Vol.2009-EMB-14 No.3
2009/7/24

分散ストレージの安全性検証

大　森　洋　一[†1]

概要: 組込みソフトウェア開発において，品質向上を目的としたモデルベース開発の採用事例が増えつつある．しかし，モデルベース開発の上流工程への適用拡大は，要求の曖昧さがモデルの厳密性を損なうという問題がある．本稿では，開発の初期段階において，ドメイン固有言語を用いたモデル記述を行なう手法のネットワークを含むストレージシステムへの適用について報告する．ドメイン固有言語の定義には，機能とふるまいのそれぞれに適したフォーマルメソッドを用い，より厳密な記述および検証を行なう手法を検討する．このとき，組み合わせるフォーマルメソッドを互いにもう一方の制約とみなして一貫性を保証しなければならない．

Formal Verification of Safety on A Cross Network Storage

Yoichi Omori[†1]

Abstract: Model based development is increasingly and widely adopted to improve design quality in embedded software developments. Volatility of requirements in early phases of software design, however, hampers to extend the application of model based development, because such a uncertainty makes the model treacherous. This paper reports on a case of model based development method using a domain specific language (DSL) to a network storage system in early phases. The DSL is corroborated by a functional formal method and a behavioral formal method, and a strict description and verification method is discussed. The important thing to assure is that descriptions in each formal method is constraints to another, respectively.

†1 九州大学 大学院システム情報科学研究院
Graduate School of Information Science and Electrical Engineering, Kyushu University

1. はじめに

1.1 組込みソフトウェア開発の課題

IPA による最新の年次調査である「２００７年版組込みソフトウェア産業実態調査」[26]によると，組込みソフトウェア開発の課題として「設計品質の向上」を事業責任者のおよそ半数が挙げており，最多となっている．その一方で，プロジェクト管理者はプロジェクトの７割以上が品質に関して計画通りもしくは計画以上であったが，およそ半数のプロジェクトは実施期間が予定を超過したと答えている．したがって，現場では，時間をかけて担当部分の品質を確保する技術は確立しているが，製品として組み合わせた場合に，事業責任者の期待する水準に達していない場合がままあると考えられる．さらに，品質問題が発生したことによる対策費は２００５年から一貫して増加している．

また，同調査によると，９割近くが複数のモデリング手法を使った開発を行なっており，モデルの利用が広まっている．モデルを用いた段階的詳細化により要求と設計に一体化することは可能であることは知られているにも関わらず，先に述べた乖離が生じていることから，モデルベース開発の適用範囲を仕様記述へ広げる工夫が必要である．つまり，モデルベース開発において，要求を仕様化する段階と設計および実装を行なう段階に一貫性をもたせられていないと考えられる．

1.2 モデルベース開発

組込みソフトウェア開発において，モデルベース設計を用いる利点として，コンポーネント化による設計の再利用がある．これは，主として開発期間の短縮を目的として，プロダクトライン開発[18] などとの組み合わせにより，プログラムあるいはライブラリよりも高レベルの設計情報の再利用を目指すものである．プロダクトライン開発では，対象領域における複数回の開発における再利用によるコスト回収を前提として，ソフトウェアドメインあるいは問題ドメインをモデル化する．

他の目的として，早期の欠陥発見によるソフトウェア品質の向上が挙げられる[19][28]．ここで欠陥とは仕様と設計の不一致を指し，モデル化，特に複数の観点からのモデル化により，要求の記述不足や要求仕様に対する設計の不備を検証可能とする．古典的な V 字モデルでは，要求に対する不備の検出は統合テスト時に行なわれるので，不具合の修正コストが非常に高くなる．モデルベース設計では，構成時の検証が可能であり，比較的短いサイクルで不具合を修正できる．この場合，単一製品開発であっても，モデルベース設計を採用する意義がある．

Vol.2009-EMB-14 No.3
2009/7/24

本研究では，主に後者の立場から，フォーマルメソッドの利用によるモデル品質の向上について述べる．ただし，両者は独立ではなく，再利用されるコンポーネントの品質は一定以上である必要があり，逆に早期の品質改善による開発期間の短縮が期待できる．

