ファクトベースを用いたヘルプドキュメントの生成

辻 将悟[†] 山本喜一[†] 平井航一[†]

ユーザがGUIベースのアプリケーションを利用するときに，操作対象を指定し，機能を実行するための動作を行うという操作モデルに基づき定義したファクトベースと，これを利用したヘルプドキュメントの動的生成手法およびテクニカルライタが使用する制作支援環境を提案する．このアプローチを用いることによってテクニカルライタは，同様な記述を何度も繰り返さずにすむだけでなく，アプリケーションのリリース直前の集約的な作業を軽減し，アプリケーションの機能追加変更などにともなう改訂にも柔軟に対応することができる．ユーザにとっても自分のスキルレベルに応じて適量な情報を得ることができるようになる．本研究は，現実のアプリケーションに応用することを目指し，OpenOffice.orgの表計算アプリケーションであるCalcを対象として実現している．

Generation of a Help Document Based on Factbases

Shogo Tsuji,[†] Yoshikazu Yamamoto[†] and Koichi Hirai[†]

We propose a framework of dynamic generation for user's manual of GUI-based software and its authoring environment for technical writers. This framework and environment are based on our proposed fact base. The fact base is defined on the basis of the following operation model of GUI-based application. When a user uses GUI-based application, he/she basically specify the target object that he/she operates and takes an appropriate action to apply the application's function to the object. Our approach will give several advantages as follows to both of technical writers and users. (1) it is not necessary to write the same sentence repetitively. (2) it can reduce time-consuming and labor-intensive packaging task immediately before the application release. (3) it can update the documents flexibly and quickly in accordance with the change of the application's functionality. (4) the end-user can get appropriate information suited for his/her skill level. The aim of this research is to apply the result to practical application software, so that we selected Calc that is a spreadsheet application provided by OpenOffice.org as the target software.

1. はじめに

昨今，複雑な機能を持つアプリケーションが数多く使われるようになり，その一方でコンピュータのユーザは小学生から高齢者まで広がり，ユーザのスキルレベルはかつてないほど多様化している．多くのアプリケーションでは，ある作業目標を達成するための操作手順を説明したタスク指向型ドキュメント（以下，ヘルプドキュメントと呼ぶ）や，機能や用語の解説などを網羅的に記述したリファレンス形式のユーザ文書が付属されているが，その中でも前者のヘルプドキュメントは，ユーザがアプリケーションの使い方を習得するために有用であり利用頻度が高い[4]．**図1**にOpenOffice.org[5]の表計算アプリケーションであるCalcのヘルプドキュメントの例を示す．アプリケーションに付属されるヘルプドキュメントは，通常ハイパーテキスト形式で記述され，オンラインヘルプの一部として提供される．

エンドユーザにとってヘルプドキュメントは有用であるが，その一方でテクニカルライタにとってこれを制作するのは，それほど高いスキルは要求されないものの，その制作過程においてほとんどがルーチン化された面倒で時間のかかる作業である．

ヘルプドキュメントの制作にあたり，通常テクニカルライタは，アプリケーションが提供する機能を1つずつ実行しその反応を記録する，ということを繰り返し，アプリケーションの機能体系を把握することから始める．次に，ヘルプドキュメントに含める作業目標を決定し，機能体系をもとにその作業目標を達成する操作手順をまとめ実際に記述していく．同時にヘルプドキュメント全体の構成についても決定する．ある程度完成してきた段階で，内容や質の検証を行う．検証

† 慶應義塾大学大学院理工学研究科開放環境科学専攻
School of Science for Open and Environmental Systems, Graduate School of Science and Technology, Keio University

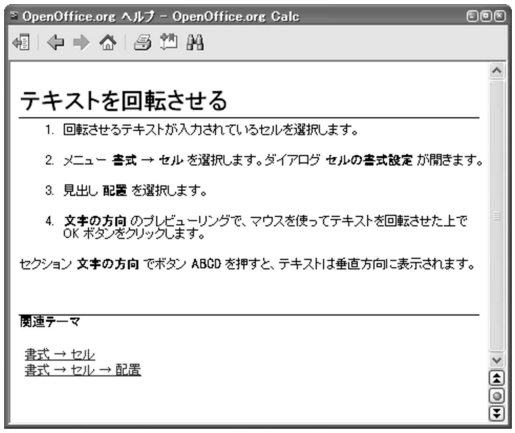

図 1 Calc に付属するオンラインヘルプの例
Fig. 1 Example of online help included in Calc.

段階では，文書スタイルや使用される用語が全体を通じて統一されているかといった細かなチェックも行われる．このような制作・検証過程が繰り返し行われ最終的なヘルプドキュメントとして完成する．必要であれば多言語に対応するために成果物を目的の言語に翻訳しなければならない．

また，テクニカルライタの作業時間の多くは，出荷後のメンテナンスに費やされているという報告もある[6]．内容に誤りがあった場合にはもちろんであるが，アプリケーションのバージョンアップによりシステム構成や機能が追加・変更された場合には，新しく制作する場合と同様の制作・検証過程を再度適用したり，あるいは成果物全体に散在する該当箇所を探し出し 1 つ 1 つ修正していかなければならない．

このような制作過程は，一般的にソフトウェア開発ライフサイクルにおいて後回しにされる傾向があり，アプリケーションが多機能化・複雑化し，日々その開発・供給サイクルの短縮化が図られる今日では，テクニカルライタの制作支援環境や，ドキュメントの動的生成技術の必要性がいっそう高まっている[11]．

本稿では，ユーザが GUI ベースのアプリケーションを利用するときに，操作対象を指定し機能を実行するための動作を行う，という操作モデルに基づいたヘルプドキュメント生成のためのアプリケーションモデル（以降，“ファクトベース” と呼ぶ）を提案する．また，本手法が現実のアプリケーションに応用可能であることを示すために，Calc を対象としてファクトベースを利用したヘルプドキュメントの生成とテクニカルライタのための制作支援環境について説明する．

2. ファクトベースおよびシナリオを用いたヘルプドキュメントの生成

ヘルプドキュメントを制作する観点から GUI アプリケーションにおけるユーザの作業を考察すると，次に示す特徴があることが分かる．

- 作業目標を達成する各手順は，Calc の “書式メニュー” のような対象と “押す” という動作の組合せによって表現できる．
- カスケードメニューが一般的になり，共通のメニュー階層をたどって末端の異なるメニューを実行する場合が多い．たとえば Calc では，“マクロの記録” や “マクロの実行” 機能を実行するために，“ツール”→“マクロ” という共通のメニュー階層をたどることになる．
- 説明する作業目標や，同じ作業目標であってもユーザのスキルレベルなどに応じて，多様な達成方法が考えられるが，それに含まれる 1 つ 1 つの手順は複数の作業目標で共通している場合が多い．たとえば Calc で，セルの書式設定や装飾について説明する場合，“書式”→“セル” というメニュー階層をたどり “セルのフォーマット” ダイアログを表示するという行為は，説明書の中に何度も登場するだろう．
- アプリケーションによらず，多くの場合キーボードやマウスを入力デバイスとして利用する．
- キーを “押す”，“放す”，マウスボタンを “押す”，“放す”，“クリックする” など，アプリケーションが変わってもユーザの基本動作は同じである．

