Implement an Intelligent ArcView User Interface Using SNePS

Jun Xu
Department of Geography, SUNY at Buffalo
junxu@acsu.buffalo.edu

Abstract An intelligent natural-language user interface between ArcView and SNePS (the semantic processing system) was implemented. The interface will accept the input command and query in natural language format, transfer it to Avenue command, and then apply it in ArcView. It will apply two kinds of functions. The first one is the ordinary functions in ArcView, such as add themes, query on themes, and identify features. The second one is spatial reasoning and fuzzy knowledge query. In the second case, we have to tell system the basic knowledge, which is also input as natural language.

Key Words: ArcView, SNePS, Natural language

1. Introduction

A main objective of GIS is to allow the user of the system to interact vicariously with actual or possible phenomena of the world (Mark, 1989). The user should be thinking about a problem domain, or a phenomena or a set of phenomena, and not about a computer or program or application (Mark, 1991). A good human-computer interface will help people focus on the work they want to do.

Traditional GIS software, such as ARC/INFO uses command line interpreters. It is difficult for a user to remember so many commands. ArcView and ArcMap use a windows interface, so people can operate GIS by direct manipulations. Direct manipulation is easier than a command line interpreter, but it is still difficult for a user who is not familiar with GIS. He/She will have to find the tools or buttons that can implement the functions. A natural language interface is a best way for naïve users to deal with computers easily. With the advent of real time speech-understanding system, spoken natural language is likely to become very important form of system interaction for GIS (Mark, 1991). It will be useful for both naïve users and GIS experts.

SNePS, the semantic network processing system, is a knowledge representation system which can process natural language. In this paper, an intelligent natural language interface

between ArcView, a geographic information management system, and SNeBR, the SNePS Belief Revision system was implemented. The interface will accept input commands and queries in natural language format, transfer it to Avenue commands, and then apply it in ArcView. The interface can also do the work of spatial reasoning.

2. Related work

Presenting knowledge in natural language is a very important issue in building a natural language interface. Multiple representations of objects are needed in geographic information system. Haller and Mark (1990) addressed the problem of generating natural language to express spatial relations between or among geographic entities. They used SNePS in multiple representations of geographic locatives. Zhan and Mark (1992) described an object-oriented spatial knowledge representation schema in CLIPS/COOL environment.

An intelligent natural language user interface has been implemented by connecting ARC/INFO with SNACTor, the SNePS acting component (Shapiro, et al. 1992). The communication medium between ARC/INFO and SNePS is the file system. Commands from SNACTor to ARC/INFO are issued by generating little command files; result from command executions in ARC/INFO are brought back into SNACTor via watch files. The SNACTor side of the communication is defined in a Common Lisp file. Its function is to translate the request into a set of AML commands, and write them to a command file. Then SNACTor will wait for ARC/INFO to implement the commands. The ARC/INFO side of the communication makes use of AML. ARC/INFO look into the interface directory, if the command file exists, it executes it, then delete the command file and write the result to the watch file. Once the command file is deleted, SNACTor adds the result of the command to its knowledge base, and then is ready to accept the next natural language requirement.

Another intelligent GIS interface was designed by Emergency management team in PSU (Rauschert, et al. 2002). They developed a Dialogue-Assisted Visual Environment for GeoInformation (DAVE_G), which uses a multimodal, multi-user GIS interface that can access GIS data through voice and gesture. It is based on a multimodal interface framework and applies ArcObjects for providing necessary GIS functionalities. The hand trajectory captured by a camera is used to recognize gestures and support the interpretation of a user's spoken command. The recorded human voice is processed by standard speech recognition software. The speech and hand gesture are integrated to generate a command.

3. Knowledge representation and reasoning with semantic network

3.1 Knowledge representation in SNePS

SNePS is an intensional, propositional, semantic-network knowledge representation system that is used for research in artificial intelligence and in cognitive science (Shapiro and Rapaport, 1995). SNePS is an intensional knowledge representation system. It supports multiple representation of what could be one physical object (Shapiro and Rapaport, 1987). So it should be able to represent anything and everything expressible in natural-language (Shapiro and Rapaport, 1992).

