A Geometrical Data Application of the Cellular Data System

Toshio Kodama  
School of Engineering  
University of Tokyo and Maeda Corporation  
Tokyo, Japan  
kodama@ken-mgt.u-tokyo.ac.jp  
kodama.ts@jcity.maeda.co.jp

Tosiyasu L. Kunii  
Morpho, Inc.  
Tokyo, Japan  
kunii@ieee.org

Yoichi Seki  
Software Consultant  
Tokyo, Japan  
gamatak51@hotmail.com

Abstract— In this “cloud” computing era, various kinds of values are being created in cyberworlds at every moment, causing the worlds to become increasing complex and making it difficult for us to recognize the whole. Consequently, we need a more powerful mathematical background that can accurately model the cyberworlds in the “cloud” as they are. We consider the Incrementally Modular Abstraction Hierarchy to be appropriate for modeling the dynamically changing cyberworlds by descending from the most abstract homotopy level to the most specific view level, while preserving invariants. We have developed a data processing system called the Cellular Data System based on IMAH. In this paper, we introduce basic geometrical design into CDS, and apply CDS to develop the core logic of a route search system, used in most traffic industries but costly due to its complexity, to verify its effectiveness.

Keywords-component; incrementally modular abstraction hierarchy, formula expression, edge open map, tension map

I. INTRODUCTION

Cyberworlds are information worlds formed in a cloud either intentionally or spontaneously, with or without design. As information worlds, they are either virtual or real, and can be both. In terms of information modeling, the theoretical ground for the cyberworlds is far above the level of integrating spatial database models and temporal database models. They are more complicated and fluid than any other previous worlds in human history, and are constantly evolving. The number of companies that conduct business in cyberspace, such as Google and Face book is increasing and the market is growing remarkably. On the other hand, in general business application system, the scale of systems becomes larger and the specifications of systems changes more frequently, development and maintenance becomes more difficult, leading to higher costs and delays. In recent years, huge business systems as mainstay systems in large organizations have been built in the cloud but the problem of combinatorial explosion in development has not been solved. The era of cloud computing requires a more powerful mathematical background to model the cyberworlds and to prevent combinatorial explosions. In the cloud, every business object and business logic should be expressed in a unified form to eliminate discontinuity between systems or functions and to meet changes in user requirements. The needed mathematical mechanisms are: 1. disjoint union of spaces by an equivalence relation; 2. change in spaces to preserve invariants; 3. attachment of different spaces by an equivalence relation; 4. space with dimensions as a special case; 5. modeling of the T0-separation axiom, in addition to Hausdorff axioms, in topology. We consider the Incrementally Modular Abstraction Hierarchy (IMAH) that one of authors (T. L. Kunii) proposes able to satisfy the above requirements, as it models the architecture and the dynamic changes of cyberworlds from a general level (the homotopy level) to a specific one (the view level), preserving invariants while preventing combinatorial explosion [1]. It also benefits the reuse of information, guaranteeing modularity of information based on the mechanism of disjoint union. Unlike IMAH, other leading data models do not support the disjoint union or the attaching function by equivalence relation. In this research, one of the authors (Y. Seki) proposed a finite automaton called Formula Expression as a development tool to realize IMAH. One of authors (T. Kodama) has actually designed spaces and implemented the data processing system called the Cellular Data System (CDS) using Formula Expression. In this paper, we put emphasis on practical use by taking up some examples. First, we design geometrical objects and functions in Formula Expression on the presentation level and implemented them. We demonstrate the effectiveness of CDS by developing a general business application system of a route search system and abbreviate the process of implementing application programs.

