表形式テンプレートにより網羅性と一貫性を確保したITシステム基盤アーキテクチャ設計手法

大嶽 隆児

A Method to Design IT Infrastructure Architecture Using Tabular Templates to Satisfy Completeness and Consistency

Ryuhji Ohtake

様々な非機能要求を満足する高品質なITシステム基盤を設計するためには, 立場が異なるステークホルダーの関心事にフォーカスした複数のビューでアーキテクチャを記述するアプローチが有効である. しかし, モデル要素の定義やそれらの関係についてステークホルダー間で認識が異なっていると, 複数のビューで記述されたアーキテクチャのモデル間で整合性やトレーサビリティーを確保することが難しくなる. また, 多くのダイアグラムや文書フォームの間でモデル要素やその属性が重複して記述され不整合を起こしやすい. 本論文は, 網羅性と一貫性を満たす分かりやすいアーキテクチャを様々な観点で設計するために, オブジェクト指向のアプローチをITシステム基盤のアーキテクチャ記述に適用し, タイプに分類したモデル要素(コンポーネント・タイプ, ノード・タイプなど)とそれら関係を記述する正規化した表形式のテンプレートを活用した設計手法を提案する.

For designing a high-quality IT system infrastructure satisfying various NFRs (Non-Functional Requirements), it is an effective approach to describe its architecture in multiple views focusing on the concerns and standpoints of stakeholders in different positions. However, if the stakeholders' understanding of the definitions of model elements and their relationships are different, it is difficult to attain reliable traceability and consistency among architectural models described from various viewpoints. Moreover, the duplication of model elements and their attributes to various diagrams and textural documents may also cause inconsistency. This paper applies an object-oriented approach to describe the architecture of an IT system infrastructure, and proposes a design method using "typed" model elements (for example, Component Type and Node Type) and several normalized tabular templates describing relationships among them in order to design easy-to-understand architectures to satisfy completeness and consistency from various viewpoints.

Key Words & Phrases : アーキテクチャ, 非機能要求, ITシステム基盤, コンポーネントモデリング, オペレーショナル・モデリング
IT architecture, non-functional requirements, IT system infrastructure, component modeling, operational modeling

1. はじめに

オンデマンド・ビジネスに迅速に対応するためには, 多くの制約の下で様々な要求を満足する高品質なITシステム基盤をアプリケーション開発に先行して構築することが求められている. 文献[1]で紹介しているように, 品質と制約を定義する非機能要求(Non-functional Requirements, 以下NFRと略す)は, ITシステム基盤のアーキテクチャに大きく影響する. しかし, 多くのITシステムのプロジェクトでは, アプリケーションとITシステム基盤を概念, 表記法, 用語の意味が異なる手法で別々のチームが設計しているため, NFRを担当するチームがあいまいになりがちである. そのため, NFRの実現に必要な技術的機能の割り当て漏れやインタフェースの不整合が開発局面で発覚する, パフォーマンスやセキュリティーなどの重大な

提出日 : 2004年8月30日　再提出日 : 2005年9月29日

品質問題がテスト局面や本番稼動後に発生する，などのリスクがある．これらのリスクを低減する方法の一つとして，要求されるユースケース・シナリオの実現性をアーキテクチャの記述の段階で評価するアプローチが有効である．

文献[2]，[4]，[5]，[6]，[7]で紹介しているように，IBMでは，ITアーキテクチャの記述をArchitecture Description Standard（以下ADSと略す）[2]として標準化し，アプリケーションとシステム基盤のアーキテクチャを統合して設計する手法を導入している．ADSでは，機能的側面（Functional Aspect）をコンポーネント・モデリング（以下CMと略す）で，また，運用基盤的側面（Operational Aspect）をオペレーショナル・モデリング（以下OMと略す）でアーキテクチャを記述する．CMは，コンポーネントによりシステム機能の構造を定義し，OMは，コンポーネントを配置するロケーションやノードによりシステム構成の構造を定義している．特に，システム構成の構造は，NFRに大きく依存するため，コンポーネントを分散配置可能なプレゼンテーション，実行，データのデプロイメント・ユニット（以下DUと略す）としてNFRの特性でグループ化する概念を導入している．また，文献[3][8]で紹介しているように，複数のNFRを満足する品質の高いITシステム基盤を設計するためには，アプリケーション実行環境だけでなく，ステークホルダーの関心事にフォーカスした複数のビュー（例えば、セキュリティーや運用管理など）でアーキテクチャを記述することが必要になる．

実際に幾つかのプロジェクトでADSをベースにしたアーキテクチャの記述を試行してみると，次の課題が明らかになった．

（1）アーキテクチャを構成するモデル要素やそれらの関係について，ステークホルダー間で認識が異なっているため，複数のビューで記述されたアーキテクチャのモデル間で整合性やトレーサビリティーを確保することが難しい．

（2）一般的に基盤のアーキテクチャには，オブジェクト指向の分類や汎化（はんか）の手法が取り入れられていないため，多くのダイアグラムやフォームの間で，モデル要素やその属性の記述が重複して冗長的になり不整合を起こしやすい．

