

Configuration description, Deployment and  
Lifecycle Management Working Group

D. Bell, T. Kojo, P. Goldsack, S. Loughran,  
D. Milojevic, S. Schaefer, J. Tatemura, and P. Toft

草案:2004年4月9日

最終版:2005年8月7日

## コンフィグレーション・デスクリプション・デプロイメント・ライフサイクルマネジメント (CDDL M) ファウンデーション文書

### 本文書の位置づけ

本文書はコンフィグレーション記述・デプロイメント・ライフサイクルマネジメント(CDDL M) 言語の仕様に関する情報を提供するものである。文書の配布に制限はない。この仕様は、サービスコンフィグレーション記述、デプロイメント、およびライフサイクルマネジメントを共に構成する4つの仕様に関係している。これらの仕様については、以下の3つのパラグラフで説明する。詳しくはこれらの仕様を参照のこと。

### 著作権に関する注意

グローバル・グリッド・フォーラム(2002, 2005)がすべての著作権を有する。

### 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、採用されているフレームワークに関し、オープン・グリッド・サービス・アーキテクチャ(OGSA)という考え方を実現させるには、Web サービスやそのデプロイメントを設定し、ライフサイクルマネジメントを維持するための手立てが必要である。グローバル・グリッド・フォーラム(GGF)内の CDDL M ワーキンググループが作成した本文書では、CDDL M を概観し、要件の記述や使用例の解析を提示している。さらに、GGF や他の標準団体、さらには産業界や学術分野における関連作業についても概説する。本文書は、言語、コンポーネントモデル、基本的な CDDL M サービスについてより詳しく解説する後続文書のためのたたき台となるものである。

### CDDL M の仕様

CDDL M グループが作成した文書は5つある。以下でそれらを解説する。

CDDL M ファンデーション文書は、他の文書のたたき台となるものであり、この分野の紹介や、機能面での要件、使用例、ハイレベルなアーキテクチャを解説したものである。さらにファウンデーション文書では、CDDL M グループと、他のワーキンググループおよび産業界の他の関連作業との比較を行っている。

SmartFrog 言語仕様は、コンフィグレーション記述とデプロイメントを主に意図した言語を記述するものである。この言語は宣言型なので、属性値ペアをサポートしている。さらに、インヘリタンス、リファレンス(レイジーを含む)、パラメタリゼーション、プレディケート、スキーマもサポートしている。SmartFrog 言語は、CDL(次の段落を参照)と同じ機能を持つが、XML ベースではない。SmartFrog 言語は CDL の以前から存在し、CDL を作成する際にモデルとして使用された。この 2 つの言語はコンパチブルである。CDL は主に機械を意図して作られたものであり、SmartFrog は人間を対象にしたものである。

CDDL M コンフィグレーション文書言語(CDL)は、XML ベースの言語であり、CDDL M コンポーネントモデルで定義されたコンポーネント(デプロイメント・オブジェクト)によって構成されているシステムコンフィグレーションの宣言型記述のためのものである。デプロイメント API は、システムのデプロイメント・ライフサイクルを管理するため、CDL で書かれたデプロイメント記述子を使用する。CDL は、特定のデータ依存性を保持しながら動的にデータをアサインすることができるよう、値参照を含め、コンポーネントのプロパティ(名前、値、種別)を記述する手段である。システムはコンポーネントの階層構造として記述される。CDL にはまた、プロトタイプに基づいたテンプレート機能(プロトタイプ参照)もある。これによりユーザは、システムを記述する際に、コンポーネントのプロバダが与えたコンポーネントの記述を参照することができる。

CDDL M コンポーネントモデルとは、デプロイされたリソースのライフサイクルに関するデプロイメント・オブジェクトを作成する際の要件を概説したものである。各デプロイメント・オブジェクトは、CDL 言語を使って定義されており、そのインプリメンテーションにマップされている。デプロイメント・オブジェクトは、管理されたリソースにおけるライフサイクルオペレーションのために、WS-リソース・フレームワーク(WSRF)に準拠した「コンポーネント・エンドポイント」を提供する。また、CDDL M コンポーネントモデルは、制御可能なアグリゲート・ライフサイクルや、このプロセスを可能にするオペレーションを提供する目的で、オブジェクトと CDDL M デプロイメント API のやりとりを管理する規則を定義している。

デプロイメント API は、対象となる 1 つあるいは複数のコンピュータに対してアプリケーションをデプロイするための、WSRF ベースの SOAP API である。システムをデプロイできるコンピュータのセットは、いずれも 1 つあるいは複数の「ポータル・エンドポイント」をホストする。ポータル・エンドポイントは、新しい「システム・エンドポイント」を生成する手段を与える WSRF リソースである。システム・エンドポイントは、デプロイされたシステムを表している。呼び出し側はシステム・エンドポイントに対してファイルをアップロードし、デプロイメントのためにデプロイメント記述子をサブミットすることができる。システム・エンドポイントは、コンポーネントモデルの仕様からすれば、実質上、1 つのコンポーネントであり、その仕様で定義されているプロパティやオペレーションを実装するものである。さらにシステム・エンドポイントにより、デプロイされたシステム内でリファレンスを解析する機能が追加されるため、これを使ってリモートの呼び出し側がコンポーネントの状態を調べることができる。

目次 (訳注:原文の目次ページとの番号と一致していませんが、本文の見出し番号に合わせてい

ます)

図のリスト

表のリスト

はじめに

1 CDDL M ワーキンググループと本文書の目的

2 機能上の要件

- 2.1 基本機能要件
- 2.2 セキュリティ要件
- 2.3 リソース管理要件
- 2.4 システムプロパティ要件 (非機能性要件)
- 2.5 コンフィグレーション記述言語 (CDL) の要件
- 2.6 サービスインタフェース要件

3 使用例

- 3.1 グリッド環境における Web サービス
  - 3.1.1 概要
  - 3.1.2 ユーザ
  - 3.1.3 シナリオ
- 3.2 商用データセンタ
  - 3.2.1 概要
  - 3.2.2 ユーザ
  - 3.2.3 シナリオ
  - 3.2.4 関連リソース
- 3.3 IT インフラストラクチャと管理
  - 3.3.1 概要
  - 3.3.2 ユーザ
  - 3.3.3 シナリオ
  - 3.3.4 関連リソース
  - 3.3.5 使用例の状況解析
- 3.4 ユーティリティコンピューティングモデル
  - 3.4.1 概要
  - 3.4.2 ユーザ
  - 3.4.3 シナリオ
  - 3.4.4 関連リソース
- 3.5 複雑なビジネスサービスの使用例
  - 3.5.1 概要

- 3.5.2 ユーザ
  - 3.5.3 シナリオ
  - 3.5.4 関連リソース
  - 3.6 Web サービス・デプロイメント・プロジェクト
    - 3.6.1 概要
    - 3.6.2 ユーザ
    - 3.6.3 シナリオ
    - 3.6.4 関連リソース
    - 3.6.5 使用例の状況解析
  - 3.7 OGSA プラットフォームに対する機能要件
  - 3.8 OGSA プラットフォームサービスの活用
  - 3.9 セキュリティに関する考慮
  - 3.10 性能に関する考慮
  - 4 ハイレベル CDDLM アーキテクチャ
    - 4.1 コンフィグレーション記述言語の仕様
    - 4.2 基本サービスの仕様
    - 4.3 コンポーネントモデルの仕様(表現と実装)
  - 5 関連 GGF、その他の標準団体のワーキンググループ
    - 5.1 GGF OGSA
    - 5.2 GGF OGSA 基本実行サービス
    - 5.3 OASIS WSDM
    - 5.4 OASIS SPML および WS-プロビジョニング
    - 5.5 CIM
    - 5.6 DCML
    - 5.7 WS-通知
    - 5.8 GRAAP
    - 5.9 JSDL
  - 6 関連する作業
  - 7 今後の作業
  - 8 セキュリティに関して
  - 9 編集者情報
  - 10 寄稿者
  - 11 謝辞
- 知的財産権について  
著作権情報  
参考文献

## 図のリスト

- 図 1 プロビジョニング・レイヤ
- 図 2 プログラム実行環境での使用例
- 図 3 IT インフラストラクチャと管理
- 図 4 ユーティリティコンピューティングモデル
- 図 5 複雑なビジネスサービスの概略図
- 図 6 CDDLML のハイレベルアーキテクチャ
- 図 7 CDDLML と他の管理コンポーネントとのやりとり

## 表のリスト

- 表 1 CDDLML の範囲

## はじめに

複雑な分散型サービスをデプロイするには、サービスのコンフィグレーションや管理に関係した数多くの問題がある。必要なサービスコンフィグレーションをいかに正確に記述するかという問題から始まり、サービスを自動的に繰り返しデプロイし、管理し、削除する方法の問題まで、さまざまである。記述の問題には、サービスやリソースの要素をすべて表記する方法や、サービスのテンプレート、サービスの組み合わせ、正確性のチェックをいかに行うかといった問題なども含まれる。デプロイメントの問題には、分散したリソースを横断してオペレーションを自動化したり、正しく順序立てたりする方法や、サービスのライフサイクルマネジメント、サービスの完全なる削除、セキュリティなどの問題も含まれる。これらの問題に対処するには、個々の Web サービスを設定し、デプロイすることなど、いくつかのレベルでグリッドコンピューティングと高度に関係することになる。さらに、共同で作動している数多くの Web サービスから構成されたコンポジット・システムとも関係している。

CDDLML は、たとえば OGSA 内の基本実行サービス(Basic Execution Service、BES) (BES は CDDLML に対してデプロイメントのリクエストをすることがある)や WS-アグリメント(リソースのアロケーションを容易にするもの)、OASIS WSDM(管理を容易にするもの)など、GGF や OASIS の他の分野とも関係している。

CDDLML をプロビジョニングと混同してはならない。後者はより広い意味を持っている。プロビジョニングは、予定された作業を完了するために必要なリソースを取得、管理するプロセスである。バッチジョブを走らせるために必要となるリソースには、ハードウェア(たとえばプロセッサ、メモリ、ストレージ、ネットワークリソースなど)、ソフトウェア(アプリケーション、データ、使用可能なライセンスなど)、その他の変更・設定が可能な項目(ユーザアカウントなど)がある。プロビジョニングのサイクルには、以下の作業が含まれる。

1. タスク開始の一定時間前に、使用可能なプールからリソースをスケジューリングする。
2. タスク開始時に使用する特定のリソースセットをアロケートする。
3. アロケートのあとでリソースをデプロイする。

4. デプロイされたリソースのライフサイクル、アロケーションした時からタスクが終了する時まで管理する。

5. タスクが終了したらリソースを開放する。

上記のステップのうち、CDDLMLは3と5、および4の中でデプロイメントに関連した部分を扱う。1と2はCDDLMLでは取り扱っていない。これらの用語やその他の用語については、OGSA用語集を参照のこと[1]。

#### 1. CDDLML ワーキンググループと本文書の目的

CDDLML WG は、サービスのコンフィグレーションを記述する方法、グリッドにサービスをデプロイする方法、サービスのデプロイメント・ライフサイクルの管理方法(インスタンスの作成、イニシエーション、スタート、ストップ、リスタートなど)を扱っている。WG の目的は、サービスの諸分野、アプリケーション・コンフィグレーション、デプロイメント・ライフサイクルマネジメントの研究者、開発者、プラクティショナ、理論家を動員し、同分野のコミュニティがさらに広範囲な努力を必要としている事項について探ることにある。CDDLML WG では、次の4つの仕様を与えている。

- 1) SmartFrog に基づいたコンフィグレーション記述言語仕様[2]
- 2) XML ベースのコンフィグレーション記述言語仕様[3]
- 3) コンポーネントモデル仕様(コンポーネントの表現と実装)[4]
- 4) デプロイメント API 仕様[5]

本文書の目的は以下の通りである。

- 1) 同じ課題のもとに CDDLML WG のメンバーを招集し、今後の成果物への理解と方向性を共有する。
- 2) この分野のコミュニティに WG が今後提供していくものについて提示し、コミュニティが早期にフィードバックできるようにする。
- 3) GGF の内外における他の RG/WG と連携する場を設ける。

CDDLML とプロビジョニングサイクルの関連性をさらに拡大するため、デプロイメントをプロビジョニングサイクルの一部として位置づける。プロビジョニングサイクルは、データセンタのリソースのアロケーションやコンフィグレーションを時間と共に変化するシステム環境や使用負荷に合わせて最適化するものである。プロビジョニングは、いくつかの段階から構成された概念的サイクルプロセスである。これらの段階は次のように分けられる(図1参照)。1)実行と監視、2)解析とプロジェクション、3)リソース・アロケーション計画、4)デプロイメント。プロビジョニングプロセスは、ハードウェアからアプリケーションのレイヤに至るまで、対象となるシステムのさまざまなレイヤを管理する。アプリケーション管理レイヤは、アプリケーション・プログラム、コンテンツデータ、特定のアプリケーションのDBスキーマなどといったコンポーネントを扱う。インフラストラクチャ管理レイヤは、ウェブサーバ、アプリケーションサーバ、DBMS などのミドルウェアコンポーネント、そしてオペレーティングシステムも

扱う。さらにインフラストラクチャ管理レイヤは、サーバ、ストレージ、ファイアウォール、ロードバランサ、ネットワークスイッチ等のハードウェア・コンポーネント(リソース)も扱っている。