2. フォーマルメソッドの導入

組込みソフトウェアの品質向上に向けて，さまざまな取組がなされている．

例えば，故障モード影響解析や故障木解析といった，これまでハードウェアの品質向上に有効だった手法をソフトウェア設計に応用する試み[9)7)16)]や，QFD を利用した方法[31)]が提案されている．ただし，抽象度の高さや組合せの複雑さなど，ソフトウェアとハードウェアの故障に関する特性の違いから，単純な応用は困難であり，研究が続けられている[32)]．

フォーマルメソッドは，数学的な概念に基づいて問題をモデル化する手法であり，内外歴史がある．抽象的なモデルに対して数学的性質が検証可能という特長から，生成物の検査だけでなく，構成時の保証ができるので，前述のような品質向上の取組と相補的に適用できる．これまでにも組込みソフトウェア品質の向上を目的とした応用は数多い[8)11)12)]．

フォーマルメソッドの適用は，大きく記述と検証に分かれる．記述段階は対象とする問題を，注目する性質に沿ってモデル化する最終的に実行可能なプログラムの生成を考慮する場合，計算機言語への対応のさせやすさから，集合論および述語論理に基づく機能記述あるいはプロセス代数に基づくふるまい記述が多く用いられる．前者には VDM[14)], Z/B method[1)] など，後者には CSP[20)], CafeOBJ[17)] などがある．検証段階でははそれぞれの論理系において，命題など検証すべき性質の表明が満たされるかを確認する．

このように，フォーマルメソッドの適用はソフトウェア品質の向上に有効であると考えられるが，本研究で目的とするモデルベース開発の上流工程への適用拡大という観点でみると，次のような点が未熟である．

- 非機能要求検証
  フォーマルメソッドのほとんどが機能仕様を対象としており，非機能仕様の記述および検証手法の研究は，ふるまいの検証とリアルタイム性に関する部分を除いて，十分に確立していない[13)4)15)]．したがって，非機能仕様を機能仕様へ変換したり，レビューなど他の手法と組合せる必要がある．
  非機能要求は，多様かつ優先度が低いものも含まれるので，変更や修正が頻発する．また定量的な要求も多く，機能の有無より判定が難しいので，論理体系の構築が難しい．
- 自然言語記述との対応付け

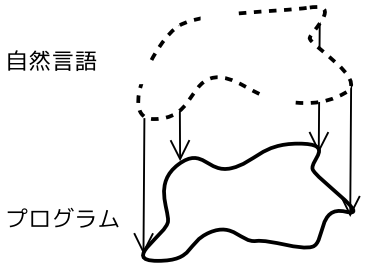

図 1 人手による直接変換
Fig. 1 Direct Conversion by a progrmmer

初期の要求記述には，対象となる問題ドメインの専門家の協力が欠かせない．一般に，問題ドメインの専門家は情報科学の専門家ではなく，その知識は自然言語あるいは非形式的な図式で表現される場合がほとんどであり，厳密さに欠ける．意味のある検証のためには，モデルの要素に，きちんと意味を与えなければならない[6)10)5)]．
この問題を避けるため，フォーマルメソッドは基本的に，自然言語ではなく，独自の言語による記述を行なう．これがフォーマルメソッドの習得および情報科学の非専門家による利用の障壁となっている．

3. 自然言語によるモデル記述

第 2 節で述べた問題を解決するために，本稿ではドメイン固有言語 (Domain Specific Language, 以下 DSL) を用いる手法を検討する．

ここまでの検討で，既存のフォーマルメソッドの枠組をそのまま，ソフトウェア開発の上流工程へ適用するのは困難であることが分かった．情報科学の非専門家とやりとりするには自然言語を用いる必要がある．この観点から，日本語による仕様記述からのプログラム生成が研究されている．

