管理番号＃１６－０２

平成１７年度
研究開発成果報告書

多次元ナレッジマネジメントを可能とする高度ペタバイトXMLストレージの研究開発

委託先：　㈱メディアフュージョン

平成１８年４月

情報通信研究機構

平成１７年度　研究開発成果報告書
（一般型）

「多次元ナレッジマネジメントを可能とする高度ペタバイト XML ストレージの研究開発」

目　次

1　研究開発課題の背景

社会状況から見る背景

IT 社会の進展により、コンピュータシステムが扱うデータのサイズはテラバイト（＝１０の１２乗バイト）級に達しつつある現状である。今後も、データのサイズは必然的に大きくなり、数年以内にはテラバイト級を超えて、ペタバイト（１０の１５乗バイト）級が現実のものとして視野に入ってくる。ペタバイト級のデータの例としては、次のようなものが挙げられる。

・３次元医用画像（サブミリメートル単位）：１万人分
・高精細ビデオ（ハイビジョン画質，１本２時間）：６万人分
・ ヒトゲノムの塩基配列データ（３０億塩基対）：１００万人分
・ 無線装置（IC タグ，IC カード，携帯電話からの位置信号）：１００万台×１０日分

数年前までは、ペタバイト級のデータを扱うことなど全くの夢物語であった。しかし、メインメモリの低価格化（2004 年 4 月現在、1M バイトあたり 20 円）・ハードディスクの低価格化（2004 年 4 月現在，1G バイトあたり 60 円）の進展によって、ペタバイト級のハードディスクとテラバイト級のメインメモリが、数億円程度のコストで入手可能となってきた。メインメモリ及びハードディスクは、おおよそ１年半で半分に値下がり続けている状況（ムーアの法則）であり、この１０年以内には、必ずペタバイト級が現実のものになると見込まれる。一方、６４ビットアーキテクチャ（６４ビットのアドレス幅）・６４ビットＯＳ（Linux, Windows, Solaris など）の普及が急速に進んでいる現状である。６４ビット版の Windows は､現在ベータ版が､インターネットで配布されている状況である。従来の３２ビットのシステムでは、扱える実メモリのサイズ・ファイルサイズともに３２ビットの壁（４ギガバイト）があったのだが、６４ビットのシステムでは、遥かに大きな実メモリとファイルが扱え、ペタバイト級のデータを扱うのに必要なハードウエアとオペレーティングシステムは整いつつある。

ペタバイト級データシステムの必要性– 産業社会への貢献–

ペタバイト級のデータを扱えるシステムの登場によって、新たなビジネスチャンス・高度な行政サービス等が開拓できる。例えば、医療分野では、患者の電子カルテのデータベース作りがすでに各所で始まっているが、ベッド数数百にもなる大病院では、１日のカルテ作成数は１０００以上にも達するし、各カルテは、測定装置（脳波，心電図，レントゲン，３次元医療画像など）のデータを含み、数十メガバイトから数十ギガバイトに達する。法律によって、カルテの１０年間保存が義務付けられているから、管理すべきカルテ数は、数百万件に達し、データの総数はテラバイト級に達する。しかし、テラバイト級のデータを扱えない現状では、こうしたカルテのデータの多くの電子化を断念せざるを得ない。一方、プラントなどの複雑な設備が組み合わさった状況において（工場，発電所など）は、各装置に取り付けられた数百万個以上に達するセンサーから、絶えず収集されるログを蓄積し、「そのようなログが記録されているときに、どのような故障が起きるか」などの分析・解析を行い、安全管理に役立てることが、実際の生産現場で期待されている。現状では、すべてのログのデータサイズはペタバイト級に達し、すべてを蓄積することは無理である。現場担当者の勘によって必要なデータを抜き出すことを行わざるを得ない。

