ソフトウエア工学 93−15
（1993．7．9）

Duma:マルチメディア文書エディタフレームワーク上の UIMS

大津 隆史† Michael A. Harrison‡

†松下電器産業株式会社 情報システム研究所
E-mail:ohtsu@isl.mei.co.jp

‡Computer Science Division, University of California, Berkeley
E-mail:harrison@cs.berkeley.edu

マルチメディア文書エディタフレームワーク MMDEF 上のユーザインタフェース (UI) 管理システム Duma (Data-based User interface management system for Multimedia Application) の設計及び実装について報告する。MMDEF は、メディア非依存の編集サービスを提供するマルチメディア文書エディタの核として、ユーザに違和感を与えることなく外部定義メディアを文書の構成要素として適応可能にすることが目標である。Duma は、アプリケーション・UI 間の実行時の対話を表現したインタラクタモデルと、UI の設計とプロトタイピングとをインタラクタモデルに基づいて行うことができる「データに基づく UIMS アーキテクチャ」とを特徴とし、UI において外部定義メディアの適応性を実現した。我々は、MMDEF 上のマルチメディア文書エディタのプロトタイプ Ensemble において Duma のプロトタイプを実装し、その有用性を検証した。

Duma: A UIMS for Multimedia Document Editor Framework

Takashi Ohtsu† Michael A. Harrison‡

†Information Systems Research Lab., Matsushita Electric Ind. Co., LTD.
1006 Kadoma, Kadoma, Osaka 571 JAPAN
E-mail:ohtsu@isl.mei.co.jp

‡Computer Science Division, University of California, Berkeley
571 Evans Hall, Berkeley, CA 94720, U.S.A.
E-mail:harrison@cs.berkeley.edu

This paper describes *Duma*: a *Data-based User interface management system for Multimedia Application*, which is embedded in a multimedia document editor framework (MMDEF). MMDEF is the core of a multimedia document editor, which can adapt to medium types and enables the users to work on documents composed of multimedia objects, including objects of newly defined types. Duma features **interactor model** that abstracts runtime interaction among applications and UI's, and **data-based UIMS architecture** that isolates UI design stages based on that model. The prototype of Duma is implemented on the Ensemble system which is a prototype editor on MMDEF. We claim that Duma provides adaptable UI models to MMDEF and one seamless multimedia document editor to the users.

1 背景

CPU 能力の飛躍的向上と周辺機器の進化は、デスクトップコンピュータによる映像・音声等マルチメディアオブジェクトの処理を容易にしつつある。この傾向を受け、DTP 分野においては、マルチメディア文書を対象とした編集支援システムが注目されている。例えば MS Word [3] は、そのプラグイン機能により、外部定義されたメディアの接続を実現している。これらのシステムの多くは、テキスト以外のメディアオブジェクトをテキストメディアの特殊オブジェクトとして扱うことによりマルチメディア文書を表現し、被接続メディアの適応性を保証している。一方、それらの編集環境においては、図 1(A) に示す様に、ユーザは文書編集時にメインエディタと各メディア用サブエディタとの間で明示的な往来を強いられる。この様な、ユーザにメディアを常に意識させる文書編集環境は、メディアの種類と組み合せ数の増大によりその操作性の複雑度が爆発し、近い将来本質的な破綻を来すと考えられる。

この背景を踏まえ、カリフォルニア大学バークレイ校では、アンサンブルプロジェクト [7] においてマルチメディア文書エディタ Ensemble の研究・開発が進められている。Ensemble は MMDEF (MultiMedia Document Editor Framework) と呼ばれるフレームワーク上に実装されている。MMDEF 上の編集支援システムでは、各メディアがそれぞれ等しく扱われるため、図 1(B) で示す様に、ユーザにメディアの違いを意識させないシームレスな文書編集環境を提供可能である。Ensemble の画面例を図 2 に示す。

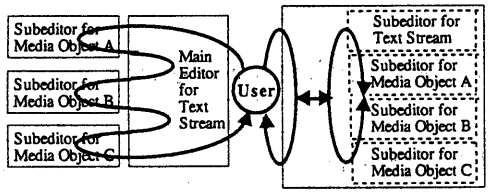

図 1: マルチメディア文書の編集環境

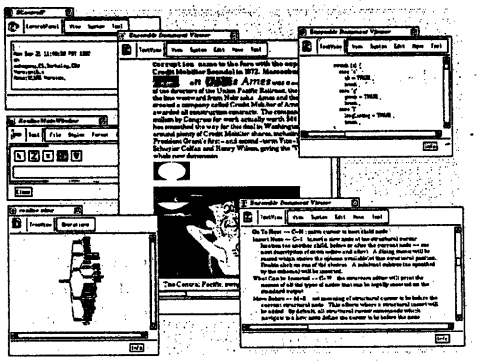

図 2: Ensemble の画面例

MMDEF は、シームレスな編集環境という点で、各メディアに一貫性のとれたユーザインタフェース (UI) を提供し、ユーザインタフェース管理システム (UIMS) としての役割が重要である。本報告では MMDEF 上の UIMS である Duma (Data-based User interface management system for Multimedia Application) の設計とその実装について述べる [8]。Duma の特徴は、1) メディアを問わず、ウィジット等の UI コンポーネントとアプリケーション (AP) 間の対話をモデル化できるインタラクタ・モデル、2) インタラクタ・モデルに基づいて、UI・AP 間の適合性を保証し、対話的な UI 開発環境を提供する「データに基づく UIMS アーキテクチャ」にある。図 3 に、MMDEF、並びに Duma を含む Ensemble のアーキテクチャを示す。

