メタデータにより処理ノードを自動判別する映像コンテンツ管理ワークフローの実現

添田　真史† 　小林　隆志†† 　横田　治夫††

† 東京工業大学 工学部 情報工学科　〒 152–8552 東京都目黒区大岡山 2–12–1
†† 東京工業大学 学術国際情報センター　〒 152–8550 東京都目黒区大岡山 2–12–1
E-mail: †soeda@de.cs.titech.ac.jp, ††tkobaya@gsic.titech.ac.jp, yokota@cs.titech.ac.jp

あらまし　本稿では，分散管理ワークフローの実現方法として，メタデータを用いた処理ノードの自動判別手法を提案し，その適用例として映像コンテンツ管理ワークフローを考える．映像・音楽などのマルチメディアコンテンツを効率よく管理する手段として，コンテンツに関するメタデータを用いる方法がある．特に最近では，メタデータを XML によって記述する方法が主流になりつつある．本稿で実現する映像コンテンツ管理ワークフローは，ワークフロー内の各処理ノードが XML で記述されたメタデータを読み取り，コンテンツに対する処理内容を判断し，その処理をするのに最も適したノードに作業を渡す．我々は，Java を用いた試作システムの実装を行い，その有効性を確認した．
キーワード　ワークフロー，マルチメディア処理，メタデータ管理，XML，負荷分散

Metadata Based Automatic Processing-Node Selection for Audio-visual Contents Management Workflows

Masafumi SOEDA†, Takashi KOBAYASHI††, and Haruo YOKOTA††

† Department of Computer Science, Faculty of Engeneering, Tokyo Institute of Technology
Ookayama 2–12–1, Meguro-ku, Tokyo, 152–8552 Japan
†† Global Scientific Infomation and Computing Center, Tokyo Institute of Technology
Ookayama 2–12–1, Meguro-ku, Tokyo, 152–8550 Japan
E-mail: †soeda@de.cs.titech.ac.jp, ††tkobaya@gsic.titech.ac.jp, yokota@cs.titech.ac.jp

Abstract　We propose a method of selecting nodes with metadata to process a workflow and apply it to audio-visual contents management workflows. One of the means for efficient multimedia content management is using the metadata about the content. Recently, it has been mainstream to describe metadata using XML. In the audio-visual contents management workflows discussed in this paper, each node reads the metadata written in XML, executes operations discribed by the metadata and dispatches the work to other nodes suited for the next operations. We implemented a prototype using Java and proved the validity of the proposed method.
Key words　workflow, multimedia processing, metadata management, XML, load balancing

1. はじめに

ワークフローの定義は，ビジネスプロセスとその一部の自動化である [1]．ワークフロー管理システムの導入によって，ビジネスプロセスにおいてさまざまなメリットが得られる．

ワークフロー管理の実現方法は各種考えられるが，大きく分けると

（1） 集中管理手法
（2） 分散管理手法
（3） エージェントによる手法

の 3 種類に分類できる．本稿では信頼性や拡張性を考え，(2) の分散管理の実現方法として，メタデータで記述された手順に適する処理ノードを自動判別する手法を提案する．ノードとは，本稿ではワークフロー内で処理を行うコンポーネントを指す．

提案手法の適用例として，ここでは映像コンテンツ管理ワークフローを考える．映像コンテンツのデジタル化が進み，テレビ局だけでなく，企業内や教育機関内などにおいても映像コンテンツのメディア変換・コード変換・編集・アーカイビングな

どが行われるようになってきている．それらの作業は，同一のコンピューティング資源や人的資源の環境下で行えるものではなく，場合によっては部署や組織をまたがることもあり，分散管理が適する分野であると考える．また，映像コンテンツの場合には，コンテンツ自体とメタデータの切り分けが明確であるため，提案する手法の特徴を示しやすいという利点もある．

部署や組織をまたがって作業の分散管理を行う環境では，限られたコンピューティング資源や人的資源を有効に利用することや，作業を中断したり故障中などで利用できない資源は割り当てないようにすることなどの工夫が必要である．また，映像コンテンツはインスタンスごとに処理内容が異なるため，処理の柔軟な変更が可能でなければならない．そのためには，処理の手順をメタデータに記述し，その内容に従って，次に行う処理に最適なノードを動的に決定していくアプローチが向いていると考える．このようなアプローチは，今後，Web サービスなどにおけるワークフローにも展開していくことが可能だと考える．

映像コンテンツを管理するためのメタデータは，MPEG-7 [2] や TV-Anytime Forum [3] などの標準化団体が仕様を定めている．これらの仕様は XML Schema で記述されているため，メタデータは XML を用いて表現される．XML は汎用的なデータ記述言語であり，最近注目を集めている．また，MXF [4] は，コンテンツとメタデータを統合したファイル形式である．この規格はワークフローに適用することを視野に入れているが，現時点では完全には規格化されていない．

