ＳＯＡにおいて、
いかにしてサービスを定義するか

株式会社岡村製作所　情報システム部
システム開発担当　　大塚　周一
システム開発担当　　下鉢　高之

【目次】

１　はじめに 1
２　ＳＯＡ導入の背景 1
２．１　現行システムの課題 1
２．２　ＳＯＡ導入の狙い 2
３　サービス決定アプローチ 2
３．１　ユースケース分析 2
３．１．１　ユースケース／アクターの識別 3
３．１．２　ユースケース図の作成 4
３．１．３　ユースケース記述の作成 4
３．２　クラス分析 5
３．２．１ 要素の抽出 5
３．２．２「責務」の割り当て 6
３．２．３　ユースケースの実現 8
３．３　サービス識別 11
３．３．１　サービスの決定 11
３．３．２　サービス一覧／仕様の作成 11
４　まとめ 12
５　むすび 12

【論文要旨】

ＳＯＡが注目されるようになって数年が経過したが、サービスとしてどの範囲の業務をどういった粒度で括るかについては、未だ明確なアプローチがないように見受けられる。その結果、ＳＯＡ導入のほとんどの事例が、技術的な基盤の導入と小さな範囲でのサービス化に留めている。

本論文では、弊社で２００８年より始動した生産管理システムの再構築プロジェクトにおいて、ＳＯＡを導入することにより、旧来の部分最適化されたシステムを、疎結合で再利用性の高いシステムへ変革させることを狙った事例を紹介する。その結果、ビジネスプロセスから適切なサービスを切り出すアプローチの方法に関しては、ある程度確立できたと考える。

サービスの再利用性が高く、ビジネスプロセスの変更に対して迅速な対応を容易にするＳＯＡの恩恵を受けるためには、独立性を備え、扱いが容易であり、適度な大きさを備えた粒度でサービスが構築されなければならない。そのためには、サービスを定義する段階で、実装における導入範囲の大小は別として、可能な限り広範囲の業務分析を行うことが望ましいと考える。

本事例がＳＯＡ導入の足がかりの一助になれば幸いである。

【論文キーワード】

（ＳＯＡ）（サービス）

１　はじめに

ＳＯＡ（Service Oriented Architecture：サービス指向アーキテクチャ）が注目されるようになって数年が経過したが、サービスとしてどの範囲の業務をどういった粒度で括るかについては、未だ明確なアプローチがないように見受けられる。その結果、ＳＯＡ導入のほとんどの事例が、技術的な基盤の導入と小さな範囲でのサービス化に留めている。

しかし、サービスの再利用や置き換えによる、ビジネスプロセスの変更に対して迅速な対応を容易にするといったＳＯＡの恩恵を受けるためには、適切な粒度でサービスの構築がなされていなければならない。それは、独立性を備えた、ユーザーにとって扱いが容易である、適度な大きさを備えた粒度である。仮に小さな範囲でシステムのサービス化を行った場合、後にＢＰＭ（Business Process Management：ビジネスプロセス管理）への適用に進む段階にて支障が生まれるであろう。

弊社はレガシーシステムからの脱却を目標に、２００８年より生産管理システムの再構築プロジェクトを始動しており、ＳＯＡを導入することにより、旧来の部分最適化されたシステムを、疎結合で再利用性の高いシステムへ変革させることを狙った。その中で、ＵＭＬ（Unified Modeling Language：統一モデリング言語）を用いてサービスの粒度と内容を決めていった。それは、以下のようなアプローチの方法である。

（１）ビジネスプロセスから、可能な限り全量のユースケースを作成する。
（２）ユースケースを分析して、ビジネスの要素を洗い出す。
（３）クラス図、シーケンス図を用いて、要素を適切な単位の責務でまとめる。
（４）単一の役割を持つサービスの括りを決める。

その結果､ビジネスプロセスから適切なサービスを切り出すアプローチの方法に関しては、ある程度確立できたと考える。

本論文では、その中で先行して行った生産計画業務のオープンシステム化の事例を通して、どのようにサービスの粒度と内容を決めていったかを手順と留意点を中心に述べる。なお、ＳＯＡの考え方についての詳細や、システムの詳細設計に関わる部分については割愛させていただく。

２　ＳＯＡ導入の背景

２．１　現行システムの課題

弊社の生産管理システムは、旧来のメインフレーム上に構築されたシステムを、業務変化や事業の拡大などに対応するために改修を重ねながら３０余年に渡って使用してきた。その結果、工場ごと、事業ごとに個別ルールの多い、部分最適化されたシステムとなっており、このことは以下のような弊害をもたらしている。

