきだろうか？どのようにそのキャッシュサービスを設定すべきだろうか。キャッシュサービスには

どのようにして適切なリソース（メモリ、ディスク、CPU など）を与えるべきか？そのキャッシュサービスは、どのような種類のサービス・アグリーメントを作ることができるだろうか。どのような種類のアグリーメントを必要とするだろうか。

同様に、ユーザがレガシ・プログラムを作動させたいとしよう。バイオインフォマティクス、電子設計オートメーション、天気予報、航空宇宙産業、金融サービス、その他多くのさまざまなアプリケーション分野や産業において、入力データやパラメータを（多くの場合ファイルの形で）取り込み出力を生み出す、ある種のアプリケーションがある。これらのアプリケーション例の多くは、「エンバラシングリ・パラレルのパラメータ・スイープ」と呼ばれる分野で動かされている。バイオインフォマティクスにおけるよい例が、BLAST である。BLAST は DNA やタンパク質の配列をターゲットとなるデータベースと比較し、類似度を表示するアプリケーションである。航空宇宙分野の例としては、航空シミュレーションで使用される計算流体力学（CFD）のプログラムである Overflow がある。そうしたレガシ・アプリケーションを実行する際には、次のような問題に対処しなければならない。プログラムをどこで走らせるか。データファイルと実行ファイルをどうやって実行場所にステージングするか。実行に不具合が生じた時に何が起きるか。実行は再試行できるか。再試行できるならばどうやればよいのか。以下では、BLAST を標準的な例として使用することにする。

より形式的には、EMS は、作業単位の配置、「プロビジョニング」、作業終了までの管理など、作業単位の実行に関連する問題を扱う。これらには以下の問題が含まれる。ただし、以下の問題がすべてではない。

- 実行場所候補の検索：ある作業単位が、メモリ、CPU、バイナリタイプ、使用できるライブラリやライセンスなどのリソース上の制約を満たす場合、その実行場所はどこであるか。その場合、実行場所の候補を絞るポリシー上の制約には、どのようなものがあるか。
- 実行場所の選択：作業単位を実行「できる」場所がわかった場合、それをどこで実行「すべきか」が問題になる。この問題の解答には、異なる目的関数を最適化したり、異なるポリシーやサービスレベル・アグリーメントの施行を試みるさまざまな選択アルゴリズムが関係する。
- 実行準備：作業単位がどこかで実行されるからといって、それがセットアップなしに実行できるとは限らない。セットアップには、ローカルな実行環境を整えるための、バイナリやライブラリのデプロイメントやコンフィグレーション、あるいはデータのステージングやその他の作業が含まれる。
- 実行開始：すべての準備が整うと、実際に実行を開始し、それを適当な場所に登録するなど、他の関連アクションも実施する。
- 実行管理：実行が始まると、終了するまでは管理、監視が必要となる。実行ができない場合はどうなるか。アグリーメントと合致しない場合はどうか。別の場所で再実行する

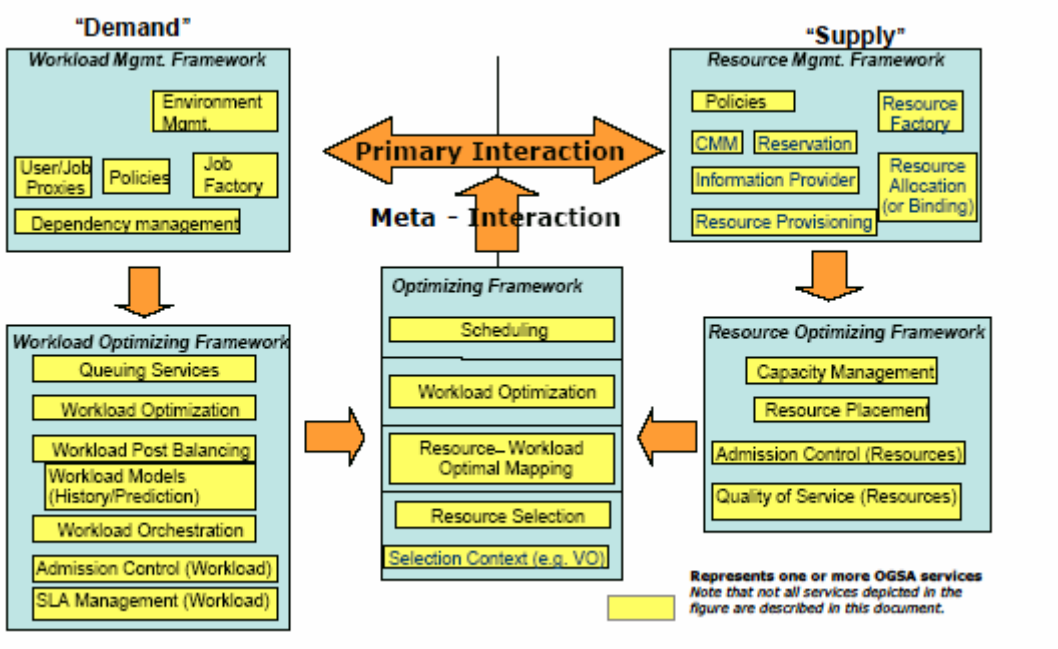
べきか。ステートはどうか。再起動性を確保するために定期的にステートをチェックすべきか。その実行は、なんらかのフォルト検出スキームおよびリカバリスキームに組み込まれているか。

以上が EMS が取り扱う主要な問題である。上に見たように、あらゆる範囲のタスクが関係し、他の OGSA 機能が定義することになっている多くの他の OGSA サービス（たとえばプロビジョニング、ロギング、レジストリ、セキュリティなど）との相互作用が絡んでくる。詳しくは 3.4.7 節を参照のこと。

EMS が重要になるのは、単純に静的環境を想定して UDDI などのレジストリを使用することができないためである。数多くの設定とともに使用されるグリッドでは、使用できるリソースの集まりやその負荷が極めて変化しやすく、高いレベルの信頼性を必要としているということが想定される。たとえば、動的にプロビジョニングされる計算環境では、あるアプリケーションが使用するリソース集合は時間とともに変化することがあり、アプリケーションの要件やサービスレベル・アグリーメントを満たすためにリモートリソースを一時的に使用する必要性も出てくることがある。同様に、予期せぬ故障に対応したりサービスレベルの保証を満たすには、使用可能なりソースを見つけ出し、それらを再実行させることが必要になることもある。アプリケーションのニーズを監視し、アプリケーションが終了するまで動的にそのニーズに対応する必要性が、共通のテーマである。

3.4.2 アプローチ

ソリューションは、EMS の問題を置換可能なコンポーネントしたサービスから成り立っている。各使用例は、これらのサービスのうち別々の部分を用いて目的を達成している。しかし一般的には、EMS が供給側と需要側から構成されていると考えることができる。供給側は、CPU、ディスク、データ、メモリ、サービスなどのリソースを提供する。需要側はそれらのリソースを使用する。ワークロード管理システムやワークフローシステムなどのツールは、需要側にある。リソースを管理、提供し、ポリシーやサービス・アグリーメントを実施するサービスは、供給側にある。図４は一般的な構図を描いたものである。



（図示したサービスのすべてが本文書に記載されているわけではないことに注意。）

図４：EMS の概念的メタモデルにより全体を需要側と供給側に分割

EMS サービスにより、アプリケーションが基盤となるリソースに対し、その物理的場所やアクセスのメカニズムによらず、協調アクセスすることができる。EMS サービスは、使用可能なリソースとグリッドアプリケーションの要件を自動的に満たしてエンドユーザがリソースに容易にアクセスできるようにする上で、重要なものである。

EMS は協調して動作する数多くのサービスから成り立っている。これらのサービスを以下で解説する。解説に進む前に、いくつかの注意とコメントを列記しておく。

まず、すべてのサービスが常時使用されているわけではない。グリッドの実装によってはいくつかのサービスを必要としなかったり、あるいはいくつかのサービスを別のサービスに取り込むことで直接利用しないこともある。私たちは一般に、この機能をこれまで異なるグリッドの実装で何度も目にしてきた機能に関連するサービスに分解しようとしてきた。これらの機能とはすなわち、通常使用する必要がない機能というだけでなく、１つあるいは複数の異なる実装がインターフェースに求められる機能である。

第二に、ここで示す定義は概要にすぎない。本文書の目的はサービスを完全に定義することではなく、むしろ、重要なコンポーネントとその間のより高いレベルの相互作用を確認する意図がある。

第三に、これらの定義とサービスが、単にレガシ“ジョブ”の実行だけでなく、一般的な Web サービスの実行に適用できる点を強調したい。

最後に、本文書では「リソースハンドル」の存在を仮定している。リソースハンドルとは、リソースとそれに関連したステート（ステートがある場合）に対する抽象名（3.9.4.1

節および 3.4.7.2 節参照）である。さらに、リソースハンドルを「リソースアドレス」に結びつけるメカニズム（その定義は本文書の範囲外）が存在するものと仮定する。リソースアドレス（3.4.7.2 節および 3.9.4.1 節のアドレスを参照）は、リソースと通信する必要のあるプロトコル固有の情報を含んでいる。以下では、リソースハンドルとして RH、リソースアドレスとして RA という記号を用いる。

3.4.3 EMS サービス

EMS サービスには大きく分けて以下の３つの分類がある。

- プロセシング、ストレージ、実行ファイル、リソース管理、プロビジョニングを形成するリソース
- ジョブ管理とモニタリング・サービス
- 作業単位を実行する場所を集団的に決めるリソース選択サービス

さらにデータ管理サービス（3.4 節）、セキュリティサービス（3.6 節）、ロギングサービス（3.8.3.3 節）が利用できるものと仮定する。これらのサービスとの相互関係については、本文書の今後の版で作成予定である。

3.4.4 リソース

3.4.4.1 サービスコンテナ

サービスコンテナ（以下ではたんに「コンテナ」と呼ぶ）は、動いているエンティティを「包み込む」ものである。エンティティは、「ジョブ」（以下で解説）であっても、作業中の Web サービスであっても構わない。コンテナの例としては、キューイング・サービスや UNIX ホスト、J2EE ホスティング環境、あるいはコンテナの集合（ジョブコンテナのファサードや VO）などがある。コンテナには、受け入れ可能な実行ファイルの種類、OS バージョン、インストールされたライブラリ、ポリシー、セキュリティ環境などの静的情報と、負荷や QoS 情報などの動的情報を記述するリソース特性が含まれている。

コンテナは、WSDM に管理されたリソースのマネージャビリティ・インタフェースの一部を実装している。基本サービスコンテナを超えてさらなるサービスを提供する拡張インタフェースが期待される。

コンテナは、クライアントの目に触れる他のリソースとのさまざまな関連性を持っている。たとえばコンテナには、データコンテナとの互換的な関連性が含まれることがある。この互換的な関連性は、あるコンテナ内で作動しているエンティティが特定のデータコンテナ内の永続的なデータにアクセス可能であることを示すものである。他の管理されたリソースとしては、デプロイされたオペレーティングシステムや物理的ネットワークなどがある。

最後に、コンテナは、予約サービス、ロギングサービス、情報サービス、ジョブ管理サービス、プロビジョニングサービスを利用すると考えられる。

3.4.4.2 永続的なステート・ハンドル・サービス（PSHS）

永続的なステート・ハンドル・サービス（PSHS）は、永続的なステートの「場所」を把握するものである。このようなサービスは、ファイルシステム、データベース、階層型ストレージシステムなどを含むさまざまな方法で実装される。PSHS は、扱う永続的なステートに対する「リソースハンドル」（RH）を取得するメソッドを持っている。RH の形態は、そのステートの実際の格納方法による。永続的なステートの「リソースアドレス」（RA）は、ファイルシステムのパス名であったり、データベース内のプライマリキー値であったりする。重要な点は、データに直接アクセスするために RA を利用することができるということである。

PSHS は、WSDM に管理されたリソースのマネージャビリティ・インタフェースを実装している。基本データコンテナを超えてさらなるサービスを提供する拡張インタフェースが期待される。PSHS はまた、所有する RH を管理するメソッドを持っている。その中には、RH を別の PSHS に渡すことも含まれる。これにより、マイグレーションと複製の両方が可能になる。

PSHS に対する別の見方としては、マウントポイント、データベースキー、パスなどの固有のメカニズムを使ってデータ効率をあげる方法について情報を提供するメタデータ・レポジトリが PSHS であるとみなすことができる。

PSHS は、データサービスではない点に注意が必要である。PSHS は、実行中のエンティティのステートがどこに保管されているかを記録する手段であり、それにより必要であれば素早くアクセスすることができるのである。

3.4.5 ジョブ管理

3.4.5.1 ジョブ

OGSA-EMS によるジョブの定義では、従来のジョブの概念を取り込み、拡張している。ジョブは、特定の作業単位（すなわち、実行中の BLAST のようなアプリケーションやサービスのインスタンス）について知っておくべきことすべてをカプセル化している。ジョブは管理対象となる最小単位である。ジョブは、作業単位のマネージャビリティ・アスペクトを表すが、実際に動いているアプリケーションや作業単位の実行アスペクトとは異なる。（ただし、本文書では「ジョブ」という言葉を「作業単位」の代わりに非公式に用いることがある。たとえば、「ジョブをサブミットする」「ジョブを実行する」といった表現で用いる。）

ジョブは、WSDM が管理するリソースのマネージャビリティ・インタフェースの一部を実装している。ジョブは、明確なリソースハンドル（RH）で名前が付けられる。ジョブは、リソースが全くコミットされていない場合でも、リクエストされた瞬間に作成される。ジョブは、実行ステート（たとえば開始、中断、再起動、停止、終了などのステート）、リソ

ースのコミットメントとアグリーメント、ジョブ要件などの記録を取っており、その多くはジョブ・ドキュメントに保管されている。

ジョブ・ドキュメントは、ジョブのステートを記述するものである。すなわち、サブミッションの記述（JSDL[JSDL]）、得られたアグリーメント、そのジョブステータス、ユーザのメタデータ（クレデンシャル等）、ジョブの起動回数などが記載されている。ステートには、実行アプリケーションプログラムの内部メモリといったアプリケーション固有の詳細は含まれない。

ジョブ・ドキュメントは、ジョブのリソース・プロパティとしてエクスポーズされる。その論理的ビューは、１つあるいは複数の（もしくは多数の）サブドキュメントから構成される大きなドキュメントに対するものである。これらのサブドキュメントは、別々に取り出すことができる。サブドキュメントの編成は、さらなる仕様によって決められる。

3.4.5.2 ジョブマネージャ

ジョブマネージャ（JM）は、１つのジョブあるいは一組のジョブ実行を最初から最後まで行うために必要な段階をすべて隠蔽する、より高いレベルのサービスである。一組のジョブを（ワークフローや依存グラフのように）構造化することも（相互関係のないジョブのアレーのように）非構造化することもできる。JM は、ユーザと相互作用し、ユーザ代表としてジョブを管理するポータルと見ることもできる。

JM は、実行計画サービス（3.4.6.1 節参照）、デプロイメントシステムとコンフィグレーションシステム、コンテナ、モニタリング・サービスと関係する可能性が高い。さらに JM は、不具合や再起動を取り扱うことができ、リソースへのジョブのスケジューリングや、アグリーメントや予約を集めることもできる。

JM は、管理可能なエンティティから成る WSDM コレクションのマネージャビリティ・インタフェースをおそらく実装することになる。WSDM コレクションは、その構成要素が公開するメソッドのうちのいくつかを、コレクションのメソッドとして公開することができる。

１つのジョブあるいは一組のジョブを起動する際に使用するサービスをオーケストレーションする場合、JM が関与することになる。たとえば、アグリーメントをネゴシエートしたり、コンテナと相互作用したり、モニタリングやロギングサービスをコンフィグレーションするなどして JM が関わる。JM はまた、JM が管理するジョブの集まりからジョブリソース・プロパティを集約することもある。

JM の例を以下にあげる。

- 「ジョブ」を受け、優先順位を付け、処理のために異なるリソースへ分散させる「キュー」。（JobQueue[JobQueue]や Condor[Condor]に類似したもの。）この JM は、ジョブの記録を取り、ジョブに優先順位を付ける。そして、QoS のファシリティや未実行ジョブの最大数、ジョブを配置するサービスコンテナの集合体を持っていることがある。

- ジョブのデータや要件を集め、ジョブのスケジューリングを行い、結果を返すために、エンドユーザとやりとりするポータル。
- ジョブの記述、QoS の要件、その要件間の依存関係、初期データセット（初期マーキングを使ったデータフローグラフと考えればよい）を受け取り、おそらくはいくつもの不具合をくぐり抜けてワークフローを完結させるためにスケジューリングや管理を行うワークフローマネージャ。（この場合、ノードもワークフロー・ジョブマネージャの 1 つである。）（DAGman[2] の一部と似た概念である。）
- パラメータが若干異なるだけのに多様なジョブの集まりを受け取り、終了まで管理するアレー・ジョブマネージャ。例としては Nimrod[Nimrod]がある。

3.4.6 選択サービス

3.4.6.1 実行計画サービス（EPS）

実行計画サービス（EPS）は、ジョブとリソースの間で「スケジュール」と呼ばれるマッピングを構築するサービスである。スケジュールは、サービスとリソース間のマッピング（リレーション）であり、時間制限がついている場合が多い。スケジュールは、その代わりとなるいくつかの「スケジュールデルタ」に拡張されることがある。スケジュールデルタは、基本的に「スケジュールのこの部分がうまくいかなかった場合は、代わりにこれを試してください」と呼びかけるものである。

