

セットトップボックス(STB)や デジタルテレビ等の 電子番組ガイド(EPG)におけるデータ管理

はじめに:

電子番組ガイド(EPG)は、デジタルテレビにおいて最も高度に進化した対話型のブラウジング・サービスを提供し、ユーザがプログラム・リストや場合によっては特定コンテンツへの制御アクセスを検索し、フィルターし、カスタマイズすることを可能にします。セットトップボックスの新規開発において、これらの機能を実現するにあたって、データ管理の手法を検討することが重要になってきました。確実なソリューションを模索した結果、セットトップボックスメーカー数社は既に市販のデータベース技術を取り入れ、自社のセットトップボックスに組み込みはじめました。

本ホワイトペーパーでは、EPG におけるデータ管理の手法を考察し、開発者の方々へのより良いソフトウェアの提案を行います。

また、本資料においては、デジタルテレビの新基準とセットトップボックス技術概要を描き、データ管理に求められる機能を考察し、番組ガイドにおける典型的なデータオブジェクトおよび各オブジェクトの相互関係を示します。ソリューションを検討していく際に、データ管理の性能を改善し、合理化された設計によってリソース要求を最小限にする比較的新しいデータ管理手法(インメモリデータ・ベース・システム(IMDS))に焦点を当てていきます。コード例およびサンプル・データ・ベーススキーマは、市販のデータ・ベース・システムを利用することにより高められた、セットトップ・ボックスにおけるデータ管理効率に注目します。

ガイロジック株式会社

<http://www.gailogic.co.jp/db>

db@gailogic.co.jp

Copyright 2002, McObject LLC

序章：

デジタルテレビはネットワークを利用したマルチメディア技術のうち、間違いなく最も進んでいます。他のアプリケーションが主流へ移動するために躍起になっている一方、デジタルテレビは既に何百万ものユーザに何百ものチャンネル、明瞭な画像およびCD よりよい品質のオーディオを提供しています。アナログからデジタルへの移行は10年もしくはそれ以上かかるかもしれませんが、結局はすべての放送局がデジタルの領域にシフトしていくと考えられます。従来の受動的なビデオに慣れたユーザはデジタルテレビのインタラクティブなデータサービスにしばしば驚きを隠せません。中でもオーディオ・ヴィジュアルデータのコンテンツサービスが注目されています。

電子番組ガイド(EPG)はチャンネル数が多いデジタルテレビの中で、最も重要なデータ貯蔵場所の1つです。EPGにより、ユーザは現在利用可能なチャンネルを確認し、リモコンから番組リスト上をポイントすることによって現在の番組を受信することができます。また、他にEPGが提供する機能としては、ユーザが設定した特定の基準に合わせて、ユーザにとって最も関心のある番組を表示したり、数日先の番組情報を提供する機能などがあります。EPGはもはや未来の夢ではありません。ケーブル受信機と衛星受信機により今日提供されている機能です。映画/スポーツ/ニュース/ビジネス/家族/子供向け/教育などさまざまなテーマで番組を選ぶことができます。簡単な操作で好みのチャンネルリストを作成したり、リモコン上のボタン1つでEPGを表示することが可能です。受信機の機種によっては、レーティングに合わせて特定のコンテンツをブロックし、表示しないようにすることができます。

システム設計者は、これらの機能には検索、フィルタ、ソート、選択および保存など重要な諸機能が伴うものとして認識しています。実際に、他のどの機能よりもEPGがデジタルテレビ受信機に最新のデータ管理コンポーネントを追加しています。ちなみにデジタルテレビ受信機は(IRD: Integrated Receiver Decoders)としても知られています。肥大化するEPG機能をサポートし続けるには、受信機の組込みソフトウェアにデータベース「層」を組込むことによって、開発者は最適なデータ設計、トランザクションおよびデータ保全性のサポート、パフォーマンスおよびCPUとOS利用状況に対するデータ管理タスクのインパクトの最小化などを図る必要が増しています。

従来開発者はこのような要求に自製のデータ管理コンポーネントで対応していました。このような解決法も時にはパフォーマンスやメモリフットプリントなどの要求を受けません。しかし、扱うデータ量が増大した場合、セットトップボックス自身に対する要求が増えた場合など、自製のデータ管理ではアップグレードが困難なことから、QAがないがしろになったり、開発サイクルが予定よりも長引く場合があります。

IRD開発者は今後ますます長年にわたって培われた安定性、拡張性およびスケーラビリティを持つ商用のデータベースに注目していくと考えられます。しかしながら、多くの場合、商用のデータベースはリアルタイム性を欠くことが理由でデジタルテレビ受信機開発からは除外されることがあります。これは民生機器であるデジタルテレビのユーザはネットサーフィンのように、複数の番組間を「サーフィン」する際に待ち時間があるとは思わないからです。典型的なデータベースは強力な処理能力をプラットフォーム側に求めますが、これではデジタルテレビ向けのデータベースとは言えません。IRD開発メーカーはコストを最小化するためにRAM容量およびCPUパワーも最小化するからです。

さて、セットトップボックス等に市販のデータベースを利用する時に直面する問題は市場に存在するデータベースと名のつく製品が多く、情報が氾濫していることです。セットトップボックスの設計者は多くの場合、データベースの専門家ではありません。本ホワイトペーパーにおきましては、セットトップボックス開発者のデータベース選びをお手伝いするために、いわゆるEPG データベースにより定義される典型的なデータオブジェクトと各オブジェクト間の相互関係およびIRD が要求するデータ管理について考察を進めます。具体的には市販のデータベースのうち、比較的新しい部類のデータベースであるオンメモリデータベースについて考察を進めます。オンメモリデータベースは従来の（ディスクを利用した）データベースにともなう不要なオーバーヘッドのほとんどを排したEPG に最適なデータベースです。

