3.1 減災情報共有データベースの開発

（防災科学技術研究所，産業技術総合研究所）

3.1.1 目的と概要

システムを構築する場合には，データベースのモデル選択が重要である．そこで，防災で扱う地理情報に着目し，位相暗示型データ構造（KIWI+）と位相明示型データ構造（RDBMS）の特徴的なデータ構造のデータベースを開発し，その特徴から最終年度で扱うデータベース構造を決定する事を目的とする．減災情報共有データベースは，取り扱う情報の時間的広がりを考慮して，時空間データベースとした．

3.1.2 位相暗示型データベースの実装

データベース管理システム（DBMS：Data Base Management System）のバックエンドとして位相暗示型データ構造(KIWI+)を使用する手法である.この手法は後で述べるデータベースと違い，位相を暗示的に保持することを特徴とする．

(1) 基本的な処理の流れ

基本的な処理の流れは以下のようになる．

- 接続の待ち受け・確立

  TCP ポートをバインドして待ち受け,クライアントからの TCP コネクションを確立させる．複数の接続が来た場合は，それぞれ個別の通信として扱う．本処理はデータベースの機能と切り離し別ツールとして提供される．

- MISP による命令の読み込み

  MISP による命令を読み込み・構文解析し，XML 形式のデータから KIWI+形式のデータへ変換するために事象テーブルを介してデータを読み込む．

- 処理

  DBMS に対し，実際のデータ更新・検索処理を行う．

- 結果の送信

  データ更新・検索の結果を KIWI+形式のデータから XML 形式のデータに変換し送信する．

(2) データ格納

位相暗示型データ構造(KIWI+)をデータベースに用いる場合には,次の事を考慮する必要がある． MISP で送られてきた XM を，XML⇔関係モデル，関係モデル⇔KIWI+の２段階に分けて考える．中間に関係モデル（関係／テーブル）を置くことで，データベース管理システム利用者の利便性を高めることができる．XML⇔関係モデルの変換は，XML の木構造を一筆書きでたどり（トリーウォーク）線形にして（フラッティング），関係のタプ

ル（レコード．レコードタイプについても同様.）を得て行う．関係のテーブル名および項目名に XPath（タグ名称をフルパスで表現したもの）を用いることにより，XML と関係との一意的な対応を保証する．XML の繰り返し構造については，その部分を別の関係とし，繰り返し分のデータをタプルで表現する．

a）関係（テーブル）の構造

関係 R の名称 RN，R に所属する項目 F の名称 FN および F の型 FT，項目の並び順を<F*>としたとき，関係の構造は，図 3.1-1 のように表すことができる．

R = <RN, F*>, F = <FN, FT>
または，
R = <RN, (<FN, FT>)*>

<a, b>：順序付けられた集合
*：繰り返し

図 3.1-1　関係の構造

MySQL のようにジオメトリ（図形または形状）を表現できる RDBMS では，FT として POINT，LINE，POLYGON，等のジオメトリクラスが指定できるようになっている．

KIWI+の構造を模式的に表すと，そのデータの基本要素は，Connector（点の物理レコード．名称 CN，型名 ConnectorID，属性 CA（時空間情報を含む）.），Vector（線の物理レコード．名称 VN，型名 VectorID，属性 VA（時空間情報を含む）.）であり，時空間上の位置を占める．Connector，Vector のどちらにも属性値を記述できるが，関係モデルと異なりジオメトリの存在を前提としている．Vector は，一部か全部の地物輪郭形状，または，一部か全部の地物間境界線を表す．データは，空間を矩形で切り取ったパーセルに対応したファイルに格納されるため，パーセルの境界を横切る Vector は，パーセル境界との交点で切断されている．すなわち，物理レコードを得るための２次元的なブロッキング処理である．論理的な線の形状は，同一 VectorID で検索することにより得ることができる．なお，論理的な形状を厳密に議論したい場合は，パーセル境界上の点がデータ管理上のブロッキング処理により便宜的に生成されたものなのか，交差点のようなノードや道路形状を補間する点（形状補間点）のような正規の形状データなのかを区別する必要があり，KIWI+の規定に加えることにしている．

KIWI+はデータベース構造の物理スキーマを中心に記述しているが，KIWI+の実体定義を用いて論理的なジオメトリ（点・線・面・体）を表現することもできる．実体定義の模式的な構造は，実体 e の名称 eN，型 eT（点'P'・線'V'・面'F'・体'B'），その形状の構成候補となる Vector 種別（VectorID）のリスト，実体に所属する Connector 種別（ConnectorID.