II. IMAH AND FORMULA EXPRESSION

A. The Incrementally Modular Abstraction Hierarchy

The following list is the Incrementally Modular Abstraction Hierarchy to be used for defining the architecture of cyberworlds and their modeling:

1. The homotopy (including fiber bundles) level  
2. The set theoretical level  
3. The topological space level  
4. The adjunction space (or attaching space) level  
5. The cellular space level  
6. The presentation (including geometry) level  
7. The view (also called projection) level

In modeling cyberworlds in cyberspaces, we define general properties of cyberworlds at the higher level and add more specific properties step by step while climbing down the Incrementally Modular Abstraction Hierarchy.
The properties defined at the homotopy level are invariants of continuous changes of functions. The properties that do not change by continuous modifications in time and space are expressed at this level. At the set theoretical level, the elements of a cyberspace are defined, and a collection of elements constitutes a set with logical calculations. When we define a function in a cyberspace, we need domains that guarantee continuity such that the neighbors are mapped to a nearby place. Therefore, a topology is introduced into a cyberspace through the concept of neighborhood. Cyberworlds are dynamic. Sometimes cyberspaces are attached together, an exclusive union of two cyberspaces where attached areas of two cyberspaces are equivalent. It may happen that an attached space is obtained. These attached spaces can be regarded as a set of equivalent spaces called a quotient space that is another invariant. At the cellular structured level, an inductive dimension is introduced into each cyberspace. At the presentation level, each space is represented in a form which may be imagined. At the set theoretical level, the elements constitutes a set with logical calculations. When elements of a cyberspace are defined, and a collection of elements is no confusion, ×, ( ) can be abbreviated. + means disjoint union and is expressed as Σ specifically and × is also expressed as Π. In short, you can say "a formula consists of an addition of terms, a term consists of a multiplication of factors, and if the () or {} bracket is added to a formula, it becomes recursively the factor". In Formula Expression, four maps (the expansion map, the bind map, the division map, the attachment map) are defined [9].

Figure 1. An example of e-manufacturing on an adjacency space level

B. The Definition of Formula Expression

Formula Expression in the alphabet is the result of finite times application of the following (1)-(7).

1. a (a ∈ Σ) is Formula Expression
2. unit element ε is Formula Expression
3. zero element φ is Formula Expression
4. when r and s are Formula Expression, addition of r+s is also Formula Expression
5. when r and s are Formula Expression, multiplication of r×s is also Formula Expression
6. when r is Formula Expression, (r) is also Formula Expression
7. when r is Formula Expression, [r] is also Formula Expression

Strength of combination is the order of (4) and (5). If there is no confusion, ×, ( ) can be abbreviated. + means disjoint union and is expressed as Σ specifically and × is also expressed as Π. In short, you can say "a formula consists of an addition of terms, a term consists of a multiplication of factors, and if the () or {} bracket is added to a formula, it becomes recursively the factor". In Formula Expression, four maps (the expansion map, the bind map, the division map, the attachment map) are defined [9].

C. The Grammar that generates Formula Expression

The grammar that generates Formula Expression is defined as follows:

We assume that Σl is a set of ideograms and its element is w ∈ Σl.

G = ([E, T, F, id], {Σe, +, ×, (, ), [ , ]}, P, E),
P = {E→T[E]+T, T→F[T][E][F→{E}][E][id, id→w]}

Here, E is called a formula, and T is called a term, and F is called a factor, and id is called an identifier; + is called an addition operation, × is called a multiplication operation, ( ) or [ ] bracket is called a 1st bracket, and { } is called a 2nd bracket. If you paraphrase the above-mentioned grammar, you can say "a formula consists of an addition of terms, a term consists of a multiplication of factors, and if the 1st or 2nd bracket is added to a formula, it becomes recursively the factor". And when a term is a component of a formula, we say that the formula has the term. And when a factor of the bracket that includes a term is a component of a formula, we say that the formula includes the term. It is the same with a term and a factor. An example is shown below.

The term "a(b+c){d+e}" has factors "a", "(b+c)", "{d+e}", and includes factors "a", "b", "c", "d", "e".

D. A Conditional Formula Search

A function for specifying conditions defining a condition formula by Formula Expression is supported in CDS. This is one of the main functions, and the map is called a condition formula processing map. A formula created from these is called a condition formula. "!" is a special factor which means negation. Recursivity by () in Formula Expression is supported, so that the recursive search condition of a user is expressed by a condition formula. The condition formula processing map f is a map that gets a disjoint union of terms that satisfies a condition formula from a formula. When condition formula processing is considered, the concept of a remainder of spaces is inevitable. A remainder acquisition map g is a map that has a term that doesn’t include a specified factor. If you assume x to be a formula and p, !p, p+q, p×q, !(p+q), !(p×q) to be condition formulas, the images of (x, p+q), (x, p×q), (x, !(p+q)), (x, !(p×q)) by f, g are the following:

\[ f(x, p) = g(x, \lnot p) \]
\[ f(x, \lnot p) = g(x, p) \]
\[ f(x, p+q) = f(x, p) + f(g(x, p), q) \]
\[ f(x, p\times q) = f(f(x, p), q) \]
\[ f(x, !(p+q)) = g(g(x, p), q) \]
\[ f(x, !(p\times q)) = g(f(f(x, p), q)) \]

