Vol.2021-SE-207 No.5
2021/3/1

仕様記述言語SOFLを用いたガラス検索システムの開発と考察

木下 綾乃[†1,a)]　劉 少英[1,†1]

概要：本発表は，仕様記述言語 SOFL を一般的な商用システムに適用した際の効果について，検証・考察するものである．これまでの SOFL は徐々に普及しているものの，そのほとんどは安全性を重視したシステムの開発に対してである．そこで，これまであまり利用されてこなかった一般システム向けに改良された SOFL についてその有効性を確認するため，適用事例としてガラス検索システムの開発を行い，SOFL の記法に関する技術の習得から仕様作成・実装・テストという各工程ごとに作業時間や利点，課題点等を洗い出していく．その結果，習得から実際に適用可能になるまでのコストが少しかかってしまうという課題が見つかったが，その一方で SOFL の改良により仕様作成のコスト削減と品質の向上をバランスよく行えること，またテストケースの作成が容易であることといった利点があることが分かった．

Development and Consideration of A Glass Search System Using SOFL

1. はじめに

システム開発に広く用いられる開発モデルは，要件定義・仕様作成・設計・実装といった工程を踏んでいる．このようなモデルにおいて，ユーザが開発者に要求を伝えそれをプログラマーが実装していくというやり取りの中で，認識に誤差が生じ手戻りが発生してしまうことも少なくない．

SOFL は，形式仕様記述 [1] により曖昧さの少ない仕様を定義することで厳密な仕様定義と設計を可能にし，そのような手戻りやバグの発生を減らすことができる手法である．この技術はこれまで踏切の制御等の，安全性を重視するシステムに主に用いられてきた．

しかし，コストやリソースの制約の大きい一般的な商用システムの開発には，いまだ浸透していないという現状がある．そこで本研究では，SOFL をこれまでに使用されたことのない応用領域のシステム開発に適用する．その後，開発の各工程に関して, SOFL の活用効果について利点と課題等を分析していく．

[1] 情報処理学会
IPSJ, Chiyoda, Tokyo 101–0062, Japan
[†1] 現在，広島大学
Presently with Hiroshima Uniersity
[a)] b172484@hiroshima-u.ac.jp

2. SOFL の技術

SOFL（Structured Object-Oriented Formal Language）とは，Liu が提案し [2]，1998 年に基本設計を完成した [3] 構造化・オブジェクト指向・形式手法の利点を組み合わせた仕様記述言語である．形式仕様記述は数学的表現を用いた仕様記述法であり，自然言語による解釈の曖昧さを回避するとともに，論理学や離散数学を基礎としていることで仕様の整合性を保証している [4]．SOFL では，この表記法を用いて非形式仕様・半形式仕様・形式仕様の 3 段階に分けて仕様を詳細化し，徐々に設計に近い形へと変化させていく．また，形式仕様による記述に加えデータフローダイアグラムを設計し，グラフィカルな表記を利用したユーザとのコミュニケーションの強化を図っている．

2.1 SOFL の 3 段階仕様記述技術

SOFL の最も特徴的な技術が 3 段階仕様記述技術である．非形式仕様・半形式仕様・形式仕様の 3 段階を経て，徐々に機能の修正・追加を行いながら，より詳細な設計へと変化していく [5].

1) 非形式仕様では，自然言語を用いて機能の抽出を行う．まず，ユーザからの要求をヒアリングした結果を用い，要件を可能な限り取り込んでシステムの主要な機能を

列挙する．その後，それらの機能をいくつかの細かい機能にさらに分解していく．また，記述した機能を実現するためのいくつかのデータリソースも記述する必要がある．この段階では詳細なデータ構造や型について考慮する必要はないが，機能やデータに関して何らかの制約がある場合は仕様の最後に記述しておく．