実体の代表点.）の集合からなる. Vector は, 複数実体の形状要素となり得るが, Connector は，たかだか１つの実体に所属する．したがって，Vector の属性値は，Connector の属性値と異なり,実体の属性値を一意的に構成することは保証されていない.実体定義により，KIWI+で記述されたデータベースの直接的な管理プログラムは，自動的に実体形状を生成することが可能となる（**図 3.1-2**，**図 3.1-3**）.

Vector = <VN, VectorID, VA>
Connector = <CN, ConnectorID, CA, eN>
e = <eN, eT, VectorID*>

図 3.1-2　Vector 定義，Connector 定義，実体定義

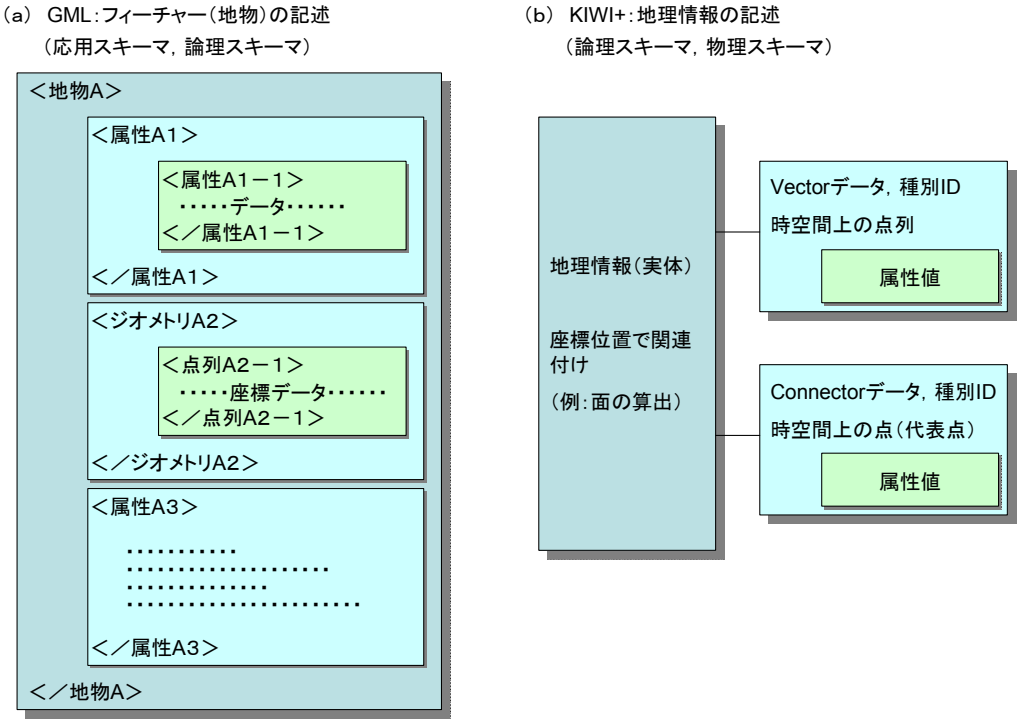

図 3.1-3　Vector 定義，Connector 定義，実体定義

b）変換アルゴリズム

事象テーブルの作成：

変換処理は，事象テーブルに従って実行されるため，前処理として事象テーブルを作成する必要がある．この場合，以下の３つの利用ケースが考えられる．

- 事前にデータベース管理者が作成できる場合
  同一機関内，協定機関間でデータ変換を行う場合は，データベース管理者が，対象の XML のスキーマおよび KIWI+のスキーマを知っているか知り得ることができため，

