データベースシステム 108－2
（1996．5．23）

ハイパーメディアシステムのための柔軟性の高いメディア管理データベース

佐伯 剛幸† 元木 誠‡

†NEC C&C 研究所 ‡NEC 第二製造 SI 事業部

本稿では、ハイパーメディア構築支援システムのメディア管理データベースにおいて、従来のハイパーメディアシステムの設計ではほとんど考慮されていなかったクラス構造の良構造化による柔軟性をデザインパターンを適用して実現したので報告する。本メディア管理データベースでは、ハイパーメディアシステムで特に必要とされる柔軟性である、メディア独立性、スケーラビリティ、ホットスポット形状への対応、多様な利用方法への対応を実現しており、利用者および他コンポーネントからの拡張やカスタマイズの要求に対してシステム全体を作りかえずに容易に対応することができる。

A Flexible Media Database for Hypermedia Systems

Takayuki Saeki† Makoto Motoki‡

†NEC C&C Research Laboratories ‡NEC 2nd Manufacturing Industries SI Division

In this paper, we report a flexible media databese for hypermedia systems. This flexibility is realized by good designs of software architecture using design patterns, the importance of which is little considered among hypermedia researchers until now. This media database deals with particularly important flexibility, such as media independence, scalability, the shape of hot spots and various usage. As a result, requests from clients for customization or extension are easily solved.

1 はじめに

我々は、ハイパーメディアアプリケーションを容易に構築および利用するための支援システムであるPCハイパーメディア(仮称)の設計開発を行なっている。PCハイパーメディアは、ひもとき[HKTH94]の機能をベースとして、そのソフトウエア構造を再構成して作り直し、PC上で実現したものである。ひもときの特長である柔軟なアプリケーション設計機能、多様な検索、容易な利用者インタフェースなどの機能を継承しつつ、スケーラビリティ、オープン性などを実現している。

本稿では、PCハイパーメディアの構成コンポーネントのうちのメディア管理データベースに対して、オブジェクト指向設計技術の一つであるデザインパターンを適用することにより柔軟性の高いソフトウエア構造を実現した点を中心に報告する。本稿では、メディア管理データベースにおいて特に必要とされる柔軟性として、メディア独立性、スケーラビリティ、ホットスポット形状への対応、多様な利用方法への対応の4つを上げ、それらをデザインパターンを適用して実現する方法について説明する。上記の柔軟性により、メディア管理データベースは利用者および他コンポーネントの開発者からの拡張やカスタマイズの要求に対して、システム全体を作りかえずに容易に対応することができる。

本稿の構成は以下の通りである。第2節でPCハイパーメディアの概要と必要とされる柔軟性について説明する。第3節ではデザインパターンの概要を記述する。第4節ではメディア管理データベースにおけるデザインパターンの適用による柔軟性の実現について述べる。第5節では関連研究について述べる。最後に、第6節でまとめを述べる。

2 PCハイパーメディア

本節では、PCハイパーメディアの概要説明と、メディア管理データベースのクラス構成および必要とされる柔軟性について述べる。

2.1 システム構成

PCハイパーメディアのシステム構成を図1に示す。PCハイパーメディアではシステム全体を、表示系(表示GUI、表示制御系)、ノードリンク管理系(ハイパーリンクエンジン)、メディア管理系(メディア管理データベース)の3つに分割しており、それらに編集ツールが結合されている。利用者は、PCハイパーメディア上にアプリケーションを構築する。

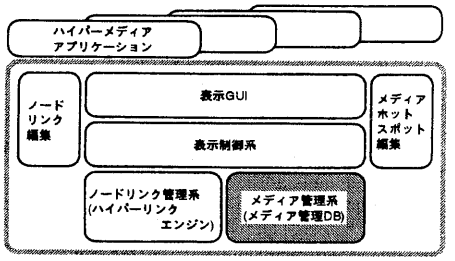

図 1: PCハイパーメディアのシステム構成

システム全体を、表示系、ノードリンク管理系、メディア管理系に分割しているため、それぞれの構造を独立に変更することができる。例えば、同じデータを使用していても表示方法を変更したり、表示方法は変更せずにデータを入れ換えたりすることを容易に行える。

