クラウド環境における Web サービスの設計とライフサイクル管理*

細野 繁^{*1}, 赤坂 文弥^{*2}, 木見田 康治^{*2}, 下村 芳樹^{*3}

Design Method and Lifecycle Management for Cloud-Compliant Web Services

Shigeru HOSONO^{*1}, Fumiya AKASAKA, Koji KIMITA and Yoshiki SHIMOMURA^{*1} NEC Knowledge Discovery Research Laboratories
1753, Shimonumabe, Nakahara-ku, Kawasaki, Kanagawa, 211-8666 Japan

Cloud services have provided application interfaces consisting of software programs and hardware resources through the net. The cloud services have created a trend in the industry – making ownership of software and hardware products for their system unnecessary. Accordingly, web services are developed through combining these interfaces leased from the cloud. However, this trend has also highlighted the gaps between design and operation. The permeation of virtualization technologies for servers and storages require developers to focus on resources as they become design choices. Likewise, operators are required to have concern over application components as the cloud service interfaces enable rapid prototyping and staging of web applications. However, both parties are not able to combine the application logic and resources as each is supervised by the other party. To address this issue, this paper focuses on resource modeling. This approach enables separation and integration of both application logic and resources in design, test, and operation phases, and it is carried out as follows. The first method enables reasonable requirement definitions while run-time resources in the cloud are referred and adapted to fit. The second one streamlines architectural design of both software and virtualized resources comprehensively, in accordance with Axiomatic Design, DSM and Design-for-X. The third enables independent test of functionality of applications and non-functional requirements, e.g. security and performance, by separating logic and resources of the target web service. The last method enables run-time resource management by determining non-functional requirements. These methods are verified through a prototype, making management of the service lifecycle easier.

Key Words : Service Design, Service Lifecycle Management, Product-Service Systems, Cloud Computing

1. 緒 言

記録媒体など計算機資源の急激な低価格化とネットワーク環境・CPUの高機能化などを背景に、近年、データセンタに豊富で高性能なハードウェア（以降は HW）が集約されるようになった。また、オペレーティングシステム（OS）・ストレージ・ネットワーク（以降は NW）の仮想化技術の進展により、データセンタに置かれたソフトウェア（以降は SW）及び HW 資源を必要な期間、必要な量だけ切り出して、ネットワークを通じて利用することが容易になった。これらの遠隔に置かれた計算機資源はクラウド⁽¹⁾と称され、そのインタフェースを利用して、企業向けおよび最終顧客向けの Web サービスの設計・構築・運用・管理がなされるようになる。システム構築者は、Web サービス提供者の要件を受けてクラウド事業者から提供される計算機資源の「機能」を組み合わせる Web サービスを構築するため、設計・構築が試行し易くなる。この開発形態では、クラウドから提供される、

* 原稿受付 2012 年 4 月 2 日

^{*1} 正員, 日本電気(株) 情報・ナレッジ研究所 (〒211-8666 神奈川県川崎市中原区下沼部 1753)^{*2} 正員, 首都大学東京大学院 システムデザイン研究科 (〒191-0065 東京都日野市旭が丘 6-6)^{*3} 正員, フェロー, 首都大学東京大学院 システムデザイン研究科

E-mail: s-hosono@bu.jp.nec.com

実装済みの SW やミドルウェア (以降は MW) を多く利用することから、開発期間が短くなることが期待される。しかしながら、Web サービスを構成する SW を動作させるために必要な実行資源 (MW, HW および NW) は、クラウド事業者委ねられているため、システム構築者は、その状態を直ちに把握し難い。そのため、Web サービスの実行資源を含めた設計に困難さをもたらす。一方、Web サービスの運用時には、処理トランザクション数など業務レベルの品質要件の維持に困難さをもたらす。これは、クラウド事業は、HW および NW 資源を提供・管理する事業¹を母体とするため、従来より業務レベルの要件を直ちに得る手段を持っていないためである。このように、従来、企業内のインハウス・システムとしてスクラッチから設計・構築・管理を行ってきた Web サービス²と、前提や実現構造が異なるため、新たな設計方法や管理方法が求められている。

そこで、本論文は、設計学およびライフサイクル工学に基づいて、クラウド環境に対応した Web サービスの設計および運用に必要となるモデリング方法を示し、その 1 ライフサイクル全体をシステム構築者およびクラウド提供者が管理可能にすることを目的とする。

本論文の構成は、次の通りである。2 章で、クラウド環境における Web サービスの全体構造 (サービスシステム) に生じる課題を分析する。3 章で、このサービスシステムにおいて、設計・管理方法と、ライフサイクル管理を実現するアーキテクチャを提案する。続いて 4 章で、このアーキテクチャに基づくツールの試作を示し、具体化する。5 章で評価を行い、6 章で議論をまとめ、今後の展望について述べる。

2. クラウドの浸透に伴うサービスの構造変化

2・1 SW・HW 製品の機能展開

クラウド事業者は、そのデータセンタにある計算機資源を、IaaS (Infrastructure as a Service), PaaS (Platform as a Service), SaaS (Software as a Service) といったクラウドサービスとして提供している。これらのクラウドサービスは、従来、個々の製品として提供されていた HW, SW や業務 AP を「機能」に展開し、再構成してサービス提供したものと言える。つまり、IaaS は、HW や OS の機能を、PaaS は更にデータベースやアプリケーション (AP) サーバなどミドルウェアを含めて機能を再構成したものを見做せる。同様に、SaaS は IaaS および PaaS の基盤の上に、業務 AP が動作するように、機能を再構成し、そのインタフェースをサービス提供したものである (図 1)。