The primary data structure of SNePS is a semantic network. A Semantic network is a labeled, directed graph whose nodes represent entities and whose arcs represent binary relations between entities (Shapiro and Rapaport, 1995). It is a propositional semantic network, which means that all information, including propositions, "facts", etc, is represented by nodes. There are different kinds of nodes. Base nodes are distinguished by having no arcs coming from them. A base node is assumed to represent some entity—individual, object, class, property, etc. No two nodes represent the same, identical entity. Molecular nodes may represent propositions. Arcs represent the relations between nodes. The network in figure 1 represents that Buffalo is west to Rochester and Rochester is west to Syracuse.

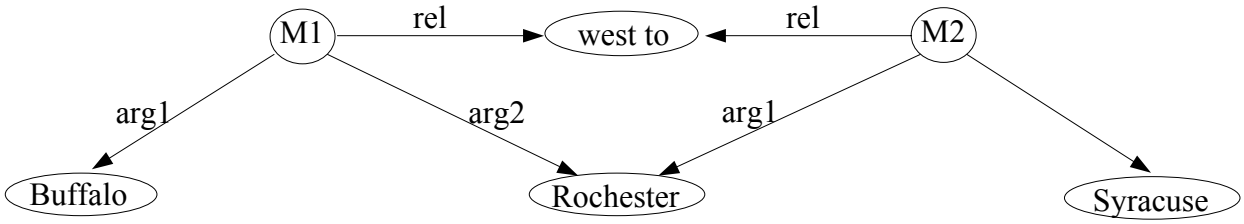


Figure 1. A semantic network representation of spatial relations

3.2 Knowledge inference

SNePS can not only represent information, but also infer information from what it has been told. With path-based inference, SNePS can infer an arc between two nodes from existence of a path of arcs between the same two nodes (Shapiro, 1991). For example, in the semantic network in figure 1, we can define a path between Buffalo and Syracuse, then SNePS can infer that Buffalo is west to Syracuse.

To do inference, a knowledge base has to be built. A SNePS-based knowledge base could be built by a human informing it using natural language. The following is the annotated example run in SNePS. The lines begin with ":" is what the user input to SNePS, the following line is the response of SNePS. At first, tell SNePS some basic information.

```
: Buffalo is a city.
I understand that Buffalo is a city.
: Rochester is a city.
I understand that Rochester is a city.
: Syracuse is a city.
I understand that Syracuse is a city.
: Cities are objects.
I understand that cities are objects.
: Buffalo is west to Rochester.
I understand that Buffalo is west to Rochester.
: Rochester is west to Syracuse.
I understand that Rochester is west to Syracuse.
```

The propositions that Buffalo, Rochester and Syracuse are cities are represented as M1, M2 and M3 respectively in Figure 2. The molecular node M4 represents the proposition that Buffalo is west to Rochester, and M5 represents that Rochester is west to Syracuse. If we ask whether Buffalo is west to Syracuse, SNePS will answer "yes" because we have already defined the path between arcs. M6 represents that Buffalo is west to Syracuse.

```
: Is Buffalo west to Syracuse?
Yes, Buffalo is west to Syracuse.
```

Now if we ask SNePS whether Rochester is east to Buffalo, SNePS does not know the answer because there is no such information in the network.

```
: Is Rochester east to Buffalo?
I really don't know if Rochester is east to Buffalo.
```

We have to tell SNePS that if a city is west to another city then the second city is east to the first city.

```
: If a city is west to an object then the object is east to the city.
I understand that if a city is west to a object then the object is east to the city.
```

We ask SNePS the same question again. SNePS will find that Buffalo is west to Rochester in the network. And from the rule we have just told it, it can refer that Rochester is east to Buffalo. The molecular node M7 in Figure 2 is built in the network automatically. This time SNePS gives the positive answer.

```
: Is Syracuse east to Buffalo?
```

```
Yes, Syracuse is east to Buffalo.
```

Tell SNePS more information. The proposition that I-90 is a highway is represented by node M8, and the proposition that Buffalo is on I-90 is represented by node M9 in Figure 2.

```
: I-90 is a highway.
I understand that I-90 is a highway.
: Buffalo is on I-90.
I understand that Buffalo is on I-90.
```

If we ask whether Buffalo is on highway, SNePS will find from the network that Buffalo is on I-90, and I-90 is a highway, so it will give the positive answer.

```
: Is Buffalo on I-90?
Yes, Buffalo is on I-90.
```

Figure 2 shows the whole semantic network that has been built.