PCハイパーメディアにおける典型的な処理として、図2に動画ホットスポットナビゲーションの処理流れを示す。矢印についている番号は処理の順序を表しており、概略的には、ポイントした座標からホットスポットを検出し、それに対応するメディアを取得して表示する処理を行なう。

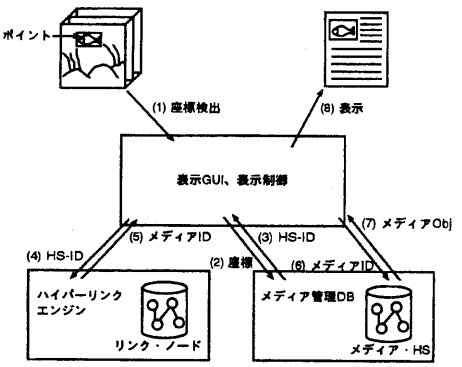

図 2: 動画ホットスポットナビゲーション

2.2 メディア管理データベース機能概要

PCハイパーメディアの構成コンポーネントとしてメディア管理データベースが提供する主な機能は以下の通りである。

1. メディア情報およびメタ情報(シーン、ホットスポット)の管理
2. ノードとメディアの対応関係管理
3. 表示制御からの要求に対するメディアのプレイ

4. 画面上でポイントされた座標からのホットスポット検出

2.3 クラス構成

メディア管理データベースのクラス構成をOMT [RBPEL91]の記法を用いて記述したものを図3に示す。メディア管理データベースは、大きく分けて**MDB(メディアデータベース)部**と**メディアライブラリ部**の2層に分割される。

MDB部は、**MDBマネージャ(MDBManager)**クラスと**MDB**クラスから構成され、メディアをクラスタリングするための構造を提供する。MDBマネージャはMDBオブジェクトの集合を管理する。ナビゲーション時には、表示系は必ずMDBマネージャを経由してメディア管理データベースにアクセスする。MDBはメディアオブジェクトの集合を管理するための単位であり、一つのMDBが一つのデータベースまたはファイルに対応する。

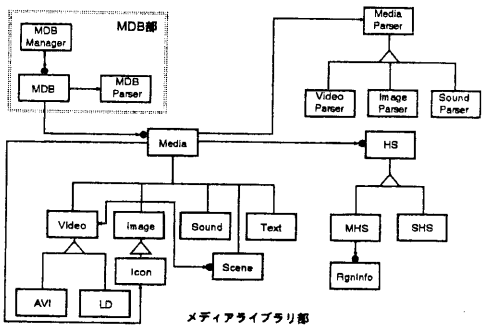

図3: メディア管理データベースのクラス構成

メディアライブラリ部は、**メディア(Media)**クラス、**シーン(Scene)**クラス、**ホットスポット(HS)**クラスから構成される。メディアは、メディアデータを管理するクラスである。メディアの派生クラスとして、動画(Video)、静止画(Image)、音声(Sound)、テキスト(Text)が存在し、動画にはさらにAVIとLD、静止画にはアイコン(Icon)という派生クラスが存在する。

メディアには、シーン、ホットスポットが関連付けられる。シーンは、メディアの派生クラスとして定義され、動画全体の内の注目する範囲を扱う。ホットスポットは、メディアを画面表示したときの注目している部分を示す領域情報を持ち、メディア上のボタンである。ホットスポットの派生クラスとして、動画ホットスポット(MHS)と静止画ホットスポット(SHS)が存在する。領域情報(RgnInfo)クラスは、動画ホットスポットが持つフレームごとのホットスポット領域情報を管理する。

