大規模組込システムの要求分析、システム方式設計、そして、ソフトウェア設計までをつなぐモデルベース設計手法

Model base design technique to perform seamlessly until the software design through the system architecture design from the requirements analysis of large-scale embedded systems.

藤原 啓一*
Keiichi Fujiwara

システム設計とは、システム要求を各開発フェーズごとに適した粒度に詳細化し、ソフトウェアで実現すべきこととハードウェアで実現すべきことに分ける作業である。本稿では、実際に開発現場で利用している、システム要求分析～ソフトウェア要求分析までシームレスに繋ぐ設計手法を示す。特に、方式設計を機能と非機能に分けて記述することで、システムアーキテクチャの本質である非機能の実現方式を浮き彫りにすることを特徴としている。

System design is to refine the system requirements to a granularity suitable for each development phase, and system architecture is work to divide system functions into hardware implementation and software implementation. This paper shows a design technique that connects seamlessly to the system requirements analysis - software requirements analysis using in the actual development site. In particular, it is characterized in that highlight the implementation method of a non-functional in the essence of system architecture by describing divided into functional and non-functional features.

1. まえがき

近年、組込ソフトウェア開発の規模は増大し、数百人が、これに従事することが珍しくなくなってきた。そして、具体的な開発ガイドラインを持たない組織は、「人月の神話」が説明するとおりの生産性低下と混乱をもたらす結果となっている。その解決策の一つは設計トレーサビリティ確保可能な設計プロセスの導入であるが、実現場では、要求をソースコードに直接紐付けた状態と変わらない程度の方法も多い。本稿では、組込システム開発に適用し、その設計詳細化を追跡できることが確認できた手法を紹介する。これは、IEEE1220をベースに大規模な情報系ソフトエア開発の設計ガイドライン（参考文献［1］）を入れ込み、組込ソフトウェア開発に適用できるようテーラリングしたものである。具体的手法としては、SysMLをベースに、ドメインモデルの活用、システム要求分析からシステム方式設計のつなぎにICONIXプロセスを取り入れることで、システム設計フェーズからソフトウェア開発までをシームレスにつなぐことができている。

2. 組込システム開発へのSysML適用の課題

SysMLは設計記法を提供してくれるが、作成図の利用方法には自由度があり、大規模システムに適用する際には、設計プロセスと利用図の関係性をSysML利用者が利用ガイドとしてまとめる必要がある。使用上決定すべき内容の例としては、要求図と要件記述している平文との関係性、要求図とユースケースの使い分け、作成図間のトレーサビリティの取り方等である。

そこで、本手法では、下記の点を詳細に定義した。

①要件抽出には、要求図とユースケースを、それぞれの役割を明確にして活用する。
②ドメイン分割は要求分析段階で定義するドメイン辞書の意味論の区別から行う。
③システム要求仕様と方式設計を結ぶ手法として、ICONIXプロセスによるオブジェクト指向分析を前提

＊関西事業部

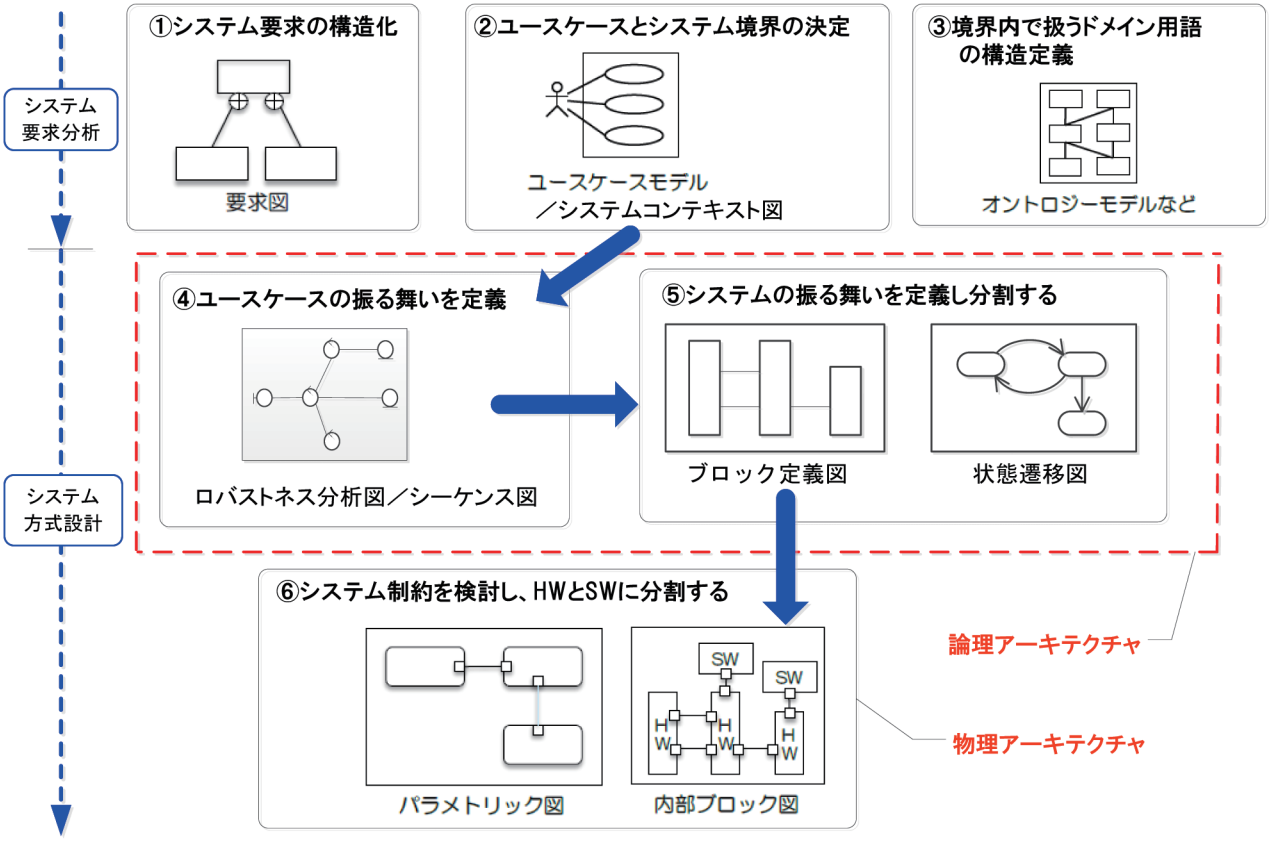


図１　本システム設計手法の概要手順

として、ユースケース駆動を利用する。

④サブシステム分割では、分割基準に従って、要求仕様から論理サブシステム定義をまず行い、論理サブシステムを元に物理サブシステムを作成する。

3. 本システム設計手法の概要

本設計手法の概要を図１に示す。要求図とユースケースで要件を表現し、要求から方式設計のつなぎには、ロバストネス分析図を利用するICONIX手法を用いる。

本設計手法・方法論の各開発フェーズにおいて用いる設計アプローチを表１にまとめる。

3.1　本設計手法の特徴

本設計手法は、次のような特徴を持つ。

(1)　設計要素間のトレーサビリティは設計手法の中に組み込まれている。

モデルベース開発において、上位から下位に進める際、形式的に決められる手順を可能な限り具体化した。上位の設計要素と下位の設計要素の粒度の違いを明確にするとともに、上位要素から下位要素への変換方法も可能な限り具体化した。

本設計手法を表現できるツール、設計書間の要素を結

表１　キーとなる設計アプローチ

開発フェーズ	キーとなる設計アプローチ
システム要求分析	要求図、ユースケース、USDM、ドメイン知識モデル
システム方式設計	ロバストネス分析によるサブシステム抽出、アーキテクチャ・スタイル