Fig.1 Recomposed Software/Hardware Functions

これらのクラウドサービスは、従来、Web サービスを構成してきた SW・HW 製品の一部を代替しつつある。結果、Web サービス全体は製品とクラウドサービスが組み合わさったシステムとなる。このシステムを理解し、Web サービス構築に利用するには、製品とサービスがそれぞれ備える「機能」を中心に、機能とその提供関係を捉えることが適切である。この考え方は、製品とサービスから成るシステム (Product-Service Systems; PSS) ⁽²⁾と

¹ Web サービスを実行するための HW 資源をデータセンタ内の HW 資源から貸し出す、ホスティングサービスなどがある。

² 本論文では、インタフェース仕様が規定された狭義の Web サービスではなく、企業向け・個人向けに提供される Web アプリケーション全般を「Web サービス」と総称する。

して、製造業の事業を主な対象として議論されてきた。本論文は、この考え方に基づいて、設計方法およびライフサイクル管理方法を提案する。

2・2 機能担体の所有権

Web サービスシステムを俯瞰すると、Web サービスを構成する SW・HW は、従来、Web サービス提供者により全て管理されてきた。一方、クラウド環境を利用した Web サービスは、SW・HW 機能の実体がクラウド事業者管理される。このように、クラウド環境の浸透に伴い、SW および HW 機能の担手（担体）の所有権に変化が生じる。この Web サービスシステム全体は、AP SW（以降は AP）を開発し実行資源の準備を行う「システム構築者」、システム構築者に Web サービス開発を発注し最終顧客に提供する「Web サービス提供者」、Web サービスを収容する「クラウド事業者」、および Web サービスを利用する「最終顧客」の関係で示せる（図 2）。

Fig.2 Ownership of Traditional and Cloud-based Web Services

この変化を開発プロセスに沿って詳細に見ると、従来、設計・構築工程では AP（担体）と、AP の実行資源（担体）はシステム構築者に所有権があるが、運用中には、これらは全て Web サービス提供者に所有権が渡される。一方、クラウド環境を利用した場合には、開発中の AP はシステム構築者にあるが、クラウド環境への配置時に、AP は Web サービス提供者に所有権が渡され、AP を動作させるための実行資源はクラウド事業者委ねられる。結果、クラウド事業者は、クラウド環境に異なる Web サービス提供者の AP を収容するが、AP の実行資源の状態は、直接 AP の挙動に影響するため、AP の要件に応じて資源を制御・管理する責務が生じる。

従来の Web サービスの構築では、開発と運用の役割が明確に分離されており、双方の部門が連携し、協調作業することは見られなかった。しかし、クラウド環境での Web サービス構築を期待通りに短期間に行うには、必然的に設計と運用がより密接に結びつくことが求められる。しかしながら、前述した所有権の変化によって、設計時に必要となる実行資源の状態を得難く、また、運用時に設計情報を得難いままであるため、設計・構築作業と運用作業に重なりが生じる部分（図 3 の円形）において、双方に情報連携し易くする仕組みが必要となる。以降、Web サービスの 1 ライフサイクルにおける設計と運用の障壁について考察を進めていく。なお、クラウド環境では、複数の AP に対して AP 実行資源が永続的に再利用されるため、終末工程（End-of-life）における AP の廃棄の判断や、廃棄した際の AP 実行資源の再利用方法が議論と対象になり得る。本稿では、特に 1 世代のライフサイクルの完結性を議論するため、これらの終末工程におけるサービス管理は扱わないこととする。

Fig.3 Lifecycle Phases

2・3 ライフサイクル上の課題と既存研究

前節の考察により、Web サービスの設計時点において、実際に活用できる実行資源をモデル化し、設計対象に含められる手段が必要とされる。また、Web サービスの運用時には、業務レベルでのサービスレベルを保証するため、業務要件に基づいて実行資源を管理する手段が必要とされる。

Web サービスの設計や管理に関する従来の研究は、サービス指向アーキテクチャ (Service-Oriented Architecture; SOA) に基づくアプローチで、主に Web サービスのインターフェース設計に観点が置かれていた。この領域では、Web サービスの業務ロジックとその実装方法及びシステムアーキテクチャを中心に議論されてきた。近年は、より上位概念の業務モデルや複数のサービス間の関係を考慮したシステム構成方法にも、議論が広がられている⁽³⁾。しかしながら、クラウド環境によって表出した設計や管理の課題は、従来の取り組みでは扱われてこなかった。

一方、著者らは、開発支援環境として、リポジトリを介して Web サービスの設計と運用を間断なく支援するフレームワークを提案した⁽⁴⁾⁽⁵⁾ (付録)。本フレームワークは、各工程において設計者および運用者の作業を支援するツール群を網羅的に提供している。しかし、工程間の情報連携方法が不完全で、前記の設計と運用に生じた困難さを解消するに至っていない。

そこで、本稿では、設計、実装、検証、運用の各工程において、実行資源を含めた機能担体のモデリング方法を具体化する。これは、Web サービスを構成する「AP」と「実行資源」をそれぞれモデル化するものである。その結合と分離によって、実行資源を含めた設計と、要件を考慮した運用監視・管理を可能にする。