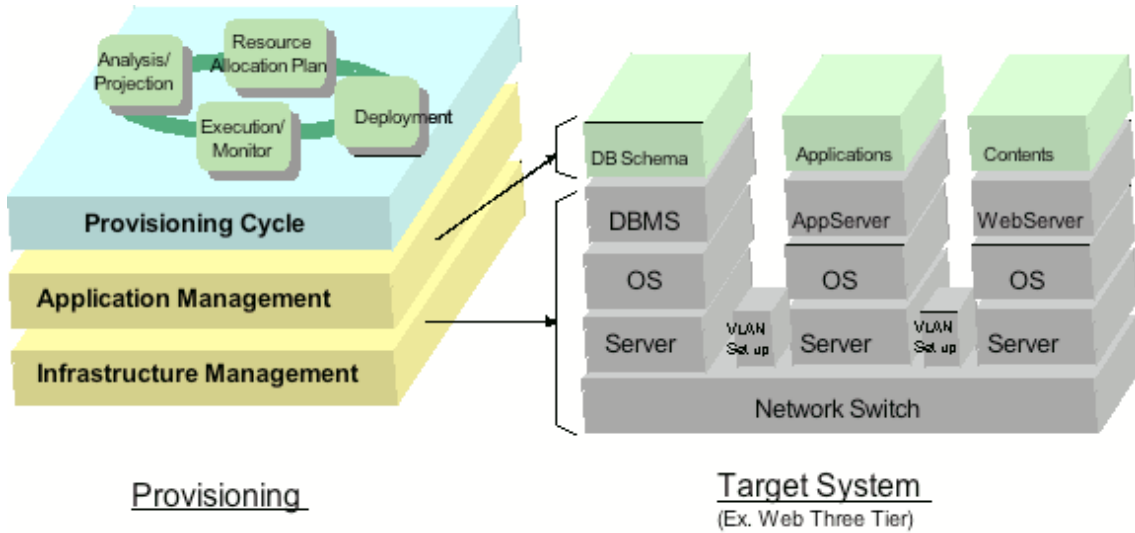


図1 プロビジョニング・レイヤ

本文書の以下の部分では、リソースからコンフィギュレーションやデプロイメントを行うソフトウェアを表すサービスコンポーネントと、サービスをデプロイするハードウェアリソースを表すサービスコンポーネントを区別して考える。

	Business Process	Application Management (Demand)	Infrastructure Management (Supply)
Resource Allocation Plan	Deployment Process	<b>Scope of CDDL M</b>	
Deployment		Lifecycle Management	
Execution /Monitor			
Analysis/ Projection			

表1 CDDL M の範囲

表1は、CDDL M の範囲を表している。そこにはアプリケーションレイヤとインフラストラクチャレイヤの両方に対するデプロイメントのフェーズが含まれている。さらに CDDL M の範囲には、対象となる環境の中で走っているデプロイされたソフトウェアについて、そのデプロイメントとライフサイクルの管理を行う基本的プロセス管理も含まれる。

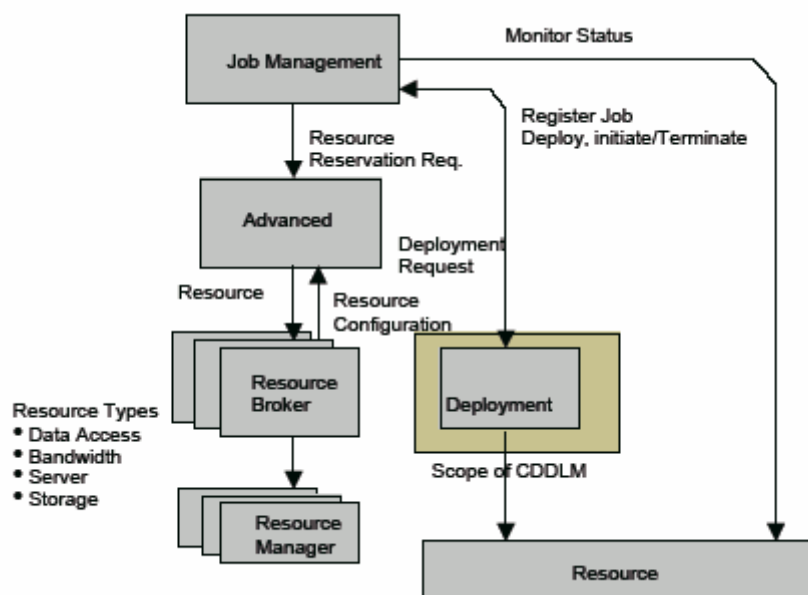


図2 プログラム実行環境での使用例

図2は、プログラム実行環境で想定される CDDL M の使用例を示したものである。デプロイメントとライフサイクル管理のサービスは、スケジューラ、あるいはマニュアル / 自動ジョブ管理によって起動される。コンフィグレーションは、コンポーネント記述セットの1つとして提供されるリソースブローカなどのサービスによって CDDL M の枠外から決められる。CDDL M では、それぞれの対象コンポーネントの管理は OASIS WSDM 管理インタフェースを通じて行われるものと想定している。

## 2 機能上の要件

### 2.1 基本機能要件

- ・サービスコンフィグレーションの記述: CDDL M の仕様により、分散したサービスのコンフィグレーションの表し方が記述される。記述には以下の事項が含まれる。
- サービスのコンポーネントを定める手段。すなわち、あるサービスを構成するにはどのような要素を組み合わせればよいのかという点を定める手段。
- 各サービスコンポーネントに関連したコンフィグレーションデータを記述する手段。
- コンフィグレーションデータの要素を互に関連づける手段。たとえば 1 カ所だけにある必要なホ

ストの名前を指定するためにこの手段を使うことができるが、記述全体を通じて整合性を持つことが前提である。

-サービスの記述を拡張あるいは修正する手段。選択したコンフィグレーションデータの要素を上書きしたり拡張することで実行する。

-サービスの記述を組み合わせる手段。たとえば、コンポーネントサービスの記述の組み合わせからコンポジット・サービスを規定する。

・サービスのデプロイメント: CDDL M では、サービスの起動のためにサービスの記述をデプロイするメカニズムを規定している。このデプロイメント・メカニズムには以下の事項が含まれる。

-どのサービスコンポーネントにデプロイメントの必要があるかを判断するため、サービスの記述を解釈する

-サービスコンポーネントを正しいリソースにロードする手段、およびそこにコンフィグレーションデータを与える

-サービスをリソースからアンデプロイできるようにする

-サービスをデプロイ、アンデプロイするために、サービスコンポーネントがデプロイメント・メカニズムに干渉できるようにする

-同じリソースセット内で同時に多くのサービスをデプロイし、管理する

・サービスライフサイクル管理: CDDL M では、サービスコンポーネントがデプロイメント・メカニズムによって管理されるよう、サービスコンポーネントが必要とする動作を指定する。これには以下の事項が含まれる。

-明確に定義されたコンポーネントのステートを使った特定のコンポーネント・ライフサイクルモデル

-ライフサイクルのステート変化に対応する特定のコンポーネント・エンリポイント

-各コンポーネントのライフサイクル・ステートを定める手段

-管理インタフェースを通じて、デプロイされたすべてのコンポーネントを見つけ出す手段

-コンポーネントを不具合からリカバリ(リスタート)させる手段

## 2.2 セキュリティ要件

・デプロイメントシステムは以下のようなセキュリティ・プロパティを提供できなければならない。

-デプロイされたサービスは、外部からの攻撃に対し、リソースにマニュアルインストールされた場合と比較して、同程度に安全でなければならない。自動デプロイメントだからといって安全性が低くなってはいけない。

-デプロイメントは、信頼の置ける対象リソースに対して信頼の置けるソースから行われなければならない。

-デプロイメントシステム内の管理通信は、互いに信頼の置けるリソースとサービスコンポーネントの間だけに限定される。

-認可されたユーザは、とくにファイアウォールの後ろから、サービスリクエストをリモートでサブミットできる。

-アプリケーションをデプロイする権利を有するからといって、そのアプリケーションやアプリケーションのデータに無条件にアクセスできるわけではない。

-アプリケーションとデータは、デリケートなものである。そこには独自のアルゴリズムやクレジットカードの情報などの重要なデータが含まれているかもしれない。したがって、その情報を対象となるホストへ取ってくる際に使用するメカニズムは、安全である必要がある。

・採用されたセキュリティのプラットフォームは、グリッドやウェブの標準的なサービスセキュリティメカニズムと整合性のあるもので、これを利用できるものでなければならない。

・CDDL M の最初の仕様では、デプロイされたサービス(あるいはデプロイされたサービスの集まり)に対しての信頼性は一階層のみである。すなわち、リソースやサービスコンポーネントは、信頼されるか信頼されないかのいずれかである。

### 2.3 リソース管理要件

CDDL M システムはリソースを直接管理することはないが、ユーザが必要としマネージャが許諾したリソースにおける論理記述・物理記述をシステムが知っておく必要がある。この情報は、アプリケーションのコンポーネントをどこでデプロイするかを決める際に使用する。現時点でデプロイメントの正確な対象を規定するには早すぎる。これは、OGSA 基本実行サービス(BES)など、関連する WG で決着がついていないためである。デプロイメントの対象は、物理的リソースであるかもしれないし、リソースのコンテナかもしれない。CDDL M WG は、デプロイメントのためのインタフェースや対象オブジェクトを厳密に決定する上で、GRAAP や BES などの他のグループと継続的に接触をしていく予定である。

・リソースの記述: サービスがデプロイされるリソースの記述

・パラメータリゼーション: メモリサイズ、プロセッサの種類や速さ、ディスクサイズなどのリソース性能を記述するパラメータ

・関連性: 社内のコンピュータなど、特定のハードウェアリソース間の関連性

・組み合わせ: より大きなエンティティを構成するようリソースを組み合わせる

・インヘリタンス: 以前のリソースの記述を継承してリソースに漸進的变化を加える

・ランタイム・バインディング: ランタイムでのみ知ることができる Web サービスやデータベースサーバのアドレスなど、コンポーネントをランタイムで結びつけられるようにする。

### 2.4 システムプロパティ要件(非機能性要件)

CDDL M、とくに基本サービスでは、以下のような非機能面での要件を遵守する必要がある。

・スケーラビリティ: CDDL M では、スケーラビリティについて制限を与えない。あるいは与えるにして

も最小限にとどめる。スケーラビリティは、基礎にあるネットワーク・インフラストラクチャとそれがサポートするトランスポート・プロトコルによって制約を受けるものだからである。唯一の例外が、ランタイム・バインディングである。ランタイム・バインディングは、レイジーな依存関係を分解するためにデプロイメントを遅くすることがある。

- ・高アベイラビリティ: CDDLM は、デプロイメントのリクエストに対して高いアベイラビリティを保持しなければならない。とくに、リクエストの負荷や不具合が高い場合でも CDDLM はこれに対応しなければならない。

- ・自己修正: CDDLM は、不具合が生じた際にサービスを修正できなければならない。たとえば、一部分が機能しなくなったり、他の複製された CDDLM の基本サービスにリクエストをリダイレクトした場合に、CDDLM が自らのインフラストラクチャを再起動する必要がある。

- ・障害復旧: CDDLM は、大きな障害が発生した場合でも機能を継続し復旧することができなければならない。すなわち最低限でも、デプロイメントサービスの重要な部分に不具合が生じた場合、サービスを複製するサポートや、復旧のためのおそらくより複雑なサポートを提供する必要がある。

- ・完全自動化: CDDLM は、求められない限りはユーザ入力が必要としない完全自動化を行う。とくに、他のサービスをデプロイしたり、不具合からの復旧やプロビジョニングのリクエストに応える作業などを自動で行うことができなければならない。CDDLM の基本サービスは、短期的リクエストから長期的実行までさまざまな時間レベルでこれらを遂行する必要がある。また、自動化されたビルドプロセスの間などに、デプロイメントがプログラムによって起動されることが可能でなければならない。

- ・使用性: システムは、エンドユーザがシステムの詳しい知識なしに使う上で、十分シンプルである必要がある。言語と管理のプロセスは、どちらも学習したり使用したりする上で比較的簡単なものでなければならない。

## 2.5 コンフィグレーション記述言語 (CDL) の要件

宣言型記述: コンフィグレーションの記述は、宣言型である必要がある。この記述は、一連のオペレーションではなく、リソース間の依存関係を記述する一組の宣言でなければならない。この宣言型記述は、デプロイメントやライフサイクルマネジメントのため、CDDLM の実装が分散したリソースを横断して作業の正しい順序立てを動的に行うことができるよう、十分な情報を提供する必要がある。

XML ベース: この言語は XML ベースでなければならない。よくできた記述は、よくできた XML ドキュメントであるべきである。また、通常のプラットフォームで広く使用されている、Web サービスに適した XML のサブセットである必要がある。とくに、XML スキーマ (WXS) は、CDL 言語の構文を記述するのに適している。ただし、そのフォーマット (ファセットやアンサインドロングなど) の問題箇所は、可能であれば避けなければならない。

XML フォーマットは、ネームスペースを知っている必要がある。各バージョンの CDL では、要素や属性のためのそれ自身のネームスペースを定義する。CDL の実装では、ネストしたデータを適

切な他のネームスペースで処理することを想定する必要がある。この言語はまた、安全なコンテキストで解釈されるよう、設計する必要がある。安全なコンテキストでは、エントリの拡大やスキーマやファイルをリモートで取り込むことが可能であるが、それは管理された方法で行われなければならない。

モジュール方式: CDDL M がデプロイするサービスは、複数のサブシステムから構成されていることがある。CDL は、モジュラー方式で各サブシステムを表現できなければならない。

記述には以下の点が含まれる必要がある。

- ・各サブシステムのコンフィグレーション(コンフィグレーション・パラメータの定義)
- ・(コンフィグレーション・パラメータやライフサイクルのステータスといった点での)サブシステム間の依存関係

動的コンフィグレーション: CDL は動的なコンフィグレーションの以下のような使用例に対して適用できなければならない。

- ・いくつかのコンフィグレーション・パラメータ値は、デプロイメント・タイムの前に決定することはできない。
- ・コンフィグレーションはランタイムで変わる可能性がある。
- ・環境に変化が生じる。(たとえばシステムの不具合)
- ・CDDL M の実装に変化がある。

