平成 26 年度 博士論文

IT サービスにおける
ライフサイクル管理方法論

首都大学東京大学院

システムデザイン研究科

ヒューマンメカトロニクスシステム学域

博士後期課程 11989505

細野 繁

主査 下村 芳樹 教授

概要

IT 産業においては，ソフトウェア，ハードウェア，ネットワークの製品事業が成熟し，他の産業と同様に，製品自体の販売から，構築や保守など製品に付随するサービスの販売比率が高まっている．近年拡がりを見せるクラウド環境は，更にこのサービス比率を飛躍的に高めようとしている．クラウド環境は，データセンタ上に置かれた豊富なソフトウェア，ハードウェア，ネットワークリソースをクラウドサービスとして提供し，従来の製品および付随サービスの多くを代替するためである．

このクラウドサービスが浸透するに伴い，システム構築サービスに関するビジネスエコシステムにクラウドサービスプロバイダが新たなステークホルダとして加わることとなる．これにより，IT サービスプロバイダ（システムインテグレータ）に求められる役割が変化しつつある．従来，IT サービスプロバイダは，ソフトウェアおよびハードウェア製品を利用して，顧客の案件毎に固有のシステム（Web サービス）を設計・構築してきた．この事業は，システムの実装に価値があるため，ソフトウェア・ハードウェア製品価格と作業担当者の実績工数を元にした事業モデルが成り立つ．しかし，クラウドサービスは予め検証済みのソフトウェア・ハードウェア機能を提供し，IT サービスプロバイダのシステム実装や検証作業の多くを不要にする．また，クラウドサービスは，単位期間の価格を抑えてこれらの機能を提供するため，顧客との契約を継続させることで事業モデルを成り立たせる必要がある．そのため，IT サービスプロバイダは，この事業環境の変化に伴って，これらの機能を組み合わせて，効用の高い Web サービスを設計し，顧客と合意し，設計〜運用担当まで協力して，迅速に開発し提供することが求められるようになる．

この変化は，IT サービスプロバイダおよび顧客の価値の源泉が，システム（Web サービス）を構成するモノの機能性から，相対的にシステム全体の効用へと広がった，と捉えることができる．つまり，IT サービスプロバイダ内で完結できた，システム実装や検証作業から，効用の高いシステムの提案・設計・プロトタイプ検証・改善作業へと，顧客との協働作業が必要なものに遷移した，と捉えることができる．

サービスを「提供者からの行為が受給者に対して何らかの変化をもたらすもの」とする定義に拠れば，前述のクラウドサービスを媒介とした変化は，製品指向の事業モデルから，サービス指向の事業モデルへ変革を促している，と理解できる．すなわち，仕様通りに動作する機能を実装し，それを顧客に納めることに価値を置いた事業モデルは製品販売の指向であるのに対し，顧客と設計〜運用までの工程全体を通じた顧客との協働活動により，顧客に対して価値を生み出し，対価を得るサービス指向に変化している．この変化に対応するため，サービスプロバイダは，サービスシステムのバリューチェーン全体を把握し，

そのライフサイクルを通じて顧客と継続的に協働していくための方法を必要とする．加えて，サービスプロバイダは，特定の顧客に 1 対 1 でサービス提供するだけでなく，多くの顧客にも同時に展開・提供できるように，過去の開発で得られた知見を蓄積し，多くの案件に適用可能にしておくことが求められる．

以上の要請に対して，企画・設計・構築・運用・分析工程を俯瞰し，サービスシステムのライフサイクルを一貫した管理方法論が必要である．これまでのサービス工学研究では，顧客分析，機能設計，提供構造の決定など，サービスライフサイクルの特定の工程に焦点を当てた議論が行われてきた．しかし，提案・設計・プロトタイプ開発・提供までの工程を跨って，知識を蓄積し，それを活用して効用の高いサービスを効率的に生産するプロセスおよびライフサイクル管理の観点では，これまで十分に議論されていない．

以上を鑑み，本論文は，システム／サービスを企画・設計・構築・運用・分析するまでの構造を，多様なステークホルダから成るサービスシステムとして捉え，その**「サービスライフサイクルを管理する方法論を提供すること」**を目的する．具体的には，本論文では，以下に挙げる 3 点を具体的な達成項目とする．

(1) ライフサイクル工程のモデリング方法の確立

本研究では上流～下流までの工程の情報が間断なく連携するモデリング方法を提案する．

ハードウェア製品の製造は，その開始時点で仕様が確定し，生産の各工程での作業や期間が明確化されている．一方，サービスの開発～運用工程には多くの協働作業が含まれ，各工程での作業や期間は，人に依存し易い．分析～企画・設計に至る過程では，顧客と企画者の間で仕様の合意形成が図られ，実装～提供の過程では，設計担当と運用担当の間で設計情報の共有が必要となる．これらのやり取りにおいて，ステークホルダの役割に応じて要求～機能～実体へと変換されていく情報間の差異を捉え，これらのステークホルダ間の共通理解を得ることが必要である．

ここで，クラウドの浸透に伴い，このステークホルダ間の関係において，サービス提供側のステークホルダが多様化することに着目する必要がある．従来の Web サービスは，システムインテグレータが，ソフトウェアおよびハードウェア全体を設計・構築し顧客に提供するものであった．しかし，クラウドを前提とした Web サービスは，設計と運用でシステムインテグレータとクラウドサービスプロバイダの 2 つのステークホルダによって業務ロジック，および実行リソースが独立に管理されるようになる．一方，顧客にとっては，利用する機能はいずれのステークホルダから提供されるかに関わらず，透過的に利用できることが望ましい．そのため，このようなステークホルダの多様化によって生じた，工程間の分離した情報を連結させる方法が必要である．

そこで，実行状態に基づく要件定義，ソフトウェアコンポーネントの実行リソースへの

割り当て，機能検証環境，非機能検証環境により，これらの分断された情報を連結できるライフサイクル工程のモデリング方法を提案する．このモデリングにおいて，DSM（Design Structure Matrix）を拡張し，設計データとして表現や管理し難かった，非機能要件も機能要件と同様にモデルデータとして扱う．これにより，機能とリソースの分離と統合を行い，機能と非機能の検証を可能にする．更に，ライフサイクル全体を網羅するように，ライフサイクル工程を細分化したモデリングを示す．公理的設計の考え方に基づき，要件→機能→リソース割り当ての転写関係により，モデル間の属性関係を定義し，工程を間断なく網羅したモデルチェーンを構築する．

以上のモデリングは，異なる役割を持ったステークホルダ間の共通言語として作用するため，ステークホルダ間の相互認識を深められる．また，上流工程から下流工程までモデルデータとして保持できるため，開発の後工程で上流工程の設計の振り返りも容易になり，システム／サービスの開発を，手戻りなく迅速に進めることが可能になる．

⑵ ライフサイクル知識の資産化方法の確立

本研究では，前述のモデリング方法を活用してライフサイクル全体を包含するデータを構築し，これを別のサービス開発で再利用し易くするための資産化方法を提案する．

（1）で開発した各モデルを元にサービスのライフサイクル全体を網羅した，サービスモデルチェーンのデータ群をライフサイクル知識として構築する．このライフサイクル知識を資産として保持することで，別の顧客に対して同じ Web サービスを開発する際に再利用できる．サービスモデルチェーンには，開発に必要な設計情報が整列されるため，開発全体の作業や成果物の雛形になるためである．しかし，一般に，同じ Web サービスの開発であっても，全ての要件が同じことは稀で，顧客毎にユーザインタフェースや部分的な機能要件の差異が生じる．そこで，このサービスモデルチェーンを有効に多くの案件に適用できるように，カスタマイズによる再利用を想定した知識の蓄積方法が必要である．蓄積する知識は，再利用する際に，より柔軟に要件の差異に応えられること，また，より多くの案件に適用できること，すなわち，マス・カスタマイゼーションを指向した知識の蓄積方法が必要である．

この実現には，多品種少量生産に適したプロダクトラインエンジニアリングの考え方を応用し，対象とする Web の用途やシステム構成などの特性を類型化するドメイン開発と，ドメイン毎にカスタマイズの元になるパタン開発を行う．更に，これらのパタンを通じた情報を一元管理するために，共通リポジトリであるパブリック・サービスポートフォリオを設計する．また，個々のサービス開発案件で，このパタンをカスタマイズして再利用し易くするために，個別案件用のリポジトリであるプライベート・サービスポートフォリオの実現方法を導入する．

これにより，従来，モノの生産に活用されてきた，マス・カスタマイゼーションの概念をサービスの生産においても実践できる．この開発方法は，サービス生産の短期化と同時に，生産の原価低減を達成するもので，範囲の経済性および規模の経済性を得られる．こ

れにより，ライフサイクル知識を蓄積し，それを多くの顧客案件に提供する IT サービス事業を実現できる．

⑶ ライフサイクル知識の構築・活用プロセスの確立

本研究では，サービスの設計～構築までに関わるステークホルダの協働作業において，上記に示した資産化されたライフサイクル知識を有効に活用する方法および実践プロセスを提案する．

効用の高いサービスを効率的に生産するには，サービスモデリングとサービスライフサイクル知識の資産化方法により蓄積された知識を，その利用者の役割とユースケースに沿って活用し易くすることが必要である．そこで，(1) 顧客と IT サービスプロバイダや，開発と運用部門，開発部門と企画・管理部門間などライフサイクル内のステークホルダ間の協働支援，(2) 過去のサービス開発と別のサービス開発の開発部門などライフサイクルを跨ったステークホルダ間の協働支援と，これらの方法を実践させるためのプロセスを示す．

IT サービスプロバイダと顧客企業担当者間でサービス設計を進める際，要求・構造・振舞いなど特定の観点で絞った情報量のダイアグラムで表現し直すことが合意形成に有効である．そこで，サービスモデルチェーンをある観点に基づき動的に再構造化し，ステークホルダが求める表現形式で提示し，顧客と IT サービスプロバイダ間の協働を支援する．

また，開発成果物を資産化する際，その資産の再利用シナリオや資産化の意図を保持できるように，再利用戦略ポリシーとして形式化し，開発部門間の協働を支援する．一方，運用管理の保守性や，改良のし易さなど，実装・実行工程での要件や制約を併せて考慮し，これらの多目的に最も適合する開発資産を活用させることで開発部門間の協働を支援する．

また，モデルチェーンの構築過程において，各モデルに紐付いた開発者の作業量をデータとして定量化する．この定量データを含む設計・運用プロセスのパタン化によって，モノの生産プロセス管理と同様に，管理部門が開発部門のサービス設計・運用プロセスの比較分析・管理を可能とし，開発部門と企画・管理部門との協働を支援する．

以上に示した支援方法を実際の開発に定着させるため，ソフトウェア工学の分野で近年導入されつつある状態で開発プロセスを捉える考え方を拡張し，上記に示した手法を漏れなく行うタスクとして提示する．これにより，開発プロセスを通じて開発者に上記に示した支援方法を実行するように規律を与える．このように，ステークホルダの役割に基づいて，ライフサイクル知識の構築や利用を促進させる．これにより，従来プロセス指向であった開発方法を，タスク基点に捉え直したことで，各ステークホルダと協働したサービス設計～運用が可能となる．

以上により，本論文は，サービスの企画・設計・構築・運用・分析工程に渡るライフサイクル知識を蓄積・活用し，異なる役割を持ったステークホルダが協働し，合理的にかつ効率的にサービスを設計～運用していく方法を示した．

目次

図目次

表目次

第1章 序論

目次

1.1 研究背景

1.1.1 クラウドコンピューティングの浸透に伴うSIサービスの変化

クラウドコンピューティングは，ネットワークで接続された計算機のソフトウェアやハードウェアリソースを，利用者が遠隔から利用可能にする技術の総称である．この技術を活用したクラウドサービスは，利用者にデータセンタなどに置かれた計算機リソースを必要な期間，必要な量だけ提供している．

総務省 スマート・クラウド研究会報告書［総務省 2010］では，ここ 10 数年で急速な普及を遂げてきたインターネットや，それを支えるブロードバンド基盤の構築が進み，ICT（情報通信技術）の活用のあり方が大きく変わろうとしていると考察している．特に，クラウドサービスは，利用者が必要な計算機リソースを必要な時に，必要な量だけサービスとして利用できる従来とは全く異なる情報通信システムの活用策であり，情報通信分野におけるパラダイムシフトが起きつつあることを指摘している．

本報告書の企業等のクラウドサービスの導入意向に関するアンケート調査に基づいた，クラウドサービス市場の規模の推計では，2015 年時点で 4 倍強の約 1 兆 81 百億円になることが見込まれている．市場の年平均成長率は 30.5%と極めて高い．その構成要素を見ると，2015 年時点において SaaS 市場が 62.5 %を占めているが，IaaS（Infrastructure-as-a-Service）市場及び PaaS（Platform-as-a-Service）市場がそれぞれ約 34 百億円まで拡大することが見込まれる，としている．

上記の推計は現時点でのクラウドサービスの導入意向をベースとしたものであるが，行政，医療，教育，農林水産業等におけるクラウドサービスの普及，スマート・クラウド基盤の構築等を政策的に支援することで，クラウドサービス市場は 2015 年時点で 56 百億円程度の新市場の創出が見込まれ，クラウドサービス市場は約 2 兆 37 百億円の規模に達する（約 2 兆円の新市場創出）ものと見込んでいる（図 1-1）．

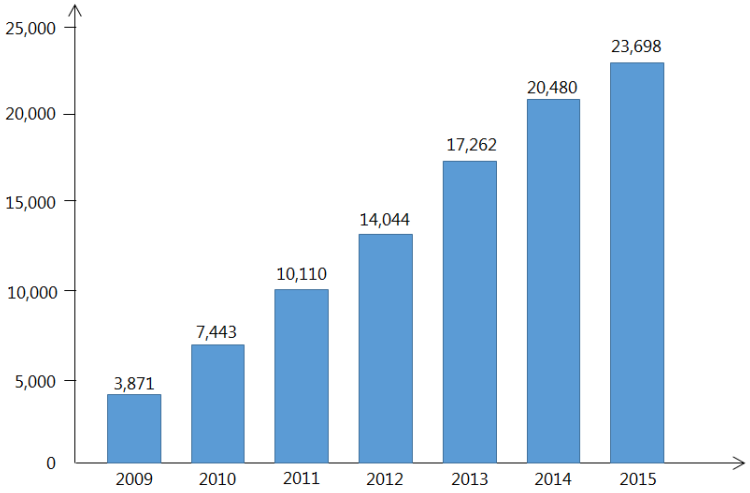

図 1-1 クラウドサービスの事業規模見通し［総務省 2010(改)］

報告書では，更に，従来のシステム開発は，ネットワーク・ハードウェア・OS・ミドルウェア等，それぞれの分野の専門技術者による対応が主であったが，クラウドコンピューティングではシステム全体を把握した上で，全体最適でのシステム設計・開発スキルが求められることを指摘している．

このように，IT 産業の変化は，IT サービスプロバイダの役割に変化を与えつつある．クラウドサービスの浸透は，Web サービスのシステム構築を担う IT サービスプロバイダのシステムインテグレーション（SI）サービス[1]に変化を与えている．従来の SI サービスのビジネスモデルでは，Web サービスを構成するソフトウェア・ハードウェア製品の費用と，Web サービスの実装に要した開発工数に対して，対価を得ていた．このビジネスモデルは請負契約に基づき，Web サービスの実装完成システムと，開発中に作成した機能仕様書やテスト報告書，評価データなど開発成果物を一式顧客に納めている．これは，IT サービスプロバイダ（システムインテグレータ）側には成果物やデータは残らない形態のため，売り切り型のビジネスと見做せる．

しかし，クラウド上に Web サービスを構築することが前提になると，ソフトウェア製品・ハードウェア製品の機能やビジネスロジックを実装したソフトウェアコンポーネントが，クラウド上の PaaS（Platform as a Service）や SaaS（Software as a Service）としてサービス提供されるため，実装や検証作業の多くが不要になる，そのため，従来の SI サービスのビジネスモデルは成り立ちにくい．一方，クラウドから豊富に提供されるソフトウェア・ハードウェア機能を利用できるため，Web サービスをスクラッチから実装するよりも開発作業が容易になる．そのため，IT サービスプロバイダは，顧客の要件に忠実に

[1] 技術者がコンピュータシステムを組み上げる作業をサービスとして提供するもの．なお，本用語を含めた IT サービス固有の用語は，付録 B にまとめて示す．

Web サービスの機能実装や検証を行うことよりも，寧ろ，現状の Web サービスを分析し，よりビジネス効果の高い Web サービスを企画・提案し，クラウドから提供されるソフトウェア・ハードウェア機能のインタフェースを効率良く組み合わせることが求められる．更に，短期間に Web サービスのプロトタイプを開発し，実務で用いる Web サービス実行環境を迅速に提供することが価値となる．このように，クラウドの浸透に伴い，IT サービスプロバイダは，よりビジネス視点や全体システム観点の見識を発揮して，ライフサイクルの上流工程や下流工程においても顧客と継続的に協働して，Web サービスを迅速に企画・設計・構築・提供することが重要である．この協働作業を IT サービス事業の主要なサービスである SI サービスとして確立するには，協働プロセスに対価を支払う準委託契約で行うことが適切と考えられる．しかし，本形態の契約はシステム完成品のような成果目標物が予め決まらず，SI サービスの単価が抑えられがちである．そのため，請負契約から準委託契約に基づく継続的なビジネスへ移行が進みにくい．そこで，IT サービスプロバイダは，開発・実行で得た知見をライフサイクル知識として保持し，それをコンピタンスとして多くのサービス開発に再利用できれば，多くの顧客と継続的な契約関係を構築し，顧客 1 件当たりのサービス価格は低くとも，全体で十分利潤を確保し得る．つまり，ライフサイクル知識を蓄積（ストック）し，それを多くの顧客案件に提供する仕組みを備えることで，売り切り型のビジネスモデルからの移行を促進できる．

1.1.2 本論文の問題設定

以上の要請に対して，企画・設計・構築・運用・分析の工程を通じて，関連するステークホルダの観点の考慮が必要である．これまでのサービス工学研究では，顧客分析，機能設計，提供構造の決定など，サービスライフサイクルの特定の工程に焦点を当てた議論が行われてきた．しかし，提案・設計・プロトタイプ開発・提供までの工程を跨ったステークホルダの扱う知識を包括して蓄積させること，および，この知識を活用して効用の高いサービスを効率的に開発〜運用するまでライフサイクルを管理することに関して，これまで十分に議論されていない．そのため，サービスを提供するまでの構造および提供時の構造を多様なステークホルダから成るサービスシステムとして捉え，ステークホルダが上記の知識を構築・保持・活用できる体系的な支援方法を開発し，このサービスライフサイクル管理方法をサービス開発者に提供することが必要である．

1.2 本研究の目的

本研究の目的は，サービスを企画・設計・構築・運用・分析するまでの構造を，多様なステークホルダから成るサービスシステムとして捉え，その設計～運用までの

「サービスライフサイクルを管理する方法論を開発する」

ことである．具体的には，以下に挙げる 3 点を達成する．

(1) ライフサイクル工程のモデリング方法の確立

上流～下流までの各工程で扱う情報を規定し，これらの工程間の情報が間断なく連携できるモデリング方法が必要である．

ハードウェア製品の製造は，その開始時点で仕様が確定し，生産の各工程での作業や期間が明確化されている．一方，サービスの開発～運用工程には多くの協働作業が含まれ，各工程での作業や期間は，人に依存し易い．分析～企画・設計に至る過程では，顧客と企画者の間で仕様の合意形成が図られ，実装～提供の過程では，設計担当と運用担当の間で設計情報の共有が必要となる．これらのやり取りにおいて，ステークホルダの役割に応じて要求～機能～実体へと変換されていく情報間の差異を捉えて，漏れなく情報を伝達し，これらのステークホルダ間の共通理解を得ることが必要である．

ここで，クラウドの浸透に伴い，このステークホルダ間の関係において，サービス提供側のステークホルダが多様化することに着目する必要がある．従来の Web サービスは，システムインテグレータが，ソフトウェアおよびハードウェア全体を設計・構築し顧客に提供するものであった．しかし，クラウドを前提とした Web サービスは，設計と運用でシステムインテグレータとクラウドサービスプロバイダの 2 つのステークホルダによって業務ロジック，および実行リソースが独立に管理されるようになる．一方，顧客にとっては，利用する機能はいずれのステークホルダから提供されるかに関わらず，透過的に利用できることが望ましい．そのため，このようなステークホルダの多様化によって生じた，工程間で分離して管理される情報を連結させる方法が必要である．

(2) ライフサイクル知識の資産化方法の確立

次に，ライフサイクルを通じた設計～運用知識を資産化し，別のサービス開発で再利用し易くする方法が必要である．

(1) で開発した各モデルを元にサービスのライフサイクル全体を網羅した，サービスモデルチェーンを構成し，そのインスタンスとなるデータをライフサイクル知識として構築する．このライフサイクル知識を資産として保持することで，別の顧客に対して同じ Web サービスを開発する際に再利用できる．サービスモデルチェーンには，開発に必要な設計情報が整列されるため，開発全体の作業や成果物の雛形になるからである．しかし，一般

に，同じ Web サービスの開発であっても，全ての要件が同じことは稀で，顧客毎にユーザインタフェースや部分的な機能要件の差異が生じる．そこで，このサービスモデルチェーンを有効に多くの案件に適用できるように，カスタマイズによる再利用を想定した知識として蓄積する方法が必要である．蓄積する知識は，再利用する際に，より柔軟に要件の差異に応えられること，加えて，より多くの案件に適用できること，すなわち，マス・カスタマイゼーションを指向した知識の蓄積方法が必要である．

(3) ライフサイクル知識の構築・活用プロセスの確立

サービスの設計～構築までに関わるステークホルダの協働作業において，上記に示した資産化されたライフサイクル知識を有効に活用する方法および実践プロセスが必要である．

効用の高いサービスを効率的に生産するには，サービスモデリングとサービスライフサイクル知識の資産化方法により蓄積された知識を，その利用者の役割とユースケースに沿って活用し易くすることが必要である．そこで，(1) 顧客と IT サービスプロバイダ，開発と運用部門，開発部門と企画・管理部門間など，ライフサイクル内のステークホルダ間の協働支援，(2) 過去のサービス開発と別のサービス開発の開発部門などライフサイクルを跨ったステークホルダ間の協働支援，および，(1) (2) の方法を実践させるための支援プロセスが必要である．

1.3 本論文の構成

本論文の構成は，図 1-2 に示す通りである．

第 2 章では，IT サービスの特徴について述べ，本研究の目的ならびに具体的な達成項目を明らかにする．サービスの捉え方を議論し，システムとしてサービスの提供構造を捉える必要性を示す．そして，あるべきサービスシステムの設計と構築および移行に必要となる要件を明確化する．

第 3 章では，上流～下流まで各開発工程におけるモデリング対象を規定し，全工程を網羅するモデリング方法を提案する．サービスモデリングに関する先行研究について概説した後，それらを踏まえた本研究のアプローチについて説明を行う．そして，工程間の分断した情報を統合するモデリング方法を提案する．

第 4 章では，サービスのライフサイクル知識の資産化方法を説明する．(1) サービス設計モデルを間断なく連鎖させたサービスモデルチェーンのデータ構築と，(2) これをマス・カスタマイゼーションの元とするため，パタン化し，再利用可能な形に蓄積する方法について述べる．

第 5 章では，本章では，サービスのライフサイクル知識の構築と活用プロセスについて議論する．第 3 章，第 4 章の成果を活用し，全体をサービスライフサイクル管理方法論にする．

第 6 章では，第 3 章，第 4 章，第 5 章において提案した，サービスライフサイクル管理方法論を IT サービス事業の現場で実践し，Web サービスの開発事例で，その有効性の検証と考察を行う．

第 7 章では，本論文の結論および今後の展望を述べる．

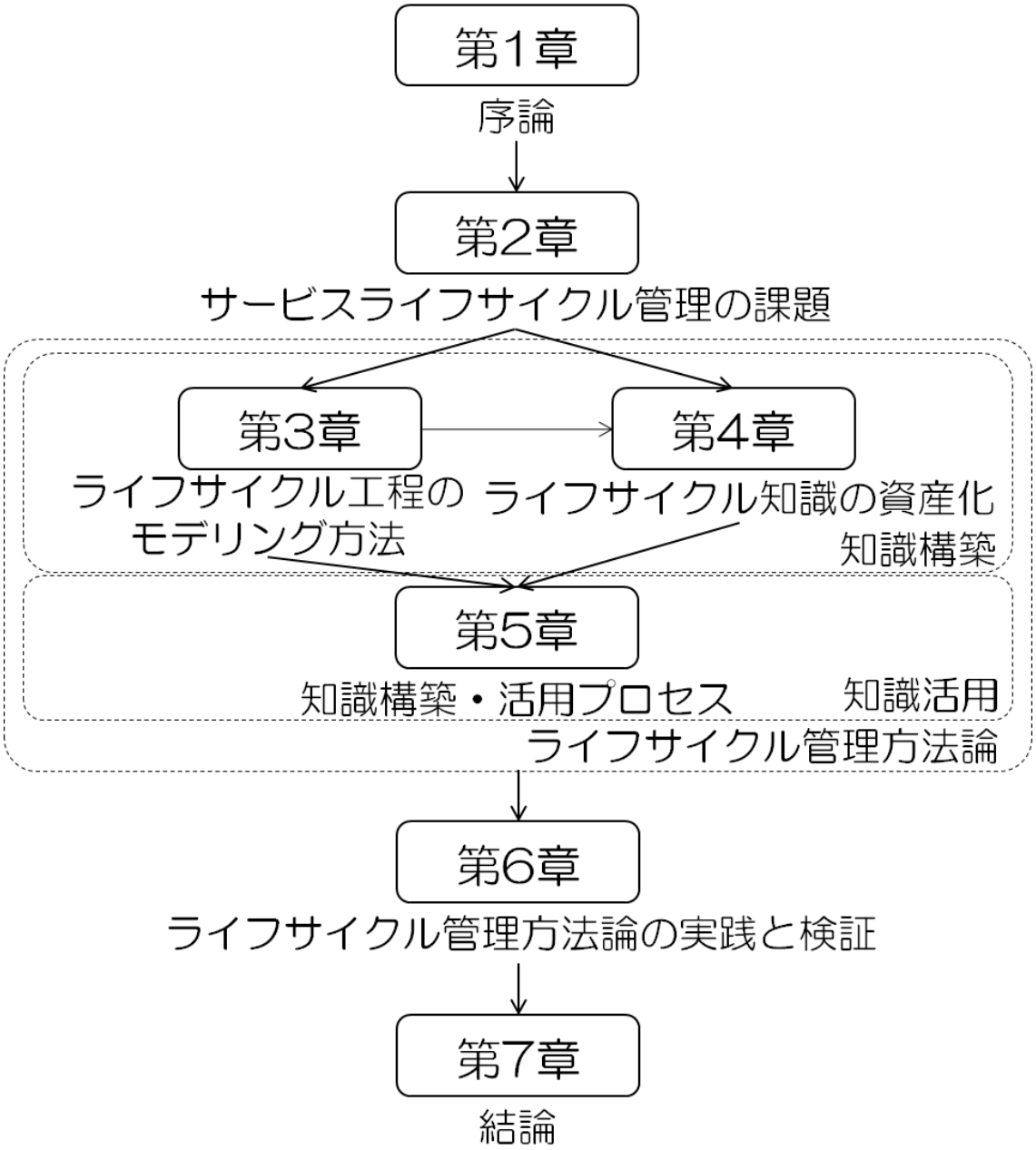
第1章
序論
第2章
サービスライフサイクル管理の課題
第3章
ライフサイクル工程の
モデリング方法
第4章
ライフサイクル知識の資産化
知識構築
第5章
知識構築・活用プロセス
知識活用
ライフサイクル管理方法論
第6章
ライフサイクル管理方法論の実践と検証
第7章
結論

図 1-2 本論文の構成

第２章 サービス化の進展とサービスライフサイクル管理の課題

目次

2.1 はじめに

本章では，サービスの捉え方を議論し，システムとしてサービスの提供構造を捉える必要性を示す．そして，あるべきサービスシステムの設計と構築および移行に必要となる要件を明確化する．

第 2 節では，サービスを機能提供の関係として捉えることを示す．機能の利用場面だけでなく，機能の企画・設計・構築や運用の場面においても機能提供関係があり，これらを捉えるには，製品とサービスが複合したシステム，すなわちサービスシステムとして，その提供構造を捉えることが必要であることを示す．

第 3 節では，IT サービスについて説明し，サービスシステムの構造と，その設計・構築および移行に必要とされる観点を明らかにする．

第 4 節では，前節までの考察を踏まえて，本研究の位置づけを述べる．最後に，次章以降で述べる本研究において提案する手法の概要を述べる．

2.2 サービスの捉え方

2.2.1 サービスの定義

下村らは，サービスの設計を科学的に理解し，モデル化するための「サービス工学」と，これを用いて付加価値の高いサービスを効率的に設計するための「サービス設計方法論」に関する研究を行っている［下村 2005］．この研究においては，サービスを

サービスの供給者であるプロバイダが，対価を伴って受給者であるレシーバ（受け手）が望む状態変化を引き起こす行為

と定義している（図 2-1）．

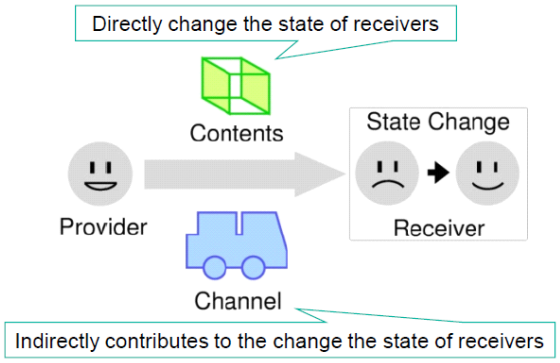

図 2-1 サービスの基本定義［下村 2005］

このコンテンツとチャネルによるサービスの定義は，ユーザに何らかの状態変化を起こす「機能」と，その「機能提供の媒体」である．

つまり，サービスの設計とは，ユーザに与える価値を機能と機能提供の媒体の総体を設計することである．この考え方に基づけば，機能は，サービス（コト）を構成する機能も，製品（モノ）を構成する機能も同じである．すなわち，従来，製品として提供してきた機能を，新製品の機能として提供することも，製品に付随するサービスの機能として提供することも同様に設計できる．

2.2.2 サービス化とは

前節のサービス設計の考察に基づけば，機能中心に価値提供を考えること，すなわち，サービス化（Servitization）とは，従来の製品売りの事業を，製品の備える機能や，製品の案内や提供などを付随する機能を含めて，全体をサービスとして組み立て直すことである．これは，製品およびサービス全体を，機能中心に売ること（Functional Sales）と捉えるこ

と［Mont2002, 2004］である．

機能中心に製品およびサービス全体を捉えるには，製品・サービスが備える個々の機能を基点に考察するのではなく，製品とサービスの提供手段である ICT プラットフォームや人的・物的リソースが有機的に結びついたシステムの観点で，分析することが重要になる．

2.2.3 サービスシステム

製品および付随する機能を含めた全体を設計することは，サービスを構成するシステムを設計することである．下村らは，サービスをコンテンツとチャネルによって，相互作用するものと定義している．青山は，「サービスを供給する総体を考え，サービスプロセスの複合体としてサービスシステムを認識する」としている［青山 2008］．このように，サービスシステムとは，相互作用を伴うプロセスの複合体と定義できる［人見 1990］．

この構造は，顧客が，サービスを**認知**（Access），その内容を**確認**（Check-in），その内容を**評価**（Diagnosis），その機能を**利用**（Delivery），機能の利用を**完了**（Check-out），利用後に**評価**（Follow-up）するまでの各体験過程において，サービスプロバイダと相互作用を伴うプロセスの複合体である．この視点に基づいて，サービスプロバイダは，企画・設計・構築・運用・改善と各工程において，顧客に対して相互作用を伴うプロセスを設計することが必要である．

以上に示した考え方は，サービスシステム［Maglio2006］［Spoher2006］［Simon1996］や製品サービスシステム（Product-Service System : PSS）の考え方と同様である．サービスシステムとは，顧客に対して製品の提供だけでなく，企画・設計・製造から完成後の管理運営・メンテナンスなどの支援まで含めたシステムを売ることで，製品の付加価値を高めるための概念である［Baines2007］．

2.3.4 サービスシステムのライフサイクル

本節では，サービスシステムを企画・設計・構築し，さらに継続的に運用していくために，ライフサイクル管理の点から，関連する取り組みを議論する．

IT サービス管理のベストプラクティスをまとめ，ガバナンスのフレームワークとしたものに，ITIL（IT Infrastructure Library）がある．これは，英国政府刊行物出版局より 1989 年以降発行された一連の書籍で，IT サービス提供の全ての側面を備えるべく，改訂されている．このフレームワークは事業と顧客双方の観点から提供する IT サービスの品質の継続的な測定と改善指針を与えるもので，近年，多数の国々の IT サービスの提供者に取り入れられている．2007 年に改訂された ITIL バージョン 3 では，サービスライフサイクル管理の概念を導入し，戦略・設計・移行・運用・継続的改善の視点で，IT サービスの全ライフサイクル工程を網羅した管理フレームワークを示している［Iqbal2007］（図 2-2）．各視点をまとめた各々の書籍には，システム開発部門が開発する IT システムをシステム運用と基

盤の観点で，開発したシステムを適切に移行・導入する観点や行うべき項目が体系化されている．

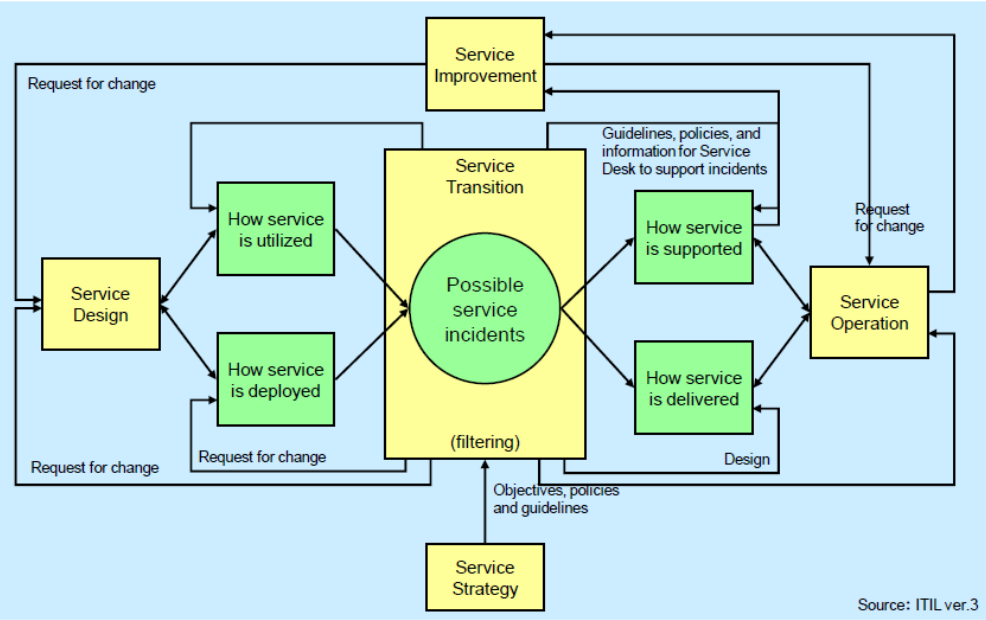

図 2-2 ITIL サービスライフサイクル［Iqbal2007(改)］

このように，戦略，設計，移行，運用，改善を一貫して考えることが重要で，顧客に製品を売り切るような関係ではなく，継続した関係を構築するために，運用を見据えた設計や，改善が容易な運用が必要になる．そのため，IT サービスのライフサイクルの流れに沿って，各工程の情報を有機的に結び付けることが必要である（図 2-3）．

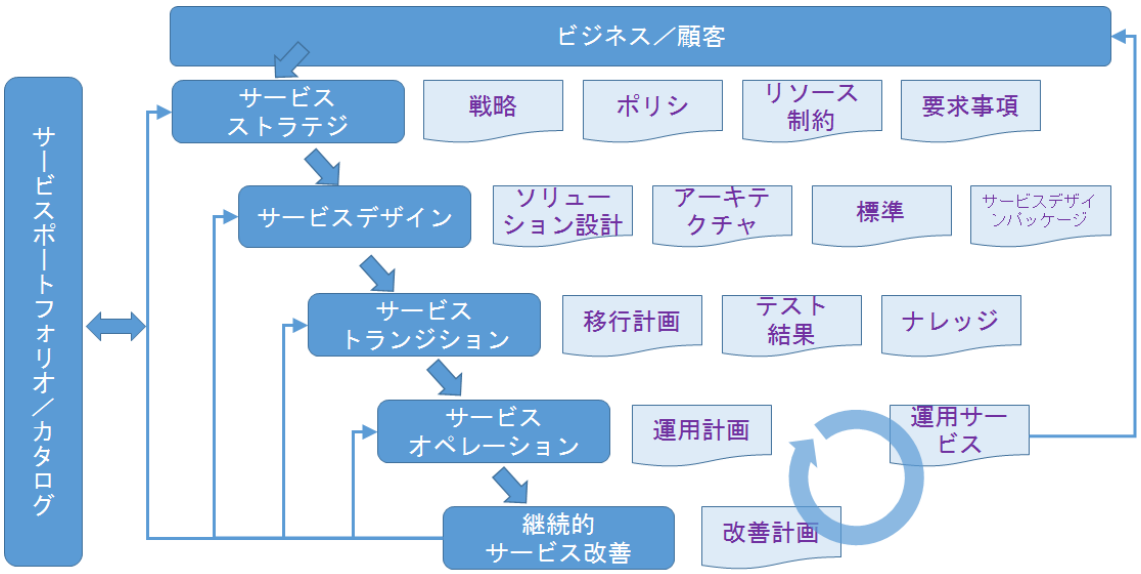

図 2-3 IT サービスライフサイクルの流れ［ITIL2008(改)］

以降，本論文では，サービス全体の機能提供プロセスおよび機能提供構造の仕組みを議論していく．この全体の仕組みを「サービスシステム」，またサービスシステムの企画から更改や新規のシステムに置換するまでの過程をサービスシステムのライフサイクルと称する．

以上，2.2 節では，サービスの捉え方，サービス化の進展に伴いサービスシステムを考えることの必要性，更にサービスシステムのライフサイクルを理解し，管理する取り組みについて述べた．

以降の節では，IT サービスシステムのライフサイクルに沿って，考察を進めていく．クラウドサービスを媒介とした変化は，すなわち，製品指向の事業モデル（フロー型）から，サービス指向の事業モデル（ストック型）へ変革を促している．仕様通りに動作する機能を実装し，それを顧客に納めることに価値を置いた事業モデルは製品販売の指向であるのに対し，顧客と設計～運用までの工程全体を通じた顧客との協働活動により，顧客に対して価値を生み出し，対価を得るサービス提供の指向に変化している．つまり，サービスプロバイダは，サービスシステムのバリューチェーン全体を把握し，顧客のライフサイクルを通じて継続的に協働していくための方法を必要とする．そのため，サービスシステムのライフサイクルの視点が不可欠である．

2.3 IT 領域におけるサービス化の進展

2.3.1 IT サービス

IT 業界において，コンピュータシステムとその運用・管理全体を IT サービスと称している．これは，コンピュータシステムとそれを運用し管理することが，事業や個人の問題解決の遂行を手助けするためである［山路 2006］．経済産業省の IT サービス継続ガイドラインでは，「IT サービスは，組織における業務の遂行に際して必要となる IT 及び IT に関連する体制の組み合わせによって提供される機能である」と定義している［経済産業省 2008］．このように，IT サービスとは，問題解決の道具としてのコンピュータシステムと，それを企画・設計・構築・運用し，問題解決の道具が機能する仕組み，および支援の総称である．

この IT サービスは，アナリストやシステムインテグレータなど人が介在したサービスと，コンピュータシステムとして提供するサービスが複合したものである．IT サービスを構成する Web サービスは，広義にはコンピュータシステム上で動作するソフトウェアとハードウェアの組み合わせ，ビジネス用途などのアプリケーションとして動作するコンピュータシステムを指す．World Wide Web（以降は，Web と略称する）で使用される各種技術の標準化を推進する国際非営利団体の World Wide Web Consortium（ワールド・ワイド・ウェブ・コンソーシアム; W3C）では，Web サービスを次のように定義している．「Web サービスとは，ネットワーク上の計算機と計算機が相互運用可能な通信をサポートするために設計されたソフトウェアシステムのことである．」更に，次のように狭義に定義している．「また，計算機処理可能な形式（特に，Web サービスのインタフェースを，人間もプログラムも理解できるように XML 形式で記述するための言語 Web Services Description Language; WSDL）で記述されたインタフェースを有する．他のシステムは，その記述で定められた方法で，その Web サービスと SOAP（Simple Object Access Protocol）[2]メッセージを用いてやり取りを行う．一般的に，SOAP メッセージは他の Web に関する標準と併せて XML（eXtensive Markup Language）でシリアライズされ，HTTP（Hyper Text Transfer Protocol）を用いて転送される」［W3C 2004］．

本研究では，上記に示したアプリケーションとして動作する広義の Web サービスと，その設計，構築，運用に関わる開発者や運用者の活動を含めたものを IT におけるサービスシステムとして捉える．本研究での IT サービスシステムの捉え方を図 2-4 に示す．IT サービスシステムは，コンピュータシステムとヒューマンシステムで構成する．コンピュータシステムは，Web サービスとそれを動作させる IT 基盤のソフトウェア製品・ハードウェア製品，またはクラウドサービスで構成する．ヒューマンシステムは，このコンピュータシステムを利用する，カスタマー・エンドユーザと，その導入・設定・運用・保守を行うコンサルテーション・SI サービス・運用保守サービスで構成する．これらの構成に基づくと，

[2] XML をベースとしたメッセージ交換のためのプロトコル仕様

IT サービスは，コンピュータシステムと，その導入・設定・運用・保守を行うコンサルテーション・SI サービス・運用保守サービスを合わせたものである．

この IT サービスシステムは，コンピュータシステムの進化に伴い，形態を変えてきている．近年，従来のコンピュータシステムをクラウドと呼ばれる環境への置換が進んでいる［Armbrust2009］．これは，クラウドから提供されるビジネスロジックや，基盤ソフトウェア，ハードウェアを利用して，Web サービスを構成できるためである．次節では，クラウドの浸透に伴うサービスシステムの変化について，考察する．

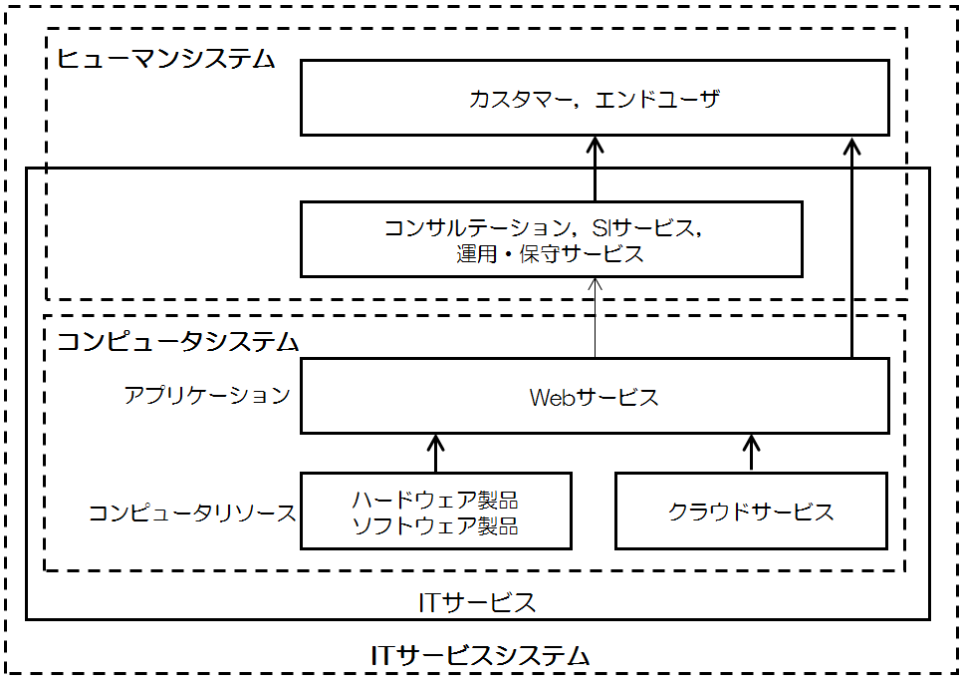

図 2-4 IT サービスシステムの構成

2.3.2 クラウドの浸透に伴う事業モデルのパラダイムシフト

2.3.2.1 クラウドの浸透

近年，クラウドサービスが急速に普及しつつある．クラウドは，記録媒体やネットワークリソースの急激な低価格化等を背景に，これまで偏在化していた計算機リソースを再び集約し，豊富で高機能なハードウェアやソフトウェアがデータセンタに集中的に置くものである．OS やストレージの仮想化技術の進展により，これらの計算機リソースを必要な分だけ切り出し，利用することが可能になった．クラウドサービスプロバイダは，この技術を利用して，データセンタ内の計算機リソースを，ネットワークを通じてクラウドサービスとして提供している．ハードウェアや OS の機能は IaaS（Infrastructure-as-a-Service）サービスとして，更にデータベースやアプリケーションサーバ等ミドルウェア機能を PaaS（Platform-as-a-Service）サービスとして提供している．また，これらの上位として業務

アプリケーションを SaaS（Software-as-a-Service）を提供している．これらのサービスは，従来のソフトウェア・ハードウェア構成スタックを再構成したものと言える．IaaS はハードウェアや OS 機能を再構成したもの，PaaS は IaaS にミドルウェアを含めて機能を再構成したもの，SaaS は，その上位に業務アプリケーションの機能を再構成したものである（図 2-5）．

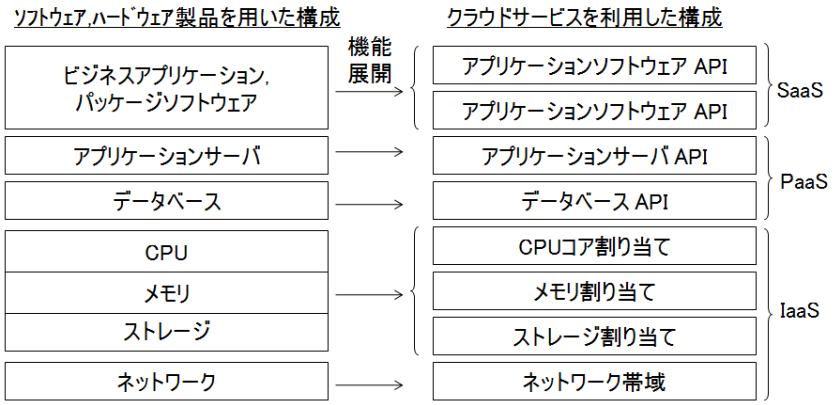

図 2-5 クラウドサービスの構成［細野 2013b(改)］

2.3.2.2 事業モデルのパラダイムシフト

このクラウドサービスの導入は，IT システムを構成するソフトウェア・ハードウェア製品を置き換えることである．この変化に伴い，システム構築・管理を担う IT サービスプロバイダは，従来のソフトウェア・ハードウェア製品の統合販売を主体としたビジネスモデルから，サービス提供の知識を蓄積（ストック）し，それを活用して多くの顧客と継続的なサービス提供を行うビジネスモデルへのシフトが求められる．このシフトを上手く進めるには，機能提供関係で製品とサービスから成るシステム全体を捉え，バリューチェーンや顧客価値の適切な設計が必要になる．

クラウドの浸透は，IT サービスのシステム構造にも変化をもたらす．クラウドを利用した IT システムの構築は，従来の企業内への構築に比べ，設計・運用に関わるステークホルダに違いが生じる．すなわち，従来のシステムインテグレータ，Web サービスのプロバイダ，最終顧客の三者モデル（B_1toB_2toC）という構造に対して，クラウドを利用した場合には，システムインテグレータ，Web サービスのプロバイダ，クラウドサービスプロバイダ，最終顧客の四者モデル（B_1toB_2toC，B_3toB_2）という構造になる（図 2-6）．

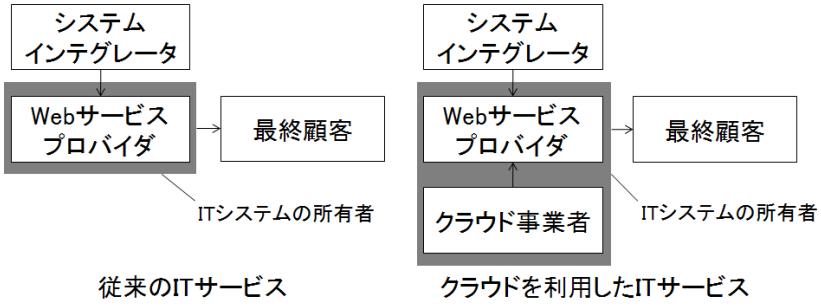

図 2-6 IT サービスのシステム構造の変化［細野 2013b(改)］

この構造において，SI サービスは Web サービスを構築するサービスを Web サービスプロバイダに提供するもの，運用サービスは Web サービスを安定実行させるサービスを Web サービスプロバイダに提供するもの，Web サービスのプロバイダは Web サービスを最終顧客に提供するものである（図 2-7）.

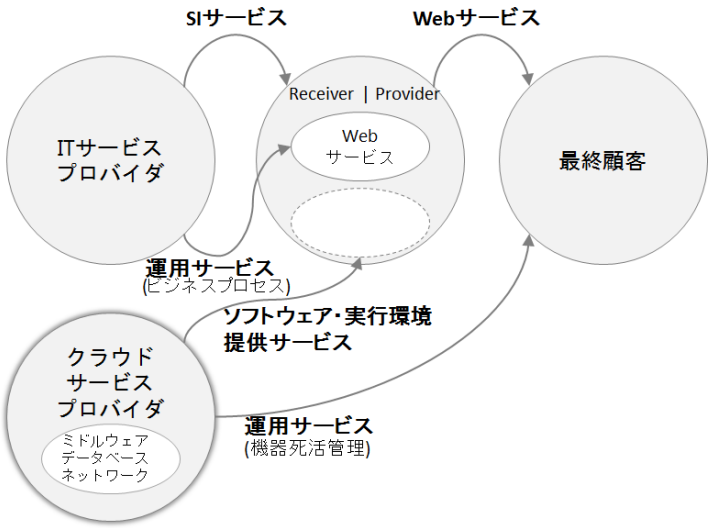

図 2-7 クラウドに対応した IT サービス［細野 2013b(改)］

2.3.2.3 提供価値のシフト

事業モデルのパラダイムシフトは，IT サービスプロバイダからの IT システム構築ソリューションにも変化をもたらす．クラウドは検証・管理済みのソフトウェア，ハードウェア機能を提供するため，Web サービスを構成するアプリケーションソフトウェアをソースコードから開発し仕様通りの機能を実装する方法よりも，ソフトウェア・ハードウェア機能を効果的に組み合わせて付加価値の高い Web サービスを設計する方法が重要となる．この変化に伴い，IT サービスプロバイダは顧客企業のシステムの実装や検証を担う役割から，上流工程において新たなビジネス／サービス設計を行う役割や，下流工程においてサービスの改善提案を行えることがより求められる（図 2-8）.

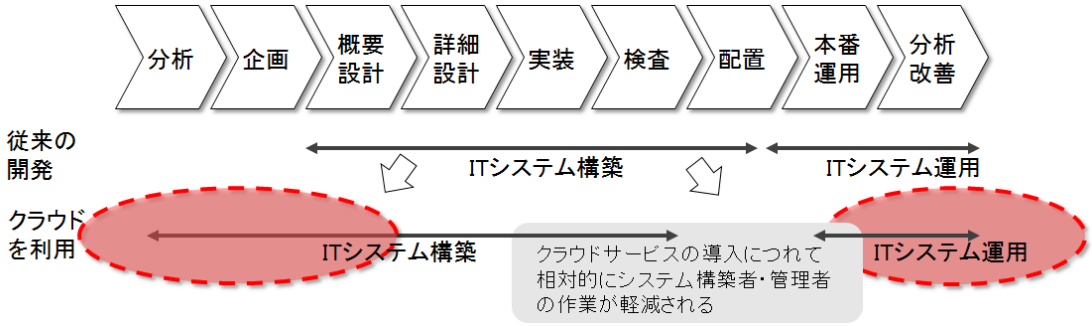

図 2-8 IT サービスプロバイダの役割［細野 2013b(改)］

2.3.3 サービス化の進展に伴う顧客価値

従来の IT サービスは，所有権を移すビジネスであり，その価値の源泉は製品販売とシステム構築に置かれている．しかし，クラウドサービスがソフトウェア・ハードウェア製品を代替することから，価値提供の工程は，ライフサイクルに沿って拡がる．サービス化の進展は，プロバイダが顧客に対して長期的に妥当な価格でサービス提供を行うことを促す．このように，価値の源泉は変わっていくこと［Norman2001］を捉えることが必要である．

以上の考察からサービス化の進展に伴い，顧客価値（value proposition）の源泉は，特定の顧客に対して 1 回限り開発された，開発成果物から，他の顧客にも適用できる再利用可能な開発知識に遷移したと言える．この進展は，次のように要約できる（図 2-9）．

初期段階：製品所有指向（product-oriented）

1. 価値の源泉である開発成果物の所有権は顧客に移転される．(この顧客向けのシステムは請負契約の下，機能保証を行う前提で開発される．)
2. 顧客価値の源泉である開発成果物は各顧客に固有のものであるため，顧客毎に専用の開発成果物が 1 回限り開発される．

進行段階：機能利用指向（use-oriented）

1. 顧客価値の源泉であるデータの所有権は，できるだけ多くのサービスプロバイダに保持される．(製品販売中心のビジネスと対称的に，契約は作業工数の実績に基づく)
2. 顧客価値の源泉は，繰り返し再利用することを前提とした開発知識で構成する．
3. 構造化されたデータをそのまま再利用可能なだけでなく，様々な顧客の要求にカスタマイズして対応できる．

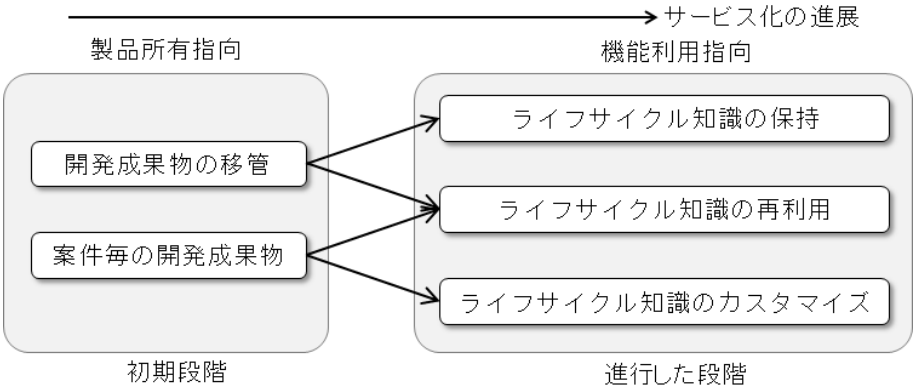

図 2-9 サービス化の進展に伴う顧客価値のシフト［Hosono 2013a(改)］

2.3.4 所有から利用中心のサービスへの移行

クラウドの浸透は, Web サービスのシステム開発を担うシステムインテグレーション(SI)サービスの契約形態にも変化を与えている. 従来の SI サービスのビジネスモデルでは, Web サービスを構成するソフトウェア・ハードウェア製品の費用と，Web サービスの実装に要した開発工数に対して，対価を得ていた．このビジネスモデルは請負契約に基づき，Web サービスの実装完成システムと，開発中に作成した機能仕様書やテスト報告書，評価データなど開発成果物を一式顧客に納めている．これは，SI サービスのプロバイダ（システムインテグレータ）側には成果物やデータは残らない形態のため，売り切り型のビジネスと見做せる.

