GUI アプリケーションの開発支援に関する研究

Research on GUI Application Development Support

白 銀 純 子

2002 年 6 月

目 次

図 目 次

表 目 次

第1章　はじめに

近年、様々な業務の分野にコンピュータが普及し、種々のアプリケーションが開発されている。それに伴い、コンピュータに関する知識を持たない人であってもコンピュータを利用する機会が増えている。そのため種々のツールの使いやすさであるユーザビリティ[1]が、アプリケーションに関しても重要視されてきている。特にGUI（Graphical User Interface）は、アプリケーションにおいて直接エンドユーザが接する部分であるため、ユーザビリティに対して大きな影響を与える。そこで、ユーザビリティの評価法やユーザビリティを意識したGUIデザインの研究も多く行われており、GUIの使いやすさは大きな関心が寄せられている[2][3]。

また、GUIアプリケーションを効率的に設計・開発することを支援するために、様々な研究やツールの開発が行なわれている。しかし、それにもかかわらず、GUIアプリケーションの構造は非常に複雑であるため、開発に多くの時間がかかり、開発者にとって大きな負担となっている。さらに、近年のユーザビリティ重視の傾向から、GUIに関してはプロトタイプを作成してそのプロトタイプを評価し、改良するということが繰り替えされるため、GUIの変更が頻繁に起こり、開発者の負担を増加させる一因にもなっている。

1.1　ユーザビリティ

アプリケーションの使い勝手は、「ユーティリティ」と「ユーザビリティ」に分類することができる。「ユーティリティ」は、そのツールを使って何ができるかやその効率などの機能及び性能を表す。「ユーザビリティ」はそのツールの使用方法のわかりやすさや記憶のしやすさなどの使用感を表す。従来、アプリケーションはユーティリティを重視して開発が行われてきた。しかし近年、コンピュータに関する知識を持たないエンドユーザが増加するに従い、ユーティリティだけでなくユーザビリティも重視されるようになってきた。

ユーザビリティは、主に以下のユーザビリティ特性から構成される[4]。

学習容易性

はじめてアプリケーションを使用する際に、目的の作業をいかに早く正しく行うことができるか、操作方法をいかに早く習得できるかを表す。そのアプリケーションを使用したことのないユーザを選び、ある一定の習熟度に達するまでの時間を測定することで学習容易性を評価する。コンピュータの初心者にとって重要な指標である。

効率性

アプリケーションの操作方法の習得後、ある作業をいかに早く正確に行うことができるかを表す。一定の習熟度に達しているユーザを選び、そのユーザが特定の作業を行う際に必要な時間を計測することで効率性を評価する。

記憶容易性

連続してそのアプリケーションを使用するのではなく断続的に使用する場合、使用しなかった期間の後に使用した際に、そのアプリケーションの操作方法をどの程度記憶しているか、つまり再学習の労力を表す。一定期間そのアプリケーションを使っていなかったユーザに対して、ユーザビリティを評価するテストを行うか、あるいはアプリケーションを試用したユーザに対して種々のコマンドに関する質問をする記憶テストを行うことで記憶容易性を評価する。そのアプリケーションを断続的に使用するユーザにとっては重要な指標である。

エラー頻度

アプリケーションが発生させるエラーではなく、ユーザの操作ミスによるエラーの発生しやすさや、エラーが発生した際の回復のしやすさを表す。他のユーザビリティ特性を評価する際に、エラーが発生した回数も併せて計測することにより、エラー頻度を評価する。

主観的満足度

ユーザがアプリケーションを使っていて楽しいと感じることができるか、満足することができるかを表す。そのアプリケーションの試用後に、ユーザに簡単な質問をすることで主観的満足度を評価する。

1.2　GUI アプリケーション開発の現状

ユーザインタフェースの構築に関しては、複雑な構造を持つ GUI アプリケーションの開発労力軽減すること、ユーザビリティに着目して GUI を評価することなどが研究され、

ツールが開発・製品化されている。

アプリケーション開発労力の軽減に関しては、統合開発環境として、Forte for Java[5] や VisualAge[6]、JBuilder[7] など多くのツールが商用化されている。これらのツールは、GUI に関してはウィジェットをビジュアルに作成・配置することができ、ウィジェットの大きさや色、文字のフォントなどもビジュアルに設定することができる。しかしこれらは、アプリケーションの設計結果を手動で実装に結び付けなければならず、設計と実装との間に矛盾が起こる可能性がある。また研究段階では、UML のユースケース図やクラス図、コラボレーション図などから GUI を生成する手法 [8][9] やタスクモデルを作成してそこから生成する手法 [10][11] など様々なものが提案されている。しかしこれらは、生成のための特別な記述が必要なため、GUI 生成のために開発者はその記述方法を学習しなければならず、開発者に対する負担が大きい。

GUI を評価することに関しては、USINE[12] や SHERLOCK[13] などが提案されている。USINE は、ユーザのイベントが発生する条件着目し、タスクモデルを定義する。そしてユーザの操作に対して履歴を収集し、その履歴からタスクモデルを作成し、あらかじめ定義されているタスクモデルと比較することで、ユーザが操作の過程で間違いを起こした箇所とその間違いの種類（イベントの事前条件を満たしていなかった、必要のないウィジェットを操作してしまった、など）を特定することにより、GUI の間違いの起こりやすいところを判断する手法である。また SHERLOCK は、GUI の各ウィンドウ内のウィジェットの配置に着目し、各ウィジェットの位置や大きさ、色、各ウィンドウの上下左右のマージン、などを計測し、ウィジェットの配置の一貫性を判断する手法である。しかしユーザビリティは、最終的にはエンドユーザの主観によって判断されるため、使いやすい GUI を一概に定義することは不可能である。従って、上記のような、操作ミスの起こりやすい箇所の発見や、ウィジェット配置の一貫性が保たれていない箇所の発見といった、消極的な支援しかなされていないのが現状である。

1.3 開発支援手法

本研究では、エンドユーザの意見を反映させたアプリケーションの開発を行い、ユーザビリティを高めていく際に、アプリケーション開発者を統合的に支援する手法を提案する。特に、大規模なアプリケーションや複雑な GUI を開発する際には、アプリケーションの設計結果と実装との間の矛盾がないことが重要となる。GUI に関しては、アプリケーション内の各ウィンドウに、ウィジェットが過不足なく配置されることが重要である。各ウィンドウにおいて、設計上配置されるべきウィジェットが配置されていない、設計上配

置されないウィジェットが配置されている、あるいは設計とは異なるウィジェットが配置されているなどした場合は、それを修正し、設計結果と一致させるために大きな労力が必要となり、開発者にとっては大きな負担となる。そこで本研究では、アプリケーションの GUI 部の実装を自動化することにより、設計結果と実装との間の矛盾の発生をなくし、開発者の負担を軽減することを目指す。具体的には、以下の 3 つの手法によって支援を行う。

- エンドユーザ指向 GUI プロトタイプ生成手法
- 設計結果の実装への自動反映手法
- ウィジェット変更時のプログラム自動変更支援手法

以下において、それぞれの手法の概略を述べる。

エンドユーザ指向 GUI プロトタイプ生成手法

ユーザビリティを向上させるためには、早期にプロトタイプを作成し、そのプロトタイプをエンドユーザの手によって評価し、プロトタイプを改良していくことを繰返すことにより、エンドユーザの意見を反映させることが効果的である [4]。このためには、エンドユーザにも理解できるようなアプリケーションの外部的な振舞いの記述をもとに GUI を生成し、これを利用者に示し、その結果をフィードバックさせることにより、GUI のデザインを決定することが有効である。

エンドユーザにも理解できるようなアプリケーションの外部的な振舞いの記述として、ユースケース図 [14][15] がある。ユースケース図はアプリケーションが提供する機能とシステムの外部要素とのインタラクションを表現したものであり、主に UML によるアプリケーション開発の分析フェーズで利用されるモデルである。また、アプリケーションが提供する機能の処理の流れを記述したものをシナリオと呼ぶ。このシナリオは、記述形式は決められていないが、自然言語で記述されることが多いため、エンドユーザでも容易に理解することができると考えられる。

そこで本手法では、記述されたユースケース図とシナリオからアプリケーションの GUI 部のプロトタイプを生成する手法について提案する。まず、開発者がユースケース図とシナリオの記述を行う。本手法では、シナリオは自然言語で箇条書きで記述することとし、箇条書きで記述された 1 つ 1 つの項目をイベントと呼ぶ。そして、記述された全てのイベントの中から、同一動作を表すイベントを統合する。これにより、アプリケーション全

体のイベントの流れを表す向グラフ（シナリオ併合グラフ）が生成される。生成されたこのグラフに対して、開発者が、誤って同一とみなされているイベント、及び同一とみなされていないイベントを指定することにより、シナリオ併合グラフが完成する。そしてこの時点で GUI におけるユーザに対する入力・表示・ユーザイベント発生項目、そしてそれらの項目のうち、同じウィンドウ上に存在する項目を開発者が指定する。それらの情報及び、シナリオ併合グラフから抽出された GUI 制御構造を元に GUI プロトタイプが生成される。

設計結果の実装への自動反映手法

そして、アプリケーションを完成させるためには、アプリケーションの設計結果をもとにしてアプリケーション処理部分の実装を行い、デザインが決定された GUI との結合を行う必要がある。現状では、仕様からアプリケーション処理部分への対応付け及び GUI 部との結合は手動で行われている。しかし大規模なアプリケーションになるほど、仕様とアプリケーション処理部との対応付け、及び GUI 部との結合には誤りが起こりやすく、その誤りを修正するために膨大な労力と時間を必要とする。また、そのような誤りを起こさないよう注意して実装を行うことは、開発者にとって大きな負担となる。そこで、仕様から GUI 部とアプリケーション内の関数の骨組み、GUI 部と関数との連結部分、及び関数の制御構造を自動生成することが、上記のような誤りをなくし、アプリケーション開発者の労力を軽減するために効果的である。

そこで本手法では、GUI アプリケーション全体の制御構造をペトリネットを用いて記述し、その記述された結果から GUI アプリケーションを生成する手法について提案する。具体的にはペトリネットに対し、プレースにウィジェット情報、つまり 1 つのプレースに、アプリケーション内の一つのウィンドウを対応させ、そのウィンドウに配置するウィジェットを開発者が指定する。またトランジションにユーザイベントのバインド情報、つまりそのトランジションで起こるユーザイベントの種類と、そのイベントをバインドするウィジェットを指定する。そしてこれらの情報をもとに、ペトリネットで記述されたアプリケーションの制御構造に従って、GUI プログラム、関数の骨組み及びステートマネージャが生成される。GUI プログラムはウィジェットの生成、ウィジェットに対するイベントの生起などの GUI に関するプログラムであり、関数の骨組みは処理内容の書き込まれていない関数とそのつながりのプログラムである。そしてステートマネージャは、アプリケーション実行中のトークンの位置を監視し、ペトリネットの忠実に従ってアプリケーションを動作させる機構であり、このステートマネージャによって、生成されたアプリ

ケーションがペトリネットに忠実に従って動作することが保証され、開発者は関数同士のつながりなどに注意することなく、各関数の処理内容のみを記述することでアプリケーションが完成する。

ウィジェット変更時のプログラム自動変更支援手法

また、GUI はアプリケーションとエンドユーザとの直接の接点であるため、そのユーザビリティはそのアプリケーションのエンドユーザの経験や好みなどに大きく依存する。また、開発段階のユーザビリティの評価では発見されず、運用段階に入ってはじめて使い勝手の悪い部分が発見されることも多々ある。従って GUI は、開発段階ではもちろん、運用段階においてもデザインを変更する必要がある。この GUI のデザインの変更では、GUI の各ウィンドウ内のウィジェットの位置や大きさなどのレイアウトだけでなく、ウィジェットそのものの種類を変更することも多い。しかし現状では、ウィジェットの種類を変更すると、そのウィジェットとアプリケーション処理部との連結部分は、アプリケーション開発者が手動で結合しなおさなければならない。大規模なアプリケーションでは変更が必要なウィジェットも多く、それに伴って修正箇所も多くなる。従って結合の際の誤りも多くなるため、その修正に大きな労力が必要となり、また開発者も誤りなく修正するために注意を払う必要があり、開発者には大きな負担となる。従って、GUI のユーザビリティの向上には、GUI の変更容易性が重要となる。

そこで 本手法では、この GUI の変更容易性を実現するために、GUI のデザイン変更時のアプリケーション処理部とウィジェット変更箇所を自動的に結合する手法を提案する。具体的にはあらかじめウィジェットを 5 つの役割に分類しておく。そして分類されたウィジェットが持つメソッドも、データの設定、データの取得などの処理ごとに分類しておく。そして GUI 構築時に、各ウィジェットが持つ役割の中で、どの役割を果たすかを指定しておき、ウィジェット変更時にその役割に応じて変更可能なウィジェットの一覧の中から、変更するウィジェットを選択する。これにより、変更前のウィジェットのメソッドが、変更後のウィジェットの、その分類された処理に対応するメソッドに変換される。

1.4　本論文の構成

本論文では、2 章で エンドユーザ指向 GUI プロトタイプ生成手法の詳細及び開発したシステムを用いて実際に GUI プロトタイプを生成し、その評価結果と考察を述べる。3

章で 設計結果の実装への自動反映手法の詳細及び開発したシステムを用いて実際に GUI アプリケーションを生成し、その評価結果と考察を述べる。4 章で ウィジェット変更時のプログラム自動変更支援手法の詳細及び開発したシステムを用いて実際に GUI アプリケーション中のウィジェットを変更し、その評価結果と考察を述べる。最後に 5 章で本論文のまとめを述べる。

第2章　エンドユーザ指向 GUI プロトタイプ生成手法

2.1　はしがき

GUI アプリケーションのユーザビリティを向上させるためには、早期にプロトタイプを作成し、そのプロトタイプをエンドユーザの手によって評価し、プロトタイプを改良していくことを繰返すことにより、エンドユーザの意見を反映させることが効果的である [4]。このためには、エンドユーザにも理解できるようなシステムの外部的な振舞いの記述をもとに GUI を生成し、これを利用者に示し、その結果をフィードバックさせることが有効である。

このような条件を満たす記述にユースケース [14][15] がある。ユースケースはアプリケーションが提供する機能を表現したものであり、主に UML によるアプリケーション開発の分析フェーズで利用されるモデルである [15]。

以上を踏まえ、本手法では、ユースケース図からアプリケーションの GUI 部のプロトタイプ（GUIプロトタイプと呼ぶ）を生成する手法について提案する。具体的には、ユースケース図と自然言語で記述されたシナリオをもとにして GUI の制御構造を抽出し、プロトタイプに組み込むことことを考える。GUIプロトタイプを生成するためにユースケース図とシナリオを選択した理由は以下の通りである。

1. ユースケースはエンドユーザを中心として記述されるため、エンドユーザの視点を容易に取り入れることができる。

2. ユースケース図及びシナリオは、UML を用いたアプリケーション開発の要求分析の段階において作成されるため、GUIプロトタイプの生成のための特別な記述を開発者に求める必要がない。

3. 設計結果であるユースケース図及びシナリオの記述から GUIプロトタイプを生成することで、設計結果と生成された GUIプロトタイプの間に矛盾がないことが保証さ

れる。

4. ユースケース図及びシナリオは、アプリケーション開発に関する知識を持たないエンドユーザであっても理解しやすいものである。従って、記述したユースケース図及びシナリオを、GUIプロトタイプの評価時に、操作手順を記したドキュメントとして利用することができる。さらに 3. の理由により、このドキュメントと GUIプロトタイプの間には矛盾がないことが保証される。

2.2　本手法の概要

2.2.1　ユースケース図とシナリオ

図 2.1 及び図 2.2 に、本手法で用いるユースケース図とシナリオの例を示す [14][15]。この例は、レンタルビデオ業務のアプリケーションの一部である。ユースケース図を構成する要素は以下の通りである。

アクター モデル化対象のアプリケーションとやりとりをする人や物であり、ユースケース図中では人形のシンボルで表される。図 2.1 では「店員」がアクターである。

ユースケース アクターとアプリケーションとの間の対話をモデル化したものであり、ユースケース図中では楕円の中にユースケース名を記述して表す。図 2.1 では「ビデオを貸し出す」、「ビデオを検索する」、「統計をとる」、「会員 ID を入力する」、「ポイントを還元する」がユースケースである。

アクターとユースケースとの間の関係を図示したものをユースケース図と呼ぶ。また、ユースケースは以下の要素で構成される。

シナリオ ユースケースでのエンドユーザやアプリケーションの動作・処理の流れの 1 つを記述したものである。図 2.2 は、ユースケース「ビデオを貸し出す」のシナリオの 1 つで、「貸出処理」という名前をしている。ただしこの例は、シナリオ「貸出処理」の一部である。

事前条件 ユースケースの開始時にアプリケーションが満たさなければならない条件である。

事後条件 ユースケースの終了時にアプリケーションが満たさなければならない条件である。

本手法ではシナリオは自然言語で箇条書きで記述する。そして箇条書きで記述された各項目をイベントと呼ぶ。本手法においては、シナリオとは、イベントを発生する順番に並べたものであると考える。

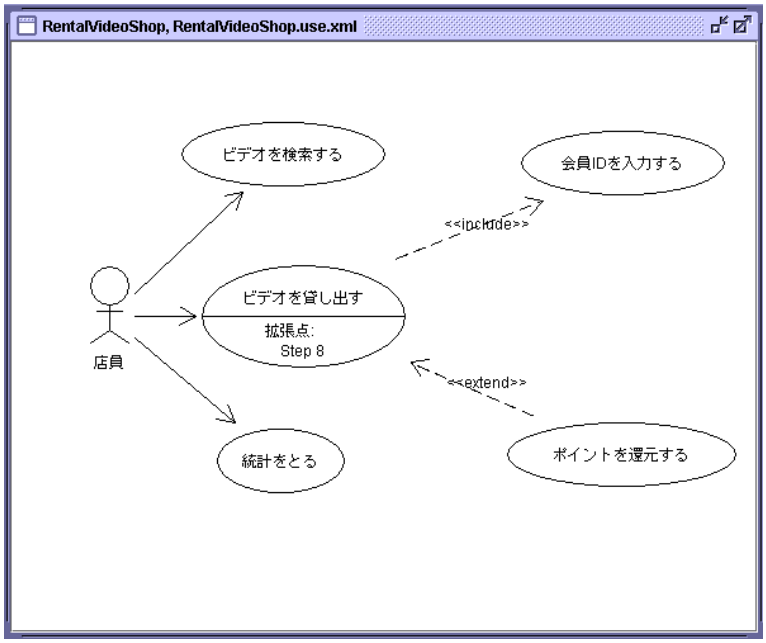


図 2.1: ユースケース図例

シナリオ（主シナリオ）: 貸出処理	
1. 「ビデオ貸出」を押す	8. ビデオのタイトルを表示する
2. include （会員 ID を入力する）	9. 返却日を表示する
3. 不正利用履歴を調べる	10. 料金を表示する
4. ビデオ ID を入力する	11. 「了解」を押す
5. 貸出期間を選択する	12. 会員データベースを更新する
6. 「OK」を押す	13. ビデオデータベースを更新する
7. 料金を計算する	

図 2.2: シナリオ例（ビデオの貸出）

1 つのアプリケーションにおいて、全てのイベントが異なる動作を表すことはまれである。つまり、通常は、表現は異なるが同一の動作を表すイベントが存在する。そこで本手法では、ユースケース図及びシナリオを記述する段階で、同一の動作を表しながら、かつ表現が異なるイベントについて指定しておく。同一の動作を表すイベントは、異なるシナリオ、異なるユースケースのものであってもかまわない。また、ある一連のイベントの中では、イベントを実行する順序を決めておく必要のない場合（どの順でイベントを実行してもかまわない場合）や、あるイベントの流れと別のイベントの流れが、並行して実行される必要がある場合がある。このような、実行順序を変更してもかまわないイベントや、並行して実行されるイベントも併せて指定する。

包含・拡張ユースケース

複数のユースケースで、あるイベントフローを共有している場合、そのイベントフローを独立したユースケースで記述することができる。あるユースケースのイベントフローの一部を別のユースケースを利用して表現することができ、この場合のユースケースの関係を包含関係という。図 2.1 では、「会員 ID を入力する」ユースケースが「ビデオを貸し出す」ユースケースに包含されるという関係が表現されている。ここで、図 2.1 より、包含されるユースケースは、図 2.2 より、「ビデオを貸し出す」ユースケースの「貸出処理」シナリオの Step 2 において実行されることがわかる。また、ユースケース図中においてユースケースの包含関係は、包含するユースケース（ビデオを貸し出す）から包含されるユースケース（会員 ID を入力する）に向かって破線のアークを結び、破線アーク上に”⟨⟨include⟩⟩” と記述することにより表す。

さらに、アプリケーションの選択的な振舞いと必須の振舞いを区別する場合に、選択的な振舞いを独立したユースケースで記述することができる。このユースケースの関係を拡張関係といい、図 2.1 では、「ポイントを還元する」ユースケースが「ビデオを貸し出す」ユースケースの拡張ユースケースであることを表している。この例では、「ビデオを貸し出す」ユースケースの「貸出処理」シナリオの Step 8 「料金を計算する」の直前で一定の条件を満たした場合に、「ポイントを還元する」ユースケースが実行されることを表している。この Step 8 のイベントを拡張点と呼ぶ。また、ユースケース図中においてユースケースの拡張関係は、拡張するユースケースを上下に分割し、上部にユースケース名、下部に拡張点を記述する。そして、拡張ユースケース（ポイントを還元する）から拡張元のユースケース（ビデオを貸し出す）に向かって破線のアークを結び、破線アーク上に”⟨⟨extend⟩⟩” と記述することにより表す。

副シナリオ

シナリオは、ユースケースにおける処理の流れの 1 つを記述したものである。しかし 1 つのユースケースにおける処理の流れは必ずしも 1 つとは限らず、複数存在する場合もある。従ってそれら複数の処理の流れをそれぞれシナリオとして記述し、ユースケースに複数のシナリオを持たせることが多々ある。この場合、最も一般的なイベントの流れを記述したシナリオを「主シナリオ」と呼び、主シナリオ以外のシナリオを「副シナリオ」と呼ぶ。特に、主シナリオの代替となるイベントの流れを記述したシナリオを「代替シナリオ」、エラーが発生するシナリオを「例外シナリオ」と呼ぶ。図 2.2 は、ユースケース「ビデオを貸し出す」の主シナリオである。また、このユースケースの副シナリオを図 2.3 及び図 2.4 に示す。図 2.3 は代替シナリオであり、たまったポイントを利用してビデオを借りる場合のシナリオである。図 2.4 は例外シナリオであり、度々貸出期間を超過するなどして不正利用の回数がたまってしまったため、貸出が拒否される場合のシナリオである。

シナリオ: ポイントによる貸出処理	
1. 「ビデオ貸出」を押す	8. 料金を計算する
2. include （会員 ID を入力する）	9. ビデオのタイトルを表示する
3. 不正利用履歴を調べる	10. 料金を表示する
4. ビデオ ID を入力する	11. 返却日を表示する
5. 貸出期間を選択する	12. 「了解」を押す
6. 「ポイント利用」にチェックする	13. 会員データベースを更新する
7. 「OK」を押す	14. ビデオデータベースを更新する

図 2.3: 副シナリオ例 1（代替シナリオ）

シナリオ: 不正利用履歴による貸出拒否処理
1. 「ビデオ貸出」を押す
2. include （会員 ID を入力する）
3. 不正利用履歴を調べる
4. 不正利用の回数が多いため、貸出拒否を通知する

図 2.4: 副シナリオ例 2（例外シナリオ）

2.2.2　GUI プロトタイプ

本手法で生成する GUI プロトタイプは、各ウィンドウに必要なウィジェットが過不足なく配置されていることを確認することと、各ウィンドウの外観を評価・決定するために用いることを仮定している。このため、各ウィンドウに配置するウィジェットや GUI の制御構造のみを生成し、各入力項目に対する制約等は生成しない。本手法では、生成された GUI は、実際にアプリケーションを利用するエンドユーザ自身の手によって評価することにより、エンドユーザにとって使いやすい GUI を構築することを前提としている。従って、評価を行うエンドユーザは、各ウィンドウに配置されているウィジェットの大きさや位置などの外観のみに注目をして評価をするべきであると考えている。ユーザに提示する GUI プロトタイプは、エンドユーザがユーザビリティのみに着目をして評価をすることができるものであることが重要である。

2.2.3　GUI プロトタイプ生成指針

本手法では、ユースケース図とシナリオからアプリケーションの GUI プロトタイプを生成する。ユースケース図とシナリオには、以下のような特徴がある。

- ユースケース
  - アプリケーションの機能を表現する
  - アプリケーションと外部との接点を表す
- シナリオ
  - イベントの発生順序を表す

つまり、1 つのシナリオでユースケース中の処理の流れの 1 つを表すため、ユースケースに対して記述されたシナリオを縒り合わせることにより、そのユースケースにおけるイベントの流れを把握することができる。従って、記述されたユースケースを全て縒り合わせることにより、アプリケーションの全体的なイベントの流れを把握することができる。そして、アプリケーションの全体的なイベントの流れから、GUI の制御構造を抽出することができる。

具体的に、シナリオを縒り合わせる方法として、同一動作を表すイベントの統合を行う。1 つのアプリケーションにおいて記述されるイベントは、全て異なる動作を表すわけ

ではなく、同一の動作を表すものも多々ある。図 2.2、図 2.3、図 2.4 の 3 つのシナリオに記述されているイベントの総数は 31 個である。これらのイベントを、異なる動作を表すもののみ抽出すると、

- 「ビデオ貸出」を押す
- 不正利用履歴を調べる
- 貸出期間を選択する
- 「OK」を押す
- ビデオのタイトルを表示する
- 返却日を表示する
- 会員データベースを更新する
- include（会員 ID を入力する）
- ビデオ ID を入力する
- 「ポイント利用」にチェックする
- 料金を計算する
- 料金を表示する
- 「了解」を押す
- ビデオデータベースを更新する
- 不正利用の回数が多いため、貸出拒否を通知する

の 15 種類となる。従って、シナリオのみでは図 2.5 のように単一のイベントの流れのみしか把握することができないが、同一動作を表すイベントを統合することにより、図 2.6 のような有効グラフを生成することができ、アプリケーションの全体的なイベントの流れを把握することができると考えられる。

