独立行政法人情報処理推進機構 委託

2012 年度ソフトウェア工学分野の先導的研究支援事業
「実用性が高い形式工学手法と支援ツールの研究開発」
成果報告書

平成 25 年 1 月

学校法人法政大学

本報告書は独立行政法人情報処理推進機構　技術本部　ソフトウェア・エンジニアリング・センターが実施した「2012 年度ソフトウェア工学分野の先導的研究支援事業の公募による採択を受けて法政大学情報科学研究科（研究責任者　劉　少英）が実施した研究の成果をとりまとえたものである.

目次

研究成果概要 ····· 1

１.研究の背景および目的 ····· 10
1.1 背景 ····· 10
1.2 研究課題 ····· 11
1.2.1 形式手法の適用への挑戦的な課題 ····· 11
1.2.2 形式工学手法の提案 ····· 12
1.2.3 SOFL 形式工学手法の改善策 ····· 16
1.3 研究の意義 ····· 18

２.実施内容 ····· 19
2.1 研究アプローチ ····· 19
2.1.1 研究の全体像 ····· 19
2.1.2 関連するこれまでの研究について ····· 23
2.1.3 研究目標 ····· 27
2.2 研究の活動実績・経緯 ····· 28
2.2.1 研究活動実績 ····· 28
2.2.2 内部・外部打合せの実施状況 ····· 28
2.3 研究実施体制 ····· 33

３.研究成果 ····· 35
3.1 研究目標 1「非形式仕様からアニメーションシステムのアーキテクチャーの構成方法の確立」 ····· 35
3.1.1 当初の想定 ····· 35
3.1.2 研究プロセスと成果 ····· 35
3.1.3 実用化へ向けた課題と問題点 ····· 44
3.2 研究目標 2「半形式仕様からアニメーションシステムのアーキテクチャーの構成方法の確立」 ····· 45
3.2.1 当初の想定 ····· 45
3.2.2 研究プロセスと成果 ····· 46
3.2.3 実用化へ向けた課題と問題点 ····· 52
3.3 研究目標 3「形式仕様パターンの定義と仕様パターンシステムの構造の確立」 ·· 53
3.3.1 当初の想定 ····· 53
3.3.2 研究プロセスと成果 ····· 54
3.3.3 実用化へ向けた課題と問題点 ····· 60
3.4 研究目標 4「形式仕様パターン知識の表現，検索，および適用仕組みを提案する」 ····· 60
3.4.1 当初の想定 ····· 60

3.4.2 研究プロセスと成果 ····· 61
3.4.3 実用化へ向けた課題と問題点 ····· 70
3.5 研究目標 5「形式仕様パターンシステムを支援するツールの一部の作成」 ····· 70
3.5.1 当初の想定 ····· 70
3.5.2 研究プロセスと成果 ····· 71
3.5.3 実用化へ向けた課題と問題点 ····· 78
3.6 研究目標 6「CDFD から機能シナリオを自動的に生成するツールの作成」 ····· 78
3.6.1 当初の想定 ····· 78
3.6.2 研究プロセスと成果 ····· 79
3.6.3 実用化へ向けた課題と問題点 ····· 89

４.考察 ····· 91
4.1 研究により判明した効果や問題点等 ····· 91
4.1.1 設定した到達目標の達成 ····· 91
4.1.2 他の類似研究との比較 ····· 95
4.1.3 事例研究による提案した SOFL 技術の評価 ····· 96
4.2 今後の課題 ····· 102
4.2.1 技術と支援ツールの成熟度に関する課題 ····· 102
4.2.2 新たに見いだされた課題 ····· 104

参考文献 ····· 106

研究成果概要

１.研究の背景

近年，ソフトウェアシステムに潜む欠陥は，ATM の障害や自動車，携帯電話のリコールなど，システムの信頼性・安全性の問題を引き起し，延いてはシステムトラブル，ダウンによる莫大なエネルギー損失と，大きな環境問題，社会問題に発展しかねない．この状況になっている主な原因は，ソフトウェアの大規模化に伴い，従来の開発方法では品質を維持できないことが最大の原因といえる．ソフトウェア開発プロセスの上流アクティビティ（要件分析，仕様，設計など）に使われている自然言語や図や表などは，明確な文法または意味論を定義していないため，曖昧性や意味不明の表現などが要求仕様，設計仕様など上流文書に含まれている．このため，上流文書の整合性・正当性の検証や仕様からプログラムへの変化や作成されたプログラムの検証などことは，正しく効率的に行うことが困難である．

形式手法は，数学や論理学に基づく形式的体系を用意し，その体系に基づき，システムの記述や分析を行うものである．適切に使えば，形式手法により開発されたプログラムに対してテストの必要性がない価値があると言われる．これはあくまで理論上の話で，実際のソフトウェア開発プロジェクトに適用する際，様々な挑戦的な課題が残っている．
それら課題を解決するために，SOFL 形式工学手法が提案された．形式手法と違い，形式工学手法が目指しているのは，形式仕様記述技術の利用によって，企業ですでに定着している要求分析と設計の図言語，開発手法，テスト，インスペクション（inspection）など技術の系統性，厳密性，有効性，および自動化程度を向上させることである．特に，SOFL 三段階形式仕様記述技術が，要求分析のための非形式仕様と半形式仕様の作成によって形式設計仕様を作成する系統的なアプローチを提供している．

２.研究課題

SOFL 三段階技術は，既存の形式手法と比べて実用性が高くなっているが，三つの重要な課題が残っている．第一に，作成された非形式仕様，半形式仕様，および形式仕様の内容をどのように顧客に容易に確認してもらい，必要なフィードバックを獲得できるか．第二に，形式手法を上手に使えない開発者に対して，どのように補助すればシステム要求と設計のアイディアを明確に理解しながらそれを反映する形式仕様を容易に作成できるか．第三に，SOFL 形式工学手法をどのように効率的かつ容易に応用できるか．

３.研究成果

これら問題点を解決するために，本研究では，SOFL 三段階形式仕様記述技術を始めとするこれまでの研究実績をさらに発展させ，仕様アニメーション手法および形式仕様パター

ンに基づく操作の事前条件と事後条件を作成するアプローチを提案，これら手法とアプローチを SOFL 三段階形式仕様記述プロセスに統合することによって,漸進的な形式仕様作成手法を確立，更にこの手法を効率的に支援するソフトウェアツールのプロトタイプを開発した．本研究成果報告書において，仕様アニメーションとは仕様に記述された機能，データ項目，および制約を動的に表現することである.具体的には実施方法により，非形式仕様アニメーション，半形式仕様アニメーションおよび形式仕様アニメーションに分けているが，共通点は仕様の内容を動的に表現することである．

3.1 非形式仕様アニメーション

非形式仕様アニメーションとは，SOFL 非形式仕様に記述されているシステムの機能（function)，データリソース，および制約（constraints）を，現実世界の仮想的な環境で動的に表現することである．目的としては，作成された非形式仕様の内容を，顧客に確認してもらい，顧客から必要なフィードバックを獲得し，非形式仕様を徐々に完成させることである．アニメーションが顧客の完全な要求を発見することに役に立つ技術になるためには，非形式仕様を作成することを伴いアニメーションを徐々に行う漸進的なアプローチを提案した．この中で，研究の焦点は，どのように非形式仕様によってアニメーションシステムを作成できるかという課題である．

これに対しては，次のスデップを提案した．

(a)非形式仕様を作成する．

(b)非形式仕様に基づき，非形式仕様で記述されたシステムの「機能構造図」を構築する．

この構造図は，顧客が一番関心をもっている機能らを中心に選出し，それをアニメーション化するために必要な GUI 部品とその機能を実現するために必要な内部機能も適切に利用する関係を示す．また，機能の実現に必要なデータリソースと制約も適切に示す．

(c)機能構造図によって非形式仕様のアニメーションシステムを作成する.顧客が一番関心を持っている機能に対して，適切な GUI を作成，必要なインタフェースも作成，開発者の理解によってその機能の意味を反映するアニメーションを作成する．

(d)非形式仕様のアニメーション化を行う．

(e)顧客からのフィードバックをもらい，非形式仕様を修正する．

3.2 半形式仕様アニメーション

SOFL 半形式仕様は，非形式仕様を詳細化して得たユーザの要求をより明確に定義している仕様であり，関連している機能，データリソース，および制約を SOFL モジュールにまとめている．構造としては，SOFL 半形式仕様が，SOFL モジュールの集合である．

半形式仕様アニメーションとは，SOFL 半形式仕様に含まれているモジュールにおいて定義されているプロセス（操作）の機能を，一つずつ動的に表現することである．目的としては，半形式仕様で定義されたプロセスのインタフェース，入力変数，出力変数，これら型，外部変数と型，および期待されている振る舞いを，開発者（つまり，半形式仕様の作成者）と顧客に確認してもらい，必要なフィードバックを獲得し，半形式仕様を，期待される形に改善する．

アニメーションが開発者と顧客の要求した通りのプロセス機能を正しく定義することに役に立つ技術になるためには，半形式仕様を作成することを伴いアニメーションを徐々に行う漸進的なアプローチを提案した．この中で，研究の焦点は，プロセスの入出力の値，外部変数の値，および振る舞いをどのように動的に表現すると開発者と顧客が分かりやすくなるのか研究課題である．

これに対しては，次の手順を提案した．

(a)非形式仕様に基づき，半形式仕様を作成する．

(b)半形式仕様に含まれるモジュールを，一つずつ選出する．一つのモジュールに対しては，その中に含まれているプロセス仕様を，一つずつ選出する．

(c)一つのプロセス仕様に基づき，そのプロセスのインタフェース，入出力の値，外部変数の値，および振る舞いを動的に表現するアニメーションシステムを作成する．

(d)プロセス仕様のアニメーションを行う．

(e)開発者と顧客からのフィードバックをもらい，半形式仕様を修正する．

3.3 形式仕様アニメーション

SOFL 形式仕様は，基本的には階層的なモジュールと階層的な CDFD（Condition Data Flow Diagram）から構成されたものである．CDFD は，イベント駆動の操作意味論を持つ形式的データフロー図であり，システムのアーキテクチャーをデータの流れの観点から描画する．CDFD に対応するモジュールには，その CDFD の各部品（例えば，プロセス，データフロー，データストア）を，SOFL 形式仕様記述言語で明確に定義される．

形式仕様アニメーションとは，CDFD から「システム機能シナリオ」を導出し，それの振る舞いを，入力によって出力を出すプロセスを動的に表現することである．目的としては，開発者と顧客に，形式仕様で設計したシステムの機能シナリオを確認してもらい，システム機能シナリオの整合性も確保することである．この中で，主な研究課題としては，CDFD からどのようにシステム機能シナリオを自動的に導出するか，導出されたシステム機能シナリオの振る舞いをどのように動的に表現するか，入力と出力変数の値は，どのように生成できるかという問題点が含まれている．このような課題を全て解決するためには，長い期間の研究が必要である．本研究では，CDFD からシステム機能シナリオの自動導出技術だけの研究を中心として取り込んできた．

この点に関しては，CDFD をグラフとして扱い，CDFD の「開始プロセス」から入力データフローを確定，CDFD の「最終プロセス」から出力データフローを確定，その入力データフローから出力データフローへ到達するデータフローパスは，システムの機能シナリオとして形成する．具体的な生成手法とアルゴリズムは，研究成果の節で詳しく紹介する．

3.4 形式仕様パターンに基づくプロセスの形式仕様作成アプローチ

形式仕様パターンとは，形式仕様記述言語の知識とその知識を適用した経験を反映する一定の種類の論理式を作成するための仕様テンプレートである．一般的には，一つの仕様パターンに，次の四つの部分が含まれている．

(a)パターン名

(b)パターンの説明
(c)パターンの構成要素
(d)パターン解答（solution).

パターン名は，該当するパターンが表す論理式の性質を反映する名前である．その名前を見ると，このパターンの適用によってどんな論理式を生成することが分かる訳である．パターンの説明は，該当パターンの目的を自然言語で説明する．パターンの構成要素は，期待される論理式を作成するために必要なコンポーネントを説明する．最後に，パターン解答は，該当パターンの構成要素の型によってよく使われている形式的論理式を示す．これら論理式の中で一番適切なもろを選択する．

コンピュータプログラムの機能を表現するには，形式仕様パターンは，「関係」，「情報検索」，および「情報更新」という三種類に分けることができる．「関係」というパターン種類には，二つの数式間のあらゆる関係（例えば，＞，＜，＝）を表現する論理式を形成するパターンを含める．「情報検索」というパターン種類には，必要な情報を検索する意味を表す論理式を形成するパターンを含める．「情報更新」というパターン種類には，あるデータ構造に保存されている情報を一定の方法で更新する意味を表現する論理式を作成するパターンを含める．

本研究で提案した形式仕様パターンに基づく形式仕様を作成するアプローチの主な目的は，このアプローチを支援するツールを通じて，ユーザ（開発者）と自然言語で対話することによってユーザの明確な機能要求を確認した上で，その機能を表現する形式的論理式を自動的に作成することである．これを実現するためには，仕様パターンを知識としてコンピュータに保存しておいて，支援ツールの GUI を通じてユーザと自然言語で対話しながら，コンピュータアルゴリズムがそのパターンを適用し，ユーザに適切な指示を出し，必要な入力を求め，このようにユーザとコンピュータが対話しながら，最終的に形式的論理式を作成する．この間，ユーザは SOFL のような形式仕様記述言語を直接に使う必要がないので，一般の実務者によっては，より簡単に形式仕様を作成することができる．

この技術の中で，コンピュータからユーザに適切な指示を出せるためには，仕様パターン知識を適切な形式でコンピュータに保存することが大事である．ユーザ等の対話によって形式仕様を漸進に形成するプロセスは，イベント駆動システムの一種であるため，有限状態遷移図（FSM）を用いて，全ての仕様パターン知識を保存する形式を採用している．ユーザの入力は，イベントとして扱い，それによって適切な状態遷移を行う．一つの状態に到達すると，その状態から次の状態を選択するために必要な入力を求める指示をユーザへ出す．このように対話しながら，最終的に形式仕様を作成する．

3.5 SOFL 三段階漸進的な形式仕様記述手法

高信頼ソフトウェアシステムを開発するためには，上質な要求と設計仕様が極めて重要である．上質な要求仕様とは，記述した全ての要求機能，データリソースおよび必要な制約は顧客が希望したものと完全に一致することである.同じように,上質な設計仕様とは，正当性を持つ設計者の意図と完全に一致する設計仕様と意味する．問題点としては，どのようにすれば，上質な要求仕様と設計仕様を素早く作成することができるかということで

ある．この問題点は，ソフトウェア産業にとって非常に重要であるが，まだ解決していない．

本研究では，既存の SOFL 三段階形式仕様記述プロセスに，非形式仕様，半形式仕様，および形式仕様アニメーションを統合して，漸進的な形式仕様記述手法を確立した．この手法でソフトウェアシステムの設計仕様を作成することは，図 1 に示したようなプロセスになっている．

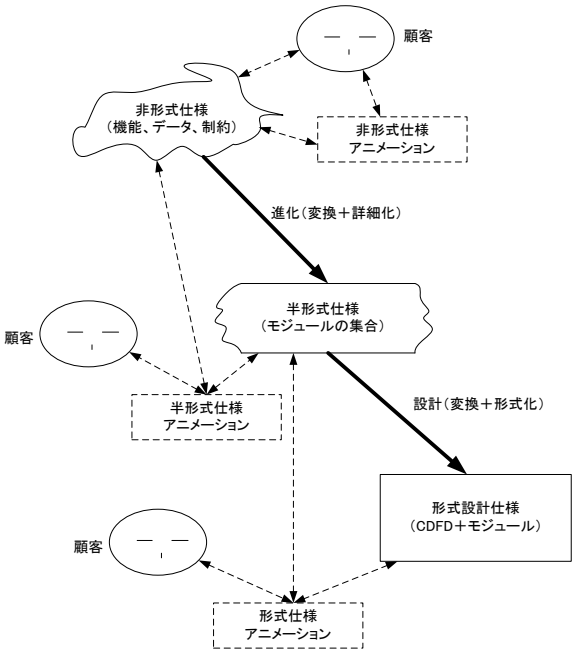


図1　SOFL三段階漸進的形式仕様記述プロセス

顧客とコミュニケーションを取りながら，応用領域の資料や現実世界のシステムに関する知識などを検討することから始め，開発するシステムに対する要求を徐々に理解する．その上で，システムの非形式仕様を作成する．この仕様の中に，要求されたシステム機能，その機能を実現するために必要なデータリソース，および機能またはデータリソースに対する制約が簡潔に記述される．機能を記述する際，「抽象と分解」の手法を適用し，大きい機能は小さい機能に分解して，階層的に機能を比較的詳細に記述する．同じように，データリソースおよび制約の部分も階層的に記述することもできる．

一般的には，非形式仕様の内容が，開発者（例えば，要求工学エンジニア）に作成される．その内容は，システムの要求を提出した顧客に確認してもらわなければ，方向性的な間違いが仕様に含まれる可能性が高い．このため，非形式仕様のアニメーションを実施することによって，その内容を確認することができる．アニメーションをしながら，開発者は顧客とのコミュニケーションを強化することができ，顧客のフィードバックももらえる．獲得した顧客のフィードバックによって，非形式仕様の完全性を改善させ，記述されたシステム機能，データリソース，および制約間の関係をより明らかにする．このような改善された非形式仕様は，次に半形式仕様を作成する基礎になる．

非形式仕様が自然言語（例えば，日本語）で記述されたため，意味は明確にしていない表現がたくさん含まれている可能性が高い．この問題と解決，非形式仕様の内容は，より明確に理解できるためには，半形式仕様を作成する．

半形式仕様は，基本的に非形式仕様を進化することによって得たものである．非形式仕様の進化は，二つの行動が含まれている．一つは，「変換」であり，もう一つは，「詳細化」である．「変換」というのは，非形式仕様の全体の構造および記述された機能，データリソース，および制約から，半形式仕様の全体の構造およびモジュールの内容へ変換することである．これを実現するために，次の三つのことをするのは有効である．

(a)非形式仕様に記述された関連しているシステム機能，データリソースおよび制約を，SOFL モジュールに結合させる．

(b)データリソースを適切なモジュールで，型と変数として形式に宣言する．この宣言によってそのデータリソースの意味を明確に定義できる．

(c)システム機能を適切なモジュールでプロセスとして定義する．

この定義において，入力，出力，および状態変数を含むプロセスのインタフェースを形式に定義し，プロセスの振る舞いを自然言語で表す事前条件と事後条件により定義する．更に，非形式仕様に記述された制約は，半形式仕様において，不変条件として定義或いは関連するプロセスの一部として定義する．

このように作成した半形式仕様の内容は，開発者と顧客の本音に合致しているかどうかを確認することが必要である．このため，半形式仕様アニメーションを行う．このアニメーションの主な目標は，半形式仕様で定義されたプロセス仕様のすべての面，定義されたデータ型，および関連している不変条件を確認することである．プロセス仕様にたいしては，そのインタフェースおよび事前条件と事後条件を確認する．インタフェースに関しては，プロセス名，プロセスの入力変数名と型，出力変数名と型，状態変数名と型を確認する．これによって関連しているデータ型の定義や状態変数の宣言なども検証する．プロセスの振る舞いを定義する事前条件と事後条件は，基本的に自然言語で記述されているため，プロセスの振る舞いのアニメーションは，テストケースと期待される結果を作成した上で適切に行う．ただし，事前条件と事後条件は形式に定義されていないため，このアニメーションを系統的に行うのは困難である．

このアニメーションに基づき行ったアニメーションによって，開発者と顧客からのフィードバックを獲得することができ，それによって半形式仕様を改善することが実現できる．このように作成したシステムに対する顧客の要求を比較的明確に定義した半形式仕様は，次の段階で行うシステム設計仕様作成の基礎になる．

システム設計の目的は，半形式仕様で定義したプロセスを適切に統合することを伴い，必要な新たなプロセスを追加することによって，システムのアーキテクチャーを確定し，システム全体の機能をアーキテクチャーからプロセスまですべて形式的に定義する．これを達成するために，二つのことを完成しなければならない．

(d)関連しているプロセスを適切に統合してシステムのアーキテクチャーを反映する CDFD を作成する．

(e)作成された CDFD に含まれているプロセス，データフローおよびデータストアというコンポーネントを，対応するモジュールに形式的に定義する．

具体的には，プロセスの振る舞いは，事前条件と事後条件を論理式で形式に定義する．データフローは，適切な型で宣言する．データストアも適切な型で宣言する．宣言された型やストア変数に関する制約は，不変条件で定義する．

このように作成された形式設計仕様は，顧客と開発者の要求を正しく反映しているかどうかを確認するために，その形式仕様のアニメーションを実施する．このアニメーションによって，システムレベルの視点から様々な具体的な機能を，分かりやすい形で動的に表現する．これを実現するために，次の二つの手順を取る．第一に，システムのアーキテクチャーを表すCDFDからすべての「システム機能シナリオ」を導出する．一つのシステム機能シナリオは，一種の入力データフローから出力データフローまでのデータフロー列であり，システムの一つの独立な部分機能を表す．第二に，システム機能シナリオを，一つずつアニメーションを実施する．選定された一つのシステム機能シナリオに対し，入力データフローの値とそれに対応する出力データフローの値を生成し，そのシナリオに含まれているプロセスの「実行」を，動的に表現する．この表現によって，開発者と顧客にそのシナリオの振る舞いを確認してもらう．

3.6 SOFL三段階漸進的な形式仕様記述手法の支援ツール

従来のSOFL三段階形式仕様記述技術に，前述した非形式仕様アニメーション，半形式仕様アニメーション，形式仕様アニメーション，および仕様パターンに基づく形式仕様作成アプローチを統合して得た「SOFL三段階漸進的形式仕様記述技術」（SOFL－GSTと呼ぶ）を，効果的に応用するためには，ソフトウェア支援ツールが不可欠である．本研究では，次の三つの支援ツールを含めているSOFL－GSTを支援する綜合ツールのプロトタイプを開発してきた．

(a)SOFL非形式仕様，半形式仕様，および形式仕様の作成を支援するGUIエディタ（SOFL-SpecTool）

(b)SOFL非形式仕様アニメーションと半形式仕様アニメーションを支援するツール（SOFL-AnimationTool）

(c)形式仕様パターンに基づくSOFLプロセスの形式仕様を作成するアプローチを支援するツール(SOFL-PatternFS)

SOFL-SpecToolは，よく設計したGUIを提供し，各レベル仕様の作成を効率的に支援する．非形式仕様を作成する際，仕様を表現するアリアを自動的に提供し，システム機能，データリソース，および制約という三部分のタイトルを自動的に準備する．各部分の内容を読みやすくするため，一つ項目の前に適切な番号を付ける．それらの番号は，階層的な形で使い，新たな機能を記述するときにその番号を自動的に配布してくれる．更に，データリソース節と制約節に定義された一つの項目（データ項目または制約項目）には，それ項目と関係がある機能項目を付けるサービスも提供している．

半形式仕様を作成する際,SOFLモジュールに含まれる全ての部分(例えば,定数の宣言，型の宣言，外部変数の宣言，不変条件の定義，プロセス仕様，関数の定義）を自動的に準備する．必要に応じて，各部分の内容は，ユーザが作成する．SOFL手法によると，半形式仕様に対しては，CDFDの作成は選択できる．すなわち，必要であれば，CDFDを作成し，必

要でなければ，CDFD を作成しない．どちらケースでも，支援ツールが対応できる．
形式仕様を作成する際，まず CDFD の作成から始まる．CDFD の描画を効率的に支援するためには，CDFD のコンポーネントの形（例えば，プロセス，データフロー，データストア，不確定構造，放送構造など）を表すアイコンを画面に適切な場所で表示する．一つのアイコンを選択すると,そのアイコンが表す CDFD のコンポーネントを簡単に描くことができる．描かれた CDFD のコンポーネントを修正したり，追加したり，保存したりする様々な機能も提供している．更に，CDFD の構文の正しさおよびある意味的な間違いを自動的に検出することができる．

描かれた CDFD に対し，それを対応する SOFL モジュールのアウトラインを自動的に生成する．つまり，このモジュールの各部分のタイトル（例えば，const，type，var，inv，behav，process など）を自動的に生成する．ユーザが各部分の内容を入力することが必要である．また，支援ツールは，CDFD とモジュールの整合性を自動的に維持することができる．例えば，CDFD に使っていないプロセスは，対応するモジュールに必ず定義しない．

SOFL-AnimationTool は，Microsoft の Add-in Express ソフトウェアを用いて設計された部品とテンプレートの使用を支援するツールである．プログラミング言語 C#を用いてこのアニメーション支援ツールのプロトタイプを開発した．この支援ツールによって，ユーザが SOFL 非形式仕様或いは半形式仕様に定義された機能または操作のアニメーションを容易に作成することができる．このため，支援ツールには，各種のデータ構造を表すデータアイコンおよび各種の操作の振る舞いを動的に表現する機能テンプレートが予め用意されている．非形式仕様と半形式仕様の意味は，必ず明確に定義してある訳ないため，このようなアニメーションシステムの構築は，自動的に行うことができない．必要に応じで，開発者が適切なアニメーションシステムを開発する．但し，その時，この支援ツールを使えば，効率がよくなるし，エディットしやすいし，間違いを避けられることも可能である．

SOFL-PatternFS は，ユーザと対話するために適切な GUI を提供している．その上で，コンピュータに保存されている形式仕様パターン知識の有限状態遷移図によって，ユーザに必要な入力をしてもらうために適切な指示を出すことができる．ユーザと自然言語（例えば，英語，日本語）で対話しながら，最終的にプロセスの形式仕様が自動的に作成される．

3.7 事例研究による提案した SOFL 技術の評価

本研究で提案した SOFL 三段階漸進的な形式仕様記述手法の支援ツールの品質を向上させ，その形式仕様記述手法の効果を評価するため，「JR 東日本の Suica カードシステム」，「旅行会社システム」および「スマート交通信号システム」という三つの事例研究を行った．

(1) JR 東日本の Suica カードシステム

事例研究は，JR 東日本の Suica カードシステムを参考にして，SOFL 技術によって非形式仕様と半形式仕様を作成した上で，形式設計仕様を作成した．また，仕様アニメーションを通じて，仕様の内容を検証した．本 Suica カードシステムは，次の三つの大きい機能が含まれている．(a)Suica カード，(b)改札機，(c)券売機．Suica カードの機能は大きく分

けて以下の三個である．初期化，カード情報の更新(チャージ，駅利用，一般利用，定期券登録等)，払い出し（ユーザからのカードの破棄の際の払い出し）である．改札機の機能は大きく分けて次の 2 つである．乗車処理と降車処理である．乗車処理ではユーザが乗車した際に定期券であるか，または最低限の乗車賃がチャージされているかを確認している．また降車処理では，残高と照会し料金が足りない場合はエラーを返す等の処理を行なっている．券売機としての機能は現金を受け付けて，スイカカードにチャージを行い，おつりを払い出すという機能である．

本事例研究で開発した非形式仕様は，約 99 行（1 行には 1 つの機能，データリソース，または制約が記述されている）の長さであり，半形式仕様は，約 148 行（1 行には SOFL の 1 つの形式的宣言または自然言語の機能表現が記述されている）の長さであり，形式仕様は，約 153 行(1 行には SOFL の 1 つの宣言または論理式が記述されている)の長さである．本事例研究は，1 人の協力研究者によって完成した．これによって，個人レベルで SOFL 技術は有効に適用することが確認した．

(2)旅行会社システム

旅行会社システムの事例研究に関しては，JTB オンライン旅行予約システムの機能を参考した上で，旅行会社システムの機能要求，データ要求，および制約などを決めていた．上質な形式設計仕様を最終的に構築するために,本研究で SOFL 三段階漸進的形式仕様記述技術を応用して，非形式仕様，半形式仕様，形式仕様を作成し，仕様アニメーションを行った．

本事例研究で開発した非形式仕様は，約 39 行の長さであり，半形式仕様は，約 218 行の長さであり，形式仕様は，約 391 行の長さである．本事例研究で確認したことは，本研究で開発した技術が何人を含めるチームにも有効に応用されられることである．

(3)スマート交通信号システム

スマートになる交通信号システムは，車，交通管理センター，および交通信号機から構成されている．すべての車には，センサが付けられ，ワイヤレスコミュニケーション能力がある．車は，交通信号の交差点エリアに入ると，その交通信号機の情報は交通管理センターに送られる．交通管理センターは，交差点エリアに入った車とその交差点の交通信号機の情報を収集し，それによって毎週その交通信号機の信号のディスプレイの時間を計算し,信号機に信号のディスプレイ時間を調整する命令を出すことを主なタスクとして働く．交通信号機は，基本的に信号をディスプレイすることと，交通管理センターからの命令を受け取って，赤と青信号のディスプレイの時間を調整する．

この交通信号システムの形式仕様を作成するために，非形式仕様の作成から始め，半形式仕様の作成と仕様アニメーションを通じて，システムに対する要求を十分に理解した上で，最終的に形式仕様を作成した．

本事例研究で開発した非形式仕様は，約 22 行の長さであり，半形式仕様は，約 188 行の長さであり，形式仕様は，約 367 行の長さである．本事例研究によって，本研究で開発した支援ツールに含まれた 10 個の重要なバグを発見した.支援ツールの実装者はそれらバグを検討した上で，適切に修正した．

１．研究の背景および目的

1.1 背景

近年，ソフトウェアシステムに潜む欠陥は，ATM の障害や自動車，携帯電話のリコールなど，システムの信頼性・安全性の問題を引き起し，延いてはシステムトラブル，ダウンによる莫大なエネルギー損失と，大きな環境問題，社会問題に発展しかねない．特にインターネット，スマートフォンなど急増する情報通信システムに使われる基盤ソフトウェア OS および様々な応用ソフトウェア（オンライン銀行システム，航空券予約購入システムなど）には高い信頼性・安全性の確保が必要であり，開発期間とコストを大幅に減らすことはソフトウェア産業にとって喫緊の重要課題であり，依然として解決していない．この状況になっている主な原因は，ソフトウェアの大規模化に伴い，従来の開発方法では品質を維持できないことが最大の原因といえる．ソフトウェア開発プロセスの上流アクティビティ（要件分析，仕様，設計など）に使われている自然言語，図や表などは，明確な文法または意味論を定義していないため，曖昧性や意味不明の表現などが，要求仕様，設計仕様など上流文書に含まれている．このため，上流文書の整合性・正当性の検証や仕様からプログラムへの変化や作成されたプログラムの検証などは，正しく効率的に行うことが困難である．2006 年 6 月 9 日の朝日新聞の「バグ頻発デジタル製品」と題した記事が日本のデジタル商品の開発の問題点をよく指摘していた．これら問題点を徹底的に解決するためには，厳密的かつ使いやすい産業的な開発手法が望まれている．

形式手法は，数学や論理学に基づく形式的体系を用意し，その体系に基づき，システムの記述や分析を行うものである．具体的に述べると，形式手法は，形式仕様記述技術，詳細化技術，および形式的な検証技術を含んでいる．

様々な形式仕様記述技術が存在しているが，産業で適用可能なものは，VDM-SL（Vienna Development Method - Specification Language）[1]，Z[2]，B-Method [3]，および Event-B[4]であろう．これら形式仕様記述技術の特徴は，操作（operation）の機能を事前条件（pre-condition）と事後条件（post-condition）で定義し，定義された操作は，システムのアーキテクチャーに統合してシステム全体の機能を定義する．

詳細化技術は，抽象仕様から具体的仕様（またはプログラム）へ正しく変換するための技術である[5,6]．様々な必要な変換ルールを系統的に提供し，それらルールを適切に応用することによって，抽象仕様から具体的仕様へ正しく変換することを保証する．この技術の特徴は，選定されたルールを正しく適応すれば，詳細化された具体的仕様は，必ず抽象仕様を満たす，すなわち，具体的仕様は，抽象仕様で定義された機能を正しく実現していることである．もし抽象仕様がユーザの要求を完全に正しく定義すれば，この詳細化技術によって作成された最終的なプログラムは，テストや検証などを行わなくても，必ず正確である．このような技術は，VDM や Z や Event-B 形式手法に導入されている．

形式検証技術は，述語論理と Hoare 論理に基づくプログラムの重要な性質と正確性を証明する技術である．プログラムの正確性の証明に関しては，述語論理に基づき提案されたプログラムの意味論を表す Hoare 論理を用いて，そのプログラムが予め書かれた事前条件

と事後条件を満たすかどうかを厳密に証明することが可能である.

形式手法は，適切に使えば，開発されたプログラムに対してテストの必要がないことが価値であると言われる．これはあくまで理論上の話で，実際のソフトウェア開発プロジェクトに適用する際，様々な挑戦的な課題が残っている.

1.2 研究課題

本節では，形式手法の適用への挑戦的な課題，それを対処するために提案された形式工学手法,および具体的な形式工学手法 SOFL(Structured Object-oriented Formal Language)の改善策について論述することにより，本研究の研究課題を明らかにする.

1.2.1 形式手法の適用への挑戦的な課題

既存の形式手法の有効性については，専門家のなかで意見が分かれている．形式手法がすでに企業で有効に応用されていると主張している研究者は,イギリスの York 大学の Jim Woodcock, Newcastle 大学の John Fitzgerald, デンマークの Peter Gorm Larsen[7], B-Method と Event-B を設計した Jreamon Abril 氏[3,4]，および Dines Bjorner[8]が代表として挙げられる．その一方，形式手法は，企業で有効に適用するにはまだ未成熟と主張した研究者もいる．その代表としては，Virginia 大学の John Knight[9], McMaster 大学の David Parnas[10]，ヨーク大学の John McDermid[11]である.

私たちの長年の研究と経験によると，既存の形式手法は，企業の環境で適用する際，次の主な挑戦的な課題が残っていることが明らかになっている.

(1) 要求分析とシステム設計に役に立つ形式仕様の記述手法がほとんどない.

高信頼性システム開発の企業でしばしば使ってきた VDM, Z,B-Method,Event-B という形式手法は，形式仕様記述言語およびその言語の構文と意味を提供しているが[7]，形式仕様の作成によってシステムの要求分析や設計を具体的に行う手法を提供していない．このため，一般の実務者は，これら形式仕様記述言語を勉強しても，実際のシステム開発において,形式仕様をどのように考えて作成すればシステムの要求分析と設計に役に立つのかは,余り分かっていない．この結果，形式手法の勉強会に出席しても，実務者が多いこともあるが，その後適用するケースが非常に少ない.

(2) 形式仕様の作成は，開発者の抽象化能力と数学の知識が求められるため，一般の実務者にとっては難しい.

形式仕様は，論理学，集合論，代数など数学に基づく設計された形式言語で記述されたものである．簡潔かつ明確な形式仕様を作成するためには，開発者の抽象化能力と関係する数学の知識を持つことが求められる．しかし，ソフトウェア開発現場で離散数学やプログラム意味論,論理学など数学の知識をしっかり勉強した実務者が相対的に少ない.また,それらの数学の知識をよく知っても,要求されたシステムの機能を抽象的に表現するには,困難に直面していることもある[12].

(3) 形式仕様に基づく開発者は顧客とのコミュニケーションが難しい.

形式仕様は数学に基づく設計された言語で記述されるため，一般の顧客にとって理解することは容易ではない．特に，システムの規模が大きくなると，形式仕様の複雑性が高くなり，この問題はより深刻になってしまう．システムの要求分析や設計に形式仕様記述技術を有効に使えるようになるには，この課題を解決しなければならない.

(4) 形式仕様の進化と維持には，時間，コストおよび忍耐力が求められる.

