平成 26 年度

信州大学工学部情報工学科卒業論文

画面遷移モデルに基づく
シナリオ作成支援

海尻・小形研究室

学籍番号　11T5048H

中村　哲真

目次

1 序論

1.1 背景

ソフトウェア開発において，設計は上流工程で行われる．この時，設計に含まれる問題は最小にすべきである．なぜなら，設計はソフトウェア開発全体の礎となり，設計に含まれる問題は完成したソフトウェアに含まれてしまうためである．また，設計に問題がある場合，早期の発見が重要である．なぜなら，問題を解決する際，問題に関わる成果物をすべて修正する必要があり，工程が進むとその問題の影響を受ける成果物が多くなるためである．

設計段階に問題を検証する方法として，入力保存機能に着目したモデル駆動ユーザビリティ評価法の提案と評価[1]が提案されている．この手法では UML (Unified Modeling Language) を拡張し，評価対象のシステムを表す画面遷移モデルと，ユーザーの入力を想定するシナリオを定義している．これを用いて，ユーザーが感じる入力負担の大きさを定量的に評価する．この手法を用いることで，上流工程においてユーザビリティの評価が可能となる．

しかし，この手法に必要なモデルを手動で作成することは負担が大きい．以下に既知の問題点を示す．

- 記述量が多い．類似した，あるいは同一の値を複数入力することがある．
- 拡張記法が多く，正しくシナリオを作成するには拡張記法の理解が必要．
- 構成が複雑である．シナリオが正しく書かれていないと正しい評価が不可能になる．

これらように，入力保存機能に着目したモデル駆動ユーザビリティ評価法の提案と評価[1](以下，「対象手法」) は利用する際の難度が高い．手法の難度が高い場合，実際の開発では適用されない可能性が高くなる．

1.2 目的

本研究は，ソフトウェア開発の上流工程において，ユーザビリティの問題を発見する手法の支援を行うことを目的とする．上流工程でユーザビリティの評価を行う手法は存在する[1]が，利用難度が高く，実用には適さない．そこで，利用難度を低減するため，シナリオの作成を支援するツールを開発する．

対象手法の問題点および本研究における解決方針を述べる．

第一に，シナリオを作成する際，大量の値を入力する必要がある．このとき，入力する値の中には同一のものや類似しているものがあり，容易に補完できる．従って，入力された値をツールが記憶し，必要な場所で自動入力する．これにより，値の入力量を低減させる．

第二に，拡張記法が多い．対象手法は UML を拡張しており，手法に不慣れな場合正しく記述できない．これに対して，拡張記法をツールが補完し，手法に不慣れな人のシナリオ作成を補助する．

第三に，シナリオの構成が複雑である．対象手法は五個以上のオブジェクト図を正しく記述しなければならない．本研究では，オブジェクト図への記述をツールが行い，手法を利用する人（以下，「開発者」）の負担を軽減する．

また，本研究の評価として実際にツールを使用し，正しいシナリオを記述可能か検証を行う．

1.3 論文の構成

第二章では対象手法[1]に関する語句を説明する．第三章では関連研究について述べる．第四章では手法の説明を行い，第五章では評価について述べる．第六章ではまとめと今後の課題について述べる．

2　用語説明

2.1　画面遷移モデル

画面遷移モデルとは，システム上の画面間の遷移を定義するモデルである．本研究ではこれに基づいてシナリオの作成支援を行う．

対象手法ではクラス図上に画面および遷移が記述される (図 1)．画面遷移モデルの記法を定義するメタモデルを図 2 に示す．

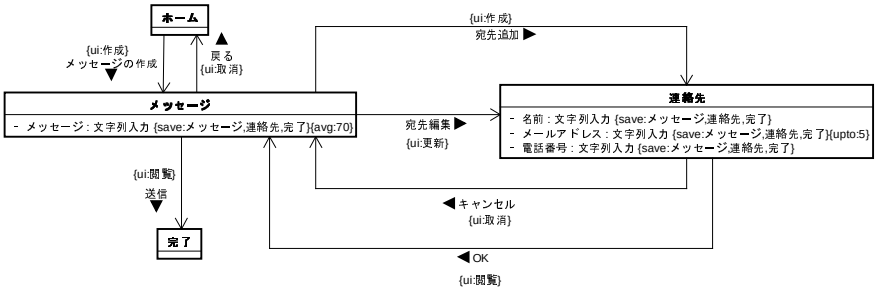

図 1　画面遷移モデル

画面遷移モデルの記法について説明する．括弧内に，対応するメタモデルの要素を示す．

画面 (Screen) はシステム上の画面を表し，クラスとして定義される．図 1 の「ホーム」，「メッセージ」，「連絡先」，「完了」のように，画面名がクラス名として表される．また，遷移 (Transition) を関係として，入力項目 (InputItem) を属性として持つ．

