システム基盤設計における仕様の抽出と記述 ――形式仕様記述と設計検証自動化の試み

Extracting and Describing Specification for Systems Infrastructure Design —— Attempt toward Formal Specification and Automated Design Verification

今 泉 正 雄

要 約 システム基盤の設計には，基盤に対する要求仕様のほか，これを構成するハードウェアおよびソフトウェアモジュール，モジュール間の接続性といった仕様への理解が必要となる．こうした仕様の抽出は必ずしも容易ではないが，これを明確に記述することで，設計の妥当性を示すことができる．また，この記述に形式仕様記述言語を使うことで，製品仕様自体の検証や，システム設計検証の自動化への道も拓ける．

Abstract For designing systems infrastructure, it is necessary to understand specifications of both software and hardware modules, and inter-module connectivity. To extract these specifications is not always easy, but description of these specifications gives the validity of design. Furthermore, by using formal specification description language, it will be possible to verify the module specification itself and to automate system design verification.

1. は じ め に

日本ユニシスは1990年代よりミッションクリティカルな世界でもシステム基盤のオープン化を提案しており，OEM製品を含む自社製品を中心に多くの構築，稼働実績を作ってきた．しかし，時代と共に多様な他社ベンダ製品を扱わざるを得ない局面が増えるにつれ，個別の接続性の確認に追われるようになってきている．たとえば，特定のサーバに対して，あるストレージが接続できるか，あるソフトウェアがインストールして使用できるか，等である．オープンシステムによる基盤構築は，ハードウェア製品やソフトウェア製品がオープン規格によって標準化されてはいるものの，各製品間の接続確認が取れないまま構築し稼働させた場合，将来予期せぬトラブルが発生し，その際に製品ベンダのサポートが得られない等の問題も起こり得る．このため，規格レベルの仕様確認では不十分なのである．

この接続性の確認には，基盤設計フェーズでの仕様確認と基盤構築後の動作検証が不可欠である．前者は製品の規格の他，製品ベンダの保証（certification）を確認すべきである．この規格や保証は，接続性における製品仕様と考えられるが，その情報は各製品のライフサイクルに従う流動性があること，接続製品が複数ベンダにまたがる場合には各社の方針，思惑といったものに左右されがちであること，等のため，確認は困難を伴う．しかしコストをかけて抽出した製品仕様は，本来の基盤設計とともに記述して残すことで，設計妥当性の保証となり，記述した仕様を蓄積することで，新たな基盤設計やその検証を容易にする．

本稿では主に，高可用性システムの基盤構築を例に，必要な製品の仕様を記述する．その後，それらの製品を使ったシステムを設計し，この設計の検証を試みる．

2. システム基盤設計とモジュール間接続検証

情報システムの構築においては，アプリケーションの構築とともに，それを動作させるためにシステム基盤の構築が必要である．本章では，モジュールの視点からシステム基盤を定義し，モジュール間接続の問題を論ずる．また様々なハードウェアモジュールをラックに設置する例を用いて，設計の検証方法について考察する．

2.1 システム基盤と基盤設計

本稿におけるシステム基盤に含まれるモジュールには，サーバ，ストレージ，ネットワーク等のハードウェアと，オペレーティングシステム，ミドルウェア等の基本ソフトウェアとがある（表1）．

表1 情報システムを構成する主要モジュールと基盤の範囲

ソフトウェア	データベース(DB)アプリケーション／WEBアプリケーション	システム基盤
	データベースマネジメントシステム(DBMS)	
	WEBサーバ／APサーバ	
	ジョブ管理ソフトウェア	
	オペレーティングシステム(OS)／言語処理系	
	クラスタソフトウェア／ボリュームマネージャ(VM)	
ハードウェア	サーバ／コンソール／ストレージ	
	ラック／電源設備／SAN／ネットワーク	