形式仕様記述技術を，要求仕様の作成に適用することは大勢の研究者が主張している[8]．しかしながら，私たちの研究と経験によると，このアプローチには大きい問題点がある．要求分析というシステム開発の初期段階において，形式仕様を作成するためには，時間と努力が必要である．もしここで書いた形式仕様は，そのまま最終的に作成されるプログラムのアーキテクチャーに使われるなら，問題がないと思うが，実際には，これは不可能に近い．理由は次の通りである.

要求仕様を作成した後，システムの設計（抽象設計または詳細設計）が行われるのは普通である．この間，要求された機能をコピュータ上で実現するためには，適切なアーキテクチャー，データ構造，および効率的なアルゴリズムが求められる．このため，元々書かれた形式要求仕様書は，特に，操作のインタフェース，機能など，変更あるいは修正しなければならない．これでは時間もかかるし，コストも生じるし，開発者の忍耐力も求められる．大規模のシステム開発には，コスト，時間などの制約が厳しいため，このような状況に陥るのは望ましくない.

1.2.2 形式工学手法の提案

前述した形式手法の課題を解決するためには，1989 年から 1997 年までの形式手法を伝統的なソフトウェア工学技術に統合する研究活動により，正式に「形式工学手法」という新たなアプローチが提案された[13,14]．形式手法と違い，形式工学手法が目指しているのは，形式仕様記述技術の利用によって，企業ですでに定着している要求分析と設計の図言語，開発手法，テスト，インスペクション（inspection）など技術の系統性，厳密性，有効性，および自動化程度を向上させることである.

形式工学手法の代表的な技術としては，劉研究グループで開発された SOFL（Structured Object-oriented Formal Language）形式工学手法である[12,14]．SOFL 手法は，SOFL 形式仕様記述技術，形式仕様に基づくインスペクション，および形式仕様に基づくプログラムテスト技術を含めている.

SOFL 形式仕様記述技術には，「SOFL 形式仕様記述言語」および「SOFL 三段階形式仕様記述アプローチ」が含まれている．SOFL 形式仕様記述言語は，既存の構造化分析・設計技術に使用されるデータフロー図，VDM-SL 形式仕様記述言語，及びオブジェクト指向プログラミング言語の仕組みを統合することにより開発された言語である．特徴としては，「テキストの数学表現だけではなく，明確に定義された構文と意味論を持つ分かり易いデータフロー図も使う．更に，その図でシステムのアーキテクチャーを定義した上で，システムの操作やデータ項目などの部品をテキストの形式言語で定義する方式である」ことを強調

する．SOFL 仕様言語で書かれた形式仕様の一般構造を図 1-1 で説明する．

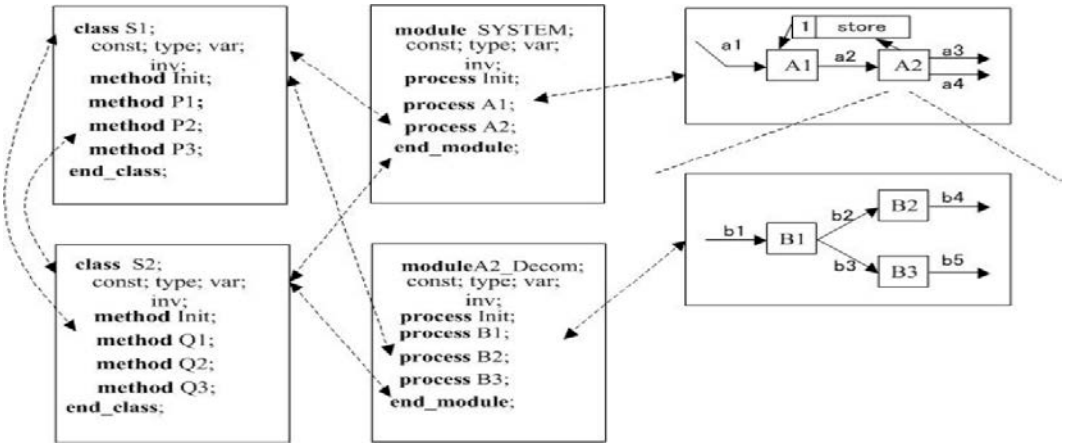


図1-1　SOFL形式仕様の一般構造

CDFD(Condition Data Flow Diagram)は，データフローとペトリネットを合わせたモデルでプロセス(操作)とデータの流れを表すことによって，システムのアーキテクチャーを表現する．モジュール(module)は，CDFD に使用されているソフトウェアコンポーネント(例えば，プロセス，データフロー，データストア)を形式的に定義する．クラス(class)は，CDFD に使用されているデータ項目(例えば，データフロー，データストア)がオブジェクトとして使われる場合，それらのオブジェクトを表す変数を定義するために必要な型として使われる．開発現場でこのような形式仕様を正しくかつ完全に作成するために，ユーザとのコミュニケーションをしながら機能要求とシステム設計のアイディアを明らかにしなければならない．要求分析と設計を容易かつ厳密に行うために，「非形式仕様記述」，「半形式仕様記述」，および「形式仕様記述」という「SOFL 三段階形式仕様記述アプローチ」は，SOFL 形式仕様記述技術のもう一つの重要な技術として確立された．

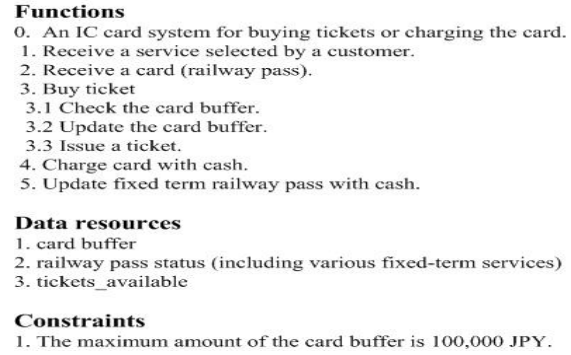
実際に開発した IC カードを用いた切符購入システムの記述例のスナップショットを参照しながら，SOFL 三段階仕様記述の例を紹介する．

(1) 非形式仕様記述(図 1-2 参照)

システム機能(functions)，データリソース(Data resources)，データと機能の制約条件(Constraints)を定義し，構造化された自然言語で記述する．図 1-2 は英語で書かれた IC カードを用いた切符購入システムの記述例である．システムの機能手順，データリソースと制約条件を簡潔に表現している．

複雑なシステムの場合は，定義された高いレベルの機能は低いレベル機能に分解することができる．このような分解は，データリソースや制約条件にも適用する．この結果，非形式仕様の一般的な構造は，階層的な機能記述，階層的なデータリソースの記述，および

階層的な制約条件の記述という三部分となる．



Functions
0. An IC card system for buying tickets or charging the card.
1. Receive a service selected by a customer.
2. Receive a card (railway pass).
3. Buy ticket
3.1 Check the card buffer.
3.2 Update the card buffer.
3.3 Issue a ticket.
4. Charge card with cash.
5. Update fixed term railway pass with cash.

Data resources
1. card buffer
2. railway pass status (including various fixed-term services)
3. tickets_available

Constraints
1. The maximum amount of the card buffer is 100,000 JPY.

図1-2　非形式仕様の事例

(2) 半形式仕様記述(図 1-3 参照)

半形式仕様記述の主な目的は二つである．一つは，非形式仕様の構造と意味をより明確に定義する．このため，SOFL 言語のモジュール記述の仕組みを利用する．もう一つの目的は，明確に表現したシステム要求を，後で行う抽象設計のために，より変換しやすい形でアレンジ（arrange）する．

これら目標を達成するために，非形式仕様に基づき，次の三つの行動を取る．第一に，非形式仕様に定義された関連しているシステム機能，データリソース，制約条件を SOFL モジュールにまとめる．第二に，必要なデータ型を定義し，データリソースを変数として宣言する．第三に，制約条件は，不変条件(invariant)として定義またはあるプロセス仕様に定義し，すべてのシステム機能を SOFL プロセスで事前条件と事後条件により定義する．但し，論理式で表す不変条件やプロセスの事前条件と事後条件を，理解し易くする事を保証するため，すべて自然言語で記述する．

```
module Purchase_Ticket_and_Charge_Card;
type
 Card = composed of
          buffer: nat0
          railpass_status: RailPassStatus
         end;
RailPassStatus = composed of
                   status: [<Ordinary>, <Student>, <Senior>]
                   period: [<3 Months>, <6 Months>, <12 Months>]
                  end;
ServiceRequest = composed of
                   Servive_choice: [<Buy Ticket>, <Charge Card>,
                                    <Update Railpass>]
                   money_amount: nat0
                  End;
Ticket = composed of
           number: nat0   /*unique number for each ticket*/
           value: nat
          end;
var
 card: Card;
 ticketPool: set of Ticket;
 inv
  The buffer of card should be less than or equal to 100,000 JPY.

process Receive_Card(Inserted_card: Card )
 ext wr card: Card
 pre true
 post assing the information of the inserted card to the state variable
      card for the other processes in this module to use.
end_process;
process Buy_Ticket(ticket_price: nat)
 Ext wr card: Card
 pre true
 post (1) check whether the ticket_price is less than the card buffer.
      (2) if (1) is true, a ticket can be bought, and the card buffer is
          updated.
      (3) if (1) is false, then an error message is issued.
end_process;
process Charge_Card(amount: nat )
 ext wr card: Card
 pre true
 post If the addition of the existing amount of the card buffer and the
      input amount for charging the card is greater than 100,000 JPY,
      then an error message of exceeding the limit is issued; otherwise,
      add the input amount to the card buffer.
end_process;
process Update_RailPass(term_status: RailPassStatus, cash: nat)
 ext wr card: Card
 ...
end_process;
process Service_Selection(service_req: ServiceRequest)
 ...
end_process;
end_module;
```

図1-3　半形式仕様の事例

(3) 形式仕様記述(図 1-4, 1-5 参照)

最後に形式仕様に，システムアーキテクチャを CDFD で定義し，モジュールを VDM-SL に似ている形式仕様記述言語で記述する．その中で，必要なデータ型，変数を形式的に宣言し，プロセスの機能を事前条件と事後条件で定義する．このように可視化 CDFD と厳密な事前条件と事後条件で仕様を作成することは，仕様記述過程が簡単になるだけではなく，作成された仕様が検証可能な状態になる．

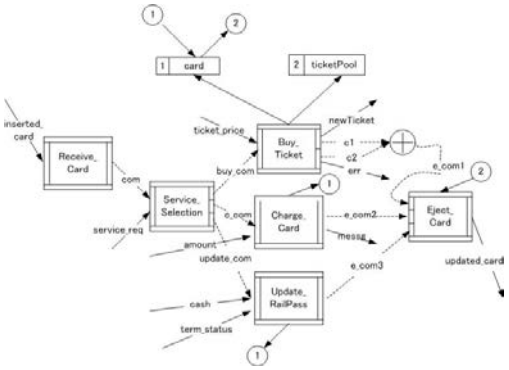


図1-4　CDFDの事例

```
module Purchase_Ticket_and_Charge_Card;
type
 …    /*The same as that of the semi-formal specification */
var
 card: Card;
 ticketPool: set of Ticket;
 inv
  card.buffer <=100000;   /* Formalization of the informal invariant*/

process Receive_Card(inserted_card: Card ) com: sign
 ext wr card: Card
 pre true
 post card = inserted_card
end_process;
Process Buy_Ticket(ticket_price: nat, buy_com: sign)
                    newTicket: Ticket, c1: sign | c2: sign, err: string
 ext wr card: Card
      wr ticketPool: set of Ticket
 pre true
 post if ticket_price <= ~card.buffer
       then newTicket = get(ticketPool)  and
            newTicket.price = ticket_price and
            card = modify(~card, buffer -> ~card.buffer - price) and
            ticketPool = diff(~ticketPool, [newTicket])
        else err = "Your card has no sufficient money for the ticket."
end_process;
process Charge_Card(c_com: sign, amount: nat )
                        e_com2: sign, messg: string
 ext wr card: Card
 pre true
 post  if  ~card.buffer + amount <= 100000
       then card = modify(~card, buffer -> ~card.buffer + amount) and
            messg = "the charge is successful."
       else messg = "your amount is over the limit."
end_process;
process Update_RailPass(update_com: sign, cash: nat, term_status: RailPassStatus)
                          e_com3: sign
 ext wr railpass_status : RailPassStatus
 …
end_process;
process Service_Selection(com: sign, service_req: ServiceRequest)
 …
end_process;
end_module;
```

図1-5　形式モジュール仕様の事例

既存の形式手法(例えば，VDM,Z，B-Method，Event-B など)と比べて，SOFL 手法の特長は，簡単さ(Simplicity)，可視性(Visualization)，および明確さ(Preciseness)をバランス良く持つことができ，要求分析に非形式仕様と半形式仕様の記述，設計に形式仕様の記述というように，形式手法とソフトウェア工学手法を融合していることである．SOFL 手法の実用性がより高いということを，企業からの受託研究プロジェクトで確認したが，前述した形式手法に直面している挑戦的な課題はすべて解決した訳ではない．特に，三段階作成された要求仕様と設計仕様による顧客とのコンミュニケーションの問題，プロセスの形式仕様作成の困難さ，および支援ツールの不足問題がまだ解決していない．

1.2.3 SOFL 形式工学手法の改善策

SOFL 形式工学手法の前述した問題点を解決し，企業へ技術移転を成功させるために，本研究では，次の研究課題を解決することを目指してきた．

(a)SOFL 非形式仕様の記述，解釈，および進化をどのようにすれば，完全かつ抽象レベルで正しいユーザ要求を獲得できるか．(b)SOFL 半形式仕様の作成，解釈，および進化をどのようにすれば，ユーザと開発者が合意した個々の機能要求を獲得できるか．(c)SOFL 形式仕様の作成，解釈，および進化をどのようにすれば，ユーザと開発者が合意したシステム機能要求を獲得できるか．

(d)どのようにすれば，SOFL 形式仕様を一般の実務者でも簡単に記述することができるか．

(e)SOFL 三段階形式記述技術をどのように系統的に支援すれば，完全に，正しく，かつ効率的に形式仕様を作成することができるか.

SOFL 非形式仕様記述が目指すことは，ユーザ要求を完全に獲得することである．そのためには，作成された非形式仕様の内容をユーザに適切に解釈され，説明できることが必要である．伝統的なやり方は，開発者が非形式仕様に記述された機能，データリソース，および制約を別々に口頭またはペンと紙で説明する方法である．しがしながら，この方法では，非形式仕様の内容を現実世界の環境で分かりやすく説明することができないため，効果は限られている.

SOFL 半形式記述作成の目的は，非形式仕様の曖昧性を取り除き，要求されたシステムに使われているデータリソース，機能を定義する操作仕様，および制約を定義する不変条件などの表現をより明確にし，関連する機能，データリソース，および制約を，SOFL モジュールにまとめて定義することである.これは,システムの形式設計仕様作成の基礎になる．一般的には，非形式仕様を半形式仕様記述においてより明確に表現するために，機能，データリソースや制約などを詳細化することが避けられない．この時点で，新たな決定が必要である．このような決定は，開発者がユーザと深く相談せず自分が決めてしまうケースが普通である．このため，半形式仕様の操作のインタフェースやデータ型や機能などをユーザと開発者に確認しなければならない．ただし，今まではこのような確認に有効な技術が欠如している.

SOFL 形式仕様作成は，システムの抽象設計を行い，システムのアーキテクチャーを設計した上で，各部品（例えば，プロセス，データフロー，データストア）を明確に定義する．このように定義した部品がシステムのアーキテクチャーのコンテクストで正しく取り込まれるかどうかは，検証することが必要である．このような検証は，形式仕様自体の整合性を保証するだけではなく，開発者とユーザが納得するかどうかという正当性も保証しなければならない.この検証の結果によって,形式仕様を適切に進化させることが必要である.

形式仕様の作成は，開発者がユーザのシステムに対する要求を深く理解する重要な手段である．しかしながら，形式仕様の作成に開発者が一定の抽象化能力と数学の応用能力を持つことが求められるので，企業での一般の実務者には難しいと感じるのが現状である．どのようにすれば，一般の実務者としても，システムの形式仕様を作成することができるのかは，一つの挑戦的な課題である.

SOFL 三段階形式仕様記述技術は，システムの形式仕様作成の唯一の具体的なアプローチを示す手法である．この技術を企業へ導入するには，ソフトウェア支援ツールがないと，なかなか難しいであろう．この問題を解決するために，SOFL 形式仕様作成支援ツールを劉研究グループで開発した[15]．しかし，それらのツールは，プロトタイプであるため，エラーがよく発生し，SOFL の三段階形式仕様記述技術を系統的に支援することができなかった．相対的に安定して成熟した支援ツールをどのように開発するかが重要な課題である．更に，支援ツールを用いて，SOFL 三段階形式仕様記述技術および仕様アニメーション技術が実際のソフトウェア開発において有効がどうかということも解明しなければならない課題である．本研究成果報告書において，仕様アニメーションとは仕様に記述された機能，データ項目，および制約を動的に表現することである.具体的には実施方法により，非形式仕様アニメーション，半形式仕様アニメーションおよび形式仕様アニメーションに分けて

いるが，共通点は仕様の内容を動的に表現することである．

1.3 研究の意義

前述した課題を解決することを目指す本研究の意義は，次の三つのポイントにある．

(a)従来の形式手法を企業へ技術移転する時に遭遇した困難を克服するために，「形式工学手法」という新たな道筋を示し，形式手法とソフトウェア工学という二つの分野で，学術および実践的な貢献ができ，日本のこの分野での研究を世界に発信することができる．

(b)さらに，本研究で確立した SOFL 三段階の漸進的な形式仕様記述技術を更に加えることで，実用性が高い SOFL 形式工学手法を，ソフトウェア産業へ導入でき，社会への還元が実現可能である．安全性が重要であるシステムの開発に適用するだけではなく，一般のエンタープライズソフトウェア，および組み込みシステムの開発にも有効に適用できる．また，この技術を適切なビジネス管理に取り込むことによって，より有効なグローバルビジネスモデルを創出することが可能である．

(c)従来の形式手法とソフトウェア工学の間のかけ橋の役割を果たす「形式工学手法」という研究分野の発展を促進することができる．

２．実施内容

2.1 研究アプローチ

2.1.1 研究の全体像

SOFL 形式工学手法で開発したシステム仕様に関しては，三つの重要な課題がある．第一に，作成された非形式仕様，半形式仕様，および形式仕様の内容をどのように顧客に容易に確認してもらい，必要なフィードバックを獲得できるか．第二に，形式手法を上手に使えない開発者に対して，どのように補助すればシステム要求と設計のアイディアを明確に理解しながら，それを反映する形式仕様を容易に作成できるか．第三に，SOFL 形式工学手法をどのように効率的かつ容易に応用できるか．

これら問題点を解決するために，本研究では，SOFL 三段階形式仕様記述技術を始めとするこれまでの研究実績をさらに発展させ，非形式仕様，半形式仕様，形式仕様の各アニメーション手法，および形式仕様パターンに基づく操作の事前条件と事後条件を作成するアプローチを提案,これら手法とアプローチを SOFL 三段階形式仕様記述プロセスに統合することによって，漸進的な形式仕様作成手法を確立，更にこの手法を効率的に支援するソフトウェアツールのプロトタイプを開発してきた．

(1) 非形式仕様アニメーション

非形式仕様アニメーションとは，SOFL 半形式仕様に記述されているシステムの機能（function)，データリソース，および制約（constraints）を，現実世界の仮想的な環境で動的に表現することである．目的としては，作成された非形式仕様の内容を，顧客に確認してもらい，顧客から必要なフィードバックを獲得し，非形式仕様を徐々に完成させることである．アニメーションが顧客の完全な要求を発見することに役に立つ技術になるためには，非形式仕様を作成するに伴いアニメーションを徐々に行う漸進的なアプローチを提案した．この中で，研究の焦点は，どのように非形式仕様によってアニメーションシステムを作成できるかという課題である．

これに対しては，次のステップを提案した．

(a)非形式仕様を作成する．

(b)非形式仕様に基づき，非形式仕様で記述されたシステムの「機能構造図」を構築する．この構造図は，顧客が一番関心を持っている機能等を中心に選出し，それをアニメーション化するために必要な GUI 部品と，その機能を実現するために必要な内部機能を適切にどう利用するかの関係を示す．また，機能の実現に必要なデータリソースと制約も適切に示す．

(c)機能構造図によって非形式仕様のアニメーションシステムを作成する.顧客が一番関心を持っている機能に対して，適切な GUI と，必要なインタフェースも作成，開発者の理解を得て，その機能の意味を反映するアニメーションを作成する．

(d)非形式仕様のアニメーションを行う．

(e)顧客からのフィードバックをもらい，非形式仕様を修正する

(2)半形式仕様アニメーション

SOFL 半形式仕様は，非形式仕様を詳細化して得たユーザの要求をより明確に定義している仕様であり，関連している機能，データリソース，および制約を SOFL モジュールにまとめている．構造としては，SOFL 半形式仕様は SOFL モジュールの集合である．

半形式仕様アニメーションとは，SOFL 半形式仕様に含まれているモジュールにおいて定義されているプロセス（操作）の機能を，一つずつ動的に表現する．目的としては，半形式仕様で定義されたプロセスのインタフェース，入力変数，出力変数，これら型，外部変数と型，および期待されている振る舞いを，開発者（つまり，半形式仕様の作成者）と顧客に確認してもらい，必要なフィードバックを獲得し，半形式仕様を，期待される形に改善する．

アニメーションが開発者と顧客の要求した通りのプロセス機能を正しく定義することに役に立つ技術になるためには，半形式仕様を作成するに伴いアニメーションを徐々に行う漸進的なアプローチを提案した．この中で，研究の焦点は，プロセスの入出力の値，外部変数の値，および振る舞いをどのように動的に表現すると，開発者と顧客が分かりやすくなるかが研究課題である．

これに対しては，次の手順を提案した．

(a)非形式仕様に基づき，半形式仕様を作成する．

(b)半形式仕様に含まれるモジュールを一つずつ選出する．一つのモジュールに対しては，その中に含まれているプロセス仕様を一つずつ選出する．

(c)一つのプロセス仕様に基づき，そのプロセスのインタフェース，入出力の値，外部変数の値，および振る舞いを動的に表現するアニメーションシステムを作成する．

(d)プロセス仕様のアニメーションを行う．

(e)開発者と顧客からのフィードバックをもらい，半形式仕様を修正する．

(3) 形式仕様アニメーション

SOFL 形式仕様は，基本的には階層的なモジュールと階層的な CDFD（Condition Data Flow Diagram）から構成されたものである．CDFD はイベント駆動の操作意味論を持つ形式的データフロー図であり，システムのアーキテクチャーをデータの流れの観点から描画する．CDFD に対応するモジュールにはその CDFD の各部品（例えば，プロセス，データフロー，データストア）を SOFL 形式仕様記述言語で明確に定義される．

形式仕様アニメーションとは，CDFD から「システム機能シナリオ」を導出し，それの振る舞いを入力によって出力するプロセスを動的に表現することである．目的としては，開発者と顧客に形式仕様で設計したシステムの機能シナリオを確認してもらい，システム機能シナリオの整合性も確保することである．この中で，主な研究課題としては，CDFD からどのようにシステム機能シナリオを自動的に導出するか，導出されたシステム機能シナリオの振る舞いをどのように動的に表現するか，入力と出力変数の値はどのように生成できるかという問題点が含まれている．このような課題を全て解決するためには，長い期間の

研究が必要となる．本研究では，CDFD からシステム機能シナリオの自動導出技術だけの研究を主に取り込んできた．

この点に関しては，CDFD をグラフとして扱い，CDFD の「開始プロセス」から入力データフローを確定，CDFD の「最終プロセス」から出力データフローを確定，その入力データフローから出力データフローへ到達するデータフローパスは，システムの機能シナリオとして形成する．具体的な生成手法とアルゴリズムは，研究成果の節で詳しく紹介する．

(4) 形式仕様パターンに基づくプロセスの形式仕様作成アプローチ

形式仕様パターンとは，形式仕様記述言語の知識とその知識を適用した経験を反映する一定の種類の論理式を作成するための仕様テンプレートである．一般的には，一つの仕様パターンに，次の四つの部分が含まれている．

(a)パターン名
(b)パターンの説明
(c)パターンの構成要素
(d)パターン解答（solution）

パターン名は該当するパターンが表す論理式の性質を反映する名前である．その名前を見ると，このパターンの適用によってどんな論理式を生成するかが分かる．パターンの説明は該当パターンの目的を自然言語で説明する．パターンの構成要素は期待される論理式を作成するために,必要なコンポーネントを説明する．最後にパターン解答は，該当パターンの構成要素の型によってよく使われている形式的論理式を示す．これら論理式の中で一番適切なものを選択する．

コンピュータプログラムの機能を表現するには，形式仕様パターンは「関係」,「情報検索」，および「情報更新」という三種類に分けることができる．「関係」というパターン種類には，二つの数式間のあらゆる関係（例えば，＞，＜，＝）を表現する論理式を形成するパターンを含める．「情報検索」というパターン種類には，必要な情報を検索する意味を表す論理式を形成するパターンを含める．「情報更新」というパターン種類には，あるデータ構造に保存されている情報を一定の方法で更新する意味を表現する論理式を作成するパターンを含める．

本研究で提案した形式仕様パターンに基づく形式仕様を作成するアプローチの主な目的は，このアプローチを支援するツールを通じて，ユーザ（開発者）と自然言語で対話することによってユーザの明確な機能要求を確認した上で，その機能を表現する形式的論理式を自動的に作成することである．これを実現するためには，仕様パターンを知識としてコンピュータに保存しておき，支援ツールの GUI を通じてユーザと自然言語で対話しながら，コンピュータアルゴリズムがそのパターンを適用し，ユーザに適切な指示を出し，必要な入力を求め，このようにユーザとコンピュータが対話しながら，最終的に形式的論理式を作成する．この間，ユーザは SOFL のような形式仕様記述言語を直接に使う必要がないので，一般の実務者にとっては，より簡単に形式仕様を作成することができる．

この技術の中で，コンピュータからユーザに適切な指示を出せるためには，仕様パターン知識を適切な形式でコンピュータに保存することが大事である．ユーザ等の対話によって形式仕様を漸進的に形成するプロセスは，イベント駆動型であるため，有限状態遷移図

（FSM）を用いて，全ての仕様パターン知識を保存する形式を採用している．ユーザの入力はイベントとして扱い，それによって適切な状態遷移を行う．一つの状態に到達すると，その状態から次の状態を選択するために必要な入力を求める指示をユーザへ出す．このように対話しながら，最終的に形式仕様を作成する．

(5) SOFL三段階漸進的形式仕様記述技術の支援ツール

従来の SOFL 三段階形式仕様記述技術に，前述した非形式仕様アニメーション，半形式仕様アニメーション，形式仕様アニメーション，および仕様パターンに基づく形式仕様作成アプローチを統合して得た「SOFL 三段階漸進的形式仕様記述技術」（SOFL-GST と呼ぶ）を，効果的に応用するためには，ソフトウェア支援ツールが不可欠である．本研究では，次の三つの支援ツールを含んだ SOFL-GST を支援する綜合ツールのプロトタイプを開発してきた．

(a)SOFL の非形式仕様，半形式仕様，および形式仕様の作成を支援する GUI エディタ（SOFL-SpecTool）

(b)SOFL の非形式仕様アニメーションと半形式仕様アニメーションを支援するツール（SOFL-AnimationTool）

(c)形式仕様パターンに基づく SOFL プロセスの形式仕様を作成するアプローチを支援するツール(SOFL-PatternFS)

SOFL-SpecTool は，よく設計された GUI を提供し，各レベル仕様の作成を効率的に支援する．非形式仕様を作成する際，仕様を表現するエリアを自動的に提供し，システム機能，データリソース，および制約という三部分のタイトルを自動的に準備する．各部分の内容を読み易くするため，一つ項目の前に適切な番号を付ける．それらの番号は，階層的な形で使い，新たな機能を記述するときにその番号を自動的に配布してくれる．更に，データリソース節と制約節に定義された一つの項目（データ項目または制約項目）には，その項目と関係がある機能項目を付けるサービスも提供している．

半形式仕様を作成する際，SOFL モジュールに含まれる全ての部分（例えば，定数の宣言，型の宣言，外部変数の宣言，不変条件の定義，プロセス仕様，関数の定義）を自動的に準備する．必要に応じて，各部分の内容はユーザが作成する．SOFL 手法によると，半形式仕様に対しては，CDFD の作成を選択できる．すなわち，必要であれば CDFD を作成し，必要でなければ CDFD を作成しない．どちらケースでも支援ツールは対応できる．

形式仕様を作成する際，まず CDFD の作成から始まる．CDFD の描画を効率的に支援するためには，CDFD のコンポーネントの形（例えば，プロセス，データフロー，データストア，不確定構造など）を表すアイコンを画面に適切な場所で表示する．一つのアイコンを選択すると，そのアイコンが表す CDFD のコンポーネントを簡単に描くことができる．描かれた CDFD のコンポーネントを修正，追加，保存する様々な機能も提供している．更に，CDFD の構文の正しさ，および意味的な間違いを自動的に検出することができる．

描かれた CDFD に対し，それを対応する SOFL モジュールのアウトラインを自動的に生成する．つまり，このモジュールの各部分のタイトル（例えば，const，type，var，inv，behav，process など）を自動的に生成する．ユーザが各部分の内容を入力することが必要である．

また,支援ツールは CDFD とモジュールの整合性を自動的に維持することができる.例えば,CDFD に使っていないプロセスは，対応するモジュールに定義しない.

SOFL-AnimationTool は Microsoft の Add-in Express ソフトウェアを用いて設計された部品とテンプレートの使用を支援するツールである.プログラミング言語 C#を用いてこのアニメーション支援ツールのプロトタイプを開発した．この支援ツールによって，ユーザが SOFL 非形式仕様或いは半形式仕様に定義された機能または操作のアニメーションを容易に作成することができる．このため，支援ツールには各種のデータ構造を表すデータアイコンおよび各種の操作の振る舞いを動的に表現する機能テンプレートが予め用意されている．非形式仕様と半形式仕様の意味は，必ず明確に定義してある訳ではないため，このようなアニメーションシステムの構築は自動的に行うことができない．必要に応じで，開発者が適切なアニメーションシステムを開発する.但し,その時この支援ツールを使えば,効率がよくなり，エディットしやすく，間違いを避けることも可能である.

SOFL-PatternFS は，ユーザと対話するために適切な GUI を提供している．その上で，コンピュータに保存されている形式仕様パターン知識の有限状態遷移図によって，ユーザに必要な入力をしてもらうために適切な指示を出すことができる．ユーザと自然言語（例えば，英語，日本語）で対話しながら，最終的にプロセスの形式仕様が自動的に作成される.

具体的な技術の説明は，研究成果の内容の章で詳しく説明する.

2.1.2 関連するこれまでの研究について

本節では，本委託研究以前に実施していた関連研究についてまず説明し，その上で本委託研究との関係を簡潔に説明する.

