ネットワーク型映像制作ソフトウェア群の開発

稲蔭 正彦 [1)]　玉山 武 [2)]　斎藤 賢爾 [3)]
Masa Inakage　Takeru Tamayama　Kenji Saito

1) 慶應義塾大学環境情報学部 〒252-0816 神奈川県藤沢市遠藤 5322　inakage@sfc.keio.ac.jp
2) 慶應義塾大学 SFC 研究所　takeru@imgl.sfc.keio.ac.jp
3) 慶応義塾大学大学院 政策・メディア研究科　ks91@sfc.wide.ad.jp

概要

映像制作においてネットワークを介してコラボレーションを可能とするための映像制作支援ソフトウェアの開発を行った。開発の具体的な内容は、「脚本およびビデオストーリーボード記述言語」、「ノンリニア編集ソフトウェア」、「ネットワークと映像制作を考慮したバージョン管理の仕組み」である。

特徴としては、ノンリニア編集ソフトウェアが、上記記述言語を読み込むことができるため、プリプロの段階からポスプロの段階までを統一的に扱うことができる点にある。開発プラットフォームは GNU/Linux を想定し、開発ツールおよびライブラリなどは OpenSource として公開されているリソースを使用する。また、開発したソフトウェア群は、 OpenSource として公開する予定である。

1　背景および目的

近年のネットワークの高速化や放送技術の進展に伴い、映像コンテンツの需要が急速に高まっている。しかし、一方でコンテンツを制作する側では、CPU やメモリなどのハードウェアは IT 革命の進展により進歩しているものの、対して画質の向上により実際の作業に恩恵を受けているとは決して言えない。また、プロジェクト管理や作業の効率化という側面においては全くと言っていいほど、IT 革命による恩恵は受けていない。また、画質の向上はデータの増加やネットワークトラフィックの増加、演算処理の増加を促し、これが現在ひとつの大きな問題となっている。現在、高速なネットワークが普及しつつあり、コンテンツ制作においてもネットワーク上でコラボレーションが可能になりつつあり、またグリッドコンピューティング、P2P などネットワーク上のコンピュータが協調して動くためのフレームワークが整いつつある。そこで、ネットワーク上でのコラボレーションを前提とした映像編集のためのソフトウェア群の開発を行う。ここで「群」という表現を使用したのは、ひとつの単体パッケージソフトウェアではなく、小さなソフトウェアの集合やライブラリの集合を開発し、同時にそれぞれが協調して動くことでひとつのシステムを提供することができるように開発をするという意図による。映像制作全体の流れを考慮して、効率化をはかるという目的も含まれている。

2　目的

3　それぞれの機能

3.1　分散バージョン管理

コラボレーションでは、各自が持つ素材を実験的に組み合わせて新しい物を作り、評価し、やり直すといった行為が繰り返される。人間のそのような活動を支援するためには、素材や、その組み合わせの変更の履歴を保存し、必要な時に呼び出せることが必要である。すなわち、バージョン管理の実現が望まれており、その基礎となるバージョン付けについ

て、この研究開発の一環として考えることにした。人間のグループによる活動は、対等な仲間同士、すなわちピアグループを基礎としている。コンピュータネットワークにおいて、ピアグループが情報を共有するためには、マルチキャストにより分散ストレージを維持するというアプローチが考えられるが、バージョン管理を実現するためにはストレージの一貫性を保証する必要があるため、この場合、マルチキャストのメッセージを全順序 (total order) で順序付けることが考えられてきた。ところが、全順序を保証するマルチキャストプロトコルは高コストであり、帯域を多く消費し、また規模透過性に乏しい傾向があった。この研究開発では、この問題に対し、ストレージの一貫性の保証に対する要求を弱め、限定的な期間において全順序が失われることを許すことにより、帯域の消費を抑えた軽量なバージョン付けプロトコルを開発した。失われた全順序は、決定的なアルゴリズムにより回復される。順序付けが一時的であるにせよ失われることの不便は、常に因果順序 (causal order) を保証することにより解消した。