整合性: CDL は、CDDL M の実装がパラメータ間の整合性を管理できるよう、コンフィグレーション・パラメータ値の間の依存関係を定めることができなければならない。

CDL は、依存関係の記述について次の点をサポートしている。

- ・他のコンフィグレーション・パラメータから導き出した値のアサイメント
  - ・値が満足すべきインテグリティ制約(アサーション)
- (注意: これは値や変化のプロパゲーションが一方向であることを意味する)
- パラメータ値は次の時にアサインされ、検証される。
- ・定義の時間(バインディング時間)
  - ・デプロイメント・タイム

インテグリティ: CDDL M の実装は、複数のコンポーネントに対し、ライフサイクルのステータス(インスタンスを作成した、初期化した、作動中など)のインテグリティを管理する必要がある。CDDL M の実装がサービスをデプロイ、開始、復旧、停止、削除するための一連の作業を正確に生成するよう、CDL が保証しなければならない。

CDL は以下のインテグリティの機能をサポートする。

- ・コンポーネントの状態を変える前に、必要なコンフィグレーションが正しく行われたことを CDDL M の実装が確認できる。(たとえばコンポーネント A は B が初期化される前に作動している必要がある、など)
- ・CDDL M の実装が、デプロイメント作業の際にサービスをデプロイできないことを検知した場合、その作業を安全にキャンセルできる。(たとえばすでにデプロイしたコンポーネントを削除する、

など)

- ・コンポーネントの状態が環境によって変わり(たとえばコンポーネントがシステムの不具合により役に立たなくなった場合など)、その結果インテグリティが損なわれた場合、CDDL M の実装はインテグリティを復活させる一連の作業を生成することができる。(たとえばコンポーネントを正しい順番でリスタートさせる、など)

コンポーザビリティ: サービスコンフィグレーションの記述全体は、複数のコンフィグレーション・ベンダが提供する複数の記述を組み合わせて構成する必要がある。CDL では、既存の記述を参照することで新しく組み合わせた記述を定義する方法が与えられていなければならない。たとえばユーザは、コンフィグレーション・パラメータおよび(または)コンフィグレーションの依存関係を追加したり無効にしたりするなどして、システムベンダが提供したコンフィグレーション記述をカスタマイズすることができる。

発見(参照およびバインディング): CDL では以下の点がサポートされている必要がある。

- ・外部のコンフィグレーション記述の参照
- ・コンポーネント(デプロイされるパッケージ)の参照・コンポーネントのデプロイメント機能を提供するサービスの参照

セキュリティ: CDDL M のセキュリティに関する要件は、Web サービス / グリッド / XML のセキュリティ標準を CDL に組み込むことで達成すべきである。

拡張性: CDL は、ユーザが拡張性の要素を記述に取り入れることができるようにする必要がある。たとえば、ユーザがセキュリティやポリシーの記述を追加する必要があるが出てくることもあるはずである。

## 2.6 サービスインタフェース要件

CDDL M サービスに対する通信は、Web サービス SOAP インタフェースを通じて行われる。このサービスインタフェースの要件を以下に列記する。

- ・ファイアウォールを通じたりリモートアクセスをサポート
- ・グリッドや Web サービスのセキュリティモデルとの統合
- ・メッセージが盗聴されたり、いたずらされたり、再生されることがあってはならない。また、よく知られた外部攻撃に対して強くなければならない。
- ・メッセージは幂等である必要がある。
- ・ステータス情報は機械が読める形式で提供されなければならない。
- ・デプロイやアンデプロイの作業がサポートされていなければならない。アンデプロイは、データの残ったリソースをきれいにするものである。
- ・インタフェースは拡張性が必要(言語、おそらくはアクション)
- ・サービスが他のサービスを通じてデプロイするための十分な機能(連鎖法)
- ・ライフサイクルのイベント / ステータスの変化通知がプロバゲートする必要がある。
- ・ファイルのリモートシステムへのアップロードがサポートされている必要がある。

### 3 使用例

使用例に関しては、OGSA の使用例[6]に基づいている。

#### 3.1 グリッド環境における Web サービス

##### 3.1.1 概要

Web サービスを使えば、グリッド環境内のいかなるリソースでも表示し、仮想化することができる。リソースとは、実際に動いているアプリケーション・プログラム、ミドルウェア、オペレーティングシステム、さらにはサーバ、ストレージ、ファイアウォール、ロードバランサ、ハブスイッチなどのハードウェアを指す。

アプリケーション・デプロイメントのコンテキストでは、デプロイされるアプリケーションやサーバのコンフィギュレーションを定義するなど、いくつかのアクションがデプロイメントに含まれる。デプロイメントでは、デプロイされるすべてのコンテンツを集め、それを適切な順序でホスティング環境に送り、そしてミドルウェアやアプリケーションをイニシエートする。

プロビジョニングのコンテキストでは、システムコンフィギュレーションを定義するなど、アプリケーション・デプロイメント上でのアクションがデプロイメントに含まれる。ファイアウォールやその他のハードウェアの設定、決められた設定すべてを適切な順番でダウンロードしセットアップすること、設定の施行など、これらを目的としたネットワーク構造やアクセスポリシーもこのシステムコンフィギュレーションの 1 つである。

##### 3.1.2 ユーザ

Web サービスは抽象化されたモデルであるため、グリッドコンピューティングの幅広いユーザ全体を考慮する必要がある。本節では、大規模アプリケーション開発と品質保証(QA)の両方のシナリオを考えることにする。この場合ユーザは、アプリケーション開発エンジニアあるいは QA エンジニアのいずれかである。データセンタのオペレータなどは、データセンタ・プロビジョニングのシナリオに登場する別のタイプのユーザである。これについては次の節で検討する。

アプリケーション開発の際や大規模システム・シナリオの QA の間は、エンジニアは、ハードウェア、オペレーティングシステム、ミドルウェア、ネットワークを適切に設定するために、アプリケーション・プログラムを繰り返しアップデートし、ダウンロードする。エンジニアはまた、多くの異なるコンフィギュレーションでアプリケーションを吟味するため、将来の使用に予想される数多くの異なる環境において正確な実行を保証することができる。

通常のアプリケーション開発部署や QA 部署では、エンジニアは長期間にわたって多くのアプリケーションを扱わなければならない。他のシステムでこれまで使っていた定義を再利用できるよう、あらゆるコンフィギュレーションを定義する上で、整合性のとれた再利用可能な方法が必要となる。また、対象となるアプリケーション特有の違いだけを定義する必要がある。いくつかの共通点のある多くの異なるコンフィギュレーションを調べる際にも、この方法は有用である。

##### 3.1.3 シナリオ

#### 1) コンフィグレーションの定義

エンジニアは、GUI やテキスト形式の定義言語のツールで、アプリケーションやシステムのコンフィグレーションを最初に定義する。過去に行われた定義や別の場所での定義を参照することもできる。また、既存の定義を拡張、修正したり、そこから新しい定義を導出したりすることもできる。このことは生産性を高め、定義に混入しうる新たなエラーを低減することにつながる。エンジニアはまた、システムをいかに適切に設定し、アプリケーションをいかに転送しイニシエートするかといった手順についても、適切に定義しなければならない。

#### 2) アプリケーション・プログラムを書く、修正する

これは通常のソフトウェア開発に近い。エンジニアは、使い慣れた開発環境でいかなるアプリケーションでも書くことができ、仮想的にコンパイルすることもできる。

#### 3) システムコンフィグレーションのセットアップ

エンジニアが最初のコーディングを終えると、ホスティング環境をハードウェア、オペレーティングシステム、ミドルウェアに合わせて適切に設定する必要がある。このコンフィグレーション・セットアップの全プロセスは、エンジニアがデプロイメントを開始した際に自動的に行われる。

#### 4) プログラム転送とテスト

システムが適切にセットアップされた後は、自動的にアプリケーション・プログラムの転送をシステムが開始する。プロセス定義に従って、システムはソフトウェアコンポーネントの転送と起動を計画する(順序付け作業も含む)。

エンジニアは、転送する時間を節約するため、次の段階で使うことのない不要なコンテンツや変更を行わないコンテンツを転送せずにコンテンツの部分転送を行うことができる。

#### 5) 他のコンフィグレーションのテスト

エンジニアは、さまざまな理由でコンフィグレーションを変更したい場合がある。デバッグの際にコンフィグレーションに誤りを見つけ、修正する必要があることもある。また、異なるコンフィグレーションによるアロケーションを検討したいと考える場合もある。そのためシステムは、新しいコンフィグレーションを指定することで、適切な順序に並べられたアクションにより、自動的にコンフィグレーションを変更することが可能である。

#### 6) 作業サポートとメンテナンス

コンフィグレーション定義は標準に準拠しているため、他社の環境にも適合するべきものである。したがって顧客や作業サポート会社、メンテナンス会社などに、アプリケーションコードと一緒にコンフィグレーション定義を受け渡すことができる。これによりそのアプリケーションは、シームレスにこれらの会社に転送され、サポート、メンテナンスを受けることができるのである。

### 3.2 商用データセンタ

#### 3.2.1 概要

今日の商用データセンタは、さまざまなタイプのホスティングサービスを提供している。それらは、共有、コロケーション、管理インフラストラクチャ、アプリケーションホスティングの4種類に分類できる。

- ・共有サービスは、低コストの初心者向けウェブホスティングサービスを提供するサービスである。各ユーザウェブサイトには別々のコンピュータを使用する代わりに、数十ものサイトを同じ物理的サーバに同居させる。
- ・コロケーションサービスでは、ユーザはサービスプロバイダからラック、ケージ、キャビネットを借り受け、データセンタに自分のサーバを設置することができる。そして自前の技術者を送り込み、メンテナンス作業や不具合に対処させることができる。
- ・管理インフラストラクチャサービスは、ウェブサーバ、データベースサーバ、アプリケーションサーバ、サーバ・コンフィグレーション、システムオペレーション、セキュリティ、監視、レポート、ネットワークなどの、専用ホスティングサービスである。
- ・アプリケーションホスティングサービスは、アプリケーションホスティング環境にある。この環境では、データセンタは専用のサーバやアプリケーションを使い、アプリケーションや基盤となるインフラストラクチャの日々のオペレーションやメンテナンスをすべて引き受ける。

ソフトウェアのデプロイメント、コンテンツ、システムコンフィグレーションのセットアップは、管理インフラストラクチャサービスやアプリケーションホスティングサービスで重要な役割を演じる。データセンタは、ユーザのリクエストに応じて、ハードウェアとソフトウェアの両方に適切なシステムコンフィグレーションを提供する。ハードウェアコンフィグレーションには、ロードバランサ、ファイアウォール、VLANスイッチなどのコンフィグレーションが含まれ、これによりユーザは1つの安全な独立したネットワークパーティションを持つことができる。このネットワークパーティションは、データセンタの外部に対して、適切に保護された状態で接続されているものである。ソフトウェアコンフィグレーションには、オペレーティングシステムやミドルウェアなどが含まれる。さらに、アプリケーションホスティングサービスの場合は、アプリケーションソフトウェアも含まれる。

自動デプロイメントは、少なくとも次の3つの点においてデータセンタのビジネスにとって重要となる。今日のデータセンタでは、ユーザのリクエストを受けてからデプロイメントを完了し、機器とソフトウェアのすべてをセットアップするまでかなりの時間(たいていの場合数週間)かかる。この準備期間は、リソースの利用に直接影響が及ぶため、ユーザにとってもデータセンタにとっても重要である。デプロイメントの正確性もまた、データセンタビジネスにとって大切な点である。今日の典型的なデータセンタにおけるデプロイメントプロセスは、多くのマニュアル作業を必要とする。この作業にはサービスの誤設定につながりかねないミスを犯す危険性が多く含まれている。データセンタは、新しいコンフィグレーションをテストするために十分な時間を割く必要がある。セキュリティもこのビジネスに重要な問題である。ソフトウェアとデータは、データセンタの外部から保護されなければならないだけでなく、場合によってはデータセンタのオペレータからも保護される必要がある。これは、データセンタのユーザに対して秘密にすべき情報が含まれているからである。

### 3.2.2 ユーザ

データセンタのオペレータは、2つの側面からデプロイメントを取り扱う。オペレータが新しいシステムコンフィグレーションのリクエストを受け取ると、そのユーザのリクエストをデプロイメントサービスの入力となるコンフィグレーション定義に読み替える。デプロイメント・コンポーネントの数は、デプロイメントの範囲によって、数十から数千になる。準備が整うと、ソフトウェア、データ、ハードウェアのそれぞれのコンフィグレーションが対象となる環境にデプロイされる。これはオペレータがマニュアルで行うことも、管理システムが自動で行うこともある。

### 3.2.3 シナリオ

#### 1) コンフィグレーション定義

システムコンフィグレーションの要件について、ユーザがデータセンタのエンジニアに話を持ちかけたとしよう。その際、性能、信頼性、アベイラビリティの要件は、サーバハードウェア、データストレージ、サーバ間帯域幅などの種別、そしてデータセンタ外へのチャネルなどといった特性のシステムコンフィグレーションに読み替えられる。これはさらに、二重サーバや RAID ストレージなどの高い信頼性を目的とした特定のコンフィグレーションに対する要請や、高性能を目的としたサーバアレイに対する要請などにつながる。

ソフトウェア要件には、アプリケーションが置かれるウェブサーバ、アプリケーションサーバ、DBMS などのオペレーティングシステムやミドルウェアの種別やバージョンも含まれる。ユーザはまた、ウェブサーバ、DBMS の初期データコンテンツも指定してくる。そのコンテンツ情報には、ミドルウェアのための初期セットアップデータやユーザレジストリなども含まれている。コンテンツの一部がデータセンタのオペレータに対して機密である場合もある。

