小規模ソフトウェア開発における形式手法の活用

堀　武司

Utilization of Formal Methods on Small-scale Software Development

Takeshi HORI

キーワード：形式仕様記述，モデル検査

1．はじめに

北海道には主に札幌周辺を中心としてソフトウェア開発などのIT産業が集積しており，年間売上高 約3,800億円，従業員数約２万人の規模を有する[1)]。一方，そのかなりの部分は小規模な事業者で占められている。これらの小規模事業者におけるソフトウェア開発は，少数の技術者により「職人」的に行われる場合が多い。そのため，開発プロセスの管理が十分に行われておらず，製品の品質保証は個々の技術者の力量に委ねられている点が問題である。これらの問題を解決するための正攻法は，ISO9000やCMMIなどの枠組みに従い社内の開発プロセス改善をすすめることであるが，人員と予算の制約が大きい小規模な開発現場では,文書作成等の間接的作業負荷が増大するプロセス管理体制の導入は必ずしも現実的ではない。

著者は，上述の小規模ソフトウェア開発において品質向上を実現するための方策の一つとして，ソフトウェア開発の新しい技術である「形式手法」の活用の可能性に着目した。

形式手法とは，ソフトウェアの仕様定義や設計などを，数学や論理学などを基盤とする技術によって行うことで，高品質なソフトウェア開発の実現を目指す手法の総称である。形式手法の中には，ソフトウェアの仕様定義や設計を自然言語や図表の代わりに，より厳密な数学的表現で記述することで曖昧さを排除する「形式的仕様記述」や，設計内容の検証を数学的理論とコンピュータ支援ツールを用いて自動的・網羅的に行う「形式検証」などの技術が含まれる。

これまで，形式手法は主に極めて高い信頼性が要求される航空宇宙，運輸，医療などの分野では使われてきたものの，手法導入のコストが高いこと，高度な専門知識を持つ技術者の確保が難しいことなどから，一般のソフトウェア開発における利用はまだ広がっていない。しかしながら，検証支援ツールの整備，検証に使われるコンピュータ自体の能力向上などにより，一般的な開発現場でも形式手法の活用が現実的なものとなってきた。

本研究では，北海道内の中小ソフトウェア企業等におけるソフトウェア開発に対して，形式手法の技術普及を推進し品質と生産性の向上を図ることを目的として，基礎的な調査や評価，および技術者向け教材の開発に取り組んだ。

2．中小企業での導入に適した手法・ツールの調査

形式手法と呼ばれる手法は，広く知られているものだけでも100種類を越え，その技術的内容や得意とする適用対象も様々である。また，日本語で得られる現場向けの技術情報もまだ少ない。そのため，中小企業の技術者が形式手法の導入を検討しようとした場合，どの手法から始めるべきかを判断することは容易ではない。

そこで著者は，中小企業への導入に際してどのような手法が適切かを判断し，今後取り組みを進める手法の絞り込みを行うため，主要な形式手法について，手法の概要，適用分野，および支援ツールや技術情報の整備状況などに関する調査を行った。同様の調査は既にいくつか公開されている[2)]が，今回の調査では特に，中小ソフトウェア企業での技術導入を前提とし，以下の観点に留意して調査を行った。

- 習得が比較的容易であり，何らかのプログラム言語を習得した平均的技術者が，無理なく理解できる水準の難易度であること。
- 日本語による技術情報がある程度整備されており，国内における研究会活動などが活発であること。
- 手法の実践に必要な支援ツール等が，オープンソース，無償，もしくは安価で入手できる。

事業名：経常研究
課題名：中小規模ソフトウェア開発への形式手法導入に関する研究（22～23年度）

表１　主要な形式手法の比較

手法	記述方法	支援ツール (*は無償利用可)	検証技術	習得の難易度	国内での技術情報	備考
VDM	VDM 言語	VDMtools * Overture *	アニメーション	低	比較的豊富	国内で最も普及
Z 記法	Z 記法	ZETA* CZT*	-	中～高	比較的豊富	ツール環境が弱い
B メソッド	B 言語	Atelier B * ProB *	定理証明 モデル検査 アニメーション	中～高	入門書一冊	強力な検証機能 記述制約が厳しい
SPIN	専用言語 (Promela)	SPIN *	モデル検査	中	比較的豊富	
UPPAAL	状態遷移図 (GUI)	UPPAAL	モデル検査	低	入門書一冊	GUI を備える 時間制約を取扱う事が出来る
CSP	CSP (プロセス代数)	FDR PAT * CSP-Prover	モデル検査 定理証明	中～高	入門書一冊	国内でのコミュニティ活動が活発