本論文は，網羅性と一貫性を満たす分かりやすいアーキテクチャを様々な観点で設計するために，オブジェクト指向の手法をITシステム基盤のアーキテクチャ記述に適用し，タイプに分類したモデル要素（コンポーネント・タイプ，ノード・タイプなど）とそれら関係を記述する正規化した表形式のテンプレートを活用した設計手法を提案している．以下，2章で，提案手法に適用するモデル化の概念や用語の意味をソフトウェア・エンジニアリングの分野の論文を引用して紹介し，3章で，提案手法について，ADSを紹介した上で本論文の新たな拡張について述べる．

2．アーキテクチャの記述に適用するモデル化の概念

ソフトウェア・エンジニアリングの分野では，文献[3]，[10]，[11]，[12]のように，様々なモデル化の概念が紹介されている．この章では，これらを参照して，3章の表形式テンプレートによるIT基盤アーキテクチャ設計手法で使用するモデル化の概念や用語の意味を整理して紹介する．なお，文献[9]のUML2.0を参考にしてダイアグラムを記述した．

2.1 区画（Partition/Segmentation）

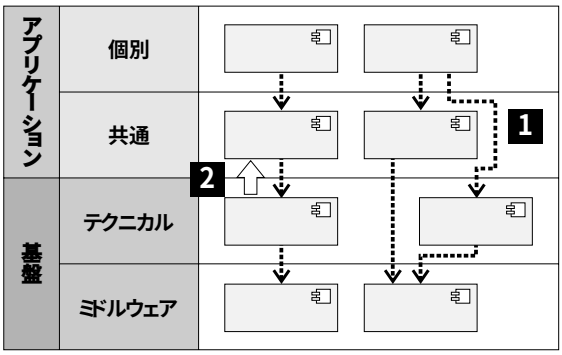
- システムを同じ抽象レベルで区画に分割し，各区画は，対等な関係で独立している．または，依存関係の低い疎結合のインタフェースで互いにサービスを提供する．
- 同一のモデル要素は，異なる区画に所属できない．

（適用例）

― システムと外部システムのスコープの境界を確定する．
― 分割したサブプロジェクトに合わせてサブシステムを定義する．
― 同じレイヤー内で独立性の高いサブシステムやコンポーネントに分割する．

2.2 レイヤー

- システムを異なる抽象レベル（Abstraction Level）で階層的（Hierarchical）に分割し，下位層が上位層にインタフェースによりサービスを提供する．
- 上位層が下位層を利用（Uses）する片方向の依存関係があり，上位層より下位層が汎化（はんか）されている．
- 上位層は，直下層だけでなく，すべての下位層を



出典：Software Architecture Documentation in Practice: Documenting Architectural Layers（[3] p.11 - 22）を参照して著者により作図

図1．レイヤー

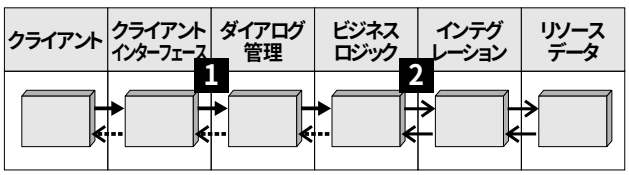
利用してもよい(Relaxed Layer). (**図1の番号1**)
- 下位層は, 上位層を呼び出し(Calls)できるが, 利用せず依存しない. (**図1の番号2**)
- 同じモデル要素は, 異なるレイヤーに所属できない.

(適用例)
- ― アプリケーションと基盤をレイヤーに分離する.
- ― 個別アプリケーションとフレームワーク/共通ライブラリをレイヤーに分離する.

注)本論文では, 文献[3]のレイヤーとティアーの定義に準拠している.

2.3 ティアー

- システムを分散環境で序列的に分割して, プロトコルにより通信する.
- ティアーは, プロセス, コンテナ, 論理ノード, または, 物理ノードとしてモデル化され, 内部にソフトウェア実行環境のレイヤーが存在する.
- クライアント・サーバーでは, レイヤーと同様にクライアントがサーバーを片方向で利用する依存関係がある. (**図2の番号1**)
- ピア・トゥ・ピアでは, 区画と同様に互いに独立した両方向の低い疎結合の依存関係がある. (**図2の番号2**)



出典:Software Architecture Documentation in Practice: Documenting Architectural Layers ([3] p. 23)を参照して著者により作図

図2. ティアー

(適用例)
- ― J2EEデザインパターン(クライアント, プレゼンテーション, ビジネス, インテグレーション, リソース)などを分散実行環境に分割する[13].
- ― セキュリティ・ポリシーが異なるネットワークを信頼度が低い方から高い方を保護するようにゾーンに分割する.

2.4 分類(Classification)と汎化(Generalization)

- 個々のモデル要素の特性を汎化したタイプに分類する.
- 汎化されたタイプから特化したサブタイプを階層的に定義できる.
- タイプから継承して具体的なインスタンスを定義する.
- 異なる関心事を別系統で分類した複数のタイプから多重継承できる.