これらの要件は、データセンタのエンジニアが GUI やテキスト形式の定義言語を使って、一組のコンフィグレーション定義に読み替えられる。

近い将来、完全に自動化されたユーティリティコンピューティングが実現する前に、ユーザは特定のアプリケーションのためのコンフィグレーションを複数種類用意するよう求めることがあるだろう。これによりユーザは、たとえば生産とシステムテストの間で切り替えを行うなど、コンフィグレーション間の切り替えができるようになるからである。

#### 2) デプロイメント

データセンタのオペレータは、コンフィグレーションの定義に基づいて、そのコンフィグレーションをセットアップし、設定されたハードウェア上のソフトウェアやコンテンツをデプロイする。オペレーティングシステムやミドルウェアなど、ソフトウェアの一部は設定済みのサーバにすでにインストールされていて、インストールの手間がかからない場合もある。

オペレータは、デプロイメントに使用できる必要なハードウェアをアロケートする必要がある。これはオペレータがマニュアルで行うこともできるが、管理システムが自動で行うこともできる。

オペレータは、コンフィグレーションの定義に基づいて、ハードウェアをセットアップし、そのハードウェア上の必要なソフトウェアすべてをデプロイする。この作業は完全に自動化されているものと

想定されている。ユーザが保有するすべてのアプリケーションソフトウェアとコンテンツは、デプロイメントの前に、データセンタやデータセンタからアクセスできる場所に転送する必要がある。デプロイメントはオペレータが起動することができる一方、自動で起動することもできる。

### 3.2.4 関連リソース

デプロイメントに関与するリソースの種類は、データセンタが提供するサービスの種類によって変わる。最低限のシナリオでは、デプロイメントシステムはデプロイされたアプリケーションソフトウェアのみを扱う。この種のシナリオでは、デプロイメントシステムは、アプリケーションソフトウェアのバージョン、そしてそのソフトウェアが必要とする初期パラメータやデータを管理する。ミディアムレンジのシナリオでは、ウェブサーバ、アプリケーションサーバ、DBMS など、さまざまな種類のインフラストラクチャソフトウェアのデプロイメントが含まれている。こうしたシナリオの多くは、ソフトウェアとともに初期データやユーザレジストリのデプロイメントも必要とする。フルレンジのシナリオでは、サーバ、ストレージ、ロードバランサ、ファイアウォール、ネットワークスイッチなどのさまざまなハードウェアのセットアップも含まれる。

## 3.3 IT インフラストラクチャと管理

### 3.3.1 概要

大企業の中では通常、オペレーショングループとITグループが独立して設置されており、これらはエンドユーザの計算リソースに対するデータセンタの作業を管理する仕事を別々に任されている。エンドユーザの計算リソースには、各部署のサーバやワークグループのサーバも含まれる。中小企業では、これらのサービスは1つの部署に統合されている場合が多い。ここで扱う使用例では、IT領域に重点を置き、さらに場合によっては、上述のデータセンタでの使用例のように、データセンタが提供するサービスへのより広範囲なインタフェースにも焦点を当てる。

現在幅広く用いられているケースでは、通常次の分野のIT問題を扱う際にシステム管理ソフトウェアが使用されている：コンフィグレーション管理、ソフトウェア・ディストリビューション、アセット管理、ライセンス管理、ヘルプデスク。CDDLIM では、おもにコンフィグレーション管理とソフトウェア・ディストリビューションに重きを置くことになるが、残りの機能の多くについてもグリッドのどこか別の場所で使用できるようになるはずである。

通常の使用パターンでは、エンドユーザはある代表的なサービスに接続する必要がある。ユーザはまず、サービスとのやりとりのために正しいソフトウェアにアクセスできなければならない。これはすなわち、エンドユーザのワークステーションに対して、アプライオリに、あるいは特定のサービスが使用された際に、従属コンポーネントをデプロイする必要があるということを意味する。また、このサービスはいくつかのサービスコンフィグレーションの1つであるかもしれない。サービスが、ある特定のサーバやサーバ群、メインフレーム、スーパーコンピュータに格納されていることもある。さらに、あるワークグループの使用できるワークステーション群で走らせることもある。これらのいずれのシナリオにおいても、正しいソフトウェアのデプロイメントやコンフィグレーションを行うこと、そしてサービ

スの最中での管理を確実に行うことが重要である。これらのインスタンスは、複数のジョブが複数の計算クラスタに渡って LAN や WAN を張り巡らしている中で、グリッドのより広いコンテキストの範囲内で管理することが必要である。

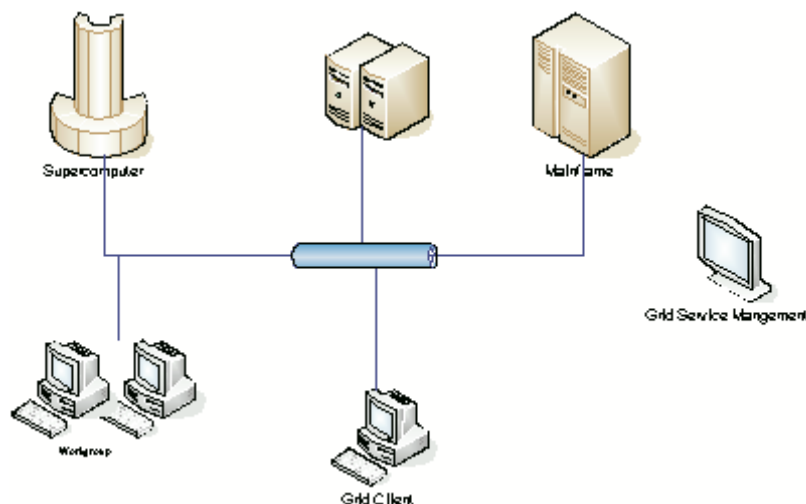


図 3 IT インフラストラクチャと管理

IT の役割はエンドユーザのニーズに確実に応えることである。ユーザからは責任をもってデプロイメントの作業や、そのデプロイメントをデータセンタのオペレーション部門のスタッフと調整することが求められる。デプロイメントがデータセンタでおもに行われるとしても、IT はコンフィグレーション、デプロイメント、イベントの情報にアクセスする必要がある。

### 3.3.2 ユーザ

この使用例での最終的な顧客は、物理的サービスそのものを消費する企業の知識労働者である。企業の IT スタッフとオペレーションスタッフも参加者の一員である。多くの企業では、ソフトウェア開発チームが顧客のためのサービスのデプロイメントやコンフィグレーションで中心的な役割を果たしている。

知識労働者は、業界特有の仕事を行うためにグリッド上の抽象サービスと毎日やりとりを行う。これらのサービスは、処理を行うコマンドをバッチ指向であるいはリアルタイムでサブミットするものである。ユーザは物理的実行環境をこれらのサービスのために作るよう要請し、また、ソフトウェアをコンフィギュア、デプロイするよう求める。実行の間、エンドユーザは実行仮定を監視し、最終処理まで見守る。

多くの企業の体制では、こうした利用のシナリオは、たいていの場合 1 つの物理的サイト内で済まされることになる。しかし、中央リポジトリあるいはリモートリポジトリへのアクセスは、データ、あるいはメインフレームや専用計算クラスタなどの特別な計算リソースにアクセスする上で必要になることが多い。このシナリオに参加しているユーザの数は非常に多く、たいていの場合 1 つのオフィス施設または 1 つのサイトあたり数千人にものぼり、しかも世界にはそうしたサイトが数十、数百とある。さ

らにそこには、大きな計算施設のある中央データセンタサイトが1つあるいは複数設置されていることが普通である。

### 3.3.3 シナリオ

#### 1) 一般サービス

世界的な投資銀行のある為替トレーダが、毎日変動する為替を評価する新しいモデルを考えていた。彼はこのモデルについてエンジニアリングチームと協議し、サービスを開発するよう依頼した。そのサービスが設計され、その要件はオペレーションチームによって検討された。そのサービスはデータセンタに置かれ、世界の為替価格のこれまでのデータベースにアクセスする専用サーバ群に配置することにした。

トレーダは、サービスが使えるようになるのを待っていた。エンジニアリングチームは、ウェブベースのユーザインタフェースを作り、トレーダが異なるコンフィグレーションでサービスのインスタンスを作成できるようにした。さらにトレーダには、さまざまな種類のクエリをいろいろなパラメータや形式でサブミットできる画面も用意された。しかしトレーダの個人的 PC には一般的な使用目的のウェブブラウザ以外に必要なものはなかった。

#### 2) エンドユーザ・デバイスデプロイメント

その為替トレーダは、モデルの性能に満足し、サービスを走らせて得られた結果を解析するオンライン解析処理(OLAP)を行う必要があると判断した。エンジニアリングチームは、そのサービスをフロントエンドするための Visual Basic アプリケーションを構築することを提案した。銀行からは変動する為替を表示する画面が送られてくるため、彼はそのサービスに常時アクセスできるようにしている必要があった。

サービス開発の一環として、彼はエンジニアリングチームに対し、彼が使っているワークステーションに必要なに応じてデプロイできる1つのサービスとしてその解析アプリケーションをモデル化し、コンフィギュアするよう依頼した。こうして、彼がそのサービスをウェブから起動すると、彼のワークステーションは OLAP アプリケーションにあわせて設定されるようになった。

#### 3) ハイエンド P2P 計算サービス

その為替トレーダは、自分のモデルが長期にわたる取引には有効だが、毎日の取引活動には向いていないと判断した。そこで彼はモデルを修正することにした。この修正では、過去の履歴データを必要としない代わりに多くの同時計算パワーを必要とすることになる。彼は上司と話し合った。上司は、彼が取引を行っている間は使われることのないその部署の余分なワークステーションを使うよう提案した。

エンジニアリングチームは、一度に複数のマシンにデプロイできるようサービスを修正する方法について議論した。エンジニアリングチームはその答をオペレーションチームに伝え、さらにオペレーションチームは IT スタッフにその要請を送った。IT スタッフとオペレーションチームは、チームの

ワークステーションをいかにしてデータセンタのリソースマネージャの管理下に置くかを話し合った。あるいは別のリソースプールを構築した方がやりやすいかどうかを議論した。

チームはローカルのリソースマネージャを利用することに決めた。リソースマネージャは、為替取引チームのワークステーションから得たリソースだけに特化したものである。トレーダがシミュレーションを行うと、リソースマネージャが使用できるワークステーションあるいはその一部をアロケートし、サービスのデプロイメントと起動を行う。

### 3.3.4 関連リソース

これらのシナリオでは、グリッドシステムは非常に多くのさまざまなリソースを管理しなければならない。データセンタではいくつかの異なるハードウェアシステムがストレージ、クエリ、計算のために使用されている。しかしシステムはおもに、サービスコンポーネントの正しいコンフィグレーションを確実に行うことに責任がある。アベイラビリティ、および(あるいは)ストレージシステムなどの他のコンポーネントへのコネクティビティをシステムが保証する必要があるものの、過去の市場データが安定したストレージ施設に保管されているということが前提になっていることが多い。

Web サービスのデプロイメントと管理は、そのライフサイクルとエンドユーザのソフトウェアコンポーネントを通じて行う必要がある。そしてそのソフトウェアコンポーネントは、ユーザの PC に影響を持つシナリオにあわせて管理されているものである。ピアツーピア・システムの場合、ピア・ワークステーション全体が、アプリケーションソフトウェアと Web サービスライフタイムを通じて、オペレーティングシステムから管理されなければならない。

その性格上、コンポーネントは高度に分散している。さまざまなネットワーク・セグメントに置かれていることが多く、たいていの場合地理的に分散している。

### 3.3.5 使用例の状況解析

リソース管理サービスなど、これにさらに上積みするバックグラウンドサービスを現在まで数多く検討してきた。しかし、これまで実際に何がどの程度実装されてきたかは明らかではない。

## 3.4 ユーティリティコンピューティングモデル

### 3.4.1 概要

最近ユーティリティコンピューティングがかなり認知されるようになってきた。ユーティリティコンピューティングは、データセンタをうまく利用(整理統合)することで計算リソースに一度接続すれば何度でもプロビジョニングでき、ユーティリティのように計算リソースを扱うことのできるモデルである。おもな利点としては、コストを低減できることがあげられる。同じ数の IT を使い、同じ物理的スペース、同じパワー消費にも関わらず、より多くのシステムをサポートできるからである。このモデルでは、自動的にすばやくシステムインストール(再パーティショニング、プロビジョニング、アプリケーションやサービスのインストール、管理など)ができることが重要である。

ユーティリティコンピューティングをデプロイメントに関係づける 1 つの方法は、PlanetLab、仮想

化マシン、一般のグリッドのような他のアーキテクチャのニーズと同類のものである。これらのどのアーキテクチャでも、リソースのサブセットを獲得すること、必要なソフトウェアをデプロイすること、需要が変化した場合にリソースのプロビジョニングを継続すること、などのニーズがある。図 4 にユーティリティコンピューティングモデルが示されている。

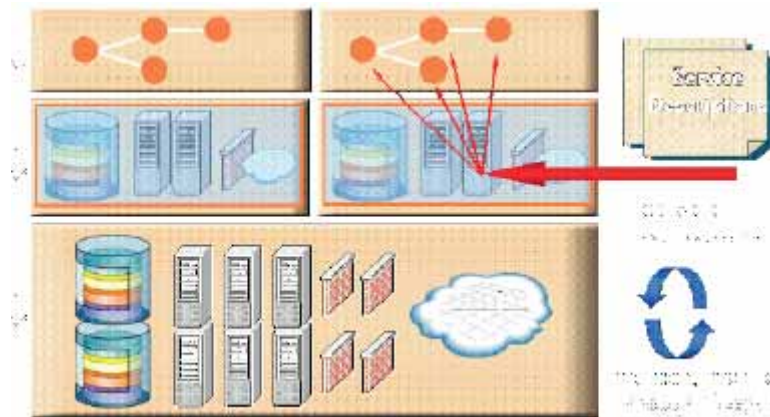


図 4 ユーティリティコンピューティングモデル: このモデルは、仮想化に基づいてデータセンターのリソースの整理統合を可能にするものである。自動的なコンフィグレーション、デプロイメント、ライフサイクルマネジメントが重要な役割を果たす。