3. サービス設計およびライフサイクル管理方法

設計・構築工程においてクラウド内の実行資源の構成をモデル化し、運用工程において、AP の業務要件をモデル化し、それぞれ設計・管理可能にする。設計・構築工程と、運用工程の環境において、これらのモデル化対象は、図 4 に灰色矩形で示す領域である。このモデルを用いて、(1) 実行状態に基づいて要件定義を行うこと、(2) AP コンポーネントに実行資源を割り当てる設計を行うこと、(3) 機能検証と、クラウド環境の実行資源を用いて非機能検証を行うこと、(4) 非機能要件に適合しているか監視・維持すること、の 4 つの設計・管理方法により、ライフサイクルを管理する。次節より、モデル化と、それぞれの方法および活用場面を述べる。

Fig.4 Bridging Design and Operation Environments - Concept

3・1 実行資源の状態に基づく要件定義方法

Web サービスの開発を始める際、まず、Web サービス提供者が最終顧客からの機能要求 (Functional Requirements; FR) と、性能、保守性、拡張性、セキュリティ等の非機能要求 (Non-Functional Requirements; NFR) を理解し、システム構築者に要求事項を提示する。システム構築者は、クラウド内の資源の振舞いを考慮して、実現可能な要件として Web サービス提供者と合意して、開発作業を進める。

クラウド環境では、クラウド事業者が豊富な物理資源を、利用者毎に仮想資源として分割提供するが、その内部構成は他の AP の HW 資源やインタフェース等を一部共有している場合がある。結果、実際の実行資源の性能諸元は、他の仮想資源の影響を受け、クラウド内の実行資源の状態や振舞いは、外部仕様の初期値や期待値と異なることがあり得る。例えば、複数の利用者の AP が同時にストレージに書き込み処理を行う場合、資源競合が起これ、それぞれの AP の動作が遅くなる等の事象が発生し得る。そのため、CPU コア、ストレージなどの資源の論理的な設計を行っても、実際に配置、実行して実行資源の状態や振舞いの確認が必要となる。

そこで、これらの実行資源の状態をモデル化するために、実行資源を監視するモニターを導入し、システム構築者が、要件定義を行う際、実行資源の振舞いや利用可能な性能範囲を参照可能にする（図 5）。ここで、非機能要件の項目とそのレベルに対して、対応する資源群とその性能値のルールを予め定義しておく。例えば、Web サービスの性能としてレスポンス時間に関する要件は、CPU の性能・コア数、ネットワーク帯域、記録媒体への I/O 時間などの資源の組み合わせとしてルール化する。このルールを用いて、時点における利用可能な資源範囲を動的に更新表示する資源カタログを実現する。このルールは、過去の資源割り当て実績に新規実績を反映させ、継続的に更新し、資源カタログの有効性を定常的に保つものである。

これにより、設計時において、実際にクラウド環境で利用できる資源量や、その状態を確認できるため、要求事項から具体的な要件への擦り合わせが容易になる。よって、達成不可能な要求をそのまま要件として定義することが無くなり、AP の実装・運用工程で生じ得る実行資源の制約を、予め設計工程で考慮できる。そのため、運用工程に入り AP を実行資源に配備した段階で実行資源の不足が判り改めて設計変更を行うなど、手戻り作業も低減し得る。更に、ここで定義した要件はリポジトリに蓄積し、後の運用工程で非機能要件の充足を確認する際に利用できるようにする。

Fig.5 Adjusting Requirements within Available Resources

3・2 AP の機能担体および実行資源の割り当て設計

次に、要件を実現するための業務ロジックおよび実行資源の設計を行う。クラウド環境の要素技術である仮想化技術は、サーバやストレージなど実行資源を仮想化し、論理的に扱えるようにしており、システム構築者の設計の柔軟さを与えている。しかし、これらの仮想サーバ/ストレージは自由に設定可能なパラメータを多く持つため、設計対象とその組み合わせの自由度を増すこととなった。そのため、従来、AP の業務ロジック設計・実装のみであった、システム構築者の責務は仮想サーバや仮想ストレージなどの実行資源の最適設計まで含まれることになった。この拡大した設計作業を支援するため、公理的設計⁽⁶⁾、DSM (Design Structure Matrix)⁽⁷⁾⁽⁸⁾⁽⁹⁾、および DfX (Design for X)⁽¹⁰⁾の考え方を基礎に、次のように設計プロセスを単純化する⁽¹¹⁾。

設計プロセスの単純化

Web サービスと合意した機能要件や非機能要件を実現するために、AP を構成する SW 機能や、それを実行させる HW 機能の設計を行う。クラウド環境を利用した AP の場合、仮想化された HW 機能を実 HW に割り付ける作業までが必要となる。この設計プロセスは、公理的設計におけるドメイン間の転写関係に対応付けられる。公理的設計では、機能要求 (FR) を設計パラメータ (Design Parameter; DP) に転写し、更にプロセス変数 (Process Variable; PV) に転写する関係で設計プロセスを単純化する。この考え方を拡張し、性能、保守性、安全性、可用性、移行性などの非機能要求 (NFR) 群も同列に扱うことでドメイン間の関係を包括して整理できる（図 6 a）。これにより、機能要件、非機能要件から実行資源の割り当てまでの設計手順を明確化できる。ここで、開発した Web サービスは、いずれも更改していくことが予見される。そこで、従来のサービス指向アーキテクチャ

(Service-oriented Architecture, SOA) の考え方が、システムアーキテクチャを構成する SW コンポーネントの独立性を高める指針を示しているのと同様に、モジュール間の独立性を高くする観点を含める。これは、独立性が低い場合には、更改する際に、その依存関係の多さから部分的な置換が困難になるためである。