これらの条件を考慮した上で，いくつかの手法を候補として選択し，調査を行った（表１）。

2.1　形式仕様記述言語

一般的なプログラム言語の概念に近い形で，ソフトウェアのデータ構造や関数の機能仕様の定義などを扱うための手法として，VDM，Z記法，Bメソッドなどに代表される「モデル規範型（model-oriented）」と分類される手法がある。これらの手法は，いずれも論理学と集合論による数学的な基盤に基づいており，記述する対象を「状態」とそれに対する「操作」の観点から表現する。この考え方は，状態を「変数」，操作を「関数，メソッド」と言い換えればC，FORTRAN，Javaなどの手続き型プログラム言語と同じであり，一般のソフトウェア技術者にも理解しやすい手法といえる。これら３手法に関する調査結果を以下に示す。

⑴　VDM

VDMは，モデル規範型の代表的な手法であり，形式仕様記述手法として最も普及が進んでいる手法の一つである。国内の産業界でもFelica ICカードに関するものなど[3]いくつかの適用事例が報告されている。VDMに関する支援ツールとしては，SCSK社のVDMtools，オープンソースのOvertureなどがあり，特に前者は国内企業による開発であるためサポート体制等も整っている。これらのツールでは，モデルの文法やデータ型のチェック機能に加えて，インタプリタ上でモデルを動作させること（仕様アニメーション）が可能である。アニメーションによるモデルの動作確認やテストが，VDMにおける主要な検証手段である。アニメーションによる検証は直感的に分かりやすい技術であるが，完全に網羅的な検証をテストで実現することは困難であり，検証能力の面では他の手法よりも弱い。

⑵　Z記法

Z記法は，Jean Raymond Abrialらによって提案された手法であり，VDMと並び代表的な形式仕様記述言語として知られている。ただし，中小企業への技術導入を前提に考えた場合，全般的に学術研究寄りであること，支援ツールのサポートが弱いことなどの理由から，検討候補からは除外した。

⑶　Bメソッド

Bメソッドは，Z記法と同じくAbrialが提案した手法であり，数学的背景はVDMやZ記法と共通であるが，より仕様の検証の側面を重視した手法である。Bメソッドでは，モデルから論理学の証明問題（証明責務）を生成し，これを定理証明系で証明することでモデルの正しさを保証する。テストによる検証とは異なり，検証された事柄に関しては100%正しいことが数学的に保証される。また，Bメソッドは単なる仕様記述だけの手法ではなく，仕様から設計，実装を導出する過程の正しさを証明によって保証することで，最終的に正しいプログラムを得るという考え方（Correct by Construction）の実践を目的としている。

Bメソッドの支援ツールとしては，フランスClearSy社のAtelier-Bがあり商用利用も含め無償で利用できる。Atelier

Bでは，B言語による仕様記述，内蔵された自動定理証明系によるモデルの証明，Cなどのプログラム言語へのコード変換など，Bメソッドによる開発作業を一貫して実施できる。

以上の結果より，手法自体の難易度の低さ，汎用性の高さの観点から，VDMを主要な手法の一つとして選択した。

候補の一つとして検討したBメソッドは，良好な支援ツール環境と強力な検証機能の面で非常に優れているが，その一方でモデル記述の自由度に大きな制約がある。特に，文字列等の可変長データの扱いや，モジュール間をまたいだデータ書き換え操作に関する制約が厳しく，業務系ソフトウェアなどを含む幅広いソフトウェア開発への適用は，現時点では難しいと判断した。

2.2 モデル検査手法・ツール

モデル検査は，対象の振る舞いに関する仕様検証を目的とした形式手法の一分野であり，近年，組込みソフトウェア分野などを中心に産業界での普及が急速に広がっている手法である。

モデル検査では，対象を状態遷移モデルとして表現し，そこである性質が成り立っているかどうかを，コンピュータ上の支援ツールによってモデルの状態空間を網羅的に探索して確認する。モデルの状態空間全体を網羅的に検査できれば，定理証明による検証と同様，検証された性質が100％正しいことが保証される。ただし，ある程度以上複雑なモデルをそのまま記述すると，状態空間が爆発的に増大してしまい網羅的検証は困難になる。そのため，検証したい内容に合わせてモデルを抽象化し状態数を削減するなどの工夫が必要になる。