### 3.4.2 ユーザ

ユーティリティコンピューティングモデルにおける CDDLML の利用者は、さまざまな技術力や管理能力を持った次のような参加者から構成されている。

**ユーティリティ管理者:** ユーティリティ管理者はおそらくもっとも技術力のある利用者であり、ユーティリティや CDDLML の使用をもっとも左右する存在である。ユーティリティの所有者は、ユーティリティの部分を実パーティショニングし、別々のサービスプロバイダに再アロケートすることができる。彼らはまた、これらのパーティションを何人かのユーザが使用した後に削除し、そこにデフォルトの形、あるいはカスタマイズした形でインストールを行うことができる。さらにユーティリティの所有者は CDDLML を使い、管理目的でユーティリティ固有のサービスを運用することができる。

**サービスプロバイダ (SP):** SP は、パーティション内でサービスを提供するためにユーティリティの一部を貸し出したり所有したりすることができる。SP は、アロケートされたパーティション内で、ユーザに提供するのに必要なサービスをインストールする。負荷が変動するに従い、SP は、必要なサービスをインストールしデプロイするリソースの貸し出しを、増やしたり減らしたりすることができる。

**サービスユーザ:** サービスユーザは (SP やユーティリティ所有者と比べて) CDDLML に最小限のアクセスしかできない。そして通常は、追加コンポーネントの要請に応じてサービスを拡張したり、コンフィグレーション・パラメータを変更することができる。この機能のほとんどは、SP が有効にしてサービスユーザが起動するものである。

### 3.4.3 シナリオ

### 1) ユーティリティデータセンタの動的プロビジョニングのサポート

あるサービスプロバイダは、提供するサービスが人気を集め負荷が上昇していることを目の当たりにした。幸い、SP が計算ユーティリティの所有者と話し合い、必要に応じてキャパシティを拡大する合意を取り付けていた。負荷が限界値を越えると、追加するサーバを使用可能なリソースプールに要求する。そして数分以内に必要なシステムとアプリケーションソフトウェアとともにサーバがインストールされる。しばらくしてこのサービスへの関心が低下し、負荷が限界値を下回ることとなり、余分なリソースが使われなくなる。CDDLML を使うことで、余分なリソースはリリースされ、数分のうちに消し去られる。

このシナリオでは、CDDLML はリソースのアロケーションをする上で他のプロビジョニングツールの力を借りるが、サービスの必要に応じたコンフィギュアやデプロイ、アンデプロイは CDDLML が対応する。

### 2) 複数のユーティリティデータセンタを横断する動的プロビジョニングのサポート

あるサービスプロバイダが、世界中でサービスのインストールを行った。このサービスの人気が高まり、世界中で金額が変動するようになると、彼はあまり使われていないサイトに負荷を移動させることでお金を節約できることに気づいた。ヨーロッパでの負荷が高まり、彼は平均的なユーザの応答時間をあまり低下させることなく、いくつかのサービスをアメリカにデプロイした。SLA を遵守せず、通常、貴重なリソースをローカルにオンデマンドでアロケーションするリクエストだけがリモートサイトに移され、そこでは同じサービスがオンデマンドでデプロイされる。CDDLML を使ってリソースはデプロイされ、サービスプロバイダの利益を最優先してオンデマンドに応じてあちらこちらでリリースされる。

このシナリオでは、CDDLML はリソースのアロケーションをする上で他のグリッドプロビジョニングツールの力を借りるが、サービスの必要に応じたコンフィギュアやデプロイ、アンデプロイは CDDLML が対応する。

### 3) サービスのデプロイメント

ある企業が、複雑で大きなサービスをデプロイしている。このサービスは、数多くの外部サービスに依存したものである。個々のサービスは信頼の置けるものであるが、新しくデプロイしたサービスの信頼性は、当初はあまり高い可能性がある。(同じシナリオは、最初のサービスデプロイメントの際にも考えることができるが、そのコードはまだ安定したものではない。) ユーティリティはベータテストのコンフィグレーション用にパーティショニングされ、そこで実行されるサービスにはとくに注意が払われる。これらのサービスは厳重な監視が行われ、いずれかのサービスに不具合が生じた場合には、そのサービスに依存する別のサービスとともに再度デプロイする。それらのサービスが安定しても監視は続けられるが、ハートビート、複製、その他の信頼性に対するサポートは回数を削られる。

このシナリオでは、ライフサイクルマネジメントが予定していたイベントだけでなく予想外の不具合

にも責任を負う。その場合サービスは、不具合が生じるとあらゆる依存関係を確実に保ったまま再度デプロイされる。そして不具合の生じたサービスに依存したサービスも再起動される。

#### 3.4.4 関連リソース

このシナリオでは、CDDLML はユーティリティコンピューティングリソースをまとめて管理する必要がある。この種のリソースにグリッド標準を使うおもな利点は、ユーティリティ間のサブミッションにある。これによりユーティリティを横断してサービスをデプロイできる。その場合、分散したユーティリティは地理的に離れ離れになっており、別の大陸に存在することもありうる。

### 3.5 複雑なビジネスサービスの使用例

#### 3.5.1 概要

複雑なビジネス・コラボレーション(このグリッド使用例の範囲内の話だが)は、複数のグリッド・タイプ、複数のグリッド・ユーザ、および複数の利用モデル(バッチおよびリアルタイム)を組み合わせたものである。このコラボレーションの管理と調整には、分散したリソースセットを束ねるだけでなく、いろいろな抽象化を必要とする問題がある。この抽象化は、一部のユーザにとってはハイレベルなものであり、別のユーザにとっては複雑で詳細なものである。この使用例の要点は、意味のあるビジネスタスクを行うために既存のサービスを動的にプロビジョニングすることにある。この既存のサービスは、ソフトウェア(パッケージソフトウェアや社内ソフトウェア)、データ、ハードウェアを組み合わせる。動的なプロビジョニングは、多くの設定(国ごと、地域ごと、あるいは集中型)において繰り返し行われることが普通であり、標準化のための共通コンフィグレーションを保証するものである(そして既存のワークから構築できるコンフィグレーションモデルを推進するものである)。

このデプロイメントは、従来のハードウェアやインフラストラクチャソフトウェアのデプロイメントから抜け出し、既存のリソースの動的デプロイメントによる動的なシステムプロビジョンのモデルを目指すものである。

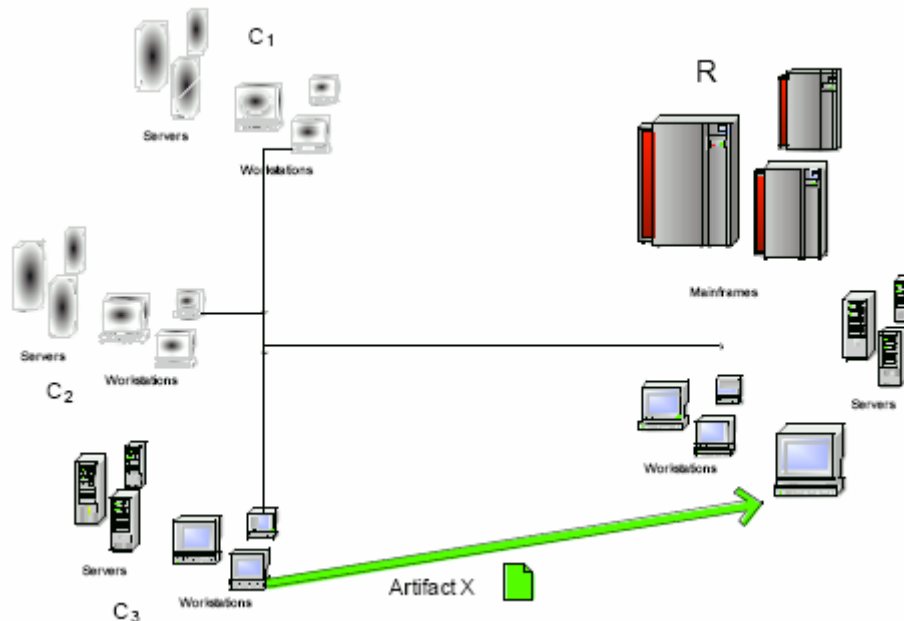


図5 複雑なビジネスサービスの概略図

この図では、互いに関連する4つのユーザグループを1つの大きな組織で表現している。ある国のグループC3が人、アプリケーション、ハードウェアのリソースを組み合わせ使用してプロセスを実行し、あるタスクを行っているとする。C3に依存しているある地域(複数の国にまたがる地域)のグループRが、国のタスクが終了した時点で、そのタスクから得たアーチファクトXを用いて同じようなタスクを行う。他の2つの国のグループも図には描かれているが、計算キャパシティ(マシン)のプロビジョニングを除いて、このシナリオには関与していない。

### 3.5.2 ユーザ

この場合のユーザとは、サービスのコンシューマである。ユーザは、技術的あるいはビジネス的な観点でプロセスを監視し、プロセスが終了するのを(そしてその結果出てくるオンライン・タスクリポートあるいは印刷されたタスクリポート)を待つ。サービスコンシューマは、正式なビジネス要件やその場だけのビジネス要件からコンフィグレーションを定義するため、ビジネスアナリストと作業を行うことがよくある。

システムマネージャは、ライブシステムの監視と問題調査のため、コンフィグレーションとライフサイクルマネジメントを使用する。

### 3.5.3 シナリオ

#### 1)リージョナルリスクシステムおよびローカルリスクシステムの動的デプロイメント

このユースケースは、世界的な投資銀行のリスク解析プロセスを一般化したものである(また、世界的組織、地域組織、国組織など、異なる組織レベルで共通の処理を行う際に手本となるものであ

る。)。こうした使用例では目のタスクを行うためにサービスを組み合わせることが必要になる。これらのサービスは、組織の異なる部署から派生する機会が多い(ビジネスラインの取引データセットや、さまざまな内外のソースからの市場データなど)。このシナリオには、取引のポートフォリオに関するリスク解析を必要とする国ベースのトレーダが登場する。これは、取引データセット、計算サービス、市場データサービスなどのコンフィグレーションに影響を及ぼす。このシナリオでは、(複数の国からなる)地域のリスクマネージャが同じような解析を行う必要があるために、より複雑化する。ただしこの解析を行う際に、地域のリスクマネージャは、その地域におけるすべての国々から得たデータセット(コンポジット・ポートフォリオ)やより多様な計算サービスと市場データサービスのリソースセットを使用することになる。さらにその地域のプロセスは、ある特定の状態になったときにのみ、国を単位としたトレーダの解析を使って開始される。

この使用例では、リソースの依存関係と優先順位が重要であり、適切なスイッチング対策が必要となる(たとえば、地域のプロセスが開始された場合には、他の国ベースのリソースよりも高い優先度で使用される、など)。予測可能なネットワーク監視、フォルト監視、計算監視(ネットワーク・ウェザー・サービスNWS[7][6]など)を行う手段を組み入れる必要がある。デプロイメントサービスは、こうした手段や、デプロイされたサービスの状態に関する情報を使って、再度のデプロイメントを行うのである。

## 2) テスト・デプロイメント

リスク解析プロセスは複雑であり、その信頼性を保つには、テストシステムをデプロイすることが必要である。テストシステムは、あらかじめわかっている入力データと結果から構成されているものである。デプロイメントと解析が終了すると、あらかじめわかっている結果と計算で得られた結果を比較する。このテストは、生産プロセスへの依存関係を見るものでもある。

### 3.5.4 関連リソース

このシナリオは、単純ではあるが、さまざまな組み合わせのリソースを伴うかなり標準的なものである。リソースの多様性には、計算能力の幅(デスクトップPCから大規模クラスターまで)、ネットワークの多様性、分散したデータなども含まれる。これらのリソースはいずれも標準的なバッチ処理(このシナリオの最重要ポイント)からよりアドホックな利用まで、分散型ステート管理サービスと組み合わせながらいろいろな方法で使用される。

この複雑な組み合わせでより高いレベルのサービスを考えると、計算、データ、パッケージ化されたアプリケーション(たとえばコマンドライン・インタフェースから起動するコードなど)、サービス、レガシ・サービスなどを絡めることになる。したがってこの使用例のコンテキストでは、次のような例が考えられる。

#### 1) コマンドライン・インタフェースでパッケージ化されたアプリケーションを使用する国ベースの計算サービス

- 2) 組織内 Web サービスである地域の計算サービス
- 3) Java コネクタ API[8]を伴ったレガシ・メッセージ指向ミドルウェア (MOM) である地域の市場データサービス。
- 4) データベースに保管された国の市場データ

市場データ、取引データ、解析コード、パッケージ化されたアプリケーション、実行インフラストラクチャ (マシンやネットワーク) を集めたものがグリッド計算の根幹となる。この異機種での組み合わせを提供するサービスのデプロイメントは、マネージド・ビジネスサービス指向アーキテクチャにとって重要である。

### 3.6 Web サービス・デプロイメント・プロジェクト

#### 3.6.1 概要

この使用例では、デプロイメントが開発サイクルにいかんして組み込まれるかを示す。そこからは、CDDL M と自動デプロイメントが作業システムの単純なデプロイメントに比べてより大きな役割を果たすことになる様子が見て取れる。すなわち、CDDL M と自動デプロイメントは、作業システムそのものを提供するプロセスの中核をなすものである。これは過去のプロジェクト[9]から学んだことに基づいている。