これらのモジュールを使用した基盤設計・構築の流れを図1に示す．このうち基盤設計とは，(1)要求仕様に基づきシステム全体をハードウェアやソフトウェアモジュールに分割し(2)それらを具体的な製品に置き換え(3)各製品に必要なパラメータを決定する，一連のプロセスのこととする．

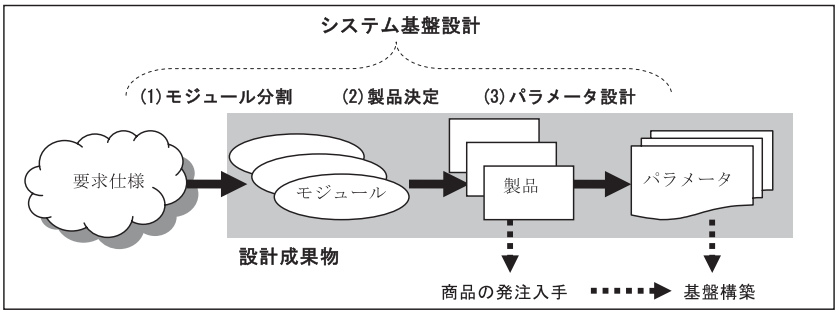

図1 システム基盤の設計から構築の流れ

2.2 モジュールおよび製品の接続性

システム基盤設計において，モジュール分割の際に表1に見られる各種モジュールを，接続性を考慮しながら選択することになる．ここではハードウェア同士の接続性以外に，ハードウェアとソフトウェアの接続性（アクセスの可能性等）や，ソフトウェア同士の接続性（連携の可能性等）も考慮する（表2）．

表2　モジュール間接続の例

ハードウェア同士の接続	サーバとストレージ サーバとネットワークスイッチ
ハードウェアとソフトウェアの接続	サーバとOS ストレージとボリュームマネージャ
ソフトウェア同士の接続	OSとDBMS APサーバとDBMS

こうした接続性は，各モジュールを実際の製品に置き換えたときに，検証の必要性が生まれる．そこで本稿では，基盤設計の中でも特に，製品決定プロセスにおける，製品同士の接続性検証に焦点を当てる．製品を発注した後に接続性に問題が見つかっても手遅れであるため，ここでの検証は非常に重要である．

なおパラメータ設定については，それ次第で接続の可否が決まるケースも多いが，紙面の都合で本稿では割愛する．

2.3　検証の観点とシステムモデル

基盤設計の様々な成果物を，机上で一括して検証することは難しいため，検証の観点を定めた上で，その観点において正しいことを検証していく．このためにまず，ある観点において必要なモジュールとモジュール間の接続性について規定した「システムモデル」を考える．

たとえば，サーバ，ストレージ，ネットワークスイッチ等のハードウェアモジュールで構成されるシステムを，1台のラックに納めることを考える．このシステムモデルを「ラックシステム」とし，ラックとそれに格納する各種ハードウェアモジュール，およびその接続性を定義する．

ラックシステムにおいては，サイズに関して次のような接続性の確認が必要である．

・　各モジュールの幅はラック幅に一致すること．
・　各モジュールの奥行きはラックの奥行きを超えないこと．
・　全モジュールの高さの合計が，ラックの高さを超えないこと．

また消費電力に関しては，次の確認が必要である．

・　全モジュールの消費電力量が，ラックあたりの許容電力量を超えないこと．

このほか，重量や発熱量といったものも，ラックと個々の機器との接続性の問題として考える必要がある．

2.4　モジュールと製品の仕様記述

ラックシステムの検証に必要なモジュールおよび製品の仕様は，幅，奥行き，高さといったサイズや重量，消費電力や発熱量などであった．一方，サーバのメモリサイズやCPUスペックなどは，システムのパフォーマンスを考える上では重要であるが，ラックシステムにおいては直接関係のない仕様である．