本手法はこれらの特徴に基づき，アプリケーションに関連する “対象” と “動作” および “対象＋動作” という形式の並びで表現するアプリケーションが提供する機能をファクトとして定義し，これらを集積したファクトベースを用いる．ヘルプドキュメントは，機能ファクトの並びで作業手順を示したシナリオという記述に基づいてファクトを収集し生成することができる．

以降，ファクトおよびシナリオについて詳述する．なお，ファクトおよびシナリオ，その他関連するリソースデータの記述には XML（eXtensible Markup Language）を用いており，各 XML 記述を形式的に定めるために XML スキーマ記述言語である XML Schema を用いている．

2.1 ファクト

前記の “対象” と “動作” および “対象＋動作” という形式の並びで記述する機能を表現するために，次の (1)〜(6) のファクトを定義する．これら 6 つのファク

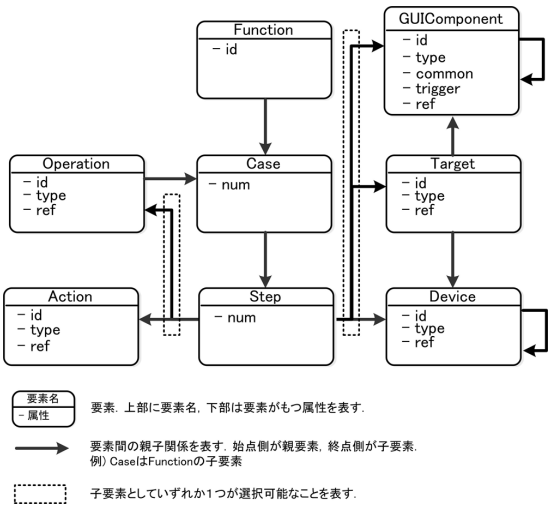

図 2 ファクトベースを構成する要素間の関係
Fig. 2 Relationship between factbase composition elements.

トと，ファクトを記述するための付加的な要素として追加した Case 要素（選択的な実行手順を記述するために用いる）および Step 要素（実行順序を記述するために用いる）を含めたファクトベース全体のスキーマは**図 2** のようになる．図 2 で，各要素を表す四角の下半分にその要素が持つ属性を記す．図中の矢印は各要素間の親子関係を表し，それぞれ始点側が親要素，終点側が子要素を表している．たとえば Case 要素は Function 要素の子要素である．また，図中の点線囲みは子要素としていずれか 1 つが選択可能なことを表している．

(1) GUIComponent（GUI 構成要素）

ウィンドウ，ダイアログボックス，メニュー，ボタンなどアプリケーションの画面を構成する要素．Calc のメニューバーのファクト記述例を次に示す．なお，各要素の id 属性と type 属性は，それぞれ GUI 構成要素の識別子と種類を表している．

```
<Component id="RootWindow_Menu"
           type="MenuBar">
 <Component id="RootWindow_Menu_File"
            type="PulldownMenu">
  <Component id="RootWindow_Menu_File_New"
             type="CascadeMenu">
   <Component
    id="RootWindow_Menu_File_New_Document"
    type="MenuItem"/>
   <Component
    id="RootWindow_Menu_File_New_Spreadsheet"
    type="MenuItem"/>
       :
  </Component>
  <Component id="RootWindow_Menu_File_Open"
             type="MenuItem"/>
       :
 </Component>
 <Component id="RootWindow_Menu_Edit"
            type="PulldownMenu">
  <Component id="RootWindow_Menu_Edit_Cut"
             type="MenuItem"/>
  <Component id="RootWindow_Menu_Edit_Copy"
              type="MenuItem"/>
       :
 </Component>
     :
</Component>
```

(2) Device（デバイス情報）

キーボード，マウスなどの入力機器を構成するキー，左ボタン，右ボタンなど．2 ボタンマウスのファクト記述例を次に示す．

```
<Device id="Mouse">
 <Device id="2ButtonMouse">
  <Device id="LeftButton"/>
  <Device id="RightButton"/>
 </Device>
</Device>
```

(3) Target（対象）

GUIComponent や Device では表現しきれない “対象” を定義する．たとえば，“セル範囲” は，次のように他のファクトを参照する ref 属性を用いて GUIComponent の 1 つである “セル” を参照し，その集合として定義する．

```
<Target id="RootWindow_CellField_Cells"
        type="CellRange">
 <Component ref="RootWindow_CellField_Cell"/>
</Target>
```

(4) Action（基本動作）

マウスボタンを “押す”，“放す”，あるキーを “押す” など，ユーザが行うそれ以上分解できない動作．ダブルクリックは “クリックする” を 2 回連続する動作として定義することができるため，基本動作には含めない．マウスボタンを “押す”，“押しながら”，“放す” という基本動作のファクト記述例を次に示す．

```
<Action id="Press"/>
<Action id="Pressing"/>
<Action id="Release"/>
```

(5) Operation（操作）

Action では表現できない動作で，Device と Action の組，GUIComponent と Action の組，複

数の Operation の組合せなどで定義する．たとえば，“ドラッグする” という Operation は，“マウスの左ボタンを押しながら”，“マウスを移動する”，“マウスの左ボタンを離す” の並びによって次のように記述できる．

```
<Operation id="Drag">
 <Case num="1">
  <Step num="1">
   <Device ref="LeftButton"/>
   <Action ref="Pressing"/>
  </Step>
  <Step num="2">
   <Device ref="Mouse"/>
   <Action ref="Move"/>
  </Step>
  <Step num="3">
   <Device ref="LeftButton"/>
   <Action ref="Release"/>
  </Step>
 </Case>
</Operation>
```

(6) Function（機能）

アプリケーションが提供する機能を “対象 + 動作” の形式の並びで定義する．各 Function は，主にアプリケーションのメニューやツールアイコンによって表現されている機能に対応する．Calc の “コメントの挿入” 機能を実行するためのファクト記述例を次に示す．

```
<Function
 id="Execute_RootWindow_Menu_Insert_Note">
 <Case num="1">
  <Step num="1">
   <Component ref="RootWindow_Menu"/>
   <Action ref="Find"/>
  </Step>
  <Step num="2">
   <Component
    ref="RootWindow_Menu_Insert"/>
   <Operation ref="LeftClick"/>
  </Step>
  <Step num="3">
   <Component
    ref="RootWindow_Menu_Insert_Note"/>
   <Operation ref="LeftClick"/>
  </Step>
 </Case>
</Function>
```

(1)〜(6) のファクトのうち，Device，Action，Operation については，アプリケーションに依存せず，一度定義しておけばそのほとんどを再利用することができる．GUIComponent および Function については，アプリケーションの GUI 構成仕様から自動生成することが可能であり，アプリケーションによって追加・削除があるものの，その多くを再利用することができる．ファクトベースの自動生成について，詳しくは 3.1 節を参照されたい．