本稿では，XML で記述されたメタデータを用いて映像コンテンツの管理を行うワークフローの実現方法を述べる．そして，Java により試作システムを実装し，その有効性を確認する．このワークフローの特徴は，ワークフロー内の各処理ノードが，メタデータを用いて，次の処理を行うノードの候補の中で最も負荷が小さいノードを自動判別し，コンテンツを送信することである．また，ノードが故障していた場合には，そのノードは自動的に選択から外される．

本稿の構成を以下に述べる．まず，映像コンテンツ管理のプロセスと，メタデータに記述すべきメタ情報を示す．次に，ワークフローの実現方法を述べ，ノードの動作について説明する．さらに，処理の分岐と待ち合わせの実現方法について述べる．次に，ワークフローのシステム構成と，実際の実験システムを用いたワークフロー処理の例を示す．最後に本稿のまとめと今後の課題を述べる．

2. 映像コンテンツ管理のプロセス

ワークフローを構成するには，ワークフローで自動化するビジネスプロセスを定義しなければならない．ここでは，映像コンテンツ管理のプロセスについて説明する．

まず，撮影した映像を PC に取り込むなどの方法により，動画ファイルを用意する．コンテンツと動画ファイルは一対一に対応するとは限らない．例えば，AVI ファイルは 2GB を超えるファイルを作ることができないため，1 つのコンテンツを 1 つのファイルに収めることができず，複数の動画ファイルに分割することがある(注1)．また，逆に 1 つの動画ファイルが複数のコンテンツを持つこともある．

次に，取り込んだコンテンツにさまざまな処理を行う．処理の順序はコンテンツによって異なる．以下に処理の例を挙げる．

- トランジション
  - カット
  - フェードイン・フェードアウト
  - ワイプ
- エフェクトの適用
  - ビデオエフェクト
    - モザイク
    - ゴースト
    - シャープ
  - オーディオエフェクト
    - コーラス
    - リバーブ
- 画像のスーパーインポーズ
- オーディオレート・オーディオ形式の変更
- オーディオのミキシング
- フレームレートの変更
- フレームサイズの変更
- エンコード
  - DV 形式から MPEG-1 形式
  - DV 形式から RealMedia 形式
  - MPEG-1 形式から RealMedia 形式

映像コンテンツが完成したら，指定された保存場所にアーカイブする．コンテンツをエンコードする前にアーカイブし，エンコードしたものもアーカイブするというように，コンテンツを複数回アーカイブすることもある．

3. メタデータ

コンテンツに関するメタデータは，XML で記述する．コンテンツ 1 つに対して，そのメタ情報を表す XML ファイルを 1 つ作る．以下に，メタデータに記述する内容を示す．

コンテンツの情報　ファイルに依存しないコンテンツの内容を表す．コンテンツの撮影日時やタイトルなどを記述する．この情報は，コンテンツをワークフローに登録した時点から変化することはない．

ファイルの情報　コンテンツを構成する動画ファイルの情報を表す．コンテンツは，複数の動画ファイルからなることがある．このため，コンテンツを構成する動画ファイルへのポインタと，そのファイルのうち，コンテンツが必要としている部分の時間情報をメタデータに記述する．このほかに，ファイル形式・圧縮形式・フレームレート・ビットレートも記述する．

(注1)：参照 AVI や AVI2 を用いることにより，2GB を超えるファイルを作ることもできる．

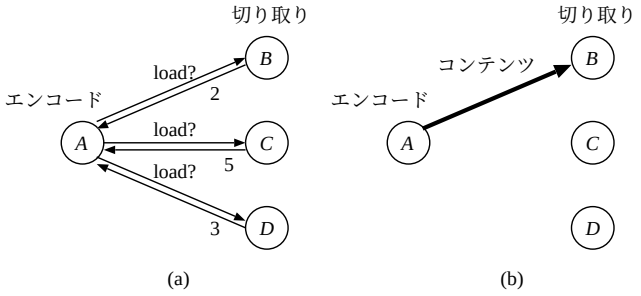


図 1　ノード間のコンテンツのやり取り

編集処理の手順と編集履歴　コンテンツに対する処理を記述する．ワークフロー内の各ノードはこの情報を読み，編集処理の内容を判断する．また，完了した処理に編集履歴を書き込むことにより，完了した処理とそうでないものを区別し，次に行うべき処理内容を判断することができる．

この中で，編集処理の手順と編集履歴以外の項目は、MPEG-7 [2] や TV-Anytime Forum [3] のメタデータ仕様で記述することができる．本稿では簡単化のため，メタデータを独自の仕様で記述する．

4.　映像コンテンツ管理ワークフロー

2. で述べたプロセスを複数のノードで行うワークフローを考える．各編集処理を担当するノードの間でコンテンツをやり取りし，映像コンテンツ管理を行う．1 つのノードは，1 つ以上の処理を担当する．本稿では，簡単のため担当する処理数を 1 とする．ノードが担当可能な処理は，ワークフロー実行時に予め決めておく．

