China Historical GIS Spatio-Temporal Database and Web Implementation

Merrick Lex Berman - August 2001

1. CHGIS Database Design
2. Implementation Problems
3. Tracking Spatio-temporal Entities
4. Web Implementation of Spatio-Temporal Searches & Mapserver

1. CHGIS Database Design

The draft Spatio-Temporal Database Design, by Lawrence Crissman (Griffith University), called for the construction of relational tables in a row-versioned database.[1] The primary objective of the database was to identify historical administrative units in China, and to show both their hierarchical relationships within the Chinese administration and their changes over time. It was proposed that each change of any particular unit be reflected in the database by the addition of a new row, with beginning and ending dates recording the temporal extent of the unit during one "instance." Although the database design sought to keep track of various attributes of the records as needed in related tables, there were three fairly fundamental issues that needed to be addressed:

(a) redundant storage of attributes for each level in the administrative hierarchy
(b) use of hierarchical geocodes (which cannot easily handle multiple parents)
(c) inability to add intervening levels into the hierarchy without requiring excessive renumbering of child unit geocodes, or necessitating the re-organization of the database to accommodate new levels in the hierarchy

In terms of redundant storage of attributes, the original database design called for the entry of both the Romanized and Chinese placenames for EACH level in the adminstrative hierarchy for EVERY row. In other words, the record for Huangzhai Township (located in Yangqu County, Taiyuan Municipality, Shanxi Province,) would have to include all of these names and their units in separate fields. A partial list is seen in Figure 1.

Place_PY	Place_HZ	Unit_PY	Unit_HZ	Lvl_3_PY	Lvl_3_HZ	Lvl3_u_py	Lvl3_u_hz	Lvl_2_PY	Lvl_2_HZ
Huangzhai	黄寨	Zhen	鎮	Yangqu	陽曲	Xian	縣	Taiyuan	太原

Figure 1: Administrative Hierarchy Attributes for a single record

The strategy was to include the information in the tables while organizing each record into the appropriate place in the administrative hierarchy, then to assign a specific **historical administrative code** for each level in the hierarchy, so that the text entries could be swapped out for the codes. Using Guobiao codes[2] as an example, the code might consist of three levels with two-digits each. Shanxi Province = 14, Taiyuan = 01, and Yangqu = 22. The Huangzhai Township record could then be recorded in an abbreviated format, by referring to its higher level units, as in Figure 2.

record_code	Place_PY	Place_HZ	Unit_PY	Unit_HZ	Lvl_3_code	Lvl_2_Code	Lvl_1_Code
140122AA	Huangzhai	黄寨	Zhen	鎮	22	01	14
140122	Yangqu	陽曲		縣	22	01	14
140100	Taiyuan	太原		市		01	14
140000	Shanxi	山西		省			14

Figure 2: Replacing redundant entries with adminstrative codes

The use of historical administrative codes represents a considerable savings in tablespace, but also leads us to the second issue, which is that we can only use this coding method to define a single inflexible hierarchy. In dealing with historical geography, we may not always be working within the official administrative hierarchy of a single regime…indeed, as in the case of China, we may need to describe many competing claims on geographic and political space, even though the actual control of the contested space may not have belonged to either side. For example, we might have the case of the Dali Kingdom, which existed independently for many centuries, fighting wars with both Chinese and Tibetan states, while all three laid claims to the same territory.

Let us suppose that China had a nominal claim to an administrative office, Dali Xian, located under the nominal jurisdiction of Yunnan Fu. Following the previous example, we might have the administrative attributes and codes shown in Figure 3.

record_code	Place_PY	Place_HZ	Unit_PY	Unit_HZ	Lvl_3_code	Lvl_2_Code	Lvl_1_Code
3344	Dali	大理	Xian	縣		44	33
3300	Yunnan	雲南	Fu	府			33

Figure 3: Example of adminstrative codes for Dali Xian, during the Qing Dynasty

However, the Dali Kingdom might refer to the same place as Yangjuhu, a city in the Dali Kingdom…and, of course, they might not be referring to the same space, but a different somewhat overlapping area. We would have to compile a separate set of administrative levels for each case of overlap, in this case, we could create a parallel hierarchy, as shown in Figure 4.

record_code	Place_PY	Unit_PY	Lvl3_code	Lvl2_code	Lvl1_code
8899	Yangjuhu	Cheng		99	88
8800	Dali	Guo			88

Figure 4: Parallel administrative hierarchy

Then we need a workaround table that shows which units in one hierarchy are related to units in the other hierarchy, as in Figure 5.

record_code	Place_PY	Alt_Hierarchy
3344	Dali	8899

FIGURE 5: Alternate Hierarchy Table linking units in parallel hierarchies

Although this will allow us to discover the alternate hierarchies it is quite cumbersome to keep track of, and it reveals a serious issue that cannot be resolved: the need to allow for creation of new, intervening levels in the administrative hierarchy.

The coding system described up to this point relies on the idea of setting up a known number of levels in the administrative hierarchy in advance of entering information into the database. If, at any point, we discovered a NEW level that must be placed between any two existing levels, it would require the renumbering of ALL the RECORDS in the database, from the intervening level down to the lowest level in the hierarchy. For example, Figure 6 demonstrates that if we had to create an intervening level between Dali Kingdom and Yangjuhu, that would force a new level to be placed in the Yunnan Fu – Dali Xian relationship also.

record_code	Place_PY	Lvl2_code	Lvl-NEW-code	Lvl1_code
330044	Dali	44	00	33
330000	NEW-LEVEL		00	33
330000	Yunnan			33

record_code	Place_PY	Lvl2_code	Lvl-NEW-code	Lvl1_code
882299	Yangjuhu	99	22	88
882200	NEW-LEVEL		22	88
880000	Dali Kingdom			88

Figure 6: Creating new levels in the administrative hierarchy forces renumbering

Such a massive reorganization of the database, precipitated by the discovery of new information is clearly unacceptable. In the next section we will look at specific problems encountered in the database implementation and the reasons why the database has been redesigned according to a simpler part-whole relationship model.

2. Implementation Problems.

During the initial data collection and data entry stage, undertaken by a team of researchers at the Center for Historical Geography (Fudan University),[3] the entry of redundant administrative hierarchy information into spreadsheets proved to be fairly tedious, although it did have the advantage of allowing for a visual verification of the hierarchical attributes. See for example the table in Figure 7.

place_present	dyn_ch	prov_	level2_py_name	level2_ch
山西曲沃县西南隘口	清朝	山西	Pingyang Fu	平阳府
山西潞城市西安昌	清朝	山西	Lu'an Fu	潞安府
山西左云县北宁鲁堡	清朝	山西	Shuoping Fu	朔平府
山西大同市西南安宿疃	清朝	山西	Datong Fu	大同府

Figure 7: Redundant entry of administrative level information

Note the columns in the center, for Dynasty and Province, which were redundant for this entire file. As we began to assemble files for the Qing Dynasty, the need for reducing tablespace and eliminating redundancy became more pronounced. Although the use of a code number to replace the text strings in each of the administrative hierarchy fields would have saved some disk space, it did not deal with the inherent redundancy of storing all of the hierarchical level values for each record. The idea of maintaining exact coded equivalents for each of the administrative hierarchy levels was utilized in the coding system for the China Historical GIS developed by the late Robert Hartwell.[4]

pinyin	admin_type	sup_prov	province	circuit	prefect	dep_pref
Xincheng	Xian	Ming dynasty	Jingshi		Baoding	
Xincheng	Xian	Ming dynasty	Shandong		Ji'nan	
Xincheng	Xian	Ming dynasty	Zhejiang		Hangzhou	
Xincheng	Xian	Ming dynasty	Jiangxi		Jianchang	
Xincheng	Xian	Ming dynasty	Guangxi		Qingyuan	
Xincheng	Suo	Ming dynasty	Guizhou			Annanwei
Xincheng	Wei	Ming dynasty	Jingshi			Beiping