ローカルなデータ格納場所としてのEPG

EPG データはケーブルもしくは衛星等からリアルタイムに送信される番組情報から構成されます。EPG ストリーム中の情報更新サイクルはさまざまで、場合によっては数時間のような長い場合もあります。このような理由により、セットトップボックスはユーザによる操作やEPG 情報からのクエリがあった際にデータを「待つ」ことはできません。実際にはセットトップボックス（受信機）は情報を事前に収集し、保持しているのです。EPG データが受信され、セットトップボックスのDRAM 中にデータがローカルに格納されることにより、ユーザは番組の検索、フィルタ、ソートおよび選択をすることが可能になります。デジタルテレビのユーザが通信状況による遅延を感じないようにしているのは番組ガイド情報をローカルに（受信機内に）格納しているからです。

諸規格の概観

デジタルテレビ向けのMPEG-2通信および処理標準規格において、データフォーマットについての定義はないものの、EPGデータ搬送用のプライベートな放送メカニズムが定義されています。米国においては、Advanced Television Systems Committee (ATSC)が” *ATSC A/65 Program and System Information Protocol (PSIP) for Terrestrial Broadcast and Cable.* ” という標準規格を発行しています。この規格ではMPEG-2データストリーム中でEPGデータが搬送される方法が定義されています。欧州では、the European Telecommunications Standards InstituteがDigital Video Broadcast (DVB) 規格を発行し、同様のニーズに対処しています。

上記の各規格はEPGに対して、番組の放送局がPSIP（米国）もしくはDVB（欧州）プロトコルに準拠しながらデジタルビデオストリームの中に番組紹介データを載せる方法を定義しています。これらの規格が作られた意図は、異なるメーカーの受信機が同じデータを元に包括的な番組ガイドを作成することを可能にすることです。米国および欧州市場向けに作られた他の規格と同様、PSIPとDVB間の相互運用はできません。EPGデータは任意の番組の全デジタルコンテンツのうち、せいぜい0.5～3%しか占めませんので、MPEG-2ストリームでATSCおよびDVB両方のプログラムガイドデータを伝送することは可能です。実際にはこのようなことは行われていません。ひとつのケーブルシステムが多数の異なるMPEG-2ストリームを伝送する場合や、MPEG-2ストリームから番組紹介データをリアルタイムで取得するのに時間がかかったり、この仕組みがシステム依存である場合があります。このような理由により、米国内のデジタル・ブロードキャスティング・ネットワークではEPGデータを受像機に送るのに帯域外のデータ伝送線を使用しています。

EPGのために策定されたATSC基準

U. S. MPEG-2デジタルビデオシステムに関して、ATSC A/65標準規格において、仮想チャンネルの導入により単一RFチャンネル上に複数番組を載せる場合を解決しています。ユーザにとって仮想チャンネルは通常のチャンネルのように見えますが、実際にはFCCが各チャンネルに割り当てた周波数で伝送されているではありません。A/65では周波数、変調方法、テキスト名、チャンネルのタイプ（アナログオーディオ/ビデオ、デジタルa/v、オーディオのみ、データ）およびユーザがアクセスする際使用するチャンネル番号を規定しています。

放送局は自身のチャンネル番号をブランド構築のための投資と考えています。実際に放送局のロゴマークには通常チャンネル番号が入っています。仮想チャンネルには「2部構成」チャンネル番号の概念があり、各放送局は元からある主要チャンネルのほかに2時的なチャンネルを持つことができます。デジタルテレビ放送に際し、「2部構成」チャンネル番号のうち主要な「メジャー」チャンネルと呼ばれるチャンネルはアナログ放送で使われるEIA/FCCと同様である必要があります。「2部構成」チャンネルの残りである「マイナー」チャンネルはメジャーチャンネルにより定義されるチャンネル内の複数チャンネルから特定のサービスを認識するために使われます。ユーザから見れば、以前は例えばチャンネル番号「4」のみのところが、「4. 1」、「4. 2」、「4. 3」と続く番号ができることとなります。

上記の Protokol では、放送に含まれるペアレンタルアドバイザリ・データも標準化しています。A/65標準規格には地域ごとにアドバイザリシステムを定義しているRating Region Table (RRT) (番組の分類に関する地域情報テーブル) および、特定の番組イベントをRRTにおいて定義されている各付けのレベルに関連させるのに使われているコンテンツ・アドバイザリ・ディスクリプタが含まれます。このようにして、地域ごとにことなる基準を適用することを可能にしています。

今日のEPG ではケーブル、衛星およびローカル番組情報がすべて1つの読みやすい番組ガイドとしてシームレスにまとめられています。包括的なEPG と呼ばれるものには次のものが含まれています：

- ・衛星、ケーブルもしくは地上波ソースからのローカルチャンネル
- ・タイトル、番組概要、カテゴリ、ペアレンタルアドバイザリシステムによる分類、俳優、監督、プロデューサ、批評家による各付けなどの構造化された情報。これにより、受信機は情報内の関連した項目を見つけることができます。このように情報を構造化することにより、IRD はより使いやすいユーザインタフェースおよび便利な機能を提供できるようになります。
- ・多言語サポート。実際に、すべてのテキストフィールドは複数の言語で表されることがあります。1台のIRD にて、任意のテキストについて利用可能なすべての翻訳文を保持することも、ユーザにより選択された言語の翻訳文のみを保持することもできます。
- ・ウィングのサポート。ウィングではビデオとともに包括的なインターアクティ