(1) 本委託研究以前に実施されていた委託研究に関連する研究について

本委託研究の基礎技術とする SOFL 形式仕様記述言語が示したような形式手法をソフトウェア開発技術に統合する研究は，90 年代から活発になってきた．例えば，Bryant 研究者は，Oxford 大学で開発された形式仕様記述言語 Z と Yourdon データフロー図を統合して構造化形式仕様作成方法を提案し[28],Fraser 研究グループはデータフロー図と VDM(Vienna Develoment Method）を統合して VDM より分かりやすい形式仕様作成方法を提案した[29]。また,オストラリアの Queensland 大学で形式仕様記述言語 Z にオブジェクト指向設計の仕組みを導入することによって開発されたオブジェクト Z および VDM にオブジェクト指向プログラミング仕組みを導入して提案された VDM++は，オブジェクト指向設計の形式仕様を作成することができる[24,30]．提案された OCL（Object Constraint Language）形式仕様記述言語は，UML（Unified Modeling Language）図で表す意味を明確に定義していない設計を明確に定義するために使われる[25].OCL を支援するツールも開発された[26].また，Butler 研究グループは，UML と B-Method を統合するアプローチを提案した[27].この提案の中で，UML を用いてシステムの構造を表現し，その中に含まれているコンポーネント（例えば，データ項目，操作など）を B-Method で明確に定義する．国内では，北陸先端技術大学院大学（JAIST）の二木厚吉研究グループは，90 年代から代数型の形式手法 OBJ とオブジェクト指向設計を統合して CafeOBJ を提案し[31]，それに関連する証明技術を開発して

きた[32]．九州大学の荒木啓二郎研究グループは，VDM を企業へ導入して様々な適用方法を検討し，形式仕様記述技術の利点を明らかにした[33]．

作成された形式仕様は，顧客の要求を正しく反映しているかどうかを確認するために仕様アニメーションやプロトタイピングなどアプローチが提案された．Bumbulis 研究グループは，ソフトウェアコンポーネントに基づくユーザインタフェースの開発における形式手法とプロトタイピングを統合した手法を提案した[16]．基本的なアイディアは，形式仕様を作成し，それをアニメーションすることによって，システムの振る舞いを観察することができるし，それの重要な性質を形式的に検証することも可能である．但し，形式仕様をアニメーション化するために，仕様から実行できるコードへ自動的に変換することが必要である．この条件を満たすためには，形式仕様の表現方式に対して制約しなければならない．Morrey 研究グループと Kazmierczak 研究グループも，同じような特徴を持つ Z 形式仕様記述言語で作成された形式仕様のアニメーションアプローチを提案した[17, 18]．また，Virzi 研究グループと Sefelin 研究グループは，low-fidelity プロトタイピングが要求の発見とシステムの設計のアイディアを示すために使える補助技術として提案された[19, 20]．Low-fidelity プロトタイピングは，紙ベース或いはコンピュータベースという二種類がある.共通の特徴は,プログラミング言語で実行できるコードを書く必要がなく，既存のツール（紙，ペン，ソフトウェアなど）を用いて素早く簡単に顧客の要求やシステムの設計の基本的なアイディアを適切に表現することができる．

実務者に効率的に Z 形式仕様を作成してもらうために，Stepney 研究グループは形式仕様パターンという概念を提案した[21]．さらに，Z 仕様を作成する際対処しなければならない問題を解決するために，六種類の仕様パターンを定義した．それは，表現パターン（presentation pattern），応用領域パターン（domain pattern），慣用パターン（idiom pattern），アーキテクチャーパターン（architecture pattern），構造パターン（structure pattern），および発展パターン（development pattern）である．Ding 研究グループは，形式仕様を効率的に構築するために，性質保証詳細化パターン（property-preserving refinement patterns）アプローチを提案した[22]．Konrad 研究グループは，いくつかの産業界の組み込みシステムを分析した上で，リアルタイム仕様パターンアプローチを提案した[23]．これらパターンを適用する際，仕様を作成する開発者は，仕様パターンをよく理解した上で，適切なパターンを選択して応用する．

(2) 研究責任者が本委託研究以前に実施していた委託研究に関連する研究について

本委託研究以前に，1996 年から 2002 年までに渡って既存の形式手法 VDM，データフロー図,およびオブジェクト指向手法を統合して研究開発した SOFL 三段階形式仕様記述技術を踏まえ，本委託研究に関連する次の三つの研究を実施している．

(a)SOFL 形式仕様作成の支援ツール．

(b)形式仕様パターンの定義と分類に関する研究調査．

(c)SOFL 三段階形式仕様記述プロセスにプロトタイピング技術の統合．

SOFL 形式仕様作成の支援ツールに関しては，SOFL 形式仕様に使われる CDFD の描画と対応モジュールの作成というアクティビティを支援することに集中して研究していた．その結果，CDFD を作成する簡単なエディット機能を提供し，対応するモジュールのアウトライ

ンを自動的に作成することができる．但し，この GUI エディタは，あまり安定していないし，多くの必要な機能が提供されていない．例えば，SOFL 非形式仕様と半形式仕様の作成を支援する仕組みは提供されていない．

形式仕様パターンの定義と分類についての研究調査は，形式仕様パターンとは何か，一つのパターンの構造は何か，パターンの使い方は何か，パターンの表現は何か，どんなパターンが必要かなど課題について既存の関連研究を調査したり，研究グループ内で議論したり，小さい事例研究によってそれらの概念の本質を研究した．目的としては，システムの操作の機能に関するアイディアが明確に分かるなら，形式仕様記述技術を使えない一般の実務者でも，簡単に形式仕様を作成することができるように支援することである．この研究によって，本研究の研究者は，形式仕様パターンについて基本概念，目的，将来の使い方などについてある程度明らかにしていた．但し，この研究は非常に未熟で，進展は限られている．従って，本委託研究の中で，この研究を続けることを考えた．

SOFL 仕様に基づく顧客とコミュニケーションを強化するために，SOFL 三段階形式仕様記述プロセスに，プログラミングによるプロトタイピング技術を導入する可能性に関する研究を実施した．プロトタイピングとは，あるプログラミング言語（例えば，C#，Java など）で，要求仕様を理解した上で，顧客が要求するシステムの動的なモデルを作成することである．このようなモデルのレベルは，いろいろ可能性がある．あるプロトタイプモデルは，必要な操作の形だけを用意して，その振る舞いは全然提供しないが，他のプロトタイプモデルは，一定の入力を受け取って，ある程度の振る舞いを示すことができる．但し，この研究によって明らかになったのは，プロトタイピングには時間がかかるし，得たプロトタイプのコードの品質が良くないため，最終のシステムプログラムへ進化させることができないことである．これでは現在，企業でよく採用しているアジャイルソフトウェア開発アプローチではあまり役に立たない．より速く顧客の要求を確認できる仕様アニメーション技術の方がよいことが明らかになった．

(3) 本委託研究以前に実施していた研究と本委託研究との関係について

前述した以前に実施した研究は，本委託研究の基礎になっている．本研究でそれらの成果を踏まえ，更に発展してより成熟な技術を確立した．具体的な関係は，以下の通りである．

① 本研究以前に実施した SOFL 形式仕様作成の支援ツールについての研究を踏まえ，本委託研究は次の新たな進展を獲得した．

(a) 本研究で開発した SOFL-SpecTool は，形式仕様の作成だけではなく，非形式仕様と半形式仕様の作成も支援する．

(b) 非形式仕様を作成する際，システムの機能，データリソース，および制約項目の前に適切な階層番号を自動的につけることができる．

(c) 非形式仕様，半形式仕様，および形式仕様の階層的なモジュール構造を GUI で表示することができる．

(d) 形式仕様に使われる CDFD をより効率的に描けるためには，CDFD の全てのコンポーネントのアイコンを GUI で表示している．

(e)綺麗な CDFD を描けるために，CDFD の画面が自由に拡大することができるし，CDFD の構文も自動的にチェックすることができる．
(f)CDFD によって対応するモジュールのアウトラインを自動的に生成し，CDFD とモジュールのコンポーネントの一致性を自動的に保証できる．
(g)高いレベルのプロセスの機能を低いレベルの CDFD に分解する機能を支援する．

② 本委託研究以前に実施していた SOFL 三段階形式仕様記述プロセスにプロトタイピング技術を統合することは，本研究で開発している SOFL 三段階形式仕様記述プロセスに仕様アニメーション技術を導入する基本的なアイディアに似ている．但し，本研究での提案は，プログラミングを実施する必要がなく，それより簡単に使える Microsoft の Add-in Express ソフトウェアを用いて設計された部品とテンプレートの使用を支援するツールで，非形式仕様と半形式仕様のアニメーションを行う．同じ効果を期待でき，実現する技術はより簡単である．Microsoft の PowerPoint ソフトウェアを使えば，実務者の誰でも簡単に仕様のアニメーションを実施することができる．この技術を実現するためには，本研究でその技術を支援するツールのプロトタイプを開発した．具体的には，次の進展を得た．
(a)SOFL 非形式仕様に定義された機能間の依存関係を反映する「機能構造図」を定義し，それによって非形式仕様のアニメーションを実施するアプローチを確立した．
(b)SOFL 半形式仕様に定義されたモジュールによって顧客が最も関心を持っているプロセスのアニメーションを実施するアプローチを確立した．
(c)Microsoft の Add-in Express ソフトウェアを用いて，SOFL 非形式仕様と半形式仕様のアニメーションを作成する支援ツールを開発した．

③ 形式仕様のアニメーションについては，本研究以前に CDFD によって生成されたシステム機能シナリオの一つずつのアニメーションを実施するアプローチを提案していたが，そのアプローチの詳細および支援ツールはまだできていなかった．本研究は，以前の研究成果を踏まえ，次の具体的な進展を確保した．
(a)CDFD からシステム機能シナリオを自動導出するアルゴリズムと支援ツールを実現した．
(b)一つのシステム機能シナリオによって,そのシナリオのアニメーション条件を形成する手法を確立した．
(c)一つのシステム機能シナリオの振る舞いをアニメーションする機能を支援ツールで実現した．

④ 形式仕様パターンに基づく操作の形式仕様を作成するアプローチについては,本研究以前に実施した研究が，形式仕様パターンの非形式定義と分類に集中していた．本研究では，次の具体的な新たな進展を得た．
(a)形式仕様パターンの形式的定義と分類を明らかにした．
(b)知識とする形式仕様パターンのコンピュータ内部での階層的な有限状態遷移図の表現形式を設計した．
(c)仕様パターン知識の検索と適用に必要なアルゴリズムを提案した．

(d)仕様パターンに基づく操作の形式仕様作成アプローチの支援ツールを開発した.

2.1.3 研究目標

本委託研究以前の研究状況，8 ヶ月という本研究の研究期間，および劉研究室の研究体制を考えた上で,本研究の到達目標を設定した.具体的には,次の到達目標になっている.

① SOFL の非形式，半形式，および形式仕様のアニメーション技術を確立し，既存の SOFL 三段階形式仕様記述プロセスに統合する手法を確立する.

② 形式仕様パターンに基づき形式仕様を容易かつ効率的に作成する技術を確立する.

③ 上記①，②(b)で述べている手法，技術を含む SOFL 三段階技術の支援ツールのプロトタイプを開発する.

上記①で述べている非形式，半形式と形式仕様のアニメーション技術によって，システム要求に対するユーザと開発者からのフィードバックを有効に獲得し，開発者とユーザとのコミュニケーションを促進することができると考える．上記②で述べている形式仕様パターン技術によって，形式仕様を容易かつ効率的に作成することができるだけでなく，システム設計のアイディアを開発者から抜き出す効果も期待できると考える．上記③で述べている支援ツールは，以上の①，②の手法，技術を含む SOFL 三段階技術を実用化するために必要な技術である.

上記の到達目標を達成するために，各技術に関する手法の部分と支援ツールに分けて考えた．基本的には，研究目標として，手法の一部分の内容を確立することを目指し，支援ツールの一部分を開発することを考えて設計した．残っている時間内で，支援ツールの開発を続けると共に，開発したツールの機能のテスト，事例研究，および支援ツールの完成と計画した.

具体的には，研究目標は，次の通り設定した.

(a)非形式仕様からアニメーションシステムのアーキテクチャーの構成方法を確立する.

(b)半形式仕様からアニメーションシステムのアーキテクチャーの構成方法を確立する.

(c)形式仕様パターンの定義と仕様パターンシステムの構造を確立する.

(d)形式仕様パターン知識の表現，検索，および適用仕組みを提案する.

(e)形式仕様パターンシステムを支援するツールの一部を作成する.

(f)CDFD から機能シナリオを自動的に生成するツールを作成する.

目標(a)の焦点は，SOFL 非形式仕様に基づくアニメーションシステムの構造をどのように構築するかという課題である．目標(b)の焦点は，SOFL 半形式仕様に基づくアニメーションシステムの構造をどのように構築するかという課題である.目標(c)の焦点は,形式仕様パターンの形式的な定義および役目によって仕様パターンの分類の定義を確立することである.目標(d)の中心課題は,仕様パターンが知識としてどのように表現し,その表現形式により，どのように必要に応じて仕様パターンを検索かつ適用できるかという問題である.目標(e)の主な課題はどのように形式仕様パターンに基づくプロセス形式仕様作成アプローチを有効に支援するかということである．目標(f)の焦点は，SOFL 形式仕様アニメーションを実施するために,まず CDFD からシステム機能シナリオというデータフローパスをどのように自動的に抜き出すかという問題である.

2.2 研究の活動実績・経緯

この節では，研究活動実績，内部・外部打合せの実施状況等の実施状況に分けて説明する．

2.2.1 研究活動実績

本研究の活動実績を表 2-1 に示す．

表2-1　研究実施実績

作業項目	6月	7月	8月	9月	10月	11月	12月	1月
1. 研究準備 (a)PowerPointを用いて非形式仕様と半形式仕様のアニメーション技術についての研究	→							
(b)形式仕様のアニメーション技術についての研究	→							
(c)過去の形式仕様パターンの研究を参考しながら、パターンシステムの構造、知識表現、およびパターン知識の適用についての研究	→							
(d)既存の支援ツール開発の経験に基づき本研究に計画された支援ツール開発についての研究	→							
2. 中間目標の達成 a. 非形式仕様からのアニメーションシステムのアーキテクチャー構成方法の確立		→	→	→				
b. 半形式仕様からのアニメーションシステムのアーキテクチャー構成方法の確立		→	→	→				
c. 形式仕様パターンの定義と仕様パターンシステムの構造の確立		→	→	→				
d. 形式仕様パターン知識の表現、検索、および適用仕組みの提案		→	→	→				
e1. 形式仕様パターンシステムを支援するツールの一部作成		→	→	→				
e2. CDFDから機能シナリオを自動的に生成するツールの作成		→	→	→	→	→		
3. 中間報告の準備				→	→			
4. 最終成果のとりまとめ （1）中間報告及び各研究者からの助言をフィードバックする						→		
（2）成果報告書の構成案						→	→	
（3）成果報告書の作成						→	→	
（4）成果概要プレゼン資料								→
5. 成果物の納品								→

2.2.2 内部・外部打合せの実施状況

本節で内部打合せの実施状況と外部打合せの実施状況に分けて別々に説明する．

(1) 内部打合せの実施状況

研究チームの内部で，次の通り研究打合せを実施していた．

① 前述の 2.2.1 で述べていたように，毎月のプロジェクト会議で，本研究責任者は各協力研究者と情報交換を行い，計画した研究内容の完成状況を確認，遭遇した課題を議論，新たな研究活動を計画した．

② 本研究責任者は隔週に 1 回のペースで，各協力研究者と個別にその研究者が担当している研究内容について打合せを実施した．そこで，新たな課題または問題点が分かったら，迅速に議論して解決方法を決めていた．

(2) 研究チーム外部打合せの実施状況

本研究を推進するために，研究チーム外部の専門家，企業で実際のソフトウェア開発状況を把握している研究者，第 19 回アジア太平洋ソフトウェア工学国際会議（APSEC2012）に出席していた研究者と研究打合せを実施した．

具体的には，次の活動を実施した．

① 大連理工大学および大連海事大学のソフトウェア工学分野で活躍している研究グループの教員と大学院生たちと，実用性が高い SOFL 形式工学手法と SOFL 支援ツールに関する技術を，7 月 25 日から 27 日に渡り議論した．具体的に，実施した打合せの内容と明らかにしたことは，次の通りである．

(a) SOFL 形式工学手法の支援ツールの Graphical User Interface（GUI）の設計について議論した．SOFL 形式工学手法の支援ツールの GUI は情報量が足りないと操作し難い問題点が指摘された．特に，CDFD を描くために必要なアイコンが GUI に適切に表現しておらず，CDFD と対応するモジュール仕様の表現は分かりづらい．更に，SOFL 非形式仕様，半形式仕様，および形式仕様を作成するために一つにまとめる階層的な構造がないため，支援ツールの有用性が弱いということが明らかになった．

(b) SOFL の非形式仕様アニメーションと半形式仕様アニメーションをソフトウェアツールによってどのように支援すれば，顧客と仕様の作成者間のコミュニケーションを強化することと，顧客からのフィードバックを簡単に得られることに集中して議論した．その結果，throw away prototyping と PowerPoint アニメーションシステムの活用という二つの選択が分かった．前者は時間がかかるが，後者は既存の PowerPoint ソフトウェアに新たなコンポーネントを加えて，新たなアニメーション用のソフトウェアツールの開発が必要だと分かった．

(c) 既存の SOFL 形式工学手法の長所と短所について，SOFL 支援ツールの機能およびソフトウェア品質の保証の視点から議論した．その時の SOFL 支援ツールには，形式仕様記述プロセスの自動化を支援する機能が不足していることを，この議論によって明らかにした．この意見をその後行った形式仕様パターンに基づく形式仕様作成の支援ツールの開発に取り込んだ．

(d) SOFL 形式工学手法と支援ツールをソフトウェア要求分析，システム設計，および実装されたコードのテスト工程の支援方法について議論した．特に，仕様間のトレーサビリティを対話的または自動的に構築する上で，半形式仕様と形式仕様のアニメーションを実施するアイディアを認識した．

(e) SOFL 支援ツールの分散システム，組み込みシステムの開発に適用する際の問題点に

ついて討論した．リアルタイム性質と分散したコンピュータ間のコミュニケーションの仕組みを形式仕様で記述した上で，アニメーションを通じて検証する可能性が明らかになった．具体的には，時間変数を宣言し，プロセスの事前条件と事後条件において，プロセスが実行する時間をその変数によって適切に定義し，その定義した時間をアニメーションすることによってそのプロセスのリアルタイム性質を確認することが可能である．また，分散コンピュータそれぞれをプロセスで表し，プロセス間のデータフローコミュニケーションをアニメーションすることにより，分散コミュニケーションの仕組みの正当性を確認することができるということも明らかにした．

② ソフトウェア開発方法について活躍している四川大学ソフトウェア学院の教員と大学院生・大学生と本研究の支援ツールの様々な課題について 9 月 11 日から 14 日までに渡り議論した．具体的に，実施した打合せの内容とそれによって明らかにしていたことは，次の通りである．

(a) SOFL 形式仕様アニメーションを実現するために形式仕様に基づくコードのテスト技術を利用する可能性について議論した．それにより，SOFL のシステム機能シナリオに含まれるプロセスの形式仕様からそのシナリオの振る舞いを定義する論理積を導出した上で，その論理積に基づくテストケースの自動生成することによって，その機能シナリオの振る舞いを動的にアニメーションすることが可能であるアイディアを理解した．

(b) SOFL 支援ツールの問題点について議論した．その結果，SOFL の非形式仕様から半形式仕様へ，さらに形式仕様へ変換することを支援する機能がないので，SOFL 三段階仕様技術の適用性に問題点が判明した．

(c) SOFL 形式工学手法と支援ツールは，要求分析，設計，およびユーザとのコミュニケーションの強化に応用することについて討論した．これにより，二つの新たな機能を開発し，既存の SOFL 支援ツールに追加する必要性が分かった．一つ目は，非形式仕様の制約セクションにおいて定義された制約項目は，どちらのデータ項目または機能に関係があるかを記録する仕組みとその仕組みの支援が必要である．二つ目は，非形式仕様のアニメーションは，現実世界の仮想環境でユーザインタフェースに集中してチェックすることが重要であることを認識した．

(d) SOFL 支援ツールをソフトウェア開発産業へ適用する際の有用性（usability）について議論した．これにより，SOFL 形式仕様に含まれたシステムのアーキテクチャーを定義する CDFD を描くときに，同じレベルの CDFD の規模が制限されているため，大規模システムのモデリングにはプロセスの分解活動を支援する機能が必要であることを確認した．更に，企業で一つの開発チームには管理者，クライント，設計者，プログラマなど多様な人が参加しているため，全員が SOFL の形式仕様を容易に理解できる訳ない．SOFL 形式仕様によってシステム機能についての検討を有効に行うためには，階層的な形式仕様の CDFD とモジュールを抽象的に表現する機能と，抽象表現と階層的な表現の間の連接機能を SOFL 支援ツールに追加する必要があることが分かった．

③ 本研究の進む方向と研究内容は，同じ分野での位置付けを正しく理解するためには，7月5日に開催したワークショップの場で，日本国内で形式手法とソフトウェア工学分野の活躍している研究者と指導者からの意見と経験を聞いた．特に，北陸先端技術大学院大学の二人の先生が既存の CafeOBJ 形式手法と形式仕様の形式検証技術と，国立情報研究所の一人の先生のテストとモデル検査の融合技術についての紹介により，本研究で目指す実用的な形式工学手法の研究開発に役に立つ経験，教訓を得た．その上で，本研究責任者は，本研究の基本アイディアを先生方に説明し，方向性と具体的な技術の役割などを議論してもらい，本研究で開発している技術の将来性を確認した．

④ 本研究の成果は，企業にとってどのように役に立つかと，改善すべきところを確認するために，産業技術研究所（AIST）から三名の研究者を法政大学に招聘して 11 月 7 日に共同ゼミを開催していた．このゼミで，本研究責任者は，まず“実用性が高い SOFL 形式工学手法と支援ツール”について講演し，本プロジェクトで研究している SOFL 三段階形式仕様記述技術，仕様アニメーション，および仕様パターンによる形式仕様の作成の基本原理および支援ツールの開発の現状について説明した．その後，AIST の三名の研究者からいろいろな質問を受けて，次の要望が分かった．

(a) SOFL 三段階形式仕様記述プロセスの中で，できれば上流仕様から下流仕様への変換は，完全に自動化することが望ましい．

(b) 非形式仕様からその仕様のアニメーションシステムを自動的に作成することができれば良い．

(c) 形式仕様を作成してから，その形式仕様をシステムの実装や検証に有効に利用することが望ましい．

これら要望の中で，(a)と(b)に対して完全に実現することは不可能であることを説明し，(c)に対しては，形式仕様に基づくプログラムの作成とプログラムのテストを行う可能性があり，それを実現することが本研究の目標ではないが，将来の発展に取り込むことが可能と本研究責任者が説明した．

その後，三人の協力研究者から，“SOFL 三段階形式仕様記述技術の支援ツール”，“非形式仕様と半形式仕様のアニメーション支援ツール”，および“形式仕様パターンに基づく形式仕様の作成支援ツール”について紹介した．これらの紹介に対して，AIST の三名の研究者から質問を受け，議論を行った．それにより，次の要望が分かった．

(d) 半形式仕様の内容は形式仕様を作成するときに，自動的に再利用することができれば設計者にとって便利である．

(e) 半形式仕様を形式仕様に再利用するときに CDFD との一致性を守ることが課題であり，必要である．

(f) アニメーションシステムを作成するために，ユーザが新しい部品や機能のテンプレートをその支援ツールに自由に追加することができる機能があれば便利である．現状としては，新しい部品やテンプレートを支援ツールに追加するときに，プログラマが，プログラムの作成によって実現する．但し，本研究プロジェクト以内で，その機能を追加する時間がないということも分かった．

(g)非形式仕様のアニメーションに使用された部品や機能のテンプレートなどは半形式仕様のアニメーションに再利用することができれば，コストの減少に役に立つ.

(h)仕様パターンに基づく形式仕様作成の支援ツールは，非形式仕様または自然言語で書かれた論理式を理解することができれば，便利である．このポイントに対し，実現できないと本研究者は判断した．但し，規定した自然言語であれば，可能性があると考えられるが，それを実現するには，時間が必要であるため，本研究内ではできない.

(i)形式仕様作成の支援ツールの有効性を確かめることが必要である．これに対して，本研究の事例研究の部分で対応する.

⑤ 形式手法の分野で世界有名なシンガポール国立大学の先生を11月12日に法政大学へ招聘し，本研究の内容について打合せを実施した．本研究責任者は，まず“実用性が高いSOFL形式工学手法と支援ツール”の目的と具体的な技術について説明した.これに対し，その先生はこのような研究開発が非常に重要だと賛成した上で，次のポイントを指摘した.

(a)形式仕様パターンに基づく形式仕様の作成には，十分な経験を反映する仕様パターンが大事である．それらの仕様パターンの作成は，挑戦的な課題である.

(b)SOFL三段階形式仕様記述技術は，企業の環境で実際に適用することが重要である.

これに対しては，本研究責任者と協力研究者が，指摘された課題についてその先生と議論した．本研究の期間内でそれらの課題を取り込むことはできないが，将来の研究で対処することが必要だと考えた.

⑥ 第19回アジア太平洋ソフトウェア工学会議(APSEC2013)で形式手法，要求工学，およびシステム検証・テスト分野で活躍している研究者と，本研究の内容を含む様々な課題についての情報収集と打合せを実施した．具体手的には，次の活動が本研究に関連している.

1)デンマークのDines Bjorner教授の基調講演“A Survey of Formal Methods in Software Development”によって，ソフトウェアモデリングが，(a)応用領域知識のモデリング，(b)要求工学(要求仕様作成)，および(c)システムモデリング(設計仕様作成)という三つの段階に分けていることが分かった．第(a)段階で，数学概念を用いて応用領域知識を抽象的に定義することが重要であり，第(b)段階では，数学に基づく設計された形式仕様記述言語でユーザ要求を明確に定義することが役に立つ．この数学言語は，VDM, Z, B-Method, Event-B, RAISEなどが挙げられた．第(c)段階では，これら形式仕様記述言語でシステムを設計しながら，システムの機能を明確に定義することができる．この講演で主張した形式手法の有効性に対していくつか厳しい質問があった．講演者は，形式手法が実際のソフトウェア開発に有効であることを説明したが，本研究責任者を含める多くの研究者が納得しなかった.但し，この講演により，欧州の国々の形式手法に対する立場，応用状況，および直面している課題の情報を理解することができた．ため，良い結果でした.

2)イギリスの Jeff Kramer 教授の基調講演“Wither Software Architecture？”によって，ソフトウェアアーキテクチャーについての歴史，研究課題，および Kramer 教授自身が研究開発したソフトウェアアーキテクチャー表現言語などを理解した．既存のアーキテクチャー表現言語はシステム構造の意味を抽象的に表現する価値があることが分かった．但し，このようなアーキテクチャー言語はまだ企業で適用していないことも事実である．本研究責任者からの質問で，Kramer 教授と SOFL 言語にあるアーキテクチャーを表現する図言語 CDFD についての議論を行い，今後の CDFD を分散システムに適用への発展に役に立つ認識を得た．

3)Requirement Modelling and Analysis セッション，Formal Methods I,III,IV セッションに出席し，システムのセキュリティ性質をシステムの開発プロセスで保証する技術，自然言語で表した UML の Use-Case の分析方法，システム要求のトレーサビリティの活用アプローチ，システムのモデル検査，分析，およびテストなど様々な技術について情報収集と議論した．これら技術は，本研究で開発している SOFL 三段階漸進的形式仕様記述プロセスにどのように適用できるかを考えていた．直接的に適用することはできないが，世界の同分野で本研究の位置づけが分かった．SOFL 三段階漸進的形式仕様記述技術は，独特的なアイディアとアプローチを持つ世界の唯一の形式工学手法であることを明らかにした．

2.3 研究実施体制

本研究の研究実施体制は，研究責任者劉少英と劉研究室に所属している 12 名の大学院生のアルバイト研究者から構成されている．各研究者のプロフィールと研究の役割についての説明は，次の通りである．

(1) 本研究責任者　劉少英

① 学歴(大学卒業以降)

’82.01西安交通大学計算機科学学士 修了

’86.04西安交通大学ソフトウェア工学修士 修了

’92.10英国マンチェスター大学・形式手法・博士　(Ph.D)修了

② 職歴

’82～’88 西安交通大学･計算機科学，工程学科助教

’91～’93 英国ヨーク大学コンピュータ科学科研究員

’93～’94 英国ロンドン大学コンピュータ科学科　研究助手

’94～’00 広島市立大学情報科学部計算数理学科 助教授

’00～’01 法政大学情報科学部コンピュータ科学科　助教授

’01～現在 法政大学情報科学部コンピュータ科学科　教授

③ 主な論文・著書

1)著書

"Formal Engineering for Industrial Software Development using the SOFL Method", S. Liu, Springer-Verlag, March 2004, 424 pages, ISBN 3-540-20602-7.

2)論文

- “Formal Specification-Based Inspection for Verification of Programs”, S. Liu, Y. Chen, F. Nagoya, J. McDermid, **IEEE Transactions on Software Engineering**, 38(5), 2012, pp.1100-1122.
- “A Rigorous Method for Inspection of Model-Based Formal Specifications”, S. Liu, J. McDermid, Y. Chen, **IEEE Transactions on Reliability**, Vol. 59, No. 4, December, 2010, pp. 667-684 .
- “A Framework for Integrating Formal Specification, Review, and Testing to Enhance Software Reliability”, S. Liu, T. Tamai, S. Nakajima. **International Journal of Software Engineering and Knowledge Engineering**, 21(2), 2011, pp. 259-288 .
- “Integrating Prototyping into the SOFL Three-Step Modeling Approach”, F. Zainuddin, S. Liu. **13th International Conference on Formal Engineering Methods (ICFEM 2011)**, LNCS, Springer, Durham UK, Oct. 25-28, 2011, pp. 163-178 .
- “A Pattern System to Support Refining Informal Ideas into Formal Expressions”, X. Wang, S. Liu, H. Miao, **12th International Conference on Formal Engineering Methods (ICFEM 2010)**, LNCS, Springer, 17-19 Nov. 2010, Shanghai, China, pp. 662-677 .
- “SOFL: A Formal Engineering Methodology for Industrial Applications”, S. Liu, A Jeff. Offutt, C. Ho-Stuart, Y. Sun, M. Ohba, **IEEE Transactions on Software Engineering**, Vol. 24, No. 1, January 1998, pp. 24-45.

④ 役割

(a)本研究の総括.

(b)非形式仕様アニメーション，半形式アニメーション，および形式仕様アニメーションの仕組みに関する研究

(c)形式仕様パターンに基づく操作の形式仕様作成アプローチの原理に関する研究

(d)SOFL三段階漸進的形式仕様記述技術の支援ツールの設計とテストに関する研究.

(2)アルバイト研究者

劉研究室に所属している 14 名の大学院生のうち，12 名が本委託研究に参加し，本研究の次の研究活動に参加した.

(a)非形式仕様作成，半形式仕様作成，および形式仕様作成を含む SOFL 三段階形式仕様記述技術の支援ツールの設計と実装.

(b)形式仕様パターンに基づく操作の形式仕様作成の支援ツールの設計と実装.

(c)非形式仕様アニメーションと半形式仕様アニメーションの支援ツールの設計と構築.

(d)形式仕様アニメーションの原理の提案と支援ツールの一部の実装.

(e)上述の各支援ツールの統合.

(f)開発した支援ツールを用いた簡単化された JR 東日本 Suica カードシステム，航空券予約システム,およびスマート交通信号システムの事例研究によるその支援ツールのテストと改善策の提案.

３．研究成果

3.1 研究目標１「非形式仕様からアニメーションシステムのアーキテクチャーの構成方法の確立」

3.1.1 当初の想定

(1) 想定する仮説等

非形式仕様に記述されている機能，データリソース，および制約を考慮する上で，顧客の関心度が高い機能をアニメーションするシステムの構造を定義する．基本的には，一つの機能のアニメーションを実施するために,その機能に関わるインタフェース,振る舞い,および内部操作が必要である．このような要素を考えた上で非形式仕様からアニメーションするシステムの構造を研究する．想定される課題としては，PowerPoint でのアニメーションに必要なコンポーネント，インタフェースパターン，および計算パターンの設計と適用仕組みの定義にある．

(2) 当初の到達目標

Eclipse と Microsoft の.NET 環境を参考にし，PowerPoint でアニメーションに必要なコンポーネント，インタフェースパターンを設計し，全ての機能に使われる可能な計算パターン(例えば,「関係表現」,「情報検索」，および「情報更新」)を定義し設計する．この上で，これらのコンポーネント，インタフェースパターンおよび計算パターンを有効に組立て，要求される機能のアニメーションができる仕組みを構築する．

(3) 当初の期待される効果

提案した非形式仕様のアニメーションアプローチの期待される効果としては，顧客と開発者間のコミュニケーションを強化し，顧客からのフィードバックを容易に獲得し，非形式仕様の改善に支援することである．

3.1.2 研究プロセスと成果

(1) 研究プロセス

この研究については，次の三つのステップを取った．

① 第 1 ステップ：非形式仕様の機能に対する関心度の決定方法を研究
② 第 2 ステップ：高い関心度を持つ機能についてアニメーションシステム構造の作成方法を研究
③ 第 3 ステップ：第三歩，アニメーションシステムの構造から PowerPoint アニメーションの作成方法を研究．

(2) 具体的な研究成果の内容

本研究の成果としては，三つである．一つ目は，SOFL 非形式仕様に記述されたシステム機能の関心度を，基本的に記述された機能の階層的な構造のトップレベル機能を顧客の最も関心を持っている機能として選出する方法を確立した．二つ目は，SOFL 非形式仕様において定義された顧客が最も関心を持っている機能とその機能を実現するために必要なユーザインタフェースと内部機能の関係を表す「機能構造図」というアニメーションシステム構造の作成方法を確立した．三つ目は，その「機能構造図」に基づき非形式仕様の PowerPoint アニメーションの作成方法および支援ツールを開発した．この結果は，基本的に当初の想定する仮説通りに進んできて，当初の到達目標も達成した．但し，当初の期待される効果に対する達成状況については，完全に評価できていない．その理由は，二つである．一つは，時間がないためである．もう一つは，開発された支援ツールは，まだプロトタイプであり，アニメーションシステムの作成にとって必要なアイコンやテンプレートなどの要素が不十分である．

次に，SOFL 非形式仕様からアニメーションを実施するまでの主な技術点を比較的詳細に紹介する．

① 非形式仕様に基づく「機能構造図」の生成

非形式仕様アニメーションは，仕様に定義された顧客の関心が最も高いシステム機能を，分かりやすく動的に表現することを目指す．一つの機能を動的に表現するために，図 3-1-1 に示したように，適切な GUI (Main UI, level 1)，機能自体 (FunctionName, level 2)，および内部操作 (Internal process, level 3) という三つのコンポーネントが構築されることが必要である．

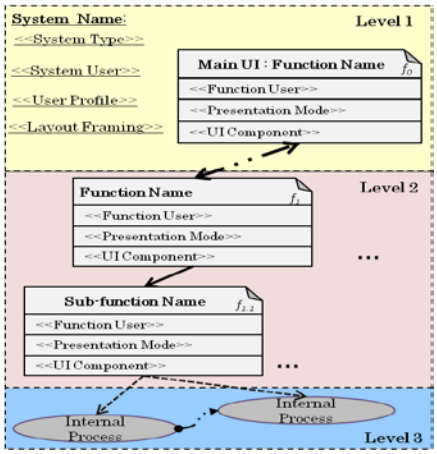


図3-1-1　機能構造図の一画

GUI は，まずアニメーションするシステム全体の関連情報を定義する．これは，システム型(System type)，システムユーザ(System user)，ユーザのプロフィール(User profile)，およびレイアウト構造 (Layout framing) を含む．システム型は，システムの種類を示す．

例えば，ウィンドウズに基づくシステム（Windows-based system），ウェブに基づくシステム（Web-based system），またはモーバイルに基づくシステム（Mobile-based system）などを選択できる．システムユーザは，アクセスに無制限ユーザ，アクセスに制限があるユーザ，または両方を選択できる．ユーザのプロフィールは，本システムを使う人間の特徴を定義する．例えば，ユーザは必ず 20 歳以上の女性であるとか，レイアウト構造は，GUI で使うコンポーネントを示すとか，フレームの数，ボタンの位置，メニューの位置など．

次に，アニメーション機能を実現するために必要な入力，出力，メッセージのディスプレイエリアなどを明確に表現する．このため，いくつかの関連情報項目を明確に定義することが必要である．それらの項目は，次の通りである．

(a)機能名 (Function name)．これは，アニメーションする機能の名前を示す．この項目は，アニメーションを見るユーザにどんな機能のアニメーションを実施していることを知らせる．

(b)機能のユーザ（Function user）．これは，アニメーションする機能のユーザの種類を示す．例えば，アクセスに無制限のユーザ，アクセスに制限があるユーザ，または両方

(c)表現モード (Presentation mode)．これは，アニメーションのスタイルを示す．例えば，機能の振る舞いを示すのみアニメーション，対話的アニメーション，あるいは他のユーザが定義するスタイルでアニメーション．

(d)インタフェースコンポーネント（UI components）．これは，GUI に使える必要なコンポーネントを示す．例えば，情報の検索用ボタン，メッセージのディスプレイ用ウィンドウエリア，情報の更新用メニュー項目など．

機能自体は，開発者が理解したその機能の振る舞いを表す全てのパターンの表現である．一つのパターンは，その機能を実現する操作の入力，GUI 状態遷移図，および出力から構成された複合データ項目である．このデータ項目によってその機能の振る舞いの動的な表現を作成する．また，本機能は，他の顧客が最も関心を持っているある機能を使う可能性もある．

内部操作は，その機能の振る舞いを表現するために必要な操作（例えば，必要な量の計算，データ項目の検索，情報の更新，データ項目の追加・削除など）である．これら操作は，直接に顧客と関係がないが，顧客が最も関心しているその主な機能の振る舞いを実現するために必要なものである．

非形式仕様において記述されたすべての顧客が最も関心を持っている機能に対して，このような三つのコンポーネント間の関係を表現する図の集合は，非形式仕様に基づき生成される機能構造図である．

② 非形式仕様のアニメーションシステムの実施

前述した機能構造図によって非形式仕様のアニメーションを実施する．このため，主な機能に関わるその三つのコンポーネントの詳細を決め，機能の入力，出力，振る舞い，必要なメッセージなどをどのように動的に表現するかを明らかにすることが必要である．

コンポーネントの詳細を決めるためには，以下を参考にすることが必要である．

(a)非形式仕様において記述された機能の内容

(b)開発者が要求機能に対する理解および開発者の経験

(c)顧客からの意見

③ 非形式仕様アニメーションの事例

本節で具体的なトラベルエージェントシステム ITAS（Inspiring Travel Agent System）を事例として使い，上述した非形式仕様アニメーションシステムの作成における主な技術を説明する．

ITAS は基本的に二つの種類のサービスを提供する．一つはオンライ航空券予約サービス，もう一つはホテルの予約サービスである．さらに，これらサービスに関連するサービスも提供する．例えば，旅程管理，領収書の発行など．このシステムを利用するすべてのユーザは予め登録することが必要である．

このシステムのビジネス内容を十分に理解した上で，システムに対する要求の非形式仕様を作成した． これによって機能構造図を作成した．図 3-1-2 に作成した非形式仕様と機能構造図間の関係を示している．

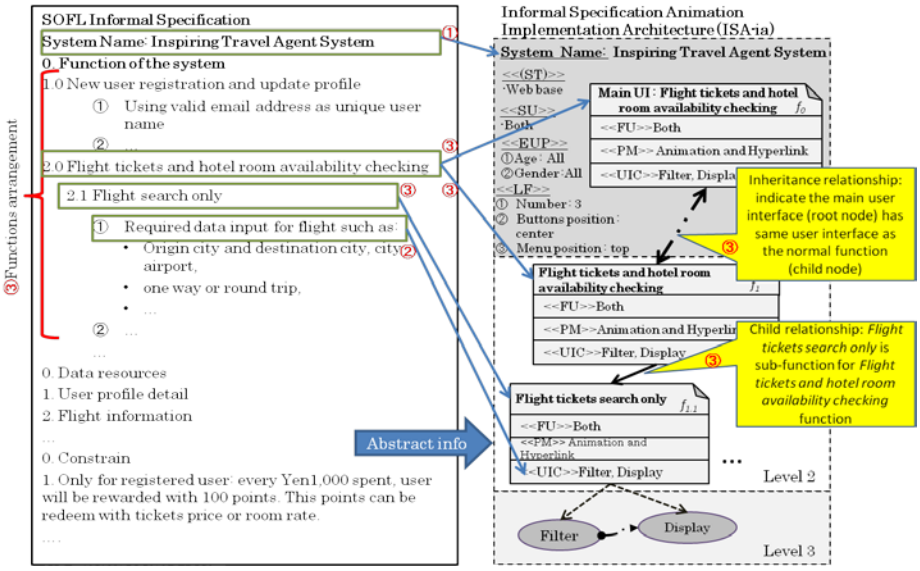


図3-1-2　非形式仕様と機能構造図間の対応関係

機能構造図に記述されたシステムの関連情報項目と主な機能の関連情報項目は，図 3-1-3 に示した通り，次の事前に良く設定された質問に対して答えながら発見していた．