び付けられるツールを用いることで、わざわざトレーサビリティマトリクスを作成したり、フェーズ間にまたがって二重の表現を維持する必要がなくなるため、設計効率を上げることができる

(2)　要求から機能の抽出手順を明確化

構造化設計手法では、機能の定義が設計者によってばらつく。本設計手法では、要求分析から方式設計に至る過程を手法として形式化することで機能の抽出方法が明確になり、機能定義のばらつきが少なくなる。

(3)　ドメイン知識の定義とその活用方法を明示

ドメイン知識は設計を通じて利用する辞書の意味も持つが、それが論理設計の大きな分割の枠組みとなることを明確に示した。要求分析の段階で、静的な実世界をドメイン知識の定義でモデル化し、動的な実世界をユース

ケースでモデル化する。

⑷　要求仕様から論理アーキテクチャ設計を経由して物理アーキテクチャに至る

論理アーキテクチャと物理アーキテクチャの違いを明確にし、アーキテクチャ設計手順の考え方を整理した。このようにすることで、第三者のレビューが容易となる。

4. システム要求分析

システム要求仕様を表現する図法として、SysMLの①要求図と②ユースケース（とユースケース記述）、ならびに、③USDM（Universal Specification Describing Manner）、を利用する。

4.1　システム要求仕様の表現方法

要求図では、システムに関わる利害関係者が求める、システムが持つべき能力や満たすべき条件をツリー構造で記述する。それは、①目的、②戦略、③機能的振る舞い、④横断的関心事、⑤非機能要件から構成する（図２）。

①目的は、さらに上位の製品企画書等を参考に作成され、複数の目的があり、それぞれが互いに矛盾していて良い。③機能的振る舞いは、ユースケースの視点でUSDMを利用して振る舞いを中心に要求仕様を記述する（4.3章）。⑤非機能要件も、例えばSQuaREを記述構造のフレームとして、USDMの表形式で表現する。④横断的関心事はシステム要求仕様作成段階での共通仕様である。そして、この要求図にドメイン知識の定義（4.6章）を加えた物が、システム要求仕様となる。

要求図の作成には、ゴール指向を用いる。最初に達成するゴールを明確にし、そのゴールに達するためのサブゴールに分割する。分割にはAND分割、OR分割、Includeのいずれかを用いる。

4.2　設計対象が部品のユースケース表現

設計スコープは、我々が設計対象とするシステムの範囲（システム境界）を表す。設計対象のユースケースは、この設計スコープと適切に対応しなければならない。どの設計スコープにするかによって、ユースケースのアクター／目的が異なる（図３）。

開発対象が、サブシステムBの場合、その直接のアクターは、サブシステムAとなる。しかし、サブシステムBのユースケースを考える場合、それは、ドライバをアクターとするシステムユースケースが振る舞うシーンに対応させなければならない。なぜなら、サブシステムAとサブシステムBのシーンは、システムユースケースのシーンを、アクターを変えて記述しただけのものだからである。

図４に、電動パーキングブレーキを題材に具体例を示

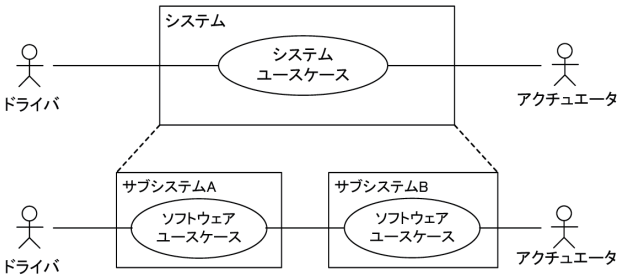


図３　システム境界とユースケースの関係

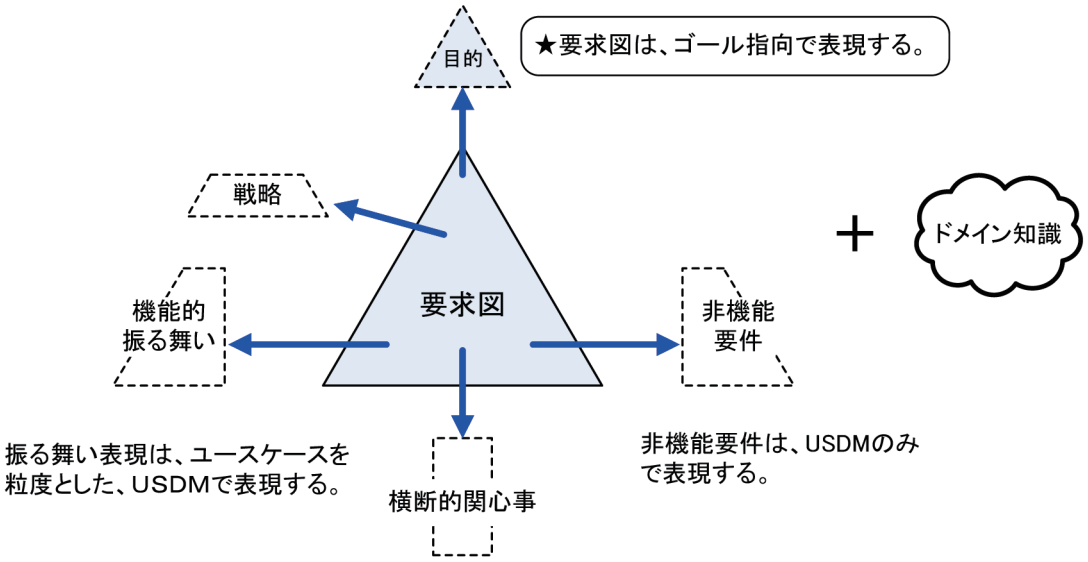


図２　要求図の構成　及び、要求仕様の構成

す。ドライバがパーキングブレーキを掛ける場合、手動でブレーキを掛ける／手動でブレーキを外す／アクセスを踏むと自動でブレーキを外す等のシーンが考えられる。

開発対象がブレーキを直接駆動するサブシステムのソフトウェア開発だとしても、ドライバから見たシステムのユースケースを考えることで、開発システムが対応しなければならないシーンをすべて抽出することができる。この時、準正常や例外は、ユースケースシナリオの拡張として記述することになる。

ユースケースの粒度は、各ユースケース間に順序が発生しないレベルで留める。

4.3　ユースケース単位でUSDMを使って機能仕様を記述する。

ユースケースは、アクターが、システムの中をブラックボックスとして、システムの外側から見えるシステムの振る舞いを表現したものである。USDMは、システム内部の振る舞い（例：データ変換仕様等）を表現したものである。ユースケースとUSDMの関係を図５に示す。ユースケースとUSDMを対応づけるルールは以下のとおりである。

・ユースケース名（ユースケース全体）をUSDMの上位要求に対応づける。
・各ステップはUSDMの下位要求あるいはグループに展開する。
・各ステップの実現方法を仕様に展開する。

ステップ毎の実現仕様としては、イベントのインタフェース仕様、ステップ実行直後に行うシステムの振る舞い、判断、データ変換処理等を考える。

4.4　非機能要件の表現

非機能要件は、USDMを用いて表現する。階層構造フレームとして、ISO25010（SQuaRE）を用いる。

4.5　要求仕様のプロダクトライン化

組込システムは、ほぼ同じだが仕向け先ごとに少しだけ仕様が異なるということが多い。したがって、仕様レベルで仕向け先間の違いを見渡せることが、アーキテクチャ設計を効率化させることにつながる。仕様の数は、数十仕様/KLの規模であり、100KL程度で数千仕様数になるので、プロダクトラインの可変点をフィーチャーツリーで表現するのは現実的では無い。そこで、USDMの仕様項目に可変性の欄を設けて、選択仕様を表現する（図６）。