以上に述べた設計ドメインの抽象化と観点に沿って、設計～実行までのプロセスを、次の3つのステップで単純化する (図 6 b)。

Step1. SW コンポーネントの最適な粒度 (機能群の分割) 設計を行う

Step2. 前記 SW コンポーネントの仮想資源 (仮想サーバ群) への最適な割り当て設計を行う

Step3. 前記仮想資源 (仮想サーバ群) の実体サーバへの最適な割り当て設計を行う

(a) Axiomatic Design Approach

(b) Production Steps based on Axiomatic Design

Fig.6 Production Steps

ここで、Step1, Step2 はシステム構築者に必要な情報で論理的な検証の範疇である。一方、Step3 はクラウド事業者に必要な情報で物理的な振舞いの検証である。本節および次節でそれぞれについて述べる。

Step 1. 機能担体の最適設計

前項に示した各ステップにおいて、AP を構成する SW コンポーネントの粒度や、SW コンポーネントを収容する仮想サーバの台数、また、物理台数の台数など、各機能を担うモジュール (担体) を同定する必要がある。DSM を適用することで、機能やモジュールの依存関係をモデル化し、機能担体間の独立性を確保した設計が可能となる。DSM は、縦軸と横軸に機能名やモジュール名を並べた配列を用い、依存関係のある交差点にマークを並べた配列を用い、依存関係がある交差点にマーク付けを行う。配列の対角線に沿ってマーク付けされたクラスタが多く並ぶように再配置することで、機能群を束ね、機能独立性の高いモジュール構成を見出せる。

SW コンポーネントとその構成要素である API (Application Interface) は、機能担体の機能と等価と見做せるため、この DSM を用いて、適切な粒度の SW コンポーネントを設計する。

Step 2. 非機能要件を考慮した実行資源の割り当て設計

次に、この機能担体を実行資源に割り当てる。このとき、機能担体の独立性だけでなく、応答性能やデータ管理上のセキュリティなど複数の非機能要件を合わせて考慮する必要がある。これは、非機能要件の実現は、実行資源に依存しているためである。そこで、DfX の思考を元に DSM を拡張し、複数の設計観点を同時に考慮できるようにする。この DSM には、配列の行・列の交点に機能の呼び出し関係の有無を記載する。行・列を入れ替えることで、対角線に沿ってクラスタ群を形成し、良い設計であるかを判断する。ここで、各交点は 0~1.0 の中間値を用いる。これは、機能間の直接呼出しと、間接的な依存関係を統合表現したものである (図 7)。ここでは、API など機能担体 (モジュール) のインタフェース (機能) 呼び出しを行う関係を直接的な依存関係とし、インタフェース間の非機能要件の繋がりは間接的な依存関係とする。これらの直接および間接的な依存関係の総和が 1.0 になるように、配分設定を行う (表 1)。表 1 の例では、直接的依存と間接的な依存にそれぞれ 0.5 ずつ配分している。間接的な依存は、更に性能と、安全性の 2 つの非機能要件にそれぞれ 0.3 と 0.2 を配分している。例えば、2 つの機能間に、機能依存と安全性の依存関係があるとき、合計の 0.7 を重み付けされた値として DSM の該当セルに配置する。これを対角線上に合計値が最大となるように並べ替えを行う (図 7)。従来、非機能要件の設計への

反映方法は設計者の経験や主観に依存していたが、本手法により、モデリングのデータとして定量的に扱えるため、非機能も考慮したモジュール群の実行資源への割り当て方を客観的に評価できる。

以上により、Web サービスの機能担体として、「AP」と「実行資源」のそれぞれを設計する手順が具体化された。これにより、開発中の Web サービスが機能および非機能要件の独立した検証も可能になる。次節に、その方法を示す。

Table 1 Dependency of DfX

	Importance	Module 1
Functional Dependency	0.5	-
Performance	0.3	Func1, Func2, Func3
Security	0.2	Func1, Func2, Func3

	1	2	3	4	5	6	7	8	9	10
1		1.0	0.3							0.5
2	1.0		0.3						1.0	1.0
3	0.3	0.3			0.7				0.7	0.7
4					0.7	0.6	1.0	0.6		
5			0.7	0.7		0.7	0.3	0.3		
6				0.6	0.7		1.0	0.6		
7				1.0	0.3	1.0		1.0		
8				0.6	0.3	0.6	1.0			
9	0.6	1.0	0.7							0.6
10	0.5	1.0	0.7						0.6	

Fig.7 Extended DSM

3・3 実行資源の分離と結合による機能・非機能検証

クラウドでは大規模データを処理し易い豊富な HW 資源を持ち、それらの活用を前提とした Web サービスの機能要件も多い。このような要件に対応するため、実行資源には、分散並列処理機構や大規模ファイルシステムなどのミドルウェアも必要とされる。このような実行資源を要する AP を検証するため、システム構築者の開発 PC 内に実行資源を構築することはコストが大きく、実行性が低い。また、非機能は実運用環境に挙動が左右されるため、実環境で検証する必要があるが、機能は仮想的な実効環境で評価できるため、機能検証を行うためのスタブ機構（ツール）を構築者の手元の PC（ローカル PC）に導入し、機能と非機能の検証範囲を分ける。このスタブ機構は、クラウド上の Web サービス実行資源（担体）をシミュレーションするもので、この上で機能の動作検証を可能にする。これにより、実行資源（担体）を Web サービスの設計時に「分離」でき、システム構築者は業務ロジックである AP のみを検証できる（図 8）。以上により、AP の試作中に、AP プログラムの更新の度に、遠隔にあるクラウド環境に再配置・再設定する検証準備が減り、AP 検証効率の向上効果も得られる。