しかし，クラウド上に Web サービスを構築することが前提になると，従来の SI サービスのビジネスモデルは成り立ちにくい．これは，ソフトウェア・ハードウェア製品機能や，ビジネスロジックを実装したソフトウェアコンポーネントが，クラウド上の PaaS（Platform as a Service）や SaaS（Software as a Service）としてサービス提供されるため，実装や検証作業の多くが不要になるためである．一方，クラウドから豊富に提供されるソフトウェア・ハードウェア機能を利用できるため，Web サービスをスクラッチから実装するよりも開発作業が容易になる．そのため，SI サービスのプロバイダは，現状の Web サービスを分析し，よりビジネス効果の高い Web サービスを企画・提案し，クラウドの機能インタフェースを効率良く組み合わせ，短期間にプロトタイプを開発し，本番の Web サービス実行環境を迅速に提供することがより求められる.

このように，クラウドの浸透に伴い，SI サービスのプロバイダは，よりビジネス視点や全体システム観点の見識を発揮して，ライフサイクルの上流工程や下流工程においても顧客と継続的に協働して，Web サービスを迅速に企画・設計・構築・提供することが必要である．このとき，SI サービスのプロバイダは，開発・実行で得た知見をライフサイクル知識として保持し，それをコンピタンスとして多くのサービス開発に再利用することが重要

になる（図 2-10）.

これは，モノの売り切り販売から，知識を蓄積（ストック）し，継続的な機能提供を行うビジネスモデルへの進展とも見做せる．これは，Vargo，Lusch が示したサービス・ドミナント・ロジック（SDL）［Vargo 2004］に照らし合わせると，顧客に対して，IT システムの実装品提供というモノ主導の交換価値から，顧客との IT システム企画・設計・構築・運用の協働作業を媒介として，顧客のビジネス価値を向上すること，すなわち，コト主導の使用価値で SI サービスを設計し直すことへの進展とも言える．

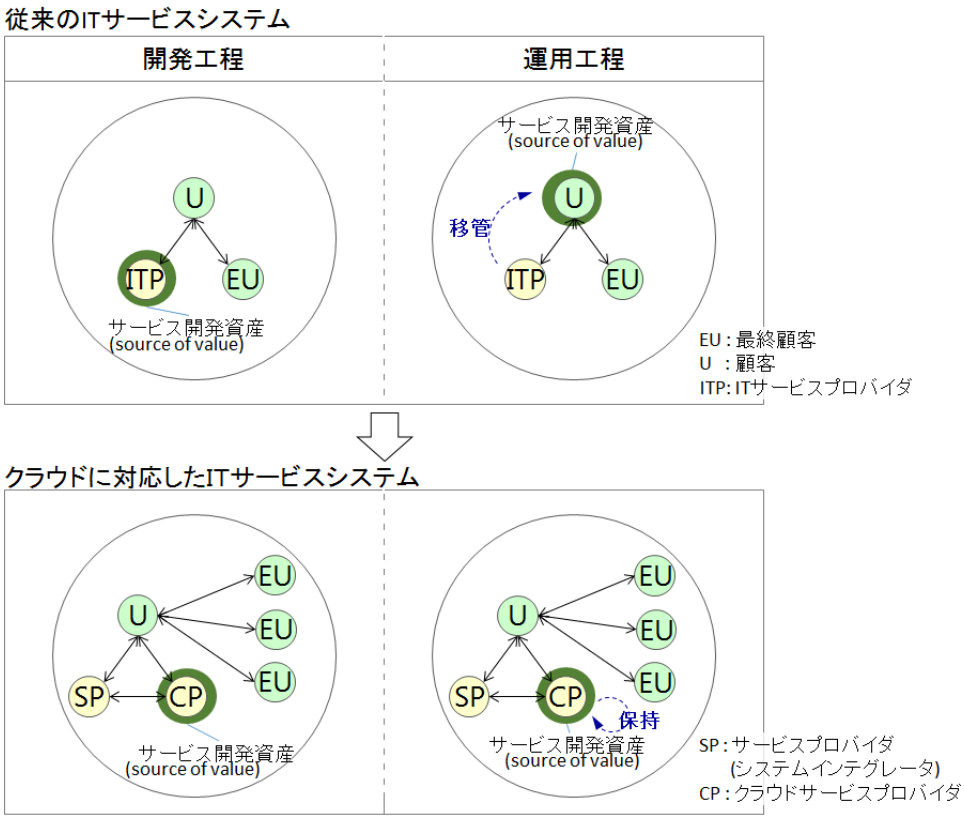

図 2-10 製品所有と機能利用を指向したビジネスエコシステム［Hosono2013a(改)］

このサービス主体の事業モデルに移行するには，サービスプロバイダがコンピタンスを活かし多くの顧客と 1 対多の関係を継続して構築することが鍵となる．そのためには，サービスプロバイダが，常に顧客と協働する関係を保てるように，上流～下流までサービスを協働して開発し，運用する仕組みを体系的に備えることが必要である．

2.4 本研究の位置づけ

2.4.1 本研究の範囲

本研究では，サービスの企画・設計・構築・運用・改善までの全工程に対して一貫したモデリング方法を開発する．各工程でモデリングした情報が相互に連携するように，モデリングしたデータ間の連携と一元管理の仕組みを開発する．モデリング方法により得られた全工程のデータを，類似した別のサービスの開発サイクルで用いることができるように，カスタマイズし易い形にデータ群を資産化する方法を開発する．更に，設計者や運用者，管理者，顧客など全工程内の工程間およびライフサイクル間に関係するステークホルダが，その役割から，資産化されたデータを有効に活用する方法を開発する（図 2-11）．

上記のモデリング方法，資産化方法，活用方法について，IT サービスを対象として議論を進める．また，クラウドの浸透をサービス化の進展と捉えて，サービス化の進展に伴うステークホルダの構成変化やステークホルダ間の新たな協働に着目し，モデリング方法と資産化方法の実践プロセスの議論を進める．

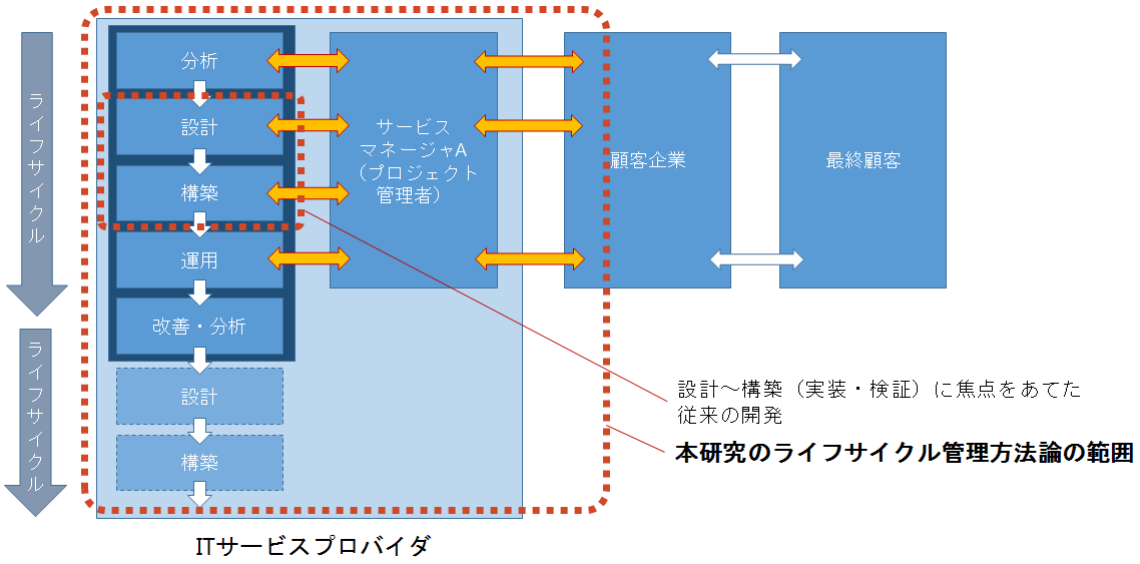

図 2-11 本研究の範囲

2.4.2 本研究の提案内容の概要

本研究では，1.2 節において整理した目的・課題に対し，前節で示した範囲について，ライフサイクル管理方法論を示す．提案する方法論の全体像と概要を次に示す（図 2-12）．

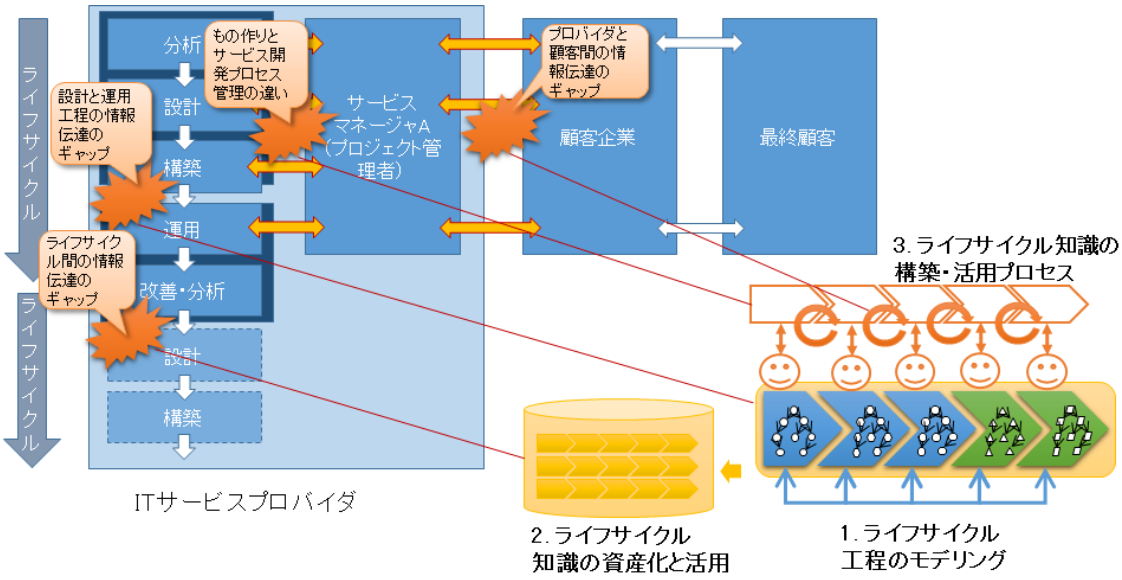

図 2-12 ライフサイクル管理方法の概要

以下に，本研究での提案方法の概要を示す．既存研究との違いなどの詳細は，第 3～5 章の各章で述べる．

本研究では，上流～下流まで各開発工程におけるモデリング対象を規定し，全工程を網羅するモデリング方法を提案する．

ハードウェア製品の製造は，その開始時点で仕様が確定し，生産の各工程での作業や期間が明確化されている．一方，サービスの開発工程では，サービス提供者内のみならず，ユーザ，顧客になり得るステークホルダからの要求獲得など，ステークホルダ間で多くの協働作業を行い，合意形成を図る必要がある．更に，サービスの提供工程では，顧客と契約を合意し，それを維持する必要がある．分析～企画・設計に至る過程では，顧客と企画者の間で仕様の合意形成を図り，実装～提供の過程では，設計担当と運用担当の間で設計情報を共有する必要がある．これらのやり取りにおいて，開発者の役割に応じて使用する情報の粒度（抽象～具象）の差異を埋め，これらのステークホルダ間の共通理解を得ることが必要である．

そこで，従来，自然言語での要件記述やノウハウとして暗黙的に扱われていた上流設計工程を含めて，モデリングにより形式化を図る．要件→機能→リソース割り当ての転写関係により，サービスモデルの連鎖としてサービス設計・運用プロセスを網羅する．従来のモデリングは，各モデルが独立していたのに対し，本研究が提案するモデリングは，各モ

デルで規定される属性間の関係を明確化し，工程間のデータ連携を間断なく実現し得るものである．また，DSM（Design Structure Matrix）の拡張により，設計データとして表現や管理し難かった，非機能要件も機能要件と透過的にモデルデータとして扱うことを可能にする．更に，設計～運用の各工程で関わる開発者の役割に対応して，細粒度に分割したモデリングを示し，これらのモデルを公理的設計の考え方に基づいて，モデルチェーンを構築する．この一連のモデリングは，異なる役割を持ったステークホルダ間の共通言語として作用するため，ステークホルダ間の相互認識を深められる．また，従来自然言語で書かれた上流工程の情報も，正規化されたデータとして記録されるため，開発の後工程で上流設計の振り返りも容易になる．これにより，システム／サービスの開発を，手戻りなく迅速に進めることが可能になる．

前述のモデリング方法を用いて，設計・運用プロセスの再利用に向けた資産化方法を提案する．開発した各モデルを元にサービスの設計・運用プロセス全体を網羅する，サービスモデルチェーンを構築する．このサービスモデルチェーンを資産として保持することで，別の顧客に対して同じ Web サービスを開発する際に再利用できる．サービスモデルチェーンには，開発に必要な設計情報が整列されるため，開発全体の作業や成果物の雛形になるからである．しかし，一般に，同じ Web サービスの開発であっても，全ての要件が同じことは稀で，顧客毎にユーザインタフェースや部分的な機能要件の差異が生じる．そこで，このサービスモデルチェーンを有効に多くの案件に適用できるように，カスタマイズを前提とした，マス・カスタマイセーションを指向する．この実現には，多品種少量生産に適したプロダクトラインエンジニアリングの考え方を応用し，ドメイン開発とカスタマイズの元になるパタン開発を行う．

これにより，従来，モノの生産に活用されてきた，マス・カスタマイゼーションの概念をサービスの設計～提供においても実践できることを示す．この設計～提供方法は，サービス生産の短期化と同時に，生産の原価低減を達成するもので，範囲の経済性および規模の経済性効果の高さを得られる．サービス価格の上昇を抑え，顧客が持続してサービスを利用し易くなる．よって，サービスシステムの維持・安定に寄与し，ストック型の事業を実現できる．

更に，上記 2 つの方法に加えて，顧客と提供者，および異なる役割を持った提供者内のステークホルダ間の協働作業を支援する方法を示し，全体を効用の高いサービスシステムのライフサイクル管理方法として体系化する．

効用の高いサービスを効率的に生産するには，上記 2 つの方法に加えて，(a) 顧客と IT サービスプロバイダや，開発と運用部門，開発部門と企画・管理部門間などライフサイクル内のステークホルダ間の協働支援，(b) 過去のサービス開発と別のサービス開発の開発部門などライフサイクルを跨ったステークホルダ間の協働支援と，これらを定着させるためのプロセスが必要である．

IT サービスプロバイダと顧客企業担当者間でサービス設計を進める際，要求・構造・振舞いなど特定の観点で絞った情報量のダイアグラムで表現し直すことが合意形成に有効である．そこで，サービスモデルチェーンを特定の観点で動的に再構造化し，ステークホルダが求める表現形式で提示し，顧客と IT サービスプロバイダ間の協働支援する．

また，開発成果物を資産化する際，その資産の再利用シナリオや資産化の意図を保持できるように，再利用戦略ポリシーとして形式化し，開発部門間の協働を支援する．一方，運用管理の保守性や，改良のし易さなど，実装・実行工程での要件や制約を併せて考慮し，これらの多目的に最も適合する開発資産を活用させることで，開発部門間の協働を支援する．

また，モデルチェーンの構築過程において，各モデルに紐付いた開発者の作業量をデータとして定量化する．この定量データを含む設計・運用プロセスのパタン化によって，モノの生産プロセス管理と同様に，管理部門が開発部門のサービス設計・運用プロセスの比較分析・管理を可能とし，開発部門と企画・管理部門との協働を支援する．

以上に示した支援方法を実際の開発に定着させるため，ソフトウェアをアジャイル開発で行う際,その進行状態や優先すべき作業を可視化する Software Engineering Method and Theory（SEMAT）を拡張し，上記に示した手法を漏れなく行うタスクとして提示する．これにより，開発プロセスに規律を与え，支援方法を定着させる．

このように，人の活動を可視化し，設計・運用プロセスの中で，明示的に扱えるようにしたことで,開発資産の有効な活用が可能となる．従来プロセス指向であった開発方法を，タスク基点に捉え直したことで，各ステークホルダと協働したサービス生産が可能となる．

2.4.3 本研究の位置づけ

本項では，既存研究との比較により，本研究の位置づけを述べる．本研究は，サービスを提供するまでの構造，および提供時の構造を，多様なステークホルダから成るサービスシステムとして捉え，設計から運用までのサービスライフサイクルを管理する方法論である．この方法論は，従来の製品販売主体から，サービス主体に IT サービスを捉える．これにより，従来のビジネスモデルからサービス中心のビジネスモデルに遷移を促せる．この方法論を構成する 3 つの研究の位置づけを以下に整理する．

2.4.3.1 ライフサイクル工程のモデリング方法

従来の IT サービスのモデリングは，①Web サービスを構成するソフトウェアの機能設計を正しく行うために，機能設計の根拠となる，要件定義およびユーザのユースケースをモデル化の対象としている．また，②IT サービスを継続的に改善できるために，サービス指向アーキテクチャ(SOA)として，ソフトウェアシステム全体と各モジュールの設計指針を与えている．これらのモデル化では，ソフトウェアシステムを実行するハードウェア・ネットワークなどのリソースは，ソフトウェアの機能に紐付いたものであり，モデル化の対

象となっていない.

ソフトウェアを実行するリソースも仮想化され，クラウドサービスプロバイダにソフトウェアと別に機能提供されるようになると，ソフトウェアの機能を発現するにはリソースもモデル化し，ソフトウェアとリソースを包括して Web サービスの機能を設計する必要がある．しかし，これらは別のステークホルダによって管理され，更に，ライフサイクル上，異なる工程で管理されるため，ソフトウェアとリソースを包括して扱うことが困難である．現状では，その手段が無く，実装したソフトウェア機能モジュールをクラウドに配備して，機能を発現しているか否かを確認している．

そこで，本研究のアプローチは，クラウドサービスプロバイダが管理するリソースのモデル化を図ることで，機能設計においてソフトウェアとリソースとの情報の統合を可能にする. リソースと前後の工程のモデリングにおいて, DSM と DfX（Design for X）[Paul 1996]の考え方を応用することで，機能と非機能の検証が容易になる効果も得られる．本アプローチのポイントは，①工程の分割により各モデル化の範囲を決め，②リソースのパラメータの可視化と，リソースのモデリングおよび割り当てによって，モデル間の連携関係を作ることにより，情報の間断無いモデルを構築する点にある.

従って，両者の違いは，機能に形態を与える実体のモデル化と，機能から実体への転写関係によりモデル間の連携を考慮している点にある．本研究の特長は，工程を跨って情報連携できる，ライフサイクルを一貫したモデリングにある.

2.4.3.2 ライフサイクル知識の資産化方法

一度作成したソフトウェアシステムの設計書やソフトウェアコードなどの開発成果物を類似したソフトウェアシステムの開発に体系的に活用する方法に，ソフトウェアプロダクトラインエンジニアリング［Clements 2001］のアプローチがある．ソフトウェアプロダクトラインエンジニアリングは，設計～製造までの工程を対象とし，コア部分と案件に依存した派生部分の分離を図っている．プロダクトラインエンジニアリングでは，ソフトウェア機能モジュールを対象とし，実行リソースを十分に考慮していない.

Web サービスの機能は，ソフトウェアと実行リソースの組みで発現できるので，配備先となるリソースも含めて再利用可能な資産を設計する必要がある．また，複数の顧客に効率的に展開するための仕組みも必要になる.

そこで，本研究のアプローチは，設計～製造～配備までを対象とし，ソフトウェア機能モジュールに加えて，配備先のリソースまで対象に含める（異なるステークホルダによって設計と運用が行われる場合も含む）．また，複数の開発プロジェクトから参照できるリポジトリと，個々のプロジェクトで用いるリポジトリを分離して設計する．本アプローチのポイントは，①開発段階でなく，運用工程まで含めてカスタマイズ点を導出し，②それをパタン化したライフサイクル知識としてリポジトリに保管し，③各サービス開発プロジェクトにおいて，それを選択して，カスタマイズして再利用できる点にある.

従って，本研究と従来研究のアプローチの違いは，資産化する工程の範囲の差にある．本研究の特長は，クラウドサービスプロバイダが扱うリソース割り当ておよびリソース管理までの工程を資産化およびカスタマイズの対象に含め，その資産をサービス設計者が利用できることにある．

2.4.3.3 ライフサイクル知識の構築・活用プロセス

従来の活用プロセスは，新たな開発方法論や開発パタン・成果物を構築することに主眼があり，構築されたパタンに基づくやり方で現場のやり方を変えることを求めるものである［Zhang 2007］．従って，設計者や実装者などのステークホルダには着目していない．一方，開発成果物を活用する点で，ステークホルダに着目した研究は，設計意図の形式化［荒井/Arai1998］や，設計者の多目的・多様性［石川 2010］の考慮があり，限定されたステークホルダ間での意思伝達を行っている．

新たな開発方法論や開発パタン・成果物の導入は，現場の設計者や実装者間，提供者と顧客との間のプラクティスとなじまず，導入と実践が進まない状況を変える必要がある．また，企画・設計・構築・運用・次の開発までのライフサイクルには，設計者や実装者間，提供者と顧客間などの多様なステークホルダ間で，再利用に関する意図の伝達やプラクティスを考慮する必要がある．

そこで，本研究のアプローチは，企画～運用までにサービス提供に関わるステークホルダの役割と，ステークホルダ間のプラクティスを中心に知識構築・活用プロセスを扱う．サービスシステムおよびライフサイクル全体を通じて，主要なステークホルダ間のインタラクションに，①顧客と IT サービスプロバイダ間，②設計部門間（ライフサイクル間），③開発と管理部門間の協働があることに着目し，そのインタラクションの中でライフサイクル知識を有効に活用させるタスクを設計する．本アプローチのポイントは，①提供者側の設計者・運用者間（工程間）および提供者と受給者間，②管理者と開発者（設計者および運用者）間（上位視点・下位視点），③プロジェクト間（ライフサイクル間）の開発者間のそれぞれについて，構築したライフサイクル知識の活用方法を示した点と，④これらの方法を定着させるための規律を定義し，実践方法を示したことにある．

従って，従来研究と本研究のアプローチの違いは，開発方法論や成果物などのデータ起点に置くか，役割やプラクティスなどのステークホルダ起点に置くか，にある．本研究の特長は，多様なステークホルダとその関係（アクターネットワーク）をサービスシステムおよびサービスライフサイクルの主要な構成要素として扱い，ステークホルダ起点でライフサイクル知識の構築と利用タスクを設計している点にある．

2.4.3.4 本研究の特長

以上の議論を整理すると，本研究の特長は次のようにまとめられる．

- 本研究は，製品販売主体のサービスから，サービス主体のサービスに系統立てて遷移させていくためのサービスライフサイクル管理方法論である．

- 従来のモデリング方法は，ライフサイクルの各工程において必要とされる情報に対して，モデル化を図っている．これに対して，本研究では，工程間の情報連携に着目し，サービス化に伴うステークホルダの分離に伴う，工程間の情報分離を埋めるモデリング方法である．実体のモデル化と，機能から実体への転写（実体化）関係によりモデル間の連携を考慮している点に差異がある．

- 従来の資産化方法は，要件定義～実装までに提供者が開発した機能をモジュール化することである．これに対して，本研究は，企画～運用まで対象工程を拡げ，開発者および運用者が開発・運用した情報（異なるステークホルダによって設計と運用が行われる場合も含む）を蓄積し，サービスの開発者が利用できる資産化方法である．クラウドサービスプロバイダが扱うリソース割り当ておよびリソース管理までの工程を資産化およびカスタマイズの対象に含め，その資産をサービス設計者が利用できる点に差異がある．また，ドメイン開発によるカスタマイズインタフェースの導出と，パタン開発によるカスタマイズの元になるデータの構築方法である．

- 従来のプロセス研究は，開発方法論や成果物などのデータを起点に置いたアプローチである．これに対して，本研究は，サービスのライフサイクル知識を構築し，利用するステークホルダの役割を起点に，そのステークホルダの行動指針・行動支援に着目した構築・活用プロセスを示す．すなわち，多様なステークホルダとその関係（アクターネットワーク）をサービスシステムおよびサービスライフサイクルの主要な構成要素として扱い，ステークホルダ起点でライフサイクル知識の構築と利用タスクを設計している点に差異がある．

2.5 おわりに

本章では，本研究において採用するサービスの定義とその特徴について述べ，サービスシステムの捉え方を明らかにした．クラウドの浸透に伴うパラダイムシフトを，サービスシステムの進展として考察し，顧客価値の源泉に変化が生じていることを明らかにした．この考察を踏まえて，IT サービスシステムを生産する課題について整理し，次章以降で論じるライフサイクル工程のモデリング方法，ライフサイクル知識の資産化方法，および活用プロセスの位置づけを述べた．

第 2 節では，サービスを機能提供の関係として捉えることを示した．機能の利用場面だけでなく，機能の企画・設計・構築や運用の場面においても機能提供関係があり，これらを捉えるには，製品とサービスが複合したシステム，すなわちサービスシステムとして，その提供構造を捉えることが必要であることを示した．

第 3 節では，IT サービスについて説明し，IT サービスと，その設計・構築および移行に必要とされる課題を明らかにした．サービスシステムの定義，および IT サービスシステムの位置づけを考察し，IT サービスにおいて，工業化の進展に伴い，大きな変化点であることを明らかにした．これまでは，IT サービスとは，システムインテグレーション（SI），あるいは，死活管理を行う運用サービスを指していた．しかし，所有から，利用への大きなパラダイムシフトに対応するため，ライフサイクルを通じて，サービスシステムの考え方で，IT サービスを捉えることが必要である．以上のように，IT サービスプロバイダは，このパラダイムシフトに合わせた価値提供を考える必要性を明らかにした．

第 4 節では，前節までの考察を踏まえて，本研究の位置づけを述べた．

本研究の範囲を示し，次章以降で論じるサービスライフサイクル管理方法の概要と，各研究の特長について説明した．

第３章 ライフサイクル工程のモデリング方法

目次

3.1 はじめに

本章では，上流～下流まで各開発工程におけるモデリング対象を規定し，全工程を網羅するモデリング方法を提案する．サービスモデリングに関する先行研究について概説した後，それらを踏まえた本研究のアプローチについて説明を行う．そして，ライフサイクルを一貫して設計支援するためのモデリング方法を提案する．

第 2 節では，サービスモデリングに関する先行研究を示し，サービスモデリングの課題を考察する．

第 3 節では，設計の流れや設計観点に関するモデリング方法として，DfX（Design for X），DSM（Design Strucrture Matrix）の目的について述べ，その特徴を考察する．そして，これらを統合して，設計プロセスを単純化する方針を示す．

第 4 節では，第 2 節の方針を踏まえて，サービスモデリングの基本的な考え方を示す．工程間の情報が連携できる，サービスモデリング方法を提案する．

第 5 節では，前節のサービスモデリングを上流～下流までのモデリングを細分化して，サービスライフサイクルを網羅したモデリングを示す．ゴール設計，サービスシステム設計，機能・品質設計，機能担体および実行リソースへの割り当て設計，実行リソースの分離と結合による機能・非機能検証，非機能要件に基づく実行リソースの監視の各モデリングを公理的設計の考え方で，サービスモデルチェーンを構築する．

第 6 節では，これらのモデリング方法に対して評価と考察を行う．

3.2 ライフサイクル工程のモデリングに関する要件

3.2.1 モデリングの要件

サービスの設計は, 要件定義を行い, 要件を元に概要設計を行うステップで進んでいく. 要件定義は，顧客のビジネス目標を達成するために顧客を理解し要求を獲得する工程，顧客の現状のビジネスとの差異から要求を明らかにする工程，更に，この要求を元にサービス設計に向けた要件を定義する工程から成る．概要設計では，定義された要件が，機能要件，あるいは，非機能要件であるかを判別し，機能要件を達成するための機能群の定義と，非機能要件を実現するための機能およびアーキテクチャの定義を行う．次に，これらの定義と，プロバイダ側のシステム，ビジネス上の制約と擦り合わせを行い，設計解を導出するステップが続く．以上のステップを通じて，サービスの上流設計を完成し，以降，サービスを実装するためのパラメータを決定する詳細設計，実装，検証，導入，評価とステップが進んでいく．

要求や要件は，顧客と言葉で交わされた内容を，自然言語で記述し，共通理解を図ってきた．この書き方に曖昧さや，情報の不完全さが含まれやすく，開発の後工程で初めて気づくこともある．そのため，開発の手戻りの大きな要因となってきた.

そこで，要件定義を形式的かつ客観的に記述するための試みがなされている．要求や要件は，ダイアグラムのような標準的でかつ可視化できるモデルで表記しにくいため，項目を定義する．IEEE 803 では，ソフトウェアの要求定義を行っている［IEEE830］．要求を階層構造で表現し，仕様を漏れなく抽出する表現方法として，USDM（Universal Specification Describing Manner）がある［清水 2010］．この表現方法は，仕様は要求の中の動詞にあるという考え方に基づき，要求を振舞いとして表現し，仕様はその振舞いが及ぶ範囲の中で導き出す．要求と合わせて理由を記述することで，要求の意図や背景に関する情報を併せ持つことができる．これは，国内のシステムインテグレーションの開発文書形式として広く用いられている Microsoft Excel の表形式の記述と親和性が高く，仕様と設計要素間の関係を示すマトリクスと組み合わせて仕様と設計内容のトレーサビリティの確保に応用できる．そのため，近年，現実解として，国内のシステムインテグレーション事業で USDM の採用が図られている.

この要求から仕様を定義する過程において，漠然とした希望や期待する効果などの要求は，実現対象とするシステムの要件に置き換えがなされる．すなわち，希望や期待などの「要求」は，システムで提供し得る「機能要件」と，システムの構成などで達成できる「非機能要件」として具体化される．機能要件は，顧客の要求との対応が明快であるため，これまで，UML（Unified Modeling Language）［UML 2005］が採用されてきた．一方，非機能要件は，機能と対応づけられるものと，物理的な機能に対応付けられず，アーキテク

チャで実現するものがある［IPA2011］．また，オペレータなど人手の操作や行動で実現するものがある．これらを形式化するために，非機能要求グレード［IPA2010］では，標準的な非機能要件をリストにして体系化している．特に，アーキテクチャに関わる部分では，UML を補完する形で，SysML（System Modeling Language）が定義され，次第に，導入が図られている（図 3-1）．これは，機能，振舞い，制約，および要求も含んでいる（図 3-2）［SysML2008］.

この UML および SysML は，特定の観点でモデルを表記する方法を示しているが，各モデルで技術される項目間の関連性について網羅的に規定していない.

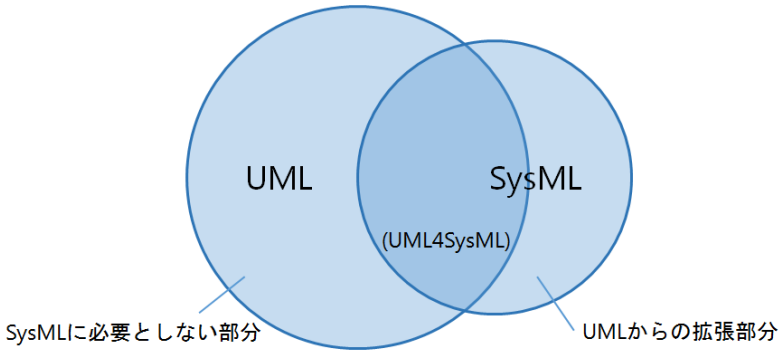

図 3-1 SysML と UML の関係［SysML 2011(改)］

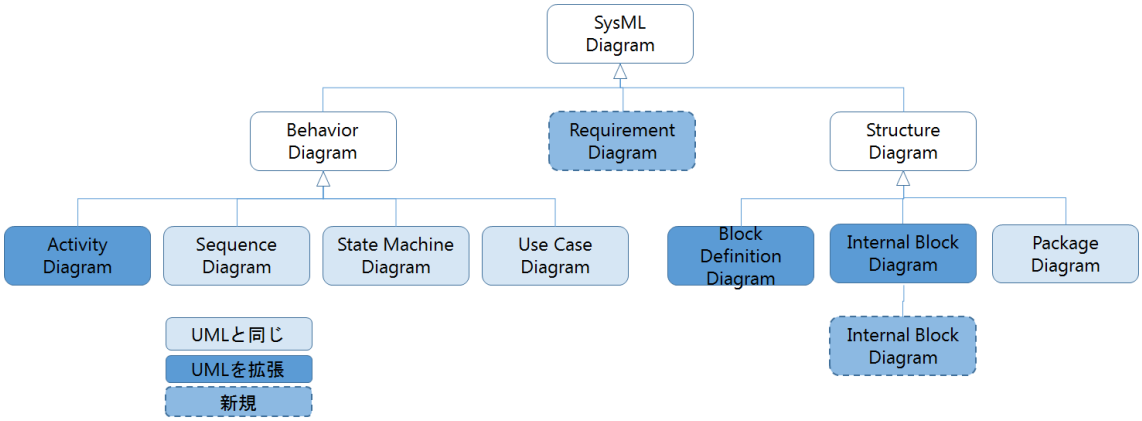

図 3-2 SysML ダイアグラム種類［SysML2011(改)］

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ある．SOA は，システムを柔軟に変更可能にするため，また，再利用性を高めるため，アプリケーションを構成するソフトウェアをコンポーネント化し，それらを組み合わせてシステムを作る設計手法である．これを体系付けるため，参照アーキテクチャが提案され，各レイヤでのモデル化対象が定義されている（図 3-3）.

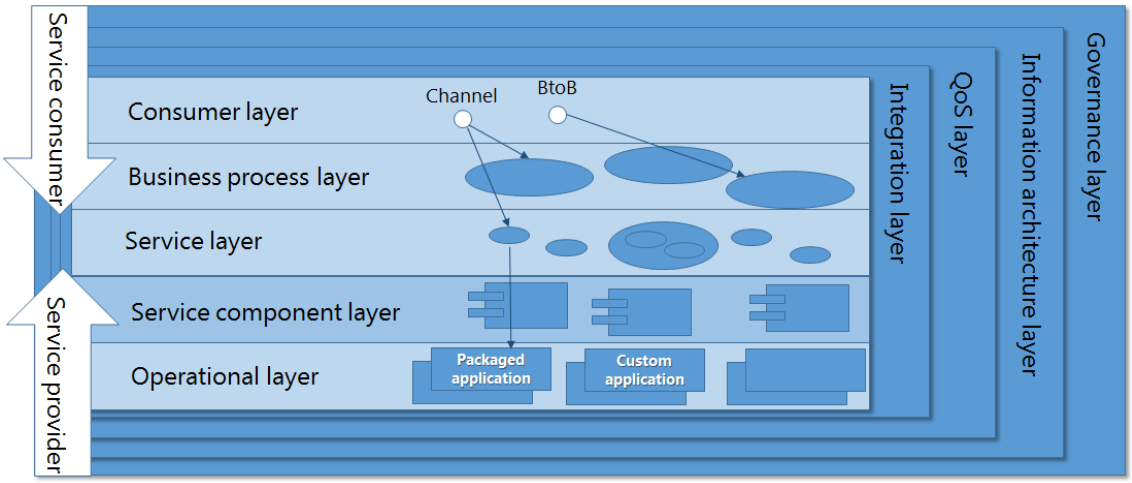

図 3-3 SOA 参照アーキテクチャ：ソリューションスタック［Arsanjani 2007(改)］

以上に示したように，IT サービスの研究は，これまで主に Web サービスの実装方法論としてサービス指向アーキテクチャ(Service Oriented Architecture: SOA)に焦点があり，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について議論が中心になされてきた．近年は，より上位概念を含む業務モデル［IIBA2009］や，複数のサービス間の関係まで考慮したシステム構成方法［Zhang 2007］にも議論が拡げられている．しかし，設計～運用に跨ったプロセス全体を規定するに至っていない．また，マーケティング視点での設計方法［Shostack 1982］［Ramaswamy 1996］も，具体的な生産方法まで詳細には言及されていない．

一方，サービス工学では，Service CAD［Arai 2004, 2005］［Hara 2006］［Kimita 2009］により，サービスシステムの概念設計を計算機上で可能にしている．サービス工学では，ビューモデル，フローモデル，スコープモデルの 3 つの主要なサブモデル概念を導入している．ビューモデルは，顧客に対する価値提案の内容を，機能・実体・属性の構造として表現し，可視化したモデルある．フローモデルは，サービス提供に関わるパートナーのネットワーク関係を表現するモデルである．スコープモデルは，ビューモデルのセットにより，特定の提供者・需給者間の関係を表現するモデルである．これらサブモデルは，サービスの提供時点の構造を，特定の視点に関する情報を含む形でモデル化することで，要求から機能および非機能要件に，更にその仕様定義，実装，機能・非機能へのリソースの割り当てまで，設計～運用プロセスを定式化することまで，現状扱われていない．また，サービス提供プロセスの表現に，サービスブループリントがある．これは顧客とのフロントエンドからバックエンドまでのサービスの機能提供の構成を表現するモデルである．サービス工学では，従来のブループリントを拡張し，人の活動と製品の挙動を統合的に記述可能にしている［Hara2009］．しかし，サービスブループリントは提供時のプロセスに視点があり，設計～提供に至る開発プロセスのモデル化と，プロセスを進行させるための手順

については十分にモデル化できない.

以上に示したように，既存のモデリング方法では，開発プロセスのある工程および提供時点で必要となる視点・観点を形式的に表現できるようにしたものである．従って，開発プロセスに沿って，連続性のあるモデリングと，モデルを跨ったデータ間の関連性についての考察が十分になされていない.

3.2.2 サービスモデリングの課題

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ソフトウェア・ハードウェアは，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，ソフトウェア・ハードウェア機能の実体がクラウドサービスプロバイダに管理される．このように，クラウド環境の浸透に伴い，ソフトウェアおよびハードウェア機能の担い手（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，Web アプリケーションソフトウェアを開発し実行リソースの準備を行う「システムインテグレータ」，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 サービスプロバイダ」，Web サービスを収容する「クラウドサービスプロバイダ」，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 3-4）.

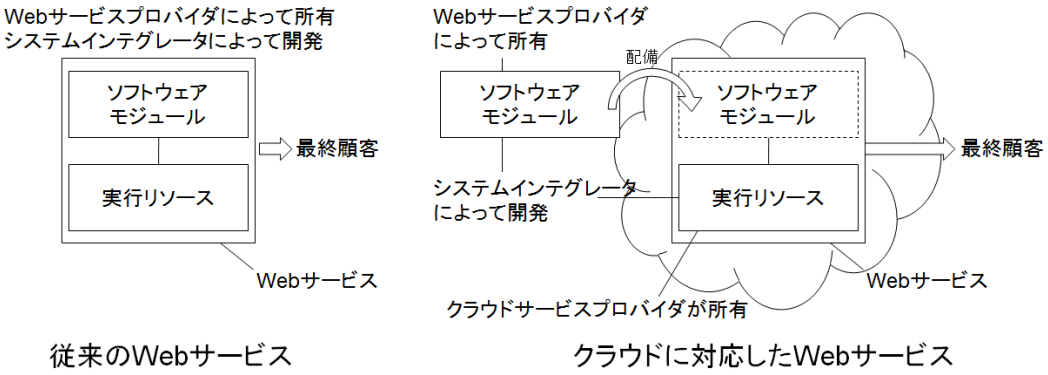

図 3-4 サービスコンテンツの所有権［細野 2013a(改)］

開発プロセスに沿って詳細に見ると，従来，設計・構築工程ではアプリケーション（担体）と，アプリケーションの実行リソース（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 サービスプロバイダ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，開発中のアプリケーションはシステム構築者にあるが，クラウド環境への配置時に，アプリケーションは Web サービスプロバイダに所有権が渡され，アプリケーションを動作させるための実行リソースはクラウドサービスプロバイダに委ねられる．結果，クラウドサービスプロバイダは，クラウド環境に異なる Web サービスプロバイダのアプリケーションを収容するが，アプリケーションの実行リソースの状態は，直接アプリケーションの挙動に影響するため，アプリケーションの要件に応じてリソースを制

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，協働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リソースの状態を得難く，また，運用時に設計情報を得難いままである．この問題は，設計・構築作業と運用作業に重なりが生じる部分において，双方に情報連携し易くするモデリングで解決できる．そこで，Web サービスの設計時点において，実際に活用できる実行リソースをモデル化し，設計対象に含められる方法が必要である．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リソースを確保する手段が必要である．

ソフトウェアを実行するリソースも仮想化され，クラウドサービスプロバイダにソフトウェアと別に機能提供されるようになると，ソフトウェアの機能を発現するにはリソースもモデル化し，ソフトウェアとリソースを包括して Web サービスの機能を設計する必要がある．しかし，これらは別のステークホルダによって管理され，更に，ライフサイクル上，異なる工程で管理されるため，ソフトウェアとリソースを包括して扱うことが困難である．現状では，その手段が無く，実装したソフトウェア機能モジュールをクラウドに配備して，機能を発現しているかを確認している．

そのため，サービスのモデリングにおいて，リソースをモデル化し，それを機能モジュールのモデルと統合できることが，特に必要となる．

以上に示した通り，モデリングの課題は次の通りである．

1. **実行状態に基づいて要件定義を行うこと**
2. **アプリケーションコンポーネントに実行リソース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リソース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**

これにより，設計・構築工程においては，クラウド内の実行リソースの構成をモデル化し，運用工程においては，アプリケーションの業務要件をモデル化し，それぞれ設計・管理可能になる．このモデルを用いて，特に，従来のモデリングでは網羅仕切れなかった領域（図 3-5 破線矩形）の，上記 4 つの設計・管理方法を実現し，ライフサイクルの管理を可能にする．

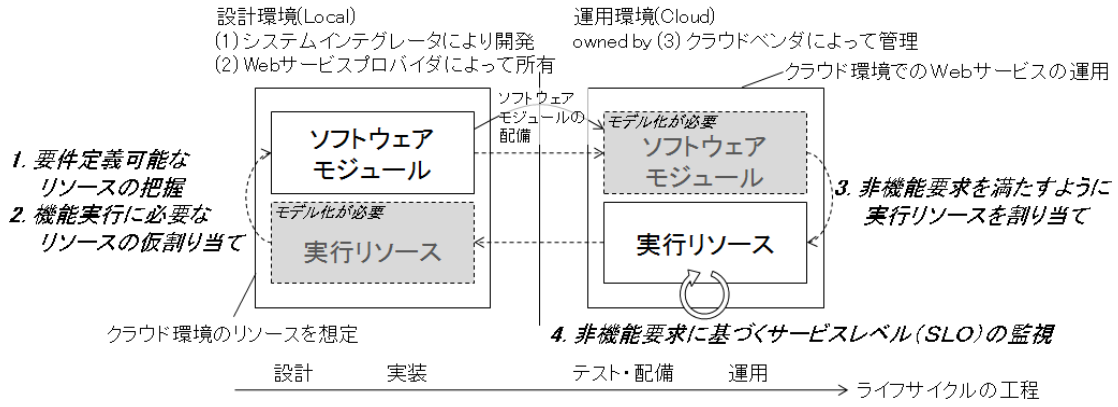

図 3-5 開発環境と運用環境のモデル連携［細野 2013a(改)］

3.3 設計プロセスにおける依存関係の単純化方法

設計とは，問題解決のための着想や思考を具現化し，検証しながら実体を与えるまでの作業である．吉川は，設計過程は概念操作であるため，操作可能な表現を定めることが必要と指摘している．吉川の一般設計学では，設計公理を示した［吉川 1979］．抽象概念を操作することで，人間は新たな人工物を創造できる．この抽象概念の操作を，位相空間を用いて定義する．機能空間の要素を，属性空間に写像する概念で，設計解を求める作業を表している．本研究は，要求から機能，機能から実体を与えるまでの過程を通じて，一連のサービスモデリングを行う．この機能空間から属性空間に写像する考え方で，このモデル間の情報の連携をとる．

本研究では，特に非機能要件とその属性空間への写像を行う．その実現方法として，DSM［Browning 2001］［Steward1981］および DfX（Design for X）［Paul 1996］の考え方を応用し，その統合により依存関係を単純化していく方法を議論する．更に，次節以降で，公理的設計の考え方に基づき，モデル間のパラメータの転写関係を規定し，設計プロセスの過程を単純化する方法を議論する．

3.3.1 DSM に基づく依存関係の整理と単純化

DSM（Design Structure Matrix）は，マトリクス表記により，工程や組織の設計において依存関係を整理して単純化する［Eppinger1997,2012］［Whitfield 2002］．DSM は，アクティビティベース，チームベース，コンポーネントベース，パラメータベースを代表とする，ある観点でシステムの依存関係を見出すものである．

生産の各過程での設計は，DSM により単純化できる．図 3-6 に，あるシステムをツリーで分解表現した構造と，その各要素の関係をマトリクスで表現した DSM を示す．この DSM は，機能やモジュールの依存関係をモデル化したものである．縦軸と横軸に機能名やモジュール名を並べたマトリクスを用い，依存関係がある交差点にマーク付けを行う．このマトリクス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を見出せる．

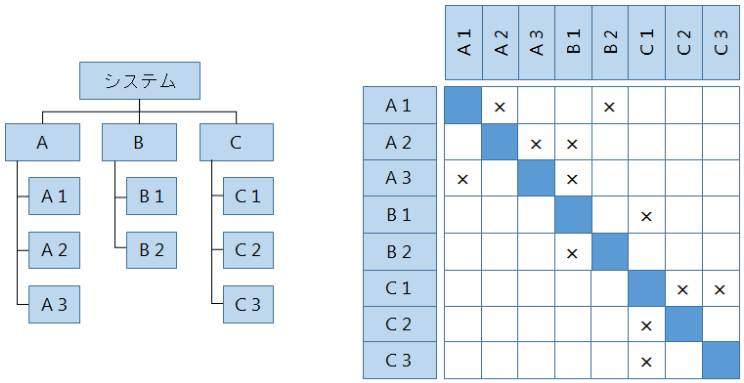

図 3-6 システムの分解構成と DSM［Eppinger2012(改)］

3.3.2 DfX に基づく複数観点の統合

DfX（Design for X）[Paul 1996]は，品質やコストなど，多様な観点を設計過程に取り込む方法である．大富は，DfX とは，製品開発において企画から設計に移行する際，論理的にプロジェクトの性質を解析し，それを踏まえて焦点を定め，以降の開発活動に有効な個々の設計手法（DfX の X）を選択し，投入計画を立てる活動，としている［大富 2005］．

このように複数の観点を一般技術者が考慮して設計を進め，他の技術者と客観的に評価できるような方法が必要になる．この複数の観点は，サービス設計において，性能や安全性などが挙げられる．このような観点は，ソフトウェア工学分野では，非機能要件と総称される．本論文では，この非機能要件を明示的に扱い，機能要件と非機能要件に区別して，サービスの設計を行えるようにする．

DSM に DfX の考え方を組み合わせることで, DfX をモデリング方法の中で実践できる．

3.4 工程間の情報連携を実現するモデリング方法

3.4.1 実行リソース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件（Functional Requirements; FR）と，性能，保守性，拡張性，セキュリティ等の非機能要件（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，クラウド内のリソース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，クラウドサービスプロバイダが豊富な物理リソースを，利用者毎に仮想リソースとして分割提供するが，その内部構成は他の Web サービスのハードウェアリソース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，実際の実行リソースの性能諸元は，他の仮想リソースの影響を受け，クラウド内の実行リソースの状態や振舞いは，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 Web サービスが同時にストレージに書込み処理を行う場合，リソース競合が起こり，それぞれの Web サービス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，ストレージなどのリソースの論理的な設計を行っても，実際に配置，実行して実行リソースの状態や振舞いの確認が必要となる．

そこで，これらの実行リソースの状態をモデル化するために，要件を，実行リソース属性とその取り得る属性値の組み合わせに変換し，これらのリソースの状態を監視する仕組みを導入する．更に，リソース全体の状態に合わせて，前述の属性値の範囲を補正する仕組みをを備える．これにより，実現可能性が不明確な「要求」を，実現可能性のある属性と属性値を持った「要件」に定義し直すことができる．

具体的には，実行リソースを監視し，非機能要件の項目とそのレベルに対して，対応するリソース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 時間などのリソースの組み合わせとしてルール化する．このルールを用い，図 3-7 に示すように時点における利用可能なリソース範囲を都度最新の状態が把握可能になる．これにより，システム構築者が，要件定義を行う際，実行リソースの振舞いや利用可能な性能範囲を適切に設計できる．この具体例として，要件定義画面を図 3-8 に示す．図の右上のダイアログは，リソースカタログで，時点のリソースの状態に基づき，非機能要件を満たすリソースの組み合わせとリソースの設定範囲を提示する．

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リソース量や，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，Web サービスの実装・運用工程で生じ得る実行リソースの制約を，予め設計工程で考慮できる．そのため，運用工

程に入り Web サービスを実行リソースに配備した段階で実行リソースの不足が判り，改めて設計変更を行うなど，手戻り作業も低減し得る．ここで定義した要件を用いて，後の運用工程で非機能要件の充足を確認できる．

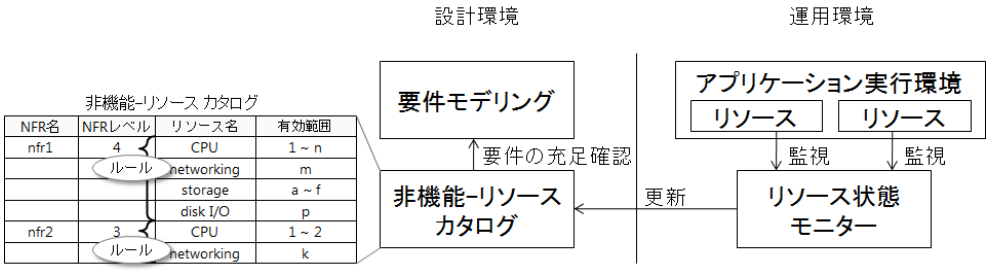

図 3-7 リソースカタログの実践例［細野 2013a(改)］

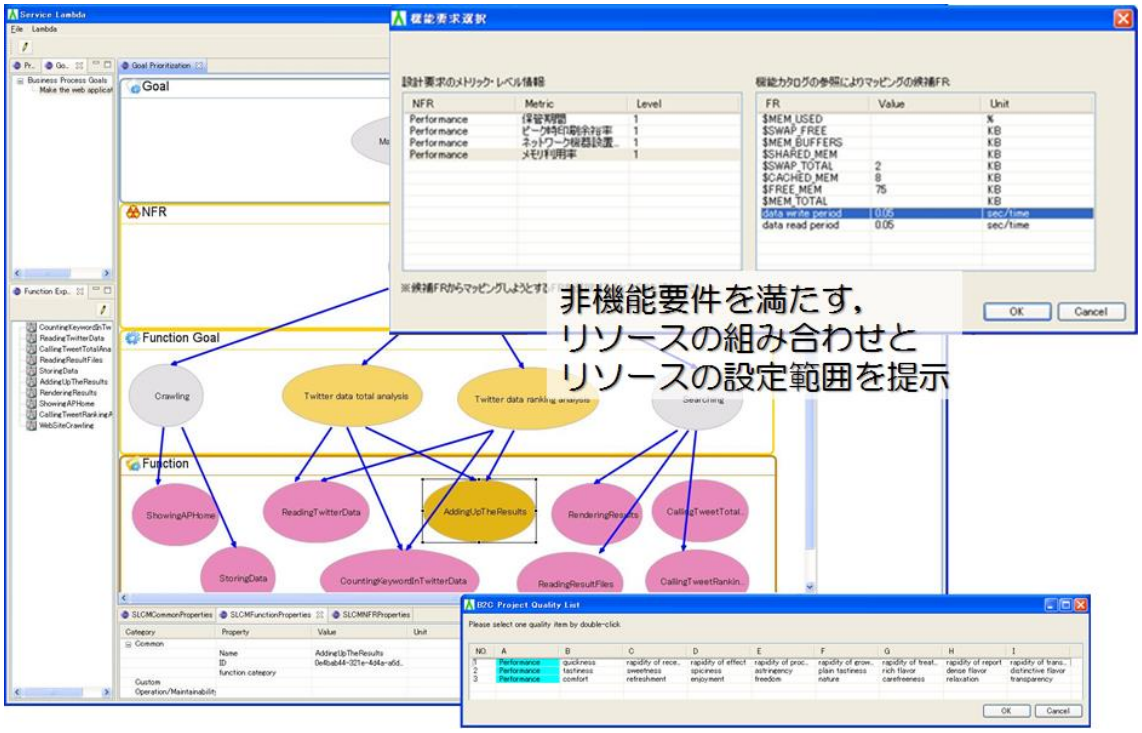

図 3-8 要件定義画面の実践例［Hosono2012b(改)］

3.4.2 機能モジュールおよび実行リソースの割り当て設計

3.4.2.1 アーキテクチャ設計の３ステップ

クラウドの利用を前提とした環境では，アプリケーションソフトウェアの機能や，アプリケーションを実行するミドルウェアの機能，またはオペレーティングシステムやネットワーク，ストレージなどのインフラストラクチャの機能を利用し，Web サービスを構築する．このクラウド環境を実現する主要な要素技術は，サーバやストレージの仮想化技術で

ある．この仮想化技術によって，CPU やハードディスクなどの物理的なハードウェアリソースと，仮想的に割り当てられる CPU コア数やストレージ量など論理的なハードウェア機能が分離される．これらの分離されたモジュールをインタフェースとして，インタフェースを組み合わせて，Web サービスを構成することになる．

従来の Web アプリケーション開発においては，物理的・論理的の区別なく一体として扱ってきた．一方，クラウド環境でのアプリケーション開発は，分離された部分がアプリケーション構成を作るためのインタフェースとして切り出されるため，従来に比べて設計インタフェースの数が増える．

アプリケーション設計は，性能や保守性を高めるために，基本的な設計指針として，複数の設計インタフェースのモジュール化を行う．仮想化・クラウド環境下 においては，ハードウェアとソフトウェアが分離独立されるため，アーキテクチャ設計において，両者を同時に考慮することが必要である．

また，クラウド環境でのアプリケーションは，クラウドを提供するデータセンタ内に配備されたサーバ群やストレージ群の中から，仮想化された機能を呼び出している．クラウドアプリケーションは，アプリケーションを構成するモジュールが，複数の仮想マシンに配置されることや，リモートのデータストアを利用することから，分散システムとなる．このアプリケーションの設計方法は，これらの仮想化された機能がどの物理的なハードウェアリソース上で動作するか，十分に考慮できない．これは，複数の機能間，あるいは複数のアプリケーション間で，同一の物理的なハードウェアリソースを共有する可能性があり，結果として，期待される機能の性能設計が困難なためである．

そこで，機能モジュールおよび実行リソースの割り当て設計を分離して行う．機能モジュールの設計を行う際，モジュールの持つ機能の独立性の単位で分離を図る．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件（Non-Functional Requirements; NFR）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 3-9）．これにより，機能要件，非機能要件から実行リソース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，利用が進むにつれ，いずれ更改される．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ソフトウェア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．

この設計ドメインの抽象化により，設計プロセスは，次のステップに整理できる（図 3-9）．

Step1 ソフトウェアコンポーネントの最適分割設計

Step2 ソフトウェアコンポーネントの仮想サーバへの最適リソース割り当て設計

Step3 仮想サーバの実体サーバへの最適リソース配備

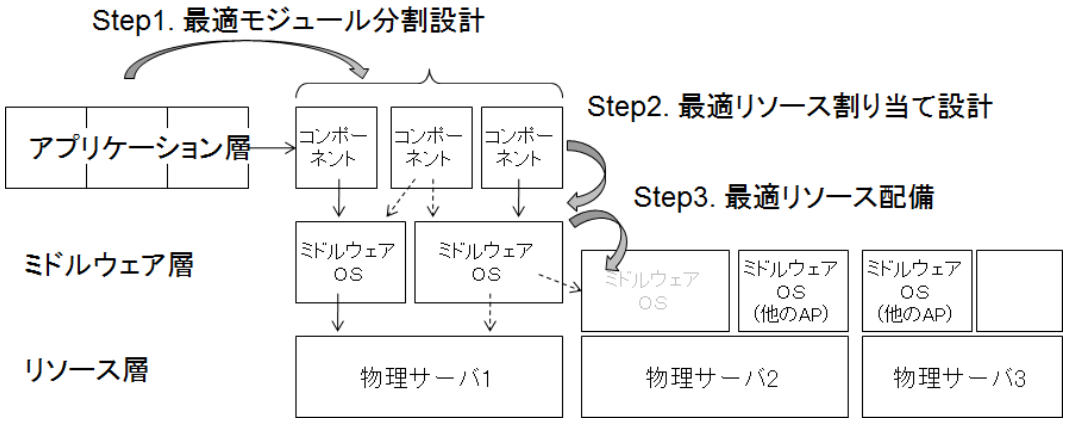

図 3-9 アーキテクチャ設計の 3 ステップ［Hosono2013a(改)］

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 はクラウドサービスプロバイダに必要な情報で物理的な振舞いの検証である.