ブアプリケーションを提供しています。ウイंकに関する詳細はwww.wink.comをご参照下さい。

- ・ フォーマット化されているテキスト。フォーマット形式としては、単純なテキストから完全なHTML までさまざまなレベルのマークアップ言語の形式が含まれます。IRD はEPG 運用者のルールに基づいてマークアップ言語を解析します。

動作環境

ケーブル、衛星および地上波デジタル受像機は通常、比較的小さな容量のDRAM を主要ストレージとした組込みシステムとして構築されます。セットトップボックスはPCやワークステーションで使用されているようなOS ではないOS で動作している場合があります。マルチタスクのリアルタイムOS によってのみ、IRD に必要な高速データ転送および高いパフォーマンスを実現することができます。

一方では、セットトップボックスが機能面で進化するのに伴い、組込みソフトウェアはより強力になり、1つのシステムに複数のアプリケーションが存在するようになりました。表示機能および基本的なテレビ機能のほかに、Eメール、ビデオ・オン・デマンド、パーソナル・ビデオ・レコーダなどさまざまなアドイン機能が出現しています。これらの機能はEPG とともにインテグレートされている場合があります。このような場合、EPG はタイムリーにデータを格納するだけでなく、他の機能に検索機能を提供する必要があります。このような場面で、EPG にはデータベースが必要となります。この場合、データベースはアプリケーションの下に位置するデータ格納場所として機能したり、EPG へのAPI を供給する役目を担います。この基盤となるデータベースはOS とシームレスにインテグレートされていなければなりません。またデータベースはデータ格納や検索に関する複雑な要求に耐えられるよう強力なものである必要がある一方で、CPUを独占してはいけません。このCPUを独占しないという特徴はパーソナル・ビデオ・レコーダなど他のタスク処理の上で必要となります。

ストレージ管理要求：EPGデータ

オンスクリーン番組ガイドは、大きく分けて次の2つの部分から構成されます：1) EPGプレゼンテーションエンジンと呼ばれることもあるアプリケーションソフトウェア（セットトップボックス中で動作します）2) アプリケーションソフトウェアが受け取り、表示用にフォーマットするデータ。EPGが実装されているセットトップボックスは外部ソースから頻繁にガイドデータを更新される必要があります。IRDは番組ガイドデータを取り出し、メモリ (DRAM) に格納します。また、フルスクリーン・グリッド・ガイドもしくはアクティブ・ビデオ上にオーバーレイ表示するためにデータを保持します。

この意味において、EPGストレージ管理サブシステムはEPGコンテンツとメモリリソースの使用に際して競合します。なぜなら、アプリケーションのメモリフットプリントの最小化によりコンテンツが充実し、エンドユーザの満足感が向上するからです。

上記の標準規格に基づき、IRDはEPGコンテンツを形成するために以下のデータ・オブジェクトを保持します。

チャンネル関連データ。

衛星、ケーブル、地上波いずれのチャンネルに対しても1チャンネルが割り当てられます。

各チャンネルは最少でも次の点で特徴を持つことになります。

- 各サービスへのアクセスのための2部構成チャンネル番号（メジャーおよびマイナー）
- テキスト名（最大7文字まで）
- サービスの物理的な伝送形式（周波数、変調モード）
- チャンネルの「ソースID」（各国プログラムソースデータベース中の固有名を参照します。）
- サービス形式（アナログテレビ、デジタルテレビ、音声のみ、データのみ）

各チャンネルに特有の他のタイプのデータは、サービスがアクセスコントロールされているかどうかを明示するフラグ、および「拡張テキスト」がサービス(オプションのディスクリプタ)のテキスト記述を提供することができるかどうかに関する表示を含んでいます。

イベントデータ。

イベントにより各チャンネルのスケジュール情報が伝送されます。各イベントは次の要素により構成されます：

- イベント開始時間
- イベント持続時間
- イベントタイトル（テキスト）
- オptional追加ディスクリプティブテキストへのポインタ。下記のプログラムデータにはシノプシス、キャスト、ディレクタ等が含まれます。
- プログラム・コンテンツ・アドバイザー・データ（オプション）
- キャプション・サービス・データ（オプション）
- 使用可能な言語をリストアップ可能なオーディオサービス・ディスクリプタ（オプション）

イベントは通常スケジュール内で構築されます。スケジュールにはスタート時間とイベントのリストが含まれます。各チャンネルのスケジュールの完全なものには互いにリンクした多数のスケジュールにより構成されます。

プログラムデータ。

プログラム情報にはプログラムタイトル、オプションによりプログラムの概要、カテゴリ、分類情報および条件付きアクセス情報が含まれます。

レーティング地域データにより任意の地域の「レーティングシステム」を定義します。レーティングシステムは多数のレーティングディメンジョンにより特徴づけられます。各ディメンジョンは複数のレーティング・レベルで構成されます。ケーブルで使われるレーティングディメンジョンの典型的な例はMotion Picture Association of America (MPAA) システムです。MPAAディメンジョン内のレベルには「G」、「PG (ペアレンタル・ガイダンス)」、「PG-13」、「R」および「NC-17」があります。

カテゴリデータ。

カテゴリシステムにより、EPGが使用するプログラムカテゴリの階層構造と概要情報が定義されます。EPGのライフサイクル内でもカテゴリシステムは動的に変更されます。新しく更新された定義はEPGに送られ、挿入される必要があります。