(適用例)
- ― 個別業務画面を共通する特性で分類・汎化したリッチクライアントとしてコンポーネント・タイプを定義する. (**図3の番号1**)
- ― リッチクライアントのサブタイプを定義する. (**図3の番号2**)
- ― 別系統で分類した複数のタイプから多重継承してサブタイプを定義する. (**図3の番号3**)
- ― サブタイプを継承してインスタンスとして個別業務画面を定義する. (**図3の番号4**)

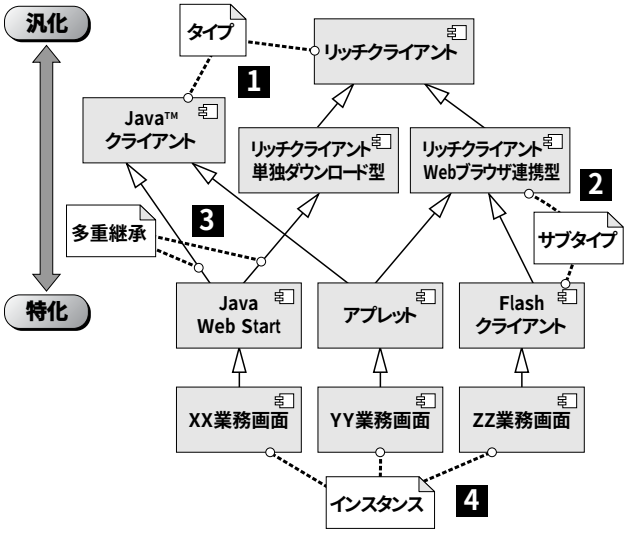


図3. 分類(Classification)と汎化(Generalization)

2.5 集約(Aggregation)

- モデル要素を階層的に複合構造(入れ子)で上位のモデル要素に組み込んで抽象化する.
- 合成集約(Composite Aggregation)では, 同じモデル要素を複数のモデル要素に組み込むことができない.
- 共有集約(Shared Aggregation)では, 同じモデル要素を異なるモデル要素に参照により含めることができる.

(適用例)

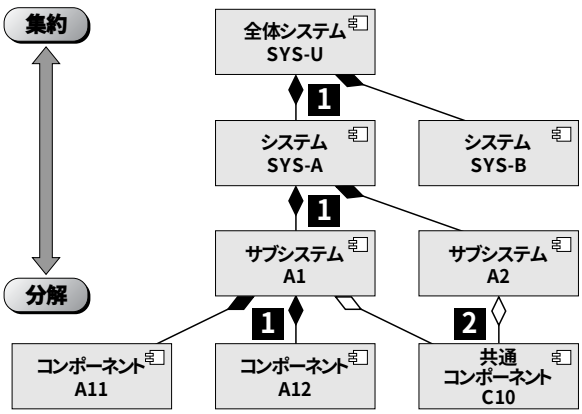


図4. 集約(Aggregation)

— 分割された機能を階層的にコンポーネント，サブシステム，システム，全体システムへ合成集約で定義する．（**図4の番号1**）
— 共通コンポーネントを複数のサブシステムへ共有集約で定義する．（**図4の番号2**）

3．表形式テンプレートによるITシステム基盤アーキテクチャ設計手法

この章では，オフィス製品で容易に作成できる表形式のテンプレートを活用したIT基盤アーキテクチャ設計手法について，ADSを紹介した上で本論文の新たな拡張について述べる．なお，各図・表の例は，抽象化した単純なモデルで表現している．

3.1 システム・コンテキストとシステム・インタフェース

システムの構造を定義するためには，最初にシステムのスコープを定義する．ADSでは，プロジェクトのスコープで定義したシステム（**図5のSYS-A**）を中心にして，システム・コンテキスト（**図5の内側の長方形**）を記述する．しかし，プロジェクトのスコープがビジネスのスコープの一部分であることが多く，ビジネスを実現するためには，全体システム（**図5のSYS-U**）のシステム・コンテキスト（**図5の外側の長方形**）に拡大してアーキテクチャをとらえる必要がある．提案手法では，ビジネスのスコープで図5に示すシステム・コンテキスト図を記述している．

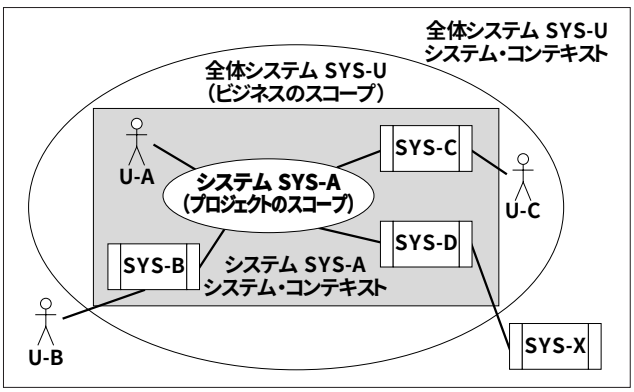


図5．システム・コンテキスト図

提案依頼書や計画書の段階では，ビジネスのスコープでシステムの構造が明確に定義されず，要件定義や外部設計でスコープが変更になることも多い．ビジネスのスコープで，ビジネスの実現に必要な複数のシステムから構成されるシステム（System of Systems）[14]としてとらえ，下位システム間での要求の漏れや重複を上位システム（Superordinate System）[15]のスコープ（図5のSYS-U）で調整し，システム・コンテキストを継続的に保守する必要がある．