3.4.2.2 機能モジュールの最適設計

Step 1 として，アプリケーションを構成するソフトウェアコンポーネントの粒度や，ソフトウェア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，機能モジュール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マトリクスを用い，依存関係のある交差点にマークを並べたマトリクスを用い，依存関係がある交差点にマーク付けを行う．マトリクス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．

ソフトウェアコンポーネントとその構成要素である API（Application Interface）は，機能モジュールの機能と等価と見做せるため，この DSM を用いて，適切な粒度のソフトウェアコンポーネントを設計する．

3.4.2.3 非機能要件を考慮した実行リソースの割り当て設計

Step2 として，機能モジュールを実行リソースに割り当てる．このとき，機能モジュール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，実行リソース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，マトリクスの行・列の交点に機能の呼び出し関係の有無を記載する．例えば，ソフトウェアモジュール群に含まれる機能群を，それぞれマトリクスの行および列に並べ，あるソフトウェアモジュールの機能 a が，別のソフトウェアモジュールの機能 b を呼び出す場合に，呼び出し関係があるとする．この行・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することで，独立性の高いモジュール設計を行う．

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 3-10 に，機能 1～機能 10 の関係で表記した具体例を示す．ここでは，API など機能モジュール（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 3-1）．この例では，直接的依存と間接的依存にそれぞれ 0.5 ずつ配分している．間接的依存は，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．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リソースへの割り当て方を客観的に評価できる．

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

図 3-10 拡張した DSM

表3-1 依存性に関するDfX［Hosono2013a(改)］

	重要度	モジュール1
機能呼び出し（直接依存）	0.5	-
性能	0.3	機能1，機能2，機能3
安全性	0.2	機能1，機能2，機能3

以上に示した機能モジュールの最適設計，および非機能要件を考慮した実行リソースの割り当て設計を行う詳細手順を以下に示す．

1. アプリケーションを構成する機能群と，機能間の呼び出し関係を依存関係として入力する
2. 次に，データトランザクションに関する性能要件などアプリケーションのロジックとして特定の機能間に関連がある場には，これらの非機能要件を入力する．
3. 上記の情報を元に，マトリクスを表示する．
4. 再配置計算を行う場合は，前記の情報を用いて再配置計算を実行する．行・列の配置変換によるモジュール化は，マトリクスを手動で操作する．あるいは，行・列の数が多い場合，例えば次のように遺伝的アルゴリズムを利用して自動計算を行う．
   ① 一定数の染色体オブジェクトを作成する．染色体オブジェクトは，マトリクスの個々の行を示す．例えば，{4, 5, 1, 3, 2}のように表現する．
   ② ルーレットルールを用いて，2 つの染色体オブジェクトを選択する．
   ③ 事前に与えた確率に基づき，染色体オブジェクトに突然変異を発生させる．
   ④ 各染色体オブジェクトに対して，適応度の評価関数を実行する．終了条件を満たした場合，処理を停止する．満たしていない場合は，その高い値をもつ染色体オブジェクトを選択し，処理②に戻る．

   次に，再配置計算後の機能モジュール設計マトリクスを表示し，機能モジュールの構成を提示する．
5. 前記で決定したアプリケーションのモジュール構成に対し，モジュールを配置するリソースを決定するステップを実行する．
6. まずアプリケーションを構成するモジュール群と，モジュール間の呼び出し関係を依存関係として入力する．
7. 次に，可用性・保守性・移行性・安全性など非機能要件の観点で，特定のモジュール間に関連がある場合には，これらを間接的な依存関係として入力する．
8. 前記の情報を用いてマトリクスを確認する．表示されたマトリクスに対し，情報の十分性を確認する．見直し不要で，モジュール配置が適正である場合は，終了する．
9. 再配置計算を行う場合は，前記の情報を用いて再配置計算を実行する．行・列の配置変換によるモジュール化は，マトリクスを利用者が手動で操作する．あるいは，行・列の数が多い場合には前述と同様に遺伝的アルゴリズムを利用して自動計算を行う．次に，再配置計算後のモジュール配置設計マトリクスを表示し，機能およびリソース構成モジュールの構成を提示する．
10. アプリケーションのモジュール配置構成に対し，実際に割り当てるリソースを決定する．

以上の通り，Web サービスの機能モジュールとして，「アプリケーション」と「実行リソース」のそれぞれを設計する手順を示した．この手順により，ソフトウェア機能のモジュール化，およびアプリケーションの論理的・物理的な配置場所の設計が容易となる．

以上に示した方法を行うモデリングと，モジュール分割の実践例を示す（図 3-11）.

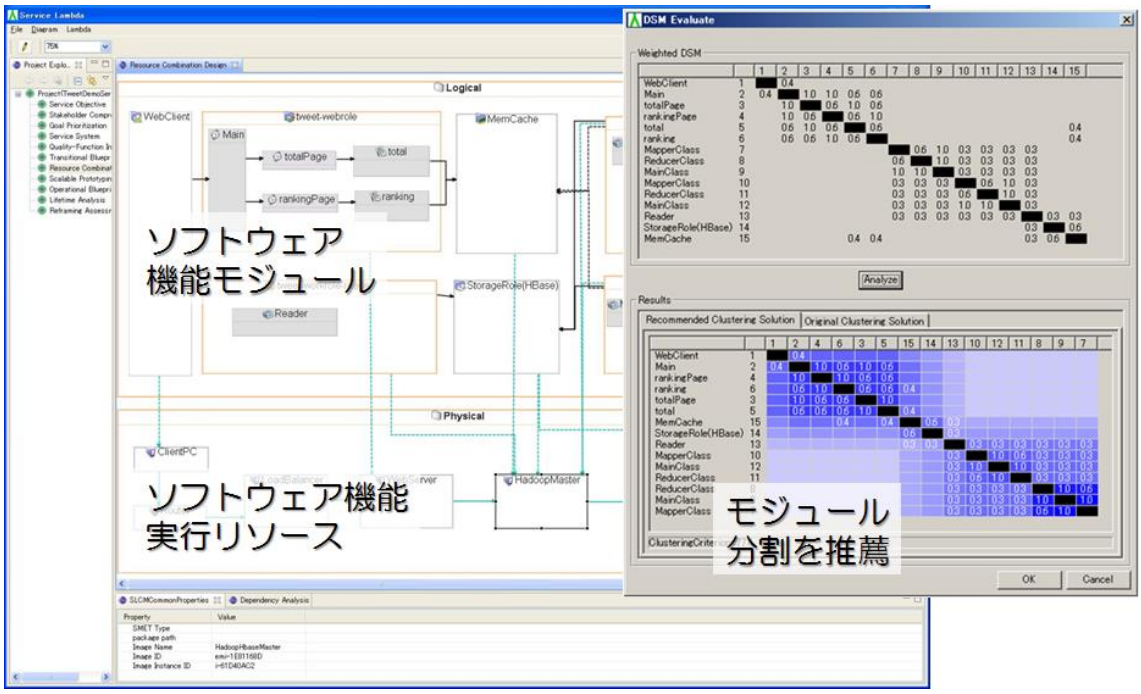

図 3-11 アーキテクチャ設計画面の実践例

本節に示した方法により，ソフトウェア機能のモジュール化が判断しやすくなる効果が得られる．これは，機能モジュール設計マトリクスにより，性能を考慮した機能分割が容易と成るからである．また，モジュール配置設計マトリクスにより，拡張性や保守性を考慮したモジュールの配置方法が分かるためである.

また，アプリケーション配置場所が判断しやすくなる．これは，リソース割当設計マトリクスにより，仮想マシン間の依存関係が分かり，運用管理者が，クラウドサービスで用いる仮想マシンをどの物理マシン上で動かすのか，計画を立て易くなるためである.

3.4.3 実行リソースの分離と結合による機能・非機能検証

3.4.3.1 実行リソースの分離

クラウドでは大規模データを処理し易い豊富なハードウェアリソースを持ち, 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，実行リソースには，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．このような実行リソースを要するアプリケーションを検証するため，システム構築者の開発 PC 内に実行リソース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．そこで，機能は仮想的な実行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 サービス実行リソース（担体）の仮想環境であり，この上で機能の動作検証を可能にする．これにより，実行リソース（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アプリケーションのみを検証できる（図 3-12）．以上により，アプリケーションの試作中に，アプリケーション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，アプリケーション検証効率の向上効果も得られる．

3.4.3.2 実行リソースへの割り当て

本節では，3.4.2.1 節で示した Step3 を実行する．アプリケーションの機能検証をローカル PC で行う．ローカル PC にクラウドのリソースを代替するソフトウェア（ミドルウェア）およびハードウェアを用意し，これらのリソースにアプリケーションソフトウェアモジュールを仮想的に割り当てて，機能検証を行う．

機能検証を行った後，アプリケーションソフトウェアモジュールをクラウド環境にアップロードし，クラウド内の実際のリソースに割り当てて配備する．このとき，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，同じハードウェアリソースやネットワークリソースにデータを書き込むなど，仮想サーバ間の影響が生じる．

そこで，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（図 3-12）．

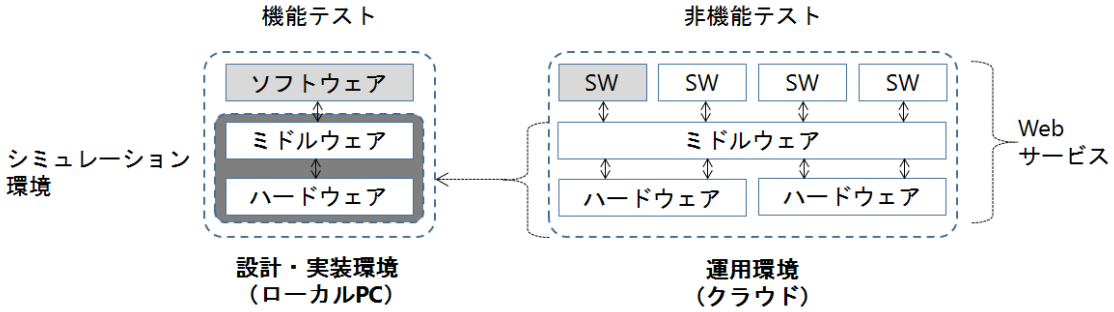

図 3-12 実行リソースへの割り当て

図 3-13 に，機能要件検証を行う環境の例を示す．これは，ローカル PC 上で実行されるもので，ソフトウェアモジュールの機能検証を行う．同様に，図 3-14 に，非機能要件検証を行う環境を示す．これは，Web サービスの実提供環境であるクラウド上で実行されるもので，ソフトウェアモジュールにクラウド環境のネットワークやハードウェアリソースを割り当てることで，応答性能に関する要件など，非機能要件の検証を行う．

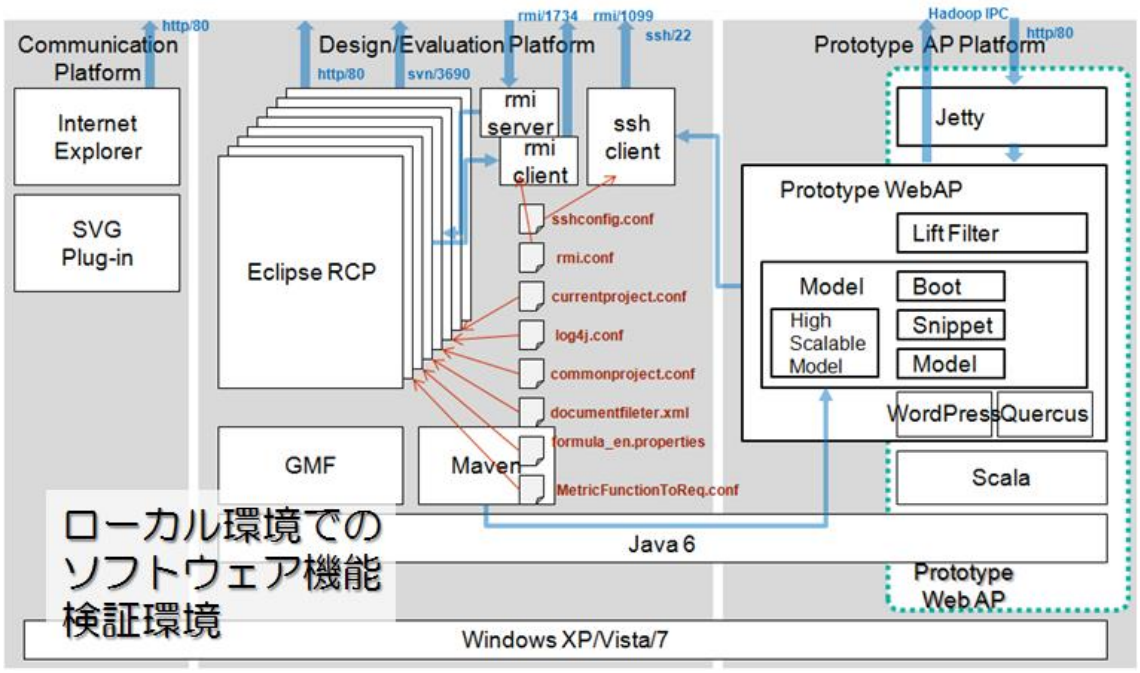

図 3-13 機能要件検証環境の実践例

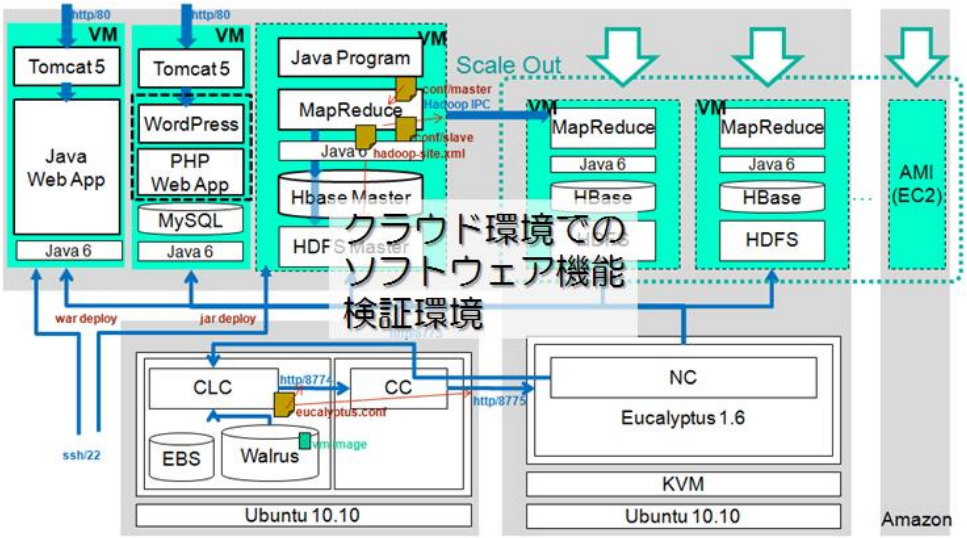

図 3-14 非機能要件検証環境の実践例

3.4.4 非機能要件に基づく実行リソースの監視

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 時間の稼動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，記録媒体の書込み速度など，複数の計算機リソースの組み合わせで実現される．特に，これらの業務レベルとリソース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リソースレベルの監視対象・監視値（Service Level Objectives; SLO）に変換し，実際に割り当てるリソース属性を決定する．同時に，SLO に対する監視を開始する．以上により，再びアプリケーションのロジックと実行リソースを「結合」して，Web サービス提供者およびクラウドサービスプロバイダは，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 サービスの本番稼動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．

この監視機構の例を次に示す（図 3-15）．

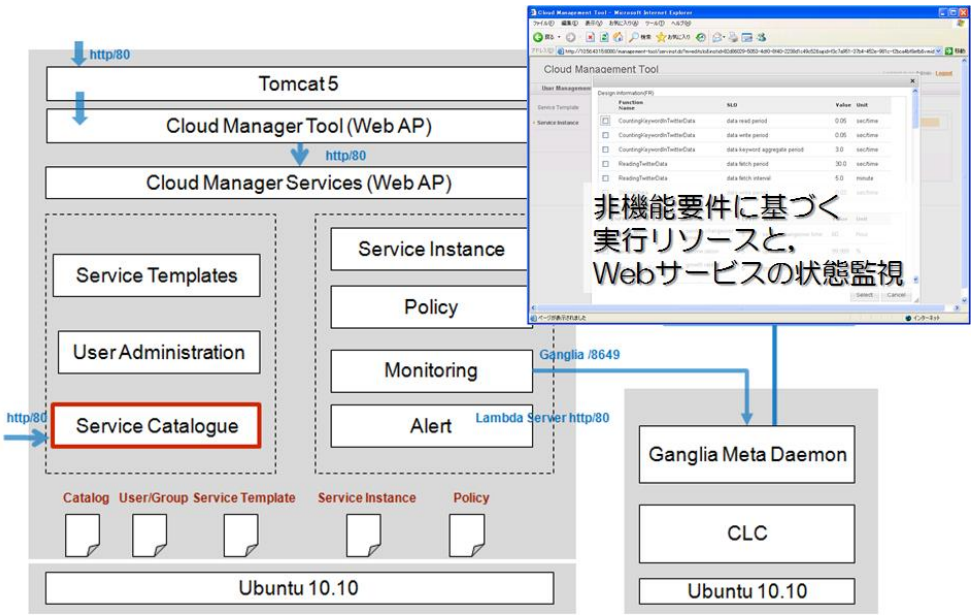

図 3-15 非機能要件充足監視の実践例

以上に示した，(1) 実行状態に基づいて要件定義を行うこと，(2) アプリケーションコンポーネントに実行リソース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リソース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 つの方法を実行する，具体例を図 3-16 に示す．

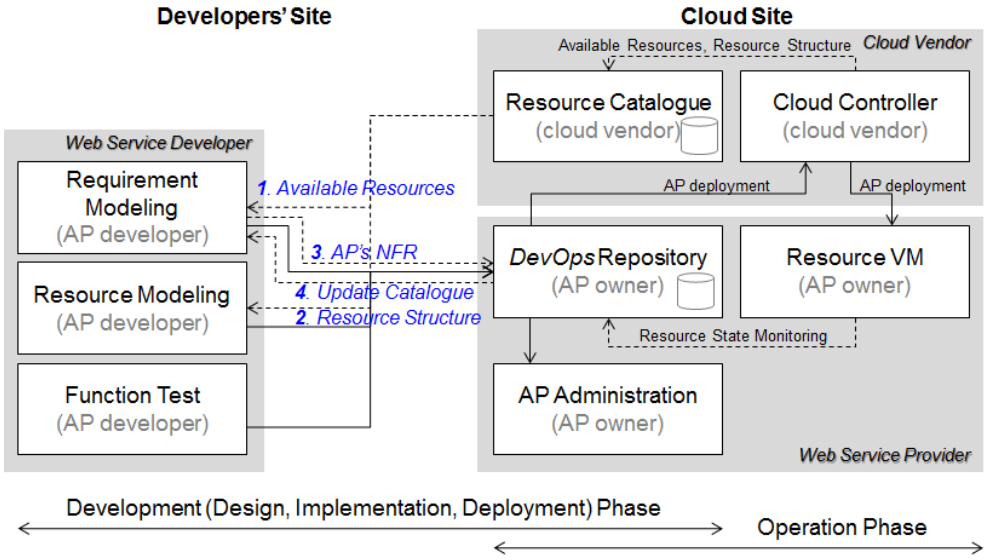

図 3-16 開発環境と運用環境のモデル連携の例

本節では，4 つの方法により，ライフサイクル工程を網羅した．しかしながら，設計～運用まで，より細かな役割の開発者が関わり，各々，より詳細な情報を扱う必要がある．そこで，4 つの方法を補完する，細粒度のモデリングが必要となる．次節では，これらのモデリング方法について述べる．

3.5 細粒度のモデリングとサービスモデルチェーンの構築

3.5.1 細粒度のモデリング方法

3.5.1.1 細粒度のモデリング

前節に示したモデリング方法は，ライフサイクルの工程を大きく区切ったため，サービス開発部門の個々の役割に応じて，更に観点を詳細化する必要がある．そこで，次のように前節で示したモデリングを補完する，細粒度のモデリングを行う．このモデリングは，サービス設計者，アプリケーション実装者，サービス管理者，クラウド管理者のそれぞれが，その役割に応じて扱う情報を扱うものである（図 3-17）．

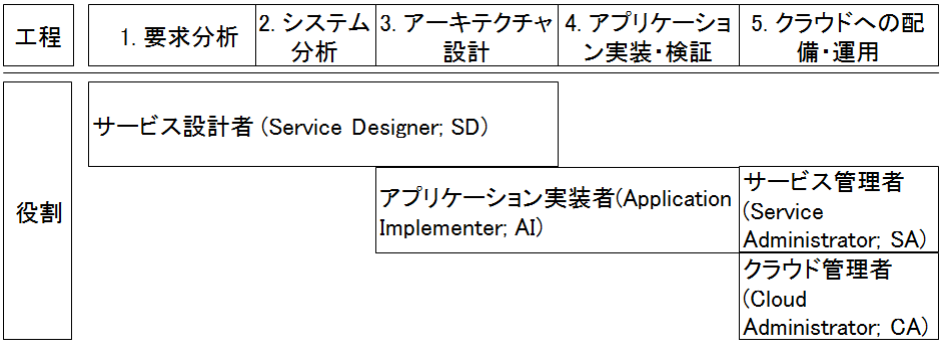

図 3-17 ライフサイクルに関わる開発者の役割

これらの役割を持つ開発者が，モデリングする内容を（表 3-2）に示す．サービス設計者は，目的から機能・非機能要件への展開，およびサービスシステム境界の設計を行う．アプリケーション実装者は，機能モジュールの設計および機能・非機能要件の実現検証を行う．サービス管理者およびクラウド管理者は，機能・非機能要件が満たされているか否かを監視・評価する．

表 3-2 細粒度のモデリング

モデリング観点	細粒度のモデリング	説明
目的から機能・非機能要件への展開	サービス目標モデリング	目的を構造化する
	ステークホルダ抽出モデリング	バリューチェーンを設計する
	要求・機能展開モデリング	目標から機能への転写を行う
サービスシステム境界の設計	サービスシステムモデリング	サービスシステムの境界を同定する
	機能・品質モデリング	機能と品質の関係をモデル化する
機能モジュールおよび実行リソースへの割り当て設計	ユースケースモデリング	ユースケースを定義する
	リソース割り当てモデリング	機能モジュールから，実行リソースへの割り当て設計を行う
リソースの分離・結合による機能・非機能要件の実現検証	プロトタイピング	アプリケーションのプロトタイプを実装する
	非機能要件監視・評価	アプリケーションのプロトタイプを実行する

表 3-2 に示したリストを実現するモデリング方法の一覧を図 3-18 に示す.

図 3-18 細粒度に分割したモデリング方法

以上の細分化したモデリングにより，各開発者の役割がそれぞれ詳細な情報をモデルに反映させ，工程間で漏れなく情報を構造化できる．以降の節では，各モデリングの概要と，図 3-18 中にハイライトした,サービスの開発に特徴的なモデリング方法について説明する．（従来のソフトウェア開発においても必要としたモデリング方法は目的の説明に留める．）

3.5.1.2 サービス目標モデリング

サービスの目的と，それを実現するための要求の関係を構造化する．バランススコアカード（BSC）の考え方に沿って，財務の視点，顧客の視点，イノベーションと学習の視点，内部業務プロセスの視点のそれぞれ領域での要求と，これらの要求間に関連のあるものをリンク関係で構造化する．内部業務プロセスの領域を下位レイヤとして，目的から，内部業務プロセスの要求までの関係を構造化する．この内部業務プロセスの要求を，後工程で，機能に展開する対象にする．

3.5.1.3 ステークホルダ抽出モデリング

サービスの内部業務プロセスに，携わるステークホルダを抽出し，ステークホルダ間の機能提供関係から，バリューチェーンを構築する．

顧客企業の戦略，ビジネスプロセス，開発プロセス，ステークホルダと関連付ける．ステークホルダには業務プロセスや開発プロセスに直接関わるステークホルダと，業務プロセスや開発プロセスには直接関わりがないものの，戦略や要件に関係し，意思決定・レビューを行うステークホルダがある．直接的に関わるステークホルダに加えて，戦略の重要視する観点との適合度を用いて間接的に関わるステークホルダの抽出を行う．[久野，細野 2009]

3.5.1.4 開発ブループリント

サービスの企画から構築工程までのプロセスにおいて，ステークホルダ抽出モデリングで得たステークホルダ間の機能提供関係をモデル化する．これは，ステークホルダをエンティティとして，企画から構築工程までのステークホルダ（顧客およびプロバイダ内の各役割）間の機能提供関係を表記したものである．例えば，上流設計者が作成する要求仕様書は，機能設計者が機能設計書を作成するための情報源となる．これらのステークホルダ間には，要求仕様の情報を提供する機能と，それを受け取る関係，すなわち機能提供関係がある．これらの関係によるモデリングを企画から構築工程までのプロセスに対して行う．

3.5.1.5 要求・機能展開モデリング

要求・機能展開モデリングは，サービス目標モデリングで抽出した要求に対して，目的設定で定義した目的を達成する，機能要件および非機能要件を構造化する．本モデリングは，サービス化の進展（クラウドの浸透）に伴い生じたギャップを埋める，4 つのモデリングの 1 要素である．

3.5.1.6 ユースケースモデリング

サービスを構成するソフトウェアアプリケーションの振舞いについて，詳細にモデリングする．本モデリングは，UML のユースケース図，シーケンス図に準じたモデリング方法である．

3.5.1.7 サービスシステムモデリング

設計対象となるサービス全体構成の範囲を明らかにするため，ユーザの振舞いを起点に，サービスシステム境界の設計を行う．

ユーザがサービスを見つけ，利用するシナリオから，提供する機能を網羅的に定義し，提供に必要な仕組みを設計するには，サービスシステムの範囲を規定する必要がある．そこで，ユーザのサービスを認識し，利用し，評価するまでの体験過程をモデル化し，各過程でのユーザの振舞いと，それに対する機能提供の関係から，サービスシステム全体の境界を決定する．このサービスシステム境界の範囲で，機能設計の最適化や改善を行う．

具体的には，次のようにモデリングを行う．ユーザを，ペルソナを用いてモデル化し，そのサービスの機能を使う過程を構造化する．ペルソナは，サービスレシーバ（受け手）側のユーザペルソナと，サービスプロバイダ側のロールペルソナの 2 種類で定義する．ユーザペルソナは，各工程で分割したユースケースと，その重要度をリストで持つ．一方，ロールペルソナは，ユーザペルソナあるいは他のロールペルソナに提供する機能をリストで持つ．

1. 対象セグメントとなるユーザグループを決定するため，アンケート等を通じて，品質にどのくらい重要性を見出しているか，情報を収集する
2. これらの収集されたデータをクラスタ化し，ビジネス戦略に基づき，対象セグメントに適した特性を持つグループを選択する
3. 多変量解析などの方法により，品質要素に対する重要度を抽出する．これを重要度リストとして保持する
4. 対象としたグループに属する実際の人物に対し，デプスインタビューを行う
5. 重要値との類似度を計算し，最も高い類似性を持っている人を特定する．該当人物から得た情報をユーザのペルソナの補足情報として用い，ユーザペルソナを完成させる
6. サービスプロバイダおよびレシーバ内のステークホルダと，ステークホルダ間の機能提供関係を個々に確定していくことで，サービスの提供機能群を確定し，サービスシステム全体の構成を確定する．
7. 各ユースケースとロールペルソナの機能と 1 対 1 の対応関係をとる．
8. ユーザペルソナのサービスの各体験過程に対し，各ユースケースがロールペルソナの何れかの機能に関連するか否かを全て確認し，関連するものはレシーバとプロバイダの対応表（RPM 表）に入力する．ロールペルソナに対応する機能が存在しない

場合には，ロールペルソナの抽出が不十分と判断する．この場合は，完全に両側が対応するまで，プロバイダ側から提供する機能の抽出を繰り返す．

9. 対応関係が決定している場合，それぞれのユーザペルソナ側の重要度およびロールペルソナの重要度の値を正規化し，ペルソナと RPM 表を完成させる．
10. ユーザペルソナとロールペルソナの各重要度値を比較し，差異の有無を確認する．
11. 差異がある場合，プロバイダは，差異が生じないように機能を再設計する．

サービス開発者は，どれだけ多くのステークホルダを考慮する必要があるか確認することが困難である．しかし，上記のステップにより，機能に必要かつ十分なステークホルダが抽出できる．これにより，サービスシステム境界を決定できる．このサービスシステム境界内が，サービス機能の品質レベルの設計を最適化する範囲となる．

上記に示した手順を実践する構成を図 3-19 に示す. この構成に基づいたモデリング画面の例を図 3-20 に示す. ここに示すように, ユーザがサービスを発見し, 利用の確認を行い, 実際に利用し，評価し，更に，継続利用するか否かを判断する過程がモデル化できる．各過程対し，サービスプロバイダから提供すべき機能を対応付ける．このペルソナモデリングにより，オペレータなどが人手で提供する機能も，オペレータなどが管理し IT で提供される機能も透過的に表現されるため，機能全体を包括したサービスシステムのモデリングが可能となる．

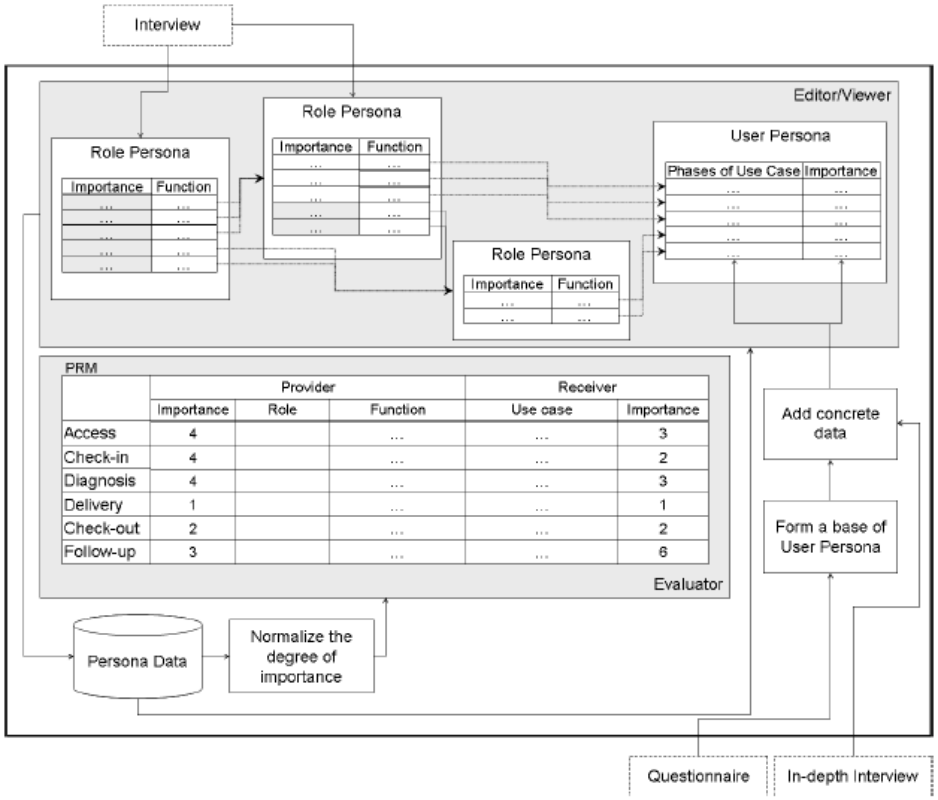

図 3-19 サービスシステム境界の設計の全体

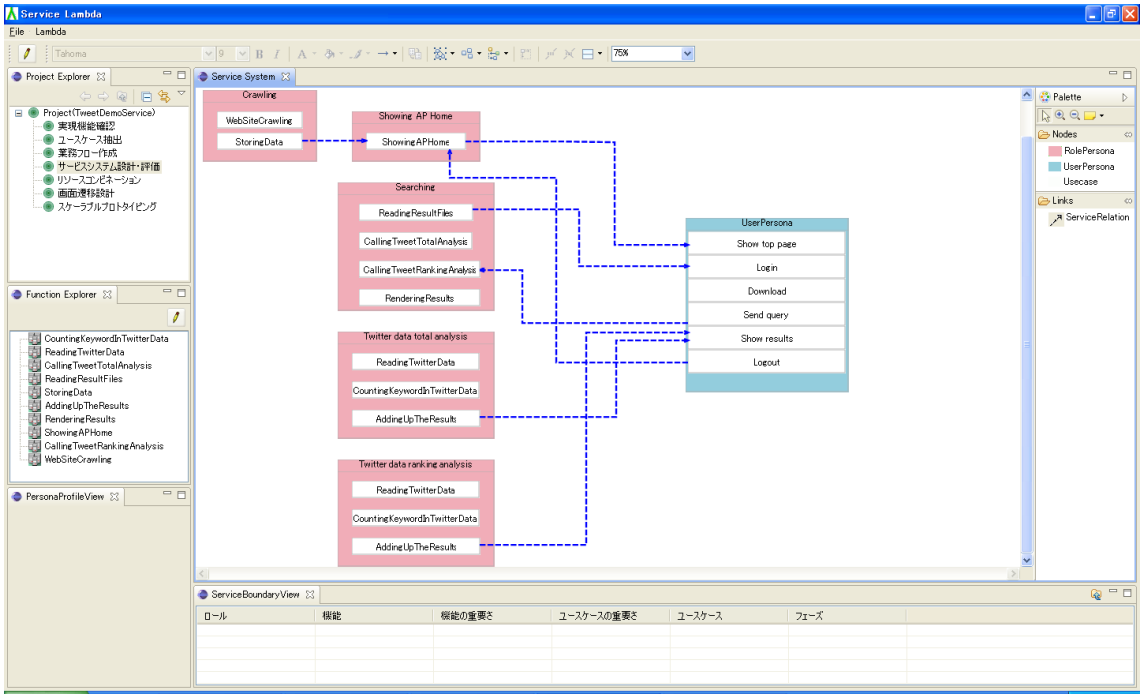

図 3-20 サービスシステム設計画面の実践例

3.5.1.8 品質・機能モデリング

サービスの機能と品質の関係をモデリングする．対象とするサービスが BtoB 向けおよび BtoC 向けの場合，それぞれに対して品質要求テンプレートとなるリストを用意し，これらのリストから，各機能に求められる品質とそのレベルを定義する．品質は，非機能要件と紐づけられるため，このモデルを次の機能再編成モデリングにおいて利用する．

3.5.1.9 機能再編成モデリング

サービスの機能の組み合わせと，個々の非機能要件および提供コストへの影響とトレードオフを可視化し，シミュレーション実行し，最適な機能群の構成を導出するためのモデリングを行う．可視化・シミュレーション実行する際に，前工程で定義した品質・機能モデリングの構造化データを利用する．

3.5.1.10 業務フローモデリング

ソフトウェア開発で一般に行われている業務フロー設計と同様のモデリングを行う．

3.5.1.11 リソース割り当てモデリング

ソフトウェアモジュールを，その実行リソースに割り当てるための設計を行う．本モデリングは，サービス化の進展（クラウドの浸透）に伴い生じたギャップを埋める，4 つのモデリングの 1 要素である．

3.5.1.12 画面遷移モデリング

ソフトウェア開発での画面設計と同様なモデリングを行う. 開発する対象が Web サービスの場合，顧客起点で画面遷移のユースケースに基づいたモデリングを行う.

3.5.1.13 プロトタイピング

Web サービスの機能検証を行う. 本モデリングは, サービス化の進展（クラウドの浸透）に伴い生じたギャップを埋める，4 つのモデリングの 1 要素である.

3.5.1.14 運用プロセスブループリント

サービス提供時における，顧客と提供者間の機能提供関係，および流通など提供者のバックエンドとの機能提供関係をモデリングする． これは，ステークホルダをエンティティとして，運用工程におけるステークホルダ（顧客およびプロバイダ内の各役割）間の機能提供関係を表記したものである.

3.5.1.15 品質要因分析

開発の過程で作りこまれた品質の要因を分析する．開発ブループリントにおいて，各ステークホルダ間の機能提供関係において，要件→機能→リソースへと変換が段階的に行われていく．このとき，機能の受け手側のシステム・ビジネス上の制約が，変換に含まれる場合，後工程での品質劣化要因となる．本モデリングでは，これらの変換の過程をトレースする．このトレースにより，この品質と顧客要件間の関係［Tayleor 1994］の構造を明らかにする.

3.5.1.16 クラウド配備実行

クラウド配備実行は，ソフトウェアアプリケーションモジュールを，クラウド側に配備し，クラウド内のミドルウェアおよびハードウェアリソースと結合させて，Web サービスを実行可能にする．本モデリングは，サービス化の進展（クラウドの浸透）に伴い生じたギャップを埋める，4 つのモデリングの 1 要素である.

3.5.1.17 非機能要件の充足監視

非機能要件の充足監視機構は, サービスの非機能要件が満たされているか, 監視を行う. 本機構は，要求・機能展開モデリングで定義された要件と，実際の実行中の振舞いとの対応を取り，非機能要件を満たした振舞いとなっているか否かを判定する.

3.5.2 モデルチェーンの構築

3.5.2.1 公理的設計に基づく設計プロセスの単純化方法

Suh は，設計過程を設計公理として，表現した［Suh 1990,2001］. 全ての人工物における設計は，「顧客領域」,「機能領域」,「実体領域」,「プロセス領域」の 4 つの領域と，領域

間の写像関係によって説明している（図 3-21）.

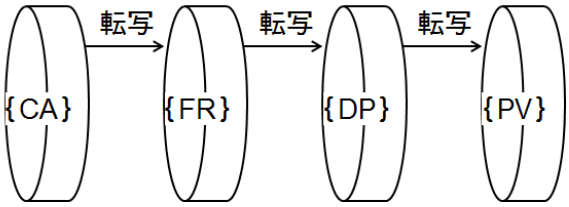

図 3-21 公理的設計における 4 つの設計領域

顧客領域には，顧客の期待を示す顧客属性（Customer Attribute: CA）を記述する．この CA は，機能要件（Function Requirement: FR）に写像できる．同様に，FR は，実体として DP（Design Parameter）に写像できる．更に，この DP は，生産するための変数である PV（Process Variable）に写像できる．以上の転写関係によって，設計工程をモデル化した.

この考え方は，様々な設計対象に普遍的に当てはまるため，これらをサービスの要求定義から機能設計を行う過程に適用できる．機能及び非機能要件は，ソフトウェアの機能および機能の組み合わせ方（アーキテクチャ）に，更に，これらはソフトウェアモジュールの設定パラメータとソフトウェアモジュールを動作させるミドルウェアおよびハードウェア機能に，転写される．これらのミドルウェアおよびハードウェア機能は，物理的なメモリやハードディスクなどのリソースに割り当てられる．以上に示した転写関係は，2 者間の制約関係である．転写が完全に行える場合，図 3-22 のように概念で転写関係を示せる.

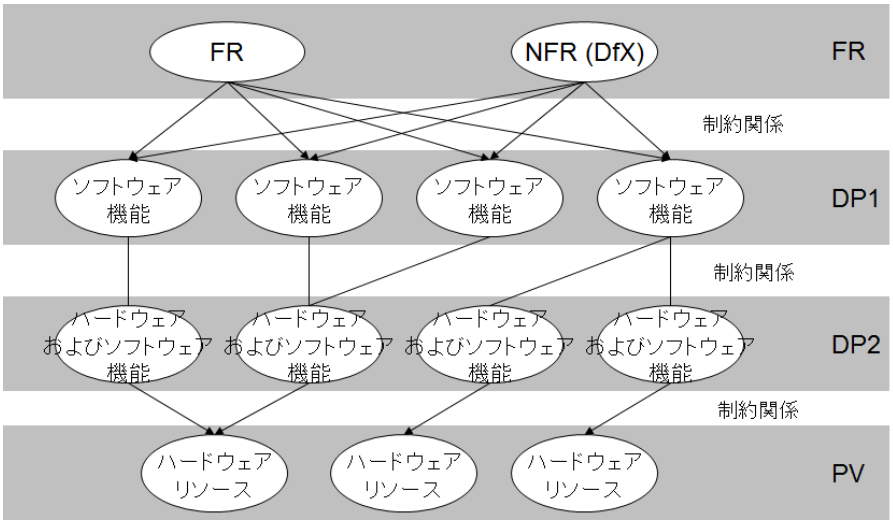

図 3-22 要件・機能・リソース割り当てのステップ

3.5.2.2 サービスモデルチェーンの構築

パラメータの転写関係を，モデル間のデータ連携で表現する．このデータ連携を，細分化したモデル間の関係に適用し，モデル間に間断の無い情報連携を実現する．このように

して，構築できたモデル群を**サービスモデルチェーン**と称する（図 3-23）.

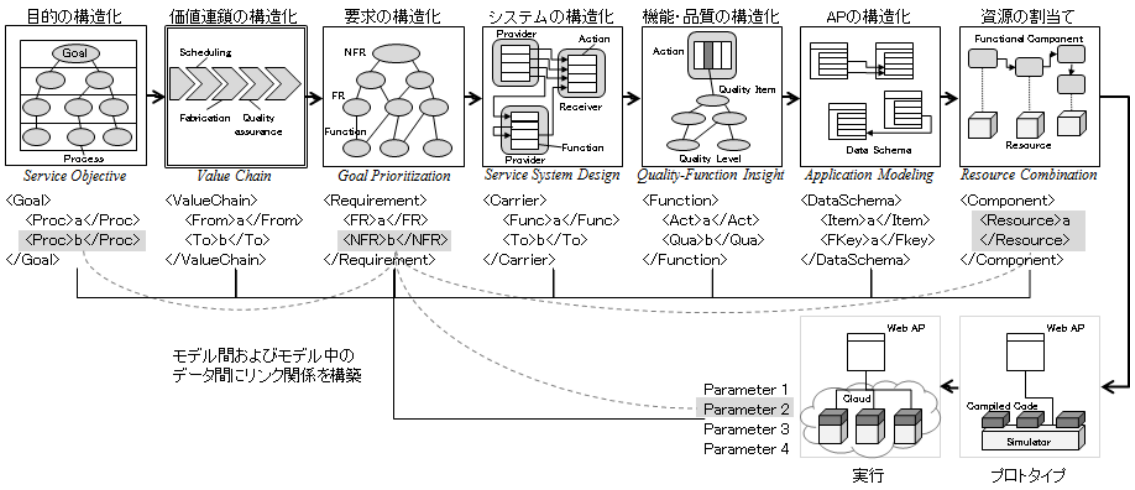

図 3-23 サービスモデルチェーン

3.6 モデリング方法の評価と考察

3.6.1 モデリング方法のクラウドサービスへの適用

4 つの提案方法は，リポジトリを中心に，一連の 1 つのフレームワークとして機能することが可能である．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[3]の考え方を実践・実装するための機構になり得る．

この機構をクラウドサービスへ適用した具体例を図 3-24 に示す．ここでは，ある 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する．このリポジトリを，クラウド環境の仮想サーバ内に配置する．リポジトリは，Web サービス案件毎に用意することで，案件間で各案件の秘匿情報を隔離し，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，クラウドサービスプロバイダは，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 Web サービス化して，高機能かつ高付加価値のクラウドサービスを提供できる．

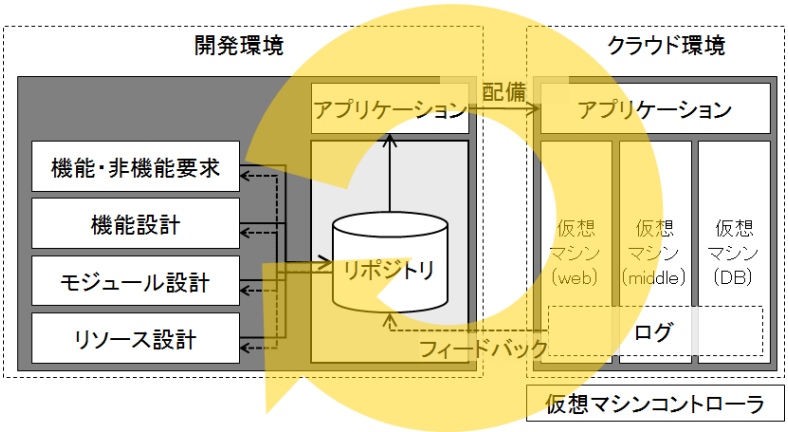

図 3-24 モデリング方法のクラウドサービスへの適用

3.6.2 ステークホルダの共通言語となるサービスモデルチェーン

ハードウェア製品の製造は，その開始時点で仕様が確定し，生産の各工程での作業や期間が明確化されている．一方，サービスの開発工程には多くの協働作業が含まれ，各工程での作業や期間は，人に依存し易い．分析～企画・設計に至る過程では，顧客と企画者の間で仕様の合意形成が図られ，実装～提供の過程では，設計担当と運用担当の間で設計情報の共有が必要となる．

本章で述べたサービスモデルチェーンは，標準的な記法で，かつ客観的な情報となるため，これらの異なる役割を持った担当者間で情報交換や合意形成する際の共通言語になる．

[3] DevOps とは，開発者(developer)と運用者(operator)が連携し開発・運用を行うプラクティスである．

3.6.3 サービスモデルチェーンの考察

サービスモデルチェーンを, Web サービスの開発に用いることで, 次の効果が得られた.

機能・非機能への展開設計では，機能カタログから機能候補を提示した．カタログデータから Web サービスの実装・運用過程で生じ得るリソース制約を，予め設計過程で考慮することが可能になり，下流工程での設計の見直し機会を抑えられた.

サービスシステム境界の設計では，アプリケーションを構成する機能モジュールと要件定義との対応を確認しながら設計した．これにより，先に要件定義と突き合わせを行うことで要件定義の対応漏れを検出できた.

機能モジュールおよび実行リソースへの割り当て設計では，機能独立性の評価を行いながら，アプリケーションの仮想マシンへの配置構成を設計した．ツールは，この設計情報を元にクラウド上に検証済みミドルウェアを含む仮想マシンを自動で構築した.

リソースの分離・結合による機能検証では，前工程で設計された機能モジュールを流用して，Web サービスのロジックの雛型を生成し，実装を効率化した.

また，リソースの分離・結合による機能・非機能検証では，Web サービスを実行するミドルウェアのパラメータは，要件定義過程で蓄積したレベル値を再利用して自動設定を可能とし，運用設計を容易にした.

このように，サービスモデルチェーンによって，設計作業を効率化し，設計内容を高品質化できた．以上により，ITIL で示された，戦略，設計，構築，移行，運用，改善を網羅したモデル群（図 3-25）となる.

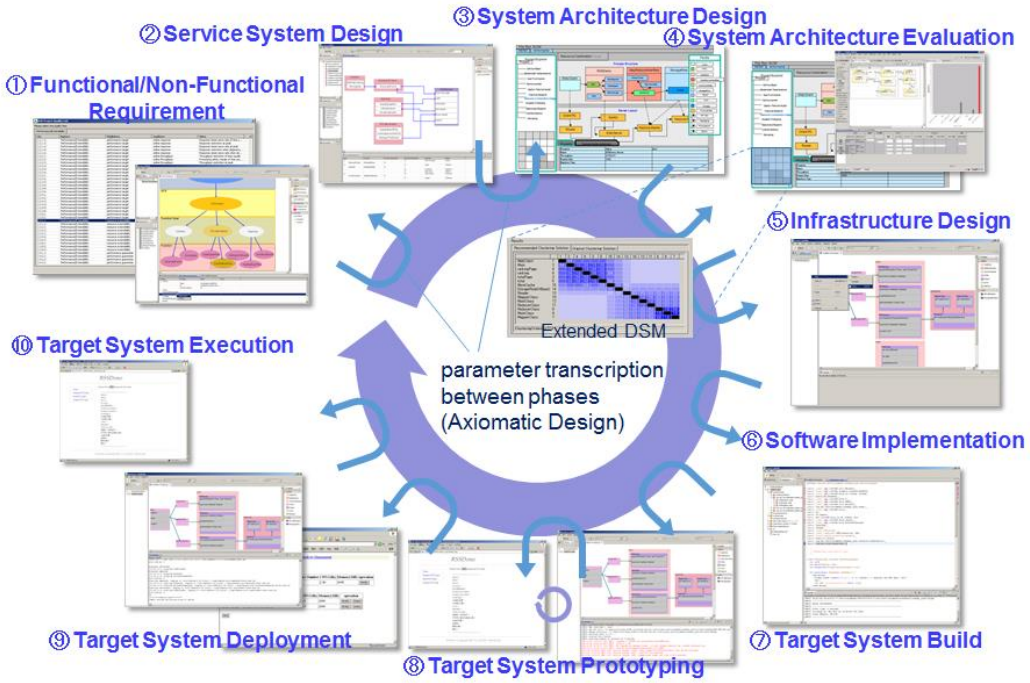

図 3-25 モデリングツール群［Hosono 2012b(改)］

これによりサービスシステムのサービスライフサイクルを通じ，パラメータを循環させることができる（図 3-26）.