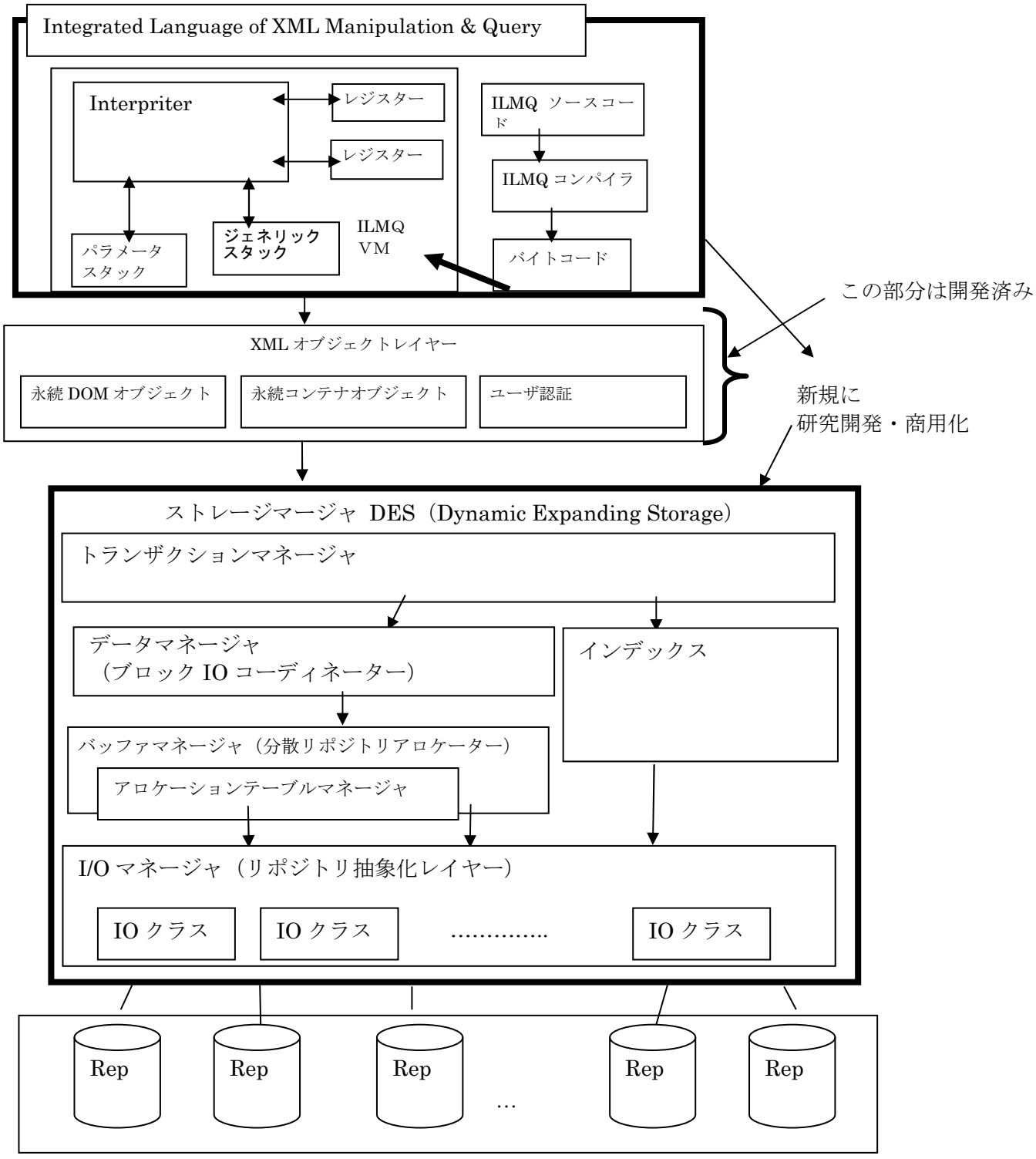
以上のように、ペタバイト級のデータを扱えるコンテンツ管理システムが実現できれば、製造業、流通業、小売業などの産業分野・行政分野・科学技術分野にお

いて、これら分野向けの新たなＳＩシステムの需要を喚起でき、さらに、多量に蓄積されたデータの山を使って新しいマーケティング・製造管理・研究開発その他新サービスの開拓が可能となり、産業の活性化につながる。以上が、ペタバイト級のデータベースシステム開発と実用化を着想した背景である。

２　研究開発の全体計画

２－１　研究開発課題の概要

研究開発概念図



本研究における、開発範囲は大きく分けて 2 つに分けられる。
既存の XML オブジェクトレイヤーに対して、一元的な操作を実現する「Integrated Language of XML Manipulation & Query」。また、XML オブジェクトレイヤーでのオブジェクト集合を永続的に保存するためのストレージマネージャである「Dynamic Expanding Storage」である。
「Integrated Language of XML Manipulation & Query」は構造的に 2 つに分けられる。言語体系、ならびに実行形式にコンパイルを行う言語コンパイラ（上記図では「ILMQ ソースコード」、「ILMQ コンパイラ」、「バイトコード」と表記）と、言語コンパイラによって生成された実行形式であるバイトコードを、実行するための実行環境（上記図では「ILMQ VM」と表記）である。
ストレージマネージャについても、構造的にいくつかのコンポーネントに分けられる。
ロックや、更新などデータベースにおけるトランザクション一般を扱う「トランザクションマネージャ」。XML オブジェクトを物理ファイル上へ永続化する際のブロックの割り当て等を管理する「ブロック I/O コーディネーター」。XML オブジェクトの効率的な問い合わせを行うための「インデックス」。物理ファイルに対する I/O を効率化する「バッファマネージャ」。ならびに、物理ファイルへの I/O を抽象化する「I/O マネージャ」である。
XML データベースを構成するコンポーネントはこれ以外にも多数必要とされ、実装されているが、本研究において主に設計、実装されるコンポーネントはこれらのものとなる。