4.1　負荷分散

負荷の偏りを抑えるために，他のノードにコンテンツを渡して処理をさせる際に，次の処理を担当できるノードの中から，故障しておらず，負荷が最も小さいノードを選んでコンテンツを送信する方法を考える．負荷を問い合わせても応答がないノードが存在する場合は，そのノードに何らかの障害が発生しているとみなして送信を行わない．

図 1 に負荷分散の手法を示す．図 1 では，エンコード処理を担当するノード A が，映像の切り取り処理を担当するノード B, C, D のいずれかにコンテンツを送信しようとしている．この場合，ノード A はノード B, C, D に負荷を問い合わせる．図 1（a）のように，ノード B, C, D からの負荷の応答がそれぞれ 2，5，3 であったとすると，図 1（b）のように，ノード A は最も負荷が小さいノード B にコンテンツを送信する．

ノードの負荷は，PC の CPU 使用率や，ノードで処理の開始を待っているコンテンツのキュー長などで測定する．コンテンツの編集処理を人間が行う場合でも，キュー長によりその人間の作業の量を判断することができる．

4.2　他ノードの認識

他のノードに処理を渡すためには，全てのノードが，どのノードが何の処理を行っているかを知っている必要がある．ノードが必要とする情報は

（1） ノードのアドレス
（2） ノードが担当する処理

の 2 つである．ノードのアドレスとして，IP アドレスまたはホスト名を用いる．

ノードが他のノードの情報を得る方法は，静的な方法と動的な方法の 2 種類ある．静的な方法では

$$\text{処理 } a \text{ を行うノード：} A_1, A_2, A_3, \ldots$$
$$\text{処理 } b \text{ を行うノード：} B_1, B_2, B_3, \ldots$$

のように，全てのノードのアドレスと処理内容を登録したテーブルを用意し，ワークフローを起動する．コンテンツを送信する際は，このファイルから得たノードの情報を利用する．この方法は実現が容易であるが，いったんワークフローを起動するとノードの追加や削除ができないという欠点がある．

動的な方法では，ノードのアドレス情報を一元管理するサーバを用意する．これは，DNS サーバのようなものと考えることができる．コンテンツを送信する際は，サーバに最適なノードを尋ねる．従って，ノードはサーバのアドレスのみを知っていればよい．この方法は，ワークフローの起動時にノードの追加や削除が行えるという利点があるが，サーバを用意するコストがかかるという欠点がある．ここでは，簡単化のため前者の方法を用いる．

4.3　ノードの処理手順

ワークフロー内の各ノードは，コンテンツとそのメタデータを用いて，以下の処理を行う．

（1） コンテンツを受け取り，メタデータの内容を読み，自分のノードに割り当てられた処理を行う．処理は，部分的には人間が介在することもある．処理が終わったら，今の処理によって変更されたコンテンツのメタ情報を自動的に書き換える．処理を待つコンテンツが複数ある場合は，キューに格納し，順番に処理する．
（2） メタデータ内の，自分が行った処理が記述されている箇所に処理の履歴を記入し，次の処理内容を得る．担当する処理がアーカイブの場合，自分のノードでコンテンツに対する処理が終わることがある．その際は指定されたノードに，このコンテンツに対する処理が完了したことを知らせる．特に指定がなければ，コンテンツをワークフローに追加したノードを通知先とする．
（3） 次の処理を行う全てのノードに負荷を問い合わせることで最適なノードを決定し，そこにコンテンツを送信する．次の処理がアーカイブの場合は，このノードでアーカイブ処理を行ったのち（2）に戻る．

4.4　ワークフローの動作

ここでは，簡単な映像編集作業の例を用いて，ワークフローの動作を説明する．まず，あるコンテンツ A を考える．コンテンツ A は，DV 形式の動画ファイル file.avi の，2 分から 5 分までの 3 分間からなる．このコンテンツを RealMedia 形式にエンコードしたのち，不要部分を切り取り，アーカイブしたいと

```
<Main>
  <Content>
    <Title>ContentA</Title>
    <FileFormat>AVI</FileFormat>
    <CODEC>DV</CODEC>
    <Framerate>30</Framerate>
    <SegmentList>
      <Segment id="id00">
        <FileName>file.avi</FileName>
        <MediaTimePoint>T00:02:00</MediaTimePoint>
        <MediaDuration>T00:03:00</MediaDuration>
      </Segment>
    </SegmentList>
  </Content>
  <Process>
    <Operation type="add"/>
    <Operation type="dVToRealMediaEncode"
               codec="rv9" bitrate="128000"/>
    <Operation type="realMediaCut" id="id00"/>
    <Operation type="archive" place="c:\movie\"/>
  </Process>
</Main>
```

図 2　ワークフローに追加された際のメタデータ

する．

この際のコンテンツ A に関するメタデータと処理内容を図 2 に示す．Content タグにはコンテンツと動画ファイルの情報が，Process タグには編集処理の手順が記述されている．