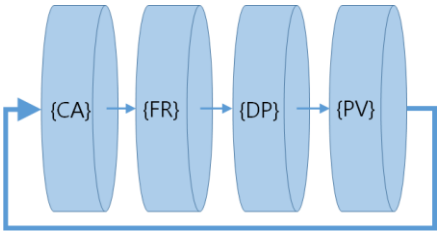

図 3-26 工程間のパラメータの関係

以上に示した通り，リソースと前後の工程のモデリングにおいて，機能とリソースの分離と統合が可能となり，機能と非機能の検証を容易にした.

また，各モデルで規定される属性間の関係を明確化し，工程間のデータ連携を間断なく実現．これにより，顧客と設計者間，および設計者と運用者間にそれぞれにおいて，正確な情報の伝達を可能にした.

3.7 おわりに

本章では，設計意図の伝播を支援するサービスモデリング方法を示した．本手法は，顧客と設計者間，および設計者と運用者間それぞれにおいて，設計意図を正確に伝播するためのモデリング方法である．

第 2 節では，サービスモデリングに関する先行研究を考察した．課題の考察では，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．

第 3 節では，モデリングの考え方を示した．設計の流れや設計観点に関するモデリング方法として，DfX，DSM の目的について述べ，その特徴を考察した．そして，これらを統合して，依存関係の整理と単純化する方針を示した．

第 4 節では，工程間の情報連携を実現するモデリング方法を示した．また，機能・非機能要件を包括してモデル化した．この方法によって，ライフサイクルの工程間に生じた情報のギャップを埋められることを示した．

第 5 節では，細粒度のモデリングと，公理的設計の考え方に基づき，要件→機能→リソース割り当ての転写関係により，サービスモデルの連鎖としてサービスモデルチェーンの構築方法を示した．

第 6 節では，サービスモデリング方法およびサービスモデルチェーンのクラウドサービスへの適用方法を示し，効用が得られることについて考察した．

以上に示した通り，以下の課題を解決した．

1. **実行状態に基づいて要件定義を行うこと**
2. **アプリケーションコンポーネントに実行リソース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リソース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**

本研究のアプローチは，クラウドサービスプロバイダが管理するリソースのモデル化を図ったことで，機能設計においてソフトウェアとリソースとの情報の統合を可能にした．リソースと前後の工程のモデリングにおいて，DSM と DfX の考え方を応用したことで，機能と非機能の検証が容易になる効果も得られた．また，機能要件と非機能要件の分離と統合を行えることを示した．

従来，自然言語での要件記述やノウハウとして暗黙的に扱われていた上流設計工程を含めて，モデリングにより形式化を図った．公理的設計の考え方に基づき，要件→機能→リソース割り当ての転写関係により，サービスモデルの連鎖としてサービス設計・運用プロセスを網羅した．従来のモデリングは，各モデルが独立していたのに対し，本研究が提案したモデリングは，各モデルで規定される属性間の関係を明確化し，工程間のデータ連携を間断なく実現した. また, DSM の拡張により, 設計データとして表現や管理し難かった，非機能要件も機能要件と透過的にモデルデータとして扱うことを可能にした．以上のモデリングは，異なる役割を持ったステークホルダ間の共通言語として作用するため，ステークホルダ間の相互認識を深められる．また，従来自然言語で書かれた上流工程の情報も，正規化されたデータとして記録されるため，開発の後工程で上流設計の振り返りも容易になる．これにより，システム／サービスの開発を，手戻りなく迅速に進めることが可能になった.

以上の通り，サービス企画〜運用までに工程に関わる，異なる役割を持ったステークホルダがそれぞれ扱う情報を形式化し，相互に参照し，ステークホルダ間の共通言語となるモデルを構築できた.

本章で示したモデリング方法は，ステークホルダ間の共通理解に用いるだけでなく，そのデータ構造を活用により，構造の操作によるモデルの挙動の確認や，データ構造の類似性に基づいたモデルの再利用も可能である.

IT サービスシステムを構成する Web サービスの開発において，そのプロトタイプ開発と動作検証は，作成と実行は試行を繰り返す作業となる．そこで，本章で示したモデリング方法は Web サービスの概念設計において，動作検証に応用できる．例えば，設計した機能の振舞いを机上で評価すべく，本モデリング方法に，概念設計段階で決定されるサービス構造の定性関係を加えることで，サービスシステムの挙動をシミュレーションし，設計検証を行える.

また, 開発した各モデルのデータ構造は, XML を初めとする標準化された構造表現とし, ファイルシステムに保管することで, 新たな Web サービス開発において, モデルを復元し, 追加，変更することで，モデルを再利用して開発を進められる.

第４章 ライフサイクル知識の資産化と活用方法

目次

4.1 はじめに

本研究では，サービスのライフサイクル知識の資産化に向けて，(1) サービスの設計モデルを間断なく連鎖させたサービスモデルチェーンと，(2) これをマス・カスタマイゼーションの元とするため，パタン化する方法について述べる．

第 2 節では，開発知識の資産化と再利用に関する要件について述べる．先行する取り組みとして，マス・カスタマイゼーションおよびプロダクトラインエンジニアリングについて述べ，サービスシステムの生産の観点での活用指針を示す．

第 3 節では，サービスのライフサイクル知識の資産化方法を示す．サービスドメイン開発とライフサイクルパタン開発により，サービス設計・運用プロセスのパタンを構築する．次に，サービスポートフォリオを具現化し，このパタンをサービスポートフォリオにて管理し，再利用する方法を示す．

第 4 節では，クラウドに対応した Web サービスにおいて，前述の資産化方法を具体的に示す．ドメイン開発とライフサイクルパタン開発により，Web サービスシステムの設計・運用プロセスを資産化する．

第 5 節では，サービスシステムの設計・運用プロセスがデータ構造化された，ライフサイクルパタンを利用して，次のサービス開発で利用する方法を述べる．

4.2 ライフサイクル知識の資産化と再利用に関する要件

4.2.1 資産化と再利用に関する要件

サービスの開発過程で得られる設計書，ソースコードなどの生成物だけでなく，顧客のビジネス上の価値基準や顧客視点で発生するリスク情報も含めて資産を構築し，これを複数のサービス開発プロジェクトに適用可能な形に体系化する方法が必要とされる．これにより，各プロジェクトを遂行する過程で，類似したものを再開発する無駄を無くせる．具体的には，企画〜開発データを定式表現し，多くのサービス開発案件へ適用し易くするため，最適なカスタマイズポイントを加えたモデルに再構造化することが必要である．

4.2.2 マス・カスタマイゼーション

マス・カスタマイゼーションとは，同じ仕様の製品を多量生産するマス・プロダクションと，注文生産するカスタマイゼーションを混合させ，個別仕様の製品を大量に生産する業態である．

Pine はマス・カスタマイゼーションの具体的な展開について，次の 5 つのタスクを示している［Pine1992，1993］．

(1) 標準化された製品やサービスにカスタム化したサービスを付加する．
(2) カスタマイズできる製品やサービスを作る．
(3) 提供時のカスタマイズを推進する．
(4) バリューチェーン全体に渡り迅速な対応を実現する．
(5) 最終製品やサービスをカスタマイズするために，コンポーネントをモジュール化する．

これらのタスクを設計・運用プロセスで実践することで，カスタマイズレベルが向上することを示している．第 3 章に述べたサービスモデルチェーンは，バリューチェーンそのものをデータ構造化したものである．

4.2.3 プロダクトラインエンジニアリング

4.2.3.1 ソフトウェアプロダクトラインエンジニアリング

ソフトウェアプロダクトラインエンジニアリング（Software Product Line Engineering）は，類似したアプリケーションソフトウェア製品を効率良く開発するための開発パラダイムである［Paul 2005］．ソフトウェアプロダクトラインは，特定の市場セグメントやミッションの用件を満たす，共通フィーチャー（代表的な特性）を備えたコア資産として構築できる．これは，(1) 再利用資産を開発する**ドメインエンジニアリング**と，(2) 固有のアプリケーションソフトウェアを開発する**アプリケーションエンジニアリング**の 2 つの開発プロセスから成る（図 4-1）．ドメインエンジニアリングとは，ドメイン知識を整理して，共通利用できる再利用資産を開発するアクティビティ（開発者の活動）である．一方，アプリケーションエンジニアリングとは，具体的な要求に基づき，これらの資産を利用してソフトウェア製品の開発を行うアクティビティである．

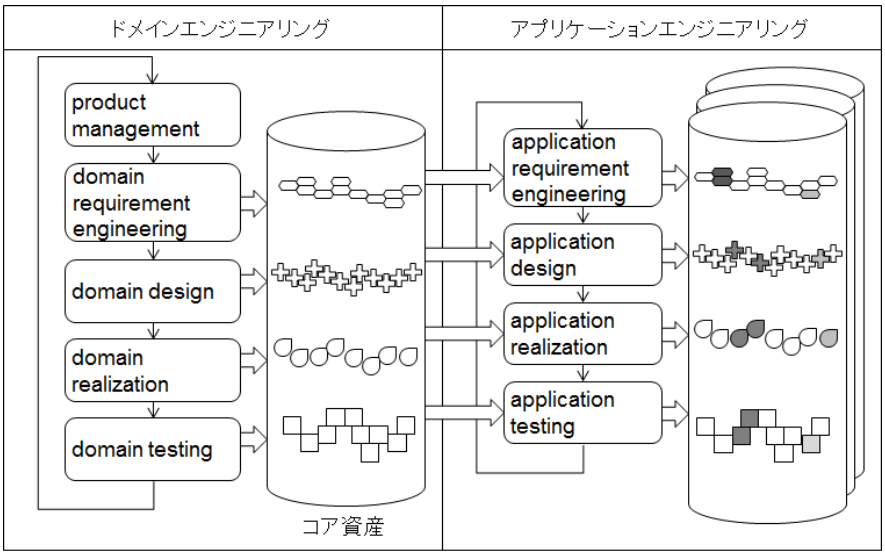

図 4-1 プロダクトライン構築［Pahl 2005(改)］

ドメインエンジニアリングおよびアプリケーションエンジニアリングは，次のタスクによって達成される．

(1) ドメインエンジニアリング

- ソフトウェアプロダクトラインの共通部分と可変部分を定義する
- ソフトウェアプロダクトライン構築対象のアプリケーションの組を定義する（ソフトウェアプロダクトラインの範囲を定義する）
- 目的の可変性を達成するドメインを定義し，再利用可能なアーティファクト（成果物）を構築する

(2) アプリケーションエンジニアリング

- プロダクトラインのアプリケーションを開発する際，再利用が可能なドメイン資産をできるだけ多くアーカイブする
- プロダクトラインアプリケーションの開発中に，ソフトウェアプロダクトラインの共通性と可変性を利用する
- アプリケーション要件，アーキテクチャ，コンポーネント，およびテストなどのアプリケーションのアーティファクト（成果物）を文書化する．そして，これらをドメインアーティファクトに関連付ける
- アーキテクチャから決定される要件に沿って，可変性を，コンポーネントやテストケースに紐付ける
- アーキテクチャ，コンポーネント，およびテスト上のアプリケーションとドメインの要件との違いの影響を推定する

ソフトウェアプロダクトラインエンジニアリングのプラクティスでは，アプリケーションエンジニアリングは対象アプリケーションの数に応じて 1 つ以上になるが，ドメインエンジニアリングは唯一のプロセスになる．

ドメインエンジニアリングを行う過程で，フィーチャーを構造的に表すためのモデリングを行う．**フィーチャーモデル**は，ソフトウェアプロダクトライン間の共通性と可変部分を識別する手法である．フィーチャーは検討中のシステムの機能と品質特性を記述できる．フィーチャーモデリング手法は，フィーチャーツリーを生成する機能を階層構造に分解可能にする．フィーチャーは，次のサブフィーチャーに分けられる．

- **必須フィーチャー**（mandatory）
- **選択フィーチャー**（optional）
- **代替フィーチャー**（alternative）

これらのフィーチャーと，可変点と，可変項目の関係を図 4-2 に示す．必須フィーチャーは，共通部分と見做せる．選択フィーチャーおよび代替フィーチャーは，可変部分と見做せる．

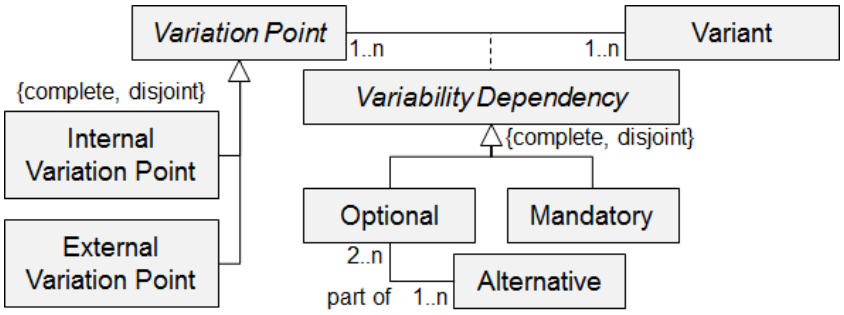

図 4-2 フィーチャーモデリング［Pahl 2005(改)］

これにより，フィーチャーツリーと可変ダイアグラムは，アプリケーションソフトウェアの構造を視覚化できる（図 4-3）．このツリーでは，構成要素がオプションや選択であるか可変性をモデル化している．個々の製品仕様に合わせて各可変性を決めると，対象製品の構成に必要なフィーチャーのみのツリーが得られる．このようなステップで，製品固有のツリーを導出し，可変性の特定と管理を行い，柔軟でかつ再利用性の高いソフトウェア製品を設計することができる．

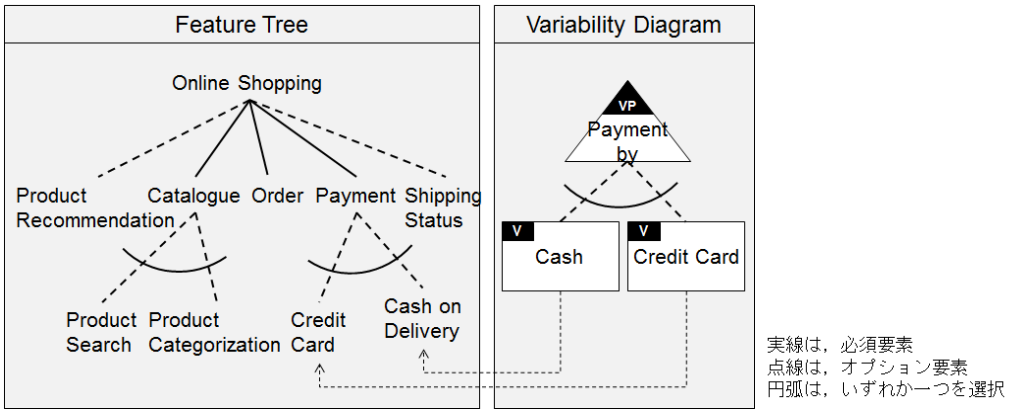

図 4-3 フィーチャーツリーと可変ダイアグラム [Pahl 2005(改)]

4.2.3.2 ソフトウェアファクトリ

ソフトウェアファクトリは，ソフトウェアのスキーマに基づいて，ソフトウェアのテンプレートを使用して，拡張ツールを設定する．そのため，製品ファミリーのための生産設備を提供するソフトウェアプロダクトラインエンジニアリングのアプローチを支援する基盤と見做せる．ソフトウェアファクトリは，実証済みのパタンと実践手順をカプセル化し，特定の目的に合った資産の統合セットを提供し，特殊な開発および実行環境を持つ．要件，論理および技術的なアーキテクチャ，テストスイート，展開トポロジ，運用基盤，メンテナンス計画，移行経路，ガイドライン，コードサンプル，テンプレート，ライブラリ，モデル，構成ファイルなど，ツール，プロセスおよびコンテンツが含まれる．これらの資産は，サービスプロバイダによって，プロセスの分割，契約の定義，機能設定，展開，検証，要件のトレース，および影響分析などに使用される．資産は，カスタマイズ可能であり，テンプレートは，選択，適応，構成，組み立て，およびパラメータ化ができる．これにより，様々な機能や運用の幅広いソフトウェアシステムを生産できるファクトリを実現している．

これらのソフトウェアプロダクトラインエンジニアリングとソフトウェアファクトリのアプローチは，これまでアプリケーションソフトウェアの開発に限られていた．サービスシステムの生産に対応するため，システム全体の設計とリソース管理を行えるように拡張する．

4.2.4 ライフサイクル知識の資産化と活用方法の課題

これらのソフトウェアプロダクトラインエンジニアリングとソフトウェアファクトリのアプローチは，これまでアプリケーションソフトウェアの開発に限られていた．ソフトウェアプロダクトラインエンジニアリングとソフトウェアファクトリのアプローチは，共通／カスタマイズ部分の対象は，業務ロジックを構成する部分，すなわち，機能設計を中心とした開発工程のみであった．サービスシステムの生産に対応するため，システム全体の設計とリソース管理を行えるように拡張が必要である．クラウド環境を前提とした開発を実現するには，システムインテグレータとクラウドサービスプロバイダに跨ったライフサイクル知識を蓄積すること，および業務ロジックを構成する部分だけでなく，リソース割り当てまで含め，開発～運用まで工程を拡げ，各工程にカスタマイズポイントを入れることが必要である．

以上に示した通り，開発知識の資産化と再利用に関する課題は，次の通りである．

1. **ライフサイクルを通じて，異なるステークホルダに跨った知識をプロバイダに蓄積させること**
2. **顧客の異なる要件に対応して再利用できるように，蓄積した知識のカスタマイズを容易にすること**

次節より，上記の課題解決方法を示す．

4.3 ライフサイクル知識の資産化方法

サービス設計･運用プロセスの資産化方法は, 第 3 章で述べたモデリング方法を用いて, 設計・運用プロセスの再利用に向けた資産化方法とする.

従来，個々の設計モデルあるいはソフトウェアコンポーネントの資産化に留まっていたのに対し，本手法は，ライフサイクル設計・運用プロセス自体を資産化の対象とする.

また，プロダクトラインエンジニアリングの考え方を応用し，カスタマイズインタフェースとモジュール最小構成を同定する.

4.3.1 開発～運用工程の知識の統合

4.3.1.1 カスタマイズポイントの導出

第 3 章で示した各モデルから構築したサービスモデルチェーンは，サービスの設計・運用プロセス全体を網羅している．このサービスモデルチェーンをライフサイクル知識として保持することで，別の顧客に対して同じ Web サービスを開発する際に再利用できる．サービスモデルチェーンには，開発に必要な設計情報が整列されるため，WBS（work breakdown structure）[4]として開発全体の作業や成果物の雛形になるからである．しかし，一般に，同じ Web サービスの開発であっても，全ての要件が同じことは稀で，顧客毎にユーザインタフェースや部分的な機能要件の差異が生じる．そこで，このサービスモデルチェーンを有効に多くの案件に適用できるように，カスタマイズを前提とした，マス・カスタマイセーションを指向する．この実現には，多品種少量生産に適したプロダクトラインエンジニアリング［Clements2001］の考え方を応用し，次のように，ドメイン開発とカスタマイズの元になるパタンの開発を行う.

ライフサイクル知識の再利用性を最大化するため, 次のようにコアと可変点を定義する. ライフサイクルを指向したドメインとし，モデル中の必須属性をコア，選択・代替属性を可変とする．モデル間のデータ連結に必須となるものは，**必須**（mandatory）とし，追加属性の場合は，**選択**（optional）とする．また，代替となる属性 a は**代替**（alternative）を表す.

資産の再利用において，再利用性と適応性の間にトレードオフがある．再利用可能な資産を最大化するために，パラメータの型は決定論的な方法で指定する．全てのモデルチェーンは主に必須（mandatory），追加データは選択（optional）属性であり，他者との依存関係は，代替属性（alternative）である．これに基づき，次のように，各モデルのこれらの属性を定義づける.

- **サービス目標モデリング**（Service Objectives）: financial, customer, innovation

[4] WBS とは，プロジェクト管理を行うための計画手法で，プロジェクト全体を詳細な作業に細分化し，階層構造などで管理する手法である.

perspectives は必須，これらの関係性は選択（optional）.

- **ステークホルダ抽出モデリング**（Value Chain）: stakeholders は, 必須. 一方, value chain の流れを構成する条件は代替（alternative）.
- **要求・機能展開モデリング**（Goal Prioritization）: operational processes, requirements functions は，必須．要求から機能への転写は，設計者の選択であるため，代替（alternative）.
- **サービスシステムモデリング**（Service System Design）: functional components は，必須．Functional component 間の functional flow は，代替（alternative）.
- **品質・機能モデリング**（Quality Function Insight）: quality と function は必須．non-functional requirements は，性能と信頼性のように互いに影響するので，代替（alternative）.
- **ユースケースモデリング**（Application Use Case/Data Model/System Flow）: ソフトウェア機能コンポーネントの実装の選択は多様であるため，ソフトウェアコンポーネントとデータストアは必須，代替（alternative），または選択（optional）.
- **リソース割り当てモデリング**（Resource Combination Design）: Runtime environments は必須．仮想マシンのパラメータは選択（optional）または代替（alternative）
- **プロトタイピング**（Application Prototyping）: ソフトウェアとリソースコンポーネントの間に制約関係がある場合，機能モジュール間の関係は，選択（optional）かつ代替（alternative），または，選択（optional）か代替（alternative）のいずれか.
- **非機能要件監視・評価**（Application Execution : Execution stacks は必須．そのパラメータの一部に，互いに依存関係がある場合，代替（alternative）.

これらの任意の代替の属性が可変点と見做し，新しいサービスの開発のための設計のカスタマイズ可能なインタフェースとして定義できる（表 4-1）.

表 4-1 カスタマイズポイント

	工程(Phase)	必須(Mandatory)	選択(Optional)	代替(Alternative)
1	目的の構造 Service Objective	finance, customer, innovation	relationships	
2	バリューチェーンのモデリング Value Chain	stakeholders		conditions for a flow of value chain
3	ゴールと機能の関係のモデリング Goal Prioritization	requirements, functions		transcription from requirements to functions
4	システム構造のモデリング Service System Design	functional components		functional flows
5	機能と品質の関係のモデリング Quality Function Insight	quality and function		non-functional requirements
6	ユースケースのモデリング Application Use Case	software components	software components	software components
7	機能のリソースへの割り当て Resource Combination Design	runtime environments	parameters	parameters
8	機能要件の検証 Application Prototyping		relationships between functional carriers	
9	非機能要件の検証 Application Execution	execution stacks		parameters

4.3.1.2 カスタマイズポイントを含むライフサイクルパタン構築

サービスドメインのインスタンスとして，ライフサイクルパタンを開発する．モデルチェーン全体が開発に必要になる場合と，部分的に必要になる場合がある．ドメイン特性を，関心事と定義し，関心事に沿って，モデルチェーンの中から必須部分をパタンとして導出する．

以上に示した手順を含め，設計・運用プロセスの構築と利用は次のステップで実践する（図 4-4）．

ステップ 1：サービスのドメイン開発（domain development）

対象サービスを分析し，対象サービスのライフサイクルを通じた関心事を明確化する．例えば，実装指向，プロセス指向などの関心事から，サービ

スドメインを特定する．このドメイン特性に基づき，各モデルに含まれる属性の特性を mandatory（必須），alternative（代替），optional（選択）のいずれかに決定する．

ステップ 2：ライフサイクル知識のパタン構築設計

モデリングツールを利用して，サービスモデルチェーンを構築する．サービスモデルチェーンを構成するモデルについて，alternative，optional 属性に対して，カスタマイズインタフェースを指定する．

ステップ 3：ライフサイクル知識のパタン構築

上記のサービスモデルチェーンから秘密情報や特定のサービス固有情報を除き，共通リポジトリに保持する．

ステップ 4：ライフサイクル知識のパタン利用

サービスを開発する際の初期データとして，保管されたサービスモデルチェーンを使用する．要件に合わせてカスタマイズインタフェースに対してカスタマイズを行い，サービス開発を進める．

ステップ 3 では，共通リポジトリにライフサイクルパタンを蓄積し，ステップ 4 では，共通リポジトリに保管されたライフサイクルパタンを，案件毎に用意した開発用リポジトリにインポートし，カスタマイズ開発を行う．以上のステップを図 4-4 に示す．

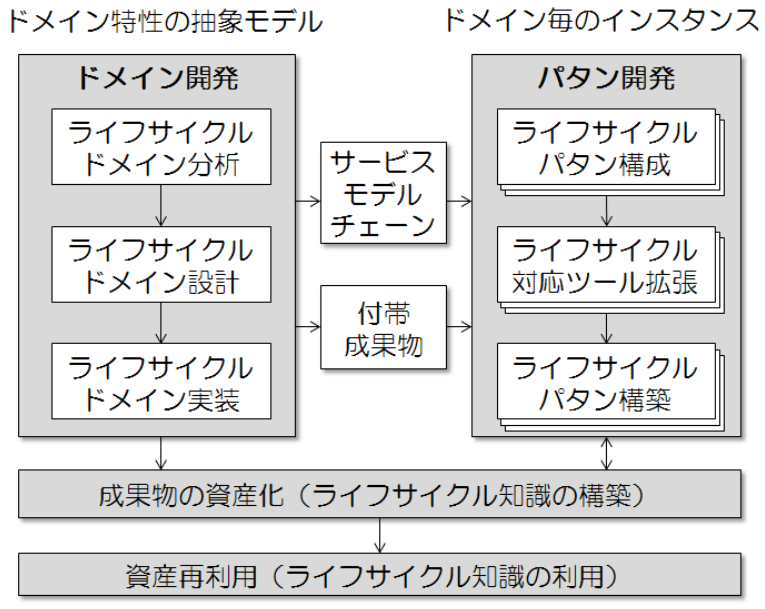

図 4-4 プロダクトライン型のサービス開発［Hosono 2012a(改)］

4.4 モデルチェーンからのライフサイクル知識構築

4.3.1 に示したステップを，データセンタにおける Web サービス開発に適用し，3 つのドメイン，および 4 つのパタンを開発し，カスタマイズポイントを決定する．

まず，モデル間の間断の無い情報連携から，サービスモデルチェーンを構築し，ドメイン開発を行う．次に，サービスモデルチェーンから，そのインスタンスとなるデータを開発し，パタン化し，これを設計・運用プロセスを表すデータとして資産化する．

4.4.1 サービスドメイン開発

サービスドメインの開発として，サービスモデルチェーンの開発を行う．これは，図 4-5 に示すように，あるドメインに必須となるモデルを束ね，サービスモデルチェーンを構築する．このサービスモデルチェーンは，アプリケーションエンジニアリングを行う際の雛形となる．

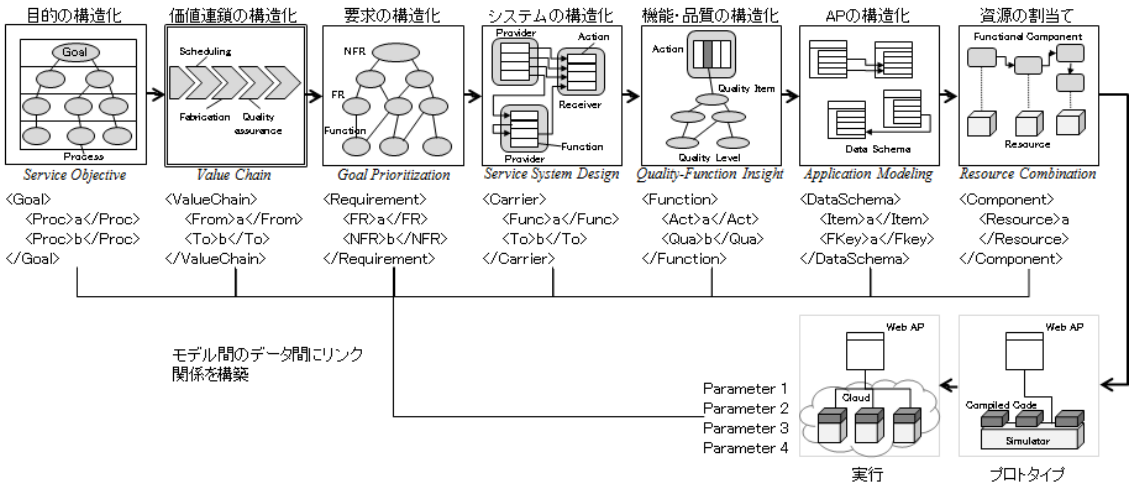

図 4-5 サービスモデルチェーンによるドメインの雛形［Hosono 2012a(改)］

クラウドコンピューティング，クラウドサービスの浸透に伴うユーザ要件の動向から，クラウドに対応した Web サービスに対して特徴的となった，次のドメインを定義できる．

(1) 管理者視点のライフサイクル管理に関するドメイン

ライフサイクル全体を管理するドメインである．クラウドサービスシステムの異なるステークホルダによって制御されているように，開発と運用の間のギャップを埋めるため，高いレベルのサービス管理が必要とされる．この考察に沿って，このドメインは，**サービス目標モデリング**（Service Objective），**ステークホルダ抽出モデリング**（Value Chain），**要求・機能展開モデリング**（Goal Prioritization），**サービスシステムモデリング**（Service System Design），**品質・機能モデリング**（Quality-Function Insight）で構成する．

(2) 開発者視点の開発プロセスに関するドメイン

開発プロセスに関するドメインである．クラウドに対応したサービスはクラウドサービスから提供される検証済みの機能を組み合わせることにより開発を進めるため，短期に開発・提供することが求められる．これは反復的な開発よりも，寧ろアジャイル開発に繋がる．この考察に沿って，このドメインは，**要求・機能展開モデリング**（Goal Prioritization），**ユースケースモデリング**（Application Use Case/Data Modelling/System Flow），**リソース割り当てモデリング**（Resource Combination Design），**プロトタイピング**（Application Prototyping），**非機能要件監視・評価**（Application Execution）で構成する．

(3) 実装者視点のシステムアーキテクチャに関するドメイン

サービスシステムのアーキテクチャに関するドメインである．クラウドで大規模なデータ処理アーキテクチャを設計し，スマートデバイスークラウド間連携のシステムアーキテクチャを設計する方法が必要である．この考察に沿って，このドメインは，**要求・機能展開モデリング**（Goal Prioritization），**サービスシステムモデリング**（Service System Design），**ユースケースモデリング**（Application Use Case/Data Modelling/System Flow），**リソース割り当てモデリング**（Resource Combination Design），**プロトタイピング**（Application Prototyping），**非機能要件監視・評価**（Application Execution）で構成する．

以上によって，開発したサービスドメインを図 4-6 に示す．

	Webサービスの特徴	開発したドメイン
1	サービスマネジメント	**管理者視点のライフサイクル管理**
2	シナリオベースのWebサービス	**開発者視点の開発プロセス**
3	クラウドとモバイル端末との連携	**実装者視点のシステムアーキテクチャ**
4	クラウド上でのビッグデータ分析	

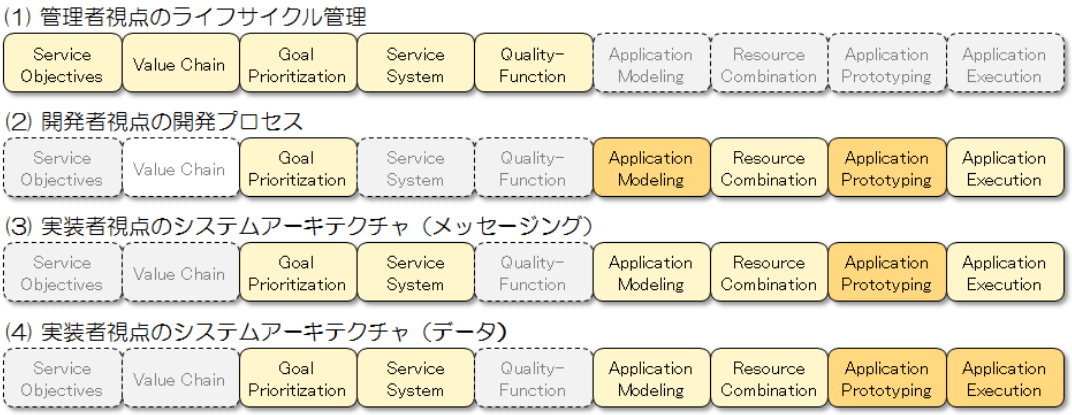

図 4-6 開発したサービスドメイン

4.4.2 ライフサイクルパタン開発とカスタマイズインタフェース

前節に示したサービスドメインに対して，そのインスタンスとなるライフサイクルパタンを開発する．ライフサイクルパタンは，サービスモデルチェーンの一部または全てである．図 4-7 に示すように，各ドメインに特化した部分を，ライフサイクル全体を示すサービスモデルチェーンから切り出し，データ化する．この切り出しによって，必要最小限でかつ，再利用性のライフサイクルパタンのデータを構築する．

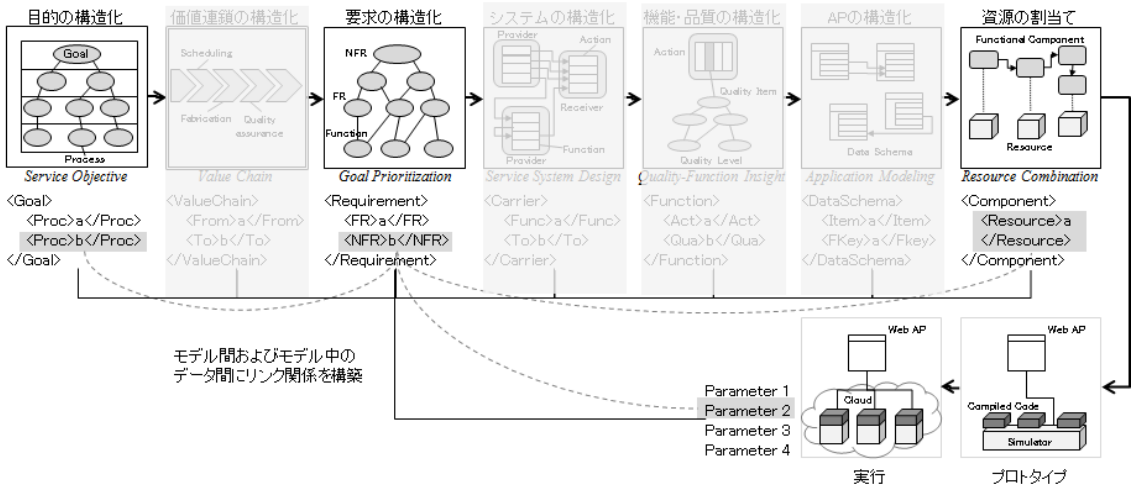

図 4-7 ドメインに特化したサービスモデルチェーン

サービスドメインから，(1) ライフサイクル全体，(2) 工程間の情報連携，(3) スマートデバイスとサーバ（クラウド）の連携，(4) 大量データ（ビッグデータ）の分析の 4 つのパタンを主要なライフサイクルパタンとして導出した．このパタンには，開発工程でなく，実装したソフトウェア機能を実行リソースであるクラウドに配備し運用する運用工程を含む．

(1) Web サービスライフサイクル管理パタン

このパタンは，基本的なサービス管理を提供し，管理機能をサポートする．

(2) 振舞い駆動 Web サービス開発・運用パタン

このパタンは，アジャイル開発手法に焦点を当てている．これは，動作記述とテストのための実装やアプリケーションフレームワークのための軽量言語が含まれる．

(3) スマートデバイス-クラウドアプリケーション開発・運用パタン

このパタンは，クラウド対応の Web サービスシステムのアーキテクチャに焦点を当てる．これは，モバイル環境とクラウド環境が連携して動作するアプリケーション

パタンである．パタンには，スマートデバイスとクラウド側のプロトタイプ環境の両方を含み，また，補助ソフトウェアライブラリなどの外部ツールを備える．

(4) 大規模並列アプリケーション開発・運用パタン

クラウド対応のアーキテクチャの他のパタンは，非機能要件（NFR），特にスケーラビリティに焦点を当てる．このパタンはアクターモデルや map-reduce モデル[5]など，大規模なデータ処理のためのミドルウェアによって強化される．これらは NFR を維持しつつ，スケール性および伸縮性のあるサービス実行を行うためのパタンである．

	ドメイン	開発したパタン
1	マネジメント指向	**Webサービスライフサイクル管理パタン**
2	トレーサビリティ指向	**振舞い駆動Webサービス開発・運用パタン**
3	アーキテクチャ指向	**クラウドとモバイル端末との連携パタン**
4	データ指向	**クラウド上でのビッグデータ分析パタン**

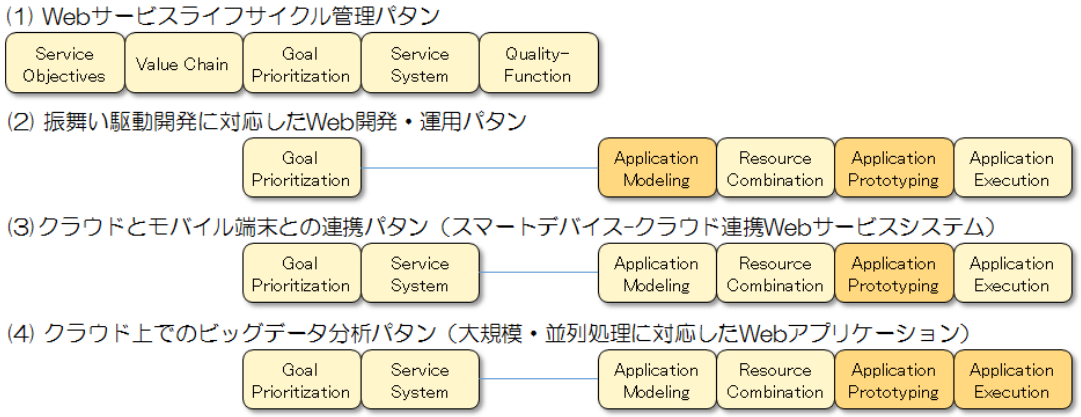

図 4-8 開発したライフサイクルパタン

以上に示したように，本研究のアプローチは，設計～製造～配備までを対象とし，ソフトウェア機能モジュールに加えて，配備先のリソースまで対象に含めた．ここで，配備先のリソースは，クラウドサービスプロバイダによって管理されるが，本研究のアプローチは，このリソース設計と配備まで含めて，資産化しているため，運用工程も含めてカスタマイズポイントの設定が可能となった．

[5] アクターモデルや map-reduce モデルは，Web サービスを構成する計算モデルの例である．

4.5 ライフサイクル知識の再利用

4.5.1 サービスポートフォリオ

4.5.1.1 サービスポートフォリオ

サービスポートフォリオは，サービスの企画～構築までの設計情報や，サービス内容や提供状態，投資状態など，ライフサイクルを通じた全てのデータを一覧できるように，包括した概念である．本節では，前節に示したライフサイクル知識を蓄積し，活用するための概念としてサービスポートフォリオを対応付けする（図 4-9）．ライフサイクル知識は，サービスシステムの目的，バリューチェーン，アーキテクチャ，機能設計，リソース設計，実装，機能・非機能検証，運用監視を網羅しているためである．

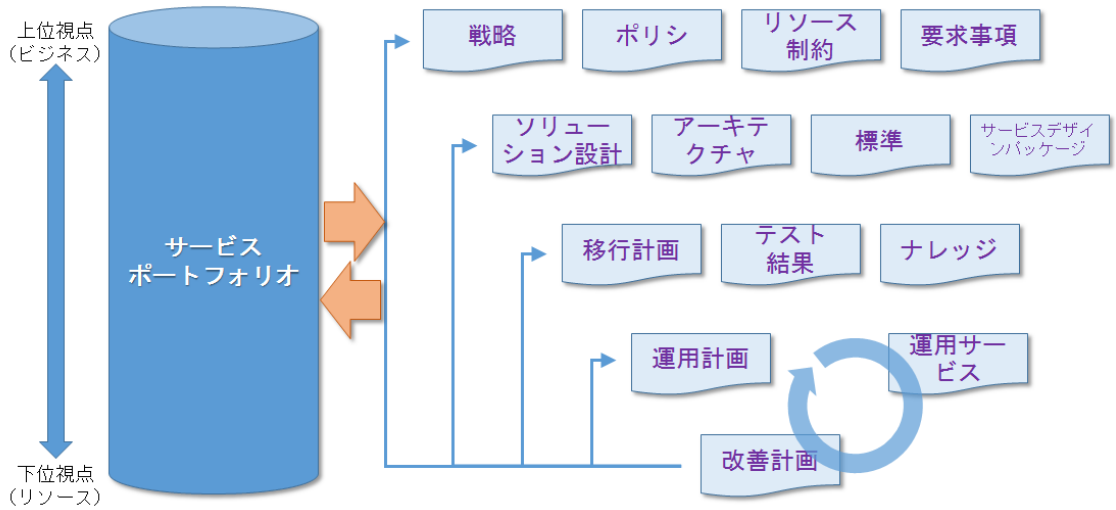

図 4-9 サービスポートフォリオ

サービス開発を行うプロバイダは，サービスポートフォリオに蓄積された情報から必要とされるものを選択し，プロバイダ内の開発者に対してサービス実行を支援する技術情報を提示する一方，顧客に対してサービスの内容に関するビジネス情報を提示する．（図 4-10）

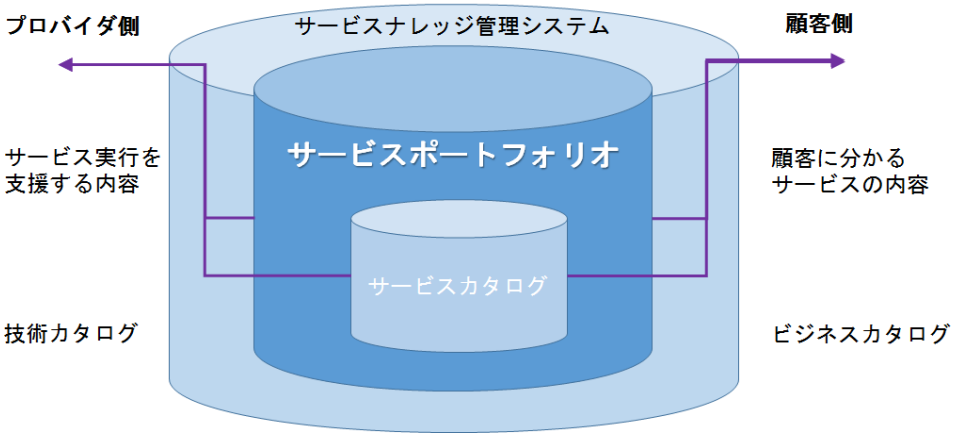

図 4-10 サービスポートフォリオの概念構成［ITIL2008(改)］

4.5.1.2 パブリック・サービスポートフォリオ

前節までに，サービスを組み立てるプロセスを形式化する方法を示した．各サービスモデルの構成属性をカスタマイズのインタフェースにすることで，適用可能な工程（範囲）を拡大し，Web サービスの設計から提供までのプロセスが異質化した顧客要求に柔軟に対応できるようにした．これにより，顧客の要求の変化に迅速に対応しつつ，規模の経済性とモジュール化による範囲の経済性を同時に獲得できる．従来，モノの生産に活用されてきた，マス・カスタマイゼーションの概念をサービスの生産においても実践できる．これを実践するために，この設計・運用プロセスを示すライフサイクルパタンを各サービス開発プロジェクト開発の雛形データとして，公開・共有されるパブリック・サービスポートフォリオに保管する．

4.5.1.3 プライベート・サービスポートフォリオ

開発プロジェクト間で共有するパブリック・サービスポートフォリオに，サービスモデルチェーン，及び実行環境を資産化する．

各サービス開発プロジェクトで作成・活用するライフサイクル情報を蓄積するリポジトリとして，プライベート・サービスポートフォリオを導入する．パブリック・サービスポートフォリオに資産化されたこれをプライベート・サービスポートフォリオにインポート可能にすることで，新規案件での開発の際に直ちに動作確認できる状態から開発を開始できる．このサービスモデルチェーンの各モデルにカスタマイズインタフェースを組み込むことで，提供段階のカスタマイズだけではなく，上流設計～実装段階の各工程においてカスタマイズが可能になる．

以上に示したパブリック・サービスポートフォリオと，プライベート・サービスポートフォリオの関係を示す（図 4-11）．

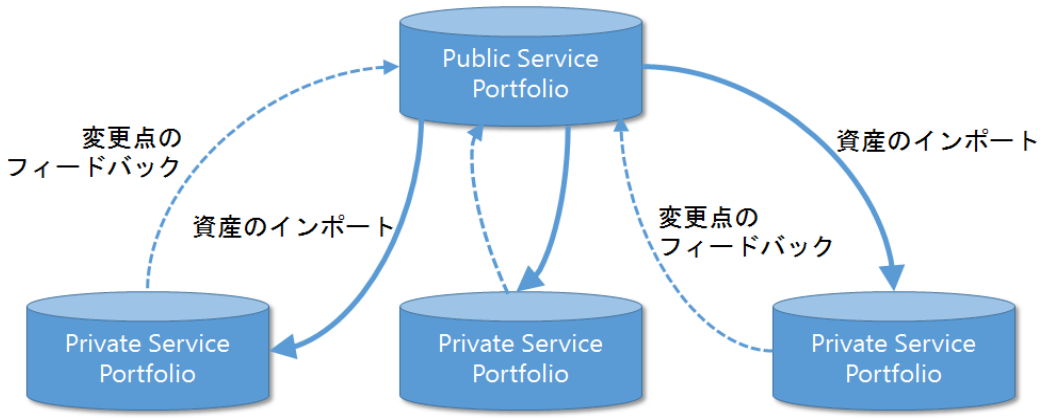

図 4-11 パブリック／プライベート・サービスポートフォリオの関係

4.5.2 サービスポートフォリオを用いた知識の資産化と再利用

サービスポートフォリオは，効率的にこれらのサービスのライフサイクルパタンを管理するための鍵となる．サービスポートフォリオは，管理要件，設計，実装，運用データのリポジトリで構成され，個別に共通のサービス固有のデータを管理する必要がある．

サービスポートフォリオを実装するために，2 つのポートフォリオを導入する．プライベート・サービスポートフォリオは，あらゆる開発プロジェクトによって共有されている各アプリケーションプロバイダとパブリック・サービスポートフォリオが所有するクラウドアプリケーションに使用する．サービスドメインとサービスのライフサイクルパタンを開発している間，パタンは，案件固有の業務情報など秘匿性の高いデータを除外し，コア資産は顧客の特定のデータを有することを見越して最小限に抑えた後，パブリック・サービスポートフォリオにエクスポートする．そして，それぞれのサービスモデルはライフサイクルパタンとして使用できる．

この機構に従って，サービス開発の新規プロジェクトは，それをインポートすることで開始できる（図 4-12）．サービス開発者は，このインポートしたデータを初期データとして用いることで，開発の初期段階から Web サービスを実行して動作を確認できる．このデータを元に，カスタマイズを行い，目的の Web サービスを開発する．このような手順で開発し，システム構築者は，簡単かつ迅速に顧客の要件を達成できる．

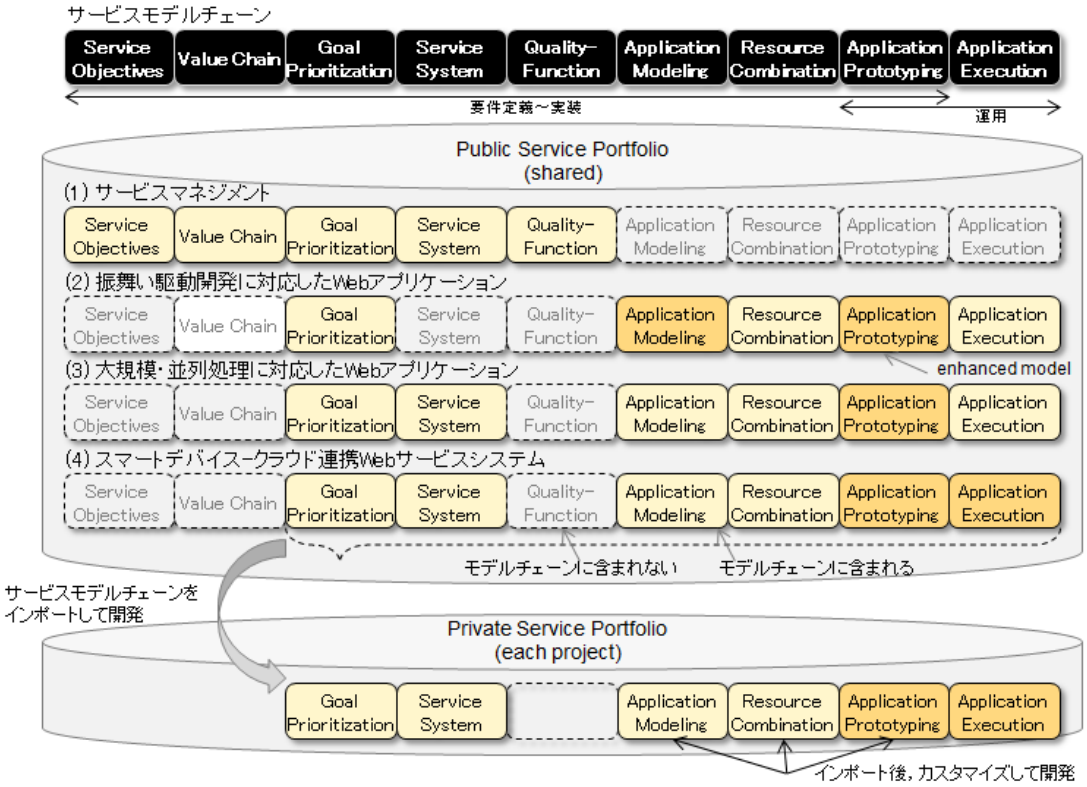

図 4-12 ライフサイクル知識の再利用［細野 2013b(改)］

4.6 おわりに

本章では，サービスライフサイクルの知識を資産化する方法を示した．この上流設計から運用までを包括したライフサイクル知識の資産化と活用方法は，蓄積した知識活用を前提とした Web サービスの開発方法に，一定の標準形を与えた．また，サービスモデルチェーンの構築とライフサイクル知識の処理機構により，即時に設計・運用情報やサービス開発の進捗状態を可視化し，ステークホルダ間の相互理解に有効なことを示した．これにより，IT サービスプロバイダは，クラウドに対応し，体系的・組織的な Web サービス構築サービスの提供が可能となった．(1)提供者側に蓄積できる仕組みを構築したこと，(2)設計・運用プロセスのパタンを構築したことにより，複数の顧客企業の案件に適用し易くした．

第 2 節では，上流設計から運用までを包括したモデルチェーンの構築を示した．第 2 節では，開発知識の資産化と再利用に関する要件について述べた．先行する取り組みとして，マス・カスタマイゼーションおよびプロダクトラインエンジニアリングについて述べ，サービスシステムの生産の観点での活用指針を示した．

第 3 節では，サービスのライフサイクル知識の資産化方法を示した．サービスドメイン開発とライフサイクルパタン開発により，サービス設計・運用プロセスのパタンを構築する．次に，サービスポートフォリオを具現化し，このパタンをサービスポートフォリオにて管理し，再利用する方法を示した．パブリック・サービスポートフォリオと，プライベート・サービスポートフォリオを分けたことで，再利用が容易になった．

第 4 節では，クラウドに対応した Web サービスにおいて，前述の資産化方法を具体的に示した．ドメイン開発とパタン開発により，Web サービスの設計・運用プロセスのデータ構造を資産化した．

第 5 節では，資産化されたサービスシステムのライフサイクルパタンを利用して，次の開発に活かす方法を述べた．

本研究では，サービスモデルチェーンを有効に多くの案件に適用できるように，カスタマイズを前提とした，マス・カスタマイゼーションを指向した．この実現には，多品種少量生産に適したプロダクトラインエンジニアリングの考え方を応用し，ドメイン開発とカスタマイズの元になるパタン開発を行った．プロダクトラインエンジニアリングの考え方を応用し，カスタマイズインタフェースとモジュール最小構成を同定した．一連の設計モデルデータに対して，プロダクトラインエンジニアリングの考え方を応用し，ドメイン開発とパタン開発を行い，カスタマイズインタフェースを備えた設計・運用プロセスをモジ

ュール化した.

これにより，従来，モノの生産に活用されてきた，マス・カスタマイゼーションの概念をサービスの生産においても実践できることを示した．この開発方法は，サービス生産の短期化と同時に，生産の原価低減を達成するもので，範囲の経済性および規模の経済性効果の高さを得られる．サービス価格の上昇を抑え，顧客が持続してサービスを利用し易くなる．よって，サービスシステムの維持・安定に寄与し，ストック型の事業を実現できる．

以上により，本章では，以下の課題を解決した.

1. **ライフサイクルを通じて，異なるステークホルダに跨った知識をプロバイダに蓄積させること**
2. **顧客の異なる要件に対応して再利用できるように，蓄積した知識のカスタマイズを容易にすること**

一方，次の課題が残されていることも明らかとなった.