また，スコープだけでなく，他のシステムやアクターとのインタフェースを明確にすることも重要になり，提案手法では，システム・コンテキスト図を補完するために，システム・インタフェース表（**表1**）に整理している．

表1．システム・インタフェース表

全体システムSYS-U レベル0					
システム		機能		インタフェース	
レベル1		レベル2		レベル3	
From	To	From	To	SYS-A	タイプ
U-A	SYS-A	機能A0	機能A1	機能A11	HTTP
SYS-A	SYS-C	機能A2	機能C1	機能A23	CICS®
	SYS-D	機能A4	機能D1	機能A42	MOM
SYS-B	SYS-A	機能B2	機能A3	機能A31	RMI
省略					

From：サービス利用/要求側　To：サービス提供側
注）ユーザーU-AがシステムSYS-Aにアクセスするクライアント端末を機能A0として，システムSYS-Aのスコープに含めている．このクライアント端末を他のシステムと共通化する場合は，独立したシステムとして定義することもできる．

システム・コンテキスト図では，対象のシステムがブラックボックスで表記されシステム内部の構造が明確になっていないが，システム機能の構造を定義する過程で，文献[16]で紹介されているように，サブシステム（表1のレベル2）の内部にインタフェースの責務を実装するファサードのコンポーネントを明確に定義し，表1のレベル3機能に記入して網羅性を確認する．この段階のインタフェースは，個々のデータ項目の定義ではなく，インタフェース技術方式を記述する．また，特に外部システムの制約でインタフェースが変更できない場合には製品レベルで記述する．

3.2 要求-機能対応

システム機能の構造を定義するアプローチには，トップダウンとボトムアップがある．プロジェクト計画の段階では，まず，トップダウンのアプローチにより，前提・制約になっているプロジェクトの体制も考慮して，並行開発やテストが実施し易いサブプロジェクトに合わせてシステム機能の構造を分割する．次に，ボトムアップのアプローチにより，全体システムのスコープで他のシステムとインタフェース（表1）とシステム・コンテキストのスコープを調整し，要求の構造から機能の構造へ洗練する．文献[17]では，トップダウンとボトムアップを組み合わせたミドルアップのアプローチと呼んでいる．

文献[5]のCMの手法で紹介しているように，ADSでは，機能の責務を割り当てるコンポーネントを識別し，分割（2章2.1節の区画），構造化（2章2.5節の集約），階層化（2章2.2節のレイヤー）のプロセスを反復してシステムの機能構造を洗練している．しかし，ダイ

アグラムだけでは，本来の要求と定義された機能の対応関係の網羅性やトレーサビリティーを確保しにくいため，提案手法では，分割と構造化のプロセスに，スプレッドシートなどで容易に管理できる要求-機能対応表（**表2**）を導入している．要求の構造から機能の構造に洗練する過程（表2の例，集約，分割，昇格，移管など）を記録することによりトレーサビリティーを確保しやすくなる．

要求と機能の関連は，プロジェクトの全局面でトレーサビリティーを管理するべきであるが，特に提案依頼書に対する提案書や計画書に対する要件定義書を作成する場合には必須になることが多い．

表2.　要求-機能対応表

全体システムSYS-U　レベル0				
システム機能			要求	
レベル1	レベル2	レベル3	レベル2	レベル3
SYS-A	機能A1	機能A11	要求A1	要求A11
		機能A12＊1	要求A1	要求A12＊1 要求A13＊1
省略				
	機能A2		要求A2	（要求A21）＊2
		機能A22＊3		要求A22＊3
省略				
	機能A3＊2	機能A31		要求A21＊2
		機能A32＊4		要求B12＊4
省略				
SYS-B	機能B1		要求B1	要求B11
				（要求B12）＊4
	機能B2		要求B2	要求B21
省略				

＊1　要求A12と要求A13を機能A12に集約
＊2　要求A21を上位レベルの機能A3に昇格
＊3　要求A22を機能A22と機能A23に分割
＊4　SYS-Bから要求B12をSYS-Aの機能A3に移管

3.3　ユースケース・タイプ

オブジェクト指向による開発も一般的になり，ユースケースにより要求を記述するプロジェクトも多くなっている．文献［7］で紹介しているように，アーキテクチャを設計する段階では，基盤の観点から処理パターンにまとめるアプローチが有効である．提案手法では，プロジェクト・スコープのシステムSYS-Aをホワイト・ボックス化したレベル2システム・コンテキスト図の上に処理フロー（**図6**の破線）を図示し，アクターとのインタフェース，処理の実行方式，連携方式，NFRなどの特性で分類・汎化したユースケース・タイプとして**表3**に整理している．