Figure 2 is each image by the map \( f \). It is obvious that any complicated condition formula can be processed by the combinations of the above six correspondences.

A simple example is shown below.

Formula:

\[
\text{animal\{color+size\}\{flesheating(bear\{brown+big\}+monkey\{brown+small\}+orangutan\{darkbrown+big\}+tiger\{brown\times black+big\}+fox\{brown\times white+small\}+bear\{black+big\})\}+grasseating(horse\{white+brown+middle\}+panda\{black+white+big\}+zebra\{black+white+middle\}+giraffe\{yellow\times black+verybig\}+elephant\{gray+verybig\}+mouse\{gray+verysmall\})
\]

User requirement: “information about a horse and a zebra in \( x \) is required”

A condition formula: “hor\(s\)e+ze\(b\)ra”

\( f(\text{formula}, \text{“hor\(s\)e+ze\(b\)ra”}) \)

\( = f(x, \text{horse}) + f(g(x, \text{horse}, \text{zebra}) \)

\( = \text{animal\{color+size\}\{grasseating(horse\{white+brown+middle\}+zebra\{black+white+middle\})} \)

III. THE DESIGN OF GEOMETRICAL DATA AND APPLIED MAPS

A. The design of a point, continuous segment line(s) and a polygon

We define geometrical objects of a point, continuous segment line(s) and a polygon on Formula Expression as follows.

1. A point is expressed as an identifier
2. Continuous segment line(s) are expressed as multiplication of identifiers of points

For example, continuous segment lines which consist of points \( a, b \) and \( c \) are expressed as \( axbxc \).

(3) A polygon is expressed as a multiplication of identifiers whose ends are equal.

For example, a triangle (a type of a polygon) which consists of identifiers \( a, b \) and \( c \) is \( axbxcxa \) (or \( bxcxa\times b or c\times axbxc \)).

In this paper, numerical values such as the length of a segment line and the area of a polygon are not defined.

B. Left/right tension maps

The left tension map \( f \) is defined this way: If you assume that \( a, b, c, d \) and \( e \) are arbitrary factors and the entire set of formulas to be \( A, f: A \rightarrow A \), the map \( f \) is the following:

1. \( f: a \times b \times c \times d \times e, c \rightarrow c \times (b \times a + d \times e) \) (Figure 5)
2. \( f: a \times b \times c, d \rightarrow a \times b \times c \) (except in (1))

In the same way, the right tension map \( g \) is defined as follows:

1. \( g: a \times b \times c \times d \times e, c \rightarrow (a \times b + c \times d \times e) \times c \) (Figure 6)
2. \( g: a \times b \times c, d \rightarrow a \times b \times c \) (except in (1))

In short, a given term is tensed to the left or the right by a specified factor through the maps \( f \) and \( g \). In these, the specified factor is called a tension factor.

C. Left/right edge open maps

The left edge open map \( h \) is defined this way: If you assume that \( a, b, c, d \) and \( e \) are arbitrary factors and you assume the entire set of terms to be \( A, h: A \rightarrow A \), then \( h \) is the following:

1. \( h: c \times (b \times a + d \times e) \rightarrow a \times b \times c \times d \times e \) (Figure 6)
2. \( h: a \times b \times c \rightarrow a \times b \times c \) (except in (1))

In short, a given term is tensed to the left or the right by a specified factor through the maps \( f \) and \( g \). In these, the specified factor is called a tension factor.
In the same way, the right edge open map \( i \) is defined this way: If you assume that \( a, b, c, d \) and \( e \) are arbitrary factors and you assume the entire set of terms to be \( A \), \( i: A \rightarrow A \), then map \( i \) is the following:
1. \( i: (a\times b+e\times d)\times c \rightarrow a\times b\times c\times d\times e \)
2. \( i: a\times b\times c \rightarrow a\times b\times c \) (except in (1))