２－２　研究開発目標

２－２－１　最終目標（平成１８年３月末）

「多次元ナレッジマネジメント実現のための新言語体系の創設及び
ペタバイト実現のためのストレージの動的な高度拡張技術の研究開発」

1） XML に対して一括した操作・問い合わせが行える新しい言語体系設計・実装し、そのために必要な各項目ごとの設計・実装・テストを行う。（これまでに実現できなかったことへの挑戦）
2） XML データベースの大規模化に伴い、ストレージの動的拡張（データベース容量が不足した場合、その容量を追加する機能）を実現する。複数の物理的なリポジトリ（データベースの実体を収めたファイル）を統合し、論理的な一つのリポジトリ(DES Dynamic Expanding Storage)とすることを可能とするシステムを研究開発することで、テラバイトを超えるストレージを扱うことを可能とする。（これまでの技術にはない、最新の技術）

ア）【サブテーマ１】XML に対する統合的問い合わせ操作言語の研究開発

ＸＭＬに対するクエリー、ノード操作、更新、トランザクション管理を行える言語（Integrated Language of XML Manipulation & Query）を設計し、その言語によってＸＭＬに対する統合的な操作を実現する。
そのための研究課題として次のような項目があげられる。

統合的問い合わせ操作言語（Integrated Language of XML Manipulation & Query）の設計・実装

・統合的問い合わせ操作言語の設計
・同言語を解釈、実行するための仮想マシンの設計・実装
・仮想マシンの入力となるバイトコードを生成するバイトコードコンパイラを実装

最終目標
　ＸＭＬデータベースに対する操作（クエリー、ＤＯＭノード操作・更新、トランザクション）を一括して実行するための言語体系を確立する。

イ）【サブテーマ２】多次元ナレッジマネジメントを実現するストレージの動的な拡張技術の研究開発

　XML データベースの大規模化に伴い、ストレージの動的拡張（データベース容量が不足した場合、その容量を追加する機能）を実現する。
複数の物理的なリポジトリ（データベースの実体を収めたファイル）を統合し、論理的な一つのリポジトリ(DES Dynamic Expanding Storage)とすることを可能とするシステムを研究開発することで、テラバイトを超えるストレージを扱うことを可能とする。その研究課題としては次のような項目が挙げられる

●動的拡張を実現するストレージマネージャー（Dynamic Expanding Storage）の設計・実装
（詳細項目）
・トランザクションマネージャー
・データマネージャー（ブロック IO コーディネータ）
・バッファーマネージャー（分散リポジトリアロケーター）
・アロケーションテーブルマネージャー
・IO マネージャー（リポジトリ抽象化レイヤー）

最終目標
　ストレージの動的拡張を実現、高速な処理を実現する。
最終目標は今まで出来なかった新しい技術（ストレージの動的拡張）が実現できるようにする、さらに商用に耐えれる商品とすること。

　なおサブテーマ２については、本提案において研究指導者となる九州大学大学院システム情報科学研究院牧乃内教授及び金子助教授の指導を受けながら、研究開発を実施する。

２－２－２　中間目標（平成１８年１月末）

ア）【サブテーマ１】XML に対する統合的問い合わせ操作言語の研究開発

●統合的問い合わせ操作言語（Integrated Language of XML Manipulation & Query）の設計・実装

・統合的問い合わせ操作言語の設計
・同言語を解釈、実行するための仮想マシンの設計・実装

中間目標
XML への統合的な問い合わせを実現するための基本設計を終了し、同言語を解釈・実行するための仮想マシンの設計・実装を完了する。

イ）【サブテーマ２】多次元ナレッジマネジメントを実現するストレージの動的な拡張技術の研究開発

●動的拡張を実現するストレージマネージャー（Dynamic Expanding Storage）の設計・実装
（詳細項目）
・トランザクションマネージャー
・データマネージャー（ブロック IO コーディネータ）
・バッファーマネージャー（分散リポジトリアロケーター）
・アロケーションテーブルマネージャー
・IO マネージャー（リポジトリ抽象化レイヤー）