遷移と入力項目の抽象クラスとして，操作項目 (OperatableItem) が存在する．これは操作可能な項目を表し，平均入力量 (MetricsOfUserEffort.avg)，一画面入力限度数 (MetricsOfUserEffort.upto) を持つ．また，項目がどのような意図で操作されたかを示すユーザー意図 (UserIntention) も持つ．

対象手法では，これらは拡張記法で記述される．平均入力量は，一項目に対するユーザーの平均操作回数を表し，例えば「avg:70」と記述される．一画面入力限度数は，一画面で一項目に入力できる値の個数の上限を表す．例えば「upto:5」と記述される．また，ユーザー意図は，「ui:作成」のように記述される（図 1）．

ユーザー意図は以下の五種類に分類される．

作成 (Create)　値を作成する
閲覧 (Browse)　値を変更しない

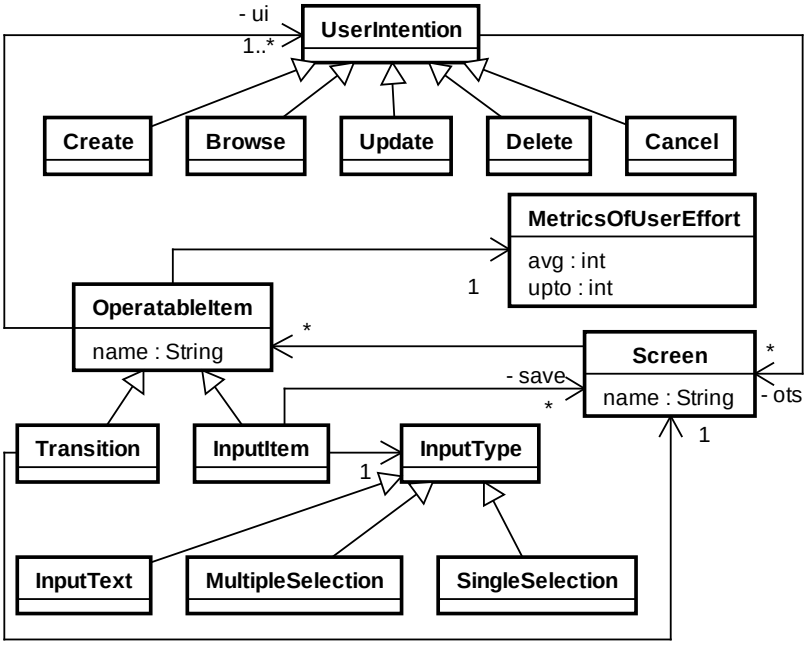

図 2　画面遷移モデルのメタモデル

更新 (Update)　値を更新する
削除 (Delete)　値を削除する
取消 (Cancel)　遷移元の値の作成，更新を取り消す

また，ユーザー意図は操作対象画面 (UserIntention.ots) を持つ．これはユーザー意図の対象となる画面を表し，操作対象が遷移先でない場合に明示される．

入力項目は入力型 (InputType) を持つ．これは入力項目の型として表される（図 1）．入力型は三種類あり，文字列入力 (InputText)，単一選択 (SingleSelection)，複数選択 (MultipleSelection) に分類される．

また，入力項目は保存範囲 (InputItem.save) も持つ．これは値の保存範囲を示し，保存範囲外の画面へ遷移した場合に破棄される．例えば，「メッセージ」画面クラスの「メッセージ」属性は，「ホーム」に遷移する際に破棄される（図 1）．

2.2 画面インスタンス

画面インスタンスは，図 3 のように画面 (Screen) をインスタンス化したもので，各画面で入力された値が格納される．

また，これには以下の六種類の拡張記法が定義されている．

`<<delete>>` 画面インスタンスが削除されたことを示す．

`<<update>>` 画面インスタンスが更新されたことを示す．

`<<cancel>>` 入力された値が取り消されたことを示す．

`<<destination>>` 意図の中で最後に到達する画面インスタンスであることを示す．

`\a` 入力値および画面インスタンス名に対する拡張記法．入力値がシステムによる自動入力であることを示す．

`\,` 入力値に対する拡張記法．一つの項目に対して複数の値を入力する際に用いる．

正しいシナリオを作成するには，拡張記法を適切に記述する必要がある．本研究では記述の誤りを防ぐため，ツールによる拡張記法の自動補完を行う．

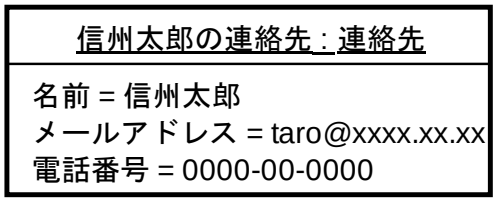

図 3 画面インスタンス

2.3 シナリオ

シナリオとは，想定されるユーザーの入力や操作を記述するものである．

対象手法のシナリオでは，想定されるユーザーの入力が画面インスタンスとして記述される．また，個々の画面遷移の操作順序は記述されない．画面遷移に対して細かく操作順序を記述した場合，画面遷移モデルが変更された際にシナリオを修正する必要があるためである．特に，対象手法では入力された値の保存範囲を考えるため，値の保存範囲を変更した場合にも画面遷移の順序が変化する．