2) 半形式仕様では，1 つの主要な機能に対して 1 つのモジュールを作成する．各モジュールには形式記述によるデータ型の定義と，いくつかのプロセスと呼ばれる機能の定義が含まれる．各プロセスには，それぞれの機能に対する入出力変数と，機能を実行する前と後の状態を表す事前条件・事後条件が記述されるが，半形式仕様の段階ではこの 2 つの条件を自然言語と入出力変数を用いて表す．

3) 最後に半形式仕様をもとにして，形式仕様を記述していく．半形式仕様を作成した際に自然言語で記述していた事前条件・事後条件を詳細化しながら形式手法の記述法に基づいて記述する．

2.2 一般システムに適用するための改良

大きな改良点は 2 点ある．

まず，3 段階仕様記述の工程に GUI 設計を組み込むことである．新たな手法ではこれまでの半形式仕様を作成する段階に GUI 設計を加え，非形式仕様・半形式仕様＋ GUI 設計・形式仕様を三段階仕様記述としている [6].

GUI 設計を取り入れることによって，文字による仕様だけではわからない部分の仕様を補うこと，またグラフィカルな情報があることによってユーザとのコミュニケーションを円滑にすること等の効果が見込める．

もう一つは，半形式仕様から形式仕様への詳細化を一部の機能にのみ行うことである．これまでの SOFL は主に安全性を重視したシステムに利用されるものであったためリソースの制限が少なかったが，一般システムに適用する場合，これまで通りの完全な仕様をすべて完成させるのはコストが大きすぎる．そこで，形式仕様による詳細な仕様を必要とする複雑な機能にのみ形式仕様を作成することで，品質を保ちつつコストの削減を行うことを可能にしている．

3. ガラス検索システムの開発

本研究では，一般システムに SOFL を適用した際の効果について検証するため，防音ガラス検索システムの開発を行う．前提条件として，第一筆者はこれまで SOFL についての知識はなく，開発の準備段階として SOFL の習得を行う．

よって，研究全体としては，

(1) SOFL の習得
(2) 要件定義
(3) SOFL による仕様作成
(4) 実装
(5) テスト
(6) 考察

という流れで行う．そのため，技術の習得から開発のすべての段階についてを考察の対象とする．

3.1 防音ガラス

開発のテーマとしている防音ガラスは，顧客の希望に見合った商品をすぐに判断することは難しいとった課題を抱えている．それぞれの防音ガラスには，防音の性能を表す透過損失という指標が周波数ごとに存在し，この値の傾向によってどのような音に対して遮音性が高いのかを判断する．最適なガラスの提案が難しい原因としては，防音ガラスの種類が膨大なために人の手による作業で防音性能を比較し適したものを選び出すには時間がかかってしまうことなどが考えられる．

そこで，開発するシステムでは遮音したい騒音の種類を検索条件として入力することで，その騒音の周波数帯に対して透過損失の値が高いガラスを計算することとする．また，その他の機能として防音ガラスのデータ管理，騒音データの管理，システムの利用者のログイン用データの管理といったものを想定する．

3.2 仕様と GUI 作成

ここからは，開発の内容に入っていく． 仕様作成のフェーズでは，2 章で紹介した SOFL の 3 段階記述法に従って仕様を作成する．

1) まず，非形式仕様を作成する．本システムの主な機能は，ガラスの検索・一覧表示，防音ガラス・騒音データ・社員・顧客の情報管理等である．これらの主な機能を記述したのち，各機能をさらに詳細化して記述する．

ここで，実際に作成した非形式仕様を図 1 に示す．

2 行目に記述された「最適なガラスの検索」という機能に関して必要な機能をさらに詳細に分解する．この作業により，検索を行い結果を表示するまでの流れを 4 つの段階に細分化することができ，半形式仕様を作成するための基盤となった．図 1 に示されたガラスの検索機能以外の 5 つの機能に対しても同様の細分化を行う．