中間目標
ストレージの動的拡張を実現するための基本設計を終了し、上記項目の実装をほぼ完了し、実用化に向けたβ版の配布に備える。

２－３　研究開発の年度別計画

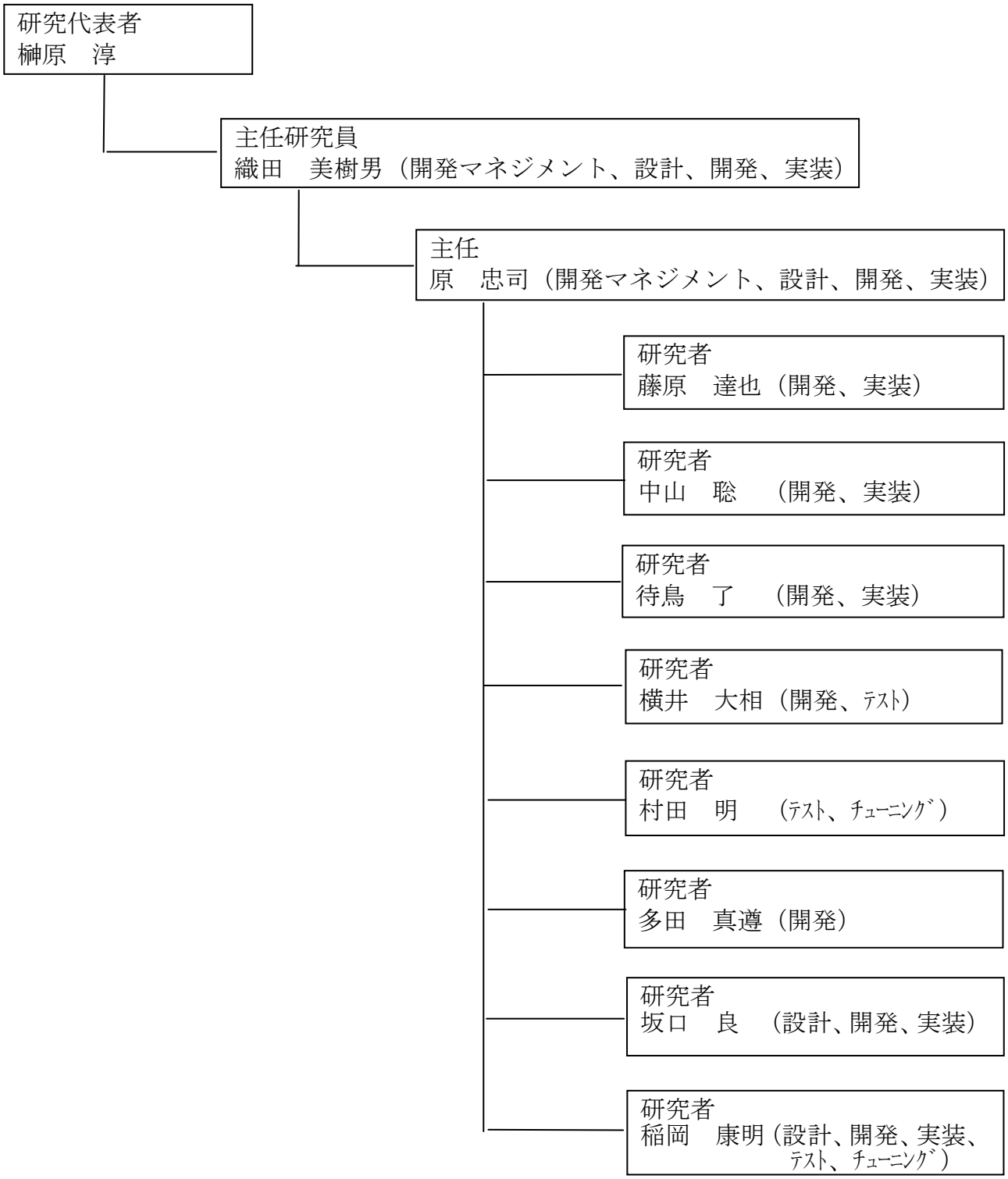
（金額は非公表）

研究開発項目	16 年度	17 年度	18 年度	19 年度	20 年度	計	備　考
ペタバイトを実現する新言語体系の創設とストレージ動的拡張の研究開発 ア）新言語体系の（XML に対する総合的問い合わせ操作言語）研究開発 イ）多次元ナレッジマネジメントを実現するストレージの動的な拡張技術の研究開発 テスト用サーバリース費用	→	→	→				
間接経費							
合　　計							

