メディアフレームワークに基づくコンテンツ保護処理方式と自律制御型情報カプセルの実現

阿部剛仁[†] 谷口展郎[†] 森賀邦広[†]
塩野入理[†] 櫻井紀彦[††]

知的財産として価値を持つディジタル情報（コンテンツ）を流通させる際に，流通の各段階において，コンテンツの提供者が意図しない不正な利用を防ぐための機能が求められており，様々なコンテンツ保護方式が提案されている．本論文では，メディアフレームワーク（MF）の特徴を整理したうえで，MF を利用した拡張性の高いコンテンツ保護処理の実現方式について提案する．またそれらの機能を，コンテンツの利用条件を自律的に判断・制御する情報カプセルと組み合わせる方法について述べ有効性の検討を行う．さらに実際の MF を用いたプロトタイプ実装について報告し，動作検証と解決すべき課題等について考察する．

Content Protection Mechanism Based on Media Framework and an Implementation for Autonomous Information Container

Takehito Abe,[†] Noburou Taniguchi,[†] Kunihiro Moriga,[†]
Osamu Shionoiri[†] and Norihiko Sakurai[††]

Recently, there are emerging needs for technology to protect valuable digital content from unfair use in networked information sharing environment. In this paper, we propose a mechanism for content protection based on media framework (MF). We discuss features of MF and define an abstract model of it. Then we analyze the model and propose several content protection techniques that exploit extensibility of MF. In addition, we present a method to apply the techniques for autonomous information container (AIC). Prototype implementation of AIC supporting MF content protection mechanism showed feasibility of the proposed method.

1. はじめに

DSL 等の高速ネットワークの急速な普及により，一般ユーザによるネットワークを利用したマルチメディアデータの流通が拡大している．これらのマルチメディアデータを再生する際，Microsoft Windows や MacOS 等のオペレーティングシステムでは，メディアデータを取り扱うための基盤システムとして，DirectShow [1]，QuickTime [2] と呼ばれるメディアフレームワーク（以後,「MF」と呼ぶ）が用いられている．MF は，個々のメディア処理機能を個別モジュールの集合体として実装することで，多様なメディアに対応する柔軟性と拡張性を確保するとともに，利便性の高いメディア処理 API を提供し，メディアフォーマットや符号化に関する高度な知識がなくても，プレーヤ等のアプリケーションを容易に作成可能にするという利点を持つ．一方で，ディジタル情報は複製が容易で劣化がなく，保管や輸送のコストがきわめて低いという特徴を持つため，著作権等を有する経済的・社会的に価値のあるディジタル情報（以後,「コンテンツ」と呼ぶ）の流通においては，それらの不正な利用を防止し，知的財産権を保護する仕組みが必要とされているが[3)~5)]，MF にはコンテンツ保護の仕組みがないため，コンテンツの保護処理については別途考慮する必要がある．

我々はこれまでに，コンテンツ保護の方式として，自律制御型情報カプセル Matryoshka を提案している[6)~8)]．自律制御型情報カプセルは，暗号化コンテンツデータ，コンテンツメタ情報，利用許諾条件，暗号復号情報，メディア処理機能，カプセル内データへのアクセス手段等をカプセルへ内包し，外部からの利用要求に対して，種々の情報を基に自律的に利用をコントロールできるという利点を持つ[9),10)]．しかしなが

† NTT サイバーソリューション研究所
NTT Cyber Solutions Laboratories
†† NTT 情報流通プラットフォーム研究所
NTT Information Sharing Platform Laboratories

ら，これらの自律制御型情報カプセルでは，メディアの処理機能を内包する必要があるため，各種メディアへ対応する際の実装の煩雑さや，カプセルサイズの肥大化による伝送効率の低下等の問題がある．MF の利点を生かしたコンテンツ保護処理方式をカプセル化技術と組み合わせることにより，効率が良く柔軟性の高い自律制御型情報カプセルを実現可能であると考えられる．

本論文では，まず MF にメディアデータ保護処理機能を組み込む方式（以後,「MF 拡張コンテンツ保護方式」と呼ぶ）を提案し検討を行う．その後，MF 拡張コンテンツ保護方式を自律制御型情報カプセル化技術と組み合わせ，種々のメディアフォーマットへの容易な対応を可能とするコンテンツ保護システムを提案する．また，実際にプロトタイプを作成して実現性の確認と評価を行った結果についても述べる．

以降 2 章において，MF の特徴を整理したうえで MF においてコンテンツ保護機能を実現する方式を検討し考察を行う．3 章では，前記 MF 拡張コンテンツ保護方式を自律制御型情報カプセルにおいて実現する方法を提案する．続く 4 章において，実際の MF を用いたプロトタイプ実装を紹介するとともに課題や展望について考察し，最後にまとめを行う．

2. MF 拡張コンテンツ保護方式

2.1 MF の特徴

メディアフレームワーク（MF）は，ある端末環境において，動画像・音声等からなるマルチメディアストリームデータを処理するためのシステム全体，もしくはハンドリングするための API 群である．代表的 MF には，Microsoft 社の DirectShow，Sun Microsystems 社が提供する Java Media Framework [11]，Apple 社の提供する QuickTime 等がある．こうした MF を用いることで，メディアフォーマットや符号化に関する高度な知識がなくても，簡単な API の操作によってメディアの再生や制御を行うアプリケーションを作成することが可能になっている．MF の機能については，大きく分類すると以下のように整理することができる．