Code	Level1_h	Level2_h	Level3_h	Level4_h	Level5_h
M001100220025	0	11	0	22	0
M002500210026	0	25	0	21	0
M004100210027	0	41	0	21	0
M004500260022	0	45	0	26	0
M008500230032	0	85	0	23	0
M009500003722	0	95	0	0	37
M001100002422	0	11	0	0	24

Figure 8: Hartwell administrative coding

An example of Hartwell's table is shown in Figure 8, showing seven records with ambiguous placenames but completely different positions in the administrative hierarchy. The units are clearly and easily differentiated using Historical Administrative Codes, shown on the bottom half of Figure 8. The advantage of using unique codes for disambiguation of placenames works perfectly in the Hartwell datasets, which are limited to single slices in time, and which are kept in separate files for each Dynasty. However, when we introduce multiple temporal instances of each unit, and must allow for creation of additional

intervening hierarchical levels as they are discovered, the use of the fixed number of fields and concatenated historical codes becomes overly complex.

The construction and syntax of the hierarchical coding system was originally conceived of as a tool for uniquely identifying an instance of a particular historical unit, not only in terms of its administrative rank, but also the spatial data version, and temporal sequence. It was proposed that each value would be held separately in its relevant field, but that these would all be concatenated into a single string, that could act as a unique ID for the historical instance and its spatial representation of any particular unit. The idea was to have "one unique code assigned to each historical instance of an administrative unit." The resulting code was to have looked something like the string shown in Figure 9.

Figure 9: Concatenated Historical Administrative Code

According to Crissman's proposal, the first two postions "Za" would be used to indicate the Dynastic Period, and then a series of two digit codes would be used for each level in the adminstrative hierarchy, alternating from numerical two-digit codes, to capitalized two-letter alphabetic codes, back to two-digit codes, then to lower-case alphabetic codes, and so on, down to the lowest level needed. Crissman also proposed removing the codes for internening levels that had NULL values, as in the example where "acBA" have no intervening numeric code. In addition to the administrative coding system, brackets were proposed to contain references to Spatial Data Versions, as in "SV2" and Variants "V3," where for example, a particular subset of the spatial data had several different possible variants even within the same overall spatial dataset.

Finally, Crissman proposed to keep track of the temporal instance of the changes in the historical place entities as a number following the brackets. Crissman proposed that the actual temporal sequence could be reflected in this decimal by using intervening decimal values. For example, if we had temporal instances 1, 2, 3, and needed to add an intervening unit between 2 and 3, we could use 2.5. Later, if a new instance needed to be added between 2 and 2.5, we could use 2.3, and so on, ad infinitum.

Although some deal of time was spent trying to modify and redesign the proposed coding system, when it came to implementing any type of codes into RDBMS, there was no way to justify burdening the database developers and the users with such a complicated and awkward scheme. Despite the simplicity and convenience of using geocodes in certain situations, my own conclusion was that there were simpler and better ways to keep track of the same information.

First, the idea of using such a long and absurdly complex code as the unique ID for each record had to be thrown out. There was no reason to use ANYTHING except auto-incremented unique KEY_ID numbers for each record in the database. The idea of trying to parse and autocheck the weird and complex administrative codes, which might typically be 25 to 30 characters long was ridiculous. Since we only expect to have about 100,000 or fewer records in the first implementation, a simple auto-incremented KEY-ID would be no more than six digits, and could be EASILY checked to see if there were any duplicates or missing numbers in the sequence. As for the attribute information, such as spatial data

versions, or beginning and ending dates, they should be kept in their own fields. Since the unique KEY_ID will identify each historical instance, there is no need to create artificially coded values for any of these attributes.

Second, the hierarchical coding implied knowing what the top level was in every case. For example, if we started the first level with a code indicating the Dynasty, then all of the subsidiary units must lie within the spatial and temporal bounds of the Dynasty. This will suffice only so far as we agree on the adminstrative history of a particular dynasty…whereas, in fact, the beginning and ending years of Dynasties do not always coincide. One Dynastic clan might cling on for years or decades with various territorial claims and strategems, in which case the OVERLAP from the top level unit of one Dynastic hierarchy to another would have to be artificially bridged. The Alternate Hierarchy Table, suggested above, is one solution, but why not simply use the same device in the form of a part-whole relational model, which would allow us to redefine our hierarchies however we like?

The part-whole model is essentially a way to indicate which records are PART-OF some other units, using a many-to-many table. In this case, there is no need for specific hierarchical codes, and each record can simply be given an auto-incremented randomly generated unique identifier, or KEY_ID. In practice, the part-whole model would look like the example in Figure 10, where Dali is shown as part of Yunnan, and Yangjuhu is shown as part of Dali Kingdom.

KEY_ID	Place_PY	PART-OF
44	Dali	33
33	Yunnan	
99	Yangjuhu	88
88	Dali Kingdom	

Figure 10: Part-Whole relationship model

Now let's consider how we could take advantage of this model to redefine a hierarchy using a separate many-to-many table. For instance, let's just take a look at some places in Shanxi Province, shown in the part-whole model in Figure 11.

KEY_ID	Place_PY	Place_HZ	PART-OF
22	Shanxi sheng	山西省	
23	Taiyuan shi	太原市	22
24	Datong shi	大同市	22
25	Yangquan shi	陽泉市	22
26	Pingding	平定	25
27	Yu xian	盂縣	25
28	Changzhi shi	長治市	22
29	Changzhi	長治	28
30	Lucheng	潞城	28
31	Xiangyuan	襄垣	28

Figure 11: Shanxi Province – Part-Whole model

Here we see that Taiyuan, Datong, Yangquan, and Changzhi, are all "part-of" Shanxi Sheng, in other words immediately subordinate to Shanxi in the administrative hierarchy. Pingding and Yu Xian are also shown to be immediately under Yangquan Shi; while Changzhi, Lucheng, and Xiangyuan counties are all under Changzhi Shi. The latter three units are not shown as explicitly subordinate to Shanxi, but a query of their parent unit and on up the chain will reveal the administrative hierarchy.

3. Tracking Spatio-Temporal Entities

Now let us turn to the problem of tracking changes in records over time. Based on the example shown in Figure 11, imagine that we wanted to redefine the Shanxi hierarchy based on our own definition of religious sect activity rather than administrative divisions. Perhaps there was a secret society called the Red Mustachios, wildly popular in Lucheng, Pingding and Datong, but practically unseen in the rest of Shanxi. Any record in our database can be easily rearranged in a related table, as shown in Figure 12.

Group_rec	KEY_ID	Name	Name_HZ	Part_of_Group
1		Red Mustachios Sect	紅鬍子	
2	24	Datong shi	大同市	1
3	26	Pingding	平定	1
4	30	Lucheng	潞城	1

Figure 12: Example of a user-defined Alternate Group Table

This type of User-Defined Alternate Group Table can easily sort and discover relationships between existing records in the database. The same solution would apply to units that spanned from one dynasty to another, since there is no limit to placing another

higher level unit above any other. But what if, instead of defining higher order groups of smaller units, we want to add an intermediate unit into the hierarchy? After all, the addition of intervening units, either in spatial terms or temporal terms, is an essential flexibility that we must build into the database. Unfortunately, if an intervening level is added into the database, we cannot avoid renumbering CHILD units. In fact, there seems to be no way around this problem.