注）１　経費は研究開発項目毎に消費税を含めた額で計上。また、間接経費は直接経費の３０％を上限として計上（消費税を含む。）。
　　２　備考欄に再委託先機関名を記載
　　３　年度の欄は研究開発期間の当初年度から記載。

3　研究開発体制

3－1　研究開発実施体制



４　研究開発実施状況

４－１　XML に対する統合的問い合わせ操作言語の研究開発

４－１－１　序論、サブテーマの内容、研究開発全体から見た位置づけ

序論

現状、規格化されているＸＭＬの操作体系は、検索は XPath,XQuery、操作は DOM ノード、というように、複数の規格の集合となっている。
XML の操作を行う場合には、その複数の規格を理解し、矛盾なく調整を行いながら実装を行う必要があり、又、規格間の基本概念の相違から発生する実行効率の低下を受け入れながらも最小限に抑える必要がある。
そこで、その規格で規定された操作を満たしつつ統合された操作体系を実装するために、ＸＭＬの操作に適した操作言語を考案し、実装することとする。

サブテーマの内容

ＸＭＬに対する操作を考える上では、更新、参照ともに DOM ノードの操作が最小限の単位となる。
よって、現状の W3C における DOM ノードの操作規格である XML-DOM 規格を元に、更新、参照に必須の機能を抽出する。
また、検索に関しては、同様に規格化されている XPath、XQuery を元にその検索機能を抽出する。
抽出された機能に対して、それを言語要素として実現するような言語を考慮する。
言語としては、通常用いられる言語としての利便性、学習性から、手続き型言語とし、以下のような特徴を持つものとする。

変数　Xpath、Xquery で使用される Literal,Boolean,Numeric,NodeSet 型を扱えるようにする。

式、組み込み関数　ＸＭＬ－ＤＯＭにおいて使用されるノード間の移動、Xpath,Xquery において使用されるノードフィルタリングの機能を実現する。

制御構造　変数の状態による条件分岐、ループ等を行えるものとする。

研究開発全体から見た位置づけ

研究開発のメインテーマである多次元ナレッジマネジメントの中核であるＸＭＬストレージに対して、統一された操作言語を与える。
これによって、現状、各種言語に対して各々用意されていたプログラミングインターフェイスを簡略化する。
簡略化によるメリットとしては、使用者の学習容易性、可読性、メンテナンス性向上などの変更容易性によるシステム構築のコストダウン、操作粒度が縮小することからくるアプ

リケーションに対する適合性の向上などがあげられる。
また、統一されたインターフェイスにより、他の言語、システム等との柔軟な連携が容易となる。

４－１－２　実施状況

統合的問い合わせ操作言語は、その言語を実際の命令に翻訳するための言語コンパイラと、そのコンパイラによって生成されたバイトコードを実際に実行する仮想マシンという、２つのコンポーネントとして開発する。
これにより、言語と実行環境のインターフェイスが簡素化、抽象化され、お互いの依存性を抑えることが可能となる。
また、言語コンパイラによって翻訳されたバイトコードは再利用可能であり、実行時における言語コンパイラの翻訳時間による速度低下を避けることが出来る。
初期に実装し動作を確認した実行環境において、スタック操作の最適化が不十分であるとの認識から、汎用スタックの増設、命令の追加、およびそれに対応したコンパイラの変更を行った。

４－１－２－１　統合的問い合わせ操作言語の設計

４－１－２－１－１　言語仕様

言語仕様の抜粋を以下に示す。

・プログラム := 関数定義 | 関数定義 複合式 | 複合式

・関数定義 := 関数式 | 関数式 関数定義

・複合式 :=　式| 式 複合式

・式 := 代入式 | ループ式 | 条件式 | ブロック式
　　| 出力式 | Self 式 | 値定義 | リターン式;

・値定義 := 数値型 | 文字列型 | 論理型 | ノードセット型　;

・演算式 := 部分式 '+' 演算式 | 部分式 '-' 演算式 |　　'-' 部分式 | 部分式

・部分式 := 括弧式 '*' 部分式 | 括弧式 '/' 部分式 | 括弧式;

・括弧式 :=　'(' 演算式 ')' | 変数 | 数値式 | 関数呼び出し
　　| '(' インクリメント ')' | '(' デクリメント ')'