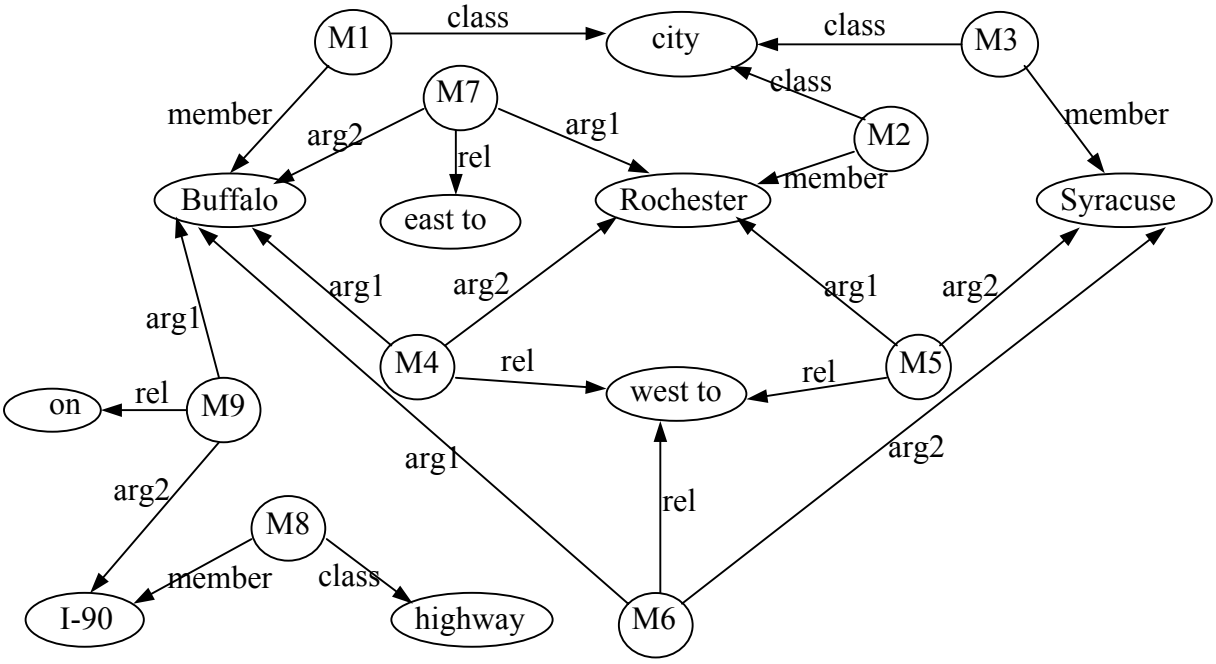


Figure 2. An example of semantic network

4. Interface between ArcView and SNePS

Unlike ARC/INFO system, ArcView uses windows interface. Many functions are implemented with menus, buttons, or tools. All these menus, buttons and tools are associated with some embedded scripts written in Avenue. Avenue is the customization and development environment for ArcView. It is written in ArcView script file. Users can write Avenue codes in

script files to implement the composite functions of those menus, buttons and tools in ArcView. With Avenue, users can also integrate ArcView with other applications. To operate an ArcView project in natural language, we need to implement an interface that integrates ArcView with SNePS.

In this paper, the same protocol used by Shapiro et al (1992) in the ARC/INFO SNACTor system is used, which uses the file system as the communication medium. SNePS issues a command, generates a command file, and waits for a result to be generated. ArcView reads the command file, executes the command, generates a result, writes the result to a result file if needed, and waits for the next command. There are two sides of the project, ArcView side and SNePS side.

4.1 SNePS side

In SNePS side, it gets input information, parses it, and performs the corresponding action function. These primitive actions tell SNePS to do some actions. In this project, primitive actions transfer the actions need to be performed to the form of Avenue commands. For example, to turn on a theme, the action "turnon" is defined:

```
(define-primaction turnon (object1)
  (setf thisobject (coerce-node (lex-node object1) :string))
  (setf cmd (format nil "av.run(\"TurnonTheme\", \"~a\")" thisobject))
  (just-do-it (format nil "Turn on theme ~a" thisobject) cmd))
```

The action "turnon" formats a Avenue command "*av.run("TurnonTheme", "aTheme")*", and write it to the command file. This command will run an embedded ArcView script *TurnonThemes*.

When the user wants to stop Arcview, a stop action is performed. Stop action is defined as following:

```
(define-primaction stop ()
  (just-do-it "quit" "_continue = false"))
```

This will write to *next.cmd* an Avenue sentence "_continue = false", which set the Avenue global variable "_continue" to false.

