WikiBOK における編集競合問題とその解決について

Editing Systems with Conflict Detector for WikiBOK

矢吹　太朗

概　要

木構造で表現される知識体系（Body of Knowledge, BOK）を集団で構築するためのシステムについて検討する．このシステムは，直感的な操作性と版管理機能，競合検出機能を持たなければならない．直感的な操作性は，木構造をファイルシステムのフォルダ階層で表現し，ファイルシステムのための GUI を利用することで実現できる．しかし，ファイルシステムを集団で利用するための共有フォルダやバージョン管理システム（Subversion）は，版管理及び競合検出の点で，BOK 構築には適さないことがわかる．そこで，BOK 構築に適した版管理及び競合検出を実現する手法を紹介し，それを活用するシステムの可能性を議論する．

1　序　論

特定の分野を構成する概念をまとめたものとして，知識体系（Body of Knowledge, BOK）がある．その一例である，Computing Curricula 2001 Computer Science に収録された CS Body of Knowledge（CSBOK）は，計算機科学のための BOK である（Joint Task Force, 2001）．

CSBOK を作成したのは IEEE Computer Society と Association for Computing Machinery の統合タスクフォースであるが，このような一部の有識者によってではなく，不特定多数の作業者によって BOK を構築しようという試みがある．増永らによる WikiBOK である（増永，2012）．

WikiBOK は，Wiki のような誰でもアクセスできるウェブアプリケーション上で BOK を構築することを目指して開発されている．しかし，MediaWiki のような既存の Wiki システムをそのまま WikiBOK で採用することはできない．なぜなら，WikiBOK における BOK は木構造であることが想定されているが，既存の Wiki システムは木構造を構築しやすいものではないからである（すべての BOK が木構造だというわけではないが，CSBOK は木構造になっている．木構造でない BOK の例として，PMBOK（Project Management Institute, 2008）が挙げられる）．そこで，集団で BOK を構築するための新たな

Yabuki Taro　千葉工業大学社会システム科学部プロジェクトマネジメント学科

システムが必要になる．

本稿で対象とするシステムは，作業者がノードやエッジを生成したり，エッジを付け替えたりしながらBOKのための木構造（BOK木）を構築するためのものである．たとえば，１人の作業者がノードを生成し，それをBOK木に反映させることにすれば，BOK木に実際にノードが追加される．つまり，作業者の操作が直接BOK木に反映される．これは現在のWikiBOKが採用している方針である．集団でBOK木を構築するシステムは，作業者にBOK木を直接操作させず，複数の作業者が同じノードを作成した場合に限ってBOK木にノードが追加されるといった，間接的な方法（渋谷，2012）を採用することもできるが，本稿ではそのような方法は考慮しない．

本稿の第2節ではBOK構築システムの候補を列挙する．第3節ではBOK構築システムが持つべき機能を確認する．第4節ではBOK構築システムの機能の中では特に重要な，競合検出とマージについて議論する．

2　構築システムの候補

集団でBOK木を構築するためのシステムの候補を検討する．木構造を直感的に操作できるシステムとして最初に挙げられるのは，コンピュータのオペレーティングシステムの多くで採用されているファイルシステムであろう．順序性（3.5項を参照）を考慮しないなら，木構造のノードが子ノードを持っている状態は，ファイルシステムのフォルダの中にサブフォルダがある状態で表現できる．

本節では，ファイルシステムを集団で利用する方法として，共有フォルダ（2.1項）と，バージョン管理システムの一種であるSubversion（2.2項），独自のシステム（2.3項）を検討する．

2.1　共有フォルダ

共有フォルダはファイルシステム上の特定のフォルダを集団で共有する仕組みである（図１）．

共有フォルダには次のような利点がある．

・主要なOSでサポートされている．
・作業コピーを簡単に作れる．
・デジタル化されたものなら何でも扱える．
・ファイルシステムと統合できて，一度接続してしまえば直感的に使える．

共有フォルダには次のような欠点がある．

・順序性を実現しにくい．
・同名フォルダ［ファイル］の存在に気付きにくい．現在のWikiBOKでは，BOK木の中に同名のノードは存在できないことになっている．しかし，ファイルシステムでは同一フォルダ内でなければ同名フォルダ［ファイル］が存在できる．つまり，WikiBOKの仕様はファイルシステムでは実現しにくい．
・競合を検出できない．複数の作業者が同じ実体にアクセスするため，ある作業者の作業結果を別の作業者が気付かずに壊してしまう危険がある．ファイルへのア