EPS は通常、実行時間、コスト、信頼性等の目的関数を最適化しようとする。EPS がスケジュールを定めることはなく、単にスケジュールを生成するだけである。スケジュールの制定は、通常、JM が行う。EPS は、情報サービスや候補セット・ジェネレータ（CGS、以下参照）を使用する可能性が高い。たとえば、はじめに CSG を呼び出して一組のリソースを取得し、次にそれらのリソースに関する現在の情報を情報サービスからさらに取得し、そしてスケジュールを作るために最適化関数を実行するのである。

3.4.6.2 候補セット・ジェネレータ（CSG）

CSG の基本概念はきわめて単純である。CSG は、作業単位を実行するリソース・セットを決めるものである。すなわち、「どこで実行されるか」ではなく、「どこで実行することができるか」を決定するのである。そこには、どのようなバイナリが使えるかという問題や、アプリケーションの特別な要件（たとえば 4GB メモリや 40GB テンポラリ・ディスクスペース、○○ライブラリのインストール）、セキュリティや信頼性の問題など、さまざまな問題が出てくることがある。（セキュリティや信頼性の問題としては、たとえば「ピュアコンピューティング協会がリソースをグレード A＋と認定しない限り、私のジョブをそのリソース上で走らせたくない」、「私のバイナリが安全であると保証されない限り、そこで走らせてもらえない」、「私のクレジットカードは使えますか？」などの例がある。）

候補セット・ジェネレータ（CSG）は、RH と名付けられたジョブを走らせることのでき

るコンテナ・セット（より正確に言えば、コンテナの RH）を生成する。検索対象とするリソースは、特定のサービスに対してデフォルト設定にすることもできるし、パラメータとして与えることもできる。

CSG はおもに、EPS によって、あるいは EPS 関数を実行する JM などの他のサービスによって呼び出されることになっている。CSG は、情報サービスを使用し、ジョブ・ドキュメントの中から適切な箇所を取り出すためにジョブへアクセスし、プロビジョニングとコンテナサービスと相互作用して特定の実行のためにコンテナを設定できるかどうかを決める。

3.4.6.3 予約サービス

予約サービスは、リソースの予約管理、会計サービス（予約を行うのに料金がかかる場合がある）との相互作用、予約の取り消しなどを行う。これは独立したサービスというよりは、コンテナや他のリソースからの予約を受けて管理するためのインタフェースというべきものである。予約自体は、署名したアグリーメント・ドキュメントであることが多い。

予約サービスは、グリッド上の予約可能なあらゆる種類のリソースに対して共通のインタフェースを提示する。予約可能なリソースには、CPU やメモリなどの計算リソース、可視化のためのグラフィックパイプ、ストレージスペース、ネットワーク帯域幅、特殊目的の機器（電波望遠鏡など）等がある。ただしこれだけに限定されるわけではない。

予約は、リソースブローカがネゴシエートやリセルを行う可能性があることから、より低いレベルの予約の集まりに対する集合体と考えることができる。

一般的に、多くの異なるサービスが予約サービスを使用する。たとえば JM は、管理しているジョブの集団に対して予約をすることがある。あるいは EPS は、特定のジョブの実行計画を保証するために予約を利用することがある。予約の作成がジョブに対するプロビジョニング・ステップと関連している場合もある。

3.4.7 OGSA の他の部分との相互作用

この節では EMS と、OGSA の他の部分との相互作用について詳述する。

3.4.7.1 デプロイメントとコンフィグレーションのサービス

作業単位がサービスやデータコンテナを使用する前に、それを設定、あるいは別のリソースとともにプロビジョニングする必要がある。たとえばあるホスト上で BLAST を走らせる前に、BLAST の実行ファイルとコンフィグレーション・ファイルがそのホストにアクセス可能であることをユーザが確認する必要がある。さらに詳しい例で言えば、複雑なアプリケーションのコンフィグレーションや適切なデータベースのインストール、あるいは、あるホストを計算リソースとして使用するための最初の段階としてそのホストに Linux をインストールすることもそうである。

3.4.7.2 ネーミング

OGSA-EMS は、OGSA のネーミングを使用する。これについては 3.9.4.1 を参照のこと。たとえば、アベイラビリティやロード・バランシングのためにチェックポイントや再起動機能を持った高性能のジョブ・キューイング・システムにおいては、ジョブのアドレスにより特定のマシンにおけるジョブの場所が指定される。抽象名は、場所とは無関係に普遍的な方法でジョブを特定する。たとえば抽象名は、ジョブのマイグレーションの前後で同じ名前でなければならない。ヒューマン指向の名前とは、それが使われるコンテクストを言及することであいまいさをなくすことができるユーザフレンドリな短いジョブ名であるかもしれない。

3.4.7.3 情報サービス

端的にいえば、情報サービスはリソースに関する属性メタデータのデータベースである。EMS では、情報サービスは多くの異なるサービスによって使われている。たとえばコンテナは属性に関する情報を公開し、それにより CSG サービスがジョブのコンテナが適切なものであるかどうかを評価することができる。また、EPS は情報サービスから VO に関するポリシー情報を読むことがある。さらに、情報サービスを使って PSHS 自体を実装することもできる。情報サービスがどのように情報を取得するかは指定されていないが、「新しさ」がデータの属性の 1 つであると考えている。この点で、OGSA の情報サービスは、Globus の MDS サービス[Globus MDS]や Legion[Legion]における集合体と同様のものである。

3.4.7.4 モニタリング

何かを単にスタートさせ、あとは放っておくだけでは、不十分な場合がある。アプリケーション（多くの異なるサービスやコンポーネントを含むこともある）は、多くの場合、継続的に監視する必要がある。これはフォルトトレランスや QoS などの理由によるものである。たとえば、スケジューラにあらかじめ自分を選ぶようにさせてあるホストの場合、そのホストに関する条件が変わることもある。その際、スケジューリングをやり直す必要性が生じる。

3.4.7.5 フォルト検出とリカバリのサービス

フォルト検出とリカバリのサービスは、モニタリングの一部である場合とない場合があり、トレードオフ・パフォーマンスとリソースの利用を可能にするステートレスな機能のため、簡単なスキームを管理するためのサポートも含んでいる。またこのサービスには、シングルスレッドの（プロセス）ジョブのチェックポイントやリカバリを管理するやや複雑なスキームや、MPI ジョブのような分散型ステートのアプリケーションを管理するさらに複雑なスキームも含まれている。

3.4.7.6 会計検査、課金、ロギングのサービス

会計検査、ロギング、課金のサービスは、OGSA が成功するかどうかにかかわる重要な要素である。そこには料金を設定するためにスケジューラがリソースとやりとりする機能や、リソースが会計や課金のサービスとやりとりする機能が含まれる。たとえば、ロギングがこれらすべての基礎にあるとすると、

- メータリングは、そのログを使ってリソースの使用を記録する。
- 会計検査は、そのログを永続的に、そしておそらくは否認不可の形で活用する。
- 課金は、OGSA からは定義されない別のサービスであり、請求書や不渡りを作成するために、会計検査を利用することや、さらには（または）ログやその他のデータのメータリングを利用することがある。

3.4.7.7 会計

リソースによっては、クレジットカードのように、支払いに関してユーザを十分信頼できるかどうか見定める必要がある。スケジューラは、コンテナなどの特定のリソースと同様に、会計サービスとやりとりをする必要がある。

3.4.8 シナリオ例

以上のようなサービスをもっともよく理解する方法は、その具体的な使われ方を見ることである。ここでは例として、システムパッチツール、データキャッシュサービスのデプロイメント、レガシ・アプリケーションの実行の 3 つのケースを選んで紹介する。

3.4.8.1 ケース 1：システムパッチツール

オペレーティングシステムのパッチやライブラリのアップデートを多数のホストに対して実施する必要に駆られることがよくある。これにはいくつかの方法がある。よく使われる方法の 1 つは、システムの各ホスト上でスクリプトを走らせ、適当なファイルをコピーさせるのである。スクリプトの起動には、rsh や ssh などのツールがよく用いられるが、こうしたスクリプトはシェルスクリプトと呼ばれ、パッチをあてる一連のホスト常で繰り返し適用される。別の方法としては、ホストが自ら定期的にアップデートが必要であるかどうかをチェックし、必要であるならば何らかのスクリプトを走らせて OS をアップデートさせるのである。

EMS を使うと、こうした作業をいろいろな方法で行うことができる。OS バージョン番号がコンテナに保持されたメタデータであり、情報サービスにより収集されるものとしよう。また、作業の目的はパッチを一部しか持っていないすべてのオペレーティングシステムにパッチをあてることにあるものとする。この場合、おそらく最も簡単に問題に対処する方法は、あるしきい値以下の OS バージョン番号を持ったコンテナのリストに対して、最

初に情報サービスをクエリすることである。そしてリスト内の各コンテナ上でパッチサービスを走らせるよう、ジョブマネージャ（JM）に指示するのである。この場合 JM は、どこでサービスを走らせるべきなのかわかっているので、実行計画サービス（EPS）とやりとりする必要がない。その代わりに、JM は各コンテナと直接やりとりする。さらにおそらくは、パッチサービスをコンテナ上で実行するためにデプロイメントやコンフィグレーションともやりとりする。（デプロイメントとコンフィグレーション・サービスは、最初にパッチサービスをインストールする際に必要となることがある。）

3.4.8.2 ケース 2：データキャッシュサービス

データキャッシュサービスは、多数のクライアントの代表としてデータ（ファイル、実行ファイル、データベース・ビュー等）をキャッシュし、そのプライマリコピーとの一貫性を保とうとするものだとしよう。あるクライアントがキャッシュサービスを要請した場合、そのクライアントの場所、既存のキャッシュの場所やロードなどに応じて、ハンドルを既存のキャッシュサーバに送ることもできるし、あるいは新しいキャッシュサーバを作ることもできる。

新しいキャッシュサービスのインスタンスを作成することにした場合は、データキャッシュの置き場所を決めるために EPS が起動される。EPS はそのサービスを走らせることのできる場所を探すために CSG を利用する。その際、クライアントへのローカリティの概念により制約を受ける。場所が決まれば、キャッシュサービスのインスタンスがコンテナ上で作成されることになる。

3.4.8.3 ケース 3：レガシ・アプリケーション

3 例目は、やや典型的なシナリオである。あるユーザがレガシである BLAST ジョブを走らせたいと思っているとし、そのユーザがポータルあるいはキュー・ジョブマネージャとやりとりしているものとしよう。BLAST ジョブを始めるには 4 つの基本的なステップがある。

1．ジョブ定義の段階：入力ファイルは何か、サービスレベルの要件は何か、（例：ジョブは明日の正午までに終了しなければならない。）そのジョブに関してはどのアカウントに課金するのか、などである。
2．使用できるリソースを見つけ、ジョブの実行に必要なリソースを選び出す。
3．スケジュールやそれに関連する事柄（リソースのプロビジョニングや会計など）をすべて決める。
4．ジョブが終了するまで監視する。何らかの理由でジョブが終了しなかった場合、サービスレベルのアグリーメントによっては、ジョブの再実行が必要になる。

EMS を使ってこの例を実現するため、JM は（たとえば JSDL[JSDL]で書かれた）適切なジョブ記述により新たなレガシ・ジョブを作成する。つぎに JM はスケジュールを手に

入れるために、EPS を呼び出す。EPS はつぎに CSG を呼び出す。CSG は、ジョブの実行場所をバイナリ・アベイラビリティとポリシーの設定に基づいて決めるために情報サービスを呼び出す。EPS は最初にその情報サービスが正確であることをサービスコンテナに確認し、そのサービスコンテナを選択する。EPS はスケジュールを JM に返す。JM はつぎにジョブの実行環境を整えるために、必要であれば予約サービスや、デプロイメント、コンフィグレーションなどのサービスとやりとりする。その際、データコンテナとのやりとりも含まれることがある。ジョブを開始するためにサービスコンテナが起動される。会計や会計検査の追跡には、ロギングサービスが使われる。ジョブが停止すると、コンテナがジョブマネージャに通知する。ジョブの停止が異常であれば、全サイクルを再実行することもある（図 5 参照）。

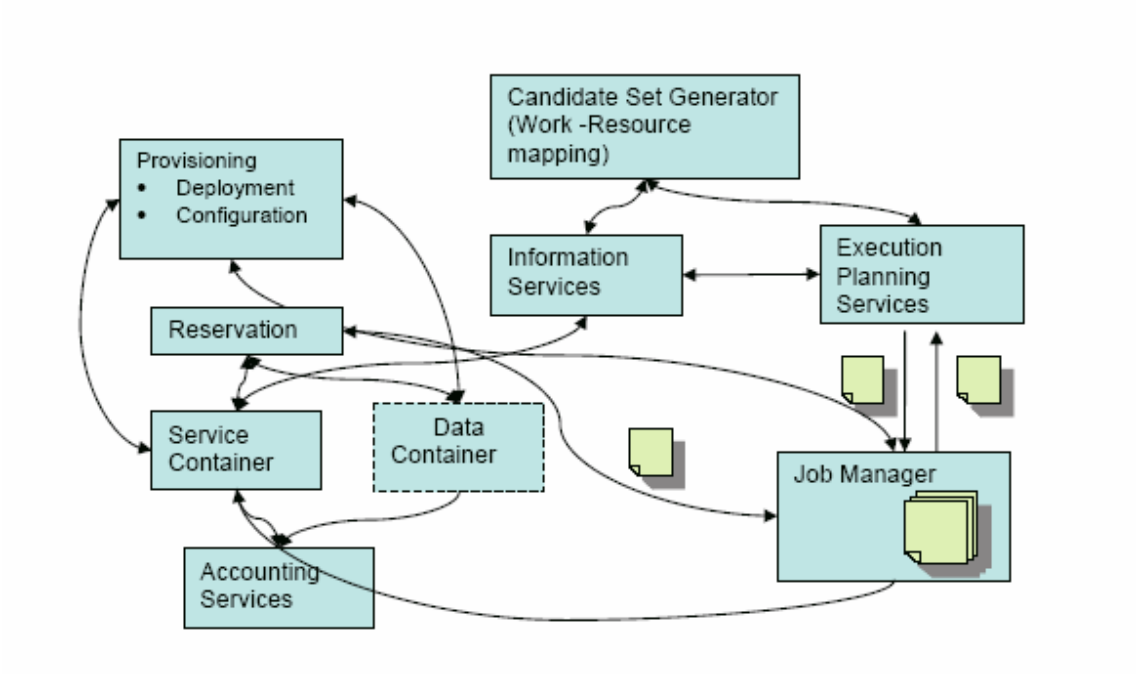


図 5：レガシ BLAST ジョブの実行における EMS サービスの相関

3.5 データサービス

OGSA のデータサービスは、データリソースの移動、アクセス、アップデートに携わっている。

3.5.1 目的

データサービスは、データを必要な場所に移動させたり、複製したコピーの管理、クエリやアップデートの実行、データを新しいフォーマットに移行させるなどの作業を行う際に使用する。データサービスはまた、OGSA のデータサービスや他のデータ、とくにデータそのものの起源を記述するメタデータの管理に必要な機能を提供する。

たとえば、実行管理サービス（EMS）がどこかに保存されているデータにアクセスする必要が生じたとしよう。EMS はデータサービスを使ってそのデータにリモートでアクセスするのだろうか。それともローカルのマシンにコピーをステージングするのだろうか。EMS はデータの一部をローカルにキャッシュするだろうか。そのデータは複数の場所で使用できるだろうか。データサービスはどのようなポリシー制限やサービス保証を提供するだろうか。

逆に、データ・フェデレーション・サービスがいくつかの異なる場所に保存されたデータに対してスキーマを定義する場合を考えてみよう。このスキーマに対するクエリは、基礎となるリソースにどのようにマップすればいいのだろうか。データの結合や移行はどこで行えばいいのだろうか。データはどこに送ればいいのか。どのようなサービス品質が保証されているのだろうか。

ある種のメタデータを除いて、こうしたデータ機能はあらゆる特定のデータに意味を規定することはない。データサービスを利用する OGSA の他のサービス（情報サービスなど）は、独自の意味を付け加えることがある。

3.5.2 モデル

3.5.2.1 データリソースの種類

データリソースは、データのソースとシンクとして機能するエンティティを指す。グリッドの異機種混合的な性格により、数多くの異なる種類のデータをサポートしなければならない。それには以下のようなものも含まれる。ただし以下がすべてを網羅しているわけではない。

- フラットファイル：もっとも単純なデータ形態は、長さの決まったレコードのように、アプリケーション固有の構造を持ったファイルである。こうしたファイルには、通常のリードとライトの動作でアクセスできる。いくつかのファイルフォーマットは、データベースのようなクエリをサポートしている。たとえば、リレーショナル・テーブルのようにクエリを行うことのできるカンマで区切られた数値や、XML クエリ[XQuery]を使ってクエリを行うことのできる XML ファイルなどもその一例である。データアクセスサービスはこれらをサポートしており、新しいファイルフォーマットに対する特別なクエリもサポートするよう拡張することもできる。
- ストリーム：無限長のシーケンスから構成することのできるデータ値を、ストリームと呼ぶ。データアクセスサービスは、ストリームに対するクエリや移行をサポートしている。
- DBMS：数種類のデータベース管理システムがグリッドの一部となることがある。これには、リレーショナル・データベース、XML データベース、オブジェクト指向データベースなどが含まれる。
- カタログ：カタログは他のデータサービスを構造化し、追跡するものである。カタログ

の簡単な例が、ファイルの集合をリストアップするディレクトリである。ネスト化したディレクトリは、階層ネームスペースと同等のものである。

- 派生：いくつかのデータは、他のデータに対する非同期のクエリや移行の結果である。これらの派生は、単独のアイテムとしてではなく、有限のストリームのように扱われることが多い。
- データサービスそのものが、データを生成するセンサデバイスやプログラムのように、別のサービスのデータリソースとなる場合がある。