ただし、制御系の場合、仕様が同じでも、前提条件が機種毎に異なることも多く、その部位（状態遷移、デシジョンテーブルでの表現）での可変性を明示できるように工夫することも必要となる。

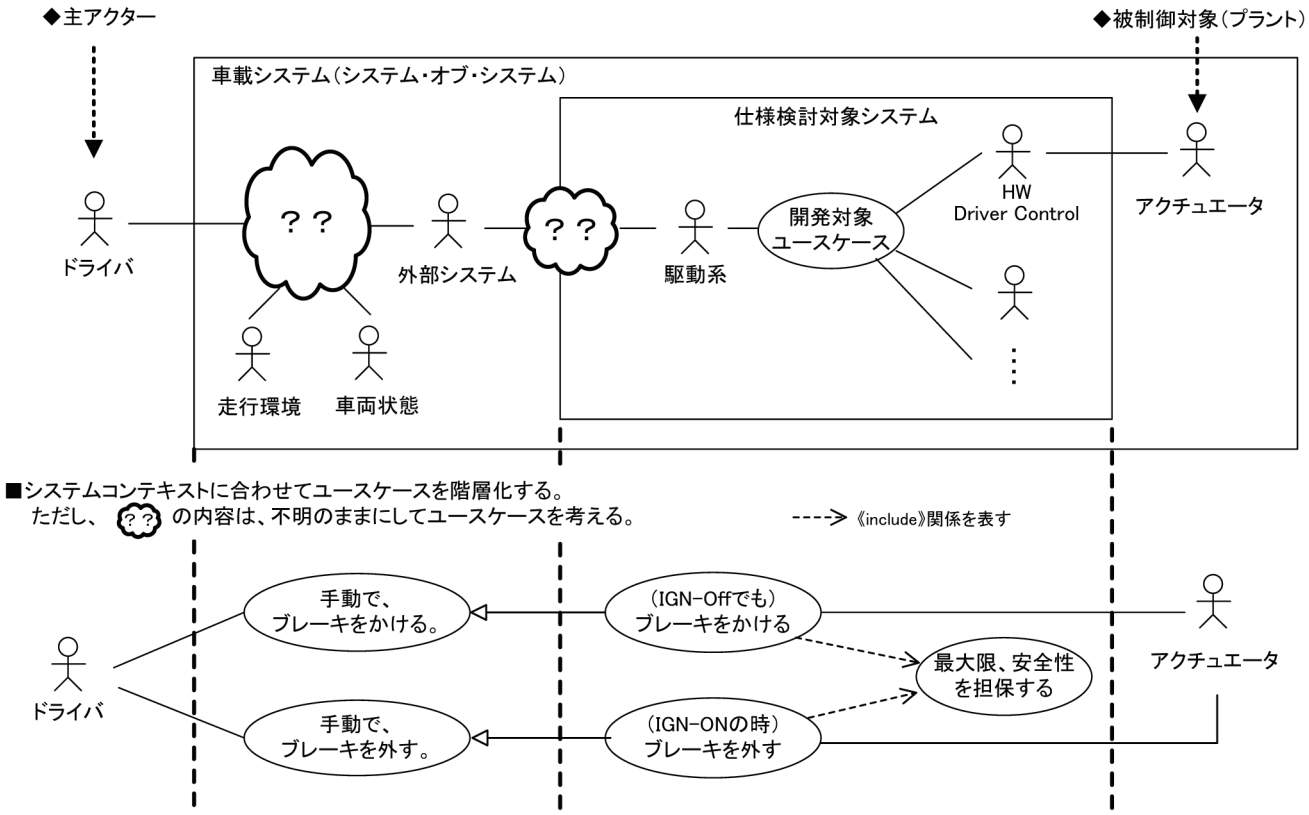


図４　ユースケースの例

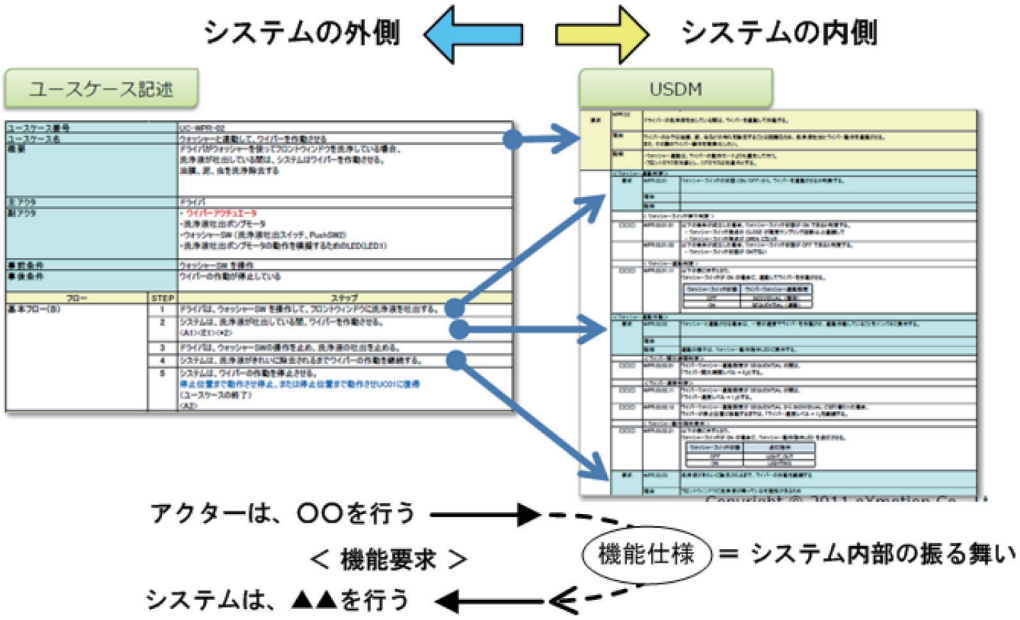


図５　ユースケースとUSDMの関係

				可変性			
					A社	B社	C社
要求	xxx			選択	●	●	●
	理由						
	説明						
	要求	01		必須	●	●	●
		理由					
		説明					
		仕様	001	代替1	●		
		仕様	002	代替2		●	
		仕様	003	代替3			●
		仕様	004	必須	●	●	●

図６　USDMによるプロダクトラインの可変性表現

4.6　ドメイン知識の定義

ドメインとはある特定の分野を示す言葉である。要件定義と並行して、ドメイン内で使用する物や概念を抽出して、その意味とそれらの間の関係性を構造(ドメイン構造図)として定義する。それらは当該システム開発におけるドメイン知識であり、個々の用語解説はドメイン辞書（用語集）となる。

ドメイン知識を整理するには、「もの」と「こと」の関係を分析する。「もの」は静的で時間に依存しない実態（エンティティ）を示し、「こと」は出来事が発生した時間を持つイベントを示す。そして、静的な「もの」の変化は「こと」を通じて行われる。

ドメイン構造図は、厳密にはオントロジーを用いて構築するのが良いが、実際の開発現場では、概念データモデル作成と同様の方法で良い。ドメイン構造図は、ドメインに存在する「もの」を４種類の関係、①汎化（is-a)、②集約(has-a)、③関連（assosication)、④依存（use）を用いてモデル化する。関連は「こと（イベント)」を表現するのに用いる。

ドメイン毎に作成するドメイン辞書は、例えば、次のような項目で構成する。

・カテゴリ：ドメイン知識として分類した大きな枠組み
・名称　　：用語の名称
・種類　　：アクター／リソース／属性
・意味　　：用語の内容
・値域　　：属性の場合、それが取り得る値の範囲。