しかし，順序が完全に存在しなければ，シナリオが成り立たない．そこで，シナリオを意図と呼ばれる単位で分割する．これはユーザーの行動指向のことである．例えば，商品を購入する，カートに入れた商品を修正する等である．このとき，意図の方向性を明確にするため，最後の画面は明示される．これにより，シナリオに大まかな順序が記述される．

シナリオは以下の 5 種類のオブジェクト図で記述され，ユーザーの入力はオブジェクト図上に画面インスタンスで記述される．

開始 (Start)　シナリオの起点となる画面インスタンスを一つ記述する

事前条件 (Precondition)　シナリオ開始前に存在する画面インスタンスを任意の数記述する

意図 (Intention)　各意図で入力される画面インスタンスを記述する

事後条件 (Postcondition)　シナリオ完了後に存在している画面インスタンスを記述する

終了 (End)　シナリオの終点となる画面インスタンスを一つ記述する

なお，シナリオに対して意図のオブジェクト図は複数存在し，他のオブジェクト図は一つずつ存在する．

3 関連研究

4 提案手法

本研究では，シナリオの作成を支援するツール (以下，「本ツール」) を設計し，実装した．プラグインにより拡張可能であり，対象手法が利用していることから，本ツールはastah*のプラグインとして実装した．また，astah* Plug-in は Java で記述する必要があるため，実装は Java で行い，GUI 部品の表示は awt，swing を利用した．

astah* Plug-in は，astah*に機能を追加する仕組みである．本ツールでは，astah* Plug-in の拡張タブビューを利用した．これは astah*の下部のフレームを，swing のコンポーネントとして操作可能にするもので，astah* Plug-in に用意されたインターフェース (IPluginExtraTabView) を実現すると利用できる．このインターフェースを介して，swing の JPanel に配置した GUI 部品を astah*エディタに表示する．

本ツールは画面遷移モデルを解釈して対象のシステムの画面をシミュレートし，GUI で表示する．これを開発者が，実際に対象のシステムを利用するように操作する．この操作の内容を本ツールが記憶し，シナリオを生成する．

これにより，以下の効果を期待する．第一に，拡張された記法をツールが補完し，初学者がシナリオを作成した際の文法的な誤りを防止する効果．第二に，以前入力した値をツールがすべて記憶し，適宜挿入することで入力量を低減する効果である．

図 4 に本ツールの構造を示す．また主要なクラスを解説する．

SupportSystem　本ツールのエントリーポイントである．シナリオ作成の進行や，Browser，ScenarioCreator，AbstractModelObserver 等の主要なクラスの管理を行う．また，GUI を実際に表示する．astah*の場合，このクラスが IPluginExtraTabView を実現し，astah*上に GUI を配置する．

Browser　本ツールの核となる部分である．このクラスは Screen（画面）の表示と画面遷移を行う．GUI 部品の構成を担う ComponentManager を持ち，画面遷移に応じて画面を表示する．また，開発者が入力したシナリオの具体値および遷移の意図を管理し，History へ格納する．詳しくは，ComponentManager は 4.1.3 章，Browser は 4.3.3 章で説明する．

ScenarioCreator　シナリオを生成するクラスである．History に格納されたシナリオの具体値と，画面インスタンスの集合である Precondition（事前条件）から，シナリオ

