さまざまな視点に合わせた仕様書の作成・維持の支援手法

Support method for preparation and maintenance of specifications tailored to various perspectives

酒井 雄太（キヤノンアイテック株式会社）

研究概要

業務システムの開発においては，技術背景の異なるステークホルダーに対して，仕様に関する情報をさまざまな視点や抽象度で表現し，議論や妥当性確認を促す必要がある．そのような場面において，仕様書は，通常，技術者が作成し，仕様に関する情報を仕様記述として集約したものとなるが，例えば，技術者ではない顧客には読みにくいといった課題や，読み手のさまざまな視点に合わせた複数の仕様記述間に不整合が生じるといった課題がある．そこで本研究では，形式手法のひとつである VDM (Vienna Development Method) によって仕様記述を行い，その仕様記述から機械処理によって，読み手のさまざまな視点に合わせた仕様記述を生成することにより，仕様書の作成・維持の支援手法を提案する．本提案手法による仕様書は，複数の仕様記述間の不整合を防ぎつつ，技術者ではない読み手にも読みやすいという特徴を持つ．

Abstract In the development of business systems, it is necessary to represent the contents of the specification in a variety of perspectives and levels of abstraction to promote discussion and validation by stakeholders with different technical backgrounds. A specification document, a description aggregating the specification contents, is usually created by engineers and thus leads to issues such as difficulty for non-technical customers to read and inconsistency between specification descriptions prepared for various viewpoints of readers. In response to these issues, this work proposes a method for supporting construction and maintenance of specification documents. In the method, a specification description is constructed by using VDM (Vienna Development Method), one of formal methods, and then descriptions for various viewpoints of readers are automatically generated. This method leads to specification documents that prevent inconsistency between multiple descriptions and are comprehensible for non-technical readers.

1. はじめに

1.1 業務システムの開発

著者は，所属する組織において，業務システムの開発に携わっている．業務システムの開発においては，技術背景の異なるステークホルダーに対して，業務システムの機能や画面といった，仕様に関する情報をさまざまな視点や抽象度で表現し，議論や妥当性確認を促す必要がある．例えば，現場担当者においては，現場で業務システムをどのように使うかということについて興味があるので，業務システムの基本的な使い方の視点で表現した仕様を示すことが，議論や妥当性確認を促すことにつながる．

1.2 業務システムの開発における仕様書の課題

技術背景の異なるステークホルダーに対して議論や妥当性確認を促す必要がある場面において，技術者と技術者ではない読み手とのコミュニケーションのために，仕様書が用いられる．仕様書は，通常，技術者が作成・維持し，仕様に関する情報を仕様

記述（仕様をある形式で表現したもの）として集約したものである．

このような仕様書の課題のひとつは，技術者ではない読み手には読みにくいということである．例えば，読み手が現場担当者である場合，以下に挙げるような問題が生じうる．

(1) 読み手は，どの記述を読むべきか分からないため，仕様書の全ての記述に向き合う必要があり，限られた時間内に理解することができなかったり，レビューを完了させたりすることができない
(2) 読み手は，具体的な動作を理解したいとき，複数の記述を横断的に読む必要がある
(3) 読み手は，内部のロジックや外部の業務管理システムとの連携についての記載については，関心がない
(4) 読み手は，普段行なっている業務がどのように変化するのかを想像することができず，業務が楽になるのか分からない

上記の問題を扱うために，読み手に合わせた仕様記述を用意することが多い．しかし，読み手のさまざまな視点に合わせた複数の仕様記述間に不整合が生じるという課題がある．不整合が生じる原因として，ある仕様記述から，視点や抽象度を変えた仕様記述を作成する際に，欠陥が混入することや，ある仕様記述を更新したときに，その仕様記述から作成されている仕様記述が更新されないことなどが挙げられる．

2. 本研究の提案

2.1 さまざまな視点に合わせた仕様書の作成・維持の支援手法

そこで本研究では，仕様に関する一通りの情報が集約されたものとして，形式手法のひとつである VDM (Vienna Development Method) [1] によって形式仕様記述[2]を行い，その仕様記述から機械処理によって，読み手のさまざまな視点に合わせた仕様記述を生成することによる，仕様書の作成・維持の支援手法を提案する．VDM においては，仕様記述言語によりシステムのモデルを厳密に記述する．本研究では，VDM による厳密な仕様記述が機械処理可能であることを活かし，その記述から状態遷移図やシーケンス図といった表現形式の仕様記述を生成する手法を提案する．

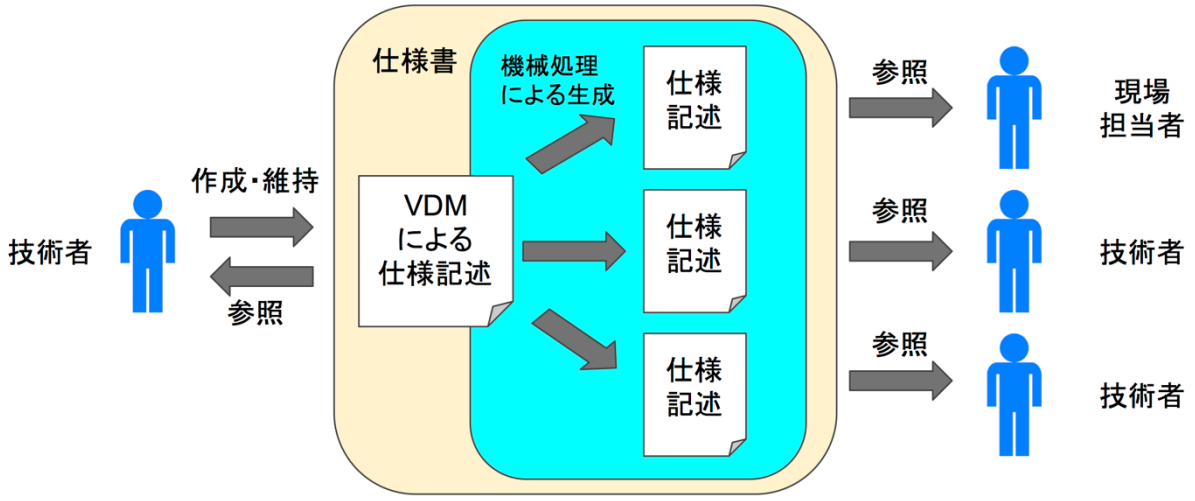

図 1 提案手法の概念

図 1 にて，本提案手法の概念を説明する．VDM による仕様記述は，技術者により作成・維持され，仕様に関する一通りの情報が集約されたものである．これは，さま

ざまな視点で読むことができるが，技術者ではない読み手には読みにくいものである．そこで，VDM による仕様記述から，現場担当者や技術者といった読み手の視点に合わせて，機械処理による仕様記述の生成を行う．