基本的に，システム基盤設計における製品仕様は，一般に膨大な仕様の中から必要十分なものだけを抽出することが大切である．ただし，検証の観点が複数あり，それぞれに対するシステムモデルが作成される場合においては，各モデルが必要とするモジュール仕様の最小公倍数的なものが記述され，製品仕様として抽出される必要がある．

3. モジュール仕様の抽出

本章では，高可用性データベースシステムの構築を例に議論する．ここで高可用性（HA - High Availability）システムとは，様々な障害を予想してハードウェア（特にサーバ）を冗長化し，サービスを提供し続けるシステムを指すが，本稿では稼働率には言及しない．

HA システムには，サービスを複数のサーバで起動する方式と，単一のサーバ上で動作するサービスを，何らかの障害時に他のサーバに切り替える（フェイルオーバ）方式がある．

3.1 HA データベースシステムの概要

ここでは 2 ノードで構成される，フェイルオーバ型の HA データベースシステムを考える．データベース本体は，各ノードからアクセス可能な共有ストレージ 1 台にもつものとする．

このシステムでは，サービスやサーバを監視し，障害時には DBMS プロセスの他，ストレージへのアクセス（論理ボリュームのインポートやファイルシステムのマウント），ネットワークからのアクセス（論理 IP アドレスの起動）をスタンバイ側に切り替える（図 2）．オープンシステムにおいては，こうした監視やサービス起動，停止などをクラスタソフトウェアによって実現するのが一般的である．

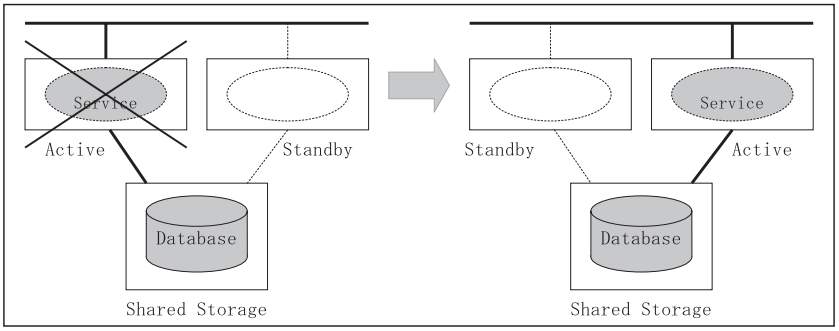

図 2　フェイルオーバ型の HA データベースシステムとその動作

3.2 モジュールの抽出

図 2 の HA データベースシステムを実現するための主なハードウェア，およびソフトウェアモジュールを抽出したのが，表 3 である．

表 3　フェイルオーバ型 HA データベースシステムを構成する主なモジュール

ハードウェアモジュール	サーバ　2台
	ストレージ　1台
ソフトウェアモジュール	オペレーティングシステム（OS）
	ボリュームマネージャ（VM）
	データベース・マネジメントシステム（DBMS）
	クラスタソフトウェア

表 3 の中でボリュームマネージャは，物理ディスクを論理ボリューム化し，サーバからのアクセスに対し様々な機能を提供するソフトウェアである．共有ストレージを使用するフェイルオーバ型の HA システムにおいては特に，複数サーバに対するアクセスの排他制御の機能が重要である．

3.3　クラスタソフトウェアの接続性

表3に抽出した各モジュールの接続性について確認する．接続性に関する製品仕様は，ライフサイクルが比較的短い製品や，特殊なソフトウェア製品に記載される傾向がある．たとえば，サーバは導入可能なOSを仕様として規定し，DBMSはインストール可能なOSを規定する（図3）．