#### 3.6.2 ユーザ

直接の利用者は、開発チームとオペレーション・チームのメンバーである。彼らは自分たちのローカルシステム (開発チームのシステム) にデプロイするか、あるいはオペレーション・チームが管理するリモートの生産サイトにデプロイすることになる。これとは別に、外部からアクセスできる SOAP エンドポイントにコールしている間接的利用者が存在する。間接的利用者は、自分のアプリケーションを統合しようとしており、バグレポートの主なソースとなる人物である。この利用者の開発スケジュールを手助けするためには、アクセス制限をかけた生産サイトがそのプロジェクトの初期の段階から起動されており、定期的に更新されている必要がある。

#### 3.6.3 シナリオ

##### 1) 開発システムのデプロイメント

ある Web サービスプロジェクトが、外部の顧客のためにソフトウェアチームによって開発されている。そのアプリケーションは、Web サービスとして作られているが、その実装の詳細は呼び出し側からは見えない状態にある。外部の利用者はサニタイズされた WSDL を目にするだけである。

この Web サービスでは SVG イメージを提供する。このサービスは計算集約型であり、リモートのイメージ保管庫 (HTTP 経由でリードオンリーアクセスが可能) と一時的なデータキャッシュ (あらゆるシステムからのリード/ライト・アクセスが可能) を利用している。コールには 2 から 10 秒かかり、負荷がかかっている状態では 1 つのマシンが並行して扱えるリクエスト数がきわめて低くなる。グリッドア

ーキテクチャを使うと、必要に応じて動的にリソースをアロケートすることで、ピーク時の負荷でも処理できるようになる。その際重要なコンポーネントとなるものの一つが、動的に負荷バランスをとるフロントエンドである。このフロントエンドは、受け取ったリクエストをバックエンドの実行エンジンに送る役目をする。負荷が軽いときには、フロントエンドは自分の仕事をし、負荷が重いときは新しいリソースをアロケートし、フォワードする。これにはプーリングが使われる。

サービスをスケジュール通りに届けるために重要になるのが、最先端のデプロイメント中心の開発プロセスである。この開発プロセスは、「継続的統合 (Continuous Integration)」の概念に基づき、「継続的デプロイメント (Continuous Deployment)」の考え方でそれを拡張したものである。

開発者は自分たちのシステム上で、ローカルマシンへのデプロイメントのテストやデバッグを行う。より正確に言えば、開発者はオペレーションが与える OS イメージを持った仮想化マシンにデプロイする。これは、開発者のシステムと生産ユニットとをより分離させることで行われる。開発者は、開発ボックスではなく、生産システムに似たプラットフォーム上でつねにテストを行うことになる。

サーバシステムは、ソースコードの保管場所における変化を継続的に監視している。そしてそのコンテンツが変化した場合には、リビルドとユニットテストをトリガする。ユニットテストがパスすると、サーバシステムはローカル・グリッドにデプロイされる。ローカル・グリッドでは機能テストがサーバシステムに対して行われる。

開発プロセスで重要であるのは、たとえコンフィグレーションが行われたものであっても、テストで記入されうる不具合としての全てのデプロイメントの問題を処理する際のポリシーである。このためデプロイメントが完了するとすぐに、回帰を探してサーバに対して一連のテストを行う。その結果は開発チームに報告される。リビルドとリデプロイは完全に自動化されており、SCM チェックイン・イベントでトリガされるため、システムは1時間に最高4、5回のデプロイメントを行うことができる。

夜になると空き CPU リソースが増えるため、すべての開発マシンを含むより大きなグリッドが作られ、全面的な負荷テストがシステム上で毎晩行われる。朝になると、夜間のビルドの結果が適切であれば、そのコードが生産前サイトにデプロイされる。これは、更新の際に思わぬ副次的効果が発生しても、それをうまく処理するために昼間に行われる。副次的効果とは、たいていの場合、クライアントアプリケーションとの相互運用性の問題である。生産システムへのデプロイメントは完全に自動化されているものの、作業チームだけがそれを実行できる。この生産システムは、DMZ の他のサーバとリソースの一部を共有することもできる高仕様サーバである。

## 2) ライブ・デプロイメント

サービスが開始されると、システムはモニターされ、更新される。開発段階で見られる初期の問題としては、ロードバランサが他のノードの状態を完全に把握していなければ、適切な状態にないシステムに対して、そのシステムのリクエストキューが短いという理由でリクエストを送ってしまうことがある。この問題は、各ノードの状態チェックをこれまで以上に行うことで解決する。すなわち、内部状態チェックをトリガする各ノードの URL をロードバランサが詳しく調べるのである。したがってライブネスは、ノードにアプリケーションが存在しているかどうかで決まるのではなく、アプリケーションが

自身の状態を以下の方法で調べる能力によって決まるのである。

- ・テストイメージを出力し、高品質のビットマップと比較する。テストイメージにより、必要なフォントがすべて存在しているかどうか分かる。
- ・ファイルを一時的な格納庫に保存し、そのファイルのタイムスタンプがローカルマシンのタイムスタンプに近いものであることをベリファイする。これは、タイムゾーンやクロックに差があると混乱を引き起こすからである。
- ・コンフィギュアしたヘルパーサイトの DNS リクエストを作成する。
- ・その他のデプロイメント時間の不具合。これは重大なシステム障害のために簡単に調べることができるものであり、またこれを使ってシステム障害のテストを行うこともできる。

サービスが実行されると、さまざまな負荷下で計算集約的な機能が提供される。ただし、開発と生産の計算リソースの両方を分け合えることから、動作コストは低く抑えられる。

#### 3.6.4 関連リソース

このシナリオでは、開発者、オペレーション、継続的統合のビルドツールは、CDDLIM という手段を使ってアプリケーションを開発システム、テストシステム、生産システムにデプロイする。当初の利点は、最終目的とは関係のない、高速で単純なデプロイメント・メカニズムにある。つまり生産サイトへのデプロイメントは、開発システムへのデプロイメントと比べても複雑さに変わりがない。ただし、生産のデプロイメント記述子はいくぶん複雑なものである。長期的な観点での利点は、グリッドアーキテクチャを動的リソース・アロケーションとともに使用することで、サービスの実行の費用効果が高くなることにある。

開発ボックスは、仮想的な PC VM 内のローカルシステム上で採用される最初のリソースである。VM イメージはオペレーションから提供され、開発者はこのローカルイメージに接続し、コードをデプロイする。自動テストデプロイメント・プロセス(これ自身 CDDLIM を使ってデプロイされることもある)は、使用可能なシステムを LAN で接続したグリッドを使う。可能であれば使われていないワークステーションも活用する。生産システムは、オペレーションが何を提供するとしても、たとえばそれが専用のハードウェアであったり、共有リソースであったりしても、それを使用することができる。サービスが動いている間は、オペレーションが何を提供するかという選択を固定化させる必要はない。

システムの多くは計算ノードである。ルータ・コンポーネントとファイル保管庫は別のシステムである。開発システムとテストシステムは、ローカルのファイル保管庫を使用することができる。生産には、よりすぐれたソリューションが必要であるが、LAN の範囲内でのソリューションに限られる。ネットワーク接続されたファイルサーバは、その基本的なソリューションの 1 つであろう。なおファイルサーバは出力を一時的に保管するためだけのものなので、RAID-5 のフルストレージデバイスは過剰である。

### 3.6.5 使用例の状況解析

このシナリオは、既存の(文書化されている)Web サービスプロジェクト [9]に由来し、仮定に基づいて作られたものである。その中で検討された開発プロセスは、プロジェクトのテスト方法やデプロイ方法を既存のものから変更するために用いられた[10]。デプロイ先には 3 箇所ある。開発者のシステム、ローカルのテストグリッド、そして生産サーバ群である。これらはいずれも独自のコンフィグレーションを持っているものの、共通のデプロイメント・メカニズムとして CDDLm を共有している。このシナリオの特筆すべき点は、そのデプロイメントの速度である。デプロイメントは、開発システム上で数分ごとに行われ、テスト・フレームワーク上で1時間に数回行われる。顧客はこの Web サービスの外部ユーザなので、システムの毎日のビルドにアクセスする必要がある。これはすなわち、生産前サイトでさえも毎日更新されることを意味する。

システムの多くが実際には開発システム上にホストされた仮想化 OS イメージであることが、話を複雑にすることも、しないこともある。われわれは、プレコンフィギュアした OS イメージを開発者に受け渡す集中型デプロイメント・メカニズムを想定している。開発者は、ロックダウンした生産システムのエミュレーションをオペレーションにより提供する。夜を徹したテストで仮想化オペレーティングシステムがかなり使用されることは、想像に難くない。その結果、チームは、生産サイトの物理アーキテクチャをより適切にシミュレートできるであろう。

以下では、3.1 節から 3.6 節に示した使用例を取り上げ、OGSA の機能要件、プラットフォームサービスの利用、セキュリティや性能の問題をまとめる。

### 3.7 OGSA プラットフォームに対する機能要件

検出とブローカリング: 上に示したシナリオでは、ハードウェアやソフトウェアのリソースは、デプロイメントのための準備が整っているものと仮定していた。ハードウェアリソースの検出やブローカリングは、デプロイメントの範囲外の問題である。

デプロイメント: 本書で説明

仮想組織: デプロイメントは、仮想組織を構築、維持するためのプロセスの一部である。

ポリシー: デプロイメントには、システムのフォルト、リカバリ、実行エラー、その他の例外的状況といった特別な出来事に対するポリシーが必要である。

複数のセキュリティインフラストラクチャ: ここで取り上げたサービスは、本質的に異なる複数のシステムを使うことができる。これらのシステムは、Web サービスそのものへのアクセス方法とは違い、セキュリティに関しては独自方法を使う場合やレガシな方法を使う場合がある。

リソースの仮想化: サーバ・プールの管理を可能にするために、上で使われている Web サービスが

専用リソースに固定化されていないことを前提にしなければならない。さらに、P2P の利用シナリオでは、動的プロビジョニングの必要性が示されている。

プロビジョニング: 上記の使用例は、プロビジョニングの中のデプロイメントの部分を目的としている。

メータリングと会計: この種の環境では、サービスの所有者がエンドユーザに対し、さまざまな種類の会計基準でサービスの課金を行えることが必要である。

監視: すぐに終わるサービスと長く生き残るサービスがあるため、サービスレベルが保たれ、Web サービスに伴う問題に対処する手段が用意されているということについて、ユーザが確信を持てることが重要である。上記の使用例のうち 2 件に登場する金融環境では、技術停止による金銭的損害を避けるため、問題に素早く対処できる方法が必要となる。ライフサイクルマネジメントは、サービスの健全な状態を維持するための監視サービスに依存している。

災害復旧: アベイラビリティの高い環境の多くでは、災害復旧はサービスが作動できるようになるための基本的な必要条件である。

自己修復: 自己修復が必要になる場合がある。デプロイメントサービスは、不具合の生じた状態を検出し、再デプロイすることが可能でなければならない。

レガシアプリケーション管理: この環境における Web サービスにとって、レガシアプリケーションとのやりとりは通常のことである。デプロイメントが問題なく行われたことを確認し、不具合について正確な報告を行うためには、グリッド・デプロイメントシステムがレガシ管理システムと通信できる必要がある。

管理: ユーザグループは、デプロイメントのプロセスと現況に関する報告を簡単に管理できる必要がある。同じインフラストラクチャ内で動いているさまざまなコンフィグレーションを管理し、その正確性や規定されたシステムポリシーを遵守していることを保証する有効な機能が必要である。

認証、認可、計上(AAA): サービスのデプロイメントのリクエストは、認証、認可、計上する必要がある。Web サービスに対するセキュリティ上の要件やソリューションの標準を遵守するためである。

リソース・アロケーション: CDDLM は、デプロイメントを行うことで、プロビジョニング・ソリューション全体に貢献している。しかし、リソース・アロケーションは、これとは別に提供されるものと想定している。

フォルトの扱い: 監視により不具合が見つかった場合は、サービスを再度デプロイするために、CDDLML がフォルト・ハンドリング・モジュールとやりとりを行う。

### 3.8 OGSA プラットフォームサービスの活用

サービスインタラクション・サービス: 多くの場合、共通のインフラストラクチャに複数のサービスが共存しており、相互に作用しあっている。その際、サービスが互いを見つけ出し、調整しあうためのロビジョンが必要である。

セキュリティサービス: ここに定義した環境では、サービスのオペレーションを行う上で情報の安全が重要なポイントとなる。サービスは、そのホスト環境が定義するセキュリティモデルの範囲内に納まるものでなければならない。

データサービス: さまざまなデータセットは、すでに紹介したシナリオやその他のシナリオにおいて、Web サービスのオペレーションに欠かせない部分である。

基本実行サービス: 為替取引チームの VO リソースをアロケートする目的で、すでに概要を示した P2P シナリオや複雑なビジネスシナリオを実現するには、実行管理を使うことが必要条件となる。

リソース管理サービス: 紹介したシナリオは、OGSA プラットフォームサービスのうち、このカテゴリに属するいくつかサービスの実装を定義するものである。

### 3.9 セキュリティに関する考慮

これらのシナリオのデプロイメントサービスは、VO やユーザが指定した他のシステムの範囲外にあるサービスやデータを切り離し、保護することで、特定のアプリケーションを実行する安全な環境を与えられたポリシーに沿って適切に設定することができるものである。

デプロイメントは、コンフィグレーションの定義、デプロイできるソフトウェア、コンテンツの保管に関し、エンド・ツー・エンドのセキュリティを保証するものでなければならない。これらのソフトウェアやコンテンツをすべてデプロイされたソフトウェアやコンテンツに移行しなければならない。

デプロイメントに関するデータは、データセンタの外部からの妨害や、他のデータセンタのユーザやそのシステムからの妨害に対し、防御されている必要がある。コンテンツの一部は法人ユーザに対して機密の内容を含んでおり、これらはデータセンタのオペレータでさえもアクセスできないように要請されている。