本提案手法により作成・維持される仕様書は，複数の仕様記述間の不整合を防ぐことができたり，技術者ではない読み手にも読みやすかったりする．なぜならば，ある仕様記述から機械処理によって生成された仕様記述は，欠陥が混入する可能性が低いし，機械処理を自動化することによって，常に整合性を維持することができるからである．さらに，仕様書に，読み手や状況に応じた仕様記述が存在し，読むべき仕様記述を取捨選択することが容易だからである．

なお，仕様に関する一通りの情報には，VDM によって記述される機能仕様やユースケース記述などに加えて，図による画面イメージが含まれると考えるが，本研究においては，VDM による部分に絞って検討する．

また，機械処理により生成された仕様記述の修正は，VDM による仕様記述を修正することで実現する．このような手順とする背景として，通常，機械処理により生成された仕様記述は，VDM による仕様記述に比べ情報量が少ないものとなり，情報量の少ない仕様記述の変更から，情報量の多い仕様記述の変更を一意に決めることが困難だということがある．具体的な運用として，レビューにおいて，参加者は機械処理により生成された仕様記述に対して赤入れを行い，技術者はその結果と一致するように，VDM による仕様記述を修正するといったことが考えられる．

2.2 本研究の実装

本研究では，本提案の第一歩として，以下の提案手法に基づいた仕様記述生成機能を実装する．

(1) 現場担当者向けの仕様記述（シーケンス図）の生成機能
(2) 技術者向けの仕様記述（画面遷移図とシーケンス図）の生成機能

これらの仕様記述生成機能によって生成された仕様記述は，現場担当者や技術者が，業務システムについてある視点から理解することを促す．表 1 に，視点の具体例を挙げる．

表 1 本研究で生成の仕組みを検討する仕様記述とその視点

仕様記述	図の種類	視点
現場担当者向けの仕様記述	シーケンス図	業務システムの基本的な使い方を理解する
技術者向けの仕様記述	画面遷移図	画面遷移に関する仕様を俯瞰し，全体像を理解する
	シーケンス図	業務システムの具体的な動作の一例を通して，仕様の理解を促す

3 例題を用いた提案手法の説明

3.1 VDM による仕様記述

本提案手法を構築，試行するにあたり，機器のメンテナンス業務を管理するシステムを例題として設定し，例題についての VDM による仕様記述を行なった．本研究では，仕様記述言語として VDM++言語を採用した．また，機械処理による仕様記述の生成を実施可能とするために，表 2 の要求事項を満たすような記述とした．

表 2 本提案手法の VDM による仕様記述に対する要求事項

要求	定めたルール
各画面に関する仕様が記述されており，画面毎にまとまった形で容易に抽出できること	各画面を，1 つの VDM++クラスで記述する
各 UI 要素に関する仕様が記述されており，画面に紐づく形で容易に抽出できること	各 UI 要素を，画面について記述した VDM++クラスのインスタンス変数で表現する
画面遷移に関する仕様が記述されており，遷移元と遷移先の画面名が容易に抽出できること	画面遷移を，画面について記述した VDM++クラスの操作の戻り値にて，遷移先の画面のインスタンスを返すことで表現する
基本的な使い方が記述されていて，また，それを実行したときの内部的なことも含めた振る舞いについての情報を抽出できること	画面に対する一連の操作の一例を記述する
	全ての操作に対して，ログ出力を記述する

これらを踏まえた，具体的な仕様記述の例を以下に示す．前述の要求事項との対応は，仕様記述中のコメントとして示した．

```
-- このクラスは ログイン画面 についての仕様記述である
class ログイン画面 is subclass of 画面
instance variables
-- UI 要素をインスタンス変数として表現する
public ID フィールド : token := mk_token("NULL");
public パスワードフィールド : token := mk_token("NULL");
    〈省略〉
--画面遷移を伴う操作の戻り値は 次の画面のインスタンスである
public ログインボタンを押下する : () ==> 画面
ログインボタンを押下する() ==
    (
        --操作に対して，VDM++を実行した際に出力されるログが埋め込まれている
        io.print("ログイン画面.ログインボタンを押下する¥n");
        let
            sys = システム`システムインスタンス,
            認証情報確認結果 = sys.認証情報を確認する(ID フィールド, パスワードフィールド)
        in
            if 認証情報確認結果 = true
            then
                (
                    io.print("ログイン画面.タスク一覧画面に遷移する¥n");
                    return new タスク一覧画面();
                )
            else return self
    )
    〈省略〉
end ログイン画面
```

図 2 ログイン画面の VDM による仕様記述の例

```
public 基本シナリオを実行する : () ==> ()
基本シナリオを実行する() ==
    (
        io.print("ユースケース.基本シナリオを実行する¥n");
        現ユーザ.起動する();
        現ユーザ.ログインする(mk_token("Login"), mk_token("Password"));
        現ユーザ.タスクを選択する(mk_token("顧客0001"));
        現ユーザ.タスク情報を取得する();
        現ユーザ.作業内容を登録する(mk_token("Work1"));
        現ユーザ.交換部品を登録する(mk_token("P0001"), mk_token("Parts1"));

        現ユーザ.作業内容を取得する();
        現ユーザ.交換部品を取得する();
        現ユーザ.作業結果を送信する();
    )
```

図 3 画面に対する一連の操作を VDM により記述した例

3.2 機械処理による仕様記述の生成

次に，先に作成した VDM による仕様記述から，機械処理による仕様記述の生成を行なった．本研究では，現場担当者向けの仕様記述（シーケンス図）と技術者向けの仕様記述（画面遷移図とシーケンス図）が対象である．