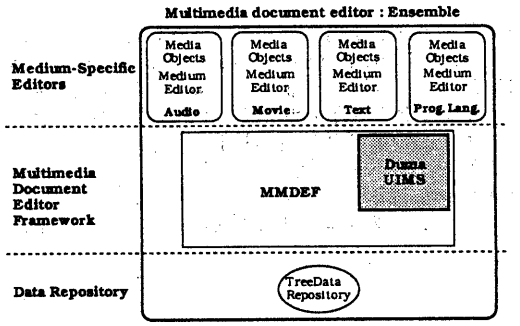

図 3: マルチメディア文書エディタ Ensemble のアーキテクチャ

以下、Duma における UIMS アーキテクチャの基本的な考え方を述べた後、MMDEF の構成に続いて、Duma のデータモデル、プロトタイプを紹介し、プロトタイプ実装を終えての結論の後と今後の展望を添える。

2 UIMS アーキテクチャ

グラフィカル UIMS

Seeheim モデル [12] の提唱以来、数多くの UIMS が報告され、最近はウィンドウ・アプリケーション数の増大と共に、グラフィカル UIMS であるウィジットが重要視されている。一方、既存の Motif、OpenLook、ET++ [13] 等のウィジットは、UIMS 本来の使命である UI・AP の分離という点では不充分といえる。例えば、UI 部でのプルダウンメニューからラジオボタンへの変更といった要求に対し、AP 部でのソースコード修正を不要とする UIMS は数少ない。これは、UI・AP 間で実行時に交換されるデータ型及びその際の対話モデルが、UIMS ではなくプログラマによって保守されていることに起因する。グラフィカル UIMS をはじめ、従来の UIMS アーキテクチャは、主に UI・AP 間の対話における各イベントのやりとりを重視し、対話の持つ意味ならびに対話で交換されるデータについて UI・AP 間共通のモデルを持たないため、AP 側が要求するデータ型と UI 側を通してユーザから得るデータの適合性管理についての支援は困難である。

データに基づく UIMS

実行時の UI・AP 間で行われる対話を抽象データ型として明示的にサポートする UIMS を「データに基づく UIMS (Data-based UIMS)」と呼ぶ。本報告で紹介する Duma は、この UIMS アーキテクチャに基づいて設計されている。

Duma 設計の動機となった ITS [14] は初期の「データに基づく UIMS」の一つといえる。UI・AP 間で交換されるデータは表形式で畜えられ、ユーザは UI コンポーネントを通して対話的にこの表にアクセスし、変更されたデータに従い AP 側が起動される。対話時に表示する UI コンポーネントの決定には if-then 形式のルールが用いられる。このルールでは、if 部に変更されたデータの属性条件、then 部に必要な UI コンポーネントの属性が記述され、予め AP 起動前にコンパイルされる。従って、ルールを AP 実行中に修正することはできず、その結果、種々の UI コンポーネントのプロトタイプが難しい。

Selectors [5] は、HP ACE 環境で用いられる高級ウィジットであり、ITS における UI コンポーネントの属性に相当する記述をウィジット内部に納めている。即ち、Selectors にはデータ選択における対話モデルがウィジット内部に隠蔽されている点で、ITS の拡張といえる。一方、[5] では、Selectors が扱うデータ型の拡張性、選択以外での対話モデルの拡張性には触れられず、対話的に UI コンポーネントをプロトタイピングするインタフェースについて述べられていない。これらは、メディアオブジェクト・サブエディタの拡張性を主眼とするマルチメディア文書編集システムには必須であると我々は考える。

3 MMDEF アーキテクチャ

MMDEF の研究目標は、メディアに依存しないマルチメディア文書編集サービスを明らかにすることにより、各々のメディア固有の編集機能や UI を調停し、文書編集時にエンドユーザから各メディア間の境界を隠蔽することである。MMDEF は複数のメディエータと呼ばれるソフトウェアモジュールから成り、それらの各々が異なる文書編集サービスを提供する。現在、MMDEF は 5 つのメディエータ、即ち、ツリーデータ、ストラクチャ、プレゼンテーション、エディタ、及び UI の各メディエータから構成される。図 4 は MMDEF アーキテクチャの概要を示したものである。Duma の設計指針は UI メディエータのそれに一致する。

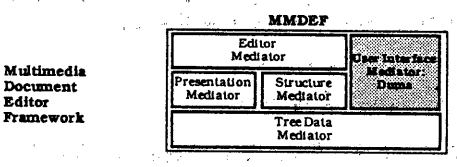

図 4: MMDEF アーキテクチャ

MMDEF で扱われる文書オブジェクトは各々一つの文書クラスに属し、文書クラスはメディエータ間で共有管理される。各文書クラスは複数のメディアから構成され、文書クラス自身も一メディアとみなされる。文書クラスは外部定義可能であり、MMDEF はその文書クラスのマルチメディア文書エディタに対する適応性を保証する。不可分なメディアはプリミティブメディアと呼ばれ、Text、Movie はその例である。文書クラスは、固有の編集方法に関してサブエディタを持つことができる。