2.4 必要とされる柔軟性

メディア管理データベースで必要とされる柔軟性として、メディア独立性、スケーラビリティ、ホットスポット形状への対応、多様な利用方法への対応を上げる。以下では、上記の柔軟性の説明およびその必要性を述べる。これらの柔軟性を実現するために、デザインパターンを適用する。

なお、以降では、メディア管理データベースのオブジェクトをアクセスする外部コンポーネントを総称してクライアントと記述する。

2.4.1 メディア独立性

ここでメディア独立性とは、すべてのメディアを同一インタフェースで扱えるようにし、どのメディアを扱っているかをクライアントに意識させないようにすることを指す。ハイパーメディアでは、動画、静止画、音声、テキスト、CGなど多種多様なメディアを扱う。また、利用者からの新規メディアの追加要求が往々にして発生する。この場合、メディア独立性が実現されていれば、新規メディアを追加してもクライアントは影響を受けずにそのまま使用できる。

2.4.2 スケーラビリティ

ここでスケーラビリティとは、扱うデータの規模への対応のことを指している。利用環境により扱うデータの規模は異なる。個人情報を管理するシステムやハイパーメディアを用いた利用者インタフェースなどを構築する場合には扱うデータは小規模で済むが、各種の情報を提供する電子××館やタウン情報システムなどでは大規模になる場合が多い。

このように利用する目的により異なる規模に対応するために、システムを永続化機構に依存しないようにし、必要に応じてOODB、RDB、ファイルなどさまざまな永続化機構から適正なものを選択できる必要がある。また、新規に、利用者が使用している永続化機構を追加する可能性もある。永続化機構への非依存性はプラットフォームに依存しないという点からも重要である。

2.4.3 ホットスポットの形状への対応

ホットスポットの形状は、付加する対象に応じて、矩形、円などを自由に選択できる必要がある。例えば、動画では被写体が交差するため、いくつかのホットスポットが重なり合うことが往々にして生ずる。この場合、ポイントに対する反応の精度を上

げるには、ホットスポットの形状が付加する対象を含みかつ余分な領域が少ないことが望ましい。

多様なホットスポット形状を実現する際には、形状に関する実装をクライアントに影響を与えずに柔軟に変更できるようにする必要がある。また、新規のホットスポット形状の追加を容易に行なえる必要がある。

2.4.4 多様な利用方法への対応

ハイパーメディアでは、通常、ナビゲーションのフェーズとオーサリングのフェーズがある。ナビゲーション時には、クライアントはメディア管理データベースのサブシステムを利用する必要はない。一方、オーサリング時には、サブシステムレベルのインタフェースを利用する必要がある。また、オーサリング時に必要なサブシステムレベルのインタフェースも、どのレベルのオブジェクトを編集するかにより異なる。

このように、クライアントとメディア管理データベースとの結合度が利用方法により異なる。この場合、それぞれの利用レベルに応じて、サブシステムの直接利用は妨げずにサブシステムの構成要素を隠蔽する単純なインタフェースを提供する必要がある。これより、システムが階層化され必要な機能を切り出せるようになる。

3 デザインパターン

デザインパターンは、システムの再利用性、柔軟性を高めるクラスの設計構造であり、パターン名、目的、適用可能性、解法、結果などをカタログ化している [GHJV94]。デザインパターンを適用する利点としては、設計の再利用によるソフトウェアの品質向上、設計経験の整理、設計者の間での共通の用語としての役割などをあげることができる。

デザインパターンは、適用範囲と目的の 2 つの観点から分類される。適用範囲にはクラスとオブジェクトがある。クラスに適用されるパターンは継承を、オブジェクトに適用されるパターンはコンポジションを使用することにより再利用性を実現している。

目的としては、生成、構造、振舞いがある。生成に関するパターンは生成されるクラス名、組合せ、責任分担、過程を隠蔽する。構造に関するパターンは構造の柔軟性を高めるクラス、オブジェクトの分割および合成の方法を規定する。振舞いに関するパターンは振舞いの柔軟性を高めるクラス、オブジェクトの分担および協調の方法を規定する。