（１）業務の変更要件が部分的な範囲での変更にもかかわらず、システム変更範囲が広範囲となってしまっており、ビジネス変化に即応できない。
（２）システムの利便性を享受できる範囲とそのシステムを保守、運用するためのコストが適正であるかの評価をする仕組みを持っていないため、無駄な情報資産を保有し続けている。

（３）システム開発投資が正味の業務範囲以外にも波及しているため、費用対効果が悪化している。
（４）システムインフラが複雑化しているため、新技術の採用には多大なコストとリスクの覚悟を迫られる。新技術への対応が遅れ、情報技術の外部調達に支障が出ている。

これらの弊害は、他の業務領域にて既に構築されている、いくつかのオープン化されたシステムについても言えることである。そのため、新しいシステム構築手法の確立は急務であった。

２．２　ＳＯＡ導入の狙い

本プロジェクトでは、ＳＯＡを導入することにより以下の効果を狙った。
（１）共通のビジネスプロセスについては再利用可能なサービスとして構築する。これにより、今後、事業分野の異なる工場へ展開する際、最小限の工数でのシステム導入を可能とする。
（２）サービスの役割を明確化し疎結合とすることで、ビジネスルール変更時の開発工数を最小限にとどめる。
（３）ＳＯＡの考え方に基づくシステム構築手法を確立し、他の業務分野におけるシステム開発についてもこれを適用する。

３　サービス決定アプローチ

先にも述べたとおり、ＳＯＡの導入検討を行うにあたって、まず始めに悩まされる問題がサービスの粒度についてである。本項では、今回の事例において検討し認識した考慮ポイントを中心に、サービス決定までのアプローチについて述べる。

３．１　ユースケース分析

弊社では今までシステムを構築するにあたり、まず画面設計があり、それを実現するために実装を意識しながら必要な機能に落とし込む設計方法を採用してきた。しかし、この方法では実装上の都合が優先され、サービスが提供する価値（＝再利用性）や、サービスの括り（＝疎結合性に影響）が蔑ろにされる可能性が高いため、今回サービスを検討する上では不適と考えた。

ユースケースとは「システムがどう動くか」ではなく「ユーザーがシステムを使って何をするか」に着目した表現方法である。今回の事例においては、以下の点を理由に、まずユースケースを作成し、それらを分析して抽出される要素を適切な単位の責務でまとめることで、単一の役割を持つサービスの括りを決めていくこととした。（図１参照）。
（１）ビジネスプロセスと要求される機能の関連性が明確となるため、ユーザーの立場からしても違和感のないサービスを作成することができる。
（２）ビジネスプロセスから可能な限り全量のユースケースを洗い出すことで、網羅性を確保した上で、サービスの設計が行える。

なお、ここでいうビジネスプロセスとは業務目的を達成するまでの一連の作業であり，ユーザーが行う操作（＝オペレーション）ではないことに注意しなければならない。

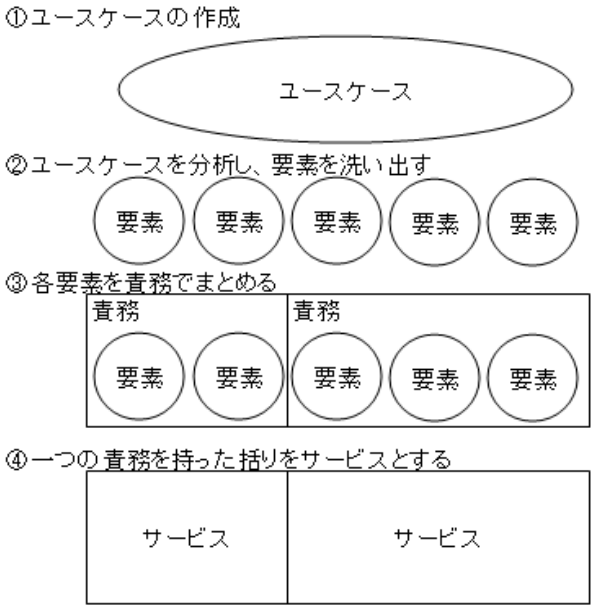

図１：ユースケースを元にしたサービスの設計

ユースケース分析による成果物は以下の通りである。

（１）ユースケース一覧

全ユースケースを列挙し、属性情報と併せて一覧化。

（２）ユースケース図

ユーザーとシステムの関係性を静的なモデルとして図式化。

（３）ユースケース記述

ユースケースについて、起点となるビジネスプロセスや処理の流れを文章化。

３．１．１　ユースケース／アクターの識別

ユースケース分析を行うにあたり、まずはじめに行うのがユーケース／アクターの識別である。一般的にユースケースとは、システムがユーザーに提供できる機能を１つのユースケースとして抽出し、ユーザー（もしくは外部システム）とシステムとのやり取りを描いたものが多く、粒度についても使用用途によりさまざまな定義がある。