事前に事象テーブルを作成し，変換アルゴリズムに与えることができる．

- 事前にデータベース管理者が作成できない場合

  前述 1）以外の場合，データベース管理者は，XML のスキーマか KIWI+のスキーマの一方を知っているか，両方とも知らないと考えられるため，事前に事象テーブルを作成することができない．この場合，変換アルゴリズムを実装したプログラムは，入力データを分析して，上記の一方を作成し，未知のもう一方を仮に作成する機能（事象テーブル作成支援機能）を持つ必要である．自機関または相手機関のデータベース管理者は，その結果を確認し，事象テーブルを修正・完成して，再変換を実行する．仮作成においては，形状（ジオメトリ）とそれに所属する属性（ジオメトリ以外の項目）を判別する必要がある．また，仮の名称をユニークに生成する必要がある．以下に，変換の方向に応じた形状とそれに所属する属性の判別方法を示す．

  ・ 関係→KIWI+変換の場合

  関係のジオメトリ項目は，KIWI+の実体に対応し，次に出現するジオメトリ項目の手前か終了までの属性を，当該実体の属性とする．先行したジオメトリ以外の項目があった場合は，これらも最初に出現したジオメトリ項目に対応する実体の属性とする．

  ・ 関係←KIWI+変換の場合

  KIWI+の実体は，関係のジオメトリ項目に対応し，実体に所属する属性を項目として関係を生成する．

3）事象テーブルに未定義の事象を追加する場合

未定義の事象について，前述 2）と同様の処理を行う必要がある．

以下，完成した事象テーブルを与えた場合の変換アルゴリズムを示す．記述は，構造化日本語による．

関係→KIWI+変換アルゴリズム：

各関係について以下を繰り返す．

⇒ 各項について以下を繰り返す．

⇒ 項目の XPath をキーに事象テーブルと照合する．

⇒ 項目に eN が対応する場合，実体 e の形状を出力する．

注：面の場合，実体定義の最初の VectorID で Vector を出力する．属性がない場合は，最初に見つけた所属 ConnectorID で Connector を面内位置に出力する．

⇒ 項目に eN.ConnectorID が対応する場合，その ID の Connector を属

性に加えて生成する．項目が繰り返し構造の場合は，属性項目を複数持つ属性とする．Connector のデータレコードサイズがオーバーした場合は，同一 ID（種別）の Connector を同一位置に生成して溢れたデータを格納する．

注１：ジオメトリの型に応じて，生成する位置座標が異なる．

・点：その位置

・線：始点の位置

・面：面内の位置

注２：虚空間に関する処理は省略しているので注意．

関係←KIWI+変換アルゴリズム：

事象テーブルの各事象について以下を繰り返す．

⇒ 事象名（XPath）を関係名として検索する．

⇒ 事象の各項目について以下を繰り返す．

⇒ 項目名（XPath）を関係項目名として検索する．

⇒ 項目が eN の場合，実体定義から形状を算出し座標(列)を関係のジオメトリ項目値として出力する．

⇒ 項目が eN.ConnectorID の場合，実体 e に所属する Connector の属性値を項目値として出力する．Connector 属性値の項目数が２以上の場合は，繰り返し構造を持つ属性値として出力する．この場合，溢れた属性値を格納した Connector（同一 ID，同一座標）の属性値も出力する．

位相暗示型データ構造は，以下のような利点がある．

- データサイズが小さい

  位相暗示型データ構造は，データベースにデータ実体として格納する基本データ構造を点と線とし（属性は各々に付属），面と実体（事象）はその型の定義から計算で自動生成するしくみになっている．このため，データの時間変化を記録してもデータベースはコンパクトになり，データ交換においても点・線のみを遣り取りすればよいため効率的である．

- 位相の暗示的な記述

  データの 3 次元空間および時間の中での存在位置の関係によって関連の有無を表し，関連を知りたいときに計算で求める方法である．データ更新の場合は，更新対象データを書き換えるだけで済み，更新の影響が伝播しない．履歴管理においても，更新対象データの書き換え後の結果が増えるだけですむ．この特徴は，地理情報のように膨大なデータ量を対象にする場合，非常に重要である．