また，非形式仕様では可能な限りでリソースを記述しておく必要がある．非形式仕様の段階では，

- ガラスの性能データ
- 騒音の周波数データ
- 社員のアカウント情報
- 顧客のアカウント情報

といったデータリソースが必要であると予想されるため，これらを簡単に記載しておく．

システムの機能
(1) 最適なガラスの検索
(2) 商品のガラスの一覧表示
(3) ガラスの詳細データ表示
(4) 社員アカウントにログイン
(5) 顧客のアカウント情報管理
(6) 社員のアカウント情報管理

1.1.最適なガラスの検索
(1) 条件の選択ができる（騒音の優先順位）
(2) 検索の実行
(3) 候補を数種類出力
(4) グラフの出力

図 1 非形式仕様 (一部抜粋)

```
1  module Search-dec1
2    type
3      Glass = composed of
4                  id: nat
5                  classification: string
6                  name: string
7                  t_class: T_class
8                  3octave: 3Octave
9                  price: nat
10         end
11
12   process Search_Glass(first: Noise_Data|
13    second: Noise_Data|third: Noise_Data)
14     glass: Glass|e_msg: string
15     ext rd g_list
16     pre first が与えられている
17     post first ->
           lower から upper の間で平均値を計算
18          or
19          second ->
                first の結果の上位５つで同じ計算
20          or
21          third ->
                second の結果の上位３つで同じ計算
22          or
23          条件に一致しない場合は
                e_msg を出力
24   end-process;
25 end-module;
```

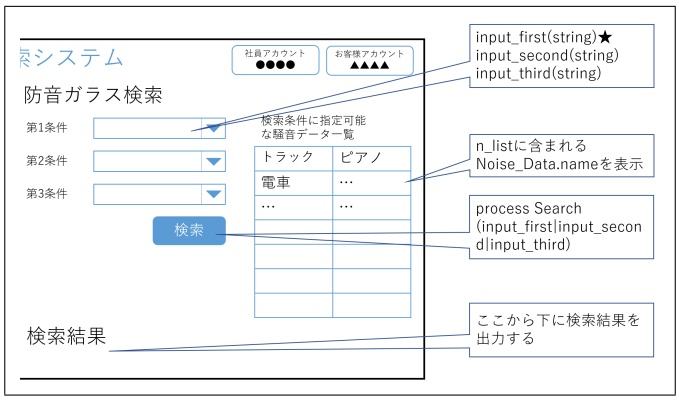

図 2 GUI 設計（ガラス検索画面）

2) 非形式仕様をもとに半形式仕様の記述と GUI 設計を行う．非形式仕様で記述した 6 つの主な機能に対応したモジュールを作成する．左のソースコードは検索機能の半形式仕様の一部を示したものである．
モジュール上部の type の項目には，ガラスと騒音のデータ型を形式仕様の記述に従って定義している．例えば，ガラスのデータは複数の変数を持ち，整数型の ID（id: nat と示される項目）のように記述される．
プロセスとして記述されているのは，検索機能の計算部分である． 事前条件（pre）として検索条件の一つ目の騒音データである first が与えられていること，また事後条件（post）として入力条件に対する防音性能の平均値を計算し，上位のガラスを検索結果とすることとしている． これらの条件については，図 2 に示すように自然言語で表現することとする．
図 3 には検索画面の GUI 設計案を示す．半形式仕様で記述したプロセス名と GUI のアクションを対応させながら設計を行う．この時，半形式仕様に記述していない新たな機能が必要であることや，データ型の変更が必要となった場合には，半形式仕様の追加や変更を行う．