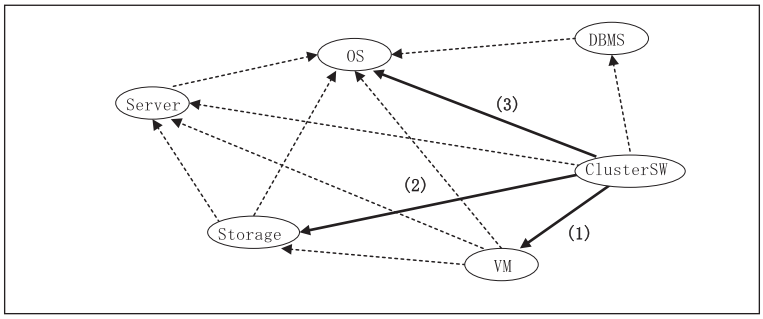

図3　各モジュールの接続性と本稿での記述範囲

次章では，汎用的なHAシステムを定義するにあたり，クラスタソフトウェアの仕様として，ボリュームマネージャ，ストレージ，OSとの関係の記述を試みる．

4.　モジュール仕様と製品仕様の記述

クラスタソフトウェアを中心とするモジュールの仕様は以下のとおりである（図3参照）．

（1）クラスタソフトウェアは，使用可能なボリュームマネージャを限定する．

共有ストレージを持つフェイルオーバ型のHAシステムを構成するため，ファイルシステムの排他制御を行うことがある．このために，スタンバイノードに対する論理ボリュームのデポート処理，アクティブノードに対するインポート処理を行うボリュームマネージャ製品をクラスタソフトウェアが規定することがある．

（2）クラスタソフトウェアは，使用可能なストレージを限定する．

複数サーバが接続可能な共有ストレージが主たる対象である．ただしHAシステムの場合，単に複数サーバが接続可能であるだけでなく，コントローラや電源の冗長性，ディスクのホットスワップ機能なども重要となる．

（3）クラスタソフトウェアは，インストールおよび実行可能なOSを規定する．

他のソフトウェア同様，クラスタソフトウェアもインストールおよび実行可能なOSであることが必須要件である．特にサーバを含むハードウェアの監視や各種プロセス，IPアドレスなどの監視，起動，停止などを行う必要があるため，一般のソフトウェアとは異なる要件が求められる．

本章では，各モジュールと，モジュールに対応する具体的な製品の仕様を記述する．製品共通の性質をモジュール仕様として予め記述することで，個々の製品仕様の記述が容易になる．

4.1　仕様記述のための言語

各モジュールおよび製品の仕様記述には，形式仕様記述言語[1][2]を用いる．形式仕様記述（Formal Specification）は主に，ソフトウェア設計のために開発された技法で，システムが何をすべきかを，集合論や論理式などの数学によって記述する．

ソフトウェア設計では状態遷移を記述するため，事前条件・事後条件を必要とするが，基盤設計においては状態の変化の考慮は通常必要ないため，不変条件の記述ができれば十分である．ただし，ストレージのディスク増設や，OS に対するパッチ適用など，将来的に見込まれる変化を記述する，といった使い方も考えられ，形式仕様記述言語がこの目的のために必ずしもオーバスペックとは言えない．

代表的な仕様記述言語としては，Z 記法[3]，VDM-SL[4]，B メソッド[5][6]等があるが，本稿ではこのうち Z 記法を採用する．Z 記法はシンタックスが単純であり，また数学記号がそのまま使えるため，予備知識がなくとも理解が容易と考えられる．なお，Z 記法は 1990 年代に日本ユニシスでも多く実証[7]されており，2002 年には ISO の標準化[8]もなされている．

4.2　ボリュームマネージャとクラスタソフトウェアの仕様記述

ボリュームマネージャは一つの名前を持ち，クラスタソフトウェアは複数のボリュームマネージャを使用できる，というモジュール仕様を，図 4 に記述する．

1 行目では，*VMname*（ボリュームマネージャの名前の集合）を所与の型として定義する．なお Z 記法において型はすべて集合[*1] として定義されるが，所与の型においては個々の要素については言及しなくてよい．