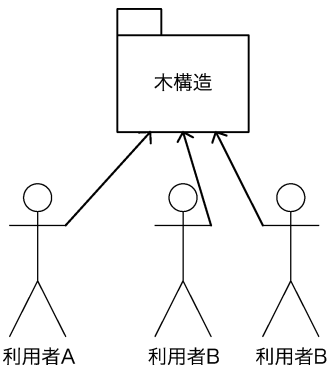

図１　共有フォルダ
（１つの実体に複数の作業者が同時にアクセスする．）

クセスをアプリケーションレベルで排他的にすることはできるが，それによって問題が解決されるわけではない．

・版管理ができない．古い版に戻せるようにするためには，作業者が自分でコピーを取るような運用をしなければならない．しかし，作業者が個別にコピーを作ると，コピー同士の新旧がわかりにくくなるという問題がある．

集団で１つのマインドマップを編集できるウェブアプリケーション BeautifulMind.io も，本稿執筆時点では「版管理ができない」という問題があり，BOK 構築には利用しにくい状態である（動作画面は図２のとおり．http://beautifulmind.io/において MIT ライセンスで公開されている．版管理は追加予定機能リストには入れられている）．

2.2　バージョン管理システム

バージョン管理システムはフォルダとファイルの版管理をするシステムである．バージョン管理システムは大きく分けて，フォルダとファイルのデータベースであるリポジトリが１つだけのもの（集中型）と複数あるもの（分散型）の２種類があるが，本稿では集中型のバージョン管理システムの１つである Subversion（上平，2006）を取り上げる（図３）．

Subversion の基本的な利用手順は以下のようになる（１，２は一度だけ行えばよい）．

１．リポジトリを作成する．
２．リポジトリにフォルダをインポートする．
３．リポジトリからフォルダをチェックアウトして，作業コピーを作る．
４．作業コピーで作業する．
５．作業コピーをリポジトリに反映させる（コミット）か，リポジトリを作業コピーに反映させる（アップデート）．

作業者の最新のアップデート（アップデート経験が無ければチェックアウトだが，本稿ではそれも「最新のアップデート」と呼ぶことにする）の直後の状態を BASE，作業コピーを WORK，リポジトリの最新状態を HEAD と呼ぶことにする．

Subversion は版管理をサポートしており，コミット時に振られるリビジョン番号によって，BASE と HEAD は自動的に比較される．

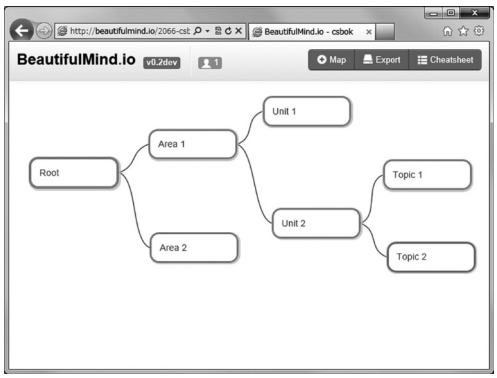

図２　BeautifulMind.io の動作画面
（他の利用者による編集結果がリアルタイムに反映される．）

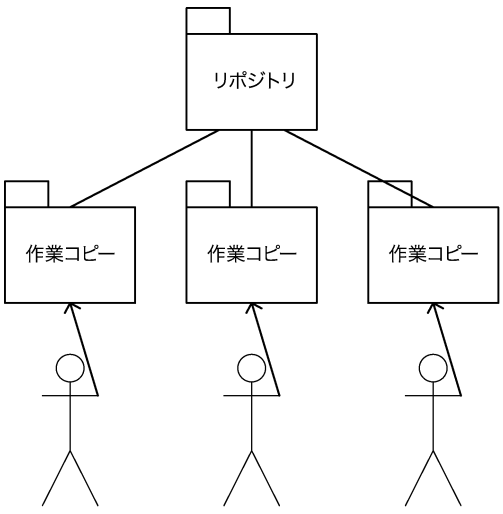

図３　Subversion
（作業者ごとに作業コピーを作って作業する．）

ある作業者（Alice とする）がコミットを試みると，BASE より HEAD のほうが新しい（リビジョン番号が大きい）かどうかが調べられる．BASE と HEAD が同じなら，Alice の最新のアップデート以降，誰もコミットしていないことになる．その場合はコミットを実行して，Alice の WORK をそのまま HEAD としてよい．BASE より HEAD が新しいなら，別の作業者（Bob とする）がコミットしていたことになる．その場合，Alice のコミットによって Bob の作業結果が破壊されないように，Bob の作業結果である HEAD と，Alice の作業結果である WORK の間に競合が存在するかどうかを調べなければならない（図 4）．