プログラマが日本語による仕様を読み，直接プログラムを生成する場合，図 1 のように，対象とするシステムが表現する意味の境界は曖昧になりがちであり，これを実装中に解釈して，対応するプログラムを作成することになる．このため，できあがるプログラムの品質はプログラマの技量に依存し，仕様と対応する部分と対応しない部分ができてしまう．

80 年代の研究は，計算機言語を含む形式言語を自然言語へ対応付けることによって，自然言語による仕様に厳密な意味を与えた[25)27)]．対応を逆にたどることで，自然言語記述を

Vol.2009-EMB-14 No.3
2009/7/24

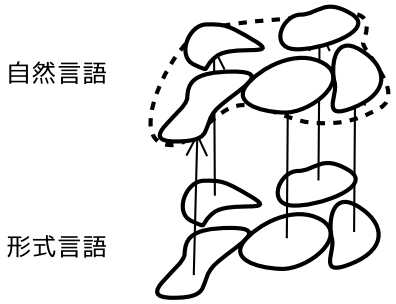

図 2 形式言語から自然言語への対応付け
Fig. 2 Mapping from formal method to natural language

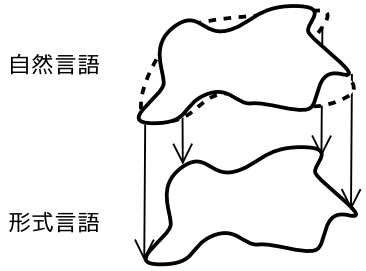

図 3 自然言語から形式言語への対応付け
Fig. 3 Mapping from natural language to formal method

形式言語記述へ変換できる．

この方式の問題は，自然言語による仕様といいながら，実質的に形式言語で書くことになってしまうところである．図 2 に示すように，形式言語を理解していないと不自然な文法や用語の制限が多くみられ，識別子などを日本語で表現するだけで，表現したい要求をモデルとして表現するのが困難であった[⋆1]．

90 年代の研究は，日本語処理研究の進捗と計算機の演算能力の向上に従い，自然言語記述の機械的かつ正確な解釈を目標としている．

和田らは，仕様に書かれる日本語は

- 名詞のほとんどが業務用語である
- 動詞や形容詞の種類は比較的少数であり，処理や動作を表現する
- 数理的な比較や条件が記述される

という特徴があり，若干の文法知識による補完で，ネットワーク型の構文木が構成できることを示した[24)]．

西田らは，構文解析に格フレーム解析を応用し，仕様のように語彙が少ない場合は，曖昧さの排除に特に有効であることを示した[30)]．

小林らは，同じく格フレームによる構文解析を，限定した問題領域に応用し，係受けの関係に基づいた語義の曖昧さを解消するのに有効であることを示した[23)]．

これらの方式は，図 3 のように，自然言語記述の意味を一意に定め，計算機言語を含む形式言語に対応させる手法であり，ライブラリ開発やドメインを限定したモデリングで成果を挙げている．

しかし，本研究で対象とするより上流の工程では，頻繁に要求変更が行なわれる．また，仕様が早期に確定する場合は問題ないが，アジャイル開発の事例でしばしば指摘されているように[3)]，組込みシステム開発においても仕様の確定が下流工程までずれこむ場合がしばしばあり，そうした暫定的な仕様の意味を一意に定めるのはコストに見合わない．

本研究では，上流工程に関して，次のような条件を仮定する．

- 要求記述の段階は，本質的に対話が必要である
  要求は発注者も把握していない部分があり，仕様をまとめる際の対話的な過程によって確定していく．すなわち，自然言語による仕様記述には一意に定まる意味がない段階が存在する．
  そもそも形になっていない要求は，設計者が必要とする情報を発注者に確定してもらわなければならず，必然的に対話的な作業が必要となる．
- 詳細化されるまで表現されない暗黙の仕様が存在する
  上流工程における暗黙の仮定には，人命の安全性など改めて書く必要がないと考えられる要求がある．すべての条件を書き尽くすわけにはいかないので，こうした普遍的な要求は暗黙の仮定となる．
  詳細化の際に，こうした考慮すべき普遍的な要求を見逃すことがないような工夫が必要になる．