3) 最後に半形式仕様の自然言語で記述された条件を形式記述で表した形式仕様を作成する．
従来の SOFL の場合，すべてのプロセスに対して詳細化を行っていたが，本研究では複雑な計算を必要とする検索モジュールのみ形式仕様を作成し，その他のモジュールに関しては半形式仕様を最終的な仕様としている．作成した検索機能の形式仕様に含まれている，検索プロセス（process Search_Glass）を次に示す．

```
pre bound(first) and
  if bound(second)
    then true
  else
       then not bound(third)
post forall[x: inds(g_list)]|
  ave1(x)=Sum(g_list, x, first, 1)/(
      first.upper_limit-first.lower_limit
      +1)
  max_ind1 = Max_Inds(ave1, 5, 1)
  and
  if bound(second)
    then
       forall[x: elems(max_ind1)]|ave2(x
           )=Sum(g_list, x, second,
           1)/(second.upper_limit-second
           .lower_limit+1)
       max_ind2 = Max_Inds(ave2, 3, 1)
       and
    if bound(third)
      then
      forall[x: elems(max_ind2)]|ave3(x
          )=Sum(g_list, x, third, 1)/(
          third.upper_limit-third.
          lower_limit+1)
      max = Max(ave3, 1)
      and
      glass = g_list(max)
        else glass = g_list(max_ind2(1))
  else glass = g_list(max_ind1(1))
```

事前条件は入力値である 3 つの騒音データに関するものであり，

- 第 1 条件
- 第 1 条件と第 2 条件
- 第 1 条件と第 2 条件と第 3 条件

のいずれかの組み合わせの場合のみ真であると判定する．半形式仕様の段階では，少なくとも第一条件のみ入力されていればプロセスを実行可能であるとしていたが，形式仕様の詳細化を行うことによってさらなる未然のエラーを防ぐことができている． 事後条件では，まずリスト内のすべてのガラスについて，1 つ目の検索条件である騒音の周波数帯に対して防音性能を表す透過損失の平均値を計算する．その後透過損失の平均値が高い上位 5 つのガラスを抽出し，2 つ目，3 つ目の条件に対しても同様の計算を行う．

ここで，SOFL による半形式仕様・形式仕様では，機能の定義であるプロセス以外に，関数を定義することが可能である．そのため，透過損失の高いガラスを抽出する条件式を事後条件の部分に直接記述するのではなく，Function Max として個別に定義している．事後条件が長くなってしまう場合や繰り返される処理に関しては，あらかじめ関数を定義しておくことで，実装がさらに容易になることが見込める．

3.3 実装

作成した半形式仕様・形式仕様をもとに，実装を行っていく．

SOFL によるモジュールは，必要なデータ型の定義と，関連する細かい機能が一つにあらわされているため，オブジェクト指向のクラスのような形になっている．それにより，ひとつのモジュールに対してひとつのクラスを作成することで実装を行う．

必要なデータ型については，すべて仕様に型定義を行っているため，すべての型についてクラスの作成を行う．その際に，仕様に記述されていない処理が必要となる場合は適宜補う必要があるが，この時以下の作業を行う．

- プロセスの記述が必要なレベルの重要な機能である場合，半形式仕様または形式仕様に追加する．
- プロセスではなく関数レベルの定義で十分な小さな機能の場合，半形式仕様または形式仕様には追加せず実装のみを行う．

これにより，テストの工程においてより正確なテストケースの作成が可能となると同時に，すべての処理を記述する必要がないため効率的な作業が行える．

3.4 テスト

テストケースの生成は，作成した半形式仕様・形式仕様をもとに行う．仕様に記述した各プロセスの事前条件から，入力値のテストパターンを作成する．ここで，

```
process Test(X: nat, Y: nat) Z: nat
  pre X<5 and bound(Y)
  post (X<Y and Z=X+Y) or (X>=Y and Z=X-Y)
end-process;
```

というプロセス（Test）に関する事前条件と事後条件を用いて表 1 に例を示す．

表 1 入力値のテストパターン例

	X<5：true	X<5：false
X<Y：true	X=3, Y=4, Z=7	X=6, Y=7
X<Y：false	X=3, Y=2, Z=1	X=6, Y=5

事前条件と事後条件に含まれる X，Y に関する条件式の真偽を組み合わせると表 1 のような入力のパターンが考えられる．

このとき，X=3 である 2 つの入力のパターンについては，事後条件に従い，X=3，Y=4 の場合は Z=7 となり，X=3，Y=2 の場合は Z=1 となると予想される．しかし，X=6 である右列のパターンは，事前条件に反するためプロ