3.2 脚本、ストーリーボード記述言語の開発

ネットワーク上で映像制作のコラボレーションを行うためには、イメージの共有をいかに行うかが特に重要となる。そこで、プリプロダクションでもっとも重要な要素となる、脚本およびストーリーボードを記述するための言語 (XML によるもの) を開発する。次で説明するノンリニア編集ソフトウェアとのデータ互換 (XML による) をとることで、よりイメージを共有しやすいビデオストーリーボードを簡単に作成できるようにする。本提案ソフトウェアによって、プリプロダクションからポストプロダクションまでを ひとつの編集ラインを意識しながら行うことが可能となる。

3.3 スクリプティング機能を備えたノンリニア編集ソフトウェア

編集のためのスクリプティング環境の開発を行った。このソフトウェアは、モジュールの集合という形で提供することで、柔軟性・拡張性・総合性を持たせる。同時に、プラグインという形で機能の追加が可能な設計にし、多くの開発者が関わることができるように設計となっている。

4 開発内容

4.1 分散バージョン管理

4.1.1 分散バージョン管理モデル

分散バージョン管理をモデル化してアプローチする上で、ソフトウェア開発におけるバージョン管理を参考にした。大規模なコンピュータソフトウェアの開発には数多くのプログラマが関わる。かつ、開発プロジェクトの運営コストの低減のためには、人材を長距離、移動させずに開発に参加させたいという要求がある。また近年、特に活発に行なわれているオープンソースのソフトウェア開発では、世界中の様々な場所からプログラマが自発的にプロジェクトに加わり、開発を行なっている。これらのことから、コンピュータソフトウェアの開発では、地理的に離れたプログラマ同士の作業環境をネットワークによって結び、分散バージョン管理を実現する SCM (Software Configuration Management; ソフトウェア構成管理) ツールが発達している。

このバージョン管理のモデルを、遠隔での映像編集システムではピアグループにより実現する必要があり、新たにピアグループによる分散バージョン管理をモデル化した。

モデル化にあたっては、基本的にはサーバによる分散バージョン管理で用いられる抽象をそのまま受け継いだ。ただし、ピアグループによる分散バージョン管理では、管理者の役割をユーザが分担して担い、デポ (=リポジトリ) をそれぞれのマシンが持つ必要がある。この際のモデル化の選択肢としては次があった:

- 分散ハッシュテーブル方式を用い、各ピアが持つデポに格納される内容を拡散させる。すなわち、バージョン管理されるべきファイルの識別子からハッシュ値を計算し、その値に対応する

ノード上のデポのみにそのファイルの変更履歴を置く。
- 各ピアが等しいデポのコピーを持つ。

負荷分散の面では直観的には前者が望ましいと考えられるが、一度に変更されるファイルがすべて同一のデポで管理されないとなると、更新のアトミック性の実現が複雑になると考えられる。そこでこの研究開発では後者を採用した。(後述するように実際に後者の方が負荷はより分散する。) ただし、映像編集では扱うファイルのサイズが巨大であり、過去のすべての履歴を全ノードが冗長に持つことは現実的ではない。そこでデポの種類を 2 つに分け、大規模なストレージを持つマシンでは過去のファイルのすべての履歴を保持し、その他のマシンではデポにはファイルのフィンガープリントのみを保存する設計とした。

4.1.2 要求

ピアグループによる実現では、それぞれのピアは自分が参加していないプロジェクトに関して履歴を管理する必要がないため、デポとワークスペースが(扱うファイルの範囲という意味において)1 対 1 に対応するという特徴を持つ。デポ間の一貫性をどのようなレベルで、どのように保証するかが、このモデルを実現する上での課題となる。一貫性はできるだけ厳密に保証されるのがよいが、分散システムにおける一貫性の保証については通信や計算力のコストの問題がある。

そこで、実用上は困らないが、できるだけ一貫性に対する要求の弱いバージョン管理を検討した結果、ピアグループによる分散バージョン管理が満たすべき特性は次ではないかとの仮説を持つに至った。

特性 1: 更新の因果順序は破れない。
[理由]
あるユーザが連続して行なった 2 つの更新のうち、後者が前者を打ち消すようなものがあるとすると、この更新順序がノードに依って入れ替わることは避けなければならない。このことから更新の FIFO が破れないという条件が必須となる。