この作業の結果を用いてドメイン分割（後述）を行ったり、開発全体で使用する用語を統一したりするので、非オブジェクト指向言語で開発する場合でも必要となる。

5. システム方式設計

システム方式設計の主たる目的は、システム要求をハードウェア／ソフトウェアのどちらで実現するかを決めることである。その実現の過程で、一段階詳細なサブシステム分割や、そのサブシステムを構成するコンポーネントを洗い出す作業が必要になるのであって、ソフトウェアの実装に向かって、どこまでも詳細化を進めていくのではないことに注意が必要である。

システム方式設計は、①ドメインの分割、②論理アーキテクチャの決定、③物理アーキテクチャの決定の順に行う。以降、それぞれを詳細に説明する。

5.1　ドメイン分割

前述の「ドメイン知識の構造定義」で範囲を区切ったドメイン単位にシステムを分割する。ドメイン分割は意味論の変化点の分割であり機能分割ではない。図７に電動パーキングブレーキを題材に機能分割とドメイン分割

の違いを示す。縦方向の分割が機能分割であり、横方向の分割が、ドメイン分割である。

ドメイン辞書に定義される「もの」「こと」を、取り扱う専門用語、知識としての取扱が異なるドメインに分割し、ドメイン単位毎に開発者を分けることで、開発に必要な知識を絞ることができ、効率よく開発が進めることができる。

例えば、アプリケーションドメイン開発者は、外部環境やハードウェア制約といった複雑なロジックを意識することなく開発ができるが、一方、ハードウェア制御ドメイン開発者は、ハードウェアの振る舞いを知り、その制御方法に関する知識を豊富に持たなければならない。ドメイン辞書における知識ドメインの分割境界は、保守性を高めるという意味で、方式設計においてレイヤ分割の一部となる（図８）。

ドメイン間の関連はブリッジで示す。ブリッジは、あるドメインの概念から、別のドメインの概念への対応を示す方法として記述する。

実開発では、ブリッジを明示的に記述する必要は少なく、インタフェース仕様書等で代用できることが多い。

		機能分割 ⇓		
		手動 Apply	手動 Release	自動 Release
ドメイン分割 ⇒	アプリケーション			
	ハードウェア制御			

図７　ドメイン分割と機能分割

表２　ブリッジの例

アプリケーションドメイン	ハードウェアドメイン
手動 Apply	xx API（Apply）
手動 Release	yy API（Release）

5.2　設計対象の階層レベル決め

大規模なシステムは、階層構造に分割しながら設計を進める。その構成要素の大きさを図９に示す。サブシステムはドメイン内に収まり、サブシステムがドメイン境界をまたがることは通常ない。サブシステムは、複数階層に渡って分割して良い。

5.3　機能要求から論理アーキテクチャ、非機能要求から物理アーキテクチャを決定する。

機能要求、非機能要求から物理アーキテクチャを作成する手順を図10に示す。

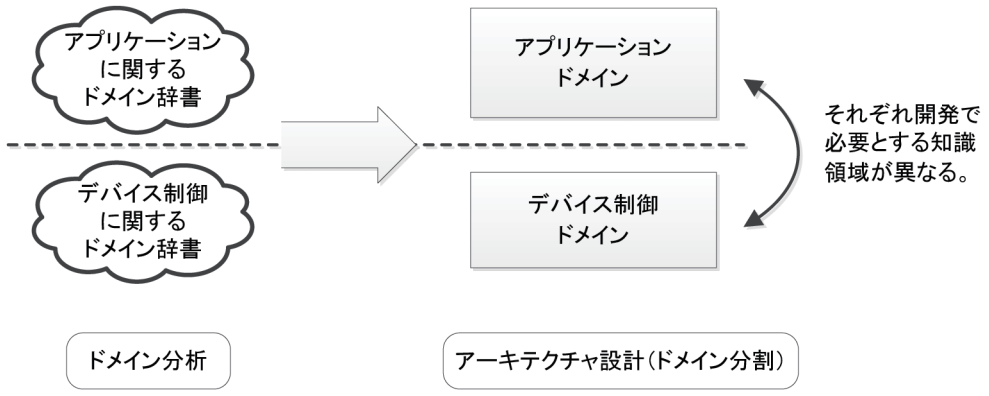


図８　ドメイン辞書とドメイン分割（アーキテクチャ設計）の関係

	レベル	論理（機能）分割	物理（実装）分割	
システム方式設計	0	システム		タスク ※分割単位のどれかをタスクに割り付けるという意味
	1	ドメイン	レイヤ	
	2	論理サブシステム	物理サブシステム	
ソフトウェア方式設計	3	論理コンポーネント	物理コンポーネント	
	4		クラス or 関数	
	5		クラス内のメソッド分割	
	6		メソッド内の内部設計	

図９　システム分割の階層構造

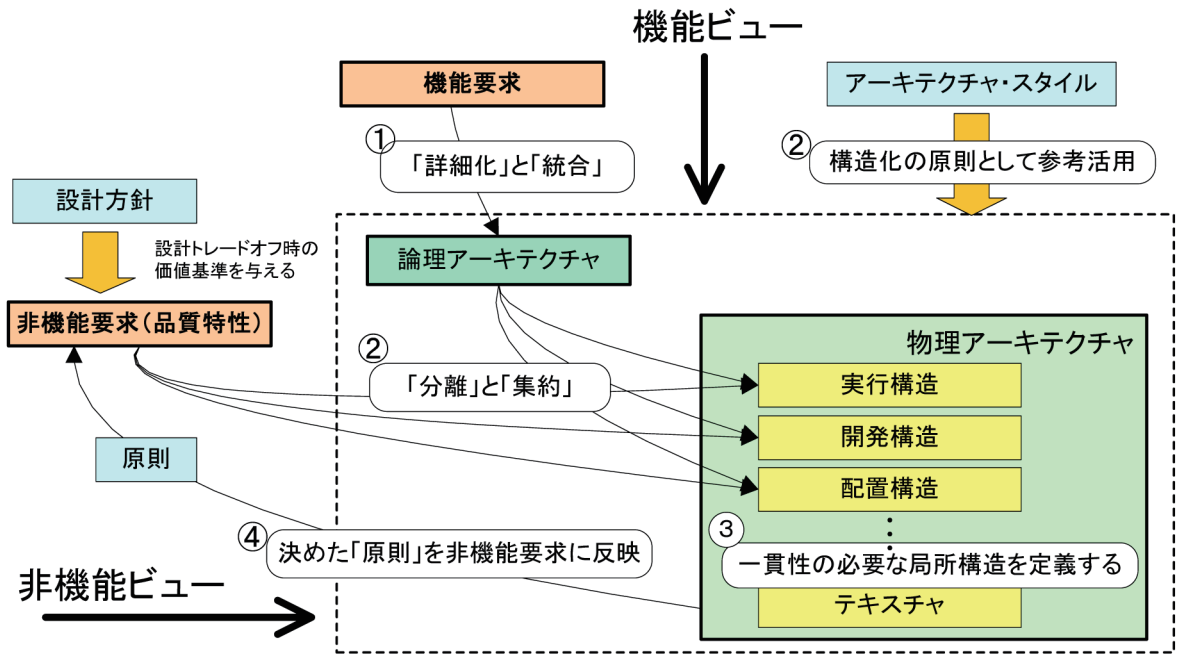


図10　論理アーキテクチャから物理アーキテクチャへの変換手順

主として機能要求から論理アーキテクチャ（サブシステム）を作り、論理アーキテクチャに対して非機能を考慮することで、物理アーキテクチャを作る。

①機能要求の実現として、論理的なアーキテクチャ（静的構造と動的構造）を表現する。この時、「静的構造要素の凝集度を高く、要素間の結合度は疎にする」基準のみを考え、実装上の制約を排除した形で論理アーキテクチャを表現する。