- DBMS の利用制限

  位相暗示型データ構造には他に，KIWI+形式の最小セットであるDiMSIS形式，および，車載を考慮した組込型で参照関係を展開しアクセス効率を更に向上したKIWI形式(JIS D0810)がある.KIWI+形式は財団法人日本道路地図協会における道路ネットワークデータ管理で利用され，DiMSIS形式は３０ほどの自治体に業務利用され，KIWI形式はカーナビケーション用ディスクの国内標準として広く利用されている．しかし，RDBMSにようにフリーなものが存在するわけではなく，KIWI+事務局（URL：http://www.drm.jp/KiwiPLUS/）を通した利用制限がある．

このような利点がある反面，次のような課題も挙げられる．

- 面と実体の生成処理

  位相暗示型データ構造は，データベースにデータ実体として格納する基本データ構造を点と線とし（属性は各々に付属），面と実体（事象）はその型の定義から計算で自動生成するしくみになっている．このため，面と実体の生成には処理系（ソフトウェア）の負担が大きくなる．

- トランザクション機能

  トランザクションとは，複数の処理をひとまとめの処理としてまとめたもので，処理の途中で何らかのエラーが発生した場合でもデータの整合性を保つためのものである．特に災害時はネットワークの突発的な切断や電源断等の可能性も考慮に入れておかなければならず，データの整合性が保たれていないと無用な混乱がもたらされる．トランザクション処理機能は，データベースの管理処理として，1 から作成するには開発に相応のコストが発生する．RDBMS にはすでに基本的に備わっている機能であるが，本プロジェクトで開発した DBMS には本機能は備わっていない．

- 検索プログラムの負荷

  データの関連をデータベース上に持たないため,検索プログラムの負担が大きくなり，空間的な演算のアルゴリズムを工夫する必要がある．

3.1.3 位相明示型データベースの実装

データベース管理システムのバックエンドとして既存の RDBMS (Relational Database Management System) を使用する手法である．この手法は位相暗示型データ構造と違い，位相を明示的に保持することを特徴とする．

(1) 基本的な処理の流れ

基本的な処理の流れは以下のようになる．

- 接続の待ち受け・確立

TCP ポートをバインドして待ち受け，クライアントからの TCP コネクションを確立させる．複数の接続が来た場合は，それぞれ個別の通信として扱う．

- MISP による命令の読み込み

  MISP による命令を読み込み，構文解析する．必要に応じて入力データを読み込むための情報を RDBMS 内の管理テーブルから取得する．

- 処理

  RDBMS に対し，実際のデータ更新・検索処理を行う．

- 結果の送信

  データ更新・検索の結果を XML 文書の形式で送信する．

(2) データ格納

MISPによる要求の読み込みからRDBMSに対する処理に関して例を挙げて説明する.なお，扱うスキーマによってはさまざまな処理手順が必要であるが，説明の都合上まずは単純な例で説明する．基本的には，登録されたスキーマを RDBMS 上の表の形に変換した後，データを表に格納することによって外部からの要求(データの入力・削除・更新・取得)に答えるという方法を取る．

まず，クライアントが以下のような RegisterFeatureType 要求を発行し，スキーマを登録したとする．ここでは，名前空間 http://www.infosharp.org/test/dictionary に属する DictionaryEntry 要素を定義しており，DictionaryEntry 要素はその子要素として文字列型の key と value という要素を持つ．

```
<?xml version="1.0"?>

<misp:RegisterFeatureType xmlns:misp="http://www.infosharp.org/misp">

   <xsd:schema
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.infosharp.org/test/dictionary"
        misp:id="http://www.infosharp.org/test/dictionary.xsd"
        xmlns:dic="http://www.infosharp.org/test/dictionary">

     <xsd:element   name="DictionaryEntry"
                    type="dic:DictionaryEntryType"/>

        <xsd:complexType name="DictionaryEntryType">
            <xsd:sequence>
              <xsd:element name="key" type="xsd:string"/>
              <xsd:element name="value" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>

   </xsd:schema>

</misp:RegisterFeatureType>
```

このスキーマによって妥当な XML 文書と認められる文書は以下の例のようなものである.

```
<dic:DictionaryEntry xmlns:dic="http://www.infosharp.org/test/dictionary">
        <dic:key>foo</dic:key>
        <dic:value>bar</dic:value>
</dic:DictionaryEntry>
```