⋆1 現在のフォーマルメソッドで用いられる形式言語の検査ツールの多くは，日本語の識別子を利用可能である．

4. 事例研究

本章では，移動端末を利用する際のローカルおよびリモートのデータベース処理を対象としたデータ安全に関する事例を扱う．

4.1 対象システム

事例として取り上げる，ネットワークで結合された複数のデータベースからなるシステムの構成を図 4 に示す．このシステムでは，中央サーバ S とネットワーク経由で接続する移動計算機 C_i からなり，S と C_i 間に静的通信チャネルが存在する．S と C_i はストレージを内蔵し，ユーザは直接 S にアクセスすることはない．

このような環境で，データの安全性を保証するために，全域チェックポイントを保証する通信システム[22] の構築を行なう場合を考える．実際のシステムとしては，携帯電話をクライアントとした大規模データ通信や IC カードシステムの課金処理が該当する．

前提として，以下の 2 点を仮定する．いずれかが成り立たなければ，より簡単に安全性の保証が可能となる．

- C_i は低性能
  C_i ストレージ容量および演算能力はごく小さいとする．すなわち，C_i のストレージは永続的でなく，必要なデータがすべて保存できるとは限らない．また，演算量の大きな圧縮・伸長処理によるデータサイズ縮小も行なわない．

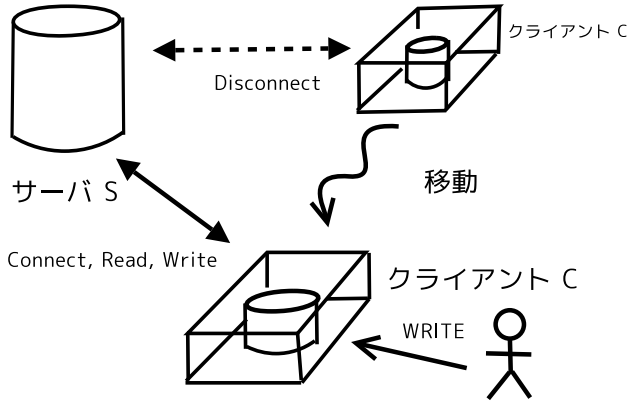

図 4 移動分散ストレージ
Fig. 4 Distributed Storage

- ネットワークは低信頼性
  通信路では，あらゆる時点で，意図しない切断が発生する可能性がある．ただし，通信データの致命的な誤りは生じないものとする．本稿では扱わないが，低レベルでの冗長性やエラー補償が有効である範囲を想定する．

動的な通信シナリオは，通信が書き込み途中で切断される場合を対象とする．簡単のため，サーバ S と単一のクライアント C についてのみ考慮する．

正常な手順として以下のような処理を想定し，切断が生じた場合のデータ不整合を検証する．

(1) 外部から C への書き込み
(2) C から S への接続要求
(3) S から C への接続確認
(4) C から S への書き込み要求
(5) S から C への書き込み許可
(6) C から S への書き込み (複数回)
(7) S から C への書き込み確認 (複数回)
(8) C から S への書き込み完了通知
(9) S から C への書き込み完了確認
(10) C から S への接続解除要求
(11) S の接続解除

4.2 DSL 定義

本研究では，自然言語の可読性と形式言語の検証可能性を両立させるために，自然言語による語彙をもつ抽象度の高い DSL の利用を提案する．

DSL により，対象ドメイン知識の自然言語による表記と，より形式的な計算機ドメインの言語をマッピングする．モデルの検証や分析はフォーマルなモデルで行ない，問題ドメインの専門家とのやりとりは自然言語で行なうことが可能になる．

図 5 のように，これはモデルベース設計の詳細化手順と一致しており，フォーマルメソッドを用いてモデルおよび詳細化の変換時に意味を保証するという点が異なる．

これまでの研究では，自然言語と形式言語の結び付きは一対一になることが想定され，しばしば文法的に置換により，直接的に対応づけられていたが，フォーマルメソッドを用いて意味を定義することにより，対応関係をより柔軟にすることが可能となる．