モデル検査の代表的な手法，ツールとしては，SPIN，UPPAAL，PAT などがある。これらのツールは，いずれも

- 状態遷移モデルを対象とすること
- 複数のプロセスが相互に関連しながら並行に動作する「並行プロセス」の記述ができること
- 時間の概念について拡張された論理学である時相論理を用いて，検証条件を記述すること

などの点で共通しているが，モデルの記述方法やツールの使い勝手の面では差異がある。

SPINでは，C言語に似た専用の言語（Promela）を用い，手続き型プログラム言語に近い形でモデルを表現する。

一方，UPPAALは標準でグラフィカル・ユーザ・インターフェース（GUI）を備えており，状態遷移図をツール上で直接描画できる。また，モデルのアニメーションや検証結果の表示もGUI画面上で対話的に行うことができる。

PATは，並行プロセスを扱うための数学体系（プロセス代数）の一種であるCSP（Communicating Sequential Processes）を用いてモデルを記述する。モデルの記述はテキストベースであるが，アニメーションや検証結果の表示はGUI上で行うことができる。

上記以外の特長としては，UPPAALとPATでは，離散的な状態遷移だけではなく，時間付きオートマトンによって時間の概念を直接取り扱うことができる点が挙げられる。これらの機能は，リアルタイムシステムにおける時間制約の検証などに特に有用である。

これらのツールはいずれも実業務への適用に十分な能力を持っているが，状態遷移図を直接扱えることによる直感的なわかりやすさを重視し，UPPAALを当面の主要なツールとして選択した。

ただし，UPPAALは比較的高額な商用製品であり，その点が中小企業への導入の障害として懸念されるため，長期的にはPATなど他のツールへの移行も検討している。PATの最新リリース版（v3.5.0）ではUPPAALに類似したGUIベースのモデリング機能なども追加されており，UPPAALで蓄積した技術ノウハウが活用できると思われる。

3．実業務への適用試験

2章で選定した手法に関して，実際の開発業務における有効活用の可能性について，更に詳細な検討をすすめるため，いくつかの具体的な開発事例への適用試験を行った。

3.1 形式仕様記述手法の適用試験

形式仕様記述手法の適用試験の一つとして，過去に著者が関わった開発事例の一つである橋梁点検ロボット車「橋竜」[4]（図１，㈱帝国設計事務所）を題材として，VDMによる仕様記述を試みた。

「橋竜」は，ビデオカメラ等を搭載したロボットアームによって橋梁桁下部の遠隔点検を行うシステムであり，著者は，カメラからの点検画像データとロボットアーム等から得られた位置座標データを自動的に記録・管理し，３次元CGによる可視化表示，点検報告書の自動生成などを行う「点検作業支援ソフトウェア」（図２）の試作開発を分担した。

対象ソフトウェアの大まかなアーキテクチャを図３に示す。ロボットとの通信処理などが含まれている以外は，概ね業務系ソフトウェアに近い内容である。これらのコンポーネント毎に，VDMによる仕様記述の可能性を検討し，可能な部分については実際の仕様記述の試行を行った。

「点検データ管理」コンポーネントは，点検作業で得られる各種データ（写真，位置情報，時刻，コメント等），点検対象の橋梁の基礎データなどを管理し，データ構造，データ間の関連，およびデータの参照・更新操作などを実現している。過去の開発では，設計にUMLクラス図やE-R図(図４）を用いていたが，それらと同等の内容をVDMによって書き直すことができた。また，記述作業においては，VDM自体の制約によって意図通りの記述が出来なくなる，といったこともほとんどなく，十分な記述能力を有していることが確認

できた。

一方，それ以外のコンポーネントに関しては，VDMによる記述の適用は難しいと判断した。例えば「画面表示」コンポーネントの場合，３次元CG処理のためのライブラリ Java3Dを利用するための処理が大半を占めており，独自のロジックは少ない。これらの部分に対するVDMの適用は可能ではあるものの，得られるメリットが少ない。

本ソフトウェアのアーキテクチャ設計では，アプリケーション固有のデータ処理は，表示やユーザ・インターフェースに関する処理とは分離され，「点検データ管理」コンポーネント内に集約されている。そのため，記述すべき仕様も当該コンポーネントに集約されており，この部分のみへのVDMの適用でも十分メリットは大きいといえる。