ユースケース・タイプの追加・変更は，基盤のアーキテクチャに大きく影響するため継続して保守する必要があり，表形式にすることで網羅性を確認しやすく保守も容易になる．

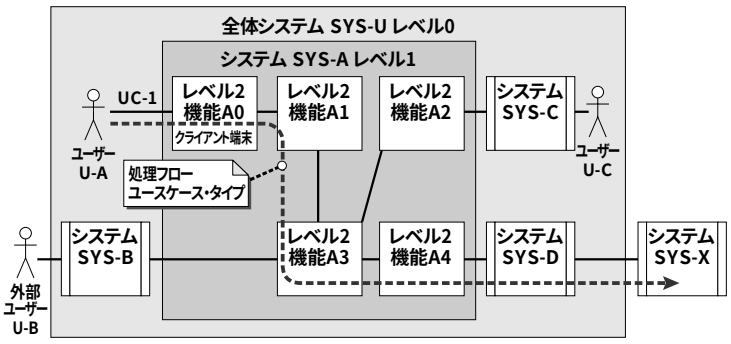


図6.　レベル2システム・コンテキスト図

表3.　ユースケース・タイプ表

アクター	ID	ユースケースタイプ	..	連携	NFR
U-A	UC-1	外部連携照会	..	SYS-D 同期	応答時間：5秒 利用時間：24x7
……	……	……	..	…….	……

3.4　コンポーネント・タイプ

文献［2］，［5］のCMの手法で紹介しているように，ADSでは，高凝集性（Cohesion）と疎結合性（Weak Coupling）を考慮しながら，CMの分割や構造化の手法でコンポーネントによりシステム機能の構造を定義する．さらに，CMの階層化の手法でアプリケーションのコンポーネントからNFRを実現する技術的な機能を基盤のコンポーネントに分離してレイヤー（図1）を定義している．また，文献［2］，［6］，［7］，［8］のOMの手法で紹介しているように，ティアー（図2）に分散配置可能なプレゼンテーション，実行，データのDUとしてコンポーネントをノードに配置してシステム構成を定義している．

しかし，コンポーネントをDUにグループ化するアプローチを実際のプロジェクトで試みると，CMとOMが別々のチームにより行われているため，ステークホルダー間でコンポーネントとDUの関連に対する認識が異なっていることが明らかになった．

両者間の認識を一致させるため，提案手法では，DUをNFRの特性で分類・汎化したコンポーネント・タイプと定義するアプローチを導入している．分割・構造化で定義した表2の機能（表2では，レベル3）をティアーに分散配置可能なプレゼンテーション，実行，データ別に分離し，共通する属性をコンポーネント・タイプ記述書（**表4**）にまとめ，表2の機能をティアーとレイヤーに階層化した表5を作成する．**表5**の個別アプリケーションに表4のコンポーネント・タイプを関連づけ，関連するNFRを実現する技術的

表4.　コンポーネント・タイプ（DU）記述書

実行DU　AE-A3	
コンポーネント・タイプ	トランザクションA3型
処理方式	ステートレス・セッション
応答時間	5秒以内
スループット	200 TPS
障害停止許容時間	1分以内
以下省略	

表5.　コンポーネント・タイプ一覧表

レベル2　機能A3					
レベル3機能	T層	L層	調達	ID	コンポーネント・タイプ（DU）
A31	U	A	開発	AU-1	RMIファサードA3型
A32	E	A	開発	AE-1	トランザクションA3型
	E	C	開発	CE-1	共通ライブラリL1型
	E	T	再利用	TE-1	EJBフレームワークF1型
	E	M	製品	ME-11	WAS-AP1型
省略					

T層：ティアー（Tier）：分散配置の階層構造
　U：プレゼンテーション（アクターとのインタフェース）
　E：実行プロセス
　D：データストア
L層：レイヤー（Layer）：ソフトウェアの階層構造
　A：個別アプリケーション
　C：共通アプリケーション
　T：テクニカル・コンポーネント
　M：ミドルウェア

な機能を割り当てるレイヤーと調達方法を定義する．

このアプローチでは，CMによるコンポーネントの抽出と，OMによるDUの定義を統合して行うことができるので，アプリケーションとシステム基盤の間でNFRの責務の割り当てが明確になり，また，網羅性とトレーサビリティーも確保しやすくなる．

3.5　基盤パターン

文献[2]，[6]，[7]，[8]のOMの手法で紹介しているように，地理的に離れたロケーションやセキュリティ・ポリシーや運用管理主体が異なるゾーンとして配置場所の制約をモデル化し，DUに割り当てられたNFRの特性を満足するロケーション／ゾーンにノードを定義してシステム構成の構造を設計する．

しかし，大規模なシステムでは，すべてのDUを一つのOMとして設計するとモデル要素が多く複雑になるため，提案手法では，ユースケース・タイプ別に基盤パターン表（**表6**）にまとめ，表4のコンポーネント・タイプ（DU）をロケーション／ゾーンとノードにマッピングするアプローチを導入している．

ユースケース・タイプの種類が多い場合には，共通部分，特殊部分にユースケース・タイプを分離してから基盤パターン表にまとめると重複する記述が少なくなる．また，基盤パターンを決める上で，IBM Patterns for e-business［18］のアクセス統合やアプリケーション統合などの参照アーキテクチャを参考にするとアーキテクチャの長所や考慮点を整理しやすい．