②論理アーキテクチャを踏まえて、設計方針や非機能要求を考慮して、実装上の制約を解決するための構造を、実行構造、開発構造、配置構造等に分けて表現する。この時、後述の品質特性間のトレードオフを考慮する。実装上の制約解決には、アーキテクチャ・スタイルを参考として利用する。

③一貫性の必要な局所構造をテクスチャとして定義する。

④以降の設計の為に、決めた原則（従うべき事項）を、非機能要求に反映して残す。

5.4　論理アーキテクチャを決定する。

ユースケースから論理サブシステムを抽出する流れを、図11に示す。

論理アーキテクチャ設計の主たる目的は、論理サブシステムを決定することである。それを決定する過程で論理サブシステムの振る舞いや、それを構成するコンポーネントの振る舞いを設計せざるを得ず、自然と振る舞いが決まる。システムを論理サブシステムに分割するには、ロバストネス分析を利用する。機能的振る舞いの具体的要求仕様は、USDMを使ってユースケース毎に作成されている（準正常、例外を含む）。従って、システムの要求仕様を元に、サブシステムを抽出する手順は、次のとおりとなる。

①ユースケース毎に、その仕様を見ながらロバストネス分析図を作成する（詳細化）。

②全ロバストネス分析図を串刺しに見て（視点変更）、類似のコントールを集める。集めたコントロール群をサブシステムとする（統合）。この時、「サブシステム内の凝集度を高く、サブシステム間の結合度は疎になるようにする」統合基準を用いる。サブシステムの仕様は、コントロールにつながるUSDMの仕様すべてである。

ユースケースは、複数の機能から構成されるサービスと見なせる。つまり、機能（とその集まりであるサブシステム）とユースケースは直交関係にあると見ることができる。

5.5　物理アーキテクチャを決定する

物理アーキテクチャ設計は、論理アーキテクチャ設計で明確にした論理サブシステム内の構成要素（論理コンポーネント：BCE）を、インフラ（CPU、ネットワーク、ストレージ/メモリ、OS）も一つの境界として意識し、非機能要件を満足するよう物理サブシステムに再構成する作業である。論理アーキテクチャから物理アーキテクチャに変換する手順を図12に示す。

①非機能要件の内、優先順位の高いものから、一つひとつを品質シナリオと見なして、その実装方法を検討する。実装方法は、アーキテクチャ・スタイルに紐付く、いくつかの実現手段の中から適切な手段を選択す

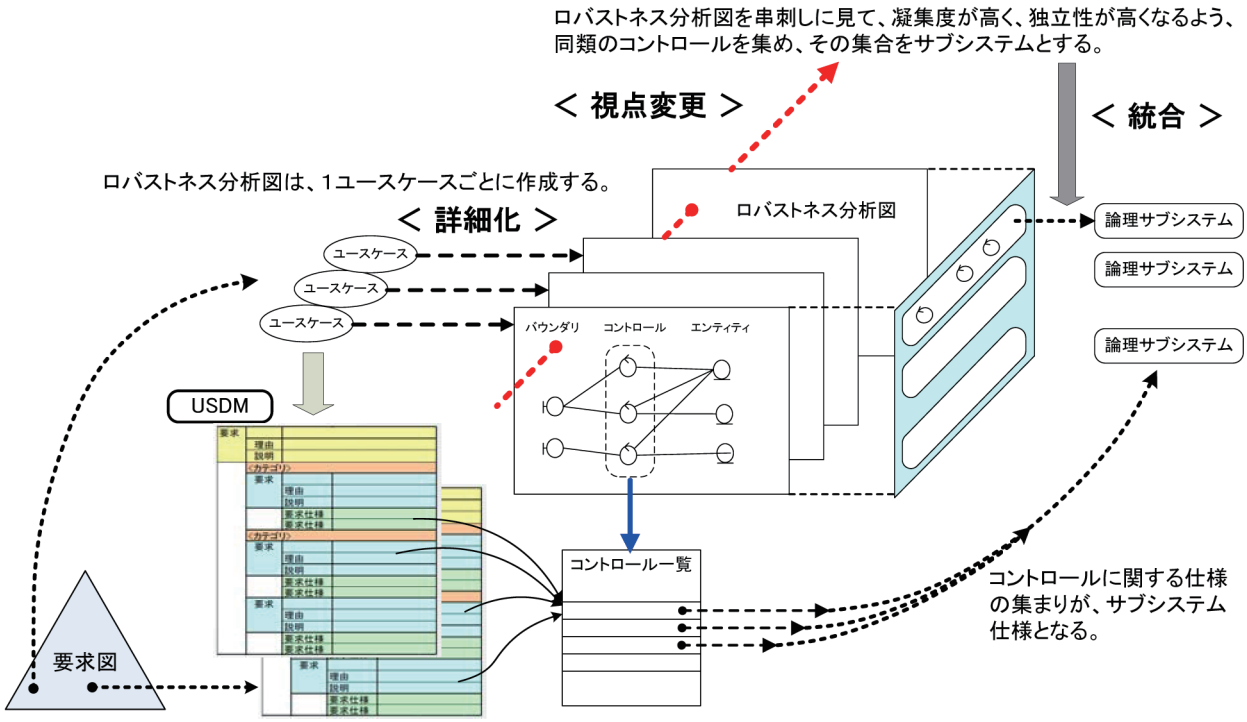
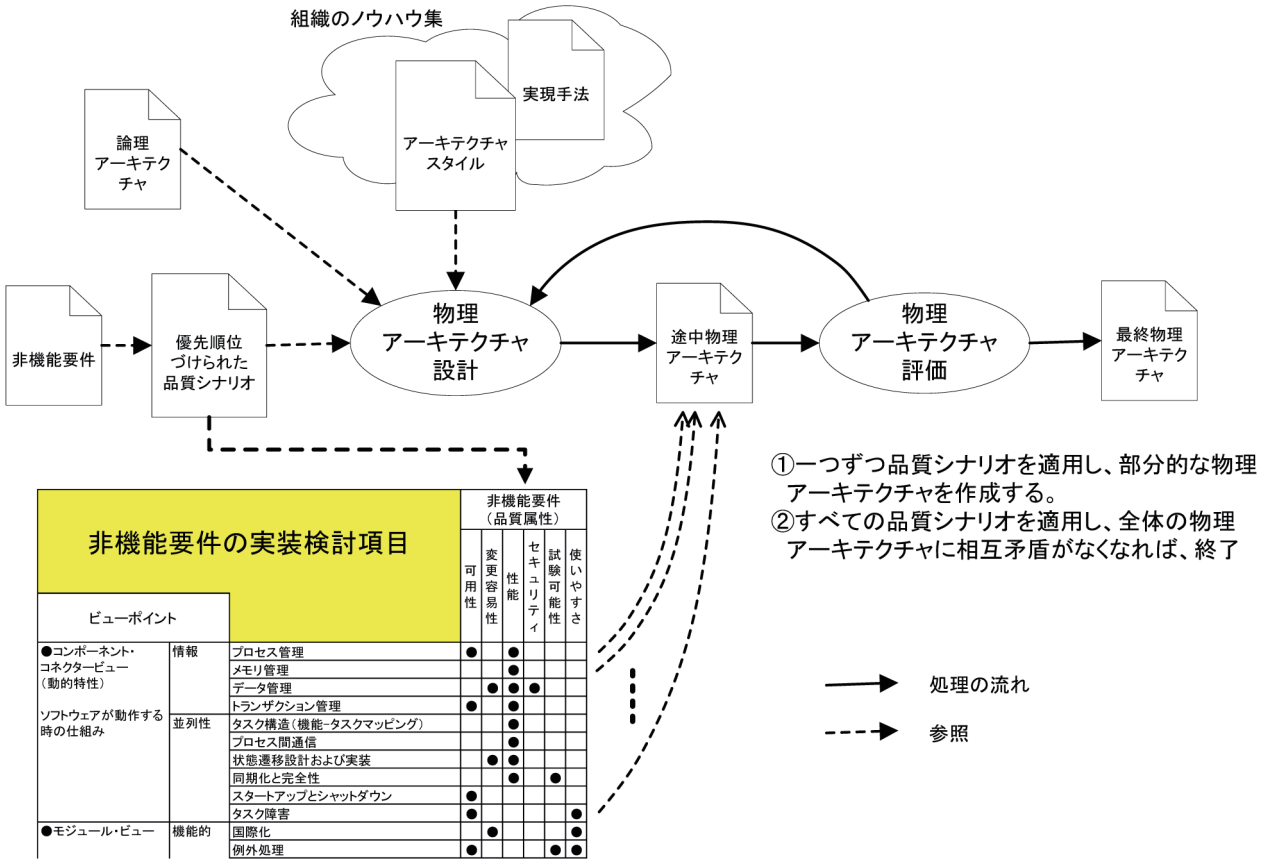