3.5.2.2 シナリオ例

データサービス機能は、グリッドの根幹に関わるものである。本節ではデータサービスの機能が幅広い作業をサポートするものとしていかにして利用されているかについて、いくつかの例を紹介することにする。

- リモートアクセス：OGSA のデータサービスのもっとも単純な利用例は、グリッドを横断してリモートデータリソースにアクセスすることである。データサービスは通信メカニズムをクライアントから隠し、必要であればリモートデータの正確な所在地も隠す。これを最適化するには、ローカルリソースにデータの一部をキャッシュする必要がある。
- ステージング：リモートソース上でジョブが実行される場合、ジョブを実行する準備の整ったリソースに入力データを移動し、さらにそのジョブの結果を適切な場所に移動するため、データサービスを利用することが多い。
- 複製：アベイラビリティの向上とアクセス時間の短縮のため、同じデータをグリッド内の複数の場所に保存することができる。
- フェデレーション：OGSA のデータサービスでは、別々に作成、維持されてきた複数のデータソースからデータを取り込む仮想データリソースを作ることができる。クライアントが仮想リソースをクエリした場合、そのクエリはサブクエリとオペレーションにコンパイルされる。これらは、基盤となるフェデレートされたリソースから適切な情報を引き出し、適切なフォーマットでその情報を返すものである。
- 派生：OGSA のデータサービスでは、あるデータリソースから別のデータリソースを自動的に生成することができる。
- メタデータ：OGSA のデータサービスや他のデータなどを記述するデータは、グリッドに欠かせないものである。もっとも単純な形態のメタデータは、OGSA のデータサービスに対して別の使い方をするだけである。さらに OGSA では、データとメタデータの間のリンクを維持するためのサポートも提供している。

3.5.2.3 供給と需要のメタモデル

データサービス・アーキテクチャは、OGSA の他の部分同様、供給側と需要側から成り立っていると見ることができる。供給側は、ファイル、データベース、ストリーム、サー

ビスといったデータリソースを提供する。さらに以下で議論するようなリソース固有のインタフェースや仮想化のインタフェースも提供する。データの移行やメタデータをサポートするサービスや、ポリシーやサービスのアグリーメントを施行するサービスも、供給側のものである。これに対して需要側には、アクセスリクエストを生成、最適化するサービスや、サービスレベル・アグリーメントを管理するサービスなどがある。データの場所を複製したり、フェデレートや最適化を行うサービスは、供給側と需要側とを仲介するものである。以下の図６にこの関係を示す。

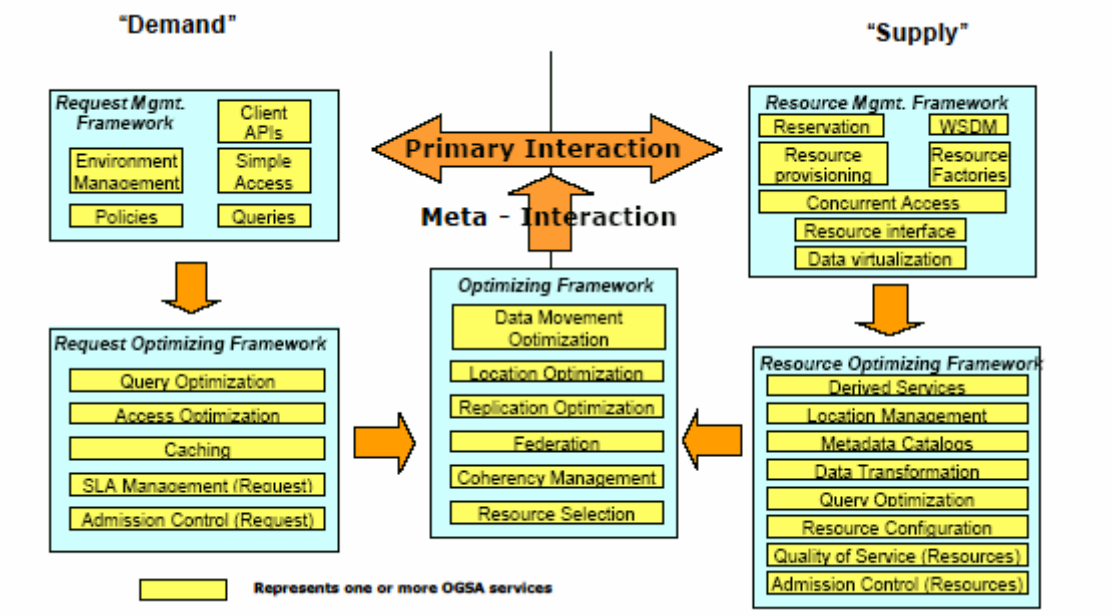


図６：データの概念的メタモデルにより供給側と需要側に分類

3.5.3 機能

本節では OGSA のデータサービスが提供する機能について解説する。それぞれ機能ごとに、それを実行するのに必要なサービス（の集合体）がある。実装の際は、これらの機能のうち、一部だけを提供してもよい。

3.5.3.1 透過性と仮想化

分散型システムには、多様なデータリソースが含まれることがある。これらのリソースは、データの構造化のためにそれぞれ別のモデルを用いることがある。また、そのデータを保存するのに別々の物理メディアを用い、データ管理のために別々のソフトウェアシステムを利用することがある。さらに、データを記述するために別々のスキーマを用い、データにアクセスするために別々のプロトコルやインタフェースを使うことがある。データはローカルでもリモートでも保存でき、唯一しか存在しない場合も複製されている場合もある。またデータをマテリアライズしたり、需要に応じて取り出すこともある。OGSA の

データサービスは、これらのデータリソース上で「仮想化」を定義できる。仮想化は、以上のような違いを隠し、その違いにとらわれることなくデータリソースを操作できるようにする抽象ビューである。

逆に、データサービスはクライアントにこれらの違いを無視させるものの、その違いを利用したいと考えるクライアントがいてもおかしくない。たとえば、あるデータベースに対して特定のクエリ言語を使ったり、利用するデータリソースの場所を指定したいと思うクライアントもいるだろう。あるクライアントは、データへのネイティブアクセスを求めるかもしれないし、別のクライアントはデータリソースのパフォーマンス・パラメータを調整したいと思うかもしれない。OGSA のデータサービスではこうしたクライアントをサポートするため、クライアントが仮想化インタフェースを迂回してリソース固有のインタフェースに直接アクセスできるようにしている。このような階層化インタフェースのおかげで、クライアントは自分たちに最適なパワーとアブストラクションを選択できるのである。

たとえば基本的なファイルの I/O は、POSIX や同様のシステムと同じように、リードやライトの作業を提供するインタフェースにより与えられる。リードやライトの作業を提供するサービスは、キャッシング、複製、最適化したデータ移行のように、高度な最適化を行うことがある。しかし仮想化したインタフェースは、これらの詳細をクライアントから見えないようにしている。より詳しい制御を求めるクライアントは、リソース固有のインタフェースを使って、キャッシュサービス、複製サービス、データ移行サービスなどを操作することができる。ただしその際、相応の透過性が失われる。

3.5.3.2 クライアント API

多くのユーザが OGSA のデータサービスを、NFS、CIFS、JDBC、ODBC、ADO、POSIX IO、XQuery などの既存の API を利用するレガシ・アプリケーションとともに活用したいと考えているだろう。クライアントに新しいインタフェースを使わせるには費用もかかり、実用的でない場合が多い。しかし OGSA のサービスを、これらの既存の API をエミュレートするラッパーでくるむことは簡単にできる。ただし、OGSA 自体が、そうしたラッパーを定義することはない。

通常これらのラッパーは、既存の API の操作を対応するメッセージにマップし、OGSA のデータサービスに送る。そのためラッパーからは、OGSA の全機能のうちの一部にだけしかアクセスできない。ラッパーから使用できる OGSA の機能の範囲は、関連する API の目的によって変わる。

3.5.3.3 拡張データタイプのサポートと操作

OGSA のデータサービスとともに使用されるデータリソースをすべて予測することは不可能である。どのような場合でも、新しいデータリソースがいずれ作成される。上述の階

層アーキテクチャでは、新しいタイプのリソースにアクセスする新しいデータサービスを追加することができる。

同様に、あるリソース上で必要となるすべての操作を予測することは不可能である。上述の階層インタフェースは、OGSA のマニュアルに示された操作以外に、新たな操作を追加するサービスもサポートしている。仮想化レイヤでは、クライアントはこうした拡張オプションを無視することもできるが、それを必要とするクライアントは、必要に応じて仮想化レイヤを迂回することができる。

3.5.3.4 データロケーション管理

OGSA のデータサービスでは、ある場所から別の場所に高い信頼性のもとでデータを移行することができる。オリジナルデータのコピーをとる場合やオリジナルを完全に移動させる場合がある。データをそれ以上不必要に移行することがないよう、与えられた場所でデータをキャッシュすることもできる。データキャッシュは、たとえばそのライフタイムやアップデートの一貫性に基づいて設定することができる。データはまた、複数のコピー間で複製することができるため、冗長性によりアベイラビリティを向上されることができる。

各ユーザに対するサービスにより、ユーザは自分のデータをアップロードし、上述のファシリティで管理することができる。他のユーザに許可するアクセスについては、セキュリティの設定で制御することができる。

3.5.3.5 単純アクセス

単純アクセスサービスでは、データリソースから（論理的）連続バイトを読み出したり書き込んだりする操作ができる。アクセスされるリソースは、ローカルであってもリモートであってもよい。仮想化インタフェースがデータの場所に関する詳細を隠してしまうからである。これによりデータサービスは、データの場所とデータアクセスの両方を最適化することができる。

3.5.3.6 クエリ（構造化アクセス）

OGSA のデータアクセスサービスには、構造化データリソースにクエリを利用するメカニズムがある。単純なケースで言えば、リレーショナル・データベース上で SQL クエリを走らせたり、XML データベース上で XML クエリを走らせることができる。あるいはテキストファイル上で正規表現を使うこともできる。他のサービスは、複数のドキュメント内でテキスト検索をしたり、フェデレートしたデータベース上で分散型クエリを実施することができる。

同期クエリは、リクエストに応答してデータを返すが、非同期クエリは派生データを新しいリソースとしてエクスポーズする。サービスはまた、クエリの結果を、他の特定の複

数のサービスに送ることもできる。

クエリサービスは、クエリをリソースに送る前にそれを最適化することができる。リソースはそのクエリをさらに最適化し、データへの同時アクセスのような問題に対処することもできる。

データ・フェデレーション・サービスは、受け取ったそれぞれのクエリを解析し、分散型データリソース上で走らせるサブクエリを作成する。このサービスはまた、ネットワークのトラフィックを最小限に抑えるために、中間処理の場所を指定できる。データ・フェデレーション・サービスはさらに、ワークフロー実行エンジンが効率的に作業をスケジューリングできるよう、これらのエンジンに対して関連情報を提供することができる。

3.5.3.7 移行

データサービスそのものがデータを移行することがある。たとえばデータサービスは、データを移動させたりアップデートする前に、１つのフォーマットから別のフォーマットへ変換したりフィルタをかけることができる。データサービスは、サービスの内部で実行されるストアド・プロシージャをサポートしており、サービスをある形のコンテナにすることができる。データの移行については、それを何らかの操作により直接的に起動することや、一定の条件に応じて自動的に起動するようプログラムすることもある。3.5.3.8 データのアップデート

OGSA のデータサービスでは、データリソースをアップデートするメカニズムが、データリソースのセマンティクスに応じていろいろある。カタログに関しては、作成、名前の付け替え、削除などの作業がアップデートに含まれる。構造化ファイルやデータベースについては、エントリのアップデートなどの作業がある。ストリームやそのほかのファイルに関しては、作業の大部分が新規データの追加に限定される。

データサービスは、アップデート作業のために何らかの形のトランザクション作用を指定することができる。データリソースに複製されたバージョンが存在していたり、データリソースが派生データサービスのソースである場合、アップデートはその複製されたデータリソースや派生データリソースにも適用される。この場合、さらには、いくつかのクライアントが同じデータリソースをアップデートしている場合は、データサービスはさまざまな形で一貫性の維持を行うことができる。これによりたとえば、クライアントが自分のアップデート結果を自ら作成したクエリを使ってつねに知ることができる。

3.5.3.9 セキュリティ・マッピングの拡張

データベース管理システムは、高度なセキュリティメカニズムを実施することが多い。システムによっては、個々のタプルのレベルで多彩なオペレーションやアクセスコントロールを提供しているものもある。そこで OGSA のデータサービスでは、ユーザ、オペレータ、アプリケーションがそうしたシステムが提供するより強力なコントロールにアクセス

できるよう、OGSA の標準セキュリティ・インフラストラクチャを拡張することをサポートしている。

3.5.3.10 データリソース・コンフィグレーション

データリソースでは、高度なコンフィグレーションに関するオプションが与えられることが多い。これらのオプションは、OGSA のデータサービスを通じてクライアントが使用するようにできる。さらにデータサービスは、サービスが提供するリソースの仮想化を設定するための操作を追加することもできる。たとえばリレーショナル・データサービスでは、データサービスのデータ仮想化の一部となるデータベースリソースからの特定のテーブルや、仮想化内部のテーブルとして利用されるデータベースに関するビューが、許可される。

3.5.3.11 メタデータ

メタデータ・サービスは、OGSA のデータサービスやその他のサービスに関するメタデータを保管するデータサービスである。OGSA では、OGSA サービスとそれを記述するメタデータとの間のリンクの維持がサポートされている。このサポートには、メタデータの一貫性を維持するためのプロビジョンが含まれる。

OGSA のデータサービスのためのメタデータには、データを記述するスキーマへのリファレンスなど、データ構造に関する情報が含まれている。サービスによっては、これは実用的ではない。データリソースが頻繁に変わるスキーマを数多く持っているからである。この場合、スキーマの情報はサービス自体が提供する。

3.5.3.12 起源

データサービスのメタデータには、データの起源や質に関する情報も含まれる。この情報は、リソース全体に関するもの、あるいは各構成部に関するもの、ときには個々の要素に関するものの場合がある。これにより、データを生成するサービスや他のプロセスもメタデータの一貫性を維持する必要性が出てくる。データの起源に関する完全な情報があれば、そのデータをもともと作成したワークフローをたどることで、データを再構築することができる。

3.5.4 プロパティ

プロパティは、機能的として定義されるものではなく、さまざまなサービスに適用されるアーキテクチャのアスペクトである。機能はサービスインタフェース内のエンティティによって定義されるが、非機能的プロパティは、これよりもグローバルでセマンティックなロールを持っている。

3.5.4.1 スケーラビリティ

OGSA のデータサービスは、データセットのサイズ、データセットの数、データフローのサイズ、サイトの数など、いくつかの大きさに関して大規模なスケールを取り扱っている。

3.5.4.2 サービスの品質

OGSA のデータサービスは、保証付きデリバリや参照整合性など、さまざまなレベルの QoS を提供している。

3.5.4.3 一貫性

データを複製、キャッシュ、派生させる際、さまざまな一貫性に関する操作が利用できる。同じく、メタデータ・カタログのデプロイメントや相反するアップデートの解決のため、さまざまなマスタリング・ストラテジやピアリング・ストラテジがサポートされている。

3.5.4.4 パフォーマンス

OGSA のデータサービスは、データのコピーや移動を最小限に抑えるように設計されている。これはグリッドの全体のパフォーマンスに重要な影響を持っている。

データ移行サービスは、帯域幅、利用状況、パケットサイズなどのモニタリング情報を活用している。これにより、合意されたサービスの品質に合うよう、データセットを移動する際に最良の方法を選ぶことができる。

3.5.4.5 アベイラビリティ

OGSA のデータサービスは、ネットワークの故障やその他の不具合が生じた場合、段階的な機能低下に関する機能がある。たとえばクエリサービスは、ソースの一部しか使用できない場合に、結果を部分的に返すよう設定することができる。

3.5.4.6 法的、倫理的制約

OGSA のデータサービスは、法的、倫理的ポリシーがその操作に影響を与えるような環境下で作動させなければならない場合がある。たとえば、個人データにアクセスし、そのユーザが実行できる操作に制限を加えることのできるようなエンティティを限定するポリシー（機密性ポリシー）もありうる。プライバシー上の問題から、個人に関するクエリを制限することもある。ただし場合によっては、あるグループに関する結果を、たとえば平均収入や収入合計額などのように、全体の結果として返すようなクエリをそのポリシーが許可することはある。

データのコピー作成に関する権利を制限する著作権によって、データが制約を受けるこ

とも多い。EU では、同じような（ただし異なる）「データベース権」という制限が、データベースだけに適用されている。

OGSA が提供するセキュリティメカニズムでは、これらの制限を指定することができる。セキュリティメカニズムをデータサービスとともに使用すると、（テーブルなどの）グループやリソース内の要素といったレベルで適用されるポリシーの詳細に関して、メカニズムが許可を与えなければならない。複雑なデータを含むシナリオは、これらのメカニズムのテストケースをとくに必要とする。

3.5.5 OGSA の他の部分との相関

本節では OGSA のデータサービスと他の部分とのやりとりについてまとめる。

3.5.5.1 トランザクション

データサービスは、トランザクションに関する古典的な例であり、多くのデータリソースがトランザクションのサポートを別々に提供している。トランザクションを分散型システムで実行する方法はいくつかある。通常の ACID トランザクションもその一例である。別の例としては、分散型データベースで同期化を行う２フェーズコミットがある。さらに、「タイムワープ」コーディネーションもサポートされている。これにより、サービスが投機的に