(1) キャプチャデバイスやファイルシステムから読み込んだデータを，抽象化した入力オブジェクトとし，取扱いを単純化する．
(2) MPEG-1，AVI，DV 等のメディアフォーマット固有の処理（Encode/Decode，Multiplex/Demultiplex 等）を隠蔽し，単一の手順で様々なフォーマットに対応する．
(3) メディアストリーム中のビデオ・オーディオ等個々のトラックに対し時刻管理，同期処理を行う．
(4) 端末のサウンドボード・グラフィックチップ等ハードウェア環境や，個々のデバイスドライバの制御方法等を隠蔽し，効率良くメディアのレンダリングを行う手段を提供する．

MF では上記メディアの処理機能をモジュール化し，それらモジュールを組み合わせることで，メディア処理フロー全体を網羅する．MF を利用する最大の利点は，アプリケーションが端末の様々なハードウェア環境，多種多様なメディアフォーマットごとに，独自の処理機能を作成することなく，簡単にメディア処理を行うことが可能な点にある．これにより新たなハードウェアやコーデック（Codec: Encoder/Decoder）が登場した場合も，MF が対応すればアプリケーションはほとんど変更なく対応可能となる．

2.2 抽象化 MF モデル Abstract Media Framework（AMF）の定義

MF 拡張コンテンツ保護方式について検討を進めるにあたり，実在する個別の MF について各々議論を行うと煩雑になるため，MF として必要な基本機能を含む抽象的化モデル Abstract Media Framework（AMF）を定義し，以後このモデルを用いて提案方式を説明することにする．以下に AMF で定義した各モジュールの機能と構成を説明する（**図 1** 参照）．

Media Resource：キャプチャデバイス，ファイルシステム，ネットワーク等の情報資源を表す．
Source Object：Media Resource で示されたマルチメディアデータを抽象化したデータストリーム．
Processor：マルチメディアデータをパースする Multiplexer（Mux）/Demultiplexer（Demux）と，Encoder/Decoder の機能を持つ．データの加工編集機能を持つことも可能である．
Renderer：Processor から受け取ったデータを，Output Device で表示・出力する．

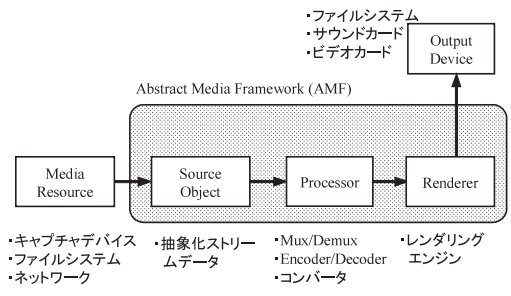

図 1 抽象化メディアフレームワーク
Fig. 1 Abstract media framework (AMF).

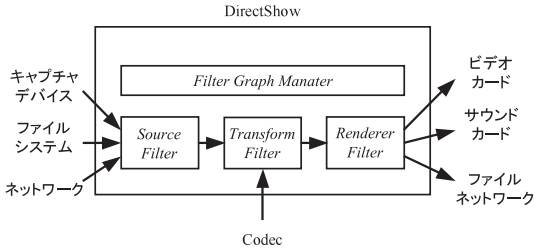

図 2 DirectShow のアーキテクチャ
Fig. 2 DirectShow.

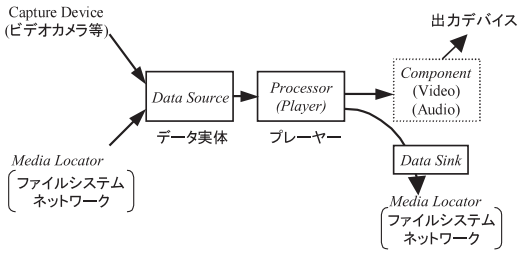

図 3 Java Media Framework のアーキテクチャ
Fig. 3 Java Media Framework (JMF).

Output Device：ディスプレイやスピーカ等の出力装置．ファイルシステムやネットワークも含めることができる．

メディアデータは MF 内で適切に同期を保たれながら図中矢印の順序で処理される．Media Resource により示されるコンテンツデータは，抽象化されたデータオブジェクト Source Object として準備され，Processor によって読み込まれる．Processor で必要なメディア処理が行われた後 Renderer へデータが受け渡され，各種デバイスで表現可能なデータ形式に変換されて Output Device へ出力される．

実際の MF を AMF と比較し，モデルの相違点や妥当性について検証する．**図 2** は Microsoft 社が提供する DirectShow のアーキテクチャである．メディアデータの取得は，ファイルシステムやネットワーク等，取得先ごとに用意される *Source Filter*で行われ，*Transform Filter*に受け渡される．*Transform Filter*にはパーサ（Mux/Demux），デコンプレッサ（デコーダ），フォーマットコンバータ（エンコーダ＋α）等の種類があり，各々でメディアの処理が行われる．*Renderer Filter*では，受け取ったデータをモニタやスピーカを通して再生する．これらを AMF のモジュール構成と比較すると，*Source Filter*が Source Object に，*Transform Filter*が Processor に，*Renderer Filter*が Renderer に対応し，AMF のモデルでうまく表現することができる．