次に、ある更新の内容を受けて、ユーザが別のファイルを修正しなければならないことがある。例えば、コンピュータソフトウェアにおいては、自分が使っているライブラリの API が変更されたような場合である。この場合も、ノードに依って更新順序が入れ替わると混乱が生じるし、そうでない場合との区別を更新の意味解釈を抜きにシステムが行なうことはできないので、因果順序が守られることが望ましい。逆に、因果順序と独立した全順序については、メッセージを整流させるシーケンサとしてのサーバを用いる場合でも、通信路の遅延があるため、オフラインでコミュニケーションを行なっていたとしてもユーザの意図通りにできるとは限らない。例えば電話で話しながら 2 人のユーザが順番に更新を行なっても、実時間で後に行なった操作がサーバに先に到達することがあり得るため、確実に順序付けたい場合には、サーバからの完了通知を待ち、因果順序に基づいて更新を行なう必要がある。

これらのことから因果順序が常に満たされることをシステムの要件とした。

特性 2: 更新の全順序はいずれ回復する。
[理由]
ユーザの意図通りにならないとしても、全順序が満たされない場合、同じファイル識別子に対する同じリビジョン番号が示す内容が、ノードに依って異なるという状況が起き得るため望ましくない。かといって、常に全順序を保証するとなるとコスト高である。

そこで、全順序が破れている可能性が検出された場合、すべてのノードで同一の内容を示すようにリビジョン番号が付け替えられることをシステムの要件とした。

4.1.3 プロトコル設計

■局所的情報に基づく自律的なバージョン付け

プロトコルの設計に当たっては、余分なメッセージを用いず、各ノードが自律的な判断に基づいてバー

ジョン付けを行なうことにより帯域を節約することが望ましい。このプロトコルが満たすべき順序付けとして、因果順序 (常に) と全順序 (破れてから回復してよい) がある。このうち、因果順序については、メッセージに全ノードの論理時計を載せることにより、アプリケーションに譲渡する契機を受信側で自律的に判断できるプロトコルが知られている。全順序については、シーケンサとなるノードを用いない限りこのようなことはできない。シーケンサとなるノードを導入すると、ピアグループによる分散バージョン管理のモデルが崩れるため、それは避けるとすると、何らかの形で条件を緩めることにより、自律的な判断を可能にしなければならない。この研究開発では、全順序は常に満たしている必要はなく、順序の破れが検出されてから回復すればよいので、受け取った 2 つの更新を決定的アルゴリズムにより比較することで一意に順序付けを行なうという手法を採用できる。

■因果順序に基づく耐競合バージョン付けプロトコル

[プロトコルの概要]

提案するプロトコルの概要は次の通りである:

1. 各ノードは、自ノードにおいて更新が行なわれた際、変更リストを信頼できるマルチキャストにより送信する。その際、メッセージにはベクタ時計を含める。
2. メッセージを受け取ったノードは、ベクタ時計による因果順序の判定アルゴリズムに則って、自ノードのデポへの譲渡が可能なメッセージを判定する。
3. 因果順序を満たすメッセージがある場合、それが競合条件を満たしていなければ、すぐにデポに譲渡する。
4. 競合条件を満たしている場合、決定的なアルゴリズムにより、問題となっている 2 つのリビジョンのうち採用するリビジョンを選択する。採用されなかったリビジョンは、当該リビジョンの枝リビジョンに位置付けられる。採用されないリビジョンが複数ある場合は、枝リビジョンの間で更に決定的なアルゴリズムによる順序づけを行なう。
5. 採用されなかったリビジョンを元に編集を行なっているワークスペースを持つノードでは、ユーザへの通知を行ない、競合解決を促す。
6. 一連の解決が終了すると、リビジョンの示す内容は全ノードで一致している。

4.1.4 プロトコル詳細設計

■ベクタ時計による因果順序保証

ベクタ時計により因果順序を保証するプロトコルは Schiper らにより発案され、Birman らの ISIS Toolkit でも採用されている。

基本的なアイデアは次の通りである:

1. ノード毎に、メッセージの送受信等のイベントの発生時に加算されるカウンタ (論理時計) を持ち、その初期値について同意しているものとする (例えば 0)。
2. ノードは、自分以外のノードについても、自分が知る限りの論理時計の値を保持している (初期値は与えられているので、少なくとも初期値を示している)。
3. メッセージをマルチキャストする際は、自ノードの論理時計と、自分が知る限りのその他のノードの論理時計の値をメッセージに添付して送付する。
4. メッセージを受け取ったノードは、自分が知る、送り元の論理時計の値よりもメッセージに添付された値の方が 1 つだけ大きい場合、自分が知る送り元の論理時計の値を更新する (そうでない場合、キューにメッセージを保管する)。