4.2 ArcView side

In ArcView side, it receives Avenue command from SNePS, and runs it. All the Avenue files are restored as scripts in ArcView projects. The main script is *MyStartupScript* which is written in Avenue:

```
MsgBox.Info("Please input command in Sneps.", "To start...")
f = "$Home/arcview/interface/next.cmd".asFileName
```

```
File.Delete(f)
_continue = true

while (_continue)
  if (File.Exists(f)) then
    cmdFile = TextFile.Make(F, #FILE_PERM_READ)
    cmd = cmdFile.Read(cmdFile.GetSize)
    curScript = Script.Make(cmd)
    If (curScript.HasError.Not) then
      curScript.DoIt("")
    end
    cmdFile.Close
    File.Delete(f)
  end
end

av.GetProject.close
av.Quit
```

Set *MyStartupScript* as default start up script of the ArcView project. Whenever ArcView is started, *MyStartUpScript* will be running, and performing a command loop as long as the global variable *_continue* is TRUE. When it detects the existence of file *next.cmd*, it will read it, and execute the Avenue command in the file. The sentence *"curScript.DoIt("")"* in *MyStartupScript* will execute the commands in the command file. Then command file is deleted. The script will go on doing this loop till *_continue* is set to FALSE. It is done by stop action in SNePS.

4.3 Connection between SNePS and ArcView

The connection of SNePS and ArcView happens in the *interface* direction. Every time SNePS gets a command, it transfers it into Avenue command, and saves it in a command file *next.cmd* in *interface* direction. Usually the commands in next.cmd are not Avenue commands operate on ArcView directly. They are commands call embedded scripts. After *MyStartupScript* in ArcView detects the existence of *next.cmd*, it will execute this command, and run the corresponding script. For example, if the command in *next.cmd* is "*ac.run("TurnonTheme", "aTheme")*", the script TurnonTheme is called by the sentence *"curScript.DoIt("")"* in *MyStartupScript*. The variable *"aTheme"* in the command is the name of the theme the user wants to turn on. The variable *"self"* in *TurnonTheme* will get the value of *"aTheme"*. Script TurnonTheme sets the theme to be visible if it is not:

```
theview = av.GetActiveDoc
theTheme = theView.FindTheme(self)  'self get the value of aTheme
if (theTheme.IsVisible.not) then
  theTheme.SetVisible(TRUE)
end
```

Because the action "turn on a theme" only needs to display the result in the window, so we don't need a result file from ArcView. For some actions, such as "find" need to query on the map

and get result from the map, the result will be written to a result file named *comout*. SNePS reads the result from the result file and prints it out.

5. Compared with previous work

This project uses the same structure and protocol as the previous ARC/INFO SNACTor project. But the method to implement the protocol is a little different. Since ARC/INFO uses command line, and its Macro language AML allows the user to write program containing arbitrary ARC/INFO commands, SNePS only need to translate input language to ARC/INFO commands, and write them to command file. But there is no language can operate on ArcView directly. The ArcView Avenue language must be written in a script file in an ArcView project, and usually a set of commands is needed to apply one function. So those sets of Avenue commands which apply one ArcView function have to be written in a script file in ArcView. The sentences in the command file are only commands to run those script files.

The previous ARC/INFO SNACTor project only implements a few ARC/INFO operations, and can only get simple query result from ARC/INFO. This is because of the limit of ARC/INFO itself. ARC/INFO can only display one map layer in one frame, and can only query on this displayed layer, whereas ArcView can display multi-layer of maps in one frame, and query with more than one map. Although the demo shown in this paper has not applied this complicate query in natural language at present, it is possible.

The different of this project with DAVE_G is that this project use SNePS to represent the knowledge, so it can not only operate on the GIS system, but also remember the represented knowledge, and infer the knowledge that has not been told. Whereas DAVE_G is only a interact environment that the user can operate on GIS system, although natural language is used in this interface, it is translated into GIS command by speech recognition software. The result of implement the command is displayed, and there is not feedback information from the GIS system. And there is no representation and reasoning about the knowledge. But DAVE_G use spoken language and gesture. It is more human friendly.

6. Sample run

This is an annotated sample run of this project. This demo is run on a SUN workstation in Geographic Information and Analysis Laboratory (GIAL) at Department of Geography. The output result has been edited for saving space. User input in SNePS package is shown at "*" prompts, user input in SNePSUL package is shown at "-->" prompts, and user input in natural

language is shown at ":" prompts. The system's response follows after the user input separated by a blank line.