Subversion には競合検出機能があり，BASE より HEAD が新しいとき，コミットできるかどうかは自動的に調べられる．Bob の作業と Alice の作業に競合が無いなら，Alice の WORK が HEAD にマージされ，競合があるならコミットは中止される．

この競合検出機能はアップデート時にも活用される．何らかの作業を行った後で，HEAD を WORK に反映しようとするとき，BASE より HEAD が新しいかどうかが調べられる．BASE と HEAD が同じなら，Alice の作業中にリポジトリは更新されていないため，アップデートの必要は無い．BASE より HEAD が新しい場合には，競合の有無が調べられ，競合が無いなら HEAD が WORK にマージされ，競合があるならアップデートは中止される．

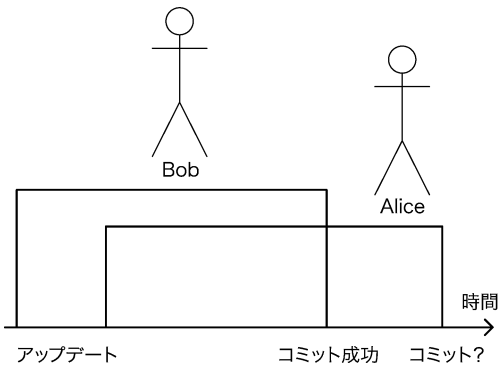

図 4　Subversion の版管理
（Alice の最新のアップデートからコミットまでの間に Bob がコミットしていると，競合検出が必要になる．）

Subversion には次のような利点がある（最初の 3 つは共有フォルダと共通）．

・主要な OS でサポートされている．
・作業コピーを簡単に作れる．
・デジタル化されたものなら何でも扱える．
・版管理機能が組み込まれている．
・競合検出機能が組み込まれている．

Subversion には次のような欠点がある（最初の 2 つは共有フォルダと共通）．

・順序性を実現しにくい．
・同名フォルダ［ファイル］の存在に気付きにくい．
・直感的な操作性という点で，共有フォルダより劣っている．
・競合検出機能のフォルダ階層への対応が不十分で，フォルダの名前を変更しただけでも，新しいフォルダを作ったと解釈され，コミットもアップデートもできなくなる．たとえば，フォルダ A の中にフォルダ X がある状況で，Alice が A の名前を B に変更してコミットした後で，Bob が X の名前を Y に変更すると，コミットもアップデートもできなくなる（図 5）．

2.3　独自のシステム

共有フォルダや Subversion の欠点は，独自のシステムを開発すればすべて解決できる．たとえば，図 5 の Subversion の欠点は，木構造を図 6 のようなテーブルで表現し，次のような 2 つの SQL 文で変更を実現するようなシステムを導入することで解決できる．

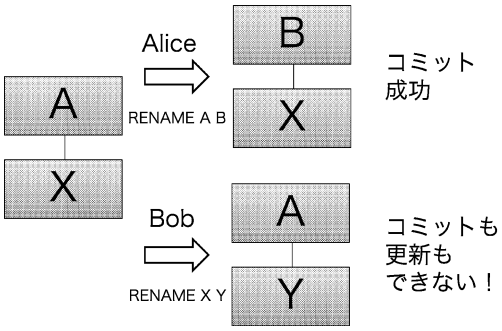

図5 Subversion の欠点
（フォルダ階層を変更するとコミットも更新もできない事態に陥る.）

```
UPDATE TABLE SET first=B WHERE id=1 AND first=A
UPDATE TABLE SET first=Y WHERE id=3 AND first=X
```