今回の事例では、ここで作成されるユースケースを元にサービスの設計を行うことになる。そこで、ユースケースについて分析した結果抽出される要素と、ビジネスプロセスの関連性を明確にするために、ユースケースは「何のために」「何を」「どうする」が具体的になるよう文章形式で表すことを意識した。これは、ユースケースの粒度を揃える意味でも重要となる。例えば、生産計画担当者の業務として、

「(受注の）納期不満足を解消するために、生産計画を変更する」

「生産能力の余剰を解消するために、生産計画を変更する」

といった２つのユースケースを抽出しているが、これらはいずれも「生産計画を変更する」という１つの業務行為で達成できる。しかし、サービスの設計にユーザーの視点を持ち込むことを考えると、目的までを含んだかたちで業務と機能の関連性が見えていることが重

要となる。

３．１．２　ユースケース図の作成

ユースケース／アクターの識別が終わると、それらの関係を静的に表すユースケース図の作成を行う。ユースケース図の単位については基本的に業務領域単位で描かれるが、ここでも無意識のうちにデータモデルや物理的な制約といったものに引きずられないように気をつけなければならない。また、この時点で各ユースケースが使用する共通機能が明確となっている場合は､関連付けられた別ユースケースとして共通機能を切り出しておく(図２参照)。

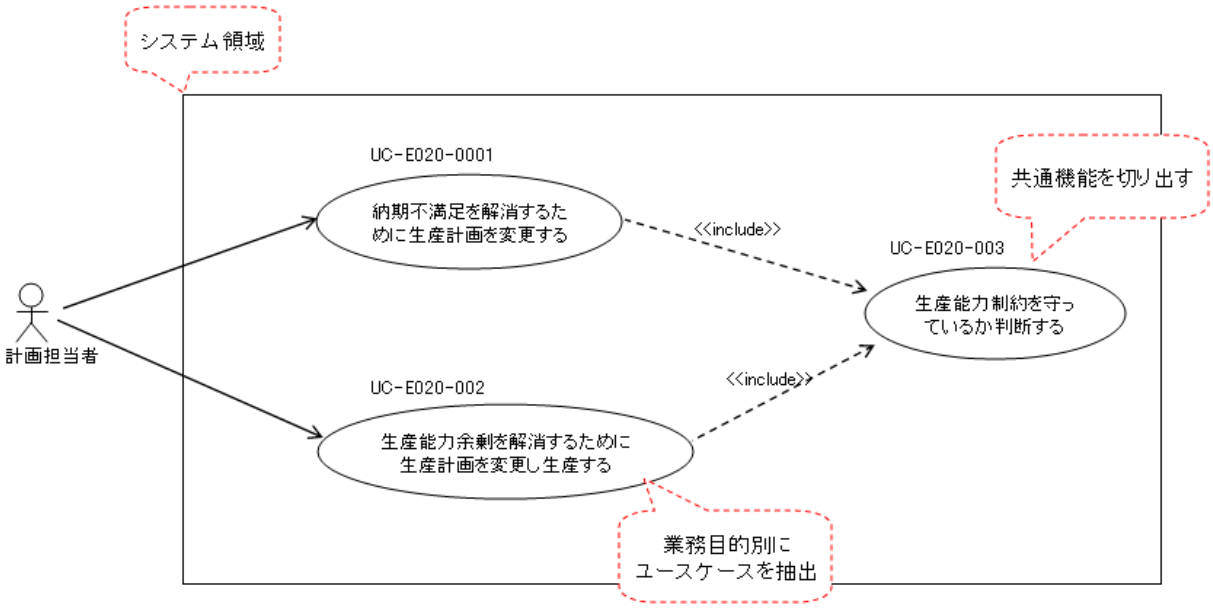

図２：ユースケース図

３．１．３　ユースケース記述の作成

次に各ユースケースについての詳細な内容を文章で表現する。この段階で行われる詳細な内容とは、あくまでブラックボックスとしての記述に留め、実装技術やシステム内部の細かな振る舞いは記載しない。今回のプロジェクトでは、以下に示す項目が記載された成果物をユースケース記述とした。

（１）ユースケースが実行される元となるビジネスプロセスやイベント。
（２）イベントフロー(処理の流れ。基本フローと代替フローの２種類。)
（３）事前条件と事後条件。

また、ここで登場する語句の定義や計算方法、処理／データ制約については直接ユースケース記述には記載せずに、別途、ビジネスルールとして関連性がわかる形でまとめていく。これにより、繰返し登場する語句やビジネスロジックの内容を一元的に管理することができる。例えば､｢受注の納期不満足｣に関して､｢受注の応答納期が希望納期よりも遅い｣とった具体的な内容についてはビジネスルールに記載し、ユースケース記述には参照先がわかる形での記載に留めた。