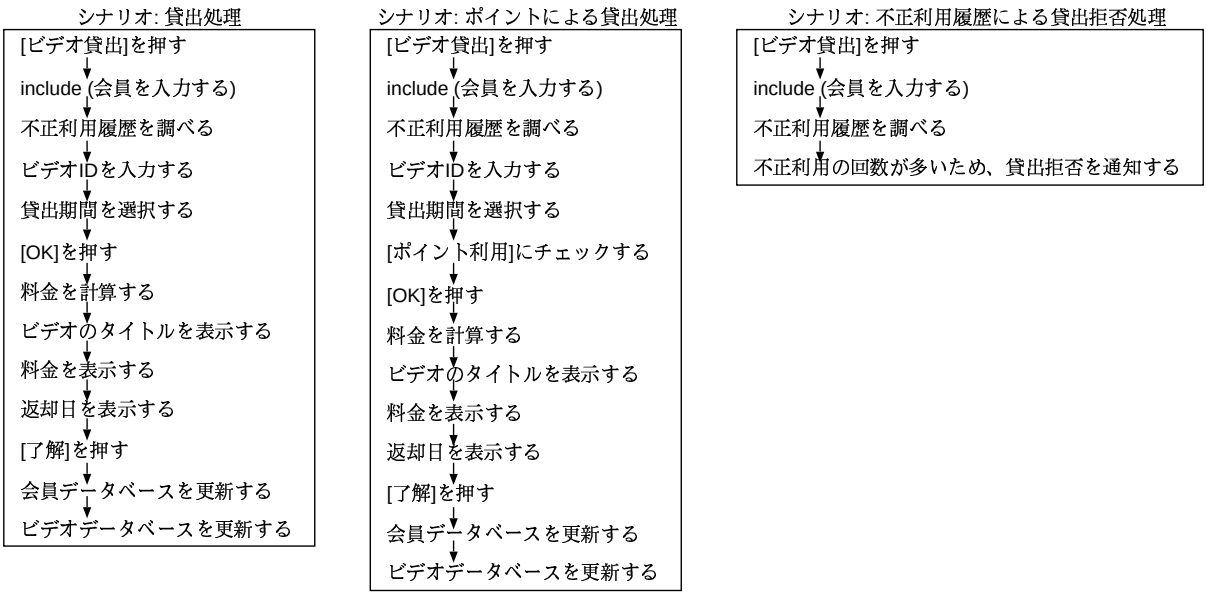


図 2.5: 個々のシナリオでのイベントの流れ

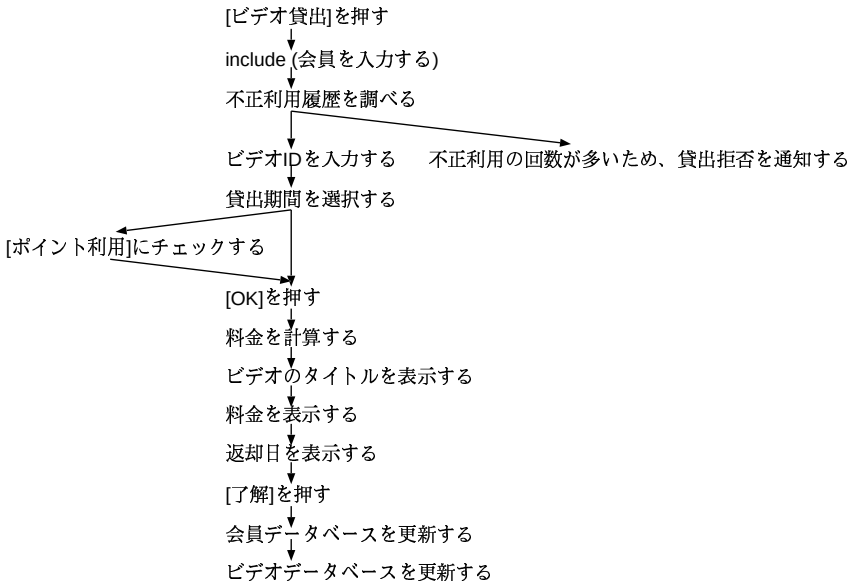


図 2.6: シナリオの縒り合わせによるイベントの流れ

2.3　GUI プロトタイプ生成手順

本手法では、以下の 5 つのステップで GUI プロトタイプを生成する。

Step 1 ユースケース図・シナリオ記述

Step 2 シナリオ併合グラフ生成

Step 3 シナリオ併合グラフ修正

Step 4 インタラクション項目とウィンドウ内構成設定

Step 5 GUI プロトタイプ生成

これら 5 つのステップに基づいて GUI プロトタイプを生成するためのシステム GUPPY（GUi Prototype Program generation sYstem）を開発した（図 2.7）。以下において、5 つのステップについて詳しく述べる。

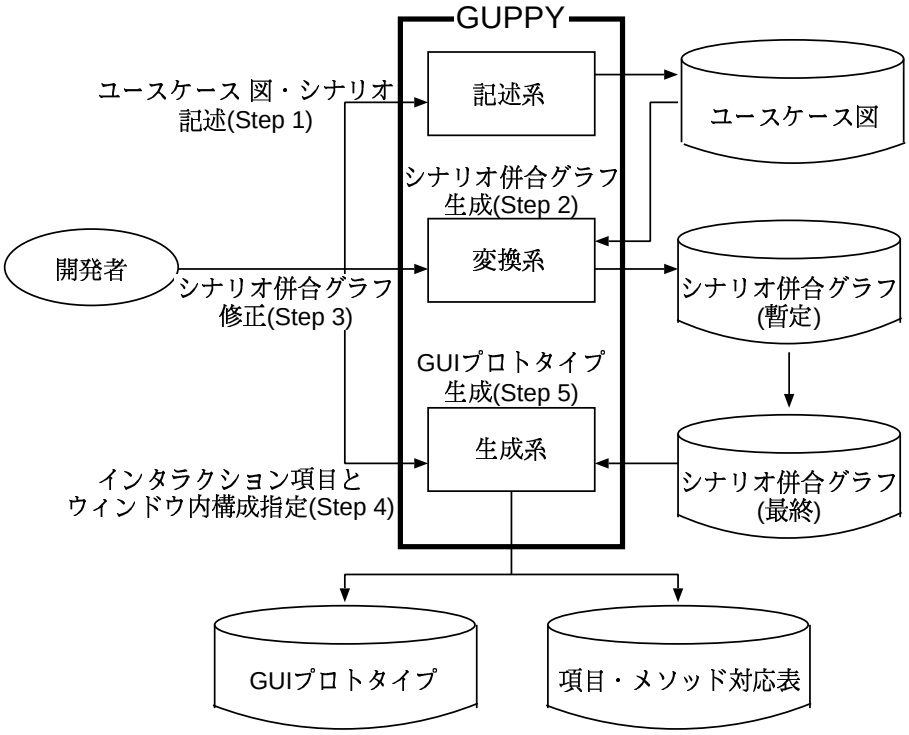


図 2.7: システム構成

2.3.1 ユースケース図・シナリオ記述

GUPPY を用いてユースケース図及びシナリオを記述する。2.2.1 で述べた、同一の動作を表すイベントや、実行順序が変更されてもかまわないイベント、並行して実行されるイベントもこの時点で指定する。

2.3.2 シナリオ併合グラフ生成

GUI プロトタイプを生成するためには、入力されたユースケース図及びシナリオから、アプリケーションの全体でのイベントの流れを抽出する必要がある。このために、シナリオ内のイベントをノードとし、イベントの発生順序をアークとして表現した有向グラフを生成する。このとき、全く同じ文字列であるイベントは同一の動作を表すものとみなし、

統合する。また、ユースケース図・シナリオ記述のステップで同一の動作を表すイベントと開発者によって指定されたイベントも同一ノードとして統合する。できあがった有向グラフをシナリオ併合グラフと呼ぶ。

シナリオ併合グラフを生成するためのアルゴリズムを以下に示す。

```
/* 同一動作イベントの特定 */
アプリケーション内の全イベントを、文字列として昇順に整列する;
for(int n = 1; n <= アプリケーション中の全イベント数; n++) {
  n 番目のイベントを読む;
  if(n-1 番目と n 番目のイベントは文字列として同じである)
    n 番目と n-1 番目のイベントは同一動作を表すという情報を格納する;
  else if(n 番目のイベントは、別のイベントと同じ動作を表すという情報が、
          開発者から与えられている)
    n 番目のイベントと、指定されている別のイベントは同一動作を表すという
    情報を格納する;
}
/* 同一動作イベントの統合 */
for(int k=1; k<= ユースケース数; k++) {
  for(int m = 1; m<= k 番目のユースケースが持つシナリオ数; m++) {
    for(int n = 1; n<= m 番目のシナリオが持つイベント数; n++) {
      n 番目のイベントを読む;
      if(n 番目のイベントに同じ動作を表すイベントが存在する) {
        同一の動作を表すイベント (E) を特定する;
        if(n-1 番目のイベントと E の 1 つ前に実行されるイベント (E') は
           同一動作イベントとして統合されている) {
          if((n-1 番目のイベントと n 番目のイベントは、この順に実行される)
            &&(イベント E' とイベント E は、この順に実行される)) {
            n 番目のイベントと E を統合する;
```

```
    } else if((n-1 番目のイベントと n 番目のイベントは、
                どちらを先に実行してもかまわない)
            &&(イベント E' とイベント E は、
                どちらを先に実行してもかまわない)) {
        n 番目のイベントと E を統合する; }
    } else if(n 番目のイベントと E は異なるユースケースに属する) {
      if(n 番目のイベントのユースケースと E のユースケースは、
          同一のアクターにより起動される) {
        if(n 番目のイベントのユースケースと E のユースケースは、
            包含関係でも拡張関係でもない)
            n 番目のイベントと E を統合する; }
    } else {
     if(n 番目のイベントと E は、異なるシナリオに属する)
        n 番目のイベントと E を統合する;
    } } } } }
```

シナリオ併合グラフ生成アルゴリズム

シナリオ併合グラフで用いられるアークの種類を図 2.8 に示す。また、上記のアルゴリズムを用い、2.2.1 章のシナリオ例から GUPPY によって生成されたシナリオ併合グラフを図 2.9 に示す。シナリオ併合グラフ上の 1 つ 1 つのノードがイベントを表す。

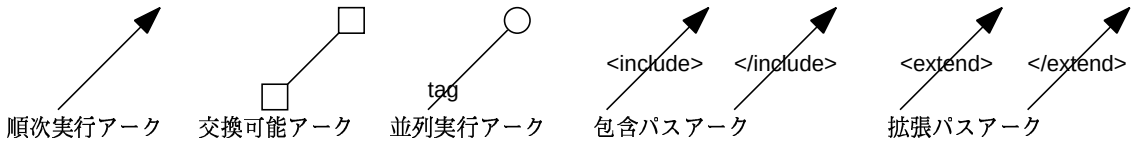


図 2.8: アーク定義

順次実行アーク

矢印の始点のイベントから終点のイベントへの順でイベントが実行される。

交換可能アーク

このアークで結ばれたイベントは、どちらを先に実行してもかまわない。

並列実行アーク

同じ “tag” を持つアークで結ばれたイベントのセットは、並行に実行される。

包含パスアーク

包含関係のユースケースのイベントの流れを表す。<include> で始まり、</include> で終わる。

拡張パスアーク

拡張関係のユースケースのイベントの流れを表す。<extend> で始まり、</extend> で終わる。

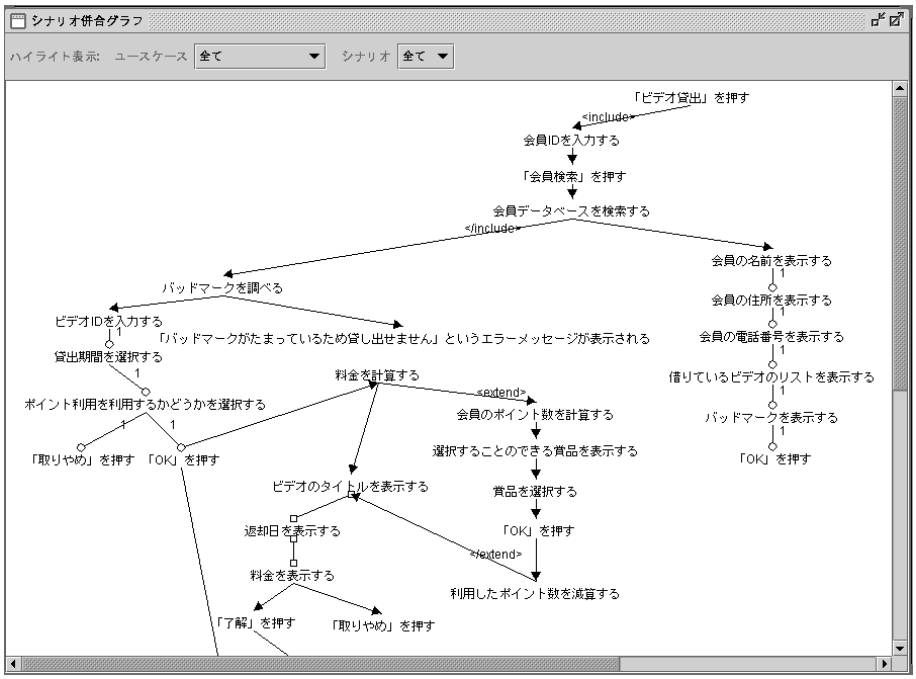


図 2.9: シナリオ併合グラフ例

シナリオ併合グラフは、開発者によって同一の動作を表すと指定されたイベント、及び文字列として全く同じであるイベントを統合することにより生成される。上記のアルゴリズムのオーダは、アプリケーション中の全イベント数を n とすると、イベントのソートにはシェルソートを用いているため、オーダは $O(n^{1.2})$、同一動作イベントの特定及び同一イベントの統合のオーダはそれぞれ $O(n)$ である。従って、上記のシナリオ併合グラフ生

成アルゴリズムのオーダは $O(n^{1.2})$ となる。シナリオ併合グラフの生成の際には、イベントに対して自然言語解析を行い、自然言語として同一の動作を表すイベントを探すことはしていない。このことにより、シナリオ併合グラフ生成アルゴリズムのオーダが $O(n^{1.2})$ ですみ、自然言語解析を行うよりも比較できないくらい短時間でシナリオ併合グラフを生成することができる。

2.3.3 シナリオ併合グラフ修正

同一の動作を表すイベントは、ユースケース図・シナリオ記述のステップで開発者によって必ずしも全て指定されているとは限らない。また異なる動作を表すイベントであっても、誤って同じ動作を表すものとして指定されていることがある。さらに GUPPY では、文字列として同じイベントを同じ動作を表すとみなし、統合しているが、同じ文字列であっても異なる動作を表す場合がある。そこで、生成されたシナリオ併合グラフを開発者に提示し、開発者が修正を行う。これにより、より適切なシナリオ併合グラフが得られる。

また、シナリオ併合グラフ生成・修正後にユースケース図やシナリオに変更があった場合、新しく生成するシナリオ併合グラフに、生成・修正済みのシナリオ併合グラフの情報、すなわち同一動作を表すイベントや、各ノードの座標などの情報を反映させることができる。

2.3.4 インタラクション項目とウィンドウ内構成設定

ユースケース図とシナリオのみでは、ユーザが入力する項目、ユーザに表示する項目、ユーザイベントが起こる項目（これらを総称してインタラクション項目と呼ぶこととする）、及び、それらの項目のうち、どの項目とどの項目が同一ウィンドウ上に存在するかという、GUI プロトタイプの各ウィンドウの構成情報を得ることはできない。そこで、シナリオ併合グラフ修正ステップの終了後に、これらの情報を指定する。この指定はシナリオ併合グラフ上で行う。

インタラクション項目

イベントの文字列の中からインタラクション項目を選択し、その項目が以下のうちのどの分類にあたるかを選択する。

- 入力項目
  - キーボードから入力される項目
    - 文字列を入力する項目
    - 数値を入力する項目
  - 幾つかの選択肢から選択される項目
    - 複数の選択肢の中から 1 つのみを選択する排他的な選択項目
    - 複数の選択肢の中から 1 つ以上を選択可能な項目

    またこの選択項目では、上記 2 つの分類の指定以外に、選択肢に関する指定も行う。選択肢が静的に決定しているか、あるいはアプリケーション実行中に動的に決定されるかを指定し、静的に決定している場合には、その選択肢のリストを入力する。
- 表示項目
  - 文字列が表示される項目
  - 画像が表示される項目

例えば「名前を入力する」というイベントでは、「名前」という項目がインタラクション項目であるとすると、この項目は入力項目であり、キーボードから文字列を入力する、という指定する。

ウィンドウ内構成

1 つのウィンドウに、インタラクション項目のうちのどの項目が属するかをイベント単位で指定する。例えば、「名前を入力する」、「『OK』を押す」という 2 つのイベントにおいて、「名前」、「OK」がインタラクション項目であり、この 2 つが同じウィンドウに属する場合、これら 2 つのイベントが同一ウィンドウを構成するという指定する。

2.3.5　GUI プロトタイプ生成

シナリオ併合グラフ修正ステップで完成したシナリオ併合グラフ及びインタラクション項目とウィンドウ内構成設定のステップで指定された情報をもとに GUI プロトタイプを

生成する。生成される GUI プロトタイプは、プログラミング言語 Java で記述される。各ウィンドウに配置するウィジェットは、インタラクション項目とウィンドウ内構成設定のステップで設定された項目の種類をもとに GUPPY が自動的に決定する。表 2.1 は、項目の分類と決定されるウィジェットとの対応例である。ウィジェットの候補が複数ある場合には、数値入力の項目であれば数値の範囲の大きさ、選択項目であれば選択の候補の項目数などに応じて決定される。現状では、GUPPY によって各項目に割り当てられるウィジェットは、Java の Swing[16] において提供されているウィジェットの一部のみであり、独自に開発したコンポーネントなどを割り当てることはしていない。

表 2.1: ウィジェット対応表

項目	形式		ウィジェット
入力	キーボード	文字列	JTextField
		数値	JSlider, JTextField
	項目選択	排他的選択	JRadioButton, JComboBox, JList
		複数選択可	JCheckBox, JList
表示		文字列表示	JLabel
		画像表示	JPanel
ユーザイベント			JButton, JMenu

本手法により GUI プロトタイプを作成し、それを評価・変更した後、実際のアプリケーションの GUI 部として利用する場合、 GUI 部とアプリケーションの処理部を連結し、データの授受を行えるようにしなければならない。GUPPY では GUI プロトタイプを生成する際に、ソースコード中に各ウィジェットへのデータの設定やデータの取得を行うメソッドを生成している。そのメソッドが、インタラクション項目とウィンドウ内構成設定のステップで指定されたどの項目に対応するか、そしてそのメソッドが果たす役割の簡単な記述などのメソッドと項目間の対応表（メソッド対応表）も併せて生成する。これにより、GUI 部とアプリケーションの処理部は、GUPPY により用意されたメソッドを利用してやりとりを行うことができる。従って、開発者は、メソッド対応表を参照することにより、GUI 部のプログラムの内部を理解する必要なく、GUI 部とアプリケーション処理部の連結を行うことができる。メソッド対応表は、XML を用いて記述される。

図 2.10 が GUPPY によって生成された GUI プロトタイプの 1 つであり、図 2.11 がこのプロトタイプと併せて生成されたメソッド対応表の一部である。対応表には、各項目に

割り当てられたウィジェットの種類とウィジェット名、開発者によって指定された項目名、その項目が属するユースケース・シナリオ・イベント名、そしてその項目へのデータの設定・取得のメソッドの一覧と、メソッドの簡単な説明が記述される。

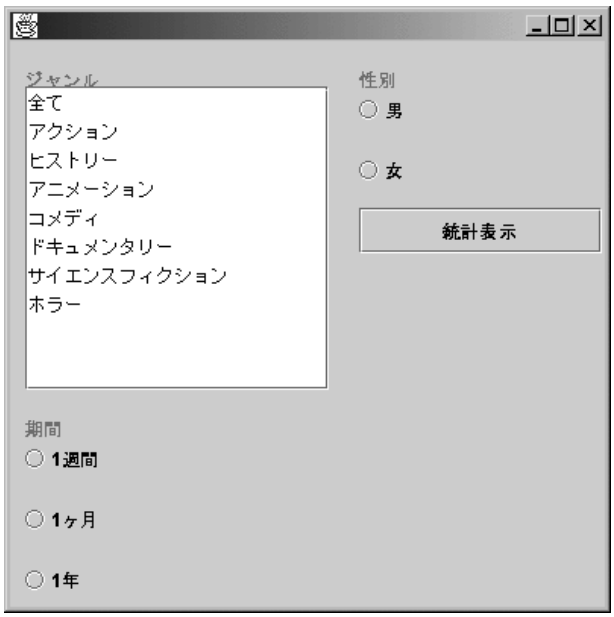


図 2.10: GUI プロトタイプ例

```
<ウィンドウ クラスファイル名="GeneratedPrototype40.java">
  <インタラクション項目 項目名="ジャンル"
    ウィジェットの種類="JList"
    ウィジェットの名前="list3"
    属するユースケース名="統計をとる"
    属するシナリオ名="ビデオの統計"
    属するイベント名="ジャンルを選択する">
      <メソッド メソッド名="public DefaultListModel getList3Model()"
        役割="「list3」に表示されているデータモデルを取得します。" />
      <メソッド メソッド名="public void setList3(Object[] data)"
        役割="「list3」に配列 data の要素のリストを表示します。" />
      <メソッド メソッド名="public void addList3Element(Object o)"
        役割="「list3」に表示されているリストにオブジェクト o を追加します。" />
  </インタラクション項目>
  ..............
</ウィンドウ>
```

図 2.11: メソッド対応表

このメソッド対応表に記述される情報（ノードとその意味）は、以下の通りである。

⟨**ウィンドウ**⟩**:** 生成された GUI プロトタイプ中のウィンドウを表す。このウィンドウに属する全てのインタラクション項目を子ノードとして持つ。属性は以下の通りである。

クラスファイル名: GUI プロトタイプでは、1 ウィンドウを 1 クラスとして生成する。この属性は、そのクラス名を表す。

⟨**インタラクション項目**⟩**:** ⟨ウィンドウ⟩: が表すウィンドウ上に配置されたインタラクション項目である。このインタラクション項目に割り当てられたメソッドを子ノードとして持つ。属性は以下の通りである。

項目名: このインタラクション項目の名前を表す。

ウィジェットの種類:　このインタラクション項目に割り当てられたウィジェットの種類を表す。

ウィジェットの名前:　このインタラクション項目に割り当てられたウィジェットの、プログラム中での名前を表す。

属するユースケース名:　このインタラクション項目が属するユースケースの名前を表す。

属するシナリオ名:　このインタラクション項目が属するシナリオの名前を表す。

属するイベント名:　このインタラクション項目が属するイベントの名前を表す。

⟨ **メソッド** ⟩**:**　⟨ インタラクション項目 ⟩: が表すインタラクション項目に割り当てられたメソッドである。属性は以下の通りである。

メソッド名:　このメソッドの名前である。

役割:　このメソッドが果たす役割についての簡単な説明である。

2.4　評価

GUPPY を使用して、2.2.1 章の例を含むレンタルビデオ業務のアプリケーション、文献 [14] に掲載されている通信販売会社のアプリケーション、及び本システム GUPPY 自身という 3 つのアプリケーションについて GUI プロトタイプを生成した。その結果について以下に述べる。

2.4.1　ウィンドウ内構成比較

さらに、上記 3 つのアプリケーションに対して、あらかじめ作成しておいたアプリケーションと生成された GUI プロトタイプについて、各ウィンドウに配置されたウィジェットを比較した。通信販売会社のアプリケーションでは、文献 [14] に、ユースケース例と共にウィンドウ例が掲載されていたため、そのウィンドウ内のウィジェットと比較した。その結果を表 2.2 に示す。ただし、GUI プロトタイプの各ウィンドウ上に配置されるインタラクション項目は、比較対象のアプリケーションと同様になるように設定した。「プロトタイプ中」は GUI プロトタイプにおいて生成されたウィジェットの総数、「アプリケーション中」は、作成されていたアプリケーション中で利用されているウィジェット数、「非生成」

は、作成されていたアプリケーションには存在しており、GUI プロトタイプには生成されなかったウィジェット数、「異種類」は、あらかじめ作成されていたアプリケーションと生成された GUI プロトタイプで、項目としては同じでありながら異なる種類で生成されてしまったウィジェット数である。ただしこれらのウィジェット数には、JPanel 等のウィンドウ内でのウィジェットの配置を整えるための敷き物として利用されていたウィジェットは含まれていない。

表 2.2: ウィジェット数比較

	貸ビデオ	通信販売	GUPPY
プロトタイプ中	169	61	232
アプリケーション中	191	81	203
非生成	22	5	15
異種類	42	19	77

表 2.2 において、「非生成」のうちの多くは、多くのウィンドウで上部に配置される、ウィンドウの目的を記したラベルであった。通信販売会社のアプリケーションでは、例えば「住所」という 1 つの項目で「郵便番号」と「住所」という 2 つ以上の項目が作成されている場合もあった。「異種類」の内訳の主なものを以下に示す。

1. 選択項目が一定数以上、あるいはアプリケーション実行時に選択項目が変化する場合には JList が生成されたが、アプリケーション中では JRadioButton あるいは JComboBox が利用されていた。

2. 文字列の入力・表示のために、GUPPY は JTextField 及び JLabel を生成したが、この文字列が複数行の入力・表示を必要とするものであったため、アプリケーション中では JTextArea が利用されていた。

3. アプリケーション中では、エラーや確認のメッセージの表示に、Java で用意されている、エラーや確認のメッセージを簡単に表示することのできるウィジェットを利用していたが、GUPPY では単なる表示項目として扱われ、JLabel が生成されていた。

「プロトタイプ中」と「アプリケーション中」のウィジェット数が異なるのは、上記 1. により、プロトタイプ中では JList という 1 つのウィジェットで項目選択を表したが、アプ

リケーション中では項目 1 つ 1 つが JRadioButton 等のウィジェットであったこと、プロトタイプ中には、上記で述べたウィンドウの目的を記したラベルが生成されていなかったこと、そして上記 3. により、プロトタイプ中ではエラーや確認のメッセージを単なる表示項目と JButton 等で生成されたが、アプリケーション中ではエラーや確認のメッセージのための 1 つのウィジェットが利用されていたためである。

表 2.2 の結果より、レンタルビデオ店のアプリケーションでは約 75%、通信販売会社のアプリケーションでは約 69%、GUPPY では約 67% のウィジェットが、あらかじめ作成されていたアプリケーションのウィジェットと一致した。従って、GUPPY によって生成される GUI プロトタイプは、各ウィンドウ上のウィジェット構成が実用に近い形で生成されると考えられる。この結果は、エンドユーザによるユーザビリティの確認のためには十分である。また、生成されたプロトタイプを洗練し、実際のアプリケーションの GUI 部とする場合の基盤としても十分利用することができると考えられる。

2.4.2　同一動作イベント数の計測

表 2.3 は、生成されたシナリオ併合グラフの傾向を示している。「イベント総数」は各アプリケーションにおけるイベントの総数、「あらかじめ指定」は開発者によって指定された、同一の動作を表すイベントの数、「文字列として同一」はシナリオ併合グラフ生成の過程で、文字列として同一であり、GUPPY によって同一動作を表すと見なされたイベントの数、「非同一」は開発者によって指定された同一動作を表すイベントも、文字列として同一イベントもないイベントの数を計測した結果である。ただし、文字列として同一のイベントであり、かつ開発者によって同一動作を表すと指定されたイベントは、「あらかじめ指定」として計測されており、「文字列として同一」には含まれていない。

表 2.3: 同一動作イベント計測結果

	貸ビデオ	通信販売	GUPPY
イベント総数	310	360	469
あらかじめ指定	104	130	134
文字列として同一	100	83	171
非同一	106	147	164

表 2.3 の結果より、「あらかじめ指定」のイベントと「文字列として同一」のイベント

を合計すると、ビデオレンタル業務のアプリケーションで約 66%、通信販売会社のアプリケーションで約 59%、GUPPY で約 65% が他のイベントと同一であることがわかる。このようにアプリケーション内で発生するイベントは、他のイベントと全く異なる動作を表すものよりも、何らかのイベントと同一の動作を表すものの方が多い。これにより、同一の動作を表すイベントを統合しシナリオ併合グラフを生成することで、アプリケーションのイベントの流れを抽出することができることがわかる。