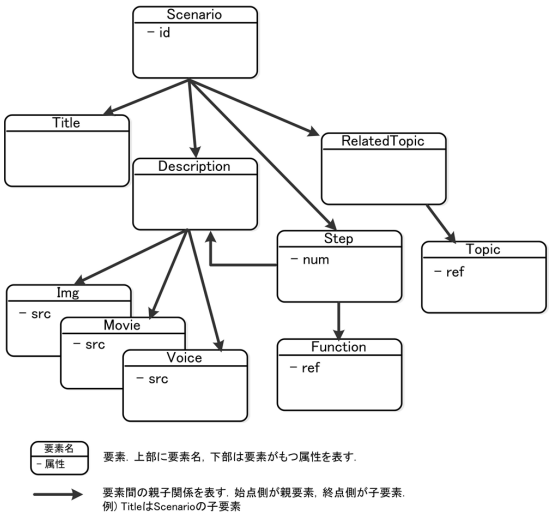

図 3 シナリオを構成する要素間の関係
Fig. 3 Relationship between scenario composition elements.

2.2 シナリオ

シナリオは部品化されたファクトを収集してヘルプドキュメントを生成するために用いる．従来のように，低レベルな手順（たとえば，“書式メニューをクリックする” など）を 1 つずつ記述しなくても，アプリケーションの機能に相当する，より高レベルな手順に沿って Function を並べてシナリオを記述するだけで，具体的な手順が自動的に得られるので，ユーザのスキルレベルや理解度に合わせて，さまざまな課題設定や手順のヘルプドキュメントを効率的に作成することができる．

シナリオを構成する要素の関係を **図 3** に示す．図 3 で，各要素を表す四角や矢印の意味は図 2 と同様である．シナリオは基本的には Function の並びになるが，タイトル（Title），補足説明（Description），画像（Img），動画（Movie），音声（Voice），関連リンク（RelatedTopic および Topic）も記述することができる．これらは自動的に生成されるものではないが，ユーザの理解を助けるために補助的に付加する情報である．

次にシナリオの具体例を示す．

```
<Scenario id="InsertingNotes">
 <Title>コメントの挿入</Title>
 <Description>
```

```
  すべてのセルはコメントを挿入できます.
 </Description>
 <Step num="1">
  <Function ref="SelectCell"/>
  <Description>
   <Img src="InsertingNotes_image0.png"/>
  </Description>
 </Step>
 <Step num="2">
  <Function
   ref="Execute_RootWindow_Menu_Insert_Note"/>
  <Description>
   <Img src="InsertingNotes_image1.png"/>
  </Description>
 </Step>
 <Description>
  コメントは，詳細ヒントを有効にしている場合，
  コメント付きのセルをマウスポインタで指し示すと
  表示されます.
 </Description>
 <RelatedTopic>
  <Topic ref="ShowNotes">
   コメントの表示方法
  </topic>
 </RelatedTopic>
</Scenario>
```

これは Calc で “コメントの挿入” 方法を説明するシナリオである．“セルの選択” 方法や “コメント” メニューの場所を知っているユーザにとっては，上記の Function レベルの手順（つまり，“セルを選択” し “コメントメニューをクリック” する）が分かれば十分である．一方，“コメント” メニューの場所を知らない場合には，Function の ref 属性に指定された ID で参照する Function をファクトベースから取得し，これを “対象 + 動作” の形式の並びに展開することで，より詳細な手順を得ることができる．

また，Topic 要素の ref 属性に，関連するシナリオの ID を指定することによって，一般的なヘルプドキュメントにもあるような “関連項目” をユーザに示すことができる．

2.3 ヘルプドキュメントの生成

シナリオに基づくファクトの収集からヘルプドキュメントの生成に至る流れを **図 4** に示す．最終的に得られるヘルプドキュメントは現在の実装では **図 5** のようになるが，これが唯一の可能な生成物ではない．収集するファクトのレベルを要求に応じて動的に変えることによって，ユーザのスキルレベルや理解度に応じて説明の詳細度をあらかじめ調整した文書を生成することができるようにもなる．たとえば，上級者向けであれば簡潔な箇条書きの説明だけを生成し，初心者の場合には詳細な手順を図を含めた文書として生成する．

また，用途や閲覧するアプリケーション，表示デバイスに応じて，HTML や PDF など柔軟に出力形式を選択することもできる．さらに，複数の言語でシナリオおよびファクトに対応する言語リソースを定義しておくことによって，多言語に対応したヘルプドキュメントを効率的に作成することもできる．

図 4 に沿って具体的な処理の流れを次に示す．処理は大きくファクトの収集と文書生成の 2 つの工程からなる．前工程では，まずシナリオを先頭から順に解析し，Function が見つかった場合に，その ref 属性が参照するファクトを取得する．図 4 の例では，“Execute_RootWindow_Menu_Insert_Note” という ID を持つ Function をファクトベースから取得している．このとき必要であれば，この Function に対応する適切な言語（日本語，英語など）で記述された文書素片を言語リソースから取得する．

より詳細な説明が必要な場合は，先に取得した Function に含まれる Step 要素の解析を継続する．この Step 要素は 2.1 節 (6) で述べたように “対象＋動作” の形式で記述されているので，それぞれ記述されているファクトと同一の型で ref 属性が参照するファクトを取得する．たとえば図 4 では，`<Component ref="RootWindow_Menu"/>` は ID が “RootWindow_Menu” である GUIComponent を指し，`<Action ref="Find"/>` は ID が “Find” の Action を指している．

文書素片の取得は，先の Function の場合と同様にして行われるが，ここでは “対象”，“動作” に対応する文書素片を別々に得ただけで，これらをもとに適切な操作指示文を生成しなければならない．

図 4 の “テキスト生成モジュール” は，定義された文生成規則に基づき，“対象” と “動作” を表すファクトから操作指示文を生成する．最も単純な文生成規則は，“対象” と “動作” の間に助詞の “を” を補うことで，4 章の実装例でもこの規則を適用している．

シナリオの解析とファクトの収集が完了した結果，前工程の出力として，表示スタイルや構造など文書体裁に関連する情報を含まないコンテンツ情報を，XML 形式の中間生成物として得る．続く後工程では，この中間生成物に適切な文書体裁情報を適用し，最終的なヘルプドキュメントを生成する．コンテンツ情報と文書体裁情報を分離することによって，前工程を再度実行することなく，1 つの中間生成物から多様な体裁の文書を生成できるという利点がある[3]．