セスは実行されず Z は出力されない．また，bound(Y) という条件については Y に null でない有効な値が与えられていることを示すものであるため，Y に入力を与えないテストパターンをさらに追加し，プロセスが実行されないことを確認する必要がある．自然言語による条件に対しても同様にして，事前条件と事後条件で示している条件式の真偽を用いてテストケースを作成する．

また，仕様による入力パターン以外に，GUI 設計からもテストケースを作成する．今回のガラス検索システムについては，ボタンの動作や画面遷移，入力値の制限（文字数，数値，文字列）といったテスト項目は GUI 設計をもとにして作成している．

このようにして，すべての必要な項目に対してテストパターンを作成すると，テストケースは 105 件となった．作成したケースに従いテストを行った結果，105 件のうちバグの件数は 22 件であった．

4. 考察

以上のように，防音ガラスの検索システムを開発した結果を用いて，SOFL を一般システムに適用する効果について考察する．

考察の方法としては，開発にかかった時間やテストのデータといった情報や 実際に開発を行った使用感をもとに，SOFL の利点と課題点を洗い出し，課題の解決策の提案等を行う．

4.1 仕様作成までの工程に関する考察

SOFL の習得から仕様作成に関して，注目するべき点は 4 点あげられる．

第一に，SOFL の利点として前後の条件のみに焦点を絞って機能の定義が行えることがあげられる．機能の詳細な内部構造について考える必要なく機能の定義ができることで，仕様作成者のシステムの構造に対する理解を助けると同時に，詳細化をする各段階で最も重要な部分にだけ注目して考えられるわかりやすさがあった．

第二に，SOFL で用いる条件の記述の仕方は，これまでのプログラミングの経験からはあまり身につかない記法であるため，習得にコストがかかるという課題があげられる．そのため, 実用可能なレベルまで習得し，既存の開発プロセスに統合するためには時間がかかってしまうことが懸念される．しかし，SOFL の記述法の難しさは経験により解決することができるものであるから，使いこなすことができるようになれば品質の高いシステム開発が可能となり，結果的にコスト削減が見込めると考えられる．

第三に，2.2 節で述べた改良点により，これまでの SOFL と比較して大幅に時間的コストが削減できるという点である．本研究では，半形式仕様から形式仕様への詳細化を検索機能にのみ行った．その結果を用いて，仕様の記述にかかる時間がどれくらい削減できたかということを半形式仕様と形式仕様の行数をもとに比較する．

検索機能以外の半形式仕様のうち，事前条件・事後条件の行数は 53 行であり，これらの単純な条件はそれぞれ形式仕様に書き換えた際に 1 行で記述されると仮定する．検索機能については，事前条件・事後条件を合わせて 73 行である．このことから，すべての条件に対して形式記述による詳細化を行った場合は 126 行であるところを，検索機能の 73 行のみ記述したことで，約 42%の作業時間を削減できたといえる．

ただし，本システムはデータの操作と参照を主な機能とするものであったため，条件が単純で形式仕様への詳細化が少なかった．そのため改良の効果が大きく出て大幅なコストの節約につながったが，オンライン上で他のサービスとの通信が必要なものやもっと複雑な機能をもつシステムの場合は大きく違った結果がみられる可能性がある．今後他のシステムに対しても多くの適用例を集められれば，各システムの特性ごとに SOFL の利用に適したものとそうでないものを分析することも可能であろう．

その他の仕様作成に関する留意事項としては，テストケースの作成にかかわる内容でもあるが，形式仕様を作成するかどうかの判断基準として，半形式仕様に記述された自然言語による条件から十分なテストケースが作成できるかという点が一つのポイントとしてあげられる．

4.2 実装に関する考察

続いて実装時に関しては，まず第一に，モジュールとそれに含まれるプロセスという仕様の形態が，クラスとメソッドにうまく対応することで容易に実装が行えるという点が最大の利点と言える．モジュールに従って実装を行っていくことで，あらかじめ与えられた条件により自然と整合性の取れたシステム開発をおこなうことが可能である．