に実行されたり、トランザクションが異常終了した場合にその実行をロールバックすることが可能となる。

一般的に言って、データサービスだけでなく OGSA のサービスのすべてのやりとりに対してトランザクションがサポートされるべきである。このレベルのサポートは、本文書バージョン１では議論していなかった。なお、データサービスは、トランザクションの働きを自ら保証することができる。

3.5.5.2 ロギング

OGSA の他のサービス同様、データサービスは会計検査などにロギングサービスを使用する。逆に、ロギングサービス自体が、ログを保管するためにデータサービスを利用することがある。

3.5.5.3 実行管理サービス

データサービスは、実行管理サービス（OGSA-EMS）と密接なつながりがある。OGSA-EMS は、必要な場所にデータをステージングするために、データサービスを利用する。OGSA-EMS の実行計画サービスは、候補となる計算リソースからデータにアクセスするコストを考慮しなければならない。

3.5.5.4 ワークフロー

ワークフローはデータアクセスサービスに対し、クエリの結果をサードパーティに送り、データの移動に伴って移行サービスを適用するよう指示する。このタスクには、ワークフローサービスがデータアクセスサービスの機能に関して詳しい知識を持っている必要がある。また、分散型クエリサービスへのコールにより、他のマシン上でいろいろなデータの移動や作業が始まる場合がある。このためデータサービスは、ワークフローの実行がネットワークや他のリソースへの分散型クエリの影響を考慮するように、ワークフローマネージャに情報を提供することができる。

3.5.5.5 プロビジョニング

ストレージスペースやサービス自体のプロビジョニングのほかに、データサービスも、データセットをデータリソースにアップロードするためにプロビジョニングの助けを借りなければならない。あるデータサービスにフィルタやカッターをアップロードするためのサポートを必要とするデータサービスもある。

3.5.5.6 リソース予約

データサービスは、操作のために特定のリソースを予約する必要がある場合がある。たとえば、ファイルを移行するには、ストレージスペースとネットワーク帯域幅が必要となる。また、分散型クエリシステムでは、結合の操作を行うために計算パワーがこれに加えて必要となる。同じように、データサービスはリソースを予約するためのインタフェースを提供する必要がある。

3.5.5.7 検出

データサービスは、サービス自体を登録するためだけでなく、サービスが保管しているデータセットを登録するためにも、検出サービスを使用する必要がある。検出サービスはまた、スキーマの定義の場所も登録できる。

3.5.5.8 セキュリティ

セキュリティサービスでは、リソース固有のアイデンティティやロールに OGSA のアイデンティティやロールをマップすることができる。VO 管理サービスも同様に、これらのマッピングを制御する。またセキュリティサービスでは、データ移行の際の整合性と暗号化を調べることもサポートされている。

3.5.5.9 ネットワーク管理

ネットワーク管理は、大量のデータの移動を計画する際に重要となる。データサービスは、移行するデータ量や移行に際して与えられた時間的制約に合わせて OGSA のサービス

がネットワークパラメータを設定できるように、必要な情報（おそらくは最初のトリガ）を提供する。

3.5.5.10 ネーミング

OGSA のデータサービスは、データセットをネーミングするために OGSA ネーミング（3.9.4.1 節）を使う。たとえば複製されたファイルシステムでは、アドレスは特定のマシン上の１つのファイルの場所を指定するものである。抽象名は、ディレクトリサービスを使うなどして、場所とは無関係の方法でファイルを特定する。ヒューマン指向の名前は、ユーザフレンドリな短いファイル名であり、それが使われているコンテクストを参照することであいまいさをなくしている。

3.5.5.11 通知

通知サービスを使ってデータベーストリガを外部化することができる。データベース管理システムでは、一定の条件（トリガ）に合致した際に起動されるアクションを指定するメカニズムを与えている場合が多い。OGSA では、こうしたトリガに通知サービスを起動させるようにすることが簡単にできる。

さらに、通知サービスを実行した場合、データサービスを使って、通知メッセージを保管したりクライアントのクエリをサポートすることがある。

3.6 リソース管理サービス

3.6.1 目的

リソース管理では、グリッド内のリソースに対しいくつかの形で管理を行う。OGSA グリッドには、リソースに関連した３種類の管理[OGSA RM]が存在する。

- リソース自体の管理（ホストのリブート、ネットワークスイッチにおける VLAN の設定など）
- グリッド上でのリソースの管理（リソースの予約、監視、制御など）
- リソースで構成された OGSA インフラストラクチャの管理（レジストリサービスの監視など）

3.6.2 モデル

OGSA グリッドでは、異なる種類のインタフェースによりいろいろな形の管理が実現されている。これらのインタフェースは、表２の中央の列に示したように、３つのレベルに分類することができる。これは図７の右側にも示されている。

表２：管理種別とインタフェースの関係

管理種別	インタフェースのレベル	インタフェース

リソースそのものの管理	リソース・レベル	CIM/WBEM、SNMP など
	インフラストラクチャ・レベル	WSRF、WSDM など
グリッド上でのリソース管理	OGSA ファンクション・レベル	ファンクショナル・インタフェース
OGSA インフラストラクチャの管理		特定のマネージャビリティ・インタフェース

それぞれのレベルとインタフェースについては以下で詳しく説明する。その際、実装（たとえば実装を行うサービス）に目を向けるのではなく、マネージャビリティ・インタフェースに焦点を当てることにする。１つのサービスが複数のインタフェース（機能性の面では互いに関連性のないインタフェース）を実装したり、それが示す機能性から切り離されたりしていることがあることに注意しよう（たとえばあるリソースに対するマネージャビリティ・プロバイダは、そのリソースからは切り離されている）。したがってサービスに基づいた記述は不正確であり、代わりにインタフェースに基づいた記述を行うことにする。

図７に示したように、OGSA の機能は、その機能を実行するのに必要なリソースにおける機能にまで拡張することで、これらすべてのレベルをカバーしている。図７ではインタフェースを小さな丸で表している。

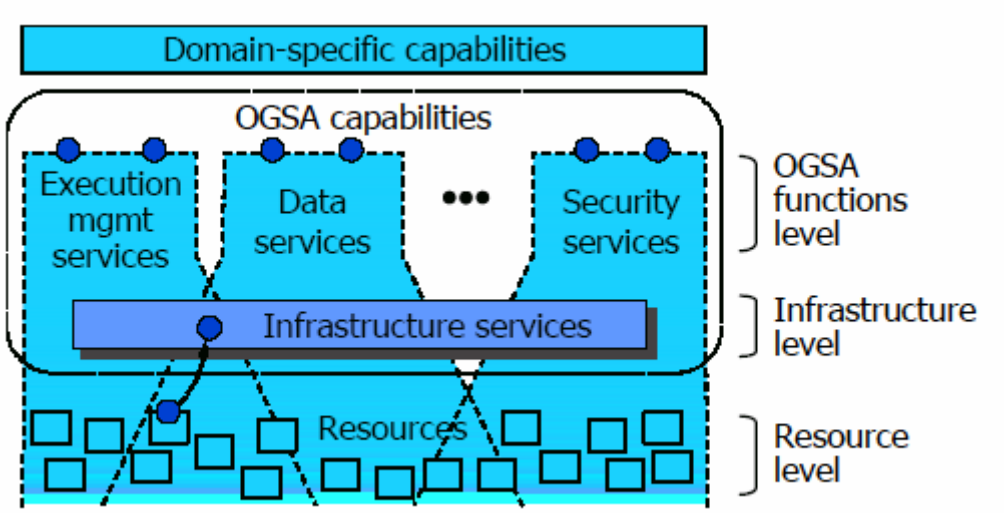


図７：OGSA における管理のレベル

リソース・レベルでは、リソースはその固有のマネージャビリティ・インタフェースを通じて直接管理される（離散的なリソースの場合、SNMP、CIM/WBEM、JMX、専用インタフェースなどのインタフェースがある）。このレベルでの管理には、監視（たとえばイベントなどのリソースのステートを取得する）、セットアップと制御（たとえばリソースのス

テート設定)、検出などがある。これらのリソースはリソースモデルにより与えられる記述に従って管理される。リソースモデルは、リソースのプロパティ、動作、イベント、それらの関係について定義するものである。

インフラストラクチャ・レベルでは、OGSA 環境におけるマネージャビリティと管理の両方に対して基礎を与え、リソースの基本的管理が行われている。複数のサプライヤが提供する非常に多くの多様なリソース（および、限られた数のリソースマネージャ）を統合するため、この基本的管理作業の標準化が必要となる。インフラストラクチャ・レベルでは以下のインタフェースが与えられる。

- 基本マネージャビリティモデル：このモデルは、WS リソース[WS-RF]としてリソースを表現し、検出やアクセスなどを行うための標準的な Web サービスの手段を通じて OGSA 内のリソースが操作されるようにするものである。このモデルにより、検出、停止、イントロスペクション、監視などを通じて、リソースは少なくとも最低限には管理可能となる。リソースに対するリソースモデルは、基本マネージャビリティモデルからアクセスする。

  １つの基本マネージャビリティモデルと、おそらくは１つのリソースモデルを使用すれば、リレーションシップ（依存関係やコンポーネントの定義)、作業状況、統計などの一貫性を、管理の全階層を通じて保つことができる。
- 一般マネージャビリティ・インタフェース：これは OGSA の機能を実行するすべてのサービスに共通するものである。このマネージャビリティ・インタフェースには、イントロスペクション、監視、サービスの生成・破棄といった機能がある。

OGSA ファンクション・レベルには２種類の管理インタフェースがある。図７では、各機能の上に２つの丸として描かれている。

- ファンクショナル・インタフェース：OGSA のいくつかの共通機能（OGSA EMS など）は、リソース管理の形態をとっている。これらの機能を提供するサービスは、ファンクショナル・インタフェースを通じて機能を公開する（ジョブの生成や破棄など)。
- マネージャビリティ・インタフェース：各機能には特定のマネージャビリティ・インタフェースがあり、それを通じて機能が管理される（たとえばレジストリやジョブマネージャの監視)。マネージャビリティ・インタフェースは、一般マネージャビリティ・インタフェースを拡張し、その機能の管理に特有のマネージャビリティ・インタフェースであればどのようなものでも追加できる。

3.6.3 管理機能

インフラストラクチャ・レベルでの管理機能は、OASIS WSDM が提供する。OASIS WSDM は、IT 業界におけるマネージャビリティの重要な標準となることが期待されているものである。WSDM は、Web サービスの管理（MOWS）[MOWS]と Web サービスを使っ

た管理（MUWS）[MUWS part1][MUWS part2]に関して、別々のドキュメントを作成しているところである。MUWS では、基本マネージャビリティモデル、すなわちリソースの表し方や、ステートの表現や操作、リソース間の関係など、OGSA に共通の基本的マネージャビリティ機能の表し方が与えられている。MOWS では、OGSA グリッド内のサービスを管理するための一般マネージャビリティ・インタフェースが与えられている。リソースモデルについては、OGSA で使用されるリソースモデルとして CIM が現在検討中にある。CIM は、その幅や拡張性から、セキュリティ、実行管理、自己管理等、OGSA の複数の機能を管理する作業に使用することができる。

OGSA ファンクション・レベルでは、リソース管理機能として、通常の分散型リソース管理作業や IT システム管理作業などがある（これらに限られているわけではない）。こうした作業はポリシーに基づくことがある。すなわち、認証スキーム、移行プロトコル選択、QoS メトリクス、プライバシポリシーなどの要件を満たすために導入されたポリシーアサーションを施行することができる。OGSA のリソース管理機能には、リソースの予約や監視、制御、そして VO 管理やセキュリティ管理、問題の特定やフォルト管理、ポリシー管理（すなわちポリシーそのものの管理）、サービスの集合体や検出サービス、メータリング、デプロイメント、検出などの機能性が含まれている。

3.6.4 プロパティ

3.6.4.1 スケーラビリティ

管理アーキテクチャは、場合によっては数千ものリソースに対応する必要がある。このスケーラビリティを達成するには、階層的な方法かつ（あるいは）ピアツウーピアな（フェデレートされた／コラボラティブな）方法で管理を行わなければならない。OGSA はそうした管理を許可する必要がある。

3.6.4.2 相互運用性

管理アーキテクチャは、ソフトウェア、ハードウェア、サービスの境界線を、たとえば異なる製品間の境界を超えて拡大することが可能でなければならない。そのため標準化された幅広い相互運用性が「ストーブパイプ」を避ける上で重要となる。

3.6.4.3 セキュリティ

セキュリティに関しては、管理に２つのアスペクトがある。「セキュリティの管理」は、認証管理、認可、アクセスコントロール、VO、アクセスポリシーなどのセキュリティ・インフラストラクチャの管理を対象としたものである。一方、「安全な管理」は、管理作業においてセキュリティ・メカニズムを使用することを意味する。管理は、それ自身のインテグリティを保証し、リソースの所有者と VO のアクセスコントロール・ポリシーに準拠できなければならない。

3.6.4.4 信頼性

管理アーキテクチャは、不具合を１つでも生じさせてはならない。信頼性を高めるために、１つのリソースが複数のサービスによって仮想化されることもある。その際、それぞれのサービスは管理のエンドポイントとして URL を１つ公開する。このような場合、マネージャビリティ機能を提供するシステムは、メトリクスなど特定のクエリのために、１つのリソースを仮想化する複数のサービスの結果をマネージャビリティ・プロバイダがア統合しなければならないことを知っている必要がある。

3.6.5 OGSA の他のサービスとのやりとり

インフラストラクチャサービスは、インフラストラクチャ・レベルで WSRF や WSDM といったインタフェースを実装する。これらのインタフェースがリソースの基本管理作業を定義するため、OGSA のすべてのサービスはリソースを使う際にこれらのインタフェースを使うことになる。

情報サービスは、とくに監視のために、多くの OGSA サービスが使用することになるリソース管理機能を提供している。たとえば、リソースの検出に使用するレジストリや問題の特定に使用するロギングがその例である。

実行管理サービスとデータサービスは、リソース管理機能を実行作業やストレージ関連作業の検出、プロビジョニング、監視などに使用するコンシューマである。実行管理サービスとデータサービスはまた、OGSA ファンクション・レベルでは、リソース管理機能のプロバイダでもある。

自己管理サービスは、グリッド上のリソース（アプリケーションやデバイス、あるいはその場所）を制御するため、リソース管理サービスのファンクショナル・インタフェース上で利用される。自己管理サービスはまた、（レジストリの管理や、レジストリがオーバーロードした場合にインスタンスをさらにデプロイするなど）グリッドのインフラストラクチャ自体を制御するため、リソース管理サービスの特定のマネージャビリティ・インタフェース上で使用される。同様のことは、セキュリティサービスにも当てはまる。セキュリティサービスは、たとえばファンクショナル・インタフェースや特定のマネージャビリティ・インタフェースの両方に対して、アクセスを制御するものである。

3.7 セキュリティサービス

本節ではセキュリティサービスの目的、モデル、機能、プロパティについて解説し、OGSA の他の部分とのやりとりについて議論する。

3.7.1 目的

OGSA のセキュリティサービスは、（仮想）組織の内部でセキュリティ関連のポリシーを

施行するものである。[Grid Anatomy][Grid Physiology][VO Security]

一般的には、セキュリティポリシーの施行の目的は、より高いレベルのビジネス目標を達成できるようにすることにある。ポリシーを施行すればするほど関連コストが上昇し、ポリシーの志向を緩めれば緩めるほど損失が膨らむため、微妙なバランスが生じるケースが多い。一方、ポリシーの要請が厳しいほど複雑さや柔軟性の欠如が増し、それにより潜在的な利益や報酬に悪い影響が及びかねない。法的な規則や規制によって、関連するセキュリティポリシーの遵守が命じられる場合もある。

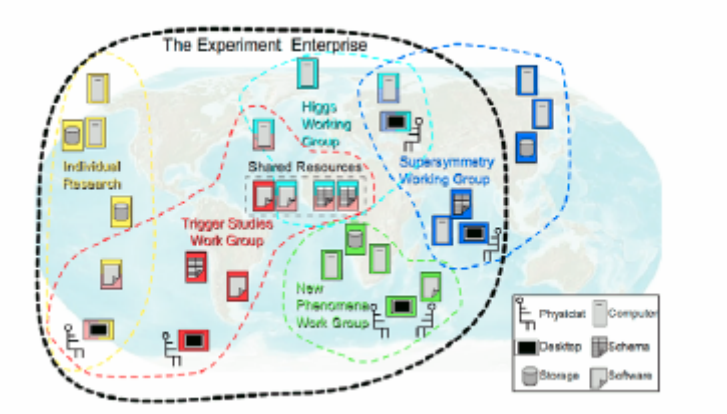


図８：組織横断的な協力関係

グリッドに固有のアプリケーションが複数の管理ドメインにわたって広がる（図８参照）ことにより、各ドメインが各々のビジネス目標を達成できる。それはさらに、それぞれのドメインが個別に独自のポリシーを構築、施行しているということを意味する。これらのポリシーは、複雑さや厳格さにおいて大きく差がある。グリッドアプリケーションにおける一連の作業に関連したあらゆるやりとりは、ドメインがローカルに施行するポリシーと VO に対して設定されたポリシー、すなわち組織横断的な（ビジネス上の）合意事項、の両方を遵守したものでなければならない。

ビジネスとそれに関連するセキュリティの目標を達成するため、たとえばメッセージのインテグリティと機密性に関する施行や、互いにやりとりしているエンティティの認証、最小認証長、安全なロギングと監査、責任の分別、侵入や外部進出の検知、認可ポリシーの確認、最小特権作業、必須のアクセスコントロール・メカニズム、任意のアクセスコントロール・メカニズム、環境のトラストレベルやアシュアランスレベル、アプリケーションのアイソレーション、DoS 攻撃の回避、冗長性、トレーニングなどを、セキュリティポリシーで規定することができる。