(A (X))

id	first	rest	contents
1	A	2	…
2	3	nil	
3	X	nil	…

(B (Y))

id	first	rest	contents
1	B	2	…
2	3	nil	
3	Y	nil	…

図6 独自のシステムによる木構造の表現と，それを用いた図5の解決

独自のシステムは，共有フォルダや Subversion の利点をすべて取り入れ，欠点をすべて排除できる．しかし，その開発には手間がかかるという大きな欠点がある．

3 要求仕様

前節で挙げたシステム（共有フォルダと Subversion，独自システム）を踏まえ，集団で BOK 木を構築するシステムに必要だと思われる要素を確認する．

3.1 作業コピー

ノードやエッジの追加や削除といった基本的な操作の結果を BOK 木に反映させる方法には，操作のたびに BOK 木を更新する方法と，複数の操作をまとめて BOK 木を更新する方法がある．操作のたびに BOK 木を更新すると，ある作業者の実験的な試み，つまり他の作業者には見せる必要のないものも，他の作業者に見えてしまう．これが他の作業者の作業の邪魔になる危険がある．そこで，他の作業者にも見せられる段階になってはじめて BOK 木を更新する方法を採用したい．この方法は，作業コピーによって実現できる．

作業コピーは，個人の作業のための BOK 木のコピーである．作業者は，BOK 木の作業コピーを作り，作業コピー上で BOK 木を編集し，他の作業者にも見せられる段階になったら作業コピーを元の BOK 木に反映させる．

共有フォルダでは，フォルダをローカルにコピーすれば作業コピーとなる．Subversion では，最初にリポジトリをローカルにチェックアウトするが，それによって作業コピーが必然的に作られる．

3.2 版管理

版管理は，BOK 木への変更が確定するたびに番号（リビジョン番号）を振って版を管理する仕組みである．この仕組みによって，複数ある作業コピーのうち，どれが最新かといったことが簡単にわかるようになり，作業コピーを BOK 木に反映させやすくなる．

共有フォルダには版管理機能が組み込まれていないため，古い作業コピーで新しい結果を上書きしてしまう危険がある．Subversion には版管理機能が組み込まれているため，そのような危険は無い（2.2 項を参照）．

3.3 競合検出

ある作業者（Alice とする）がコミットを試みるとき，その BASE と HEAD が等しけれ

ば, そのままコミットして, WORK を HEAD にすることができる．BASE より HEAD が新しければ，Alice の最新のアップデートの後で別の作業者（Bob とする）がコミットしているため，Alice の WORK をそのまま HEAD にすることができない．Alice の WORK のうち，Bob の作業結果と競合しないもののみを，BOK 木に反映させるのがよい．

Alice の WORK のうちで，Bob の作業結果と競合するものがあるかどうかを調べるのが競合検出機能である．作業コピーを使う BOK 木構築システムにおいて，競合検出機能は必須である．

共有フォルダに競合検出機能は無い．Subversion には競合検出機能があるが，フォルダ階層を木と見なすような使い方に適したものにはなっていない（2.2 項を参照）．

3.4 直感的な操作

木構造の編集は，直感的に操作できる GUI 上で行うのが望ましい．フォルダ階層で表現した木構造をシェルコマンドで操作したり，XML で表現した木構造をテキストエディタで編集したりしながら BOK 木を構築していくことも可能だが，そういう方法を採用すると，BOK 構築における作業者の技術的ハードルが高くなってしまう．

共有フォルダと Subversion はファイルシステムと統合されており, そのための GUI にも大部分の作業者が慣れていると思われる．そのため，フォルダ階層を BOK 木と見なすようにすれば，ノードやエッジの生成や削除は直感的に行えるだろう．

3.5 順序性

木構造は，あるノードの子ノード群に順序性があるものと無いものに分けられる. BOK 木に順序性が必要かどうかには議論の余地があが，現行の WikiBOK には順序性は導入されていない．

共有フォルダと Subversion は，その基盤となるファイルシステムの性質を引き継いでいる．ファイルシステムには同一フォルダのサブフォルダに順序性は無い．そのため，共有フォルダでも Subversion でもフォルダ階層を BOK 木と見なす場合，その木に順序性は無い．

3.6 多様なファイル

本稿では，BOK として文字列のラベルが付いたノードからなる木を想定しているが，文書や画像，動画などの補足的な資料を BOK に添付したいという要望はありえる．BOK に任意の形式のファイルを添付できれば，そのような要望に対応できる．