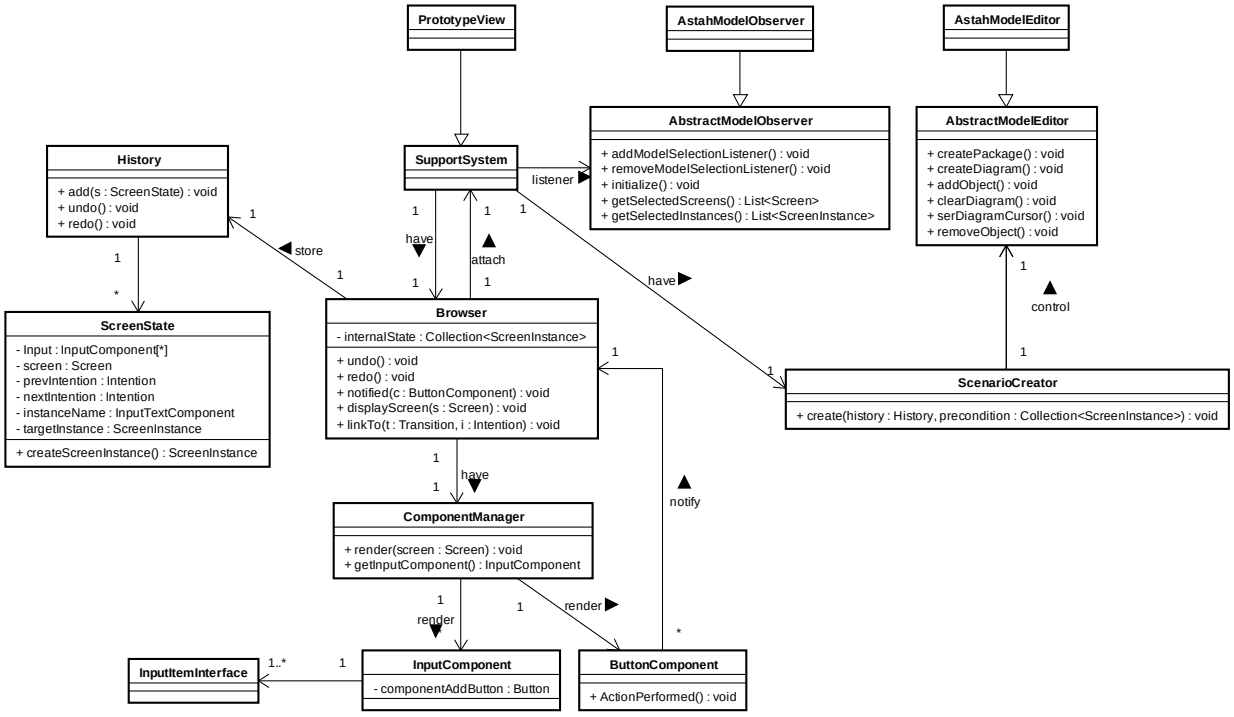

図 4　シナリオの作成支援ツール構造

を生成する．詳しくは 4.4 章で説明する．

AbstractModelObserver　これは抽象クラスで，対象とするエディタの違いを吸収する．エディタ上のクラスや画面インスタンスの選択状態が変化したとき通知を受け取り，登録されたリスナーのメソッドを実行する．また，選択された要素を取得する．

4.1　画面遷移モデルの解釈

本ツールでは対象のシステムを擬似的に再現する．そのため，画面遷移モデルを解釈して，画面を構成し，画面遷移を実現する．この内，画面遷移モデルの解釈と画面の構成は ComponentManager で実装している．

Screen（画面）から GUI を構成する．図 5 に，対象となる手法の画面遷移モデル (一部) と，それに関係する本ツールの構造を示す．

Screen は OperatableItem を持つ．対象手法ではこれを「操作項目」としており，ユーザーが操作可能な項目である．従って，本ツールではこれを GUI 部品に相当するものとして扱う．OperatableItem の実体は Transition（遷移）もしくは InputItem（入力項目）

である．従って，Transition には操作可能な GUI 部品を，InputItem には入力可能な GUI 部品を割り当てる．本ツールでは Transition が ButtonComponent に，InputItem が InputComponent に対応している．

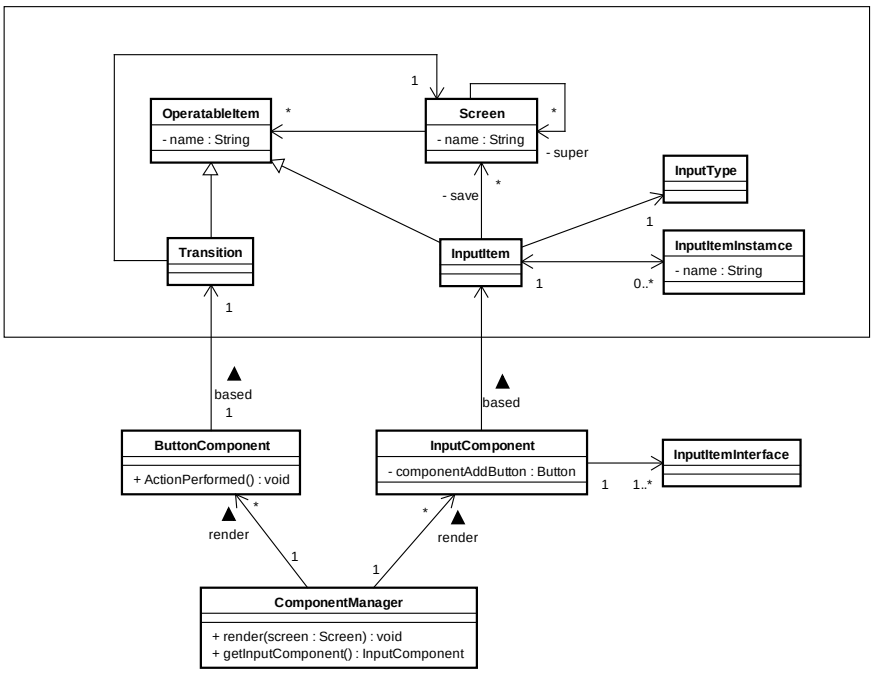

図 5　対象手法と本ツールの関係 (上：対象手法)