表6.　基盤パターン表

ユースケース・タイプ：　UC-1 外部連携照会				
ロケーション		本社		
ゾーン		イントラネット	ハイセキュア	
ノードタイプ		APサーバーA型	EAIサーバーC型	DBサーバーA型
アプリケーション	A	AU-1 RMIファサードA3型 AE-1トランザクションA3型	なし	AD-1 Web業務用RDB型
	C	CE-1 共通ライブラリL1型	CE-2 電文変換ユーティリティー	なし
システム基盤	T	TE-1 EJBフレームワーク	TE-2 MQ-CICSフレームワーク	なし
	M	ME-11 WAS-AP型 ME-21 MQ-CL型 ME-31 DB2-CL型	ME-21 MQ-MG型 ME-41 CICS-CL型	ME-32 DB2-MG型
	P	PE-1 Linux-WAS型	PE-A2 AIX-MQ型	PE-A3 AIX-DB2型

レイヤー（Layer）：　コンポーネント・タイプ（DU）
アプリケーション　A：個別アプリケーション・コンポーネント
　C：共通アプリケーション・コンポーネント
システム基盤　T：テクニカル・コンポーネント
　M：ミドルウェア
　P：プラットフォーム

文献[2]，[6]，[7]，[8]で紹介しているように，ADSでは，最初にアプリケーションの実行に必要な技術非依存の概念モデルを作成し，次に技術仕様を選択した仕様モデルにより，NFRの実現に必要なテクニカル・コンポーネントやノードを追加して製品に依存しないアーキテクチャを定義する．最後に技術仕様を実装する具体的な製品を選択する物理モデルに変換するプロセスを採用している．

このプロセスは，プロジェクト計画や要件定義など異なる局面で繰り返し行われるため，網羅性とトレーサビリティーの確保が重要になる．基盤パターン表に整理することで，必要な機能を割り当てるレイヤーとティアーが明確になり，機能の漏れやプロジェクト体制のチーム編成とのギャップを見つけやすくなる．

3.6　NFRドメイン別のノード・タイプ

文献[3]，[8]で紹介しているように，すべてのNFRに必要なDUやノードを1つのモデルで表記すると複雑になり判読性が低下する．そのため，アプリケーション実行環境のドメインで分類したタイプだけではなく，ステークホルダーの関心事にフォーカスした特定のNFRのドメインで抽象クラスとして分類・汎化（はんか）したノード・タイプを定義して（**図7**），異なるビューで簡潔なアーキテクチャ記述を作成するアプローチが有効である．

提案手法では，アプリケーション実行環境の基盤パターン表にNFRドメイン別の基盤パターン表（表6）

を追加して，多次元のビューでアーキテクチャを記述するアプローチを導入している．

3.7 モデル・ノード

大規模なプロジェクトでは，ノード数が多く導入サーバーを個別に設計すると，共通属性が重複して記述され作業効率や保守性が悪くなる．

提案手法では，モデル・ノードを継承するインスタンスとして個別の導入サーバーを定義するアプローチを導入している．モデル・ノードは，アプリケーション実行環境ドメインだけでなく，NFRドメインで定義した複数のノード・タイプを多重継承させ（図7），正規化して表現したモデル・ノード表（**表7**）としてまとめる．詳細な技術仕様，製品仕様などは，抽象クラスのコンポーネント・タイプ（DU）やノード・タイプ別に共通属性を正規化して仕様書に記述することにより，重複が少なくなり作成・保守が容易になる．

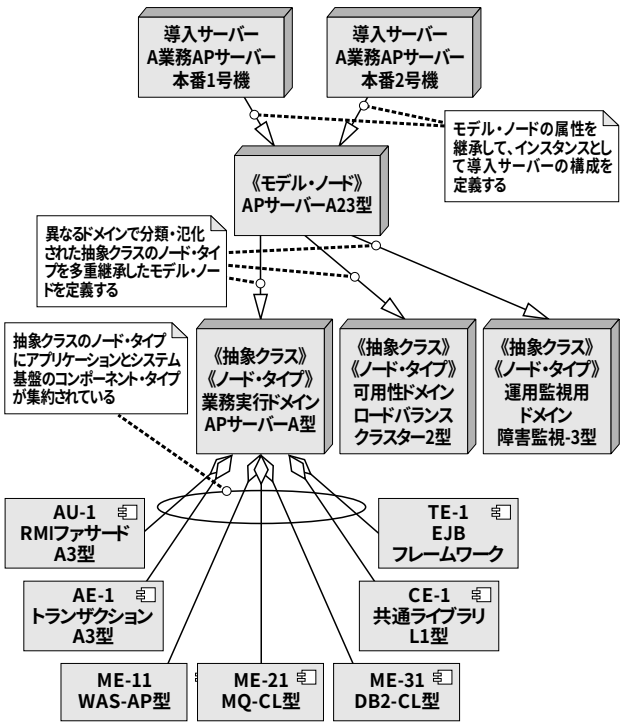


図7. ノード・タイプとモデル・ノード

表7. モデル・ノード表

抽象クラス	ノード・タイプ			
モデル・ノード	業務実行ドメイン	可用性ドメイン	運用監視ドメイン	・・
APサーバー-A23型	APサーバー-A型	ロードバランス・クラスター-2型	障害監視-3型	・・
APサーバー-B21型	APサーバー-B型	ロードバランス・クラスター-2型	障害監視-1型	・・
・・・・・	・・・・・	・・・・・	・・・・・	・・