2.4 関 連 研 究

ヘルプ文を属性に持つ操作ステップノードからなるタスクモデルに基づいてヘルプドキュメントを生成

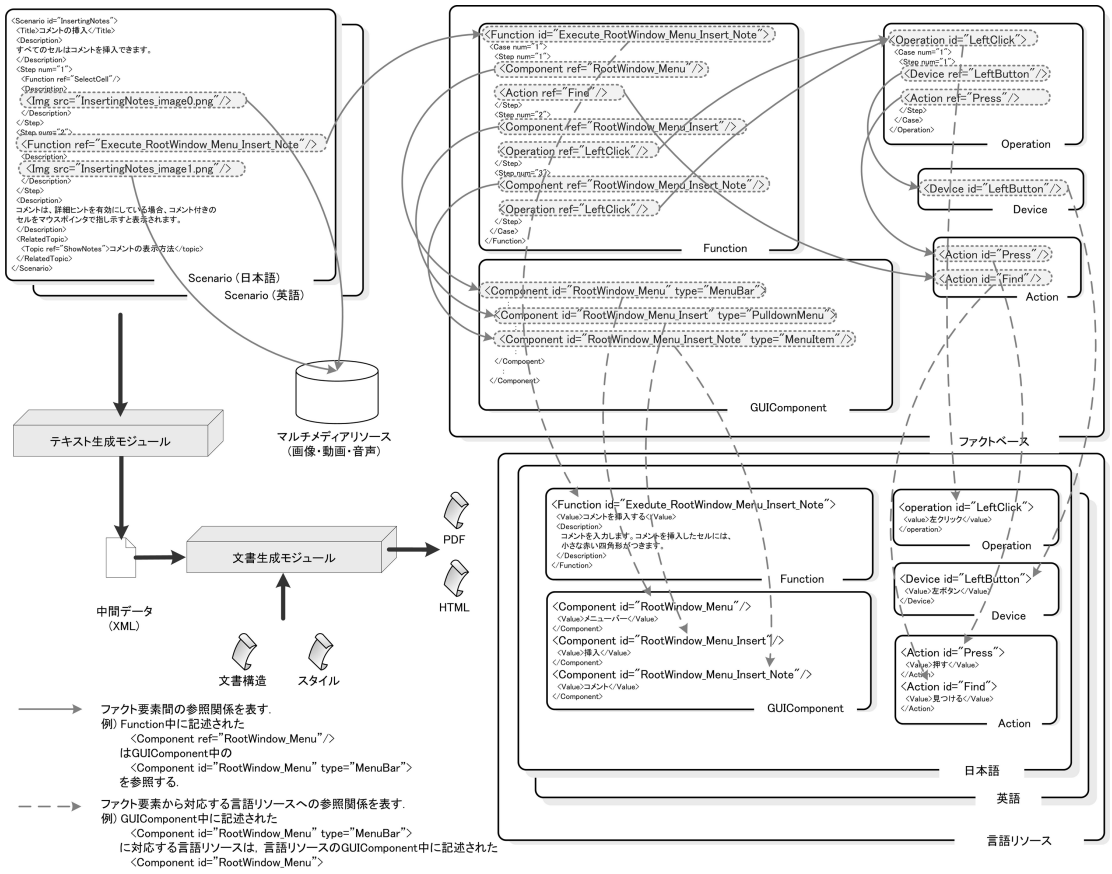

図 4 ヘルプドキュメント生成の流れ
Fig. 4 Process of document generation.

図 5 ヘルプドキュメントの生成例
Fig. 5 Example of generated document.

する手法は，これまでも多く提案されている．本手法では，要求に応じてタスクモデルを解析しヘルプドキュメントの構成を決定し，これに沿って操作ステップノードに対応するヘルプ文を並べていくことでヘルプドキュメントを生成する．

Paris らの開発した **ISOLDE** システム[6] や **DRAFTER**[7] システムでは，ヘルプドキュメントの再利用性やメンテナンス性を高めるために，操作ステップノードに対応するヘルプ文をさらに粒度の細かい文書素片に分割して管理することができる．この文書素片は，実行主体（ユーザやシステムなど）と動作（開く，表示する，クリックするなど）と対象（ファイル，カーソル，ボタンなど）の 3 種類に分類され，操作ステップノードに対応するヘルプ文を生成するために利用される．文よりも粒度の細かい文書素片からヘルプドキュメントを生成するという点においては本稿で提案する手法と同様であるが，ヘルプ文をもとにして何を文書素片として切り出すかはテクニカルライタの判断に委ねられている．

頻繁に利用する一連の操作ステップをグループ化して管理することも，再利用性やメンテナンス性の向上に役立つ．**ISOLDE** システムや **DRAFTER** システムをはじめとしてすでに一般的な方法になっている．グループ化についてはこれ以外にも，“上位の目的による個々の操作の意味付けを行いヘルプドキュメントの理解度を上げる[10]”という目的もあり，**epiplex**TM,9) などの商用ツールではこのために操作ステップのグループ化をサポートしている．

以上の従来手法で問題なのは，文書素片の切り出しにしてもグループ化にしても，操作ステップを基点としてテクニカルライタが自らの判断で行わなくてはならない点である．無駄を少なく再利用性やメンテナンス性を高めるためには，事前に綿密な計画が必要であると考えられるが，その判断はテクニカルライタの経験やスキルに大きく依存するものと思われ，またその作業負担も大きい．

これに対して本稿で提案する手法では，ヘルプドキュメント生成に必要なアプリケーションモデルを 2.1 節に示した 6 つのファクト要素で規定してしまうことによって，3.1 節で示すようにファクトベースを機械的に構築することができるようになるため，テクニカルライタを煩わせることがなくなる．したがって，テクニカルライタは従来手法のように機能を実行するための詳細な手順を気にする必要がなく，シナリオを基点としてトップダウンにヘルプドキュメントを制作することが可能になると考えている．

3. 編集環境

本稿で提案する枠組みを実際のアプリケーション開発に応用するには，ファクトベースや言語リソース，シナリオを効率的に作成する必要がある．XML Schema で形式が規定されているので，市販の XML 編集アプリケーションなどを利用して必要な XML データを手作業で作成することはできるが，それには XML に関するある程度の知識が必要になり作成者の負担は大きい．

本章では，1 つの解決方法として CSV（Comma Separated Values）形式で記述された単純な構造の元データから，ファクトベースと言語リソースを生成する方法，およびアプリケーションの実際の操作記録からシナリオの骨組みを生成し，必要な編集作業を行うシナリオ制作・編集アプリケーションを紹介する．

3.1 ファクトベースおよび言語リソースの作成

ファクトベースや言語リソースの作成者は，必ずしも XML に関する知識を必要としない．つまり，XML

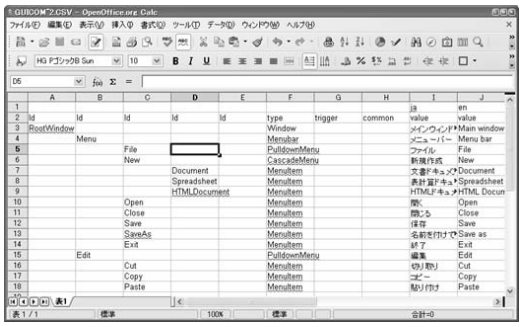

図 6 メニューバー GUIComponent を生成する CSV データ
Fig. 6 CSV data transformed into menubar GUIComponent elements.