At the beginning of the demo, reset the network, and define relations and path of the network.

```
* (demo "kb.sneps")

File /home/grad/geo/junxu/arcview/kb.sneps is now the source of input.

* (resetnet t)

Net reset

;;; Arc labels for predicates and relations
(= (define member class sub sup lex object property rel arg1 arg2
          object3) NEW-RELATIONS)

(MEMBER CLASS SUB SUP LEX OBJECT PROPERTY REL ARG1 ARG2 OBJECT3)

* (define-path class (compose class (kstar (compose member- class)) (kstar
(compose sub- sup))))
CLASS implied by the path (COMPOSE CLASS (KSTAR (COMPOSE MEMBER- CLASS))
                           (KSTAR (COMPOSE SUB- SUP)))
CLASS- implied by the path (COMPOSE (KSTAR (COMPOSE SUP- SUB))
                            (KSTAR (COMPOSE CLASS- MEMBER)) CLASS-)
```

Then go to SNePSUL package and read in the grammar and the lexicon files. The grammar file used here is the grammar file in SNeRE demo. The lexicon file defines the words and phases will be used in operating ArcView.

```
* ^^
--> (lexin "av-lex.dat")
undefined- (NIL NIL NIL NIL NIL NIL NIL)
("view" "theme" ..... "to" "of")
--> (atnin "av-grammar.dat")
Atnin read in states: (END GTHRESH ..... S NIL)
```

Now set some variables that control input reading and tracing, and attach the primitive functions with LISP functions.

```
--> (setq \*terminating-punctuation-flag\* '("." "!" "?")
         \*infertrace\* nil
      parser::\*debug\* nil)
NIL
--> (attach-primaction
   (build lex "snsequence") snsequence
    ......
   (build lex "stop") stop)
T
```

It is ready to parse the input natural language. Go to the natural language input and parse loop.

```
--> (parse -1)

 ATN parser initialization...
```

```
Input sentences in normal English orthographic convention.
Sentences may go beyond a line by having a space followed by a <CR>
To exit the parser, write ^end.
```

Now we can parse the input commands, and operate on ArcView. The next few sentences define classes view and document, their relations, and the rules on operating them in ArcView.

```
: Views are documents.

I understand that views are documents.

: After openning a document the document is opened.

I understand that after performing open on a document, the document
is opened.

: After openning a document the document is active.

I understand that after performing open on a document, the document
is active.

: After closing a document the document is not opened.

I understand that after performing close on a document, the document
is not opened.

: After closing a document the document is not active.

I understand that after performing close on a document, the document
is not active.

: After activating a document the document is active.

I understand that after performing activate on a document, the
document is active.

: After activating a document the document is opened.

I understand that after performing activate on a document, the
document is opened.

: After openning a document another documents are not active.

I understand that after performing open on a document, another
document is not active.

: After activating a document another documents are not active.

I understand that after performing activate on a document, another
document is not active.

: Before adding a theme to a view the view must be active.

I understand that before performing add on a theme and a view,
the view must be active.

: After adding a theme to a view the theme is in the view.

I understand that after performing add on a theme and a view,
the theme is in the view.
```

```
: A plan to achieve that a document is opened is to open the document.

I understand that a plan to achieve that a document is opened
is by performing open on the document.

: If a document is opened then a plan to achieve that the document is active
is to activate the document.

I understand that if a document is opened then a plan to achieve that
the document is active is by performing activate on the document.

: If a document is not opened then a plan to achieve that the document is
active is to open the document.

I understand that if a document is not opened then a plan to achieve that the
document is active is by performing open on the document.

: A plan to achieve that a theme is in a view is to add the theme to the view.

I understand that a plan to achieve that a theme is in a view
is by performing add on the theme and the view.

: Before turn off a theme in a view the view must be active.

I understand that before performing turn off on a theme and a
view, the view must be active.

: Before turn on a theme in a view the view must be active.

I understand that before performing turn on on a theme and a
view, the view must be active.
```

The next few sentences tell SNePS that view1 and view2 are members of view.

```
: View1 is a view.

I understand that view1 is a view.

: View2 is a view.

I understand that view2 is a view.
```

Ask SNePS if view1 is a document. Since we have already told it that view1 is a view, and views are documents, SNePS knows that view1 is a document.

```
: Is view1 a document?

Yes, view1 is a document.
```