サービスモデルチェーンは，Web サービス一般の開発プロセスに適用できるため，汎用性が高い．クラウドサービスプロバイダが提供する PaaS 環境で実行する Web サービスを想定しているため，社内システムと連携した Web サービスや，他のクラウド環境に跨って実行される Web サービスなど，多様な Web サービスの構成には対応できない点に本手法の限界があることも分かった．これらは，従来，サービス指向アーキテクチャ（SOA）分野で開発されてきた，ビジネスプロセス管理（Business Process Management: BPM）技術を更に組み合わせていく必要がある.

資産化した設計・運用プロセスには，ドメイン固有のビジネスルールや，法律などの情報が暗黙的に埋め込まれている．同じドメイン内で再利用する際には問題にならないが，ドメイン間での再利用性を高めるには，これらを考慮する必要がある．そのため，ビジネスルールや制約条件を，設計・運用プロセスのデータおよび背景となる情報から抽出し，構造化することが必要になる．この実現には，自然言語処理やパタンマッチングを用いて，ビジネスルールや法律に関する情報を設計・運用プロセスの情報から抽出し，分離，構造化する方法が考えられる.

また，製品指向（product-oriented），利用指向（use-oriented），成果指向（result-oriented）のサービス特性に基づく，ライフサイクルパタンの類型化も必要である.

第５章 ライフサイクル知識の構築・活用支援プロセス

目次

5.1 はじめに

本章では，サービスのライフサイクル知識の活用プロセスについて議論する．

第 2 節では，パタン化の取り組みと，その再利用の課題を分析する．そして，ステークホルダが協働して開発する際の課題を明らかにする．

第 3 節では，ライフサイクル知識の構築・活用プロセスを説明する．顧客と IT サービスプロバイダ間の協働支援，開発部門間の協働支援，および開発部門と企画・管理部門との協働支援について述べる．更に，これらの技法・協働活動を定着させるための規律について述べる．

5.2 ライフサイクル知識の構築・活用に関する要件

5.2.1 先行研究

5.2.1.1 ソフトウェア開発方法論とアジャイル開発手法

ソフトウェア開発方法論は，予め顧客からの要件が決められている場合に対して，計画的に設計から製造，検査までの手順を順に定義したウォーターフォール型に基づいてきた．

一方，顧客からの要件が開発中に変更されたり，追加されることが予測されたりする場合に対して，イテレーティブに手順を繰り返す方法も取られてきた．顧客の要件が更に不確定なままに，プロトタイプ開発を試行する場合には，最小の機能を実装して顧客と確認を行い，新たに現れた要件を追加実装して，要求の十分さの確認作業を繰り返すものである．この手法は，アジャイル開発と総称される．

クラウドサービスが広く用いられると，短納期の開発が前提となり，Web サービスの実装は，簡単かつ容易になる．これに伴い，継続的なプロトタイピングを通じて，顧客の要件を達成することができるアジャイル開発方法が，適用される．全ての顧客の要件が決定されていなくとも，クラウドから提供される検証済みのソフトウェア機能を組み合わせて，Web サービスの開発を進められる．

このアジャイル開発方法は，顧客や開発者間の協働を促す一方，開発の進度が適正か否か判別しにくい点も生じている．そこで，近年，協働作業を促進する一方，開発の各工程にに一定のガバナンスを行い，開発作業に規律を与える，ディシプリンド・アジャイル・デリバリー方法論が提案されている［Ambler 2012］.

しかし，この手法は，ソフトウェアの機能の実装と提供を主体に捉えており，要求～機能・非機能～リソース割り当てを系統的に行う設計手順の観点や，再利用を前提とした資産の構築と利用方法に関する観点が十分になされていない.

5.2.1.2 ステークホルダに着目した研究

ライフサイクル設計，セットベース設計，設計意図の研究アプローチがある．梅田らは，ライフサイクル計画を製品ライフサイクルプロセスに導入することで，製品製造の持続性向上を図っている［Umeda2012］．この方法を実践するには，ライフサイクル戦略と外部環境要因間の関係性と，再利用を前提とした開発の現場作りに向けた導入方法を，より具体化していく必要がある．一方，Finch, Ward によるセットベース設計［Finch1995, 1997］を石川，井上は応用し，PSD 手法（Preference Set-based Design Method）に纏めている［石川 2010］［Inoue2009］．この方法は，設計者間の不確定性を考慮したもので，設計者の設計意図を反映した選好度を導入し，多目的を折衷した設計仕様を範囲値として導出することで，開発中に変更の少ない製品設計を可能にしている．本手法は，設計工程における目的の多様性に注目しているため，運用要件など，実行工程での条件や制約を設計工程

で考慮しておくことには困難さが残る．

一方，荒井らは，意図を伝達する取り組みとして，設計意図を部品間の影響関係で記述し CAD への統合を図っている［荒井/Arai1998］．設計工程以降の意図や制約と擦り合わせるためには，同様に，統合型 CAD アーキテクチャを改善する余地がある．

5.2.2 協働作業の課題

開発過程の成果物や知見を再利用するために，第 4 章で，サービス設計・運用プロセスを網羅したライフサイクル知識の資産化方法を示した．従来，各開発工程で独立して設計情報やソフトウェアコンポーネントを対象に資産化の取り組みが行われてきたが，工程間の資産の連携がとれないため，各工程で資産の流用による限定的な効果のみが得られていた．本手法では，資産化の対象を設計・運用プロセス全体に拡げたことで，サービスの生産管理までを可能にしている．コアとなる最小構成の開発～運用プロセスを資産の単位として同定している．このモジュールには，各工程の設計モデル情報に加えて，各設計モデル情報の作成や利用に関わる人手の協働作業が定量データとして含まれる．更に，このモジュールをパタン化したことで，人手の関与が多く含まれるサービス設計・運用プロセスにおいても，モノの生産プロセス管理と同様に，進捗度合いなど定量的な管理・分析を可能にしている（図 5-1）．このモジュールをソフトウェアプロダクトラインと同様に，類似したサービス開発に向けた標準プロダクトライン（コア）とし，個々の顧客の要件の違いに合わせて，カスタマイズ開発や生産管理のカスタマイズを実現している．

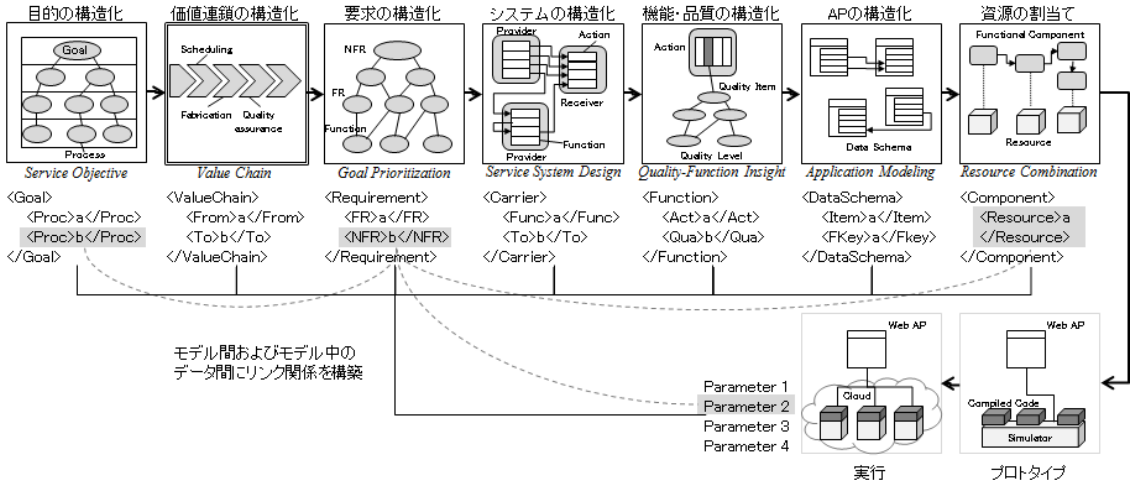

図 5-1 モジュール化［Hosono2012a(改)］

しかし，この資産が十分に効果を発揮できるのは，開発するサービスの仕様の違いが，サービス開発に向けた標準プロダクトライン（コア）のカスタマイズ・インタフェース（可変ポイント）に含まれる場合に限られており，柔軟性や適応性が低い．資産化の対象工程

が広いため，資産の構築時点と利用時点では，月単位あるいは年単位の経過に伴う環境変化の影響を受けるため，全ての資産を構築時の想定通りに再利用することは困難である．例えば，資産を構成するソフトウェアのバージョンアップなど，資産自体が変化するため，資産の適合性が変わり得る．また，ビジネスルールの更新など資産の利用手順が変化するため，その利用範囲が変わり得る．そのため，資産の適合性を確保するために，全て，部分，あるいは構成要素を再利用するのかなど，再利用対象を細分して扱う方法が必要とされる．

また，クラウドの浸透に伴い，システムが構築し易くなることから，要件や仕様の詳細が確定されないまま，アジャイル開発やラピッドプロトタイピングを通じて，短期間に繰返し開発を行いながら徐々にターゲットに近づけていく開発が行われるようになる［Scott2012］．これは，要件や仕様に不確定性（値の範囲）が生じ，過去の開発情報の適用選択肢が多くなり，結果として，最も適切なものを選びにくく，開発成果物を再利用し難くなる．このような条件の下では，適合性が不十分なまま資産を再利用して開発を進めるため，修正や変更する箇所が増え，開発工数を期待するほど低減できない．また，クラウド環境を利用したシステム構築では，クラウド上の検証済みの機能を利用することから，短期に開発・本番実行できることがシステム開発の要件ではなく，前提条件となる．そのため，開発時点において，ソフトウェアやサービスの機能要件に加えて，運用管理の保守性や改良のし易さなど，実装・実行工程での要件や制約を合わせて考慮する方法が必要とされる．

上記に示した方法は，IT サービスシステムにおけるコンピュータシステムの範囲で解決できる方法である．効用の高いサービスを効率的に生産するには，上記 2 つの方法に加えて，(a) 顧客と IT サービスプロバイダや，開発と運用部門，開発部門と企画・管理部門間などライフサイクル内のステークホルダ間の協働支援，(b) 過去のサービス開発と別のサービス開発の開発部門などライフサイクルを跨ったステークホルダ間の協働支援と，これらを定着させるためのプロセスが必要である（図 5-2）．

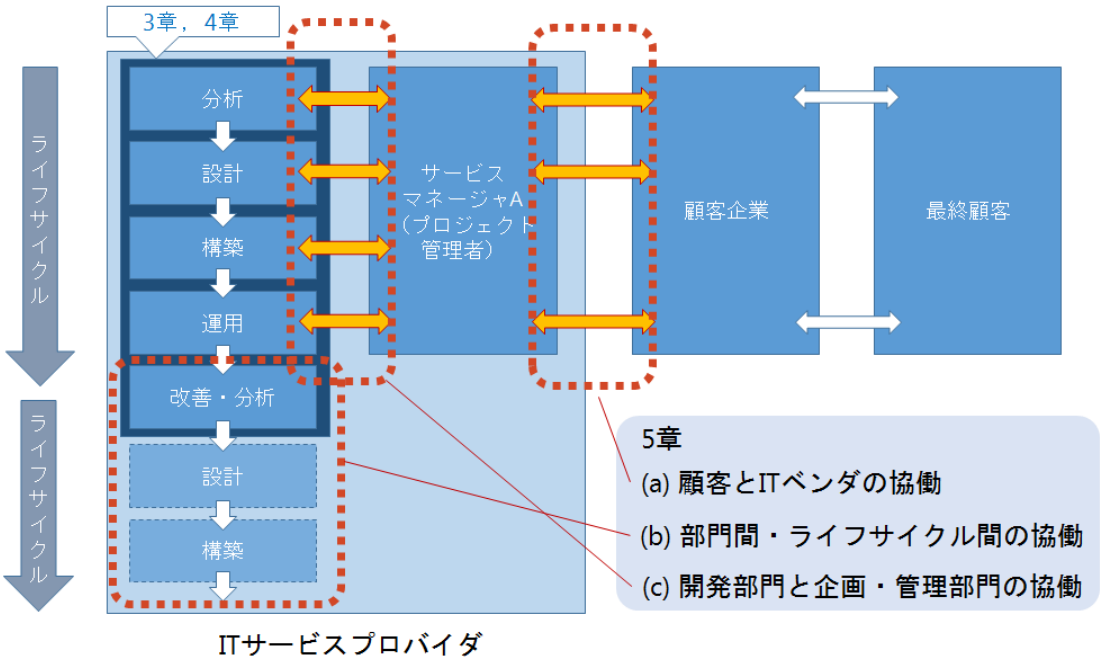

図 5-2 協働作業の課題

更に，新たな開発方法論や開発パタン・成果物の導入は，現場の設計者や実装者間，提供者と顧客との間のプラクティスとなじまず，導入と実践が進まない状況を変える必要がある．また，企画・設計・構築・運用・次の開発までのライフサイクルには，設計者や実装者間，提供者と顧客間などの多様なステークホルダ間で，再利用に関する意図の伝達やプラクティスを考慮する必要がある．

IT サービスプロバイダと顧客企業担当者間でサービス設計を進める際，合意形成に有効となる情報を提示し，顧客と IT サービスプロバイダ間の協働支援する仕組みが必要である．

また，開発成果物を資産化する際，その資産の再利用シナリオや資産化の意図を保持できるようにして，開発部門間の協働を支援する仕組みが必要である．また，モノの生産プロセス管理と同様に，管理部門が開発部門のサービス設計・運用プロセスの比較分析・管理を可能とし，開発部門と企画・管理部門との協働を支援する仕組みが必要である．

以上に示した支援方法を実際の開発に定着させ，開発プロセスに規律を与える方法が必要である．ライフサイクル知識の構築・活用の課題は，次の通りである．

1. **顧客とプロバイダ間，開発部門間，および開発部門と企画・管理部門間の協働支援**
2. **協働作業を定着させる方法**

次節より，上記のそれぞれに対する解決方法を示す．

5.3 ライフサイクル知識の構築・活用プロセス

5.3.1 ステークホルダ間の協働支援

5.3.1.1 開発者と運用者間の協働支援

第 2 章で議論したように，クラウドの浸透に伴い，開発はシステムインテグレータが，運用はクラウドサービスプロバイダが担うようになる．第 3 章のサービスモデリングと，第 4 章で示したそのパタン化の過程で，開発者と運用者間の協働に必要な情報が自ずと生成される．すなわち，サービスモデルチェーンの構築とパタン化の操作は，この開発と運用工程の間で必要十分な情報を相互に参照可能にしているため，開発者と運用者間の協働を支援している．

5.3.1.2 顧客と IT サービスプロバイダ間の協働支援

設計者や運用者等の異なる役割間でサービス設計情報をやり取りする際，その共通言語として，第 3 章で示したモデル群の表記が有効である．しかし，IT サービスプロバイダと顧客企業担当者間でサービス設計情報をやり取りし，合意形成する場合には，顧客担当者にとって情報が多く，意思決定には過多となり得る．そこで，顧客の合意形成に必要な情報量に抑え，特定の観点に絞った情報量のダイアグラムで表現し直すことが有効である．例えば，システム構成設計では，SysML の要求・構造・振舞い・パラメトリックダイアグラム等の標準モデル表記法がある．

そこで，蓄積されたモデルチェーンを利用して，これらの表現形式に沿った再構造化を行う．このダイアグラムは，異なる過程で決定されたデータを統合し，機能モジュール間の関係を示す．要求ダイアグラムは要求-機能展開モデリングのデータ構造の一部を，構造ダイアグラムはサービスシステムモデリングのデータ構造の一部を抽出する．また，振舞いダイアグラムは，ユースケースモデリングのデータ構造の一部を，パラメトリックダイアグラムは，サービスシステムモデリングのデータ構造の一部とリソース割り当てモデリングのデータ構造を統合する．例えば，顧客と IT サービスプロバイダ間で，要求の確認を行う際，IT サービスプロバイダは，要求・機能展開モデリングで作成したモデル情報から，リソースの範囲値などの詳細な属性値は省き，要求のサブ構造のみを部分的に抽出し，要求ダイアグラムを表示する．顧客と相互理解が進み，要件を確定させる際に，前述の属性値までダイアグラムに表示し，顧客との議論の対象に合わせた表示を行う．

この機構を図 5-3 に示す．この再構造化したデータは，例えば，システムモデルのダイアグラムを表現した SVG（Scalable Vector Graphics）形式の XML で出力し，Web ブラウザ上に，大きさの変更や移動が可能で，特定の観点に絞った情報量のダイアグラムとして描画する（図 5-4）．更に，このダイアグラム表示を通じて得られた気付きを，コメントとして入力可能にし，これをライフサイクルパタンのデータ構造に加える．これにより，協

働作業で得た知識をサービスモデルチェーンに取り込むことが可能となる．

このダイアグラム生成と表示のように工程を跨った情報，業務レベルからシステム・実装レベルを含む情報の再構造化によって，CAD ツールとして新たな設計視点をステークホルダに与えることができる．これにより，設計者と実装者や，顧客企業の担当者など異なるステークホルダ間で，設計過程で得た客観的な情報を用いて認識を共有し，相互理解を深められる．

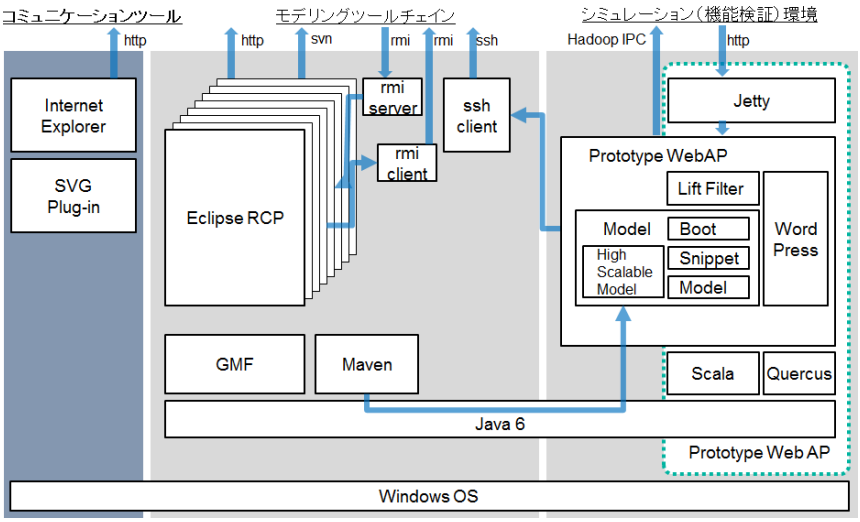

図 5-3 サービスポートフォリオの拡張［Hosono2012b(改)］

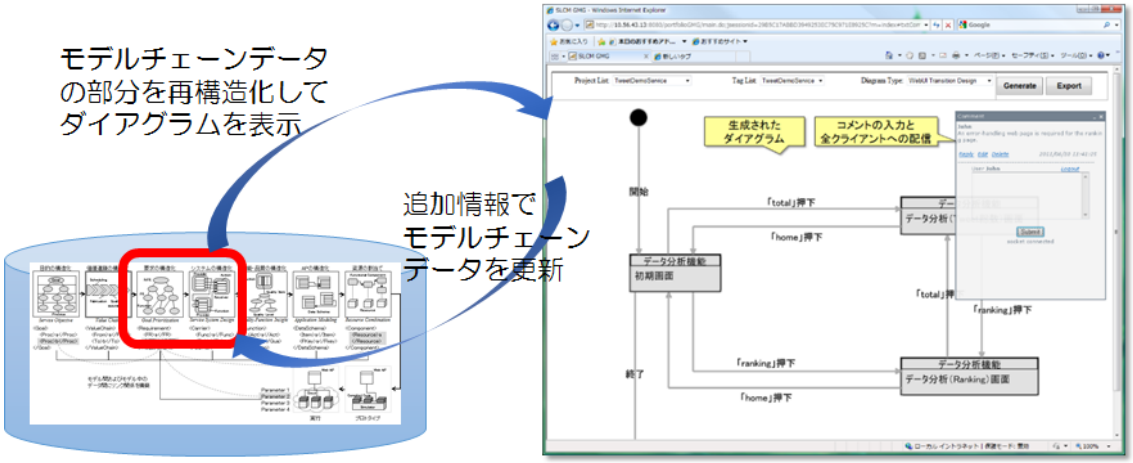

図 5-4 生成したダイアグラムの例

5.3.1.3 顧客と IT サービスプロバイダ間の協働支援

開発部門と企画・管理部門との協働支援として，**タスクチェーン**を構築し，前節で述べたサービスモデルチェーンとの統合を図る．

タスクチェーンは，開発の各工程で行う作業をタスクとし，タスク間を連結したものである．各タスクには，開発成果物や，作業実績などの項目と実績を，属性と属性値として

データを保持できる．このタスクチェーンは，設計・運用プロセスに沿って，サービスモデルチェーンと対応付けられる．このタスクチェーンとサービスモデルチェーンを，リンク付けする．このタスクチェーンの属性に人件費どの工数実績値を与え，人間系の作業実績を含めサービスモデルとタスクモデルを統合したモデルチェーンに構造化する（図 5-5）．

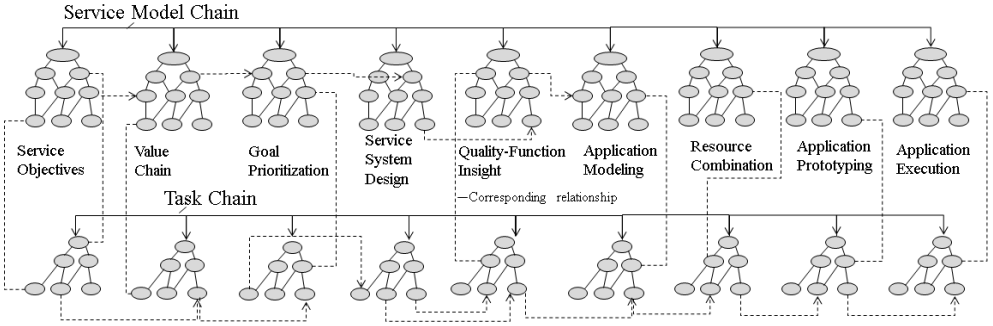

図 5-5 サービスモデルチェーンとタスクの関連付け［Hosono2013b(改)］

開発工数は，プロジェクト管理ツールから取得できる．プロジェクト管理ツールは，予定工数を書いたタスクチケットを発行し，実績値と合わせることができる．これらのステークホルダのリソース情報は，タスクチェーンに統合する（図 5-6）．そして，サービスモデルチェーンとタスクチェーンを共通リポジトリに保持し，次回以降の開発に活用できる．タスクチェーンに含まれる実績値は，すなわち，人のコスト情報であるため，これをサービス開発に必要な標準コストと見做せる．これを，プロジェクト管理に用い，他の Web サービス開発をする際の尺度として比較し，進捗状況の進度の測定が可能になる．

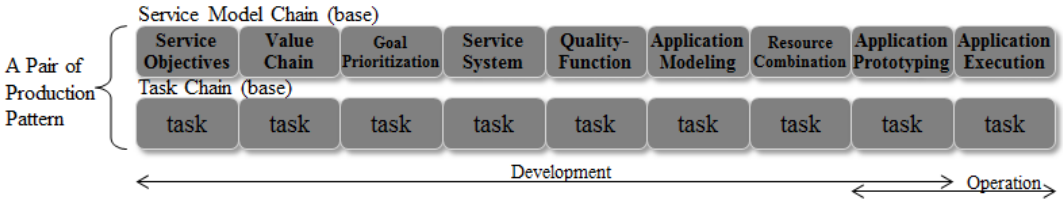

図 5-6 サービスモデルチェーンとタスクチェーン［Hosono2013b(改)］

第 4 章で示したライフサイクルパタンには，人の作業情報は含まれていなかったが，以上に示した方法により，ライフサイクルパタンに実績情報の付与が可能となる．

以上に示した方法を体系化し，フレームワークを構築した．このフレームワークに含まれる技法を現場の開発プロセスに適合させ，ライフサイクルパタンの構築～利用を定着させるために，開発者の協働を支援する方法について述べる．

5.3.2 ライフサイクル間の協働支援

5.3.2.1 再利用戦略ポリシー

クラウド環境を利用したシステム構築では，クラウド上の検証済みの機能を利用することから，短期に開発・本番実行できることがシステム開発の要件ではなく，前提条件となる．そのため，開発時点において，ソフトウェアやサービスの機能要件に加えて，運用管理の保守性や改良のし易さなど，実装・実行工程での要件や制約を合わせた考慮が必要とされる．

セットベース設計は，設計者間の不確定性を考慮したもので，設計者の設計意図を反映した選好度を導入し，多目的を折衷した設計仕様を範囲値として導出することで，開発中に変更の少ない製品設計を可能にする．この考え方を応用し，作られた資産の意図と，設計工程や運用工程の意図や制約を擦り合わせて，最も有利なライフサイクルパタンを特定する．具体的には，開発時点において，ソフトウェアやサービスの機能要件に加えて，運用管理の保守性や，改良のし易さなど，実装・実行工程での要件や制約を併せて考慮するため，開発プロセスに沿って，設計工程の設計意図（designers' intention）と，運用工程の実行意図（operators' intention）を合成し，その結果と最も適合するライフサイクルパタンを特定する．企画者や設計者などによる設計観点だけでなく，後工程にある実装者，運用者などの実行観点（制約）をも予め考慮して，適合性の高いソフトウェアや文書などのライフサイクルパタンをリポジトリから絞り込んで抽出する．

一方，ライフサイクル間を跨って，過去の案件の開発文書を参照し，活用する際，必ずしもファイル全てをそのまま再利用できるとは限らず，個人情報を除いた後のファイルや，ファイルは破棄し記憶領域のみを再利用することになる．このような利用時の判断や処理が都度必要になるのは，主にライフサイクルパタンの構築時に再利用のシナリオや方針・意図が情報として残されていないことに拠る．そこで，ライフサイクルパタン構築時点において，再利用計画を細分してデータ化できるように，再利用戦略ポリシーを Reuse，Remanufacturing，Recycling に関するルールで定義し，ライフサイクルパタンの管理に用いる．

再利用戦略ポリシーは，ライフサイクルパタンの再利用性の更新方法を示す情報として定義する．再利用戦略ポリシーは，ライフサイクルパタンを特定する条件と，その条件に当てはまるライフサイクルパタンの更新方法を対応付けて示す．例えば，ライフサイクルパタンを特定する条件は，ライフサイクルパタンの ID や属性を用いる．ライフサイクルパタンの名称や構造, 属性, 属性値などの特徴に基づき, そのまま再利用できるものを Reuse，再構成すれば再利用できるものを Remanufacturing，素材にまで分解した後に再利用できるものを Recycling とした，再利用戦略ルールとする（図 5-7）．このルールをライフサイクルパタン保管時の分別手段，利用時の検索手段として用い，それぞれ対応するリポジトリに保管することで，ライフサイクルパタン利用時点において，登録時の再利用方法の想

定を反映した，細粒度構成のライフサイクルパタンを開発者に提示できる．また，ライフサイクルパタンの構築後に再利用戦略ポリシーを追加することで，例えば，ライフサイクルパタンを構成するライブラリや部品の仕様が変更されるように，ライフサイクルパタン構築時に予見しなかった変更に対して，ライフサイクルパタンの利用時に抽出手段として用いる．

その他にも例えば，ライフサイクルパタンの再利用性が関数で表される場合，ライフサイクルパタンの再利用性は，ライフサイクルパタンの構築時点からの経過時点に応じて数値として算出できる．再利用性を示す関数（有効資産関数）は，例えば，ライフサイクルパタンを構築する構築期間，ライフサイクルパタンの安定利用が予想される安定期間，及びライフサイクルパタンの部分的な改変が予想される部分改変期間の各区間における故障率の変化の仮説をワイブル分布等によりモデル化し，バスタブ曲線等で表す．

また，再利用戦略ポリシーは，ライフサイクルパタンの有効期間や条件などタイマーを示すルールも定義できる．このタイマールールをライフサイクルパタンのリポジトリに連動させることで，例えば，一定時間経過の後，特定のライフサイクルパタンをリポジトリから削除するなど，無効になったライフサイクルパタンの自動管理手段として用いる（図 5-7）．

```
// flag rule
if $Asset.name has <name>
    then $Asset.flag = { REUSE | REMANUFACTURING | RECYCLING }

// asset timer
if ($Asset.date == <date>)
    then $Asset.flag = { REUSE | REMANUFACTURING | RECYCLING }

// content mask
if $Asset.content contains $ProjectSpecificWord
    then $Asset.content.replace(<from_name>, <to_name>);
```

図 5-7 再利用戦略ポリシー［Hosono2014(改)］

5.3.2.2 再利用戦略ポリシーとライフサイクルパタンの統合

再利用戦略ポリシーと，ライフサイクルパタンの統合は，次の仕組みとステップで実現する．まず再利用戦略ポリシーを入力し，再利用戦略ポリシーのリポジトリに保管する．次に，再利用戦略ポリシーがタイマー監視に関するかを判定し，関するものはライフサイクルパタンの監視機構にて，監視を開始する．次に，ライフサイクルパタンとして登録するファイル等の基本データを入力する．次に，ライフサイクルパタンの元となる基本データに，保管済みの再利用戦略ポリシーを適用し，Reuse，Remanufacturing，Recycling に適合したライフサイクルパタンをそれぞれ Reuse リポジトリ，Remanufacturing リポジトリ，Recycling リポジトリに振り分けて保管する．各リポジトリを常時監視し，再利用戦略ポリシーのタイマー条件に従って，例えば Reuse リポジトリに保管されたライフサイクルパタンは改変が必要になったと判断し，Remanufacturing リポジトリに移す．

新たなサービスを開発する場合は，次のようにライフサイクルパタンを取得する．適用計画と，現状のサービスシステム構造を示す，現状のサービスシステム構造（As-Is サービスシステム構造）と，目的のサービスシステム構造（To-Be サービスシステム構造）を入力する．As-Is サービスシステム構造と，To-Be サービスシステム構造の差分に適合するライフサイクルパタンを, Reuse リポジトリおよび Remanufacturing リポジトリから検索する．（ここで，Recycling リポジトリのライフサイクルパタンは素材に分解されており，再利用の準備コストが高いため，検索対象から外す.）

取得したライフサイクルパタンをそのまま再利用できない場合は，当初の仮説に基づく再利用戦略ポリシーが適正でないと判断し，補正手続きを行う．例えば，Reuse にあり，そのまま利用できずに変更を行った場合は，再利用戦略ポリシーの振り分けルールを Remanufacturing に変更する．同様に，Remanufacturing に在って再利用時に分解が必要になった場合は，Recycling に変更し，リポジトリ中の再利用戦略ポリシーを上書きする．更に，これらの変更は対象のライフサイクル見積りとの差異を示すものであるため，この差異に基づき有効資産関数を補正する．

以上のライフサイクルパタンを用いて，ライフサイクルパタン構築と利用の投資対効果をシミュレーションにより把握できる．適用計画を入力すると，ライフサイクルパタン検索部を通じてライフサイクルパタンを Reuse リポジトリ，Remanufacturing リポジトリから検索し，ライフサイクルパタンの効果を計算し，結果を表示・描画する．この表示結果をライフサイクルパタンの導入計画・施策の立案に活用する．

5.3.2.3 意図の統合による再利用に適した資産の抽出

開発時点において, ソフトウェアやサービスの機能要件に加えて, 運用管理の保守性や, 改良のし易さなど，実装・実行工程での要件や制約を併せて考慮するため，開発プロセスに沿って, 設計工程の設計意図（designers' intention）と, 運用工程の実行意図（operators' intention）を合成し，その結果と最も適合するライフサイクルパタンを特定する．企画者や設計者などによる設計観点だけでなく，後工程にある実装者，運用者などの実行観点（制約）をも予め考慮して，適合性の高いソフトウェアや文書などのライフサイクルパタンをリポジトリから絞り込んで抽出する．次に，設計意図・実行意図の統合方法とライフサイクルパタンの抽出手順を具体的に示す.

5.3.2.4 ライフサイクルパタンの抽出手順

まず，開発プロセスに沿って，ソフトウェア開発における V 字型モデルのように設計工程と実装・実行工程のステークホルダを対応付けてモデル化する．更に，設計工程での想定値と実際の状況や制約との差異を測るため，設計工程と実装・実行工程の各工程をペアで管理する．図 5-8 に示すように，このモデルデータを上流から下流工程までを順に完成させていくことで，ライフサイクル上の多視点を漏れなく抽出する．モデルデータの各要素には，ある要件（設計変数）に対する目的設計変数（objective design parameter）の可能

性分布を設計意図，実行意図として保持する．

再利用に用いるライフサイクルパタンの絞り込みは，この可能性分布と，目的設計変数のとり得る値を実行可能性分布（possible requirements）（図 5-8）を用いる．ある目的設計変数に関して，設計意図からの可能性分布と，実行意図からの可能性分布を統合した結果と，実行可能性分布の共通範囲を設計解とする．他の目的設計変数も同様に行い，解集合を得る．これらの解集合に近い値を包含するライフサイクルパタンを最も再利用に適したライフサイクルパタンとして同定し，Reuse または Remanufacturing リポジトリから抽出する．この手順の具体例を次節に示す．

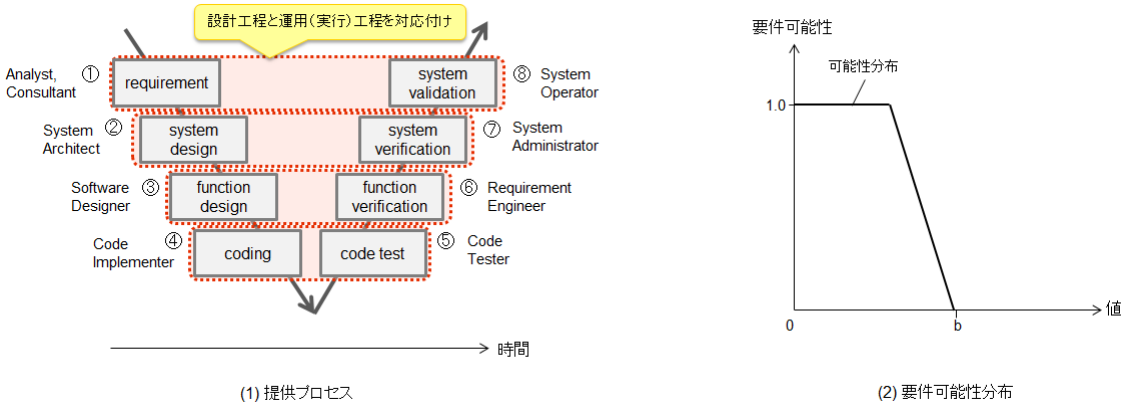

図 5-8 設計意図と運用意図の対応および可能性分布［Hosono2014(改)］

5.3.2.5 ライフサイクルパタンの抽出例

性能やセキュリティなど要件(設計変数)に関する設計者の意図分布を示す設計意図と，実行者の意図分布を示す実行意図と，目的設計変数名を入力する．ここで，設計意図および実行意図は，実際の分布状態を近似するために，分布曲線として正規分布を用いる．（この分布曲線は，過去の Web サービス開発でのソフトウェアモジュールやリソースの持つパラメータの設定実績とその分布に拠る．ここでは，汎用性の高い）次に，設計変数の設計意図および，実行意図のそれぞれについて，目的設計変数の導出関数を用いて，目的設計変数とその可能性分布を導出する（図 5-9）．

抽出する際には，まず，目的設計変数のとり得る値を実行可能性分布と，上記の可能性分布の共通範囲を設計解とする．（無い場合には，設計解が存在しないと判断して，ライフサイクルパタンの検索を行わずに停止する．）目的設計変数名の導出元となった設計変数名を用いて，Reuse，Remanufacturing リポジトリからライフサイクルパタンを検索し，目的設計変数と合致するライフサイクルパタンを抽出，出力する．

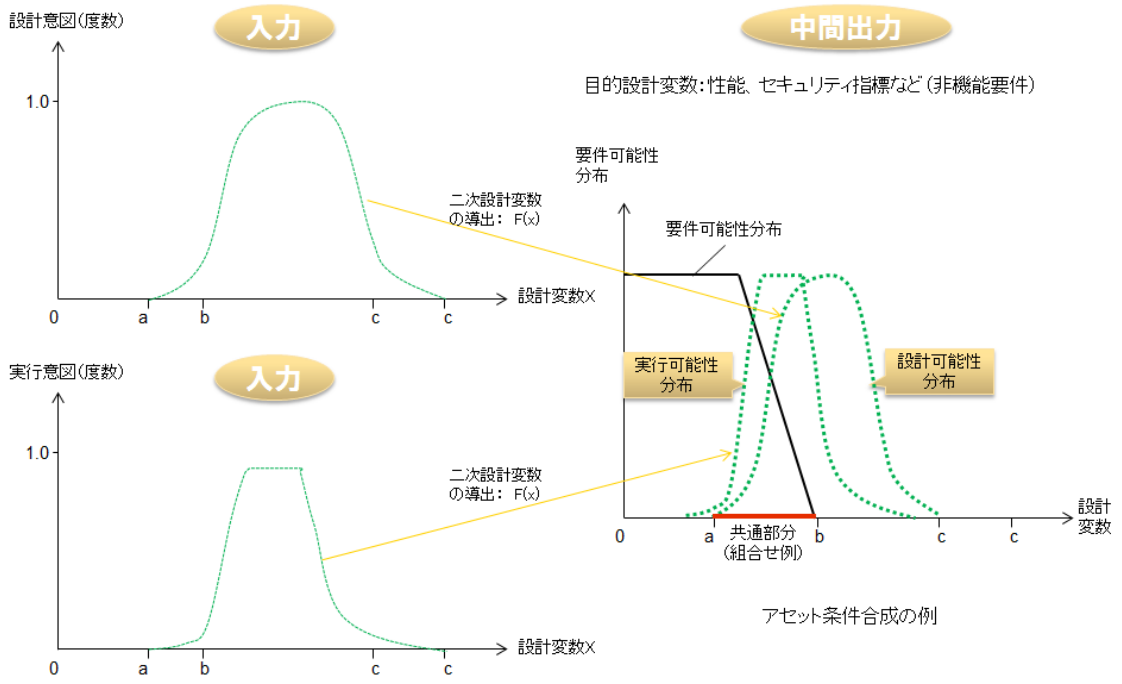

図 5-9 設計意図・実行意図の合成の例［Hosono2014(改)］

以上により，ステークホルダの意図が開発文書などの成果物と共に形式化され，適切なライフサイクルパタンを特定して利用できる．これにより，設計者と運用者間およびライフサイクルを跨った設計者間の協働が容易になる．

5.3.3 ライフサイクル知識の構築・活用を定着させる規律

5.3.3.1 現場の開発プロセス

第 3 章，第 4 章で示したモデリングの適切な定義の仕方とこれを構築・管理・適用する方法が有効である．しかし，これらの方法を用意しただけでは，開発現場で活用されるとは限らないため，開発現場に定着させることが必要である．

現場のサービス開発プロセスは，業種や業種内の領域に応じて異なった標準が用いられている．従来，新たなソフトウェア技術が開発される度に，汎用的な開発方法論が作られ，これらの現場の開発プロセスを刷新すべく，統一的な導入が図られてきた．しかし，現場の開発方法・プロセスを代替するには，その業種 SI の固有の要件に合わせて擦り合わせが必要であり，また，新たな方法論導入の教育コストが大きいことから，必ずしも新たな開発方法論・プロセスの導入は成功していない．

そこで，資産であるライフサイクルパタンの構築と再利用に基づく開発方法論として，現場の開発プロセスの代替を志向するのではなく，現場のサービス開発プロセスを尊重したプロセス適合技術を開発する．資産の設計・構築，集約・管理，流通・再利用の各スキ

ームで，開発者が達成すべき項目を規定することや，項目の達成状況から資産化の状態を可視化し，開発マイルストンに向けて次に取るべきアクションのガイドが必要になる．

5.3.3.2 SEMAT カーネルアルファ

現場の開発プロセスに適応させる方法として，近年，ソフトウェア工学では，Jacobson らが提唱する SEMAT（Software Engineering Method and Theory）が提唱されている［Jacobson2013］．これは，成果物を基点とし開発工程の進度を状態でモデル化する考え方である．この考え方は，従来の新たな開発方法論を開発し現場に押し付けるアプローチとは異なり，現場の開発プロセスと成果物を基点とした，開発プロセスの管理を可能とする．SEMAT は，理論，証明された原理，およびベストプラクティスに基づき，ソフトウェア開発方法の再建を目指したものである［Jacobson2013］［OMG2013］．SEMAT では，アルファ（alpha, abstract-level progress health attribute）によって，活動すべき目標を指示できるようにしている．この活動すべき目標の観点（関心事）の単位でアルファを定義し，アルファ全体でカーネルを構成する．カーネルアルファは，成果の達成に着目し，開発活動の進捗と健全性について，個別ではなく全体を検討可能にする．具体的には，カーネルアルファは Customer，Solution，Endeavor の 3 つの基本関心事で構成される．Customer については，実際の使い方と開発するソフトウェアシステムに関する事項が含まれる．Solution についての関心は，ソフトウェアシステムの仕様と開発に関連する事項が含まれる．Endeavor に関する関心は，開発チームと，その仕事のやり方に関連する事項が含まれる．カーネルは基本アルファとして，Opportunity，Stakeholder，Requirement，Software System，Team，Work，Way of Work のアルファを定めている．各アルファは，ソフトウェア開発の進捗状況や健康状態を理解するためのチェックリストを規定している．アプリケーションソフトウェアは，プロトタイピングを繰り返して開発する際，アルファカードは，開発チームが開発工程のどこに位置しているか，把握し易くし，次にどうすべきか指針を示す．ライフサイクルを通じてアルファカードを揃えることにより，開発チームは，次のように，開発プロセスの位置を理解し，バランスのとれた，まとまりある方法で開発を進められる（図 5-10）．

1. 最初の状態を右側，最後の状態を左側になるように，テーブル上にアルファカードを並べる
2. 各状態を確認し，その状態を達成しているか否かを調べる
3. 状態を達成した場合，左にアルファカードを移動する．達成していない状態になるまで，この手順を繰り返す
4. 上記で特定したカードと残りのアルファカードを右に移動する

チームメンバで，このカードを用いて現在の開発状態を特定すると，次に達成すべき状態は何か，認識を合わせることができる．また，次の反復（イテレーション）開発で次に

ターゲットとすべき状態はどれかを選択できる．チェックリストは完了の基準を明示するため，次に目標とすべき状態を整理して確認できる．

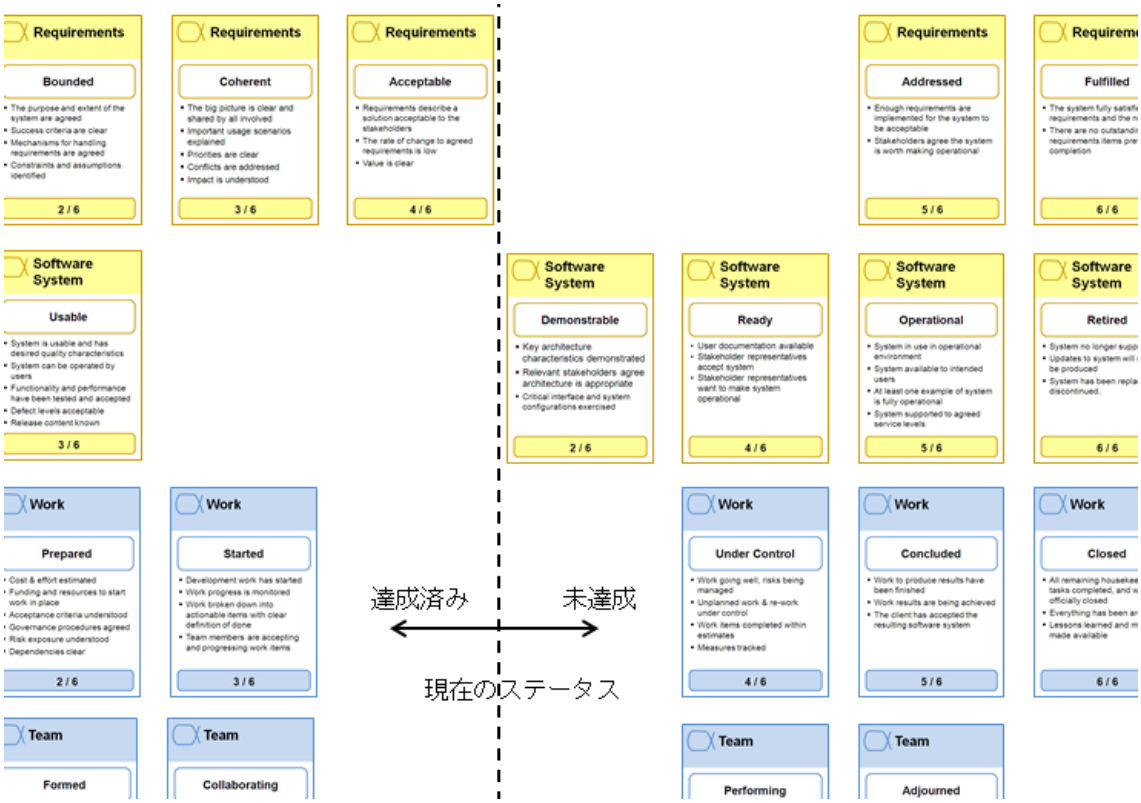

図 5-10 アルファカード［Jacobson2013］

5.3.3.3 資産化適応アルファ

カーネルアルファを，サービス開発向けに拡張し，新しいサービス開発に規律を与える．アルファの“Software System”を，“Service System Platform”として再定義する．また，新たなアルファとして**資産化適応アルファ**（Asset Adaptation Alpha）を定義する．

資産化適応アルファは，(1)再利用に適した資産の特定，(2)資産の適合性判断，(3)資産のカスタマイズ利用，(4)資産の再利用計画，(5)資産の構築，(6)資産の有効性評価，の状態として構成する（表 5-1）．「再利用に適した資産の特定」は，関連したサービスモデルチェーンの確認，資産リポジトリから再利用するためのサービスモデルチェーンの選択，および，後から表出した要件を満たせるサービスモデルチェーンの選択のタスクで達成する．「資産の適合性判断」は，サービスモデルチェーンの中から利用可能な成果物を特定，および，利用のための要件と作業成果物の違いを識別のタスクで達成する．「資産のカスタマイズ利用」は，作業成果物を利用に向けたカスタマイズ，および，適合するサービスモデルチェーンが無い場合に再利用のために新しい成果物を作成するタスクで達成する．「資産の再利用計画」は，再利用に適した成果物の指定，および，成果物の再利用計画の立案のタスクで達成する．「資産の構築」は，成果物のセットをリポジトリにサービスモデルチェ

ーンとして格納, および, リポジトリに格納されている再利用計画のタスクで達成する.「資産の有効性評価」は，成果物の有効性を評価，および，リポジトリ内のサービスモデルチェーンのリフレッシュのタスクで達成する.

ライフサイクルの過程で，これらの状態を監視し，状態を達成するためのタスクを律することで，資産化のプラクティスを開発の現場に定着できる．前節に示したアルファカードとして資産化適応アルファを実践することで，開発現場で行うライフサイクル知識の構築，選択，カスタマイズ，再利用のステップに規律を与えられる．この状態による管理方法は，これまで各開発現場で実践されてきた開発プロセスを損なうことなく，適応させて組み込める.

以上に示した資産化適応アルファを含めて,拡張したカーネルアルファを示す(表 5-2).