D. Other applied maps
-A common identifier detection map-
This map is used to obtain a disjoint union of common identifiers among terms. A simple example of detecting common identifiers from two different terms is shown below.
\[ f'(a\times b\times c\times d', e\times b\times f\times d') = b + d \]

E. Implementation
This system is a web application developed using JSP and Tomcat 5.0 as a Web server. The client and the server are the same machine. (OS: Windows XP; CPU: Intel Pentium 3, 2.1GHz; RAM: 2.1Gbyte; HD: 20GB) The following is the coding for the left/right tension map. The focus is the recursive process (line 7, in bold) that is done if a coming numerical calculation is of the type ().

```
1 term = null; factor = null;
2 while(factor is not null){
3 term = getTerm(factor);
4 while(term is not null){
5 factor = getFactor(term)
6 if(factor is of the type ()){
7 factor = id (the contents);
8 } factor = getLeft/RightFactor(factor);
9 newLeft/RightFactor +=newLeft/RightFactorXLeft/RightFactor
10 newTerm = newTerm + newLeft/RightFactor;
11 newFormula = newFormula + newTerm;
12 }
13 return newFormula;
```

IV. DEVELOPMENT OF THE CORE LOGIC OF A ROUTE SEARCH SYSTEM

A. Outline
We take up the simple example of the core logic development of a route search system, which is extremely important in most traffic industries, in order to show how simplify data modeling when you use CDS. Here, actual data and functions are simplified to focus on verifying development of core processing without losing generality.

Firstly, terms for each train line are designed as a topological space. Secondly, terms expressing train lines are created for data input according to the space design and added to a formula, forming a disjoint union of topological spaces. Thirdly, route searches are done using the maps of CDS to satisfy user requirements.

B. The design of topological spaces
There are three kinds of data objects in a route modeling: 1. a radial line which has a beginning point and an end point, 2. a loop line (such as a train loop line or bus loop) and 3. a crossing point (such as a transfer station or an intersection of streets). These three data architectures are designed according to the design (IIIA) as follows.

1) A radial line
A radial line is designed as a segment line where each identifier is a train station.

2) A loop line
A loop line is designed as a polygon where each identifier is a train station.

3) A crossing point
A crossing point is designed as an attaching point [11]

C. Data input according to the design and data output through the maps
Assume that there is a red line that runs east-west (whose stations are \( x, b, d \) and \( y \)), a blue line that runs north-south (whose stations are \( f, h, c \) and \( g \)) and a green loop line (whose stations are \( a, b, c, d \) and \( e \)) and that there are transfer stations \( (b, c, d \) and \( e) \) as crossing points, as shown in Figure 7.

![Figure 7. An example of a topological space of train lines](image)

You create formulas for each topological space according to the space design and add them together. The data created is the following:

\[ formula1: x\times b\times d\times y+f\times e\times h\times c\times g+x\times b\times a\times e\times d\times c \]

D. Data output through the maps
If you want to know all routes from the starting point \( f \), firstly, you get a term which includes an identifier \( f \) as a starting point from \( formula1 \) using the condition formula processing map \( G \) (II.D).

\[ G (formula1, 'f') = f\times e\times h\times c\times g \] (term1, the blue line)

Secondly, if you use the common identifier detection map (II.D) for term1, you can detect ‘\( e \)' and ‘\( c \)' as attaching points with the terms \( c\times b\times a\times e\times d\times c \) (term2, the green line). Next, you can get the term \( e(a\times b\times c+d\times c) \) from term2 through the left tension map (II.B) by the attaching point ‘\( e \)' and attach it to term1. The result is the following.

\[ f\times e(a\times b\times c+d\times c)+ h\times c\times g \] (term3)

You do the same with attaching point ‘\( c \)' and get the result below.

\[ f\times e((a\times b\times c+d\times c)+ h\times c((b\times a+d\times e\times a)+ g)) \] (term4)
In the same way, you attach the term of the red line by attaching points to term 4 and you get the result below.