4.1.1　ButtonComponent

ButtonComponent は操作可能なボタンとして扱われる．また，内部的には Transition の他に UserIntention（ユーザー意図）を持つ．これは，どのような意図で遷移可能かを示す．

ButtonComponent は押下されたとき，Browser へ押下されたことを通知する．また，Browser は通知された時，どの意図で遷移を行うか選択肢を提示する．この時提示される選択肢は，ButtonComponent が持つ UserIntention によって生成される．さらに，提示された意図を選択すると，その意図による遷移が行われる．

また，本ツールでは特別な選択肢として意図のない遷移も提示される．これはシナリオには書き出されない遷移で，シナリオ上明示的に行う必要のない遷移が該当する．

遷移には，更新，削除のような，既存の画面インスタンスに変更を加える意図が存在する．この時，変更を加えるインスタンスを指定する必要がある．そのため，対象になる可

能性のある画面インスタンスが選択肢に追加される．

遷移および意図に関する詳細は章 4.2 で説明する．

4.1.2 InputComponent

内部に複数の入力項目 (InputItemInterface) を持つため，InputComponent は複数の GUI 部品の集合として扱われる．入力項目は，対象手法の InputType および InputItemInstance に相当するインターフェースである．本ツールでは値の入力可能な GUI 部品として扱われ，値の格納，取り出しと，GUI 部品の取得をメソッドとして持つ．

内部に複数の入力項目を持つため，基となる InputItem が複数の値の入力を許可していた場合，複数の値の格納が可能になる．複数の値の入力については，2.1 章で説明した一画面入力限度数がこれに当たる．

また，入力項目を追加するボタンは InputComponent に格納され，指定入力項目を削除するボタンは InputItemInterface に格納される（図 6）．

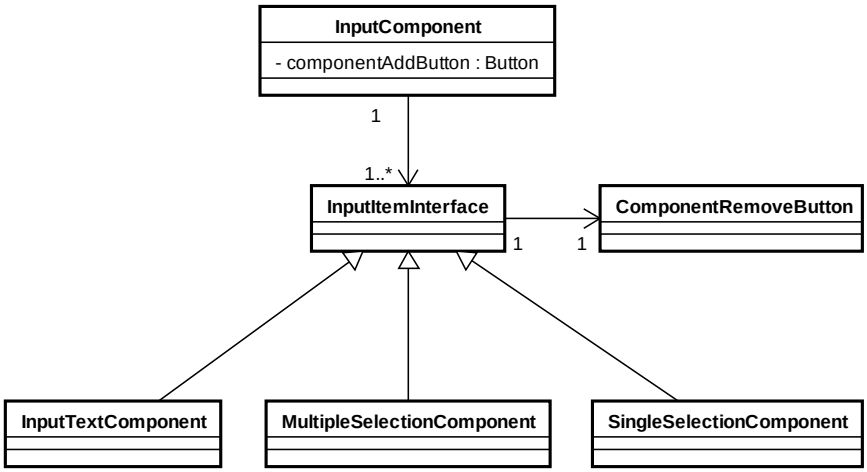

図 6　InputComponent の構造

InputItemInterface は InputType に従い，以下の三種類の実装を持つ．

InputTextComponent（**文字列入力**）　InputText に対応し，文字列の入力が可能な部品として扱われる．

MultipleSelectionComponent（**複数選択**）　MultipleSelection に対応し，値の選択および文字列の入力が可能な部品として扱われる．

SingleSelectionComponent（**単一選択**）　SingleSelection に対応し，値の選択および文字

列の入力が可能な部品として扱われる．

選択項目は，文字列の入力が可能な部品として扱われる．これは選択肢が画面遷移モデル上で定義されていないためである．

4.1.3 GUI の構成

GUI の構成は ComponentManager が行う．Screen（画面）から ButtonComponent および InputComponent を生成し，配置する．レイアウトは 4 つの領域にわかれ，図 7 のように上段に画面インスタンス名と undo/redo，下段に操作項目，中央に入力項目，左にシナリオ作成ボタンが配置される．

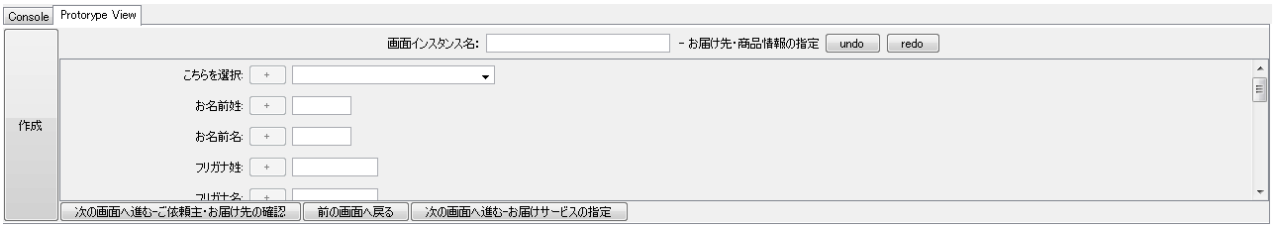

図 7　構成された GUI 部品

ButtonComponent を配置する時，図 7 の「次の画面へ進む」のように，同一の名前の操作項目が定義されていることがある．実際に表示すると判別がつかないため，同一の名前の操作項目が配置される場合には，ボタンのラベルを「操作項目名-遷移先の画面名」に変更し，判別可能にする．