2.4.3 自然言語処理技術によるシナリオ併合グラフ生成

シナリオ併合グラフの生成に自然言語処理の技術を取り入れる試みとして、奈良先端科学技術大学院大学で開発された形態素解析ツール茶筌 [17] を本システムに組み込んで実験を行った。茶筌を用いてイベントを語に分割し、その分割された語が同義語を持つ場合、その語を同義語を置き換えることにより、語単位で表現の異なるイベントも同一の動作を表すものとして統合をし、同義語の置換を行わない場合と行う場合でシナリオ併合グラフの生成に要する時間を計測した。その結果を表 2.4 に示す。「生成時間（同義語変換なし）」は、同義語の置換を行うことなくシナリオ併合グラフを生成する際にかかった時間（ミリ秒）、「生成時間（同義語変換あり）」は、同義語の置換を行ってシナリオ併合グラフを生成する際にかかった時間（ミリ秒）、及び「新たに同一と認識」は、同義語を置き換えることによって GUPPY が新たに同一と認識したイベントの数である。また、これらの時間は、それぞれのアプリケーションにおいてシナリオ併合グラフを 5 回ずつ生成し、各々の生成にかかった時間の平均を計算したものである。同義語の置換は、茶筌によって分解された語を、Jungle 社のデジタル類語辞典 [18] を用いて同義語を検索し、その結果を本システムの同義語のデータベースに登録をすることにより行った。

表 2.4: 同義語変換によるシナリオ併合グラフ生成結果

	貸ビデオ	通信販売	GUPPY
生成時間（同義語変換なし）（msec）	22	26	24
生成時間（同義語変換あり）（msec）	95690	101965	138381
新たに同一と認識	18	0	17

計測環境 CPU: Athlon 1.2GHz, Memory: 256MB

この結果より、同義語の置換を行わずにシナリオ併合グラフの生成は 20 ミリ秒程度の

時間ですむが、同義語の置換を行った場合には 2 分前後もの時間がかかり、置換を行わない場合に比べて生成に要する時間が大幅に増加していることがわかる。また、置換を行うことによって新たに同一の動作を表すと認識することのできたイベントは、イベント全体の数の 1% にも満たない。この結果より、同義語の置換を行った場合に要するこの時間は、実用に耐えうる範囲を大きく越えていると考えられる。従って、現状のコンピュータの性能において、シナリオ併合グラフ生成に自然言語処理の技術を取り入れることは現実的ではない。

2.4.4　インタラクション項目とウィンドウ内構成の設定手順

インタラクション項目とウィンドウ内構成設定の手順について、ビジュアルに GUI 構築することのできる、既存のツールと比較する。既存のものは通常、以下の手順に従う。

1. 各ウィンドウに配置する項目を決定する。
2. ウィンドウを作成する。
3. そのウィンドウ上に配置したい項目に割り当てるウィジェットを選択する。
4. ウィンドウ上の配置したい場所にそのウィジェットを置く。
5. ウィジェットの大きさや名前などを設定する。

GUPPY における項目設定やウィンドウ内構成設定の内容もこれと同じであり、その労力も、既存のツール利用時と同等である。

2.5　関連手法

GUI アプリケーションの開発支援に関しては、GUI のモデル化手法や RAD（Rapid Application Development）ツール、タスクモデルの構築による GUI プログラムの自動生成、仕様からの GUI プログラムの自動生成などが研究・実用化されている。また、シナリオ記述に関しても研究が行われている。

GUI のモデル化手法

GUI のモデル化手法に関しては、UML の GUI への拡張記法 [19] などが提案されている。しかし、これらはモデル化されたものをプログラミングに結び付けることに関しては支援されていない。

RAD ツール

VisualBasic[20] や Delphi[21] などが開発・製品化されている。これらは、各ウィンドウにビジュアルにウィジェットを配置していくことができるため、容易に GUI を構築することができる。しかし大規模なアプリケーションでは、設計と実装を誤りなく結び付けることが重要となってくる。これは、大規模なアプリケーションにおいて設計と実装の間に誤りが起きた場合には、その修正に大きなコストが必要となるためである。しかしこれらのツールでは、設計と実装の間のマッピングは支援していないため、これらは大規模なアプリケーションの開発には不十分であると考えられる。

仕様記述からの GUI プログラムの自動生成

ユースケースのシナリオとしてコラボレーション図を記述し、そこからアプリケーションの GUI 部のプロトタイプを生成するという研究 [8] や E-R モデルでアプリケーション内のデータ間の関係や各データの属性などを記述し、ウィンドウ遷移などの GUI の制御構造をペトリネットで記述することにより、GUI を生成するという研究 [22] が提案されている。これらは、GUI 部のプログラムの生成のための特別な記述を付加したり用意しなければならず、開発者に大きな負担をかけると考えられる。また、生成されたプログラムにはアプリケーションの全体的な GUI の制御構造が含まれていないものもあり、本手法が目指す GUI プロトタイプとしては、実用性に欠けると考えられる。

タスクモデルによる GUI プログラムの自動生成

アプリケーション内のユーザイベントや処理を、タスクと呼ぶ小さな単位でモデル化し、そこから GUI 部のプログラムの生成を行うということが提案されている [10][11]。しかしこれらは、実行順序が変わってもかまわないタスクの流れや、あるいは並行して実行されるタスクの流れに関しては支援していない。

シナリオ記述形式

本手法で GUI プロトタイプ生成に用いるシナリオは、自然言語で記述したものを対象とする。シナリオの記述形式の 1 つとして Use Case Maps[23] が提案され、利用されてきている。Use Case Maps は、シナリオ内の同一の動作を表すイベントを統合しイベントの流れを図示することにより、ユースケースを構造化し、視覚化するための記述形式である。これには、記述を支援するツールも開発されている（UCM Navigator[24]）が、記法を学ばなくてはならない、手動でユースケースを構造化し

なければならないなど、Use Case Maps を利用することは容易ではない。本手法では、記述したシナリオをそのままアプリケーションのエンドユーザに提示し、エンドユーザが GUI プロトタイプを評価することを想定している。従って本手法では、アプリケーション開発に関する知識を持たないエンドユーザにとっても理解しやすい記述形式である必要があり、この点で Use Case Maps より本手法の方が適していると考えられる。

2.6　あとがき

本手法では、エンドユーザの視点を取り入れて GUI の設計・実装を行うことを目的として、ユースケース図及びシナリオから GUI のプロトタイプの自動生成を行う手法について提案した。本手法では、仕様であるユースケース図及びシナリオを用いて GUI 部のプログラムを生成するため、アプリケーションの設計結果と GUI 部の実装の間に矛盾がないことを保証することができる。またシナリオは自然言語で記述されているため、アプリケーション開発に関する知識を持たないエンドユーザであっても理解しやすいものである。従って、エンドユーザがプロトタイプの評価をする際に、生成に用いたユースケース図とシナリオを、プロトタイプと矛盾のない操作手順を示したドキュメントとしてそのままエンドユーザに提示することができる。

今後の課題としては、

- 生成可能なウィジェットの種類を増やすこと
- ウィンドウ内の入力・表示項目構成に対して、ウィジェット構成の異なる複数のウィンドウを生成し、エンドユーザの選択の幅を広げること
- 生成された GUI プロトタイプを用いたアプリケーション開発の支援手法を確立すること

などが挙げられる。

第3章　設計結果の実装への自動反映手法

3.1　はしがき

近年，豊かなグラフィカル・ユーザ・インタフェース（GUI）を持ったソフトウェアが多く求められている．しかし，GUI アプリケーションの構造は非常に複雑であるため，開発者にとって大きな負担となっている．この GUI アプリケーション開発においては，次の 2 点が特に重要であると考えられる．

条件 1 アプリケーション全体の中で，各処理局面に応じて，必要とされる構成要素（ウィジェットと呼ぶ）を持ったウィンドウを配置すること

条件 2 人間工学に基づいた機能性や操作性を考慮し，ユーザの要求や好みを反映した（以降，ユーザビリティと呼ぶ）GUI が与えられること

従来のアプリケーション開発では，ウィンドウを 1 つ 1 つ個別に作成し，それを組み合わせることによってアプリケーションを開発するため，ウィンドウ間の関連やウィンドウを越えたウィジェット間の関連は考慮することが困難であり，結果として用途の同じウィジェットを複数作成したり，必要なウィジェットを欠いてしまうなど，アプリケーションの正常な動作に支障をきたすことがある．従って，各ウィンドウに適切なウィジェットを配置するための支援が必要となる．また，GUI のレイアウトは，ユーザが感じるアプリケーションの使いやすさと密接に関わっており，どのようなレイアウトであれば使いやすいかということは，個々のユーザの好みや経験等に大きく依存する．これまでの GUI 生成系によって自動生成されるウィジェットの配置・形状は，予め定められたアルゴリズムにより配置・形状の決定を行っており，ユーザの特殊性を反映することができない．本手法では，この 2 点を満たすソフトウェアを開発するための一つの手法として，GUI アプリケーションの生成に対する新しいアプローチを試みる．

本手法では，次の 2 点を中心とする開発支援システムについての提案を行う．

- 条件 1 を満たすために，アプリケーション全体のダイアローグ記述から，必要な要素が全て包含された GUI と，アプリケーションの制御の枠組みを生成する．

- 条件 2 を満たすために，本システムが生成した GUI を，ユーザの要求を満たすように，既存のツールを使って容易にユーザの要求を反映できる．

3.2　本手法の概要

本手法では、与えられたダイアローグ記述を入力として GUI アプリケーションの骨組みを自動生成する。アプリケーションの使いやすさに大きくかかわるアプリケーションの外観については、個人の好みや経験等に大きく依存するため、外観についての情報を詳細に与えることはせず、自動生成後に、インタフェースビルダ等の GUI の外観設定をすることのできる既存のツールを用いてユーザの要求を反映する。

ダイアローグ記述の方法としては、定義が単純なペトリネット [35] を選んだ。これは、設計結果からのソフトウェア生成と同様に、GUI の自動生成の可能性の追求を主目的にし、これに対して実用性をそこなわないようなアーキテクチャ、入力方式を模索してきた結果、GUI アプリケーションの構造を記述するためにはペトリネットが適しているという結論に至ったためである。特に、多くの GUI アプリケーションでは、ユーザが複数の動作を選択的に起動し、それらを並行して操作できるように構築されており、ペトリネットでは、このような並行動作を容易に記述することができる。また、GUI アプリケーションは、ユーザがウィンドウに対して入力を行うと、それに応じて何らかの処理が行われるというイベント駆動型のものが多く、ペトリネットではトランジションを用いて、そのようなアプリケーションの挙動を表現することができるため、GUI アプリケーションの構造を記述するために適している。

以下において本手法の特徴について述べる。

3.2.1　外観変更

本システムでは、設計時に記述したペトリネットをもとに、仮の GUI を自動生成する。こうすることにより設計時と実装時の GUI の構成が一致することが保証される。しかしこの際の GUI の外観は、本システムが自動的に決定するため、ユーザビリティは十分には考慮されていない。そこで、GUI アプリケーションの外観については、GUI が自動生成された後に、開発者がユーザと協議をし、ユーザビリティを考慮して変更を行う。この外観変更には特別なツールを必要とはせず、XF[32] や Tclbombur[36] などの既存のツールを用いて行うことができる。この利点は以下の通りである。

- アプリケーション設計時に必要なものは、アプリケーションの制御構造と、各制御の局面におけるウィンドウの構成情報であり、細かなウィンドウの外観の設定情報は不要であるべきである。
- 個々のウィンドウの外観について細かく、簡単に設定できるツールはすでに多く存在するため、ウィンドウの中に必要な構成要素が含まれていれば、外観設定は容易にできる。

3.2.2 アプリケーション開発における役割分担

GUI アプリケーション開発に携わる人を、

- ダイアローグ記述担当者
- GUI 定義担当者
- プログラミング担当者

の 3 つに役割分担することを想定している。ダイアローグ記述担当者は、アプリケーション設計に関する十分な知識を持っており、アプリケーションのダイアローグ部の決定を担当する。GUI 定義担当者は、ユーザビリティ、例えば誤操作を減らすためにボタンやメニューを大きくする、あるいはボタンを小さくしてワークスペースを大きくする、等について考慮し、GUI アプリケーションの外観を既存のツールを用いて変更することが業務である。プログラミング担当者は、アプリケーション本体の実装を行う。以上より、最初にダイアローグ記述担当者がダイアローグを記述すると、GUI 定義担当者とプログラミング担当者は、それぞれ独立して作業を行うことができる。また、これらを担当する人の資質は互いに異なっており、明確な役割分担は、アプリケーションの効率的な開発において重要である。

3.2.3 ペトリネットの記法

通常のペトリネットにはウィンドウの構成等の GUI に関する情報が書かれておらず、また、GUI アプリケーションの制御構造を簡潔に記述するには不十分である。そのため本手法では、

- GUI アプリケーションの各処理局面で必要なウィジェットを定義し、そのウィジェットの最少限の情報をペトリネットに付加する。
- GUI アプリケーション開発に適したペトリネットの記法を新たに定義し、それを用いる。
- ペトリネットを記述するための様々な機能を持った専用エディタを構築し、それを用いてペトリネットを記述する。

3.2.4　ステートマネージャの生成

既存の手法では、設計結果を実装に反映させる際には手作業でマッピングを行わなければならない。この場合、設計結果と実装結果が一致しないという結果が起こり得る。そこで本手法では、自動生成された GUI アプリケーションを、記述されたペトリネットに忠実に従って動作させるために、ステートマネージャと呼ぶ機構を用意した。ステートマネージャには、トランジションに対応づけられた全ての関数が登録されており、ステートマネージャはアプリケーション実行中のトークンの位置を監視し、発火条件の整ったトランジションの関数を起動する。ステートマネージャをアプリケーションに埋め込むことにより、自動生成された GUI アプリケーションは、ペトリネットに忠実に従って動作することが保証され、設計結果と実装結果が一致することとなる。これによりプログラミング担当者は、トランジションに対応づけられている関数に関しては、その処理内容のみを記述すればよく、関数間の呼び出し等の関係には注意を払う必要がなくなる。

以上より、GUI アプリケーションの開発に適した環境のもとでダイアローグ記述を行うことができる。

3.3　アーキテクチャ

本システムの全体的な構成図を図 3.1 に示す。

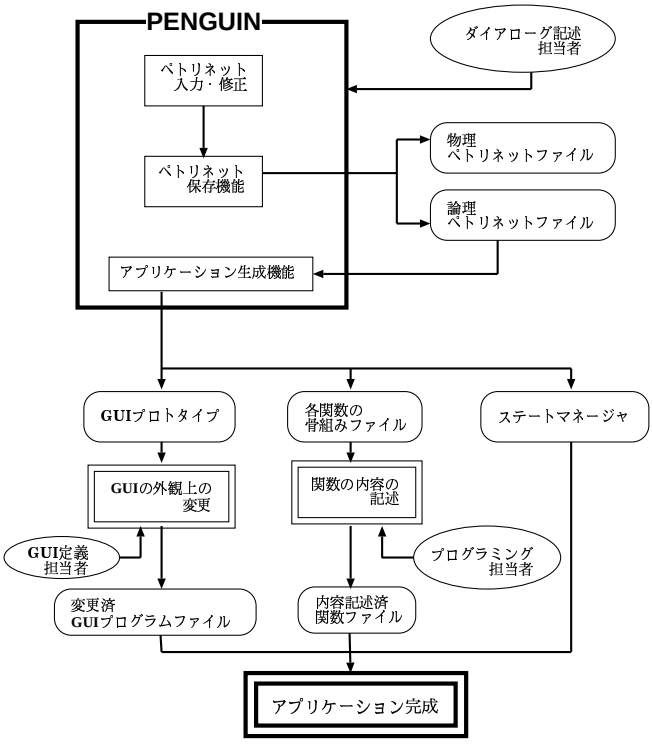


図 3.1: 全体的な構成図

本システムの構成では、まず、新たに開発したペトリネットエディタ “PENGUIN”（Petrinet Editor for Navigator and Graphical User INterface）を使って、ダイアローグ記述担当者がペトリネットを記述する。この結果を保存すると、PENGUIN は、プレース、トランジション、アークの座標を記述したファイル（物理ペトリネットファイルと呼ぶ）と、そのプレース、トランジション、アークの接続情報をテキスト形式で表現したファイル（論理ペトリネットファイルと呼ぶ）を作成する。このうち、論理ペトリネットファイルを PENGUIN に用意されている GUI 生成系にかけると、

1. 各関数の骨格（処理内容の書き込まれていない関数とそのつながり）
2. GUI プロトタイプ（ウィジェットの生成、ウィジェットに対するイベントの生起などの GUI に関するプログラム）
3. ステートマネージャ

の 3 種類のプログラムが生成される。プログラミング担当者が実際の関数の中身（GUI 部分以外の処理内容）を 1. のプログラムの中に記述し、GUI 定義担当者が 2. の GUI プロトタイプを変更する。ステートマネージャには開発者側の変更は必要なく、この 3 者を結合することによりアプリケーションが完成する。

3.4　ペトリネット記述法

図 3.2 に、PENGUIN のメインメニューを示す。このメインメニューにおいて”File” から”New” を選択し、アプリケーション名を入力し、作成するファイル名を入力することでペトリネットを記述するキャンバスが表示される。ペトリネットは、そのキャンバス上にビジュアルに記述する。すでに作成してあるペトリネットを編集する場合は、メインメニューから”File” の”Load” を選択し、アプリケーション名と作成してあるファイル名を入力する。ペトリネットが 1 つのキャンバスで収まらない場合には、複数のキャンバスに分割して記述することができる。

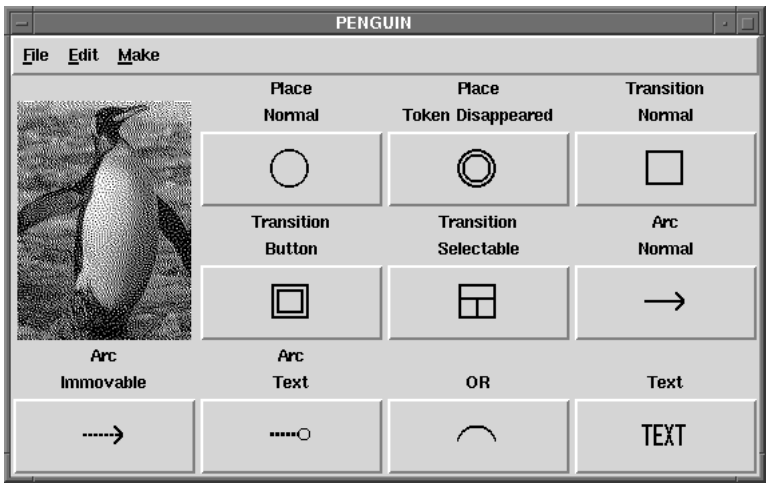


図 3.2: PENGUIN のメインメニュー

3.4.1　PENGUIN でのペトリネットの表記法

図 3.3 に標準的なペトリネットの表記法を示す。ペトリネットとは多くのシステムに適用可能なグラフィックで数学的なモデル化ツールであり、有効グラフの一つである [35]。

ペトリネットは、プレース、トランジション、アーク、トークンという 4 つの要素から構成される。

プレース システムの静的な状態を表し、事象生起の条件となる。

トランジション 事象を表す。トランジションが生起することを、トランジションが発火する、という。

アーク プレースとトランジションを結び、状態遷移の方向を表す。

トークン プレース中に存在し、システムの動的な状態を表す。トランジションが発火することでプレースからプレースへと遷移する。

PENGUIN では、プレースを円、トランジションを四角形、アークを矢印で表記する。

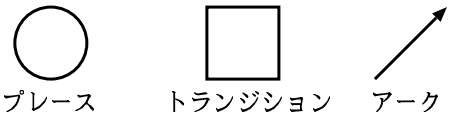


図 3.3: ペトリネットの表記法（1）

PENGUIN で使用するペトリネットは、一般的によく使用されるペトリネットであるプレース・トランジションネットに、抑止アークを加えた拡張ペトリネット [39] と呼ばれるペトリネットに、GUI アプリケーションに頻繁に使用される、又は記述に必要となるペトリネットの表記法を追加定義したものである。抑止アークとは、このアークの入力プレースがトークンを持つ場合には、その出力トランジションを発火させないというアークである。図 3.4 に PENGUIN でのプレース、トランジション、アーク及び抑止アークの表記法を示す。

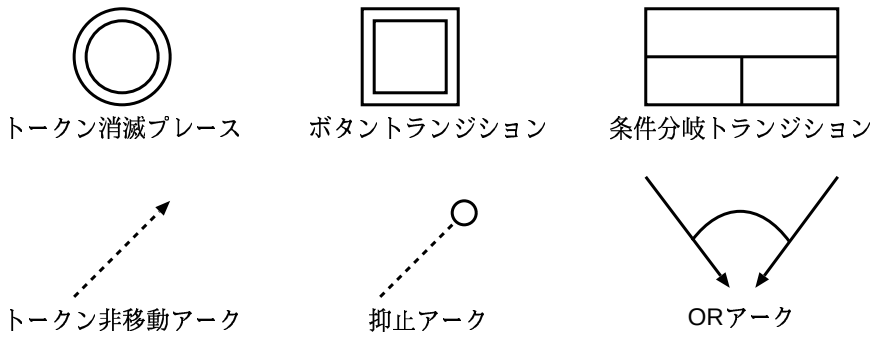


図 3.4: ペトリネットの表記法（2）

ボタントランジション　GUI アプリケーションでは、表示されたウィンドウ上のボタンを押す、またはメニューから選択することによって、何らかの処理が行われることが多々ある。ボタントランジションとは、そのような場合に使用されるトランジションであり、ボタンが押されたりメニューが選択されることによって発火するというトランジションである。GUI アプリケーションでは、ウィンドウ上のボタンが押されたりメニューが選択されることよって、関数が呼び出されたりウィンドウが表示されるなど何らか処理が行われることが非常に多い。図 3.5 にボタントランジションの使用例を示す。

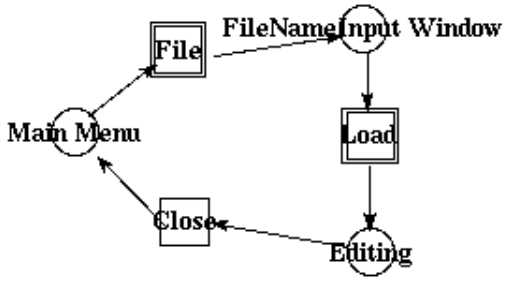


図 3.5: ボタントランジションの例

この例は、ファイルを開き、それを編集する場合の処理の流れを表している。まず最初に、ファイルを開くために”File” というメニューを選択する。そして表示されたウィンドウにファイル名を入力し、”Load” というボタンを押すと、ファイルが開いて編集をすることができる。図 3.5 の例では、”File” というメニューを選択する際のトランジション ”File” と、ファイルを読み込む際のトランジション”Load” にボタントランジションが使用されている。

条件分岐トランジション

ある処理が行われた後、その処理結果等によって次に行われる処理が異なる場合が多々ある。条件分岐トランジションとは、そのような場合に使用されるトランジションであり、アプリケーション内のある処理が行われる前後の条件によって、このトランジションが発火後にトークンが移動するプレースが決まるトランジションである。トランジション自身の名前を上段に、分岐条件の名前を下段に記述する。図 3.6 に条件分岐トランジションの使用例を示す。

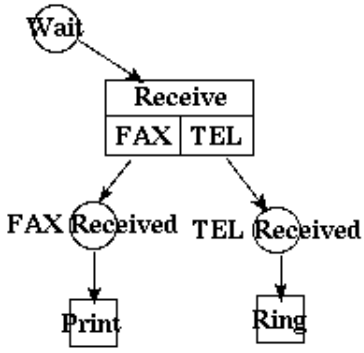


図 3.6: 条件分岐トランジションの例

これは、ある FAX の処理の流れの一部を表している。FAX が受信待ちの状態にある際にデータが送られてきた場合、そのデータが FAX のデータならばその内容を印刷し、送られてきたデータが電話のデータならば、呼び出し音を鳴らす、というものである。ここでは、FAX が受けとったデータが FAX のデータであるか電話のデータであるかによって、次に行われる処理が異なる。従って、データを受けとることを表した”Receive” というトランジションに条件分岐トランジションが使用されている。

トークン非移動アーク

GUI アプリケーションでは、例えばメニューから順次複数の処理を選択して、それらの処理を並行して行うことができる場合が多い。この場合、メニューが配置されているプレースには、必ずトークンが 1 つ存在しなければならない。トークン非移動アークとは、このような場合に使用されるアークであり、アークの終点のトランジションが発火すると、そのトランジションの出力プレースと、このアークの入力プレースの双方にトークンを移動させるアークである。図3.7にトークン非移動アークの使用例を示す。

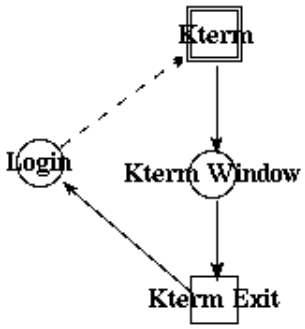


図 3.7: トークン非移動アークの例

この例は X ウィンドウのシステムの一部であり、ウィンドウを開く際の処理の例である。ユーザがログインした状態で、ポップアップメニューから”Kterm” を選択すると、kterm ウィンドウが表示される。この例では、kterm ウィンドウを開くトランジションは”Kterm” である。そして kterm ウィンドウはいくつでも表示させることができるため、kterm ウィンドウが 1 つ表示された後もプレース ”Login” には、トークンが残る必要がある。従って、プレース ”Login” とトランジション ”Kterm” を結ぶアークにトークン非移動アークが使用されている。

抑止アーク

何らかの処理を行う際に、その処理は、他のある処理が行われていない場合に行われる、という条件が存在する場合がある。抑止アークは、このような場合に使用されるアークであり、アークの始点のプレースにトークンがあると、その終点のトランジションが発火できないというアークである。図 3.8 に抑止アークの使用例を示す。

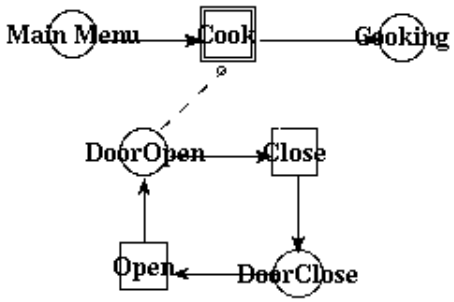


図 3.8: 抑止アークの例

この例は、電子レンジの処理の流れの一部を表したペトリネットである。電子レンジは、”Cook” というボタンを押すと調理が始まる。しかし、電子レンジのドアが開いた状態では調理を始めることができない。従って、電子レンジのドアが開いた状態であることを表すプレース ”DoorOpen” というプレースと、調理を始めるトランジション”Cook” を抑止アークで結び、ドアが開いている状態では調理を始められないようにしている。

OR **アーク**

異なる 2 つの処理のうち、どちらか 1 つの処理が行われれば、ある処理を行うことができる場合がある。OR アークとは、このような場合に使用されるアークであり、あるトランジションの異なる 2 つ入力プレースとトランジションの間のアークを結び、その 2 つのプレースのうち、どちらか一方にトークンがあれば出力トランジションを発火させるというアークである。これを標準記法で記述すると、ペトリネットの図が複雑になってしまうため、このアークを定義した。図 3.9 に OR アークの使用例を示す。