本手法ではこれを保存するため，以下のような表を RDBMS 上に構築する．表のそれぞれのカラム名は XPath のサブセットであり，要素(ここでは DictionaryEntry)を基準にした相対の XPath を使用する(実際には XPath には多様な文字が入り，そのままでは正しいカラム名ではないため XPath を一意なキーに変換したものを使う)．カラムの型は，スキーマで指定された型(xsd:string 等)をもとに，それに相当する SQL の型に変換したものを使用する.

{http://www.infosharp.org/test/dictionary}DictionaryEntry

key	value

クライアントが以下のデータ登録要求によって，前述の例のデータを登録すると，表は次のようになる．

```
<?xml version="1.0"?>

<misp:Transaction xmlns:misp="http://www.infosharp.org/misp">

  <misp:Insert>
    <dic:DictionaryEntry
         xmlns:dic="http://www.infosharp.org/test/dictionary">
            <dic:key>foo</dic:key>
            <dic:value>bar</dic:value>
    </dic:DictionaryEntry>
  </misp:Insert>

</misp:Transaction>
```

{http://www.infosharp.org/test/dictionary}DictionaryEntry

Key	value
Foo	bar

この例では文字列しか扱っていないが，時刻や地理形状も同様に扱う．例えば，文字列型の key・value の他，ポリゴン型の外形と時刻を加えたスキーマを登録された場合は，それぞれそれに対応したカラムを追加した表を作成する．RDBMS 側で地理データ型を扱える機能がある場合はそれを利用し，複雑な階層的データの場合でも展開してフラットに並べる（ここでは詳細に示さないが，XMLSchema では繰り返し構造(ある要素が 1 つ以上連続して表われる等)を記述することが可能であるが,これはフラットには展開できないため別表などで管理する必要がある)．以下に地理形状・時刻・階層的データを含んだ，もう少し複雑なスキーマと変換された表を示す．

```
<?xml version="1.0"?>

<misp:RegisterFeatureType xmlns:misp="http://www.infosharp.org/misp">

        <xsd:schema
           xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns:XML="http://www.opengis.net/XML"
           targetNamespace="http://www.infosharp.org/test/building"
           misp:id="http://www.infosharp.org/test/building.xsd"
           xmlns:bld="http://www.infosharp.org/test/building">

          <xsd:element name="Building" type="bld:BuildingType"/>

          <xsd:complexType name="BuildingType">
            <xsd:sequence>
              <xsd:element name="outline"
                           type="XML:GeometryPropertyType"/>
              <xsd:element name="constructedTime" type="xsd:dateTime"/>
              <xsd:element name="area" type="bld:AreaType"/>
            </xsd:sequence>
          </xsd:complexType>

          <xsd:complexType name="AreaType">
            <xsd:sequence>
              <xsd:element name="buildingArea" type="xsd:double"/>
              <xsd:element name="grossFloorArea" type="xsd:double"/>
            </xsd:sequence>
          </xsd:complexType>

        </xsd:schema>

</misp:RegisterFeatureType>
```

outline	constructedTime	area/buildingArea	area/grossFloorArea
Polygon((10,20),(30,40) (0,0))	2000/01/01T00:00:00Z	100.0	500.0

このように手法 B では，地理形状，時間的形状と地物の属性を同等に扱う．そのため，形状の変化や属性の変化を時系列で管理することが可能であり，観測時刻・推定開始時刻・復旧予定時刻のような複数の時刻や，河川の平常時の形状と現在および過去の浸水域の形状等，複数の形状を一つの地物に付随する情報としてまとめて扱うことができる．また，属性ごとに時刻属性を付けることで，属性単位で階層的に時刻を付けて管理することも可能である．これらをフラットな表に変換して扱うことにより，クライアントが指定する検索条件が複雑であっても，これをダイレクトに RDBMS に伝える事ができ，検索の効率化が