3.2.1 シーケンス図の生成

本研究のシーケンス図は，VDM による仕様記述を実行することによる出力（実行ログ）に対してシーケンス図への変換処理を行うことで得られる．本研究では，VDM による仕様記述の実行に Overture Tool [3] を利用した．また，シーケンス図への変換処理は，実行ログから，クラス間の操作呼び出しログを抽出し，さまざまな図の生成ツールである PlantUML [4] が解釈可能な形式のファイルを生成する手段と，そのファイルを入力としてツールを実行し，画像データとしてシーケンス図を得る手段で構成される．さらに，この変換処理は，読み手に合わせて，シーケンス図に掲載する情報を取捨選択することができる．例えば，現場担当者はシステムに関する情報は不要であるので，操作呼び出しログの抽出処理において，システムクラスの操作呼び出しログは除外する，といったことである．これらにより，生成されたシーケンス図の例を図 4 に示す．

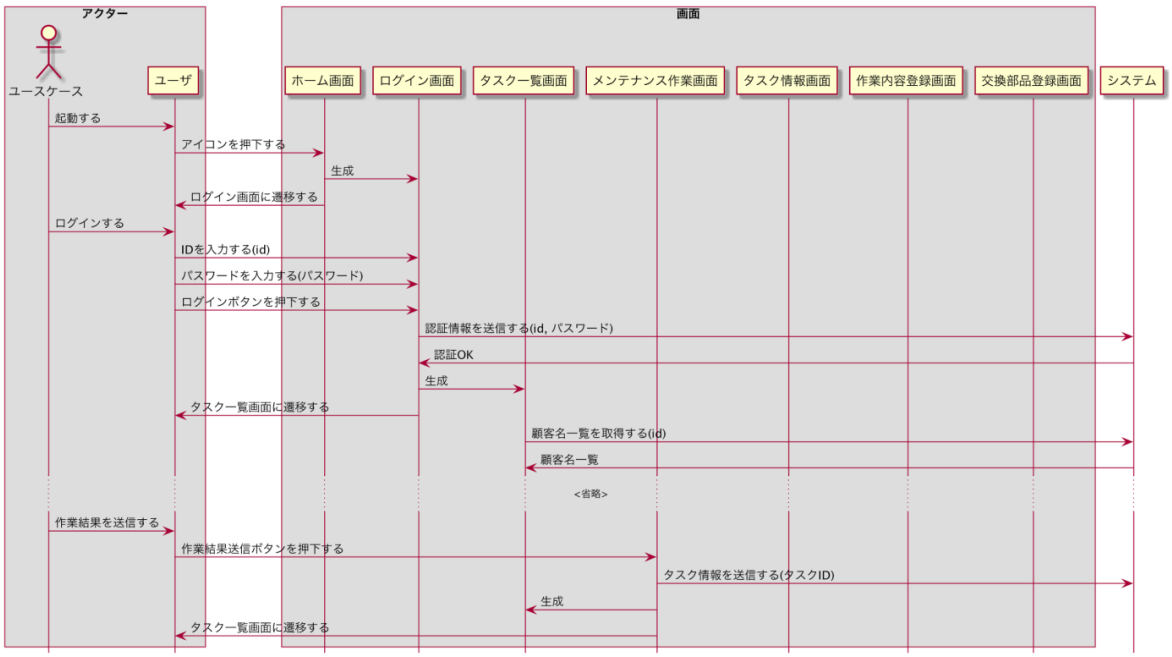

図 4 機械処理により生成された仕様記述(シーケンス図)の例

3.2.2 画面遷移図の生成

本研究の画面遷移図は，VDM による仕様記述に対して変換処理を行うことで得られる．画面遷移図への変換処理は，VDM による仕様記述のうち，画面の仕様にあたるものを抽出し，さらに，それらに記述されている UI 要素と画面遷移に関する情報を抽出した上で，ツールで解釈可能な形式のファイルを生成する手段と，そのファイルを入力として PlantUML を実行し，画像データとして画面遷移図を得る手段で構成される．なお，画面遷移図の生成は，アクティビティ図生成機能を利用した．これらにより，生成された画面遷移図の例を図 5 に示す．

この画面遷移図からは，業務システムの画面遷移に関する仕様を俯瞰し，全体像を理解することができる．例えば，本研究の実装では，画面毎に仕様記述がされているが，画面遷移の仕様を理解するためには，それらの仕様記述を横断的に読む必要がある．本図ならば，画面遷移に関する仕様がひとつの図にまとまっているので，理解することが容易である．