今日のIRDの中には次のような追加データが存在します：

- お気に入りチャンネルリスト—この機能により、ユーザはEPGを編集し、興味あるチャンネルだけ表示するようにすることができます。
- EPG初心者のためのヘルプ画面。新しい機能がダウンロードされて必要である場合に更新されます。
- ケーブルシステムおよびローカルオフ=エアプログラミング。米国内には約12,000のケーブルシステムが存在します。IRDはこのすべてに対して、適切なサービスを提供する必要があります。
- 画像や音声など、IRDに対して受信することが要求され得る広告および他のマテリアル
- ウィンクサポートデータ

ストレージ管理必要条件：検索能力

これらのデータオブジェクトおよび受信機に必要な機能に対して、下に位置するデータ管理モジュールは以下のような検索メソッドを露出する必要があります。

- 通常、EPGデータ要素（オブジェクト）には独自の識別子が割り当てられます。識別子はプログラムが関連プログラムデータを結合する際に使用されます。データ管理モジュールにはこの識別子に基づいた高速の検索メカニズムが求められます。
- ソースID検索メカニズム。ソースIDにより各チャンネルの独自プログラミングソースが識別されます。各チャンネルのスケジュールは共通のソースIDにより関連付けられたプログラム情報により完全なものとして構築されます。各チャンネルのプログラミングデータを取得するために、IRDはEPGデータベース中にソース

IDに基づいて全てのプログラミングソースを配置する必要があります。

- 取得された全プログラムもしくはチャンネルのリストなど、同じ型のオブジェクトのシーケンシャルフェッチ。
- メジャーおよびマイナー番号もしくはチャンネルコールサイン（タイトル）によるチャンネルのソート。ストレージ管理には実行時にユーザの好みにより数字順もしくはアルファベット順でチャンネルのソート順序を変更可能なメカニズムが要求されます。
- 全チャンネルにわたっての番組放映時間などの代替ソートメカニズム。

ストレージ管理への要求事項：メモリ管理

ローカルEPGデータストレージに利用可能なメモリ容量は通常、非常に制限されています。一般的に、EPG処理およびEPGデータ格納にRAMを多く割くことができれば、IRDの性能は向上します。RAMの使用状況を左右するIRDの機能（特長）としては次のものが挙げられます：何日分の番組情報を保存するか、番組内容紹介記述の量、サポート言語、グラフィックス。このような状況を考えると、ストレージ管理モジュールはIRDに実装されているメモリを効率的に使用する必要があります。ストレージ管理は多様なデータ・アライメントをサポートする必要があります—例えばEPGデータの多くは1バイト構成です。現実として、EPGデータの約90パーセントはダイナミック・テキストです。ストレージ管理は可変長文字列など動的なデータをメモリを過度に要求するデータ形式で格納すべきではありません。このような方法では、容易にメモリの限界容量を消費してしまいます。

放送サービス・プロバイダは通常IRD に対して、3 日、7 日もしくは14 日分の番組データを保存するように要求します。3 日分の場合は、IRD は常に将来の2 日分と当日分の番組データを蓄えることとなります。EPG は「境界条件」を上手に扱う必要があります。データ格納用のメモリ領域を使いきったときには、データ管理はEPG に「クリーンアップ」プロシージャを実行させて期限切れデータを削除し、再利用のためにメモリをメモリプールに戻します。組込み用のOS には効率的にメモリを使用するメモリマネージャが無い場合もあるので、EPG はOS 非依存のメモリ管理を実装していることが理想です。

ストレージ管理への要求事項：マルチスレッドアクセスとトランザクションメカニズム

EPG データベースは同時に複数のスレッド（タスク）からアクセスされます。スレッドの例としては、EPG データベースへの新しいデータの書き込み、ユーザインタフェースからのクエリー、条件アクセスクエリー、ウィングサポートなどがあります。新しいデータが得られ次第常駐領域に書き込みを行うタスクなど、タスクの中にはデータベースへのアクセスに際して優先順位を割り当てる必要のあるものがあります。この場合、優先順位が低いタスクはバックグラウンドで、もしくはCPUのアイドル時間に実行されます。このように、ストレージ管理モジュールにはEPG へのマルチスレッド機能とトランザクション優先順位づけメカニズムの実装がもとめられます。

ストレージ管理への要求事項：パフォーマンス

EPG データは高速に伝送されなければなりません。実際の伝送速度はプロバイダにより

異なります。しかし、利用可能な帯域を最大限に利用することを試みているのは全てのプロバイダに共通しています。IRD のソフトウェアはEPG データベースに受信パケットデータを格納する前に、受信パケットの処理を行います。このときの処理としては、フレームからのEPG データの組み立て、データの解凍、フィルタリングなどが含まれます。一般的に伝送速度は数十メガビット/秒です。これに伴い、ストレージ管理にはメモリ指向の検索アルゴリズムに加え、特別な高速トランザクション制御メカニズムが必要になります。この他にも、EPG データベース管理モジュールにはCPUを独占しないことが求められます。なぜなら、ビデオ録画などIRD の他の重要な機能も同様にCPUサイクルを取得する必要があるからです。EPG ストレージアルゴリズムにはCPUサイクルの利用を最小限に抑えるような工夫が必要です。

インメモリ・データベースシステム：EPGデータベース管理のためのレシピ

EPGデータベースを維持していくことは複雑かつ難しい役目です。歴史的に、組込み以外の分野では、複雑なデータをデータベースで管理することにより、データの保全、データ間の複雑な関係の構築、迅速で効率的なデータアクセスなどを定型的な手法で実現してきました。自製のコードの代わりに、確立されたデータベースAPIを使用した市販のデータベースを導入することにより、コーディング、デバッグ、メンテナンスなど開発者の負担を軽減することが可能です。しかしながら、組込み以外の分野で使われていた「伝統的な」データベースはCPUおよびメモリを多量に消費します。また、このようなデータベースにはリアルタイムパフォーマンスを期待することはできません。