期待できる．この他にもバックエンドに既存の RDBMS を利用することにより，以下のような利点がある．

- トランザクション機能
  トランザクションとは，複数の処理をひとまとめの処理としてまとめたもので，処理の途中で何らかのエラーが発生した場合でもデータの整合性を保つためのものである．特に災害時はネットワークの突発的な切断や電源断等の可能性も考慮に入れておかなければならず，データの整合性が保たれていないと無用な混乱がもたらされる．トランザクション処理機能は，データベースの管理処理として，1 から作成するのは開発にコストがかかるが，RDBMS にはすでに基本的に備わっている機能であり，バックエンドとして RDBMS を選択した場合，それをそのまま利用することができる．
- 多数のデータの処理の効率的な検索
  一般に RDBMS は，多数のデータ取り扱いに関して効率的な検索アルゴリズムを用いている．アルゴリズムとしては，B-Tree や R-Tree やそれらを改良したものを用いることが多い．R-Tree は，データに偏りがあった場合(例えば，情報が地理上の一地域に集中して存在している場合等)でも動的に探索木を再構成することで高速に検索が可能である．総データ数を N，検索結果データ数を M としたとき，O(logN) + M のオーダーで検索することが可能であり，多数のデータが登録された時でも高速に検索できる．
- 成熟した技術基盤
  RDBMS は，広く使われている技術であり，世界中の業務で使用されている実績がある．これまでの実績の中で，すでに大きなトラブル等が出尽くしており，それらはすでに解決済みである．新たにデータベースの管理システムを作成することに比べてバックエンドとして信頼することができる．
- 複数キーでの検索
  災害時には，地物形状での検索に留まらず地物に附属する情報での検索が必要である．例えば，「被害報告があり，まだ確認をしていない建物だけを検索する」などである．複数キーでの検索は RDBMS の基本機能として備わっており，高速な検索が期待できる．
- アップデート
  高速化・機能強化等 RDBMS 自身の改良も精力的に行われている．バックエンドに RDBMS を採用した場合，このような改良に対して少ないコストでその恩恵を受けることが出来る．RDBMS は通常，SQL という標準的なインターフェースを実装しているため，新しく良い RDBMS が使用可能になった場合でも，同様にその恩恵を受けることができる．
- フリーな RDBMS の存在
  フリーな RDBMS が存在する．低コストで導入でき，ライセンスに縛られない自由な改変が可能である．このようなフリーな RDBMS を利用すれば，開発のみでなく，導入す

る際にもライセンスを購入する費用は必要なく，手続きを取る必要はない．

このような利点がある反面，次のような課題も挙げられる．

- SQL の方言
  各 RDBMS は SQL という標準的なインターフェースを実装しているが，それぞれ独自の拡張や，微妙な差位が存在する．利点として RDBMS の開発をこちらでやらなくとも改良されたものが使えると述べたが，その恩恵を受けるためには一つの RDBMS に依存せず，標準的なものをできる限り使うように気をつけて実装する必要がある．
- バックエンドに致命的な問題があった場合の対処
  バックエンドを独自開発せずに別システムを採用した場合，バックエンド自身に致命的な問題が発生した場合に問題になることがある．上の「成熟した技術基盤」という項で述べたように，RDBMS の場合致命的な問題はあまり存在しないと思われるが，もし致命的な問題が見付かった場合，それを避けるためには，そのバックエンドに精通しておくか，もしくは前述の SQL の方言を吸収できるようにしておかなければならない．
- インストール
  バックエンドに別システムを採用した場合，インストールを個別に行なうことになると一体型に比べてインストールの手間がかかる．ただし，フリーなものの中には組み込み型として使えるものも多く，これらを採用することは可能である．

3.1.4 まとめ

システムを構築の重要な要素であるデータベースのモデル選択を行うため，2 種類の特徴的なデータベース構造を検討した．それらは位相暗示型データ構造（KIWI+）と位相明示型データ構造（RDBMS）である．前者はオープンソース化が難しく，位相暗示型データ構造という特殊なデータ構造であるため，取り扱える技術者が少ないという制限から発展性が難しいことを考慮して減災情報共有データベースとしては採用しないことにした．したがって，減災情報共有データベースには位相明示型データ構造（RDBMS）を用いることとする．