以下、各メディエータについて順に述べる。

3.1 ツリーデータ・メディエータ

文書論理構造をはじめ、MMDEF で用いられる基本データはツリーで表現され、ツリー・レポジトリに納められる。ツリーデータ・メディエータは、これらツリーデータに対する効果的な操作インタフェースを提供する。特徴的なツリー・メディエータのサービスの一つは、複数のクライアントが同時に一つのツリーデータを共有する場合に用いられるバーチャルツリークラスである。即ち、バーチャルツリークラスは、操作対象となるツリーデータを受けとり、その基本構造をベースツリーとして保持し、操作要求を出したクライアントに対して、ベースツリーから導出されたバーチャルツリーを渡す。当初、ベースツリーとバーチャルツリーとは同じ構造を持つが、以降のベースツリー上の構造上の操作は即座に各々のバーチャルツリーに反映され、一方、バーチャルツリーへの操作は他のバーチャルツリーやベースツリーへ直接反映されない。従って、各メディエータは同一のツリーデータに対し、個別に自身の情報をアノテートすることができ、バーチャルツリーを一時的なデータ表現手段として用いることができる。また、個々のバーチャルツリーからベースツリーへの構造変更要求は排他的に行われ、DBMS にみられるトランズアクションと類似の機能を有する。

3.2 ストラクチャ・メディエータ

ストラクチャ・メディエータは編集中における文書論理構造の一貫性を保証する。即ち、ストラクチャ・メディエータは、各文書クラスごとに定義された論理構造に関するルールセットにより、文書構造の変更要求に対する是非を判断する。このルールセットはストラクチャ・スキーマと呼ばれ、外部アスキーファイルにより与えられる。例として、文書クラス Memo のストラクチャ・スキーマを図 5(A) に示す。ストラクチャ・スキーマより導出された文書オブジェクトの論理構造は、ドキュメント・ツリーと呼ばれる。Memo クラスに属する文書として導出されたドキュメント・ツリーの例を図 5(B) に示す。

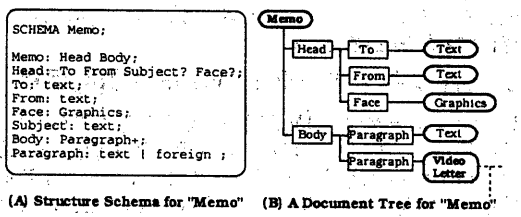

図 5: Memo ストラクチャ・スキーマとドキュメント・ツリー

3.3 プレゼンテーション・メディエータ

プレゼンテーション・メディエータは、文書論理構造からメディア非依存の文書表現用データを導出する [4]。この文書表現用データはプレゼンテーション・ツリーと

呼ばれ、ビットマップディスプレー、プリンタ、スピーカ等出力デバイス依存の各フォーマッタにより処理される。ビットマップディスプレーに表示された文書(の一部)はビューと呼ばれ、図2に示したウィンドウは個々のビューに対応する。ビューを生成するAPはビューワと呼ばれる。

プレゼンテーション・ツリーの導出は、各文書クラスごと固有のルールセットに基づいて行われる。このルールセットはプレゼンテーション・スキーマと呼ばれ、外部アスキーファイルにより与えられる。文書クラスMemoに与えられたプレゼンテーションスキーマの一部を図6に示す。

```
MEDIUM text;

PRESENTATION memo FOR memo;

DEFAULT
    BEGIN
    Width = AllChildren . Width;
    Height = AllChildren . Height;
    VertPos: Top = LeftSib . Bottom;
    HorizPos: Left = Parent . Left;
    Justify = Parent . Justify;
    FontFamily = Parent . FontFamily;
    Bold = No;
    Italic = No;
    Size = Parent . Size;
    LineSpacing = Parent . LineSpacing;
    Indent = 0;
    Visible = Yes;
    RightMargin = 504;
    END;
```

```
RULES
    Memo :
        BEGIN
        Width = 432;
        HorizPos: Left = 72;
        VertPos: Top = 36;
        FontFamily = "schoolbook";
        LineSpacing = 1.2;
        Size = 14;
        Justify = LeftJustify;
        END;
    To :
        BEGIN
        HorizPos: Left = RightSib . Left;
        Italic = Yes;
        END;
    From :
        BEGIN
        HorizPos: Left = RightSib . Left;
        Italic = Yes;
        END;
```

図6: Memoプレゼンテーションスキーマ

3.4 エディタ・メディエータ

エディタ・メディエータは、マルチメディア文書に対するユーザの編集要求を、各メディアのサブエディタのコマンドの手続きとして解釈・実行することにより複数のサブエディタ機能を一つの統合エディタ機能としてユーザに提供する[10]。例えば、表示文書中の複数のメディアオブジェクトへ発行された編集要求に対して、エディタ・メディエータはプレゼンテーション・メディエータから得る表示位置情報とストラクチャ・メディエータから得る論理構造情報とを用いて、対象となるメディアオブジェクト及びそのサブエディタを特定し、編集要求を転送する。