Basically, the issue is this: let us suppose that we discover a particular date when the Red Mustachios split into two different sub-sects —the Northern Sect, which controlled Datong, and the other, the Southern Sect, controlling both Pingding and Lucheng. If we need to differentiate between the earlier incarnation of the UNITED sect, and the later development of the SPLIT sect, we must DUPLICATE and RENUMBER the records in our database, as shown in Figure 13.

Group_rec	KEY_ID	Name	Name_HZ	Part_of_Group	began	ended
1		Red Mustachios Sect	紅鬍子		780	902
2	24	Datong shi	大同市	5	780	844
3	26	Pingding	平定	5	780	844
4	30	Lucheng	潞城	5	780	844
5		Red Mustachios Sect (Before Split)		1	780	844
6		Northern Sect		1	844	902
7	24	Datong shi	大同市	6	844	902
8		Southern Sect		1	844	902
9	26	Pingding	平定	8	844	902
10	30	Lucheng	潞城	8	844	902

Figure 13: Need to create new records when hierarchy changes

Here you can see that the addition of an intervening level necessitated some crucial changes:

(a) a new record had to be added to define the UNITED sect, before the split

(b) the CHILD units of the UNITED sect had to have their ENDED dates and their PART-OF-GROUP IDs updated to reflect the ENDING of their former UNITED identity

(c) new SUB-SECT records had to be created for the NORTHERN and SOUTHERN sects

(d) the original CHILD units had to be DUPLICATED and renumbered as CHILD units of the new SUB-SECTS

The preceding scenario is a good example of our spatio-temporal entity problem. In a nutshell, we must determine a way to capture the smallest atomic units, both spatially and temporally, in the database, and enable them to be split or merged into new units without having to restructure the database. Therefore, although we are giving up some table space by the creation of new rows in the database whenever a change occurs, the adoption of the part-whole model instead of the historical administrative coding model is the more viable alternative. After all, the new rows are created, but the only MANDATORY attribute needed to define any record's place in the administrative hierarchy is the "part-of" field, which represents a many-fold savings of disk space. The original administrative coding model called for ten administrative levels to be defined, so we are saving many times the disk space that would have been required to store the codes.

Early on in the discussion of the database design, Prof. G. William Skinner strongly advised us to focus on tracking settlements purely by location, rather than their administrative status at any particular date.[5] The idea was that we must have a method of easily discovering the history of a particular place across time, regardless of it various roles in the historical records. Even though a particular county office might be established somewhere, and then later abolished, re-established, moved to a new location, then relocated back to the original place, we can assume (lacking any evidence to the contrary) that the human settlement and the local people at the original location carried on with their daily lives throughout the entire parade of administrative changes, affected by but not entirely dependent on what the authorities did.

To expedite the tracking of settlements we proposed the use of a "Site-Code" which was essentially the lat – long coordinates of the settlement, as derived from the contemporary locations in the ArcChina populated places point coverage.[6] The coordinates were to be concatenated together, with a prefix to indicate the origin of the spatial data, for example "AC" meaning the point was based on an existing record in ArcChina, or "FD" for points added by the researchers at Fudan University.

KEY_ID	Place_PY	Place_HZ	Site-Code
301	Cheli	車里	AC100.891E22.091N
1005	Jinghong	景洪	AC100.891E22.091N

Figure 14: Site Codes

In Figure 14 we see an example showing two historical instances in our database which were established at the same location. Regardless of how many different instances there might be in the database for Cheli or Jinghong, they could all be easily discovered using the Site Code. As we assembled the data, the actual coordinates were entered in separate fields, and in the end, there seemed to be no advantage in creating the concatenated code for this purpose either. We decided to keep the coordinates separate, which is much more useful for a number of reasons, and to keep track of the Spatial Data Source in a separate field, as in Figure 15.

KEY_ID	Place_PY	Place_HZ	long	lat	spatial data
301	Cheli	車里	100.891	22.091	ArcChina
1005	Jinghong	景洪	100.891	22.091	ArcChina

Figure 15: Site locations defined by coordinates

Now that our database model includes a unique ID for each historical instance of a place, as well as a coordinate location, and temporal span (using beginning and ending dates), we can begin to think about how we want to define our spatio-temporal queries. But first, let us step back a pace and try to determine exactly what we need to query and what we are defining as an historical instance.

We set out to capture each change of any particular historical administrative unit in the database by the addition of a new row, with beginning and ending dates recording the temporal extent of the unit during one "instance." But what changes are we tracking and what constitutes a change necessitating the creation of a new row in the database? Furthermore, what constitutes a particular "historical administrative unit?"

It will be better to broaden the scope of the database, and to think of the administrative units being tracked as one of many possible types of "entities." Our database model is an attempt to keep track of the processes of change that transform these entities across time. Since the process is continuous and our information rather incomplete, the best we can hope to do is to create records for instances, or "snapshots," of the objects that define the entity at known points in time, while allowing for new intervening records to be introduced later on in case we discover some more information. As for the objects that define the entity, they are made up of spatial, temporal, and thematic attribute sets.[7]

Each one of our database records might also be thought of as event-driven instances. In other words, the creation of a new record in the database, must be predicated upon a particular event that changed one of the objects defining our entity. The entity can be thought of as the sum of a series of instances over time, each instance being triggered by some kind of event. In this sense the entity spans the entire temporal extent of all its component instances, just as there must be a spatial extent that includes all the spatial features contained in the dataset.

For each instance, what should be held as the determining factor in creating a new instance? Should we accept changes in location, area, name, administrative status, and feature type as events that would cause creation of a new instance? Should one event be considered primary, enough to create a new instance, while changes of the other attributes be secondary events, and kept track of in tables related to a single instance? How we answer these questions will determine the nature of the entity that we are trying to describe.

If we take spatial changes as primary, we might start out with a scenario as shown in the left half of Figure 16 at Time 1. The spatial change occurs at Time 2, when polygon 523 expands and acquires and area formerly part of polygon 524. (The red dashed line indicates the old boundary.)