Schema で規定されている作成者が最低限記述しなければならない要素値と属性値を，目的の XML 構造に変換可能な構造で表現することができればよい．一例として，カンマ区切りの単純な構造を持つ CSV 形式で記述した元データから，ファクトベースおよび言語リソースを生成する方法を紹介する．CSV 形式は企業や大学では広く知られたデータ形式で，市販の表計算アプリケーションでもそのまま扱うことができる．

図 6 の Calc のメニューバーの GUI 構成を階層構造で表現した CSV データから，2.1 節 (1) の例で示した GUIComponent と，対応する言語リソースを生成することができる．各 GUIComponent の ID は，親要素の ID に自身を表す名前を連結したものになるので，その要素が位置する階層の深さによって一意に決定することができる．

同様にしてダイアログについても CSV データから生成することができる．ただしこの場合は，GUIComponent の trigger 属性に，このダイアログを出現させるための Function ID を指定しておく．この trigger 属性は後述する Function の自動生成に利用する．GUIComponent のほか，Device，Target，Action，Operation についても，同様の方法でファクトを生成できることを確認している．

Function は，次に説明するようにその多くを自動生成することができる．アプリケーションが提供する機能のほとんどは，先に生成した GUIComponent の末端ノードが表す GUI 部品に関連付けられている．すなわち当該 GUIComponent の最上位の祖先ノードから，末端ノードまでを順にたどることにより，関連付けられた機能の実行手順を知ることができる．たとえば，“セルの書式設定” ダイアログを出現させるには，先のメニューバーを表現した ID が “RootWindow” である GUIComponent から，ID が “RootWindow_Menu_Format_Cells” の末端ノードまでを順にた

どっていけばよい．

また，各 GUIComponent に対する動作は，その type 属性に基づいて決定することができる．たとえば，type 属性が “pulldown” である GUIComponent に対しての動作は，Operation ID が “LeftClick” の “左クリック” となる．結果として，Calc の “マクロの記録” 機能を実行するための Function は，次のように生成される．

```
<Function
 id="Execute_RootWindow_Menu_Tools_Macros_Record">
 <Case num="1">
  <Step num="1">
   <Component ref="RootWindow_Menu"/>
   <Action ref="Find"/>
  </Step>
  <Step num="2">
   <Component
    ref="RootWindow_Menu_Tools"/>
   <Operation ref="LeftClick"/>
  </Step>
  <Step num="3">
   <Component
    ref="RootWindow_Menu_Tools_Macros"/>
   <Action ref="Wait"/>
  </Step>
  <Step num="4">
   <Component
    ref="RootWindow_Menu_Tools_Macros_Record"/>
   <Operation ref="LeftClick"/>
  </Step>
 </Case>
</Function>
```

ダイアログで提供される機能については，対応する GUIComponent のルートノードに trigger 属性を指定することにより，そのダイアログを出現させるまでの手順を含めた Function を生成することができる．Calc の “セルの書式設定” ダイアログの “配置” タブにある “ABCDCircle” で文字方向を設定する Function の例を次に示す．

```
<Function
 id="Execute_AlignmentRotation">
 <Case num="1">
  <Step num="1">
   <Component ref="RootWindow_Menu"/>
   <Action ref="Find"/>
  </Step>
  <Step num="2">
   <Component
    ref="RootWindow_Menu_Format"/>
   <Operation ref="LeftClick"/>
  </Step>
  <Step num="3">
   <Component
    ref="RootWindow_Menu_Format_Cells"/>
   <Operation ref="LeftClick"/>
  </Step>
  <Step num="4">
   <Component
    ref="Cells_Dialog"/>
   <Action ref="Wait"/>
  </Step>
  <Step num="5">
   <Component
    ref="Cells_Dialog_Alignment"/>
   <Operation ref="LeftClick"/>
  </Step>
  <Step num="6">
   <Component
    ref="Cells_Dialog_Alignment_TextDirection"/>
   <Action ref="Find"/>
  </Step>
  <Step num="7">
   <Component
    ref="Cells_Dialog_Alignment_
         TextDirection_ABCDCircle"/>
   <Operation ref="LeftClick"/>
  </Step>
 </Case>
</Function>
```

この Function は，“セルの書式設定” ダイアログの GUIComponent のルートノードに，trigger 属性として “Execute_RootWindow_Menu_Format_Cells” を指定することによって得られる．

GUIComponent の common 属性は，ある機能を実行する手順が複数通りある場合に，それらを 1 つの Function に集約するために記述する．Calc の “行の挿入” 機能を表現した Function の例を次に示す．

```
<Function
 id="Execute_InsertRows">
 <Case num="1">
  <Step num="1">
   <Component ref="RootWindow_Menu"/>
   <Action ref="Find"/>
  </Step>
  <Step num="2">
   <Component
    ref="RootWindow_Menu_Insert"/>
   <Operation ref="LeftClick"/>
  </Step>
  <Step num="3">
   <Component
    ref="RootWindow_Menu_Insert_Rows"/>
   <Operation ref="LeftClick"/>
  </Step>
 </Case>
 <Case num="2">
  <Step num="1">
   <Component ref="RootWindow_MainToolbar"/>
   <Action ref="Find"/>
  </Step>
  <Step num="2">
```

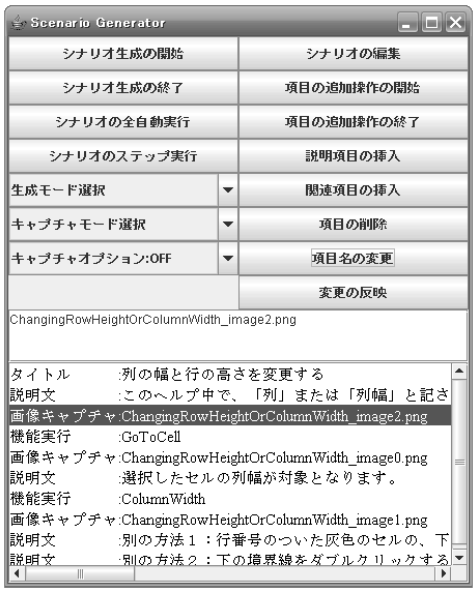

図 7 Calc 用シナリオジェネレータ
Fig. 7 Scenario Generator for Calc.

```
   <Component
    ref="RootWindow_MainToolbar_InsertRow"/>
   <Operation ref="LeftClick"/>
  </Step>
 </Case>
</Function>
```

この機能の実行方法は，Calc のメニューバーからメニュー階層をたどる方法と，ツールバーのアイコンメニューから直接実行する方法の 2 通りが存在するので，それぞれの GUIComponent の common 属性に “Execute_InsertRows” を記述することによって，1 つの Function に集約することができる．