次の *VolumeManager* は「スキーマ」で，その変数として vm を持つ．コロンの後は変数の型を示すが，ここでは *VMname* を記述している．

また *ClusterSoftware* もスキーマである．変数 *vmset* の型は *VMname* の冪集合（Power set）[*2] として定義している．

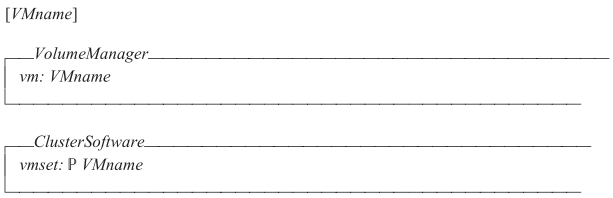

図 4　ClusterSoftware と VolumeManager の仕様

4.3　ボリュームマネージャ製品の仕様記述

具体的なボリュームマネージャ製品として，*VeritasVolumeManager*[9]（*vxvm*）および，*SolarisVolumeManager*[10]（*svm*）の二つの仕様を記述する（図 5）．

VMname を所与の型から要素（*vxvm*，*svm*）を定義する列挙型に書き換え，*VolumeManager* のスキーマを拡張し定義している．

スキーマの拡張はオブジェクト指向におけるクラス継承に相当する．これはスキーマ上段に親のスキーマ名を記述することで成される．スキーマの下段には変数の初期化（ここでは名前の設定）を記述する．

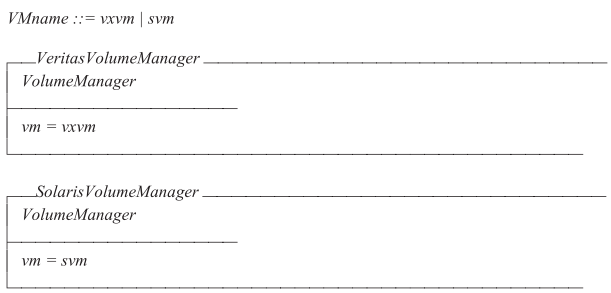

図 5　VolumeManager の拡張による製品の仕様

4. 4　クラスタソフトウェア製品の仕様記述

具体的なクラスタソフトウェア製品として，*VeritasCluster*[11] と *SunCluster*[12] の二つを *ClusterSoftware* の拡張として図 6 に記述する．前者が使用可能なボリュームマネージャは *vxvm* のみ，後者は *svm* と *vxvm* である．

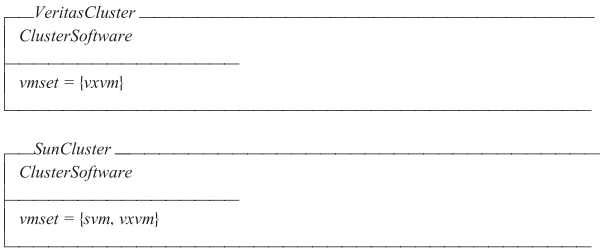

図 6　ClusterSoftware の製品記述

4. 5　ストレージの追加

図 4 の *ClusterSoftware* に，使用できるストレージを追加する（図 7）．ここで *STGname* はストレージ名を意味する所与の型として定義している．

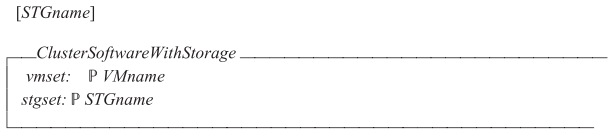

図 7　ClusterSoftware へのストレージの仕様追加

図 7 は自然な仕様追加に見えるが，実はこの仕様は *SunCluster* への拡張に問題がある．リリースノートの外[13]に，ボリュームマネージャとして *vxvm* が認められないストレージが見つかるからである．この事実から帰納的に推論すると，*ClusterSoftware* の仕様は，ストレージ名からそれに対応するボリュームマネージャ名の集合への写像[*3] として定義すべきだろう（図 8）．