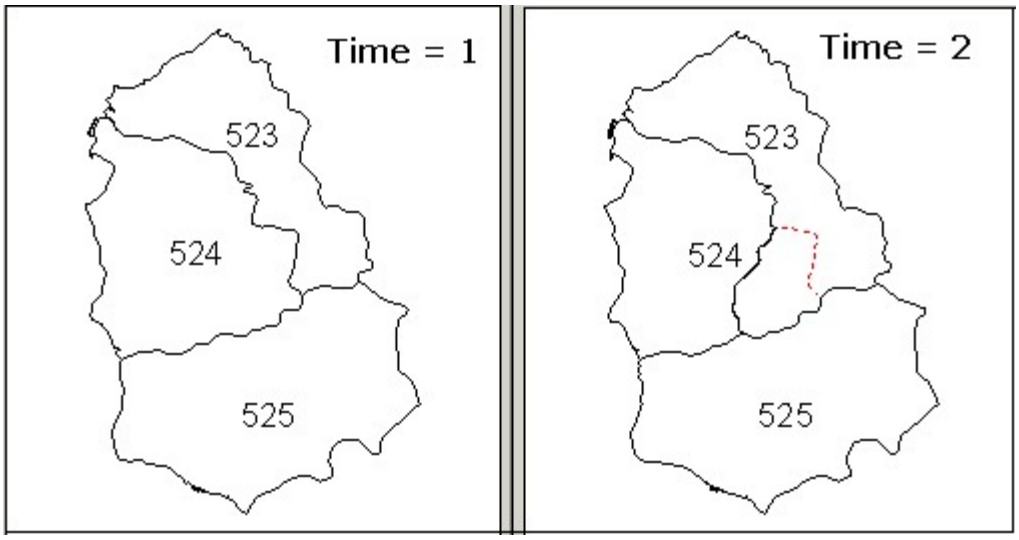

Figure 16: Time One – Spatial Changes as primary events

In this case we would have to create two new instances in the database, because both 523 and 524 have changed. In our spatial dataset, we have chosen to implement the space-time composite model, in which separate polygons are stored, then reassembled to form a time-specific coverage as needed.[8] This model is based on exhaustive spatial coverage, with new spatial objects created for changes in existing spatial objects only. In order to recompose polygons based on particular units at a particular slice in time, we begin with a base map that was created as a complete representation of the basic spatial topology for the desired region. Then the changes are followed forwards or backwards in time as needed, to locate the set of spatial objects needed to represent our selected features for the specific slice in time. In the previous example, the polygons for TIME 2 would be discovered, then the arc separating the two polygons with the same KEY_ID would be removed (the dotted red line in Figure 17).

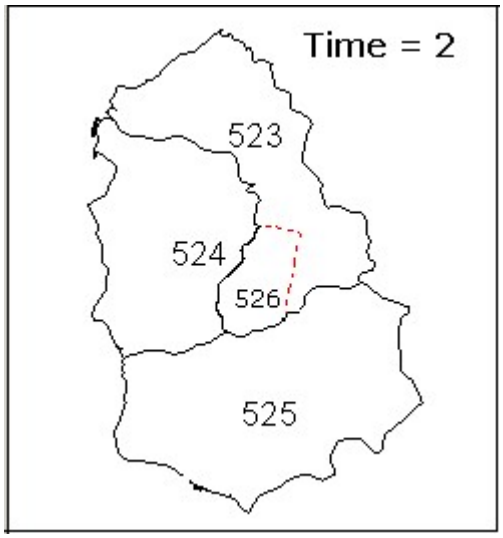

Figure 17: Space-time composite model

Now the question is, what if, in between Time 1 and Time 2, in other words, BEFORE the event that forced us to create an instance for Time 2, the administrative area represented by polygon 524 had a change in placename? Let's say previously it was called Pingding, and subsequently it was called Luqiao. Should we create a new instance, or should we create tables to keep track of thematic attributes (such as placename) as temporally definable sub-events? If we chose to keep track of such sub-events, our Time 1 polygon 524 instance, would have two independent timelines, one for the entire instance, and one to keep track of name changes, as illustrated in Figure 18.

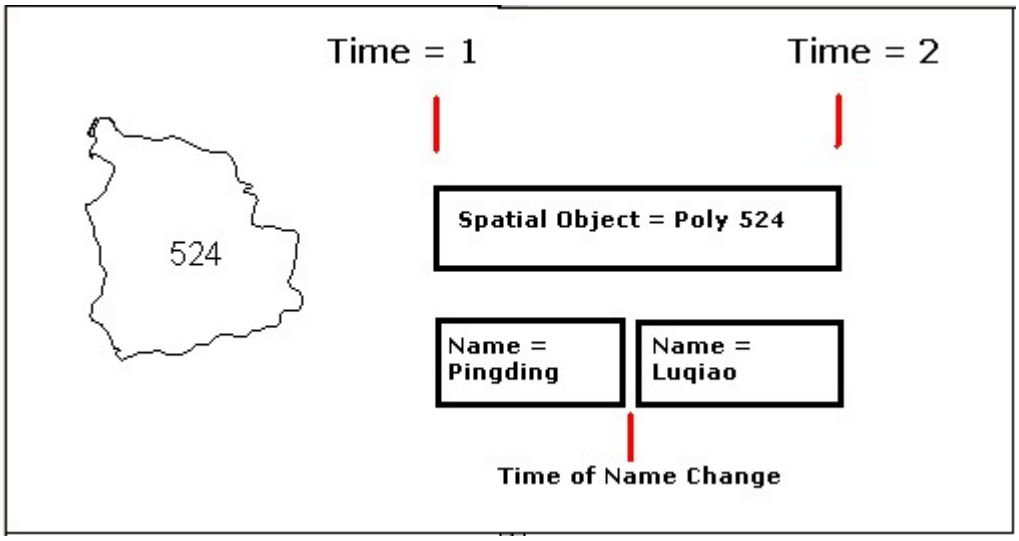

Figure 18: Tracking independent timelines if allowing sub-events

If other attributes, such as adminstrative status, and feature type, are also allowed to change WITHIN a single instance, then the parallel timelines not only begin to populate, but they do so completely out of synchronization. There is no reason to think that name changes will automatically occur at the same time as administrative status changes, or feature type changes, although it is possible. The problem with allowing this sort of temporal diversity is that we can no longer match one instance with its object's attributes without doing additional temporal queries any possible sub-events. How would a query for places named "Pingding" with the administrative status of "Fu" be accomplished if we were to follow the precedent of spatial changes as the primary event for new instances, as shown in Figure 19?

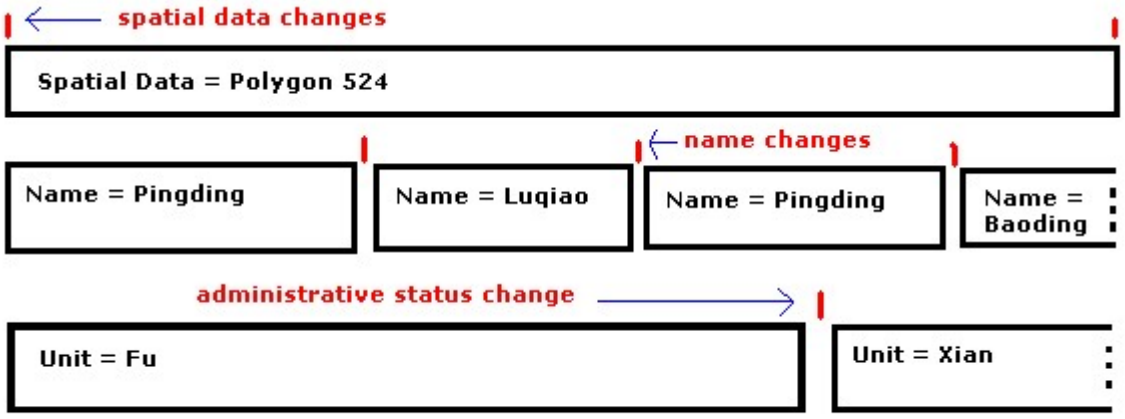

Figure 19: Multiple timelines for sub-events

Here the problems become obvious. Pingding may have had a name change to Luqiao, then back to Pingding again. Meanwhile, the administrative status may have continued as Fu all along, until the second Pingding was suddenly demoted to a Xian. Finally, Pingding's name was changed again to Baoding, which kept the status of Xian, and incidentally, kept both this status and name forward into the next instance, created by the spatial event shown in Figure 16. This patchwork of parallel and unsynchronized attribute data is simply going to play hell with our SQL.