Now tell SNePS some members of theme: *statetheme*, *citytheme*, *laketheme*, *rivertheme*, and *roadtheme*. They are the names of the themes going to be added in the ArcView.

```
: Statetheme is a theme.

I understand that statetheme is a theme.

: Citytheme is a theme.

I understand that citytheme is a theme.

: Laketheme is a theme.
```

```
I understand that laketheme is a theme.

: Rivertheme is a theme.

I understand that rivertheme is a theme.

: Roadtheme is a theme.

I understand that roadtheme is a theme.
```

Ask SNePS to perform something on ArcView. First, ask it to open a view. After opening a view, SNePS will believe that view1 is opened, and is active.

```
: Open view1.

I understand that you want me to perform open on view1.
Now  doing: open view1
```

Then add a theme to view1. Since perform "adding theme to a view" has a precondition that the view must be active, and it is satisfied, SNePS adds *statetheme* to view1. The states map of United States will be displayed in the view (figure 3a).

```
: Add statetheme to view1.

I understand that you want me to perform add on statetheme and view1.
Now  doing: Add theme statetheme to view1.
```

Add a theme to an unopened view view2. Adding theme to a view has a precondition, that the view must be active. Since the precondition is not satisfied, SNePS will achieve the goal that view2 is active. View2 is not open, to achieve that view2 is open is to open it, so SNePS will open view2 at first, and then add *laketheme* in view2. We can see view2 is opened in ArcView, and the map of lake is displayed in view2. SNePS believes that view2 is open and active, and *laketheme* is in view2. Then add *rivertheme* to view2 (figure 3b).

```
: Add laketheme to view2.

I understand that you want me to perform add on laketheme and view2.
Now  doing: open view2
Now  doing: Add theme laketheme to view2.

: Add rivertheme to view2.

I understand that you want me to perform add on rivertheme and view2.
Now  doing: Add theme rivertheme to view2.

: Is view2 active?

Yes, view2 is active.
```

Add a theme to an inactive view view1. Adding theme to a view has a precondition that the view must be active. Since the precondition is not satisfied, SNePS will achieve the goal that

view1 is active. View1 is already opened, to achieve that view1 is open is to active it, so SNePS will activate view1, and then add *roadtheme* in view1. View1 is activated, and the map of road is added to view1 (figure 3c). SNePS believes that view1 is active, and view2 is not active.

```
: Add roadtheme to view1.

I understand that you want me to perform add on roadtheme and view1.
Now  doing: activate view1
Now  doing: Add theme roadtheme to view1.

: Is view2 active?

No, view2 is not active.

: Is view1 active?

Yes, view1 is active.
```

We can query on ArcView using SNePS. For example, find New York State in *statetheme*. SNePS will get the respond from ArcView how many features are selected.

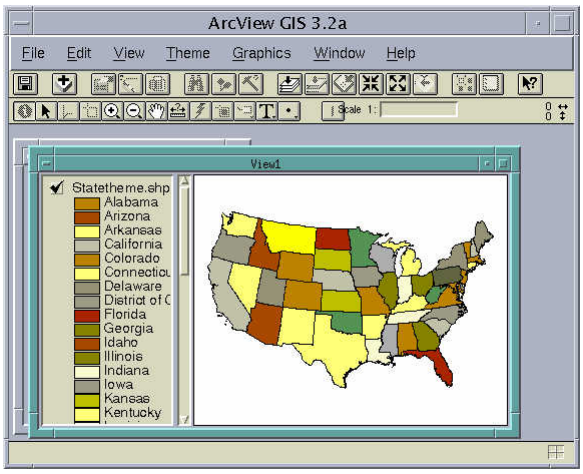
```
: Find New York in statetheme.

I understand that you want me to perform find on New York and statetheme.

Now  doing: query New York in statetheme.
There are 1 features selected.
```

Zoom in a theme will zoom to the selected features in the theme. If we ask SNePS to perform zooming in *statetheme*, ArcView will zoom to New York State in view1 (figure 3d). Zoom out will zoom to the full extent in the view.