4.4.1 追　　加

ワークフローにコンテンツが追加されると，ノードは最初の Operation タグに，コンテンツを追加したことを示す履歴を書き込む．次に，まだ履歴が書き込まれていない Operation タグを読む．次の処理の type 属性は dVToRealMediaEncode であり，DV 形式から RealMedia 形式へのエンコードを表している．従って，コンテンツを追加されたノードは，エンコードを担当する全てのノードに現在の負荷を問い合わせ，最も負荷が小さいノードにコンテンツを送信する．

4.4.2 エンコード

DV 形式から RealMedia 形式へのエンコードを担当するノードは，このコンテンツを受け取り，履歴の書かれていない最初の Operation タグの内容を読み込む．bitrate="128000", codec="rv9" より，ビットレート 128000bps，圧縮形式 rv9 の RealMedia 形式にエンコードするよう指定されている．そこでエンコーダを起動し，指定された形式にエンコードする．エンコードの処理によって，メタデータのファイル形式，圧縮形式の内容が変わり，フレームレートの代わりにビットレートが記述される．ファイル名は file.rm となる．エンコード処理を終えると，今の処理が書かれた Operation タグに履歴を書き込んだのち，次の処理を得る．次の処理は RealMedia 形式の映像の切り取りであるため，この処理を担当する全てのノードに負荷を問い合わせ，最小の負荷を得たノードにコンテンツを送信する．

4.4.3 切り取り

RealMedia 形式の切り取り処理を担当するノードは，このコンテンツを受け取り，Operation タグの内容を読み込む．id="id00"とあるため，id00 のセグメントを切る．そこで，動画ファイルを切るためのプログラムを起動し，file.rm を 2 分から 5 分まで切り取ったファイルを作る．新しくできたファイルの名前は，ここでは file-0.rm とする．ファイル名の変更をメタデータに反映し，さらにファイルの時間情報も変更する．切り取り処理を終えると，履歴を書き込み，次の処理を得る．次の処理はアーカイブであるため，予め定められたノードにコンテンツを送信する．

```
<Main>
  <Content>
    <Title>ContentA</Title>
    <FileFormat>RealMedia</FileFormat>
    <CODEC>rv9</CODEC>
    <Bitrate>128000</Bitrate>
    <SegmentList>
      <Segment id="id00">
        <FileName>file-0.rm</FileName>
        <MediaTimePoint>T00:00:00</MediaTimePoint>
        <MediaDuration>T00:03:00</MediaDuration>
      </Segment>
    </SegmentList>
  </Content>
  <Process>
    <Operation type="add">
      <History name="host0"
               timePoint="2003-01-09T20:58"/>
    </Operation>
    <Operation type="dVToRealMediaEncode"
               codec="rv9" bitrate="128000">
      <History name="host1"
               timePoint="2003-01-09T21:08"/>
    </Operation>
    <Operation type="realMediaCut" id="id00">
      <History name="host2"
               timePoint="2003-01-09T21:10"/>
    </Operation>
    <Operation type="archive" place="c:\movie\">
      <History name="host3"
               timePoint="2003-01-09T21:12"/>
    </Operation>
  </Process>
</Main>
```

図 3　全ての処理が終わった際のメタデータ

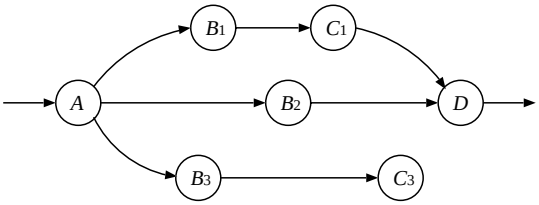


図 4　分岐と待ち合わせがある場合のコンテンツの流れ

4.4.4 アーカイブ

アーカイブ担当のノードは，c:\movie\ にコンテンツを置き，4 つ目の Operation タグに履歴を書き込む．次の処理はもうないため，指定したノードに，コンテンツ A に対する処理が完了した旨を伝える．この場合は通知先が指定されていないため，コンテンツをワークフローに追加したノードに通知する．メタデータより，ワークフローにコンテンツ A を追加したノードは host0 ということがわかるため，アーカイブ担当のノードは host0 に処理完了の通知をする．これで，コンテンツ A に対する処理は全て終わったことになる．この際のメタデータを図 3 に示す．Process タグ内にある履歴の name 属性は PC のホスト名，timePoint 属性はコンテンツに対する処理が終了した日時を表す．

5. 処理の分岐・待ち合わせ

4. で述べたワークフローでは，コンテンツに対して順番に 1 つずつ処理を行うことしかできない．ここでは，ワークフロー

の処理を拡張し，処理の分岐と待ち合わせを実現する方法について述べる．

処理の分岐と待ち合わせの例を図 4 に示す．ノード A で処理されたコンテンツは，B_1，B_2，B_3 の 3 個のノードに送信される．3 個に分かれたコンテンツはそれぞれ別々に処理され，そのうち B_1，B_2 に送信された処理はノード D で待ち合わせを行う．ノード D では 2 つのコンテンツが揃ってから，それらをまとめて処理する．また，B_3 に送信された処理のように，待ち合わせをすることなく C_3 で処理を完了することもある．