4 デザインパターンの適用

本節では、2.4節であげたメディア管理データベースで必要とされる柔軟性をデザインパターンを適用して実現する方法について述べる。なお、以降のオブジェクト図では、クラスが提供するインタフェースに対する実装を、円から点線でつなげた四角の中に記述する。

4.1 メディア独立性

メディアが持つインタフェースを統一するためには、すべてのメディアの抽象クラスとなるメディアクラスを用意しそこでインタフェースを宣言し、具象サブクラスで実装を定義する。表示、再生などのインタフェースは、これにより統一できる。一方、この際に問題になるのは、メディアと関連を持つホットスポットオブジェクト生成のインタフェースである。

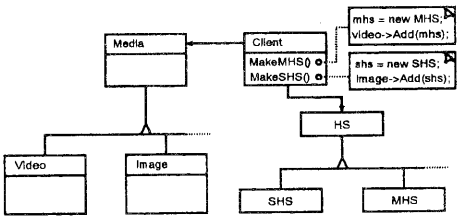

図 4: クライアント側で生成を行なう場合

ホットスポットオブジェクトの生成をメディアで行なう必要性は、クライアントで生成を行なうといくつかの問題が発生するためである。図 4では、クライアント側で、動画ホットスポットオブジェクトを生成する関数 MakeMHS() と静止画ホットスポットオブジェクトを生成する関数 MakeSHS() を定義している。この方法では、クライアント側が、動画の場合には動画ホットスポット、静止画の場合には静止画ホットスポットを生成するということを知らなければならないことや、それにより生成対象のオブジェクトや生成方法の変更がクライアント側に及ぶことなどが問題になる。さらに、新規メディアの追加の影響がクライアントにも及ぶ。

以上のことから、ホットスポットオブジェクトの生成のインタフェースをメディアクラスで統一した形で提供する必要がある。さらに、メディアごとに生成するホットスポットは変えなければならない。上記の問題点を解決するために、**Abstract Factory パターン**を適用する。Abstract Factory パターンでは、以下を実現することが可能である。

1. 生成するクラス名、組合せ、責任分担等の隠蔽

2. オブジェクト生成機構の局所化
3. 部品集合の容易な変更、サブクラスによる追加

適用後の構造を図5に示す。図5において、メディアクラスの具象サブクラスである動画クラスおよび静止画クラスが、ホットスポットを生成するfactory(工場)になっている。メディアクラスでは、CreateHS()というインタフェースを宣言しており、実際にどのクラスのオブジェクトを生成するかについての実装はサブクラスで定義する。この場合、動画クラスのCreateHS()では動画ホットスポットオブジェクトの生成および自分に対する追加を、静止画クラスのCreateHS()では静止画ホットスポットオブジェクトの生成および追加を行なう。

メディアオブジェクトを扱うエディタなどのクライアントでは、メディアの種類を意識する必要はなく、メディアクラスのサブクラスのオブジェクトに対してCreateHS()を呼び出せば適正なホットスポットオブジェクトが生成される。また、メディアクラスはファイルを解析するためのパーサクラスと関連を持つが、それらについてもメディアごとに適正なものが生成される。

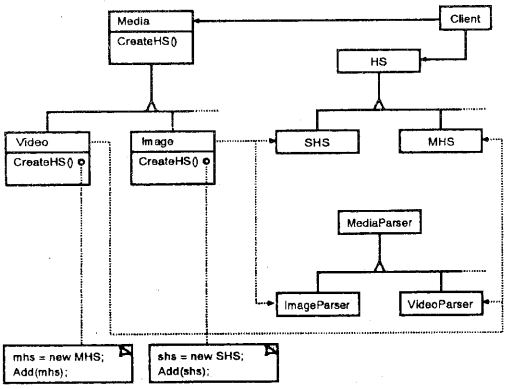

図 5: Abstract Factory パターンの適用

新規メディアが追加する場合には、メディアクラスまたはそれを継承するクラスのサブクラスとして定義する。さらに追加するメディアに固有のホットスポットが存在する場合には、ホットスポットクラスのサブクラスとして定義し、新規メディアクラスのCreateHS()でそのホットスポットが生成されるように定義する。このように、新規メディアの追加をクライアントに影響を与えずに容易に行なうことができる。