DSL の定義は，語彙と文法からなる．語彙に関しては，移動端末のストレージに関する

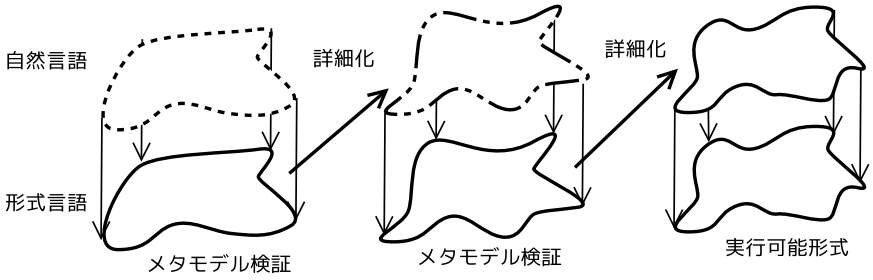

図 5 DSL による詳細化
Fig. 5 Refinement of specifications on natural language

用語や機能について，JISX-6319 および JISX-6320 のプロトコルに関する規定を参考にした[29]．具体的には，データフォーマット，プロトコルと状態遷移，エラー処理などの物理量に依存しない部分を対象として，想定したシナリオに沿って分析したこの結果，最上位で 94 語のキーワードを抽出し，それらの関連を整理した．キーワードのなかに関連も含まれる．

キーワードのほとんどは業務用語であり，明確な定義が存在する．ただし，ふるまいに関する直接的な記述はなく，必要機能の規定になる．このため，プロトコルに関しては，文献[22] およびシナリオに基づく独自の規定を用いた．

自然言語部分の文法に関しては，特に制約しない．本研究は自然言語処理には踏み込まず，意味解釈はマップした形式言語上で行なう．形式言語上の結果を自然言語にフィードバックする工程も必須としない．

表 1 通信に用いられる命令
Table 1 Commands for communication

命令名	自然言語	引数	方向	確認
connect	接続	クライアント ID	$C \rightarrow S$	あり
disconnect	切断	—	$S \rightarrow C$	なし
accept	許可要求	コマンド種別	$C \rightarrow S$	あり
remove	取消	シーケンス番号	$S \leftrightarrow C$	あり
complete	完了	シーケンス番号	$S \leftrightarrow C$	あり
read	読み出し	データ	$S \leftrightarrow C$	あり
write	書き込み	データ	$S \leftrightarrow C$	あり

操作は，文献[29] に定義されている共通コマンドのうち，関係するコマンド 7 種類とその確認を表 1 のように定義した．

DSL の形式言語部分は，ふるまいを CSP モデル，機能を VDM モデルで記述した．これは，第 2 節で述べたように，フォーマルメソッドにはそれぞれに記述のしやすい領域があり，ネットワークのような非同期処理には CSP が，状態機械の記述には VDM が適しているからである．さらに，設計時の見落としを防ぎ安定性を高めるためには，複数の視点によるモデル化が有効である．自然言語と形式言語の対応は表形式で管理する．

4.3 ふるまいの記述

定義した DSL による記述から， CSP を用いてふるまいに関するモデルを記述する．CSP によるモデリングでは，イベントと対応するプロセスの発見が重要である．

本事例では，シナリオの系列をそのままイベントとして表現している．結果を図 6 に示す．通信の切断はプロセス中には生じないので，プロセス間の遷移について検証すればよいことが保証される．

4.4 機能の記述

CSP によるふるまいモデルの状態機械として表現しやすい要素を， VDM を用いてモデルとして記述する．モデリング中は，ふるまいモデルと機能モデルのどちらとしても記述できる部分が混在する．例えば，構造的な階層性をもつようなシステムでは，サブシステムの粒度により，同じ操作が，ふるまいとしてみえる場合と機能としてみえる場合がある．