以下では，図 4 の処理を例として説明する．なお，ノード A, B, ... が担当する処理をそれぞれ a，b，... とし，ノード名に添え字がある場合は処理名にもつけることとする．

5.1 分　　岐

提案手法では，以下の手順により分岐の処理を実現することができる．分岐数を n とする．

（1） メタデータをコピーし，n 個用意する．これらをメタデータ 1，メタデータ 2，...，メタデータ n とする．
（2） メタデータ i に対し，分岐後の処理の i 番目に印をつける ($1 \leqq i \leqq n$).
（3） コンテンツと n 個に分かれたメタデータをそれぞれのノードに送信する．ノードは，印がついている処理を行う．

従って，図 4 の例ではノード A において，分岐後の処理 $b_1 \rightarrow c_1$，に印をつけたメタデータ，b_2 に印をつけたメタデータ，$b_3 \rightarrow c_3$ に印をつけたメタデータを作成し，それぞれをコンテンツとともにノード B_1，B_2，B_3 に送信する．

5.2 待ち合わせ

各ノードは独立して送信先ノードの決定を行うため，図 4 のようにノード B_2，C_1 が同じノード D にコンテンツを送信する保証はない．従って，何らかの方法により全てのコンテンツを同じノードに集める必要がある．それには，次の 2 種類の方法がある．

5.2.1 事前に送信先を決定

分岐処理を行う前に待ち合わせ処理を行うノードを決定する方法である．これは，ノードをワークフロー起動時から予め決定しておく方法と，分岐処理を行う際に決定する方法とがある．

ワークフロー起動時に送信先を予め決定しておく方法では，次の処理が待ち合わせを行う処理である場合，ノードはその処理を行うノードに負荷を尋ねることはせず，ある特定のノードに必ず送信する．これは，最も簡単な方法であるが，負荷分散を考慮しておらず，効率が悪くなる可能性がある．また，分岐処理を行う際に，待ち合わせ処理を行うノードの負荷を測定し，最適なノードを決定することもできる．図 4 の例では，ノード A での分岐処理の際にこの処理を行う．事前に決定しておく場合は，待ち合わせ場所をメタデータに記述する必要がある．

分岐処理を行う時点で決定する方法では，分岐処理を行うノードで，待ち合わせ処理を行うノードを決定し，その旨をメタデータに記述する．この場合は，分岐処理を行った時点の負荷を用いて待ち合わせ処理を行うノードを決定する．

```
<Process>
      :
    <Operation type="a" .../>
    <Operation>
        <Thread>
            <Operation type="b1" .../>
            <Operation type="c1" .../>
        </Thread>
        <Thread>
            <Operation type="b2" .../>
        </Thread>
        <OpenThread>
            <Operation type="b3" .../>
        </OpenThread>
    </Operation>
    <Operation type="c"  .../>
      :
</Process>
```

図 5　分岐と待ち合わせがあるメタデータ

5.2.2 最初に処理を終えたノードが送信先を決定

待ち合わせを行うノードのうち，最初に処理を終えて負荷を尋ね始めたノードが送信先を決める方法である．

待ち合わせ処理を行うコンテンツが複数ある際にそれらを区別して処理するために，待ち合わせの組を一意に定める必要がある．そこで，コンテンツがワークフローに追加された際のホスト名と時刻，そして待ち合わせ処理を行う組の通し番号を ID として用いる．ホスト名と時刻によりコンテンツを一意に定めることができる．これは，以下の理由による．

（1） 待ち合わせを行うコンテンツは，分割される前は同時刻に同ノードで追加された 1 つのコンテンツである．
（2） あるノードである時刻に追加されたコンテンツの数は高々1 つである．

さらに待ち合わせ処理の通し番号を加えることにより，同一コンテンツが同一の待ち合わせ処理を複数行う際にもこれらを区別することができる．

4.2 で述べた 2 種類のノードの認識方法のうち，動的な認識方法を用いた場合は，サーバが最初に処理を終えたノードを記憶しておくことにより実現できる．静的な認識方法を用いた場合は，以下の手順によりコンテンツの送信先を決定する．この方法では，デッドロックを回避するため，負荷を尋ねる順位を揃えておくことが必要である．

（1） 最高順位のノードに負荷を尋ね，ID を記憶させる．
（2） その ID に対して既に負荷が尋ねられていたら，負荷を尋ねる作業を中断する．以降は待ち合わせを行う候補のノードを巡回し，マークされた待ち合わせ場所を探し続ける．マークされたノードが見つかったら，そこにコンテンツを送信する．
（3） 次の処理を行うことのできる全てのノードに負荷を尋ね，最適なノードを待ち合わせ場所としてマークし，そのノードにコンテンツを送信する．
（4） 全てのコンテンツが待ち合わせ場所に揃ったら，初めのノードに記憶させた ID を解放する．