表 5-1 資産化適応アルファ

A	再利用に適した資産の特定	✓ 関連したサービスモデルチェーンの確認
		✓ 資産リポジトリから再利用するためのサービスモデルチェーンの選択
		✓ 後から表出した要件を満たせるサービスモデルチェーンの選択
B	資産の適合性判断	✓ サービスモデルチェーンの中から利用可能な成果物を特定
		✓ 利用のための要件と作業成果物の違いを識別
C	資産のカスタマイズ利用	✓ 作業成果物を利用に向けたカスタマイズ
		✓ 適合するサービスモデルチェーンが無い場合, 再利用のために新しい成果物を作成
D	資産の再利用計画	✓ 再利用に適した成果物の指定
		✓ 成果物の再利用計画の立案
E	資産の構築	✓ 成果物のセットをリポジトリにサービスモデルチェーンとして格納
		✓ リポジトリに格納されている再利用計画
F	資産の有効性評価	✓ 成果物の有効性を評価
		✓ リポジトリ内のサービスモデルチェーンのリフレッシュ

表 5-2 拡張したカーネルアルファ

Concern （関心）	Alpha （アルファ）	State / Checklist 1 （状態）	State / Checklist 2 （状態）	State / Checklist 3 （状態）	State / Checklist 4 （状態）	State / Checklist 5 （状態）	State / Checklist 6 （状態）
Customer	Stakeholders	Recognized: - Stakeholders have been identified - There is agreement on stakeholder groups to be represented - Responsibilities of stakeholder representative defined	Represented: - Stakeholder representatives appointed - Stakeholder representative agreed to take on responsibilities and authorized - Collaboration approach agreed - Representatives respect team way of working	Involved: - Stakeholder representative carry out responsibilities - Stakeholder representative provide feedback and take part in decisions in timely way	In Agreement: - Stakeholder representatives agree their input is valued and respected by the team - Stakeholder representatives agree with priorities	Satisfied for Deployment: - Stakeholder representatives provide feedback on system from their stakeholder group perspective - Stakeholder representatives confirm system ready for deployment	Satisfied in Use: - System has met or exceed minimal stakeholder expectations - Stakeholder needs and expectations are being met
Solution	Requirements	Conceived: - The need for a new system is clear - Users are identified. - Initial sponsors are identified	Bounded: - Success criteria are clear - Mechanisms for handling requirements are agreed on - Constraints and assumptions are identified	Coherent: - The big picture is clear and shared by all involved - Important usage scenarios are explained - Priorities are clear - Conflicts are addressed	Acceptable: - Requirements described a solution acceptable to the stakeholders - The rate of change to agreed-on requirements is low - Value is clear	Addressed: - Enough requirements are implemented for the system to be acceptable - Stakeholders agree the system is worth making operational	Fulfilled: - The system fully satisfies the requirements and the need - There are no outstanding requirement items preventing completion
	Service System Platform	Architecture Selected: - Architecture selected that address key technical risks - Criteria for selecting architecture agreed - Buy, build, and reuse decisions made	Demonstrable: - Key architecture characteristics demonstrated - Relevant stakeholders agree architecture is appropriate - Critical interface and system configurations exercised	Usable: - System is usable and has desired quality characteristics - System can be operated by users - Functionality and performance have been tested and accepted - Defect level acceptable.	Ready: - User documentation available - Stakeholder representatives accept system - Stakeholder representative want to make system operational	Operational: - System in use in operational environment - System available to intended users -- System supported to agreed service levels	Retired: - System no longer supported - Updates to system will no longer be produced - System has been replaced or discontinued
	Asset Adaptation	Asset Identified: - Existence of related production patterns confirmed - A production pattern for reuse selected from asset repositories - Work products for reuse identified to meet afterwards identified requirements	Difference Identified: - Work products in the production pattern for reuse determined - Differences between requirements and the work products for reuse identified	Asset Adapted: - Work products for reuse customized - New work products for reuse created when no production pattern is applicable	Reuse Planned: - Work products specified for reuse - Reuse of work products planned	Asset Stored: - A set of work products stored as a production pattern into repositories - Reuse plan stored into repositories	Validity Assessed: - Validity of work product assessed - Production patterns in repositories refreshed
Endeavour	Way of Working	Principles Established: - Principles and constraints established - Principles and constraints committed to - Practices and tools agreed to - Context learn operates in understood	Foundation Established: - Key practices and tools ready - Gaps that exist between practices and tools analyzed and understood - Capability gaps analyzed and understood - Selected practices, and tools integrated	In Use: - Use of practices and tools regularly inspected - Practices and tools being adapted and supported by team - Procedures in place to handle feedback	In Place: - All members of the team are using the way of working - All members have access to practices and tools to do their work - Whole team involved in inspection and adaptation of way of working	Working Well - Way of working is working well for team - Team members are making progress as planned - Team naturally applies practices without thinking about them - Tools naturally support way of working	Retired: - Way of working no longer in use by team - Lessons learned are shared for future use

5.4 おわりに

本章では，資産開発時の想定と異なる利用時点の環境や，ステークホルダの意向に対して，適応性の高い資産の提供を可能にした．カーネルアルファ導入により資産化・再利用プラクティスに規律を与えた．

第 2 節では，パタン化の取り組みと，その再利用の課題を分析した．そして，ステークホルダが協働して開発する際の課題を明らかにした．

第 3 節では，ライフサイクル知識の構築・活用プロセスを説明した．顧客と IT サービスプロバイダ間の協働支援，開発部門間の協働支援，および開発部門と企画・管理部門との協働支援について述べた．更に，これらの技法・協働活動を定着させるための規律について述べた．

クラウド環境の浸透に伴い，開発サイクルが早まり，顧客の課題が明示的でなく，サービスを探索していく場合，協働作業を伴う繰り返し行い開発を進める必要があった．このとき，プロダクトラインエンジニアリングの考え方で，コア資産のカスタマイズによる進め方だけでは，顧客と合意形成して開発を進められないため，ステークホルダの意図やタスクを基点に設計・運用プロセスを管理する必要があった．

本研究のアプローチは，企画～運用までにサービス提供に関わるステークホルダの役割と，ステークホルダ間のプラクティスを起点に知識構築・活用プロセスを扱った．サービスシステムおよびライフサイクル全体を通じて，主要なステークホルダ間のインタラクションに，①顧客と IT サービスプロバイダ間，②設計部門間（ライフサイクル間），③開発と管理部門間の協働があることに着目し，そのインタラクションの中でライフサイクル知識を有効に活用させるタスクを設計した．

本アプローチのポイントは，① 提供者側の設計者・運用者間（工程間）および提供者と受給者間，② 管理者と開発者（設計者および運用者）間（上位視点・下位視点），③ プロジェクト間（ライフサイクル間）の開発者間のそれぞれについて，構築したライフサイクル知識の活用方法を示した点と，④これらの方法を定着させるための規律を定義し，実践方法を示した点にある．

従来の研究では，開発方法論や成果物などのデータ起点に置いていたが，本研究では役割やプラクティスなどのステークホルダ起点に置いた．本研究の特長は，多様なステークホルダとその関係（アクターネットワーク）をサービスシステムおよびサービスライフサイクルの主要な構成要素として扱い，ステークホルダ起点でライフサイクル知識の構築と利用タスクを設計している点にある．

本研究で示したように，再利用戦略ポリシーの導入では，設計の背景，目的，などを戦略という形で，抽象化した．また，設計・運用プロセスのパタン化では，一連のサービスモデルチェーンに加えて，各設計モデルの構築工程におけるステークホルダ間の協働作業をデータとして扱い定量化する方法を示した．また，ステークホルダのタスクを必ず実践するように促すため，カーネルアルファを用いて，各工程でのタスクを明らかにした．このように，人の活動を可視化し，設計・運用プロセスの中で，明示的に扱えるようにしたことで，資産を有効に活用することが可能となった．プロセス指向であった開発方法をタスク基点に見直したことで，ステークホルダと協働してサービスの開発が可能となった．これにより，モデルの作成や利用を行う際に，ステークホルダのアクティビティに，ステークホルダを十分に考慮したサービスライフサイクル管理方法に変えられた．

1. **ステークホルダの役割に基づき，顧客とプロバイダ間，開発部門間，および開発部門と企画・管理部門間の協働支援**
2. **協働作業を促進・実践させるプラクティス**

を示した．

以上により，サービスのライフサイクルを通じて効果が最大化され，IT サービスプロバイダは，効果的に彼らの能力を発揮することができる．これにより，ステークホルダが協働し，合理的にかつ効率的に Web サービスを設計し，運用できる．

一方，次の課題が残されていることも明らかとなった．

サービスシステムの構築過程において，関わるステークホルダは，その役割の達成に至るまでに，幾つかの項目を達成する必要がある．ある時点において行う必要がある項目にのみ関心があたっているため，ステークホルダ間の協働作業において意思疎通が行き届かない原因となる．そこで，役割に関連付けられる関心事に着目し，その関心に合った細粒度の情報の提示や見落としているタスクの提示方法が必要である．これには，アクターネットワークを構成するステークホルダの役割と，各役割の持つ関心事をモデル化するアプローチが考えられる．

また，更なる開発の効率化に向けて，製品指向（product-oriented），利用指向（use-oriented），成果指向（result-oriented）のサービス特性に基づく，協働作業の類型化や，サービスの廃棄あるいは継続的改善の判断方法が必要である．

第６章 ライフサイクル管理方法論の実践と検証

目次

6.1 はじめに

本章では，第 3 章，第 4 章，第 5 章において提案した，サービスライフサイクル管理方法を IT サービス開発の現場で実践し，クラウドに対応した Web サービスの開発・提供事例に適用することで，その有効性について検証を行った．

第 2 節では，サービスライフサイクル管理方法論の実践について述べる．まず，サービス LCM フレームワークを示す．このフレームワークは，モデリングツールと，サービスリポジトリで構成する．モデリングツールは，ゴールモデリング，システムモデリング，リソース割り当て，プロトタイピング環境，およびサービス監視を備える．サービスポートフォリオは，パブリック・サービスポートフォリオとプライベート・サービスポートフォリオで構成する．また，協働支援方法を，現場で行われているプロジェクト管理の仕組みに統合し，実践する．

第 3 節では，検証対象となるクラウドを対象とした Web サービスの概要を述べる．

第 4 節では，サービス LCM フレームワークの適用結果について述べる．

第 5 節では，サービスの生産性向上の観点と，ビジネスモデル（ビジネス上のエコシステム）の観点で議論する．

6.2 サービスライフサイクル管理方法論の実践

6.2.1 サービス LCM フレームワーク

6.2.1.1 サービス LCM フレームワークの概要

本節では，第 3 章，第 4 章，第 5 章で提案した方法の実践として，**サービス LCM フレームワーク**について述べる．

第 3 章に示した，細粒度にした各モデリング方法を設計ツールとして，実装した．また，4 章に示したライフサイクル知識の資産化と活用の仕組みを，設計ツールで開発した情報を一元管理し，取り出せるようにしたリポジトリとして実装した．また，第 5 章に示したライフサイクル知識の構築・活用プロセスを，設計ツールとリポジトリとの情報のやり取りの仕組みに実装し，更に一般のプロジェクト管理ツールと統合した．

本フレームワークを構成する設計ツールは，次のモデリングツール群で構成した（表 6-1）．

表 6-1 サービス LCM フレームワークのモデリングツール群

	目的	ツール名	モデリング内容
1	顧客の要求を構造化	*Service Objectives*	サービスの目標を定義する
2	顧客の要求を構造化	*Stakeholder Comprehension*	サービス提供までのバリューチェーンを定義する
3	機能要件を満たすシステム設計	*Goal Prioritization*	目標を実現する要件と，要件を実現する機能を定義する
4	リソースを考慮したステム設計	*Service System Design*	機能モジュールと，機能モジュール間の関係を定義する
5	リソースを考慮したステム設計	*Application UML Modeling*	アプリケーションソフトウェアコンポーネントを定義する
6	リソースを考慮したステム設計	*Resource Combination Design*	アプリケーションのリソースへの配置を定義する
7	機能要件を満たしているか	*Application Prototyping*	アプリケーションのプロトタイプ開発と機能検証を行う
8	非機能要件を満たしているか	*Cloud Execution Stack, Cloud Controller*	クラウドの実行リソースを制御する，アプリケーションの非機能検証を行う
9	設計・運用情報が工程間で分断されない	*Service Portfolio*	アプリケーション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

本フレームワークを構成するリポジトリは，モデリングツール群で作成・利用する情報を一元管理し，かつ再利用するためのリポジトリとして動作するように実装した（図 6-1）．

この一連のモデリングツール群は，表 6-1 で示したモデル群に対応する．各モデルの実装例を付録 A に一覧する．

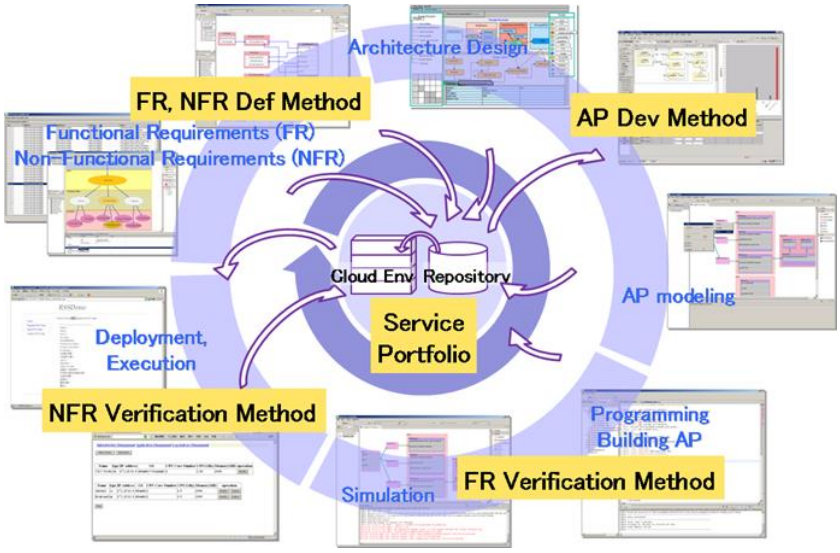

図 6-1 フレームワークの概要［Hosono2012b(改)］

このフレームワークの実装アーキテクチャを図 6-2 に示す．

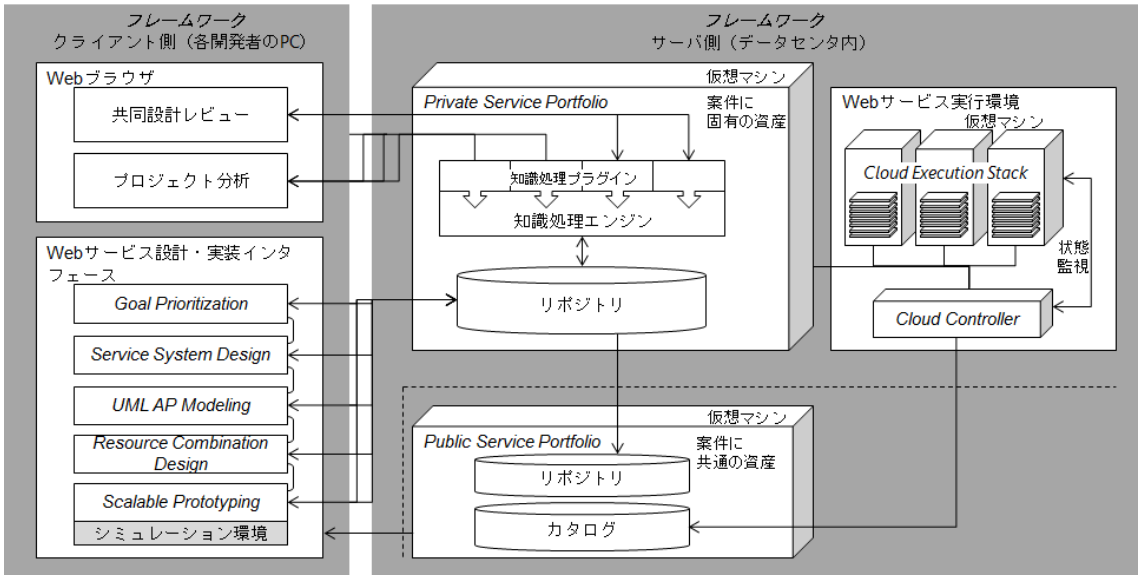

図 6-2 アーキテクチャの概要［Hosono2012b(改)］

次節より，各モデリングツールの詳細を示す．

6.2.1.1 ゴールモデリング

要求-機能展開モデリングを行うツール *Goal Prioritization* は，Web サービスを構成する個々の機能要件と，性能，保守性，拡張性，セキュリティ等の Web サービス全体に対する非機能要件を定義する．例えば，セキュリティの実現手段は認証機能で実現するように，非機能要件の多くは機能要件に紐付けられる．一方，高可用性の確保には，ハードウェア機能要件に加え，運用管理者による通知などのアクティビティ（機能要件）も必要となる．このように個々の非機能要件を実現する，物理的，あるいは人手で賄う機能をツリー構造で記述し，非機能要件と実現機能の転写関係を構造化する（図 6-3）．

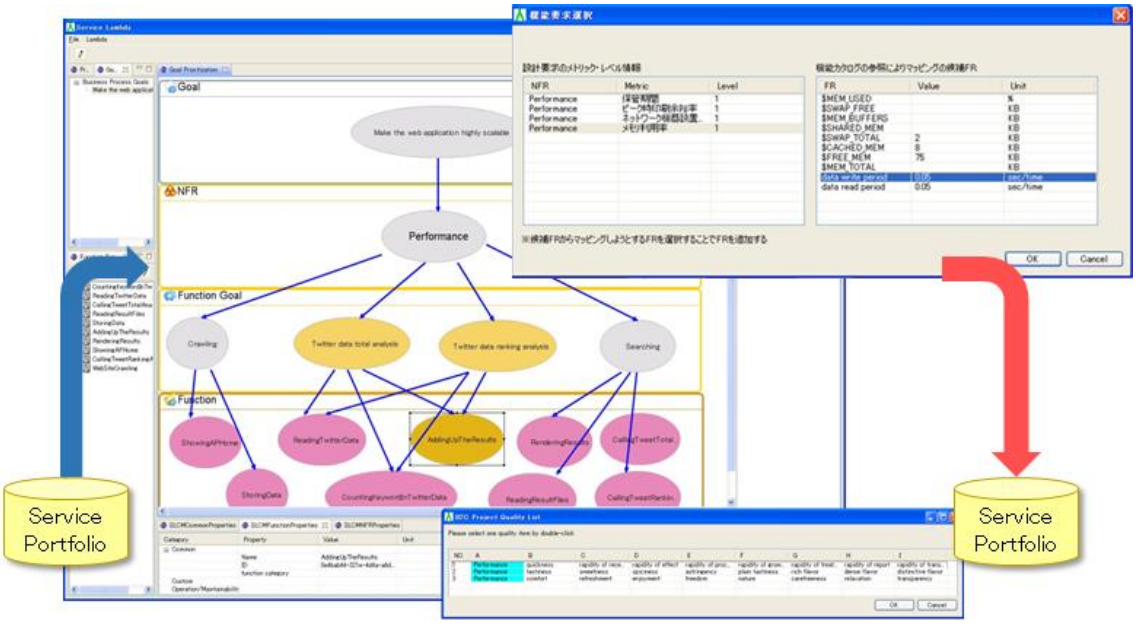

図 6-3 要求-機能展開モデリング画面の例［Hosono2012b(改)］

次に，**機能-リソースカタログ**を表示するリソース範囲提示画面について述べる．前述の機能要件は，クラウド上で利用可能な機能とそのパラメータで定義する必要がある．しかし，クラウド環境では，運用中に，他の Web サービスが同じハードウェアリソースを共有している場合に影響を受け，クラウド上のリソースの状態や振舞いが変化し得る．例えば，複数の利用者・企業の Web サービスが同時にストレージに書込み処理を行う場合，リソース競合が起こり，各々の Web サービスの動作が遅くなる等の事象が発生するため，利用可能なパラメータ範囲が変化し得る．このように，運用側に委ねられたリソースの状態を知る必要があるため，サーバ側のリソースの監視機構からリソースの振舞いや利用可能な性能範囲を得て，表示する．また，非機能要件項目とそのレベルに対して，対応するリソース群とその性能値のルールを予め保持する．このルールを用いて，リソース群と，その値を動的に更新するカタログを構築する（図 6-4）．このカタログの提示により，要件定義の際に，実際にクラウド環境で利用できるリソースの量や状態を常に確認できるため，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．結果として，後工程での手戻りの発

生リスクを低減できる.

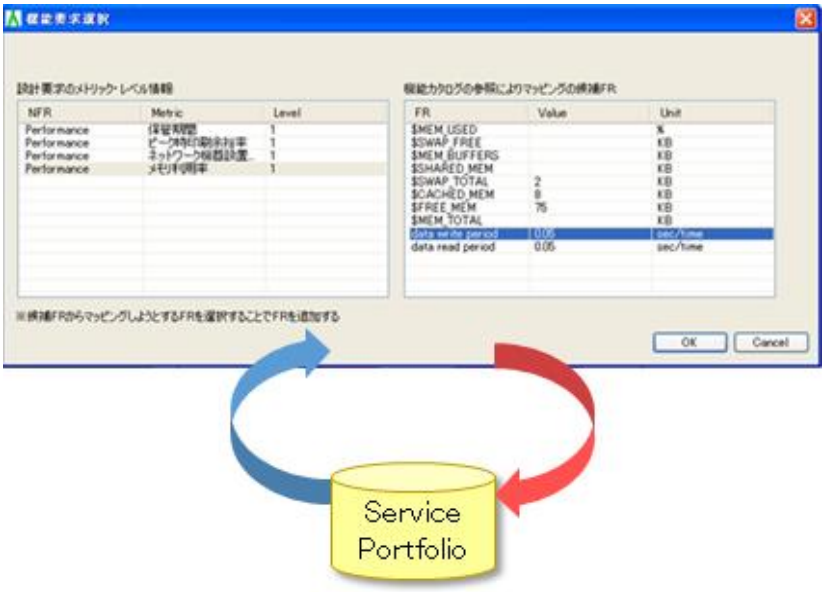

図 6-4 リソース範囲提示画面の例［Hosono2012b(改)］

6.2.1.2 システムモデリング

サービスシステムモデリングを行うツール *Service System Design* は，定義された機能・非機能要件を元に，サービスを構成する機能フローをモデリングする．これは，Web サービスの利用者 (ペルソナ) の振舞いを中心にモデリングを行う．*Service System Design* の利用者が Web サービスプロバイダ側の個々の機能を利用するシナリオを記述し，提供者側の機能と機能提供を行う担体を特定する．シナリオは，ユースケースを時系列に沿って，サービスを**認知**（Access），その内容を**確認**（Check-in），その内容を**評価**（Diagnosis），その機能を**利用**（Delivery），機能の利用を**完了**（Check-out），利用後に**評価**（Follow-up）する体験過程に分解し，各過程で行う利用者のアクションを記載する．このアクションに対応する，提供者側のサービス機能（物理的および人手で賄う機能）を対応づける．この操作を繰り返し，サービスシステム全体をモデリングし，システムの境界を同定する（図 6-5）．このサービス境界内において実装機能を取捨選択し，費用対効果の高いサービスシステム構成を決定する．

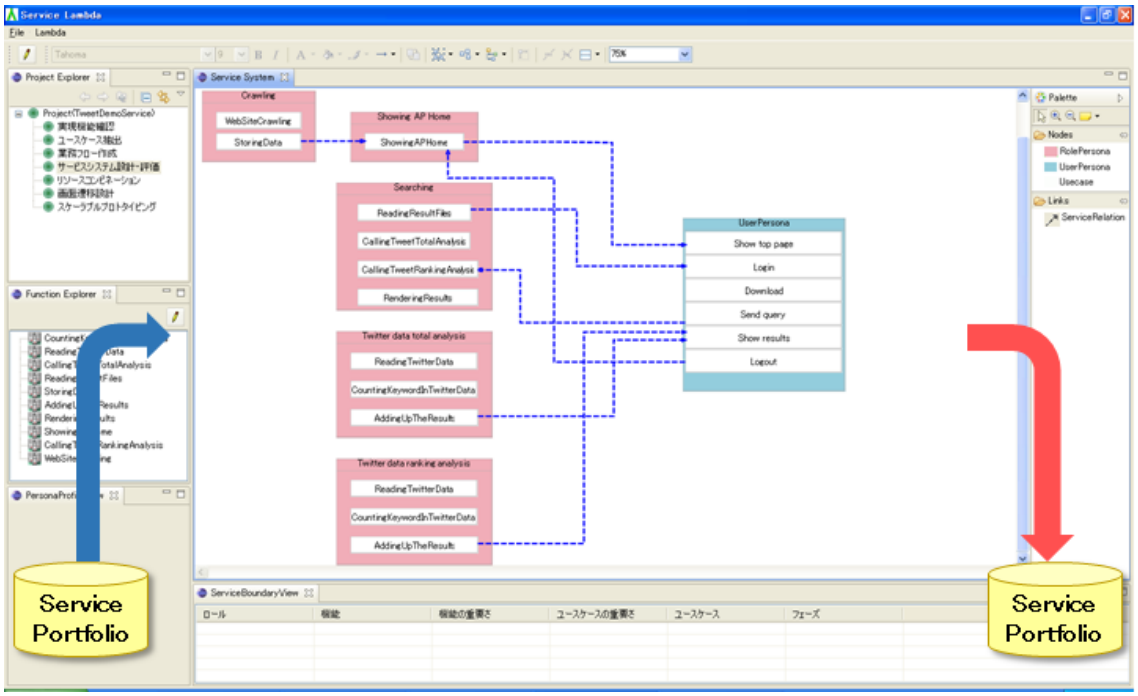

図 6-5 サービスシステム設計画面の例［Hosono2012b(改)］

6.2.1.3 リソース割り当て設計

リソース割り当て設計を行うツール *Resource Combination Design* は，ソフトウェアモジュールをハードウェアリソースに割り当てを行う．従来の IT システム開発では，ハードウェア製品やミドルウェア製品の選択範囲が限られていたため，システム開発者は，Web サービスの業務ロジックの設計・実装に集中できた．他方，クラウドコンピューティングの要素技術である仮想化技術は，サーバやストレージを仮想化し，論理的に扱う．そして，これらの仮想化サーバ／ストレージに設定可能なパラメータの増加は，リソース設計の自由度の増大を招く．そのため，システム開発者には，Web サービスの業務ロジックだけでなく，仮想サーバや仮想ストレージ等のリソースまで含め，包括的な設計を行うことが求められる．このリソース割り当て設計画面を以下に示す（図 6-6）．

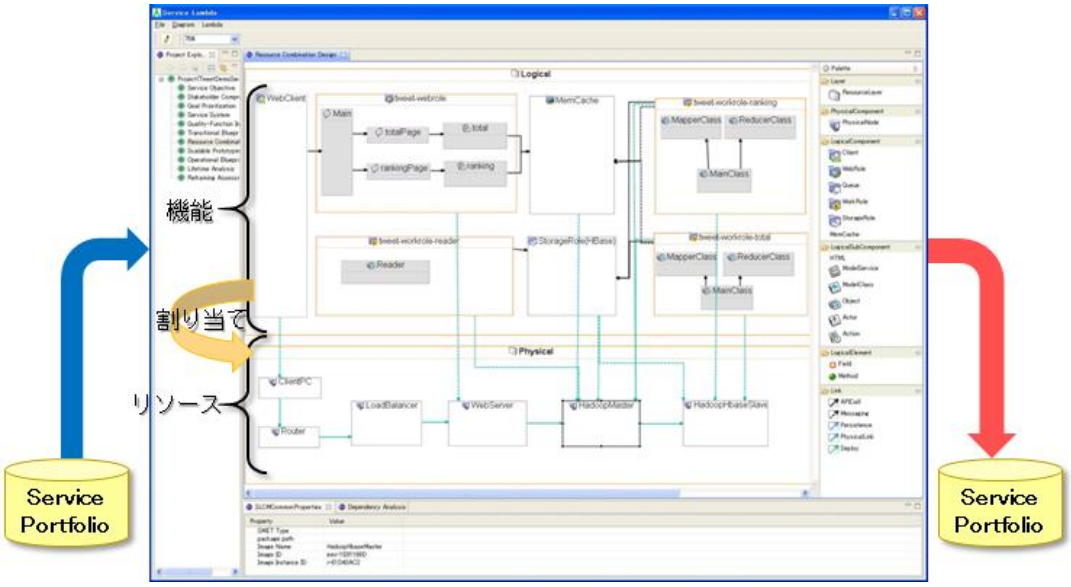

図 6-6 リソース割り当て設計画面の例［Hosono2012b(改)］

Resource Combination Design に，**モジュール分割設計機能**（*Extended DSM*）を実装した．各ステップでは，それぞれ性能，保守性，安全性，可用性，移行性等の非機能要件を合わせて考慮する必要がある．そこで，Design-for-X（DfX）の思考のもとで 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．マトリクスの交点の値は，0 または 1 の 2 値ではなく，複数の非機能要件を 0～1.0 の中間値で入力可能にする．機能，非機能要件の重み付けを行い，その総和が 1.0 になるように，交点の値を設定することで，多面から依存関係を同時に考慮したモジュール設計が可能となる．これらにより，ソフトウェア機能のモジュール化, 及び Web サービスの論理的･物理的な配置場所の設計が容易となる．

図 6-7 は，遺伝的アルゴリズムを応用して，このマトリクスのモジュール分割計算を自動化し，リソース割り当て設計の評価を行った例である．［Albers2009］が示した，導出の自動化を応用し，分割計算を行う．閾値によって多段に色付けされた矩形が複数の設計観点を考慮した結果，導出されたモジュールの設計解が示される．機能要件に比べ，非機能

要件の設計・実装は，従来，開発者の経験や主観に拠るところがあったが，本方法により個々の非機能要件も客観的な設計パラメータとして包括的に扱うことができる．

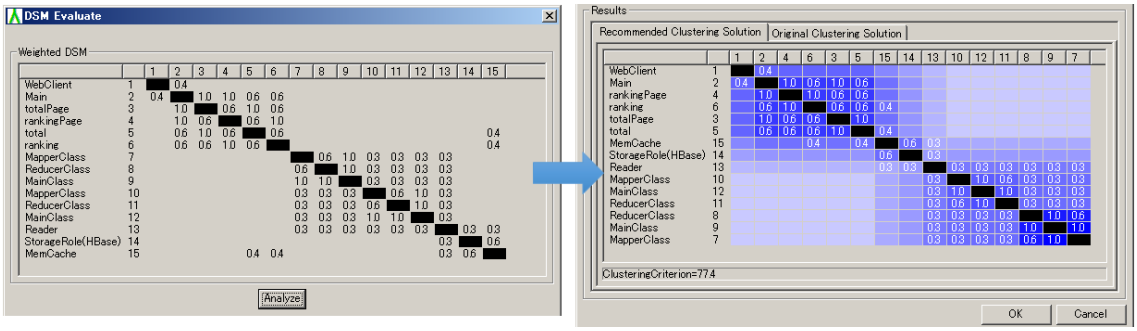

図 6-7 モジュール分割設計画面の例［Hosono2012b(改)］

6.2.1.4 プロトタイピングと機能要件・非機能要件の検証環境

機能検証（プロトタイピング）環境 *Application Prototyping* は，ソフトウェアモジュールのプロトタイプ実装とその機能検証を行う．クラウド環境では，大規模・並列に計算を行うことを前提に大規模ミドルウェアや大規模ファイルシステムを備える．そのためアプリケーションの機能検証を行う場合に，検証環境をシステム開発者の手元の PC 等，ローカル環境に構築することは困難であった．そこで，機能検証と非機能検証範囲を分離し，機能検証を行うためのスタブ機構である機能検証環境をクライアント側に用意する（図 6-8 右）．これにより，Web サービスのプロトタイプ開発を手元の PC で実行することが可能になる．遠隔にあるクラウド環境に開発した Web サービスを配置し検証する手間が減るため，検証効率を向上できる．

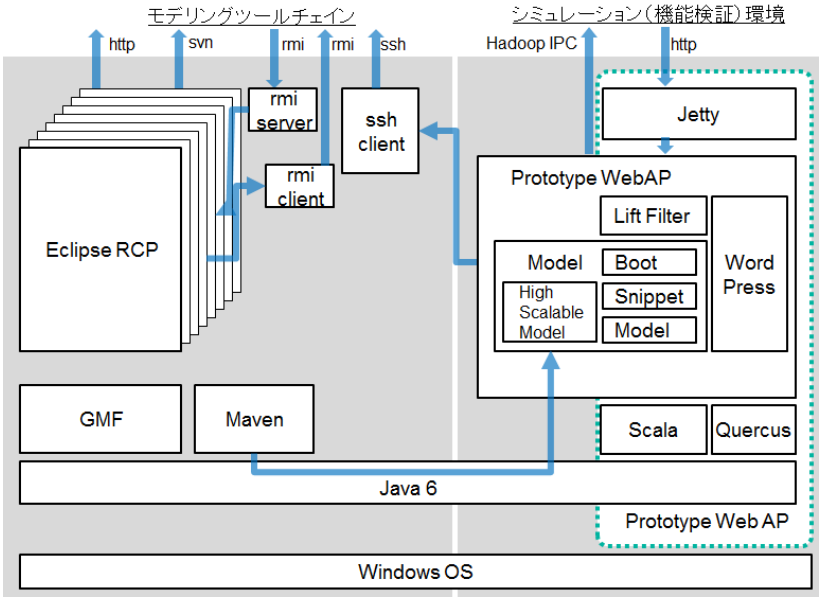

図 6-8 機能要件の実現検証環境（クライアント側）［Hosono2012b(改)］

次に，**非機能検証環境** *Cloud Execution Stack* の概要を述べる．

機能検証の後に，サーバ側の検証済みミドルウェアを含む仮想マシン群からなる非機能検証環境（*Cloud Execution Stack*）に Web サービスを配置し，実際のリソース環境でしか行えない，性能等の非機能要件の検証を行う．

図 6-9 に，機能要件の実現検証環境および非機能要件の検証環境を示す．図 6-9 左部は，クライアント側（図 6-8）の検証環境で処理が困難な並列分散処理をシミュレートする環境である．図 6-9 右部は，サービスライフサイクル情報を保持する機構で，機能要件の検証に関する情報を供する．図 6-10 に，非機能要件の実現検証環境を示す．この環境は，並列分散処理を行う機構で，性能など非機能要件の挙動を確認するものである．また，図 6-11 に，非機能要件の実行監視機構を示す．これは，図 6-10 の非機能要件の達成状態を監視するものである．

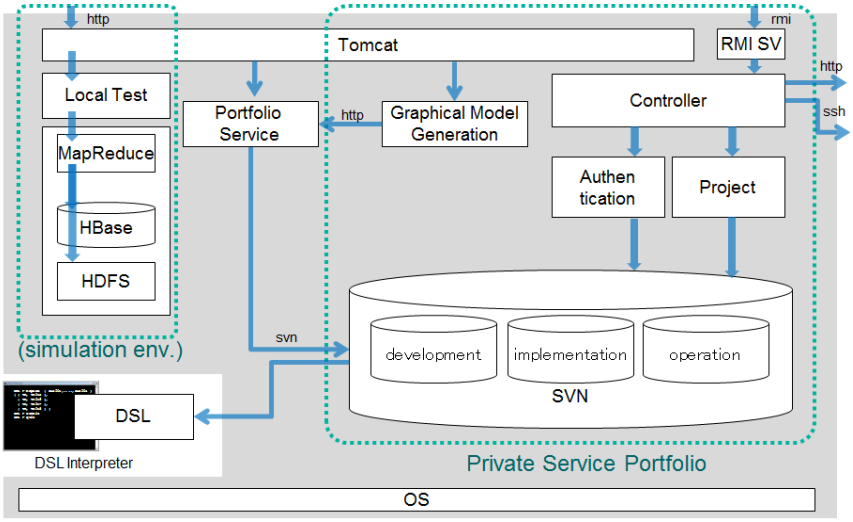

図 6-9 機能要件および非機能要件の実現検証環境［Hosono 2012b(改)］

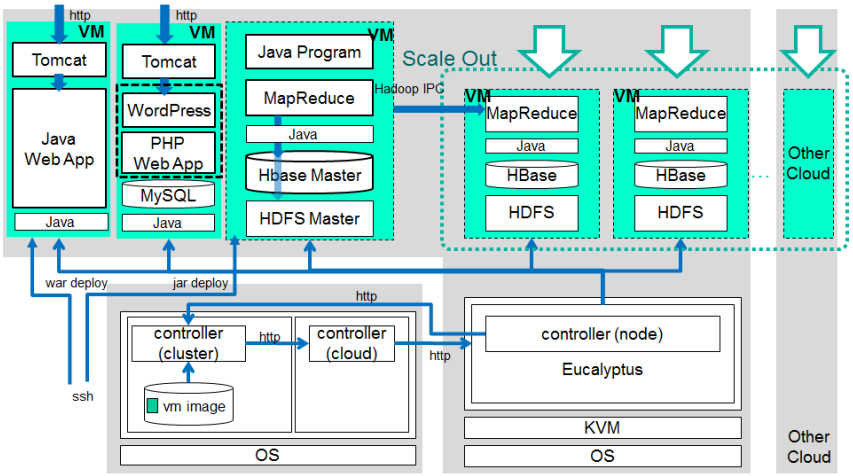

図 6-10 非機能要件の実現検証環境［Hosono 2012b(改)］

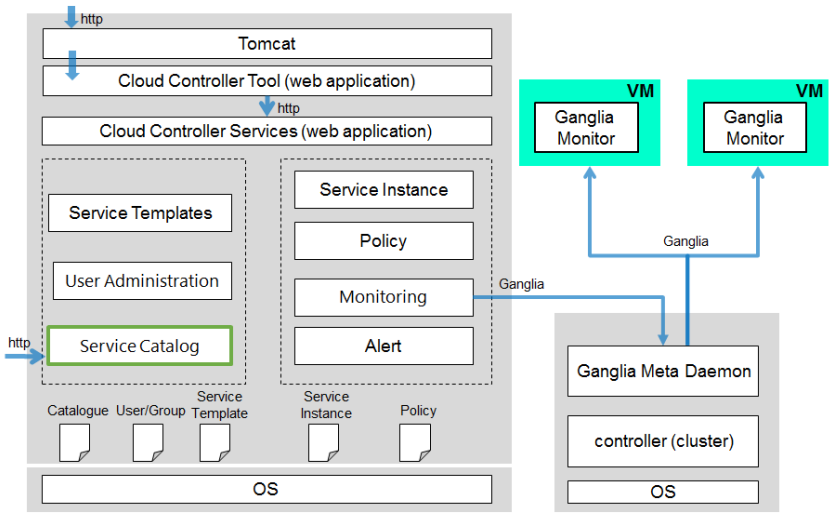

図 6-11 非機能要件の実行監視機構［Hosono 2012b(改)］

非機能要件の実行監視機構（*Cloud Controller*）は，**非機能要件の達成状況を**監視する，概要を述べる．Web サービスの非機能要件には，業務観点で性能や可用性についてトランザクション数や 24 時間の稼動時間等がある．これらの要件は，CPU やネットワーク，ハードディスクの書込み速度等，複数の計算機リソースの組み合わせで実現される．これらの業務レベルとリソース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．これを用いて，運用開始時に *Service Portfolio* に蓄積された非機能要件レベルから，リソースレベルの監視対象と監視（SLO: Service Level Objectives）に変換し，実行環境の *Cloud Execution Stack* を監視する．図 6-12 に監視設定の例を示す．管理者は，この *Cloud Controller* からの監視通知を受けて，運用中に非機能要件を満たしているかを確認できる．

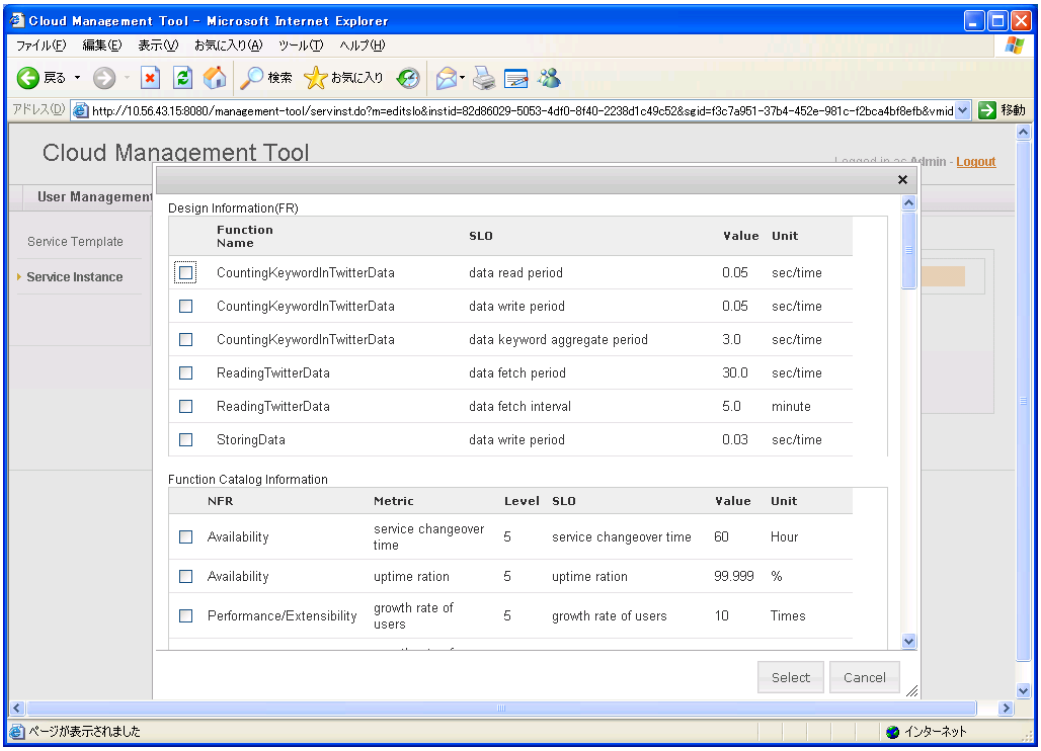

図 6-12 非機能要件の実行監視機構（ビューア）［Hosono 2012b(改)］

6.2.1.5 サービスポートフォリオ

サービスポートフォリオ（*Service Portfolio*）は，サービスライフサイクル情報を一元管理する．これまでに示したモデリングは，相互に関係があり，サービスポートフォリオ（*Service Portfolio*）においてモデリングデータを統合し，サービスモデルチェーンのデータを構築する（図 6-13）．*Service Portfolio* の主要構成は，リポジトリと，ライフサイクル知識の検索や組み替えなどを行う，DSL（Domain Specific Language）インタープリタ等のデータ処理機構である．ライフサイクル知識の処理機構は，各モデリングツールから生成された XML データ群からサービスモデルチェーンを構築する．更に，第 4 章で示したライフサイクルパタンを構築し，工数実績であるタスクチェーンとの統合を行い保管する．この処理機構は，ライフサイクルパタンの XML データ構造から，工程の前後方向への追跡や複数の設計工程を跨ったデータ連携を行う．この構造により，次工程のモデルとの整合性検証や設計データの後工程への引き継ぎが可能になる．

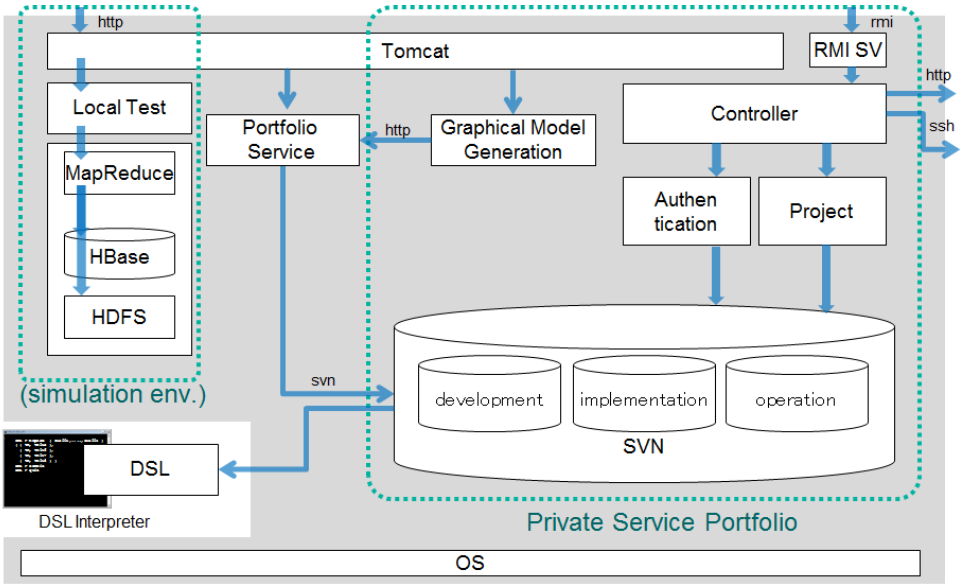

図 6-13 サービスポートフォリオの実装［Hosono 2012b(改)］

6.2.2 協働タスク

6.2.2.1 協働開発

効用の高いサービスを効率的に生産するには，第 3 章，第 4 章に示したモデルの構築と再利用に加えて，(a) 顧客と IT サービスプロバイダや，開発と運用部門，開発部門と企画・管理部門間などライフサイクル内のステークホルダ間の協働支援，(b) 過去のサービス開発と別のサービス開発の開発部門などライフサイクルを跨ったステークホルダ間の協働支援と，これらを定着させるためのプロセスが必要である．

そこで，次節より，顧客と提供者，および異なる役割を持った提供者内のステークホルダ間の協働作業を支援する方法を示し，全体を効用の高いサービスシステムのライフサイクルを管理する方法として体系化する．

6.2.2.2 進捗管理

ライフサイクルパタンの再利用は，全体の開発段階のために拡張できる．Web サービスを開発する間，実践者間の頻繁な相互作用は進展が曖昧な状態を任意の開発時点において評価する．アーンドバリュー分析[6]は，生産進捗状況を評価するための方法である．獲得した価値の分析を通じて，計画された値（PV）が評価日に与えられた予算と定義される．実際のコスト (AC) は，実際に使用されるコストとして与えられる．アーンドバリュー (EV) は，測定日の時点において決定される．PV，AC 及び EV の値は以下の式で与える．

```
PV = Σ (planned man-hours × a unit price)                    (1)
AC = Σ (actual man-hours × a unit price)                     (2)
EV = Σ (planned man-hours × a unit price × progress rate)    (3)
```

PV，AC と EV の値は，プロジェクトの開始日，推定終了日，反復時間数，サイクルを完了する予定日などのデータから計算する．これらのデータは，サービスポートフォリオに格納されたデータから，サービス開発の任意の時点でデータを取得し，EV 値を生成できる．PV，AC，と EV で，アセスメントおよび各反復開発の終了予定日の時点で生産進捗が視覚化できる（図 6-14）．このユースケースでは，生産パタンは，次のサイクルに対応する部分のチェーンは，生産期間の過去の実際の記録を提供するように，「予測終了日時」と「次の開発サイクルのために必要な工数」のための標準データを提供と工数を与える．

上記に示したように，モデルチェーンおよびタスクチェーンは，ライフサイクルコストを見積もる尺度になるため，サービス開発投資の意思決定を判断する際の指標になる．

[6] アーンドバリュー分析（earned value analysis）とは，作業の到達度を金銭などの価値に換算した出来高（earned value）で捉える分析手法．

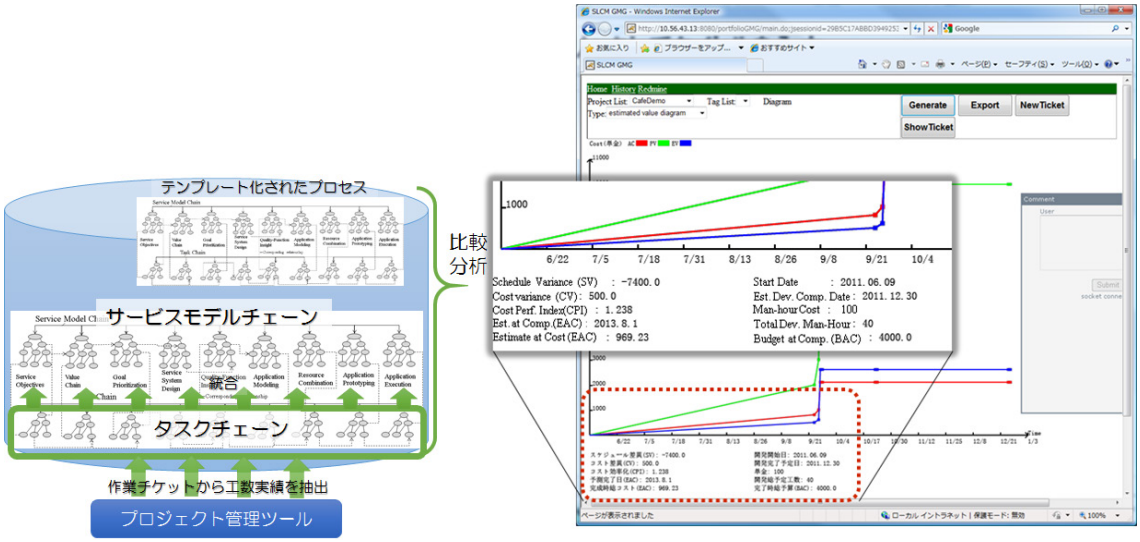

図 6-14 サービス開発進捗のアセスメント

6.2.2.3 タスク管理

本節では，提案したフレームワークと，その規律を現場で実践する方法を議論する．カーネルアルファを実践する際，その都度，アルファカードをテーブルの上に並べて議論し，進めるのは，小規模の開発・少人数の開発チームによる場合に限られる．そこで，小規模から大規模開発まで，本提案を実践できるようにするには，開発管理を行うプロジェクト管理システムに組み込んで実践するのが現実的である．ここでは，プロジェクト管理の方式として，チケット駆動開発（ticket-driven development; TiDD）に着目する．チケット駆動開発は，作業指示書のようなものとして，チケットを使う．アルファカードをチケットに見立てることにより，カーネルアルファを実践できる（図 6-15）．

ここでは，プロジェクト管理ソフトウェアの Redmine[7]を用いる．Redmine はオープンソースのプロジェクト管理ソフトウェアで，タスク管理，進捗管理，情報共有を行える．バグトラックシステム（BTS）としても活用される．このソフトウェアでは，チケットの発行により，開発者が次に行うタスクを指示できる．各チケットは，チケットの大分類を示すトラッカーに必ず紐付けられる．この仕組みで，チケットの状態を，どの役割の開発者にどのように遷移させるのか，ワークフローの定義を行うことができる．この仕組みを使い，次のように運用する．予め，アルファカード群をチケットとして，保持しておく．ここで，カーネルアルファの並びに対応した順序関係を保持しておく．開発作業はチケットが入力となり，開発文書やプログラムなどの成果物が出力となる．この成果物が確定した段階を，アルファカードの項目が達成された状態とする．履歴をコメントとして追記し，進捗状態はチケットに書き込まれる．チケットの集計機能によって，開発の状態を確認できる．このように扱うことで，アルファカードが，開発の作業指示書となるため，開発に関わるステークホルダの行動に規律を与えられる（図 6-15，図 6-16）．

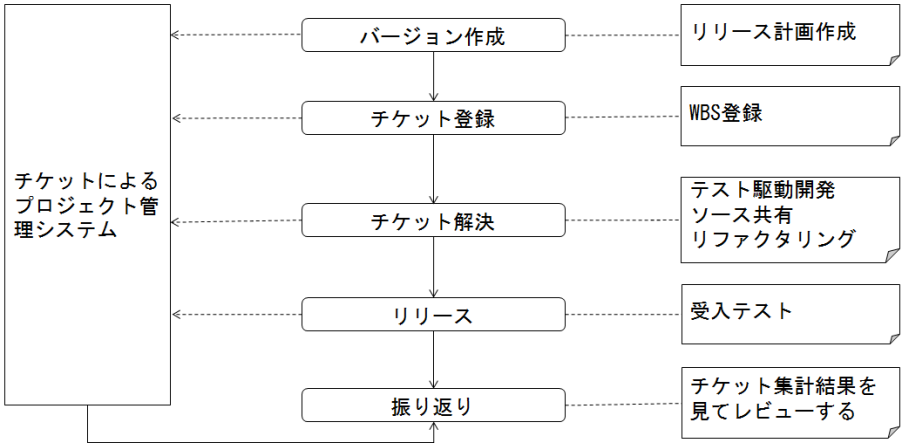

図 6-15 チケットによるタスク実行管理

[7] Redmine は，オープンソースのプロジェクト管理ソフトウェアで，プロジェクトのタスク管理，進捗管理，情報共有を行える．http://redmine.jp/overview

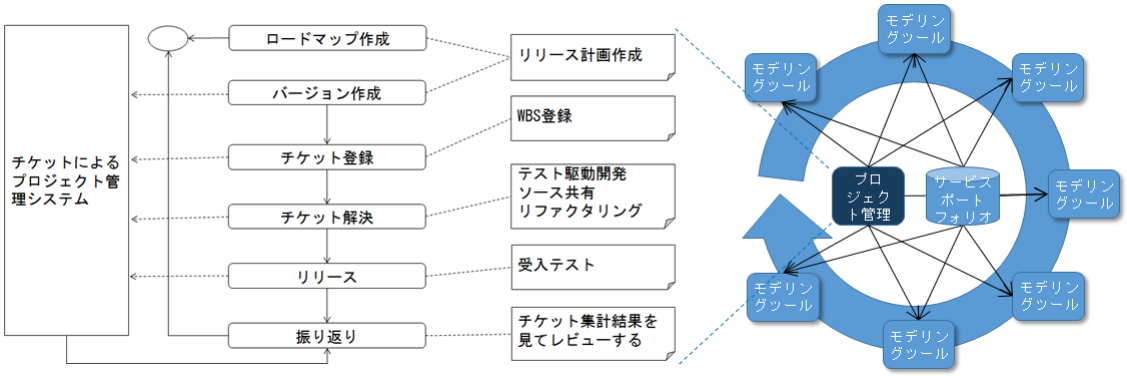

図 6-16 プロジェクト管理ツールとの統合

以上の例により，アルファカードのチケットへの置き換えにより，規律を実践できることが判った．このように，チケット開発では，個別のチケットの状態は明確になるが，全体の開発プロセスの中で，開発の前後関係を把握することが，困難となりがちである．ここで示したカーネルアルファは，個々のアルファが，個別の観点を示しており，そのアルファを構成する状態群の並びによって，全体の見通しを予め把握できた．

6.3 事例

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．

これまで述べた開発方法と，その支援ツールをデータセンタ上の Web サービス開発に適用した．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 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（図 6-17）．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，解析結果を短時間に応答する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．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 などのリソースの組み合わせ候補がリソースカタログに表示され，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，*Resource Combination Design* の *Extended DSM* に，各アプリケーションの各機能を配置し，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 *Extended DSM* に加味した．*Extended 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．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 *Application Prototyping* を用いてアプリケーションの各機能の実装・機能検証を行い，各仮想サーバに配備した．

運用にあたり，*Cloud Controller* 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対応づけされた．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，ライフサイクル工程のモデリング方法に対応したツール群を利用して，系統立てて設計～運用までを行えることを確認できた．特に，工程間で設計の見直しなど手戻りが発生せず，また，実装したソフトウェアモジュールを実行環境のクラウドへの配備・実行作業が容易であることを確認できた．

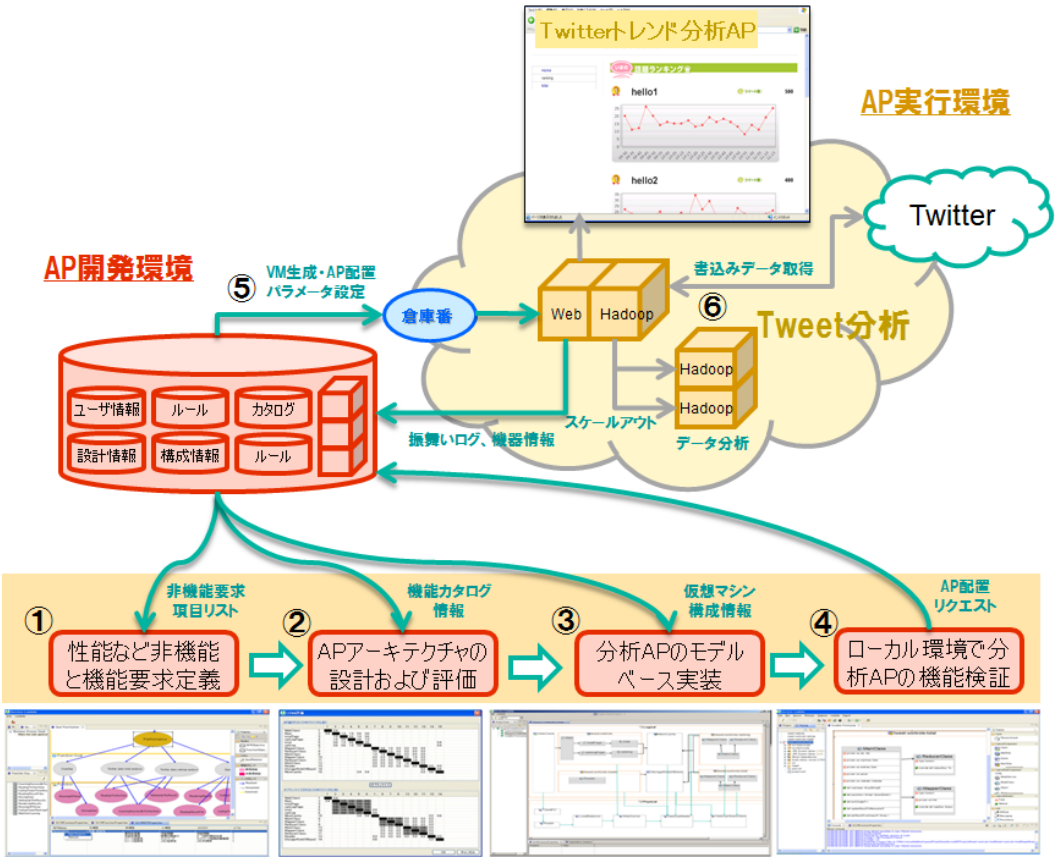

図 6-17 事例 1：トレンド分析サービス［Hosono 2012b(改)］

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 6-18）．この Web サービスは，Web ブラウザとスマートデバイス（携帯端末）の 2 種類のユーザインタフェースを持つ．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．本事例では，事例 1 で作成したライフサイクルパタンとモデリングされた情報を参照し，モデリングを行った．**要求–機能展開モデル設計画面**を用い，アプリケーションサーバを機能目標として構造化し，リクエストの受付，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，*Application Prototyping* を用いて，ローカル PC 環境でアプリケーションを実装し，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．この後，クラウドに配備し，*Cloud Controller* 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．