図 3 は Sun Microsystems 社が提供する Java Media Framework（JMF）のアーキテクチャである．JMF では Capture Device による入力もしくは *Media Locator*によって示されるロケーションより得られるコンテンツデータから，*Data Source*と呼ぶ抽象化したデータオブジェクトを作成する．*Processor*は，*Data Source*からデータを読み込み，メディア処理を行って出力デバイスへ送付する．AMF のモジュール構成と比較すると，*Media Locator*が Media Resource に，*Data Source*が Source Object に，*Processor*が Processor に対応できる．ただし Renderer については若干説明を要する．Audio/Video データを出力する場合，Renderer 相当の機能は JMF として提供されておらず，Java 標準ライブラリの *Component*がこれを担う．また，ファイルシステムに出力する場合は，*Data Sink*と呼ばれる書き込み用のデータオブジェクトが Renderer に相当する．

2.3 AMF における MF 拡張コンテンツ保護方式

コンテンツ保護方式について，ここでは暗号化したコンテンツのデータストリームに対し，MF を用いた符号復号処理（デコード）等のメディア処理を行い，その過程において暗号復号処理を行って，端末でコンテンツを正常に再生するための方式について述べる．方式の提案と検討は前節の AMF に対して行い，Media Resource はファイルシステム，入力データは映像と音声をそれぞれ 1 トラックずつ含むマルチメディアストリームと想定する．コンテンツ再生のため，AMF はいずれかのモジュールで暗号の復号情報を取得し，データの復号処理を行う必要がある．AMF において実際にデータ実体を処理対象としている以下の 3 つのモジュール，Source Object，Processor，Renderer において，モジュールの内部もしくはデータ入出力過程のいずれかで処理を行う方式を述べる．

(1) Source Object 改造型（S 型）

Source Object は Processor から見ると，単純なストリームデータとして扱われる．Processor から指定されたデータ位置（ポインタ）とサイズに応じて必要なデータを切り取り出力する機能を持つ．Source Object 改造型の保護方式では暗号化データを入力とし，データの取得要求に対して，暗号復号情報を用いて復号化したデータを出力するものとなる．

(2) Processor 改造型（PC 型，PP 型）

Processor は MF におけるメディア処理の中核モジュールであり，内部ではさらに様々なサブモジュールが連携して処理を行う．**図 4** に一般的な Processor の構成例を示す．Processor 内部では，Source Object から読

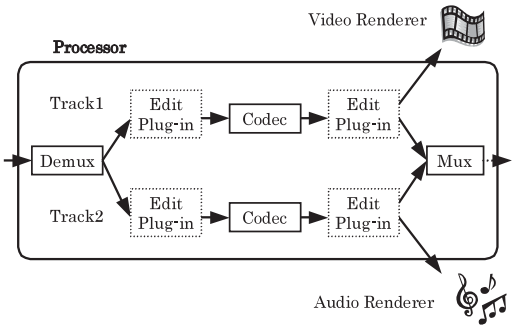

図 4 AMF における Processor 構成例
Fig. 4 Processor in AMF.

み込んだコンテンツデータを Demultiplexer（Demux）で単一のメディアストリームに分解し，Codec を通じて符号復号化処理を行い，そのままの単一メディアストリーム単位か，もしくは再び Multiplexer（Mux）で合成したマルチメディアストリームとして Renderer に出力する．コンテンツに何らかの編集処理が必要な場合は，Codec の前後に何らかのプラグインを挿入することも可能である．

Processor の改造を行う方法としては，内部サブモジュール自体の Codec に改造を加える方法（PC 型）と，その前後に新たなサブモジュールを追加する方法（PP 型）の 2 通りが考えられる．

PC 型は Demux から暗号化状態の単一メディアデータを入力とし，暗号復号とメディア符号復号化を行った後に，Renderer が表現可能なメディアデータを出力する Codec モジュールとして実装される．

PP 型は通常 Demux と Codec の間に挿入する方式をとり，PC 型同様暗号化状態の単一メディアデータを入力として受け取り，暗号復号を行った後，復号化したデータを出力する Plug-in モジュールとして実装される．

(3) Renderer 改造型（R 型）

Renderer に入力されるデータは，すでに Processor でメディア固有の処理が行われた後のデータであり，動画であれば 1 フレーム単位でのビットマップデータの状態になっている．つまり，このモジュールで暗号復号する場合は，データは暗号化された状態のまま Processor を通過する必要がある．標準の Processor で処理を可能にするためには，コンテンツの暗号化等の処理を，メディアの符号化規則にはまったく違反することなく行わなくてはならない．このような方法で画像・音声メディアを隠蔽・保護する仕組みとして，一般的にスクランブルと呼ばれる手法[12]が適用可能である．R 型では，スクランブルされたメディアデータを入力とし，スクランブル解除を行って Output Device に元の状態のメディアデータを出力する機能を持たせる．

2.4 MF 拡張コンテンツ保護方式の評価

前節において提案した MF 拡張コンテンツ保護方式について，以下の（a）〜（f）の項目について評価し有効性を検討する．

（a）上位アプリケーションのへの影響

MF を用いる場合，コンテンツを利用する上位のアプリケーションは，規定のメソッドにより種々のメディアデータに対して統一的に操作を行うことができる．MF に保護機能を持たせることにより，これらの処理手順に影響が及ばないことが重要である．この点ではいずれの方式ともモジュール内で機能が完結しており，通常の MF と同様に扱うことができる．