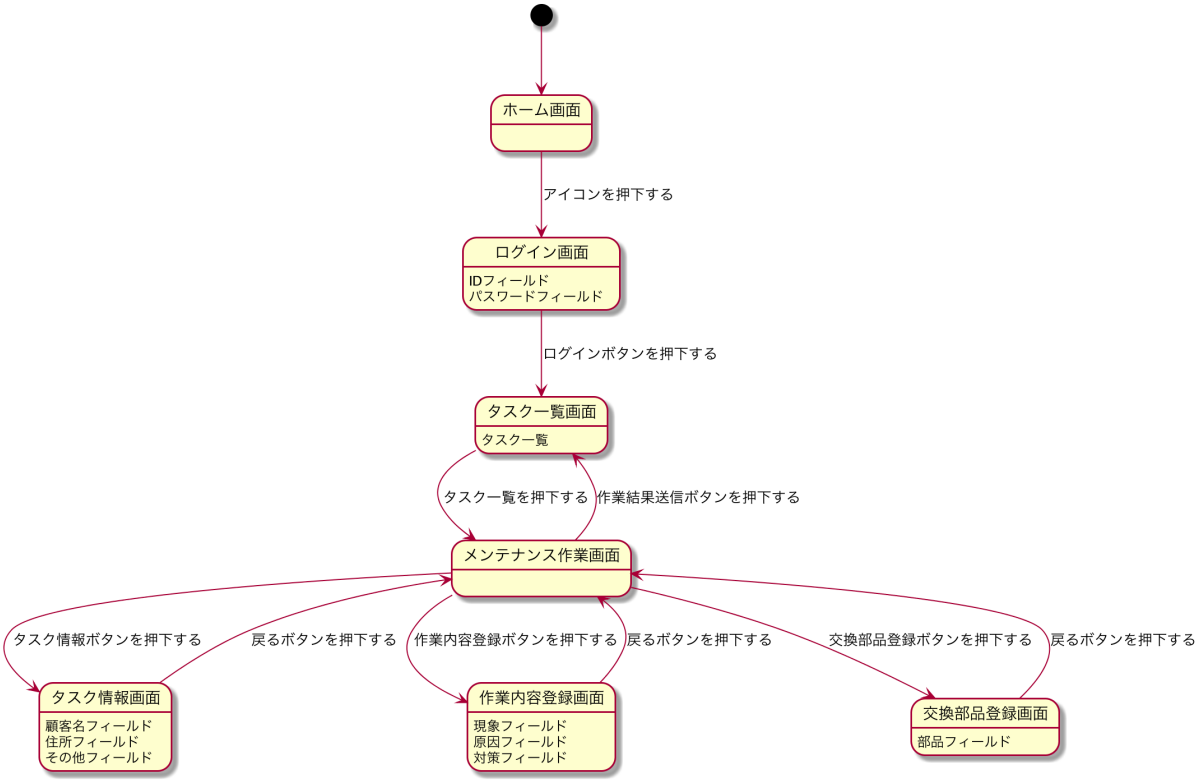

図 5 機械処理により生成された仕様記述(画面遷移図)の例

4. 評価と考察

4.1 生成機能について

本研究の実装による仕様書生成機能は，本提案の第一歩となるものであり，本提案で期待する初期の効果を確認することができたと言える．具体的には，VDM による仕様記述から，機械処理により仕様記述（シーケンス図と画面遷移図）を生成することで，複数の仕様記述間の不整合を防ぐことができた．また，シーケンス図では，業務システムの基本的な使い方を表現することができた．普段の業務では，さらに，図による画面イメージといった情報を含めて，操作マニュアルのような形で示すこともあるが，後述する提案手法の拡張によって，それは実現可能だと言える．

なお，本仕様書生成機能は，500 行程度の少ない量の仕様記述で，さまざまな視点の仕様記述を機械処理よって生成することができた．

4.2 提案手法について

本提案手法の目指すことは，読み手のさまざまな視点に合わせた仕様記述を生成し，かつ，複数の仕様記述間の不整合を防ぐことである．そこにたどり着くためには，本提案手法を拡張することや，本提案手法を開発プロセスに適用するといったことが必要であると考える．

本提案手法の今後の拡張として，例えば，本研究で作成した VDM による仕様記述であっても，「システムが持つインターフェースの一覧」のような仕様記述は簡単な拡張で実現できると考えられる．また，前述した VDM による仕様記述の要求事項を拡張し，本研究では扱わなかった図による画面イメージと組み合わせて，静的プロトタイプ[5]のような仕様記述を生成することや，業務に関することがらを含めた VDM による仕様記述を作成して，業務フロー図[6]のような仕様記述を生成することの実現方法は容易に考えられる．

さらに，対象とするシステムを増やすことも可能だと考えられる．本研究では典型的な業務システムを対象にしたので，VDM による仕様記述に対する要求事項として，画面に関する仕様などを設けたが，組み込みシステムといった，別のシステムを対象とする場合は，前述のものとは異なる要求事項を定める必要がある．しかし，本提案手法の考え方は，別のシステムを対象とする場合にも活用できる．

本提案手法を開発プロセスに適用するためには，ステークホルダーに応じた視点の選定すること，視点に合わせた仕様記述を生成するための手法の検討と実装を行うこと，VDM による仕様記述を更新したら，機械処理による仕様記述を再生成するといった実際の運用についてのこと，読み手からのフィードバックに基づき提案手法を改善するといったことを定める必要がある．

5. おわりに

本研究では，形式手法のひとつである VDM によって仕様記述を行い，その仕様記述から機械処理によって，ステークホルダーのさまざまな視点に合わせた仕様記述を生成することにより，仕様書の作成・維持作業の支援手法を提案した．

本研究では，機械処理によって画面遷移図とシーケンス図を生成する，本提案手法に基づいた仕様記述作成機能の実装を行なった．これらの仕様記述は，技術者や現場担当者に役立つ視点であると考えた．本研究の実装は，本提案手法の第一歩となるものであるが，その特徴である，複数の仕様記述間の不整合を防ぎつつ，技術者ではない読み手にも読みやすいということを示し，また，今後の拡張の可能性を示すことができた．

今後は，本提案手法の拡張によって，つまり，多様なステークホルダーとのより深い議論や妥当性確認のための仕様書を生成できるようにすることで，要求獲得活動の品質向上につなげたい．

6. 今後の展望

下記に，本研究における今後の展望を列挙する．

- よりさまざまな仕様記述への生成機能を構築
- 他のシステムへの適用を通した手法の洗練
- 技術者ではない読み手による評価

謝辞

本論文の執筆に当たり，演習コース II アドバイザーの九州大学 大学院 荒木啓二先生，主査の栗田太郎氏，副主査の石川冬樹氏，研究員の宮本陽子氏，日本科学技術連盟の事務局の皆様に大変お世話になりました．厚くお礼申し上げます．

参考文献

[1] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, and Marcel Verhoef, 酒匂 寛(訳), VDM++によるオブジェクト指向システムの高品質設計と検証 , 翔泳社 2010.
[2] 荒木啓二郎，張漢明：プログラム仕様記述論，オーム社，2002.
[3] Overture Tool http://overturetool.org/
[4] PlantUML http://plantuml.com/
[5] 川西 裕幸，栗山 進，潮田 浩，UX デザイン入門，日経 BP 社 2012
[6] 高安 厚思，システム設計の謎を解く，ソフトバンク クリエイティブ株式会社 2013

付録

本稿の 3 章で示した，VDM による仕様記述と，機械処理により生成された仕様記述(シーケンス図と画面遷移図)について，作成したファイルを付録として掲載する．なお，使用したライブラリに関する記載は省略する．