上記のような操作言語で記述されたテキスト列からバイトコード列を生成する。
バイトコード生成時に発生する冗長性をなるべく減少させるために、簡易な最適化を行っている。
バイトコードの１命令は、以下のような体系となっている。

・オペランド　　２バイト
・レジスタ参照もしくは直値（複数可）　ｎバイト

オペランドは、２バイトの値で、命令ごとにユニークな値が定められている。
レジスタ参照もしくは直値の部分には、レジスタの番号を現す値、もしくは実際の値（論理地、数値、文字列等）を記述する。

通常はこのような命令を複数並べた形をとる。命令のシーケンスの最後には、命令の終了を表すターミネータを置く。

・命令（複数可）	ｎバイト
・ターミネータ	２バイト

このような構造の命令を順次仮想マシンに送ることで、実際の動作を行うこととする。

４－１－２－１－２　追加仕様

上記の仕様で開発し、動作確認を行った統合的問い合わせ操作言語(以下操作言語と記述)について、設計上不十分であったと認識されるスタック操作の最適化に対応するため、仮想マシンに対して追加される、汎用スタックの増設、命令の追加に対応するための修正を行った。
これによって、仮想マシンに対して行われた性能向上に対して、操作言語レベルでの対応を行った。

４－１－２－２　操作言語を解釈、実行するための仮想マシンの設計・実装

言語コンパイラによって生成されたバイトコードを逐次実行する。
バイトコードを収めるための主記憶を用いることなく、状態遷移によって動作する。
仮想マシンは以下のような構造を持っている。

レジスタ

仮想マシンは複数のレジスタと呼ばれる値を格納、演算するための領域を持つ。
レジスタには以下の値を格納することができる。

・ 文字列
文字列。1 文字は 2 バイトで表される。Unicode 体系で表すことの可能な文字を収める。
他の型への変換が可能。

・ 数値
実数。倍制度の浮動小数点型。演算、および他の型への変換が可能。

・ 論理値
True、False の 2 値で表すことができる型。他の型への変換が可能。

・ ノードセット
XML-DOM ノードのノードセット。複数のノードに対する参照を扱うことが可能である。ノードセット同士の集合演算、および他の型への変換が可能。
主に、検索におけるノードセットのフィルタリング操作、検索結果の保存に使用される。

・ バイト列
任意のバイト列。各々の XML-DOM ノードは任意のバイト列と結びつけることが可能であり、その任意のバイト列はこのバイト型として扱うことができる。

・ バイトコード
バイトコード自体を収める。条件分岐、ループなど、主に制御構造のために使用する。

・ ステータス
仮想マシン自体のステータス。

スタック

値を先入れ先出し方式で保持する。以下の２種類のスタックがある。

パラメータスタック
パラメータの授受に使用する。

汎用スタック
一般的なアルゴリズムを実現するために使用する。
当初汎用スタックは１本のみであったが、４本の汎用スタックを使用できることとした。
また、各々のスタックの操作も、単純なＬＩＦＯだけではなく、ＦＩＦＯ操作も可能とした。

レジスタ、スタックに収められた値は、XML ストレージとのインターフェースともなっており、その変数に対する演算、操作がすなわち XML ストレージへのアクセスとなる。

これらの基本構造に対して、操作言語を実行可能とするための命令列を実装したが、これらの扱われ方から、頻繁に見られるバイトコードについての単一命令化、および汎用スタックの増設に伴うアルゴリズム実現方法の改善等を行い、性能向上を図った。

４－１－２－３　仮想マシンの入力となるバイトコードを生成する言語コンパイラの実装
すべてのスタック操作を一つの汎用スタックにて行っていたため、論理的には不要なスタック操作が発生し、生成されるコードが冗長となっていた。また、スタック操作は比較的低速であるため、実行速度の面からも問題があった。
そこで、スタック操作を複数のスタックに分散し、可能な限りスタック操作を最適化させるという方針の下にコード生成を行うよう改良を施した。また、不必要なスタック操作を行わず、レジスタの効果的な利用を行うよう改良を行った。
また、コンパイル後のバイトコードにおいて頻繁に見ることができるコード列に対して、同内容の処理を行う単一の命令を新規に設計、実装を行うことで、実行時におけるパフォーマンスの向上、ならびに、実行時に仮想マシンの負荷削減を実現している。

４－１－３　まとめ
言語インターフェースから XML ストレージに対する各種操作を一元的に行うという、当初の目的は達成されたと思われる。また、言語インターフェイスの速度についても、スタ