その他の論理時計については、もしメッセージに添付された値が、自分が知る値よりも大きい場合には、因果的に先行する、自分の知らないメッセージが存在することを示しているので、そのメッセージが到着するまで、受け取ったメッセージをアプリケーションに譲渡するのを待つ。

ピアグループによる分散バージョン管理では、マルチキャストは更新時のみ行なうようにできる。そ

して、サーバによる分散バージョン管理のモデルではシステムで一意だった変更番号を、ノード毎にローカルな値として持ち、変更番号を論理時計として用いることでこのプロトコルを利用できる。

■耐競合性の実現による全順序の緩やかな回復

因果順序が満たされ、デポに譲渡されたメッセージは、以下のマルチデポにおける競合条件を満たすかどうかを検査される:

[マルチデポにおける競合条件]

デポにある最新のリビジョン ≥ メッセージのファイルのリビジョン

競合条件を満たす場合、メッセージのファイルのリビジョンと、その番号に対応する、デポにおけるリビジョンとの間で、決定的アルゴリズムによりどちらを採用するかを判定する。無効となったリビジョンは、当該リビジョンの枝リビジョンに割り当てられる。無効となったリビジョンが複数ある場合は、枝リビジョンの間で更に決定的なアルゴリズムによる順序づけが行なわれる。無効となったリビジョンを元にして編集中のファイルがワークスペースにある場合、ユーザに通知して競合解決を促す。

[更新の順序を決定するアルゴリズム]

1. 対象となるファイルの内容が一致する場合、両者は区別しない。
2. 対象となるファイルの指紋を MD5 等の一方向ハッシュ関数 (システム毎に一意) により算出し、値の大きい方を採用する。
3. 稀に、ハッシュ値が一致する場合は、その旨を記録するマークを付け、参照される際にはユーザに警告する。

4.1.5 シミュレーションテストと結果

次の 2 つのプログラムを作成した:

- 本開発で提案するプロトコルに基づき、ピアグループによるバージョン付けをシミュレートするプログラム
- サーバによるバージョン付けをシミュレートするプログラム (分散ハッシュテーブルを用いたバージョン付けとも等価)。

各々のプログラムはノード数を引数として受け取り、ある 1 つのファイルのリビジョンの推移について、予め決められた条件でシミュレートし、ネットワークを流れたメッセージ数を出力する。ピアグループによるバージョン付けの場合は、ベクタ時計の比較回数も出力する。

[実験] 次の条件でシミュレーション実験を行なった:

1. ノード数が 1, 5, 10, 50, 100, 125, 250, 500 の場合について計測する。
2. 50% のノードがランダムな間隔で編集と更新 (必要な場合は競合解決も) を行ない、残りの 50% が通知を受けて同期する。
3. 各ノードは、10 回の更新操作をローカルに検出すると停止する。

実際には、例えば 500 人が参加する開発/制作プロジェクトで、250 人がある特定のファイルに対して更新する権限を持つということはまずない。上の条件は過剰な負荷をシステムに与えるためのものである。シミュレートされたメッセージ数を図に示す。

サーバによる場合と比較して、ピアグループによるバージョン付けの方が、ネットワークを流れるメッセージ数は少ないという結果となった。

[考察]

この結果は、この研究で開発したプロトコルが局所性に優れていることを示していると考えられる。ピアグループによる分散バージョン管理では、更新時以外にはメッセージはネットワークを流れない。一方、サーバによる分散バージョン管理、あるいは分散ハッシュテーブルを用いた分散バージョン管理では、更新だけでなく、同期や編集などの操作もすべてネットワークを介する。逐次、サーバへの問い合わせを行なうので、ネットワークの帯域の消費が

大きくなると考えられる。

4.1.6 脚本、ストーリーボード記述言語

脚本から編集までを一貫して行うための一要素として、脚本およびストーリーボードを記述するための言語の開発をおこなった。これまで、脚本やストーリボードを作成する場合、慣習的な書式というもは存在したが、それをきちんと書式化し、さらにタグ付け言語などによる規定は存在しなかった。