- ファイル一覧
- ユースケース.vdmpp
- ユーザ.vdmpp
- 画面.vdmpp
- ホーム画面.vdmpp
- ログイン画面.vdmpp
- タスク一覧画面.vdmpp
- メンテナンス作業画面.vdmpp
- タスク情報画面.vdmpp
- 作業内容登録画面.vdmpp
- 交換部品登録画面.vdmpp
- シーケンス図.svg
- 画面遷移図.svg

```
class ユースケース
instance variables
現ユーザ : ユーザ := new ユーザ();
io : IO := new IO();

operations
public 基本シナリオを実行する : () ==> ()
基本シナリオを実行する() ==
    (
        io.print("ユースケース.基本シナリオを実行する¥n");
        現ユーザ.起動する();
        現ユーザ.ログインする(mk_token("Login"), mk_token("Password"));
        現ユーザ.タスクを選択する(mk_token("顧客 0001"));
        現ユーザ.タスク情報を取得する();
        現ユーザ.作業内容を登録する(mk_token("Work1"));
        現ユーザ.交換部品を登録する(mk_token("P0001"), mk_token("Parts1"));

        現ユーザ.作業内容を取得する();
        現ユーザ.交換部品を取得する();

        現ユーザ.作業結果を送信する();
    )
post
    (
        現ユーザ.現顧客名 = mk_token("顧客 0002")
        and
        現ユーザ.現現象情報 = mk_token("現象 2")
    );
end ユースケース
```

図 6 ユースケース.vdmpp

```
class ユーザ
types
public ID型 = token;
public パスワード型 = token;
public タスク名型 = token;

values
instance variables
public 現画面 : 画面 := new ホーム画面();
public 現顧客名 : token := mk_token("NULL");
public 現顧客住所 : token := mk_token("NULL");
public 現その他情報 : token := mk_token("NULL");
public 現現象情報 : token := mk_token("NULL");
public 現原因情報 : token := mk_token("NULL");
public 現対策情報 : token := mk_token("NULL");
public 現交換部品集合 : set of 交換部品登録画面`部品フィールド型 := {};
io : IO := new IO();

operations
public 起動する : () ==> ()
起動する() ==
    (
        io.print("ユーザ.起動する¥n");
        現画面 := narrow_(現画面, ホーム画面).アイコンを押下する();
    )
pre isofclass(ホーム画面, 現画面)
post isofclass(ログイン画面, 現画面);

public ログインする : ID型 * パスワード型 ==> ()
ログインする(id, パスワード) ==
    let s = narrow_(現画面, ログイン画面)
    in (
        io.print("ユーザ.ログインする¥n");
        s.IDを入力する(id);
        s.パスワードを入力する(パスワード);
        現画面 := s.ログインボタンを押下する();
    )
pre isofclass(ログイン画面, 現画面)
post isofclass(タスク一覧画面, 現画面)
     or
     isofclass(ログイン画面, 現画面);

public タスクを選択する : タスク名型 ==>  ()
タスクを選択する(タスク名) ==
    (
        io.print("ユーザ.タスクを選択する¥n");
        現画面 := narrow_(現画面, タスク一覧画面).タスク一覧を押下する(タスク名);
    )
pre isofclass(タスク一覧画面, 現画面)
post
    (タスク一覧画面にタスク名が含まれている(現画面~, タスク名) and
        isofclass(メンテナンス作業画面, 現画面))
    or
    (not タスク一覧画面にタスク名が含まれている(現画面~, タスク名) and
        現画面 = 現画面~);

public タスク情報を取得する : () ==> ()
タスク情報を取得する() ==
    (
        io.print("ユーザ.タスク情報を取得する¥n");
        現画面 := narrow_(現画面, メンテナンス作業画面).タスク情報ボタンを押下する();
        現顧客名 := narrow_(現画面, タスク情報画面).顧客名フィールド;
        現顧客住所 := narrow_(現画面, タスク情報画面).住所フィールド;
        現その他情報 := narrow_(現画面, タスク情報画面).その他フィールド;
        現画面 := narrow_(現画面, タスク情報画面).戻るボタンを押下する();
    )
pre isofclass(メンテナンス作業画面, 現画面)
post sameclass(現画面, 現画面~);
```

図 7 ユーザ.vdmpp (前半)

```
public 作業内容を登録する : token ==> ()
作業内容を登録する(work) ==
    (
        io.print("ユーザ.作業内容を登録する¥n");
        現画面 := narrow_(現画面, メンテナンス作業画面).作業内容登録ボタンを押下する();
        let s = narrow_(現画面, 作業内容登録画面) in s.現象を入力する(work);
        let s = narrow_(現画面, 作業内容登録画面) in s.原因を入力する(work);
        let s = narrow_(現画面, 作業内容登録画面) in s.対策を入力する(work);
        現画面 := narrow_(現画面, 作業内容登録画面).戻るボタンを押下する();
    )
pre isofclass(メンテナンス作業画面, 現画面)
post sameclass(現画面, 現画面~);

public 作業内容を取得する : () ==> ()
作業内容を取得する() ==
    (
        io.print("ユーザ.作業内容を取得する¥n");
        現画面     := narrow_(現画面, メンテナンス作業画面).作業内容登録ボタンを押下する
();
        現現象情報 := narrow_(現画面, 作業内容登録画面).現象フィールド;
        現原因情報 := narrow_(現画面, 作業内容登録画面).原因フィールド;
        現対策情報 := narrow_(現画面, 作業内容登録画面).対策フィールド;
        現画面     := narrow_(現画面, 作業内容登録画面).戻るボタンを押下する();
    )
pre isofclass(メンテナンス作業画面, 現画面)
post sameclass(現画面, 現画面~);

public 交換部品を登録する : token * token ==> ()
交換部品を登録する(型番, 名称) ==
    (
        io.print("ユーザ.交換部品を登録する¥n");
        現画面 := narrow_(現画面, メンテナンス作業画面).交換部品登録ボタンを押下する();
        let s = narrow_(現画面, 交換部品登録画面)
        in s.交換部品を追加する(mk_交換部品登録画面`部品フィールド型(型番, 名称));
        現画面 := narrow_(現画面, 交換部品登録画面).戻るボタンを押下する();
    )
pre isofclass(メンテナンス作業画面, 現画面)
post sameclass(現画面, 現画面~);

public 交換部品を取得する : () ==> ()
交換部品を取得する() ==
    (
        io.print("ユーザ.交換部品を取得する¥n");
        現画面 := narrow_(現画面, メンテナンス作業画面).交換部品登録ボタンを押下する();
        現交換部品集合 := narrow_(現画面, 交換部品登録画面).部品フィールド集合;
        現画面 := narrow_(現画面, 交換部品登録画面).戻るボタンを押下する();
    )
pre isofclass(メンテナンス作業画面, 現画面)
post sameclass(現画面, 現画面~);

public 作業結果を送信する : () ==> ()
作業結果を送信する() ==
    (
        io.print("ユーザ.作業結果を送信する¥n");
        現画面 := narrow_(現画面, メンテナンス作業画面).作業結果送信ボタンを押下する();
    )
pre isofclass(メンテナンス作業画面, 現画面)
post isofclass(タスク一覧画面, 現画面);

functions
public タスク一覧画面にタスク名が含まれている : タスク一覧画面 * タスク名型 -> bool
タスク一覧画面にタスク名が含まれている(taskListScreen, taskName) ==
    taskName in set taskListScreen.タスク一覧テーブル;
end ユーザ
```