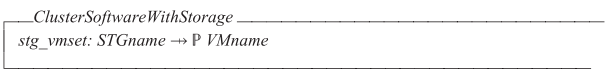

図 8　修正された ClusterSoftware の仕様

このモジュールの仕様記述は，具体的なクラスタソフトウェア製品とストレージを選択する際，使用可能なボリュームマネージャ製品の確認が必要であることを思い起こさせる．図9は，図8でクラスタソフトウェアの仕様にストレージを追加する際に，必ずボリュームマネージャ製品との対応を記述することを強制した結果である．

こうした記述の強制は，その後の製品仕様の読み忘れを防止することにつながるが，これは形式仕様記述言語を用いる利点の一つである．

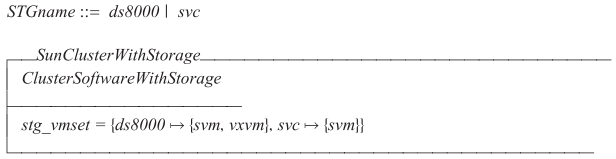

図9　SunCluster で使用可能なストレージとボリュームマネージャの定義例

4.6　オペレーティングシステムの追加

ClusterSoftware に動作可能なオペレーティングシステムを追加する（図10）．

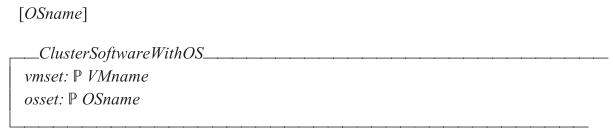

図10　ClusterSoftware へのオペレーティングシステムの仕様追加

図10の仕様に基き，一つのシステム上には複数のサーバがあるために，これらのオペレーティングシステムがすべて *osset* に含まれることは当然確認すべきである．これに加え，すべてのサーバ上のオペレーティングシステムが同じであることは必要だろうか．

マルチプラットフォーム上で動作可能な *VeritasCluster* のリリースノート[11]には「クラスタ内のすべてのノードが同じプロセッサアーキテクチャを使用し，同じオペレーティングシステムを実行していることが必須」という記述が見られる．しかし Solaris と AIX は UNIX という意味で同じと言えるのか，Solaris9 と Solaris10 といったバージョンの違いは無視できるのだろうか．こうした疑問は，自然言語の曖昧さから生じるが，厳密な仕様記述言語に翻訳する過程でこそ意識される問題かもしれない．これも形式仕様記述言語を用いる利点の一つと言える．

5.　システム設計と検証

本章では，具体的な製品を当てはめた HA システムを設計し，その妥当性を検証する．これに先立ち，各モジュールをどのように検証するかを規定したシステムモデルを定義する．

システムモデルとシステム設計の関係は，モジュール仕様と製品仕様の関係に相当する．すなわちシステム設計は，システムモデル中の各モジュールをそれぞれ，具体的な製品に置き換えたものとして定義される．

5.1　HA システムモデルの定義

図4のモジュール仕様に基づき，クラスタソフトウェアとボリュームマネージャのみの簡単

な HA システムモデルを図 11 に示す．

スキーマ内の上段 2 行は，*ClusterSoftware* と *VolumeManager* のスキーマを参照することの宣言であるが，これはそれぞれクラスタソフトウェア，ボリュームマネージャという二つのモジュールを使用することを表している．次の *ret!* は出力用の変数で，その型 *Status* は *ok*, *ng* の 2 値の列挙型として定義されている．スキーマ下段には検証の方法を記述する．すなわち *ClusterSoftware* に定義される *vmset* が *VolumeManager* に定義される *vm* を含めば *ok*，含まなければ *ng* を出力する．