OGSA のセキュリティアーキテクチャ上のコンポーネントは、さまざまなシステムを安全に相互運用できるよう、セキュリティに関して多く利用されているモデルやメカニズム、プロトコル、プラットフォーム、テクノロジーなどをサポート、統合、一元管理する必要がある。こうしたコンポーネントは、既存のセキュリティアーキテクチャやセキュリティ

モデルとの統合を、プラットフォームやホスティング環境を横断して行うことができなければならない。すなわちセキュリティアーキテクチャは実装にとらわれないものでなければならず、その結果いかなる既存のセキュリティ・メカニズム（たとえば Kerberos [Kerberos V5]や PKI [PKI]）であってもインスタンスを作成できる。またアーキテクチャは拡張可能なものでなければならず、その結果新しいセキュリティサービスが現れた場合にそれを取り入れることが可能になる[RFC2903][WS Security Whitepaper][Liberty]。さらにアーキテクチャは、既存のセキュリティサービスと統合可能なものでなければならない。複数のドメインやホスティング環境を横断するサービスは、互いにやりとりができる必要がある。このため、プロトコル、ポリシー、アイデンティティといった複数のレベルでの相互運用の必要性が出てくる。さらに、状況によってはアプリケーションの実行前にサイト間の信頼関係を構築することができないこともある。参加しているドメインが異なるセキュリティ・インフラストラクチャ（Kerberos や PKI など）を持っている場合、それらのセキュリティ・メカニズムの間で何らかのフェデレーションを通じ、必要とされる信頼関係を築くことが必要である。

3.7.2 モデル

本節では OGSA のセキュリティサービスの根拠や仕様を与えるためにモデルの説明を行う。ここに示すモデルは、数学的には正式なモデルではないが、施行されるセキュリティポリシーを記述する言語を与え、そのポリシーに関する共通理解を深めるためのものである。

一般には、あるコンテクストの範囲内でエンティティがメカニズムを通じて相互作用している様子を観察することができる。エンティティとは、ユーザ、サブジェクト、サービスなどを指している。相互作用をするメカニズムは、メール、電話、HTTP、SOAP、SSL/TLS など、さまざまな通信手段をすべて含んでいる。コンテクストは相互作用を明らかにするものであり、相互作用を 1 つのマシンだけにローカライズさせることができる。あるいは VO 内でのサービスの呼び出し、および（または）安全な関係の確立、および（または）分散型トランザクションに関係している。

これらのエンティティ、メカニズム、コンテクストは、すべて属性やプロパティの集まりで記述される。属性のタイプや値は、エンティティの特定に用いられるものもあり、あるいは分類やグルーピングに使用されるものもある。さらにこれらの属性のうちのいくつかは「必然的」なものである。必然的な属性とは、外部権限に問い合わせることなく独力でエンティティを特定できるものである。必然的属性の例としては、共通の秘密、プライベートとパブリックの 2 つのキー、指紋などがある。その他の属性値はすべて、属性の発行者や認証機関によって、エンティティの必然的属性と本質的に結びつけられている。

セキュリティポリシーは、これらのさまざまなエンティティ、相互作用メカニズム、コンテクストに関するステートメントであり、関連する属性値、プロパティ、およびそれら

の間の関係について制限を規定するものである。ポリシーステートメント（または規則）は、エンティティ（アイデンティティやユーザ属性など）、リソース（エンドポイントなど）、環境特性（時間、場所、目的、リクエストする側やリクエストパスの信頼性レベルなど）を使って表現される。これらのポリシーは、認証、認可、信頼性、アイデンティティ・マッピング、デリゲーション、保証レベルといったいろいろな側面を扱うものである。

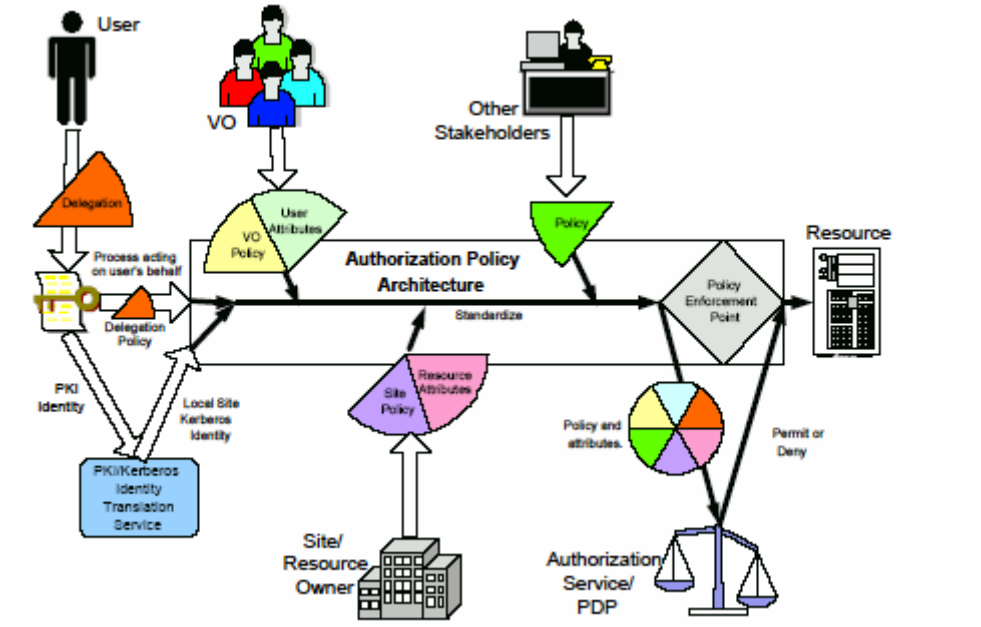


図 9：ポリシーの決定と施行に対する入力例

ここに取り上げるモデルは、セキュリティポリシーの管理、表現、発行、検出、通信、検証、施行、融和などを行う相互作用パターンを持ったエンティティとして、セキュリティサービスを定義するものである。言い換えれば、セキュリティポリシーの施行は最終的な目標であり、セキュリティサービスはその目標を達成するために設計され、開発されたものである。

モデルを具体化するため、これらのエンティティや相互作用メカニズム、コンテクストを特定し、その属性や共通の関係について一部を議論する。図 9 はリソースの仕様を保護するためのポリシーの施行例を描いている。モデルを理解するには、図 9 の例を、相対する 2 つの点からたどっていくとよい。1 つは最初のユーザ認可であり、もう 1 つは施行された認可である。さらに、認可ポリシーが施行場所で許可を出した場合にユーザがリソースにアクセスする許可だけを得るようにするとモデルが理解しやすくなる。リソースの認可ポリシーは、たとえばユーザ名、ユーザの役割、VO におけるユーザのメンバーシップなど、関連エンティティの属性値に対する規則として表現される。ユーザの最初の認証により、必然的属性値が示され、検証される。たとえば与えられたパブリックキーに合致するプライベートキーを持っているかどうかが調べられる。通常、ポリシーが派生属性で表されて

いるということと、最初の認証がキーだけしか与えないということとは相容れないようだが、ポリシーに表現された属性に対してキーを結びつけるマッチング属性のアサーションを見つけることで、この問題は解決される。図 9 にはいくつかのセキュリティサービスが描かれている。これらは、ユーザのパブリックキーの信用証明を Kerberos の信用証明にフェデレートし、施行されたポリシーのステートメントで使用されている特定の属性のバインディングをアサートしているいろいろなソース（VO やローカルサイトなど）から属性を取得するために用いられる。

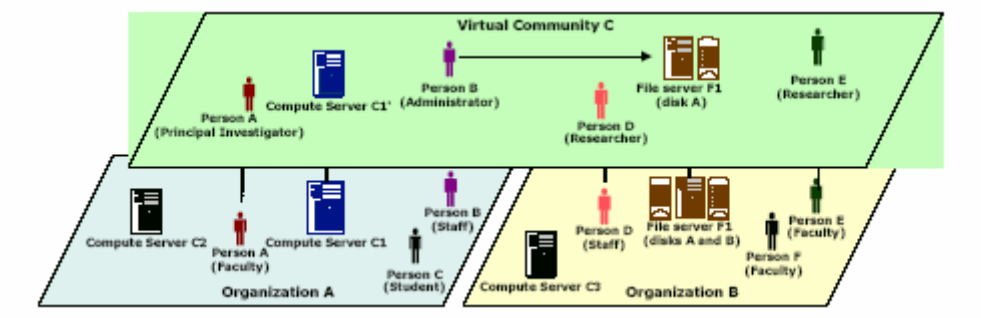


図 10：仮想コミュニティの概念図

ここに示したモデルは、一般の Web サービスのセキュリティモデルやその他の分散型計算アーキテクチャと概念的に変わりはない[RFC2903][WS Security Whitepaper][Liberty]。しかしグリッドアプリケーションは、組織横断的な設定に関連するエンティティやパターンと、それらの相互作用を可能にするセキュリティサービスにとりわけ着目している[VO Security][Role-Based VO][EU DataGrid][Fine-Grain Auth][Fine-Grain Auth RM][Dynamic Access Control][PRIMA][Grid Auth Framework][Grid AAA Req][CAS][SAZ][GS Security]。こうした相互作用は、ビジネスアプリケーションではよく見られるものである。ビジネスアプリケーションでは、ビジネス間の相互作用（あるいは企業内の組織間相互作用）が同じような要件や設定を生み出している。たとえば図 10 には 2 つの組織とそれにまたがる仮想コミュニティが描かれている。この仮想コミュニティはそれ自身のポリシーによりコントロールされている。仮想コミュニティのコンテクスト内のエンティティは、それ自体が固有の属性やプロパティを持っており、これらはホームドメインにおける属性やプロパティとは関連性を持たない別のものである点に注意しよう。これにより、このセキュリティサービスモデルは、複数のポリシーを並行して施行することに対応する必要がある。なおこのポリシーは、それ自身の適切なコンテクストの範囲内で評価されなければならない。

3.7.3 シナリオ例

3.7.3.1 デジタルライブラリ

教材用デジタルライブラリ・プログラムが、ある公的機関により運用されている。国内

の多くの学校や公的図書館がこのライブラリ・プログラムに参加している。プログラムでは、参加校の教員や生徒、あるいは図書館に対し、それまで学校や図書館でそれぞれ保管してきたデジタルブック、デジタルビデオ、デジタル写真、その他あらゆるデジタル媒体などの教材を共有する手段が与えられている。参加校や参加図書館は、たとえばユーザやリソースの登録や削除など、ユーザやリソース（教材）の管理にそれぞれ責任がある。学校によっては、たとえば大学などは、いくつかのサブ組織から構成されている場合がある。その場合、大学内の各サブ組織はそれ自体で VO を形成しており、その大学全体の VO はこれらサブ組織の VO の集合体となる。このシナリオを図 11 に示した。

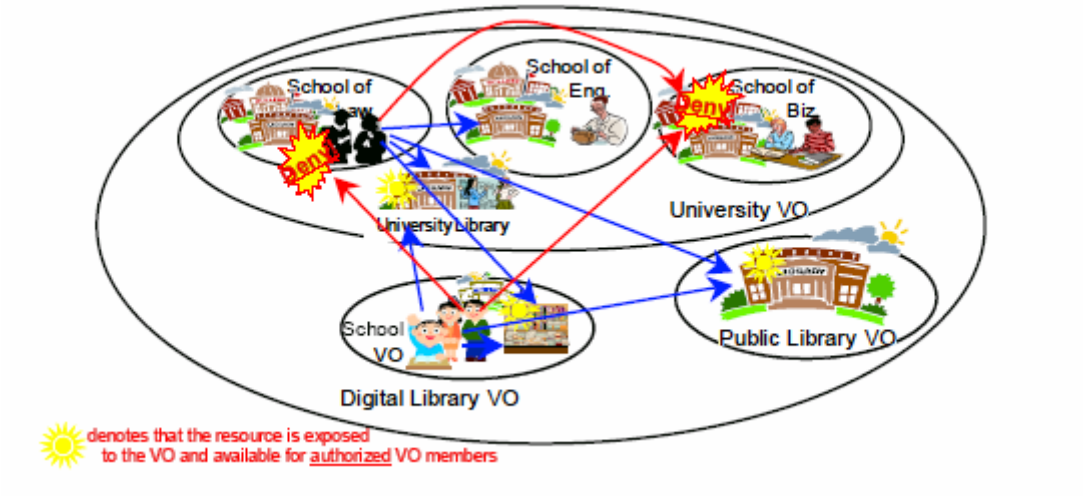


図 11：VO の例：デジタルライブラリ

ライブラリ・プログラムのユーザが共有教材にアクセスする方法に関して、ポリシーの例を以下に示す。

・ 参加校に通う全学生が、学生向け教材にリード・アクセスできる。
・ 全教員が学生向け教材と教員向け教材にリード・アクセスできる。
・ 参加校により承認された教員は、新しい教材を登録することができる。

大学の図書館は、大学内からのアクセスは許可するものの、デジタルライブラリ VO からのアクセスは許可しない。

3.7.3.2 最小特権のデリゲーション

権利のデリゲーションは、他のエンティティを代表してサービスを実施する際に必要となる基本機能である。この権利デリゲーションに関連して、サービスによってはコンプロマイズされて権利を不適切な方法で利用するものが出てくるという危険性がある。この危険性を抑えるには、サービスが本当に必要とする権利のみをデリゲートするように制限する必要がある。この「最小特権デリゲーションモデル」では、起動したサービス・オペレーションと権利の正確な「量」とを合致させることができなければならない。権利の正確な量という要件は、自明なものではない。多くのグリッドアプリケーションは、ジョブの

概念を使用しており、そこではアプリケーション独自の言語でジョブのディレクティブが定められている。このジョブの要件は、検出、ブローカ、スケジューラ・サービスによってリソースの機能やアベイラビリティとの適合が行われる。こうしたジョブ・ディレクティブやリソースの機能の表現に使われている言語は、必要な同等の権利を表現する際に使用されているディレクティブと調和できるものでなければならない。適合しない場合はデプロイメントに終わる可能性が高く、その場合、ジョブのディレクティブを確実に実行するためにサービスに過剰に多くの権利が与えられる必要が出てくる。図 12 にこの問題を図示した。

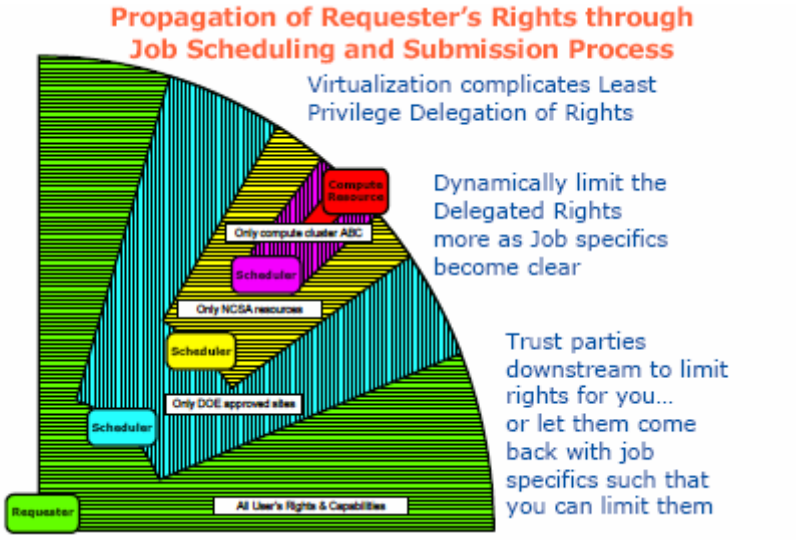


図 12：最小特権デリゲーションとジョブ記述

3.7.3.3 分散型環境における安全なロギング

ロギングサービスや、リコンシリエーションを目的としたログへの安全なアクセスは、サービスやそれに関連するログが別々の管理ドメインに置かれているような分散型設定において、ますます難しい問題となっている。図 13 にその状況を示した。

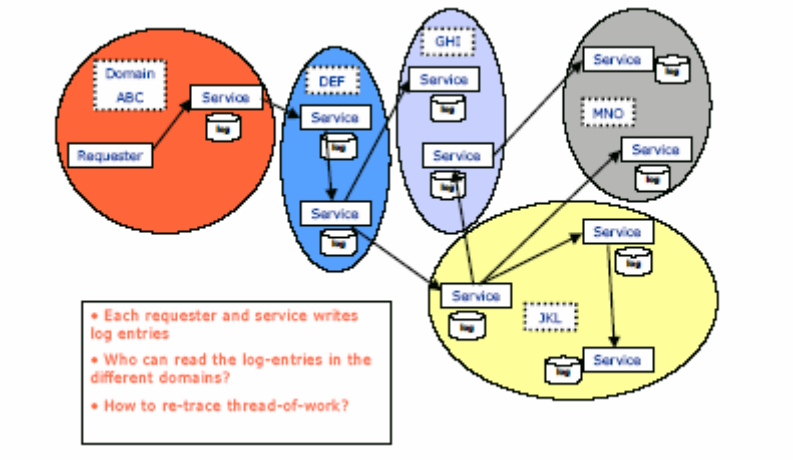


図 13：分散型環境における安全なロギング

ログが保護されたものであり、勝手にさわることができず、メッセージの統合ができるものであるためには、ロギングサービスがセキュリティの特徴を有している必要がある。そうしたレベルにまで高度化できれば、イベントを安全に記録する監査の概念に到達する。ここで言うイベントとは、セキュリティイベント、ビジネスイベント、トランザクションイベントなど、あらゆるイベントを意味する。セキュリティのサービスやインフラストラクチャは、イベント・インフラストラクチャが消費することのできるセキュリティイベントの生成を、セキュリティイベントへの監査や反応が可能になるように取り扱う必要がある（たとえば侵入阻止システムは一連の DoS イベントに対して反応する）。