5.3 メタデータの記述方法

分岐と待ち合わせがある処理をメタデータに記述するには，それぞれの分岐の処理命令を `Thread` または `OpenThread` タ

```
<Process>
      :
    <Operation type="a" ...>
        <History .../>
    </Operation>
    <Operation>
        <Thread do="true">
            <Operation type="b1" .../>
            <Operation type="c1" .../>
        </Thread>
        <Thread>
            <Operation type="b2" .../>
        </Thread>
        <OpenThread>
            <Operation type="b3" .../>
            <Operation type="c3" .../>
        </OpenThread>
    </Operation>
    <Operation type="d" .../>
      :
</Process>
```

(a) ノード B_1 に送信されたメタデータ

```
<Process>
      :
    <Operation type="a" ...>
        <History .../>
    </Operation>
    <Operation>
        <Thread>
            <Operation type="b1" .../>
            <Operation type="c1" .../>
        </Thread>
        <Thread>
            <Operation type="b2" .../>
        </Thread>
        <OpenThread do="true">
            <Operation type="b3" .../>
            <Operation type="c3" .../>
        </OpenThread>
    </Operation>
      :
</Process>
```

(b) ノード B_3 に送信されたメタデータ

図 6　メタデータの分割

```
<Process>
      :
    <Operation type="a" ...>
        <History .../>
    </Operation>
    <Operation>
        <Thread do="true">
            <Operation type="b1" ...>
                <History .../>
            </Operation>
            <Operation type="c1" ...>
                <History .../>
            </Operation>
        </Thread>
        <Thread do="true">
            <Operation type="b2" ...>
                <History .../>
            </Operation>
        </Thread>
    </Operation>
    <Operation type="d" .../>
      :
</Process>
```

図 7　待ち合わせを行う処理が終わった際のメタデータ

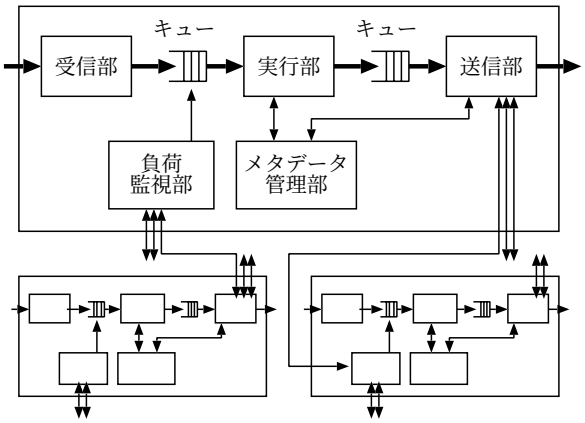


図 8　ノードのシステム構成

グでくくる．Thread，OpenThread タグは分岐処理の単位である．前者は分岐したのちに待ち合わせ処理があることを，後者は分岐したのち待ち合わせ処理をせずに終了することを意味する．以下では，これらのタグでくくられた処理をスレッドと呼ぶ．図 5 に，図 4 の例の処理を記述したメタデータを示す．

次の処理が分岐である際は，メタデータをスレッドの個数だけコピーし，次の処理を行うノードが担当するスレッドに do 属性を付与し，その値を true とする．OpenThread の場合は，そのスレッド内の処理のみを行えばよいため，分岐処理以降の Operation タグを削除する．

図 6（a）にノード B_1 に送信されたメタデータ,（b）にノード B_3 に送信されたメタデータを示す．各ノードが担当するスレッドに do="true"とある．ノード B_3 で行われる処理は，待ち合わせ処理を行わない OpenThread であるため，図 6（b）では c の処理が削除されている．

待ち合わせを行うノードでは，それぞれのメタデータから，分岐した処理の do="true"が付いている Thread 要素と，do 属性がない Thread 要素を交換する．OpenThread 要素は，待ち合わせをすることがないため削除する．全てのスレッドに do="true"が付いた時点で，全てのコンテンツが揃い，分岐処理の履歴が完成する．図 7 にノード D での待ち合わせを終えたメタデータを示す．こののちノード D において，コンテンツの処理が行われる．

6. システム構成

我々が提案するワークフローの，各ノードの構成を図 8 に示す．図の太い矢印はコンテンツの流れを表す．

1 つのノードは，以下の部品より構成される．

メタデータ管理部　コンテンツに関するメタデータである XML ファイルに対し，読み込み・書き出し・変更の操作を行う．

キュー　コンテンツに対する処理を格納するキューである．受信したコンテンツと処理が終わったコンテンツはキューに蓄えられ，順番に処理・送信が行われる．

負荷監視部　実行待ちのキュー長を監視する．他のノードに負荷を尋ねられた際に，負荷の値として現在のキュー長を答える．

受信部　コンテンツが送られてくるのを待つサーバである．コンテンツが送られてきたら，ファイルを受信するスレッドを作り，受信を開始する．受信スレッドは，ファイルの受信が終わったら，ワークフローの処理を開始する．ノードが担当する処理がコンテンツの追加だっだ場合は，コンテンツを受信することはない．

表 1　ホスト名と担当する処理の対応

処理	ホスト
追加	gerbera
エンコード	acacia, bluebottle
切り取り	clover, lavender
アーカイブ	crocus

実行部　コンテンツに対して，切り取り・エンコードなどのさまざまな処理を行う．ノードが担当する処理によって，この部分の内容は異なる．自動化できない処理の場合は，人間が編集処理を行うこともある．