We may someday be able to create object orientated databases to take care of this type of problem for us, but for time being I propose that the database model be implemented with **attributes frozen for each instance**. By knowing in advance that the attributes for a particular instance will not be changing for the duration of the temporal extent of the instance, we gain two key advantages: first, our queries and reporting processes are infinitely simpler; and second, we can re-organize the database according to any of the attributes as a primary factor, by re-assembling the instances for which preceding and subsequent units had the same attribute. This is demonstrated in Figure 20.

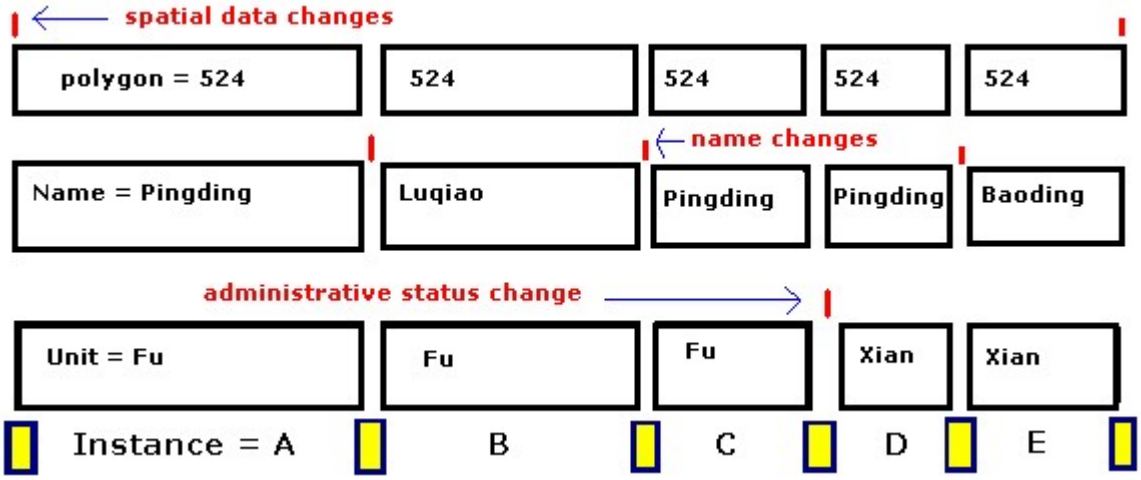

Figure 20: Each Instance with frozen attribute values

By holding attributes frozen for each instance we will greatly simplify our query processes. A hit on Instance C will be able to retrieve its Name (Pingding), Adminstrative status (Fu) and Polygon (524), without any additional sorting or sub-querying. Certainly in the future we will find other attributes that may overlap the instances in the model we have adopted here, but as long as we give equal weight to spatial data change, name change, and feature type change (in this example, administrative status is the feature type), we should have considerable flexibility. Furthermore, since we still do not know exactly which factor is determining our definition of "entity" that we want to keep track of, the division of the instances into the basic spans of time in between each event, allows us to create our own event specifications, just as we did for user-defined groups.

For example, if one researcher was primarily interested in administrative status, a table could be assembled by regrouping the basic instances according to admin status. Meanwhile another researcher who was only interested in spatial changes, could easily trace unchanged polygons and group them together using a relational table. This flexibility for the end-user should be sufficient reason to adopt the frozen attribute model for each temporal instance.

Now, assuming that we have implemented the database along the "instance with frozen attributes" model, we will immediately discover that the beginning and ending dates for each instance become the major dependency.[9] This poses a serious obstacle since historical records are notoriously vague and contradictory when it comes to exact dates. Let's take an example: we know that there are instances A, B, C, and D, and that they certainly occurred in this sequence. However, we have only a beginning date for A and D, and ending dates for C and D. Our uncertainty is depicted in Figure 21.

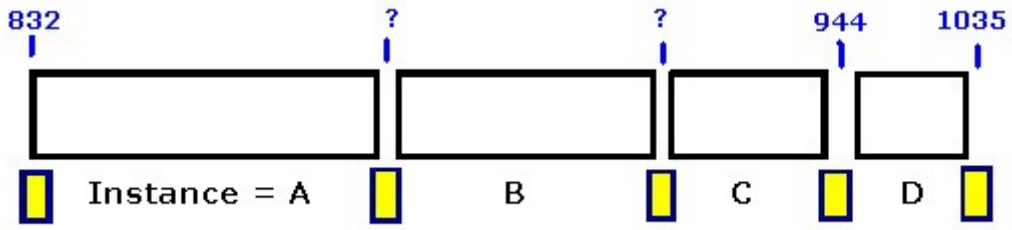

Figure 21: Beginning and Ending Date uncertainty

One way to deal with this problem is to utilize the Temporal Sequence Table, which was originally proposed in the Crissman Database Design to keep track of the sequence of units as they changed over time. The idea is to define a unit as "PrecededBy" another unit. In this way we could keep accurate track of temporal sequence, without having to define exactly the temporal extent. Also, in order for temporal searches to discover the correct units, I propose that we use a combination of the next available **preceding beginning date**, and or the next available **subsequent ending date**, together with an uncertainty flag, here shown as "*." The basic idea is shown in Figure 22.

Name	begin_date	end_date	PrecededBy
A	832	944*	
B	832*	944*	A
C	832*	944	B
D	944	1035	C

Figure 22: Showing Uncertain Temporal Sequence

In this example, we might search for units that were extant in the year 850. The results of our query would include instances A, B, and C, and would also tell us that the beginning dates for B and C are uncertain, as are the ending dates for A and B. Even so we would be able to discover all of the units based on temporal search, including instance B, which has no recorded temporal extent at all!

Now that we've put together the major building blocks of our spatio-temporal database, let's turn to the issues surrounding querying the CHGIS datasets over the Internet.

3. Web Implementation of Spatio-Temporal Searches & Mapserver

Although the CHGIS project is focussed on the production of the GIS datasets themselves, one of my priorities has been to develop a web-implementation that will allow users to browse and query the datasets over the Internet. In the designing the web interface, my priorities were to:

a. not require any installation of software, except for standard browsers, such as Netscape and Internet Explorer

b. use encodings and font sets that are easy to install and use with the standard browsers

c. allow the users to search by placename, administrative areas, feature type, and by time

d. enable the user to preview the spatial data by using a webmap server

e. avoid overly-complex webpages to speed up the query response time

By implementing an easy to use web interface, I hoped to enable a wide audience to access the data as a reference tool, and by including the webmaps, users could get a sense of what the CHGIS datasets contain before downloading the actual datasets for their own use. Numerous studies of Internet usage have shown that delays of more than several seconds for page requests will drive users away from a website. Therefore, I knew perfectly well that very very few potential users would bother to spend hours downloading the CHGIS data just to see what the contents were like. Whereas, a fast preview of the data, with a limited range of search criteria, would be much more likely to engage users and provide them with

samples of the data contents. In short, the web implementation emphasized speed, simplicity, and consistency, over extensive user customization and analytical functions.

The general overview of the web implementation is shown in Figure 23.