\[ f \times e((a \times b((x+d \times y)+c)+d((b \times x+y)+c))+h \times c((b((x+d \times y)+a)+d(((b \times x+y))+e \times a))+g)) \] (term 5)

You can know all routes from \( f \) in term 5. Next, if you want to know all routes from \( f \) to \( x \), you use the condition formula processing map \( G \) and expansion map \( H \) [11] for term 5.

\[ H(G(\text{term 5}, 'f \times x')) \]
\[ = H(f \times e(a \times b \times x+d \times b \times x)+h \times c(b \times x+d \times b \times x)) \]
\[ = f \times e \times a \times b \times x+f \times e \times d \times b \times x+f \times e \times h \times c \times b \times x+f \times e \times d \times b \times x \]

From the result, you can know all routes you want. The figure of all routes from \( f \) to \( x \) is shown in Figure 8.

![Figure 8. All routes from f to x](image)

Next, if you want to know all routes from \( f \) to \( y \) via \( d \), you use the map \( G \) and \( H \) for term 5 again.

\[ H(G(\text{term 5}, 'f \times d \times y')) \]
\[ = H(f \times e(a \times b \times d \times y+d \times b \times y)+h \times c(b \times d \times y+d \times b \times y)) \]
\[ = f \times e \times a \times b \times d \times y+f \times e \times d \times b \times y+f \times e \times h \times c \times b \times d \times y+f \times e \times d \times b \times d \times y \]

In the same way, from the result, you can get the information you want. The figure of all routes from \( f \) to \( y \) via \( d \) is shown in Figure 9.

![Figure 9. All routes from f to y via d](image)

E. Considerations

In general, it becomes complicated to model the core logic of a route search system because the general modeling of undirected graph data and the obtaining of its subsets by functions is difficult using the existing data model. As a result, in many cases, the development of core logic of the route search system is costly and the functionality or space design is actually limited. This example shows that if you only have to design formulas for the train line spaces such as in IV.B and input the route data according to the design, you can output the required data using the maps of CDS, thereby reducing the number of application programs. In other words, you can simplify data modeling for the route search using CDS.

V. LITERATURE REVIEW

The distinctive features of our research are the application of the concept of topological process, which deals with a subset as an element, and that the cellular space extends the topological space, as seen in Section 2. Relational OWL as a method of data and schema representation is useful when representing the schema and data of a database [2][5], but it is limited to representation of an object that has attributes. Our method can represent both objects: one that has attributes as a cellular space and one that does not have them as a set or a topological space. Many works applying other models to XML schema have been done. The motives of most of them are similar to ours. The approach in [8] aims at minimizing document revalidation in an XML schema evolution, based in part on the graph theory. The X-Entity model [9] is an extension of the Entity Relationship (ER) model and converts XML schema to a schema of the ER model. In the approach of [6], the conceptual and logical levels are represented using a standard UML class and the XML represents the physical level. XUML [10] is a conceptual model for XML schema, based on the UML2 standard. This application research concerning XML schema is needed because there are differences in the expression capability of the data model between XML and other models. On the other hand, objects and their relations in XML schema and the above models can be expressed consistently by CDS, which is based on the cellular model. That is because the tree structure, on which the XML model is based, and the graph structure [3][4][7], on which the UML and ER models are based, are special cases of a topological structure mathematically. Entity in the models can be expressed as the formula for a cellular space in CDS. Moreover, the relation between subsets cannot in general be expressed by XML.

VI. CONCLUSIONS

We have developed a data processing system called the Cellular Data System (CDS) based on IMAH. In this paper, we introduced the basic geometrical spaces (a point, continuous segment line(s) and a polygon) and the maps (left/right tension maps, left/right edge open maps) into CDS, and successfully applied it to the core logic development of a route search system. Using this, the core logic development of the system becomes much simpler, reducing the number of application programs. As a result, use of CDS can make developers more creative, while preventing combinatorial explosions.
You can see that geometrical design often appears in our surroundings. The examples are below (Figure 10, 11).

![Figure 10. A baby arm before the edge open function](image1)

![Figure 11. A baby arm after the edge open function](image2)

REFERENCES