Step 3. 実行資源の実体への割り当て

AP の機能検証をローカル PC で行い、次に、この AP をクラウド内の資源に割り当てて配備する。物理サーバ（実体）上に複数の仮想サーバを配備する際には、仮想サーバ上で動作する複数の Web サービスが、同じ HW 資源や NW 資源にデータを書き込むなど、仮想サーバ間の影響が生じる。ここでも、同様に、非機能要件を考慮した DSM により、実体サーバに割り当てる仮想サーバ群を同定する。仮想マシンを物理サーバに実際に割り当てた後、仮想サーバ上の Web サービスの非機能要件のみを検証する。

Fig.8 Functional/Non-Functional Test Targets

3・4 非機能要件に基づく実行資源の監視

Web サービスの非機能要件は、業務観点の性能や可用性要件として、トランザクション数や 24 時間の稼働時間等が指定される。これらは、機能群に加えて、CPU やネットワーク、記録媒体の書き込み速度など、複数の計算機資源の組み合わせで実現される。特に、これらの業務レベルと資源レベルの対応関係を定式化し、サービスレベルから監視レベルへの変換機構を用意する。例えば、性能要件に対して、CPU 使用率や、通信トラフィック数などの組み合わせとその範囲値に変換する。これにより、運用開始時に、リポジトリに蓄積された非機能要件レベルから資源レベルの監視対象・監視値 (Service Level Objectives; SLO) に変換し、実際に割り当てる資源属性を決定する。同時に、SLO に対する監視を開始する。以上により、再び AP のロジックと実行資源を「結合」して、Web サービス提供者およびクラウド事業者は、非機能要件を満たす実体への配備と、運用中に業務レベルの非機能要件を順守しているかを検証できる。この結果、Web サービスの本番運用までの移行作業が軽減し、最終顧客に対して Web サービスの提供も早められる。

以上に示した 4 つの方法を実行するために、具体化した支援環境のアーキテクチャを図 9 に示す。

Fig.9 Bridging Design and Operation Environments – Architecture

4. 設計およびライフサイクル管理支援ツールの試作と評価

前節に示した方法とアーキテクチャを、(1) 要件定義ツール、(2) 統合設計ツール、(3) 機能検証ツール、(4) クラウド監視ツールにより具現化し、その実現可能性を評価する。図 9 の DevOps Repository は、開発工程と運用工程の情報を統合して扱うリポジトリで、設計と運用環境を橋渡しする役割を果たす。このリポジトリを介して、3 節で示した 4 つの方法が連動し得る。次節より、それぞれの実現方法を示す。

4・1 資源カタログを利用した要件定義ツール

要件定義ツールは、業務のゴール、非機能要件、機能要件を構造化する (図 10 a)。この構造化は、ゴールを実現する非機能要件と、それを実現する機能要素の関係で定義する⁽¹²⁾ため、多段の木構造で表現される。この構造化では、機能 (実行資源を含む) は何らかの非機能要件に関連があるものとし、全ての機能が 1 つ以上の非機能要件に関連づけて定義する。この構造化によれば、アジャイル型で短期に繰り返し開発³を進める場合、非機能要件を見直す際に、関連する機能群を同定できるので、開発に影響する範囲を特定し易くする効果も得られる。

³ アジャイル型の開発は、プロトタイプ設計・実装を行いながら新たに要件を獲得し、必要最小限の要件でシステムを短期に開発するプロジェクト管理方法。短期開発を前提とした開発でも用いられるため、クラウド環境での中小規模の Web サービス開発と親和性が高い。この開発は、大規模システム開発向けに適したウォーターフォール型の開発と対比される。

この要件定義ツールを利用して、可用性、性能、保守性、受容性、安全性、セキュリティ、環境などの非機能要件項目を確定させる際、対応する実行資源の組み合わせと、その諸元値（範囲）を表示するカタログ（図 10 b）を表示する。これにより、クラウド環境の状態に適合した、実現可能な要件定義が可能になり、後の工程で実行資源の実際の振舞いに合わせて要件定義の見直しを行う必要がなくなり、手戻りリスクの発生も低減できる。

Fig.10 Requirements Modeling and NFR-Resource Dynamic Catalogue

4・2 機能担体および実行資源への割り当て設計を支援するツール

統合設計ツールは、AP ロジックを GUI コンポーネントで記述し、3.2 節に示した AP の機能担体および仮想サーバや物理サーバなど実体への割り当て設計方法を支援する（図 11 a）。従来の開発ツールは、AP のロジックを実装する範囲に留まっていたが、このツールでは実行資源への割り当てまで一貫して行える。

拡張 DSM は、3.2 節に示した方法を実装したもので、この設計ツールの一部として機能する。機能モジュール間の依存関係を評価するのに用いる。配列の行・列を入れ替えることで、対角線に沿ってマークのついたセルから成るクラスタを作成する。システム構造の柔軟性を確保するため、良いモジュールの設計戦略として、多くのセルが対角線上に集め、かつ、各クラスタ内の数値の合計が大きくなるように並べ替えを行う。ここで、各セル

の値は中間値を持つことから、遺伝的アルゴリズムを用いた並べ換え方法⁽¹³⁾を導入することで、並べ替え戦略の実行を自動化する⁽¹¹⁾⁽¹³⁾⁽¹⁴⁾⁽¹⁵⁾。その並べ替え結果の提示(図 11 b)により、設計判断を支援する。