送信部　他のノードの負荷監視部にノードの負荷を問い合わせ，最適な送信先ノードを決め，コンテンツを送信する．

7.　ワークフロー処理の例

ここでは，Java で実装した実験システムを用いて，実際のワークフロー処理の様子を示す．ノードの負荷はキュー長で測定し，他のノードの情報は静的に持っているとする．また，処理の分岐・待ち合わせは考慮に入れていない．

システム内の各ノードは，以下のコンポーネントを持っていればどのような PC でも構わない．

- JRE 1.4.0 以降
- 自分の担当する処理を行うことができる映像編集ソフトウェア
- ネットワーク環境
- 十分なハードディスク容量

ただし，コンテンツの追加またはアーカイブを担当するノードには，映像編集ソフトウェアは不要である．

コンテンツに対する処理のうち，現在のシステムで実装が完了しているものは，次の 3 つである．

- DV 形式から RealMedia 形式へのエンコード
- RealMedia 形式の動画の切り取り
- アーカイブ

今回の実験で行う処理は，DV 形式から RealMedia 形式へのエンコードを行ったのち，必要な部分のみを切り取り，アーカイブするというものである．以下では，DV 形式から RealMedia 形式へのエンコードを単にエンコードと呼ぶ．図 9 に本実験のワークフローの構成を，表 1 に本実験で用いる PC のホスト名と担当する処理の対応を示す．図 9 のノード内の文字は，PC のホスト名の頭文字を表す．

図 10 は，実験システムのスクリーンショットである．ウィンドウのタイトルは，担当する処理とホスト名を表している．ウィンドウの中には，処理を待つキューの状態が示される．add ボタンはコンテンツを追加するためのもので，その他の処理を行うノードには付いていない．

現在，エンコードを担当する 2 つのノードの状態は，図 11 のようになっている．図の目盛りは，キューの状態を表す．従って，図 11（a）の acacia と（b）の bluebottle でエンコード処理を待っているコンテンツは，それぞれ 1 個，2 個である．

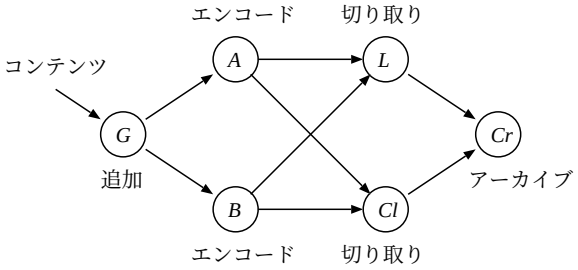
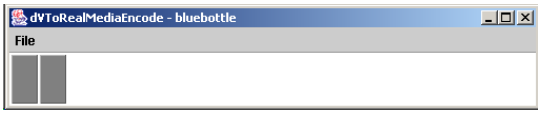


図 9　実験システムのワークフロー構成

図 10　ノードのスクリーンショット

(a) acacia



(b) bluebottle

図 11　エンコード担当ノードのキュー

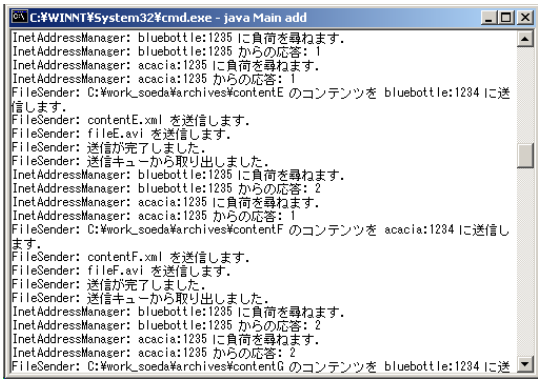
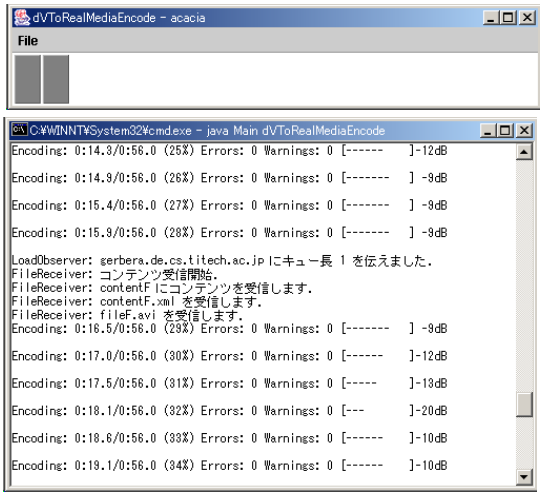