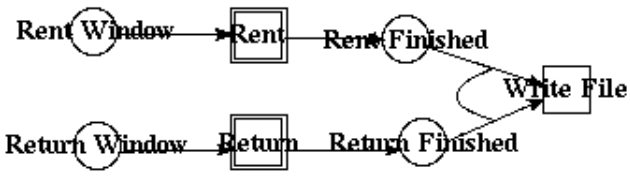


図 3.9: OR アークの例

この例は、レンタルビデオ店の管理アプリケーションのうち、ビデオの貸出の際の処理の一部と、返却の際の処理の一部を表している。貸出の処理と返却ではともに、処理が終った後、ファイルにデータを書き込む。この場合、ファイルへのデータの書き込みは、貸出の処理、または返却の処理のどちらかが終った後に行われるため、貸出または返却の処理が終ったことを表すプレース ”Rent Finished” ・ ”Return Finished” とファイルに書き込むことを表すトランジション ”Write File” を結ぶ 2 本のアークを、OR アークで結んでいる。

トークン消滅プレース

GUI アプリケーションでは、メニューから順次複数の処理を選択して、それらの処理を並行して行っていくことができる場合がある。そのような場合、そのメニューが配置されているプレースは、常時 1 つのトークンを持っておく必要があるが、同時にそのプレースは、複数のトークンを持っていてはならない。トークン消滅プレースとは、このプレースに複数のトークンが配置されると、そのうちの一つのみを残してその他のトークンを消すというプレースである。図 3.10 にトークン消滅プレースの使用例を示す。

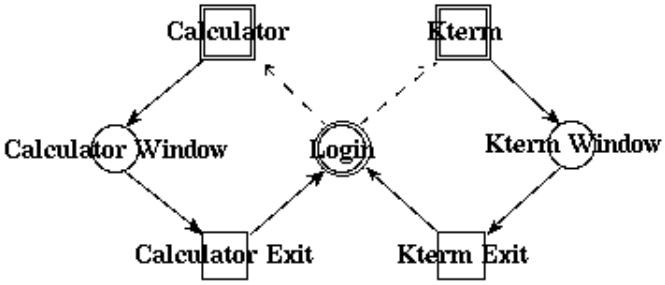


図 3.10: トークン消滅プレースの例

これは、X ウィンドウのシステムの一部で、ユーザがログインした状態で、ポップアップメニューから ”Kterm” を選択すると kterm ウィンドウが開き、”Calculator” を選択すると電卓ツールが表示される、という処理の流れである。”Kterm” を表示しても電卓ツールや別のウィンドウを表示することができ、また、逆も行うことができるので、ログインした状態を表すプレース ”Login” から、kterm や電卓を表示するトランジション ”Kterm” 及び ”Calculator” を結ぶアークにはトークン非移動アークが用いられている。また、ポップアップメニューから一度に選択することができるものは 1 つだけなので、プレース ”Login” にトークン消滅プレースが使用されている。

3.4.2 詳細情報の入力

PENGUIN においてペトリネットが保持する情報は、プレースが保持する情報、トランジションが保持する情報及びアークが保持する情報の 3 種類となる。それらの情報のうち、GUI 以外の情報を、それぞれプレースの詳細情報、トランジションの詳細情報及びアークの詳細情報と呼ぶ。

プレースの詳細情報

図 3.11 に、PENGUIN のプレースの詳細情報設定ウィンドウを示す。

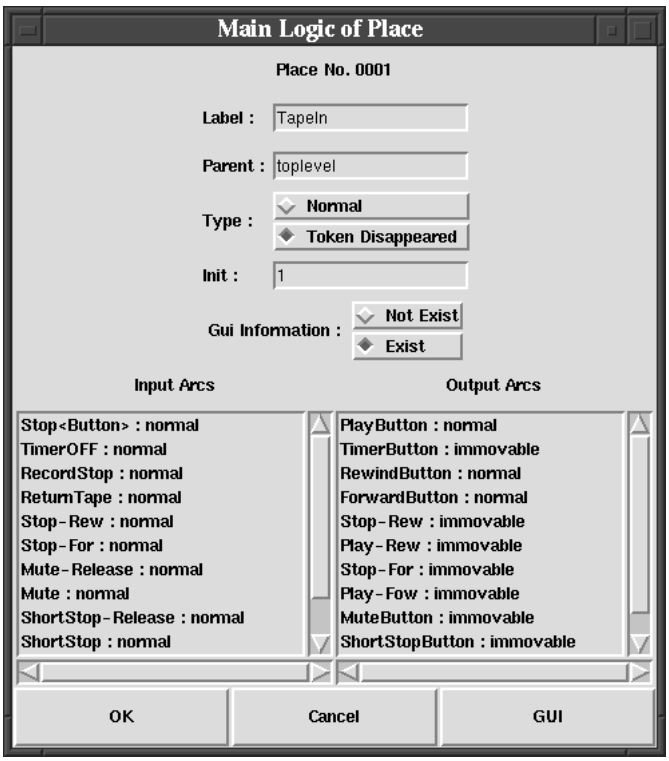


図 3.11: 詳細情報設定ウィンドウ（プレース）

ここで設定する情報及び表示される情報は以下の通りである。

- 個々のプレースを識別するための名前
- プレースが階層構造になっている場合、その親になっているプレースの名前
- プレースの種類（標準のプレースまたはトークン消滅プレース）
- GUI アプリケーション起動時にプレースが保持するトークンの個数
- GUI に関する情報の有無
- 入力トランジションとアークの種類のリスト
- 出力トランジションとアークの種類のリスト

トランジションの詳細情報

図 3.12 に、PENGUIN のトランジションの詳細情報設定ウィンドウを示す。

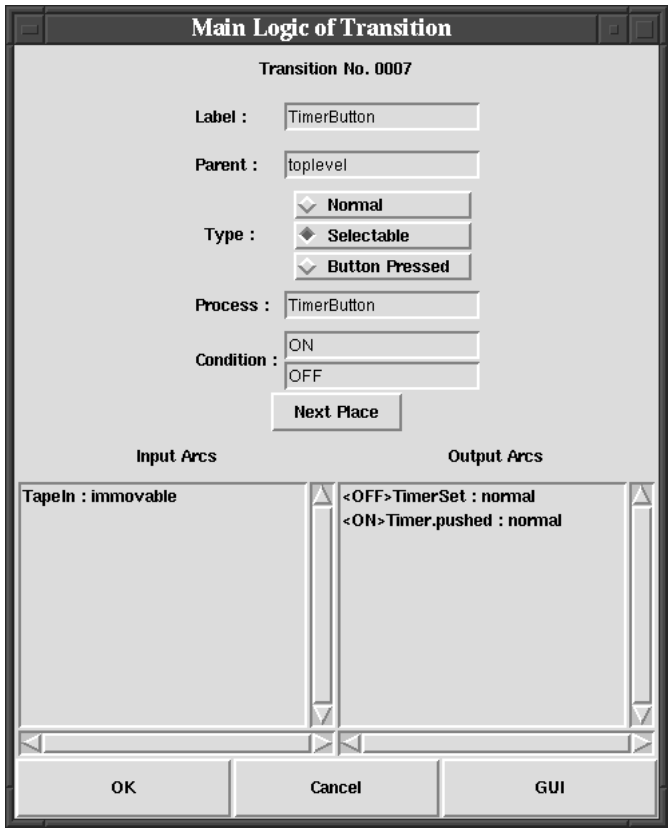


図 3.12: 詳細情報設定ウィンドウ（トランジション）

ここで設定する情報及び表示される情報は以下の通りである。

- 個々のトランジションを識別するための名前
- トランジションが階層構造になっている場合、その親になっているトランジションの名前
- トランジションの種類（標準のトランジション、ボタントランジション、または条件分岐トランジション）
- トランジションが発火することによって呼び出される関数名

- 分岐の条件部分の名前（条件分岐トランジションのみ）
- 出力プレースの種類、個数とその間のアークの種類の指定
- 入力プレースとアークの種類のリスト
- 出力プレースとアークの種類のリスト

3.4.3　アークの詳細情報

アークが保持する詳細情報は、以下の通りである。

- アークの種類（標準のアーク、抑止アーク、またはトークン非移動アーク）
- トランジションの発火によって移動するトークンの数（標準のアークのみ）

3.5　GUI 情報

本手法では、ペトリネットに GUI に関する情報を持たせることによって GUI アプリケーションの骨組みの自動生成を行う。GUI に関する情報は、主に、各ウィンドウに配置するウィジェットの情報と、各ウィジェットに対するユーザイベントのバインドに関する情報がある。本手法では、ウィジェットに関する情報はプレースに、ユーザイベントのバインドに関する情報はトランジションに持たせる

3.5.1　プレースの GUI 情報設定

図 3.13 に、PENGUIN のプレースの GUI 情報設定ウィンドウを示す。

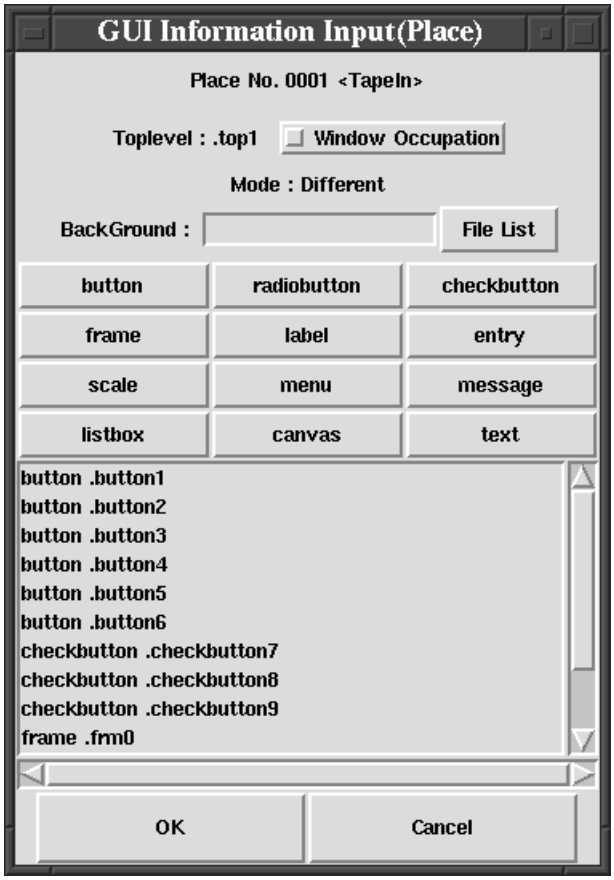


図 3.13: GUI 情報設定ウィンドウ（プレース）

プレースには GUI 情報としてウィジェット情報を持たせる。一つのプレースに、アプリケーション内の一つのウィンドウを対応させ、そのウィンドウに配置するウィジェットを、ダイアローグ記述担当者が指定する。指定できるウィジェットとしては、ボタン、エントリ、メニューボタンなど 12 種類を用意している。指定されたウィジェットの一覧は、このウィンドウ中央にリストとなって表示される。ダイアローグ記述担当者は、そのプレースに必要なウィジェットを指定し、ウィジェットの名前やそのウィジェットでイベントが起こった場合の返値など、そのウィジェットに関する最少限の情報を入力する。このダイアローグ記述の段階では、ウィジェットの大きさやウィンドウ上での位置、色等の細かな外観に関する設定は行わない。具体的に、ダイアローグ記述の段階で設定する情報を設定しない情報は、以下の通りである。

設定する情報

- 各ウィンドウに配置するウィジェットの種類、個数と各ウィジェットが持つべき最少限の情報（ウィジェットの名前、ウィジェットにイベントが起こることによって返される値等）
- このウィンドウが表示されているときに、他のウィンドウへの操作を許可するかどうか（ウィンドウの占有権の設定）
- 背景が画像である場合の、背景のファイル名

設定しない情報

- ウィジェットの大きさや、ウィンドウ内での位置等のレイアウトに関係する情報
- ウィジェットの色や文字のフォント等の属性値

PENGUIN で作成することができる 12 種類のウィジェット及び、各ウィジェットに対して設定する情報は以下の通りである。

button

ユーザイベントを受け取ることで何らかの処理を行うために最もよく使われるウィジェットである。PENGUIN において設定する情報は以下の通りである。

- プログラム内部で使用されるボタンウィジェットの名前（PENGUIN が自動的に決定）
- ボタンウィジェットが表示する外見上の名前
- ボタンウィジェットを押した時に値が設定される変数の名前

radiobutton

複数の項目の中から 1 つの項目を排他的に選択するためのウィジェットである。PENGUIN において設定する情報は以下の通りである。

- 1 セットのラジオボタンウィジェットの数
- この 1 セットのラジオボタンウィジェットが選択された時に、選択された項目の値が設定される変数の名前

- プログラム内部で使用されるラジオボタンウィジェットの個々の名前（PENGUIN が自動的に決定）
- ラジオボタンウィジェットが表示する個々の外見上の名前
- それぞれのラジオボタンウィジェットが選択された時に設定される変数の値

checkbutton

複数の項目の中から幾つかの項目を選択するためのウィジェットである。PENGUIN において設定する情報は以下の通りである。

- 1 セットのチェックボタンウィジェットの数
- プログラム内部で使用されるチェックボタンウィジェットの個々の名前（PENGUIN が自動的に決定）
- チェックボタンウィジェットが表示する個々の外見上の名前
- 個々のチェックボタンが選択されている状態の時に変数に設定される値
- 個々のチェックボタンが選択されていない状態の時に変数に設定される値
- それぞれのチェックボタンウィジェットが選択されているまたは選択されていない状態の値が設定される変数

frame

主に、他のウィジェットをウィンドウに配置する際の敷き物として使用されるウィジェットである。PENGUIN において設定する情報は以下の通りである。

- プログラム内部で使用されるフレームウィジェットの名前（PENGUIN が自動的に決定）

label

1 行の文字列を表示するためのウィジェットである。PENGUIN において設定する情報は以下の通りである。

- プログラム内部で使用されるラベルウィジェットの名前（PENGUIN が自動的に決定）
- ラベルウィジェットが表示する外見上の名前

entry

1 行の文字列入力を行うためのウィジェットである。PENGUIN において設定する情報は以下の通りである。

- プログラム内部で使用されるエントリウィジェットの名前（PENGUIN が自動的に決定）
- エントリウィジェットに入力された文字列が設定される変数

scale

一定範囲の数を設定するために使用されるウィジェットである。PENGUIN において設定する情報は以下の通りである。

- プログラム内部で使用されるスケールウィジェットの名前（PENGUIN が自動的に決定）
- スケールウィジェットが表示する外見上の名前
- スケールウィジェットの始まりの数値
- スケールウィジェットの終わりの数値
- スケールウィジェットの長さ
- スケールウィジェットに設定されている値を表示するかしないかの選択
- スケールウィジェットに表示された数字の刻みの印の間隔
- スケールウィジェットを水平方向に置くか、垂直方向に置くかを選択

menubutton

複数の事象を選択的に起動させる場合に使用される。PENGUIN において設定する情報は以下の通りである。

- プログラム内部で使用されるメニューボタンウィジェットの名前 P（PENGUIN が自動的に決定）
- メニューボタンウィジェットが表示する外見上の名前
- プルダウンメニューの情報
  - プルダウンメニューの種類

    command ユーザイベントによって何らかの処理を行う

    radiobutton 複数のラジオボタンメニューから 1 つを排他的に選択する

    checkbutton 複数のチェックボタンメニューから幾つかの項目を選択する

    cascade メニューを階層化する。階層化された先のメニューに関する情報は、「プルダウンメニューの情報」と同様の情報を設定する。
  - プログラム内部で使用されるプルダウンメニューの名前 P（PENGUIN が自動的に決定）
  - プルダウンメニューが表示する外見上の名前

message

複数行にわたる文字列を表示するウィジェットである。PENGUIN において設定する情報は以下の通りである。

- プログラム内部で使用されるメッセージウィジェットの名前（PENGUIN が自動的に決定）
- メッセージウィジェットが表示する文字列

listbox

1 行に 1 つずつの項目の一覧を表示させるウィジェットである。多くの項目の中から 1 つの項目を選択する場合や何らかの項目の一覧を表示させる場合に使用される。PENGUIN において設定する情報は以下の通りである。

- プログラム内部で使用されるリストボックスウィジェットの名前（PENGUIN が自動的に決定）
- リストボックスウィジェットにスクロールバー（縦方向、横方向）をつけるかつけないかを選択
- リストボックスウィジェットが表示する項目

canvas

図を表示または描画するために使用されるウィジェットである。PENGUIN において設定する情報は以下の通りである。

- プログラム内部で使用されるキャンバスウィジェットの名前（PENGUIN が自動的に決定）
- キャンバスウィジェットにスクロールバー（縦方向、横方向）をつけるかつけないかを選択

text

複数行にわたる文字列をを表示または入力するためのウィジェットである。PENGUIN において設定する情報は以下の通りである。

- プログラム内部で使用されるテキストウィジェットの名前（PENGUIN が自動的に決定）
- テキストウィジェットにスクロールバー（縦方向、横方向）をつけるかつけないかを選択

3.5.2 トランジション GUI 情報設定

図 3.14 に、PENGUIN のトランジションの GUI 情報入力ウィンドウを示す。

図 3.14: GUI 情報設定ウィンドウ（トランジション）

トランジションには GUI 情報として、ユーザイベントのバインド情報を持たせる。ダイアローグ記述担当者は、そのトランジションで起こるユーザイベントの種類（右ボタンクリック、フォーカス入力など）と、そのイベントをバインドするウィジェットを指定する。GUI 情報を持ったトランジションは、そのトランジションで指定されたイベントが起こることによって発火する。

ペトリネットでは、トークンによりその動作が制御される。従って、アプリケーション起動時に複数のプレースにトークンを持たせたり、トランジション発火後に複数のプレースにトークンを移動させることにより、複数のウィンドウを同時に起動させることが可能である。また、本手法では、GUI 情報を持ったプレースがトークンを保持している場合に、ユーザはそのプレースのウィンドウに対して操作が可能となる。従って、複数のプレースが同時にトークンを保持することにより、ユーザは複数のウィンドウを並行して操作することができる。

ここで設定する情報は以下の通りである。

ウィンドウ消去　このトランジションが発火した際に表示されているウィンドウのうち、このトランジションが発火したことによって消去されるウィンドウを設定する。図 3.15 は、消去するウィンドウを設定するためのウィンドウである。消去するウィンドウは、そのウィンドウの情報を保持しているプレースの名前で指定する。

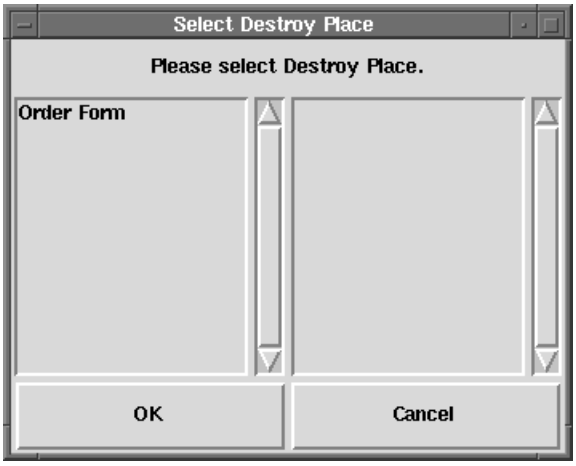


図 3.15: 消去ウィンドウの選択

ユーザイベント設定　このトランジションを発火させるユーザイベントを指定する。イベント名を指定するウィンドウを図 3.16 に示す。

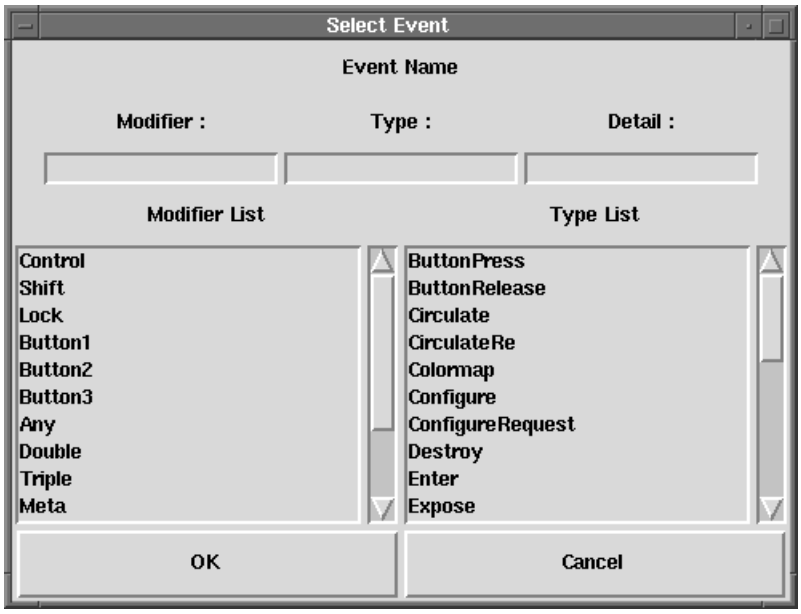


図 3.16: イベント名入力ウィンドウ

イベントは以下の Modifier、Type、Detail を組み合わせて指定する。構文は

$$< Modifier - Modifier - ... - Modifier - Type - Detail >$$

のようものである。ただし、Modifier は 0 個以上で指定する。また、少なくとも Type または Detail のどちらかが指定されていれば、他のものは省略することができる [37][38]。

Modifier Shift キーや Control キーが押された状態でイベントが発生した、あるいはマウスがダブルクリックされたなど、そのイベントが発生した状態や発生のしかたなどを示すもの

Type イベントのタイプ（ボタンを押す”ButtonPress”、キーボードのキーを押す”KeyPress”など）

Detail 特定のボタン（マウスの第 1 ボタン”1”）やキーボードの文字を示している特別な情報

バインドウィジェット選択 ユーザイベントをバインドするウィジェットを選択する。

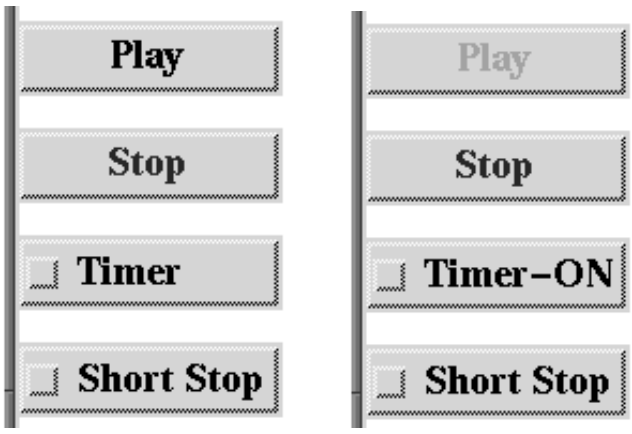
3.6　PENGUIN の補助機能

PENGUIN には、以上のような基本操作だけでなく、ペトリネットの記述を容易にしたり、記述の労力を軽減するためのいくつかの機能が用意されている。この機能を以下に示す。

3.6.1　ペトリネット記述時の機能

(1) ウィジェットの状態変化指定

この機能は、GUI アプリケーションの実行中にウィジェットの色や名前等の属性値等を変化させる場合に、どのように変化させるかを指定することができる機能である。例えばビデオデッキの制御アプリケーションでは、録画予約のタイマーが “OFF” の状態の時にタイマーボタンを押すと、再生ボタンが使用可という状態から使用不可という状態になり、タイマーボタンの表示が “Timer” という状態から “TimerSet” という状態に変化することが考えられる（図 3.17）。



ウィンドウ 1　　ウィンドウ 2

図 3.17: 状態変化の例

この例では、状態変化前のウィンドウ 1 においてチェックボタン”Timer” を押すと、ウィンドウ 1 のボタン”Play” がウィンドウ 2 のように使用不可になり、ウィンドウ 1 のチェックボタンも”Timer” に名前が変わる。

このようにウィジェットの状態を変化させることは、アプリケーション利用者が誤操作を行うことを防いだり、利用者にとって使いやすいアプリケーションを作成するために非

常に有効であり、多くの GUI アプリケーションで採用されている。

本手法では、ウィンドウもウィジェットの一つであると考えており、この機能を用いることにより、特定のウィンドウに対して占有権を付与したり、その占有権を解除することも可能である。

このウィジェットの状態変化を PENGUIN で表現するには、状態変化が起こった直後のプレースにおいて、状態変化が起こる前のプレースを指定し、続いて状態変化の種類を指定することが必要である。

図 3.17 の状態変化をペトリネットで記述したものが図 3.18 である。プレース”TapeIn” を状態変化前、プレース”TimerSet” を状態変化後とする。この例ではまず、プレース”TimerSet” において、”TimerSet” は ”TapeIn” のウィンドウの状態が変化したものであるということを指定する。そして”TimerSet” において実際の変化のしかたを指定する。この機能では、個々のウィジェットの状態変化だけでなく、新しいウィジェットの配置や表示されているウィジェットの削除といった指定も行うことができる。

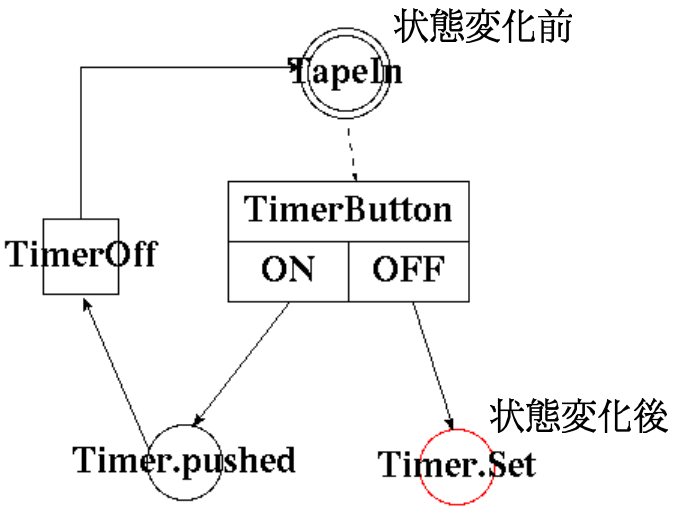


図 3.18: 状態変化指定の記述例

(2) プレース・トランジションの自動生成指定

この機能は、プレースの出力トランジション、及びトランジションの出力プレースを半自動生成する機能である。ペトリネットでは、1 つのプレースが複数の出力トランジション、あるいは 1 つのトランジションが複数の出力プレースを持っている場合がある。その複数の出力を 1 つ 1 つ記述していくということは、ダイアローグ記述担当者にとって

大きな負担である。

このような場合、プレースの出力トランジションを、その種類と個数、そしてそれらをつなぐアークの種類を指定することによって、出力トランジションを生成することができる。生成されたトランジションは、名前等の詳細情報を持っていないため、生成後にダイアローグ記述担当者が詳細情報を指定する。同様にして、トランジションの出力プレースを生成することもできる。

また、GUI アプリケーションでは、ボタンが押されることによって処理が行われたりウィンドウが表示されるなど、何らかの変化が起こることが多い。このため、あるウィンドウ上にボタンウィジェットが配置された場合には、そのボタンウィジェットの数と同数のボタントランジションが、そのウィンドウについての情報を持つプレースの出力トランジションとして、ダイアローグ記述担当者が明示的に指定することなく自動生成される。

図 3.19 は、第 3.9 章のビデオデッキの制御システムのペトリネットの一部である。このペトリネットでは、ビデオデッキの操作ウィンドウ内に配置されている様々なボタンのうちの一部の処理について記述しようとしているものである。操作ウィンドウについての情報は、プレース”TapeIn” が保持している。操作ウィンドウ上に配置されているボタンが押されると、各ボタンに関連付けられた関数が呼び出され、処理が行われる。従って、プレース”TapeIn” は複数の出力トランジションを持っていることとなる。この例では、その複数の出力トランジションのうちの一部が自動生成されている。