S と C に分けて記述した結果を図 7 と図 8 に示す

4.5 考　　察

CSP モデルと VDM モデルとの変換は手動で行なったが，モデリングに必要な情報が明らかになっているため，作業量は大きくない．ただし，両者で共通する識別子が同じ概念を指しているかは保証できない．実際， CSP で記述したふるまいを VDM における機能の組合せとして表現する場合，補助機能が必要になり，結果として操作の識別子が一致しなくなった．機械的な変換は可能であるが，上流工程では人間の介在は必須であるため，可読性とどちらを優先するかを考慮しなければならない．

今回のように単純なトップダウンな開発では，解消されないモデリングの矛盾は VDM の記述に表れる．この段階での検証にツールが非常に有用であった[21]．

本事例では，完成された規格書を参考にしたため，要求の矛盾や語彙の曖昧さといった問題は，ほとんど生じなかった．新規のアプリケーション開発などで，要求が整理されていない段階への応用について，より詳細な評価を行ないたい．

WRITE = connect → ack_connect → (accept_write → ack_accept_write)
→ (write → ack_write) → (complete_write → ack_complete_write) → disconnect

CONNECT = connect → ack_connect ⊓ (reject → STOP) ⊓ (deny → STOP)

REQUEST_WRITE = accept_write → ack_accept_write
⊓ (ack_accept_write_error → SKIP)
⊓ (ack_accept_write_bound_error → SKIP)
⊓ (ack_accept_write_align_error → SKIP)
⊓ (ack_accept_write_type_error → SKIP)
⊓ (ack_accept_error → STOP) ⊓ STOP

WRITE_DATA = write → ack_write ⊓ (ack_write_error → SKIP)
⊓ (ack_write_fatal_error → STOP) ⊓ SKIP ⊓ STOP

REQUEST_WRITE_COMPLETE = complete_write → ack_accept_write
⊓ (ack_accept_write_error → SKIP) ⊓ (ack_accept_error → STOP) ⊓ STOP

REQUEST_READ = accept_read → ack_accept_read
⊓ (ack_accept_read_error → SKIP) ⊓ (ack_accept_error → STOP) ⊓ STOP

READ_DATA = rad → ack_read ⊓ (ack_read_error → SKIP)
⊓ (ack_read_fatal_error → STOP) ⊓ SKIP ⊓ STOP

REQUEST_READ_COMPLETE = complete_read → ack_accept_read
⊓ (ack_accept_read_error → SKIP) ⊓ (ack_accept_error → STOP) ⊓ STOP

DISCONNECT = disconnect → STOP ⊓ (reject → SKIP)

図 6 CSP モデル
Fig. 6 CSP Model

```
class 『サーバ』
types
『client_id』 = nat1;
『ブロック番号』 = nat;

instance variables
connection_list : seq of client_id;
write_list : seq of client_id;
ストレージ : 『ストレージ』

operations
public
connection : () ==> Ack_connection
post len connection_list = len connection_list~ + 1;

public
write : 『ブロック番号』 * ブロックデータ ==> Ack_write
write(ブロック番号, データ) == is not yet specified
pre client_id in elems write_list
post ストレージ(ブロック番号) = ブロックデータ;

private
test_write : 『ブロック番号』 ==> Block_State
test_write(ブロック番号) == is not yet specified
pre ブロック番号 < len ストレージ;

end 『サーバ』
```

図 7 VDM サーバモデル
Fig. 7 VDM server model

```
class 『クライアント』
types
『client_id』 = nat1;
『ブロック番号』 = nat;

instance variables
client_id = 『client_id』;
ストレージ : 『ストレージ』

operations
public
dicconnection : () ==> ()
dicconnection == is not yet specified;

public
write : 『ブロック番号』 * ブロックデータ ==> Ack_write
write(ブロック番号, データ) == is not yet specified
pre client_id in elems write_list
post ストレージ(ブロック番号) = ブロックデータ;

private
test_write : 『ブロック番号』 ==> Block_State
test_write(ブロック番号) == is not yet specified
pre ブロック番号 < len ストレージ;

end 『クライアント』
```