以上により決定された AP のロジックと、実行資源の設計情報を用いて、実装し、検証を行う。

(a) Software-Resource modeling tool: *Resource Combination Design*

design evaluation

auto realignment by GA

(b) Design evaluation function: *Extended DSM*

Fig.11 Software and Resource Modeling

4・3 開発環境での機能検証環境とクラウド環境での非機能検証環境

機能検証ツールは、モデルベースの AP 実装・機能検証環境である。クラウド環境では、AP の扱うデータが実行中に急激に増加することがある。そのため、これらに対応すべく、このツールは、Web サービスのフロントエンドと、大規模・並列処理を行うバックエンドを分け、メッセージングにより非同期処理を行い易い WebRole, Queue, WorkerRole, Actor, Cache などの SW コンポーネントを予め用意する。このツールは、AP のソースコードをコンパイルし、得られたバイナリファイル (AP プロトタイプ) を実行するための実行資源を備える (図 12 a)。このクラウド内の実行資源のシミュレーション環境で、ローカル PC 上で機能要件の検証が可能となる。特に大規模データ処理は、ローカル PC の HW 資源の制約からシミュレーション環境を再現できないため、一部をクラウド側の実行資源で代替処理し、見かけ上ローカル PC で機能検証を完全に行えるようにする (図 12 b, c)。クラウド環境に SW コンポーネントをローカル PC からアップロードして配備し、初めて実行検証できる環境に比べて、パッケージング・アップロード・配備というステップが不要となり、開発手順も簡略できる⁴。

(a) Overview of AP prototyping

(a) Client-side development tools

(b) Cloud-side execution environment

Fig.12 AP Prototyping Environments for Functional Test

次に、性能などの非機能要件の検証は、AP を実行する場所の構成や状態に依存するため、クラウド環境で行う。これは、統合設計ツールで作成した AP の SW コンポーネントと実行資源の構成情報を元に、AP の配備先と

⁴ クラウド事業者によっては、AP 実装工程でのクラウド環境の占有期間や、AP のアップロード時の通信量に対して課金する。そのため、設計～実装工程において、クラウド資源の占有や、通信を伴うステップが減ることは、システム構築者およびサービス提供者に有利となる。

なる仮想サーバを作成する。次に、前述の機能検証済みのバイナリファイル（AP プロトタイプ）を仮想サーバに配備して、非機能検証を行う（図 13 a）。

Fig.13 Cloud Environment for Non-functional Test

4・4 非機能要件の適合監視ツール

クラウド監視ツールは、クラウド AP を実行するためのミドルウェア群（実行スタック）を制御し、仮想サーバの生成や停止などを行う。これは監視ライブラリを埋め込んだ仮想サーバを監視し、仮想サーバ／仮想マシン（Virtual Machine, VM）の振舞いや、性能を得ることができる（図 13 b 右）。また、要件定義時に指定された非機能要件に対して、適切な監視対象・監視値（SLO）の組み合わせと値域の候補を提示する。これにより、クラウド管理者は、AP のロジックレベルの要件を実行資源レベルの項目に置き換えて、監視設定を行える（図 14）。

また、このツールは、資源カタログデータベースを備える（図 13 b 左）。このカタログデータベースには、仮想サーバの CPU、メモリサイズ、ストレージボリューム等の仕様や、実行状態に基づく性能値が保管される。また、非機能要件に対応する実行資源の組み合わせルールを保持する。このデータベースとルールは、4.1 節の資源カタログと連携し、クラウドの運用から設計へ情報をフィードバックする。

Web service requirements

SLOs to meet requirements

Fig.14 Cloud Monitor Architecture and SLO Sets for NFR

5. 設計支援およびライフサイクル管理ツールの適用評価

5・1 適用評価

前節までに示した試作ツールをクラウド環境に適用し、その有効性を議論する。事例 1 として、ソーシャルネットワークサービス (SNS) に逐次書き込まれたデータを用い、Web ブラウザに動向分析した結果を表示する Web サービスと、事例 2 として、展示会などイベント向け広報ブログサイト (Web サービス) を構築した。事例 2 を例に、4 つの方法に沿って Web サービス提供までの流れを図 15 に示す。

Fig.15 Design and Operation Steps of a Cloud-compliant Web Service

事例 1 の SNS 分析サービスは、大量に書き込まれたコメントを定期的に収集し、利用者からの要求に対してコメント内容の傾向から動向を分析し、その結果を Web ブラウザに描画する Web サービスである。この Web サービスは、大量のコメントデータを解析する機能要件と、解析結果の応答を短時間に行う非機能要件が特徴である。「要件定義ツール」を用いて、機能目標として、データの収集、累積分析、期間ランキング、分析結果の描画を抽出し、更に、その実現機能として、リクエスト受付、データ読み込み、キーワードカウント、集計、データ保管、描画機能を構造化した (cf. 図 10 a)。更に、10MByte の想定コメントデータに対して、リクエスト受付～描画までを 10 秒以内に行う性能要件 (非機能要件) に対して、割り当て可能なメモリや CPU などの資源の組み合わせ候補が資源カタログに表示され (cf. 図 10 b)、この範囲において要件定義および設計値を確定した。次に、「統合設計ツール」の拡張 DSM に、各 AP の各機能を配置し、規定時間内にリクエストを処理する性能要件と、データ保全に係る安全性要件と非機能要件である性能要件を DSM に加味した。DSM の並べ替えの結果、キーワードカウント・集計を行うモジュールと、リクエスト受付・データ保管・描画を行うモジュールが分離された (cf. 図 11)。これに基づき、各モジュールを独立した 2 台の仮想サーバ a と b に割り当てることとした。ローカル PC の「機能検証ツール」を用いて AP の各機能の実装・機能検証を行い、各仮想サーバに配備した。