図 8 ユーザ.vdmpp (後半)

```
class 画面
instance variables
protected io : IO := new IO();
end 画面
```

図 9 画面.vdmpp

```
class ホーム画面 is subclass of 画面
operations
public アイコンを押下する : () ==> 画面
アイコンを押下する() ==
    (
        io.print("ホーム画面.アイコンを押下する¥n");
        io.print("ホーム画面.ログイン画面に遷移する¥n");
        return new ログイン画面();
    )
end ホーム画面
```

図 10 ホーム画面.vdmpp

```
class ログイン画面 is subclass of 画面
instance variables
public IDフィールド : token := mk_token("NULL");
public パスワードフィールド : token := mk_token("NULL");

operations
public IDを入力する : token ==> ()
IDを入力する(id) ==
    (
        io.print("ログイン画面.IDを入力する¥n");
        IDフィールド := id;
    );

public パスワードを入力する : token ==> ()
パスワードを入力する(pass) ==
    (
        io.print("ログイン画面.パスワードを入力する¥n");
        パスワードフィールド := pass;
    );

--画面遷移を伴う操作の戻り値は 次の画面のインスタンスである
public ログインボタンを押下する : () ==> 画面
ログインボタンを押下する() ==
    (
        io.print("ログイン画面.ログインボタンを押下する¥n");
        let
            sys = システム`システムインスタンス,
            認証情報確認結果 = sys.認証情報を確認する(IDフィールド, パスワードフィールド)
        in
            if 認証情報確認結果 = true
            then
                (
                    io.print("ログイン画面.タスク一覧画面に遷移する¥n");
                    return new タスク一覧画面();
                )
            else return self
    )
post
    let
        sys = システム`システムインスタンス,
        認証情報確認結果 = sys.認証情報を確認する(IDフィールド, パスワードフィールド)
    in
        -- IDとパスワードが正しければ, タスク一覧画面に遷移する
        (認証情報確認結果 = true and isofclass(タスク一覧画面, RESULT))
        or
        -- IDとパスワードが正しくなければ, 画面遷移しない
        (認証情報確認結果 = false and RESULT = self);

end ログイン画面
```

図 11 ログイン画面.vdmpp

```
class タスク一覧画面 is subclass of 画面
values
システム : システム = システム`システムインスタンス;

instance variables
顧客名一覧 : システム`顧客名一覧型
     := システム.顧客名一覧を返す(システム.現ユーザ ID);
public タスク一覧テーブル : set of token := rng 顧客名一覧;

operations
public タスク一覧を押下する : token ==> 画面
タスク一覧を押下する(タスク名) ==
     (
          io.print("タスク一覧画面.タスク一覧を押下する¥n");
          io.print("タスク一覧画面.メンテナンス作業画面に遷移する¥n");
          return new メンテナンス作業画面();
     )
post isofclass(メンテナンス作業画面, RESULT);
end タスク一覧画面
```

図 12 タスク一覧画面.vdmpp

```
class メンテナンス作業画面 is subclass of 画面
operations
public タスク情報ボタンを押下する : () ==> 画面
タスク情報ボタンを押下する() ==
     (
          io.print("メンテナンス作業画面.タスク情報ボタンを押下する¥n");
          io.print("メンテナンス作業画面.タスク情報画面に遷移する¥n");
          return new タスク情報画面();
     );

public 作業内容登録ボタンを押下する : () ==> 画面
作業内容登録ボタンを押下する() ==
     (
          io.print("メンテナンス作業画面.作業内容登録ボタンを押下する¥n");
          io.print("メンテナンス作業画面.作業内容登録画面に遷移する¥n");
          return new 作業内容登録画面();
     );

public 交換部品登録ボタンを押下する : () ==> 画面
交換部品登録ボタンを押下する() ==
     (
          io.print("メンテナンス作業画面.交換部品登録ボタンを押下する¥n");
          io.print("メンテナンス作業画面.交換部品登録画面に遷移する¥n");
          return new 交換部品登録画面();
     );

public 作業結果送信ボタンを押下する : () ==> 画面
作業結果送信ボタンを押下する() ==
     (
          io.print("メンテナンス作業画面.作業結果送信ボタンを押下する¥n");
          let システム = システム`システムインスタンス
          in システム.タスク情報を送信する(システム.現タスク ID);
          io.print("メンテナンス作業画面.タスク一覧画面に遷移する¥n");
          return new タスク一覧画面();
     )
post isofclass(タスク一覧画面, RESULT);
end メンテナンス作業画面
```

図 13 メンテナンス作業画面.vdmpp

```
class タスク情報画面 is subclass of 画面
values
システム : システム = システム`システムインスタンス;

instance variables
タスク情報 : システム`タスク情報型
     := システム.タスク情報を返す(システム.現タスク ID);
public 顧客名フィールド : token := タスク情報.顧客名;
public 住所フィールド : token := タスク情報.住所;
public その他フィールド : token := タスク情報.その他;

operations
public 戻るボタンを押下する : () ==> 画面
戻るボタンを押下する() ==
     (
          io.print("タスク一覧画面.タスク一覧を押下する¥n");
          io.print("タスク一覧画面.メンテナンス作業画面に遷移する¥n");
          return new メンテナンス作業画面();
     );
end タスク情報画面
```