(a)誰が本システムのユーザになるか．そのユーザのプロフィールは何か．

(b)誰がアニメーションする主な機能のユーザになるか．

(c)いくつかのレイアウト構造が必要であるか．形はどんなものであるか．

(d)どんな内部操作が必要であるか．

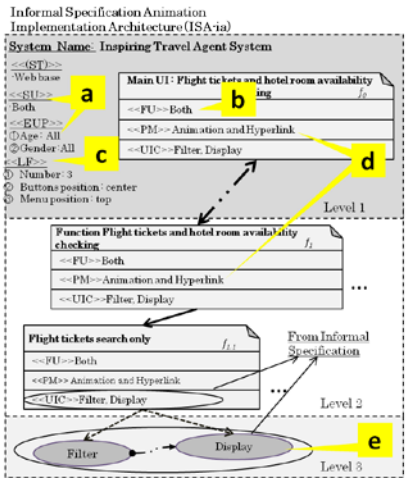


図3-1-3　システムと主な機能の関連情報項目

生成された機能構造図と非形式仕様の内容に基づく，支援ツールを用いてその仕様のアニメーションシステムを作成した．図 3-1-4 に機能構造図と実施したアニメーション画面との関係を示している．

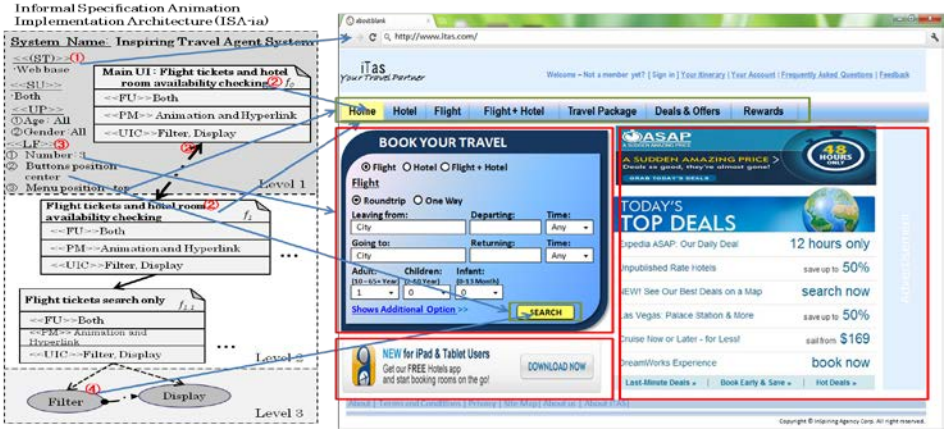


図3-1-4　機能構造図と作成したアニメーション画面との関係

④　非形式仕様アニメーションの支援ツール（プロトタイプ）

非形式仕様アニメーションを素早く行うことができるために，Microsoft の PowerPoint ソフトウェアにアニメーション用のコンポーネントを追加することによって開発した．この支援ツールは，非形式仕様アニメーションと半形式仕様アニメーションと共に支援する機能を提供している．本節では，非形式仕様アニメーションの支援機能に集中して説明する．これは，支援ツールの全体構造とアニメーション用のコンポーネントアイコンおよびテンプレートを含める．半形式仕様アニメーションの特有の支援機能は，後で半形式仕様アニメーションについて論述する節に詳しく紹介する．

(3) 支援ツールの全体構造

Microsoft の PowerPoint ソフトウェアに追加した仕様アニメーション用の主なコンポーネントは，図 3-1-5 に示したように，次の四つである．

(1)SOFL 非形式仕様アニメーション（SOFL Informal Specification Animation）

(2)SOFL 半形式仕様アニメーション（SOFL Semi-Formal Specification Animation）

(3)SOFL 仕様エリア（SOFL Specification）

(4)非形式仕様の機能構造図（Informal Specification Animation Implementation Architecture）

これらコンポーネントは，GUI においてリボンタブ（Ribbon tab）で表され，そのタブをクリックすると，対応する機能を果たす．

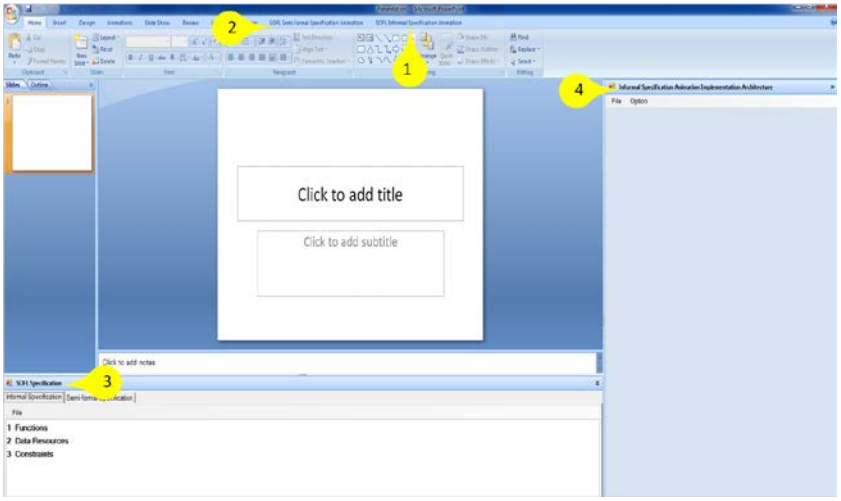


図3-1-5　MicrosoftのPowerPointに追加したコンポーネント

SOFL 非形式仕様アニメーションのリボンタブに，図 3-1-6 に示す通り，コンポーネントアイコンと機能テンプレートが含まれている．

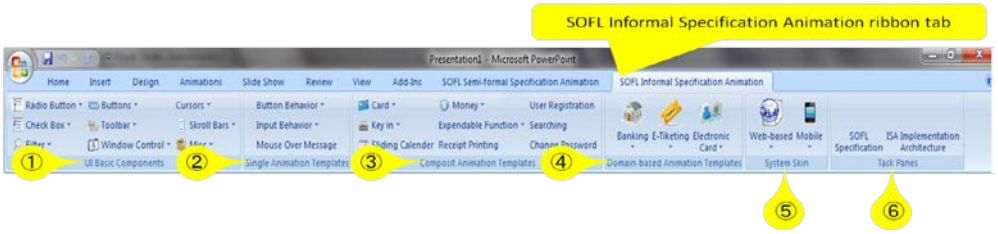


図3-1-6　Microsoft PowerPointに追加したコンポーネントアイコンとテンプレート

これらコンポーネントアイコンとテンプレートは，四グループに分類して，GUI での表示と役割を，表 3-1-1 に説明している．

表3-1-1　非形式仕様アニメーション用のコンポーネントの説明

グループ	コンポーネント (ボタン)	特徴，役割および事例
① ユーザインタフェールの基本コンポーネント	• Radio Button • Check box • Filter • Buttons • Toolbar • Window Control • Cursors • Scroll Bars • Misc	特徴： • Basic user interface components that always found in any kind of system user interface. • The components are provided in vector image that can be resized according to user requirement • The components are static without any animation 役割 • It is used to represent actual user interface components that exists in normal system user interface. 事例 • The example for Radio Button usage is illustrated in 図 3-1-7..
② 単一の機能アニメーションテンプレート	• Button Behavior • Input Behavior • Mouse Over Message	特徴 • One or more components with animated feature thATMimicking simple single system user interface component behaviour. • This animation template only involves animated components in a single slide 役割 • Instance simple animated component that can be applied in any suitable area in the informal specification animation project. 事例 • The example illustrates in 図 3-1-8..
③ 複合機能アニメーションテンプレート	• Card • Key In • Sliding Calendar • Money • Expendable Function • Receipt Printing • User Registration • Searching • Change Password	特徴 • More complex animation template that constructed from more than one graphic component and mimicking more complex user interface behaviour. • Reusable and customizable for both graphic components and the animation features according to user requirement. 役割 • Focus on popular function that normally seen in

		a system • By using the template able to reduce informal specification animation construction time. • Source of breaking the ice for beginners in constructing complex animation behaviour. 事例 • Omitted for the sake of similarity to the example given in 図 3-1-8.
④ ドメイン別機能アニメーションテンプレート	• Banking • E-Ticketing • Electronic Card	特徴 • Collection of complex animation templates that grouped according to specific domain. 役割 • Ready made animation template that by using them able to reduce informal specification animation construction time. • Source of breaking the ice for beginners in constructing complex animation behaviour. 事例 • The example illustrates in 図 3-1-9.
⑤ システム型	• Web-based • Mobile	特徴 • Selection of special slide master that reflect system environments. • Once applied the selected system skin, it will reflect the entire slide in the pptx file. 役割 • Create instance system environment according to the system theme 事例 • The example illustrates in 図 3-1-10.
⑥ タスクパーネ	• SOFL Specification • Functional Structure Diagram	特徴 • Alternative button to open the task panes 役割 • Alternative way to launch SOFL specification that includes informal and semi-formal specification, and functional structure diagram task panes.

(4) 各種のコンポーネントとテンプレート

非形式仕様アニメーションのために，各種のコンポーネントとテンプレートが支援ツールに導入された．図 3-1-7 にユーザインタフェースコンポーネントの使用事例が示されている．図 3-1-8 に機能アニメーションテンプレートの使用事例が示されている．アニメーションを効果的に行うために，ドメイン別のコンポーネントと機能テンプレートの使用が有効であろう．図 3-1-9 にドメイン別機能アニメーションテンプレートの使用事例が含まれている．図 3-1-10 にシステム型テンプレートの使用事例が示されている．

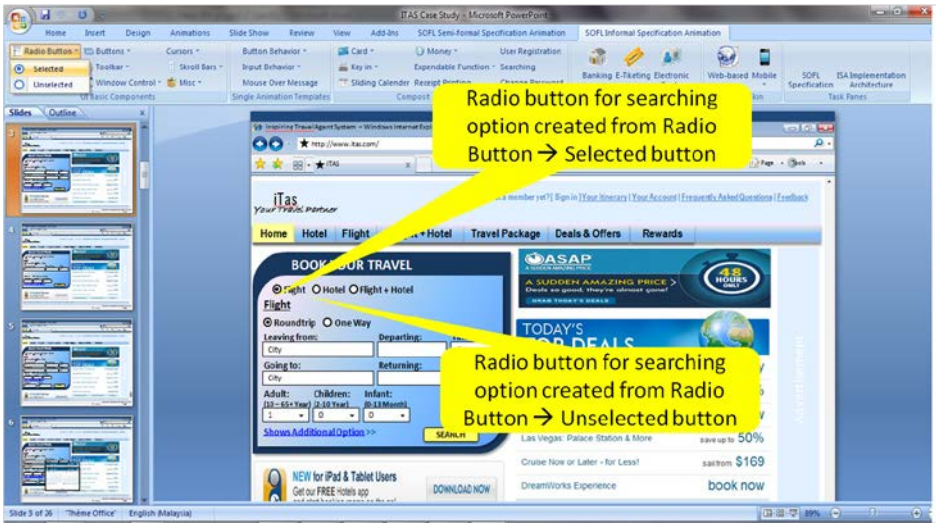


図3-1-7　ユーザインタフェースコンポーネントの使用事例

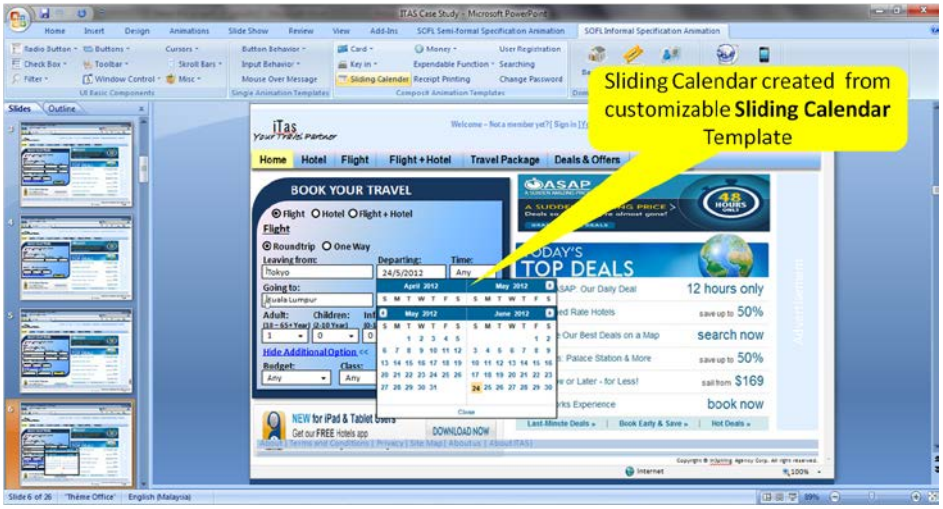


図3-1-8　機能アニメーションテンプレートの使用事例

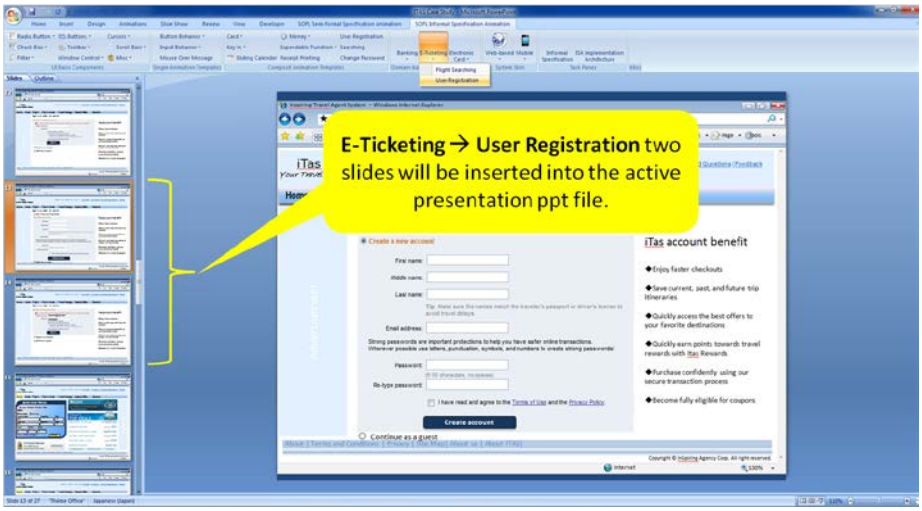


図3-1-9　ドメイン別機能アニメーションテンプレートの使用事例

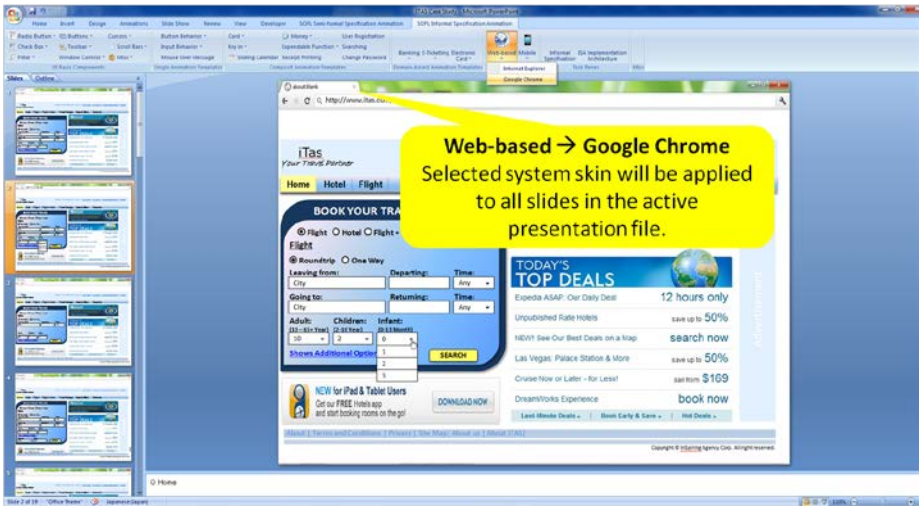


図3-1-10　システム型テンプレートの使用事例

3.1.3 実用化へ向けた課題と問題点

(1) 課題と問題点

現時点は，非形式仕様からアニメーションシステムの作成までのプロセス，方法，および支援ツールのプロトタイプができていて，実用化するためには次の二つの課題が残っている．一つは，非形式仕様の内容から「機能構造図」を系統的かつ厳密的に導出する手法を実現していない．もう一つは，導出された「機能構造図」によりどのように系統的にアニメーションシステムを構築するかをまだ解明していない．

問題点としては,非形式仕様の内容は,曖昧性を持つ自然言語で書いたものであるため,

それによって機能構造図の系統的な導出手法を厳密に定義することが不可能に近いことである．同じような問題点は，機能構造図によりアニメーションシステムの系統的な構築方法にもある．

(2) 将来の応用方法

将来の応用方法に関しては，いろいろな可能性がある．一つの可能性は，SOFL 非形式仕様を用いてソフトウェアシステム開発する場合，本研究で提案したアニメーションアプローチに従って，顧客と協力しながらアニメーションを行うことによって，開発者と顧客間のコミュニケーションを強化することができ，非形式仕様に対するフィードバックを顧客から獲得でき,完全かつ正しい非形式仕様を作成することができる.もう一つの可能性は，顧客は SOFL 非形式仕様を書き，それによってアニメーションシステムを作成，開発者に顧客自身の具体的な要求を説明する．この方法は，特に日本で仕様を書いて，海外で設計と実装を行うビジネスモデルに最適である．

3.2 研究目標 2「半形式仕様からアニメーションシステムのアーキテクチャーの構成方法の確立」

3.2.1 当初の想定

(1) 想定する仮説等

SOFL モジュールに定義されたデータ構造，不変数，およびプロセス仕様に基づき，プロセスの振る舞いをアニメーションするシステムの構造を導出する方法を研究する．想定される課題としては，半形式仕様からアニメーションシステムのアーキテクチャーの導出方法にある．

(2) 当初の到達目標

半形式仕様に記述された機能要件を，ユーザに対して関心度の優先順位を付け，関心度が高い機能要件を「インタフェース操作」として定義，それを支える機能要件を「内部操作」として定義して，システムの操作ツリーを作成する．一つのインタフェース操作に対し，入力，出力，および実行中に出される状態を表現できる GUI (Graphical User Interface)を設計する．このように一般的な考え方で半形式仕様からアニメーションシステムのアーキテクチャーの導出を実現する．

(3) 当初の期待される効果

当初の期待される効果は，半形式仕様に定義されているプロセスの仕様が，アニメーションを通じてその仕様の全ての面を,開発者と顧客に素早く確認してもらうことによって，ユーザ要求の定義に大きい間違いを，開発の早い段階で取り除くことができる．

3.2.2 研究プロセスと成果

(1) 研究プロセス

この研究については，次の三つのステップを取った．

① 第 1 ステップ：SOFL 半形式仕様からどのようにアニメーションの対象を選出するかという課題を研究．

この中に，データ構造からアニメーションの GUI を生成する研究も含まれている．

② 第 2 ステップ：アニメーションの対象のどんな側面をどのように動的に表現すべきかを研究．

この研究には，不変条件のアニメーションによる表現の研究も含まれている．

③ 第 3 ステップ：プロセスのアニメーションを実施するためにどのような支援ツールが必要かということを研究．

この研究の中で，プロセスのアニメーションのために必要な基本パターン或いはテンプレートに関する研究も行った．

(2) 具体的な研究成果の内容

研究成果としては，三つである．

(a)半形式仕様からアニメーションの対象とするプロセスの選出方法を確立した．

(b)プロセス仕様アニメーションの方針を明らかにした．

(c)プロセスアニメーションの支援ツールを構築した．

次に，これら成果を，一つずつ詳しく紹介する．

① 半形式仕様からアニメーションの対象とするプロセスの選出方法

半形式仕様は，非形式仕様の表現に存在した曖昧性をある程度解消し，ユーザの要求を明確に定義したものである．構造としては，SOFL モジュールの集合である．一つのモジュールには，システム機能を定義したプロセスが，インタフェース，事前条件と事後条件によって定義されている．インタフェースには，入力変数，出力変数，および状態変数をデータ型で明確に宣言してある．使ったデータ型は，モジュールの型（type）セクションに明確に定義されている．

このような半形式仕様のアニメーションの目的は，定義されたすべてのモジュールの内容は，開発者と顧客が期待する通りに記述しているかどうかを確認することである．プロセスの振る舞いを定義する仕様は，モジュールの中心コンポーネントであるし，そこからモジュールにおいて定義されたすべての他のコンポーネント（例えば，データ型の定義，不変上の定義など）に繋がる．この現状を考えると，半形式仕様アニメーションを実施するために，すべてのモジュールに定義されたすべてのプロセス仕様をアニメーションすべきである．

問題は，どのような方針でアニメーションするプロセス仕様を選出するか．これに関しては，次の手順を取る．

第 1 ステップ：顧客と開発者は，関心が高い機能についてのモジュールを優先してアニメーションの対象として選択する．

第 2 ステップ : 選択されたモジュールに定義されたプロセス仕様を，「開始プロセス」，「中間プロセス」，「停止プロセス」の順で，アニメーションの対象として選出する．開始プロセスとは，入力変数の中で，本モジュールの外から流れてきたデータフローを表す変数が含まれているプロセスである . 中間プロセスとは , すべての入力変数と出力変数が，本モジュールに定義されたプロセスから生成されたデータフローを表す変数であるプロセスである．停止プロセスとは，出力変数のなかで，本モジュールの外へ流れていくデータフローを表す変数が含まれているプロセスである．このような順でプロセスのアニメーションを実施すると，開発者と顧客がそのモジュールの機能を理解しやすくなるという効果があると考えている．

時間の制限によって全てのプロセス仕様のアニメーションの実施が不可能場合は，開発者または顧客によって関心度が高いプロセス仕様を優先して選出する方法を採用することが可能である．

第 3 ステップ : 選出されたプロセスの振る舞いをアニメーションするためには，そのプロセス仕様に関わるデータ構造から適切の GUI を作成する．その GUI は，基本的に SOFL 言語の CDFD に使われているプロセスの図表現になる．

② プロセス仕様アニメーションの方針

アニメーションの対象として選出された一つのプロセス仕様に対して，次に挙げるすべての側面をアニメーションの実施によってチェックする．

(a) すべての入力変数とそれを宣言するデータ型

(b) すべての出力変数とそれを宣言するデータ型

(c) 本プロセスに繋がるすべてのデータストアを表す状態変数とそれを宣言するデータ型

(d) 本プロセスに関係がある不変条件

(e) 事前条件

(f) 事後条件

これらコンポーネントを動的に表現するためには，プログラムテストの基本アイディアを利用する．すなわち，プロセスの振る舞いを適切に表現する具体的な値をプロセスの入力変数とそれを対応する出力変数および状態変数に生成し，入力データフローを動かしながら対応する出力データフローを作るという動的な表現を実施する．生成された入力と出力の具体的な値は ,「テストケース表」というデータ構造に入れておいて , プロセス仕様のアニメーション時に，入力と出力間の関係および入力と出力に破られた不変条件の状況を示すために使われる . このテストケース表の構造と中身は , 図 3-2-1 に示した通りである .

a	b	Input (IN)	c	d	Invariants (INV)	a	b	Output (OT)	c	d
variable	Data type	Sample Data	Data Store		inv_1 inv_2 e inv_n	variable	Data type	Sample Data	Data Store	
vIN_1	tIN_1	sIN_1	$idIN_1$ or $xdIN_1$			vOT_1	tOT_1	sOT_1	$idOT_1$ or $xdOT_1$	
vIN_2	tIN_2	sIN_2	$idIN_2$ or $xdIN_2$			vOT_2	tOT_2	sOT_2	$idOT_2$ or $xdOT_2$	
…	…	…	…			…	…	…	…	
vIN_n	tIN_n	sIN_n	$idIN_n$ or $xdIN_n$			vOT_n	tOT_n	sOT_n	$idOT_n$ or $xdOT_n$	

図3-2-1　半形式プロセス仕様のアニメーション用のテストケース

プロセス仕様のアニメーションを，開発者と顧客に容易に理解してもらうために，システムのアーキテクチャーを表現する CDFD に使われる単一プロセスの図表現を利用して，入力を入れて，出力を出すというアニメーションを表すことになる．この表現は，図 3-2-2 に示した通りである．

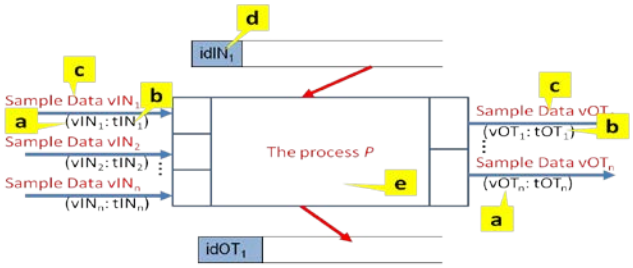


図3-2-2　プロセスの半形式仕様アニメーションの図表現

さらに，入力データと出力データの構造を，適切な図表現を使用すれば，アニメーションの効果が改善される可能性が高い．どのような図表現を使用するかは，応用領域によって選択が違うかもしれないが，抽象レベルで統一した表現方式を設計することが可能である．

③ プロセスアニメーションの支援ツール

半形式プロセス仕様アニメーションの支援ツールは，Microsoft の PowerPoint ソフトウェアに必要なアニメーションコンポーネントと機能テンプレートを追加して構築した．図 3-2-3 に示したように，半形式仕様アニメーションリボンタブをクリックすると，アニメーション用のすべてのコンポーネントと機能テンプレートをディスプレイする．

図3-2-3　PowerPointに追加されたコンポーネントとテンプレート

これらコンポーネントは，二つのグループに分類している．一つのグループは，単一のプロセス仕様アニメーションを実施するために必要なコンポーネントを含める．もう一つのグループは，データ項目をアニメーションするために必要なコンポーネントを含める．この二つのグループのコンポーネントの詳細は，表 3-2-1 に簡潔に説明している．

表3-2-1　半形式仕様アニメーション用のコンポーネントの分類

グループ	コンポーネント（ボタン）	特徴, 役割 および事例
① 単一プロセス仕様アニメーション	• Drop down list for INPUT • Drop down list for OUTPUT • Get the process button	特徴 • User needs to select combination value both for INPUT and OUTPUT. • The selected number is according to the number of input and output for the single process that need to be animated • By clicking the Get the Process button will insert a slide that contains the single process animation template with appropriate input and output for the process. • In the template we also provide a test case table that shows all related items and their relation that can be found in the single process. 役割 • Animate the process described in the semi-formal specification. The animation is emphasized on five main items: a. Input/output variable, b. Input/output data type c. Input/output sample data d. Process invariant e. Data resources • Those items are displayed in two forms (both are animated) a. Test case table The test case table is used to show the possible input variable combination including the possible output produced from the process. b. Diagrammatic way Mimicking the input output process, suitable image can be used to make the animation more meaningful.

		事例 • 図 3-2-4 に示した通り.
② データ型のアニメーション		特徴 • Basic user interface components that always found in any kind of system user interface. • The components can be resized according to user requirement • The components are static not provided with any animation 役割 • The animation used to visualize the data type behavior. • Purposely to simplify the explanation process for those who are not familiar with the formal data type definition either in the semi-formal specification or its animation. 事例 • 図 3-2-5 に示した通り.
	• Numeric • Character • Enumeration • Boolean • Set • Sequence • Composite • Product • Map • Union	

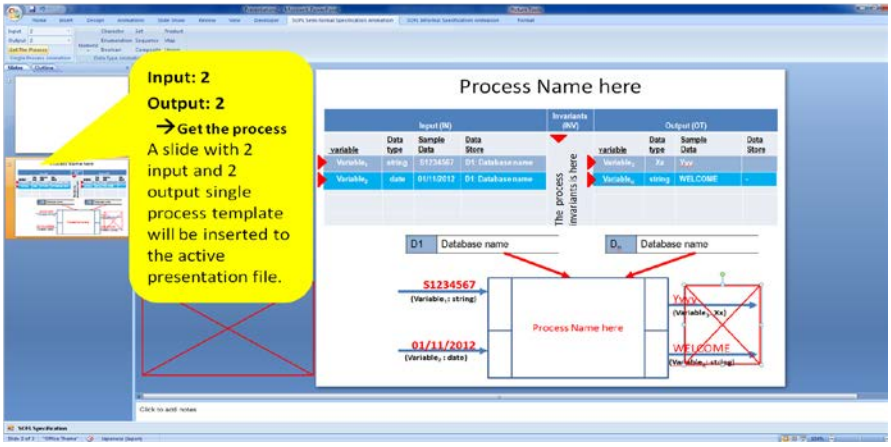


図3-2-4　単一プロセスアニメーション用のテンプレート

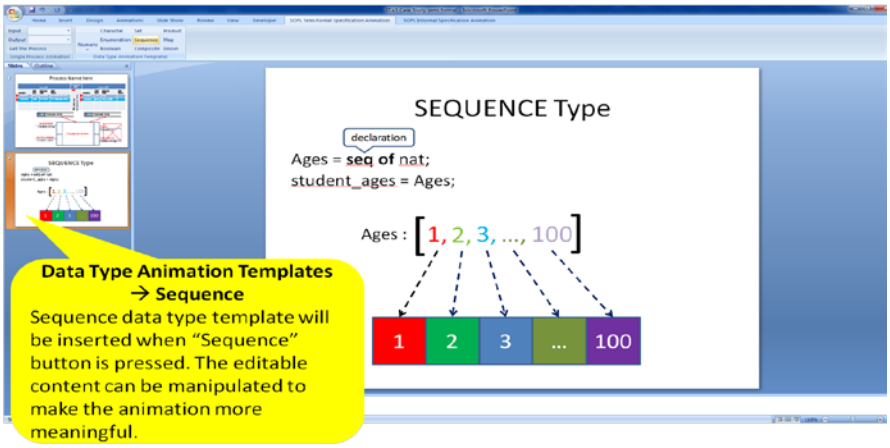


図3-2-5　データ項目のアニメーション用のテンプレート

④ 半形式仕様アニメーションの事例

非形式仕様のアニメーションに使用された同じトラベルエージェントシステム ITAS を事例として使い，半形式仕様のアニメーションの前述した技術について説明する．

図 3-2-6 に半形式仕様に定義されたプロセス CCAuthentication_Charge のアニメーションシステムを示している．このシステムには，テストケース表とそのプロセスの図表現が含まれている．このシステムを，Microsoft の PowerPoint のアニメーション機能を利用してそのプロセスのアニメーションを実施することができる．このアニメーションを実施する内容を表 3-2-2 に表現し，簡潔に説明している．

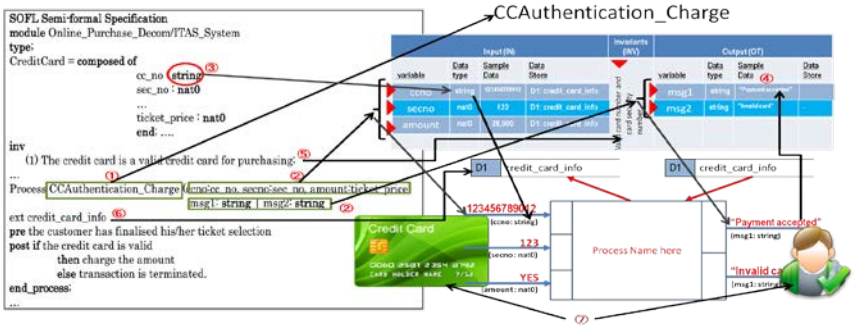


図3-2-6　半形式仕様アニメーションの事例

表3-2-2　プロセスCCAuthentication_Chargeのアニメーション

アニメーションの対象とするプロセスの情報	プロセスアニメーションの内容
① プロセス名 → “CCAuthentication_Charge”	• High light on process name as the header of the animation slide
② 入力 → 三つの入力値: ccno, secno, amount 出力 → 三つの出力値: msg1, msg2	• In the test case table, the input and output are represented in different lines • In the graphical representation, the input is represented by three arrows coming into the process and the output is represent by two arrows exiting from the process.
③ データ型 → cc_no : **string**	• As illustrated in Fig. 3-2-5 in ③
④ テストデータ　→ “Payment accepted”	• The process behavior for this page is searching information based on the given criterion.
⑤ 不変条件 → 有効なクレジットカード	• Demonstrate the card validity violation behavior of the credit card.
⑥ データストア → “credit_card_info”	• “credit_card_info” is used to retrieve and store related information shown by the arrows direction “from” and “into” the process
⑦ イマージュ	• Related images are used to make the animation more lively and understandable.