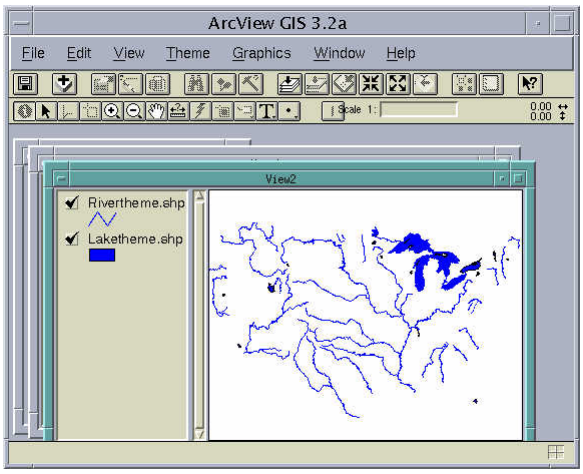
```
: Zoom in statetheme.

I understand that you want me to perform zoom in on statetheme.
Now  doing: Zoom to selected in statetheme.

: Zoom out statetheme.

I understand that you want me to perform zoom out on statetheme.
Now  doing: Zoom to full extent.
```

Add *citytheme* to view1 and query on *citytheme*. The selected city is highlighted (figure 3e).

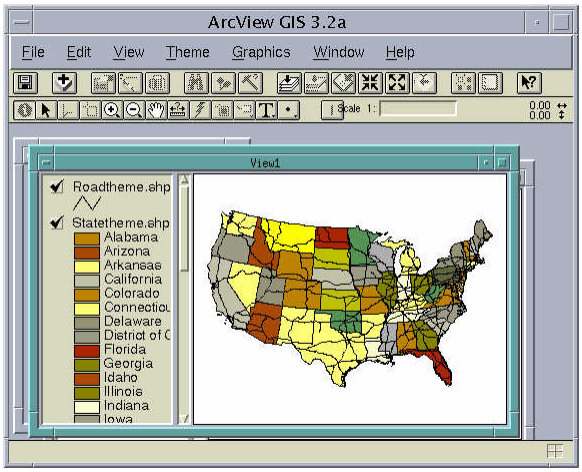
```
: Add roadtheme to view1.

I understand that you want me to perform add on roadtheme and view1.
Now  doing: activate view1
Now  doing: Add theme roadtheme to view1.

: Find San Diego in citytheme.

I understand that you want me to perform find on San Diego and citytheme.
Now  doing: query San Diego in citytheme.
There are 1 features selected.

: Zoom in citytheme.
```



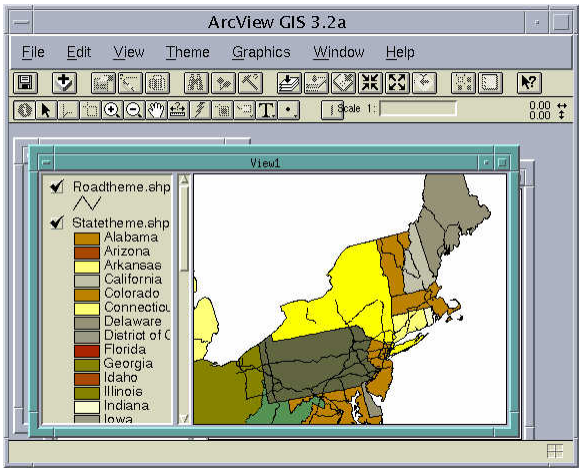
a



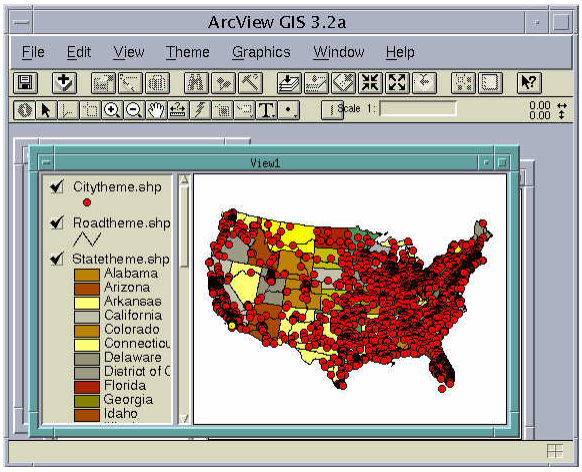
b



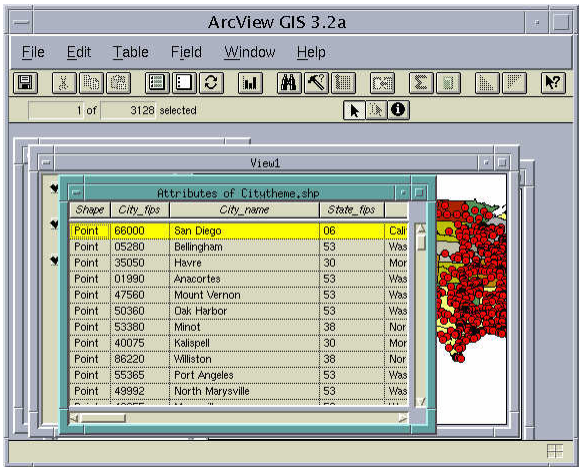
c



d



e



f

Figure 3. Illustration of sample run in ArcView

```
Now  doing: Zoom to selected in citytheme.
```