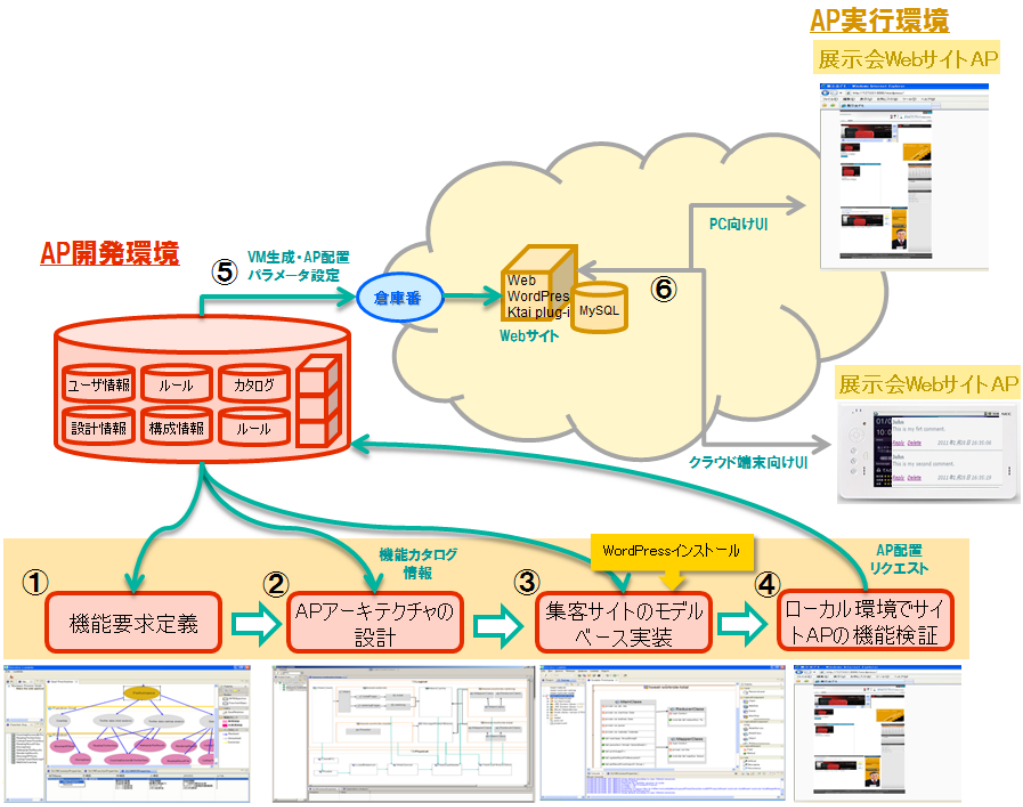

図 6-18 事例 2：リモートアクセスサービス［Hosono 2012b(改)］

以上の事例での開発において，文書名あるいはソフトウェアパッケージ名に含まれるキーワードで，その再利用性を区分する再利用戦略ポリシーを定義した．再利用戦略ポリシーとして，「共通」をファイル名に含む場合は Reuse を，また，プロジェクト固有リストに含まれる企業名を外し，Remanufacturing をラベル付けするルールを定義した．また，タイマーを表す再利用戦略ポリシーとして「有効期間が 2～9 か月目まであること」を示すルールを定義した．ライフサイクルパタンの元データとなる，開発仕様書や，運用手順書な

どの全ての開発文書と，各々の作成時間・コストなどの付帯情報に対し，再利用戦略ポリシーを適用した．以上の結果，プロジェクト固有の情報が除かれた成果物が再利用リポジトリ群に保管された．例えば，「共通仕様文書」「通信プログラム」ファイルは Reuse ラベルを付けて，元のデータにプロジェクトや製品固有情報を含んでいた「運用手順」「データベース接続プログラム」は Remanufacturing ラベルを付けて,「データディスク」「メモリ」情報は Recycling ラベルを付けてリポジトリに保管された．

次に，上記で構築したモデルチェーンのデータをパタンとして蓄積した．再利用に適したライフサイクルパタンの特定に向けて，性能に関する要件に適合したライフサイクルパタンの特定処理を次のように行った．ここでは，性能の設計変数としてメッセージ処理性能について着目した．まず，メッセージ処理性能に関する要件，開発の V 字型モデルで表現される提供プロセスを入力した．要件に関して，性能目標値は，単位時間当たりの要求メッセージの処理数が，0～80 件の範囲は性能が不十分で不可，80～100 件は性能がやや不十分で望ましく無い範囲，100～150 件は性能が適正な範囲，150～170 件は性能がやや過剰で望ましくない範囲，170 件以上は性能が過剰で不可とした．提供プロセスは，メッセージ処理スレッドプログラムの開発に関し，①機能設計，②詳細設計，③実装，④運用で構成した（V 字型モデルでは，①と④，②と③が対応）．次に，To-Be サービスシステム構造（クラウドを利用した Web サービスアプリケーションアーキテクチャ）のモデルを入力した．

次に，設計者の意図分布を示す，設計変数の設計意図と，実行者から見た意図分布を示す，実行意図と，目的設計変数名「メッセージ処理件数」を入力した．ここで設計変数の設計意図には，性能を重視して，①機能設計として，スレッド数 7 を中央値に，②詳細設計として，接続受付のバックログサイズに不定（無限）を設定した．また，実行意図には，他の連携システムと性能レベルを合わせるため，③実装として，接続受付のバックログサイズに 10 を中央値に，④運用として，スレッド数に 6 を中央値に設定した．ここで，①と④には，それぞれ分布曲線を与えた．

検索条件の合成処理では，設計変数の設計意図および，実行意図のそれぞれについて，目的設計変数の導出関数を用いて，目的設計変数を計算した．ここで，目的設計変数の導出関数は，

$f(x) = x \times 20\ (0 \leqq x \leqq 10)$，$f(x) = 200\ (x > 10)$

（x はスレッド数，f(x) は処理性能（メッセージ処理数／単位時間））

を与えた．ここで，①と④に着目すると，①はメッセージ処理数：140 件／単位時間となる．また，④はメッセージ処理数：120 件／単位時間となる．

検索条件の合成処理では，目的設計変数名「メッセージ処理件数」の導出元となった「単位時間当たりのメッセージ処理性能」を用いて，ライフサイクルパタンを保管するリポジトリから，目的設計変数との共通範囲と合致するものを得た．ここでの合成方法は，2 者の

分布の足し合わせにより行った．

このように，目的設計変数が 80～170 の範囲に絞り込まれたことから，目的設計変数の導出関数を用いて，4～8（＜8.5）スレッドの範囲が導出された．これをスレッドの仕様の設計解とし，ライフサイクルパタンを保管するリポジトリから，前述の範囲値を持つライフサイクルパタンの各ファイルの構造を検索し，スレッドに該当する箇所の値が合致した機能仕様書ファイル，詳細設計書ファイル，ソフトウェアプログラムおよび運用マニュアルなどを抽出した．抽出されたライフサイクルパタンの中から，適合度の高いものから順に開発者に推薦表示した．これをサービス開発の雛形データとして，これに含まれるファイル群を用い，カスタマイズして開発を行った．

以上により，別のステークホルダに管理された情報をモデル化した．これらをライフサイクルパタンとして，再利用できることを確認できた．

6.4 適用結果

6.4.1 ライフサイクル知識の構築

事例では，次の効果が顕著であった．(1) クラウドの実リソースに基づく要件定義，(2) アプリケーションの機能モジュールおよび実行リソースの割り当て設計により，クラウド環境のリソースをモデル化したことで，アプリケーション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の見直しなど，膨らみがちになる保守・改善作業を局所化できた．

本事例では，(3) 実行リソース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 のリソースのみで検証した．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，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．

クラウドコンピューティングは，実装とテストのコストを削減し得る．しかしながら，コスト効果は，アプリケーションの設計に直接関係するため，十分にその効果を得られない．このフレームワークは，そのようなリスクと軽減できる．事例 1 で示したように，このフレームワークは，ユーザがアプリケーションをクラウド上の仮想マシンにアップロードと配備を行う作業を多く代替するためである．また，非機能要件を満たすようにリソースの構成を試行錯誤で決める作業も代替するためである．同様に，事例 2 では実行基盤に依存した，オープンソースソフトウェアの組み合わせや外部ライブラリの使用およびパラメータの構成設計などの作業を代替する．このように，これらの事例は，ユーザが低コストでかつ短期間に柔軟なアーキテクチャをクラウド上に構築できることを示した．

6.4.2 ライフサイクル知識の活用による協働生産

クラウドの利用によって，ソフトウェア機能モジュールのリソース割り当てが新たな設計対象となったが，従来の開発スキームには明確な役割定義が無いことが分かった．そこで，新たに**リソース計画者**（Resource Planner）を導入し，上流設計～運用までの役割と担当範囲を図 6-19 のように定義した．開発と運用工程のギャップを埋めるライフサイクルパタンの活用を，具体的な役割として実践するためにも，この役割を導入することの必要性を確認できた．

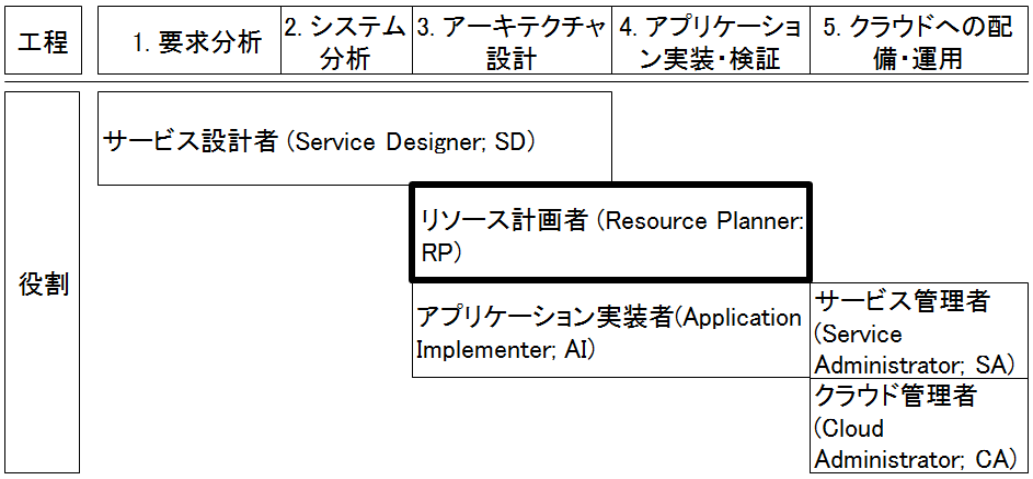

図 6-19 設計～運用までの役割定義

6.4.3 生産性の改善

前節の役割定義に沿って，要求分析～検証部分に適用した．結果，従来の方法では，要件定義 4 週，実装期間 5 週，検証評価 3 週程度要したが，本方法では，全体で 4 週間強の期間で開発できるとの評価を得た．

サービス開発者の評価の結果，モデリング時のデータ入力が最小化された点，および，モデリングツールにおいて，モデルチェーンの関係から，モデルデータ間の整合性のチェックを行えた点が大きいことが分かった．

次節に，各モデリング方法が，生産性の向上に効果のあった点について述べる．

6.5 サービスライフサイクル管理の効果

6.5.1 サービスの生産性向上に関する考察

事例評価から，IT サービスシステムのライフサイクル管理方法によって，クラウドに対応した Web サービスを従来に比べ半分に近い期間で迅速に提供できた．クラウド上の検証済みの実行環境である（*Cloud Execution Stack*）を用いる点を差し引いても，従来の開発方法に比べて 3 割程度という著しい開発期間短縮を達成した．本方法では，工程間の情報連携・整合性確認まで行えたため，期間短縮できたと言える．この効果の要因を以下にまとめる．

6.5.1.1 機能-リソースカタログを利用した要件定義ツール

Goal Prioritization は，業務のゴール，非機能要件，機能要件を構造化する．この構造は，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（実行リソースを含む）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．この構造化により，アジャイル型で短期に繰り返し開発を進める場合，非機能要件を見直す際に，関連する機能群を同定できるため，開発に影響する範囲を特定し易くする効果も得られた．

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，対応する実行リソースの組み合わせと，その諸元値（範囲）を表示するカタログを表示した．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，後の工程で実行リソース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た．

6.5.1.2 モデル間の自動データ連携

クライアント側のモデリングツール群は，公理的設計の考え方でモデル間を連結した．このアプローチは，顧客ドメイン，機能ドメイン，物理ドメインおよびプロセスドメインから成る 4 つのドメイン間の属性値の転写により設計プロセスを簡素化した．各モデリングツールは，前の開発工程で確定した開発データを自動的に引き継ぐ．要件定義工程では，要件に基づいて，**機能-リソースカタログ**から候補の機能を示す．カタログデータは，過去にクラウドに配備された Web サービスの振舞い実績の範囲に基づいている．*Resource Combination Design* で定義された機能の十分性は，目標の優先順位付けに定義された要件と照合することで検証する．*Application Use Case/Data Modelling/System Flow* のエディタで，同じダイアグラムを改めて記述し直さなくて済むように，*Service Portfolio* を介して，ダイアグラム情報の同期を行う．*Resource Combination Design* で定義されたリソース構成は，実装中の Web サービスを収容するためのクラウド上に，開発工程を先回りして仮想

マシンの構築に使用する. *Application Prototyping*の初期画面は, 前工程における *Resource Combination Design* で定義されているアプリケーションのロジックを示す.

以上に示した通り，この自動データ共有の仕組みは，各工程でデータ入力を削減し，その開発効率を向上できた.

6.5.1.3 サービスポートフォリオからの逐次データ取得

Service Portfolio は，設計・開発データや文書を保管し，任意の時点でのアクセスを可能にした.

多くの場合，開発の初期段階での文書は通常，後工程の管理者またはオペレータと共有されていない．これは，サービス品質の低下につながる開発と生産の間のコミュニケーションギャップの原因となっている．しかし，この機構は，全てのサービス開発従事者の間で開発の知識を共有するための完全な環境を提供するので，このリスクを減らせた.

また，要件定義書の記述向けの標準的グラフィカル表記は殆ど無いため，開発者は，自然言語で記述してきた．しかし，本手法は，格納されたデータを再構造化し，標準化されたフォーマットでシステムアーキテクチャモデルを形成し，広く使用されている Web ブラウザ上にダイアグラムとして描画する．このダイアグラムは，協働作業に関わるアーキテクトや実装者のような様々な役割，および異なるレイヤの情報を持つ担当者間の共通言語となる．この共通言語によりステークホルダ間の共通認識が得られ易いため，IT サービスシステムの管理者が顧客と交渉し，合意形成する際にも有効である.

6.5.1.4 Web サービスのプロトタイプ実行

Application Prototyping は，開発者の開発環境で Web サービスのプロトタイピングを行えるツールである．これは，大規模なデータを扱う Web サービスの検証も可能にしている．スケール性を要する大規模なデータ処理に必要な環境と同等の環境であるため，開発者の開発環境で Web サービスを実行できる．クラウドに接続する際，データセンタから課金されるが，この手法により，開発者が実施段階で連続的にクラウドに Web サービスを展開するため，このような状況を避けることができた.

Web サービス実装者は，これまで，Web サービスのプロトタイプをクラウド環境に配備した後に，Web サービスの検証を行ってきた．本ツールは，[Buyya2009] らが行ってきた従来のシミュレーションと異なり，このシミュレーションツールは，Web サービス実装者の手元にある PC で Web サービスのシミュレーション実行を行い, 機能検証を可能にした.

6.5.1.5 設計～運用まで定型化された設計・運用プロセス

Goal Prioritization を用いた要件定義では，機能カタログから機能候補を提示した．カタログデータから Web サービスの実装・運用工程で生じ得るリソース制約を，予め設計工程で考慮することが可能になり，下流工程での設計の見直し機会を抑えられた.

Service System Design を用いた機能モジュール設計では，Web サービスを構成する機

能モジュールと要件定義との対応を確認しながら設計した．これにより，*Goal Prioritization* で定義され *Service Portfolio* に蓄積された要件定義と突き合わせを自動的に行うことで，要件定義の対応漏れを検出し，手戻り工数を削減できた．

Resource Combination Design を用いたリソース割り当てでは，機能独立性の評価を行いながら，Web サービスの仮想マシンへの配置構成を設計した．ツールは，この設計情報を元にクラウド上に検証済みミドルウェアを含む仮想マシンを自動で構築した．結果，開発者は Web サービスのロジック実装のみに集中でき，構築・検証作業の効率が向上した．

Application Prototyping での機能検証は，*Resource Combination Design* で設計され *Service Portfolio* に保管された機能モジュールを流用して，Web サービスのロジックの雛型を生成し，実装を効率化した．

また，*Cloud Controller/Cloud Execution Stack* は，Web サービスを実行するミドルウェアのパラメータは，要件定義工程で蓄積したレベル値を再利用して自動設定を可能とし，運用設計を容易にした．

このように，サービスモデルチェーンによって，設計作業を効率化し，設計内容を高品質化できた．

6.5.1.6 フレームワークのアーキテクチャの利点

Application Prototyping のプラグインアーキテクチャは，Java のバイトコードと互換のある軽量言語と，そのアプリケーションフレームワークに対応している．この機構によって，Web サービスの多くの種類の実行が可能になる．もう一方のプラグイン可能なアーキテクチャは，サービスポートフォリオ側に存在する．この機構により，高度なデータ分析を行う Web サービスの実行も可能にした．このように，これらのプラグイン可能なアーキテクチャは，Web サービス開発の柔軟性とカスタマイズ性の点で有利であった．

Cloud Execution Stack は，非機能要件の達成を検証するステージング環境を提供している．クラウドに対応した Web サービスは，仮想マシンのコントローラは，仮想マシン，大規模データ処理ミドルウェアやデータベース，アプリケーションフレームワークとビジネスコンポーネントで実行が可能となる．この Web サービスシステムの下位レイヤの設計は，Web サービスの検証まで行う必要があるため，システムインテグレータの手間のかかる作業となる．しかし，*Cloud Execution Stack* の検証プラットフォーム環境をシステムインテグレータが利用することで，検証作業を不要とした．このため，システムインテグレータは，下位レイヤのプラットフォームを設計し検証するタスクを削減でき，Web サービスのロジックの開発に集中できた．このように，実行環境をパッケージングしたことで，モジュール性が高くなり，開発者はリソースの設計や割り当て作業に工数を割かずに済むため，有利である．

本フレームワークは，別の見方をすると，DevOps の考え方を具体化するプラットフォームとも言える．DevOps は，従来役割が分断されていた開発～運用工程間の作業を緊密に

連携させるもので，事業変化にも対応し易くする［Haight2010,2011］．具体的には，協働開発，配備・環境設定・監視・メンテナンスなどの作業で得た情報を開発～運用の担当者にフィードバックし，環境変化により迅速に対応し易くする．これにより，プロジェクトの進行を早め，業務品質を向上できる［Kenefic2011］［Gartner2010］．本提案のフレームワークは，これらの人手のプラクティスを技術支援するものである．本フレームワークを開発～運用の現場に導入することで，システム管理者やオペレータの，各工程担当者に対する支援作業を軽減できる．結果として，開発～運用を通じて，各役割がそれぞれ本来の役割に専念できるようになった．

6.5.2 サービスのエコシステムに関する考察

本プラットフォームは，サービスとしてのアプリケーションプラットフォームの可能性を持つ．データセンタ事業者は APaaS[8]サービスとしてのフレームワークを提供でき，システムインテグレータは，開発環境として使用できる．APaaS プラットフォーム上でビジネスアプリケーションのプロバイダとの共同開発を通じて，Web サービスを提供する［Gartner 2009］．

従来の Web サービス開発では，システムインテグレータが開発し，開発者のローカル環境で Web サービスの動作を検証し，Web サービスを再構築し，顧客のサイトでそれを確認してきた．しかし，本研究で可能としたビジネスモデルは，クラウド上で同じ環境を共有することにより，システムインテグレータと Web サービスプロバイダとの間での配備・検証作業を減らせる．以上によって，3 つのステークホルダ間の連続的な関係を構築できる（図 6-20）．

[8] APaaS（Application Platform as a Service）とは，マルチテナント型のリソース共有技術を利用して，各ユーザ（テナント）がカスタムアプリケーションをクラウドで実行できるようにした環境である．

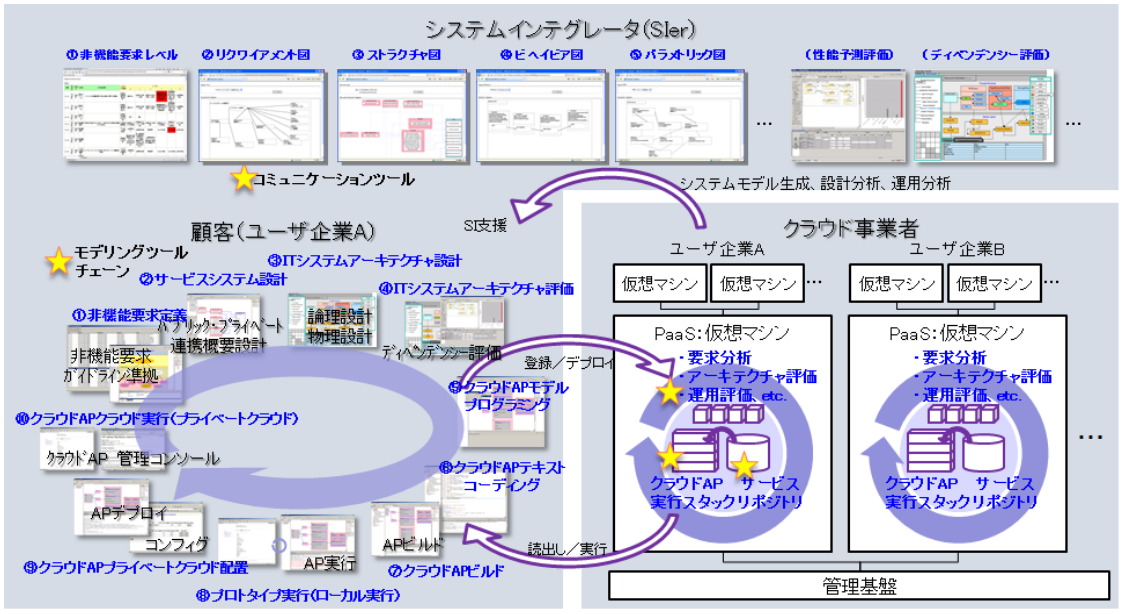

図 6-20 ビジネスエコシステム［Hosono2010(改)］

以上に示したフレームワークを用いることで，Web サービスの種類と開発手順を絞り込める．本フレームワークは，固有の要件が多い大規模開発よりも，比較的サービスのパタンが明確な中小規模のサービス案件（Small and Medium Business, SMB）に有効である．この規模のサービス開発では，小規模な開発チームで実践されてきたアジャイル開発のプラットフォームとして本フレームワークを適用し易い．本フレームワークをその開発プラットフォームとして利用することで，アジャイル開発を支援できる．

6.6 おわりに

本章では，第 5 章において提案した，プロセスを実際のサービス事例に対して提供することにより，提案するサービスライフサイクル管理方法論の有効性について検証を行った．

第 2 節では，サービスライフサイクル管理方法論の実践について述べた．まず，サービス LCM フレームワークを示した．このフレームワークは，モデリングツールと，サービスリポジトリで構成した．モデリングツールは，ゴールモデリング，システムモデリング，リソース割り当て，プロトタイピング環境，およびサービス監視を備えた．サービスポートフォリオは，パブリック・サービスポートフォリオとプライベート・サービスポートフォリオで構成した．また，協働支援方法を，現場で行われているプロジェクト管理の仕組みに統合し，実践方法を示した．

第 3 節では，検証対象となる Web サービスの概要を述べた．

第 4 節では，サービス LCM フレームワークの適用結果について述べた．

第 5 節では，サービスシステムの生産の観点で考察する．生産性向上の観点と，ビジネスモデルの観点で議論した．

以上に示した通り，本章では，これまで示した方法を利用して，ライフサイクル方法論の実践と検証を示した．

第 3 章，第 4 章，第 5 章で示した方法の考え方は，クラウド上で実行すること，それに伴い，IT リソースを扱うことを除けば，IT サービスに適用は限定されない．IT サービスに限定したことで，ライフサイクルを一巡する管理方法の確認が容易であったとも言える．

今後は，他の領域のサービスに対して，同様の方法論を適用し，その効果を確かめることが必要である．

第7章 結論

目次

7.1 結論

本研究では，クラウドコンピューティングの浸透に伴う，モノ売りからサービス事業への変化に対応したサービスライフサイクル管理方法を示した．

ライフサイクル工程のモデリング方法では，ステークホルダ間の共通言語となるモデルの定義方法と，ライフサイクルを通じて情報に間断のないモデリングの連鎖を示した．これにより，サービス企画〜運用までに工程に関わる，異なる役割を持ったステークホルダがそれぞれ扱う情報を形式化し，相互に参照し，ステークホルダ間の共通言語となるモデルを構築した．

ライフサイクル知識の資産化方法では，生産プロセスを資産化する方法と，パタン化およびカスタマイズインタフェースの定義方法を示した．これにより，サービス企画〜運用までの開発プロセスに関わる，異なる役割のステークホルダが扱う情報を一括して保持し，サービス設計者がこの情報をより多くのサービス開発案件に適用し易くした．

ライフサイクル知識の構築・活用プロセスでは，ステークホルダの役割を起点に活用方法と，知識構築・利用プラクティスの開発プロセスへの適合のさせ方を示した．

以上の方法によって，IT サービスプロバイダは，サービス開発の上流〜下流工程の知識を集約して保持し，有効に活用できるようになった．この知識を活用し，顧客と継続的な関係を広い工程に渡り，維持し易くなった．これにより，従来の売り切り型から，サービス中心のストック型の事業への移行を促進できる．

以上により，サービスのライフサイクル知識の構築と再利用は，サービスのライフサイクルを通じて最大化され，IT サービスプロバイダは，効果的に彼らの能力を発揮することができる．これにより，ステークホルダが協働し，合理的にかつ効率的にサービスを開発し管理できる．

7.2 今後の展望

本研究では，IT サービスシステムのライフサイクルの中で，サービスを正しく作り，提供することに重点をおいた．提供者に対し，従来モノで提供した機能をサービスの機能で設計し直せること，また，開発知識の集約・利用により生産性を向上させることに，価値を訴求した．顧客に対し，サービス提供の速さと品質の高さに，価値を訴求した．本研究で示した方法は，今後，次のように発展させられる．

(1) サービスのモデリング方法

機能要件，および非機能要件の検証は，クラウドに配備して確認する準備が容易ではないため，プロトタイピングとその実行環境による実践方法を示した．これは，Web サービスを対象としたため，ソフトウェアの実装によるプロトタイプ検証が容易であったとも言える．

一般のサービスでは，必ずしもプロトタイプの作成と実行は容易ではない．そこで，概念検証を行うために，設計した機能の振舞いを実装・配備後ではなく，設計段階で机上で評価・検証すべく，モデリング方法にサービス構造の定性関係を加えて，サービスシステムの挙動シミュレーションを行うことが考えられる．

(2) ライフサイクル知識の資産化方法

資産化した設計・運用プロセスには，ドメイン固有のビジネスルールや，法律などの情報が暗黙的に埋め込まれている．同じドメイン内で，再利用する際に，これらは問題とならないが，ドメイン間での再利用性を高めるには，これらを考慮する必要がある．そのため，ビジネスルールや法律を資産化した設計・運用プロセスのデータおよび背景となる情報から抽出し，構造化する必要がある．これには，自然言語処理やパタンマッチングによる，ビジネスルールや法律に関する情報を設計・運用プロセスの情報から抽出し，分離する方法が考えられる．

また, 製品指向 (product-oriented), 利用指向 (use-oriented), 成果指向 (result-oriented) のサービス特性に基づく，ライフサイクルパタンの類型化も必要である．

(3) ライフサイクル知識の構築・活用プロセス

サービスシステムの構築過程において，関わるステークホルダは，その役割の達成に至るまでに，幾つかの項目を達成する必要がある．ある時点において行う必要がある項目にのみ関心があたっているため，ステークホルダ間の協働作業において意思疎通が行き届かない原因となる．そこで，役割に関連付けられる関心事に着目し，その関心に合った細粒度の情報の提示や見落としているタスクの提示方法が必要である．これには，アクターネットワークを構成するステークホルダの役割と，各役割の持つ関心事をモデル化するアプ

ローチが考えられる.

また，更なる開発の効率化に向けて，製品指向（product-oriented），利用指向（use-oriented），成果指向（result-oriented）のサービス特性に基づく，協働作業の類型化や，サービスの廃棄あるいは継続的改善の判断方法が必要である.

更に，ライフサイクル研究としての発展の方向性に，以下を挙げる.

⑷ 利用価値の設計と管理

今後のサービスライフサイクル研究では，サービスのライフサイクルの中で，提供したサービスをユーザが使うことによって生まれる価値と，その管理方法を具体化する必要がある．提供者に対し，顧客の利用価値を設計・管理すること，顧客に対し，継続利用による価値の訴求する方法が必要である.

謝辞

謝辞

本論文は，筆者が首都大学東京大学院システムデザイン研究科 下村研究室 博士課程後期の期間において行った研究成果を纏めたものです．その間，多くの方々にご指導，ご協力頂いた．ここに深く御礼申し上げます.

はじめに，本論文の主査および筆者の指導教員である

首都大学東京大学院システムデザイン研究科
ヒューマンメカトロニクスシステム学域 教授 下村芳樹先生

に心より感謝の意を申し上げます．下村先生には，筆者が大学に在籍する以前より，サービスの設計に関わる研究活動に多くの助言を頂き，設計について深く考える機会を与えて下さいました．また，産学官を連携したサービス研究の機会を与えて下さり，自身の視野と研究開発の幅を広げることができました．今後も，よろしくご指導下さいますよう，お願い申し上げます.

また，本論文の副査をお引き受けくださいました

東京大学大学院工学系研究科 教授 梅田靖先生
首都大学東京大学院システムデザイン研究科 教授 高間康史先生
首都大学東京大学院システムデザイン研究科 教授 久保田直行先生

に謹んで感謝の意を申し上げます．審査過程において，先生方からは，様々な観点からご指導を頂きました．自身の研究の意味づけを見つめ直す機会を得ることができました.

筆者の勤務する企業の研究所においては，サービスライフサイクル管理をテーマに研究活動を行い多くの示唆を得ました．この場をお借りして，関係者の皆様に御礼申し上げます.

サービス工学研究会の参加団体の皆様には，研究会を通じて，サービスの現場の課題や取り組みを共有して頂き，多くの示唆を頂きました．この場をお借りして御礼申し上げます.

2015 年（平成 27 年）3 月

細野 繁

参考文献

英語文献

[Albers 2009] Albers, A., Sedchaicharn, K., Sauter, C., Burger, W., 2009, 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, 241-252.

[Ambler 2012] Ambler, S.W., Lines, M., 2012, Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise, IBM Press.

[Arai 1998] Arai, E., Akasaka, H., Wakamatsu, H., Shirase, K., 2000, Description Model of Designers Intention in CAD System and Application for Redesign Process, JSME International Journal C, 43/1: 177-182.

[Arai 2004] Arai, T. and Shimomura, Y., 2004, Proposal of Service CAD System, A tool for Service/Product Engineering, Annals of the CIRP, 397-400.

[Arai 2005] Arai, T. and Shimomura, Y, 2005, Service CAD System. Evaluation and Quantification, Annals of the CIRP, 463-466.

[Armbrust 2009] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M., 2009, 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

[Arsanjani 2007] Arsanjani, A., Zhang, L.J., Ellis, M., Allam, A., Channabasavaiah, K., 2007, Design an SOA solution using a reference architecture - Improve your development process using the SOA solution stack, developerWorks, IBM.

[Baines 2007] 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., 2007,

State-of-the-art in product service systems, IMechE, Part B: Journal of Engineering Manufacture, 221: 1543-1552.

[Browning 2001] Browning, T.R., 2001, Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions, IEEE Transactions on Engineering Management, 48/3: 292-306.

[Buyya 2009] Buyya, R., Ranjan, R. and Calheiros, R.N., 2009, Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities, Paper presented at the 2009 High Performance Computing and Simulation Conf., 1-11.

[Clements 2001] Clements, P., and Northrop, L., 2001, Software Product Lines: Practices and Patterns, Addison-Wesley Professional; 3rd edition.

[Eppinger 1997] Eppinger, S.D., 1997, A planning method for integration of large-scale engineering systems, Paper presented at the 11th Int'l Conf. on Engineering Design, 199-204.

[Eppinger 2012] Eppinger, S.D., Browning, T.R., 2012, Design Structure Matrix Methods and Applications, MIT Press.

[Finch 1995] Finch W.W., Ward A.C., 1995, Generalized Set Propagation Operations over Relations of more than Three Variables, Artificial Intelligence for Engineering Design. Analysis and Manufacturing, 9/3: 231-242.

[Finch 1997] Finch, W.W., Ward, A.C., 1997, A Set-based System for Eliminating Infeasible Designs in Engineering Problems Dominated by Uncertainty, 1997 ASME Design Engineering Technical Conferences, 179-187.

[Fisk 2000] Fisk, R.P. Grove, S.J. and John, J., 2000, Interactive service marketing, Boston, Houghton Mifflin.

[Gartner 2009] Gartner Research, 2009, APaaS: A Step to a 'Killer App' for Cloud Computing?, ID Number: G00168152.

[Gartner 2010] Gartner Research, 2010, DevOps: Born in the Cloud and Coming to the Enterprise, ID Number: G00208163.

[Hara 2006] Hara, T., Arai, T. and Shimomura, Y., 2006, A Concept of Service Engineering: A Modeling Method and A Tool for Service Design, Proc. of IEEE International Conference on Service Systems and Service Management, 13-17.

[Hara 2009] Hara, T., Arai, T. and Shimomura, Y., 2009, A CAD system for service innovation: integrated representation of function, service activity, and product behavior, Journal of Engineering Design, Special issue on PSS, Vol. 20, No. 4, 367-388.

[Haight 2010] Haight, C., 2010, DevOps: Born in the Cloud and Coming to the Enterprise, Gartner Research, ID Number: G00208163.

[Haight 2011] Haight, C., 2011, DevOps: The Changing Nature of System Administration, Gartner Research, ID Number: G00211703.

[Hosono 2014] Hosono, S. and Shimomura, Y., 2014, Asset-based Service Production under Uncertainty. Procedia CIRP, Vol. 16, 247-252.

[Hosono 2013a] Hosono, S. and Shimomura, Y., 2013, Towards Facilitating Servitization in IT Service Systems, Paper presented at the 1st International Conference on Serviceology, 26-129.

[Hosono 2013b] Hosono, S. and Shimomura, Y., 2013, Towards Establishing Production Patterns to Manage Service Co-creation, Paper presented at the 5th CIRP Industrial Product-Service Systems Conference, 95-106.

[Hosono 2012a] Hosono, S. and Shimomura, Y., 2012, Towards Establishing Mass Customization Methods for Cloud-compliant Services, Presented at the 4th CIRP Industrial Product-Service Systems Conference, 447-452.

[Hosono 2012b] S. Hosono, 2012, A DevOps framework to shorten delivery time for cloud applications. International Journal of Computational Science and

Engineering, Vol. 7, No. 4, 329-344.

[Hosono 2010] Hosono, S., Huang, H., Hara, T., Shimomura, Y. and Arai, T., 2010, A Lifetime Supporting Framework for Cloud Applications, Presented at the 3rd IEEE International Conference on Cloud Computing, CD-ROM.

[IEEE830 1998] IEEE Standard 830-1998, IEEE Recommended Practice for Software Requirements Specifications

[IIBA 2009] International Institute of Business Analysis: Business Analysis Body of Knowledge Version 2.0, http://www.theiiba.org/

[Inoue 2009] Inoue, M., Ishikawa, H., 2009, Set-Based Design Method Reflecting the Different Designers' Intensions, the 19th CIRP Design Conference, 404-411.

[Iqbal 2007] Iqbal, M., et al., 2007, ITIL Lifecycle Publication Suite (Version 3): Continual Service Improvement, Service Operation, Service Strategy, Service Transition and Service Design, Stationery Office.

[Jacobson 2013] Jacobson, I., Ng, P.W., McMahon, P., Spence, I. and Lidman, S., 2013, The Essence of Software Engineering: Applying the SEMAT Kernel, Addison-Wesley.

[Kenefick 2011] Kenefick, S., 2011, DevOps: Extending Agile to Increase Synergy Between Development and Operation, Gartner Research, ID Number: G00214696.

[Kimita 2009] Kimita, K., Yoshimitu, Y., Shimomura, Y. and Arai, T., 2009, A Customer Value Model for Sustainable Service Design, CIRP Journal of Manufacturing Science & Technology 1/4: 254-261.

[Maglio 2006] Maglio, P., Srinvasan, S., Kreulen, J. and Spohrer, J., 2006, Service Systems, Service Scientists, SSME, and Innovation Journal of Communications of the ACM, 49/7: 81-85.

[Mont 2002] Mont, O., 2002, Functional Thinking - The role of functional sales and product service systems for a function-based society, IIIEE, Lund University, Sweden. http://www.naturvardsverket.se/Documents/publikationer/620-5233-0.pdf?pid=2892

[Mont 2004] Mont, O., 2004, Product-service systems: Panacea or myth?, Ph.D. thesis, IIIEE, Lund University, Sweden.

[Norman 2001] Norman, R., 2001, Reframing Business - When the Map Changes the Landscape, John Wiley & Sons, Ltd.

[OMG 2013] The Object Management Group (OMG), 2013, Documents Associated with Essence - Kernel and Language for Software Engineering Methods 1.0 - Beta 1, http://www.omg.org/spec/Essence/1.0/Beta1/PDF.

[Paul 1996] Paul, G., Beitz, W., 1996, Engineering Design – A systematic Approach, 2nd Edition, London Springer.

[Paul 2005] Pohl, K., Bockle, G., Linden, F., 2005, Software Product Line Engineering: Foundations, Principles and Techniques, Springer.

[Pine 1993] Pine, J., 1993, Mass Customizing Products and Services, Planning Review, 21/4: 6-13.

[Pine 1992] Pine, J., 1992, Mass Customization: The New Frontier in Business Competition, Harvard Business School Press Books.

[Pine 1993] Pine, J., Victor, B. and Boyton, A., 1993, Making Mass Customization Work, Harvard Business Review, 71/5: 108-119.

[Ramaswamy 1996] Ramaswamy, R., 1996, Design and Management of Service Processes, Addison-Wesley, Reading.

[Scott 2012] Scott W. Amber, 2012, Mark Lines, Disciplined Agile Delivery, IBM Press.

[Shostack 1982] Shostack, G.L., 1982, How to Design a Service, European Journal of Marketing, 161: 49-63

[Simon 1996] Simon, H.A, 1996, The Sciences of the Artificial, The MIT Press.

[Steward 1981] Steward, D.V., 1981, The design structure system: a method for managing the design of complex systems, IEEE Transactions on Engineering Management, 28/3: 71-84.

[Spohrer 2006] Spohrer, J. and Maglio, P., 2006, The Emergence of Service Science: Toward systematic service innovations to accelerate co-creation of value. (http://www.almaden.ibm.com/asr/SSME/jspm.pdf)

[Suh 1990] Suh, N.P., 1990, The Principles of Design, Oxford University Press, New York.

[Suh 2001] Suh, N.P., 2001, Axiomatic Design, Oxford University Press, Oxford.

[SysML 2008] OMG: SysML ver.1.1, http://www.sysmlforum.com/docs/specs/OMGSysML-v1.1-08-11-01.pdf

[Umeda 2012] Umeda, Y., Takata, S., Kimura, F., Tomiyama, T., Sutherland, J.W., Kara, S., Herrmann, C., Duflou, J.R., 2012, Toward integrated product and process life cycle planning–An environmental perspective, CIRP Annals - Manufacturing Technology, 61/2: 681-702.

[UML 2005] Object Management Group: Unified Modeling Language (UML) version 2.0, http://www.omg.org/spec/UML/

[Tayleor 1994] Taylor, S.A. and Baker, T.L., 1994, An assessment of the relationship between service quality and customer satisfaction in the formation for consumer' purchase intentions, Journal of retailing and consumer services, 70/2: 163-178.

[Vargo 2004] Vargo, S.L. and Lusch, R.F., 2004, Evolving to a New Dominant Logic for

Marketing, 68/1: 1-17.

[W3C 2004] http://www.w3.org/TR/ws-arch/#whatis

[Whitfield 2002] Whitfield, R.I, Smith, J.S, Duffy, 2002, Identifying Component Module", In Proceedings of the 7th International Conference on Artificial Intelligence in Design, 15-17.

[Zhang 2007] Zhang, LJ., Zhang J. and Cai, H., 2007, Service Computing, Springer-Verlag,

日本語文献

[IPA 2003] 情報処理振興事業協会: EVM 活用型プロジェクト・マネジメント導入ガイドライン,
http://www.meti.go.jp/policy/it_policy/tyoutatu/evm-guideline.pdf

[IPA 2010] 非機能要件グレード,
http://www.ipa.go.jp/sec/softwareengineering/reports/20100416.html

[IPA 2011] 情報処理推進機構, 2011, 非機能要件とアーキテクチャ分析 WG 報告書, ver1.0.

[荒井 1998] 荒井栄司，赤阪秀和，若松栄史，白瀬敬一，1998，CAD における設計意図モデルと修正設計への適用, 日本機械学会論文集（C 編）, 64/62: 384-389.

[青山 2008] 青山和浩，2008，サービスシステムと生産システム，第一回システム創生学学術講演会，48-51.

[石川 2010] 石川晴雄, 2010, 多目的最適化設計 - セットベース設計手法による多目的満足化 -, コロナ社.

[大富 2005] 大富浩一，2005，製品開発における上流設計の重要性とその方法，東芝レビュー，60/1: 30-35.

[久野, 細野 2009] 久野綾子，細野繁，2009，IT システムサービス開発におけるステークホルダ抽出・管理方法の提案. FIT(電子情報通信学会・情報処理学会) 情報科学技術フォーラム講演論文集 8(1), 327-332.

[経済産業省 2008] 経済産業省，2008，IT サービス継続ガイドライン.

[経済産業省 2010] 経済産業省，2010，技術戦略マップ（サービス工学分野）.

[古宮 1998] 古宮誠一，加藤潤三，永田守男，大西淳，佐伯元司，山本修一郎，蓬莱尚幸，1998，インタビューによる要求抽出作業を誘導するシステムの実現方法，情報処理学会研究報告ソフトウェア工学（SE），100（1998-SE-121）：99-106.

[清水 2010] 清水吉男，2010，[入門+実践]要求を仕様化する技術・表現する技術 -仕様が書けていますか？，技術評論社.

[下村 2005] 下村芳樹，原辰徳，渡辺健太郎，坂尾知彦，新井民夫，冨山哲男，2005，サービス工学の提案 - 第1報，サービス工学のためのサービスのモデル化技法-，日本機械学会論文集 C 編, 71/702: 315-322.

[総務省 2010] 総務省，スマート・クラウド研究会報告書，http://www.soumu.go.jp/main_content/000066036.pdf

[妻屋 2010] 妻屋彰，矢是秀明，森永英二，竹内一博，内田済，荒井栄司，田浦俊春，2010，設計情報・設計意図統合型 CAD の研究（部品間の影響関係記述による概念設計支援），日本機械学会論文集（C 編），76/771: 18-25.

[内藤 2009] 内藤耕，2009，サービス工学入門：経験と勘に頼るサービスから科学的・工学的手法へ，東京大学出版会.

[人見 1990] 人見勝人，1990，生産システム工学，共立出版.

[細野 2013a] 細野繁，赤坂文弥，木見田康治，下村芳樹，2013，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，日本機械学会論文集 C 編，79/808: 504-519.

[細野 2013b] 細野繁，下村芳樹，2013，クラウドに対応した Web サービスの生産方法 -Service Lambda for Cloud Computing による具体化と検証-，日本経営工学会論文誌, 63/4: 245-257.

[山路 2006] 山路幹夫，武内真弓，石坂浩之，2006，ITIL による運用管理の実際，ソフトリサーチセンター.

[吉川 1979] 吉川弘之，1979，一般設計学序説：一般設計学のための公理的方法：，精密工学会誌，45/536: 906-912.

研究業績

学術論文（査読あり）

[1] **S. Hosono** and Y. Shimomura: Asset-based Service Production under Uncertainty. Procedia CIRP, Vol. 16, pp.247-252, 2014.

[2] **細野繁**, 赤坂文弥, 木見田康治, 下村芳樹: 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. 日本機械学会論文集 C 編, Vol. 79, No. 808, pp. 504-519, 2013.

[3] **細野繁**, 下村芳樹: クラウドに対応した Web サービスの生産方法 -Service Lambda for Cloud Computing による具体化と検証-. 日本経営工学会論文誌, Vol. 63, No. 4, pp. 245-257, 2013.

[4] **S. Hosono**: A DevOps framework to shorten delivery time for cloud applications. International Journal of Computational Science and Engineering, Vol. 7, No. 4, pp. 329-344, 2012.

[5] 木見田康治, 赤坂文弥, 太田卓見, **細野繁**, 下村芳樹: サービスモジュール化設計のための依存性分析. 精密工学会誌, Vol.78, No.5, pp. 395-400, 2012.

国際会議（査読あり）

[1] C. Sathawornwichit and **S. Hosono**: Consistent Testing Framework for Continuous Service Development. In Proceedings of the 2014 IEEE Asia-Pacific Service Computing Conference (APSCC2014), CD-ROM, IEEE, Fuzhou, China, 2014.

[2] E. Numata and **S. Hosono**: Disciplines for Collaborations in IT Service Development. In Proceedings of the 2nd International Conference on Serviceology (ICServe2014), pp.131-134, The Society for Serviceology, Kanagawa, Japan, 2014.

[3] **S. Hosono** and Y. Shimomura: Asset Development and Reuse Strategy in Cloud-compliant Service Development. In Proceedings of the 15th International Conference on Precision Engineering (ICPE 2014), pp.311-316, the Japan Society of Precision Engineering, Ishikawa, Japan, 2014.

[4] **S. Hosono** and Y. Shimomura: Towards Facilitating Servitization in IT Service Systems. In Proceedings of the 1st International Conference on Serviceology (ICServe2013), pp. 126-129, the Society for Serviceology, Tokyo, Japan, 2013.

[5] **S. Hosono** and Y. Shimomura: Towards Establishing Production Patterns to Manage Service Co-creation. In Proceedings of the 5th CIRP Industrial Product-Service Systems Conference (IPSS 2013), pp. 95-106, CIRP, Bochum, Germany, 2013.

[6] C. Sathawornwichit and **S. Hosono**: Consistency Reflection for Automatic Update of Testing Environment. In Proceedings of the 2012 IEEE Asia-Pacific Service Computing Conference (APSCC2012), pp. 335-340, IEEE, Guilin, China, 2012.

[7] **S. Hosono** and Y. Shimomura: Towards Establishing Mass Customization Methods for Cloud-compliant Services. In Proceedings of the 4th CIRP Industrial Product-Service Systems Conference (IPSS 2012), pp. 447-452, CIRP, Tokyo, Japan, 2012.

[8] **S. Hosono** and Y. Shimomura: Application Lifecycle Kit for Mass Customization on PaaS Platforms. In Proceedings of the 8th IEEE World Congress on Services (SERVICES 2012), pp. 397-398, IEEE, Honolulu, Hawaii, U.S.A., 2012.

[9] **S. Hosono**, J. He, X. Liu, L. Li, H. Huang and Shuichi Yoshino: Fast Development

Platforms and Methods for Cloud Applications. In Proceedings of the 2011 IEEE Asia-Pacific Service Computing Conference (APSCC 2011), pp. 94-101, Jeju, South Korea, 2011.

[10] K. Kimita, **S. Hosono** and Y. Shimomura: A Service Structural Analysis based on Functional Dependency. In Proceedings of the 2011 International Design Engineering Technical Conference (IDETC2011), pp. 617-626, The American Society for Mechanical Engineering (ASME), Washington DC, U.S.A., 2011.

[11] T. Ota, K. Kimita, **S. Hosono** and Y. Shimomura: An Efficient Conflict Detecting Method for Service Design. In Proceedings of the 2011 International Design Engineering Technical Conference (IDETC2011), The American Society for Mechanical Engineering (ASME), pp. 627-634, Washington DC, U.S.A., 2011.

[12] **S. Hosono**, M. Min, K. Kimita, F. Akasaka, Y. Shimomura, T. Hara and T. Arai: Architecture Design and Assessment with Design Matrix. In Proceedings of the 7th IEEE World Congress on Service (SERVICES 2011), IEEE, pp. 83-84, Washington DC, U.S.A., 2011.

[13] **S. Hosono**, K. Kimita, F. Akasaka, T. Hara, Y. Shimomura and T. Arai: Toward Establishing Design Methods for Cloud-based Systems. In Proceedings of the 3rd CIRP Industrial Product-Service Systems Conference (IPSS 2011), CIRP, pp. 195-200, Braunschweig, Germany, 2011.

[14] **S. Hosono**, H. Huang, T. Hara, Y. Shimomura and T. Arai: A Lifetime Supporting Framework for Cloud Applications. In Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD 2010), IEEE, pp. 362-369, Miami, U.S.A., 2010.

[15] K. Kimita, F. Akasaka, **S. Hosono** and Y. Shimomura: Design Method for Life Cycle Oriented Product-Service Systems Development. In Proceedings of the 43rd CIRP Conference on Manufacturing Systems (MS 2010), CIRP, pp. 281-288, Vienna, Austria, 2010.

[16] F. Akasaka, **S. Hosono**, M. Nakajima, K. Kimita and Y. Shimomura: Service Engineering: Requirement Analysis for the Improvement of Product-Service Systems.

In Proceeding of the 11th International Design Conference, the Design Society, pp. 117-126, Dubrovnik, Croatia, 2010.

[17] F. Akasaka, **S. Hosono**, K. Kimita, M. Nakajima and Y. Shimomura: Requirement Analysis for Strategic Improvement of a B2B Service. In Proceedings of the 2nd CIRP Industrial Product-Service Systems Conference (IPSS 2010), CIRP, pp. 117-124, Linkoeping, Sweden, 2010.

[18] K. Kimita, F. Akasaka, **S. Hosono** and Y. Shimomura: Design Method for Concurrent PSS Development. In Proceedings of the 2nd CIRP Industrial Product-Service Systems Conference (IPSS 2010), CIRP, pp. 283-290, Linkoeping, Sweden, 2010.

[19] **S. Hosono**, T. Hara, Y. Shimomura and T. Arai: Prioritizing Service Functions with Non-Functional Requirements. In Proceedings of the 2nd CIRP Industrial Product-Service Systems Conference (IPSS 2010), CIRP, pp. 133-140, Linkoeping, Sweden, 2010.

[20] **S. Hosono**, A. Kuno, M. Hasegawa, T. Hara, Y. Shimomura and T. Arai: A Framework of Co-creating Business Values for IT Services. In Proceedings of the 2009 IEEE International Conference on Cloud Computing (CLOUD-II 2009), IEEE, pp. 167-174,Bangalore, India, 2009.

[21] **S. Hosono**, M. Hasegawa, T. Hara, Y. Shimomura and T. Arai: A Methodology of Persona-centric Service Design. In Proceedings of the 19th CIRP Design Conference (Design 2009), CIRP, pp. 541-546, Cranfield, UK, 2009.

国内会議（口頭発表）

[1] 木見田康治，森下佳樹，**細野繁**，沼田絵梨子，榊啓，下村芳樹: サービス可用性評価のための定性シミュレーションモデル，デザインシンポジウム 2014, pp. 472-477, 2014.

[2] 沼田絵梨子, **細野繁**: サービス共創プロセスにおける協働作業を促進する規律，デザインシンポジウム 2014, pp.515-519, 2014.

[3] **細野繁**，下村芳樹: アセットの構築と活用によるサービスの協働生産．日本機械学会第 24 回設計工学・システム部門講演会講演論文集, CD-ROM, 2014.

[4] 森下佳樹，斎藤純平，木見田康治，**細野繁**，沼田絵梨子，榊啓，下村芳樹: サービス可用性評価のための定性シミュレーション, 2014 年度精密工学会秋季学術講演会, pp. 653-654, 2014.

[5] 森下佳樹，平川貴文，栗田雄介，木見田康治，**細野繁**，下村芳樹: サービス改善のための定性シミュレーションモデルに関する研究, 2014年度精密工学会春季大会学術講演会第21回「精密工学会学生会員卒業研究発表講演会」論文集, pp. 87-88, 2014.

[6] 平川貴文，栗田雄介，木見田康治，**細野繁**，下村芳樹: 定性シミュレーションに基づくサービスのモデル化手法，2014 年度精密工学会春季大会学術講演会講演論文集，pp. 409-410, 2014.

[7] 斎藤純平，栗田雄介，木見田康治，**細野繁**，下村芳樹: 定性シミュレーションを用いたサービスの故障要因解析手法，2014 年度精密工学会春季大会学術講演会講演論文集，pp. 423-424, 2014.

[8] **細野繁**，下村芳樹: サービス開発におけるアセット構築と再利用戦略．日本機械学会第 23 回設計工学・システム部門講演会講演論文集, No. 13-22, 2013.

[9] 平川貴文，栗田雄介，**細野繁**，木見田康治，下村芳樹: 定性シミュレーションを用いた高信頼サービスの設計手法, 2013 年度精密工学会秋季大会学術講演会講演論文集, pp. 709-710, 2013.

[10] 平川貴文，栗田雄介，太田航介，**細野繁**，木見田康治，下村芳樹: サービス品質安定化のための定性サービスモデル，2013 年度精密工学会春季大会学術講演会講演論文集，pp.

1005-1006, 2013.

[11] **細野繁**, 下村芳樹: サービス設計知識の構造化と活用方法. 日本機械学会第 22 回設計工学・システム部門講演会講演論文集, No. 12-17, pp. 717-726, 2012.

[12] **細野繁**, 下村芳樹: IT サービスのマス・カスタマイゼーションに向けたモデル・チェーン構築の提案, 2012 年度精密工学会春季大会学術講演会講演論文集, pp. 731-732, 2012.

[13] **細野繁**, 赤坂文弥, 木見田康治, 下村芳樹: クラウド環境におけるサービスライフサイクル管理. 日本機械学会第 21 回設計工学・システム部門講演会講演論文集, No. 11-23, pp. 436-441, 2011.

[14] 木見田康治, **細野繁**, 下村芳樹: 機能の依存関係に基づくサービス構造分析, 2011 年度精密工学会春季大会学術講演会講演論文集, pp. 593-594, 2011.

[15] 木見田康治, 赤坂文弥, 太田卓見, **細野繁**, 下村芳樹: 公理的設計を用いたサービス構成要素間の依存性分析. Design シンポジウム 2010 講演論文集,産業技術大学院大学, 東京, CD-ROM, 2010.

[16] **細野繁**, 原辰徳, 下村芳樹, 新井民夫: ビジネス価値の共創に向けた設計支援 LCM フレームワーク. 日本機械学会第 20 回設計工学・システム部門講演会講演論文集, No. 10-27, pp. 314-319, 2010.