3.2.3 実用化へ向けた課題と問題点

(1) 課題と問題点

課題は，以下の二つである．

① SOFL 半形式仕様とそのアニメーション手法を，実務者に勉強させる必要がある．SOFL 非形式仕様の場合，このような勉強はあまり必要でなく，一般の実務者はできる．しかしながら，半形式仕様の場合，SOFL 形式仕様記述言語のほとんどの要素が含まれているので，それを学ばなければならない．今までの大学での経験によると，ほとんどの実務者は学ぶことができ，容易に使用することができるので問題ない．

② 支援ツールの機能の改善と使用できるデータ項目を表すアイコンや操作の振る舞いを表す機能テンプレートなどを追加することが必要である．

問題点としては，このような技術が，開発現場でどのような効果があるか，どのような改善すべき点があるかということが未知数である．

(2) 将来の応用方法

将来の応用方法としては，以下の二つが想像できる．

① グローバルソフトウエア開発プロジェクトに適用する．

具体的には，SOFL を用いて半形式仕様を作成し，アニメーションを通じてその整合性と正当性を保証した後，海外の実装者に渡す．こうすると，実装者は半形式仕様を理解するときに，そのアニメーションに参考でき，正確な理解が保証できる．今まで海外発注ソフトウェアプロジェクトの場合，要求仕様を日本で非形式言語（例えば，日本語）により作成され，海外の実装グループに渡してプログラムを作成するパターンが多い．この中で，一番大きい問題点は，海外の実装グループがその要求を正しく理解していない可能性が高い．この問題を解決するためには，SOFL の半形式要求仕様を日本で作成，その上で半形式仕様に定義されている重要なプロセス（操作）の振る舞いのアニメーションシステムを作成，それから海外で実装してもらう．この場合，海外の実装グループは，半形式仕様のアニメーションシステムによって，要求された機能，データ構造および制約を正しく理解する可能性が高いため，実装するプログラムの信頼性も高くなるだろう．

② 要求仕様レビューの道具として使用できる．

具体的には，開発者が SOFL 半形式仕様のアニメーションを作成する．このために，開発者は，半形式仕様に定義されているプロセスのすべての側面を明確に理解しなければならない．この理解によって，半形式仕様で見て分かる間違いをまず取り除く．その後，開発同士にピアレビューしてもらい，更にエラーを発見する．

3.3 研究目標 3「形式仕様パターンの定義と仕様パターンシステムの構造の確立」

3.3.1 当初の想定

(1) 想定する仮説等

形式仕様記述言語を用いて操作の形式仕様を作成することが，産業界の一般の実務者にとって困難であることが現状である．この問題点を解決するために，形式仕様パターンに基づく操作の形式仕様を作成することは有効であろう．本研究では，形式仕様パターンの構造および各要素の役割を定義する．更に，仕様パターンの分類を行い，分類されたパターンのツリー構造を設計する．想定される課題としては，形式仕様パターンの分類方法とパターンの完全性の保証方法である．

(2) 当初の到達目標

プログラムの仕様で定義できる機能は，「関係表現」，「情報検索」，および「情報更新」の三種類に分けられ，各種類の機能を表現する様々な仕様パターンを作成する．更に，そ

れらのパターンを効率的に適用するために，ツリー構造で管理する．また，仕様パターンの完全性に対しては，実験と自習によって仕様パターンシステムを徐々に完成する．

(3) 当初の期待される効果

期待される効果は，以下の二つである．

① 形式仕様パターンのより合理的な定義と形式化は,仕様パターンに基づく形式仕様作成の支援ツールの基礎として使える効果がある．
② 形式仕様パターンの適切な分類システムは,支援ツールに知識として入れる仕様パターンの完全性を確保するために検証できる効果がある．

この研究結果は，本技術の使用者に対して直接な効果がないが，本技術を実現するために効果がある．

3.3.2 研究プロセスと成果

(1) 研究プロセス

この研究については，次の三つのステップを取った．

① 第 1 ステップ：本研究以前に検討した形式仕様パターンの定義をもう一度研究し，足りない部分を追加し，より合理的かつ形式的な定義を達成した．
② 第 2 ステップ:形式仕様で表現できる操作機能を分析，その上で形式仕様パターンを，同じ目的を持つグループに分類する方法を研究，分類した仕様パターンを表すツリー構造を設計した．
③ 第 3 ステップ：パターンの完全性についての実験と自律学習の仕組みを研究した．

(2) 具体的な研究成果の内容

研究成果としては，二つである．
(a)形式仕様パターンの合理的かつ形式的な定義を明らかにした．
(b)分類された仕様パターンを表す「仕様パターンシステム構造」を設計した．
次に，この二つの点に関して具体的に説明する．

① 形式仕様パターンの合理的かつ形式的な定義

前述したように，形式仕様パターンとは，形式仕様を作成するテンプレートである．一つのシステムの形式仕様には，仕様のアーキテクチャー，データの宣言，操作仕様，および関数の定義などの側面がある．本研究は，操作仕様の事前条件と事後条件の論理式を作成するために，既存の経験を反映する形式論理式テンプレートに集中した．このため，これから紹介する形式仕様パターンは，実際に形式論理式を作成する仕様パターンである．

仕様パターンの構造に次の四つの具体的な項目を含める．
(a)パターン名 (name)
(b)パターンの説明 (explanation)

(c)パターンの構成要素（constituents）
(d)パターンの解答（solution）

パターン名は，パターンの名前を示し，パターンが適用されるときに ID として使われる．パターンの説明は，このパターンが表現できる機能および使える状況について説明する．パターンの構成要素は，このパターンによって形式論理式を表現するために必要なコンポーネント要素を規定する．パターンの解答は，形式論理式を形成するために必要な具体的な道筋（テンプレート）を示す．図 3-3-1 に示した仕様パターンは，この構造を説明する事例である．この事例の中で，*dataType*(*x*)は，変数 *x* の型を表示，*constraint*(*T*)は，型 T の値についての性質または制約を表す．

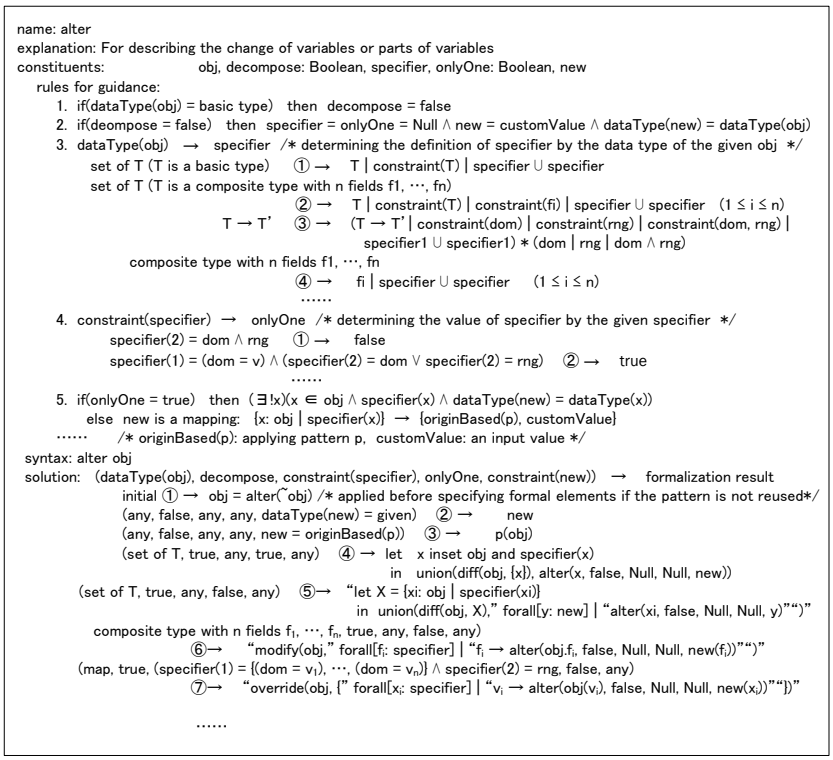


図3-3-1　仕様パターン“alter”の構造

図 3-3-1 に示した事例の仕様パターン名は，“alter”である．パターンの説明は，変数の値の一部または全部の内容を更新するために使えるパターンとして説明している．パターンの構成要素は，五つの要素を挙げていると共に，それら五つの要素を定義するときに必要なガイダンスも提供している．パターンの解答は，最終的に形成できる形式論理式の構造と要素を示している．その要素のなかで，ある要素は，他の仕様パターンを適用することによって得られる．

このように定義した仕様パターンは，コンピュータに理解させるためには，知識としてコンピュータに保存しなければならない．コンピュータは，そのパターン知識を理解した

上で，開発者に適切なガイダンスを提供しながら最終的な形式論理式を形成する．これを実現するために，仕様パターンの定義を数学によって明確に定義することが必要である．このため，本研究では，前述した仕様パターンの非形式的定義を踏まえ，次の形式的定義を提案した．

定義２ 仕様パターンＰは，6 項組（id, $expl$, E, PR, Φ, ψ）として定義される．ただし，

- id はパターンの唯一のアイデンティティ．
- $expl$ は仕様パターンＰが表現できる機能の説明．
- E は形式論理式に必要な要素の集合．
- PR は制約の集合．
- Φ： $\Phi_E \cup \Phi_R$は，E に含まれている要素を定義するルールの集合．ただし，
  - Φ_E ：$E \to E$ は下記の条件を満たす要素の定義する順番を決める関数である．
    - $\exists_{e_0 \in E} \bullet e_0 \notin ran(\Phi_E)$ (e_0は最初に定義するべき要素)
    - $\forall_{e \to e\prime \in \Phi_E} \bullet e \neq e'$ (e → e’ は要素 e’ が要素 e の後定義すべきという意味を表す)
  - Φ_R ：E → ℘(R)は E に含まれている要素が定義された後適用すべきルールを表す関数である．ただし，
    - R：$\wp(\mathrm{PR}) \to \wp(\mathrm{PR})$ は条件$\forall_{PR_i \to PR'_i \in R} \bullet PR_i \cap PR'_i = \varnothing$を満たす新しい制約を導出するルールの集合を表す．
    - $\forall_{e \to R\prime \in \Phi_R} \bullet \forall_{PR_i \to PR\prime_i, PR_j \to PR\prime_j \in R\prime} \bullet \forall_{pr \in PR_i} \bullet pr \Rightarrow \exists_{pr' \in PR_j} \bullet \neg pr$

  (ルール集合Φ_Rのなかのルールは，毎回一つだけ適用される)
- ψ：$\wp(\mathrm{PR}) \to string$ は，$\forall_{PR_i \to s_i, PR_j \to s_j \in \psi} \bullet \forall_{pr \in PR_i} \bullet pr \Rightarrow \exists_{pr\prime \in PR_j} \bullet \neg pr'$を満たす形式論理式を生成するルールの集合．

例えば，図 3-3-2 に前述した仕様パターンの形式的定義が示されている．

id	alter	
expl	For describing the change of variables or parts of variables	
E	{*obj*, *decompose*, *specifier*, *onlyOne*, *new*}	
PR	{reuse, *dataType(obj)* = *char*, decompose = false, dataType(obj) = set, specifier(1) = {(dom = v_1), …, (dom = v_n)} ∧ specifier(2) = rng, onlyOne = true, dataType(new) = given, ...}	
Φ	Φ_E	{*obj* → *decompose*, *decompose* → *specifier*, *specifier* → *onlyOne*, *onlyOne* → *new*}
	Φ_R	{obj → {{dataType(objs) = char □ dataType(objs) = nat □ … } → {decompose = false}, ...}, {*decompose* → {{decompose = false} → { specifier = onlyOne = Null ∧ new = customValue ∧ dataType(new) = dataType(obj)}, ...}, ...}
Ψ	{{dataType(obj) = set, decompose = true, onlyOne = false} → "**let** X = {xi: obj | specifier(xi)} **in union**(**diff**(obj, X)," forall[y: new] | "alter(xi, false, Null, Null, y)"")", …}	

図3-3-2　仕様パターン“alter”の形式的定義

② 分類された仕様パターンを表す仕様パターンシステム構造

すべての可能な仕様パターンが効率的に管理，検索，適用されるためには，表現の機能によって適切に分類される．この分類は，抽象的な機能から具体的な機能まで階層的に表現されている．抽象的に見ると，仕様パターンは，基本的に次の三種類に分けている．(a)関係パターン（Relation pattern），(b)検索パターン（Retrieve pattern），および(c)再造パターン（Recreate pattern）．一つの抽象パターンから，さらに分解し，より具体的な特徴を持つ仕様パターンが含まれる．図 3-3-3 に示した通り，すべての仕様パターンは，このように階層的な構造でパターンシステムを構成する．

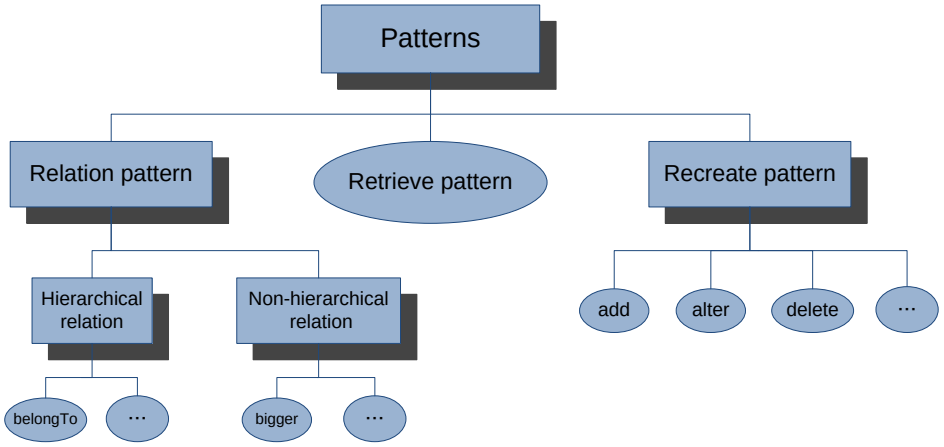


図3-3-3　形式仕様パターン分類システム

関係パターンは，オブジェクト間のある関係を表現するために必要なテンプレートを提

供している.関係パターンは,階層的関係パターンと無階層関係パターンに分けっている.階層的関係パターンは，階層的なオブジェクト間の関係を表現するために使えるパターンである．例えば，仕様パターン“belong To”は，一つの要素は一つの集合のようなオブジェクトに所属する関係を表現するパターンである．無階層関係パターンは，同じレベルのオブジェクト間の関係を表すために使用できるパターンである．例えば，仕様パターン“bigger”は，同じレベルの整数間の関係を表すパターンである．

検索パターンは，基本的に複合データオブジェクトに所属するデータ項目の参照を表現するパターンである．このような表現は，いくつかの複合オブジェクトを使うことが可能である．

再造パターンは，変数が表すオブジェクトの内容を更新または生成する機能を表現するために使えるパターンである．例えば，パターン“add”は，新しいデータ項目を追加する機能を表すパターンである．

仕様パターンを応用するために，図 3-3-3 に示した仕様パターン分類システムを参考する上で，次の手順を取る．

1)適切な仕様パターンを検索する．

開発者が形式論理式を作成する際，まずコンピュータに自分の要求を自然言語で伝える．これを理解したコンピュータは，仕様パターン分類システムにおいて適切なパターンを検索する．このような検索は，その仕様パターン分類システムの階層的な構造のトップノードから始め，形成したい論理式の性質を考えながらその検索を完成する．適切な仕様パターンが見つけられたら，その仕様を選択する．

2)仕様パターンを適用する．

この場合，選択した仕様パターンのすべての構成要素を提供することによって形式論理式が最終的に形成される．このプロセスは，開発者とコンピュータ間の対話によって徐々に完成する．図 3-3-4 に，ATM（Automated Teller Machine）ソフトウェアの引き出す（“withdraw”）機能の中に，銀行口座の残高を更新する形式論理式を形成する仕様パターン“alter”の適用プロセスを示し,図 3-3-5 に最終に形成された形式論理式が示されている．

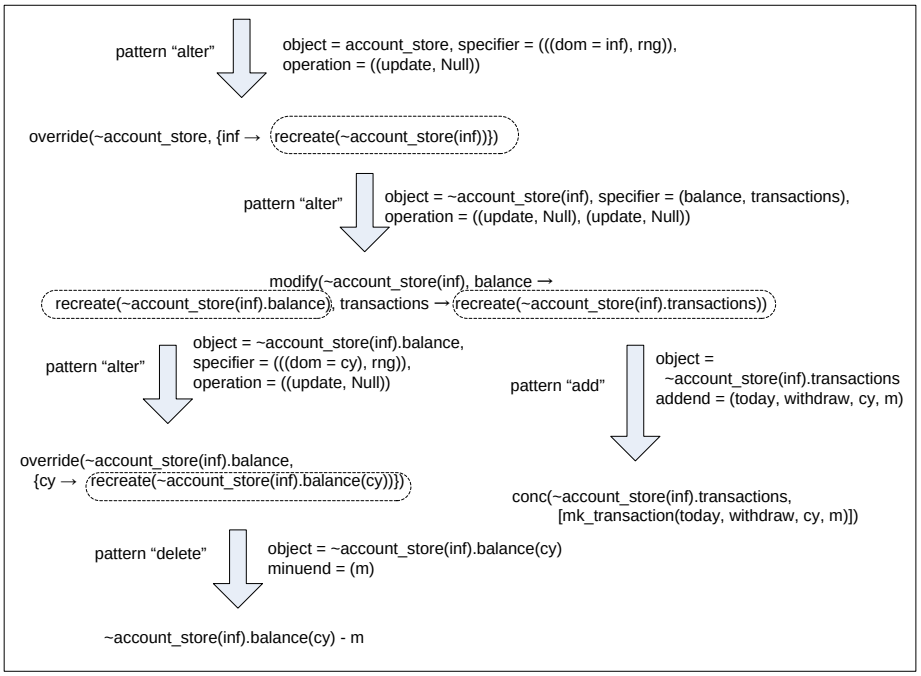


図3-3-4　仕様パターン“alter”の適用プロセス

$$
\begin{aligned}
&override(\tilde{}account\_store,\\
&\quad \{inf \rightarrow modify(\tilde{}account\_store(inf),\\
&\qquad balance \rightarrow override(\tilde{}account\_store(inf).balance,\\
&\qquad\qquad \{cy \rightarrow \tilde{}account\_store(inf).balance(cy)\ \text{-}\ m\}),\\
&\qquad transactions \rightarrow conc(\tilde{}account\_store(inf).transactions,\\
&\qquad\qquad [mk\_transaction(today, withdraw, cy, m)])\})\})
\end{aligned}
$$

図3-3-5　最終に形成された形式論理式

3)パターンの完全性についての実験と自律学習の仕組み

仕様パターンの完全性に関しては，二つのレベルの定義を提案した．一つは，SOFL 形式仕様記述言語に設定された全てのデータ型と演算子を表現できる仕様パターンシステムは，完全性を持っているシステムである．この定義により，仕様パターンシステムの完全性の検証をチェックリストによる検査に基づいて簡単に実現することができる．もう一つは，全てのアルゴリズムの機能を表現できる仕様パターンシステムは，完全性を持っているシステムである．既存のパターンシステムから必要なパターンを見つけられない時に，ユーザから新しい仕様パターンを作成して仕様パターン知識ベースに入れる．完全に自律学習の仕組みではないが，現実的な仕組みである．

3.3.3 実用化へ向けた課題と問題点

(1) 課題と問題点

課題としては，分類された形式仕様パターンは，全ての可能な機能を表現するために足りるかどうかが決められないことである．これを形式的に証明することができないが，対応策を提案することが可能である．これに関しては，今後の研究で完成する予定である．

(2) 将来の応用方法

将来の応用方法としては，以下の二つが考えられる．

① 系統的に分類された仕様パターンシステムが，VDM や SOFL など形式仕様記述言語を用いてソフトウェア要求仕様或いは設計仕様を作成する実務者が利用可能である．このパターンシステムに参考することによって，形式仕様をより容易に作成することができ，機能要求の分析およびシステムの設計をより効率的かつ効果的に実現することができる．

② 仕様パターンに基づく操作の形式仕様作成アプローチの支援ツールの構築に使える.すなわち，この仕様パターン分類システムは，その支援ツールを構築する基礎として使われる．

3.4 研究目標 4「形式仕様パターン知識の表現，検索，および適用仕組みを提案する」

3.4.1 当初の想定

(1) 想定する仮説等

形式仕様パターンに基づく操作の形式仕様作成アプローチを実現するためには，仕様パターン知識を，コンピュータにおいて適切に表現することが重要である．この上で，仕様パターン知識の効率的な検索および検索により得た仕様パターンの効果的な応用方法を明らかにしなければならない．

(2) 当初の到達目標

仕様パターン知識の目的は，コンピュータアルゴリズムが開発者に適切なガイダンスを提供するために参考するものと考慮した上で，三つの目標を設定した．

① 仕様パターン知識を表示する状態遷移図を定義する．

② 仕様パターン知識の表現を支援ツールに実装する XML 形式の設計とそれにより仕様パターンを検索・適用するアルゴリズムを設計する．

③ 検索で得た仕様パターンによって開発者へガイダンスを提供する方法を確立する．

(3) 当初の期待される効果

ソフトウェア開発者にとっては，直接に期待される効果がないが，仕様パターンに基づく形式仕様作成アプローチの支援ツールの開発に，次の二つの効果がある．

① 仕様パターン知識をコンピュータにおいて適切に表現，検索，および応用することがで

きる.
② 仕様パターンによって開発者へ適切なガイダンスを提供することができる.
開発された支援ツールによって,形式仕様をより効率的かつ容易に作成することができる.

3.4.2 研究プロセスと成果

(1) 研究プロセス

この研究については，次の三つのステップを取った.
①第 1 ステップ：仕様パターン知識の有限状態遷移図表現を研究．具体的には，状態は何を表す，状態遷移はどんな効果を出す，また，仕様パターン間の利用方法をどのように状態遷移図で表現するかということを研究.
②第 2 ステップ：有限状態遷移図で表現した仕様パターン知識の中から，必要な知識を検索するアルゴリズムを研究.
③第 3 ステップ：開発者へ提供するガイダンスの構造，内容，およびコンピュータ画面での表示方法などを研究.

(2) 具体的な研究成果の内容

研究成果としては，三つである.
① 形式仕様パターン知識を表現する有限状態遷移図の明確な定義をした.
② 仕様パターン知識の表現を支援ツールに実装する XML 形式を設計した.
③ XML 形式により仕様パターンを検索・適用するアルゴリズムを設計した.　次に，これら三つの進展について詳細に説明する.

① 形式仕様パターン知識を表現する有限状態遷移図の明確な定義
仕様パターン知識は，形式論理式を作成するプロセスの中で，支援ツールが次の取るべき行動についてユーザへ適切なガイダンスを提供するために使われる．ユーザからの入力はイベントとして扱われ，それによって支援ツールが次の状態に移して，その状態からさらに発展するために必要な入力をユーザに求める．このようにユーザと支援ツールの対話をしながら，最終的に形式仕様が作成される.
このような対話的な支援ツールを構築するために，すべての仕様パターン知識を，有限状態遷移図によりまとめて表示する．このような有限状態遷移図（FSM）は，次の様に定義されている.

定義２　有限状態遷移図 A は，以下の条件を満たす９項組$(Q, q0, F, R, I, G, \varphi, \delta, \lambda)$として定義される．ただし,

(a) Q *は有限の状態の集合,*
(b) $q0 \in Q$ *は*Q*に所属する開始状態,*
(c) $F \subset Q$*は*Q *の部分集合であり，終了状態の集合,*
(d) R *は３項組*(V, V', θ)*として定義される．ただし,*

(d1) V *はシステム変数の集合,*

(d2) $\mathbb{V}'$ *は値の集合,*

(d3) $\theta: \mathbb{V} \to \mathbb{V}'$ *は変数* $v \in \mathbb{V}$ *の値を求める関数*

(e) $\mathbb{I}$ *は符号の集合,*

(f) $\mathbb{G}$ *はガード条件の集合,*

(g) $\varphi: Q \to R$*は状態に使われる変数の値を求める関数,*

(h) $\delta: Q \times (I \times \wp(G)) \to Q$*はAが次に行うべき1ステップの動作を指定する状態遷移関数,*

(i) $\lambda: Q \times (I \times \wp(G)) \to I$*は現時点の状態とユーザの入力によって次のユーザへ提供する出力を求める出力関数である.*

このように定義された有限状態遷移図Aには，Iの要素がユーザへ提供するガイダンスまたは必要な返事であり，G の要素が制約を表す．例えば，Aは，条件(i，G) $\in$ $Acc_A(s)$が成立する状態s に立っている場合，iをユーザへ提供することによって，ユーザに求める新たな入力$\lambda(s,(i,G))$が生成され，条件g $\in$ Gが満たすなら状態$\delta(s,(i,G))$に到達する．

図3-4-1にAを命名した簡単な状態遷移図を示す.この図の中に使われた符号の文法と意味を次の通りである．ΣCは，ユーザへ提供するすべてのガイダンスまたはデータ項目を表す．&choice/output（例えば，&c/d）はchoice（例えば，c)がすでに選択された事実を表し，output（例えば，d）を出力（例えば，ユーザへ提供したガイダンスまたはデータ項目）として生成することを示す．*req(v)* は変数*v* に具体的な値を提供する要請を表す．状態に付けている等式（例えば，状態Sに付けているvar1 = v1，var2 = v2）は，その状態の中身を示す.

状態遷移図Aは，四つの状態が入っている状態集合$Q_A = \{s, s', s'', s'''\}$ を持っている.Aは，開始状態Sから始め，もしユーザからの入力aを受け取ったら，次の状態Ｓ’に移すと共に，変数var2に新しい値を提供する要請をユーザに出力する．前の状態と比べて，状態Ｓ’では変数var1のみをv1 + aに更新し，var2の値を維持している．もしユーザがbを入力し，変数var1が条件var1 > 5を満たすなら，次の状態*s’’*に移すことに伴い，Ｃに保存されているすべてのガイダンスとデータ項目をユーザに出力する.状態*s’’*では，変数var2が値bに更新され，var1の値が変わらないである．もしｃが入力されたら， 終了状態*s’’’*に移す.同時に，項目*d*を出力する．この状態遷移図の実行が終了するときに，値*var1 - var2*をＡの出力として提供する．

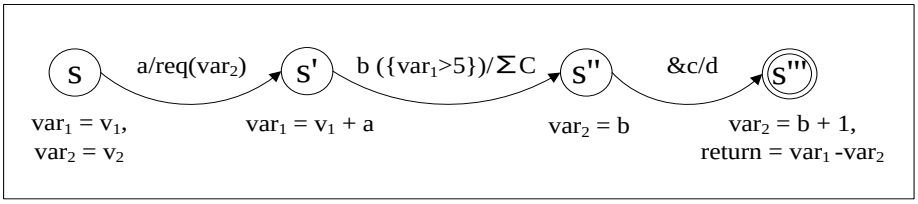


図3-4-1　FSMの事例

一般的には，一つの仕様パターンを適用するときに，他の仕様パターンを使う可能性がある．このようなパターンの引用使用を有効に表現するために，階層有限状態遷移図(HFSM) (Hierarchical FSM) を導入することが必要である．

定義２　階層的な有限状態遷移図Ａ*は，２項組 (F, □)* として定義される．ここで，*F* は，有限状態遷移図の集合，□ は *F* に定義された構成関数である．この構成関数が，次の三種類の構成関係を反映する．

(a)低いレベル状態遷移図は，高いレベルのある状態と解釈する．

(b)低いレベル状態遷移図は，高いレベルの出力をする値を提供する．

(c)低いレベル状態遷移図は，高いレベルの状態に使われている変数に代入する値を提供する．

図 3-4-2 にこれら三つの構成関係を説明する事例を示している．図 3-4-2H_1において，FSM A_1 とその中の状態 u' を解釈する状態遷移図$\square(u')$ が含まれている．図 3-4-2H_2において，FSM A_2 とその開始状態 s の出力する値を提供する状態遷移図 A_3 が含まれている．図 3-4-2H_3において，FSM A_4 と終了状態 w5 に使われている変数 v に代入する値を定義する FSM A_5，A_6，A_7が含まれている．

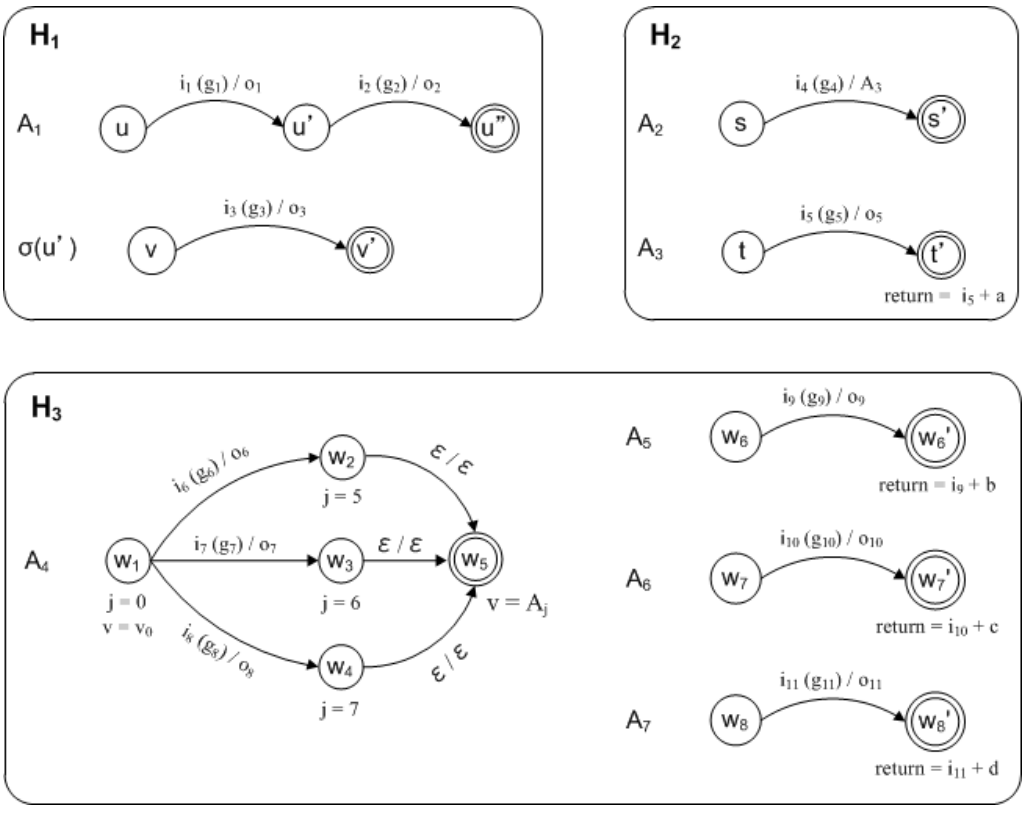


図3-4-2　各構成関係を示す階層有限状態遷移図の事例

上述した定義により仕様パターン知識は，階層有限状態遷移図（HFSM）が作られる．図 3-4-3 に示した前述した仕様パターン“alter”の適用を表現する HFSM のトップ FSM のように，トップレベルの FSM は，三つのパス集合α，β およびγ から構成される．ここで，α の要素は，関連する仕様パターンの構成要素の定義方法を規定する．βの要素は，形式論理式を生成するプロセスを説明する.γの要素は,間違い入力の発見と訂正方法を定義する.

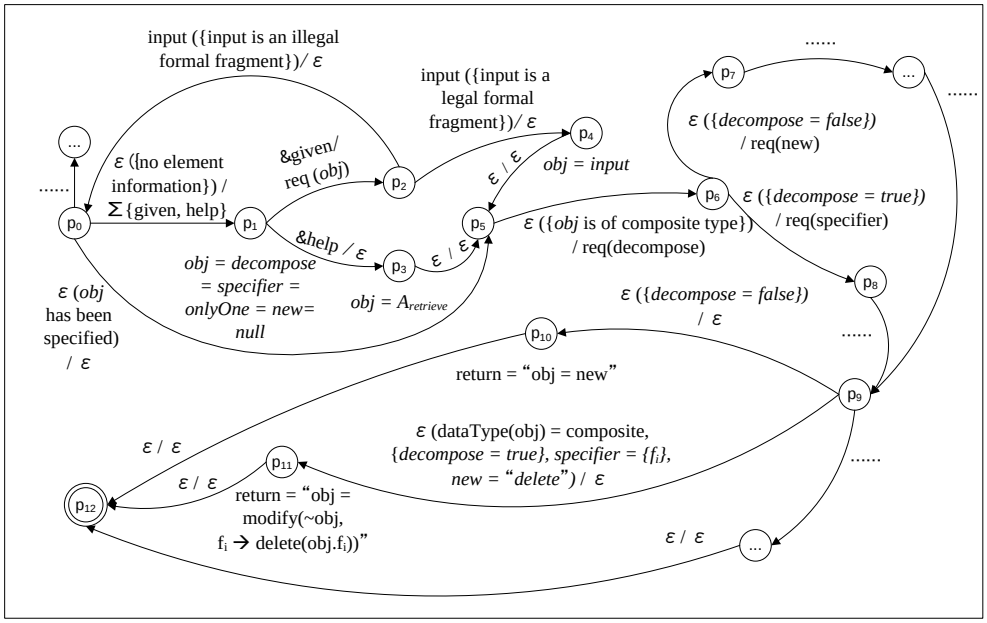


図3-4-3　仕様パターン“alter”の適用を表現するHFSM のトップFSMの事例

② 仕様パターン知識表現の支援ツールに実装する XML 形式の設計

この節では，まず有限状態遷移図で表す仕様パターン知識を支援ツールに実装する XML ファイル形式を紹介する．次に，その XML ファイルに基づき仕様パターンを検索および適用するアルゴリズムを説明する.

1)仕様パターン知識の表現を支援ツールに実装する XML 形式の設計

前述のように，仕様パターン知識は，階層有限状態遷移図 HFSM の形で表現されるが，実際に支援ツールにその表現がどのように実装するかは，解決しなければならない問題である．本研究では，その状態遷移図を，XML フォーマットでの保存方法を提案，支援ツールに実装する方法を実現した.表 3-4-1 に HFSM のすべてのコンポーネントの中身を表示するために使われるタグを示す.

表3-4-1　HFSMのコンポーネントの表示に使われるタグ

XML タグ	役割
<FSM>	Identifying FSMs included in a HFSM
<state>	Identifying states in a HFSM
<transition>	Identifying transitions originating from certain states
<input>	Identifying inputs in transition labels
<guard>	Identifying guard conditions in transition labels
<output>	Identifying outputs in transition labels
<dest>	Identifying destination states of transitions
<inf>	Identifying variable information on states
<name>	Identifying names of elements, system variables or low-level FSMs
<explain>	Identifying explanation on elements written in natural language
<value>	Identifying values of elements or system variables
<type>	Identifying types of elements or system variables
<def>	Identifying definitions of elements
<nestedFSM>	Identifying information of low-level FSMs
<forall>	Identifying formal expressions generated by all the members of a set
<func>	Identifying pre-defined functions for calculating specific values
<operator>	Identifying names of pre-defined functions
<para>	Identifying parameters of pre-defined functions or element information of low-level FSMs

これら XML タグを活用することによって，HFSM の各レベルの FSM を適切に表示することができる．図 3-4-4 に単一 FSM，分解によって得た低いレベル FSM，および FSM に含まれる状態と変数の値の情報の事例を示す．

<table>
<tr>
<td>

```
<FSM name =""">
  <state type="initial">
    <name></name>
    <inf> </inf>
    <transition>
      <input></input>
      <output></output>
      <dest></dest>
    </transition>
  </state>
  <state type="accept">
    <name></name>
  </state>
</FSM>
```

</td>
<td>

```
<nestedFSM>
    <name></name>
    <para type =""">
      <name></name>
      <value>
        <para></para>
        <type></type>
        <value></value>
      </value>
    </para>
    <para>...</para>
</nestedFSM>
```

</td>
<td>

```
<inf type="">
  <para></para>
  <type></type>
  <explain></explain>
  <value></value>
</inf>
```

```
<value>
  <nestedFSM></nestedFSM>
  <func></func>
  <forall></forall>
</value>
```

</td>
</tr>
<tr>
<td>an individual FSM</td>
<td>a low-level FSM</td>
<td>variable information on states and value information</td>
</tr>
</table>

図3-4-4　HFSMのXMLフォーマットの事例

支援ツールに使われるすべての仕様パターンの情報（状態遷移図，状態に付けている変数と値，状態遷移に伴う必要な入力と出力など）を，XML ファイルに保存する．例えば，前述した仕様パターン“alter”の XML ファイルは図 3-4-5 に示した通りである．

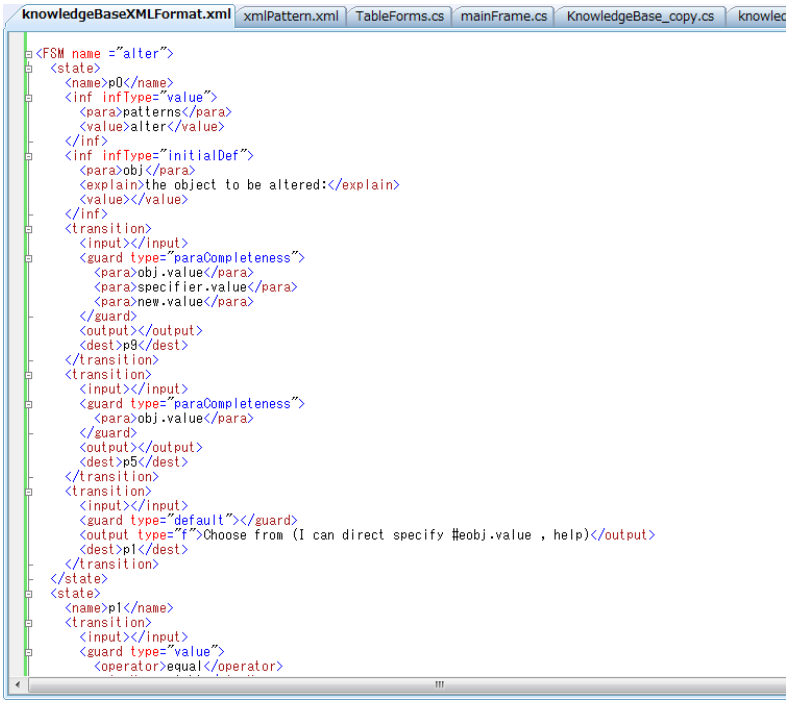


図3-4-5　仕様パターン“alter”のXMLファイルのスナップショット

③ XML 形式により仕様パターンを検索・応用するアルゴリズム

XML 形式の仕様パターン知識を検索および応用するアルゴリズムは，三つの機能を持っている．次に，それら三つの機能を一つずつ簡潔に説明する．

1)タグにより XML ファイルから必要な情報を検索する.