図 14 タスク情報画面.vdmpp

```
class 作業内容登録画面 is subclass of 画面
values
システム : システム = システム`システムインスタンス;

instance variables
タスク情報 : システム`タスク情報型
     := システム.タスク情報を返す(システム.現タスク ID);

public 現象フィールド : token := タスク情報.作業内容.現象;
public 原因フィールド : token := タスク情報.作業内容.原因;
public 対策フィールド : token := タスク情報.作業内容.対策;

operations
public 戻るボタンを押下する : () ==> 画面
戻るボタンを押下する() ==
     (
          io.print("作業内容登録画面.戻るボタンを押下する¥n");
          io.print("作業内容登録画面.メンテナンス作業画面に遷移する¥n");
          return new メンテナンス作業画面();
     );

public 現象を入力する : token ==> ()
現象を入力する(現象) ==
     (
          io.print("作業内容登録画面.現象を入力する¥n");
          現象フィールド := 現象;
     );

public 原因を入力する : token ==> ()
原因を入力する(原因) ==
     (
          io.print("作業内容登録画面.原因を入力する¥n");
          原因フィールド := 原因;
     );

public 対策を入力する : token ==> ()
対策を入力する(対策) ==
     (
          io.print("作業内容登録画面.対策を入力する¥n");
          対策フィールド := 対策;
     );
end 作業内容登録画面
```

図 15 作業内容登録画面.vdmpp

```
class 交換部品登録画面 is subclass of 画面
types
public 部品フィールド型 :: 型番 : token
                          名称 : token;

values
システム : システム = システム`システムインスタンス;

instance variables
タスク情報 : システム`タスク情報型
     := システム.タスク情報を返す(システム.現タスク ID);

public 部品フィールド集合 : set of 部品フィールド型 := 部品フィールド集合を生成する(タスク
情報.交換部品集合);

operations
public 戻るボタンを押下する : () ==> 画面
戻るボタンを押下する() ==
     (
          io.print("交換部品登録画面.戻るボタンを押下する¥n");
          io.print("交換部品登録画面.メンテナンス作業画面に遷移する¥n");
          return new メンテナンス作業画面();
     );

public 交換部品を追加する : 部品フィールド型 ==> ()
交換部品を追加する(parts) ==
     (
          io.print("交換部品登録画面.交換部品を追加する¥n");
          部品フィールド集合 := 部品フィールド集合 union { parts };
     );

public 交換部品を変更する : nat1 * 部品フィールド型 ==> ()
交換部品を変更する(index, parts) ==
     (
          io.print("交換部品登録画面.交換部品を変更する¥n");
          部品フィールド集合 := 部品フィールド集合 union { parts };
     );

functions
public 部品フィールド集合を生成する : システム`交換部品集合型 -> set of 部品フィールド型
部品フィールド集合を生成する(交換部品集合) ==
     { mk_部品フィールド型(交換部品.型番, 交換部品.名称) | 交換部品 in set 交換部品集合 &
true };
end 交換部品登録画面
```

図 16 交換部品登録画面.vdmpp

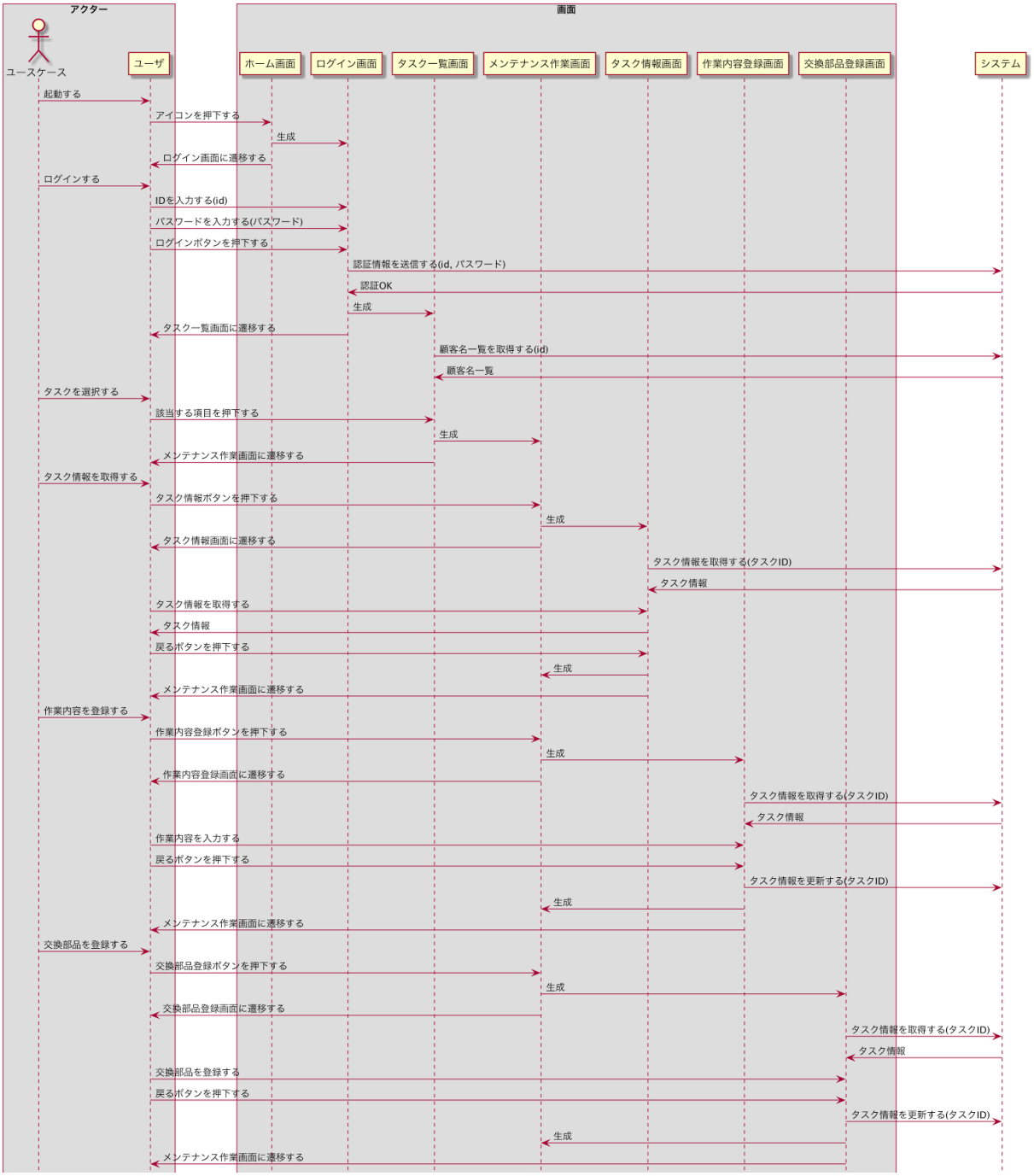

図 17 シーケンス図.svg (前半)