ック、レジスタの使用状況の改善、および命令の統合等によって全体的な向上を見せた。しかしながら、製品化を行うためには、コンパイラにおける最適化機能の向上を図る必要があり、また、さらなる速度向上のために、汎用レジスタを使用しない完全なスタックマシンの実装等を行う必要があると考える。

４－２　多次元ナレッジマネジメントを実現するストレージの動的な拡張技術の研究開発

４－２－１　序論、サブテーマの内容、研究開発全体から見た位置づけ

序論

多次元ナレッジマネジメントの中核であるＸＭＬストレージにおいては、主に速度的な問題から、そのストレージ容量を事前に設定し、後日の変更は不可能であった。
本研究開発は、速度要求に対しては現状にとどめ、その制限のみを緩和する手法を探ることとした。

サブテーマの内容

　ＸＭＬストレージは、その扱う情報を納めるファイルと密接な関係にある。
　ファイル上の位置を“参照”と呼び、ＸＭＬストレージ上の原子的な情報は全てその参照によって連結することで構造をあらわしている。
　ストレージ容量の変更にはそのファイルの物理的構造の変更が不可欠となるが、構造記述には多分に物理的な側面のある“ファイル上の位置”そのものを使用しているため、原子的な情報のファイル上での位置を変更等は事実上、不可能に近い。
　これを可能とするためには、ストレージ上の現状の参照、構造等を全く変化させることなく容量追加に伴う構造の変化のみを行う必要がある。
　本研究では、そのような変更を可能とするための抜本的なファイル構造の変更について考慮する。

研究開発全体から見た位置づけ

　ＸＭＬストレージを運用開始後に拡張可能とすることで、システム全体の拡張性を高めることを主眼とする。このために行うＸＭＬストレージの基本構造の変更に伴うシステム全体の変更は最小限とする。

４－２－２　実施状況

XML ストレージを構成するファイルは、構造を異にする複数の領域からなる。
各々の領域は、ビットマップ、ブロック、インデックス、ハッシュテーブルと呼ばれ、それぞれ独自の構造を持つ。
各々の構造は、1 つの物理的ファイル上にその順で配置されている。
上記のうち、領域拡張に対しして構造を変化させる必要のある部分は、ビットマップ、ブ

ロック、インデックスの領域である。各々の領域について、その構造から考慮することのできる領域拡張の手法を実装する。
また、リポジトリのより効率的な利用を目指し、リポジトリ構造自体の変更もすることとした。変更の要点は、以下の２点である。

・ ブロック領域をそのブロックサイズごとに２種類とする。
・ 同一の内容のデータを一元化する。

以下では、これらの変更についての内容を記述する。

４－２－２－１ ビットマップ領域
ビットマップ領域は、後述するブロックの使用状況を管理するための構造である。
ビットマップ領域は、複数のビットマップブロックの集合を 3 重の木構造とすることで構成されている。
このような木構造の利用により、ひとつのビットマップブロックが保持するビット数の 3 乗までのブロックを管理可能となる。
この構造の領域拡張における問題は、ビットマップブロック間の構造連携を演算から求めていることに起因する。
詳細には、領域が拡張された場合、ビットマップも同様に拡張する必要があるが、ビットマップはその木構造の階層ごとに連続しているという前提から演算を行うため、木構造の各々の階層ごとのビットマップブロック数を変更できない、ということになる。
よって、領域拡張後の最大サイズを事前に指定し、それに準じた構造としておき、後の拡張に備えるという手法を用いた。

４－２－２－２ ブロック領域
ブロック領域は、原始的な情報を単位として扱う領域で、実際の情報は全てここに収められる。ブロック領域の大きさの一定なブロックを複数収めるが、常にその先頭から使用されていく。よって、その使用されている最終のブロック以降については、構造的には自由に拡張が可能であるが、実際にはブロック領域以降の構造であるインデックス領域、ハッシュテーブル領域の再配置を行う必要がある。
上述した理由により、再配置を行わず拡張を行うため、インデックス領域、ハッシュ領域については、別ファイルを用い、ブロックの終端＝ファイルの終端とすることで、構造変化を伴わない自由な拡張を可能とした。

４－２－２－３ インデックス領域
インデックス領域は参照を基本単位とし、その参照を複数収める構造となっている。
インデックス領域の大きさは、ブロック領域の大きさと比例していることが求められる。
インデックス領域は比較的単純な構造であるため、上述のブロック領域の拡張と同様の手法、すなわち物理ファイルの終端＝論理構造の終端とすることで、拡張可能となる。