CDDLML の基本前提は、グリッド内でサービスをデプロイ、コンフィギュアすることである。考慮すべきセキュリティの基本的問題点は2つある。1つは、サービスのデプロイメント自体がホストネットワークのセキュリティポリシーを破ってはならないという点である。2つめは、デプロイされたサービス

のコンフィグレーションが、決められた導入ポリシーが要請するような安全性を確保している必要があるという点である。

### 3.10 性能に関する考慮

システムは、性能に関して以下を含むさまざまな側面を考慮する必要がある。

- 1) 決められたコンポーネント、サーバ、その他のハードウェアデバイスの数に関するスケーラビリティ
- 2) コンポーネント記述のコンパイル速度あるいはインタープリット速度
- 3) デプロイされたコンポーネントのサイズや数、あるいはサーバやその他のハードウェアデバイスの数に対するデプロイメント速度

それぞれ定義されたシナリオにおいて、デプロイメントのパフォーマンスが与える影響は異なる。一般化 Web サービスのシナリオやエンドユーザデバイスのシナリオでは、デプロイメントはサービスを利用する前に静的に行われる。その場合、スケーラビリティや相対的なデプロイメント速度がおもな性能上の問題となる。

システムは、かなりの数の物理的デバイスやエンドポイントを取り扱うための能力、そしてコンフィグレーションとデプロイメントの依存関係やプロセスステップをある程度の数まで取り扱うための能力を、それぞれ考慮するためにスケールできるものでなければならない。その際、システムが実際に使用可能なものとなるためには、デプロイメントが適切な時間内に終了する必要がある。大企業の場合は、潜在的なスケールに合わせてデプロイメントが並行して行われる必要があり、さらに(または)デプロイメントがブロードキャストできるものでなければならない。

ピアツーピアの分散型計算の場合は、サービスのインスタンス作成時間はたいした長さとはならず、サービスが走っているプロセッサ経過時間の一部分を占めるに過ぎない。そうでなければ、サービスを上述のようにデプロイする意味がなくなるからである。したがって、デプロイメントサービスは、費用解析やデプロイメント計画決定のプラットフォームに収まるものでなければならない。

## 4 ハイレベル CDDL M アーキテクチャ

CDDL M は 3 つのメインコンポーネントから構成されている。1 つはコンフィグレーション言語であり、デプロイされるサービスを記述する際に使用される。2 つめは基本サービスであり、デプロイメントをリクエストする際に使用される。3 つめはコンポーネントモデルであり、サービスのライフサイクル管理をサポートする。これら 3 つのコンポーネントを図 6 に示した。本節では、これらをより詳しく解説する。2 つの言語スペック (SmartFrog および XML ベース) とランタイムバインディング・スペックによりコンフィグレーション記述言語が記述される。基本サービスとコンポーネントモデルは、対応するスペックで記述される。

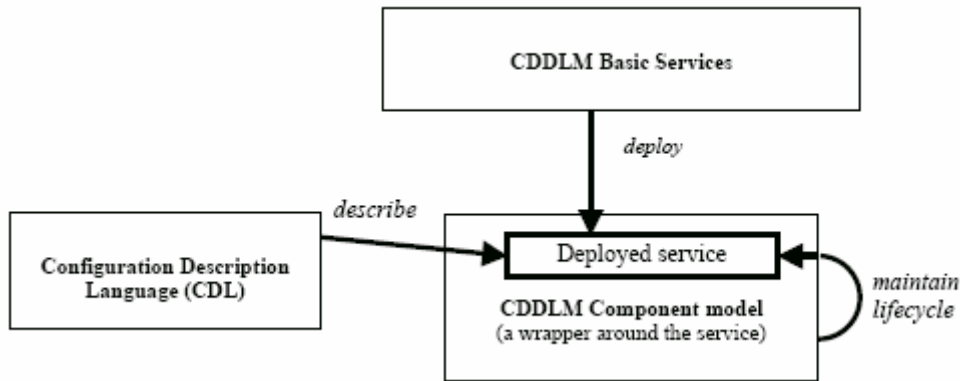


図 6 CDDL M のハイレベルアーキテクチャ

図 7 は、グリッド内で CDDL M が他のコンポーネント、とくにプログラム実行とリソース予約、とどのように関係しているかを図示している。とりわけプログラム実行は、実行スタックの上位にあり、デプロイメントのリクエストはプログラム実行から CDDL M の基本サービスに対して出される。このリクエストは CDL で書かれたデプロイメントテンプレートを受け渡す。サービスがデプロイされる予定のリソースは、CDDL M の起動の前に、WS-アグリメントなど他の予約プロトコルを使って予約される。そして CDDL M サービスの起動に伴い、予約されたリソースが特定されるのである。

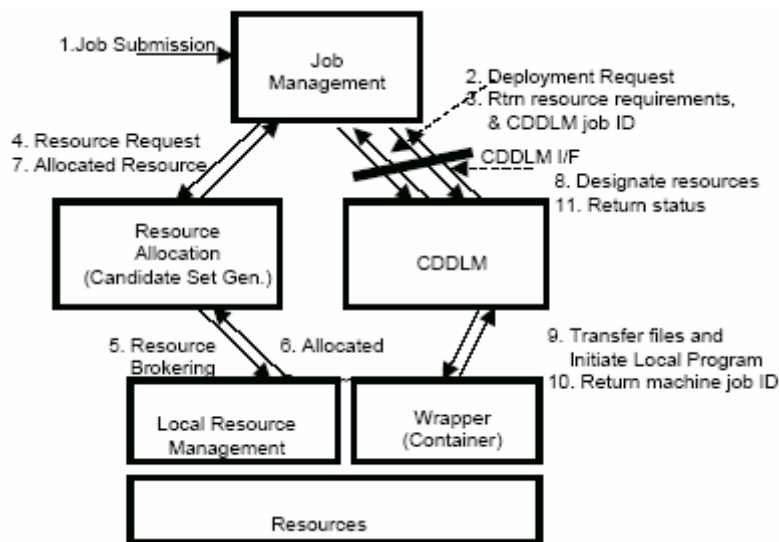


図 7 CDDL M と他の管理コンポーネントとのやりとり

#### 4.1 コンフィグレーション記述言語の仕様

コンフィグレーション記述言語には、少なくとも 2 種類ある。1 つは SmartFrog ベースのものであり、もう 1 つは XML ベースのものである。サービスコンフィグレーションは、CDL で書かれたテンプレートで表現されている。各コンフィグレーションは、フロントエンド言語で読み替えたものから構成され

ており、そこには対象をより具体的に記述するためにテンプレートとして使用されるリゾルブ前のパラメータやリファレンスも含まれる。これらのパラメータやリファレンスは、デプロイメントの前に一部リゾルブされるが、いくつかはリゾルブされないまま保持され、デプロイメントの間や実行の初期化作業の間にリゾルブされることになる。

#### 4.2 基本サービスの仕様

CDDLMLのおもなサービスはデプロイメントである。デプロイメントサービスは、CDLで書かれた対象となるサービスのコンフィグレーション記述を受け取り、解析をし、整合性をベリファイし、コンポーネントのリファレンスをリゾルブする。ただし、それがリゾルブ可能であり、パッケージをデプロイメントできる形で構成している場合に限る。つぎに対象サービスのインスタンスを作成する役割を持つサブコンポーネントを起動し、対象サービスをそのライフサイクルに渡ってインストール、イニシエート、スタートなどの処理を行う。サービスに不具合が生じた場合は、デプロイメントサービスが再度起動される。

CDDLMLは、コンフィグレーションやライフサイクルの機能をサポートしているものの、提供するものはデプロイメントサービスのみである。CDDLMLがコンフィグレーションやライフサイクルを提供しないおもな理由は、これらが他の標準団体、とくに WSDM により扱われているからである。さらにコンフィグレーションとライフサイクルの管理機能は、どちらもアプリケーション固有のものである。あるサービスのコンフィギュレーションは、そのサービスの助けがなければ行うことができない。われわれは、コンフィグレーションというものはまさに管理機能であり、CDDLMLの範囲外でそのように扱うべきものであると強く信じる。CDDLMLが提供する既存のコンフィグレーションとライフサイクルの機能は、おもにデプロイメントを補助するものである。したがって、クライアントはCDDLMLが実行する複雑なコンフィグレーションをリクエストすることができる。同様に、デプロイされたサービスに不具合が生じた場合にはCDDLMLが不具合を検出し、依存関係を調べ、そのサービスおよび依存関係にある他のすべてのサービスを必要であれば再起動する。

#### 4.3 コンポーネントモデルの仕様(表現と実装)

CDDLMLがデプロイしたサービスは、いずれもインタフェースの基本セットを実装し、コアとなる抽象コンポーネントモデルを定義する。Webサービスや実際にインスタンスを作成したソフトウェアアリス間の間には多対多の関係が存在しているため、CDDLMLのコンポーネントモデルは、デプロイメントコンテナ内でサービスの階層構造をゆるやかにグループ化する手段を定義する。管理対象のWebサービスはいずれも、インタフェースをコンフィグレーション、ライフサイクルマネジメント、メンテナンスの3つのカテゴリで実装する。コンポーネントモデルはまた、サービスとそのコンポーネントの関係についてもサポートしている。

### 5 関連 GGF、その他の標準団体のワーキンググループ

#### 5.1 GGF OGSA

OGSA-WS の設立綱領に次の文章がある。「OGSA ワーキンググループの目的は、OGSA サービスの要件、機能、優先度、相互関係を文書化することにより、OGSA サービスの今後の開発に対する統合的アプローチを構築することにある。われわれがまず個別に調べたいトピックスは、共通のリソースモデルとサービスドメインのメカニズムであるが、取り扱う正確な対象は初期の議論で決定する予定である。

本 WG では、主要なサービスに対する要件を定義、精査、概説する OGSA アーキテクチャロードマップ文書を作成する。特定のサービスに関する詳しい仕様の開発は、他の(既存の、あるいは新しい)WG で行われると期待される。」

CDDLML は、アーキテクチャに位置づけられ、グリッドアーキテクチャのデプロイメントとライフサイクルマネジメントの役割を果たすものである。OGSA のブランド設定に対して WG が与えた条件は次の通りである。これらは OGSA アーキテクチャに位置づけられた基本条件標準コンポーネントとみなすことができる。

- 1) グリッドプラットフォーム(OGSI、WSRF など、CDDLML の実装時に採用されるものすべて)の範囲内でのオペレーション
- 2) OGSA 文書が定めるアーキテクチャに適合したワーク
- 3) 本 WG および OGSA 関連の他の WG と積極的に連動
- 4) OGSA を「形容詞」として使用

## 5.2 GGF OGSA 基本実行サービス

基本実行サービスは、OGSA の基本コンポーネントの1つである。そこにはプログラムの実行に必要な機能も含まれる。詳しい議論が WG 内で行われているところであるが、CDDLML の機能はアーキテクチャに位置づけられ、アーキテクチャ内で適切に使用されるべきものである。

## 5.3 OASIS WSDM

CDDLML は、OASIS/WSDM(Web サービス分散管理)に関係しているが、デプロイメントに重点を置いたものである。また、OGSA の一部としてプロビジョニングにも関係している。しかし CDDLML はプロビジョニングツールではない。CDDLML はデプロイメントツールとして、プロビジョニングに使用できるのである。ワークロードの監視、天候やトレンドの解析や未来予測、リソース・アロケーションの計画、新たにアロケートしたリソースのデプロイメントやコンフィグレーションなどを行うため、グリッドジョブにリソースをアサインあるいはアロケートする作業としてプロビジョニングを定義した場合、リソースが最低限の機能を有することをプロビジョニングが保証する責任がある。この作業が終了すると、プロビジョニングされたハードウェア上でのデプロイメントやコンフィグレーションを含む Web サービスのソフトウェアライフタイムに対して CDDLML が責任を負うことになる。この意味で、CDDLML はデプロイメントとコンフィグレーションだけを扱っているのである。Web サービスを通じて物理的 IT リソースを取り扱うためには、CDDLML が SMTP や GMPLS などの多くの既存の管理仕様と関連づいている必要がある。CDDLML はまた、CMM や OGSA と密接に結びついているが、CDDLML とこれら

他のワーキンググループとの間には重なる部分があったとしても限定的である。

WSDM-TCは、Webサービス管理を定義するために作られた。そこには分散型リソースを管理するためのWebサービスのアーキテクチャや技術が含まれている。TCはまた、Webサービスのモデルを管理可能なリソースとして開発することになっている。このTCは、DMTF(関連するCIMスキーマに関する技術ワーキンググループとともに社業を行っている)、GGF(OGSAリソースモデルに関する団体)、W3C(Webサービスアーキテクチャ委員会)など、これに限らないが、このような他の標準団体が次々に行っているさまざまな活動とも協力していくことになっている。また、セキュリティTCや管理指向のTCなど、他のOASIS TCとも連結している。

#### 5.4 OASIS SPML および WS-プロビジョニング

サービスプロビジョニングマークアップ言語(SPLM、[www.openspml.org](http://www.openspml.org))は、OASIS([www.oasis-open.org](http://www.oasis-open.org))の標準の1つであり、プロビジョニングサービスポイント間で情報を交換するためのXMLベースのプラットフォームを定義するものである。SPMLはユーザ、リソース、サービスのプロビジョニング情報を交換するためのプラットフォームであり、OASISプロビジョニングサービス技術委員会が定めたものである。SPMLはユーザのプロビジョニングに重点を置いており、その定義は「プロビジョニングは、ユーザやシステムのアクセス権限、あるいは電子発行されたサービスに関係したデータを管理(セットアップ、修正、破棄)する上で必要なあらゆるステップを自動化するものである」となっている。