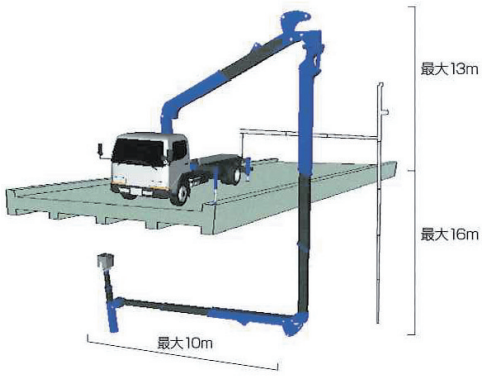

図１　橋梁点検ロボット車「橋竜」

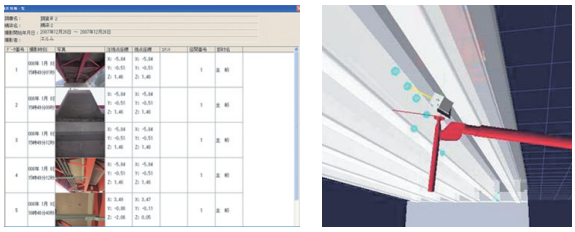

図２　点検作業支援ソフトウェア

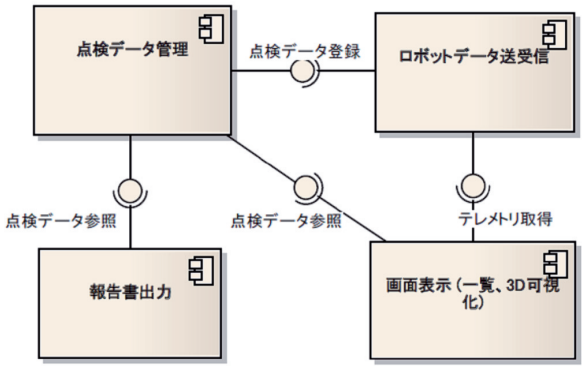

図３　点検作業支援ソフトウェアのアーキテクチャ

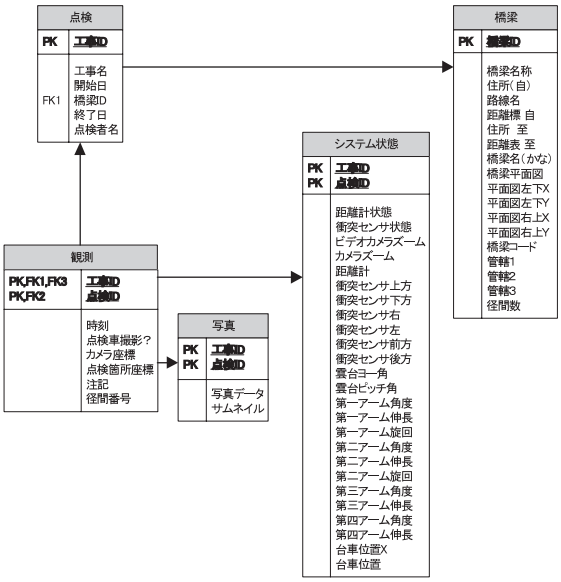

図４　橋梁点検データの構造の一部

3.2　モデル検査ツールの適用試験

組込みマイコン上などで動作する制御ソフトウェア開発に対する形式手法の適用試験の一つとして，NEXCESS名古屋大学組込みソフトウェア技術者養成プログラム[5]の教材に含まれる例題「カップラーメンタイマ」を題材として，モデル検査ツール UPPAAL の適用試験を行った。

例題「カップラーメンタイマ」では，２つのスイッチと２つのLEDを備えたマイコン上で，タイマの起動・停止や，LED表示によるタイムアウト通知などの制御を行うソフトウェアを開発する。

例題の設計では，リアルタイムOS上で並行動作する２つのタスク（タイマ，LED点滅）と４つの周期ハンドラによりシステムの機能を実現しており，各タスクの設計は状態遷移図として記述されている（図５）。

これらの設計文書に基づき，タスク，ハンドラの状態遷移モデルをUPPAALによる並行プロセスとして記述を行った。タイマ制御タスク，およびLED点滅タスクを記述したUPPAALモデルをそれぞれ図６，図７に示す。いくつかの中間状態が追加されているが，概ね元の状態遷移図の構造をそのまま反映した形でUPPAALプロセスを記述することができた。