４－２－２－４ ブロック領域をそのブロックサイズごとに２種類とする。
ブロック領域は、元来ＤＯＭノード構造をあらわす情報を収めることのできる最小のサイズに設定されている。（具体的には１００バイトに満たないサイズ）
このため、論理的には大きなブロックサイズを必要とする構造、たとえばＢ－ＴＲＥＥ等をシステム上に構築した場合、性能を出すことが困難である。
この問題に対して、比較的大きなブロック（たとえば１ブロック 4Kbyte）を使用することで克服が可能であり、その実現のため、複数のブロックサイズを扱えるよう拡張を行った。

４－２－２－５同一の内容のデータを一元化する。
リポジトリ領域中に格納されるデータについて、比較的に静的なデータ(更新等が起こりにくいデータ)についてはこれを一元化し、リポジトリの使用効率を高める工夫を施した。

上記手法を全て実装することによって、現状、XML ストレージに対して事前に定められた大きさまでの任意の単位での拡張が可能となった。
上記の XML ストレージに対する修正に対応するため、データベースシステム中の以下のコンポーネントに対して変更を行った。

４－２－２－６　トランザクションマネージャ
従来単一のデータ構造に対しての出力のみを考慮して設計されていたトランザクションマネージャについて、複数の論理的に分割された構造に並列にアクセス可能となるように、トランザクションマネージャの修正を行った。これによって、今回の変更要点である、ブロック領域をそのブロックサイズごとに分けるという基本方針をトランザクション側でサポートすることが可能となった。

４－２－２－７　データマネージャ(ブロック I/O コーディネータ)
これについても、トランザクションマネージャと同様に、単一のデータ構造に対しての出力のみを考慮して設計されていた物に対して、複数種類のデータ構造に並列にアクセス可能となるように修正を行った。

４－２－２－８　バッファマネージャ(分散リポジトリアロケータ)
複数種類のデータ構造(ファイル)に対して、一元的にアクセスできる環境を実現するため、バッファマネージャについても修正を施した。
これにより、異なるブロック領域を持つ複数のリポジトリファイルを単一のシステムで操作することが可能となり、トランザクションマネージャ、およびデータマネージャで必要とされる要件を満たすことができた。

４－２－２－９　アロケーションテーブルマネージャ
上位レイヤーのコンポーネントから隠蔽された形で、空きブロックの管理を行うコンポーネントである。
ファイル上の空きブロックを管理する、アロケーションマネージャについて、複数のファイル上の空きブロックを一元的に管理できるように修正を施した。
これにより、複数のファイル上の空きブロックを一元的に管理することが可能となり、データマネージャ、および I/O マネージャで必要な要件を満たすことができた。

４－２－２－１０　I/O マネージャ(リポジトリ抽象化レイヤー)
バッファマネージャが複数種類のデータ構造に対してアクセスできるように修正されたことに伴い、より上位のレイヤーであるトランザクションマネージャや、データマネージャなどに均一のインターフェースを提供する I/O マネージャについても、これをサポートする機能追加を行った。これによって、バッファマネージャで物理的ファイルにバインドされたデータ領域を、統一されたインターフェースでより上位のレイヤーで使用できるようになった。

４－２－３　まとめ

元来、拡張に対する配慮を行っていない構造に対して、その構造の変化を行わずに拡張のみを行うということは非常に困難だと考えられるが、今回はそれに起因する性能の低下などを起こさない範囲でその実装が完了した。また、ブロックサイズを複数管理することは困難であると思われたが、リポジトリの論理的構造を再定義し、それに沿ったリファクタリングを行った等の効果もあって、比較的スムーズに行うことができた。今回の開発によってリポジトリの論理的な構造拡張に対しての一定の効果が得られたと考えられるが、製品化に当たっては、ビットマップの動的拡張が課題として残っている。また、今回の研究の成果からより効率的なリポジトリの論理構造を構築できる可能性が発見され、今後の開発のポイントとなっている。

４－３　総括

３月末現在で個々の課題の解決はほぼ完了しているが、製品化のために各機能のチューニング・信頼性の向上・各種のテスト及びチューニングを相当行う必要がある。
また、日本の某情報家電メーカーから更なる性能向上の要求も出ており、それも含めて今後製品化に向けて一部作り直し等を含めて開発を数年続ける予定である。

５　参考資料・参考文献

５－１　研究発表・講演等一覧

本研究開発に関する研究発表・講演等はございません。