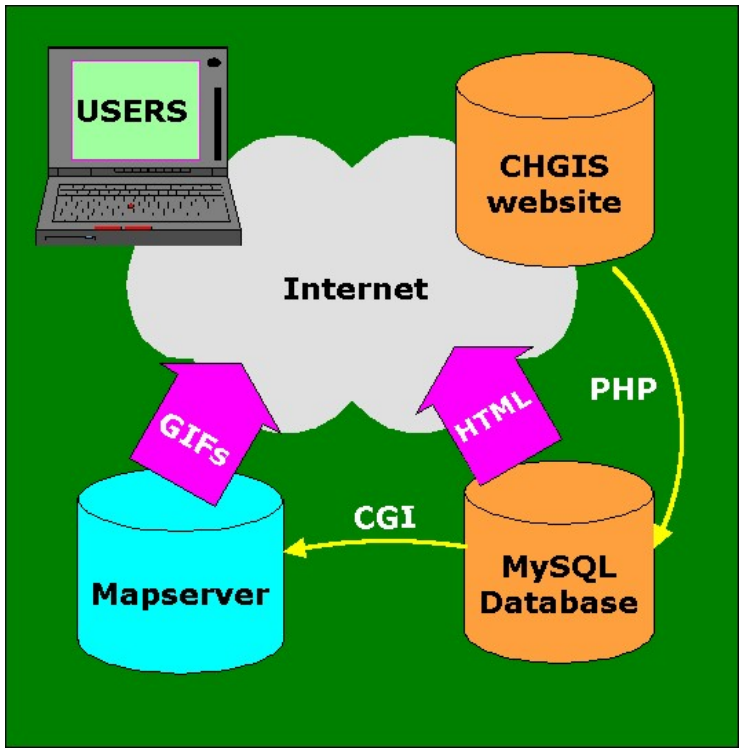

Figure 23: Overview of CHGIS Web Implementation

Users simply go to the CHGIS website. Search engines are available as forms that pass input variables to PHP scripts. The PHP scripts connect to, and pass queries to a MySQL database running on a dedicated server. The results of the search are sent back to the user's browser as ordinary HTML. However, embedded in the HTML are links which can send additional PHP requests to MySQL, or can send CGI calls to Mapserver, also running on the same dedicated server. A typical webform is shown in Figure 24, for searching contemporary Provinces, Prefectures, and Counties (circa 1991).

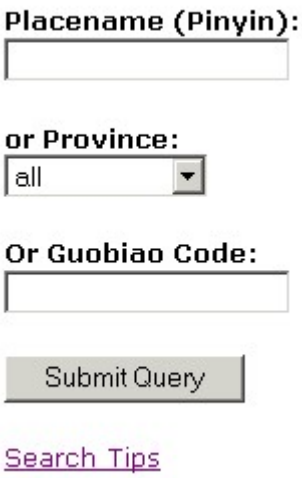

Figure 24: Counties Search Form

The results appear as a list of hits, which contain embedded calls for additional searches. For example, if we searched for the placename “Baisha” we would get results as shown in Figure 25.

Figure 25: Results of Counties Search

Note the links that appear as “Search Harvard Map Collection” and “Search Harvard Yenching Library.” These links are composed on the fly as part of the PHP script, and create a new request to different datasets based on some input variables that were found during the preceding search…in this case, the provinces. So if we were to continue to by clicking on the first link, we would automatically search the holdings of the Harvard Map Collection for “Hainan,” the location of the first Baisha record. The output is shown in Figure 27.

Harvard Map Collection Search Results
2 map records found

Country / Region depicted:	**China**
Publication Date:	**1935**
Vernacular Language:	**Chinese**
Map Scale, :	**1 :190,000**
Pusey Call Number :	**7823 HAI6 1935**
Map Series Area Description:	**Hainan Island**
Number of Index / Locator Sheets:	**1**
Total Number of Map Sheets in Series:	**12**
Detailed Description:	**Maps of Hainan Island, Chinese General Land Survey, 1935. Library has complete coverage of index, but not all of island is represented (southernmost regions not covered).**
Guide Map Showing **Approximate** Area of Map Series:	**Get Map (available for China only)**

Figure 27: Map collection search results

The first of four records in the map collection show a series of maps published in 1935. There are twelve actual map sheets in the series, at a scale of 1:190,000. Note the link at the bottom, which is yet another embedded query, this time a CGI call to Mapserver. In this case, the map collection dataset contains a specific geographic extent, in the form of bounding box coordinates, and when the user clicks on the link, those maximum north, south, east, and west coordinates are sent to Mapserver, which zooms in to that extent using contemporary 1992 county boundary maps, as shown in Figure 28.

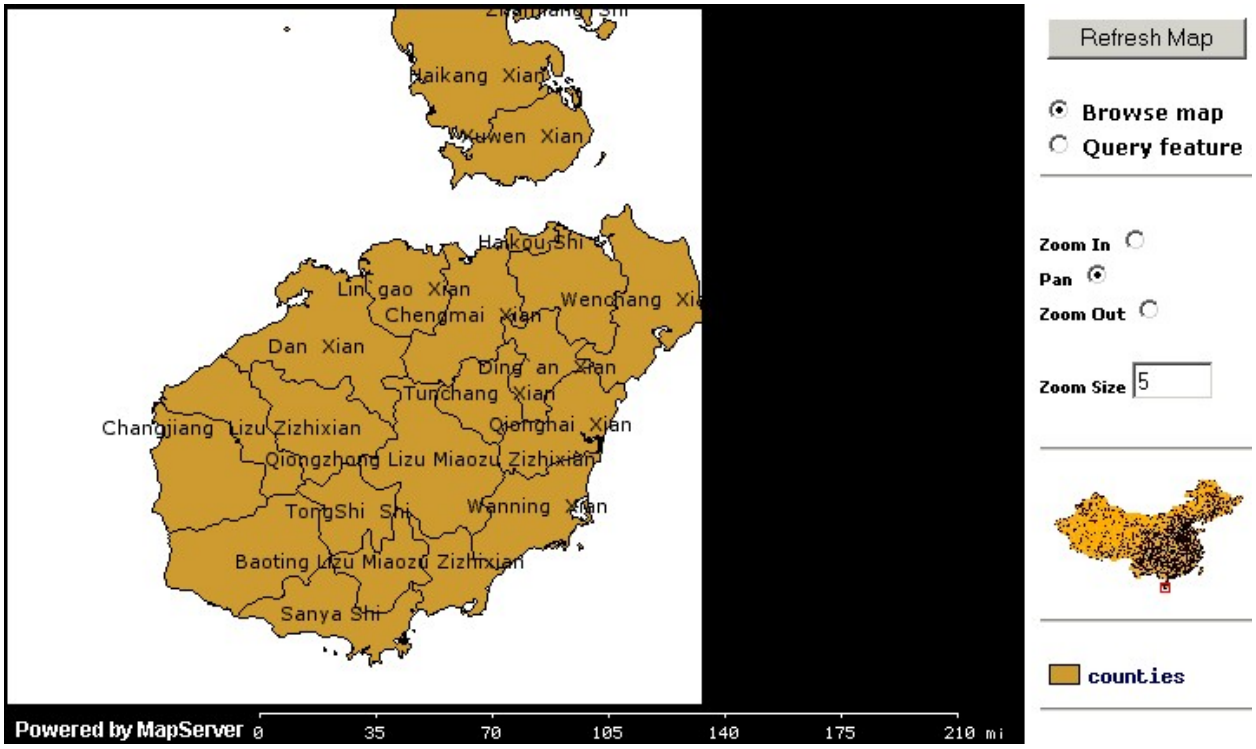

Figure 28: Webmap of Hainan

Note the small guide map of China on the lower right. A red bounding box indicates the extent of the featured map. This interface allows for zooming in or out, and querying features to see the attribute data contained in the underlying GIS shapefiles. If for example, we selected "Query Feature" from the top right, then used the mouse to click on the county shown at the bottom of the Hainan map, "Sanya Shi," we would see the results shown in Figure 29.