この検索のために，情報を検索する必要なメソッドを持つクラスを設計した．図 3-4-6 にそのようなメソッドを示す.

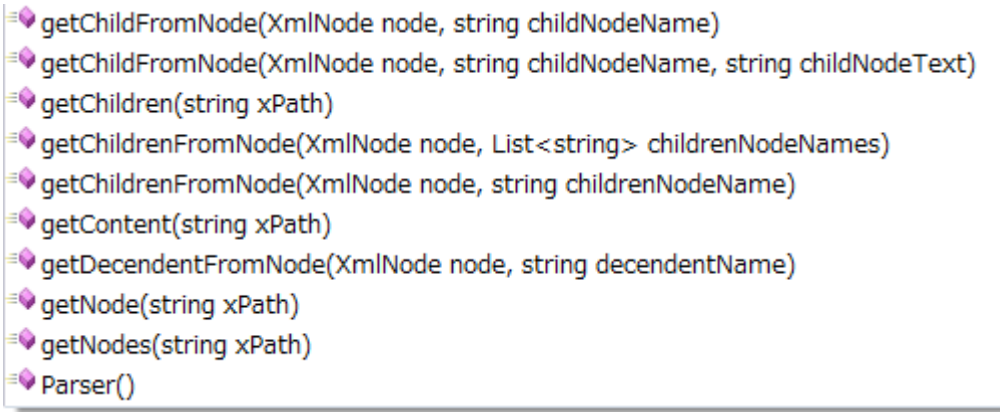


図3-4-6　XMLファイルの情報を検索するメソッドの事例

2)検索によって得た情報に含まれている符号を分析し，適切なデータ構造へ変換する.

HFSM の意味を表現するために，必要な符号が状態遷移と変数に使われる．HFSM において状態遷移を実施するときに，それら符号が適切に分析しなければならない．この分析によってそれら符号の意味を理解し，ユーザへ提供するガイダンスに使われる．図 3-4-7 にモジュールmと currentObj のコンテキストの中で文字列 str から必要な表現を検索するアルゴリズムを示す.

```
String generateObjFromStr(String str, Module m, object currentObj){
    if(str is empty)
        return currentObj;
    else{
        if(currentObj == null){
            i = 0;
            while(str[0, i] is not an element name){
                i++;
            }
            set e as the element named str[0, i];
            return generateObjFromStr(str[i+2, str.length], m, e);
        }
        else if(currentObj is an element){
            preStr = str;
            if str contains "."
                set preStr as the string before the first "." of str;
            if(preStr == "value"){
                set v as the value of the element currentObj;
                return generateObjFromStr(str[preStr.length, str.length], m, v);
            }
            else if(preStr == "name"){
                set n as the name of the element currentObj;
                return generateObjFromStr(str[preStr.length, str.length], m, n);
            }
            else if(preStr == "type"){…}
            …  // other possible values of preStr
        }
        else if(currentObj is a type){…}
        else if(currentObj is a variable){…}
        … // other situations
    }
}
```

図3-4-7　情報を検索する再帰アルゴリズム

3)状態によって支援ツールの次の振る舞いを決める.

仕様パターン知識を利用して，現在の状態と関連する状態遷移に付けている情報に基づき，支援ツールの次の振る舞いを決まる．これは仕様パターン知識を応用するプロセスの中の基本タスクである．図 3-4-7 に選択された仕様パターンを応用するアルゴリズムを示す.この中で,A_{select} は選択された仕様パターンの応用を記述する HFSM を表す(図 3-4-8).

Algorithm Utilization of Pattern knowledge represented in HFSM

```
cf = A_select; cs = q0_cf; input = return = null;
while(cs ∉ F_cf || states is not empty){
  set variable input as the input from the user;
  if(cs ∉ F_cf){
    ns = o = null;
    foreach (i, G) in Acc_cf(cs){
      if(i == input && ∀ g ∈ G · g){
        ns = δ_cf(cs, (i, G)); o = λ_cf(cs,(i, G));}
    if(σ_HF(cs) ≠ ∅ || ∃v ∈ Vφ_cf(cs) · σ_HF(v)≠ ∅ || σ_HF(o)≠ ∅){
      states.push(ns);
      if(σ_HF(cs) ≠ ∅)
        set ns as the initial state of one of the FSMs in σ_HF(cs);
      if(∃v ∈ Vφ_cf(cs) · σ_HF(v)≠ ∅){
        for each v ∈ Vφ_cf(cs) that satisfies σ(v)≠ ∅{queue.push(v);}
        if(σ_HF(cs) = ∅){set ns as the initial state of
                  one of the FSMs in σ_HF(v) where v ∈ Vφ_cf(cs);}
      else{specify variables according to φ_cf(cs)}
      if(σ_HF(o)≠ ∅){
        queue.push(o);
        if(σ_HF(cs) = ∅ && σ_HF(o)= ∅){
          set ns as the initial state of a FSM in σ_HF(o);}}
      else{display o to the user;}
  else{
    if(return ≠ null ){
      replace the corresponding part in queue[top] with return;
      if(σ_HF(queue[top]) = ∅){
        temp = queue.pop();
        if(temp is output){display temp;}
        else{specify variables based on temp;}}}
  ns = states.pop();}
cs = ns; cf = A where cs ∈ Q_A;}
```

図3-4-8　HFSMにおいて表現された仕様パターンを応用するアルゴリズム

このアルゴリズムは，A_{select}で表すHFSMのトップレベルFSMの開始状態から始め，ユーザの入力によって状態遷移を実施しながらユーザへ必要なガイダンスを提供する．そのガイダンスによってユーザからの新たな入力を受け取って，次の状態遷移を決める．必要であれば,高いレベルのFSMから低いレベルのFSM中に状態遷移を展開していくこともある．終了状態に到達するまでこのような状態遷移を実行続ける．終了状態に到達したら，期待される形式論理式が生成される．

3.4.3 実用化へ向けた課題と問題点

(1) 課題と問題点

現時点は仕様パターン知識の表現方法，検索と適応アルゴリズムまでできていて，実用化するためには以下の課題が残っている．

① 大規模有限状態遷移図の表現方法．この課題を解決するために，どのような表現方法を採用すると，仕様パターン知識の検索と開発者へのガイダンスの提供が，効率的に行えるかの問題点がある．

② 有限状態遷移図の更新，拡張，および削除方法．この課題を解決するために，それらの操作を実施するによってできあがる新たな有限状態遷移図の整合性をどのように確保できるという問題点がある．

(2) 将来の応用方法

将来の応用方法としては，以下の二つが想像できる．

(a)知的ソフトウェア工学環境を構築するために,一定のソフトウェア開発知識表現式として使用できる．

(b)ソフトウェア開発プロセスを定義する技術として使える．

3.5 研究目標 5「形式仕様パターンシステムを支援するツールの一部の作成」

3.5.1 当初の想定

(1) 想定する仮説等

コンピュータにおいて保存されている仕様パターン知識に基づく操作の形式仕様作成アプローチを有効に支援するために，支援ツールが必要である．このアプローチの実現性を示すことができる支援ツールのプロトタイプを開発することが想定できる．このツールでは，開発者と対話をしながら，操作の形式仕様をコンピュータによって自動的に生成することに集中して支援する．

(2) 当初の到達目標

当初の到達目標としては，基本的にそのアプローチを有効に支援するツールのプロトタイプを開発することである．このようなツールに関しては，次の機能を提供する．

(a)支援ツールを使っている開発者と対話できる GUI を開発する．

(b)開発者と対話しながら,必要な操作の形式仕様を徐々に作成する支援ツールのプロトタイプを開発する．

(3) 当初の期待される効果

期待される効果は，二つである．

(a)形式仕様記述言語を知らない実務者でも，形式仕様を効果的に作成することができる．

(b)形式仕様を作成することにより，開発者は要求に対する理解を深める．

3.5.2 研究プロセスと成果

(1) 研究プロセス

この研究については，次の三つのステップを取った．

① 第 1 ステップ：支援ツールの全体を設計した．

② 第 2 ステップ：その後，C#で支援ツールを実装した．

③ 第 3 ステップ：実装した支援ツールを事例研究によってテストし，改善した．

(2) 具体的な研究成果の内容

研究成果としては，二つである．

(a)開発者と分かりやすい形で対話できる GUI を開発した．

(b)仕様パターンに基づき形式仕様を生成する支援ツールのプロトタイプを開発した．

次に，これら成果を詳細に説明する．

① 開発者と分かりやすい形で対話できる GUI

ユーザと支援ツールの対話を効率的に支援するために,分かりやすい GUI が必要である．一番実現しやすい方法は，自然言語で対話することであるが，自然言語の表現が理解しにくいことと伝える情報量が限られているため，本研究で適切な図表現によって支援ツールとユーザ間の対話を実現する方法を提案し，実装してきた． この図表現技術には，二つの具体的な技術が含まれている．一つは，ユーザへ提供するガイダンスの図表現における必要な図表現コンポーネントである．もう一つは，ガイダンス間の階層的な関係を表す「要求ツリー構造」である．

② ユーザへ提供するガイダンスの図表現における必要な図表現コンポーネント

ユーザへ提供するガイダンスは，指定されたデータ型の中身をユーザに入力してもらうために必要な指示である．それらを適切な図表現コンポーネントで表し，ユーザからの入力を認識できるために，ユーザへのガイダンスは，いくつかの事前に定義したガイダンスの種類に分けている．一つの種類のガイダンスに対して，一つの図表現コンポーネントが設計された．これらガイダンスの種類は，次に具体的に説明している．

- *varValue* — 指定された変数の値を入力してもらうガイダンス．この種のガイダンスに対して，一つの Textbox という図表現コンポーネントがディスプレイされる．
- *strValue* — 文字列を入力してもらうガイダンス．この種のガイダンスに対して，一つの Textbox という図表現コンポーネントもディスプレイされる．
- *constraint* — オブジェクトに対する制約を入力してもらうガイダンス．この種のガイダンスにたいして，一つの Textbox という図表現コンポーネントもディスプレイされる．
- *boolValue* — 真または偽を入力してもらうガイダンス．この種のガイダンスに対して，一つの ComboBox という図表現コンポーネントがディスプレ

イされる.

- *choice* ― いくつかの項目を選択してもらうガイダンス．この種のガイダンスに対して，選択する項目が表すオブジェクトの種類によって違う図表現コンポーネントがディスプレイされる．次は，四種類の違う図表現コンポーネントを挙げる.
  - ComboBoxTreeView という仕様パターンの集合から一つのパターンを選択してもらうガイダンスの図表現コンポーネント.
  - RadioButton を持つ主パネルのサブパネルという指定された要素のいくつかの定義から一つを選出するガイダンスの図表現コンポーネント.
  - CheckBox を持つ主パネルのサブパネルといういくつかの項目から一つを選択するガイダンスの図表現コンポーネント.
  - ComboBox という他のガイダンスの図表現コンポーネント.
- *composite* ― 違うガイダンス種類のガイダンスを含めるガイダンス．この種のガイダンスに対して，主パネルにいくつかの違うサブパネルという図表現がディスプレイされる.

- *set* ― 同じガイダンス種類のガイダンス集合を表すガイダンス．この種のガイダンスに対して付けられる図表現は，その集合のガイダンスの種類によって動的に決められる.

③ ガイダンス間の階層的な関係を表す「要求木構造」

一つの形式仕様を形成するために，ユーザへ提供する多数のガイダンスを階層的にディスプレイすることが必要である．そのガイダンス間の階層的な関係を理解しやすい形で表現するには，本研究で提案した要求木構造によって実現した．この木構造において一つのノードが必要な要求を検索するガイダンスの集合を表す．図 3-5-1 に仕様パターン“alter”が応用されるときに使われたガイダンスの図表現の一部が説明されている.この図表現は，左側と右側に分けっている．左側は，要求木構造を表し，右側は選択された木構造のノードに対応するガイダンスの詳細を示す.

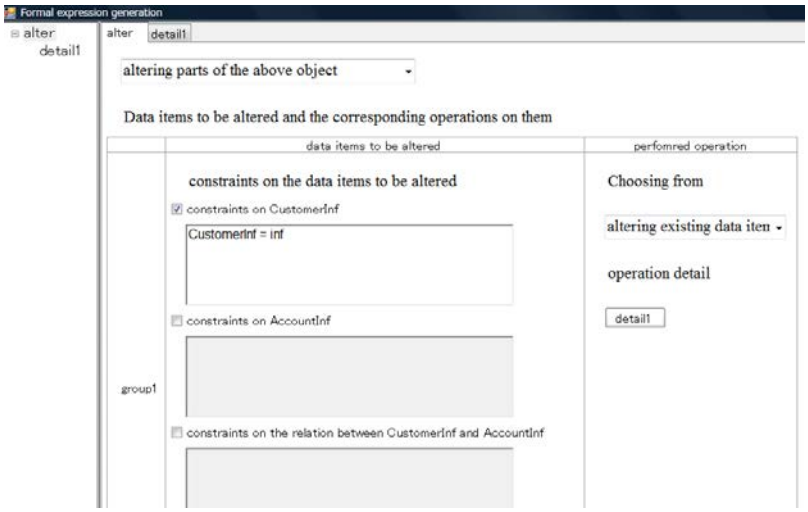


図3-5-1　仕様パターン“alter”が応用された時のガイダンスの図表現

④ ガイダンスの図表現を生成するアルゴリズム

前述で説明したガイダンスの図表現を生成するために，適切なアルゴリズムが必要である．そのアルゴリズムは，仕様パターン知識を表現する階層有限状態遷移図 HFSM によって形成される．基本的なアイディアは，その HFSM を適用するときに，状態遷移に付けている要素の値を入力する要請“req(e)”を図表現へ適切に変換することである．図 3-5-2 に文字列“req(e)”から図表現を生成する再帰的アルゴリズムが示されている．このアルゴリズムを実行するには，要素 e，この要素を定義するガイダンスの parent 要素，およびガイダンスをディスプレイするパネルが必要である．

```
showGuidance(Element e, Element parent, Control panel){
    set eValue as the expected value of e;
    if(parent == null){
        if(eValue is a variable)
        {
            add a label to panel for displaying the request for e;
            add a TextBox to panel with its name set as the name of e;
        }
        else if(eValue is an item chosen from a set of candidate items)
        {
            create a ComboBox with the candidate items and add it to panel;
        }
        else if(eValue is a set of values of the same type)
        {
            if(eValue is a set of composite values with fields f1, f2, …, fn)
            {
                create a TableLayoutPanel tlp with n columns and one row;
                attach a "add an element to the set" Button and set its action for click;
                set tlp(0, i) as fi;
                i = 1;
                for each children element e' of e{
                    j = 1;
                    for each field f' of e' {
                        showGuidance(e', e, tlp(i, j));
                        j++;
                    }
                    i++;
                }
            }
            else if(eValue is a composite value){…}
            … // for other kinds of eValue
    }
    else{
        set parentValue as the expected value of parent;
        if(parentValue is set of composite values){
            if(eValue is of the type of field f of the subtype of parentValue){…}
            else{…}
        }
        else if(parentValue is a composite value){…}
        … // for other kinds of parentValue
    }
}
```

図3-5-2　ガイダンスの図表現を生成するアルゴリズム

⑤ ユーザからの入力を収集するアルゴリズム

ガイダンスの図表現によって，ユーザが要請されたデータ項目の中身を入力することになっている．それら入力されたものは，分析し，適切なデータ構造へ変換することが必要である.この情報によって次のガイダンスの図表現を作り出すアルゴリズムは実行される．図 3-5-3 にユーザからの入力を収集するアルゴリズムが示されている．

```
List<Inf> getInf(Element e, Control c){
    Create a new list infs;
    set eValue as the expected value of parent;
    if(c is TextBox){
       create a Inf containing the information of the text of c;
       add the Inf to infs;
    }
    else if(c is TableLayoutPanel){
       if(eValue is a set of values of the same type){
          if(eValue is a set of composite values with fields f1, f2, …, fn){
             i = 1;
            for each value belonging to eValue{
                  j = 1;
                 for each f belonging to value{
                    set e' as the element corresponding to f;
                    newInfs = getInf(e, c(i, j), e');
                    add each Inf in newInfs to infs;
                 }
            }
          }
          else if(…){…}
       }
       else if(…){…}
    }
    else if(c is ComboBox){…}
}
```

図3-5-3　入力情報を収集するアルゴリズム

⑥ 支援ツールの適用事例

一つの銀行システムを事例として使い，その支援ツールの機能を簡潔に試した．その中で，「引き出す」（*withdraw*）という機能の形式仕様を作り出すプロセスを，ここで重点的に説明する．

図 3-5-4 にその支援ツールの全体の GUI 構造が示されている．左側に仕様パターンを分類しているパターンシステム（function selection）が表されている．左側にプロセス（processes）の仕様を形成するために必要なモジュールの情報（例えば，定義された型，宣言された状態変数，不変条件など）が示されている．

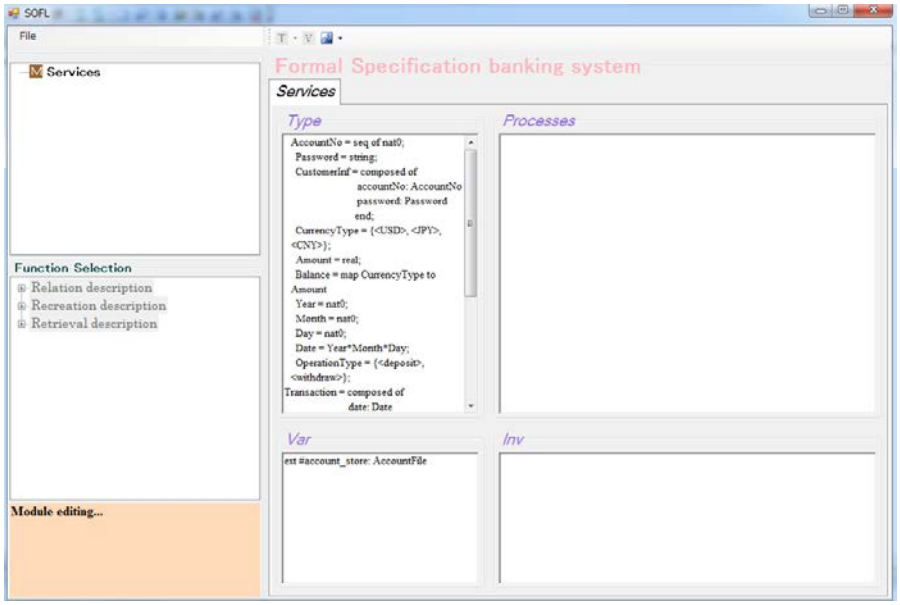


図3-5-4　支援ツールGUIの構造のスナップショット

プロセス「引き出す」(*withdraw*) は，*account_store* というデータストア変数の値を更新する機能を持っている．この機能の形式論理式を形成するために，ユーザが「function Selection」エリアにある「Recreation description」を選択し，それに応じてコンピュータが“alter”という仕様パターンを選択した．図 3-5-5 にパターン“alter”が選択された状態が示されている．

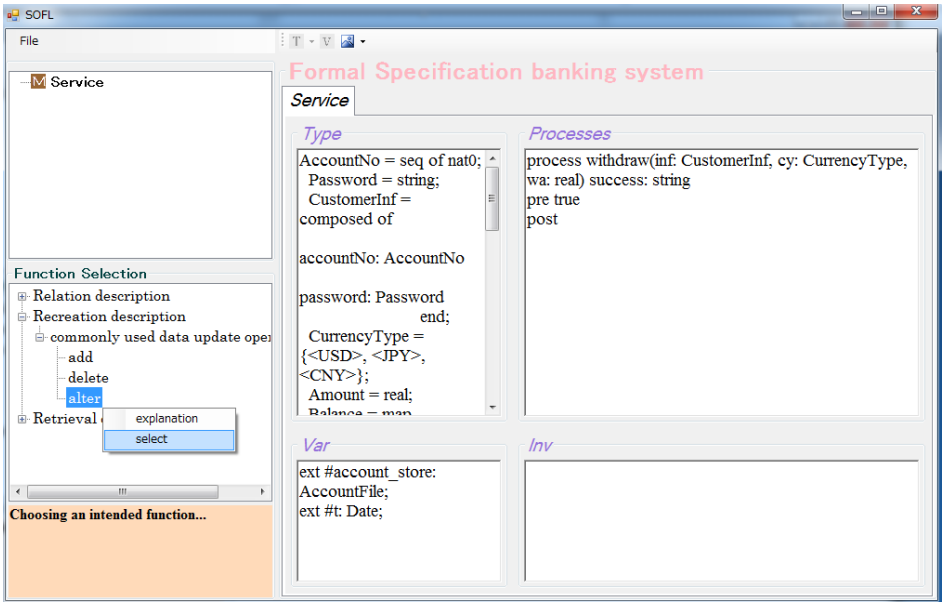


図3-5-5　仕様パターン“alter”が選択された状態のGUIのスナップショット

次に，このパターンの有限状態遷移図 FSM によって，ユーザとの対話をしながら最終的

な形式論理式が作り出される．図 3-5-6 にその対話のプロセスの中の一つのシーンが示され，図 3-5-7 に最終的に生成された形式論理式を表示する GUI のスナップショットが示されている．

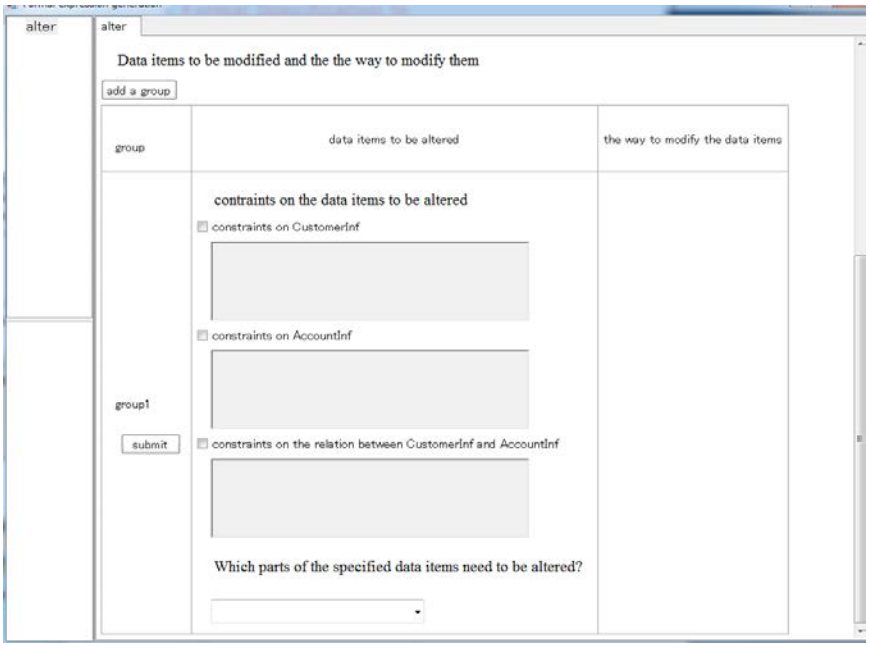


図3-5-6　ガイダンスの図表現のスナップショット

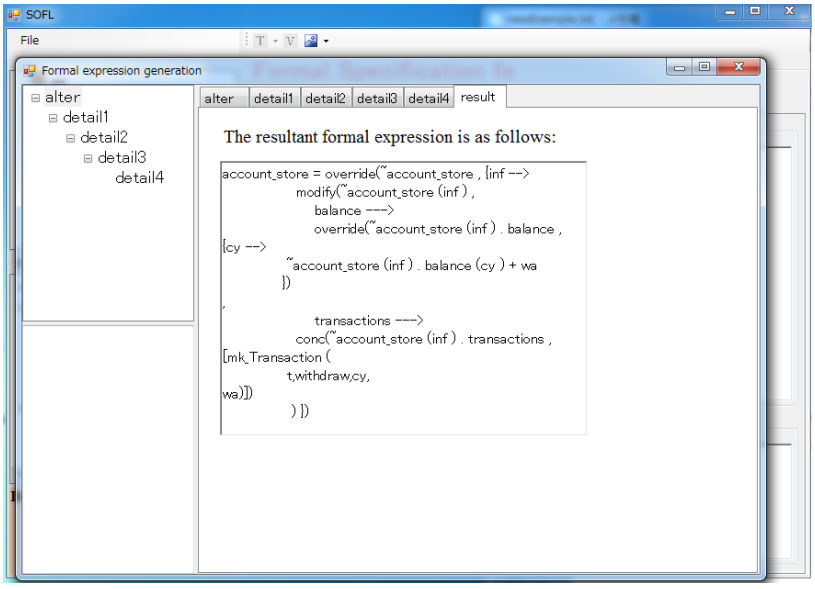


図3-5-7　形成された形式論理式をディスプレイするGUIのスナップショット

3.5.3 実用化へ向けた課題と問題点

(1) 課題と問題点

現時点は仕様パターンに基づく形式仕様作成の支援ツールのプロトタイプを構築したが，実用化するためには，以下の三つの課題が残っている．

(a)支援ツールにより多くの仕様パターンを追加することが必要である．この課題を解決するためには，どのような仕様パターンを設計すべきであるかの問題点がある．

(b)開発者と対話する GUI の有効性を評価することが必要である．この課題を解決するためには，評価を実施する実験にどのように企業の実務者が利用できるかの問題点がある．

(c)自動生成した形式仕様の正しさをどのように保証するか．この課題を解決するためには，生成された形式仕様の正確性をどのように検証できるかの問題点がある．

(2) 将来の応用方法

将来の応用方法としては，以下の二つが想像できる．

(a)ソフトウェア要求仕様の作成或いは設計仕様の作成に使われる．

(b)自然言語で書かれたソフトウェア要求仕様を深く理解するために使える．

3.6 研究目標 6「CDFD から機能シナリオを自動的に生成するツールの作成」

3.6.1 当初の想定

(1) 想定する仮説等

SOFL 形式仕様には，CDFD でシステムのアーキテクチャーを定義し，その中身は，モジュールで形式に定義する．このような形式仕様の意味を動的に表現するアニメーションには，CDFD からシステム機能シナリオを導出することが必要である．これを達成するために，CDFD の構造を自動分析した上で，システム機能シナリオを抜き出すことが想像できる．

(2) 当初の到達目標

次の三つを目標として制定した．

(a)データフローループを含めない CDFD からシステム機能シナリオを自動生成するアルゴリズムを実現する．

(b)データフローループを含む CDFD からシステム機能シナリオを自動生成するアルゴリズムを実現する．

(c)支援ツールを開発する．

(3) 当初の期待される効果

期待される効果としては，形式仕様を三段階によって効率的に作成することと形式仕様のアニメーションを効果的に実施することができる．これによって形式仕様の整合性と正当性の検証をより短い時間の中で，効果的に実現することができ，顧客に形式仕様に記述された機能およびデータ構造を確認してもらうことができる．

3.6.2 研究プロセスと成果

(1) 研究プロセス

この研究については，次のステップを取った.

① 第 1 ステップ：SOFL 三段階漸進的な形式仕様記述手法の支援ツールを構築.

② 第 2 ステップ：CDFD からシステム機能シナリオの自動生成技術を研究.

③ 第 3 ステップ：システム機能シナリオの自動生成の支援ツールを構築.

(2) 具体的な研究成果の内容

この部分の研究成果は，SOFL 三段階漸進的な形式仕様記述手法の支援ツールと CDFD からシステム機能シナリオの自動生成技術を含める.

① SOFL 三段階漸進的な形式仕様記述手法の支援ツール

仕様アニメーションを SOFL 三段階仕様記述技術に統合して得た SOFL 三段階漸進的な形式仕様記述手法の支援ツールは，SOFL 三段階形式仕様記述技術の支援ツール，形式仕様パターンに基づく形式仕様作成の支援ツール，および各レベルの仕様アニメーションの支援ツールを統合したツールである．形式仕様パターンに基づく形式仕様作成の支援ツールと仕様アニメーションの支援ツールは，前述で詳しく説明したが，SOFL 三段階形式仕様記述技術の支援ツールは，詳しく述べていない．ここで，それに集中して説明する．この支援ツールは，次の機能を提供している.

1)仕様の作成と編集のための GUI 構造

各レベルの仕様を作成かつ編集するために使える GUI は，図 3-6-1 に示したように基本的に左，中，右という三つの部分に分割している．左部分は，Module Explorer という非形式仕様，半形式仕様および形式仕様のファイル情報を管理するウィンドおよび形式仕様のコンポーネントの性質（properties）を表示するウィンドを含める．中部分は，FormalCDFDDrawBoard という形式仕様に使われる CDFD を描画するウィンドである．右部分は，CDFD に対応するモジュールの形式仕様を記述するウィンドである．一つの作業（例えば，半形式仕様の作成）に対して，いくつかのウィンドを組合わせて使う可能性がある.

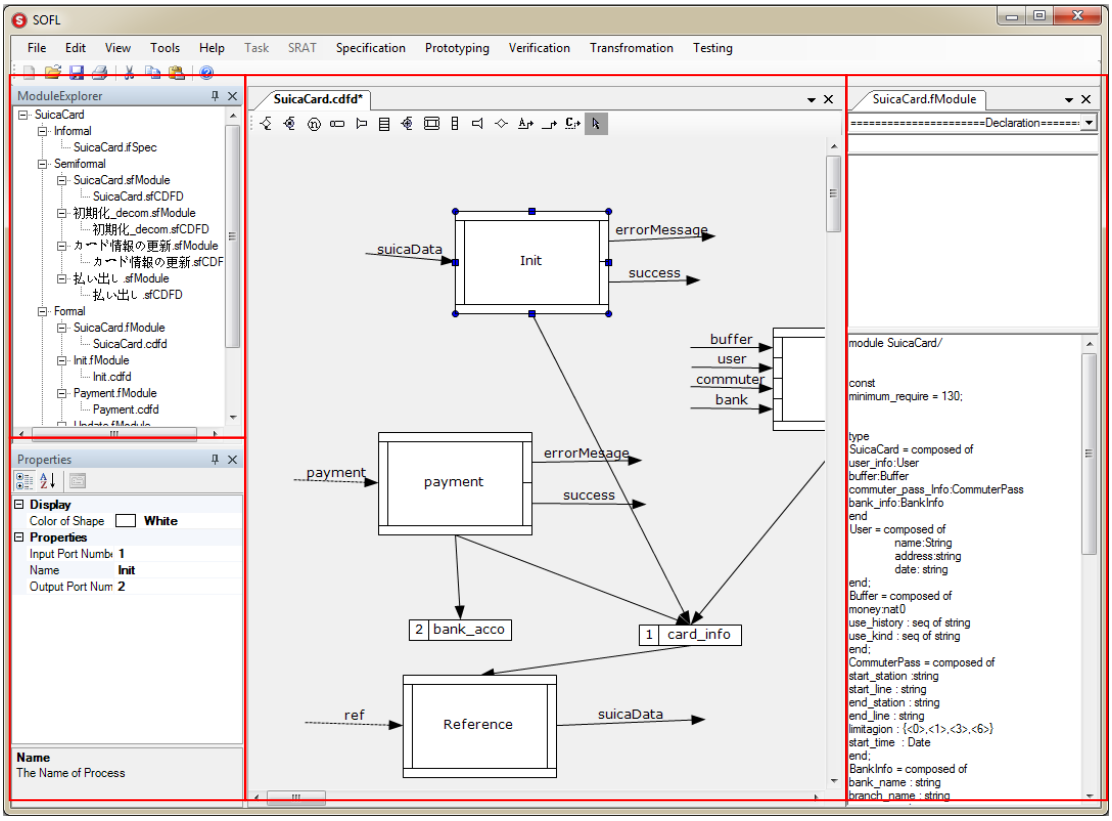


図3-6-1　SOFL三段階形式記述技術の支援ツールのGUI

2)プロジェクト階層

新しいソフトウェアシステムの仕様を作成するときに，一つの新しいプロジェクトが生成される．このプロジェクトには，非形式仕様，半形式仕様，形式仕様，およびクラス（SOFL仕様においてオブジェクト変数を宣言するためにユーザが定義した型）が作成されることができる．半形式仕様と形式仕様に対しては，CDFD が描くことが可能であるが，半形式仕様には，必ずしも CDFD が作成される必要がない．この関係は，図 3-6-2 に示した通りである．

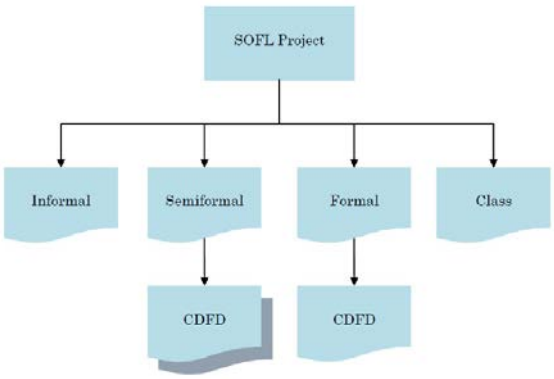


図3-6-2　SOFLプロジェクトの構造

ユーザは,Module Explorer というウィンドに各レベルの仕様のファイルを作成したり，編集したりすることができる．図 3-6-3 はこの作業の様子を示す.

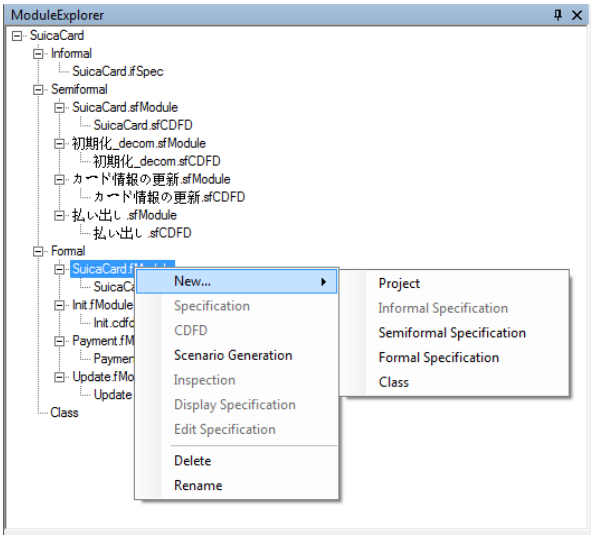


図3-6-3　Module Explorerウィンドに仕様ファイルを生成する様子

3)非形式仕様エディタ

ユーザは，非形式仕様エディタを通じて，非形式仕様を作成する．このエディタは，システム機能，データリソースおよび制約という非形式仕様に記述しなければならない部分のタイトルを自動的に生成し，各部分の階層的な項目に番号を自動的につける機能も提供している．さらに，各部分の項目は，他の部分の項目に関係があることを示すためには，その項目に自動的に連結する機能も提供している．例えば，図 3-6-4 に示したように，データリソース項目 2.2 buffer (F1.2.1)は，この項目がシステム機能部分において定義された機能 1.2.1 項目と関係があることを表している.