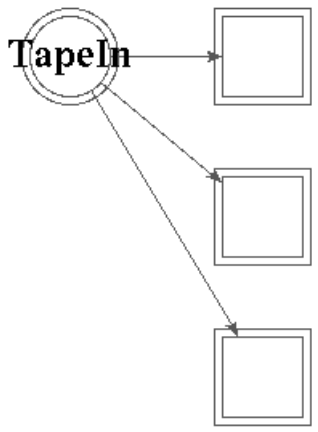


図 3.19: トランジションの自動生成例

(3) プレースの類似指定

1 つのアプリケーション内の異なる複数のウィンドウに配置されているウィジェットの種類やその個数等の構成が類似している場合がある。この機能はこのような場合に便利なであり、類似した複数のウィンドウのうちの 1 つを作成しておくと、その他は作成済みのウィンドウを保有するプレース名を指定するだけで作成できる。そして、作成したウィンドウを必要に応じて変更し、求める構成のウィンドウを作成することができる。

(4) ウィジェット変更機能

PENGUIN では、各ウィンドウにどのウィジェットを配置するかはダイアローグ記述担当者が決定する。しかし、GUI アプリケーションのユーザビリティを考慮して最終的な外観を決定することは GUI 定義担当者の役割であるべきである。例えば項目選択を行う場合に、ラジオボタンを使うかリストボックスを使うかなどというウィジェットの選択も GUI 定義担当者が行うべきものであると考えている。ウィジェット変更機能とは、ペトリネットが記述された後に、GUI 定義担当者が、ダイアローグ記述担当者が指定したウィジェットの種類を変更するための機能である。この機能では、例えばラジオボタンからはリストボックスあるいはプルダウンメニュー等というように、個々のウィジェットから変更することのできるウィジェットを限定することにより、ダイアローグ記述担当者が記述したペトリネットの構造を壊さないウィジェットを変更することができる。

(5) チェック機能

ペトリネットの記述時には、例えばプレースとプレースをアークで結ぶなど、ダイアローグ記述担当者が誤ったペトリネットを記述する場合がある。PENGUIN には、そのような誤った記述をしないようにチェックを行う機能が用意されている。この機能は、誤った記述が行われると、その場でエラーメッセージを表示し、その記述を無効とする。この機能により、ダイアローグ記述担当者は容易に正しいペトリネットを記述することができる。

3.7　GUI アプリケーション生成

ダイアローグ記述担当者によってペトリネットが記述され、そのペトリネットを PENGUIN の GUI 生成系（図 3.1）に入力すると、PENGUIN は

- 各関数の骨格（処理内容の書き込まれていない関数とそのつながり）
- GUI プロトタイプ（ウィジェットの生成、ウィジェットに対するイベントの生起などの GUI に関するプログラム）
- ステートマネージャ（アプリケーション実行中にトークンを監視することにより、ペトリネットに忠実に従ってアプリケーションを動作させるための機構）

の 3 種類のプログラムを生成する。この生成について詳細を述べる。

3.7.1　入力

ダイアローグ記述担当者がペトリネットの図を記述し、それを保存すると、PENGUIN は、そのペトリネットの接続情報などをテキストで表現した論理ペトリネットファイルを生成する。GUI 生成系は、この論理ペトリネットファイルを入力として GUI アプリケーションを生成する。論理ペトリネットの形式を以下に示す。

プレース

P　[*ID*]　⟨*label*⟩　[*parent*]　[*type*]
[*input transition*]:[*arc type*]:[*the number of input tokens*] のリスト
[*output transition*]:[*arc type*]:[*the number of output tokens*] のリスト
[*the number of tokens*]

トランジション

T　[*ID*]　⟨*label*⟩　[*parent*]　[*type*]
[*input ilace*]:[*arc type*]:[*the number of input tokens*] のリスト
[*output place*]:[*arc type*]:[*the number of output tokens*] のリスト

P / T: 項目がプレース（P）であるかトランジション（T）であるかを識別

ID: 便宜的にプレース及びトランジションに割り当てられる一意な ID

⟨*label*⟩**:** プレース / トランジションの名前

*parent***:** プレース / トランジションが階層構造になっている場合の親（階層構造になっていない場合は”toplevel” と表記）

*type***:** プレース / トランジションの種類

- プレース

  n: 標準のプレース

  d: トークン消滅プレース

- トランジション

  n: 標準のトランジション

  b: ボタントランジション

  s: 条件分岐トランジション（”s-分岐数” という形で分岐の数とともに表現）

input transition / *input place***:** 入力トランジション / 入力プレースの ID（入力トランジションが条件分岐トランジションの場合は、”⟨ 分岐名 ⟩” という形式で、どの分岐条件からアークが入力するかを表現）

output transition / *output place***:** 出力トランジション / 出力プレースの ID（このトランジションが条件分岐トランジションの場合は、”⟨ 分岐名 ⟩” という形式で、どの分岐条件からアークが出力するかを表現）

*arc type***:** 入力 / 出力アークの種類

n: 標準のアーク

h: 抑止アーク

m: トークン非移動アーク

the number of input tokens / *the number of output tokens***:** 入力 / 出力トークンの数

*the number of tokens***:** アプリケーション起動時にプレースが保持しているトークンの数（プレースのみ）

3.7.2　生成アルゴリズム

論理ペトリネットファイルから、PENGUIN 中のアプリケーション生成機能がアプリケーションの骨組みと GUI プロトタイプを生成していく手順を以下に示す。

1. 初期マーキングのあるプレース（アプリケーション起動時にトークンを持っているプレース）を探し、そのプレースの GUI に関するプログラムを生成する。
2. 1. のプレースを起点とし、アークをたどってトランジションを探索していき、3. 〜 6. を繰り返す。
3. 発見したトランジションの出力プレースが GUI 情報を持つ場合は 4.、GUI 情報を持たない場合は 5. の処理を行う。
4. 3. のトランジションで呼び出される関数に、GUI 部分の処理を行う専用の関数（GUI 専用関数と呼ぶ）を挿入する。その後、6. の処理を行う。
5. 3. のトランジションからさらにアークをたどり、発見したトランジションで呼び出される関数を 3. で呼び出される関数の中に挿入する。その後 6. の処理を行う。
6. 3. のトランジションからアークをたどってトランジションを探索し、発見されていないトランジションが存在した場合には 3. の処理を、全てのトランジションを発見し終った場合には 7. の処理を行う。
7. 最後に GUI 専用関数を作成し、その中にウィジェットの生成やウィジェットに対するイベントの生起等の GUI に関する全てのコードを記述し、GUI プロトタイプを生成する。

生成されたプログラムは、 PENGUIN 同様、プログラミング言語 Tcl/Tk で書かれている。

3.7.3　出力

GUI 生成系が生成するものは、ウィジェットの生成やウィジェットに対するバインドなどの GUI に関するプログラムが記述された GUI プロトタイプと、処理内容の書き込まれていない関数と、関数の骨組みが記述されたファイルを生成する。しかし、ペトリネットではトランジションが発火するための規則を持っていたり、本手法で使用するペトリネッ

トには、本手法独自に付加した拡張記法が存在するため、生成されたプログラムは、単独では、入力されたペトリネットに沿って動作しない。そこで GUI 生成機能は、自動生成されたプログラムを、ペトリネットに沿って動作させるためのプログラムを、関数の骨組みファイルと同時に生成する。このプログラムをステートマネージャと呼ぶ。このステートマネージャは以下のことを行う。

- アプリケーション実行中に、トークンが存在するプレースと、個々のプレースが保持するトークンの数を取得する。
- 発火条件（入力プレースに、必要な数のトークンが揃った場合）を満たしたトランジションの関数を実行させる。
  - ユーザイベントが起こることによって発火するトランジションについては、発火条件が揃わなければ、イベントが発生しても、発火させない。
- 関数の実行後（トランジションの発火後）に、出力アークに設定された数のトークンを出力プレースに配置する。

3.8 プロトタイプの変更

3.8.1 PENGUIN の前提

3.2.1 で述べたように、アプリケーションの外観については PENGUIN が自動的に決定している。ユーザビリティを考慮した GUI を構築するために、自動生成後に、GUI 定義担当者が既存のツールを用いて外観変更を行う。この外観変更の段階で設定するものは

- ウィンドウ内でのウィジェットの位置や大きさ
- ウィジェットの色や文字のフォント

等の外観に関する事柄である。また、3.6.1(4) で述べたウィジェット変更機能を用いることにより、アプリケーションの制御構造に関わらないようにウィジェットを変更することも可能である。ただし、

- 新しいウィジェットの追加
- 既存のウィジェットの削除

といった、アプリケーションの制御にかかわるような設定は想定していない。本手法では、ダイアローグを用いてアプリケーションの制御構造を記述するため、この時点ではウィンドウ中に必要なウィジェットは過不足なくすべて含まれているということができる。従って、この外観変更の段階では、GUI 定義担当者はそれらの表現形式、配置、大きさ等の外観に気を使えばよい。

3.8.2　インタフェースビルダ

PENGUIN では自動生成されたプログラムは既存のツールを使用してその外観を変更することを想定している。既存のツールとしては、インタフェースビルダというものが考えられる。インタフェースビルダとは、GUI の表示部分の開発を支援するツールである。しかしこれは、作成したプログラムのウィジェットの配置を変えたりすることもできるため、PENGUIN をサポートするシステムとして利用することができる。PENGUIN を用いてアプリケーションを作成した場合に、外観変更に使用することができるインタフェースビルダの例は以下の通りである。

XF　プログラミング言語 Tcl/Tk で構築されたインタフェースビルダで、公開されているアプリケーションである [32][40]。これは多くのインタフェースビルダとは異なり、Tcl/Tk スクリプトを記述することに近い形で各ウィジェットを定義していく。つまり、

1. ウィジェットの一覧から配置したいウィジェットを選択し、「追加」ボタンを押すことで、ウィンドウ上にウィジェットが追加される。
2. 追加されたウィジェットに対して、frame ウィジェットへなどの入れ子構造を設定し、配置を設定する。
3. 作成されているウィジェットに対して色やフォントなどのオプションを設定する。

という形で定義する。また、そのウィジェットの配置も、ウィジェットをビジュアルに移動するのではなく、ジオメトリマネージャ（ウィジェットの位置と大きさを決定するもの）である packer または placer を利用し、ウィジェットの配置設定用のフォームから設定する。従ってある程度 Tcl/Tk の言語体系の知識がなければ利用することは難しい。packer 及び placer とは、以下のような特徴がある。

packer　Tcl/Tk において最も頻繁に利用されるジオメトリマネージャで、マスタ（frame ウィジェットやウィンドウのような、ウィジェットの敷物の役割を果た

すウィジェット）のある一つの辺に接するようにスレーブ（マスタの上に配置されるウィジェット）を配置していく。ウィンドウ全体の大きさが変わってもウィジェット間のレイアウトを保持したい場合に有用である。

placer ウィジェットを絶対座標あるいは相対座標、またはその組合せで指定して配置するジオメトリマネージャである。絶対座標と相対座標を組合せて配置を指定できるため、例えば、スレーブは絶対座標でその幅と高さを固定し、相対座標でマスタの中央に位置するということが可能である。ウィジェットの位置や大きさをより細かく設定したい場合に有用である。

Tclbomber Tcl/Tk 専用のインタフェースビルダであり、XF と異なり商品である。一般的なインタフェースビルダの様な機能は一通り揃っている [40]。つまり、

- マウスによるビジュアルなウィジェットの配置機能
- カット＆ペースト機能
- グリッド機能
- あるウィジェットにならって他のウィジェットを整列させる機能

を有している。また、Tclbomber の固有の機能として、

- 二次元グラフを作成する機能
- 表を作成する機能

が用意されている。しかし、確かに Tclbomber を利用することで容易にスクリプトを記述することができるが、細かな設定については必ずしも設定しきれるというわけではない。特に、Tclbomber 生成されたスクリプトがやや独自のものとなっているため、二次加工をすることも難しい。

3.9 アプリケーション生成例

図 3.20 に PENGUIN を使って実際に入力したペトリネットの入力例を示す。この例は、ビデオデッキの制御システムで、ビデオの再生と録画予約の際の処理の流れの一部を表している。

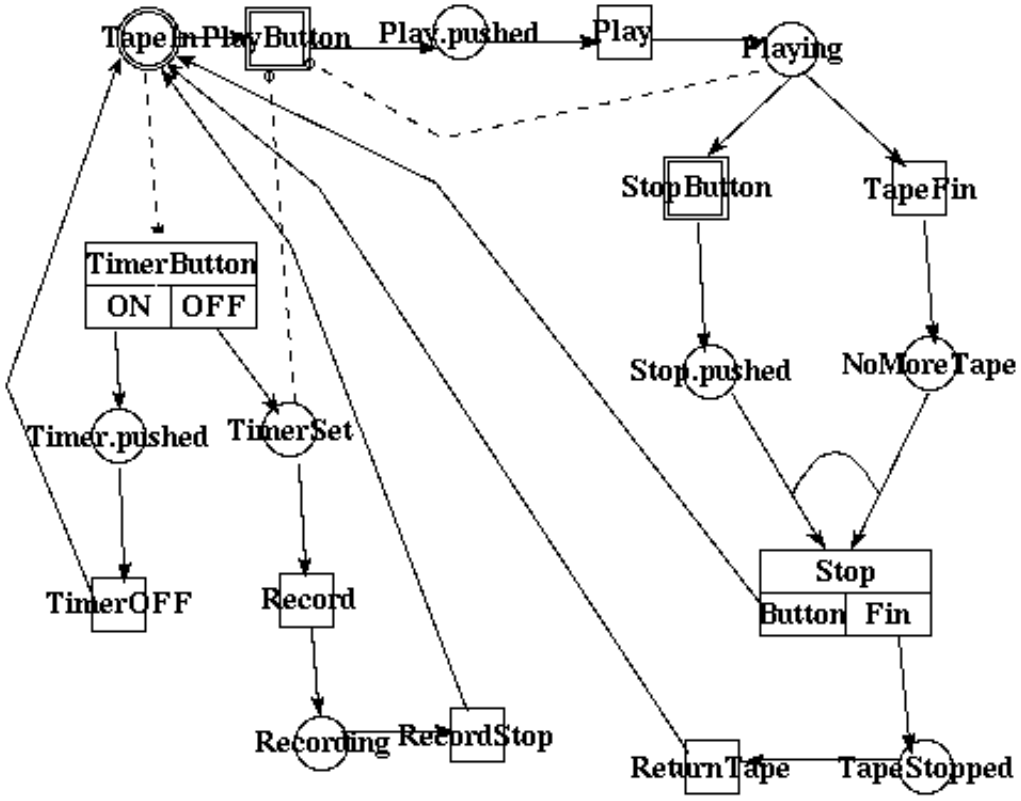


図 3.20: ペトリネット記述例

このビデオデッキの制御システムは、ビデオテープが入った状態で「再生ボタン」を押すとビデオが再生され、再生中に「ストップボタン」が押された場合またはビデオテープが終った場合にはビデオの再生が止まる。ビデオテープが終ってしまって再生が終了した場合にはテープが巻き戻される。ビデオの再生が止まる条件は、「ストップボタン」が押された、又は、ビデオテープが終ってしまったという 2 つの条件のうちのどちらかであるため、トランジション ”Stop” に入力している 2 本のアークは OR アークで結ばれている。また、テープが終ったことにより再生が止まった場合には巻き戻しをするため、トランジション ”Stop” は条件分岐トランジションが用いられている。

また一方で、タイマーボタンが ON であった場合にタイマーボタンを押すと、タイマーが OFF となり、次のイベント待ちの状態になる。タイマーボタンが OFF であった場合は、タイマーがセットされ、ある時間に録画が始まる。タイマーボタンは、ON 状態と OFF

状態を同じボタンで切替えるため、プレース ”TapeIn” とトランジション ”TimerButton” を結ぶアークにはトークン非移動アークが使用され、トランジション ”TimerButton” は条件分岐トランジションが用いられている。また、タイマーが ON の状態の時にはビデオは再生されず、また、ビデオの再生中に「再生ボタン」を押しても何も処理されないため、「再生ボタン」を表すトランジション ”PlayButton” と、プレース ”Timer Set” 及びプレース ”Playing” を抑止アークで結び、これらのプレースにトークンがある場合は、トランジション ”PlayButton” が発火しないようにされている。

3.9.1 論理ペトリネットファイル

図3.20に示されているペトリネットの論理ペトリネットファイルの一部を図3.21に示す。

```
P 0001 <TapeIn> toplevel d
<Button>0005:n:1 0009:n:1 0010:n:1 0006:n:1 0015:n:1 0019:n:1 0022:n:1
0001:n:1 0007:m:1 0011:n:1 0012:n:1 0015:m:1 0016:m:1 0019:m:1 0020:m:1
1
P 0002 <Play.pushed> toplevel n
0001:n:1
0002:n:1
f
P 0003 <Playing> toplevel n
0002:n:1 0016:n:1 0020:n:1
0003:n:1 0004:n:1 0001:h:1
0
T 0001 <PlayButton> toplevel b
0001:n:1 0007:h:1 0003:h:1
0002:n:1
T 0002 <Play> toplevel n
0002:n:1
0003:n:1
T 0003 <StopButton> toplevel b
0003:n:1
0004:n:1
}
```

図 3.21: 論理ペトリネットファイル例

3.9.2　GUI プロトタイプ生成例

図 3.20 のペトリネットが記述された後、3.7.2 のアルゴリズムに沿って、GUI プロトタイプとアプリケーションの骨組みを実際に生成していく手順を以下に示す。

1. 初期マーキングのあるプレースを探し、そのプレースの GUI に関するプログラムを生成する。初期マーキングのあるプレースは ”TapeIn” のみであるため、プレース ”TapeIn” を保持しておく。同時に、プレース ”TapeIn” が保持する GUI 情報をもとにして、このプレースの GUI に関するプログラムを生成する。

2. 初期マーキングのあるプレース ”TapeIn” を起点として ”PlayButton”、”TimerButton”、”Play”... のようにアークをたどってトランジションを探索していく。

3. 発見したトランジションの出力プレースが GUI 情報を持つ場合は 4.、GUI 情報を持たない場合は 5. の処理を行う。

4. 3. のトランジションで呼び出される関数に、GUI 専用関数（”GUI_Display_*ID*” という名前となる）を挿入する。その後、6. の処理を行う。

5. 3. のトランジションからさらにアークをたどり、発見したトランジションで呼び出される関数を 3. で呼び出される関数の中に挿入する。つまり、一方では

   PlayButton → **Play** ···

   またもう一方では

   TimerButton → **TimerOFF** ···
   ↘
   Record ···

   のように関数の流れを組み立てていく。その後 6. の処理を行う。

6. 3. のトランジションからアークをたどってトランジションを探索し、発見されていないトランジションが存在した場合、つまり ”Play”、”StopButton”、”TapeFin”、”Record” では 3. の処理を、全てのトランジションを発見し終った場合、つまり ”ReturnTape”、”RecordStop”、”TimerOff” では 7. の処理を行う。

7. 最後に GUI 専用関数を作成し、各プレースの GUI 情報から GUI 部分のプログラムを記述する。

このようにして生成したプログラムの一部を図 3.22 を示す。

```
# This File is TCL/TK Program File by PENGUIN
# This Application is <VideoDeck>

. configure -height 550 -width 550
canvas .back -bg white -height 550 -width 550
place .back -x 0 -y 0
button .button1 \
     -text ''Power'' \
     -height 2 \
     -command ''StateManage_CheckToken 1 1 .button1''
     -command ''StateManage_CheckToken 1 1 .button1''
place .button1 -x 10 -y 40 -height 30
button .button2 \
     -text ''Play'' \
     -height 2 \
     -command ''StateManage_CheckToken 1 1 .button2''
# This Process is Transition <PlayButton>.
proc PlayButton { ST1 ST2 ST3} {
    # <-- Insert Code
  StateManage_StoreToken 1 ''''
 }
# This Process is Transition <Play>.
pproc Play { ST1 ST2 ST3} {
    # <-- Insert Code
  StateManage_StoreToken 2 ''''
 }
# This Process is Transition <StopButton>.
proc StopButton { ST1 ST2 ST3} {
    # <-- Insert Code
  StateManage_StoreToken 3 ''''
 }
```

図 3.22: 自動生成されたプログラム例

3.9.3　外観変更例

3.2.1 でも述べた通り、PENGUIN を使用して作成した GUI アプリケーションの外観については PENGUIN が自動的に決定しており、ユーザの好みや操作性等は反映されていない。従って、自動生成された GUI アプリケーションの外観は、GUI 定義担当者が既存のツールを用いて変更を行う。実際に PENGUIN を使用して作成した GUI アプリケーション中のウィンドウの 1 つを図 3.23 に、それをインタフェースビルダ XF を使用して変更したウィンドウの例を図 3.24 に示す。

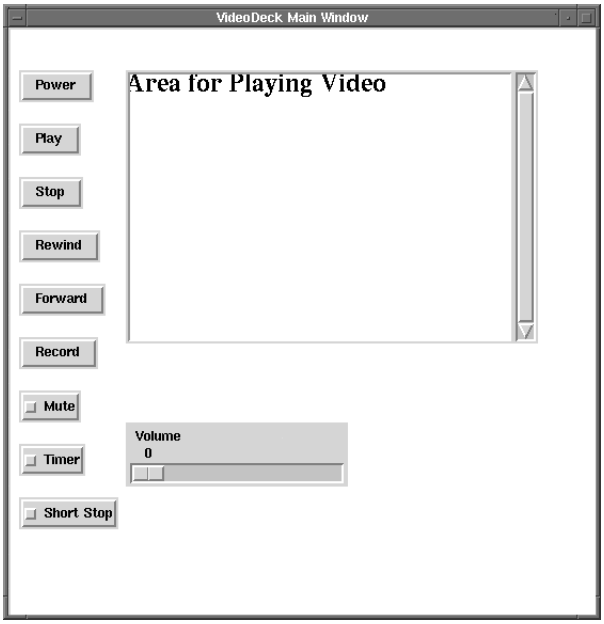


図 3.23: 生成されたウィンドウ例

図 3.24: 変更したウィンドウ例

個々のウィンドウをどのように変更するかは、アプリケーションを使用するエンドユーザの好みや操作性が関係しているため、一意に定めることはできない。重要なことは、この時点ではウィンドウ中に必要なウィジェットはすべて含まれており、GUI 定義担当者はそれらの配置や大きさのみに気を使えばよい、ということである。

3.10 評価

PENGUIN を使用して、図 3.20 を含むビデオデッキの GUI アプリケーション、貸ビデオ店の GUI アプリケーション、及び本システム”PENGUIN” という 3 つのアプリケーションを作成した。以下ではこの例を分析し、本手法の有効性を評価する。各アプリケーションの規模を表 3.1 に示す。この表における各項目が表すものは以下の通りである。

プログラム（行）: 生成されたプログラムの行数

ウィジェット（個）: 生成されたウィジェットの総数

ペトリネット（枚）: 分割した物理ペトリネットの枚数

全プレース: ペトリネット中のプレースの総数

全トランジション: ペトリネット中のトランジションの総数

全アーク: ペトリネット中のアークの総数

GUI プレース: GUI 情報を持つプレースの数

GUI トランジション GUI 情報を持つトランジションの数

平均ウィジェット 1 ウィンドウあたりのウィジェット数の平均

この表より、3 つの例のウィジェット数を平均すると、1 ウィンドウあたり 8 〜 9 個のウィジェットを持っていることがわかる。

表 3.1: ペトリネットと生成されたプログラムの内訳

	ビデオデッキ	貸ビデオ	PENGUIN
プログラム（行）	476	712	2801
ウィジェット（個）	26	99	366
ペトリネット（枚）	6	11	32
全プレース	23	48	174
全トランジション	41	58	197
全アーク	76	133	483
GUI プレース	2	16	34
GUI トランジション	20	34	97
平均ウィジェット	13.50	5.50	10.76

3.10.1　開発したアプリケーションの性質

貸ビデオ店の管理アプリケーションを、PENGUIN とプログラミング言語 Tcl/Tk を用いる、Tcl/Tk によって全てのプログラム・コードを記述する、VisualBasic を用いるという 3 つの手法で作成した。大部分の関数がユーザのイベントによって起動されるイベント駆動型のアプリケーションであり、かつ広く利用されている典型的な GUI アプリケーションであると考えて、貸ビデオ店のアプリケーションを取り上げた。作成しはじめてか

表 3.2: 各手法によるアプリケーション開発

手法	PENGUIN + Tcl/Tk	Tcl/Tk	VisualBasic
時間（分）	460	520	710
行	3410	1018	2782
ウィジェット	168	170	133
関数	145	21	47

ら完成するまでの時間と、生成したプログラムの総行数、アプリケーション内で宣言されている関数の総数を表 3.2 に示す。

PENGUIN で構築したアプリケーションの外観は XF を用いて変更した。このうち、PENGUIN+Tcl/Tk と Tcl/Tk の二つの手法によるアプリケーション構築では、プログラムの行数と関数の数に大きな違いが現れた。この理由は次の 2 点である。

- PENGUIN でプログラムを自動生成すると、GUI を表示するための専用の関数が生成される。この GUI 表示専用の関数の数は、初期マーキングのないプレースのうちの GUI 情報を持ったプレースと同数である。

- PENGUIN で自動生成したプログラムは、その GUI 部分が変更される。XF を用いて変更を行うと、1 ウィンドウの変更のために、ウィジェットの属性などの様々な情報が書き込まれた 2 つの関数が生成される。

しかし、増加している関数は全て GUI の表示に関するものであり、アプリケーションの制御構造とは関係ないものである。従ってアプリケーションの制御構造が複雑化することはなく、増加した関数に関しては、プログラムの記述や保守の際には考慮する必要はない。アプリケーションの作成には、作成した人の各手法に対する熟練度等が評価結果に影響する。従ってこの結果は、本手法がアプリケーション開発にかかる時間の短縮に貢献するという証明にはならないが、他の手法と比べても実用的には問題ないものであることがわかる。

アプリケーションの実行時間に関しては、増加している関数やプログラムが GUI 表示に関するもののみであるため、GUI 表示のみに着目して実行時間の測定を行った。その結果が表 3.3 である。この測定は、PENGUIN と Tcl/Tk を用いる手法と、Tcl/Tk によって全てのプログラム・コードを記述するという手法で作成した 2 つの貸ビデオ店の GUI アプリケーションを、Sun SPARC Station 上でそれぞれ 5 回実行し、各ウィンドウが表

表 3.3: ウィンドウ表示時間の比較

	PENGUIN + Tcl/Tk	Tcl/Tk
ウィンドウ 1	12165（9）	9863（9）
ウィンドウ 2	36980（24）	33824（26）
ウィンドウ 3	11726（17）	16717（16）
ウィンドウ 4	8824（18）	7906（16）
ウィンドウ 5	6281（11）	7304（15）

示されるためにかかった時間を測定し、平均したものである。括弧内はそのウィンドウに配置されているウィジェットの数であり、測定時間の単位はマイクロ秒である。ただし、表 3.3 における各ウィンドウの比較は、その同様の構成をしたウィンドウを用いて行っている。