（b）MF 標準モジュールの流用性

MF では，標準で各種メディアに対応する多くのフォーマットパーサ・コーデックの処理モジュールが準備されており，追加のモジュールが開発元より随時提供される場合もある．MF 拡張により保護機能を追加する場合，これらのモジュールを可能な限り流用することで，種々のフォーマット・コーデックへの対応を容易にすることが可能である．S 型，PP 型，R 型では，暗号の復号機能をメディア固有の処理を行う（サブ）モジュールと独立して実装しているため，既存のフォーマットパーサ・コーデック（サブ）モジュールを利用することができる．PC 型は必要な種類のコーデック機能を，自ら作成しなければならない．

（c）モジュール改造の容易性

MF のモジュールへ暗号復号機能等の必要な改造を行う際には，種々のフォーマット・コーデックに固有の知識を必要とせず，容易に汎用性の高いモジュールを作れることが望ましい．S 型，R 型はコンテンツのフォーマット・コーデックに影響なく，それらの知識も必要ない．PP 型はモジュールの作成には前述の知識を必要としないが，挿入場所・方法を検討するうえでフォーマットに関する知識が必要な場合がある．PC 型ではコーデック機能についての知識が必要になる．

（d）コンテンツデータ保護処理方法の柔軟性

提供者らがコンテンツに保護処理（暗号化）を行う場合は，コンテンツの種類や利用者の想定等に応じて，暗号アルゴリズムや強度等の暗号化条件を柔軟に選択できることが望ましい．S 型は，コンテンツに対して一般的なストリームデータの暗号化手法がそのまま利用できるため，暗号化強度や頻度を自由に選択して処理することが可能である．PP 型の場合もストリーム

データに対する自由な暗号化が行えるが，Demux は暗号化状態のコンテンツデータを各トラックに分離する必要があり，データの暗号化は各トラックデータに対して行う必要がある．逆に考えると，コンテンツのトラックごとの選択的暗号化を望む場合にはこの型が有効ともいえる．PC 型の場合は，コーデック固有のデータ処理単位で暗号場所や頻度を決定する必要がある場合もある．R 型は前節でも述べたとおり，画像・音声等メディアの種類ごとに，符号化メディアの復号後に解除可能なスクランブル処理が行われていなくてはならない．動画メディアは符号化過程で非可逆な圧縮処理が行われる場合がほとんどであるため，様々な符号化・逆符号化過程を経ても復元可能なスクランブル手法でコンテンツデータの保護処理を行う必要があり，その方法は非常に限定される．

(e) コンテンツデータ保護処理の容易性

コンテンツデータの暗号化を行うモジュールを作成する際，各種フォーマット・コーデックに固有の知識を必要とせず，できるだけ容易に作成できることが望ましい．S 型はどのような種類のコンテンツに対しても同様の方式で暗号化が可能である．PP 型は (d) で述べたようにトラック単位での暗号化になるため，暗号化の際もトラックを分離する必要がある．MPEG-2 を例に考えると，MPEG-2 の Transport Stream（MPEG2-TS）もしくは Program Stream（MPEG2-PS）に含まれるパケットのうち，Video Stream，Audio Stream のデータのみを選択的に暗号化しなくてはならず，フォーマットに関する知識が必要になる．ただし MPEG-2 等の場合では，市販の編集ツールも充実しており，フォーマット規則に従いデータを選択することは比較的容易である．PC 型では，暗号化ツールにおいても対応するコーデック等が必要であり，特定フォーマット・コーデックごとに対応する必要がある．R 型においても，スクランブルを行うためには，各フォーマット・コーデック対応の専用ツールを作成する必要がある．

(f) 不正モジュール置換による MF の改ざん耐性

MF は各モジュールの機能を分離し，接続方法を明確に定義にしているが，このことはアプリケーション開発の面では大変有利である半面，コンテンツ保護機能に対するクラッキング対策面からみると脅威となる．個別のモジュールが置換されることにより，MF 自体が意図しない不正な処理を行うよう改ざんされる危険性を持つ．

それらの対策としては，データを受け渡す MF のモジュール間で相互認証を必須としたり，利用するモジュールすべてに署名を行い，第三者的なモジュールで利用する MF モジュールの正当性を事前に認証したりする方法が考えられる．

前者の方法では MF 全体のセキュリティポリシを見直し，モジュール間の接続時に認証を行うよう仕様を変更する必要があり，さらにモジュール間通信時の認証処理は MF のパフォーマンス低下させる可能性もある．

一方後者の方法は，MF のモジュール認証機能を事前に提供し，MF 利用時には確実に認証機能を実行させる仕組みを提供する必要がある．

また，OS や MF のセキュリティポリシを変更せず不正 MF への対策を行う簡易な方法として，MF を利用するアプリケーションが，MF を構成するダイナミックリンクライブラリやクラスファイルのサイズやハッシュ値を検査し，所定の値と比較し認証することが考えられる．しかしこの方法では，ライブラリ等改ざんをすべて検知できるわけではなく，バージョンの違い等で認証に失敗するという問題も残される．

パーソナルコンピュータにおけるコンテンツ保護機能の実装では，ハードウェア，OS，VM（Virtual Machine）等の各階層でハッキングの可能性があり，対策は利便性・柔軟性・効率性とのトレードオフになっているが，MF についても同様であり，状況に応じて適切な方法を選択する必要がある．

MF 拡張コンテンツ保護の各方式による安全性の違いを比較した場合，モジュール置換による MF の不正な改ざんの可能性を考慮すると，MF 内で可能な限りコンテンツデータが暗号化状態で処理された方が耐性は高いと考えられる．その点においては，S 型，PP 型，PC 型，R 型の順で後者ほど耐性が高くなっていく．たとえば，S 型で入力データを復号した場合，Processor をデータクラッキング用のモジュールに置換されてしまうと，暗号が復号された状態で取り出される危険性があるが，R 型では，データが通過する最後のソフトウエアモジュールであり，基本的に置き換えによるクラッキングの可能性は低い．