$$Status ::= ok \mid ng$$

HASystem

$$\Xi ClusterSoftware$$
$$\Xi VolumeManager$$
$$ret!: Status$$

$$vm \in vmset \Rightarrow ret! = ok$$
$$vm \notin vmset \Rightarrow ret! = ng$$

図 11　HA システムモデル

図 8 のクラスタソフトウェアの定義に基づき，ストレージを含めた検証をする場合には，図 12 のように拡張する．

HASystemWithStorage

$$\Xi ClusterSoftwareWithStorage$$
$$\Xi VolumeManager$$
$$\Xi Storage$$
$$ret!: Status$$

$$stg \in \mathrm{dom}\ stg\_vmset \land vm \in stg\_vmset\ stg \Rightarrow ret! = ok$$
$$stg \notin \mathrm{dom}\ stg\_vmset \lor vm \notin stg\_vmset\ stg \Rightarrow ret! = ng$$

図 12　ストレージ拡張した HA システムモデル

5. 2　HA システムの設計

HA システムの設計は，前節で定義したいずれかの HA システムモデルの各モジュールを具体的な製品に置き換えることで実現する．Z 記法ではモジュールを定義する参照スキーマを，製品を定義する参照スキーマに書き換えることで実現できる．図 13 は図 11 の参照スキーマである *ClusterSoftware*，*VolumeManager* をそれぞれ *SunCluster*，*SolarisVolumeManager* に置き換え，定義したものである．

System1

$$HASystem$$
$$\Xi SunCluster$$
$$\Xi SolarisVolumeManager$$

図 13　SunCluster と SolarisVolumeManager を使用した設計

5. 3　設計の検証

図 13 の設計は，次の図 14 に記述した定理を証明することで検証できる．

theorem *t1*
　System1 ⇒ *ret!* = *ok*

図 14　設計検証のための定理

図 14 の定理は人手で証明することも，Z/EVES[14]等の証明器を用いて半自動的に証明することも可能である．しかし，図 13 の設計で *SunCluster* を *VeritasCluster* に置き換えると，*VeritasCluster* は *vmset* の中に *svm* を含まない（図 6）ため証明に失敗する．

6. お　わ　り　に

広範囲に散在する，あるいは明文化されていない製品仕様の中から，基盤設計やその検証に必要な仕様を正しく抽出することは簡単なものではないが，その際仕様を記述しておくことで，第三者が設計を再検証できるだけでなく，その後の設計とその検証にも役立つ．

本稿ではモジュール，製品仕様等の記述を，形式仕様記述言語で行ったが，これには次のような意義が認められた．まず，厳密に仕様を書くことで，自然言語の曖昧さに気づき，また言語外の仕様を帰納的に導く必要が生じること，次に設計記述の標準化を促すこと，さらにその設計に対し自動検証の可能性を見出せることである．

形式仕様記述言語としては今回，Z 記法を採用したが，記述者の観点からは継承による拡張が容易にできることが望ましい．基盤設計においては，類似の機能を持つ多種の製品を扱う必要があるため，これらの抽象記述と各製品の具象記述ができることが望まれる．また本稿では言及できなかったが，複数のシステムモデルから新たなシステムモデルを定義したいことがある．たとえば，HA システムモデルと独立にデータベースシステムモデルを定義し，それらの継承として HA データベースシステムモデルを定義する場合である．これはオブジェクト指向でいう多重継承にあたる．Z 記法でこれらの継承を模すことが可能だが，より積極的にオブジェクト指向記述を行うには，VDM++[15]が言語仕様，ツールともに強力である．VDM++ ではクラスのインスタンスが生成でき，設計をシステムモデルのインスタンスとして生成可能か否か，というテストによる検証方法がとれる．本稿の基となる国立情報学研究所での研究[16]では，この VDM++ を採用した．

最後に国立情報学研究所にて筆者にさまざまな形式手法をご教授いただいた先生方に感謝したい．中でも，指導教官でもあった田口研治先生には形式仕様記述の手ほどきから，Z 記法の実践までご教授いただいた．また B メソッドでは來間啓伸先生，VDM-SL，VDM++ では石川冬樹先生にそれぞれ実践的なご指導をいただいた．