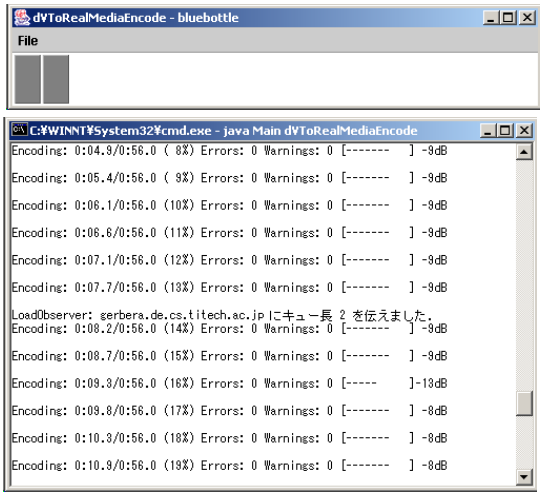
図 12　gerbera からのコンテンツの送信

この際，gerbera はワークフローに追加されたコンテンツ contentF をエンコード担当のノードに送信しようとしている．この際の gerbera のコンソール画面を図 12 に示す．図の 11 行目から 14 行目で，エンコードを担当する 2 つのノードに負荷を問い合わせている．

bluebottle と acacia から，それぞれ 2，1 という応答があったため，10 行目で，キュー長の小さい acacia に contentF を送信している．これにより bluebottle のキュー長は 1 つ増える．contentF 送信後のエンコード担当ノードの状態を図 13 に示す．contentF が送信されたため，acacia のキュー長が 1 から 2 に変わっている．この間に，bluebottle は現在のエンコード処理が終わっていないため，キュー長は変化していない．



(a) acacia



(b) bluebottle

図 13　gerbera が contentF を送信したあとのエンコード担当ノード

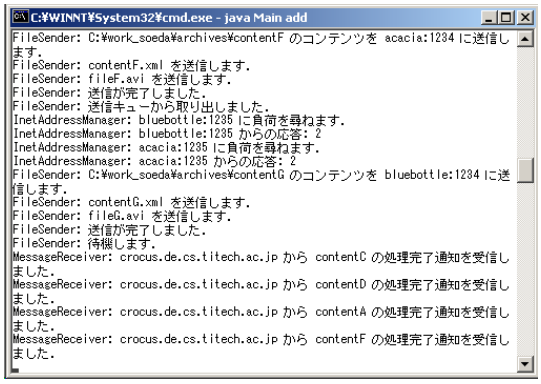


図 14　crocus からの処理完了通知

アーカイブが終了した時点で，処理の完了通知を送信する．ここでは，処理の完了通知の送信先を指定していないため，コンテンツを追加したノードに通知を送る．図 14 のように，アーカイブを担当するノード crocus から gerbera に処理完了通知が来る．

8.　まとめと今後の課題

8.1　ま　と　め

本稿では，XML で記述したメタデータを用いて処理ノードを自動判別する手法を提案し，その適用例として，映像コンテンツ管理ワークフローを考え，その実現方法について述べた．さらに，試作システムを実装し，映像コンテンツの処理を行うワークフローを構築する実験を行った．

まず，映像コンテンツ管理のプロセスについて述べ，ワークフローが自動化すべき処理を示した．コンテンツ管理に用いるメタデータに記述すべきコンテンツのメタ情報を示した．作業を分散管理するノードの機能について述べ，コンテンツの処理に最適なノードを選択する方法を示した．これにより，複数のノード間で効率よく映像コンテンツ管理を行うことができる．さらに，ノードがどのように処理を行うかを例を挙げて示した．また，ノードでの処理を拡張し，処理の分岐・待ち合わせを実現する方法を述べた．そして，実験システムの構成を示し，Java を用いた実装により，提案したワークフローが実現可能であり，提案手法が有効であることを確認した．

8.2　今後の課題

本稿では，ノードに例外が生じた際の動作を考慮していない．そのため，ワークフローが操作しているファイルのロックや，ノードに障害が発生した際の復旧方法，人間が処理のやり直しを命じた際の動作について検討する必要があると考える．さらに，ノードの動作が単純であるため，別々にワークフローに追加されたコンテンツを組み合わせて処理を行うことができるような，複雑な待ち合わせ処理の手法を検討する必要があると考える．

また，実装では処理を待つコンテンツのキュー長のみでノードの負荷を測定していたが，自動化できる処理については，CPU の使用率やクロック数などを考慮に入れ，負荷の測定方法を改善する必要がある．さらに，現在実装した処理は，アーカイブのほかには 2 種類しかないため，上記の機能を実装するとともに，自動化できる処理や，人間が介在する処理などのプロセスを増やし，より多機能なワークフローにする必要があると考える．

謝　　辞

本研究の一部は，文部科学省科学研究費補助金基盤研究（14019035）の助成により行われた．

文　　献

[1] “The Workflow Management Coalition”. http://www.wfmc.org/.
[2] “The MPEG Home Page”. http://www.cselt.it/mpeg/.
[3] “TV-Anytime Forum”. http://www.tv-anytime.org/.
[4] “Material Exchange Format (MXF) File Format Specification (Standard)” (2002). http://www.g-fors.com/publicdocs/pdf/mxf10-p2-format.pdf.