協調ハイパーメディアシステムにおけるDBトランザクション方式について

5W−3

井上 直樹,　松井 眞理夫
NTTデータ通信株式会社

協調ハイパーメディアシステム (CHMS)

我々はグループウェアの基盤環境として複数ユーザ共同利用型ハイパーメディアシステム（協調ハイパーメディアシステム：CHMS）を開発している。CHMSはグループで共有する資源を機能部品としてカプセル化し、OODBに永続オブジェクトとして管理する。この機能部品群をModel-Rep.-Pref.-Widgetという4つの構成要素からなるフレームワーク [1] に則って視覚化し、各メンバがリアルタイムに共有資源を操作できる環境を提供する。

背景および目的

CHMSはグループウェアの実行環境だけでなくプログラミング環境をも含んでいる。従ってユーザは機能部品の変更／新規作成が可能である。機能部品の定義には、自身が操作された際の処理を記述する必要があるが、部品群はOODBによって管理されているためDBへのトランザクション処理を記述することになる。しかし記述者にはDBの存在を意識させないようにトランザクション処理は隠蔽されていることが望ましい。

一方で機能部品のデータ構造は多様なだけでなく、ユーザの再定義により動的に変化する。そのため対応するトランザクションの形態も統一性を欠き、記述者がDBを意識せざるを得ないプログラミング環境となる恐れがある。この問題を解決するため、データ構造に依存しないトランザクション処理のフレームワークを規定することが、本研究の目的である。

イベントループ構造

GUIを備えたアプリケーションの多くは、ユーザの操作をイベントとして扱い、処理する方式をとっている。この場合プログラムの構造は、「イベントの取得」「イベントの種類に応じた処理」の繰り返しといったイベントループ構造をとる。

トランザクション方式

CHMSでは上述のイベントループを一単位としてショートトランザクションを規定する方式を採る。つまり後述するタイミングで、各イベントループに最大一度のショートトランザクション処理を行なうこととし、記述者からはその処理を隠蔽する。（なおロングトランザクションに関してはこの限りでない。）

A Study on Transactions for Cooperative Hypermedia System
Naoki INOUE, Mario MATSUI
NTT DATA COMMUNICATIONS SYSTEMS CORPORATION

本トランザクション方式は、ユーザ操作の内容に従って次の二種類に分類される。

(1) 操作が1イベントで完了する場合

これは、1回のイベントループで処理が完了する場合である。そのため、該イベントで生成される処理手順群を1トランザクションと規定して、イベントループ終了時にDBへのアクセスを実行する。

本例としては、「機能部品の削除処理」等が挙げられる。この場合、deleteキーを押下した際に発生する1つのキーボードイベントによって、操作対象のオブジェクトを削除することができる。

(2) 操作が複数イベントの連係からなる場合

この場合は、関連する複数イベントを(a)操作開始イベント、(b)二番目以降の操作イベント群、(c)操作終了イベントの三つに分けることができる。次にこの三つのイベントの場合に従って処理手順を説明する。なお図1に処理の概略を図示する。

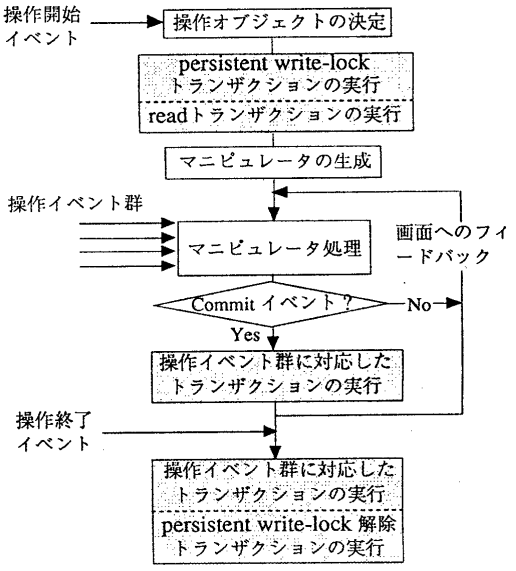

図1　(2) の場合の処理の流れ

(a) 操作開始イベントの処理

操作対象の機能部品におけるModel, Rep., Pref. のどの部分に処理を施すかを決定し、persistent write-lockトランザクションを実行する。このロックはwrite-lockをかける権利を占有するものである。従って他ユーザは更新伝播処理等を行う際にread-lockをかけるこ