3.2 シナリオの作成

シナリオ作成者に対して，XML に関する知識やファクトベースの内容を意識させることは，負担が大きく好ましくない．筆者らが Calc 向けに開発した**図 7** の “シナリオ制作・編集アプリケーション”（以降，シナリオジェネレータと呼ぶ）は，対象アプリケーションの実際の操作記録から，シナリオの骨組みとなる Function の列を生成することができるので，シナリオ作成者の負担を軽減することができる．**図 8** に，シナリオ生成に至る流れを示す．

シナリオジェネレータは主に次の機能を提供する．

(1) シナリオの作成

図 7 の “シナリオ生成の開始” ボタンを押すと，Calc 上で行われる操作の記録が開始される．シナリオ作成者が，作成したいシナリオに沿って操作を実行し，必要ごとに説明文や画面例としてスクリーンショットを追加し，最後に “シナリオ生成の終了” ボタンを押すと，シナリオを記述した XML データが生成される．

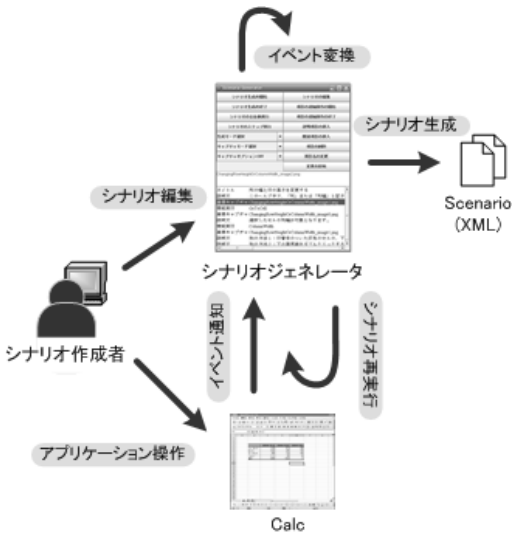

図 8 シナリオ生成の流れ
Fig. 8 Process of scenario generation.

(2) シナリオの確認

図 7 の “シナリオの編集” ボタンを押すと，選択したシナリオデータの内容が図 7 の下部に表示される．また，“シナリオのステップ実行” ボタンを押すと，シナリオに記述された手順を 1 ステップずつ実行できるので，シナリオの妥当性を繰り返し検証することができる．

(3) シナリオの編集

図 7 の “項目の追加操作の開始”，“項目の追加操作の終了”，“説明項目の挿入”，“関連項目の挿入”，“項目名の変更”，“項目の削除” を利用して，何度でも手順の変更や説明文の修正などを行うことができる．

(4) 多言語対応

一度作成したシナリオを他の言語に翻訳する場合には，対象のシナリオを選択し，図 7 の “生成モード選択” で作成したい言語を選択し，“シナリオの全自動実行” ボタンを押す．これにより，シナリオが自動的に再実行され，翻訳が必要な箇所ではシステムから入力を促されるので，実際の手順を確認しながら説明文やスクリーンショットなど，他の言語に翻訳する必要がある項目を編集することができる．

4. 実装例：オンラインヘルプの動的生成システム

本手法に基づくオンラインドキュメンテーション環境の実現例として，**図 9** に示す Calc 用のオンラインヘルプシステムを実装した．サーバアプリケーションは Java Servlet と JSP（Java Server Pages）を用いて開発した．

ユーザからのシナリオおよび表示言語の要求に応じて，サーバアプリケーションが 2.3 節の処理を実行し，**図 10** のようなヘルプ文書をユーザの Web ブラウザに返す．2.3 節および図 4 で説明した中間生成物の XML データから最終的な HTML への変換には XSL（eXtensible Stylesheet Language）を用いた．図 10 の十字アイコンを押すと折り畳まれた内容が展開されるので，ユーザがより詳細な手順を知ることができる．また，説明文の右端に配置されたアイコンを押すと，該当する手順のスクリーンショットがポップアップウィンドウに表示される．

別の実装例として，ユーザのスキルレベルを初心者，中級者，上級者の 3 段階に分け，ユーザが申告したレベルに応じた詳細度のヘルプを表示するオンラインヘルプシステムも開発した[8]．本システムの場合には，ユーザのスキルレベルに対応する内容だけが生成される．たとえば上級者であれば箇条書きの簡潔な文章だけを生成し，初心者の場合には図を含めた詳細な手順までを生成する．

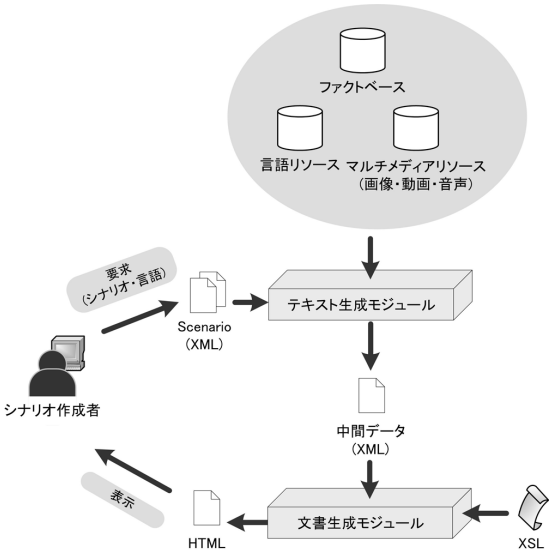

図 9 オンラインヘルプシステム
Fig. 9 Online help system.

図 10 ヘルプ文書例
Fig. 10 Example of help document.

5. 評価および議論

本章では，4 章のオンラインヘルプシステムについての結果と，それに対する評価を示すとともに，本手法の利点やテクニカルライタの従来の作業手順に与える影響について議論する．

5.1 本手法の有効性

本手法の有効性を示すために次のような実験を行った．なお，実験に利用したファクトベースに含まれる各要素数を**表 1** に示す．このうち最も要素数が多い GUIComponent は，3.1 節で述べたように Calc の GUI 構成仕様をもとに作成した CSV データから生成した．また Function についても，同節で述べたとおりそれ以外のファクトから自動的に生成したものである．

手作業で制作されている Calc のヘルプドキュメントから，**図 11** に示すように文書構成が “タイトル”，“概要”，“手順”，“補足” となっているものを 7 つ選び出し，それらの “手順” 部分に記述されたステップ数と，それ以外の付加的な項目（“タイトル”，“概要”，“補足”）の数を**表 2** に示す．また，4 章のオンラインヘルプシステムで同じ 7 つのシナリオを作成し，Function レベルのステップ数とこれを展開して得られる詳細なステップ数，および付加的な項目数を**表 3** に示す．

Calc のヘルプドキュメントの場合（表 2）は，すべ

表 1 ファクトベースに含まれる要素数
Table 1 The number of elements contained in factbase.