この結果を平均すると、PENGUIN を用いて構築したアプリケーションでは、1 個のウィジェットを表示するために約 995 マイクロ秒、全てのプログラム・コードを記述するという手法で構築したものでは、922 マイクロ秒かかるということになる。以上の結果より、PENGUIN を用いて生成したアプリケーションを XF を用いて外観変更を行った場合に、プログラム・コードを全て人手で記述したものよりもわずかに実行速度が遅くなることがわかった。しかしこれはアプリケーション利用者が遅いと感じるほどのものではなく、実用には問題ないものである。また、このように、Tcl/Tk という処理速度の遅いプロトタイプ言語を用いて実装を行った場合でも、実用的に問題ない実行時間が得られるため、他の言語を用いて実装を行った場合は、表 3.3 で示した時間以下でウィンドウが表示できると考えられる。従って、この結果より、本手法を用いて開発した GUI アプリケーションは、GUI の表示時間については、実用には問題ないことがわかる。

3.10.2　アプリケーション開発の手間

また、これらの例の中からいくつかのウィンドウを選んで XF を使用してウィンドウの変更を行ったところ、表 3.4 の結果が得られた。

表 3.1 より、ウィジェット数を平均すると、 1 ウィンドウあたり 9 〜 10 個ほどのウィジェットを持っていることとなる。これらの例の中からいくつかのウィンドウを選んで XF を使用してウィンドウの外観変更を行ったところ、表 3.4 の結果が得られた。以上より 3

表 3.4: ウィンドウの変更における手間

	時間（分）	ウィジェット（個）
例 1	6	10
例 2	21	28
例 3	20	25
例 4	13	20
例 5	12	17

つの例の平均を求めると、9 個ほどのウィジェットを持っている 1 ウィンドウを変更するためには、ウィジェットをどのように配置するか、あるいは外観変更に使用するツールに対する GUI 定義担当者の熟練度などにもよるが、約 6 〜 7 分で十分であるということが得られた。

3.10.3 補助機能の有効性

3 つのアプリケーションのペトリネットを記述する際に使用した PENGUIN の補助機能のうち、表 3.5 にプレース・トランジションの自動生成指定、表 3.6 にプレースの類似指定についてその使用頻度を示す。

表 3.5: 自動生成されたプレース・トランジションの個数

	通信販売	貸ビデオ	PENGUIN
プレース	0	2	47
トランジション	5	48	115
ボタントランジション	5	46	84

表 3.5 では、自動生成を行ったプレース・トランジションの数と、自動生成されたトランジションのうちのボタントランジションの数を示している（3.6.1 参照）。この表及び表 3.1 の全トランジション数より、

- それぞれのシステムにおいて使用されているトランジションのうち、約 63 %が自動生成可能であった。

- 自動生成されるトランジションの約 80 %がボタントランジションであった。

ということがわかった。以上より、プレース・トランジションの自動生成機能を用いてペトリネットを記述すると、ペトリネット記述の労力は大幅に削減できる。

表 3.6: プレースの類似指定

	通信販売	貸ビデオ	PENGUIN
全ウィンドウ数	3	16	34
基本ウィンドウ数	0	3	5
作成ウィンドウ数	0	9	26

表 3.6 では、それぞれのアプリケーションが持っているウィンドウの数（全ウィンドウ数）と類似指定を行う際にもととなったウィンドウの数（基本ウィンドウ数)、類似指定を行って作成されたウィンドウの数（作成ウィンドウ数）を示している。例えば、貸ビデオ店のアプリケーション（全 16 ウィンドウ）においては、3 種類のウィンドウをもとに 9 ウィンドウが作成されている。この表より、アプリケーション内の約 66 %のウィンドウは、ウィジェットの種類やその数といった構成によって、少数（約 15 %）の基本ウィンドウのどれかに類似しているということがわかる。従って、ダイアローグ記述担当者が一から作成するウィンドウはアプリケーション内の少数のウィンドウであり、その他の多くのウィンドウは、類似指定を用いて生成し、それを修正することにより作成することができる。

3.10.4　PENGUIN の記述容易性

ペトリネットと同様にアプリケーションの論理構造を記述することができる記述法の一つに状態遷移図がある。状態遷移図で記述できるものは全てペトリネットでも記述できる [39] ので、記述能力ではなく、記述の容易性という観点からの比較が必要となる。

GUI の発展により、一つのウィンドウ内に多数のウィジェットが含まれるようになってきている。そして、これらのウィジェットへの操作に対する順序関係の制約が緩和されてきている。本システムを用いた設計においては、GUI の生成、及び GUI 部とアプリケーション処理部との間のインタフェースの生成という観点から、アプリケーションの制御にかかわるウィジェットごとの記述が必要となる。例えば、ビデオデッキの制御アプリケー

ションにおけるタイマー録画においては、ビデオテープの挿入と時間・チャンネル等の設定の双方を行うことが必要であるが、この間の順序関係は規定されていない。この場合に二つの操作を明示した設計を状態遷移図を用いて行おうとすると、二つの操作がなされる順序の全ての組合せに対して状態を記述することが必要である。一方、ペトリネットでは、トークンの発火則を用いることにより、操作の順序の組合せを気にすることなく、容易に記述を行うことができる。本手法では、このような記述が頻繁に現れるようなアプリケーションを対象としており、ペトリネットが持つ記述容易性を選択した。

3.10.1、3.10.2、3.10.3 及び 3.10.4 より、既存の GUI アプリケーション構築ツールによりアプリケーションを開発する、又はプログラムを記述していく等の手間を考えると、GUI アプリケーション構築の労力は軽減されていることがわかる。

3.11 関連手法

これまでの GUI アプリケーション開発支援の分野においては、ダイアローグ記述、視覚的言語、GUI アプリケーションの自動生成、対話的な GUI アプリケーション構築ツール、コンポーネントによるアプリケーション構築ツールなどが研究されている。

ダイアローグ記述

ダイアローグ記述に関する手法 [25] では、ユーザイベントを処理してユーザとアプリケーションとの対話を制御することを主な目的としており、状態遷移図や文法を用いるものなどが提案されている。しかし、効率的に GUI アプリケーションを開発するための支援については大きな注意が払われていない。

視覚的言語

視覚的言語では、Prograph[26] や VisaVis[27] などが研究・実用化されている。これらの言語は、テキストベースの言語と同様に、アプリケーションの全体的な制御構造を把握するための支援がなされていないため、3.1 の条件 1 及び 2 を満たすアプリケーションを開発することは困難である。

GUI アプリケーションの自動生成

GUI アプリケーションの自動生成に関する手法では、データベースから自動生成するもの [28] やプログラムから自動生成するもの [29] 等が研究されている。しかしこれらは、ウィンドウの動的な振舞いの制御のための支援を欠いているなど、実用性に欠ける面がある。またこれらは、アプリケーションの全体的な制御構造を把握するための支援を行っておらず、3.1 の条件 1 及び 2 を満たすアプリケーションを開発するためには不十分である。

対話的に GUI アプリケーションを構築するツール

対話的に GUI を構築するツールとしては、Visual Basic[20] や Delphi[21] などのイベント駆動型アプリケーション構築ツールなどが開発されている。これらのツールは GUI アプリケーションのイベント駆動型以外のソフトウェアや、GUI 部分以外のアプリケーション部分の開発については支援していない。また、アプリケーションの制御構造全体を把握するための支援を行っていないため、アプリケーション内の各処理局面において、ウィンドウに必要なウィジェットを過不足なく配置されているかどうかを容易に確認することは困難である。

コンポーネントによるアプリケーション構築ツール

コンポーネントによるアプリケーション構築ツールとしては、Java Studio[33]、APPGALLERY[34] などが製品化されている。しかしこれらは、あくまでコンポーネントという標準化されたプログラムを組み合わせてアプリケーションを作成するものなので、標準化が困難な業務固有のアプリケーションで利用することは難しい。

3.12　あとがき

3.12.1　まとめ

本システム ”PENGUIN (Petrinet Editor for Navigator and Graphical User INterface)” は、開発者にとって大きな負担となっている GUI アプリケーションの開発労力を様々な面で軽減する目的で開発されたシステムである。PENGUIN では、GUI 情報を持ったペトリネットを使用して GUI アプリケーションの制御構造を記述し、それをもとにして GUI アプリケーションの自動生成を行なう。

ペトリネットは通常、GUI についての情報を持っていないため、ペトリネットのみでは GUI アプリケーションの骨組みを自動生成することはできない。従って本手法では、ペトリネットに GUI についての情報を付加することにより、アプリケーションの自動生成を行う。プレースにウィンドウに関する情報、つまり、各ウィンドウに配置するウィジェットの種類と個数、そしてそれらのウィジェットについての最少限の情報を持たせる。また、トランジションにユーザイベントのバインド情報を持たせる。

PENGUIN で自動生成されるものは、

- 各関数の骨格（処理内容の書き込まれていない関数とそのつながり）
- GUI プロトタイプ（ウィジェットの生成，ウィジェットに対するイベントの生起などの GUI に関するプログラム）
- ステートマネージャ（GUI アプリケーション実行中のトークンの位置を監視し、トランジションの関数の起動を制御することにより、GUI アプリケーションを記述されたペトリネットに忠実に従って動作させるための機構）

である。この時点では、アプリケーションの外観については PENGUIN が自動的に決定しており、ユーザの好み等は反映されておらず、関数の処理内容も記述されていない。そのため、GUI 定義担当者が GUI のレイアウトを既存のツールを用いて修正し、プログラミング担当者が関数の処理内容を記述することで GUI アプリケーションを完成させる。

3.12.2 PENGUIN の利点

PENGUIN には、以下のような利点がある。

- 複雑な GUI アプリケーションの開発が容易で，全体像を容易に把握することができる．
- ウィンドウの外観に関して，ユーザビリティを考慮した GUI を容易に構築することができる．

3.12.3 今後の課題

今後の課題としては、以下のようなことを挙げることができる。

- 機能の追加・削除や GUI 部のウィジェットの追加・削除など，ペトリネットの表面・内部における変更が生じるような保守を行う場合に，支援を行うシステムを構築すること
- コンポーネントウェア技術の概念を採り入れ，再利用性を向上させること

などについての拡張を行うことが考えられる。

第4章　ウィジェット変更時のプログラム自動変更支援手法

4.1　はしがき

コンピュータの爆発的な普及により、コンピュータに関する知識を持たない多くの人も、様々な理由によりアプリケーションを利用することが多くなった [1]。このような人がアプリケーションを使うためには、アプリケーションの機能（ユーティリティ）だけでなく使いやすさ（ユーザビリティ）が重要となる。これまでアプリケーションは、ユーティリティ重視、つまり 1 つのアプリケーションが持つ機能を増やすことを重視して開発が行われてきたが、上記の背景を考慮し、近年ではユーザビリティも重視されるようになってきた。

ユーザビリティを重視してアプリケーション開発を行うには、エンドユーザの手によって GUI を試行・評価し、その結果を反映して改良することが効果的である [4]。しかし、ユーザビリティを重視して GUI を構築するには、エンドユーザが求める GUI をエンドユーザ自身が明確に提示するだけではなく、評価を元に開発者が GUI を再構成し、エンドユーザに再評価してもらうという過程を繰り返すことも効果的である。そこで、ユーザビリティを向上させるためには、変更を前提として GUI を構築することが重要となる。

GUI は、ユーザビリティ上の問題からアプリケーションの開発段階での変更はもちろん、実際の運用段階でも、ユーザの好みの変化や他のアプリケーションとの兼ね合いなどにより、変更の要求が生じる場合がある。このような場合には、運用中のアプリケーションの GUI を変更する必要があり、これを考慮して GUI を構築する必要がある。

ウィジェットの利用に関しては、例えば JComboBox に設定する項目の適切な数や、JLabel に設定する文字列の適切な長さのような、指針は存在しない。従って実際のアプリケーションを使う状況やユーザの好みなどにより、開発者が適切であると考えたウィジェットの種類をエンドユーザによる評価次第で変更することは多々ある。

図 4.1 の例は、(a)、(b) ともに、選択したファイルを削除する際の例である。(a) は、JComboBox というウィジェットを利用し、(b) は DnDSelect というウィジェットを利用

している。

JComboBox を用いた例では、項目を選択し、「OK」を押すと処理が行われる。項目としてファイル名を登録し、「OK」を押すことによって行われる処理として、ファイル削除の処理を結び付けておくことで、ファイル削除を行う。DnDSelect は、与えられた項目名とイメージを上部に表示し、その項目を下部に表示されているイメージ上にドラッグ&ドロップすることで、その項目の名前を受け取り、処理が行われる。この例では、上部の項目として、ファイルの名前とファイルのイメージ（ファイル 1 〜 4）を与え、下部のイメージとしてごみ箱のイメージを与える。そして下部のイメージにファイル削除の処理を結び付けておくことで、ファイル削除を行う。

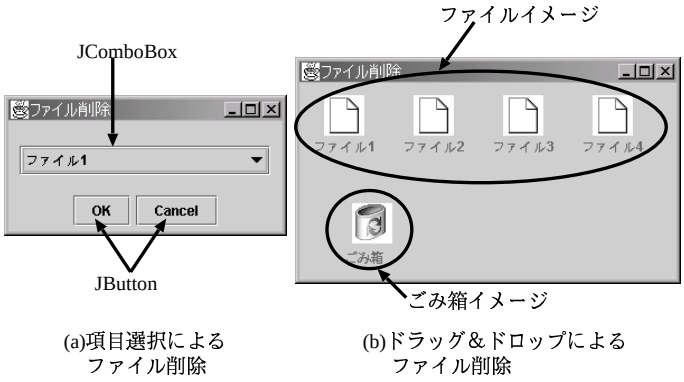


図 4.1: GUI 変更例

対象となるファイル数が多い場合には (a) の方が適しているが、少数の場合には (b) の方が一覧性の点で優れている。また、アプリケーションの開発段階では、対象となるファイルの数が未知であったために (a) を利用していたとしても、アプリケーションの運用段階において、対象となるファイルの数が少数であることが判明すれば、視認性という観点から、(b) への変更が要求されることも考えられる。

しかし、現状ではアプリケーションの GUI 部分を変更すると、その GUI 部に関連したアプリケーション処理部分との結合などもプログラミングしなおさなければならない。この際、GUI が複雑な構造を持っていた場合、プログラミングしなおす箇所を見つけることは困難であり、バグが生じないように正しく修正するように注意しなければならない。また、GUI の構造が単純であっても、修正箇所が多い場合には、修正箇所の発見及び修正に多大な時間と労力がかかり、バグも生じやすくなる。

従って、本研究ではこのようなウィジェットの種類の変更時におけるプログラム修正の労力を軽減することを目的として、GUI の変更容易性を実現することを目指し、GUI アプリケーションを生成し、その GUI を変更する。このために、ウィジェットの種類の変更の際に、GUI 部分とアプリケーション処理部を自動的に再結合する手法について提案する。まず本システムでは、 GUI のレイアウトを既存のアプリケーション開発環境のようにビジュアルに構築する。その上で、GUI 部とアプリケーション処理部の結合を手動でプログラミングしなおすことなく、ウィジェットの種類を交換することができる。また、一連の処理を実行するための GUI があった場合、それらのウィジェットをコピーして他の種類のウィジェットに交換することで、追加のプログラミングをすることなく、ユーザの多様なタスク実行操作方法を実現できる。

GUI の変更容易性の具体的な実現方法としては、あらかじめウィジェットの種類をその機能に応じて 5 つの役割に分類し、ウィジェットが持つメソッドも、その処理内容に応じて分類しておく。そして GUI 構築時に、個々のウィジェットが果たす役割を割り当てることで、そのウィジェットが別の種類に交換されたときに、交換前のウィジェットのメソッドを、交換後のウィジェットのメソッドのうち、同じ処理を行うものと自動的に交換する。

また、本研究ではプログラミング言語 Java を対象として GUI アプリケーションの生成及びその GUI の変更を行う。例として利用するウィジェットは、Java の Swing パッケージのものである。

4.2 本手法の特徴

本手法では、GUI の変更を容易に行うために、GUI 部とアプリケーション処理部との結合を保ったまま、ウィジェットの変更の支援を行う。

本手法の大きな特徴として、以下の 3 つが挙げられる。

1. ウィジェット変更に伴うプログラムの自動修正

2. ライブラリに用意されていないウィジェットの利用

GUI 変更によってプログラムを手動で修正する場合、プログラムの修正箇所の発見に大きな労力が必要となり、またプログラム修正時にバグが生じることもある。そこで 1. により GUI 変更に伴うプログラムの修正箇所の発見のための労力やプログラム修正時に生じるバグを削減することができる。またウィジェットは、同じ機能を果たすものであっ

ても、見た目や操作感に合わせて様々な種類のウィジェットが用意されている。それらのウィジェットの組み合わせにより、多様な GUI を作成することができ、ユーザの選択の幅を広げることができる。そこで 3. により、ライブラリに用意されていないウィジェットであっても変更可能とすることで、さらに多様な GUI を作成することができるようになり、ユーザの選択の幅もより広くなるため、ユーザビリティの向上にも貢献することとなる。

4.3　GUI アプリケーション生成指針

本手法では、あらかじめウィジェットの種類を、その役割ごとに分類している。アプリケーション開発時には、ウィジェットの役割を中心にユーザのタスクを分析し、役割をウィジェットと結びつけることで GUI を構築する。

4.3.1　ウィジェットの役割と手続き

ウィジェットの役割は、表示、入力（選択による入力）、選択、アクション要求、コンテナという 5 つに大きく分類することができる。これらの役割を果たすために、ウィジェットには、個々の役割に応じて共通の処理が存在する。例えば、「入力（選択による入力）」ならば、「入力されている値を返す」という処理により、ユーザの入力した値が、入力値として取得されることになる。以降、これら共通の処理を、手続きと呼ぶこととする。

本手法では、ウィジェットの種類をこれらの役割で分類することで、同じ役割を果たすことのできるウィジェット同士を交換した際に生じる、プログラムの修正部分（値の参照、取得などの、役割を果たすための共通な手続き）を自動変更する。

GUI としての機能を果たすために、ウィジェットには多数のメソッドが用意されている。本手法ではこれらのメソッドのうち、前述の 5 つの役割を果たすために用意されたものを扱うこととする。5 つの役割及びその役割に属するウィジェットの例、そして役割を果たすための主な手続きは以下の通りである。

表示　文字列や画像を表示する

例　JLabel, JList

手続き
- 表示する値を設定する
- 表示している値を削除する
- 表示している値を取得する

選択 ユーザ操作によって true か false かの 2 つの状態を反転させることによって、その項目の選択状態を表す

例 JRadioButton, JComboBox

手続き
- 選択用の値を設定する
- 選択用の値を削除する
- 選択された値を取得する

入力（選択による入力） キーボードからの入力や項目選択によって文字列を取り扱う

例 JTextField, JTextArea

手続き
- 入力されている値を取得する
- 入力されている値を削除する

アクション要求 ユーザ操作によって何らかの処理を行うよう要求を発する

例 JButton, JMenuItem

手続き
- ユーザ操作によって呼び出されるメソッドを持つオブジェクトを 登録する
- ユーザ操作に対して呼び出されるメソッドを持つオブジェクトを登録から削除する

コンテナ ウィジェットを配置するための敷物の役目を果たす

例 JFrame, JPanel

手続き
- ウィジェットを追加する
- ウィジェットを削除する

本手法では、ウィジェットが持つ、これらの手続きに対応する各メソッドをあらかじめウィジェット変換表に記述している。ウィジェット変換表には、上記の分類ごとにメソッド名、引数の型、返り値の型が記述されている。ライブラリに用意されていないウィジェット種類を使用する場合、そのウィジェットの種類を役割で分類し、手続きに対応するメソッドとともにウィジェット変換表に追加することで使用することができる。

4.3.2 GUI 構築の流れ

タスクの分析

本手法では、GUI 構築の観点からタスクの分析を行う。シナリオ等から GUI が必要となるタスクを抽出し、GUI 部とアプリケーション処理部で共通して使用される変数を取り

出す。そして、アプリケーション処理部と GUI 部において、その変数に対する扱いを決定する。具体的には、アプリケーション処理部では、その変数に対して行う処理内容と、その際に呼び出すメソッドを決定する。GUI 部は、その変数はユーザが入力するものか、あるいはユーザに表示するものか、というその変数の役割を分析し、変数に対して必要となるウィジェットの役割を決定する。そして、その変数に対するウィジェットの責務を設定する。責務とは、あるウィジェットに割り当てられた、GUI 中における具体的な役目を指す。また、それらの役割を果たすために必要な手続きも併せて決定する。最後に、その変数に対して決定されたウィジェットの役割ごとに、具体的なウィジェットを割当てる。

例として、時計の時刻設定のタスクを挙げて説明を行う。まず、このタスクを詳細に記述すると、「ユーザによる時刻の入力値を、時刻設定の処理を行う手続きに引数として渡す」となる。次に、このタスクの中で GUI 部とアプリケーション処理部に共通に使用される変数は、「時刻の入力値」が挙げられる。それらの変数を詳細化し、時刻の入力値として、時・分・秒の 3 つの変数とすることを決定する。そしてアプリケーション処理部では、これら 3 つの変数を使用し、呼出すメソッドを「setTime(int hour, int minute, int second)」と決定し、実装を行う。一方 GUI 部は、時・分・秒の 3 つの値をユーザが入力し、ユーザによる決定でこのタスクを実行するため、入力のためのウィジェットが 3 つと、アクション要求のウィジェットが 1 つ必要となる。そしてそれぞれのウィジェットに対し、「時の入力」、「分の入力」、「秒の入力」及び「時刻設定アクション」という責務を設定する。また、タスクを処理するためには、3 つの入力ウィジェットに対して、「入力値を取得する」という手続きが必要となる。

タスク分析結果による GUI 構築

タスクの分析に基づき、開発者はまず、本システムによってウィジェットを配置し、責務とその手続きを設定する。そして、アプリケーション処理部のメソッド呼出しや GUI 部との結合をプログラミングする。

時計の時刻設定タスクの例では、タスクの分析結果に基づいて入力のためのウィジェット A を 3 つ、アクション要求のためのウィジェット B を 1 つ配置し、それぞれに役割と責務を与えることにより、GUI を構築する。そして、タスクを実行するために必要となる手続きを、

〈秒の入力. 入力値を取得する〉

のように、具体的なメソッド名ではなく、手続き名を用いて構築する。開発者は、〈秒の入力. 入力値を取得する〉という GUI 部の手続きを、具体的なメソッド名の呼出しと同

様に扱い、アプリケーション処理部のメソッド呼出しの際にこの手続きの返り値を引数として与えることで、アプリケーション処理部の処理内容を記述する。

4.3.3 ウィジェット変換表

ウィジェット変換表とは、GUI としての役割ごとにウィジェットを分類した結果を記述したものである。GUI の機能を果たす際に必要となる手続きに対応するウィジェットのメソッドと、それらのメソッドのインターフェース（引数の数、引数の型、返り値の型）も併せて記述されている。

ウィジェット変換表は、本システム内に用意されたウィジェット変換表記述ツールを用いて記述し、その結果は XML 形式で保存される。ウィジェット変換表のノードには、以下のようなものがある。

⟨**Widget**⟩ **:** 1 つのウィジェットの情報を表す。

⟨**Name**⟩**:** ⟨Widget⟩ の子ノードで、ウィジェットのクラス名を表す。

⟨**Behavior**⟩**:** ⟨Widget⟩ の子ノードで、ウィジェットの振る舞いを表す。

役割ノード ⟨Behavior⟩ の子ノードで、ウィジェットの役割を表す。具体的なノードは、以下の通りである。

⟨**View**⟩**:** 「表示」の役割を表す

⟨**Input**⟩**:** 「入力」の役割を表す

⟨**Select**⟩**:** 「選択」の役割を表す

⟨**Action**⟩**:** 「アクション要求」の役割を表す

⟨Container⟩:] 「コンテナ」の役割を表す

またこのノードは、以下の属性を持っている。

plurality: 「plurality = ”true”」であれば配列変数ウィジェット、「plurality = ”false”」であれば単一変数ウィジェットを表す

手続き: ウィジェットの役割を果たすための手続きを表す。例えば、「入力されている値を取得する」の場合、⟨get_data⟩ というノードとなる。

⟨**argument**⟩: 手続きノードの子ノードで、手続きの引数を表す。このノードは、以下の属性を持っている。

no: この引数の順番を表す。

type: この引数の型を表す。

⟨**return**⟩: 手続きノードの子ノードで、手続きの返値の型を表す。

⟨**listener**⟩: ⟨Action⟩ の子ノードで、このウィジェットによって呼び出されるリスナのクラスを表す。

⟨**action_method**⟩: ⟨Action⟩ の子ノードで、⟨listener⟩ によって呼び出されるメソッド名を表す。

⟨**component_type**⟩: ⟨Container⟩ の子ノードで、このウィジェットに追加することのできるウィジェットのクラスを表す。

図 4.2 は、JRadioButton というウィジェットを「選択」の役割を持つものとしてウィジェット変換表に登録した際の XML 記述の例である。

```
<Widget>
  <Name>javax.swing.JRadioButton</Name>
    <Behavior>
      <Select plurality="false">
        <!-- public void setText(String t) -->
        <set_data>setText
          <argument no="1" type="java.lang.String" />
          <return>void</return>
        </set_data>
        <!-- public void setText(new String("")) -->
        <delete_data>setText
          <argument no="1" type="java.lang.String">new String("")
          </argument>
          <return>void</return>
        </delete_data>
        <!-- public String getText() -->
        <get_data>getText
          <argument no="1" type="" />
          <return>String</return>
        </get_data>
        <!-- public String getText() -->
        <get_selected_data>getText
          <argument no="1" type="" />
          <return>String</return>
        </get_selected_data>
      </Select>
  </Behavior>
</Widget>
```

図 4.2: ウィジェット変換表例

本手法ではアプリケーションの GUI 部を、責務とその手続きから構築し、ウィジェット変換表を用いることにより、その GUI 部を実際のプログラムコードに変換する。図 4.3 は、責務 1 と責務 2 にクラス 1 のウィジェットを割り当てている場合の変換である。

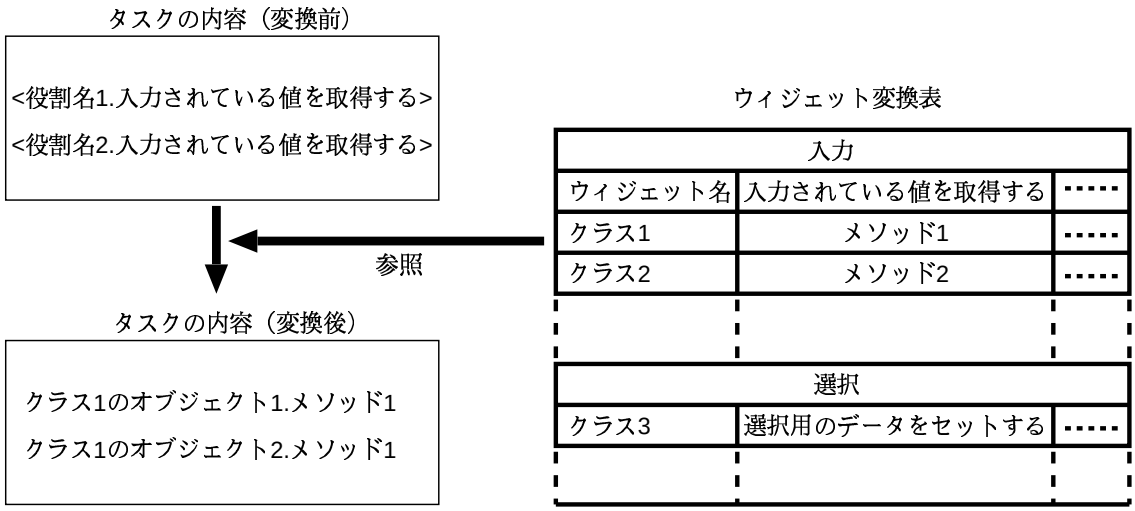


図 4.3: ウィジェット変換表によるプログラムコードへの変換

ウィジェット変換表は、事前に役割ごとに分類したウィジェットとそのメソッド名を記述した XML 形式のファイルを読み込ませることで設定する。ライブラリに用意されているウィジェットは、あらかじめウィジェット変換表に登録済みであるため、開発者が登録する必要はない。ライブラリに用意されていないウィジェットを利用する場合、開発者はウィジェットの仕様に従って分類を行い、ウィジェット変換表に追加することで利用することができるようになる。