一方で、データベースの新しい技術であるインメモリ・データベース(IMDS)は従来のデータベースにあった機能のうち、不要なものは削除し、また他の機能についても合理化を進めた設計により、要求がタイトである組込みシステムに最適なデータベースになっています。設計当初より、メモリ上での実装のみを考慮しているため、IMDSはディスクI/Oやキャッシュなどオーバーヘッドの大きな機能を排除し、高速な処理を実現しています。このようなIMDSの一つであるMcObject社のeXtremeDBはセットトップボックスを含む組込み機器に最適な設計になっています。具体的な例をご覧いただくために、本ホワイトペーパーの残りの部分では、実際にeXtremeDBを使用してセットトップボックスのデータ管理層にIMDSをインテグレートした場合のメリットを説明致します。特定のデータ型のサポート、オブジェクト間の関連付け、検索手法など、EPGの要求に応えるためにも開発の段階でeXtremeDBはその価値を発揮します。IMDS特有のメリットではないものもありますが、これらの機能のサポートはEPGを開発する上では切実な要素です。これらの機能については、IMDSの説明の後に詳述します。

メモリ管理：メイン・メモリ・データベース

上記のとおり、EPGデータ管理は高い伝送速度のサポートを要求します。またデータベースにはCPUとメモリの使用を最小限に抑えることが求められます。ディスクベースのデータベースと比較すると、eXtremeDBは以下のような方法で、メモリ容量および関連するオーバーヘッドを排除しています：

- アプリケーションとデータベース間をつなぐためのオーバーヘッドはありません。eXtremeDBはEPGのコードとしっかりリンクしています。
- 余分な層を排除—データがメモリ上に存在することを前提とした設計のため、

eXtremeDBはデータ管理を大幅に合理化し、ディスクベースのDBMSにありがちなオーバーヘッドを排除しています。

- 検索アルゴリズムをメモリアクセスに対して高度に最適化（ディスクベースのデータベースはディスクI/Oを最小化するように最適化されています）
- 検索用の翻訳処理－eXtremeDBはメモリ上の各データ要素が保存されている場所を直接ポイントします。対照的に、従来のDBMSはブロック番号とオフセット値をポイントします。このようなデータベースではブロックの場所を特定し、メモリにロードしメモリバッファ中の適当な場所を見つける必要があります。
- eXtremeDBではバッファ管理の必要がありません。対照的に、従来のDBMSではディスクからの新しいデータはメモリバッファ内のデータを書き換えます。従って常にメモリバッファの内容をディスクにライトする必要があります。
- eXtremeDBはデータへの直接アクセスを提供します。ディスクベースのDBMSではアプリケーションがメモリバッファ中のデータ要素にアクセスすることはありません。そのかわりに、データはメモリの他の場所にコピーされます。その結果オーバーヘッドは増加します。

メモリ管理：メモリマネージャ

eXtremeDB は限られた容量のメモリで動作するように設計されています。eXtremeDB の独自メモリマネージャはデータおよびインデックスレイアウトのオーバーヘッドを最小化するように最適化されています。eXtremeDB ランタイムはOS のmalloc()とfree()のいずれも使用していません。malloc()およびfree()はリアルタイムOS の環境ではメモリ使用の点で非効率である場合があるので、eXtremeDB では高度に最適化された独自のメモリマネージャを採用しています。メモリマネージャはオブジェクトのパッキングを行い、オブジェクトのサイズを小さく抑え、ある範囲でアプリケーションからの制御を可能にします。

トランザクションおよびデータへのマルチスレッドアクセス

データベースは動作環境を最大限利用する必要があります。EPGのデータ管理はフルにACIDをサポートする必要がありますが、同時にデータベースはセットトップボックスの組込みマルチタスク環境を活用する必要があります。この状況では、同時に実行されるタスクは数多くありません。また、同時に実行されるタスクがすべてライトオブジェクトへのライトアクセスを要求することはほとんどの場合ありません。したがって、現実の要求を考慮すると、複雑なトランザクションの同期化を実現するよりも、メモリフットプリントを最小化し、ライトトランザクションをシリアライズすることによりトランザクション・マネージャを簡略化することが重要です。

eXtremeDBでは同時に複数のリード要求を許容しています。すなわち、複数の「リード」トランザクションを容認しています。「ライト」トランザクションはデータベース・ランタイムへの排他アクセス権を持ちます。協調動作に関するこのようなアプローチにより、オーバーヘッドを伴うロックメカニズムやデッドロック防止のための複雑な問題をデータベースの保全性を保ちつつ、回避することを可能にしています。ライト・トラン

ザクシヨンのシリアライゼーションはEPGプログラマからみてトランスペアレントです。トランザクシヨンのマネージャは「軽い」ので、適切に設計されかつ実装されたトランザクシヨンは迅速に実行され、シリアライゼーションはパフォーマンス上の問題にはなりません。

eXtremeDBではトランザクシヨンの優先順位付けをサポートしています。実行時に各トランザクシヨンに対して、優先順位に応じた値を割り当てるのが可能です。このトランザクシヨンへの優先順位付け機能を使うことにより、EPGは必要であれば、実行時にトランザクシヨンの実行を「ブースト」することができます。（例えば、高速伝送中に転送バッファのクリーンアップを行うことなどが挙げられます。）

eXtremeDB によるEPG データのマッピング

特定のネイティブ・データ型、オブジェクト間の関係および検索手法が利用可能であることにより、効率的なEPGを構築することが他の状態に比べ、大幅に容易に行うことが可能です。例えば、EPGデータは多くの場合、ツリー構造になっています。従来の（ディスクベースの）データベースがこの問題を単純なbツリーで対応するのに対し、eXtremeDBでは、複合データタイプの関係を実装するのに適した強力なツールを用意しています。具体的には、eXtremeDBはベクタ、構造体およびオプションのデータフィールドをサポートしています。ベクタとは固定型データの任意の大規模なストリームです。構造体の宣言では型を示し、異なる型を持ち得る要素を特定します。構造体および単純な型はオブジェクトの定義を構築するためにブロックを構成します。構造体は他の構造体の要素として使用可能です。