ファクト	個数
GUIComponent	261
Device	12
Target	2
Action	10
Operation	10
Function	122

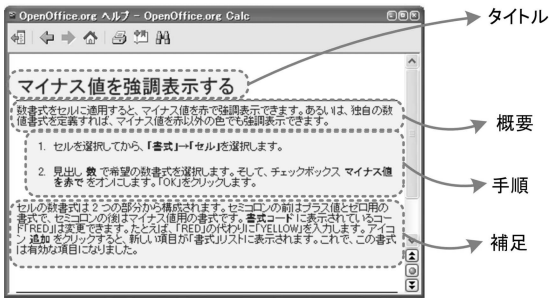

図 11 ヘルプドキュメントの文書構成例
Fig. 11 Example of document structure of a help document.

表 2 Calc のヘルプドキュメントに含まれる項目数
Table 2 Number of items described in help documents of Calc.

ヘルプドキュメント	ステップ数	付加的項目数	合計
テキストを回転させる	4	2	6
テキストを複数行にする	2	2	4
マイナス値を強調表示する	2	3	5
列の幅と行の高さを変更する	2	3	5
表のオートフォーマットを使う	4	3	7
表のテーマを選択する	3	2	5
表の回転（入れ替え）	5	3	8

表 3 生成されたヘルプドキュメントに含まれる項目数
Table 3 Number of items described in generated help documents.

ヘルプドキュメント	Function レベルのステップ数	詳細ステップ数	付加的項目数	合計
テキストを回転させる	2	12	2	16
テキストを複数行にする	2	12	2	16
マイナス値を強調表示する	2	12	3	17
列の幅と行の高さを変更する	2	6	3	11
表のオートフォーマットを使う	2	8	3	13
表のテーマを選択する	1	2	2	5
表の回転（入れ替え）	4	13	3	20

ての項目を手作業で入力するので，たとえば “テキストを回転させる” では 6 項目の入力作業に対して 6 項目分の情報量が得られると考える．一方，本システム（表 3）では，ファクトベースが事前に構築されていることを前提とするが，Function レベルの手順はシナリオジェネレータによって生成され，また詳細な手順に関してはシステムによって自動生成されるので，手作業による入力は付加的な項目だけと考えられる．したがって，“テキストを回転させる” では 2 項目の入力作業に対して 16 項目分の情報量が得られると考える．

以上から，“テキストを回転させる” については，

表 4 Calc と生成されたヘルプドキュメントとの比較
Table 4 Comparison between Calc and generated help document.

ヘルプドキュメント	入力作業項目数の割合	ヘルプドキュメントに含まれる項目数の割合
テキストを回転させる	0.3	2.7
テキストを複数行にする	0.5	4.0
マイナス値を強調表示する	0.6	3.4
列の幅と行の高さを変更する	0.6	2.2
表のオートフォーマットを使う	0.4	1.9
表のテーマを選択する	0.4	1.0
表の回転（入れ替え）	0.4	2.5

Calc のヘルプドキュメントに対して本システムでは約 0.3 倍の入力作業によって，2.7 倍の情報量を得ることができると考えられる．他についても同様にまとめたものを**表 4** に示す．

表 4 の結果から次のことが分かる．対象アプリケーションの GUI 構成において深い階層に配置されている GUI 部品を対象として動作を適用する手順を含む場合（“テキストを回転させる”，“テキストを複数行にする”，“マイナス値を強調表示する” の場合）には，本手法が効果的に働き，少ない入力作業で多くの情報量（項目数）を提供することができる．また，単純に Function レベルの手順が多ければ多いほど（“表の回転（入れ替え）” の場合），ファクトベースから生成される詳細手順も多くなるので，本手法が効果的に働く．また，5.3 節で説明するヘルプ文の結合によってファクトベースから生成される詳細手順の数が仮に半減した場合でも，表 4 においてヘルプドキュメントに含まれる項目数の割合は最大でも 2.0 倍になってしまうが，入力・修正作業における本手法の効果が損なわれることはない．

次に，一般的な手法のヘルプドキュメント生成システムとの比較について考察する．比較対象として **epiplex**^{TM} をとりあげる．**epiplex**^{TM} は，ユーザの実際の操作イベント列を記録し，それをもとに編集を加えヘルプドキュメントを制作するツールであるが，このとき記録される操作イベント列は，本提案手法における Function レベルの手順を展開した詳細手順のレベルと同等である．したがって，仮に本システムと同じ内容のヘルプドキュメントを制作する場合には，単純に考えると，本システムにおいて当該ヘルプドキュメントの元となっているシナリオ記述に含まれる Function の個数分のグループ化にかかる作業量が余計に必要になると考えられる．ただし，このとき **epiplex**^{TM} による操作イベント列の記録と，シナリオジェネレータによる Function 列の記録にかかる作業

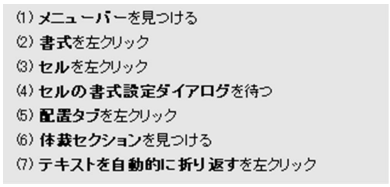

図 12 生成文書の質の問題（図 10 より抜粋）
Fig. 12 Quality problem of generated document.

量が同程度であり，その他付加的な情報の記述作業量も同程度であるとする．以上のことから，**epiplex**^{TM} のように操作イベントを基点にして，それらをボトムアップに組み上げてヘルプドキュメントを制作する手法の場合には，当該ヘルプドキュメントに含まれる Function レベルの手順が多ければ多いほど制作に必要な作業量が多くなると考えられる．

5.2 修正の手間

テクニカルライタの作業時間の多くが修正作業に費やされていることは 1 章で述べた．本手法を用いた場合，GUIComponent に対応する GUI 部品の名称変更や，Function レベルの手順の修正であれば，該当するファクトあるいは言語リソースを 1 カ所修正するだけで，それを直接的にあるいは間接的に参照しているシナリオに基づいて生成されたヘルプドキュメント以外に影響を及ぼすことなく変更が反映される．シナリオレベルの場合には，該当箇所をそれぞれ修正する必要があるが，ヘルプドキュメントはシナリオに基づいて生成されておりシナリオの修正によるヘルプドキュメントの整合性は保証されているので，シナリオジェネレータで新たな Function レベルの手順を記録するだけでよく，詳細な操作手順はファクトベースから生成されるので従来手作業の場合と比べて少ない作業負担で修正することができる．

5.3 生成文書の質

自然言語生成の視点からすると，本稿で採用した文生成規則では，人間が理解する文章として十分な質とはいえない．

たとえば，**図 12** の手順 (1)〜(7) について，手順 (1)(4)(6) はユーザにアプリケーションの操作を要求するものではないので，冗長あるいは意味のないものと感じられてしまうだろう．また，ユーザの操作に対するアプリケーションの挙動が示されていないことが，特に初心者には “何が起こるか分からない” という心的負担を与える可能性がある．

前者の問題については，図 12 の手順 (1)(2), (4)(5), (6)(7) では，対応する GUIComponent を参照するとそれぞれの “対象” どうしが GUI 構成上階層関係になっていることが分かるので，これを利用して “メニューバーの 書式を左クリック”, “セルの書式設定ダイアログの 配置タブを左クリック”, “体裁セクションの テキストを自動的に折り返すを左クリック”，のように手順をまとめて生成する方法が考えられる．後者の問題については，ユーザの動作に対するアプリケーションの挙動を，やはり “対象＋動作” の形式で記述できるように改良することで，たとえば図 12 の手順 (3)(4) について “セルを左クリックするとセルの書式設定ダイアログが表示される” のように生成することができると考えている．