また、MMDEFでは、同一のドキュメント・ツリーを異なるビューで表示することが可能である。このような場合、エディタ・メディエータは、異なるビューで発行された編集要求とその結果とを対応するビューすべてに反映させることにより、マルチユーザによる同時編集を可能にする枠組を提供する。

3.5 UIメディエータ：Dumaの設計目標

UIメディエータは、MMDEFに接続されたサブエディタ及びビューワ等のAPに対し、メディア非依存の一貫したUIモデルを提供し、対話的な設計・カスタマイズが可能なUI開発環境を実現する。即ち、MMDEFにおける外部メディア及び文書クラスの適応性に基づき、これらを扱うUIも適応性を持たせることにより、各メディア固有のUIの共有と再利用環境とを目指し、複数メディアから成る文書クラスのUI設計を容易にする。

UIメディエータはUI・AP間の対話をモデル化し、そのモデルに基づいて実行時の対話処理をするエージェントを備える。また、AP実行時において、UI・AP側がそれぞれエージェントとのみ通信可能となる枠組をUIMSアーテクチャとしてUIメディエータが提供する。このエージェントの特徴は、与えられた対話モデルにより、対話実行時にどのタイプの値が交換され、対話の詳細外部仕様を実現するウィジットとしてどのUIコンポーネントが適合するかを理解している点である。例えば、本エージェントにより、文字列を返すプルダウンメニューと同じく文字列を返すラジオボックスとは、交換可能なUIコンポーネントとして理解される。従って、従来のUIMSに較べ、対話の本質的意味に近づいたUIサービスが可能となり、AP設計とUI設計との分離がより明瞭となる。

UIメディエータの最終的な目標は、上で述べた対話モデルとこのモデルに基づくUIコンポーネントとを一つのユーザインタフェースメディアとしてMMDEFに提供することにある。これにより、MMDEFで提供される文書編集サービスをUI開発環境として最大限利用することができ、一貫性のとれた対話的UI設計が可能であると考えられる。これは、Two-Viewアプローチ[1]でとられた対話設計手法を、マルチメディア分野へ大きく前進させたものといえる。加えて、既存のマルチメディアとUIメディアを複合させた文書クラスの定義により、マルチメディア文書自身にユーザインタフェースを埋め込むことも可能となる。Embedded Buttonsアプローチ[2]は、その一例であると考えられる。

4 インタラクタ・モデル

本報告では、マルチメディア文書編集におけるUI・AP間の対話を、対話シナリオに基づいたUI・AP間でのメディアオブジェクトの受け渡しであると位置づける。例えば、あるUI・AP間の「従業員リストからユーザが指定した従業員一名を得る」という対話においては、対応する対話シナリオは「スクローラブルなペーンウィンドウに順次表示された従業員リストから、ハイライトされた従業員一名の名前を返す」であり、受け渡されるメディアオブジェクトは「大津隆史」という名前属性を持つ従業員オブジェクト考えられる。

Dumaのインタラクタは、この対話シナリオとメディアオブジェクトとを各メディア共通のモデルで表現した、対話の抽象モデルである。インタラクタは、次の3つのオブジェクトから成る。即ち、アプリケーション側からインタラクタにアクセスするためのバーチャル・インタラクタ(VI)、ユーザ側からインタラクタにアクセスするためのインタラクタ・ウィジット(IW)、そして、バーチャル・インタラクタとインタラクタ・ウィジットを接続するリアル・インタラクタ(RI)である。一つのインタラクタは次のいずれかのタプルで表現される。

```
1) (a VI, a RI, a IW)
2) (a VI, a RI)
3) (a RI, a IW)
```

多くのインタラクタは上記1)で表現される。2)で表されるインタクタをアプリ定義インタラクタ、3)で表されるインタラクタをユーザ定義インタラクタと呼ぶ。以下それぞれのオブジェクトについて詳細を述べる。

4.1 バーチャル・インタラクタ

バーチャル・インタラクタはAPからインタラクタへアクセスするためのインタフェースを提供するオブジェ

クトである。バーチャル・インタラクタでは対話シナリオの型をインタラクタ・タイプとして管理する。インタラクタ・タイプは、対話時に交換されるメディアオブジェクトと、対話シナリオの詳細外部仕様に依存しない部分とを持ち、対話の本質的な意味を表現する。現在、バーチャル・インタラクタは簡単な決定木として記述されている。インタラクタ・タイプの例を図7に模式的に示す。インタラクタ・タイプの各々の葉はMMDEFに知られたメディアオブジェクトのクラス名であり、ベースタイプと呼ばれる。ベースタイプは互いにクラス階層を持つ。図中(A)では、ベースタイプFONTNAMEを示し、FONTNAMEのとる値はHelvetica、Courier等が挙げられる。一方、インタラクタ・タイプでの中間ノードはCHOICE(k)ファンクションのインスタンスである。ここで引数kは自然数かあるいは予約語ANY、EVERYのいずれかであり、同時に選択できる自分の子ノードの数を表すものとする。厳密には、CHOICEファンクションは子ノードの種類によって次の二つの意味を持つ。

- 子ノードは唯一のベースタイプBASETYPEの場合
  CHOICE(k)は、BASETYPEとして有効な値が最大k個選択可能であることを表わす。
- 子ノードは複数の中間ノードの場合
  CHOICE(k)は、自身のサブツリーのうち選択可能なものが最大k個であることを表わす。