3.8 統合ウォークスルー

文献［1］で紹介しているように，ADSでは，CMとOMで同じユースケースのシナリオを共有し，CMで

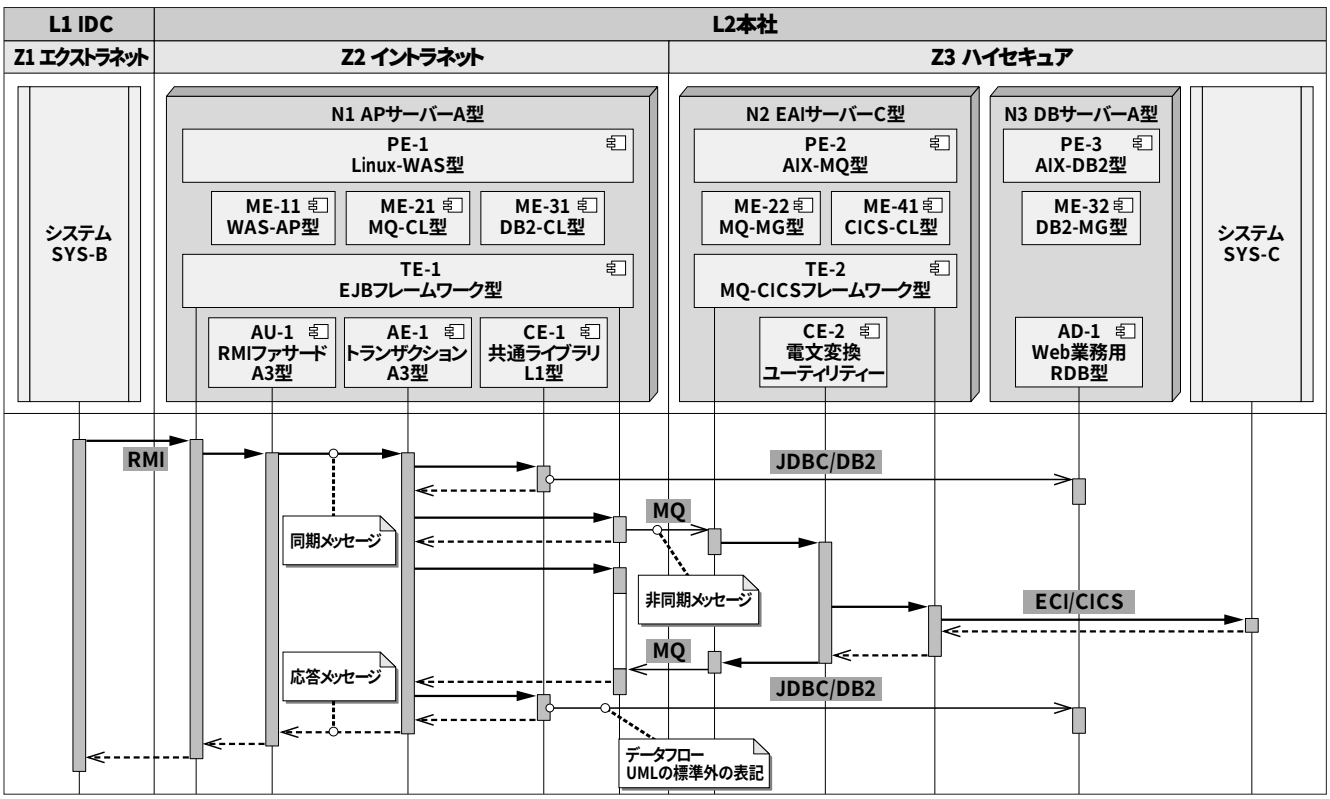


図8. 統合ウォークスルー図

は, コンポーネント相互作用図により, OMでは, ウォークスルー図により, ユースケース・シナリオの実現性を動的なアーキテクチャ・モデルで評価する. しかし, 実際のプロジェクトでは, 異なるチームにより, 別々に設計を行っているため, 両者の間にあるギャップを見つけにくい.

提案手法では, CMのコンポーネント相互作用図(シーケンス図)にOMのロケーション／ゾーンとノードを統合してウォークスルーする手法(**図8**)を導入している. この統合ウォークスルー図により, ネットワーク経由でリモート接続になるコンポーネント間の相互作用に必要なプロトコルやトランザクションのスコープが明確になり, セキュリティーやパフォーマンスのリスクや障害時の回復機能などの割り当て漏れが発見しやすくなる.

4. おわりに

本論文では, オブジェクト指向のアプローチを以下のようなITシステム基盤のアーキテクチャ記述に適用する手法を提案した.

- 様々なNFRのドメインで抽象クラスとしてコンポーネントやノードをタイプに分類・汎化(はんか)する
- ドメイン別にコンポーネント・タイプをノード・タイプに集約する
- ドメイン別のノード・タイプをモデル・ノードが多重継承する
- モデル・ノードからインスタンスとして個別の導入サーバーを継承する

また, これらのモデル要素間の関係を正規化した様々な表形式のテンプレートで記述してトレーサビリティーを確保している.