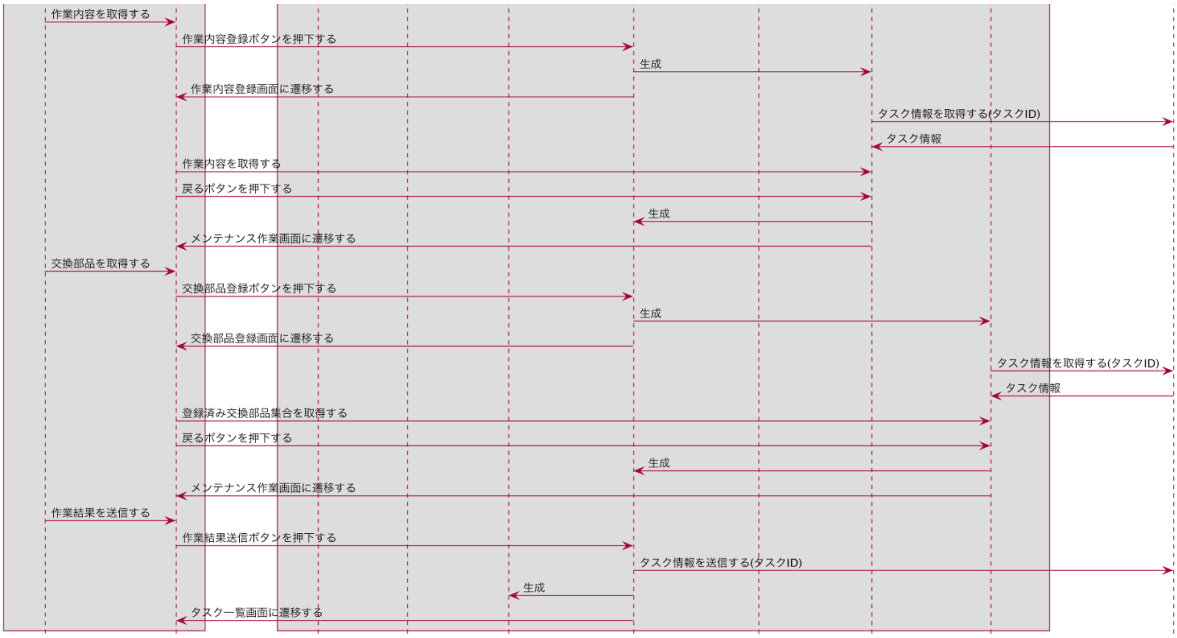

図 18 シーケンス図.svg (後半)

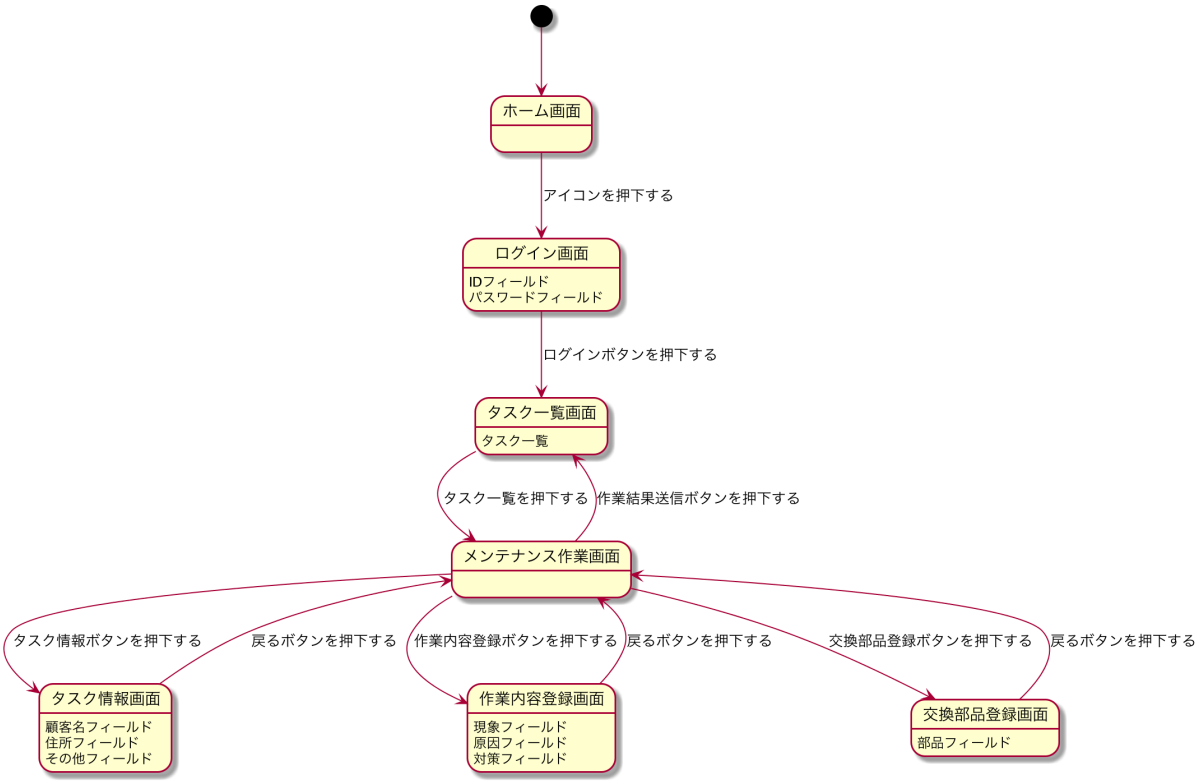

図 19 画面遷移図.svg