３．２　クラス分析

ユースケース分析では、生産計画業務に関するビジネスプロセス及び、本プロジェクトで取り込む新しいルールを目的別に明らかにした。本項では、これらユースケースをシステムとしてどのように実現するか検討を行う。まずはユースケースからルールや情報などビジネスの要素を漏れなく洗い出し、次に各要素を適切な責務に割り当て、そして分析クラス図・シーケンス図を利用してシステムの全体像及び処理の流れを検討する。結果として、以下の項目が達成されることに注力する。

（１）ユースケースが正しく実現できること。
（２）予見される再利用と機能拡張が容易であること。
（３）抜け、重複、無駄となる記述がないこと。
（４）サービスの設計を意識した責務が作成されていること。

クラス分析での成果物は以下の通りである。

（１）分析クラス図

ユースケースで洗い出された要素を組み合わせて作られる責務を担うクラス同士の、静的な関係を表現した図。主に、責務の割り当てを検討するために使用する。

（２）分析シーケンス図

ユースケース記述のイベントフローを表現し、一連の処理の動的な流れを記述した図。主に、機能に漏れがないかどうかを確認する。

３．２．１ 要素の抽出

クラス図を作成するにあたり、各クラスの元となる要素をユースケースから抽出する。ここで言う「要素」とは、ユースケース記述において登場した、あるいは登場していないがユースケースを実現するのにあたって必要と考えられる情報、人、振る舞い、ユーザーインターフェース（UI）のすべてである。

たとえば「生産計画を作成する」では、以下の要素が抽出される（図３）。

・生産計画 UI　　（ユーザーインターフェース）
・生産計画情報　　（情報）
・生産計画情報を照会する　（人、振る舞い）
・生産計画情報を追加する　（人、振る舞い）
・受注情報　　（情報）
・受注情報を照会する　　（人、振る舞い）

ここで、ユースケース分析で登場しなかった言葉も要素となる。ユースケース記述で「生産計画 UI」という言葉は登場しなかったが、ユースケース図においてアクターとユースケースがシステム境界を越えて結ばれているため、外部とのインターフェースが存在する（３．１．２　図２参照）。

注意点として、複数のユースケースで使われているひとつの共通の要素を異なる言葉で書き出してしまうと、設計時に別物と認識してしまい機能が重複してしまう恐れがある。複数の設計者が並行して分析する場合は、打合せ等を通じて用語を統一化し、重複を防ぐことが重要である。

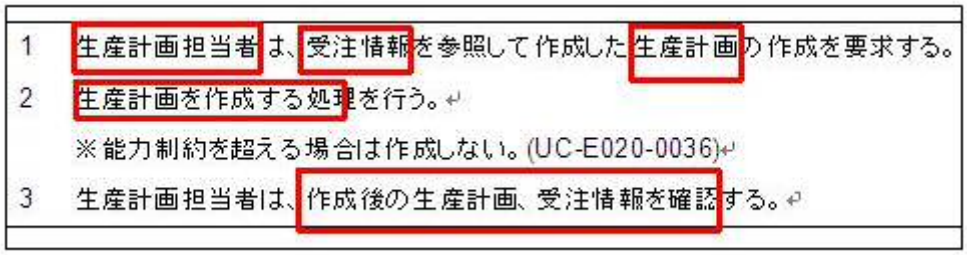

図３ ユースケース記述からの要素抽出

３．２．２「責務」の割り当て

次に、ばらばらに洗い出された各要素を、概念的に同じものを扱う「責務」を持つクラスにグループ化する。この際、UML で定義されている 3 種類のクラスに分類し、それぞれのクラスに対して明確な１つの責務を与える。

・バウンダリー

システムの境界を担う。「～インターフェース」と命名する。

具体的には、ユーザーインターフェース、及び外部システムインターフェースを用意した。

・エンティティ

情報を担う。「～情報」と命名する。

論理データモデリングは､本来ならばクラス分析と並行して行うのが理想と思われるが、本プロジェクトではデータモデリングが先行していた。分析クラス図では必要以上に詳細になることを避けるため、データをいくつかのグループに分けて１つのエンティティとして扱うこととした（図４）。