共有フォルダと Subversion は，ファイルシステムがその基盤となっているため，デジタル化されていればどんなものでも簡単に添付できる．

3.7 同名ノードの禁止

WikiBOK では, BOK 木に同名のノードが存在することは許されない．この規則を守るためには，作業者が同名ノードを作らないように気を付けるか，同名ノードを作れないシステムがなければならない．

共有フォルダと Subversion は，ファイルシステムがその基盤となっているため，同一フォルダ内でなければ同名のフォルダやファイルが存在できる．つまり，共有フォルダや Subverion をそのまま使う場合には，同名ノードは存在できないという規則を，システムの仕様として実現するのは難しい．

3.8 Undo と Redo

BOK 木への編集をやり直すのが Undo, やり直した操作をあらためて実行するのが Redo である. BOK 木をフォルダ階層で表現し，共有フォルダで編集する場合には，ファイル

システムがサポートする範囲での Undo と Redo が可能である．Subversion を使う場合は，コミットされたものはすべて版管理されるため，Undo と Redo は完璧に実現できる．Subversion の作業コピーでは，共有フォルダと同様，ファイルシステムがサポートする範囲での Undo と Redo が可能である．

3.9 その他

集団で何かを作り上げるシステムの成功のために必要と思われるものは他にもある．例として，Wikipedia の成功のために導入が検討されたもののうち，WikiBOK でも導入を検討すべきだと思われるものを以下に挙げる（リストの一部は（Lih, 2009＝2009）を参考にしている）．

・キャメルケースによるリンク（簡単なリンク生成）
・フリーリンク（簡単なリンク生成）
・GNU Free Documentation License（参加しやすいライセンス）
・Diff の表示（変更箇所の容易な把握）
・名前空間：利用者（貢献者の自己顕示欲）
・名前空間：記事（百科事典本体の分離）
・名前空間：ノート（議論の場の分離）
・井戸端（Wikipedia 自体についての議論の場・議論の保存）
・「ルールを無視しよう」（参加しやすくするためのスローガン）
・IRC（迅速な議論）
・テンプレート：スタブ（できたばかりのページを目立たせる効果）
・テンプレート：要出典（出典の提示要求の簡便化）
・テンプレート：sofixit（参加しやすくするためのスローガン）
・API（機械的な編集作業）
・管理者：oversight（完全な削除．法的な問題等への対処）
・管理者：削除（ページの削除）
・管理者：ブロック（アカウント停止．荒し対策）
・保護（更新禁止．荒らし対策）
・サンドボックス（練習の場）
・最近更新したページ（監視・参加の簡便化）
・ウォッチリスト（監視）
・差し戻し（更新の差し戻し・事故への対応）
・削除投票（ページを削除するかどうかの決定）
・削除依頼（ページの削除依頼）
・Quickpolls（ユーザをブロックするかどうかの投票．荒らし対策）
・Three Revert Rule（3 回差し戻されたページを更新禁止にする機能．荒らし対策）
・多言語化（ページを複数言語で並行して作成できるようにすること）
・承認（ページがある程度の品質を持っていることの証）
・IP のブロック（特定の IP からの編集禁止．荒らし対策）
・ClueBot NG のような不適切なページを削除する BOT
・カテゴリ関連機能
　—ページのカテゴリへの追加
　—カテゴリへのページの追加
　—カテゴリの名前変更
　—カテゴリの移動
　—カテゴリの削除
　—カテゴリの俯瞰機能

4 競合検出

Wiki における競合検出方法を参考に，BOK 構築における競合検出方法を提案する．

4.1 Wiki における競合検出とその解決

Wiki では次のような手順で競合を検出する．

1．Alice が編集を開始する．
2．Bob が編集を開始する．
3．Bob がコミットを試み，成功する．
4．Alice がコミットを試みる．
(a) 最新版（Bob の編集結果）と比較する．
(b) 編集箇所が重複しているか？
・重複がなければコミット可能であるため，コミットする．
・重複があればコミットはできない．自分の作業を破棄する，あるいは編集に戻るなどといった，コミット以外の方法をユーザは選ばなければならない．

この競合検出方法は，BOK 木構築には向かない．Wiki はページ単位のコミットであるため，複数の作業者が同一のページを編集しない限り競合は発生しない．しかし，BOK 木を Wiki で管理し，BOK 木のノードが Wiki のページに対応するような方法を採用すると，木を編集する際には複数のページが同時に変更されるため，競合が発生しすぎるのである．BOK 木全体が Wiki の 1 ページに対応するような方法を採用して，テキストベースの diff で競合を検出しながら BOK 木を構築するのも難しいだろう．