とができる。なお、Model, Rep., Pref. の保持する情報は次表のような違いがある。

Model	全メンバに共有される、機能部品に本質的な情報（テキストの文字列等）
Rep.	移動／変形処理等を行なう際の操作権情報 Pref.が未定義の場合のデフォルト情報
Pref.	各人の好みに相当する情報 （テキストの色、フォントの大きさ等）

次に操作対象となるオブジェクトへのreadトランザクションを実行して必要な情報を得た後、後述するマニピュレータの生成を行なう。

(b) 二番目以降の操作イベント群の処理

これらの処理は、マニピュレータと呼ばれる非永続オブジェクトを導入することで対処する。マニピュレータは操作の１イベント毎に対応する処理を行い、ユーザにグラフィカルフィードバックを与える。しかし、１イベント毎の処理結果はマニピュレータ内にローカルデータとして保持されており、DBへは反映されない。

なお操作中における変更経過を他ユーザに反映させたい場合は、永続オブジェクトへのアクセスを記述すればよい。永続オブジェクトが変更された場合は、システムがcommitイベントを発生させマニピュレータの処理中でもDBへのトランザクション処理が自動的に行われる。

(c) 操作終了イベントの処理

操作終了イベントが発生した時点で、マニピュレータに保持されているローカルデータを基にトランザクション処理を実行する。次に(a)でかけたpersistent write-lock を解除するトランザクションを実行する。なお、一連のイベント群の途中でシステムダウン等の障害が発生した場合は、最後にcommitした時点の状態にまで復帰する。

(2) のタイプのトランザクション処理の具体例

●複数メンバで共有されている台紙上に貼られた機能部品の移動処理（図２参照）

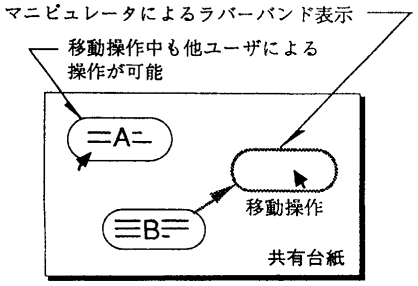

図２　共有台紙における移動操作

図２ではA, B 二つの機能部品が貼られているが、この場合Bを移動している時でも他ユーザはAに対する操作を行ないたい。移動処理は台紙のレイアウト情報を変更する処理だが、台紙にpersistent write-lockをかけるとAの移動処理が行なえなくなる。従って台紙ではなく、BのRep.に対してpersistent write-lockをかける。Rep.は移動処理における操作権情報を保持しており、他ユーザはBに対して移動処理が行なえなくなるが、Aに対しては自由に操作が可能である。

移動処理を開始するとユーザの移動操作に応じてラバーバンドを表示するマニピュレータが起動される。ユーザがマウスボタンをリリースした時点で終了イベントが生成され、台紙上のレイアウト処理を変更するトランザクション処理が実行される。

●テキスト編集中における他ユーザによるレイアウト変更処理

テキスト部品を複数メンバで共有している場合、作成者がテキストを編集している最中においても、それをライブコピーしている他ユーザは、テキストのフォントサイズを変更する等のレイアウトを変更する処理を行ないたい。

テキスト部品の編集はModelの変更であり、フォントサイズの変更はPref.の変更であるため、persistent write-lockトランザクション実行時に衝突は起こらず、同時に両者を変更することが可能である。

フォントサイズの変更は、メニュー選択等で発行される１イベントによって操作が完了する(1)のタイプの処理であり、直ちに変更処理は実行される。一方テキストの編集は、(2)のタイプの処理のため、テキスト編集のフィードバックを与えるマニピュレータが起動され、編集中のテキスト情報の変化は編集者のみにしか反映されない。編集が終了すると、テキストの内容を変更するトランザクションが発生し、他ユーザへ編集結果が反映される。

まとめ

本論文では、イベントループを単位とするトランザクション処理のフレームワークについて述べた。現在プログラミング環境については開発中であり、インタプリタ言語を提供する予定である。このインタプリタでは、本フレームワークに則ったトランザクション処理を内部機構として実現することで、DBへのアクセス部分は記述者から見えないものとなっている。なお、現在用意されている機能部品群は単純なものに過ぎないが、多くの高機能部品群が記述された場合の本トランザクション処理の有効性について、今後検討する予定である。

参考文献

[1] 牛田:「共有オブジェクトのGUIフレームワークについて」, 情報処理学会第45回全国大会, 5W-1, 1992