4.2 スケーラビリティ

スケーラビリティ実現のため、PCハイパーメディアでは、ファイルを使用しメモリに展開する場合とオブジェクト指向データベースPERCIO[TsKi94]を使用する場合の2種類の永続化機構を実現している。

スケーラビリティを実現する際には、永続化機構に依存する部分が広範囲にわたることが問題になる。データのロードやセーブなどの処理以外に、クラス定義、ポインタ型や集合型などの定義の方法、メモリアロケート演算子(new)の記述などが、永続化機構により異なる場合が多い。プラットフォームからシステムを独立にするためのデザインパターンとして前述のAbstract Factoryパターンがある。具体的には、factoryを使用して生成対象オブジェクトをプラットフォームごとに決められたクラスのオブジェクトに一括して切替えることにより、プラットフォームへの非依存性を実現する。

しかし、本件のようにクラス定義の記法が違うような場合にAbstract Factoryパターンを適用して動的な切替を実現しようとすると、構造が複雑になり過ぎてしまい保守性や効率の面で問題が生じる。このようなことから、永続化機構を動的に切替えることは行なわず、条件コンパイル文を用いたハードコーディングによる切替えを行ない、クライアントにはどの永続化機構を使用している場合でも同じインタフェースを提供するようにする。

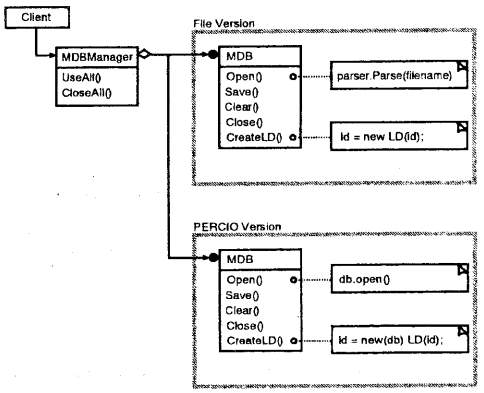

図 6: スケーラビリティへの対応

以上の方針に基づき実現した構造を図6に示す。データのロード、セーブ処理を行なうインタフェースは共通化し、メモリアロケート演算子は違いはAbstract Factoryパターンの Factory Methodである Create××()関数において吸収する。例えば、Open()は、ファイル版ではファイルを解析しメモ

リ展開し、DB 版ではデータベースのオープンを呼び出す。また、CreateLD() は、PERCIO を使用する場合は new でデータベースオブジェクトを指定している。このように、どの永続化機構を使用しているかを、クライアントに影響を与えずに済むことができる。また、永続化機構の追加も、変更部分が局所化されているため静的ではあるが簡単に行なうことができる。

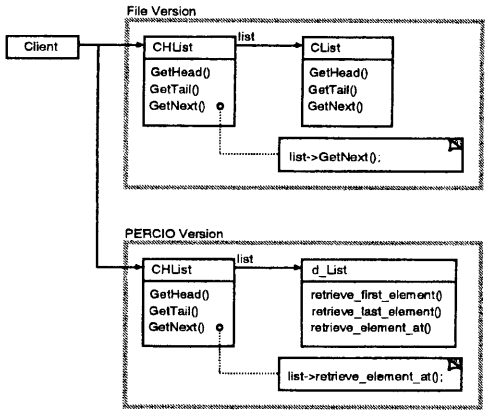

図 7: Adapter パターンの適用

また、クライアントとメディア管理データベースの間でのデータ集合の受け渡しに使用する集合クラスのインタフェースを統一するために **Adapter パターン**を適用している。Adapter パターンは、インタフェースに互換性のないクラスを組み合わせることができる。適用した構造を図7に示す。図7では、PERCIO で提供されるリストである d_List のインタフェースと、VC++ で提供されるリストである CList のインタフェースを統一するためのクラスとして CHList を定義している。CHList を使用しているクライアントは、どちらのリストが実装されているかを意識しないで使用することができる。