このように比較的容易な改良でも，ユーザにとって分かりやすい表現にすることができるが，Paris らの開発した **ISOLDE** システム[6] や **DRAFTER**[7] システムのように，自然言語生成分野の研究成果を積極的に採用すればより自然な表現のヘルプドキュメントを生成することも可能ではないかと考えている．

しかしながら，ソフトウェアのヘルプドキュメントということであれば，文書は短く簡潔なものほど学習に効果的であるとの考え方が主流であり[1]，また表現や文法の不完全さと学習効果の相関はあまりないという報告[2] もあることから，本研究の現在の実装レベルであっても，問題のない範囲であると考えている．これについては，今後多くの実験を通じて評価していかなければならない．

5.4 テクニカルライタの作業手順に与える影響

本手法を採用することによる 1 章で述べたテクニカルライタの一般的な作業手順に与える影響は，次のようになると考えている．

(1) 作業目標の決定

いかにエンドユーザのスキル向上に役立つ課題設定をするかは，テクニカルライタが本来の才能やスキルを発揮すべき仕事であり，この作業については本手法を採用した場合も変わらず行う必要がある．ただし，従来手法のように機能の実行方法に関する細かな手順の記述や，用語の統一性，誤字・脱字，文法の誤りなど軽微だが修正の手間がかかる作業が必要ない分，本作業に集中することができるようになると考えている．

(2) 操作手順の記述

Function レベルの手順については，シナリオの作成にともないテクニカルライタが作成しなければならないが，それよりも詳細な操作手順についてはファクトベースから生成されるので考慮する必要はなくなる．また **epiplex**^{TM} など，実際に記録した操作をもとにヘルプドキュメントを制

作するツールでは，この段階で記録した操作の整理（必要のない操作の削除や，操作列のグループ化）をする必要があるが，シナリオジェネレータを利用した場合には，直接シナリオの骨組みとなる Function の列が得られるので，先のような手間のかかる作業をする必要がなくなる．

(3) 全体構成の決定

たとえば，課題を難易度別やテーマ別にカテゴリ分けするといったように，ヘルプドキュメントの理解度に影響を与えるような全体構成の決定については，従来どおりテクニカルライタが作業する必要がある．

(4) 質の検証

課題の設定や Function レベルの手順の選定など理解度により影響を与えるような教育的な質の検証については，従来どおりテクニカルライタが行わなければならない．しかしながら，ファクトベースから生成される操作手順に限定されるが，その部分の用語の統一性，誤字・脱字，文法の誤りなどを気にする必要はなくなる．

(5) 制作段階における (1)〜(4) の繰返しと出荷後のメンテナンス

この段階においては，基本的には繰返しのたびに (1)〜(4) で述べた作業分の負担軽減が可能になるが，1 章でも言及したとおり，テクニカルライタの作業時間の多くが出荷後のメンテナンスに費やされていることから，この段階での作業負担の軽減はより重要であると考えている．

6. ま と め

本稿では，ユーザが GUI ベースのアプリケーションを利用するときに，操作対象を指定し，機能を実行するための動作を行うという操作モデルに基づき定義したファクトベースと，これを利用したヘルプドキュメントの動的生成手法およびテクニカルライタが使用する制作支援環境を提案した．今後アプリケーションの多機能化・複雑化がますます進み，その開発・供給サイクルが短縮化され，さらにエンドユーザも多様化していく中で，提案した枠組みが，エンドユーザ，テクニカルライタの双方にとって大きな利点があることを示した．

多様なアプリケーションにわたってファクトベースをさらに充実させ，本手法の適応可能性について今後十分な検証を行っていく．

謝辞 本研究は，経済産業省の委託を受け，財団法人日本規格協会情報技術標準化研究センターにおいて組織されたソフトウェアの開発・生産技術の標準化調査研究オンラインドキュメンテーションワーキンググループで，国際標準化を目指し進めている調査研究活動の成果をもとに行われたものである．本調査研究で深い知見に基づく示唆に富んだ意見をいただいた企業関係者の皆様，委員会メンバ，オブザーバ諸氏に深く感謝する．

参 考 文 献

1) Carroll, J.M. (Ed.): *The Nurnberg funnel: designing minimalist instruction for practical computer skill*, The MIT Press, Cambridge, MA 02142 (1990).
2) Colineau, N., Paris, C. and Vander Linden, K.: An Evaluation of Procedural Instructional Text, *Proc. International Natural Language Generation Conference*, pp.128–135 (2002).
3) Guerrieri, E.: Software Document Reuse with XML, *Proc. International Conference on Software Reuse*, pp.246–254 (1998).
4) Knabe, K.: Apple guide: A case study in user-aided design of online help, *CHI '95: Conference companion on Human factors in computing systems*, New York, NY, USA, pp.286–287, ACM Press (1995).
5) OpenOffice.org. http://www.openoffice.org/
6) Paris, C., Colineau, N. and Lu, S.: Automatically Generating Effective Online Help, *International Journal on E-Learning, Special Issue on Technologies for electronic documents for supporting E-Learning*, pp.83–103 (2005).
7) Paris, C. and Vander Linden, K.: DRAFTER: An Interactive Support Tool for Writing Multilingual Instructions, *IEEE Computer*, Vol.29, No.7, pp.49–56 (1996).
8) 品川一貴，平井航一，山本喜一：ユーザ文書のオンライン生成，情報システム学会第 1 回研究発表大会論文集 (2005).
9) サイバネットシステム株式会社：epiplex™. http://www.cybernet.co.jp/epiplex/
10) 松本文隆：ユーザマニュアルにおける理解指向と操作指向，技術報告，情報メディア (1994).
11) 財団法人 日本規格協会情報技術標準化研究センター：平成 17 年度ソフトウェア製品の品質改善の標準化に関する調査研究成果報告書 (2005).

(平成 18 年 11 月 30 日受付)
(平成 19 年 5 月 9 日採録)

辻　　将悟（正会員）

平成 18 年慶應義塾大学大学院理工学研究科後期博士課程修了．現在，同大学大学院共同研究員．知的ヘルプシステムに関する研究を行っている．日本ソフトウェア科学会，ヒューマンインタフェース学会各会員．

山本　喜一（正会員）

慶應義塾大学工学部管理工学科卒業．管理工学専攻修士，工学博士．現在，同大学理工学部情報工学科教授．知的ヘルプシステム，オンラインドキュメンテーション等インタフェースに関する研究を行っている．IEEE，ACM，ヒューマンインタフェース学会，日本ソフトウェア科学会，情報システム学会各会員．情報システム学会理事．

平井　航一

平成 17 年慶應義塾大学理工学部情報工学科卒業．現在，同大学大学院理工学研究科修士課程在学中．オンラインドキュメンテーションの研究に従事．