図 8 VDM クライアントモデル
Fig. 8 VDM client model

5. おわりに

本稿では，モデルベースによるソフトウェア開発の上流工程への適用拡大を目的として，仕様記述に自然言語に基づいた DSL を用いることにより，情報科学の非専門家にも理解しやすい仕様記述を提案した．同時に，形式的に記述した DSL の意味による間接的なマッピングにより，仕様の修正に対する柔軟性を損なわずに，検証可能性と可読性を両立させるよう試みた．残された課題として次のようなものがある．

- モデルの切り分け
  フォーマルメソッドの一種である Event-B では，ふるまいモデルを主としながらも，B Method と互換性のある機能モデルも記述できる[2)]．本稿ではこれに対し，ふるまいモデルを CSP，機能モデルを VDM で記述した．異なるモデルに同じ記法を用いると，両者をまたがる修正は容易になるが，モデリングの混乱を招きやすい．
  今回の事例では，隠れた情報である状態を意識することで切り分けられた．状態は，どちらの手法においても重要な要素ではあるが，中心となる概念ではない．しかし，状態を介することで，両手法の記述が交換可能となり，あるいは一貫性の検証に有用であった．
- マッピングにおける意味の定義
  本稿でとりあげた例題は，実システムの極一部にとどまる．実用的なシステムに応用するためには，ある程度の大きさの問題領域に対して， DSL を定義するとともに，フォーマルメソッドへのマッピングの意味定義をする必要がある．
- アーキテクチャ記述
  より上流の仕様により決定されるのは，単なるライブラリでなく，エラー処理などふるまいも含めたアーキテクチャに相当する．アーキテクチャ記述のためには，リアルタイム性など非機能用件や複数の製品の作りわけに対応する記法も重要になる．他のアーキテクチャ記述言語との対照や，問題領域ごとのアーキテクチャ特性の検討を行なわなければならない．

謝辞 本研究の一部は，科学技術振興機構の産学共同シーズイノベーション化事業における平成２０年度採択課題「安心・安全な高信頼性システムを構築するためのソフトウェア要求分析・仕様記述・検証フレームワークの開発」として行なわれたものである．