4.2 3-way XML マージ

木構造の競合を検出したりマージしたりする方法の 1 つに 3-way XML マージがある (Lindholm, 2004)．3-way XML マージは，BASE と HEAD，WORK の 3 つの XML のマージ手法である．3 つ組{BASE，HEAD，WORK}の 3-way XML マージでは，HEAD はすでにコミットされたものであるため，BASE から HEAD への変更はすべて採用される．それに対して，BASE から WORK への変更は BASE から HEAD への変更と競合しないもののみが採用される．

BOK 構築に 3-way XML マージを利用すると，図 5 では失敗していたマージが図 7 のように可能になる．

3-way XML マージならば，図 8 のような，より複雑な状況でもマージできる．図 8 では，Alice が X の名前を P に変更し，Z の下に B を生成してコミットに成功した後で，Bob が X の名前を Q に変更し，Z を Y の下に移動してコミットを試みる．P の名前は競合しているため，先に成功した Alice の操作が優先される．両者の他の操作は競合しないため，すべて採用できる．

BOK 木構築に 3-way XML マージを利用する手順は次のようになる．

1．木を XML に変換する
2．3-way XML マージでマージを試みる
3．エラーがあるか？
・エラーが無ければコミット可能である

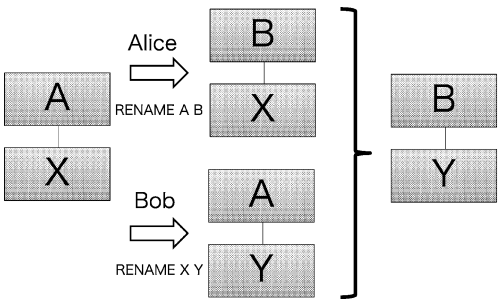

図 7 3-way XML マージ 1
（図 5 の問題が解決される．）

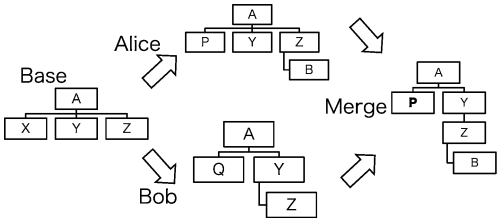

図 8 3-way XML マージ 2

ため，コミットする
・エラーがあればコミットは失敗である．コミット以外の方法をユーザは選ばなければならない（4.2.2 を参照）
4．XML を元のデータ構造に戻す

3-way XML マージの詳しい手順は以下のようになる（作業前のリビジョンが n だと仮定する）．

1．Alice が編集を開始する（$BASE_{Alice}$ のリビジョンは n）．
2．Bob が編集を開始する（$BASE_{Bob}$ のリビジョンは n）．
3．Bob がコミットを試み，成功する（HEAD のリビジョンは $n+1$）．
4．Alice がコミットを試みる（$BASE_{Alice}$ と HEAD のリビジョンが比較される）．
・$BASE_{Alice}$ と HEAD のリビジョンが同じなら，コミットを実行する（$WORK_{Alice}$ が HEAD になる）．
・$BASE_{Alice}$ より HEAD のほうが新しいなら，
(a) $\{BASE_{Alice}, HEAD, WORK_{Alice}\}$ で 3-way XML マージを試みる．
(b) エラーがあるか？
—エラーが無ければコミット可能であるため，コミットする．
—エラーがあればコミットはできない．コミット以外の方法を Alice は選ばなければならない（4.2.2 項を参照）．

4.2.1 作業コピーのアップデート

3-way XML マージは，コミット時だけでなくアップデート時にも利用できる．3 つ組{BASE, WORK, HEAD}で 3-way XML マージを試みれば，BASE から HEAD への変更のうち，BASE から WORK への変更と競合しないものだけを採用できる．つまり，自分の作業（WORK）を失わずに，最新版（HEAD）を取り込むことができる．

4.2.2 エラーへの対応

3 つ組{BASE, HEAD, WORK}の 3-way XML マージでは，BASE から WORK への変更のうち，BASE から HEAD への変更と競合するものは採用されない．採用されないものがある場合，作業者には次のような選択肢がある．