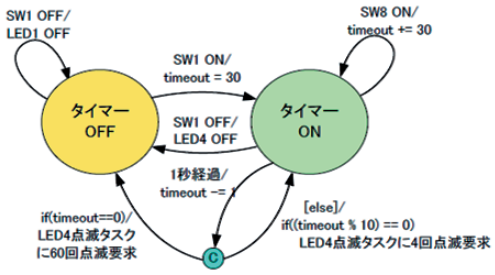

図５　タイマ制御タスクの状態遷移図

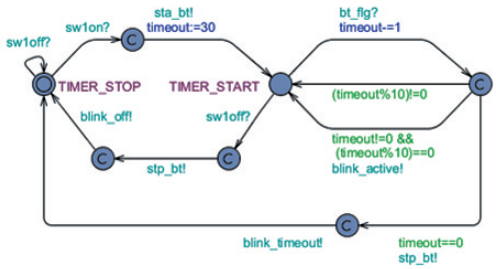

図 6　UPPAALによるタイマ制御タスクの記述

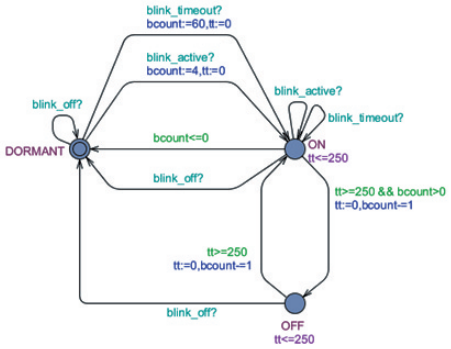

図 7　UPPAALによるLED点滅タスクの記述

並行動作するUPPAALプロセス間の同期処理は，チャネルを用いたプロセス間通信として記述した。チャネルの通信処理は次のように記述される。

送信側 channel_name!

受信側 channel_name?

チャネルによる通信は同期型通信であり，送受信双方が実行可能となった場合のみ実行され，それ以外の場合は処理がブロックされる。

元の例題では，タスク，ハンドラ間の通信はμITRON OSの通信機能であるイベントフラグ等で実装されているが，UPPAAL上ではこれらのイベントフラグに対応するチャネルを用い，ほぼ同じ構造でモデルを記述することができた。

これらの結果から，リアルタイムOS上で実装されたソフトウェアの設計は，その構造を保ちつつ比較的容易にUPPAALモデルとして記述できることが確認できた。この事例では既に存在する設計や実装から形式モデル記述を行ったが，UPPAALのモデルをμITRON OSのタスクや通信機能を用いて実装することも容易である。

4．教材開発と技術者教育の試行

形式手法の技術導入において最も大きな課題の一つは，形式手法を習得し，使いこなすことができる技術者の育成である。近年は国内においても産業界向けの入門書や教育コースが充実してきているが，中小企業の技術者向け教材としてはまだ十分とはいえない。著者は，道内中小企業における形式手法の実践的な普及促進を図るための活動の一環として，道内の中小企業技術者を対象とした形式手法の教育コースのための教材開発を行った。

4.1　教材のテーマと概要

教材では，検査ツールによって設計不具合を発見できる，という導入効果が明快でわかりやすいこと，検査作業はコンピュータが全自動で行うため検査に関する高度なスキルが不要であることといった特長から，初めて形式手法に取り組む受講者にも受け入れられやすいモデル検査について取り扱うこととした。また，具体的な手法，ツールとしては，GUIを備えており直感的に理解しやすいUPPAALを選択した。

コースの内容は半日～１日程度で実施することを想定し，形式手法自体の概論的な内容は最小限とし，実際にUPPAALツールを使用した実習作業を中心とする現場の技術者向けの構成とした。カリキュラムの構成を以下に示す。

- 形式手法の概要
- UPPAALにさわってみよう！（ツールの使い方）
- プロセスの記述
- チャネル通信
- 時間制約の記述
- 検証式の書き方と時相論理
- 実習課題: 哲学者の食事問題のモデリング
- UPPAAL適用事例の紹介

4.2　実習用例題：哲学者の食事問題のモデリング

受講者がモデル検査技術とUPPAALの特長を一通り体験できるように配慮したコンパクトな実習用例題として，並行処理の有名な問題である「食事する哲学者」を題材とする例題を作成した。

「食事する哲学者」問題では，図 8 のようにN人の哲学者とN本のフォークが円卓に配置された場面を考える。哲学者はそれぞれ任意のタイミングで思索と食事の動作を繰り返す。食事の際に，哲学者は左右のフォークを順に手に取り，食事が済むとフォークを卓上に戻す。フォークは両隣の哲学者により共有されるため，共有リソースの排他制御に関する例題として取りあげられる。