Layer: China Counties 1990

GUOBIAO CODE 1991	CHINESE NAME	PINYIN NAME	ADMIN UNIT	CAPITAL CHINESE NAME	CAPITAL PINYIN NAME	CAPITAL ADMIN UNIT
460201	三亚市	Sanya Shi	Urban Districts	三亚市	Sanya Shi	City

Map of Selected Feature

Figure 29: Results of "Query Feature" on webmaps

The method of producing embedded calls to PHP and CGI are shown in Figures 30 and 31.

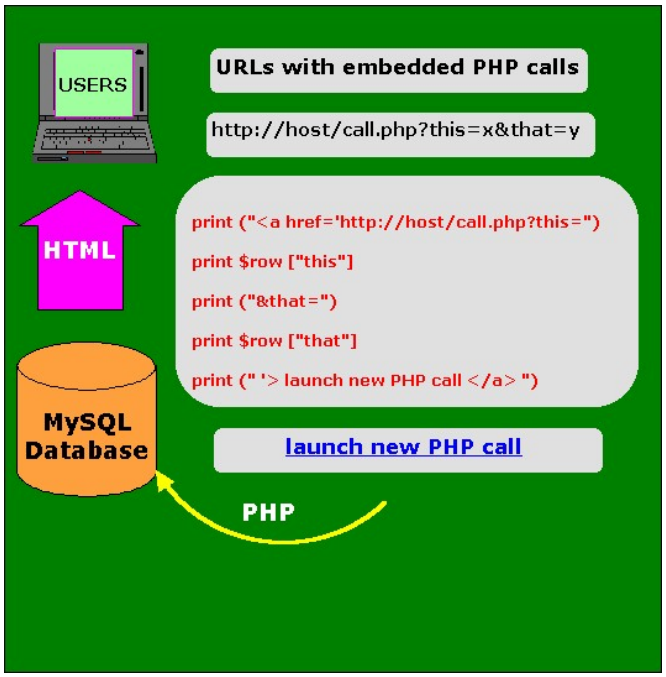

Figure 30: Embedded PHP calls

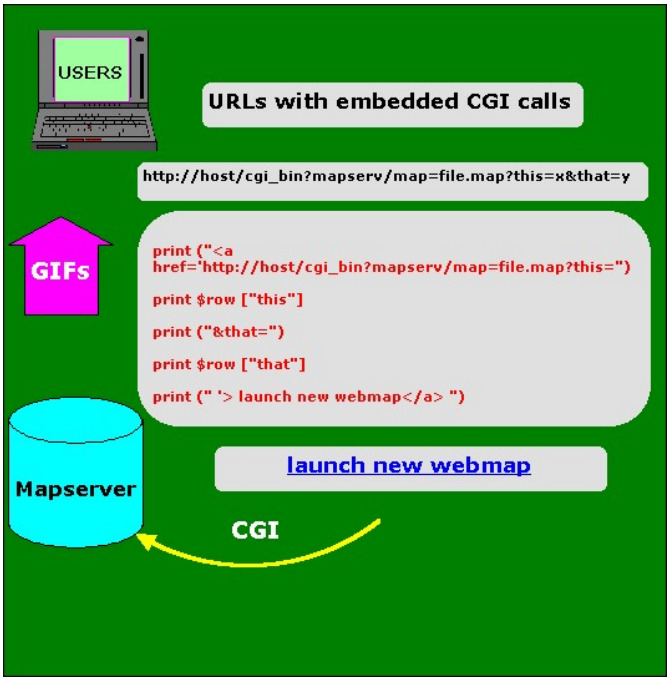

Figure 30: Embedded CGI calls to Mapserver

Mapserver is quite versatile and easily customizable using an HTML template for output. This has the advantage, not only of designing the look of the output, but also allowing us to call the correct Character Set Encoding. Similarly, the output of the feature queries are sent to the user as HTML, allowing us to choose the character set (as in Figure 29, showing BIG5 output). The screen shown in Figure 28 is the default template, but we can also redesign the output to suit our needs. For the Qing 1820 placename search, I have actually embedded two separate calls to Mapserver and output them to a table along with the attributes found in the MySQL database. For one call the location of the point for "Sanya" is shown in the context of the complete 1820 Qing Dynasty province map, as shown on the right in Figure 32.

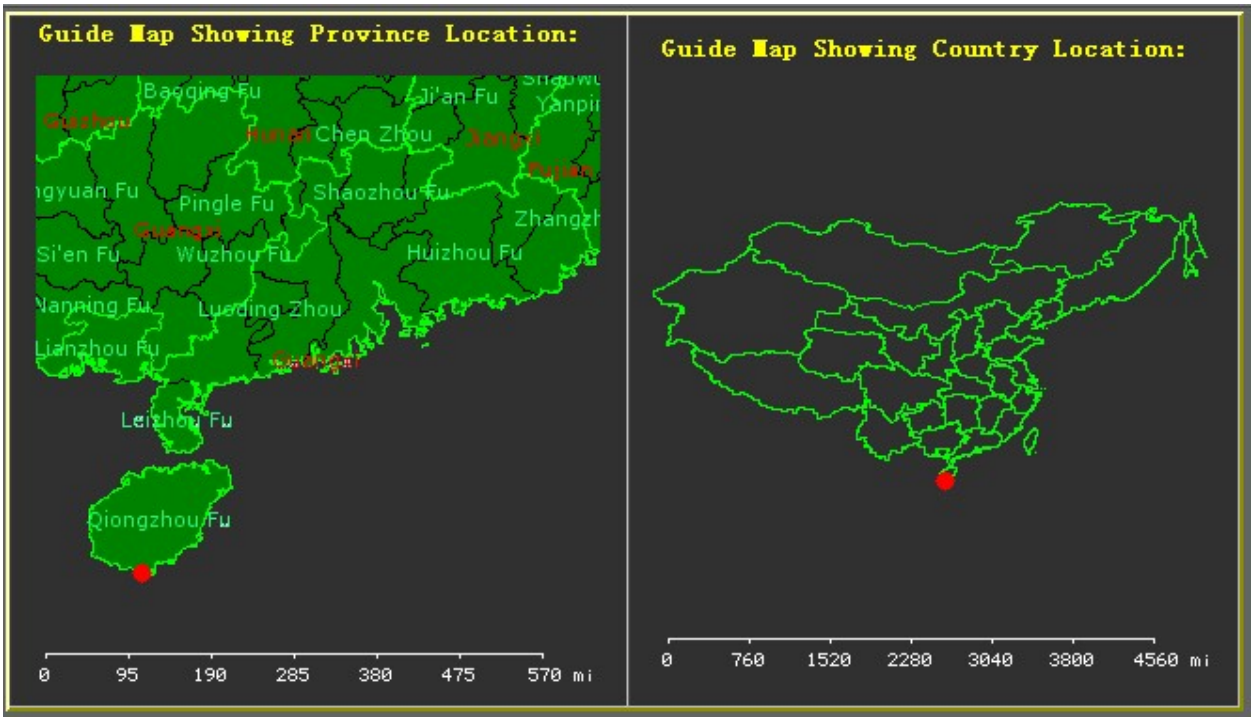

Figure 32. CHGIS Qing Dynasty, 1820, placenames webmap

For the second call, the mapserver zooms in to the geographic extent of the Province containing Sanya, which during the year 1820 was Guangdong, as shown on the left of Figure 32.