3.7.4 機能

本節では、セキュリティ機能を解説し、対応するセキュリティサービスや使用パターンとどのような関係にあるかを説明する。サービスや使用パターンのより詳しい説明は、本文書の今後の版で与えることになっている。

例として、リクエスタとサービスプロバイダの両方が、ポリシーの遵守のために別々のインフラストラクチャ・セキュリティサービスへ呼び出しする様子を図 14 に示した。図では、コールアウトがスタブ内から行われ、アプリケーションの外部で透過的に行われていることに注意しよう。ポリシーの施行の多くをこのように行うことができる。これには、セキュリティ関連コードを最小限にとどめることができるというアプリケーション・ディベロッパにとっての利点がある。

さらに図 14 で明らかなように、リクエスタ、サービスプロバイダ、VO のそれぞれのポリシーにサービスの呼び出しが遵守するよう、別々の組織で管理されているいろいろなセキュリティサービス・インスタンスに対して呼び出しが行われている。

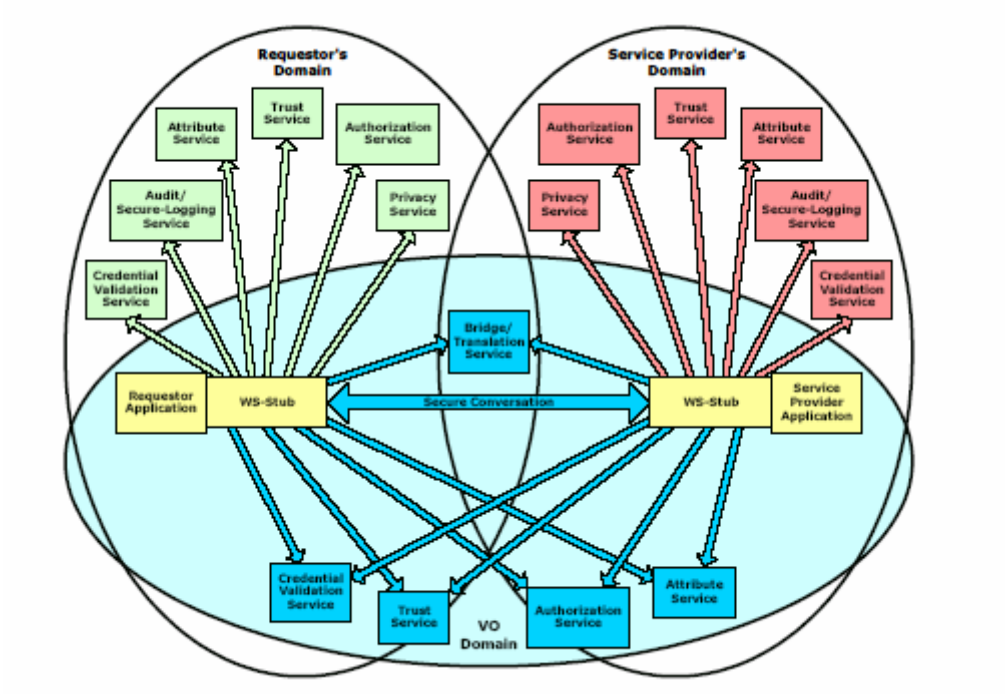


図 14：仮想組織の設定におけるセキュリティサービス

機能とそれに対応するセキュリティサービスを以下に列記する。

- 認証：認証は、アサートされたアイデンティティの証拠検証を扱うものである。この機能は図 14 のクレデンシャル検証サービスとトラストサービスの一部である。サービスリクエスタがアサートされたユーザ ID に適切なパスワードを与え、両者の組み合わせを評価することも認証の例の１つである。別の例としては、Kerberos メカニズムを通じたサービスリクエスタの認証や、サービスプロバイダのホスティング環境へのチケットの送付などがある。後者は、そのサービスに対するインスタンスが作成される前にチケットの信頼性を見極めるものである。
- アイデンティティ・マッピング：図 14 のトラストサービス、属性サービス、ブリッジ／トランスレーションサービスは、あるアイデンティティ・ドメインにあるアイデンティティを、別のアイデンティティ・ドメイン内のアイデンティティに変換する機能を持っている。例として X.509V3 デジタル証明書の内部で扱われる X.500 識別名（DN）の形のアイデンティティを考えよう。一組の所有者 DN、発行者 DN、証明書シリアル番号は、所有者のアイデンティティやサービスリクエスタのアイデンティティを持っていると考えられる。この例におけるアイデンティティ・ドメインの範囲は、認証機関が発行する一連の証明書であろう。その証明書がサービスリクエスタのアイデンティティを運ぶ際に使用されるとすれば、ポリシーを通じたアイデンティティ・マッピング・サービスがそのサービスリクエスタのアイデンティティを、（たとえば）ホスティング環境のローカルプラットフォームレジストリに対して意味のあるアイデンティティにマッ

プすることができる。アイデンティティ・マッピング・サービスは、サービスリクエスタの認証とは関係がなく、むしろ厳密に言えば、ポリシードリブンのネーム・マッピング・サービスである。

- 認可：認可サービスは、ポリシーに基づいたアクセスコントロールの決定を明らかにすることに関わっている。認可サービスは、認証されたサービスリクエスタのアイデンティティを具現化するクレデンシャルを入力として使用し、サービスリクエスタが要請するリソースに対して、サービスリクエスタがそのリソースにアクセスすることを認可されているかどうかポリシーに基づいて明らかにするものである。OGSA に準拠したサービスのホスティング環境は、アクセスコントロール機能を提供するものと想定されており、施行されているアクセスコントロール・ポリシーの精度に応じて認可の抽象サービスをさらに公開することが適当である。
- クレデンシャルな対話：図 14 のトラストサービス、属性サービス、ブリッジ／トランスレーションサービスは、ある種のクレデンシャルから別の種類（または別の形）のクレデンシャルへ変換するというクレデンシャル変換サービスを行う。これには、エンティティ（サービスリクエスタやサービスプロバイダ）に関係したグループメンバーシップ、特権、属性、アサーションを調整する作業なども含まれている。たとえばクレデンシャル変換サービスは、Kerberos クレデンシャルを認可サービスが要求する形に変えることができる。このポリシードリブンのクレデンシャル変換サービスは、サービスが使用するかもしれないさまざまな種類のクレデンシャルに関し、その相互運用を容易に行うことができる。クレデンシャル変換サービスは、アイデンティティ・マッピング・サービスを使用すると思われる。
- 監査と安全なロギング：監査サービスは、アイデンティティ・マッピング・サービスや認可サービスと同様に、ポリシードリブンである。監査サービスはセキュリティ関連のイベントを追跡記録する役割を持っている。結果として得られた監査記録を分類し、解析を行うことで、必要なセキュリティポリシーが施行されているかどうか調べる。監査およびそれに続く分類作業は VO 内のセキュリティ管理者が行い、決められたアクセスコントロールや認証に関するポリシーを VO が遵守しているかどうか判断する。
- プライバシ：プライバシサービスは、個人を特定できる情報（PII）をポリシードリブンで分類することに主に関与している。サービスプロバイダやサービスリクエスタは、このプライバシサービスを用いて PII を保管することができる。こうしたサービスを使って、VO のプライバシポリシーを明確にし、施行することができる。

関連するセキュリティコンポーネント間の関係について、関連サービスの層構造として図 15 に別の観点から示した。

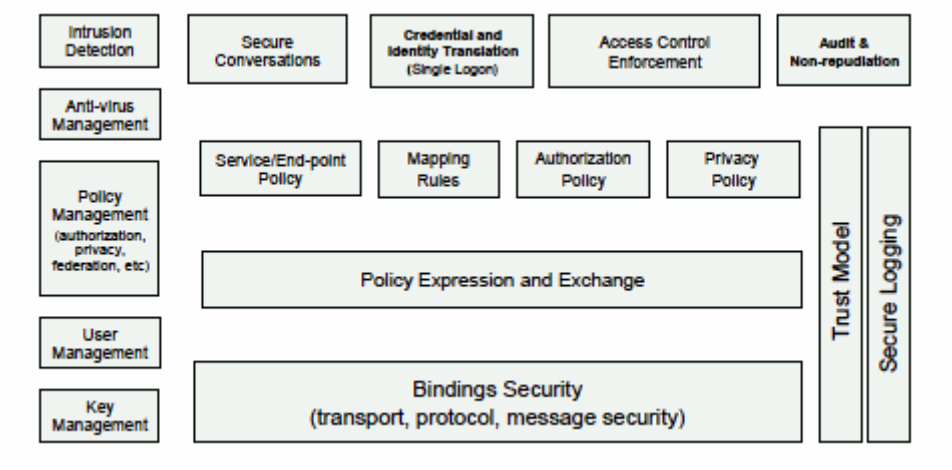


図 15：グリッドセキュリティモデルのコンポーネント

サービスリクエスタやサービスプロバイダが使用するセキュリティインタフェースは、すべて OGSA の範囲内で標準化する必要がある。標準に準拠した実装は、既存のサービスや決められたポリシーを、コンフィグレーションを通じて利用することができる。特定のセキュリティ関連インタフェースを標準に準拠して実装することで、それに関連するおそらくは別のセキュリティサービスが提供できるようになる。

3.7.5 プロパティ

一般的に言ってセキュリティサービスのプロパティは、施行すべきポリシーに沿った技術要件によって変わりうる。たとえば、決められたポリシーの施行を可能にするには、最大待ち時間、応答時間、アベイラビリティ、リカバリなどのプロパティに関係するセキュリティサービスが一定のサービスレベルに達している必要がある。

セキュリティサービスを実装すると、多くの場合、必要なプロパティを他のサービスを使うことで取得することができる。たとえば属性情報サービスは、データサービスを使って LDAP や RDBMS のアサーション情報にアクセスすることができる。そしてデータサービスのデータミラーリング機能を使って、決められたセキュリティポリシーの施行に準拠する上で必要となるアベイラビリティプロパティを取得することができる。

3.7.6 他の OGSA サービスとのやりとり

一般に OGSA サービスの呼び出しは、いずれも関連するセキュリティポリシーの施行を前提としている。場合によっては、このセキュリティポリシーの施行は、インプリシットでハードコードされている。あるいは別の場合では、外部のセキュリティ・インフラストラクチャサービスをプラグインあるいはコールアウトする機能が、デプロイメントに不可欠である。この点では、OGSA のあらゆるサービスは、セキュリティサービスに依存し、その上に重なり合って構成されているといえる。

セキュリティサービスと要件は、より高いレベルにある他の OGSA サービスに密接に結びついている場合がある。たとえば、属性サービスの実装では、OGSA データサービスを使って、レジストリやデータベースからポリシー関連の情報を引き出すことができる。この点で、セキュリティサービスは他の OGSA サービスのコンシューマであるといえる。

3.8 自己管理サービス

3.8.1 目的

IT インフラストラクチャを所有したり操作したりすることの難しさやコストについて、それを低減する方法の１つが自己管理であると考えられている。自己管理環境では、コンピュータ、ネットワーク、ストレージデバイスなどのハードウェアコンポーネントや、オペレーティングシステム、ビジネスアプリケーションなどのソフトウェアコンポーネントなどといったシステムコンポーネントが、自ら設定、修正、最適化を行う。

こうした自己管理の属性は 3.8.4 節に詳しく解説されているが、それによると、IT システムを設定、修正、最適化する作業は、そのシステムのコンポーネントがビジネス・ニーズに応じて見定めた状況に基づいて行われ、いずれも同じ技術によって実行される。これらの直感的で協調的な特徴が組み合わさることで、企業は少ない人的資源で効率的に作業を進めることができ、さらにコストの削減や、変化に対応する組織の能力を高めることができる。たとえば自己管理システムでは、新しいリソースがデプロイされるとそのまま最適化が行われる。これは従来の実施方法とは大きく異なるものである。従来は、デプロイメントの前に、そのリソースが効率よく動くよう、膨大な解析が必要とされていたのである。

自己管理属性は OGSA の中でも普及が期待されているものの、自己管理システムに含まれるいずれのサービスもこれらの属性をすべて、あるいはその一部を実施することができるということはない。むしろこれらの属性は、システムの全体としての自律的性質の一部である。

自己管理の主な目的の１つは、システム管理の難しさやコストを低減するため、できるだけ自動でいくつかのサービス（あるいは分類によってはリソース）に対してサービスレベルの到達を図ることにある。作業環境では、ソリューション・コンポーネントの動作のさまざまなアスペクトを、コンポーネントのディベロッパにはあらかじめ指定することができないような方法で制御することが必要になる場合が多い。自己管理システムにおいては、そのような制御は、システムコンポーネントの動作を左右するビジネス目的のポリシーをデプロイすることで行われる。その役割を担うのがサービスレベル・マネージャである。サービスレベル・マネージャ（SLM）はポリシーを設定、調整する役割を持ち、ビジネスの目的に全体的に合致するよう、管理されているリソースやサービスの動作をシステムの状態に応じて変更する。SLM 自体は、それを実装する際に埋め込まれたポリシーや、他の SLM から取得したポリシーにより管理されている。したがって、サービスが自己管理

のアスペクトを持つことができるだけでなく、自己管理作業に関わることもできる。

異なる SLM による組み合わせや階層構造があり、それによりシステム動作やシステムの初期設計の複雑さを相当程度に抑えることになる。過程において、単純な SLM を取り込むことで複雑な SLM を構成することができるからである。

自己管理機能は、OGSA の重要な部分であるものの、まだ準備段階にあり、自己管理のアスペクトの一部のみここで解説する。さらに詳しい解析については、本文書の今後の版で与えることにする。

3.8.2 基本属性

自己管理のさまざまな段階において集団として必要になる属性の集まりを以下に示す。概念レベルでのそれらの存在を議論するとともに、その実装については何の仮定も行わない。

3.8.2.1 サービスレベル・アグリーメント

サービスレベル・アグリーメント（SLA）には、サービスプロバイダとサービスのユーザとの間のビジネスや IT のアグリーメントなどがある。SLA は、その目的に関して測定可能な量で表現した指針を示し、管理対象となるサービスやリソースの目標を与えるものである。

3.8.2.2 ポリシー

SLM の動作やその制御下にあるマネージャブルなリソースは、ポリシーを使ってコントロールされる。自己管理するエンティティの動作を支配するポリシーは、サービスレベル・アグリーメント（SLA）から取得したものであり、マネージャブルなリソースの使用を左右する管理コンテクストの一部としてデプロイされる。SLM は、その SLM の動作をつかさどるポリシーに基づいて、動的管理インフラストラクチャにおけるリアルタイムの変化をとりまとめる。

3.8.2.3 サービスレベル・マネージャ・モデル

このモデルは、さまざまな SLM や人間のオペレータが、設計段階では互いについて知識を持たなくても、互いにやりとりをして理解しあうことができるように、SLM のインスタンスを作成し、SLM とともに作業を行う枠組みを与えるものである。SLM モデルには、インタフェース・コンポーネントが存在する。さらに、SLM やそのコンポーネントの管理サービス（管理されたサービスではない）の表現、それらのサービスが形成するコントロールループや SLM のインスタンシエーションやメインテナンスなどもある。

SLM は通常、いろいろなサービスアクティビティを制御、調整するために使われるジェネリック・コントロールループ・パターンに基づいてモデル化される。このパターンは循

環し、監視、解析、プロジェクション、アクションの段階から構成されている。SLM は、サービスレベル達成のアクティビティを実施する。サービスレベルの管理は、3.8.4.1 節でさらに詳しく解説する。

3.8.3 シナリオ例

自己管理機能は、グリッドに欠かせないものである。本節では、現実に使われている 2 つの例を紹介する。それは、ジョブレベル管理とグリッドのシステムレベル管理である。同じような概念は、セキュリティなど他のシナリオに対して自己管理がどのように機能するかを解析することで得ることができる。以下のシナリオで使われている用語は、OGSA 実行管理サービス機能と商用データセンターの使用例において使われているものである。

3.8.3.1 ジョブレベルの管理

IT ビジネスのアクティビティマネージャは、ジョブをサブミットし、そのジョブの SLA とネゴシエートする。ジョブはアグリーメントを満たすように実行しなければならない。IT ビジネスのアクティビティマネージャは、後になって新たなビジネス要件を付け加えるためにそのアグリーメントとネゴシエートする必要が出てくるかもしれない。ネゴシエートはジョブマネージャとともに行われ、既存のアグリーメントを書き換えることになる。アグリーメントが書き換えられると、リソースの要件がサービスレベル達成ループで再計算され（解析とプロジェクションの段階。図１６参照)、条件変更の結果としてリソースのアロケーションやデプロイメントなどのプロビジョニングの作業が開始される（アクションの段階)。プロビジョニング作業の後、アプリケーションサーバや DBMS などの実行可能なリソースを含め、リソースは必要とするジョブのコンポーネントを開始させることができる状態となる。

IT ビジネス・アクティビティマネージャは、プロビジョンされたリソースに対して、監視サービスやメータリングサービスを使ってロードやリソースの利用率を監視することができる。IT ビジネス・アクティビティマネージャはまた、ジョブマネージャから実行中のジョブの状態を取り出すことができる。