参 考 文 献

1) Abrial, J.-R.: *The B-Book: Assigning Programs to Meanings*, Cambridge University Press (1996).
2) Abrial, J.-R., Butler, M., Hallerstede, S. and Voisin, L.: An Open Extensible Tool Environment for Event-B, *Proceedings of the 8th International Conference on Formal Engineering Methods*, pp.588–605 (2006).
3) Ambler, S.W.，株式会社オージス総研 (監訳)：アジャイルモデリング，翔泳社 (2003).
4) Denford, M., Leaney, J. and O'Neill, T.: Non-Functional Refinement of Computer Based Systems Architecture, *Proceedings of the 11th IEEE International Conference and Workshop on Engineering of Computer-Based Systems*, pp.168–177 (2004).
5) Dierks, H. and Lettrari, M.: Constructing Test Automata from Graphical Real-Time Requirements, *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, Springer, pp.433–454 (2002).
6) Dietz, C.: Graphical Formalization of Real-Time Requirements, *Proceedings of the 4th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, Springer-Verlag, pp.366–384 (1996).
7) Dong, W., Wang, J., Zhao, C., Zhang, X. and Tian, J.: Automating Software FMEA via Formal Analysis of Dependence Relations, *Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference*, IEEE Computer Society, pp.490–491 (2008).
8) Gerhart, S., Craigen, D. and Ralston, T.: Experience with formal methods in critical systems, *IEEE Software*, Vol.11, No.1, pp.21–28 (1994).
9) Goddard, P.L.: Software FMEA techniques, *Proceedings of Annual Reliability and Maintainability Symposium*, pp.118–123 (2000).
10) Guoqing, W., Fengdi, S., Min, W. and Weiqing, C.: Requirements specifications checking of embedded real-time software, *Journal of Computer Science and Technology*, Vol.17, No.1, pp.56–63 (2002).
11) Hansen, K.M.: Validation of a Railway Interlocking Model, *Proceedings of Industrial Benefit of Formal Methods*, pp.582–601 (1994).
12) Hinchey, M.G. and Bowen, J.P.(eds.): *Applications of Formal Methods*, Prentice Hall (1995).
13) Holzmann, G.J.: The Model Checker SPIN, *IEEE Transactions on Software Engineering*, Vol.23, No.5, pp.279–295 (1997).
14) Jones, C.B.: *Systematic Software Development using VDM*, Prentice-Hall (1990).
15) Macedo, H.D., Larsen, P.G. and Fitzgerald, J.S.: Incremental Development of a Distributed Real-Time Model of a Cardiac Pacing System Using VDM, *Proceedings of the 15th International Symposium on Formal Methods*, pp.181–197 (2008).
16) Needham, D.M. and Jones, S.A.: A software fault tree key node metric, *Journal of Systems and Software*, Vol.80, No.9, pp.1530–1540 (2007).
17) Ogata, K. and Futatsugi, K.: Modeling and Verification of Distributed Real-Time Systems Based on CafeOBJ, *Proceedings of the 16th IEEE international conference on Automated software engineering*, IEEE Computer Society (2001).
18) Pohl, K.，Böeckle, G.，vander Linden, F.J.，林好一 (訳)，吉村健太郎 (訳)，今関剛 (訳)：ソフトウェアプロダクトラインエンジニアリング，エスアイビーアクセス (2009).
19) SESSAME WG 2: 組込みソフトウェア開発のためのオブジェクト指向モデリング，翔泳社 (2006).
20) Schneider, S.: *Concurrent and Real Time Systems: The CSP Approach*, John Wiley & Sons, Inc. (1999).
21) CSK システムズ：http://vdmtools.jp/.
22) 寺田雅人，井上美智子，増澤利光，藤原秀雄：分散移動システムにおける全域チェックポイントについて，電子情報通信学会技術研究報告 comp98-74，Vol.98, No.562, pp.17–24 (1999).
23) 小林吉純：日本語による電話サービス仕様記述における表現の多様性と意味の同一性の認識，電子情報通信学会論文誌， Vol.J82-D1, No.8, pp.1035–1048 (1999).
24) 和田　孝，土田賢省，杉山高弘，阪田全弘，富田兼一，宮下洋一：プログラム自動生成のための日本語仕様記述言語，情報処理学会研究報告, 1988-PRO-016，Vol.1988, No.27, pp.33–40 (1988).
25) 杉尾俊之，武内　惇，椎野　努：日本語をベースにした仕様記述言語 NBSG における手続記述について，情報処理学会研究報告, 1983-SE-034，Vol.1983, No.55, pp.73–78 (1984).
26) 情報処理推進機構：２００７年版組込みソフトウェア産業実態調査　報告書，技術報告，独立行政法人 情報処理推進機構 (2007).
27) 大石東作：ソフトウェアの「日本語による仕様書」作成支援システム，情報処理学会研究報告, 1988-SE-059，Vol.1988, No.32, pp.1–7 (1988).
28) 日経エレクトロニクス（編）：組み込みソフトウエア２００７　モデルに基づく開発方法論のすべて，日経 BP (2006).
29) 日本規格協会（編）：JIS ハンドブック 情報記録媒体，日本規格協会 (2007).
30) 西田富士夫，高松　忍，谷　忠明：要求仕様における日本語表現と形式表現間の相互変換，情報処理学会論文誌， Vol.29, No.4, pp.368–377 (1988).
31) 小松由香里，吉原真也，石橋慶一，秋山義博，中谷多哉子，片峯恵一，鵜林尚靖，橋本正明：QFD による組込みソフトウェア分析・設計の品質管理モデリングに関する一考察，プロジェクトマネジメント学会誌， Vol.7, No.5, pp.15–20 (2005).
32) 山科隆伸，森崎修司，飯田　元，松本健一：保守開発型ソフトウェアを対象としたソフトウェア FMEA の実証的評価，ソフトウェア品質シンポジウム 2008 発表報文集，pp.157–164 (2008).