以上，各 MF 拡張コンテンツ保護方式についての評価を**表 1** にまとめる．保護機能を MF へ組み込む大きな利点の 1 つは，動作環境に準備された各種フォーマットパーサ，コーデック，Audio/Video デバイスハンドラを活用し，セキュアなコンテンツ利用アプリケーションを簡易に構築可能な点にある．そのような観点から実用性を勘案すると，項目 (b),(c) の評価が低い PC 型は，特定コーデックの利用を想定し，MF のコーデックに対して十分な知識を有する一部のケースを除き，有効性は限られるといえる．一方，R 型は

表 1 各コンテンツ保護方式の評価
Table 1 Evaluation of content protection techniques.

		評価観点					
		(a)	(b)	(c)	(d)	(e)	(f)
Source Object 改造型		○	○	○	◎	◎	△
Processor 改造型	Codec	○	×	×	△	×	○
	Plug-in	○	○	△	○	△	○
Renderer 改造型		○	○	○	×	×	◎

(a) 上位アプリケーションのへの影響
○：標準の MF と利用手順が同様で，コンテンツフォーマット等によりアプリケーションの実装方式に影響なし．

(b) MF 標準モジュールの流用性
○：一般的に実装規模が大きく，多種用意する必要があるフォーマット・コーデックのモジュールをそのまま流用可能．

(c) モジュール改造の容易性
○：モジュール作成の際に各種フォーマット・コーデック固有の知識を必要としない．
△：一部フォーマットの知識を必要とする．

(d) コンテンツデータ保護処理方法の柔軟性
◎：暗号アルゴリズム・強度，暗号化範囲をコンテンツデータ全体について自由に設定可能．
○：暗号アルゴリズム・強度，暗号化範囲をトラック単位で自由に設定可能．
△：暗号アルゴリズム・強度のみ自由に設定可能．
×：暗号化の方式はかなり限定される．

(e) コンテンツデータ保護処理の容易性
◎：コンテンツデータの暗号化を行うモジュールの作成において，各種フォーマット・コーデック固有の知識を必要とせず，共通モジュールとして作成可能．
△：各種フォーマットごとの知識・処理が必要．

(f) 不正モジュール置換による MF の改ざん耐性
◎：モジュール置換による不正が不可能．
○：置換は可能だが，不正なモジュールの作成が困難．
△：OS や MF 処理方式の知識を有する者による不正な置換が可能．

(b),(c) の評価は高いものの，各コーデックに対応したスクランブル方式を準備する必要があるため，(d),(e) の評価が低くなり，スクランブル技術を持つ特定コーデックを利用する場合のみ有効と考えられる．

MF 拡張方式の利点を生かす保護方式としては，S 型，PP 型が有効であり，実装の容易さを重視する場合は前者が，モジュールの改ざん耐性を重視する場合は後者が適する．また，項目 (d) においては，コンテンツデータ全体を対象として制御柔軟度の評価を行っているが，トラックごとのデータの制御を望む場合には，PP 型の方が実装が容易になる場合もある．

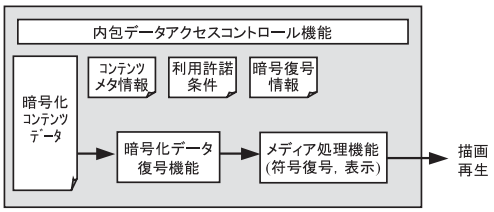

図 5 自律制御型情報カプセル
Fig. 5 Autonomous information container.

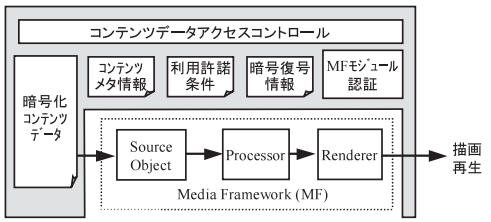

図 6 MF 拡張コンテンツ保護方式利用カプセル
Fig. 6 Autonomous information container based on Media Framework.

3. MF 拡張コンテンツ保護方式を利用した自律制御型情報カプセル

3.1 MF 拡張コンテンツ保護方式の自律制御型情報カプセルへの適用

図 5 は自律制御型情報カプセルの構成を示す図である．自律制御型情報カプセルには，暗号化コンテンツデータのほか，アクセスコントロール機能，コンテンツメタ情報，利用許諾条件（利用回数，利用者期間，利用者限定，利用端末限定等），暗号復号情報，およびデータの暗号復号機能とメディア関連処理機能が内包されている．カプセルの再生時には，暗号復号等のデータ保護処理と，符号復号等のメディア処理を行ったうえで，コンテンツを画像や音声として出力する．

自律制御型情報カプセルにおいて，MF 拡張コンテンツ保護方式を利用するための基本的な構成方法を **図 6** に示す．暗号復号処理とメディア関連処理は MF 側に配置し，カプセルには主にコンテンツ固有の情報のみを内包する．ただし 2.4 節で論じたように，安全性を高めるための MF モジュール認証機能等を新たに内包する必要がある．カプセル内のコンテンツデータは MF で処理され，カプセルが提供するユーザインタフェース（UI）を通して外部に出力される．