図7(A)で、親ノードはkの値1を持ち、唯一のベースタイプの葉を持つ。従って、インタラクタ・タイプGETFONTNAMEは「フォント名のうち一つを選ぶ」という意味を持つ。明らかに、このインタラクタ・タイプは対話実行時に用いられるUIコンポーネント等の詳細外部仕様とは独立である。以降の説明のため、フォントサイズを一つ得るインタラクタ・タイプGETFONTSIZEを図7(B)に示す。また、MS Wordにみられるようなフォントフェース選択を表したインタラクタ・タイプの例を図7(C)に示す。ここでOTHERTYPEFACEのとる値は、Bold、Italic、Shadow、Outlineであり、NORMALTYPEFACEは値Normalのみを持つ。インタラクタ・タイプの各ノードは属性を持つことができ、デフォールト値等についての情報を収めることができる。

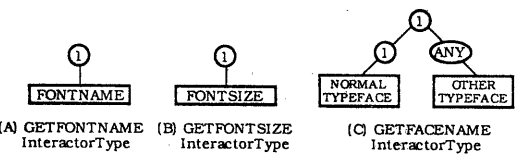

図7: インタラクタ・タイプ

4.2 インタラクタ・ウィジット

インタラクタ・ウィジットは、複数のボタン、メニュー、ウィンドウ等からなるUIコンポーネントのオブジェクトである。Selectors自身もインタラクタ・ウィジットの一種と考えられる。インタラクタ・ウィジットの外観例を図8に示す。バーチャル・インタラクタ同様、インタラクタ・ウィジットもインタラクタ・タイプによって型が与えられる。現設計では、インタラクタ・ウィジットに与えられるインタラクタ・タイプは一つのCHOICEファンクションと一つのベースタイプとから成る。ここでベースタイプは、対応するインタラクタ・ウィジットが扱うことができるオブジェクトのクラスを示す。

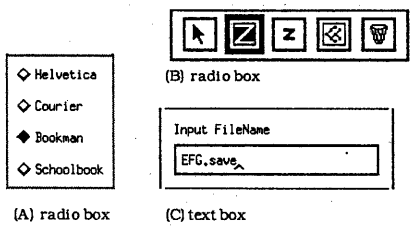

図8: インタラクタ・ウィジット

例えば、単純なポップアップメニューのインタラクタ・ウィジットはCHOICE(1)であると考えられ、ラベルやアイコンなどUI側からの対話を必要としないインタラクタ・ウィジットはCHOICE(0)である。また、図8(C)に示すテキストプロンプトボックスはTextベースタイプのCHOICE(1)であり、トグルボタンはBOOLEANベースタイプのCHOICE(1)であるといえる。

4.3 リアル・インタラクタ

リアル・インタラクタは、バーチャル・インタラクタとインタラクタ・ウィジットとの間をバインドする役目を持つオブジェクトである。即ち、リアル・インタラクタは実行時に双方のインタラクタ・タイプの適合性をチェックし、適合するならば両者を接続し、UI・AP間の通信を可能にする。この適合性チェックのアルゴリズムの一例を図9に示す。図9では、最初にバーチャル・インタラクタのタイプに現れるベースタイプがすべてインタラクタ・ウィジットのタイプのベースタイプと適合するかをチェックしている。ここでベースタイプAがベースタイプBに適合するということは、BがAのサブクラスとして定義されていることに等しい。

5 Dumaにおける UIMS アーキテクチャ

Dumaにおける「データに基づくUIMSアーキテクチャ」は、前節で述べたインタラクタ・モデルに従って、サブエディタやビューワなどMMDEF上のウィンドウ指向APのUI設計を支援する。Dumaでは、UIの設計段階を大きく次の5つに分割した。つまり、メディアアクション、ダイアログ、スタイルコンポジション、レイアウトコンポジション、そしてコマンドディスパッチャである。図10にDumaの全体構成を示し、以下各設計段階について詳細を述べる。

5.1 メディアアクション

メディアアクションでは、MMDEFに管理されたメディアオブジェクトのベースタイプを用いて、インタラクタ・タイプの設計を行うインタフェースが与えられる。インタラクタ・タイプの仕様設計は、AP側の設計及びインタラタ・ウィジットの設計とは独立して行われることに注目されたい。

```
CheckTypesOf(ITV:InteractorType of a VirtualInteractor,
             ITW:InteractorType of an InteractorWidget){
  // check all base types
  for (all leaves of ITV){
    if(basetype of leaf is not compatible the base
      type of ITW) return ERROR;
  }
  // check CHOICE types
  for (traverse ITV){
    if(current node is CHOICE(x) node){
      if(children are base types)
         currentnode.maxchoice = x;
      if(children are CHOICE nodes)
         currentnode.maxchoice =
          maximum selectable items at this level
          considering x and the maxchoice of each child;
      }
  }
  Node node = rootNode of ITV;
  if(node.maxchoice <= y ) // where ITW  has a type
                           // of CHOICE(y)
    return AGREE;
  else
    return ERROR;
}
```