4.3 ホットスポット形状への対応

ホットスポット形状の実装を柔軟に変更できるようにする一つの方法として、図8に示すように継承を用いて形状ごとにサブクラスを用意する方法がある。クライアント側では、利用する形状ごとに扱うオブジェクトを切替えることになる。この方法の問題点としては、静止画ホットスポットクラスに対して形状ごとのサブクラスが存在し、領域情報クラスに対しても形状ごとのサブクラスが存在するというように、類似のクラス階層が多数できてしまうことである。別の方法としては、すべ

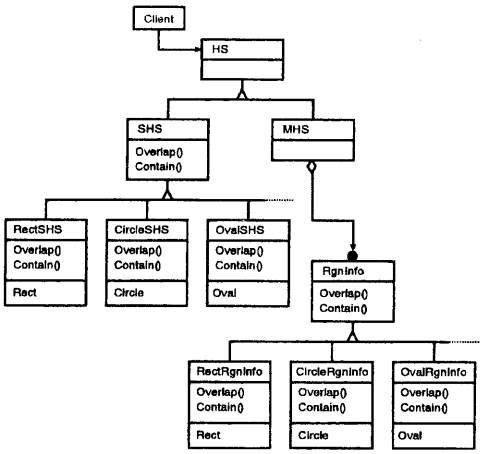

図 8: ホットスポットクラスの継承による方法

ての形状を一つのクラスで実現することが考えられる。例えば、静止画ホットスポットクラスですべての形状に対するデータ構造を持つようにする。しかし、この場合には、OverlapRect()、OverlapCircle() のように形状ごとにインタフェースを分ける必要が生じ、クライアントは形状に応じて呼び出すインタフェースを変えなければならない。

以上に述べた方法で問題であるのは、ホットスポットを扱うクラスで共通に使用されるべき形状に関する実装が、個々のクラスごとに実現されることである。これを解決するためには、形状の実装に関する部分を別クラスとして分ければよい。上記の要求を実現するパターンとして **Bridge パターン**がある。Bridge パターンは以下を実現する。

1. クラスと実装を分離し独立な変更を可能にする
2. 実行時における実装の切替えを可能にする
3. クライアントに実装変更の影響を波及させない

Bridge パターンを適用した結果の構造を図9に示す。ホットスポットの形状に関する実装のインタフェースを宣言する抽象クラスとして Region クラスを用意する。さらに、Region クラスの具象サブクラスとして、RectRegion クラス、CircleRegion クラス、OvalRegion クラスを用意し、それぞれの形状にあったデータ構造および実装を定義する。静止画ホットスポットクラスおよび領域情報クラスと Region クラスはコンポジションの関係を持つ。静止画ホットスポットおよび領域情報に対して Overlap() を呼び出すと、属性 region でつながっている Region オブジェクトの Overlap() が呼び出される。

ホットスポットオブジェクトを利用するクライア

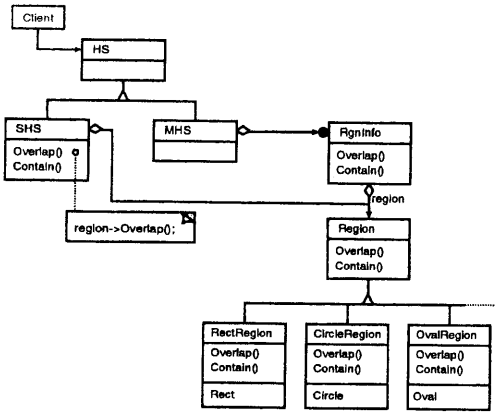

図 9: Bridge パターンの適用

ントであるエディタなどは、ホットスポットの形状に応じて RectRegion クラス、CircleRegion クラス、OvalRegion クラスのいづれかのクラスのオブジェクトを region に設定することにより、呼び出すインタフェースを変えずに形状の実装を変更することができる。また、Region クラスのサブクラスを定義することにより、扱える形状を追加することができる。