また，InputComponent を配置するとき，どの値の入力項目か判別可能にするため，項目の名前をラベルとして付加する．

4.2 シナリオ作成の手順

本ツールによる，対象手法のシナリオとして正しいシナリオを作成する手順を，開発者が行う操作を基に説明する．また，これに係る内部動作は 4.3 章で説明する．

ここで，最終的に出力されるシナリオを単に「シナリオ」，シナリオで想定されるユーザーの操作を「想定される操作」，入力を「想定される入力」とする．

本手法ではシナリオの作成支援を行うので，シナリオ作成の開始時点で画面遷移モデルは既に完成しているものとする．

シナリオの作成は，開発者が「シナリオ作成開始」ボタンを押下すると開始される（図

8)．シナリオの作成開始時に完成した画面遷移モデルを読み込む必要があるため，開発者による操作が必要になる．

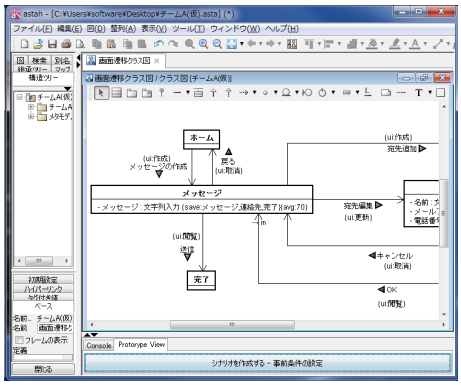

図 8　シナリオ開始前の画面

最初に，事前条件 (Precondition) を作成する．事前条件はシナリオが始まる前のシステムの状態である．これは画面インスタンスの集合として表される．これを作成するには，任意の画面から画面インスタンスを作成するか，既存の画面インスタンスを再利用し，集合に追加する．具体的には，画面もしくは画面インスタンスを選択すると図 9 のように入力項目が表示されるので，そこに具体値を入力する．入力後，「事前条件にインスタンスを追加する」ボタンを押下すると事前条件に追加される．

事前条件の作成に関するツール側の処理は，4.3.2 で説明する．

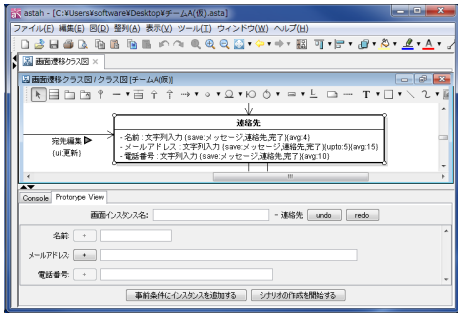

図 9　事前条件設定時の画面

次に，想定される操作と想定される入力からシナリオ本体を作成する．本ツールは，画面遷移モデルから対象のシステムの画面を擬似的に再現する．これを実際に操作することで，シナリオの本体を作成していく．画面の擬似的な再現は，シナリオ内で最初に表示さ

れる画面を選択して「シナリオ作成の開始」ボタンを押下すると開始される．システムの擬似再現に関する詳細は 4.3.3 章で説明する．

再現された画面に対して，想定される入力の値を実際に入力する．この時，複数の値を入力する場合は入力項目の隣の「＋」ボタンを押下する．また，入力項目を削除する場合，隣の「×」ボタンを押下する（図 10）．

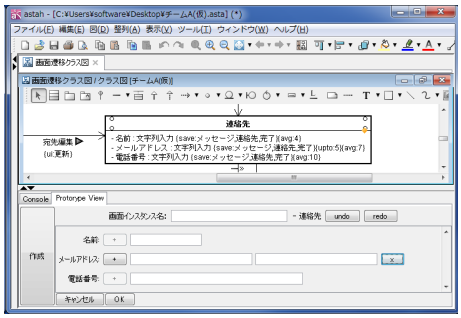

図 10　コンポーネントの追加，削除ボタン

値の入力後，想定される操作に従って遷移を行う．下部のボタンを押下すると，遷移意図を選択するメニューが出現する（図 11）．これはどのような目的で遷移を行うかを示すもので，対象の手法に則って「閲覧意図で遷移する」（閲覧），「画面インスタンスを作成する」（作成），「画面インスタンスを更新する」（更新），「画面インスタンスを削除する」（削除），「入力を取り消す」（取消）が存在する．また，本ツールには，明示する必要の無い遷移に用いる「意図なしで遷移する」も存在する．