カプセル内のデータを読み込む方法としては 2 通り考えられる．1 つはカプセルに内包するデータの一部をファイルとして一時的に展開し標準 Source Object から読み込む方法である．もう 1 つはカプセルに内包

するデータを読み込める Source Object を作成する方法である．

前者の読み込み方法では，Source Object の処理機能にはまったく改変が必要ないため，2 章で述べた PC 型，PP 型，R 型の場合は，全体の改造が Processor/Renderer の特定のモジュールに限定されるという利点がある．ただし，コンテンツファイルのサイズが巨大である場合は，一時的ではあるものの，動作環境のファイル資源を無駄に消費し，書き込み操作等のアクセスにも時間を要する．また，コンテンツファイルを暗号化し，アクセス方法を隠蔽しているとしても，カプセルからコンテンツデータをファイルとして外部に出力することは，セキュリティの観点からも問題がある．

一方後者では，Processor がカプセルに内包された状態のコンテンツデータを直接読み込むことが可能である．コンテンツデータは必要に応じてメモリ上に展開されるのみであり，端末資源の利用効率や処理速度の点で優れている．また，コンテンツデータを端末のファイルシステムに書き込むことがないため，クラッキングに対する耐性についても優位である．MF 拡張コンテンツ保護方式として S 型を用いる場合は，ともに Source Object への改造であり，カプセルデータからの特殊なデータ読み込みと暗号復号処理機能を分離が困難な同一モジュールにすることは，安全性の観点からも好ましい．

3.2 自律制御型情報カプセルの設計方針

図 5 に示すように，通常の自律制御型情報カプセルではメディア処理機能すべてをカプセルに実装する必要があるため，種々のメディアフォーマットへの対応には実装コストが高く，カプセルサイズが肥大化するといった問題があった．たとえば先に述べた Matryoshka では，メディアの種類に応じてカプセルに内包する再生用プログラムを独自に作成する必要があり，初期プロトタイプの Matryoshka では，対応可能なメディアフォーマットは，MPEG Audio Layer3（MP3），Bitmap 画像（BMP），JPEG 画像，Text 形式に限られていた．これら音声や静止画メディアのコーデック等は比較的サイズが小さいが，映像を含むマルチメディアコンテンツのコーデックではサイズが肥大化する可能性が大きい．MF 拡張コンテンツ保護方式と連携する方式では，MF が種々のコンテンツに対するメディア処理機能を持つため，実装コストを下げ，またカプセルサイズを低く抑えることが可能になる．このとき，MF 拡張コンテンツ保護方式の拡張モジュールについては，環境にプレインストールしておくことも，カプセルに内包することも可能である．カプセルへ内包した場合，たとえば S 型では**図 7** のような構成になる．コンテンツ保護の側面からはカプセルに内包していた方がよい．しかし PC 型や R 型の場合は，必要な機能を考慮すると，モジュールの実装サイズ自体が大きくなりやすい．特に PC 型では内包するコンテンツの種類に応じて必要とするモジュールの数も増大することから，結果的にカプセルの肥大化につながる危険がある．最終的には MF の特性と MF 拡張方式の選択により，内包するかプレインストールかを判断すべきであろう．

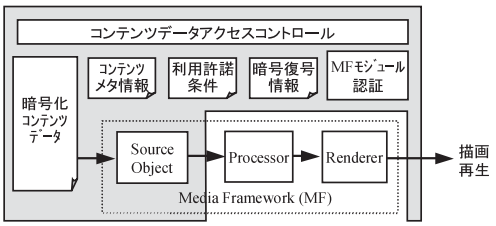

図 7 S 型保護方式利用カプセル
Fig. 7 Autonomous information container based on S-type content protection method.

自律制御型情報カプセルを MF 拡張コンテンツ保護方式と組み合わせる際に新たに考慮すべき点としては，2.4 節（f）で論じた不正 MF 対策がある．これについては MF モジュール認証機能をカプセル内に内包し，データを引き渡す際に必ず MF の認証を実行する仕組みを持たせる方法を提案する．これによりメディア処理過程における不正なアクセスを極力防ぐ効果が期待できる．

4. MF を利用した自律制御型情報カプセルの実装

4.1 JMF 利用自律制御型情報カプセル

4.1.1 JMF 利用自律制御型情報カプセルの実装

2 章で AMF を用いて提案してきた MF 拡張コンテンツ保護方式を適用し，JMF を用いた自律制御型情報カプセルのプロトタイプ実装を行った．MF の拡張方針は 2.4 節の評価結果から，S 型と PP 型を採用した．カプセル内のデータ読み込み方法に関しては，3.2 節で行った安全性とパフォーマンスの議論から，Source Object の対応による直接読み込み方法を用いた．以下の 2 種類の自律制御型情報カプセル（以後，J-Capsule と呼ぶ）を作成した．

J-Capsule(A)：PP 型（Demux-Codec 間実装）
J-Capsule(B)：S 型

図 8 に J-Capsule の構成を示す．カプセル内の各

図 8 JMF を利用した自律制御型情報カプセル
Fig. 8 Autonomous information container based on JMF.