4.4 多様な利用方法への対応

通常、今までに述べたようにパターンを適用していくと、小さなクラスが導入され、サブシステムは再利用性が増しカスタマイズが容易になる。しかしその一方で、サブシステムをカスタマイズする必要のないクライアントにとっては、サブシステムの利用が難しくなる。これを解消するために、複雑なサブシステムに対して単純なインタフェースを提供する必要がある。

ナビゲーション時には、複数の MDB に属するメディアを扱うので、MDB にまたがったより高いレベルのインタフェースが必要である。一方、オーサリング時には、ある特定の MDB またはメディアを編集するため、より低いレベルのインタフェースが必要になる。

これを実現するために **Facade パターン**を導入する。Facade パターンでは、facade オブジェクトがクライアントから出された要求を適当なオブジェクトに委譲する。Facade パターンでは、以下を実現できる。

1. クライアントからのサブシステムの隠蔽
2. クライアントとサブシステム間の結合を低める

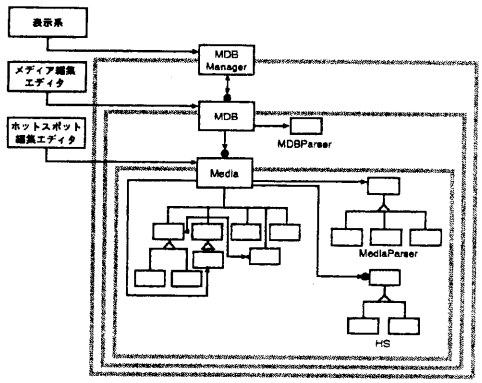

図 10: Facade パターンの適用

3. システムの独立性を高め、階層化を実現する

導入後の構造を図 10に示す。ナビゲーション時は MDB マネージャオブジェクトが facade になり、クライアントである表示系は MDB マネージャを通してメディア管理データベースにアクセスする。実際の処理流れを示すために、MDB マネージャに対して GetScene() を呼び出した時のインタラクションダイアグラム [JCJO92] を図 11に示す。

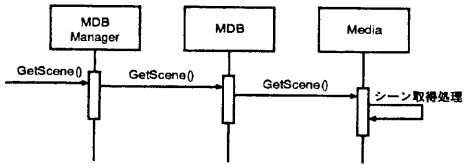

図 11: GetScene() の処理流れ

GetScene() は、ホスト ID、MDBID、メディア ID、シーン ID を引数として呼び出され、対応するシーンを取得する処理を行なう。GetScene() が呼ばれると、MDB マネージャはホスト ID と MDBID から MDB を特定しそれに対して GetScene() を呼び出す。MDB では、メディア ID を基にメディアを特定し GetScene() を呼び出す。メディアでは、シーンの集合からシーン ID に対応するシーンを取得する。このように、表示系は GetScene() というインタフェースを知っていればよく、実際の処理はコンポジションを通して下の方へ順番に委譲される。

一方、メディアのオーサリング時には、MDB オブジェクトが facade になり、クライアントであるメディア編集用エディタは MDB オブジェクトを通してサブシステムにアクセスする。また、ホットス

ポットのオーサリング時には、メディアオブジェクトがfacadeになり、ホットスポット編集用エディタはメディアオブジェクトを通してサブシステムに対してアクセスする。オーサリング作業で必要なパーサなどは、クライアントから隠蔽される。