図11　ユースケースからサブシステムを抽出する



非機能要件の実装検討項目

ビューポイント			非機能要件（品質属性）					
			可用性	変更容易性	性能	セキュリティ	試験可能性	使いやすさ
●コンポーネント・コネクタービュー（動的特性） ソフトウェアが動作する時の仕組み	情報	プロセス管理	●		●			
		メモリ管理			●			
		データ管理		●	●	●		
		トランザクション管理	●		●			
	並列性	タスク構造（機能-タスクマッピング）			●			
		プロセス間通信			●			
		状態遷移設計および実装		●	●			
		同期化と完全性			●		●	
		スタートアップとシャットダウン	●					
		タスク障害	●					●
●モジュール・ビュー	機能的	国際化		●				●
		例外処理	●				●	●

図12　論理アーキテクチャから物理アーキテクチャに変換する手順

る。適用すべき実現手段がアーキテクチャ・スタイルに無い場合は、アーキテクトが生み出すことになる。

②ひとつの品質要求に対応する実装方法を検討する度に、それまでに検討できている実装方法と今適用しようとしている実装方法に矛盾があるかどうかを検討する。矛盾点がある場合は、優先順位の高い要求が残るよう実装方法間の矛盾を解決する。

③非機能要件に対応するすべての実装方法が検討でき、それらの間に矛盾が無いことが確認できたら、それを最終物理アーキテクチャとする。

品質要求のそれぞれは、独立ではなく相関関係がある為、検討の繰り返しにより、徐々に、実装方法と非機能要件に矛盾が無いように設計していくのである。

表３に品質特性間の相互関係を示す（参考［17］）。品質特性間には、ある品質要求を強く求めると、他方の要求を弱めなければ、実現方法が、どんどんと難しくなるという性質が存在する。例えば、保守性を高める為に、データアクセスへの隠蔽度を高めた方が良いが、データへのアクセス速度は遅くなる。

物理アーキテクチャを評価する毎に、品質特性の優先順位やシナリオが多少変換するので、それは記録し、非機能要求として残す。

5.6　論理アーキテクチャから物理アーキテクチャへの変換

物理サブシステムは、複雑に絡み合う非機能要求の実現性を設計することになるため、一旦、機能と結合性/凝集性制約だけから論理サブシステムを作り、その結果からの変更を行う方が設計根拠が明確になり、十分な設計レビューを行うことができるようになる。

物理サブシステムへの変換は、論理サブシステムとその構成要素を対象として、高凝集性と疎結合という原則を遵守しながら、非機能要求を満足するように、論理サブステムの境界とインタフェース、責務の変更を行う（図13）。

ここで、物理サブシステムとして再構成する論理サブシステムの構成要素は、ロバストネス分析によって分割されているBCEコンポーネントである。

論理アーキテクチャから物理アーキテクチャへの変換形態は、次の３種類がある。

①１対１：論理コンポーネントに割り当てられた責務が、１つの物理コンポーネントによって完全に満足されるならば、１つの論理コンポーネントは、単一の物理コンポーネントと同じになるので、論理サブシステムと物理サブシステムも同じになる。

表３　品質特性間の相互関係

	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪
①可用性（Availability）								＋		＋	
②性能効率性（Efficency）			－		－	－	－	－		－	－
③柔軟性（Flexibility）		－		－		＋	＋	＋			
④インテグリティ（Integrity）		－			－				－		－
⑤相互運用性（Interoperablity）		－	＋	－			＋				
⑥保守性（Maintainability）	＋	－	＋					＋			
⑦移植性（Portability）		－	＋		＋				＋		－
⑧信頼正（Reliability）	＋	－	＋			＋				＋	＋
⑨再利用性（Reusability）		－	＋	－	＋	＋	＋	－			
⑩堅牢性（Robustness）	＋	－						＋			＋
⑪使用性（Usability）		－								＋	

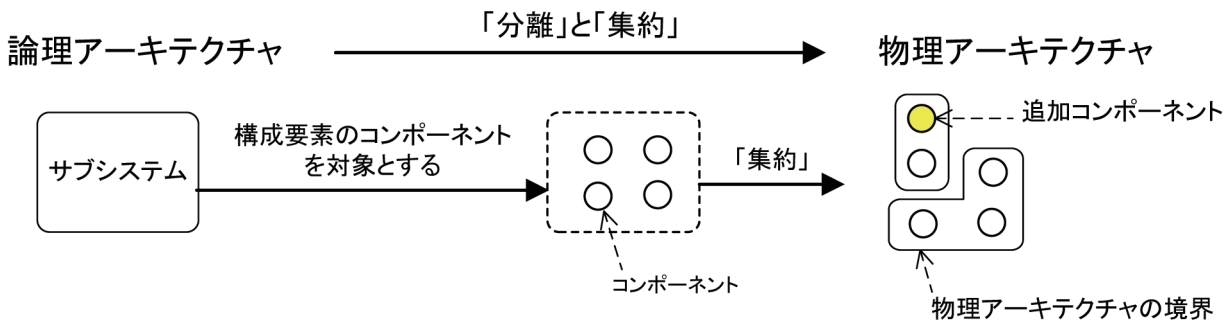


図13　論理アーキテクチャから物理アーキテクチャへの変換

②１対多（図14）：例えば、そのコンポーネントが１つの製品パッケージで完全に実現できず、部分的に製品を使い、部分的に開発しなければならないとすると、１つの論理コンポーネントは複数の物理コンポーネントによって実現されることになる。

③多対多（図15）：性能要求を満足するように２つの論理コンポーネントを単一の物理コンポーネントに一本化させた場合（例：タスクにまとめる）、複数の論理コンポーネントが単一の物理コンポーネントによって実現されることになる。