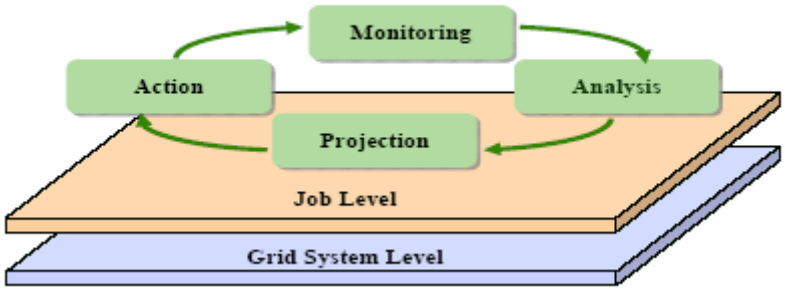


図１６：シナリオ例：グリッドシステムレベルとジョブレベル

3.8.3.2 グリッドシステムレベルの管理

グリッドシステムレベルでは、リソースの利用率を向上させ、実行中ジョブの SLA を維持することが目的である。グリッドシステムのサービスレベル・マネージャは、予想されるロードの上昇に対応するために新たなリソースを追加したり、コスト削減のために余分なリソースを開放したりする必要性が出てくることがある。ジョブにアロケートされたリソースは、グリッドシステム上の他のジョブに対する優先度など、ポリシーに基づいて調整する必要がある。

解析とプロジェクションの段階では、使用できるリソースや現在のロードに関する情報、そして予想される今後の利用率やロードについて評価が行われる。利用率の上昇が予想される場合、グリッドシステムで使用できるリソースプールに対してリソースを追加するプロビジョニングを行う（アクション）ことがある。これはたとえば、他のシステムからリソースを取ってきたり、優先度の低いジョブが現在使用しているリソースを開放したりするなどして行われる。

3.8.4 機能

自己管理は、基本的には自己設定、自己修正、自己最適化である。この点において、自己管理は他の OGSA のカテゴリーとは本質的に異なる。すなわち、自己管理を実行する際に使われるコンポーネントだけでなく、その方法が関係してくるのである。この方法とは、「ソース」と呼ばれ、環境の変化に応じながらコンポーネントが相互作用する様子、コントロールループの作られ方、システムがインテリジェントに機能する様子などを指している。自己管理を行うメカニズムは次のように説明される。

- 自己設定メカニズムは、IT の専門家が与えたポリシーを使って、IT システムの変化に動的に適応するものである。そうした変化は、リクエストのプロビジョニングを行うことに通じ、さらにはおそらくワークロードの多大な増加あるいは減少などにより、たとえば新しいコンポーネントのデプロイメントや既存のコンポーネントの削除などが行われる。
- 自己修正メカニズムは、リソースやサービスの不適当な作動あるいはそれらが行っている不適当な作業を検出することができる。さらに IT 環境に影響を与えることなくポリシーに基づいて修正作業を行うことができる。自己修正には、自己保護の要素も含まれている。すなわちコンポーネントは、敵対的な動作が発生するとそれを検知し、コンポーネントが攻撃を受けにくくするように修正アクションを起こすのである。敵対的動作には、無認可アクセスや無認可使用、ウイルスの感染や増殖、サービス拒絶攻撃などが含まれる。
- 自己最適化メカニズムは、エンドユーザやビジネスのニーズに合うよう、効率を最大限に高める調整を行うことができる。調整アクションは、SLA を施行することで全体的な

利用率や最適化を改善するため、リソースを再アロケーションすることに相当すると言うこともできる。自己最適化は実装の際に自己設定を使用する。

自己設定、自己修正、自己最適化のメカニズムは、互いに独立でないことに注意すべきである。これらは、IT システムの 1 つのあるいは複数のアスペクトのコンフィグレーションに対して変更を加えるため、共同で作業するのである。

これらの OGSA サービスカテゴリは、自己管理を行うためにすべて利用される。さらに以下の節では、自己管理に大切な特別機能要件に焦点を当てる。ただし、他の OGSA サービスカテゴリに対しては同じようには当てはまらない。

3.8.4.1 サービスレベル管理

サービスレベル管理は、必要な QoS が維持されるようにするものである。3.8.1 節に示したように、サービスレベル管理は SLM の動作を通じて行われる。こうしたサービスレベル達成の作業について、ここでさらに詳しく解説する。

- 監視：SLM は監視コンポーネントを通してリソース・インストルメンテーションを受け取り、処理する。サービスの実行やリソース利用の監視（たとえばリソースのロードや利用率、サービスコンポーネントの動作状況などを監視し、不具合を検出すること）は、OGSA の一般的なリソース管理機能が提供する監視サービスを使用して行うことができる。そのような監視情報は、解析段階で入力として利用する。
- 解析とプロジェクション：決められたポリシーと SLA に対する準拠状況を評価、決定するために、インストルメンテーションに対して解析を行う。マネージャはリソースが QoS の目的に合致し、決められたポリシーの範囲内で作動しているかどうかを知る必要がある。解析はまた、履歴とプロジェクトされた要件に基づいて今後のリソースの作動を予測することができる。システム動作が全体的な目標に合致しない場合、影響の及ぶ範囲で組み込まれたリソース集団に対して変更を加えるようマネージャは別のアクションを検討し、与えられたポリシーにしたがってアクションプランを選択する。
- アクション：マネージャは、基礎にある管理されたリソースとやりとりすることで、あるいはシステムの他のアスペクトを扱う他のマネージャと通信することで、そのプランを実行する。たとえばワークロード・マネージャは、サービスレベルの目的やポリシーに合うよう、プロセッサクラスタ内で実行されているタスクの優先度とプロセス共有について調整を行うことができる。あるいは、リソースプールに対するローカルなアクションが（ポリシー違反または制約上のため）不可能であったり効果がなかったりする場合は、リソースマネージャが他の管理機能に対して状況を改善するよう「アピール」することができる。たとえばワ―クロードマネージャは、クラスタに対してプロセッサをさらに追加するようプロビジョニング・リクエストを出すことができる。

このコントロールループは連続的に実行され、与えられたサービスレベルの目的に対するシステムの現状を評価し、システムをそれに見合う形に持って行くために必要ならば調整を加える。こうした作業を成功させるには他のサービスが必要となる。たとえばキャパシティ・プランニング、リソースのエンタイトルメント、問題と要因の解析、予想解析などのサービスである。

多くの管理コンポーネントがサービスレベル達成の作業に関与することがある。こうした管理作業が求める QoS や SLA の要件は、きわめて多様である。たとえば、プロセッサのキャパシティや利用率（ワークロード管理）などのパフォーマンス達成目的や、セキュリティレベルのような定性的目的なども含まれる。サービスレベル達成作業は、互いに直接的、間接的に影響を及ぼし合う。したがってコントロールループやサービスレベル・マネージャにおける異なる管理コンポーネント間の関係や実行順序は、かなりルーズであるべきである。

3.8.4.2 ポリシーおよびモデルに基づいた管理

サービスレベル・マネージャは、その動作をコントロールするポリシーに基づいて、動的管理インフラストラクチャにおけるリアルタイムの変更を組織的に行うものである。自己管理における情報は、SLM や管理下にあるサービスに関するポリシーやモデルによって基本的に支配されている。

3.8.4.3 エンタイトルメント

エンタイトルメントは、適切なリソースを得るために他のリソースホルダ（他の SLM、人間のオペレータ、リソースプールなど）とネゴシエートすることに関係している。ネゴシエートのプロセスでは、さまざまな SLA を比べてどのリソースニーズがより重要かを見つける。エンタイトルメントは、リソース予約の前段階であり、予約の 1 つのオプションとしてよりルーズなバインディングを提供するものである。これに対して、リソース予約はリソースへのアクセスを保証するものである。

3.8.4.4 プランニング

プランニングは、サービスの最適な要件を、そのサービスが必要とするリソースに関して計算することを指している。これは、問題や原因が特定された場合や、高いレベルからコマンドを受けた場合に、初期の段階で行われる。

3.8.4.5 キャパシティ管理

キャパシティ管理には、リソースの現在の状況や要件を更新することに関連した実際のアクションも含まれる。たとえば在庫との連携、アセット管理、リソースを現在の位置からサービスドメインへ移動、リソースをサービスドメインからストレージエリアや別のド

メインへの移動、といった作業もキャパシティ管理である。

3.8.4.6 プロビジョニング

デプロイメントやコンフィグレーションなどの副次的作業を含め、プロビジョニングは、リソースを予定された利用目的に向けて準備する際、自己管理作業をサポートするために行われる重要な作業である。プロビジョニングは、デプロイメントのコンテンツやコンフィグレーションの記述を維持するアプリケーションコンテンツ・サービスなど、他の多くの OGSA サービスからサポートを受けている。

3.8.4.7 分析

- 問題と原因の解析：急な問題や検知された問題の要因を見つけ出すため、監視データを解析する。フィルタリング機能やコリレーションなどもここに含まれる。
- 予測解析：今後のリソースニーズの計画を立てたり、将来の問題点を考えたりする目的で、未来の動きを予測するために監視データを解析する。

3.8.5 プロパティ

サービスレベル管理コンポーネントは、ポリシーに基づいた自己管理機能を管理インフラストラクチャの複数の QoS 特質に対して実行する。重要な QoS の特質としては、アベイラビリティ、セキュリティ、パフォーマンスなどがあるが、これらに限られているわけではない。こうした重要な QoS の特質は、SLA に取り入れられている測定可能な目標を使って表現されている。アベイラビリティは「機能停止時間」で測られ、必要とされるセキュリティ機能は、あるクラスのサービスが関係するユーザに与える特権として表現される。パフォーマンス特性は、よく知られたスループットや応答時間の目標値で表される。

3.8.5.1 アベイラビリティ

これは、サービスの不具合率を表す数的指標をとくに示すものである。そうした不具合にはあらゆる種類があり、システムクラッシュなどのマクロなケースや、スループットの低下やバッファのあふれによってサービスが提供できなくなるといった観測しにくいケースもある。

3.8.5.2 セキュリティ

セキュリティは IT 環境のきわめて重要なアスペクトである。セキュリティ違反、セキュリティ違反率、ユーザの設定や削除にかかる時間などのアイデンティティ管理指標等もこれに含まれる。

3.8.5.3 パフォーマンス

これは、たとえばシステムがどのように作動しているかを定量的に表すデータ（CPU の積算負荷率など）など、測定量や数的指標を与えるものである。これらは特定のサービスのパフォーマンス指標の計算に関与する。

3.8.6 OGSA の残りの部分とのやりとり

自己管理は、OGSA の他のほとんどすべてのアスペクトと相互作用する。必要な相互作用についてはまだ作業中のため、期待される相互作用の例としてすべてを網羅せず、一部のサービスのみここでは列記することにする。

- 検出は、新しいリソースやサービスを検出、統合するものである。
- ロギングと監視は、システムの状態を知る上で必要な情報を提供するものである。
- リソース予約は、リソースの利用をさらに予測可能なものにする。
- ワークフローは、SLM が異常な状態を解決する際に実行しなければならないアクションを、自動化するものである。
- コンポジションは、簡単な SLM から複雑な SLM あるいは高レベルの SLM を構築するものである。
- セキュリティ、とくに認証と認可は、いろいろなシステムコンポーネントが管理アクションに関与する際に重要となる。
- リソース管理、とくにサービスやリソースのマネージャビリティモデルは、管理されているサービスやリソースの説明を与えるべきものである。SLM はこの情報を読み、変化する環境やより高いレベルからのコマンドに対応して情報に応じて作動する。サービスやリソースをどのようにモデル化すればよいのかを決める作業も、さらに行う必要がある。

3.9 情報サービス

3.9.1 目的

グリッド環境のアプリケーション、リソース、サービスに関する情報に対し、効率的にアクセス、操作する機能は OGSA の重要な機能である。本節では情報という言葉を、ステータスの監視に使用する動的データやイベント、検出に使用する比較的静的なデータ、ロギングされるあらゆるデータの意味として用いる。とくに情報サービスは、信頼性やセキュリティ、パフォーマンスのためのさまざまな QoS 要件を満たす必要がある。OGSA の情報サービスは、パブリケーションからコンサンプションまでカバーする。パブリケーションの前段階（ネットワークパケットのスニフィング）やコンサンプション後に生じることについては、対象外である。

1 つの情報サービスだけであらゆる情報デリバリパターンや QoS を取り扱うように設計することは可能かもしれないが、OGSA にとっては情報サービスが複数あった方が役に立つ。そのうちのいくつかは一般的なものであり、またいくつかは特定の使用例に合わせて

最適化したものである。ただし、限られた使用例にしか応えることのできないような不完全な情報サービスにならないよう、注意が必要である。問題は、ドメイン固有のセマンティクスや QoS に対する一般性や要件の要請についてバランスを取ることである。理論的には一般性は、できるだけ多くの使用例の要件をカバーする共通の（高レベルな）機能や特徴を抜粋することで達成することができる。共通のセマンティクスにより相互運用可能な実装が実現できるが、それらは高レベルすぎて望んだとおりの動作を十分にエクスポーズしない可能性がある。サービスのユーザビリティを落とすことなくアブストラクションのレベルをどれくらい高めるかは、さらなる検討が必要な問題である。

OGSA の情報サービスのクライアントには、実行管理サービス、会計サービス、問題特定サービス、リソース予約サービス、リソース使用サービス、アプリケーション監視などが含まれる。ただしこれらに限られているわけではない。相互運用性や再使用を促すには、インフォメーションサービス自体が通知機能（たとえば WS 通知[WS-N]）などの OGSA インフラストラクチャ機能の上に構築される必要がある。情報サービスはまた、データアクセスや分散クエリ処理といったその他の OGSA 機能を利用することもできる。

3.9.2 モデル

情報サービスの特徴は、情報のソースに関する要請（たとえば静的か動的か、またはパブリケーション・レートなど）やその目的（検出、ロギング、監視など）、あるいは QoS 要件などの要素に大きく関係している。しかし情報サービスには、同様に繰り返し発生する構造が見られる。情報は、もともとのプロデューサ、あるいはプロデューサの代わりに作業している仲介（ロギングサービスや通知ブローカなど）を通じて、コンサンプションに使えるようになる。1 つあるいは複数のコンシューマが 1 つあるいは複数のプロデューサから情報を得たいとする場合と、1 つあるいは複数のプロデューサが 1 つあるいは複数のコンシューマへ情報を送りたいとする場合がある。プロデューサとコンシューマは、分離されるべきであり、互いについてあらかじめ知識を持つ必要はない。コンシューマはプロデューサ（あるいは仲介）に接触し、コールによって情報を引き出すことができる。あるいはコンシューマが、情報が使えるようになった際にそれを受け取るサブスクリプション・メカニズムを使用することもできる。

OGSA は、情報サービスを施行する際に使用するデータモデルや、情報をクエリする際に使用する言語について、規定はしていない。現行のシステムは、XML や XPath／XQuery クエリ言語（たとえば Globus　MDS [Globus MDS]）に基づいたシステム、あるいはリレーショナルモデルと SQL クエリ言語（たとえば R-GMA[R-GMA]）を使用するシステムに、大まかに集約される。

メタデータは、情報の構造やプロパティ、使用について記述した情報（イベントやメッセージなど）と関係している。相互運用性のためには、OGSA 情報サービスの標準的なイベントスキームが必要である。最高のパフォーマンスにより相互運用性が問題ではないと

きなど、場合によっては、ユーザ定義の最適化されたイベントのほうがより適切なことがある。

プロデューサやコンシューマは、情報サービスを使って、自分たちに関する詳しい記述をクエリで使えるようにすることで、互いを検出することができる。この目的で、特別な分散レジストリやポイントツーポイントのメカニズムが使用できる。プロデューサやコンシューマの種別や、それらがプロデュースあるいはコンシュームする情報、さらにはそれらのエンドポイント URL なども記述に含まれる。

3.9.3 シナリオ例

3.9.3.1 ディレクトリシナリオ

ユーザは、いくつかの必要な機能上の基準やその他の基準に合致するサービス記述をロケートする必要がある。ユーザは、サービス記述がパブリッシュされた場所をセントラルディレクトリサービスにクエリし、答をもらう。ディレクトリに誰が情報をパブリッシュできるか、誰がそれをクエリする許可を得ているかということは、ディレクトリの所有者が決める。UDDI は、レガシ・ディレクトリサービスの 1 例である。OGSA ディレクトリサービスは、WSRF サービスグループの概念に基づいている。

3.9.3.2 ロギングシナリオ

多くの分散型計算使用シナリオは、ロギングサービスを必要とする。これらのシナリオには、問題特定、使用メータリング、不具合からの回復、トランザクション処理、セキュリティなどに基づいたシナリオも含まれる。

問題特定シナリオは、次のように進む。ある開発者が災害復旧のアプリケーションコードを書いている。会社のロギングに関するプログラミング・ガイドラインに沿って、彼はログステートメントをコードに入れ、予期せぬ状況に関する情報を提供するようにする。彼は、候補となるフェイルオーバ・リソースに対する属性を取得するために RDBMS にアクセスするよう、アプリケーションの一部をコーディングする責任がある。彼はアプリケーションのこの部分を、RDBMS の不具合を検出してそれを参照するためのログ記録を生成するようにコーディングを行う。アプリケーションの開発が終わると、ハンドラ・チェインがログ記録を処理する作業環境に、そのアプリケーションをデプロイする。この環境では、重大度レベル（致命的か、あるいは警告や情報程度か）に基づいて、オペレータの設定したレベルを超えた記録がファイルハンドラによってログに保存される。間違った新しいセキュリティポリシーがデプロイされると、コードは異常な状態に達し、基礎にある RDBMS がそのアプリケーションへのアクセスを禁止するとの記録をログに残す。オペレータは、この重大度のログ記録を保管するために作業環境をすでに設定している。不具合が生じた後は、ログのコンシューマであるアナリストが、コンソール・アプリケーションを使ってログを詳しく調べる。そして RDBMS の問題を指摘するログステートメントを見