図 9: インタラクタ・ウィジット

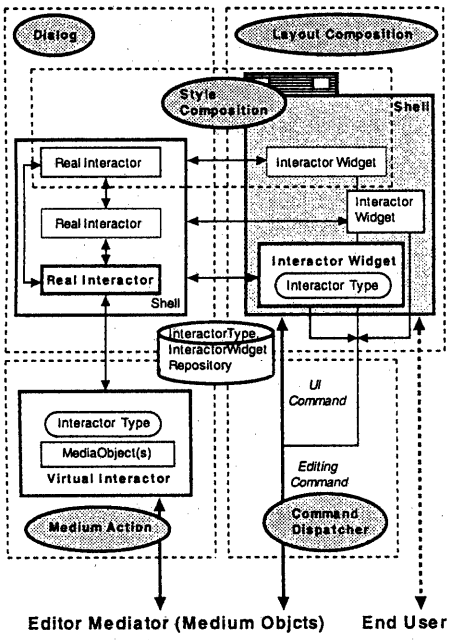

図 10: Duma の UIMS アーキテクチャ

5.2 ダイアログ

ダイアログでは、関係の深いインタラクタの集合を一つのシェルとして設計するインタフェースが与えられる。シェルはインタラクタの一つであり、複数のインタラクタに共通のスコープを提供すると考えられる。なお、一つのインタラクタは複数のシェルにまたがることができる。通常、各々のシェルには、ウィンドウの外見を持つインタラクタ・ウィジットが一つ割当られる。また、シェル内のリアル・インタラクタ間の制約は、属するシェルに直接記述される。

例として、フォント情報をユーザから得る際に、フォントについての名前、サイズ、フェースタイプの入力を同時に促すようなダイアログの設計の場合、AP 設計者は、FontSelectDialog と呼ばれるシェルを、先に示した GETFONTNAME、GETFONTSIZE、GETFACENAME のインタラクタ・タイプを持つインタラクタの集合オブジェクトとして定義する。

5.3 スタイルコンポジション

スタイルコンポジションでは、各インタラクタに対しインタラクタ・ウィジットを割当るための設計インタフェースが与えられる。即ち、4.3 章で述べたインタラクタ・ウィジットの適合性アルゴリズムに従って、UI 設計者は、対話的にインタラクタ・ウィジットを指定することにより、実行中の UI コンポーネントのプロトタイピングを行う。また、インタラクタ・タイプに基づいてレポジトリ内の適合可能なインタラクタ・ウィジットを検索することも可能である。

5.4 レイアウトコンポジション

レイアウトコンポジションでは、インタラクタ・ウィジットの表示レイアウト設計を行うインタフェースが与えられる。本レイヤでの基本的なサービスは、3.3章で述べた MMDEF のプレゼンテーション・メディエータのサービスと共通であると考えられる。レイアウトモデルとしては、TEX で用いられるボックス・グルーモデルや多くの汎用ウィジットで用いられる水平・垂直方向整列モデルなどがあり、UI 設計者がいずれかのレイアウトモデルを選択できる。

さらに、UI 設計者は任意の AP 側と直接バインドされないユーザ定義インタラクタをシェルに追加しレイアウトすることが可能である。ユーザ定義インタラクタにより、UI 独自の付加的な情報を AP 側の変更なしに提供することができる。レイアウト例として、先に示したシェル FontSelectDialog から生成された画面表示例を図 11に挙げる。この画面構造は図 12 で示される。

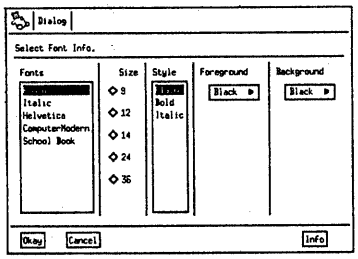

図 11: FontSelectDialog シェルの画面例

5.5 コマンドディスパッチャ

コマンドディスパッチャでは、サブエディタや Duma が処理可能なコマンドを各シェルのポップアップメニュー、メニューバー及びキーボードに割付ける設計のためのインタフェースが与えられる。Duma ではコマンド発行を行うインタラクタの型として、ベースタイプ COMMAND が定義されている。即ち、コマンド発行可能なメニューや

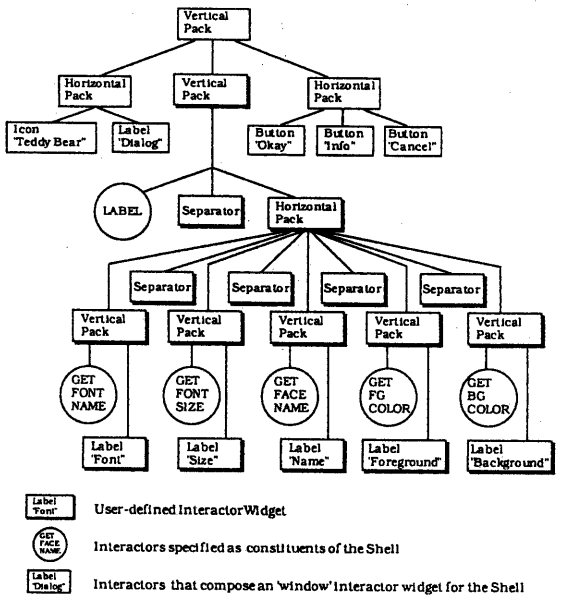

図 12: FontSelectDialog シェルの構造

ボタン等は、COMMAND 型からなるインタラクタ・タイプを持つインタラクタであり、実行時のユーザ操作によりコマンドを MMDEF のエディタ・メディエータに送信する役目を持つ。例えば、シェルの開閉など UI コンポーネント独自の動作は、Duma への UI コマンドとしてエディタ・メディエータから Duma 自身にルーティングされる。