WS-プロビジョニングはSPML v2.0への入力として提案されたものである。WS-プロビジョニングは、プロビジョニングシステム間の相互運用性を容易にするため、そして、ソフトウェアベンダにプロビジョニングツールを整合性の取れた形で提供してもらうようにするために、APIやスキーマを記述している。その仕様は、リソースのプロビジョニングとデプロビジョニング、ターゲットデータやターゲットスキーマの情報の取得といった、プロビジョニングシステムに共通するおもなエンティティやオペレーションのためのモデルを定めるものであり、さらに、プロビジョニングされた状態のライフサイクルを記述、制御するためのメカニズムを提供するものである。

#### 5.5 CIM

CIM(共通情報モデル)は、分散型管理タスクフォース(DMTF、[www.dmtf.org](http://www.dmtf.org))が開発、標準化したものである。CIMは、ネットワークや企業環境における管理情報全体を記述するための実装中立のスキーマの共通データモデルである(<http://www.dmtf.org/standards/cim>)。統一モデリング言語(UML)に基づき、1つのメタモデルの範囲内で数多くのリソース、アプリケーション、サービスに対するモデルを含有している。CIMモデルは、UMLまたはMOF(管理対象オブジェクト形式)テキストで書き表されている。

#### 5.6 DCML

データセンタマークアップ言語(DCML)は、新たに作られたコンソーシアム([www.dcml.org](http://www.dcml.org))であ

り、「データセンタ環境、データセンタのコンポーネント間の依存関係、これらの環境の管理と構築を司るポリシーなど記述するベンダー中立でオープンな言語」を提案している。DCML は、データセンタのコンフィグレーションと変更の自動化、コンフィグレーションや利用状況などのデータセンタ表現の可視化の向上、データセンタのフォルトの監視、などに関与している。それは、アダプタビリティの考え方、ベストプラクティスや標準およびポリシーなどの表現の考え方、異機種混在(多様性)の考え方に基づいたものである。DCML は、プラットフォーム、ハードウェア、ネットワーク、ストレージ、ソフトウェアを扱う5つの異なる仕様から成り立っている。

### 5.7 WS-通知

システムがユーザに対して通知を送り返す必要が出てくる。この通知は、ユーザ自身の Web サービスインスタンスに対して出された同期コールによって作成することはできない。それは、ユーザがファイアウォールの外に存在している場合やオフラインである場合があるからである。したがって、われわれのニーズに合った非同期通知メカニズムを使う必要がある。WS-通知は、こうしたメカニズムか、あるいは、ファイアウォールを通した安全なコールバックを行うために GGF が推奨する他の通知メカニズムとなる可能性がある。

### 5.8 GRAAP

グリッドリソース・アロケーションアグリメント・プロトコル(GRAAP)は、グリッド内に分散したリソースのアロケーションを取り扱うものである。これはグリッド内で使用されているさまざまなスケジューラが管理する多様なリソースを取り巻く「スーパースケジューリング」サービスを通じて作られたものである。GRAAP ワーキンググループは、スーパースケジューラ(グリッドレベル・スケジューラ)と、このサービスに対して構成するブロックとしての、グリッド内でリソースを予約、アロケートするのに必要なローカルのスケジューラとの間のプロトコルを扱っている。CDDLMLとGRAAPは互いに補い合うことでプロビジョニングをサポートしている。GRAAPは図7の左側を表している。

### 5.9 JSDL

ジョブサブミッション記述言語(JSDL)は、言語バインディングとは独立の抽象標準言語となる仕様を与えるものである。JSDL との比較で言えば、CDDLML は、JSDL が関与しないデプロイメントを中心に扱っている。一方 JSDL は、一般的なバッチシステムのスケジューリングの詳細を理解している。GRAAP や OGSA 基本実行と同様に、JSDL と CDDLML は相補的にプロビジョニングモデル全体に貢献している。

## 6 関連する作業

本節では、CDDLML の問題に関連する技術を概観する。CDDLML の興味を中心は、サービスのデプロイメントのためにリソースがアロケートされることから始まる。したがってわれわれは、起動中のサービスを引き渡すためにソフトウェアをリソース上でデプロイ、コンフィギュアすることのできる技

術に関心がある。この技術には、OS のインストールの自動化ツール、ノードイメージングツール、アプリケーション管理ツールなども含まれる。個々ではリソースとしての計算ノードに焦点をあてるが、その原則は他の種類のリソースにも拡張できる。

Red Hat Linux (<http://www.redhat.com/>) の Kickstart パッケージのような OS のインストーションツールは、OS やアプリケーションパッケージのインストールプロセスを自動化するものである。その際、一般にはコンフィグレーションファイルやレスポンスファイルを使用する。この作業は自動であるが、ノードの準備には通常かなりの時間がかかる。Altiris (<http://www.altiris.com/>) や Rembo (<http://www.rembo.com/>) などのノードイメージングツールは、この機能を拡張し、ゴールドイメージのライブラリからディスクイメージ全体をノード上でカスタマイズしたりインストールしたりすることができるようにするものである。これは、スピードは速いが、対象となるノードのコンフィグレーションに適合する上で、一般に柔軟性に欠ける。

これら 2 つのアプローチは、いずれもソフトウェアをデプロイする機能が、どちらかという静的なものである。ノードのコンフィグレーションが変化しつつあり、動的な場合でもそれを管理することができるシステム、たとえば LCFG (<http://www.lcfg.org/>) などは、あらゆるノードのコンフィグレーションのデータベースを持っている。1 つあるは複数のノードのコンフィグレーションが変化させるときには、この変化をデータベースに追加し、変化の影響のあるノードのコンフィグレーションを適切に変えるのである。こうした技術は、デフォルト状態(ただし完全に稼働できる状態)にあるノードが与えられれば、コンフィグレーションを迅速に調整して特定のサービスに貢献することができる。

上述したアプローチの難点の 1 つは、いくつかのリソースを横断して 1 つのリソースをデプロイする際の調整能力にある。この能力は、CDDL M の関心の中心に位置するものであり、SmartFrog [10] などのシステムに組み込まれている。SmartFrog などではコンフィグレーション・ドリブンのシステムを作成することが可能であるが、そのコンフィグレーション・ドリブンのシステムにおいては、複雑なソフトウェアのデプロイメントを、サービスコンフィグレーションデータに基づいて、分散したリソースを横断する形でオーケストレーションすることができる。

静的なデプロイメントの方法のさらなる弱点は、デプロイメントを確実に成功させるために、ソフトウェアパッケージのインストールの際のコンフリクトを避け、パッケージ内のソフトウェアを 1 対 1 で結び付けられるよう、相互依存関係に関する情報を保持する必要があることである。Softtricity 社の SoftGrid システムは、ソフトウェアをオンデマンドでリソースにデプロイし、ノード上に実際にインストールしなくてもソフトウェアが機能するようにすることで、この問題を解決している。同社のシステムは、仮想化技術を用いることで、再コンフィグレーションプロセスにより時間とともにノードが劣化することがないようにし、ソフトウェアが適切な特定のコンフィグレーションで問題なくデプロイされることを保証している。

CDDL M の目標は、GGF の目指すところ、より具体的に言えば OGSA アーキテクチャ、におもに関係している。しかし、ハイレベルな CDDL M アーキテクチャは、おもに p2p JXTA アーキテクチャあるいは同様に SmartFrog ベースのアーキテクチャなど、他のアーキテクチャにも適用できる。ただしその適用に関しては、本文書、および本ワーキンググループが当初作成予定の仕様の範囲を超

えることである。

## 7 今後の作業

1. 計画したすべての仕様を完成する。
2. OGSA との連携を図る。(基本的文書へのフィードバック、仕様のフォローアップ)
3. 他の GGF グループとくに、DAIS-WG、CMM-WG、GRAAP-WG、JSDL-WG、OGSA-WG 基本サービス、WFM-WG、EG-RG.DRMAA-WG、CGS-WG との連携を維持する。
4. GGF に入らない標準団体、とくに DMTF アプリケーションとユーティリティーコンピューティング、OASIS の WSDM と WSRF、DCML との関係を構築する。
5. 産業、学術との関係を構築する。
6. 少なくとも 2 つの相互運用可能なリファレンス実装を作る。
7. デプロイされたサービスの可視化とセンサ。
8. コンフィグレーションプロファイルの例を提示する。

## 8 セキュリティに関して

セキュリティの問題は本文書を通して広く扱った。

## 9 編集者情報

David Bell

Department of Information Systems and Computing,

Brunel University,

Uxbridge, Middlesex, UB8 3PH, United Kingdom.

Phone: +44 (0) 1895 203397

Email: david.bell@brunel.ac.uk

Patrick Goldsack

Internet Systems and Storage Laboratory

Hewlett-Packard Laboratories Mail Stop HPLB

Filton Rd., Stoke Gifford, Bristol BS34 8QZ, United Kingdom

Phone: +44 117 312 8176

Email: patrick.goldsack@hp.com

Takashi Kojo

Solution Platform Software Division

NEC Corporation

2-11-5 Shibaura, Minato-ku, Tokyo Japan 108-8557

Phone: +81-3-5476-4386

Email: [kojo@bk.jp.nec.com](mailto:kojo@bk.jp.nec.com)

Steve Loughran

Internet Systems and Storage Laboratory

Hewlett-Packard Laboratories

Filton Rd., Stoke Gifford, Bristol BS34 8QZ, United Kingdom

Phone: +44 117 312 8717

Email: [steve\\_loughran@hpl.hp.com](mailto:steve_loughran@hpl.hp.com)

Dejan Milojcic

Internet Systems and Storage Laboratory

Hewlett-Packard Laboratories

Mail Stop 1183, 1501 Page Mill Road, Palo Alto, CA 94304

Phone: (650) 236 2906

Email: [dejan@hpl.hp.com](mailto:dejan@hpl.hp.com)

Stuart Schaefer

Chief Technology Officer

Softricity, Inc.

27 Melcher Street, Boston, MA 02210, 617-695-0336

<http://www.softricity.com>

[sschaefer@softricity.com](mailto:sschaefer@softricity.com)

Junichi Tatemura

NEC Laboratories America, Inc.

10080 North Wolfe Road, Suite SW3-350, Cupertino, CA 95014-2515

Phone: +1-408-863-6021

Email: [tatemura@sv.nec-labs.com](mailto:tatemura@sv.nec-labs.com)

Peter Toft

Internet Systems and Storage Laboratory

Hewlett-Packard Laboratories, Mail Stop HPLB

Filton Rd., Stoke Gifford, Bristol BS34 8QZ, United Kingdom

Phone: +44 117 312 8728

Email: peter.toft@hp.com

#### 10 寄稿者

本仕様への寄稿に関し、Jem Treadwell、Hiro Kishimoto、Greg Newby の各氏、そして GGF 編集者に感謝の意を表したい。

#### 11 謝辞

この作業の一部は、HP、Softtricity、NEC の支援のもとで行われた。

#### 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。

#### 著作権情報

Copyright © Global Grid Forum (2002-2005). All Rights Reserved.

本文書およびその翻訳物は、複製し、他者に提供することができる。本文書に関してコメントや別の説明を与えている派生著作物、あるいは本文書の普及を助ける派生著作物についても、そのままあるいは部分的に、制約なしに作成、複製、発行、頒布が可能である。ただしそのような複製物や派生著作物については、上記の著作権情報と本段落に書かれていることを記載しなければならない。ただし、GGFや他の組織に対する著作権情報や参考文献を削除するなどといった本文書自体の改変は、いかなる形であっても禁止する。なお、グリッド提言(Grid Recommendations)を開発する目的で改変が必要ならば、その限りではない。この場合、GGFドキュメントプロセスで決められた著作権に対する手続きを遵守する必要がある。また、英語以外の言語に翻訳する際に改変が必要な場合も、その限りではない。

上に与えた制限付き許諾は永続的なものであり、GGFあるいはその後続組織または委嘱組織が無効にすることはない。

本文書とそこに含まれる情報は保証されたものではない。グローバル・グリッド・フォーラムは、こ

ここにおける情報の使用がいかなる権利をも侵害していないこと、市販性、特定目的との適合性に関するいかなる黙示保証をも侵害していないことを含む(ただし必ずしもこれらに限定されない)明示あるいは暗示の保証を拒否するものである。

#### 参考文献

- [1] Treadwell, J. (ed.) Open Grid Services Architecture Glossary of Terms. Global Grid Forum OGSA-WG. GFD-I.044, January 2005.  
<http://www.ggf.org/documents/GWDI-E/GFD-I.044.pdf>.
- [2] Configuration Description, Deployment, and Lifecycle Management SmartFrog-Based Language Specification, <http://forge.gridforum.org/projects/cddlmgw/document/CDDLMSmartFrogCDL/en/1>.
- [3] Configuration Description, Deployment, and Lifecycle Management (CDDLML) XML-Based Language, Document in preparation
- [4] Configuration Description, Deployment, and Lifecycle Management (CDDLML) Component Model, Document in Preparation.
- [5] Configuration Description, Deployment, and Lifecycle Management (CDDLML) Deployment API, Document in Preparation.
- [6] Foster et al. "Open Grid Services Architecture Use Cases,"  
[https://forge.gridforum.org/docman2/ViewProperties.php?group\\_id=42&category\\_id=0&document\\_content\\_id=923](https://forge.gridforum.org/docman2/ViewProperties.php?group_id=42&category_id=0&document_content_id=923)
- [7] University of California, Network Weather Service, <http://nws.cs.ucsb.edu/>
- [8] Sun Microsystems, Java Connector API, <http://java.sun.com/j2ee/connector/>
- [9] *When Web Services Go Bad*, Loughran, 2002.  
[http://www.iseran.com/Steve/papers/when\\_web\\_services\\_go\\_bad.html](http://www.iseran.com/Steve/papers/when_web_services_go_bad.html)
- [10] *Making Web Services that Work*, Loughran, 2002.  
<http://www.hpl.hp.com/techreports/2002/HPL-2002-274.html>
- [11] SmartFrog reference manual  
<http://www.hpl.hp.com/research/smartfrog/papers/sfReference.pdf>.