種処理手段はすべて Java のクラスとして実装している．内包する情報は，Java クラスファイル，動画コンテンツデータ，コンテンツメタ情報である．クラスファイルには，コンテンツの暗号復号と表示を行う表現手段，利用の可否を判定する利用制御手段，カプセルの動作を管理する制御手段を有し，メタ情報には利用制約条件等の情報が記録されている．これらの情報を含むファイルを Java Archive（JAR）形式でアーカイブすることによってカプセルを生成する．カプセルは Java のアプリケーションとして Java Virtual Machine（JavaVM）上で動作する．コンテンツメタ情報およびコンテンツデータ自身には，結合化と暗号化処理を行い，一連のデータ列としてカプセル内に格納する．このデータ列は，外部からは JAR 内の 1 ファイルとして認識されるが，その内部には一種のファイル内ファイルが存在し，個々の情報へのアクセスは，カプセルに内包される特定のメソッドを通してのみ可能となる．ほかの手段によるアクセスを制限することで，データの漏洩と改ざんが行われることを防いでいる．その他のカプセルの機能として，利用制御手段に利用者認証（パスワード），利用期限・期間・回数限定機能を実装した．これらの利用動作制御は，コンテンツメタ情報に記された利用制約条件と，カプセルの利用履歴情報を基に，利用制御手段によって行われる．また，利用履歴情報はカプセル内に追加記録され，ユーザが行うカプセルの起動・終了，および動画再生・停止の操作を記録している．

図 9 は JMF の *Player*における処理フローである．

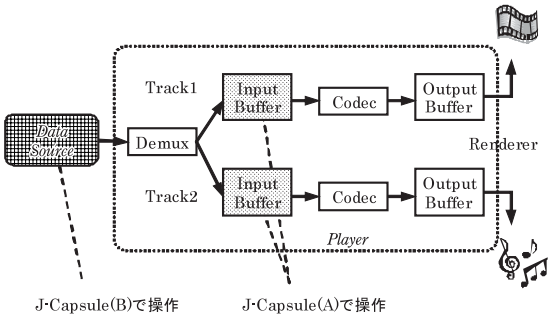

図 9 JMF における *Player*の処理
Fig. 9 Data flow of *Player* module in JMF.

*Player*モジュールは JMF において *Processor*モジュールのサブセット版であり，ファイルへの書き出し等の機能が削除されている．図中のドットのハッチングは，J-Capsule (A) で処理を行う JMF のサブモジュールを示す．JMF における *Player*は同期をとるために Codec の前後でバッファリングを行っているが，本カプセルは Codec 前の Input Buffer において，暗号の復号処理を行う機能を追加している．JMF において，Buffer へのデータの読み込みは *Player*によって行われる．Buffer 内のデータのどの位置からどのサイズを読み込むかは *Player*が決定する．図 9 の格子のハッチングは，J-Capsule (B) で処理を行うサブモジュールを示す．*Player*から発せられたデータの取得要求に対して，データが暗号化等の処理がされている場合には，内部で復号処理等を行ったデータを応答する機能を *Data Source*に実装している．

4.1.2 JMF 利用自律制御型情報カプセルの評価

以下の条件のコンテンツを使用して，J-Capsule (A), J-Capsule (B) の検証を行った．
Media Format：MPEG-1 System
Video：320x240 ピクセル，30fps
Audio：Layer-II，44 kHz，Joint-Stereo，112 kbps
File Size：5111 KB，再生時間：34 sec

各々のカプセルは，十分な処理性能を持つパーソナルコンピュータ（Pentium III 933 MHz，Windows2000）の環境においては，いずれも問題なく再生された．しかし処理性能の低いパーソナルコンピュータ（Pentium II 400 MHz，Windows98SE）の環境で再生したところ，J-Capsule (A) は変わりなく再生できたのに対し，J-Capsule (B) は一部でコマ落ちが確認された．これは以下のような理由で J-Capsule (A) の方式が，より処理効率が高かったためと考えられる．

PP 型では各トラックのメディアフォーマットに固有のバッファリング単位で暗号化した．これに対し，S 型では，フォーマットの Demux に必要な情報を取得

するときの小規模データ読み込みに対しても，暗号復号処理を行う必要がある．このことが復号処理効率の差を生み，低速環境で J-Capsule (B) のコマ落ちを招いたものと思われる．Java ベースの JMF は高い処理性能を必要とするため，高ビットレートのコンテンツを扱う場合には，十分な検証が必要であると思われる．

このほか J-Capsule (B) では，独自の *Data Source* モジュールに，データを引き渡す相手がデータ書き出し機能を持たない *Player*モジュールであることを認証してからデータを送り出すことで，復号後のコンテンツデータの流出を防止する機能を持たせている．JMF の *Player*モジュールは，画像・音声を出力デバイスへレンダリングする機能を有していることから，Renderer の置換によるクラッキングも不可能であり，この点では安全性が保たれている．一方で，Java のクラスファイルは逆コンパイル等によるリバースエンジニアリングに対する耐性が低く，JavaVM 等システムの情報についても開示されており，改ざん等による攻撃も考えられる．本論文のスコープ外であり詳細には言及しないが，実用化を行う際には，コードの難読化等 Java の持つ特性について十分な配慮を行うことが必要になると考えている．

また，MPEG-1 以外のフォーマットのコンテンツ（QuickTime Movie［Video：H263，Audio：ULAW］）を含む J-Capsule (B) を作成し，Windows，Linux，Solaris，MacOS X の各環境で，問題なく再生されることを確認した．これにより，JMF が存在する環境であれば，本実装のカプセルが，マルチフォーマット・マルチプラットフォームで利用可能なことが確認できた．