また、XML でこの言語を規定することにより、XML 文書から HTML や PDF、TEX などの文書への変換を容易にする。そのため、この開発のテーマであるネットワークを介したコラボレーションを拡大するものとなっている。

4.1.7 脚本の記述言語

脚本の書式は、欧米式、日本式の大きく 2 つがある。本開発では、できるかぎりその両方に対応するべく、言語の開発を行った。定義したタグのリストを以下に示す。

[脚本構成要素]

ヘッダ	<HEAD></HEAD>

[ヘッダの構成要素]

タイトル	<TITLE></TITLE>
作者	<AUTHOR></AUTHOR>
登場人物	<ROLE id = ””></ROLE>
名前	<NAME></NAME>
年齢	<AGE></AGE>
梗概	<SYNOPSYS></SYNOPSYS>
シーン	<SCENE></SCENE>
	シーン番号 no = ”1”
	場所 place = ”ext”
	時間 time = ”day”
	設定 setting = ”spaceship”
シーンへのディレクション	<Action Described>
ダイアログ（台詞）	<DIALOGUE>
	ロール（役） role = id
ディレクション	<DIRECTION></DIRECTION>
ブレイクダウンマーク	<BREAKDOWN/>
改ページマーク	<PAGEBREAK/>

具体例

```
<ROLE id=”1”>
<NAME>JAMESON</NAME>
<AGE>30</AGE>
</ROLE>
.....
< SCENE no=”49” place=”INT”
        setting=”ROOM-SPACESHIP”
        time=”DAY”>
<ACTIONDESCRIBED> Annie, John, Jameson, ...
</ACTIONDESCRIBED>
<DIALOGUE role=”1”>
      Roll it, Max
</DIALOGUE>
<DIRECTION>

     The lights dim. As...

</DIRECTION>
<BREAKEPAGE/>
</SCENE>
```

4.1.8 ストーリーボード記述言語

ストーリーボードの記述言語は、上記脚本に加え、それをブレイクダウンした情報を扱うことができるようにしてある。具体的には <CUT></CUT> 要素を用意し、そのブレイクダウンされたあとのカットという単位を用意した。また、<IMG/> タグを用意することでそのストーリーボードが指し示すイメージの位置を指定できる。

4.2 ノンリニア編集ソフトウェア

コンピュータ上で映像を編集するためのソフトウェアの開発を行った。このツールは、大きく３つの構成要素にわかれている。

1. タイムライン編集ソフトウェア
2. ビン（ファイルをいれておく入れ物のやくわり）
3. ビューワー

上記のようにそれぞれをべつべつのソフトウェアとして提供することにより、それぞれが、映像編集用途以外の役割を担うことができ、デジタルで映像を作るための有益なツールとなる。たとえば、ビンは、ファイルを探したり、その情報を見るための映像に特化したファイラーとしての役割を持ち、ビューワーは画像や映像のビューワーとしての役割を果たすことが可能である。また、これらのツールはクロスプラットフォームを念頭に開発をおこなっており、Windows, Unix, MacOS 上で動作させることが可能である。ただし、開発ターゲットは Linux を想定しているため、他のプラットフォーム上で正しく動作させるためには、検証および、多少の改変が必要であると思われる。使用したライブラリは、

- wxWindows
  `http://www.wxwindows.org`
- Simple DirectMedia Layer(SDL)
  `http://www.libsdl.org`

である。

4.2.1 タイムライン編集ソフトウェア

タイムライン編集ソフトウェアは、前述した脚本・ストーリーボード記述言語を読み込むことができるように設計を行った。そうすることで、映像作品の制作において早い段階からこのツールを利用でき、プリプロダクションからポストプロダクションまでを一貫した体制で行うことを可能とする。また、タイムライン編集ソフトウェアの状態を保存しておくためのプロジェクトファイルは SMIL2.0 をベースに映像編集のための拡張をおこなった言語により記述を行っている。

4.2.2 ビン

ビンは、ファイルの情報を表示し、その状態を保存しておくためのソフトウェアである。これは、タイムライン編集ソフトウェアのプロジェクトと一対一対応した状態を保持する。タイムライン編集で使用するファイル郡を保持しておくソフトウェアでもためである。ビンからタイムライン編集ソフトウェアへのドラッグアンドドロップを可能とし、そのまま編集ラインへファイルを載せることを可能としている。