運用にあたり、「監視ツール」で、前述の性能要件が、ストレージへのアクセス時間や通信頻度、セッション数・時間などの監視項目の目標値にマップされた。運用試行において、処理対象期間のデータが倍増したため、処理性能目標値 (性能要件) を超える可能性が要件に加わった。そこで、データ保管に関わるモジュールの仮想サーバ b は、そのままとし、データ解析に関わるモジュール仮想サーバ a を 1 台追加し、並列構成とした。

本事例では、3 章、4 章に示した 4 手法のうち、次の効果が顕著であった。(1) クラウドの実資源に基づく要件定義、(2) AP の機能担体および実行資源の割り当て設計により、クラウド環境の資源をモデル化したことで、AP 設計とその配置設計が容易になり、設計工程の作業を削減できた。また、この適切なモジュールの分割と、(4) 非機能要件に基づく監視により、運用中の新たな要件にも部分構成の強化のみで対処し、Web サービス全体の構造見直しなど膨らみがちな保守・改善作業を局所化できた。

事例 2 のブログサイトは、展示会の広報・集客用途として、イベント開催者と参加者が情報共有を行う Web サービスである (図 15 右)。この Web サービスは、Web ブラウザとスレート端末の 2 種類のユーザインタフェースに持つ機能と、また、展示会の前後の短期間のみサービス提供するため、ごく短期間に Web サービスを構築することが要件の特徴である。「要件定義ツール」を用い、AP サーバを機能目標として構造化し、リクエスト受付、クライアントの端末種別判定、端末種類別の描画機能を構造化した。次に、「機能検証ツール」を用いて、ローカル PC 環境で AP を実装し、2 種類のユーザインタフェースに対応した描画結果の出力シミュレーションを行った (cf. 図 12 a)。この後、クラウドに配備し、「監視ツール」で応答時間が著しく遅延していないことを確認し、運用を開始した。

本事例では、(3) 実行資源の分離と結合による機能・非機能検証、による効果が顕著であった。2 種類のユーザインタフェースの実装作業において、システム構築者は実装中のプロトタイプの挙動確認に、機能検証ツールを用い手元の PC の資源のみで検証した。検証するまでに、遠隔のクラウド環境へ実行ファイルをその都度、送信・配備・動作準備を行うステップが無いため、検証工程の作業を削減できた。

以上のように、設計と運用工程間の情報連携が進み、設計～運用工程の作業が効率化されたこと⁵、また、設計～運用までのライフサイクル管理を首尾一貫して行えることを確認できた。

5・2 考察

前節に示した開発の効率化は、本提案のモデリング方法および 4 つの提案方法により得られた。これらの方法は、図 16 に示す通り、リポジトリ (図 16 中央) を中心に、一連の 1 つのフレームワークとして機能した。この機構は、設計工程と運用工程の情報共有や共同作業の基礎となり、設計と運用がより一体化する DevOps⁽¹⁶⁾の考え方を実践・実装するための機構とも言える。

本適用評価では、1 つの Web サービス開発を想定し、リポジトリを開発環境内に配備し、システム構築関係者間で共有した。このリポジトリを、クラウド環境の仮想サーバ内に配置し、Web サービス案件毎に確保・隔離することで、Web サービス案件情報の安全性を担保した開発・実行フレームワークをクラウド環境に構築できる。このような仕組みを備えることで、クラウド事業者は、設計～運用までが一貫した、Web サービス設計・運用フレームワークをサービス化 (Application Platform as a Service; APaaS⁽¹⁷⁾) して、高機能かつ高付加価値のクラウドサービスの事業化が可能になる⁶。

Fig.16 Bridging Design and Operation Environments

6. 結 語

本論文では、クラウドサービスの浸透に伴い、Web サービスのシステム構造に変化が生じ、ライフサイクル上の情報連携が困難になる要因を特定した。また、この情報連携の困難さを解消するため、設計環境と運用環境の

⁵ 実際の SI 担当部門が評価を行った結果、従来のスクラッチから開発する方法に比べ、開発期間を全体で 3 割程度、削減できた。(これは、付録に示すフレームワークとしての適用評価である。) この削減効果は、4 章、5 章で示した開発・運用作業の簡略化によって得られた。

⁶ 実行環境と開発環境も合わせたミドルウェア群 (設計・実行プラットフォーム) が充実し、Web サービスの設計から構築までを簡便にする PaaS (Platform-as-a-Service) である。

双方で設計・運用に関わる情報をモデル化し、連携する方法を示した。実行資源のモデリング方法、割り当て方法、機能・非機能の検証方法、監視方法の4つを示し、これらを具体化する設計・管理支援ツールを示した。これらによって、Web サービスの設計と運用の課題を解決し、設計と運用の情報を循環させて、ライフサイクルを通じた設計～実行管理が可能となった。また、これらの方法の実現により、Web システムの設計～運用までのプロセスが効率化され、Web サービス開発期間を短縮化する効果も得られた。