6 Duma プロトタイプ

Duma プロトタイプは C++で記述され、1節で述べた MMDEF のプロトタイプエディタ Ensemble の一モジュールとして、SUN SparcStation 上で稼働している。現時点で、インタラクタの各オブジェクト及び UIMS アーキテクチャの基本部分の実装を終えた。現在、Ensemble から MMDEF に提供されるプリミティブメディアはテキスト、静止画、及び簡単な動画に限定されている。また、プログラミング言語も一メディアとして MMDEF に接続されている。

Duma プロトタイプでの UI 設計において、対話的プロトタイプが可能なものは UI 仕様言語 USPEC により外部アスキーファイルに記述される。UPSEC は MMDEF のテキストエディタによる直接編集、ツリーエディタ等種々のツールによる間接編集が可能である。UPSEC は AP 実行中にロードされ、USPEC 上での変更は即座に UI 上の変更として実行中の AP に反映される。以下に、Duma プロトタイプでの UI 設計例について述べる。

バーチャル／リアル・インタラクタ仕様

インタラクタ・タイプの仕様は C++ で直接記述されている。例えば、先に示したインタラクタ・タイプ GETFONTNAME、GETFONSIZE は図 13 で示されるフラグメントで表される。ここで IDT_ONE、IBT_FONTNAME、IBT_FONTSIZE はそれぞれファンクション CHOICE(1)、ベースタイプ FONTNAME、ベースタイプ FONTSIZE を示す。これらのインタラクタ・タイプを用いて、インタラクタの登録を図 14 に示す様に行う。図 14の2行目では、図 13で定義された GETFONTSIZE タイプのインタラクタ size が生成され、ツリーデータ・レポジトリに蓄えられることを示す。

```
new InteractorType("GETFONTNAME", IDT_ONE, IBT_FONTNAME);
new InteractorType("GETFONTSIZE", IDT_ONE, IBT_FONTNAME);
```

図 13: インタラクタ・タイプの生成

```
registerInteractor("body", "LABEL");
registerInteractor("size", "GETFONTSIZE");
registerInteractor("name", "GETFONTNAME");
registerInteractor("style", "GETFACENAME");
registerInteractor("bgcolor", "GETCOLOR");
registerInteractor("fgcolor", "GETCOLOR");
```

図 14: インタラクタ仕様

シェル仕様とインタラクタ・ウィジット割付仕様

図 15 では、先に述べたシェル FontSelectDialog を図 14 で登録した 6 つのインタラクタの集合型オブジェクトとして記述する例を示す。同時に、図 15では、各インタラクタに割付けるインタラクタ・ウィジットの指定も行っている。例えば、3行目において、body リアル・インタラクタは SIMPLE_LABEL インタラクタ・ウィジットに割当てられている。この割当の整合性はリアル・インタラクタによりチェックされ、必要であればツリーデータ・ポジトリより適合可能なインタラクタ・ウィジットが選ばれる。

```
DIALOGSET FontSelectDialog
BEGIN
    PARTS = (%body,   SIMPLE_LABEL),  // label
            (%fonts,  ONE_OF_M_LIST), // list selection box
            (%size,   ONE_OF_M_RBTN), // radio button
            (%style,  ONE_OF_M_LIST), // list selection box
            (%fgcolor,ONE_OF_M_PBTN), // push button
            (%bgcolor,ONE_OF_M_PBTN); // push button
END
```

図 15: インタラクタ・ウィジットへの割付け (USPEC)

レイアウトスタイル仕様

レイアウトスタイルは、インタラクタ・ウィジットからなる木構造として記述される。二次元配置を司るインタラクタ・ウィジットとして、VPack、HPack はそれぞれ子を垂直／水平方向に整列させることができる。図 16は、先に述べたシェル FontSelectDialog におけるインタラクタ・ウィジットのレイアウトスタイル仕様の USPEC による記述を示す。

6.1 コマンドディスパッチャ仕様

コマンド・ディスパッチャ仕様はキーボードコマンド仕様、プルダウン／ポップアップメニュー仕様とから成り、各々 UPSEC で記述できる。図 17にメニュー仕様を記述

```
DIALOGSET FontSelectDialog
BEGIN
  STYLE = VPack(%body,Sep(),
            HPack(VPack(SLabel("Font"), %fonts),Sep(),
                  VPack(SLabel("Size"),%size), Sep(),
                  VPack(SLabel("Style"),%style),Sep(),
                  VPack(SLabel("Foreground"),%fgcolor),Sep(),
                  VPack(SLabel("Background"),%bgcolor)));
END
```

図 16: シェル FontSelectDialogのレイアウト仕様 (USPEC)