・マージ結果でコミットする（作業の一部は失われる）
・WORK に戻り，作業コピーのアップデート（4.2.1 項）を検討する
・HEAD を採用する（作業は失われる）
・BASE に戻る（作業は失われる）

5 結論と展望

本稿では，集団で BOK 木を構築するためのシステムについて検討した．検討したシステムは，共有フォルダと Subversion，独自のシステムである．

共有フォルダには版管理機能と競合検出機能がないため，BOK 木構築には向いていないことがわかった．この問題は，Beautiful-Mind.io のような，ウェブブラウザを使って集団で木を構築するシステムにも共通している．

Subversion には版管理機能と競合検出機能が備わっている．しかし，BOK 木をフォルダ階層で表現して Subversion を利用しようとすると，競合が発生しすぎるため，BOK 木構築には向いていないことがわかった．

版管理によって競合が発生している可能性があると判断されたら，本当に競合が発生しているかどうかを調べ，競合があるならそれについて報告し，無い場合は複数の変更をマージしなければならない．競合検出とマー

ジには木構造を意識したアルゴリズムが必要だが，そのようなアルゴリズムの候補として，3-way XML マージを検討した．

独自のシステムを開発すれば，既存のシステムの欠点をすべて克服できるが，開発に手間が掛かるという問題があった．開発の手間を最小限にとどめるためには，ファイルシステムの GUI を採用し，コミットやアップデートの際には 3-way XML マージを用い，リポジトリには木構造を XML 形式で保存するようなシステムが有望であろう．フォルダ階層と XML の相互変換を行うソフトウェアさえあれば，システムが完成するからである（図 9）．変換ソフトウェアが FTP や WebDAV などの標準的なプロトコルをサポートすれば，タッチデバイスのような，一般的な PC とは異なるデバイス上でも容易に BOK 木を構築できるようになるだろう．

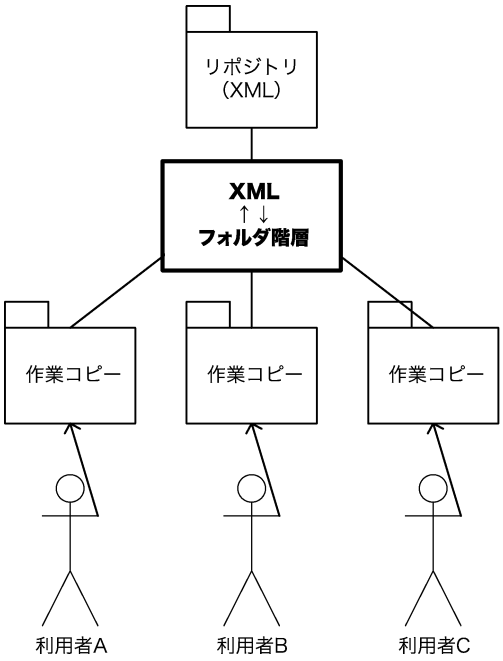

図 9　3-way XML マージを利用する独自システム（開発が必要なのは XML とフォルダ階層を相互変換する太字部分のみ．）

参考文献

[1] 上平哲（2006）『入門 Subversion』秀和システム

[2] The Joint Task Force on Computing Curricula IEEE Computer Society and Association for Computing Machinery (2001) *Computing Curricula 2001 Computer Science*

[3] Lih, Andrew (2009) *The Wikipedia Revolution, Hyperion*＝（2009）千葉敏生（訳）『ウィキペディア・レボリューション』早川書房

[4] *Lindholm, Tancred (2004) A Three-way Merge for XML Documents. Proceedings of the 2004 ACM symposium on Document engineering*: 1-10

[5] 増永良文，石田博之，伊藤一成，伊藤守，清水康司，荘司慶行，高橋徹，千葉正喜，長田博泰，福田亘孝，正村俊之，森田武史，矢吹太朗（2012）「知識体系（BOK）創成支援システム WikiBOK の研究･開発」『第 3 回ソーシャルコンピューティングシンポジウム講演論文集』67-72

[6] Project Management Institute (2008) *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*, Project Management Institute

[7] 渋谷俊介，伊藤一成，増永良文，矢吹太朗，佐久田博司（2012）「集団の知識構造を抽出する WikiBOK システムの拡張」『第 4 回データ工学と情報マネジメントに関するフォーラム（DEIM2012）』