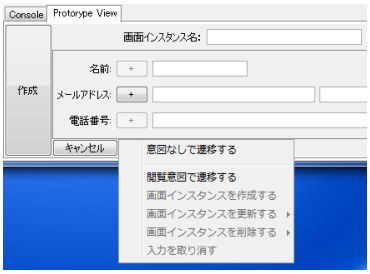

図 11　遷移意図の選択

本ツールにおける六種類の遷移の意図について以下に述べる．

作成　遷移先の画面インスタンスを新たに作成するときに行う．例えば，顧客情報を新しく追加する場合である．

閲覧　遷移先の画面インスタンスを変更しないが，明示的に遷移する必要がある場合に行う．対象の手法では，特定の遷移により画面インスタンスが破棄される可能性があるシステムを想定している．これにより，遷移自体が意味を持つ場合がある．

更新　既に存在する画面インスタンスを更新するときに行う．例えば，既存の顧客情報を変更する場合である．「画面インスタンスを更新する」にカーソルを合わせると，既存の画面インスタンスを選択するサブメニューが出現する（図 12）．

削除　既に存在する画面インスタンスを削除するときに行う．例えば，特定の顧客情報を消去する場合である．更新と同じように，「画面インスタンスを削除する」にカーソルを合わせると，既存の画面インスタンスを選択するサブメニューが出現する

取消　現在の画面インスタンスの作成，更新を取り消すときに行う．例えば，顧客情報の変更を行おうとしたが，取りやめた場合である．

意図なし　遷移先の画面インスタンスを変更せず，かつ明示的に遷移する必要が無い場合に行う．対象手法では明確な操作手順は記述ないため，すべての遷移を記録する必要はない．従って，シナリオ上明示する必要のない遷移はすべて意図なしになる．

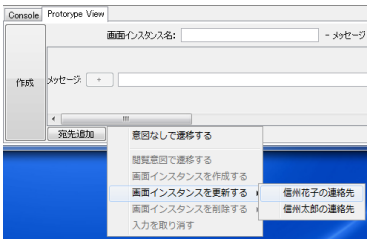

図 12　画面インスタンスの選択

遷移先で想定される入力の値を入力し，さらに遷移を行い，シナリオの本体を構成していく．

想定される操作をすべて行った後，シナリオを生成する．

左側の「作成」ボタンを押下すると，シナリオの名前を入力する画面に移る（図 13）．シナリオの名前を入力し，「作成」を押下すると，シナリオが本ツールにより生成される．シナリオの生成に関する詳細は 4.4 章で説明する．

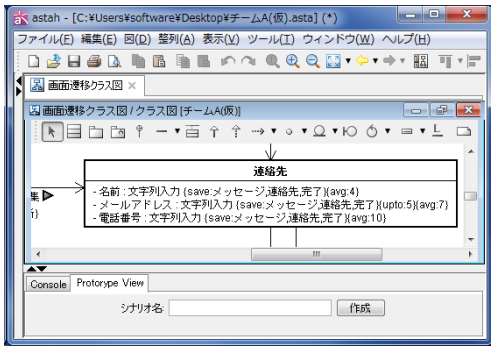

図 13　シナリオの名前の入力

4.3　シナリオを作成する際の内部処理

4.3.1　初期化

シナリオを作成する前に，前回作成したシナリオの影響を防ぐため，各クラスの初期化を行う．この時，Browser，ScenarioCreator，ModelListener を初期化する．

4.3.2　事前条件の作成

事前条件は，シナリオ開始前のシステムの状態を表す画面インスタンスの集合である．

本ツールでは，事前条件を作成する際，開発者が選択した画面クラスまたは画面インスタンスを読み取り，表示する．これの実現には，選択状態の変化を認識すること，選択された要素を取得することが必要である．

そのため，モデルの状態を監視するインターフェース，AbstractModelObserver を定義した．これには，モデルの選択状態が変化したときに通知を受けるリスナーを設定するメソッドと，現在選択されている画面クラスと画面インスタンスを取得するメソッドが定義される．

本ツールでは astah*を利用するため，AstahModelObserver を実装した．astah*では選択状態が変化した際に通知を受け取るリスナーが設定可能なので，astah*から受けた通知をそのまま設定されたリスナーに通知する．また，astah*では選択されている要素が取得できる．選択されている要素が画面クラスか画面インスタンスの場合，それらの集合を構成して選択された要素を取得可能にする．

これを利用して，選択された画面クラスもしくは画面インスタンスを取得する．また，ComponentManager を用いて，取得した画面を表示する．

また，「事前条件を追加」ボタンを押下された際には，入力された値から画面インスタンスを生成し，事前条件の集合に追加する．

4.3.3 対象システムの擬似再現

対象システムの擬似的な再現には，Browser を使用する．開発者は擬似的に再現したシステムを実際に操作して，本ツールはその操作履歴 (History) を保持する．