[17] 木見田康治, **細野繁**, 下村芳樹: 構成要素間の依存性を考慮したサービスモジュール設計. 日本機械学会第 20 回設計工学・システム部門講演会講演論文集, No. 10-27, pp. 496-501, 2010.

[18] **細野繁**: 設計～運用情報の統合によるサービスライフサイクル管理と、APaaS 基盤への適用. 電子情報通信学会技術研究報告. 人工知能と知識処理, 110(172), pp. 49-52, 2010.

[19] 久野綾子, **細野繁**: IT システムサービス開発におけるステークホルダ抽出・管理方法の提案. FIT(電子情報通信学会・情報処理学会) 情報科学技術フォーラム講演論文集 8(1), pp. 327-332, 2009.

研究会・ワークショップ

[1] **細野繁**, 下村芳樹: 開発アセットの構築と計画的な利用を促進する規律, 情報処理学会ソフトウェア工学研究会 ウィンターワークショップ 2014 イン大洗, pp. 85-86, 大洗ホテル, 茨城, 2014.

[2] **細野繁**, 下村芳樹: マス・カスタマイズ開発に向けたモデルチェーンの構築と利用, 情報処理学会ソフトウェア工学研究会ウィンターワークショップ 2013 イン那須, pp. 69-70, ラフォーレ那須, 栃木, 2013.

[3] **細野繁**, 黄河, 原辰徳, 下村芳樹, 新井民夫: クラウドに対応したアプリケーションの開発・実行フレームワーク, 情報処理学会ソフトウェア工学研究会ウィンターワークショップ 2011 イン修善寺, pp. 61-62, ラフォーレ修善寺, 静岡, 2011.

[4] **細野繁**: サービスの機能設計・品質設計方法の提案, 情報処理学会ソフトウェア工学研究会ウィンターワークショップ 2010 イン倉敷, pp. 61-62, 倉敷市芸文館／倉敷アイビースクエア, 岡山, 2010.

寄稿・招待講演

[1] **細野繁**: まえがき「転のものづくり」, 日本機械学会誌 6 月号, (第 117 巻, 第 1147 号), pp.363, 2014.

[2] **細野繁**: クラウドを利用した情報サービスの構築・運用, 第 17 回 ISS スクウェア水平ワークショップ, 中央大学, 2010.

[3] **S. Hosono**: Applying Service Engineering to E-Learning Services, サービス生産性協議会 (SPRING) サービス工学セミナー, 東京大学, 2008.

受賞

[1] 日本機械学会, 設計工学・システム部門, 部門貢献賞, 2014.

特許

[1] Patent No.8924918 （Dec. 30, 2014）An Evaluation Apparatus, an Evaluation Method and an Evaluation Program (Oct. 15, 2014)
特許第 5464305 号（2014 年 1 月 31 日登録），国際 WO2013/108334（2013 年 7 月 25 日公開），特願 2013-537995（2012 年 12 月 12 日出願）「評価装置，評価方法，および，評価プログラム」

[2] Patent No. 8752012（June 10, 2014）, Application No. 13/881,223, Process Evaluation Device, Program and Method.
特許第 5370618 号（2013 年 9 月 27 日登録），国際 WO2013/042617（2013 年 3 月 23 日公開），特願 2013-517502（2012 年 9 月 7 日出願）「工程評価装置，プログラム，および，方法」

[3] 特許第 5681279 号（2015 年），国際 PCT/JP2012/069932（2012 年 7 月 31 日公開），特願 2013-517501（2013 年 4 月 17 日出願）「試験装置，システム，プログラム，及び，方法」

[4] 特許第 5651895 号（2014 年 11 月 28 日登録），特開 2011-221782A（2011 年 11 月 4 日公開），特願 2010-90105（2010 年 4 月 9 日出願）「公理的サービス設計方法，装置，プログラム」（共同出願）

[5] 特許第 5412905 号（2013 年 11 月 22 日登録），特開 2009-66684（2010 年 9 月 30 日公開），特願 2009-66684 号（2009 年 3 月 18 日出願）「ライフサイクル情報処理装置，方法，プログラム」

[6] 特許第 5648239 号（2014 年 11 月 21 日登録），特開 2011-221781（2011 年 11 月 4 日公開），特願 2010-90097（2010 年 4 月 9 日出願）「サービス再編成評価装置，方法，プログラム」（共同出願）

[7] 特許第 5435210 号（2013 年 12 月 20 日登録），特開 2010-256949（2010 年 11 月 11 日公開），特願 2009-102669 号（2009 年 4 月 21 日出願）「ステークホルダ抽出・管理装置，方法，プログラム」（筆頭発明者：久野綾子）

[8] 特許第 5386861 号（2013 年 10 月 18 日登録），特開 2009-301305（2009 年 12 月 24 日公開，特願 2008-154801 号（2008 年 6 月 13 日出願）「プロセス管理装置」

[9] 国際 PCT/JP2012/008200（2013 年 7 月 18 日出願），特願 2012-002392（2012 年 1 月 10 日出願）「サービスレベル管理装置，プログラム，及び，方法」

[10] 国際 PCT/JP2011/051533（2012 年 12 月 6 日公開，2012 年 8 月 16 日出願），特願 2010-024453（2010 年 2 月 5 日出願）「Web サービス構築管理方法，装置，プログラム」

[11] 特開 2010-231497（2010 年 10 月 14 日公開），特願 2009-078308（2009 年 3 月 27 日出願）「サービスシステム設計・管理支援方法，装置及びプログラム」（共同出願）

[12] 特願 2014-209635（2014 年 10 月 14 日出願）「プロダクトライン構築装置，プロダクトライン構築方法，プロダクトライン構築プログラム」（筆頭発明者：沼田絵梨子）

[13] 特願 2014-209634（2014 年 10 月 14 日出願）「アセット構築装置，アセット構築方法，アセット構築プログラム」（筆頭発明者：沼田絵梨子）

[14] 特願 2014-087444（2014 年 4 月 21 日出願）「タスク特定装置，タスク特定方法及びタスク特定プログラム」（筆頭発明者：沼田絵梨子）

[15] 国際 PCT/JP2012/061154（2014 年 8 月 7 日公開, 2012 年 4 月 19 日出願）特願 2013-512419（2013 年 9 月 30 日出願）「アプリケーションアーキテクチャ設計方法，アプリケーションアーキテクチャ設計装置，および記録媒体」

[16] 特願 2013-162131（2013 年 8 月 5 日出願）「情報処理装置」

[17] 特願 2013-161208（2013 年 8 月 2 日出願）「アセット情報管理装置，方法，及びプログラム」

[18] 特願 2013-16128（2013 年 8 月 2 日出願）「アセット抽出装置，アセット抽出方法及びプログラム」

[19] 特願 2013-138897（2013 年 7 月 2 日出願）「影響範囲分析システム，影響範囲分析方法および影響範囲分析プログラム」

[20] 特願 2013-043174（2013 年 3 月 5 日出願）「サービス提供構造管理装置，サービス提供構造管理方法，および，プログラム」

[21] 国際 PCT/JP2013/000431（2014 年 7 月 31 日公開，2013 年 1 月 28 日出願）「Method and system for transforming specification scripts to program code」（筆頭発明者：Sathawornwichit Chaiwat）

[22] 国際 PCT/JP2012/069937（2014 年 2 月 6 日公開, 2012 年 7 月 31 日出願）「A monification management apparatus, a modification management method and a modification management program」（筆頭発明者：Sathawornwichit Chaiwat）

[23] 国際 PCT/JP2013/005436（2014 年 3 月 27 日公開），特願 2012-204426（2012 年 6 月 26 日出願）「プロセスデータベース，プロセス管理装置，プロセスデータベース作成方法，プロセスデータベース検索方法，および，プログラム」

[24] 特願 2012-142900（2012 年 6 月 26 日出願）「資産化装置，資産化方法および資産化プログラム」

付録

目次

付録 A：モデリングツール一覧

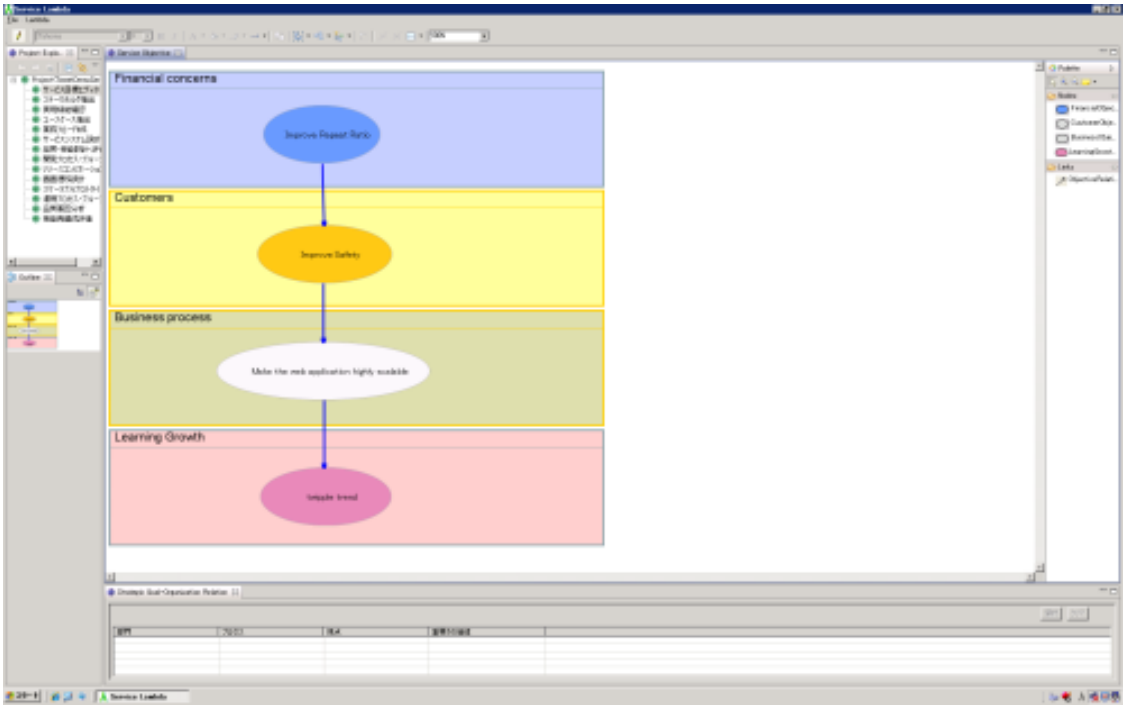

図 A-1 サービス目標モデリング画面

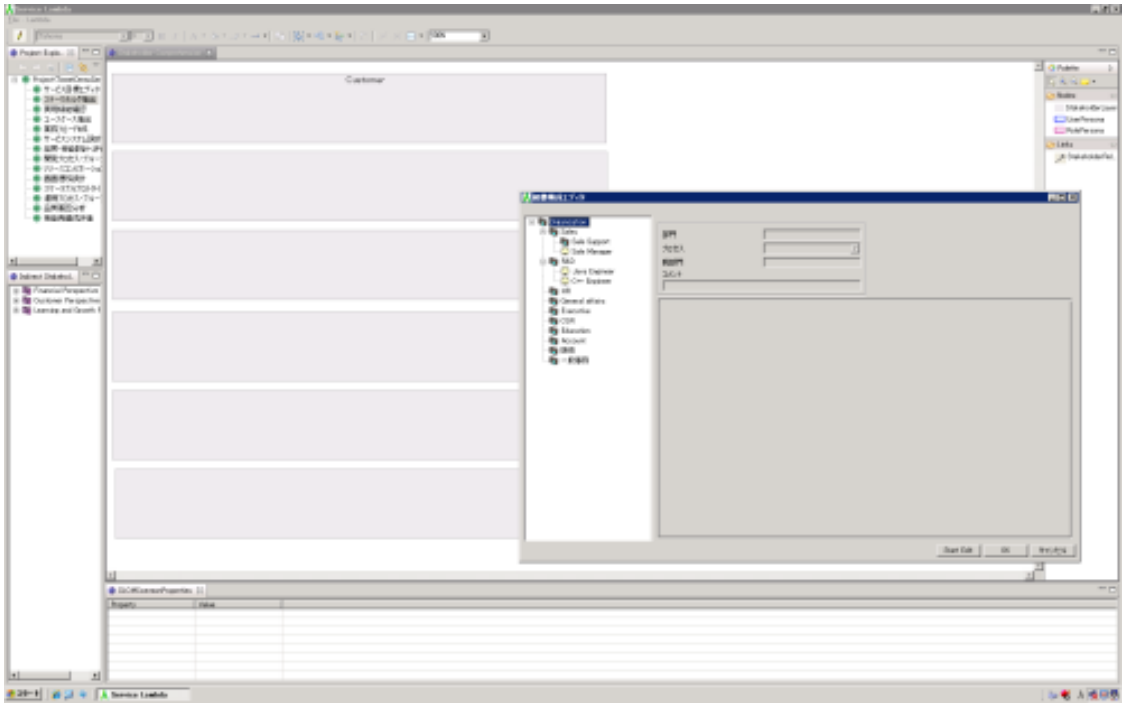

図 A-2 ステークホルダ抽出モデリング画面

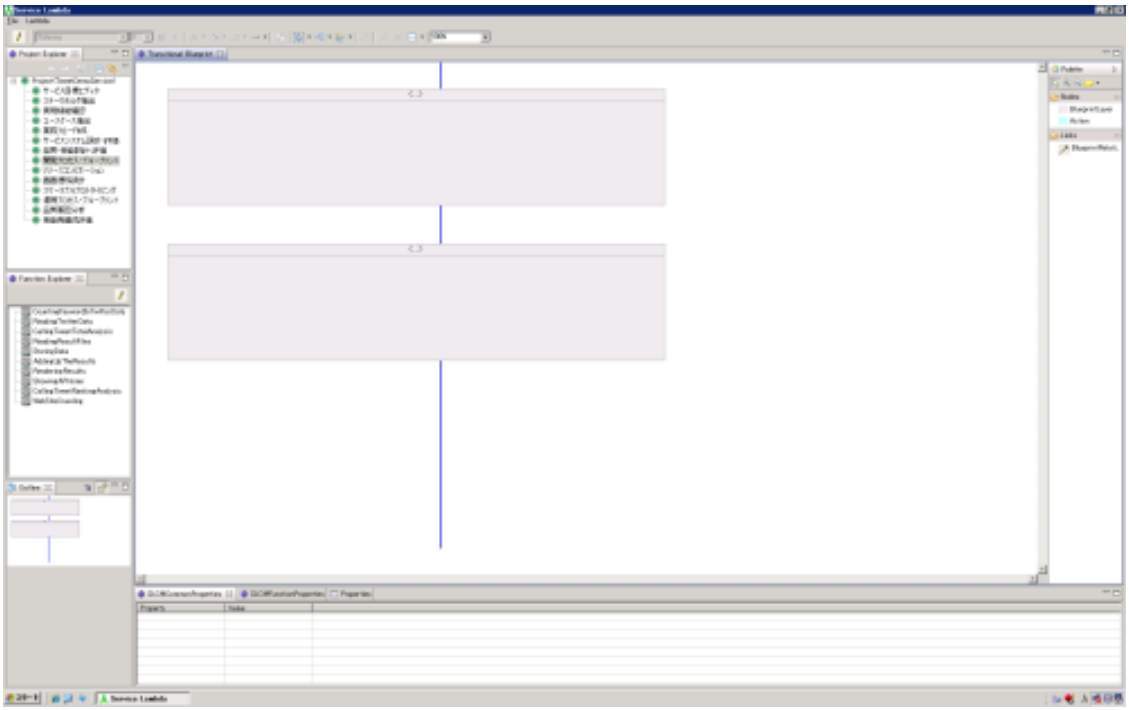

図 A-3 開発プロセスブループリント画面

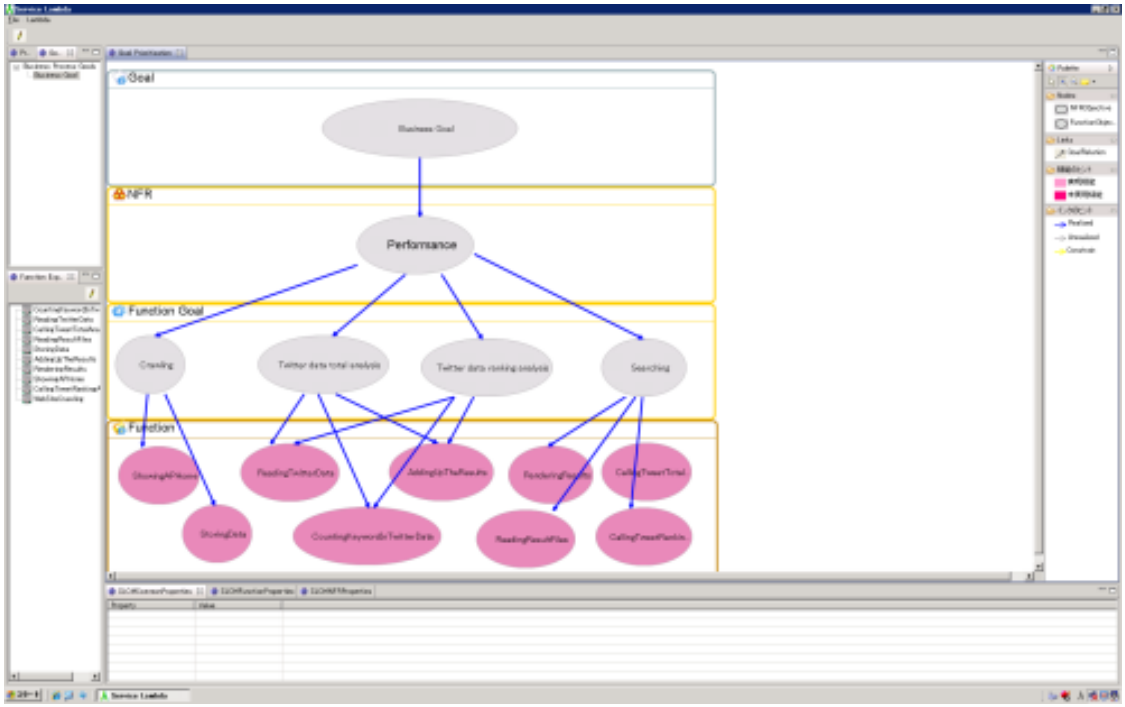

図 A-4 要求・機能展開モデリング画面

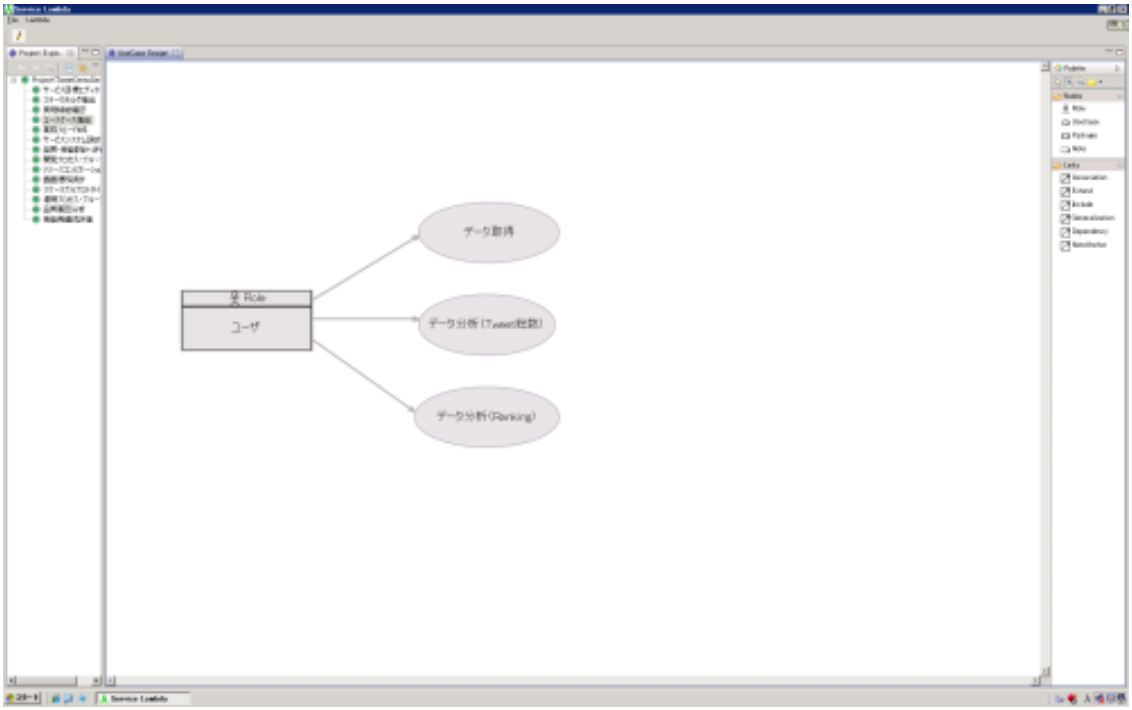

図 A-5 ユースケースモデリング画面

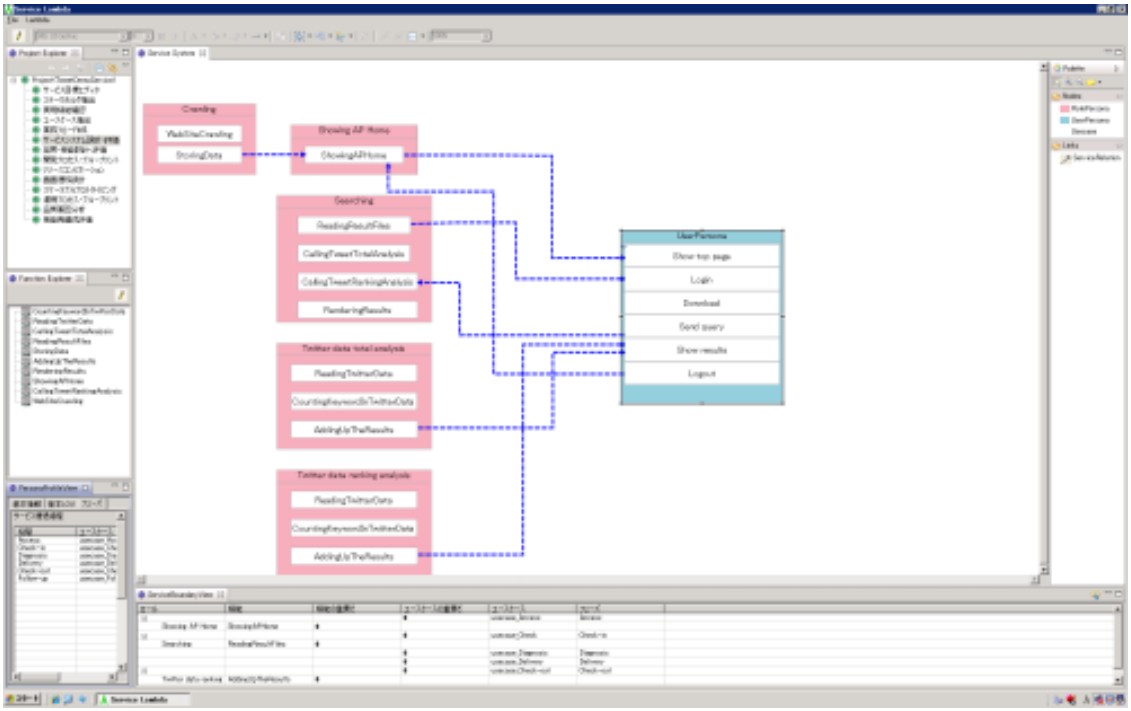

図 A-6 サービスシステムモデリング画面

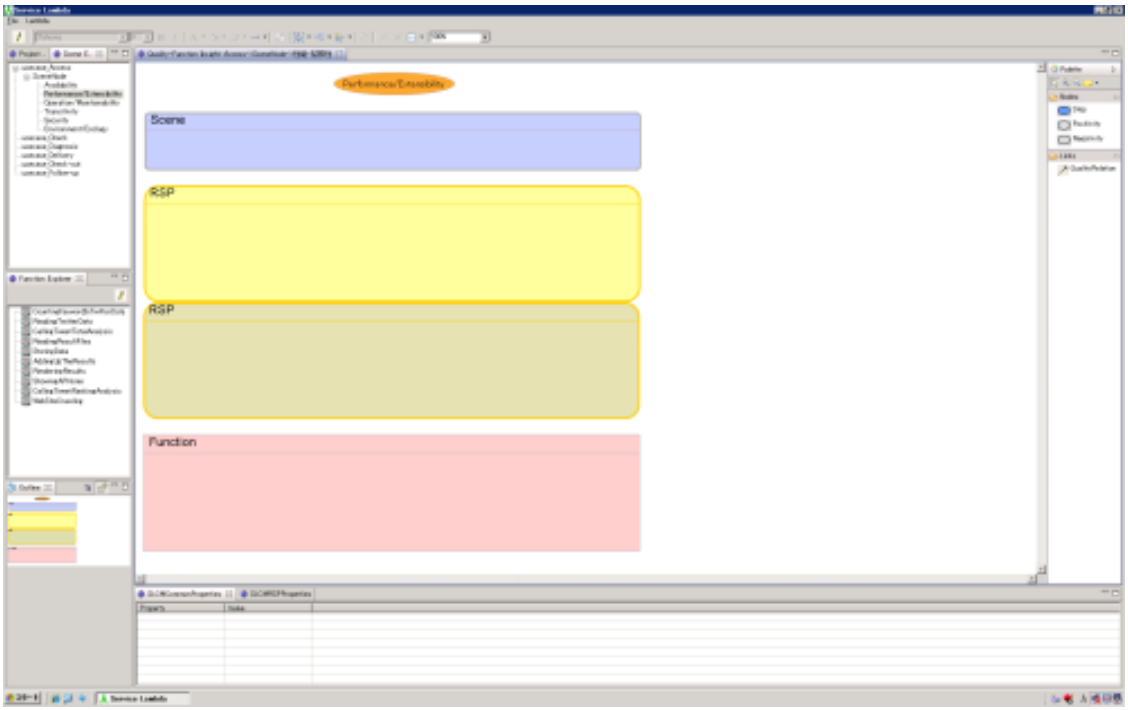

図 A-7 品質・機能モデリング画面

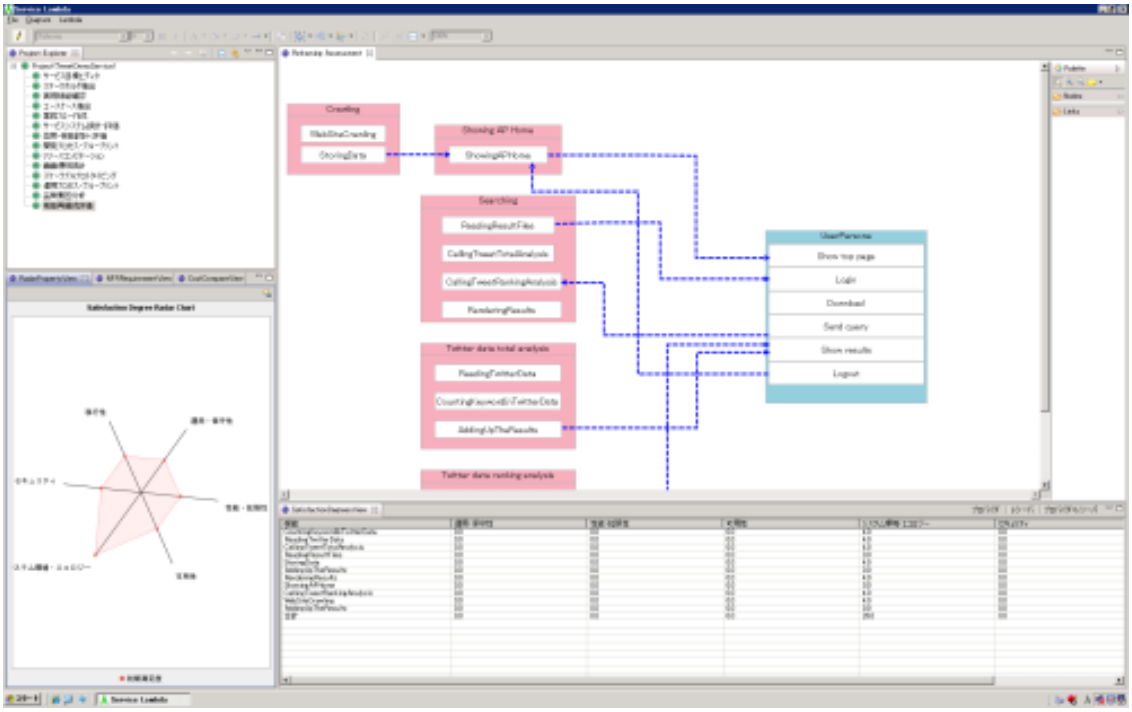

図 A-8 機能再編成評価画面

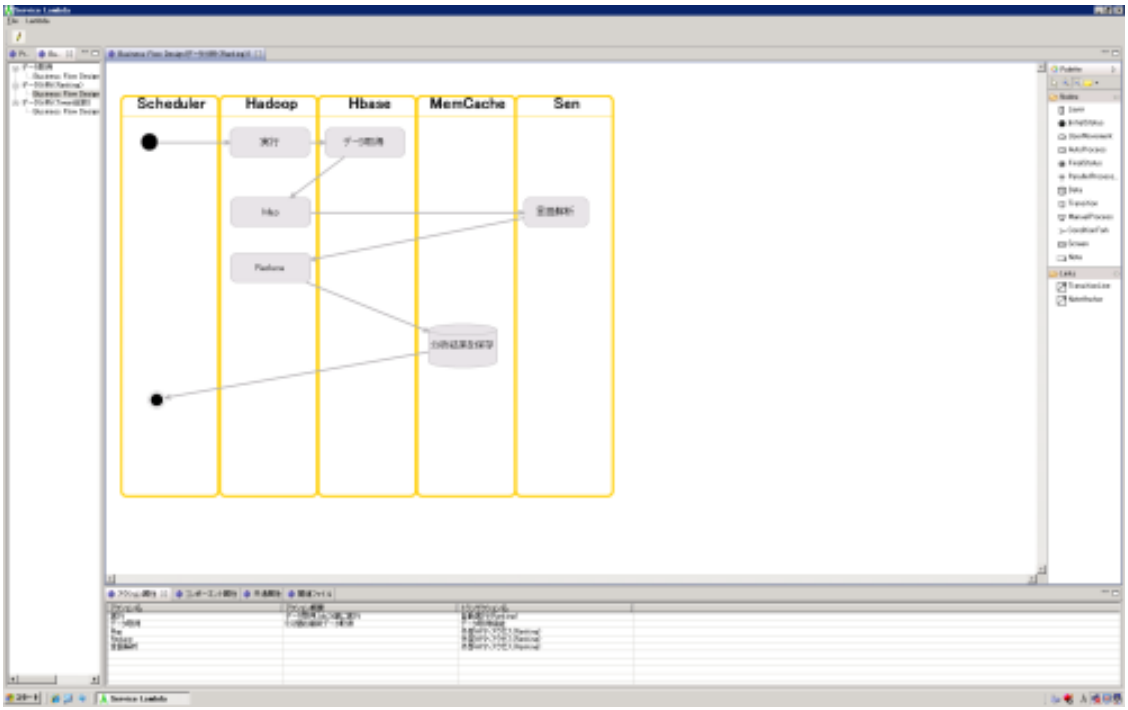

図 A-9 業務フローモデリング画面

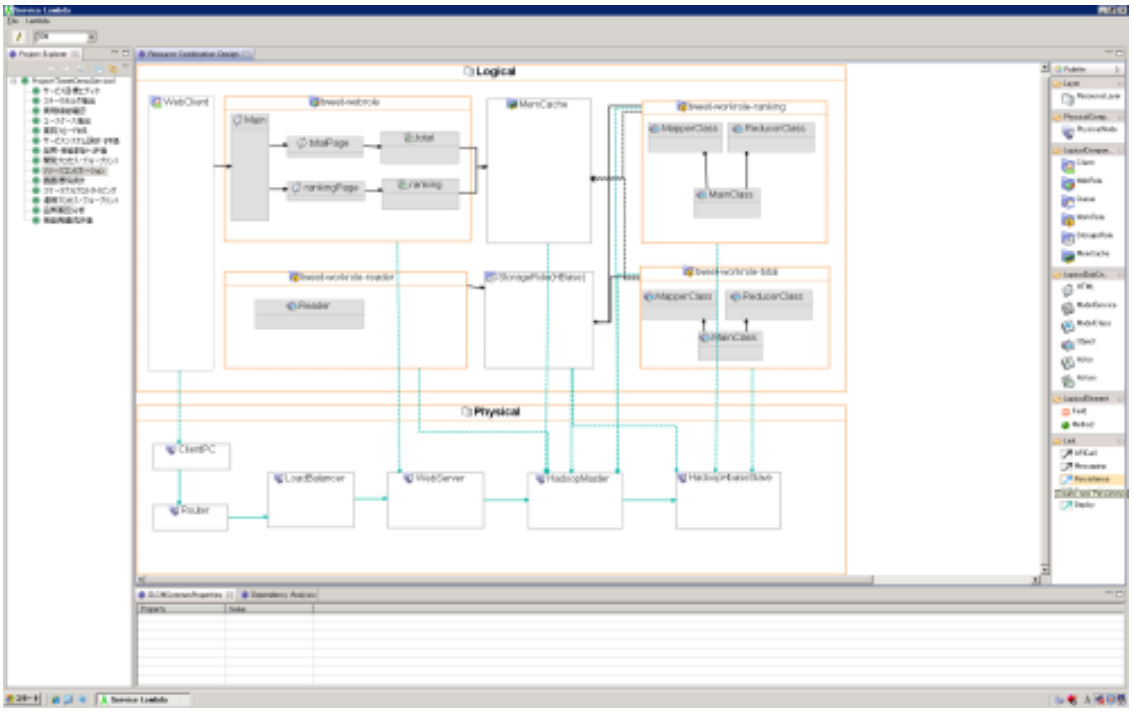

図 A-10 リソース割り当てモデリング画面

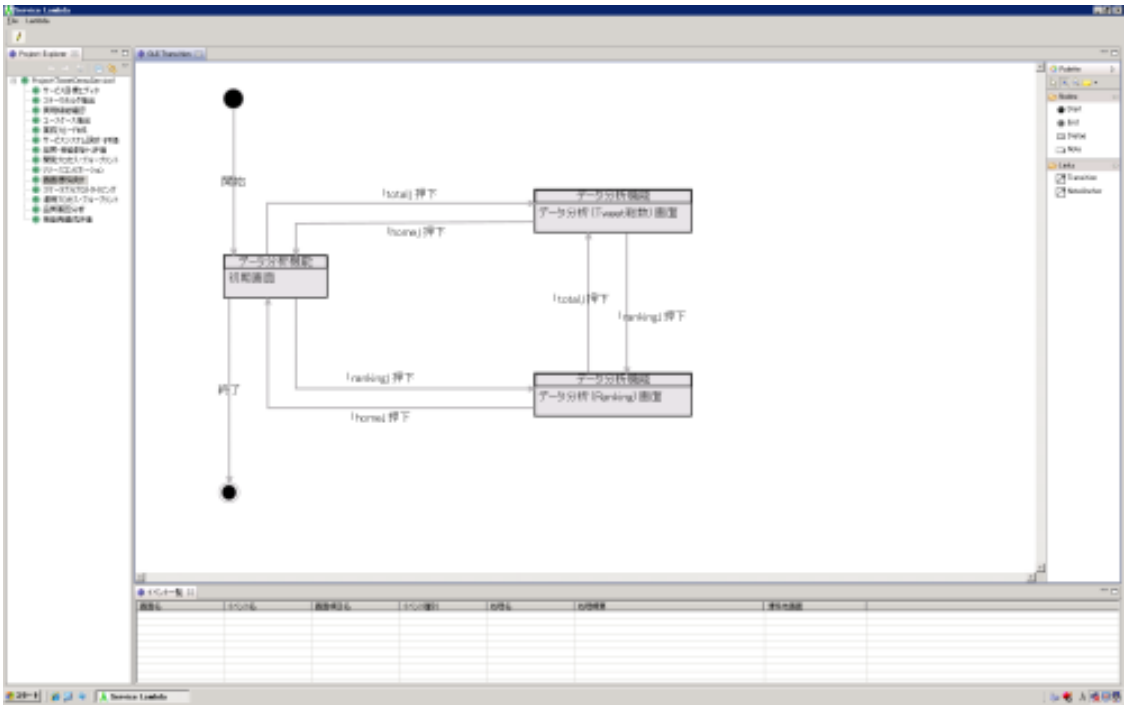

図 A-11 画面遷移モデリング画面

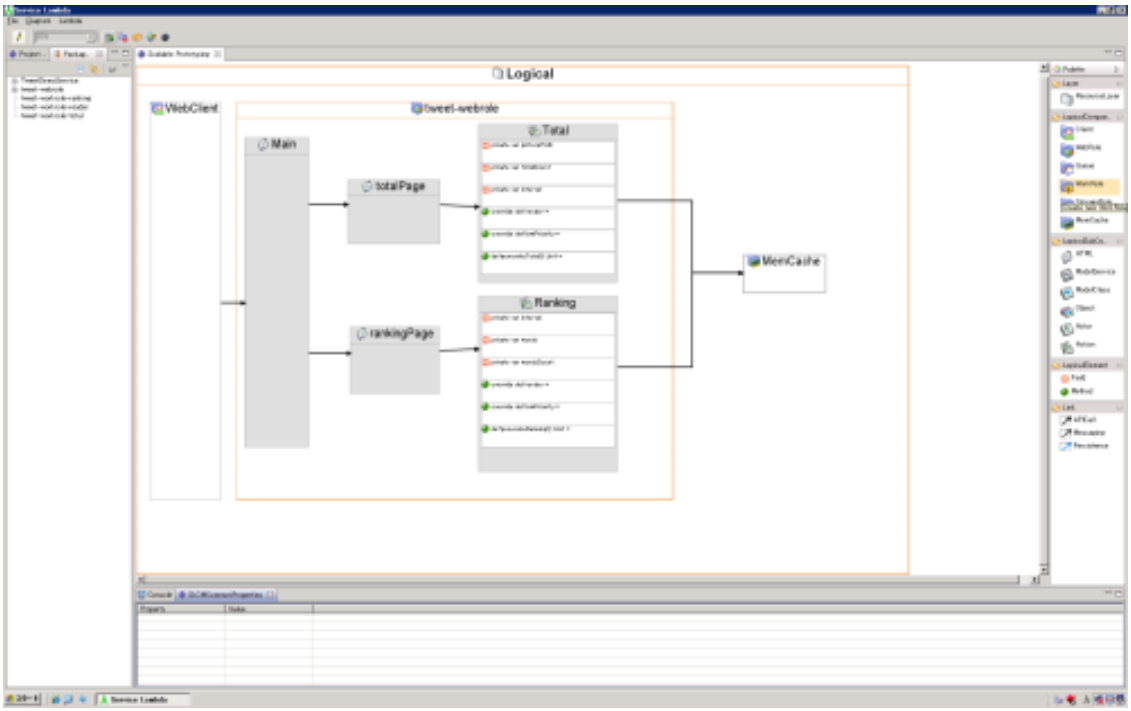

図 A-12 プロトタイピング画面

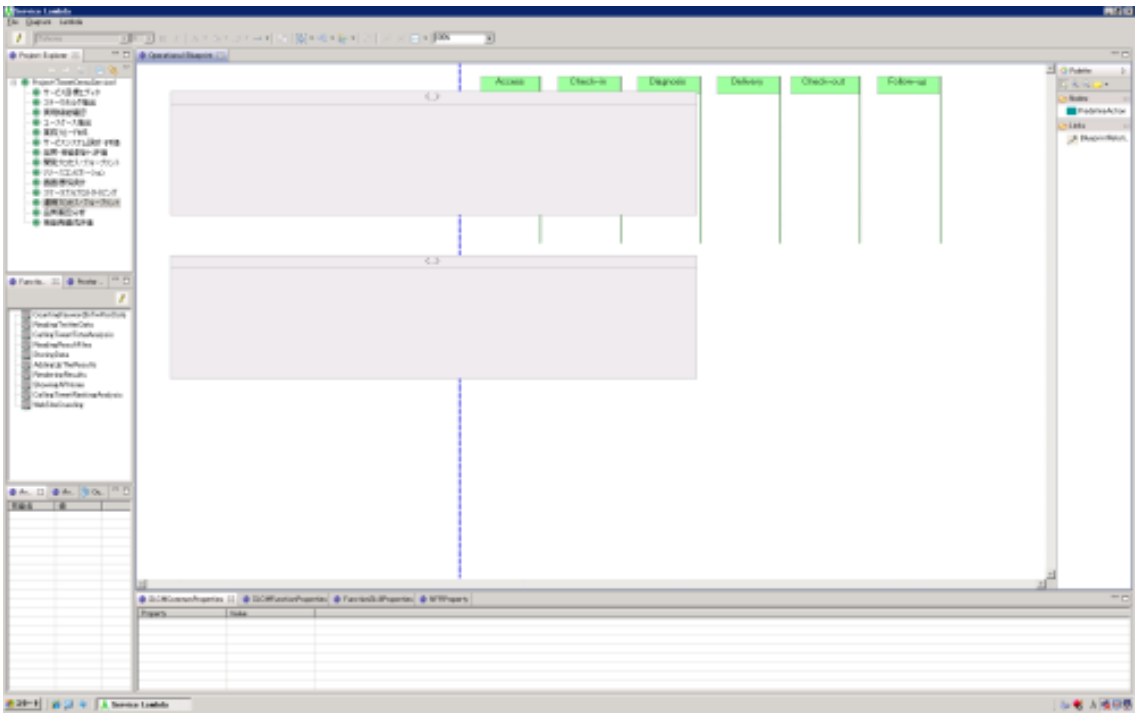

図 A-13 運用プロセスブループリント画面

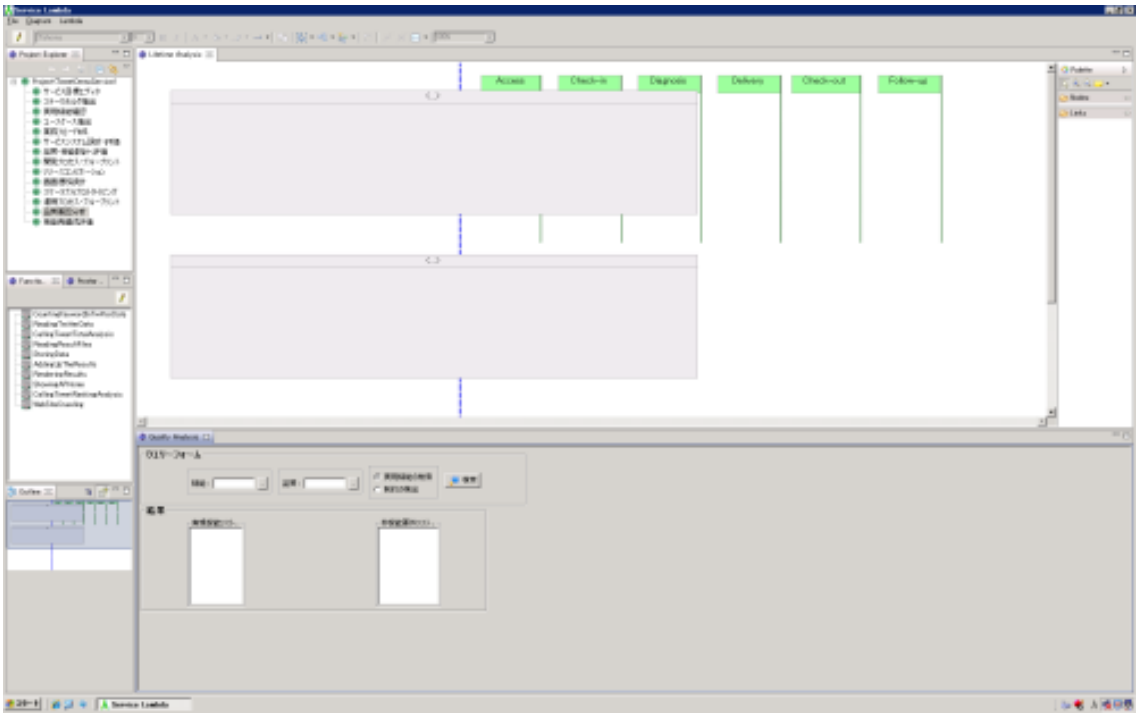

図 A-14 品質要因分析画面

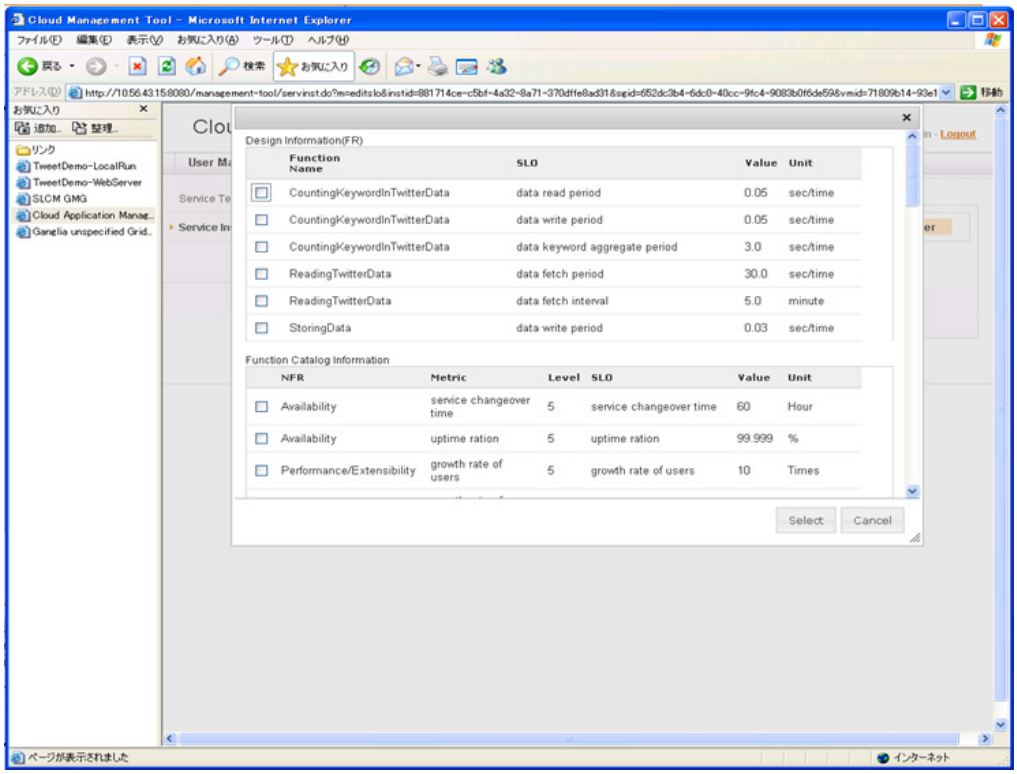

図 A-15 非機能要件監視・評価画面 1

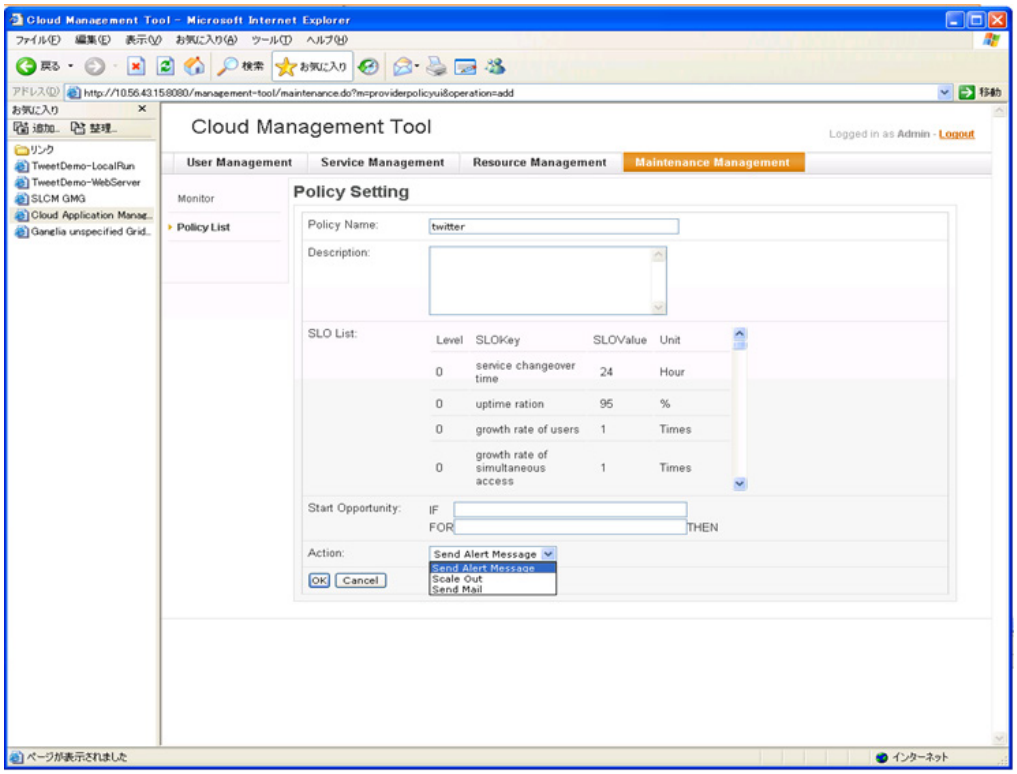

図 A-16 非機能要件監視・評価画面 2

付録 B：用語集

表 B-1 用語集

用語	説明
ICT（アイシーティー）	Information and Communication Technology の略称．情報通信技術を表す IT に，コミュニケーションの概念を加えたもの．
ITIL（アイティーアイエル，アイティル）	IT Infrastructure Library の略称．IT サービスマネジメントにおけるベストプラクティス集．1989 年にイギリス政府の CCTA によって公表され，以降，更新が続けられている．
アジャイル開発	ウォーターフォール型に代表されるプロセスおよび手続きを重視したソフトウェア開発手法に反し，迅速かつ適応的に開発を行う軽量な開発手法の総称．反復（イテレーション）と呼ばれる短い開発サイクルを繰り返し，徐々に機能を追加開発し顧客の要件に適合させることで，不要な開発を行うリスクを低減する．
IT サービス	Information Technology の略称．情報通信技術を総称したもの．
IT サービスプロバイダ	IT 分野のベンダで，コンピュータシステムやネットワークシステム等のコンサルティング・システム提案・開発・運用・保守を行う企業の総称．
Web サービス（ウェブサービス）	様々なプラットフォーム上で動作する異なるソフトウェア同士が相互運用するための標準的な手段を提供するもの．狭義には，HTTP などのインターネット通信方法を利用し，SOAP（Simple Object Access Protocol）と呼ばれる XML 形式のメッセージの送受信技術と，それを適用したサービスのこと．
SI（エスアイ）	System Integration の略称．SI を行う事業を SI サービスとも言う．また，この事業に従事する人を SIer または SE（System Engineer）と言う．
SOA（エスオーエー）	Service Oriented Architecture の略称．サービス指向アーキテクチャとも称する．アプリケーションなどを部品化し，それらを組み合わせてシステムを作る設計手法．
クラウドコンピューティング	ネットワークを通じてコンピュータリソースを利用す

	る形態．クラウドと略す場合もある．
クラウドサービス	従来，利用者が手元のコンピュータで利用していたデータやソフトウェアを，データセンタなどから機能ネットワーク経由で機能として利用者に提供するもの．SaaS（Software as a Service）は，アプリケーションソフトウェアの機能を，PaaS（Platform as a Service）は，データベースやアプリケーションサーバなどプラットフォームの機能を，IaaS（Infrastructure as a Service）は，ストレージや通信などの機能を提供する．
クラウドサービスプロバイダ	クラウドサービスを提供する事業者．自社のデータセンターに配備されたハードウェア，ソフトウェアの計算機リソースをネットワークを通じて，利用者に契約期間，貸し出しするサービスを提供する．
SEMAT（シーマット，セマット）	Software Engineering Method and Theory の略称．Ivar Jacobson，Bertrand Meyer，Richard Soley によって創設されたソフトウェアエンジニアリングの共通理解に向けた団体および運動． (cf.) http://www.semat.org, http://www.semat.jp/
SysML	System Modeling Language の略称．OMG（Object Management Group）によって公開された，システムをモデリングするための記述言語．
システムインテグレーション	情報システムの企画，設計，開発，構築，導入，保守，運用の一部または全てを請け負うサービス．
システムインテグレータ，SE（エスイー）／SIer（エスアイヤ）	システムインテグレーションを行う企業または，その担当者の総称．企業を指す場合，SI 企業とも言う．
WBS（ダブリュービーエス）	Work Breakdown Structure の略称．
チケット駆動開発	ソフトウェア開発の作業をタスクに分割し，バグ管理システムのチケットに割り当てて開発プロジェクトの進行を管理する開発方法
DevOps（デブオプス）	開発（Development）と運用（Operation）を組み合わせた造語で，開発部門と運用部門が密に連携して，運用側の要件を元に開発し，ビジネスルールなどの設計情報を元に運用方法を高度化する手法や概念．この実践により，小規模な開発とリリースを繰り返せる．
バグ管理システム	ソフトウェア開発において，発生したソフトウェアバグを管理するシステム．この管理の単位を一般にチケ

	ットと呼ぶ.
UML（ユーエムエル）	Unified Modeling Language の略称．OMG（Object Management Group）によって公開された，ソフトウェア工学におけるオブジェクトモデリングのために標準化した仕様記述言語.
USDM（ユーエスディーエム）	Universal Specification Describing Manner の略称. 要求を階層構造で表現し，仕様を漏れなく抽出する手法.