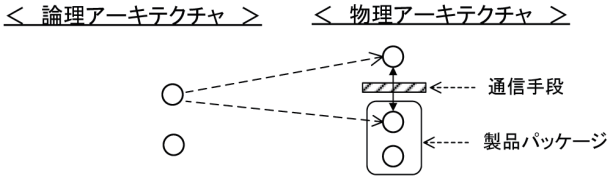


図14　論理から物理への変換が１対多の関係

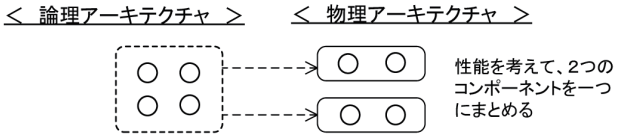


図15　論理から物理への変換が多対多の関係

5.7　アーキテクチャ・スタイル集

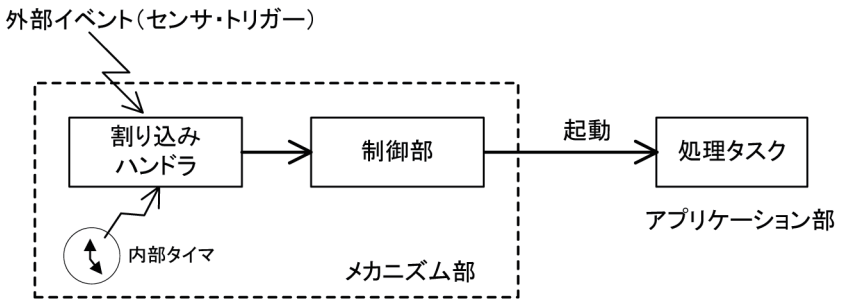
物理アーキテクチャの実現を効率的に考えるために、アーキテクチャ・スタイルを利用する。品質特性とそれを解決するアーキテクチャ・スタイルとの関係を表現する資料を組織として整備しておくことで、利用効率性を高めることができる。ただし、システム方式設計の目的は、ハードウェアで実施することとソフトウェアで実施することの境界を明確化することなので、システム設計レベルで行うアーキテクチャ構造とソフトウェア開発でおこなうべきアーキテクチャ構造とを区別しておくことが肝要である。アーキテクチャ・スタイルの例をいくつか示す。

⑴　割込アーキテクチャ・スタイル（図16）

例として、『フェールセーフにウォッチドッグを使わない場合、イベントが入らないことを検知してフェールセーフイベント処理タスクを起動する』スタイルを示す。

外部イベントが一定期間入らず、内部タイマのみが割り込んで来た場合、外部イベントが発生していないと判断する。それは、外部イベント発生元の故障と判断できるので、制御部にて、そのフェールセーフに対応する制御を実施する。イベント駆動型のアーキテクチャを採用する場合は、多くの検討事項があるので、予め検討項目と対策を一覧にしておく。

⑵　監視／制御・スタイル（図17、表４）



■イベント駆動型：　割込みをトリガーに処理を実施。

図16　割込アーキテクチャ・スタイル

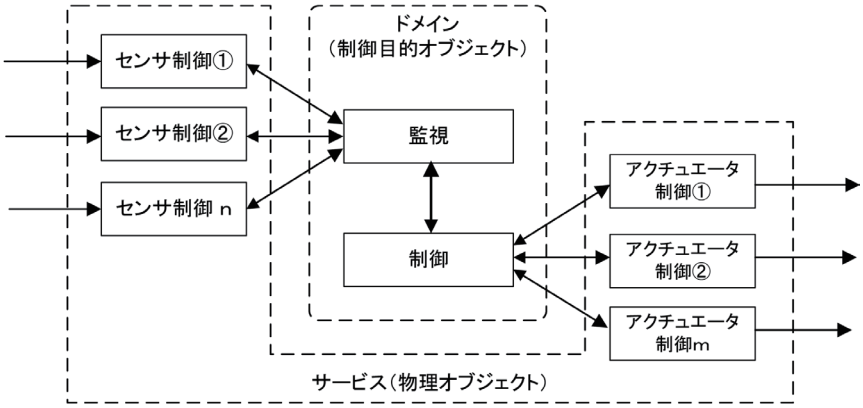


図17　監視／制御・スタイル

センサを監視していて、状況変化に応じて、アクチュエータを制御するスタイルである。これをドメインモデルに対応させると入出力部をサービス部、入出力部からのデータを加工する制御部分をドメイン部と見なせる。

サービス部がソフトウェア／ハードウェアの実現境界を分けている。

6. ソフトウェア要求分析

システムにサブシステムが一つの場合、システムユースケースとソフトウェアユースケースは同じになる。その場合でも、組込システムでは、必ず２つのドメイン（アプリケーション、ハードウェア制御）に分かれる。システムが複数のサブシステムに分割された場合を考える。システム方式設計段階で、サブシステムを構成する要素毎に、その動作タイミング、入力／振る舞い／出力（IPO）が設計されているので、それをサブシステムの単位で記述し直せば、ソフトウェア要素仕様となる。さらに、ソフトウェア実現として、例えば、利用するアルゴリズムを指定させたい場合は、それを仕様として追記する。

表４　構成要素の説明

名称	別称	意味	処理内容
ドメイン	制御目的 オブジェクト	全体制御（制御したいこと）を表現	制御の競合調整 状態遷移 データ間の加工処理
サービス	物理 オブジェクト	制御対象機器を表現	センサ制御（入力） アクチュエータ制御（出力）

7. システム設計書の構成とトレーサビリティ

設計書の構成を図18に示す。これら成果物間の関係が双方向に辿れるようにトレーサビリティマトリクスを作成する。

8. 設計品質の形式検証

有識者が、その設計根拠が正しいか、上位設計からの展開は十分か等をレビューすることで、設計の正しさを確認することになるが、ここでは、ある程度、形式的に検証する方法を説明する。

8.1　要求の網羅性と十分性の検証

上位と下位の設計書間のトレーサビリティの数の関係を「成果物項目間の多重度」と称する（表５）。多重度は、設計手法や組織での記述レベルが決まれば一定の範囲内に収まる。その範囲を超えてトレーサビリティが取られている場合は、設計内容が浅かったり、別の設計書に記述すべき内容が書かれていたりする設計の質の悪さと相関がある。このことを利用してシステム要求仕様書の網羅性と十分性の一次判断ができる。