＊ 1　a:S において S は集合であるが，これは a は集合 S の要素を取り得ることを表している．
＊ 2　与えられた集合の部分集合を考え，その全体を集めて集合としたもの．次節の例である {svm, vxvm} の冪集合は，｛φ，{svm}，{vxvm}，{svm, vxvm}｝となる．
＊ 3　図 9 の写像，stg_vmset = {ds8000→ {svm, vxvm}, svc→ {svm}｝では．定義域 dom stg_vmset = {ds8000, svc} であり，stg_vmset ds800 = {svm, vxvm}，stg_vmset svc = {svm} となる．

参考文献
［1］ 中島震，「ソフトウェア工学の道具としての形式手法」，NII Technical Report, 2007 年 7 月
［2］ 山崎利治，「形式仕様のための読書案内」，ユニシス技報，日本ユニシス，Vol.18 No.4 通巻 60 号，1999 年 2 月
［3］ Spivey, J. M., “The Z Notation – A Reference Manual Second Edition”, Prentice Hall International (UK) Ltd, 1992
［4］ “VDMTools – The VDM-SL Language Manual ver1.0”, CSK SYSTEMS CORP, 2009
［5］ “B Language Reference Manual Version 1.8.6”, ClearSy
［6］ 來間啓伸，「B メソッドによる形式仕様記述」，近代科学社，2007 年 12 月
［7］ 李春瑞，染谷誠，「Z 開発試行報告―仕様言語の必要性と Z によるプログラム開発の実際」他，ユニシス技報，日本ユニシス，Vol.18 No.4 通巻 60 号，1999 年 2 月
［8］ “Information technology — Z formal specification notation — Syntax, type system and semantics”, ISO/IEC 13568: 2002
［9］ “Veritas Storage Foundation and High Availability Solutions 5.0 Hardware Compatibility List”, Symantec, 2010. 8, ftp://exftpp.symantec.com/pub/support/products/Foundation_Suite/283161.pdf
［10］ “Solaris Volume Manager Administration Guide”, Oracle, 2010, http://download.oracle.com/docs/cd/E19683-01/806-6111/806-6111.pdf
［11］「Veritas Cluster Server リリースノート― Solaris 5.1」，Symantec，2009, ftp://ftp.support.veritas.com/pub/support/products/ClusterServer_UNIX/346120.pdf
［12］ “Sun Cluster 3.2 2-08 Release Notes”, ORACLE, 2000. 7, http://wikis.sun.com/display/SunCluster/(Japanese)+Sun+Cluster+3.2+2-08+Release+Notes
［13］ “ORACLE SOLARIS CLUSTER STORAGE PARTNER PROGRAM”, ORACLE, 2010.8, http://www.oracle.com/technetwork/server-storage/solaris-cluster/partnerprogram-cluster-168135.pdf
［14］ Mark Saaltink, “The Z/EVES 2.0 User’s Guide”, ORA Canada, 1999 年 10 月
［15］ “VDMTools – The VDM++ Language Manual ver1.0”, CSK SYSTEMS CORP, 2009
［16］ 今泉正雄，「形式手法を用いた高可用性クラスタシステムのモデル化と検証」，平成 21 年度トップエスイーポスター発表, http://www.topse.jp/pdf/h21poster.pdf
上記参考文献に記載の URL は，2011 年 2 月 3 日時点での存在を確認.

執筆者紹介　今　泉　正　雄（Masao Imaizumi）
1990 年日本ユニシス(株)入社．2009 年 4 月より 1 年間，国立情報学研究所のトップエスイープロジェクトにおいて外来研究員として高可用性システム開発への形式手法の適用について研究．現在はユニアデックス(株)にて，高可用性クラスタシステムを中心とした基盤システムの設計，構築を行っている．