4.2.3 ビューワー

ビューワーは、タイムラインやビンと通信を行い、それらからの命令に従って、映像を表示するソフトウェアである。また、このソフトウェアは単体でも動作し、画像、連番画像、ムービーの表示・再生が可能である。表示できる画像、映像は、プラグインで追加することを可能とする設計としたため、任意の開発者が新たな対応フォーマットを自由に追加することが可能である。

4.2.4 プラグイン機構

映像、画像、音のフォーマットは多数存在しているため、これらのソフトウェアに共通のプラグイン機構を用意した。

5 開発成果と特徴

5.1 本研究開発の成果

5.1.1 分散バージョン管理

- 特定のサーバとその管理コストを不要にする、ピアグループによる分散バージョン管理のモデル化を行ない、
- 当該モデルに沿ったバージョン付けの問題を、因果順序の保証と緩やかな全順序回復により解決し、
- かつ、サーバによる分散バージョン管理と比較して、帯域の使用効率と計算量の面で性能の劣化を防ぐことができた。

5.1.2 脚本・ストーリーボード記述言語

これまでの慣習的な様式から、形式的な記述言語をよういすることで、明確なフォーマットを指定す

ることが可能となり、また、XML で記述することにより、データベースとしての役割、各種文書フォーマットへの容易な変換を可能とする。さらに、後述のノンリニア編集ソフトウェアに取り込むことで、これまで完全に分離していたプリプロダクションとポストプロダクションとを一貫したものとして扱うことが可能となる。

5.1.3 ノンリニア編集ソフトウェア

上記の記述言語で書かれたファイルを読み込むことができる点が特徴のひとつであり、もっとも優位な点である。それらを利用することにより、映像制作におけるプリプロダクションからポストプロダクションまでを見通して行うことが用意となる。将来的には、現場で撮影したものをそのまま取り込んで簡単な編集ができるようにしたいと考えており、プリプロダクション、プロダクション、ポストプロダクションすべてを通して利用できるツールとなることを目指している。また、タイムラインツール、ビンツール、ビューワーツールが独立したソフトウェアとして動作することも特徴のひとつである。これらは、それぞれノンリニア編集ソフトウェアとして以外の利用方法が考えられるからである。同時にプラグイン機構によりサポートするファイルフォーマットを増やすことができる。

6 今後の課題と展望

6.1 分散バージョン管理

6.1.1 規模透過性の実現

現状では、ノード数が数百のオーダーになると、帯域の使用効率および計算量の面で、サーバによるバージョン管理ほどではないが、著しい性能低下が起こることが予想される。これについては、管理単位の分割等を用いることにより、バージョン管理との統合的なアプローチにより解決を図れる見込みがあり、更に検討を行なう。

6.1.2 実装

分散バージョン付けに関しては、今期の開発ではアルゴリズムの検証のためのプロトタイプ開発に終わっているので、実際に利用できるソフトウェアとして実装を早急に完成させたい。 この研究が提案するプロトコルでは、競合解決が完了するまで、更新履歴の全順序が失われる場合があるため、あるファイルのあるリビジョン番号が示す内容が、ある期間、ノード間で一致しないことがあり得る。このことが実用上、問題となるのか、問題となるとすればどの程度、重大かについて調査したい。

6.2 脚本・ストーリボード記述言語

今回開発した記述言語は、これまでの制作の例を分析して定義したものである。しかしながら、実際の現場でこれらが必要かつ十分に機能するかのテストは行っていない。今後は、実際の制作に利用することで評価を行い、十分に検証を行った後に、標準化を行いたいと考えている。

6.3 ノンリニア編集ソフトウェア

今回の開発では、開発期間が短期であったため、実用に耐えうるソフトウェアとしては十分な機能をもちあわせていない。たとえば、サポートするファイルフォーマットの数や、サポートするタイムコードの種類などである。また、GUI の操作に関しても、設計段階で描いた理想的な GUI の 6 割程度しか実装できていない。今後、これらの開発を続けてることで、商用のソフトウェアと同等以上の機能をもったソフトウェアへと成長させていく予定である。