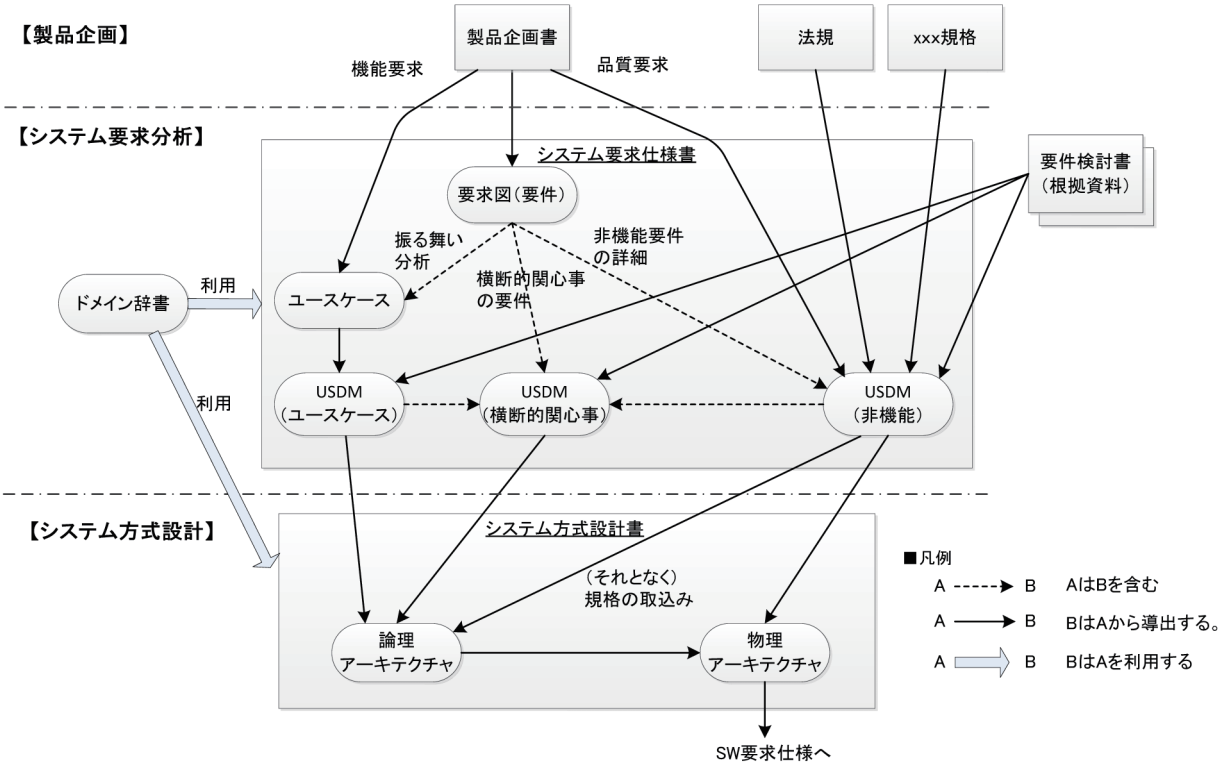


図18　設計書の構成とトレーサビリティ

表５　設計書間の多重度

	USDM仕様項目（数）			コントロール（数）			クラス（数）			プログラム規模（KL）		
	最小	最大	単位	最小	最大	単位	最小	最大	単位	最小	最大	単位
ユースケース（UC）	5.0	15.0	仕様/UC	5.0	10.0	個/UC	8.0	20.0	クラス/UC	1.0	2.0	KL/UC
USDM仕様項目				3.0	5.0	仕様/個	3.8	5.9	仕様/クラス	14.0	22.0	仕様/KL
コントロール（数）							1.0	3.0	個/クラス	88.0	178.0	Line/個
クラス										10.0	18.0	クラス/KL
メソッド										15.0	25.0	メソッド/KL

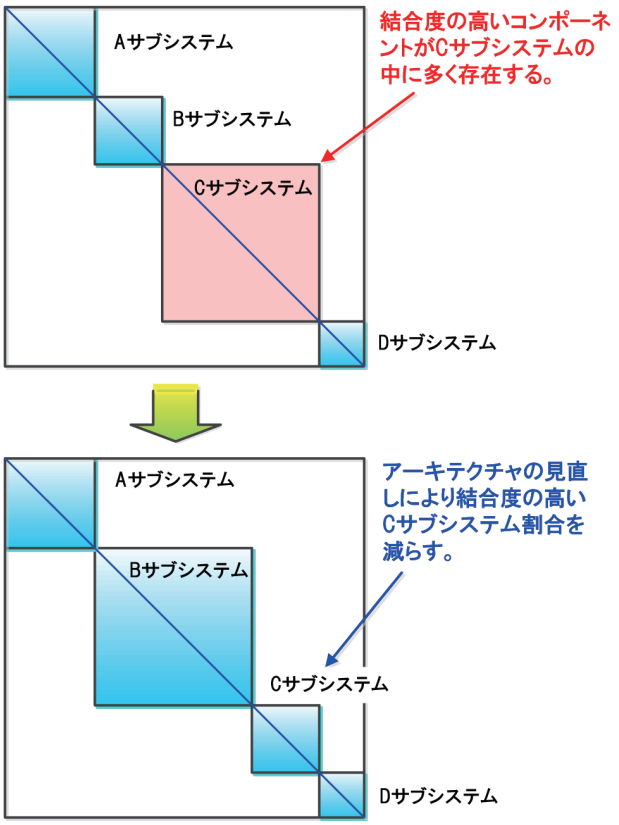


図19　DSMによるサブシステム構成の見直し

8.2　ドメイン知識による検証

ドメイン境界でブリッジを介して適切に用語の読み替えがなされており、ドメイン内ではドメイン辞書に定義されている用語のみで設計文章が記述されているかを確認する。例えば、ハードウェアの入出力は、アプリケーションドメイン側の用語とそのブリッジで表現され、ハードウェアが何をするかは、ハードウェアドメインの用語で記述されていることを確認する。また、ドメイン知識（用語辞書）を使って、そこに定義されていないシノニム（同義語）が設計書の中に使われていないことを確認する。これらはツールを利用することで、かなり自動化できる。

8.3　アーキテクチャの検証

DSM（Dependency Structure Matirx）を利用してソフトウェアアーキテクチャの分析・検証を行う。DSMはソフトウェアアーキテクチャの構造分析結果であるサブシステム間の依存関係をマトリクスで表現したものである。

論理／物理サブシステムを決定する際、結合度と凝集度を指標としてDSMで構造状態を見て、サブシステム間の結合度が保守性を著しく欠くものでないことを確認する。そして、問題ある場合は、サブシステムの構成を変更する。結合度は、バグを発生させる影響が最も強いものの一つだからである。

9. むすび

本設計手法は、一つ一つは新しくはないが、その組合せである全体的な設計の流れとつなぎ方は、有識者が暗黙的に行っていることを明示的にした一つの例である。組織全体で利用することで組織の設計能力は向上することは分かっているが、有識者にとってはまどろこしく、初心者にとってはギャップが大きく感じられ、設計根拠を第三者がトレースし易く管理したり、設計手順を全員が手抜かりなく実施するには、さらに細かな手順や方法において、個別疑問解決指導と牽引力が必要であり、導入敷居は高い。その敷居を少しでも下げ、多くのプロジェクトで導入が図られるよう、手法間をつなぐ方法のさらなる手順化や具体的適用例の開示、本手法で設計を進める際の支援ツールの開発を行う所存である。

参考文献

(1)　要求から詳細設計までをシームレスに行うアジャイル開発手法 藤原啓一 MSS技報（2014）
(2)　モデルに基づくシステムズエンジニアリング 西村秀和 日経BP社（2015）
(3)　SysML/UMLによるシステムエンジニアリング入門　ディム・ワイルキエンス　星雲社（2012）

⑷ 実践SysML　鈴木繁、山本義高　秀和システム（2013）
⑸ 組み込みUML　eUMLによるオブジェクト指向組み込みシステム開発 渡邊博之他 翔泳社（2002）
⑹ システムアーキテクチャ構築の実践手法 ピーターイールズ、ピーター・クリップス 翔泳社（2010）
⑺ 実践ソフトウェアアーキテクチャ Len Bass他 日刊工業新聞社（2005）
⑻ システムアーキテクチャ構築の原理 ニック・ロザンスキ他 翔泳社（2008）
⑼ リアルタイムUML第２版　ブルース・ダグラス,翔泳社（2001）
⑽ リアルタイムUMLワークショップ　ブルース・ダグラス　鈴木尚志訳,翔泳社（2009/12）
⑾ ソフトウェアアーキテクチャ　岸知二,野田夏子,深澤良彰,共立出版（2005/6）
⑿ UMLモデリング入門 児玉公信,日経BP社（2008）
⒀ 上流工程UMLモデリング 浅海智晴,日経BP社（2008）
⒁ 組込みソフトウェア開発はなぜうまくいかないのか　岩田宗之、日科技連（2007）
⒂ 【改訂第２版】要求を仕様化する　技術表現する技術　清水吉男,技術評論社（2010/6）
⒃ ユースケース駆動開発実践ガイド　ダグ・ローゼンバーグ、マットステファン、佐藤竜一他訳、翔泳社（2007/10）
⒄ IPA SEC編：要求工学・設計開発技術研究部会　非機能要求とアーキテクチャWG2006年度報告書、IPA（2007）
⒅ 組み込みソフトウェアの設計＆検証　藤倉俊幸、CQ出版（2006/9）
⒆ リアルタイム実現のための自律オブジェクト指向、岩橋正実 CQ出版（2002）

執筆者紹介

藤原 啓一

1985年入社。防衛のソフトウェア開発に従事以降 カーナビ、気象レーダ、ニューラルネット、医用画像、衛星通信、業務系システム、通信システム、車載制御システム等のシステム設計、ソフトウェア開発に従事。2015年より副事業部長、兼技術部長。