これらのデータ構造はネイティブEPGデータの表現を飛躍的にシンプルにします。これらによって、EPG情報は通常データオブジェクトに分割され、クラスに組み入れられます。オブジェクトクラスにはチャンネル、スケジュール、プログラム（番組）などが含まれます。各オブジェクトは型によって定型化されます。しかし、一般的にはオブジェクトは基本要素とディスクリプタと呼ばれることもある付加要素とから成り立っています。基本要素はオブジェクトの型に従って厳格な構造で定義されています。付加要素は多くの場合、オブジェクトの基本部分に付加されます。付加要素が無い場合もあれば、多くの付加要素が付加される場合もあります。通常、識別する目的で、すべてのEPGオブジェクトは共通プールにおいて、固有の整数識別子が割り当てられます。この番号は *object_id* と呼ばれており、オブジェクトの持続期間にオブジェクトのシリアルナンバーの役目を担います。

チャンネル・オブジェクトの仮想的な記述を考えてみましょう。チャンネルは目的地であります。通常は番号、名前、ロゴであり、ユーザからはテレビ番組にアクセスする際の1つの存在として認識されます。例1に図示されているチャンネルオブジェクトはチャンネルとそのコンテンツを識別しています。

```
channel_object
{
    object_id;
    time_acquired;
    source_id;
```

```

    major_number;
    minor_number;
    short_name_size;
    short_name();
    indicator;
    if (indicator == 1) {
        expression_size();
        expression();
    }
    descriptors()
    {
    [text | category | channel_content | audio_service ]
    }
}

```

例1. 典型的なチャンネル・オブジェクト

上記の中間コード中のフィールドの定義は以下のとおりです:

object_id	この32 ビットフィールドがオブジェクトを唯一の存在として識別可能にしています。
time_acquired	オブジェクトがIRD に取得された回数
source_id	この16ビットフィールドがチャンネル毎のプログラミングソースを識別可能にしています。source_idによりIRDはチャンネルをスケジュールにリンクします。
major_number	この16ビットフィールドにより、当該チャンネルのメジャー番号が識別されます。
minor_number	この16ビットフィールドにより、当該チャンネルのマイナー番号が識別されます。
short_name	チャンネルのショートネームを表す文字列のバイトデータ
indicator	1ビットのフィールド。セットされている場合は、オブジェクトは関連した表示を含んでいることを表します。この表示は条件付きアクセスエンジンがオブジェクトをガイドに入れるか、排除するかを決定する際に使用されることがあります。
text	このフィールドは当該チャンネルのテキストによる記述です。必ずしも存在するフィールドではありません。
Category	このフィールドがある場合は、当該チャンネルの分類に使われます。
channel_content	このフィールドがある場合は、当該チャンネルの代替情報を提供します。
audio_service	このフィールドがある場合は、当該チャンネルで利用可能な言語の情報を含みます。

以下の例2においては、上記の部分コード用のeXtremeDBスキーマがチャンネルオブジェクトを定義しています。

```

struct ID {
    uint4 object_id
}

```

```

};
declare oid ID [200000];

/* Class Channels
*/
struct Expression {
    string str_expression;
};
/* These descriptors are used in various objects */
struct Text {
    string str;
};
struct Category {
    . . .
};
struct AudioS {
    . . .
};
compact class channel {
    mco_time time_acquired;
    uint2 source_id;
    uint2 major_number;
    uint2 minor_number;
    string short_name;
    uint1 indicator;
    optional Expression opt_expression;
    /* descriptors */
    optional Text this_text;
    optional Category this_category;
    optional Content this_content;
    optional AudioS this_audio;
    tree <major_number, minor_number> chan_number;
    tree <short_name> chan_name;
    oid;
};

```

例2.

eXtremeDBのデータ定義言語(DDL)はC言語の構造体、動的な文字列、オブジェクト識別子およびインデックスをサポートしています。上記の例で修飾子*optional*がどのように使われているかについて、注目してください。Expressionフィールドをはじめ、すべてのディスクリプタが*optional*として宣言されています。optionalとして宣言されることの意味は、当該のフィールドはデータベースに格納される場合とされない場合があることを表しています。フィールドが格納されない場合は、ランタイムはデータレイアウトにこのフィールド用のスペースを確保しません。compactクラス修飾子はクラス要素のトー

タルサイズを64Kに制限します。クラス要素にはEPGのロウデータだけでなく、eXtremeDBが必要とするオーバーヘッドも含まれます。しかし、compact宣言はクラスデータを維持するのに必要であるオーバーヘッドの削減を飛躍的に行います。(オブジェクトが64K内にフィットすることがわかっているため、eXtremeDBでは4バイトのオフセットではなく2バイトオフセットを使用することができます。この2バイトの節約はオブジェクトの数が数千にもものぼる場合を考えると、重要な節約であることがわかります。)