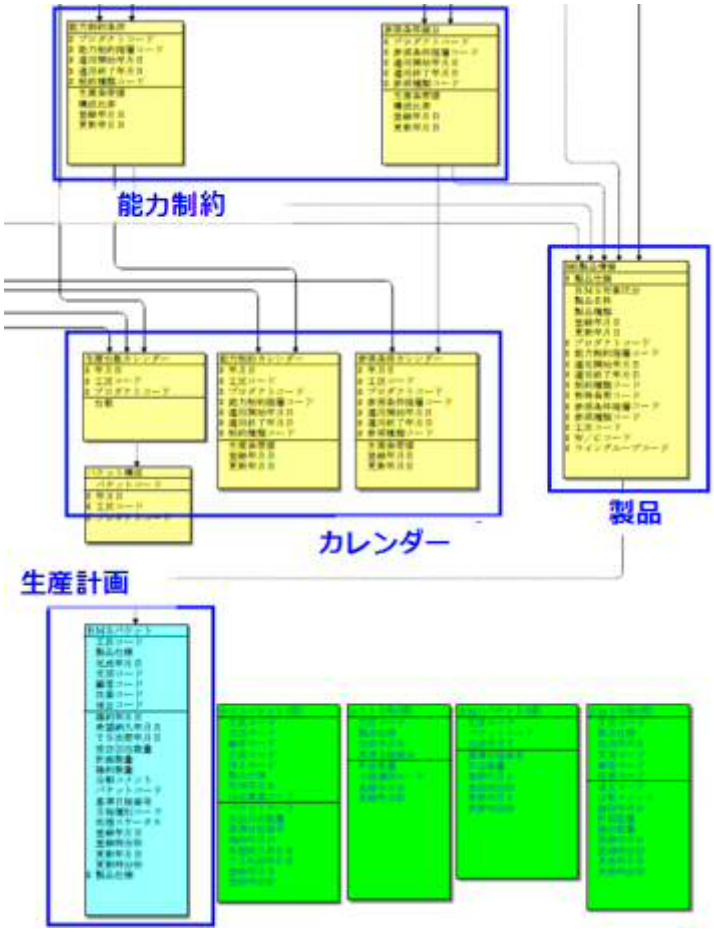

図４　エンティティの分類

・コントロール
　振る舞いを担う。
　基本的に、１つのエンティティに関わる振る舞いの責務を担う１つのコントロールが設定される。バウンダリー、エンティティとは重複しない名称とする。なお、今回はバウンダリーについても対応するコントロールを設定したが、これは実装において UI 側の処理とビジネスロジック側の処理を明確に分けたかったためであり、この段階で実装方式が見えていない場合には特に設定する必要はない。

　上記でも触れたが、本プロジェクトでは、正規化されたデータモデル上の情報をいくつかのグループに分け、それらを扱うことを各クラスの「責務」と考えた。
　ところで、今回は正規化されたデータモデルを意識しすぎたため、トランザクション情報とマスター情報を異なる責務として考えてしまった。例えば、トランザクション情報「生産計画情報」を扱う責務とマスター情報「能力制約情報」を扱う責務は、別々のものと考えた（図５-a)。ところが、「能力制約情報」は「生産計画情報」を変更する際にほぼ必ず使われ、両者は強く関連している。仮に正規化されていない状態から、ユースケースを元に業務目線で責務を考えたとすると、両者は１つの責務に内包されるべきであった。(図５