4.3.4　GUI 変更指針

ウィジェットを交換する際には、まず本システムによって交換するウィジェットを選択する。これにより本システムは、ウィジェット変換表を参照し、交換前のウィジェットとそのメソッドを、交換後のものに置き換える。図 4.4 は、責務 1 を、クラス 1 のウィジェットからクラス 2 のウィジェットに変更した場合の変換である。

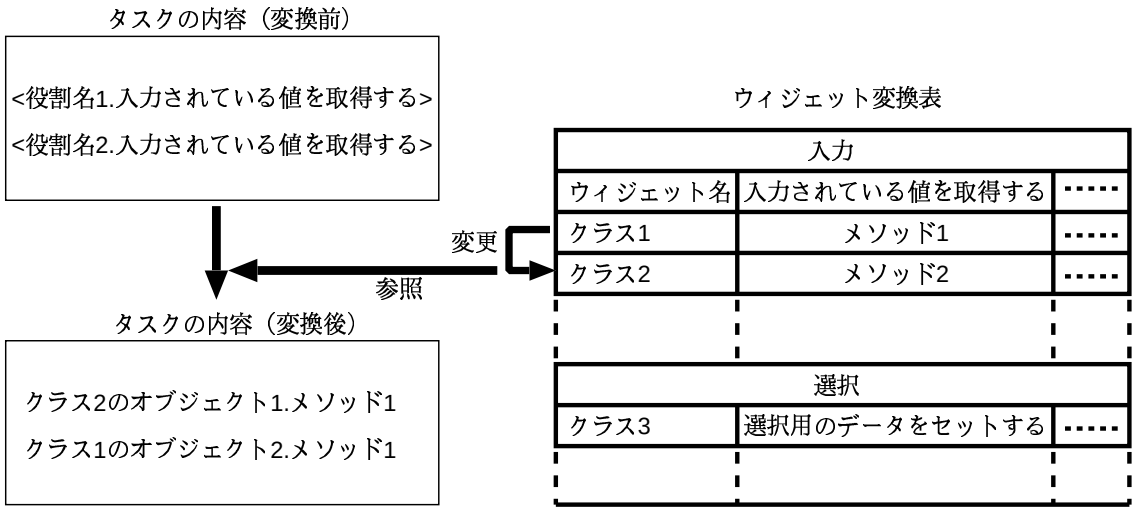


図 4.4: ウィジェット変更後のウィジェット変換表による変換

4.3.5 メソッド交換

ウィジェットの種類を交換する場合、自動変更する必要があるプログラム部分は、ウィジェットの生成と役割を果たすためのメソッドの呼出しである。ウィジェットの生成部分は、交換元のウィジェットの種類と交換先のウィジェットの種類は本システムが把握しているため、自動的に変更することができる。

また、メソッドの交換を行う際には、メソッドの「引数の数」、「引数の型」、「返り値の型」を変更する必要がある。役割を果たすためにウィジェットが持っている変数の形態は、単一の変数である場合と配列の変数である場合がある。単一の変数を持つウィジェットと配列の変数を持つウィジェットを交換する場合、これらの変数にアクセスするメソッドの引数の数が異なることが多い。この場合、ウィジェットが持つ変数に対するアクセス方法は類似しているため、メソッドは交換可能である。ウィジェットが持つ変数の形態と、その形態が異なるウィジェットを交換する際のメソッドの交換方法の詳細は 4.4 章で述べる。引数の型及び返り値の型に関しては、ユーザの操作によって設定・取得される値の型は、boolean 型か int 型、あるいは String 型のどれかであるため、Java において用意されている変換用のクラスを用いることで、自動的に変換することができる。

4.4　ウィジェットの持つ変数特性

ウィジェットの種類の交換に伴ってメソッドの自動交換を行う場合、引数の型と数、返り値の型が、交換元と交換先のメソッドで異なると、適切な形に変換しなければならない。本手法では、ウィジェットが役割を果たすために持つ変数に着目して、ウィジェットの種類を交換した際に生じるメソッド間の変換を行う。

4.4.1　単一変数ウィジェットと配列変数ウィジェット

ウィジェットは、役割を果たすために内部に変数を持っている。ウィジェットが上記の 5 つの役割を果たすための手続きは、これらの変数にアクセスすることで実現される。これらのアクセス対象となる変数は、単一変数と配列変数の場合に大別できる。

例えば、JLabel は「表示」のための変数を 1 つ持っており、これにアクセスするメソッドは、「表示する値を設定する」に対応する「void setText(String text)」、「表示している値を取得する」に対応する「String getText()」等がある。一方、JList は JLabel と同様に「表示」としての役割を果たすことができるが、JLabel と違い、役割を果たすための変数を配列の形で持っている。従ってこれらの変数にアクセスするメソッドは、「表示する値を設定する」に対応する「void setListData(Object[] listData)」、「表示している値を取得する」に対応する「Object getModel().getElementAt(int index)」となり、配列の変数にアクセスするために複数の値を一度に設定したり、変数の順番を指定して操作することとなる。本手法では、前者を単一変数ウィジェット、後者を配列変数ウィジェットと呼ぶ。

本手法では、この 2 つの変数ウィジェットのどちらかのウィジェットと、これらの組合せのウィジェットを対象とする。この 2 つの変数ウィジェット以外で構成されるウィジェット（複数の変数に 1 つのアクセスメソッドしか対応していない場合）の自動変換はユーザに注意を喚起するだけで特別なサポートはしないものとする。これは、本手法が対象としている Java の Swing に属するウィジェットの大部分が両者の変数ウィジェットに分類できるためである。また、新規に追加するウィジェットとして、JavaBeans が利用される可能性が高いが、こちらのアクセスメソッドに対する構成規則も該当するためである。

4.4.2　ウィジェットのグループ化

単一変数ウィジェットと配列変数ウィジェットを交換する場合には、複数個の変数から 1 個の変数へ、1 個の変数から複数個の変数への変換が必要となる。ある役割の単一変

数ウィジェット複数個に対し、同じ役割の配列変数ウィジェット 1 個を対応させる必要がある。

本手法では、複数の単一変数ウィジェットをグループ化することで一連の変数を 1 つのメソッドで扱うことができるように単一変数ウィジェットのメソッドを変換する。また、グループ化された単一変数ウィジェットが持つそれぞれの変数に配列のように番号を付加することで、配列変数ウィジェットの配列の各要素と自動的に対応づけることが可能である（図 4.5）。

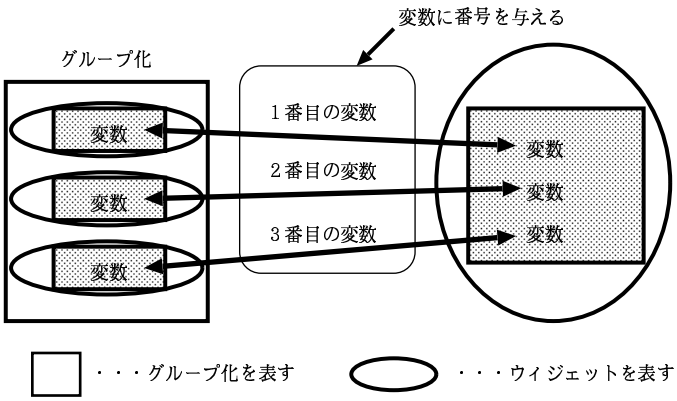


図 4.5: グループ化と変数の対応

以下において、単一変数ウィジェットから配列変数ウィジェットへの交換の例として、「選択」や「入力（選択による入力）」の役割を持つウィジェット同士（JRadioButton と JList）の交換について述べる。

「選択」及び「入力（選択による入力）」の役割を果たすためには、「選択用の値を設定する」及び「選択された値を取得する」という手続きがある。JRadioButton の場合、単一変数ウィジェットであるため、「選択用の値を設定する」に対応するメソッドも「void setText(String text)」となり、引数は設定する値そのものを与えることとなる。そのため、複数の選択項目を持つ GUI として単一変数ウィジェットを使うと、選択項目の数だけウィジェットを生成・設定する必要が生じる。また、単一変数ウィジェットの場合、「選択された値を取得する」に対応するメソッドは通常用意されていない。従って、その項目に対応するウィジェットの「選択状態を取得する」という手続きを用いて、この結果が true であれば「設定された値を取得する」という手続きによって値を取得し、「選択された値を取得する」を実現する。つまり、2 つの手続きを組み合わせて利用する必要がある。

一方、配列変数ウィジェットである JList の場合、「選択用の値を設定する」に対応するメソッドは、「void setListData(Object[] listData)」となり、引数が複数の変数を表す配列型となる。そのため、複数の選択項目を持つ GUI として配列変数ウィジェットを使う場合、値の設定は 1 つのメソッドで処理することができる。また、「選択された値を取得する」という手続きに対するメソッドも用意されており、ユーザがウィジェット中の項目を選択状態にしていると、その選択状態にされた項目を取得することができる。JList の場合、「Object[] getSelectedValues()」が「選択された値を取得する」に対応するメソッドとなる。

逆に、配列変数ウィジェットを単一変数ウィジェットに交換する場合、「選択された値を取得する」は、まず、単一変数ウィジェットの各ウィジェットで「選択されている状態を取得する」という手続きを用いて選択されているウィジェットを抽出する。その上で選択されているウィジェットだけ「選択用の値を取得する」という手続きを使って値を配列に格納して取得するようにプログラムを変換する。他の役割のウィジェットに関しても、単一変数ウィジェットと配列変数ウィジェットの間のメソッドの変換は同様に行う。

4.5　本システムの構成

本手法では、あらかじめウィジェットの種類を、その役割ごとに分類している。アプリケーション開発時には、ウィジェットの役割を中心にユーザのタスクを分析し、役割をウィジェットと結びつけることで GUI を構築する。本システムの構成を図 4.6 に示す。以下において、選択したファイルの削除の例（図 4.1）を用いて GUI の構築からウィジェットの種類の変更までの手順は以下の通りである。

1. タスク分析
2. GUI 構築
   - **2.2** ウィジェットの配置
   - **2.3** 役割・責務の設定
   - **2.4** GUI 処理部の構築
   - **2.5** GUI 処理部とアプリケーション処理部の結合
3. GUI 変更

以下においてそれぞれの手順について詳しく述べる。

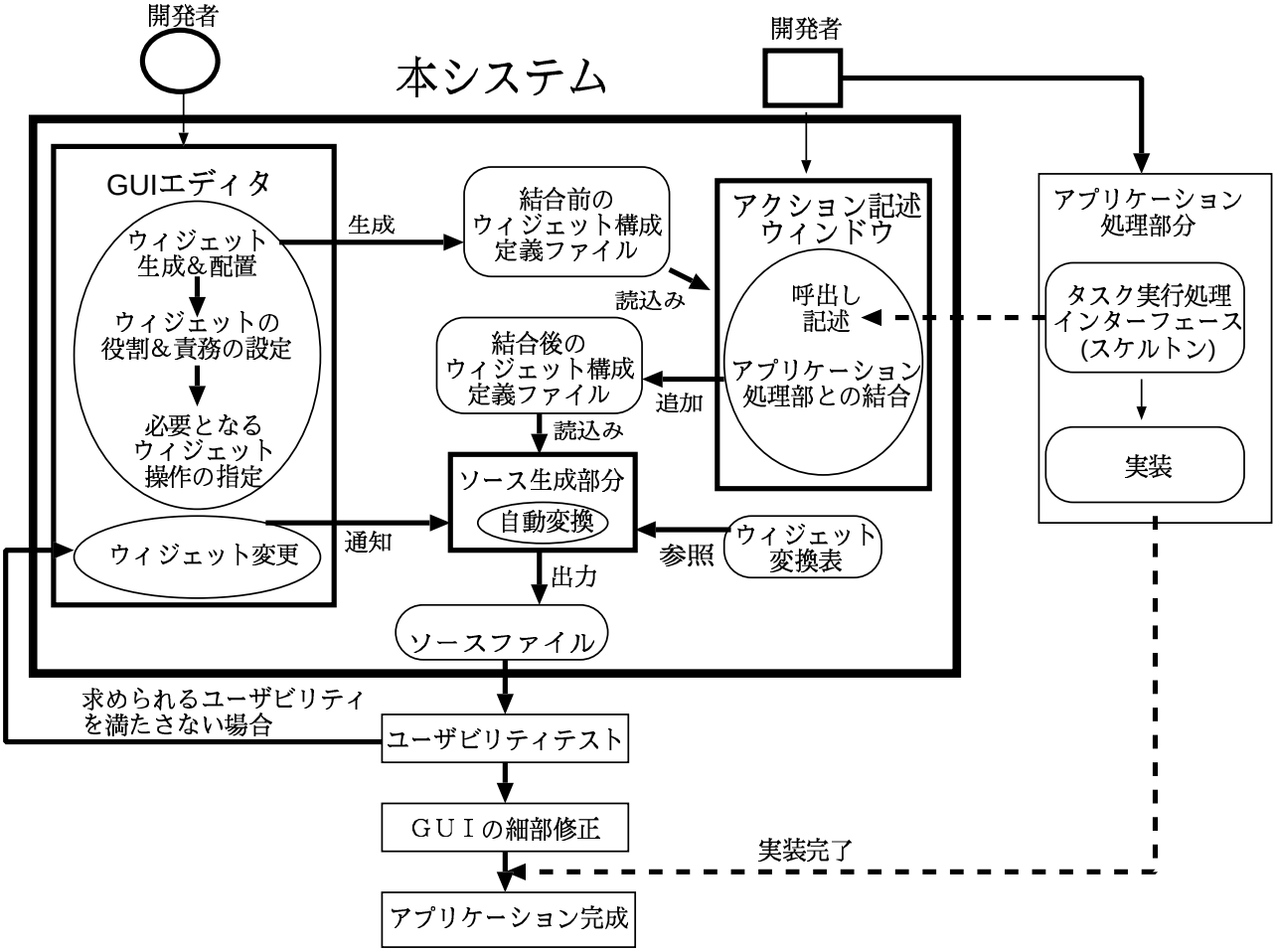


図 4.6: システムアーキテクチャ

4.5.1 タスク分析

まず開発者が、「ファイルを選択して削除する」というタスクを分析する。この例の場合、共通の変数として、「ファイル名」が考えられる。これに対し GUI 部では、「選択による入力」の役割を持つ「ファイル名選択」、及びユーザによるタスク実行を促すための「アクション要求」の役割を持つ「削除アクション」というそれぞれの役割及び責務と、

タスクを実行するために「選択されたファイル名を取得する」、「入力されたファイル内容を取得する」という手続きが必要となる。一方、アプリケーション処理部では、「ファイル名」という変数を引数とする「ファイル入力メソッド」が必要となる。

4.5.2　GUI 構築

ウィジェットの配置

開発者はタスク分析の結果をもとにして、本システムの GUI エディタを用いて既存のアプリケーション開発環境のようにビジュアルにウィジェットをウィジェットを配置する。この際、ウィジェットの初期設定として必要な値を設定する。

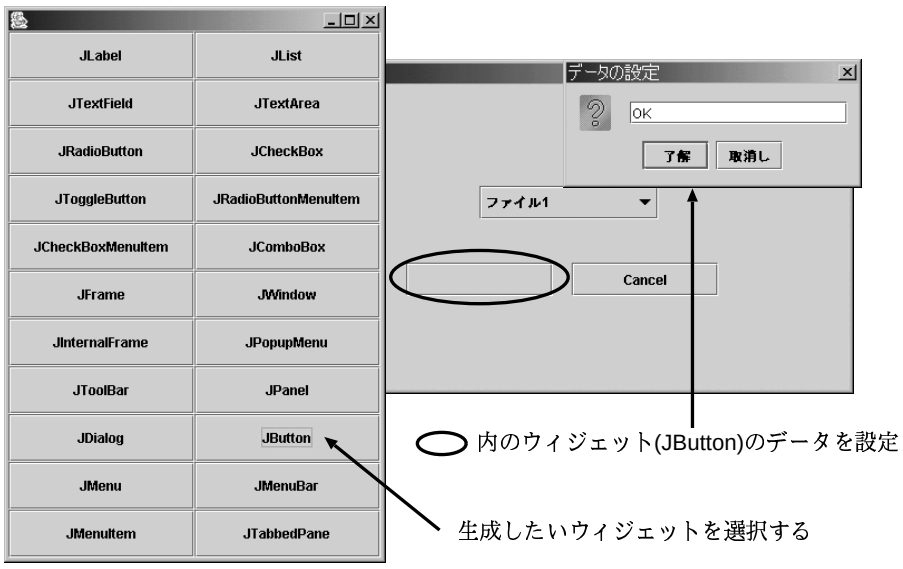


図 4.7: ウィジェットの配置と値の初期設定

役割・責務の設定

ウィジェットを配置した後、そのウィジェットが果たす責務を入力し、かつ、5 つの役割から、そのウィジェットが表す役割を選択する。図 4.1(a) の例では、JComboBox によって表現されているウィジェットの責務を「ファイル選択」、役割を「入力（選択による入力）」と設定している (図 4.8)。

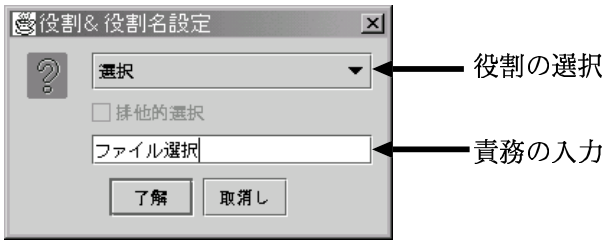


図 4.8: 役割と責務の設定

GUI 処理部の構築

次に、ユーザからのイベントが起こった時に、処理を行うために値を受け渡すウィジェットに割り当てられた責務を、アクション要求の役割を持つウィジェット（ユーザの操作によって呼び出される処理を行うメソッドを持つウィジェット）と関連付ける。

例では、「ファイル選択」を「削除処理」に関連付けている。値のやりとりを行うウィジェットの手続きを記述する際には、5 つの役割のうち、関連付けされたウィジェットの役割の手続きの中から必要な手続きを指定する。この例では、「選択された値を取得する」という手続きが指定されている（図 4.9）。この手続きの選択により、「選択されたファイル名」を取得できることとなる。

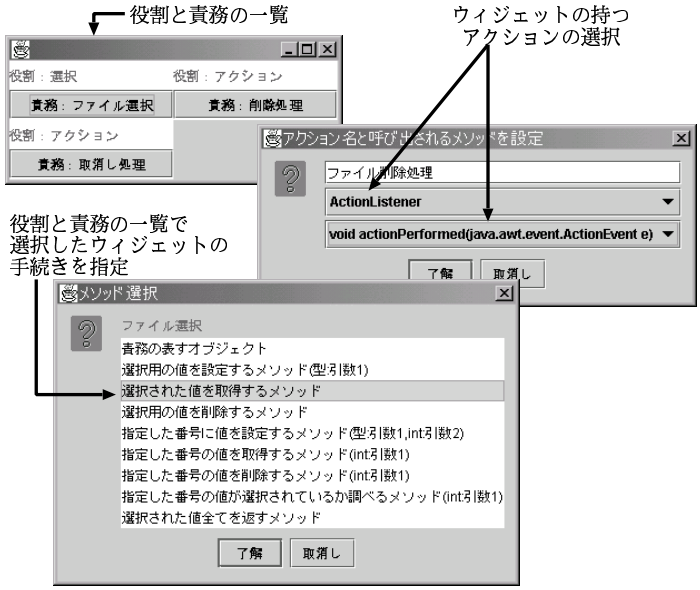


図 4.9: 役割と手続きの関連付け

GUI 部とアプリケーション処理部の結合

続いて、本システムの処理記述ウィンドウを通して、メソッドの呼出し等のアプリケーション本体のプログラム記述を行い、ユーザの操作に対する処理を完成させる（図 4.10）。このプログラム記述では、選択したウィジェットの手続きの記述はメソッドと同様に扱う。この際、引数が必要となる時は、一定の形式に従って記述する。返り値の型も同様である。この内容がウィジェット構成定義ファイルに追加される。本システムでは、ウィジェットのレイアウト情報などもウィジェット構成定義ファイル中に記述される。

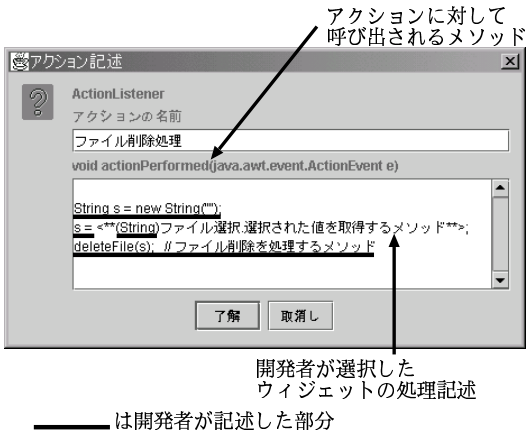


図 4.10: ユーザの操作に対する処理の記述例

以上の情報をもとにして GUI アプリケーションの GUI 部分に関するソースコードを生成する。

ウィジェット構成定義ファイルには、ウィジェットの位置や大きさなどのレイアウトに関する設定や、ウィジェットに設定された役割と責務、図 4.10 のプログラム記述などが記述される。

4.5.3 GUI 変更

ウィジェットに対して指定した役割をもとにして、同じ役割を果たす別のウィジェットの種類を選択することでウィジェットの種類を交換することができる。図 4.1 の変更例では、JComboBox ウィジェットの役割である「選択」をもとにして、同じ「選択」の役割を持つウィジェット候補の一覧から DnDSelect ウィジェットを選択し、交換している（図 4.11）。この際、JComboBox に割り当てられていた「ファイル選択」という責務は、DnDSelect の責務として割り当てられる。

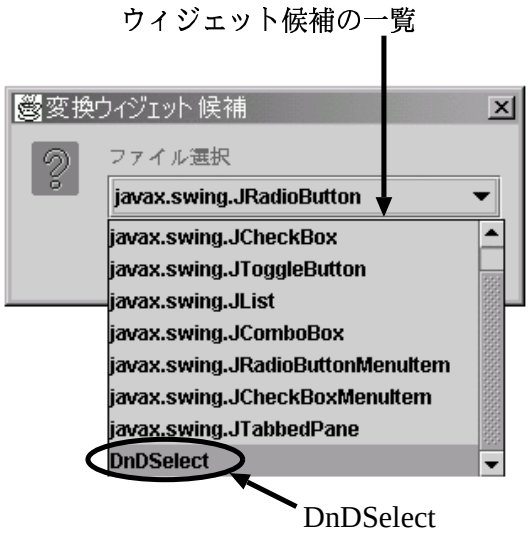


図 4.11: 変換候補の選択

本システムは、ウィジェット変換表（図4.12）を利用することで、ウィジェット構成定義ファイルに記述してあるウィジェットの処理を自動的に変換する。図 4.12 の例の「JComboBox」と「DnDSelect」は、予め同じ「入力（選択による入力）」の役割に分類され、その役割の「選択された値を取得する」というウィジェットの手続きに対して、「getSelectedItem」と「getDroppedText」というメソッドを対応するものとして分類している。返り値の型は、ユーザの操作に対するプログラム記述（図 4.10）の時に開発者が指定された形式に自動変換する。DnDSelect ウィジェットを利用する際には、ファイルイメージとごみ箱イメージは手動で設定する必要があるが、選択項目名を設定する等のウィジェットの処理は自動的に変換される。

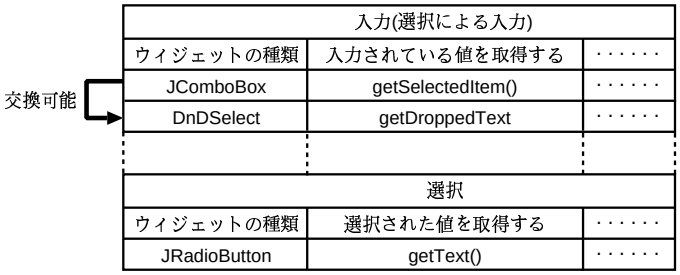


図 4.12: ウィジェット変更処理

4.6 評価

4.6.1 GUI 変更作業

次に、本システムを使用した場合と、1 からプログラミングした場合、既存のアプリケーション開発環境を使用した場合に対して、カレンダーアプリケーション [41] の GUI を構築し、それぞれの方法で GUI を変更した。なおこのアプリケーションは、文献 [41] に掲載されている例を用いた。アプリケーション開発環境は、Forte for Java[5] を使用した。変更前のアプリケーションのウィンドウ例を図 4.13 に、変更後のウィンドウ例を図 4.14 に示す。

このアプリケーションは、日付ごとに一行のメモを記入して参照するものである。図 4.13 は、変更前のものは、月の表示と 1 ヶ月ごとの切り替えに JTabbedPane という、タブを用いて排他的に項目を選択することのできるウィジェットを使用しており、タブをダブルクリックすることで 1 ヶ月ごとのメモの一覧を表示できる。図 4.14 は、変更後のものは、月の表示を 12 個の JCheckBox に変更し、これらを多重選択を可能した。こちらは、それぞれの月のカレンダーをウィンドウごとに表示させるようにし、メモの一覧を表示させるためのユーザイベントを、月のカレンダーのウィンドウの年月表示をダブルクリックすることで実現するように変更したものである。そして変更前と変更後の GUI を比較した。