この手法を効果的に適用するためには, ITシステム基盤のアーキテクチャのモデル要素について, 属性レベルのデータモデルを作成して, コンポーネントやノードの属性を記述する抽象クラスとそれらを関連付ける関連表を明確に定義する必要がある.

今後は, アプリケーション開発に適用されているモデリング・ツールをITシステム基盤のアーキテクチャのモデリングに活用する方法を追及しようと考えている.

謝辞

本論文の作成にあたっては, 著者が手法を試みた複数のプロジェクトのメンバー(柿本達彦氏, 梅内伸氏, 藤岡里織氏, 久波健二氏, 白敏治氏, 羅軍氏, 成瀬晶子氏, 川端美和子氏)やITアーキテクト研修講師コミュニティーのメンバー(長井浩氏, 吉田幸彦氏)より多くの助言を頂きました. あらためて深謝いたします.

参考文献

[1] 山下眞澄, 情報システムの設計思想　第2回要件と制約, 日経ITプロフェッショナル, pp.106-111, 2004.4
[2] R. Youngs, et al., A standard for architecture description, IBM System Journal Volume 38, No. 1, pp.32-50,1999 http://www.research.ibm.com/journal/sj/381/youngs.html, 2004.7.20
[3] Bachmann et al., Software Architecture Documentation in Practice: Documenting Architectural Layers, CMU/SEI Special Report CMU/SEI-2000-SR-004, March 2000, http://www.sei.cmu.edu/publications/documents/00.reports/00sr004.html, 2004.7.20
[4] 吉田幸彦, 榊原彰, 情報システムの設計思想　第1回「ITアーキテクチャ」とは何だろう, 日経ITプロフェッショナル, pp.98-103, 2004.3
[5] 菊間裕二, 情報システムの設計思想　第3回コンポーネント・モデリング, 日経ITプロフェッショナル, pp.120-125, 2004.5
[6] 長井浩, 情報システムの設計思想　第5回運用基盤的側面のモデル化(オペレーショナル・モデリング), 日経ITプロフェッショナル, pp.148-153, 2004.7
[7] 佐藤浩司, Webホスティング・インフラストラクチャのアーキテクチャ検討と基本設計, IBM ProVISION No.35, pp.46-55, Fall 2002 http://www-6.ibm.com/jp/provision/no35/
[8] 山本久好, 榊原彰, オペレーショナル・モデリングにおける非機能要求の効果的検証方法, IBM ProVISION No.41, pp.85-92, Spring 2004 http://www-6.ibm.com/jp/provision/no41/
[9] UML 2.0 Superstructure Specification, OMG Adopted Specification, ptc/03-08-02, August 2003
[10] Edward V. Berard, Abstraction, Encapsulation, and Information Hiding, The Object Agency, http://www.toa.com/pub/abstraction.txt, 2004.7.30
[11] Jens Kaasb_ll, Abstraction and concretizing in information systems and problem domains: Implications for system descriptions and theoretical frameworks, ISCO, 1995 http://citeseer.ist.psu.edu/7839.html, 2004.7.20
[12] Walter F. Tichy , A Catalogue of General-Purpose Software Design Patterns, in Proc. Technology of Object-Oriented Languages and Systems (TOOLS 23), IEEE Computer Society, 1998 http://wwwipd.ira.uka.de/~tichy/entwurfsmuster.html, 2004.8.10

[13] Deepak Alur et al, Core J2EE Patterns, http://www.corej2eepatterns.com/, 2005.8.10
[14] Mo Jamshidi, System-of-Systems Engineering - a Definition, General Chairman, IEEE SMC 2005, http://ieeesmc2005.unm.edu/SoSE_Defn.htm, 2005.8.10
[15] Jason Bloomberg, Using the RUP for Enterprise e-business Transformation, IBM developerWorks, 2001, http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/jan01/UsingtheRUPforEnterpriseebusinessTransformationJan01.pdf, 2004.7.20
[16] Fredrik Ferm, The what, why, and how of a subsystem, IBM developerWorks, 2003, http://www-106.ibm.com/developerworks/rational/library/1761.html, 2004.7.20
[17] Joaquin Miller and Rebecca Wirfs-Brock, How Can a Subsystem Be Both a Package and a Classifier?, Wirfs-Brock Associates, http://www.wirfs-brock.com/PDFs/how%20can%20a%20subsystem%20be%20both.pdf, 2005.8.10
[18] IBM Patterns for e-business, http://www-128.ibm.com/developerworks/patterns/ja_jp/, 2005.8.10

日本アイ・ビー・エム株式会社
ICPシニアITアーキテクト
大嶽 隆児 Ryuhji Ohtake

[プロフィール]
1981年に日本IBM入社．分散システム担当SEとしてさまざまな業種のプロジェクトを経験した後，長野オリンピックの大会情報システム(Info'98)のチーフITアーキテクトを担当．シドニーオリンピックのテクニカルアドバイザーを経て，第一線のITアーキテクトとして多くのe-businessプロジェクトに従事．社内のITアーキテクト研修の講師も担当．

ohtakro@jp.ibm.com