Facadeパターンを用いることにより、構造が階層化される。このため、システムの分離を容易に行なうことができ、例えば、ホットスポットのオーサリング機能をのみを切り離して、分散オーサリングを実現することが容易に可能となる。

5 関連研究

ハイパーメディアを設計するためのモデルは多数提案されている。例えば、Dexterハイパーテキスト参照モデル[HaSc94]は、ハイパーテキストシステム全体のレイヤ構造を規定しており、実行時層、蓄積層、要素内部層の3層構造を規定している。さらに、Dexter参照モデルにおけるクラス構造が提案されている[Grøn94]。[Grøn94]では、ナビゲーション時のさまざまなノードリンク構造を実現するための蓄積層におけるクラス階層構造をフレームワークとして規定している。ただし、本稿で提案しているクラス構造のように柔軟性の実現を目的とはしていない。ナビゲーションのノードリンク構造やスキーマ構造を設計するためのモデルとしては、オブジェクト指向に基づくOOHDM[ScRo95]などが提案されている。これらのモデルでは、主に、設計のためのダイアグラムを規定している。

一方、オブジェクト指向フレームワークにおける位置付けを考えると、デザインパターンを適用したオブジェクト指向フレームワークはいくつか提案されている。例えば、ネットワーク管理のためのオブジェクト指向フレームワーク[HüJE95]や機械管理用のオブジェクト指向フレームワーク[Schm95]がある。ただし本稿と同様に、これらのフレームワークでは性能面での評価については述べていない。

6 おわりに

ハイパーメディアシステムにおけるメディア管理データベースに特に必要とされる柔軟性として4つを上げ、それらをデザインパターンを使用して実現する一方法を提案した。このうち、スケーラビリティに関しては、効率などの問題からハードコーディングする方法で実現した。提案した構造を用いることにより、利用者および他コンポーネントの開発者からの拡張やカスタマイズの要求に対してシステム全体を作りかえずに容易に対応できる。

速度やメモリ効率などの性能面での評価は、今後の課題である。定性的には、アクセスするオブジェクトの数が増加しているため、その分速度は遅くメモリ消費量は増えることになる。ただ、ハイパーメディアの場合は処理が対話的に行なわれるため、速度に関しては問題にはならないことも予想される。その他、クラス構造のメトリクスを用いた評価も行なう必要がある。

謝辞

本システムを設計するにあたり有益な御助言を頂いた弊社C&C研究所 鶴岡邦敏課長、御指導頂いた同 友納正裕主任に深謝致します。

参考文献

[GHJV94] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.

[Grøn94] K. Grønbæk, Composites in a Dexter-Based Hypermedia Framework, European Conference on Hypermedia Technology(ECHT), 1994.

[HaSc94] F. Halasz, M. Schwartz, The Dexter Hypertext Reference Model, Communications of the ACM, 37(2), 1994.

[HKTH94] 平田, 川崎, 高野, 原, ネットワーク環境下における動画ハイパーメディア実装方式, Proceedings of Advanced Database System Symposium, 1994.

[HüJE95] H. Hüni, R. Johnson, R. Engel, A Framework for Network Protocol Software, ACM Conference on Object Oriented Programming Systems, Languages and Applications(OOPSLA), 1995.

[JCJO92] I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard, Object-Oriented Software Engineering - A Use Case Driven Approach, Addison-Wesley, 1992.

[RBPEL91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorenson, Object-Oriented Modeling and Design, Prentice Hall, 1991.

[Schm95] H. Schmid, Creating the Architecture of a Manufacturing Framework by Design Patterns, ACM Conference on Object Oriented Programming Systems, Languages and Applications(OOPSLA), 1995.

[ScRo95] D. Schwabe, G. Rossi, The Object-Oriented Hypermedia Design Model, Communications of the ACM, 38(8), 1995.

[TsKi94] 鶴岡, 木村, オブジェクト指向データベースPERCIOの機能と構成, NEC技報, Vol.47, No.6, 1994.