操作は，ScreenState を単位として操作履歴に保存される．ScreenState には入力された項目 (input)，対象の画面 (screen)，遷移の意図 (prevIntention, nextIntention)，画面インスタンス名 (instanceName)，意図の対象のインスタンス (targetInstance) が格納される．

また，操作履歴に対して Undo，Redo 操作を実現した．値の入力は間違える可能性があり，その際に最初から作成し直すことは明白にコストが高いためである．実装は二つのスタックを用いた（図 14）．通常の入力値はスタック history へ格納していく．Undo 操作を行う際には history から一つ取り出し，一時的なスタック temporary へ格納する．Redo 操作を行う際には temporary から一つ取り出し，history へ格納する．

図 14　未完成

また，Browser は現在シミュレートしているシステムの内部状態 (internalState) を，画面インスタンスの集合として持つ．これにより，更新の意図や削除の意図で遷移する際に，既存の画面インスタンスを再利用できる．しかしこの時，Undo/Redo 操作による内部状態の変更が問題になる．取り消された操作により値が更新されていた場合，更新される前の値が保存されていないため，内部状態に不整合が起こる．

この問題に対応するため，二つの解決策を考案した．

一つは，ScreenState に内部状態を持たせ，Undo/Redo する度に内部状態を更新する方法である．この方法はメモリーの消費が大きいが，計算時間を抑えられる．

もう一つは，Undo/Redo する度に内部状態を再度計算しなおす方法である．この方法はメモリーの消費は小さいが，計算時間がかかる．

本ツールは astah*の一部として実行されるため，astah*本体への影響を考慮してメモリ空間の占有を最小限に抑える必要がある．従って，今回は後者の方法を採用した．

4.4 シナリオの生成

最後に，シナリオを生成する．これは本ツールが自動で行う．また，実際にオブジェクト図を操作する必要があるため，AstahModelEditor を使用する．

シナリオを生成する際，入力された具体値と遷移意図が格納された History と，シナリオを開始する前の状態を表した画面インスタンスの集合である Precondition を用いる．以下に生成の手順を示す．

最初に，Precondition を生成する．これには与えられた Precondition をすべてオブジェクト図に書き出す．

次に，Start を生成する．一番初めの画面インスタンスは History の底に格納されているため，これをオブジェクト図に書き出す．

次にシナリオの核となる Intention を生成する．History を順番に参照し，オブジェクト図に書き出していく．この時，画面に入る際の意図が無い場合，シナリオの構成上必要がないと判断し，書き出されない．

また，意図の変わり目は意図 Create による遷移と意図 Update による遷移の間にあると考えた．意図 Update では意図 Create で作成されたインスタンスに変更を加えるため，意図の方向性が異なるといえる．そこで，意図 Create による遷移の後を状態 Create とし，意図 Update による遷移の後を状態 Update して，状態が変化したときを意図の変わり目とした．また，初期状態は最初に行われた意図 Create もしくは意図 Update の遷移により決定する．

Intention の生成と並行し，Postcondition を生成する．Postcondition は最終的なシナリオの状態で，Precondition と各 Intention を合成したものになる．そのため，Intention と同時に生成し，オブジェクト図に書き出す．

最後に End を生成する．History の最後の要素をオブジェクト図に書き出す．

5　評価

5.1　評価方法

[この章は評価計画が確定した後，記述する．]

5.2　考察・結果

[この章は評価実験の後，記述する．]

6　結論

6.1　まとめ

本論文では，ソフトウェア開発の上流工程においてユーザビリティに関する問題を発見するための手法[1]の作成支援ツールを実装，説明した．ツールによって入力量を削減し，手法の利用コストを低減した．また，手法独自の記法をツールによって補完し，シナリオ作成の誤りを低減した [評価実験に依る].

本ツールにより，上流工程においてユーザビリティ評価が行われ，より品質の良い製品が開発されることを期待する．

6.2　今後の課題

今後の課題として，コンポーネントの表現の柔軟化が挙げられる．現在，コンポーネントは決まった種類の部品のみ定義されている．しかし，現実にはコンポーネントがコンポーネントを内包したり，他のコンポーネントに関係付けられたりと，様々なコンポーネントが存在する．これの改善案として，コンポーネントの表現を拡張可能にする案が挙げられる．これにより，現実のシステムに近いコンポーネントの構成が行えるようになり，より現実に近いシナリオの作成が可能になる．

これにあわせて，複数選択の選択肢の指定も定義できるよう拡張を行う．

また，他の手法に対して支援を行うことも今後の課題とする．今回の対象手法は一つの側面からの評価であり，ソフトウェアの評価にはより多くの側面からの評価が求められる．画面遷移モデルが存在し，テストシナリオの生成を目的とする手法であれば，本ツールと同様に支援を行える．

参考文献

[1] 小形真平，早川弘基，海谷治彦，海尻賢二，入力保存機能に着目したモデル駆動ユーザビリティ評価法の提案と評価，コンピュータソフトウェア，Vol.29，No.1(2012)，pp.1-6