EPGには通常、番号順もしくはコールサイン順にソートされているチャンネルのリストを表示することが求められます。上記のコード片ではチャンネルクラスに対してインデックス2つを定義しています。chan_numberインデックスはメジャーおよびマイナー番号によるソート/検索を提供し、chan_nameインデックスはショートネームによるソート/検索チャンネルを提供します。EPGはソートされたチャンネルリストをナビゲートするために、および2つのインデックス間を迅速にスイッチするためにeXtremeDBのカーソルAPIを使用することができます。

不要な通信を回避し、かつリソースを節約するために、EPGのオブジェクトは保存されるのは一回でも、他の多くのEPGオブジェクトから参照できる必要があります。このようなコンテンツの中には、内容の記述がなくまた、放映するたびに毎に内容に差異がないニュース番組などがあります。EPG送受信の際にニュースが送信されるのは1回かもしれませんが、しかし、毎夕6時に参照されるかもしれませんが。このような情報を参照可能な状態に維持する場合は、専用のオブジェクト識別子を使います。eXtremeDBははじめからオブジェクト識別子をサポートしており、**oid**に基づいた超高速な検索機能をはじめ特別な「参照」**-ref**データ型を提供します。上記の例ではchannelクラスは**oid**を使用して宣言されています。このことは、サービスプロバイダによりすべてのEPGオブジェクトに割り当てられた専用のオブジェクト識別子に相当します。eXtremeDBランタイムは実行時にobject idの「独自性」を行使し、他のEPGオブジェクトからのチャンネルオブジェクトの参照を容易にします。

木構造を構築する際には、多くの場合、EPGオブジェクトは構造体の可変長アレイを含みます。例えば、以下の例3におけるschedule objectはチャンネルスケジュールの一部を定義しています。多様なschedule objectが1つの完全なスケジュールを作り上げています。schedule objectはsource_idを使って参照チャンネルに関連付けられています。加えて、各schedule objectは開始時間、持続時間およびイベント数の情報を持っています。

object_id	この32 ビットフィールドによりオブジェクトが特定のものとして、識別されます。
time_acquired	IRD がオブジェクトを取得した時間
source_id	この16ビットフィールドにより、このスケジュールオブジェクトにあるプログラミングソースが識別されます。
start_time	この32ビットフィールドにより、スケジュールの開始時間が識別されます。
duration	当該オブジェクトにより提供されるスケジュール情報の総時間数を表示します。
number_of_events	この8ビットフィールドはスケジュールのイベント数を表示します。

event_start_time イベントの開始時間
event_duration このフィールドはイベントの持続時間を表示します。
program_object_id この32ビットフィールドは当該時間帯のプログラムオブジェクトのobject_idを定義します。

```

schedule_object
{
  object_id;
  time_acquired;
  source_id;
  start_time;
  schedule_duration;
  number_of_events;
  for (i = 0; i < number_of_events; i++)
  {
    event_start_time;
    event_duration;
    ref program_object_id;
  }
}

```

例3.

eXtremeDBはベクタを使ってスケジュールを表現します。アプリケーションにあるオブジェクトモデルが、スケジュール内のイベントのような既にはじめから動的な構造の項目を含む場合はベクタを使用すると便利です。例4ではスケジュール化されたイベントを組み立てる際にeXtremeDBがどのようにベクタを使用するかを示しています。

```

struct Event {
    time start_time;
    uint2 duration;
    ref program_id;
};
compact class schedule {
    time time_acquired;
    uint2 source_id;
    uint4 start_time;
    uint2 duration;
    vector <Event> events;
    tree <source_id, start_time> lineup;
    oid;
};

```

例4.

プログラムデータを表現するために、eXtremeDB は例5 のようにクラスプログラムを定

義します。構造体Event が相当するプログラムのoid を参照していることに注目してください。各番組はさまざまなチャンネルからさまざまな時間帯に送られます。しかし、番組に関連付けられているデータである番組の内容に関する記述や番組の分類などがデータベースに格納されるのは一度です。また、分類システムに関する記述は簡略化されていることに注意してください。分類システムの詳細についてはATSC 標準規格をご参照下さい。

```

struct dimension {
    uint1 level;
    uint1 value;
};
struct rtt { /* rating region table */
    vector <dimension> dim;
    string text; /* the text that describes this rating dimension
*/
};
struct content {
    vector <rrt> region;
};
class program {
    time time_acquired;
    string title;
    optional content this_content;
    optional text this_text;
    oid;
};

```

例5.

IRD が各チャンネルの完全なスケジュールを必要とすることはしばしばあります。eXtremeDB のAPI を使用してこの要求を実現することはとても容易な作業です。プログラムのフローは以下の図1 のとおりです。

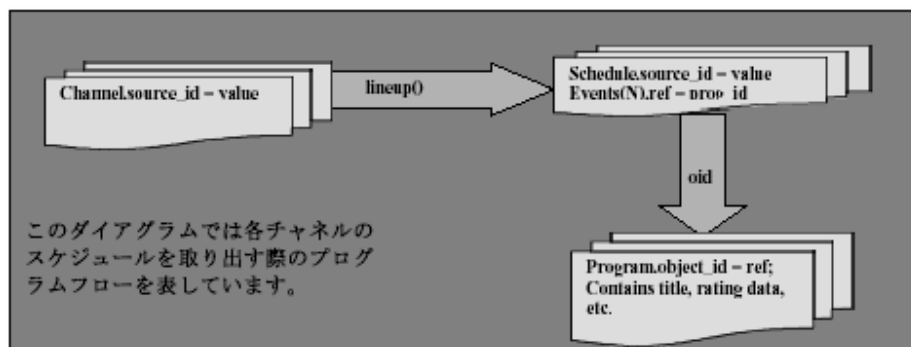


図1.