In order to create the two separate views, I first generated a maximum extent bounding box for each Province polygon in the 1820 GIS coverage. The bounding box coordinates were then entered into four separate fields in a related province extent table, as in max_west, max_south, max_north, max_east. This table is then joined to the table containing all the individual records in the database on the fly for each PHP query. When querying the MySQL database, whenever there is a hit, the the latitude - longitude coordinates for the found record are used to draw the point on the national level map (right side), then the bounding box values from the JOINED province extent table are used to ZOOM IN to the extent of the correct province (left side). Finally the point is drawn a second time on the province level map.

The mapserver is configured with three map layers. The top layer is the points layer, which is blank by default, but allows the drawing of the points on the fly, based on the

latitute – longitude coordinates sent as variables in the CGI call (shown in Figure 31). The middle layer contains the boundaries of all Qing 1820 Provinces. The bottom layer contains the Qing 1820 Fu boundaries. There is a MAXIMUM visible scale limitation on the Qing 1820 Fu boundaries layer, so that when the national map on the right is drawn, the Fu layer is not seen, but for any of the Province level ZOOMs on the left, the Fu layer is visible and is autolabeled.

Since the demonstration dataset contains only data for the year, or temporal slice, of 1820, the search engine does not include a temporal search. However, a working example of temporal searching for the web implementation has been completed. The temporal search is based on entering a single date as an input variable. This date is first checked against "Beginning Date" fields in the database to make sure that the input date is equal to and greater than the Beginning Date. Then the input date is also checked against the "Ending Dates" field to make sure that the input date is NOT greater than the ending date.

The SQL used is essetially as follows:

>>code sample start

```
if (empty($year) )
{$eyear = '25000';}
else {$eyear = $year;}

if (empty($year) )
{$byear = '-2500000000';}
else {$byear = $year;}

SELECT DISTINT FROM temporal_data

WHERE (end_date <= '$eyear' OR end_date = ' ' )

AND (begin_date >= '$byear' OR begin_date = ' ' )
```

>>code sample end

The first two clauses just arbitrarily set a beginning and end year for blank entries to 2.5 billion years BCE and 25,000 CE. This assumes that there will be no entries in the database earlier or later than those two years. These could easily be changed, of course. Note that there is actually only one year being input by the user, and the variable is renamed to be BOTH $eyear and $byear, then passed to the query.

The SELECT section just checks the input year against both beginning and ending years in the database. A working sample of this script is posted at:
http://www.dbr.nu/vortex/test/dig_gaz/dig_gaz3.html

Finally, the web implementation must deal with the issue of discovering parent and child units as necessitated by the part-whole relationship model. This requires a looped search, which we have provisionally solved using PHP. The idea is to define some units as parents in the administrative hierarchy, using the partOf model, as illustrated in Figure 33.

KEY_ID	Name	Unit	PartOf
544	Nanjiang	Xian	502
502	Baozhong	Diqu	25
25	Sichuan	Sheng	

Figure 33: Simple Administrative Hierarchy defined by PartOf field

Here we see that Nanjiang is part of Baozhong, which is part of Sichuan. Sichuan has no entry in the PartOf field, which means it is at the top of the hierarchy. Therefore our search for the parents of Nanjiang must first discover the parent #502, then search in the KEY_ID column to find 502 and search for its parent, and so on, until a blank value in the PartOf field is found, which stops the search loop.

A provisional solution is shown in the following sample code:[10]

>>code sample start

```
if ($row = mysql_fetch_array ($result) )
{
        do {

        $loop = $row["PartOf"];
        print $row["Name"];
        if($loop != "")

                {
                        $result = mysql_query ("
                                select *
                                from datafile
                                where Key_ID = '$loop'
                        ");

                $loop = $row["PartOf"];

                }

} while($row = mysql_fetch_array($result));
```

>>code sample end

Basically the idea is for the value in PartOf to be passed back and searched repeatedly in the key_id field by declaring it as a variable: $loop. As long as $loop doesn't return as null it keeps going.

The results can be formatted to look something like:

```
Nanjiang Xian
        Baozhong Diqu
                Sichuan Sheng
```

See also the working sample at: http://www.dbr.nu/vortex/partof/partOf_test3.php

Certainly the database model and web implementation for CHGIS described here has much room for improvement. Hopefully, the problems and solutions described here will be of some use to others working on similar projects, and can lead to further research on spatio-temporal querying, visualization, and information discovery and exchange among distributed databases. In particular, CHGIS would now like to focus on several areas:

To develop a means of comparing and retrieving data based on identified historical administrative divisions from distributed archives, such as the historical search engines of Academia Sinica.

To develop a means of visualizing spatial and temporal uncertainty, based on arc segments coded with uncertainty values or approximate date values.

To develop a means of incorporating specialized research by individuals (who make use of CHGIS as a basemap) into the datasets accessible to CHGIS users.

To develop a way to visually compare CHGIS layers with other datasets, perhaps with the help of TimeMap software.[11]

During the next two years, with continued support from Henry Luce Foundation, we expect that working models for all four of these research areas can be put into use, while we continue to push the CHGIS datasets backwards in time.

Notes:

1. Crissman, Lawrence. "Draft Database Design and Geocoding System." CHGIS, Dec 2000. http://www.people.fas.harvard.edu/~chgis/work/design/chinastdb_1210.doc

2. Guobiao Codes are the National Standard Codes for Administrative Divisions of China, also known by their publication name "GB 2260." For more info on Guobiao Codes see: Berman, M. "Guobiao Code Extensions," Feb 2000, http://dbr.nu/data/pubs/papers/lex_may00.pdf

3. Fudan University, Center for Historical Geography, CHGIS Staff, see: http://fas.harvard.edu/~chgis/work/docs/fudan_staff.jpg

4. Robert Hartwell's Historical GIS, see paper: Bol, Peter. "Overview of Work on an Historical GIS of China." Jan 2000. http://fas.harvard.edu/~chgis/data/pubs/Bol_Hartwell_GIS.doc
   See also Hartwell GIS Datasets: http://fas.harvard.edu/~chgis/data/hartwell/

5. See also Skinner's Qing Macroregional Analysis: http://qing.ucdavis.edu

6. ArcChina, the official basemap for the CHGIS project, is available from ESRI. See: http://gisstore.esri.com/acb/showdetl.cfm?&DID=6&Product_ID=309&CATID=15

7. A very thorough model for spatio-temporal entities is found in Claramunt, Christophe and Theriault, Marius. *Toward Semantics for Modelling Spatio-Temporal Processes*

Within GIS, in "Advances in GIS Research II," Edited by M.J. Kraak and M. Molenaar. London: Taylor & Francis, 1997.

8. The classic treatment of space-time composite modeling and opimizing spatio-temporal databases is found in Langran, Gail. "Time in Geographic Information Systems." London: Taylor & Francis, 1992. (See especially Chapters 3 and 4, pp 27-67.)

9. For an overview of temporal modeling see: Johnson, Ian. "Mapping the Fourth Dimension." Sep 1997. http://www.archaeology.usyd.edu.au/research/time_map/documentation/caa_1997/index.html The temporal boundary problem is discussed in detail in Langran, *ibid*, pp 27-44.

10. Special thanks to Miho Nakanishi for devising this code.

11. TimeMap Software, developed by Ian Johnson, Archeological Computing Laboratory, University of Sydney, is described at: http://www.archaeology.usyd.edu.au/research/time_map/