-b)。場合によっては、データモデルの非正規化を行う必要もある。

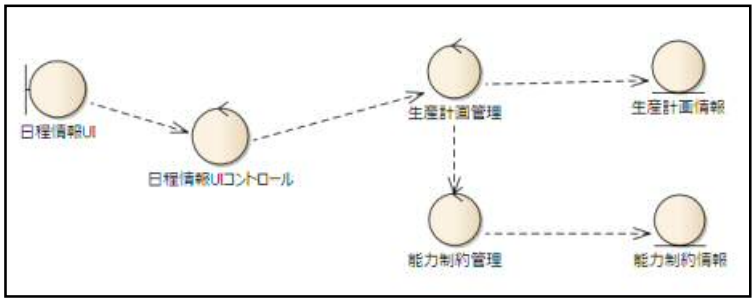

図５-a 正規化されたデータモデルを意識した場合
（「能力制約情報」を扱う責務がある）

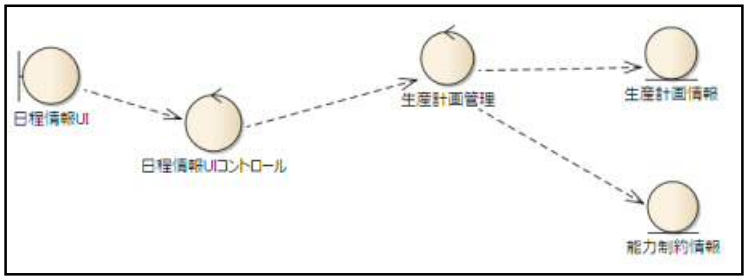

図５-b ユースケースを元に業務で責務を考えた場合
（「能力制約情報」を扱う仕事は「生産計画」の一部）

さて、この責務をもつクラスは、提供するサービスの候補となる。よってクラスの大きさはサービスの粒度に深く関係することを意識して責務の割り当てをすべきである。

責務の名前は、実クラス名のように名前から中の詳細機能が想像できてしまうような名前ではなく、たとえば「生産計画」「製品」などといった、粒度は大きいが一意の単語で言い表されるように心がけ、「日程」など曖昧な名称とならないように気を付けた。なぜなら、理解し辛い名前を付けてしまうと、得てして「何でも屋」の巨大クラスとなり、再利用もできず変更にも弱くなってしまう恐れがあるからである。重要なのは、クラスが一意の責務を持つ、すなわち独自に強く関連する振る舞いを担い、関連しない振る舞いは担わないことである。これにより、クラス同士が疎結合となり、再利用性が高く変更に強い構造になる。これを実現するには、「名は体を表す」を肝に銘じ、レビューを通じて適切な名前、適切な粒度かどうかを複数の人間で合意する必要がある。

なお、外部システムとのインターフェース部は、今後、外部要件により変更する可能性が高い。その際の影響を最小限に抑えるため、「外部インターフェース」を責務としてこの時点で分けた。

３．２．３　ユースケースの実現

ここまでの分析で、明確な「責務」を持ったクラスを作成した。次に、これらを組み合わせてユースケースを実現する方法を考える。

まずは、クラス同士の静的な関連を表すクラス図を作成し、本来、他のクラスが担うべき責務を持ってはいないか、責務の大きさに統一感があるかどうか、などの観点をチェックする。たとえば、似通った機能を持っている場合、クラス同士で相互的にやり取りする回数が多い場合、また関連する情報や機能が他のクラスと類似している場合は、責務の統合を検討する必要がある。このように、あらゆる角度からクラスの統合・分割などを検討する。

次に、各ユースケースの処理の流れをシーケンス図で表す。そして、上記クラスの組み合わせのみで無理なく実現できているかを各ユースケースと照らし合わせて確認する。ここで機能の漏れや、責務の割り当てが適切でないなどと気づいた場合は、再びクラス図に立ち返って修正する。

なお、ここでは各ユースケースの実現性を確認することが目的であるため、実装上の細かい制約には敢えて目をつむる。例えば、情報の生存期間などは「必要な処理は一度行っておけばいつでも使用可能」とし、後の工程で考える問題と割り切る。また、重複や抜け漏れを防ぐために、振る舞いの表現や用語はユースケースと同じ文言とすべきである。

ここでは一例として「生産計画」を挙げる。以下のような改善点が見つかったため、クラス図とシーケンス図の修正を行った（図６-a、図６-b）。

・「日程情報 UI」(バウンダリー)から、「生産計画管理」(コントロール)を介さずに「生産計画情報」(エンティティ)にアクセスしていた。
　→情報取得はコントロールの役割のため、修正。
・「生産計画管理」で、「製品情報」を直接取得して処理していた。
　→「製品情報」の取得は「製品情報管理」の責務のため、修正。