本稿で議論したクラウド環境は、同じ或いは異なる利用者が繰り返し、同じ計算機資源を継続して使い続ける環境でもある。今後、次の世代の Web サービスに更改していく方法や、現行の Web サービスを廃棄した上、再設計する方法が重要となる。本稿で示した、設計・運用の手順の枠組みにより、設計・運用プロセス自体の資産化も可能となる。今後の研究として、このプロセス資産に含まれるライフサイクルコストを分析・活用することで、次の Web サービスの改善サイクルとして、Web サービスのリユース、リマニュファクチャリング、リサイクルの戦略支援を行うことが考えられる。

謝 辞

本研究の一部は、首都大学東京ならびに東京大学との「サービス工学に関する共同研究」として実施した。また、実証評価は、NEC 中国、NEC ビッグロブの協力を得た。

付 録

著者らは、Web サービスの構築および実行フレームワークを開発した（図 17）。このフレームワークは、上流の要求分析から詳細設計、実装、配備までの開発工程と、配備、実行までの運用工程を支援するもので、次の設計・実装ツールを持つ。アーキテクチャの主要構成を図 18 に示す。

Fig.17 Overview of Framework

- ①サービスの目標を定義する *Service Objectives*
- ②サービス提供までのバリューチェーンを定義する *Stakeholder Comprehension*
- ③目標を実現する要件と、要件を実現する機能を定義する *Goal Prioritization*
- ④機能担体と、機能担体間の関係を定義する *Service System Design*
- ⑤AP SW コンポーネントを定義する *Application UML Modeling*
- ⑥AP のリソースへの配置を定義する *Resource Combination Design*
- ⑦AP のプロトタイピング環境の *Application Prototyping*
- ⑧AP のライフサイクルを通じて設計・運用情報を一元管理する *Service Portfolio*
- ⑨クラウドの実行資源を制御する *Cloud Controller*

本稿では、1 ライフサイクルにおける設計と運用に生じた障壁を解消し、ライフサイクル管理の完全性を備えるために、③⑥⑦⑨および⑧の考察を深めた。それぞれが連携・連動するように、拡張した部分を軸に議論した。

Fig.18 Architecture of Framework

文 献

- (1) Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M., "Above the Cloud: A Berkeley View of Cloud Computing", US Berkeley Electrical Engineering and Computer Sciences, <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf> (参照日 2013年11月5日).
- (2) Baines, T.S., Lightfoot, H., Steve, E., Neely, A., Greenough, R., Peppard, J., Roy, R., Shehab, E., Braganza, A., Tiwari, A., Alcock, J., Anugs, J., Bastl, M., Cousens, A., Irving, P., Johnson, M., Kingston, J., Lockett, H., Martinez, V., Michele, P., Transfield, D., Walton, I., Wilson, H., "State-of-the-art in product service systems", *IMechE, Part B: Journal of Engineering Manufacture*, Vol. 221 (2007), pp. 1543-1552.
- (3) Zhang, L.J., Zhang J., Cai, H., *Service Computing* (2007), Springer-Verlag.
- (4) Hosono, S., Huang, H., Hara, T., Shimomura, Y., Arai, T., "A Lifetime Supporting Framework for Cloud Applications", *In IEEE Proceedings of the 3rd International Conference on Cloud Computing* (2010), pp. 362-369.
- (5) Hosono, S., He, J., Liu, X., Li, L., Huang, H., Yoshino, S., "Fast Development Platforms and Methods for Cloud Applications", *In IEEE Proceedings of the Asia-Pacific Services Computing Conference* (2011), pp. 94-101.
- (6) Suh, N.P., *Axiomatic Design* (2001), Oxford University Press.
- (7) Steward, D.V., "The design structure system: a method for managing the design of complex systems", *IEEE Transactions on Engineering Management*, Vol.28, No.3 (1981), pp. 71-84.
- (8) Eppinger, S.D., "A planning method for integration of large-scale engineering systems", *In Proceedings of the 11th International Conference on Engineering Design* (1997), pp. 199-204.
- (9) Browning, T.R., "Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions", *IEEE Transactions on Engineering Management*, Vol. 48, No. 3 (2001), pp. 292-306.
- (10) Paul, G., Beitz, W., *Engineering Design – A systematic Approach*, 2nd Edition (1996), London Springer.
- (11) Hosono, S., Kimita, K., Akasaka, F., Hara, T., Shimomura, Y., Arai, T., "Toward Establishing Design Methods for Cloud-Based Business Platforms", *In Proceedings of the 3rd CIRP Conference on Product Service Systems* (2011), pp. 195-200.
- (12) Hosono, S., Hara, T., Shimomura, Y., Arai, T., "Prioritizing Service Functions with Non-Functional Requirements", *In Proceedings of the 2nd CIRP Conference on Product Service Systems* (2010), pp. 133-140.
- (13) Hosono, S., Min, M., Kimita, K., Akasaka, F., Shimomura, Y., Hara, T., Arai, T., "Architecture Design and Assessment with Design Matrix", *In IEEE Proceedings of the 7th World Congress on Services* (2011), pp.83-84.
- (14) Albers, A., Sedchaicharn, K., Sauter, C., Burger, W., "A Method to Define a Product Architecture Early in Product Development Using Contact and Channel Model", *In Proceedings of the International Conference on Engineering Design* (2009), pp. 241-252.
- (15) Whitfield, R.I., Smith, J.S., Duffy, "Identifying Component Modules", *In Proceedings of the 7th International Conference on Artificial Intelligence in Design* (2002), pp. 15-17.
- (16) Gartner Research, "DevOps: Born in the Cloud and Coming to the Enterprise", ID Number: G00208163 (2010).
- (17) Gartner Research, "APaaS: A Step to a 'Killer App' for Cloud Computing?", ID Number: G00168152 (2009).