したUSPECを示し、図18に図17で示されたMENUBARの階層図を示す。

```
MENUSET Default BEGIN
  MENUBAR    (View, Edit(Insert, Delete));
  POPUPMENU (Insert, Delete);
  DEFINE View BEGIN
    LABEL="View"; MEDIUM= TextEd;
    ITEMS=("Open","editor SuperEditor OpenDocument"),
          ("Save","editor SuperEditor SaveDocument"),
          ("Export","editor TextEd ExportDocText"); END
  DEFINE Insert BEGIN
    LABEL="Insert"; MEDIUM= TextEd;
    ITEMS=("Node","editor StructEditor InsertNode"),
          ("Query","editor StructEditor QueryInsert");END
  DEFINE Delete BEGIN
    LABEL="Delete"; MEDIUM= TextEd;
    ITEMS=("Atom","cursor DeleteAtom"),
          ("AtomBack","cursor DeleteAtomBack"),
          ("Word","cursor DeleteWord");                END
  DEFINE Edit BEGIN
    LABEL="Edit"; MEDIUM= ;
    ITEMS=("Redraw","editor StructEditor Repaint"); END
END
```

図 17: メニュー仕様 (USPEC)

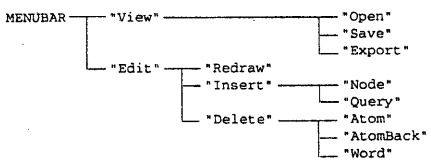

図 18: メニュー階層

7 結論

本報告ではマルチメディア文書エディタフレームワーク MMDEF 上の UIMS である Duma について述べた。現在 Ensemble は、扱うことができるメディア数は限られるが様々なビューワ・エディタ等の AP を備えており、全体で 10 万行近くのコード量[1]の規模を持つ。AP は各々 Duma によって主要な UI を管理されている一方、Duma を含む UI 部のコード量は全体の１０～１５%程度の割合を占めていることを考慮する[2]と、Duma の特徴であるインタラクタ・モデル及び「データに基づく UIMS アーキテクチャ」は MMDEF 上での UI 再利用を促し、全体のコード量圧縮に大きく貢献したと考えられる。

今後の課題として、Duma で支援する UI 設計から C++ レベル記述を排除し、USPEC による一貫設計を実現することが挙げられる。現在の USPEC による対話的な UI 設計は、そのデータ構造がツリーであることから、ツリーエディタ [11] を用いて間接的に行うことが可能であるが、更に UI 設計に適したエディタへと調整する。また、インタラクタ・ウィジットの種類をより豊富にし、Duma におけるデータレポジトリを MMDEF の UI データベースとして位置づけ、その検索や内部データ表現の研究に取り組む。

更に重要な項目として、インタラクタ・タイプ自身の構造について改良を加え、柔軟な対話表現を構築可能とすることが必要であると考える。その一アプローチとして、インタラクタにコマンド言語およびそのウィジットである Tcl・Tk [9] を埋めこむ作業を進めており、その成果が期待される。

最後に、本研究に対し多大な協力を頂いたアンサンブルプロジェクトのメンバーに深謝いたします。

[1] GNU flex, bison, g++ genclass 等が生成したコード量を除く。

[2] 文献 [6] を参照されたい。

参考文献

[1] Gideon Avrahami, Kenneth P. Brooks, and Marc H. Brown. A two-view approach to constructing user interfaces. *Computer Graphics*, 23(3):137–146, July 1989.

[2] Eric A. Bier. EmbeddedButtons: Documents as user interfaces. In *proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, pages 45–53, November 1991.

[3] Microsoft Corporation. *Microsoft WORD User's Guide*. Microsoft Corporation, Redmond, WA, 1992.

[4] Susan L. Graham, Michael A. Harrison, and Ethan V. Munson. The Proteus presentation system. In *ACM SIGSOFT symposium on software development environments*, pages 130–138, Tyson's Corner, VA, 1992.

[5] Jeff Johnson. Selectors: Going beyond user-interface widgets. In *proceedings of CHI '92 Conference: Human Factors in Computing Systems*, pages 273–279, NY, May 1992.

[6] Brad A. Myers and Mary Beth Rosson. Survey on user interface programming. In *proceedings of CHI '92 Conference: Human Factors in Computing Systems*, pages 195–202, NY, May 1992.

[7] College of Engineering. *EECS/ERL 1993 Research Summary*. The Regents of University of California, Berkeley, CA 94720, 1993.

[8] Takashi Ohtsu. Duma: a data-based user interface management system for multimedia application. Master's thesis, Computer Science Division, University of California, Berkeley, Berkeley, CA 94720, October 1992.

[9] John K. Ousterhout. An X11 toolkit based on the tcl language. In *USENIX Summer Conference Proceedings*, pages 105–115, 1991.

[10] Derluen Pan. Mosaic: A framework for an extensible media editor. PhD thesis proposal, September 1992.

[11] John L. Pasalis. Realize: An interactive graphical data structure presentation and rendering system. Master's thesis, Computer Science Division, University of California, Berkeley, Berkeley, CA 94720, 1992.

[12] Gunther E. Pfaff, editor. *User Interface Management Systems*. Spring-Verlag, Berlin, 1985.

[13] Andre Weinand, Erich Gamma, and Rudolf Marty. ET++ – an object-oriented application framework in C++. In *OOPSLA 1988 Proceedings*, pages 46–57, September 1988.

[14] Charles Wiecha, William Bennett, Stephen Boies, John Gould, and Sharon Greene. ITS: A tool for rapidly developing interactive applications. *ACM Transactions on Information Systems*, 8(3):204–236, July 1990.