つけ、セキュリティポリシーを修正するようデータベース管理者に指令を出すのである。

3.9.3.2.1 グリッド監視アーキテクチャ（GMA）シナリオ

ある科学者が、30 分以内に終了しなければならない対話型シミュレーション／レンダリングのジョブを走らせたいと考えた。それには十分に速く、十分なメモリを搭載し、高速ネットワークで接続された 100 個の計算エレメントが必要である。彼は最新の情報を必要とし、複雑なクエリを OGSA 情報サービスに送る。すぐに彼は、クエリに対応した計算リソースに関する情報を手に入れる。使用可能なリソース候補の中から彼は一番近くにあるものをいくつか選び、ジョブを走らせる。

クエリに答える際、情報サービスはまず、関係する計算、ストレージ、ネットワークの各リソースに対する情報のプロデューサを見つけ出し、必要な情報を手に入れてから、クエリを満足させるリソースを特定するために「ジョイン」する。科学者は自分のジョブを監視し、実行環境で部分的な不具合が生じた場合に対処する必要がある。それにはおそらく GMA ベースのサービスを用いる（その情報が関連するタイムスタンプを持っている場合）。GMA は、マルチパーパスの情報サービスアーキテクチャの 1 例である[GMA]。

3.9.3.3 プロデューサ／コンシューマ・パターン

プロデューサとコンシューマは、ときおり互いに直接対話することができる。プロデューサとコンシューマの直接のやりとりが適切ではない場合、あるいは不可能な場合は、プロデューサを仲介を使ってコンシューマから切り離すという基本パターンが広く用いられている。プロデューサ－仲介－コンシューマというパターンにおいては、プロデューサはデータを仲介に渡し、コンシューマは仲介からデータを引き出す。一般的に、これはあらゆるデータ保管が行っているパターンである。すなわち、プロデューサはファイル、RDBMS、ODBMS などへ書き込みを行い、コンシューマはそこから読み出しを行うのである。仲介は、プロデューサとコンシューマのインタフェースを支援するだけでなく、管理インタフェースもサポートすることができる。管理インタフェースは、データ保管の読み込みや書き出しとは直接関係していない機能を制御するものである。記録ポリシーやバックアップポリシーなどを制御するオペレーションは、管理インタフェースを通じてアクセスする。

分散型計算では、コンシューマに情報を提供する仕事を持ついくつかのインフラストラクチャサービスが、上述のようなパターンに従う。これらのサービスを広く情報サービスと呼ぶことにする。そこにはネームサービス、ロギングサービス、通知サービスも含まれる。読者はこれらの概念に一般的に通じているものと仮定し、各サービスに関する OGSA 独自の議論は行わないことにする。

これらのサービスは、いずれもドメイン固有の特定のセマンティクスやサービスの品質を満たすように進化してきた。それぞれのサービスは、基礎となるデータ保管のために一般目的の RDBMS を使って実装することができるが、その結果、望ましい特殊なサービス

品質を不必要な機能のために犠牲にする場合が多い。たとえば、高速書き込みをサポートしたログサービスは、フル機能 RDBMS のパフォーマンスとフットプリントのコストが極めて高い場合が多い。ネームサービスや通知サービスでも同様のことが起きる。

WS 通知[WS-N]からは、OGSA 情報サービスが使うインタフェースのコアセットが与えられる。その機能には、パブリッシュ／サブスクライブ・モデル、興味のあるアイテムを「トピックス」として知られるサブスクライバに組織化するメカニズム、パブリケーションやサブスクリプションの管理インタフェースなどが含まれる。基盤にあるメッセージング･インフラストラクチャのサービス品質を扱う他の Web サービス仕様についても､OGSA 情報サービスは開発のためにこれを考慮すべきである。たとえば WS 信頼性[WS-Reliability]は、ネットワークやシステムに不具合が生じた場合でも、信頼の置けるメッセージ伝達を可能にするものである。

3.9.4 機能

ネーミング、検出、メッセージ伝達、ロギング、監視の各機能について定義する。

3.9.4.1 ネーミングスキーム

従来の分散型システムでは、２層あるいは３層のネーミングスキームが通常はサポートされている。OGSA では、OGSA ネーミングと呼ばれるネーミングサービスが３レベル仕様となっている。名前をつけられた OGSA のエンティティは、いずれも（オプションの）ヒューマン指向の名前、抽象名、アドレスに関連付けられている。

ヒューマン指向の名前は通常、人間が読むことのできるもので、ネームスペースに属すことができる。ネームスペースは、普通は階層構造をしており、統語的制約がある。階層構造をしたネームスペースでは、名前の各部分を特定のコンテクストにマップすることができる。たとえば UNIX のファイル名 node1:/var/log/error では、var のコンテクストは node1 と名づけられたマシンのルートディレクトリであり、log ディレクトリのコンテクストは、var である。そして error ファイルのコンテクストは log である。OGSA ネーミングでは、ヒューマン指向の名前を１つに限定することはない。多くの異なるネーミングスキームが存在するため、使用できるスキームをすべて本文書で紹介することはできない。

抽象名は、特定の場所を指定しないパーシステントな名前である。抽象名の他の特性、たとえば一意性などは、検討中である。更新可能なリファレンスのある WSRF エンドポイントリファレンス[WS-RF]や Legion オブジェクト識別子などは、抽象名の１例である。本文書の範囲外となるが、ヒューマン指向の名前を抽象名にマップするメカニズムが必要である。

アドレスは、エンティティの場所を示すものである。アドレスの例としては、エンドポイントのアドレスと、WSRF エンドポイントリファレンス[WS-RF]、メモリアドレス、IP アドレス/ポートのペアにおけるリファレンスプロパティとの組み合わせも、アドレスの１

例である。本文書の範囲外であるが、抽象名をアドレスに結びつけるメカニズムが必要である。

OGSA では、リソースハンドルの存在を仮定している。リソースハンドルとは、リソースとそれに関連したステート（ステートが存在する場合）の抽象名である。

3.9.4.2 検出

広く必要とされる機能の１つが、サービスやリソースの検出である。ディレクトリ（あるいはレジストリ）は明らかな１つの答であるが、これに限られているわけではない。ディレクトリが他の解決策と違う点は、それが「最新」の情報のためにパーシステントなストレージを持っており、検索のために最適化されているということである。大量のクエリに対する待ち時間の短い応答が求められる。ディレクトリは、スケーラビリティのために複製することができる。

あるいは、情報を編集したものや情報のガイドを（たとえば Google のように）「インデックス」に保存することもできる。中央からコントロールされるレジストリと違って、インデックスは誰でも作成することができる。

別の方法としては、ピアツーピア検出がある。これは、Web サービスがピアのネットワークにおける１つのノードとなり、合致するものがあるかどうか周囲に動的にクエリを送るものである。クエリはネットワークを通じてノードからノードへと伝わり、合致するものが見つかるまで、あるいはある伝達回数に達するまで、または他の停止基準に達するまで、これが続けられる。さらに別の検出サービスとして、GMA ベースの一般的なサービス（下記参照）もある。

3.9.4.3 メッセージ伝達

プロデューサとコンシューマは、メッセージの交換をすることで相互作用するが、これは共通のメッセージング・インフラストラクチャが行う。このインフラストラクチャは、メッセージのコピーを対象となる相手にいかにして送るかという問題だけに的を絞っており、メッセージが最初にどのように作られたかということには関係しない。プロデューサは、関係するコンシューマに直接メッセージを送るか、あるいはプロデューサをコンシューマから切り離す仲介（メッセージブローカ）を利用する。後者の場合、プロデューサはメッセージをブローカに送り、ブローカが責任を持ってそのメッセージを対象となる相手に転送する。プロデューサ（あるいは仲介）は、通知機能や、（プロデューサとコンシューマが互いを検出する）ファインダサービスのような追加機能を提供することもできる。メッセージブローカは、メッセージを安定なストレージに保管し、そこから転送することができる。その際のメッセージ伝達は、パーシステントである必要はないが、信頼の置けるものでなければならない。

3.9.4.4 ロギング

OGSA のロギングサービスは、ログ・アーチファクトのプロデューサとコンシューマの間に立つ仲介である。プロデューサは、ログ・アーチファクトを次々に書き、コンシューマはそのログ記録を読む（ただし更新は行わない）。基本的なログのセマンティクスを一般的に受け付けるようにするため、そして既存のインプリメンテーションを使うため、OGSA ロギングサービスは、既存のロギング・インプリメンテーションがもつ重要な特性をサポートする必要がある。ある１つのロギングの仲介に対して複数のプロデューサやコンシューマがいる場合があり、どちらも記録にフィルタをかけていることがある。ロガーサービスでは、メッセージ交換はパフォーマンス向上のために最適化されており、その記録はパーシステントなストレージに一定の期間保管されている。

3.9.4.5 監視

順番をつける目的でフィールドを持っている情報（タイムスタンプやシーケンス番号など）は、監視に利用することができる。OGSA の監視サービスは、アプリケーションにもリソースにも等しく使うことができる。リアルタイムのアプリケーションなど、場合によっては、監視サービスに厳しい要件（たとえば高い更新頻度や高パフォーマンスなど）が課せられることがある。そうした場合、特別目的のサービスが必要になることがある。

3.9.4.6 一般的な情報と監視のサービス

上記の機能の組み合わせを提供する一般的な OGSA サービスは、エンドユーザにさらなるフレキシビリティを提供することができる。（サービスやアプリケーションを含む）一般的なグリッドリソースにとって、リソースに関する利用可能な情報の量は膨大なものであり、ネットワーク全体に広がっている。また、更新も頻繁である。このような空間で検索を行うと、無用な待ち時間が生じる可能性がある。それらの情報を制御可能な方法で管理するには、情報源の検出と情報の伝達とを分離することが重要である。検索は情報のソースとシンクを特定するためだけに使用するべきである。リソースに関するメタデータを保管するには、特定目的のディレクトリを使う。その場合ディレクトリは、動的な OGSA 環境で予測される高頻度のアップデートに対処しなければならない。それぞれのプロデューサ／コンシューマのペアは、お互いの間を行き来するデータの量を、コンシューマのクエリに合うよう制限することができる。このモデルは、情報のソースとシンクを見つけるためのメカニズムと、１つの検索可能なチャネルへの情報伝達とを組み合わせたメッセージブローカとは異なるものである。このアプローチの利点については GMA ドキュメント[GMA]に記載されている。

このような一般的なサービスのユーザは、システムの複雑な面を理解していなくても、あらゆる情報をそのサービスに投入することが可能でなければならない。その際、利用目的（検出、監視など）は関係ない。ユーザは、OGSA ではどのような情報を使用できるの

か、その情報の種類と構造はどこで定義されているべきなのか、ということを最初に明確にする必要がある。ユーザはまた、特定のプロパティやポリシーを指定したいと考えるかもしれない。そこには、情報の保持方法、保持期間、伝達保証、パーシステンシ、アクセスコントロールなども含まれる。コンシューマは、サブスクリプショントピック等を使って、関心のある情報にフィルタをかけることもできる。また、クエリ表現で定義されたプレディケートを通じてコンシューマに届けられたイベントをさらに改良するためのクエリもサポートしている。

より高度なユーザは、サービスの内部作業を深く理解したいと考えるかもしれない。一般的な（GMA ベースの）サービスにおいて情報に対するリクエストが出されると、「仲介」機能を使ってレジストリやスキーマを検索し、情報の適切なソースを特定することになる。長期にわたるクエリの場合、ソースが消失していたり新しい関連ソースがオンラインに現れていることから、サブスクライブされたコンシューマを仲介がアップデートする。仲介はまた、関連するプロデューサに配るクエリの計画を立て、その結果をまとめる作業も行う。

3.9.5 プロパティ

3.9.5.1 セキュリティ

コンシューマとプロデューサに対する認証と認可のルールにより、両者は安全に情報を交換することができる。プロデューサとコンシューマに関するメタ情報（両者の存在や、プロデュースあるいはコンシュームする情報の種類など）を検出する作業も、セキュリティのルールに従う必要がある。メータリング、認証、認可など、サービスによっては安全なメッセージ伝達（暗号化など）が必要になることがある。

3.9.5.2 サービスの品質

さまざまなレベルの QoS が OGSA 情報サービスにより提供されている。信頼の置ける、保証されたメッセージ伝達などもそうである。OASIS WS-RM TC により定義された WS 信頼性により、こうした高い信頼性のメッセージ伝達のプロパティが提供されている [WS-Reliability]。

3.9.5.3 アベイラビリティ／パフォーマンス／スケーラビリティ

情報システムは、OGSA において重要な役割を持っている。OGSA の他の機能の多くは、情報システムを利用しているため、情報システムはつねに使用可能な状態になければならず、とくに部分的な不具合についてはトレラントである必要がある。情報システムの多数のクライアントは、情報をすぐに受け取れるものと期待し、長時間待つことができない。それには高性能なシステムが必要になる。非常に多くのリソース、サービス、アプリケーションが情報をプロデュース、コンシュームすることがあるため、広いネットワークに渡

ってシステムがスケーラブルでなければならない。

3.9.6 OGSA の他の部分とのやりとり

標準イベントデータモデルは、プロデューサからコンシューマへ情報を渡す操作を行う。認識された標準イベントスキーマを使うことで、VO 内のさまざまな異なるソースからイベントを検出、処理することができる。理想的には、仲介のコストを避けるため、リソースイベントモデルとインフラストラクチャイベントモデルは密接に提携しているべきである。さらにイベントスキーマは、ドメイン固有のイベントに対応するため、拡張が可能である必要がある。OASIS WSDM-TC で現在開発中のイベントスキーマは、既存の CIM イベントモデルに対応し、OGSA イベントデータモデルの基礎として使用できると考えられる。

通知メカニズムあるいはその拡張版は、プロデューサとコンシューマが互いを検出した場合に、プロデューサからコンシューマへイベントを届けるために使用される。また、システムの他のコンポーネント間でイベントをやりとりする際にも使用される。

セキュリティサービスは、異なるコンポーネント間の認証・認可に必要となる。

分散型クエリ処理は、GMA 内の情報のプロデューサとコンシューマの間で、仲介がクエリの計画を立てる段階で必要になるものである。

単一障害点を避けてスケーラビリティを向上させるため、メタデータ（ディレクトリやレジストリ）の格納場所を分散させて複製を作るうえで、複製メカニズムが必要となる。複製したコピー間に一時的な不一致があったとしても、場合によっては問題がないことがある。その場合は、一貫性のないメタデータや古くなったメタデータに対して情報システムが強固である必要がある。

4 セキュリティ事項

セキュリティ事項はセキュリティサービスの節（3.7 節）で議論している。

編集者情報
Ian Foster
Distributed Systems Laboratory
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439, USA
Tel: +1-630-252-4619
Email: foster@mcs.anl.gov

Hiro Kishimoto
グリッドコンピューティング・バイオインフォマティクス研究所

富士通研究所
神奈川県川崎市中原区上小田中 4-1-1
Tel: +81-44-754-2628
Email: hiro.kishimoto@jp.fujitsu.com

Andreas Savva
グリッドコンピューティング・バイオインフォマティクス研究所
富士通研究所
神奈川県川崎市中原区上小田中 4-1-1
Tel: +81-44-754-2628
Email: andreas.savva@jp.fujitsu.com

寄稿者

本文書への寄稿に関し、Jeffrey Frey, Takuya Mori, Jeffrey Nick, Chris Smith, David Snelling, Latha Srinvasan, Jay Unger に感謝する。

謝辞

多くの共同研究者に対し、本文書で扱ったトピックスに関する議論について、感謝の意を表する。とくにMario Antonioletti、Takuya Araki、Jamie Bernardin、Shel Finkelstein、Steve Fisher、Dennis Gannon、Kate Keahey、Carl Kesselman、Takashi Kojo、Peter Kunszt、Simon Laws、Tan Lu、James Magowan、Susan Malaika、David Martin、Nataraj Nagaratnam、Dave Pearson、Benny Rochwerger、John Rofrano、Ellen Stokes、Paul Taylor、Steve Tuecke、Sachiko Wada、James Warnes、Philipp Wieder、Ming Xuに対し感謝する。（アルファベット順。記載漏れのあった方にはお詫びいたします。）

また、本文書の草稿を読み、コメントを寄せていただいた方々にも感謝したい。本文書の読みやすさや正確性を改善する上で貴重なコメントであった。

用語

用語に関しては、別冊のOGSA専門用語[OGSA Glossary]を参照。

知的財産権について

本文書に記載された技術の実装や使用に関連すると考えられるいかなる知的財産権およびその他の権利についても、GGFは、その有効性や範囲に関して特定の立場をとるものではない。また、こうした権利下にあるいかなるライセンスについても、それが使用できるあるいは使用できない範囲について、特定の立場をとるものではない。さらに、これらのいかなる権利についても特定する努力をGGFが行うものではない。出版やライセンス保証

のために用意した権利請求書、および、開発者や本仕様書の使用者が所有権を使用する上での一般的なライセンスや許可を取得する作業の結果については、GGF事務局から入手可能である。

関係者の方々は、本文書で薦められていることを実践する上で必要なあらゆる著作権、特許、特許利用、その他の所有権に対して注意を向けるようGGFは希望する。情報はGGF委員長までお寄せいただければ幸いである。（連絡先についてはGGFのウェブサイトを参照。）

参考文献

（原文参照）