図 8　「食事する哲学者」問題

例題では3人の哲学者と3本のフォークをそれぞれ並行プロセスとしてモデリングした（図9）。

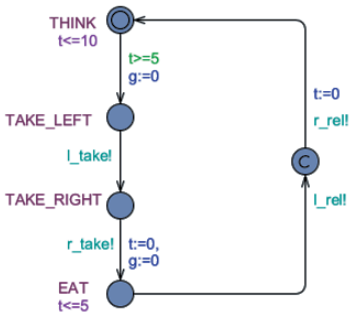

図9　哲学者のUPPAALモデル

「食事する哲学者」問題では，全ての哲学者が同時に左フォークを取得した場合にデッドロック（その先の動作が一切進まなくなる状態）に陥ることが知られている。対策を行っていないモデルに対してUPPAALの検証機能を用いると，デッドロック発生が検出され，さらにその時点までの実行履歴を示すシーケンス図(図10）が出力される。

さらに，この例題ではUPPAALの時間制約機能の活用例として，哲学者の思索，食事の時間をそれぞれ5～10分，0～5分に設定している。このモデルに対してUPPAALの検証機能を使うと，例えば「ある哲学者が食事を取れるまでの待ち時間は，最長でも15分である」といった，システムの最悪応答性能に関する性質の検証を試すことができる。

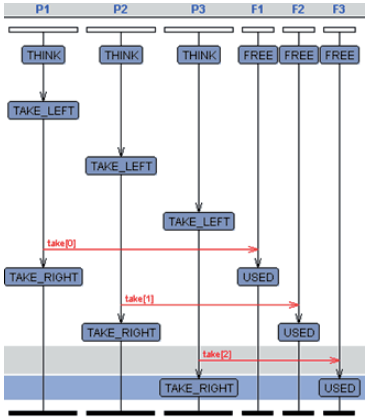

図10　デッドロック発生までの実行履歴

4.3　技術セミナの実施と評価

開発した教材の評価と，道内企業への技術普及を目的として，「ソフトウェアモデル検査技術研修」を実施した（国際競争力の強化に向けた技術系人材育成事業：苫小牧市テクノセンター，平成23年1月）。

研修の受講者は15名であったが，1日の座学と実習により，ほぼ全員がUPPAALによる簡単なツールの使用法を理解し，用意した例題のモデル作成と検証作業を完了することができた。

研修終了後に行った受講者アンケートの結果では，大半の受講者から「状態遷移モデルに対して検証ができる点が良いと思った」という主旨のコメントがあり，実際の開発現場においても状態遷移モデルに基づく設計が広く浸透しており，それらとの親和性の高いモデル検査に対するニーズが高いことが確認できた。

5．まとめ

北海道内の中小企業における小規模ソフトウェア開発に対して形式手法の導入を推進するため，主要な形式手法・ツールの調査と評価，および技術者育成のための教材開発などの取り組みを実施した。

形式手法は技術的難易度や導入コストが高いとされてきたが，形式手法の適用試験や教材を用いた技術研修の結果から，中小企業技術者が形式手法を習得することは十分可能であること，開発現場における形式手法に対する潜在的ニーズが十分に高いことが確認できた。

今後は，業務系ソフトウェアに対するVDM，組込み制御ソフトウェアに対するUPPAAL，PATなど，比較的導入しやすいと判断された手法・ツールに絞って教材等の整備を行うとともに，道内企業との連携により，実際のソフトウェア開発業務を適用対象とした事例研究を推進していく予定である。

6．引用文献

1）一般社団法人北海道IT推進協会編，北海道ITレポート2012，http://www.hicta.or.jp/report/pdf/2012.pdf，（2012）

2）独立行政法人情報処理推進機構ソフトウェア・エンジニアリング・センター，形式手法適用調査，http://www.ipa.go.jp/sec/reports/20100729.html，(2010)

3）栗田太郎・太田豊一・中津川泰正．：モバイルFeliCa ICチップ開発における形式仕様 記述手法導入の効果と課題．ソフトウェアシンポジウム2005 論文集，pp. 73-80，ソフ トウェア技術者協会，（2005）

4）堀武司・中西洋介・波通隆：橋梁点検システムの開発，工業試験場報告 No.308，pp.7-14，(2009)．

5）NEXCESS 名古屋大学組込みソフトウェア技術者養成プログラム，http://www.nces.is.nagoya-u.ac.jp/NEXCESS/，(2008)