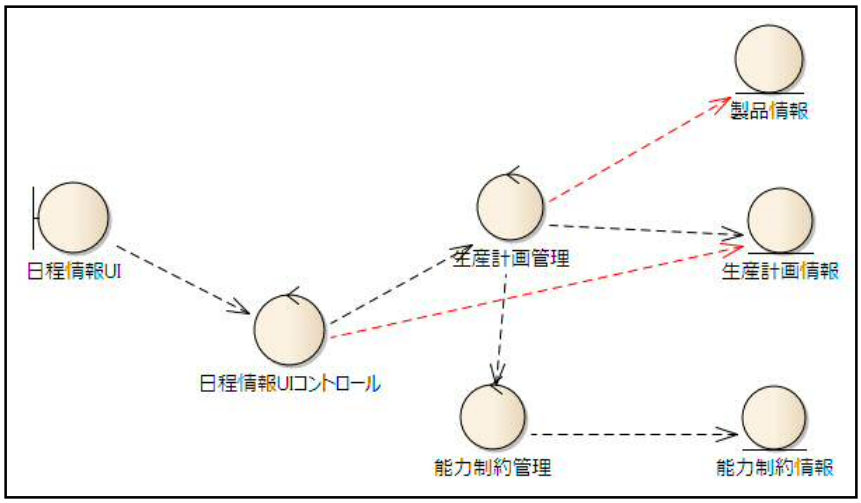

図６-a 分析クラス図の修正（修正前）

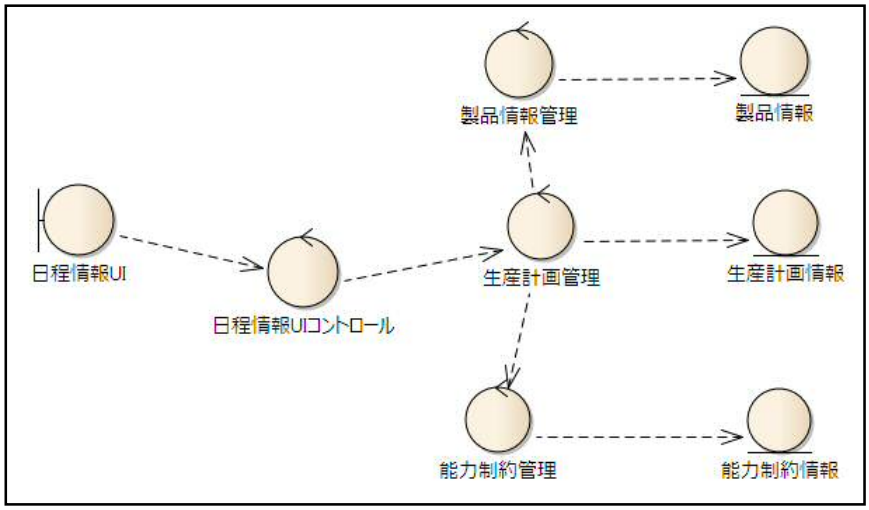

図 6 -b 分析クラス図の修正（修正後）

最後に、「生産計画」の最終的な分析クラス図を図 7 に示す。修正の際に拠り所となった考え方については、都度メモを図中に残すことで、後に思い出しやすいように工夫した。ここで責務を明確に認識することが、適切なサービス識別を行うための拠り所となるため、クラス分析では責務が曖昧な振る舞いや情報が無い状態にすることが重要だと考える。