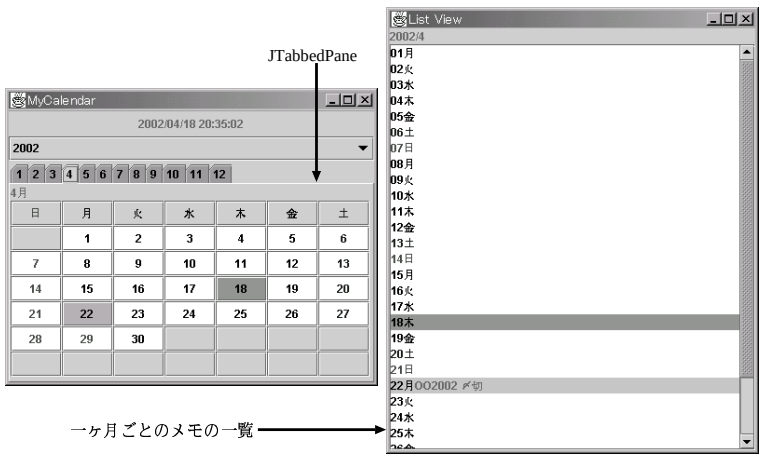


図 4.13: 評価用アプリケーション変更前

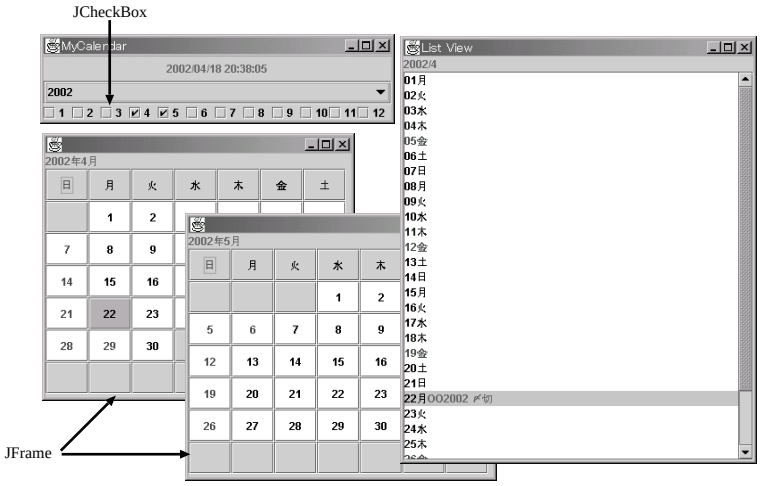


図 4.14: 評価用アプリケーション変更後

比較の結果を表 4.1 に示す。表 4.1 は、「手動で構築」は 1 からプログラミングしたもの、「開発環境」はアプリケーション開発環境を使用したもの、「本システム」は本システムによって構築・変更したものを示す。これらによる変更に関して、「削除」及び「追加」した命令文の数、それらの命令文が「自動」と「手動」のどちらの変更に該当するか、自動修正をさせるためのユーザのシステムに対する「操作」の回数を示したものである。「操作」は、アプリケーション開発環境の場合、「表示させる文字を入力する」や本システムの場合、「責務をつける」などのある単位のユーザの操作回数を数えるものとした。

表 4.1: GUI 変更作業の比較

	削除	追加	自動	手動	操作
手動で構築	13	32	/	45	/
開発環境	18	134	114	38	40
本システム	17	47	54	10	20

表 4.1 の結果より、GUI を変更した時に単に変更部分が少ないものは「削除」と「追加」の合計が 45 個の「手動で構築」であるが、本システムでは、システムに対して操作することで変更部分のうち、約 84% を自動修正することができる。その結果、手動の修正個所は 10 箇所となり、3 つ手法の中で最小である。手動で修正を行った箇所は、現時

点の本システムではサポートしていなかったウィジェットの操作の部分と、JTabbedPane のような排他的選択しかできない 1 つのウィジェットから複数の JCheckBox のように多重選択できるものへの変更の部分である。また、操作方法等が異なるため一概には比較することはできないが、操作回数も開発環境と比べて約半分となっており、本システムでは既存の開発環境よりも GUI の変更の労力は少ないと考えられる。「追加」の項目において、本システムと同様にグラフィカルな配置によってウィジェットの生成を行う既存の開発環境に比べて、本システムの変更数が約 35% に収まっているのは、既存の開発環境の場合ではウィジェットの個数分だけ、それぞれ命令が必要であるところを、本システムでは単一変数ウィジェットのグループ化により、これらのグループを配列のように扱うことで、数行で処理を行うことができるためである。

以上より、本システムでは、1 からプログラミングする、あるいは、既存のアプリケーション開発環境を利用するよりも容易にウィジェットの種類の変更とそれに伴うアプリケーション本体の修正を行うことができることを示すことができた。

4.6.2 GUI 構築作業

評価のために、本システム内に用意されているウィジェット変換表記述ツールを、本システムを利用して構築した。その際の GUI 構築の作業工程を、1 からプログラミングした場合と、既存のアプリケーション開発環境を使用した場合に対して比較・分析した。

表 4.2 は、本システム内に用意されているウィジェット変換表記述ツールを本システムで構築した際に本システムを用いて行った操作の内訳である。それぞれ、ウィジェット 1 つに対して行われる作業を 1 回として数えた。「ウィジェットの配置作業」は、ウィンドウ上にウィジェットを配置した回数、つまり作成したウィジェットの数と同一である。「初期値の設定」は、各ウィジェットが保持する値の初期値の設定である。「役割と責務の設定」は、各ウィジェットに対して役割と責務を割り当てた回数である。そして「手続きの指定」は各ウィジェットが保持する値に対してアクセスするために指定した手続きの個数である。また「割合」は、総作業回数に対して各項目が占める割合である。

表 4.2: GUI 構築作業の内訳

	回数	割合
ウィジェットの配置作業	28	14%
初期値の設定	15	7%
手続きの指定	137	66%
役割と責務の設定	26	13%

本システムでは、ウィジェットの配置や初期値の設定は、既存のアプリケーション開発環境と同様に、

1. 配置するウィジェットを選択する
2. ウィンドウ上で、そのウィジェットを配置する位置を指定する
3. ウィジェットの大きさやウィジェットが持つ値の初期値を設定する

という手順で行うことができる。従って表 4.2 の「ウィジェットの配置作業」と「初期値の設定」にかかる労力は、既存の開発環境と同様であると考えられる。「手続きの指定」は、既存の開発環境では各ウィジェットの値にアクセスするためにメソッドを指定することにあたり、本システムではメソッドそのものの名前ではなく 4.3.1 で示したメソッドの種類を選択するものである。従ってこれも、必要な労力は既存の開発環境と同様である。また、これら「ウィジェットの配置作業」、「初期値の設定」及び「手続きの指定」は、本システムではこれらはビジュアルに指定することができ、プログラムコードは自動生成される。従ってこれらは、1 からのプログラミングと比較して必要は労力は少ないものと考えられる。役割と責務の設定は他の手法では行われないが、表 4.2 では 13% と、ウィジェットの配置作業の割合とほぼ同等である。従って他の手法比べて責務の設定は、ウィジェットを配置する際に設定するデータが 1 つだけ増えることに相当する。

また、ウィジェットの責務に関して、今回使用したウィジェットの数は 28 個であり、そのうち責務を設定していないウィジェットは 4 個であった。これは、今回のアプリケーションでは手続きを必要としないウィジェットである。コンテナの役割として配置のみに使用したウィジェットと、同じく値を設定したラベルがこれに該当する。また、設定した責務は 26 個で、そのうち 2 つのウィジェットを 2 つの役割と責務を設定した。本手法では、これらの役割の重複もシステムを通じて変更するため、他の手法のようにウィジェッ

トを生成し、役割を果たすための機能を再構築する必要がなく、負担とはならないと考えられる。

以上より、本システムを利用して GUI を構築する際には、他の手法と比較して、労力が大きく増加することとはならないと考えられる。

4.7 関連手法

GUI アプリケーションの開発に関しては、アプリケーション開発環境、GUI の自動生成、GUI 変更時の自動プログラム修正などが研究・実用化されている。

アプリケーションの開発環境

アプリケーションの開発環境に関しては、VisualAge[6] や JBuilder[7] などの種々のツールが開発され、製品化されている。これらの製品では WYSIWYG（What You See Is What You Get）が実現されており、GUI に関してもビジュアルにウィジェットを配置し、それぞれの色や大きさ等の属性もビジュアルに設定することができる。しかしこれらの開発環境では、GUI の構築後にウィジェットの種類を変更する場合には、交換されたウィジェットとアプリケーション処理部は手動で連結しなおす必要がある。従ってこれらは GUI に関しては、構築容易性に対して大きな注意が払われており、ウィジェットの種類の変更容易性に対しては大きな注意は払われていない。

GUI の自動生成

GUI の自動生成に関しては、タスクと呼ばれる処理の単位をモデル化し、そこから自動生成するもの [11] や、UML のコラボレーション図から生成するもの [8] など様々な手法が提案されている。しかしこれらは GUI 自動生成のみに注目されており、ウィジェットの種類の変更に関しては注意が払われていない。

GUI の自動変更

GUI 変更時の自動プログラム修正として、アプリケーションのソースコードから GUI を生成し、その GUI をカスタマイズする機能を提供するツール [42] や、アプリケーション実行時に GUI をカスタマイズする手法 [30] などが研究されている。しかし、独自に作成したウィジェットなどライブラリに用意されていないウィジェットの種類を追加することは困難であるため、GUI 変更の自由度に制限が加わることとなる。

4.8　あとがき

本研究では、GUI の変更に伴うウィジェットの種類交換時のウィジェットとアプリケーション処理部との自動再結合を実現する手法を提案した。同じ役割を果たす単体のウィジェット間の交換だけでなく、ウィジェットのグループ化により、単体のウィジェットと複数のウィジェットを交換した際の自動修正も実現した。

今後の課題としては、

- サポートできるウィジェット操作の範囲を広げること
- 本システムを介さずプログラムを変更した場合でも、本システムでの再構築ができるように改良すること

などが挙げられる。

第5章　おわりに

5.1　本研究の成果

本論文では、複雑な GUI アプリケーションの開発に際して開発者の負担を軽減し、また GUI のユーザビリティの向上を図るための手法を提案し、その有効性について評価を行った。

以下に、提案した手法の概要と得られた成果を述べる。

エンドユーザ指向 GUI プロトタイプ生成手法

エンドユーザの視点を最大限に反映させて GUI を構築することを目的として、ユースケース図とシナリオという、エンドユーザにとっても理解しやすい記述を用いてアプリケーションの GUI 部のプロトタイプを生成する手法を提案し、この手法を実現するために GUPPY というシステムを開発した。ユースケース図とシナリオは、ソフトウェア開発保守ライフサイクルの要求分析という上流行程における成果物であり、これらを実装というライフサイクルの下流行程に結び付けることは困難であると言われるが、結び付けることができたということは大きな成果である。また、ユースケース図とシナリオという、UML を用いたオブジェクト指向アプリケーション開発の要求分析の段階において作成される記述を用いるため、GUI プロトタイプの生成のための特別な記述を開発者に求める必要がない。これは他の研究・提案されている手法では見られない利点である。

実際の評価においては、3 つのアプリケーションの GUI プロトタイプを生成し、生成されたウィジェットについて、あらかじめ開発されていた同一アプリケーションと比較を行った。その結果、70% 前後のウィジェットが、実際のアプリケーションと一致した。この結果により、生成された GUI プロトタイプは、実際のアプリケーションの GUI 部の基盤として、十分利用することができることを示すことができた。

設計結果の実装への自動反映手法

アプリケーションの設計結果と実装の間の矛盾を解消することを目的として、ペトリネットを用いて GUI アプリケーションの制御構造を記述し、そこからアプリケーションの GUI 部のプログラムと関数の骨組みを生成する手法を提案し、この手法を実現するために PENGUIN というシステムを開発した。この手法は設計結果を直接実装に結び付けるため、矛盾が起こることなく設計結果と実装を結び付けることができたということは大きな成果である。また、ペトリネットの特性を生かして、アプリケーション実行時のペトリネットのトークンの位置を監視し、アプリケーションの動作をペトリネットに忠実に従わせるためのステートマネージャという機構も併せて生成した。このステートマネージャの存在により、アプリケーションがペトリネットに忠実に従って動作することが保証されるため、プログラミング担当者は、関数同士の関係やアプリケーションの制御に注意する必要がなく、PENGUIN によって生成された各関数の処理内容のみに注意し、記述することでアプリケーションが完成する。これは、プログラミング担当者の労力軽減という点で大きな成果である。

本手法の評価としては、あるアプリケーションを 3 つの手法で開発した。その結果、本手法は他の手法の約 65%〜88% の時間で開発することができた。アプリケーションの開発には、開発者の各手法に対する熟練度等が評価結果に影響するため、一概に比較することはできないが、この結果により、本手法は他の手法と比べても実用的には問題ないものであることがわかる。

ウィジェット変更時のプログラム自動変更支援手法

ユーザビリティを向上させるための GUI の変更に対して、変更容易性を実現することを目的として、ウィジェットの種類を変更した時の変更されたウィジェットとアプリケーション処理部とを自動的に再結合する手法を提案した。GUI アプリケーション開発を支援するためのツールや手法は様々なものが存在するが、ウィジェットが変更されたときに変更されたウィジェットとアプリケーション処理部を結合し直すための手法は少なく、また提案されている手法も、ライブラリに用意されているウィジェットを扱うことは可能であるが、ライブラリに用意されていないウィジェットを扱うことはできない。従って、変更されたウィジェットとアプリケーション処理部を自動的に再結合し、さらにライブラリに用意されていない新しいウィジェットを利用するための手法を提案したことは大きな成果である。

本手法の評価として、あるアプリケーションの GUI を変更した結果、アプリケーション処理部で修正が必要な部分のうち、約 84% が自動修正された。この結果により、本手法を用いると、ウィジェット変更時のアプリケーション処理部の変更のための労力を大幅に削減することができるということが示すことができた。

5.2 今後の課題

今後の課題としては、以下のようなものが挙げられる。

GUI アプリケーションの操作の効果的な学習方法の提案

アプリケーションの多機能化に伴い、GUI もその操作方法が複雑化している。このような GUI アプリケーションの操作方法を習得することは困難であり、大きな労力が必要となっている。これは、企業などで必要なアプリケーションを導入する際に、ユーザの教育に膨大な時間やコストがかかることとなり、新たなアプリケーションを導入する上で大きな障害となる。そこで、大規模なアプリケーションの操作方法を効率的に学習し、習得することを支援するための手法が必要である。

種々のウィジェットの利用支援

アプリケーションが多機能化を続け、また近年のユーザビリティ重視の傾向により、ウィジェットの種類自体が増加してきている。それらのウィジェットは、従来から存在するウィジェットと同じ機能を持ちながら操作感が異なるものから、従来のものとは全く別の機能を持ったもの、従来のものを組み合わせた複合ウィジェットまでさまざまなものが存在する。このようなウィジェットは、多機能なアプリケーションを効率的に利用するため、またユーザビリティ向上のために不可欠なウィジェットとなりつつある。しかし、これらのウィジェットには、プログラム中での利用方法が複雑なものもあり、アプリケーション開発者の負担を増加させる原因になる可能性がある。従って、これら複雑なウィジェットやライブラリに用意されていないウィジェットを効率的に利用することを支援する手法が望まれる。

ライフサイクル全体にわたる支援

ソフトウェア開発ライフサイクルは、要求分析・設計・実装・テスト・保守の 5 つの局面から成り立っている。このうち本研究では、主に設計・実装・保守の 3 つの局面についての支援を行った。今後は、本研究において支援を行っていない要求分析及びテストの局面を支援する手法について提案し、ソフトウェア開発ライフサイクルの 5 つの局面を統合的に支援することが望まれる。

謝辞

本研究を進めるにあたり、御指導頂いた深澤良彰教授に深く感謝致します。また、本研究について様々なご支援、ご指示を頂いた上田和紀教授、筧捷彦教授、二村良彦教授はじめ、早稲田大学理工学部情報学科の諸先生方に深く感謝致します。

最後に、様々な御助言、御助力を下さった早稲田大学理工学部情報学科深澤研究室研究員の皆様に感謝致します。

参考文献

[1] X. Ferre, N. Juristo, H. Windl and L. Constantine: *Usability Basics for Software Developers*, IEEE Software Vol.18, No.19, pp.22-29 (2001)

[2] A. Lecerof and F. Paterno: *Automatic Support for Usability Evaluation*, IEEE Transactions of Software Engineering, Vol.24, No.10, pp.863-pp.887 (1998)

[3] 山岡俊樹, 鈴木一重, 藤原義久編著, (社) 人間生活工学研究センターユーザインタフェース設計委員会 SIDE 実証研究会編: 構造化ユーザインタフェースの設計と評価, 共立出版 (2000).

[4] ヤコブ・ニールセン著, 篠原稔和監訳, 三好かおる訳: ユーザビリティ・エンジニアリング原論, トッパン (1999).

[5] Forte for Java: http://www.sun.co.jp/forte/ffj/

[6] VisualAge: http://www-6.ibm.com/jp/software/ad/vajava/

[7] JBuilder: http://www.borland.co.jp/jbuilder/

[8] M. Elkoutbi, I. Khriss and R. K. Keller: *Generating User Interface Prototypes from Scenarios*, IEEE International Symposium on Requirements Engineering (RE'99) (1999).

[9] E. Arisholm, H. C. Benestad, J. Skandsen and H. Fredhall: *Incorporating Rapid User Interface Prototyping in Object-Oriented Analysis and Design with Genova*, Nordic Workshop on Programming Environment Research (NWPER'98) (1998)

[10] P. P. Silva, To. Griffiths and N. W. Paton: *Generating User Interface Code in a Model Based User Interface Development Environment*, Proc. of Advanced Visual Interfaces(AVI2000) (2000).

[11] M. Ikeda, Y. Takata and H. Seki: *Formal Specification and Implementation Using a Task Flow Diagram in Interactive System Design*, Proc. of The 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCE2001) (2001)

[12] A. Lecerof and F. Paterno: *Automatic Support for Usability Evaluation*, IEEE Transactions of Software Engineering Vol. 24, No. 10, pp863-887 (1998)

[13] R. Mahajan and B. Shneiderman: *Visual & Textual Consistency Checking Tools for Graphical User Interfaces*, IEEE Transactions on Software Engineering, Vol. 23, No. 11 (1997)

[14] G. シュナイダー, J. ウィンターズ著, オージス総研訳, 羽生田栄一監訳: ユースケースの適用:実践ガイド , ピアソン・エデュケーション (2000).

[15] グラディ・ブーチ著, オージス総研オブジェクト技術ソリューション事業部訳: UML ユーザーガイド , ピアソン・エデュケーション (1999).

[16] C. S. Horstmann and G. Cornell 共著, 福龍興業訳: コア Java 2 Vol1 基礎編 (2000).

[17] 茶筌, http://chasen.aist-nara.ac.jp/

[18] デジタル類語辞典,
http://www.junglejapan.com/ruigo/

[19] P. P. Silva and N. W. Paton: *UMLi: The Unified Modeling Language for Interactive Applications*, Proc. of Third International Conference on the Unified Modeling Language (UML2000) (2000).

[20] Microsoft Corporation: *Microsoft Visual Basic 6.0 Programmer's Guide*, Microsoft Press, (1998).

[21] S. Teixeira and X. Pacheco: *Delphi 5 Developer's Guide*, Sams, (1999).

[22] C. Janssen, A. Weisbecker and J. Ziegler: *Generating User Interfaces from Data Models and Dialogue Net Specifications*, Proc. of the Conference on Human Factors in Computing Systems(CHI'93) (1993).

[23] R.J.A. Buhr: *Use Case Maps as Architectural Entities for Complex Systems*, IEEE Transactions on Software Engineering, Vol.24, No.12, pp.1131-1155 (1998).

[24] UCM Navigator:
http://www.UseCaseMaps.org/tools/ucmnav/index.html

[25] Olsen, D. R., Jr., “User Interface Management Systems,” Morgan Kaufmann Publishers, Inc. 1992

[26] Cox, P. T., Giles, F. R., T. Pietrzykowski, “Prograph: a step towards liberating programming from textual conditioning,” IEEE Workshop on Visual Languages, pp.150-156, 1989

[27] J. Poswig, K. Teves, G. Brankar, C. Moraga, “VisaVis - Contributions to Practice and Theory of Highly Interactive Visual Languages,” IEEE Workshop on Visual Languages, pp.155-161, 1992

[28] 白田由佳利, 飯沢篤志, “データベーススキーマ情報からの GUI 自動生成,” 電子情報通信学会論文誌 (D-I), vol.J80-D-I, no.1, pp.71-80, 1997

[29] 北村操代, 杉本明, “生成・カスタマイズ方式による GUI 構築手法の提案とクラスライブラリ GhostHouse による実現,” 情報処理学会論文誌, Vol.36, no.4, pp.944-957, 1995

[30] J. L. Oliveira, C. B. Medeiros and M. A. Cilia: *Active Customization of GIS User Interfaces* IEEE International Conference on Data Engineering (ICDE’97) (1997)

[31] NeXT Computer, Inc, “Interface Builder Tutorial, Draft Version,” NeXT Computer, 1991

[32] S. Delmas, “XF,” ftp site at *ftp.cs.tu-berlin.de*

[33] JavaSoft, “Java Studio,” http site at
http://www.sun.com

[34] 長谷川裕行, “APPGALLERY で超簡単 Windows プログラミング入門,” ソフトバンク, 1997

[35] 青山幹雄, 内平直志, 平石邦彦, “ペトリネットの理論と実践,” 朝倉書店, 1995

[36] SRA, “Tclbombur,” http site at
http://www.sra.co.jp

[37] 宮田重明・芳賀敏彦 共著: Tcl/Tkプログラミング入門、オーム社、1995年

[38] John K. Ousterhout[著]、西中芳幸・石曽根信 [訳]、Tcl/Tkツールキット、ソフトバンク株式会社、1995年

[39] 村田忠夫, “ペトリネットの解析と応用,” 近代科学社, 1992

[40] 浅野理森 [著]: はじめてのTcl/Tk、技術評論社、1995年

[41] 大村忠史著Swingによる JavaGUIプログラミング, 株式会社カットシステム（1998）.

[42] 北村操代, 杉本明:生成・カスタマイズ方式による GUI構築手法の提案とクラスライブラリ GhostHouseによる実現, 情報処理学会論文誌, Vol36, No.4, pp.944-957 (1995).

[43] マイケル・モリソン, ジェリー・エイブラン著, 福井眞吾, 久野禎子, 久野靖訳：続・Java言語入門 新しいフレームワークと API, 株式会社プレンティスホール出版

研究業績

論文

◎ **プロトタイプのビジュアル・カスタマイズによる GUI の自動生成手法**
白銀 純子, 深澤 良彰
情報処理学会論文誌 Vol.41, No.7 (2000)

- **モデル/ビュー分離アーキテクチャBeaM の機構とその評価**
鷲崎 弘宜, 白銀 純子, 深澤 良彰
情報処理学会論文誌, Vol.42, No.10 (2001)

国際会議

◎ **Method of User-Customizable GUI Generation and Its Evaluation**
Junko Shirogane and Yoshiaki Fukazawa
Asia Pacific Software Engineering Conference(APSEC'98) (1998)

◎ **Widget-allocation oriented design method and its support tool**
Junko Shirogane and Yoshiaki Fukazawa
4th. World Conference on Integrated Design and Process Technology (IDPT'99) (1999)

- **A Light-Weight Broker for GUI Applications**
Hironori Washizaki, Junko Shirogane and Yoshiaki Fukazawa
International Workshop on Domain Oriented Systems Development (DOSD1999) in Asia Pacific Software Engineering Conference (APSEC'99) (1999)

- **Model-View Separation Architecture BeaM and Its Evaluation**
Hironori Washizaki, Junko Shirogane and Yoshiaki Fukazawa
4th. Joint Conference on Knowledge-Based Software Engineering (JCKBSE2000) (2000)

◎ **Scenario-based Extraction of GUI Control Structure and Its Prototype Generation**
Junko Shirogane and Yoshiaki Fukazawa
Software Engineering and Applications (SEA2001)

◎ **GUI Prototype Generation by Merging Use Cases**
Junko Shirogane and Yoshiaki Fukazawa
Intelligent User Interfaces (IUI2002) (2002)

◎ **A Method of Scenario-based GUI Prototype Generation**
Junko Shirogane and Yoshiaki Fukazawa
3rd ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'02) (2002)（発表予定）

査読付きシンポジウム

- **ビジュアル・カスタマイズ方式によるソフトウェアの開発手法**
  白銀 純子, 深澤 良彰
  ソフトウェア工学の基礎ワークショップ (FOSE'98) (1998)
- **MVC 指向アーキテクチャ BeaM の機構とその評価**
  鷲崎 弘宜, 白銀 純子, 深澤 良彰
  ソフトウェア工学の基礎ワークショップ (FOSE'99) (1999)
- **コンポーネント技術に基づく Undo/Redo 機能実装手法**
  鷲崎 弘宜, 白銀 純子, 深澤 良彰
  プログラミングおよび応用のシステムに関するワークショップ (SPA2000) (2000)
- **メタ情報を用いたコンポーネント指向ソフトウェア開発支援**
  鷲崎 弘宜, 白銀 純子, 深澤 良彰
  オブジェクト指向 2000 シンポジウム (2000)
- **安全なコンポーネント拡張に基づく共通機能付加技法**
  鷲崎 弘宜, 白銀 純子, 深澤 良彰
  オブジェクト指向 2000 シンポジウム (2000)
- **細粒度コンポーネント環境における履歴クラスタリング**
  鷲崎 弘宜, 白銀 純子, 深澤 良彰
  ソフトウェア工学の基礎ワークショップ (FOSE2000) (2000)

- **ユースケースの縒合せによる GUI プロトタイプ生成手法**
  白銀 純子, 深澤 良彰
  ソフトウェア工学の基礎ワークショップ (FOSE2001)

研究会

- **GUI の自動生成におけるプロトタイプ修正方式とその評価**
  白銀 純子, 深澤 良彰
  電子情報通信学会, 信学技報 SS97-33 (1997)
- **真のプラガブルアーキテクチャをめざして**
  鷲崎 弘宜, 白銀 純子, 深澤 良彰
  ウィンターワークショップ・イン・高知 (1999)
- **ソフトウェアアーキテクチャにおける再利用性と実行効率**
  鷲崎 弘宜, 白銀 純子, 深澤 良彰
  サマーワークショップ・イン・小樽 (1999)
- **仕様からの適用型ユーザナビゲーションシステムの生成**
  原田 真太郎, 白銀 純子, 深澤 良彰電子情報通信学会, 信学技法 SS2000-21 (2000)
- **ユースケース記述からの GUI 生成手法**
  白銀 純子, 深澤 良彰
  ウィンターワークショップ・イン・金沢 (2001)
- **細粒度コンポーネント環境への着目と Undo 機能追加実装**
  鷲崎 弘宜, 白銀 純子, 深澤 良彰
  ウィンターワークショップ・イン・金沢 (2001)
- **シナリオからの GUI 制御構造の把握とプロトタイプ生成**
  白銀 純子, 深澤 良彰
  電子情報通信学会, 信学技報 SS2000-60 (2001)
- **GUI 部品の変更時のプログラム自動変更支援**
  渡部 宏志, 白銀 純子, 深澤 良彰
  電子情報通信学会, 信学技報 SS2001-37 (2002)

全国大会

- **ペトリネットからの GUI の自動生成について**
  白銀 純子, 深澤 良彰
  情報処理学会第 55 回全国大会 (1997)
- **アプリケーション部品についての MVC パターンの適用**
  鷲崎 弘宜, 白銀 純子, 深澤 良彰
  情報処理学会第 58 回全国大会 (1999)
- **動的モデリングからの適応型ユーザナビゲーションシステムの開発**
  原田 真太郎, 白銀 純子, 深澤 良彰
  情報処理学会第 59 回全国大会 (1999)
- **マルチメディアアプリケーション開発のためのパターン抽出と整理**
  増田 航也, 白銀 純子, 深澤 良彰
  情報処理学会第 60 回全国大会 (2000)
- **ユースケース記述にもとづく GUI 開発手法**
  白銀 純子, 深澤 良彰
  第 61 回情報処理学会全国大会 (2000)
- **ウィジェット操作の抽象化による GUI 部品の変更支援**
  渡部 宏志, 白銀 純子, 深澤 良彰
  情報処理学会第 63 回全国大会 (2001)
- **自然言語解析技術の応用によるシナリオからのイベントフロー抽出手法**
  菊川 正人, 白銀 純子, 深澤 良彰
  電子情報通信学会 2002 年総合大会 (2002)