4.2 DirectShow を利用した自律情報型カプセルの実装

MF として DirectShow を利用したカプセルを作成し，Windows 環境で動作検証を行った．基本的なファイル構造は OLE の複合ファイルを利用し，保護方式は S 型とした．実装の詳細は JMF 利用の場合と重複が多いため割愛する．DirectShow は Windows の標準的な環境に含まれていることから，MF の準備に関して問題が少なく，OS のネイティブコードで動作するので性能が良い．本実装では，6 Mbps の MPEG-2 コンテンツにおいてもカプセルが正常に動作することを確認した．ただし MF 自体が Windows に特化された実装であるため，カプセルも Windows 環境でしか動作しない．実用化を行う際は，マルチプラットフォーム性，動作パフォーマンスを考慮して，適切な実装方式を選択する必要がある．

5. ま と め

本論文では，マルチメディアデータの処理基盤であるメディアフレームワーク（MF）の構成モジュールを拡張し，コンテンツ保護機能を実現する方式について検討した．MF の機能モジュールの一部に復号機能を持たせることで，暗号化されたコンテンツを利用時に復号し，MF の汎用性・拡張性を維持したまま，コンテンツの解読を可能とする 4 つの方式を提案した．各提案方式については，その利便性，開発容易性，安全性の観点から評価を行った．また，これを自律制御型情報カプセルと組み合わせた新しいコンテンツ保護システムを提案し，実際の MF である JMF，DirectShow 上で動作するプロトタイプの実装を行って，フィージビィリティを確認した．

今後はカプセル化コンテンツだけでなく，ストリーミング配信に MF を拡張したコンテンツ保護方式を適用する方法について検討したい．また，MF の基本思想は単独の端末に閉じている必要がなく，分散環境においても有効であり，機能モジュールの分散配置方法等の検討を行いたいと考えている．さらに MPEG-4/21 [13]で検討が続けられているコンテンツの保護機能に対して，どのような保護手法が適用可能で，有効であるかの検討を行い，提案していきたいと考えている．

参 考 文 献

1) http://www.microsoft.com/japan/msdn/library/default.asp?url=/japan/msdn/library/ja/jpdx8_c/ds/default.asp
2) http://developer.apple.com/quicktime/
3) 岸上順一：電子化知的財産とコンテンツ ID，情報処理学会研究報告，EIP-11, pp.1–4 (2000).
4) 田中克己：マルチメディアコンテンツのアクセスアーキテクチャ，*Proc. Advanced Database Symposium* (*ADBS'97*), pp.1–8 (1997).
5) 野方英樹：著作権仮技術と DAWN2001，2001 年映像情報メディア学会年次大会，S2-3, pp.415–418 (2001).
6) 谷口展郎，森賀邦広，久松正和，櫻井紀彦：マルチメディア情報ベースとその格納単位 Matryoshka，情報処理学会 DICOMO'99 シンポジウム論文集，p.207 (1999).
7) 加賀美千春，森賀邦広，塩野入理，櫻井紀彦：コンテンツ流通における自律管理を目的としたカプセル化コンテンツ Matryoshka，情報処理学会研究報告，DPS 97-18, pp.99–104 (2000).
8) 阿部剛仁，谷口展郎，塩野入理：Java を用いた動画配信カプセルの実装，情報処理学会マルチ

メディア通信と分散処理ワークショップ論文集, pp.229–234 (2000).
9) Kidawara, Y., Tanaka, K. and Uehara, K.: Encapsulating Multimedia Contents and A Copyright Protection Mechanism into Distributed Objects, *Proc. 8th International Conference on Database and Expert Systems Applications* (*DEXA'97*), pp.293–302 (1997).
10) Payette, S. and Lagoze, C.: Flexible and Extensible Digital Object and Repository Architecture (FEDORA), *Proc. 2nd European Conference on Research and Advanced Technology for Digital Libraries*, pp.41–59 (1998).
11) http://java.sun.com/products/java-media/jmf/
12) Fujii, H., Taniguchi, N. and Yamanaka, Y.: Scrambling Digital Image for Distribution Through Network, *Proc. PTC'96*, p.447 (1996).
13) http://www.cselt.it/mpeg/

(平成 14 年 12 月 27 日受付)
(平成 15 年 6 月 10 日採録)

(担当編集委員 増永 良文)

阿部 剛仁(正会員)

1993 年早稲田大学理工学部材料工学科卒業. 1995 年同大学院理工学研究科修士課程修了. 同年 NTT 入社. 以来, マルチメディアデータ保護技術, コンテンツ流通管理システムの研究開発に従事.

谷口 展郎(正会員)

1992 年東京大学工学部機械工学科卒業. 1994 年同大学院工学系研究科機械工学専攻修士課程修了. 同年 NTT 入社. 以来, 画像検索システム, ネットワーク情報流通システム等の研究開発に従事.

森賀 邦広

1991 年横浜国立大学工学部生産工学科卒業. 1993 年同大学院工学研究科博士課程前期修了. 同年 NTT 入社. コンテンツ流通管理・保護システム等の研究開発に従事.

塩野入 理(正会員)

1986 年 NTT 入社. 汎用コンピュータ統合化アーキテクチャ, コンテンツ流通管理, 権利保護等の研究開発に従事.

櫻井 紀彦(正会員)

1979 年早稲田大学理工学部電気工学科卒業. 同年日本電信電話公社(現 NTT)入社. ファイル記憶階層アーキテクチャ, マルチメディアデータベース, コンテンツ流通管理・保護システムの研究開発に従事. 電子情報通信学会, 映像情報メディア学会各会員.