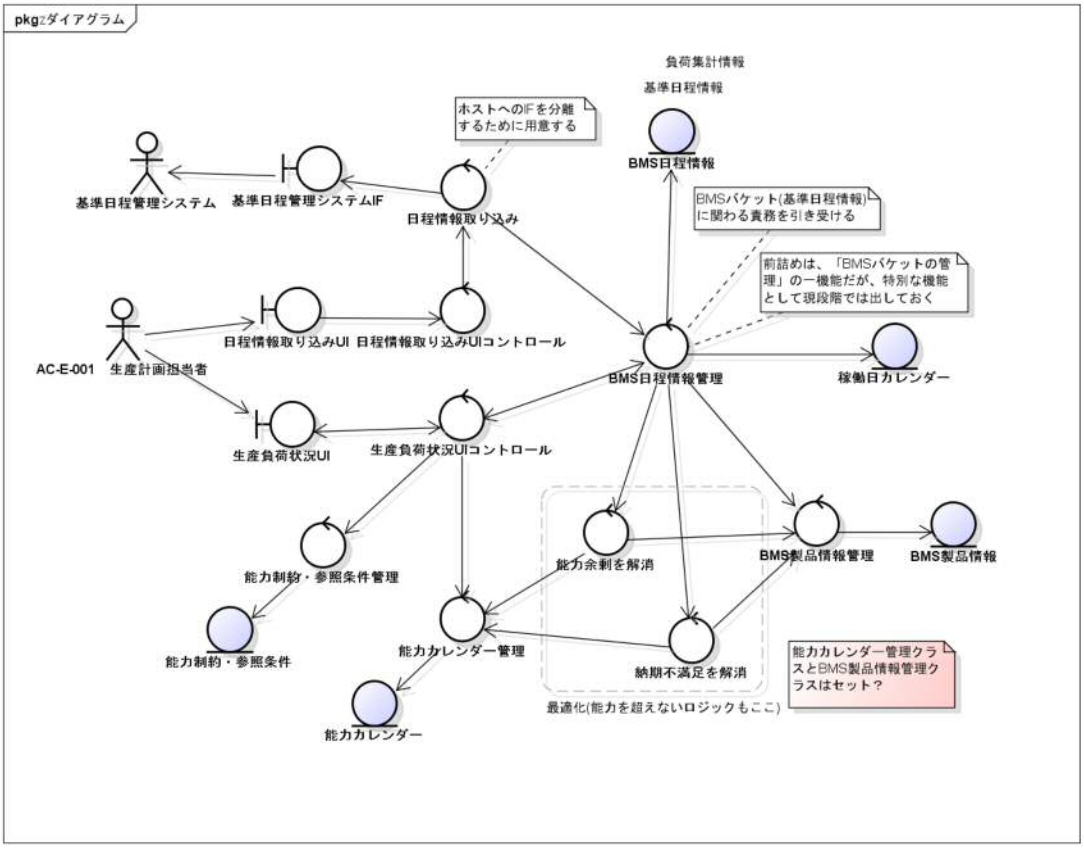

図 7 「生産計画」の分析クラス図

３．３　サービス識別

クラス分析が終わると、そこで作成されたクラスのうち、再利用性の観点から公開するべき責務を持ったものをサービスとして抽出し、それらの仕様を決定していく。基本的にはクラスとしてまとめられた括りが、責務の一貫性と親密度の観点からそのままサービスの括りとなる。しかし、この段階で注力しなければならないのは、あくまでサービスを利用するのはユーザーであるという前提に立ち、ユーザーが利用判断をする際に違和感／過不足のないものになっているかどうかを確認することである。そのためには、サービスの入力情報／出力情報が明確であり、かつ、そのサービスを利用することにより、ビジネスとしての効果を得られる括りであることが大切である。

サービス識別によるアウトプットは以下の通りである。

（１）サービス一覧

全サービスとそれらに含まれる操作を列挙し、属性情報と併せて一覧化

（２）サービス仕様

各サービスについての仕様を文章化

３．３．１　サービスの決定

前述の通り、基本的にはクラス分析で作成されたクラスの中から、サービスとするものを選択することが、主な作業となる。この際、ユースケース、論理データモデル、クラス分析の結果のすべてを見渡し、必要によっては各作業に戻って修正を行いながら、ユーザーの視点で適切なサービスの要素（括り／名称／入力情報／出力情報）を決めていくことが大切である。また、今まで触れられてこなかった範囲外の業務領域についても、ある程度、予測を含めながら考慮して作業を行うべきである。それにより、より汎用性に富んだサービスとなり、今後、ユーザーから見たサービスの変更を最小限に留めることにつながる。そのために、広い範囲の業務知識を持った人材や、場合によってはユーザーを含めた議論を通して、最終的なサービスを決めていく必要がある。

３．３．２　サービス一覧／仕様の作成

サービスが決定されると､次にそれらサービスについての詳細な内容を文章で表現する。今回のプロジェクトでは、以下に示す項目が記載された成果物をサービス一覧／仕様として残した。繰り返しにはなるが、これら成果物はあくまでブラックボックスとしての記述に留め、外部に提供される情報のみを明確化することに注力する。

（１）そのサービスがどのビジネスプロセス及びユースケースで利用されるかの紐付け。

（２）そのサービスが含まれる機能領域。

（３)サービスとして提供する機能概要と､そのサービスが提供する操作(オペレーション)がどのような物か。

（４)そのサービスが提供する操作を呼び出すための事前条件及び､事後条件(サービスの呼出結果としてシステムがどう変化していることが正しいか)。

４　まとめ

本論文では、事例を通してサービスを決定するまでのアプローチを、反省点を交えて述べた。今回の事例に関しては、結果からすると、サービスがユーザーの扱いやすい単位となっているかには疑問が残る。疎結合の観点からしても、サービス間の連携が密となっている箇所が多く見受けられる。しかしこの事例を通して、ビジネスプロセスから適切なサービスを切り出すアプローチ方法に関しては、ある程度確立できたと考える。

現在ＳＯＡの考え方については、未だ試行錯誤している段階である。また、ＳＯＡのメリットがサービスを再利用する段階にならないと享受できないという特性と相まって、導入後の効果を明確に実感することができていない。そのため、今回のプロジェクトにおいても焦点をどこまで広げるかは大きな議論となった。

しかし、ＳＯＡのメリットを最大限に受けるためには、より広範囲の業務の分析からサービスの粒度をじゅうぶんに吟味することが不可欠である。実装における導入範囲の大小は別として、サービスの括りを決める段階までは、可能な限り業務分析の範囲を広げることが望ましい。それが可能ではない場合でも、広い範囲の業務知識を持った人材がサービスの決定を行うアプローチに関わるべきである。

また、業務や機能をまとめるというとＳＯＡのイメージがつきにくいが，情報をある単位でまとめて、それに関わるすべての業務，機能を包括した括りをサービスとすると考えやすいと感じた。そのためには、旧来のＤＯＡ（Data Oriented Approach：データ指向アプローチ）やＯＯＡ（object oriented analysis: オブジェクト指向分析）といった手法のスキルも必須となるであろう。

５　むすび

ＳＯＡの技術的な基盤は成熟期を迎えたが､クラウドコンピューティングへの適用など、アーキテクチャとしての必要性は益々高まってきている。あらためて、ＳＯＡ導入の成功においては、ビジネスプロセスから適切なサービスを切り出すことが重要となることを強調したい。今回のこの事例が、ＳＯＡ導入の足がかりとしての一助になれば幸いである。