Ask SNePS to show the table of *citytheme*. The attribute table of *citytheme* will be displayed in ArcView, and the record of the selected city is promoted to the top of the table (figure 3f).

```
: Show table of citytheme.

I understand that you want me to perform show on table and citytheme.
Now  doing: Show the table of citytheme.
```

Perform stop will quit ArcView.

```
: stop arcview.

I understand that you want me to perform stop on arcview.
Now  doing: quit

: End of /home/grad/geo/junxu/arcview/kb.sneps demonstration.
```

7. Future work

Currently the interface can only perform the basic query on one single map and one single field in the attribute table. It cannot do complex query such as “Find all the cities have more than 1,000,000 populations”. It cannot query from two maps either, such as “Find all the cities in the distance of 5 miles to Intersate 80”. The present grammar file is not enough to parse such complicate sentences. The future work to improve the query function needs to verify the grammar file.

The multiple representations of the equivalent objects is another work to be done in the future. For example, if we can represent the proposition such as “my favorite city” and “near”, “far”, we can query questions such as “find my favorite city in city map”, “find cities near Buffalo”.

In this project, only the input knowledge is represented in the network, the knowledge required from the query is not represented. For example, we get “New York City” is one of the cities having more than 1,000,000 populations. If we can build this knowledge into the network, we can use it later when someone asks “Does New York City has more than 1,000,000 populations?” So that it becomes a real intelligent interface.

Reference

Haller, S. M., and Mark, D., 1990. Knowledge representation for understanding geographic locatives. Proceedings, 4th International Symposium on Spatial Data handling, July, 1990, v.1, 465-477.

Mark, D., 1989. Cognitive Image-Schemata and Geographic Information: Relation to User View and GIS Interface. Proceedings, GIS/LIS'89, Orlando, 551-560.

Mark, D., 1991. User interfaces for geographic information systems: Toward a research agenda. Proceedings of the Eleventh Annual ESRI User Conference, v. 2, pp.525-530.

Rauschert, I., Agrawal, P., Fuhrmann, S., Brewer, I., Wang, H., Sharma, R., Cai, G., & MacEachren, A, 2002. Designing a Human-Centered, Multimodal GIS Interface to Support Emergency Management. ACM GIS'02, 10th ACM Symposium on Advances in Geographic Information Systems, Washington, DC, USA, November, 2002.

Shapiro, S.C., 1991. Cabels, paths, and "subconscious" reasoning in propositional semantic networks. In Principles of semantic networks: explorations in the representation of knowledge, Edited by John F. Sowa, Morgan Kaufmann publishers, inc.

Shapiro, S.C., Chalupsky, H., Chou H., and Mark, D., 1992. Intelligent user interfaces: Connecting ARC/INFO and SNACTor, a semantic network based system for planning actions. In *Proceedings of the Twelfth Annual ESRI User Conference, V. 3*, pages 151-165. Environmental Systems Research Institute, Redlands, California, 1992.

Shapiro, S.C., and Rapaport, W.J., 1987. SNePS considered as a fully intensional propositional semantic network. In N. Cercone and G. McCalla (eds) The Knowledge Frontier, Essays in the Representation of Knowledge, Springer-Verlag, New York, 363-315.

Shapiro, S.C., and Rapaport, W.J., 1992. The SNePS family. Computer for Mathematic application, 23(2-5): 243-275.

Shapiro, Stuart C. and Rapaport, William J. (1995), "An Introduction to a Computational Reader of Narrative", in Judith Felson Duchan, Gail A. Bruder, & Lynne E. Hewitt (eds.), Deixis in Narrative: A Cognitive Science Perspective (Hillsdale, NJ: Lawrence Erlbaum Associates): 79-105.

Zhan, F., Mark, D., 1992. Object-oriented spatial knowledge representation and processing: formalization of core classes and their relationships. Proceedings, Fifth International Symposium on Spatial Data Handling, Charleston, South Carolina, v.2, 662-671.