EPGはまず、lineupツリーベース・インデックスに基づいたスケジュールをナビゲートすることで各チャンネルに関連付けられたすべてのスケジュールオブジェクトを配置します。

このグループの各スケジュールに対して、EPGはイベントのベクタを読み込みます。各イベントはprogram_idをポイントします。すなわち、イベントのすべてのテキスト情報を持っているプログラムオブジェクトのオブジェクト識別子の役目を担います。この参照方法によって、EPGは関連するprogram_objectの位置を確定し、イベントに関する記述および他の関連情報をリードします。以下のコード片では、アプリケーションごとにeXtremeDBによって生成されるAPIを使用してこのクエリーを実現しています。eXtremeDBを利用したアプリケーションはeXtremeDBのスキーマコンパイラが生成したAPIを使い、eXtremeDBデータベースのオブジェクトを格納、リードおよび操作します。このことは静的な独自ナビゲーションAPIもしくはSQLなどの静的な標準APIを利用した他の多くのデータベースと対照的です。eXtremeDBのAPIは常にアプリケーションに合わせて生成され、あたかもデータベースがアプリケーションのニーズに合わせてつくられたように、データベースとアプリケーションの直感的なインテグレーションを実現します。

```
uint1 channels_schedule(mco_db_h db, uint2 source_id_from_channel)
{
    /*
     * t is a transaction handle
     * csr is a database cursor
     * sch is a handle to a schedule object
     * epgoid is the object id structure defined in the schema
     * event is a handle to an Event object. Event is a part of a
     schedule that has descriptive information attached
     * prog is a handle to a program object - the descriptive
     information for the event could be shared between
     multiple events
     */
    mco_trans_h    t = 0;
    mco_cursor_t   csr;
    MCO_RET        rc;
    Schedule       sch;
    MyEPG_oid      epgoid;
    Event          event;
    Program        prog;
    Char           title[MAX_TITLE];
    uint2          real_size;
    uint2          schedule_source_id;
    uint2          number_of_events;
    uint2          I;

    rc =
        mco_trans_start(db, MCO_READ_WRITE, MCO_TRANS_FOREGROUND, &t );
    if(rc)
        goto err;
    /* instantiate a cursor for the 'lineup' index */
```

```

rc = schedule_lineup_index_cursor(t, &csr);
if (rc)
    goto err;
/* search the 'lineup' index and position the cursor with the
   supplied argument 'source_id_from_channel' */
rc = schedule_lineup_search( t, &csr, MCO_GE,
                           source_id_from_channel, 0 );

/* set up a loop to iterate over the 'lineup' index */
for(; rc == MCO_S_OK; rc = mco_cursor_next(t, &csr))
{
    /* obtain the schedule and... */
    schedule_from_cursor (t, &csr, &sch);
    /* ... read the source_id field out of it */
    schedule_source_id_get( &sch, &schedule_source_id);

    /* have we enumerated all the schedules for the channel */
    if (schedule_source_id != source_id_from_channel)
        break;

    /* the current schedule has the same source as the channel,
       * so we read all events that make the schedule. */
    /* find out the size of the events vector */
    schedule_events_size (&sch, &number_of_events);

    /* scan through them */
    for (i = 0; i < number_of_events; i++) {
        /* obtain a handle to the event within the vector */
        schedule_events_at (&sch, i, &event );
        /* read the program's identifier */
        Event_program_id_get ( &event, &pgoid );
        /* locate the program that contains textual
           information about the event. Use extremely fast
           oid-based search */
        rc = program_oid_find (t, &pgoid, &prog );
        if (rc)
            /* no information available for this time period */
            break;
        /* read and print out the title. In reality this
           information is forwarded to the presentation
           engine for further processing and display */
        program_title_get
            (&prog, title, MAX_TITLE, &real_size);
        printf("%s\n", title);
    }
}

```

```
err:  }  
}
```

まとめ

EPG (エレクトロニック・プログラム・ガイド) はもはや印刷された番組ガイドの代替物にとどまるものではありません。インテリジェントに構築され、頻繁に更新されるコンテンツにより拡張されたEPGは詳細な番組情報、ユーザの好みに合わせた番組ソート機能、番組観覧に関する個人の嗜好の記録などを提供して行くものと思われます。EPG技術とデジタルビデオレコーダの組み合わせによるパーソナル・ビューイング・サービスはパーソナルテレビジョンの可能性を拡大していくでしょう。

EPG の開発および保守を任せられた会社にはPC と異なるテレビスクリーンおよびリモコンに関する特別な知識と経験が求められます。同様にデータ処理、それもデコーダ内にあるメモリ容量という制約下でのデータ処理についても重要な知識が求められます。EPG のためのデータベース管理モジュールの開発はそれだけでも大きな挑戦ですが、IRD 開発者はEPG オペレータ規則に従う必要があります。

McObject 社のeXtremeDB のようなインメモリデータベース(IMDSs)はEPG アプリケーションの下で動作するものとしては理想的なデータベース管理システムを提供します。メモリフットプリントが小さいのみならず、多くの組み込み用OS のマルチスレッド環境をサポートし、eXtremeDB ランタイムはこのデータベース技術がCPUおよびRAM 使用量を最小化していることを例証しています。またeXtremeDB はコンパクトなストレージレイアウトを提供し、新たな検索手法およびEPG に要求されるデータ型をサポートし、EPG コードの効率的な開発および保守の方法を導きます。