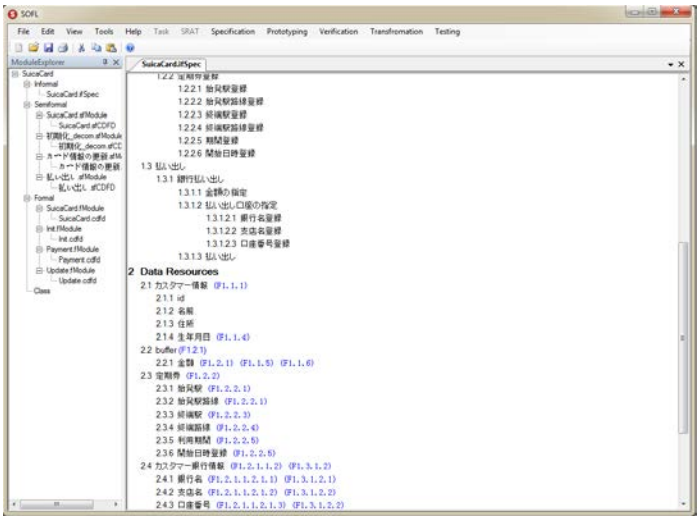


図3-6-4　非形式仕様エディタ

4)半形式仕様エディタ

SOFL 半形式仕様は，いくつかのモジュールから構成されたものである．一つのモジュールにおいて定義する項目は,SOFL 言語によって決まっている.図 3-6-5 に説明したように，ユーザが予め用意された項目リスト（例えば，型定義，変数定義，不変条件定義，プロセス仕様など）から作成したい項目を選択し，その中身をエディタエリアで作成する．作成した内容は，display エリアで示す.

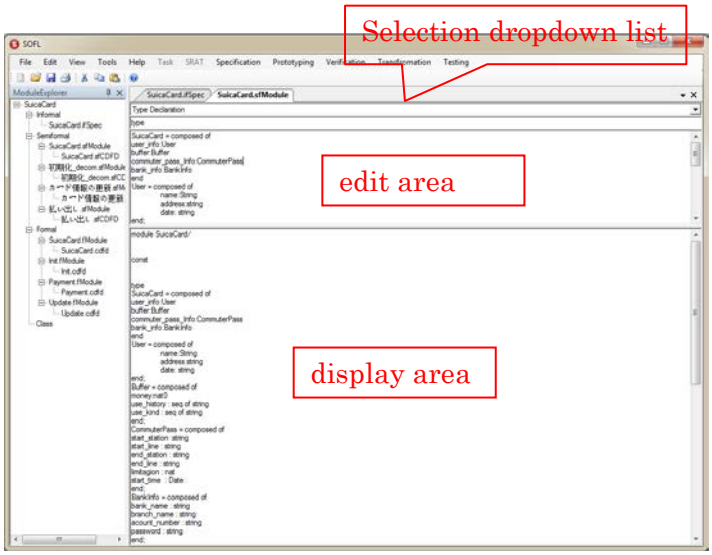


図3-6-5　半形式仕様エディタ

5)形式仕様エディタ

形式仕様エディタは，半形式仕様エディタと基本的に同じ機能を提供しているが，プロセスの形式仕様を仕様パターンに基づく作成する支援ツールを利用できる．図 3-6-6 にそのツールを使う時の画面が示されている．また，本エディタは，CDFD とモジュールの一致性を守る機能も提供している．この一致性によって，CDFD に使われているプロセスの入力

データフロー，出力データフロー，データストア，およびプロセス名などは，すべてそのままを対応するモジュールに維持される．このため，形式仕様にプロセス，データフロー，データストアなどを更新するときに，必ず CDFD から編集始める．CDFD に起こった更新は，対応するモジュールに一致性を守りながら自動的に反映する．

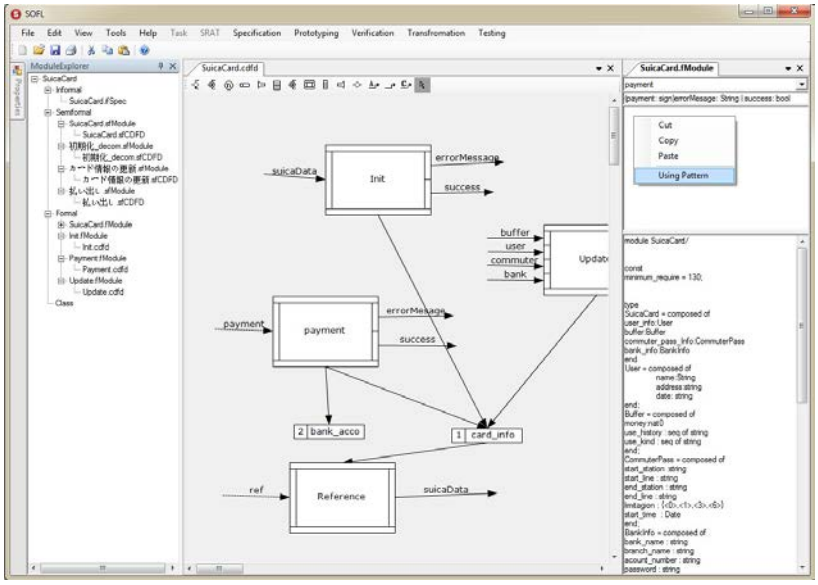


図3-6-6　仕様パターンを使う時の画面のスナップショット

6)CDFD 描くボード（CDFD Draw Board）

GUI の中部は，CDFD を描くエリアである．CDFD を効率的に描くためには，CDFD に使えるすべてのコンポーネントのアイコンは，メニューに示している．ユーザが描きたいコンポーネントのアイコンをそのメニューから選択すると，描くエリアで簡単にそのコンポーネントの図表現を描くことができる．描かれた CDFD を編集したり，コンポーネントを増減したり，プロセスのインタフェースの要素を変更したりすることが簡単に作業することができる．図 3-6-7 に CDFD 描くボードのスナップショットを示している．

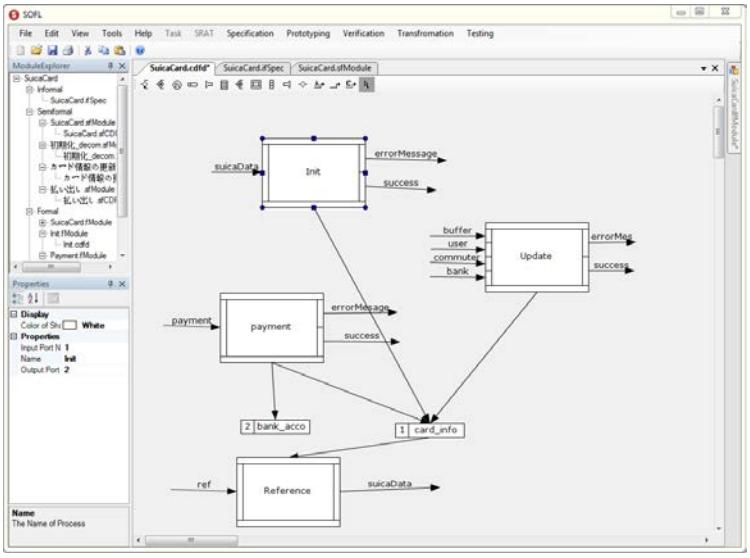


図3-6-7　CDFD描くボードのスナップショット

7)プロセスの分解

大規模システムの形式仕様は，一般的に階層的 CDFD で表現する．それを作成する方法としては，高いレベルのプロセスを低いレベルの CDFD に分解することである．本支援ツールは，このようなプロセス分解の機能も提供している．図 3-6-8 に分解するプロセスの選択した状態が示されている．分解して得た低いレベルの CDFD を書くためには，同じような画面が用意される．

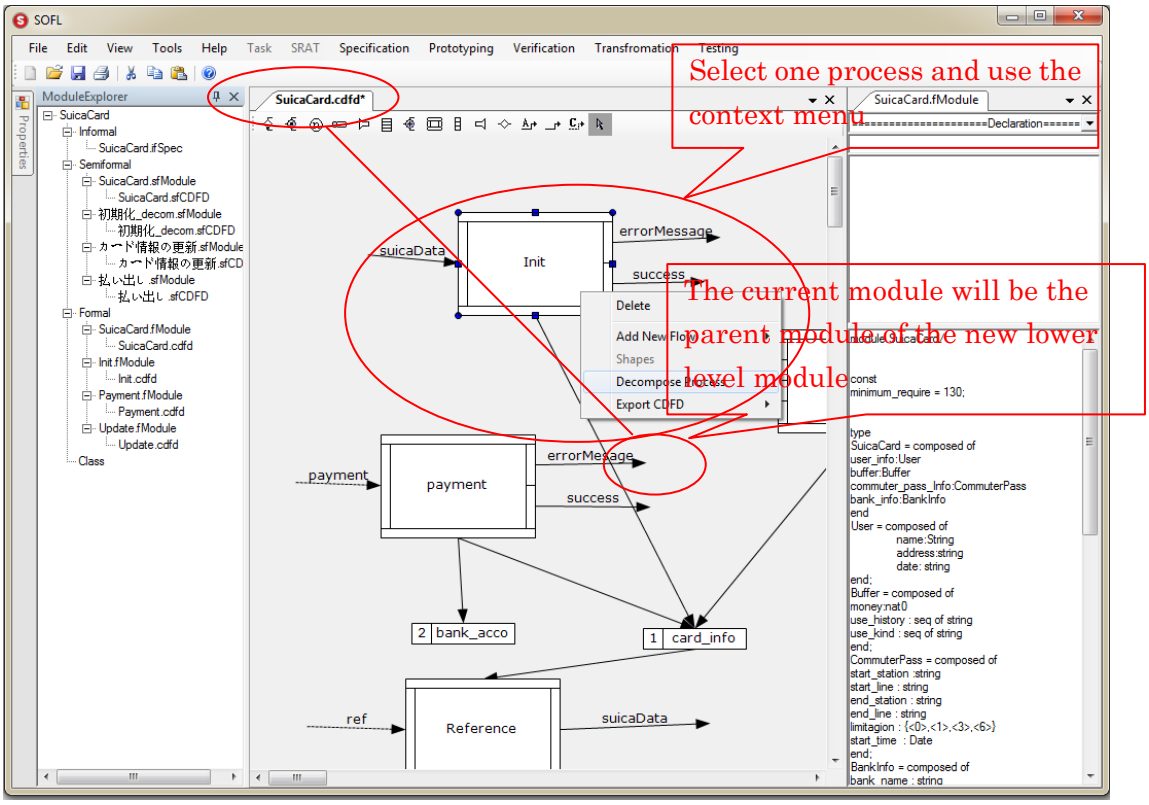


図3-6-8　プロセスの分解

8)仕様のエクスポート

作成されたすべての仕様（非形式，半形式，および形式仕様）を，MS Word ファイル および RTF ファイルへエクスポートすることができる．また，CDFD も JPEG，WMF，BMP，および PNG フォーマットのファイルへエクスポートすることもできる．このようにエクスポートされたファイルは，他の文書を作成するソフトウェアに使われられるため，非常に便利である．

② CDFD からシステム機能シナリオの自動生成技術

研究成果としては，二つである．

(a)CDFD からシステム機能シナリオを自動生成するアルゴリズムを実現した

(b)システム機能シナリオの生成の支援ツールを構築した．

次に，これらそれぞれの機能を詳しく紹介する．

1)CDFD からシステム機能シナリオを自動生成するアルゴリズム

本アルゴリズムの基本アイディアは，まず CDFD から入力データフローと出力データフロー間の関係をより細かく表示するグラフへ変換し，それによりシステム機能シナリオが自動生成されるという方法である．CDFD からグラフへ変換するために，プロセスの構造から入力ポートと出力ポートの集合に分解し，それからすべて関係がある入力ポートと出力ポートを矢印で結ぶ．例えば，図 3-6-9 にプロセス Process から入力ポートと出力ポートの組合せを表現するグラフへ変換することが示されている．

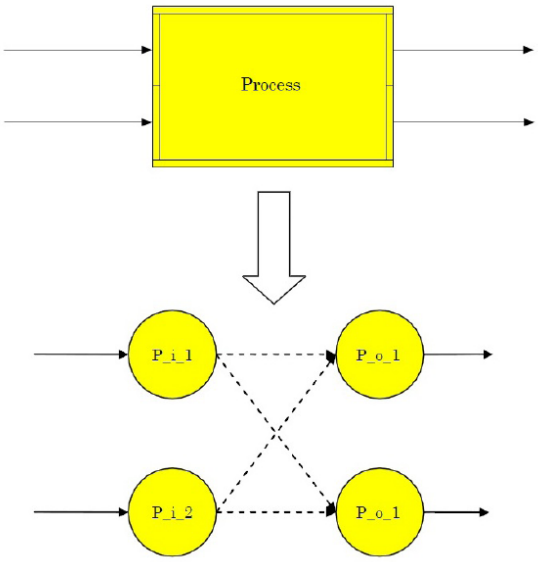


図3-6-9　プロセスからグラフへの変換

この図の中で，P_i_1 は，プロセス Process の一番目の入力ポートを表し，P_o_1 は同じプロセスの一番目の出力ポートを表す．　このような方法を，図 3-6-10 に示した簡単化された ATM システムを定義する CDFD に適用すると，図 3-6-11 に示した入力ポートと出力ポートの組合せを反映するグラフになる．

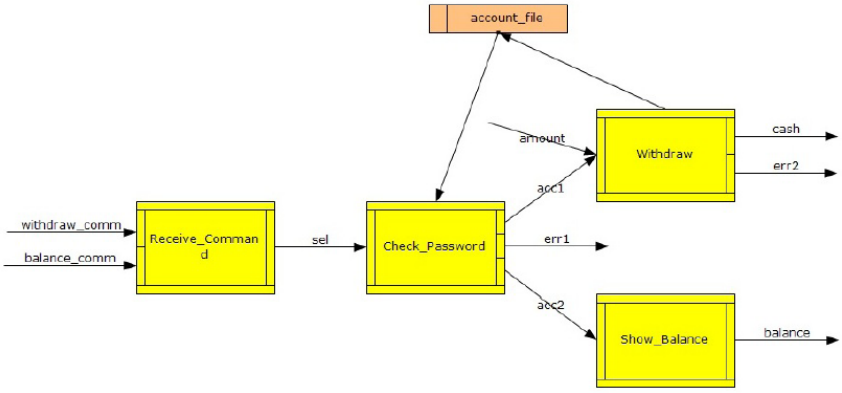


図3-6-10　簡単化されたATMシステムのCDFD

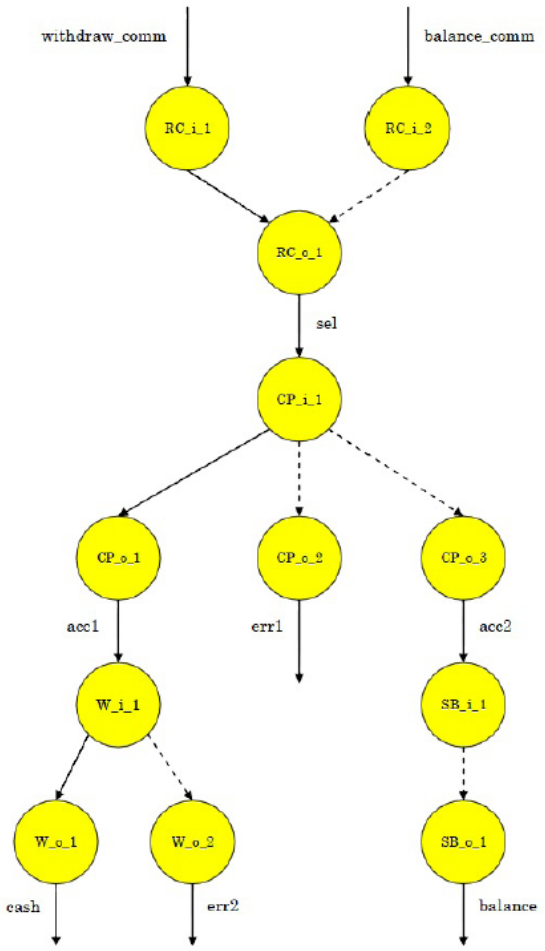


図3-6-11　入力ポートと出力ポートの組合せを反映するグラフ

このグラフの開始ノードから終了ノードへのすべてのデータフローパスは，そのシステムのすべての機能シナリオとなる．図 3-6-12 に CDFD からシステム機能シナリオを自動生成する画面のスナップショットが示されている．

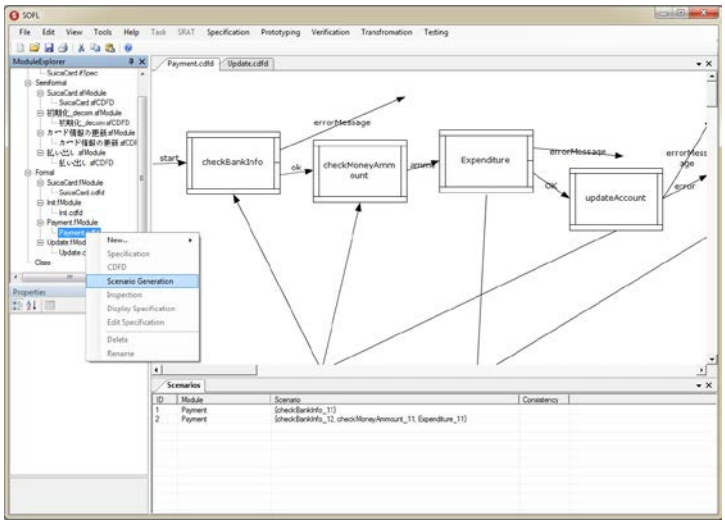


図3-6-12　システム機能シナリオの自動生成画面のスナップショット

③ システム機能シナリオの自動生成の支援ツール

システム機能シナリオの自動生成機能は，SOFL 三段階形式仕様記述技術の支援ツールに一つの機能として実装されている．具体的な CDFD を受けると，この CDFD から生成されるすべての機能シナリオが提供される．そのなかで，一つの機能シナリオを選択すれば，そのシナリオのアニメーションを実施することが可能である．ただし，アニメーションのために必要なテストデータの自動生成は，本支援ツールにはまだ実現していない．ユーザからの入力したテストケースを使い，システム機能シナリオをアニメーションすることができる．例えば，図 3-6-13 に示した ATM システムの CDFD から選択されたシステム機能シナリオをアニメーションするステップ 1，2，および 3 は，図 3-6-14，3-6-15，および 3-6-16 に示されている．

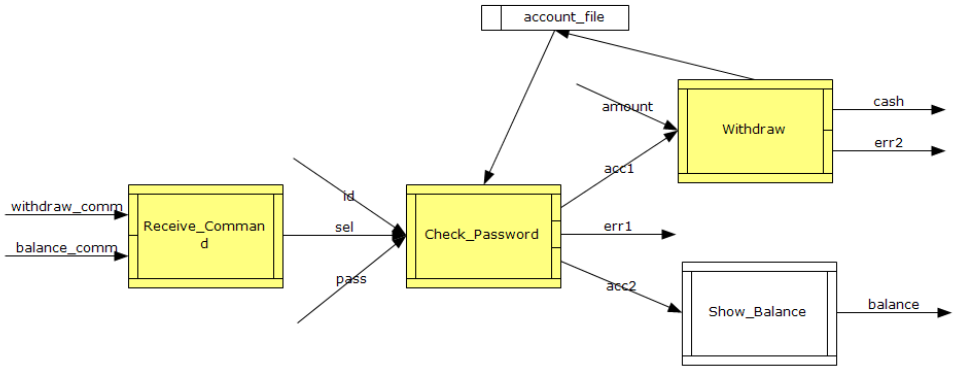


図3-6-13　ATMシステムのCDFDから選択された機能シナリオ

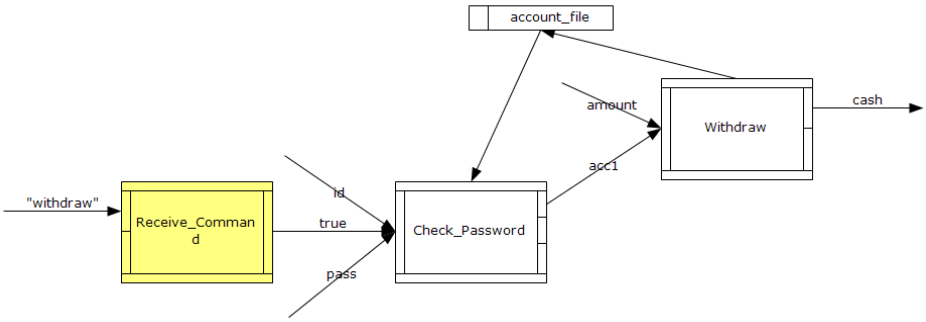


図3-6-14　アニメーションのステップ1

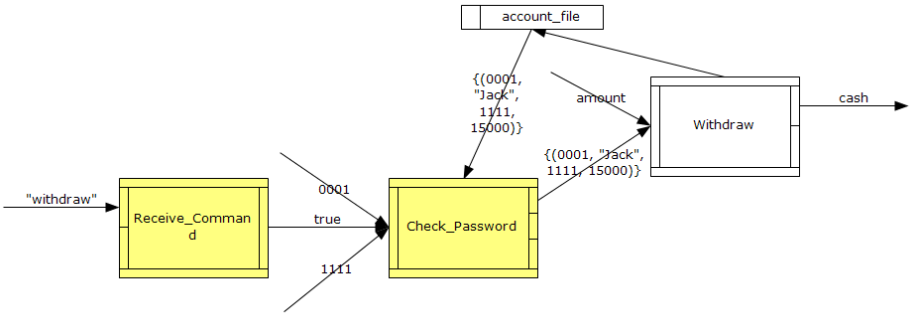


図3-6-15　アニメーションのステップ2

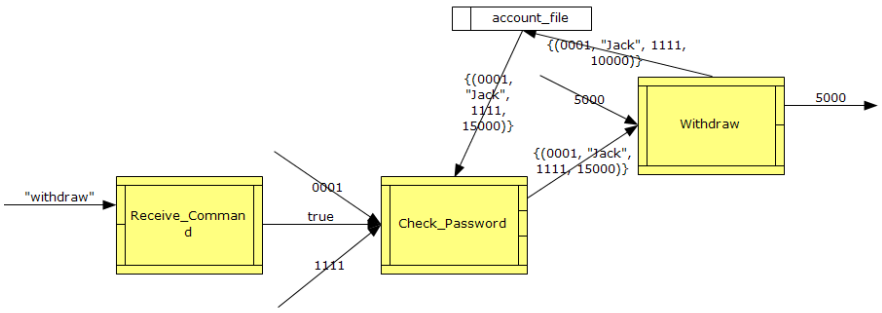


図3-6-16　アニメーションのステップ3

3.6.3 実用化へ向けた課題と問題点

(1) 課題と問題点

現時点では，SOFL 三段階漸進的な形式仕様記述技術の支援ツールおよび CDFD からシステム機能シナリオを自動的に生成することの支援ツールの構築ができていて，実用化するためには以下の四つの課題が残っている．

(a)SOFL 三段階漸進的な形式仕様記述技術の支援ツールの有効性の評価が必要.

この課題を解決するためには，どのように企業の実務者に SOFL 三段階漸進的な形式仕様記述技術を応用するかの問題点がある.

(b)生成されたシステム機能シナリオの中で,正当性を持つシナリオをどのように自動または半自動的に選択できるか.

この課題を解決するためには，シナリオに含まれているプロセスの事前条件と事後条件の内容をどのように活用するかの問題点がある.

(c)形式仕様のアニメーションを実施するためには,システム機能シナリオに含まれているプロセスの形式仕様を考慮することが必要である.

この課題を解決するためには，プロセスの形式仕様をどのように分解するかの問題点がある.

(d)形式仕様アニメーションをどのように動的に表現するか.

この課題を解決するためには，アニメーションのためにテストケースをどのように自動的に生成するかの問題点がある.

(2) 将来の応用方法

将来の応用方法としては，二つが想像できる.

(a)SOFL を用いたソフトウェア開発プロジェクトに適用できる. 作成された SOFL 形式仕様を効率的にアニメーションを実施することによって,ソフトウェアの信頼性を向上させることができる.

(b)SOFL 形式仕様で定義したソフトウェアプロジェクト管理プロセスの実施状況を説明するときに使える. システム機能シナリオの自動生成によって，ソフトウェアプロジェクトの実施するいろいろなシナリオを動的に表現することができるため,プロジェクトの管理者がプロジェクトの実施状況を容易に把握することができる.

４．考察

4.1 研究により判明した効果や問題点等

本研究で開発した SOFL 三段階漸進的な形式仕様記述手法と支援ツールは，形式工学手法研究分野で代表的な SOFL 形式工学手法を，さらに発展して，より実用性が高い形式工学手法の道筋を示すことができた．設定した到達目標を達成し，形式仕様記述プロセスに，仕様アニメーションおよび仕様パターンによる形式仕様作成アプローチを統合した特徴を持つ世界で最新なシステム要求仕様と設計仕様作成技術を確立した．本節で，到達目標の完成度，他の類似研究との比較，および本研究の成果による論文作成について具体的に説明する．

4.1.1 設定した到達目標の達成

本研究に設定した三つの到達目標の完成度について，次に説明する．

(1)SOFL の非形式，半形式，および形式仕様のアニメーション技術を確立，既存の SOFL 三段階形式仕様記述プロセスに統合する手法を確立する．

高信頼ソフトウェアシステムを開発するためには，上質な要求と設計仕様が極めて重要である．上質な要求仕様とは，記述した全ての要求機能，データリソースおよび必要な制約は顧客が希望したものと完全に一致することである.同じように,上質な設計仕様とは,正当性を持つ設計者の意図と完全に一致する設計仕様を意味する．問題点としては，どのようにすれば，上質な要求仕様と設計仕様を素早く作成することができるかということである．この問題点は，ソフトウェア産業にとって非常に重要であるが，まだ解決していない.

本研究では，既存の SOFL 三段階形式仕様記述プロセスに，非形式仕様，半形式仕様，および形式仕様アニメーションを統合して，漸進的な形式仕様記述手法を確立した．この手法でソフトウェアシステムの設計仕様を作成することは，図 4-1 に示したようなプロセスになっている．

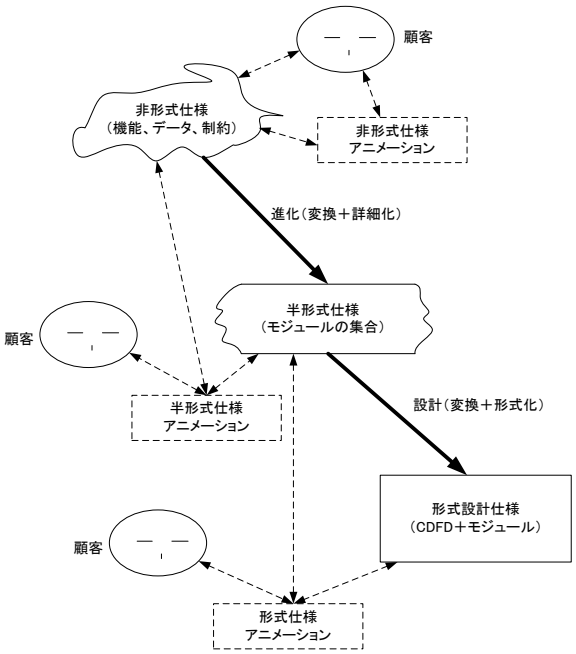


図4-1　SOFL三段階漸進的形式仕様記述プロセス

顧客とココミュニケーションを取りながら，応用領域の資料や現実世界のシステムに関する知識などを検討することから始め,開発するシステムに対する要求を徐々に理解する．その上で,システムの非形式仕様を作成する．この仕様の中に,要求されたシステム機能，その機能を実現するために必要なデータリソース，および機能またはデータリソースに対する制約が簡潔に記述される．機能を記述する際,「抽象と分解」の手法を適用し，大きい機能は小さい機能に分解して，階層的に機能を比較的詳細に記述する．同じように，データリソースおよび制約の部分も階層的に記述することもできる．

一般的には，非形式仕様の内容は，開発者（例えば，要求工学エンジニア）によって作成される．その内容は，システムの要求を提出した顧客に確認してもらわなければ，方向性的な間違いが仕様に含まれる可能性が高い．このため，非形式仕様のアニメーションを実施することによって，その内容を確認することができる．アニメーションをしながら，開発者は顧客とのコミュニケーションを強化することができ，顧客のフィードバックももらえる．獲得した顧客のフィードバックによって，非形式仕様の不完全性を改善させ，記述されたシステム機能，データリソース，および制約間の関係をより明らかにする．このような改善された非形式仕様は，次に半形式仕様を作成する基礎になる．

非形式仕様が自然言語（例えば，日本語）で記述されたため，意味を明確にしていない表現がたくさん含まれている可能性が高い．この問題を解決し，非形式仕様の内容を，より明確に理解できる様にするためには，半形式仕様を作成する．

半形式仕様は，基本的に非形式仕様を進化することによって得たものである．非形式仕様の進化は，二つの行動が含まれている．一つは,「変換」であり，もう一つは,「詳細化」である.「変換」というのは，非形式仕様の全体の構造および記述された機能，データリソース，および制約から，半形式仕様の全体の構造およびモジュールの内容へ変換すること

である．これを実現するために，次の三つが有効である．

(a)非形式仕様に記述された関連しているシステム機能，データリソースおよび制約を，SOFL モジュールに結合させる．

(b)データリソースを適切なモジュールで，型と変数として形式に宣言する．この宣言によってそのデータリソースの意味を明確に定義できる．

(c)システム機能を適切なモジュールでプロセスとして定義する．この定義において，入力，出力，および状態変数を含むプロセスのインタフェースを形式に定義し，プロセスの振る舞いを自然言語で表す事前条件と事後条件により定義する．更に，非形式仕様に記述された制約は，半形式仕様において，不変条件として定義或いは関連するプロセスの一部として定義する．

このように作成した半形式仕様の内容は，開発者と顧客の本音に合致しているかどうかを確認することが必要である．このため，半形式仕様アニメーションを行う．このアニメーションの主な目標は，半形式仕様で定義されたプロセス仕様のすべての面，定義されたデータ型，および関連している不変条件を確認することである．プロセス仕様に対しては，そのインタフェースおよび事前条件と事後条件を確認する．インタフェースに関しては，プロセス名，プロセスの入力変数名と型，出力変数名と型，状態変数名と型を確認する．これによって関連しているデータ型の定義や状態変数の宣言なども検証する．プロセスの振る舞いを定義する事前条件と事後条件は，基本的に自然言語で記述されているため，プロセスの振る舞いのアニメーションは，テストケースと期待される結果を作成した上で適切に行う．ただし，事前条件と事後条件は形式に定義されていないため，このアニメーションを系統的に行うのは困難である．

このアニメーションに基づき行ったアニメーションによって，開発者と顧客からのフィードバックを獲得することができ，それによって半形式仕様を改善することが実現できる．このように作成したシステムに対する顧客の要求を比較的明確に定義した半形式仕様は，次の段階で行うシステム設計仕様作成の基礎になる．

システム設計の目的は，半形式仕様で定義したプロセスを適切に統合することを伴い，必要な新たなプロセスを追加することによって，システムのアーキテクチャーを確定し，システム全体の機能をアーキテクチャーからプロセスまですべて形式的に定義する．これを達成するために，二つのことを完成しなければならない．

(a)関連しているプロセスを適切に統合してシステムのアーキテクチャーを反映する CDFD を作成する．

(b)作成された CDFD に含まれているプロセス，データフローおよびデータストアというコンポーネントを，対応するモジュールに形式的に定義する．

具体的には，プロセスの振る舞いは，事前条件と事後条件を論理式で形式に定義する．データフローは，適切な型で宣言する．データストアも適切な型で宣言する．宣言された型やストア変数に関する制約は，不変条件で定義する．

このように作成された形式設計仕様は，顧客と開発者の要求を正しく反映しているかどうかを確認するために，その形式仕様のアニメーションを実施する．このアニメーションによって，システムレベルの視点から様々な具体的な機能を，分かりやすい形で動的に表現する．これを実現するために，次の二つの手順を取る．第一に，システムのアーキテク

チャーを表すCDFDからすべての「システム機能シナリオ」を導出する．一つのシステム機能シナリオは，一種の入力データフローから出力データフローまでのデータフロー列であり，システムの一つの独立な部分機能を表す．第二に，システム機能シナリオを，一つずつアニメーションを実施する．選定された一つのシステム機能シナリオに対し，入力データフローの値とそれに対応する出力データフローの値を生成し，そのシナリオに含まれているプロセスの「実行」を，動的に表現する．この表現によって，開発者と顧客にそのシナリオの振る舞いを確認してもらう．

この目標には，次の四つの具体的な目標が含まれている．

(a)非形式仕様のアニメーション技術の確立

(b)半形式仕様のアニメーション技術の確立

(c)形式仕様のアニメーション技術確立

(d)それらのアニメーション技術をSOFL三段階形式仕様記述プロセスに統合する手法の提案．

目標(a)に対して，節3.1「研究目標1」で述べた通り，SOFL非形式仕様に基づく仕様のアニメーションシステムを構築する方法と支援ツールを開発したことにより，この目標を達成した．

目標(b)に対して，節3.2「研究目標2」で説明した通り，SOFL半形式仕様に基づく仕様のアニメーションシステムを構築する方法と支援ツールを開発したことにより，この目標を達成した．

目標(c)に対して，節3.6「研究目標6」で述べた通り，システムのアーキテクチャーを反映するCDFDからシステム機能シナリオの自動生成アルゴリズムと支援ツールを開発したことにより，この目標を達成した．

これら研究によって，仕様アニメーションの方法論がほぼ確立された．非形式仕様アニメーションは，システムと個々の機能のGUIを決めるために顧客と一緒に実施される．半形式仕様アニメーションは，プロセス仕様のすべての側面（例えば，入力変数と型，出力変数と型，状態変数と型，事前条件と事後条件）を確認するために開発者と顧客と一緒に実施される．形式仕様アニメーションは，設計されたシステム機能を確認するために，自動または対話式で実施される．

目標(d)に対して，前述したSOFL三段階漸進的形式仕様記述プロセスおよび「研究目標6」に説明したこのプロセスの支援ツールを開発したことにより，この目標を達成した．

(2)仕様パターンに基づき形式仕様を容易かつ効率的に作成する技術の確立．

この目標に対して，節3.3「研究目標3：形式仕様パターンの定義と仕様パターンシステムの構造の確立」，節3.4「研究目標4：形式仕様パターン知識の表現，検索，および適用仕組みを提案する」および節3.5「研究目標5：形式仕様パターンシステムを支援するツールの一部の作成」で論述した通り，形式仕様パターンの定義およびそれに基づき分類した全ての仕様パターンを表示する仕様パターンシステムの木構造を確立，形式仕様パターン知識の階層的な有限状態遷移図の表現，それによるパターン知識の検索と適用仕組みを提案，および仕様パターンによる形式仕様作成アプローチの支援ツールを開発したことにより，この目標を達成した．

4.1.2 他の類似研究との比較

本研究で提案した SOFL 三段階漸進的形式仕様記述技術と支援ツールに似ている世界で他の研究は，私たちの調査の限り，ほとんどないと言える．但し，単純な目標を実現するために提案された個々の技術の基本アイディアが，本研究で開発した技術の中のある具体的な技術に似ているところがある．

Bumbulis 研究グループは,ソフトウェアコンポーネントに基づくユーザインタフェースの開発における形式手法とプロトタイピングを統合した手法を提案した[16]．基本的なアイディアは，形式仕様を作成し，それをアニメーションすることによって，システムの振る舞いを観察することができるし，それの重要な性質を形式的に検証することも可能である．但し，形式仕様をアニメーションするために，仕様から実行できるコードへ自動的に変換することが必要である．この条件満たすためには，形式仕様の表現方式に対して制約しなければならない．Morrey 研究グループと Kazmierczak 研究グループも，同じような特徴を持つ Z 形式仕様記述言語で作成された形式仕様のアニメーションアプローチを提案した[17, 18]．これら技術と比べて，本研究で提案した仕様アニメーション技術は，形式仕様だけではなく，非形式仕様と半形式仕様にも適用できる．更に，形式仕様のアニメーションを実施するために，形式仕様から実行できるコードへ変換する必要がなく，形式仕様そのものだけによる導出したアニメーション条件に基づきテストケースを自動的に生成した上で仕様のアニメーションを実施することが可能である．