続いて，通常の開発プロジェクトでは，仕様作成者とプログラマは異なることがほとんどであるが，SOFL を用いた開発の場合は仕様作成者がプログラミングにそのまま携わるのが望ましいと考えられる．その理由として，SOFL による三段階仕様記述技術を用いることで仕様を作成しながらシステムの構造や機能に対する理解を深める効果があるため，よりその効果を活かすことができるからである．通常の開発についても共通することであるが，システムに対してより深く理解できている状態にもかかわらず新たな開発者に引き継いで実装を行うことは非効率であると言える．

実装を行っていく中で，仕様に記述されていない機能が必要となったときは，3.3 節で示したような処理を行う必要がある．プロセスとして記述する必要のある機能に関しては，仕様の改善を行っておくことで，テストの自動生成にも対応することができる．

4.3 テストに関する考察

まず，新たに取り入れた 2.2 節の改良点に着目する．本研究では仕様作成の段階で，詳細な形式仕様は検索機能のみ記述している．4.1 節より時間的コストの削減は十分に行えていることを示したが，テストを行った際に，すべてのバグのうち仕様に関するバグの割合がわずか 5%であったことから品質が十分保たれていると言える．そのため，新たな改良を取り入れることによって，一般システムに適したコストと品質のバランスが取れていると考えられる．

その他のバグに関して，この開発では，テストケース 105 件中 24 件とバグ数が少し多い結果となった．原因として考えられるのは，第一筆者の開発経験の不足により予期せぬバグが内在していたことが大きいと考えられるが，一方で多くのパターンを網羅することができており，効果的なテストが行えていたといえるだろう．

また，SOFL にはテストケースの自動生成を行う技術がある [7]．今回の開発についてはテストの自動生成ツールを用いてはいないが，実装時に不足するプロセスの記述を適切に追加していれば，より効果的にツールを活用し有効なテストを行うことも可能であると予想される．

5. おわりに

本研究では，SOFL による仕様定義の一般的な商用システムに対する効果を検証した．ガラス検索システムの開発を通して，SOFL の習得から三段階仕様記述法を用いた仕様定義，実装とテストについての考察を行った．

一般システム向けの改良点として一部の複雑な機能にのみ詳細化を行ったが，コストの削減と品質の保持が適正なバランスで可能になっていることが分かった．SOFL 仕様からプログラム作成やテストケースの生成を行うことは，経験を積むごとにその効率を増しよりよい効果を得られる．よって，継続してこの技術を用いていかに活かすことができるかが重要である．

また，本研究で開発したようなデータベースと接続しデータの操作や参照を行う機能が多いシステムには，改良を行った SOFL の活用が非常に適していた．しかし，より複雑な機能を持っていたりオンライン上でのやり取りが必要なものは同様の効果を見られるかどうか確かではない．今後は，より多くのシステム開発に新たな SOFL を適用し，異なる条件下での検証も行う必要があるだろう．

参考文献

[1] 中島震. 形式手法入門：ロジックによるソフトウェア開発. オーム社, 2012.
[2] S. Liu and Y. Sun. Structured methodology + object-oriented methodology + formal methods: Methodology of sofl. *IEEE Press*, Vol. 6, No. 10, pp. 137–144, 1995.
[3] S. Liual. Sofl: A formal engineering methodology for industrial applications. *IEEE Transactions on Software Engineering*, Vol. 24, No. 1, pp. 24–45, 1998.
[4] 赤間世紀. 形式手法教科書：「論理学」を用いた、「ソフト工学」への数学的アプローチ. 工学社, 2012.
[5] Shaoying Liu. *Formal Engineering for Industrial Software Development using the SOFL Method*. Springer-Verlag. ISBN 3-540-20602-7.
[6] Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia. *Agile Formal Engineering Method for Software Productivity and Reliability*, 2018.
[7] Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia. *A 'Vibration' Method for Automatically Generating Test Cases Based on Formal Specifications*, 2018.