Virzi 研究グループと Sefelin 研究グループは，low-fidelity プロトタイピングが要求の発見とシステムの設計のアイディアを示すために使える補助技術として提案された[19, 20]．Low-fidelity プロトタイピングは，紙ベース或いはコンピュータベースという二種類がある．共通の特徴は，プログラミング言語で実行できるコードを書く必要がなく，既存のツール（紙，ペン，ソフトウェアなど）を用いて素早く簡単に顧客の要求やシステムの設計の基本的なアイディアを適切に表現することができる.但し,これら研究論文には,どのように low-fidelity プロトタイピングを系統的に実施することができるかについて述べていない.本研究で提案した Microsoft の PowerPoint ソフトウェアに基づく非形式仕様と半形式仕様のアニメーション技術は,これら研究論文で報告された low-fidelity プロトタイピングの基本アイディアに一致しているが，アニメーションを行う系統的なアプローチを提供している．例えば，SOFL 非形式仕様に記述された顧客が最も関心を持っているシステム機能を選出し，それらの機能一つずつに対して，GUI の構成や関連の内部操作の連結などの情報を反映する「機能構造図」を作成し，この機能構造図に基づき，動的なアニメーションシステムを作成する．また，SOFL 半形式仕様に定義されたプロセス一つずつに対してアニメーションシステムを系統的に構築する．目的は，プロセスのインタフェースと振る舞いを確認することである．形式仕様アニメーションには，システムのアーキテクチャーを表現する CDFD から抜き出された機能シナリオの振る舞いを,テストケースによって動的に確認できる．

実務者に効率的に Z 形式仕様を作成してもらうために，Stepney 研究グループは形式仕様パターンという概念を提案した[21]．さらに，Z 仕様を作成する際対処しなければならない問題を解決するために，六種類の仕様パターンを定義した．それは，表現パターン

（presentation pattern），応用領域パターン（domain pattern），慣用パターン（idiom pattern），アーキテクチャーパターン（architecture pattern），構造パターン（structure pattern），および発展パターン（development pattern）である．Ding 研究グループは，形式仕様を効率的に構築するために，性質保証詳細化パターン（property-preserving refinement patterns）アプローチを提案した[22]．Konrad 研究グループは，いくつかの産業界の組み込みシステムを分析した上で，リアルタイム仕様パターンアプローチを提案した[23]．これらパターンを適用する際，仕様を作成する開発者は，仕様パターンをよく理解した上で，適切なパターンを選択して応用する．本研究で提案した形式仕様パターンに基づく形式仕様作成技術と支援ツールは，これら研究グループが提案した仕様パターンの目標と概念を共有しているが，解決する問題のレベルと手段が違う．解決する問題のレベルに対しては，上述した他の研究グループが開発者に向け仕様パターンを提案したことに対し，本研究で提案した私たちのアプローチは，コンピュータに向け仕様パターンを提案していた．前者の場合，開発者は仕様パターンを読んで理解しなければならないし，選択した仕様パターンを適用するときに形式仕様言語を使用しなければならない．後者の場合，コンピュータは仕様パターンを理解し，それによって開発者に質問したり必要なデータを入力してもらったりする自然言語で対話しながら，要求した通り形式仕様を最終的に自動的に作成する．この場合，開発者は，直接に形式仕様言語を使用しない．

SOFL 形式仕様記述言語が示したような形式手法をソフトウェア開発技術に統合する研究は，90 年代から活発になってきた．例えば，VDM にオブジェクト指向プログラミング仕組みを導入して提案された VDM++は，オブジェクト指向設計の形式仕様を作成することができる[24]．提案された OCL（Object Constraint Language）形式仕様記述言語は，UML（Unified Modeling Language）図で表す意味を明確に定義していない設計を明確に定義するために使われる[25]．OCL を支援するツールも開発された[26]．また，Butler 研究グループは，UML と B-Method を統合するアプローチを提案した[27]．この提案の中で，UML を用いてシステムの構造を表現し，その中に含まれているコンポーネント（例えば，データ項目，操作など）を B-Method で明確に定義する．これら研究で提案された技術は，本研究で開発した SOFL 三段階漸進的な形式仕様記述技術と支援ツールの中に含まれている形式仕様記述言語だけと相当する技術であるが，どのように応用領域の知識から形式仕様を作成するかという具体的なプロセス或いは系統的な手法を提供していない．これに対し，SOFL 三段階漸進的な形式仕様記述技術と支援ツールは，応用領域の知識に基づき顧客の要求を反映する非形式仕様と半形式仕様の作成を通じて，最終的に形式設計仕様の作成までの系統的なプロセスを提供している．この技術の効果は，本研究で実施した事例研究によってほぼ確認した．ただし，形式仕様記述技術の技を詳しく知っていない企業の開発現場の実務者にこの技術の有効性を説明するために，実務者が事例研究に直接に参加することは一番有効な方法と考える．

4.1.3 事例研究による提案した SOFL 技術の評価

本研究で提案した SOFL 三段階漸進的な形式仕様記述手法の支援ツールの品質を向上させ，その形式仕様記述手法の効果を評価するため，「JR 東日本の Suica カードシステム」，

「旅行会社システム」および「スマート交通信号システム」という三つの事例研究を行った．あまり時間がない状況の中で，これら事例研究を行わなければならないため，実施した全ての事例研究の問題は簡単化され，事例研究の規模も適切に設定された．また，既存の他のソフトウェア開発手法と比較して本研究の成果を評価することができなかった．この点に関しては，今後の研究で対処することを考えている．

(1) JR 東日本の Suica カードシステム

現実の JR 東日本の Suica カードシステムを参考して，新たな機能を追加した Suica カードの部分システムの仕様を作成した．この部分システムには，次の 3 つの大きな機能の仕様を作成した．(a) Suica カード，(b)改札機，(c)券売機．Suica カードの機能は大きく分けて以下の 3 個である．初期化，カード情報の更新(チャージ，駅利用，一般利用，定期券登録等)，払い出し（ユーザからのカードの破棄の際の払い出し）である．改札機の機能は大きく分けて次の 2 つである．乗車処理と降車処理である．乗車処理ではユーザが乗車した際に定期券であるか,または最低限の乗車賃がチャージされているかを確認している．また降車処理では，残高と照会し料金が足りない場合はエラーを返す等の処理を行なっている．券売機としての機能は現金を受け付けて，スイカカードにチャージを行い，おつりを払い出すという機能である．

この Suica カードシステムの形式設計仕様を作成するために，まず非形式仕様を作成した．図 4-2 に Sucia カードの情報更新の非形式仕様のスナップショットが示されている．これによってアニメーションを行い，次は半形式仕様を作成した．図 4-3 に半形式仕様に定義された変数，制約，プロセスのスナップショットが示されている．その半形式仕様をアニメーションによって検証した後，最終的に形式仕様を作成した．図 4-4 に Suica カードのトップレベル CDFD が示され,図 4-5 に Suica カードのトップモジュールの形式仕様のスナップショットが示されている．

1.1.2 カード情報の更新
1.1.2.1 金額の更新
1.1.2.1.1 チャージ
1.1.2.1.1.1 券売機による更新
1.1.2.1.1.2 銀行振り込みによる更新
1.1.2.1.1.2.1 カスタマーの銀行口座情報登録
1.1.2.1.1.2.1.1 銀行名登録
1.1.2.1.1.2.1.2 支店名登録
1.1.2.1.1.2.1.3 口座番号登録
1.1.2.1.1.2.1.4 パスワード登録
1.1.2.1.1.2.2 銀行口座からの振込み
1.1.2.1.2 一般利用
1.1.2.1.2.1 利用金額指定
1.1.2.1.2.2 バッファーとの不足をチェック
1.1.2.1.2.3 不足がある場合は追加の料金指定
1.1.2.1.3 駅利用
1.1.2.1.3.1 最低利用金額があるかチェック
1.1.2.1.3.2 定期券区間であるかチェック
1.1.2.1.3.3 改札機から出る際に利用金額があるかチェック
1.1.2.2 定期券登録
1.1.2.2.1 始発駅登録
1.1.2.2.2 始発駅路線登録
1.1.2.2.3 終端駅登録
1.1.2.2.4 終端駅路線登録
1.1.2.2.5 期間登録
1.1.2.2.6 開始日時登録
1.1.2.2.7 名前登録
1.1.2.2.8 住所登録
1.1.2.2.9 生年月日登録

図4-2　Suicaカードの情報更新の非形式仕様のスナップショット

```
var
card_info:SuicaCard;


inv
        3.1  金額は常に0以上5万以下でなければならない
        3.2  定期券利用を行う際はすべての情報が記載されなければならない
        3.3  払い出し先は銀行口座でなければならない

        3.5  銀行振り込みを行う場合は、カスタマー銀行情報がなければならない
        3.6  カスタマー銀行情報はすべて入力されている必要がある
        3.7  カスタマー銀行情報がパスワード以外入力されている必要がある

process init(suicaData:SuicaCard)結果:Bool
ext wr #card_info
pre 情報がすべてnilでないことを保障する。
post 引数に与えられた情報をカードに書き込みを行う。成功したらBoolを返す。
end_process;

process カード情報の更新()
decom カード情報の更新;
end_process;

process 払い出し()
decom 払い出し;
end_process;
```

図4-3　半形式仕様に定義された変数，制約，プロセスのスナップショット

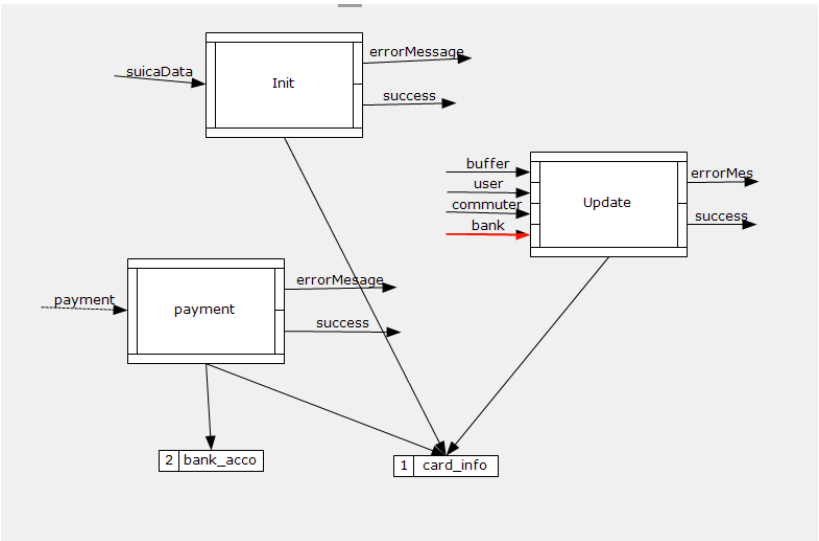


図4-4　Suicaカードのトッポレベル CDFD図

```
var
#card_info : SuicaCard;

inv
card_info.buffer.money>=0 and card_info.buffer.money<=50000;


process Init(suicaData: SuicaCard)errorMessage: String | success: Bool
decom Init;
end_process

process payment(payment: sign)errorMesage: String | success: bool
decom Payment;
end_process

process Update(buffer: Buffer | user: User | commuter_pass: CommuterPass | bank: BankInfo)errorMessage: String | success: Bool
decom Update;
end_process
```

図4-5　Suicaカードのトップモジュールの形式仕様のスナップショット

本事例研究で開発した非形式仕様は，約 99 行（一行には一つの機能，データリソース，または制約が記述されている）の長さであり，半形式仕様は，約 148 行（一行には SOFL の一つの形式的宣言または自然言語の機能表現が記述されている）の長さであり，形式仕様は,約 153 行(一行には SOFL の一つ宣言または論理式が記述されている)の長さである．本事例研究は，一人の協力研究者によって完成した．これによって，個人レベルで SOFL 技術は有効に適用することが確認した．

(2) 旅行会社システム

本事例研究では，JTB オンライン旅行予約システムの機能を参考にした上で，本研究の目標として，開発する旅行会社システムの機能要求，データ要求，および制約などを決めていた．上質な形式設計仕様を最終的に構築するために，本研究で SOFL 三段階漸進的形式仕様記述技術を応用して，次の三つのタスクを完成した．

① SOFL 三段階形式仕様記述プロセスに従って，非形式仕様，半形式仕様，形式仕様を作成した．

非形式仕様は，JTB オンライン旅行予約システムの機能を参考した上で，必要な機能，データリソース，および制約が階層的に記述されている．機能としては，利用者の登録，ローグイン，ローグアウト，利用者のプロフェールの更新，団体旅行の予約，航空券の予約，現地バスの予約，およびホテルの予約など機能が含まれている．

半形式仕様には，非形式仕様が詳細化され，関連する機能，データリソース，および制約が SOFL モジュールにまとめて，より明確に定義できた．システム全体の内容を把握するために，必要な CDFD を描いた上で，モジュールを決めた．

形式仕様は，システムの設計に着眼して，システムのアーキテクチャーを反映する階層的 CDFD をトップダウンの方法で構築した．半形式仕様の中で定義されたデータ型，状態変数，不変条件，およびプロセス仕様などできるだけ再利用していた．ただし，システム設計が行われることが必要であるため，既に定義されたデータ型，状態変数，不変条件，プロセス仕様などは，形式設計仕様に修正されたり，新たなコンポーネントが定義される可

能性もあった.

② 仕様アニメーション

各レベルの仕様の中身を顧客に確認してもらうために，アニメーションを行っていた．非形式仕様のアニメーションを,支援ツールを用いて連続的な PowerPoint スライドを準備した上で実施していた．このアニメーションの主な役割は，非形式仕様に定義されたすべての顧客に注目される機能のインタフェースを確認した.半形式仕様のアニメーションは，同じように実施されたが，基本的に定義されたプロセスの振る舞いを確認する目的になっていた．形式仕様のアニメーションに関しては，CDFD からシステム機能シナリオを支援ツールによって生成して，それらシナリオをインスペクションしていた．支援ツールにシステム機能シナリオのアニメーションをまだ実現していないため，自動導出されたシステム機能シナリオのアニメーションを行うことができなかった．図 4-6 に非形式仕様アニメーションのスナップショットが示され，図 4-7 に半形式仕様アニメーションのスナップショットが示されている.

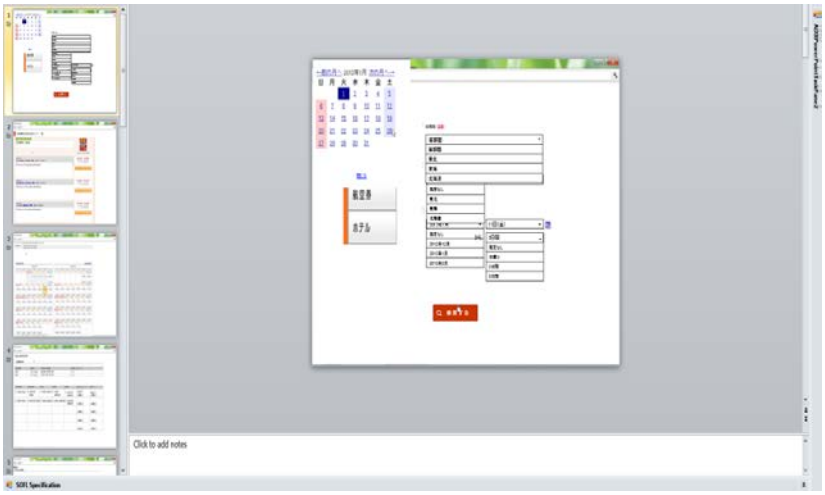


図4-6　非形式仕様のアニメーションのスナップショット

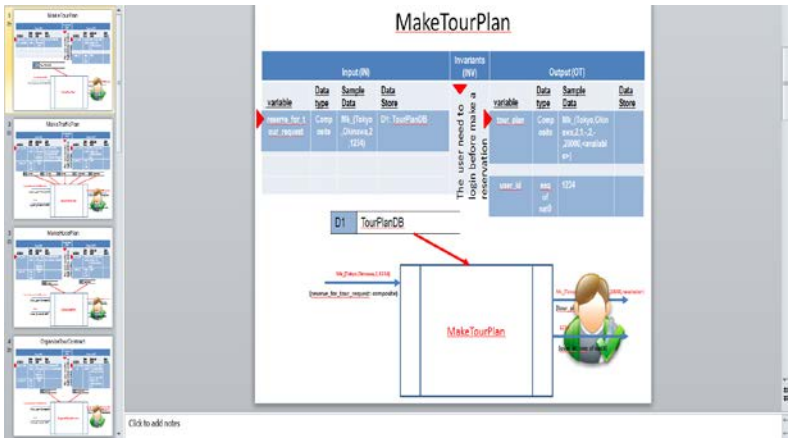


図4-7　半形式仕様アニメーションのスナップショット

③ グループにおいて本研究で開発した技術を使えることが確認された.

本事例研究は，二人の協力研究者が協力しながら行っていた．非形式仕様を記述したときに，二人はXP（eXtreme Programing）の原理を適用し，お互いに助言しながら共同作業によって仕様記述を完成した．半形式仕様を書いた時に，二人はタスクを分割して自分の部分に集中して作業を完成していた．ただし，作業しながら，二人は定義する仕様のコンポーネントについて議論したり，検査したり，訂正したりしていた．形式仕様の構築は，ほとんど同じような方法で完成した．各レベルの仕様アニメーションを行ったときに，共同作業で完成した.

本事例研究で開発した非形式仕様は，約39行（1行には1つの機能，データリソース，または制約が記述されている）の長さであり，半形式仕様は，約218行（1行にはSOFLの1つの形式的宣言または自然言語の機能表現が記述されている）の長さであり，形式仕様は，約391行（1行にはSOFLの1つの形式的宣言または論理式が記述されている）の長さである．本事例研究で確認したことは，本研究で開発した技術が何人を含めるチームにも有効に応用されられることである．時間の制限があったため，既存の他の開発手法と系統てきに比較することができなかったが，二人の以前の開発経験と比べて，SOFL技術はより有効であることを感じた.

(3) スマート交通信号システム

本事例研究では，既存の交通信号システムよりスマートになる交通信号システムを開発することを目指して，SOFL技術を用いてシステムの形式設計仕様を最終的に作成していた.

図 4-8 にスマート交通信号システムが示されている．このシステムは，車（vehicle），交通管理センター（Traffic Management Center），および交通信号機（Traffic Light Controller）から構成されている．すべての車には，センサが付けられ，ワイヤレスコミュニケーション能力がある．車は，交通信号の交差点エリアに入ると，その交通信号機の情報は交通管理センターに送られる．交通管理センターは，交差点エリアに入った車とその交差点の交通信号機の情報を収集し，それによって毎週その交通信号機の信号のディスプレイの時間を計算し，信号機に信号のディスプレイ時間を調整する命令を出すことを主なタスクとして働く．交通信号機は，基本的に信号をディスプレイすることと，交通管理センターからの命令を受け取って，赤と青信号のディスプレイの時間を調整する.

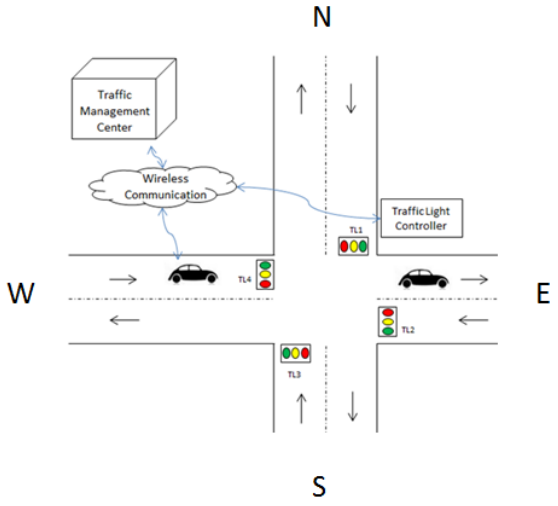


図4-8　交通信号システム

既存の交通信号システムと比べて，スマート交通信号システムの特徴は，信号機は，同じ色の信号をディスプレイする時間は，交差点付近の道路上に車の数によって動的に自動的に調整している．この調整する基準としては，車が多い方向に対して，赤信号の時間が短くなり，青信号の時間が長くなる．同じように，車が少ない方向に対して赤信号の時間が長くなり，青信号の時間が短くなる．これにより，交差点付近の交通は，スムースに流れることが保障されられる．

この交通信号システムの形式仕様を作成するために，非形式仕様の作成から始め，半形式仕様の作成と仕様アニメーションを通じて，システムに対する要求を十分に理解した上で，最終的に形式仕様を作成した．

本事例研究で開発した非形式仕様は，約 22 行の長さであり，半形式仕様は，約 188 行の長さであり，形式仕様は，約 367 行の長さである．本事例研究によって，本研究で開発した支援ツールに含まれた 10 個の重要なバグを発見した．支援ツールの実装者はそれらバグを検討した上で，適切に修正した．

4.2 今後の課題

本研究で開発した技術の成熟度を向上させることに関連する課題と今後の研究予定，および新たに見いだされた課題と今後の研究予定について，本節で簡潔に述べる．

4.2.1 技術と支援ツールの成熟度に関する課題

本研究は設定された到達目標をすべて達成したが，提案した SOFL 三段階漸進的な形式仕様記述技術と支援ツールが産業界で効果的に適用するためには，その中で関連する個々の技術の成熟度を向上させなければならない．この問題を徹底的に解決するためには，いくつかの重要な課題が残っている．本研究の期間が短いため，この課題を対処する時間がな

い．次に，これら課題を一つずつ述べる．

(1) 非形式仕様アニメーションの支援ツールの機能の強化

本研究で開発した非形式仕様アニメーションの支援ツールは，提案した SOFL 非形式仕様に基づくアニメーションシステムの構築アプローチを支援する基本機能を実現したが，次のまだ実現していない二つの機能を支援ツールに追加する必要がある．

① SOFL 非形式仕様からアニメーションシステムを構築する基礎とする「機能構造図」への変換

本研究で開発した支援ツールは，他のソフトウェア（例えば，Microsoft の Visio）で作成された機能構造図を GUI でディスプレイすることができるが，直接にその構造図を作成したり，編集したりすることを支援する機能がまだできていない．この機能を既存の支援ツールに追加する研究開発は，今後の課題となる．

② 新たなコンポーネントアイコンと機能テンプレートを非形式仕様アニメーションの支援ツールに自由に追加する機能

現在の非形式仕様アニメーションの支援ツールには，一般の応用システムの機能のアニメーションを構築するために必要な基本コンポーネントアイコンと機能テンプレートが実装されているが，すべての可能な応用ソフトウェア用のコンポーネントアイコンと機能テンプレートをどのように設計して支援ツールに追加するかは，今後の研究課題となる．想像できる解決方法としては，ユーザ（例えば，開発者，顧客など）が必要なコンポーネントアイコンと機能テンプレートを支援ツールに自由に追加できる機能が実現されることである．

(2) SOFL 形式仕様アニメーションの支援ツールの機能の強化

形式仕様アニメーションの支援ツールに関しては,本研究で設定した CDFD からシステム機能シナリオを自動的に導出する目標を達成したが，導出されたシステム機能シナリオのアニメーションを実施するプロセスを支援する機能がまだ実現されていない．一つのシステム機能シナリオをアニメーションするために，まずアニメーション条件が形成され，それに基づきテストケースが生成され，そのテストケースを用いてアニメーションを行うというプロセスが本研究で明らかにしたが，この技術を有効に適用するためには，そのプロセスに含まれているすべての行動を支援する必要がある．これは今後の研究において重要な課題となる．

(3) ドメイン限定仕様アニメーションの支援

記述された仕様が顧客の仕様に本当に合っているか，抜け漏れがないかを確認できるようなアニメーションを効果的に実施するためには，ドメイン限定仕様アニメーションの支援機能が必要である．本研究で開発した仕様アニメーション支援ツールは，一般のソフトウェアシステムの仕様アニメーションを目指しているが，ドメイン限定のアニメーションコンポーネントや機能テンプレートなどを追加することによって，より効果的かつ効率的

に仕様アニメーションを行うことが考えられる．さらに，仕様アニメーションは，基本的に仕様に記述された機能，データ項目，制約などを顧客の頭の中にあるものと同じかを検証することであるため，顧客に仕様アニメーションの実施プロセスに直接に参加してもらうことが，顧客からの確認を得やすいし，顧客からのフィードバックも効率的に獲得できる．

(4) 形式仕様パターンに基づく形式仕様作成アプローチの支援ツールの強化

本研究で開発した仕様パターンに基づく形式仕様作成アプローチの支援ツールは，そのアプローチの実現の可能性を示すプロトタイプである．すべての応用ソフトウェアの機能を表現するためには，仕様パターン知識がまだ不足である．どのような仕様パターン知識が用意されると，すべての応用ソフトウェアの構造，データ型，および操作機能を表現でき，ユーザへ効果的なガイダンスを提供することができるかは，今後の研究課題となる．この課題の徹底的な解決は，数多くの仕様パターンが設計され，その支援ツールに追加されることが必要である．これはかなり困難であることが想像できる．一方，もしこの技術は，具体的なドメイン（domain,応用領域）に集中すれば，仕様パターン知識の設計が簡単になるし，支援ツールにも簡単に実現できると考えられる．

(5) SOFL 形式仕様の構文と型分析

仕様パターンに基づく形式仕様作成アプローチを用いて作成したプロセスの形式仕様は，正しい構文を保証し，型の一致性も維持する．但し，そのアプローチを使わずプロセスの形式仕様を作成する場合，正しい構文と型の一致性を保証することができない．この問題を解決するためには，SOFL 言語で作成した形式仕様に対して構文分析と型の一致性のチェックを自動的に行う必要がある．これを実現するには，かなり時間が必要であるため，本研究の期間内で実現していない．この課題は，今後の SOFL 形式仕様からプログラムへの自動変換に関する研究の中で，解決する予定がある．

4.2.2 新たに見いだされた課題

確立した技術および開発した支援ツールの成熟度に関する上述の課題の以外の新たな課題も，本研究で見いだされた．次に，それらの課題を一つずつ述べる．

(1) プロセス機能シナリオの自動生成

形式仕様アニメーションを実現するために，CDFD から導出されたシステム機能シナリオを一つずつ実行しなければならない．そのため，アニメーション条件を生成し，それに基づきテストケースを生成することが必要である．但し，アニメーション条件をシステム機能シナリオから生成するために，そのシナリオに関わるプロセスの事前条件と事後条件から導出する「操作機能シナリオ」が必要である．一つの操作機能シナリオは，事前条件，ガード条件と定義条件の論理積であり，操作レベルの一つの独立機能を定義している．ここで，ガード条件とは，事後条件の中で定義された入力変数のみを含める論理式である．定義条件とは，事後条件の中で定義された出力変数を含める論理式である．このような操

作シナリオをどのように自動的に導出するアルゴリズムと既存の支援ツールに実現することが必要である．

(2) 形式仕様アニメーションによる仕様トレーサビリティの構築と検証

形式仕様アニメーションアプローチは，形式仕様の正当性を検証するために提案したが，その正当性を満たすかどうかの判断は，どのように出される問題点でも解決しなければならない．このため，半形式仕様と形式仕様間のトレーサビリティの構築と検証が有効であろう．具体的には，開発者がシステム機能シナリオのアニメーションを実施しながら，そのシナリオが半形式仕様に定義されたどちらの機能を実現する関係かを発見する．これによって半形式仕様と形式仕様間のトレーサビリティを構築する．このトレーサビリティを構築しながら，形式仕様に実現したすべてのシステム機能シナリオの正当性は，半形式仕様において定義された要求を対照することにより判断できる．このように構築したトレーサビリティに基づき，顧客が半形式仕様において定義された機能から始め，そのトレーサビリティを活用して形式仕様において実現したシステム機能シナリオの正当性を検証することができる．このような技術の詳細を解明と共に本研究で開発した支援ツールに実現することは，今後の重要な研究課題となる．

(3) 仕様トレーサビリティに基づく仕様のインスペクション技術

半形式仕様に定義されたものは，形式仕様に必ず実現する．このような原理に基づく各レベルの仕様間のトレーサビリティを構築することができる．このトレーサビリティによって半形式仕様と形式仕様の正当性を検証することができる．この技術の焦点としては，具体的にどんな基準によって仕様間のトレーサビリティを構築できるか，その構築プロセスが自動化できるか，どのようにトレーサビリティによりインスペクションを実施するかという課題である．

(4) 形式仕様からプログラムへの自動変換技術

多大な努力によって作成した形式設計仕様は，システムの実装に活用することが高い価値がある．活用の一つの方法としては，形式仕様からプログラムへの自動変換である．これによって，プログラム開発の効率を向上させ，エラーが生じる危険も少なくなる．SOFL形式仕様からプログラムへの自動変換の場合，三つのレベルの変換についての研究開発が必要である．

(a) プロセスの事前条件と事後条件からプロセスの明確的な仕様（explicit specification）への自動変換
(b) プロセスの明確的な仕様からプログラムへの自動変換
(c) システムのアーキテクチャーの設計を反映する CDFD からプログラムへの自動変換

具体的にどのようにこれら自動変換を実現できるかは，今後の重要な研究である．

参考文献

[1] IFAD. “VDMTOOLS for Quality Software on Schedule – VDM-SL Toolbox User Manual, The IFAD VDM-SL Language”, The Institute of Applied Computer Science, 1999.

[2] J. Woodcock and J. Davies. “Using Z: Specification, Refinement, and Proof”, Prentice Hall, 1996.

[3] J. R. Abrial, “The B-Book: Assigning Programs to Meanings”, Cambridge University Press, 1996.

[4] J. R. Abrial. “Modeling in Event-B: System and Software Engineering”, Cambridge University Press, May 2010.

[5] R. J. Back and J. von Wright. “Refinement Calculus: A Systematic Introduction”, Springer-Verlag, 1998.

[6] C. Morgan. “Programming from Specifications”, Prentice Hall, Second Edition, 1994.

[7] J. Woodcock and P. G. Larsen and J. BicarreGUI and J. Fitzgerald. “Formal Methods: Practice and Experience”, ACM Computing Surveys, 41(4), 2009, pp. 1-39.

[8] D. Bjorner. “Software Engineering”, Springer, 2006.

[9] J. C. Knight, C. L. DeJong, M. S. Gibble, and L. G. Nakano. “Why Are Formal Methods Not Used More Widely?”, Fourth NASA Langley Formal Methods Workshop”, Hampton, Viginia, 1997, pp. 1-12.

[10] D. L. Parnas. “Really Rethinking Formal Methods”, The Computer, 43(1), January 2010, pp. 28-34.

[12] S. Liu. “Formal Engineering for Industrial Software Development Using the SOFL Method”, Springer-Verlag, ISBN 3-540-20602-7, 2004.

[13] S. Liu and Y. Sun. “Structured Methodology + Object-Oriented Methodology + Formal Methods: Methodology of SOFL”, 1st IEEE International Conference on Engineering of Complex Computer Systems (ICECCS’95), IEEE Press, Ft. Landerdale, Florida, U.S.A., November 6-10, 1995, pp. 137-144.

[14] S. Liu, A.J. Offutt, C. Ho-Stuart, Y. Sun, and M. Ohba. “SOFL: a Formal Engineering Methodology for Industrial Applications”, IEEE Transactions on Software Engineering, IEEE Computer Society Press, 24(1), January 1998, pp. 337-344.

[15] S. Liu and X. Xue. “ Automated Software Specification and Design Using the SOFL Formal Engineering Method”, 2009 World Congress on Software Engineering (WCSE 2009)”, IEEE CS Press,
May 19-21, Xiamen, China, 2009, pp. 283-289.

[16] P. Bumbulis, P.S.C. Alencer, D.D. Cowan, C.J.P. Lucena. “Combining Formal Technique and Prototyping in User Interface Construction and Verification”, 2nd Eurographics Workshop on Design, Specification, Verification of Interactive Systems (DSV-IS'95), Springer-Verlag, 1995, pp. 7-19.

[17] I. Morrey, J. Siddiqi, R. Hibberd, G. Buckberry. “A Toolset to Support the Construction and Animation of Formal Specifications”, Journal of Systems and Software, 41(3), June 1998, pp. 147-160.
[18] E. Kazmierczak, P. W. Dart, L. Sterling, M. Winikoff. “Verifying Requirements Through Mathematical Modeling and Animation”, International Journal of Software Engineering and Knowledge Engineering, 10(2), 2000, pp. 409-439.
[19] R.A. Virzi, J.L. Sokolov, and D.Karis, “Usability problem identification using both low- and high-fidelity prototypes,” in *Proc. SIGCHI Conf. Human factors in Computing Systems: Common Ground (CHI ‘96)*, M.J. Tauber, Ed. New York, NY: ACM 1996, pp. 236–243.
[20] R. Sefelin, M. Tscheligi, and V. Giller. “Paper prototyping –what is it good for? A comparison of paper- and computer-based low-fidelity prototyping,” in *Conf. Human Factors in Computing Systems CHI '03*, 2003, pp. 778–779.
[21] S. Stepney, F. Polack, I. Toyn. “An Outline Pattern Language for Z: Five Illustrations and Two Tables”. In: ZB2003. LNCS, vol. 2651, Springer, Heidelberg , 2003, pp. 2-19.
[22] X.H. J. Ding, L. Mo. “An approach for specification construction using property-preserving refinement patterns,” 23th Annual ACM Symposium on Applied Computing, ACM, 2008, pp.797–803.
[23] B.H.C.C. Sascha Konrad. “Real-time specification patterns”, 27th International Conference on Software Engineering, New York, ACM, 2005, pp.372–381.
[24] E. Durr, J. Katwijk. “VDM++: A Formal Specification Language for Object Oriented Designs”, International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Europe92), Prentice Hall, 1992, pp. 63-78.
[25] J. Warmer, A. Kleppe. “The Object Constraint Language: Getting Your Models Ready for MDA”, Person Education, Inc., 2003.
[26] H. Hussmann, B. Demuth, F. Finger. “Modular Architecture for a Toolset Supporting OCL”, Science of Computer Programming, 44(1), 2002, pp. 51-69.
[27] C. Snook, M. Butler. “UML-B: Formal Modelling and Design Aided by UM”, ACM Transactions on Software Engineering and Methodology (TOSEM), ACM Press, 15(1), 2006, pp. 92-122.
[28] A. Bryant, “Structured Methodologies and Formal Notations: Developing A Framework for Synthesis and Investigation,” *Proc. Z User Workshop*, Oxford 1989. Springer-Verlag, 1991.
[29] M.D. Fraser et al., “Informal and Formal Requirements Specification Languages: Bridging the Gap,” *IEEE Trans. Software Eng.*, vol. 17, no. 5, pp. 454–466, May 1991.
[30] Roger Duke, Gordon Rose. “Formal Object-Oriented Specification Using Object-Z”, MacMillan, 2000.
[31] Razen Diaconescu, Kokichi Futatsugi. “The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification”, Amast Series in Computing, Vol.6, World Scientific, 1998.
[32] Masahiro Nakano, Kazuhiro Ogata, Masaki Nakamura, Kokichi Futatsugi. “Creme: An Automatic Invariant Prover of Behavioral Specifications”, International

Journal of Software Engineering and Knowledge Engineering, 17(6): 783-804, World Scientific, 2007.
[33] 荒木啓二郎. “ソフトウェア開発現場への形式手法導入―形式手法適用の経験から得られた知見”, SEC Journal, 6(2), pp. 104-107, 2010.